Reinhard Enhuber, BSc

Comparison of Light Assignment Methods for Screen Space, World Space and Object Space

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Advisor

Dipl.-Ing. Dipl.-Ing. Jörg Müller, BSc
Institute of Computer Graphics and Vision

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Markus Steinberger, BSc
Institute of Computer Graphics and Vision

Graz, Austria, Feb. 2019

Abstract

Adding many lights to a scene enhances the visual fidelity and motivates the usage of light assignment. Light assignment is a non-trivial problem and the question is: Which method is fast and achieves efficient shading?

In order to examine this, we compare the screen space methods: tile-based, dense cluster-based, and sparse cluster-based light assignment. Additionally, for each of these methods, two variants with implicit and explicit bounds are investigated. Furthermore, two new methods are introduced: firstly the grid-based light assignment, which is a world space method and secondly the model-based light assignment, an object space method. In this work, we first optimize the group size parameter and the intersection test. Then the methods are analyzed by regarding their view dependency, and performance.

For all methods, the frame time grows linearly with the light count and can be described by slope and the y-intercept, which we call offset. Sparse cluster-based light assignment with explicit bounds has the lowest slope, the highest offset, the lowest standard deviation of the frame time and a low dependency of the light count distribution on depth complexity. Dense cluster-based light assignment with implicit bounds and grid-based light assignment have similar characteristics: high slope, low offset, and medium standard deviation. Model-based light assignment has a highly diverse frame time across the tested scenes and a high to very high slope.

For 1 000 or more lights, the best method is sparse cluster-based light assignment with explicit bound. For less than 1 000 lights, dense cluster-based with implicit bounds or grid-based light assignment performs better. For the tested scenes, model-based light assignment has a too high slope to be of practical use. The bad performance is caused by large models which cover a large portion of the scene. However, in our opinion, the performance could be significantly improved by adapting the model subdivision. Grid-based light assignment and model-based light assignment have the advantage that the light assignment can be precomputed for static lights. Moreover, the global scope of the light assignment enables the usage for object space shading.

Kurzfassung

Die Verwendung einer großen Anzahl von Lichtern in einer 3D Szene kann die visuelle Wiedergabetreue steigern und motiviert somit zur Verwendung von Light Assignment. Light Assignment ist kein einfach zu lösendes Problem. Es stellt sich die Frage: Welche Methode ist schnell und erreicht gleichzeitig effizientes Shading?

Um dies zu untersuchen, vergleichen wir mehrere Screen Space Methoden: Tile-Based, Dense Cluster-Based und Sparse Cluster-Based Light Assignment. Darüber hinaus untersuchen wir für jede Methode jeweils zwei Varianten. Eine mit impliziten und die andere mit expliziten Begrenzungen. Zudem werden zwei neue Methoden vorgestellt: Grid-Based Light Assignment, eine World Space Methode, und Model-Based Light Assignment, eine Object Space Methode. In dieser Arbeit optimieren wir als Erstes den Parameter für die Gruppengröße und den Überschneidungstest. Dann werden die Methoden nach ihrer Blickabhängingkeit und ihrer Geschwindigkeit analysiert.

Für alle Methoden wächst die Bildberechnungszeit linear mit der Anzahl der Lichter. Dieses Wachstum kann mit einer Steigung und dem y-Achsenabschnitt, welchen wir hier als Basis bezeichnen, beschrieben werden. Sparse Cluster-Based Light Assignment mit expliziten Begrenzungen hat die niedrigste Steigung, die höchste Basis, die geringste Standardabweichung der Bildberechnungszeit und die niedrigste Abhängigkeit der Verteilung der Lichtanzahl von der Tiefenkomplexität. Dense Cluster-Based mit impliziten Begrenzungen und Grid-Based Light Assignment haben ähnliche Charakteristika. Beide haben eine hohe Steigung, eine niedrige Basis und eine mittelgroße Standardabweichung. Model-Based Light Assignment hat eine höchst unterschiedliche durchschnittliche Bildberechnungszeit in den getesteten Szenen und eine hohe bis sehr hohe Steigung.

Für 1 000 oder mehr Lichter ist die beste Methode Sparse Cluster-Based Light Assignment mit expliziten Begrenzungen. Für weniger als 1 000 Lichter sind Dense Cluster-Based Light Assignment mit impliziten Begrenzungen oder Gird-Based Light Assignment besser geeignet. Für die getesteten Szenen hat Model-Based Light Assignment eine zu hohe Stei-

gung um einen praktischen Nutzen zu liefern. Die niedrige Geschwindigkeit wird durch die großen Modelle verursacht, welche einen großen Bereich der Szene abdecken. Allerdings könnte unserer Meinung nach die Geschwindigkeit durch Anpassung der Szenenunterteilung der Modelle signifikant verbessert werden. Grid-Based und Model-Based Light Assignment haben den Vorteil, dass das Light Assignment für statische Lichter vorberechnet werden kann. Zudem ermöglicht der globale Gültigkeitsbereich des Light Assignment eine Nutzung in Object Space Shading.

Affidavit

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

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First of all, I would like to thank my advisor for his guidance and patience. Particular thank to my supervisor for his ongoing motivation. I am grateful for all the people I have met during my studies in Graz and while working on my master thesis at the ICG. Much appreciation is also due to the staff of the Mensa for providing me with warm healthy meals and to the coffee machine for keeping me awake. Thanks to my friends who showed me that there is something else beyond university. For example, the Sägewerk, which we have visited countless times for some inexplicable reason. Big words of thanks also go to my brother who had once helped me move to Graz and my sister for the extremely fast proofreading of this work. Last but not least I owe the biggest thanks to my hard-working parents for their constant support.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Goals 2

2 Related Work **5**

2.1 Tiled Shading 5

2.2 Clustered Shading 7

2.3 Light Assignment in World Space 9

2.4 Data Structures 10

2.4.1 Light Assignment Input 10

2.4.2 Light Assignment Output 10

2.5 Construction Algorithms 11

2.6 Intersection Tests 12

3 Methods **15**

3.1 Tile-Based Light Assignment 17

3.2 Cluster-Based Light Assignment 18

3.2.1 Depth Index 18

3.2.2 Dense Cluster-Based Light Assignment 19

3.2.3 Sparse Cluster-Based Light Assignment 20

3.3 Grid-Based Light Assignment 22

3.4 Model-Based Light Assignment 22

3.5 Intersection Tests 23

3.5.1 Sphere vs. *AABB* and Sphere vs. *OBB* 24

3.5.2 Sphere vs. Frustum 25

4 Results **27**
4.1 Setup . . . 27
4.2 Light Distribution . . . 28
4.3 Optimization . . . 30
4.3.1 Group Size Parameter . . . 30
4.3.2 Intersection Test . . . 37
4.4 Analysis . . . 37
4.4.1 View Dependency . . . 40
4.4.1.1 Variability of Frame Time . . . 40
4.4.1.2 Depth Discontinuity . . . 41
4.4.2 Performance . . . 46
4.4.2.1 Low, Medium and High Light Count . . . 46
4.4.2.2 Frame Time Depending on Light Count . . . 48
4.5 Limitations . . . 49

5 Conclusion **53**
5.1 Summary . . . 53
5.2 Future Work . . . 55

A List of Acronyms **59**

Bibliography **61**

List of Figures

3.1 Light assignment methods. The leaves represent the acronyms of the methods. 16
3.2 Intersection tests. Each leaf represents the acronym of the corresponding method. 23

4.1 Probability density function of the light radius with mode = 0.25 and mean = 1.0. The distribution is a gamma distribution with the parameters $\alpha = 1.\overline{3}$ and $\beta = 0.75$. 30
4.2 Average frame time of different tile sizes for the tile-based light assignment methods and a light count of 10k. 33
4.3 Average frame time of different tile sizes and cluster z counts for the dense cluster-based light assignment methods and a light count of 10k. 34
4.4 Average frame time of different tile sizes and cluster z counts for the sparse cluster-based light assignment methods and a light count of 10k. 35
4.5 Average frame time of different grid sizes for the grid-based light assignment method and a light count of 10k. 36
4.6 Average frame time of different combinations of sphere vs. frustum intersection tests for the method *T-E*. The light count is 10k and the tile size is $(28, 28)$. 38
4.7 Average frame time of different combinations of sphere vs. frustum intersection tests for the method *SC-E*. The light count is 10k and the group size parameter is $(48, 48, 40)$. 39
4.8 Frame time and the corresponding mean, Standard Deviation (SD) and Relative Standard Deviation (RSD) of the different scenes. (For the method *M* the values of mean, *SD* and *RSD* are 2 259.77 ms, 84.43 ms and 0.04 for Sponza, and 1 700.07 ms, 403.30 ms and 0.24 for Viking Village.) 42

4.9 Two views extracted from the camera path of the Sponza, which differ in the degree of depth discontinuity. 43
4.10 *Left:* Heat map of the light count blended with the textures of the models. The light counts are interpolated from 0 to 150 with the colors blue to red and light counts above 150 are shown in green. *Right:* Histogram of the light count of the groups. 44
4.11 *Left:* Heat map of the light count blended with the textures of the models. The light counts are interpolated from 0 to 150 with the colors blue to red and light counts above 150 are shown in green. *Right:* Histogram of the light count of the groups. 45
4.12 Average frame time of the light counts of 100, 1k and 10k for different methods and scenes. (The average frame times of M for the light counts 100, 1k and 10k are 23.01 ms, 222.48 ms and 2 259.77 ms for Sponza, and 17.49 ms, 170.46 ms and 1 700.07 ms for Viking Village.) 47
4.13 *Top:* Average frame time depending on the light count. *Bottom:* Slope and offset of the fitted linear polynomial. (For the *shading stage*, M has for scenes Robot Lab, Sponza and Viking Village the slopes 9.23 µs/Light, 226.07 µs/Light and 169.57 µs/Light, and the offsets 0.55 ms, −1.76 ms and 0.35 ms. For *all stages*, M has for the scenes Robot Lab, Sponza and Viking Village the slopes 9.32 µs/Light, 226.13 µs/Light and 169.73 µs/Light, and the offsets 0.56 ms, −1.74 ms and 0.37 ms.) 50

List of Tables

3.1 Properties of the light assignment methods. 16
3.2 Short descriptions of the intersection tests. Note that the sphere vs. frustum intersection tests can be combined to reduce the number of false positives. . 24

4.1 Setup of the test system. 28
4.2 Properties of scenes, models, camera paths and light positions. The focus of the properties lies on the Axis-Aligned Bounding Box (AABB) sizes of the different parts. 29
4.3 Selected best group size parameters of each method with the corresponding average frame time of the three scenes. The lowest average frame time in each scene is in **bold** and the second lowest is in *italic*. The selections are based on Figures 4.2-4.5. 32
4.4 Light assignment methods grouped by the magnitude of Standard Deviation (SD) or Relative Standard Deviation (RSD) of the frame time. Each cell is sorted in ascending order. The grouping is based on Figure 4.8. 41
4.5 Light assignment methods grouped by the speed for different light counts. Each cell is sorted by decreasing speed. The grouping is based on Figure 4.12. 48
4.6 Light assignment methods grouped by the magnitude of slope or offset for the stages. Each cell is sorted in ascending order. The grouping is based on Figure 4.13. 49

1 Introduction

Contents

1.1 Motivation 1
1.2 Goals 2

1.1 Motivation

Real-time 3D computer graphic applications, especially games, require visually pleasing render output. Various features like indirect illumination, shadows, depth of field or reflection can add more realism to a scene. Adding many lights to a scene can also increase the visual fidelity and pleasure of a scene. Thus, it is important to find an efficient way to handle many lights. The main performance problem caused by this is redundant light computations of unlit or hidden view samples. Additionally, implementations like traditional deferred shading suffer from a bandwidth problem.

Redundant Light Computations The naive approach of handling many lights is to simply iterate for each view sample over all lights. This results in many redundant light computations, where the view sample is not lit by the light. Another cause of redundant light computations are hidden view samples. For example, the traditional forward shading technique suffers from this so-called overdraw problem.

Bandwidth Problem The traditional deferred shading technique solves the overdraw problem by executing the computations of lighting only for the visible view samples. The stencil buffer optimization reduces the light computations to an absolute minimum. This is achieved by executing the light computations only for the view samples which are within the rasterized light volume [5]. However, the disadvantage of this approach is the high bandwidth requirement resulting from accessing the G-Buffer individually for every light.

Thus, this approach is not suitable for current Graphics Processing Unit (GPU)s due to their relatively slow bandwidth compared to their computation speed.

Light Assignment can be used to address the problems mentioned above. It is a general concept which performs a step before shading in order to assign lights to a group of view samples. The most common shading techniques used in combinations with light assignment are forward or deferred shading. There are different light assignment methods and the most significant difference is the way how view samples are grouped. For example, tiled shading [17] subdivides the screen in tiles, clustered shading [18] subdivides the view frustum into 3D clusters and hashed shading [21] groups the view samples into cells of a world space octree. The redundant light computations are reduced by looking up the light assignment result and only computing the lighting for the visible lights within the group. The bandwidth requirement is much lower than for traditional deferred shading. This is because for the forward variant there are no G-Buffers used and for the deferred variant the G-Buffer is only looked up once for each view sample. The overdraw problem is implicitly solved in the deferred variant and can also be solved for the forward variant by adding a depth pre-pass.

1.2 Goals

Speaking very generally, the inputs for rendering are the 3D scene and the set of lights for each frame. The problem is now to efficiently compute the rendering which requires a fast light assignment and shading. The scope of this work is to compare different ways of grouping view samples together for light assignment. The contributions of this work are the introduced new light assignment methods, the optimization of the best group size parameter and intersection test, the comparison regarding the view dependency and performance and the selection of the best methods fitting for many lights and the best methods for few lights.

Methods The methods which are compared are categorized by the space in which the subdivision is based on. The screen space methods are the tile-based and dense cluster-based and sparse cluster-based light assignment method. For each of these screen space methods two variants are investigated. The first variant uses implicit bounds and the other variant uses explicit bounds for light assignment. We introduce two new light assignment methods for world space and object space. The first is called grid-based light assignment and uses a world space grid to perform the light assignment per call. Although using a world space grid in computer graphics is not new, we think we are the first to introduce it in the context of light assignment. Furthermore, to our knowledge, we are the first to introduce model-based light assignment, which performs light assignment per model.

Optimization Before analyzing the methods in depth, we optimize the group size parameter and the intersection test of each light assignment methods. The goal of the optimization of the group size parameter is to select the value of the group size parameter for each method which minimizes the average frame time of the different scenes. The second part of optimization deals with the intersection tests. The goal is to find a combination of intersection tests which minimizes the number of false positives and the computing time for the intersection test. We do this in a similar manner as optimizing the group size parameter by selecting the combination of intersection tests which minimizes the average frame time over different scenes.

Analysis The analysis consists of investigating the view dependency and performance of the light assignment methods. The view dependency time is measured by the Standard Deviation (SD) and Relative Standard Deviation (RSD) of the frame time, and the dependency on the light count distribution of the light assignment group on the depth discontinuity. The performance is measured by the average frame time for various numbers of lights. For each method, a line is fitted on the measurements to show the linearity and enable comparison. We determine for which light count range a method is suitable and which method performs best for a certain light count range.

2 Related Work

Contents

2.1 Tiled Shading 5
2.2 Clustered Shading 7
2.3 Light Assignment in World Space 9
2.4 Data Structures 10
2.5 Construction Algorithms 11
2.6 Intersection Tests 12

2.1 Tiled Shading

Olsson and Assarsson [17] introduced the simple screen space light assignment method *tiled shading*. The screen is subdivided in tiles with a fixed pixel size. Examples of such tile sizes are 16×16 [3, 14] or 32×32 [17] pixels. The method consists of the two steps light assignment and shading. In the light assignment step, the visible lights are determined for each tile. In the shading step, the lighting is computed for each view sample by using only the lights within the tile. Consequently, the number of light computations are reduced and the performance of shading is highly improved compared to a brute force approach, which takes all lights into account for shading. In this section, the advantages, the disadvantages, and some variations of tiled shading are presented.

Advantages The important advantages of tiled shading are the low bandwidth and the possibility to solve the overdraw problem. Furthermore, it offers a coherent access pattern of lights within a tile during shading, the flexibility to switch between deferred and forward version, and the support of transparency and Multisample Anti-Aliasing (MSAA). One big benefit of tiled shading is the highly reduced bandwidth requirement compared to traditional deferred shading. Traditional deferred shading reads for each lit view sample for

each light from the corresponding entry in the G-Buffer and writes the result of the lighting computations to the A-Buffer. This results in a bandwidth problem, because the read and write operations are performed for each light individually. In tiled shading, the loop order for lights and view samples is switched. Thus, looking up the G-Buffer is performed only once for each view sample which results in a much lower bandwidth requirement. The separation of light assignment and shading enables the flexibility to switch between forward and deferred shading. It is also possible to combine the approaches to support transparency. The opaque geometry is first rendered using deferred tiled shading and then the transparent geometry is rendered using forward tiled shading. The overdraw problem is trivially solved for the deferred version. Adding an extra depth pre-pass can also solve the overdraw problem for the forward version.

Disadvantages The disadvantages of tiled shading are the high view dependency and that shadow maps cannot be reused. During shading, all lights of a tile are used at once, so all the shadow maps of these lights must be available at the same time. Consequently, the shadow maps cannot be reused across these lights, which results in increased memory consumption. The main disadvantage is a high view dependency caused by the unfavorable combination of 2D light assignment and 3D geometry. Depending on the scene, the frame time might be unpredictable, which is highly undesirable for real-time applications.

MinMax By using only implicit bounds for tiled shading, the depth of a tile reaches from the near to the far plane. This is undesirable because the tile is very long in the z-direction and, as a result, light assignment results contain a lot of false positives, i.e. lights which do not influence any actual view sample within the tile. *MinMax* determines for each tile the depth range from the depth buffer [17, 23], which can highly reduce the number of false positives. According to Thomas [23], the tile depth can be faster computed using parallel reduction instead of atomic min/max operations. The problem is that the performance depends on the degree of depth discontinuity. If the geometry within the tile have low depth discontinuity the number of false positives will also be low. However, if the geometry has low depth discontinuity the number of false positives will be higher, and hence shading will be slower.

HalfZ (also called *bimodal clusters* [15]) takes the depth range computed from the depth buffer and splits the range into two halves [23]. Then a list of visible lights is computed for each half. This approach speeds up the shading because the number of visible lights per tile is reduced. However, this approach increases the light assignment work and is still highly influenced by depth discontinuities.

Modified HalfZ takes the two halves produced by HalfZ and further shrinks the depth ranges of the halves [23]. The shrinking is accomplished by looking at all depth values within the tile and computing for each half the minimum and maximum depth value. This

can further decrease the influence of depth discontinuities. However, if there is more than one depth discontinuity at least one half contains one or more depth discontinuities, which leads again to a high number of false positives, and therefore, a high shading time.

2.5D Culling is a more sophisticated technique [11]. Like MinMax this approach first determines the depth range of the view samples for each tile. It subdivides the depth range into divisions and computes a so-called geometry mask, where one bit indicates if the corresponding division contains any geometry. Also for each light, a light mask is computed, where each bit indicates if the light intersects with the corresponding division of the tile depth range. If the bit-wise and of these two masks is non-zero, the light is added to the list of visible lights of the tile. The view dependency is much lower compared to the other variants. However, the usage of implicit bounds for each division results in false positives.

2.2 Clustered Shading

Clustered shading is a screen space light assignment method, which groups view samples with similar attributes into clusters. Olsson et al. [18] proposed two variations on which attributes to chose. In the first variant, the clusters are based only on the quantized 3D position within the view frustum and in the second variant the clusters are additionally based on the quantized surface normal. 3D clusters are more common in the literature [20, 23], and thus we will only focus on the first variant. In this variant, the screen is subdivided by two dimensions, like tiled shading, and the depth is exponentially subdivided in view space. The subdivision can, for example, consist of a tile size of 64×64 with 16 [2] or 32 depth slices [20, 23].

Advantages The main problem of tiled shading is the high view dependency. Even by using a variation like MinMax, it is possible that due to depth discontinuities, tiles can cover a large portion of the view frustum's depth range. 3D clusters, however, are restricted to their local, much smaller bounds, which results in much lower view dependency. The number of lights per cluster is stronger connected to light density, consequently the performance is more predictable. Like tiled shading, clustered shading has a low bandwidth requirement, the flexibility to switch between forward and deferred shading, the possibility to solve the overdraw problem and the easy support of transparency and *MSAA*.

Disadvantages Minor drawbacks of clustered shading are that it is more complicated and requires an increased effort to find good parameter configurations. The most important drawback is the inefficient light assignment caused by the highly different sizes of the clusters in world space resulting from the 3D subdivision. More specifically the problems are the thin near clusters and the same subdivision in x- and y-direction for all depth

values. These problems are addressed by the practical clustered shading and cascaded clustering.

Implicit vs. Explicit Cluster Bounds The implicit cluster bounds are the full bounds of the cell within the view frustum grid without taking any actual view sample into account. The light assignment can be computed based on these implicit bounds. However, this might produce a lot of false positives. Another approach is to use the positions of the view samples and compute the explicit bounds for each cluster. This highly reduces the number of false positives due to the reduced cluster sizes. Furthermore, it is also possible to skip the light assignment of empty clusters.

Dense vs. Sparse Cluster Grid Using a dense cluster grid means that for all clusters in the grid, light culling is executed and memory for the visible lights is reserved. This is a quite simple approach and results in an easy light look up for shading. The disadvantage is that this approach is wasteful, because empty clausters cause unnecessary light culling executions and reservation of unused memory. For a sparse cluster grid, the light assignment is only executed for the visible clusters. Additionally, memory for the visible lights is only reserved for the visible clusters. The cost of finding the visible lights for each cluster becomes lower. Note that the shading time cost stays the same. The downsides of this approach are that the implementation is more difficult and the overhead of finding the clusters is added. For the forward version, a depth pre-pass becomes mandatory because the depth values are needed to identify the visible clusters.

Practical Clustered Shading modifies the clustered shading approach by handling the near and the far clusters differently [20]. In the normal clustered shading, a lot of clusters reside next to the near plane, which results in inefficient light assignment. This approach increases the depth range of the first cluster to decrease the light assignment cost. Additionally, light assignment is not performed for the full view frustum but the maximum depth is restricted to some value below the far plane. Far lights are handled in a different, approximate and more efficient way.

Cascaded Clustering The problem of small clusters in the near plane can also be mitigated by simply reducing the resolution of light assignment grid in x- and y-direction However, due to the uniform x and y subdivision of the whole view frustum clusters will become bigger. Thus, they might contain more false positives and the shading cost increases. *Cascaded clustering* [8] subdivides the frustum along the depth axis into cascades, where each cascade has its individual resolution in x-, y- and z-direction. Consequently, each cascade can be seen as a separate frustum with a limited depth range. This approach now provides a far better view frustum subdivision and results in a more efficient light assignment. The downside is that it is more complicated and choosing good subdivision

parameters, which are the number of cascaded and the resolution for each cascade, gets harder due to the increased possibilities.

2.3 Light Assignment in World Space

Performing light assignment in world space means that the view samples are grouped based on the 3D world space position. It has the advantage that light assignment itself is view-independent, which enables the precomputation of the light assignment for static lights. Compared to screen space methods it is easier to take advantage of frame-to-frame coherency of dynamic lights. It is also possible to use it for object space shading, which needs light assignment results outside the view frustum. A downside is that it might be wasteful and inefficient if used in combination with forward or deferred shading, where shading samples only exist within the view frustum. In this case, world space light assignment will result in unnecessary light assignment outside the view frustum. Also, because the subdivision is applied to the whole scene, the resolution might be too coarse within the actual view frustum, which results in inefficient shading. In the next paragraphs, we describe related work, which uses world space subdivision in a regular grid and octree.

Regular Grid To our knowledge, there is no work about a grid-based world space method used for light assignment. However, there are other approaches in the context of ray tracing or occlusion culling, which take advantage of a 3D grid to accelerate computations. Garanzha and Loop [10] use a virtual 3D world space grid to implement ray sorting, which they need for their ray tracing algorithm. For ray sorting the ray origin and the ray direction is quantized and combined to a hash value. Sorted rays will be more coherent and produce a better memory access pattern. Batagelo and Wu [7] use a 3D grid as acceleration structure for occlusion culling. Each voxel within the grid identifies local features like opacity, occlusion and spanned objects of the scene. For each frame, the grid is traversed in an approximate front-to-back order to compute a conservative set of visible object. The overall goal of this approach is to accelerate the visualization of complex dynamic scenes.

Hashed Shading is a shading technique, where the light assignment is performed as a precomputation step in a linkless octree in world space [21]. The advantage of this approach is the view independency and the frame-to-frame reuse of the light assignment result. The octree enables the adaption of cell sizes which results in more efficient shading. A big disadvantage is that hashed shading does not support dynamic lights and has a very high memory consumption. Consequently, this approach might need some more research to provide a practical solution for many dynamic lights.

2.4 Data Structures

The light assignment receives the lights as an input and stores the assigned lights of the groups as an output. The choice of the data structure of the input and the output highly influences the performance of the light assignment or shading. In this section examples for input and output data structures will be presented.

2.4.1 Light Assignment Input

The input of the light assignment consists of the lights needed for light assignment. The underlying data structure can highly affect the speed of the light assignment and can be categorized into linear and hierarchical types. In this section, we will present one example for each type. The first is the simple linear list and the other is the Bounding Volume Hierarchy (BVH).

Linear List The most simple way is to use a linear list of lights. This has the advantage that no additional step is required to build the data structure. The linear list can be used by different light assignment algorithms. For example, for analytical testing [12, 17] or rasterization of light volumes [19]. The problem of this simple data structure is that at the beginning of light assignment nothing is known about at the lights itself or the relationship between them. Thus, it is necessary to iterate over all elements of the linear list.

Bounding Volume Hierarchy Olsson et al. [18] perform light assignment using a BVH of all lights in the scene. The leaves represent the lights with a light sphere and each parent represents the union of its child nodes by a Axis-Aligned Bounding Box (AABB). The advantage of this data structure is that due to the hierarchy more lights can be eliminated with a single intersection test. Thus, the light assignment speed is increased. As a consequence of the additional overhead added by constructing the BVH and the cost of traversing the tree it is not suitable for using a scene with a small number of lights.

2.4.2 Light Assignment Output

The light assignment stores the visible lights of each group as an output. The data structure of the output determines the speed of shading and can be categorized into linear and hierarchical types. We will present three examples. The first example deals with the linear list and the other two examples deal with tiled light trees.

Linear List The most straightforward way is to simply store the light assignment result in a linear list. Each group occupies a consecutive range within the linear list, where its visible lights are stored. This enables an easy construction and shading process. The disadvantage is that, because no additional information of the lights is known, a view

sample within has to iterate over all visible lights within the group. It is possible to either store the light data directly [4] in the linear list or just store an index [12, 17, 18]. The indirect index version uses less memory but more bandwidth is required for shading due to the indirect lookup. For the direct version, it is the other way round.

Tiled Light Trees for Tiled Shading O'Donnell and Chajdas [16] introduced the so-called *Tiled Light Trees* with two variants. The first variant uses the light trees in combination with tiled shading. For each tile, a binary tree is computed, which is sorted by the center along the z-direction in view space. During shading, each pixel traverses the corresponding tree to compute the lighting. Due to the tree traverse potentially not all lights of a tile are used for computing the lighting. The shallow trees which are also sorted in depth-first order are well suited for the Graphics Processing Unit (GPU). One drawback is the more complicated construction. Moreover, tiled light trees are slower than conventional clustered shading under some view conditions, where the traverse of the tree creates too much overhead.

Tiled Light Trees for Clustered Shading Another variant of tiled light trees combines the light trees with clustered shading [16]. The view frustum is subdivided into cells just like clustered shading. First, for each cell, the list of visible lights are computed and stored in a linear list. A heuristic decides if a light tree should be computed for a cell. This variant does not suffer from slow down due to the additional overhead of the tree traverse under certain view conditions like the first variant does. The disadvantage is that the implementation gets more involved. For example, threads within a tile might create divergence by executing a cell with a linear list and a light tree at the same time. The implementation must handle that case in an efficient way.

2.5 Construction Algorithms

The core of the light assignment algorithm is the construction algorithm. This is the step where the visible lights are determined and stored. There are different ways on how to classify the construction algorithm. It is possible to distinguish if the construction is performed on the Central Processing Unit (CPU), on the *GPU* or on both. The methods could also be classified if they are gathering and scattering algorithms. A gathering algorithm gathers the visible lights for each group and a scattering does exactly the opposite by scattering each light to the affected groups. We decided to choose a more practical classification, which is used by Archer et al. [4]. In the following, we distinguish between analytical, rasterization and hybrid, which combines the two.

Analytical The most obvious way of constructing the visible lights is by iterating over the light assignment input data structure and analytical test [11, 12, 16, 17] the lights for an intersection. Analytical testing means that an intersection test of a light and a

light assignment group is explicitly done in code. For example, if it should be determined whether a finite point light intersects with a tile in tiled shading, the sphere of the light and frustum of the tile are tested with an analytical intersection test.

Rasterization The idea of determining the visible lights by rasterization is to rasterize the light volumes in a conservative way to find the affected groups. Örtegren [19] uses rasterization in combination with clustered shading. Arbitrary light volumes are supported by this approach. The only requirement is that the light volume can be represented as a convex mesh. The approach consists of the shell pass followed by the fill pass. Firstly, in the shell pass, the shell of the light volume is rasterized to the cluster grid. Thus, for a certain screen space tile, only the first cluster and the last cluster along the z-axis are hit. Secondly, the fill pass makes sure that the clusters in between are also affected by the light.

Hybrid *Hybrid lighting* [4] is a hybrid between an analytical approach and rasterization and is applied to clustered shading. Billboard quads of the lights are rasterized to find the tile intersections of the light. Then the depth range is computed of the light volume within the tile in an analytical way. The depth range is used to add the lights to the affected clusters within the depth range. The performance can be improved by adding a depth pre-pass. It is determined which cells actually contain geometry and add lights only to non-empty cells.

2.6 Intersection Tests

Intersection tests determine if two given objects intersect or not. Note that these two objects can be of different type. Common examples for a type of an object are ray, plane, sphere, cylinder, cone, triangle, $AABB$, Oriented Bounding Box (OBB), frustum or polyhedron [1, ch. 22]. For light assignment, the intersection tests can be used to test whether or not a light intersects with a light assignment group. We will discuss the tests which we need for our light assignment methods, namely sphere vs. $AABB$, sphere vs. OBB and sphere vs. frustum.

Sphere vs. *AABB* and Sphere vs. *OBB* can be performed by the same underlying method because the latter can be transformed into the former. Testing sphere vs. $AABB$ for intersection is equal to check if the distance of the nearest point in or on the $AABB$ to the sphere center is smaller or equal to the sphere radius [6]. The nearest point can be simply found by clamping the sphere center to the $AABB$. The sphere vs. OBB intersection test is very similar because an OBB can be seen as a rotated $AABB$. By applying the inverse rotation of the OBB to the OBB and the sphere the result is an $AABB$ and a sphere. Therefore, this problem can be reduced to sphere vs. $AABB$ [1, p. 977].

Sphere vs. Frustum can be tested with different approaches. We show two approaches using half spaces and the Separating Axis Theorem (SAT). For the half space method, we simply use the half spaces defined by the frustum planes to decide whether or not the sphere potentially intersects or not. For the *SAT* method a set of candidate axes is used to test if any of the axes separates the sphere and the frustum. Thatcher [22] uses the following candidate axes: frustum edges, plane normals, vertices to sphere center, nearest vertex to sphere center. Taking into account that we have the half space method and the *SAT* method with four different candidate axes we end up with five different tests. The problem is that each of these tests might produce false positives. The idea is to combine the intersection tests in such a way that light assignment is fast and the number of false positives is low. According to Thatcher [22], the half spaces method combined with *SAT* method using the nearest vertex is the best combination.

3 Methods

Contents

3.1 Tile-Based Light Assignment 17

3.2 Cluster-Based Light Assignment 18

3.3 Grid-Based Light Assignment 22

3.4 Model-Based Light Assignment 22

3.5 Intersection Tests 23

This chapter presents the different light assignment methods and the used intersection tests. Before the methods are explained the basic assumptions are discussed, which are the finite lights volume and the dynamic geometry of lights. The assumption of a finite light volume is not physically correct but still results in plausible results and enables efficient light assignment. Furthermore, it is assumed that the lights are dynamic. That means that the geometry of lights can be totally different from one frame to another. Thus, no precomputation of the light assignment is performed.

Each of the light assignment methods consists of the light assignment step and the shading step. Both steps are completely implemented on the Graphics Processing Unit (GPU). This work only focuses on the light assignment methods and, thus only one shading technique, namely forward shading, is implemented. The most significant difference between the methods is how the view samples are grouped together to a light assignment group. The presented methods are screen space, world space grid, and object space methods. The screen space methods either perform light assignment per 2D tile or 3D cluster. The world space method performs light assignment per cell in a world space grid and the object space method performs light assignment per model. Figure 3.1 gives an overview of the different methods. Table 3.1 summarizes their properties, which are discussed in the corresponding section of each method.

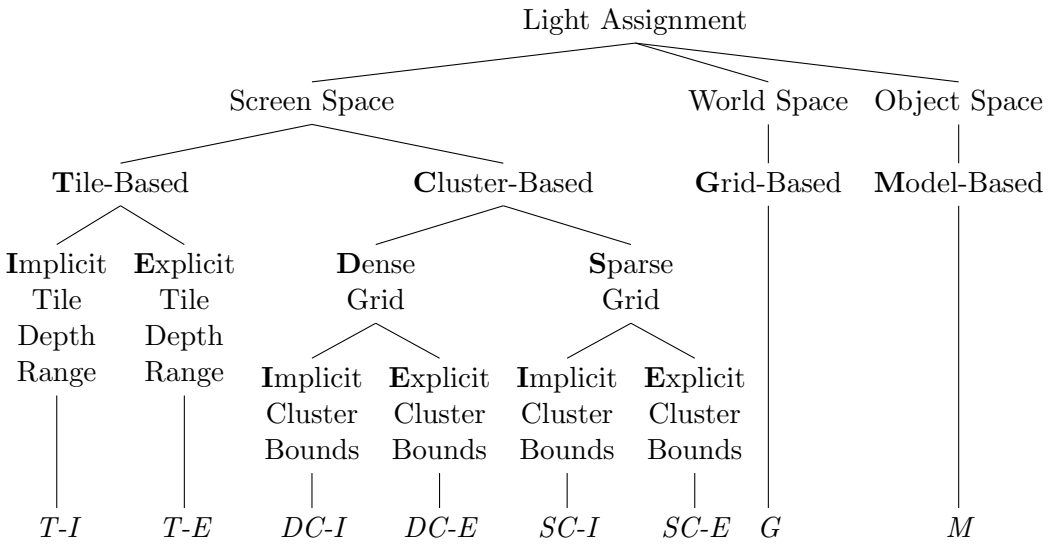


Figure 3.1: Light assignment methods. The leaves represent the acronyms of the methods.

Method	*T-I*	*T-E*	*DC-I*	*DC-E*	*SC-I*	*SC-E*	*G*	*M*
Space	Screen Space						World Space	Object Space
Intersection Test	Sphere vs. Frustum						Sphere vs. *AABB*	Sphere vs. *OBB*
Group	Tile		Cluster		Visible Cluster		Cell	Model
Parameters	Tile Size		Tile Size, Z Count				Cell Count	-
Scope	View Frustum	Samples	View Frustum	Samples	Samples	Samples	Scene	Models (≈ Scene)
Explicit Bounds	✗	✓	✗	✓	✗	✓	✗	✗
Depth Pre-Pass	✗	✓	✗	✓	✓	✓	✗	✗
Visibility and Explicit Bounds Stage	✗	✓	✗	✓	✓	✓	✗	✗

Table 3.1: Properties of the light assignment methods.

3.1 Tile-Based Light Assignment

We introduce the term tile-based light assignment as the light assignment process which assigns lights to screen space tiles of fixed pixel size. It is equal to tiled shading presented in Section 2.1, however, the name emphasizes the light assignment step instead of the shading step. Two variants are implemented. The first variant uses an implicit depth range, and the other uses an explicit depth range. By implicit depth range, we mean the full depth range from the near to the far plane. The explicit depth range covers only the depth range of the view samples within the corresponding tile. This explicit variant is the same as the MinMax optimization for tiled shading. The different stages of this method are the depth pre-pass, the explicit tile depth range stage, the light culling stage and the shading stage. Note that the first two stages are only executed for the explicit version. In the following, the data structures and the stages will be presented.

Data Structures All the data structures are linear lists. The light buffer is the input of the light culling stage and is a linear list of the light data of all lights. The light data consist of color and the geometry description of the light. For example, for a point light, the geometry description consists of position and radius. The visible light index buffer is the output of the light culling stage and is a linear list of light indices. For each tile, a fixed amount of entries is reserved at a fixed offset in the buffer. Additionally, for the explicit version, there are also the depth buffer and the tile depth range buffer. The depth buffer is the output of the depth pre-pass and stores the depth value for every pixel. The tile depth range buffer is the output of the tile depth range stage and stores the explicit tile depth range for every tile.

Tile Depth Range Stage The tile depth range stage computes the explicit tile depth range in Normalized Device Coordinates (NDC) for each tile using the depth buffer and stores it in the tile depth range buffer. Therefore, this stage requires a depth pre-pass and is only used for the explicit version. Each work group of the shader works on one tile. Note that the work group size is not equal to the tile size, because the work group size is restricted and we want to support arbitrary tile sizes. Each thread works on a disjoint subset of fragments within the tile. First, each thread computes the depth range of a subset of pixels by looking up the depth buffer and performing min and max operations. Then the shared variable depth range is initialized to $[\infty, -\infty]$. The variable is used by the threads within a work group to compute the depth range using atomic min and max operations. The result is then stored in the tile depth range buffer. Note that a tile is considered empty if its tile depth range is equal to the initialization value.

Light Culling Stage The task of the light culling stage is to compute the visible light indices of each tile and store them in the visible light index buffer. The found visible light indices are stored as consecutive entries. The end is marked with the invalid light

index -1. Note that instead of marking the end with -1, it would also be possible to explicitly store the number of visible lights of each tile in a buffer. Each work group works on one title. Now, the steps are discussed. For the explicit version, the tile depth range is retrieved. If the tile is empty the intersection tests are not performed and the visible light index buffer is marked empty. For the implicit version, the full tile depth range from the near to the far plane is used. The world space frustum is computed. After that, the threads iterate in parallel over all lights and use the sphere vs. frustum intersection test to decide if the light should be added in the visible light index buffer or not.

Shading Stage The shading stage computes the color for each view sample. For each view sample, the following steps are performed. The tile is computed by simply subdividing the screen space coordinates by the tile size. Then the lighting is computed by iterating over all visible lights of the tile and accumulating the lighting of a single light on the view sample. In the end, the color is computed by applying the texture of the model to the accumulated lighting.

3.2 Cluster-Based Light Assignment

Cluster-based light assignment is a screen space light assignment method which performs the light assignment in view-frustum aligned 3D grid. Hence, it is equal to clustered shading [18] with 3D clusters. The screen space is subdivided in tiles of fixed size like in tile-based light assignment. Additionally, the depth is exponentially subdivided in the view space. We provide two different base variants, which differ in used data structures of the grid. The first variant uses a dense grid, where light culling is executed for all clusters. The second variant uses a sparse grid, where light culling is only executed for the visible clusters. In the following sections, the derivation of the depth index and the two variants for dense and sparse grid is shown.

3.2.1 Depth Index

The formula we use for computing the depth index from the depth value differs from the one presented by Olsson et al. [18]. In the following, the formula for the depth index k for the given depth value in view space z_{vs} is derived. k also depends on the number of subdivisions in z-direction s_z and the view space depth value n of the near plane and the view space depth value of f of the far plane.

For exponential subdivision, the recursion formula of the minimum depth value n_k of the k-th slice is

$$\begin{aligned} n_0 &= n \\ n_k &= bn_{k-1},\ k \geq 1, \end{aligned} \tag{3.1}$$

where b is the base factor. Resolving this recursion formula results in

$$n_k = b^k n. \tag{3.2}$$

Using $f = n_{s_z} = b^{s_z} n$, the formula for the base factor b is

$$b = \sqrt[s_z]{\frac{f}{n}}. \tag{3.3}$$

For the depth value z_{vs} within the k-th slice,

$$n_k \leq z_{\text{vs}} < n_{k+1} \tag{3.4}$$

holds. Further, we know that $f > n$ holds. Therefore, by using Equation 3.3, $b > 1$ holds. This implies that $\log_b v$ is monotonic increasing for an arbitrary v with $v > 0$. This property and $n > 0$ further implies

$$\log_b \frac{n_k}{n} \leq \log_b \frac{z_{\text{vs}}}{n} < \log_b \frac{n_{k+1}}{n}. \tag{3.5}$$

By substituting n_k and n_{k+1} using the Equation 3.3, we get

$$\log_b \frac{b^k n}{n} \leq \log_b \frac{z_{\text{vs}}}{n} < \log_b \frac{b^{k+1} n}{n}. \tag{3.6}$$

This can be reduced to

$$k \leq \log_b \frac{z_{\text{vs}}}{n} < k + 1. \tag{3.7}$$

This implies

$$k = \left\lfloor \log_b \frac{z_{\text{vs}}}{n} \right\rfloor. \tag{3.8}$$

Thus, we derived a formula for the depth index k.

3.2.2 Dense Cluster-Based Light Assignment

The first base variant of cluster-based light assignment uses a dense grid. This means that the light culling stage is executed for all clusters and also that in the light assignment output, entries are reserved for all clusters. Like for tiled shading, there is also an implicit and an explicit version. The implicit version uses the full clusters for the intersection test. The explicit version uses the explicit Axis-Aligned Bounding Box (AABB) in *NDC* of the clusters for the intersection test. The stages are the depth pre-pass, the cluster bounds stage, the light culling stage and the shading stage. Note that the first two stages are only used for the explicit version. In the next paragraphs, the data structures and the stages are presented.

Data Structures The data structures are similar to those of tile-based light assignment. There is the light buffer, which stores the linear list of lights, and the visible light index buffer where for each cluster, a fixed range is reserved. For the explicit version, there is also the depth buffer and the cluster bounds buffer, which stores an $AABB$ in NDC for every cluster.

Cluster Bounds Stage The task of the cluster bounds stage is to use the depth values to compute the explicit $AABB$ in NDC for every cluster. Thus, this step requires a depth pre-pass and is only used for the explicit version. Before the shader is executed, every $AABB$ of the cluster bounds buffer is initialized to $[\infty, -\infty]^3$. Each thread works on one fragment. First, the cluster index is computed using the depth value of the fragment. The fragment $AABB$ is computed and is applied to the cluster $AABB$ in the cluster bounds buffer using atomic min and max operations. Note that if the cluster $AABB$ is equal to the initialization value after executing this stage, the cluster is considered empty.

Light Culling Stage The light culling stage computes the list of visible lights for each cluster and stores them in the visible light index buffer. Each work group operates on one cluster. For the explicit version, the explicit bounds are retrieved and if the cluster is not visible the light culling stage is aborted. For the implicit version, the implicit cluster bounds are computed. The cluster frustum in world space is constructed. All threads check in parallel for intersection using sphere vs. frustum intersection testing. If an intersection test returns true the thread atomically increases the visible light count. The old value is used as a relative index into the visible light index buffer to store the index of the light.

Shading Stage The shading stages compute the output color for each view sample. First, the cluster index of the view sample is computed. Then the list of visible lights of the cluster is iterated over to compute the lighting, which is a simple Phong lighting. The texture is applied to the lighting and the output color is written to the framebuffer.

3.2.3 Sparse Cluster-Based Light Assignment

Sparse cluster-based light assignment is a variant of cluster-based light assignment where the grid storing the visible light indices is sparse. Compared to the dense version the light culling stage is only executed for the visible cluster. Furthermore, memory in the light assignment output is only reserved for the visible clusters. Like for the dense version, there is also one version with implicit cluster and one with explicit cluster bounds. The stages are the depth pre-pass, the cluster bounds and cluster visibility stage, the light culling stage and the shading stage. Note that the first two stages are used for both versions because the list of visible clusters is determined before light culling. This section shows the data structures and stages of sparse cluster-based light assignment.

Data Structures The used data structures are similar to the dense version. There is the light buffer, the depth buffer and the visible light index buffer. The visible light index buffer, however, stores only entries for visible clusters. The visible cluster count buffer stores the number of visible clusters. It is set to 0 before the stages are executed. The visible cluster indices buffer is a linear list which stores the cluster indices of the visible clusters. There is no distinct buffer for the cluster bounding boxes, but they are stored in the cluster information buffer. The cluster information buffer stores attributes for each cluster. These attributes are the visibility flag, which indicates if a cluster is visible, the cluster bounding box, which is based on the view samples, and the slot within the visible cluster indices, which is also used as an offset in visible light index buffer. The visibility flag is set to 0 and the cluster bounding box is set to $[\infty, -\infty]^3$ before executing the stages.

Cluster Bounds and Cluster Visibility Stage The cluster bounds and cluster visibility stage consists of two steps. In the first step, the cluster bounds are computed. The first step is only performed for the explicit version and is performed the same way as for the dense version. The only difference is that the bounding boxes are stored in the cluster information buffer. The second step is mandatory for both the explicit and implicit version. Each thread works on one fragment. First, the cluster index is computed using the depth value in the depth buffer. Then an atomic exchange operation on the visibility flag in the cluster information buffer is performed. The new value of the visibility flag is 1. If the old value is 1, then another thread has already set the visibility flag and no further operations have to be performed. If the old value is 0, then the current thread has to perform further operations. The visible light count is atomically increased by 1 and the old value is stored as slot index in the cluster information buffer.

Light Culling Stage The light culling stage computes the visible lights for each visible cluster. The difference to the dense version is that the light culling stage is executed only for the visible clusters and the additional level of indirection in order to look up the cluster index. Each work group works on one visible cluster. Note that each visible cluster is identified by its slot index. First, the cluster index is looked up in the visible cluster indices using the slot index. The next steps are the same as for the dense version. The lights are tested in parallel and the visible lights are added to the visible light indices buffer.

Shading Stage The shading step computes the color for each view sample. Again, this step is very similar to the dense version. The difference is the indirect lookup of the slot index. Each thread computes the cluster index. Then the slot index is looked up in the cluster information buffer. The next steps are basically the same as for the dense version. The lighting is computed, using the visible lights of the visible clusters. The color is computed using the accumulated lighting and the texture color of the model.

3.3 Grid-Based Light Assignment

We introduce the new world space method grid-based light assignment. The whole scene is subdivided in a regular world space grid and light assignment is performed for each cell. Although the concept of using a world space grid as acceleration structure is not new [7, 10], to our knowledge, we are the first to introduce it in the context of light assignment. We employ a pure world space approach. Thus, we do not use a depth pre-pass to compute the implicit cell size. Therefore, we present only the implicit version and no explicit version. The data structures used are the light buffer and the visible light index buffer, which stores the visible light indices for each cell. The method consists of the light culling stage and the shading stage, which are discussed in the following.

Light Culling Stage The task of the light culling stage is to compute the list of visible lights for every cell. Each work group works on one cell. First, the cell $AABB$ in world space is computed. Then all threads check in parallel the lights using sphere vs. $AABB$ intersection test, which is discussed in Section 3.5.1. The result is then added to the visible light index buffer.

Shading Stage The goal of the shading stage is to compute the color of every view sample. First, the cell index is determined. Then the lighting is computed by iterating over the visible lights of the cell. The output color is the combination of the accumulated lighting and the texture color of the model.

3.4 Model-Based Light Assignment

We present the novel object-space method named model-based light assignment. The model bounds defined by the scene data are used to perform a per-model light assignment. A pure object space approach is employed, and thus no explicit bounds are computed. The stages are the light culling stage and the shading stage. They are similar to the other methods and are presented below.

Data Structures The data structures consist of the light buffer, the visible light index buffer, the model matrix buffer and the model bounds buffer. The visible light index buffer contains the visible lights of each model. The model matrix buffer contains the model matrices of each model. We assume that each model matrix M has the form $M = T \cdot R \cdot S$, where S is a scale matrix, R is a rotation matrix and T is a translation matrix. This enables a faster sphere vs. Oriented Bounding Box (OBB) intersection test, which is described in Section 3.5.1. The model bounds buffer stores the model $AABB$ in object space for each model.

Light Culling Stage The light culling stage determines the visible lights for every model. Each work group operates on exactly one model. First, the model matrix and object space *AABB* of the model are retrieved. The threads check in parallel the lights for intersection using sphere vs. *OBB* intersection test. Note that the intersection test takes the light sphere, the model matrix and the model *AABB* as input. The result is stored in the visible light index buffer.

Shading Stage The shading stage computes the color of each pixel based on the light culling result from the previous stage. For each model, an independent rasterization run is executed and the model index is passed as uniform to the shader. The lighting is computed by iterating over the visible lights of the model. The texture of the model is applied to the accumulated lighting and the result is written to the framebuffer.

3.5 Intersection Tests

An intersection test decides if two given object intersect or not. For light assignment, intersection tests are used to decide whether or not a light intersects with a light assignment group. Furthermore, using intersection tests for light assignment implies that the objects must be solid. This means that we consider the boundary and the interior for the intersection test. Our implementation only supports point lights, which are represented by spheres. The light groups are either frustums for screen space, *AABB*s for world space or *OBB*s for object space light assignment methods. Thus, the intersection tests we need are sphere vs. frustum, sphere vs. *AABB* and sphere vs. *OBB*. This chapter will explain these intersection tests in more detail. Figure 3.2 shows an overview of the intersection tests and Table 3.2 provides a short description of the intersection tests.

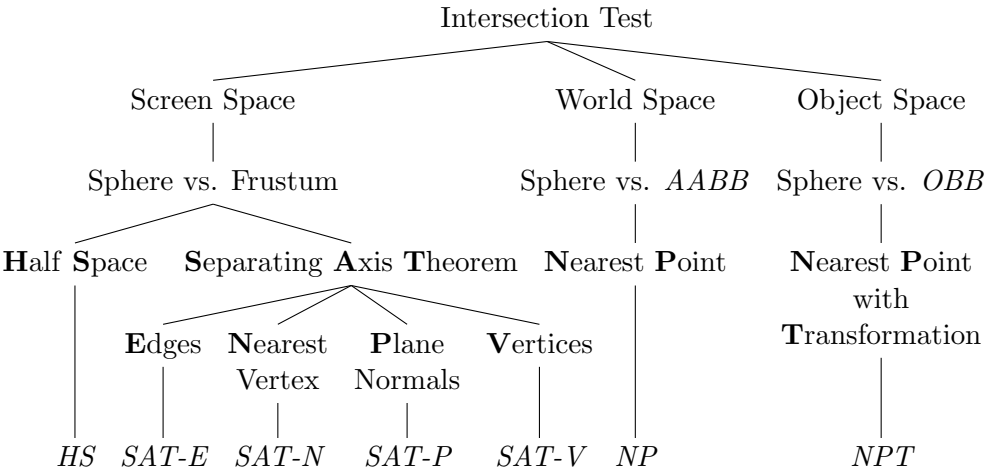


Figure 3.2: Intersection tests. Each leaf represents the acronym of the corresponding method.

Type	Method	Description	
Sphere vs. *AABB*	*NP*	The **n**earest **p**oint in or on the *AABB* to the sphere center is used to test for intersection.	
Sphere vs. *OBB*	*NPT*	**N**earest **p**oint method with **t**ransformation.	
Sphere vs. Frustum	*HS*	The **h**alf **s**paces defined by the planes of frustum are used to test if the sphere is potentially partially inside the frustum.	
	SAT-E	Checks if at least one of the following axes is a separating axis:	frustum **e**dges.
	SAT-N		center of sphere to **n**earest vertex.
	SAT-P		normals of frustum **p**lanes.
	SAT-V		center of sphere to **v**ertices.

Table 3.2: Short descriptions of the intersection tests. Note that the sphere vs. frustum intersection tests can be combined to reduce the number of false positives.

3.5.1 Sphere vs. *AABB* and Sphere vs. *OBB*

The methods to test sphere vs. *AABB* and sphere vs. *OBB* are very similar. They produce no false positives and no false negatives. Initially it will show how to solve sphere vs. *AABB* using the simple nearest point method. Then it will be explained how sphere vs. *OBB* can be transformed to sphere vs. *AABB*.

Sphere vs. *AABB* with Nearest Point Method To test sphere vs. *AABB* the nearest point method is used. At first we find the point P which is in or on the *AABB* and is nearest to the sphere center. This can easily be achieved by clamping the sphere center to the *AABB*. Then we compute the distance of P to the sphere center. If this distance is smaller or equal to the radius there is an intersection, otherwise not [6].

Sphere vs. *OBB* with Nearest Point with Transformation Method The intersection test for sphere vs. *OBB* can be transformed to sphere vs. *AABB*. An *OBB* can be defined as an *AABB* which was transformed using a matrix M. We restrict the matrix M is to the from $M = T \cdot R \cdot S$, where S is the scale matrix, R the rotation matrix and T the translation matrix. This is an adaption for the use case for model-based light assignment. Note that it is still possible to perform every sphere vs. *OBB* test because every *OBB* can be expressed as an *AABB* with a matrix of the form above. First, we decompose the given matrix into the translation matrix T, the rotation matrix R and the scale matrix S. The transformation $(R \cdot T)^{-1}$ is applied to the sphere which results again in a sphere. The radius stays unchanged and it is sufficient that the sphere center is transformed by $(R \cdot T)^{-1}$. The transformation S is applied to the *AABB* of the *OBB* which results in a scaled *AABB*. The translated and rotated sphere and the scaled *AABB* can now be used

as input for the sphere vs. $AABB$ intersection test. Therefore, it was shown how to test sphere vs. OBB using sphere vs $AABB$ and transformation.

3.5.2 Sphere vs. Frustum

The sphere vs. frustum intersection tests are more involved as the tests before. In this section, the half space method, the Separating Axis Theorem (SAT) method and the combination of these methods are presented.

Half Space Method The first method of testing sphere vs. frustum is called the half space method. We assume that the normals of the planes of the frustum point outwards. If the sphere is at least partially inside all negative half spaces of the frustum planes, then the intersection test reports true, otherwise false. Note that this method can result in false positives.

Separating Axis Theorem Method The hyperplane separation theorem states that for any two non-empty disjoint convex sets in n-dimensional Euclidean space there exists a separating hyperplane [9]. This means that there exists a separating axis which is orthogonal to the hyperplane to which the convex sets can be projected and the projected intervals of the convex sets do not overlap. We can use this SAT to decide the intersection of two convex sets. We test a set of finite candidate axes until we find a separating axis. If we do not find a separating axis we assume that the convex sets overlap. The test might produce false positives because it is possible that none of the candidate axes is a separating axis, although there exists a separating axis. Note that in our example the two convex sets are the sphere and the frustum. There are multiple possibilities for candidate axes. We use the axes sets proposed by Thatcher [22]. These are the edges, the axes from vertices to the sphere center, the axis from the nearest vertex to the sphere center and the plane normals.

Combination The half space method and all four methods of the SAT potentially produce false positives. False positives do not change the rendered output itself, but they cause unnecessary light computations. This increases the shading costs. Simply combining all tests will result in the least amount of false positives but the time cost for the light assignment will also significantly increase. Thus, we try out different combinations and evaluate which performs the best in terms of minimizing the light assignment and shading time.

4 Results

Contents

4.1 Setup 27
4.2 Light Distribution 28
4.3 Optimization 30
4.4 Analysis 37
4.5 Limitations 49

In this chapter, we provide the results of this work. The most important parts are the optimization and analysis of the methods. In the beginning, the test system, the scenes, and the camera paths are described. We define a distribution for the light which will be used for all scenes. Next, the best values for group size parameters and the best combinations of intersection tests are empirically selected. The methods are analyzed by view dependency and performance. In the end limitations of our work are discussed. Note that acronyms for the light assignment methods, which are introduced in Figure 3.1 and for intersection tests, which are introduced in Figure 3.2 are used in this chapter.

4.1 Setup

First, the setup of the evaluation is described. The configuration of the test system is shown in Table 4.1. The scenes were rendered using a resolution of 1920 × 1080 (Full HD). Moreover, the methods were implemented on the Graphics Processing Unit (GPU) using the Vulkan® Application Programming Interface (API). In the following section, the properties of the scenes and the models are presented. Additionally, the properties of the camera path and the bounding box of the light position are discussed.

Scene and Models The characteristics of the scenes with their models are presented in Table 4.2. The properties of the model subdivision are indicated by the model count

Operating System	Linux® 64 Bit (Kernel Version: 4.14.92-1-MANJARO)
CPU	Intel® Core™i5-4670K CPU @ 3.40GHz
GPU	NVidia® GeForce® GTX 780
NVidia® Driver Version	415.25
Vulkan® *API* Version	1.1.84

Table 4.1: Setup of the test system.

as well as the average and variance of the bounding box size of the models. It can be observed that for certain axis and scenes there is a very high variance, which indicates highly diverse model sizes. Note that the model sizes have a high impact on model-based light assignment.

Camera Path and Light Position Table 4.2 also lists properties of the camera path and the bounding box of the light position. Each scene has its own camera path. This is rendered and the frame time is measured for the evaluation. Furthermore, there is the bounding box of the camera path and the lights. The bounding box of the camera path is computed from camera positions. The bounding box of the lights defines the bounds in which the lights are placed. Note that the relation between these two bounding boxes gives an indication of the performance of grid-based light assignment. The bigger the light bounding box is compared to the camera path bounding box the higher the chances that unnecessary and too coarse light assignment is performed by grid-based light assignment. For the scenes Robot Lab and Sponza we choose the light bounding box to be equal to the scene bounding box. The scene bounding box of Viking Village is relatively large compared to the bounding box of the camera path. Hence, we choose the light bounding box of that scene in a different way. We scale each side of the camera position bounding box by the arbitrary value 1.5 to get the light position bounding box.

4.2 Light Distribution

This section focuses on the selection of the light distribution. The only supported light types are point lights, which light sphere is fully defined by the position and the radius. For positions, we simply choose to uniformly distribute them in the light position $AABB$. The selection of the distribution of the radii is more involved and will be discussed in this section. First, we talk about what we expect from a radius distribution. A proper light distribution is picked, namely the gamma distribution. Then we choose the mode and mean and derive formulas to compute the distribution parameters based on the mode and mean.

		Robot Lab	Sponza	Viking Village
Scene	#Triangles	470 781	262 249	4 260 750
	#Vertices	382 101	184 366	2 874 121
	AABB Size	(33.2, 15.7, 48.8)	(37.2, 15.6, 22.9)	(1 089.0, 132.9, 866.4)
Models	#Models	645	392	1 215
	AABB Size Average	(2.9, 1.9, 3.0)	(2.1, 0.8, 1.0)	(4.8, 2.5, 4.6)
	AABB Size Variance	(29.9, 5.5, 48.0)	(35.3, 1.7, 7.0)	(642.2, 31.2, 610.5)
Camera Path	Position *AABB* Size	(7.9, 1.7, 19.4)	(15.9, 5.8, 9.7)	(57.8, 5.1, 46.4)
	#Frames	1200	1200	1200
Light Position	*AABB* Size	Scene *AABB* (33.2, 15.7, 48.8)	Scene *AABB* (37.2, 15.6, 22.9)	$1.5^3\times$ Camera *AABB* (86.7, 7.6, 69.5)

Table 4.2: Properties of scenes, models, camera paths and light positions. The focus of the properties lies on the Axis-Aligned Bounding Box (AABB) sizes of the different parts.

Expectations Before a distribution of the light radii can be selected, it is necessary to discuss what is expected of the radii. We expect a lot of small lights, a medium amount of medium-sized lights and a very low amount of large lights. Very large lights which affect the whole scene should not be handled by light assignment but by a different system.

Chosen Solution: Gamma Distribution Based on the expectations we choose the gamma distribution for the light radii. The probability density function of the gamma distribution is defined as

$$p(x|\alpha,\beta) = \frac{1}{\Gamma(\alpha)\,\beta^{\alpha}} x^{\alpha-1} e^{\frac{-x}{\beta}} \text{ [13].} \tag{4.1}$$

We choose mode = 0.25 and mean = 1.0 to get the distribution shown in Figure 4.1. Thus, the selection matches our expectations. The values around mode show the small lights and the probability of large and medium-size lights can be influenced by moving mean. Note that the values are not relative but absolute values.

Derive Distribution Parameters For the sake of completeness, we will derive formulas of the distribution parameters α and β based on mode and mean. mode and mean of the gamma Probability Density Function (PDF) given in Equation 4.1 can be expressed depending on α and β by

$$\begin{aligned} \text{mode} &= (\alpha-1)\beta \text{ and} \\ \text{mean} &= \alpha\beta \text{ [13].} \end{aligned} \tag{4.2}$$

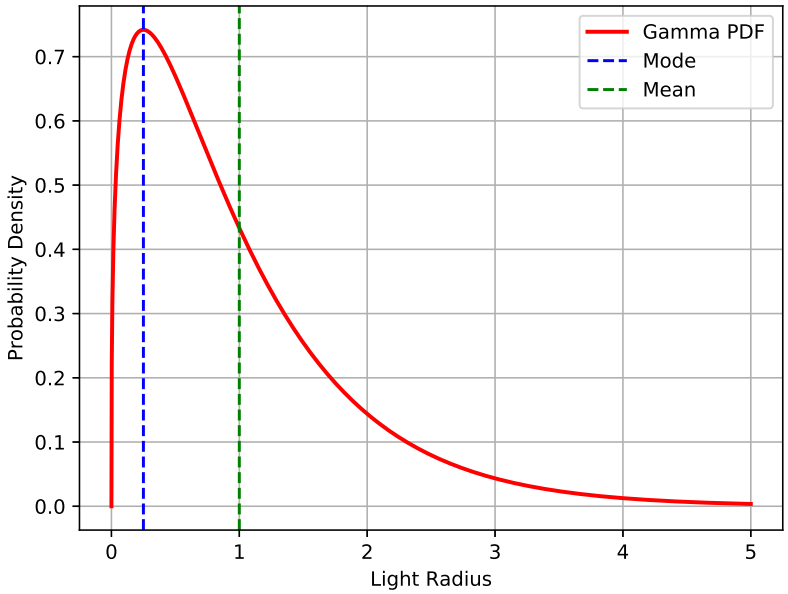


Figure 4.1: Probability density function of the light radius with mode = 0.25 and mean = 1.0. The distribution is a gamma distribution with the parameters $\alpha = 1.\overline{3}$ and $\beta = 0.75$.

This can be used to derive the formulas for the parameters α and β:

$$\begin{aligned} \alpha &= \frac{\text{mean}}{\text{mean} - \text{mode}} \quad \text{and} \\ \beta &= \text{mean} - \text{mode}. \end{aligned} \tag{4.3}$$

By applying the equations to the given mode = 0.25 and mean = 1.0 we get $\alpha = 1.\overline{3}$ and $\beta = 0.75$.

4.3 Optimization

The goal of the optimization is to find the best configurations for all methods. First, the group size is optimized and based on this the best tile-based and cluster-based method is selected. These methods are then used to search for the best combination of intersection tests.

4.3.1 Group Size Parameter

The goal of the group size parameter optimization is to select the parameter value which performs the best across all scenes. The group size parameter of each light assignment method defines the geometric size of the light assignment group, and this is why they

highly influence the performance. The specific group size parameter of the methods is the tile size for the tile-based method, the combination of tile size and cluster z-count for the cluster-based methods and the grid size for the world-space method. Note that the model-based light assignment has no group size parameter, and hence we will not optimize this method. In the following, we explain the plots, which show the average frame times of the different methods and scenes. We discuss the different cases of the measured values, which require different selection strategies. Then, the best value of the group size parameter is selected for each method. In the end, the results are summarized and the best tile-based and cluster-based methods are selected.

Plots The plots shown in Figures 4.2-4.5 are set up to enable the selection of the best parameter for each method. Each plot is specific to one method and one scene and compares the average frame time of the camera path for different parameter values. The average frame time is used as characteristic because it provides a good indication of the performance. As light count 10k is chosen, because the light assignment method should be optimized for many lights. For the intersection test *HS+SAT-N* (Half Space Method Combined with Separating Axis Theorem (SAT) Nearest Vertex), which is recommended by Thatcher [22] is used. The plots compare a fixed set of parameter values. Different values were chosen beforehand and the values are narrowed down to the presented ones.

Cases The methods can be divided into three cases, depending on how many scenes possess the same best parameter value for that method. Note that by best parameter value we understand the one that has the lowest average frame time. For each of these cases, selecting the best parameter has a different level of difficulty and needs a different strategy. For example, if at least two out of the three scenes have the same best parameter, we could simply select a parameter by a majority decision. The cases and the methods which are of this case will be discussed in the following paragraphs.

Case 1: All Scenes Have The Same Best Parameter Value If all three scenes have the same best parameter value, the decision is quite easy and the selected value is equal to this best parameter value. The only method which falls into this case is *T-I* (see Figure 4.2), where the parameter value $(28, 28)$ is the fastest for all scenes.

Case 2: Two Scenes Have The Same Best Parameter Value The second case is when two of the three scenes have the same best parameter value. By using a majority decision, the best parameter value of the method is easily selected as the parameter value which is the best for two of the three scenes. However, it might be the case that for the third scene this parameter value performs significantly worse than the other parameter values for this scene. In this instance, the majority decision might be malfunctioning. Note that this is not the case for the evaluations we performed. For *T-E* (see Figure 4.2), *DC-I* (see Figure 4.3), and *DC-E* (see Figure 4.4), there is a parameter value which is best

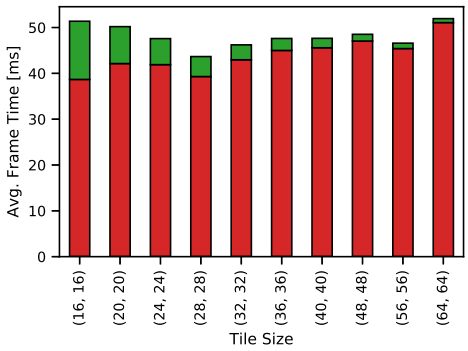
for two scenes. As for the third scene it is second best with a very similar average frame time. The selected values are $(28, 28)$ for *T-E*, $(128, 128, 64)$ for *DC-I* and $(64, 64, 40)$ for *DC-E*. For *SC-E* (see Figure 4.4) and *SC-I* (see Figure 4.4), there is also very familiar situation. For two scenes the best parameter value is the same. For the third, it is the fourth best and has a very similar average frame time. The selected values are $(48, 48, 40)$ for *SC-I* and $(48, 48, 40)$ for *SC-E*.

Case 3: All Scenes Have a Different Best Parameter Value In the third case, there is no parameter value which is the best for two or more scenes. If every scene has a different best parameter then the majority decision cannot be applied. The only method which falls into case 3 is the method *G* (see Figure 4.5). For Robot Lab the best parameter value is $(18, 18, 18)$, for Sponza it is $(22, 22, 22)$ and for Viking Village it is $(20, 20, 20)$. We select $(20, 20, 20)$, which performs well for all scenes.

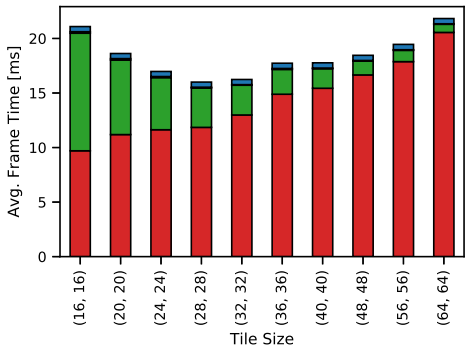
Summary At the end of the group size parameter optimization, the findings are collected and used to select the best tile-based and cluster-based method. The selected values of the three different cases can be seen in Table 4.3. This table also shows the average frame time of the selected parameter value. The best method of the tile-based methods is *T-E* because it has the lowest average frame time. We select *SC-E* as the best cluster-based method. Note that *SC-E* is the only second best cluster-based method in terms of the average frame time. However, numbers are very similar to the fastest method *DC-E*. The advantage of the sparse version is that for normal scenarios it needs far less memory for storing the visible lights per cluster, and thus it can support more lights then its dense counterpart.

Method	Parameter Name	Selected Configuration	Avg. Frame Time of Selected Configuration [ms]		
			Robot Lab	Sponza	Viking Village
T-I	Tile Size	(28, 28)	43.66	73.44	66.76
T-E		(28, 28)	16.00	18.82	23.05
DC-I	Tile Size, Z Count	(128, 128, 64)	25.82	31.64	27.05
DC-E		(64, 64, 40)	**9.14**	**8.87**	**9.50**
SC-I		(48, 48, 40)	12.93	12.27	12.07
SC-E		(48, 48, 40)	*9.75*	*9.75*	*9.90*
G	Cell Count	(20, 20, 20)	26.93	43.18	30.20

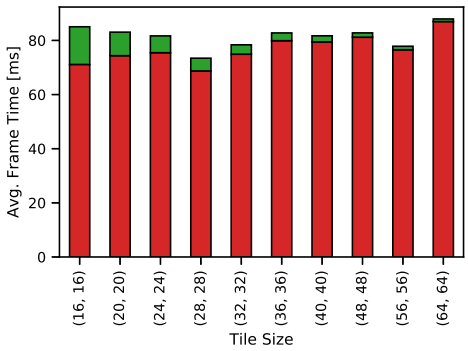
Table 4.3: Selected best group size parameters of each method with the corresponding average frame time of the three scenes. The lowest average frame time in each scene is in **bold** and the second lowest is in *italic*. The selections are based on Figures 4.2-4.5.



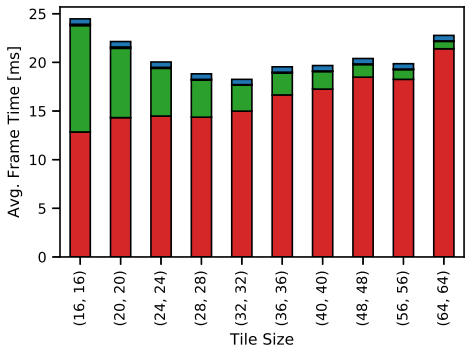
(a) Scene: Robot Lab
Method: *T-I*



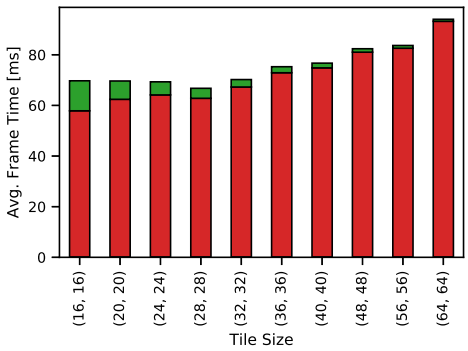
(b) Scene: Robot Lab
Method: *T-E*



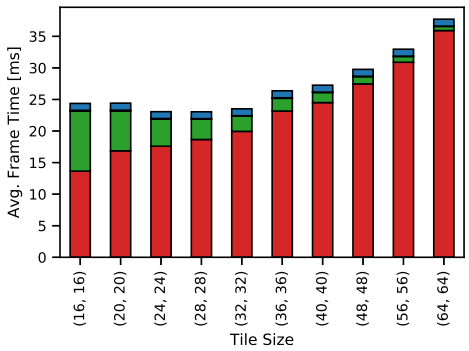
(c) Scene: Sponza
Method: *T-I*



(d) Scene: Sponza
Method: *T-E*



(e) Scene: Viking Village
Method: *T-I*



(f) Scene: Viking Village
Method: *T-E*

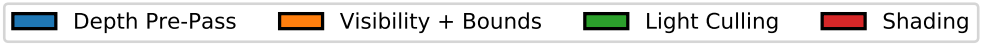
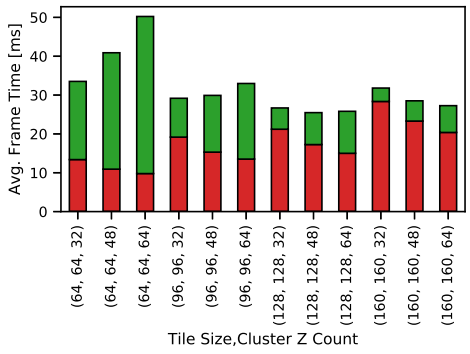
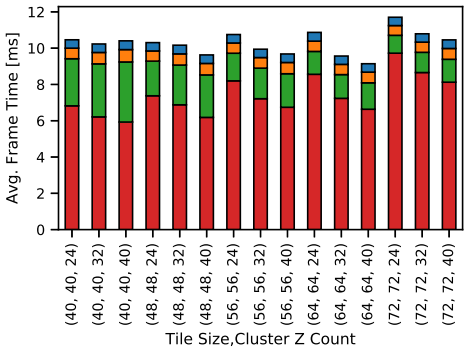


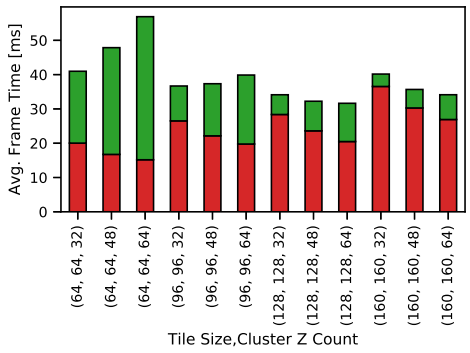
Figure 4.2: Average frame time of different tile sizes for the tile-based light assignment methods and a light count of 10k.



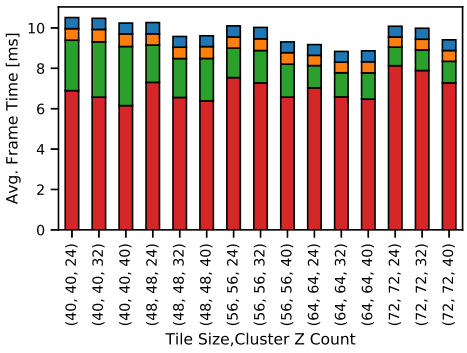
(a) Scene: Robot Lab
Method: *DC-I*



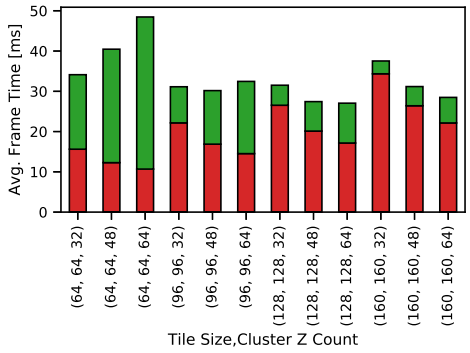
(b) Scene: Robot Lab
Method: *DC-E*



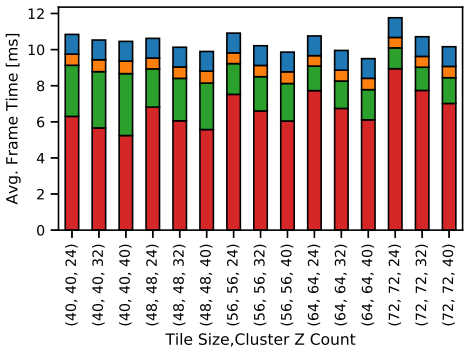
(c) Scene: Sponza
Method: *DC-I*



(d) Scene: Sponza
Method: *DC-E*



(e) Scene: Viking Village
Method: *DC-I*



(f) Scene: Viking Village
Method: *DC-E*

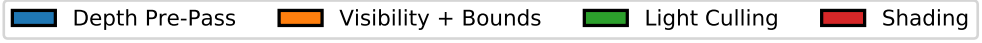
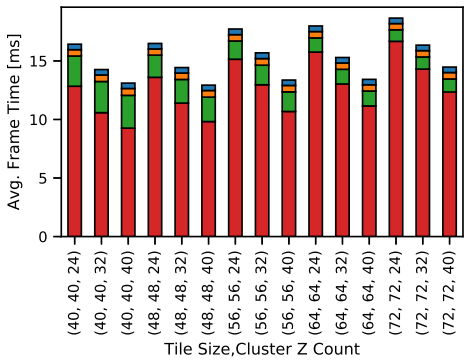
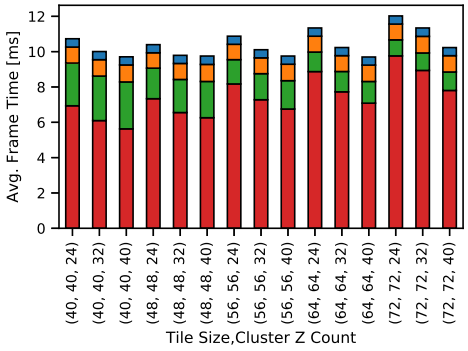


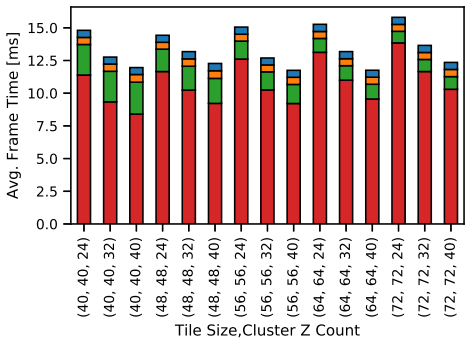
Figure 4.3: Average frame time of different tile sizes and cluster z counts for the dense cluster-based light assignment methods and a light count of 10k.



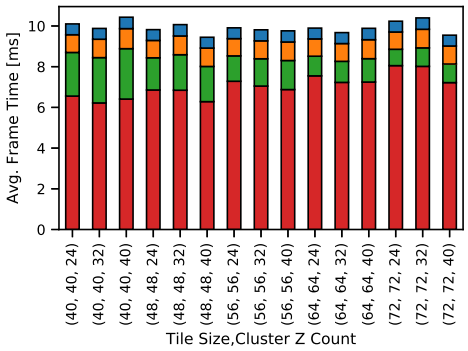
(a) Scene: Robot Lab
Method: *SC-I*



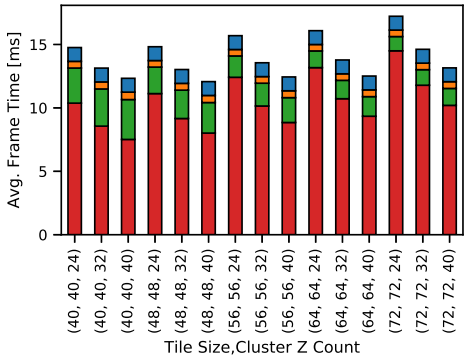
(b) Scene: Robot Lab
Method: *SC-E*



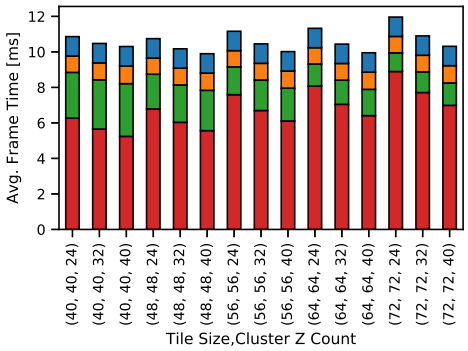
(c) Scene: Sponza
Method: *SC-I*



(d) Scene: Sponza
Method: *SC-E*



(e) Scene: Viking Village
Method: *SC-I*



(f) Scene: Viking Village
Method: *SC-E*

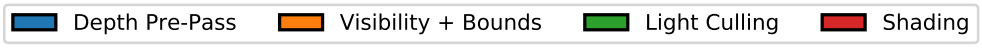
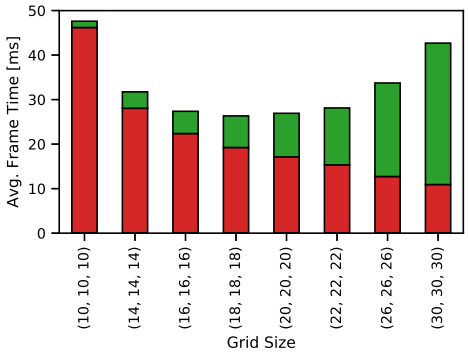
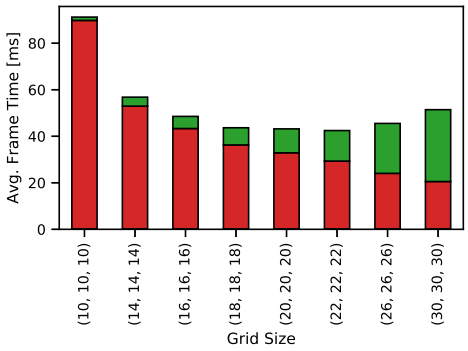


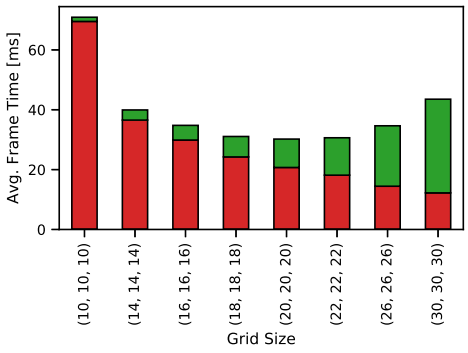
Figure 4.4: Average frame time of different tile sizes and cluster z counts for the sparse cluster-based light assignment methods and a light count of 10k.



(a) Scene: Robot Lab
Method: G



(b) Scene: Sponza
Method: G



(c) Scene: Viking Village
Method: G

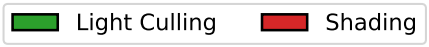


Figure 4.5: Average frame time of different grid sizes for the grid-based light assignment method and a light count of 10k.

4.3.2 Intersection Test

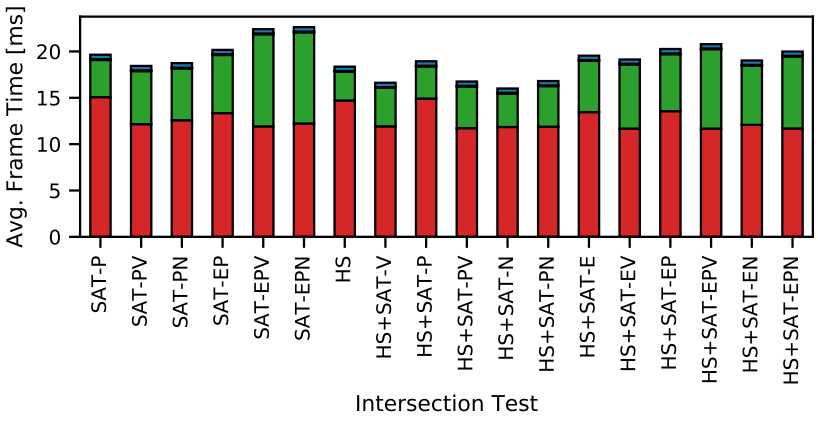
The goal of optimizing the intersection tests is to find the combination of sphere vs. frustum intersection tests which minimize the average frame time. Thus, the objective is to minimize the false positives but still perform efficient light assignment. Note that this optimization is only necessary for screen space methods because they all need to test sphere vs. frustum, for which there are multiple methods where each single one might produce false positives. For each, the object space method and the world space method, there is only one simple method which does not produce false positives regarding the given geometric shapes. Furthermore, we will only perform this optimization for the best tile-based *T-E* and the best cluster-based method *SC-E*. In this section, first the plots will be explained and then the best combination of intersection test for *T-E* and *SC-E* is selected.

Plots The plots for *T-E* can be seen in Figure 4.6 and for *SC-E* in Figure 4.7. They show the average frame time for different intersection test combinations. The light count is 10k and the group size parameters are the ones found in Section 4.3.1. Note that we do not test all possible combinations of intersection tests. It does not makes sense to test combinations where for the *SAT* method both, the nearest vertex and all vertices are tested. Specifically for the tile-based light assignment, not to take the frustum planes into account, results in a high number of false positive. Consequently, we do not test combinations where both, the half space method and also the *SAT* method with plane normals are not used.

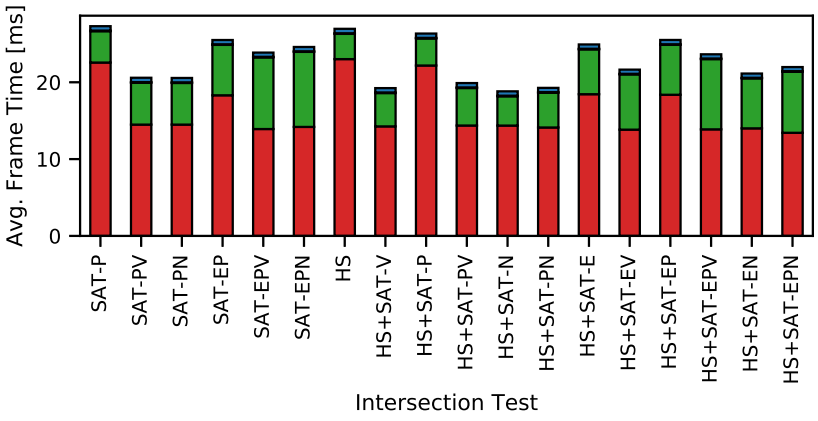
Best Intersection Test Selecting the best combination of intersection tests is very similar to the selection of the group size parameter. For *T-E* *HS+SAT-N*is the best for all three scenes. For *SC-E* *HS+SAT-N*is the best for two scenes and is also among the best for the third scene. Consequently, we choose *HS+SAT-N*as the best intersection test for both methods. Furthermore, because these two methods are very similar to the other screen space methods, we expect that *HS+SAT-N*is also the best for those methods. Note that the shading result of the full combination *HS+SAT-EPV* is very similar to that of *HS+SAT-N*. This indicates that *HS+SAT-N* does not contain that many false positives. The advantage of *HS+SAT-N* compared to *HS+SAT-EPV* is that the light assignment is faster, and hence the frame time is lower. Note that our findings match with the results of Thatcher [22], who also reported *HS+SAT-N* as the best performing intersection test.

4.4 Analysis

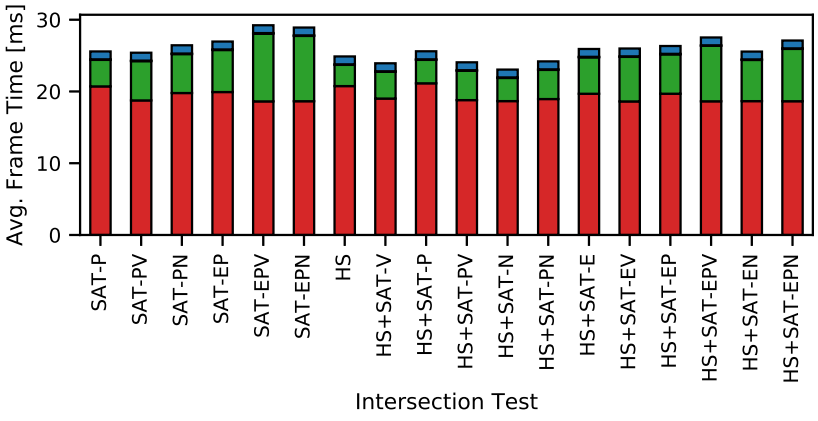
The goal of the analysis is to investigate the view dependency and the performance of each method. The evaluation uses the optimized group size parameter and intersection test, which were determined in the previous section. In this section, initially the view dependency and then the performance will be studied.



(a) Scene: Robot Lab



(b) Scene: Sponza



(c) Scene: Viking Village

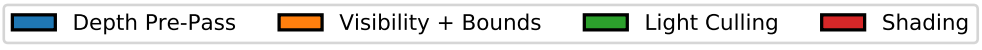
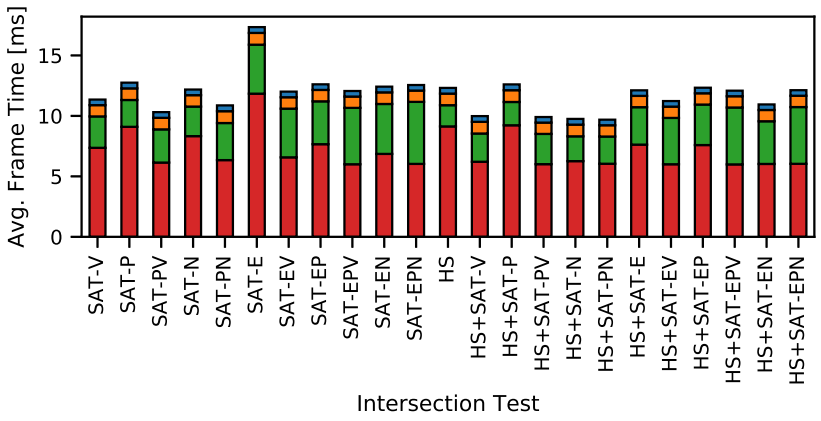
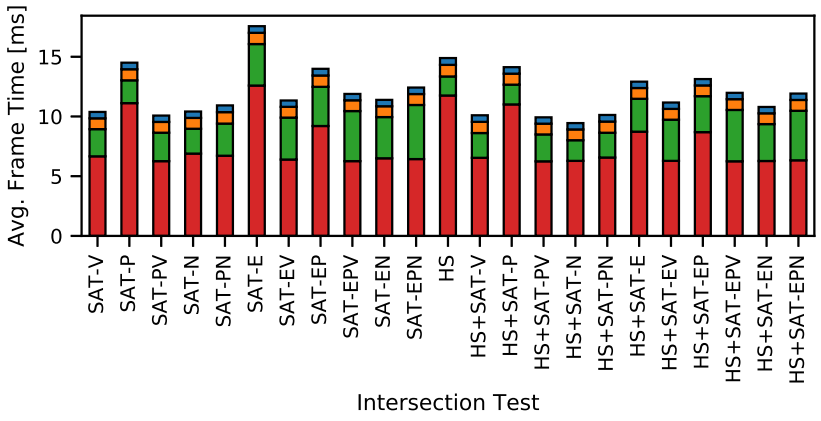


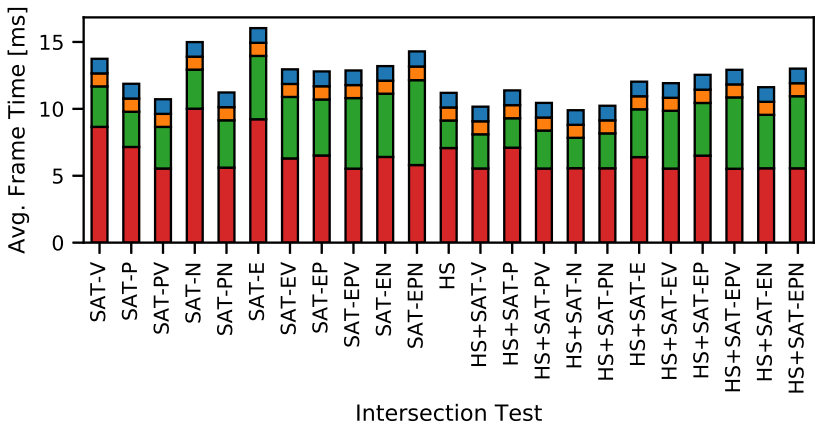
Figure 4.6: Average frame time of different combinations of sphere vs. frustum intersection tests for the method *T-E*. The light count is 10k and the tile size is $(28, 28)$.



(a) Scene: Robot Lab



(b) Scene: Sponza



(c) Scene: Viking Village

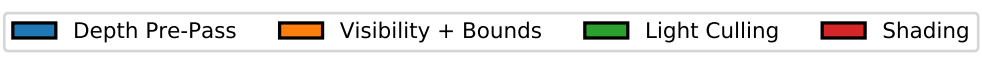


Figure 4.7: Average frame time of different combinations of sphere vs. frustum intersection tests for the method *SC-E*. The light count is 10k and the group size parameter is $(48, 48, 40)$.

4.4.1 View Dependency

View dependency of the frame time is an important aspect of a light assignment method. It determines how predictable the frame time is. Note that predictable frame times are an important requirement of real-time applications. In the following, the Standard Deviation (SD) and the Relative Standard Deviation (RSD) of the frame time are investigated. Then the impact of depth discontinuities on the light count distribution of the light assignment groups are analyzed.

4.4.1.1 Variability of Frame Time

The view dependency of the frame time can be seen as the variability of the frame time. The variability can be measured, for example by the *SD* or the *RSD*. For the evaluation, the camera paths of the scenes are rendered for different methods and many lights (light count = 10k). This measured frame time is used to compute the values of *SD* and *RSD*. In the following, we group the light assignment methods by the magnitude of *SD* and *RSD* and compare them.

Standard Deviation The *SD* of the different methods and scenes are shown in Figure 4.8. The *SD* and *RSD* of this Figure are grouped by the magnitude and can be seen in Table 4.4. *M* and *T-I* show a very high *SD*. *M* shows an extremely high *SD*, which is not meaningful due to the high mean. *G*, *DC-I* and *T-E* have medium *SD*. Interestingly, *G* and *DC-I* have similar values among the scenes. Also note, the reduction of the *SD* from *T-E* to *T-I*. The sparse cluster methods *SC-I* and *SC-E* have the lowest *SD*s. Again, the explicit version has a lower *SD*.

Relative Standard Deviation Figure 4.8 shows, beside the *SD*, also the *RSD*, which is the *SD* divided by the mean value. In other words, the *RSD* gives a measure for the variability relative to the mean value. Compared to the *SD* there is not such a big diversity among the values of the methods. However, we still categorize the values into high, medium and low *RSD*. *T-I* and *T-E* have the highest *RSD*, where the explicit version has a lower value. *DC-I* and *SC-I* have medium values. The methods with the lowest values in declining order are *SC-E*, *G* and *M*. Remarkably, *M* has the lowest relative and at the same time the highest *SD*. However, this is due to the enormous mean value of *M* which results in the low *RSD*. Additionally, it can be observed that the explicit versions have lower *RSD* then the implicit versions. Also, *SC-E* has a low *RSD* and at the same time the lowest *SD*.

Summary The basic conclusion is that every method shows a variability of frame time regarding different views. In other words, all presented methods are view-dependent. From the high difference of *SD* and modest difference of *RSD* values between implicit and explicit versions, we conclude that the explicit version has lower view dependency. *DC-I*

and *G* have similar *SD* values. Hence, they have a similar degree of view dependency. There is a modest difference in the *RSD* values, which can be explained by the higher mean value of *G*. An important observation is that *SC-I* and *SC-E* show the lowest view dependency. *SC-E* is the best method regarding the view dependency and has low *SD* and low *RSD*. Also interesting is that *M* has extremely high *SD*, and thus it has a very high view dependency. *M* has the lowest *RSD* but still a very high *SD*, which we consider the more important value. Consequently, we conclude, that *M* is highly view-dependent.

	SD	*RSD*
Low	*SC-E*, *SC-I*	*G*, *M*, *SC-E*, *SC-I*
Medium	*T-E*, *DC-I*, *G*	*DC-I*
High	*T-I*, *M*[a]	*T-E*, *T-I*

[a]Very High

Table 4.4: Light assignment methods grouped by the magnitude of Standard Deviation (*SD*) or Relative Standard Deviation (*RSD*) of the frame time. Each cell is sorted in ascending order. The grouping is based on Figure 4.8.

4.4.1.2 Depth Discontinuity

In this section, we investigate the influence of the depth discontinuity on the light count distribution of the light assignment groups. The light count distribution is important because it is an indicator of how good the light assignment is and of how fast the shading is. We only study the following methods: *T-E*, *SC-E*, *G* and *M*. For each of these methods, we examine views with low depth discontinuity and a view with high depth discontinuity. This views can be seen in Figure 4.9. For each of the two views and the four methods, we created a heat map and a histogram of the light group count. In the following text, the different methods are analyzed, compared and in the end, the results are summarized.

Tile-Based Light Assignment with Explicit Bounds Figure 4.10 shows the heat maps and histograms for the methods *T-E* and *SC-E*. For *T-E* and low depth discontinuity, the most groups are in the light count range of 0 to 25. For high depth discontinuity, a lot of groups are still in the range 0 to 25. However, a lot of groups are in the range 25 to 75. Some groups are even be above 75. The heat map indicates that a lot of groups with a high light count are accessed , and as a result the frame time is high.

Sparse Cluster-Based Light Assignment with Explicit Bounds The method *SC-E* shows different behavior to *T-E*. For both methods, the histogram for low discontinuity is similar, i.e. the most groups are in the range 0 to 25. For the view with high discontinuity, there are still a lot in the range 0 to 25 and only a few in the range 25 to 50. It looks as if the light count distribution has moved a little bit to the right. The heat map reveals

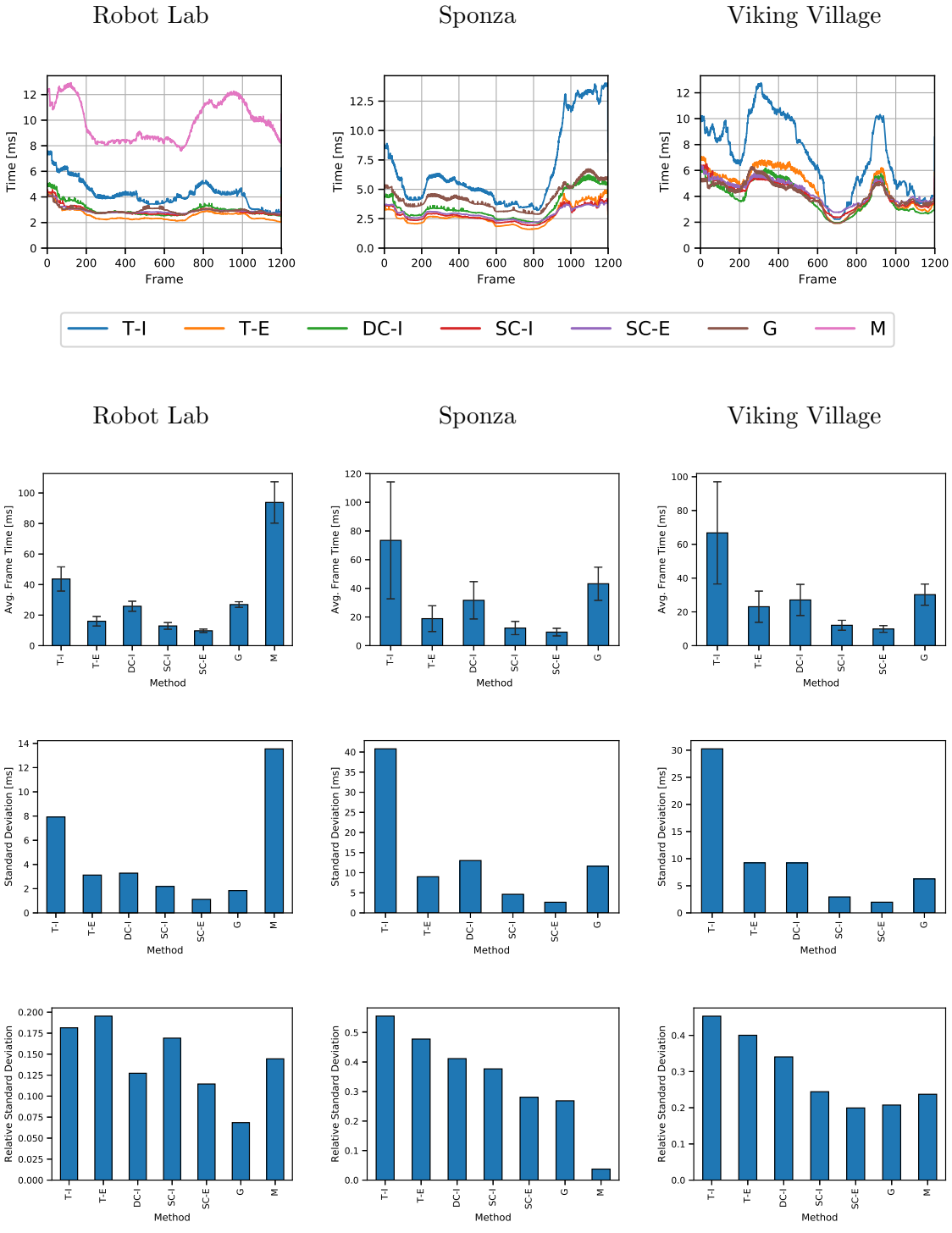


Figure 4.8: Frame time and the corresponding mean, *SD* and *RSD* of the different scenes. (For the method *M* the values of mean, *SD* and *RSD* are 2 259.77 ms, 84.43 ms and 0.04 for Sponza, and 1 700.07 ms, 403.30 ms and 0.24 for Viking Village.)

that groups of the shaded pixels have a much lower light count then for *T-E*. According to that, *SC-E* is less dependent on the depth complexity than *T-E*.

Grid-Based Light Assignment The heat map and histogram of the light assignment methods *G* and *M* are shown in Figure 4.11. For the *G* we marked all groups with light count 0 into a distinct bin. There are a lot of groups with light count 0 because the method uses a dense grid. For both views of *G*, a lot of groups have light counts 1 to 25 and also 25 to 75. Note that both views have the same light count distribution because the method operates on the whole scene and the light assignment itself is independent of the view. The heat map reveals that a lot of groups with a medium amount of lights are accessed. As a consequence, the performance is not as good as *SC-E*.

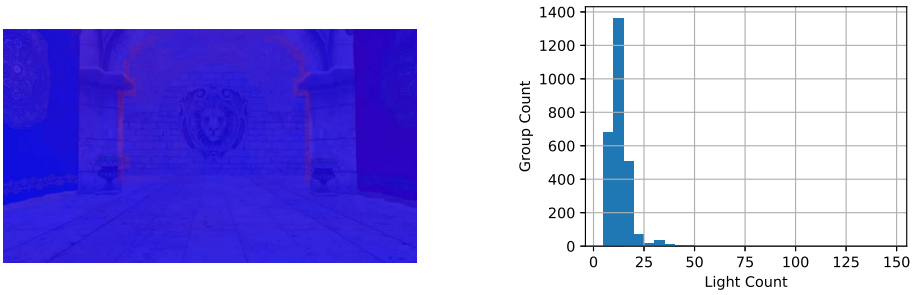
Model-Based Light Assignment For the other method *M* the two histograms are also equal. Again for the same reason, that *M* performs light assignment globally per scene. The light count distribution, however, is quite different compared to *G*. There are a lot of groups with the light count 0 to 50. This alone might result in at least moderate performance. However, the problem is that this method produces groups with an extremely high amount of lights. The heat maps of *M* show that this group with the extremely high amount of lights are accessed. This translates into a very high frame time.

Summary In the end, we summarize the finding of the light count distributions. Both, *T-E* and *SC-E* have a view-dependent light count distribution. However, *SC-E* has only a low dependency and *T-E* has a high dependency. Because the light count distribution mostly contains a small number of lights, *SC-E* has the best shading performance of the four methods. The light count distribution of *G* and *M* is view-independent, because their light assignment is view-independent. Due to the wide light count distribution, *G* performs worse than the screen space methods. For the method *M*, the outliers with the high light count have a very negative impact on the frame time.

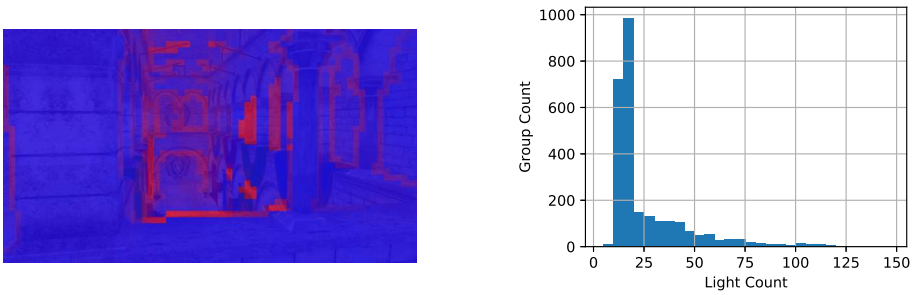
(a) View with low depth discontinuity. (Frame: 125)

(b) View with high depth discontinuity. (Frame: 1 161)

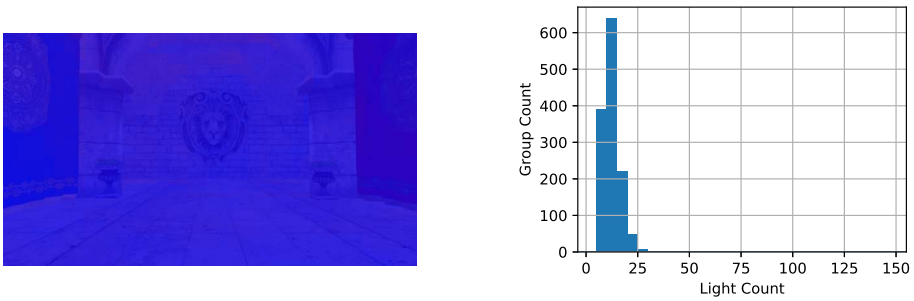
Figure 4.9: Two views extracted from the camera path of the Sponza, which differ in the degree of depth discontinuity.



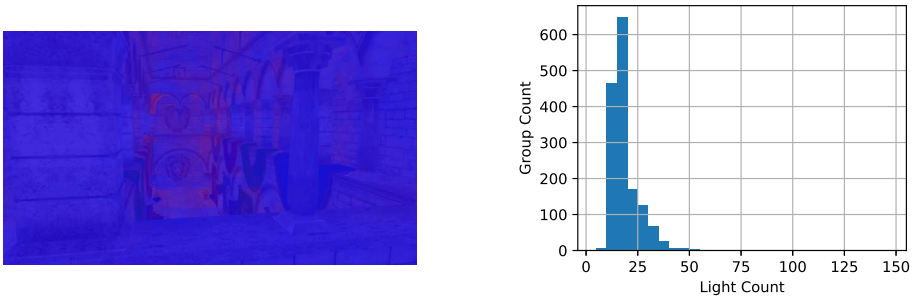
(a) Depth Discontinuity: Low, Method: *T-E*



(b) Depth Discontinuity: High, Method: *T-E*

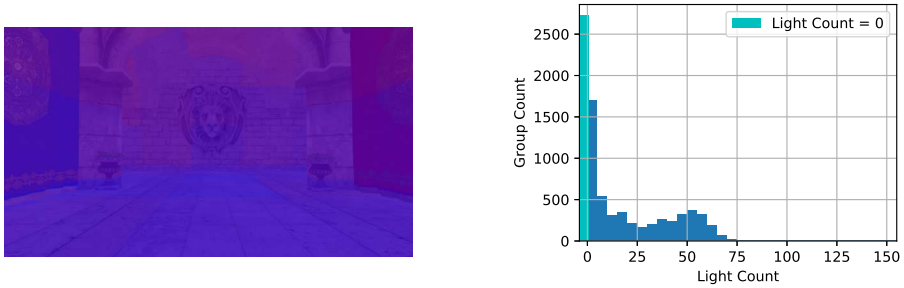


(c) Depth Discontinuity: Low, Method: *SC-E*

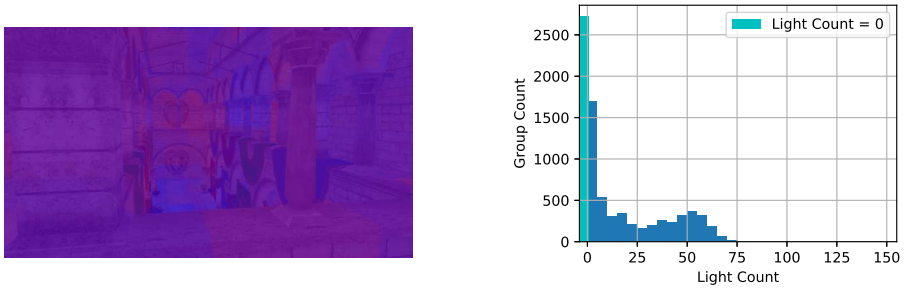


(d) Depth Discontinuity: High, Method: *SC-E*

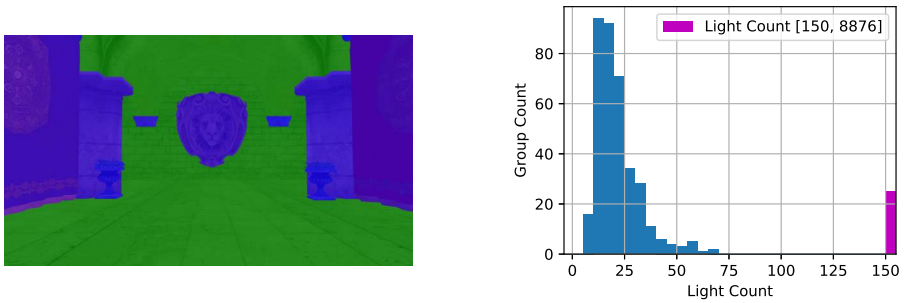
Figure 4.10: *Left:* Heat map of the light count blended with the textures of the models. The light counts are interpolated from 0 to 150 with the colors blue to red and light counts above 150 are shown in green. *Right:* Histogram of the light count of the groups.



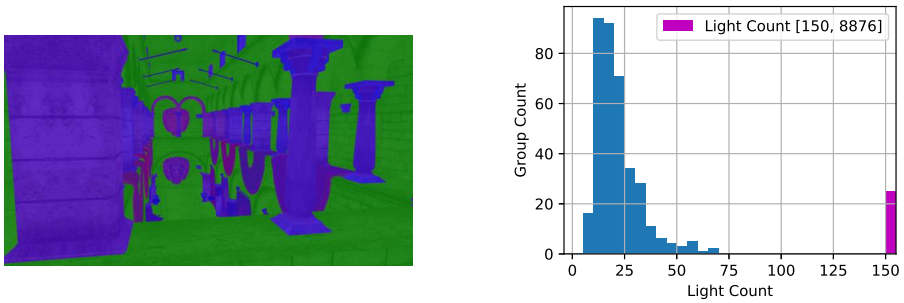
(a) Depth Discontinuity: Low, Method: G



(b) Depth Discontinuity: High, Method: G



(c) Depth Discontinuity: Low, Method: M



(d) Depth Discontinuity: High, Method: M

Figure 4.11: *Left:* Heat map of the light count blended with the textures of the models. The light counts are interpolated from 0 to 150 with the colors blue to red and light counts above 150 are shown in green. *Right:* Histogram of the light count of the groups.

4.4.2 Performance

In general, the average frame time of a light assignment method depends on the number of lights. We are interested in how well a method performs for different light counts. Furthermore, we want to know how intensive the average frame time of a method increases with the light count. In the first part, the average frame time is analyzed for three scenarios with a low, a medium and a high light count. In the second part, a linear polynomial is fitted on the sampled average frame time of the different numbers of lights. The resulting slope and offset of this linear polynomial are further investigated.

4.4.2.1 Low, Medium and High Light Count

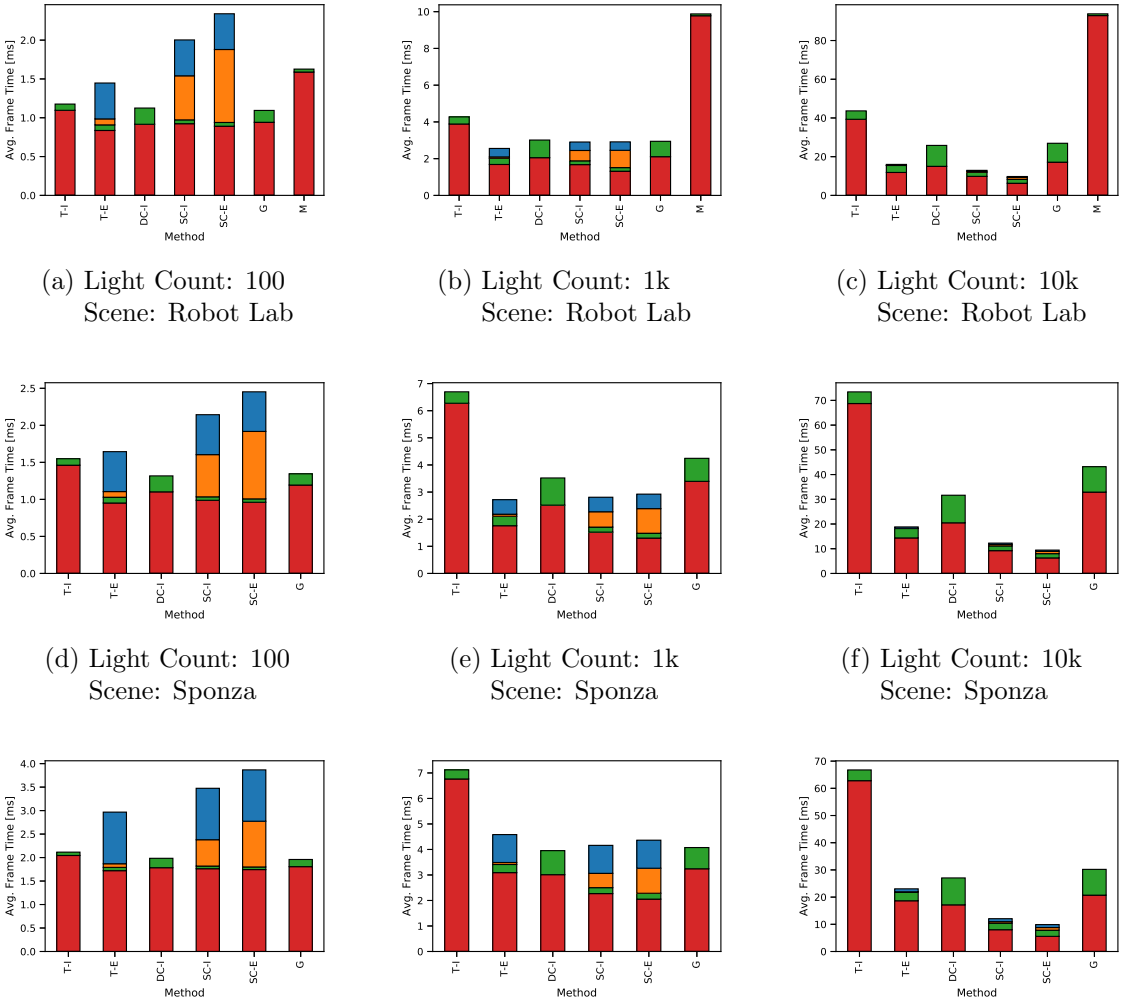
The average frame time of the different methods is computed for a low (100), a medium (1k) and a high (10k) number of lights. Figure 4.12 shows these computations. For each light count, the methods are categorized into four categories by their speed. A single method is categorized by its worst behavior among the three scenes. For example, *M* behaves much worse for Sponza and Viking Village then for Robot Lab. Thus, we consider the performance of Sponza or Viking Village where the method is very slow. The result of this categorization can be seen in Table 4.5. The results are used to decide for each method for which light count it is suitable and also which method is best for a given light count. In the end, we summarize the results.

Suitable Light Count for Method Based on Table 4.5 we can decide for which light counts a method is suitable. We consider a light count suitable for a method is in the category *fast* of that specific light count. *T-I* is only fast for a low number of lights. Hence, it is only usable for that light count. The explicit version *T-E* is not suitable for a low number of lights, due to the overhead of the depth pre-pass and the tile depth range stage. However, it is fit for a medium number of lights. *DC-I* and *G* are both suitable for low and medium light counts. Note that they both have again a similar behavior. *SC-I* and *SC-E* are suitable for a medium and a high number of lights, where the explicit version is faster. They are too slow for a low number of lights because the overhead due to the depth pre-pass and the cluster bound stage is too high. *M* is extremely slow for Sponza and Viking Village except for Robot Lab where it has a medium performance. In brief, it is considered not suitable for any light count.

Best Method for Light Count Table 4.5 can also be used to find the best method for each light count. For a low number of lights, the best method is *DC-I*. But also *G* or *T-I* are suitable. For a high number of lights, the best method is *SC-E* but also *SC-I* is suitable. For a medium number of lights, the best method is *T-E*.

Summary Summarizing, there is no single best method for all light counts. *DC-I* is best for a low number, *T-E* is best for a medium and *SC-E* is best for a high number of

lights. Every method, except M, is suitable for at least one of the tested light counts M performs bad for Robot Lab and catastrophically for Sponza and Viking Villiage. It was shown that the explicit versions are faster for a higher number compared to the implicit version. Also, *DC-I* and G have similar characteristics among different light counts.



(a) Light Count: 100 Scene: Robot Lab

(b) Light Count: 1k Scene: Robot Lab

(c) Light Count: 10k Scene: Robot Lab

(d) Light Count: 100 Scene: Sponza

(e) Light Count: 1k Scene: Sponza

(f) Light Count: 10k Scene: Sponza

(g) Light Count: 100 Scene: Viking Village

(h) Light Count: 1k Scene: Viking Village

(i) Light Count: 10k Scene: Viking Village

Figure 4.12: Average frame time of the light counts of 100, 1k and 10k for different methods and scenes. (The average frame times of M for the light counts 100, 1k and 10k are 23.01 ms, 222.48 ms and 2 259.77 ms for Sponza, and 17.49 ms, 170.46 ms and 1 700.07 ms for Viking Village.)

		Light Count		
		Low (100)	Medium (1k)	High (10k)
Speed	Fast	*DC-I*, *G*, *T-I*	*T-E*, *SC-I*, *SC-E*, *DC-I*, *G*	*SC-E*, *SC-I*
	Medium	*T-E*	*T-I*	*T-E*
	Slow	*SC-I*, *SC-E*		*DC-I*, *G*
	Very Slow	*M*	*M*	*T-I*, *M*

Table 4.5: Light assignment methods grouped by the speed for different light counts. Each cell is sorted by decreasing speed. The grouping is based on Figure 4.12.

4.4.2.2 Frame Time Depending on Light Count

We show that the frame time is a linear function of the light count for every method. First, methods are executed for different light counts in the range of 100 to 10k lights. The average frame time of these executions is fitted on a linear polynomial using the least mean square method. Both, the measured average frame time and the linear fit can be seen in Figure 4.13. The linear function can be fully defined with slope and offset. They are used to compare the methods and to conclude form slope and offset for which light range the methods are best.

Comparing Methods Based on Figure 4.13 we compare the methods *T-I* vs. *T-E*, *SC-I* vs. *SC-E*, *DC-I* vs. *G* and *M* vs. the others. By comparing *T-I* and *T-E* it can be seen that both have similar slopes for light assignment. Note that *T-I* is a little bit better. However, the slope for shading is much lower for *T-I*. Because shading is the dominating factor, *T-E* is much slower than *T-I* for a high number of lights. The comparison of *SC-I* and *SC-E* shows similarities to the previous comparison. The slope of both methods is also similar for the light assignment and the slope for shading is lower for the explicit version. Note that the difference between the shading slope is not as big as between *T-I* and *T-E*. Also, for many lights the average frame time of *SC-I* and *SC-E* is much lower than for *T-I* and *T-E*. The slope and offset of *DC-I* and *G* are quite similar. They have a high slope for light assignment, a medium slope for shading and a medium slope for all stages. *M* has some interesting properties. It has the lowest slope for light assignment and the lowest offset for light assignment and shading. Nevertheless, the performance is very bad due to the very high slope for shading.

Slope Table 4.6 lists the slope and offset of each method grouped in low, medium and high magnitude. In the following, the slope is analyzed. It can be observed that values of the slope of shading are much higher than the slope for light assignment. As a consequence, the slope of all stages is dominated by the shading stage. Comparing the methods, it can be seen that *M* and *T-I* have a high slope. Hence, these methods are not fit for many

lights. *SC-E* and *SC-I* have the lowest slope, and therefore these methods are fit for many lights.

Offset Table 4.6 is used to draw conclusions based on the offset of the frame time. *SC-I* and *SC-E* have a high offset. Therefore, they are not suitable for a low number of lights. *M* has the lowest offset. Nevertheless, its slope is extremely high. and hence it is not even fit for a low number of lights. The other methods, *T-I*, *G* and *DC-I* which have also a low offset do not have such a high slope as *M*. Consequently, those methods are suitable for a low number of lights.

Summary Generally speaking, for all methods, it holds that they have a linear growth and the overall performance is dominated by shading. The extremely high slope of *M* results in a very bad performance. *SC-I* and *SC-E* are best fit for many lights and bad for just a few lights. *T-I*, *G* and *DC-I* are suitable for a low number of lights. *DC-I* and *G* have a very similar slope and offset. The explicit versions have a lower slope than their implicit counterpart. As a consequence, the explicit versions are more suitable for a higher number of lights than the implicit versions.

	Slope		Offset		
	Light Assignment	Shading/ All	Light Assignment	Shading	All
Low	*M*, *SC-E*, *SC-I*	*SC-E*, *SC-I*	*DC-I*, *T-I*, *G*, *M*	*M*, *T-I*, *G*	*M*, *T-I*, *G*, *DC-I*
Medium	*T-E*, *T-I*	*T-E*	*T-E*	*T-E*, *DC-I*	*T-E*
High	*G*, *DC-I*	*DC-I*, *G*, *T-I*[a], *M*[b]	*SC-I*, *SC-E*	*SC-I*, *SC-E*	*SC-I*, *SC-E*

[a] Very High
[b] Extremely High

Table 4.6: Light assignment methods grouped by the magnitude of slope or offset for the stages. Each cell is sorted in ascending order. The grouping is based on Figure 4.13.

4.5 Limitations

There are limitations regarding the light assignment methods, the shading, the setup including the light distribution and also the way we performed the optimization and the analysis. These limitations are listed in the following paragraphs and are ordered form the less important to the most important.

Light Types The only light types which were implemented are point lights. For example, other types like spotlights are not supported. However, our system allows the

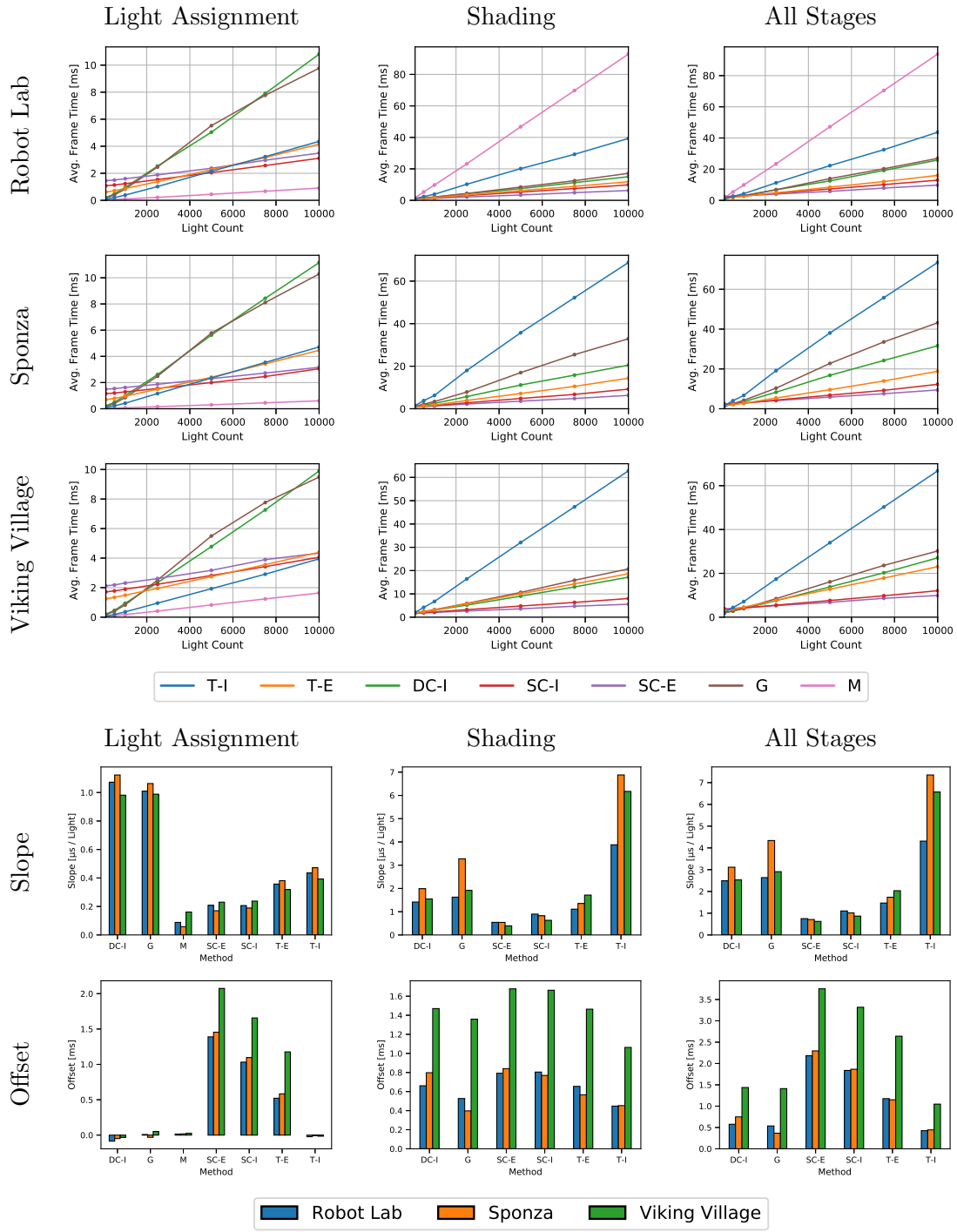


Figure 4.13: *Top:* Average frame time depending on the light count. *Bottom:* Slope and offset of the fitted linear polynomial. (For the *shading stage*, *M* has for scenes Robot Lab, Sponza and Viking Village the slopes 9.23 µs/Light, 226.07 µs/Light and 169.57 µs/Light, and the offsets 0.55 ms, −1.76 ms and 0.35 ms. For *all stages*, *M* has for the scenes Robot Lab, Sponza and Viking Village the slopes 9.32 µs/Light, 226.13 µs/Light and 169.73 µs/Light, and the offsets 0.56 ms, −1.74 ms and 0.37 ms.)

extension with other types. The hardest part will be adding new intersection tests.

Analysis There are more aspects of the light assignment methods which could also be analyzed. For example, it would be interesting to test the light assignment methods for lights higher than 10k. Or to test view dependency not only for one single light count - we used 10k lights - but for different light counts. Furthermore, it would be possible to investigate how the *SD* depends on the light count.

Procedure of Optimization For the optimization, we first selected the group size parameter and then the intersection test. Besides the fact that we already used the best intersection test for the group size parameter optimization this procedure could be seen as incomplete. For complete optimization, it would be necessary to test all combinations of intersection test group size parameters. However, this might result in other problems due to the combinatorial explosion. Furthermore, we believe that the best intersection test will still be *HS+SAT-N* and thus the optimization might still yield the same best configurations.

Group Size Candidates of Optimization We did not investigate the full possibilities of the group size parameter for optimization. For example, we only tested quadratic tile sizes and cubic grid sizes. Also, we did not investigate tile sizes with the same ratio as the render target. It also seems more likely that grid sizes which have a similar ratio as the light bounding box perform better. Depending on the selected scenes the best grid size might be totally different for different scenes. Thus, optimizing the world space grid light assignment method which performs on a global level over multiple scenes might not be a good idea.

Model Subdivision The model-based light assignment methods showed very bad performance for the used scenes. The model subdivision was not adapted to the needs of this method and we think that changing the model subdivision can highly improve the performance. We think that the model-based approach could be at least as fast as the grid-based method, because the scene can be exactly subdivided like the grid method. Furthermore, it would be interesting to compare the performance of the same scene with different model subdivisions.

Different Validity Scopes of Light Assignment Methods The methods which were compared have a highly diverse validity scope. The screen space methods only operate within the view frustum and world space and object space work methods operate on the whole scene. Thus, comparing the performance between these methods could be considered unfair. The world space and object space methods could be adapted to only apply their method within the view frustum. Also, the explicit screen space methods have access to more information, namely the depth buffer, to further reduce the visible lights within a

group. A fair comparison could be done if the world space and object space methods would only be applied to the view frustum and by taking the depth buffer into account to provide explicit bounds.

Light Distribution The optimization and the analysis are based only on one specific distribution of the radius. It is very likely that the optimal group size parameter is different for other distributions. It would be interesting to compare the consequences of different distributions on the optimal group size parameter and performance.

Overdraw Problem There are multiple methods which use a depth pre-pass. They only use the depth pre-pass to compute the explicit bounds of the light assignment groups. However, the depth buffer could also be used to solve the overdraw problem, by providing the depth buffer to the shading stage. We decided not to use the depth buffer because we are only interested in improvement caused by the reduced number of lights. However, this has the disadvantage that the methods are slower and do not use their full potential.

Shading The implementation only supports forward shading, because we focus more on the light assignment methods itself. However, other approaches like deferred shading or even object space shading would be interesting. The methods can be easily extended with deferred shading because only a subset of the view samples of forward shading is used. Nevertheless, not all methods are fit for the usage of object space shading. The screen space methods operate in the view frustum. Hence, they are not suitable for object space shading. The proposed world space and object space methods work on a global level and are fit for object space shading.

5 Conclusion

Contents

5.1 Summary 53
5.2 Future Work 55

5.1 Summary

The fundamental property of a light assignment method is how the view samples are grouped together. Each light assignment method consists of the light assignment step and the shading step. Firstly, in the light assignment step, the visible lights of every group are determined. Then, in the shading step, the lighting is computed by looking up the visible lights of the group of the view sample. In this work, different methods were compared.

Methods Among the methods which were compared are tile-based and cluster-based screen space light assignment. We introduced grid-based and model-based light assignment, which, as far as we know, are novel methods. Grid-based light assignment subdivides the scene into a regular world space grid. Model-based light assignment performs light assignment per model.

Optimization The group size parameter and the intersection test were optimized. In the first step of the optimization, different values for the group size parameters were tested for each method. Then for each method, the value of the group size parameter which performs best among different scenes was selected. It was revealed that tile-based light assignment with explicit bounds is the tile-based method with the lowest average frame time. Furthermore, it was shown that the best cluster-based method in terms of speed and memory is sparse cluster-based light assignment with explicit bounds. In the second step of the optimization, different combinations of intersection tests were evaluated for

the best tile-based and cluster-based method. This demonstrated that the combination of half space method with frustum planes test and Separating Axis Theorem (SAT) with the nearest vertex to sphere center as the candidate axis is the best performing intersection test.

Analysis of View Dependency The analysis indicated that the frame time of every method is view-dependent. It was shown that the explicit bounds and clustering reduces the Standard Deviation (SD) and the Relative Standard Deviation (RSD) of the frame time. Regarding the SD, the sparse cluster-based method has the lowest value. Tile-based light assignment with explicit bounds, dense cluster-based light assignment with implicit bounds and grid-based light assignment have medium SD. Tile-based light assignment with implicit bounds and model-based light assignment have a high SD. The light count distribution of sparse cluster-based light assignment with explicit bounds shows that it is far less dependent on depth complexity than tile-based light assignment with explicit bounds. The model-based and grid-based light assignment work on a global level, and therefore their light assignment output is independent of the view.

Analysis of Performance We investigated the dependency of the average frame on the light count. It was shown that the frame time grows linear with the light count for every method. Using explicit bounds results in a reduction of the slope but an increase of the offset. Thus, methods with explicit bounds perform better for a higher number of lights but worse for a smaller number of lights. Similar characteristics of dense cluster-based with implicit bounds and grid-based light assignment were observed. The reason for that might be that both methods compute the light assignment on a dense grid. There is no single method which performs well for all light count ranges. For a low number of lights, we recommend dense cluster-based with implicit bounds or grid-based light assignment because they both have a low offset. For a medium number of lights, tile-based light assignment with explicit bounds could be used because it has a medium offset and a medium slope. For a high number of lights, sparse cluster-based with explicit bounds should be used because it has the lowest slope. It was shown that model-based light assignment is very slow for all light counts. The reason for that is the big models which span a big portion of the scene, and hence they intersect with many lights.

Takeaway Messages Finding a good group size parameter for multiple scenes can be cumbersome. Speaking generally there is no best group size parameter for all scenes, views or light distributions. We confirmed the work of Thatcher [22] that half space of plane normals in combination with SAT method with the nearest vertex to sphere center is indeed the best intersection test in terms of the lowest average frame time. The best method for many lights is sparse cluster-based light assignment with explicit bounds. It has a low slope, low view dependency of frame time and low dependency of the light count distribution on the depth complexity and can be implemented using less memory due to

the sparse grid. The downside is that it is slow for a few lights due to the high offset. The new methods grid-based light assignment and model-based light assignment have the benefit that the light assignment could be precomputed for static lights and that it could be used for object space shading. However, grid-based light assignment has shown bad performance for many lights. Model-based light assignment has shown extremely high shading time due to the large models intersecting with a lot of lights. We believe, however, that the performance can be significantly improved by adapting the subdivision of the models in the scene.

5.2 Future Work

Many aspects of the presented light assignment methods could be improved. The light assignment can be improved by using a hierarchy of lights, by computing the visible lights by rasterization or by reusing light assignment results across frames. Furthermore, extensions are presented which are specific to screen space, world space or object space methods. In the end, we will discuss using light assignment in combination with object space shading.

Hierarchical Light Assignment A hierarchical data structure can be used to improve either the time spend for light assignment or shading. For example, a Bounding Volume Hierarchy (BVH) could be computed at runtime from the list of lights [12]. The BVH can then be used as the input for the light assignment. A speedup is achieved because it is no longer necessary to iterate over all lights but only to visit a subset of lights by traversing the hierarchy. Moreover, it is possible to store the result of the light assignment step in a hierarchy. For example, for each light assignment group, a tree of visible lights [16] could be computed. Then during shading, for each view sample, the tree is traversed to visit the necessary lights. Thus, the lighting is not computed for all visible lights in the group but only for the traversed ones. A possible problem of this approach is that the overhead added by the hierarchy might in some cases be higher than the benefit. Also, the implementation is harder and must employ an efficient access pattern.

Rasterized Light Assignment Instead of purely relying on analytical testing, the light assignment could be implemented by using rasterization. For example, Örtegren [19] uses rasterization in combination with clustered shading. The light volumes are rasterized and the clusters are filled with the intersecting lights. [4] presented a hybrid solution of rasterization and analytical testing. We believe it is also possible to use rasterization for a world space grid method by rendering the lights using an orthogonal projection.

Reuse Light Assignment Across Frames The current implementation computes the complete light assignment for each frame independently. This makes sense if lights totally change from frame to frame. However, in practical use cases, it is more likely that the

lights only slightly change the position or the radius from one frame to the next frame. Therefore, it makes sense to take advantage of this fact. For example, we could compute the light assignment only for the lights which moved or which got visible in the view frustum and additionally reuse the light assignment from the lights which are not moved. Another approach of reusing light assignment would be to compute the light assignment in a conservative way and use it for more than one frame. This effectively decouples shading and light assignment and enables asynchronous light assignment. For example, the asynchronous light assignment could be performed on the Central Processing Unit (CPU), while the shading is executed at the same time on the Graphics Processing Unit (GPU).

Extending Screen Space Light Assignment There are a lot of variants of screen space light assignment methods, which we did not implement. Variants of tile shading which we could implement in the future are HalfZ, modified HalfZ [23] or 2.5D Culling [11]. However, more promising in terms of performance are the variants of clustered shading. For example, practical clustered shading [20] and cascaded clustering [8] address the problematic subdivision scheme of clustered shading, which causes redundant light assignment due to the small sizes of some clusters.

Extending World Space and Object Space Light Assignment There are extensions to the world space and object space light assignment which can improve performance. For example, such extensions are view frustum culling or explicit bounds based on a depth pre-pass. It would also be interesting to create a hybrid method, where the models are subdivided into a grid and light assignment is performed for each cell of each model. The grid-based light assignment could also be changed in such a way that the grid is not applied on the whole scene but only on a world space $AABB$ containing the view frustum or the view samples. Furthermore, it would be possible to subdivide the scene into a world space octree like hashed shading [21].

Light Assignment for Object Space Shading The light assignment methods presented in this thesis all use forward shading. However, it would be interesting to use an object space shading approach. First, we discuss which of the presented light assignment methods are suitable for object space shading. The methods can be categorized into the scope which is taken into account for light assignment. The model-based and the grid-based method take the whole scene into account. Hence, they are fit for object space shading, which might access any sample within a scene. The implicit screen space methods take only the view frustum into account. By extending the view frustum by a small factor and taking a bigger space into account it might be usable for some variants of object space shading. However, in general, this is insufficient, and, thus, these methods are not suitable. The explicit screen space methods take only view samples into account. This is insufficient for object space shading because hidden geometry might be shaded.

Summarizing, only the grid-based and the model-based light assignment method produce correct results for object space shading. However, they are quite slow. Another idea is to develop a light assignment method specifically designed for object space shading. One possibility would be to take the world space positions of the shading samples into account. First, the world Axis-Aligned Bounding Box (AABB) of all shading samples is computed. Then the $AABB$ is subdivided into a grid and light assignment is performed for each cell. This could be further improved by computing the explicit bounds of the cells.

A

List of Acronyms

AABB	Axis-Aligned Bounding Box
API	Application Programming Interface
BVH	Bounding Volume Hierarchy
CPU	Central Processing Unit
DC-E	Dense Cluster-Based Light Assignment with Explicit Bounds
DC-I	Dense Cluster-Based Light Assignment with Implicit Bounds
G	Grid-Based Light Assignment
GPU	Graphics Processing Unit
HS	Half Space Intersection Test
M	Model-Based Light Assignment
MSAA	Multisample Anti-Aliasing
NDC	Normalized Device Coordinates
NP	Nearest Point Method Intersection Test
NPT	Nearest Point Method with Transformation Intersection Test
OBB	Oriented Bounding Box
PDF	Probability Density Function
RSD	Relative Standard Deviation
SAT	Separating Axis Theorem
SAT-E	Separating Axis Theorem with Edges Intersection Test
SAT-N	Separating Axis Theorem with Nearest Vertex to Sphere Center Intersection Test
SAT-P	Separating Axis Theorem with Plane Normals Intersection Test

SAT-V	Separating Axis Theorem with Vertices to Sphere Center Intersection Test
SC-E	Sparse Cluster-Based Light Assignment with Explicit Bounds
SC-I	Sparse Cluster-Based Light Assignment with Implicit Bounds
SD	Standard Deviation
T-E	Tile-Based Light Assignment with Explicit Bounds
T-I	Tile-Based Light Assignment with Implicit Bounds

Bibliography

[1] Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Hillaire, S., and Iwanicki, M. (2018). *Real-Time Rendering, Fourth Edition*. A. K. Peters/CRC Press. (page 12)

[2] Anagnostou, K. (2017). How Unreal Renders a Frame part 2. `https://interplayoflight.wordpress.com/2017/10/25/how-unreal-renders-a-frame-part-2/`. Online; accessed 23 January 2019. (page 7)

[3] Andersson, J. (2009). Parallel Graphics in Frostbite - Current & Future. SIGGRAPH '09: ACM SIGGRAPH 2009 Courses, New York, NY, USA. ACM. `http://s09.idav.ucdavis.edu/talks/04-JAndersson-ParallelFrostbite-Siggraph09.pdf`. (page 5)

[4] Archer, J., Leach, G., Knowles, P., and Schyndel, R. (2018). Hybrid Lighting for Faster Rendering of Scenes with Many Lights. *Vis. Comput.*, 34(6-8):853–862. (page 11, 12, 55)

[5] Arva, J. and Aila, T. (2003). Optimized Shadow Mapping Using the Stencil Buffer. *Journal of Graphics Tools*, 8(3):23–32. (page 1)

[6] Arvo, J. (1990). Graphics Gems. chapter A Simple Method for Box-Sphere Intersection Testing, pages 335–339. Academic Press Professional, Inc., San Diego, CA, USA. `http://dl.acm.org/citation.cfm?id=90767.90844`. (page 12, 24)

[7] Batagelo, H. and Wu, S.-T. (2002). Dynamic scene occlusion culling using a regular grid. In *Proceedings. XV Brazilian Symposium on Computer Graphics and Image Processing*, pages 43–50. (page 9, 22)

[8] Billeter, M. (2015). Real-Time Many-Light Management and Shadows with Clustered Shading: Many-Light Rendering on Mobile Hardware . SIGGRAPH '15: ACM SIGGRAPH 2015 Courses, New York, NY, USA. ACM. `https://newq.net/dl/pub/s2015_mobile.pdf`. (page 8, 56)

[9] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*, chapter 2.5.1, page 46. Cambridge University Press. (page 25)

[10] Garanzha, K. and Loop, C. (2010). Fast Ray Sorting and Breadth-First Packet Traversal for GPU Ray Tracing. In *Computer Graphics Forum*, volume 29, pages 289–298. (page 9, 22)

[11] Harada, T. (2012). A 2.5D Culling for Forward+. In *SIGGRAPH Asia 2012 Technical Briefs*, SA '12, pages 18:1–18:4, New York, NY, USA. ACM. (page 7, 11, 56)

[12] Harada, T., McKee, J., and Yang, J. C. (2012). Forward+: Bringing Deferred Lighting to the Next Level. In Andujar, C. and Puppo, E., editors, *Eurographics 2012 - Short Papers*. The Eurographics Association. (page 10, 11, 55)

[13] Krishnamoorthy, K. (2006). *Handbook of Statistical Distributions with Applications*, chapter 15, pages 185–194. Statistics: A Series of Textbooks and Monographs. Chapman and Hall/CRC. (page 29)

[14] Lauritzen, A. (2010). Deferred Rendering for Current and Future Rendering Pipelines. SIGGRAPH '10: ACM SIGGRAPH 2010 Courses, New York, NY, USA. ACM. `http://bps10.idav.ucdavis.edu/talks/12-lauritzen_DeferredShading_BPS_SIGGRAPH2010_Notes.pdf`. (page 5)

[15] Lauritzen, A. (2012). Intersecting Lights with Pixels: Reasoning about Forward and Deferred Rendering. SIGGRAPH '12: ACM SIGGRAPH 2012 Courses, New York, NY, USA. ACM. `http://bps12.idav.ucdavis.edu/talks/03_lauritzenIntersectingLights_bps2012.pdf`. (page 6)

[16] O'Donnell, Y. and Chajdas, M. G. (2017). Tiled Light Trees. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '17, pages 1:1–1:7, New York, NY, USA. ACM. (page 11, 55)

[17] Olsson, O. and Assarsson, U. (2011). Tiled Shading. *Journal of Graphics*, GPU:235–251. (page 2, 5, 6, 10, 11)

[18] Olsson, O., Billeter, M., and Assarsson, U. (2012). Clustered Deferred and Forward Shading. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, pages 87–96, Goslar Germany, Germany. Eurographics Association. (page 2, 7, 10, 11, 18)

[19] Örtegren, K. (2015). Clustered Shading - Assigning arbitrarily shaped convex light volumes using conservative rasterization. Master's thesis, Blekinge Institute of Technology. (page 10, 12, 55)

[20] Persson, E. (2015). Real-Time Many-Light Management and Shadows with Clustered Shading: Practical Clustered Shading. SIGGRAPH '15: ACM SIGGRAPH 2015 Courses, New York, NY, USA. ACM. `https://newq.net/dl/pub/s2015_practical.pdf`. (page 7, 8, 56)

[21] Tegelaers, M. W. (2018). Forward and Deferred Hashed Shading for Real-time Rendering of Many Lights. unpublished paper. (page 2, 9, 56)

[22] Thatcher, L. (2014). Optimisations of the light culling algorithm in a Forward+ Rendering Pipeline. (page 13, 25, 31, 37, 54)

[23] Thomas, G. (2015). Advancements in Tiled-Based Compute Rendering. `http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Thomas_Gareth_Advancements_in_Tile-Based.pdf`. (page 6, 7, 56)