SEMI DYNAMIC LIGHT MAPS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BEKİR ÖZTÜRK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MULTIMEDIA INFORMATICS

SEPTEMBER 2019

Approval of the thesis:

SEMI DYNAMIC LIGHT MAPS

submitted by **BEKİR ÖZTÜRK** in partial fulfillment of the requirements for the degree of **Master of Science in Multimedia Informatics Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, Graduate School of **Informatics**

Assist. Prof. Dr. Elif Sürer
Head of Department, **Multimedia Informatics**

Assoc. Prof. Dr. Ahmet Oğuz Akyüz
Supervisor, **Computer Engineering, METU**

Examining Committee Members:

Assoc. Prof. Dr. Hüseyin Hacıhabiboğlu
Multimedia Informatics, METU

Assoc. Prof. Dr. Ahmet Oğuz Akyüz
Computer Engineering, METU

Prof. Dr. Tolga Can
Computer Engineering, METU

Prof. Dr. Tolga Kurtuluş Çapın
Computer Engineering, TED University

Assoc. Prof. Dr. Yusuf Sahillioğlu
Computer Engineering, METU

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Bekir Öztürk

Signature :

ABSTRACT

SEMI DYNAMIC LIGHT MAPS

Öztürk, Bekir
M.S., Department of Multimedia Informatics
Supervisor: Assoc. Prof. Dr. Ahmet Oğuz Akyüz

September 2019, 58 pages

One of the biggest challenges of real-time graphics applications is to maintain high frame rates while producing realistically lit results. Many realistic lighting effects such as indirect illumination, ambient occlusion, soft shadows, and caustics are either too complex to render in real-time with today's hardware or cause significant hits to frame rates. Light mapping technique offers to precompute the lighting of the scene to speed up expensive lighting calculations at run-time. This allows rendering high quality lights from a high number of light sources even on low-end devices. The primary drawback of this technique is that scene state that is dependent on the precomputed data cannot be changed at run-time. This includes intensity, color, and position of light sources as well as position and visibility state of light map illuminated objects. This property of light maps significantly decreases the interactability of applications. In this thesis, we present a method to remove some of these restrictions at the cost of additional texture memory and small CPU/GPU workload. This allows changing color and intensity properties of selected light sources at run-time while keeping the benefits of light mapping technique. It is also becomes possible to change visibility state of selected objects. Our algorithm computes the light maps separately for each light source. Regions shadowed by each selected object are also captured and stored. These maps are later combined at run-time to correctly illuminate the scene. Despite the increase in the generation time of precomputed data, overhead of the method at run-time is low enough to make it useful in many real-time applications.

Keywords: Light mapping, global illumination, indirect illumination

ÖZ

YARI DİNAMİK IŞIK HARİTALARI

Öztürk, Bekir
Yüksek Lisans, Çokluortam Bilişimi Bölümü
Tez Yöneticisi: Doç. Dr. Ahmet Oğuz Akyüz

Eylül 2019 , 58 sayfa

Gerçek zamanlı grafik uygulamaların başlıca zorluklarından biri gerçekçi ışıklandırmalara sahip sonuçlar elde ederken yüksek kare sayısı değerini koruyabilmektir. Dolaylı aydınlatma, ambient occlusion, yumuşak gölgeler, geçirgen yüzeylerde ışık kırılmaları gibi etkilerin günümüz donanımlarında gerçek zamanlı resmedilmesi çoğunlukla mümkün değildir. Mümkün olduğu durumlarda ise kare sayısında ciddi düşüşlere sebep olmaktadır. Işık haritalama yöntemi yüksek donanım gerektiren gerçekçi ışık hesaplamalarını ön aşamada gerçekleştirerek sahnenin gerçek zamanlı resmedilmesini hızlandırmaktadır. Bu yöntem, çok sayıda ışık kaynağı tarafından yayılan ışığın gerçekçi davranışlarının alt seviye donanımlarda dahi gerçek zamanlı işlenebilmesine olanak sağlamaktadır. Işık haritalama yönteminin en büyük eksiği, sahnenin önceden gerçekleştirilen ışık hesaplamaları sırasınnda kaydedilmiş olan vaziyetinin uygulamanın çalışması esnasında değiştirilmemesini şart koşmasıdır. Bu durum ışık kaynaklarının konum, renk ve ışık şiddetlerinin; nesnelerin ise konum ve görünürlük durumlarının değiştirilememesi anlamına gelmektedir. Işık haritalarının bu özelliği, bu tekniğin etkileşimli bir çok uygulamada kullanılmasının önüne geçmektedir. Bu tezde, doku belleğinde, işlemci ve ekran kartı iş yükünde bir miktar artışa karşılık, bahsedilen kısıtların bir kısmını ortadan kaldıracak bir yöntem önerilmiştir. Kaldırılan kısıtlamalar, seçilen ışık kaynaklarında ışık renginin ve şiddetinin değiştirilebilmesinin yanı sıra seçilen nesnelerde nesne görünürlüğünün değiştirilebilmesini mümkün kılmaktadır. Önerilen yöntemde ışık haritaları her bir ışık kaynağı için ayrı olarak hesaplanmaktadır. Nesnelerin oluşturduğu gölgeler de ön hesaplamalar sırasında tespit edilerek haritalara eklenir. Bu haritalar uygulamanın çalışması esnasında birleştirilerek sahnenin o anki vaziyetine uygun ışık haritaları üretilir. Ön hesaplama sürelerindeki artışa rağmen, yöntemin çalışma esnasında iş yükü bir çok uygulamada kullanılmasını imkan sağlayacak şekilde azdır.

Anahtar Kelimeler: Işık haritalama, günışığı aydınlığı, dolaylı aydınlatma

ACKNOWLEDGMENTS

For guiding me into the academic world of computer graphics, for his endless support during my study, and for the patience he has shown me to the last minute of writing this thesis; I sincerely thank my advisor Assoc. Prof. Dr Ahmet Oğuz Akyüz. It was his invaluable guidance that got me to publish and present my work at an amazing event such as SIGGRAPH. There are no words that can describe how grateful I feel for everything he has done.

For the tolerance they have shown to my courses, homeworks and exams during my work at their companies, and for the guidance they have provided both in an out of work; I would like to thank my teachers and bosses Refik Toksöz (Reo-Tek), Emrecan Çubukçu and Kıvanç Çubukçu (Kreatin Studios).

From the very beginning of my life to this date, for considering all my problems as their own, for putting even the smallest of my needs before their's and for always being there for me; I would like to thank my parents. I know that there is no way I can repay for the things they have done for me.

For her endless support at every step of my life, for her motivational speeches, praises, amazing desserts, for setting me into the right path whenever I go astray; I would like to thank my joy, my muse, my dear wife Hatice Kübra.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ vi

ACKNOWLEDGMENTS ix

TABLE OF CONTENTS x

LIST OF FIGURES xiii

LIST OF ABBREVIATIONS xvii

CHAPTERS

1 INTRODUCTION 1

2 RELATED WORK 3

- 2.1 Precomputed Data in Computer Graphics 3
  - 2.1.1 Photon Mapping 3
  - 2.1.2 Environment Mapping 4
  - 2.1.3 Irradiance Maps 5
- 2.2 Textures in Graphics 5
  - 2.2.1 Bump Mapping 6
- 2.3 Light Mapping 6
  - 2.3.1 Baking 7
  - 2.3.2 Reacting to Changes 8

2.3.3 Directional Light Maps 9

3 LIGHT MAPPING 11

4 ALGORITHM 15

4.1 Offline Stage 15

4.1.1 Marking of Semi Dynamic Objects and Light Sources 15

4.1.2 Calculating Contribution of Static Light Sources 16

4.1.3 Calculating Contribution of Each Light Source 16

4.1.4 Calculating Contribution of Each Object 17

4.1.5 Reducing Texture Size 20

4.2 Run-Time Stage 24

4.2.1 Reconstructing Base Maps 24

4.2.2 Reconstructing Contribution Maps 24

4.2.3 Shadowing Contribution Maps 25

4.2.4 Generating The Final Light Map 28

4.2.5 Updating The Light Map 29

5 RESULTS 33

5.1 Baking Times 34

5.1.1 Number of Semi Dynamic Light Sources 34

5.1.2 Number of Semi Dynamic Objects 35

5.2 Storage Space 37

5.3 Run-Time Memory 42

5.4 Run-Time Processing Costs 42

6 CONCLUSIONS 49

6.1 Limitations 50

  6.1.1 Color Bleeding 50

  6.1.2 Indirect Illumination with Semi Dynamic Objects 51

  6.1.3 Static Light Sources with Semi Dynamic Objects 51

6.2 Future Work 51

REFERENCES 53

LIST OF FIGURES

FIGURES

Figure 1.1 Using high quality rendering techniques, light maps (left) can generate more realistic results than real-time lighting (right). 2

Figure 2.1 With progressive photon mapping, number of photons can be progressively increased without allocating additional memory. Figure is taken from [1] 4

Figure 2.2 On the left, a cylinder bumped with horizontal stripes is shown. Right image is the same cylinder rendered as a silhouette. Notice that bumps don't contribute to the object's silhouette since geometry is not changed. Bump effect is only illusionary. 6

Figure 2.3 Each mipmap level stores the indirect lighting of a horizontal slice of the zone. Each pixel stores the indirect lighting of a vertical slice within the horizontal slice. Image taken from [2]. 7

Figure 3.1 Sample scene consists of 4 point light sources of different colors and 1 slightly yellow directional light source. The only objects are a small cube and a plane. 11

Figure 3.2 Light map uniquely maps the surface of each object. 12

Figure 3.3 Sharp changes in the lighting cannot be correctly captured in low resolution light maps as shown in the left image. This issue is much less noticeable on the right image which uses 16 times more pixels for the same light map. 12

Figure 3.4 Final light map of *Dungeon* scene. 13

Figure 3.5 The light that is visible in the rendered image is not computed at run-time. It is simply retrieved from the light map from the correct regions. 14

Figure 4.1 Image on the left shows the rendered sample environment. Right image is the light map. 15

Figure 4.2 Image on the left shows the state of the scene during the calculation of the contribution map of the light source. Image on the right is the contribution map of the light source. 17

Figure 4.3 Shadow maps store how much light is shadowed by an object from a light source. On the left, shadowed contribution maps that are used to generate shadow maps is shown for each object. Resulting shadow maps are shown on the right. 19

Figure 4.4 Shadow maps are mostly black and contain little data. Unused texture space can be saved. 20

Figure 4.5 Light patches are placed into atlases, where each channel contains some of the patches. 22

Figure 4.6 Light patches from corresponding atlases are placed onto a texture to create a contribution map. This process is repeated for each semi dynamic light source. 26

Figure 4.7 Depending on which semi dynamic objects are enabled, previously generated contribution map is shadowed. 27

Figure 5.1 Images of samples scenes *Library* (left) and *Dungeon* (right) . . 34

Figure 5.2 For a scene with n semi dynamic light sources, $n + 1$ baking operations are required. This looks like the baking times will be $n + 1$ times the baking time of conventional light maps. Yellow line shows this expected rate. Blue and red lines are the actual rate of bake time compared to conventional light maps. Notice how slowly bake times increase compared to the expected. 35

Figure 5.3 In sample *Library* and *Cemetery* scenes which contain 8 light sources, light maps were generated for 8 times where each time an additional light source is marked as semi dynamic. Each additional semi dynamic light source causes an increase in the baking time. 36

Figure 5.4 Average time spent on each bake operation decreases with each additional bake. 37

Figure 5.5 Maps generated by baking process should be processed, which increases the generation time of semi dynamic light maps. Graphs show light map generation times for *Library* (top) and *Cemetery* (bottom) scenes. 38

Figure 5.6 Bake times increase slowly compared to the increase in the number of semi dynamic objects. However, semi dynamic light maps can still take significantly longer to generate than conventional light maps. Note how generation of light maps for the *Library* scene where there are 8 semi dynamic light sources and 10 semi dynamic objects takes almost 24 times longer. 39

Figure 5.7 Semi dynamic light map generation times in sample scenes *Library* and *Cemetery* which contain 8 and 1 semi dynamic light sources respectively. It is important to note that baking times for *Library* scene increase much faster than *Cemetery* scene. This is expected since *Library* scene contains 7 times more light sources than *Cemetery* scene. . 40

Figure 5.8 Average time spent on a single baking operation significantly decreases as the number of consecutive bake operations increases. . . . 41

Figure 5.9 Semi dynamic light maps efficiently store light values for texels which only receive light from at most 2 light sources. This allows reduced storage size of light maps. Semi dynamic objects, however, always cause an increase in the storage. 44

Figure 5.10 Semi dynamic objects usually increase required memory since each semi dynamic object uses additional texture space in the atlas. This does not always apply to semi dynamic light sources which can be more efficiently stored as explained in 5.9. 45

Figure 5.11 In *Cemetery* scene, semi dynamic objects only cast shadows to the ground. A single light patch is usually sufficient to store the shadows. Therefore, auxiliary data increases very slowly when semi dynamic object count increases in the graph below. 46

Figure 5.12 Run-time processing costs start as low as 0.25 milliseconds, but can be significant in complex scenes. 47

Figure 5.13 The cost of updating the light maps at run-time can increase to significant values. However, it is possible to keep a relatively smooth frame rate by executing the update operation in multiple frames or in separate threads. 47

Figure 6.1 Red color "bleeds" from the cube to the floor. 50

LIST OF ABBREVIATIONS

EM	Environment Map
DRM	Diffuse Reflection Map
B	Base map of the scene
C_l	Contribution map of light source l
S_{o_l}	Shadow map of object o created by light source l
3D	Three dimensional

CHAPTER 1

INTRODUCTION

Lighting is a very important aspect of rendering that significantly effects the realism of generated image. Many high quality rendering techniques use a ray tracing based approach to simulate the behaviour of light [3] [4]. These methods yield very good looking results that are sometimes indistinguishable from photographs. Lights reflecting off of surfaces, refractions, subsurface scattering are some of the effects that can be simulated with these methods. However, it is often not possible to use these methods in real-time applications since rendering times are too high to achieve high frame rates in today's hardware. A common approach to this problem is to use precomputed data [3] [5] [6] [7]. This data contains information about the lighting of the scene. At run-time, an efficient method is used to compute correct light values and generate realistically lit images using this data. One of the mainly used lighting techniques in real-time applications is light mapping [8] [9]. In this technique, all the surfaces in the scene are uniquely mapped to a texture called light map. During the precomputations, lighting of the scene is simulated using a high quality realistic method. Received light per unit area is calculated and stored in texels for all the surfaces in the scene. This operation is known as "baking". At run-time, lighting of a point can easily be calculated by looking up the value from the light map. Light maps don't store information about the source of the light. Multiple light sources can contribute to the value of a single texel. A great advantage of this is that light map size and run time performance is independent of the number of lights in the scene. Area lights, indirect illumination, ambient occlusion [10] are some of the effects that can be efficiently used with light mapping as can be seen in Figure 1.1.

There are also drawbacks of using light mapping. In common implementations of light mapping, texels contain only intensity value. Direction of the light is not stored. For this reason, light maps are said to be 'view independent'. This means that view-dependent techniques cannot be simulated using light maps, such as specular highlights and bump mapping.

Another and possibly the biggest drawback of using light maps is the restrictions imposed on the scene's state. After baking of the light map is complete, light received at any point in the scene is already decided. Therefore, no actions can be taken to change the lighting of the scene without re-baking the light map. For instance; intensity, color or position of the light sources cannot be changed. Objects cannot be moved or removed from the scene. Newly added objects will not be illuminated. Removed objects will continue to cast shadows. These problems can be avoided by excluding these objects and light sources from light maps. In this case, realism of the rendered image will significantly drop since lighting calculations will have to be done

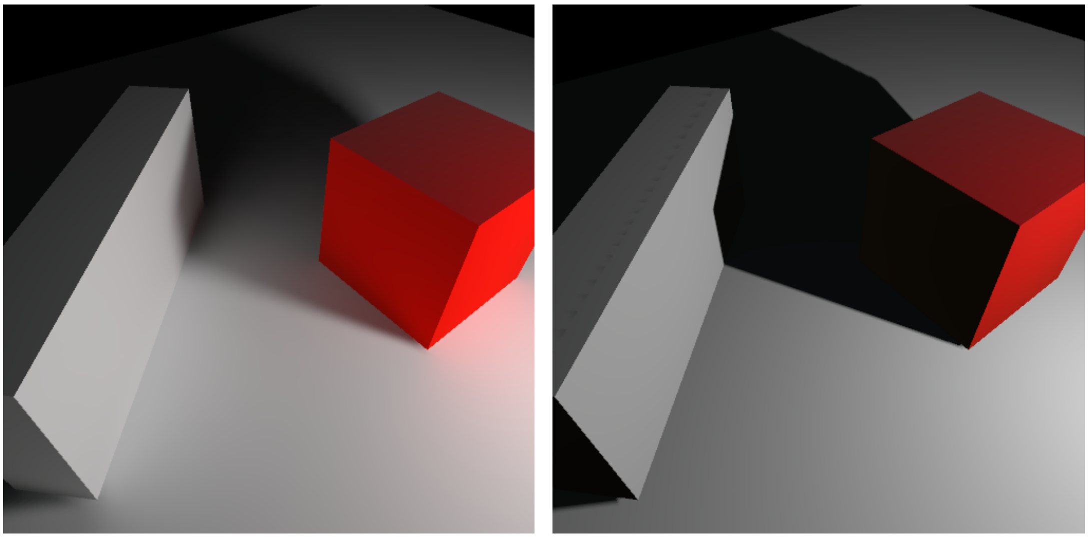


Figure 1.1: Using high quality rendering techniques, light maps (left) can generate more realistic results than real-time lighting (right).

using real-time low quality methods. These restrictions strictly limit the interactivity of applications.

In this thesis, a method is proposed to allow changes to selected objects and light sources without compromising the realism offered by light mapping. Specifically, proposed method allows the following modifications: changing the color and intensity of light sources, removing or adding objects to the scene. Unlike common light mapping methods, light map is generated separately for each light source. This makes it possible to change the intensity and color of each light source independently. Executing an additional step at run-time is necessary to combine light source specific maps into a final light map. Similarly, shadows of objects are stored independently in additional maps. Depending on the visibility state of the object, shadows are applied into the final light map.

CHAPTER 2

RELATED WORK

2.1 Precomputed Data in Computer Graphics

Precomputing lighting to reduce rendering times is a very common procedure used in many high quality rendering techniques [11] [12] [4] [13] [14]. Main idea consists of processing lights and geometry in the scene and storing the results in vertices, textures, files or memory to be used during rendering. In this section some of the techniques that benefit from precomputed lighting is reviewed.

2.1.1 Photon Mapping

Photon mapping is one of the techniques that takes advantage of a precomputed light data [3]. During the first pass of this two-pass method, a large number of photons are sent from each light source into the scene. Traveling photons are stored on the surfaces they hit. During the second pass, rays are sent from eye into the scene and value of each pixel is calculated using the photons stored around the point within a certain radius.

To generate realistic results, a large number of photons is required [15]. This significantly increases memory requirements since photons cannot be removed from memory until rendering completes. Instead of storing photons, Good and Taylor [7] used a Spherical Harmonic Light Map to store accumulated photons as spherical harmonic coefficients [6]. With this method, required memory is independent of the number of emitted photons.

Hachisuka et al. [1] proposed another method to accumulate photons incrementally without keeping them all in memory. After rendering the scene with raytracing, any number of photon tracing passes with limited number of photons can be issued as can be seen in Figure 2.1. Each pass requires the statistical data generated at the end of the previous step. Because of this requirement, all passes need to be executed sequentially. Each completed pass provides a better quality lighting as photon count increases and photon accumulation radius for each intersection point decreases.

Knaus and Zwicker later removed the data carried over from one photon tracing pass to the other to allow executing passes in parallel [16].

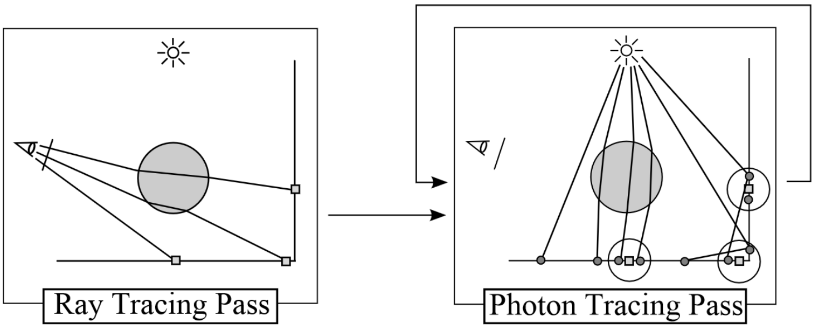


Figure 2.1: With progressive photon mapping, number of photons can be progressively increased without allocating additional memory. Figure is taken from [1]

2.1.2 Environment Mapping

Rendering reflections often requires expensive raytracing operations which is not possible in interactive applications [17] [18]. Environment maps (EM) store the reflections around a point to approximate the results in much shorter times for mirror-like objects [19] [20] [21]. Generation of an EM starts with selecting a point within or near reflective objects. From this point, scene view is captured and saved into textures. Capturing can happen in multiple ways. Earlier implementations used raytracing and only captured a single direction or two opposite directions, storing them in one and two textures respectively [5] [22]. Today, cube maps are frequently used since they accurately capture the view for most general purpose applications [23]. When rendering a point, reflected color is sampled from EMs instead of sending a reflected ray into the scene.

In cases where reflective object is greatly distant from the surrounding environment, changes to the position of the object can be ignored during reflection calculations. However, in small environments with large reflective objects, an environment map can only be used around the point it was captured [24] [23]. When reflective object is moved, a different EM captured at the new point should be used. Capturing a new EM for every possible position of the object is impractical, if not impossible. One of the solutions to this problem is to generate a limited number of EMs along the path of the moving object [25] [26]. Two closest EMs can then be blended. Distance of the object to EMs can be used as blending weight. However, this approach yields unrealistic results as objects appear and disappear from reflections when weight shifts from one EM to the next. Instead of simple blending, Meyer and Loscos [27] offered a solution to reconstruct a new EM at the object's position. This method uses a height map to calculate intersection of a reflected ray with the environment instead of using raytracing. Therefore, it works best with 2.5 dimensional, urban-like environments. However, a voxel based approach can be used in full 3D environments. Unlike common EMs, EMs proposed in this method also store the distance of each intersection point to the capture point of the EM. During rendering, reflected ray is generated and its intersection with the environment is calculated using the height map. Then, light coming from the direction of the point is sampled from two EMs. If both EMs have a direct line of sight to the point, final reflected color is computed by blending the two sampled values. Depending on the scene setup, however, EMs might not always see this point. This can happen when there is an object blocking the view of an EM. This situation can be detected using the distance value stored in EMs. If stored distance in the direction of intersection point is smaller than the distance to the intersection point, it can be concluded that intersection point was occluded by a closer object. Sampled value cannot be used to calculate reflections in this case.

Environment maps can be also used to render reflections on diffuse surfaces. For a point on a diffuse surface with normal N, this requires accumulating values sampled from EM for each reflected direction R where $R \cdot N > 0$ holds. Although this is a very heavy operation to execute in real-time, it can be precomputed into a texture called "Diffuse Reflection Map" [22]. Resulting texture looks like a blurred version of the EM and and can be scaled down to lower resolutions without compromising quality.

2.1.3 Irradiance Maps

Miller's Diffuse Reflection Maps is an effective way to calculate reflections on diffuse surfaces. It offers good quality and only requires a single sample to fetch reflection at any direction. However, generating DRMs takes a considerable amount of time as integrating over a hemisphere of EM is required for each pixel of DRM. Ramamoorthi and Hanrahan showed that using only 9 parameters, any diffuse reflection map can be closely estimated [6]. The main idea is to use spherical harmonic coefficients. Due to the blurry nature of DRMs, values of high frequency coefficients are small enough to be neglected. Therefore, using only 3 lowest frequencies (constant, linear and quadratic) is sufficient. When compared to the generation of DRMs, calculating spherical harmonics coefficients takes significantly less time since integration over EM is only needed to be done 9 times as opposed to once for each pixel.

On static objects where coefficients are stored on the surface, all the possible reflected directions on a surface point form a hemisphere instead of a sphere. This is because a point can only be viewed from one side and no reflected directions exists on the back side of the surface. Taking advantage of this fact, Habel and Wimmer showed that it is sufficient to use only 6 spherical harmonics coefficients for irradiance normal mapping [28] of static objects. Instead of a full sphere, these 6 coefficients represent a hemisphere. Unused hemisphere is completely omitted.

2.2 Textures in Graphics

Catmull was the first to map textures onto 3D objects [29]. It was later used in many different work to increase the detail of rendered surfaces [24] [30] [31]. In this section, we will briefly review a common usage of texture mapping in real-time rendering, namely bump mapping.

2.2.1 Bump Mapping

Early renderers, both offline and real-time, generated images with perfectly smooth surfaces [21]. This significantly decreases the realism of the results as surfaces are rarely that smooth in real life. Using higher number of polygons to add bumps and wrinkles is one of the solutions, but it quickly increases render times. Another method is to use diffuse textures [29]. However, it is not possible to correctly illuminate details on diffuse textures as they only contain color information.

To fix this problem, Blinn mapped 2D surface normal data onto 3D objects [32]. Normal direction of each point on the surface was updated using the mapped texture and lighting was calculated using new normal vector. This allowed for much realistic bump details while keeping the polygon count unchanged. This method was later used in other studies [33] [34] and various improvements was proposed to add hardware support for bump maps [35] [36] [37] [38].

Bump maps only affect the surface normal per pixel and does not modify the geometry. Therefore, bumps of a surface does not change the object's silhouette as can be seen in Figure 2.2.

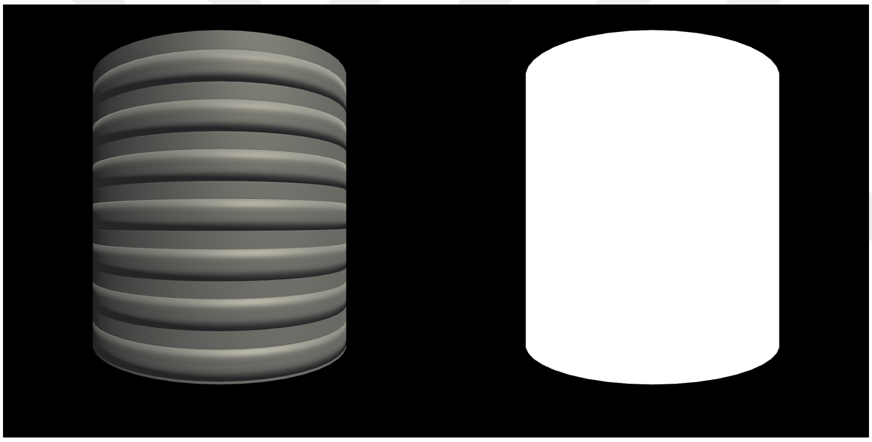


Figure 2.2: On the left, a cylinder bumped with horizontal stripes is shown. Right image is the same cylinder rendered as a silhouette. Notice that bumps don't contribute to the object's silhouette since geometry is not changed. Bump effect is only illusionary.

2.3 Light Mapping

In previous section, we reviewed some methods that store lighting in various formats and structures. Even though the methods offer great contributions to efficient rendering methods, each method has some disadvantages that makes it unsuitable for generic cases. Light mapping is a technique that uses textures to store irradiance of diffuse surfaces [39] [8]. Each surface represented in the texture is uniquely mapped to it via texture coordinates. Resulting texture is known as "light map" and the process of generating a light map is called "baking". During rendering, all lighting calculations of baked objects are omitted and light value sampled from light map is used instead. For any environment, size of the light map is only dependent on the surface area of mapped objects and texel size in world units. Therefore, neither required memory nor rendering time increases as following properties increase:

- Quality of captured lighting
- Vertex count
- Light count.

One of the very interesting usages of textures to store indirect illumination was used in game Prototype 2 [2]. Global illumination needs to be rendered for large game zones as well as static and dynamic objects in it. Player also has the ability to climb buildings and glide down which requires a solution that works on multiple height levels. A single 512x512 texture with mipmaps was used to store indirect illumination and ambient occlusion of an entire zone. An example texture can be seen in Figure 2.3. Since most of the details are on the ground, including street lights, stores, walls, trees, post boxes; bouncing light is mostly present at this level. Therefore, the original light map is used at ground level. As height increases, lower quality mipmaps are used. First mipmap levels from the ground has only a few meters between them, while higher level mipmaps cover larger distances in height.

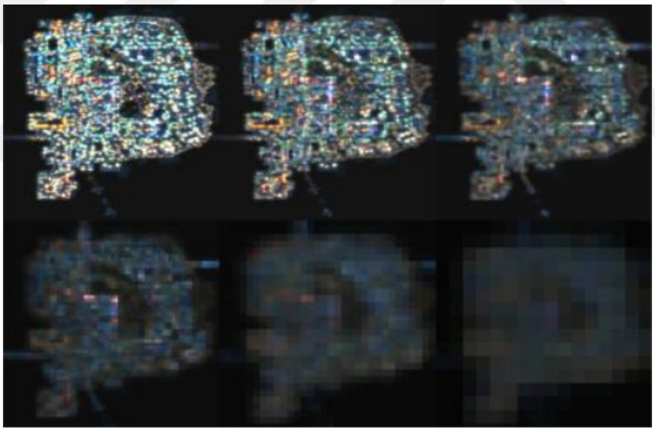


Figure 2.3: Each mipmap level stores the indirect lighting of a horizontal slice of the zone. Each pixel stores the indirect lighting of a vertical slice within the horizontal slice. Image taken from [2].

2.3.1 Baking

There are two common methods used for storing baked data in scene. First one involves storing light data in vertex attributes. Second method uses textures and samples light value of each pixel in pixel shader [40]. Vertex-based method is more efficient since it doesn't require large textures and it doesn't do texture sampling at pixel shader. It also avoids costs of generating non-overlapping UV coordinates for triangles prior to baking. For large polygons under detailed lighting, however, storing light values in vertices often results in low quality results [41]. Texture-based method offers better details independent of triangle size if memory requirements can be met.

To cope with the shortcomings of vertex-based method, one of the very common methods is dividing existing polygons into smaller ones [41] [42] [43] [9]. Li et al. [44] subdivided mesh into smaller triangles where required. An input lightmap texture is analyzed and necessary subdivision level is calculated for each triangle. This results in a higher number of triangles. However, lightmap texture can be completely removed after vertex baking is complete. According to the presented results, rendering time of vertex-baked subdivided meshes are comparable to texture-baked unmodified meshes, while memory requirements are significantly reduced [44].

For most of the triangles in a typical scene, vertex-baking is sufficient. Rest of the triangles, however, cannot be lit with vertex stored light data since it is not possible to store high frequency details on only 3 vertices. Schäfer et al. [45] merged vertex-baking and texture-based-baking into a hybrid solution. Proposed method uses vertices to store lighting by default for efficiency reasons. For triangles where vertex storage is insufficient and quick changes in lighting causes artifacts, textures are used. An algorithm first detects for which triangles the texture-based approach is required. Texture coordinates for light map sampling are only generated for these triangles. To prevent artifacts at the shared edge of two triangles that use different methods, a custom shader is used for blending.

2.3.2 Reacting to Changes

In most light baking methods, a change in the scene requires a complete re-baking of light maps, even if the change is only affecting a small part of the scene. Long baking times prevent designers to quickly iterate and achieve the desired look. Using a Many-Light Global Illumination approach, Luksch et al. [46] proposed a method to update only changed parts of the light map. Procedure involves detecting virtual point lights that were affected by the changes in the scene. These VPLs are then removed from the corresponding point cluster and new VPLs are generated according to the new state of the scene.

In another study, Luksch et al. [47] has combined multiple virtual point lights into virtual polygon lights to reduce baking time of light maps. Despite the overhead of grouping VPLs into polygon lights, this method provides a significant speedup to overall baking process. To allow interactive editing, light map is temporarily generated with a direct-indirect hybrid illumination method when a light source or object is moved. This process starts with subtracting the affected light from light map and a real-time direct illumination is temporarily used for these light sources. As high quality light map with indirect illumination is generated, it is blended together with existing one. While this allows faster iterations during editing, light map generation times are not low enough to be used in real-time applications, even on high-end devices. Depending on the application, however, baking light-maps at run-time is not always impossible. Hoffman and Mitchell has managed to update the light map of an outdoor environment for the changing light conditions [48]. Baking time was reduced to 6 seconds with the help of approximations and precomputations. Although this cannot be considered real-time, it provides some sense of interactivity.

2.3.3 Directional Light Maps

Conventional light maps store diffuse reflection of each texel and view direction does not change how the rendered point looks. For this reason, light maps are said to be "view-independent" [10] [9]. While this allows for faster rendering in interactive applications, it prevents using some of the view-dependent methods such as normal mapping. To add directionality to light maps, 3 light maps were used instead of one in Valve’s Source Engine [49]. Each of the light maps store light coming from one direction where each direction is a basis vector of a pre-defined basis in tangent space.

CHAPTER 3

LIGHT MAPPING

In this chapter, details of light mapping technique will be given.

Light mapping of a scene starts after models are created and placed. Light sources should also be placed and adjusted with correct color and intensity settings to produce the desired look. Rendered image of a sample scene that is ready to be light mapped is given in Figure 3.1.

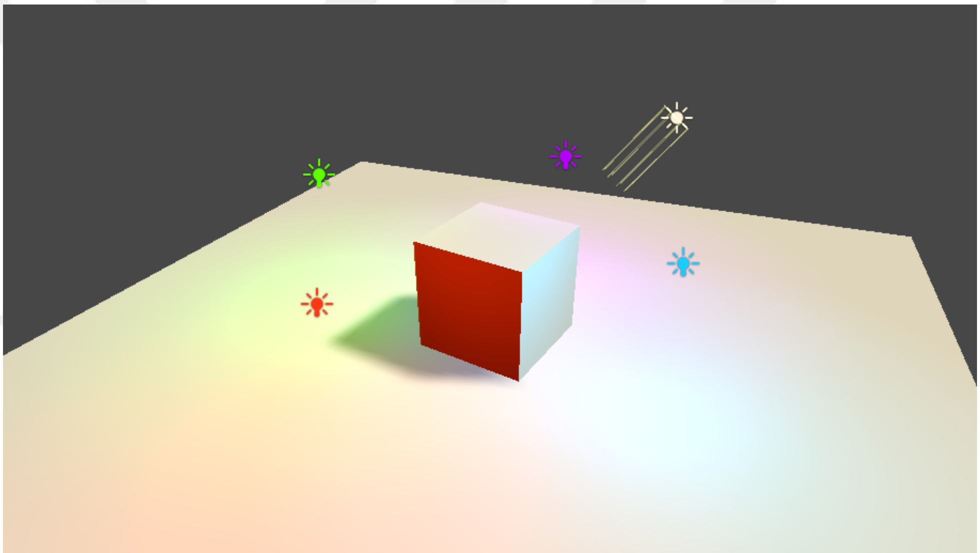


Figure 3.1: Sample scene consists of 4 point light sources of different colors and 1 slightly yellow directional light source. The only objects are a small cube and a plane.

Light mapping requires that each of the objects that will have its lights precomputed should be uniquely mapped to a region on the light map. Figure 3.2 shows how the triangles in the scene are mapped. Large rectangle on the left represents the surface of the ground plane on the light map. 6 rectangles on the bottom-right of the light map belongs to the 6 surfaces of the cube.

Since each surface is mapped to a limited number of pixels on light map, resolution of the light map is significantly important at determining the quality of the results. Especially around the areas where a hard shadow is generated by a bright light source, a higher resolution is necessary to capture the sharp changes in the lighting (Figure 3.3). As an alternative to increasing the resolution of the light map, surfaces that contain sharp changes in the lighting can be assigned to larger areas on the light map.

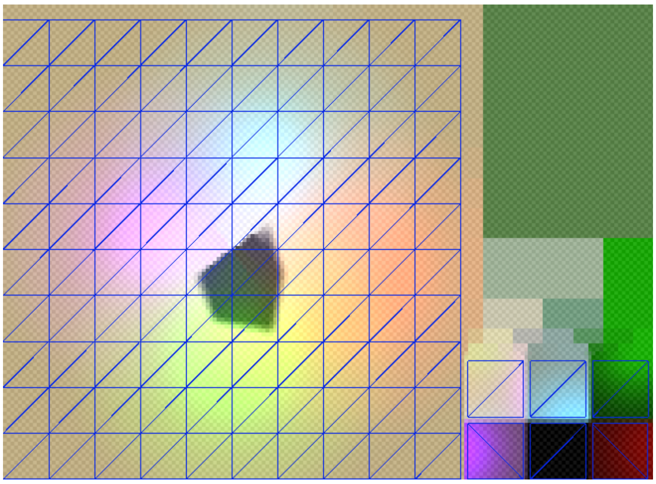


Figure 3.2: Light map uniquely maps the surface of each object.

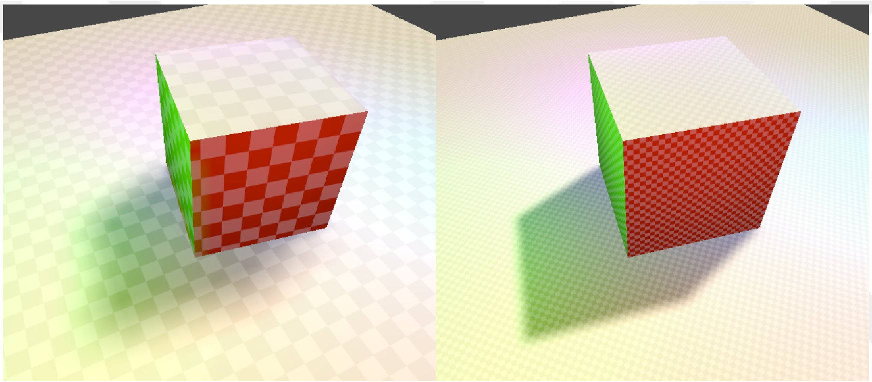


Figure 3.3: Sharp changes in the lighting cannot be correctly captured in low resolution light maps as shown in the left image. This issue is much less noticeable on the right image which uses 16 times more pixels for the same light map.

Surface area is another important factor at determining the size of the mapped area of a surface. Larger surfaces should be mapped to larger areas on the light map than small surfaces to provide a uniform light map resolution in world space. An example of this can be seen in Figure 3.2 where the plane has a much larger area in the light map than the cube.

During the generation of the texture coordinates that maps each surface to the light map, it is important to have padding between the surfaces that do not share an edge in the geometry. This necessity originates from the filtering of the light map at run-time. To avoid pixelated results caused by using point filtering during the sampling of the light map, bilinear or trilinear filtering is frequently used. This means that the value sampled from a pixel will be affected by the neighboring pixels. When sampling the light value of a surface from a light map with no padding between the surfaces, sampled value will contain some of the light from the neighboring pixel. In the cases where the neighboring pixel belongs to a non-neighboring surface, light from that surface will incorrectly contribute to the illumination.

The next and the final offline step of light mapping is the generation of light map. Finding the correct values for the initially black pixels of the light map is the main objective of this step. A ray-tracing based method like photon mapping can be used to trace the light traveling within the scene. Intensity and color of the light are stored in the light map where the photon is absorbed by the surface (Figure 3.4). This process is known as "light map baking".

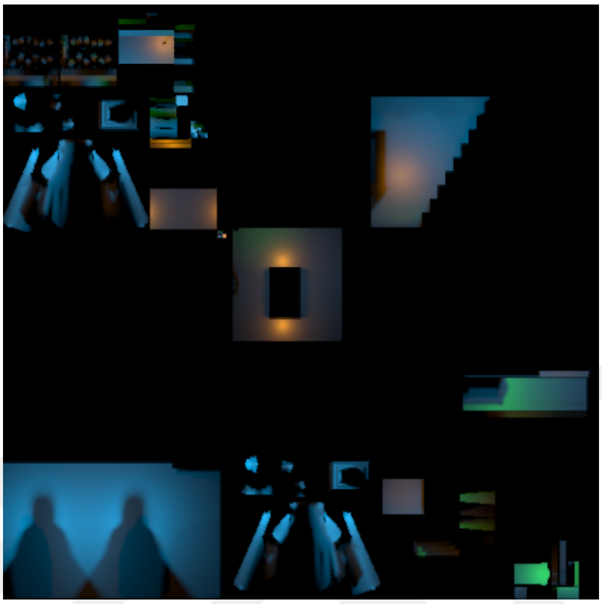


Figure 3.4: Final light map of *Dungeon* scene.

At run-time, the light map texture should be loaded onto GPU. During the color evaluation of a point on a baked surface, the light map is sampled. Sampling happens with the texture coordinates generated during the offline stage (Figure 3.5). The acquired light value is then multiplied by the albedo of the surface to produce the final color of the point.

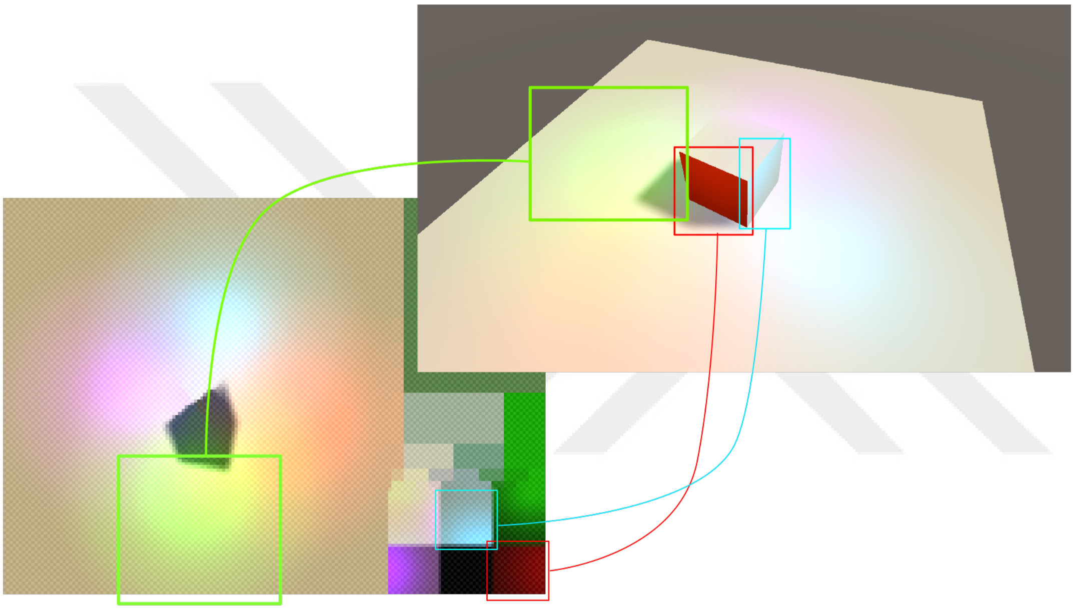


Figure 3.5: The light that is visible in the rendered image is not computed at run-time. It is simply retrieved from the light map from the correct regions.

CHAPTER 4

ALGORITHM

In this section, a detailed explanation of the algorithm will be given. Similar to classical light mapping approaches explained in Chapter 3, operations happen in 2 major stages: offline and run-time. Each stage will be explained separately.

A sample scene was used to better explain each of the steps in both stages. The scene consist of a plane, a box, a cylinder, and a sphere. There is one point light source above the objects with a slightly yellow color. A rendered image and the light map of the scene is given in Figure 4.1.

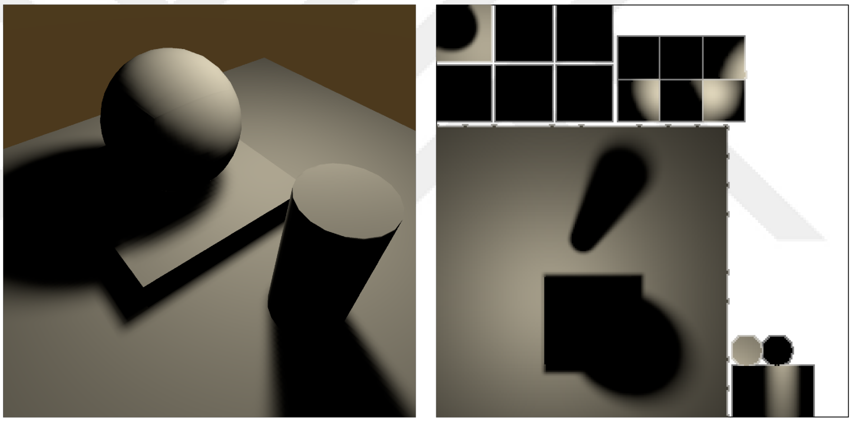


Figure 4.1: Image on the left shows the rendered sample environment. Right image is the light map.

4.1 Offline Stage

4.1.1 Marking of Semi Dynamic Objects and Light Sources

Each baked light source that will have semi dynamic properties needs to be processed during offline stage. This process is time consuming as it requires multiple light map baking operations. It also increases the amount of memory used at run-time. Therefore, it is important to consider a light source "semi dynamic" only if its properties need to be modified at run-time. At this step, we mark each light source that needs to be semi dynamic. This allows us to distinguish fully static light sources from semi dynamic light sources during both offline and run-time stages.

Similarly, each baked object that needs to be shown or hidden at run-time needs to be processed accordingly during offline stage. This increases the time spent on precomputations as well as memory requirements at run-time. To avoid these costs, an object should only be considered semi dynamic if its visibility state needs to be changed at run-time. Together with light sources, each object that needs to be semi dynamic is marked at this step. This allows us to distinguish fully static objects from semi dynamic objects during both offline and run-time stages.

In the sample scene mentioned above (Figure 4.1), sphere, cylinder, box, and the only light source of the scene was marked as semi dynamic. Ground plane is not marked. It will be fully static at run-time.

4.1.2 Calculating Contribution of Static Light Sources

Static light sources and objects are known to not change during run-time. Therefore, generating separate maps to store their contributions is unnecessarily. Very similar to generating a conventional light map, baking the light maps with these light sources and objects can be used to calculate the contribution of static components of the scene.

Algorithm 1 explains this process.

Algorithm 1 Calculating the contribution of static light sources and objects

```
for each light source $l$ that is marked as semi dynamic do
    Turn off $l$
end for
for each object $o$ that is marked as semi dynamic do
    Set $o$ to not cast any shadows
end for
$B \leftarrow$ Generate light maps by "baking", which will contain the contribution of
    static light sources and objects.
return $B$
```

Resulting map is called the "Base Map" of the scene and can be represented with B.

4.1.3 Calculating Contribution of Each Light Source

To be able to change properties of a light source, its contribution to scene's lighting needs to be known. In this step, we will calculate contribution of semi dynamic light source l to the light map of the scene. Resulting map is called "Contribution Map of l" and it is represented by C_l.

This process is explained in Algorithm 2.

Acquired contribution map is a single channel texture that stores all the texels that are illuminated by the light source as well as the intensity of the light at each texel.

Algorithm 2 Calculating the contribution of light source

```
for each light source $k$ in the scene do
    Turn off $k$
end for
for each semi dynamic object $o$ in the scene do
    Set $o$ to not cast any shadows
end for
Turn on $l$
$l.color \leftarrow white$
$l.intensity \leftarrow 1$
$t \leftarrow$ Generate the light map of the scene.
Remove 2 of the channels of $t$. {Since all the channels store the same data, which
    2 of the channels to discard is unimportant.}
return $t$
```

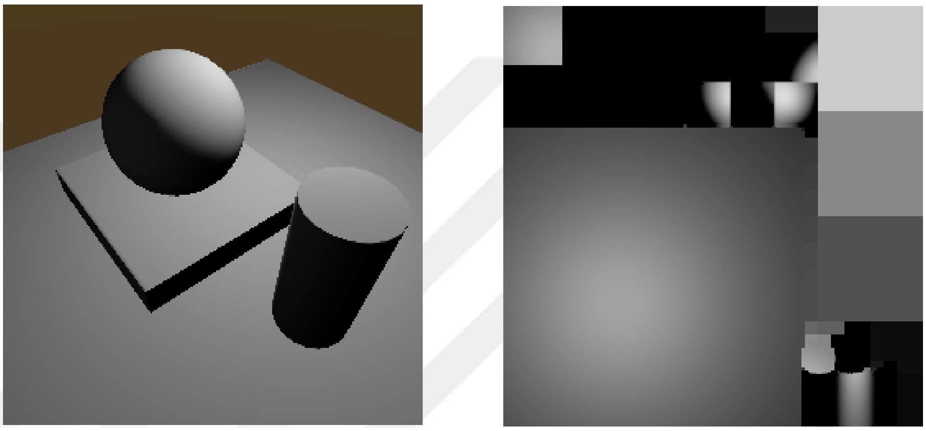


Figure 4.2: Image on the left shows the state of the scene during the calculation of the contribution map of the light source. Image on the right is the contribution map of the light source.

As mentioned, color and intensity of the light source was set to white and 1 respectively. This allows us to store the contribution maps in a more compact way. If color of the light source was preserved in this process, generated map would contain 3 channels. However, since each texel in this map is enlightened by the same light source, color of each pixel would be the same. Storing the color as auxiliary data instead of including it in every pixel on the map significantly reduces the storage space costs as only one of the channels needs to be stored. Setting intensity of the light source to 1 allows us to ignore it at run-time stage calculations. This way, contribution map of a light source can directly be multiplied by the desired intensity value at run-time.

4.1.4 Calculating Contribution of Each Object

In classical light mapping, changes to the baked objects at run-time are not reflected to light map. In most cases, this results in a completely wrong lighting of the scene. For instance, a baked cube under a light source will continue to cast shadow even after it was removed from the scene. It is necessary to know all the affected pixels of the light map as well as the severity of the effect at each pixel to correctly remove the shadow. In this stage, we calculate the effect of each object to the scene's illumination. With semi dynamic lights, however, scene does not have a single state. Whenever a light source is turned on or off, scene state changes. Effect of the object to the illumination of the scene should be calculated separately for each state. This quickly becomes unmanageable as semi dynamic light source count increases, since a scene with n light sources has 2^n states.

Fortunately, an object's effect does not need to be calculated separately for each state of the scene. Instead of storing the effect of the object to the light map, we can store the effect of the object to a light's contribution map. In this case, state of other lights becomes irrelevant for two reasons:

1. A light source's contribution to a scene's light map is independent of other light sources.
2. Amount of light of a single light source that is blocked by an object is independent of other light sources in the scene.

Therefore, the effect of each object to each light's contribution map is calculated. For each semi dynamic object, n calculations are necessary for a scene with n semi dynamic light sources.

Steps given in Algorithm 3 is executed to calculate the effect of semi dynamic object o to semi dynamic light l.

Algorithm 3 Calculating the contribution of semi dynamic object o to semi dynamic light l

```
for each light source $k$ in the scene do
    Turn off $k$
end for
for each semi dynamic object $m$ in the scene do
    Set $m$ to not cast any shadows
end for
Turn on $l$
$l.color \leftarrow white$
$l.intensity \leftarrow 1$
Set $o$ to cast shadows
$t \leftarrow$ Generate the light map of the scene.
Remove 2 of the channels of $t$. {Since all the channels store the same data, which
    2 of the channels to discard is unimportant.}
$t \leftarrow C_l - t$ {Illustrated in Figure 4.3}
return $t$
```

Resulting map is called the "shadow map of object o on light l" and it can be represented with S_{o_l}.

The acquired shadow map stores all the texels that were illuminated by light source l and was shadowed by object o. Value of each pixel also stores how much of the light

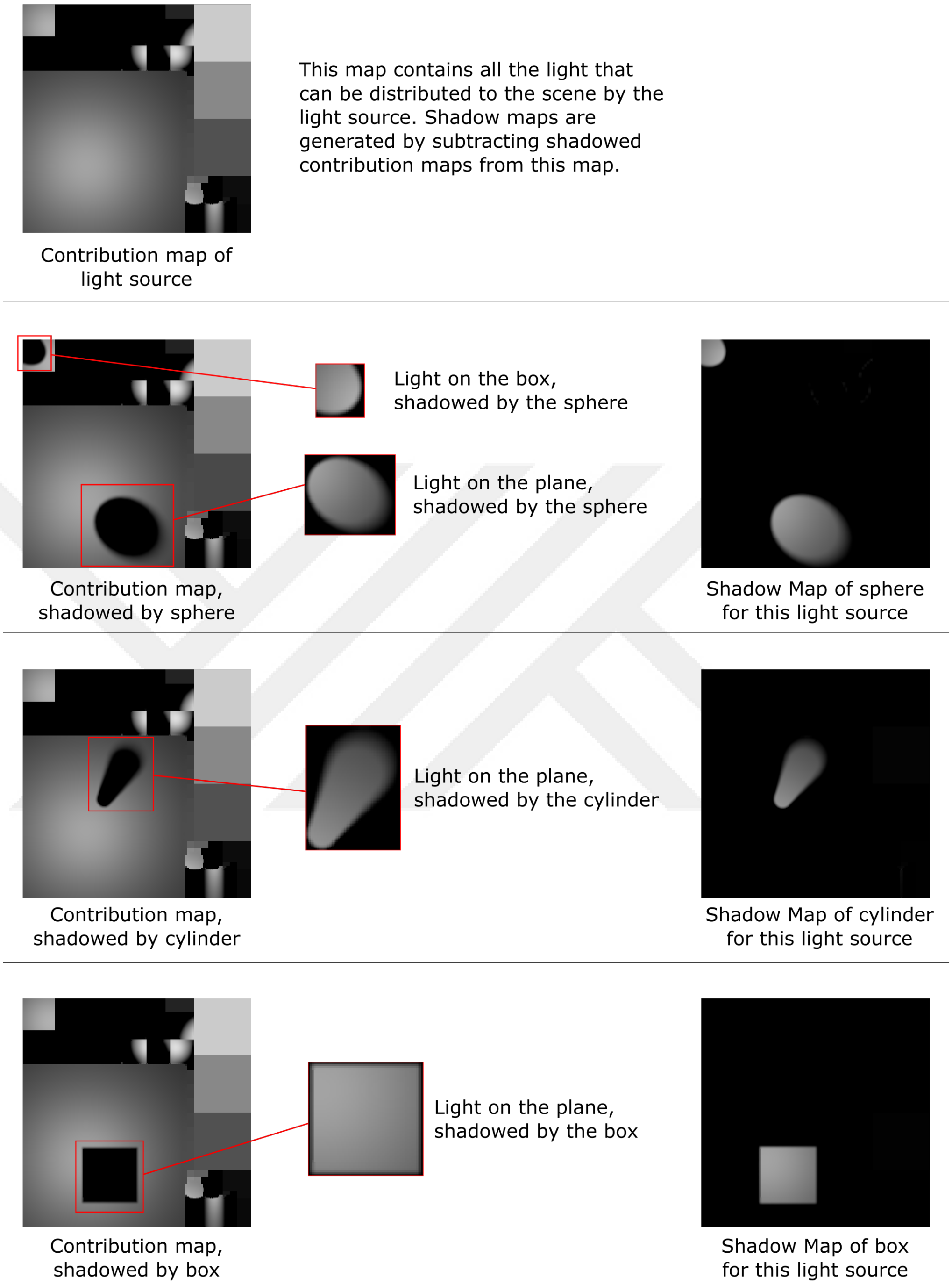

This map contains all the light that can be distributed to the scene by the light source. Shadow maps are generated by subtracting shadowed contribution maps from this map.
Contribution map of light source
Light on the box, shadowed by the sphere
Light on the plane, shadowed by the sphere
Contribution map, shadowed by sphere
Shadow Map of sphere for this light source
Light on the plane, shadowed by the cylinder
Contribution map, shadowed by cylinder
Shadow Map of cylinder for this light source
Light on the plane, shadowed by the box
Contribution map, shadowed by box
Shadow Map of box for this light source

Figure 4.3: Shadow maps store how much light is shadowed by an object from a light source. On the left, shadowed contribution maps that are used to generate shadow maps is shown for each object. Resulting shadow maps are shown on the right.

was shadowed. For fully opaque objects, intensity of the shadow is the same as the intensity of the light at that point. This results in a completely black shadow since no light can pass through the object. For semi-transparent objects, however, some of the light may still be illuminating the texel. Semi-transparent objects are beyond the scope of this thesis.

4.1.5 Reducing Texture Size

Light maps store the light falling onto the polygons of baked objects. Each polygon that receives light should have a unique area on the light map. This means that increasing the polygon surface area causes an increase in the light map size. It is possible to reduce the light map size by removing polygons that don't receive any light. However, this is rarely useful as almost all of the polygons in a typical scene receive some light from a light source.

Contribution maps are different in this regard. Despite being as large as the light map of the scene, contribution maps only contain light from a single light source. For scenes where a single light source only illuminates a small part of the environment, contribution maps contain many black regions with zero light. These regions can be removed to reduce the memory requirements.

The same logic can also be applied to shadow maps. In an exemplary shadow map S_{o_l}, only the region that was being illuminated by light source l which was then shadowed by object o contains non-zero pixels. This corresponds to a small area within the texture as can be seen in Figure 4.4. All the remaining area in the map contains black pixels which can be discarded.

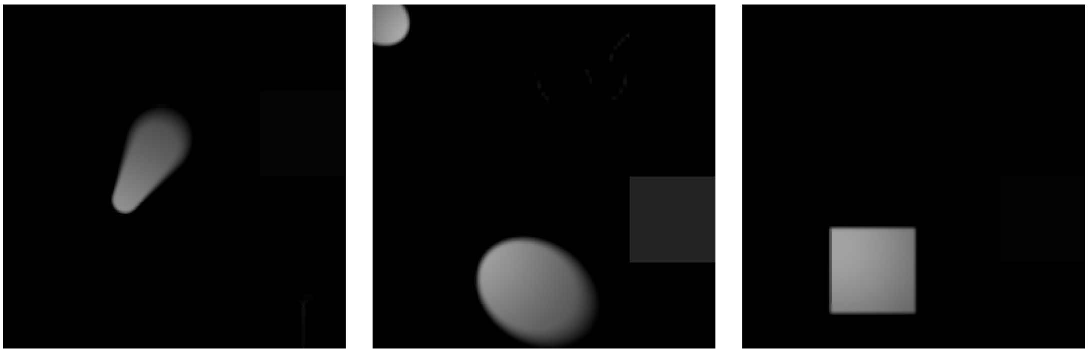


Figure 4.4: Shadow maps are mostly black and contain little data. Unused texture space can be saved.

Similar to contribution and shadow maps, base maps may also contain unused sections. In fact, in scenes where there are no static light sources, base map is completely black.

For each base map, contribution map, and shadow map that was generated so far, Algorithm 4 is executed.

Each of the textures generated at line 14 is called a "Light Patch".

Algorithm 4

```
for each map $m$ from all base, contribution, and shadow maps do
    for each baked object $o$ in the scene do
        let $b$ a bounding box on the light map with zero area
        let $a \leftarrow o.mappedAreaOnLightMap$
        for each pixel $p$ in $m$ within the borders of $a$ do
            if $p \neq black$ then
                enlarge $b$ to contain $p$
            end if
        end for
        if $b.area \neq 0$ then
            $t \leftarrow$ create a black texture with dimensions $b.width$ and $b.height$
            $t.channelCount \leftarrow m.channelCount$
            Copy all the pixels within $b$ from $m$ into $t$
            Store $t$
        end if
    end for
end for
```

Reconstruction of base maps, contribution maps, and shadow maps from light patches will be necessary in later stages. However, this cannot be done using only light patches. This is because it is not known from which position each light patch was taken. Therefore, position data of each light patch needs to be stored. This data is stored in an array as pairs where each pair consists of

- Texture, representing the light patch,
- Position in the light map that the given patch was taken from.

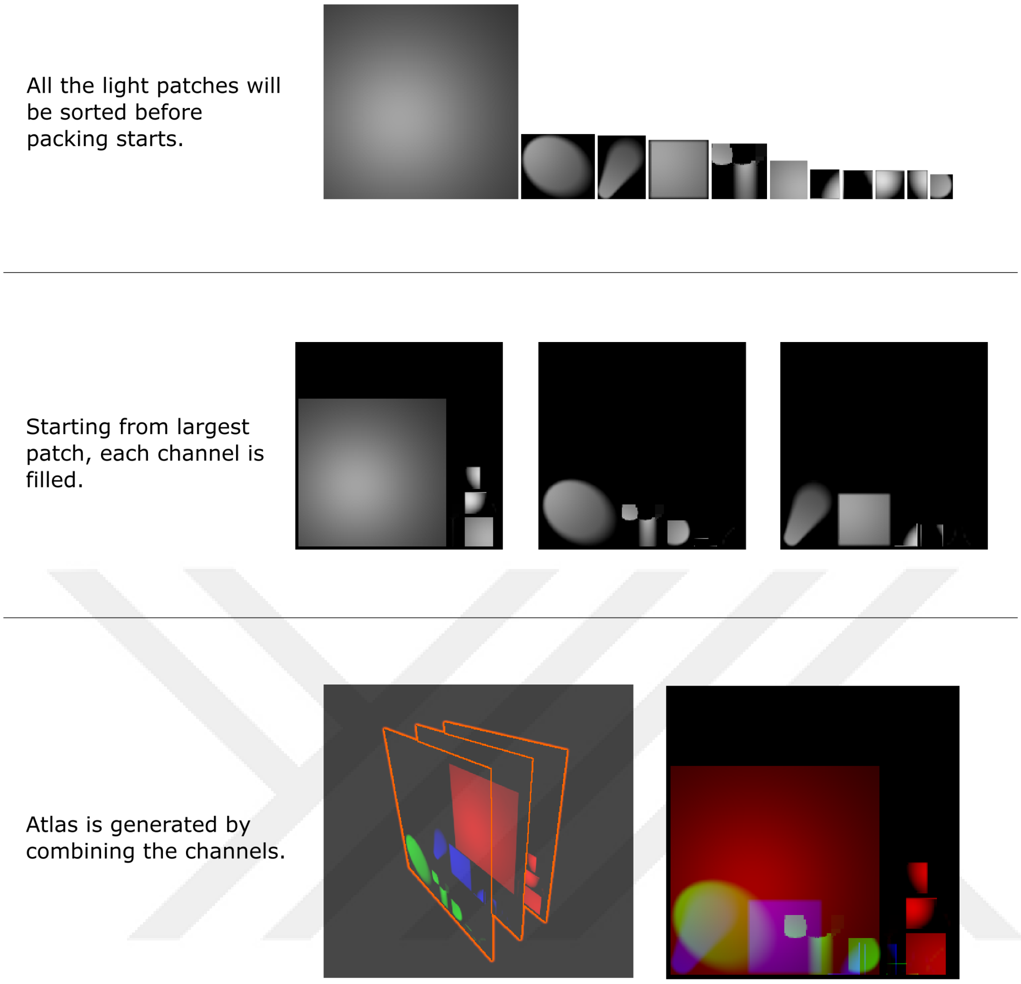
In our implementation, position represents the index of the leftmost-bottom pixel of the light patch in light map.

In the last step, many textures were generated. These textures are significantly small compared to the size of the light map. They also store only a single channel data. Using many small textures creates a significant overhead during rendering compared to using a few large textures, since it is not possible to take advantage of batching [50], [51].

In this step, all the light patches are combined into large textures called *atlases*. Because of hardware limitations on texture sizes, it is not always possible to put all the light patches into a single texture atlas. In this case, multiple smaller atlases are generated.

Figure 4.5 gives a brief explanation of the process, which happens in the following steps:

```
let $patches$ be an array containing all the light patches.
let $postponedPatches$ be an empty array.
sort $patches$ first by $channelCount$ and then by $max(width, height)$, both in decreasing order
```



All the light patches will be sorted before packing starts.

Starting from largest patch, each channel is filled.

Atlas is generated by combining the channels.

Figure 4.5: Light patches are placed into atlases, where each channel contains some of the patches.

```
4:  while $patches.size() \neq 0$ do
5:      $t \leftarrow$ An empty atlas texture of the smallest size with 3 channels is created. {In our implementation, smallest size is 256x256 pixels.}
6:      for each light patch $p$ in $patches$ do
7:          select a channel $c$ in $t$
8:          $area \leftarrow$ starting from the leftmost-bottom pixel, search $c$ for an empty area large enough to store $p$
9:          if $area$ is found then
10:             copy $p$ into area $area$ of channel $c$ of texture $t$
11:         else if not searched in all the channels yet then
12:             go to line 7 and try with a different channel
13:         else
14:             $option \leftarrow DECIDE(enlarge, createnewtexture)$ {Atlas size is insufficient. Either a new atlas will be created or $t$ will be enlarged. Decision making of this process will be explained later.}
```

```
15:       if $option = enlarge$ then
16:          enlarge the texture
17:          goto line 8
18:       else
19:          place $p$ into $postponedPatches$
20:       end if
21:    end if
22:  end for
23:  $patches.clear()$
24:  $SWAP(patches, postponedPatches)$
25: end while
```

In line 14, a decision needed to be made to decide whether to create a new atlas or to double the size of the existing atlas. It is not possible to choose one option over the other since both options has some advantages and disadvantages. Always choosing to enlarge the existing atlas creates unnecessarily large atlases in most cases. Always choosing to create a new atlas results in a large number of small atlases, which defeats the purpose of atlasing.

For instance, consider the case where an atlas of size X was generated and filled with light patches. Placing the next light patch has failed because of insufficient space in the atlas. Atlas size will be increased to $4X$, of which $3X$ will be completely empty, if we choose to double width and height components. If light patch is small and there are no other light patches to fill the rest of the space, most of the $3X$ will be wasted. If, however, total size of the rest of the light patches is close to or greater than $3X$, it results in a better batching to store all of them in the existing atlas, instead of creating a new one.

Therefore, the choice is made by comparing size of the current atlas $Size_a$ to the total size of all the remaining light patches $Size_l$. If

$$Size_a * 2 < Size_l$$

and current atlas is below the texture size limit of the hardware, dimensions of the atlas are doubled. Otherwise, a new atlas is created.

In later stages, atlases will be used to access each light patch. To be able to retrieve a light patch from generated atlases, following data is necessary:

- Index of the atlas
- Channels containing the data of the light patch
- Position of the leftmost-bottom pixel of the patch within the atlas
- Size of the patch

This data is stored in a file to be used in run-time stage.

4.2 Run-Time Stage

In light mapping, static objects use light maps to correctly illuminate surfaces. At this stage will be focusing on constructing a light map from light patch atlases that were generated during offline stage. Unlike conventional light maps, constructed light map will contain correct illumination of baked surfaces according to the scene state at run-time.

Steps in this stage will need to be executed at least once at the beginning of the application to generate the first light map. First light map can be used until the scene state changes. A change in the scene's state can be caused by

- changing the intensity of a semi dynamic light source
- changing the color of a semi dynamic light source
- showing or hiding a semi dynamic object.

Since existing light map cannot be used to correctly illuminate the scene after a change, generating a new light map is necessary before rendering the next frame.

4.2.1 Reconstructing Base Maps

Before processing of the semi dynamic objects and light sources begin, static light sources are handled. At this step, base map that was created during offline stage will be reconstructed using light patches. This process is given in Algorithm 5.

Algorithm 5 Reconstructing the base map

```
A three-channel black texture $t$ is created. Size of this texture is the same as the
size of the scene's light map.
for each light patch $p$ in the atlas do
    if $p$ belongs to base map then
        Using the auxiliary data stored at 4.1.5, determine the original position of $p$.
        To the determined position on $t$, copy pixels of $p$ from containing atlas as
        shown in Figure 4.6.
    end if
end for
return $t$
```

4.2.2 Reconstructing Contribution Maps

Reconstruction of the light map requires combining the contribution of each baked or semi dynamic light source. Therefore, reconstruction the contribution map of each semi dynamic light source necessary.

Algorithm 6 Reconstructing the contribution maps

```
for each semi dynamic light source $l$ do
    if $l.turnedOn \neq true$ then
        continue
    end if
    $t \leftarrow$ create a single channel black texture where size of the texture is the same as the size of the scene's light map.
    for each light patch $p$ that belongs to $l$ do
        $area \leftarrow p.mappedAreaOnLightMap$
        copy $p$ into area $area$ of $t$ as shown in Figure 4.6
    end for
    $t$ is the contribution map of $l$
end for
```

This process happens in the following steps:

Each contribution map stores the unshadowed light from a single semi dynamic light source. Depending on which semi dynamic objects are displayed, some regions of these maps will need to be shadowed. There is also no color information stored yet, since contribution maps contain intensity only.

For simplicity, algorithm given here stores a light map sized texture for each semi dynamic light source. This may not be desired since required texture memory will linearly increase with each light source. Fortunately, storing all the contribution maps is not necessary. Final effect of a single light source can be calculated independently of other light sources. Therefore, it is possible to sequentially process each light source so that a total of one light map sized texture will be used for any number of light sources.

This algorithm can also be implemented in other ways to take advantage of a GPU. Instead of copying values pixel by pixel on CPU, an orthographic camera can be used to generate the same results by rendering correctly positioned quads with an additive shader. An example of this technique is explained in 4.2.3.

4.2.3 Shadowing Contribution Maps

Contribution maps store the full potential of a semi dynamic light source at illuminating a scene. Displayed semi dynamic objects, however, will shadow some of the light from these light sources.

In this step, we remove the light shadowed by semi dynamic objects to acquire "Shadowed Contribution Maps". A shadowed contribution map for light l can be represented with sC_l.

The following algorithm is executed in a virtual 3D environment and explains the steps for shadowing a contribution map:

```
for each semi dynamic light source $l$ do
    let $C_l$ be the contribution map of $l$ generated in the previous section.
```

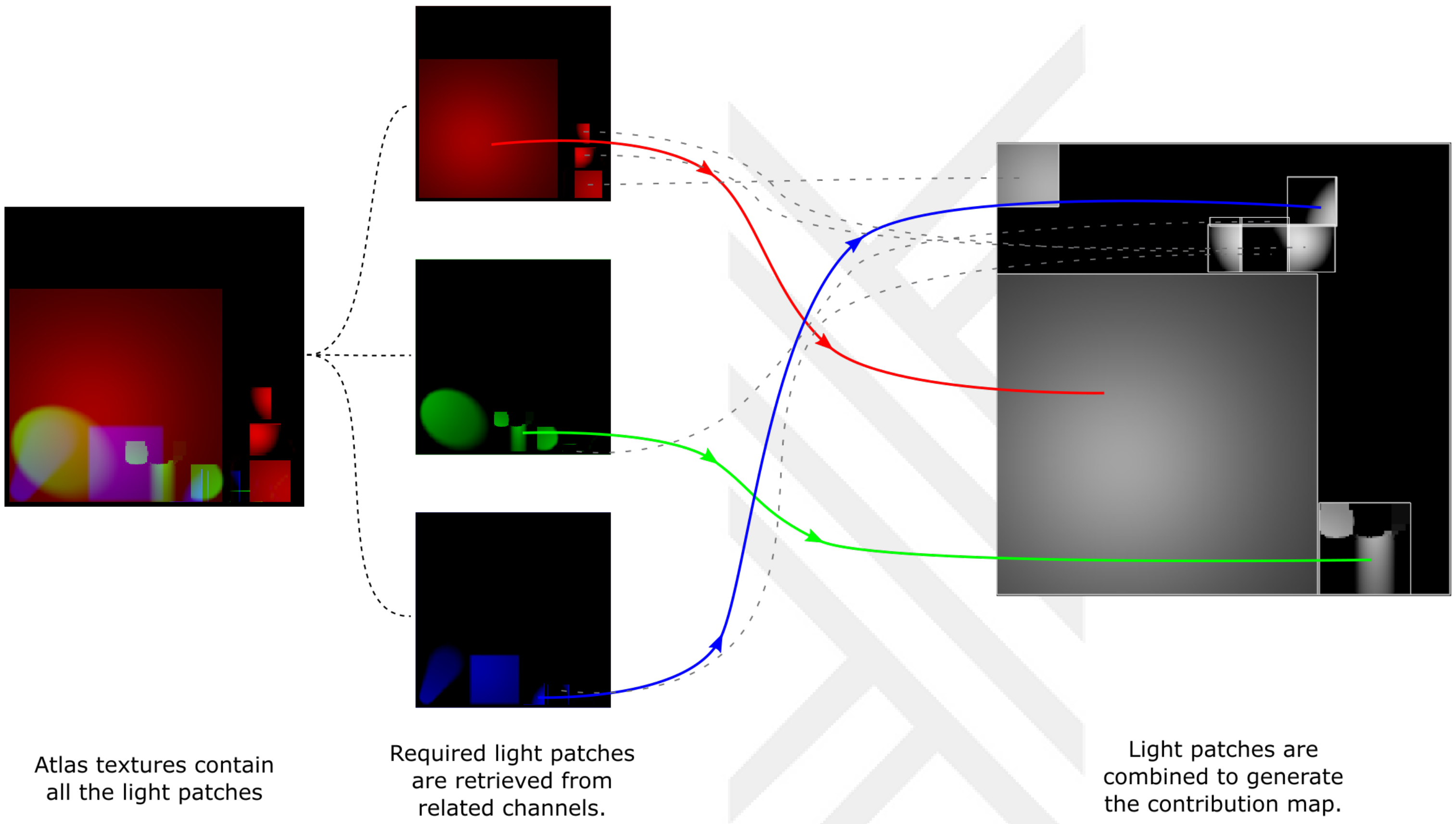
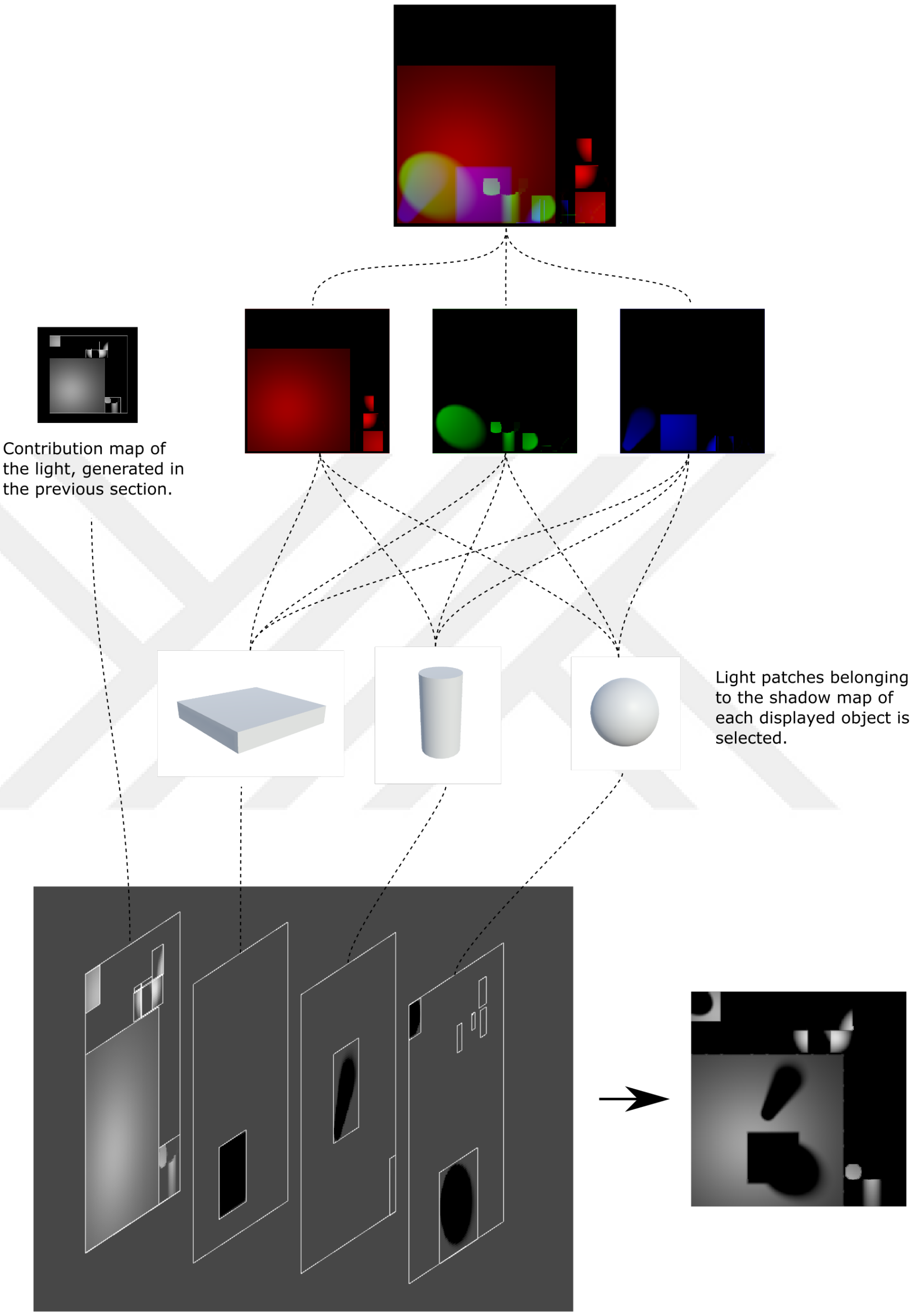

Atlas textures contain
all the light patches
Required light patches
are retrieved from
related channels.
Light patches are
combined to generate
the contribution map.

Figure 4.6: Light patches from corresponding atlases are placed onto a texture to create a contribution map. This process is repeated for each semi dynamic light source.



Contribution map of the light, generated in the previous section.

Light patches belonging to the shadow map of each displayed object is selected.

Figure 4.7: Depending on which semi dynamic objects are enabled, previously generated contribution map is shadowed.

```
 3:   let $camera$ be an orthographic camera. Aspect ratio of the camera is 1. Camera
      is set to render onto $C_l$. This can be done using frame buffers in OpenGL.
 4:   for each semi dynamic object $o$ in the scene do
 5:      for each light patch $p$ that belongs to $S_{o_l}$ do
 6:         $area \leftarrow p.mappedAreaOnLightMap$
 7:         $q \leftarrow$ a quad is placed in front of the camera. Using the auxiliary data
            generated during 4.1.5 for $p$; position, size, and texture coordinates of the
            quad is determined. From the camera's view, quad is aligned perfectly to
            the position where light will be removed from $C_l$. Texture coordinates of
            the quad map to the region that contains the light patch in the atlas.
 8:         for rendering of this quad select a shader that subtracts calculated values
            from target texture.
 9:         select the atlas texture that contains $p$ to be used by the shader.
10:      end for
11:   end for
12:   $camera.render()$ { After rendering completes, $C_l$ will have light removed
      from regions where a semi dynamic object was casting shadow. At this step,
      shadowed contribution map of light $l$ is ready.}
13: end for
```

Shader used in step 8 was programmed in a way that it subtracts values from render target. Values that will be subtracted will be sampled from light patches. Despite being stored in a 3-channel atlas, light patches in this step has only one channel. Therefore, only the correct channel should be retrieved from the atlas. This process is also handled by this shader. Which channel to use is passed to the shader as a mask stored in 3 floats. Only the selected channel has value set to 1. Values for other channels is set to zero.

```
float3 atlasSample = sampleTexture(atlas, uv);
float shadowedLight = dot(atlasSample, channelMask);
return shadowedLight;
```

In the shader code above, dot product was used to get the resulting value. This allows for retrieving the correct value regardless of the channel, while avoiding using if statements that cause branching on GPUs.

4.2.4 Generating The Final Light Map

In previous section, we have acquired shadowed contribution maps which contain the light falling onto each texel per light source. Desired light map is expected to contain *the total amount of light* falling onto each texel. In this section, we will combine shadowed contribution of each light source and the base map to generate the final light map.

One of the very important properties of light sources has been omitted so far: *color*. Since the intensity of the light source is already available in the maps, color can easily be applied by a simple multiplication.

Generation of the final light map happens in the following steps:

```
let $t$ be the light map of the scene
$t.fill(black)$
$t \leftarrow t + B$ {copy basemap onto texture}
for each reconstructed shadowed contribution map $sC_l$ do
    for each pixel $p$ sampled from $sC_l$ do
        $c = p * l.intensity$
        $c = c * l.color$ { This will convert pixel from single channel to 3-channel.}
        $t[p] = t[p] + c$ { Copy calculated value onto final light map. }
    end for
end for
$t$ contains the light map of the scene.
```

During the generation of final light map, three important actions were taken to ensure that resulting light map reflected the current state of the scene:

1. Contribution map of a light source is only processed if the light source was turned on.
2. Shadow maps of an object is processed only if the object was not hidden.
3. Values added to the final light map from a light source was multiplied by the intensity and the color of that light source.

Similar to the approach explained in section 4.2.3, operations in this section can be executed with the help of a GPU. In our implementation, a shader was used to multiply and add pixels onto target texture.

4.2.5 Updating The Light Map

Since light maps only represent the lighting of the scene for a single state, changes to the scene will invalidate the existing light map. In this case, light map should be regenerated. This can simply be accomplished by repeating the run-time stage algorithms explained so far in order. This correctly generates a new light map, and the time that is needed to update the light map is independent of how many objects or light sources were modified. However, in cases where only the contribution of a small number of light sources were changed, repeating the whole process is not needed. Existing light map can be updated with only a few steps.

Consider the case where there are n semi dynamic light sources in a scene. Contribution of a light source was changed by one of the two means:

- color or intensity of the light source was changed
- visibility of a semi dynamic object that can shadow this light source was changed

Updating the light map has become necessary. Reconstructing the light map would require reconstructing n shadowed contribution maps (one for each light source). These maps can then be colorized and combined with the base map to generate the final map. Since the update was caused by a change in the contribution of only one light source, the same light map can be generated in a simpler way as explained in Algorithm 7.

Algorithm 7 Efficiently updating the light map

```
let $l$ be the light source that was modified
let $t$ be the existing light map
$scm \leftarrow$ calculate shadowed contribution map of $l$ as explained in 4.2.3 for the scene state *before it was modified*
$scm \leftarrow scm * l.previousColor * l.previousIntensity$
$t \leftarrow t - scm$
$scm \leftarrow$ calculate shadowed contribution map of $l$ as explained in 4.2.3 for the scene state *after it was modified*
$scm \leftarrow scm * l.newColor * l.newIntensity$
$t \leftarrow t + scm$
return $t$
```

In Algorithm 7, shadowed contribution map of l was calculated and stored in scm twice. If visibility of semi dynamic objects has not changed, a single calculation is sufficient and the value of scm can be reused. Otherwise, recalculation of shadowed contribution map is necessary since newly calculated map will contain different shadows than previous one. If change happens only in the color or intensity of a light source and not in any of the objects, the algorithm can be further simplified as in Algorithm 8.

Algorithm 8 Efficiently updating the light map (Only a light source changes)

```
let $l$ be the light source that was modified
let $t$ be the existing light map
$scm \leftarrow$ calculate shadowed contribution map of $l$ as explained in 4.2.3
$c \leftarrow l.newColor * l.newIntensity - l.previousColor * l.previousIntensity$
$t \leftarrow t + scm * c$
return $t$
```

For updating the contribution of a single light source, Algorithm 7 executes 2 shadowed contribution map generation operations. If the light map reconstruction operation was executed from the beginning without using the existing light map, n shadowed contribution map generations would be needed for n light sources. Which method to use for updating the light map can be decided in the following way:

```
let $n$ be the total number of semi dynamic light sources
let $m$ be the number of semi dynamic light sources, for which the contribution must be recalculated.
if $2 * m < n$ then
    modify existing light map without a complete recalculation
else
    recalculate the light map from the beginning
end if
```

CHAPTER 5

RESULTS

In this chapter, results of the algorithm for different scenes will be presented. Baking times, storage size of generated light maps, required memory at run-time, processing costs at run-time will be compared. Input scenes used in the comparisons differ in semi dynamic light count, semi dynamic object count and total surface area of baked objects.

All tests are executed on a desktop computer with following specifications:

- Intel i7-4790 3.6GHz CPU
- NVIDIA GeForce GTX 770 2GB
- 8 GB RAM
- Samsung 840 EVO 250GB SSD

Sample scenes used in the tests are described as follows:

- *Library Scene:* is a small, rectangular, indoor environment with all four walls covered with bookshelves. Longer edge of the room has 4 bookshelves on each side where short edge has 3, adding up to 14 bookshelves in total. There are 2 additional bookshelf pairs at the center of the room with a small distance between the pairs. Lights are evenly distributed within the room and all the lights are at the same height from the ground. All of the shelves create some shadow on at least one other shelf. Most illuminated points in the map receive light from 3 different light sources. Scene consists of 26000 triangles defined by 40000 vertices.
- *Cemetery Scene:* is a square outdoor environment with a single large plane as the ground. On the ground, there are gravestones, trees, statues, and rocks. There is one dim directional light source in the scene. Other light sources are point light sources evenly distributed throughout the environment. Shadows of objects rarely overlap. Aside from the self shadowing of objects, all the shadow generated by the objects fall onto the ground plane. Scene consists of 55000 triangles defined by 75000 vertices.
- *Dungeon Scene:* is a room with large walls and small objects. There are statues, wheels, a table, a candle, a gravestone, and a painting in the room. There are 8

point light sources. Shadows of the objects in the room mostly do not overlap. Scene consists of 2200 triangles defined by 3400 vertices.

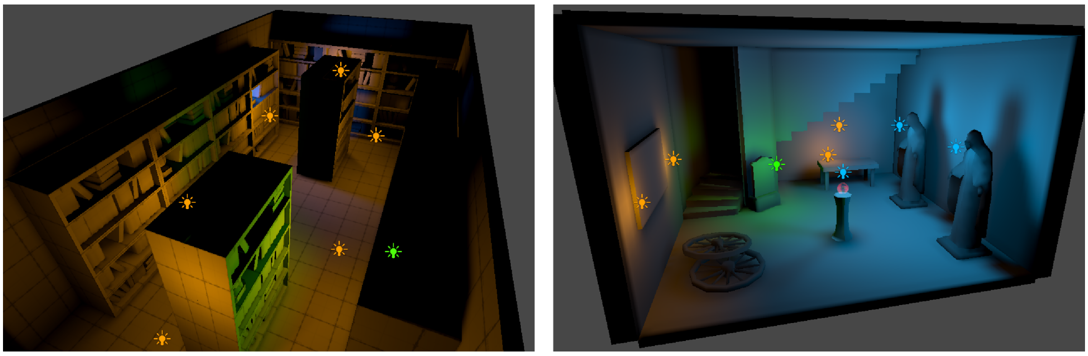


Figure 5.1: Images of samples scenes *Library* (left) and *Dungeon* (right)

5.1 Baking Times

Most time consuming process of light mapping is baking. Depending on the complexity of the scene and the quality of the results, baking can take seconds to days. Semi dynamic light maps require repeating the baking process many times. A different scene state is captured with every bake and the results are compared to generate contribution maps (4.1.3) and shadow maps (4.1.4). Therefore, time spent on baking semi dynamic light maps is significantly higher than baking of conventional light maps.

There are multiple factors that are unique to semi dynamic light maps that affect the baking times. These factors and their effects will be discussed in this section.

5.1.1 Number of Semi Dynamic Light Sources

For a scene with n semi dynamic lights and no semi dynamic objects, baking process needs to be repeated an additional n times. Initially this might look like the time spent on generating the light maps will be n times more. However, this is not the case thanks to photon mapping based light map generation approaches. For the conventional case, a single baking operation requires casting p photons per light. In total, $n * p$ photons are cast for n light sources. For semi dynamic case, each baking process only casts photons from a single light source. Each of the n operations casts only p photons which adds up to $n * p$ photons in total. Therefore, baking times increase at a much lower speed than semi dynamic light source count. This relation can be observed in Figure 5.2. Total semi dynamic light map generation times are given in Figure 5.3.

Aside from the advantage gained over photon based baking methods, there are other factors that decrease the time spent on consecutive baking operations. For instance, light map uv generation for static surfaces and scene geometry processing are only needed to be done once during the first baking operation. This data can be used in the

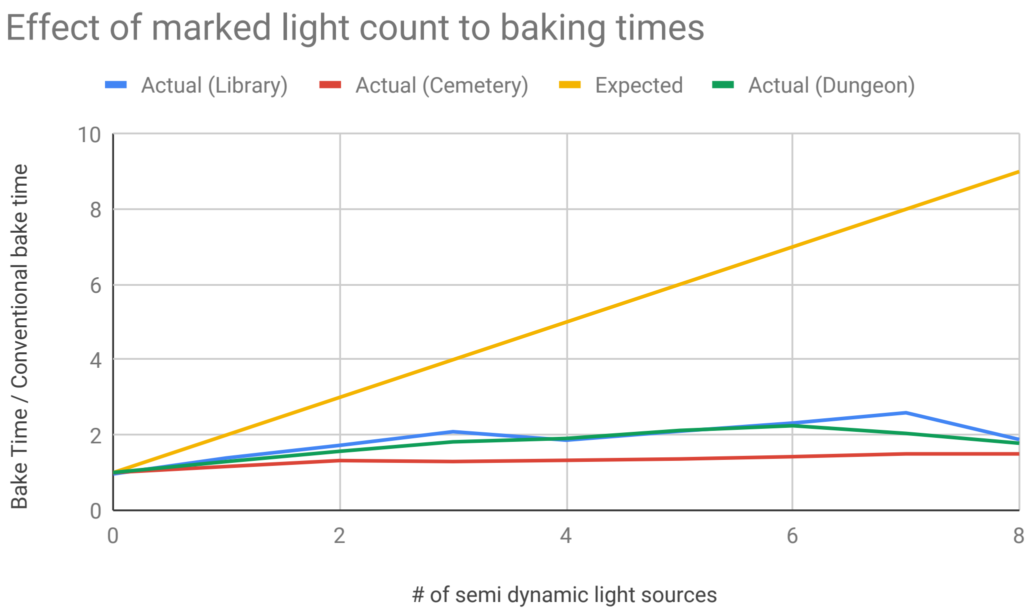


Figure 5.2: For a scene with n semi dynamic light sources, $n + 1$ baking operations are required. This looks like the baking times will be $n + 1$ times the baking time of conventional light maps. Yellow line shows this expected rate. Blue and red lines are the actual rate of bake time compared to conventional light maps. Notice how slowly bake times increase compared to the expected.

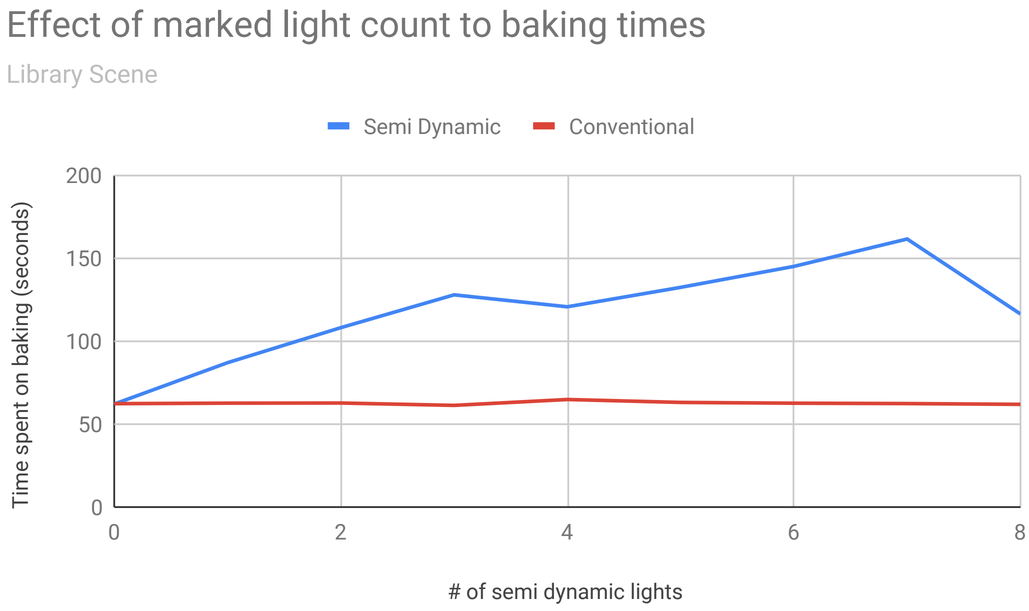
remaining bakes, further lowering the cost of additional bakings. Compared to the first bake operation, average time spent on each of the remaining bake operations can be seen in Figure 5.4.

Generation of semi dynamic light maps does not only consist of multiple executions of baking operations. There is also the cost of processing the generated maps as shown in Figure 5.5. This cost depends on many factors including light count, area of overlapping shadows, surface area of illuminated semi dynamic objects, vertex count of all the baked objects etc. It is also dependent on the implementation.

5.1.2 Number of Semi Dynamic Objects

Number of semi dynamic objects is another factor that significantly affects light map generation times. For a scene with n semi dynamic light sources, marking an object as semi dynamic requires n more baking operations. Total baking operations needed for a scene with n semi dynamic light sources and m semi dynamic objects is $1 + n * m$. However, this does not mean that the baking will take $n * m$ times more, as explained in section 5.1.1.

Similar to the results observed in section 5.1.1, average time spent on a single bake operation quickly decreases with increased number of semi dynamic objects. This can be observed in Figure 5.8.



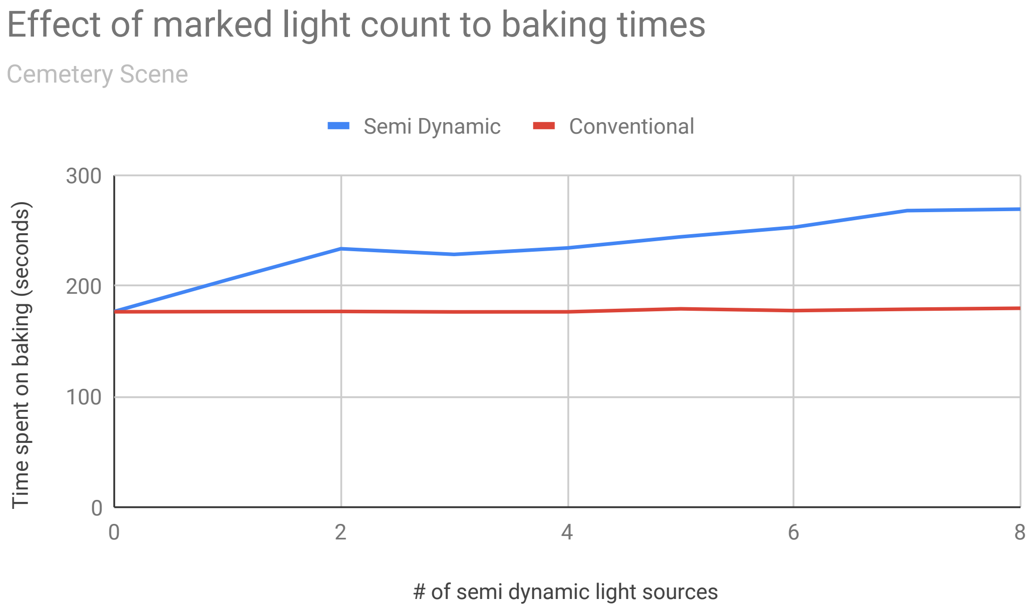


Figure 5.3: In sample *Library* and *Cemetery* scenes which contain 8 light sources, light maps were generated for 8 times where each time an additional light source is marked as semi dynamic. Each additional semi dynamic light source causes an increase in the baking time.

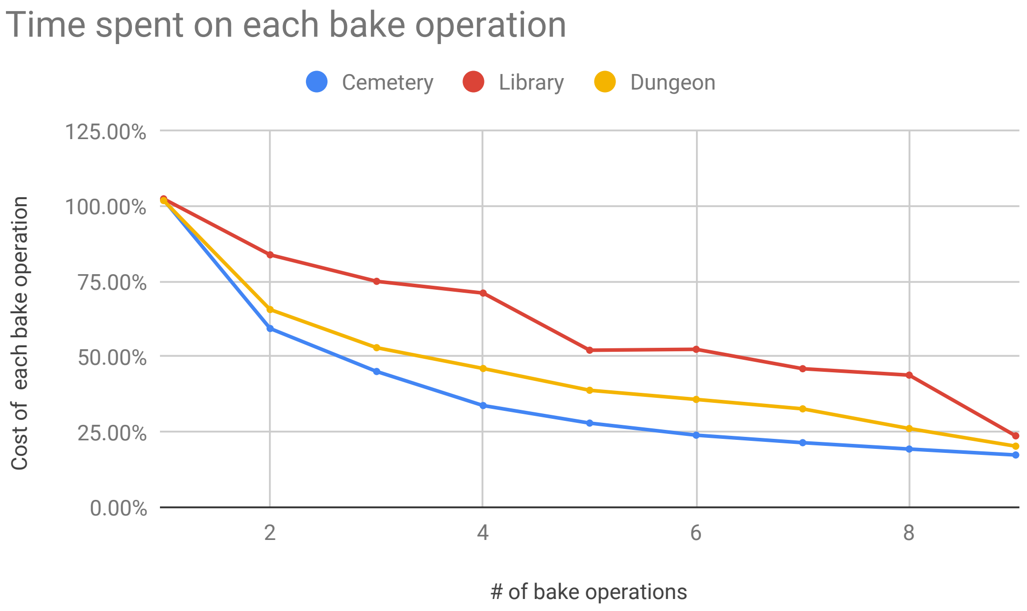


Figure 5.4: Average time spent on each bake operation decreases with each additional bake.

Number of required baking operations can be reduced if object/light interactions can be determined earlier. Normally, light maps are baked for each object/light pair. If it can be determined that the object does not receive any light from the light source, this baking operation can be omitted. Measuring the distance between the light source and the object can be used to determine the interaction. If the distance is large enough, the light reaching the object can be ignored. Note that this only holds if there are no mirror-like surfaces in the scene which reflects more light than they receive. Otherwise, light can travel any distance without losing energy.

5.2 Storage Space

Conventional light maps require storing textures that are large enough to contain uniquely mapped regions for each baked object in the scene. When surface area of baked objects increase, required light map size increases as well. Light maps do not group lights by their sources. Total accumulated light is stored in each texel. Therefore, increasing the number of light sources does not increase the size of the light map.

Semi dynamic light maps are different in this regard. Even though fully static light is still accumulated in texels, light from semi dynamic light sources are stored separately. Therefore, storage size is expected to grow as more light sources are marked as semi dynamic.

When storing light of each light source in a different map, using a single channel

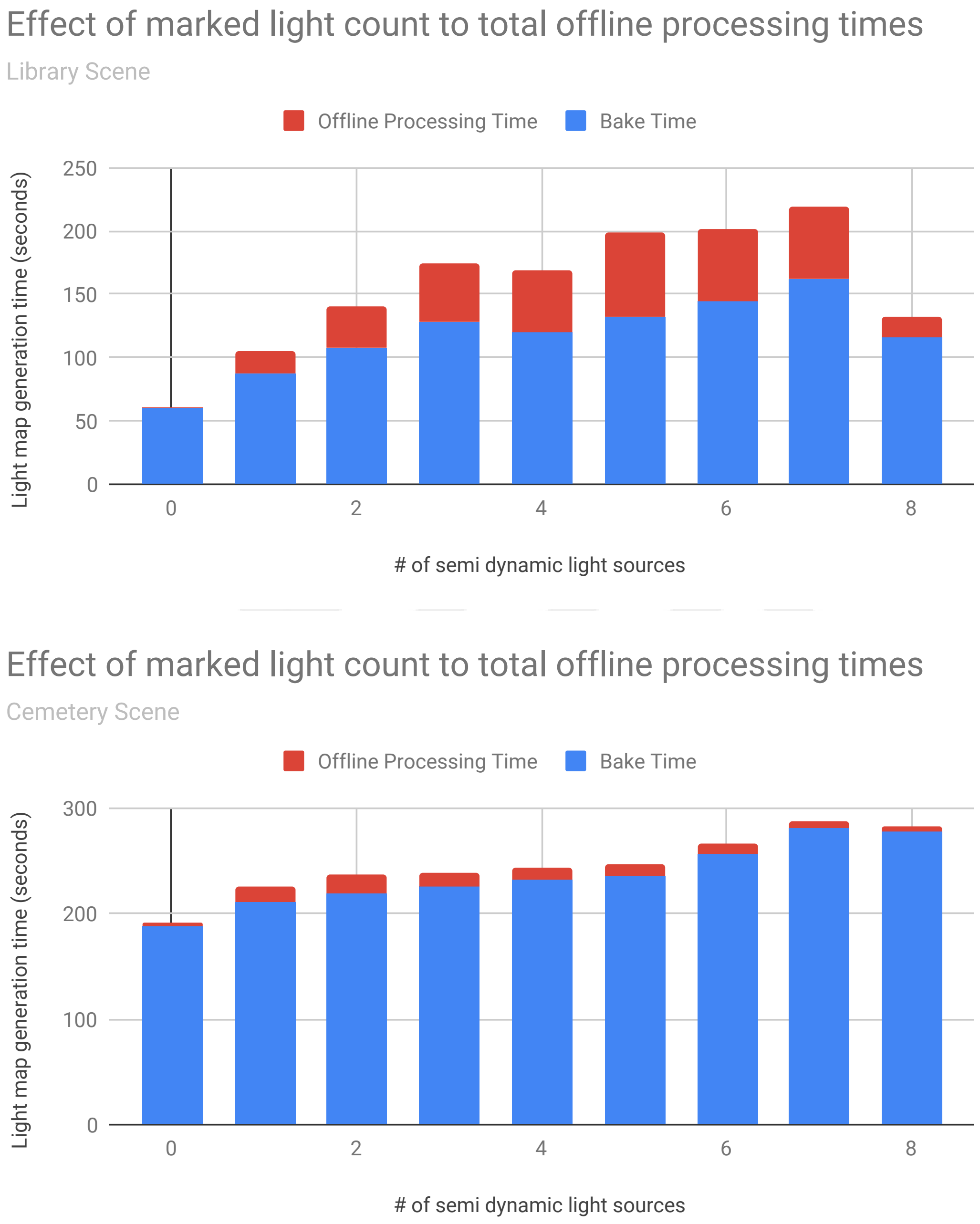
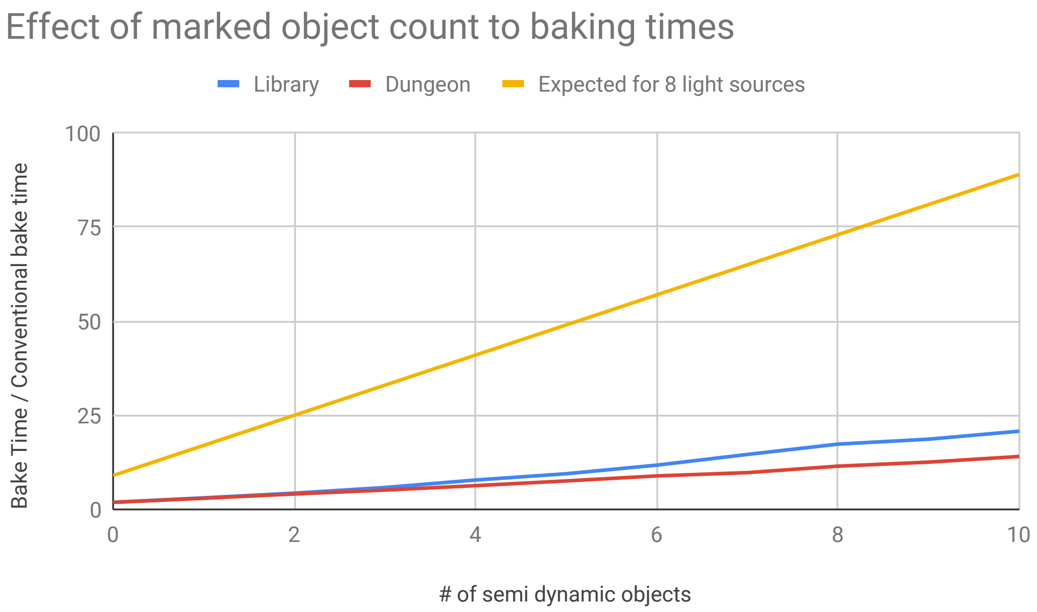


Figure 5.5: Maps generated by baking process should be processed, which increases the generation time of semi dynamic light maps. Graphs show light map generation times for *Library* (top) and *Cemetery* (bottom) scenes.



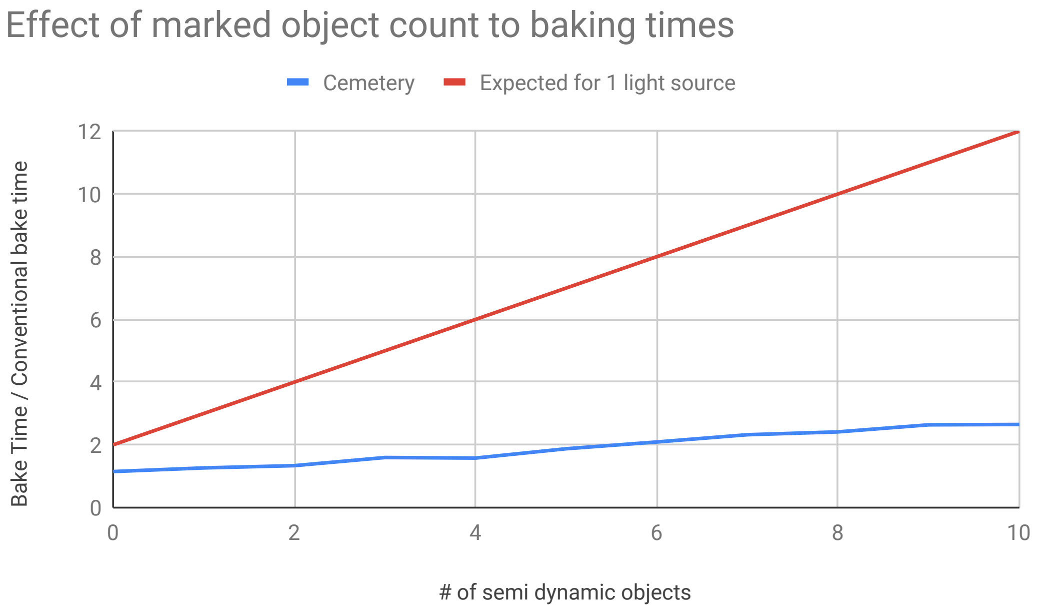


Figure 5.6: Bake times increase slowly compared to the increase in the number of semi dynamic objects. However, semi dynamic light maps can still take significantly longer to generate than conventional light maps. Note how generation of light maps for the *Library* scene where there are 8 semi dynamic light sources and 10 semi dynamic objects takes almost 24 times longer.

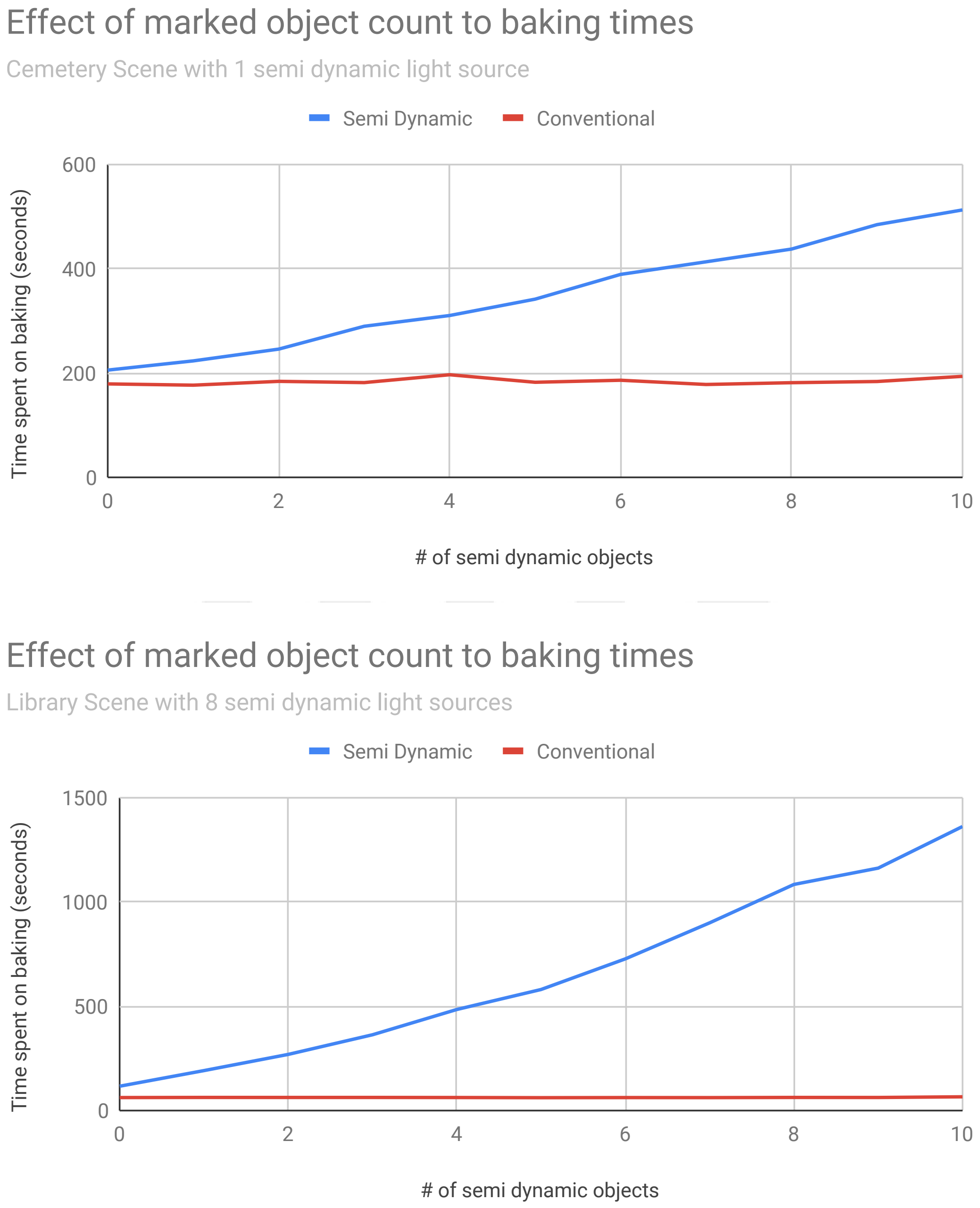


Figure 5.7: Semi dynamic light map generation times in sample scenes *Library* and *Cemetery* which contain 8 and 1 semi dynamic light sources respectively. It is important to note that baking times for *Library* scene increase much faster than *Cemetery* scene. This is expected since *Library* scene contains 7 times more light sources than *Cemetery* scene.

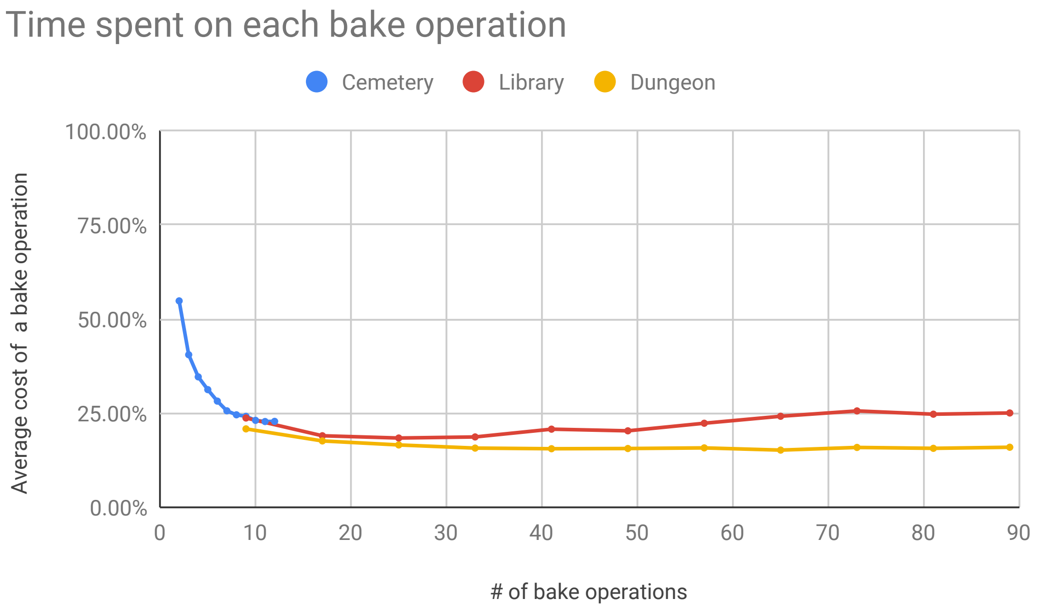


Figure 5.8: Average time spent on a single baking operation significantly decreases as the number of consecutive bake operations increases.

texture is sufficient instead of a three channel texture. This is possible because all the pixels in the map contains the same color value with different intensities. Color of the map can be stored independently in a small data structure. Since space required to store the color is very small compared to the size of the texture, storage requirement of the color can be ignored. In this case, the following can be stated:

- Using light accumulating texels, storing light from n different light sources requires exactly 3 channels.
- Using separate channels for each light source, storing light from n different light sources requires n channels.

Please note that if n is less than 3 for a texel, using separate channels for each light source occupies less storage. For this reason, storage space can increase or decrease depending on the scene. For scenes where each texel receives light from 3 or more light sources, storage space typically increases. For scenes where light from different sources do not frequently overlap, storage requirements can be lower than conventional light maps.

For performance and compatibility reasons, size of atlases and light maps generated in this thesis are all power of two (256x256, 512x512, 1024x1024 etc.). Since it is not always possible to fully fill this space, almost all maps contain some unused regions. To better present the increase in the actual used space, only the regions that contain non-zero pixels are measured in Figure 5.9.

In *Cemetery* scene, light sources are distributed coarsely and light from different light sources rarely overlap. There is only one directional light that overlaps with every other light source. Since most of the texels are only illuminated by at most 2 different light sources, storing contribution of each light source in a different channel reduces required texture space.

5.3 Run-Time Memory

Reconstruction of the semi dynamic light map is a crucial step that needs to be executed before the first frame is rendered at run-time. Reconstruction requires copying light patches from atlases into initially a black light map texture. Therefore, atlases needs to be loaded into memory.

During the reconstruction of light maps, contribution of each light source needs to be calculated. Contribution maps can only be generated on black textures. It is not possible to directly construct contribution maps on the existing light map without using an intermediate texture. Therefore, an additional texture at the size of the light map is required as well. For scenes where there are multiple textures used as light maps, only one intermediate texture at the same size with the largest light map is sufficient.

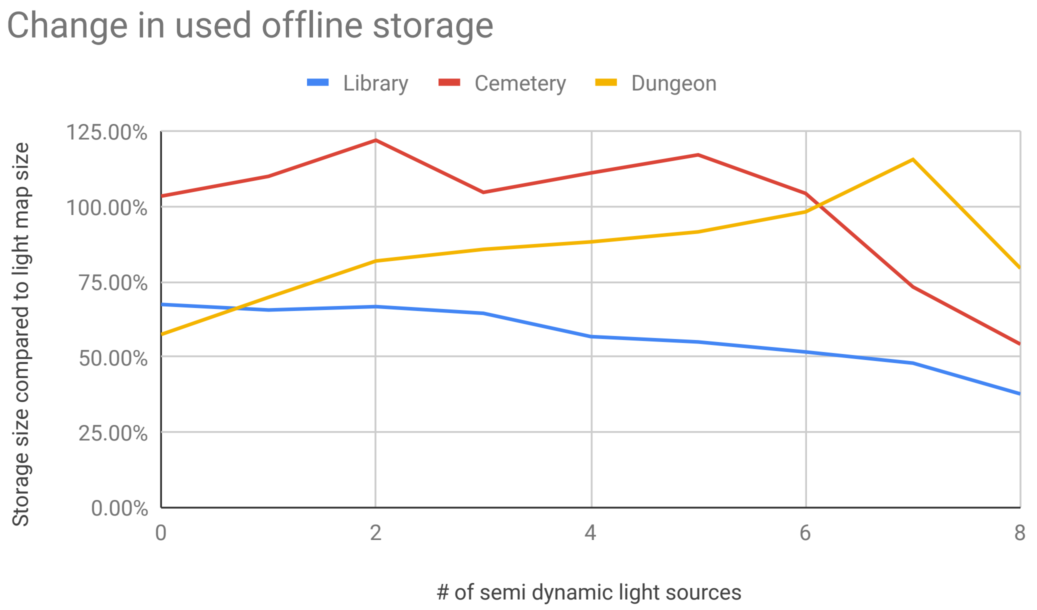
Figure 5.10 show the additional required texture memory to construct and update semi dynamic light maps in sample scenes. Since required memory is mainly dependent on the atlas size, it can be observed that the graphs for required texture memory is very similar to the graphs for required storage space.

Auxiliary data is another factor that increases run-time memory requirements. This data maps light patches from atlases to light maps. In our implementation, each mapping costs 35 bytes. Total memory required by auxiliary data in *Library* and *Cemetery* scenes can be seen in Figure 5.11. Size of this data is usually very small compared to atlas size and can be ignored.

5.4 Run-Time Processing Costs

Before the first frame and whenever the scene state changes, light maps need to be regenerated at run-time. Depending on the number of light patches, light map and atlas sizes cost of this process may increase. In our tests, generation of light maps took as low as 1 milliseconds on average. In complex scenes with 8 semi dynamic light sources and 8 semi dynamic objects, cost went as high as 1.97 milliseconds (Figure 5.12 and 5.13). Even though we believe that these numbers are low and can be afforded in many applications on a large variety of hardware, it may effect the smoothness of some applications running on low-end devices. However, it should be noted that updating of the light maps is only necessary when a change in the scene's lighting state occurs. Unless color or intensity of a semi dynamic light source changes or a semi dynamic object is shown or hidden, there is no need to execute light map generation process.

In the case of an extreme scenario where the cost of updating the light map causes a freeze noticeable by the user, update process can be divided into smaller tasks to be executed over multiple frames. In this case, contribution of only some of the lights can be calculated in each frame. During the last frame where the update will be reflected to the screen, previously generated maps can be quickly combined to generate the updated light map. When implemented in CPU, this process can also be executed in multiple threads to take advantage of multi core CPUs.



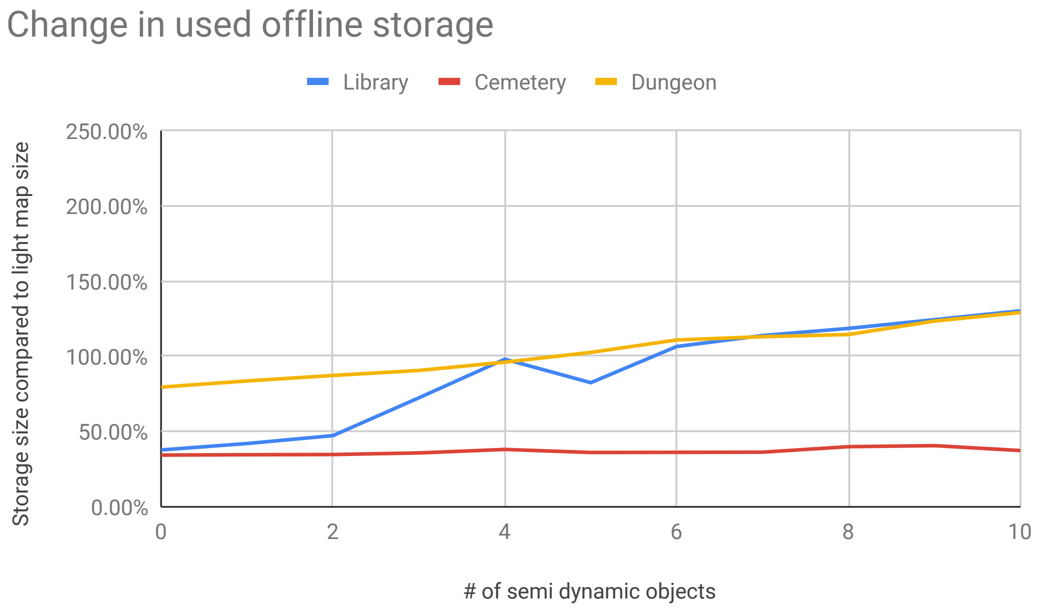


Figure 5.9: Semi dynamic light maps efficiently store light values for texels which only receive light from at most 2 light sources. This allows reduced storage size of light maps. Semi dynamic objects, however, always cause an increase in the storage.

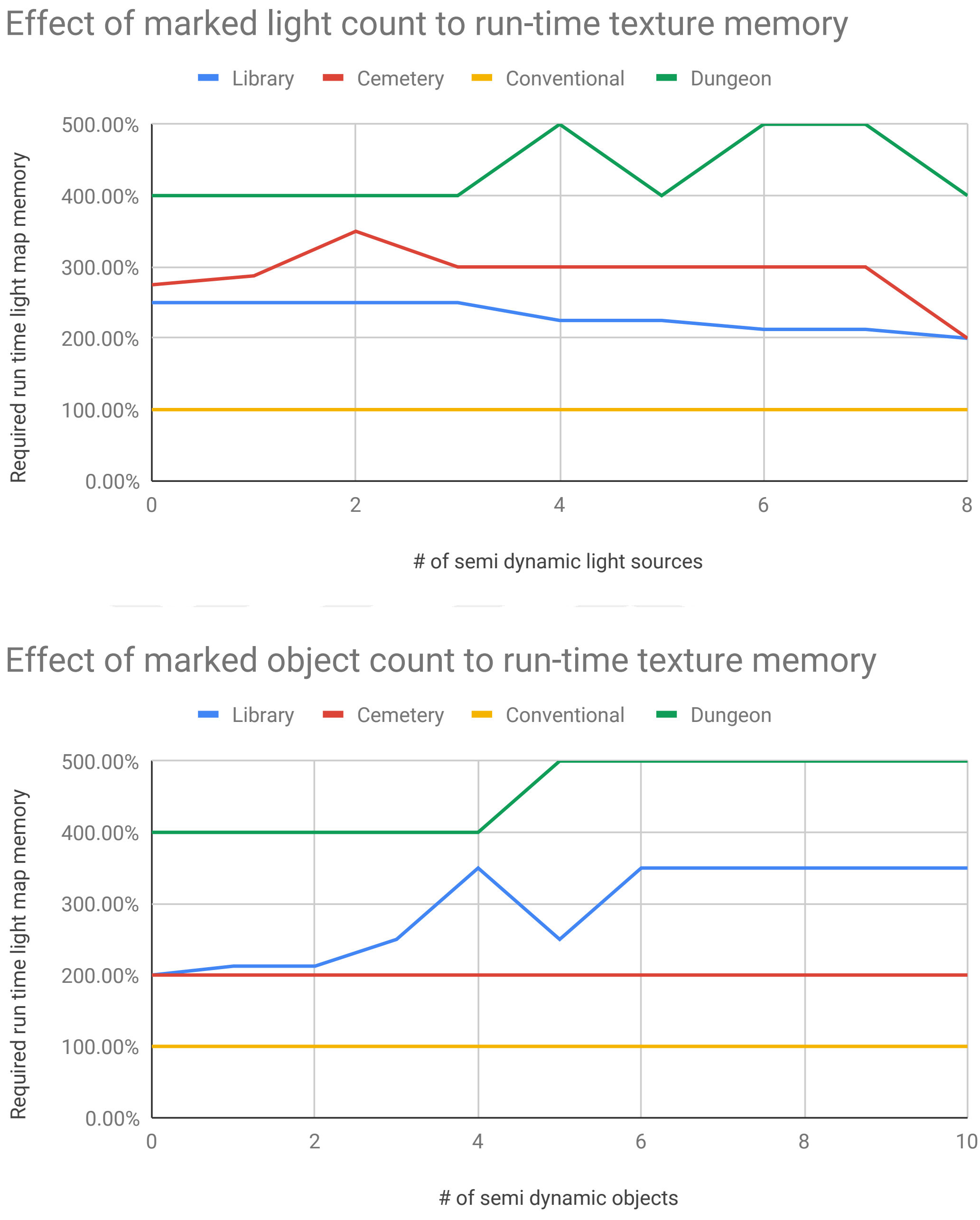


Figure 5.10: Semi dynamic objects usually increase required memory since each semi dynamic object uses additional texture space in the atlas. This does not always apply to semi dynamic light sources which can be more efficiently stored as explained in 5.9.

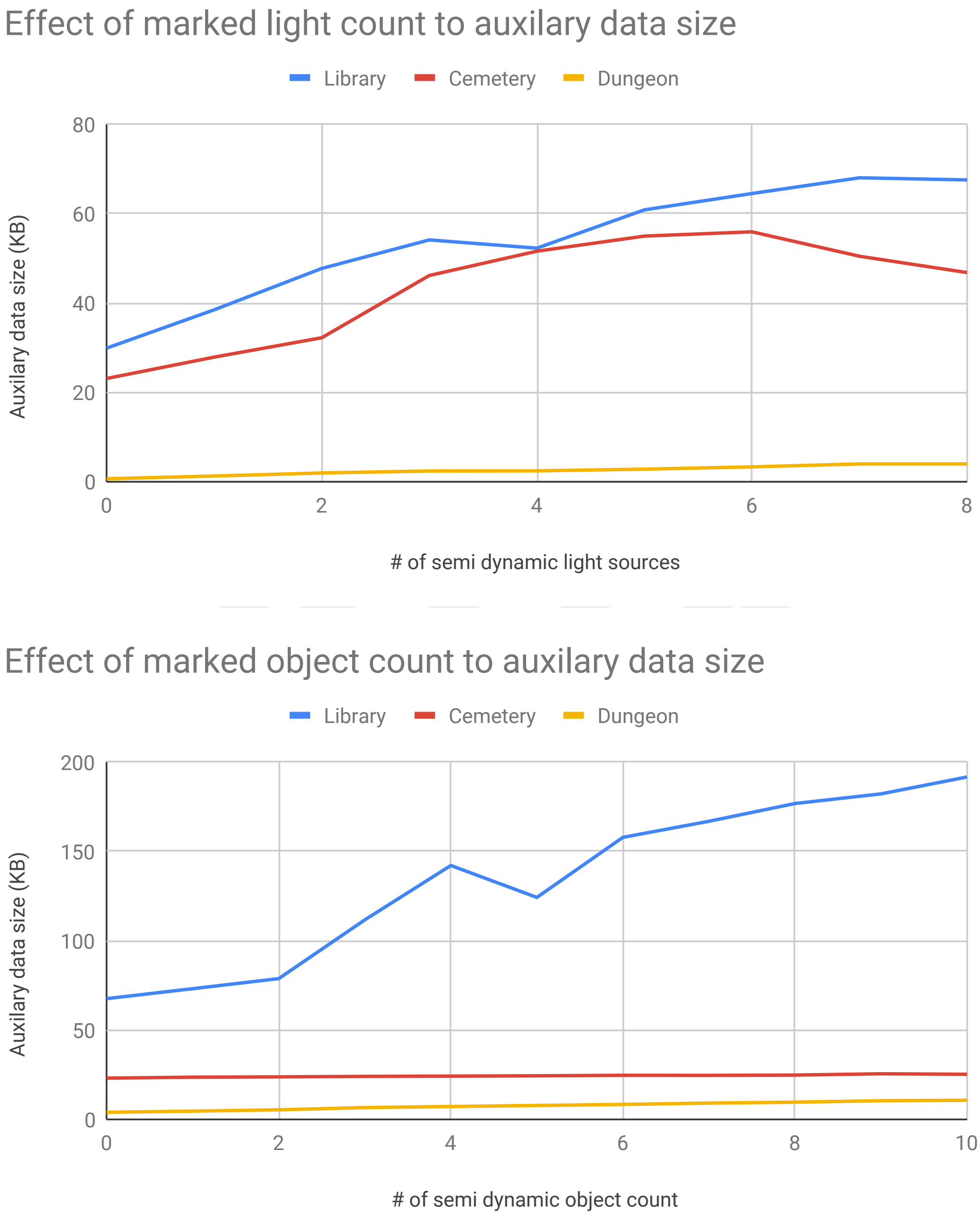


Figure 5.11: In *Cemetery* scene, semi dynamic objects only cast shadows to the ground. A single light patch is usually sufficient to store the shadows. Therefore, auxiliary data increases very slowly when semi dynamic object count increases in the graph below.

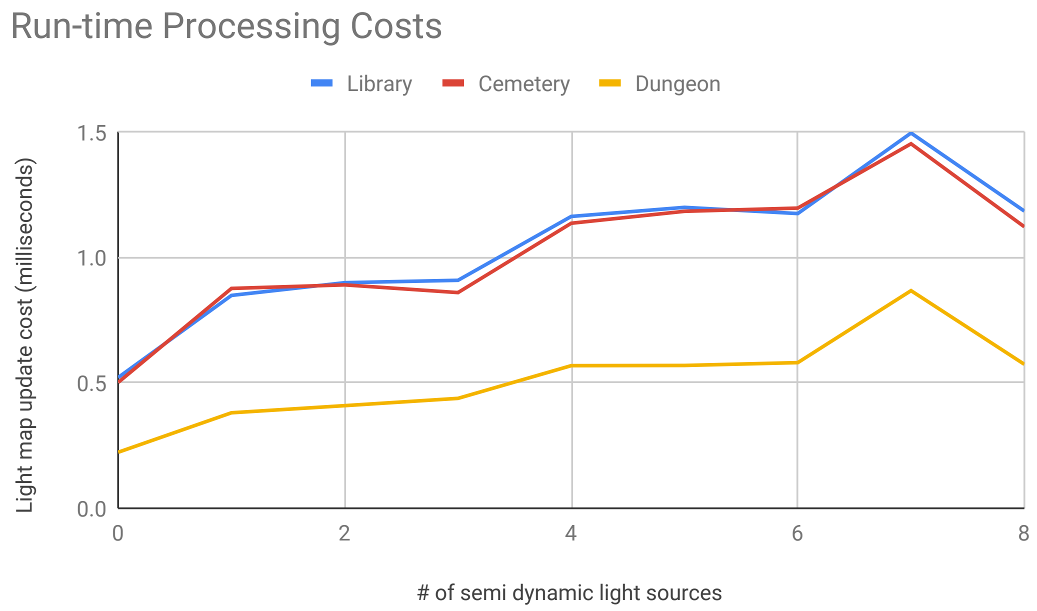


Figure 5.12: Run-time processing costs start as low as 0.25 milliseconds, but can be significant in complex scenes.

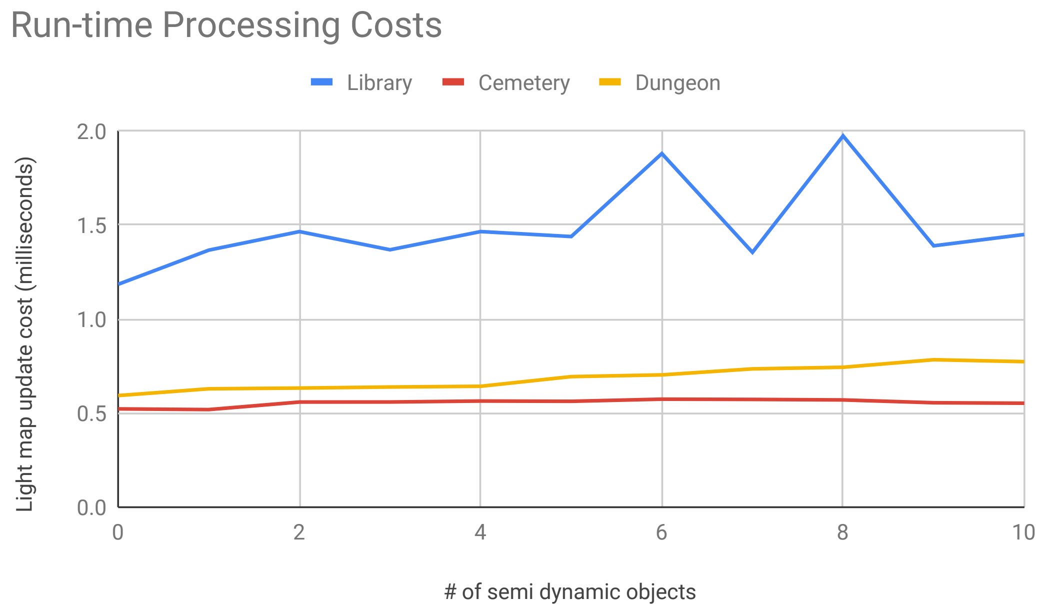


Figure 5.13: The cost of updating the light maps at run-time can increase to significant values. However, it is possible to keep a relatively smooth frame rate by executing the update operation in multiple frames or in separate threads.

CHAPTER 6

CONCLUSIONS

In this thesis, we proposed a new method to remove some of the restrictions of light mapping. By storing the contribution of light sources in separate textures, our method allows modifications to light color and intensity at run-time. Similarly, shadows of each object is detected and stored during offline stage. This makes it possible to show or hide baked objects without re-baking the light maps. Parts of the precomputed textures which do not contain any light is removed to reduce storage requirements. At run-time, these textures are merged and colored to create the light map that correctly represents the state of the scene. When there is a change in the state of the scene, this process is repeated to update the light map.

With aforementioned restrictions removed, light maps can be used in interactive applications with partly dynamic objects and light sources. This way, the run-time benefits of light maps can be retained while user is offered more ways to interact with the environment. For instance, a street light at an outdoor environment with night time lighting can be fully baked, and the player can shoot the light with a gun to turn it off. Without semi dynamic light maps, simulating such scenario would require either calculating the illumination at run-time or removing the user's ability to shoot the light. First option would mean less realistic results and higher system resources. Second option restricts the player and reduces the interactivity of the application.

A wide range of applications can benefit from using semi dynamic light maps. The cost of using semi dynamic light maps, however, is open to debate. As the desired interactivity of the scene increases, associated costs of generating and updating the light maps increases as well. The costs include precomputation times at offline stage, storage space requirements, computational costs at run-time and memory requirements at run-time.

Precomputation of light maps is usually the most time consuming step of conventional light mapping. With semi dynamic light maps, this cost quickly increases as the number of semi dynamic lights and objects increase. In our tests, precomputation of light maps took up to 25 times the time of precomputing conventional light maps. This value was observed in a scene where there are 8 semi dynamic light sources and 10 semi dynamic objects, however it can go much higher when more objects and light sources are marked as semi dynamic. On the other hand, this process is suitable for parallel execution, and significant speed ups can be achieved by using additional hardware.

Storage space requirements of semi dynamic light maps can be lower than conventional light maps as well as higher. Most graphics applications today use a large number of textures for diffuse and normal mapping of 3D environments. We believe that the increase in the storage space will be insignificant compared to the total size of the application. Storage size can also be kept low by limiting the number of semi dynamic light sources and objects.

Increased run-time memory is another important drawback of semi dynamic light maps. While a conventionally light mapped scene only requires the light map to be on memory, light patch atlases are also needed in semi dynamic light mapping.

We believe computational costs at run-time should be affordable in most applications since these costs appear only when an update to the light map is necessary. It is also possible to reduce this cost by taking advantage of multi-threading.

6.1 Limitations

6.1.1 Color Bleeding

Color bleeding can be explained by a surface's ability to partially absorb an incoming light and cause a change in it's color before reflecting it. This light later falls on another surface, causing an illumination. In this case, reflective object may appear to be illuminating surrounding objects. For instance, white light reflected from a red wall produces reddish colors on the floor in Figure 6.1.

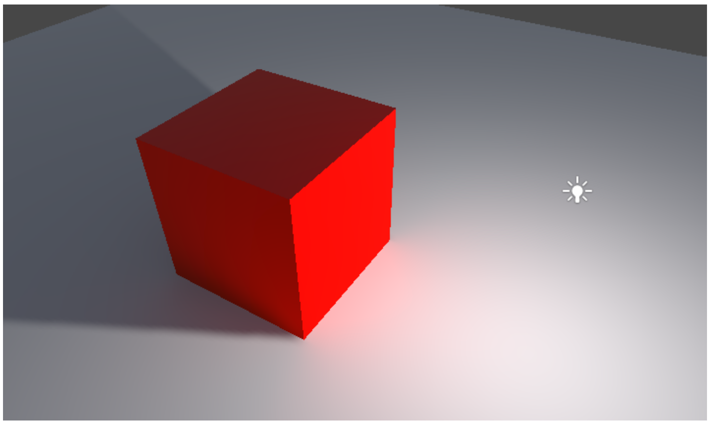


Figure 6.1: Red color "bleeds" from the cube to the floor.

One of the limitations of semi dynamic light mapping is the lack of support for color bleeding. Since each semi dynamic light source is stored only as intensity in one of the channels of a texture, a light source can only cast light of a single color. Color bleeding, however, causes a single light source to generate illumination of multiple colors. As long as semi dynamic objects and light source are not involved, it is still possible to use color bleeding with static light sources and static objects.

6.1.2 Indirect Illumination with Semi Dynamic Objects

Indirect illumination happens when a light illuminates a point after being reflected from at least one surface. Semi dynamic light mapping assumes that an object can only cause a reduction in the illumination. In other words, existence of an object cannot result in an increased illumination of another object. Therefore, semi dynamic light maps ignore the light reflected from semi dynamic objects. Semi dynamic objects can still *receive* indirect illumination. It is also possible to use indirect illumination with fully baked objects and semi dynamic light sources.

6.1.3 Static Light Sources with Semi Dynamic Objects

Another limitation of semi dynamic light mapping is the interaction between semi dynamic objects and static light sources. Shadows of objects are only stored for semi dynamic light sources. An object's affect to the base map is unknown at run-time. Therefore, semi dynamic objects should not be illuminated by static light sources. When such a case is encountered, one of the following options can be chosen:

- Mark each light source that illuminates the semi dynamic object as semi dynamic.
- Unmark the object so that it is no longer semi dynamic.
- Ignore the conflict. In this case, object will not cast any shadows to the light emitted from static light sources.

6.2 Future Work

Supporting indirect illumination caused by semi dynamic objects may be possible if each semi dynamic object is also treated as a light source. This will likely increase the precomputation, storage and run-time costs. Tests can be done to check if the costs are low enough to make the method useful in some applications.

Shadow maps store how much light was removed. For fully opaque objects, the amount of the removed light always equals to the amount of existing light. In this case, storing the amount of removed light is unnecessary. Using a single bit to store if a texel is shadowed or not is sufficient. This can decrease the storage space costs. However, computational costs at run-time might increase since additional bitwise operations will need to be performed.

REFERENCES

[1] T. Hachisuka, S. Ogaki, and H. W. Jensen, "Progressive photon mapping," in *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, (New York, NY, USA), pp. 130:1–130:8, ACM, 2008.

[2] K. O'Conor and J. Blommestein, "Lighting the open world of new york zero for prototype 2," in *ACM SIGGRAPH 2012 Talks*, SIGGRAPH '12, (New York, NY, USA), pp. 34:1–34:1, ACM, 2012.

[3] H. W. Jensen, "Global illumination using photon maps," in *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, (London, UK, UK), pp. 21–30, Springer-Verlag, 1996.

[4] P. S. Heckbert, "Adaptive radiosity textures for bidirectional ray tracing," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, (New York, NY, USA), pp. 145–154, ACM, 1990.

[5] W. Heidrich and H.-P. Seidel, "View-independent environment maps," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '98, (New York, NY, USA), pp. 39–ff., ACM, 1998.

[6] R. Ramamoorthi and P. Hanrahan, "An efficient representation for irradiance environment maps," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, (New York, NY, USA), pp. 497–500, ACM, 2001.

[7] O. Good and Z. Taylor, "Optimized photon tracing using spherical harmonic light maps," in *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, (New York, NY, USA), ACM, 2005.

[8] R. Bastos, M. Goslin, and H. Zhang, "Efficient radiosity rendering using textures and bicubic reconstruction," in *Proceedings of the 1997 Symposium on*

Interactive 3D Graphics, I3D ’97, (New York, NY, USA), pp. 71–ff., ACM, 1997.

[9] E. A. Kopylov, A. B. Khodulev, and V. L. Volevich, “V.volevich: The comparison of illumination maps technique in computer graphics software,” 06 1999.

[10] A. Iones, A. Krupkin, M. Sbert, and S. Zhukov, “Fast, realistic lighting for video games,” *IEEE Comput. Graph. Appl.*, vol. 23, pp. 54–64, May 2003.

[11] M. F. Cohen and D. P. Greenberg, “The hemi-cube: A radiosity solution for complex environments,” in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’85, (New York, NY, USA), pp. 31–40, ACM, 1985.

[12] H. Weghorst, G. Hooper, and D. P. Greenberg, “Improved computational methods for ray tracing,” *ACM Trans. Graph.*, vol. 3, pp. 52–69, Jan. 1984.

[13] K. Myszkowski and T. L. Kunii, “Texture mapping as an alternative for meshing during walkthrough animation,” in *Photorealistic Rendering Techniques* (G. Sakas, S. Müller, and P. Shirley, eds.), (Berlin, Heidelberg), pp. 389–400, Springer Berlin Heidelberg, 1995.

[14] W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel, “Illuminating micro geometry based on precomputed visibility,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’00, (New York, NY, USA), pp. 455–464, ACM Press/Addison-Wesley Publishing Co., 2000.

[15] W. Jarosz, H. W. Jensen, and C. Donner, “Advanced global illumination using photon mapping,” in *ACM SIGGRAPH 2008 Classes*, SIGGRAPH ’08, (New York, NY, USA), pp. 2:1–2:112, ACM, 2008.

[16] C. Knaus and M. Zwicker, “Progressive photon mapping: A probabilistic approach,” *ACM Trans. Graph.*, vol. 30, pp. 25:1–25:13, May 2011.

[17] A. Appel, “Some techniques for shading machine renderings of solids,” in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS ’68 (Spring), (New York, NY, USA), pp. 37–45, ACM, 1968.

[18] T. Whitted, “An improved illumination model for shaded display,” *Commun. ACM*, vol. 23, pp. 343–349, June 1980.

[19] D. Voorhies and J. Foran, “Reflection vector shading hardware,” in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’94, (New York, NY, USA), pp. 163–166, ACM, 1994.

[20] P. Haeberli and M. Segal, “Texture mapping as a fundamental drawing primitive,” 2011.

[21] P. S. Heckbert, “Survey of texture mapping,” *IEEE Comput. Graph. Appl.*, vol. 6, pp. 56–67, Nov. 1986.

[22] G. S. Miller and C. R. Hoffman, “Illumination and reflection maps,” in *ACM SIGGRAPH 1984 course notes*, SIGGRAPH ’84, 1984.

[23] N. Greene, “Applications of world projections,” in *Proceedings of Graphics Interface and Vision Interface ’86*, GI + VI 1986, (Toronto, Ontario, Canada), pp. 108–114, Canadian Information Processing Society, 1986.

[24] J. F. Blinn and M. E. Newell, “Texture and reflection in computer generated images,” *Commun. ACM*, vol. 19, pp. 542–547, Oct. 1976.

[25] B. Cabral, M. Olano, and P. Nemec, “Reflection space image based rendering,” in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’99, (New York, NY, USA), pp. 165–170, ACM Press/Addison-Wesley Publishing Co., 1999.

[26] Z. S. Hakura, J. M. Snyder, and J. E. Lengyel, “Parameterized environment maps,” in *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D ’01, (New York, NY, USA), pp. 203–208, ACM, 2001.

[27] A. Meyer and C. Loscos, “Real-time reflection on moving vehicles in urban environments,” in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST ’03, (New York, NY, USA), pp. 32–40, ACM, 2003.

[28] R. Habel and M. Wimmer, “Efficient irradiance normal mapping,” in *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D ’10, (New York, NY, USA), pp. 189–195, ACM, 2010.

[29] E. E. Catmull, “A subdivision algorithm for computer display of curved surfaces (cstd-74-006),” 1974.

[30] E. Catmull and A. R. Smith, “3dd transformations of images in scanline order,” in *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’80, (New York, NY, USA), pp. 279–285, ACM, 1980.

[31] R. L. Cook, “Shade trees,” *SIGGRAPH Comput. Graph.*, vol. 18, pp. 223–231, Jan. 1984.

[32] J. F. Blinn, “Simulation of wrinkled surfaces,” *SIGGRAPH Comput. Graph.*, vol. 12, pp. 286–292, Aug. 1978.

[33] E. Nakamae, K. Kaneda, T. Okamoto, and T. Nishita, “A lighting model aiming at drive simulators,” in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’90, (New York, NY, USA), pp. 395–404, ACM, 1990.

[34] P. Wrotek, A. Rice, and M. McGuire, “Real-time bump map deformations,” in *ACM SIGGRAPH 2004 Posters*, SIGGRAPH ’04, (New York, NY, USA), pp. 1–, ACM, 2004.

[35] I. Ernst, D. Jackél, H. Rüsseler, and O. Wittig, “Hardware supported bump mapping: A step towards ingber quality real-time rendering,” in *Workshop on Graphics Hardware*, 1995.

[36] M. Peercy, J. Airey, and B. Cabral, “Efficient bump mapping hardware,” in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’97, (New York, NY, USA), pp. 303–306, ACM Press/Addison-Wesley Publishing Co., 1997.

[37] I. Ernst, H. Rüsseler, H. Schulz, and O. Wittig, “Gouraud bump mapping,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS ’98, (New York, NY, USA), pp. 47–ff., ACM, 1998.

[38] M. Kligard, “A practical and robust bump-mapping technique for today’s gpus,” pp. 1–39, 10 2001.

[39] J. Arvo, “Backward ray tracing,” in *In ACM SIGGRAPH ’86 Course Notes - Developments in Ray Tracing*, pp. 259–263, 1986.

[40] D. Larsson, “Pre-computing lighting in games,” *SIGGRAPH 2010 courses*, 2010.

[41] M. F. Cohen, D. P. Greenberg, D. S. Immel, and P. J. Brock, “An efficient radiosity approach for realistic image synthesis,” *IEEE Computer Graphics and Applications*, vol. 6, pp. 26–35, March 1986.

[42] D. R. Baum, S. Mann, K. P. Smith, and J. M. Winget, “Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions,” in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’91, (New York, NY, USA), pp. 51–60, ACM, 1991.

[43] P. Heckbert, “Discontinuity meshing for radiosity,” pp. 203–226, 05 1992.

[44] Y. Li, G. Zhou, C. Li, X. Qiu, and Z. Wang, “Adaptive mesh subdivision for efficient light baking,” *The Visual Computer*, vol. 28, pp. 635–645, Jun 2012.

[45] H. Schäfer, J. Süßmuth, C. Denk, and M. Stamminger, “Memory efficient light baking,” *Computers & Graphics*, vol. 36, no. 3, pp. 193 – 200, 2012. Novel Applications of VR.

[46] C. Luksch, M. Wimmer, and M. Schwärzler, “Incrementally baked global illumination,” in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D ’19, (New York, NY, USA), pp. 4:1–4:10, ACM, 2019.

[47] C. Luksch, R. F. Tobler, R. Habel, M. Schwärzler, and M. Wimmer, “Fast lightmap computation with virtual polygon lights,” in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D ’13, (New York, NY, USA), pp. 87–94, ACM, 2013.

[48] N. Hoffman and K. Mitchell, “Real-time photorealistic terrain lighting,” in *In Proceedings of the 2001 Game Developers Conference*, Game Developers Conference 2001, 2001.

[49] J. Mitchell, G. McTaggart, and C. Green, “Shading in valve’s source engine,” in *ACM SIGGRAPH 2006 Courses*, SIGGRAPH ’06, (New York, NY, USA), pp. 129–142, ACM, 2006.

[50] D. Verbeiren and P. Lecluse, “Optimizing 3d applications for platforms based on intel® atom™ processor, white paper,” 2010.

[51] NVIDIA, “Improve batching using texture atlases, sdk white paper,” 2004.

TEZ İZİN FORMU / THESIS PERMISSION FORM

ENSTİTÜ / INSTITUTE

Fen Bilimleri Enstitüsü / Graduate School of Natural and Applied Sciences ☐

Sosyal Bilimler Enstitüsü / Graduate School of Social Sciences ☐

Uygulamalı Matematik Enstitüsü / Graduate School of Applied Mathematics ☐

Enformatik Enstitüsü / Graduate School of Informatics ☒

Deniz Bilimleri Enstitüsü / Graduate School of Marine Sciences ☐

YAZARIN / AUTHOR

Soyadı / Surname : Öztürk
Adı / Name : Bekir
Bölümü / Department : Çokluortam Bilişimi

TEZİN ADI / TITLE OF THE THESIS (**İngilizce** / English) : Yarı Dinamik Işık Haritaları / Semi Dynamic Light Maps

TEZİN TÜRÜ / DEGREE: **Yüksek Lisans** / Master ☒ **Doktora** / PhD ☐

1. **Tezin tamamı dünya çapında erişime açılacaktır. /** Release the entire work immediately for access worldwide. ☒
2. **Tez iki yıl süreyle erişime kapalı olacaktır.** / Secure the entire work for patent and/or proprietary purposes for a period of **two year**. * ☐
3. **Tez altı ay süreyle erişime kapalı olacaktır.** / Secure the entire work for period of **six months**. * ☐

* *Enstitü Yönetim Kurulu Kararının basılı kopyası tezle birlikte kütüphaneye teslim edilecektir.*
A copy of the Decision of the Institute Administrative Committee will be delivered to the library together with the printed thesis.

Yazarın imzası / Signature **Tarih** / Date 06.08.2019