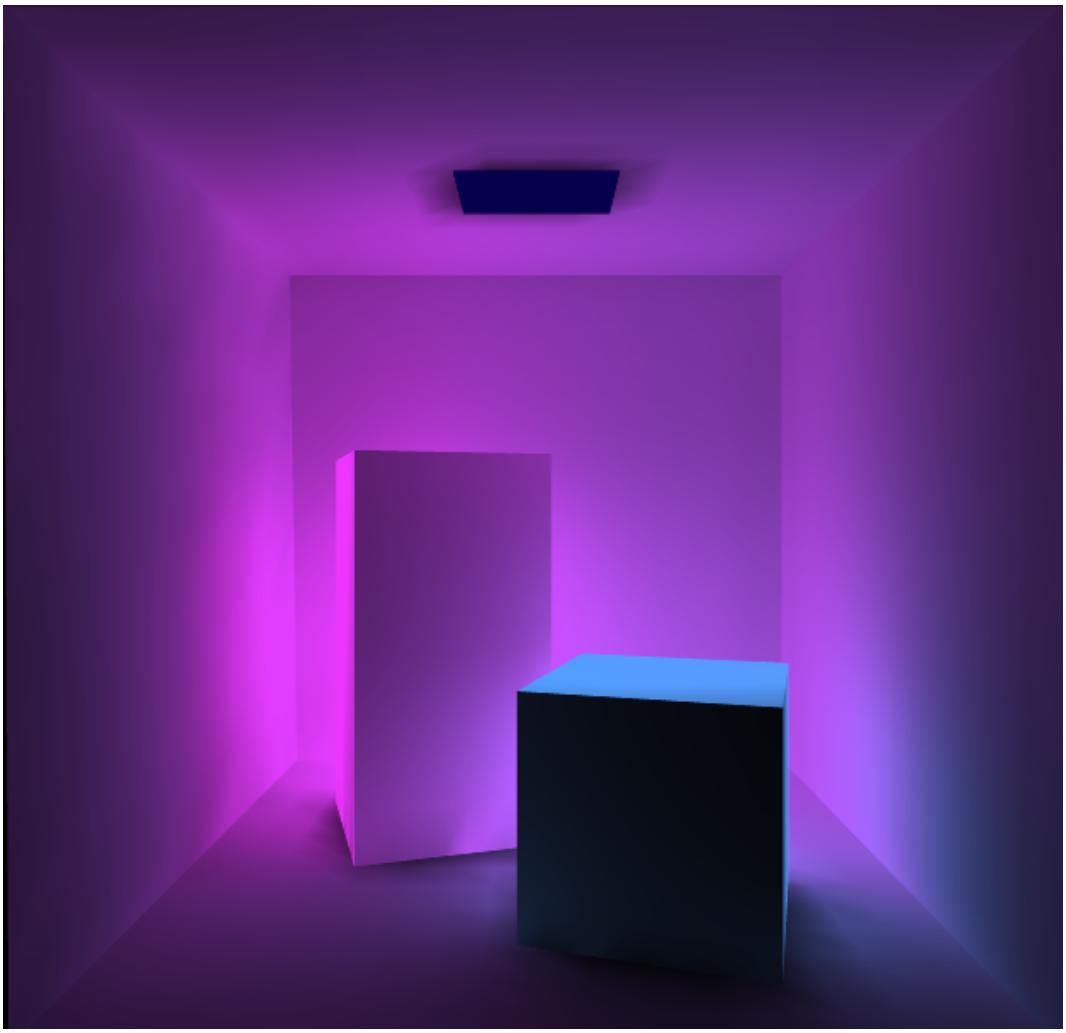
TU Delft

DaisyRiot: Radiosity, spectral rendering and fluorescent materials



Honours programme 2019

V. Hoveling
M. Roelvink

Abstract

This report documents the DaisyRiot global illumination renderer. The renderer applies the radiosity method and uses spectral rendering in order to simulate fluorescent materials: materials can reflect energy of a given frequency at another frequency. Rendering is accelerated by NVIDIA OptiX and Cuda parallelization. In this document you will find details of the design of the renderer, its architecture, a discussion of its results and performance and a reflection on its future. All source code is released online: github.com/asylunatic/DaisyRiot.

Keywords— radiosity, global illumination, spectral rendering, fluorescence

1 Introduction

The radiosity approach to global illumination has been studied for over three decades [4]. With the rise of new frameworks such as NVIDIA Optix [8], which put real time raytracing on the horizon, we see good reason to bring new life to radiosity as a solution for global illumination and explore it's possiblities further.

1.1 Research problem & approach

Our research focused on the question: How can we enable new materials in radiosity rendering? Materials in radiosity are notoriously hard, as the radiosity equation is based on the assumption of perfectly diffuse materials. Properties such as specularity violate that assumption directly.

We have therefore focused on modeling a property that does not violate the assumption of diffuse materials, yet allows us to bring a new range of images. We do this by enabling materials that can influence the wavelengths of the energy they reflect: fluorescence. We use the newest frameworks to speed up the renderer and enable an interactive experience.

2 Related work

2.1 Radiosity rendering

Radiosity is a global illumination algorithm based on thermodynamics: it models how energy moves through a space. In radiosity, all surfaces are subdivided into small 'patches'. Any two patches I and J have a view factor $F_{I\to J}$ that describes how much energy is reflected off of patch I to J. It is modeled by the following integral:

$$F_{I\to J} = \frac{1}{A_I}\int_{A_I}\int_{A_J}\frac{\cos\theta_I\cos\theta_J}{\pi s^2}dA_I dA_J \tag{1}$$

The two patches have a reciprocity relation:

$$\begin{aligned} A_i * F_{I\to J} &= A_J * F_{J\to I} \\ or: F_{J\to I} &= \frac{A_i * F_{I\to J}}{A_J} \end{aligned} \tag{2}$$

The radiosity of a patch is the energy per unit area leaving the patch surface per discrete time interval and is the combination of emitted and reflected energy. The radiosity of a patch I is formalized as:

$$B_I = E_I + \sum_{J=1}^{n} F_{J\to I} B_J \rho_I \tag{3}$$

in which B is the radiosity, E the emissiveness and ρ the reflectiveness of the surface. This is the radiosity equation, which we can also represent as a linear system of equations:

$$B = E + \rho K B \tag{4}$$

where K is a square matrix of view factors. Solving this system yields the radiosity of each patch.

2.2 Spectral rendering

In spectral rendering, a scene's light transport is modeled after individual or small bands of wavelengths, as opposed to, for example, the tristimulus values model (in which color measurement relies on a system of only three values). Simulation of the light spectrum enables a more physically accurate simulation of the scene. To display any image to screen, the spectral color space must be converted to screen color space (sRGB).

2.3 Fluorescence

Fluorescence is the emission of light by a substance that has absorbed light or other electromagnetic radiation. Generally, the emitted light has a longer wavelength (lower energy) than the absorbed radiation. In other words: fluorescent materials can bend light from one wavelength to another. [3] developed a mathematical model for fluorescence, which was implemented in the Rayshade raytracer. To the best of our knowledge, the model has not been combined with radiosity rendering before.

2.4 Cuda & Optix

Both CUDA and Optix are frameworks for parallel computing created by NVIDIA. CUDA is an API which allows for use of the GPU for general purpose processing. It is designed to work with programming languages such as C and C++. CUDA gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. Optix is a ray tracing API that builds on top of CUDA: instructions regarding what a ray should do are supplied in in the form of CUDA kernels, enabling the ray tracing process to be simulated in parallel.

3 Design

We will outline the most important design choices in our presented renderer. We have identified five areas in which important design decisions were made. We will discuss each of those in the following subsections:

3.1 View factor calculation

The integral that describes the view factor in Equation 1 can only be computed directly for very simple cases, for most cases it cannot. There exists a large body of work on approximating the view factors. We have considered three approaches:

1. **Nusselt analog:** With this method, the view factor $F_{I \to J}$ is calculated by projecting J onto the hemisphere of the I, and then projecting further from the hemisphere onto a unit circle around the center of I, as illustrated in Figure 1. The percentage of the unit circle covered by J is equivalent to $F_{I \to J}$. However geometrically pleasing and intuitive, the Nusselt analog is not perfect: any patch has only one center against which all other patches are measured and ignores the rest of the surface. This means that if I is very large and J is very small, but perfectly centered above the center of I, then $F_{I \to J}$ would be disproportionally large.

2. **Riemann sum approximation:** The integral that describes the view factor (Equation 1) can be approximated with a Riemann sum: to obtain the differential areas of any patches I and J, both I and J are subdivided first. As we use triangles for patches (see Section 4), I and J are each split up into four triangles, as illustrated in Figure 2. The integrands between all of the subdivided patches are calculated and weighted by the area of I.

3. **Stochastic approach:** From a patch I, a set of rays is shot according to a probability distribution that models reflected light off diffuse surfaces. The view factor $F_{I \to J}$ for any patch J in the scene is then retrieved by calculating the percentage of rays from I that hit J. This works for simple scenes. However, the amount of rays required for a good estimate of the view factor grows with the amount of patches in the scene and becomes very large very quickly. Therefor is this an undesirable approach for complex scenes.

Take note that for both the Nusselt analog and the Riemann sum approximation, an additional visibility factor was taken into account: to determine whether any two patches I and J can also actually 'see' each other (and can therefor transfer energy by radiation), a set of rays is shot from random points on I to random points on J, according to a uniform random distribution. The visibility factor boils down to the ratio of rays that hit J over the total amount of rays. The view factor $F_{I \to J}$ is multiplied by the visibility factor.

After having considered and tried all three approaches, the Riemann sum approximation came out as the most accurate and reliable method. Thus we

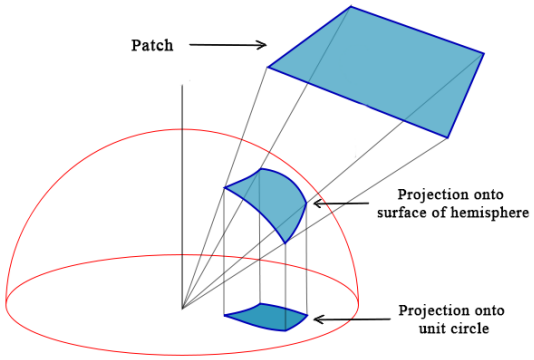


Figure 1: Nusselt analog

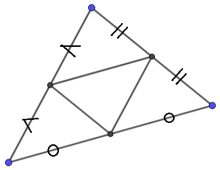


Figure 2: Subdivided triangle

used this approach to calculate the view factors in the program.

3.2 Matrix representation

For each patch, the view factors to all other patches are stored as a row in the radiosity matrix. This matrix describes the flow of energy through the scene:

$$\begin{bmatrix} 0 & F_{0\to1} & F_{0\to2} \\ F_{1\to0} & 0 & F_{1\to2} \\ F_{2\to0} & F_{2\to1} & 0 \end{bmatrix}$$

The radiosity matrix takes up a quadratic amount of space: n^2 for any scene with n triangles. For larger scenes, this grows to problematic numbers. A characteristic of the matrix is that for complex scenes, it is rather sparse: many more triangles are occluded to one another, than visible [2]. This provides a convenient opportunity to compress the matrix, by storing only the non-zero entries. We chose a sparse matrix representation that used this storage-efficient solution (see also Section 4).

3.3 Solving the equation

There are several ways to solve the linear system to obtain the radiosity values for each patch. We have chosen to implement the progressive approach: We iteratively calculate an approximate solution, based each time on the previous approximation [10]. Each iteration corresponds to a light bounce in the scene. We then formulate the radiosity in the scene as

$$B = E + \sum \rho K B \tag{5}$$

in which B, E and ρ are vectors containing the radiosity values, emission and reflectiveness, respectively, of the surface for each patch. K represents the radiosity matrix. With each pass, the system converges towards the solution, this convergence is ensured by the property of diagonal dominance in the radiosity matrix. The convergence of the calculations and the iterations of light bounces in the scene is illustrated in Figure 3.

Figure 3: The first three light bounces in a scene (from left to right)

3.4 Spectral rendering & fluorescence

To facilitate spectral rendering in radiosity, wavelengths are sampled discretely [7]. The larger the number of sampled wavelengths, the closer the approximation. When we assume that all wavelengths are decoupled, the solution for the radiosity at wavelength λ would be independent of the solution at some other $\grave{\lambda}$. So to find the radiosity values, we'd solve a set of linear equations for each individual λ.

However, as mentioned in Section 2.3, fluorescence breaks the assumption of decoupled wavelengths. To facilitate this, each material is modeled with a parameter M, which is a matrix that represents the transfer

of energy from one wavelength to another [9]. M is an m x m matrix for m number of samples. For a non-fluorescent material, this matrix has non-zero values only on the diagonal. For fluorescent materials, M is a triangular matrix, as the radiated energy is lower than the absorbed energy.

We adjust the solution to the linear equations (Section 3.3) to take fluorescence into account:

$$B = E + \rho \sum K \grave{M} B \tag{6}$$

in which B is an n x m matrix for n number of patches and m number of wavelength samples, E is an n x m matrix containing the emissive values, ρ an n x m matrix with the reflectivity of each sampled wavelength per patch, K is the n x n radiosity matrix, and $\grave{M}$ a 3d matrix of dimensions m x n x m, containing the matrix M for each patch. Take note that for each sampled wavelength, the dimension m is incremented, so with more samples comes a higher computational cost[2].

For simplification, we have chosen ignore time dependence between frames (as fluorescent materials cease to glow nearly immediately when the radiation source stops).

3.5 Rendering the model

To display the scene on screen, it is first traced with primary rays from a given camera and view direction. For each pixel, we obtain the patch index and barycentric coordinates. The pixels' color values are interpolated between the adjecent patches (see also Section 4). To render the image to screen, the spectral color model is converted to CIE XYZ color space [11] and consequently to RGB.

4 Implementation

The renderer is implemented in C++ and CUDA. What follows is a high level overview of the implementation of the program.

First, a Wavefront object, specified in a config file, is loaded into the program. It is required that the object is triangulated and sufficiently subdivided: the triangles are interpreted as the radiosity patches. From the materials file, corresponding to the object file, the reflective and emissive colors are read in RGB color space. From these RGB values, the spectral values are sampled, as described and provided by [5]. A triangle-adjacency lookup table is built for the geometry, in which the adjacent triangles for any triangle can be found. This facilitates interpolation over adjacent triangles when rendering.

Next, the radiosity matrix is constructed. Given the independent and repetitive nature of the calculations, this part was perfect for parallelization and has been implemented in CUDA. The ray shooting to determine the visibility of all patches (see Section 3.1), is parallelized too, by the NVIDIA OptiX framework. The view factors are stored in a sparse matrix data structure provided by the Eigen library. As the sparse matrix representation of the Eigen library was used to efficiently handle the large size, it seemed a natural consequence to apply Eigen as well for the consequtive calculations for solving the linear system. This would also be a convenient way to obtain vectorisation, by representing arrays of patch values per sampled wavelength as a matrix. However, the performance of Eigen relied so much on compiler specific optimization, that we opted to implement the vectorization ourselves.

After converging radiosity equations for the scene, the spectral values of each patch are converted to and cached in RGB color space, to save on the costly color conversions otherwise required with each frame update. To render the scene to screen, it is traced with primary rays in OptiX. Then, for each pixel, the color is interpolated over its corresponding triangle and the adjecent triangles. The resulting image is rendered as a texture on a plane using OpenGL.

5 Results

Figure 4 shows a selection of images that demonstrate some typical features of the renderer:

5.1 Wavelength sampling

For visual comparison, we show two renderings of the Cornell box with different sets of sampled wavelengths: Figures 4a and 4b. As described in Section 3.4, there is a tradeoff between the number of sam-

pled wavelengths and the computational costs. We started with a suggestion by [7], that a set of four samples can in most cases provide a good balance: 456.4, 490.9, 557.7, and 631.4 nm. Unfortunately, we found that these did not work very well for our case, as one can see in Figure 4a. Eventually, we found good results when sampling wavelengths between 350 and 650 nm with uniform intervals of 50 nm (7 samples in total), as seen in Figure 4b.

5.2 Colored light sources

Figures 4e and 4f demonstrate the application of colored light sources in the renderer. In Figure 4e, a model is lit from two sides, allowing the shadow on the right to contain more red, and the shadow on the left to contain more blue. Figure 4f illustrates a pink and a green light source, which are both partially obscured by a structure, allowing for colorful light patterns on the walls.

5.3 Fluorescence

As illustrated in Figure 4c and 4d, the emissiveness of the fluorescent materials differs from any other emissive surface, as the light is only emitted from where it can see the blacklight. It is literally a reflection of invisible light: only the reflection is visible. Observe the faint violet tint of the black light sources: The black light was sampled with an emissive peak at 350 nm, well outside of the visible range, however we also sampled a smaller peak at 400 nm, as is often found in the spectra of black light sources: their dark blue filters often leak a bit around 400nm, causing the violet glow.

5.4 Typical artefacts

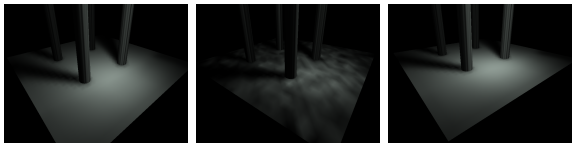
Most notable artifacts in the images are visible gouraud shading when the scene is not sufficiently subdivided. Objects can also appear as slightly floating when the scene is not sufficiently subdivided, as one can observe in Figure 4b. Adaptive subdivision [1] should be an appropriate solution to this.

6 Evaluation

The program has had many cycles of improvement. Multiple implementations have been tried and improved, providing a good opportunity to compare some of the approaches empirically.

6.1 View factor calculation

Three different approaches to calculating the view factor have been implemented, as described in Section 3.1. We provide a visual comparison in Figure 5. It is clear that the Riemann approximation gives a better result than the stochastic approach. The Nusselt analog produces a good result in this comparison, but starts giving skewed results when presented with not equally subdivided scenes.



(a) Riemann approximation. (b) Stochastic approach. (c) Nusselt analog.

Figure 5: The rendering of a temple using three different approaches

6.2 Implementation performance

All view factor calculations were initially implemented on the CPU. For any scene containing more detailed geometry (>7000 triangles), the program would take over two hours to compute the radiosity matrix for the scene. Implementing this part in CUDA resulted in a calculation time of 1.7 minutes for that same scene: a speedup of approximately 70. Table 1 shows a more detailed overview of the performance of the calculations when accelerated by CUDA and when executed on the CPU.

7 Future work

We suggest briefly three areas for future work on the project that we consider particularly exciting.

	CUDA	CPU
Setup	1,60	1,69
Calculating matrix:	112,00	8292,72
View factors	33,15	
Visibility factors	78,85	
Solving linear system	4,15	5,83
Rendering	0,21	0,33
Total time	229,97	8300,57

Table 1: Performance CUDA vs cpu implementation: Calculation times are in seconds, for a scene of 7712 triangles. Take note that the CPU implementation does not separate the calculations of the view factors from the visibility, hence only the total time for calculating the matrix is provided

7.1 Interactive materials editor

Currently, the spectral values for the materials are sampled from RGB values, which provides a convenient way to define materials: the materials can be inspected visually in an editor such as Blender, and then read from a materials file. However, it would be a great addition to have an interactive materials editor with the renderer itself. In the editor, the materials could be previewed and the scene updated (relatively) quickly. The manipulation of materials could be done by allowing manual adjustments to the curves that represent the reflective and emissive spectra of a material. The fluorescent color could be edited in a similar fashion. Alternatively, the fluorescent properties could also be edited with more detail for how and which wavelengths bend to what frequency, even enabling materials that might not be fully physically correct, but perhaps visually interesting.

7.2 Additional spectral light sources

The only light sources currently emitting UV light are the custom defined blacklight sources. Of course these are not the only sources of UV light in the real world. Actually, most emissive sources have complex emissive spectra with peaks, that are not as nice and smooth as the curves that we now use to sample from RGB to spectral values. Adding sources for, for example, sunlight would be a major addition to the program, as fluorescence also happens in daylight, which makes the color of a neon shirt pop so bright. These light sources would best be integrated together with the materials editor as descibed in Section 7.1, as that could provide a framework to assign such specific materials.

7.3 Simulation of X-ray fluorescence

Having set up a framework for spectral rendering with fluorescence, it would probably possible to visualize wavelengths outside of the visible spectrum and enable simulations of phenomena such as x-ray fluorescence. X-ray fluorescence is the emission of fluorescent X-rays from a material after being exposed to high-energy X-rays or gamma rays [6].

8 Conclusion

We have given an overview of the DaisyRiot renderer, which combines radiosity with spectral rendering and novelly supports fluorescent materials.

Our approach to radiosity uses a Riemann sum approximation to calculate the view factors, which we store in a sparse matrix representation. We chose an iterative approach to solving the radiosity equation and then modeled the solution to the equation for all wavelengths as a set of matrix operations that takes fluorescence into account.

We've parallelized parts of the implementation with CUDA and Optix. We saw that this provided us with a big speedup for setting up the matrix.

The results show that we found a good set of sampling wavelengths at 7 uniformly spaced intervals. We also see that fluorescent materials are emissive in a unique way and so bring the possibility for a new range of images. Typical artifacts appear when the mesh is not sufficiently subdivided, for which a solution exists.

We have enabled a new kind of material in radiosity and hope that brings new visualisations. This can possibly be made easier in the future with the roadmap we've laid out for the editing of materials and light sources.

References

[1] Michael F Cohen, Donald P Greenberg, David S Immel, and Philip J Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer graphics and Applications*, 6(3):26–35, 1986.

[2] Michael F Cohen and John R Wallace. *Radiosity and realistic image synthesis*. Elsevier, 2012.

[3] Andrew S Glassner. A model for fluorescence and phosphorescence. In *Photorealistic Rendering Techniques*, pages 60–70. Springer, 1995.

[4] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH computer graphics*, volume 18, pages 213–222. ACM, 1984.

[5] Wenzel Jakob and Johannes Hanika. A low-dimensional function space for efficient spectral upsampling. *Computer Graphics Forum (Proceedings of Eurographics)*, 38(2), March 2019.

[6] Ron Jenkins. *X-ray fluorescence spectrometry*, volume 265. John Wiley & Sons, 2012.

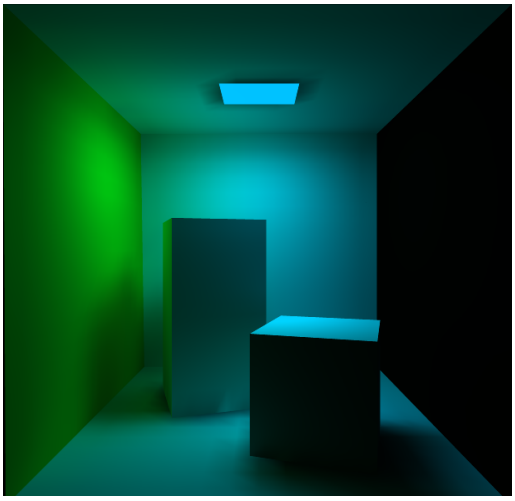
[7] G. W. Meyer. Color calculations for and perceptual assessment of computer graphic images. 1986.

[8] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4):66:1–66:13, July 2010.

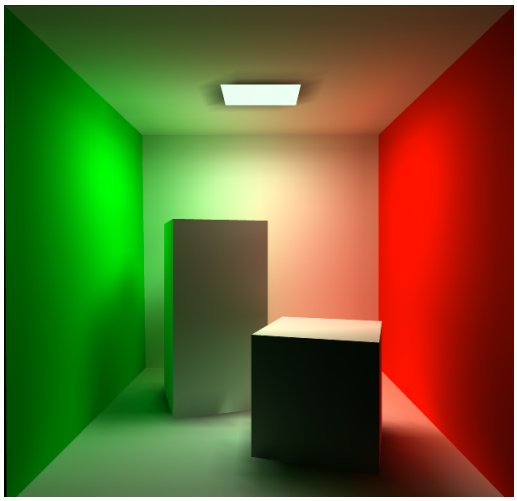
[9] Georgios Sakas, Peter Shirley, and Stefan Müller. *Photorealistic rendering techniques*. Springer Science & Business Media, 2012.

[10] Francois X Sillion, Claude Puech, et al. *Radiosity and global illumination*, volume 1, pages 36–38. Springer, 1994.

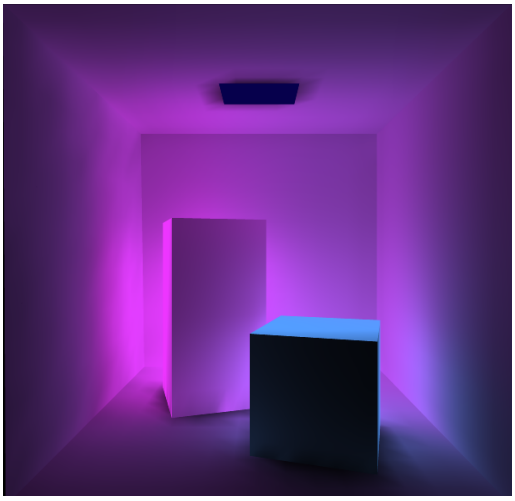
[11] Thomas Smith and John Guild. The cie colorimetric standards and their use. *Transactions of the optical society*, 33(3):73, 1931.



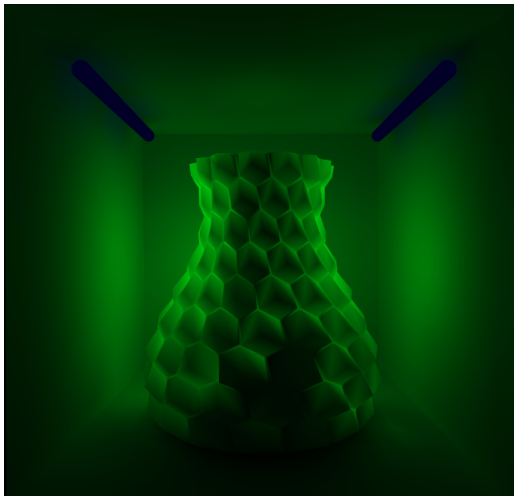
(a) Spectral rendering of the Cornell box, sampled with wavelengths as suggested by Meyer.



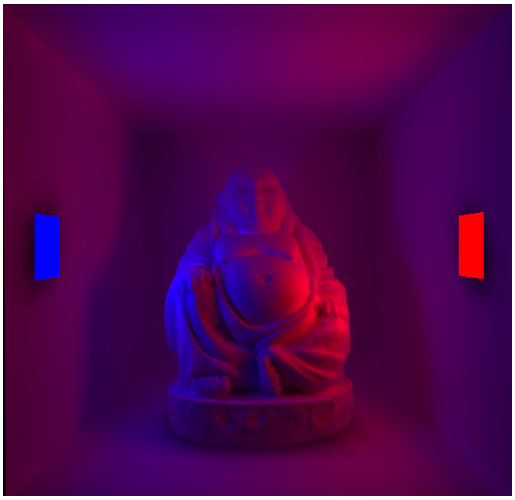
(b) Spectral rendering of the Cornell box with wavelengths sampled in 50 nm intervals between 350 and 650 nm.



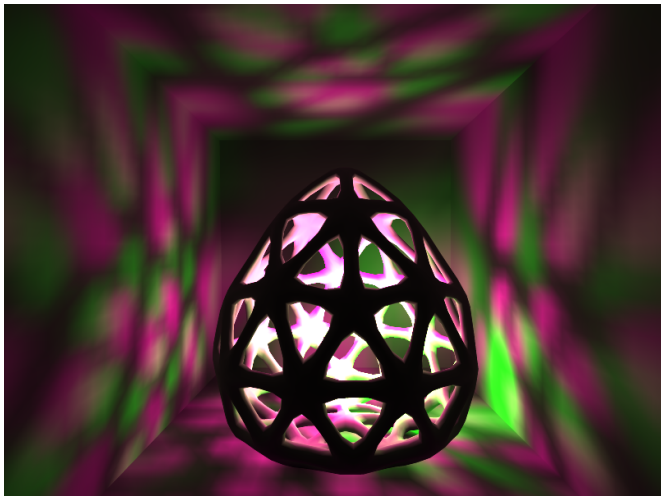
(c) Black light interpretation of the Cornell box with a black light and blue and pink fluorescent cubes.



(d) Black light scene with a green fluorescent object lit by two black light tubes.



(e) Scene with red and blue light sources.



(f) Scene with partially occluded pink and green lights.

Figure 4: A selection of results of the renderer.