CHALMERS

Light probe cloud generation for games

Master of Science Thesis in Interaction Design and Technologies

PETRAS SUKYS

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2014

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Light probe cloud generation for games

PETRAS SUKYS

© PETRAS SUKYS, June 2014.

Examiner: ULF ASSARSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: Light probe cloud generated in the Sponza scene. Several light probes are placed all around the scene to represent different illumination situations depending on the geometry nearby (see Section 4.3).

Department of Computer Science and Engineering
Göteborg, Sweden, June 2014

Abstract

The purpose of this thesis is to explore and research posibilities of automating the process of virtual light probe placement in a 3D scene. At the moment, probes are placed manually by artists, thus it is consuming time which could be used for other purposes. The primary focus of this research is to find and analyze possible algorithms for placing probes in arbitrary scene and to find an approach to use those probes in a real time application. Two different placement algorithms were analyzed and a sufficiently efficient probe cloud structure was implemented. The results from both placement algorithms were similar in terms of location. Furthermore, a data structure for irregularly placed light probes was implemented. The structure empowered fast interpolation of the cloud to approximate lighting in an arbitrary position. The resulting cloud was tested in a real-time renderer with several scenes. The lighting contribution from the cloud was sufficiently correct and the performance impact (disregarding precalculation) was negligible.

Sammanfattning

Syftet med detta examensarbete är att undersöka och utforska möjligheten att automatisera placeringen av virtuella ljussonder i en 3D-scen. I dagsläget måste dessa placeras manuellt av en 3D-konstnär, vilket är tidskrävande. Arbetets fokus ligger i att hitta och analysera algoritmer som kan placera sonder i en godtycklig scen samt att försöka hitta ett sätt att använda dessa sonder i en realtids-applikation. Två olika placeringsalgoritmer analyserades och en godtagbart effektiv sondmolnsstruktur implementerades. Resultaten från de båda placeringalgoritmerna var likartade med avseende på placeringen av sonderna. En datastruktur för oregelbundet placerade ljussonder var också implementerad. Strukturen möjliggjorde en snabb interpolering av molnet för att approximera ljussättningen i godtyckliga punkter. Molnet som genererades testades på flera olika scener i en traditionellt implementerad renderare. Ljussättningen från molnet bedömdes vara korrekt nog och påverkan på prestandan (ej medräknat förarbetet) var försumbar.

Acknowledgements

This master's thesis project was done at the Autodesk Gothenburg office, and I, the author, would like to thank everyone at the office for the great experience. I would like to thank Magnus Petersson, for making this thesis possible and Niklas Harryson for the presented opportunity. Furthermore, I am grateful Ragnar Cederlund, Anders Lindqvist, and Rasmus Bonnedal for the help and support during this thesis. Thanks to my examiner at Chalmers, Ulf Assarsson, for feedback and help on this thesis. I appreciate all the corrections and reviews by my colleagues and friends: Simon Iversson, Fotini Boyiatzi, Andra Kucinskaite, Ignas Gerulaitis. Lastly, I am grateful for feedback, input and opposition by my oponent Karl Schmidt.

Contents

1 Introduction

The current state of the computer graphics hardware and the emphasis on development of such hardware enables more complex computations to be performed while rendering computer graphics. I want to explore the possibility of employing this rapid increase of computational power to enhance the quality of computer rendered graphics. The higher quality of rendering, the more valuable a computer graphics product is, let it be a product for industry (automotive, industrial design, etc.), a rendered video (image) or interactive application such as a computer game.

A major part of rendered image quality is depending on correct lighting. This thesis work is focusing on researching and developing techniques used for lighting precalculation (baking) throughout the 3D scene (light probe cloud). This technique is relying on a compact storage of low frequency incoming lighting information using spherical harmonics. During runtime of real time applications this precalculated light information is accessed and used to add an ambient, low frequency (diffuse lighting) global illumination component to a rendered scene.

Autodesk Beast[1] is a tool developed by Autodesk for various computer graphics related lighting precomputations. It includes light probe cloud baking, but positions of probes have to be supplied, thus points in a scene have to be placed manually. The solution proposed in this work is an implementation of additional feature for the mentioned tool.

1.1 Purpose

The purpose of this thesis is to find any method or technique, which would automatically generate positions for light probes given any arbitrary scene, and a data structure for efficiently storing the light probe cloud with all the lighting information. Finally, implemention of both algorithm and data structure will be performed for automating currently the highly manual feature of the Beast.

[1] `http://gameware.autodesk.com/beast`

1.2 Problem

The problem is that currently available and used techniques for precalculating light information requires a lot of user (artist) input. Points are placed in the positions of light probes manually by the artist. It will be attempted to automate this process by analyzing the scene and trying to find the most appropriate positions and number of probes. The scene's geometry and light sources are the only factors influencing lighting environment in the scene.

In contrast to artist placed light probes, an automatically generated probe cloud will likely have irregular structure. Thus, an efficient data structure is required for both storing and accessing lighting information. The data structure, given arbitrary position in the scene, should enable the application developer to get approximate lighting information in that position.

In order to evaluate efficiency of the probe cloud generated and the data structure used, a renderer will be implemented. The renderer will be used to compare images rendered with probe cloud and without. An acceptable result would be object lit with sparsely placed light probe cloud while pertaining visual quality, not necessarily precise, but at least without any visible artifacts or discontinuities.

1.3 Limitations

Despite the advancements of computational hardware, there are several limitations. The underlying mathematics behind light probes and light capturing used in Beast relies on spherical harmonics. Spherical harmonics are used to approximate a two-dimensional function. Unfortunately, in order to be viable (more detailed explanation is in Chapter 2.3), only low order SHs can be used, thus only low frequency functions can be stored. In other words, high frequency functions stored in SH form will lose their high frequency components. Furthermore, the number of light probes should be kept as low as possible to save both the offline and the runtime memory. The lower number of light probes might cause environment with high frequency lighting changes to be under-sampled. The SH mathematics and the requirement for sparse probe cloud lead to the fact that my focus is only on low frequency lighting information, such as diffuse reflection and ambient occlusion.

Light capturing (or baking) is a precalculation process, which is performed before the actual application runtime, thus it need to be performed only once and is not required to be a real time process. Equivalently, light probe cloud generation is the same type of process, thus this work is not focusing on the performance of the algorithms in the precalculation step.

Nevertheless the previous liberation, the final use of the light probe cloud is to light a scene with global illumination in a real-time application. Thus, any kind of computations required by the algorithm or the data structure should be completed within the limits of reasonable time complexity. The performance decrease of the real-time application using mentioned structures have to be non-existent or negligible.

2 Background

This chapter includes background information about computer rendered graphics, including general problem, techniques currently available or in use and problems they pose. Furthermore, concepts behind virtual light probes for 3D scene and mathematical background of spherical harmonics will be presented.

2.1 Computer graphics and 3D rendering

A process of using computer for generating image from a defined 3D scene is called 3D rendering. The geometry of the scene is usually defined using mathematical structures such as points, planes, curves, spheres and triangles. Geometry usually contains material information: the data describing what kind of material the geometry is defining and how it should look. For example, reflective, transparent or diffuse geometry. There are various models for describing material properties, such as Lambert, Phong, Blinn-Phong, Cook-Torrance and others. In addition to the geometry and material information, scene contains various light sources which describes lighting environment in the scene. Light sources can be from simple infinitely small point lights to complex HDR environment lights. Finally, 3D scene has to have a defined camera and a projection information, or the computer program used for rendering has to have a default camera and a projection in order to render the scene to an image. The computer program performs calculations on which geometry elements are visible to which light sources, calculating shading, evaluating material properties and light interaction with it, calculating lighting and constructing the final image.

2.1.1 Rendering equation

In theory, the image generation is a matter of solving a mathematical equation which defines the light which reaches camera (or observer). There are several rendering equations described in the history of computer graphics. Here are probably the earliest defined equations:

$$I(x, x') = g(x, x')\Big[\epsilon(x, x') + \int_S \rho(x, x', x'')I(x', x'')dx''\Big] \tag{1}$$

An equation proposed by James T. Kajiya in 1986 [1]. Here, $I(x, x')$ is the intensity of light from a point x towards a point x'; $g(x, x')$ is the geometrical term determining point's x visibility from the point x';$\epsilon(x, x')$ is emitted light intensity from the point x to the point x'; $\rho(x, x', x'')$ is intensity of light received from the point x'' in surface point at x and reflected towards the point x'; S is a hemisphere above the surface at the point x.

$$I_{t,i}(f) = \epsilon_i(f) + \sum_{j=1}^{N} \sum_{d=1}^{D} \rho_i^{''}(f, d) I_{t,j}(d^{-1}) \cos\theta_d \omega_t HID(i, j, d) \tag{2}$$

This equation was proposed by David Immel et al. [2] the same year as previously mentioned work by J. T. Kajiya, but both works were independent. In Equation 2, $I_{t,i}(f)$ is an outgoing intensity from a i-th surface patch in a direction f; $\epsilon_i(f)$ is an emitted light intensity from the i-th patch in the direction f;N is a total number of patches in the scene, and D is a total number of directions per patch; $\rho_i^{''}(f, d)$ is a bidirectional reflectance of the i-th surface, or, in other words, the intensity of reflected light falling from the direction d onto the i-th surface and reflected in the direction f); $\cos\theta_d$ and ω is essentially a geometrical term for the hemisphere direction d; finally, $HID(i, j, d)$ is a simple visibility function, which equals 1 if the i-th patch sees the j-th patch in the direction d and 0 otherwise.

Equation 2 is an approximation by discretizing an integral based equation very similar to Equation 1. Thus, the solution of rendering equation in general is an integral over a hemisphere above the surface point. Unfortunately, solving such exact equations is a very time consuming process, which makes it infeasible to use with real-time applications. Real time applications usually employ faster, although not as physically correct, solution - rasterization.

2.1.2 Rasterization

Rasterization is a process when an image described in a vector form is converted to a bitmap (raster) image. In 3D graphics,it is a process of, first, transforming a 3D scene into a screen space (which is defined by camera). Then each primitive is subdivided into fragments, where each fragment is representing one pixel in the final image. Finally, the fragment is shaded according

to material properties of the geometry, light sources and visibility and resulting color is assigned to the pixel in the image (or screen).

Rasterization is very common in real time applications. Due to vector based scene description, per triangle rasterization and per fragment (pixel) shading, process can be highly parallelized. The parallelized processes are accelerated using specialized graphics processing units (GPU). Unfortunately, parallelized process means localized, too. Thus, computation of lighting and shading is usually approximated and localized. In other words, the rendering equation is solved with approximations and discretization, causing the final image to lack fidelity, real world phenomena or physically correct appearance.

2.1.3 Ray tracing

Raytracing is another technique used to render a 3D scene into an image. It is based on the physical process of photons traveling from light sources, reflecting and refracting from various surfaces and reaching the camera (or viewer's eye).

Real life light sources emit vast amount of photons. Most of them get absorbed on impact with surfaces, scattered due to atmosphere or just bounce off outside the scene. Only a fraction of the total emitted photons reach the eye. Simulation of photons leaving light sources and tracing which surfaces they hit, if and when they reach the eye is a very inefficient process. Such process yield very noisy results or the time required to acquire usable results is not acceptable. On the other hand, tracing photons backwards is a better solution.

For each pixel (assuming no super sampling in use) in the final image, a ray (which is traditionally called primary) is casted towards the scene. From a point where the ray intersects geometry, additional rays (shadow rays) are casted towards light sources in order to evaluate if the surface point is lit up. Furthermore, its color is evaluated according to surface's material information. Finally, extra rays (secondary rays) are casted towards specific directions from the intersection point. Usually, the directions are determined by a BRDF - bidirectional reflectance distribution function, a function determining how light is reflected. Secondary and forthcoming rays proceed to trace the photon

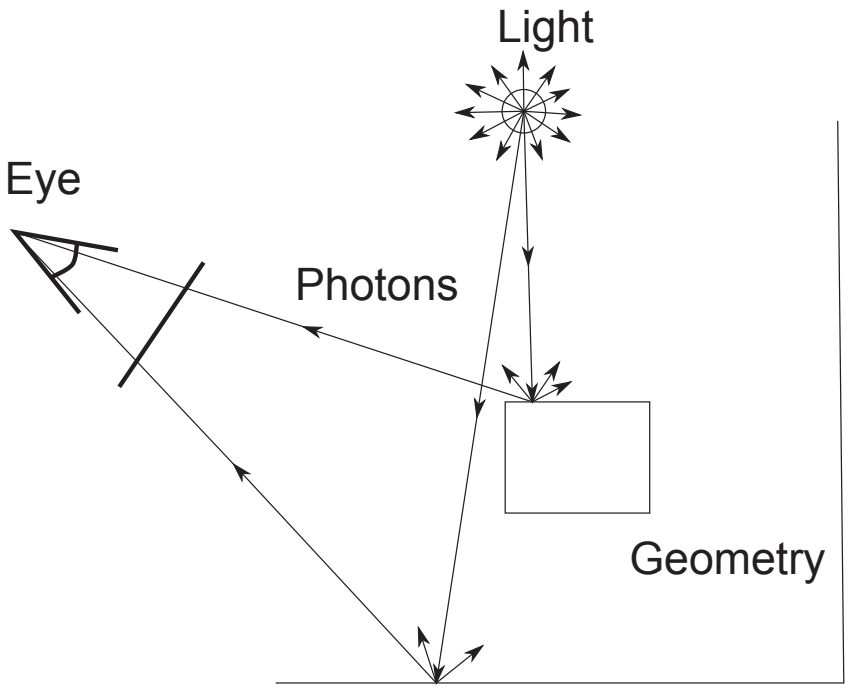


Figure 1: Photons traced from the light source

while evaluating shading and lighting until a certain termination criteria is met. Such criteria could be an amount of rays traced, depth of the photon path, or insufficiently high color contribution to the final pixel.

This way it is guaranteed that a traced photon reached the camera and carries significant contribution towards the final image. Although it is still approximated and discretized process, it is physically more correct and is used to achieve realistic images. Unfortunately, a lot of rays have to be casted to generate final image, therefore, ray-tracing has not been used extensively in real-time applications. There are a lot of techniques developed to further refine the ray tracing algorithm, such as path tracing, Monte Carlo tracing, final gather, and irradiance caching.

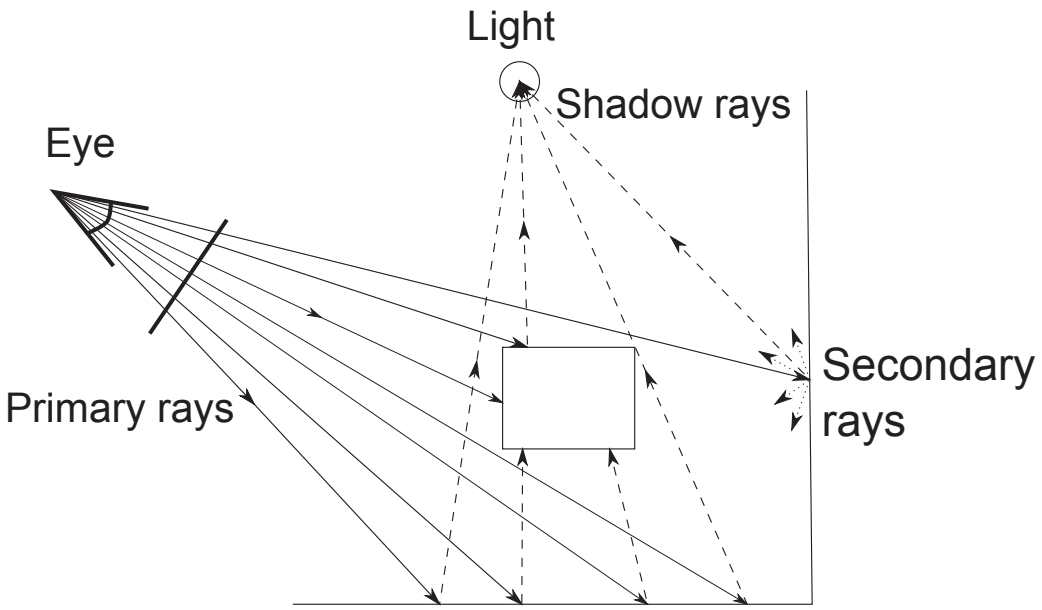
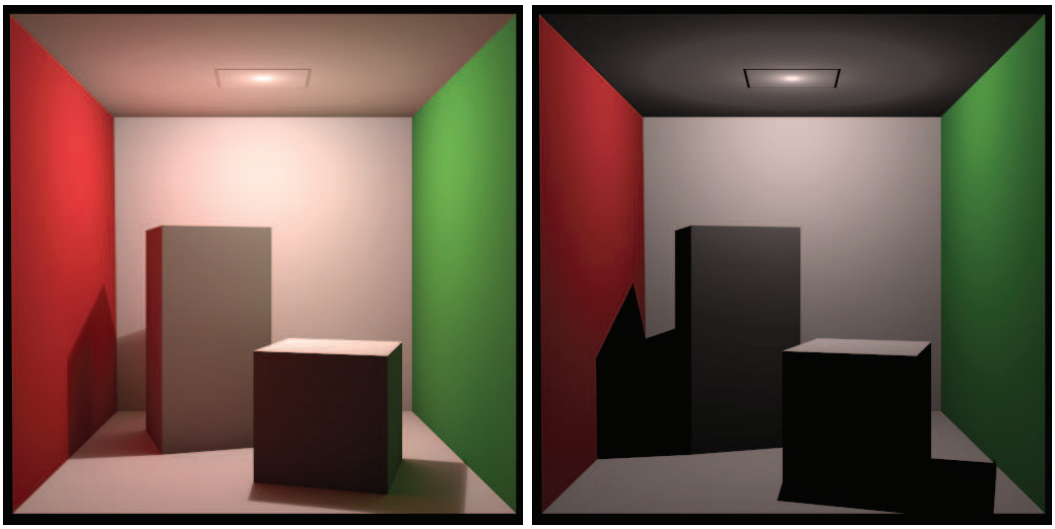


Figure 2: Rays traced from the camera

2.2 Global illumination

The advantage of ray-tracing is that, by simulating how the light interacts with the scene, it is possible to create more realistic lighting effects and better looking materials. These effects are very difficult or even nearly impossible to achieve using vector rasterization techniques. Techniques which allow to approximate such effects and implement them to a certain degree usually are inferior in quality. In Figure 3 there is depicted side-by-side two images. One has been ray-traced. It has indirectly illuminated surfaces, ambient occlusion, soft edged shadows, . The other image was rendered by rasterizing. It is clearly visible that rasterized image looks less natural and physically accurate.

Although scene in Figure 3 has very basic materials (diffuse), global illumination effects are very apparent. The whole scene has more light, because surfaces are lit not only by direct illumination from light source, but from rebounded light from other surfaces, too. In Figure 3a, the ceiling is brighter,



(a) (b)

Figure 3: Cornell box scene: a) pathtraced b) rasterized with high quality shadow maps.

because it is illuminated by all the other surfaces. While in Figure 3b it is only illuminated by single light source extremely close to it. In addition, globally illuminated scene has partially illuminated shadowed areas. Finally, surfaces facing brightly lit and colored green and red walls are illuminated by the color these walls are reflecting.

2.3 Spherical harmonics

Despite the fact that most of the global illumination effects are hard to achieve, some of them are being implemented in real-time applications with some approximation. For example: ambient occlusion emulated in screen space, a glossy surface reflections created with low resolution environment maps, volumetric light shafts rendered with a radial blur pass for the light sources, or precomputed and baked lighting information directly into textures. This chapter explains another technique used to simulate ambient global illumination effects - spherical harmonics.

2.3.1 Mathematical background

Basis functions and approximation

There are various methods how to approximate mathematical one dimensional functions. Most common ways are expansion of a function into Taylor [3] or Fourier [4] series. Simply speaking, the function at a certain point meeting criteria, such as continuity, definition domain, and limits, may be represented as an infinite sum of scaled certain functions (Equation 3).

$$\begin{aligned} f(x) &= \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = \\ &= f(x) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + ... \end{aligned} \tag{3}$$

Of course, numerically evaluate an infinite sum is impossible, thus, if such sum has no analytical solution, the resulting sum is usually an approximation with a finite number of terms. For example, in Equation 4 (proof of this equation is cumbersome and outside of the scope of this paper) and Equation 5 is displayed that $sin(x)$ function is fairly approximated just with several Taylor series terms ($sin(pi/2)$ rounded to 10 digits equals[2] to 0.70710678112).

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + ... \tag{4}$$

$$\sin\left(\frac{\pi}{4}\right) \approx \frac{\pi}{4} - \frac{\left(\frac{\pi}{4}\right)^3}{3!} + \frac{\left(\frac{\pi}{4}\right)^5}{5!} - \frac{\left(\frac{\pi}{4}\right)^7}{7!} + \frac{\left(\frac{\pi}{4}\right)^9}{9!} = 0.7071067829 \tag{5}$$

Taylor series are represented by a infinite polynomial. Each argument raised to power $(x, x^2, x^3, ...)$ in series are called basis functions. Therefore, evaluated factorials with sign $(1, -1/3!, 1/5!, ...)$ are coefficients or scale factors for the mentioned basis functions.

2.3.2 Definition

Both Taylor and Fourier series are an example of function approximation using basis functions in one argument (1D) function space. Thus, spherical

[2] According to Wolfram Alpha, `https://www.wolframalpha.com/input/?i=sin%28pi%2F4%29`

harmonics could be considered as a 2D counterpart of fore mentioned approximations. With a set of infinite series of spherical harmonics, it is possible to represent any spherical[3] function. Spherical harmonics are used to compute electron configurations, represent gravitational and magnetic fields, other parts of physics, and, of course, computer graphics. The specific set of spherical harmonics forms an orthogonal system. That set of spherical harmonics is called Laplace's spherical harmonics and usually denoted Y_ℓ^m. Orthogonality of spherical harmonics is a very valuable property in computer graphics, which will be discussed later, in Section 2.3.3.

Spherical harmonics as basis function

From mathematical perspective, spherical harmonics are specific case of solutions to Laplace's equation [5]. Spherical harmonics are based on family of polynomials called *Associated Legendre polynomials*[4]. These polynomials are defined by two integer numbers, which in turn defines the spherical harmonics.

$$Y_l^m(\theta,\phi) = \begin{cases} \sqrt{2}K_l^m \cos(m\phi)P_l^m(\cos\theta), & m > 0 \\ \sqrt{2}K_l^m \cos(-m\phi)P_l^{-m}(\cos\theta), & m < 0 \\ K_l^0 P_l^0(\cos\theta), & m = 0 \end{cases} \tag{6}$$

$$K_l^m = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}} \tag{7}$$

Equation 6 defines real spherical harmonics. Here, l and m are integers defining spherical harmonic, where $l > 0$ and $-l \leq m \leq l$; K_l^m is a normalization factor; $P_l^m(x)$ is the fore mentioned associated Legendre polynomial.

The defining parameter l is usually called "band", and the value of this parameter determines the order of spherical harmonic. The higher the order, the more accurate representation of the approximated function. In Figure 4

[3]Since spherical harmonics are defined on a surface of sphere, the approximated function has to have the same definition domain

[4]Legendre polynomials are defined on imaginary numbers. Associated Legendre polynomials are defined only on real numbers

is depicted spherical harmonics of the first 3 bands. The red color represents positive values and the green color represents negative values of the spherical harmonic functions.

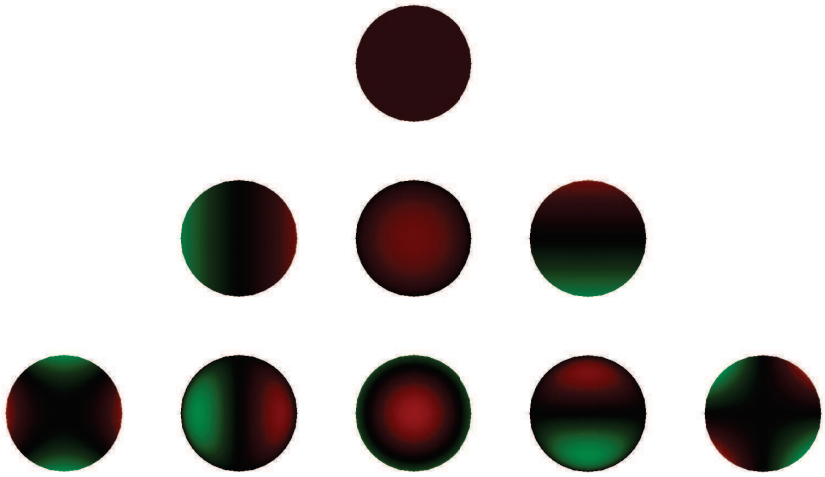


Figure 4: First three bands of spherical harmonics.

Functions approximated with spherical harmonic basis are expressed in the same way as with Fourier or Taylor series. An actual representation would be an infinite sum of spherical harmonics multiplied by coefficients, thus approximated version is only a sum of finite number of terms.

$$f(\theta,\phi) \approx \tilde{f}(\theta,\phi) = \sum_{i=0}^{l} \sum_{j=-i}^{i} \tilde{f}_{i,j} Y_i^j(\theta,\phi) \tag{8}$$

In Equation 8, $f(\theta,\phi)$ is a function being approximated; $\tilde{f}(\theta,\phi)$ is an approximated function; $\tilde{f}_{i,j}$ is the coefficient for the j-th spherical harmonic function of the i-th band; finally, $Y_i^j(\theta,\phi)$ is the spherical harmonic basis function of order i and number j. For the sake of readability and ease, coefficients could be

indexed altogether, without the need for separate indexes i and j (Equation 9). The number of total coefficients equals to the squared number of bands in use . Thus, the upper limit for the sum in Equation 9 equals to $(l+1)^2-1$ so that the sum iterates over all coefficients with the starting index of 0.

$$\begin{aligned} f(\theta,\phi) \approx \tilde{f}(\theta,\phi) = \sum_{i=0}^{(l+1)^2-1} \tilde{f}_i Y_i(\theta,\phi) \\ Y_i = Y_l^m, \quad \text{where } i = l(l+1)+m; \end{aligned} \tag{9}$$

2.3.3 Properties

Two of the first computer graphics applications of spherical harmonics were presented by Jan Kautz, Peter-Pike Sloan, and John Snyder in 2002 [6] [7]. The main two reasons why spherical harmonics have a very high use in computer graphics are compactness and orthogonality.

Compactness

First of all, function represented using spherical harmonics (as with any other basis) is just a set of coefficients. The basis functions are static, thus can be stored separately. Since spherical harmonics represent spherical functions, they enable to define complex spherical functions with very few coefficients. For example, an approximation with only 2 bands defines a function with only 4 coefficients. Although such approximated function loses its high frequency fidelity, for certain applications it is not necessary.

Orthogonality

Another valuable property of the spherical harmonics is orthogonality. Orthogonality property comes from orthogonality of underlying Legendre polynomials. Orthogonal polynomials are defined by this equation:

$$\int_{min}^{max} F_m(x)F_n(x)dx = \begin{cases} 0, & for\, n \neq m \\ c, & for\, n = m \end{cases} \tag{10}$$

$$\int_S A(\theta,\phi)B(\theta,\phi)d\theta d\phi = \sum_{i=0}^{(l+1)^2-1} a_i b_i \tag{11}$$

Orthogonality of Legendre polynomials makes spherical harmonics basis orthogonal, too. Orthogonality property leads to the simplified calculation of product integration. Computationally intensive integration over the surface of a sphere is reduced to several multiplication and addition operations (Equation 11). Those operations can be performed extremely fast on modern graphic processing units.

Linear interpolation

Finally, spherical harmonic functions are easily linearly interpolated. I. e. given two different spherical harmonic functions, it is possible to smoothly interpolate and generate intermediate functions. The interpolation is done by linearly interpolating each and all coefficients of the spherical harmonic representation of given functions. All the resulting functions will be smoothly and gradually changing from one spherical harmonic function to another. An example of linear interpolation is depicted in Figure 5.

Figure 5: Linear interpolation of mainly purple and yellow-red spherical harmonic functions.

2.3.4 Use in computer graphics

Most of the uses of spherical harmonics in computer graphics comes from the properties described above. The most usable property is the compactness.

World-space ambient occlusion

Given any point in the 3D scene, it is possible to determine how much that point is occluded by neighboring geometry. Highly occluded scene point will be darker (less illuminated) than non-occluded one [8]. Unfortunately, it is quite difficult to compute real time occlusion, since it requires search of the

scene for possible occluders. This is due to the need of casting rays to detect nearby geometry. Using spherical harmonics, it is possible to precalculate and store an approximated occlusion function for any given point. Therefore, it is possible to store the ambient occlusion function with only several floating point coefficients. Although this function would be very approximate and would lack detailed shadows, but that is sufficient since the ambient occlusion is regarded as soft, diffuse phenomena.

Material information

Another use for the spherical harmonics is to describe a BRDF [9] of a surface. Although, the low-frequency nature of spherical harmonics disallows high frequency BRDFs, it is very useful to define diffuse surfaces. A function approximated with spherical harmonics for each color channel is an approximated function of BRDF for a diffuse surface. As in previous example, each spherical harmonic function is represented with just a few coefficients. Usually spherical harmonics up to second order are used, which means only 4 coefficients to describe a single color channel and 12 coefficients to describe a full color function.

Compact light representation and integration over hemisphere

In addition to representing surface BRDF, spherical harmonics can be used to represent light source. To be more specific, it can be sued to define infinitely far away light source, such as an environment map or a sky dome. Environment maps represented as spherical harmonics would lack the fidelity, but again, spherical harmonics take up considerably less memory than textures used for texture based cube or spherical maps. Furthermore, the orthogonality property of spherical harmonics enables to calculate environment dependent lighting. Traditionally in real time applications, color of a surface fragment equals to the color of the material multiplied by color of the light. Referring to the rendering equation (Equation 1 and Equation 2), a perfect solution would be to integrate over all the directions over the hemisphere and then multiply color of the light from that direction with color of the material looking from that direction. Having in mind that both surface BRDF and (infinitely far away) light source can be represented by spherical harmonics, the approximated solution of rendering equation is just integral of product of the BRDF

and the light's spherical harmonics over the hemisphere. Which, due to orthogonality, is efficient operation of vector multiplication of the representation coefficients.

Radiance transfer

Finally, indirect illumination can be precomputed and premultiplied for all the samples, so that during runtime surfaces would be lit not only from directly evaluating light source but as well from secondary light bounces from other surfaces.

All these effects can be combined in one equation which resembles the previously mentioned lighting equations (Equation 2).

$$\begin{aligned} c &= \int_S AO(\theta,\phi)BRDF(\theta,\phi)(DL(\theta,\phi)+IL(\theta,\phi))d\theta d\phi = \\ &= \sum_{i=0}^{(l+1)^2-1} ao_i \cdot brdf_i \cdot (dl_i + il_i) \end{aligned} \tag{12}$$

In Equation 12, AO is ambient occlusion function; $BRDF$ is bidirectional reflectance function; DL is direct lighting function (such as environment light); IL is indirect lighting function.

2.4 Light probes

Light probing is a technique used to capture light environment in real life for later use in computer graphics. It is done by taking several digital photos of a sphere (very smooth, usually made of silver) in an environment to be captured. These images usually have different exposure times to capture the environment in HDR (High Dynamic Range). In case of need for a diffuse lighting information, smooth but non-reflective sphere is used such as plastic. After those images were captured and processed they can be used in a 3D rendering application to simulate lighting from that specific real world scene. Of course, high resolution and dynamic range image might not be necessary and use up to much of memory. In that case, lower resolution image or other compressed way of storing information may be used. Especially for the diffuse light probes: when the light has only low frequency changes throughout the environment.

The same technique is being applied to virtual 3D scenes. Using "virtual light probes", light information is captured at a point in a 3D scene and then, in the same way as captured from real life scene, used to light the objects in the very same virtual scene. Light for the static part of the scene has to be captured only once. It could be done before the application runtime and can include time consuming calculations such as mentioned in the Section 2.2. This way it is possible to have a very realistic lighting information about the scene with a very small computational overhead. Unfortunately, light probes in the scene has to be placed manually and the implementation of the runtime application still has to determine if and which virtual light probes to use for object lighting.

3 Analysis

The aim of this work is to investigate possible ways for automating the creation of light probe clouds. It was split into two parts - placement and data structure. The first part is about finding the right positions where to place virtual light probes in order to capture the light environment as correctly as possible. The second part is to find or create a data structure which would help efficiently use the data in the generated light probe cloud.

Due to various reasons, the two parts were developed in parallel. For example, to evaluate the generated light probe cloud it was necessary to have an algorithm how to access the data in it. Furthermore, in order to implement certain data structure aspects, the generated probe cloud had to meet certain criteria.

3.1 Placement

The sole purpose of this work is to automate or ease the work of lighting artists. Currently, artists examines a scene "by eye" and determines where the light probes need to be placed to represent the whole scene correctly. It is very manual work and could be tiresome and lengthy. In order for this automation to be useful, it should work on any arbitrary scene of any size. Thus, no other information except the scene's geometry, material and lighting is supplied. For example, collision information, passable and non-passable areas, camera transformation limits, etc. Such extra information would be application-dependant and not useful in a universal case. The work was done using only the geometry, material and lighting data.

During the early research, no solution for the analytical light probe placement was found. Thus, the first approach to automate this process was inspired by already in-use techniques in other parts of computer graphics. The second approach is based on optimization theory and solution to similar problem for environment maps by M. G. Chajdas *et al.* [10]

The primary requirement for the resulting light probe cloud is "good looking". In other words, the generated cloud should not make the scene look unnatural, cause lighting artifacts. Furthermore, the algorithm should place

light probes in a sparse way. A dense light probe cloud is not suitable, because then it could be replaced with a dense uniform grid. Unfortunately, such an approach would create a lot of light probes. Although each light probe could be compressed to a very compact spherical harmonic representation, a dense enough grid would still take up too much both offline storage space and runtime memory.

3.1.1 Spawn and merge

Adaptive techniques were known for a long time and are used in a wide range of fields. Adaptive multisampling or antialiasing (adaptive filtering) are the most common ones used in computer graphics. The idea behind the approach presented in this section is essentially an adaptive and iterative process of new light probe placement and removal.

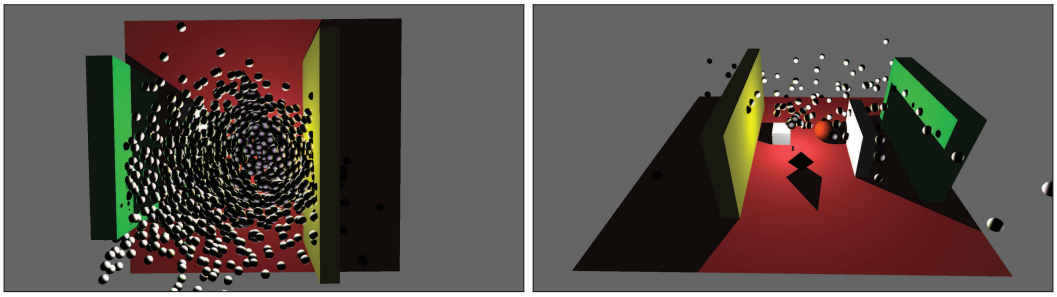
Definition

Given a scene (geometry, material, light sources), a working volume should be defined first. If there is no need for a custom working volume (such as specific area in the scene), the whole scene is used. In general, the working volume is always a bounding box. The working volume is filled with a uniform grid of light probes. The density of the initial grid may be variable from scene to scene, as one of the parameters for the whole process. The positions of light probes should be used to capture environment lighting. Then, similar probes are merged and new probes are created. If two light probes are fairly close to each other and have fairly similar representation of the captured lighting environment, each of them are adding the same information to the grid and can be merged to form a single light probe at midpoint between them. On the other hand, if two probes are not too much apart and capture significantly different lighting information, it is a sign that the lighting environment continues to vary in between them. To capture those details, an extra light probe is created at midpoint in between the original two probes. Then, the process should be repeated until any of the termination criteria is met: a) no changes have been made in last iteration; b) maximum number of iterations reached; c) maximum number of light probes reached; or any other similar to these.

Downfalls and shortcomings

During the implementation of the algorithm several downfalls were encountered, and they are valuable to discuss more thoroughly.

One of the first problems that occurred was that light probes were placed in a certain way around the light sources. Due to the fact that new light probes where inserted based on differences between two currently existing light probes, usually they would form a pattern around the light sources. This happens because the light sources are major factors influencing the lighting environment. Furthermore, due to this "clustering" around light sources, areas without or with very low light intensity are usually not sampled enough. Of course, increasing contrast or distance thresholds would tackle this problem, but: 1) it would cause undersampling or oversampling near the light sources; 2) the positions of light sources are already known, there is no need to execute an algorithm for that.



(a) (b)

Figure 6: Light probe clustering around the light source.

Another shortcoming of the algorithm was the size of the initial grid and quantity of intermediate probe cloud. This method creates probes between other probes. In the worst case scenario, another light probe is inserted between each pair of current probes. Given n light probes in the current iteration, the next iteration could have $O(n^2)$ probes after the "spawn" phase of the iteration. Thus, the effective density of the initial grid has to be limited.

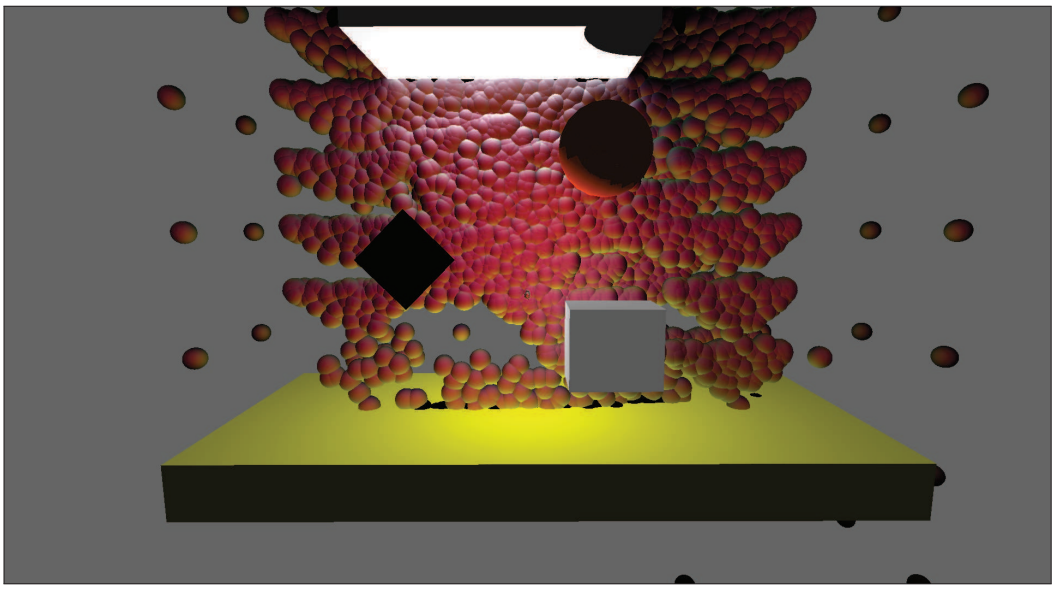


Figure 7: The pawn and merge algorithm creates a lot of light probe during the spawn phase. The bottom view of the simple scene.

3.1.2 Gradient ascent

During the background research, one paper which described a very similar problem was found. The paper written by M. G. Chajdas, A. Weis, and R.Westermann [10] and presented in SIGRAD 2011. The backbone of the idea that the authors have presented is using an irradiance gradient field to determine where the light changes the most. The next sections describe the algorithm implemented during this work, similarities and differences from the algorithm Chajdas *et al.*, presented, and problems that have occurred during the process.

Definition

The algorithm Chadjas *et al.* presented is completed in the following steps. First, an irradiance gradient field is precomputed for the whole scene. A uniform grid is generated in the scene, and at every grid vertex a HDR spherical environment map is captured. A gradient vector is calculated by summing up all the directions from the vertex and multiplying by the luminance of the respective fragment from the environment map captured at that vertex. After this grid of irradiance gradient vectors is constructed, the algorithm proceeds

with the common gradient ascent procedure. Particles are created at every vertex of the grid and are traced using classic algorithms for gradient ascent. After the gradient ascent algorithm traces several particles to one point, all those particles are marked as "potential probes". These potential probes are furthermore analyzed and compared to determine if they are similar and can be merged. In the algorithm the authors use quite an extensive set of techniques to distinguish between different images. It includes a histogram of the spherical environment map comparison and FFT-based[5] distance metric. The latter was used to determine the difference in color and brightness while the former is used to determine the difference in luminance distribution across the environment map.

Adaptations and modifications

To begin with, an implementation of the gradient ascent algorithm implemented is slightly different from the one described in the paper. Since this work is centered around light probes which are represented in spherical harmonic functions placement, the spherical environment maps were not used. Instead of the spherical environment maps, the irradiance at the uniform grid vertices is captured and stored as a spherical harmonic function. In short, light probes are placed at vertices of the uniform grid. Afterwards, the irradiance gradient is calculated in each of those points by sampling (thorough explanation may be found later, in Chapter 3.1.3) the spherical harmonic functions. Then, the position of the light probes is adjusted by gradient ascent algorithm. Continuing to next iteration, light is captured and stored at the new positions, the irradiance gradient field is resampled and recalculated, and continuing ascension while termination conditions are met. Which are either number of iterations exceeded or gradient ascension step is lower than threshold.

Downfalls and shortcomings

Due to the fact that irradiance gradients are calculated from irradiance only, the gradient ascent algorithm traces most of the light probes towards the light source. As described previously, in Section 3.1.1 and it was mentioned in the document by Chajdas *et al.*. This kind of convergence causes additional problems with this approach, because light probes are placed directly at the

[5]FFT - Fast Fourier Transformation

position of point light sources, thus making it impossible to capture direct illumination.

Furthermore, the gradient ascent is performed while calculating gradients entirely from irradiance, which means that geometry is not influencing the tracing of the light probes. This leads to light probes being traced towards geometry (for example, a brightly lit white surface). With the next gradient ascent step the probe will be positioned behind the surface of the geometry. Since the light sources behind the surface are not visible, the light probe then would be in a completely dark environment. The gradient of such a light probe would be zero length, preventing the light probe to ascend from or leave the area behind the geometry.

In addition, some initial light probes may already be in completely dark areas. Such areas include both outside of the scene and unlit, fully occluded areas in the scene. These points will not be updated by the gradient ascent algorithm because of the same reasons as in the paragraph above. Nevertheless, completely unlit light probes are containing valuable (although redundant) information, it is not possible to discard them completely.

Light decoupling

Due to previously noticed patterns for both algorithms to converge toward light sources and the fact that there are plenty of direct illumination solutions for real time applications, the gradient ascent algorithm was modified to employ only indirect illumination. Light sources (sources for direct illumination) highly contribute to the irradiance of visible points in the scene and highly contribute to the direction of the gradients. Thus, excluding direct illumination while capturing caused more uniform (less specular) light environment being captured, and prevented light probes from converging towards the light sources. Without any bright light sources, the irradiance gradient field was oriented towards lit surfaces (indirect illumination, reflected from geometry). This lead to light probes being attracted towards these surfaces. Unfortunately, as mentioned in previous chapter, most of the light probes while being traced towards geometry get placed behind it.

Second order gradient ascent

In order to prevent light probes from ascending behind the geometry, a variant of second order gradient ascent was implemented. Since there is no continuous information about irradiance field in the scene (irradiance is captured and gradients are calculated at discrete points), during each iteration of the ascent an extra light capturing is performed. To be exact: after gradient ascent, the light probe captures no irradiance (which means that the point was moved to a completely dark area), the new position of the light probe is discarded and the light probe is reset to the position before the gradient ascent. Step size is reduced during the following iterations and the previously reverted light probe is eventually ascended along the gradient direction without penetrating geometry.

3.1.3 Other aspects

Completely unlit areas

It was previously mentioned that some light probes might fall into completely unlit areas. This would generate completely black spherical harmonic functions and, in turn, gradient length would be zero. Light probes in these areas should be treated differently. Since the main task at hand is to analyze lit probes and reposition them in regards to the gradient field, completely dark light probes were ignored for the time being. After all the other probes were positioned during the gradient ascent, several extra probes were added to the cloud to account for those probes in the unlit areas. Due to the uniform initial grid, unlit areas have more than one light probe in it, thus the k-means clustering algorithm was used to convert all black light probes into several single probes representing the largest unlit areas.

Light probe sampling

Both algorithms were sampling spherical harmonic functions representing irradiance captured by virtual light probes. The first algorithm (spawn and merge) evaluates these functions to determine if they are too different, and if there is a need for another light probe. Since spherical harmonic functions are low-frequency representation, high precision evaluation for this method is not necessary. Therefore, the first sampling procedure implemented was very

crude. Spherical harmonic was sampled in six directions - positive and negative directions of the standard orthogonal basis. Such sample set was sufficient to effectively distinguish the considerably different spherical harmonic functions apart.

The second algorithm, gradient ascent, is using irradiance gradient fields. Initially, to calculate irradiance gradient from spherical harmonic, the same sampling procedure was used - six directions along the axes were sampled. Unfortunately, it was insufficient to calculate the gradient accurately. The gradient ascent using this sampling was not converging or converging very slowly. A simple but more accurate sampling was used for calculating gradient. The sample set is constructed from linearly added pairs of directions from the previously mentioned six directions; excluding pairs made from the collinear directions. This way, 20 additional directions are sampled uniformly across the surface of the sphere. This way the constructed gradient is more accurate and, in turn, the solution of the gradient ascent is found faster and more accurately.

Contrast of spherical harmonics

It was necessary to define a way to distinguish different light probes. In other words, it was necessary to have a way to compare two spherical harmonic functions. Since these functions were representing irradiance, the comparison function will be called contrast function.

The first function explored was defined as sampling and regular contrast check. Using one of the sampling techniques described above, the colors were evaluated at each sample direction. Then, contrast values between appropriate colors from different light probes were calculated. If any of these values rise above or all of these values fall below the given contrast threshold, spherical harmonics are considered respectively similar or different. Obviously, an improved sampling technique (with 26 samples per spherical harmonic) is more accurate than the one with only 6 samples.

Although the sampled contrast function is sufficiently correct in distinguishing spherical harmonics, it is not considering the whole surface of the sphere and is an approximate solution. As an alternative way to calculate the contrast, another function was implemented. The ontrast value of two spherical

harmonic functions was calculated as the mean squared error of the coefficients. Unfortunately, the spherical harmonic coefficients are unbounded real values which makes the calculated contrast value unbounded, too. Generally it is not causing any problems, because the identical spherical harmonics would have a contrast value of zero, but even slightly different functions might have extremely high value of contrast. This makes it hard or nearly impossible to set an appropriate threshold value.

Hierarchical grid

In order to explore other alternatives, a variant of hierarchical uniform grid was implemented for the merge and spawn algorithm. The merge and spawn procedure originally was based on the contrast and proximity thresholds and only one point inserted if thresholds were surpassed. The implemented hierarchical approach compared 8 light probes from the vertices of the uniform grid cubical cell. If these light probes were too different, the grid cell was subdivided into 8 sub-cells and extra light probes placed at the vertices. This approach posed very similar results (in comparison to the simple spawn and merge implementation), but incurred a formidable amount of calculation overhead. In the end, it was decided to not use it.

Light probes at the geometry

As one of the attempts to ease up the placement of the probes, in both algorithms the initial grid was either replaced or updated with light probes at geometry vertices. This includes light probes at vertices of the geometry and probes at light sources. Due to the nature of raytracing and numerical precision, these probes were offset to be slightly above the surfaces (in direction of geometry normal) or around light sources (for example, a point light source would have assigned several light probes around it).

Several problems were encountered with this approach. Since the light probes would be placed at vertices of the geometry, the amount of light probes is directly depending on the scene complexity. This leads to an extremely high probe count in complex, very tessellated environments. On the other hand, a low polygon count (or very large polygons) would cause undersampling and the point placement algorithms would converge slowly, would converge to a

very sparse and approximate result, or even diverge.

3.2 Data structure

In the second part of this work, a data structure to store all the light probes shall be created. To begin with, the data structure should allow efficiently access it with either exact spherical harmonic or interpolated combination of several. Furthermore, the data structure should not cause any lighting artifacts.

In a perfect situation, the light probe cloud could be created from an infinitely dense uniform grid. Then, for any point in a scene, the closest grid vertex would be considered as a local irradiance. Unfortunately, such conditions are not possible because of memory and (precalculation) time limitations. In the end, that is why the previous algorithms were implemented - to sparsely populate the scene with light probes which represent the lighting environment.

Both algorithms described in Section 3.1 are generating probe clouds from a uniform grid. Nevertheless, during the process the grid is deformed to fit the geometry and lighting environment more accurately. This makes any uniform data structure redundant to utilize. Thus, a data structure, which could efficiently hold and access an irregularly shaped cloud is needed.

3.2.1 Tetrahedral grid

One of the presentations in Game Developers Conference 2012 introduced an algorithm for interpolating light probes using tetrahedrons. The algorithm presented by Robert Cupisz [11] is a very viable solution. The cornerstone of the technique is to create a grid from all the light probes placed in the scene. Each grid cell would be formed from four light probes, at the vertices of the formed tetrahedron. This section explains the data structure and the methods used to construct this grid and use it in a real-time application.

Delaunay tessellation

The technique Cupisz presented is a construction of a grid-like structure from tetrahedrons. The structure for that grid proposed is that any point within all and any tetrahedrons is closer to the vertices of the tetrahedron than any

other vertex in the grid. Such requirement eliminates very thin or extended tetrahedrons. The construction of such grid is a process of creating Delaunay tessellation. Cupisz was referring to works by both D.F. Watson [12] and A. Bowyer [13] for creating the mentioned structure. Although the Delaunay and Dirichlet tessellations are related and both papers discuss both of them, Bowyer's paper is focused on the latter. The algorithm explained below is based on Watson's work since it is more focused on the Delaunay tessellation required for this application.

2D example

To begin with, the algorithm in 2D space will be described, since it is easier to visualize and grasp. The Delaunay tessellation in 2D space narrows down to a triangulation. The Delaunay triangulation is a set of triangles (where vertices of the triangles are points of the initial set) with no points of the initial set existing inside the circumcircles around the triangles.

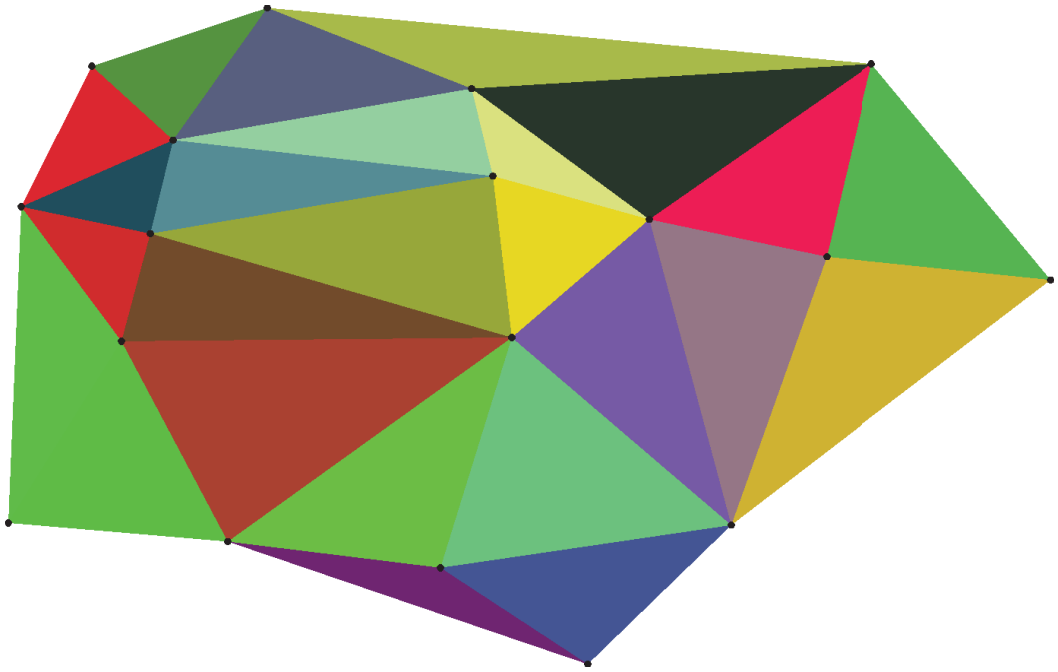


Figure 8: Example of 2D Delaunay tessellation

The algorithm requires that there is a defined initial triangulation with at least one triangle enclosing all the points in the data set. This is because the algorithm described by Watson assumes that all the points added into the triangulation are within the bounds of it. This is mainly to prevent points being inserted outside the triangulation, because such point would require to find border triangles in the current tessellation which adds unnecessary computations. For the two-dimensional case the initial condition is easily satisfied by adding five extra points. Four points at the positions of the initial set's bounding box and an extra point in the middle of the bounding box. These five points form the initial four triangles. It would be possible to create a single bounding triangle, or even choose three points from the initial set, but these approaches require more computational analysis in order to enclose all the data points from the initial set.

Having the initial triangulation, the insertion of another one (or any number) point is quite simple. For simplicity, I will use "a point intersects a triangle" with a meaning of "a point is within the circumcircle of a triangle". Our current (initial) Delaunay triangulation is a set of triangles DT with n_t number of triangles, n_p number of points. Let each point be named p_i and triangle t_j where respectively $0 \leq i < n_p$ and $0 \leq j < n_t$. To insert a point p^* into the triangulation, all the intersecting triangles have to be found. All these triangles will have to be replaced, because otherwise, after the pending point is inserted, it would still intersect them, making the Delaunay triangulation invalid. To determine what triangles will replace the intersected ones, a list of border edges has to be constructed. This is done by listing all the edges of the intersected triangles and discarding edges which appear twice (therefore, the edge is shared by two triangles and is an inner edge) in the list. Finally, a set of new triangles is created by creating triangles from each of the boundary edges and the point p^*. The proof that these newly created triangles do not invalidate the Delaunay triangulation could be found in previously mentioned paper [12].

3D and higher dimension spaces

The algorithm is extensible to higher dimensions in theory. Unfortunately, it is difficult to visualize it even in three dimensions, and almost impossible in higher dimensions. I will only describe the changes in the algorithm, which

1. Create initial Delaunay triangulation (DT) enclosing all data points
2. For each point p^* in the data points set:
   a) Find all triangles which intersect p^* (list IT)
   b) List all the edges from the intersecting triangles
   c) Remove duplicate edges. The remaining list of the edges is an outer edges (list OE)
   d) Remove intersecting triangles from list IT from the triangulation DT
   e) For each edge e^* in the list OE:
      i. Create triangle from the edge e^* and point p^*
      ii. Add it to triangulation DT.

Figure 9: Algorithm for Delaunay triangulation

are relevant to the addition of the third dimension. N.B. Previous term "a point intersects a triangle" was defined as "a point is within the circumcircle of a triangle". In three-dimensional case, "a point intersects a tetrahedron" is defined as "a point is within the circumsphere of a tetrahedron".

The essential difference is that the Delaunay tessellation is made of tetrahedrons instead of triangles. Therefore, the initial Delaunay tessellation enclosing all the points has to be a three-dimensional bounding box. The eight vertices of this bounding box will form the initial Delaunay tessellation. Analogous to the two dimensions, an extra point in the middle of the bounding box has to be inserted in order to create a valid tessellation. The algorithm for introducing new points to the initial tessellation is essentially the same and is depicted in Figure 10.

Limitations and adjustments

Theoretically the algorithm would find a unique Delaunay tessellation for both two-dimensional and three-dimensional cases. Unfortunately, numerical limitations of computer hardware causes truncation and rounding errors and during the search for intersected simplices (triangles in two dimensions

1. Create initial Delaunay tessellation (DT) enclosing all data points
2. For each point p^* in the data points set:
    a) Find all tetrahedrons which intersect p^* (list IT)
    b) List all the faces from the intersecting triangles
    c) Remove duplicate faces. The remaining list of the faces is an outer hull of the intersected tetrahedrons (list OH)
    d) Remove intersecting tetrahedrons from list IT from the tessellation DT
    e) For each face f^* in the list OH:
        i. Create tetrahedron from the face f^* and point p^*
        ii. Add it to tessellation DT.

Figure 10: Algorithm for Delaunay tessellation in 3D space

and tetrahedrons in three dimensions) false positives may be encountered. A false positive would cause invalid outer edge boundary or face hull, which would lead to creating simplices that invalidate the whole Delaunay tessellation. Such errors usually are encountered when data points are unevenly distributed (the distance to the other closest point varies significantly). Such distribution causes simplices with highly variable circumspheres (circumcircles), thus incapacitating the intersection algorithm to detect the intersecting point correctly.

Due to the same numerical limitations, a very dense light probe cloud could cause the creation of an invalid Delaunay tesselation. First of all, a probe cloud with a very high distribution variance would cause the problem defined in the chapter above. Moreover, the quantity of tetrahedrons in the tesellation is increasing exponantially depending on the quantity of light probes in the cloud. Each light probe inserted into the tesselation replaces at least one tetrahedron with at least two new tetrahedrons. Increasing amount of the tetrahedrons in the tesselation increases the required memory to store the tesselation itself.

The algorithm defined by Cupisz is working on a tetrahedral tessellation with a strictly convex hull. That requirement is essential for proper traversal (discussed later in this chapter) of the grid, especially when the target position is outside the hull of the tessellation. The point clouds generated by both spawn and merge, and gradient ascent are not guaranteed to have a convex hull structure. Additional processing would be required in order to identify and make the cloud hull convex. On the other hand, initial tessellation (created from bounding box enclosing data points) is already a convex hull which could serve this purpose. The drawback of this approach is that extra points, which were not light probes originally, were introduced into the tessellation. This is a minor hurdle, since capturing irradiance by creating extra several light probes is not an issue.

Interpolation with barycentric coordinates

The Delaunay tessellation structure is a tetrahedral grid-like structure. Such structure allows fast and efficient selection of the light probes which are relevant for lighting an object. When the object (or primitive) is within the bounds of the tetrahedron, the relevant light probes are the ones at the vertices of the tetrahedron. Furthermore, Cupisz is presenting a very convenient way to calculate how these four light probes should be combined.

The barycentric coordinate system was first time introduced by Möbius in 1827 [14]. This coordinate system defines a point in relation to the vertices of a simplex (a line segment in one, a triangle in two, a tetrahedron in three dimensions and so forth). The barycentric coordinates of the point are masses placed at vertices of the simplex while the point itself is positioned at centroid, or center of mass, of the simplex. Having in mind that the light probe cloud is stored in tetrahedral grid, any point within one of the tetrahedrons can have its irradiance function approximated by linearly composing all four light probes placed at vertices of the tetrahedron. Using the linear interpolation property described in Chapter 2.3.3, all four spherical harmonic functions representing irradiance captured by light probes are interpolated with barycentric coordinates as coefficients. The interpolated spherical harmonic function is an approximation of irradiance function at the point represented by those barycentric coordinates.

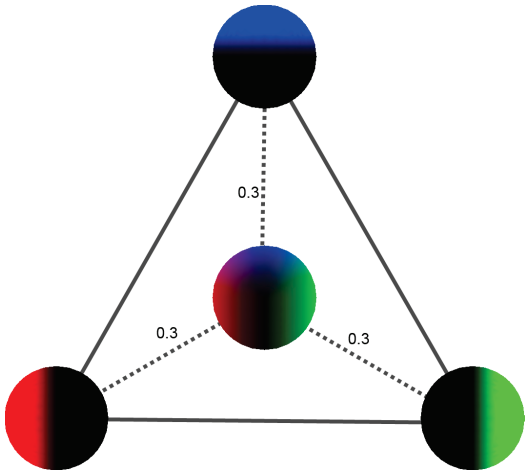


Figure 11: Example of spherical harmonic interpolation using barycentric coordinates

Although Figure 11 is depicting two-dimensional case, as mentioned in earlier chapters, visualizing it in three dimensions is rather difficult. Nevertheless, the interpolation is equivalent to the planar variant with an extra barycentric coefficient and a spherical harmonic function to interpolate from.

Traversal

Finally, in order to interpolate the four spherical harmonic functions assigned to vertices of the tetrahedron, the tetrahedron which encloses the position of a point, an object or a primitive has to be found. A linear search through all of the tetrahedrons is plausible only with a very low quantity of tetrahedrons to be real-time applicable. Again, Cupisz presented a very localized solution for this problem.

Given a point p for which we want to find a tetrahedron t such that the point p is within the volume enclosed by the t tetrahedron. First we have to choose an arbitrary tetrahedron as a starting point for the traversal. The position of the starting point is not very important, since the initial traversal is performed only once. The worst case scenario is that the traversal algorithm will have to march across the whole tetrahedral grid to find the actual tetrahedron.

The initial tetrahedron is required only for the first positional update. Later on, the previous tetrahedron is considered to be the starting point for the traversal. To update the position in the grid, it is necessary to calculate the barycentric coordinates for the point p in relation to the current tetrahedron t(for the first update - initial tetrahedron, for the following updates - previous tetrahedron). If all barycentric coordinates are positive, the object is inside the tetrahedron and no other changes are required. On the other hand, if any of the coordinates is negative, the object has moved outside the tetrahedron and its position in the tetrahedron grid must be updated. The convenience of using barycentric coordinates is that the negative barycentric coordinate indicates which way to traverse the grid. Let $p_{bary} = [p_0, p_1, p_2, p_3]$ be the point p expressed in barycentric coordinates, where p_i is corresponding to a coordinate (weight) in relation to tetrahedron's t vertices. If i-th barycentric coordinate is lower than zero, the point p is furthest from the i-th vertex of the tetrahedron t. This means, that the point p is potentially within a tetrahedron t_next which is adjacent to the face, which is opposed to the i-th vertex of current tetrahedron. Thus, the current tetrahedron t is updated to tetrahedron t_next. For better explanation, Figure 12 is depicting traversal in two-dimensional case. It is essentially the same principle with just one barycentric coordinate and one vertex less.

Of course, there is a possible situation when the point p is moved dramatically and a single step from one tetrahedron to another is not enough to find the tetrahedron. Such situation would require additional iterations of the traversal. I. e., calculating barycentric coordinates p_{bary} in relation to the new tetrahedron t, checking if there are no negative coordinates, stepping to another adjacent tetrahedron otherwise.

Drawbacks

There are several (although comparatively minor) drawbacks of this traversal algorithm. First of all, extra memory is required. In order to efficiently calculate barycentric coordinates, a transformation matrix that transforms a point position in world space to barycentric coordinates should be precalculated and stored. The size of the matrix is 3 by 3, thus extra 9 floating point numbers per tetrahedron have to be stored.

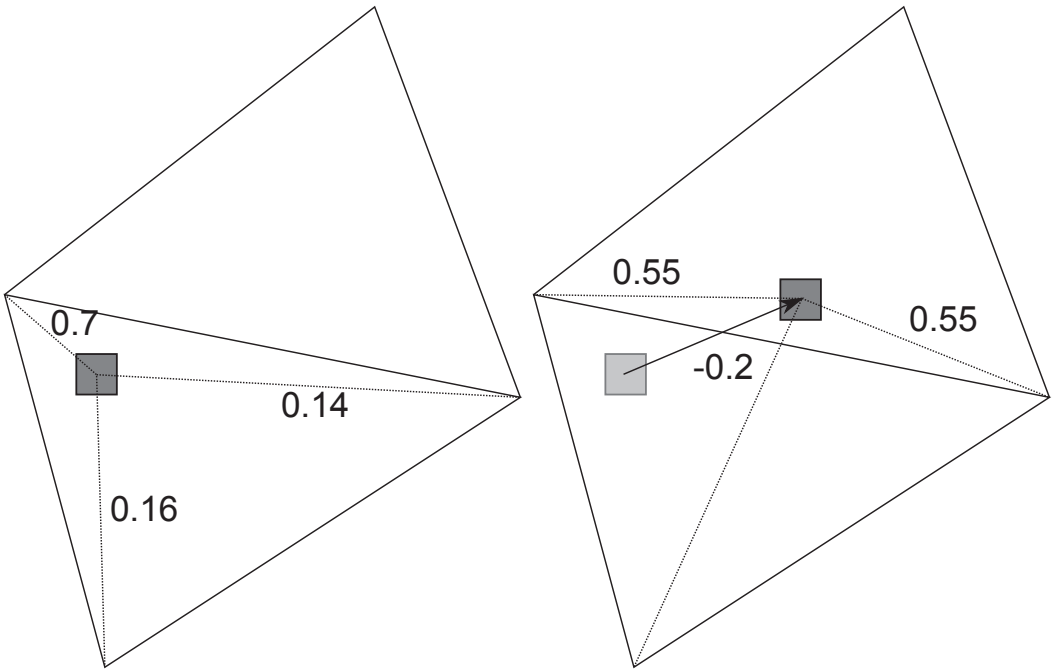


Figure 12: 2D Delaunay tessellation traversal using barycentric coordinates: a) object has non-negative barycentric coordinates; b) object has moved to another triangle, has one negative barycentric coordinate.

Secondly, this approach allows traversal only within the bounds of the whole tetrahedral grid. Because simply there are no tetrahedrons to traverse to the outside of the grid. Although the position of the point can be tracked along the surface of the outer hull and still be used to determine the closest tetrahedron, and, in turn, track which spherical harmonic functions to use to approximate irradiance.

Finally, due to numerical limitations it is possible that the traversal will continue in an infinite loop when the point is positioned exactly on a face, edge or vertex of the tetrahedron. Fortunately, the simplest solution is to keep track of already traversed tetrahedrons during the update and terminate traversal, if, according to the barycentric position, the grid has to be traversed to the already visited tetrahedron. Such situation means that point is practically on the border between two or more tetrahedrons and it does not matter which

tetrahedron will be assigned as the current one.

3.2.2 Geometry intersection

Combination of light probe placement algorithms and tetrahedral grid generator is an efficient solution to generate a sparse light probe cloud within a bounding box around a target scene. The generated structure represents lighting information in an approximated, and yet accurate due to low frequency nature of the lighting. In addition, the interpolation of the light probes is smooth across the grid, because of the linear transition between light probes selected for the interpolation. Nevertheless, the geometry and the setup of the scene might be very variable and have specific cases, causing the tetrahedral grid to be insufficiently dense in certain areas of the scene.

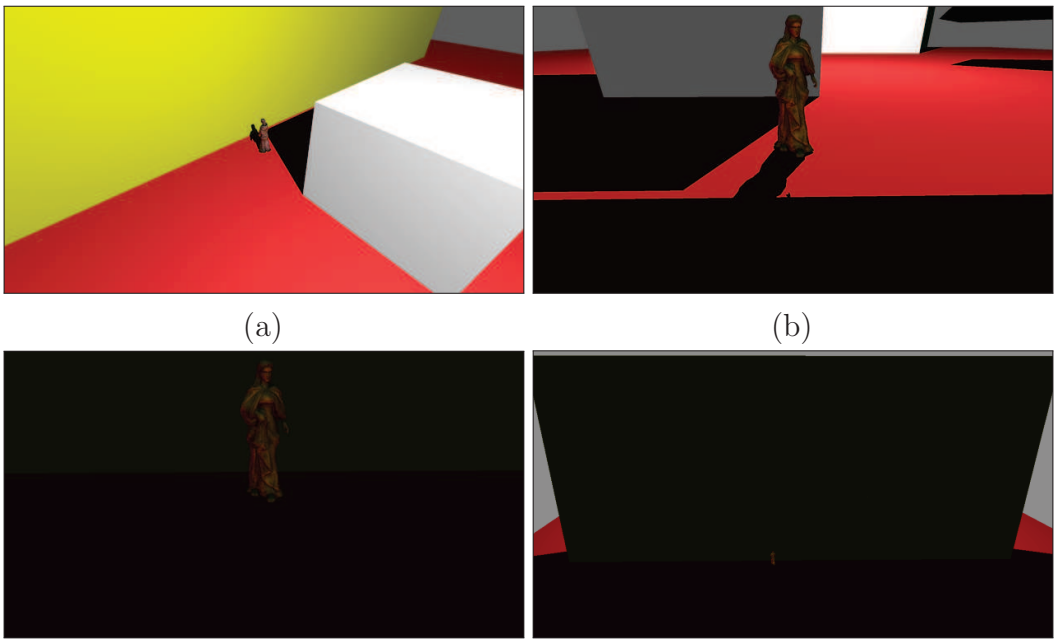
Light leaking

Light leaking is a term used by photographers to describe a phenomena caused by a faulty camera or other light-tight device [15]. Within the scope of this work, I will consider the light leaking phenomena when the object is lit up incorrectly because of the misuse of light probe across geometry from one lighting environment into another, unlit or differently lit, environment. As it is depicted in Figure 13, while the statue (object) is in front of the yellow wall, it is correctly lit by rebounding light from that wall. When the object is moved behind the wall, it is still retaining the yellow lighting although there is no light source or geometry to reflect that kind of light.

If a tetrahedron in the grid is sufficiently large (light probes/vertices are sufficiently far away), the tetrahedron's vertices could be situated in different lighting environments. In a general case it would not cause any problems, because the barycentric interpolation would smoothly interpolate the lighting information. Unfortunately, any geometry dividing and separating these vertices causes higher frequency light changes in the scene. These changes can not be represented and reconstructed by the interpolation alone.

Raytracing along the edges

A very simple and quite effective solution that I have implemented to tackle this problem is to raytrace along the edges. Given any and all edges between



(a) (b)

(c) (d)

Figure 13: Light leaking example in the simple scene: a) an object correctly lit; b) the object correctly lit in close up; c) the object moved to incorrectly lit area; d) the same as *c)*, overview.

two vertices in the tessellation, a ray is cast from one vertex to another and vice versa. Each of the rays cast are checked against geometry for intersection. The two rays are required in case of large two-sided geometry, such as thick walls: lighting environment has to be sampled on both sides of the wall. Any geometry intersecting the edge might cause a light leaking phenomena within tetrahedrons that share that edge. In order to eliminate that error in lighting, an additional point is inserted at the point of intersection. Of course, the position of new light probe is offset to prevent other ray tracing problems, as described in Section 3.1.3.

Oversampling and dense clusters

An addition of extra light probe samples is an effective approach to solving sudden changes in the lighting environment. Unfortunately, this leads to a very high number of additional light probes. I. e., the geometry between two

light probes will generate two more probes. A single vertex on the different side of geometry than the rest three of the tetrahedron will generate six more light probes: two probes for each edge connecting the single vertex on one side with each of the three vertices on the other side of the geometry. Such cases would create very localized and dense light probe clusters. These light probes, when inserted into a tetrahedral grid, create a relatively small tetrahedron. The problems arising from a relatively small tetrahedron were discussed in previous section.

First adjustment to reduce oversampling was to check if an edge is on the surface of the geometry. If both vertices of the edge are positioned on the surface of geometry, raytracing may report a hit somewhere in between those two vertices. The hit position would not contain any geometry that would cause high frequency light changes, thus an extra light probe would be redundant. This redundancy check was implemented with comparing the ray direction with the intersected surface normal. A ray cast along the surface will always be perpendicular to the normal of a surface. Thus, for the intersection with geometry to be valid, dot product of the ray direction and the intersected geometry normal has to be higher than zero.

Using both algorithms for placing light probes, some of those points are placed quite near the surface of the geometry. These points later on are inserted into the Delaunay tessellation and tetrahedrons intersecting the geometry are created. This would cause a situation described in first paragraph of this section. Very localized and dense light probe clusters need to be prevented. This is done by implementing another condition for inserting a new light probe. If a ray cast along the edge detects an intersection with geometry, a new light probe is inserted into the tessellation only if this condition is satisfied: the intersection point is sufficiently far away from both vertices of the edge. This condition will prevent from new light probes being inserted almost at the same position as current vertices. A two-dimensional example is depicted in Figure 14.

Tetrahedrons in the tessellation have varying size and varying edge lengths. The last condition relies on the distance from intersection to the vertices of the edge. Static distance threshold could invalidate the test in general. For

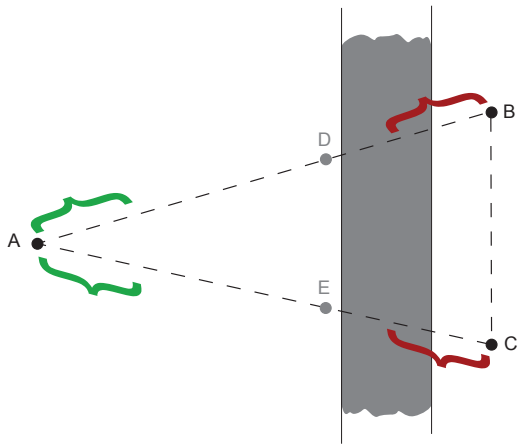


Figure 14: Condition of insertion along the edge: Curly brackets indicate the minimum required distance from the geometry. Probe A is sufficiently far away from the geometry, while probes B and C are not. New probes are going to be inserted only at (roughly) positions D and E.

example, if the edge length is smaller than the threshold, any and all intersection points would be too close to be acceptable. A dynamic, edge length dependant threshold is required.

4 Results

The whole implementation was tested on two scenes. The first one is a simple scene with very basic primitives and comparably low polygon count. Later on in the text this scene will be referred to as the simple scene. The second scene is more widely known - a model of the Atrium Sponza Palace. The model was retrieved from Crytek CryENGINE 3 website [16]. This scene will be referred as the Sponza scene.

4.1 Light probe cloud

To begin with, the results of light probe placement algorithms will be discussed. The light probe placement results is visualized in Figure 15. This result was achieved using gradient ascent algorithm. The points from the initial uniform grid in Figure 15a were traced and sparsely positioned throughout the scene. The points were placed in proximity of all lit surfaces, sampling any and all reflected radiance from those surfaces. Slightly more light probes were placed near the areas of the surfaces which are closer to the light source. The cloud generated is sparse and covers most of the scene's light variance at the same time.

Although two different algorithms were used, both posed similar results in certain aspects. For example, in Figure 16 it is shown that with spawn and merge algorithm generated probe cloud is quite sparse and light probes are positioned in approximately similar locations as in Figure 15d.

Unfortunately, it is easy to notice that some of the light probes are still remaining in rather uniform distribution. Using spawn and merge algorithm probes may retain in their initial uniform grid positions. In addition to this,the points are spawned directly between two other points, thus pertaining the linear position dependency to the original probes.

The simple scene was created specifically to test and visualize corner cases (shadowed areas, interreflection). The Sponza scene was used to resemble an actual application that light probe cloud generation could be used in. Consequently, an attempt to visualize how light probes are placed in that scene can be seen in Figure 17. The light source is roughly in the midpoint of nave.

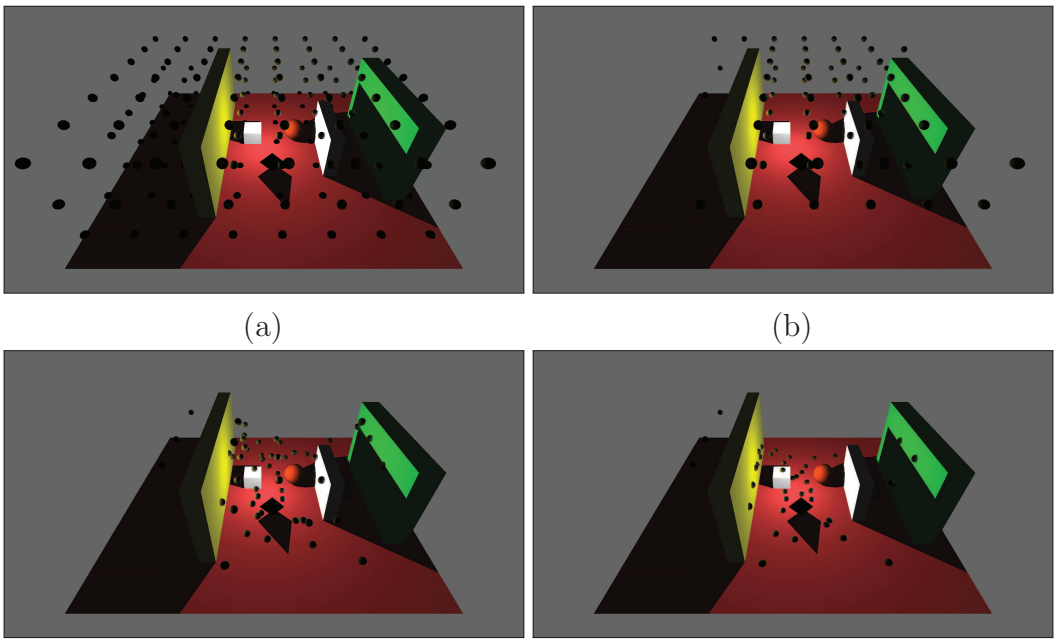


Figure 15: Light probe generation in simple scene with gradient ascent: a) initial uniform grid; b) black probes removed; c) intermediate iteration; d) final placement.

Light probes are mainly placed near the floor since it is reflecting the most light from the light source. Although the surrounding walls and flags are less reflective, they still contribute to the light probes. Due to this contribution, several light probes are positioned slightly closer to them.

In Figure 17d it is possible to see that although the right aisle is mostly shadowed (because of the flags and upper balcony), the lit up areas near the intersection of the floor and right wall have light probes.

4.2 Tetrahedral grid

A tetrahedral grid was created from the light probes placed in the previous step. The grid and tetrahedrons were created using Delaunay tessellation algorithm. Each tetrahedron in the resulting grid is defined by the following data:

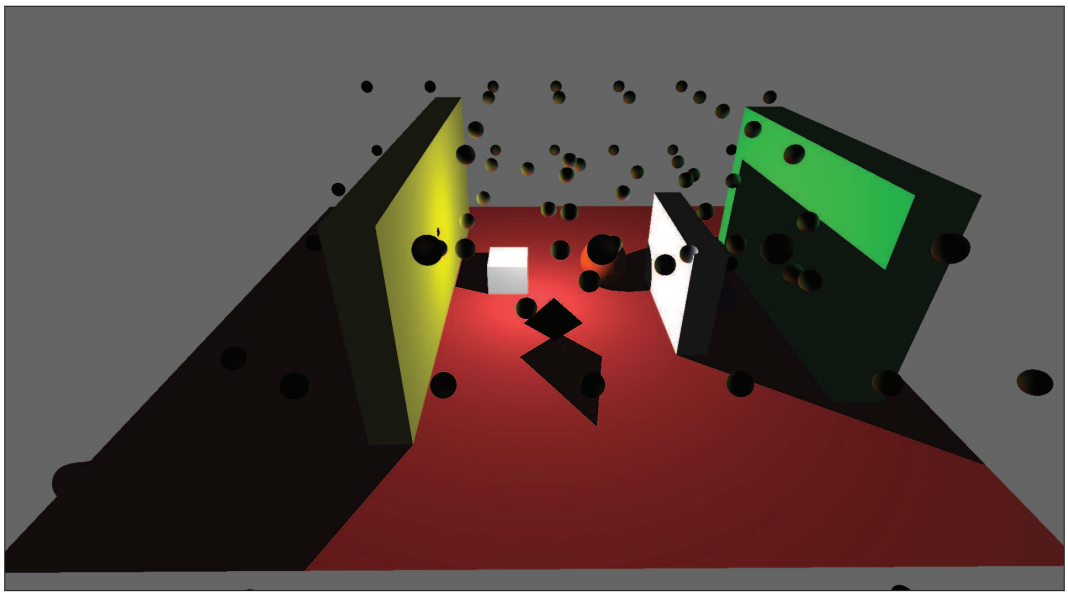
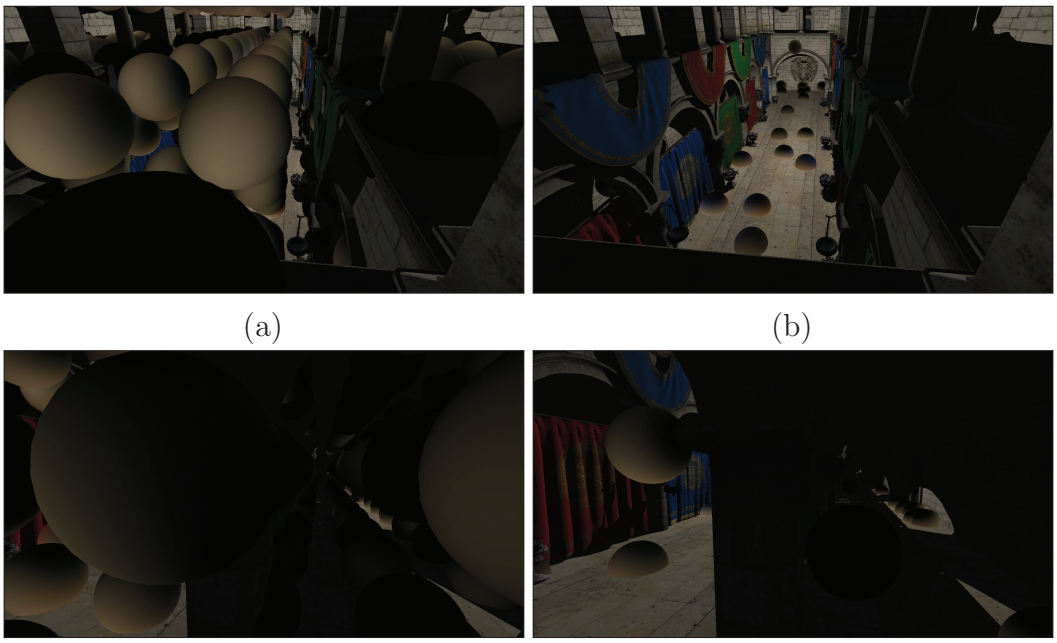


Figure 16: Light probes placed in the simple scene using spawn and merge algorithm.

1. Four indices denoting light probes in the placed light probe cloud;
2. Four indices denoting neighboring tetrahedrons;
3. Precalculated 3-by-3 matrix for transforming world space coordinates into the tetrahedron's barycentric coordinates.

In Figure 18a, the initial Delaunay tessellation is displayed. Although it is not part of the generated light probe cloud, it is essential for the proper Delaunay tessellation creation. In Figure 18d, for comparison purposes the tessellation is visualized without the initial tetrahedrons or extra light probes inserted. The formed tessellation is highly irregular and without a convex hull.

Figure 19 depicts the structure of the tetrahedral grid. Since the grid is rather tessellated and three dimensional, it is difficult to imagine the actual structure, but the mentioned figure at least provides a solid clue how it is laid out and structured in a smaller scale.



(a) (b)

(c) (d)

Figure 17: Light probe placement in the Sponza scene: a) uniformly placed probes; b) final placement; c) uniformly placed probes viewed from a corner; d) final placement viewed from the corner.

Since tetrahedral grid construction is less dependant on geometry, there are no actual differences from generated tetrahedral grid in the simple or the Sponza scenes. Figure 20 shows how light probes are associated in the tetrahedral grid.

4.3 Final result

Finally, the end result of an actual light probe cloud with tetrahedral grid implementation showing the different scene look with and without point clouds. Although, due to the dominant colors of the Sponza scene, certain effects are less visible than in the simple scene.

The first example is aimed to illustrate the interreflection. Figure 21c shows that the upper part of the statue (which is rendered without light probe cloud illumination) is completely dark on the shadowed side, while the bottom part

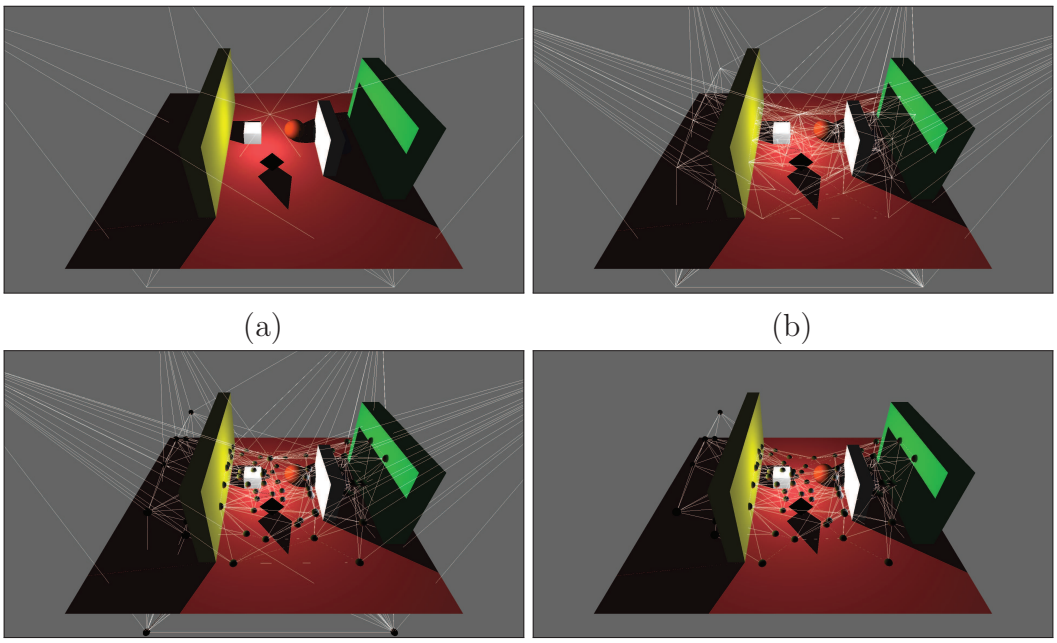
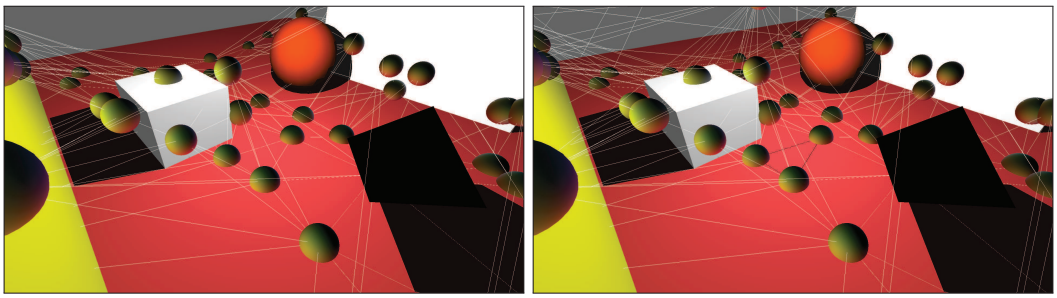


Figure 18: Tetrahedral grid in a simple scene: a) initial tessellation; b) final tesselation; c) final tesselation with light probes; d) final tesselation with light probes (initial tetrahedrons not displayed).

(which is rendered with light probe cloud illumination) is illuminated by both red floor and yellow wall.

With automatically placed light probes, shadowed areas receive illumination from lit up nearby surfaces. For example, in Figure's 22c upper part the statue is lit up by light probes which captured light from the brightly lit white wall. On the other hand, the bottom part of the statue is completely dark since it is in the sphere's (which is out of view) shadow.

In addition to that extra illumination in shadowed areas, spherical harmonic light probes reduces overshadowing from self casted shadows too. In Figure 23c the statue is roughly beneath a light source. The upper part of the statue is almost completely black because of the shadow it casts on itself. The bottom part is instead lit up since a lot of light is reflected upwards from the surface beneath the statue. Since the light source is directly above, the intensity of



(a) (b)

Figure 19: Closeup of final tetrahedral grid in the simple scene: a) without initial tessellation tetrahedrons; b) with all tetrahedrons.

the light reflected from the floor is higher than in other examples.

The final example for the simple scene is another perspective for the inter-reflected illumination. Objects in shadow may be lit up by other objects that are outside the shadow. Objects that receives significant amount of the radiance to reflect it towards objects without any direct illumination. Figure 24c displays how the bottom part of the statue, while being in the shadow casted by middle white wall, is illuminated by the green wall from one side and dimly illuminated by yellow wall from the other side.

During the application runtime, the object which is lit up using light probe cloud is lit up only by the four closest light probes. The light probes were placed on the vertices of a Delaunay tesselation. The Delaunay tesselation gurantees that for any point within any of the simplices (tetrahedrons) the four closest vertices are the vertices of the simplex. This is due to the definition of the Delaunay tesselation - no vertex can be within circumsphere of any simplex. In Figure 25 it is shown both the rendered scene, visualization of light probes, and which light probes (or which tetrahedron) is currently used to light up the object.

Thus, the statue lit up by these light probes (as displayed in detail in Figure 26) receives lighting from the wall. The upper part of the rendered screen shows that, without light probe grid information the portion of the statue which is in the shadow would be completely dark.

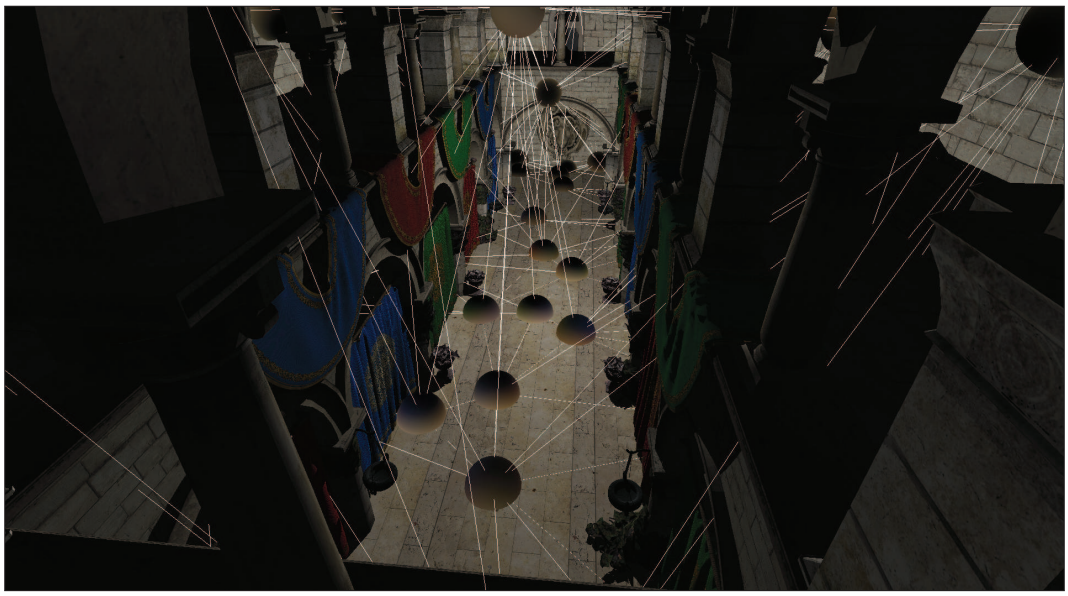
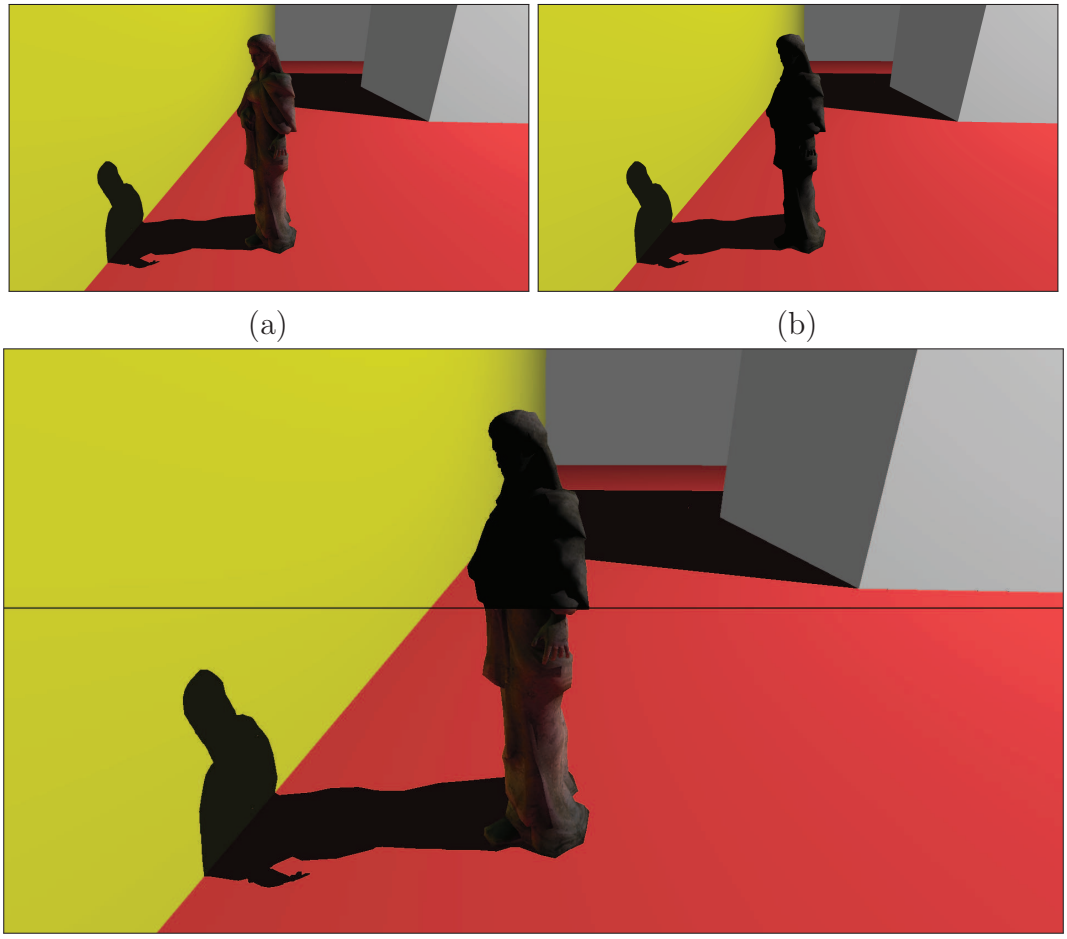


Figure 20: Tetrahedral grid in Sponza scene.

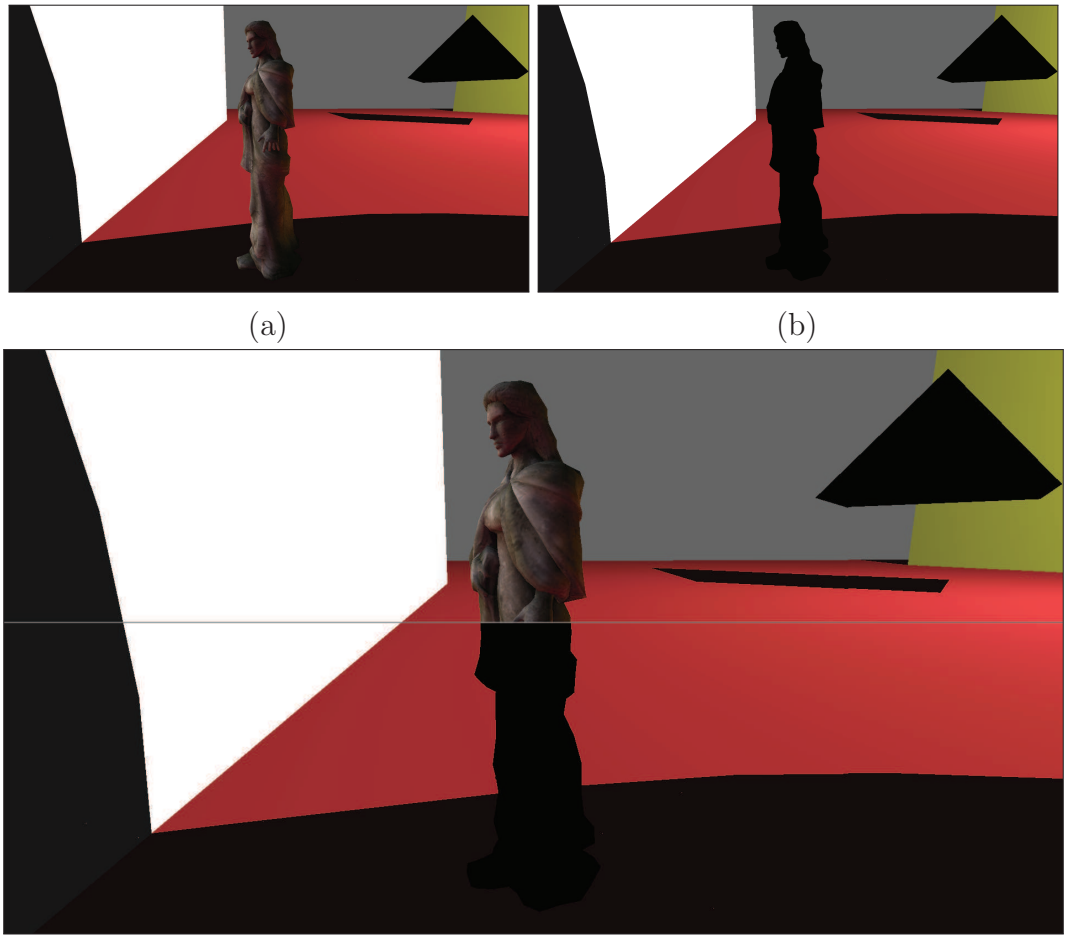
Finally, the light probe grid is created so that light reflected from all materials are captured. Although this feature is more visible and apparent in Figure 24 and other examples with the simple scene, it is noticeable in the Sponza scene too. In Figure 27 a dim blue light is casted on top of the head of the statue. That blue light is reflected from a blue curtain (or flag) in front of the statue. In the last example (Figure 28), the right side of the statue is visibly lit up by a red reflection from a red curtain.



(a) (b)

(c)

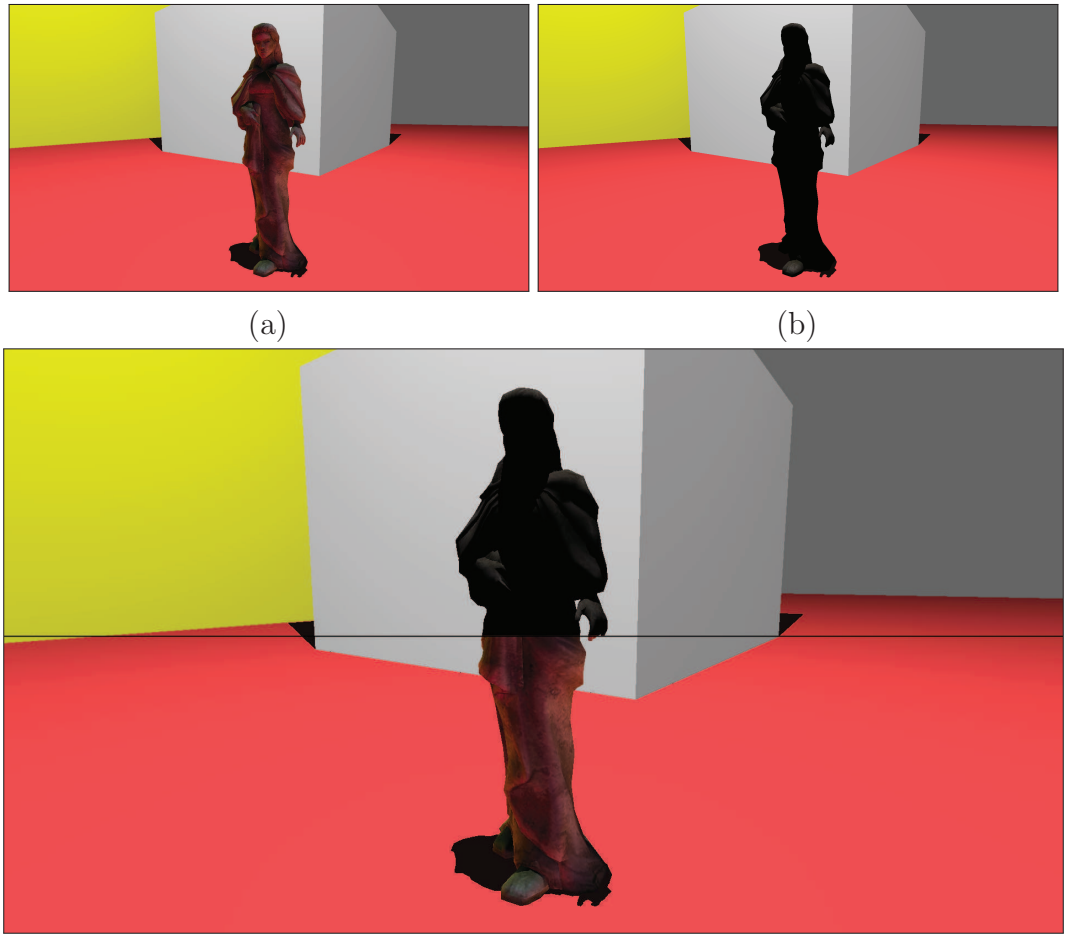
Figure 21: Final rendering example 1: re-bounced light from yellow wall illuminates statue's shadowed side. Also, red lighting from the floor on the bottom of the statue is visible.



(a) (b)

(c)

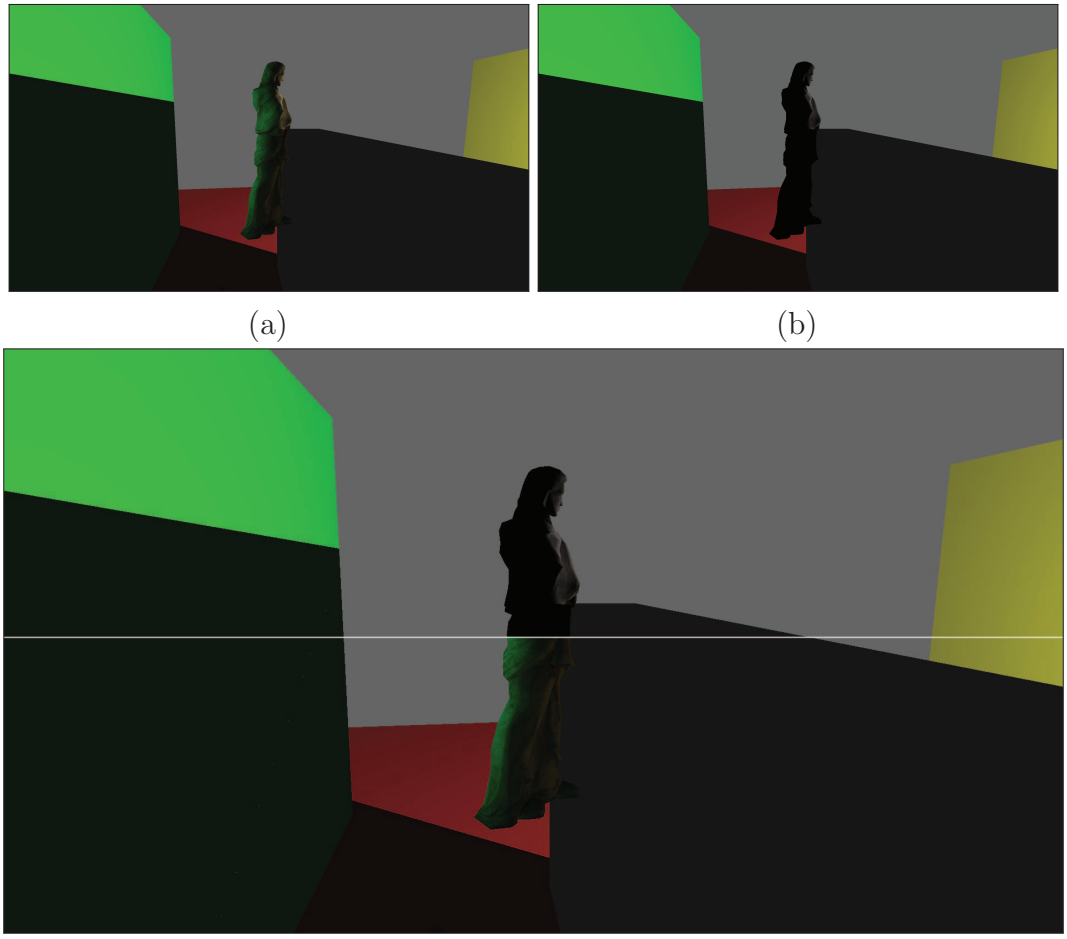
Figure 22: Final rendering example 2: Although the statue is in the shadow, white bright wall is illuminating it.



(a) (b)

(c)

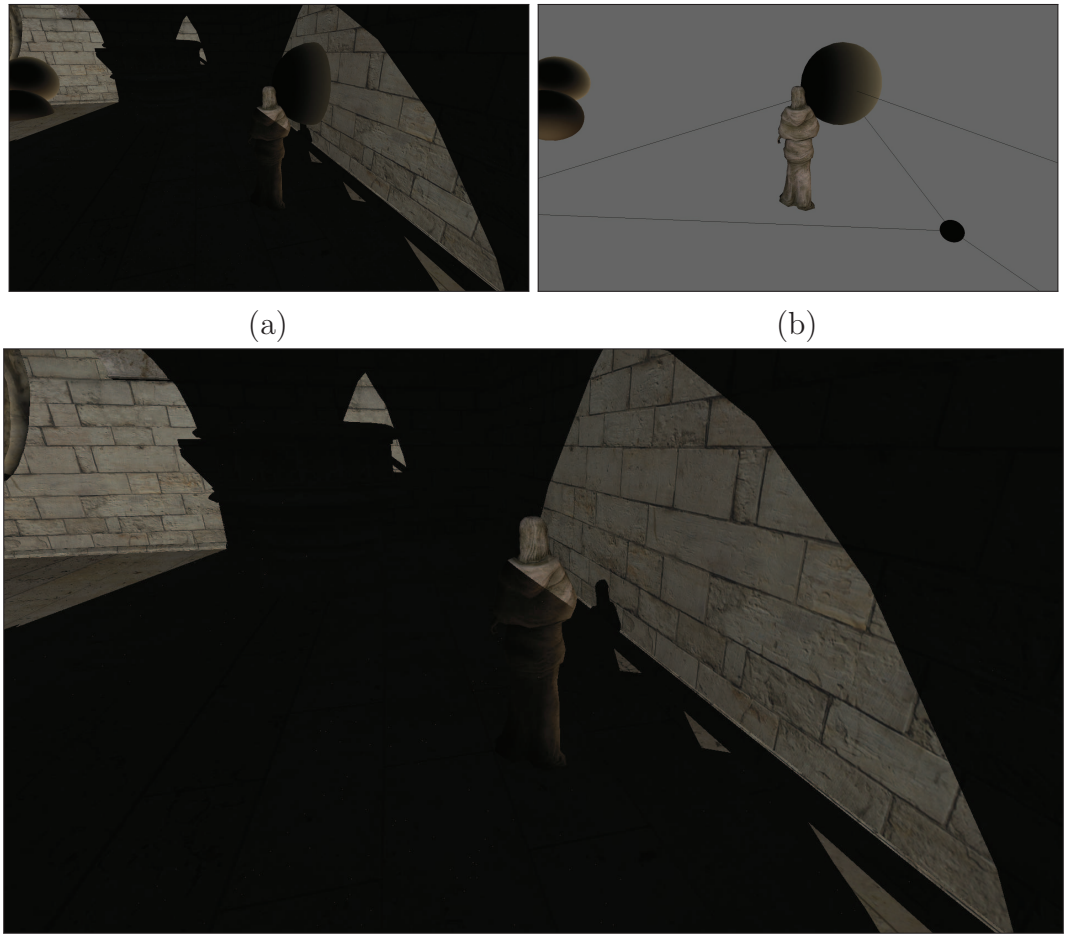
Figure 23: Final rendering example 3: The light source is almost directly above the statue. The statue is illuminated from below by the brightly lit red floor.



(a) (b)

(c)

Figure 24: Final rendering example 4: Back side of the statue is lit up by the green wall. The front lower part of the statue (even being largely in shadow) is lit up by the opposing far yellow wall.



(a) (b)

(c)

Figure 25: Final rendering example 5: a) Scene with light probes visualised; b) only light probes and tetrahedral grid cell visualized; c) actual rendering of the scene with statue lit up by the light probe cloud.

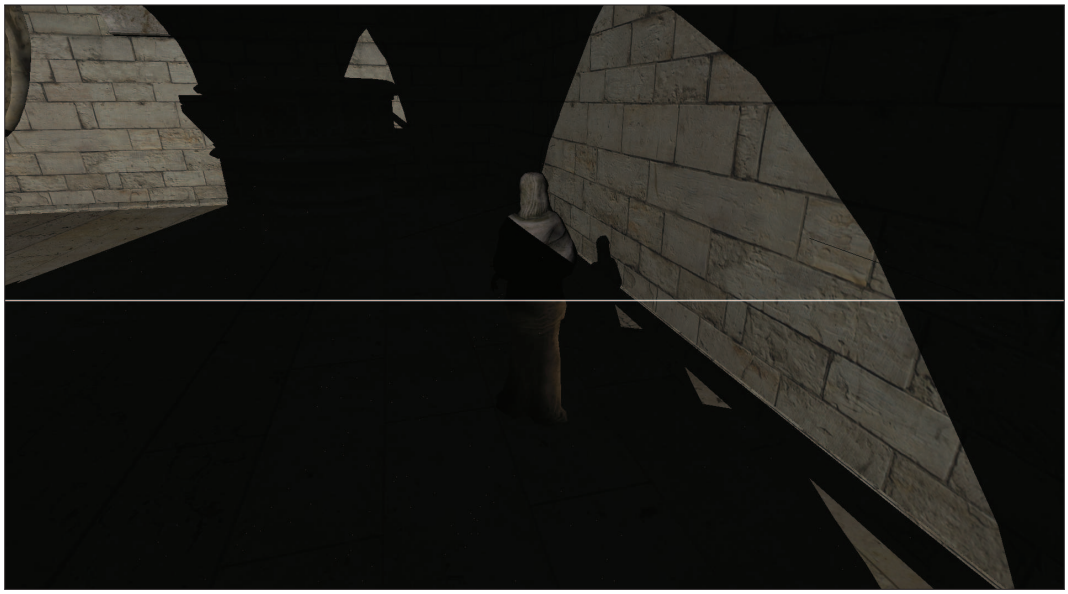


Figure 26: Final rendering example 6: Sponza scene in split-view. The statue in the bottom part of the image is lit up with the light from the wall.

Figure 27: Final rendering example 7: Top of the statue is lit up by dim blue light reflected from the blue curtain.

Figure 28: Final rendering example 8: Side of the statue is lit up by dim red light reflected from the red curtain.

5 Conclusions

In the first part of this work, two different light probe placement algorithms were analyzed, implemented and tested. The first algorithm - spawn and merge - is an adaptive approach. Adaptive processes are widely used, thus it was tested with this particular problem. In the initial stage of the method, a scene is populated with light probes and lighting information is captured in spherical harmonic functions. Afterwards, additional light probes are placed (or probes are merged) depending on the contrast value of the light captured by the existing light probes. The process is repeated until termination conditions are satisfied.

The second light probe placement algorithm implemented is a variant of gradient ascent algorithm. The initial stage of the algorithm is the same as in the spawn and merge algorithm. During each iteration of the algorithm, an irradiance gradient field is constructed from light probes already placed in the scene. Each light probe is repositioned according to the direction of the gradient in the position of the probe. The algorithm is terminated when the displacement of the probes after iteration is sufficiently small. In other words - when gradient ascent algorithm "placed" all the light probes in the local maxima of the irradiance gradient field.

In terms of results, both of the placement algorithms performed approximately equally. Despite that, the spawn and merge algorithm with a large initial grid density was performing slower (due to exponential growth of light probes during intermediate steps). Since the gradient ascent algorithm is not creating any new light probes, its completion time is linearly dependant on the density of the initial grid. Having this in mind, the gradient ascent algorithm is more reliable and robust. Of course, both these algorithms would be executed as offline precomputations, but an exponential growth of calculation time may not be preferred.

As the second part of this work, a viable data structure for storing a light probe cloud was analyzed and implemented. The main criteria for the data structure was to be able to quickly obtain irradiance data captured within spherical harmonics. In addition, the data should be interpolated, since light

probes are placed sparsely. The chosen algorithm relied on the Delaunay tessellation. Delaunay tessellation places all the light probes in tetrahedral grid (with four light probes assigned to each tetrahedron). Delaunay tessellation is constructed in such way that for any point within any tetrahedron, the four closest light probes are the ones at the tetrahedron's vertices. The transition and traversal throughout the grid is performed by evaluating the barycentric coordinates of the point in the barycentric coordinate system of the current tetrahedron.

During the process, several other techniques were implemented or tested. These includes an implementation of several approaches for calculating contrast of light probes, a hierarchical light probe grid, a k-means algorithm for grouping completely black light probes and others. Some of these secondary algorithms and techniques were used in the final compound algorithm, others were discarded due to insignificant improvement or, on the contrary, downgraded quality of the results or the performance.

Due to the requirements of one algorithm, some adjustments were introduced to other algorithms. To be specific, the Delaunay tessellation algorithm requires to have initial convex hull. In turn, it required to either find a convex hull of the light probe cloud or introduce several new light probes which would enclose already present in a convex hull. The latter was chosen since the amount of light probes introduced is insignificant and finding convex hull is not a trivial problem.

Finally, the resulting light probe cloud with the Delaunay tetrahedral grid structure was used in a rendering engine. The rendering engine was used to compare the conventional (without any global illumination) rendering with the rendering using light probe cloud. The light probes positioned at certain places around the scene captured global illumination information, thus illuminating the object with an approximated and interpolated global illumination. Such global illumination effects as ambient lighting and inter-object illumination was noticeable in the final renderings.

Since irradiance is captured during an offline process, the only calculations performed by real-time application (the renderer) are linear interpolation, eval-

uation of spherical harmonic functions, and update of the object's position in the Delaunay tessellation. The former operations are accelerated by modern CPUs and GPUs, and the latter is only one matrix multiplication per update, and the impact on the performance was not detected.

6 Future work

Possible future work could include more detailed research towards using not only indirect illumination, but direct illumination too. This would require more precise placement of light probes, especially around shadow volume edges. Shadow volume geometry in could be treated as an actual geometry since it is influencing lighting conditions drastically. However, underlying spherical harmonic functions stores only low frequency, diffuse lighting information. Thus even with a precisely placed and sufficiently sampled light probe cloud it might be difficult to capture all specular light information.

On the other hand, the algorithm does not depend on the underlying light probe basis, as long as it is possible to linearly interpolate two light probes. An alternative basis capable of preserving and storing different lighting data may be used. Nevertheless the selected basis, high frequency lighting changes will have to be sampled and cannot be expected to be reconstructed with linear interpolation.

In the scope of this work, real-time global illumination using light probe cloud was used only for dynamic objects in the scene. This is due to the fact that there are other methods to render globally illuminated static objects in real time. An example of such technique - precomputed (baked) global illumination maps. A unified algorithm to render global illumination for both static and dynamic objects would be beneficial since it would be easier to maintain for the developers and/or artists and reduce required memory for global illumination maps.

7 References

[1] Kajiya, James T. (1986), "The rendering equation", SIGGRAPH 1986, p. 143.

[2] Immel, David S.; Cohen, Michael F.; Greenberg, Donald P. (1986), "A radiosity method for non-diffuse environments", SIGGRAPH 1986, p. 133.

[3] Taylor series. Encyclopedia of Mathematics. http://www.encyclopediaofmath.org/index.php?title=Taylor_series&oldid=31211, last accessed 2014.04.22.

[4] Fourier series. Encyclopedia of Mathematics. http://www.encyclopediaofmath.org/index.php?title=Fourier_series&oldid=25550, last accessed 2014.04.22

[5] Weisstein, Eric W. "Spherical Harmonic." From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/SphericalHarmonic.html, last accessed 2014.04.15.

[6] Kautz, Jan; Sloan, Peter-Pike; Snyder, John. "Fast, Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonics", Eurographics Rendering Workshop 2002, June 2002.

[7] Sloan, Peter-Pike; Kautz, Jan; Snyder, John. "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments". Proceedings of SIGGRAPH 2002.

[8] Ambient Occlusion, GPU Gems, Chapter 17, http://http.developer.nvidia.com/GPUGems/gpugems_ch17.html, last accessed 2011.04.15

[9] Nicodemus, Fred E.; "Directional Reflectance and Emissivity of an Opaque Surface," Appl. Opt. 4, 767-775 (1965) http://www.opticsinfobase.org/ao/abstract.cfm?URI=ao-4-7-767, last accessed 2014.04.15

[10] Chadjas, M.G.; Weis, A.; Westermann, R.; "Assisted Environment Map Probe Placement", SIGRAD 2011.

[11] Cupisz, Robert; "Light probe interpolation using tetrahedral tessellations", Game Developers Conference 2012.

[12] Watson, D. F.; "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes", The Computer Journal, vol 24, no 2, pp. 167-172., 1981.

[13] Bowyer, A.; "Computing Dirichlet tessellation", The Computer Journal, vol 24, no 2, pp. 162-166., 1981.

[14] Möbius, A. F. (1827). Coxeter, H.S.M. (1969), "Introduction to Geometry, 2nd ed.", Chapter 13.7, pp. 216-221, New York: Wiley

[15] NK Guy, PhotoNotes.org, "The Photographer's Dictionary", `http://photonotes.org/cgi-bin/entry.pl?id=Lightleak`, last accessed 2014.04.07

[16] Crytek CryENGINE3 Downloads, `http://www.crytek.com/cryengine/cryengine3/downloads`, last accessed 2014.04.08.