Graz University of Technology

Institute for Computer Graphics and Vision

MASTER'S THESIS

Draft Copy: September 19, 2011

Interactive CSG Rendering of Polyhedral Bounded Objects

Qiang CHEN

Graz, Austria, July 2011

examinants

Univ.-Prof. Dr. Dieter Schmalstieg

Thesis supervisors

Dr. Alexander Bornik

Dr. Markus Grabner

Abstract

The GPU-based ray casting algorithm is an image-based method and is appropriately applied in real-time multiple volume rendering. In many real-time applications, the unimportant areas of the volume is cut away. The rendering technique, which combines with volume clipping approach, can be used to remove the specific volumes or parts of volumes in the current scene. In order to process the multiple volumes, the Constructive solid Geometry (CSG) is used to construct more complex objects. The propose of this thesis is to implement CSG rendering algorithm in the multiple volume rendering, which supports complex translucent and concave polyhedral objects in any CSG combination. The CSG algorithm allows all available volumes to be specified as a high-resolution polygon mesh, which is independent on its appearance (transfer function). Therefore, the resolution and orientation of all volumes can be modified independently from the transfer function during rendering without any performance decrease. To achieve a higher performance, the CSG rendering system is implemented in CUDA, which has full access to the high bandwidth and low latency memory unit.

The CSG rendering algorithm is a combination of the ray casting rendering algorithm and the volume clipping algorithm. Both algorithms are correlated with the depth structure of the polygonal data. The CSG operations are converted into a suitable form in the rendering pipeline and are evaluated in each depth-based polygonal segment interval in the fragment process. The optimizations of CSG and the integration with the volume clipping algorithm are discussed. Furthermore, the comparison between the CSG rendering and the ray casting multiple volume rendering is given, by including and excluding with an evaluation of the volume clipping technique.

Keywords. Volume Clipping, Constructive Solid Geometry, Ray-Casting, Multiple Volume Rendering

STATUTORY DECLARATION

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

Date: ____________________ **Signed:** ____________________

Acknowledgments

I would like to thank Dr. Alexander Bornik, Dr. Markus Grabner for their help and supervision during this project, Dr. Alexander Bornik for his support on the work with the Volume Clipping Implementation, Dr. Markus Grabner for his support on the work with the Scene Graph and Prof. Dr. Dieter Schmalstieg for his additional support and more tips.

Contents

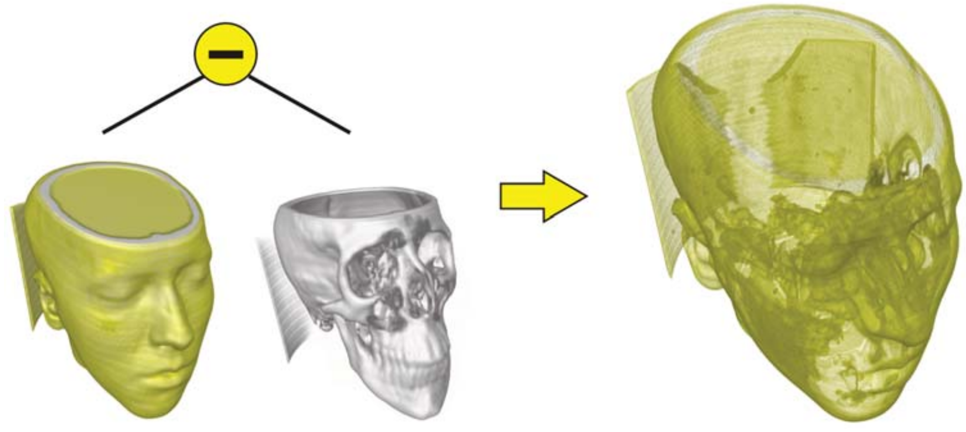
List of Figures

List of Tables

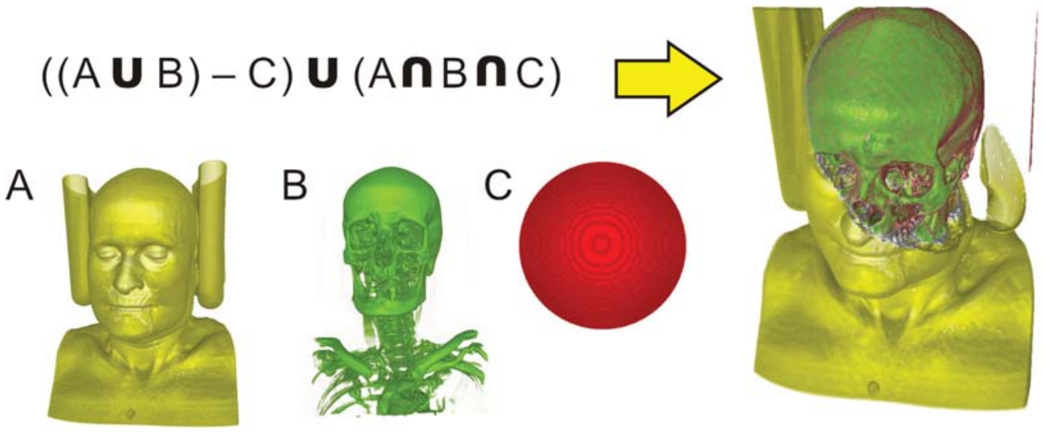
Chapter 1

Introduction

CSG is a well-established way to represent complex three-dimensional object in Computer Aided Design (CAD). Along with the increasing availability of 3D imaging modalities like Computed Tomography (CT), Magnetic Resonance Image (MRI) and 3D ultrasound, visualization of multiple datasets has became more important. However, creation of a visualization from overlapping data regions is more than just classification by means of a transfer function. Volume segmentation and/or clipping can be used to remove unnecessary detected regions by using the CSG-paradigm combined with both polyhedral and volume dataset.
The CSG model is a tree structure, in which the interior nodes are associated with the CSG operators and leafs are associated with the corresponding volume data sets derived from different imaging modalities. The semantic of the CSG model is to evaluate the CSG operations in the corresponding child nodes. The CSG representation allows users to define complex 3D solid objects by hierarchically combining simple geometric primitives with Boolean operations and affine transformations [13]. More complex objects are currently produced by this technique. The CSG operations allow to split or to merge different data sets, which are located on the leaf nodes. Fig. 1.1(a) is a CSG volume clipping example by using the Boolean subsection to cut the head bone from the original volume data and the remaining part is the outer skin. Fig. 1.1(b) illustrates a CSG application in medical field, the interested region (e.g. head bone) becomes visible by excluding the unwanted data (e.g. head skin).
In chapter 2, an Overview is given to show the related studies that have been done in the areas of CSG rendering algorithm. In the following chapter chapter 3, we introduce the CSG model, including tree transformation and pruning algorithms. The volume clipping



(a) Head - Head Bone = Head Skin



(b) View of Intersecting

Figure 1.1: Volume-Based CSG Application Examples

algorithm and the ray-casting rendering algorithm are described in chapter 4. The the implementation of our volume clipping based CSG rendering algorithm and the relevant evaluation are presented in Chap. 5 and 6, respectively.

Chapter 2

Prior Works: CSG Rendering Algorithms

Contents

The CSG rendering problem, which determines the visible parts of primitives from a particular view situation, can be regarded as two sub-problems [21]: clipping and visibility determination. The clipping parts are invisible, which reset as the transparent in the scene. The visible parts are the rendered parts in the scene, which is visible by evaluating sum-of-product.

In this chapter we review some representative studies regarding the CSG rendering algorithm. There are two main ways to render the CSG tree, namely *Obejct-space* approach and *Image-space* approach. In terms of *object-space* approach, boundary evaluation [13] [14] converts the CSG representation into the boundary representation (BRep), which is typically polygonal and can be passed directly to the rendering pipeline. This approach solves the clipping problem view-independently. Regarding *image-space* approach, the clipping and visibility problems are solved on a per-pixel basis in the fragment data. The visible segment of each pixel is dependent on the view position, object shapes and the CSG model. The image-space approach includes ray-casting method (ray-casting) and z-buffer algorithms (face-culling and depth/layered-peeling). The modern GPUs support most image-based algorithms and accelerate the rendering for the image-space approach. Table 2.1 shows the representative techniques of these two approaches. Note that all algo-

CSG rendering	algorithms	concave	volume	on-line/off-line
Image Space	Ray-Casting	$\surd$	$\surd$	on-line
	Face Culling	$\times^1$	$\surd$	on-line
	Layer/Depth Peeling	$\times^1$	$\surd$	on-line
Object Space	Boundary Evaluation	$\surd$	$\times$	off-line

Table 2.1: CSG Rendering Algorithms. All algorithms are classified with the supported of concave and volume primitives; the calculations of the mechanism are on-line or off-line. 1. the concave primitive is split into several convex primitives.

rithms support standard convex and geometry primitives. In the following sections, several algorithms based on these two approaches mentioned above are described in details.

2.1 Boundary Evaluation

Requicha et al. [14] investigated the boundary evaluation and the merging algorithms, which are applied in the Boolean operations in solid modeling system. The solid is characterized as BReps, such as faces, edges and vertices structures. Fig. 2.1 shows an example of the CSG and BReps.

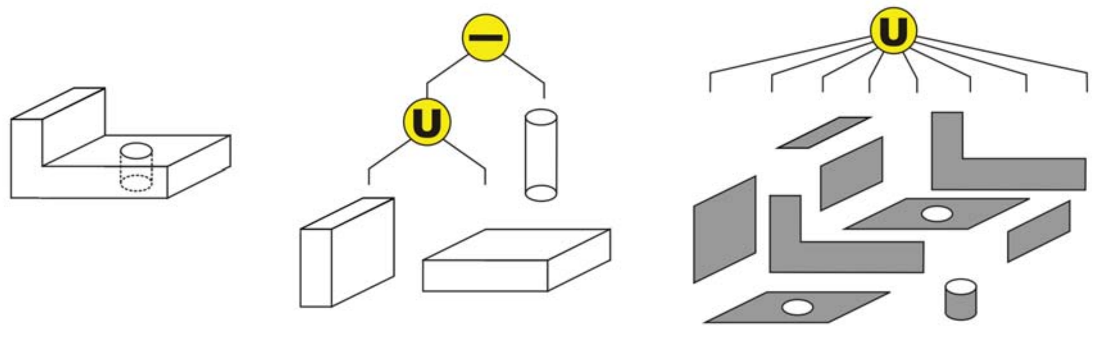
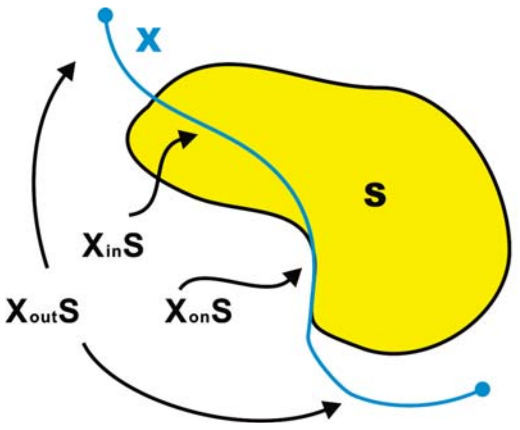
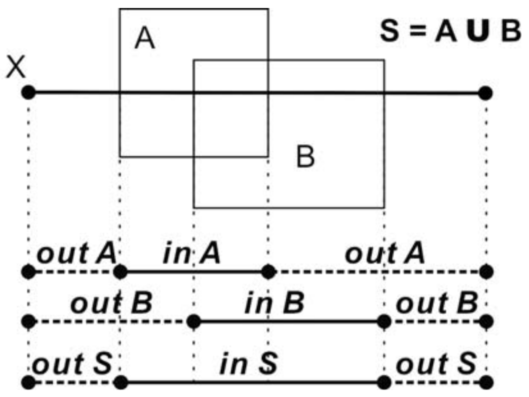

(a) Solid (b) A CSG Representation (c) A Boundary Representation

Figure 2.1: A simple Solid, an associated CSG Tree and BRep [14].

The key concept of this algorithm is the set membership classification and neighbourhood manipulation, which describe boundary evaluation and merging procedures. The set membership classification splits the given curve segment X related to a reference set S into subsets XinS, XonS and XoutS, which mean the X is inside, on the boundary of , and outside S, respectively [14] (see Fig. 2.2(a)). The classifications of geometry can be combined by using the Boolean operator (see Fig. 2.2(b)). The set membership classifica-



(a) Classification of a curve segment X



(b) Combining Classifications

Figure 2.2: Set Membership Classification[14].

tion algorithms have two tasks: (1) subdivide X into cells of uniform classification against primitives and (2) combine the binary result of classifying these cells against the primitives according to corresponding Boolean expression. The neighborhood manipulation is used to resolve the overlapping boundaries problem in the case of XonS. The CSG primitive is split into the BReps; the boundary evaluation applies the Boolean operation to the face, edge, vertex structure layers and classifies the segment into several sets related to the reference set, and then combines the classifications by using the set operators.

The boundary evaluation is appropriate for the exact evaluation of the CSG operations because the calculation of the CSG operation is divided into faces, edges and vertices. The algorithms also supports the convex and the concave polyhedral geometry. However, this algorithm needs a huge memory capacity to perform the evaluation, which exponentially grows with the number of primitives. Hence, it is difficult to support larger CSG models. Another disadvantage is that the BReps only supports geometry object.

2.2 Ray-Casting

The ray casting approach solves the clipping and visibility problems in each pixel. The clipping problem is solved in a depth-based clipping approach. For each pixel along the view direction, a ray is generated and is further cast into primitives. This algorithm splits the ray segment into different intervals to classify the interior or the exterior of the primitives.

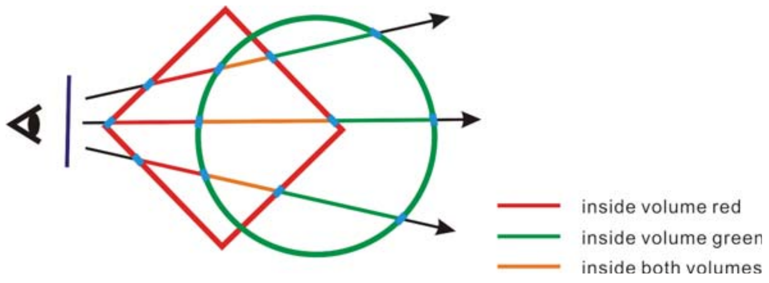
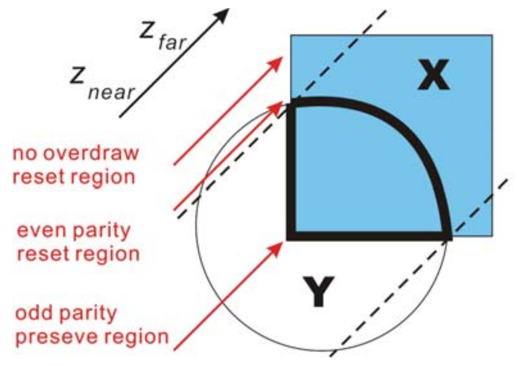


Figure 2.3: The intersection between the ray segment and primitives.

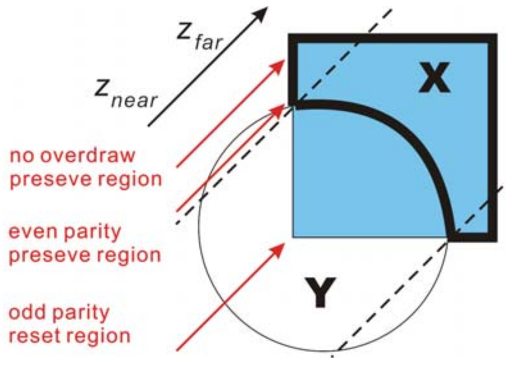
Fig. 2.3 shows that the ray casts from the view point to the primitives. Assuming that the object is defined as a polygon mesh, for each ray, an intersection test is applied to the each polygon mesh and the ray. The intersected depth values can be calculated according to the intersection. All intersected polygons are collected along the ray direction, and sorted by depth values in ascending order during the ray casting. Each pixel has its own sorting polygon list. The evaluation of CSG operations determines which split intervals (between two intersected polygons) are solid or vacant. For each ray, the exact visible segment is defined. Therefore, the ray can be only traversed in the visible region. The whole ray casting process requires multi-pass rendering, either the depth polygon list calculation or the ray traversing.
The ray casting approach maps the pixels of the viewing plan to the triangles of the objects. The advantage of the ray casting approach is its simplicity. But the performance of the standard ray casting depends on the depth complexity of the intersected objects. The ray-casting is a on-line method, supports the geometry and volume clipping, the convex and concave primitives as well. This method is appropriately applied in the CSG rendering algorithm, and is described in the later chapter in detail.

2.3 Front/Back-face Culling

The Goldfeather CSG rendering is a hardware-based z-buffer algorithm, which supports convex [5] and concave objects [6]. The CSG tree is converted into a sum-of-product form before rendering. This conversion uses the normalised rules and also ensures that the CSG expression can be rendered effectively using z-Buffer supported graphics hardware. In the standard Goldfeather CSG rendering algorithm, two z-buffers, which are responsible for the temporary and the final results, are used to present the sum-of-product. In this alogirhtm, each CSG primitive is split into the front and the back faces. According to the CSG operations, one of these two faces is cut, the front and the back faces are visible by intersection and subtraction, respectively. Then, the surface parity test is applied to check whether a surface in the z-buffer is inside or outside the object. The parity test (Fig. 2.4)



(a) Parity test for $X \cap Y$



(b) Partiy test for $X - Y$

Figure 2.4: Parity Test

is performed by counting the number of surfaces in the z-buffer. Regions of odd parity, where the parity flag is set to be one, are matched to z-buffer elements volumetrically inside an object. In this algorithm, an intersection preserves the odd parity regions and resets the even parity regions, whereas a subtraction resets the odd parity regions and preserves the even parity regions.

Fig. 2.5 shows an example of the Goldfeather CSG rendering algorithm. In each primitive, the face is clipped away from the faces of all other primitives in the local z-buffer. The resulting primitives are merged into the z-buffer by the corresponding depth-test.

There is only one z-buffer in the rendering pipeline and two methods are used to realise the second buffer. One uses half-precision of depth-value to split the z-buffer into two, while

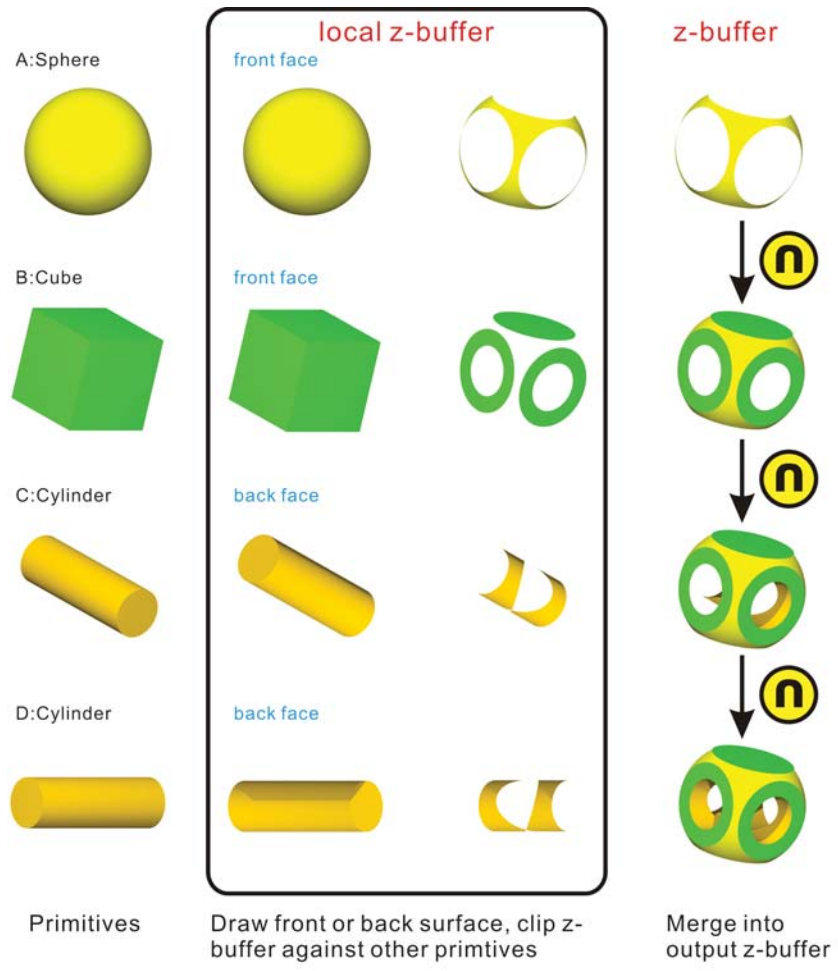
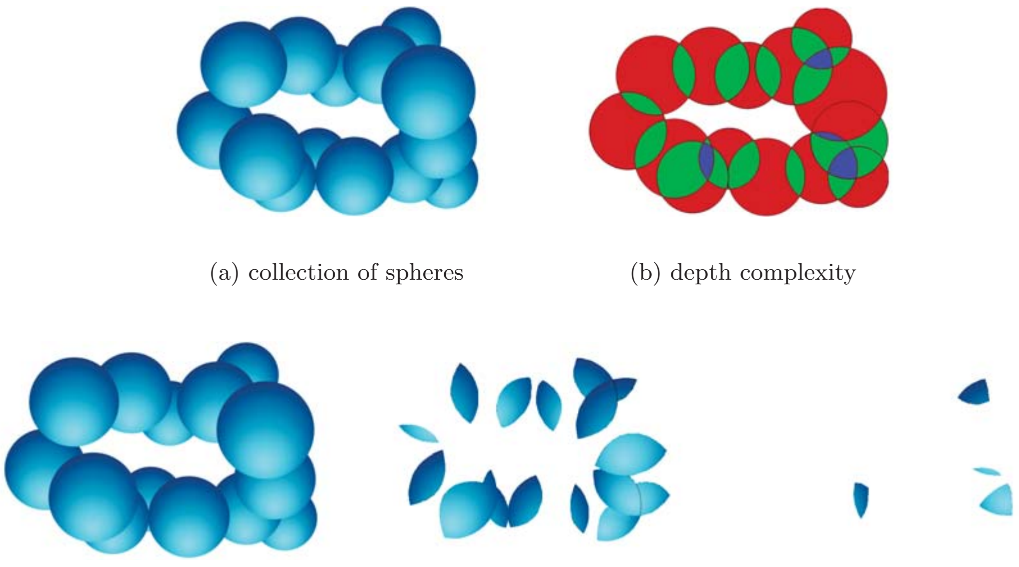


Figure 2.5: Goldfeather CSG Rendering Algorithm. $(A \cap B) - (C \cup D)$. [21]

the other stors the z-buffer as texture, which has a lower speed during the copy. Both methods have a significant performance bottleneck [24] [21], which indicates either the precision or the speed decreasing. The Goldfeather CSG rendering is an on-line method, which support the geometry and the volume objects. The algorithm does not support the concave primitives, which are cut into serveral convex primitives and lead to increase of the calculation complexity. The Goldfeather CSG algorithm only tests every primitive against all others, and requires $O\left(n^2\right)$ times memory copy in the worst case.

2.4 Layered/depth-Peeling

Stewart et al. reported the layered Goldfeather Algorithm [19], which tested the overlapping of the primitives based on the depth complexity of the scene (the maiximum overdraw, see Fig. 2.6). The algorithm extracts a layer and tests it against all primitives


(a) collection of spheres (b) depth complexity

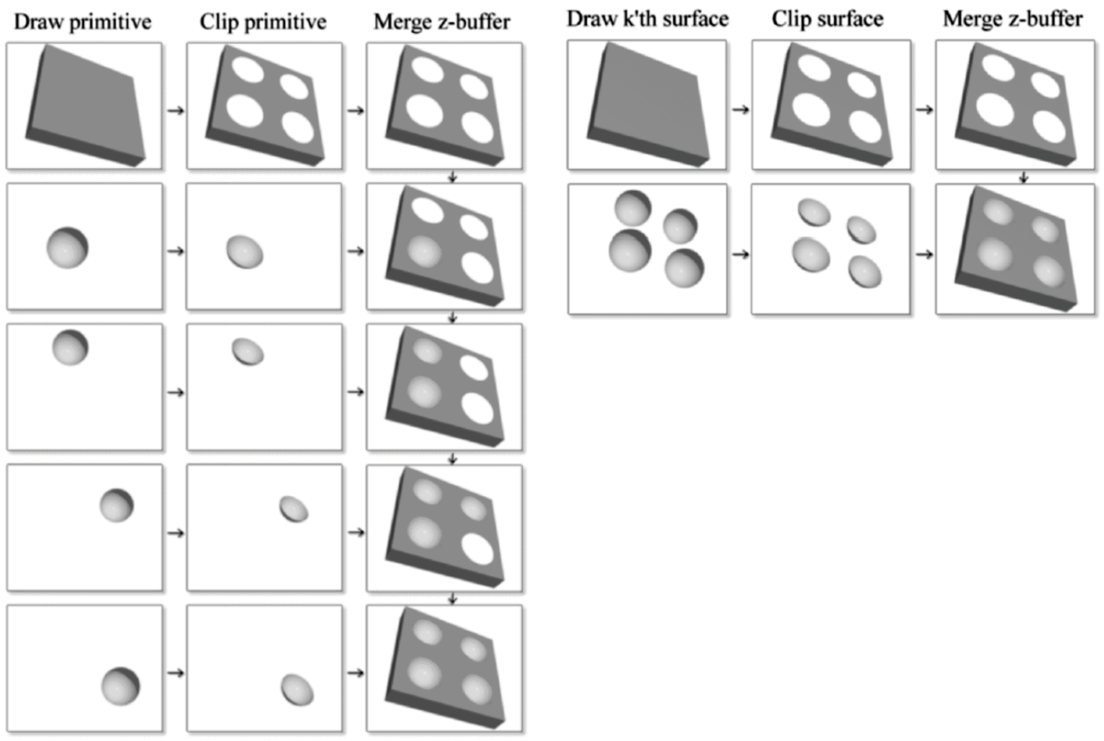
(c) 1st layer (d) 2nd layer (e) 3th layer

Figure 2.6: Layer Extraction of Sphere Front Faces, 1st layer is the objects which does not overlap with another, 2nd layer is the overlapped parts of two objects, 3th layer is the overlapped parts of three objects.

of the product. The resulting layer is stored in the temporary z-buffer and is merged into the final result. The layered Goldfeather algorithm improves the memory copy in $O\left(kn\right)$ times, where k is the maximum depth complexity of the scene. Figs 2.7(a) and 2.7(b) are the clipping examples for the standard and layered Goldfeather algorithm. The layered algorithm is better in the case with bigger n and smaller k.

Erhart and Tobbler store the binary masks of the layer clipping result in the stencil buffer, rather than the depth buffer [4]. Instead of one mask per primitive, one mask per layer is stored. The stencil data can be copied more efficiently than z-buffer. Another advantage is that the stencil buffer is an array of integer values, which can be easily copied without conversion or any loss of precision.

Stewart et al. present another improved variant, the Sequenced Convex Subtraction (SCS)



(a) Clipping on per-primitive basis (b) Clipping on per-layer basis

Figure 2.7: Clipping on Per-Primitive/Per-Layer Basis (adopted from [19])

algorithm for image-space CSG rendering [20], which only needs the linear time with respect to the number of intersections ($O(n^2)$ times by the standard Goldfeather and $O(kn)$ times by the layered Goldfeather). Details of the SCS algorithm are described in the reference [20].

All improved Goldfeather algorithms have the same properties as the Goldfeather algorithm and still uses the second z-buffer to store the temporary result. The performance bottleneck still exists but is reduced. All Goldfeather algorithms are the rasterization algorithm. Compared to the ray casting, the rasterization alogirthm computes which pixels belong to an intersection, while the ray casting computes which intersection belong to a pixel. The intersections are performed as a per-computation step of the rendering, which does not work on a per-ray level. Hence, the multi-pass rendering is required.

Chapter 3

Constructive Solid Geometry

Contents

The CSG is a way to represent complex object of base primitives in a hierarchy structure. The CSG model contents the geometry primitives and/or the volume data sets, such as CT volume, PET data set. These input data are combined by Boolean operations such as union, subtraction and intersection to create a more complex surface or volume object. The tree structure (hierarchical structure) keeps the solid geometries and Boolean operations. In order to efficiently render any CSG model, the corresponding tree is transformed into useful forms which are suitable for hardware-based CSG rendering: sum-of-products, positive form. In this chapter, we discuss the CSG transformation and simplification.

3.1 CSG Model

The CSG modelling constructs the CSG tree by user defined interactive application, which is a combination of pre-defined CSG solid primitives by using Boolean operations. The solid primitive is defined in the 3D space and is a elemental component in the CSG model. The solid primitives has two-manifold property, which require the boundary surface, a clear defined bounding between the inner and the outer region. In the CSG modelling

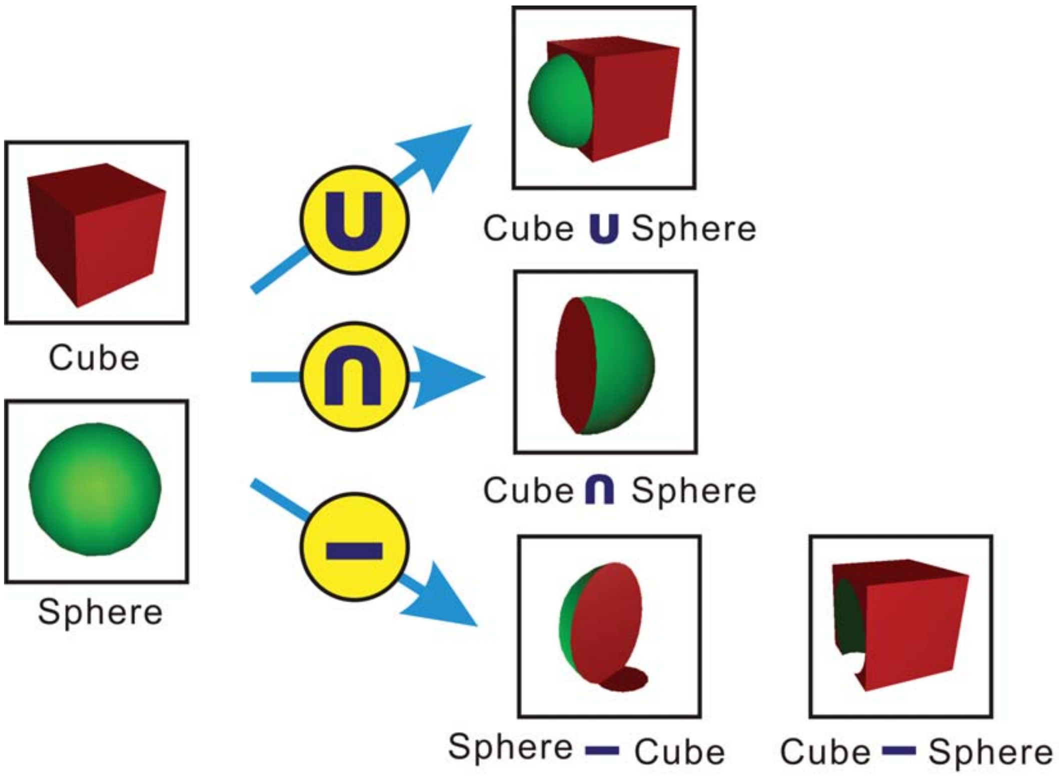


Figure 3.1: Primitives (cube, sphere) and Boolean operators (∪: intersection, ∩: union, −: subtraction.): Given two primitives, A and B, their union consists of all boundary surfaces and volume regions from either A or B; the intersection consists of the boundary surface and volume region in both primitives; and their difference, written as $A - B$ (or $B - A$ in reverse), consists of the boundary surface and volume region in A subtracting B (or in B subtracting A).

system, all CSG primitives use the transformations such as translation, rotation, scaling (affine transformation) to be inserted into the scene.
Boolean operations combine two CSG (transformated) primitives into one to construct a more complicated CSG shape. There are three Boolean operations: ([8],[21]):

- **Union** $(A \cup B)$: The resulting shape consists of all regions either in the A, in the B or in both input shapes, as well as the boundary surface.

- **Intersection** $(A \cap B)$: The resulting shape is the overlapped region of both input shapes, includes the intersected boundary surface.

- **Subtraction** $(A - B)$: The resulting shape is the region by subtracting the A from the region of the B, the surface of the A is outside of B and the surface of the B inside of A.

Fig. 3.1 illustrates examples of Boolean union, intersection and subtraction. The resulting geometry is a solid geometry, which has the two-manifold surface area, and can be further used as a CSG primitive. The hierarchical structure is usually a binary tree, in which each parent node (Boolean operations) has only two child nodes and can be evaluated by the CSG shapes or CSG primitives. The ambiguity of the CSG model can be removed by using the binary tree. In fact, the CSG model has a corresponding CSG tree with associated expression. Fig. 3.2 shows a CSG model by using a binary tree structure. The associated expression is written as $((a \cap b) - (c \cap d))$.

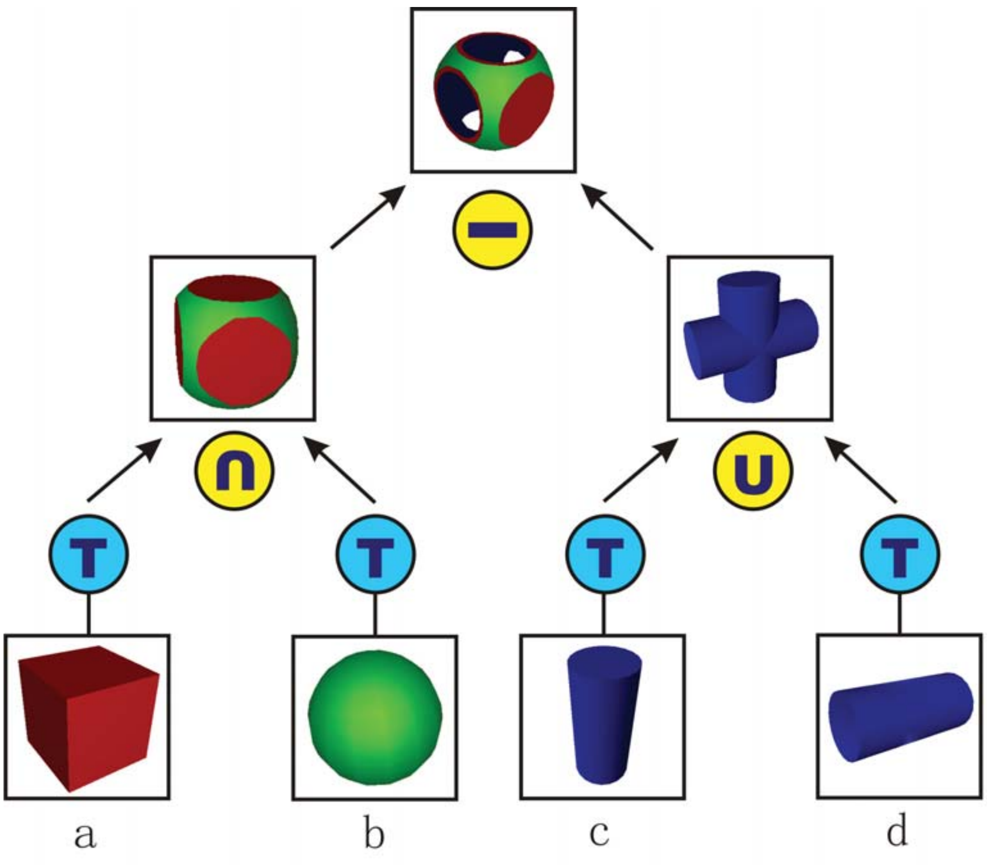


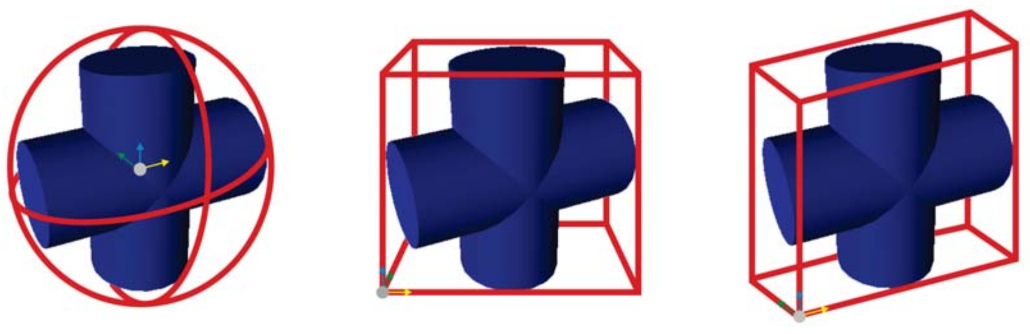
Figure 3.2: CSG objects can be represented with binary trees, where leaves represent primitives, and internal nodes represent Boolean operators. (T is the corresponding transformation which converts the geometry from object space into the world space)

3.2 Scene Graph based CSG model

Unlike the CSG model, only the solid primitives and Boolean operators are stored in the tree. The scene graph renders a 3D scene, is a use-oriented tree structure that is augumented with textures, transforms, level-of-detail, render states (such as material properties), light sources, etc. [1]. We focus on properties of the scene graph modelling in this section, and extend scene graph to a suitable interpretation of the CSG model.
Node is the fundamental element of a scene graph. There are three different types of nodes in scene graph [23], which are

- **Shape nodes**, representing 3D geometric objects.
- **Property nodes**, representing appearance and qualitative characteristics of the scene, such as surface material, drawing, or geometric transformation.
- **Group nodes**, containing nodes in the graphs. Groups collect property, shape, and other group nodes into graphs.

A shape node holds the actual geometry and the corresponding bounding volume (BV). A BV is a volume which encloses a set of objects. Fig. 3.3 shows examples of BV: Bounding



(a) Bounding Sphere

(b) Axis-aligned Bounding Boxes

(c) Oriented Bounding Boxes

Figure 3.3: Examples of Bounding Volumes

Spheres, Axis-aligned Bounding Boxes (AABBs), Oriented Bounding Boxes (OBBs). An internal group node is an implicit OR-Association and collects its BV on all child nodes in the scene graph, which contains the bounding volume hierarchy (BVH). Each node in the tree has a bounding volume that encloses the primitive in its entire subtree [1]. The BVH

is a type of spatial data structure, which can be used to identify the intersected region.
The scene graph can be used to represent the CSG model. The group nodes are implemented as the Boolean operations, whereas property nodes keep the location and the shape nodes are the primitives. The shape nodes and property nodes are leaf nodes. The scene graph is a hierAll nodes can be inserted as the child nodes in a group node.
Each type of node has its own behaviour. The scene graph can apply several actions in its tree structure, such as rendering the scene graph, computing a 3D bounding box or cumulative transformation of objects. Each action requests a traversal state, which is a collection of elements or parameters in the action at a given time [23]. Executing an action involves traversing the graph in pre-order (visiting the parent node before its child nodes) and post-order (visiting the child nodes before their parent). The traversal state can be modified during the traversing. In the CSG rendering algorithm, the system should determine which volume is present in the current scene before the rendering. One possibility is that we can use an action to evaluate the semantic of the CSG operations.
The scene graph is the real time interactive manipulation, which is able to modify the tree in the real time and is realised by the recursive call in the tree structure. The scene graph contains the data about transforms, bounding volumes, render state and animation state. Each node in the scene graph corresponds to a drawable object. The scene graph updates the data of each node in multiple ways. For example, when a new volume is inserted into the scene, the scene graph has to be updated. All transformations are updated in preorder transversal from the root downward the leaves. Afterwards, the bounding volumes are updated in post-order traversal from the leaves upward the root. The CSG rendering can check the BV as a preprocess to detect the intersected region. This process can be used in the BV pruning technique, which is explained in the following section.

3.3 Tree Transform

In the practice, the user-defined CSG tree can be very complex. The rendering system does not provide aid for any possible CSG combination. In Boolean algebra, any associated Boolean expression can be expressed in a canonical form using AND-Association and OR-Association. Here, a general way is used to convert the CSG tree into useful form suitable for rendering pipeline, namely the sum-of-products, which is an union of intersections or subtractions. A CSG tree is normalized when this tree is converted into a sum-of-products form. In this section we explain the Goldfeather algorithm [5] [6] for tree normalization.

A normalized CSG expression is denoted as:

$$P_1 \cup P_2 \cup \cdots \cup P_m \tag{3.1}$$

where each P_i is a product of CSG primitives. A single product is an expression including intersections and subtractions.

A set of production rules, which use the associative and distributive rules of Boolean algebraic properties, are applied to the CSG tree recursively. The normalization process uses the the local node information (Boolean operators of current node and its child node) to decide a production rule applied in the current node. The corresponding production rules are listed in the table 3.1. In the CSG tree, all normalized rules move the union operator

The Normalization Algorithm

E_1 :	$x - (y \cup z)$	$=$	$(x - y) - z$
E_2 :	$x \cap (y \cup z)$	$=$	$(x \cap y) \cup (x \cap z)$
E_3 :	$x - (y \cap z)$	$=$	$(x - y) \cup (x - z)$
E_4 :	$x \cap (y \cap z)$	$=$	$(x \cap y) \cap z$
E_5 :	$x - (y - z)$	$=$	$(x - y) \cup (x \cap z)$
E_6 :	$x \cap (y - z)$	$=$	$(x \cap y) - z$
E_7 :	$(x \cup y) - z$	$=$	$(x - y) \cup (y - z)$
E_8 :	$(x \cup y) \cap z$	$=$	$(x \cap y) \cup (y \cap z)$

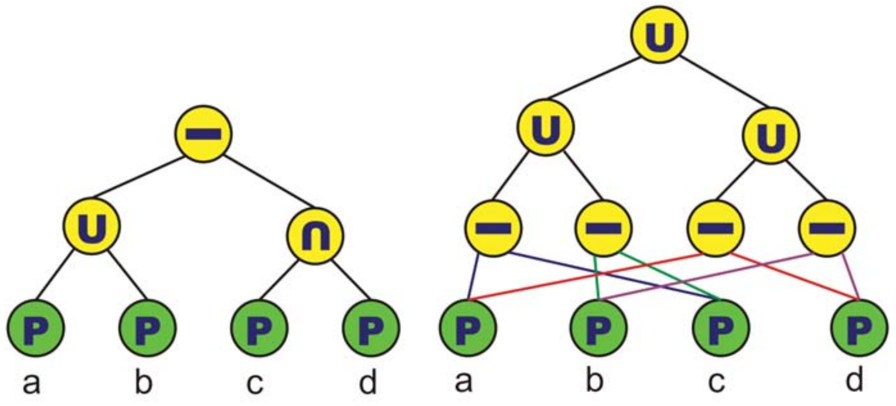
Table 3.1: Set Equivalences for Normalization[6]

```
procedure normalize(T : tree)
{reduce a CSG tree to sum − of − products form}
begin
    if T is primitive then return;
    repeat
        while T matches left side of any set equivalence
            replace with right side, apply first matching rule;
        normalize(T.left);
    until T.op is a union or (T.right is a primitive and T.left is not a union)
    normalize(T.right);
end normalize;
```

Table 3.2: CSG Tree Normalisation[6][24]

upward and the intersection and subtraction downward. The rules $E_2, E3, E5, E8, E9$ in Table 3.1 duplicate the child nodes, which lead to an explosive increments of the number of the nodes in the model. In the next section, we present the pruning techniques to simplify the normalized Boolean expression. Table 3.2 shows the pseudo code of the normalized algorithm. The normalization starting at the root node processes the CSG tree recursively in a top-down order. Goldfeather et. al. [6] guarantee that this algorithm is terminated by any given CSG tree as input and there is no redundant part or repeated primitive in the resulting CSG tree. We use a example (see Fig. 3.4(a)) to present the tree normalization algorithm as follows:

$$\begin{aligned}
(a \cup b) - (c \cap d) &\longrightarrow ((a \cup b) - c) \cup ((a \cap b) - d) && (E_3)\\
&\longrightarrow ((a \cup b) - c) \cup ((a - d) \cup (b - d)) && (E_7)\\
&\longrightarrow ((a - b) \cup (b - c)) \cup ((a - d) \cup (b - d)) && (E_7)
\end{aligned}$$



(a) CSG tree example (b) normalized CSG tree

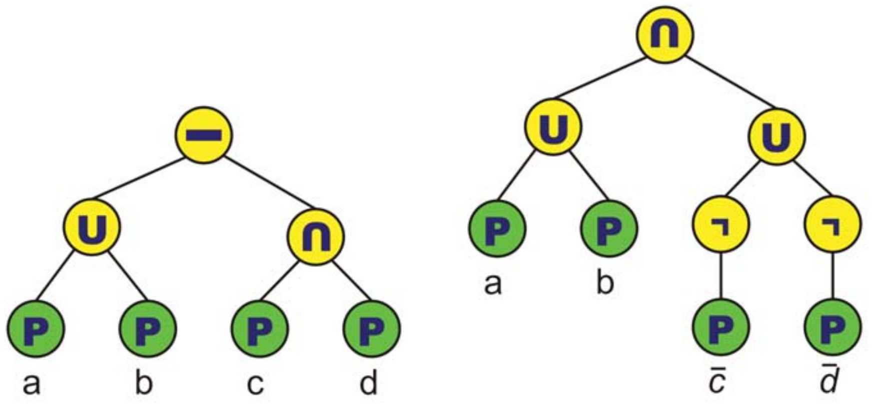
Figure 3.4: CSG Tree Transformation. The CSG tree is transformed into its normalized form.

The algorithm starts at the root and the root node is a subtraction. The rule E_3 is found and applied in the root node. The result is a permutation of the subtraction. The normalised algorithm then moves to right part of the root node, and in this case, the rule E_7 is applied. Likewise, the same rule can be used in the left part of the root node.
Rossignac and Voelcker [17] reported that the use of the positive form in the CSG representation, could be clearly detect the redundancy part and the null-object. The positive form uses the intersection, union and complement operators, which represent the negation of given operator or primitive. Any CSG tree can be transformed into positive form by

$$\begin{array}{llll} PF_1: & x - y & \longrightarrow & x \cap \overline{y} \\ PF_2: & \overline{x \cup y} & \longrightarrow & \overline{x} \cap \overline{y} \\ PF_3: & \overline{x \cap y} & \longrightarrow & \overline{x} \cup \overline{y} \\ PF_4: & \overline{\overline{x}} & \longrightarrow & x \end{array}$$

Table 3.3: Transformations of Positive Form[17]

a pre-order traversal of the tree applying, which is illustrated in table 3.3. The rule PF_1 replaces the subtraction operators by using the intersection of the negated second part. The rules PF_2, PF_3 are known as De Morgan's Laws, which exchange the intersection and union operators at negative internal nodes. This reformulation moves the negations downward to the leaf nodes (CSG primitive) and the resulting CSG tree contains intersection and union operators in the internal nodes. The negation operator represents the unbounded regular set, which makes sense when combined with other CSG parts. However, the transformed positive form is equal to the original CSG tree representation. Fig. **??** shows a example of positive form, The transformation processes in the following steps:



(a) CSG tree example (b) CSG positive form

Figure 3.5: CSG Positive Reformation. The CSG tree is converted into its positive form.

$$\begin{array}{lllr} (a \cup b) - (c \cap d) & \longrightarrow & (a \cup b) \cap \overline{(c \cap d)} & (PF_1) \\ & \longrightarrow & (a \cup b) \cap (\overline{c} \cup \overline{d}) & (PF_3) \end{array}$$

The positive reformulation of CSG trees has been used in the CSG transformation. This reformulation can simplify the notation, keep structural similarity to the sum-of-products and avoide sum-of-products the potential exponential tree growth [16].

3.4 Tree Simplification

The tree pruning can reduce the size of the CSG tree and maximize the rendering performance. In this section, several pruning techniques to simplify the normalized CSG tree (sum-of-product) are introduced. One of these techniques is algebraic tree pruning, in which the transformation processing does not include the redundant parts or repeated primitives in the CSG tree. Because the user-defined CSG model usually contains the redundant parts that can be eliminated by algebraic tree pruning. The BV of a node can be used to detect the overlaps in its subtree. The section 3.4.2 describes the BV based CSG tree pruning technique.

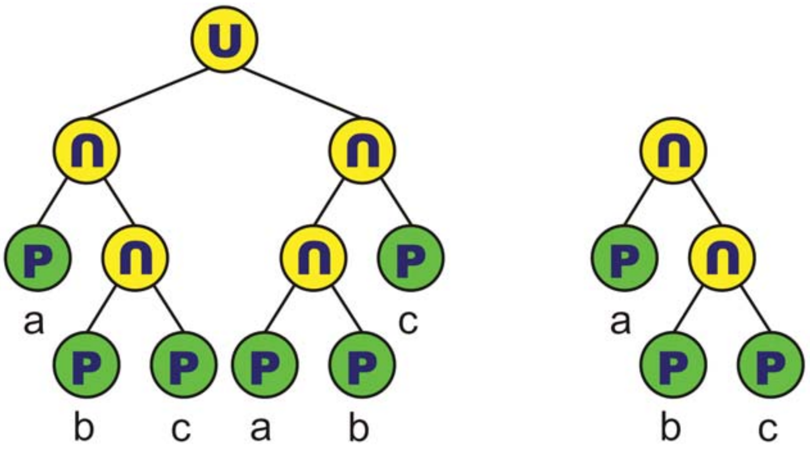
3.4.1 Algebraic Tree Pruning

Algebraic tree pruning, the simplest technique to simplify CSG trees. uses the CSG tree structure and works without any additional information about the contained CSG primitives. In this case, the laws of boolean algebra is used to simplify the redundant parts. Table 3.4 is the pruning rules of the Laws of Boolean algebra essentially for the simplification. The Boolean algebra prunes the CSG trees by a post-order traversal of the

T4 : Identity Law	(a)	$X \cap X$	$\longrightarrow$	X
	(b)	$X \cup X$	$\longrightarrow$	X
T7 : Redundance Law	(a)	$X \cap 0$	$\longrightarrow$	0
	(b)	$X \cup 0$	$\longrightarrow$	X
T8 : Redundance Law	(a)	$X \cap 1$	$\longrightarrow$	X
	(b)	$X \cup 1$	$\longrightarrow$	1
T9 : Redundance Law	(a)	$X \cap \overline{X}$	$\longrightarrow$	0
	(b)	$X \cup \overline{X}$	$\longrightarrow$	1

Table 3.4: Redundance Laws of Boolean Algebra. The symbol X denotes any primitive or CSG tree parts, 1 is the whole objects, 0 is the empty set. [21]

tree applying, in which the pruning processing starts at the leaf nodes and moves upward to the internal nodes. The algebraic pruning is used to check the duality inside each product of normalised expression. The two CSG parts are structurally different, shown in Fig. 3.6, therefore, the commutativity of the intersection or the union can be detected. In the following section, a typical encoding method to accurately identify the duality is described.



(a) Tree $(a \cap (b \cap c)) \cup ((a \cap b) \cap c)$ in the form $X \cup X$ from Table 3.4

(b) Equivalent CSG tree with the substitution $X \bigcup X \longrightarrow X$ is applied.

Figure 3.6: Algebraic CSG Tree Pruning

3.4.2 Bounding Volume Tree Pruning

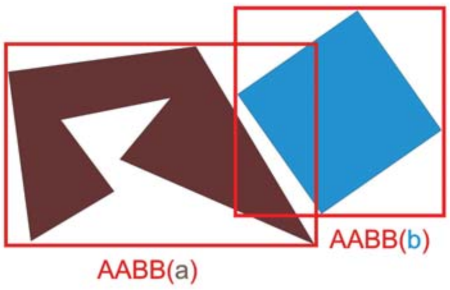
The BV can find out the relations between different parts in the scene graph. The CSG primitive has the BV which is the outer boundary of the 3D geometry data. The internal nodes are Boolean operators in the CSG tree. The BV of CSG operators are calculated from their child nodes and depends on the Boolean evaluation of the rules. Table 3.5 shows the computed BV of the Boolean operator. For example, A and B are arbitrary

$$\begin{aligned} Bound\,(x \cap y) &= Bound\,(Bound\,(x) \cap Bound\,(y)) \\ Bound\,(x - y) &= Bound\,(Bound\,(x) - Bound\,(y)) \\ Bound\,(x \cup y) &= Bound\,(Bound\,(x) \cup Bound\,(y)) \end{aligned}$$

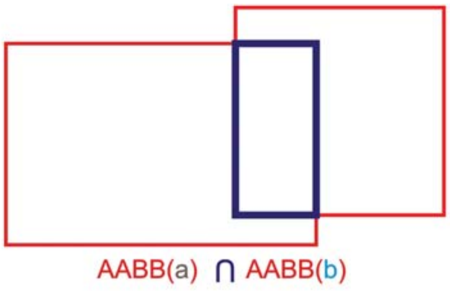
Table 3.5: CSG Operator Bounding Volume. [21] [24]

child nodes. The BV is updated in a bottom-up order, and hence, the BV tree pruning is applied in the post-order traversal of the tree. In the BV tree pruning, the empty bounding volume is removed, which indicates that the Boolean intersection and subtraction operator can be eliminated [24]:

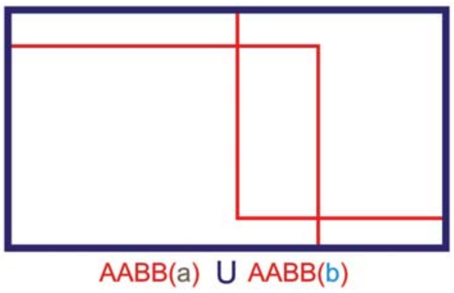
$$\begin{aligned} x \cap y &\longrightarrow \emptyset, &&\text{if Bound(A) does not intersect Bound(B)} \\ x - y &\longrightarrow x, &&\text{if Bound(A) does not intersect Bound(B)} \end{aligned}$$



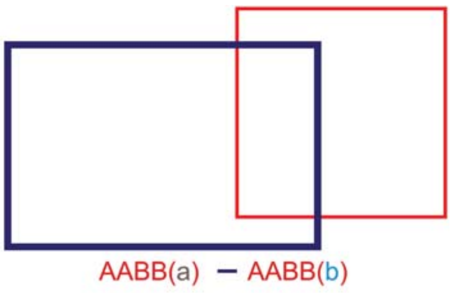
(a) Volume a and b with their BV



(b) Intersection BV



(c) Union BV



(d) Subtraction BV

Figure 3.7: Bounding Volume CSG Tree Pruning

In most cases, however, the BV of internal node increases or reduces related region by evaluating the Boolean operation. Because the Boolean operators result a non-empty BV during the pruning [21]. For instance, the bounding volume (AABBs) of two primitives are shown in Fig. 3.7(a). The BV for intersection, union and subtraction are shown in Figs. 3.7(b)-3.7(d), respectively. The volume is not intersected, but the BVs of the intersection and the subtraction are not empty, whereas the resulting volume is a empty. In this case, the BV is not fulfilled by correct evaluating the Boolean operations, a exact evaluation of data set in the volume is required.
The tree pruning techniques remove the redundant parts of the CSG primitives and are a generic pre-processing step for the CSG rendering algorithm.

3.5 CSG Rendering

The sum-of-product form can be used in the rendering pipeline to identify which primitives are rendered or clipped in the current scene. Dependent on chosen rendering algorithm, the CSG model can be completely represented. In the next chapter, we focus on the volume based clipping methods and the ray casting multiple volume rendering to visualize the necessary wanted parts of volume.

Chapter 4

Volume Clipping and Ray-Casting

Contents

In this chapter we explain solutions of the clipping and visibility problems related to the volume based rendering algorithms. The volume data set consists of voxels and transfer function. The voxels are the points in 3D space, along with an interpolation scheme that fills the in-between space. The voxels serve as grid points of a uniform grid, which is connected by edges, forming hexahedral (i.e., cube-like) cells [9]. The 3D uniform

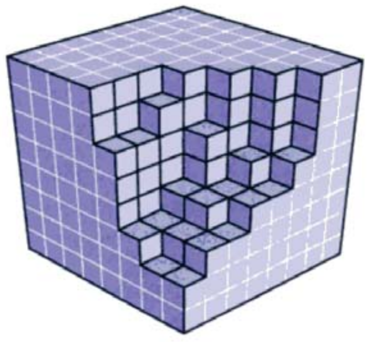


Figure 4.1: Example of Uniform Grids: 3D uniform grid with cuboid cells [9].

grid is shown in Fig. 4.1. The intensity and opacity values are assigned by evaluating the transfer function. Compared to the geometry object which has the explicit defined boundary, the volume can be only verified by the opacity inside the voxels. The CSG rendering can be regarded as a way to modify the visibility of region in multi-volume data sets. This consideration can be realised by the volume clipping concept, in which the original volumes are visible in defined selection volume. Fig. 4.2 shows design of the

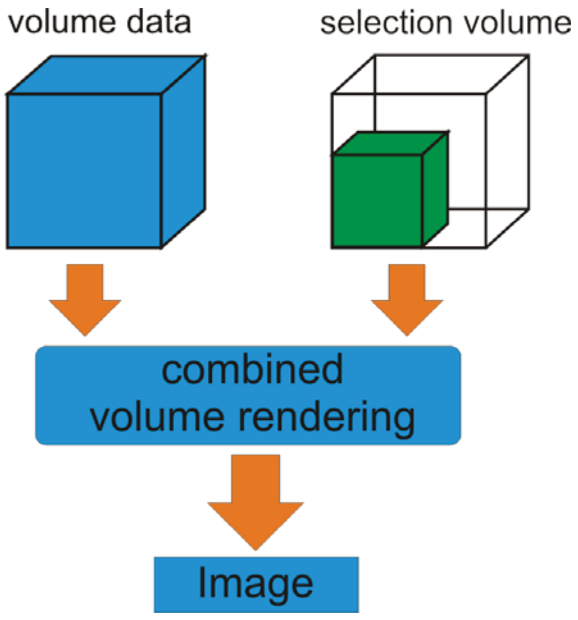


Figure 4.2: Conceptual Design of Volume Clipping. The volume data set is only visible in the selected region. [11]

volume clipping. The selection volume controls the visibility by modifying the original transfer function associated with the volume data set [11]. The selected volume parts are visible, which can be drawn by the volume rendering techniques. The remained parts are clipped during rendering.
Compared to another rendering algorithm, the ray-casting rendering algorithm is widely used in the volume rendering techniques, because of its flexibility in producing high quality images of volume data set. The algorithm is an image-space method and is supported by the depth-value of ray intersection. In the following section 4.1, a short introduction of the ray-casting rendering algorithm is given. The volume clipping technique is discribed in the subsequent Section 4.2.

4.1 Ray-Casting Rendering

The ray-casting is a typical image-order algorithm and also supports the depth cueing during the rendering. Therefore, using a ray casting technique to perform volume clipping is a common way, which the depth structure can be reused in both cases. Fig. 4.3 shows the rendering pipeline for ray casting in a single volume. For each pixel in the screen

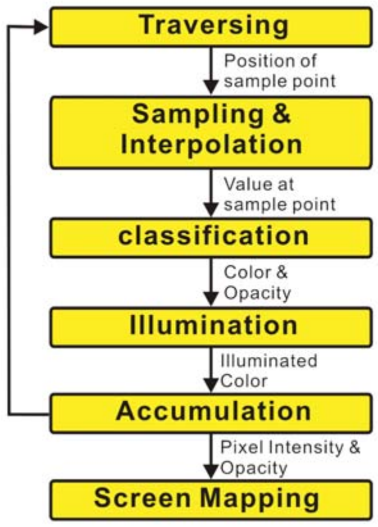
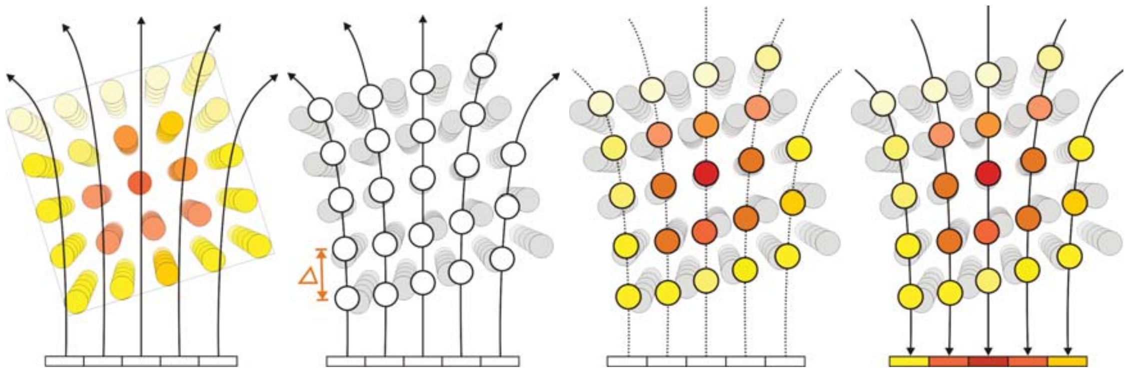


Figure 4.3: Rendering Pipeline of Ray Casting in a Single Volume.[18]

view, a ray is generated from the view point in the view direction. The ray intersects the volume and starts traversing. The traversing along the ray in the volume is sampled to points (Sampling & Interpolation) and each sampling point is assigned by a color and opacity evaluated from the transfer functions (Classification). Depending on the illumination model, the illuminated color is calculated and accumulated. A new sampling point is generated based on the given step size and this process goes iteratively forward until the end of the volume that the ray reaches. The accumulated result assigns to the corresponding display pixel (Screen Mapping). Fig. 4.4 illustrates the ray-casting example in a single volume.

When the ray intersects the volume, the intensity is calculated directly by the volume rendering integral. The emission-absorption optical model illustrates the intensity by integrating along the direction of ray flow from the entry point of the volume (the intersection point) $s = s_0$ to the end point $s = D$. The relevant equation can be approximately expressed [10]:

$$I(D) = \sum_{i=0}^{n} C_i \prod_{j=i+1}^{n} T_j \quad \texttt{with} \quad c_0 = I(s_0) \tag{4.1}$$



(a) Ray Casting (b) Sampling & Interpolation (c) Classification & Illumination (d) Accumulation & Screen Mapping

Figure 4.4: Ray Casting in a Single Volume (Δ is the sampling distance, and the same as the step size in the volume).

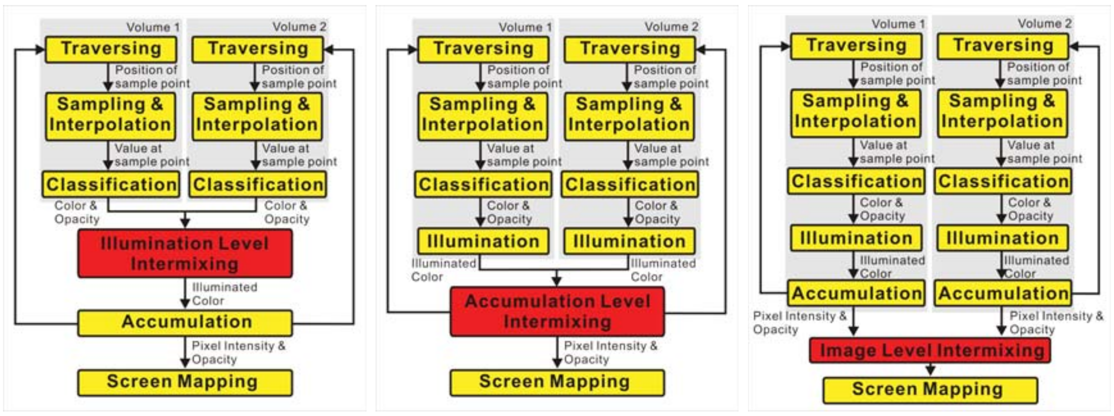
Where $C_i \approx q(s_i)\,\Delta x$ is the color contribution to the i-th sampling point, T_i is the transparency of the i-th sampling point, $\Delta x = (D - s_0)/n$ is the distance between the entry point and sampling point, n is the maximal number of sampling points. Normally, the sampling points do not match the voxels positions exactly; the corresponding interpolation is used to calculate the correct value. This approximation will be more accurate if the sampling rate is increased and the corresponding sampling distance is reduced.
In the rendering, the accumulation usually starts from the view point to the volume (front-to-back compositing) during the ray traversing. The front-to-back iteration equations are [10]:

$$\hat{C}_i = \hat{C}_{i+1} + \hat{T}_{i+1} C_i \tag{4.2}$$

$$\hat{T}_i = \hat{T}_{i+1}(1 - \alpha_i) \tag{4.3}$$

The iteration starts at the first sampling point $i = n$ and ends at $i = 0$ in the volume. The color $\hat{C}_i$ of the current iteration step increases by adding the previous color $\hat{C}_{i+1}$, and further by adding the multiplication of the source color C_i of the current iteration step and the transparency degree $\hat{T}_{i+1}$. α_i is the opacity of current iteration step and is defined by the transfer function. Compared to the back-to-front compositing schema, the advantage of front-to-back compositing is that the algorithm can be optimized by early ray termination [10].
In the ray casting based multi-volume rendering, the volume data can be intersected with

each other. There are many ways to visualize several volume data in the intersected regions, it depends on different objects contribution. Cai et al. [3] classified the volume intermixing into three levels (see Fig. 4.5), which are subsequently described in detail. The *Image level* intermixing is the simplest way, which merges two rendering images inde-



(a) Illumination Level Intermixing

(b) Accumulation Level Intermixing

(c) Image Level Intermixing

Figure 4.5: Rendering pipelines for the three mixing strategies [3] [18].

pendently by using pixel's intensity, opacity and the depth value. The *Accumulation level* intermixing combines the volumes' visual contributions step by step along the ray. The sampling point has minimal and equal step size in all volume data sets, and is mapped by the intensity and opacity from individual transfer functions. The resulting intensities and opacities are merged to be a single intensity and opacity of the sampling point, which is accumulated and assigned by the display pixel. The *Illumination level* intermixing calculates the color and opacity directly by using multi-volume illumination model.

The illumination level intermixing is only for the special application like X-ray image. The mixing of different modalities, such as CT and MRI, is difficult because the ranges of the sampling value are different from each other [18]. The accumulation level intermixing allows the application of independent transfer functions and shading style to different volumes, and supports depth cueing of the volumes compared to the image level intermixing [15]. Fig. 4.6 illustrates the accumulation level intermixing example of the multi-volume ray casting.

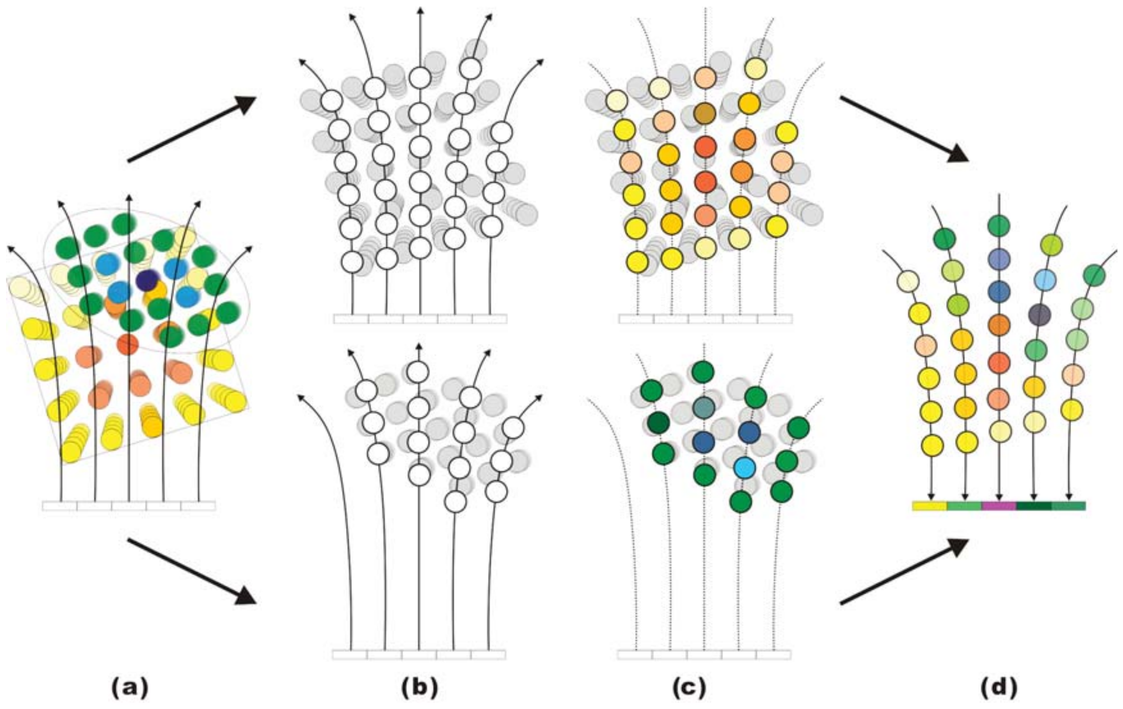


Figure 4.6: Ray Casting in Multi Volumes Accumulation Level Intermixing in Multi-Volume: (a) The ray traverses in two different volumes. (b) The sampling points are generated in both volumes by minimal step size. (c) The sampling points calculate the intensities and opacities from the volume data by interpolation. (d) The sampling points mix their intensity and opacity values, the illuminated color is calculated. The sampling points accumulate the color distribution and then is assigned to the pixel color.

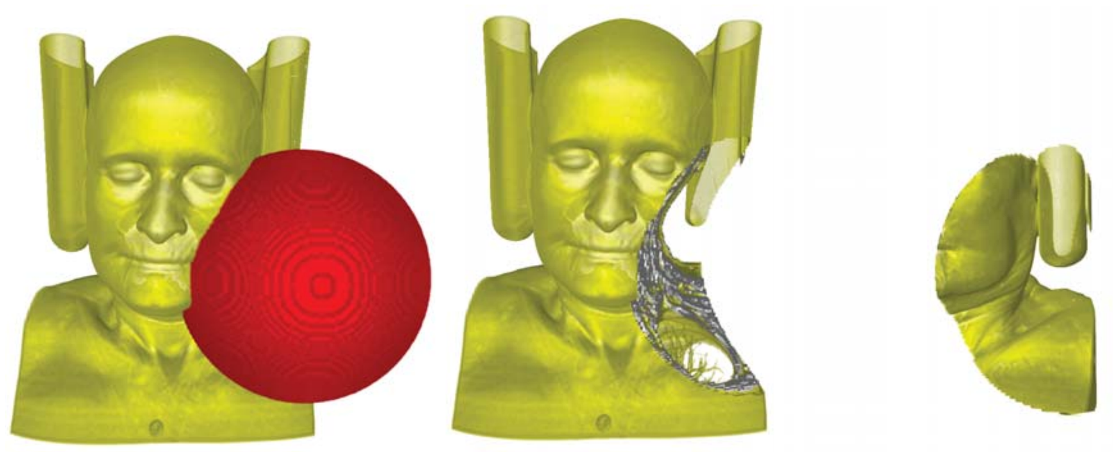
4.2 Volume Clipping Algorithms

The selection volume (see Fig. 4.2) can be used to choose visible and invisible regions. Therefore, the selection volume is applied as a binary decision [11]. A visible region is rendered according to the original transfer functions, whereas an invisible region is transparent. The selection volume is called the clipping volume. The boundary of the clipping volume is defined as clipped geometry, which is the BRep of the clipping volume. In general, the selection volume is mapped to binary decision, which is denoted as [11]:

$$\phi_S : \quad R^3 \rightarrow \{true, false\} \tag{4.4}$$

Based on the Boolean values, there are only two possible rendering states to apply the volume data with a well-defined clipping volume [22]:

- *volume probing*: the outside of the clipped geometry is clipped and the volume inside the clipped geometry remains unchanged.
- *volume cutting*: the inside of the clipped geometry is clipped and the volume outside the clipped geometry remains unchanged.



(a) volumes: head with sphere (b) volume cutting: remain head (c) volume probing: cut head

Figure 4.7: Examples of volume cutting and volume probing.

Fig. 4.7 gives an example of these two concepts. The boundary of red sphere is the clipping geometry and the head parts inside or outside the sphere are clipped, as shown in Figs. 4.7(b) and 4.7(c). There are many ways to render the volume data with the clipping volume. Weiskopf et al. [22] divided this volume clipping technique into two classes: the voxelized clipping approach and the depth-based clipping approach. In the following sections, both approaches are explained in details.

4.2.1 Voxelized Clipping Approach

The voxelized representation of the selection volume stores the visibility information in a second volume whose voxels provide the clipping information. During rendering, the calculation of the visibility is applied to the selection volume and the original volume. The

selection volume is stored in the binary values: an entry is set to 1 if the corresponding voxel is visible and is set to 0 if the voxel is clipped. The voxelized clipping is based on a surface representation of the selection volume. The voxelization of the surface and of the interior of the clipping object is required.
The ray-casting is assumed to be set in the volume rendering. During the rendering in the volumes, the original voxels are assigned by the intensity and opacity values evaluated from the transfer function. These evaluated voxels are modified by multiplying the value of the selection volume in the fragment process step. The voxels, which have been multiplied by zero, are clipped. The corresponding fragments can be removed by the opacity test [11].
One advantage is that this approach supports arbitrary clipping objects (i.e. convex and concave) and both the surface-oriented and volumetric clipping. Another advantage is that the volume clipping is feasible with older graphics hardware by using the stencil buffer and stencil test to cut away clipped fragments. However, this approach needs large memory for the voxelized selection volume and additional texture-access operations to read selection volume during volume rendering. The re-voxelization of the clipping object is very time-consuming [11].

4.2.2 Depth-Based Clipping Approach

To avoid an explicit voxelization of BRep, depth-based volume clipping only uses the BRep of the clipping object. Thus, the polygon mesh of the BRep is required. Fig. 4.8

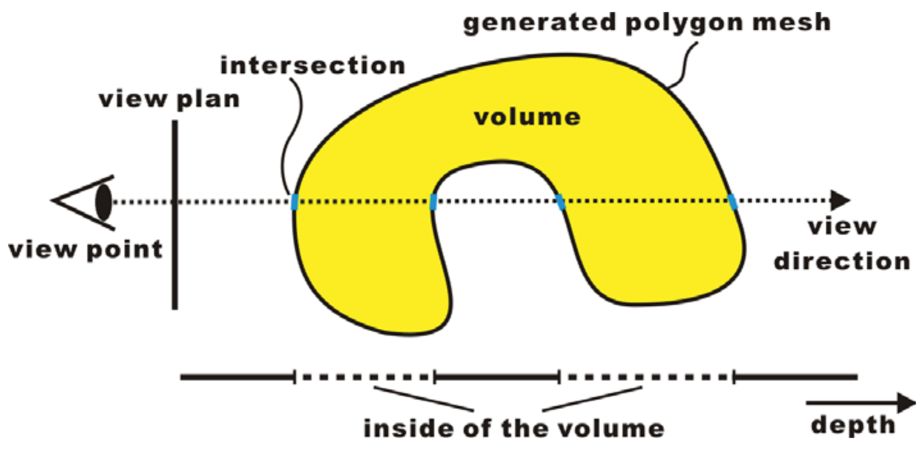


Figure 4.8: intersected Ray Segments; intervals on the depth axis that are inside the volume [22].

shows the concept of depth-based volume clipping related to depth-structure of the clipping geometry [22]. A ray is traversed from the view point to the volume. All depth values of intersection between ray and volume is stored in the depth structure. The ray is split by intersected depth values into several ray segment intervals, which are bounded by the intersection of the volume. Therefore, the ray segment intervals can be classified as the interior and exterior parts of the volume. The two-manifold property which points out that the boundary of the volume is closed and non-self intersected, ensuring correct classification of the intervals. This classification is allowed to separately deal with the ray traversing in the different volumes. Based on Equation (4.4), the volume clipping is converted from 3D volume region to 1D decision along the ray direction. Then, all intersected polygons are collected in the depth structure that is available for the corresponding pixel. During volume rendering, the depth structure can determine whether the fragment should be rendered or clipped.
In general, the depth-based clipping consists of two essential steps [11]:

1. the construction of the depth structure.

2. the depth-based clipping in volume rendering.

We explain the two possibilities of the depth structure construction. One is the *ray casting* method and other one is the *polygon rasterization* technique. The depth structure is stored on a per-pixel basis and dependent on the view point; this is a pixel-orientated mechanism. The *ray casting* based volume rendering is suitable for computing the depth structure immediately (see section 2.2). The construction of the depth structure needs multi-pass rendering. The rendering cost rises with an increased depth complexity [11].
Polygon rasterization represents the clipping volume by the polygonal mesh. All polygons are transformed into the screen space and projected onto the viewing plane. Kainz et al. [7] implemented an efficient polygon rasterized method using the coverage mask on the projected polygons of voxels in screen space, and encoded all intersected polygons on each pixel as a list. We describe this technique in the next chapter in detail.
For a single depth layer (i.e. a depth plane), the z-buffer is used to represent the depth-structure and the depth test is applied to remove the fragments. For the convex volume, the 2D texture with high-resolution texture format and three rendering passes are required. For examples, the front/back-face culling is used in the first two rendering passes, the remaining segment is rendered in the third pass. For multiple convex volume clipping

or concave volume[*] clippings, a lager number of depth structure and multiple rendering passes are required.

Compared to voxelized clipping approach, the depth based approach is an image-space algorithm. The depth structure will be rebuilt if the view point changes. The advantage of this approach is that its per-pixel accurate clipping leads to the high-quality result [11]. The rendering can update the depth-structure in each frame, the clipped objects can be translated, rotated and scaled. This clipping algorithm is suitable for real-time interactive manipulation.

The depth-based clipping is a modified volume rendering. The visible region is detected by evaluating sum-of-product form of normalized CSG model. Each pixel has its own depth sorted polygon list, which is used to construct the depth-structure by the volume clipping. We explain the volume clipping based CSG implementation in the next chapter elaborately.

[*]The concave volume should have the two-manifold boundary, which is self-closed, and does not contain any hole, the self-intersection is not allowed.

Chapter 5

CSG-enabled Polyhedral-Ray-Casting Rendering

Contents

The purpose of the CSG based rendering algorithm is to cut away the certain volume or part of volume in the rendered scene. To achieve this goal, we explain the depth-based volume clipping in ray-casting rendering, in which the rendered and clipped volumes are detected by sum-of-product expression of CSG model. As described in the previous chapter, this depth-based volume clipping requests two main steps in the rendering: i). Constructing the depth structures of the volume data set; ii). Cliping away the decided volume from the rendering system during the ray casting. The volume clipping algorithm and the ray-casting algorithm are per pixel-based (image-based) operations, which means the calculation of the depth structure is dependent on the view point. The voxels are organized in a regular 3D array to cover the volume data set. The outer boundary of voxels is the polygon representation of the volume. Kainz et al. [7] reported a new approach based on ray casting multi-volume data sets to calculate the depth structure. The applied polygon rasterization algorithm in this approach speeded up the computation of the depth structure, which is sorted in a front-to-back order and is used in our clipping algorithm. The Kainz's rendering system is implemented in CUDA, which provides the complete control over the rendering processing, indicating that all intermediate results of the rendering pipeline are available. Our CSG rendering algorithm is based on the Kainz's multi-volume rendering system, in which the sorted polygon list is available in our algorithm. Fig. 5.1

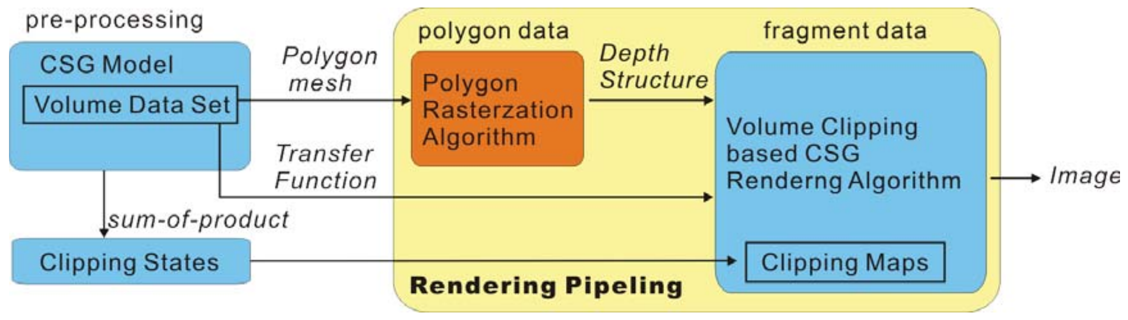


Figure 5.1: Flow Chart of the System.

shows the flow chart of the whole rendering system. In the pre-processing step, the CSG model is normalized in the sum-of-product form, which is encoded in the corresponding clipping states and is stored as the lookup texture in the rendering. Tthe polygon mesh, transfer function and corresponding location information of the input volume data sets are used in the rendering system. The depth-structure is calculated by the given polygon mesh using the special rasterization algorithm. The ray is cast into these depth-structure and split into several ray segment intervals, where the volume can be in. The ray is only traversed in these visible segment interval. Based on defined clipping map, the ray is rendered in the visible volume segments and is cut by the clipped volume segments. In the subsequent section, brief introduction of the polygon rasterization algorithm is provided and in particular, volume clipping rendering algorithm and its map encoding are explained in detail.

5.1 Polygon Rasterization Algorithm

Rendering system in a depth-based multi-volume ray casting was investigated by Kainz et al. [7]. This system supports the multi-volume data set, and rasterizates all outer boundary of the volumes into polygons. All polygons are sorted along the view direction in a front-to-back sequence. The rasterized depth structures are shown in Fig. 5.2. The outer boundary of voxels is used in the rasterization, whereby all polygons are transformed into

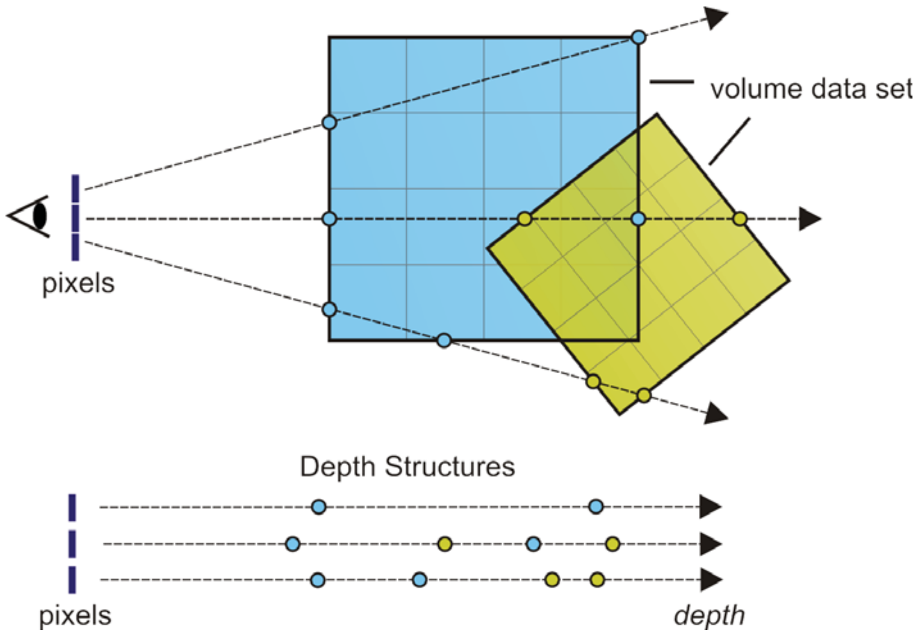


Figure 5.2: Depth Structures in Polygon Rasterization Algorithm.

the image-space, and are classified into several pixel blocks by the coverage mask, which also stores the polygon list in each pixel. A sort algorithm is used to arrange the polygons in each pixel. The polygon rasterization algorithm can be more efficiently implemented in modern GPGPUs. The depth structure in the volume clipping algorithm is exactly the sorted polygon list, which is used in our algorithm per-pixel based.

5.2 Volume clipping based CSG Rendering System

The sorted polygon list is used in the depth-based clipping algorithm. In this section, we focus on the corresponding volume clipping in the rendering. In multi-volume ray-casting rendering system, the ray is generated in each pixel, and cast into the volume. All volumes are rendered in the system, which calculates the color distribution of the corresponding transfer function in each voxel. The ray is sampled in each volume and the sampling points accumulate the illuminated color and opacity, which become a output color of final pixel. The depth-based volume clipping rendering is similar to the multi-volume ray-casting rendering. Yet, the difference is that the ray can be clipped completely in the decided depth interval. The clipped volume interval is not only dependent on the location of these two volumes, but also the used transfer function. This is primarily because the intensity and opacity of voxels are evaluated by transfer function. The opacity of the same voxel would be completely different if using another transfer function. We introduce two possible encoding methods for the volume state and the clipping state in the following two sections. The volume state is used to represent each volume-segment based depth interval. The clipping state is used to decide which volume is rendered or clipped in the current scene. The volume clipping algorithm is described in the section 5.2.2. The relevant functionality of the rendering pipeline is illustrated in the section 5.2.3.

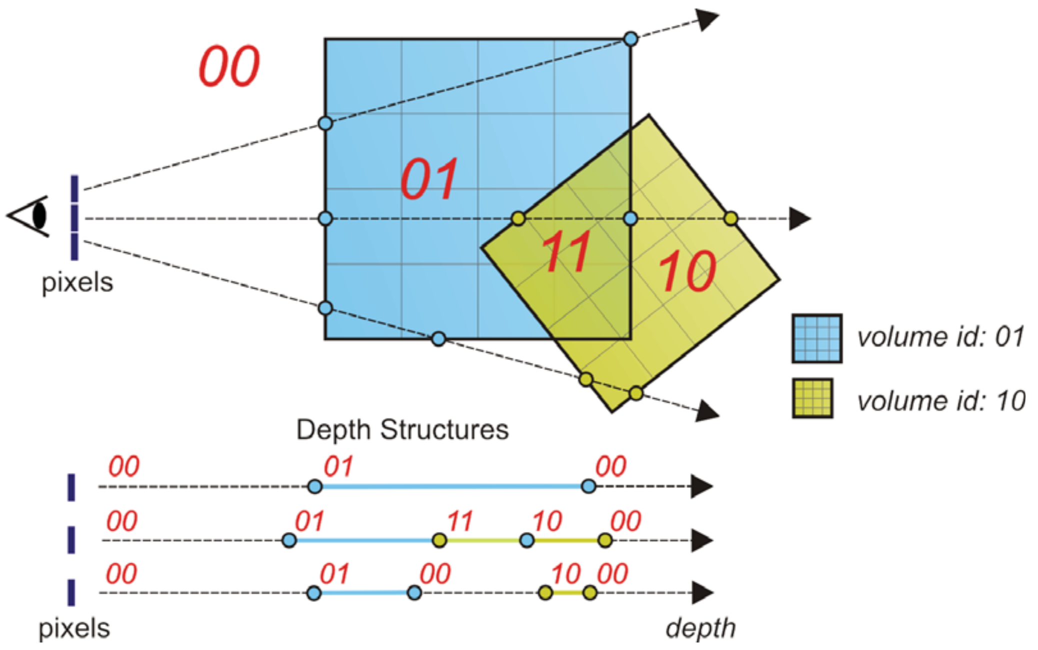
5.2.1 Volume Entry/Exit States

The polygons of the sorted depth-list split the ray into several segments along the ray. The depth interval between two neighboring entries in the sorted list defines a homogeneous segment of the ray related to intersected volume. Each segment interval gets own encoding to identify the volume state. The encoded volume provides an efficient way to describe the current volume states during the ray casting. We introduce a schema to maintain a list of bitflags. One flag, which has its own corresponding volume, is toggled each time when the ray passes a volume boundary [2]. An integer (limit to 32 volumes) or a 1D lookup texture can be used to record the entry or exit state of the volume along a ray traversal, which is associated with sorted polygons.
The bit flag *1* denotes that the ray enters into the volume, and *0* represents the exit. Each volume data set is registered in the system by an assigned volume-ID[*], which can retrieve the corresponding bit flag[†]. The Fig. 5.3 illustrates an example of the encoding of volume states:

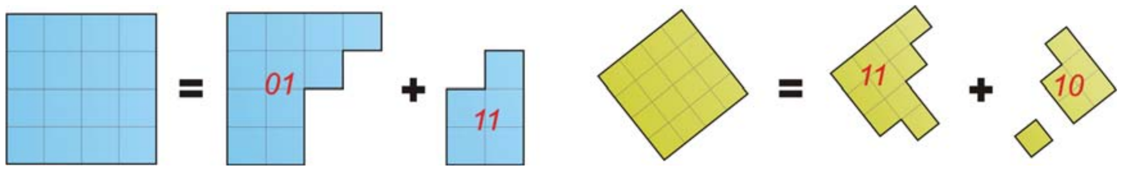
- *00*: No volume is available.
- *01*: Only the volume A is available.
- *10*: Only the volume B is available.
- *11*: Both the volume A and B are available.

The volume states change when the ray enters or leaves the volume. A volume-check occurs at the intersection point between the ray and volume. Hence, we can determine each volume state along the ray, and further, we know whether the ray is inside or outside the volume. In our case, the volume state changes when the ray comes across a new polygon. Each depth interval has the clear volume state.

[*]volume-ID: 1, 2, ...
[†]bit-1 for A, bit-2 for B, ...



(a) Volume entry/exit bit flags of volumes during the ray casting.



(b) Volume States. The voxels are split into several polygonal intervals.

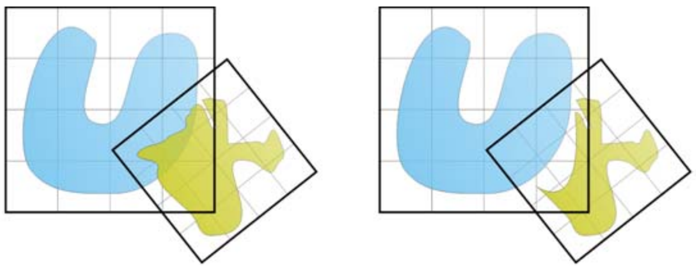
Figure 5.3: Volume Entry/Exit State.

5.2.2 Volume Clipping Algorithm

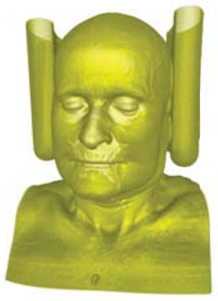
In this section, we focus on volume clipping algorithm that exploit the process of volume segments in the depth interval. A schema similar to the volume states is used to capture all clipping combinations of the appeared volume objects. The digits **1** and **0** denote that the corresponding volume is rendered and clipped in the current scene, respectively. One digit describes two possible rendering states, i.e. rendered or clipped of the volume. In the multi-volume case, the number of the digits is equal to the number of all appeared volumes, in which each digit is correlated with its volume-ID. Multiple digits can be merged into a multi-digit clipping state by using an AND-association. The following part shows the encoding of the two volumes, and lists all possible combinations (The volume A corresponds the lower digit of clipping state, B the higher):

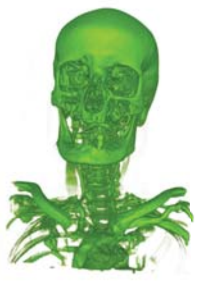
- **00**: The volume A and the volume B are clipped, i.e. a empty scene.
- **01**: The volume B is clipped away from the boundary of the volume A. A without B. (math. notation: $A - B$)
- **10**: The volume A is clipped away from the boundary of the volume B. B without A. (math. notation: $B - A$)
- **11**: The intersection of the volume A and volume B is rendered, the remaining part is clipped. (math. notation: $A \cap B$)

Based on four possible combinations of these two digits, we use an OR-association (+) to merge the several combinations of the clipping states. The term $A \cup B$, for example, illustrates a collection of **10 + 01 + 11**. Thus, the clipping state represents all possible combinations of the volumes. Note that, compared to the volume state which is affected by the voxels, the clipping state is based on the volume, whose intensity and opacity are evaluated by the transfer function in the voxels. As we mentioned in the previous section, the results of volume clipping could be completely different by using another transfer functions. Therefore, the volume clipping is dependent on the relation of two volumes, and also, theirs appearance. As shown in Fig. 5.4(a), the left and right volumes in yellow are produced by using different transfer functions. The results of intersection, union, subtraction between these two cases are also different. The Figs 5.4(b) and 5.4(c) show the same volume data set using different transfer functions.
Based on the polygonal depth interval, the volume clipping solution is that we add the rendered volumes into the current scene, by clipping away the decided parts in all intersected polygonal regions, which are defined as all intersected intervals between the rendered and the clipped volumes. All polygonal regions are limited by the bounding of voxels. Thus, the correlations between the clipping states and volume states can be established. As shown in Fig. 5.5, the clipping state **01** indicates that the CSG subtraction $(A - B)$ in the current scene. The volume A is a union of the part with volume state *01* and the part with *11*. A and B intersect in the segment interval with the volume state *11*. The clipping state **01** stands for the volume part A with volume state *01* and the volume part A with volume state *11* by substracting the volume B. The volume clipping algorithm is used in this example, and is described explicitly in the table 5.1. All regions of the volume state can be classified into three domains in our algorithm:



(a) Voxels using different transfer functions





(b) skin (c) head bone

Figure 5.4: Voxels with Different Transfer Functions. Both volumes (b, c) are the same data set, but use the different transfer functions.

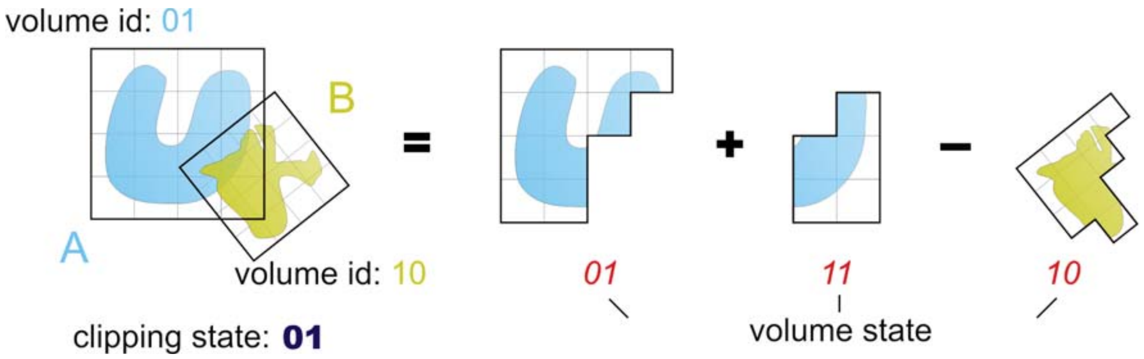


Figure 5.5: The interpretation of clipping state by volume states. The volume A subtracts volume B.

```
Step for calculation each clipping state:

detect the rendered volume (the volume with the digit 1 in the
clipping state) in each polygonal segment interval (volume state).

   *) if only one volume is found, the segment interval of the volume
     is observed.

   *) if several volumes are found, the intersected segments interval of
     these volumes are observed.

in each polygonal segment interval, detect the intersected regions between
clipped volumes (the volume with the digit 0 in clipping state)
and the observed segments of the step 1. The observed segments are then
cut away by the clipped volume at each intersected region.
```

Table 5.1: Depth-based Polygonal Volume Clipping Algorithm

i. The regions, with only rendered volumes: all volume segments are rendered in these regions (volume state *01* for clipping state **01** in the Fig. 5.5).

ii. The regions, without any volume: All volume segments are clipped in these regions (volume state *10* for clipping state **01** in the Fig. 5.5).

iii. The regions, with both rendered and clipped volumes: The rendered volume segments are cut away from the clipped volumes (volume state *11* for clipping state **01** in the Fig. 5.5).

The Fig. 5.6 interprets the clipping states of two volumes, the Boolean intersection and subtraction operations are only effected in the intersected polygonal interval. In this example, a combination of **10**, **01** and **11** stands for the whole volume, presented as $A \cup B$. In multi-volume case, we use the term $2^n - 1$ to indicate the number of the clipping states, where n means the number of volumes. The clipping state is the evaluation of the volume clipping in our depth-based volume clipping algorithm. The clipping state can be used to represent each CSG expression.

5.2.3 Volume Clipping based Ray Casting Rendering Pipeline

This section focuses on the implementation of the algorithm of the ray-casting rendering pipeline. In the previous chapter, the Boolean intersection, subtraction and union can be

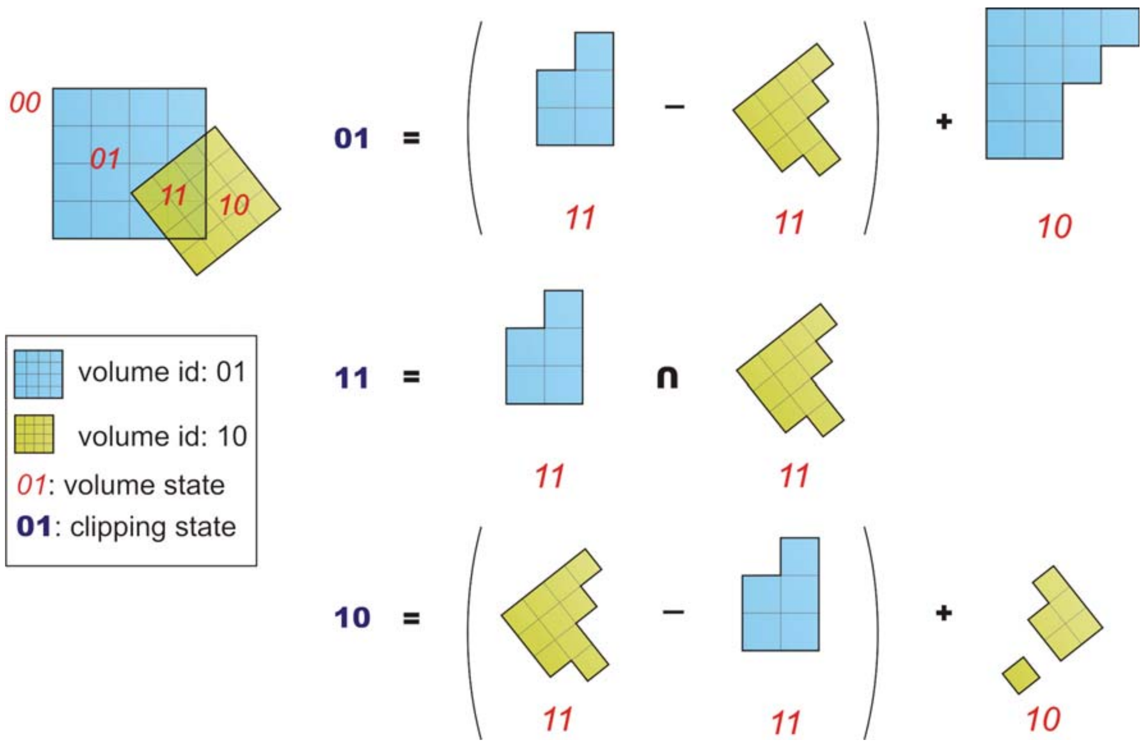


Figure 5.6: Interpreting the Clipping States into Volume States.

described by the clipping states. The CSG model is converted into the sum-of-product form, which is the collection of the intersection and the subtraction. It means that the clipping states can represent the sum-of-product form in the rendering. The volume clipping based ray-casting rendering pipeline is illustrated in Fig. 5.7. This rendering pipeline is based on Kainz's [7] multi-volume ray-casting rendering system, in which all yellow blocks in the system provide the accumulation level multi-volume rendering. The ray is sampled into sampling points during the access to the volume. Each sampling point is assigned a intensity and opacity values, which are evaluated by the corresponding transfer function. The evaluated color distributions are combined by the sample factor in the intersected region. The equal step size in the intersected region is used for all volume segments. The grey blocks represent the volume clipping algorithm.

Each volume is registered in the rendering system and corresponds a digit-bit in clipping states, which are generated from the sum-of-product form. These clipping states controls each volumes in the whole scene. During the ray-casting, the ray split the whole scene into several segment interval by using the depth structure, which is per-pixel wise the

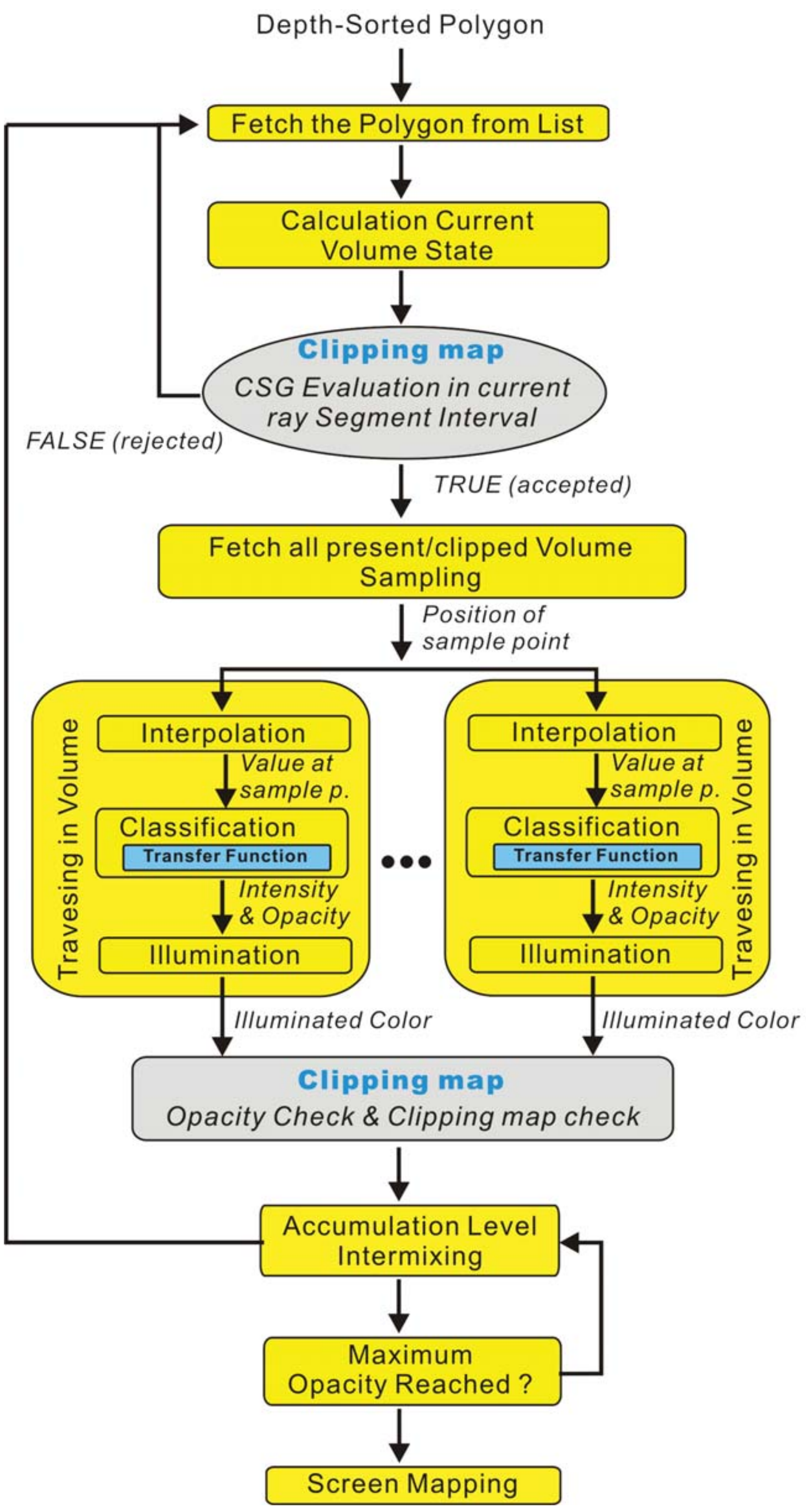
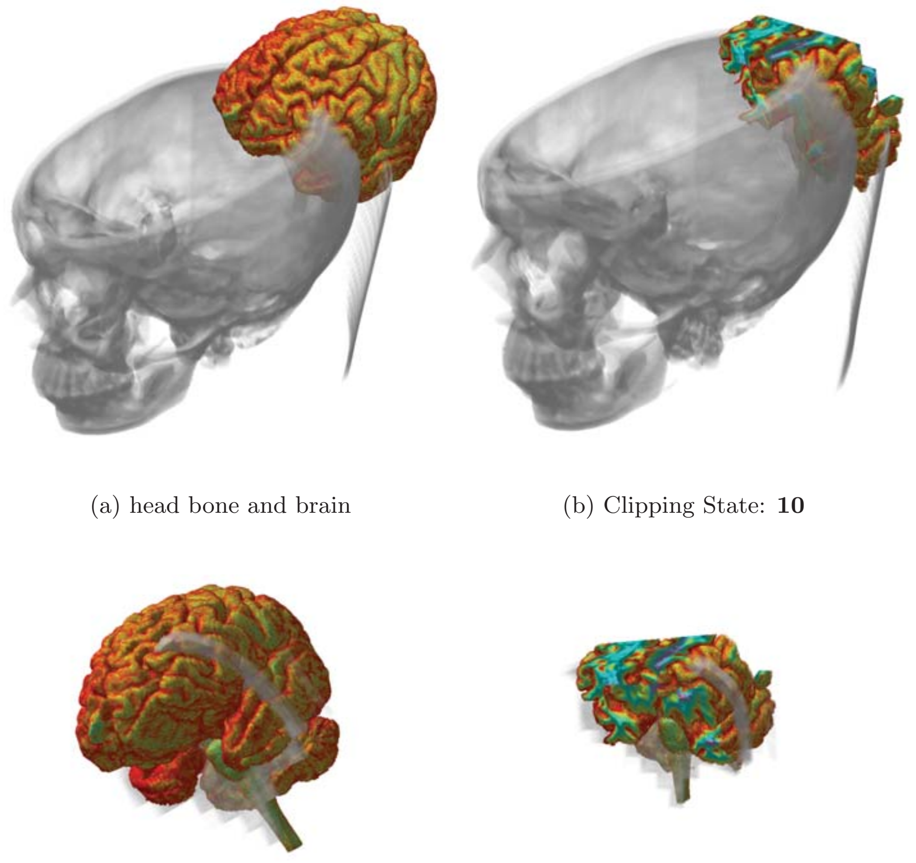


Figure 5.7: The Flow Chart of the Volume Clipping Rendering Algorithm.

sorted polygon list of all voxels in the current scene and is dependent of the view point. In each segment interval, it can exist one or more volumes, or parts of volumes. The volume clipping can be affected in each interval by using the corresponding clipping states, and deactivate the decided volumes or parts of volumes in each polygonal segment interval.
The clipping map shown in Fig. 5.7 is stored as a texture in the rendering system. At each polygonal segment interval during the ray-casting, the rendering checks the clipping map, to detect the rendered volumes which is denoted as the **1** in the clipping states and the clipped volumes which is denoted as the **0**. The segment interval, which only contains the clipped volumes, is rejected by the rendering system. Because these volumes or parts of volumes are invisible in the current scene. The segment interval, which contains at least one rendered volume and arbitrary clipped volumes are accepted by the system and further enter into our rendering pipeline. The accepted/rejected check occurs in each polygonal segment interval before the traversing.
The current segment interval is skipped and remains transparent when the rendering system is rejected. In our clipping algorithm, the rejected interval is exactly the second case (the regions which no volumes are rendered) as shown in section 5.2.2. The accepted volume segments, which have at least one rendered volume in the current segment interval (For example, see the first and third cases in section of section 5.2.2). Fig. 5.8 illustrates an example of the intermediate results of the rendering pipeline. The clipping state **10** in Fig. 5.8(b) indicates the sub-association ($head - brain$). The part of head bone without the brain is shown in the segment interval by the volume state *10*, and both volume segments are rendered in the intersected region. The interval *01* only contains the clipped brain and is rejected by the check of clipping map of the rendering. The segment interval has the voxel form as the polygon in the depth structure is split by the voxels data. In the intermediate result, we can also see the square volume, which is polygonal evaluated voxel data. The clipping state **01** in Fig. 5.8(c) keeps all parts of brain volume and also the clipped part of head bone in the intersected region. The clipping state **11** denotes the and-Association in $head \cap brain$, where the volume segments are only shown in the intersected region (see Fig. 5.8(d)).
Subsequently, the rendering pipeline is to remove the false parts from the intermediate result. The yellow blocks behind the accepted volumes are the classic ray-casting multi-volume rendering, in which the illuminated colors are mixed in the accumulate level (see section 4.1). All rendered and clipped volume segments are sampled in the equivalent step size at the intersected region. Based on the illuminated color, additional test is performed



(a) head bone and brain (b) Clipping State: **10**

(c) Clipping State: **01** (d) Clipping State: **11**

Figure 5.8: The Intermediate Result of Rendering Pipeline with Clipping Map. The segment intervals, which only contains the clipped volumes, is rejected in the rendering system. The segment intervals, which have at least one rendered volume, is passed in the rendering pipeline. The head bone gets the volume id 01 and the brain gets 10.

at each sampling point to remove the unwanted parts. Further, we extend the range of the problem by assuming a segment interval, in which M volume segments are rendered, and N volume segments are cut away from the rendered volumes. The solutions of the problem can be described as follows:

$$(P_1 \cap ... \cap P_M) - C_1 - ... - C_N \tag{5.1}$$

where P_i is the rendered volume segment with $i \in [1, M]$ and C_j is the clipped volume segment with $j \in [1, N]$. The intersected segments of all rendered volume should be calculated and each clipped segment is then cut from intersected segments.
Fig. 5.8(d) shows an incorrect example of the intersection between both volumes in the intersected region. The segments, in which only one volume is rendered, is the wrong result in the clipping state **11**. The correct result should be that both parts are visible in the intersected region of volumes. Based on above results, we classify each evaluated sampling point into the following two types, which are mainly dependent on the evaluated opacity values of the illumination model:

i. The sampling point, in which the opacity values of all rendered volume segments are not zero.

ii. The sampling point, in which at least one opacity value of rendered volume segments is zero.

The type (i) provides the correct result in the intersected region, the opacity value of both volumes is non-zero. All remaining parts (i.e. type (ii)) of the rendered segments are clipped, in which their intensity and opacity values are set to be zero. Based on the opaque property of the intersected volume segments, the correct part of intersection is found. The correct result of intersection indicates that each corresponding rendered volume should be opaque in the intersected region. At each sampling point, we perform an opacity test to obtain correct intersected region in the rendered volumes. The quality of the result is dependent on the sampling rate. Fig. 5.9 shows an improved example in

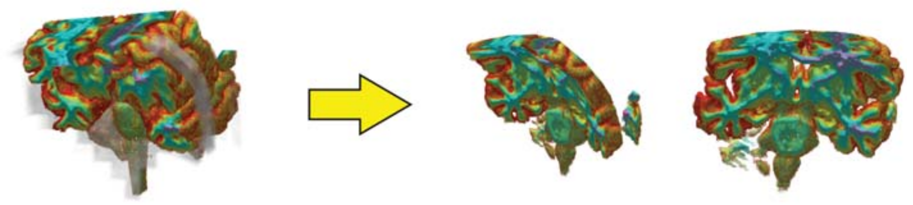


Figure 5.9: An Example of the Intersected Volume Segment. The opaque property of the intersected volume segments are correct, all remaining segments of the volumes should be clipped.

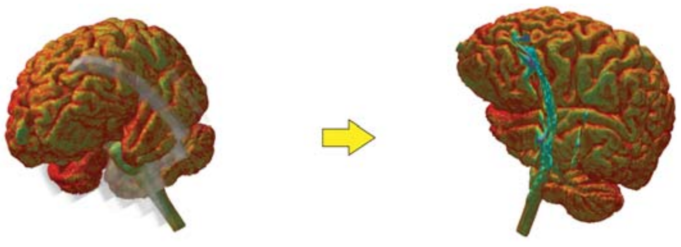
the intersected segment interval. The two intersected parts on the right side are correct results of intersection between head bone and brain volumes in Fig. 5.8(d).
In the subtraction operation, the clipped volume, which the corresponding digit in the

clipping state is **0**, is removed from the current scene in each polygonal segment interval. It means that, all opaque parts of the clipped volume should be reset to be transparent by a intersection with the rendered volume. Fig. 5.8(c) displays an example of the removal of a clipped segment from a rendered segment (clipping state: **01**). Each illuminated sampling point in the intersected region, including the rendered and the clipped segments, can be classified into following two types:

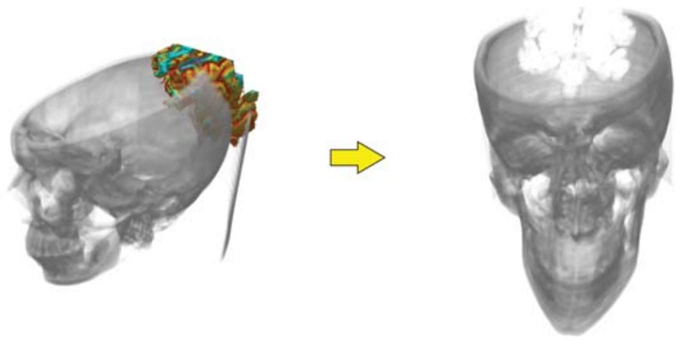
i. The sampling point, in which the opacity values of (intersected) rendered volume segment is not zero and at least one opacity value of clipped volume segment is not zero.

ii. The sampling point, in which the opacity values of (intersected) rendered volume segment is not zero and the opacity values of all clipped volume segments are zero.

The type (i) gives the correct case of clipped volume segment; the corresponding sampling points are cut away. The type (ii) is the rendered volume segment by excluding all clipped segment; the rendered volume segments are visible after evaluation. At each sampling point, a non-zero opacity of clipping segment should be cut away from the visible parts. Fig. 5.10 shows the correct results for the clipped volume segments. The intersected segments between clipped and rendered volume is cut away, and also the voxel formed redundant part of the clipped volume. In the volume clipping operation, zero-opacity test of the rendered volume segments and positive-opacity test of the clipped volume segments are exactly the wrong parts in the immediate results. A removal of both segments yield the correct results.
The opacity tests of volume occur at each sampling point in the polygonal segment interval and detect the volume existence. These tests work together with the clipping states, which can represent the CSG operations. The converted sum-of-product form can be completely described by using these two tests, indicating that the CSG operations can be described though the opacity tests and clipping state tests. In order to get the correct result, both tests should be check in each sampling point. The clipping state is stored as texture in the shared memory to get the shorter access time. Therefore, the intensity and opacity values of the sampling points are accumulated. After reaching the end of the segment interval, the rendering pipeline fetches the new polygon in the list and the pipeline starts again by checking the clipping map (see Fig. 5.7), until the end of depth structure or the saturated opacity of the sampling points.



(a) Improve the clipped brain



(b) Improve the clipped head

Figure 5.10: An Example of the Clipped Volume Segment. The clipped parts of the rendered segments are clipped.

5.2.4 Geometry Clipping

The volume clipping algorithm also supports the geometry clipping. In addition, there are three different combinations of the input data applied in the rendering, which are volume-volume, geometry-geometry and geometry-volume, respectively. The volume-volume combination has been mentioned in the previous section. Here, we discuss the clipping in the geometry-geometry and geometry-volume.

The geometry consists of outer boundary and inner volumetric region. The geometric boundary can be a complex polyhedral or concave surfaces, and the two-manifold property must be satisfied. The geometric boundary can be observed as the polygon data in the rendering pipeline and be used to construct the depth-structure. The inner volumetric region is represented as the volume segments in the ray casting rendering. Hence, our clipping algorithm can be used in both the geometry and the volume clippings. Compared to the volume clipping, the boundary of the geometry should also be calculated in the result by any geometry clipping. In the geometry-geometry case, the resulting geometric shape is closed, indicating that the geometry boundary around the volume segments is always rendered in all CSG operations. Fig. 5.11 shows examples of geometry-geometry clipping,

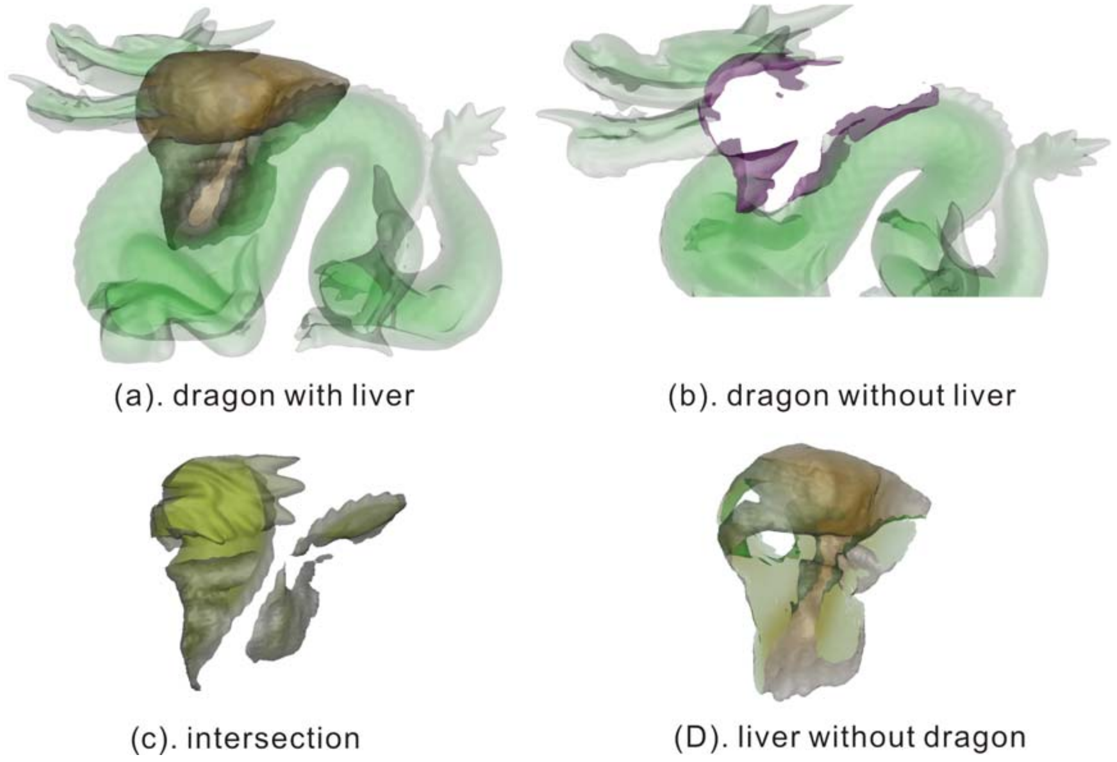
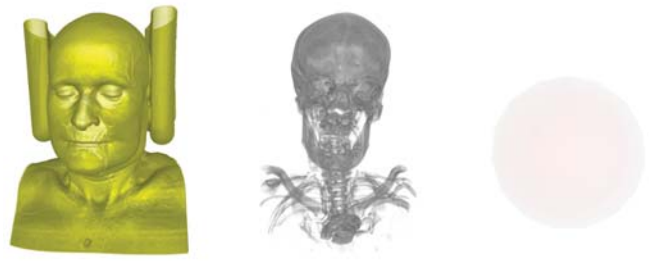


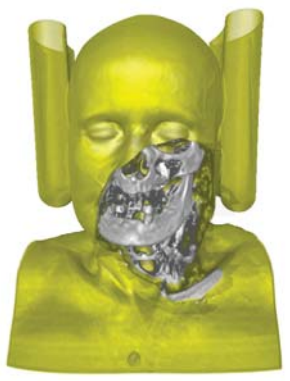
Figure 5.11: Example of the Geometry-Geometry Clipping. The geometry is closed in all result CSG shapes

where the related boundary of the clipped geometry is always drawn in the intersection and the subtraction. The intensity and opacity are also accumulated in the intersected regions.

In the geometry-volume case, we focus on the geometry clipping of the volume data set. The result of the clipping is neither a geometry nor a volume, but rather a mixture of the volumetric segment and the geometric boundary. The two-manifold property is defected. Usually the volume-geometry clippings are used in most medical image processing. For instance, a geometry sphere is overlapped by two volume data sets (see Figs. 5.12(b) and 5.16), we want clip the outer skin of the volume to detail the inner structure. The geometry clipping can be used to arriving this goal. The hidden data (The skin in examples) is removed by the corresponding subtraction operation. The geometry-volume clipping is widely used to manipulate and process the reconstructed volume data set.



(a) A volume combination of skin and bone, a volume of bone and a geometry sphere



(b) The merging of two volume data sets subtracting the geometry

Figure 5.12: Example of the Geometry-Volume Clipping. clipping state: **220+011**. sum-of-product: $((A \cap B) - C) \cup ((B \cap C) - A)$

5.3 The Improved Clipping State

In the intersected region, the clipping state is checked in each rendered and clipped volume segments at each sampling point. The run-time of fragment process grows with the increasing number of the clipping states. Rendering the whole scene of the union operation needs the permutation of digits in the the clipping state. There are three cases in two-digit clipping state, which can be calculated by using $2^n - 1$ in n-digit clipping state. However, such a clipping state encoding is not optimal in the rendering algorithm. Thereby an additional digit **2** is inserted to improve the performance efficiency. Different digits at the same location in both clipping states are replaced by the digit **2**, which indicates that the rendering system does not care the corresponding volume. The intensity and opacity of the volume can be simply merged into the rendered scene (OR-Association). The digit **2** can be written as:

$$\mathbf{dd2dd} = \mathbf{dd1dd} + \mathbf{dd0dd}$$

where d is the same digit-bit in all three terms. Likewise, multiple digits of **2** can be described as an OR-association of 2^n possible combinations, where n denotes the number of digit **2** in the clipping state.

$$\begin{aligned}
\ldots\mathbf{222}\ldots &\equiv \ldots\mathbf{122}\ldots+\ldots\mathbf{022}\ldots\\
&\equiv \ldots\mathbf{112}\ldots+\ldots\mathbf{102}\ldots+\ldots\mathbf{012}\ldots+\ldots\mathbf{002}\ldots\\
&\equiv \ldots\mathbf{111}\ldots+\ldots\mathbf{110}\ldots+\ldots\mathbf{101}\ldots+\ldots\mathbf{100}\ldots\\
&\quad\ \ldots\mathbf{011}\ldots+\ldots\mathbf{010}\ldots+\ldots\mathbf{001}\ldots+\ldots\mathbf{000}\ldots
\end{aligned}$$

The whole volume, which denotes the digit **2** in the clipping state, enters into the rendering pipeline. Visible parts of the intersected volume segments are merged into the corresponding rendered volume and are cut by the clipped volume. If the clipping state has either the digit **2** or **0**, the current scene is the combination of all visible parts of digit **2** by cutting away clipped volumes of digit **0**. Fig. 5.13 interprets the clipping states

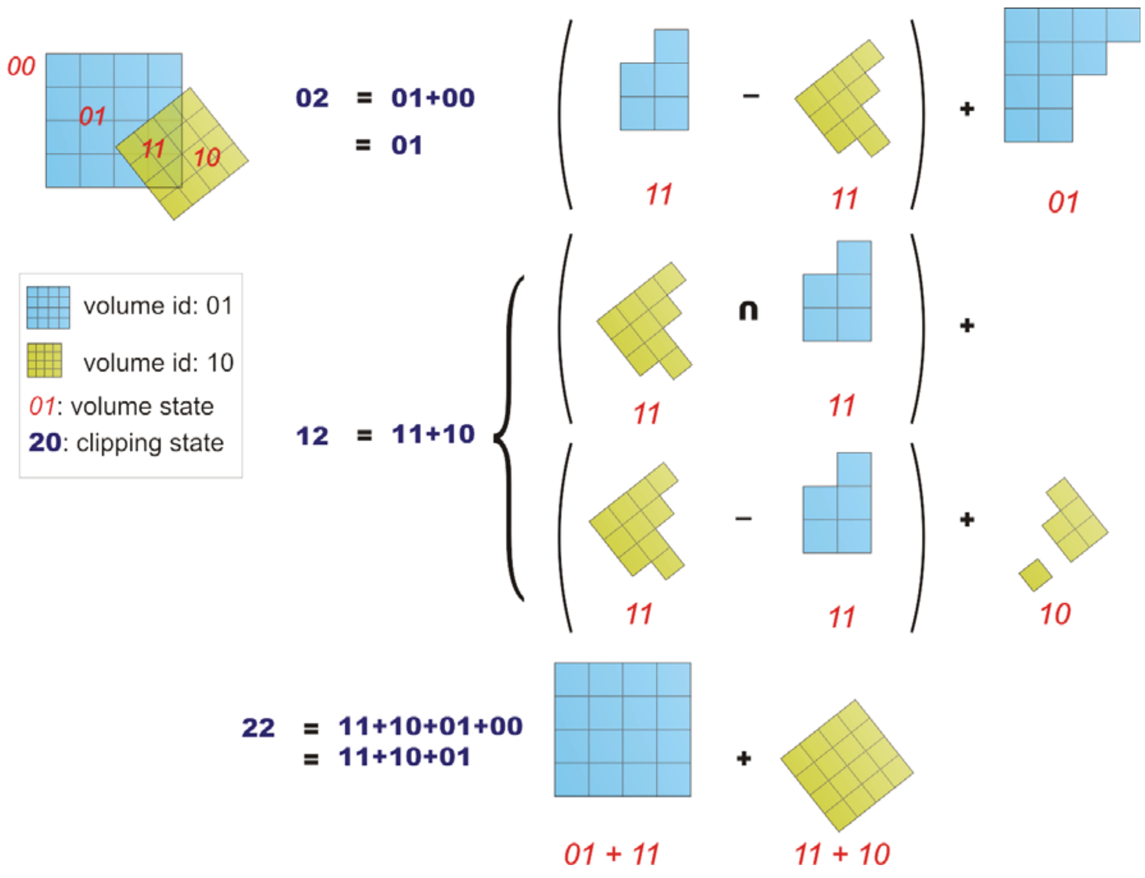


Figure 5.13: Examples of the Digit **2**.

about digit **2** of two volumes. The **02** is the same as the **01** in this example; the **12** is a combination of **11** and **10**; the **22** is the whole scene. The number of the clipping states is reduced by using the digit **2**, which also saves one lookup access in the clipping map at each sampling point. Hence, the performance is improved by using the digit **2** in the clipping map. Fig. 5.8(a) shows the whole volume which is a combination of all parts of the volumes. For example, it can be expressed as:

$$\begin{aligned} \mathbf{11} + \mathbf{10} + \mathbf{01} &\equiv (\mathbf{11} + \mathbf{10}) + (\mathbf{01} + \mathbf{00}) \\ &\equiv \mathbf{12} + \mathbf{02} \equiv \mathbf{22} \end{aligned}$$

where **11** is the intersection between head bone and brain in Fig. 5.9. The clipping state **10** is the head bone by excluding the brain (see Fig. 5.10(b)), while **01** is the brain by excluding head bone (Fig. 5.10(a)). Another examples are shown in Figs. 5.14 and 5.15,

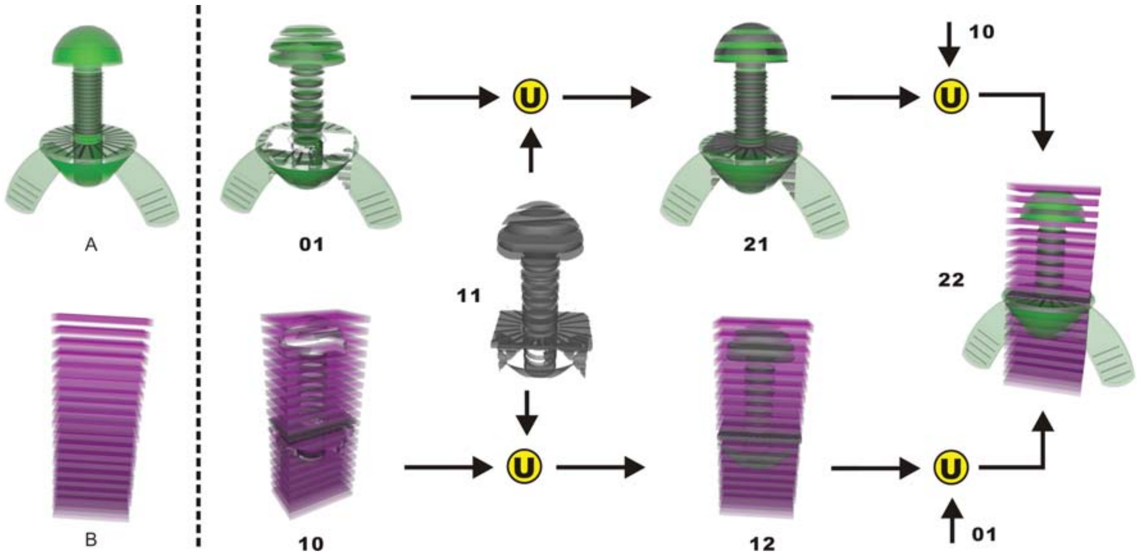


Figure 5.14: Examples of Merging Several Clipping States of Geometries: On the left side of the dash line are original geometries, on the right side are the clipped volumes with own clipping states using the above interpreted merging rules.

in which the merging process is illustrated from the single part to the whole volume. In this example, the whole scene (i.e. **22** and **222**) saves three and six lookup accesses in the clipping map at each sampling point, respectively, when compared to the original clipping state.

5.4 Data Intermixing

The section 4.1 explains the ray-casting based multi-volume rendering, in which the pixel output color combines the intensity and the opacity values of different individual volumes

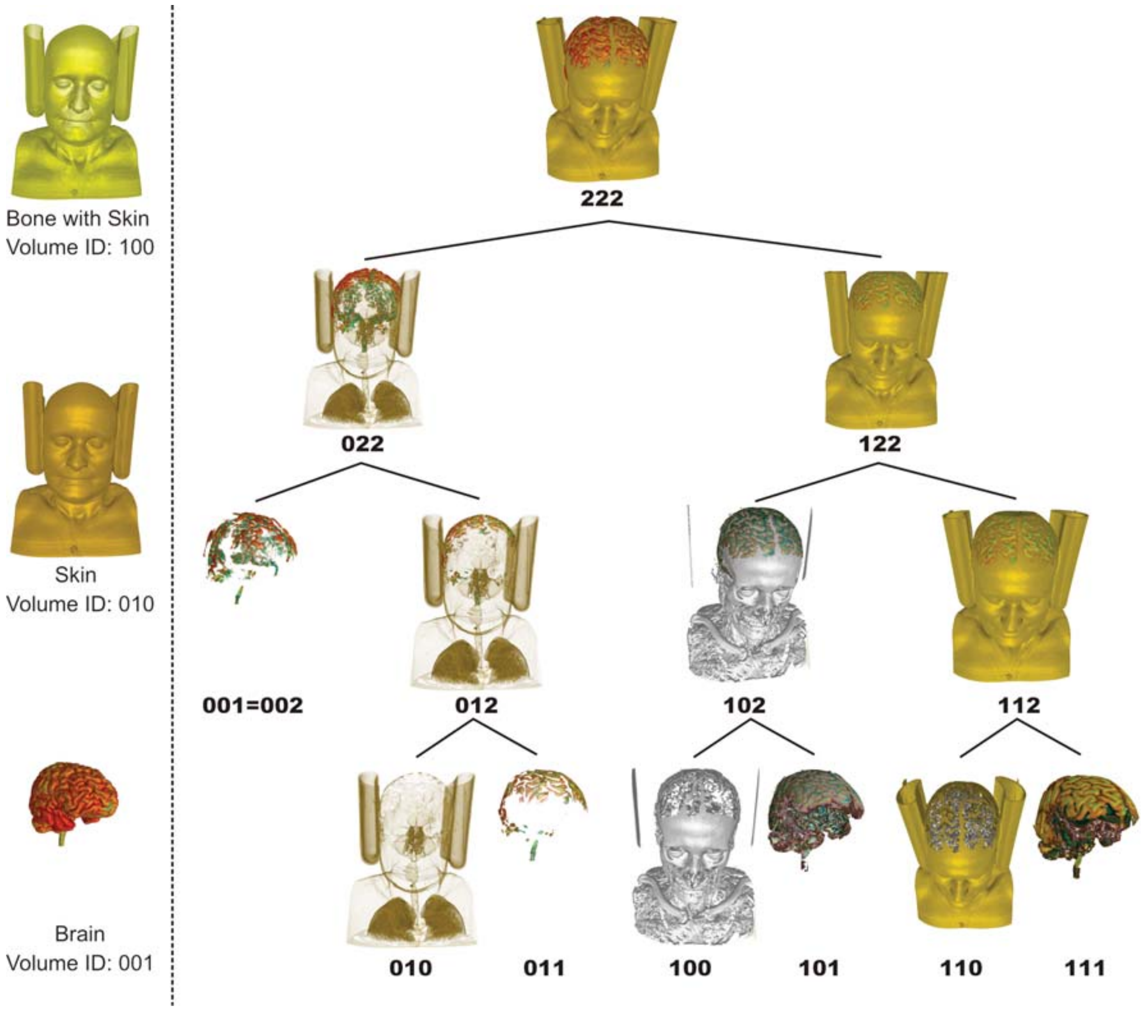


Figure 5.15: Examples of Merging Several Clipping States of volume data. On the left side of the dash line are three original volumes, on the right side are the clipped volumes with own clipping states. The merging algorithm is $X1X + X0X = X2X$, in which the X has the same digit value. (complete examples see Fig. A.4 Fig. A.5

at each sampling point in several intermixing methods. The accumulation level intermixing supports the depth value of the volume and is appropriate for the ray-casting approach. Thus, this intermixing method is usually applied in multi-volume rendering. In the intersected region of two rendered volumes, the intensity and the opacity can be merged by using the accumulation level intermixing. Note that there are three different ways to intermix intensity and opacity from different volumes during the accumulation step [3], namely *exclusive opacity*, *inclusive opacity*, and *a mix of exclusive and inclusive opacity.* The exclusive opacity calculates the intensity (I) and the opacity (α) from one of the volumes in the intersected region. Details of the calculation can be expressed as [3]:

$$I = \begin{cases} I_1 & if \;\; \alpha_1 \neq 0 \\ I_2 & else \end{cases} \quad , \qquad \alpha = \begin{cases} \alpha_1 & if \;\; \alpha_1 \neq 0 \\ \alpha_2 & else \end{cases} \tag{5.2}$$

The exclusive opacity is the priority selection, indicating that the preferred volume is selected if it is visible, whereas the remaining volumes are removed from the intersected region. Usually, the exclusive opacity is calculated by a threshold, over which the preferred volume is selected.

The inclusive opacity is a mixture of two accumulative effects, i.e. the intensities and the opacities of the two volumes, which can be weighted and normalized as[3]:

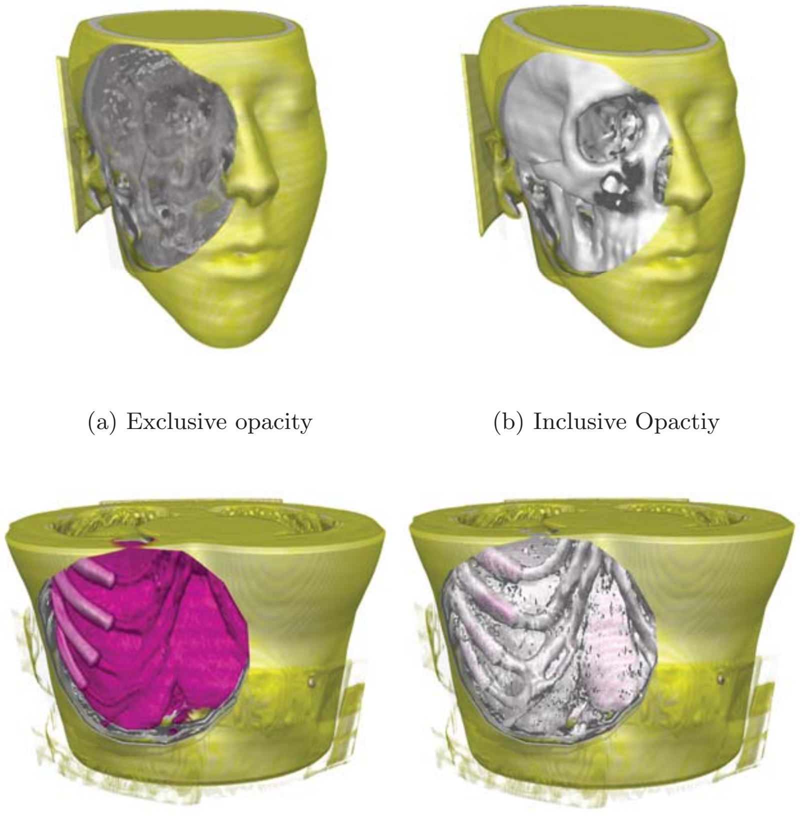
$$I = \frac{\alpha_1}{\alpha_1 + \alpha_2} I_1 + \frac{\alpha_2}{\alpha_1 + \alpha_2} I_2 \tag{5.3}$$

$$\alpha = 1.0 - (1.0 - \alpha_1) * (1.0 - \alpha_2) = \alpha_1 + \alpha_2 - \alpha_1 * \alpha_2 \tag{5.4}$$

The inclusive opacity calculates the ray traversing in each volume individually. The calculated intensity is further multiplied by the associated opacity average value. The intensity will be taken more from the high evaluated opacity than from the low opacity by using the normalized opacity in the intersected region. Likewise, Eqs. 5.3 and 5.4 can be used to intermixi multiple volumes, and examples are shown in Fig. 5.16.

Furthermore, the inclusive opacity is not only appropriate for the volume segments intermixing, but also can be used in the geometry and volume (geometry-volume) intermixing. The intensity and opacity are evaluated by the semi-transparent surface, are also accumulated in the pixel output color (see Fig. 5.17(b)). The inner volumetric segment color of geometry can be also combined into the volume (Fig. 5.17(d)). The inclusive opacity, in general, can be applied in most intermixing cases if the mapping form exists to transform the voxel value (or any object) into intensity and opacity [3].

A mix of the exclusive and the inclusive opacity represents an interactive application of both methods, in which the volumes can be included. For instance, two volumes are mixed first by using the inclusive opacity, and then mixed with the third volume by using the exclusive opacity.



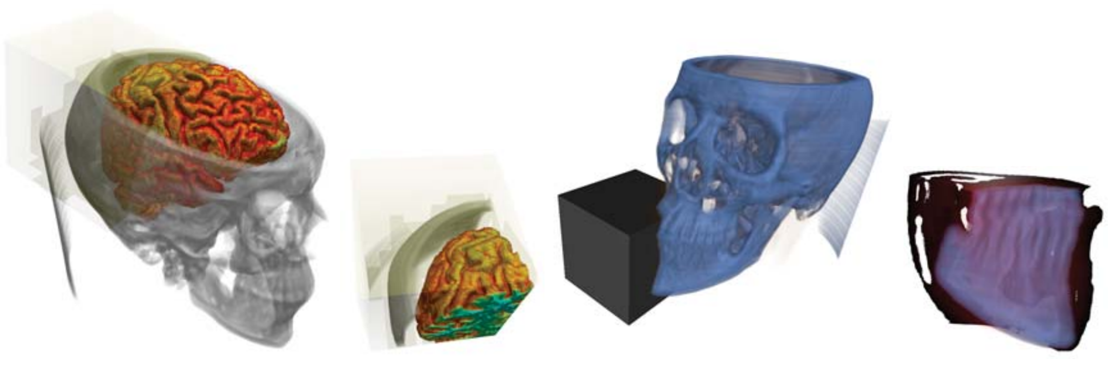
(a) Exclusive opacity (b) Inclusive Opactiy

(c) Exclusive Opactiy (d) Inclusive Opactiy

Figure 5.16: Exclusive Opacity and Inclusive Opacity. (a). A head with skin has intransparent bone, which is replaced with a semi-transparent head by using the exclusive opacity method. The semi-transparent head is prior selected. (c). A body has brown inner organs, which is replaced with a pink interior by using the exclusive opacity method. (b). (d). The intensity and opacity of intersected region are mixed by using the inclusive opacity method. (see Fig. A.6 and A.7)

5.5 CSG tree encoding

The sum-of-product form of the normalized CSG tree can be encoded into the clipping state, the encoding schema has been introduced in the section 5.2.2. Also, the encoding is appropriate for the algebraic tree pruning. The sum-of-product is an OR-Association of the products, in which the operator (+) is used in encoding, each product is the AND-Associations and consists of Boolean subtraction and intersection. The Goldfeather normalised algorithm guarantees that each product can be converted into the positive form,



(a) Semi-transparent geometry with volumes (b) Intermixing with surface boundary (c) Semi-transparent volume with geometry (d) Volumetric intermixing

Figure 5.17: Examples of Geometry-Volume Intermixing.

which consists of the Boolean intersection in inter node, and the corresponding negation is correlated the CSG primitive object [5][6]. The positive form of the product, which can be easily encoded into the clipping state, has both positive and negative parts in a product expression. The negative part is the subtrahend in the subtraction of product and the remaining parts are all positive [17]. The positive part, denoted as **1** in the clipping state, is the rendered volume in the current scene. The negative part, however, is clipped from the current scene and is denoted as *0*. The CSG expression encoding is described in the following table:

Clipping State Encoding

$S \in \{A, B\}$	sum of product	positive form	Clipping State
A has volume id *10*	$A - B$	$A\overline{B}$	**10**
B has volume id *01*	$A \cap B$	AB	**11**
	$B - A$	$\overline{A}B$	**01**
	A	A	**12**
	B	B	**21**

Table 5.2: Clipping State Encoding.

All possible combinations are listed in the table. The positive form can be simply mapped into the clipping state. Remarkably, the volume A in the single volume rendering is a part of the volume A in the CSG rendering system, which also includes the intersected region between the single volume A and other rendered volumes. As shown in table 5.2,

the volume object A is represented by the digit **1** in the volume state, while the remaining parts are expressed by the digit **2**.
The CSG objects are registered in the encoding system and assigned as a volume ID. The encoding system has a single digit for each CSG object. It means that CSG object is encoded in one bit of each product, in which the algebraic pruning method is used when the same object appears twice in the product. For instance, the expression of $A - A$, indicates that the product can be ignored, and the expression of $A \cap A$ denotes, that the second A can be ignored. Such a simplification can be also used in the multi-objects case. The sum operator (Boolean OR-association) collects all products, where the (+) is used and multiple lines exist in the lookup map. By checking the clipping state, the duality $(X + X = X)$ can be quickly detected. Basically, the duality is not easy to find in the sum-of-product form, since the sequence plays a key role, for example $A \cap (B - C) = (B - C) \cap A$.
Furthermore, each product may have redundant information to increase the rendering cost (twice rendered), which, however, can be removed from the clipping state encoding. The clipping state, which only contains the digits **1** and **0**, is the smallest evaluated unit in the rendering pipeline. A intersection of these two clipping states are empty, while subtraction of them is exactly the first clipping state. The encoding example is illustrate as follows:

$$\begin{aligned}
A - B - BC \quad &\longrightarrow \quad \mathbf{122} - \mathbf{212} - \mathbf{211} \\
&\equiv \{\, \mathbf{102} \cup \mathbf{112} \,\} - \{\, \mathbf{012} \cup \mathbf{112} \,\} - \mathbf{211} \\
&\equiv \{\, \mathbf{100} \cup \mathbf{101} \,\} - \{\, \mathbf{010} \cup \mathbf{011} \,\} - \{\, \mathbf{011} \cup \mathbf{111} \,\} \\
&\equiv \{\, \mathbf{100} \cup \mathbf{101} \,\} \\
&\equiv \mathbf{102} \qquad (A\overline{B})
\end{aligned}$$

$$\begin{aligned}
ABC \cap A \quad &\longrightarrow \quad \mathbf{111} \cap \mathbf{122} \\
&\equiv \mathbf{111} \cap \{\, \mathbf{100} \cup \mathbf{101} \cup \mathbf{110} \cup \mathbf{111} \,\} \\
&\equiv \{\, \mathbf{111} \cap \mathbf{100} \,\} \cup \{\, \mathbf{111} \cap \mathbf{101} \,\} \cup \{\, \mathbf{111} \cap \mathbf{110} \,\} \cup \{\, \mathbf{111} \cap \mathbf{111} \,\} \\
&\equiv \mathbf{111} \qquad (ABC)
\end{aligned}$$

The sum-of-product form can be easily converted into the clipping states. The clipping state is also appropriate to remove the redundant part of the form, and is also applied in the volume clipping algorithm, which can represent all CSG models in the rendering.

5.6 Summery

This chapter describes the complete volume/geometry clipping in the rendering pipeline from the pre-defined sum-of-product form of the CSG model. The clipping rendering algorithm is extended from the multi-volume ray-casting rendering algorithm, which split the volumes in each polygonal segment interval during the ray-casting. The pre-defined sum-of-product form is converted into the clipping states, which are used to evaluate the volume clipping in each polygonal segment interval. Compared to the multi-volume ray-casting rendering algorithm, the segment interval, which only contains the clipped volumes, is rejected after checking in the clipping states. Another difference is the opaque test of each volume at the sampling point during the ray-traversing. These tests work together with the corresponding digit-bit of the clipping states and the volume/geometry clipping can be accurately achieved in each sampling point. Thus, the CSG model can be applied per sampling point accuracy in the rendering pipeline.

Chapter 6

Result

Contents

6.1 Performance Measurement in the Rendering

Compared to the standard ray-casting approach, the volume clipping based CSG rendering algorithm has two additional steps during rendering, which are:

i. the checking of all clipping states in each depth interval, which finds out the polygonal segment intervals of the clipped volume.

ii. The opacity and the clipping state tests, which are to check each volume at the sampling points during the traversing in remained polygonal segment intervals.

The ray is skipped by the segment intervals of type i), in order to save the unnecessary traverse of the clipped volume. The volume clipping is realized by the opacity and the clipping state tests of type ii). A sampling point with a non-zero opacity value, which indicates **1** in the clipping state, is defined as rendered. In contrast, a sampling point with a non-zero opacity value, which indicates **0** in the clipping state, is defined as clipped. Two volumes in a clipping state, which are defined as rendered in same sampling point, are implemented as intersection operation; which are defined as rendered and clipped respectively, are implemented as subtraction operation. The digit **2** combines the digit **1**

and **0** in one, the corresponding volume segment is intersected by the rendered volume and clipped by the clipped volume. The union operation represents the multiple clipping states, which have the or-associate to each other. The clipping states are stored in the texture, a check in the clipping state costs one texture access. This check is applied by the corresponding opaque volume at each sampling point during ray traversing.
The performance of our depth-based volume clipping depends on both the number of the CSG primitives and of the clipping states. The clipping states is converted by the defined CSG model, the complexness is directly transfer by the model. The tests of the opacity and the clipping states need more performance by increasing the number of the CSG primitives and of the clipping states. This increasing will lead to a lower frame rate in the CSG evaluation of each sampling point.

6.2 Time Measurement of Volume Clipping

The ray-casting volume rendering is specified on the NVIDIA Geforce 200-Serie, which supports the CUDA capability 1.3. For the performance evaluation, we compare ray-casting volume rendering with and without volume clipping in the framework, also difference between the use of single and multiple clipping states during the volume clipping by using minor and major CSG volumes. The comparison is done on the computer (intel i7-960 3.20 GHz) with GTX 280 (1024 DRAM, 512-bit Memory Interface, 240 CUDA-cores) in Linux platform.
The first test data is medical scene with multiple volumes, which use multiple modalities or scans from different body parts. This test data uses the same data set in the solution of 256^3 and the different transfer functions (see Fig. 6.1). Table 6.1 compares the frag-

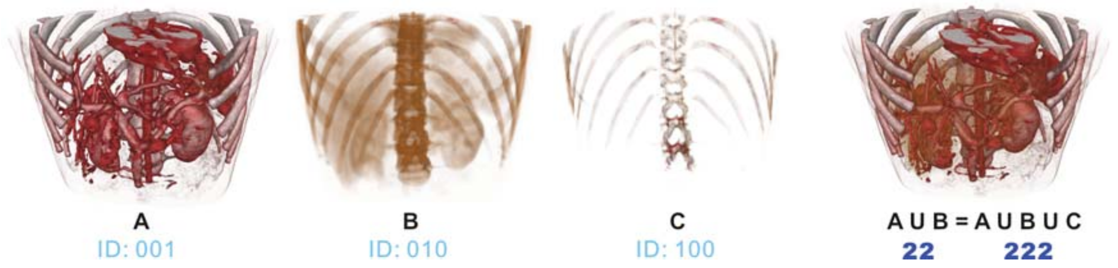


Figure 6.1: Volume Clipping of medical Liver Data. Three volumes uses the same volume data set and different transfer functions

ment process and frame times of Kainz et al. [7] ray-casting volume rendering achieved

Volumes	clipping states	average time of fragment process [ms]	average time of complete frame [ms]
2×256^3	–	130	175
1 PSI	**22**	164	224
	01+11+10	252	290
3×256^3	–	159	201
1 PSI	**222**	229	269
	4 comb. of **222**	332	377
	7 comb. of **222**	452	498
3×256^3	–	195	284
7 PSI	**222**	239	319
	4 comb. of **222**	372	457
	7 comb. of **222**	519	607

Table 6.1: Result Comparing of Liver Data Set 256^3. Comparing the ray-casting volume rendering with/without depth-based volume clipping approach, the corresponding volume clipping algorithm is applied in single/multiple clipping states; screen size: 1050×800. (PSI: polygonal segment interval)

with and without volume clipping approach. The average time[*] of the fragment process and complete frame (including the data initialization, polygon rasterization and fragment process) of CSG rendering are listed. The standard ray-casting approach without volume clipping needs the shortest time in both cases (clipping states with '–' in the table). The clipping states 22 and 222 activate the volume clipping approach. The result of the original rendering (from Kainz et al.) and of the clipping states 22 and 222[†] are exactly the same (see the right image in Fig. 6.1). The run time of the fragment process increases by activating the clipping state and will increase more with higher number of the clipping states or higher number of the input volumes. Moreover, the performance is different when the same data set splits in the several polygonal segment intervals. The first 4 entries of 3×256^3 (see the table 6.1) indicate that volumes are in the same location, therefore, only one polygonal segment interval exists. The last 4 entries (int table 6.1) are calculated in 7 polygonal segment intervals, and the three volumes have different locations. The fragment time increases with ascending polygonal segment intervals. The frame time will also increase if there are multiple polygonal segment intervals, in this case, the data initialization and polygon rasterization needs more time.

[*] The average time calculates with the mean of last 100 run times, the rendering system is run in the stable state, the first 100 run times go not into the calculation.

[†] The volume C is a part of the volume B, the union between A and C is exact the A, because the body bone of A is not transparent.

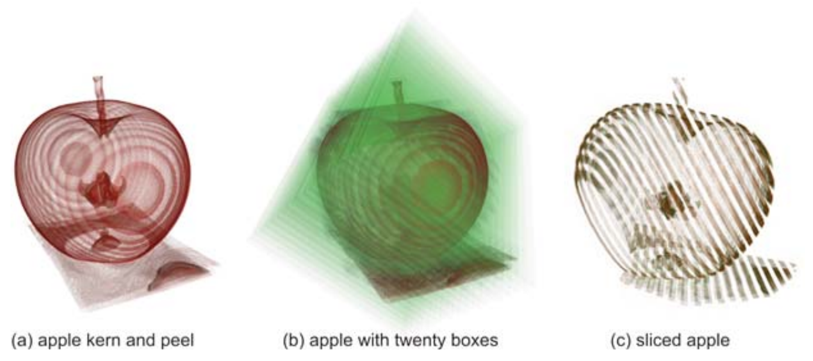


Figure 6.2: Sliced Apple: the volume apple is sliced by twenty boxes.

Volumes	clipping states	average time of fragment process [ms]	average time of complete frame [ms]
1×128^3	2	73	132
35 PSI	**11**	113	191

Table 6.2: Result Comparing of Sliced Apple 256^3. Volume data set intersects in several polygonal segment interval. screen size: 1050×800.

The apple (128^3) in Fig. 6.2 is sliced by boxes into several intervals and Table 6.2 is the resulting comparison between the original apple and the sliced apple, indicating that the fragment time increases with ascending polygonal segment intervals.

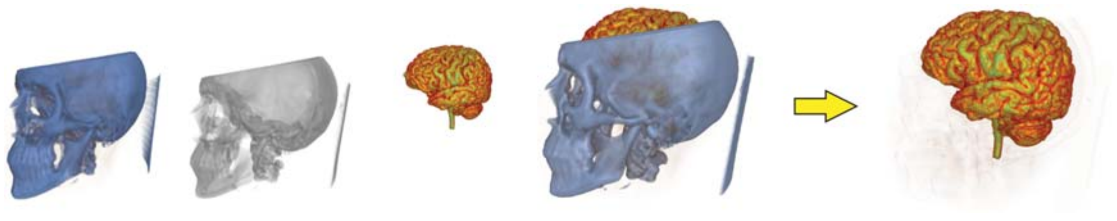


Figure 6.3: Volume Clipping of Medical Head Data. Two Head uses the same volume data set and different transfer functions

Fig. 6.3 is an example of the volume clipping in medical volume data sets and Table 6.3 is the corresponding summarized time measurements. The time measurement in Table 6.1 takes similar effect in Table 6.3. However, the coverage rate of the ray and the used transfer function of both examples are different.
Table 6.4 compares the run time of the volume clipping by major volumes in the single and multiple clipping states. All volumes use the same data set and the same transfer

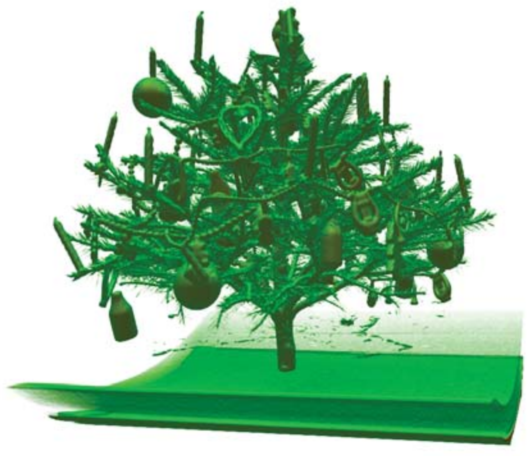
Volumes	clipping states	average time of fragment process [ms]	average time of complete frame [ms]
3 × 256^3	–	132	177
3 PSI	**202**	269	308
	102 + 002	323	369
	100+101+001+000	398	449

Table 6.3: Result Comparing of Head Data Set 256^3. screen size: 1050 × 800.

Volumes	Number of clipping states	Average time of fragment process [ms]	Average time of complete frame [ms]
12 × 256^3 [Fig. 6.1]	–	621	693
1 PSI	1/4/16	737/891/1809	817/974/1909
12 × 256^3 [Fig. 6.1]	–	886	963
11 PSI	1/4/16	1052/1340/2382	1139/1432/2493
12 × 256^3 [Fig. 6.3]	–	972	1065
1 PSI	1/4/16	1092/1465/2568	1207/1571/2681
12 × 256^3 [Fig. 6.3]	–	1326	1445
11 PSI	1/4/16	1433/2189/4115	1555/2329/ 4272

Table 6.4: Performance Measurement by multiple volumes (in single or multiple polygonal segment intervals) and multiple clipping states. screen size: 1050 × 800.

function. The results vary between single and multiple polygonal segment intervals by using different location of the volumes. Our reference limits the number of volumes data set in 12 (1GB Video RAM). Compared to the result of minor volumes (2 or 3 volumes), the ray traverses more volumes in each segment interval. The corresponding access of the opacity and the clipping states of major volumes is more required in each sampling point during the ray traversing. The run time of fragment process grows linearly with the increasing numbers of the volumes and of the clipping states (complex CSG model).
Another test data is the Christmas tree in the solution of 512^3. Two same data sets are located in the scene by moving 1 unit in the x-axis (see Fig. 6.4) and split the scene into three polygonal segment intervals.
Table 6.5 is the corresponding time measurements of the data set 512^3. The run time take similar effect as the above examples. The opaque Christmas tree is caused by breaking the ray in the intersection of the tree in the clipping state **22**. Whereat the ray further traverses in the intersected segment interval (transparent) in the clipping state **01**, until either the rendered tree segment or the end of the volume is reached. Thus, the run time


(a) Two same volume data sets

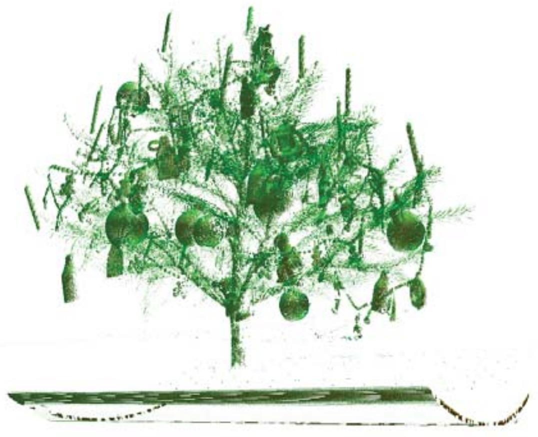

(b) The different of volumes (**10**)

Figure 6.4: Test Example of Volume Clipping. Two volumes use the same volume data set and same transfer functions, and are located by moving 1 unit in the x-axis

Volumes	clipping states	average time of fragment process [ms]	average time of complete frame [ms]
2×512^3	–	457	539
3 PSI	**22**	655	730
	01+11+10	909	987
	01	875	968

Table 6.5: Result Comparing for Data Set 512^3. screen size: 1050×800.

of fragment process needs more time in the clipping state **01** when compared with the state **22**.

6.3 Result of Interactive CSG

The first CSG example is a medical scene with multiple volumes (see Fig. A.8). The complete body consists of the volume parts, which use multiple modalities or scans from different modalities. The volume data [‡] and geometry data [§] are used as primitives in

[‡]all used volume data has the same data set in solution 256^3 and the different transfer function, see volumes in center bottom of Fig. A.8

[§]blood vessel, transparent lung, blue and green clip cylinders the CSG model. If we just want to see the blood vessel, the heart and the kidney will be hidden. A total of 11 CSG primitives are used in this example and the number of clipping states is 6 lines after transform. Table 6.6 shows the time measurement by using

CSG primitives	The number of clipping states	Average time of fragment process [ms]	Average time of complete frame [ms]
3×256^3	–	322	401
+ 8 Geometry	6	936	1012

Table 6.6: Result of CSG Liver Example. screen size: 1050×800

enabled and disabled CSG approaches. As can be see in the table, the run time will be approximately 3 times than if without using the CSG approach.
In Fig. A.9, the head volume is split step by step, whereby the inner details of the head is clearly shown in example. The scene is split into several polygonal segment intervals and

CSG primitives	The number of clipping states	Average time of fragment process [ms]	Average time of complete frame [ms]
5×128^3	–	244	313
+ 6 Geometry	6	918	1015

Table 6.7: Result of CSG Head Example. screen size: 1050×800

the evaluated segment intervals is also 5 times than the original volume data, both needs much more time in the fragment process (see Table 6.7).
Figs. A.1, A.2 and A.3 illustrate the CSG examples of geometry-geometry and geometry-volume objects. The performance of the CSG rendering algorithm is directly dependent on the number of the primitives and of the used clipping states (the complexity of the CSG model). A important feature in this CSG rendering algorithm is that the fragment time is not more different by using the varying transfer function. The CUDA based rendering algorithm is based on the power of the GPU and the primitives are limited by the the VRAM. The calculated speed limits by the available CUDA-Core Unit. The performance can be improved with upgraded processing power. In conclusion, the rendering system is able to construct the complete scene by using the CSG model.

Appendix A

Image Gallery

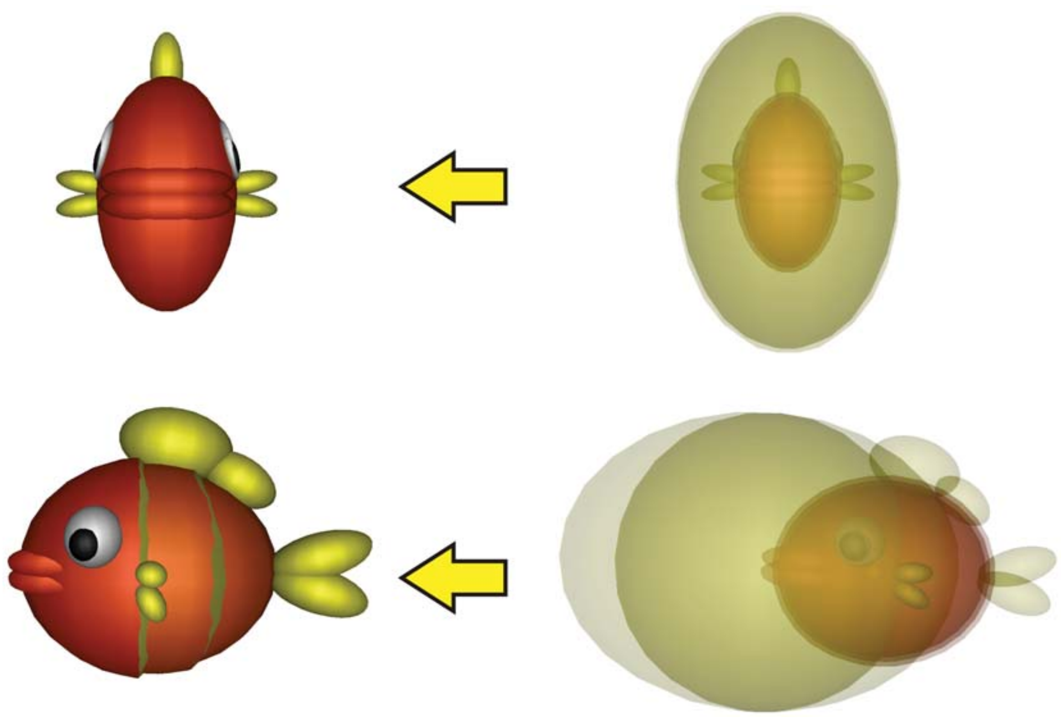


Figure A.1: CSG Example of 19 simple geometries: Fish

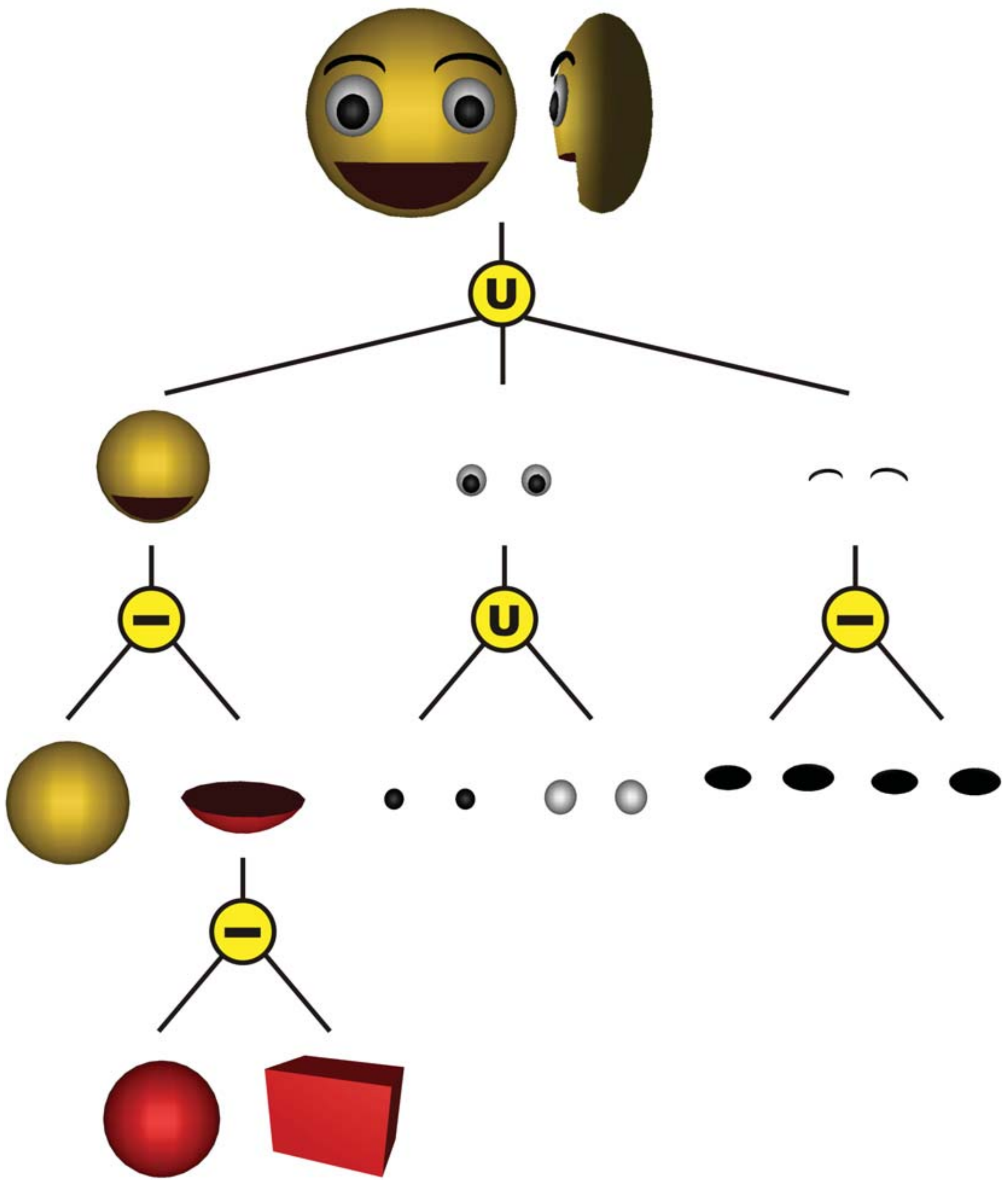
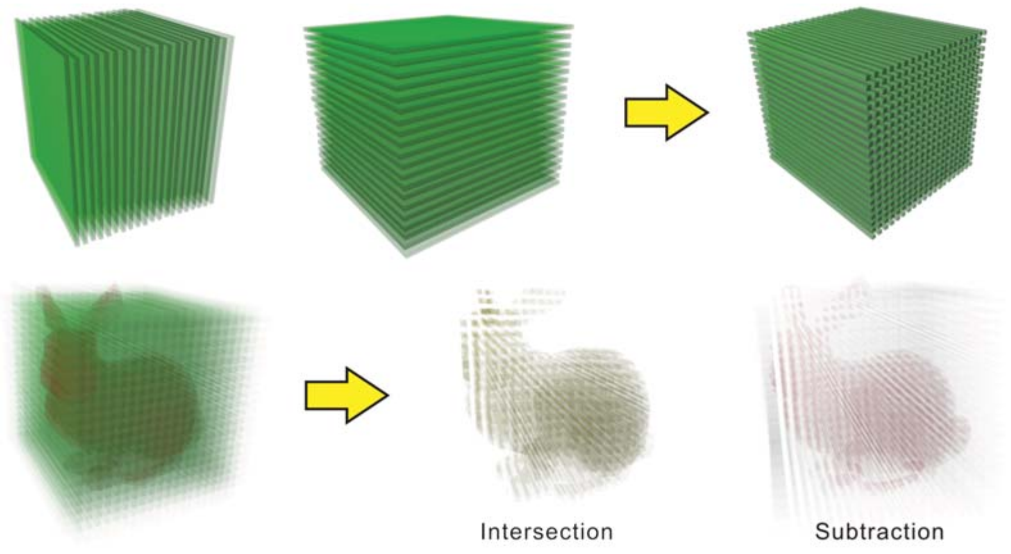
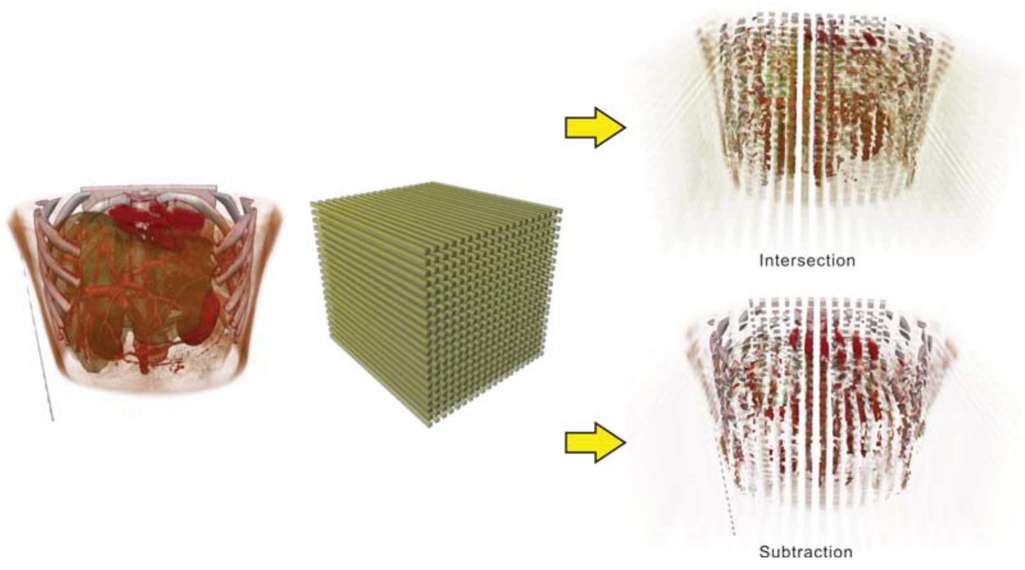


Figure A.2: CSG Example of simple geometries: Smile



(a) CSG Example of 400 rectangles with Bunny: 'voxelized Bunny' (Geometry-Geometry)



(b) CSG Example of 400 rectangles with 4 Volume Data: 'voxelized Body' (Geometry-Volume)

Figure A.3: CSG Examples by major simple geometries.

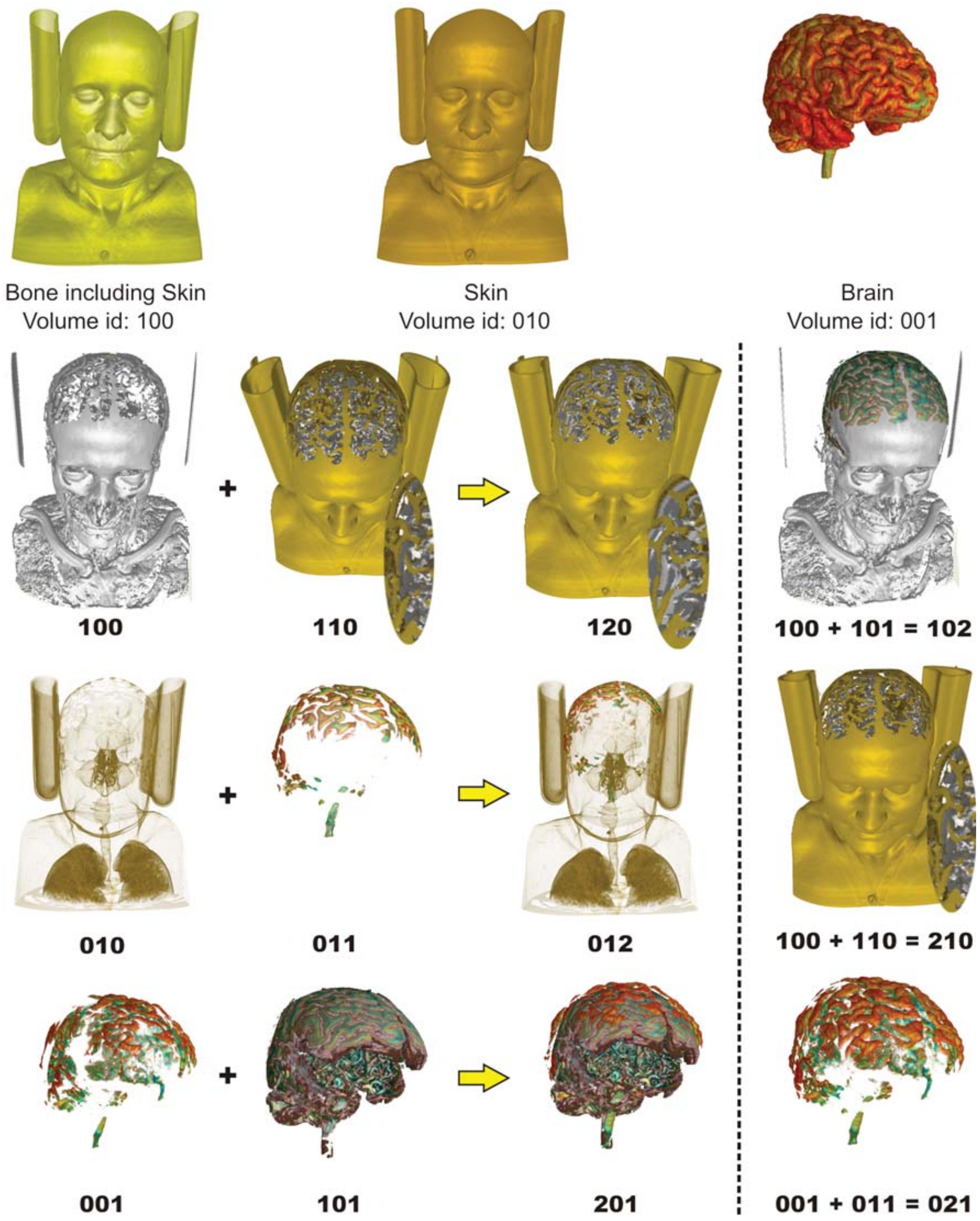


Figure A.4: Clipping State Examples. The clipping state, which contents the digit **2**, can be separable in two other clipping state (in the corresponding digit stead **2** the **1** and **0**).

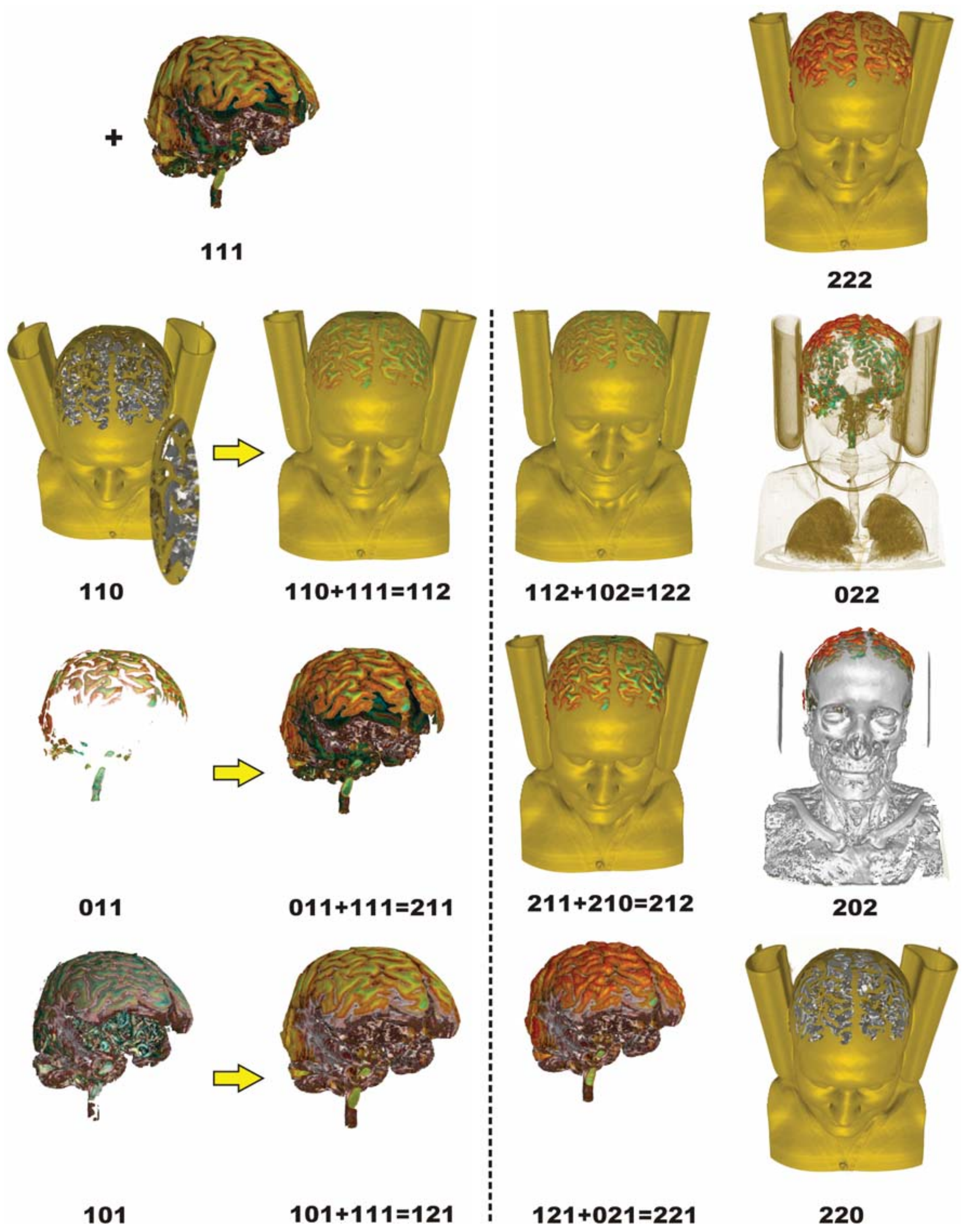
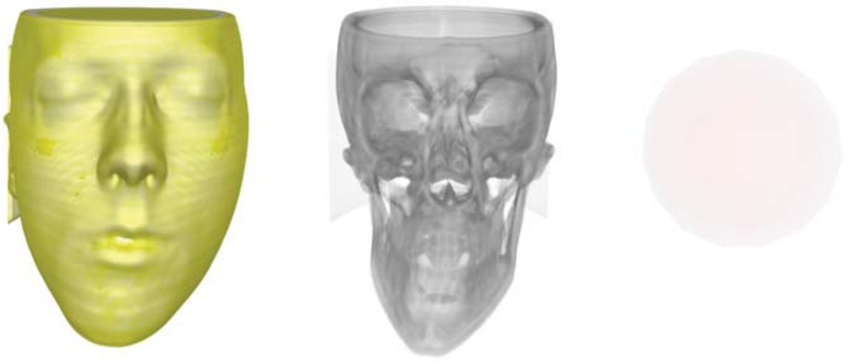
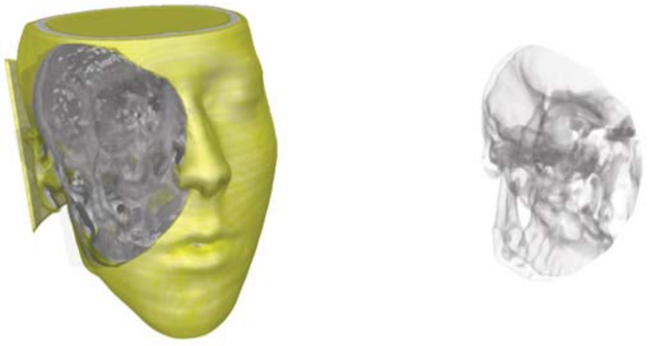


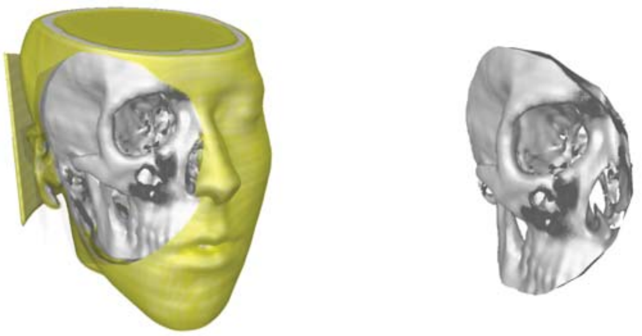
Figure A.5: The Merging of Clipping State Example. The clipping states **1** (column 3) and **0** (column 4) in same digit are inverse to each other.



(a) input volumes of the Datamixing



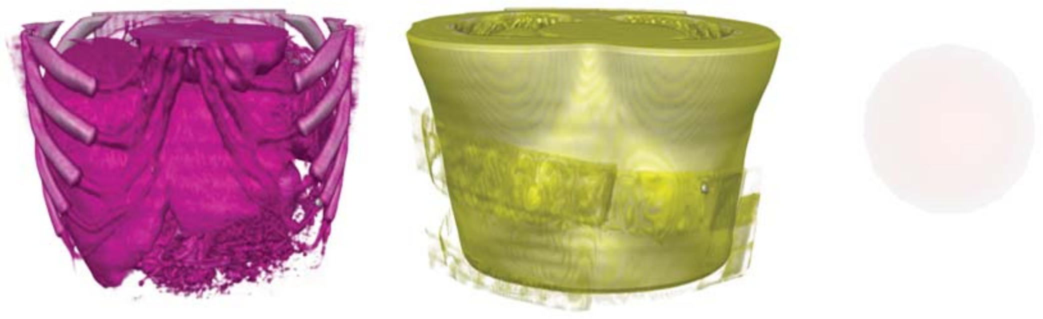
(b) the Datamixing with Exclusive Opacity



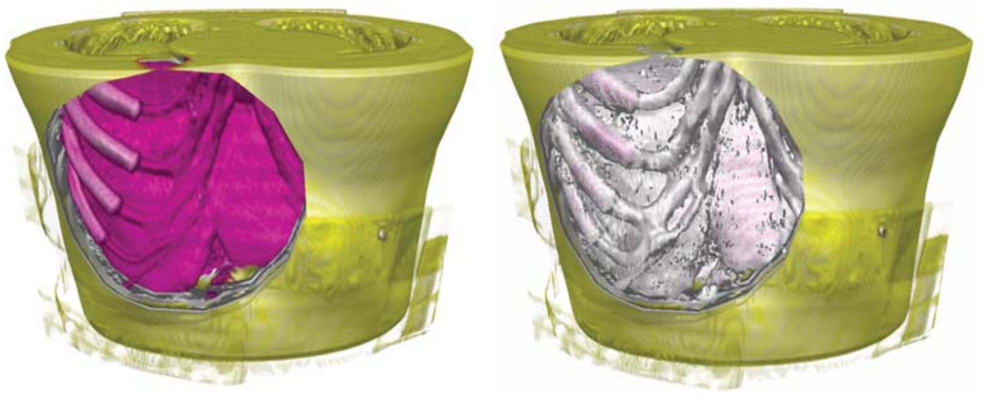
(c) the Datamixing with Inclusive Opacity

Figure A.6: Data Mixing of Inclusive and Exclusive Opacity. (a) The input heads and the geometric sphere. The volume data set head with skin has intrasparent bone. (b) The intransparent bone is replaced with a semi-transparent head by using the exclusive opacity method. The semi-transparent head is prior selected. (c) The intensity and opacity of the intersected region are mixed by using the inclusive method (weighted and normalised).

Draft Copy: September 19, 2011

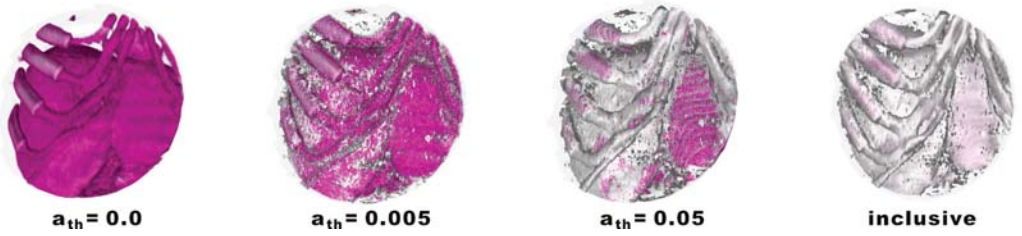


(a) input volumes of the Datamixing



(b) the Datamixing with Exclusive Opacity

(c) the Datamixing with Inclusive Opacity



(d) exclusive threshold

Figure A.7: Data Mixing of Inclusive and Exclusive Opacity. (a) The input livers and the geometric sphere. The volume data set body has silver-color inner organs. (b) The inner organs in brown is replaced with inner organs in pink by using the exclusive opacity method. The inner organs in pink is prior selected. (c) The intensity and opacity of the intersected region are mixed by using the inclusive method (weighted and normalised). (d). The exclusive opacity with different threshold in the intersection region.

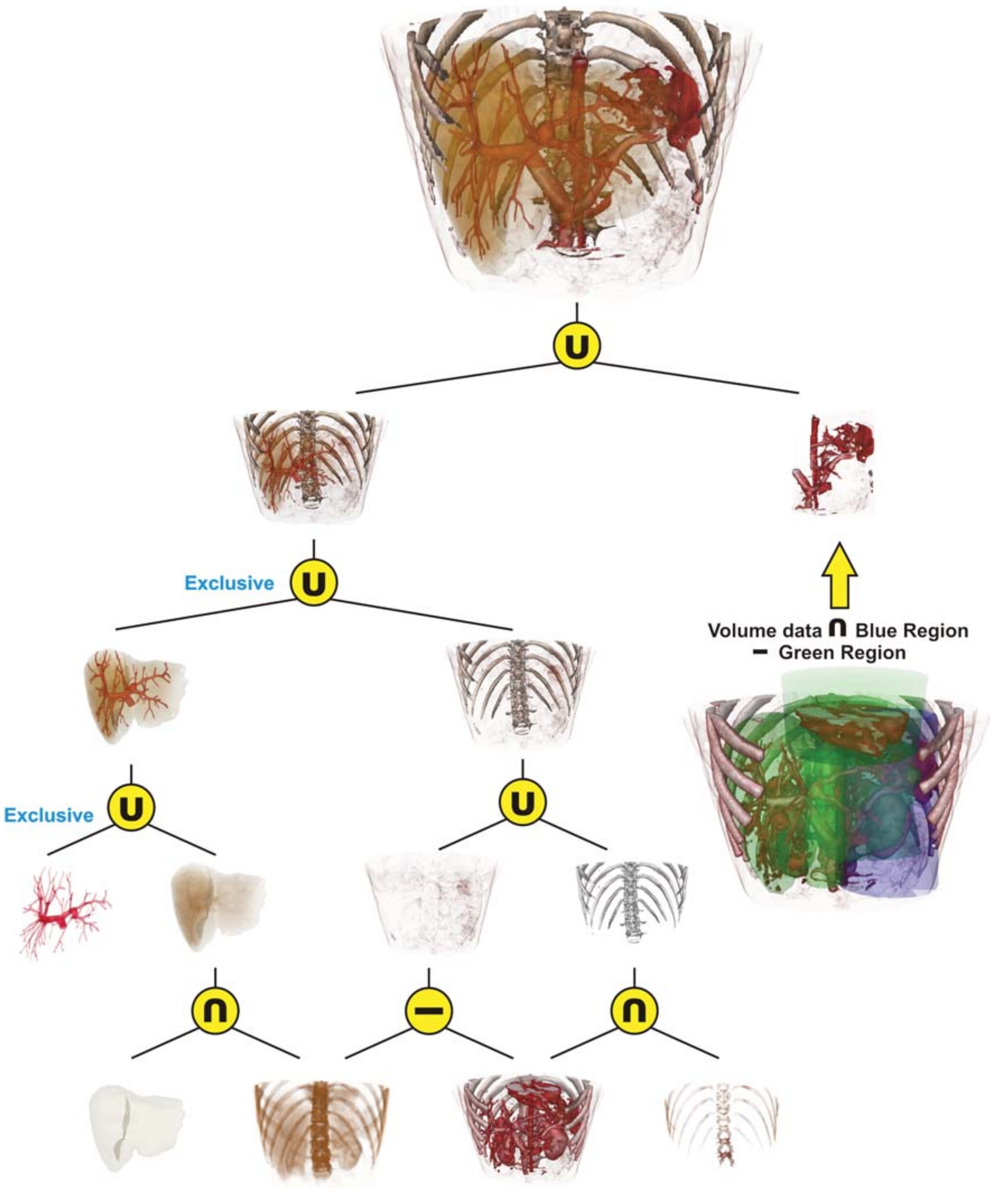
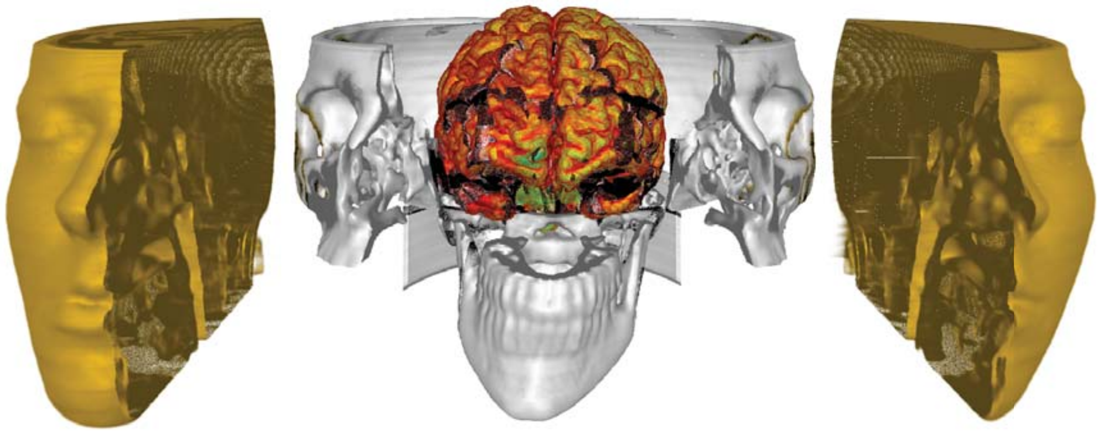
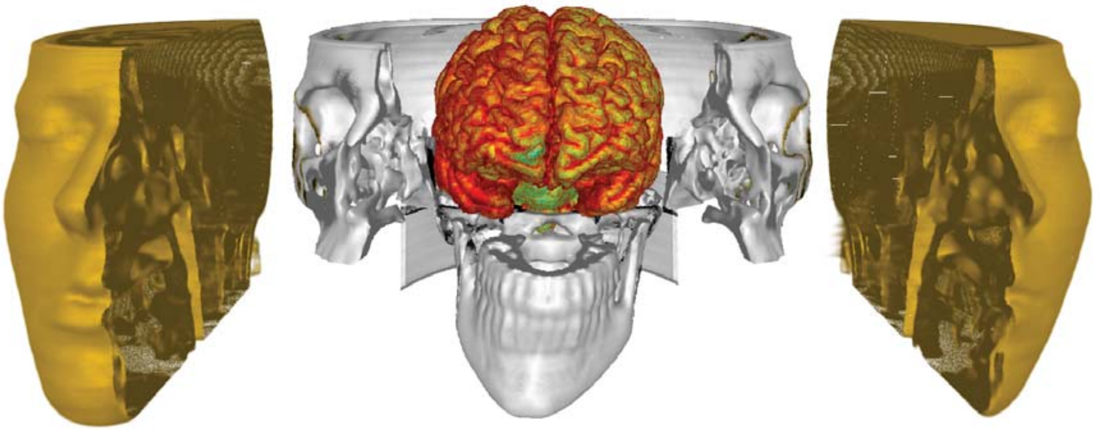


Figure A.8: Interactive CSG example. The final result is combined from 11 primitive objects, which contain the geometric objects (such as blood vessel, liver-geometry, cylinders) and volume data sets (the body with three different transfer functions).



(a) split head volume with inclusive brain



(b) split head volume with exclusive brain

Figure A.9: Interactive CSG examples. The final result is combined from 11 volume objects, which contain the geometric objects (such as blood vessel, liver-geometry, cylinders) and volume data sets (the body with three different transfer functions).

Bibliography

[1] Akenine-Moeller, T., Haines, E., and Hoffman, N. (2008). *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA.

[2] Brecheisen, R., Bartroli, A. V., Platel, B., and ter Haar Romeny, B. M. (2008). Flexible gpu-based multi-volume ray-casting. In Deussen, O., Keim, D. A., and Saupe, D., editors, *VMV*, pages 303–312. Aka GmbH.

[3] Cai, W. and Sakas, G. (1999). Data intermixing and multi-volume rendering. *Comput. Graph. Forum*, 18(3):359–368.

[4] Erhart, G. and Tobler, R. F. (2000). General purpose z-buffer csg rendering with consumer level hardware. Technical report.

[5] Goldfeather, J., Hultquist, J. P., and Fuchs, H. (1986). Fast constructive-solid geometry display in the pixel-powers graphics system. In Evans, D. C. and Athay, R. J., editors, *SIGGRAPH*, pages 107–116. ACM.

[6] Goldfeather, J., Monar, S., Turk, G., and Fuchs, H. (1989). Near real-time csg rendering using tree normalization and geometric pruning. *IEEE Computer Graphics and Applications*, 9:20–28.

[7] Kainz, B., Grabner, M., Bornik, A., Hauswiesner, S., Muehl, J., and Schmalstieg, D. (2009). Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore gpus. *ACM Trans. Graph.*, 28(5).

[8] Kirsch, F. and Doellner, K. (2005). Opencsg: A library for image-based csg rendering. In *USENIX Annual Technical Conference, FREENIX Track*, pages 129–140. USENIX.

[9] Klaus Engel, Daniel Weiskopf, C. R.-S. M. H. and Kniss, J. M. (2006a). *Real-time Volume Graphics*, chapter 1 Theoretical Background and Basic Approachs. A. K. Peters, Ltd., Natick, MA, USA.

[10] Klaus Engel, Daniel Weiskopf, C. R.-S. M. H. and Kniss, J. M. (2006b). *Real-time Volume Graphics*, chapter 7 GPU-Based Ray Casting. A. K. Peters, Ltd., Natick, MA, USA.

[11] Klaus Engel, Daniel Weiskopf, C. R.-S. M. H. and Kniss, J. M. (2006c). *Real-time Volume Graphics*, chapter 15 Volume Clipping. A. K. Peters, Ltd., Natick, MA, USA.

[12] Levoy, M. (1990). Efficient ray tracing of volume data. *ACM Trans. Graph.*, 9:245–261.

[13] Requicha, A. A. G. (1980). Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464.

[14] Requicha, A. A. G. and Voelcker, H. B. (1985). Boolean operations in solid modelling: Boundary evaluation and merging algorithms. *Proc. of the IEEE*, 73(1):30–44.

[15] Roessler, F., Botchen, R. P., and Ertl, T. (2008). Dynamic shader generation for gpu-based multi-volume ray casting. *IEEE Computer Graphics and Applications*, 28(5):66–77.

[16] Rossignac, J. (1994). Processing disjunctive forms directly from csg graphs. *CSG 94: Set-theoretic Solid Modelling: Techniques and Applications*, (4):55–70.

[17] Rossignac, J. and Voelcker, H. B. (1989). Active zones in csg for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithms. *ACM Trans. Graph.*, 8(1):51–87.

[18] Schubert, N. and Scholl, I. (2011). Comparing gpu-based multi-volume ray casting techniques. *Comput. Sci.*, 26:39–50.

[19] Stewart, N., Leach, G., and John, S. (1998). An improved z-buffer csg rendering algorithm. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '98, pages 25–30, New York, NY, USA. ACM.

[20] Stewart, N., Leach, G., and John, S. (2002). Linear-time csg rendering of intersected convex objects. In *WSCG*, pages 437–444.

[21] Stewart, N. T. (2008). *An image-space algorithm for hardware-based rendering of constructive solid geometry*, pages 11–33.

[22] Weiskopf, D., Engel, K., and Ertl, T. (2003). Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Trans. Vis. Comput. Graph.*, 9(3):298–312.

[23] Wernecke, J. (1993). *The Inventor Mentor: Programming Object-Oriented 3d Graphics with Open Inventor, Release 2.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[24] Wiegand, T. F. (1996). Interactive rendering of csg models. *Comput. Graph. Forum*, 15(4):249–261.