Reconstructive Geometry

TU Graz

Institut für ComputerGraphik
und WissensVisualisierung,
Technische Universität Graz

Dissertation

Reconstructive Geometry

submitted in partial satisfaction of the requirements

for the degree of Doctor of Philosophy (PhD)

in Information and Computer Science

by Dipl.-Math. Torsten Ullrich,

Graz University of Technology, Austria

Institute of Computer Graphics
and Knowledge Visualization

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

(date)	(signature)

Preface

A this point, I would like to use the opportunity to thank my supervisor Prof. Dr. Dieter W. Fellner and my colleagues of the Institute of Computer Graphics and Knowledge Visualization at Graz University of Technology. Highly committed colleagues and an inspiring working environment had a significant influence on the fruitfulness and the numerous achievements. In this context, I would like to thank all the coauthors of the scientific articles published within their and my work.

Abstract

The thesis “Reconstructive Geometry” by Torsten Ullrich presents a new collision detection algorithm, a novel approach to generative modeling, and an innovative shape recognition technique. All these contributions are centered around the questions “how to combine acquisition data with generative model descriptions” and “how to perform this combination efficiently”. Acquisition data – such as point clouds and triangle meshes – are created e.g. by a 3D scanner or a photogrammetric process. They can describe a shape’s geometry very well, but do not contain any semantic information. With generative descriptions it’s the other way round: a procedure describes a rather ideal object and its construction process. This thesis builds a bridge between both types of geometry descriptions and combines them to a semantic unit. An innovative shape recognition technique, presented in this thesis, determines whether a digitized real-world object might have been created by a given generative description, and if so, it identifies the high-level parameters that have been passed to the generative script. Such a generative script is a simple JavaScript function. Using the generative modeling compiler “Euclides” the function can be understood in a mathematical sense; i.e. it can be differentiated with respect to its input parameters, it can be embedded into an objective function, and it can be optimized using standard numerical analysis. This approach offers a wide range of applications for generative modeling techniques; parameters do not have to be set manually – they can be set automatically according to a reasonable objective function. In case of shape recognition, the objective function is distance-based and measures the similarity of two objects. The techniques that are used to efficiently perform this task (space partitioning, hierarchical structures, etc.) are the same in collision detection where the question, whether two objects have distance zero, is answered. To sum up, distance functions and distance calculations are a main part of this thesis along with their application in geometric object descriptions, semantic enrichment, numerical analysis and many more.

Publications

All main parts of this thesis have been – respectively will be – published in the following articles and conference contributions:

[HUF11] Sven Havemann, Torsten Ullrich, and Dieter W. Fellner. The Meaning of Shape and some Techniques to Extract It. *Multimedia Information Extraction*, XX:to appear, 2011.

[SUF11] Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. Simple and Efficient Normal Encoding with Error Bounds. *Poster Proceedings of Theory and Practice of Computer Graphics*, XX:to appear, 2011.

[SSUF11a] Thomas Schiffer, Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. Real-World Geometry and Generative Knowledge. *The European Research Consortium for Informatics and Mathematics (ERCIM) News*, XX:to appear, 2011.

[SSUF11b] Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner. Modeling Procedural Knowledge – a generative modeler for cultural heritage. *Selected Readings in Computer Graphics 2010*, XX:to appear, 2011.

[UF11a] Torsten Ullrich and Dieter W. Fellner. Generative Object Definition and Semantic Recognition. *Proccedings of the Eurographics Workshop on 3D Object Retrieval*, 4:1–8, 2011.

[UF11b] Torsten Ullrich and Dieter W. Fellner. Linear Algorithms in Sublinear Time – a tutorial on statistical estimation. *IEEE Computer Graphics and Applications*, 31:58–66, 2011.

[BBU+11] Frank Breuel, René Berndt, Torsten Ullrich, Eva Eggeling, and Dieter W. Fellner. Mate in 3D – Publishing Interactive Content in PDF3D. *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, 15:110–119, 2011.

[SUSF11] Christoph Schinko, Torsten Ullrich, Thomas Schiffer, and Dieter W. Fellner. Variance Analysis and Comparison in Computer-Aided Design. *Proceedings of the International Workshop on 3D Virtual Reconstruction and Visualization of Complex Architectures*, XXXVIII-5/W16:3B21–25, 2011.

[USB10] Torsten Ullrich, Volker Settgast, and René Berndt. Semantic Enrichment for 3D Documents: Techniques and Open Problems. *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, 14:374–384, 2010.

[USF10b] Torsten Ullrich, Christoph Schinko, and Dieter W. Fellner. Procedural Modeling in Theory and Practice. *Poster Proceedings of the 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision*, 18:5–8, 2010.

[USF10a] Torsten Ullrich, Andreas Schiefer, and Dieter W. Fellner. Modeling with Subdivision Surfaces. *Proceedings of the 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision*, 18:1–8, 2010.

[SSUF10b] Martin Strobl, Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. Euclides – A JavaScript to PostScript Translator. *Proccedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, 1:14–21, 2010.

[SSUF10a] Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner. Modeling Procedural Knowledge – a generative modeler for cultural heritage. *Proccedings of EUROMED 2010 - Lecture Notes on Computer Science*, 6436:153–165, 2010.

[SSB+10] Thomas Schiffer, Andreas Schiefer, René Berndt, Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Enlightened by the Web – A service-oriented architecture for real-time photorealistic rendering. *Kongress Multimediatechnik*, 5:41–48, 2010.

[SBU+10] Andreas Schiefer, René Berndt, Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Service-Oriented Scene Graph Manipulation. *Proceedings of the 15th International Conference on Web 3D Technology*, 15:55–62, 2010.

[USOF09] Torsten Ullrich, Volker Settgast, Christian Ofenböck, and Dieter W. Fellner. Short Paper: Desktop Integration in Graphics Environments. *Proccedings of the 2009 Joint Virtual Reality Conference of Eurographics Symposium on Virtual Environments (EGVE), International Conference on Artificial Reality and Telexistence (ICAT), and EuroVR (INTUITION) Conference*, 15:109–112, 2009.

[USF09] Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Semantic Fitting and Reconstuction. *Selected Readings in Computer Graphics 2008*, 19:69–84, 2009.

[FUFB09] Christoph Fünfzig, Torsten Ullrich, Dieter W. Fellner, and Edward N. Bachelder. Terrain and Model Queries Using Scalar Representation With Wavelet Compression. *IEEE Transactions on Instrumentation and Measurement*, 58:3079–3085, 2009.

[SRO+08] Markus Steiner, Philipp Reiter, Christian Ofenböck, Volker Settgast, Torsten Ullrich, Marcel Lancelle, and Dieter W. Fellner. Intuitive Navigation in Virtual Environments. *Proceedings of Eurographics Symposium on Virtual Environments*, 14:5–8, 2008.

[UTF08] Torsten Ullrich, Torsten Techmann, and Dieter W. Fellner. Web-based Algorithm Tutorials in Different Learning Scenarios. *World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-Media)*, 20:5467–5472, 2008.

[UKF08] Torsten Ullrich, Ulrich Krispel, and Dieter W. Fellner. Compilation of Procedural Models. *Proceeding of the 13th International Conference on 3D Web Technology*, 13:75–81, 2008.

[USF08b] Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Semantic Fitting and Reconstuction. *Journal on Computing and Cultural Heritage*, 1(2):1201–1220, 2008.

[USF08a] Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Abstand: Distance Visualization for Geometric Analysis. *Project Paper Proceedings of the Conference on Virtual Systems and MultiMedia Dedicated to Digital Heritage (VSMM)*, 14:334–340, 2008.

[FUFB07] Christoph Fünfzig, Torsten Ullrich, Dieter W. Fellner, and Edward N. Bachelder. Empirical Comparison of Data Strucutres for Line of Sight Computation. *Proceedings of IEEE International Symposium on Intelligent Signal Processing (WISP) 2007*, 1:291–296, 2007.

[SUF07] Volker Settgast, Torsten Ullrich, and Dieter W. Fellner. Information Technology for Cultural Heritage. *IEEE Potentials*, 26(4):38–43, 2007.

[USK+07] Torsten Ullrich, Volker Settgast, Ulrich Krispel, Christoph Fünfzig, and Dieter W. Fellner. Distance Calculation between a Point and a Subdivision Surface. *Proceedings of 2007 Vision, Modeling and Visualization (VMV)*, 1:161–169, 2007.

[UFF07] Torsten Ullrich, Christoph Fünfzig, and Dieter W. Fellner. Two Different Views On Collision Detection. *IEEE Potentials*, 26(1):26–30, 2007.

[UF07b] Torsten Ullrich and Dieter W. Fellner. Robust Shape Fitting and Semantic Enrichment. *Proceedings of the 2007 International Symposium of the International Committee for Architectural Photogrammetry (CIPA)*, 21:727–732, 2007.

[UF07a] Torsten Ullrich and Dieter W. Fellner. Client-Side Scripting in Blended Learning Environments. *The European Research Consortium for Informatics and Mathematics (ERCIM) News*, 71:43–44, 2007.

[FUF06] Christoph Fünfzig, Torsten Ullrich, and Dieter W. Fellner. Hierarchical Spherical Distance Fields for Collision Detection. *Computer Graphics and Applications*, 26(1):64–74, 2006.

[LOU+06] Marcel Lancelle, Lars Offen, Torsten Ullrich, Torsten Techmann, and Dieter W. Fellner. Minimally Invasive Projector Calibration for 3D Applications. *Proceedings of 3. Workshop Virtuelle und Erweiterte Realität der GI-Fachgruppe VR/AR*, 1(1):1–9, 2006.

[UF05] Torsten Ullrich and Dieter W. Fellner. Computer Graphics Courseware. *Proceedings of Eurographics 2005 Education*, 1:11–17, 2005.

[UF04b] Torsten Ullrich and Dieter W. Fellner. Modulare Inhaltserzeugung nach dem Baukastenprinzip. *DeLFI 2004: Die e-Learning Fachtagung der Gesellschaft für Informatik 2004*, 52:405–406, 2004.

[UF04a] Torsten Ullrich and Dieter W. Fellner. AlgoViz - a Computer Graphics Algorithm Visualization Toolkit. *World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-Media)*, 16:941–948, 2004.

Contents

Notation

The following notation is used throughout this book:

Scalars	$a,b,c,\ldots$
Vectors	$\mathbf{u},\mathbf{v},\mathbf{w},\ldots$
Matrices	$\mathbf{M},\mathbf{A},\mathbf{B},\ldots$
Vector spaces	$V,A,B,\ldots$
Points	$P,Q,R,\ldots$
Point sets	$\mathbf{P},\mathbf{Q},\mathbf{R},\ldots$
Affine spaces	$\mathbb{A},\mathbb{B},\mathbb{C},\ldots$
Algebras	$\mathscr{F},\mathscr{G},\ldots$
Sets	$X,Y,Z,\ldots$

1 Introduction

Information technology applications in the field of cultural heritage include various disciplines of computer science. The work flow from archaeological discovery to scientific preparation demands multidisciplinary cooperation and interaction at various levels.

This chapter describes the information technology pipeline from the computer science point of view. The description starts with the model acquisition. Computer vision algorithms are able to generate a raw 3D model using input data such as photos, scans, etc. In the next step computer graphics methods create an accurate, high-level model description.

Beside geometric information each model needs semantic metadata in order to perform digital library tasks (storage, markup, indexing, and retrieval). A structured repository of virtual artifacts completes the pipeline – at least from the computer science point of view.

The context of information technology in the field of cultural heritage is described in this chapter. In the whole thesis it will serve a field of applications for reconstruction techniques. The subsequent chapters explain reconstructive geometry in-depth including its mathematical basis and additional, illustrative applications in collision detection, line-of-sight calculation, projector calibration, and many more.

Contents

1.1 Cultural Heritage

> Une civilisation est un héritage de croyances, de coutumes et de connaissances, lentement acquises au cours des siècles, difficiles parfois á justifier par la logique, mais qui se justifient d'elles-mêmes, comme des chemins, s'ils conduisent quelque part, puisqu'elles, puisqu'elles ouvrent á l'homme son ètendue intérieure.
>
> A civilization is a heritage of beliefs, customs, and knowledge slowly accumulated in the course of centuries, elements difficult at times to justify by logic, but justifying themselves as paths when they lead somewhere, since they open up for man his inner distance.
>
> ANTOINE DE SAINT-EXUPÉRY[1]

Each civilization wants to preserve its paths back to the roots, as the legacy from the past is an irreplaceable source of inspiration and an indispensable element to understand history. The demand to preserve mankind's artifacts of history is a body of thought that obtains more and more acceptance in society. Using state-of-the-art information technology the limited resources needed to conserve cultural heritage can be turned to account efficiently. The technical process to acquire historical artifacts is a non-trivial task, even if the questions of preservation, restoration, and refurbishment shall remain unanswered within this text.

1.1.1 Acquisition Pipeline

The first step in the acquisition pipeline is to measure an object using 3D scanning technologies based on laser scanners, computer tomography, or photogrammetry. The measured data, usually represented as a point cloud or a heightfield, has to be re-engineered to receive a geometric description of the object. Although the geometric description has an added value, a further, expedient acquisition needs semantic information which classifies the object and provides a valuable context.

The acquisition pipeline starts with the physical measurement of an object. The possibilities to acquire a three-dimensional artifact varies from manual tape measurement and haptic sensor acquisition to laser range scanners and image-based photogrammetry.

All these methods have their advantages and disadvantages. From the usability point of view an image-based reconstruction is the easiest one – everyone can handle

[1] ANTOINE DE SAINT-EXUPÉRY (June 29, 1900 – July 31, 1944) Antoine de Saint-Exupéry belongs to the most popular representatives of modern, French literature. His book "Le Petit Prince" (english: The Little Prince) became world famous.

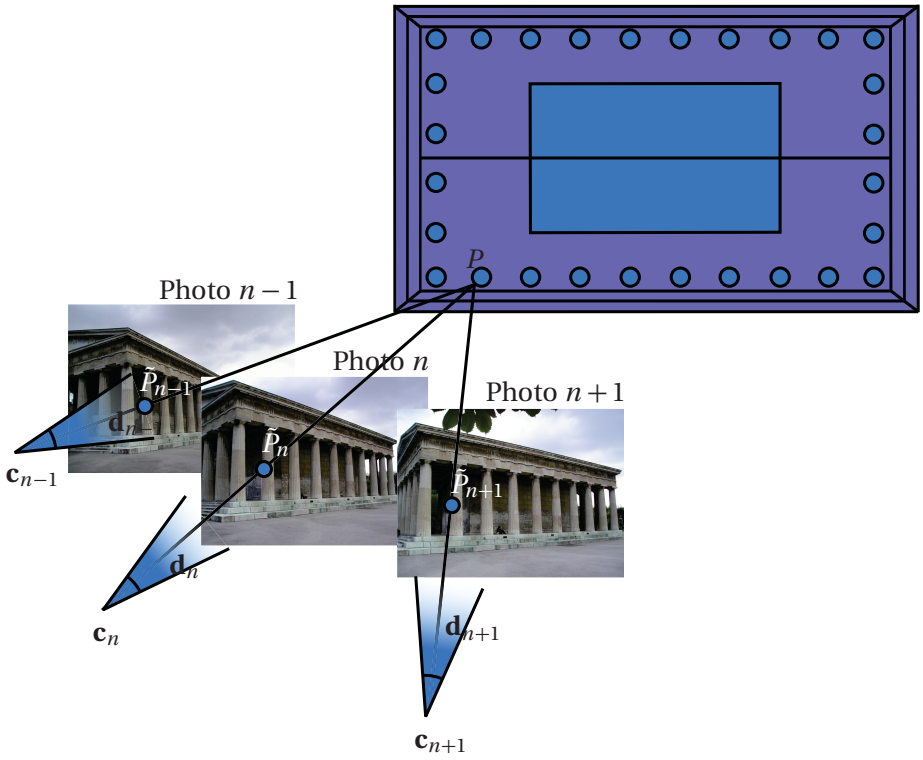


Figure 1.1: A photogrammetric reconstruction uses differences in a sequence of images to determine the 3D coordinates of an object. One of the main steps of an image-based reconstruction is to identify corresponding points $\tilde{P}_i$, which are images of the same 3D point P. Having identified a set of corresponding points in the sequence of images, the theory of epipolar geometry allows the determination of the cameras' parameters (their position $\mathbf{c}_n$ and orientation $\mathbf{d}_n$ vectors, focal length, etc.).

a camera. The complexity of this approach is hidden within the algorithms to generate a 3D model. To build a three dimensional model out of a sequence of images, three steps need to be done:

1. In the sequence of images, so-called corresponding points have to be identified. Two points in different images are corresponding, if they show the same 3D point that has been photographed. Such a constellation is illustrated in Figure 1.1.

2. Having identified a set of corresponding points in the sequence of images, the theory of epipolar geometry allows the determination of the extrinsic camera parameters (position and orientation) as well as the intrinsic camera parameters (principal point, focal length, and distortion).

At least seven pairs of corresponding points in two images are needed to determine the two cameras. Due to numerical stability aspects, any robust algorithm will use many more points to perform an error minimization.

3. The process by which a few points with already calculated coordinates in 3D are compared to a complete mapping of all pixels to the correct 3D coordinates is called dense matching. After this process, a depth value can be assigned to each pixel in a photograph transforming an image into a heightfield.

The corresponding point problem can be solved in many ways. Several common approaches can work iteratively to solve this chicken and egg problem. It is easy to identify corresponding points when the underlying 3D model is known. It also is possible to reconstruct the photographed point in 3D when its correspondences in all images are known.

Besides many techniques, a very general solution to this problem has been proposed by YOSHIHISA SHINAGAWA and TOSIYASU L. KUNII [SK98]. They use so-called multiresolutional critical-point filters to automatically match images. The main idea is to trace corresponding points recursively through a multiresolution hierarchy. To calculate the point correspondence of two points at level i the algorithm uses the already calculated correspondence at level $i-1$ and an error function. At level 0 the images have a size of 1×1 pixel and the point correspondence is trivial, as there is only one possibility. The fact that no prior knowledge about the objects is necessary, makes the algorithm attractive for a wide range of applications such as stereo photogrammetry, fully automatic morphing, object recognition, and volume segmentation.

To reduce the amount of potentially corresponding points, almost every algorithm solving the correspondence problem uses an upstream filter, e.g., a Laplace filter as shown in Figure 1.2. In many cases filters produce similar results for corresponding points in both images. For example, a point on an edge will most likely have a corresponding point that is also located on an edge. Therefore, an edge detection filter reduces the number of possible correspondences significantly.

Due to projective and epipolar geometry the images of corresponding points in a sequence of photographs are sufficient to determine the extrinsic and intrinsic camera parameters.

1.1.2 Assembling Object Parts

In practice it is not possible to acquire an object of interest with only one sequence of images or laser scanned range maps. In many cases, it is necessary to match multiple data sets into one coordinate system. This process is called registration. Depending on the complexity of the scanned object, hundreds of object parts have to be registered to create one consistent data set.

The registration is relatively easy, if the exact location and orientation of the scanner is known. This can be achieved by tracking the scanning device with appropriate hardware. As the tracking of the overall scanning process is laborious – especially for large objects like cathedrals, etc. – a software solution for registration is desired.

Figure 1.2: A Laplace filter eliminates same-color regions and highlights edges. Assuming that a point on an edge will have corresponding points that are also located on edges, a Laplace filter reduces the amount of input data to solve the corresponding point problem and to find a stable solution.

The software registration process is typically separated into a coarse and a fine registration step. During the coarse registration step all object parts are transformed into a less detailed representation. In the last years, many of these representations have been proposed. Their purpose is to reduce the quadratic complexity of pairwise data set alignment. Furthermore, the computation time is shortened by selecting only the most important features of the data set. The main task of automatic registration is to minimize the distance of overlapping parts. One of the algorithmic challenges of the coarse step is to figure out the correct arrangement of repeating object structures. This procedure takes a lot of time; in particular, if it needs user interaction.

The VISUAL COMPUTING LAB (VCL), in Pisa, which is supported by the Italian National Research Council's Institute of Information Science and Technologies, has developed an automatic registration technique. It considers the knowledge of the typically regular scanner pose pattern. Under the condition of a sufficient overlap (15-20%) and of regular scan stripes, it simplifies the sequential arrangement of range maps. To align one range map to another one, three pairs of common points are

needed. Normally the alignment problem is solved iteratively, until an error threshold condition is met.

The entire registration of this algorithm relies on a variance classification for each point of the range map regarding its neighborhood. This variance helps to cluster the range map into parts of similar curvature. As regions with high variance may be created due to scanning errors and regions of low variance do not contain enough characteristics, the algorithm does not take them into account. The iterative alignment procedure limits the choice of point pairs to points of similar variance. The fine registration of the range maps is typically done using the Iterative Closest Point (ICP) algorithm. Details of both algorithms are described in "Exploiting the scanning sequence for automatic registration of large sets of range maps" [PFC+05].

1.2 Geometric Reconstruction and Semantic Enrichment

The creation of consistent and accurate model descriptions is known as reverse engineering and comprehends fitting, approximation and numerical optimization techniques. If the underlying model description is able to describe every three dimensional object in a consistent and integrative way without the need of an additional superstructure, the reconstruction process can be called complete. Otherwise an object is described by several small parts whose orientation to each other is stored in a superstructure; e.g. a scene graph. This approach is a subpart reconstruction.

1.2.1 Fitting Techniques

In 1992 HUGUES HOPPE et al. presented "Surface reconstruction from unorganized points" [HDD+92] – an algorithm that fits a polyhedral surface to an unorganized cloud of points. The result is a polygonal mesh, which describes the complete object. Further development of polygonal reconstruction has lead to algorithms – amongst other the crust algorithm [ACDL00] – whose output is guaranteed to be topologically correct and convergent to the original surface as the sampling density increases.

As polyhedral surfaces are not the only way to describe a three dimensional object in a unified way, the class of complete reconstructions also comprises algorithms to fit radial basis functions, constructive solid geometry, or subdivision surfaces, to name a few.

The subpart fitting approaches can be divided into two categories depending on whether or not the input data has to be segmented and partitioned in advance. No preceding segmentation is needed among others by algorithms based on random sample consensus (RANSAC). The basic idea of RANSAC methods is to compute the parameters of a model from an adequate number of randomly selected samples. Then all samples vote on whether or not they agree with the proposed hypothesis. This process is repeated until a sufficiently broad consensus is achieved. Two major advantages of this approach are its ability to ignore outliers without explicit handling,

and be extended to extract multiple model instances in a data set. In this way, it is possible to determine, for example, the planes that describe the input data best. Although these algorithms can take advantage of feature extractions, they do not rely on them.

Feature extraction algorithms can determine feature lines such as crease loops and junctions, or border lines. These algorithms use principal component analysis or heuristics on local neighborhoods to classify points according to the likelihood that they belong to a feature line. These local feature vectors separate the input data into smaller regions, which can be approximated individually by a single surface as illustrated in Figure 1.3.

A sequence of tests splits a large point cloud into smaller and smaller subregions until no further subdivision is sensible. These model fragmentations are needed, for example, by subpart reconstructions based on nonuniform rational B-splines (a commonly used surface description in computer-aided design), developable surfaces, or least squares fitting techniques.

The aim of the remeshing and reconstruction process is to extract high-level geometry and to generate consistent and accurate model descriptions as shown in Figure 1.4.

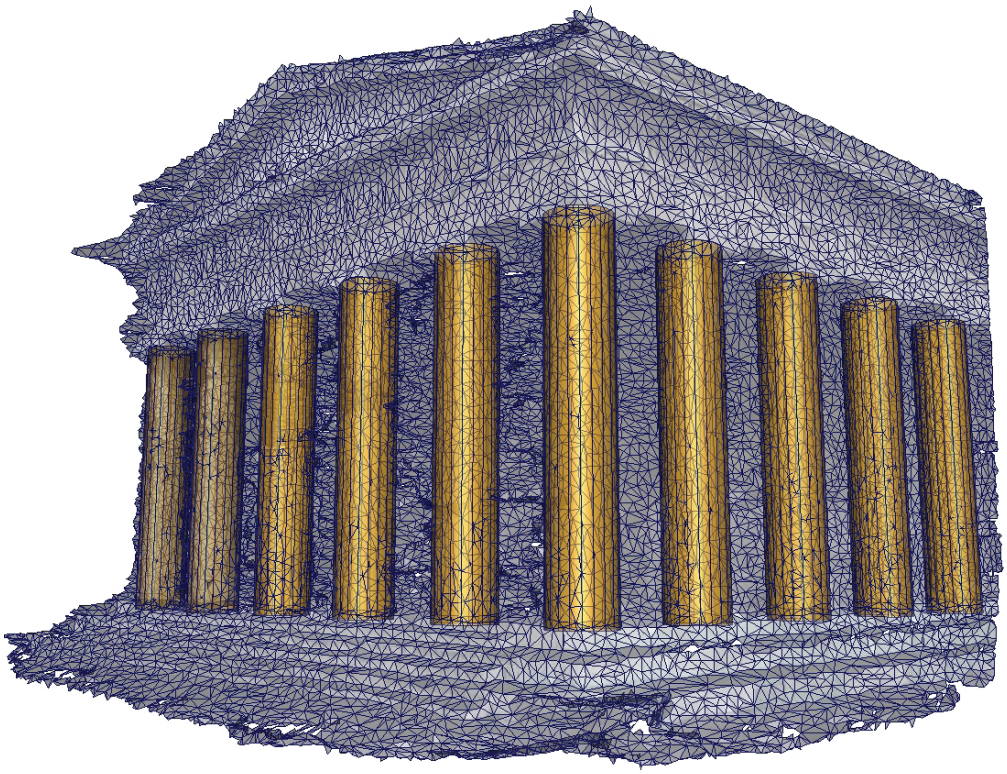


Figure 1.3: The aim of the remeshing and reconstruction process is to extract high-level geometry, such as the temple's columns, out of a low-level description.

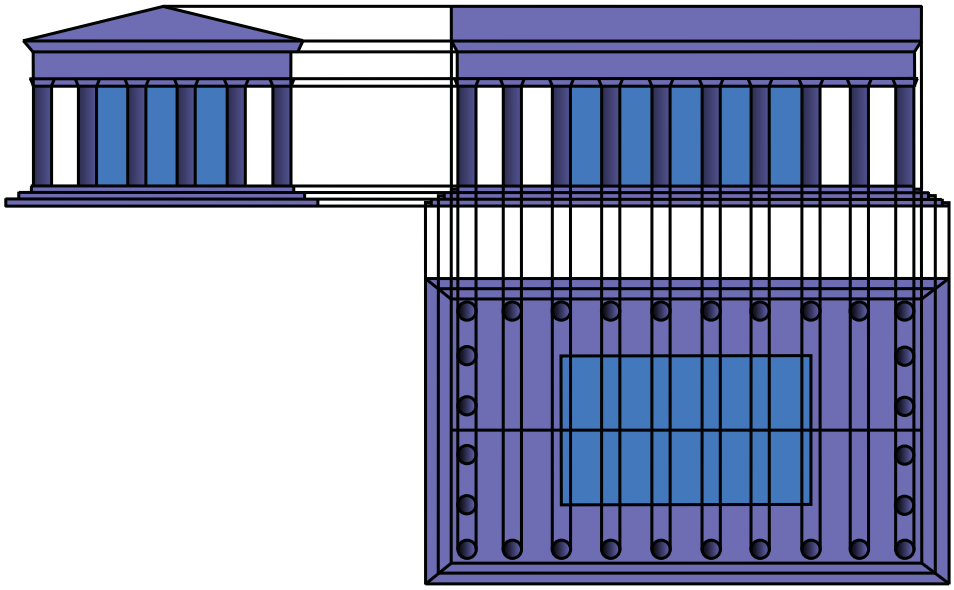


Figure 1.4: A successful reconstruction of a building ends up in a consistent and accurate construction plan.

1.2.2 Semantic Enrichment

Without semantic information scanned objects are nothing more than a heap of primitives. There would be no difference between the door knob of a house and a banana. Also, the amount of scanned data increases rapidly as the scanning process itself is now available to a wide audience. The importance of semantic metadata becomes obvious in the context of electronic product data exchange and storage or, more generally, of digital libraries. For a heap of primitives without valuable metadata, it is hard, if not impossible, to realize the mandatory services required by a digital library such as markup, indexing, and retrieval. In a very restricted and simple approach, this information may consist of attributes like title, creator, time of creation, and original place of the objects. But for many tasks this is not enough. For example, the Theseus Temple in Vienna consists of 28 Doric columns. To search for these types of columns in a large data set of scanned temples, the search engine needs to know that the temples consist of columns and what kind of columns they are. In order to allow navigation through the data sets (in 3D as well as on a semantic level), it is necessary to have a column's position within the data set of the temple. In analyzing the column, it might be necessary to find other artifacts of the same mason or the same period of time. In general, relations between objects are an essential aspect of working with cultural heritage. In 2000 the "International Committee for Documentation of the International Council of Museums" released the Conceptual Reference Model (CIDOC-CRM), a standard

for representing such relational semantic networks in cultural heritage. The CIDOC-CRM specification (in version 5.0.1) [Gro03] offers a formal ontology with 90 object classes for entities:

- actor (person)
- place
- event
- time-span
- man-made object

and 148 properties to describe all possible kinds of relations between objects that include the following:

- participated in
- performed
- at some time within
- took place at
- is referred to by
- has created.

Figure 1.5 shows a simplified cutout of a semantic network.

1.3 Open Problems and Overview

There are still many open problems in the described work flow. As the common scanning techniques are mostly optical, the scanning quality depends on viewing conditions. In complex structures such as a cathedral, there are always parts that are hard to reach (sculptural details) or scan (windows). Acceptable results are rarely produced without manual interference.

As described, the semantic enrichment is very important for working with the scanned objects. But the additional semantic information has to be extracted and added laboriously. This is a very difficult task which requires expertise. Although semantic information enables a simple search within 3D data, the problem to formulate an appropriate query is still an open question.

Another big problem dealing with cultural heritage objects is the long-term preservation. Common storage containers for digital data have a short life time compared to traditional (analog) storage like microfilms. Not only does the media have to last, but the software and hardware also have to remain available.

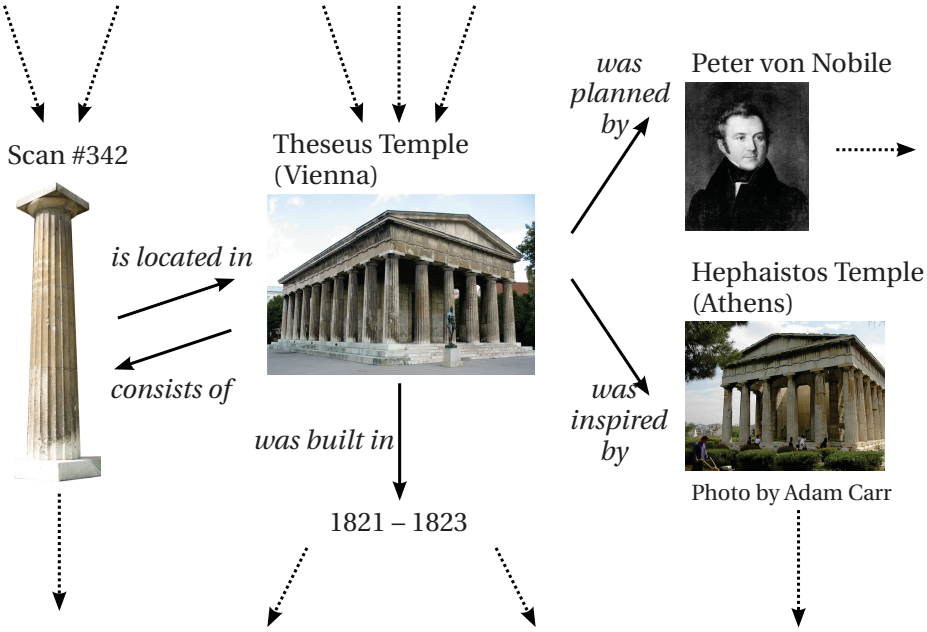


Figure 1.5: A semantic network helps when working with different cultural heritage objects. In this example, the column to the left was scanned and then connected to the semantic network of which only a cutout is shown.

This thesis concentrates on the geometric reconstruction and semantic information extraction aspect. It is divided into six main parts. The next chapter introduces the mathematical basics and notations; namely linear algebra, probability and statistics, and numerical optimization. The third chapter consists of an introduction to geometrical concepts. These techniques are the basis of computer-aided geometric design, which is presented in the subsequent chapter. The fifth chapter focuses on geometric fitting and semantic reconstruction techniques. Future prospects on this topic, given in the last chapter, complete this thesis.

Selected Readings on Digital Cultural Heritage

The work flow from an archaeological discovery into a digital cultural heritage library is a multifaceted process of substantial research. An overview of state-of-the-art techniques and open research questions can be found in "Recording, Modeling and Visualization of Cultural Heritage" by EMMANUEL BALTSAVIAS, ARMIN GRUEN, LUC VAN GOOL, and MARIA PATERAKI (Ed) [BGVGP06].

The scanning process is explained in "A Comparison of Systems and Tools for 3D Scanning" by FRANK TER HAAR, PAOLO CIGNONI, PATRICK MIN, and REMCO VELTKAMP in "3D Digital Imaging and Modeling: Applications of Heritage, Industry, Medicine and Land", [tHCMV05].

The resulting point clouds and meshes of the scanning process need to be aligned and registered to each other. "Exploiting the scanning sequence for automatic registration of large sets of range maps" by PAOLO PINGI, ANDREA FASANO, PAOLO CIGNONI, CLAUDIO MONTANI, and ROBERTO SCOPIGNO addresses this algorithmic problem [PFC+05].

The photogrammetric approach to acquire a 3D object is presented in RICHARD HARTLEY's and ANDREW ZISSERMAN's book "Multiple View Geometry in Computer Vision" published by Cambridge University Press [HZ04].

The generation of high-level geometry is the subject-matter of "Automatic Reconstruction of 3D CAD Models from Digital Scans" by FAUSTO BERNARDINI, CHANDRAJIT L. BAJAJ, JINDONG CHEN, and DANIEL R. SCHIKORE. This overview report has been published in the International Journal of Computational Geometry and Applications [BBCS99].

The geometric reconstruction is the basis for semantic information extraction. A state-of-the-art overview of this topic is summarized in "Multimedia Information Extraction" by MARK MAYBURY [May11].

Aspects on the digital cultural heritage libraries are discussed in ALFREDO RONCHI's book "eCulture – Cultural Content in the Digital Age" [Ron09].

Due to the complexity of the whole pipeline from an archaeological discovery into a digital cultural heritage library, this short bibliography cannot be complete. A starting point for current research activities can be found at the European Commission, unit "Cultural Heritage and Technology Enhanced Learning". EU-funded research (http://ec.europa.eu/research) on cultural heritage, digital libraries and digital preservation deals with leading-edge information and communication technologies for expanding access to and use of Europe's rich cultural and scientific resources. It also investigates how digital content created today will survive as the cultural and scientific knowledge of the future. This research contributes to the i2010 Digital Libraries Initiative: http://www.europeana.eu.

2 Mathematical Basis

This chapter summarizes some fundamental concepts and introduces some notations. The definitions and notations of linear algebra are mainly derived from "Geometric Concepts for Geometric Design" [BP94] by WOLFGANG BOEHM and HARTMUT PRAUTZSCH and "Wavelets for Computer Graphics" [SDS96] by ERIC J. STOLLNITZ, TONY D. DEROSE, and DAVID H. SALESIN.

The recapitulation of probability calculus and statistics is excerpted from NORBERT HENZE's lecture notes on stochastics "Stochastik – Einführung in die Wahrscheinlichkeitstheorie und Statistik" (english: Stochastics – Introduction to probability calculus and statistics) [Hen95] and from "Stochastik für Einsteiger" (english: Stochastics for Beginners) [Hen97].

Some numerical aspects of linear algebra and optimization theory complete this chapter. The short introduction on numerical analysis is based on THOMAS F. COLEMAN's "Large-Scale Numerical Optimization: Introduction and Overview" [Col91] and "Numerische Methoden der Analysis" (english: Numerical Methods of Analysis) [HP10]. Further details on numerical algorithms can be found in "Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation" [Nas90] and in "Numerical Optimization" [NW99].

Last but not least ERIC WEISSTEIN's "MathWorld"^{TM} [Wei09] and "Taschenbuch der Mathematik" (english: Handbook of Mathematics) [BSMM97] by IL'JA BRONŠTEIN et al. acted as referee.

Contents

2.1 Linear Algebra

Linear algebra introduces the fundamental concepts of vector spaces and is the foundation of vector-based descriptions of geometric objects. The axiomatic definition of the concept of vector spaces has been introduced by GIUSEPPE PEANO[1] in 1888 [Dor95].

2.1.1 Vector Space

A set V is a vector space over a field F, if it is closed under finite vector addition and scalar multiplication and if it meets the following conditions for all elements $\mathbf{u},\mathbf{v},\mathbf{w} \in V$ and any scalars $a,b \in F$.

1. Commutativity of vector addition:
$$\mathbf{u}+\mathbf{v}=\mathbf{v}+\mathbf{u} \tag{2.1}$$
2. Associativity of vector addition:
$$(\mathbf{u}+\mathbf{v})+\mathbf{w}=\mathbf{u}+(\mathbf{v}+\mathbf{w}) \tag{2.2}$$
3. Existence of zero vector:
$$\exists \mathbf{o} \in V \ : \ \mathbf{o}+\mathbf{u}=\mathbf{u} \tag{2.3}$$
4. Existence of additive inverse vector:
$$\forall \mathbf{u} \in V \ \exists -\mathbf{u} \in V \ : \ \mathbf{u}+(-\mathbf{u})=\mathbf{o} \tag{2.4}$$
5. Associativity of scalar multiplication:
$$a \cdot (b \cdot \mathbf{u}) = (a \cdot b) \cdot \mathbf{u} \tag{2.5}$$
6. Distributivity of scalar sums:
$$(a+b) \cdot \mathbf{u} = a \cdot \mathbf{u} + b \cdot \mathbf{u} \tag{2.6}$$
7. Distributivity of vector sums:
$$a \cdot (\mathbf{u}+\mathbf{v}) = a \cdot \mathbf{u} + a \cdot \mathbf{v} \tag{2.7}$$
8. Scalar multiplication identity:
$$1 \cdot \mathbf{u} = \mathbf{u} \tag{2.8}$$

A module is abstractly similar to a vector space, but it uses a ring to define scalar coefficients instead of a field. Therefore, the coefficients of modules are more general algebraic objects.

A nonempty subset U of a vector space V over a field F is called a vector subspace, if U is a vector space over F with the addition and multiplication defined on V.

[1] GIUSEPPE PEANO (August 27, 1858 – April 20, 1932) Giuseppe Peano was an Italian mathematician and a founder of mathematical logic and set theory. His axiomatization efforts are key contributions to modern mathematics.

2.1.2 Bases and Dimension

Some vectors $\mathbf{v}_1$,$\mathbf{v}_2$,... in a vector space V are said to be linearly independent, if the equation

$$c_1 \cdot \mathbf{v}_1 + c_2 \cdot \mathbf{v}_2 + \ldots = \mathbf{o} \tag{2.9}$$

is satisfied, if and only if all coefficients equal zero

$$c_1 = c_2 = \ldots = 0. \tag{2.10}$$

A collection of linearly independent vectors $\mathbf{v}_i$ of a vector space V form a basis for V, if every vector $\mathbf{u} \in V$ can be written as linear combination

$$\mathbf{u} = \sum_i c_i \mathbf{v}_i \tag{2.11}$$

for some coefficients c_i. The c_i are called coordinates of $\mathbf{u}$ with respect to the $\mathbf{v}_i$, while the products $c_i \mathbf{v}_i$ are called components of $\mathbf{u}$. If a basis for V has a finite number of elements $\mathbf{v}_1,\ldots,\mathbf{v}_n$, then V is finite-dimensional and its dimension is $\dim V = n$. Occasionally, n is given as a superscript V^n. A vector of a finite-dimensional vector space $\mathbf{w} \in V^n$ is often denoted by the vector of its coordinates $\mathbf{w} = \begin{pmatrix} w_1 & \ldots & w_n \end{pmatrix}^{\mathrm{T}}$.

All linear combinations of some vectors $\mathbf{v}_1,\ldots,\mathbf{v}_m \in V$ span a vector space. It is called linear hull and is written

$$[\mathbf{v}_1,\ldots,\mathbf{v}_m]. \tag{2.12}$$

The linear hull of a basis spans the whole vector space.

The so-called standard basis of the real, n-dimensional vector space $\mathbb{R}^n$ consists of the n unit vectors $\mathbf{e}_1,\ldots\mathbf{e}_n$. The coefficients of the j^{th}unit vector equal zero except for the j^{th}coefficient, which equals one.

2.1.3 Change of Bases

If $\mathbf{u}_1,\ldots,\mathbf{u}_n$ and $\mathbf{v}_1,\ldots,\mathbf{v}_n$ denote two bases of a vector space V^n, then each vector $\mathbf{u}_i$ can be written in terms of the basis $\mathbf{v}_1,\ldots,\mathbf{v}_n$ and vice versa:

$$\mathbf{u}_i = t_{1,i} \cdot \mathbf{v}_1 + \ldots + t_{n,i} \cdot \mathbf{v}_n \tag{2.13}$$

respectively in matrix notation

$$\begin{pmatrix} \mathbf{u}_1 & \ldots & \mathbf{u}_n \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1 & \ldots & \mathbf{v}_n \end{pmatrix} \cdot \underbrace{\begin{pmatrix} t_{1,1} & \ldots & t_{1,n} \\ \vdots & & \vdots \\ t_{n,1} & \ldots & t_{n,n} \end{pmatrix}}_{=\mathbf{T}}. \tag{2.14}$$

For an arbitrary vector $\mathbf{w}$ with representations

$$\mathbf{w} = \sum_{i=1}^{n} a_i \mathbf{u}_i = \sum_{i=1}^{n} b_i \mathbf{v}_i, \tag{2.15}$$

it follows

$$\begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \mathbf{T} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}. \tag{2.16}$$

2.1.4 Transformations

A map $\phi : V \to W$, which preserves linear combinations for all vectors $\mathbf{u},\mathbf{v} \in V$ and all scalars a,b

$$\phi(a \cdot \mathbf{u} + b \cdot \mathbf{v}) = a \cdot \phi(\mathbf{u}) + b \cdot \phi(\mathbf{v}) \tag{2.17}$$

is called a linear map. The image $\phi(V)$ of V is a vector subspace of W. For a finite-dimensional vector space V^n the image's dimension satisfies

$$\dim \phi(V) \leq n. \tag{2.18}$$

The subspace of all vectors of V, which are mapped to the zero vector $\mathbf{o} \in W$, is the kernel of ϕ, $K = \ker \phi$. Using a basis of V which contains a basis of K one finds that

$$\dim \phi(V) + \dim K = \dim V. \tag{2.19}$$

2.1.5 Inner Products and Orthogonality

An inner product $<\cdot|\cdot>$ of a vector space V is a map from $V \times V$ to the scalar field F that is

1. symmetric:

$$<\mathbf{u}|\mathbf{v}> = <\mathbf{v}|\mathbf{u}> \tag{2.20}$$

2. bilinear:

$$<a \cdot \mathbf{u} + b \cdot \mathbf{v}|\mathbf{w}> = a \cdot <\mathbf{u}|\mathbf{w}> + b \cdot <\mathbf{v}|\mathbf{w}> \tag{2.21}$$

3. positive definite:

$$<\mathbf{u}|\mathbf{v}> > 0, \qquad \forall\, \mathbf{u} \neq \mathbf{o} \tag{2.22}$$

for all vectors $\mathbf{u},\mathbf{v},\mathbf{w} \in V$ and all scalars $a,b \in F$.

In the vector space $\mathbb{R}^n$ the inner product of two vectors $\mathbf{u} = \begin{pmatrix} u_1 & \dots & u_n \end{pmatrix}^{\mathrm{T}}$ and $\mathbf{v} = \begin{pmatrix} v_1 & \dots & v_n \end{pmatrix}^{\mathrm{T}}$ is usually defined by

$$<\mathbf{u}|\mathbf{v}>=\sum_{i=1}^{n} u_i \cdot v_i. \tag{2.23}$$

In the vector space of all functions that are continuous on the closed interval [0,1] the "standard" inner product is defined as

$$<f\,|\,g>=\int_0^1 f(x)\cdot g(x)\,\mathrm{d}\,x. \tag{2.24}$$

Two vectors are said to be orthogonal, if their inner product equals zero. As a result, a collection of mutually orthogonal vectors $\mathbf{u}_1$, $\mathbf{u}_2$, ... must be linearly independent. An orthogonal basis is one that consists of pairwise orthogonal vectors. If V is an n-dimensional vector space and W a vector subspace of V, then the orthogonal complement is the vector space, which consists of all vectors in V that are orthogonal to all vectors in W.

2.1.6 Projection

A vector space V, which is composed of subspaces W_i $(i = 1,\dots,m)$ that only have the zero vector $\mathbf{o}$ in common

$$W_i \cap W_j = \{\mathbf{o}\}, \qquad\qquad i \neq j \tag{2.25}$$

is noted as

$$V = W_1 \oplus \dots \oplus W_m. \tag{2.26}$$

In this case, each vector $\mathbf{v} \in V$ has a unique representation by vectors $\mathbf{w}_i \in W_i$

$$\mathbf{v} = \mathbf{w}_1 + \dots + \mathbf{w}_n. \tag{2.27}$$

A function Π, which maps a vector $\mathbf{v}$ on a subset of its components $\mathbf{w}_1,\dots,\mathbf{w}_n$ is called projection; respectively, if all W_i are orthogonal to each other, Π is called orthogonal projection.

2.1.7 Normalization

A function $||\cdot||$ from a vector space V to the real numbers $\mathbb{R}$ such that

1. $||\mathbf{u}|| > 0$ when $\mathbf{u} \neq \mathbf{o}$ and $||\mathbf{u}|| = 0$, if and only if $\mathbf{u} = \mathbf{o}$,
2. $||a \cdot \mathbf{u}|| = |a| \cdot ||\mathbf{u}||$ for any scalar a
3. $||\mathbf{u} + \mathbf{v}|| \leq ||\mathbf{u}|| + ||\mathbf{v}||$

for all vectors $\mathbf{u}$,$\mathbf{v} \in V$ is called a vector norm or simply norm.

In the vector space $\mathbb{R}^n$ the family of p-norms

$$||\mathbf{u}||_p = \left(\sum_{i=1}^{n} |u_i|^p \right)^{1/p} \tag{2.28}$$

is of particular importance; especially the absolute value norm ($p = 1$), the Euclidean norm ($p = 2$) and the maximum norm (in the limit as p goes to infinity):

$$||\mathbf{u}||_1 = \sum_i |u_i| \qquad ||\mathbf{u}||_2 = \sqrt{\sum_i u_i^2} \qquad ||\mathbf{u}||_\infty = \max_i |u_i| \tag{2.29}$$

In the vector space of all functions that are continuous on [0,1] the L^p norms are defined as

$$||\phi||_p = \left(\int_0^1 |\phi(x)|^p \, \mathrm{d}\, x \right)^{1/p}. \tag{2.30}$$

The maximum norm for functions on the interval [0,1] is given by

$$||\phi||_\infty = \max_{x \in [0,1]} |\phi(x)|. \tag{2.31}$$

A vector $\mathbf{u}$ whose Euclidean norm equals one is said to be normalized. The Euclidean norm of a vector is called its length. In an arbitrary vector space V with an inner product $<\cdot|\cdot>$ the norm induced by the inner product is defined by

$$||\mathbf{v}||_{<\cdot|\cdot>} = \sqrt{<\mathbf{v}|\mathbf{v}>} \tag{2.32}$$

for all $\mathbf{v} \in V$.

An orthogonal basis of a vector space is called orthonormal, if it consists of normalized, pairwise orthogonal vectors. Using the notation of the Kronecker symbol δ

$$\delta_{i,j} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.33}$$

an orthonormal basis $\mathbf{u}_1, \mathbf{u}_2, \ldots$ satisfies $<\mathbf{u}_i|\mathbf{u}_j> = \delta_{i,j}$. For each basis $\mathbf{u}_1, \ldots, \mathbf{u}_n$ of $\mathbb{R}^n$ it is possible to construct an orthonormal basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ using the Gram-Schmidt orthonormalization. This process starts with a vector $\mathbf{v}_1 = \mathbf{u}_1$ and constructs $\mathbf{v}_i$ successively by subtracting the projections of $\mathbf{u}_i$ in the directions of $\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}$:

$$\mathbf{v}_i = \mathbf{u}_i - \sum_{j=1}^{i-1} \frac{<\mathbf{v}_j|\mathbf{u}_i>}{<\mathbf{v}_j|\mathbf{v}_j>} \mathbf{u}_j. \tag{2.34}$$

Having normalized the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ the resulting basis is orthonormal. To measure the angle α between arbitrary vectors $\mathbf{u}, \mathbf{v} \in V$ the definition

$$\cos\alpha = \frac{<\mathbf{u}|\mathbf{v}>}{||\mathbf{u}|| \cdot ||\mathbf{v}||}. \tag{2.35}$$

is used.

2.1.8 Eigenanalysis

A vector $\mathbf{v} \neq \mathbf{o}$ is said to be an eigenvector of a square matrix $\mathbf{M}$ with associated eigenvalue λ, if

$$\mathbf{M} \cdot \mathbf{v} = \lambda \mathbf{v}. \tag{2.36}$$

The set of all eigenvalues is called spectrum of $\mathbf{M}$. If and only if the the matrix $\mathbf{M}$ has n linearly independent eigenvectors, then $\mathbf{M}$ is diagonalizable, i.e. it can be written on the form

$$\mathbf{M} = \mathbf{T} \cdot \mathbf{D} \cdot \mathbf{T}^{-1}, \tag{2.37}$$

where $\mathbf{D}$ is a diagonal matrix with the n eigenvalues of $\mathbf{M}$ as its entries and $\mathbf{T}$ is an invertible matrix consisting of the corresponding eigenvectors. In the special case of a symmetric matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ with real entries,

- the matrix has exact n real eigenvalues according to multiplicity,
- the corresponding eigenvectors of two eigenvalues $\lambda_i \neq \lambda_j$ are orthogonal,
- furthermore the matrix $\mathbf{M}$ has n real, orthogonal eigenvectors.

2.1.9 Principal Component Analysis

The principal component analysis is an application of eigenanalysis. The main idea is to project given data points of an n-dimensional space into a lower dimensional space while preserving as much "information" as possible.

A vector in an n-dimensional vector space, whose orthonormal basis consists of $\mathbf{u}_1, \ldots, \mathbf{u}_n$, can be described via its coefficients c_i and its components $c_i \mathbf{u}_i$. To describe a set of N data vectors

$$\{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \} \subset \mathbb{R}^n, \qquad N \gg n \tag{2.38}$$

an additional center vector $\overline{\mathbf{x}}$ is used. Then, each data vector $\mathbf{x}_i$ is described by the center vector $\overline{\mathbf{x}}$ and an offset

$$\mathbf{x}_i = \overline{\mathbf{x}} + c_{i,1}\mathbf{u}_1 + c_{i,2}\mathbf{u}_2 + \ldots + c_{i,n}\mathbf{u}_n. \tag{2.39}$$

The principal component analysis determines the basis $\mathbf{u}_1, \ldots, \mathbf{u}_n$, so that the sum of quadratic errors of all offsets is minimal, if projected into a subspace of lower dimension $m < n$ with basis $\mathbf{u}_1, \ldots, \mathbf{u}_m$. As the orthogonal projection Π of a vector described by an orthonormal basis $\mathbf{u}_1, \ldots, \mathbf{u}_n$ is simply

$$\Pi(\mathbf{x}) = \Pi(c_1\mathbf{u}_1 + c_2\mathbf{u}_2 + \ldots + c_m\mathbf{u}_m + \ldots + c_n\mathbf{u}_n) \tag{2.40}$$

$$= c_1\mathbf{u}_1 + c_2\mathbf{u}_2 + \ldots + c_m\mathbf{u}_m, \tag{2.41}$$

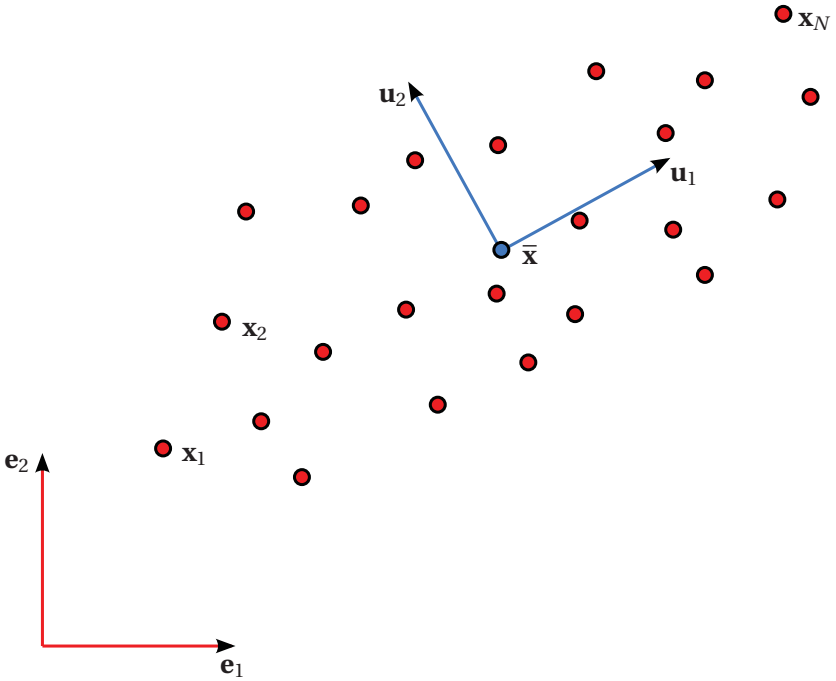


Figure 2.1: The principal component analysis is an eigenvector-based multivariate analysis. For a set of data vectors $\mathbf{x}_1, \ldots \mathbf{x}_N$ it determines an basis transformation from a given basis $\mathbf{e}_1$,$\mathbf{e}_2$ to a new, orthonormal basis $\mathbf{u}_1$,$\mathbf{u}_2$. Along the new coordinate axes the data set's variance has maximum amplitude.

the introduced error to minimize is

$$f_{PCA} = \sum_{i=1}^{N} ||\mathbf{x}_i - (\bar{\mathbf{x}} + c_{i,1}\mathbf{u}_1 + c_{i,2}\mathbf{u}_2 + \ldots + c_{i,m}\mathbf{u}_m)||^2 \tag{2.42}$$

where

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i. \tag{2.43}$$

In case of $m = n$, the error f_{PCA} would be zero due to

$$f_{PCA} = \sum_{i=1}^{N} ||\mathbf{x}_i - (\bar{\mathbf{x}} + c_{i,1}\mathbf{u}_1 + c_{i,2}\mathbf{u}_2 + \ldots + c_{i,m}\mathbf{u}_m)||^2 \tag{2.44}$$

$$= \sum_{i=1}^{N} ||c_{i,m+1}\mathbf{u}_{m+1} + c_{i,m+2}\mathbf{u}_{m+2} + \ldots + c_{i,n}\mathbf{u}_n)||^2 \tag{2.45}$$

$$= \sum_{j=m+1}^{n} \sum_{i=1}^{N} \left(\mathbf{u}_j^{\mathrm{T}} \cdot (\mathbf{x}_i - \bar{\mathbf{x}})\right)^2 = \sum_{j=m+1}^{n} \mathbf{u}_j^{\mathrm{T}} \cdot \mathbf{C} \cdot \mathbf{u}_j \tag{2.46}$$

with the so-called covariance matrix

$$\mathbf{C} = \sum_{i}^{N} (\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^{\mathrm{T}}. \tag{2.47}$$

Using Lagrangian multipliers to minimize $\mathbf{u}^{\mathrm{T}} \cdot \mathbf{C} \cdot \mathbf{u}$ subject to $||\mathbf{u}|| = 1$ leads to a minimization term

$$\mathbf{u}^{\mathrm{T}} \cdot \mathbf{C} \cdot \mathbf{u} - \lambda \mathbf{u}^{\mathrm{T}} \cdot \mathbf{u} \tag{2.48}$$

whose gradient equals zero, if $\mathbf{u}$ is substituted by an eigenvector of $\mathbf{C}$. Having sorted the eigenvectors $\mathbf{u}_i$ of $\mathbf{C}$ according to the absolute values of the corresponding eigenvalues λ_i justifies

$$f_{PCA} = \sum_{j=m+1}^{n} \mathbf{u}_j^{\mathrm{T}} \cdot \mathbf{C} \cdot \mathbf{u}_j = \sum_{j=m+1}^{n} \lambda_j. \tag{2.49}$$

2.1.10 Multiresolution Analysis

The starting point for the mathematical framework of multiresolution analysis is a nested set of vector spaces

$$V^0 \subset V^1 \subset V^2 \subset \dots . \tag{2.50}$$

The basis functions for the space V^j are known as scaling functions. The orthogonal complement of V^j in V^{j+1} is called W^j and contains all functions in V^{j+1} that are orthogonal to all those in V^j according to a chosen inner product. The basis functions of W^j are called wavelets.

In order to keep track of the different scaling functions ϕ_i^j of V^j and the wavelets ψ_i^j of W^j, they are combined to vectors:

$$\Phi^j = \begin{pmatrix} \phi_0^j & \dots & \phi_{\dim V^j - 1}^j \end{pmatrix}^{\mathrm{T}}, \quad \Psi^j = \begin{pmatrix} \psi_0^j & \dots & \psi_{\dim W^j - 1}^j \end{pmatrix}^{\mathrm{T}}. \tag{2.51}$$

As the subspaces V^j are nested, the scaling functions are refinable; that means for all $j = 1,2,\dots$ exists a matrix $\mathbf{P}^j$ such that $\Phi^{j-1}(x) = \Phi^j(x) \cdot \mathbf{P}^j$. Since the wavelet space W^{j-1} is also a subspace of V^j, a matrix $\mathbf{Q}^j$ exists satisfying $\Psi^{j-1}(x) = \Phi^j(x) \cdot \mathbf{Q}^j$. In block matrix notation this can be expressed by

$$\left(\Phi^{j-1} \,|\, \Psi^{j-1}\right) = \Phi^j \left(\mathbf{P}^j \,|\, \mathbf{Q}^j\right). \tag{2.52}$$

Figure 2.2 illustrates the relations between vector spaces V and wavelet spaces W.

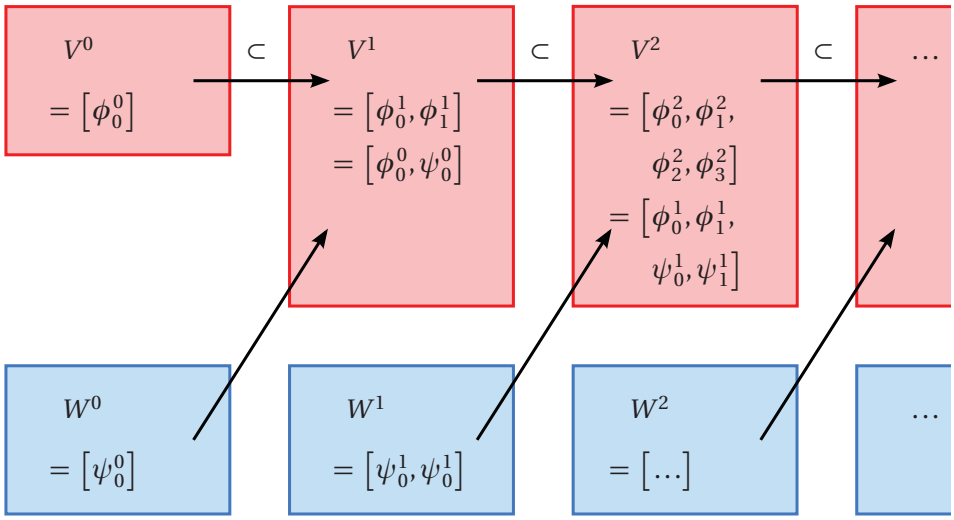


Figure 2.2: The mathematical framework of multiresolution analysis is based on nested vector spaces $V^0 \subset V^1 \subset V^2 \subset \ldots$ and orthogonal vector spaces W^0, W^1, W^2, …. Therefore, a vector in some vector space V^j can be described by scaling functions ϕ^m, wavelets ψ^n, or a combination of them.

A function in some approximation space V^j can be expressed by its coefficients c_i^j in terms of a scaling function basis

$$\mathbf{c}^j = \begin{pmatrix} c_0^j & \ldots & c_{\dim V^j - 1}^j \end{pmatrix}^{\mathrm{T}}. \tag{2.53}$$

A low resolution version $\mathbf{c}^{j-1}$ with a smaller number of coefficients can be created by downsampling – a filtering process describable by a matrix equation $\mathbf{c}^{j-1} = \mathbf{A}^j \cdot \mathbf{c}^j$. For many choices of $\mathbf{A}^j$, it is possible to capture the lost details as a vector $\mathbf{d}^{j-1}$, computed by $\mathbf{d}^{j-1} = \mathbf{B}^j \cdot \mathbf{c}^j$. If $\mathbf{A}^j$ and $\mathbf{B}^j$ (the so-called analysis filters) are chosen appropriately, $\mathbf{c}^j$ can be recovered from $\mathbf{c}^{j-1}$ and $\mathbf{d}^{j-1}$ using $\mathbf{P}^j$ and $\mathbf{Q}^j$, which are called synthesis filters:

$$\mathbf{c}^j = \mathbf{P}^j \cdot \mathbf{c}^{j-1} + \mathbf{Q}^j \cdot \mathbf{d}^{j-1}. \tag{2.54}$$

As the matrices $\mathbf{A}^j$ and $\mathbf{B}^j$ satisfy the relation

$$\left(\Phi^{j-1} \,|\, \Psi^{j-1}\right) \cdot \left(\frac{\mathbf{A}^j}{\mathbf{B}^j}\right) = \Phi^j, \tag{2.55}$$

it follows

$$\left(\frac{\mathbf{A}^j}{\mathbf{B}^j}\right) = \left(\mathbf{P}^j \,|\, \mathbf{Q}^j\right)^{-1}. \tag{2.56}$$

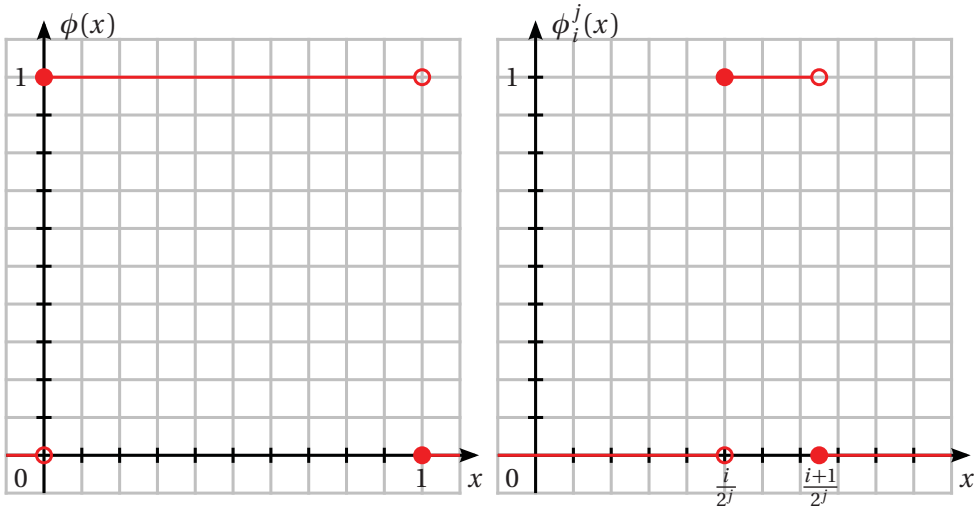


Figure 2.3: The simplest wavelet functions are called Haar wavelets. Their corresponding scaling functions $\phi_i^j(x)$ are based on a piecewise constant function $\phi(x)$. This Figure shows the function graphs of ϕ (left) and ϕ_i^j (right).

The simplest wavelet basis is named after ALFRÉD HAAR[2]. The scaling functions are a set of scaled and translated box functions:

$$\phi_i^j(x) = \phi(2^j x - i), \qquad i = 0, \ldots, 2^j - 1 \tag{2.57}$$

where

$$\phi(x) = \begin{cases} 1, & \text{if } 0 \le x < 1 \\ 0, & \text{otherwise.} \end{cases} \tag{2.58}$$

The support of a function refers to the region of the parameter domain over which the function value is nonzero. The function plots in Figure 2.3 show that the functions ϕ_i^j are supported over a bounded interval. The wavelets corresponding to the box basis are known as the Haar wavelets:

$$\psi_i^j(x) = \psi(2^j x - i), \qquad i = 0, \ldots, 2^j - 1 \tag{2.59}$$

with

$$\psi(x) = \begin{cases} 1, & \text{if } 0 \le x < 1/2 \\ -1, & \text{if } 1/2 \le x < 1 \\ 0, & \text{otherwise} \end{cases} \tag{2.60}$$

The Figure 2.4 show the functions' graphs.

[2] ALFRÉD HAAR (October 11, 1885 – March 16, 1933) Alfréd Haar was a Hungarian mathematician. He worked on analysis on groups, orthogonal systems of functions, partial differential equations and Chebyshev approximations.

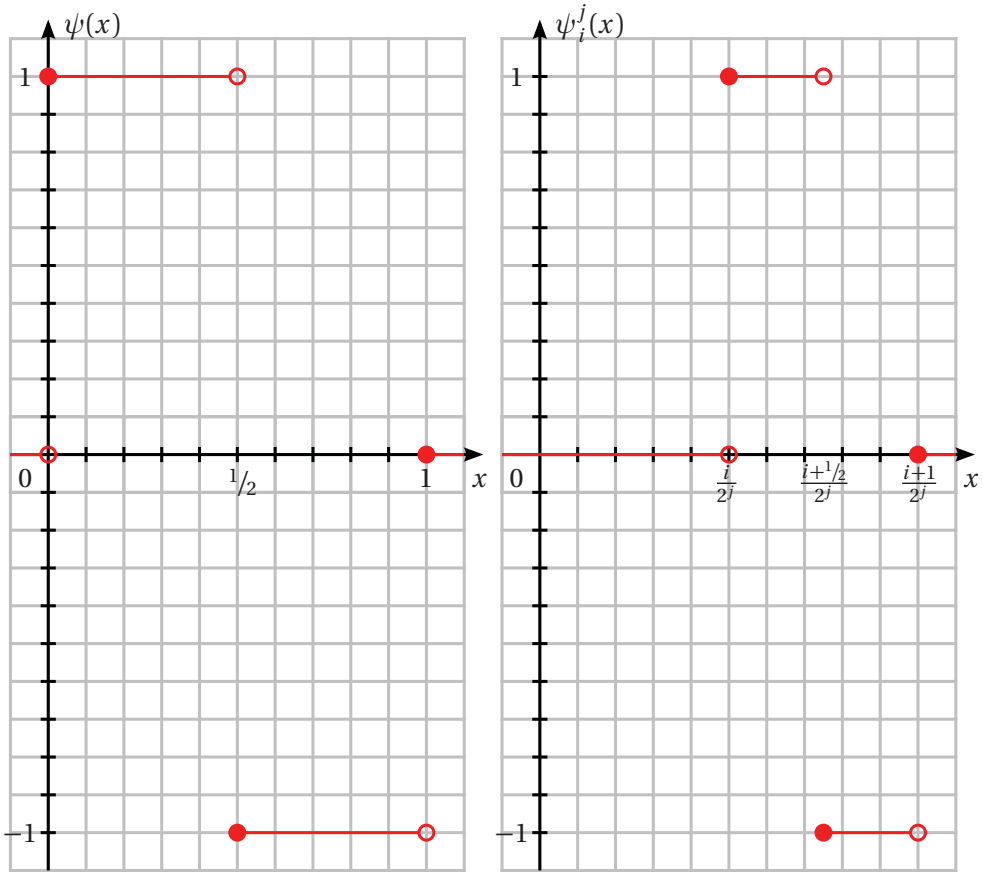


Figure 2.4: The piecewise constant function $\psi(x)$ returns the values -1, 0, and 1. Using this function the Haar wavelet functions ψ_i^j can be defined by $\psi_i^j(x) = \psi(2^j x - i)$.

The scaling functions ϕ_i^j and the wavelets ψ_i^j have some very important properties.

1. The vector space V^{j+1} can described by a basis of scaling functions ϕ_i^{j+1} as well as by a combination of the bases of V^j and W^j.

2. Every wavelet ψ_i^j in W^j is orthogonal to every scaling function ϕ_i^j of V^j.

3. If the scaling functions ϕ_i^j and the wavelets ψ_i^j are multiplied by a constant scaling factor of $\sqrt{2^j}$, both functions can be normalized; i.e.

$$\left\langle \sqrt{2^j}\,\phi_i^j \,\middle|\, \sqrt{2^j}\,\phi_i^j \right\rangle = 1 \qquad \text{and} \qquad \left\langle \sqrt{2^j}\,\psi_i^j \,\middle|\, \sqrt{2^j}\,\psi_i^j \right\rangle = 1. \tag{2.61}$$

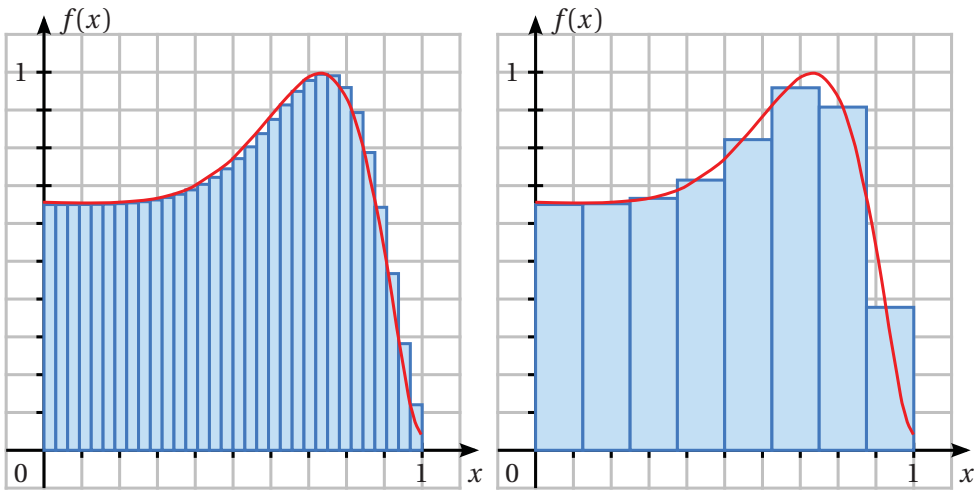


Figure 2.5: A wavelet transformation describes a function via scaling and detail coefficients. Using a reduced number of coefficients or setting some coefficients to zero approximates the original function. This Figure shows the graph of a function $f(x)$ (in red) and a sampling of it (blue, left) as well as an approximation of it (blue, right). The approximation uses a reduced number of detail coefficients.

The Haar wavelets form the simplest type of wavelets. This simplicity is revealed in the transformation algorithms. A one-dimensional Haar wavelet decomposition is shown in Algorithms 2.1 and 2.2; the corresponding reconstruction in Algorithms 2.3 and 2.4. The multiresolution property of wavelets is shown in Figure 2.5. It shows a continuous function, a discrete, regular sampling of it and its representation at a lower level of details.

Depending on the purposes other wavelet functions may be more reasonable – for example basis functions without discontinuity. INGRID DAUBECHIEShas derived and analyzed various families of wavelet functions [Dau92].

Algorithm 2.1 A single Haar wavelet decomposition step.

```
1  DECOMPOSITION-STEP
2     input   c[1..2^j] : input coefficients
3     output  d[1..2^j] : transformed coefficients
4
5     for i ← 1 to 2^(j−1)
6        d[i] ← (c[2i−1]+c[2i]) / √2
7        d[2^(j−1)+i] ← (c[2i−1]−c[2i]) / √2
8     return d
```

Algorithm 2.2 Haar wavelet decomposition in 1D.

```
1  DECOMPOSE-1D
2     input   c[1..2^j] : input coefficients
3     output  d[1..2^j] : wavelet coefficients
4
5     d ← c/√(2^j)
6     g ← 2^j
7     while g ≥ 2
8        d[1..g] ← DECOMPOSITION-STEP (d[1..g])
9        g ← g/2
10    return d
```

Algorithm 2.3 A single Haar wavelet reconstruction step.

```
1  RECONSTRUCTION-STEP
2     input   d[1..2^j] : input coefficients
3     output  c[1..2^j] : transformed coefficients
4
5     for i ← 1 to 2^(j−1)
6        c[2i − 1] ← (d[i] + d[2^(j−1) + i]) / √2
7        c[2i] ← (d[i] − d[2^(j−1) + i]) / √2
8     return c
```

Algorithm 2.4 Haar wavelet reconstruction in 1D.

```
1  RECONSTRUCT-1D
2     input   d[1..2^j] : wavelet coefficients
3     output  c[1..2^j] : reconstructed coefficients
4
5     c ← d
6     g ← 2
7     while g ≤ 2^j
8        c[1..g] ← RECONSTRUCTION-STEP (c[1..g])
9        g ← 2 · g
10    c ← √(2^j) · c
11    return c
```

For higher dimensional wavelet transformations different extensions are known. Two main approaches for extension from one-dimensional signals represented by $\mathbb{R}^n$ to two-dimensional signals in $\mathbb{R}^{n\times m}$ exist: the standard approach and the nonstandard approach.

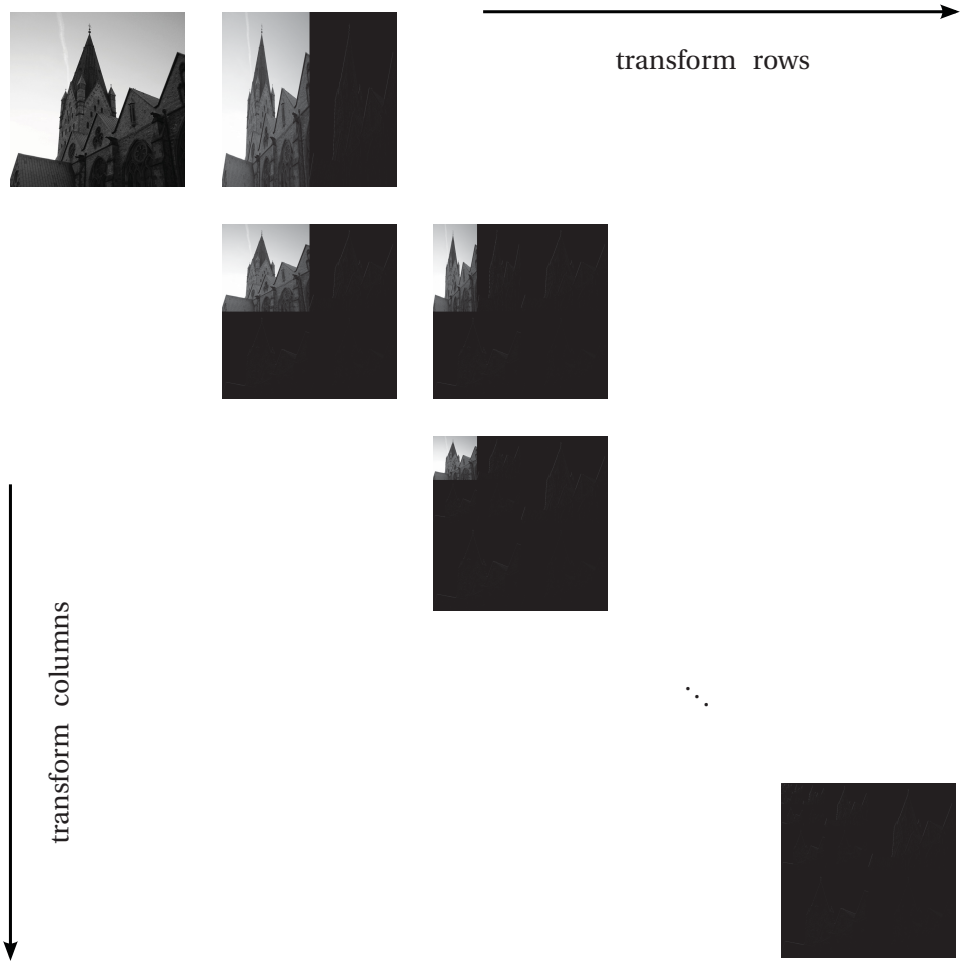


Figure 2.6: A two-dimensional data set such as an image can be Haar wavelet transformed in various ways. The so-called standard approach performs a wavelet transformation on all rows. Then it transforms all columns. The nonstandard approach, which is illustrated in this diagram, performs the wavelet transformation steps on rows and columns alternately. As a consequence down-scaled versions of the original image can be found in all intermediate results on the diagonal.

Each subimage shows the absolute values of the signal mapped to gray values. In general, the detail coefficients of a wavelet transformed signal have smaller values than the original signal; the transformed images therefore are very dark.

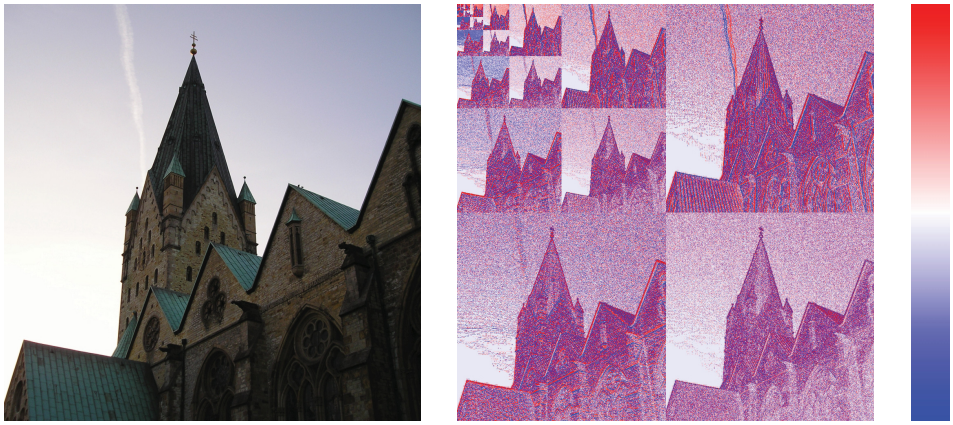


Figure 2.7: In general the detail coefficients of a wavelet transformed signal have smaller values than the original signal. Therefore the commonly-used linear gray map results in an almost black image. This Figure shows a color image of the input data set and its wavelet transformation using a nonlinear color map from blue (negative values) to red (positive values) via white (zero).

The standard approach decomposes $\mathbb{R}^{n\times m}$ into $(\mathbb{R}^n)^m$ and $(\mathbb{R}^m)^n$, whereas the nonstandard approach uses rectangular subregions. An implementation of the nonstandard approach – see Algorithms 2.5 and 2.6 – transforms the rows and columns of the input matrix alternately.

Figure 2.6 illustrates the wavelet transformation of an example data set via nonstandard approach. The example data set is a photograph of the Paderborn Cathedral St. Liborius. The cathedral has been built in the 13th century. It is the Cathedral of the Catholic Archdiocese of Paderborn. It is located in the city center of Paderborn, Germany.

Each subimage shows the absolute values of the signal mapped to gray values. In general, the detail coefficients of a wavelet transformed signal have smaller values than the original signal; the transformed images therefore are very dark. To visualize the wavelet transformed data more appropriately Figure 2.7 uses a nonlinear color map from blue (negative values) to red (positive values) via white (zero).

Algorithm 2.5 Nonstandard Haar wavelet decomposition in 2D.

```
1  DECOMPOSE-2D
2     input   c[1..2^j][1..2^j] : input coefficients
3     output  d[1..2^j][1..2^j] : wavelet coefficients
4
5     d ← c/2^j
6     g ← 2^j
7     while g ≥ 2
8        for row ← 1 to g
9           d[row][1..g] ← DECOMPOSITION-STEP (d[row][1..g])
10       for col ← 1 to g
11          d[1..g][col] ← DECOMPOSITION-STEP (d[1..g][col])
12       g ← g/2
13    return d
```

Algorithm 2.6 Nonstandard Haar wavelet reconstruction in 2D.

```
1  RECONSTRUCT-2D
2     input   d[1..2^j][1..2^j] : wavelet coefficients
3     output  c[1..2^j][1..2^j] : reconstructed coefficients
4
5     c ← d
6     g ← 2
7     while g ≤ 2^j
8        for col ← 1 to g
9           c[1..g][col] ← RECONSTRUCTION-STEP (c[1..g][col])
10       for row ← 1 to g
11          c[row][1..g] ← RECONSTRUCTION-STEP (c[row][1..g])
12       g ← 2 · g
13    c ← 2^j · c
14    return c
```

Wavelets for Image Compression

Uncompressed images need a high amount of storage space. The aim of compression techniques is to encode images – or more general signals – in a more efficient way. Wavelets offer a way to store images in a condensed form [Sah00]. The basic principle uses the fact that a wavelet transformed signal consists of many coefficients, which are almost zero. If these values are set to zero and if only the nonzero values are stored, the number of coefficients to store can be reduced significantly. The introduced error according to L^2-norm is proportional to the sum of the absolute values of all coefficients set to zero. A lossy, wavelet-based image compression uses this fact and consists of three steps:

1. Perform wavelet transformation and reduce 2D data set to 1D data set via standard, non-standard [SDS96] or fractal [Voo91] approach.
2. Sort absolute values of coefficients in decreasing order.
3. Set coefficients to zero starting with the smallest ones as long as the sum of absolute values of exchanged coefficients is smaller than a threshold.

The JPEG2000 image encoding standard [CS00] of the Joint Photographic Experts Group (JPEG) uses Daubechies-5/3 wavelets for lossless compression and Daubechies-9/7 wavelets for lossy compressions [Dau92]. For the sake of simplicity the following example uses Haar wavelets.

The example image in Figure 2.8 shows the marble statue of councilor JOAN FIVELLER created by JOSEP BOVER. The statue is part of the principal facade of the city hall in Barcelona, Spain. Its image is shown in full resolution (top) and with a reduced number of nonzero detail coefficients.

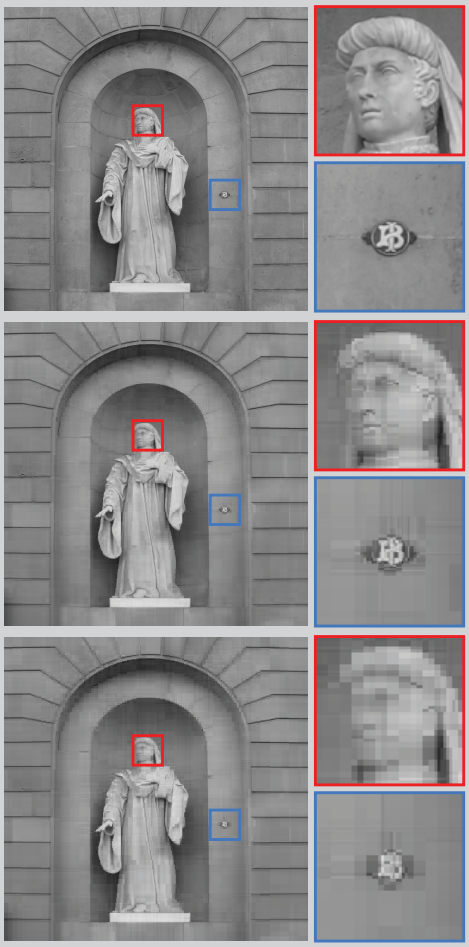


Figure 2.8: This sequence of images shows a lossy compression based on Haar wavelets using 100.0% (no compression), 1.0% and 0.5% of all coefficients. The zoomed subimages of the head and the official seal show the increasing compression artifacts.

2.2 Probability and Statistics

Probability theory is concerned with random phenomena and stochastic processes. As it is the mathematical foundation for statistics, probability theory is essential to many fields of applications that involve quantitative analysis of data.

2.2.1 Probability Space

Let Ω be a nonempty set. Then $\mathscr{F}$ is called a σ-algebra, if

1. $\mathscr{F}$ contains the empty set $\emptyset \in \mathscr{F}$.
2. If A is in $\mathscr{F}$, then its complement (in Ω) $\overline{A} = \Omega \backslash A$ also belongs to $\mathscr{F}$.
3. For an arbitrary sequence $(A_n)_n$ of subsets of $\mathscr{F}$ the union $\bigcup_{i \in N} A_i$ is in $\mathscr{F}$.

In a sample space Ω each subset $A \subset \Omega$ which belongs to $\mathscr{F}$ is called an event and is associated with a probability measure P, which obeys the axioms of probability:

1. $\forall A \in \mathscr{F} \ : \ P(A) \geq 0$,
2. $P(\Omega) = 1$,
3. $P\left(\sum_{j=1}^{\infty} A_j\right) = \sum_{j=1}^{\infty} P\left(A_j\right)$ for all sequences $(A_n)_{n \in \mathbb{N}}$ of pairwise disjunct events.

The triple $(\Omega, \mathscr{F}, P)$ is called probability space. In each probability space the monotony property

$$\forall A, B \in \mathscr{F} \ : \ A \subset B \Rightarrow P(A) \leq P(B) \tag{2.62}$$

is valid. The conditional probability of an event A assuming that B has occurred is denoted $P(A|B)$. It can be calculated via

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) \cdot P(B|A)}{P(B)}. \tag{2.63}$$

If the conditional probability $P(A|B)$ of an event A assuming event B satisfies the equation $P(A|B) = P(A)$, the events A and B are called statistically independent. In this case

$$P(AB) = P(A \cap B) = P(A) \cdot P(B). \tag{2.64}$$

More generally, if n events are independent, then

$$P\left(\bigcap_{i=1}^{n} A_i\right) = \prod_{i=1}^{n} P(A_i). \tag{2.65}$$

According to the total probability theorem the conditional probabilities $P(B|A_i)$ of an arbitrary event B and n mutually exclusive events A_1, $dots$, A_n sum to

$$P(B) = P(B|A_1) \cdot P(A_1) + \ldots + P(B|A_n) \cdot P(A_n). \tag{2.66}$$

Bayes' theorem – named after THOMAS BAYES[3] – describes the relation between a-priori and a-posteriori probabilities. For a partition of Ω in mutually exclusive events $A_1, \ldots, A_n$ and an arbitrary event B Bayes' theorem states

$$P(A_i|B) = \frac{P(B|A_i) \cdot P(A_i)}{\sum_{j=1}^{n} P(B|A_j) \cdot P(A_j)} = \frac{P(B|A_i) \cdot P(A_i)}{P(B)}. \tag{2.67}$$

A real function whose domain is the probability space and for which

1. the set $\{X \leq x\}$ is an event for any real number x and for which
2. the probability of the events $\{X = -\infty\}$ and $\{X = \infty\}$ equals zero,

is called random variable. Its probability distribution describes the range of possible values it can attain and the probability that the value of the random variable is within any measurable subset of that range.

2.2.2 Discrete Probabilities

A σ-algebra $\mathscr{F}$ is discrete, if a set of subsets $(A_i)_{i \in I}$ of Ω exists with

1. the index set I is not empty $\emptyset \neq I \subset \mathbb{N}$,
2. $\forall\, i \neq j \;:\; A_i \cap A_j = \emptyset$, and
3. $\Omega = \bigcup_{i \in I} A_i$,

so that every element $A \in \mathscr{F}$ can be described by a union of some A_j. Furthermore, if the set $\{A_i | i \in I\}$ is finite, the probability space $(\Omega, \mathscr{F}, P)$ is called finite.

For random variables in discrete probability spaces the characteristic values of expectation and variance are of importance. If for a random variable $X : \Omega \to \mathbb{R}$ of a discrete probability space $(\Omega, \mathscr{F}, P)$ the sum $\sum_{\omega \in \Omega} X(\omega) \cdot P(\{\omega\}) < \infty$ converges, the expectation value of X is defined by

$$E(X) = \sum_{\omega \in \Omega} X(\omega) \cdot P(\{\omega\}). \tag{2.68}$$

The variance of a probability variable X of a discrete probability space $(\Omega, \mathscr{F}, P)$ is

$$V(X) = E((X - E(X))^2). \tag{2.69}$$

[3] THOMAS BAYES (1702 – April 17, 1761) Thomas Bayes was a British mathematician and Presbyterian minister.

In stochastics and statistics it is convenient to omit brackets, if the short notation does not cause any confusion. The variance is then written $V(X) = E(X - EX)^2$. The square root of $V(X)$ is called standard deviation. It is noted

$$\sigma(X) = \sqrt{V(X)}. \tag{2.70}$$

To the vector space $L(\Omega, P) = \{X : \Omega \to \mathbb{R} | E(X) \text{ exists}\}$ of random variables with expectation value the following equations apply:

$$E(a \cdot X) = a \cdot E(X), (a \in \mathbb{R}) \qquad E(1) = 1 \tag{2.71}$$

$$E(X + Y) = E(X) + E(Y) \qquad E(X \cdot Y) = E(X) \cdot E(Y) \tag{2.72}$$

$$E(|X|) = |E(X)| \qquad X \leq Y \Rightarrow E(X) \leq E(Y) \tag{2.73}$$

As a result the variance can also be calculated via

$$V(X) = E(X - EX)^2 = E(X^2) - (EX)^2. \tag{2.74}$$

2.2.3 Discrete Distributions

The following description summarizes some important, discrete distributions which can be found in various fields of applications.

Laplace The discrete Laplace distribution is named after PIERRE-SIMON LAPLACE. It describes a discrete, uniform distribution. If $\Omega = \{w_1, \ldots, w_n\}$ and $P(\{w_i\}) = 1/n$, then for any set $A \subset \Omega$ the probability $P(A)$ can be calculated by $P(A) = \frac{|A|}{|\Omega|}$, whereas $|\cdot|$ denotes the cardinal number of a set. This distribution is normalized, since the sum of all probabilities $1/n$ equals one. In case of $X(w_i) = i$ its characteristics are

$$E(X) = \frac{1}{n} \sum_{i=i}^{n} x_i = \frac{n+1}{2} \tag{2.75}$$

and

$$V(X) = \frac{1}{n} \left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2 \right) = \frac{n^2 - 1}{12}. \tag{2.76}$$

Binomial In a sequence of n experiments, in which each experiment is true with probability p and false with probability $1 - p$, the probability of a random variable X to have exactly k successes is described by the binomial distribution

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad k = 0, 1, \ldots, n \tag{2.77}$$

with the binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. The notation for a random variable of binomial distribution with parameters n and p is $X \sim Bin(n,p)$.

Due to the binomial theorem

$$\sum_{k=0}^{n} \binom{m}{k} p^k (1-p)^{n-k} = 1, \tag{2.78}$$

the binomial distribution is normalized. Its expectation value is

$$E(X) = \sum_{k=0}^{n} k \binom{n}{k} k^p (1-p)^{n-k} \tag{2.79}$$

$$= n \cdot p \tag{2.80}$$

and its variance is

$$V(X) = \sum_{k=0}^{n} k^2 \binom{n}{k} p^k (1-p)^{n-k} - n^2 p^2 \tag{2.81}$$

$$= n \cdot p \cdot (1-p). \tag{2.82}$$

Geometric The geometric distribution for $k = 0,1,2,\ldots$ is defined by

$$P(X = k) = p \cdot (1-p)^k \tag{2.83}$$

and can be interpreted as "waiting for the first event", which occurs with probability p. The short notation expresses a geometric distribution as a special case of a negative binomial distribution (Nb). Therefore, its abbreviation is $X \sim Nb(1,p)$.

Using the geometric series $\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$, $(|r| < 1)$ its main characteristics can be calculated easily. P is normalized

$$\sum_{k=0}^{\infty} P(k) = \sum_{k=0}^{\infty} p \cdot (1-p)^k = p \sum_{k=0}^{\infty} (1-p)^k = \frac{p}{1-(1-p)} = 1 \tag{2.84}$$

and its expectation value and variance are

$$E(X) = \frac{1-p}{p} \tag{2.85}$$

$$V(X) = \frac{1-p}{p^2}. \tag{2.86}$$

Hypergeometric In a set of $r+s$ elements ($r,s \in \mathbb{N}$), of which r elements are tagged, the hypergeometric distribution describes the probability that of n randomly chosen elements exactly k elements are tagged. This probability, with parameters n, r, and s with short notation $X \sim Hyp(n,r,s)$, is

$$P(X=k) = \frac{\binom{r}{k}\binom{s}{n-k}}{\binom{r+s}{n}}. \tag{2.87}$$

Its expectation value and its variance are

$$E(X) = \sum_{k=0}^{n} k \cdot \frac{\binom{r}{k}\binom{s}{n-k}}{\binom{r+s}{n}} \tag{2.88}$$

$$= n \cdot \frac{r}{r+s} \tag{2.89}$$

$$V(X) = \sum_{k=0}^{n} k^2 \cdot \frac{\binom{r}{k}\binom{s}{n-k}}{\binom{r+s}{n}} - \left(n \cdot \frac{r}{r+s}\right)^2 \tag{2.90}$$

$$= n\frac{r}{r+s}\left(1-\frac{r}{r+s}\right)\frac{r+s-n}{r+s-1}. \tag{2.91}$$

More commonly used distributions are presented in "A Compendium of Common Probability Distributions" [McL99] by MICHAEL P. MCLAUGHLIN.

2.2.4 Continuous Probabilities

The σ-algebra consisting of all open sets in $\mathbb{R}^k$ is called the Borel σ-algebra B^k. If P is a probability measure on B^1, then the function

$$F : \begin{cases} \mathbb{R} & \to & [0,1] \\ x & \mapsto & F(x) = P((-\infty,x]) \end{cases} \tag{2.92}$$

is called distribution function. Any distribution function is

1. monotonic increasing; that means $\forall\, x,y, \in \mathbb{R} \;:\; x \le y \Rightarrow F(x) \le F(y)$
2. continuous from the right and
3. the limits $\lim_{n\to\infty} F(-n) =: F(-\infty) = 0$ and $\lim_{n\to\infty} F(n) =: F(\infty) = 1$ exist.

If a function $F : \mathbb{R} \to [0,1]$ has the these properties (1 – 3), then exactly one probability measure P on B^1 exists with $F(x) = P((-\infty,x])$, $x \in \mathbb{R}$. A distribution function F and its corresponding probability measure P on B^1 have the following properties:

- $\forall\, a,b \in \mathbb{R}, a < b \;:\; P((a,b]) = F(b) - F(a)$
- $\forall\, x \in \mathbb{R} \;:\; P(x) = F(x) - F(x-)$,
  whereas $F(x-)$ denotes to limit point at $F(x)$ from the left.
- F is continuous at x, if and only if $P(x) = 0$.
- F has at most countable points of discontinuity.

A Function $f : \mathbb{R} \to \mathbb{R}$ is called density function on $\mathbb{R}$, if

- $\forall\, x \in \mathbb{R} \;:\; f(x) \geq 0$ and if
- the Riemann integral of f exists with $\int_{-\infty}^{\infty} f(x)\,\mathrm{d}\,x = 1$.

The function

$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}\,t, \qquad\qquad x \in \mathbb{R} \tag{2.93}$$

has codomain [0,1], is continuous and has the distribution function properties 1 – 3. The corresponding probability measure P on B^1 is

$$P([a,b]) = \int_{a}^{b} f(t)\,\mathrm{d}\,t, \qquad\qquad (a,b \in \mathbb{R}, a < b). \tag{2.94}$$

The function f is called probability density function of P respectively density of F. A random variable X with probability density function f has an expectation value, if and only if

$$\int_{-\infty}^{\infty} |x| \cdot f(x)\,\mathrm{d}\,x < \infty. \tag{2.95}$$

In this case

$$E(X) = \int_{-\infty}^{\infty} x \cdot f(x)\,\mathrm{d}\,x. \tag{2.96}$$

Its variance can be calculated via

$$V(X) = \int_{-\infty}^{\infty} (x - EX)^2 \cdot f(x)\,\mathrm{d}\,x. \tag{2.97}$$

The equations (2.71) – (2.74) do also apply to the continuous definition of expectation value and variance.

Random Sample Consensus

A simple and elegant conceptual framework to estimate parameters is the Random Sample Consensus (RANSAC) paradigm by MARTIN A. FISCHLER and ROBERT C. BOLLES [FB81]. This technique is capable of extracting a variety of different models out of unstructured, noisy, sparse, and incomplete data.

RANSAC-based algorithms proceed by randomly taking (ideally few) samples, calculating the free parameters of a model (for example the four parameters of a plane). Then all samples of the input data set "vote", whether they agree with the hypothesis (if they are close enough to the suggested plane). This procedure is repeated a few times, and the hypothesis with the highest acceptance rate wins by "consensus".

Algorithm 2.7 outlines this principle. Samples, which agreed to a hypothesis, can be removed from the input data set and the process can be started again, basically until no samples remain.

The number of iterations which are needed until a "good" hypothesis is found can be determined stochastically. Let the input data set consist of $(r+s)$ elements of which r belong to a model, which shall be identified. If n samples are needed to generate a model instance, the probability that k randomly chosen samples belong to this model is distributed hypergeometrically; that means $P(X = k)$ can be calculated via the formula

RANSAC Iterations

Needed iterations of RANSAC algorithm until a consensus is found with probability 95 % ($p = 0.95$) / 99 % ($p = 0.99$).

Noise Level	Line ($n=2$)	Plane ($n=3$)	Sphere ($n=4$)
10 %	2 / 3	3 / 4	3 / 5
20 %	3 / 5	5 / 7	6 / 9
30 %	5 / 7	8 / 11	11 / 17
40 %	7 / 11	13 / 19	22 / 34
50 %	11 / 16	23 / 35	47 / 72
60 %	18 / 27	46 / 70	116 / 178
70 %	32 / 49	110 / 169	369 / 567
80 %	74 / 113	373 / 574	1871 / 2876
90 %	299 / 459	2995 / 4603	29957 / 46050

Table 2.1: The number of iterations until a RANSAC algorithm detects a consensus with probability p depends mainly on the number of samples n to generate a hypothesis and the ratio of samples belonging to a hypothesis (r) to all samples ($r+s$). The total number of samples plays a minor role. In this table the input data size is considered to be infinite.

$$P(X=k)=\frac{\binom{r}{k}\binom{s}{n-k}}{\binom{r+s}{n}}. \tag{2.98}$$

Therefore, the probability that at least one sample does not belong to this model is $1-P(X=n)$. If the process of model generation and testing is done in j times, the probability that always at least one sample does not belong to the current model is

$$(1-P(X=n))^j \tag{2.99}$$

respectively

$$\left(1-\frac{\binom{r}{n}}{\binom{r+s}{n}}\right)^j. \tag{2.100}$$

If p is the probability that the RANSAC algorithm returns the correct result, the probability of a failure is $1-p$. The probability of such a failure is described by the Term (2.100). Therefore this term has to equal $1-p$. Solving the resulting equation for j returns the expected number of needed iterations:

$$j=\frac{\ln(1-p)}{\ln\left(1-\frac{\binom{r}{n}}{\binom{r+s}{n}}\right)} \tag{2.101}$$

A simplified version of the equation, which does not depend on the number of data samples but on the ratio q of the number of inliers in data to the number of all data samples, leads to the simplified solution

$$j=\frac{\ln(1-p)}{\ln\left(1-(1-q)^n\right)}. \tag{2.102}$$

Table 2.1 lists some examples: let the input data set consist of points in 3D and a RANSAC-based algorithm shall identify a line, a plane, or a sphere.

The number of iterations the algorithm detects a model instance depends on the noise level of the input data and the number of samples n to generate a model.

For example, if 20 % of all points belong to a plane and the remaining points are distributed randomly, then the noise is at a level of 80 %. In this case, the algorithm will need 373 iterations to detect this plane with a probability $p=0.95$; respectively 574 iterations to ensure a probability of $p=0.99$.

Algorithm 2.7 The random sample consensus paradigm (RANSAC).

```
1  RANSAC
2    input   data[1..(r+s)] : samples
3                      n   : generator
4                          size
5                    max   : threshold
6                  limit   : threshold
7    output        model   : instance
8
9    error ← ∞
10   counter ← 0
11   while error > limit
12   and counter < max
13     indices ← CHOOSE (data, n)
14     hypothesis ← GENERATE-MODEL (
       ↪                    indices, data)
15     ratio ← CHECK-MODEL (
       ↪                    hypothesis, data)
16     if ratio < error
17       error ← ratio
18       model ← hypothesis
19     counter ← counter +1
20   return model
```

2.2.5 Continuous Distributions

Important continuous distributions with various fields of application are:

Uniform A uniform distribution has constant probability. The probability density function for a continuous uniform distribution on the interval $[a,b]$ is defined by

$$f(x) = \begin{cases} 0 & x < a \\ \frac{1}{b-a} & a \le x \le b \\ 0 & x > b. \end{cases} \tag{2.103}$$

The short notation for a random variable with uniform distribution, probability density function f and distribution function F is $X \sim U([a,b])$. Its expectation value and its variance are

$$E(X) = \frac{a+b}{2} \tag{2.104}$$

$$V(X) = \frac{(b-a)^2}{12}. \tag{2.105}$$

Exponential The exponential distribution is defined for every positive value $\lambda > 0$ by

$$f(x) = \begin{cases} 0 & x < 0 \\ \lambda e^{-\lambda x} & x \ge 0. \end{cases} \tag{2.106}$$

The expectation value and the variance of a exponential distributed random variable $X \sim Exp(\lambda)$ are

$$E(X) = \frac{1}{\lambda} \tag{2.107}$$

$$V(X) = \frac{1}{\lambda^2}. \tag{2.108}$$

Normal A random variable is normal distributed with with mean μ and variance σ^2 (in short: $X \sim N(\mu,\sigma^2)$), if it has a probability density function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-(x-\mu)^2/(2\cdot\sigma^2)}, \tag{2.109}$$

respectively

$$f(x) = \frac{1}{\sigma} \cdot \varphi\left(\frac{x-\mu}{\sigma}\right), \qquad x \in \mathbb{R} \tag{2.110}$$

with the bell-shaped, Gauss error distribution curve

$$\varphi(t) = \frac{e^{-t^2/2}}{\sqrt{2\pi}}. \tag{2.111}$$

The standard normal distribution is given by taking $\mu = 0$ and $\sigma^2 = 1$. The probability density function of the standard normal distribution is plotted in Figure 2.9. Its cumulative distribution function is $\Phi(x) = \int_{-\infty}^{x} \varphi(t)\,\mathrm{d}\,t$. It is evaluated in Table 2.2.

Normal Distribution

Table of the integral $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, \mathrm{d}\, t$

x	$\Phi(x)$	x	$\Phi(x)$	x	$\Phi(x)$
0.000000	0.500000	1.000000	0.841345	2.000000	0.977250
0.050000	0.519939	1.050000	0.853141	2.050000	0.979818
0.100000	0.539828	1.100000	0.864334	2.100000	0.982136
0.150000	0.559618	1.150000	0.874928	2.150000	0.984222
0.200000	0.579260	1.200000	0.884930	2.200000	0.986097
0.250000	0.598706	1.250000	0.894350	2.250000	0.987776
0.300000	0.617911	1.300000	0.903200	2.300000	0.989276
0.350000	0.636831	1.350000	0.911492	2.350000	0.990613
0.400000	0.655422	1.400000	0.919243	2.400000	0.991802
0.450000	0.673645	1.450000	0.926471	2.450000	0.992857
0.500000	0.691462	1.500000	0.933193	2.500000	0.993790
0.550000	0.708840	1.550000	0.939429	2.550000	0.994614
0.600000	0.725747	1.600000	0.945201	2.600000	0.995339
0.650000	0.742154	1.650000	0.950529	2.650000	0.995975
0.700000	0.758036	1.700000	0.955435	2.700000	0.996533
0.750000	0.773373	1.750000	0.959941	2.750000	0.997020
0.800000	0.788145	1.800000	0.964070	2.800000	0.997445
0.850000	0.802337	1.850000	0.967843	2.850000	0.997814
0.900000	0.815940	1.900000	0.971283	2.900000	0.998134
0.950000	0.828944	1.950000	0.974412	2.950000	0.998411
1.000000	0.841345	2.000000	0.977250	3.000000	0.998650

Table 2.2: The cumulative distribution function $\Phi(x)$ of the standard normal distribution $N(0,1)$ is tabulated for semi-positive values $x \geq 0$. Due to its point symmetry at $(0|1/2)$ (see Figure 2.9) negative values can be calculated using the symmetry condition $\Phi(x) - 1/2 = -\Phi(-x) + 1/2$.

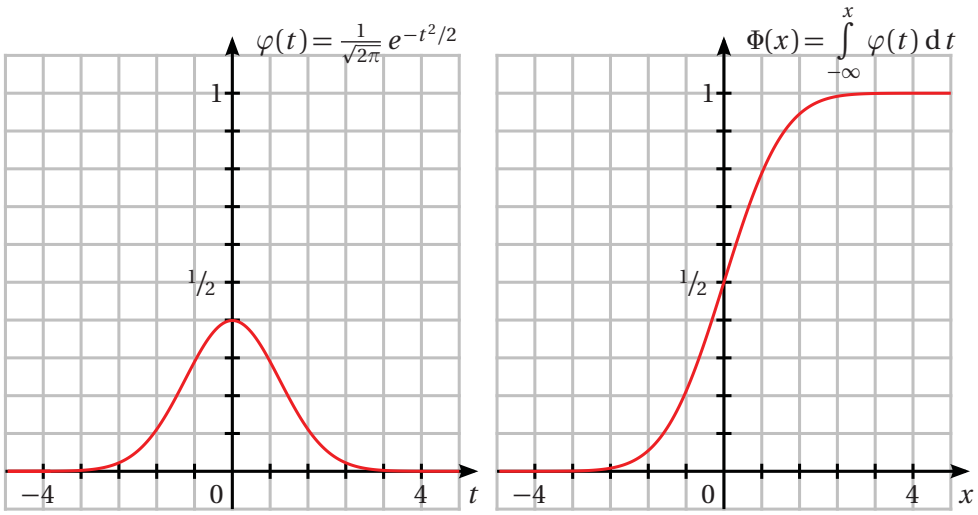


Figure 2.9: The probability density function of the standard normal distribution is the Gauss error distribution curve $\varphi(t) = \frac{1}{\sqrt{2\pi}}\, e^{-t^2/2}$. Its function plot (left) shows its characteristic bell-shaped run. Its cumulative distribution function $\Phi(x) = \int_{-\infty}^{x} \varphi(t)\,\mathrm{d}\,t$ (right) is used in various contexts, e.g. in the next Section.

2.2.6 Inequalities and Limits

The theory of probability and statistics contains many theorems which allow approximating various probability terms. According to the Tschebyshev[4] inequality for any random variable X with existing expectation value and variance, the equation

$$P(|X - EX| \geq \varepsilon) \leq \frac{1}{\varepsilon^2} \cdot V(X), \qquad \varepsilon > 0 \tag{2.112}$$

is satisfied.

The central limit theorem by Jarl Waldemar Lindeberg[5] and Paul Lévy[6] states that the sum of independent random variables will approach a normal distribution regardless of the distribution of the individual variables themselves. More precisely,

[4] Pafnuty Lvovich Tschebyshev (May 16, 1821 – December 8, 1894) Pafnuty Lvovich Tschebyshev (alternative spelling: "Chebyshev") was a Russian mathematician. He worked on number theory, complex analysis, and probabilistic theory.

[5] Jarl Waldemar Lindeberg (August 4, 1876 – December 12, 1932) Jarl Waldemar Lindeberg was a Finnish mathematician. He worked on partial differential equations, calculus of variations, probability theory and statistics. He is known for his work on the central limit theorem.

[6] Paul Pierre Lévy (September 15, 1886 – December 15, 1971) Paul Pierre Lévy was a French mathematician who became famous for his work on probability theory.

if $(X_n)_{n\geq 1}$ is a sequence of independent and identically distributed random variables with positive, finite variance $\sigma^2 = V(X_1)$ and expectation value $\mu = EX_1$, the limit

$$\lim_{n\to\infty} P\left(a \leq \frac{\sum_{i=1}^{n} X_i - n\cdot\mu}{\sigma\cdot\sqrt{n}} \leq b\right) = \Phi(b) - \Phi(a), \quad (-\infty \leq a < b < \infty) \tag{2.113}$$

converges to the differences $\Phi(b)-\Phi(a)$ of the cumulative Gaussian distribution function $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}\,\mathrm{d}\,t$. According to HANS BANDEMER and ANDREAS BELLMANN this approximation by the standard normal distribution is practicable for sequences with $n > 30$ [BB79].

A special case of the central limit theorem is the de Moivre-Laplace theorem by ABRAHAM DE MOIVRE[7] and PIERRE-SIMON LAPLACE.[8] If each random variables X_i describe a Bernoulli experiment (a random experiment whose outcome can either be "positive" with probability p or "negative" with probability $1-p$, $p \in (0,1)$), the central limit theorem gives a normal approximation to the binomial distribution: Let X_n be a random variable with binomial distribution with parameters $n \in \mathbb{N}$ and p, $0 < p < 1$. Then the central limit theorem can be rephrased to

$$\lim_{n\to\infty} P\left(a \leq \frac{X_n - np}{\sqrt{np(1-p)}} \leq b\right) = \Phi(b) - \Phi(a) \tag{2.114}$$

and

$$\lim_{n\to\infty} P\left(\frac{X_n - np}{\sqrt{np(1-p)}} \leq b\right) = \Phi(b), \tag{2.115}$$

with $a < b$, $a,b \in \mathbb{R}$.

The same way the central limit theorem can be used to approximate a random process by a normal distribution, the de Moivre-Laplace theorem can be utilized to approximate the binomial distribution. For large values of n and $k,l \in \mathbb{N}, 0 \leq k < l \leq n$ the probability

$$P(k \leq X_n \leq l) = \sum_{j=k}^{l} \binom{n}{j} p^j (1-p)^{n-j} \tag{2.116}$$

of a binomial distributed random variable X_n can be approximated by

$$P(k \leq X_n \leq l) = P\left(\frac{k-np}{\sqrt{np(1-p)}} \leq \frac{X_n - np}{np-(1-p)} \leq \frac{l-np}{np(1-p)}\right) \approx \Phi(b) - \Phi(a) \tag{2.117}$$

[7] ABRAHAM DE MOIVRE (May 26, 1667 – November 27, 1754) Abraham de Moivre was a French mathematician who pioneered the development of analytic geometry and the theory of probability.

[8] PIERRE-SIMON LAPLACE (March 28, 1749 – March 5, 1827) Pierre-Simon Marquis de Laplace was a French mathematician and astronomer. He worked on probability theory and differential equations.

with

$$a = \frac{k - np}{\sqrt{np(1-p)}}, \qquad b = \frac{l - np}{\sqrt{np(1-p)}}. \tag{2.118}$$

In many cases a better approximation can be achieved with a so-called continuity correction ($\pm 1/2$):

$$P(k \leq X_n \leq l) \approx \Phi\left(\frac{l - np + \frac{1}{2}}{\sqrt{np(1-p)}}\right) - \Phi\left(\frac{k - np - \frac{1}{2}}{\sqrt{np(1-p)}}\right). \tag{2.119}$$

The special case of a hypergeometric distribution $X_{n,r,s}$ with $\mu(n,r,s) = n \cdot \frac{r}{r+s}$ and $\sigma^2(n,r,s) = n \cdot \frac{r}{r+s} \cdot \frac{s}{r+s} \cdot \left(1 - \frac{n-1}{r+s-1}\right)$, $a < b$, $a,b \in \mathbb{R}$ reformulates to

$$\lim_{\sigma^2(n,r,s)\to\infty} P\left(a \leq \frac{X_{n,r,s} - \mu(n,r,s)}{\sqrt{\sigma^2(n,r,s)}} \leq b\right) = \Phi(b) - \Phi(a) \tag{2.120}$$

as well as

$$\lim_{\sigma^2(n,r,s)\to\infty} P\left(\frac{X_{n,r,s} - \mu(n,r,s)}{\sqrt{\sigma^2(n,r,s)}} \leq b\right) = \Phi(b). \tag{2.121}$$

2.2.7 Statistical Estimation

According to Eric Weisstein an estimate is an educated guess for an unknown quantity or outcome based on known information. The making of estimates is an important part of statistics, since care is needed to provide as accurate an estimate as possible using as little input data as possible. Often, an estimate for the uncertainty of an estimate can also be determined statistically. A rule that tells how to calculate an estimate based on the measurements contained in a sample is called an estimator [Wei09].

More formal, the initial situation consists of a probability space $(\Omega, \mathcal{F}, P)$, a random variable X and a realization (an observed value) x of X; e.g. $x = X(\omega)$ for an element $\omega \in \Omega$. The distribution is not known completely. The class of distribution is assumed and its free parameters, which shall be estimated, are denoted by ϑ. To indicate the dependency the distribution, its expectation value, its variance, etc. are written P_ϑ, E_ϑ, and V_ϑ. The set of all possible parameters ϑ is the parameter space Θ.

For some data x the probability $P_\vartheta(X = x)$ can be interpreted as function of ϑ. This function

$$L_x : \begin{cases} \Theta & \to & [0,1] \\ \vartheta & \mapsto & P_\vartheta(X = x) \end{cases} \tag{2.122}$$

maps each parameter ϑ to the probability to obtain the observed data x. L_x is called likelihood function. If L_x reaches a maximum value $\hat{\vartheta}$ for each x, the function $\hat{\vartheta}(x)$ is called maximum likelihood estimator.

A maximum likelihood estimator is a so-called point estimator. Its estimate is a single point / a single datum. As the estimated value $T(x)$ of an estimator T and the unknown parameter ϑ may differ significantly, confidence regions have been introduced.

A confidence region $C(x)$ for ϑ is a subset of all possible parameters Θ. Is $C(x)$ an interval in $\Theta \subset \mathbb{R}$ with endpoints $l(x)$ and $L(x)$, it is called confidence interval. Confidence intervals are a form of interval estimation. In contrast to point estimation it indicates the precision with which the parameter ϑ is estimated.

A confidence interval $C(x) = [l(x), L(x)]$ for ϑ is said to have confidence level $1-\alpha$, $0 < \alpha < 1$, if

$$P_\vartheta(\{x \in \Omega : \vartheta \in C(x)\}) \geq 1-\alpha, \qquad \forall\, \vartheta \in \Theta\ . \tag{2.123}$$

For random variables with binomial or hypergeometric distribution point estimators and confidence intervals can be derived explicitly. NORBERT HENZE [Hen95] derived them as follows:

For stochastically independent and each binomially distributed random variables $X_1, \ldots, X_n$ with parameters $1, \vartheta$, $\vartheta \in \Theta = (0,1)$, a point estimator is

$$T_n = \frac{1}{n} \sum_{j=1}^{n} X_j. \tag{2.124}$$

According to the theorem of de Moivre-Laplace (2.2.6)

$$\lim_{n\to\infty} P_\vartheta \left(\left| \frac{\sqrt{n}(T_n - \vartheta)}{\sqrt{\vartheta(1-\vartheta)}} \right| \leq h \right) = \Phi(h) - \Phi(-h), \qquad h > 0 \tag{2.125}$$

the central limit theorem gives a normal approximation to the binomial distribution. As $\sqrt{n}|T_n - \vartheta| \leq h\sqrt{\vartheta(1-\vartheta)}$ leads to a quadratic inequality in ϑ

$$(n+h^2) \cdot \vartheta^2 - (2nT_n + h^2) \cdot \vartheta + nT_n^2 \leq 0, \tag{2.126}$$

the endpoints of the interval are

$$l_n(X_1, \ldots, X_n) \leq \theta \leq L_n(X_1, \ldots, X_n) \tag{2.127}$$

with

$$l_n(X_1, \ldots, X_n) = \frac{T_n + \frac{h^2}{2n} - \frac{h}{\sqrt{n}} \cdot \sqrt{T_n(1-T_n) + \frac{h^2}{4n}}}{1 + \frac{h^2}{n}}, \tag{2.128}$$

$$L_n(X_1, \ldots, X_n) = \frac{T_n + \frac{h^2}{2n} + \frac{h}{\sqrt{n}} \cdot \sqrt{T_n(1-T_n) + \frac{h^2}{4n}}}{1 + \frac{h^2}{n}}, \tag{2.129}$$

and

$$h(\alpha) = \Phi^{-1}\left(1 - \frac{\alpha}{2}\right) \tag{2.130}$$

using $\Phi(h) - \Phi(-h) = 2\Phi(h) - 1$.

As the limit

$$\lim_{n\to\infty} P_\vartheta\left(l_n(X_1,\ldots,X_n) \le \vartheta \le L_n(X_1,\ldots,X_n)\right) = 1-\alpha. \tag{2.131}$$

converges to $1-\alpha$ the sequence

$$C_n(x_1,\ldots,x_n) = [l_n(x_1,\ldots,x_n), L_n(x_1,\ldots,x_n)] \tag{2.132}$$

is an asymptotic confidence interval for ϑ at confidence level $(1-\alpha)$. Table 2.2 lists some values of Φ. Corresponding inverse values can be used to calculate the values of $h(\alpha) = \Phi^{-1}(1-\alpha/2)$. A selection of values of the inverse function Φ^{-1} is listed in Table 2.3.

Inverse Cumulative Density Function Φ^{-1}

Table of the inverse cumulative density function Φ^{-1} of the function $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\,\mathrm{d}\,t$

α	$\Phi^{-1}(\alpha)$	α	$\Phi^{-1}(\alpha)$
0.500000	0.000000	$1-1/10^2$	2.326348
0.550000	0.125661	$1-1/10^3$	3.090232
0.600000	0.253347	$1-1/10^4$	3.719016
0.650000	0.385320	$1-1/10^5$	4.264891
0.700000	0.524401	$1-1/10^6$	4.753424
0.750000	0.674490	$1-1/10^7$	5.199338
0.800000	0.841621	$1-1/10^8$	5.612001
0.850000	1.036433	$1-1/10^9$	5.997807
0.900000	1.281552	$1-1/10^{10}$	6.361341
0.950000	1.644854	$1-1/10^{11}$	6.706023

Table 2.3: Many estimations and approximations depend on values of the inverse function of the cumulative density function of the standard normal distribution $N(0,1)$. This Table lists commonly used/needed values.

In case of a hypergeometric distribution confidence intervals can be determined the same way. The basic population Ω may consist of N elements of which r are marked. The quotient $\vartheta = r/N$ is unknown and shall be estimated. The relative frequency of marked elements in a sample of size n will be denoted T_n. It is a point estimator for ϑ.

The central limit theorem for hypergeometric distributions provides a normal approximation for large values of n:

$$P_\vartheta\left(\frac{\sqrt{n}\cdot|T_n-\vartheta|}{\vartheta\cdot(1-\vartheta)\cdot\left(1-\frac{n-1}{N-1}\right)}\le h\right)\approx\Phi(h)-\Phi(-h) \tag{2.133}$$

with

$$h=\Phi^{-1}\left(1-\frac{\alpha}{2}\right). \tag{2.134}$$

Similar to the binomial distribution the interval endpoints are the solution of a quadratic inequality. The lower, respectively upper bound is

$$\frac{T_n+\frac{h^2}{2n}\gamma\pm\frac{h}{\sqrt{n}}\cdot\sqrt{T_n(1-T_n)\gamma+\frac{h^2}{4n}\gamma^2}}{1+\frac{h^2}{n}\gamma}. \tag{2.135}$$

The resulting confidence interval for ϑ has confidence level $1-\alpha$, $0<\alpha<1$. The term $\gamma=1-\frac{n-1}{N-1}$ is the finite population correction factor. The quotient $n/N(\approx 1-\gamma)$ is called sampling fraction.

Please note, the precision of the confidence – respectively the length of the confidence interval – mainly depends on the sample size n and not on the sampling fraction n/N.

Linear Algorithms in Sublinear Time

Many algorithms contain linear subalgorithms, which work on a large input data set without modifying it, and which calculate a "small" result; i.e. a result whose size does not depend on the input data set and which is constant therefore [UF11b]. A typical example is a for-loop which iterates on all input samples (e.g. in the RANSAC Algorithm 2.7) and performs a calculation on each input sample independently (e.g. check a hypothesis). In such a situation a sufficient result can be obtained by an estimator in sublinear – or even constant – time.

The RANSAC algorithm shall exemplify the usage of an estimator. In this setting the input data set consists of N elements which "vote" whether they agree to an hypothesis or not. Calling an election needs $O(N)$ time; a forecast can be done much faster [HN02]: an asymptotic estimator may use the central limit theorems to construct a confidence interval (see Section 2.2.7). Omitting all negligible terms of order $O(1/n)$ leads to the interval endpoints $T_n \pm \frac{h}{\sqrt{n}} \cdot \sqrt{T_n(1-T_n)}$; respectively to the interval length

$$\Delta = \frac{2h}{\sqrt{n}} \cdot \sqrt{T_n(1-T_n)}. \quad (2.136)$$

The term of the unknown estimate T_n is limited $\sqrt{T_n(1-T_n)} \leq \frac{1}{2}$ and the constant $h = \Phi^{-1}(1-\frac{\alpha}{2})$ does only depend on the confidence level. Having set the confidence level $1-\alpha$ (see Table 2.3 for reasonable values) and the accuracy Δ, the number of needed samples to perform a forecast is

$$n = \frac{\Phi^{-1}\left(1-\frac{\alpha}{2}\right)}{\Delta^2}. \quad (2.137)$$

An example illustrates the efficiency. The data set in Figure 2.10 consists of 19 400 points, of which 5 000 belong to **one** plane. The remaining 14 400 points are distributed randomly in a cube. With a noise level of almost 75 % the expected number of iterations to detect a plane with 95 % probability is 174. This results in 3 375 600 Point-In-Plane tests.

As an estimator introduces an additional error, its confidence level has been set to 97.5 % and the RANSAC algorithm shall detect a plane with a probability of as well 97.5 %. Then the overall system is able to determine the correct result – due to stochastic independence – with a probability of $0.975 \cdot 0.975 = 95.0626$ %. In this situation the expected number of RANSAC iterations is 213 and the number of samples to take is 139. Therefore, the estimator reduces the number of Point-In-Plane tests to 29 607 single tests. A test implementation using the example data set confirms this speed-up.

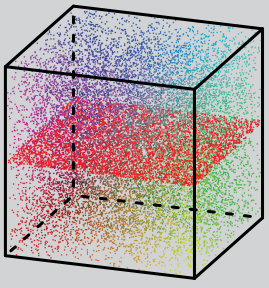


Figure 2.10: This example data set consists of 5 000 points (red) in a plane and 14 400 points randomly distributed in a cube. To determine the plane's parameters is a typical task for a RANSAC algorithm.

2.3 Numerical Optimization

An optimization problem can be represented in the following way: For a function f from a set A to the real numbers, an element $\mathbf{x}_0 \in A$ is sought-after, such that

$$\forall \mathbf{x} \in A \ : \ f(\mathbf{x}_0) \leq f(\mathbf{x}). \tag{2.138}$$

Such a formulation is called an minimization problem and the element $\mathbf{x}_0$ is a global minimum. Without loss of generality, it is sufficient to investigate minimization problems; maximization problems can be transformed to minimization problems via duality.

The maximization of a real-valued function $g(\mathbf{x})$ can be regarded as the minimization of the transformed function

$$f(\mathbf{x}) = (-1) \cdot g(\mathbf{x}). \tag{2.139}$$

Depending on the field of application, f is called an objective function, cost function, energy function, or energy functional. A feasible solution that minimizes the objective function is called an optimal solution.

Typically, A is some subset of the Euclidean space $\mathbb{R}^n$, often specified by a set of constraints (equalities or inequalities) that the members of A have to satisfy. The domain A of f is often called search space or choice set, while the elements of A are called candidate solutions or feasible solutions.

Generally, a function f may have several local minima, where a local minimum $\mathbf{x}^\star$ satisfies the expression $f(\mathbf{x}^\star) \leq f(\mathbf{x})$ for all $\mathbf{x} \in A$ in a neighborhood of $\mathbf{x}^\star$:

$$||\mathbf{x} - \mathbf{x}^\star|| \leq \delta, \qquad \delta > 0. \tag{2.140}$$

In other words, on some region around $\mathbf{x}^\star$ all function values are greater than or equal to the value at $\mathbf{x}^\star$. The occurrence of multiple extrema makes problem solving in (nonlinear) optimization very hard. The global (best) minimizer is difficult to obtain without supplying global information, which in turn is usually unavailable for a nontrivial case. Since there is no easy algebraic characterization of global optimality, global optimization is a difficult area, at least in higher dimensions.

As global optimization is not within the scope of this thesis, only a few representative techniques are summarized. Further explanations can be found in “Numerical Methods” [BP93], “Numerical Optimization” [NW99], “Introduction to Applied Optimization” [Diw03], “Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation” [Nas90], as well as in “Numerische Methoden der Analysis” (english: Numerical Methods of Analysis) [HP10]. Besides these introductions and overviews some books emphasize practical aspects – e.g. “Practical Optimization” [GMW82], “Practical Methods of Optimization” [Fle00], and “Global Optimization: Software, Test Problems, and Applications” [Pin02].

Global optimization is a fast evolving research area. Current overviews on the latest research results have been published in "Large-Scale Nonlinear Constrained Optimization: A Current Survey Algorithms for continuous optimization: the state of the art" [CGT94], "Numerical methods for large-scale nonlinear optimization" [GOT05], and in "Robust optimization – A comprehensive survey" [BS07]. The following algorithms and techniques do not give a complete overview. They just explain the main principles of global optimization.

2.3.1 Quadratic Search

The main idea behind quadratic search is to approximate an objective function f locally by a quadratic function, whose minimum can be calculated easily. In a one-dimensional case, three vales x_i, x_{i-1}, x_{i-2} are used to define a parabola $p(x) = ax^2 + bx + c$ with coefficients

$$\begin{aligned} a = \frac{1}{X} \cdot \Big(& (f(x_{i-2}) - f(x_{i-1})) \cdot x_i \\ & + (f(x_i) - f(x_{i-2})) \cdot x_{i-1} \\ & + (f(x_{i-1}) - f(x_i)) \cdot x_{i-2} \Big), \end{aligned} \tag{2.141}$$

$$\begin{aligned} b = \frac{-1}{X} \cdot \Big(& (f(x_{i-2}) - f(x_{i-1})) \cdot x_i^2 \\ & + (f(x_i) - f(x_{i-2})) \cdot x_{i-1}^2 \\ & + (f(x_{i-1}) - f(x_i)) \cdot x_{i-2}^2 \Big), \end{aligned} \tag{2.142}$$

$$\begin{aligned} c = \frac{1}{X} \cdot \Big(& (f(x_{i-2}) \cdot x_{i-1} - f(x_{i-1}) \cdot x_{i-2}) \cdot x_i^2 \\ & + (f(x_i) \cdot x_{i-2} - f(x_{i-2}) \cdot x_i) \cdot x_{i-1}^2 \\ & + (f(x_{i-1}) \cdot x_i - f(x_i) \cdot x_{i-1}) \cdot x_{i-2}^2 \Big), \end{aligned} \tag{2.143}$$

with $X = (x_i - x_{i-1}) \cdot (x_{i-1} - x_{i-2}) \cdot (x_i - x_{i-2})$. In an iteration the parabola's minimum at $-b/2a$ (if it exists, resp. if $a > 0$) is used to calculate the next iteration step x_{i+1}:

$$x_{i+1} = \frac{(f(x_{i-2}) - f(x_{i-1})) \cdot x_i^2 + (f(x_i) - f(x_{i-2})) \cdot x_{i-1}^2 + (f(x_{i-1}) - f(x_i)) \cdot x_{i-2}^2}{2 \cdot ((f(x_{i-2}) - f(x_{i-1})) \cdot x_i + (f(x_i) - f(x_{i-2})) \cdot x_{i-1} + (f(x_{i-1}) - f(x_i)) \cdot x_{i-2})} \tag{2.144}$$

In higher dimensions this approach is of limited use due to the complexity of its inherent, inverse approximation problem. In this case, the method of the steepest gradient can be utilized.

2.3.2 Gradient Descent

The steepest gradient method can minimize multivariate functions $f(\mathbf{x})$,$\mathbf{x} \in \mathbb{R}^n$. This iterative algorithm determines the negative gradient

$$\mathbf{d}_i = -\text{grad} f(\mathbf{x}_i) \tag{2.145}$$

and minimizes the one-dimensional optimization problem

$$\min_{t \geq 0} f(\mathbf{x}_i + t \cdot \mathbf{d}_i). \tag{2.146}$$

The minimum at t_i sets the next iteration step

$$x_{i+1} = \mathbf{x}_i + t_i \cdot \mathbf{d}_i. \tag{2.147}$$

The sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ converges, if f is bounded below and if its gradient grad$f$ meets the Lipschitz[9] condition

$$||\text{grad} f(\mathbf{x}) - \text{grad} f(\tilde{\mathbf{x}})|| \leq L||\mathbf{x} - \tilde{\mathbf{x}}||, \qquad L \in \mathbb{R} \tag{2.148}$$

for $\mathbf{x}$ and $\tilde{\mathbf{x}}$ in a neighborhood of

$$\left\{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}_0)\right\}. \tag{2.149}$$

In most cases the limit point of the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ is a local minimum. The inner, one-dimensional optimization is typically done using an approximation technique.

Interestingly, the convergence of the steepest gradient method can be shown with very few requirements [SHS02]. The iteration

$$\mathbf{x}_{i+1} = \mathbf{x}_i + t_i \cdot \mathbf{d}_i \tag{2.150}$$

has to satisfy three conditions with $\alpha, \beta, \gamma \in (0,1)$ and $\beta < \gamma$:

1. The search direction $\mathbf{d}_i$ does not have to be the negative gradient. The angle may vary up to $\pi/2$:

$$-\mathbf{d}_i^{\mathrm{T}} \text{grad} f(\mathbf{x}_i) \geq \alpha ||\mathbf{d}_i|| ||\text{grad} f(\mathbf{x}_i)||. \tag{2.151}$$

   This property reduces the objective function locally.

2. The objective function has to be bounded by a linear function:

$$f(\mathbf{x}_{i+1}) \leq f(\mathbf{x}_i) + \beta\, t\, \mathbf{d}_i^{\mathrm{T}} \text{grad} f(\mathbf{x}_i). \tag{2.152}$$

3. Finally, the step size may not be too small:

$$\mathbf{d}_i^{\mathrm{T}} \text{grad} f(\mathbf{x}_{i+1}) \geq \gamma \mathbf{d}_i^{\mathrm{T}} f(\mathbf{x}_i). \tag{2.153}$$

This method has a severe drawback. It requires many iterations, if the function – interpreted as a heightfield – has long, narrow valley structures. In such a case, a conjugate gradient method is preferable.

[9] Rudolf Otto Sigismund Lipschitz (May 14, 1832 – October 7, 1903) Rudolf Otto Sigismund Lipschitz was a German mathematician who worked in widely different fields on which he contributed. His work in algebraic number theory led him to study the quaternions and generalizations such as Clifford algebras. In fact Lipschitz rediscovered Clifford algebras and was the first to apply them to represent rotations of Euclidean spaces.

Automatic Differentiation

In many applications of scientific computing, it is necessary to compute derivatives of functions. According to ROLF HAMMER, MATTHIAS HOCKS, ULRICH KULISCH, and DIETMAR RATZ [HHKR97] there are three different methods to get the values of the derivatives: numerical differentiation, symbolic differentiation, and automatic differentiation.

Numerical differentiation uses difference approximations to compute approximations of the derivative values. Symbolic differentiation computes explicit formulas for the derivative functions by applying differentiation rules. Automatic differentiation also uses the well-known differentiation rules, but it propagates numerical values for the derivatives. This combines the advantages of symbolic and numerical differentiation [GW08]. Numbers instead of symbolic formulas must be handled, and the computation of the derivative values is done automatically together with the computation of the function value.

Automatic differentiation evaluates functions specified by algorithms or formulas where all operations are performed according to the rules of a differentiation arithmetic given by "C++ for Verified Computing" [HHKR97]. In the one-dimensional case, first order differentiation arithmetic is an arithmetic for ordered pairs: the first component contains the value $u(x)$ of the function $u : \mathbb{R} \to \mathbb{R}$ at the point $x \in \mathbb{R}$. The second component contains the value of the derivative $u'(x)$. The rules for the arithmetic are listed in Table 2.4. The familiar rules of calculus are used in the second component. The operations in these definitions are operations on real numbers.

An independent variable x and the arbitrary constant c correspond to the ordered pairs

$$(x,1) \text{ and } (c,0), \tag{2.154}$$

Automatic Differentiation

Differentiation arithmetic for ordered pairs.

Differentiation Arithmetic				Real Arithmetic		
(u,u')	$+$	(v,v')	$=$	$(u+v,$	$u'+v')$	
(u,u')	$-$	(v,v')	$=$	$(u-v,$	$u'-v')$	
(u,u')	$\cdot$	(v,v')	$=$	$(u\cdot v,$	$u\cdot v'+u'\cdot v)$	
(u,u')	$/$	(v,v')	$=$	$(u/v,$	$(u'\cdot v-u\cdot v')/v^2),$	$v\neq 0$

Table 2.4: Differentiation arithmetics uses the well-known rules for derivative values. In contrast to symbolic differentiation only the algorithm or formula for the function is needed. No explicit formulas for the derivatives are required.

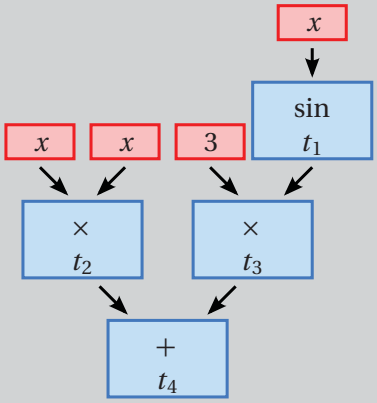


Figure 2.11: The evaluation of the term $x^2+3\sin x$ at $x_0 = 1.3$ using differentiation arithmetic (see Table 2.4) does not only return its value but also its derivative value. The computational complexity of this differentiation arithmetic (forward method) is at most a small multiple of the cost of evaluating the term itself.

since $\frac{dx}{dx} = 1$, and $\frac{dc}{dx} = 0$. If the independent variable x of a formula for a function $f : \mathbb{R} \to \mathbb{R}$ is replaced by $X = (x,1)$, and if all constants are replaced by their $(c,0)$ representation, then the evaluation of f using the rules of differentiation arithmetic gives the ordered pair

$$f(X) = f((x,1)) \tag{2.155}$$
$$= (f(x), f'(x)). \tag{2.156}$$

For example, Figure 2.11 shows an abstract syntax tree. Its evaluation at $x_0 = 1.3$ illustrates the calculation of its derivative values at intermediate subterms. For elementary functions

$$s : \mathbb{R} \to \mathbb{R} \tag{2.157}$$

the rules of differentiation arithmetic must be extended using the chain rule

$$s(U) = s((u,u')) \tag{2.158}$$
$$= (s(u), u' \cdot s'(u)). \tag{2.159}$$

This way the sine function is defined by

$$\sin U = \sin(u,u') \tag{2.160}$$
$$= (\sin u, u' \cdot \cos u). \tag{2.161}$$

The result of this structure and its corresponding operators is the algebra of dual numbers [Kel00], which can be implemented in three ways:

Many programming languages offer an overloading mechanism that replaces each real number by a pair of real numbers including the differential. Each elementary operation on real numbers is overloaded, i.e. internally replaced by a new one, working on pairs of reals, that computes the value and its differential. In this way the original program is virtually unchanged.

Another approach uses source code transformation. This technique adds new variables, arrays, and data structures into the program that will hold the derivatives and the new instructions that compute them. This approach does not depend on language features such as operator overloading.

The third way to implement automatic differentiation does not modify a program or its source, but the platform (e.g. Java Virtual Machine, .Net Common Language Runtime, etc.) it runs on.

Recent developments in the field of automatic differentiation are summarized in "Advances in Automatic Differentiation" [BBH+08].

2.3.3 Conjugated Gradients

The conjugated gradients method determines the minimum of a quadratic, convex function, but ROGER FLETCHER and COLIN M. REVES generalized the algorithm to nonlinear optimization. It is used to find the local minimum of a nonlinear function using its gradient and works, when the function is approximately quadratic near the minimum, which is the case when the function is twice differentiable at the minimum.

Starting with

$$\mathbf{x}_0, \tag{2.162}$$
$$\mathbf{g}_0 = \operatorname{grad} f(\mathbf{x}_0), \tag{2.163}$$
$$\mathbf{d}_0 = -\mathbf{g}_0 \tag{2.164}$$

the iteration of conjugated gradients generates a sequence of approximations for a local minimum via the recursion:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{d}_i, \tag{2.165}$$
$$\mathbf{g}_{i+1} = \operatorname{grad} f(\mathbf{x}_{i+1}), \tag{2.166}$$
$$\mathbf{d}_{i+1} = -\mathbf{g}_{i+1} + \beta_i \mathbf{d}_i. \tag{2.167}$$

In this recursion α_i minimizes the one-dimensional optimization problem [GJH95]

$$\min_{\alpha_i > 0} f(\mathbf{x}_i + \alpha_i \mathbf{d}_i) \tag{2.168}$$

and β_i is defined by

$$\beta_i = \frac{\mathbf{g}_{i+1}^{\mathrm{T}} \mathbf{g}_{i+1}}{\mathbf{g}_i^{\mathrm{T}} \mathbf{g}_i}. \tag{2.169}$$

The parameters α_i and β_i can be set in various ways, e.g.

$$\beta_i = \frac{(\mathbf{g}_{i+1}^{\mathrm{T}} - \mathbf{g}_i^{\mathrm{T}}) \mathbf{g}_{i+1}}{\mathbf{g}_i^{\mathrm{T}} \mathbf{g}_i} \tag{2.170}$$

as suggested in the article “Note sur la convergence de méthodes de directions conjuguées” [PR69] or

$$\beta_i = \frac{(\mathbf{g}_{i+1}^{\mathrm{T}} - \mathbf{g}_i^{\mathrm{T}}) \mathbf{g}_{i+1}}{(\mathbf{g}_{i+1}^{\mathrm{T}} - \mathbf{g}_i^{\mathrm{T}}) \mathbf{d}_i} \tag{2.171}$$

as proposed in “Methods of conjugate gradients for solving linear systems” [HS52].

2.3.4 Genetic Algorithms

As most gradient based methods optimize locally, they most likely find local minima of a nonlinear function but not its global minimum. To find a global minimum local optimization algorithms can be combined with genetic algorithms [JM90], [Mic95], [MS96] which have good global search characteristics [ONOT98]. These combinatorial methods – such as simulated annealing [Ing93] – improve the search strategy through the introduction of two tricks. The first is the so-called "Metropolis algorithm" [MRRT53], in which some iterations that do not lower the objective function are accepted in order to "explore" more of the possible space of solutions. The second trick limits these explorations, if the cost function declines only slowly.

2.3.5 Differential Evolution

The differential evolution method is described in "Differential Evolution: A simple and efficient heuristic for global optimization over continuous spaces" [SP97]. It is a parallel, direct search method based on ideas of evolution strategies. It uses n vectors $\mathbf{x}_{i,k}$ $(i = 1,\ldots,n)$ as a population for each generation k.

The members of a new generation $k+1$ are generated by a permutation and a cross-over process. For each vector $x_{i,k}$ a trial vector

$$\Delta = \mathbf{x}_{\alpha,k} + F \cdot (\mathbf{x}_{\beta,k} - \mathbf{x}_{\gamma,k}) \tag{2.172}$$

with a constant $F \in \mathbb{R}$, $F > 0$, and randomly chosen $\alpha,\beta,\gamma \in \{1,\ldots,n\}\backslash\{i\}$, which do not equal each other $\alpha \neq \beta \neq \gamma \neq \alpha$.

To increase the diversity of the parameter vectors, a cross-over process mixes $\mathbf{x}_{i,k}$ with Δ. The resulting vector $\mathbf{y}$ consists of a sequence of elements of $\mathbf{x}_{i,k}$, whereas other elements are copied from Δ.

If the new vector $\mathbf{y}$ yields a smaller objective function value and its error value of the minimization process is smaller than $\mathbf{x}_{i,k}$, then $\mathbf{x}_{i,k+1}$ is replaced by $\mathbf{y}$ otherwise the old value $\mathbf{x}_{i,k}$ is retained.

The differential evolution method for minimizing continuous space functions can also handle discrete problems by embedding the discrete parameter domain in a continuous domain. It converges quite fast and is inherently parallel, which allows an execution on a network of computers.

3 Geometry

Geometry is one of the oldest sciences [SS04]. Some of its main concepts – namely basic topology, affine geometry, projective geometry, and differential geometry – are introduced and recapitulated in this chapter.

The notations, definitions and theorems are excerpted from JOSEF HOSCHEK's and DIETER LASSER's "Grundlagen der Geometrischen Datenverarbeitung" (english: Fundamentals of Computer Aided Geometric Design) [HL89], from "Geometric Concepts for Geometric Design" by WOLFGANG BOEHM and HARTMUT PRAUTZSCH [BP94], and from the textbook on computer-aided geometry by GÜNTER AUMANN and KLAUS SPITZMÜLLER [AS93].

The section on differential geometry is based on the textbooks "Modern Differential Geometry of Curves and Surfaces with Mathematica" by ALFRED GRAY [Gra97] and MANFREDO DO CARMO's "Differential Geometry of Curves and Surfaces" [DC76].

Contents

3.1 Topology

The theory of topology is based on set theory and concerns itself with structures of sets. It generalizes many distance related concepts, such as continuity, compactness and convergence and provides many elementary definitions.

3.1.1 Topological Space

A topological space consists of a set X and a collection of subsets $\mathscr{X}$ that satisfy three conditions:

1. The empty set $\emptyset$ and the set X belong to $\mathscr{X}$; i.e. $\emptyset \in \mathscr{X}$, $X \in \mathscr{X}$.
2. The intersection of a finite number of sets in $\mathscr{X}$ is also in $\mathscr{X}$.
3. The union of an arbitrary number of sets in $\mathscr{X}$ is also in $\mathscr{X}$.

The elements of $\mathscr{X}$ are called open sets.

A topological space $(X,\mathscr{X})$ is called Hausdorff space respectively to be Hausdorff, if any two elements a, b in X have environments $A,B \in \mathscr{X}$ with $a \in A$, $b \in B$ which are disjunct $A \cap B = \emptyset$. This property is named after FELIX HAUSDORFF[1]. If Y is a subset of X in a topological space $(X,\mathscr{X})$, then the topology

$$\mathscr{X}|Y = \{U \cap Y \,:\, U \in \mathscr{X}\} \tag{3.1}$$

is induced by Y and $(Y,\mathscr{X}|Y)$ is called topological subspace whereas a product topology is the topology on the Cartesian product $\mathscr{X} \times \mathscr{Y}$ of two topological spaces whose open sets are the unions of subsets $X \times Y$, where X and Y are open subsets of $\mathscr{X}$ and $\mathscr{Y}$, respectively.

3.1.2 Maps and Bases

A topological basis is a set $\mathscr{B}$ of open sets such that every other open set of the topological space can be written as unions or finite intersections of $\mathscr{B}$.

The theory of topology introduces the concept of continuity. A map or function between two topological spaces is continuous, if and only if the preimage of any open set is open.

A homeomorphism, also called a continuous transformation, is a bijective function between elements in two topological spaces that is continuous in both directions.

[1] FELIX HAUSDORFF (November 8, 1868 – January 26, 1942) Felix Hausdorff was a German mathematician who founded point-set topology, Hausdorff spaces, and the concepts of metric and topological spaces.

3.1.3 Manifold

The so-called natural topology of $\mathbb{R}^n$ consists of $X=\mathbb{R}^n$ and $\mathscr{X}^n$. A subset $U\subset\mathbb{R}^n$ belongs to $\mathscr{X}^n$, if and only if every $X=(x_1,\ldots,x_n)\in U$ is surrounded by an n-dimensional Euclidean ball

$$B^n(X)=\left\{(t_1,\ldots,t_n)\,:\,(t_1-x_1)^2+\ldots+(t_n-x_n)^2<1\right\}, \tag{3.2}$$

which lies completely in U. A manifold is a topological space that is locally Euclidean, i.e. a topological space $(M,\mathscr{M})$ is an n-dimensional topological manifold, if

1. every point $X\in M$ has an open neighborhood $U\subset M$, $X\in U\in\mathscr{M}$, that is homeomorphic to an n-dimensional open Euclidean ball B^n

and if

2. M is Hausdorff.

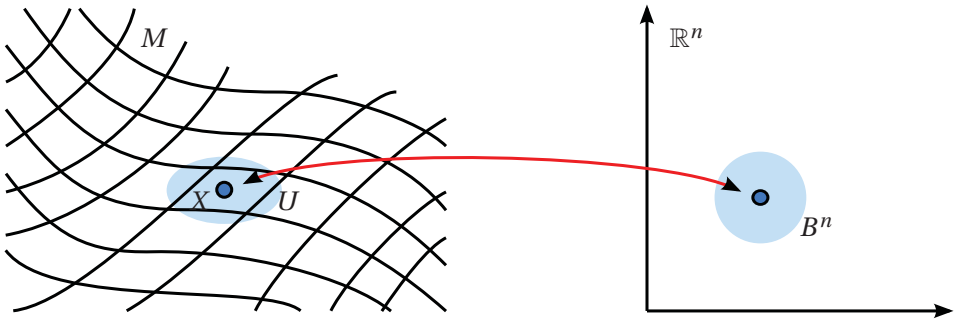


Figure 3.1: A manifold is a topological space that on a small scale resembles an n-dimensional Euclidean space; i.e. each point X has a neighborhood U that is homeomorphic to an n-dimensional open Euclidean ball B^n. Although the local structure is Euclidean, the global structure of a manifold may be more complicated.

By definition it is impossible for a manifold to include its boundary points as these points are not contained in open sets. A manifold with boundary is a Hausdorff space in which every point has a neighborhood that is homeomorphic to an open subset of Euclidean half-space

$$\mathbb{R}^n_+=\left\{(x_1,\ldots,x_n)\in\mathbb{R}^n\,:\,x_1\geq 0\right\}. \tag{3.3}$$

An n-dimensional manifold with boundary can be divided disjunctively into a set of points which form a manifold (without boundary) M and a set of points which forms the boundary of M denoted δM. The boundary δM itself is a manifold (without boundary) of dimension $n-1$. If not explicitly stated otherwise, a manifold is defined without border.

3.2 Affine Geometry

Many concepts and tools in geometric design are based on affine structures in affine spaces. An affine space is a point space with an associated linear space.

3.2.1 Affine Space

An affine space consists of

- a set $\mathbb{A}$, whose elements $\{P,Q,R,\dots\}$ are called points,
- a vector space V over a field F and
- a mapping $\omega : \left\{ \begin{array}{lcl} \mathbb{A}\times\mathbb{A} & \to & V \\ (P,Q) & \mapsto & \omega(P,Q)=\overrightarrow{PQ} \end{array} \right.$

with the compatibility properties

1.
$$\forall\, P\in\mathbb{A}\ \forall\, \mathbf{v}\in V\ \exists_1 Q\in\mathbb{A}\ :\ \overrightarrow{PQ}=\mathbf{v} \tag{3.4}$$

2.
$$\forall\, P,Q,R\in\mathbb{A}\ :\ \overrightarrow{PQ}+\overrightarrow{QR}=\overrightarrow{PR}. \tag{3.5}$$

These properties justify the notation of point-vector-additions for any point of an affine space and any vector of its associated vector space.

In case of the field $\mathbb{R}$ and the real vector space $\mathbb{R}^n$, $\mathbb{A}$ is called real affine space. A point P of $\mathbb{A}$ and a vector subspace $U\subset V$ define an affine subspace

$$\mathbb{B}=\left\{X\in\mathbb{A}\ :\ \overrightarrow{PX}\in U\right\}. \tag{3.6}$$

The dimension of an affine space is the dimension of the underlying vector space

$$\dim\mathbb{A}=\dim V. \tag{3.7}$$

According to this definition a zero-dimensional, affine space consists of a single point. A one-dimension space is a straight line, a two-dimensional space is a plane. If $\mathbb{A}$ is an n-dimensional space – written as $\mathbb{A}^n$, all $n-1$-dimensional subspaces are called hyperplanes.

If not mentioned otherwise, all affine spaces in the following text will be real, affine spaces with a finite dimension. Furthermore, the underlying real vector spaces will have an inner product $<\cdot|\cdot>$.

3.2.2 Affine Coordinate System

A tuple of $(n+1)$ points $(P_0,\ldots,P_n)$ of an affine space $(\mathbb{A}^n,V(F),\omega)$ is called a coordinate system, if the vectors

$$\overrightarrow{P_0P_1},\ldots,\overrightarrow{P_0P_n}\in V \tag{3.8}$$

are linearly independent. The point P_0 is called origin, the points $P_1,\ldots,P_n$ are called unit points and the straight line through P_0 and P_i is the i^{th}coordinate axis. For any point $Q\in\mathbb{A}$ the position vector $\overrightarrow{P_0Q}\in V$ has a unique representation

$$\overrightarrow{P_0Q}=\lambda_1\overrightarrow{P_0P_1}+\lambda_2\overrightarrow{P_0P_2}+\ldots+\lambda_n\overrightarrow{P_0P_n},\qquad \lambda_1,\ldots,\lambda_n\in F. \tag{3.9}$$

The values $\lambda_1,\ldots,\lambda_n$ are called coordinates of Q in the coordinate system $(P_0,\ldots,P_n)$. The vector $\begin{pmatrix}\lambda_1 & \ldots & \lambda_n\end{pmatrix}^{\mathrm{T}}\in F^n$ is the coordinate vector of Q. A commonly used notation is $Q(\lambda_1|\ldots|\lambda_n)$. A coordinate system $P_0,\ldots,P_n$ of an affine space is called Cartesian, if the vectors

$$\overrightarrow{P_0P_1},\ldots,\overrightarrow{P_0P_n}\in V \tag{3.10}$$

form an orthonormal basis of V.

3.2.3 Convex Hull and Barycentric Coordinates

A set of points $\mathbf{P}$ of a real, affine space $\mathbb{A}$ is called convex, if for all points $P,Q\in\mathbf{P}$ the segment

$$\left\{P+t\cdot\overrightarrow{PQ}\,:\,0\le t\le 1\right\} \tag{3.11}$$

is a subset of $\mathbf{P}$. The convex hull of a set of points $\mathbf{S}$ is the intersection of all convex sets containing $\mathbf{S}$.

If $(P_0,\ldots,P_n)$ are a coordinate system of an affine space $\mathbb{A}^n$, any point $Q\in\mathbb{A}$ can be represented by the sum of the origin P_0 and a unique, linear combination of $\overrightarrow{P_0P_i}$:

$$Q=P_0+\lambda_1\overrightarrow{P_0P_1}+\lambda_2\overrightarrow{P_0P_2}+\cdots+\lambda_n\overrightarrow{P_0P_n}. \tag{3.12}$$

Rewriting $\overrightarrow{P_0P_i}=P_i-P_0$ and collecting subterms of P_i leads to

$$\begin{aligned}Q&=1\cdot P_0+\lambda_1(P_1-P_0)+\lambda_2(P_2-P_0)+\cdots+\lambda_n(P_n-P_0) &(3.13)\\ &=\underbrace{(1-\lambda_1-\lambda_2-\cdots-\lambda_n)}_{=\lambda_0}P_0+\lambda_1P_1+\lambda_2P_2+\cdots+\lambda_nP_n. &(3.14)\end{aligned}$$

The final representation of the coordinate vector $\begin{pmatrix} \lambda_0 & \dots & \lambda_n \end{pmatrix}^{\mathrm{T}}$, $\lambda_0 + \ldots + \lambda_n = 1$ is called barycentric. In contrast to an affine coordinate system in a barycentric coordinate system none of the points $(P_0,\ldots,P_n)$ has a special interpretation as origin. Furthermore, the location of any point represented in barycentric coordinates can be determined easily as illustrated in Figure 3.2. Using the concept of barycentric coordinates the convex hull of a set of points $\mathbf{P} = \{P_0,\ldots,P_n\}$ consists of all points whose coordinates are semi-positive in the corresponding barycentric coordinate system. More general, the convex hull of any finite set of points $\mathbf{S} = \{S_0,\ldots,S_m\} \subset \mathbb{A}^n$ is

$$\left\{ \sum_{i=1}^{m} \lambda_i S_i \ : \ \forall\, \lambda_i \geq 0 \text{ and } \sum_{i=1}^{m} \lambda_i = 1 \right\}. \tag{3.15}$$

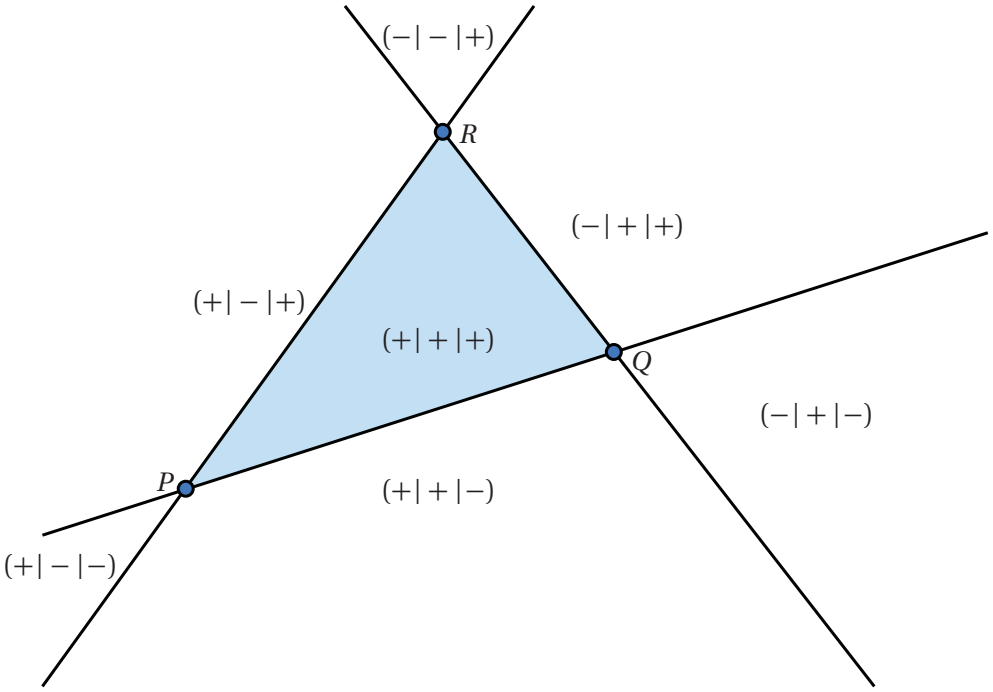


Figure 3.2: The location of points can be described by a set of reference points which define a so-called coordinate system. This illustration shows a barycentric coordinate system which consists of three points P,Q,R. Using this coordinate system the location of any point in this plane can be expressed via its unique coordinates $(\lambda_0 \,|\, \lambda_1 \,|\, \lambda_2)$. Depending on the sign of its coordinates a point belongs to one of the illustrated regions $(\pm|\pm|\pm)$. The convex hull of the points P,Q,R consists of all points of the closure of region $(+|+|+)$.

3.3 Euclidean Geometry

This Section discusses Euclidean geometry in order to introduce the mathematical background for calculating a distance. Euclidean geometry is a mathematical system attributed to EUCLID[2], whose "Elements" [Hei07] is the most successful textbook and one of the most influential works in the history of mathematics [Aum08].

3.3.1 Metric

A nonnegative function $d : X \times X \to \mathbb{R}$ describing the "distance" between objects for a given set X is called a metric, if it satisfies

$$d(x,x) = 0 \text{ and } d(x,y) = 0 \Rightarrow x = y \tag{3.16}$$

as well as the symmetry condition

$$d(x,y) = d(y,x) \tag{3.17}$$

and the triangle inequality

$$d(x,z) \leq d(x,y) + d(y,z) \tag{3.18}$$

for all $x,y,z \in X$. The most simple example, which satisfies all conditions is the discrete metric

$$d(x,y) = \begin{cases} 1, & x \neq y \\ 0, & x = y \end{cases} \tag{3.19}$$

In the field of computer-aided design (CAD) and computer graphics the Euclidean metric is of particular importance. In the following text the Euclidean distance function will be used, if not mentioned otherwise. Two points X,Y with corresponding position vectors $\mathbf{x} = \begin{pmatrix} x_1 & \dots & x_n \end{pmatrix}^{\mathrm{T}}$ and $\mathbf{y} = \begin{pmatrix} y_1 & \dots & y_n \end{pmatrix}^{\mathrm{T}}$ of an n-dimensional space have the Euclidean distance

$$d(X,Y) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}. \tag{3.20}$$

In some cases it is convenient to use the maximum metric

$$\mu(X,Y) = \max(|x_1 - y_1|, \ldots, |x_n - y_n|) \tag{3.21}$$

or the absolute value metric

$$\sigma(X,Y) = |x_1 - y_1| + \ldots + |x_n - y_n|. \tag{3.22}$$

The relationship between all these metrics in real space $\mathbb{R}^n$ is given by the inequality

$$\forall\, X,Y \in \mathbb{R}^n \;:\; d(X,Y) \leq \sqrt{n} \cdot \mu(X,Y) \leq \sqrt{n} \cdot \sigma(X,Y) \leq n \cdot d(X,Y). \tag{3.23}$$

The special case $n = 1$ leads to

$$d(X,Y) = \mu(X,Y) = \sigma(X,Y) = |\mathbf{x} - \mathbf{y}|. \tag{3.24}$$

[2] EUCLID OF ALEXANDRIA (flor. 300 BC) Euclid was a Greek mathematician and is often referred to as the "Father of Geometry". He wrote " Elements" – a mathematical and geometric treatise consisting of 13 books. It is the oldest axiomatic, deductive treatment of mathematics and was used as the basic text on geometry throughout the Western world for about 2 000 years.

3.3.2 Point Sets

The distance between a single point X and a point set $\mathbf{Y}$ can be defined using the minimum of all distances between X and a point $Y \in \mathbf{Y}$, respectively

$$d(X,\mathbf{Y}) = \min_{Y \in \mathbf{Y}} d(X,Y). \tag{3.25}$$

For two point sets there are many different ways to define an oriented distance. Oriented distances are characterized by $d(\mathbf{X},\mathbf{Y}) \neq d(\mathbf{Y},\mathbf{X})$. MARIE-PIERRE DUBUISSON and ANIL K. JAIN have analyzed the following six distance functions [DJ94]:

$$d_1(\mathbf{X},\mathbf{Y}) = \min_{X \in \mathbf{X}} d(X,\mathbf{Y}) \tag{3.26}$$

$$d_2(\mathbf{X},\mathbf{Y}) = \mathscr{K}^{\,50}_{X \in \mathbf{X}}\, d(X,\mathbf{Y}) \tag{3.27}$$

$$d_3(\mathbf{X},\mathbf{Y}) = \mathscr{K}^{\,75}_{X \in \mathbf{X}}\, d(X,\mathbf{Y}) \tag{3.28}$$

$$d_4(\mathbf{X},\mathbf{Y}) = \mathscr{K}^{\,90}_{X \in \mathbf{X}}\, d(X,\mathbf{Y}) \tag{3.29}$$

$$d_5(\mathbf{X},\mathbf{Y}) = \max_{X \in \mathbf{X}} d(X,\mathbf{Y}) \tag{3.30}$$

$$d_6(\mathbf{X},\mathbf{Y}) = \frac{1}{|\mathbf{X}|} \sum_{X \in \mathbf{X}} d(X,\mathbf{Y}) \tag{3.31}$$

where $|\cdot|$ denotes the cardinal number of a set and $\mathscr{K}^{\,j}_{X \in \mathbf{X}}$ represents the ranked distance; i.e. $\mathscr{K}^{\,0}_{X \in \mathbf{X}}$ corresponds to the minimum, $\mathscr{K}^{\,50}_{X \in \mathbf{X}}$ to the median and $\mathscr{K}^{\,100}_{X \in \mathbf{X}}$ to the maximum of all distances $d(X,\mathbf{Y}), X \in \mathbf{X}$.

While it is sensible to use the minimum function for distances between a point and a point set, nested minimum functions do not define a meaningful distance between two point sets. All point sets $\mathbf{X}$ and $\mathbf{Y}$ with non-empty intersections would have a distance of zero. The oriented Hausdorff distance, named after FELIX HAUSDORFF, does a better job. Its definition (d_5) utilizes the maximum function.

Taking the maximum of both oriented distances leads to a non-oriented distance; e.g. the non-oriented Hausdorff distance between $\mathbf{X}$ to $\mathbf{Y}$ takes the maximum of both oriented distances:

$$H(\mathbf{X},\mathbf{Y}) = \max(d_5(\mathbf{X},\mathbf{Y}), d_5(\mathbf{Y},\mathbf{X})) \tag{3.32}$$

$$= \max\left(\max_{X \in \mathbf{X}} d(X,\mathbf{Y}), \max_{Y \in \mathbf{Y}} d(Y,\mathbf{X})\right). \tag{3.33}$$

Figure 3.3 shows an illustrative example on Hausdorff calculations using d_5. The combination of the other distance functions (d_1, ..., d_4, d_6) results in

$$D_1(\mathbf{X},\mathbf{Y}) = \max(d_1(\mathbf{X},\mathbf{Y}), d_1(\mathbf{Y},\mathbf{X})) \tag{3.34}$$

$$D_2(\mathbf{X},\mathbf{Y}) = \max(d_2(\mathbf{X},\mathbf{Y}), d_2(\mathbf{Y},\mathbf{X})) \tag{3.35}$$

$$D_3(\mathbf{X},\mathbf{Y}) = \max(d_3(\mathbf{X},\mathbf{Y}), d_3(\mathbf{Y},\mathbf{X})) \tag{3.36}$$

$$D_4(\mathbf{X},\mathbf{Y}) = \max(d_4(\mathbf{X},\mathbf{Y}), d_4(\mathbf{Y},\mathbf{X})) \tag{3.37}$$

$$D_6(\mathbf{X},\mathbf{Y}) = \max(d_6(\mathbf{X},\mathbf{Y}), d_6(\mathbf{Y},\mathbf{X})) \tag{3.38}$$

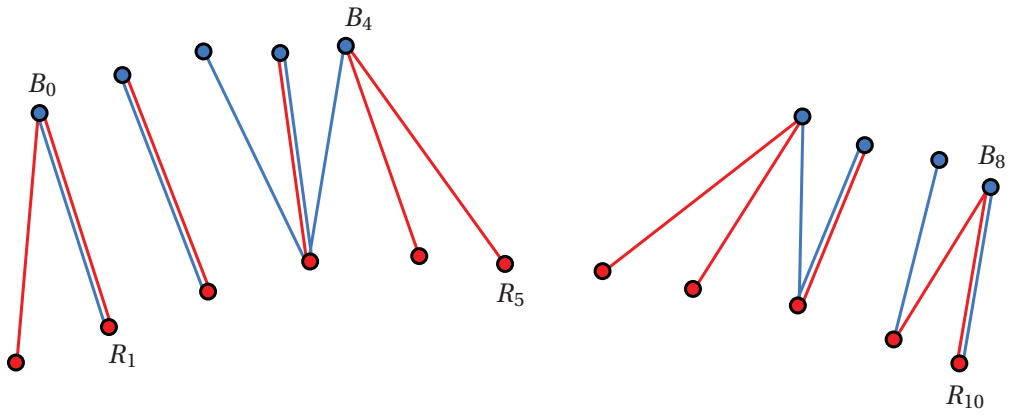


Figure 3.3: The Hausdorff metric defines the distance between two sets. For illustrative purposes each point of one set is connected with its nearest neighbor of the other set. The oriented Hausdorff distance from the blue points to the red ones can be found between B_0 and R_1 (longest blue line). The oriented Hausdorff distance from the red points to the blue ones is between R_5 and B_4 (longest red line). The maximum of both distances – the distance between B_4 and R_5 – is the Hausdorff distance between these point sets.

Please note that these functions are not metrics in contrast to the Hausdorff distance. $D_1, \ldots, D_4$ do not fulfill the condition

$$D(\mathbf{X},\mathbf{Y}) = 0 \Rightarrow \mathbf{X} = \mathbf{Y}, \tag{3.39}$$

which could be a problem for object matching. D_6 violates the triangle inequality. Nevertheless, D_3, D_4, and D_6 have some importance in the field of computer vision.

3.3.3 Signed distance

If a closed surface in $\mathbb{R}^3$ defines an inner and an outer space, it is convenient to indicate the location of a point in the sign of the measured distance. By convention points in outer space have positive distance, points in inner space have negative distance.

Distance Visualization

While distance calculation itself is a mathematical, algorithmic problem. The visualization has to deal with colors and color perception and – last but not least – it is embedded into a context.

Without loss of generality, it is sufficient to calculate distances between point sets. Other geometric representations can be converted via dense sampling. Methods for calculating distances and errors between surfaces are presented in "Metro: Measuring Error on Simplified Surfaces" [CRS98], "MESH: Measuring Error between Surfaces using the Hausdorff distance" [ASCE02], and "Abstand: Distance Visualization for Geometric Analysis" [USF08a].

In order to speed up the calculation of a Hausdorff distance, which has a quadratic runtime in a naive implementation, the samples can be stored in kd-trees [GBY91] or grid structures. These data structures and their algorithms are described amongst others by GERALD FARIN et al. [FHH03] as well as HANS-CHRISTIAN HEGE and KONRAD POLTHIER [HP02]. The nearest-neighbor-search algorithm implemented in our tool *Abstand*has an average runtime of $O(n \cdot log(n))$ and is described in "An introductory tutorial on kd-trees" [Moo91]. The calculation of normal distances (i.e. the nearest neighbor search restricted to points inside a double cone as illustrated in Figure 3.4) is based on grid hashing – a technique presented in "Optimized Spatial Hashing for Collision Detection of Deformable Objects" [THM+03].

Depending on the context it is sensible to classify the distance visualization problem into two categories: the *asymmetric*case analyzes two geometric objects assuming that the first object is the reference / nominal object. The second object is the actual object to be validated. Such a configuration can be found e.g. in the context of quality management using a CAD model as reference to check the resulting product. The *symmetric*case is characterized by the absence of a reference model. Both objects are on a par. In contrast to the asymmetric case the results of the symmetric one should not change, if the order of the objects to analyze is swapped. A typical, symmetric situation is the comparison of two range maps of a laser scanning process. If overlapping regions of aligned scans are analyzed, none of them can be considered to be the ground truth.

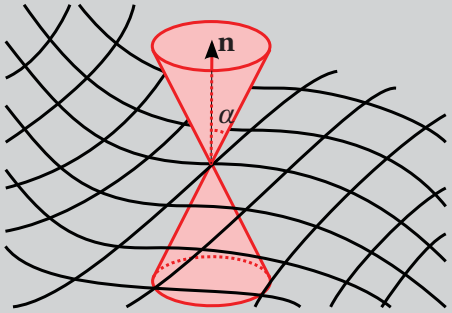


Figure 3.4: In some cases it is sensible to restrict the nearest-neighbor-search to samples inside a double cone along normal direction. Undesired sample relations at parts, which do not have a corresponding counter part, can be avoided.

In *Abstand* the distance calculation results of an asymmetric setup can be limited to one-sided distances. As one surface is considered as ground truth, the visualization emphasizes the actual object. The reference object plays a minor role in the visualization.

Its main purpose is to provide orientation in 3D – especially if the actual objects (e.g. scanned remains of a vase) are much smaller than the reference object. The actual object may also be colorized according to the distances. Solid cylinders (or prisms with a lower polygon count) are generated to visualize the distances. These distance visuals are grouped using the calculated histograms.

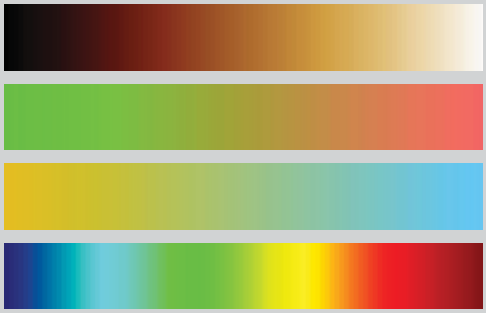


Figure 3.5: A color table should take human perception into account. For 2D data many authors propose the Black-Body-Radiation scheme (first row). According to DAVID BORLAND et al. geometry should be drawn in an isoluminant scheme (second and third rows). Depending on the context a neutral color scheme as proposed by BERNICE E. ROGOWITZ et al. (third row) is sensible. It does not use "signal colors" such as red. A deceptive and misleading color range is the rainbow color scheme (last row) which is often used in various visualization systems.

For a symmetric setup the two analyzed meshes are colored in an unobtrusive coloring. The transparency value may vary according to the signed distance. This technique enables to display inner parts which would otherwise be covered by opaque surfaces.

The geometric objects / meshes as well as the distance visuals can easily be colorized using preconfigured color scales. Most distance visualization schemes use luminance-based scales, for example the black-body radiation spectrum. For surfaces isoluminant color maps with opponent colors are suggested (see Figure 3.5). These surface colorizations do not compromise the depth perception. Neutral color tables are also available, if extra highlighting of differences is not desired. If the geometry is shown in a single color (with possibly varying transparency), the application *Abstand* proposes a color which does not belong to the color scale. Furthermore it automatically generates a legend in an appropriate range.

Having calculated the distances our application offers predefined color palettes. These schemes include color maps with good order properties in terms of human perception. An overview on colors and color perception can be found in MAUREEN STONE's field guide to digital color [Sto03]. The set of predefined color maps contains the luminance-based maps, for example the black-body radiation spectrum, with only small variations in the hue value [LH92], [BRT95] as well as the maps proposed in "Rainbow Color Map (Still) Considered Harmful" [BTI07]. For surfaces isoluminant color maps with opponent colors are suggested (see Figure 3.5). These surface colorizations do not compromise the depth perception. Neutral color tables are also available, if extra highlighting of differences is not desired. The selection of the neutral color ranges are based on "How NOT to lie with visualization" [RTB96].

3.4 Projective Geometry

According to the *Erlanger Programm* of FELIX KLEIN[3] projective geometry is the unifying frame for all other geometries. He interprets affine, metric, and Euclidean geometries as special cases of projective geometry.

3.4.1 Projective Space

A set $\mathbb{P}$ is a real projective space over the field $\mathbb{R}$, if

1. a vector space $V(\mathbb{R})$ over the field $\mathbb{R}$ and
2. a bijective mapping β between $\mathbb{P}$ and the set of one-dimensional vector subspaces of V

exist. If V has dimension $n+1$, then the projective space $\mathbb{P}(V)$ is called n-dimensional, which is abbreviated $\mathbb{P}^n$. The elements of $\mathbb{P}$ are referred as points. Each point $X \in \mathbb{P}(V)$ has a corresponding one-dimensional vector subspace $[\mathbf{x}] = \beta(X)$, $(\mathbf{x} \neq \mathbf{o})$ in V. This correspondence is commonly expressed by $X = [\mathbf{x}]$. Any two points $X = [\mathbf{x}]$ and $Y = [\mathbf{y}]$ have the following correlation

$$X = Y \Leftrightarrow \exists\, \lambda \neq 0 \;:\; \mathbf{x} = \lambda \mathbf{y} \tag{3.40}$$

with a homogeneous scaling factor λ.

A k-dimensional, real projective subspace $\mathbb{S}$ of a projective space $\mathbb{P}(V)$ is defined by a $k+1$-dimensional vector subspace $U \subset V$. $\mathbb{S}(U)$ consists of all one-dimensional vector subspaces of U. This definition ensures that every real projective subspace is a projective space.

- A projective subspace of dimension $k = 0$ of a projective space $\mathbb{P}^n$ is called point,
- a one-dimensional subspace ($k = 1$) is a projective line,
- a two-dimensional subspace ($k = 2$) is a projective plane and
- a $k = n - 1$ dimensional subspace is called a projective hyperplane.

While the intersection of two projective subspaces is a projective subspace as well, the union of two subspaces is normally not a projective subspace. For any finite-dimensional projective subspaces $\mathbb{S}_1$ and $\mathbb{S}_2$ of $\mathbb{P}$ the theorem of dimensions is valid:

$$\dim \mathbb{S}_1 + \dim \mathbb{S}_2 = \dim(\mathbb{S}_1 \cap \mathbb{S}_2) + \dim(\mathbb{S}_1 + \mathbb{S}_2) \tag{3.41}$$

[3] FELIX CHRISTIAN KLEIN (April 25, 1849 – June 22, 1925) Felix Klein was a German mathematician, known for his work in group theory, function theory, and geometry. His *Erlanger Programm* (1872), classifying geometries by their underlying symmetry groups, was a hugely influential synthesis of geometry.

where the join space $\mathbb{S}_1(U_1)+\mathbb{S}_2(U_2)$ is defined as

$$\{X\in\mathbb{P}\,:\,X=[\mathbf{x}],\mathbf{x}\neq\mathbf{o},\mathbf{x}=\mathbf{p}_1+\mathbf{p}_2,\mathbf{p}_1\in U_1,\mathbf{p}_2\in U_2\}. \tag{3.42}$$

If not stated differently, all following projective spaces will have a finite dimension.

3.4.2 Projective Coordinates

Any $k+1$ points ($k\geq 0$) of a projective space $\mathbb{P}$ are called linearly independent, if they cannot be contained in a $(k-1)$-dimensional subspace of $\mathbb{P}$.

$k+1$ linearly independent points $P_0,\ldots,P_k\in\mathbb{P}$ define exactly one k-dimensional, projective subspace $\mathbb{S}$ of $\mathbb{P}$ with $P_i\in\mathbb{S}$ ($i=0,\ldots,k$). Furthermore, $\mathbb{S}$ can be written as join space

$$\mathbb{S}=P_0+\ldots+P_k. \tag{3.43}$$

Consequently, the join space of $n+1$ linearly independent points of an n-dimensional projective space $\mathbb{P}$ is the $\mathbb{P}$ itself. Therefore, an obvious approach to define projective coordinates uses $n+1$ points:

An ordered $(n+1)$-tuple $(x_0,\ldots,x_n)$ is called projective coordinates of a point X of an n-dimensional, real projective space $\mathbb{P}(V)$ with $X=[\mathbf{x}]$ and $\mathbf{x}=\sum_{i=0}^{n}x_i\mathbf{b}_i$ using the vector space basis $\{\mathbf{b}_0,\ldots,\mathbf{b}_n\}$ of V^{n+1}.

Due to $X=[\mathbf{x}]=[\lambda\cdot\mathbf{x}](\lambda\neq 0)$ the coordinates are determined up to a scale factor which is normally omitted. The projective coordinates are unique (up to a scale factor) in respect of the basis $\{\mathbf{b}_0,\ldots,\mathbf{b}_n\}$, but not of the points $B_0,\ldots,B_n$.

Therefore, $n+1$ points are not sufficient to define a projective coordinate system.

In an n-dimensional projective space $n+2$ points are said to be in general position, if any $n+1$ points are linearly independent. The $n+2$ points

$$B_0=[\mathbf{b}_0],\ldots,B_n=[\mathbf{b}_n],E=\left[\sum_{i=0}^{n}\mathbf{b}_i\right]\in\mathbb{P}^n \tag{3.44}$$

are in general position and form a projective coordinate system. The points B_i are called fundamental points whereas the point E is called unit point. Using this coordinate system the coordinate vector $\begin{pmatrix} x_0 & \ldots & x_n \end{pmatrix}^{\mathrm{T}}$ represents the projective coordinates of X in respect of the coordinate system $\{B_0,\ldots,B_n;E\}$. As in affine spaces, a point X and its coordinate vector can be written $X(x_0|\ldots|x_n)$.

3.4.3 Affine Spaces ⟷ Projective Spaces

The interrelationship between affine spaces and projective spaces can be used in various contexts. The step from a projective space to an affine space is described by the following theorem.

In a projective space $\mathbb{P}(V)$ of dimension n a hyperplane $\mathbb{H}$ with corresponding vector space U can be represented by a linear form h:

$$X = [\mathbf{x}] \in \mathbb{H} \Leftrightarrow h(\mathbf{x}) = 0 \Leftrightarrow \mathbf{x} \in U. \tag{3.45}$$

The set $\mathbb{A}$ of all points, which do not belong to the hyperplane

$$\mathbb{A} = \mathbb{P} \backslash \mathbb{H}, \tag{3.46}$$

the mapping ω

$$\omega : \begin{cases} \mathbb{A} \times \mathbb{A} & \rightarrow & U \\ (X,Y) & \mapsto & \mathbf{x} - \mathbf{y} \end{cases} \tag{3.47}$$

with $X = [\mathbf{x}], Y = [\mathbf{y}], h(\mathbf{x}) = h(\mathbf{y}) = 1$ and the corresponding vector space U define an affine space $(\mathbb{A}, U, \omega)$. Its dimension is n.

In other words, removing a hyperplane from a projective space creates an affine space with the same dimension. The points within the hyperplane are ideal points. These points are sometimes interpreted as points at infinity. They do only depend on the choice of the hyperplane.

The step from an affine space to a projective space is a projective completion. It interprets an n-dimensional affine space $\mathbb{A}^n(V^n)$ as a hyperplane of an $(n+1)$-dimensional, affine space $\mathbb{A}^{n+1}(V^{n+1})$. An arbitrary point O which does not belong to the subspace $\mathbb{A}^n$ defines a hyperplane $\overline{\mathbb{A}^n}$ which is parallel to $\mathbb{A}^n$ and which contains O. This setting is illustrated in Figure 3.6.

Within the vector space V^{n+1} it is possible to select vectors $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n$ such that $\{\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is a basis of V^{n+1} and $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ forms a basis of V^n. These vectors are the foundation of a projective coordinate system

$$B_0 = [\mathbf{b}_0], \ldots, B_n = [\mathbf{b}_n], E = \left[\sum_{i=0}^{n} \mathbf{b}_i \right] \in \mathbb{P}^n. \tag{3.48}$$

Within this system a point $X(x_1 | \ldots | x_n)$ which belongs to $\mathbb{A}^n$ has the projective coordinates $X(1 | x_1 | \ldots | x_n)$ up to a scale factor. Points, which belong to the projective completion, are ideal points with projective coordinates $Y(0 | y_1 | \ldots | y_n)$.

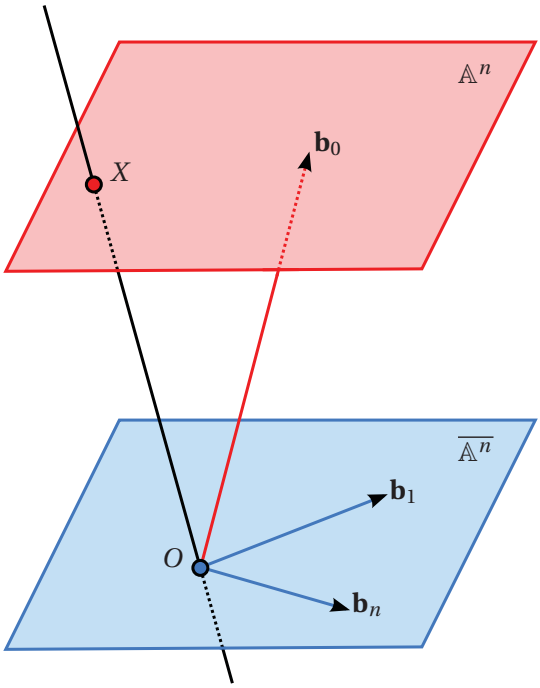


Figure 3.6: In the interrelationship between affine spaces and projective spaces a distinguished hyperplane plays a prominent role. Removing a hyperplane from a projective space creates an affine space of same dimension. Adding a hyperplane of ideal points to an affine space completes the affine space projectively.

Consequently, for each affine coordinate system $\{B_0,\ldots,B_n\}$ of an affine space $\mathbb{A}$ exists a projective coordinate system $\{B_0,\ldots,B_n;E\}$ in $\mathbb{P}$ such that

1. the ideal hyperplane can be described by the equation $x_0 = 0$,
2. a point $X(x_1|\ldots|x_n)$ in $\mathbb{A}$ with coordinates in the coordinate system $\{B_0,\ldots,B_n\}$ has the projective coordinate vector $\begin{pmatrix} 1 & x_1 & \ldots & x_n \end{pmatrix}^{\mathrm{T}}$ with respect to the coordinate system $\{B_0,\ldots,B_n;E\}$ in $\mathbb{P}$,
3. a point $X \in \mathbb{A}$ with the projective coordinate vector $\begin{pmatrix} x_0 & x_1 & \ldots & x_n \end{pmatrix}^{\mathrm{T}}$ has the affine coordinate vector $\begin{pmatrix} x_1/x_0 & \ldots & x_n/x_0 \end{pmatrix}^{\mathrm{T}}$.

To indicate the special interpretation of one coordinate axis in a projective completion the coordinate tuple $(x_0,x_1,\ldots,x_n)$ can be written $(x_1,\ldots,x_n,x_0)$ – respectively (x,y,z,w) instead of (w,x,y,z). In order to distinguish affine and projective coordinate vectors, projective coordinates will be written $\mathbf{x} = \lambda\mathbf{x}$ in the subsequent text.

3.4.4 Duality Principle

In a projective space $\mathbb{P}(V)$ of dimension n all points which belong to a hyperplane $\mathbb{H}$ with corresponding vector space U satisfy a linear equation:

$$X(x_0|\ldots|x_n)\in\mathbb{H} \Leftrightarrow h_0x_0+\ldots+h_nx_n=0. \tag{3.49}$$

This equation is symmetrical in the hyperplane coefficients and the point coefficients

$$\begin{pmatrix} h_0 & \ldots & h_n \end{pmatrix}\cdot\begin{pmatrix} x_0 & \ldots & x_n \end{pmatrix}^{\mathrm{T}}=0. \tag{3.50}$$

Consequently, the interpretation of these coefficients can be interchanged, so that the space of all hyperplanes can be considered to be another projective space called the dual space of the original space $\mathbb{P}(V)$. By symmetry the dual space of a dual space is the original space. This symmetry has a very important consequence known as duality principle. A theorem in projective geometry is a statement on figures (projective subspaces) and configurations (incidence, union, intersection). Any theorem T in a projective space $\mathbb{P}$ also holds in the dual space $\mathbb{P}^\star$ due to isomorphism.

The dual version of a theorem can be retrieved by exchanging the terms and configurations in $\mathbb{P}$ by those of $\mathbb{P}^\star$ as listed in Table 3.1. Propositions and theorems which are equivalent to their duals are said to be self-dual.

3.4.5 Projections

As the notation of a central projection is shorter in a projective formulation than its corresponding affine version, it is convenient to use a projective completion to combine both representations. GÜNTER AUMANN [AS93] derived the formula of a central projection as follows.

A three-dimensional central projection consists of a center $C(c_0|c_1|c_2)$ and a projection plane, which shall be defined by an incident point P and a normal vector $\mathbf{n}$, $||\mathbf{n}||=1$ pointing towards the half-space containing C. This situation is illustrated in Figure 3.7.

If the orthogonal projection of C into the projection plane is called H, its position vector $\mathbf{h}$ can be used to calculate the distance d between the projection center C and the image plane

$$d=d(C,H)=\mathbf{n}^{\mathrm{T}}(\mathbf{c}-\mathbf{h}). \tag{3.51}$$

As the image $X^\star$ of a projected point X has the position vector

$$\mathbf{x}^\star=\mathbf{x}+\lambda(\mathbf{x}-\mathbf{c}) \tag{3.52}$$

and fulfills the image plane's equation

$$\mathbf{n}^{\mathrm{T}}(\mathbf{x}^\star-\mathbf{p})=0, \tag{3.53}$$

Duality Principle

Statements on figures and configurations in an n-dimensional projective space $\mathbb{P}$ and their dual versions

projective space	projective dual space
$\mathbb{P}^n(V^{n+1})$	$\emptyset$
hyperplane $\mathbb{X}^{n-1}$	point X
k-dim. projective subspace $\mathbb{X}^k$	$(n-k-1)$-dim. projective subspace $\mathbb{X}^{n-k-1}$
...	...
line X^1	$(n-2)$ plane $\mathbb{X}^{n-2}$
point X	hyperplane $\mathbb{X}^{n-1}$
$\emptyset$	$\mathbb{P}^n(V^{n+1})$
join space $\mathbb{R}^k + \mathbb{S}^l$	intersection $\mathbb{R}^{n-k-1} \cap \mathbb{S}^{n-l-1}$
intersection $\mathbb{R}^k \cap \mathbb{S}^l$	join space $\mathbb{R}^{n-k-1} + \mathbb{S}^{n-l-1}$
incidence $\subset$	incidence $\supset$

Table 3.1: All propositions and theorems in projective geometry occur in two version which are dual to each other. The dual version can be inferred by interchanging the terms listed in this table. If a proposition or theorem is equivalent to its dual, it is said to be self-dual.

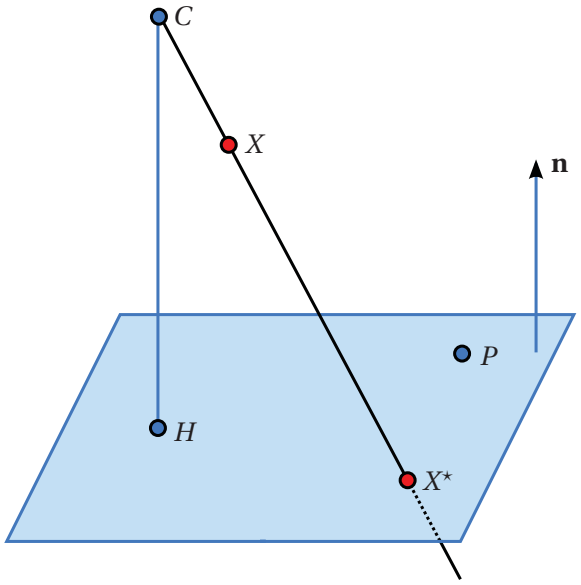


Figure 3.7: A central projection maps a point X to its image $X^\star$ whereas $X^\star$ is the intersection of an image plane with a ray, which starts at the projection center C and passes X. The image plane is defined by a point P and a normal vector $\mathbf{n}$.

the parameter λ can be determined:

$$\lambda = \frac{\mathbf{n}^\mathrm{T}(\mathbf{x}-\mathbf{p})}{\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{p})} = \frac{\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{p})}{\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{x})} - 1. \tag{3.54}$$

Hence $X^\star$ has the position vector

$$\mathbf{x}^\star = \frac{\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{p})}{\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{x})}(\mathbf{x}-\mathbf{c}) + \mathbf{c}. \tag{3.55}$$

Points whose projection ray is parallel to the image plane have to be excluded, as for those points the denominator $\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{x})$ becomes zero. Rearranging Equation (3.55) leads to

$$\begin{aligned}
\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{x})\mathbf{x}^\star &= \mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{x})\mathbf{z} + \mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{p})(\mathbf{x}-\mathbf{c}) && (3.56)\\
&= (\mathbf{n}^\mathrm{T}\mathbf{c})\mathbf{c} - (\mathbf{c}\mathbf{n}^\mathrm{T})\mathbf{x} + \mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{p})\mathbf{x} - \mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{p})\mathbf{c} && (3.57)\\
&= \underbrace{(\mathbf{n}^\mathrm{T}(\mathbf{c}-\mathbf{p})\mathbf{E}_3 - \mathbf{c}\mathbf{n}^\mathrm{T})\mathbf{x}}_{\mathbf{A}\mathbf{x}} + \underbrace{(\mathbf{n}^\mathrm{T}\mathbf{p})\mathbf{z}}_{\mathbf{t}} && (3.58)
\end{aligned}$$

with a 3×3 identity matrix $\mathbf{E}_3$.

If the point P coincides with H, the projection simplifies to

$$\mathbf{A} = d\,\mathbf{E}_3 - \mathbf{c}\mathbf{n}^{\mathrm{T}}, \qquad \mathbf{t} = (\mathbf{n}^{\mathrm{T}}\mathbf{h})\mathbf{c}. \tag{3.59}$$

Furthermore, with a coordinate system such that C has the coordinates $C(0|0|d)$ and the image plane is $x_3 = 0$, the projection of a Q with projective coordinates $\mathbf{q}$ has the homogeneous form

$$\begin{pmatrix} q_1^\star \\ q_2^\star \\ q_3^\star \\ q_0^\star \end{pmatrix} = \rho \left(\begin{array}{ccc|c} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & -1 & d \end{array}\right) \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_0 \end{pmatrix}, \qquad \rho \neq 0. \tag{3.60}$$

Hardware accelerated 3D rendering libraries, such as OpenGL [OA93], are also based on the concepts of projective geometry and use projective/homogeneous coordinates to describe geometric objects. The main advantage is the consistent representation of affine transformations. Instead of 3×3 matrix and a 3-dimensional translation vector to represent an affine transformation, a homogeneous representation uses a single 4×4 matrix. The consecutive transformations become a sequence of matrix multiplications.

But in contrast to projective geometry, in which each projected point is mapped into a plane, 3D rendering libraries use matrices, which preserve the depth value. The z value is needed for clipping and depth testing.

Projector Calibration

The goal of virtual reality (VR) is the synthesis of a convincing illusion within a virtual environment. The most important aspect in a VR system is the quality of vision because it has the greatest and most immediate impact on the human sensory system [BSdV01]. Besides the displayed content the display quality and especially its correct calibration is of big importance. Current calibration methods can be subdivided into three groups. The first and most frequently used method is the calibration by hand which is a non-practicable method for large power walls. Contrary to manual calibration the second group consists of methods with an automatic approach using embedded cameras [LS04], [RB01] or other sensor systems [LDMA+04]. These systems have the great advantage to work out of the box. Unfortunately, they need specialized or modified hardware and projectors. To overcome this disadvantage the third group uses standard video cameras that are neither embedded in the projector nor rigidly attached to it. These systems are not only suitable for mobile activities and applications [SSM01], but also for fixed installations [BMY05]. Especially in virtual 3D environments in which cameras are already installed – e.g. for tracking purposes – this approach is widely used [SHVG02], [GWN+03].

The method by MARCEL LANCELLE et al. [LOU+06] is a semi-automated calibration procedure without additional hardware. The main difference to established systems is its seamless integration into the render pipeline, which is similar to an approach by RAMESH RASKAR [Ras00] who also uses a projection matrix modification to implement projector calibrations.

In a hardware accelerated 3D application based on OpenGL [OA93] a point P is transformed to $\mathbf{q} = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}^{\mathrm{T}} = \mathbf{P} \cdot \mathbf{M} \cdot \mathbf{p}$ with the modelview matrix $\mathbf{M}$ and the projection matrix $\mathbf{P}$. The perspective division leads to the normalized screen coordinates $S(q_x/q_w \,|\, q_y/q_w)$. A 4×4 calibration matrix $\mathbf{C}$ that is applied after all other coordinate transformation matrices but before the perspective division can correct distorted displays, if for at least four reference points Q_i the user specifies corrected points R_i. The calibration matrix

$$\mathbf{C} = \begin{pmatrix} a & b & 0 & d \\ e & f & 0 & h \\ i & j & 1 & 0 \\ m & n & 0 & 1 \end{pmatrix} \tag{3.61}$$

is calculated such that all Q_i will be projected as close as possible to the target positions R_i. This matrix is a combination $\mathbf{C} = \mathbf{C}_T \cdot \mathbf{C}_{Sh} \cdot \mathbf{C}_S \cdot \mathbf{C}_{Tr} \cdot \mathbf{C}_R$ of the matrices shown in Figure 3.8.

a, b, e and f need to be optimized to correct for scaling, shearing and rotation, d and h belong to the translation. Using m and n it is possible to correct for scaling depending on the screen coordinates which represents a trapezoid distortion.

The latter case is more complex than the previous ones. The distortion is not affine and additional scaling and translational effects occur when $m \neq 0$ or $n \neq 0$ [FH05a].

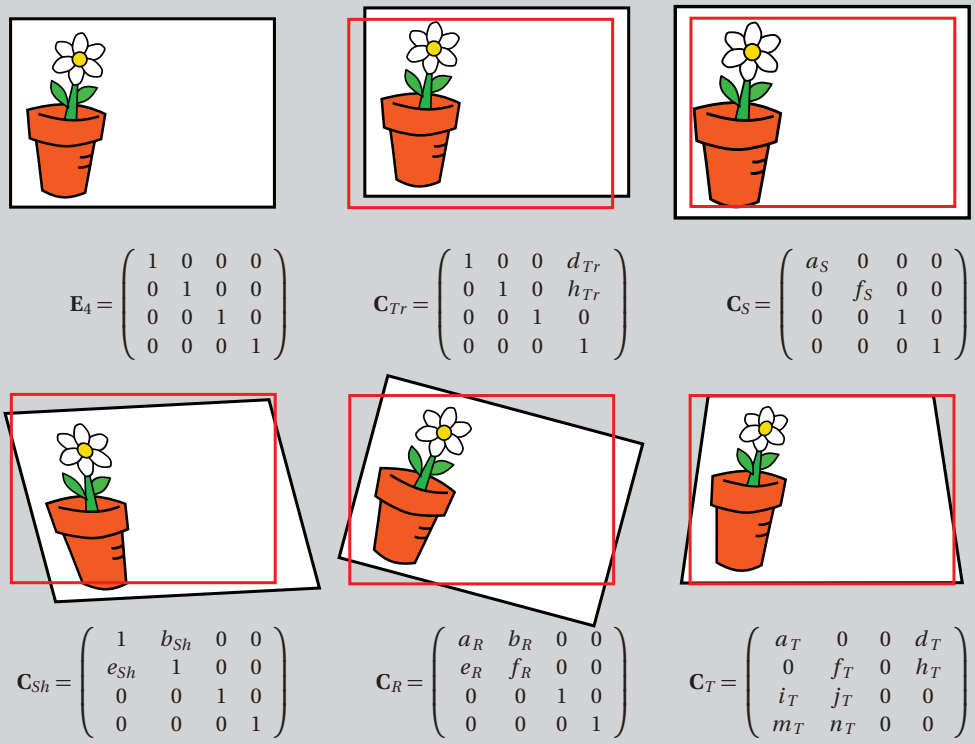


Figure 3.8: Matrices to handle translation, scaling, shearing, rotation and trapezoid deformation. These effects occur when projecting on a planar surface and can be corrected with calibration methods [LOU+06].

Also the depth values are affected during perspective division so that problems with the clipping planes may occur. Since a shearing of the near clipping plane is undesired in most applications, one solution to this problem is to set i and j to m and n respectively to realign the near clipping plane (which in turn increases distortions of the far clipping plane).

With introducing m and n in the calibration matrix the resulting depth value will be changed from q_z/q_w to $q_z/(m \cdot q_x + n \cdot q_y + q_w)$. RAMESH RASKAR addresses this problem by introducing $k = 1 - |m| - |n|$ into

$$\mathbf{C}_T = \begin{pmatrix} \star & \star & \star & \star \\ \star & \star & \star & \star \\ 0 & 0 & k & 0 \\ m & n & 0 & 1 \end{pmatrix} \tag{3.62}$$

to keep the whole visible range from the case without calibration. But still the near clipping plane is sheared resulting in strong artifacts with intersecting objects, especially with multiple projectors.

3.5 Differential Geometry

This section analyzes local aspects and properties of curves and surfaces in Euclidean space $\mathbb{E}^3$ using the methods of differential and integral calculus.

3.5.1 Differential Geometry of Curves

If $I \subset \mathbb{R}$ is an interval (the parameter domain) and

$$\varphi : \left\{ \begin{array}{lcl} I & \to & \mathbb{E}^3 \\ u & \mapsto & X(u) = X(x_1(u)|x_2(u)|x_3(u)) \end{array} \right. \tag{3.63}$$

a function of differentiability class C^r ($r \geq 1$), then the point set

$$c = \{X(u) \,:\, u \in I\} \tag{3.64}$$

is called a C^r-curve with parametrical representation φ. For a fixed parameter $u_0 \in I$ a tangent vector of c in $X(u_0)$ is defined by

$$\dot{\mathbf{x}}(u_0) = \begin{pmatrix} \dot{x}_1(u_0) \\ \dot{x}_2(u_0) \\ \dot{x}_3(u_0) \end{pmatrix} = \begin{pmatrix} \frac{\mathrm{d}x_1}{\mathrm{d}u}(u_0) \\ \frac{\mathrm{d}x_2}{\mathrm{d}u}(u_0) \\ \frac{\mathrm{d}x_3}{\mathrm{d}u}(u_0) \end{pmatrix} = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}u}(u_0). \tag{3.65}$$

The point $X(u_0)$ is a regular point with respect to φ, if $\dot{\mathbf{x}}(u_0) \neq \mathbf{0}$. Otherwise it is a singular point. If all points $X(u)$ of a curve c are regular with respect to φ, c itself is called regular. Each regular point $X(u_0)$ has a tangent line. This line passes $X(u_0)$ in direction of tangent vector $\dot{\mathbf{x}}(u_0)$.

As the definition above does not require φ to be an injection, a C^r-curve may have self-intersections. A point, which is traversed at least twice, i.e. $u_1, u_2 \in I$ with $u_1 \neq u_2$ and $X(u_1) = X(u_2)$, is called double point. A regular curve, which does not have any double points, is called a simple curve.

3.5.2 Change of Parameter

A C^r-curve $c : \mathbf{x}(u), u \in I$ may have different parametrizations. A surjective C^m function

$$f : \left\{ \begin{array}{lcl} J & \to & I \\ v & \mapsto & f(v) = u = u(v) \end{array} \right. \tag{3.66}$$

($m \geq r$) which maps onto the domain I is called an allowable change of parameter of class C^m, if it has m continuous derivatives, and if $\frac{\mathrm{d}f}{\mathrm{d}v} = \frac{\mathrm{d}u}{\mathrm{d}v}$ is nonzero for all v in J. If $\frac{\mathrm{d}f}{\mathrm{d}v}$ is positive for all v, then it is called an orientation preserving change of parameter. In this case the points of the curve are traversed in the same order both by $\mathbf{x}(u)$ and $\mathbf{x}(u(v))$; i.e. both representations have the same orientation. If $\frac{\mathrm{d}f}{\mathrm{d}v} < 0$ for all v, the two representations of a curve have opposite orientation. Note that since $\frac{\mathrm{d}f}{\mathrm{d}v} \neq 0$, the function $u(v)$ is one-to-one so that the inverse $v(u)$ exists and is an allowable change of parameter.

The terms "regular", "simple" and "tangent line" (but not "tangent vector") are invariant under allowable changes of parameter.

3.5.3 Arc Length Parametrization

For a regular C^1 curve $c : \mathbf{x}(u), u \in I$ and a fixed parameter $u_0 \in I$, the integral

$$s(u) = \int_{u_0}^{u} ||\dot{\mathbf{x}}(\xi)|| \, \mathrm{d}\,\xi \tag{3.67}$$

is the arc length of c. The map $s \mapsto u(s)$ is an allowable, orientation-preserving change of parameter. If $\mathbf{x}'$ is defined as differential of arc length

$$\mathbf{x}' = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\,s}, \tag{3.68}$$

the transformed tangent vector has normalized length $||\mathbf{x}'|| \equiv 1$. Furthermore, the length of the curve c between two points P, Q with parameters p respectively q can be calculated via

$$L_p^q = s(q) - s(p). \tag{3.69}$$

In the following text the parameter s of a curve will always be its parametrization with respect to the curve's arc length.

3.5.4 Local Coordinate System

In any point $X(s)$ of a regular C^2-curve c with $\mathbf{x}''(s) \neq \mathbf{o}$ the Frenet[4] frame

$$\{X(s); \mathbf{t}(s), \mathbf{n}(s), \mathbf{b}(s)\} \tag{3.70}$$

defines a local coordinate system. It consists of a tangent vector $\mathbf{t}(s)$, a normal vector / curvature vector $\mathbf{n}(s)$, and a binormal vector $\mathbf{b}(s)$

[4] Jean Freédéric Frenet (February 7, 1816 – June 12, 1900) Jean Freédéric Frenet was a French mathematician, who is best remembered for the Serret-Frenet formulas for a space-curve. These formulas have been discovered by Jean Freédéric Frenet, in his thesis of 1847, and Joseph Alfred Serret in 1851.

$$\mathbf{t}(s) = \mathbf{x}'(s), \qquad \mathbf{n}(s) = \frac{\mathbf{x}''(s)}{||\mathbf{x}''(s)||}, \qquad \mathbf{b}(s) = \mathbf{t}(s) \times \mathbf{n}(s). \tag{3.71}$$

For a curve which is not parametrized with respect to its arc length the following equivalent formulas hold:

$$\mathbf{t}(u) = \frac{\dot{\mathbf{x}}(u)}{||\dot{\mathbf{x}}(u)||} \tag{3.72}$$

$$\mathbf{b}(u) = \frac{\dot{\mathbf{x}}(u) \times \ddot{\mathbf{x}}(u)}{||\dot{\mathbf{x}}(u) \times \ddot{\mathbf{x}}(u)||} \tag{3.73}$$

$$\mathbf{n}(u) = \mathbf{b}(u) \times \mathbf{t}(u) \tag{3.74}$$

Figure 3.9 shows an illustrative example. These three vectors span three planes which pass $X(s)$. The osculating plane at the $X(s)$ is the plane spanned by the tangent vector **t** and the normal vector **n**; the normal plane is spanned by the normal vector **n** and the binormal vector **b**; the rectifying plane is spanned by the tangent vector **t** and the binormal vector **b**.

The curvature $\kappa(s)$ of a regular C^2-curve $c : \mathbf{x}(s), s \in I$ is defined via the second derivation

$$\kappa(s) = ||\mathbf{x}''(s)|| = \frac{||\dot{\mathbf{x}}(u) \times \ddot{\mathbf{x}}(u)||}{||\dot{\mathbf{x}}(u)||^3}, \tag{3.75}$$

whereas its torsion can be expressed via the third derivation

$$\tau(s) = \frac{|\mathbf{x}'(s)\, \mathbf{x}''(s)\, \mathbf{x}'''(s)|}{\mathbf{x}''(s)^2} = \frac{|\dot{\mathbf{x}}(u)\, \ddot{\mathbf{x}}(u)\, \dddot{\mathbf{x}}(u)|}{(\dot{\mathbf{x}}(u) \times \ddot{\mathbf{x}}(u))^2} \tag{3.76}$$

using the scalar triple product of three vectors

$$|\mathbf{a}\,\mathbf{b}\,\mathbf{c}| \equiv \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{b} \cdot (\mathbf{c} \times \mathbf{a}) = \mathbf{c} \cdot (\mathbf{a} \times \mathbf{b}) = \det(\mathbf{a}\,\mathbf{b}\,\mathbf{c}). \tag{3.77}$$

The determinant expression shows that the scalar triple product is a pseudoscalar as it reverses the sign under inversion.

3.5.5 Frenet Formulas

The inherent properties of a parametrized curve – namely curvature and torsion – and the vectors $\mathbf{t}(s)$, $\mathbf{n}(s)$, $\mathbf{b}(s)$ meet differential equations, which can be written in matrix form

$$\begin{array}{lcccc} \mathbf{t}'(s) & = & & \kappa(s) \cdot \mathbf{n}(s) & \\ \mathbf{n}'(s) & = & -\kappa(s) \cdot \mathbf{t}(s) & & +\tau(s) \cdot \mathbf{b}(s) \\ \mathbf{b}'(s) & = & & -\tau(s) \cdot \mathbf{n}(s) & \end{array} \tag{3.78}$$

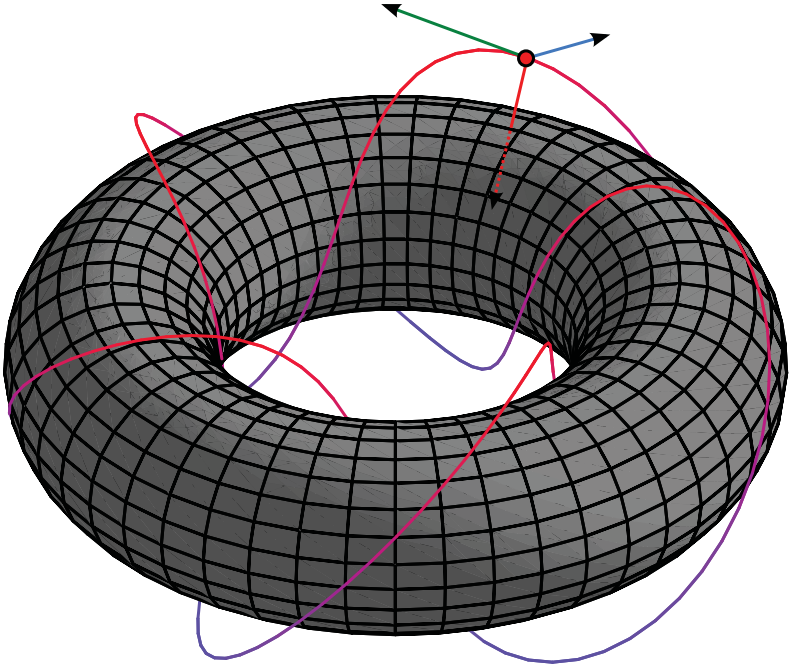


Figure 3.9: The curve $X(u) = ((8+3\cos 5u)\cdot\cos 2u \mid (8+3\cos 5u)\cdot\sin 2u \mid 5\sin(5u))$ with parameter domain $u \in [-\pi,\pi]$ is a so-call (2,5)-torus knot. A (p,q)-torus knot is a closed space curve that is looped through the hole of a torus p times with q revolutions [Gra97]. The free parameters p and q have to be relatively prime. The curve lies on an elliptical torus. For illustration purposes the circular torus along the minor axis is included.

The curve's tangent vector (green) indicates the flow direction. The normal respectively curvature vector points towards the center of the circle that approximates the curve in $X(u)$ best.

with a skew symmetric matrix

$$\begin{pmatrix} 0 & \kappa(s) & 0 \\ -\kappa(s) & 0 & +\tau(s) \\ 0 & -\tau(s) & 0 \end{pmatrix}. \tag{3.79}$$

As a consequence, a curve with curvature $\kappa \neq 0$ is planar, if and only if $\tau = 0$.

The fundamental theorem of space curves in $\mathbb{E}^3$ states that the curvature function $\kappa(s)$ and the torsion function $\tau(s)$ along a curve c determine the space curve c up to an orientation-preserving isometry.

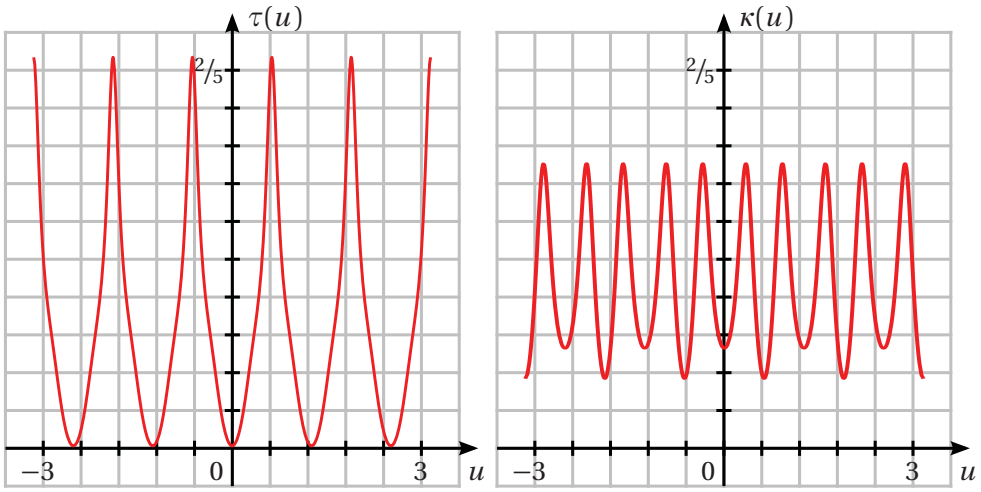


Figure 3.10: The periodicity of a (2,5)-torus knot (see Figure 3.9) is reflected in its torsion (left) and curvature (right) plots.

3.5.6 Differential Geometry of Surfaces

Analogous to the theory of curves and the previous definitions, a surface is defined over a region (an open, connected set) $G \subset \mathbb{R}^2$ by a C^r map ($r \geq 1$)

$$\varphi : \begin{cases} G & \to & \mathbb{E}^3 \\ (u,v) & \mapsto & X(u,v) \end{cases} \tag{3.80}$$

The region is sometimes called parameter domain. The point set

$$\Phi = \{X(u,v) : (u,v) \in G\} \tag{3.81}$$

is called C^r-surface with parameter representation φ. A surface point $X(u_0,v_0)$ is a regular point, if the vectors

$$\mathbf{x}_u(u_0,v_0) = \frac{\partial \mathbf{x}}{\partial u}(u_0,v_0) \qquad \mathbf{x}_v(u_0,v_0) = \frac{\partial \mathbf{x}}{\partial v}(u_0,v_0) \tag{3.82}$$

are linearly independent. Otherwise it is called singular point. The surface Φ is called regular with respect to φ, if all of its points are regular. Furthermore, Φ is a simple surface, if the map φ is an injection.

During the analysis of surfaces, curves are of particular importance. A parameter representation φ of a surface includes the definition of parameter lines, for which one parameter (u or v) is constant. More generalized, any C^r map

$$u : \left\{ \begin{array}{lcl} I \subset \mathbb{R} & \rightarrow & G \\ t & \mapsto & (u(t), v(t)) \end{array} \right. \tag{3.83}$$

into the parameter domain of a regular C^r-surface $\Phi : \mathbf{x}(u,v), (u,v) \in G$ with

$$\left(\frac{\mathrm{d}\, u}{\mathrm{d}\, t}(t), \frac{\mathrm{d}\, v}{\mathrm{d}\, t}(t) \right) \neq (0,0) \ \ \forall t \in I \tag{3.84}$$

defines a regular C^r curve – a surface curve – of Φ

$$c : \mathbf{y}(t) = \mathbf{x}(u(t), v(t)), \ \ t \in I. \tag{3.85}$$

The parameter lines are defined by the maps $u = t$ and $v = const$ respectively $u = const$ and $v = t$. The resulting surface curves can be used to define tangent vectors $\mathbf{x}_u, \mathbf{x}_v$ and tangent planes. For any regular C^1 surface $\Phi : \mathbf{x}(u,v), (u,v) \in G$ the plane

$$T(X(u,v), \Phi) = \left\{ Y \in \mathbb{E}^3 \ : \ \mathbf{y} = \mathbf{x}(u,v) + \lambda \mathbf{x}_u(u,v) + \mu \mathbf{x}_v(u,v); \ \lambda, \mu \in \mathbb{R} \right\} \tag{3.86}$$

at $X(u_v)$ is called tangent plane. For a regular, but not simple C^1-surface it is not possible to identify a unique tangent plane for a point $X(u,v) \in \Phi$ due to possible self-intersections of Φ. In this case the surface should be limited to a simple subsurface or the parameters (u,v) need to be specified.

The vector

$$\mathbf{n}(u,v) = \frac{\mathbf{x}_u(u,v) \times \mathbf{x}_v(u,v)}{||\mathbf{x}_u(u,v) \times \mathbf{x}_v(u,v)||} \tag{3.87}$$

is the so-called unit normal vector of Φ at $X(u,v)$. As the vector is normalized

$$< \mathbf{n}(u,v) | \mathbf{n}(u,v) > \equiv 1, \tag{3.88}$$

its partial derivatives hold

$$< \mathbf{n}_u(u,v) | \mathbf{n}(u,v) > = 0, \qquad < \mathbf{n}_v(u,v) | \mathbf{n}(u,v) > = 0, \tag{3.89}$$

i.e. the vectors $\mathbf{n}_u(u,v)$ and $\mathbf{n}_v(u,v)$ are parallel to the tangent plane $T(X(u,v), \Phi)$.

If $\Phi : \mathbf{x}(u,v), (u,v) \in D$ is a simple C^1-surface and $c \subset \Phi$ a regular curve containing the point $X \in \Phi$, then the tangent vector of c in X is called surface vector and the tangent of c in X is a surface tangent of Φ in X. These definitions are illustrated in Figure 3.11.

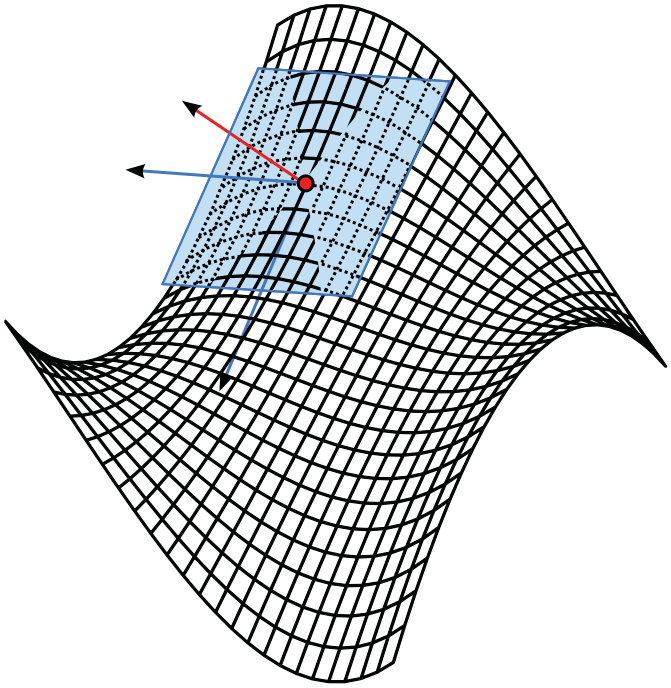


Figure 3.11: The parametrization of a surface defines two families of surface curves: the parameter lines. These surface curves (black) are often used to generate so-called wireframe representations of a surface. Furthermore, the parametrization defines the tangent vectors (blue). The corresponding tangent plane and its associated normal vector (red) do not depend on the surface's parametrization.

3.5.7 Change of Parameter

Just like curves, surfaces may have different parametrizations. The process of switching between different parametrizations is a change of parameter.

Let $\Phi : \mathbf{x}(u,v),(u,v) \in G$ be a regular C^1-surface and $R \subset \mathbb{R}^2$ a region. A surjective C^r-map

$$f : \begin{cases} R & \to & G \\ (r,s) & \mapsto & (u(r,s),v(r,s)) \end{cases} \tag{3.90}$$

is an allowable C^r-change of parameter, if the determinant of functionals

$$\begin{vmatrix} \frac{\partial u}{\partial r}(r,s) & \frac{\partial u}{\partial s}(r,s) \\ \frac{\partial v}{\partial r}(r,s) & \frac{\partial v}{\partial s}(r,s) \end{vmatrix} \tag{3.91}$$

is not zero for any $(r,s) \in R$. If the determinant is always positive, the map f is an orientation-preserving change of parameter, otherwise not.

3.5.8 Fundamental Forms

The three fundamental forms of a surface determine its metric properties – namely line element, area element, curvature. As the third fundamental form can be expressed in terms of the first and second one, it is less important.

3.5.9 First Fundamental Form

If $\Phi : \mathbf{x}(u,v),(u,v) \in G$ is a regular C^1-surface, then the coefficients

$$g_{uu}(u,v) = <\mathbf{x}_u(u,v)|\mathbf{x}_u(u,v)> \qquad g_{uv}(u,v) = <\mathbf{x}_u(u,v)|\mathbf{x}_v(u,v)> \tag{3.92}$$

$$g_{vu}(u,v) = <\mathbf{x}_v(u,v)|\mathbf{x}_u(u,v)> \qquad g_{vv}(u,v) = <\mathbf{x}_v(u,v)|\mathbf{x}_v(u,v)> \tag{3.93}$$

exist and are called first fundamental form coefficients. Due to the symmetry of a scalar product the coefficients g_{uv} and g_{vu} are equal. The determinant of these coefficients is

$$g = \begin{vmatrix} g_{uu} & g_{uv} \\ g_{vu} & g_{vv} \end{vmatrix} = g_{uu}g_{vv} - g_{uv}^2 = (\mathbf{x}_u \times \mathbf{x}_v)^2. \tag{3.94}$$

With these coefficients the first fundamental form can be defined: Let $\Phi : \mathbf{x}(u,v)$, $(u,v) \in D$ be a simple C^1-surface. A surface vector in tangent space of Φ at point $X \in \Phi$ has the representation

$$\mathbf{a} = a_u\mathbf{x}_u + a_v\mathbf{x}_v. \tag{3.95}$$

Then the first fundamental form[5] is

$$\mathrm{I}(\mathbf{a}) = g_{uu}a_u a_u + g_{uv}a_u a_v + g_{vu}a_v a_u + g_{vv}a_v a_v. \tag{3.96}$$

The coefficients g_{uu}, $g_{uv} = g_{vu}$, g_{vv} depend on the surface's parametrization in contrast to the first fundamental form. As it maps each surface vector $\mathbf{a}$ to the square of its length $\mathrm{I}(\mathbf{a}) = \mathbf{a}^2$, the first fundamental form is independent of the parametrization of the surface Φ.

[5] In differential geometry the Einstein[6]notation is used. The parameters of a surface are written with indices $X(x^1,x^2)$ instead of $X(u,v)$. The tangent vectors are written the same way $\mathbf{x}_u$, $\mathbf{x}_v$. The coefficients of the first fundamental form are then $g_{ij} = <\mathbf{x}_i|\mathbf{x}_j>$ $(i,j = 1,2)$. According to the Einstein summation convention, the summation sign can be omitted, if an index variable appears in superscript as well as in subscript position. It implies that the summation comprises all possible values. The first fundamental form is then abbreviated $\mathrm{I}(\mathbf{a}) = \sum_{i=1}^{2}\sum_{j=1}^{2} g_{ij}a^i a^j = g_{ij}a^i a^j$.

[6] Albert Einstein (March 14, 1879 – April 18, 1955) Albert Einstein was a theoretical physicist. He is best known for his theories of special relativity and general relativity. In 1921/1922 he received the Nobel Prize in Physics.

A change of parameter

$$f:\left\{\begin{array}{lcl} R & \to & D \\ (r,s) & \mapsto & (u(r,s),v(r,s)) \end{array}\right. \tag{3.97}$$

transforms the coefficients of the first fundamental form according to the chain rule to $\overline{g}_{uu}, \overline{g}_{uv}, \overline{g}_{vu}, \overline{g}_{vv}$:

$$\begin{pmatrix} \overline{g}_{uu} & \overline{g}_{uv} \\ \overline{g}_{vu} & \overline{g}_{vv} \end{pmatrix} = \mathbf{M}^{\mathrm{T}} \begin{pmatrix} g_{uu} & g_{uv} \\ g_{vu} & g_{vv} \end{pmatrix} \mathbf{M}, \tag{3.98}$$

whereas $\mathbf{M}$ denotes the fundamental matrix of the parameter transformation.

The first fundamental form of a surface can be used to perform length and angle calculations on a regular C^1-surface $\Phi : \mathbf{x}(u,v),(u,v) \in G$:

1. The arc length s of a curve $c \subset \Phi$ defined by parameter $(u(t),v(t))$, $t \in I$ between a and b $(a,b \in I)$ is

$$s(b) - s(a) = \int_a^b ||\dot{\mathbf{x}}(t)|| \,\mathrm{d}\, t \tag{3.99}$$

$$= \int_a^b \sqrt{\mathrm{I}\left(\mathbf{x}_u \frac{\mathrm{d}\, u}{\mathrm{d}\, t} + \mathbf{x}_v \frac{\mathrm{d}\, v}{\mathrm{d}\, t}\right)} \,\mathrm{d}\, t. \tag{3.100}$$

2. The angle α between two surface vectors $\mathbf{a} = a_u \mathbf{x}_u + a_v \mathbf{x}_v$ and $\mathbf{b} = b_u \mathbf{x}_u + b_v \mathbf{x}_v$ at $X(u,v)$ can be calculated via

$$\cos\alpha = \frac{\begin{pmatrix} a_u \\ a_v \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} g_{uu} & g_{uv} \\ g_{vu} & g_{vv} \end{pmatrix} \begin{pmatrix} b_u \\ b_v \end{pmatrix}}{\sqrt{\begin{pmatrix} a_u \\ a_v \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} g_{uu} & g_{uv} \\ g_{vu} & g_{vv} \end{pmatrix} \begin{pmatrix} a_u \\ a_v \end{pmatrix}} \sqrt{\begin{pmatrix} b_u \\ b_v \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} g_{uu} & g_{uv} \\ g_{vu} & g_{vv} \end{pmatrix} \begin{pmatrix} b_u \\ b_v \end{pmatrix}}}. \tag{3.101}$$

3. The area of a surface patch $\Psi = \{X(u,v) : (u,v) \in P\}$ with closed, bounded parameter domain $P \subset D$ is

$$A(\Psi) = \iint_P \sqrt{g(u,v)} \,\mathrm{d}\, u \,\mathrm{d}\, v. \tag{3.102}$$

Metric properties of maps between surfaces can also be described by the first fundamental form coefficients. If $\Phi : \mathbf{x}(u,v)$ and $\Phi^\star : \mathbf{x}^\star(u,v)$ are two C^1-surfaces with the same parameter domain G and if

$$\alpha:\left\{\begin{array}{lcl} \Phi & \to & \Phi^\star \\ X(u,v) & \mapsto & X^\star(u,v) \end{array}\right. \tag{3.103}$$

is a function which maps points with equal parameters (u,v) to each other, then α has the following metric properties:

1. α is length-preserving (i.e. a curve segment and its α-mapped image have the same length), if and only if for all $(u,v) \in D$ the first fundamental form coefficients are equal

$$g_{uu}(u,v) = g^{\star}_{uu}(u,v), \qquad g_{uv}(u,v) = g^{\star}_{uv}(u,v), \tag{3.104}$$
$$g_{vu}(u,v) = g^{\star}_{vu}(u,v), \qquad g_{vv}(u,v) = g^{\star}_{vv}(u,v). \tag{3.105}$$

2. α is angle-preserving (i.e. the angle between a pair of intersecting curves at the intersection point and the angle between the curves' images are equal), if and only if the first fundamental form coefficients

$$g_{uu}(u,v) = \lambda(u,v) g^{\star}_{uu}(u,v), \qquad g_{uv}(u,v) = \lambda(u,v) g^{\star}_{uv}(u,v), \tag{3.106}$$
$$g_{vu}(u,v) = \lambda(u,v) g^{\star}_{vu}(u,v), \qquad g_{vv}(u,v) = \lambda(u,v) g^{\star}_{vv}(u,v). \tag{3.107}$$

   are equal for all $(u,v) \in D$ up to a function $\lambda : D \to \mathbb{R}$.

3. α is area-preserving (i.e. a surface patch and its α-image have the same area), if and only if the determinant of the first fundamental form determinants (see Equation (3.94)) are equal

$$g(u,v) = g^{\star}(u,v) \tag{3.108}$$

   for all $(u,v) \in D$.

Map Projections

Geography has been an important field of differential geometry applications for over 2 000 years [Sny97]. As maps and atlases are omnipresent, they are a good example to illustrate basic concepts of differential geometry.

To simplify matters the earth is regarded as a sphere and the unit length is set to the earth's mean radius 6 371 km. In this way the earth's surface has the parametrization of a unit sphere

$$\Sigma : \mathbf{x}(u,v) = \begin{pmatrix} \cos(v)\cdot\cos(u) \\ \cos(v)\cdot\sin(u) \\ \sin(v) \end{pmatrix} \tag{3.109}$$

over the domain $[-\pi,\pi] \times [-\pi/2,\pi/2]$. To flatten earth the greek mathematician ARCHIMEDES[7] proposed a cylindrical projection [Pea90] which maps each point on earth to the intersection of a cylinder, which is tangent to the equator, and the perpendicular to earth's rotation axis through the point to map. This mapping is illustrated in Figure 3.12. Using the same parameter domain the cylinder may have the parametrization

$$\Psi : \mathbf{x}^\star(u,v) = \begin{pmatrix} \cos(u) \\ \sin(u) \\ \sin(v) \end{pmatrix}. \tag{3.110}$$

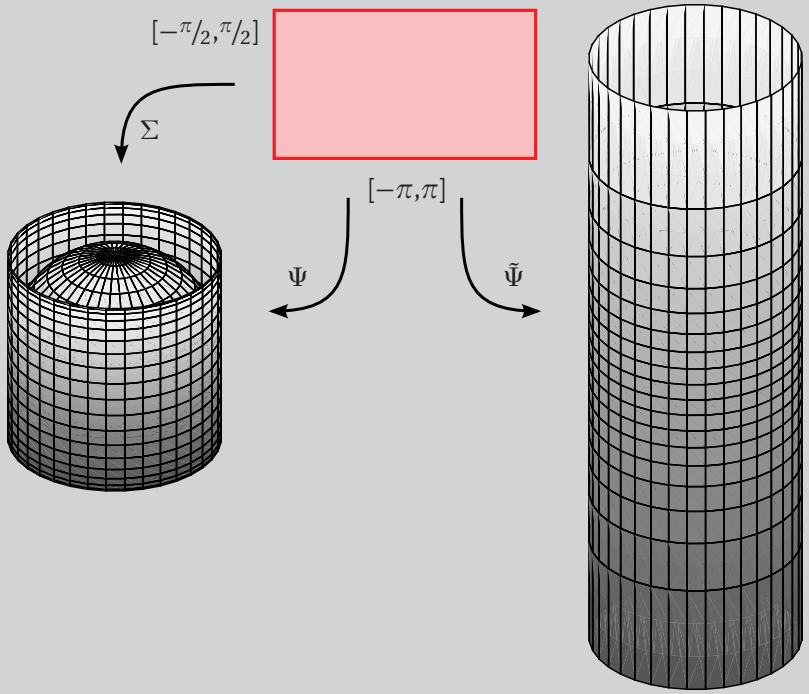


Figure 3.12: The earth can be parametrized as a unit sphere over the domain $[-\pi,\pi] \times [-\pi/2,\pi/2]$. The projection by ARCHIMEDES maps all points on the sphere to points on a cylinder (see Figure 3.13). This kind of map is area-preserving. About 1 800 years later MERCATOR modified this mapping to create the first angle-preserving map of earth (see Figure 3.14) – a valuable tool in navigation.

An analysis of both surfaces reveals the coefficients of the first fundamental form:

$$g_{uu} = \cos^2 v, \tag{3.111}$$
$$g_{uv} = g_{vu} = 0, \tag{3.112}$$
$$g_{vv} = 1, \tag{3.113}$$

and

$$g^\star_{uu} = 1, \tag{3.114}$$
$$g^\star_{uv} = g^\star_{vu} = 0, \tag{3.115}$$
$$g^\star_{vv} = \cos^2 v \tag{3.116}$$

for Σ and Ψ respectively. As the determinants are equal $g_{uu}g_{vv} - g_{uv}g_{vu} = g^\star_{uu}g^\star_{vv} - g^\star_{uv}g^\star_{vu} = \cos^2 v$ the ARCHIMEDEAN map is area-preserving (see Equation (3.108)).

For navigation purposes an angle-preserving map is more important than an area-preserving one. Unfortunately, the condition for angle-preserving mappings is not met: There is no function $\lambda(u,v)$, which fulfills the equations

$$g_{uu} = \lambda(u,v)g^\star_{uu}, \tag{3.117}$$
$$g_{uv} = \lambda(u,v)g^\star_{uv}, \tag{3.118}$$
$$g_{vu} = \lambda(u,v)g^\star_{vu}, \tag{3.119}$$
$$g_{vv} = \lambda(u,v)g^\star_{vv}. \tag{3.120}$$

Figure 3.13: This ARCHIMEDEAN map of the earth preserves areas; i.e. if two regions have the same size, then their images also have the same size.

In 1569 the cartographer GERARDUS MERCATOR[8] presented the first world map that is angle-preserving. He introduced a modified, cylindrical mapping

$$\tilde{\Psi} : \mathbf{x}^\star(u,v) = \begin{pmatrix} \cos(u) \\ \sin(u) \\ \ln\tan(\frac{v}{2} + \frac{\pi}{4}) \end{pmatrix} \tag{3.121}$$

whose coefficients $\tilde{g}_{uu}, \ldots, \tilde{g}_{vv}$ meet the conditions (Equation (3.117) – (3.120)) for angle-preserving maps. As the poles are mapped to infinity a MERCATOR map is clipped along a circle of latitude $< 90°$. The construction is illustrated in Figure 3.12 whereas the result is shown in Figure 3.14. A drawback of the result is its area distortion. It exaggerates the size of areas far from the equator.

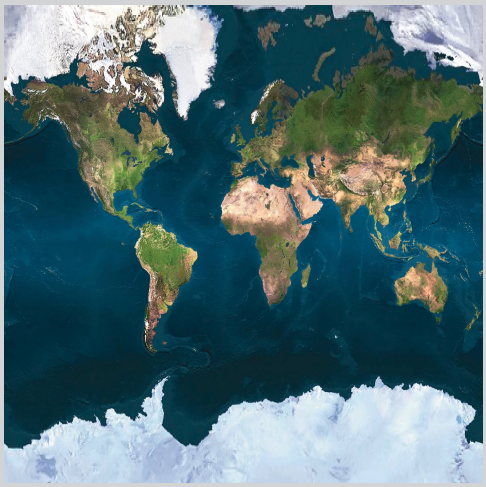


Figure 3.14: The MERCATOR map is an angle preserving map; i.e. angles measured in real world and measured in the map are equal. This is an important property for navigation.

3.5.10 Second Fundamental Form

For any regular C^2-surface $\Phi : \mathbf{x}(u,v),(u,v) \in D$ the product of the second derivatives with the normal vector can be calculated:

$$h_{uu}(u,v) = <\mathbf{x}_{uu}(u,v)|\mathbf{n}(u,v)> \tag{3.122}$$
$$h_{uv}(u,v) = <\mathbf{x}_{uv}(u,v)|\mathbf{n}(u,v)> \tag{3.123}$$
$$h_{vu}(u,v) = <\mathbf{x}_{vu}(u,v)|\mathbf{n}(u,v)> \tag{3.124}$$
$$h_{vv}(u,v) = <\mathbf{x}_{vv}(u,v)|\mathbf{n}(u,v)> \tag{3.125}$$

These are the functions of the second fundamental form of Φ. Their determinant is denoted h. Another possibility to calculate h_{ij} for any combination uu, uv, vu and vv is

$$h_{ij} = \frac{1}{\sqrt{g}} \left| \mathbf{x}_{ij}\ \mathbf{x}_u\ \mathbf{x}_v \right|. \tag{3.126}$$

Analogue to the definition of the first fundamental form, the second fundamental form is defined for a surface vector $\mathbf{a} = a_u\mathbf{x}_u + a_v\mathbf{x}_v$ of a surface Φ in $X \in \Phi$:

$$\mathrm{II}(\mathbf{a}) = h_{uu}a_u a_u + h_{uv}a_u a_v + h_{vu}a_v a_u + h_{vv}a_v a_v. \tag{3.127}$$

Under an orientation-preserving change of parameter the second fundamental form behaves the same way as the first fundamental form. Therefore, it is also invariant to these parameter changes.

To analyze a surface with respect to curvature it is convenient to start with the curvature of surface curves. If $\Phi : \mathbf{x}(u,v),(u,v) \in D$ is a simple C^2-surface and $c \subset \Phi$ a regular C^2 curve with signed curvature κ in $X(u^1(t_0),u^2(t_0)) \in c$, then the so-called normal curvature κ_n of c in X is

$$\kappa_n = \kappa \cos\theta = \kappa\ <\mathbf{n}_\Phi|\mathbf{n}_c> \tag{3.128}$$

with angle θ between the unit normal vector $\mathbf{n}_\Phi$ of Φ and the curvature vector $\mathbf{n}_c$ of c in X. If $\mathbf{h}$ does not exist, κ_n shall be zero.

It is obvious that all regular C^2 curves of a simple C^2-surface, which intersect in one point and which have the same tangent line in this point, have the same normal curvature. The maximum and minimum of all possible values of κ_n are called principal curvatures κ_1, κ_2. The corresponding tangents are the principal curvature tangents. Figure 3.15 illustrates this setting.

[7] Archimedes of Syracuse (287 BC – 212 BC) Archimedes was a Greek mathematician, physicist, engineer, inventor, and astronomer. Although few details of his life are known, he is generally considered to be the greatest mathematician of antiquity.

[8] Gerardus Mercator (March 5, 1512 – December 2, 1594) Gerardus Mercator was a Flemish cartographer. He is remembered for the Mercator projection world map, which became the standard map projection for nautical purposes because of its ability to represent lines of constant course, known as rhumb lines or loxodromes, as straight segments.

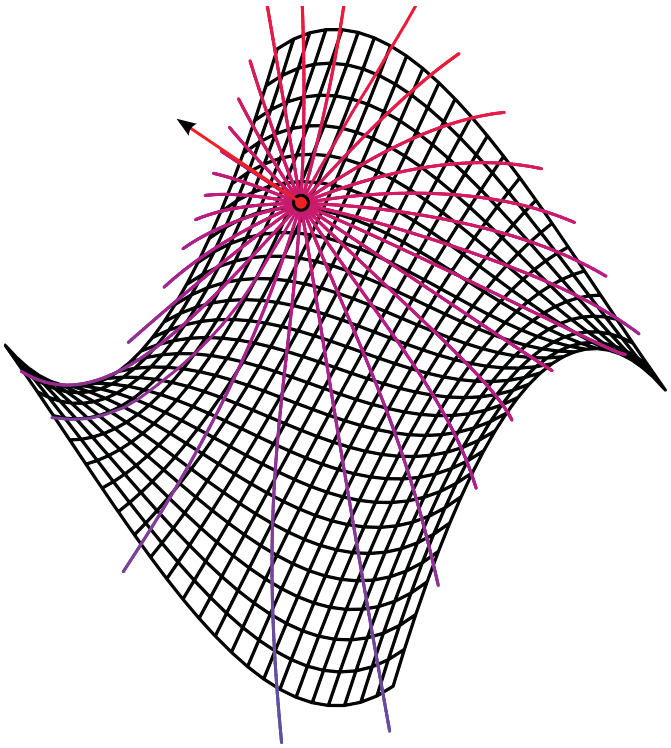


Figure 3.15: The intersection of all planes, which contain a point P and its surface normal (red) is a family of curves. All curves pass P and be ordered by increasing angles α to a surface tangent. If each curve has a signed curvature $\kappa(\alpha)$, then maximum and minimum values can be determined. These values are the principal curvatures κ_1 and κ_2.

3.5.11 Euler Curvature Formula

On a simple C^2-surface $\Phi : \mathbf{x}(u^1,u^2),(u^1,u^2) \in G$ the normal curvature κ_n in a point $X \in \Phi$ in direction $\mathbf{t}$ can be calculated via

$$\kappa_n = \kappa_1 \cos^2\theta + \kappa_2 \sin^2\theta, \tag{3.129}$$

where θ is the angle between $\mathbf{t}$ and the principal curvature tangent that corresponds to κ_1. This theorem by EULER[9] does not apply for umbilic points, i.e. points on a surface at which the curvature is the same in any direction, and for points with vanishing functions of the second fundamental form, i.e. $h_{uu} = h_{uv} = h_{vu} = h_{vv} = 0$.

[9] LEONHARD PAUL EULER (April 15, 1707 – September 18, 1783) Leonhard Euler was a Swiss mathematician who made enormous contributions to a wide range of mathematics and physics including analytic geometry, trigonometry, geometry, calculus and number theory.

3.5.12 Mean Curvature and Gaussian Curvature

Let κ_1 and κ_2 be the principal curvatures, then their mean

$$H = \frac{1}{2}(\kappa_1 + \kappa_2) \tag{3.130}$$

is called the mean curvature. The product

$$K = \kappa_1 \cdot \kappa_2. \tag{3.131}$$

is named after CARL FRIEDRICH GAUSS[10] The Gaussian curvature K and mean curvature H satisfy

$$H^2 \geq K, \tag{3.132}$$

with equality only at umbilic points, since

$$H^2 - K = \frac{1}{4}(\kappa_1 - \kappa_2)^2. \tag{3.133}$$

The mean and Gaussian curvatures can be calculated directly. The formulas

$$K = \frac{h}{g} \tag{3.134}$$

and

$$H = \frac{1}{2}(g^{\mathrm{I}}_{uu} h_{uu} + g^{\mathrm{I}}_{uv} h_{uv} + g^{\mathrm{I}}_{vu} h_{vu} + g^{\mathrm{I}}_{vv} h_{vv}) \tag{3.135}$$

with

$$\begin{pmatrix} g^{\mathrm{I}}_{uu} & g^{\mathrm{I}}_{uv} \\ g^{\mathrm{I}}_{vu} & g^{\mathrm{I}}_{vv} \end{pmatrix} = \begin{pmatrix} g_{uu} & g_{uv} \\ g_{vu} & g_{vv} \end{pmatrix}^{-1} = \frac{1}{g} \begin{pmatrix} g_{vv} & -g_{uv} \\ -g_{vu} & g_{uu} \end{pmatrix}. \tag{3.136}$$

According to Gauss's THEOREMA EGREGIUM the Gaussian curvature of a simple surface does only depend on the coefficients of the first fundamental form (including their derivatives).

[10] JOHANN CARL FRIEDRICH GAUSS (April 30, 1777 – February 23, 1855) Carl Friedrich Gauß was a German mathematician who worked in a wide variety of fields in both mathematics and physics including number theory, analysis, differential geometry, geodesy, magnetism, astronomy and optics. His work has had an immense influence in many areas.

Local Shape of a Surface

The Gaussian curvature $K = \kappa_1 \cdot \kappa_2$ characterizes the local shape of a surface.

principal curvature	local shape
$K < 0$	hyperbolic point
$K > 0$	elliptic point
$K = 0$, but κ_1 or κ_2 not zero	parabolic point

Table 3.2: All nonplanar points can be characterized according to the local shape of the surrounding surface.

Curvature on Discrete Structures

A large number of applications in computer graphics and computer-aided design exists that require an accurate estimation of differential-geometrical quantities – such as principal direction, mean curvature, etc. – at arbitrary vertices on a triangulated surface.

As differential geometry operates on continuous manifolds and surfaces [DC76], its results need to be discretized to be applicable. There are many methods for approximating differential-geometrical quantities of an underlaying surface based on its triangulation [BSSZ08]. If all methods are grouped according to their concept, two clusters can be identified.

One possibility to determine differential-geometrical quantities using a triangulated surface is to combine several theorems on curvature and to discretize the result. In this way MARK MEYER et al. [MDSB03] derived a discrete mean curvature normal operator based on the integral representation of mean curvature

$$H = \frac{1}{2\pi} \int_0^{2\pi} \kappa_n(\theta)\, \mathrm{d}\theta \qquad (3.137)$$

and the Euler-Lagrange equation for surface area minimization. This equation relates the area minimization and the mean curvature flow:

$$2H \cdot \mathbf{n} = \lim_{\mathrm{diam}(A)\to 0} \frac{\nabla A}{A}, \qquad (3.138)$$

where A is an infinitesimal area around a point P on the surface, $\mathrm{diam}(A)$ is its diameter, and ∇A is the gradient with respect to the coordinates of P.

The resulting mean curvature normal operator $\mathbf{K}$ is

$$\mathbf{K}(\mathbf{x}_i) = \frac{1}{2 \cdot A} \sum_{j \in N_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij}) \cdot (\mathbf{x}_i - \mathbf{x}_j) \qquad (3.139)$$

with $N_1(i)$ is the set of indices of 1-ring neighborhood of the i^{th}vertex. The angles α_{ij}, β_{ij} and the Voronoi-based calculation of the area A around the vertex $\mathbf{x}_i$ are illustrated in Figure 3.16.

The operator $\mathbf{K}$ returns an approximation of the surface normal vector $\mathbf{n}$ in $\mathbf{x}_i$. Its magnitude is twice the mean curvature value H in $\mathbf{x}_i$.

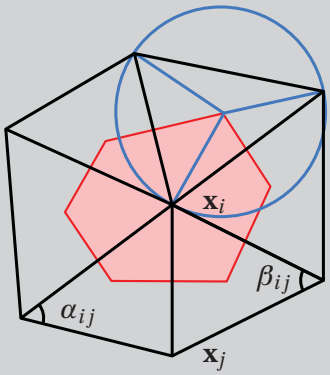


Figure 3.16: The mean curvature normal operator $\mathbf{K}$ is presented in "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds" [MDSB03] . It uses a Voronoi-based construction in the 1-neighborhood of a vertex $\mathbf{x}_i$ to approximate the mean curvature H.

In this way many operators for differential-geometrical quantities have been derived.

The second cluster of methods to calculate curvature and principal directions uses scattered data interpolation and fitting techniques [DB02]. JACK GOLDFEATHER and VICTORIA INTERRANTE developed such an algorithm [GI04]. For each point of interest $\mathbf{p}_i$ its k-nearest neighbors are transformed into a coordinate system whose origin is $\mathbf{p}_i$. In the next step, a $2^1/_2$-d surface

$$f(x,y) = \frac{A}{2}x^2 + Bxy + \frac{C}{2}y^2 + Dx^3 + Ex^2y + Fxy^2 + Gy^3 \tag{3.140}$$

is fitted to the points and their normals. Having determined the best coefficients to approximate the points and their normals, all differential-geometrical quantities can be calculated based on the continuous approximation of the underlaying surface. As in the local coordinate system the curvature only depends on the second degree terms with the coefficients A, B, C. For example, the mean curvature of f in (0,0) is

$$H = \frac{A+C}{2} \tag{3.141}$$

whereas the Gaussian curvature is

$$K = AC - B^2. \tag{3.142}$$

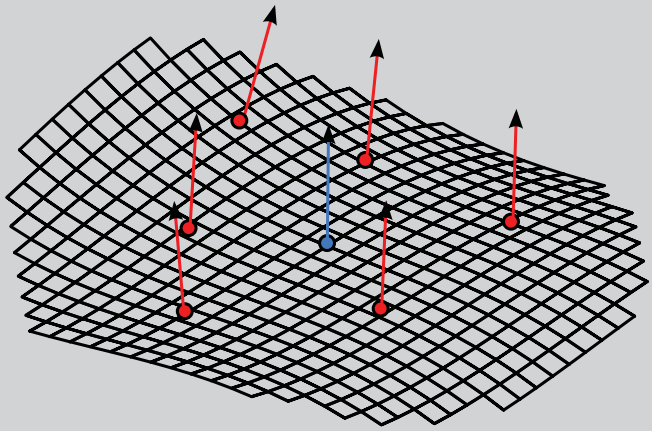


Figure 3.17: The cubic order algorithm for approximating principal curvature vectors by JACK GOLDFEATHER and VICTORIA INTERRANTE [GI04] calculates a new coordinate system, in which the vertex whose principal curvature vectors shall be calculated (blue) is the origin. Its k-nearest neighbors are also transformed into the new coordinate system. Then a surface f is fitted to approximate the underlaying surface of the samples. Then approximation f is used to calculate differential-geometrical quantities (principal curvature vectors, Gaussian curvature, etc.).

4 Computer-Aided Geometric Design

Using the geometric concepts presented in the previous chapter, computer-aided geometric design is concerned with 3D object representation and manipulation. Recent 3D object descriptions which are used in popular modeling approaches are based for example on points [ZPKG02], polygonal meshes [JLM02], NURBS-Patches [PT97], subdivision surfaces [Ma05], etc.

These descriptions can be ordered by their "description level". As a rule of thumb the more geometric primitives are needed to represent an object, the lower the "description level" of a representation type.

As a representative of a low level description the first part of this chapter analyzes heightfields whereas the second part concentrates on subdivision surfaces. Both object representations will be inspected with respect to distance calculation and collision detection – mathematical subproblems of semantic enrichment and geometric reconstruction. The creation of such CAD objects is called modeling. The modeling process – interactive and generative – will finalize this chapter.

Contents

4.1 Heightfields and Polygonal Surfaces

Heightfield data sets are required in various applications – for example to encode terrain models. Digital terrain models can be represented in two forms, as a triangulated irregular network and as a rectangular height grid. With the acquisition by airborne light detection and ranging (LIDAR) or stereo-photogrammetry especially the latter are available in good resolutions[1]. Triangular irregular networks can be generated from rectangular height grids by simplification for compact storage and for efficient rendering by computer graphics. The distance measure used for simplification is of special importance for the application [BMMKN04]. Many simulation and planning problems for electromagnetic wave propagation are based on the ray tracing model like signal strength prediction [FS97], [SW06] and antenna placement [ACN+05]. In these models, the main paths of wave propagation are evaluated by rays (geometric optics), and special wave effects are added at the intersection points of the rays with the terrain surface. Of course, the line-of-sights from the antennas account for the main propagation paths. So efficient line-of-sight (LOS) computation is an important means for terrain interrogation.

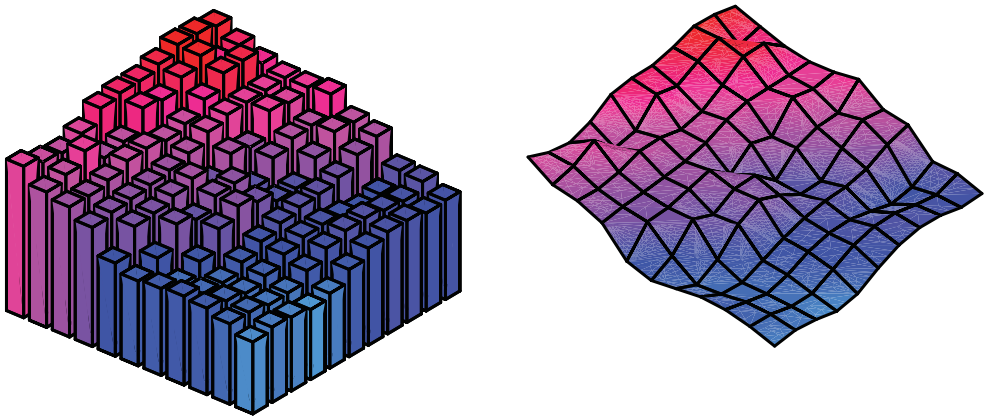


Figure 4.1: The values in a heightfield are acquired by measuring terrain heights with far-field photogrammetry or laser scanning systems [KBAP04]. From these discrete measurements, terrain surfaces can be reconstructed of various orders of continuity and of polynomial type. In this illustration both diagrams (left / right) show the same heightfield data. The question which one represents the terrain best for a special application is of considerable interest [KDS99].

[1] Earth Resources Observation and Science (EROS) Center (http://eros.usgs.gov/)

4.1.1 Line-Of-Sight Calculation

Basic data structures used for the line-of-sight computation problem are the grid structure and several tree-structured schemes. The grid structure is a sampled representation, directly storing height/distance values to a reference plane or reference surface. There have been several approaches of compressing the grid data, among them spatial hashing [THM+03] and wavelet transform [Yan00]. The kd-tree is a binary tree with splits cycling through the d dimensions. It has been invented for organizing point sets for nearest neighbor searching [Ben75] and stores data for rectangular spatial cells in its nodes. SARAH FRISKEN et al. [FPRJ00] use simultaneous splits of all dimensions resulting in a 2d-ary tree and store data for the cell corners. With bilinear interpolation on the space region, they represent a continuous scalar function. SYLVAIN LEFEBVRE and HUGUES HOPPE [LH07] covers the problem of encoding the tree topology and the tree data with a combination of techniques suitable for random access.

The most straightforward line-of-sight computation approach traverses the projection of the query ray on the domain plane and checks the ray's height against the heightfield's height at a number of points per cell [PG04]. An exact line-of-sight test requires a specific test, which depends on the terrain reconstruction and the field of application (see Figure 4.1). Possible reconstructions for non-integer grid positions include: Point reconstruction, double linear interpolation and bilinear interpolation. By checking a number p of discrete points per grid cell, large low-height regions cannot be exploited. The approach requires always $l \cdot p$ height evaluations regardless of the terrain traversed, where l is the ray length in cells.

Lately, also graphics processing units (GPU) have been used to determine line-of-sight on a terrain [SGS+05], [TSHM05]. These approaches render both the terrain surface and the ray and test, whether all line fragments are above the terrain surface. If this is the case, the query ray is a line-of-sight up to the image resolution used for rendering and for terrain reconstruction. A special hardware occlusion test is used for counting the number of visible fragments.

There is also work on slightly extended visibility problems like computing the horizon for each grid point and direction sector [Max88], [RBB01]. Originally, the resulting horizon map was invented for self-shadowing the terrain surface. In "Fast Horizon Computation at All Points of a Terrain with Visibility and Shading Applications" [Ste98] the horizon map computation is optimized for a large number of points using an algorithm of runtime $O(s \cdot k(\log^2 k + s))$ for s horizon sectors per point on k points.

4.1.2 KD-Tree Ray Casting

As a ray is defined by an origin point O and a direction vector $\mathbf{d}$, a ray gives an implicit search order on the data domain. So if the data domain is partitioned into several parts, then these parts can be searched according to the ray. Line-of-sight computation is a problem of this kind. Additionally, the parameter interval of the ray inside a domain part is available during traversal.

The simplest partition tree for a two-dimensional domain is a binary tree with domain-orthogonal partitioning planes. The parameter interval $[0,\infty]$ is subdivided by the ray-plane intersection point at parameter

$$\lambda = \frac{\mathbf{n}\cdot(\mathbf{p}-\mathbf{o})}{\mathbf{n}\cdot\mathbf{d}} \tag{4.1}$$

into the near interval $[0,\lambda]$ and the far interval $[\lambda,\infty]$. The plane is defined by an arbitrary incident point $\mathbf{p}$ and a normal vector $\mathbf{n}$. For axis-orthogonal partitioning planes the ray-plane intersection computation

$$\lambda = (\mathbf{p}_x - \mathbf{o}_x)/\mathbf{d}_x \qquad \text{resp.} \qquad \lambda = (\mathbf{p}_y - \mathbf{o}_y)/\mathbf{d}_y \tag{4.2}$$

is simple and therefore especially fast to compute.

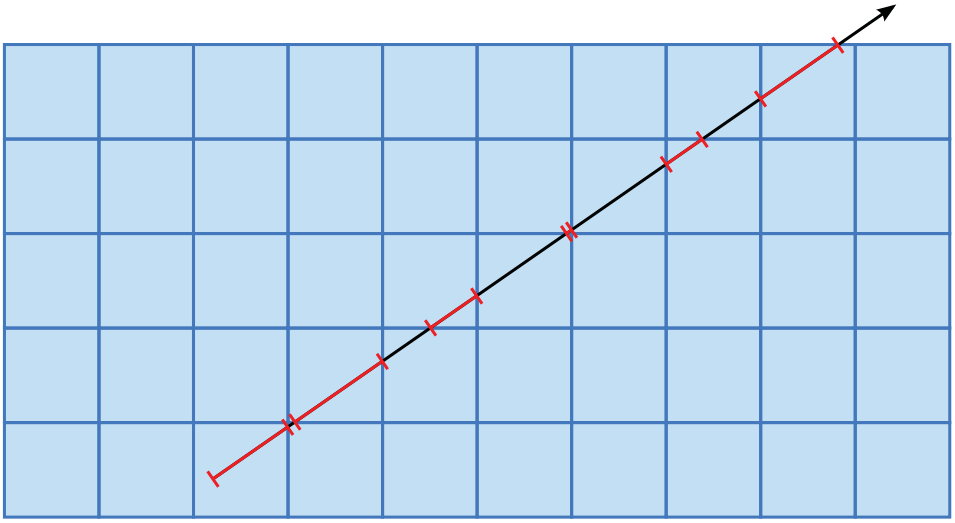


Figure 4.2: A partitioning structure like a binary tree subdivides a heightfield (blue) into small parts. It also subdivides a ray into several line segments (black / red).

The resulting partitioning tree with alternating x- and y-orthogonal planes is called kd-tree of dimension 2 [dBCvKO08]. The traversal of the kd-tree can be adapted for ray segments restricted to a parameter interval $[t_{\text{near}}, t_{\text{far}}]$. The traversal visits the near node, if and only if

$$[0,\lambda] \cap [t_{\text{near}}, t_{\text{far}}] = [t_{\text{near}}, \lambda] \tag{4.3}$$

is not empty and it visits the far node, if and only if

$$[\lambda,\infty] \cap [t_{\text{near}}, t_{\text{far}}] = [\lambda, t_{\text{far}}] \tag{4.4}$$

is not empty. This way the interesting parameter interval of the ray is always available for the current node. The special cases, where the ray is parallel to the partitioning plane ($\lambda = \infty$) or pointing away from the partitioning plane ($\lambda < 0$), are easy to handle.

So far the traversal of a 2-dimensional domain (the heightfield plane) by a kd-tree of 2 does not evaluate the height value. For a 2.5-dimensional heightfield, which is a real height function on the 2-dimensional domain, the approach can be extended. As the domain part represented by a kd-tree node shrinks with each subdivision, each leaf node of the tree represents a single entry of the heightfield grid. If each (inner) kd-tree node has the maximum height of the subtree it represents, it can be used to prune subtrees of the kd-tree from traversal. For a ray with parameter interval $[t_{\text{near}}, t_{\text{far}}]$ and height

$$z_{\text{near}} = \mathbf{o}_z + t_{\text{near}}\mathbf{d}_z \tag{4.5}$$

at entry and height

$$z_{\text{far}} = \mathbf{o}_z + t_{\text{far}}\mathbf{d}_z \tag{4.6}$$

at exit, it can intersect a node with maximum height $h_{\max}$ only if

$$z_{\text{near}} < h_{\max} \qquad \text{or} \qquad z_{\text{far}} < h_{\max}. \tag{4.7}$$

This simple extension requires only one additional real value per inner node of the kd-tree.

A heightfield uses discrete measurements and its terrain surface can be reconstructed in various ways. The question which one represents the terrain best for a special application has received considerable interest [KDS99], [PG04]. In "Empirical Comparison of Data Structures for Line-Of-Sight Computation" [FUFB07], [FUFB09] we have concentrated on point reconstruction which represents the surface as a constant within its grid cell and on bilinear reconstruction which is a degree-two surface with linear x-parallel or y-parallel intersections. Figure 4.3 shows plots of both reconstructions for samplings of the function

$$f(x,y) = -8x^3 - 12x^2 + 3xy^2 + 3y^2 + y^3, \qquad (x,y) \in [-2,2] \times [-2,2]. \tag{4.8}$$

Both reconstructions can be efficiently incorporated into a kd-tree. The key to efficiency here is that the grid environment required for the reconstruction is exactly available in one node of the kd-tree. For point reconstruction, the environment is small, 1×1. For bilinear reconstruction, it is larger, 2×2. This has some effect on the tree construction described below. For the intersection computation between the linear ray and the bilinear surface the intersection points with the 2×2 domain rectangle are calculated. The surface heights are added there by bilinear interpolation. From these values the parabola in the ray plane is uniquely determined. By equating the ray segment and the parabola the intersection problem can be solved. For intersection testing, is is computationally faster and more robust not to compute all possible (two) solutions but to compute the parameter value of the parabola apex and test the ray height against the apex height just there.

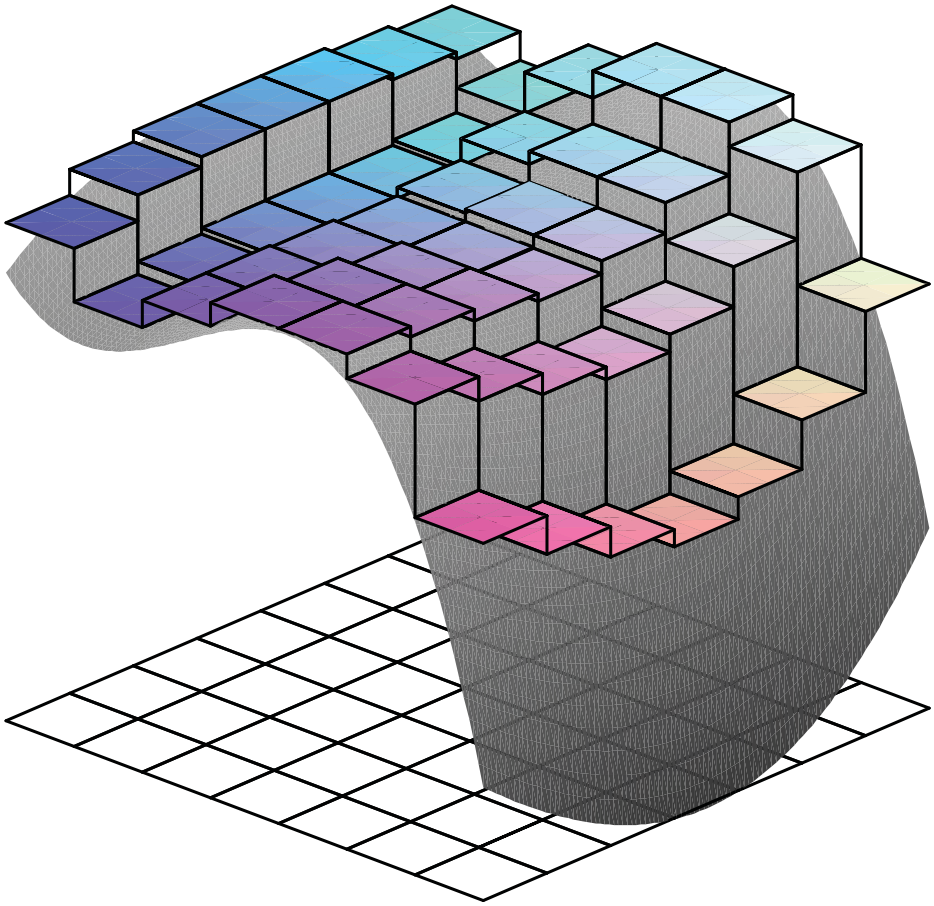


Figure 4.3: This data set has been generated by the function $f(x,y) = -8x^3 - 12x^2 + 3xy^2 + 3y^2 + y^3$ over the domain $[-2,2]^2$. The resulting samples are the input of a heightfield reconstruction. Different reconstruction types may have significant height differences as illustrated by the bilinear reconstruction (gray surface) and the maximum point reconstruction (colored quads). The question which one represents the terrain best has to be answered by the field of application [PG04] and has an important influence on the heightfield intersection test and the overall line-of-sight computation efficiency.

In principle, there are two approaches for tree construction, recursive top-down construction and iterative bottom-up construction [Ben75]. Its simplicity makes a recursive top-down construction especially impressive. It chooses a split index along a split axis which alternates between x- and y-axis. For a perfectly balanced tree, the split index is chosen at the center. A data-dependent tree construction could take the height values in the current kd-tree node into account. One could choose the split index

which creates two boxes of minimum sum of volumes. Due to the restriction of splitting planes to axis-aligned orientations, the separation of small and large heights is not so good, and the data-dependent construction is usually not worth it. For bilinear reconstruction, we process each value together with its left and lower neighbor (2×2 neighborhood). This can be achieved by creating a one row respectively one column overlap when splitting the current node (overlapped splitting). Compared to disjoint splitting, it generates four times as many nodes.

With disjoint splitting the height of the kd-tree for a heightfield of size $n \times m$ is $\log_2(n \cdot m)$, and the number of nodes is $2(n \cdot m)$. With overlapped splitting the height is $\log_2(n \cdot m)+2$ and the number of nodes is $5(n \cdot m)$. Please note, that the $n \cdot m$ leaf notes are reused in $3(n \cdot m)$ places for the overlap. The data per node consist of two pointers (4 bytes each on a 32-bit architecture), the real maximum height (8 bytes in double precision), and the real split value (8 bytes in double precision). So that the memory requirements sum up to 20 bytes per node. For comparison, the given heightfield array consists of $n \cdot m$ height values with 8 bytes per entry.

4.1.3 Nonstandard Decomposition in Max-Plus-Algebra

The wavelet-like, compressed grid for a heightfield of size $n \times m$ has the same memory requirements as the square grid of side length $\max(\tilde{n},\tilde{m})$ respectively $2 \cdot \max(\tilde{n},\tilde{m})$ with overlapped splitting whereas $\tilde{x}$ denotes the smallest power of two greater than or equal to x.

The inefficiency for non power-of-two and non-square formats can be eliminated for example by tiling techniques (as done in JPEG2000 [CS00] by the Joint Photographic Experts Group (JPEG)) or by a boolean sequence, which gives the orientation of the split, horizontal or vertical. In this way, the wavelet-like compressed grid is a different storage scheme for the kd-tree. But if sufficient memory is available, the square, power-of-two format with its implicit, 4-ary splitting is beneficial for caching and runtime efficiency.

Using the notation introduced in Section 2.1.10, the heightfield decomposition can be described by

$$c_k^{j-1} = \max(c_{2k}^j, c_{2k+1}^j) \tag{4.9}$$

and

$$d_k^{j-1} = c_{2k}^j - c_{2k+1}^j \tag{4.10}$$

using maximum coefficients c_k^j and the corresponding detail coefficients d_k^j. The reconstruction step is then calculated by

$$c_{2k}^{j+1} = c_k^j + \min(0, d_k^j) \tag{4.11}$$

and

$$c_{2k+1}^{j+1} = c_k^j - \max(0, d_k^j). \tag{4.12}$$

Figure 4.4 illustrates the results of a non-standard decomposition to a quadratic height-field of size 220×220. In the non-standard decomposition, the application of the filter alternates between columns and rows.

The analysis filter can be described in a short manner using operations in $\mathbb{R}_{\max}$ max-plus algebra [ACG+90]. In this way the filter process is very similar to one of Haar wavelets. As the algebra $\mathbb{R}_{\max}$ is only an idempotent semiring, the maximum filter does not satisfy the wavelet-filter definition.

4.1.4 Nonstandard Decomposition in Real Algebra

The previous section describes a wavelet-like decomposition where the maximum value of a region is directly available in the representation. Alternatively, a maximum computation is possible using a linear combination of scalar basis functions on the domain. The nonstandard approach of wavelet transformation to generate two-dimensional basis functions combines two one-dimensional basis functions in a tensor product. While a one-dimensional wavelet transformation decomposes a vector space V^i into

$$W^{i-1} \oplus V^{i-1} = \ldots = W^{i-1} \oplus W^{i-2} \oplus \ldots W^0 \oplus V^0, \tag{4.13}$$

Max Plus Algebra

The extensive use of the minimum and maximum functions inspires to utilize the max-plus algebra $\mathbb{R}_{\max}$ [ACG+90].

The basic operations of $\mathbb{R}_{\max}$ are maximization and addition, which are denoted $\oplus$ and $\otimes$ respectively:

$$x \oplus y = \max(x,y) \tag{4.14}$$

$$x \otimes y = x + y \tag{4.15}$$

for $x,y \in \mathbb{R} \cup \{-\infty\}$. The zero element for $\oplus$ is $\varepsilon = -\infty$ and the inverse element of $x \in \mathbb{R}$ with respect to $\otimes$ is $-x$. The rules for the order of evaluation of the $\mathbb{R}_{\max}$ operators correspond to those of conventional algebra. So $\otimes$ multiplication has a higher priority than $\oplus$ addition.

This formalism is often used to describe discrete event systems [DSVdB08]. Such an event system with only synchronization and no concurrency can be modeled by a max-plus-algebraic model the following form:

$$x_k = A \otimes x_{k-1} \oplus B \otimes u_k \tag{4.16}$$

$$y_k = C \otimes x_k \tag{4.17}$$

with $A \in (\mathbb{R} \cup \{-\infty\})^{n \times n}$, $B \in (\mathbb{R} \cup \{-\infty\})^{n \times m}$ and $C \in (\mathbb{R} \cup \{-\infty\})^{l \times n}$, where m is the number of inputs and l the number of outputs. The vector x represents the state, u is the input vector, y is the output vector of the system, and the counter k is an event counter.

the tensor product $V^i V^i$ is decomposed into

$$W^{i-1}W^{i-1} \oplus W^{i-1}V^{i-1} \oplus V^{i-1}W^{i-1} \oplus V^{i-1}V^{i-1} = \ldots = \tag{4.18}$$

$$W^{i-1}W^{i-1} \oplus W^{i-1}V^{i-1} \oplus V^{i-1}W^{i-1} \oplus \ \ldots \ \oplus V^0V^0. \tag{4.19}$$

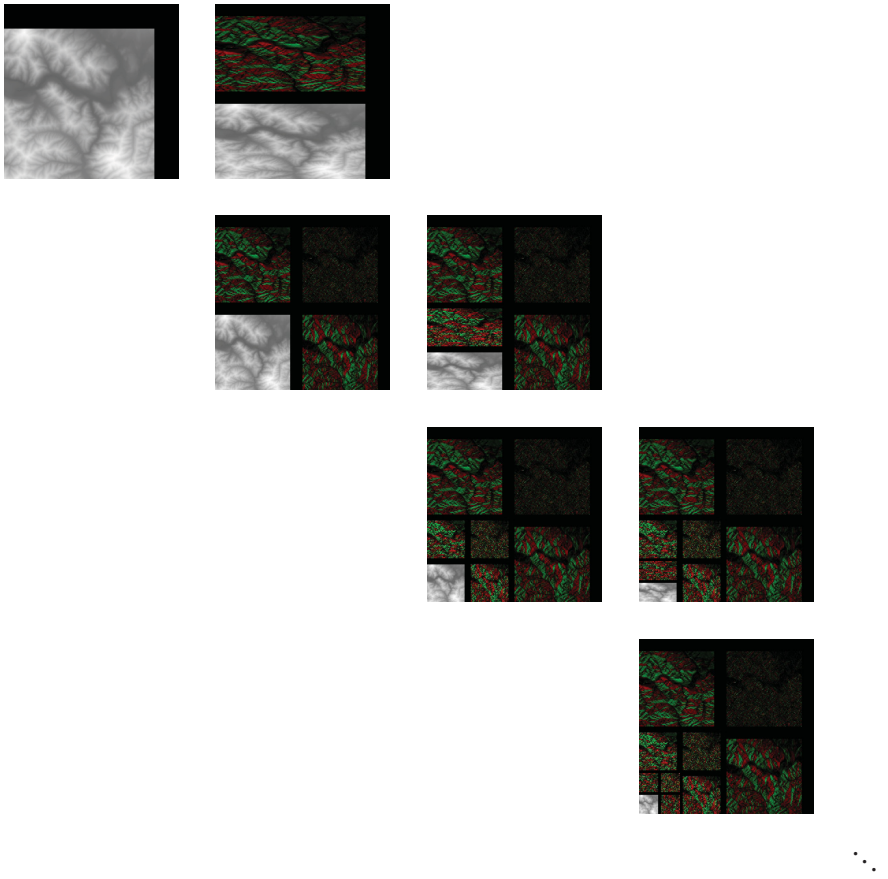


Figure 4.4: A two-dimensional heightfield (upper left) can be visualized by a gray-scale image. Its nonstandard decomposition uses a filter based on maximum operations. In each decomposition step the detail coefficients are represented by red and green pixels. Red pixels mark negative coefficients d_k^j whereas green pixels mark positive ones. A pixel's intensity encodes the difference magnitude (consequently, small differences are almost black).

A function f in space $V^i V^i$ can be represented as

$$f(x,y) = \sum_{(k,l)} c_{(k,l)} b_k^i(x) b_l^i(y) \tag{4.20}$$

with appropriate basis functions $b_k^i \in V^i$. Due to the finite small support of basis functions, it is possible to express the maximum over the rectangular domain $D_1 \times D_2$

$$\max_{(x,y)\in D_1\times D_2} f(x,y) \leq \sum_{(k,l)\in P(D_1,D_2)} c_{(k,l)} \cdot \max b_k|D_1 \cdot \max b_l|D_2 \tag{4.21}$$

$$+ \sum_{(k,l)\in N(D_1,D_2)} c_{(k,l)} \cdot \min b_k|D_1 \cdot \min b_l|D_2 \tag{4.22}$$

as a linear combination of the basis function's minima and maxima. Thereby, the union $P(D_1,D_2) \cup N(D_1,D_2)$ consists of all index pairs, for which the support of b_k intersected with D_1 is not empty (support $b_k \cap D_1 \neq \emptyset$) and support $b_l \cap D_2 \neq \emptyset$, and $P(D_1,D_2)$ are the index pairs with positive coefficients and $N(D_1,D_2)$ are the index pairs with negative coefficients, respectively. As the spaces V^j and W^j are highly structured, the minima and maxima of its basis functions on their support can be precomputed from a prototype impulse and systematically tabulated. The sum (4.21)–(4.22) incorporates an increasing number of basis functions at higher levels of the decomposition, starting with the value corresponding to $V^0 V^0$. At level j, there are $3 \cdot 2^j \cdot 2^j$ basis functions, and altogether, there are $1 + 3\sum_{j=0,\ldots,n} 2^j 2^j$ summands.

For efficiently computing a lower and an upper bound, the maximum $c^j_{\max 1}$ of all positive coefficients corresponding to space $W^j V^j$ and the minimum $c^j_{\min 1}$ of all negative coefficients corresponding to space $W^j V^j$ at level j can be precomputed. Similarly, $c^j_{\max 2}$ and $c^j_{\min 2}$ denote the maximum and minimum corresponding to space $W^j W^j$ and $c^j_{\max 3}$ and $c^j_{\min 3}$ corresponding to space $V^j W^j$. With these values a weaker upper bound with only $6n+1$ summands can be formulated:

$$\begin{aligned}
\max_{(x,y)\in D_1\times D_2} f(x,y) \leq\ & c_{(0,0)} \max\left\{b_0 \,\middle|\, b_0 \in W^j V^j\right\} \\
& + \sum_{j=1}^{n} c^j_{\max 1} \max\left\{b_1 \,\middle|\, b_1 \in W^j V^j\right\} + \sum_{j=1}^{n} c^j_{\min 1} \min\left\{b_1 \,\middle|\, b_1 \in W^j W^j\right\} \\
& + \sum_{j=1}^{n} c^j_{\max 2} \max\left\{b_2 \,\middle|\, b_2 \in W^j W^j\right\} + \sum_{j=1}^{n} c^j_{\min 2} \min\left\{b_2 \,\middle|\, b_2 \in W^j W^j\right\} \\
& + \sum_{j=1}^{n} c^j_{\max 3} \max\left\{b_3 \,\middle|\, b_3 \in V^j W^j\right\} + \sum_{j=1}^{n} c^j_{\min 3} \min\left\{b_3 \,\middle|\, b_3 \in V^j W^j\right\}.
\end{aligned} \tag{4.23}$$

Figure 4.5 sketches the domain sections for which minimum / maximum values are precomputed.

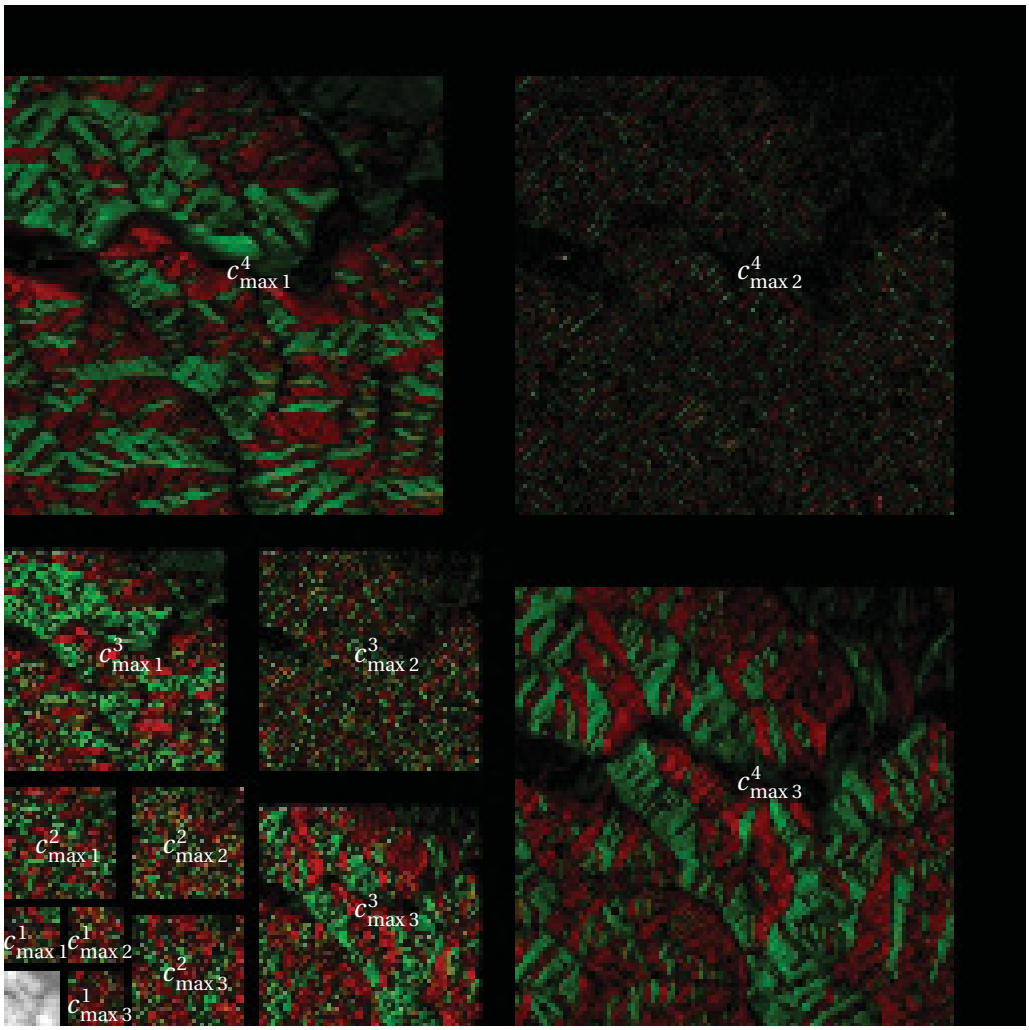


Figure 4.5: Maximum and minimum values of the basis functions can be precomputed. Based on these values an upper bound of the heightfield can be determined.

4.1.5 Optimizations and Empirical Comparison

For efficiency, some conceptual optimizations are also possible. A ray usually has more than one intersection with a heightfield. In the worst case, the query time is $O(l+log_2(n\cdot m))$ which is mainly determined by the traversal length l of the ray up to the first intersection. So it is reasonable to traverse from origin to destination forward, if an intersection is nearby the origin, and backward, if one is nearby the destination. This can be exploited, if there are hints available, for example, from results of queries to the same destination point in the neighborhood (spatial coherency).

We have implemented the kd-tree data structure and its compressed variant with

line-of-sight computation. Implementation details and a comprehensive comparison can be found in "Empirical Comparison of Data Structures for Line-Of-Sight Computation" [FUFB07] and in "Terrain and Model Queries Using Scalar Representations With Wavelet Compression" [FUFB09].

Line-Of-Sight Computations (Heightfield Representation)

Timings of line-of-sight computations depend on the implementation, the heightfield resolution and the number of query points. In this comparison queries to two ground stations per heightfield pixel are answered.

resolution	kd-tree, constant	kd-tree, bilinear	wavelet, constant	wavelet, bilinear
220×220	311 463 q/s	178 676 q/s	298 808 q/s	183 242 q/s
256×256	308 214 q/s	179 615 q/s	289 764 q/s	172 219 q/s
512×512	255 696 q/s	179 332 q/s	255 636 q/s	165 277 q/s
$1\,024 \times 1\,024$	262 440 q/s	143 456 q/s	224 383 q/s	140 322 q/s
$2\,048 \times 2\,048$	245 386 q/s	131 769 q/s	237 468 q/s	118 410 q/s

Table 4.1: Depending on the heightfield representation and the heightfield interpretation (piecewise constant / bilinearly interpolated) the performance of a line-of-sight algorithm (measured in queries per second) differs.

Table 4.1 lists the computation times for different resolutions of the same heightfield data set. All timings were taken on the same hardware and software platform. The point reconstruction is fastest as it needs to access only a single height value at the leaf level. For bilinear approximate and bilinear reconstruction, we try to keep the four leaf nodes required for the reconstruction sequentially in memory. The resulting performance is roughly $3/4$ of that of point reconstruction. In particular, it is notable that the wavelet-based storage scheme is nearly as fast as the kd-tree with fully stored inner nodes. For the more complex bilinear reconstruction in leaf nodes, the difference is even smaller.

The tests comparing search directions in Table 4.2 show that using the information where the intersection point was in the last query (at a different height or in a horizontal or vertical neighbor) is most successful. In addition, this information is very easy to exploit. Note that it is not easy to predict, if a forward or backward search has a shorter search length up to an intersection as it requires carrying out the search.

Line-Of-Sight Computations (Search Direction)

A well-known optimization technique, not only in computer graphics, uses spatial coherence. In this context the information where the last intersection point was located is reused to set the search direction.

search direction	kd-tree, point	kd-tree, bilinear	wavelet, point	wavelet, bilinear
forward	308 084 q/s	163 273 q/s	281 024 q/s	151 095 q/s
backward	316 837 q/s	179 889 q/s	291 051 q/s	177 482 q/s
last-fastest	317 766 q/s	177 758 q/s	293 648 q/s	183 242 q/s

Table 4.2: The tests concerning search direction (forward, backward or based on which one was faster in last query) on a heightfield of resolution 220×220 show that the information where the intersection point lied in the last query at a different height or in a horizontal or vertical neighbor is most successful.

Laser Scanning

A laser scanner is a device to capture and to measure the surface of an object. Most laser scanning systems are active and non-contact system [tHCMV05]; i.e. they use an active light source and detect its reflection in order to probe an object or environment:

- A time-of-flight system measures the time of a round-trip (scanner – object – scanner) of a pulse of light to calculate the distance of an object.
- A triangulation system uses a camera to detect the reflection of the laser and to determine the 3D point where the laser hit the object. Figure 4.6 illustrates this process.

Each 3D scanner has a limited field of view and can only scan surfaces that are not obscured. The result of a single scan – a so-called range map – is a heightfield in spherical coordinates.

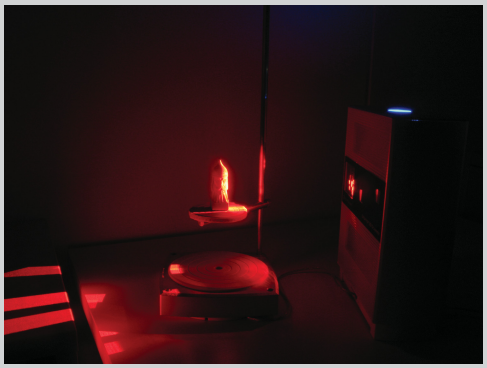


Figure 4.6: A triangulation-based laser scanner uses a camera to detect the reflection of a laser and to determine the 3D point where the laser hit the object. The NextEngine™ laser scanner uses a multi-laser precision technique to achieve a high accuracy.

The scanning device is located at the origin and the azimuthal angle θ the polar angle ϕ describe the field of view. The measured distance from a point to the origin is the radius (r,θ,ϕ). These spherical coordinates describe the 3D position of each point in a local coordinate system relative to the scanner.

In general, a single scan cannot capture the surface of a complete object. Therefore, multiple scans from different directions are needed to obtain enough information to describe the whole object. Normally, a change of position or orientation of either the scanning device or the object to scan is not respected by the local coordinate system. As a consequence, each range map has its own coordinate system and all range maps have to be aligned to each other [PFC+05]. This alignment step is one of the first steps of the scanning pipeline [BR02]. It consists of

- alignment and registration steps to transform all range maps into one reference coordinate system,
- a clean-up step to remove unwanted points from a range map,
- a fusing step to merge all scans to one mesh and
- an optional polish step to fill holes and to simplify the final mesh.

The point clouds produced by 3D scanners are usually not the final object representation.

As most applications use polygonal 3D models, NURBS surfaces, or subdivision surfaces, the scanning pipeline is followed by a reconstruction or modeling pipeline to create a computer-aided design (CAD) representation.

Figure 4.7 shows two range maps of a series of scans (left, middle) and the final result (right) – a polygonal mesh, which can be used in further reconstruction and modeling steps.

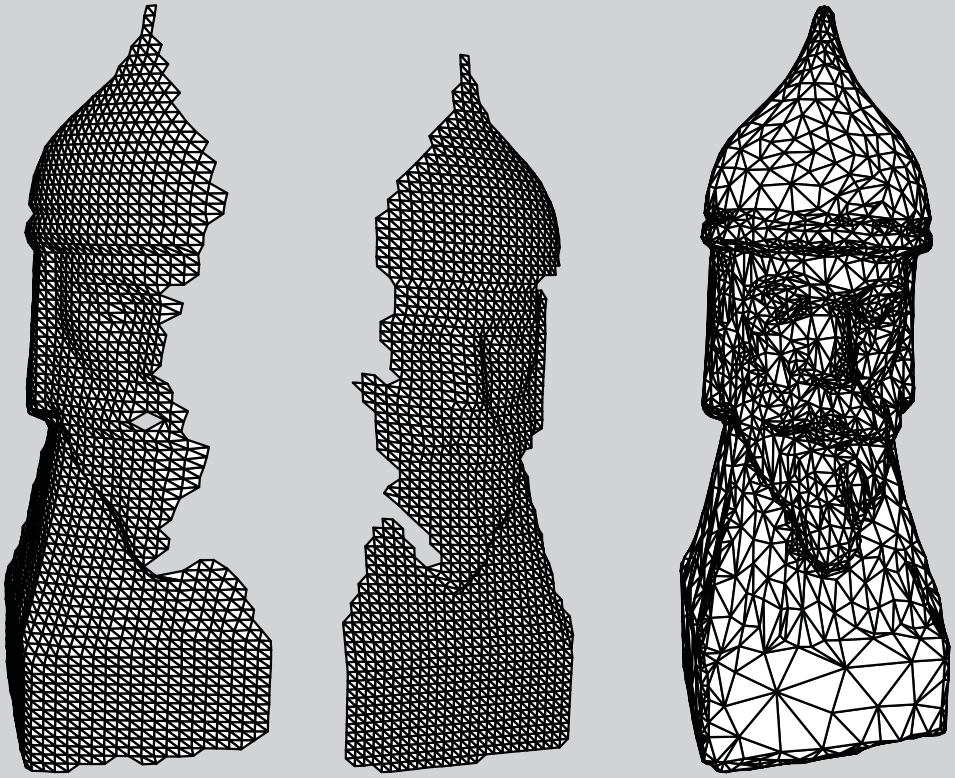


Figure 4.7: To capture a complete, non-trivial object several single scans are needed. This chess piece has been scanned in thirteen single scans: seven circular scans of a 360°scan and two bracket scans from above and from below. Each bracket scan consists of three single scans. The triangulation-based laser scanner projects a raster onto the object to scan. The two range maps (left, middle) which are part of the 360°scan show the characteristic pattern [Nex09].
At the end of the acquisition pipeline all range maps are merged to a final, polygonal mesh. A cleaned and simplified mesh (right) is the input data set of many computer-aided design tasks.

4.2 Collision Detection

In many 3D acquisition pipelines, the first measurements are stored in heightfields and range maps. The result of the 3D acquisition pipeline is a "high-level" geometry description – a CAD model. The less parameters a CAD model has, the "higher" its description. Therefore, research in computer-aided geometric design has always focused on model representations, which describe a complete 3D object only by a few parameters, control vertices, etc. While these descriptions are suitable for modeling, in the context of collision detection they are often too complex. In this case, optimized spherical heightfields [FUF06], which are similar to the ones used at the beginning of the acquisition pipeline, can be utilized to solve the collision detection problem at a later stage.

The problem of collision detection between objects is fundamental in many different communities including CAD, robotics, computer graphics, and computational geometry. The general problem of collision detection consists of three categories:

- collision detection, which tests whether two or more objects collide;
- collision determination, which determines which parts of some objects intersect; and
- collision response, which answers the question, *Which action should be taken in response to a collision?*

For practical collision detection, all approaches first consider enclosing, simplified model representations, so-called bounding volumes, of all participating models. This broad phase only extracts potential collision pairs [LM03]. The following narrow phase performs a precise collision detection for each potential collision pair with the model representation. A general approach for all model representations uses hierarchies of simple bounding volumes containing model parts again.

Researchers have proposed several bounding volumes including spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB), and discrete orientation polytopes [LM03]. Here the performance depends on the tightness of the bounding volume, the efficiency of the intersection test for the bounding volume, and the strategy for hierarchy generation. STEFAN GOTTSCHALK, MING C. LIN, and DINESH MANOCHA give a complete description of the tree construction and collision test using oriented bounding boxes in "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection" [GLM96].

Recently, researchers have modified the approaches using bounding volume hierarchies for time-critical collision handling [Eri04]. Bounding volume hierarchies with deformable models require additional time for refitting [TKH+04], and optimizations exist for special deformations [KZ05]. In the area of deformable models, researchers commonly use simpler structures like Cartesian grids and 1D arrays accessed by hashing. WILLIAM A. MCNEELY et al. built up a voxel grid for the static model, where the points of a small movable model are queried against [MPT99]. This approach guarantees the high feedback rate needed by a haptic feedback device, and it

is tailored to the haptic application domain. ARNUPH FUHRMANN et al. also use a grid to store the Cartesian distance field for a static model draped with a deformable cloth model [FSG03]. Here, as with MCNEELY's approach, one model is completely static. Problems of these approaches are the large memory consumption and the necessity to choose the grid resolution beforehand. Accessing rotated Cartesian grids is a costly operation, if done randomly and without exploiting coherence. Cartesian grids classically serve as a spatial data structure for the simulation domain. MATTHIAS TESCHNER et al. use hashing as a grid compression technique and perform collision detection between points and tetrahedral elements directly on the hashed array [THM+03]. The performance strongly depends on a uniform distribution of tetrahedral elements and points within the hashed array.

Other researchers consider image-space techniques for collision detection using graphics hardware [HTG04]. One approach is to perform ray casting through the volume of interest. It can be implemented on rasterizing graphics hardware using orthogonal projection with the depth and stencil buffers. The resulting layered depth representation can be made robust against vanishing depth intervals, but requires slow buffer read-backs in multiple passes [HTG04]. Alternative approaches avoid buffer read-backs. They detect edge points of one object inside another object by counting ray-surface intersections [KP03]. This approach is quite fast, but it is not robust in case of occluded edges. NAGA GOVINDARAJU et al. use hardware-accelerated occlusion queries instead [GRLM03]. The visibility tests allow for sorting out objects or object primitives, which do not participate in any collision within the set. Although the setup is complex it can report detailed collision information. The approaches using graphics hardware previously were fast at collision detection or point-in-volume tests, but had difficulties with collision determination, like extracting all colliding object parts.

4.2.1 Spherical Distance Fields

In contrast to multiresolution representations with wavelets [VP04] or progressive meshes [Hop98], the approach by CHRISTOPH FÜNFZIG, TORSTEN ULLRICH and DIETER W. FELLNER [FUF06] is simpler and does not reproduce the model topology exactly, but generates conservative bounding volumes for model parts efficiently.

The spherical sampling according to a single center point requires a spherical parametrization such as one with six charts derived from the box sides [PH03b]. A spherical representation allows fast model rotation and the on-the-fly generation of spherical shell bounding volumes for model parts. Others have already used spherical shells as bounding volumes for spline models [KPL98]. SHANKAR KRISHNAN et al. use this bounding volume in an oriented bounding box hierarchy (tree) to bound spline model parts more tightly [KGL+98]. The robotics literature has called the same bounding volume bi-sphere.

Axis-Aligned Bounding Boxes

The preprocessing step of a collision detection based on axis-aligned bounding boxes determines a model's bounding box with axes that align with those of the coordinate system. This is an easy minimum/maximum search over all (control) vertices in many commonly accepted model representations. Afterwards, the bounding box is split at its longest edge into two bounding boxes that are refitted to the model. This subdivision process is called recursively by all boxes down to the desired granularity. This process is illustrated in Figure 4.8. The bounding boxes are stored in a tree structure.

A single bounding box is typically stored using two points. The intersection test uses these two points to determine the boxes' projection onto the coordinate axes. The enclosing intervals on the coordinate axes can be read off easily. Two bounding boxes overlap, if and only if all coordinate intervals of both bounding boxes intersect. A two-dimensional configuration is shown in Figure 4.9. The algorithm uses the presented data structure to exclude irrelevant candidates as fast as possible. A collision check of two models starts by checking the root bounding boxes.

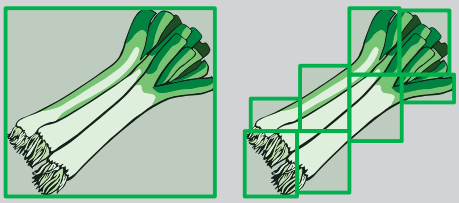


Figure 4.8: The preprocessing recursively subdivides the bounding box (left) down to the desired granularity (right).

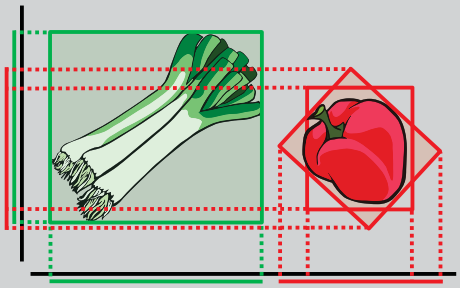


Figure 4.9: The AABB intersection test can be reduced to some interval checks.

If these boxes do not overlap, a collision is not possible. Otherwise, descending the AABB tree, both bounding volumes are refined and checked again. The last refinement level has to work on the model's primitive representation.

The AABB-based collision detection has many advantages. The algorithm is easy to understand and to implement. A simple AABB collision detection normally consists of a few lines of code. The implementation of this algorithm is numerically stable. Even the use of floating-point values is extremely stable in combination with a bounding box offset within the magnitude of floating-point precision. Furthermore, the algorithm has no restrictions concerning the underlying model representation.

Regrettably, the algorithm also has some inherent disadvantages. The alignment of all bounding boxes according to the coordinate system simplifies the intersection test, but it has suboptimal tightness in the general case. There can be a big difference between the optimal bounding box and the axis-aligned bounding box of an object as shown in Figure 4.10.

Figure 4.10: The fixed orientation of a bounding volume does not result the optimal bounding box. This figure shows the optimal orientation in contrast to the orientation in Figure 4.8.

Another problem is the inability to reuse the AABB data structure during object rotations. If an object is rotated along an arbitrary axis, its bounding boxes have to be realigned (using the object's bounding box) as illustrated in Figure 4.9 or even recalculated from the object's points.

The expected running time of an algorithm is the measure of its quality. To answer a collision query of two objects, the AABB-based collision detection algorithm traverses both bounding volume hierarchies. The required time to answer the query consists of the initial setup costs, the costs of a single bounding volume/bounding volume overlap test (multiplied by the expected number of overlap tests), and the costs of a single primitive/primitive intersection test (multiplied by the expected number of intersection tests).

In an asymptotic analysis, the number of overlap tests N defines the running time, where n denotes the number of bounding volumes each of the two query objects owns. The conditional probabilities that a pair of bounding volumes overlap can be estimated using geometric reasoning [WKZ06]. Assuming that the two root-bounding volumes overlap, the expected number of overlap tests only depends on scaling factors that relate the size of a bounding volume to the size of its children. For the sake of clarity and simplicity, these scaling factors α_x, α_y, and α_z along each axis shall be constant throughout the hierarchies of the query objects. Then the expected number of overlap tests can be estimated by

$$N(n) \leq \sum_{i=1}^{\lg n} 4^i \cdot \alpha_x^i \cdot \alpha_y^i \cdot \alpha_z^i \qquad (4.24)$$

$$\in O\left(n^{lg(4\alpha_x\alpha_y\alpha_z)}\right). \qquad (4.25)$$

Plugging this estimation into a cost function $T(n)$ yields an overall time estimate that is illustrated in Table 4.3. The frequently used implementation, which divides a bounding volume in the middle of its longest edge into two parts (with a small overlap), has a scaling product of $\alpha_x \cdot \alpha_y \cdot \alpha_z \approx 1/2$ i.e., the expected running time is linear to the number of bounding volumes.

Expected Running Time

The effect of the scaling factor product on the query time.

$\alpha_x \cdot \alpha_y \cdot \alpha_z$	$T(n)$
$< 1/4$	$O(1)$
$1/4$	$O(\log n)$
$1/2$	$O(n)$
$3/4$	$O(n^{1.58})$
1	$O(n^2)$

Table 4.3: The expected runtime costs of a AABB collision test.

GREGORY J. HAMLIN, ROBERT B. KELLEY, and JOSEP TORNERO present an algorithm for distance computation between bi-spheres and more general convex hulls of a finite number of spheres (S-topes) [HKT92], which resembles the Gilbert-Johnson-Keerthi algorithm for convex polytopes [GJK88]. The spherical shell bounding volumes at each level of the hierarchical spherical distance field discard the parts in certain sphere sectors from further consideration. Depending on the application requirements, the algorithm can report at the leaf level a single triangle per spherical shell, a triangle per model layer inside a spherical shell, and all triangles inside a spherical shell.

4.2.2 Spherical Model Representation

Like most collision detection algorithms, the approach by CHRISTOPH FÜNFZIG, DIETER W. FELLNER and myself [FUF06] consists of two parts: a preprocessing step and a testing routine that represents the intrinsic collision test. A simple 2D example outlines the main idea.

During the preprocessing step, the algorithm takes an initial model and determines a model center. According to this center point, the model is transformed into spherical coordinates and sampled into several intervals. As a result, the algorithm stores the maximum radius and the minimum radius for each sample. The basic concept used in the preprocessing step works analogously to a wavelet transform. It starts with the high-resolution object and derives objects of lower resolution by storing the differences in detail coefficients to reverse this operation. For correctness, a low-resolution model must bound all higher resolution models. Therefore, the Haar basis functions – which are used e.g. in image processing and describe an averaging and differencing process – are unsuitable. In this context, maximizing and minimizing functions have to be taken as Figure 4.11 (top) shows. Therefore, each object will be transformed into an inscribed circle and a circumference in the lowest resolution. Having transformed all objects this way, we can perform a simple and fast collision test (see Figure 4.11 (bottom)). For clarity, we use solid objects with zero minimum radius in the illustration.

At runtime, the intersection test starts with the model representation at its lowest resolution and tests whether both models collide at this resolution. If this test is positive, the algorithm will increase the level of detail. Therefore, it is important for performance that only intersecting sectors are considered further on. Figure 4.11 (bottom), which shows the various stages during a collision test, illustrates this principle. As long as there are intersecting sectors of different objects, the algorithm refines the objects. If the algorithm reaches the highest resolution of an object, the collision test is performed on the object primitives, which results in collision determination.

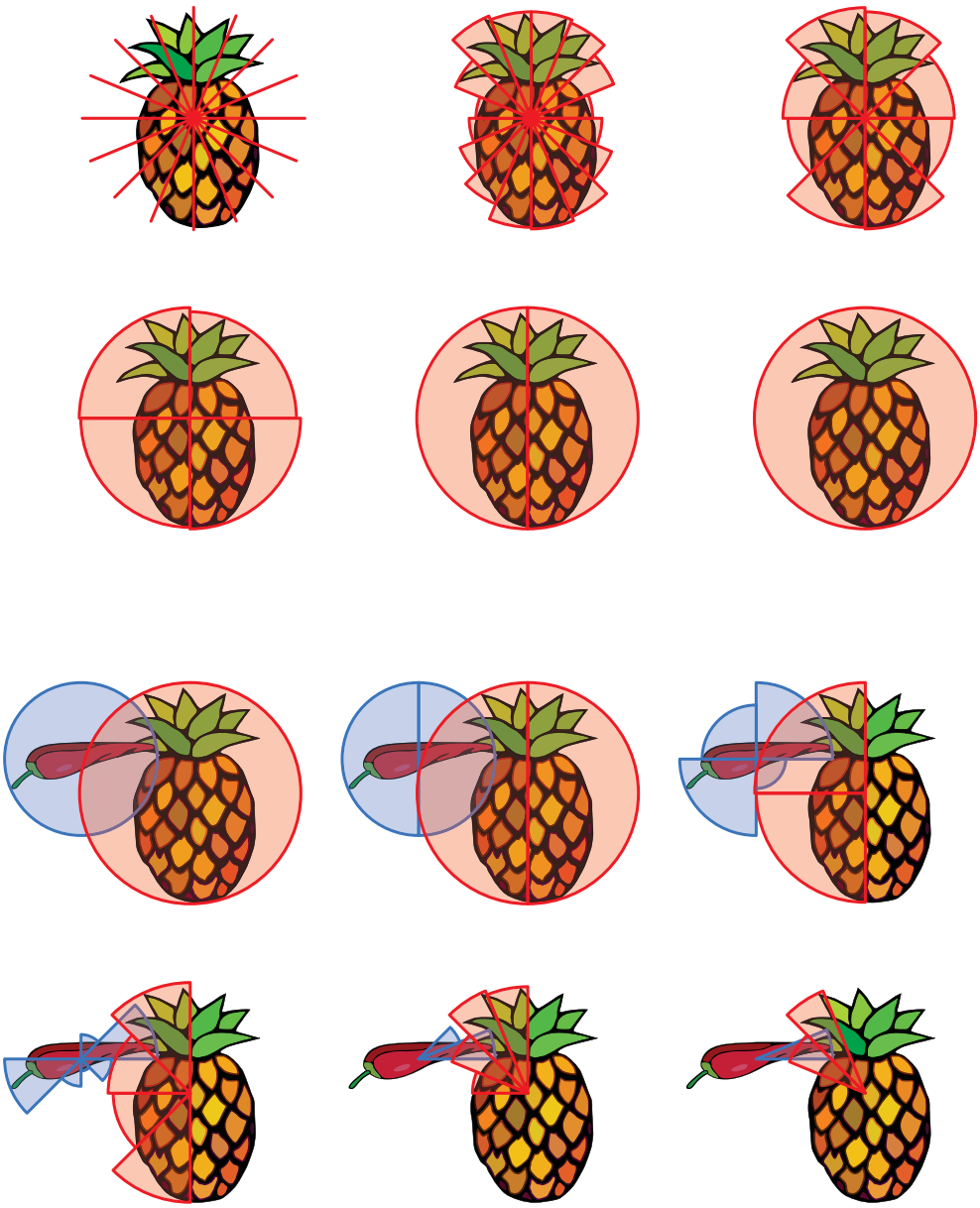


Figure 4.11: Hierarchical spherical distance fields can be used to solve the collision detection problem. During the preprocessing step (top) the distance field-based algorithm uses a spherical sampling of an object (top left) and applies maximum filters to it. The results of these filters are shown in the top rows. At the lowest level a bounding sphere encloses the whole object.
At runtime, the intersection test starts with the bounding volume at lowest resolution. As long as different bounding volumes intersect, the volumes are refined. Sectors that do not collide with other sectors are not considered further on. At highest resolution, only model parts belonging to colliding sectors have to be checked for collision. Depending on the model representation, appropriate algorithms have to be used.

4.2.3 Spherical sampling

A model is described in spherical coordinates (r,θ,ϕ) with respect to its center, an arbitrarily chosen point. The choice of the center point and its position is important for the sampling process. A center point, with respect to which the model is star-shaped, is preferable. But even if such a star exists, it is expensive to determine. Heuristic choices such as the center of an enclosing sphere or the model's mass center are good enough to serve as a sampling center. The result of the sampling is a spherical heightfield $r(\theta,\phi)$ over the parameter domain $[0,2\pi] \times [-\pi/2,\pi/2]$, which encloses the whole object. As in this parametrization, all meridians coincide with each other at the poles, an intersection test in a near-pole region would lead to many descending tests and a bad performance. Therefore, we use another sphere parametrization (see Figure 4.12). It subdivides a sphere into six separate, congruent regions and applies an angle-based parametrization to each side. A simple projection of a bounding box upon a sphere would be sufficient, but it also has some disadvantages. The projection of a bounding box grid onto a sphere results in patches of an unequal area [PH03b]. This problem can be eased by an angle-based parametrization

$$\mathbf{d}(x,y) = \frac{1}{\sqrt{1+\tan^2\frac{\pi}{4}x+\tan^2\frac{\pi}{4}y}} \begin{pmatrix} \tan\frac{\pi}{4}x \\ \tan\frac{\pi}{4}y \\ 1 \end{pmatrix} \tag{4.26}$$

with $(x,y) \in [-1,1]^2$, which delivers much better results, as Figure 4.12 shows. Having discretized and sampled the models at the beginning of the preprocessing step, the

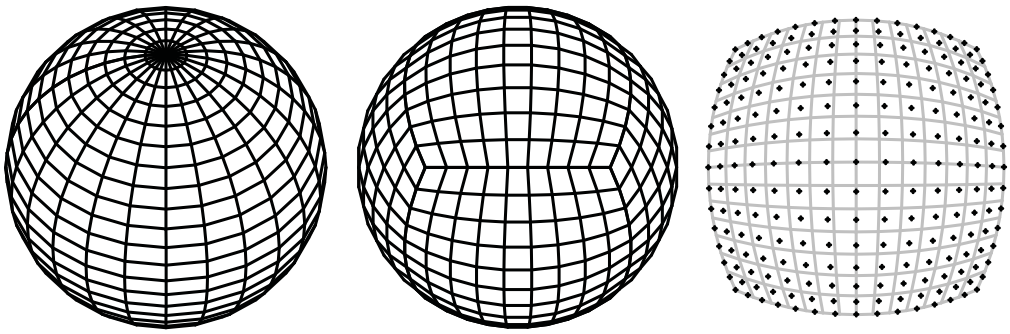


Figure 4.12: A typical sphere parametrization (left) is not preferable for a hierarchical, refinable data structure. As all meridians converge at the poles, a refinement in a near-pole region would lead to many descending paths. A box-based parametrization (middle) consists of six congruent regions and eliminates this problem. Each region has an angle-based (right, grid) parametrization. A simple cube projection (right, crosses) would result in distortions.

resulting heightfields have to be transformed to get the lower resolution representations. The algorithm can handle subsampling the six charts of the spherical representation analogous to a discretized Cartesian heightfield explained above.

4.2.4 Intersection Test

Having transformed all objects this way, it is possible to perform a simple and fast collision test. At runtime, the intersection test starts with the model representation at the lowest resolution and tests whether they collide or not. If this test is positive, the level-of-detail will be increased. Thereby, it is important for performance purposes that only intersecting sectors are considered further on.

As long as there are intersecting sectors of different objects, the algorithm refines the objects. If the algorithm reaches the highest resolution of an object, the collision test is performed on the object primitives, which determines the colliding subparts. The most relevant part concerning performance is the intersection test routine. Without a doubt, the test has to report an intersection, if there is one. However, if there is no intersection, the test may report one by mistake. The less faulty positive results are reported, the faster the algorithm works, as unnecessary refinements are omitted. Therefore, balancing accuracy and efficiency is essential.

At level zero, both objects to test are enclosed by tight spheres. The test, whether two spheres intersect each other, requires no simplification. But at all other levels of detail, the algorithm has to check two sphere sections, which it analyzes using three methods.

- So-called bounding volume tests enclose the objects to test within geometrically simpler objects. For example, a polygonal frustum might be used to enclose a sphere section. Although the intersection test of two polygonal frustums is rather simple, the number of tests increases and the enclosing quality is rather bad.

- A totally different approach to check for intersections uses interval/affine arithmetics [Bühler01]. In theory, this approach offers an exact and fast intersection test. Although such a test might exist, a practical test has not been implemented yet.

- The intersection test presented elsewhere [KPL98], [KGL+98] for spherical shells distinguishes several configurations. For each configuration, the authors derive an algebraic test for intersection. This test is suitable for our purposes, but due to its complexity and computational expense, we prefer a computationally cheaper alternative.

The following geometrical test shows reasonable performance. The objects to intersect are considered as cones as previously described with a center C, a normalized axis vector $\mathbf{a}$, a length u, and an opening radius r measured at height $C+\mathbf{a}$. If the objects, on which the intersection tests are performed, are not considered as volumetric objects but as surfaces, they are handled as truncated cones of which the cone end is

cut off at height l, $0 \leq l \leq u$. In the following, both cones and their cone parameters are distinguished by indices 1 and 2. Subject to the orientation of the cone axes, the implementation analyzes two cases. The first case deals with nonparallel axes a_1, a_2; the second one deals with parallel lines. A heuristically chosen threshold

$$\min\left(\sphericalangle(\mathbf{a}_1,\mathbf{a}_2), \sphericalangle(-\mathbf{a}_1,\mathbf{a}_2)\right) < \alpha_{\max} \tag{4.27}$$

separates the nonparallel case from the parallel one:

Nonparallel axes In the nonparallel case for both axes, we will determine the shortest distance and the respective perpendicular points P_1, P_2. Let s and t be the corresponding parameters on the axis lines. If a perpendicular point lies outside the clipped cone, it is moved toward the cone. The algorithm computes the other point as the nearest point on the axis of the other cone. In the special case where both perpendicular points are outside, we move the one that is nearest to its clipped cone.

As a quick rejection test we consider the cones as cylinders, that is, if the distance between P_1 and P_2 is greater than the sum of the maximum radii $(u_1 \cdot r_1 + u_2 \cdot r_2)$ of both cones, then an intersection is impossible.

Next, for a quick acceptance test we consider the spheres with center P_1 respectively P_2 and radius equal to the cone radius there, that is, if the distance between P_1 and P_2 is smaller than the sum of sphere radii $(s \cdot r_1 + t \cdot r_2)$, then we have an intersection.

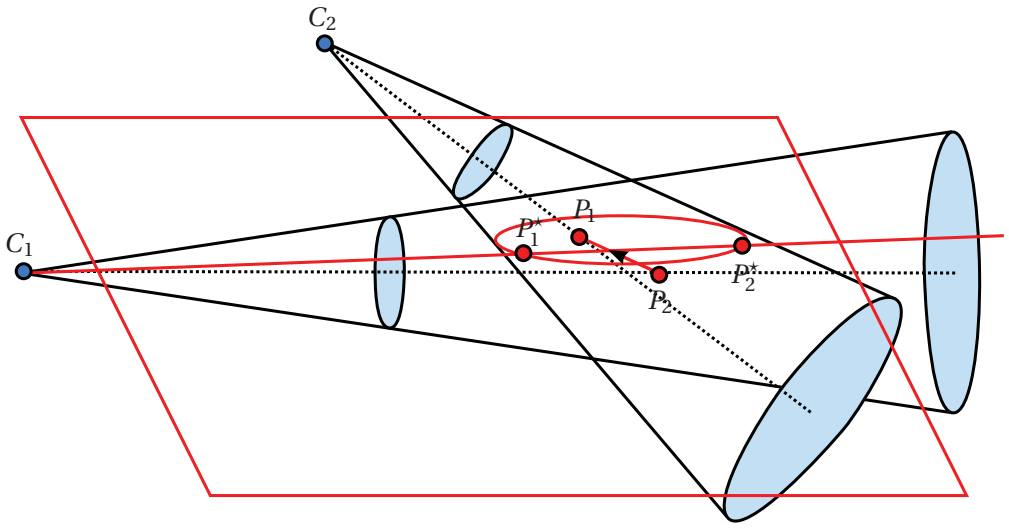


Figure 4.13: The nonparallel cone-cone intersection can be reduced to an intersection of a line and a conic section (an ellipse or a parabola; hyperbola cannot occur).

If neither a quick rejection nor a quick acceptance occurs, the algorithm performs an exact test based on a cone section. The second cone is intersected with a plane containing the first cone's center C_1 and the line through P_1 and P_2 as illustrated in Figure 4.13. This results in a well-known cone section. The first cone is reduced to a line that passes its center C_1 and the point P_1 moved within the considered plane toward the cone section's focus by length $s \cdot r_1$. The intersection test is then reduced to a 2D intersection test between a cone section and a line. The cone section of a hyperbola cannot occur here, as the considered plane always has an intersection with the other cone's axis by construction. The correctness proof can be found in "Hierarchical Spherical Distance Fields for Collision Detection" by CHRISTOPH FÜNFZIG, TORSTEN ULLRICH and DIETER W. FELLNER [FUF06].

Parallel axes In this case, the algorithm analyzes the projection of one cone onto the other cone's axis, as sketched in Figure 4.14, and vice versa. For each projection, two distance checks of points against their according radii are performed, which results in a total of four checks. The special case of parallel lines is handled separately for optimization – in this case, the whole test can be done by some interval checks, which are much faster.

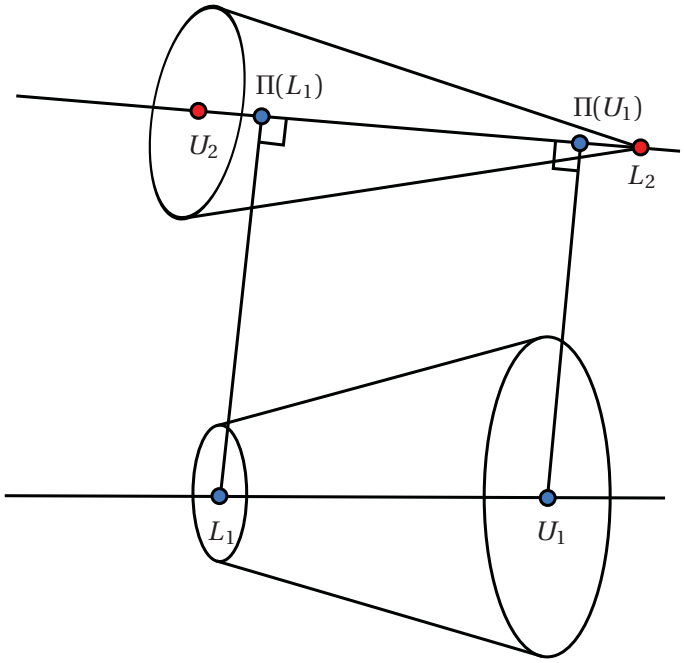


Figure 4.14: The cone-cone intersection test for parallel or "near-parallel" cases can be performed using interval checks. For these checks a cone is projected onto the other cone's axis.

Although the intersection test might give the impression that it takes a lot of time, it only moderately does so. Due to the fact that the used spherical representation allows for easy rotation and translation, the intersection test can keep up with, for example, discrete orientation polytopes (k-DOPs), if the tested objects are in arbitrary orientation to each other.

4.2.5 Technical Details

We implemented our collision detection algorithm for arbitrary models in the OpenSG scene graph system [RVB02], where the general model representation is a polygon soup.

The center point C of a model is its center of mass. For the model sampling two arrays $l_{i,j}$, and $u_{i,j}$ of power-of-two resolution $(i,j = 0,\ldots,2^m-1)$, are filled for each of the six sides. The first array l contains the minimum distance values of model points, whereas the second array u contains the maximum distance values. The entries $l_{i,j}$, $u_{i,j}$ represent the distance for model points in the infinite cone

$$(C,\mathbf{d}(x_i,y_j),r_m) \tag{4.28}$$

with axis

$$\mathbf{d}(x_i,y_j), \qquad x_i,y_i = -\frac{2^{m-1}-1/2}{2^{m-1}},\ldots,\frac{2^{m-1}-1/2}{2^{m-1}} \tag{4.29}$$

and opening $\tan(r_m)$ – see Equation (4.26).

Both the opening angles and the corresponding circle radii r_n for cones at level $n,(n = 0,\ldots,m)$, containing rectangular subparts (see Figure 4.15), can also be derived from Equation (4.26).

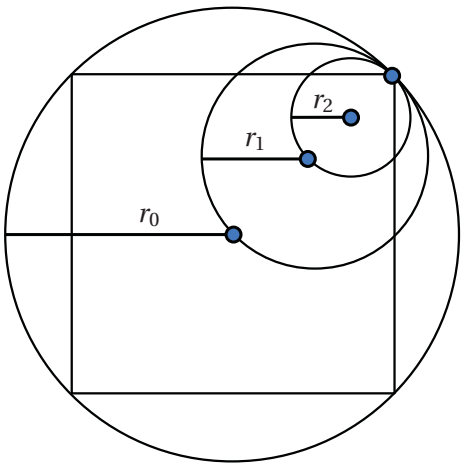


Figure 4.15: The cone radii r_n at different levels n with $n = 0,\ldots,m$ enclose the subdivided parameter square.

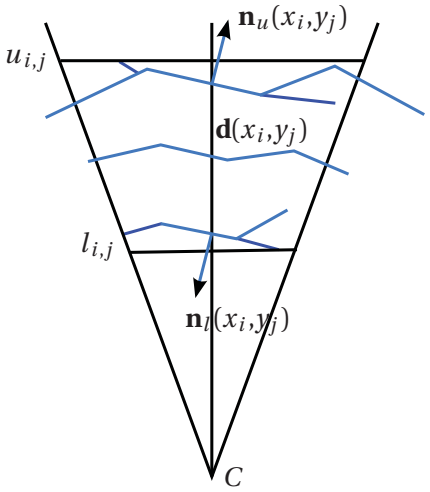


Figure 4.16: Each sphere sector is defined by its center point C and its axis $\mathbf{d}(x_i,y_j)$. During the preprocessing step it is sampled to determine the minimum and maximum distances $l_{i,j}$ and $u_{i,j}$.

We can compute these arrays in two different ways. First, we can use ray casting. For each direction $\mathbf{d}(x_i,y_j)$ with domain parameter x_i, y_j, we cast two rays (see Figure 4.16). The first ray is toward the center point – that is, with origin $C+\Delta\cdot\mathbf{d}(x_i,y_j)$ using a sufficiently large Δ, so that the ray origin lies outside the model, and direction $-\mathbf{d}(x_i,y_j)$. The second ray is from the center point outward. We continue casting rays outward to build a list of faces, pierced by the sampling direction.

During model preprocessing we collect face neighborhoods and store these as a mesh with at least partial connectivity. We do not require any special topology here. For each intersected face we visit neighboring faces until the outer face points lie outside the sampling cone. From the nearest and farthest model layers we determine the minimum and maximum distances $l(i,j)$ respectively $u(i,j)$. The list of triangles is sorted in descending order according to the triangle distance. Additionally, we store with each triangle the diameter of a bounding circle.

In the second approach, we rasterize the model triangles onto the six sides, rather than sampling the model with rays. First, we calculate the side k, $k=1,\ldots,6$ for each point P_i of the triangle (P_1,P_2,P_3). Then we rasterize the triangle in the parameter domain (θ,ϕ) for side k with a classical scanline algorithm along the coordinate ϕ. The parameters of the triangle point P_i are:

$$\theta_i = \arctan\left(\frac{(P_i-C)_{k+1 \mod 3}}{(P_i-C)_{k \mod 3}}\right) \tag{4.30}$$

$$\phi_i = \arctan\left(\frac{(P_i-C)_{k+2 \mod 3}}{(P_i-C)_{k \mod 3}}\right) \tag{4.31}$$

For each point met by the scanline algorithm, the distance value $d(\theta,\phi)$ has to be calculated. We can interpolate the distances $d_i = ||P_i - C||$ of the triangle points into

point (θ,ϕ) by the spherical barycentric coordinates b_1, b_2, b_3 of this point [CM05]. The distance $d(\theta,\phi)$ is

$$\frac{1}{d(\theta,\phi)} = b_1 \cdot \frac{1}{d_1} + b_2 \cdot \frac{1}{d_2} + b_3 \cdot \frac{1}{d_3}. \tag{4.32}$$

With these values, we then update the minimum and maximum distance values in the two arrays.

The ray casting approach is usually faster than rasterizing the triangles onto the six sides, although it can miss small model parts (of projected size corresponding to a leaf cone). For the rasterization, large triangles sticking out of a side have to be clipped beforehand. After initialization, the model representation (6×2 distance arrays) is nonstandard transformed as a heightfield using maximum and minimum filters. Given the transform data, we can implement the collision test for two objects. Each side i, ($i = 1,\ldots,6$), of the first object is tested for intersection against each side j, ($j = 1,\ldots,6$), of the second object. The intersection method maintains a queue of spherical shell pairs

$$\big((C_1,\mathbf{d}(x_1,y_1),l_1,u_1),(C_2,\mathbf{d}(x_2,y_2),l_2,u_2)\big) \tag{4.33}$$

to test for intersection. This test processes the queue in a breadth-first manner. If the intersection test for the front element of the queue is true, then one of the two cones is refined into four subcones. In our current implementation, we alternate the refinement of cones for the first model with those for the second model. When all cone pairs of this round are tested and the queue is still not empty, then we do one reconstruction step on the corresponding distance arrays. At this time, the cone pairs in the queue build a subset, where each test partner comes from the same refinement level in the corresponding distance arrays.

If we reach the leaf levels for both models, then we have to consider the lists of contained triangles for collision determination. In the simplest case of a star-shaped model, we can report an intersection and return one of the contained triangles. In the general case, all triangle pairs have to be checked for intersection. This can be optimized by using the bounding circle of the first triangle to prune the sorted list of inside triangles of the other model. During the intersection tests from different, adjacent shells, the amount of overlap between bounding circles should be as small as possible. This avoids repeating triangle intersection tests and potential duplicates in the collision result, which is important for the overall performance. Eliminating duplicates in the collision result afterwards is expensive. As an alternative approach we can also report a single triangle per layer of each model. The model layers are immediately defined by the ray casting precomputation. Each time we continue ray casting, the layer number is increased by one. But the model can also be classified into layers in a separate pass by comparing triangle distance differences with a given threshold value. The threshold value has to be chosen according to model parameters for the result with the average triangle diameter as the threshold value. Therefore, we can customize the collision determination according to the application's requirements.

4.2.6 Benchmark

To show the new collision detection method's efficiency, we ran a series of tests. Benchmarking collision detection algorithms realistically is a difficult task. This is because there is a wealth of models with different characteristics and motions relative to each other.

GABRIEL ZACHMANN proposes a benchmarking scheme for two models each contained in a unit box, where the second object performs a number of full-z rotations (in 1 000 frames) at decreasing distances relative to the first one [Zac98]. Throughout all benchmarks, we used ray casting for model sampling with 8×8 and 16×16 samples per side. For collision determination, each sample uses a sorted list of inside model triangles (with their bounding circle radii), and the information subdividing the list into layers. The benchmark *Transform Single (S)*reports a single triangle pair in a colliding shell pair, *Transform Single Layer (SL)*reports a single triangle pair per layer of a colliding shell pair, and *Transform* reports all triangle pairs in collision (without elimination of duplicates). To compare this approach with well-known collision-detection methods, we included timings using 18-DOP. bounding volumes [Zac98] and oriented bounding volumes as in the publicly available library Rapid (Robust and Accurate Polygon Interference Detection[2]). These approaches also report all triangle pairs in collision.

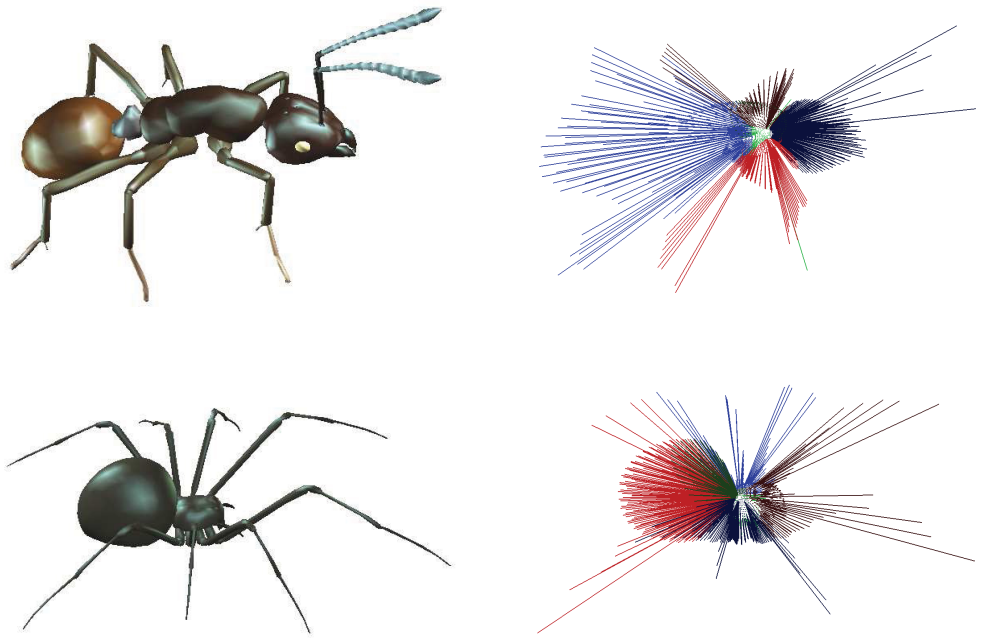


Figure 4.17: The test models of the first benchmark consist of 6 667 (ant) and 5 162 (spider) triangles. Their shapes are non-convex and non-star-shaped.

[2] UNC Research Group on Modeling, Physically-Based Simulation and Applications, http://gamma.cs.unc.edu/OBB/

Figure 4.17 shows the collision test for the models ant and spider. Both models consist of a rather small amount of triangles, but they are highly non-convex and non-star-shaped. The model ant consists of 1.7 layers per shell on average, and the spider consists of 1.8 layers on average. In the right part of the figure, the model samplings are sketched by the clipped axes of the spherical shells. For the triangle sizes in these models, the approximation with 8×8 samples per side is already sufficient. Up to the center distance of 1 unit, the few collision situations can be verified with spherical shells only. The timings (plotted in Figure 4.19) are comparable to hierarchies of 18-DOP bounding volumes. and slightly better than oriented bounding boxes in this case.

Figures 4.18 and 4.20 contains the collision test results for a tree model (1.9 layers per shell on average) colliding with a Volkswagen Beetle car model (1.6 layers per shell on average). This test demonstrates the algorithm's ability to handle all model types. Even absolutely unstructured polygon soups, as used for the leaves within the tree model, can be handled the same way as with all other types of representations without any problems.

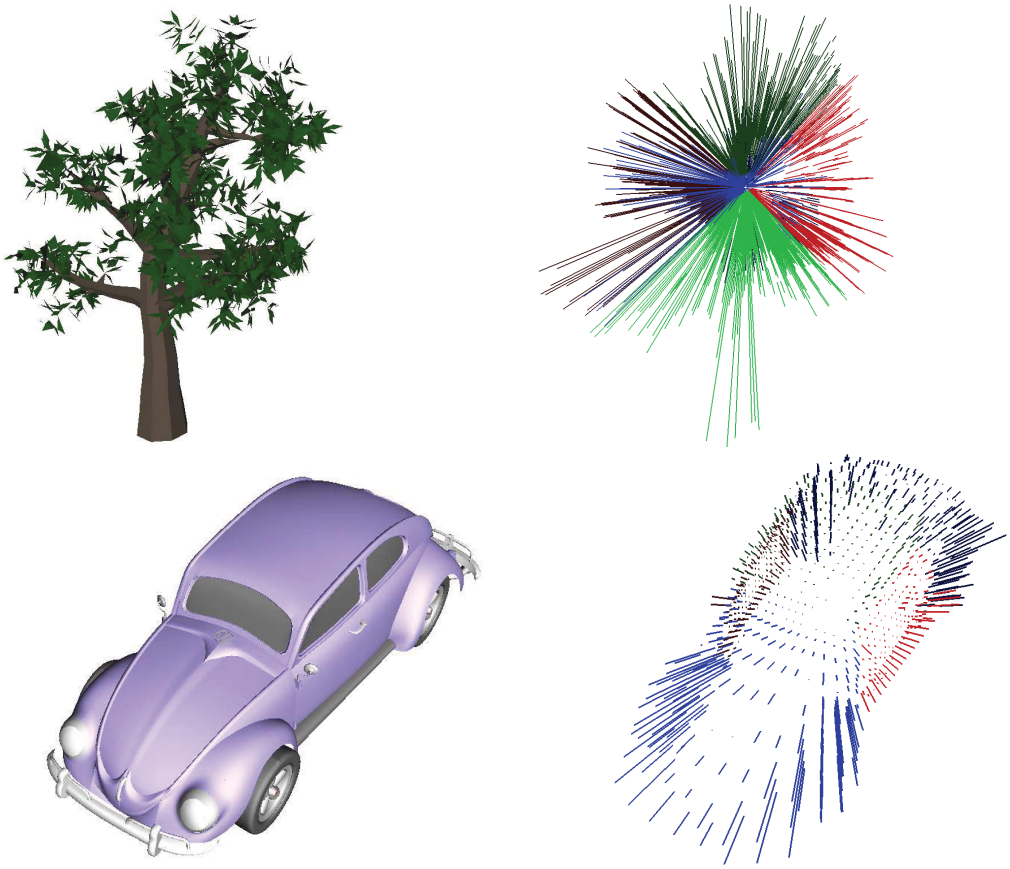


Figure 4.18: The second benchmark uses a tree model (4316 triangles) and a Volkswagen Beetle model (57243 triangles).

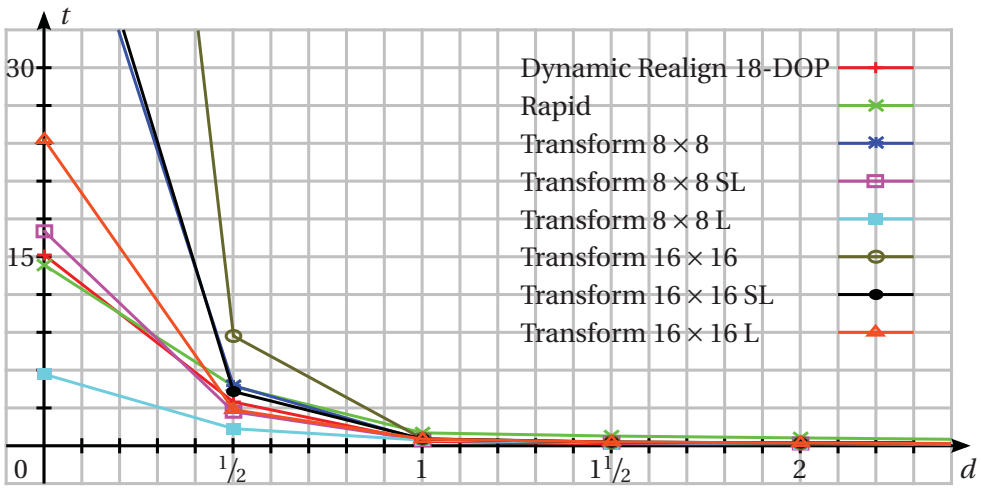


Figure 4.19: The corresponding collision time timings t [ms] of the "ant-spider" collision tests with varying distance d.

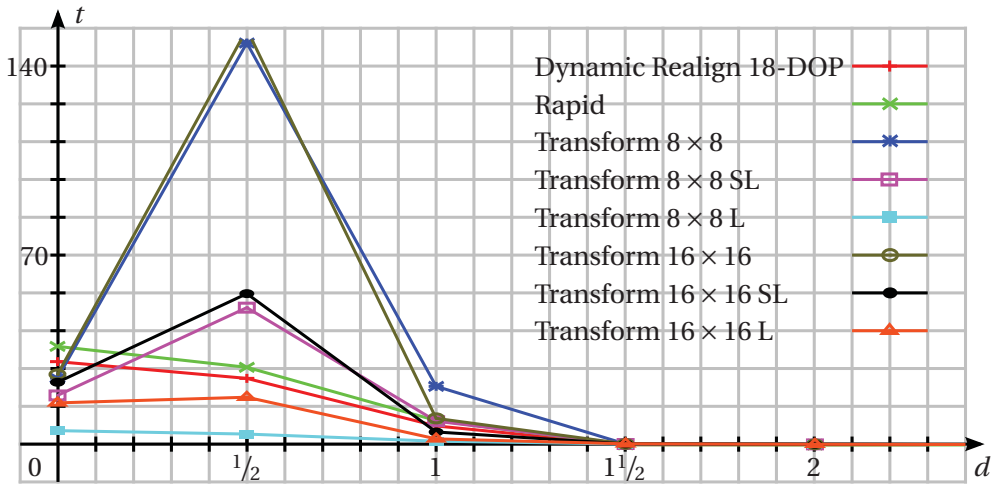


Figure 4.20: The corresponding collision time timings t [ms] of the "tree-beetle" collision tests with varying distance d.

The last benchmark series performs a collision test between the Statue of Liberty model (7.8 layers per shell on average) and the Tower of Pisa (8.4 layers per shell on average). These results (see Figures 4.21 and 4.22) should be compared with those for the tree/Beetle model. If reporting just a single triangle per spherical shell, the timings do not vary much, although the model complexity is much higher. In this mode of collision determination, collision times are independent of the triangle count. Collision time depends only on the sampling density and the volume between the inner and outer bounding shell. The models have their largest extent along the y-axis but none of the bounding volume hierarchy approaches can take advantage of this in the close-proximity situation (center distance 0). For center distance $1/2$, there are fewer spherical shell tests and more triangle to triangle tests compared to center distance 0. It is interesting that the transition from center distance $1/2$ to 0 can be used to assess the sampling density.

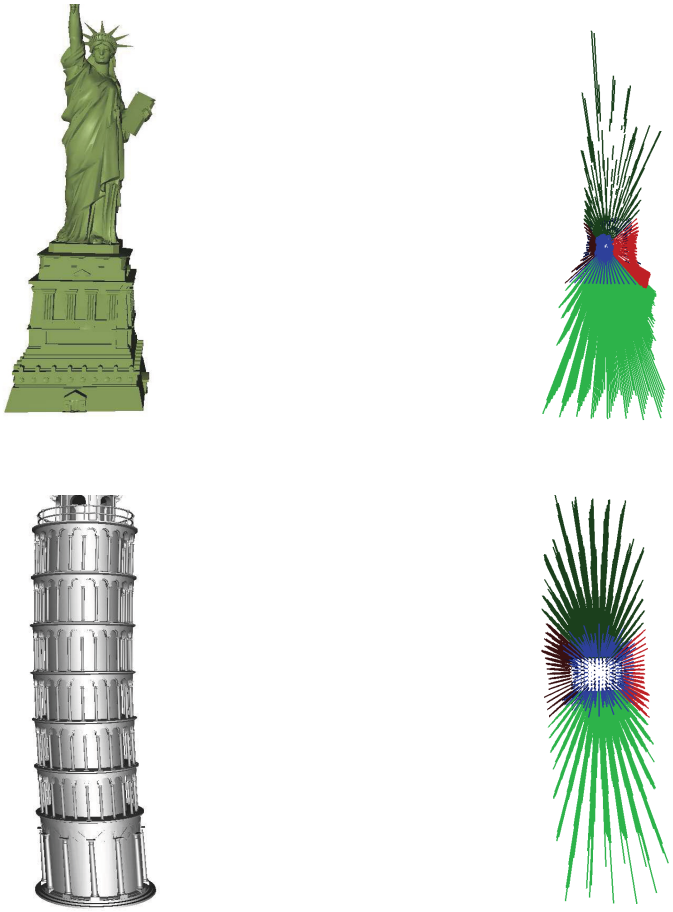


Figure 4.21: The "Statue of Liberty" model consists of 42 225 triangles. In the third benchmark it is tested against the "Tower of Pisa" (153 495 triangles).

For the triangle sizes in the models tree and Beetle the approximation with 16×16 samples per side is sufficient. For the small triangle sizes in the models Liberty and Tower of Pisa and testing all triangle pairs, the approximation with 8×8, 16×16 samples per side is not sufficient, and 32×32 samples still gives better results.

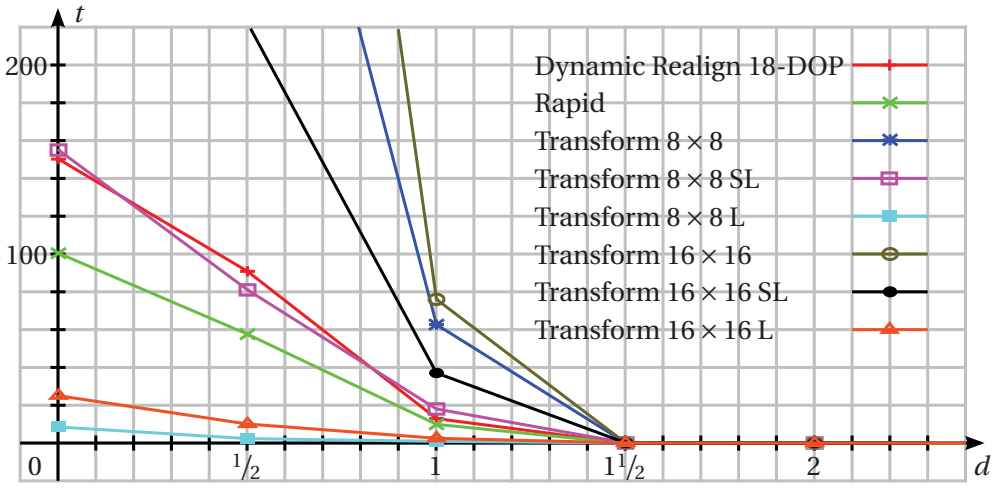


Figure 4.22: The corresponding collision time timings t [ms] of the "statue-tower" collision tests with varying distance d.

4.3 Subdivision Surfaces

An important research topic in computer-aided geometric design (CAGD) is model representation. The less parameters a model has, the "higher" its description is. Concerning this aspect subdivision surfaces, which are based on Bézier and B-spline techniques, play an important role.

4.3.1 Bézier and B-Spline Techniques

A simple modeling approach uses interpolation, which is a method of fitting a curve to a discrete set of known data points; i.e. for a set of points $\{P_0,\ldots,P_n\} \subset \mathbb{E}^3$ with parameters t_i, an interpolating curve c fulfills the condition

$$\mathbf{c}(t_i) = P_i, \qquad i = 0,\ldots,n. \tag{4.34}$$

This interpolation problem can be solved by the Lagrange[3] interpolation formula. It constructs polynomials

$$L_i^n(t) = \frac{(t-t_0)\cdot(t-t_1)\cdot\ldots\cdot(t-t_n)}{(t_i-t_0)\cdot\ldots\cdot(t_i-t_{i-1})\cdot(t_i-t_{i+1})\cdot\ldots\cdot(t_i-t_n)}, \tag{4.35}$$

which satisfy

$$L_i^n(t_k) = \delta_{ik}. \tag{4.36}$$

Therefore, the curve

$$\mathbf{c}(t) = \sum_{i=0}^{n} P_i L_i^n(t) = \sum_{i=0}^{n} P_i \prod_{j=0, j\neq i}^{n} \frac{t-t_j}{t_i-t_j} \tag{4.37}$$

solves the interpolation problem. Unfortunately, Lagrangian polynomials tend to oscillate. The more data points P_i the interpolation uses, the higher the degree of the resulting polynomial. A polynomial of high degree exhibits a great oscillation between the data points and is sensitive to changes or noise in the input data set (see Figure 4.23). Consequently, Lagrangian polynomials are not appropriate for CAGD modeling. To overcome these disadvantages Pierre Bézier[4] and Paul de Casteljau[5] developed a polynomial curve representation suitable for geometric modeling.

[3] Joseph-Louis de Lagrange (January 25, 1736 – April 10, 1813) Joseph-Louis de Lagrange, born Giuseppe Lodovico Lagrangia in Turin, was an Italian mathematician and astronomer who made significant contributions to analysis, number theory, and to classical and celestial mechanics.

[4] Pierre Étienne Bézier (September 1, 1910 – November 25, 1999) Pierre Étienne Bézier was a French engineer working for Renault from 1933–1975. His contributions to computer graphics and interactive techniques has influenced computer-aided design significantly [Rab02].

[5] Paul de Faget de Casteljau (November 19, 1930) Paul de Faget de Casteljau is a French physicist and mathematician. During his time at Citroën [dC99] he developed an algorithm for the computation of a Bézier curve.

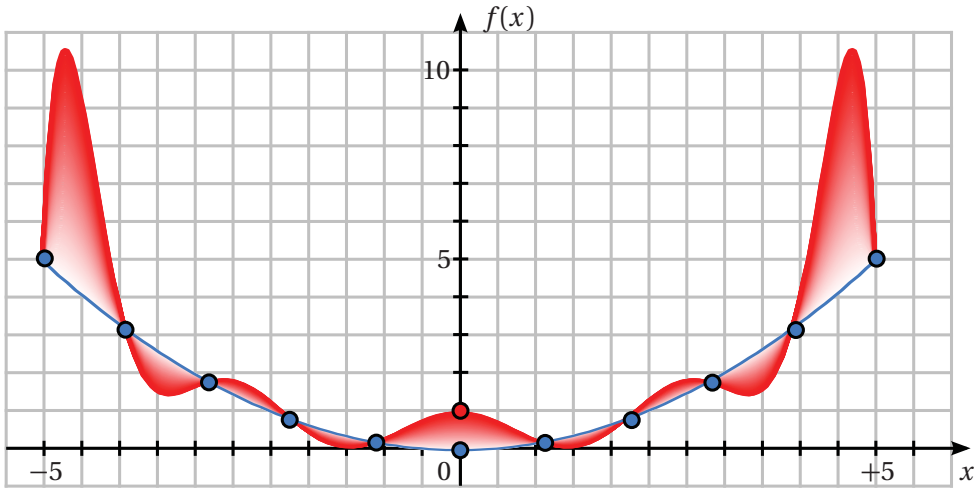


Figure 4.23: Lagrangian polynomials solve the interpolation problem; i.e. it constructs a function f such that $f(x_i) = y_i$ for a set of data points $(x_i,y_i), i = 0,\ldots,n$ (blue). The resulting polynomial has a high degree, exhibits a great oscillation between the data points and is sensitive to changes within the input data set. The red curves are the Lagrangian interpolations, if the point at the origin (blue) is exchanged by the red point.

The polynomials

$$B_i^n(t) = \binom{n}{i} t^i \cdot (1-t)^{n-i}, \qquad t \in [0,1] \tag{4.38}$$

of degree n are called Bernstein polynomials. For $i = 0,\ldots,n$ the $n+1$ polynomials B_i^n form a basis for the power polynomials of degree n and have a number of useful properties.

Positivity Each polynomial $B_i^n(t)$ is greater than or equal to zero over the parameter domain $[0,1]$:

$$\forall\, t \in [0,1]: \; B_i^n(t) \geq 0. \tag{4.39}$$

Partition of Unity For $t \in [0,1]$ the Bernstein polynomials are normalized:

$$\forall\, t \in [0,1]: \; \sum_{i=0}^{n} B_i^n(t) = 1. \tag{4.40}$$

Symmetry The Bernstein polynomials meet a symmetry equation:

$$\forall\, t \in [0,1]: \; B_i^n(t) = B_{n-i}^n(1-t). \tag{4.41}$$

Recursion Each Bernstein polynomial of degree n can be expressed via polynomials of degree $n-1$:

$$\forall\, t \in [0,1]: \; B_i^n(t) = t \cdot B_{i-1}^{n-1}(t) + (1-t) \cdot B_i^{n-1}(t). \tag{4.42}$$

Bernstein polynomials are the mathematical foundation of Bézier curves, which are defined by

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) \tag{4.43}$$

with parameter domain $t \in [0,1]$. The vectors $\mathbf{b}_i$ describe the positions of the control points / Bézier points that form the control polygon respectively the Bézier polygon. These curves have important properties, which can be deduced from the Bernstein polynomials' properties.

- A Bézier curve approximates its control polygon.
- As to the Bernstein polynomials partition the unity, the Bézier curves are invariant to affine transformations.
- Any point $p(t)$, $t \in [0,1]$ lies within the convex hull of the Bézier points.

Vector Fonts

In typography, a font is a complete character set of a typeface. In computer science two font concepts are used: bitmap fonts and vector fonts. A bitmap font describes each character by a matrix of pixels, whereas a vector font uses stroke definitions or Bézier curves.

A PostScript font [Inc85] uses cubic Bézier curves; i.e. a glyph is defined by its outline via a set of Bézier curves (see Figure 4.24). Therefore, each curve consists of four control points, of which the first (red) and the last (red) one are interpolated. The inner points (blue) define the tangents at the start respectively at the end.

The main advantage of vector fonts is its resolution independence. In contrast to bitmap fonts, vector images can be rendered at arbitrary resolution without artifacts.

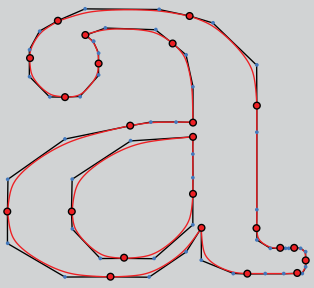


Figure 4.24: A vector font describes a character via its outline. This letter 'a' consists of 23 cubic Bézier curves.

- The derivation of a Bézier curve is a polynomial and can be written in Bézier form as well:

$$\mathbf{p}^{(k)}(t) = \frac{n!}{(n-k)!} \sum_{i=0}^{n-k} \Delta^k \mathbf{b}_i B_i^{n-k}(t), \tag{4.44}$$

$t \in [0,1]$, whereas the forward difference Δ^k is defined recursively by

$$\Delta^0 \mathbf{b}_i = \mathbf{b}_i, \tag{4.45}$$
$$\Delta^k \mathbf{b}_i = \Delta^{k-1} \mathbf{b}_{i+1} - \Delta^{k-1} \mathbf{b}_i. \tag{4.46}$$

- Consequently, the curve interpolates the first control point ($\mathbf{p}(0) = \mathbf{b}_0$) and the last one ($\mathbf{p}(1) = \mathbf{b}_n$) and the tangent vectors at these positions are simply

$$\mathbf{p}'(0) = n \cdot (\mathbf{b}_1 - \mathbf{b}_0), \qquad \mathbf{p}'(1) = n \cdot (\mathbf{b}_n - \mathbf{b}_{n-1}). \tag{4.47}$$

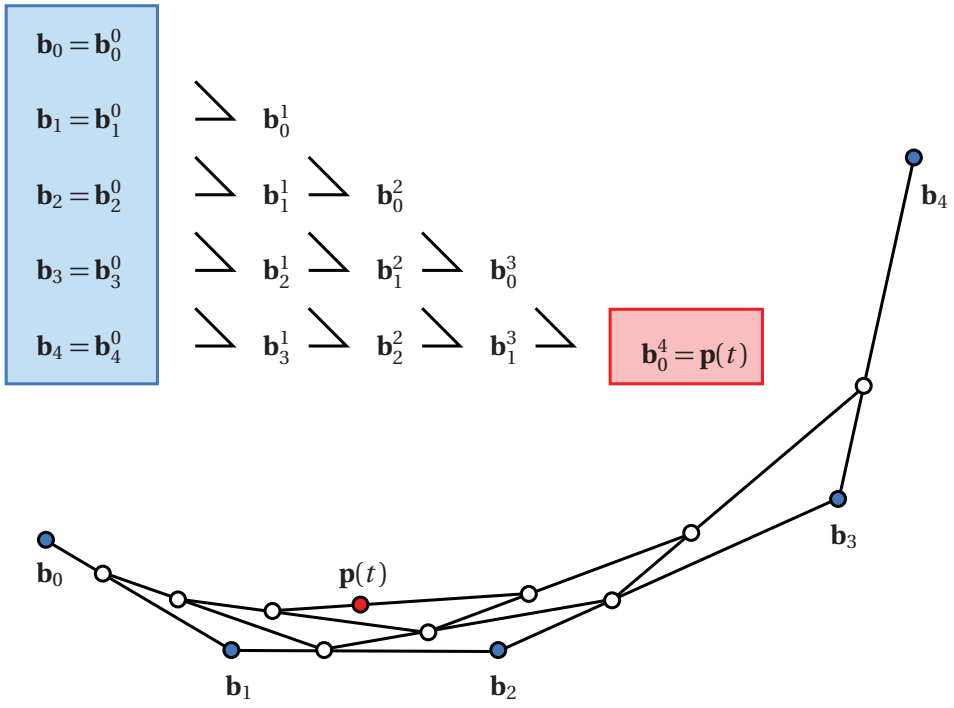


Figure 4.25: The de Casteljau algorithm uses the recursion of Bernstein polynomials to evaluate a Bézier curve at parameter $t \in [0,1]$. This process subdivides each line segment of the control polygon in a fixed ratio $t : (1-t)$ and introduces a new point. All generated points form a new control polygon, which is subdivided itself. This subdivision process is repeated until only the last control polygon is reduced to one point $\mathbf{p}(t)$. The original control polygon and all intermediate points form the de Casteljau scheme (upper left).

A Bézier curve can be evaluated by the de Casteljau algorithm, which is based on the recursion

$$\mathbf{b}_i^0(t) = \mathbf{b}_i \tag{4.48}$$

$$\mathbf{b}_i^r(t) = t \cdot \mathbf{b}_{i+1}^{r-1}(t) + (1-t) \cdot \mathbf{b}_i^{r-1}(t). \tag{4.49}$$

For any parameter $t \in [0,1]$ the algorithm determines $\mathbf{b}_0^n(t) = \mathbf{p}(t)$. A geometric interpretation of the de Casteljau scheme is illustrated in Figure 4.25. The scheme does not only evaluate a parameter, but it also subdivides a Bézier curve into two subcurves. Their control vertices are in the bottom line and in the diagonal line of the evaluation scheme. As the control polygons of the subdivided curves converge quickly towards the original curve, the subdivision process is a fast alternative to naive evaluation.

If, during the modeling process, more control vertices, respectively more degrees of freedom, are needed, it is helpful to increase the degree of a Bézier curve without changing its shape. The control vertices $\mathbf{b}_i^\star$ of a degree elevated curve are:

$$\mathbf{b}_0^\star = \mathbf{b}_0, \tag{4.50}$$

$$\mathbf{b}_j^\star = \frac{j}{n+1}\mathbf{b}_{j-1} + \left(1 - \frac{j}{n+1}\right)\mathbf{b}_j, \qquad j = 1,\ldots,n \tag{4.51}$$

$$\mathbf{b}_{n+1}^\star = \mathbf{b}_n. \tag{4.52}$$

As Bézier curves can be interpreted geometrically, which allows a modeler to predict its shape easily, they are widely used in computer graphics to model smooth curves. An undesirable property of Bézier curves is the fact that moving a single control point changes the global shape of the curve. This can be avoided with a generalization of a Bézier curve: the B-spline.

Its basis functions are defined recursively. Let $n \ll m$ and

$$T = \{t_0 \leq \cdots \leq t_n \leq \cdots \leq t_{n+m+1}\} \tag{4.53}$$

be a nondecreasing sequence of knots, then the basis functions of degree n are

$$N_i^0(t) = \begin{cases} 1, & \text{if } t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{4.54}$$

$$N_i^r(t) = \frac{t - t_i}{t_{i+r} - t_i} N_i^{r-1}(t) + \frac{t_{i+1+r} - t}{t_{i+1+r} - t_{i+1}} N_{i+1}^{r-1}(t) \tag{4.55}$$

for $1 \leq r \leq n$. If knots coincide, a common convention is to evaluate $\frac{0}{0} = 0$, so that the definition remains valid without having to formulate special rules. These basis functions have important properties.

- $N_i^n(t)$ consists piecewise of polynomials of degree n.
- Each basis function $N_i^n(t)$ has local support; i.e.

$$\forall\, t \notin [t_i, t_{i+n+1}) : \; N_i^n(t) = 0 \tag{4.56}$$

  and is semi-positive

$$\forall\, t \in [t_0, t_{m+n+1}) : \; N_i^n(t) \geq 0. \tag{4.57}$$

- The basis functions are normalized and sum up to unity.
- If t_j is a simple knot ($t_{j-1} < t_j < t_{j+1}$), then $N_i^n(t_j)$ is C^{n-1}-continuous. In case of a non-simple knot with multiplicity μ ($s = t_{j+1} = \cdots = t_{j+\mu}$) the normalized B-spline N_i^n of degree n is at least $C^{n-\mu}$-continuous.
- As a generalization of Bernstein polynomials, B-splines are downwardly compatible. Setting

$$T = (\underbrace{0,\ldots,0}_{n+1},\underbrace{1,\ldots,1}_{n+1}) \tag{4.58}$$

  turns B-splines into Bernstein polynomials over T.

Using the knot vector $T = (t_0 \leq \cdots \leq t_n \leq \cdots \leq t_{m+n+1})$, and the basis functions N_i^n a B-spline curve of degree n is defined by

$$\mathbf{p}(t) = \sum_{i=0}^{m} \mathbf{d}_i N_i^n(t), \qquad t \in [t_n, t_{m+1}]. \tag{4.59}$$

The control points $\mathbf{d}_0,\ldots,\mathbf{d}_m \in \mathbb{R}^3$ are called de Boor points named after CARL DE BOOR[6]. B-splines whose knots meet the condition $t_0 = 0$ and

$$t_{i+1} = t_i \qquad \text{or} \qquad t_{i+1} = t_1 + 1, \qquad (i = 0,\ldots,n+m) \tag{4.60}$$

are called uniform. The properties of the basis functions result in fundamental properties of B-splines.

- Due to the limited support of N_i^n, the i^{th}de Boor point affects the B-spline curve only within the parameter domain $[t_i, t_{i+n+1})$. Conversely, the shape of the curve over the parameter domain $[t_i, t_{i+1})$ is only affected by the points $\mathbf{d}_{i-n},\ldots,\mathbf{d}_i$.
- For the parameter $t_l \leq t \leq t_{l+1}$ the curve $\mathbf{p}(t)$ lies within the convex hull of the $n+1$ de Boor points $\mathbf{d}_{l-n},\ldots,\mathbf{d}_l$.
- If n control points $\mathbf{d}_{l-n+1} = \cdots = \mathbf{d}_l = \mathbf{d}$ coincide, then the curve passes these points $\mathbf{p}(t_{l+1}) = \mathbf{d}$.
- If n knots $t_{l+1} = \cdots = t_{l+n} = t$ coincide, then the curve passes the de Boor point $\mathbf{p}(t) = \mathbf{d}_l$. Especially, if the knot vector starts and ends with multiplicity $n+1$, the curve interpolates the control polygon tangentially at its ends.

[6] CARL WILHELM REINHOLD DE BOOR (December 3, 1937) Carl R. de Boor is a German-American mathematician and professor emeritus at the University of Wisconsin–Madison. He made fundamental contributions to the theory of splines and numerous applications of splines.

Similar to Bézier curves B-splines can be evaluated via a recursion. Any B-spline curve

$$\mathbf{p}(t) = \sum_{i=0}^{m} \mathbf{d}_i N_i^n(t) \tag{4.61}$$

of degree n with the knot vector $T = (t_0, \ldots, t_{m+n+1})$ can be evaluated for $t_l \leq t \leq t_{l+1}$ by

$$\mathbf{d}_i^0(t) = \mathbf{d}_i, \tag{4.62}$$

$i = 0, \ldots, m$, and

$$\mathbf{d}_i^r(t) = \left(1 - \frac{t - t_{i+r}}{t_{i+n+r} - t_{i+r}}\right) \mathbf{d}_i^{r-1}(t) + \frac{t - t_{i+r}}{t_{i+n+r} - t_{i+r}} \mathbf{d}_{i+1}^{r-1}(t), \tag{4.63}$$

$i = l-n, \ldots, l-r$ and $0 \leq r \leq n$. The curve point is then $\mathbf{p}(t) = \mathbf{d}_{l-n}^n$. This algorithm, the de Boor algorithm, is visualized in Figure 4.26 for a cubic B-spline. As cubic, uniform B-splines are very common, it is convenient to simplify and unroll the recursion. The basis functions simplify to

$$N_i = {}^1\!/_6 t^3, \tag{4.64}$$

$$N_{i-1} = {}^1\!/_6(-3t^3 + 3t^2 + 3t + 1), \tag{4.65}$$

$$N_{i-2} = {}^1\!/_6(3t^3 - 6t^2 + 4), \tag{4.66}$$

$$N_{i-3} = {}^1\!/_6(1-t)^3. \tag{4.67}$$

Together with the de Boor points $\mathbf{d}_{i-3}$, $\mathbf{d}_{i-2}$, $\mathbf{d}_{i-1}$, and $\mathbf{d}_i$ a simplified matrix representation of the B-spline curve is

$$\mathbf{p}(t) = \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}^{\mathrm{T}} \cdot \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{d}_{i-3} \\ \mathbf{d}_{i-2} \\ \mathbf{d}_{i-1} \\ \mathbf{d}_i \end{pmatrix} \tag{4.68}$$

Using a projective space B-splines can be generalized to non-uniform rational B-splines (NURBS) [FLS04], [Far99].

4.3.2 Tensor Product Surfaces

Based on the theory of curves, a surface can be constructed by sweeping a curve through space such that its control points move along curves. As shown in "Bézier and B-Spline techniques" [PBP02], these tensor product surfaces have properties analogous to the curves used for construction.

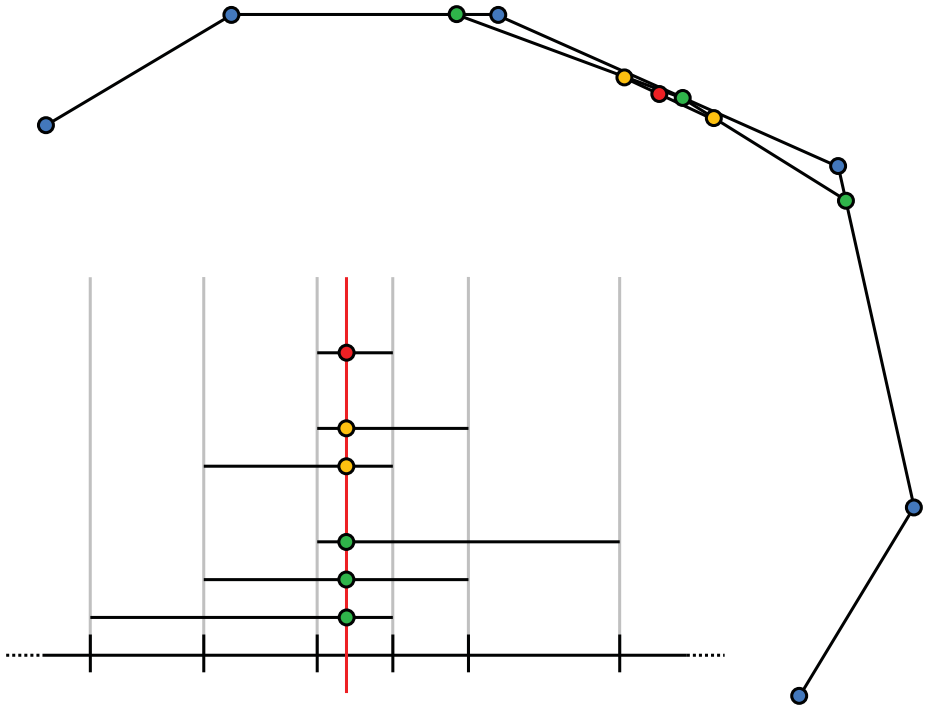


Figure 4.26: The de Boor algorithm generalizes the de Casteljau algorithm as B-spline curves generalize Bézier curves. The illustration of the evaluation process shows the local influence of control points (blue) on the resulting curve point (red). For each parameter interval only a subset of control points is used. Similar to the de Casteljau algorithm the line segments of the control polygon are subdivided, but the de Boor algorithm does not use a fixed ratio. It depends on the knot vector and the recursion level of the evaluation (lower left diagram).

In technical terms a surface is described by a curve

$$\mathbf{c}(u) = \sum_{i=0}^{m} \mathbf{a}_i A_i(u), \tag{4.69}$$

with functions $A_i(u), u \in [u_0, u_1]$ whose control points $\mathbf{a}_i = \mathbf{a}_i(v)$ are defined by other curves

$$\mathbf{a}_i(v) = \sum_{j=0}^{n} \mathbf{b}_{i,j} B_j(v), \tag{4.70}$$

with parameter domain $v \in [v_0, v_1]$. Consequently, a surface is defined by

$$\mathbf{s}(u,v) = \sum_i \sum_j \mathbf{b}_{i,j} A_i(u) B_j(v), \tag{4.71}$$

$$(u,v) \in [u_0, u_1] \times [v_0, v_1]. \tag{4.72}$$

If the functions A_i and B_j are Bernstein polynomials, then the resulting tensor product surface is called Bézier surface. In the same manner, a B-spline surface consists of basis functions N_i^n.

One disadvantage of tensor product surfaces is that all patches are based on a rectangular topology (see Equation (4.71), (4.72), and Figure 4.27).

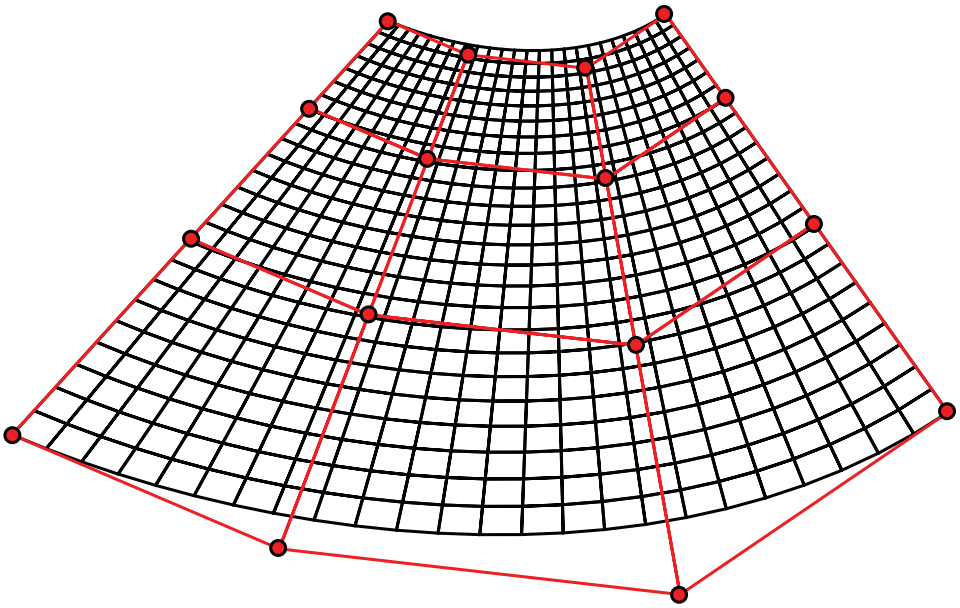


Figure 4.27: A tensor product surface is based on a curve representation. Consequently, this Bézier surface shares many properties (end point interpolation, convex hull property, etc.) with its corresponding curve type. A negative aspect of tensor product surfaces is the limitation to strictly rectangular topology (control mesh in red).

A common technique for more flexibility is trimming. A trimmed surface consists of the surface itself and an additional trimming curve in parameter space. A trimming curve is a closed 2D curve, which separates the parameter space into two parts: a valid part and an invalid part. The invalid part is removed from parameter space, respectively from 3D space [HT96]. Handling trimmed NURBS surfaces is quite difficult [LC09]. A mathematically more esthetic approach using subdivision has been presented by EDWIN CATMULL and JIM CLARK: "Recursively generated B-spline surfaces on arbitrary topological meshes" [CC78].

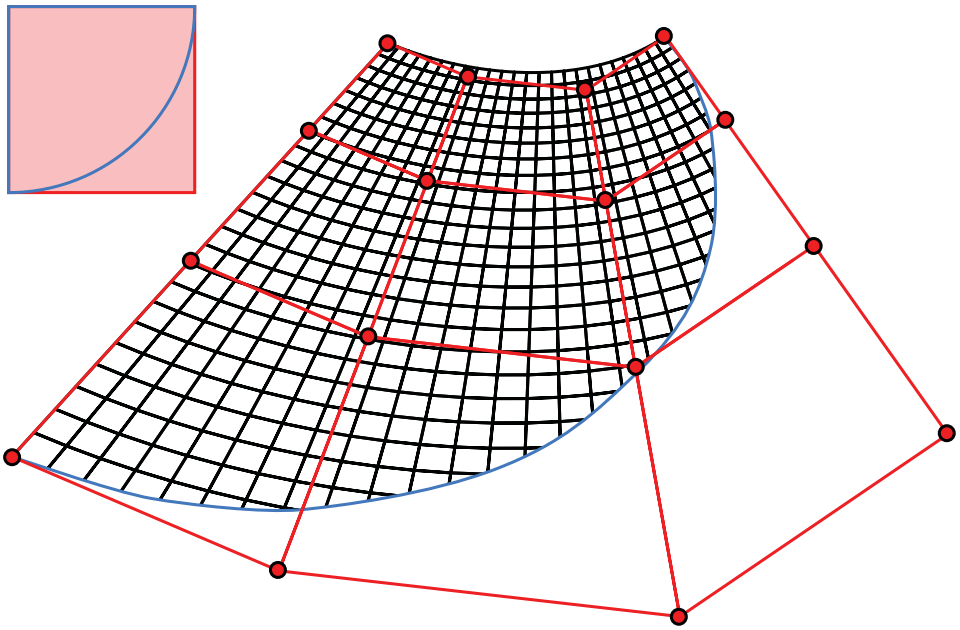


Figure 4.28: A trimmed Bézier / B-spline / NURBS surface consists of a parameter domain (upper left) and a set of control points. Furthermore, a closed curve (blue) within the parameter domain separates the domain into two parts: a valid part and an invalid part. The final surface in 3D only consists of those points whose parameters are valid [HT96].

4.3.3 Catmull-Clark Subdivision Surfaces

The Catmull-Clark subdivision scheme generalizes the evaluation of a bicubic B-spline patch to arbitrary topological meshes. A uniform, bicubic B-Spline patch consists of 16 control points

$$P = \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,0} & P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,0} & P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix} \tag{4.73}$$

and can be written

$$\mathbf{s}(u,v) = \mathbf{u}\, M\, P\, M^{\mathrm{T}}\, \mathbf{v}^{\mathrm{T}} \tag{4.74}$$

with monomial vectors

$$\mathbf{u} = \begin{pmatrix} 1 & u & u^2 & u^3 \end{pmatrix}, \tag{4.75}$$

$$\mathbf{v} = \begin{pmatrix} 1 & v & v^2 & v^3 \end{pmatrix}, \tag{4.76}$$

$(u,v) \in [0,1] \times [0,1]$ and the coefficient matrix

$$M = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}. \tag{4.77}$$

The subpatch corresponding to $(u,v) \in [0,{}^1\!/_2] \times [0,{}^1\!/_2]$ can be reparameterized by $u' = {}^u\!/_2$ and $v' = {}^v\!/_2$. The resulting surface is a bicubic B-spline patch. Its control points P' can be expressed in terms of the 16 control points P:

$$\mathbf{s}\left({}^u\!/_2, {}^v\!/_2\right) = \begin{pmatrix} 1 & \frac{u}{2} & (\frac{u}{2})^2 & (\frac{u}{2})^3 \end{pmatrix} M P M^{\mathrm{T}} \begin{pmatrix} 1 & \frac{v}{2} & (\frac{v}{2})^2 & (\frac{v}{2})^3 \end{pmatrix}^{\mathrm{T}}$$

$$= \begin{pmatrix} 1 & u & u^2 & u^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{8} \end{pmatrix} M P M^{\mathrm{T}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{8} \end{pmatrix}}_{=D} \begin{pmatrix} 1 \\ v \\ v^2 \\ v^3 \end{pmatrix}$$

$$= \mathbf{u}\, D\, M\, P\, M^{\mathrm{T}}\, D^{\mathrm{T}}\, \mathbf{v}^{\mathrm{T}} = \mathbf{u}\; M\, \underbrace{M^{-1}\, D\, M}_{=S}\, P\, M^{\mathrm{T}}\, D^{\mathrm{T}}\, (M^{-1})^{\mathrm{T}}\, M^{\mathrm{T}}\; \mathbf{v}^{\mathrm{T}} \tag{4.78}$$

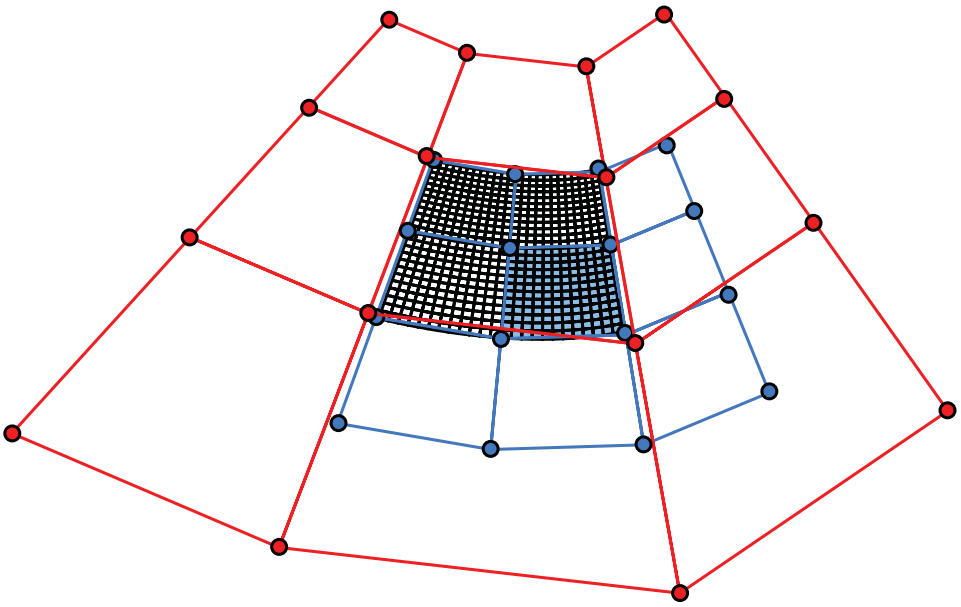


Figure 4.29: The Catmull-Clark subdivision scheme is based on the idea to describe a subpatch of a bicubic B-spline patch by a bicubic B-spline patch. The starting point is a bicubic patch (red), which generates a surface (wireframe in black), if evaluated over the domain $(u,v) \in [0,1] \times [0,1]$. Then the subsurface belonging to $[0,1/2] \times [0,1/2]$ is inspected and its corresponding control mesh (blue) is determined. The correspondences between the original mesh (red) and its subdivided version (blue) are the B-spline refinement rules. The Catmull-Clark subdivision scheme generalizes theses rules to arbitrary meshes.

The matrix product of the diagonal matrix D, the coefficient matrix M and its inverse M^{-1} is called splitting matrix S. It is constant and does not depend on the control points P:

$$S = \frac{1}{8} \begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \end{pmatrix}. \tag{4.79}$$

Consequently, the new control points are

$$P' = \frac{1}{8} \begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \end{pmatrix} \cdot \begin{pmatrix} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,0} & P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,0} & P_{3,1} & P_{3,2} & P_{3,3} \end{pmatrix} \cdot \frac{1}{8} \begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \end{pmatrix}^{\mathrm{T}}, \tag{4.80}$$

respectively

$$P'_{0,0} = \frac{1}{4}(P_{0,0} + P_{1,0} + P_{0,1} + P_{1,1}) \tag{4.81}$$

$$P'_{0,1} = \frac{1}{16}(P_{0,0} + P_{1,0} + 6(P_{0,1} + P_{1,1}) + P_{0,2} + P_{1,2}) \tag{4.82}$$

$$P'_{0,2} = \frac{1}{4}(P_{0,1} + P_{1,1} + P_{0,2} + P_{1,2}) \tag{4.83}$$

$$P'_{0,3} = \frac{1}{16}(P_{0,1} + P_{1,1} + 6(P_{0,2} + P_{1,2}) + P_{0,3} + P_{1,3}) \tag{4.84}$$

$$P'_{1,0} = \frac{1}{16}(P_{0,0} + P_{0,1} + 6(P_{1,0} + P_{1,1}) + P_{2,0} + P_{2,1}) \tag{4.85}$$

$$P'_{1,1} = \frac{1}{64}(P_{0,0} + 6P_{1,0} + P_{2,0} + 6(P_{0,1} + 6P_{1,1} + P_{2,1}) + P_{0,2} + 6P_{1,2} + P_{2,2}) \tag{4.86}$$

$$P'_{1,2} = \frac{1}{16}(P_{0,1} + P_{0,2} + 6(P_{1,1} + P_{1,2}) + P_{2,1} + P_{2,2}) \tag{4.87}$$

$$P'_{1,3} = \frac{1}{64}(P_{0,1} + 6P_{1,1} + P_{2,1} + 6(P_{0,2} + 6P_{1,2} + P_{2,2}) + P_{0,3} + 6P_{1,3} + P_{2,3}) \tag{4.88}$$

$$P'_{2,0} = \frac{1}{4}(P_{1,0} + P_{2,0} + P_{1,1} + P_{2,1}) \tag{4.89}$$

$$P'_{2,1} = \frac{1}{16}(P_{1,0} + P_{2,0} + 6(P_{1,1} + P_{2,1}) + P_{1,2} + P_{2,2}) \tag{4.90}$$

$$P'_{2,2} = \frac{1}{4}(P_{1,1} + P_{2,1} + P_{1,2} + P_{2,2}) \tag{4.91}$$

$$P'_{2,3} = \frac{1}{16}(P_{1,1} + P_{2,1} + 6(P_{1,2} + P_{2,2}) + P_{1,3} + P_{2,3}) \tag{4.92}$$

$$\tag{4.93}$$

and

$$P'_{3,0} = \frac{1}{16}(P_{1,0} + P_{1,1} + 6(P_{2,0} + P_{2,1}) + P_{3,0} + P_{3,1}) \tag{4.94}$$

$$P'_{3,1} = \frac{1}{64}(P_{1,0} + 6P_{2,0} + P_{3,0} + 6(P_{1,1} + 6P_{2,1} + P_{3,1}) + P_{1,2} + 6P_{2,2} + P_{3,2}) \tag{4.95}$$

$$P'_{3,2} = \frac{1}{16}(P_{1,1} + P_{1,2} + 6(P_{2,1} + P_{2,2}) + P_{3,1} + P_{3,2}) \tag{4.96}$$

$$P'_{3,3} = \frac{1}{64}(P_{1,1} + 6P_{2,1} + P_{3,1} + 6(P_{1,2} + 6P_{2,2} + P_{3,2}) + P_{1,3} + 6P_{2,3} + P_{3,3}). \tag{4.97}$$

The main idea of Edwin Catmull and Jim Clark is to rewrite these rules in terms of face, edge and vertex points and to generalize them.

Face point The average of four points that bound a face, for example $(P_{0,0}, P_{1,0}, P_{0,1}, P_{1,1})$, form a so-called face point. In general, a face point is

$$F_{i,j} = \frac{1}{4}(P_{i,j} + P_{i+1,j} + P_{i,j+1} + P_{i+1,j+1}) \tag{4.98}$$

Using the face points on the right-hand side expressions, the subdivision rules simplify to:

$$P'_{0,0} = F_{0,0} \tag{4.99}$$

$$P'_{0,1} = \frac{1}{4}(F_{0,0} + F_{0,1} + P_{0,1} + P_{1,1}) \tag{4.100}$$

$$P'_{0,2} = F_{0,1} \tag{4.101}$$

$$P'_{0,3} = \frac{1}{4}(F_{0,0} + F_{1,0} + P_{0,2} + P_{1,2}) \tag{4.102}$$

...

Edge point An edge point is the average of

- two points that define an edge and
- two new face points of the faces sharing the edge.

Consequently, the edge point is

$$E_{i,j} = \frac{1}{4}(F_{i,j-1} + F_{i,j} + P_{i,j} + P_{i+1,j}) \tag{4.103}$$

or

$$E_{i,j} = \frac{1}{4}(F_{i-1,j} + F_{i,j} + P_{i,j} + P_{i,j+1}). \tag{4.104}$$

The subdivision rules now simplify to:

$$P'_{0,0} = F_{0,0} \tag{4.105}$$

$$P'_{0,1} = E_{0,1} \tag{4.106}$$

$$P'_{0,2} = F_{0,1} \tag{4.107}$$

$$P'_{0,3} = E_{0,2} \tag{4.108}$$

and

$$P'_{1,0} = E_{1,0} \tag{4.109}$$

$$P'_{1,1} = \frac{1}{16}(F_{0,0} + F_{0,1} + F_{1,0} + F_{1,1} + P_{1,0} + P_{0,1} + 8P_{1,1} + P_{2,1} + P_{1,2}) \tag{4.110}$$

$$P'_{1,2} = E_{1,2} \tag{4.111}$$

$$P'_{1,3} = \frac{1}{16}(F_{0,1} + F_{0,2} + F_{1,1} + F_{1,2} + P_{1,1} + P_{0,2} + 8P_{1,2} + P_{2,2} + P_{1,3}) \tag{4.112}$$

...

Vertex point Last but not least, the average of

- the face points of the faces adjacent to a vertex point,
- the midpoints of the edges adjacent to a vertex point, and
- the corresponding vertex

form a new vertex point. For example, $P'_{1,3}$ consists of $\frac{Q+2R+S}{4}$ with

$$Q = \frac{1}{4}(F_{0,1} + F_{0,2} + F_{1,1} + F_{1,2}) \tag{4.113}$$

$$R = \frac{1}{4}\left(\frac{P_{0,2}+P_{1,2}}{2} + \frac{P_{1,1}+P_{1,2}}{2} + \frac{P_{1,3}+P_{1,2}}{2} + \frac{P_{2,2}+P_{1,2}}{2}\right) \tag{4.114}$$

$$S = P_{1,2}. \tag{4.115}$$

In general, EDWIN CATMULL and JIM CLARK reformulated the evaluation of a B-spline patch as a subdivision process. By construction the subdivision process converges to a limit surface, which is a B-spline patch – at least in the regular case. The most-general case may lead to irregular configurations with vertices having a valence not equal to four. These configurations do not have a counterpart B-spline patch.

In a sequence of subdivision steps, a point P is mapped to P', P'', … and converges to a limit point $\tilde{P}$. In a regular configuration $\tilde{P}$ can be evaluated directly via the counterpart B-spline patch using Equation (4.74):

$$\mathbf{s}(0,0) = \frac{1}{36}\left(P_{0,0} + 4P_{0,1} + P_{0,2} + 4P_{1,0} + 16P_{1,1} + 4P_{1,2} + P_{2,0} + 4P_{2,1} + P_{2,2}\right) \tag{4.116}$$

$$\mathbf{s}_u(0,0) = \frac{1}{12}\left(-P_{0,0} - 4P_{0,1} - P_{0,2} + P_{2,0} + 4P_{2,1} + P_{2,2}\right) \tag{4.117}$$

$$\mathbf{s}_v(0,0) = \frac{1}{12}\left(-P_{0,0} + P_{0,2} - 4P_{1,0} + 4P_{1,2} - P_{2,1} + P_{2,2}\right) \tag{4.118}$$

The limit points of irregular configurations can be determined via eigenanalysis [Sta98]. A summary of commonly used Catmull-Clark subdivision rules including special rules for border and crease configurations is listed in Table 4.4. An overview on subdivision surfaces in general has been presented by WEIYIN MA "Subdivision surfaces for CAD – an overview" [Ma05].

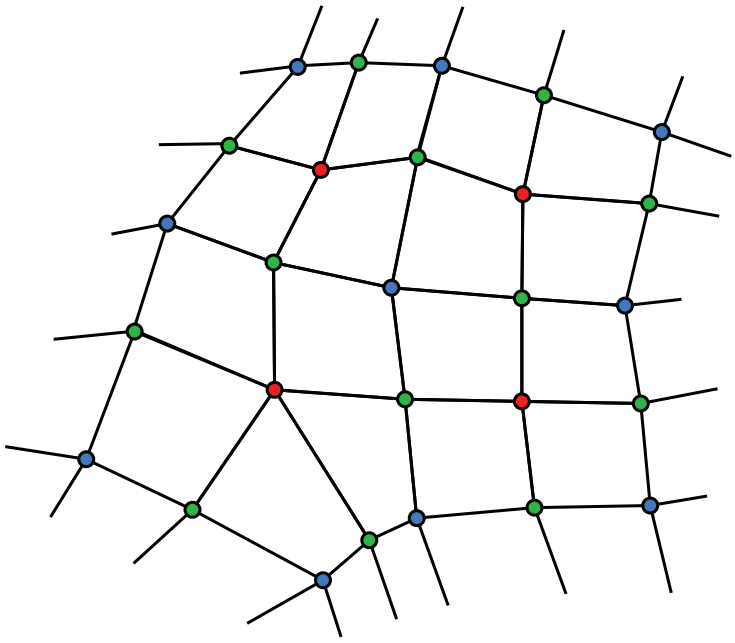
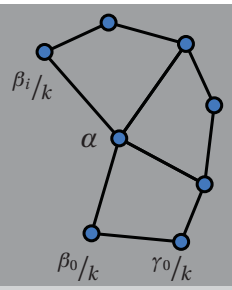


Figure 4.30: The Catmull-Clark subdivision scheme refines a mesh. In each iteration step it introduces new face points (red) and new edge points (green). Furthermore, it modifies the vertex points (blue). The result is always a quad mesh.

Catmull-Clark Subdivision Surfaces

The Catmull-Clark subdivision scheme generalizes a B-spline evaluation to meshes with arbitrary topology. For a vertex with valence k the following subdivision and limit point rules / weights apply.



classification	α	β_i	γ_i
subdivision rules			
inner vertex	$1-\frac{3}{2k}-\frac{1}{4k}$	$\frac{3}{2k}$	$\frac{1}{4k}$
border vertex / crease vertex	$\frac{3}{4}$	$\frac{1}{8}$, along border 0, otherwise	0
corner vertex	1	0	0
limit point rules			
inner vertex	$1-\frac{5}{k+5}$	$\frac{4}{k+5}$	$\frac{1}{k+5}$
border vertex / crease vertex	$\frac{2}{3}$	$\frac{1}{6}$, along border 0, otherwise	0
corner vertex	1	0	0
limit tangent rules $\mathbf{t}_1$			
inner vertex	0	$a(k)\cos\frac{2i\pi}{k}$	$\cos\frac{2i\pi}{k}+\cos\frac{2(i+1)\pi}{k}$
border vertex / crease vertex	0	±1, along border 0, otherwise	0
limit tangent rules $\mathbf{t}_2$			
inner vertex	0	$a(k)\cos\frac{2(i+1)\pi}{k}$	$\cos\frac{2(i+1)\pi}{k}+\cos\frac{2(i+2)\pi}{k}$

with $a(k)=1+\cos\frac{2\pi}{k}+\sqrt{2\left(9+\cos\frac{2\pi}{k}\right)}\cos\frac{\pi}{k}$

Table 4.4: The weights for a Catmull-Clark subdivision step, the corresponding limit point rules and its limit tangent rules are published in the original article by EDWIN CATMULL and JIM CLARK [CC78] and in subsequent research papers [HKD93], [Sta98], [BLZ00], [CRE01]. An overview (including this Table) and implementation details have been presented by TORSTEN TECHMANN [Tec04].

4.3.4 Distance Fields

The problem to determine the Euclidean distance between an arbitrary point in 3D and a free-form subdivision surface is fundamental in many different communities including computer-aided geometric design, robotics, computer graphics, and computational geometry. A lot of algorithms in the context of physical simulation, path planning, etc. have to determine this distance: an exemplary algorithm is the shape fitting approach by TORSTEN ULLRICH. An early version was based on subdivision surfaces and evaluated distances between a point cloud and some subdivision surfaces in order to fit a procedural model [UF07b]. As query time is always an issue, the goal is to choose the best algorithm for the application at hand [USK+07].

A subdivision surface is defined by an infinite subdivision process. In contrast to parametric surfaces which provide a finite evaluation algorithm, a subdivision surface may not come with a direct evaluation method at arbitrary parameter values. Currently, it can be evaluated via

Uniform subdivision If the subdivision rules are applied sufficiently often, the resulting mesh will be a tight approximation of the limit surface. For non-interpolating subdivision schemes, e.g., Catmull-Clark, the resulting mesh points will not lie on the limit surface in general. In order to decrease the deviation, limit point rules can calculate the point on the limit surface for a subdivision mesh point.

Adaptive subdivision Due to the exponential need of memory it is a good strategy to subdivide the mesh adaptively. This results in a subdivision process with varying subdivision depth but constant overall accuracy [MH00]. The use of limit point rules is essential for the connection of mesh parts with different subdivision depths.

Exact evaluation & conversion Stationary subdivision schemes, e.g., Catmull-Clark, allow an exact evaluation at arbitrary parameter values [Sta98]. JOS STAM makes use of the property that regular patches can be evaluated as uniform, bicubic B-spline patches. The region around irregular points shrinks successively when subdividing the irregular patches, and the eigenstructure of the subdivision matrix is used to determine the limit there. Sensible parametrizations for irregular patches ensure non-degenerate derivatives [BMZ04]. For Catmull-Clark subdivision, a regular quad patch can even be represented as a single bicubic Bézier patch.

For the problem of distance computation to subdivision surfaces, TORSTEN ULLRICH et al. [USK+07] propose the following classification of approaches:

Approaches based on the surface's distance field A separate scalar data structure reconstructs the (signed) distance to the closest point on the object [JBS06]. GPU approaches [BBVK04], which compute and evolve the distance field in a small narrow band around the object, also belong to this group.

Searching for surface primitives of the original representation The curved surface patches, which correspond to a face of the control mesh, are organized in a spatial data structure for the domain based on their bounding volume. Only this data structure has to be updated after model deformations. The spatial data structure is then traversed in increasing minimum distance to the query point, and the primitive's minimum distance is computed as a subproblem. A termination condition is necessary to stop the search with the correct distance value.

Searching for surface primitives derived from the original representation Instead of using the surface primitives of the original object representation directly, one derives a small set of simpler primitives from the original surface primitives. The reason could be that they offer a simpler minimum distance algorithm. In the case of subdivision surfaces, the surface's triangulation is often available also from other tasks.

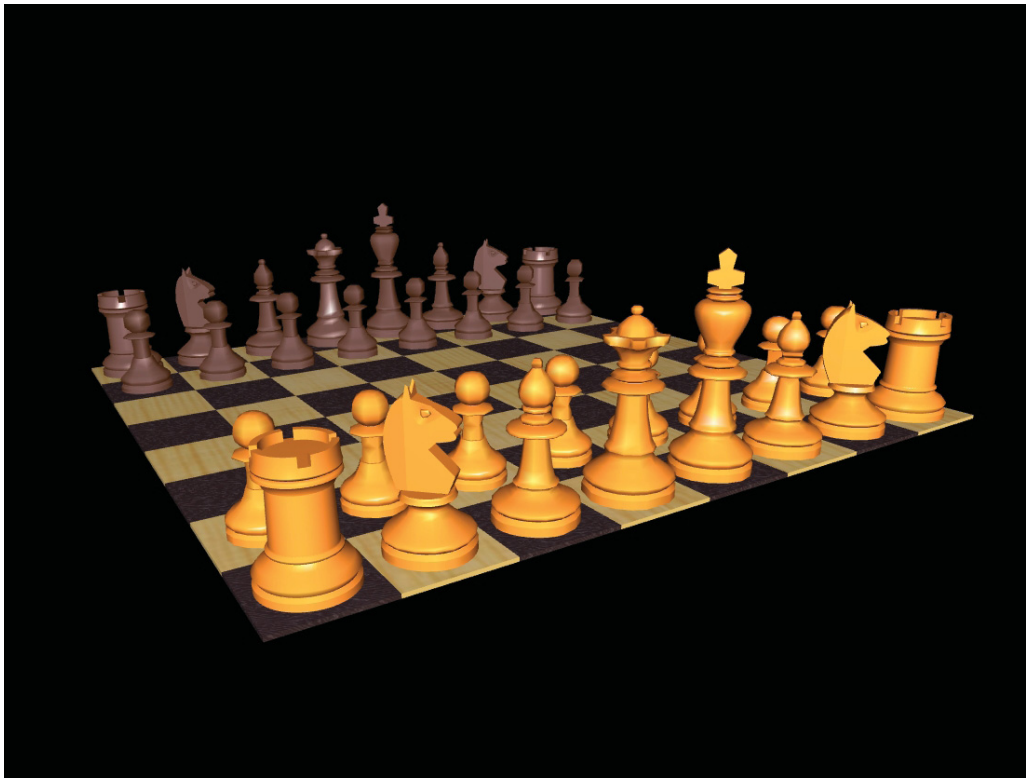


Figure 4.31: The distance between a point and a subdivision surface can be calculated in various ways. This benchmark uses chess figures modeled with subdivision surfaces. Each initial mesh has between 70 ("pawn") and 1 454 ("rook") polygons. The Figure shows all test pieces in their initial chess position.
The chess figures were created and provided by RENÉ BERNDT.

In "Distance Calculation between a Point and a Subdivision Surface", three kinds of algorithms to determine the distance between a query point and a subdivision surface are analyzed. The first group consists of three algorithms which use the triangulation of a subdivision surface. The next approach evaluates the subdivision surface on-the-fly. And the last algorithm converts it into Bézier patches. In this case distance queries are answered by a numerical minimization routine.

Uniform Triangulation

The most simple approach uses an uniform tessellation of the subdivision surface at a fixed depth to create a triangle mesh. For a tight approximation of the limit surface, the limit points of the control vertices have been used. For each query point the distance to each triangle is calculated [Jon95], and the minimum is selected. This approach does naive search without any spatial data structure.

pros The calculation is robust and its correctness can be verified easily.

cons As runtime and memory footprint of a single distance query are linear in the number of triangles and exponential in the subdivision depth, this algorithm is not useful for real world applications. The implementation has been used to verify the results of the following algorithms, but it is not considered to be a practical solution.

Hashed Triangulation

A significant speed-up can be achieved, if the triangulation is stored in a space partitioning data structure. The hashed triangulation approach is a space efficient implementation of a 3D regular grid by using spatial hashing [THM+03]. In this way, the storage requirements can be restricted arbitrarily, e.g., linear in the number of model triangles.

For a given query point, the hashed triangulation approach determines which grid cells may potentially contain the nearest triangle. Within the grid cells in question, the registered triangles are checked. According to the classification, it is based on *searching of surface primitives derived from the original object representation.*

pros The technique is easy to implement, and a well chosen grid cell size gives good query times.

cons The memory footprint is exponential in the subdivision depth which disqualifies it for many applications. Another problem is the algorithm's dependency on the choice of the grid cell size. A reasonable size takes into account the model's bounding volume as well as its face distribution within the domain (see Section 4.3.5).

Hashed Triangulation – First Hit

A further speed-up is possible, if only the distance value (not the corresponding perpendicular point) is needed, and if a small error is acceptable. In this case, only the nearest non-empty cell is checked. If no other cell is checked, the returned value may have an error up to the length of the cell's diagonal.

pros Same as Hashed Triangulation.

cons Same as Hashed Triangulation. The returned distance value is only a rough approximation.

Adaptive Subdivision

The triangulation-based distance calculations described before have large memory requirements in common. If the subdivision control mesh has to remain in memory, for any reason, the triangulation-based methods are not suitable due to their large memory requirements. An approach which does the refinement of the subdivision mesh on-the-fly has always smaller memory requirements. The implementation of the adaptive subdivision algorithm presented by VOLKER SETTGAST et al. [SMFF04] uses a hashed 3D regular grid structure to identify relevant subdivision patches. These patches are subdivided using slates as needed. According to previous classification, it uses *searching of surface primitives of the original object representation*.

pros The memory footprint is only linear in the size of the subdivision mesh due to the 3D hash table. The additional overhead during a patch evaluation is of small, fixed size and can be neglected.

Only a small preprocessing is needed. In contrast to triangulation-based approaches, this allows to modify the maximum subdivision depth and therefore adapt the accuracy of the distance calculation as needed.

cons The algorithm requires a substantial implementation.

Bézier Representation & Numerical Optimization

Some subdivision schemes, e.g. Catmull-Clark subdivision [Sta98], allow direct evaluation at arbitrary parameter values. This property can be used to formulate a distance calculation algorithm. Having identified relevant subdivision patches, the algorithm converts them into Bézier patches. For regular patches this can be done exactly. Irregular patches have to be approximated. Using a parametrization as a Bézier patch, the distance calculation can be formulated as a minimization problem in parameter space [MH03],[MK05],[Sel06]. For the resulting nonlinear minimization problem, Newton-type techniques [FR64], [Kel99] can be used with suitable start values in parameter space.

pros The memory requirements are comparable to the adaptive subdivision algorithm. As the distance calculation is reduced to a standard problem of numerical optimization (see Chapter 2.3), highly-optimized numerical libraries can be used.

cons The Bézier approximation has some additional runtime overhead, but can be cached with the subdivision mesh. The following distance minimization requires considerable tuning of the step sizes. The choice of the start parameter of the Newton-like iteration has more influence on the runtime than the size of the model.

Furthermore, the conversion of Catmull-Clark subdivision surfaces to bicubic Bézier patches is patent-registered ("Approximation of Catmull-Clark subdivision surfaces by Bézier patches", United States Patent No. 6 950 099).

4.3.5 Technical Details

In order to allow a thorough comparison of the chosen algorithms some implementation issues are discussed in detail.

Evaluation Errors

The triangulation-based methods use a fixed, uniform subdivision depth of three subdivisions. Note that the use of limit points improves the approximation error, which can be bounded by a factor times the maximum of the triangle's side lengths, where the factor depends on the model. The limit points lie in the convex hull of the Bézier control mesh instead of the convex hull of the corresponding face's 1-ring in the subdivision mesh. This error has been used to set the termination condition of the adaptive subdivision algorithm. Therefore, the adaptive version has a maximum subdivision depth of three, but it is allowed to terminate earlier, if the resulting maximum error is of same size.

The Bézier surface patches resulting from the conversion have a deviation from the Catmull-Clark surface patches only in irregular patches. But the subsequent parameter search, which works with the Bézier representation, produces an error by itself. With the termination condition in parameter space it is difficult to control the distance error because the threshold in parameter space depends on the curvature near a minimum point's parameter. In our experiments we used only a fixed threshold.

The accuracy of the First-Hit algorithm is determined by the triangulation error plus $\sqrt{3}$ times the grid cell size.

Grid Size Problems

The grid cell size is not only responsible for the algorithm's accuracy. The choice of a reasonable value affects the algorithm's performance significantly. Unfortunately, the value depends on the distribution of the cached geometric primitives (triangles, Bézier patches, etc.) within space. Without additional knowledge only some heuristics are at hand. Let d be the bounding volume's diagonal length, and p be the number of geometric primitives to hash. If all objects are distributed uniformly in their bounding volume, a grid cell size of $d/\sqrt[3]{p}$ is a reasonable choice. If the surface of a geometric object is not distributed uniformly in space (which is the normal case), the grid should be coarsened. In our implementation the grid cell size had been chosen to

$$s = \frac{d}{\sqrt[n]{p}} \qquad \text{with } 3 \leq n \leq 5, \tag{4.119}$$

which has led to feasible runtimes. An illustrative example in Figure 4.32 shows the correlation of cell grid size and evaluation time for a test object.

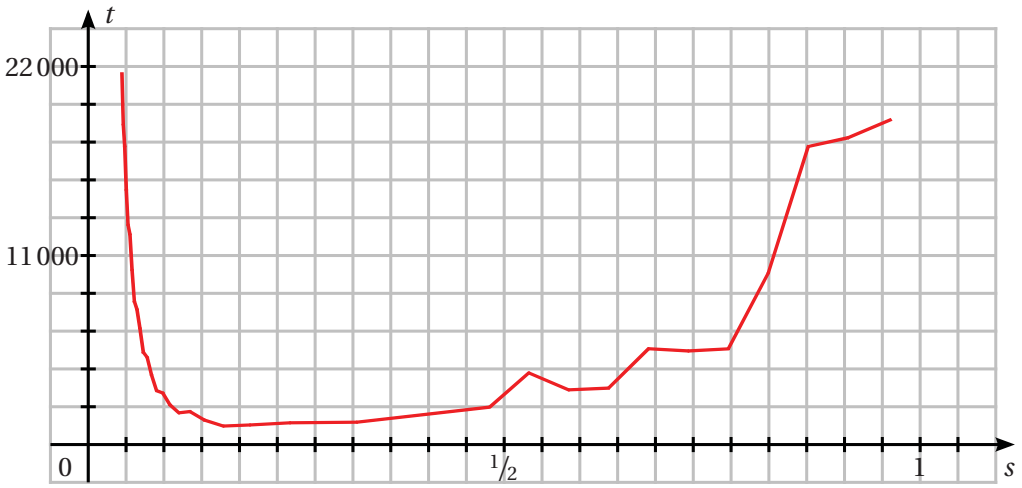


Figure 4.32: This Figure demonstrates the correlation between grid cell size and runtimes of hashing-based algorithms. The used test object "pawn" has been triangulated (8 862 triangles). All triangles reside inside the axis-aligned bounding box whose diagonal has a length of 3.94. According to the heuristics in Equation (4.119) the cell size should be between $3.94/\sqrt[3]{8\,862} \approx 0.19$ and $3.94/\sqrt[5]{8\,862} \approx 0.65$. The needed time in milliseconds to calculate the distance of 10 000 arbitrary points to the triangle mesh using the First-Hit algorithm is plotted against the used grid cell size.

Hashing

All presented, grid-based hashing algorithms use the hashing function presented by MATTHIAS TESCHNER [THM+03]. It takes the indices (x,y,z) of a grid cell and returns the hash value

$$\text{hash}(x,y,z) = (x\, p_1 \,\text{xor}\, y\, p_2 \,\text{xor}\, z\, p_3) \bmod n \tag{4.120}$$

using the prime numbers $p_1 = 73\,856\,093$, $p_2 = 19\,349\,669$, $p_3 = 83\,492\,791$. The function can be evaluated very efficiently and produces a comparatively small number of hash collisions for small hash tables of size n. The traversal within the grid structure is illustrated in Figure 4.33.

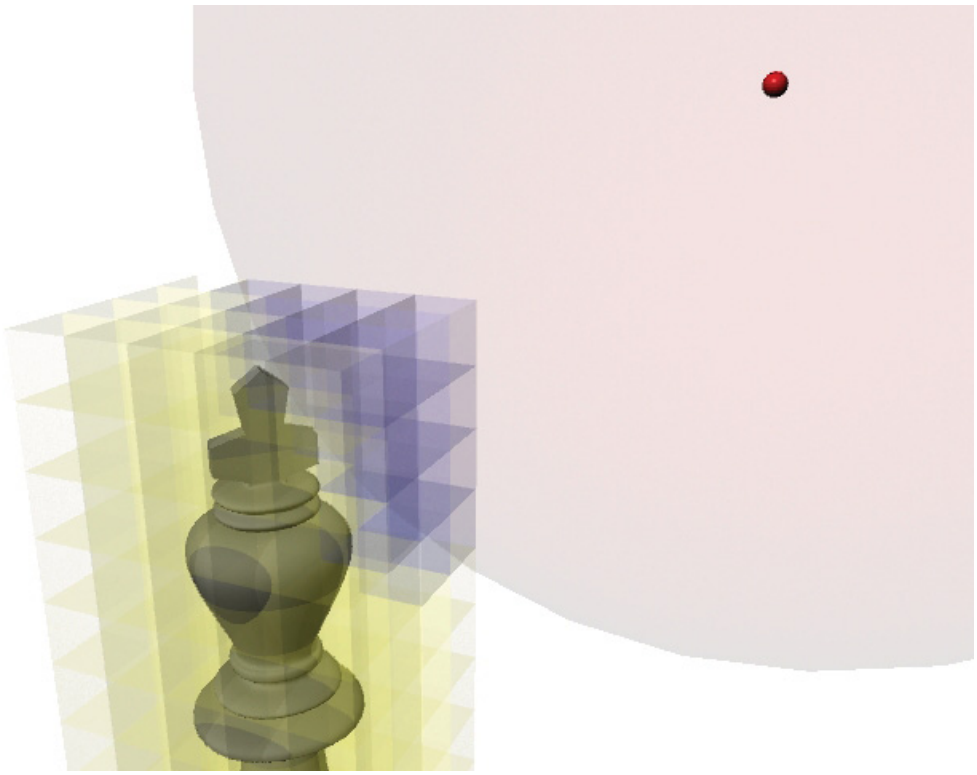


Figure 4.33: The storage of a model in a regular grid allows a fast preselection of relevant patches/triangles, which are near the query point (red). In combination with a good hash function the memory footprint is proportional to the number of model primitives.

Slates for Subdivision Surfaces

The adaptive subdivision algorithm does not modify the base mesh. Instead a separate data structure is used consisting of two so-called slates. A slate is composed of a two-dimensional array of size $(2^d+3)\times(2^d+3)$ and four one-dimensional corner arrays of size $2\cdot(v-4)$, where d is the maximum subdivision depth and v the maximum valence. For performance reasons, the slates are allocated statically as they can be reused for each face to be tessellated.

The subdivision process firstly collects the 1-neighborhood of the considered face f and stores it in the first slate. The vertices of f and the vertices of its edge neighbor faces are stored in the table. If one of the vertices of f has valence greater than four, the remaining vertices are stored in the dedicated corner arrays. Figure 4.34 illustrates this storage scheme for a quad. Other configurations and further details on slates can be found in "Adaptive Tessellation of Subdivision Surfaces" [SMFF04]. The subdivision algorithm processes the vertices row by row and stores the result of one subdivision step in the second slate. For the next step, source and destination slates

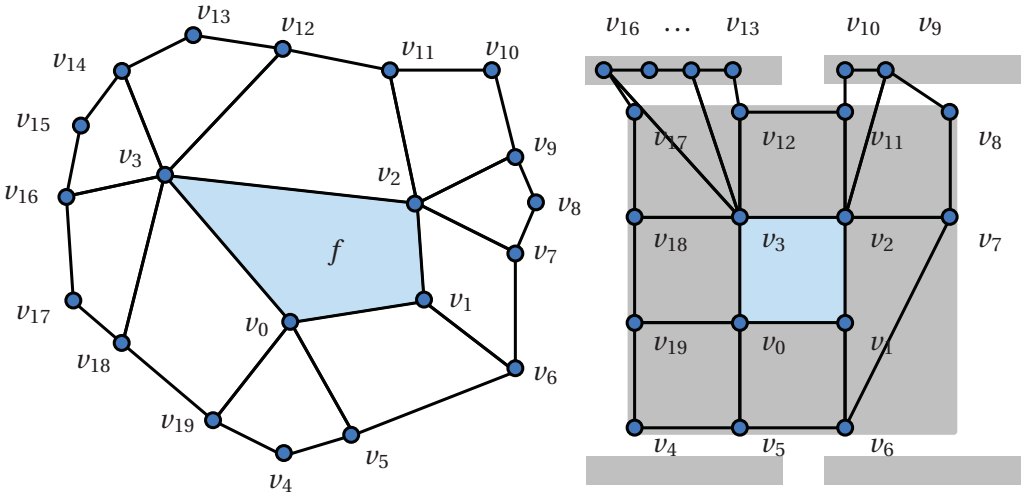


Figure 4.34: The adaptive subdivision algorithm stores the collected 1-neighborhood of a face f from the base mesh (left) in a data structure called slate (right). A slate consists of a two-dimensional array and four one-dimensional arrays.

are swapped. After two subdivision steps, the algorithm starts calculating distances from the corresponding limit points of the 25 (5×5) vertices to the query point. For the following subdivision steps, only a subpart of 9 (3×3) vertices of the table array is used, see Figure 4.35. The subpart is chosen depending on the results of the distance calculations. The process is repeated until the difference of the minimal distance for the current and the last iteration is below a user-defined threshold.

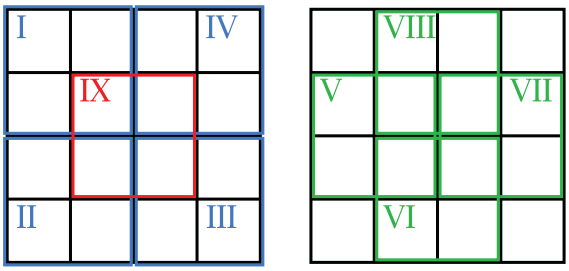


Figure 4.35: After the second subdivision step each face of the control mesh consists of 5×5 vertices. During the distance calculation only relevant subparts out of nine possibilities are processed further on. Five possible sectors are illustrated on the left, four on the right.

4.3.6 Benchmarks

The test scenario is made of six subdivision surface models.

1. **Pawn** This object consists of 70 patches. Its triangulation at subdivision level 3 has 8 862 triangles.

2. **Rook** Within the test scenario this object is the most complex one. It is composed of 1 454 subdivision surface patches, respectively 185 328 triangles.

3. **Knight** The control mesh of this model has 78 faces. Triangulated after three successive subdivisions it consists of 9 356 triangles.

4. **Bishop** The bishop is modeled using 130 patches. In this case the triangulation-based algorithm have to handle 16 542 triangles.

5. **Queen** This model has 387 subdivision surface patches which results in a triangulation with 49 508 elements.

6. **King** The king consists of a subdivision mesh with 175 faces. Its tessellation with 19 560 triangles ranges in the midfield of the test scenario.

During each test an algorithm has to calculate the distance between the test object and 10 000 arbitrary query points. The query points are uniformly distributed within a box whose volume is twice as large as the test object's axis-aligned bounding box volume. Each test model has a closed 2-manifold boundary and the query points may be located inside and outside of it; whereas the returned distance has no sign and does not distinguish between interior and exterior.

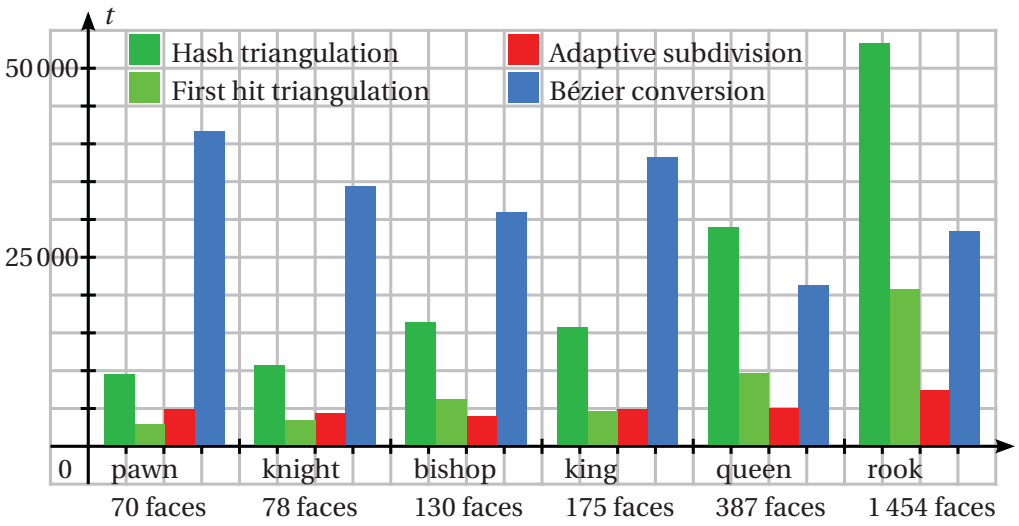


Figure 4.36: The test objects are Catmull-Clark surfaces. The smallest object – the Pawn – has a subdivision control mesh which consists of 70 faces. The most complex model is the Rook with 1 454 patches. Its triangulation at subdivision level 3 has 185 328 triangles. Within the distance calculation test each algorithm has to determine the distance between the model and 10 000 arbitrary query points. The results are measured in milliseconds.
The triangulation-based algorithms as well as the adaptive subdivision one correlates with the model's complexity in contrast to the approach using Bézier conversion and numerical optimization. This algorithm is rather determined by internal parameters (initial parameter values, step size, etc.) than by model complexity.

The runtimes of these tests are shown in Figure 4.36. The results indicate some interesting facts. Both the adaptive subdivision technique and the Bézier conversion approach use the same 3D hashed grid structure to identify relevant patches with a grid cell size of $d/\sqrt[3]{p}$, whereas d denotes the AABB diagonal and p the number of patches in the base mesh. The adaptive subdivision depends on the number of relevant patches which correlates with the model's complexity. But the Bézier conversion is rather determined by internal parameters (start values for numerical iterations, etc.) than by model complexity. This calculation overhead is almost independent from the input data and surmounts the time needed by the adaptive subdivision approach several times.

Another interesting point which can be seen in the diagram is the speed-up factor of the first-hit algorithm. Compared with the variant which checks additional grid cells in order to return the exact distance instead of an approximation the first-hit version is three times faster ($\varnothing \approx 3.09$). Of course, both algorithms use the same grid size. The number of grid cells is proportional to the number of triangles in the tessellation.

While it is normally not recommended to triangulate a subdivision surface ahead of time, the first hit version has similar timings as the adaptive evaluation technique, at least for small- and medium-sized models.

According to the benchmarks presented above, the distance between an arbitrary point and a subdivision surface should be determined using an efficient space partitioning technique such as hashed, regular 3D grid and an on-the-fly subdivision surface evaluation algorithm. The result is a distance calculation which

- needs considerably less memory than triangulation based approaches, and
- is the fastest method in most cases.

The only negative point of the adaptive subdivision method is its complex implementation. The conversion method may use numerical libraries and the triangulation methods can use wide-spread, standard techniques, whereas an efficient, on-the-fly evaluation of subdivision surfaces must be implemented efficiently for the mesh structure used.

Therefore, the triangulation-based approach with the first-hit termination might be considered for small model sizes, if the perpendicular point is not needed and if an approximation of the distance is enough. In all other cases the adaptive subdivision technique is the best choice.

4.3.7 Modeling with Subdivision Surfaces

Current research activities on subdivision surfaces comprehend, amongst others, three very important aspects. First of all subdivision surfaces have a small degree of continuity compared to Bézier-, B-splines or NURBS surfaces. Catmull-Clark subdivision surfaces are C^2-continuous respectively only C^1-continuous in irregular vertices, whereas in high-quality design C^2 continuity is often a minimum requirement [Bon09]. These requirements can be met using one of several modified subdivision surface schemes[PU98], [Pet00], [ADS06], [MP09].

Another important question in the context of subdivision surfaces is the integration into NURBS-dominated processing pipelines. As NURBS are the “standard” tool to model free-form surfaces, many subsequent steps and algorithms (simulation tools, etc.) are NURBS-based. In order to use the advantages of subdivision surfaces, open questions on conversion, integration and migration have to be answered [CADS09].

Last but not least the simple question “where to place the points” is still not answered satisfactorily, although each modeler – human or software – has to answer it. Based on case studies, heuristics for modeling with Catmull-Clark surfaces can be given [USF10a]. The way how to model with subdivision surfaces is described in various tutorials and courses [ZSS96], [DKT98], [ZSD+00]. These tutorials pursue a plan which can be summarized by the sequence

idea → subdivision surface model → real 3d object,

whereas the last production step is omitted, if only the virtual object is of interest. Within the last few years subdivision surfaces are also used in reverse engineering at a progressive rate. Reverse engineering is the inverse situation of the sequence above.

A real 3d object is the starting point and its corresponding subdivision surface model is the desired result. During the re-modeling phase the modeler – either human or algorithm – has to tackle two problems: geometry and topology. These are the two ingredients of a subdivision surface. Choosing the right topology to represent a given object by a subdivision surface reduces the number of needed control vertices significantly. The choice of a good topology is a distinction between an experienced, skilled modeler and a beginner.

In "Modeling with Subdivision Surfaces" [USF10a], we, TORSTEN ULLRICH, ANDREAS SCHIEFER, and DIETER W. FELLNER concentrate on the more general situation from the reverse engineering point of view. From this perspective a subdivision surface reconstruction has to tackle two problems, a topological problem (the layout of the control mesh) and a geometrical problem (the position of the control vertices in 3D). Unfortunately, in most articles about subdivision surface fitting [CWQ+04], [CWQ+07] the topological problem plays a minor role [MK05].

In this section the topological problem is addressed; i.e. for a chosen topology the geometry of the control mesh is optimized. The optimization results establish a relationship between topological layout and approximation quality, which allows to identify the best topology. In order to concentrate on the topological aspects the geometry is optimized automatically. The optimization uses a distance based error function [USK+07], [PW09] which is minimized by a standard approach of numerical minimization [Nas90], [GOT05]. The minimization process is straight forward and operates on the vector of all vertex coordinates of all vertices to position. Algorithmic details[7] on the geometrical optimization are described by RAINER STORN and KENNETH PRICE [SP97].

The following case studies cover the four most important situations when modeling with Catmull-Clark subdivision surfaces:

Modeling Edges The first case examines smooth edges. While sharp edges can be modeled easily with special feature rules for subdivision surfaces, smooth edges offer at least two possibilities to be designed: with a row of control vertices along the edge or with two rows of control vertices alongside the edge.

Modeling Non-Quadrilateral Configurations Almost each type of subdivision surface has a topology for which it suits best. Loop subdivision prefers triangles, Catmull-Clark subdivision generates quads, etc. Unfortunately, not every object has a favorable geometric primitive to be modeled with. Several strategies for such a configuration are possible.

Modeling Curvature The third case inspects a surface with different curvatures: hyperbolic, parabolic and elliptic. It addresses the issue of quad orientation with respect to the surface's principal curvature directions.

Modeling Inflection Points The last case analyzes a surface which is defined by a cut curve. This is a standard situation in CAD modeling.

[7] Differential Evolution for Continuous Function Optimization, http://www.icsi.berkeley.edu/ storn/code.html

For each case different variants of control meshes that are adjusted by a number of parameters are examined. These parameters are set by a numerical optimization routine which minimizes the distance [USK+07] between the nominal surface and the actual subdivision surface.

Smooth Edges

The first nominal surface is defined by the implicit equation

$$S_1 : \; x^6 + y^6 + z^6 = 1. \tag{4.121}$$

The resulting surface looks like a cube with round edges and corners (see Figure 4.37). In order to analyze how to model beveled edges two variants are inspected.

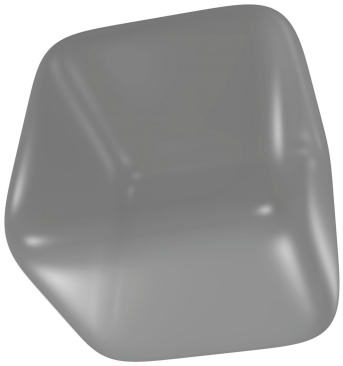


Figure 4.37: The implicit surface $x^6 + y^6 + z^6 = 1$ has a degree which cannot be reached by a cubic patch. Therefore, Catmull-Clark subdivision can only approximate it. The best way to approximate it is analyzed by testing different topologies.

Variant 1 The first variant of possible control meshes has a cube-like topological layout. It consists of 3×3 quads on each side. Due to symmetries the geometry is defined by only six parameters of three initial control points. Three parameters are used for the three coordinates (x_1, y_1, z_1) of the first control point. The fourth and the fifth parameter define the second control point. In order to avoid self-intersections as well as optimizations with constraints, these parameters are defined as positive offsets to the first control point. Consequently, they have a fixed domain and the second control point is calculated via $(x_1 + x_2, y_2, x_1 + x_2)$. As the second control point defines an edge of the control mesh cube, and as the faces of the cube are connected at/over this edge, the x and z components have to be equal. Finally the last parameter x_3 again defines an offset from x_1. Due to symmetries this is the only value needed for the third control point: $(x_1 + x_3, x_1 + x_3, x_1 + x_3)$. As the third control point is at the corner of the control mesh cube, all coordinate components must be equal. Figure 4.38 shows the initial three control points that are generated by the six parameters. The optimization

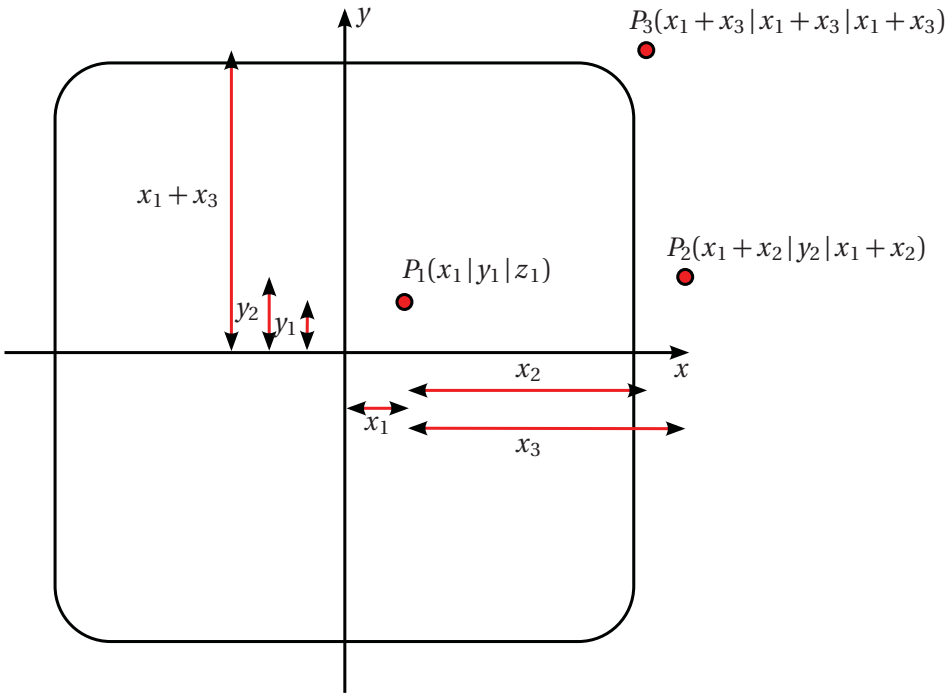


Figure 4.38: Due to symmetries six parameters are enough to define three different control points. The complete subdivision control mesh is composed of rotations and mirrorings of these points. Each of the six cube sides consists of 3×3 faces.

routine minimizes the distance between the generated surface and the reference surface (Equation (4.121)) based on uniformly distributed samplings. The error function $f_{1,1}$ is a sum of point-to-subdivision surface distances. The optimum is

$$\begin{aligned} f_{1,1}(\quad & 0.33633, \quad 0.33526, \quad 0.99830, \\ & 0.66616, \quad 0.32615, \quad 0.66319 \quad) = 7.45141. \end{aligned} \tag{4.122}$$

The three control points, which generate the complete subdivision control mesh by rotations and mirrorings, have the coordinates

$$P_1(0.33633\,|\,0.33526\,|\,0.99830), \tag{4.123}$$
$$P_2(1.00249\,|\,0.32615\,|\,1.00249), \tag{4.124}$$
$$P_3(0.99952\,|\,0.99952\,|\,0.99952). \tag{4.125}$$

Variant 2 Also the second variant is a cube-like control mesh derived from three initial control points but with five parameters. This time every side of the cube consists of four quads, the six sides of the cube are connected through beveled faces at the edges and with triangles at the corners. Figure 4.39 illustrates this control mesh.

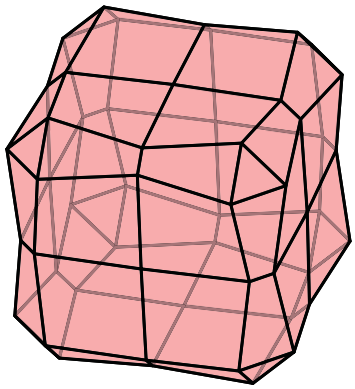


Figure 4.39: This variant uses a control mesh with explicitly beveled edges to approximate an implicit surface of degree 6.

The first parameter, z_1, is the z component of the first control point, which is located at $(0,0,z_1)$. Due to symmetries the first control point is centered on each side of the control mesh cube. The second and third parameters, x_2 and z_2, define the second control point at $(x_2,0,x_2+z_2)$. This time the sides of the cube are connected via beveled faces, therefore the offset z_2 is added to the z component of the second control point. Using the last two parameters, x_3 and z_3, the third control point is (x_3,x_3,x_3+z_3). The error of the best control mesh of this variant – according to the optimization routine – is 7.79202:

$$\begin{aligned} f_{1,2}(\quad & 0.99686, \quad 0.83273, \quad 0.14743, \\ & 0.74912, \quad 0.33157 \qquad\qquad) = 7.79202. \end{aligned} \tag{4.126}$$

Comparison Figure 4.40 illustrates both variants. Each variant is rendered with a color scheme indicating its distance to the nominal surface which is included in each rendering. It is visualized with a transparent, grayish style. The illustration shows that the first variant performs better than the second one. The second variant uses beveled edges to model the reference surface. This topology is not suitable to approximate the given surface. Even its geometrically optimized version shows "over-modeling" effects – a high frequency fluctuation which is spread out from the over-modeled parts. In this case the beveled edges disturb the low frequency parts of the surface. An in-depth analysis of the distances confirms the visualization. The average and the maximum of all distances between the nominal surface and points on the actual surface are:

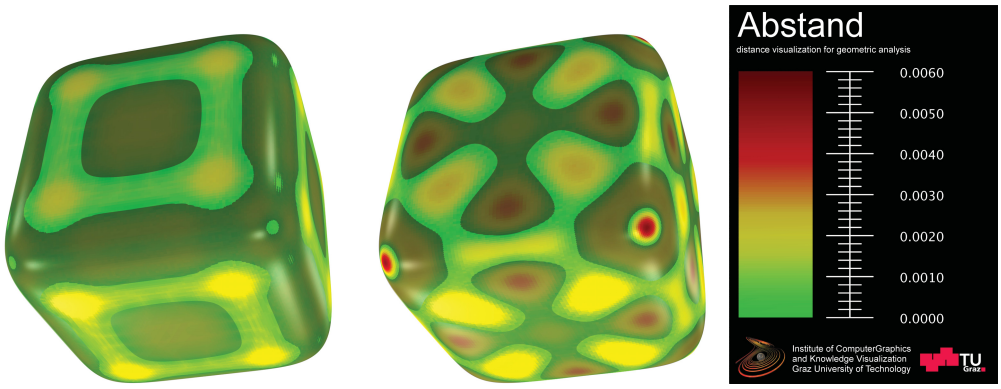


Figure 4.40: In comparison to each other the first variant performs better than the second one. Explicit modeling of beveled edges increases the risk of "over-modeling" – a high frequency fluctuation which is spread out from the beveled edge and which disturbs the low frequency parts of the surface.

	avg. distance	max. distance
Variant #1	0.00079	0.00204
Variant #2	0.00130	0.00519

Modeling Non-Quadrilateral Configurations

The second surface analysis investigates non-quadrilateral configurations. The reference surface is a so-called monkey saddle. It is a heightfield defined by

$$S_2(x,y) = \frac{x^3 - 3xy^2}{2}. \tag{4.127}$$

This surface is axially symmetric at a degree of 120°; i.e. after a $\frac{2}{3}\pi$ rotation the surface is congruent to itself. The point at the origin is a parabolic, umbilic point. Within this analysis the parameters x and y may range from -1.0 to 1.0.

As the following variants have different numbers of control vertices, the area of every control mesh is chosen to be proportional to the number of its vertices. So control meshes with more vertices have to approximate a larger area of the reference surface.

Variant 1 The first variant has 19 vertices and seven parameters, which define their heights (z component). The symmetric configuration is illustrated in Figure 4.41 which shows the topology of the control mesh and the parameters (height) of each vertex. All vertices with $x = 0$ have a fixed height of 0.0. The best optimized version of this variant has an error of 0.44914. Its parameters are

$$\begin{aligned} f_{2,1}(\quad & -0.27195, \quad -0.00007, \quad 0.00001, \\ & -0.27193, \quad -0.00007, \quad 0.27181, \\ & -0.0001 \qquad\qquad\qquad\quad) = 0.44914. \end{aligned} \tag{4.128}$$

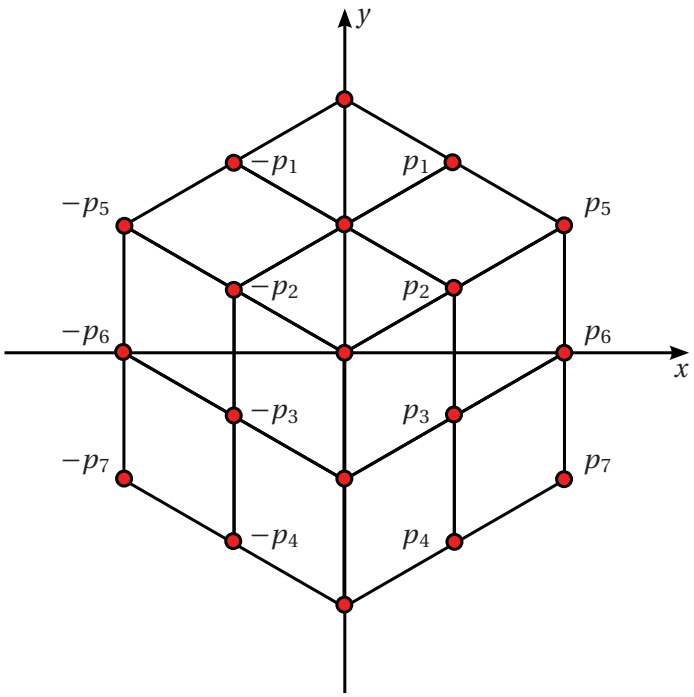


Figure 4.41: The topology of this control mesh is arranged in a 120° symmetric layout to adapt the subdivision surface to the reference.

Variant 2 For the second variant a predefined mesh with 27 vertices is used as control mesh. The heights of its vertices are controlled by twelve parameters. Figure 4.42 shows the parameters and the topology which is (in the inner part) dual to the first variant. Again, all vertices with $x = 0$ have a fixed height of 0.0.

The smallest possible error of this topological configuration is

$$\begin{aligned} f_{2,2}(\quad & 0.00148, \quad -0.07632, \quad -0.25658 \\ & -0.37829, \quad -0.07681, \quad 0.07507, \\ & 0.00052, \quad -0.37787, \quad -0.25613, \\ & 0.25710, \quad 0.37745, \quad -0{,}00072 \quad) = 0.69362. \end{aligned} \tag{4.129}$$

Variant 3 The third variant for this surface does not adapt to the reference's axial symmetry of $\frac{2}{3}\pi$ in any way. It uses a regular quadratic grid centered on the reference surface. There are five vertices along each side of the grid, so 25 vertices in total. Fixing the vertices at $x = 0$ six parameters are enough to describe all vertices. Figure 4.43 illustrates this topology. The optimization returns the optimum of this topology with an error of $f_{2,3} = 0.80376$. The parameters for this control mesh are $(-0.32781, -0.05752, 0.01131, -0.45664, 0.11135, 0.24120)$.

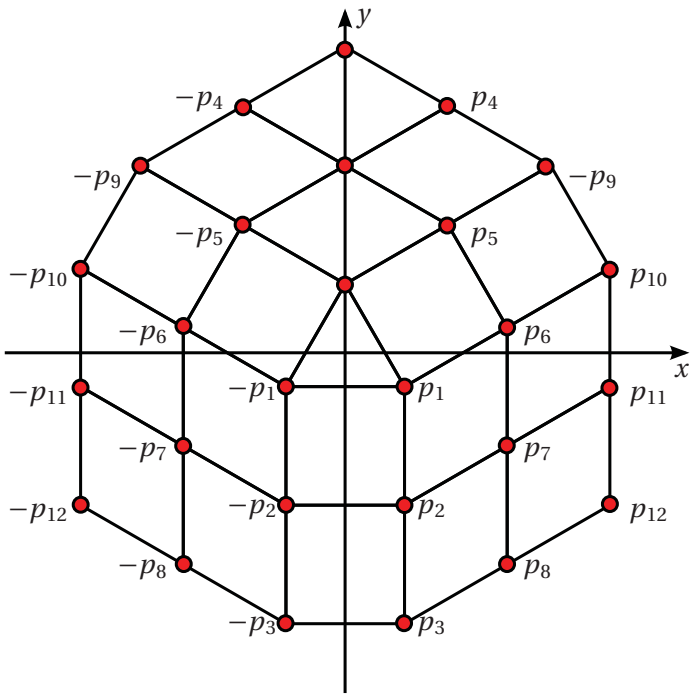


Figure 4.42: The topology of the second variant has a configuration which is dual to the first one (except for the border).

Comparison Taking the number of control vertices into account, each surface has been normalized; i.e. the area of every control mesh is proportional to the number of its vertices. Therefore, all subsequent error values are normalized and comparable to each other. In this situation the case study does not reveal a best topology but a worst one. The regular gird does not approximate the given surface well. The two variants whose topologies reflect the nominal surface's symmetry perform much better (see Figure 4.44):

	avg. distance	max. distance
Variant 1	0.00241	0.00624
Variant 2	0.00153	0.01037
Variant 3	0.00330	0.01555

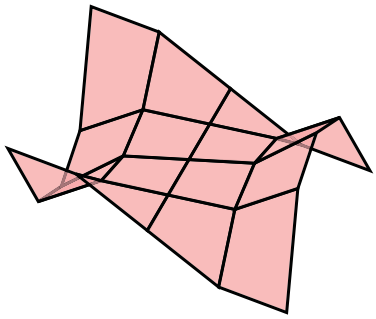


Figure 4.43: A rectangular topology leads to a very high error. Topologies that reflect the nominal surface's symmetry perform much better.

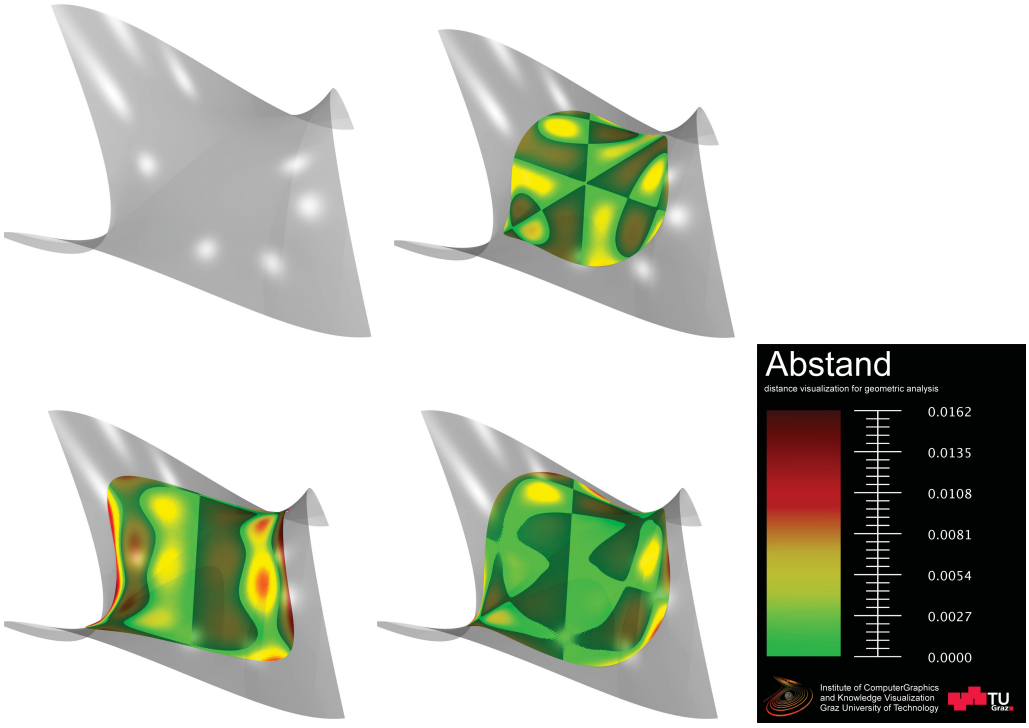


Figure 4.44: The nominal surface (upper left) is invariant to 120° rotations. The topology of variant #1 (upper right) and #2 (lower right) reflect this property whereas variant #3 (lower left) uses a simple rectangular grid. In this case topologies, which reflect the main symmetries of the reference surface, perform better.

Modeling Curvature

The third modeling study inspects a torus due to its characteristic curvatures: hyperbolic, parabolic, and elliptic [Iro05]. The reference is described by the formula:

$$S_3(u,v) = \begin{pmatrix} \big(a + b\cos(2\pi v)\big) \cdot \cos(2\pi u) \\ \big(a + b\cos(2\pi v)\big) \cdot \sin(2\pi u) \\ b\sin(2\pi v) \end{pmatrix} \tag{4.130}$$

with major radius $a = 5.0$ and minor radius $b = 3.0$. The parameters u and v are within the range 0.0 to 1.0.

Variant 1 For the first variant the parameter domain is sampled at a fixed, regular grid to get the control point positions. Two parameters, that can be set by the optimization, define the major and the minor radius of the control mesh torus. This is probably the most commonly used parametrization of a torus in modeling software.

The error of this variant after the optimization is 824.82012. The two parameters defining the major and minor radius have their optimum at 5.42375 respectively 3.38278. This variant is compared to a slanted torus control mesh.

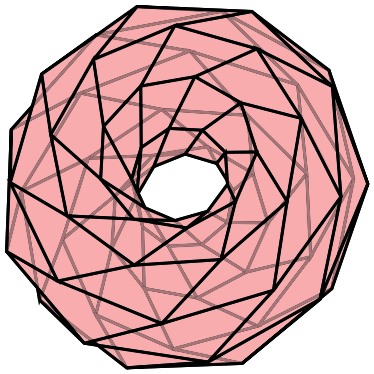


Figure 4.45: A slanted torus control mesh is a "non-standard" way to model a torus. This variant is compared to the most commonly used parametrization with a rectangular grid layout of the parameter domain.

Variant 2 A slanted torus uses a slightly different parametrization than the first one. Again, the values u and v are a regular grid in the parameter domain, but this time an offset depending on u is added to v each time. For this variant, the offset is defined as $\frac{2}{3}u$. This introduces a slant in the control mesh.

The optimization routine calculates a minimum error of 1474.60738 for this variant. The parameters defining the resulting control mesh are $a = 5.62499$ and $b = 3.43444$. Figure 4.45 shows the slanted control mesh created with these parameters.

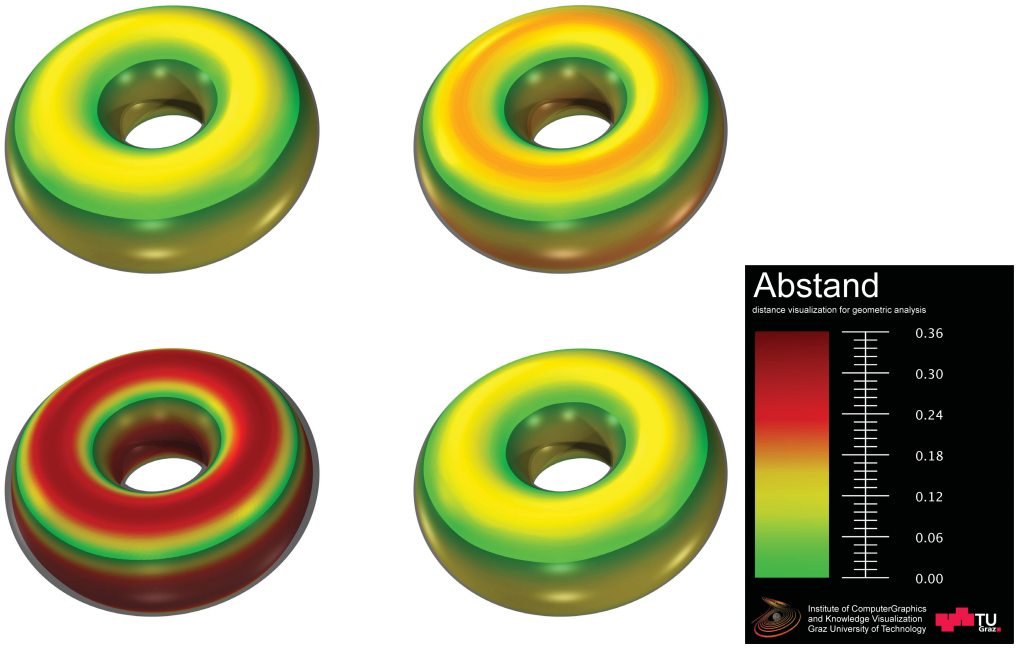


Figure 4.46: A torus can be modeled with a slant offset and without one (upper left). With increasing slant offset (upper right, lower left) the approximation error increases. If the optimization routine is allowed to set the slant offset by itself, it is set to zero (lower right).

Variant 3 This variant of the torus control mesh is basically the same as variant 2, except that the offset defining the slant is defined as $\frac{4}{3}u$. Even the optimized geometry ($a = 6.24999, b = 3.56744$) leads to a large error $f_{3,3} = 4290.40374$.

Variant 4 As it is difficult to determine appropriate slant offsets, the fourth variant also optimizes this parameter; i.e. in addition to the two parameters defining the radii of the torus control mesh, it has a third parameter defining the slant offset. The optimization process returns

$$f_{3,4}(5.42399, 3.38277, 0.00000) = 826.65427. \tag{4.131}$$

Comparison The easiest way to model a torus seems to be the best way. All variations in topology increase the approximation error (see Figure 4.46). The in-depth analysis of distances confirms this heuristic:

	avg. distance	max. distance
Variant 1	0.07806	0.12925
Variant 2	0.11145	0.19450
Variant 3	0.19887	0.35382
Variant 4	0.07806	0.12904

Modeling Inflection Points

The last case analyzes a surface which is defined by a cut curve. Blended with a straight line the result has the formula:

$$S_4(x,y) = \frac{y}{10} \cdot \sin\left(\frac{2\pi}{3}\arctan x\right) \tag{4.132}$$

The parameter x describes the cut curve in the range from -5.0 to 5.0 whereas the parameter y blends between the straight line and the curve from 0.0 to 10.0. The surface is plotted in Figure 4.47.

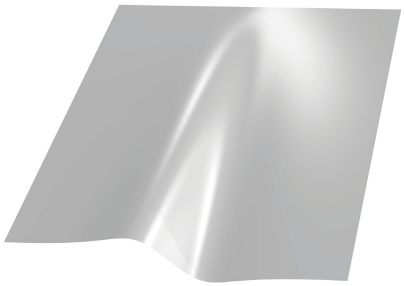


Figure 4.47: This surface is defined by a cut curve which is blended with a straight line. This is a standard situation in CAD modeling.

Variant 1 The control mesh of the first variant gets 20 parameters – 10 pairs of x position and height value z. The range of the x values (-5.0 to 5.0) has been partitioned into 10 equally-sized intervals. In each interval one and only one parameter pair is allowed. For each pair $p_i = (x_i, z_i)$ two control points are generated: one at $(x_i, 10, z_i)$ to approximate the cut curve and one at $(x_1, 0, 0)$ on the straight line. The error of the optimized variant is 6.73931:

$$\begin{aligned} f_{4,1}(\quad & -4.35546, \quad -0.30532, \\ & -3.05017, \quad -0.48656, \\ & -2.93309, \quad -0.50592, \\ & -1.58539, \quad -0.82026, \\ & -0.37453, \quad -1.29349, \\ & 0.38362, \quad 1.32794, \\ & 1.71154, \quad 0.78575, \\ & 2.30383, \quad 0.65748, \\ & 3.00657, \quad 0.48136, \\ & 4.02755, \quad 0.35095 \quad) = 6.73931. \end{aligned} \tag{4.133}$$

Variant 2 This variant uses a regular 10×3 grid covering the whole reference surface. All heights are specified as a parameter and the (x,y)-positions are fixed. Consequently, this variant operates on 30 parameters. The optimization calculates an error of 14.70749 for the best geometry. The best parameters for the control mesh are:

$$\begin{aligned} f_{4,2}(\quad & -0.00257, \quad -0.13120, \quad -0.27108, \\ & 0.00131, \quad -0.17258, \quad -0.33910, \\ & -0{,}00099, \quad -0.28033, \quad -0.55752, \\ & 0{,}00048, \quad -0.36468, \quad -0.74336, \\ & -0{,}00057, \quad -0.72072, \quad -1.40922, \\ & -0{,}00022, \quad 0.72074, \quad 1.41115, \\ & 0.00191, \quad 0.36363, \quad 0.74169, \\ & -0.00244, \quad 0.28386, \quad 0.55652 \\ & 0.00267, \quad 0.16798, \quad 0.34212, \\ & -0.00105, \quad 0.13555, \quad 0.26656 \quad) = 14.70749. \end{aligned} \tag{4.134}$$

Variant 3 The third variant has fixed x positions at prominent values of the cut curve: -5.00000 (end of range), -3.31764, -1.63528 (inflection point), -0.93159 (minimum), 0.00000 (inflection point, root), 0.93159 (maximum), 1.63528 (inflection point), 3.31764, 5.00000 (end of range). For each x position there is again one control point at $y = 0$ with height 0 and another one at $y = 10$ with the height value which has to be optimized. So this variant has nine height parameters, one for each x position. The result is

$$\begin{aligned} f_{4,3}(\quad & -0.22288, \quad -0.47240, \quad -0.67730, \\ & -1.37479, \quad -0.00353, \quad 1.37625, \\ & 0.67573, \quad 0.47305, \quad 0.22281 \quad) = 11.91346. \end{aligned} \tag{4.135}$$

Variant 4 The fourth variant is very similar to the third variant. It uses the same configuration but in contrast to fixed x positions, the optimization routine is allowed to modify these positions within an offset of $\pm\frac{1}{3}$. Therefore, this variant takes 18 parameters, nine for the height values at each x position and nine offsets for the initial x positions.

For this variant, the minimum error after the optimization is 6.17020. The parameters for this control mesh are

$$\begin{aligned} f_{4,4}(\quad & -0.25058, \quad -0.46790, \quad -0.74541, \\ & -1.18889, \quad -0.38835, \quad 1.29557, \\ & 0.76445, \quad 0.36971, \quad 0.26533, \\ & \\ & 0.04882, \quad 0.33053, \quad -0.25836, \\ & 0.28352, \quad -0.17465, \quad -0.33332, \\ & -0.01570, \quad 0.32223, \quad -0.03218 \quad) = 6.17020. \end{aligned} \tag{4.136}$$

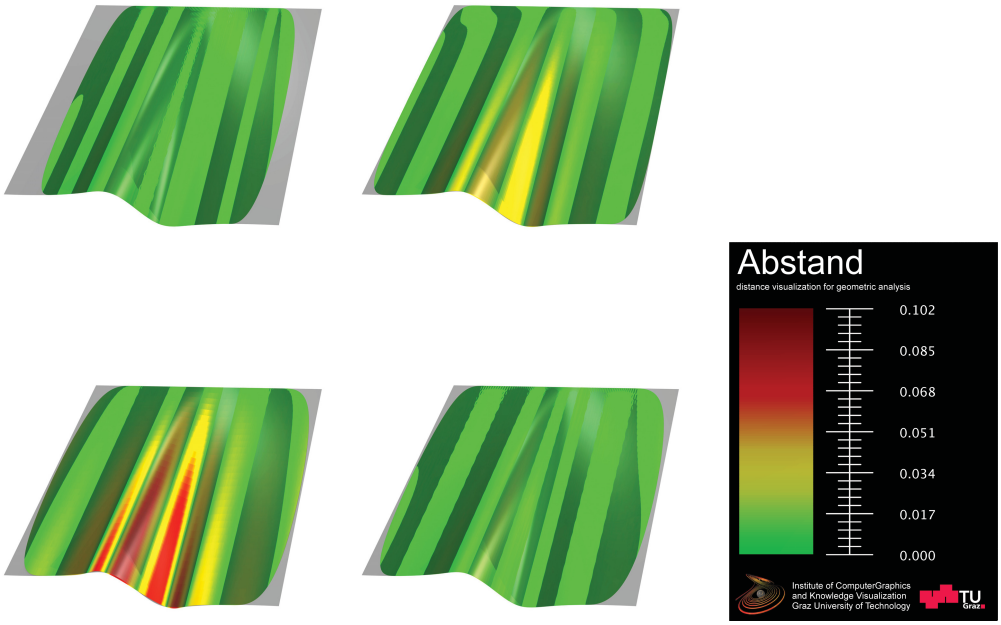


Figure 4.48: In this comparison approximations with fixed x positions are more erroneous than approximations with x positions as free parameters. The interval spacing (upper left) of the first variant and the prominent-values-of-the-cut-curve variant with additional offsets (lower right) are the best results, whereas the regular grid (upper right) and the fixed prominent-values-of-the-cut-curve (lower left) are the worst results.

Comparison The in-depth distance analysis reveals the following values:

	avg. distance	max. distance
Variant 1	0.00264	0.01044
Variant 2	0.00718	0.04152
Variant 3	0.01826	0.09219
Variant 4	0.00323	0.01460

If the area of the approximated surface is taken into account, the fourth variant is even slightly better then the first one. The approximation with the highest error level is the variant which has fixed x positions at prominent values of the cut curve (see Figure 4.48 (lower left)). Consequently, the best result and the worst result can be created with the same topology and minor differences in geometry. The additional offsets which distinguish a good from a bad result are plotted in Figure 4.49.

The figure shows some very important properties when modeling with subdivision surfaces.

- At the extreme values (minimum and maximum at ±0.93159) the control points have been moved by the optimization routine towards the direction of higher absolute gradients.
- Having moved the control points, the control polygon has fewer intersections with the nominal curve than the fixed-x-positions version.
- All versions with a small error (also the first variant with interval spacing) intersect the reference curve very close to (at $x = -1.63528$ and at $x = 0.0$) or nearby (at $x = 1.63528$) inflection points.

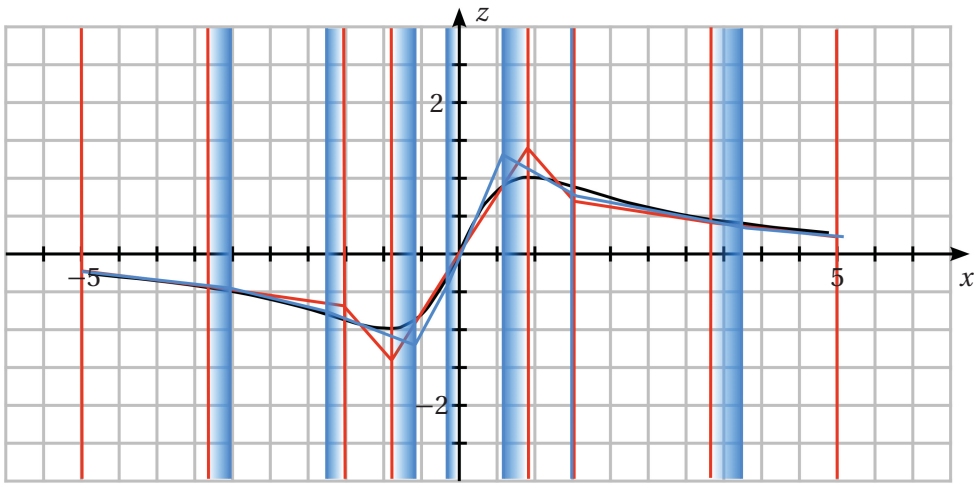


Figure 4.49: The cut curve which defines the fourth nominal surface is plotted in this diagram. Furthermore it shows the positions of the control vertices of the worst approximation (red) and the best approximation (blue). Their differences are visualized by gradients.

Modeling Heuristics

Based on the four case studies several conclusions and heuristics can be derived. The first case examines smooth edges and demonstrates the “over-modeling” effect. Modeling a local surface feature always takes the risk to disturb large parts of a model by spreading high-frequency fluctuations from the “over-modeled” parts. This effect can be avoided by a better topology, which does not model rounded edges explicitly, or by barrier lines – two or three consecutive lines of control vertices of low frequency, which suppress fluctuations due to the locally limited influence of a vertex to a subdivision surface.

While simple geometric properties – such as beveled edges – should not be considered in the topological layout of a control mesh, high-level geometric aspects play an important role. The second case study shows that global symmetries should be reflected in the control mesh. All control meshes, which did not reflect the global symmetry of the surface to approximate, caused higher errors than those with symmetrical layout. The third study approves this fact.

The last case analyzes a surface on the geometrical – not topological – level to explore the best positions for control vertices. Besides the conclusions already presented in the previous section the last study demonstrates the difficulties in predicting a good subdivision approximation without iterative optimization. All solutions with partly-fixed control vertices have a high error value.

The last two cases lead to the assumption to investigate curvature-driven modeling. In the third case study (torus), the best solution has a quad layout parallel to principal curvature lines. In the fourth case (cut curve) all good solutions have control polygons, which intersect the cut curve in its inflection points.

4.3.8 Curvature-Driven Modeling

As modeling is a time-consuming task, techniques to increase efficiency are always an important research topic. Many designs are not digitally-born, but created out of clay. Although their digital counterpart can be created automatically, interactive processes are still in use – especially in high-quality surface design. Consequently, a common task in surface engineering is surface reconstruction: from a triangle mesh of a laser scan towards a CAD representation; i.e. a Bézier surface, NURBS surface, or a subdivision surface. Besides the omnipresent usability aspects in computer science, surface reconstruction has to solve the following problems.

Curvature calculation As differential geometry operates on continuous surfaces and manifolds [DC76], the calculation needs to be adopted to discrete triangulations [GI04]. Having calculated a surface's principal curvatures κ_1, κ_2 including principal curvature directions, derived values such as Gaussian and mean curvature can be calculated.

The correspondence between surface characteristics (of a laser scan) and surface layout (of a CAD surface) is visualized in Figure 4.50.

Curvature flow In the next step, the discrete curvatures calculated at vertex positions are regarded as continuous curvature flow, into which control vertices and meshes are placed. Control elements, which are parallel or orthogonal to principal curvature directions, meet a differential equation; i.e. they are the solution of a so-called initial value problem. This is a continuous problem. Therefore, the curvatures at discrete vertex positions have to be interpolated in between.

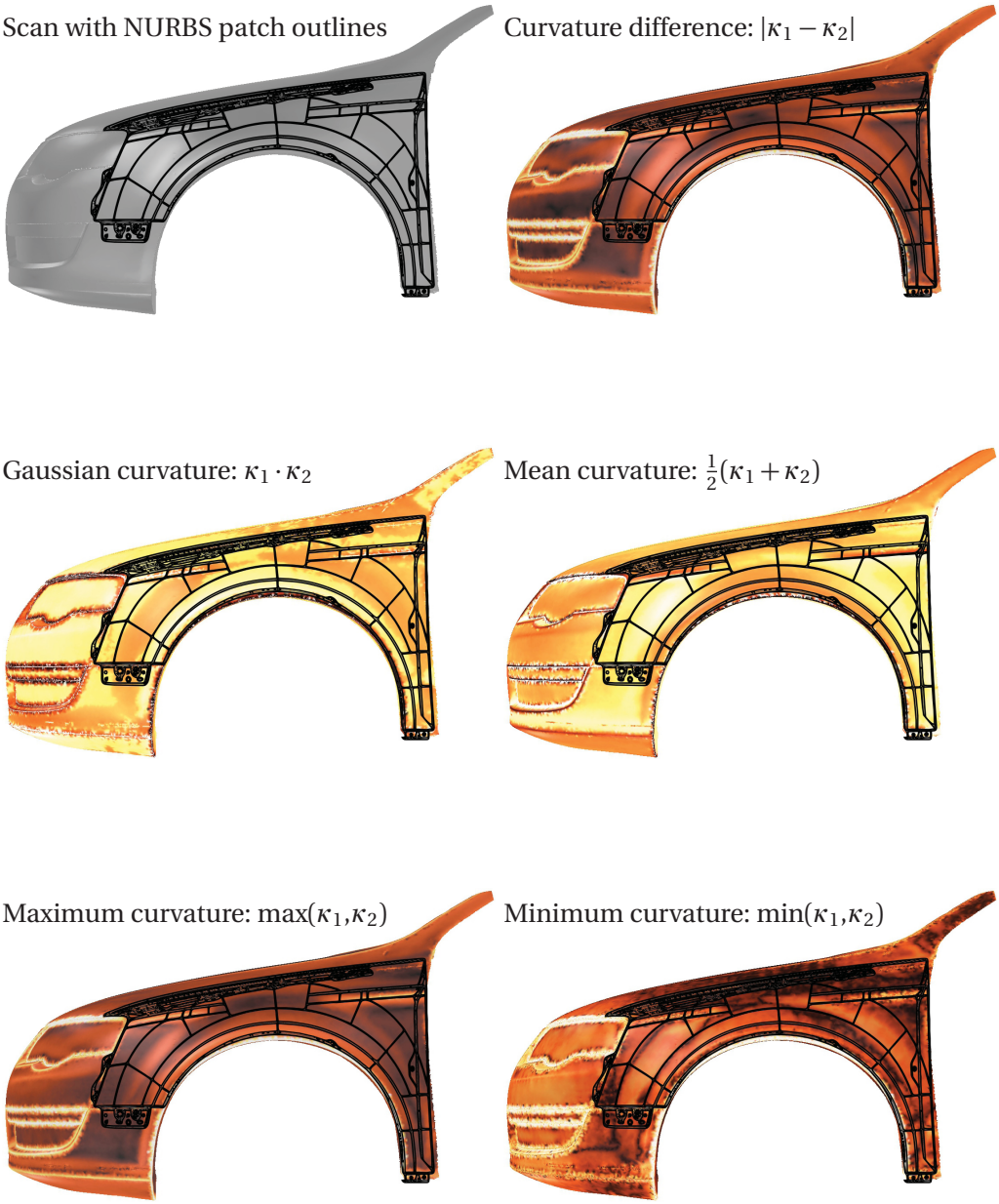


Figure 4.50: In high-quality surface engineering a clay model is scanned and reengineered resp. reconstructed with a CAD surface representation (e.g. NURBS, subdivision surface, etc.). In this example, the scan of a VW Passat front fender and its NURBS representation (upper left) is shown. As surface curvatures (κ_1, κ_2 and derived values) are an important design feature, it is natural to use them for reconstruction.
A curvature-driven patch layout can be generated using curvature tracking – a technique similar to an initial value problem of differential equations.

From the numerical point of view, curvature calculation and curvature flow tracking is very unstable. Especially, in regions with similar principal curvature values ($\kappa_1 \approx \kappa_2$), the calculation of principal curvature directions is not stable. A solution to this problem is a propagation system, in which stable regions propagate curvature directions into spherical and planar regions.

The instability issue of initial value problems is illustrated in Figure 4.51. As differential equations have been a well-established area of research, many numerically stable solvers can be found in literature [AS72], [oST10]. Single step methods – such as Euler's method – only use the differential at the current position $(\mathbf{x}_n, \mathbf{y}_n)$ to calculate the next step from $\mathbf{x}_n$ to $\mathbf{x}_{n+1} = \mathbf{x}_n + h$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h f(\mathbf{x}_n, \mathbf{y}_n). \tag{4.137}$$

The resulting path

$$(\mathbf{x}_0, \mathbf{y}_0), (\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots \tag{4.138}$$

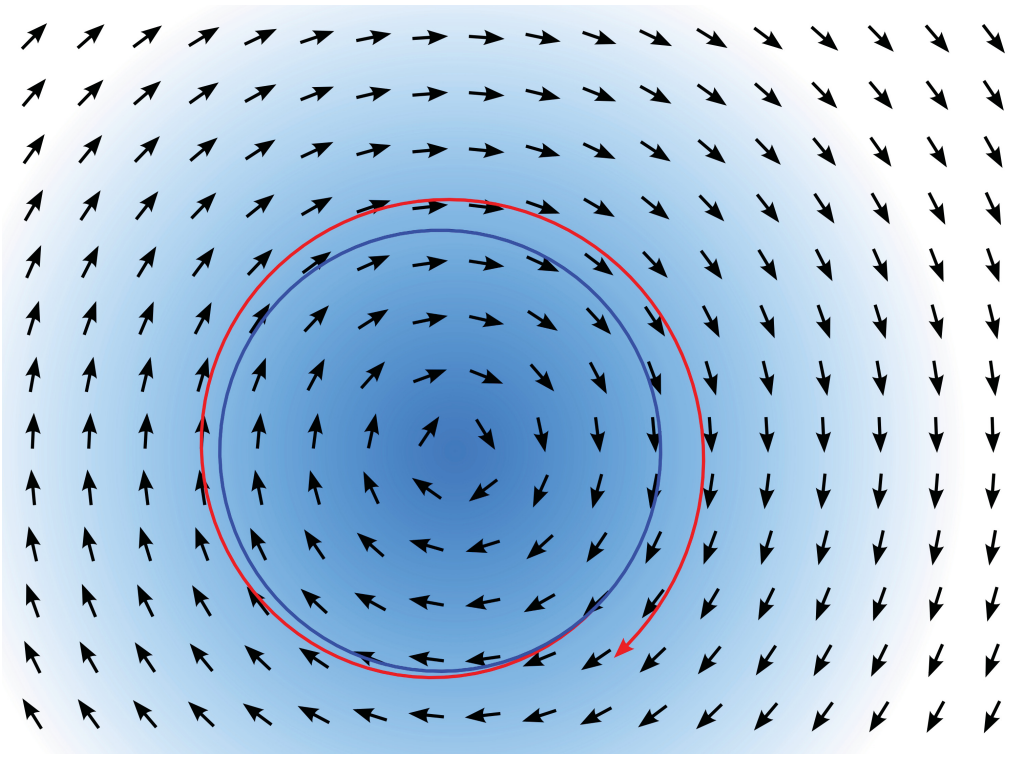


Figure 4.51: Curvature flow tracking can be interpreted as solving an initial value problem: $\mathbf{y}'(t) = f(t, \mathbf{y}(t))$ with function f and condition $\mathbf{y}(t_0) = \mathbf{y}_0$. Many differential equations cannot be solved analytically, in which case an approximation to the solution must be sufficient. If the approximation method is not very accurate and very stable, the approximation will suffer from error accumulation (red curve). In the worst case the calculated solution can differ arbitrarily from the exact solution (blue curve).

is a solution of the initial value problem. The main drawback of this technique is its error accumulation. The method increments a solution through an interval h while using derivative information from only the beginning of the interval. Consequently, the step's error is $O(h^2)$ and it is propagated to the next step. For simple applications the resulting path is a sufficient solution to the initial value problem, but in its worst case the calculated solution can differ arbitrarily from the exact result.

Multi-step methods reduce this error accumulation in each step by taking the results of multiple, previous steps into account. In this way, the calculation is not linear with respect to the differential, but of higher order. The methods of CARL D. T. RUNGE[8] and MARTIN W. KUTTA[9] realize this technique (see Table 4.5).

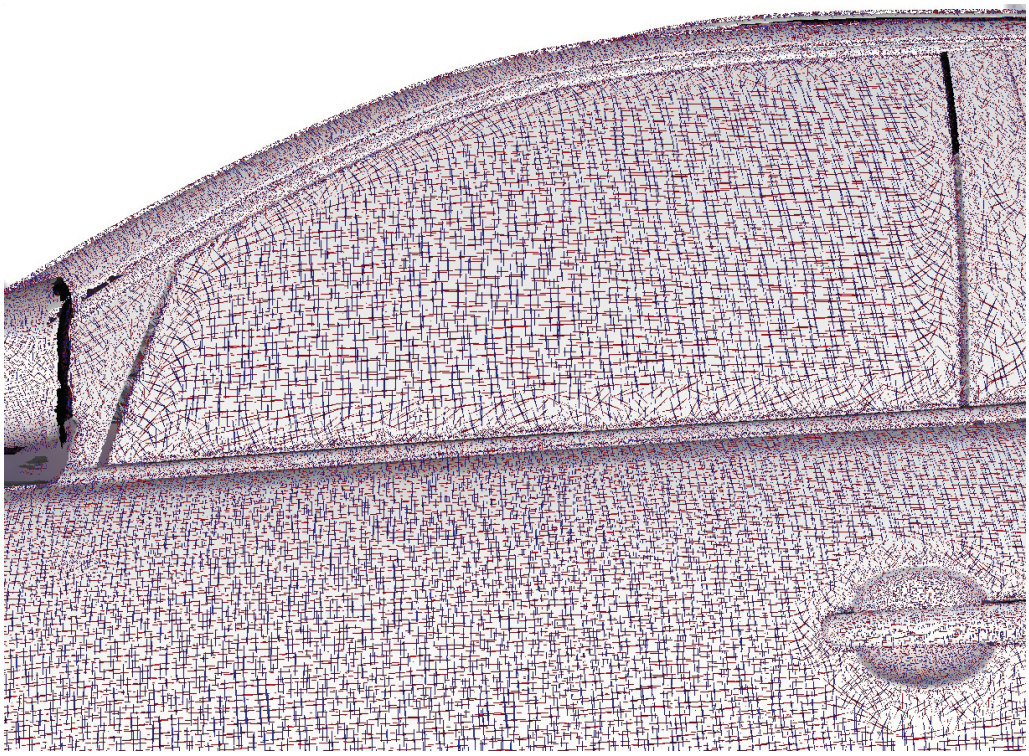


Figure 4.52: Going by curvature flow simplifies the patch layout for a human modeler. In this way the modeling task can be performed more efficiently.

[8] CARL DAVID TOLMÉ RUNGE (August 30, 1856 – January 3, 1927) Carl David Tolmé Runge was a German mathematician, physicist, and spectroscopist. He worked on differential geometry and numerical analysis.

[9] MARTIN WILHELM KUTTA (November 3, 1867 – December 25, 1944) Martin Wilhelm Kutta was a German mathematician. He co-developed the Runge-Kutta method, used to solve ordinary differential equations numerically. Furthermore, his works on aerodynamics are key contributions.

Runge-Kutta Methods

Runge-Kutta methods for first order ordinary differential equations $y' = f(x,y)$.

Second Order

$$\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{2}(k_1+k_2) + O(h^3) \\ k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n+h, y_n+k_1) \end{aligned} \tag{4.139}$$

$$\begin{aligned} y_{n+1} &= y_n + k_2 + O(h^3) \\ k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n+\tfrac{1}{2}h, y_n+\tfrac{1}{2}k_1\right) \end{aligned} \tag{4.140}$$

Third Order

$$\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{6}k_1 + \tfrac{2}{3}k_2 + \tfrac{1}{6}k_3 + O(h^4) \\ k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n+\tfrac{1}{2}h, y_n+\tfrac{1}{2}k_1\right) \\ k_3 &= hf(x_n+h, y_n-k_1+2k_2) \end{aligned} \tag{4.141}$$

$$\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{4}k_1 + \tfrac{3}{4}k_3 + O(h^4) \\ k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n+\tfrac{1}{3}h, y_n+\tfrac{1}{3}k_1\right) \\ k_3 &= hf\left(x_n+\tfrac{2}{3}h, y_n+\tfrac{2}{3}k_2\right) \end{aligned} \tag{4.142}$$

Fourth Order

$$\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4 + O(h^5) \\ k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n+\tfrac{1}{2}h, y_n+\tfrac{1}{2}k_1\right) \\ k_3 &= hf\left(x_n+\tfrac{1}{2}h, y_n+\tfrac{1}{2}k_2\right) \\ k_4 &= hf(x_n+h, y_n k_3) \end{aligned} \tag{4.143}$$

$$\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{8}k_1 + \tfrac{3}{8}k_2 + \tfrac{3}{8}k_3 + \tfrac{1}{8}k_4 + O(h^5) \\ k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n+\tfrac{1}{3}h, y_n+\tfrac{1}{3}k_1\right) \\ k_3 &= hf\left(x_n+\tfrac{2}{3}h, y_n-\tfrac{1}{3}k_1+k_2\right) \\ k_4 &= hf(x_n+h, y_n+k_1-k_2+k_3) \end{aligned} \tag{4.144}$$

Table 4.5: All Runge-Kutta methods are expressions $y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i$ with $k_i = f(t_n + hc_i, y_n + h\sum_{j=1}^{s} a_{ij}k_j)$. The coefficients a_{ij}, b_i, c_i are determined by quadrature formulas – a nonlinear system of equations, which has several solutions. This Table lists two solutions per order. Further details can be found in literature [BP93], [AS72].

Surface design Another important problem in surface reconstruction is the conflict of different objectives: scan interpolation versus surface design [TGRZ07]. On the one hand the resulting surface shall interpolate the laser scan as good as possible, on the other hand it shall meet design conditions such as bounded curvatures. In high-quality surface engineering this trade-off between conflicting objectives is solved locally by a human modeler.

In an ongoing project on high-quality surface engineering with the automotive manufacturer Volkswagen Aktiengesellschaft the researchers SVEN HAVEMANN and TORSTEN ULLRICH develop a curvature-driven modeling tool. Using subdivision surfaces instead of NURBS surfaces, connectivity and smoothness conditions can be omitted and the number of control vertices can be reduced by approximately 25%. These preliminary results will be made up as soon as the tool's evaluation phase and further benchmarks at Volkswagen will be completed.

4.4 Generative Modeling

The term generative modeling reflects a paradigm change in representing shape. The key idea is to identify a shape with a sequence of shape-generating operations, and not just with a list of low-level geometric primitives. This is a true generalization since static objects are equivalent to constant operations that have no input parameters. The benefit of this approach is that parameter dependencies can be expressed, as the output of one operation may serve as input to another. The practical consequence is that every shape needs to be represented by a computer program, i.e., encoded in some form of programming language. Although it is possible to use a general purpose programming language – such as C++ [KK11] – most approaches use a domain-specific description.

JOHN SNYDER [SK92] was the first to implementation the generative paradigm with his C-like shape language GENMOD, followed by CONAL ELLIOTT's TBAG [ESYAE94], and ALBERTO PAOLUZZI's PLaSM [PPV95]. A first general theory of generative shape was developed by MICHAEL LEYTON [Ley01]. This academic development was paralleled by the advent of parametric design in high-end CAD systems in the mid-1990s, pioneered by Pro/Engineer (PTC), soon followed by all its competitors. As a result, today almost all CAD systems use internally a procedural representation, just to mention AutoLISP (AutoCAD), MaxScript (3D Studio Max), and MEL (Maya). Nevertheless, there is still no common exchange standard for procedural models, so that even high-end programs can still exchange only static low-level geometry reliably [Pra04].

With ever increasing computing power becoming available (Moore's Law), generative approaches become more important since they trade processing time for data size. At runtime the compressed procedural description can be "unfolded" on demand to very quickly produce amounts of information that are several classes of complexity larger than the input data. The advantages of the generative approach are:

- complex models become manageable through high-level parameters [HF04],
- models are easier to store and to transmit, as only the process itself is described, not the processed data, i.e., the end result [BFH05],
- changeability and re-usability of existing solutions to modeling problems can be very much improved [HF01], and
- the smaller parameter space can lead to much better results in model-based indexing and retrieval [FH05b].

Generative modeling techniques take advantage of regularities and uniformities. Especially, frequent repetitions and models with complex interdependencies benefit from procedural approaches due to its reusability [GPMG10].

```
Lot -->
    split(z){ ~0.1  : Grass
            | ~1 : BuildingLot #split(x) { ~0.001  : Grass
                               #| ~1 : BuildingLot
                               #| ~0.001  : Grass}
            | 1 : Wall(Gray)}

School -->
    split(z){ ~0.3  : Grass
            | ~1 : split(x) { ~0.2  : Grass
                            | ~1 : SchoolBuilding
                            | ~0.3  : Grass}
            | ~0.3 : Grass}

SchoolBuilding -->
    extrude(25) color(Yellow)


Park -->
    Grass

University -->
    split(z){ ~0.4  : Grass
            | ~1 : split(x) { ~0.3  : Grass
                            | ~1 : UniversityBuilding
                            | ~0.3  : Grass}
            | ~0.3 : Grass}

UniversityBuilding -->
    extrude(30) color(Gray)

BuildingLot -->
    40%  : SixtiesBuildingLot(facade_color)
    else : NeoClasicalBuildingLot(facade_color)
```

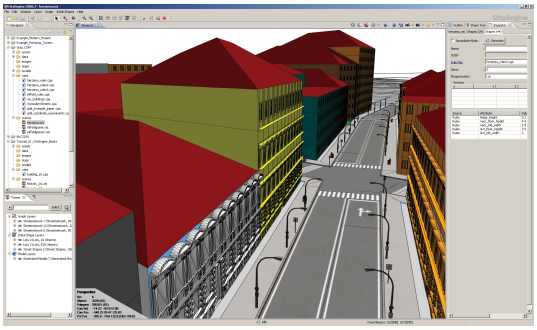


Figure 4.53: The CityEngine modeling environment (right) uses the language CGA shape. This shape modeling language (left) has its roots in L-Systems and split rules. Having split the root geometry node into houses → floors → walls → windows → … the last operations replace each abstract placeholder by prebuilt geometry (OBJ files).

4.4.1 Generative Modeling Techniques

In today's procedural modeling systems, grammars are often used as a set of rules to achieve a description. Early systems based on grammars were Lindenmayer systems [PL90], or L-systems for short. They were successfully applied to model plants. Given a set of string rewriting rules, complex strings are created by applying these rules to simpler strings. Starting with an initial string the predefined set of rules form a new, possibly larger string. The L-systems approach reflects a biological motivation. In order to use L-systems to model geometry an interpretation of the generated strings is necessary. The modeling power of these early geometric interpretations of L-systems was limited to creating fractals and plant-like branching structures. This lead to the introduction of parametric L-systems. The idea is to associate numerical parameters with L-system symbols to address continuous phenomena which were not covered satisfactorily by L-systems alone [DL05].

CGA Shape Later on, L-systems and shape grammars were successfully used in procedural modeling of cities [PM01]. YOGI PARISH and PASCAL MÜLLER presented a system that, given a number of image maps as input, generates a street map including geometry for buildings. For that purpose L-systems have been extended to allow the definition of global objectives as well as local constraints. However, the use of procedurally generated textures to represent facades of buildings limits the level of detail in the results. In later work, MÜLLER et al. describe a system [MZWVG07] to create

detailed facades based on the split grammar called CGA shape. A framework called CityEngine provides a modeling environment for CGA shape. It relies on different views to guide an iterative modeling process. Figure 4.53 shows code defining split rules written in CGA shape.

Another modeling approach presented by MARKUS LIPP et al. [LWW08] following the notation of MÜLLER [MWH+06] deals with the aspects of more direct local control of the underlying grammar by introducing visual editing. The idea is to allow modification of elements selected directly in a 3D-view, rather than editing rules in a text-based environment. Therefore principles of semantic and geometric selection are combined as well as functionality to store local changes persistently over global modifications.

Model Graphs BERND LINTERMANN and OLIVER DEUSSEN proposed a new modeling method as well as a graphical user interface (GUI) for the creation of natural branching structures [LD98]. A structure tree represents the modeling process and can be altered using specialized components describing geometry as well as structure. Another type of components can be used for defining global and partial constraints. Components are described procedurally using creation rules which include recursion. The generation of geometric data according to the structure tree is done via a tree traversal where the components generate their geometrical output.

The procedural modeling approach proposed by BJÖRN GANSTER and REINHARD KLEIN [GK07] describes an integrated framework based on structure trees in a visual language. The infix notation of the language requires the use of variables which are stored on a heap. A graph structure represents the rules used to create an object. Special nodes allow the creation of geometry, the application of operators as well as the usage of control structures. Various attributes can be set for nodes used in a graph. Directed edges between nodes define the order of execution, in contrast to a visual

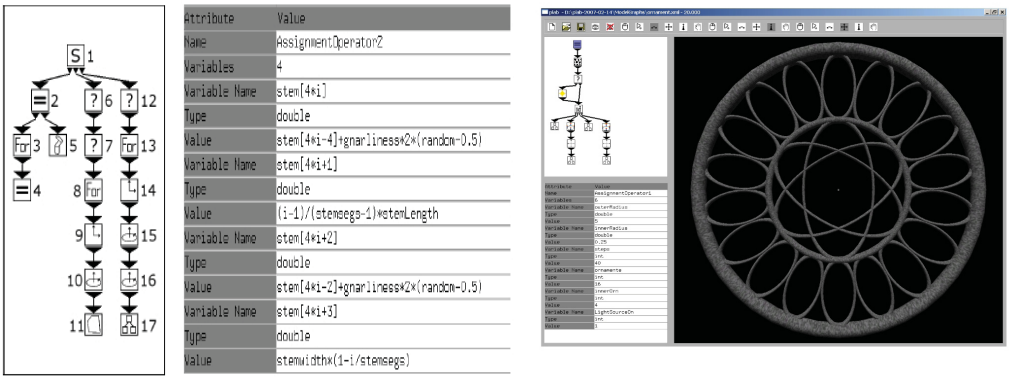


Figure 4.54: The generative modeling environment by GANSTER et al. traverses a model graph (left) with various attributes at each graph node to create geometry. This purely visual programming approach provides graph and attribute editors (middle) in an integrated environment (right) and does not show any source code to users. (*Image source:* [GK07])

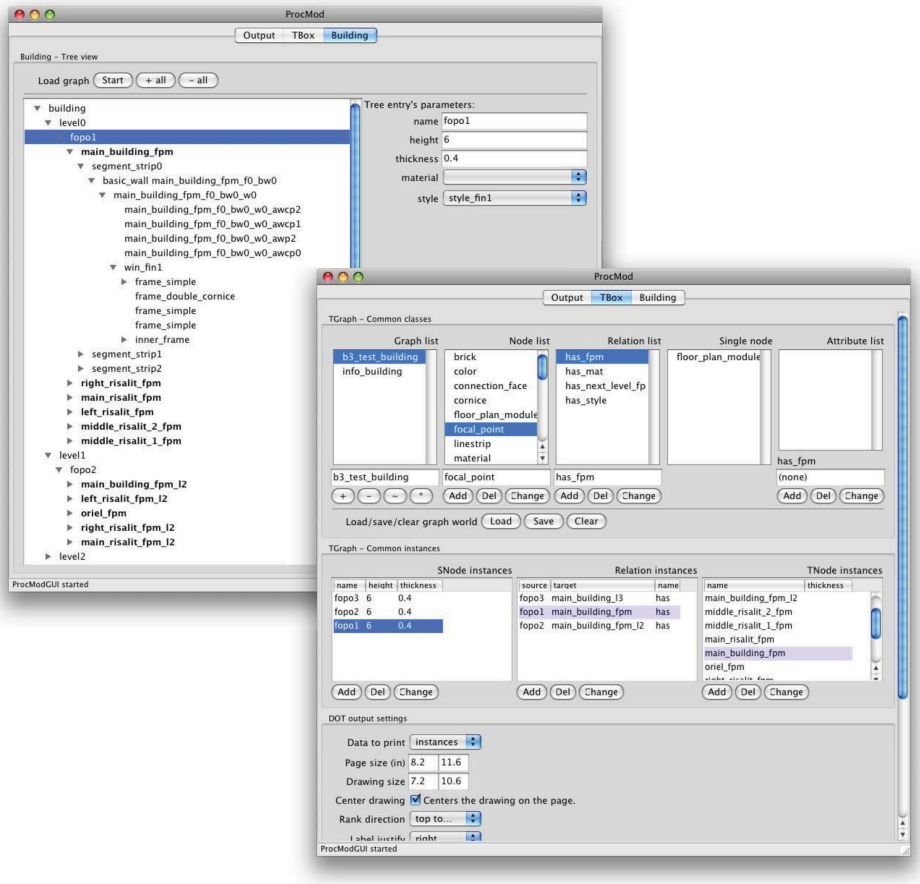


Figure 4.55: The visual modeling environment ProcMod consists mainly of a rule editor (front window) and a scene graph editor (back window). The resulting geometry is passed to Maya. (*Image source:* [Fin08b])

data flow pipeline (VDFP) where data is transported between the different stages. The framework is illustrated in Figure 4.54. It allows fast creation of complex scenes with the limitation that geometry has to be modeled on a rather low level using polygon lists.

Hierarchical Description DIETER FINKENZELLER presented another approach for detailed building facades [Fin08a] called ProcMod. It features a hierarchical description for an entire building. The user provides a coarse outline as well as a basic style of the building including distinguished parts and the system generates a graph representing the building (see Figure 4.55). In the next step, the system traverses the graph and generates geometry for every element of the graph. This results in a generated, detailed scene graph, in which each element can be modified afterwards. The current version has some limitations: for example, organic structures and inclined walls cannot be modeled.

```
# Create the "Hello World" bevelled text...
text = createNode("makeTextCurves")
setAttr(text+".text", "Hello World!", type="string")
setAttr(text+".font", "Times New Roman", type="string")

innerstylecrv = createNode("styleCurve")
outerstylecrv = createNode("styleCurve")
setAttr(innerstylecrv+".style", 0)
setAttr(outerstylecrv+".style", 0)

bevel = createNode("bevelPlus")
setAttr(bevel+".width", 0.1)
setAttr(bevel+".depth", 0.1)
setAttr(bevel+".extrudeDepth", 0.25)
setAttr(bevel+".capSides", 4)
setAttr(bevel+".numberOfSides", 4)
setAttr(bevel+".tolerance", 0.01)
setAttr(bevel+".normalsOutwards", True)
setAttr(bevel+".polyOutUseChordHeight", False)
setAttr(bevel+".polyOutUseChordHeightRatio", False)
setAttr(bevel+".orderedCurves", True)

meshtransform = createNode("transform", name="Hello_World")
mesh = createNode("mesh", name="Hello_World_Shape", parent=meshtransform)

connectAttr(text+".outputCurve", bevel+".inputCurves")
connectAttr(text+".count", bevel+".count")
connectAttr(text+".position", bevel+".position")
connectAttr(innerstylecrv+".outCurve", bevel+".innerStyleCurve")
connectAttr(outerstylecrv+".outCurve", bevel+".outerStyleCurve")
connectAttr(bevel+".outputPoly", mesh+".inMesh")
```

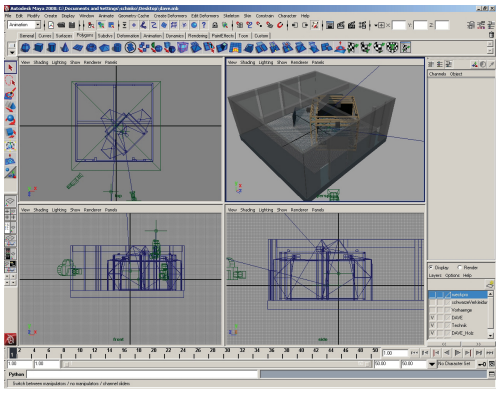


Figure 4.56: Autodesk's 3D modeling software Maya provides a complete tool set for modeling, rendering and animations (right). Its integrated programming language Python allows a user to access all Maya functions (plus additional Python resources) via scripting (left). (*Image source*:left [Baa06])

Scripted Modelers 3D modeling software packages like Autodesk Maya provide a variety of tools for the modeling process. Figure 4.56 (right) shows the graphical user interface. However, a scripting language is supplied to extend the functionality of the GUI. It enables tasks that cannot be achieved easily using the GUI and speeds up complicated or repetitive tasks. For that purpose Autodesk integrates the programming language Python. The left part of Figure 4.56 shows a Python script that generates rendered text as an example output.

When using parametric tools in modern CAD software products, geometric validity is a subject. For a given parametric model certain combinations of parameter values may not result in valid shapes. CHRISTOPH M. HOFFMANN and KU-JIN KIM propose an algorithm [HK01] that computes valid parameter ranges for geometric elements in a plane, given a set of constraints.

Functional Expressions ALBERTO PAOLUZZI suggests a functional language in the context of geometric design programming [PPV95]. The idea is to associate geometric shapes to generating functions and to pass geometric expressions as function parameters. This allows the generation of abstract methods describing geometric shape, as well as calling such methods for the purpose of modeling specific geometry. Although generated objects are always consistent in geometry – due to the validity at a syntactical level – this approach is of rather theoretical interest.

```
[ (0,0,0)
  (0.1,0,0)
  (0.1,0,2.5)
  (0,0,2.5) ] !startPos

[ ] !pos

0 1 8 {
  pop
  :startPos

  3 poly2doubleface

  0 1 8 {
    pop
   (0,2.5,3) extrude
   dup faceCW edgemate facemidpoint :pos exch append

    toggleColor setcurrentmaterial
  } forx
  pop
  :startPos { (0,0,2.5) add } map !startPos
  toggleColor setcurrentmaterial
} forx


:BoardData begin
/std7hi /colorWhite edef
/std2hi /colorBlack edef
/colorBlack /tmpColor edef
end
```

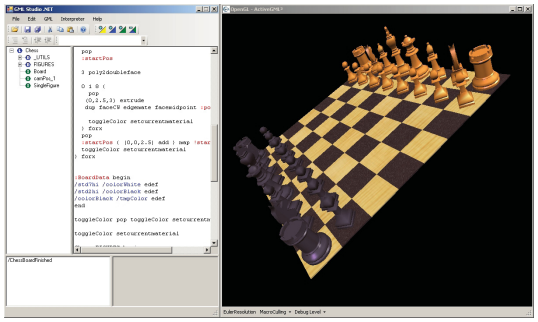


Figure 4.57: SVEN HAVEMANN's approach to procedural modeling is the Generative Modeling Language. It has a PostScript-like syntax (left) in postfix notation; i.e. operators follow their operands (e.g. 1 2 add). The development environment GMLStudio (right) integrates source code editor, outline views and an interpreter. (*Image source:* [CGKV04])

Postfix Expressions The Generative Modeling Language (GML) by SVEN HAVEMANN is a stack-based language for creating polygonal meshes. Its postfix notation takes getting used to [Rei90] and is very similar to that of Adobe's Postscript [Inc85]. It allows the creation of high-level shape operators from low-level shape operators. The GML serves as a platform for a number of applications because it is extensible and comes with an integrated visualization engine.

The generative parametric design of Gothic window tracery [HF04] shows GML's ability to handle complex geometric shapes. Figure 4.57 shows the modeling environment GMLStudio displaying a chess board. Parts of the GML code used to generate the chess board are listed alongside.

An extended system presented by ERICK MENDEZ et al. combines semantic scene graph markups with generative modeling in the context of generating semantic three dimensional models of underground infrastructure [MSH+08]. The idea is to connect a geospatial database and a rendering engine in order to create an interactive application. The GML is used for on-the-fly generation of procedural models in combination with a conventional scene graph with semantic markup. An augmented reality view of underground infrastructure like water or gas distribution systems serves as a demo application.

Generative Modeling Language

The Generative Modeling Language is based on concepts of Adobe Postscript and is designed to describe geometric shapes in 3D. It uses polygonal meshes and subdivision surfaces and provides data types such as numbers, strings, vertices, edges, faces, etc. as well as methods to create and modify these data types.

A commented, exemplary method call to create stairs is:

```
% push base vertices of base
% polygon on GML stack
[ (0,0,0) (10,0,0) (10,1,0)
(0,1,0) ]

% push additional parameters
% on stack
0.2   1   5

% call library function to
% create stairs
Lib.stairs
```

The function to create stairs takes the vertices of a base polygon (and an implicit half-edge), the length of the riser, the length of tread, and the number of steps. The design drawing in Figure 4.58 visualizes the geometric meaning of parameters. A simplified implementation of the generative stairs model respectively the `stairs` function (without any error or reasonability checks) is listed beneath

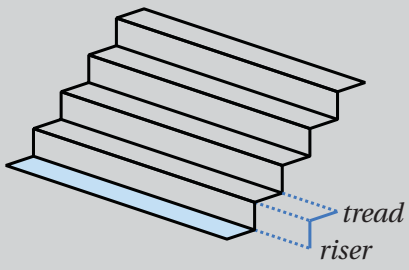


Figure 4.58: The high level parameters of a generative stairs model are its base polygon, the length of the riser, the length of tread, and the number of steps.

```
% pop parameters from stack
% and use them by names
usereg
!nSteps !nTread !nRiser !polyBase

% create base polygon from
% array of vertices
:polyBase 3 poly2doubleface

% initialize first step
0 :nRiser 3 vector3 extrude
dup faceCW edgedirection length
!nDepth

% create following steps
% with a for-loop
1 1 :nSteps
{
   pop edgemate
   0 :nRiser 3 vector3 extrude
   edgemate
   0 :nTread 3 vector3 extrude
   0 :nRiser 3 vector3 extrude
   edgemate
   0 :nDepth 3 vector3 extrude
   edgemate
} for

pop
```

Further information about the Generative Modeling Language, interpreters for various platforms and examples can be found at: http://www.generative-modeling.org

4.4.2 Procedural Model Compilation

JOHN K. OUSTERHOUT categorizes programming languages into low level, system programming and higher level languages [Ous98]. Low level languages (e.g. assembler) reflect virtually all aspects of the CPU. Writing complex programs in low level languages is a tedious task because of the small abstraction possibilities. However, because of the direct mapping of the machine architecture to the language, programs can be optimized for the executing platform (e.g. time critical applications or hardware drivers). System programming languages (e.g. C/C++/C#, Java) provide higher abstraction levels for the programmer and are typically strongly typed. In this context, typing denotes the degree to which the meaning of information is specified in advance of its use. Strong typing enables the compiler to check for data types in advance.

Higher level languages (script languages) are in most cases type-less and allow rapid development. Typically, scripting languages are interpreted; therefore, applications written in such languages provide usually less performance than compiled programs. Scripting languages are well suited to glue components together, combining the advantages of high-performance components and the high abstraction. In addition, scripting languages are generally more flexible.

The importance of scripting techniques comprehends not only areas which are dominated by programming techniques per se but also all kinds of media and environments. The wide range of scripting applications, starting with the first system scripts of mainframe computers, has now spread on server- and client-side internet applications and desktop programs. Scripted content has been included into 3D scene graphs and multimedia systems [RFM95] and is now the basis for all highly customizable environments [JF06].

In the context of modeling, scripting techniques offer a new dimension of collaborative modeling, due to its close relationship to programming. Static content can be exchanged, modified, and assembled via exchange file formats [Ros97]. Dynamic content can also be exchanged, modified, and assembled, but furthermore it allows collaborative modeling in parallel.

In order to modify a static 3D model, the modeler needs the 3D model; whereas a dynamic model can be modified on the basis of its interfaces. In the same way, in which a programming library can be changed independently from the application that uses it (as long as the interfaces remain stable), the procedures of a generative model can be changed. This separation allows the utilization of design patterns (decorator, builder, ...) known from software engineering [FFBS04] in 3D modeling. A decorator function (for example a method, which smoothes all edges) can be modified independently from the code which generates geometry.

As the interpretation of a geometric script is computationally more intensive than the handling of static geometry, optimization techniques, such as just-in-time compilation, are of great interest. Unfortunately, scripting languages tend to support features such as higher order functions or self-modification, etc. These language characteristics are difficult to compile into machine/byte-code. Therefore, we developed a

hybrid approach. In "Compilation of Procedural Models" [UKF08], TORSTEN ULLRICH, ULRICH KRISPEL, and DIETER W. FELLNER present an interpreter with an integrated compiler. In this way we speed up the script evaluation without having to remove any language features e.g. the possibility of self-modifications.

A complete compilation of a GML program would be a hard task due to its dynamic properties. ANTON M. ERTL presented a method to compile FORTH, another stack based language, to native code [Ert96], [EP97]. He creates a data flow graph by interpreting an input file and examining all stack operations, i.e. the input and output parameters of a function. The native code is then generated from this data flow graph. This method implies that the number of input and output parameters of each function is known. In the context of GML this is not possible as the interface (especially the number of input and output parameters) of a function can be changed at runtime. Therefore, a hybrid approach has been chosen. The implemented system is able to translate bounded functions (marked in the source to be permanent/unchangeable) to executable Java code.

In GML each function consists of a sequence of instructions so-called tokens. Every token generates side effects when it is executed. A simple token such as an integer pushes itself on the operand stack. A complex token may produce geometry or perform arbitrary stack operations.

To generate Java source code, the interpreter distinguishes four different stack manipulations – depending on the number of elements put on the stack. Instead of using the operand stack the compiler maintains an internal stack of strings for code generation.

No element Such a function takes a fix number of elements from the stack and does not push any elements on the stack. As long as the internal compiler stack is not empty, elements from this stack are popped. If the internal string stack is empty, source code to pop elements is generated instead (`interpreter.pop()`). These strings are passed to the function operator, which has to provide appropriate source code. The resulting code will then be written to the source file.

Example: The `print` function takes one parameter from the stack and writes it to standard out. If `"Hello World"` is on the internal compiler stack the source code `System.out.println("Hello World")` is generated. If the internal compiler stack is empty, the compiler passes `interpreter.pop()` and the resulting source code is `System.out.println(interpreter.pop())`.

One element This case is very similar to the 'No element'-case. It gets the parameters as strings and generates source code. Instead of writing the generated code into the source file it is put on the internal compiler stack.

Example: The `add` function takes two parameters from the stack and adds them. A pure interpreter pushes the result on the operand stack.

The compiling interpreter keeps the appropriate function call on its internal stack. If `3` and `5` are on the internal compiler stack, the operator may produce `Operator.add(3, 5)`. This line of code is pushed on the compiler stack.

Two or more elements In contrast to the previous cases the operator gets the needed input parameters as well as referenced containers for the results. The generated source code is written to the source file, the container references are put on the compiler stack.

Example: The function `exchange` swaps the two topmost elements of the stack. If `x` and `y` are on the internal compiler stack, the operator gets these strings and source code to access the result containers. The creation of these containers and the resulting code line of the operator are written to the source file:

```
_v1 = new ResultContainer();
_v2 = new ResultContainer();
Operator.exchange(x, y, _v1, _v2);
```

Source code strings to reference the result containers are pushed on the compiler stack.

Unknown number of elements For each element of the compiler stack an appropriate source code line to push it on the interpreter stack is generated. Then the operator's execute method is inserted into the source code. Normally this method is called by the interpreter.

At the end of the translation process the compiler generates source code to put all its internal stack elements on the interpreter stack. This guarantees a consistent interaction between compiled code and the interpreter. Functions created or modified at runtime are interpreted.

Run time analysis has shown a significant performance gain in the compiling version of the interpreter, at the cost of a bigger preprocessing overhead generated by the compilation. However, this preprocessing time gets insignificant for applications where frequent re-evaluation of a model is necessary; e.g. in interactive environments.

An important advantage of procedural modeling techniques is the included expert knowledge within an object description; e.g. classification schemes used in architecture, archaeology, civil engineering, etc. can be mapped to procedures. The generative approach scales with the object's complexity and does not depend on the object's number of vertices. Furthermore, generative models normally have perfect shapes which do not suffer from wear and tear effects. Therefore, they represent an ideal object rather than a real one. The enrichment of measured data with an ideal description enhances the range of potential applications, for example in the field of cultural heritage. A nominal/actual value comparison may indicate wear and tear effects as well as changes in style. But how are these generative models created?

In "Modeling Procedural Knowledge: A Generative Modeler for Cultural Heritage" the authors Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner present a new meta-modeler approach for procedural modeling based on the programming language JavaScript [SSUF10a]. The choice of the programming language was a process of carefully considering pros and cons. JavaScript has a variety of important aspects and features we would like to refer to. It is a structured programming language featuring a rather intuitive syntax, which is easy to read and to understand. As source code is more often read than written, a comprehensible,

well-arranged syntax is useful, which is provided by JavaScript. It also incorporates features like dynamic typing and first-class functions. The most important feature of JavaScript is that it is already in use by many non-computer scientists – namely designers and creative coders [RFM07]. JavaScript and its dialects are widely used in applications and on the Internet: in Adobe Flash (called ActionScript), in the Adobe Creative Suite, in interactive PDF files, in Apple's Dashboard Widgets, in Microsoft's Active Scripting technology, in the VRML97, in the Re-Animator framework, etc. Consequently, a lot of documentation and tutorials to introduce the language exist [VV04]. In order to be used for procedural modeling, JavaScript is missing some functionality, which we added via libraries.

Our meta-modeler approach *Euclides* differs from other modeling environments in a very important aspect: target independence. Usually, a generative modeling environment consists of a script interpreter and a 3D rendering engine. A generative model (3D data structures with functionality) is interpreted directly to generate geometry, which is then visualized by the rendering engine. In our system a model's source code is not interpreted but parsed into an intermediate representation, an abstract syntax tree (AST). After a validation process it is translated into a target language [USF10b]. The process of

parsing → validating → translating

offers many advantages as illustrated in Figure 4.59. The validation step involves syntax and consistency checks. These checks are performed to ensure the generation of a correct intermediate representation (AST) and to provide meaningful error messages as early as possible within the processing pipeline. Meaningful error messages are one of the most, if not the most, important aspect of a beginner-friendly development environment. The consistent intermediate representation serves as a basis for back-end exporters to different languages, different targets / platforms and for different purposes. As our compiler has been designed to translate and export JavaScript to other languages, it includes mechanisms to map JavaScript methods and data types to the target language as well as mechanisms to wrap already existing libraries. The *Euclides* compiler uses annotation techniques to control this mapping and wrapping process. These annotations are placed in JavaScript comments to ensure 100% compliance with the JavaScript standard. In this way low-level, platform dependent functions – such as a method to draw a single shape – are wrapped platform independently. During the bootstrapping process of a new exporter a few low-level functions need to be wrapped in this way. All other functions, methods, etc. are built upon these low-level routines. Consequently, they can be converted and translated automatically.

Currently, the framework offers translators and exporters to HTML/XML (for documentation and publishing), executable Java code and GML code (for visualization), and differentiated Java code (for numerical optimization). The differentiated Java code can be included in objective functions. Having the objective function $f(x_1,\ldots,x_n)$ as well as its partial derivatives $\frac{\partial f}{\partial x_i}$ at hand, it is possible to use standard optimization algorithms to solve a generative minimization problem efficiently. This technique is used, amongst others, in the generative reconstruction approach described in the

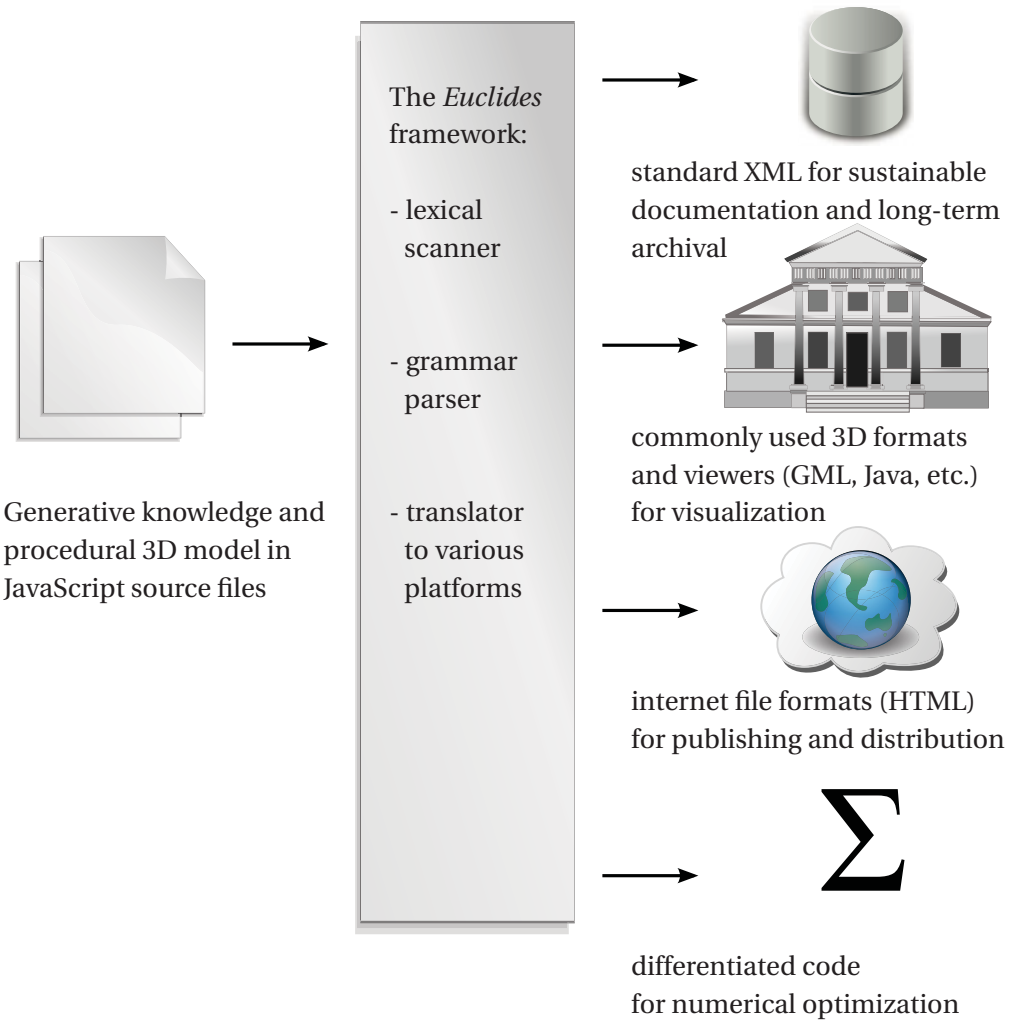


Figure 4.59: The meta-modeler approach has many advantages. Its main characteristic is its platform / target independence with various exporters for different purposes.

next chapter. It is implemented using a JavaScript to Java translator and automatic differentiation techniques (see sidebar on page 52f).

For visualization purposes the framework offers an GML translator. PostScript and all its dialects use reverse Polish notation. Although the JavaScript to GML translation might seem to be a simple infix-to-postfix rewrite for mathematical expressions, the correct translation of control flow structures is a non-trivial task, due to the fact that there is no concept of “goto” in the PostScript language and its dialects.

PostScript and GML are interpreted, stack-based languages with strong dynamic typing, scoped memory, and garbage collection. The language syntax uses reverse Polish notation, which makes the order of operations unambiguous, but reading a

program requires some practice, because one has to keep the layout of the stack in mind [Rei90]. Literals such as numbers and strings are simply put on the stack. Operators and functions take their arguments from the stack, and place their results onto the stack. Complexe data structures can be built on array and dictionary types, which are known to the interpreter, but cannot be declared to the type system. They remain arrays and dictionaries without further type information.

The JavaScript to GML translator is explained in detail in "Euclides – A JavaScript to PostScript Translator" [SSUF10b]. The translation process begins with a correct JavaScript (JS) abstract syntax tree (AST).

Data Types

In JavaScript each variable has a particular, dynamic type. It may be `undefined`, `boolean`, `number`, `string`, `array`, `object`, or `function`. GML also has a dynamical type system. Unfortunately, both type systems are incompatible to each other. Therefore, translating JS-data types to GML poses two particular problems: On the one hand, the dynamic types must be inferred at run time. On the other hand, GML's native data types lack distinct features, for example, GML-Strings cannot be accessed character-wise. We solved these problems by implementing JS-variables as dictionaries in GML. Dictionaries are objects that map unique keys to values. These dictionaries hold needed metadata and type information as well as methods which emulate JavaScript behavior.

The system translation library for GML (which every JS-translated GML program defines prior to actual program code) contains the function `sys_init_data`, which defines an anonymous data value in the sense of JS-data.

```
/sys_init_data
{
   dict begin
   /content dict def
   content begin
     /type edef
     /value edef
     /length { value length } def
   end
   content
   end
} def
```

`sys_init_data` opens a new variable-scope by defining a new, anonymous dictionary and opening it. In this new scope, another newly created dictionary is defined by the name `content`. This content-dictionary receives three entries: `type`, `value` and the method `length`. Each entry value is taken from the top of GML's stack. The newly created dictionary is then pushed onto the stack and the current scope is destroyed

by closing the current dictionary, leaving the anonymous dictionary on the stack. In GML notation, a JS-variable's content is defined by pushing the actual value and a predefined constant to identify the type of the variable (such as `Types.number`, `Types.array`, etc.) onto the stack, and calling `sys_init_data`. Consequently, `var foo = 42;` translates to

```
/usr_foo 42.0 Types.number sys_init_data def
```

As it can be seen, the translator prefixes all JS-identifiers with `usr_` (in order to ensure that all declarations of identifiers do not collide with predefined GML objects) and uses the following translations:

Undefined: Variables of type `undefined` result from operations that yield an undefined result or by declaring a variable without defining it. `var x;` leads to `x` being of type `undefined`. It is translated to

```
/usr_x Nulls.Types.undefined Types.undefined sys_init_data def
```

Boolean: Boolean values are denoted by the keywords `true` and `false`. The translation simply maps these values to equivalent numerical values in GML. The JS-statement `var x = true;` becomes

```
/usr_x 1 Types.bool sys_init_data def
```

Number: All JS-numbers (including integers) are represented as 32-bit floating point values. As GML stores numbers as 32-bit floats internally as well, we simply map them to GML's number representation. For the sake of completeness, the statement `var x = 3.14159;` is translated to

```
/usr_x 3.14159 Types.number sys_init_data def
```

String: Although GML does support strings, they cannot be accessed character-wise. We cope with this limitation by defining strings as GML-arrays of numbers. Each number is the unicode of the respective character. As GML allows to retrieve and to set array-elements based on indexes, this approach meets all conditions of JS-strings. The statement `var x = "Hello";` becomes

```
/usr_x [ 72 101 108 108 111 ] Types.string sys_init_data def
```

Array: JS-arrays allow to hold data with different types, the array's contents may be mixed. This behavior is in line with GML. Therefore, an array has a straightforward translation. `var x = [true, false, "maybe"];` is

```
/usr_x [
   1 Types.bool sys_init_data
   0 Types.bool sys_init_data
   [ 109 97 121 98 101 ] Types.string sys_init_data ]
Types.array sys_init_data def
```

Object: Objects consist of key-value-pairs. This structure is mapped to nested GML-dictionaries. The value of a variable's content is a dictionary of its own. The statement `var x = { x: 1.0, y: 2.0, z: 42};` defines an object of name `x` with key-value-pairs `x` to be 1, `y` to be 2, and `z` to be 42:

```
/usr_x dict begin
   /obj dict def obj begin
     /usr_x 1.0 Types.number sys_init_data def
     /usr_y 2.0 Types.number sys_init_data def
     /usr_z 42.0 Types.number sys_init_data def
   end obj
Types.object sys_init_data end def
```

Opening an anonymous dictionary creates a new scope. In this scope, a dictionary is created and bound to the name `/obj`. It is then opened and its members are defined, just like anonymous variables would be. The object dictionary is then closed, put on the stack, and used to define an anonymous variable. The enclosing anonymous scoping dictionary is then closed and simply discarded.

Function: JavaScript has first-class functions. Therefore, it is possible to assign functions to variables, which can be passed as parameters to other functions. In the following example, a function `function do_nothing() {}` is declared and defined. Afterwards, it is assigned to a variable `var x = do_nothing;` If we abstract away from the translation of the function `do_nothing`, the code `var x = do_nothing;` becomes:

```
/usr_do_nothing {
   %% ... definition of function omitted ...
} def

/usr_x /usr_do_nothing Types.function sys_init_data def
```

The variable `x` can now be used as a functor, which acts the same ways as `do_nothing`. Because such functors can be reassigned, it is necessary to handle functor calls (e.g. `x()`) differently than ordinary function calls (e.g. `do_nothing()`): *Euclides*creates a temporary array, which contains the functor parameters and passes this array as well as the variable referencing the function name to a system function `sys_execute_var`. This system function resolves the functor and determines the referenced function, unwraps the array and performs the function call.

Functions

In GML, functions are defined using closures, such as `/my_add { add } def`. If this function `my_add` is executed, the closure `{ add }` is put onto the stack, its brackets are removed, and the content is executed.

To execute a GML function, its parameters need to be put on the stack prior to the function call: `1.0 2.0 my_add` The result `3.0` will remain on the stack. Please note, that GML functions may produce an arbitrary number of results (left on the stack) at each function call. According to the JavaScript standard, functions always return exactly one value. The default return value is `null`, if nothing is returned otherwise. The number and names of function parameters are known at compile time. Only functors (referenced functions stored in variables) may change at run time and cannot be checked ahead of time.

Scopes As JavaScript uses a scoping mechanism different to GML, it has to be emulated. This is a rather difficult task, which has to take the following properties of scopes into account.

- JavaScript functions may call other functions or themselves.
- Called functions may declare the same identifiers as the calling functions.
- Within functions other functions may be defined.
- Blocks might be nested inside functions, redefining symbols or declaring symbols of the same name.

The translator uses GML's dictionary mechanism to emulate JS-scopes. A dictionary on the dictionary stack can be opened and it will take all subsequent assignments to GML-identifier (variables). Since only the opened dictionary is affected, this behavior is the same as the opening and closing scopes in different scoped programming languages, such as C or Java.

Thus an assignment `/x 42 def` can be put into an isolated scope by creating a dictionary (`dict`), opening it (`begin`), performing the assignment, and closing the dictionary (`end`). The following example shows how such GML scopes can also be nested:

```
dict begin                    %%
   /x 3.141 def               %% x is 3.141
   dict begin                 %%
     /x 4 def                 %% x is 4.0
   end                        %% x is 3.141
end                           %% x is unknown
```

As noted before, JavaScript supports redefinition of identifiers that were declared in a scope below the current one. Fortunately, GML exhibits just the same behavior when reading out the values of variables/keys from dictionaries of the dictionary stack. Consequently, the following example works as expected.

```
dict begin                    %%
   /x 42 def                  %%
   dict begin                 %%
     /y x 1 add def           %% y is now 43
   end                        %%
end                           %%
```

However, assignments to variables have to be handled differently in GML. The Generative Modeling Language does not distinguish between declaration and definition, any declaration must be a definition and vice versa. The translator solves this problem and uses a function of the system translation library called `sys_def`. This function applies GML's `where` operator to the dictionary stack in order to find the uppermost dictionary, where the searched name is defined. The operator returns the reference to the dictionary, in which the name was found.

Control Flow for Functions The Generative Modeling Language and all PostScript dialects lack a dedicated jump operation in control flow. Imperative functions often require the execution context to jump to a different point in the program – and to return from there as well. Fortunately, GML provides an exception mechanism. A GML exception is propagated down GML's internal execution stack until a `catch` instruction is encountered. In this way it overrides any other control structure it encounters.

```
/usr_foo {
   dict begin
   /return_issued 0 def
   { dict begin
     %% ... function body omitted ...
     end }
   { /return_issued 1 def }
   catch

   return_issued not
   { Nulls.Types.undefined Types.undefined sys_init_data } if
   end
   sys_exception_return_handler
} def
```

In this empty function skeleton, the function opens a new anonymous scope. Inside this scope `dict begin ... end` the local identifier `/return_issued` is set to `0`. Afterwards a GML try-catch-statement `{ try_block } { catch_block } catch` contains the JS-function implementation. The catch block redefines `/return_issued` to `1` to indicate that a JS-`return` statement has been executed in the function body. JS-functions without any `return` statement, automatically return `null`. A corresponding JS-return statement, e.g., `return 42;`, is translated to

```
42.0 Types.number sys_init_data end throw
```

In this example, the number `42.0` is put onto the stack. The actual function body's scope is closed `end`, and the `throw` operator is applied. The distinction of whether the end of the function body was reached by normal program flow or via a return statement determines, if a return value needs to be constructed (`null`) and put onto the stack.

Parameters to functions are simply put on the stack. The function body retrieves the expected number of parameters and assigns them to dictionary entries of the outer scope defined in the function translation. A complete example of a translated JS-function shows the interplay of all mechanisms.

The simple function

```
function foo(n) {
   return n;
}
```

is translated to

```
/usr_foo {
   dict begin
   /usr_n edef
   /return_issued 0 def
   { dict begin
     usr_n
     end
     throw
     end }
   { /return_issued 1 def }
   catch

   return_issued not
   { Nulls.Types.undefined Types.undefined sys_init_data } if
   end
   sys_exception_return_handler
} def
```

A function call, for example `foo(3)`, yields the translation

```
3.0 Types.number sys_init_data usr_foo
```

If the function `foo` is assigned to a variable `foo_functor`, the calling convention in GML would change significantly:

```
/usr_foo_functor /usr_foo Types.function sys_init_data def
```

is called via

```
[ 3.0 Types.number sys_init_data ] usr_foo_functor sys_execute_var
```

and represents the call `foo_functor(3.0);`

Exceptions The programming language JavaScript supports throwing exceptions; e.g., `throw "Error: unable to read file.";`. Its syntax is similar to a return statement. To implement such behavior, the *Euclides* translator adds a call to the predefined system function `sys_exception_return_handler` at the end of each translated function (see example above).

Throwing an exception translates into a global GML variable `exception_thrown` being set to 1, closing the current dictionary and calling GML's `throw`. The system function `sys_exception_return_handler` will check, if an actual exception is being thrown, and if so, calls `throw` again. A catch-block inside a JavaScript program would set the variable `exception_thrown` to 0.

Operators

The evaluation of expressions demands variables to be accessed. While GML provides operators that operate on their own set of types, they obviously cannot be used to access the translated/emulated JS-variables. For this reason, the *Euclides* translator automatically includes a set of predefined GML functions that substitute operators defined in JavaScript.
Performing the opposite operation to `sys_init_data`, `sys_get_value` will retrieve the data saved in a JS-variable resp. its GML-dictionary. The system function `sys_get` implements string, array and object access.

```
/sys_get {
   dict begin
   /idx exch def /var exch def

   var.type Types.string eq {
     %% ... handling strings ...
   } if

   var.type Types.array eq {
     %% ... handling arrays ...
   } if

   var.type Types.object eq {
     var sys_get_value idx known 0 eq {
       %% return null, if element doesn't exist
       Nulls.Types.undefined Types.undefined sys_init_data
     } if
     var sys_get_value idx known 0 ne {
       %% access element
       var sys_get_value idx get
     } if
   } if
   end
} def
```

Applied to a string / an array `Arr` and index `k`, it will return the element `Arr[k]`. If its parameters are an object `Obj` and an attribute `name`, the function `sys_get` executes `Obj.name`. This may result in a value, which is put on the stack or in a function, which is called. Conforming to JavaScript, it returns `undefined` for any requested elements that do not exist.
Analogous to `sys_get`, `sys_put` inserts data into strings and arrays, or defines members of objects. If `sys_put` encounters an index `k` that is out of an array's range, the array is resized and filled with values `undefined`.

The already mentioned routine `sys_execute_var` inspects a given variable. If it is a function, it will retrieve the array supplied to hold all parameters and execute the function. The dynamic binding of functions to variables requires to consider two situations at run time: The functor receives the correct amount of parameters for its function, or the number of parameters does not correspond to the referenced function. In the later case, the function is not called and `null` is returned instead.

At compile time, a function is defined to expect a concrete number of parameters. This information is kept to perform parameter checks at run time. In this way, the correct number of parameters for all functors can be determined any time.

Control Flow

The if-then-else statement corresponds one-to-one to the same GML statement. Consequently, the conditional expression is translated straightforwardly. Using the expression mapping introduced in the previous section (e.g. `sys_eq` implements the equality operator), the JS-statement `if(a == b) { c = a; } else { c = b; }` is translated info:

```
%% if (a==b)
usr_a usr_b sys_eq sys_get_value
{ %% then:
   dict begin {
     dict begin
       /usr_c usr_a sys_def
     end
   } exec end
}
{ %% else:
   dict begin {
     dict begin
       /usr_c usr_b sys_def
     end
   } exec end
} ifelse
```

The exec-statements (and their closures) stem from the fact that both sub-statements, the then-part and the else-part, are statement blocks `{ ... }`. These blocks are executed within their own, new scopes.

The Generative Modeling Language supports different types of looping control structures, which have similar names to JS-loops (e.g., both languages have a for-loop). However, the GML counterparts have different semantics (e.g., GML's for-loop has a fixed, finite number of iterations, which is known before execution of the loop body, whereas JS-loops evaluate the stop condition during execution, which may result in endless loops). The *Euclides* translator uses the GML mechanism, which is an infinite loop that can be quit using the `exit` operator.

An important problem is that control structures such as for, while and do-while are not only controlled by the loop's stop condition, but also by JS-statements such as `continue` and `break` within the loop body (besides `return` and `throw` as mentioned before). The statement `break` immediately stops execution of the loop and leaves it, whereas `continue` terminates the execution of the current loop iteration and continues with the next iteration of the loop. Therefore, an empty while loop, for example `while(false) { ... }`, is translated to

```
{ /continue_called 0 def
   { 0 Types.bool sys_init_data
     sys_get_value not { exit } if
     { dict begin
       %% ... loop body omitted ...
       end
     } exec
   } loop
   continue_called not { exit } if
} loop
```

GML's `exit` keyword terminates the current loop. This behavior is leveraged by the *Euclides* translator to implement `break` and `continue`. It uses two nested loops that will run infinitely. Prior to the begin of the inner loop `/continue_called` is set to `0`. At the top of the inner loop, the loop condition is tested. If the condition evaluates to `false`, the inner loop is exited using GML's `exit`. Otherwise a new scope is created and the loop-statement executed within that scope. During loop iterations, there are three scenarios under which a loop can terminate:

1. If the loop condition is met: When the condition evaluates to `false`, the inner loop is exited. Since `continue_called` is not set to `true`, the outer loop will terminate as well.

2. If the loop body encounters `break` (resp. GML `exit`): Again, the inner loop is left. `continue_called` will not be set to `true`, hence the outer loop will also terminate.

3. If the function returns: GML's exception throwing mechanism will unwind the stack until the catch-handler at the end of the function is encountered.

If the loop body encounters a JS-`continue` statement, `continue_called` will be set to `true` and the GML `exit` command will immediately stop the inner loop. Since the variable `continue_called` is set, execution does not leave the outer loop, however. As a consequence, `continue_called` becomes `0` again, and execution re-enters the inner infinite loop.
The do-while-statement is translated very similar to the while-statement. The only semantic differences in JavaScript are that execution will enter the loop regardless of the loop-condition and that the loop-condition is tested after loop body execution.

Due to a semantic difference of JS-`continue` in do-while-loops, this statement needs to be handled differently. If `continue` is encountered, the loop condition must still execute before the loop body is re-entered, because side effects inside the loop condition may occur (such as incrementing a counter). *Euclides* handles this problem.
Although GML has a `for` operator, it is semantically incompatible with JavaScript's one. Its increment is a constant number, and so is the limit. In JavaScript, both increment and limit must be evaluated at each loop body execution. Therefore, `for` translates just like the previous constructs by two nested loops with the increment condition repeated in outer loop (due to `continue` semantics). The statement

```
for(var i=0; i<1; i++) { }
```

becomes

```
dict begin
%% initialization (i=0)
/usr_i 0.0 Types.number sys_init_data def
{ /continue_called 0 def
   { %% condition (i<1)
     usr_i 1.0 Types.number sys_init_data sys_lt
     sys_get_value not { exit } if
     { dict begin
       %% ... loop body ...
       end
     } exec
     %% increment (i++)
     usr_i
       usr_i 1 Types.number sys_init_data sys_add
     /usr_i sys_edef
     pop
   } loop
   continue_called not { exit } if
   %% increment again (i++)
   usr_i
     usr_i 1 Types.number sys_init_data sys_add
   /usr_i sys_edef
   pop
} loop
end
```

The alternative for-in statement `for(var x in array) statement;` is semantically equivalent to:

```
for(var i=0; i<array.length; i++) {
   var x=array[i];
   statement;
}
```

This construction loops over the elements of an array and provides the loop body with a variable holding the current element.

The translation of the `switch` statement poses several difficulties:

- If a case condition is met, execution can "fall through" till the next `break` is encountered.
- If a `break` is encountered, the currently executed `switch` statement must be terminated.
- Of course, `switch` statements may be nested.

To develop a semantically consistent solution without changing the translation of JS-`break` inside switch statements (compared to loops), the `switch` statement is translated to a loop that is run exactly once. In GML it reads like `1 { body } repeat`. This way the translation of `break` shows semantically correct behavior, it terminates the loop. Consider the following JavaScript program:

```
var x = 0, y = 0;

function bar() {
   return 3;
}

function foo(i) {
   switch(i) {
     case 0:
     case 1:
     case 2:
       x = 1;
     case 4:
       x = 3;
     case bar():
       x = 2;
       break;
     default:
       y = 5;
   }
}
```

The function `foo` is translated to:

```
/usr_foo
{ dict begin
   /usr_i edef
   /return_issued 0 def
   { dict begin
     /switch_cnd_met1 0 def
     1 { usr_i 0.0 Types.number sys_init_data sys_eq
       sys_getvalue switch_cnd_met1 1 eq or {
         /switch_cnd_met1 1 def
       } if

       usr_i 1.0 Types.number sys_init_data sys_eq
       sys_getvalue switch_cnd_met1 1 eq or {
         /switch_cnd_met1 1 def
       } if

       usr_i 2.0 Types.number sys_init_data sys_eq
       sys_getvalue switch_cnd_met1 1 eq or {
         /switch_cnd_met1 1 def
         %% x = 1;
         /usr_x 1.0 Types.number sys_init_data sys_def
       } if

       usr_i 4.0 Types.number sys_init_data sys_eq
       sys_getvalue switch_cnd_met1 1 eq or {
          /switch_cnd_met1 1 def
         %% x = 3;         /usr_x 3.0 Types.number sys_init_data sys_def
       } if

       usr_i usr_bar sys_eq
       sys_getvalue switch_cnd_met1 1 eq or {
         /switch_cnd_met1 1 def
         %% x = 2;
         /usr_x 2.0 Types.number sys_init_data sys_def
         exit
       } if
     %% y = 5;
     /usr_y 5.0 Types.number sys_init_data sys_def
   } repeat
   currentdict /switch_cnd_met1 undef end
   }
   { /return_issued 1 def } catch

   return_issued not
   { Nulls.Types.undefined Types.undefined sys_init_data } if
   end
   sys_exception_return_handler
} def
```

This example shows an introduced, internal variable `/switch_cnd_metX` for traversing the case statements. As soon as a case statement condition is met, the variable `/switch_cnd_metX` is set to `true`, leading execution into every encountered case statement. The *Euclides* translator takes into account that switch statements may be nested. As it traverses the AST, it keeps book of all internal variable to ensure a unique name (`switch_cnd_met1`, `switch_cnd_met2`, ...).

The example translation shows that for `foo(3)` only the case 3 (= `bar()`) will be executed. The interpreter will break out of the `1 { } repeat` statement due to the `exit` operator. The default block will be executed in any case if execution is still inside the `repeat` statement; no further state is checked for `default`.

Example

To demonstrate the interplay of all translational building blocks, this section shows a non-recursive, subtraction-based version of the Euclidean algorithm to calculate the greatest common denominator and its translation to GML. It can be shown by induction that two successive Fibonacci numbers (named after LEONARDO FIBONACCI[10]) are the computational worst-case of the Euclidean algorithm. We use them as input data.

```
function fibonacci(index) {
   switch (index) {
     case 0:
     case 1:
       return 1;
     default:
       return fibonacci(index-2) + fibonacci(index-1);
   }
}

function gcd(a,b) {
   if (a == 0)
     return b;

   while (b != 0)
     if (a > b)
       a = a - b;
     else
       b = b - a;
   return a;
}

var x = gcd(fibonacci(5), fibonacci(6));
```

The corresponding GML code is:

[10] LEONARDO FIBONACCI (1180 – 1241) Leonardo Pisano Bigollo also known as Leonard Fibonacci was an Italian mathematician. His important book “Liber Abaci” spread the Hindu-Arabic numeral system in Europe.

```
%% function fibonacci(index) {
%%    switch (index) {
%%      case 0:
%%      case 1:
%%        return 1;
%%      default:
%%        return fibonacci(index-2) + fibonacci(index-1);
%%    }
%% }
```

```
/usr_fibonacci {
   dict begin
   /usr_index edef
   /return_issued 0 def
   { dict begin
     /switch_cnd_met1 0 def
     1 { usr_index 0.0 Types.number sys_init_data
       sys_eq sys_getvalue switch_cnd_met1 1 eq or {
         /switch_cnd_met1 1 def
       } if

       usr_index 1.0 Types.number sys_init_data
       sys_eq sys_getvalue switch_cnd_met1 1 eq or {
         /switch_cnd_met1 1 def
         1.0 Types.number sys_init_data
         end throw
       } if

       usr_index 2.0 Types.number sys_init_data sys_sub usr_fibonacci
       usr_index 1.0 Types.number sys_init_data sys_sub usr_fibonacci sys_add
       end throw
     } repeat
     currentdict /switch_cnd_met1 undef end
   }
   { /return_issued 1 def } catch
   return_issued not
   { Nulls.Types.undefined Types.undefined sys_init_data } if
   end
   sys_exception_return_handler
} def
```

```
%% function gcd(a,b) {
%%    if (a == 0)
%%      return b;
%%
%%    while (b != 0)
%%      if (a > b)
%%        a = a - b;
%%      else
%%        b = b - a;
%%    return a;
%% }

/usr_gcd {
   dict begin
   /usr_a edef
   /usr_b edef
   /return_issued 0 def
   { dict begin
     usr_a 0.0 Types.number sys_init_data
     sys_eq sys_getvalue
     { usr_b end throw }
     {}
     ifelse

     { /continue_called 0 def
       { usr_b 0.0 Types.number sys_init_data
         sys_ne sys_getvalue not { exit } if

         usr_a usr_b sys_gt sys_getvalue
         { /usr_a usr_a usr_b sys_sub sys_def }
         { /usr_b usr_b usr_a sys_sub sys_def }
         ifelse exec
       } loop
       continue_called not { exit } if
     } loop
     usr_a end throw
     end
   }
   { /return_issued 1 def } catch
   return_issued not
   { Nulls.Types.undefined Types.undefined sys_init_data } if
   end
   sys_exception_return_handler
} def

%% var x = gcd(fibonacci(5), fibonacci(6));

/usr_x
   6.0 Types.number sys_init_data usr_fibonacci
   5.0 Types.number sys_init_data usr_fibonacci
   usr_gcd
def
```

While this translation is a simple infix-to-postfix notation rewrite for mathematical expressions (e.g. the function call `gcd(fibonacci(5), fibonacci(6))` becomes basically `6 fibonacci 5 fibonacci gcd`), the correct translation of control flow structures is a non-trivial task, due to the fact that there is no concept of `goto` in the PostScript language and its dialects. "Euclides – A JavaScript to PostScript Translator" [SSUF10b] shows the first translation of JavaScript into a PostScript dialect including *all* control flow statements.

As *Euclides* offers a new access to GML, all GML users will benefit from its results. The possibility to use GML via a JS-to-GML translator reduces the inhibition threshold significantly. Everyone, who knows any imperative, procedural language (Pascal, Fortran, C, C++, Java, etc.) is familiar with the language concepts in JS and can use *Euclides*. Advanced GML users, who already know how to program in PostScript style, can use *Euclides* to translate algorithms, which are often presented in a imperative, procedural (pseudo-code) style [CSLR01]. In this way, algorithms and mathematical routines, such as a Cholesky, eigenvalue, or singular value decomposition, can easily be included in hand-written code.

Euclides has a beginner-friendly syntax and is able to generate and export procedural code for various, different generative modeling or rendering engines. This innovative meta-modeler concept allows a user to export generative models to other platforms without losing its main feature – the procedural design. In contrast to other modelers, the source code is not interpreted but translated. These translators can transform the source code to different languages; furthermore, they can perform "non-standard" compilation task, such as differentiate a function with respect to its input parameters.

5 Reconstructive Geometry

Generative modeling techniques have influenced the modeling process significantly. The resulting models may appear realistic, but they are not the reconstruction of some real objects and their geometry.

The process to bring the geometry of a real object together with a suitable shape template and to extract its main characteristic parameters is known as shape recognition and reverse engineering. This task is highly related to the shape description problem.

Reverse engineering of generative models offers a new advantage: semantic enrichment. With more and more virtual objects in model repositories, algorithms gain importance, which are able to recognize a shape, to extract its main parameters and therefore to classify a model and to enrich it semantically. This chapter offers a new approach to this challenge and presents a proof of concept.

Contents

5.1 Information Extraction

In the context of information extraction, the question to begin with is: Which semantic information can a three dimensional model be expected to contain? According to "The Meaning of 3D Shape and some Techniques to Extract It" by SVEN HAVEMANN, TORSTEN ULLRICH, and DIETER W. FELLNER [HUF11] 3D data sets are used for conveying very different sorts of information. A 3D scanning process typically produces a number of textured triangle meshes, or maybe just a large set of colored points. So a single 3D scan is conceptually very much like a photograph; it is a result of an optical measuring process, only with additional depth information. One 3D scan may contain many objects at the same time, or a set of 3D scans may contain different views of the same object. The notion of an object is highly problematic in this context, of course, and must be used with care. It may change as function of interpretation and query context.

5.1.1 2D/3D Analogy

The strong analogy between 2D images and 3D objects is useful and illustrative. Therefore, extracting semantic information can be done in 3D with similar techniques as in computer vision, for example, segmentation, object recognition, object retrieval, shape/image matching, etc. In contrast to 2D data, three dimensional objects can be arranged hierarchically, which is seldom done with images. A scene graph typically has a root node representing the origin, inner nodes are transformations, and leaf nodes contain actual 3D objects. This can express coarse-to-fine transformation chains, for example:

city → quarter → house → floor → room → table → cup.

In this case objects are typically moveable things, but can also be semantic units. The roofs and floors of a house may be separated simply because the user wishes to be able to move them in order to show what is inside the house. If the parts are not in separate scene graph nodes, they cannot be moved separately.

To summarize, the main difference between 2D images and 3D objects is that images are typically treated as self-contained units. Segmentation is typically applied for recognition purposes (foreground-background separation), but rarely for cutting an image into pieces that are stored separately and re-combined to create new images. With 3D data, this is done routinely. The consequence is that in 3D, the spatial inter-object relations are more changeable, and more significant: Collision of 3D objects is more significant than 2D collision, which typically means only that one object occludes another from the point of view of the camera. In some sense, of course, 3D subsumes 2D since an arbitrary number of virtual photographs can be shot from a 3D scene.

5.1.2 Semantic Gap

The problem of extracting semantic information from 3D data can be formulated simply as *What is the point?* to express that it is a-priori not clear whether a given point belongs to a wall, to a door, or to the ground. To answer this question is called semantic enrichment and it is, as pointed out, always an act of interpretation.

According to SVEN HAVEMANN and DIETER W. FELLNER [HF07] several research challenges have to be met. The most important challenge is the semantic gap and the meaning of shape. The goal is to assign a meaning (car, house, screw) to a given 3D model only by considering its geometry. This entails classical questions such as measures for shape similarity, shape retrieval, and query-by-example. But also more fine-grained questions such as determining dimensions, parameters, part-of relationships as well as symmetries, self similarity (ornaments, patterns) and speculation about deteriorated parts need to be considered.

Assuming that the meaning of a shape was determined, how can that information be stored in a sustainable way? Currently, there is no commonly accepted, domain independent method to store and exchange the meaning of a shape. Note that this not only requires solving the previous problems, but it additionally requires a common approach for knowledge engineering, e.g., using standardized shape ontologies to express the relations between the different shapes.

Currently, the 3D production and acquisition pipeline does not consider shape semantics sufficiently. Techniques to digitize shape are currently becoming available to a wide audience. Also shape modeling, i.e., the creation of synthetic 3D models has become more accessible due to proliferation of free, easy-to-use 3D modeling software, e.g., Google SketchUp [GL06]. As a consequence, masses of 3D data are produced. Therefore, the problems of missing shape semantics are very urgent.

Shape acquisition is a measurement process using dedicated devices like a computer tomograph, or a laser scanner, or simply sequences of uncalibrated photographs to which photogrammetry and computer vision techniques are applied [KPVG00]. In any case, the result of the measurement is typically a point cloud, either with color per point, or with texture coordinates per point and a set of texture images. The next (non-trivial) processing step converts the point cloud to a higher-level geometric surface description with less redundancy: Note that also a perfectly planar surface can yield millions of points when it is 3D-scanned, although maybe four corner points would be sufficient to represent the shape with high accuracy. This goes without saying that creating a surface from a set of points is in fact already an interpretation; strictly speaking, it is a hypothesis.

5.1.3 Digital Libraries

The idea of generalized documents is to treat multimedia data, in particular 3D data sets, just like ordinary text documents, so that they can be inserted into a digital library. For the digital library to be able to handle a given media type, it must be integrated with the generic services that a library provides.

Documents, Metadata, and Annotations

In 1998 WILLIAM J. CLINTON announced at the 150th Anniversary of the American Association for the Advancement of Science that "the store of human knowledge doubles every five years". With increasing knowledge the process of knowledge management and engineering becomes more and more important. Enriching documents by using markup techniques and by supporting semantic annotations is a major technique for knowledge management. It allows an expert to establish an interrelationship between a document, its content and its context.

Annotations made by groups or individuals, in the context of teamwork or individual work allow to capture contextual information, which can improve and support cooperative knowledge management policies; i.e. annotations can be considered under the perspective of documentation. In fact, tracking the changes and focal points of annotations implies tracing the underlying reasoning process.

This invaluable information is of extreme importance in the context of civil engineering, product life cycle management, virtual archival storage, and preservation. In these fields of applications annotation techniques for 3D documents are a vital part.

Documentation standards and annotation processes are used in various fields of applications. Unfortunately, each branch of science has slightly different definitions of bibliographical terms. TORSTEN ULLRICH, VOLKER SETTGAST and RENÉ BERNDT clarify these terms in "Semantic Enrichment for 3D Documents – Techniques and Open Problems" [USB10]:

A document is any object, "preserved or recorded, intended to represent, to reconstruct, or to demonstrate a physical or conceptual phenomenon". This definition has first been verbalized by SUZANNE BRIET in her manifesto on the nature of Documentation: "Qu'est-ce que la documentation?" [Bri51]. In MICHAEL K. BUCKLAND's article "What is a document?" various document definitions are given and compared to each other [Buc97].

A distinct, separate subpart of a document is called entity. Other authors refer to a subpart as segment. Metadata about documents or parts of documents are defined as "structured, encoded data that describe characteristics of information-bearing entities to aid in the identification, discovery, assessment, and management of the described entities." The American Library Association formalized this definition in its Task Force on Metadata Summary Report [oC99]. According to this definition metadata is always structured. Unstructured, encoded data, such as comments and free texts, are hereinafter called annotations. As metadata is always structured, it can be specified in a formal, explicit way: an ontology is a "formal, explicit specification of a shared conceptualisation". It provides a shared vocabulary, which can be used to model a domain; i.e. the type of objects and/or concepts that exist, and their properties and relations.

TOM GRUBER established this definition in his article "A translation approach to portable ontology specifications" [Gru93]. The connections between a document and its metadata or annotations are called markup instructions. They provide local or global reference points in a document.

In the context of computer-aided design a document is very often the result of a process chain. Data describing a single processing step or a document's process chain is termed paradata.

Metadata and annotations – semantic information in general – can be classified in several ways. Depending on the field of application, they can be classified according to the following criteria.

Document data type Semantic information enriches a document. As documents can be grouped according to their type, these categories can be transferred to metadata and annotations as well. In the context of 3D data the most common representations are: Boundary representation, point clouds, volume data.

Scale of Semantic Information Semantic information can be added for the entire data set or only for a fragment of the object. For some metadata like "author" it can be sufficient to mark the entire document. But 3D data creation is often a collaborative task with many people working on one complex object. For comments and detailed descriptions a specific place within the 3D data set (via anchor) is needed.

Type of Semantic Information The "Metadata Encoding & Transmission Standard" of the Library of Congress defines three types of metadata and annotation: Descriptive information (e.g. the Dublin Core metadata set [SBW02]), administrative metadata (e.g. intellectual property rights), structural metadata (e.g. hierarchical structure of a digital library object).

Type of creation The creation of semantic enrichment of 3D documents fall basically in two categories: manual or automatic. Most of the metadata (especially administrative and descriptive metadata) can be generated automatically, but depending on the domain certain fields need support from an expert.

Data organization The data organization is an important aspect thinking of the sustainability of the annotation. There are two basic concepts how programs can store annotations: Within the original documents or separated.

Information comprehensiveness Semantic enrichment can be further classified by the comprehensiveness of the information. The amount of comprehensiveness can vary from low to high in any gradation. An example for a low comprehensiveness would be the Dublin Core metadata set [Ini95] with 15 properties. In contrast, the CIDOC Conceptual Reference Model (CRM) is a scheme with a high comprehensiveness. It is a framework for the definition of relationship networks of semantic information in the context of cultural heritage [Gro03]. It offers a formal ontology with 90 object classes and 148 properties (in version 5.0.1) to describe all possible kinds of relations between objects.

This defines a library in terms of the function it provides – namely markup, indexing, and retrieval [Fel01], [FSK07]. Like any library, a digital library contains meta information for all data sets. This is insufficient for large databases with a huge number of 3D objects, because of their versatility and rich structure. Scanned models are used in raw data collections, for documentation archival, virtual reconstruction, historical data analysis, and for high-quality visualization for dissemination purposes [SUF07]. Navigation and browsing through the geometric models must be possible not only in 3D, but also on the semantic level. This cannot be done, if the library simply treats 3D data as binary large objects as it is done quite often.

5.2 Shape Description

While describing a 3D model on the geometric level is a problem that has been researched reasonably well, it is still an open-ended question how to describe the shape and its structure on a higher, more abstract level [LZQ06]. Several approaches are described in "Multimedia Information Extraction" [May11], which also illustrates the complexity of the task.

5.2.1 Description by Definition

The traditional way of classifying objects, pursued both in mathematics and, in a less formal manner, in dictionaries, is to define a class of objects by listing their distinctive properties:

> cup – a small, open container made of china, glass, metal, etc., usually having a handle and used chiefly as a receptable from which to drink tea, soup, etc.
>
> http://dictionary.reference.com

This approach is not amenable for computers not only because of the natural language used, but more fundamentally because of the fact that definitions typically depend on other definitions (e.g., container, china, glass, etc.). This often leads to circular dependencies that cannot be resolved automatically by strict reasoning, but rely on intuitive understanding at some point (hen – egg). The dictionary example also illustrates the difficulty of making implicit knowledge explicit: Most people will agree on which objects fall into the class "cup", but it is far more difficult to synthesize an explicit definition for this class. So building up a dictionary of definitions is a sophisticated and tedious task.

5.2.2 Taxonomic Examples

An alternative, non-recursive approach for describing shape is to use a picture dictionary. Each entry in the dictionary is illustrated with a photo or a drawing. This description by example approach is widely-used, for example in biology for plant taxonomy. This avoids listing an exhaustive list of required properties for each entry. However, it requires some notion of similarity and classification, simply because the decision whether object $\mathbf{x}$ belongs to class A or B requires measuring the closeness of $\mathbf{x}$ to the exemplars $\mathbf{a} \in A$ and $\mathbf{b} \in B$ of both classes.

5.2.3 Statistical Approaches and Machine Learning

A large body of literature on 2D segmentation, detection, recognition, and matching exists in the field of computer vision, based on machine learning techniques. Many of these approaches use a classifier. A classifier decides to which class an object $\mathbf{x}$ belongs; or more formal: it is a function f that maps input feature vectors $\mathbf{x} \in X$ to output class labels $y \in C = \{1, \ldots, k\}$. The feature space X is often $\mathbb{R}^d$.

The goal in machine learning is to derive f from a set of labeled training data $(\mathbf{x}_i, y_i)$. Probabilistic approaches compute the posterior probability $P(y|\mathbf{x})$ that feature vector $\mathbf{x}$ belongs to the class y. Then, the classifier can simply choose the class with the highest probability,

$$P(y = c|\mathbf{x}) = \max_{c \in C}. \tag{5.1}$$

Modeling the a-posteriori probability directly is called a discriminative model, since it discriminates between given classes. A generative model, however, uses Bayes' rule, Equation (2.67), to compute the posterior probabilities using

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y) \cdot P(y)}{\sum_{c \in C} P(\mathbf{x}|c) \cdot P(c)}. \tag{5.2}$$

The advantage is that $P(\mathbf{x}|c)$ and $P(c)$ can be learned separately, which makes the classifier more robust against partly missing data, it can handle combined features, and new classes can be added incrementally. Details can be found in the computer vision literature [Bis07], [UB05].

Many of the machine learning techniques are applicable to 3D problems [OB07], e.g., for feature-based similarity search [MGGP06], [FS06], [GCO06], [FMA+10]; some even use computer vision techniques directly. 2D computer vision is clearly ahead with respect to machine learning, so much progress can be expected when more of these techniques are lifted to 3D [JH99]. The new term visual computing was coined for the confluence of graphics and vision.

Statistical approaches clearly have their strength in discriminating object classes. However, it is generally difficult to describe objects with a "flat" feature vector. Furthermore 3D objects often have a rich hierarchical structure, e.g., a hierarchy of joints

or a graph of rooms connected by doors. In addition, feature-based object detection, e.g., of rectangular shapes, does not yield object parameters: width and height of a detected rectangle must typically be computed separately.

A good survey on content-based 3D object retrieval is provided by BENJAMIN BUSTOS et al. [BKSS07]. One example is the approach from DING-YUN CHEN et al. [CTSO03], who calculate the similarity between a pair of 3D models (taken from a 3D database) by comparing heuristically chosen sets of 2D projections rendered from the two models. Each projection is then described by image features, for instance the silhouette. The similarity between two objects is then defined as the minimum of the sum of distances between all corresponding image pairs over all camera rotations.

5.2.4 Algorithmic Description

In "Semantic Fitting and Reconstruction" TORSTEN ULLRICH, VOLKER SETTGAST and DIETER W. FELLNER present an approach to describe objects via generative scripts and parameters [USF08b]. In this case a class of geometric objects is described by a generative script respectively by an algorithm. The question whether an object belongs to a class is reduced to the question whether the object can be generated by the class' algorithm.

This combination of geometry and algorithm design offers a big advantage. Due to its close relationship to programming, generative modeling offers a new dimension of collaborative modeling. Static content can be exchanged, modified, and assembled via exchange file formats [Ros97]. Dynamic content can also be exchanged, modified, and assembled, but furthermore it allows collaborative modeling in parallel. In order to modify a static 3D model, the modeler needs the 3D model; whereas a dynamic model can be modified on the basis of its interfaces. In the same way, in which a programming library can be changed independently from the application that uses it (as long as the interfaces remain stable), the procedures of a generative model can be changed. This separation allows the utilization of design patterns (decorator, builder, ...) known from software engineering [FFBS04] in 3D modeling. A decorator function (for example a method, which smoothes all edges) can be modified independently from the code which generates geometry. To illustrate this possibility Figure 5.1 shows the result of a generative model highlighting independent subroutines in different colors. These building blocks can be arranged freely to describe a class of buildings (see Figure 5.2).

Due to the naming of functions and algorithms as well as possible markup techniques, procedural model libraries are the perfect basis for digital library tasks. The advantages of procedural modeling arise from the fact that generative models have perfect shapes. Therefore they represent an ideal object rather than a real one. The enrichment of measured data with an ideal description enhances the range of possible applications. This connection – an inverse problem called reverse engineering – is a great challenge as pointed out in "Procedural methods for 3D reconstruction" [Arn06].

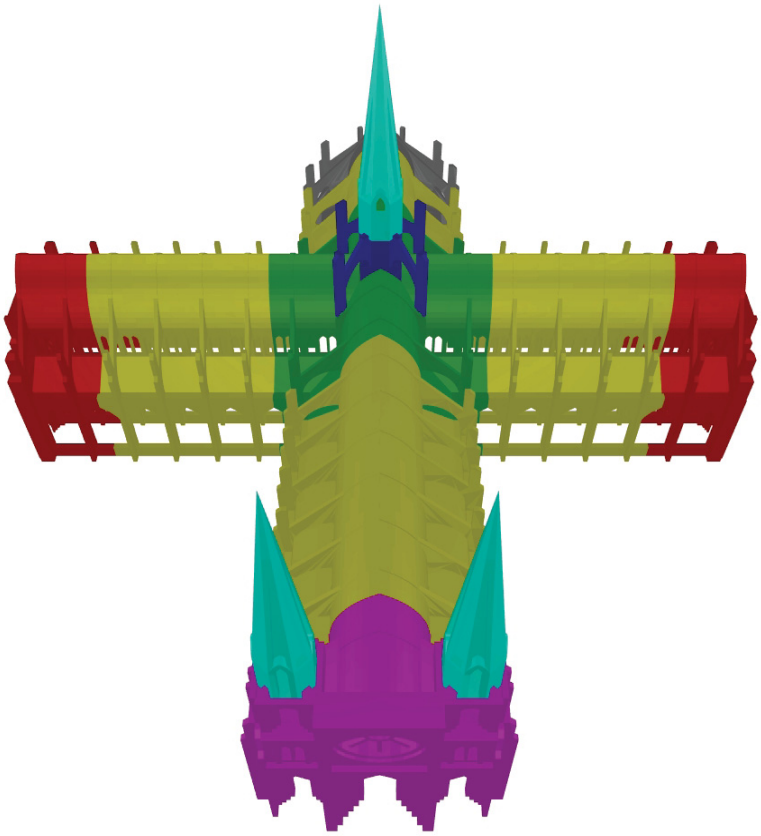


Figure 5.1: Gothic architecture is defined by strict rules with its characteristics: pointed arcs, the ribbed vaults, and the flying buttresses. These building blocks (highlighted in different colors) can be identified in every building of Gothic style. The building blocks have been created by MICHAEL CURRY, *thingiverse*.

5.3 Reverse Engineering

Given a shape, the reverse engineering problem is to answer the question: How has this shape been created? This is an inverse problem in the sense that it tries to infer the construction process. Simple examples would be to infer from points sampled from a sphere the center and radius of the sphere; or, given the shape created by a milling process, to compute the path of the milling tool. This shows that reverse engineering always makes certain assumptions about the underlying design space, i.e., the tools that were used for creating a given object. These assumptions may be wrong: Fitting a sphere to an ellipsoid or to a box yields bad results.

According to the definition by Tamás Vradi et al. [VMC97], reverse engineering requires identifying the structure of the model and the creation of a consistent and accurate model description. It comprises a number of different problems and techniques such as fitting, approximation, and numerical optimization, described in more

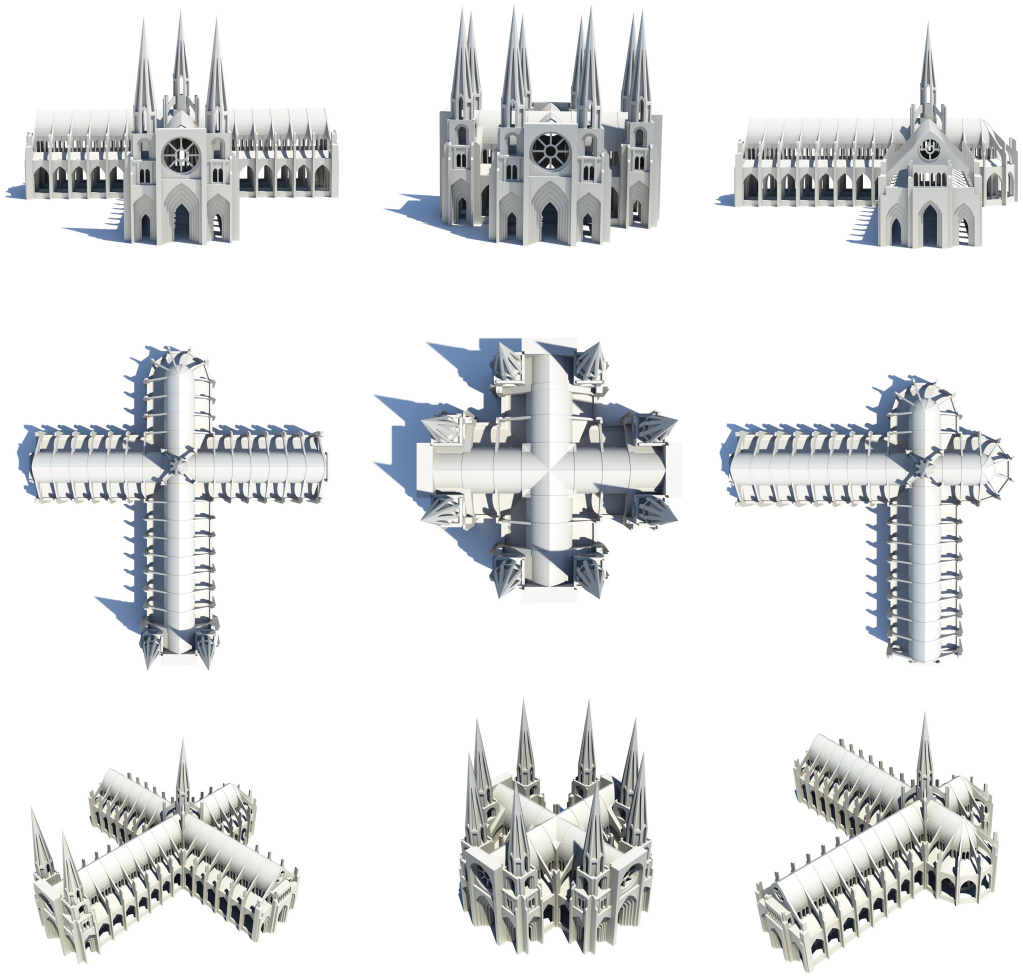


Figure 5.2: Gothic architecture flourished during the high and late medieval period. Its building blocks (pointed arcs, the ribbed vaults, and the flying buttresses) have been combined in various ways to create great churches and cathedrals all over Europe.
Using these building blocks in a generative system it is easy to generate Gothic examples (left, middle, right). This generative description and the rules it is based on can be used to define a class of Gothic architecture.

detail in the processing pipeline from FAUSTO BERNARDINI et al. [BBCS99]. Applying knowledge to reverse engineering problems improves the recovery of object models [Fis02]: "computers are good at data analysis and fitting; humans are good at recognizing and classifying patterns." ROBERT FISHER demonstrated that general shape knowledge enables to recover an object, even if the given input data is very noisy, sparse, or incomplete.

One example of a well-established, complete reverse engineering pipeline is the field of urban reconstruction. Raw data, unorganized 3D point clouds, are captured using aerial imagery processed photogrammetrically, optionally complemented by aerial or terrestrial laser scans. Using strong assumptions about the objects to be reconstructed, excellent results can be obtained fully automatically by now [KBK+01, FZ03, Rem03]. This yields a well defined set of semantic information, i.e., ground polygons and building heights as well as the "roof landscape" [ZKGGK06]. Unfortunately, this semantic information is highly domain dependent and thus, not very generally applicable. It is difficult to extend the information model to represent the number of floors, windows, entries, and walking paths, or by a detailed street model.

5.3.1 Structural Decomposition

Urban reconstruction is in fact an example for structural decomposition. The idea is to postulate a certain type of semantic structure in the data, typically "part-of" relations, and then to search and extract this structure in unstructured data such as point clouds or triangle sets.

Structural decomposition can be implemented in various ways; e.g. TAHIR RABBANI and FRANK VAN DEN HEUVEL use the constructive solid geometry (CSG) paradigm (see [Sha02]) where primitive objects (box, sphere, cylinder etc.) can be added to or subtracted from each other. So they decompose a triangulated object into a tree of CSG operations with primitive objects in the tree leafs [RvdH04]. The used primitives (box, sphere, etc.) are often part of man-made objects. Therefore, they are also used by RAOUL WESSEL and REINHARD KLEIN to learn "the Compositional Structure of Man-Made Objects for 3D Shape Retrieval" [WK10].

KIN-SHING D. CHENG et al. [CWQ+04] decompose a triangulated free-form surface into a subdivision surface. Similar to splines, subdivision surfaces define a smooth surface with a comparably coarse control mesh.

Structural decomposition is well in line with human perception. In general, shapes are recognized and coded mentally in terms of relevant parts and their spatial configuration or structure. While this was only postulated, e.g., in the influential Gestalt theory [KW05] in the late 19th century, psychologists like IRVIN BIEDERMAN have found also empirical evidence [Bie87]. One idea to operationalize this concept was proposed, among others, by MASAKI HILAGA [HSKK01], who introduces an interesting structural descriptor, the Multiresolution Reeb Graph, to represent the skeletal and topological structure of a 3D shape at various levels of resolution. Another school around BIANCA FALCIDIENO and MICHELA SPAGNUOLO [AIM06] is pursuing the idea of shape

ontologies. They propose, in the context of shape retrieval, the notion of a shape prototype represented as attributed graph with nodes containing shape descriptors [BMSF06], [MSF07].
A simple and elegant conceptual framework to extract primitive shapes is the random sample consensus (RANSAC) paradigm by MARTIN A. FISCHLER and ROBERT C. BOLLES [FB81]. In the context of 3D pattern recognition and reconstruction this technique is capable of extracting a variety of different types of primitive shapes out of unstructured, noisy, sparse, and incomplete data (see sidebar "Random Sample Consensus" on page 38f). RUWEN SCHNABEL et al. have presented a RANSAC-based framework that detects planes, spheres, cylinders, cones, and tori in massive point clouds. They use the detected objects as shape proxies that are much more efficient to render than the point cloud [WGK05, SWK07]. The approach by PÁL BENKO et al. refines this idea to process a point cloud by using a hierarchy of tests, i.e. a tree where in each node a decision is taken which kind of primitive to choose for the fitting process [BKV+02].

Another interesting refinement of the same idea has been done by RUWEN SCHNABEL et al. [SWWK07]. In addition to the detected shape they also consider the geometrical neighborhood relations between these shapes and store them in a topology graph. A query graph captures the shape configuration to be detected, for instance a pair of symmetrically slanted planes describing a gabled roof. These query graph templates represent the knowledge about the shape of an entity. The templates have to be provided by the user, again by making implicit knowledge explicit. The matching of a semantic entity to the data then corresponds to a subgraph matching of the topology graph, which can be carried out automatically.

5.3.2 Symmetry Detection

One very active branch in the field of geometry processing is the detection of shape regularities. An obvious problem is instance detection of parts and subparts. MARK PAULY and others detect symmetries on multiple levels, e.g., for architectural buildings [PMW+08], or even to detect that the deformed body of an animal is symmetric [MGP06]. The latter approach can be further extended to un-deform and straighten out a deformed symmetric shape so that it becomes symmetric to a plane [MGP07].

MARTIN BOKELOH, MICHAEL WAND, and HANS-PETER SEIDEL use a symmetry detection approach [BWS10] to create generative model descriptions. Given some 3D geometry, they find a set of rules that describes new objects, which are similar to the given one considering local similarity [BBW+08], [BWM+11], i.e., each local neighborhood of a newly created object must match some local neighborhood of the given exemplar.

To summarize, structural decomposition proceeds by postulating that a certain type of general regularity or structure exists [WXL+11] in a class of shapes [SBM+10].

Besides semantic challenges, the creation of consistent and accurate model descriptions – focused on geometry – is known as reverse engineering and reconstruction. To outline the coherences between existing reconstruction approaches it makes

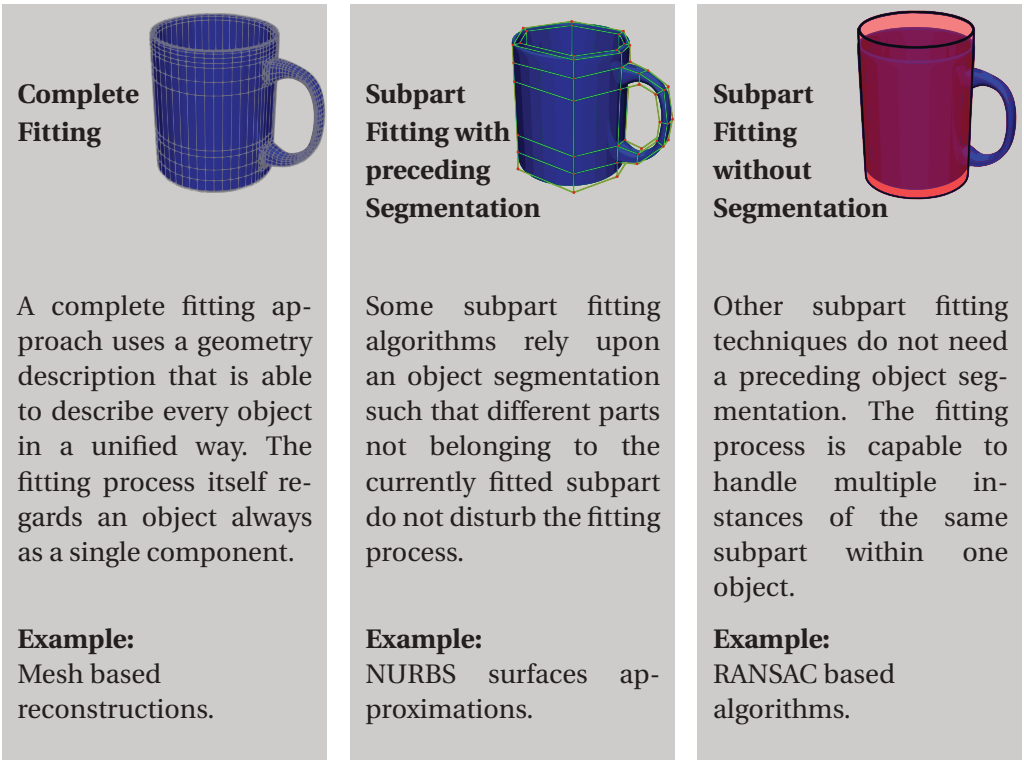


Figure 5.3: The currently known solutions to reverse engineering can be group by the underlying geometric model description. The differences arise from the uniformity of a description and its need of a pre-segmentation.

sense to classify them into three groups (see Figure 5.3) according to the model description capabilities and the need of a preceding model segmentation.

If the underlying model description is able to describe every three dimensional object in a consistent and integrative way without the need of an additional superstructure, the fitting process can be called complete fitting. Otherwise an object is described by several small parts whose orientation to each other is stored in a superstructure. This approach is a subpart fitting process.

5.3.3 Complete Fitting

In 1992 HUGUES HOPPE et al. presented an algorithm that fits a polyhedral surface to an unorganized cloud of points [HDD+92]. The result is a polygonal mesh which describes the complete object. Further development of polygonal reconstruction has lead to algorithms whose output is guaranteed to be topologically correct and convergent to the original surface as the sampling density increases [ABK98]. Besides polygonal reconstruction algorithms [GJ02], approaches based on radial basis functions

[CBC+01], constructive solid geometry [RvdH04], or subdivision surfaces [CWQ+04], [MMTP04], [CWQ+07], are able to describe three-dimensional objects in a unified way and are representatives of the complete fitting class.

5.3.4 Subpart Fitting with Segmentation

The subpart fitting approaches can be divided into two classes depending on whether the input data has to be segmented and partitioned or not.

The preprocessing step of segmentation uses feature extraction algorithms, which determine feature lines such as crease loops and junctions, or border lines [GWM01]. MARK PAULY et al. used principal component analysis on local neighborhoods to classify points according to the likelihood that they belong to a feature [PKG03]. Using local feature vectors to segment and partition the input data into smaller regions, where each of which can be approximated by a single patch [GG04], is a common approach.

In combination with a special sequence of tests [PTK05] a large point cloud can be robustly split into smaller and smaller subregions until no further subdivision is sensible. The spectrum of partitioning approaches reaches from local feature extraction [WBK08] to global shape recognition [HOP+05], [VGSR04], [ABS06] and variational shape approximation [CSAD04], [WK05]. The detection of axes of reflection or intrinsic symmetries of a model [MSHS06], [MZWVG07], [SKS06] as well as geometrical classifications using clustering [KT03] or line geometry [PWL01] offer further segmentation possibilities.

These analysis techniques are needed for example by reconstructions based on non-uniform rational B-splines (NURBS) [WPL04], developable surfaces [Pet04], or least squares techniques [Sha98].

5.3.5 Subpart Fitting without Segmentation

No preceding segmentation is needed among others by RANSAC-based algorithms [FB81], [DDSD03], [WGK05]. The basic idea of RANSAC is described in “Random Sample Consensus” on page 38f.

A method to create generative models from point clouds or range data of 3D objects has been presented by RAVI RAMAMOORTHI and JAMES ARVO in [RA99]. The algorithm uses a hierarchy of generative model templates. The root node of the hierarchy is a very simple geometric shape. Each child node within the hierarchy is a refined version of its parent. For a given point cloud the algorithm starts to fit the root node to the input data. It determines the template's free parameters represented by spline curves which match the point cloud best. Afterwards it fits the node's children to the point cloud as well. The child node with the best results according to an error function is selected to be the new parent node whose children are fitted next. In this way the algorithm refines the generative model until the end of the hierarchy is reached or the fitting error is underneath a user-defined threshold.

This approach creates concise generative models from incomplete and sparse data, but it is not able to differentiate between objects on a semantic level. Two objects which have been fitted by the same hierarchy may differ in their free parameters which are spline curves. Therefore the hierarchy of rotating generalized cylinders may represent many different objects: a banana as well as a candle-holder or a coffee mug.

5.4 Generative Object Definition and Semantic Recognition

Generative models offer a possibility to describe a shape [USF10b]. The key idea is to encode a shape with a sequence of shape-generating operations, and not just with a list of low-level geometric primitives. In its practical consequence, every shape needs to be represented by a program, i.e., encoded in some form of programming language [ÖK08], shape grammar [MWH+06], modeling language [Hav05] or modeling script [Aut07].

Within this "definition by algorithm" approach, each class of objects is represented by one algorithm M. Furthermore, each described object is a set of high-level parameters $\mathbf{x}$, which reproduces the object, if an interpreter evaluates $M(\mathbf{x})$. As this kind of modeling resembles programming rather than "designing", it is obvious to use software engineering techniques such as versioning and annotations. In this way, model M can contain a human-readable description of the object class it represents.

This encoding of semantic information can be used by the algorithm developed by TORSTEN ULLRICH and published in "Robust Shape Fitting and Semantic Enrichment" [UF07b]. Enhancements to this algorithm are described in "Semantic Fitting and Reconstruction" [USF08b], [USF09]. The latest developments, discussed in "Generative Object Definition and Semantic Recognition" [UF11b] by TORSTEN ULLRICH and DIETER W. FELLNER, enriches 3D objects semantically: the algorithm starts with a geometric object O and a generative model M. Without user interaction it determines a parameter set $\mathbf{x}_0$, which minimizes the geometrical distance between O and $M(\mathbf{x}_0)$. This distance d can be interpreted as a multidimensional error function of a global optimization problem. Therefore, standard techniques of function minimization can be used. Having found the global minimum $\mathbf{x}_0$, the geometric distance $d(O,M(\mathbf{x}_0))$ can be interpreted. A low value corresponds to a perfect match; i.e. the 3D data O is (at least partly) similar to $M(x)$, whereas a high value indicates no similarity. Consequently, the presented approach is able to semantically recognize instances of generative objects in real data sets.

As the computational complexity of global optimization depends on the dimensions of the error function, our approach uses a hierarchical optimization strategy with coarse model descriptions and few parameters at the beginning and detailed model descriptions at the end. This multi-step optimization determines free parameters successively, fixes them and introduces new parameters. This process stops, if the end of the hierarchy is reached, or if high error values indicate no object similarity.

If the generative model is regarded as a function $M(\mathbf{x})$, $\mathbf{x} \in G \subset \mathbb{R}^k$, the objective function minimizes the distance between a geometric object, without loss of generality a point cloud $\mathbf{P} = \{P_1, \ldots, P_n\}$, and a procedural description $M(x)$; i.e. $d(P, M(x))$. Due to numerical stability a weighting function is used. It reduces the disproportional effect of outlying points [Zha97]. Consequently, the objective function is

$$f(\mathbf{x}) = \psi(d(\mathbf{P}, M(\mathbf{x}))) \stackrel{!}{=} \min. \tag{5.3}$$

5.4.1 Distance Function

Section 3.3 discusses the main definitions and properties to measure a distance between two point sets. Without loss of generality it is sufficient to cover point sets; i.e. even an instance of a procedural model $M(\mathbf{x})$ will temporarily be regarded as a point set $\{M_1(\mathbf{x}), \ldots, M_m(\mathbf{x})\}$.

A fully automatic, distance-based fitting algorithm has to determine whether a fitting result is "sensible" or not; i.e. it has to interpret the measured distance. To simplify this interpretation task, the distance of a perfect match should be zero. Although each metric has to meet the identity condition (Equation (3.16)) and the symmetry condition (Equation (3.17)), a perfect match does not imply zero distance. In most fitting scenarios – especially in hierarchical fitting approaches – the generative model only describes a subpart. Therefore, even a perfect match, for example an identified, generative window within a scanned point cloud of a complete building, will have a non-zero distance. Oriented distances, which are characterized by

$$d(\mathbf{X}, \mathbf{Y}) \neq d(\mathbf{Y}, \mathbf{X}), \qquad \text{with point sets } \mathbf{X}, \mathbf{Y}, \tag{5.4}$$

solve this problem. As a consequence,

$$d(\mathbf{P}, M(\mathbf{x})) = \sum_{i=1}^{m} d(\mathbf{P}, M_i(\mathbf{x})) \tag{5.5}$$

the sum of all distances from a generated sample point M_i to the point cloud $\mathbf{P}$ satisfies the zero distance requirements mentioned above.

As the distance between a single point X and a point set $\mathbf{Y}$ is defined by the minimum of all distances between X and a point $Y \in \mathbf{Y}$, respectively

$$d(X, \mathbf{Y}) = \min_{Y \in \mathbf{Y}} d(X, Y), \tag{5.6}$$

a perfect match of a generative subpart has distance zero.

5.4.2 Weighting Function

The most commonly-used weighting function is the squared distance

$$\psi_{LSQ}(x) = x^2, \qquad x \in \mathbb{R}. \tag{5.7}$$

Squared distances to curves and surfaces do not only appear in geometric reconstruction problems, but also in registration tasks in computer vision and positioning problems in robotics. Due to the importance of the squared distance function, great effort has been made to understand the geometry of the function, which associates to each point in space the square of the shortest distance to a given curve or surface [PH03a]. In combination with an octree data structure which stores in each of its cells a local quadratic approximant of the squared distance function of a geometric object, many geometric optimization problems – like registration or surface approximation, where the solution to the problem is found iteratively with a Newton-type method – can be solved very efficiently [LPZ03].

The sum of the squares of the distances between data points and a model is used to treat the residuals as a continuous differentiable quantity. In many cases a simplified version (e.g. using only a vertical offset) is used instead of the Euclidean distance allowing a much simpler, linear analytic form.

The usage of squared distances is a mixed blessing. On the one hand it avoids square root operations. On the other hand outlying points may have a disproportionate effect on the fit. Even worse, if a model shall be fitted to a data set, least squares methods implicitly assume that the entire set of data can be interpreted by one parameter vector of the model. If this condition is not met (as illustrated in Figure 5.4), the data set has to be partitioned into regions fulfilling this condition. A reasonable weighting function is an alternative to a preceding segmentation. Furthermore, a weighting function should preserve the properties of the metric used to compute the distance. Consequently it should meet the following conditions:

1. The weighting function ψ should be non-negative, so that outliers cannot compensate each other.

2. $\psi(0) = 0$ should be fulfilled, in order to identify the best-fit solution independently of the free parameters.

Considering the requirements for a numerical, iterative solver,

3. the weighting function should be monotonic increasing on the interval $(0,\infty)$ and monotonic decreasing on the interval $(-\infty,0)$. Otherwise additional local minima complicate a numerical solution.

4. ψ should have two continuous derivatives ($\psi \in C^2$) to avoid an unnecessary reduction of the numerical tools at hand.

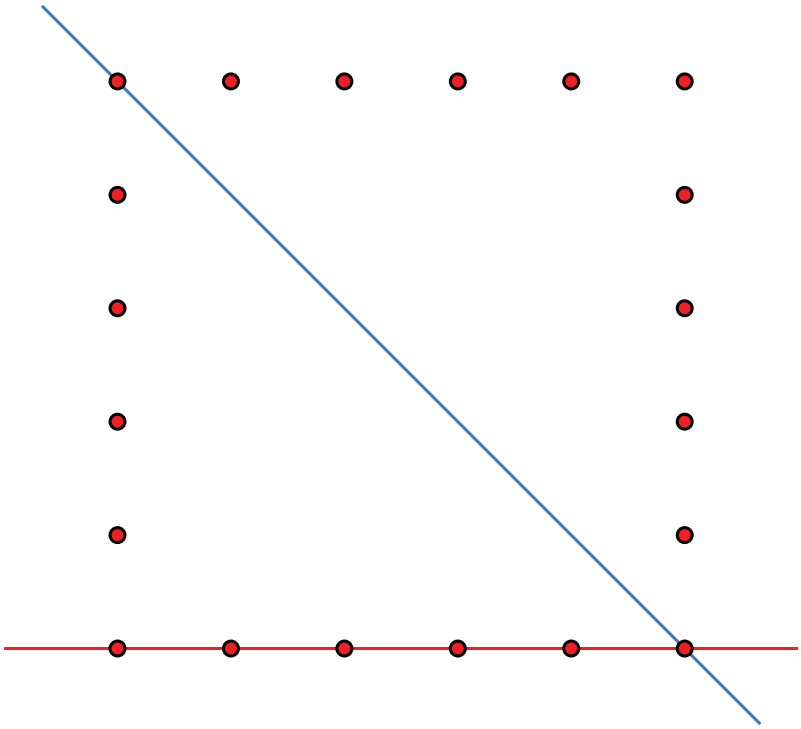


Figure 5.4: Fitting a line to a point set is a frequently needed task in reverse engineering and regression analysis. Using a least squares approach to approximate points of a regularly sampled square by a line leads to undesirable results (blue). The solution to this problem either uses a segmentation algorithm to separate the four sides and to fit them separately or uses a fitting approach, which is robust towards outliers (red).

All these conditions apply to least squares fitting $\psi_{LSQ}(x) = x^2$. To overcome the sensitivity of the least squares approach, the area of statistics has developed a wide range of methods to circumvent the limitations of traditional parametric and non-parametric methods [MMY06]; in particular, generalized maximum likelihood estimators [Zha97] and robust parameter estimation techniques [Ste99] are of big importance in this context.

The decreasing exponential approach uses a Gaussian weighting function

$$\psi_{EXP}(x) = 1 - e^{-x^2/\sigma^2}. \tag{5.8}$$

This function meets the conditions (1) to (4) mentioned previously. To overcome the disproportionate effect of outlying points the weighting function ψ_{EXP} has two important, additional properties:

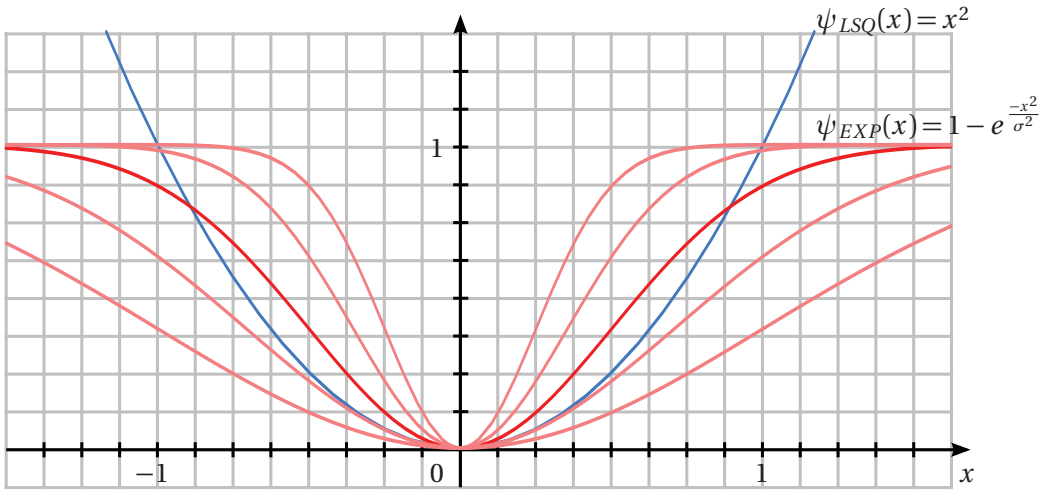


Figure 5.5: The least squares approach minimizes the sum of squared distances, $\psi_{LSQ}(x) = x^2$, which weights outlying points (blue). The function $\psi_{EXP}(x) = 1 - e^{-x^2/\sigma^2}$ used by the decreasing exponential approach (red) reduces this effect significantly. The choice of σ usually depends on the noise level of the input data.

5. The codomain of ψ_{EXP} is limited to the interval [0,1). Due to the upper limitation $\psi_{EXP} < 1$, outlying points do not disturb the overall fitting. Outlying points have only limited influence (illustrated in Figure 5.5). But in contrast to segmentation and clustering techniques all points *have* influence and regard is paid to them.

6. While the idea behind least squares is based on regression analysis, the background of the decreasing exponential approach is related to combinatorial analysis. The combinatorial process to find a model $M(\mathbf{x}) = \{M_1(\mathbf{x}), \ldots, M_m(\mathbf{x})\}$ containing the maximum number of points on a sampled surface $\mathbf{P}$ can be formulated using the Kronecker delta:

$$\delta_{M_i(\mathbf{x}),\mathbf{P}} = \begin{cases} 0, & M_i(\mathbf{x}) \notin \mathbf{P}, \\ 1, & M_i(\mathbf{x}) \in \mathbf{P}. \end{cases} \tag{5.9}$$

The formulation of the discrete, combinatorial problem as a continuous maximization problem with the possibility of handling feasible noise leads to a weighting function $\psi_{COM}(x) = e^{-x^2/\sigma^2}$. Having transformed the maximization problem into a minimization one $\psi_{COM}(x) = 1 - \psi_{EXP}(x)$ results in decreasing exponential fitting which maximizes the number of points on a model's surface.

5.4.3 Parameter Estimation

A simple plane-fitting example demonstrates the different approaches. The nonlinear least squares fitting technique to fit a plane

$$\mathbf{E}: Ax + By + Cz + D = 0 \tag{5.10}$$

to some points $P_1, \ldots, P_n$ with $P_i = \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^{\mathrm{T}} \in \mathbb{R}^3$ uses the unsigned Euclidean distance

$$d(P_i, \mathbf{E}) = \left| \frac{A \cdot x_i + B \cdot y_i + C \cdot z_i + D}{\sqrt{A^2 + B^2 + C^2}} \right|. \tag{5.11}$$

The function to minimize consists of the squared distances

$$f_{LSQ}(\mathbf{E}) = \sum_{i=1}^{n} d^2(P_i, \mathbf{E}) \tag{5.12}$$

$$= \sum_{i=1}^{n} \frac{(A \cdot x_i + B \cdot y_i + C \cdot z_i + D)^2}{(A^2 + B^2 + C^2)} \tag{5.13}$$

$$\stackrel{!}{=} \min_{A,B,C,D} \tag{5.14}$$

and takes the four model parameters with three degrees of freedom (the plane equation can be normalized). If the point set **P** consists of samples from the surface of a regular cube, the resulting plane will not contain a side of the cube but its space diagonal.

An appropriate RANSAC-algorithm will return a plane containing one side of the cube. It can handle multiple data sets and outliers very well. A basic prerequisite needed by any RANSAC approach is a solution for the inverse model description problem. A model's parameters have to be determined by a fixed number of points which define a unique model instance. This fixed number should be the minimum number of sample points needed to identify a unique solution [HZ04]. For a plane with parameters A, B, C, and D three sample points are sufficient, but this inverse problem becomes difficult or even unsolvable as soon as the model's complexity increases; for example, to fit a cylinder more points are needed and the calculation to get its position and radius is non-trivial. An overview of "Direct solutions for computing cylinders from minimal sets of 3D points" has been presented by CHRISTIAN BEDER and WOLFGANG FÖRSTNER [BF06].

In this context, the major advantage of the RANSAC approach is its ability to ignore outliers without explicit handling. This advantage without the need to solve the inverse parameter problem also applies to the decreasing exponential fitting approach. Its objective function uses ψ_{EXP} and regarding the used error functions and metrics involved the decreasing exponential method is similar to least squares techniques, but it belongs to the category of subpart fitting algorithms without preceding segmentation. It is able to find the best fit of an arbitrary subpart within a point cloud.

5.5 Implementation

The scanned data set is a point cloud $\mathbf{P} = \{P_1, \ldots, P_n\}$ and the generative model is a function $M(\mathbf{x})$, $\mathbf{x} \in G \subset \mathbb{R}^k$. The objective function of our algorithm minimizes

$$f(x) = \psi(d(\mathbf{P}, M(\mathbf{x}))) \stackrel{!}{=} \min. \tag{5.15}$$

5.5.1 Hierarchical Shape Description

In order to reduce the number of dimensions which are optimized at once, the shape template uses a hierarchy of model descriptions. Each level within this hierarchy reuses the parameters already defined in previous levels and refines the generative model. In this way the optimization problem is split up in several smaller problems. This hierarchical process is illustrated in Figure 5.6.

In each level of a hierarchy, the fitting process takes

1. the initial point cloud $\mathbf{P} = \{P_1, \ldots, P_n\}$,
2. a generative description in form of a script;
   e.g. `function_XY_shape(v_1, ..., v_k)`
3. and a corresponding validator;
   e.g. `function_XY_check(v_1, ..., v_k, ..., distances, ...)`

The point cloud and the scripted generator are the main components of the objective function to minimize. The result of this numerical optimization is a set of parameters $(v_1, \ldots, v_k)$. Besides numerical problems (local minima, divergence, etc.), this parameter set creates the "most similar" object to $\mathbf{P}$, if passed to the generator function.

The optimization routine always returns a parameter set, but the generative description and the point cloud may have nothing in common and do not have to describe the same shape. In this case, the distance values (one-sided distances, Hausdorff distance, ...) will have high values. The function of the validator script is to interpret these values.

It checks the optimization results and returns

- *reject*, if the distances are above a threshold (so that a continued fitting process does not make any sense), and if the intermediate result, including all fitting results of previous levels, are not acceptable.
- *stop*, if the distances are above a threshold, and if the intermediate result is acceptable.
- *continued*, if the distances pass the threshold. In this case, the validator also returns the next generative descriptor/validator pair to continue with.

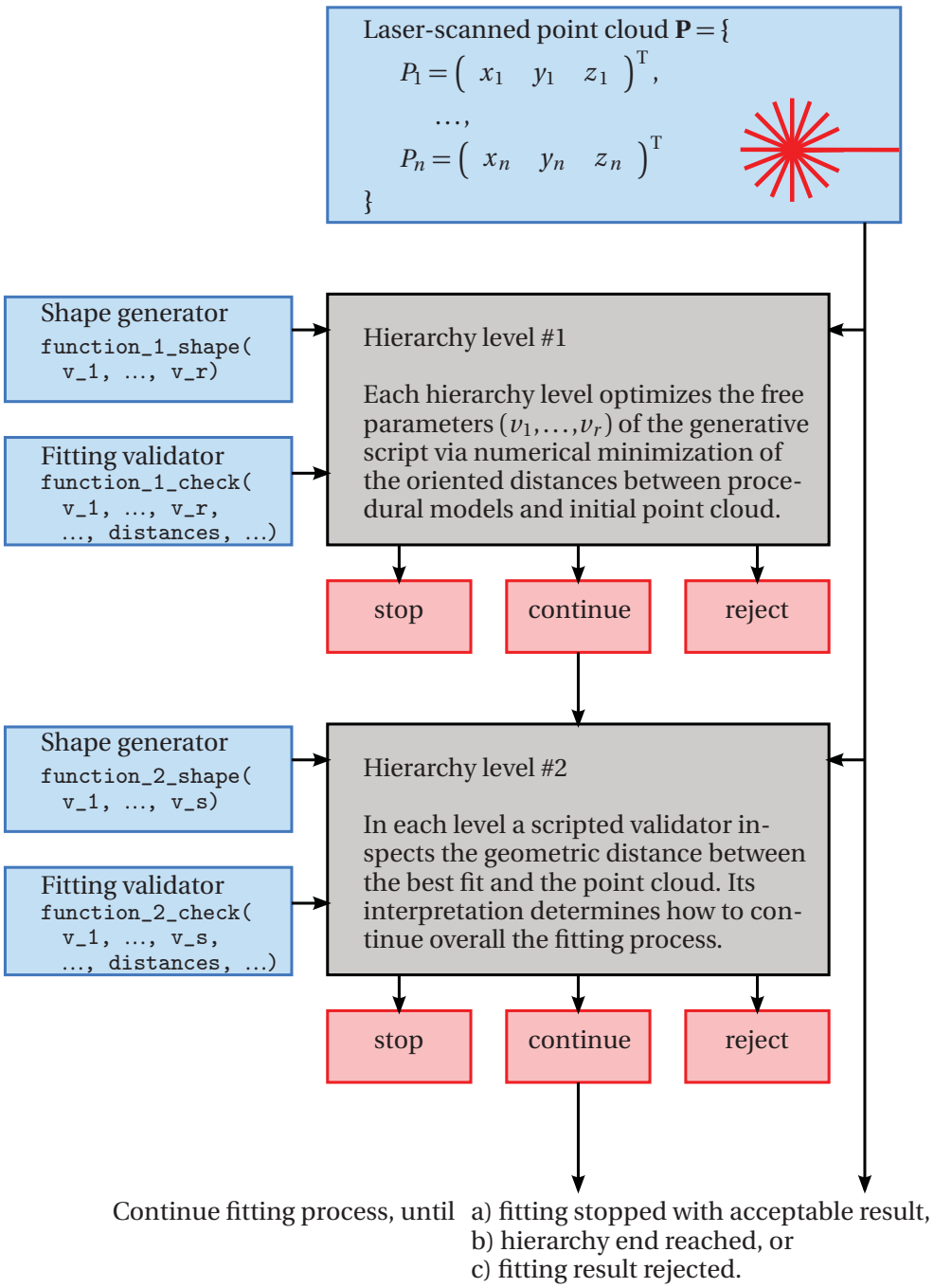


Figure 5.6: The fitting process uses a hierarchical shape description in order to reduce the number of free parameters during one optimization task (illustrated in gray). The input data (blue) does not only consist of one generative model, but of a collection of scripted subshapes and validators. The function of the validators (results in red) is to control the successive refinements and the overall fitting process.

The collection of all validators describe the overall fitting process. They implicitly comprehend a state machine respectively a directed graph. In most cases, it is organized as a tree or linear graph. From this point of view, the difference between *rejected*and *stopped* is simply a flag. In both cases the process has come to an end, but in one case the current state is marked to be an acceptable final state.

5.5.2 Fuzzy Geometry

A hierarchy of generative models contains model descriptions at different levels of resolution. Especially at early stages within the hierarchy it may not be sensible to describe geometry precisely – due to missing parameters. This problem can be solved by introducing fuzzy geometry.

Fuzzy geometry is a point cloud, in which each point is extended by a probability value σ. This value defines a normal distribution in 3D. A fuzzy geometry model consists of all overlapping normal distributions; i.e. a blurred point cloud as illustrated in Figure 5.7. This technique allows a modeler to describe diffuse geometry. It is used for all hierarchy levels except those, which may stop the fitting process with an acceptable solution. These levels produce precise geometric models based on meshes.

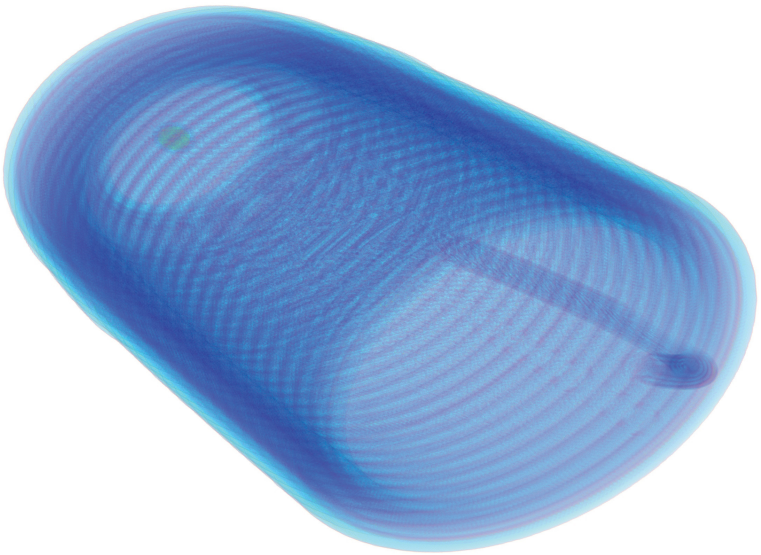


Figure 5.7: Geometry at a low resolution can be described by a fuzzy point cloud; i.e. a point cloud in which each point is smeared in space. This illustration shows such a fuzzy point cloud. Each point is drawn as a spherical volume whose radius and transparency correspond to its probability.

5.5.3 Inverse Geometry

The fuzzy models can be interpreted as an energy field – each point is an energy source whose power is quantified by σ. According to this interpretation the objective function "tries" to place the generative model, so that the scan is in a high energy area. This attraction effect is inverted by a new geometry description: inverse geometry.

Inverse geometry is a fuzzy point cloud with negative values of σ and with a modified weighting function. It uses the weighting function $1-\psi(x)$ (see Figure 5.8). As a consequence, negative points "try" to maximize the distance to the scan.

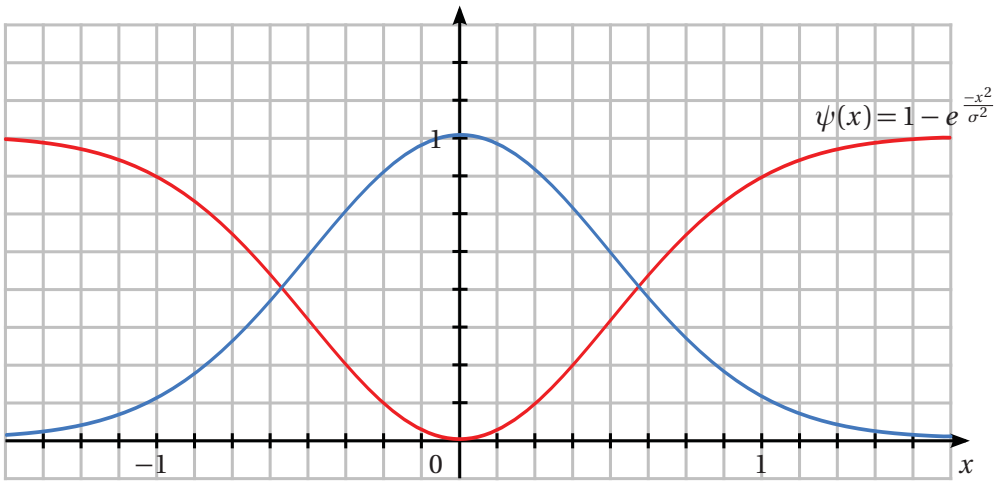


Figure 5.8: The weighting function $\psi(x)$ (plotted in red) reduces the disproportionate effect of outlying points. Furthermore, its bounded codomain $0 \leq \psi(x) \leq 1$ can be used to introduce distance-based "penalty" terms $1-\psi(x)$ (plotted in blue) in the objective function. These terms are the key idea of inverse geometry.

5.5.4 Optimization

The previous text describes the objective function f that is passed to the numerical optimization respectively minimization. The mathematical framework is summarized in Chapter 2.3.

The function f is evaluated by the numerical optimization routine, which consists of two parts. The first part uses a statistical optimization routine called "Differential Evolution" [SP97]. This algorithm is used to find a "good" starting point for the second optimization part – a conjugated gradients optimization according to Fletcher-Reeves [GMW82, GJH95, Fle00].

The objective function

$$f(x_1,\ldots,x_k) = \psi(d(\mathbf{P}, M(x_1,\ldots,x_k))), \qquad x_i \in \mathbb{R} \tag{5.16}$$

is evaluated algorithmically; i.e. $M(x_1,\ldots,x_k)$ is a method call to an a-priori unknown script, which returns 3D geometry. In order to tap the full potential of numerical optimization, the minimization routines need f as well as its partial derivatives $\frac{\partial f}{\partial x_i}$ [Gou09]. This problem is solved by a differentiating compiler. The *Euclides* metamodeler parses JavaScript and translates it to various platforms for different purposes [USF10b], [SSUF10b], [SSUF10a]. The result is differentiated Java code, which can be included in objective functions. The complete generative model description $M(x_1,\ldots,x_k)$ (including all possibly called subroutines) is differentiated with respect to the input parameters. This differentiating compiler offers the possibility to use gradient-based optimization routines in the first place. Without partial derivatives many numerical optimization routines cannot be used at all or in a limited way.

Having the objective function $f(x_1,\ldots,x_k)$ as well as its partial derivatives $\frac{\partial f}{\partial x_i}$ at hand, it is possible to use standard optimization algorithms to solve the minimization problem efficiently. The differentials are implemented using automatic differentiation techniques (see sidebar on page 52f). Furthermore, compiled scripts can be executed much faster than purely interpreted scripts [UKF08].

5.5.5 Distance Calculation

The distance computation is generally by far the most time-consuming part of the algorithm. Due to the fact that the weighting function ψ has an upper limit

$$\forall\, x \in \mathbb{R} : \psi(x) < 1 \tag{5.17}$$

and converges relatively fast to one (see Figure 5.8), the objective function can be evaluated very efficiently. The weighted distances of points "far away" (depending on a threshold calculated using σ) can be approximated by $\psi \approx 1$ and do not have to be calculated exactly. Bounding volumes, partitioning of space, and hashed grid structures speed up the evaluation of f. Only small distances are evaluated exactly as illustrated in Figure 5.9. Further details on optimal distance calculations can be found in [USK+07].

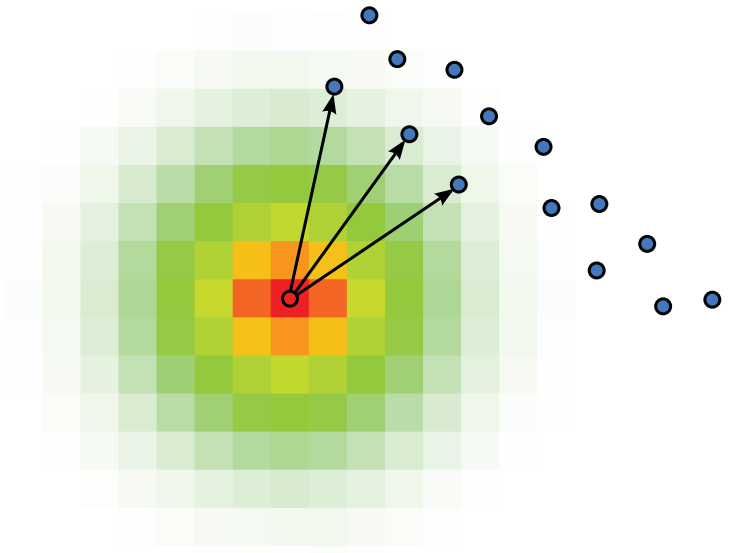


Figure 5.9: The storage of a point cloud in a regular grid structure of equally sized cubes allows a fast evaluation of a sum of ψ_{EXP}-weighted distances. For a given model only those cubes whose distance to the model is within a threshold may contain points with a weighted distance significantly smaller than one. These points have to be evaluated separately, the weighted distance of all other points – outside the colored area – can be approximated by the value 1.

5.5.6 Linear Algorithms in Sublinear Time

Another technique to speed up the optimization routine uses estimation techniques based on probability theory and statistics (see Chapter 2.2).

The sidebar "Linear Algorithms in Sublinear Time" on page 48 introduces an example, in which the hypothesis test of a RANSAC algorithm is not evaluated exactly but estimated. The optimization processes can be sped up the same way.

In this context, the key requirement to use statistical estimation is a result whose size is constant. Moreover, if the input data size is very large, such as a laser scan, then the setting is a perfect candidate for estimation techniques. Therefore, the distance function will not be evaluated completely, but approximated with sufficient accuracy. The approximation will only use a subset of points. This technique also applies to evaluations of derivatives $\frac{\partial f}{\partial x_i}$ of f.

As many optimization routines have additional subroutines to handle local minima, noisy data, and ill-conditioned problems, the number of random samples to ensure a fixed confidence level cannot be determined easily. A rule of thumb suggests for input data of size n the use of $\sqrt{n}$ samples. As estimation and exact evaluation can be exchanged easily the implementation uses an adaptive solution. The iteration starts with an estimation based on $\sqrt{n}$ random samples. A couple of iterations in the beginning and a few iterations in between use not only an estimate but also perform an exact evaluation using the complete input data set. Depending on the difference of estimation and evaluation a feedback-loop adjusts the number of random samples.

The following example applications comprehend a laser scan of the Pisa Cathedral. One data set consists of 4 449 908 points, whereas the average number of points used in its estimation is ≈ 45 000. As the error thresholds and exit condition of the optimization routine have not been modified, the optimization results do not differ significantly, whereas the minimization routine speeds-up by a factor of ≈ 90.

From the information theoretical point of view, our practical results coincide with theory. BRADFORD R. CRAIN has shown the interrelation between the concept of information according to CLAUDE ELWOOD SHANNON[1] [Mac03] and statistics [Cra77]. The correlation between entropy and the central limit theorem has been pointed out by ANDREW R. BARRON [Bar86]. The connection to approximation and estimation theory has been studied by ZHI ZONG who presents an overview in *Information-Theoretic Methods for Estimating of Complicated Probability Distributions* [Zon06]. The technique to apply statistics to algorithmic or combinatorial problems has produced interesting results in the latest research findings [BKR04], [MB07]. An overview can be found in the proceedings of the "Sublinear Algorithms" workshop [CMRS08]. An introduction with a focus on computer graphics named "Linear Algorithms in Sublinear Time–a Tutorial on Statistical Estimation" has been written by TORSTEN ULLRICH and DIETER W. FELLNER [UF11b].

5.6 Applications

In order to proof the algorithmic concept, three fields of applications are inspected in the text below. The data sets used in these applications have been created exclusively for this purpose or have been acquired in the context of information technology for cultural heritage.

In contrast to many fields of applications the context of cultural heritage distinguishes itself by model complexity, model size, and imperfection to such an extent most approaches cannot handle.

- Complexity: cultural heritage artifacts "represent a masterpiece of human creative genius"[2]. Hence, many cultural heritage artifacts have a high inherent complexity.
- Size: as of January 2011 the United Nations Educational, Scientific and Cultural Organization lists 911 cultural sites in over 150 states. They are part of the cultural and natural heritage which the World Heritage Committee considers as having outstanding universal value.
  Furthermore, an archaeological excavation may have an extent on the scale of kilometers with a richness of detail on the scale of millimeters.

[1] CLAUDE ELWOOD SHANNON (April 30, 1916 – February 24, 2001) Claude Elwood Shannon was an American mathematician and electronic engineer. He is famous for having founded information theory with his article "A Mathematical Theory of Communication" published in 1948.

[2] The Criteria for selection to be included on the United Nations Educational, Scientific and Cultural Organization (UNESCO) World Heritage List: http://whc.unesco.org/en/criteria

- Imperfection: cultural heritage artifacts are embedded into a context. Beside natural wear and tear effects, many artifacts exhibit signs of preservation, restoration, and refurbishment.

These additional constraints have to be regarded by reconstruction algorithms in the context of cultural heritage.

5.6.1 Selfsimilarity

From the mathematical point of view all input parameters of a generative model are equal. In practice, they can be divided in two groups:

1. Parameters, which describe an object's position and orientation, and
2. Parameters, which describe object attributes (e.g. width, height, etc.)

The first example concentrates on position and orientation parameters only. It is a "degenerative" model, which consists of static geometry and an isometric transformation; i.e. a transformation, which only consists of a rotation and a translation. Consequently, it has six free parameters.

Figure 5.10: The Pisa Cathedral (Duomo di Pisa) is a masterpiece of Romanesque architecture. Despite its proximity to the eye-catching and tourist-attracting Leaning Tower, the Duomo still dominates the monumental Piazza dei Miracoli in Pisa.
This photo has been taken by GEORGES JANSOONE.

Figure 5.11: One column has been extracted from the apse of the Pisa Cathedral (upper left). In combination with an isometric mapping with six degrees of freedom (translation and rotation) it forms a "degenerative" model. The fitting algorithm minimizes the objective function, which defines a measure of similarity; i.e. it identifies "copies" of the reference. Each "copy" is rendered in a different color and blended with the input data set.

The example data set is a laser scan of the Pisa Cathedral. It has been generated by the Visual Computing Laboratory at the Institute of Information Science and Technologies (ISTI) of the Italian National Research Council (CNR). The Duomo is located on the Campo dei Miracoli in the center of Pisa, Italy. The architect BUSCHETO DI GIOVANNI GIUDICE begun his masterpiece in 1064 and started the characteristic Pisan Romanesque style in architecture. Just like the whole building (see Figure 5.10), the apse of the Duomo consists of many similar columns which are arranged in arcs and rows.

In this scenario a single column has been extracted manually. It serves as a geometric reference model. Being a part of a "degenerative" model, which only consists of an isometric map applied to the geometric reference, the fitting algorithm optimizes the translational and rotational part of the isometric transformation; i.e. it searches for similar geometry. The similarity is measured by the objective function to minimize. As a consequence, the algorithm identifies "copies" of the reference. Figure 5.11 shows the reference geometry (upper left) and all identified, similar objects – each one rendered in a different color and blended with the input data set.

5.6.2 Parameter Estimation

In the next example a shape template with free parameters on object attributes is used. It describes an arcade that is arranged in an arc and takes nine parameters. Its horizontal and upright projection is shown in Figure 5.12. The shape template has nine free parameters: the center point x, y, z, the main radius R, the column radius r, the offset angle α, the opening angle β, the number of columns n and the column height h.

The fitting process has been started to detect the two arcades in the data set. Due to the low level of noise within the data set, σ^2 has been set to a rather small value. The results are positive and the algorithm has identified the two arcades. The corresponding parameters have been determined and the reconstruction is visualized in Figure 5.13.

As the Pisa Cathedral and the Leaning Tower of Pisa share the same soft soil, the cathedral's eastside already sank during construction. Corrective building measures and civil works started even before the building has been finished. These changes infringe its regular structure. Consequently, the arc's boundary columns, which are distributed absolutely regular, have higher error/distance values than the inner columns. This effect can be seen in the second arcade (Figure 5.13, upper row). This effect demonstrates an important property of the algorithm: it can only determine parameters – it does not modify the generative description. The regular structure of a script is fixed. Variations and deviations are only possible, if appropriate parameters have been made accessible. Especially, semantic errors its construction process will not be corrected.

Furthermore, this example demonstrates the negative effects of over-fitting, if inverse geometry, as described in Section 5.5.3, is not used. Without inverse geometry, the parameter intervals need to be estimated roughly and the number of columns n needs to be fixed. The range δ of each parameter has been:

$$\begin{array}{rcr|rcr} \Delta x & = & 21.0m & \Delta R & = & 1.5m \\ \Delta y & = & 21.0m & \Delta r & = & 0.2m \\ \Delta z & = & 21.0m & \Delta h & = & 2.0m \\ \Delta \alpha & = & 14^\circ & \Delta n & = & 0 \\ \Delta \beta & = & 14^\circ & & & \end{array}$$

As the generative description does not check reasonability, a high number n of columns would be a good solution. It would generate a "wall" of columns. This arched "wall" could be placed anywhere within the apse, and the result would lead to a small error, respectively to small distances. This undesirable effect can be avoided by fixed parameters and domains or by inverse geometry. In this way, the arc's property that it spans a space can be included the generative description: the space is just filled with inverse geometry.

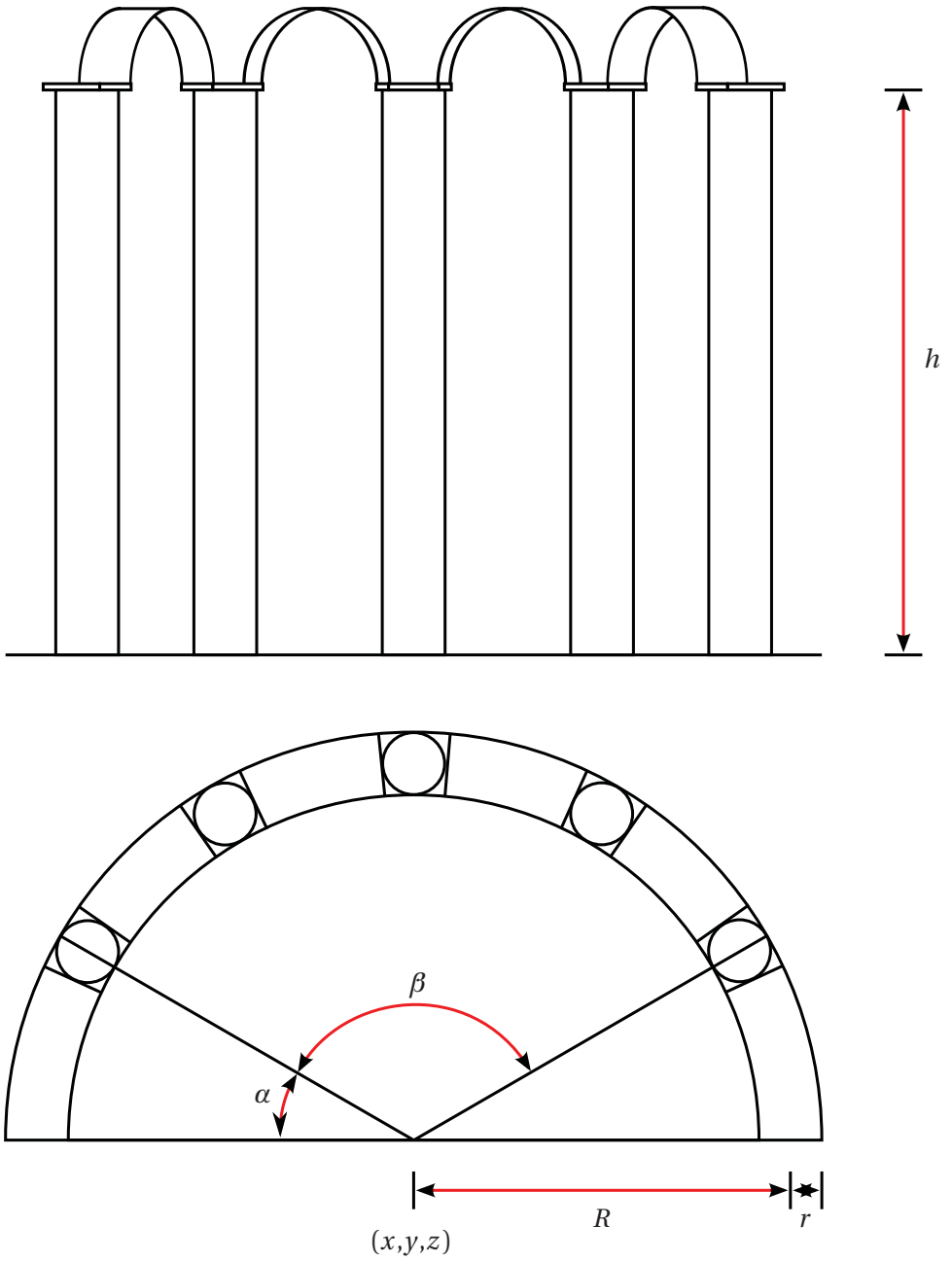


Figure 5.12: The parametric description of the circle-arcades model takes nine parameters: the three coordinates of a center point (x,y,z), a main radius R, a column radius r, an offset angle α, an opening angle β, and the number of columns n. These values define the ground construction in the xy plane which contains the center point. The last parameter h defines the columns' height. The height of the Roman arcs are determined by the column distances.

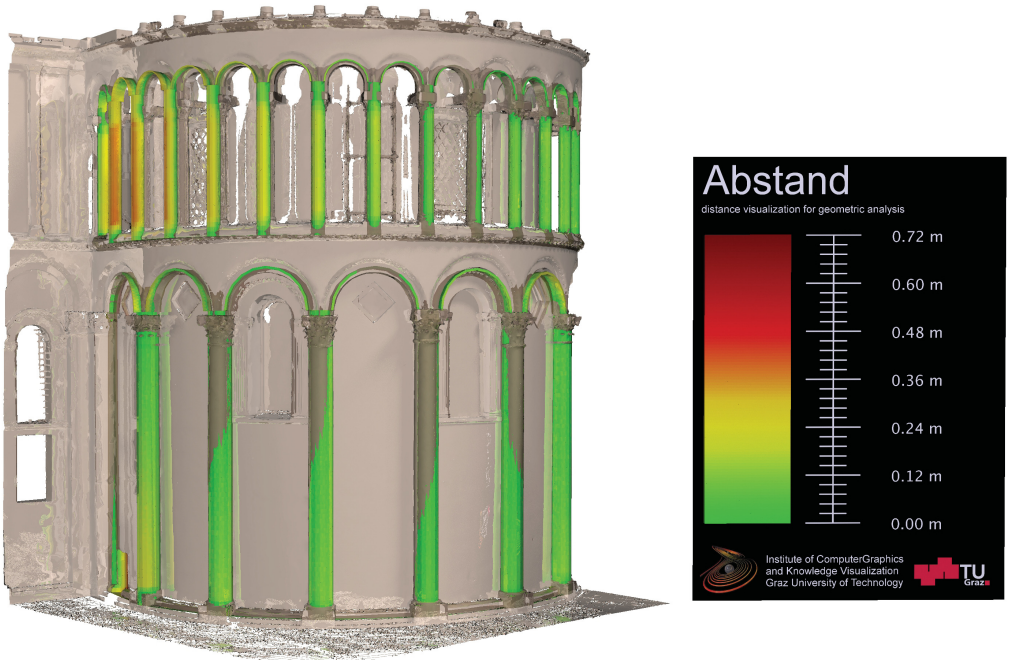


Figure 5.13: A generative model with nine parameters is fitted to the laser scan of the Duomo of Pisa. The algorithm detects two instances of the shape template and determines the parameters which describe the given geometry best. These characteristics (column height, etc.) are high-level parameters and valuable information which are needed in the context of digital libraries in order to index a model repository.

The second data set is a photogrammetric reconstruction of an inner courtyard. It is located in Graz, Austria, and belongs to the Landhaus. This Renaissance building has been constructed in 1557 by DOMENICO DELL'ALLIO[3].

Due to the high level of noise within the photogrammetric reconstruction, this example is a robustness test for the fuzzy geometry concept. This geometry representation is described in Section 5.5.2.

The generative model to fit describes an arcade. It consists of columns with a quadratic profile. Its parameters are

- the center point (x,y,z) of the first column,
- the angle α to define the arcade's orientation,
- the column width w,

[3] DOMENICO DELL'ALLIO (1505 – 1563) Domenico dell'Allio (1505–1563) was an Italian architect who has lived and worked in Styria, Austria since 1530.

- the column height h,
- the distance d between two columns, and
- the number of the columns n.

Figure 5.14 illustrates an instance of the procedural description in horizontal projection.

The quality of the input data set is rather poor. The point cloud shown in Figure 5.15 has been reconstructed with only 4 sequences of photos. As a consequence, the result has a very high level of noise. The fitting process started with $\sigma^2 = 0.025\frac{1}{m^2}$; i.e. points up to $\pm 0.13m$ away from the surface are still regarded as part of the surface. Nevertheless, the algorithm is able to identify two instances and to approximate its high-level parameters. As the input data set can hardly be considered to be a "ground truth", the results in Figure 5.15 are not distance-based color-coded.

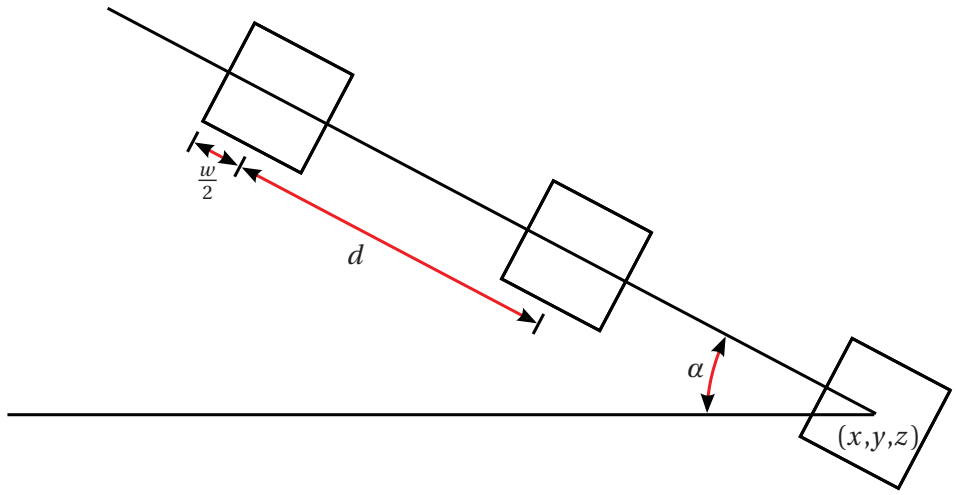


Figure 5.14: The parametric description of the row-arcades model takes eight parameters: the three coordinates of the starting point (x,y,z), an offset angle α to define the arcade's orientation, a column's width w and height h, the distance d between two columns, and the number of columns n.

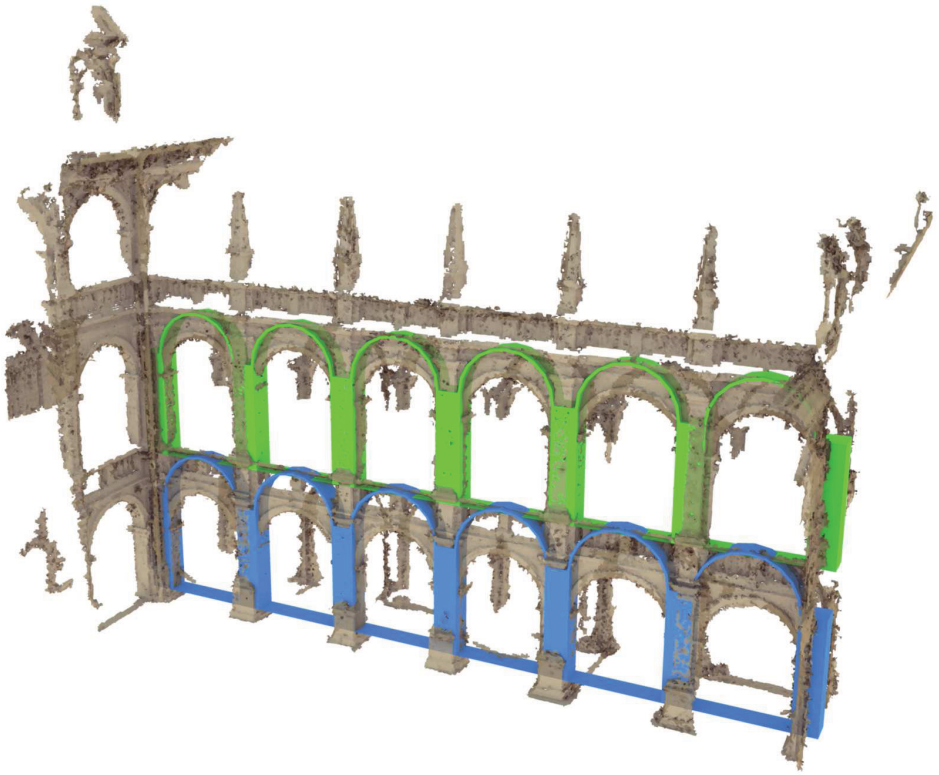


Figure 5.15: The inner courtyard of the Landhaus in Graz consists of several arcades. Its point cloud model has a very high level of noise. A noisy data set requires a tolerant fitting configuration, respectively a tolerant geometry representation such as fuzzy geometry. The concept of fuzzy geometry is described in Section 5.5.2. In this visualization not a fuzzy point cloud, but an isosurface of a fixed probability is rendered.

5.6.3 Shape Recognition

Generative modeling techniques take advantage of regularities and uniformities. Especially, frequent repetitions benefit from procedural approaches due to its reusability. In the context of shape recognition, it is obvious that a generative script cannot only identify a single object but a whole family of objects. The following example demonstrates this capability. It consists of twelve laser-scanned cups and scripted a generative model of a cup, which takes 15 input parameters: six parameters describe its position and orientation, nine parameters describe its attributes (radius, height, ...). The generative model is sketched in Figure 5.16 whereas images of the cups and their scans are visualized in Figure 5.17 and Figure 5.18 respectively.

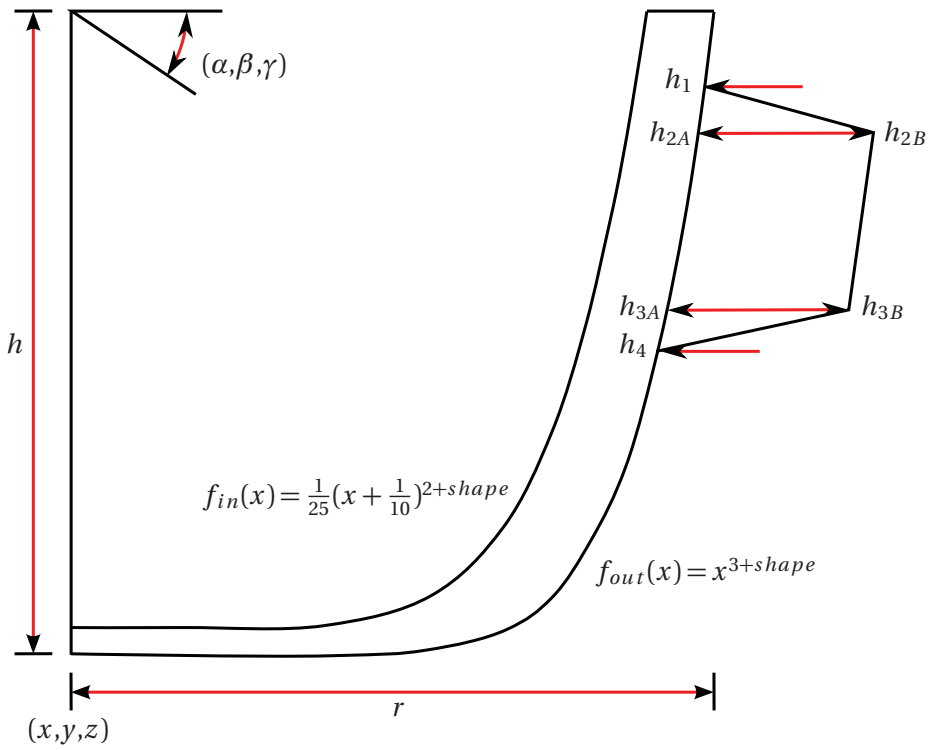


Figure 5.16: The generative cup model takes 15 parameters: (x,y,z) is the base point of the cup and (α,β,γ) define its orientation. Its shape is defined by an inner f_{in} and outer f_{out} shape function with one free parameter $shape$. These functions are rotated around the cup's main axis and scaled with the parameters r and h.
The handle is defined via six parameters, which form points in 2D (the plane of the handle); namely $(h_1, f_{out}(h1))$, (h_{2A}, h_{2B}), (h_{3A}, h_{3B}), and $(h_4, f_{out}(h_4))$. They are the control points of a Bézier curve. Its tube with a fixed diameter (10mm) defines the cup's handle.

Figure 5.17: The example data set consists of twelve cups made of porcelain. Each one has been scanned using a laser scanner. As the cups have clean and shiny surfaces, they are difficult to scan. The scan results are shown in Figure 5.18.

Figure 5.18: Each scanned cup comprehends between 15 573 (small espresso cup) and 130 973 triangles (big mug). The scan results are noisy and not-cleaned-up meshes with many holes. In this illustration the surfaces are rendered semi-transparent, so that incomplete parts (with missing inner and / or outer surface) appear brighter than complete parts.
Please note, each subimage is scaled to a common bounding box, which does not reflect the scan's real size.

The 15 parameters of the generative model are determined using three hierarchical levels.

1. The first level determines the cup's base point (x,y,z) and its orientation (α,β), as well as its height h, radius r, and $shape$. At this level the cup is rotationally symmetric; therefore, position and orientation only need five instead of six parameters. The $shape$ parameter is used in two functions f_{in}, f_{out} which define the cup's inner and outer shape (see Figure 5.16).

2. Afterwards, the algorithm determines the parameter γ – the rotational position of the handle.

3. At the last level, the parameters h_1, h_{2A}, h_{2B}, h_{3A}, h_{3B}, h_4 are determined. These parameters define four points of a Bézier curve. As h_1 and h_4 are start and end point of the handle, they are located at $(h_1, f_{out}(h1))$ resp. $(h_4, f_{out}(h_4))$, whereas the second (h_{2A},h_{2B}) and third point (h_{3A},h_{3B}) may float freely within the plane with orientation γ. The resulting Bézier curve is expanded to a 3D tube with a fixed diameter of 10mm.

This generative model M is able to describe the scanned cups and its parameters can be determined automatically by the proposed algorithm.

The fitting results are visualized in Figure 5.19. In 11 of 12 cases (92%) the algorithm is able to detect an instance of the generative cup M. In these cases the cups' properties (position, orientation, radius, height, handle shape) are determined with only a small error.

In one case (cup #5) the global optimization routine is stuck in a local minimum. Despite the local minimum, the error values is too high, so that the algorithm rejects the hypothesis of a generative cup. For illustration purposes Figure 5.19 (second row, middle) shows the last best-fit result of the optimization which would have been returned, if the algorithm had not stopped and rejected the fitting process.

In the other cases, the fitting process has been successful. Especially, the cups #1 and #12 have been fitted, although their shapes (the scan is rendered semi-transparent in gray) are not rotationally symmetric. Cup #1 (Figure 5.19, top row, left) has quadratic footprint with beveled edges; cup #12 (Figure 5.19, bottom row, right) has an octagonal footprint. In both cases the generative cup is able to describe them within a tolerable rate of variance.

Also cup #9 (Figure 5.19, next to last row, right) has been detected to be a cup, although its shape looks like a busted paper cup. Its error value passed the threshold of rejection at the first level, but it did not pass the following ones. Therefore, the algorithm perfectly identifies a cup without a handle.

The distance values of all cups are listed in Table 5.1. During the fitting process mainly one-sided distances (from the generative model to the scan) and Hausdorff distances are used. While the Hausdorff distance gives an "overall impression", the one-sided distances can be used to measure local fits; e.g. if each point on the generated handle is (in average) half its diameter away from the scan, then the cup will most probably not have a handle.

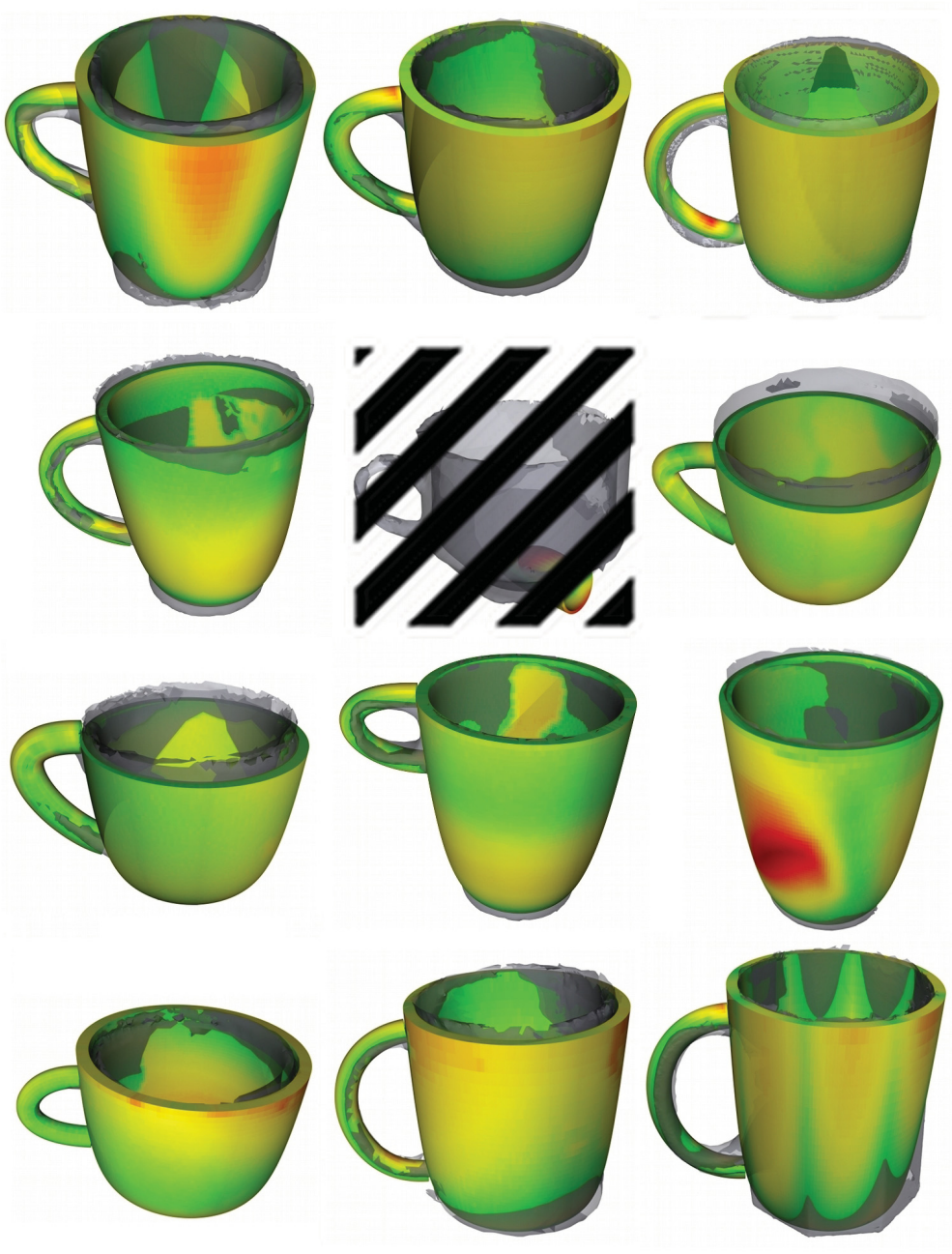


Figure 5.19: The laser-scanned cup (rendered in semi-transparent gray) have been identified as instances of the generative cup description in 11 of 12 cases. In these cases the cups' properties (position, orientation, radius, height, handle shape) are determined successfully. The detailed distance measurements are listed in Table 5.1.
Please note, each subimage is scaled to a common bounding box, which does not reflect the scan's real size.

Semantic Recognition

Table of distance measurements of the “cups” data set.

	one-sided distance average	one-sided distance maximum	one-sided distance std. deviation	Hausdorff distance
scan #1	3.37228 mm	8.04885 mm	1.99973 mm	9.24657 mm
scan #2	3.16246 mm	6.74649 mm	1.86304 mm	6.80079 mm
scan #3	4.70168 mm	8.87795 mm	2.47398 mm	10.0124 mm
scan #4	2.35580 mm	8.22992 mm	1.51475 mm	8.22992 mm
scan #5	–	–	–	68.2669 mm
scan #6	2.09885 mm	5.48440 mm	1.16270 mm	14.9031 mm
scan #7	1.75171 mm	5.55879 mm	0.92017 mm	7.46554 mm
scan #8	2.23997 mm	6.32716 mm	1.34969 mm	6.32716 mm
scan #9	2.25705 mm	10.7793 mm	1.78702 mm	10.7793 mm
scan #10	2.25851 mm	6.98503 mm	1.31063 mm	6.98503 mm
scan #11	3.91988 mm	7.81589 mm	2.13471 mm	11.4433 mm
scan #12	2.96143 mm	8.68567 mm	1.98203 mm	8.68567 mm

Table 5.1: The detailed measurements of the fitted, generated cups. The scan numbers correspond to the visualizations shown in Figure 5.17 and Figure 5.18 respectively. As cup #5 has been rejected, it does not have sensible distance values. In all other cases, the distance values have been calculated between the scan and its best-fit generative description.

This chapter presents a shape description approach based on generative modeling techniques and an algorithm, which is able to identify instances of a generative description in real-world data sets. This algorithm demonstrates the proof of concept.

The main contributions and benefits of this approach are an implementation of a generative shape description including the inverse recognition and indexing problem. Based on a generative description, the algorithm is able to identify instances of a shape template and it can determine its calling parameters.

To implement this concept, a hierarchical model description with fuzzy geometry to represent “unknown” parts of a model (parts which are fitted at lower levels within the hierarchy) is used. Furthermore, the concept of inverse geometry is implemented. It offers the possibility to describe the absence of geometry; i.e. to formulate “missing” geometry (e.g. a window is a hole in a wall).

Including a generative compiler with automatic derivation, the optimization routine can evaluate both the objective function $f(x_1,\ldots,x_k)$ as well as its partial derivatives $\frac{\partial f}{\partial x_i}$. This key feature opens up new chances to use standard optimization algorithms to solve the inverse problem efficiently. Each fitting process (with one scan and one generative hierarchy) takes only a few minutes one a single PC to finish. These timings just give an impression of the algorithm's performance. Due to open problems (e.g. algorithm's failure in test case #5), it is currently not sensible to run benchmarks at great length. As long as this problem is not solved, detailed benchmarks would not be reasonable. If the problem was caused by a premature termination of the routine to avoid local minima, the algorithm's timings might change signifcantly.

6 Conclusion & Future Work

Within this thesis the thematic complex of *Reconstructive Geometry* has been addressed. This topic comprehends geometry, numerical optimization, probability theory, as well as algorithm design, software engineering, computer graphics, and computer-aided design.

This thesis is composed of numerous scientific articles and conference papers. Its main contributions have been made in the field of collision detection, generative modeling, and semantic recognition. These contributions and their corresponding benefits are summarized in this chapter. An outlook on further research and future perspectives concludes this thesis.

Contents

6.1 Collision Detection

In reconstructive geometry many tasks and algorithms are distance based; i.e. they evaluate a distance function. Therefore, space partitioning techniques are of special interest. The same techniques are used to speed up collision detection algorithms. They answer the question, whether two objects collide i.e. have distance zero. As a consequence, this thesis places particular emphasis on collision detection.

6.1.1 Contribution

In the article "Hierarchical Spherical Distance Fields for Collision Detection" [FUF06] by CHRISTOPH FÜNFZIG, TORSTEN ULLRICH and DIETER W. FELLNER, we present an approach, which combines wavelet-like techniques with max-plus algebra [ACG+90]. Applied to spherical coordinate systems the resulting distance field is perfectly qualified for terrain and model queries as well as line-of-sight computation.

Its implementation details and a comprehensive comparison can be found in "Empirical Comparison of Data Structures for Line-Of-Sight Computation" [FUFB07] and in "Terrain and Model Queries Using Scalar Representations With Wavelet Compression" [FUFB09].

6.1.2 Benefit

Due to the fact, that our hierarchical, spherical distance field for collision detection can be implemented hardware-optimized, it is an attractive collision detection solution for embedded systems. As it meets real-time requirements, it is the first choice for time-critical systems. EDWARD N. BACHELDER from Systems Technology, Inc.[1] adopted this idea and implemented such a real-time solution on embedded hardware.

Our collision detection algorithm has been analyzed in a comprehensive comparison. Every practitioner who is confronted with the task of distance calculation can benefit from our analyses [USK+07].

6.2 Generative Modeling

The distance calculation is one of the techniques used in the context of generative modeling. Furthermore, this thesis presents a semantic recognition and reconstruction approach based on shape templates and generative modeling techniques. Consequently, procedural modeling languages, language processing and translation, as well as compiler construction are an important issue of this thesis.

[1] http://www.systemstech.com/

6.2.1 Contribution

The main contribution on generative modeling techniques is the meta-modeler approach *Euclides*. This innovative meta-modeler concept allows a user to easily export generative models to other platforms without losing its main feature – the procedural paradigm. In contrast to other modelers, the source code does not need to be interpreted or unfolded, it is translated. Therefore it can still be a very compact representation of a complex model. The process of

parsing → validating → translating

JavaScript offers many advantages. A consistent intermediate representation in terms of an abstract syntax tree, serves as a basis for back-end exporters to different languages and different platforms for different purposes [SSUF10a]. Two platforms are outstanding: differentiated Java code and Generative Modeling Language (GML).

The compiler *Euclides* regards a shape template as a function $f(x_1,\ldots,x_k)$ and generates code to execute it. Furthermore, it differentiates each function – including all subroutines – with respect to its parameters. In this way, every inverse problem involving generative modeling techniques and shape templates can evaluate f as well as its partial derivatives $\frac{\partial f}{\partial x_i}$. This technique offers the possibility to use standard optimization algorithms to solve inverse problems efficiently.

While the translation to GML may sound like a simple infix-to-postfix transformation, the correct translation of control flow structures is a non-trivial task, due to the fact that there is no concept of `goto` in the PostScript language and its dialects. *Euclides* is the first complete translator to a PostScript dialect, which covers all control flow statements [SSUF10b].

6.2.2 Benefit

As *Euclides* offers a new access to GML, all GML users will benefit from its results. The possibility to use GML via a JS-to-GML translator reduces the inhibition threshold significantly. Everyone, who knows any imperative, procedural language (Pascal, Fortran, C, C++, Java, etc.) is familiar with the language concepts in JavaScript and therefore he can use *Euclides*.

In order to make generative modeling accessible to an audience of domain experts, e.g. cultural heritage professionals, which are seldom computer scientists, new access technologies to generative modeling are needed. Only these experts have the knowledge about the inner structure of an object. If this structure should be reflected in a generative description, these domain professionals are needed to tap the full potential of generative techniques. The generative modeling framework *Euclides* offers an easy-to-use scripting approach based on JavaScript.

6.3 Semantic Reconstruction

The generative modeling framework *Euclides* has been used to design shape templates and to describe shapes. The identification of shapes is an emerging problem, whose urgency increases with the number of digital 3D objects.

6.3.1 Contribution

This thesis presents a shape description approach based on generative modeling techniques and an algorithm, which is able to identify instances of a generative description in real-world data sets. The algorithm demonstrates the proof of concept.

This proof of concept comprehends geometric techniques such as inverse geometry, fuzzy geometry and generative, hierarchical model descriptions. Inverse geometry describes the absence of geometry. This concept offers the possibility to formulate "missing" geometry (e.g. a window is a hole in a wall); whereas fuzzy geometry represents "unknown" parts of a model. In a hierarchy of model descriptions the fuzzy parts are defined in one level and are refined or replaced in a higher level of detail.

The algorithm's implementation uses the translation and compilation techniques mentioned above. Furthermore, it realizes geometric optimizations (e.g. spatial coherence in data structures) and algorithmic optimizations to perform numerical minimization in acceptable time. The algorithmic optimization uses statistical estimation to speed up the evaluation of a cost function. This technique is published in "Linear Algorithms in Sublinear Time" [UF11b].

6.3.2 Benefit

The generative shape description and recognition approach presented in this thesis is a proof of concept. In combination with annotation techniques it is now possible to index 3D objects. Generative scripts will describe 3D data (stairs, windows, cups, etc.) and the presented algorithm processes them. It will perform a fitting process and evaluate the best-fit result. Having found an instance of a generative description, this offline indexing step can copy a human-readable annotation ("This is a cup.") into the object's markup. Each identified object just references the annotation of its best-fitted shape template. Then, queries can be formulated in textual form and the retrieval is based on simple text searching.

6.4 Future Work

While the proof of concept already works today, a scalable solution of an automatically indexed 3D data base [SMKF04] will meet new challenges that will be met in the future.

6.4.1 Generative Modeling

Beside the automatic indexing of 3D data bases, the interplay of generative modeling techniques and primitives-based modeling using triangles, meshes, etc. will offer new modeling possibilities. A simple example is a laser scan that is fitted to a generative shape template. The template describes the ideal geometry whereas the scan represents a real-world data set. If the difference – the geometric offset – is stored in a generative texture map respectively offset map, then it can be reused by other instances of the same generative model. Consequently, the laser-scanned mesh can be modified using the shape template's high-level parameters. First experiments with this new modeling approach are visualized in Figure 6.1.

6.4.2 Procedural Optimization

Currently, generative modeling is a design process guided by aesthetics. The parameters used to instantiate a shape template are often set manually by a human designer or they are set according to a human design – in a fitting process. However, it does not have to be that way.

The numerical optimization minimizes a cost function, which uses distances that perform a kind of similarity measurement. Though if the design were already completely determined by the generative modeling script, its free parameters could be optimized according to new objectives; e.g. energy consumption. Concerning the energy balance of a building several factors are important: area-to-volume ratio, insulation, window size, and many more (see Figure 6.2). These factors cause mutual interactions and need a trade-off. For example, insulation does not only keep energy inside, it also keeps energy outside. In this case the free parameters of a generative building can be optimized in order to find the best trade-off of all factors. If the objective function describes the energy needed, the optimization routine will minimize the amount of consumed energy. This is one important field of application; as optimization is a permanent issue in civil engineering, many more applications of generative optimization techniques are possible. The meta-modeler concept with its differentiating compiler is a cornerstone and offers the possibility to use standard optimization routines in the first place.

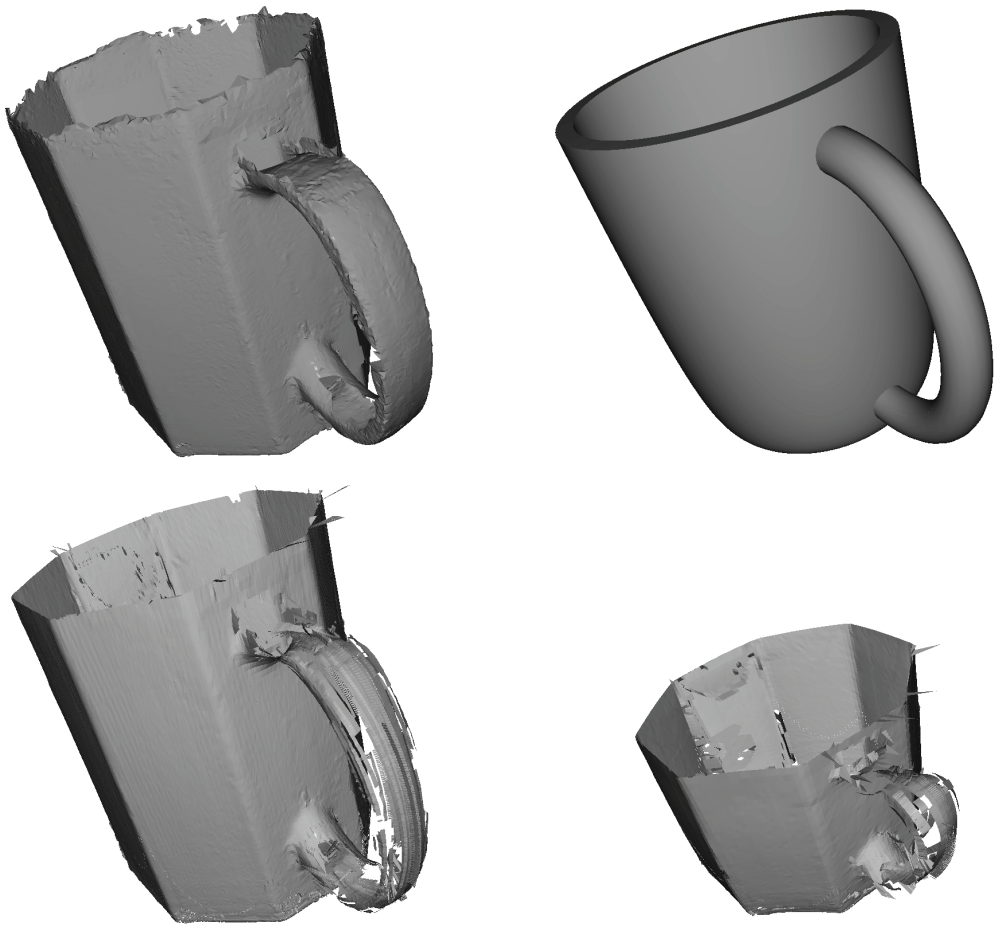


Figure 6.1: Using generative modeling techniques a shape template can be fitted to a laser scan (upper left). The fitting process determines the best-fit parameters of the procedural model, so that an instance of these parameters generates the most-similar object the shape template can produce (upper right). The geometric difference between both objects can be stored in an offset map, which can be applied to all instances of the shape template (lower left). As a consequence, the laser scan can be modified using the template's high-level parameters (lower right).

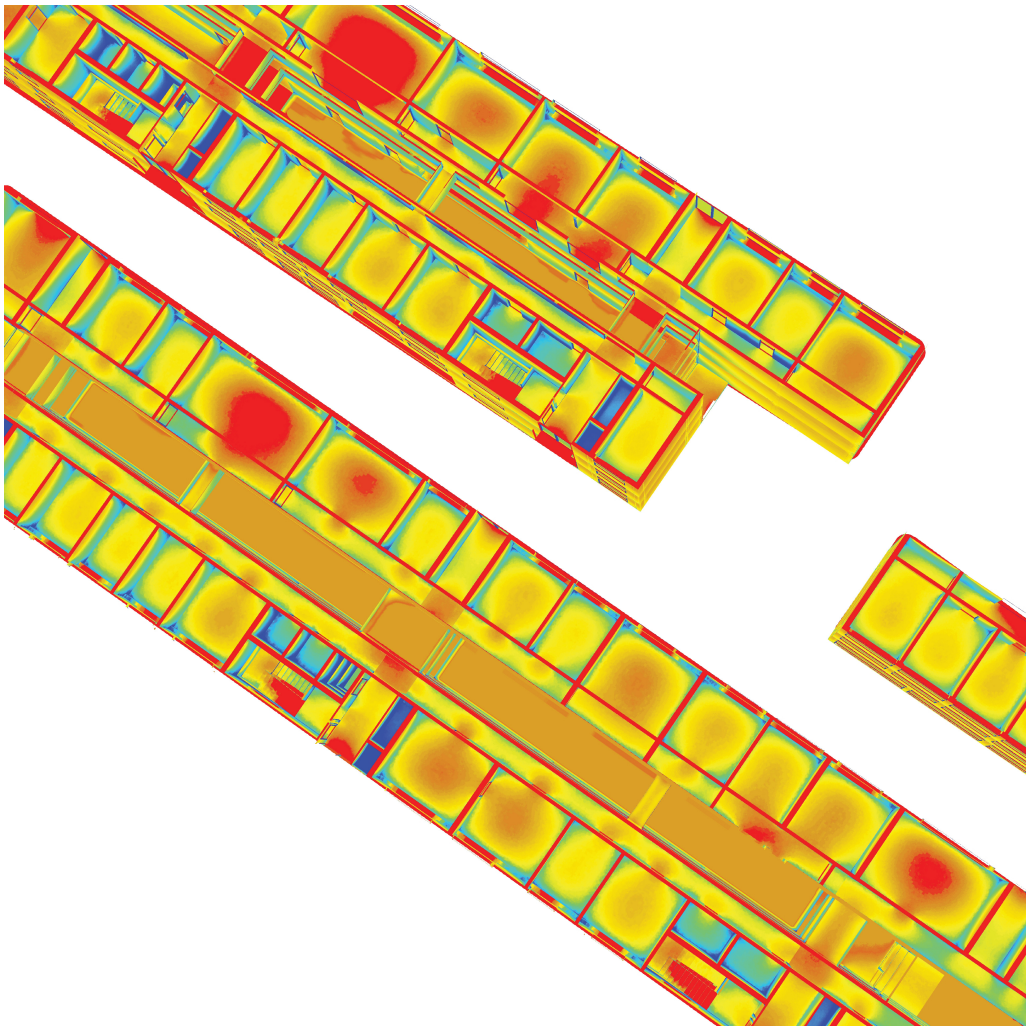


Figure 6.2: Free parameters of a generative model can be set according to aesthetics or according to a cost function; e.g. energy consumption. This concept is an important field of applications for generative optimization techniques.

Bibliography

[ABK98] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A New Voronoi-Based Surface Reconstruction Algorithm. *Computer Graphics*, 32:415–421, 1998.

[ABS06] Marco Attene, Falcidieno Bianca, and Michela Spagnuolo. Hierarchical Mesh Segmentation based on Fitting Primitives. *The Visual*, 22:181–193, 2006.

[ACDL00] Nina Amenta, Sunghee Choi, Tamal K. Dey, and Naveen Leekha. A Simple Algorithm for Homeomorphic Surface Reconstruction. *Proceedings of the 16th annual symposium on Computational Geometry*, 1:213–222, 2000.

[ACG+90] Marianne Akian, Guy Cohen, Stephane Gaubert, Ramine Nikoukhah, and Jean Pierre Quadrat. Linear Systems in (max,+)-Algebra. *Proceedings of the 29th Conference on Decision and Control*, 29:151–156, 1990.

[ACN+05] Marco Allegretti, Marco Colaneri, Riccardo Notaroietro, Marco Gabella, and Giovanni Perona. Simulation in Urban Environment of a 3D Ray Tracing Propagation Model based on Building Database Preprocessing. *Proceedings of the International Union of Radio Science (General Assembly)*, 16:1–4, 2005.

[ADNF+03] Heinz-Wilhelm Alten, Alireza Djafari Naini, Menso Folkerts, Hartmut Schlosser, Karl-Heinz Schlote, and Hans Wußing. *4000 Jahre Algebra: Geschichte, Kulturen, Menschen (english: 4000 years of algebra: history, cultures, men)*. Springer, 2003.

[ADS06] U. H. Augsdörfer, Neil A. Dodgson, and Malcolm A. Sabin. Tuning Subdivision by Minimizing Gaussian Curvature Variation Near Extraordinary Vertices. *Computer Graphics Forum*, 25(3):263–272, 2006.

[AIM06] AIM@SHAPE. A.I.M.A.T.S.H.A.P.E. – Advanced and Innovative Models And Tools for the development of Semantic-based systems for Handling, Acquiring, and Processing knowledge Embedded in multidimensional digital objects. online: http://www.aimatshape.net/, 2006.

[Arn06] David Arnold. Procedural methods for 3D reconstruction. *Recording, Modeling and Visualization of Cultural Heritage*, 1:355–359, 2006.

[AS72] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover Publications, 10 edition, 1972.

[AS93] Günter Aumann and Klaus Spitzmüller. *Computerorientierte Geometrie*. BI-Wissenschafts-Verlag, 1993.

[ASCE02] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. MESH: Measuring Error between Surfaces using the Hausdorff distance. *Proceedings of the IEEE International Conference on Multimedia and Expo 2002 (ICME)*, 1:705–708, 2002.

[Aum08] Günter Aumann. *Euklids Erbe: Ein Streifzug durch die Geometrie und ihre Geschichte*. Wissenschaftliche Buchgesellschaft, 2008.

[Aut07] Autodesk. Autodesk Maya API. *White Paper*, 1:1–30, 2007.

[Baa06] Matthias Baas. Python/Maya: Introductory tutorial. online: http://cgkit.sourceforge.net/maya_tutorials/intro/, 2006.

[Bar86] Andrew R. Barron. Entropy and the Central Limit Theorem. *The Annals of Probability*, 14(1):336–342, 1986.

[BB79] Hans Bandemer and Andreas Bellmann. *Statistische Versuchsplanung (english: Statistical Test Planning)*. Verlag H. Deutsch / BSB B. G. Teubner, 1979.

[BBCS99] Fausto Bernardini, Chandrajit L. Bajaj, Jindong Chen, and Daniel R. Schikore. Automatic Reconstruction of 3D CAD Models from Digital Scans. *International Journal on Computational Geometry and Applications*, 9:327–369, 1999.

[BBH+08] Christian H. Bischof, Martin Bücker, Paul D. Hovland, Uwe Naumann, and Jean Utke. *Advances in Automatic Differentiation*. Springer, 2008.

[BBU+11] Frank Breuel, René Bernd, Torsten Ullrich, Eva Eggeling, and Dieter W. Fellner. Mate in 3D – Publishing Interactive Content in PDF3D. *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, 15:110–119, 2011.

[BBVK04] Mario Botsch, David Bommes, Christoph Vogel, and Leif Kobbelt. GPU-based tolerance volumes for mesh processing. *Proceedings of 12th Pacific Conference on Computer Graphics and Applications*, 12:237– 243, 2004.

[BBW+08] Alexander Berner, Martin Bokeloh, Michael Wand, Andreas Schilling, and Hans-Peter Seidel. A Graph-Based Approach to Symmetry Detection. *Symposium on Volume and Point-Based Graphics*, 5:1–6, 2008.

[Ben75] Jon Louis Bentley. Multidimensional Binary Search Trees used for Associative Searching. *Communications of the ACM*, 18:509 – 517, 1975.

[BF06] Christian Beder and Wolfgang Förstner. Direct Solutions for Computing Cylinders from Minimal Sets of 3D Points. *Proceedings of the 2006 European Conference on Computer Vision*, 3951:135–146, 2006.

[BFH05] René Berndt, Dieter W. Fellner, and Sven Havemann. Generative 3D Models: a Key to More Information within less Bandwidth at Higher Quality. *Proceeding of the 10th International Conference on 3D Web Technology*, 1:111–121, 2005.

[BGVGP06] Manos Baltsavias, Armin Gruen, Luc Van Gool, and Maria Pateraki. *Recording, Modeling and Visualization of Cultural Heritage*. Taylor & Francis, 2006.

[Bie87] Irving Biederman. Recognition-by-Components: A Theory of Human Image Understanding. *Psychological Review*, 94(2):115–147, 1987.

[Bis07] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[BKR04] Tugkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear Algorithms for Testing Monotone and Unimodal Distributions. *Proceedings of ACM Symposium on Theory of Computing*, 36:1–10, 2004.

[BKSS07] Benjamin Bustos, Daniel Keim, Dietmar Saupe, and Tobias Schreck. Content-based 3D Object Retrieval. *IEEE Computer Graphics and Applications*, 27(4):22–27, 2007.

[BKV+02] Pál Benko, Géza Kós, Tamás Várady, László Andor, and Ralph Martin. Constrained Fitting in Reverse Engineering. *Computer Aided Geometric Design*, 19(3):173 – 205, 2002.

[BLZ00] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise Smooth Subdivision Surfaces with Normal Control. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 27:113 – 120, 2000.

[BMMKN04] Boaz Ben-Moshe, Joseph S. B. Mitchell, Matthew J. Katz, and Yuval Nir. Visibility Preserving Terrain Simplification - An Experimental Study. *Computational Geometry: Theory and Applications*, 28:175 – 190, 2004.

[BMSF06] Silvia Biasotti, Simone Marini, Michela Spagnuolo, and Bianca Falcidieno. Sub-part correspondence by structural descriptors of 3D shapes. *Computer-Aided Design*, 38(9):1002–1019, 2006.

[BMY05] Michael Brown, Aditi Majumder, and Ruigang Yang. Camera-Based Calibration Techniques for Seamless Multi-Projector Displays. *IEEE Transactions on Visualization and Computer Graphics*, 11:193–206, 2005.

[BMZ04] Ioana Boier-Martin and Denis Zorin. Differentiable Parameterization of Catmull-Clark Subdivision Surfaces. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 71:155 – 164, 2004.

[Bon09] Peter Bonitz. *Freiformflächen in der rechnerunterstützten Karosseriekonstruktion und im Industriedesign (english: Free-form surfaces in computer-aided car body design and industrial design)*. Springer, 2009.

[BP93] Wolfgang Boehm and Hartmut Prautzsch. *Numerical Methods*. Vieweg, 1993.

[BP94] Wolfgang Boehm and Hartmut Prautzsch. *Geometric concepts for geometric design*. A. K. Peters, Ltd., 1994.

[BR02] Fausto Bernardini and Holly Rushmeier. The 3D Model Acquisition Pipeline. *Computer Graphics Forum*, 21(2):149–172, 2002.

[Bri51] Suzanne Briet. *Qu'est-ce que la documentation?* ÉDIT - éditions documentaires industrielles et techniques, 1951.

[BRT95] Lawrence D. Bergman, Bernice E. Rogowitz, and Lloyd A. Treinish. A rule-based tool for assisting colormap selection. *Proceedings of the 6th Conference on Visualization*, 6:118–125, 1995.

[BS07] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization – A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196:3190–3218, 2007.

[BSdV01] Robert G. Belleman, Bram Stolk, and Raymond de Vries. Immersive Virtual Reality on commodity hardware. *Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging*, 7:297–304, 2001.

[BSMM97] Il'ja N. Bronštein, Konstantin A. Semendjajew, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik (english: Handbook of Mathematics)*. Verlag Harri Deutsch, 1997.

[BSSZ08] Alexander I. Bobenko, Peter Schröder, John M. Sullivan, and Günter M. Ziegler, editors. *Discrete Differential Geometry*. Birkhäuser, 2008.

[BTI07] David Borland and Russell M. Taylor II. Rainbow Color Map (Still) Considered Harmful. *IEEE Computer Graphics and Applications*, 27(2):14–17, 2007.

[Buc97] Michael K. Buckland. What is a "document"? *Journal of American Society of Information Science*, 48(9):804–809, 1997.

[Bühler01] Katja Bühler. Taylor Models and Affine Arithmetics: Towards a More Sophisticated Use of Reliable Methods in Computer Graphics. *Spring Conference on Computer Graphics*, 17:40–48, 2001.

[BWM+11] Alexander Berner, Michael Wand, Niloy J. Mitra, Daniel Mewes, and Hans-Peter Seidel. Shape Analysis with Subspace Symmetries. *Computer Graphics Forum*, 30:277–286, 2011.

[BWS10] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. A Connection between Partial Symmetry and Inverse Procedural Modeling. *Proccedings of ACM SIGGRAPH 2010*, 29:104:1–104:10, 2010.

[CADS09] Thomas J. Cashman, Ursula H. Augsdörfer, Neil A. Dodgson, and Malcolm A. Sabin. NURBS with extraordinary points: high-degree, non-uniform, rational subdivision schemes. *ACM Transactions on Graphics*, 28:46, 1–10, 2009.

[CBC+01] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, Richard W. Fright, Bruce C. McCallum, and Tim R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 28:67 – 76, 2001.

[CC78] Edwin Catmull and Jim Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, 1978.

[CGKV04] TU Graz Computer Graphics & Knowledge Visualization. Generative Modeling Language. online: http://www.generative-modeling.org/, 2004.

[CGT94] Andrew Conn, Nicholas I. M. Gould, and Philippe L. Toint. Large-Scale Nonlinear Constrained Optimization: A Current Survey. *Algorithms for continuous optimization: the state of the art*, 434:287–332, 1994.

[CM05] Giulio Casciola and Serena Morigi. Inverse spherical surfaces. *Journal of Computational and Applied Mathematics*, 176(2):411–424, 2005.

[CMRS08] Artur Czumaj, S. Muthu Muthukrishnan, Ronitt Rubinfeld, and Christian Sohler. *Sublinear Algorithms*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[Col91] Thomas F. Coleman. Large-Scale Numerical Optimization: Introduction and Overview. *Technical Report: TR91-1236*, 9:1–30, 1991.

[Cra77] Bradford R. Crain. An Information Theoretic Approach to Approximating a Probability Distribution. *SIAM Journal on Applied Mathematics*, 32:339–346, 1977.

[CRE01] Elaine Cohen, Richard F. Riesenfeld, and Gershon Elber. *Geometric Modeling with Splines: An Introduction*. A. K. Peters, 2001.

[CRS98] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[CS00] Charilaos Christopoulos and Athanassios Skodras. The JPEG2000 Still Image Coding System: An Overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, 2000.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3):905 – 914, 2004.

[CSLR01] Thomas H. Cormen, Clifford Stein, Charles E. Leiserson, and Robert L. Rivest. *Introduction to Algorithms*. B&T, 2001.

[CTSO03] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On Visual Similarity Based 3D Model Retrieval. *Computer Graphics Forum*, 22(3):223–232, 2003.

[CWQ+04] Kin-Shing D. Cheng, Wenping Wang, Hong Qin, Kwan-Yee Kenneth Wong, Huaiping Yang, and Yang Liu. Fitting Subdivision Surfaces to Unorganized Point Data using SDM. *Proceedings of 12th Pacific Conference on Computer Graphics and Applications*, 1:16–24, 2004.

[CWQ+07] Kin-Shing Cheng, Wenping Wang, Hong Qin, Kwan-Yee K. Wong, Huaiping Yang, and Yang Liu. Design and Analysis of Optimization Methods for Subdivision Surface Fitting. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):878–890, 2007.

[Dau92] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

[DB02] Ioannis Douros and Bernard Buxton. Three-dimensional surface curvature estimation using quadric surface patches. *Proceedings of Scanning*, 2:1–12, 2002.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[DC76] Manfredo Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[dC99] Paul de Faget de Casteljau. De Casteljau's autobiography: My time at Citroën. *Computer Aided Geometric Design*, 16:583–586, 1999.

[DDSD03] Xavier Décoret, Frédo Durand, Francois Sillion, and Julie Dorsey. Billboard Clouds for Extreme Model Simplification. *Proceedings of the ACM Siggraph 2003*, 22(3):689–696, 2003.

[Diw03] Urmila Diwekar. *Introduction to Applied Optimization*, volume 80 of *Applied Optimization*. Springer, 2003.

[DJ94] Marie-Pierre Dubuisson and Anil K. Jain. A Modified Hausdorff Distance for Object Matching. *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, 1:566–568, 1994.

[DKT98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision Surfaces in Character Animation. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 25:85 – 94, 1998.

[DL05] Oliver Deussen and Bernd Lintermann. *Digital Design of Nature: Computer Generated Plants and Organics*. Springer, 2005.

[Dor95] Jean-Luc Dorier. A general outline of the genesis of vector space theory. *Historia Mathematica*, 22(3):227–261, 1995.

[DSVdB08] Bart De Schutter and Ton J. J. Van den Boom. Max-plus algebra and max-plus linear discrete event systems: An introduction. *Proceedings of the International Workshop on Discrete Event Systems*, 9:36–42, 2008.

[EP97] Anton M. Ertl and Christian Pirker. The structure of a Forth native code compiler. *EuroForth '97 Conference Proceedings*, 12:107–116, 1997.

[Eri04] Christer Ericson. *Realtime Collision Detection*. Morgan Kaufmann, 2004.

[Ert96] Anton M. Ertl. Implementation of Stack-Based Languages on Register Machines. *PhD-Thesis, Technische Universität Wien, Austria*, 1:1–85, 1996.

[ESYAE94] Conal Elliott, Greg Schechter, Ricky Yeung, and Salim Abi-Ezzi. TBAG: a high level framework for interactive, animated 3D graphics applications. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 13:421 – 434, 1994.

[Far99] Gerald Farin. *NURBS for Curve and Surface Design from Projective Geometry to Practical Use*. AK Peters, Ltd., 1999.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[Fel01] Dieter W. Fellner. Graphics Content in Digital Libraries: Old Problems, Recent Solutions, Future Demands. *Journal of Universal Computer Science*, 7:400–409, 2001.

[FFBS04] Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O'Reilly Media, Inc., 2004.

[FH05a] Gerald Farin and Dianne Hansford. *Practical Linear Algebra, A Geometry Toolbox*. A K Peters Ltd., 2005.

[FH05b] Dieter W. Fellner and Sven Havemann. Striving for an adequate vocabulary: Next generation metadata. *Proceedings of the 29th Annual Conference of the German Classification Society*, 29:13 – 20, 2005.

[FHH03] Gerald Farin, Bernd Hamann, and Hans Hagen. *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer, 2003.

[Fin08a] Dieter Finkenzeller. Detailed Building Facades. *IEEE Computer Graphics and Applications*, 28(3):58–66, 2008.

[Fin08b] Dieter Finkenzeller. *Modellierung komplexer Gebäudefassaden in der Computergraphik*. Universitätsverlag Karlsruhe, 2008.

[Fis02] Robert B. Fisher. Applying knowledge to reverse engineering problems. *Proceedings of Geometric Modeling and Processing*, 1:149–155, 2002.

[Fle00] Roger Fletcher. *Practical Methods of Optimization*. Wiley, 2000.

[FLS04] John Fisher, John Lowther, and Ching-Kuang Shene. If you know b-splines well, you also nnow NURBS! *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, 35:343 – 347, 2004.

[FMA+10] ÂăAlfredo Ferreira, ÂăSimone Marini, ÂăMarco Attene, ÂăManuel J. Fonseca, ÂăMichela Spagnuolo, ÂăJoaquim A. Jorge, and Bianca Falcidieno. Thesaurus-based 3D Object Retrieval with Part-in-Whole Matching. *International Journal of Computer Vision*, 89:327–347, 2010.

[FPRJ00] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics. *Proceedings of ACM Siggraph 2000*, 17:249–254, 2000.

[FR64] Roger Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7:149–154, 1964.

[FS97] Dieter W. Fellner and Norbert Schenk. MRT - A Tool for Simulations in 3D Geometric Domains. *Proceedings of the 11th European Simulation Multiconference (ESM)*, 11:185–188, 1997.

[FS06] Thomas A. Funkhouser and Philip Shilane. Partial Matching of 3D Shapes with Priority-Driven Search. *Symposium on Geometry Processing*, 4:131–142, 2006.

[FSG03] Arnulph Fuhrmann, Gerrit Sobottka, and Clemens Gross. Distance Fields for Rapid Collision Detection in Physically Based Modeling. *Proceedings of EUROGRAPHICS GraphiCon*, 1:58–65, 2003.

[FSK07] Dieter W. Fellner, Dietmar Saupe, and Harald Krottmaier. 3D Documents. *IEEE Computer Graphics and Applications*, 27(4):20–21, 2007.

[FUF06] Christoph Fünfzig, Torsten Ullrich, and Dieter W. Fellner. Hierarchical Spherical Distance Fields for Collision Detection. *Computer Graphics and Applications*, 26(1):64–74, 2006.

[FUFB07] Christoph Fünfzig, Torsten Ullrich, Dieter W. Fellner, and Edward N. Bachelder. Empirical Comparison of Data Strucutres for Line of Sight Computation. *Proceedings of IEEE International Symposium on Intelligent Signal Processing (WISP) 2007*, 1:291–296, 2007.

[FUFB09] Christoph Fünfzig, Torsten Ullrich, Dieter W. Fellner, and Edward N. Bachelder. Terrain and Model Queries Using Scalar Representation With Wavelet Compression. *IEEE Transactions on Instrumentation and Measurement*, 58:3079–3085, 2009.

[FZ03] Christian Früh and Avideh Zakhor. Constructing 3D City Models by Merging Aerial and Ground Views. *IEEE Computer Graphics and Applications*, 23(6):52 – 61, 2003.

[GBY91] Gaston H. Gonnet and Ricardo Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley Publishing, 1991.

[GCO06] Ran Gal and Daniel Cohen-Or. Salient Geometric Features for Partial Shape Matching and Similarity. *ACM Transactions on Graphics*, 25:130–150, 2006.

[GG04] Natasha Gelfand and Leonidas J. Guibas. Shape Segmentation Using Local Slippage Analysis. *Proceedings of Symposium on Geometry Processing*, 1:219–228, 2004.

[GI04] Jack Goldfeather and Victoria Interrante. A novel cubic-order algorithm for approximating principal direction vectors. *ACM Transactions on Graphics*, 23(1):45–63, 2004.

[GJ02] Joachim Giesen and Michael John. Surface reconstruction based on a dynamical system. *Computer Graphics Forum*, 21(3):363–371, 2002.

[GJH95] Liu Guanghui, Han Jiye, and Yin Hongxia. Global convergence of the fletcher-reeves algorithm with inexact linesearch. *Applied Mathematics - A Journal of Chinese Universities*, 10:75–82, 1995.

[GJK88] Elmer G. Gilbert, Daniel W. Johnson, and Sathiya S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[GK07] Björn Ganster and Reinhard Klein. An Integrated Framework for Procedural Modeling. *Proceedings of Spring Conference on Computer Graphics 2007 (SCCG 2007)*, 23:150–157, 2007.

[GL06] Rich Gossweiler and Mark Limber. SketchUp: An Easy-to-Use 3D Design Tool that Integrates with Google Earth. *Adjunct Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology (UIST06)*, 19:3, 2006.

[GLM96] Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. *Proceedings of 1996 ACM Siggraph*, 30:171–180, 1996.

[GMW82] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1982.

[GOT05] Nick Gould, Dominique Orban, and Philippe Toint. Numerical methods for large-scale nonlinear optimization. *Acta Numerica*, 14:299–361, 2005.

[Gou09] Brian Gough, editor. *Gnu Scientific Library Reference Manual - Third Edition*. Network Theory Ltd, 2009.

[GPMG10] Eric Galin, Adrien Peytavie, Nicolas Marechal, and Eric Guerin. Procedural Generation of Roads. *Computer Graphics Forum*, 29:429–438, 2010.

[Gra97] Alfred Gray. *Modern Differential Geometry of Curves and Surfaces with Mathematica*. CRC, 1997.

[GRLM03] Naga Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware. *SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware*, 1:25–32, 2003.

[Gro03] CIDOC CRM Special Interest Group. *Definition of the CIDOC Conceptual Reference Model*. ICOM/CIDOC Documentation Standards Group, 2003.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition - Special issue: Current issues in knowledge modeling*, 5(2):199–220, 1993.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.

[GWM01] Stefan Gumhold, Xinlong Wang, and Rob MacLeod. Feature Extraction from Point Clouds. *Proceedings of 10th International Meshing Roundtable*, 1:1–13, 2001.

[GWN+03] Markus Gross, Stephan Würmlin, Martin Naef, Edouard Lamboray, Christian Spagno, Andreas Kunz, Esther Koller-Meier, Tomas Svoboda, Luc Van Gool, Silke Lang, Kai Strehlke, Aandrew Vande Moere, and Oliver Staadt. blue-c: A Spatially Immersive Display and 3D Video Portal for Telepresence. *Proceedings of ACM Siggraph 2003*, 22:819–827, 2003.

[Hav05] Sven Havemann. Generative Mesh Modeling. *PhD-Thesis, Technische Universität Braunschweig, Germany*, 1:1–303, 2005.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Proceedings of the 19th annual Conference on Computer Graphics and Interactive Techniques*, 1:71 – 78, 1992.

[Hei07] J.L. Heiberg, editor. *Euclid's Elements of Geometry*. Fitzpatrick, Richard, 2007.

[Hen95] Norbert Henze. *Stochastik – Einführung in die Wahrscheinlichkeitstheorie und Statistik (english: Stochastics – Introduction to probability calculus and statistics)*. Technische Universität Karlsruhe, 1995.

[Hen97] Norbert Henze. *Stochastik für Einsteiger (english: Stochastics for Beginners)*. Vieweg, 1997.

[HF01] Sven Havemann and Dieter W. Fellner. A versatile 3D Model Representation for Cultural Reconstruction. *Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage*, 1:205 – 212, 2001.

[HF04] Sven Havemann and Dieter W. Fellner. Generative Parametric Design of Gothic Window Tracery. *Proceedings of the 5th International Symposium on Virtual Reality, Archeology, and Cultural Heritage*, 1:193–201, 2004.

[HF07] Sven Havemann and Dieter W. Fellner. Seven Research Challenges of Generalized 3d Documents. *IEEE Computer Graphics and Applications*, 3:70–76, 2007.

[HHKR97] Rolf Hammer, Matthias Hocks, Ulrich Kulisch, and Dietmar Ratz. *C++ Toolbox for Verified Computing*. Springer, 1997.

[HK01] Christoph M. Hoffmann and Ku-Jin Kim. Towards valid parametric CAD models. *Computer Aided Design*, 33:81–90, 2001.

[HKD93] Mark Halstead, Michael Kass, and Tony DeRose. Efficient, Fair Interpolation using Catmull-Clark Surfaces. *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 20:35–44, 1993.

[HKT92] Gregory J. Hamlin, Robert B. Kelley, and Josep Tornero. Efficient Distance Calculation using the Spherically-Extended Polytope (S-tope) Model. *Proceedings of Robotics and Automation*, 1:2502–2507, 1992.

[HL89] Josef Hoschek and Dieter Lasser. *Grundlagen der Geometrischen Datenverarbeitung (english: Fundamentals of Computer Aided Geometric Design)*. Teubner, 1989.

[HN02] Marcus Hudec and Christian Neumann. *Stichproben und Umfragen (english: Random Samples and Polls)*. Institut für Statistik der Universität Wien, 2002.

[Hop98] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

[HOP+05] Michael Hofer, Boris Odehnal, Helmut Pottmann, Tibor Steiner, and Johannes Wallner. 3D shape recognition and reconstruction based on line element geometry. *Proceedings of the 10th IEEE International Conference on Computer Vision*, 2:1532–1538, 2005.

[HP02] Hans-Christian Hege and Konrad Polthier. *Mathematical Visualization: Algorithms, Applications and Numerics*. Springer, 2002.

[HP10] Jörn Höllig, Klaus andHöner and Martina Pfeil. *Numerische Methoden der Analysis*. Mathematik-Online, 2010.

[HS52] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[HSKK01] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 28:203–212, 2001.

[HT96] Bernd Hamann and Po-Yu Tsai. A tessellation algorithm for the representation of trimmed NURBS surfaces with arbitrary trimming curves. *Computer Aided Design*, 28:461–472, 1996.

[HTG04] Bruno Heidelberger, Matthias Teschner, and Markus Gross. Detection of Collisions and Self-collisions Using Image-space Techniques. *Proceedings of Winter School of Computer Graphics (WSCG)*, 1:145–152, 2004.

[HUF11] Sven Havemann, Torsten Ullrich, and Dieter W. Fellner. The Meaning of Shape and some Techniques to Extract It. *Multimedia Information Extraction*, page to appear, 2011.

[HZ04] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. ISBN 0521540518.

[Inc85] Adobe Systems Inc. *PostScript Language Reference Manual (first ed.)*. Addison-Wesley, 1985.

[Ing93] Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18:29–57, 1993.

[Ini95] Dublin Core Metadata Initiative. Dublin Core Metadata Initiative. http://dublincore.org/, 1995.

[Iro05] Mark L. Irons. The Curvature and Geodesics of the Torus. *Technical Report, Raindrop Laboratories*, 20:1–19, 2005.

[JBS06] Mark W. Jones, Andreas J. Baerentzen, and Milos Sramek. 3D Distance Fields: A Survey of Techniques and Applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[JF06] Jonas Jacobi and John Fallows. AJAX and Mozilla XUL with JavaServer Faces. *AJAX Developer's Journal*, 2:1–2, 2006.

[JH99] Andrew Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:433–449, 1999.

[JLM02] Kenneth I. Joy, Justin Legakis, and Ron MacCracken. Data Structures for Multiresolution Representation of Unstructured Meshes. *Hierarchical Approximation and Geometric Methods for Scientific Visualization*, 1:1–28, 2002.

[JM90] Cezary Z. Janikow and Zbigniew Michalewicz. A specialized genetic algorithm for numerical optimization problems. *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, 2:798 – 804, 1990.

[Jon95] Mark W. Jones. 3D Distance from a Point to a Triangle. *Technical Report CSR-5-95, Department of Computer Science, University of Wales Swansea*, 5:1–5, 1995.

[KBAP04] Karl Kraus, Christian Briese, Maria Attwenger, and Norbert Pfeifer. Quality Measures for Digital Terrain Models. *Proceedings of International Society for Photogrammetry and Remote Sensing (ISPRS) Conference*, 35:113–118, 2004.

[KBK+01] Konrad Karner, Joachim Bauer, Andreas Klaus, Franz Leberl, and Markus Grabner. Virtual Habitat: Models of the Urban Outdoors. *International Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Imaging*, 3:393–402, 2001.

[KDS99] David Kidner, Mark Dorey, and Derek Smith. What's the point? Interpolation and extrapolation with a regular grid DEM. *Proceedings of '99 GeoComputation*, 4:1–17, 1999.

[Kel99] C.T. Kelley. *Iterative Methods for Optimization*, volume 18 of *Frontier in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), 1999.

[Kel00] Max L. Keler. On the Theory of Screws and the Dual Method. *Proceedings of A Symposium Commemorating the Legacy, Works, and Life of Sir Robert Stawell Ball Upon the 100th Anniversary of "A Treatise on the Theory of Screws"*, 1:1–12, 2000.

[KGL+98] Shankar Krishnan, Meenakshisundaram Gopi, Ming C. Lin, Dinesh Manocha, and Amol Pattekar. Rapid and Accurate Contact Determination between Spline Models using ShellTrees. *Computer Graphics Forum*, 17(3):315–326, 1998.

[KK11] Lars Krecklau and Leif Kobbelt. Procedural Modeling of Interconnected Structures. *Computer Graphics Forum*, 30:335–344, 2011.

[KP03] Dave Knott and Dinesh K. Pai. CInDeR: Collision and Interference Detection in Real-time using Graphics Hardware. *Graphics Interface*, 1:73–80, 2003.

[KPL98] Shankar Krishnan, Amol Pattekar, and Ming C. Lin. Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries. *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics*, 3:177–190, 1998.

[KPVG00] Reinhard Koch, Marc Pollefeys, and Luc Van Gool. Realistic Surface Reconstruction of 3D Scenes from Uncalibrated Image Sequences. *Journal of Visualization and Computer Animation*, 11(3):115–127, 2000.

[KT03] Sagi Katz and Ayellet Tal. Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts. *ACM Transactions on G*, 22:954–961, 2003.

[KW05] Brett D. King and Michael Wertheimer. *Max Wertheimer & Gestalt Theory*. Transaction Publishers, 2005. ISBN 0-7658-0258-9.

[KZ05] Ladislav Kavan and Jiri Zara. Fast Collision Detection for Skeletally Deformable Models. *Computer Graphics Forum*, 24(3):363–372, 2005.

[LC09] Xin Li and Falai Chen. Exact and approximate representations of trimmed surfaces with NURBS and BÃl'zier surfaces. *Proceedings of IEEE International Conference on Computer-Aided Design and Computer Graphics*, 11:286–291, 2009.

[LD98] Bernd Lintermann and Oliver Deussen. A Modelling Method and User Interface for Creating Plants. *Computer Graphics Forum*, 17(1):73–82, 1998.

[LDMA+04] Johnny C. Lee, Paul H. Dietz, Dan Maynes-Aminzade, Ramesh Raskar, and Scott E. Hudson. Automatic Projector Calibration with Embedded Light Sensors. *ACM Symposium on User Interface Software and Technology*, 1:123–126, 2004.

[Ley01] Michael Leyton. *A Generative Theory of Shape*. Springer, 2001.

[LH92] Haim Levkowitz and Gabor T. Herman. Color Scales for Image Data. *IEEE Computer Graphics and Applications*, 12(1):72–80, 1992.

[LH07] Sylvain Lefebvre and Hugues Hoppe. Compressed Random-Access Trees for Spatially Coherent Data. *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, 18:339–349, 2007.

[LM03] Ming C. Lin and Dinesh Manocha. Collision And Proximity Queries. *Handbook of Discrete and Computational Geometry*, 35:1–21, 2003.

[LOU+06] Marcel Lancelle, Lars Offen, Torsten Ullrich, Torsten Techmann, and Dieter W. Fellner. Minimally Invasive Projector Calibration for 3D Applications. *Proceedings of 3. Workshop Virtuelle und Erweiterte Realität der GI-Fachgruppe VR/AR*, 1(1):1–9, 2006.

[LPZ03] Stefan Leopoldseder, Helmut Pottmann, and Hong K. Zhao. The d2-tree: A hierarchical representation of the squared distance function. *Technical Report, Institut of Geometry, Vienna University of Technology*, 101:1–12, 2003.

[LS04] Baoxin Li and Ibrahim Sezan. Automatic keystone correction for smart projectors with embedded camera. *Proceedings of 2004 International Conference on Image Processing*, 4:2829–2832, 2004.

[LWW08] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive Visual Editing of Grammars for Procedural Architecture. *ACM Transactions on Graphics*, 27(3):1–10, 2008.

[LZQ06] Yi Liu, Hongbin Zha, and Hong Qin. Shape topics: A compact representation and new algorithms for 3d partial shape retrieval. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:2025–2032, 2006.

[Ma05] Weiyin Ma. Subdivision surfaces for CAD - an overview. *Computer-Aided Design*, 37(7):693–709, 2005.

[Mac03] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[Max88] Nelson L. Max. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer*, 4:109–117, 1988.

[May11] Mark T. Maybury, editor. *Multimedia Information Extraction*. 2011.

[MB07] Mokshay Madiman and Andrew R. Barron. Generalized Entropy Power Inequalities and Monotonicity Properties of Information. *IEEE Transactions on Information Theory*, 53(7):2317–2329, 2007.

[McL99] Michael P. McLaughlin. *A Compendium of Common Probability Distributions*. Regress+, 1999.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. *Visualization and Mathematics III*, 3:35–57, 2003.

[MGGP06] J. Mitra, Niloy, Leonidas Guibas, Joachim Giesen, and Mark Pauly. Probabilistic Fingerprints for Shapes. *Symposium on Geometry Processing*, 4:121–130, 2006.

[MGP06] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM Transactions on Graphics*, 25:560 – 568, 2006.

[MGP07] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Symmetrization. *International Conference on Computer Graphics and Interactive Techniques*, 26:1–8, 2007.

[MH00] Kerstin Müller and Sven Havemann. Subdivision Surface Tesselation on the Fly using a versatile Mesh Data Structure. *Computer Graphics Forum*, 19(3):151–159, 2000.

[MH03] Ying Liang Ma and Terry W. Hewitt. Point inversion and projection for NURBS curve and surface: control polygon approach. *Computer Aided Geometric Design*, 20(2):79–99, 2003.

[Mic95] Zbigniew Michalewicz. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, 4:135–155, 1995.

[MK05] Martin Marinov and Leif Kobbelt. Optimization methods for scattered data approximation with subdivision surfaces. *Graphical Models*, 67(5):452 – 473, 2005.

[MMTP04] Weiyin Ma, Xiaohu Ma, Shiu-Kit Tso, and Zhigeng Pan. A direct approach for subdivision surface fitting from a dense triangle mesh. *Computer-Aided Design*, 36(6):525–536, 2004.

[MMY06] Ricardo A. Maronna, Douglas R. Martin, and Victor J. Yohai. *Robust Statistics: Theory and Methods*. Wiley Series in Probability and Statistics. John Wiley and Sons, 2006.

[Moo91] Andrew W. Moore. An introductory tutorial on kd-trees. *Technical Report, Computer Laboratory, University of Cambridge*, 209:1–20, 1991.

[MP09] Ashish Myles and Jörg Peters. Bi3 C2 polar subdivision. *Proceedings of International Conference on Computer Graphics and Interactive Techniques SIGGRAPH*, 28:48:1–48:12, 2009.

[MPT99] William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. Six Degree-Of-Freedom Haptic Rendering using Voxel Sampling. *Proceedings of the 26th annual Conference on Computer Graphics and Interactive Techniques*, 26:401 – 408, 1999.

[MRRT53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall W. Rosenbluth, and Augusta H. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

[MS96] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4:1–32, 1996.

[MSF07] Simone Marini, Michela Spagnuolo, and Bianca Falcidieno. Structural Shape Prototypes for Automatic Classification of 3D Objects. *IEEE Computer Graphics and Applications*, 27(4):28–37, 2007.

[MSH+08] Erick Mendez, Gerhard Schall, Sven Havemann, Dieter W. Fellner, Dieter Schmalstieg, and Sebastian Junghanns. Generating Semantic 3D Models of Underground Infrastructure. *IEEE Computer Graphics and Applications*, 28:48–57, 2008.

[MSHS06] Aurélien Martinet, Cyril Soler, Nicolas Holzschuch, and Francois Sillion. Accurate Detection of Symmetries in 3D Shapes. *ACM Transactions on Graphics*, 25:439 – 464, 2006.

[MWH+06] Pascal Müller, Peter Wonka, Simon Haegler, Ulmer Andreas, and Luc Van Gool. Procedural Modeling of Buildings. *Proceedings of 2006 ACM Siggraph*, 25(3):614–623, 2006.

[MZWVG07] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based Procedural Modeling of Facades. *ACM Transactions on Graphics*, 28(3):1–9, 2007.

[Nas90] John C. Nash. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Adam Hilger, second edition edition, 1990.

[Nex09] NextEngine. User's Guide. online: http://support.nextengine.com, 2009.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 1999.

[OA93] Review Board OpenGL Architecture. *OpenGL Reference Manual*. Addison-Wesley Publishing Company, 1993.

[OB07] Björn Ommer and Joachim M. Buhmann. Learning the Compositional Nature of Visual Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1–8, 2007.

[oC99] Committee on Cataloging, Description & Access. Task Force on Metadata - Summary Report. *American Library Association*, 4:1–16, 1999.

[ÖK08] Mine Özkar and Sotirios Kotsopoulos. Introduction to shape grammars. *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH 2008 (course notes)*, 36:1–175, 2008.

[ONOT98] Masahiro Okamoto, Taisuke Nonaka, Shuichiro Ochiai, and Daisuke Tominaga. Nonlinear numerical optimization with use of a hybrid genetic algorithm incorporating the modified Powell method. *Applied Mathematics and Computation*, 91:63–72, 1998.

[oST10] National Institute of Standards and Technology. Digital Library of Mathematical Functions. online: http://dlmf.nist.gov, 2010.

[Ous98] John K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. *IEEE Computer Magazine*, 31(3):23–30, 1998.

[PBP02] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-Spline Techniques*. Springer, 2002.

[Pea90] Frederick Pearson. *Map Projections: Theory and Applications*. CRC, 1990.

[Pet00] Jörg Peters. Patching Catmull-Clark meshes. *International Conference on Computer Graphics and Interactive Techniques*, 27:255–258, 2000.

[Pet04] Martin Peternell. Developable Surface Fitting to Point Clouds. *Computer Aided Geometric Design*, 21:785–803, 2004.

[PFC+05] Paolo Pingi, Andrea Fasano, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Exploiting the scanning sequence for automatic registration of large sets of range maps. *Computer Graphics Forum*, 24(3):517–526, 2005.

[PG04] Michael D. Proctor and William J. Gerber. Line-of-sight Attributes for a Generalized Application Program Interface. *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1:43–57, 2004.

[PH03a] Helmut Pottmann and Michael Hofer. Geometry of the squared distance function to curves and surfaces. *Visualization and Mathematics III, H.C. Hege and K. Polthier (eds.)*, 3:223–244, 2003.

[PH03b] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. *International Conference on Computer Graphics and Interactive Techniques*, 22:340 – 349, 2003.

[Pin02] Janos D. Pinter. Global Optimization: Software, Test Problems, and Applications. *Handbook of Global Optimization, P.M. Pardalos and H.E. Romeijn (eds)*, 2:515–569, 2002.

[PKG03] Mark Pauly, Richard Keiser, and Markus Gross. Multi-scale Feature Extraction on Point-Sampled Surfaces. *Computer Graphics Forum*, 22(3):281–289, 2003.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[PM01] Yogi Parish and Pascal Mueller. Procedural Modeling of Cities. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 28:301–308, 2001.

[PMW+08] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics*, 27:#43, 1–11, 2008.

[PPV95] Alberto Paoluzzi, Valerio Pascucci, and Michele Vicentino. Geometric Programming: a Programming Approach to Geometric Design. *ACM Transactions on Graphics*, 14:266–306, 1995.

[PR69] Elijah Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue Francaise Informat. Reserche Opérationnelle*, 16:35–43, 1969.

[Pra04] Micheal J. Pratt. Extension of ISO 10303, the STEP Standard, for the Exchange of Procedural Shape Models. *Proceedings of the Shape Modeling International*, 0:317 – 326, 2004.

[PT97] Les Piegl and Wayne Tiller. *The NURBS book*. Springer-Verlag New York, Inc., 1997.

[PTK05] Pralay Pal, A. M. Tigga, and A. Kumar. Feature extraction from large CAD databases using genetic algorithm. *Computer-Aided Design*, 37(5):545–558, 2005.

[PU98] Hartmut Prautzsch and Georg Umlauf. Improved Triangular Subdivision Schemes. *Proceedings of the Computer Graphics International*, 3:626–632, 1998.

[PW09] Jorg Peters and Xiaobin Wu. The distance of a subdivision surface to its control polyhedron. *Journal of Approximation Theory*, to appear:to appear, 2009.

[PWL01] Helmut Pottmann, Johannes Wallner, and Stefan Leopoldseder. Kinematical methods for the classification, reconstruction and inspection of surfaces. *Comptes rendus du Congres national de mathematiques appliquees et industrielles*, 1:51–60, 2001.

[RA99] Ravi Ramamoorthi and James Arvo. Creating Generative Models from Range Images. *Proceedings of ACM Siggraph*, 1:195–204, 1999.

[Rab02] Christophe Rabut. On Pierre Bézier's life and motivations. *Computer-Aided Design*, 34:493–510, 2002.

[Ras00] Ramesh Raskar. Immersive Planar Display using Roughly Aligned Projectors. *Proceedings of 2000 IEEE Virtual Reality*, 1:109–116, 2000.

[RB01] Ramesh Raskar and Paul Beardsley. A Self-Correcting Projector. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:504–508, 2001.

[RBB01] Holly Rushmeier, Laurent Balmelli, and Fausto Bernardini. Horizon Map Capture. *Computer Graphics Forum*, 20(3):85âĂŞ94, 2001.

[Rei90] Glenn C. Reid. *Thinking in Postscript*. Addison-Wesley, 1990.

[Rem03] Fabio Remondino. From Point Cloud To Surface: The Modeling And Visualization Problem. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS) International Workshop on Visualization and Animation of Reality-based 3D Models*, 34:228–238, 2003.

[RFM95] Arturo A. Rodriguez, Martin Fisher, and Brian Markey. Scripting Languages Emerge in Standards Bodies. *IEEE MultiMedia*, 2(4):88 – 92, 1995.

[RFM07] Casey Reas, Ben Fry, and John Maeda. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2007.

[Ron09] Alfredo M. Ronchi. *eCulture – Cultural Content in the Digital Age*. Springer, 2009.

[Ros97] Jarek Rossignac. The 3D Revolution: CAD Access for All! *Proceedings of the 1997 International Conference on Shape Modeling and Applications (SMA '97)*, 29:64–70, 1997.

[RTB96] Bernice E. Rogowitz, Lloyd A. Treinish, and Steve Bryson. How NOT to lie with visualization. *Computers in Physics*, 10:268 – 273, 1996.

[RVB02] Dirk Reiners, Gerrit Voss, and Johannes Behr. OpenSG: Basic concepts. *Proceedings of OpenSG Symposium 2002*, 1:1–7, 2002.

[RvdH04] Tahir Rabbani and Frank van den Heuvel. Methods for Fitting CSG Models to Point Clouds and their Comparison. *Proceedings of 2004 Computer Graphics and Imaging*, 1:1–6, 2004.

[Sah00] Subhasis Saha. Image Compression – from DCT to Wavelets: A Review. *Crossroads*, 6(3):12–21, 2000.

[SBM+10] Ondrej Stava, Bedrich Benes, Radomir Mech, Daniel G. Aliaga, and Peter Kristof. Inverse Procedural Modeling by Automatic Generation of L-systems. *Proccedings of EUROGRAPHICS, Computer Graphics Forum*, 29:665–674, 2010.

[SBU+10] Andreas Schiefer, René Berndt, Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Service-Oriented Scene Graph Manipulation. *Proceedings of the 15th International Conference on Web 3D Technology*, 15:55–62, 2010.

[SBW02] Shigeo Sugimoto, Thomas Baker, and Stuart L. Weibel. Dublin Core: Process and Principles. *Lecture Notes in Computer Science – Digital Libraries: People, Knowledge, and Technology*, 2555/2010:25–35, 2002.

[SDS96] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, 1996.

[Sel06] Ilijas Selimovic. Improved algorithms for the projection of points on NURBS curves and surfaces. *Computer Aided Geometric Design*, 23(5):439–445, 2006.

[SGS+05] Brian Salomon, Naga Govindaraju, Avneesh Sud, Ming Gayle, Russelland; Lin, Dinesh Manocha, Brett Butler, Maria Bauer, Angel Rodriguez, Latika Eifert, Audrey Rubel, and Michael Macedonia. Accelerating Line of Sight Computation Using Graphics Processing Units. *Proceedings of the 2005 Conference on Interservice/Industry Training, Simulation and Education*, 1:1–5, 2005.

[Sha98] Craig M. Shakarji. Least-Squares Fitting Algorithms of the NIST Algorithm Testing System. *Journal of Research of the National Institute of Standards and Technology*, 106(6):633–641, 1998.

[Sha02] Vadim Shapiro. Solid Modeling. *Handbook of Computer-Aided Geometric Design, G. Farin, J. Hoschek, and M.-S. Kim (editors)*, 20:473–518, 2002.

[SHS02] Wenyu Sun, Jiye Han, and Jie Sun. Global convergence of nonmonotone descent methods for unconstrained optimization problems. *Journal of Computational and Applied Mathematics*, 146:89–98, 2002.

[SHVG02] Tomas Svodoba, Hanspeter Hug, and Luc Van Gool. ViRoom - Low Cost Synchronized Multicamera System and its Self-Calibration. *Pattern Recognition, 24th DAGM Symposium*, 24:515–522, 2002.

[SK92] John M. Snyder and James T. Kajiya. Generative modeling: a symbolic system for geometric modeling. *Proceedings of 1992 ACM Siggraph*, 1:369–378, 1992.

[SK98] Yoshihisa Shinagawa and Tosiyasu L. Kunii. Unconstrained Automatic Image Matching Using Multiresolutional Critical-Point Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(9):994–1010, 1998.

[SKS06] Patricio Simari, Evangelos Kalogerakis, and Karan Singh. Folding meshes: hierarchical mesh segmentation based on planar symmetry. *Proceedings of the fourth Eurographics symposium on Geometry processing*, 4:111–119, 2006.

[SMFF04] Volker Settgast, Kerstin Müller, Christoph Fünfzig, and Dieter W. Fellner. Adaptive Tesselation of Subdivision Surfaces. *Computers and Graphics*, 28(1):73–78, 2004.

[SMKF04] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas A. Funkhouser. The Princeton Shape Benchmark. *Shape Modeling International*, 8:1–12, 2004.

[Sny97] John P. Snyder. *Flattening the Earth: Two Thousand Years of Map Projections*. University Of Chicago Press, 1997.

[SP97] Rainer Storn and Kenneth Price. Differential Evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[SRO+08] Markus Steiner, Philipp Reiter, Christian Ofenböck, Volker Settgast, Torsten Ullrich, Marcel Lancelle, and Dieter W. Fellner. Intuitive Navigation in Virtual Environments. *Proceedings of Eurographics Symposium on Virtual Environments*, 14:5–8, 2008.

[SS04] Christoph J. Scriba and Peter Schreiber. *5000 Jahre Geometrie: Geschichte, Kulturen, Menschen (english: 5000 years of geometry: history, cultures, men)*. Springer, 2004.

[SSB+10] Thomas Schiffer, Andreas Schiefer, René Berndt, Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Enlightened by the Web – A service-oriented architecture for real-time photorealistic rendering. *Kongress Multimediatechnik*, 5:41–48, 2010.

[SSM01] Rahul Sukthankar, Robert G. Stockton, and Matthew D. Mullin. Smarter Presentations: Exploiting Homography in Camera-Projector Systems. *Proceedings of International Conference on Computer Vision*, 1:247–253, 2001.

[SSUF10a] Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner. Modeling Procedural Knowledge – a generative modeler for cultural heritage. *Proccedings of EUROMED 2010 - Lecture Notes on Computer Science*, 6436:153–165, 2010.

[SSUF10b] Martin Strobl, Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. Euclides – A JavaScript to PostScript Translator. *Proccedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, 1:14–21, 2010.

[SSUF11a] Thomas Schiffer, Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. Real-World Geometry and Generative Knowledge. *The European Research Consortium for Informatics and Mathematics (ERCIM) News*, 86:to appear, 2011.

[SSUF11b] Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner. Modeling Procedural Knowledge – a generative modeler for cultural heritage. *Selected Readings in Computer Graphics 2010*, page to appear, 2011.

[Sta98] Jos Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1:395 – 404, 1998.

[Ste98] James A. Stewart. Fast Horizon Computation at All Points of a Terrain with Visibility and Shading Applications. *IEEE Transactions on Visualization and Computer Graphics*, 4:82 – 93, 1998.

[Ste99] Charles V. Stewart. Robust Parameter Estimation in Computer Vision. *SIAM Review*, 41(3):513 – 537, 1999.

[Sto03] Maureen Stone. *A Field Guide to Digital Color*. AK Peters, Ltd., 2003.

[SUF07] Volker Settgast, Torsten Ullrich, and Dieter W. Fellner. Information Technology for Cultural Heritage. *IEEE Potentials*, 26(4):38–43, 2007.

[SUF11] Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. Simple and Efficient Normal Encoding with Error Bounds. *Poster Proceedings of Theory and Practice of Computer Graphics*, page to appear, 2011.

[SUSF11] Christoph Schinko, Torsten Ullrich, Thomas Schiffer, and Dieter W. Fellner. Variance Analysis and Comparison in Computer-Aided Design. *Proceedings of the International Workshop on 3D Virtual Reconstruction and Visualization of Complex Architectures*, XXXVIII-5/W16:3B21–25, 2011.

[SW06] Arne Schmitz and Martin Wenig. The effect of the radio wave propagation model in mobile ad hoc networks. *Proceedings of the 9th ACM international Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, 9:61 – 67, 2006.

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, 2007.

[SWWK07] Ruwen Schnabel, Roland Wahl, Raoul Wessel, and Reinhard Klein. Shape Recognition in 3D Point Clouds. *Technical report*, 1:1–9, 2007.

[Tec04] Torsten Techmann. Unterteilungsflächen: Datenstruktur und Regeln. *Technical Report TUBS-CG*, 02:1–27, 2004.

[TGRZ07] Elif Tosun, Yotam I. Gingold, Jason Reisman, and Denis Zorin. Shape Optimization Using Reflection Lines. *Proceedings of the Symposium on Geometry Processing*, 257:193–202, 2007.

[tHCMV05] Frank ter Haar, Paolo Cignoni, Patrick Min, and Remco Veltkamp. A Comparison of Systems and Tools for 3D Scanning. *3D Digital Imaging and Modeling: Applications of Heritage, Industry, Medicine and Land*, 1:1–8, 2005.

[THM+03] Matthias Teschner, Bruno Heidelberger, Matthias Mueller, Danat Pomeranets, and Markus Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. *Proceedings of 2003 Vision, Modeling, and Visualization*, 1:47–54, 2003.

[TKH+04] Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, Marie-Paule Cani, Francois Faure, Nadia Magnenat-Thalmann, Wolfgang Strasser, and Volino Pascal. Collision Detection for Deformable Objects. *Eurographics State of the Art Report*, 24(1):1–20, 2004.

[TSHM05] David Tuft, Brian Salomon, Sean Hanlon, and Dinesh Manocha. Fast Line-of-Sight Computations in Complex Environments. *Technical Report TR05-025*, 25:1–6, 2005.

[UB05] Ilkay Ulusoy and Christopher W. Bishop. Generative versus Discriminative Methods for Object Recognition. *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:258 – 265, 2005.

[UF04a] Torsten Ullrich and Dieter W. Fellner. AlgoViz - a Computer Graphics Algorithm Visualization Toolkit. *World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-Media)*, 16:941–948, 2004.

[UF04b] Torsten Ullrich and Dieter W. Fellner. Modulare Inhaltserzeugung nach dem Baukastenprinzip. *DeLFI 2004: Die e-Learning Fachtagung der Gesellschaft für Informatik 2004*, 52:405–406, 2004.

[UF05] Torsten Ullrich and Dieter W. Fellner. Computer Graphics Courseware. *Proceedings of Eurographics 2005 Education*, 1:11–17, 2005.

[UF07a] Torsten Ullrich and Dieter W. Fellner. Client-Side Scripting in Blended Learning Environments. *The European Research Consortium for Informatics and Mathematics (ERCIM) News*, 71:43–44, 2007.

[UF07b] Torsten Ullrich and Dieter W. Fellner. Robust Shape Fitting and Semantic Enrichment. *Proceedings of the 2007 International Symposium of the International Committee for Architectural Photogrammetry (CIPA)*, 21:727–732, 2007.

[UF11a] Torsten Ullrich and Dieter W. Fellner. Generative Object Definition and Semantic Recognition. *Proccedings of the Eurographics Workshop on 3D Object Retrieval*, 4:1–8, 2011.

[UF11b] Torsten Ullrich and Dieter W. Fellner. Linear Algorithms in Sublinear Time – a tutorial on statistical estimation. *IEEE Computer Graphics and Applications*, 31:58–66, 2011.

[UFF07] Torsten Ullrich, Christoph Fünfzig, and Dieter W. Fellner. Two Different Views On Collision Detection. *IEEE Potentials*, 26(1):26–30, 2007.

[UKF08] Torsten Ullrich, Ulrich Krispel, and Dieter W. Fellner. Compilation of Procedural Models. *Proceeding of the 13th International Conference on 3D Web Technology*, 13:75–81, 2008.

[USB10] Torsten Ullrich, Volker Settgast, and René Berndt. Semantic Enrichment for 3D Documents: Techniques and Open Problems. *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, 14:374–384, 2010.

[USF08a] Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Abstand: Distance Visualization for Geometric Analysis. *Project Paper Proceedings of the Conference on Virtual Systems and MultiMedia Dedicated to Digital Heritage (VSMM)*, 14:334–340, 2008.

[USF08b] Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Semantic Fitting and Reconstruction. *Journal on Computing and Cultural Heritage*, 1(2):1201–1220, 2008.

[USF09] Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Semantic Fitting and Reconstruction. *Selected Readings in Computer Graphics 2008*, 19:69–84, 2009.

[USF10a] Torsten Ullrich, Andreas Schiefer, and Dieter W. Fellner. Modeling with Subdivision Surfaces. *Proceedings of the 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision*, 18:1–8, 2010.

[USF10b] Torsten Ullrich, Christoph Schinko, and Dieter W. Fellner. Procedural Modeling in Theory and Practice. *Poster Proceedings of the 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision*, 18:5–8, 2010.

[USK+07] Torsten Ullrich, Volker Settgast, Ulrich Krispel, Christoph Fünfzig, and Dieter W. Fellner. Distance Calculation between a Point and a Subdivision Surface. *Proceedings of 2007 Vision, Modeling and Visualization (VMV)*, 1:161–169, 2007.

[USOF09] Torsten Ullrich, Volker Settgast, Christian Ofenböck, and Dieter W. Fellner. Short Paper: Desktop Integration in Graphics Environments. *Proccedings of the 2009 Joint Virtual Reality Conference of Eurographics Symposium on Virtual Environments (EGVE), International Conference on Artificial Reality and Telexistence (ICAT), and EuroVR (INTUITION) Conference*, 15:109–112, 2009.

[UTF08] Torsten Ullrich, Torsten Techmann, and Dieter W. Fellner. Web-based Algorithm Tutorials in Different Learning Scenarios. *World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-Media)*, 20:5467–5472, 2008.

[VGSR04] George Vosselman, Ben G. H. Gorte, George Sithole, and Tahir Rabbani. Recognizing Structure in Laser Scanner Point Clouds. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 46(8):33–38, 2004.

[VMC97] Tamás Vradi, Ralph R. Martin, and Jordan Cox. Reverse Engineering of Geometric Models - An Introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

[Voo91] Douglas Voorhies. Space-Filling Curves and a Measure Of Coherence. *Graphics Gems II*, 2:26–30, 1991.

[VP04] Sébastien Valette and Rémy Prost. A Wavelet-Based Progressive Compression Scheme For Triangle Meshes: Wavemesh. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):123–129, 2004.

[VV04] Emily A. Vander Veer. *JavaScript for Dummies*. For Dummies, 2004.

[WBK08] Raoul Wessel, Rafael Baranowski, and Reinhard Klein. Learning Distinctive Local Object Characteristics for 3D Shape Retrieval. *Proceedings of Vision, Modeling, and Visualization (VMV)*, 8:167–178, 2008.

[Wei09] Eric Weisstein. *MathWorld – A Wolfram Web Resource*. Wolfram Research, 2009.

[WGK05] Roland Wahl, Michael Guthe, and Reinhard Klein. Identifying Planes in Point-Clouds for Efficient Hybrid Rendering. *Proceedings of The 13th Pacific Conference on Computer Graphics and Applications*, 1:1–8, 2005.

[WK05] Jianhua Wu and Leif Kobbelt. Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.

[WK10] Raoul Wessel and Reinhard Klein. Learning the Compositional Structure of Man-Made Objects for 3D Shape Retrieval. *Proceedings of EUROGRAPHICS Workshop on 3D Object Retrieval*, 3:39–46, 2010.

[WKZ06] Rene Weller, Jan Klein, and Gabriel Zachmann. A Model for the Expected Running Time of Collision Detection using AABB Trees. *Proceedings of the 12th Eurographics Symposium on Virtual Environments*, 1:11–17, 2006.

[WPL04] Wenping Wang, Helmut Pottmann, and Yang Liu. Fitting B-Spline Curves to Point Clouds by Curvature-Based Squared Distance Minimization. *ACM Transactions on Graphics*, 25:214–238, 2004.

[WXL+11] Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhi-Quan Cheng, and Y. Xiong. Symmetry Hierarchy of Man-Made Objects. *Computer Graphics Forum*, 30:287–296, 2011.

[Yan00] Chuan-kai Yang. Integration of Volume Visualization and Compression: A Survey. *Technical Report, State University of New York, USA*, 1:1–53, 2000.

[Zac98] Gabriel Zachmann. Rapid Collision Detection by Dynamically Aligned DOP-Trees. *Proceedings of the Virtual Reality Annual International Symposium*, 1:90–97, 1998.

[Zha97] Zhengyou Zhang. Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting. *Image and Vision Computing Journal*, 15(1):59–76, 1997.

[ZKGGK06] Lukas Zebedin, Andreas Klaus, Barbara Gruber-Geymayer, and Konrad Karner. Towards 3D map generation from digital aerial images. *Proceedings of ISPRS (International Society for Photogrammetry and Remote Sensing)*, 60:413–427, 2006.

[Zon06] Zhi Zong. *Information-Theoretic Methods for Estimating of Complicated Probability Distributions*. Elsevier, 2006.

[ZPKG02] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: an interactive system for point-based surface editing. *Proceedings of 2002 ACM Siggraph*, 21:322 – 329, 2002.

[ZSD+00] Denis Zorin, Peter Schröder, Tony DeRose, Leif Kobbelt, Adi Levin, and Wim Sweldens. Subdivision for Modeling and Animation. *SIGGRAPH 2000 Course Notes*, 1:1–116, 2000.

[ZSS96] Denis Zorin, Peter Schröder, and Wim Sweldens. Interpolating Subdivision for Meshes with Arbitrary Topology. *Proceedings of 1996 ACM Siggraph*, 23:189–192, 1996.

Biographical and historical information within this thesis are based on the books "4000 Jahre Algebra: Geschichte, Kulturen, Menschen" (english: 4000 years of algebra: history, cultures, men) [ADNF+03] and "5000 Jahre Geometrie: Geschichte, Kulturen, Menschen" (english: 5000 years of geometry: history, cultures, men) [SS04] and on articles published on the online encyclopedia "WikiPedia"[2] and JOHN J. O'CONNOR's and EDMUND F. ROBERTSON's "MacTutor History of Mathematics archive".[3]

[2] WikiPedia – the Free Encyclopedia (http://www.wikipedia.org)

[3] MacTutor History of Mathematics archive (http://www-history.mcs.st-andrews.ac.uk/index.html)

Index