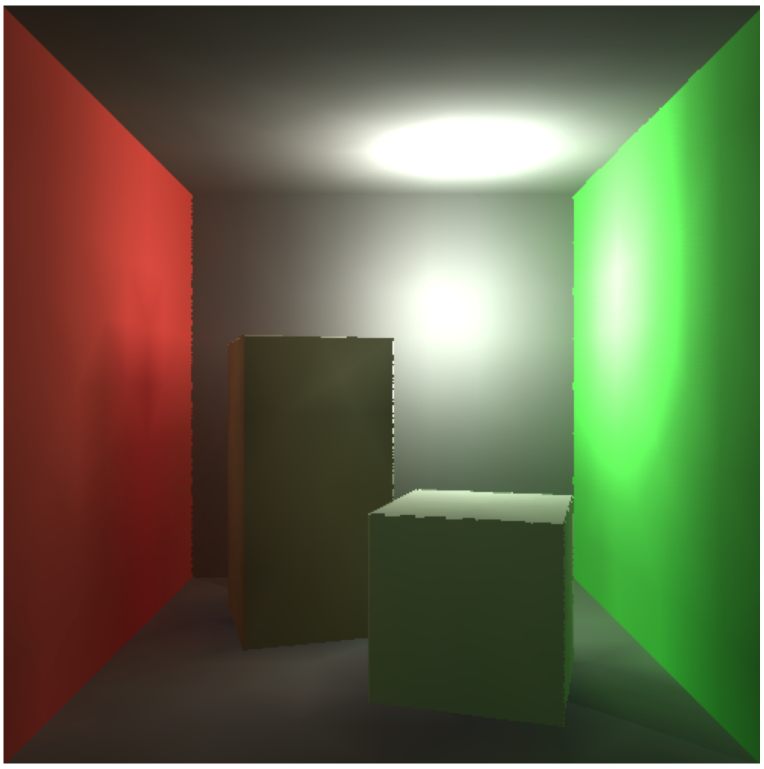
Radiosity in Realtime Rendering

DH2323 Project Report

Dan Hemgren, CDATE14



Contents

1 Abstract

This report presents a rasterizer implementation that utilizes radiosity to simulate indirect lighting in a realtime rendering enviroment. Exploiting the fact that scene geometry remains static per-frame, results from costly calculations can be stored and reused enabling realtime performance despite simulating several light bounces in the scene.

2 Introduction

Radiosity is a finite element method originally used to model heat transfer in thermodynamics[3]. As a model, radiosity is most often used in computer graphics to simulate global illumination. This is made possible by the assumption that all surfaces are Lambertian (meaning they reflect light diffusely) [1]. Central to the Radiosity model is the *diffuse rendering equation.*

$$B(\overline{x}) = B_e(\overline{x}) + \rho(\overline{x}) \int_{\overline{y} \in A} B(\overline{y}) F(\overline{x}, \overline{y}) da$$

This equation describes the radiosity at a given point $\overline{x}$ by taking into account the surfaces inherent light emission $B_e(\overline{x})$ and indirect light from all other points $\overline{y}$. A discretization of this formula yields the *radiosity equation.*

$$B_i = B_{i,e} + \rho_i \sum_j F_{ij} B_j$$

This discrete counterpart of the diffuse rendering equation is why the radiosity model is considered a finite element method. In practice, the discretization means that instead of calculating indirect light over continous surfaces, we now divide each surface into patches in which we consider the radiosity to be constant. The *form factor* F_{ij} is what ramps up the cost of the algorithm. It is a scalar value defined as

$$F_{ij} = \frac{1}{A_i} \int_{\overline{x} \in A_i} \int_{\overline{y} \in A_j} -\frac{\hat{n}_{\overline{x}} \cdot \hat{r}\hat{r} \cdot \hat{n}_{\overline{y}}}{\pi r^2} V(\overline{x}, \overline{y}) da_j da_i$$

where $\hat{n}$ is the normal at each point, r is the vector between $\overline{x}$ and $\overline{y}$, A is the area of the respective surface and V is a boolean value (1 if the two points are visible to eachother and else 0). In short, the form factor decides how much emission from patch j reaches patch i. Much like the original diffuse rendering equation, calculating the form factors across an entire surface is a costly operation and also needs discretization to fit well in a real-time environment.

These simplifications along with the assumption that every surface is ideally diffuse is what makes the radiosity model possible for real-time rendering. Furthermore, since efficient real-time rendering models for direct lighting already exists (including shadows), radiosity calculations can be relegated to only handle indirect lighting while other, less costly algorithms take care of the rest.

3 Implementation

The radiosity model is implemented using a number of simplifications

- Only static geometry is used. Subsequently, the form factors only need to be calculated once
- Instead of using a Monte Carlo-approach [3] to the visibility checks in form factor calculations, visibility is calculated by raytracing between the incircle centers of each triangle pair
- The resulting patches are interpolated using simple Gouraud-shading, foregoing more fancy interpolation schemes (i.e. reduction of Mach bands [2])
- No BSP-tree or similar structure is used for storing the patches
- Only a very simple direct lighting model is used without any real shadow mapping.
- The linear equation system of patch emissions will be solved using Gauss-Seidel relaxation

3.1 Surface division

As previously mentioned, surfaces need to be divided into *patches*. This is done in my implementation by simply dividing each triangle along the longest edge until a minimum edge length is reached.

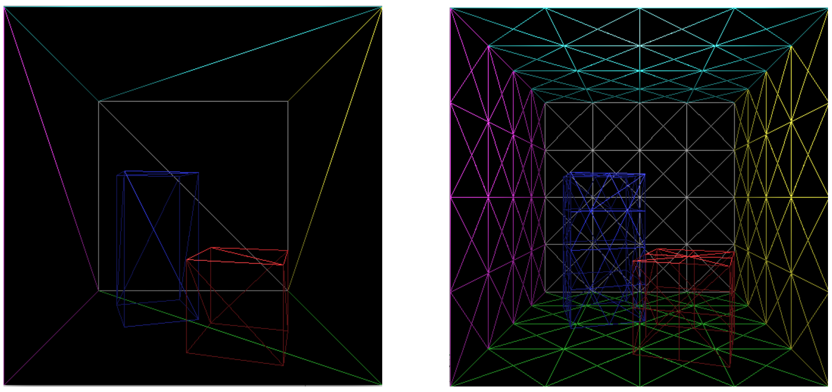

(b) Original model (c) Subdivided

Figure 1: Dividing the Cornell box into patches

During this step would be a good time to store the patches in some kind of hierarchal tree structure to more efficiently decide which vertexes affect each other during the radiosity calculations. This would probably reduce the form factor calculation time by quite a bit, even for the simple Cornell box model used. However, this is not implemented here due to time constraints, and the patches are simply stored linearly in memory.

3.2 Direct lighting

The direct light implemented is a per-pixel model taken directly from lab 3 in DH2323, with the small addition of an added shadow check via ray-tracing when calculating the patches initial emission. While the final result definitely could use shadow-mapping per-pixel, no such implementation is used.

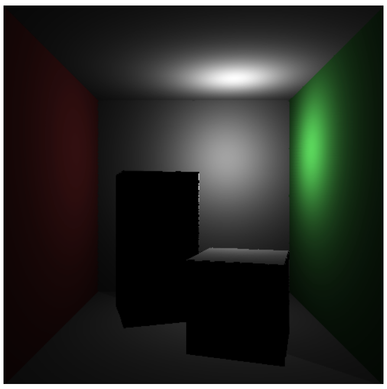


Figure 2: Direct light only. Notice how no light reaches the front of the boxes.

3.3 Radiosity

3.3.1 Calculating the Form Factors

In short, the Monte Carlo-discretization of F_{ij} is calculated as

$$F_{ij} \approx \frac{A_j}{n} \sum_{i=1}^{n} F(\overline{x}_i, \overline{y}_i)$$

for n random, uniformly distributed sample points $\overline{x}_i, \overline{y}_i$ at respective triangle. In my implementation I let $n = 1$, where the points are the incircle center of each triangle. This is quicker, but naturally much less accurate. Given that no hierarchal data structure is in place for the patches, this takes $\mathcal{O}(n^2)$ operations for n patches. However, the fact that

$$A_i F_{ij} = A_j F_{ji}$$

means that knowing F_{ij}, F_{ji} can easily be calculated, effectively halving the number of calculations [3].

3.3.2 Radiosity Per-Patch

```
begin
    OneIteration($\overline{B}^m$)
    for $i=1$ to $i=n$ do
        $B_i^m = E_i$
        for $j=1$ to $j=n$ do
            $B_i^m += \rho_i F_{ij} B_j^m$
    return $\overline{B}^m$;
```

Algorithm 1: Gauss-Seidel Relaxation [3]

Since all patches final emission depend on each other, solving the emission for each patch equals solving a system of linear equations. I originally wanted to do the inverse matrix solution mentioned by Burenius [3], but opted against it due to the libraries used (C++ standard library, GLM) having little built-in support for inversion of large matrices. Instead, I went with Gauss-Seidel relaxation, which is easily implemented using a simple for-loop (see Algorithm 1). This method could be interpreted as one light bounce being calculated per iteration until the system has converged. Only a few iterations are necessary, and for the model used the system was visibly converging already at five iterations. As Burenius points out [3], this means that iterations of relaxation can be carried out per-frame over the course of several frames to achieve real-time rendering of dynamic point lights (if re-using the form factors), giving only a slight lag to the radiosity of moving lights.

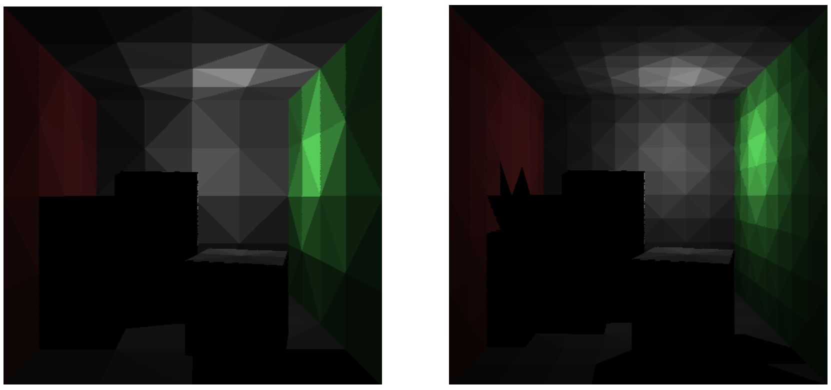
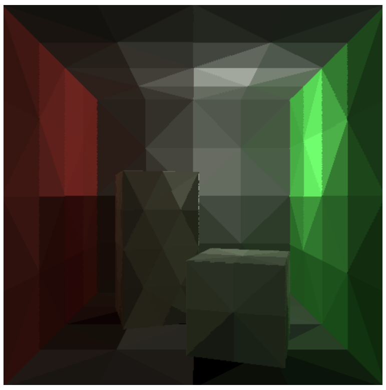

(a) 304 patches (b) 784 patches

Figure 3: Direct light per patch, no radiosity


(a) 304 patches

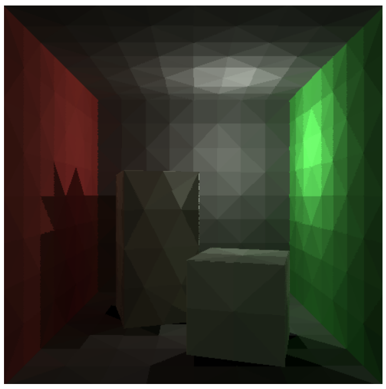

(b) 784 patches

Figure 4: Final indirect light per patch, 100 iterations of Gauss-Seidel

The starting emission values per patch pre-Gauss-Seidel can be seen in Figure 3. This can be interpreted as a single ray of light hitting each patch without any form of bouncing. The resulting light, meaning the final indirect light for each patch without per-pixel direct light, can be seen in Figure 4. Note that since each patch emission is dependent on all other patches, if a patch is given an erroneous value the error is reproduced to some degree in other patches. Thus, for a low enough patch count in my implementation (below three iterations of surface division on the model used), the resulting light is notably distorted, as pictured in Figure 5. A contributing factor to this can be the simplification of the $V(\overline{x}, \overline{y})$-function (using only the inline center as sample point).

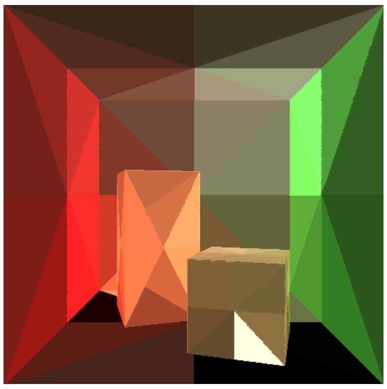


Figure 5: Too few patches resulting in saturation of red levels

4 Interpolation

To achieve a smooth transition between the patches, Gouraud-shading is applied to each patch. This is done by first averaging the color of each vertex based on the color of adjacent patches and then performing bilinear interpolation between the vertices on each triangle.

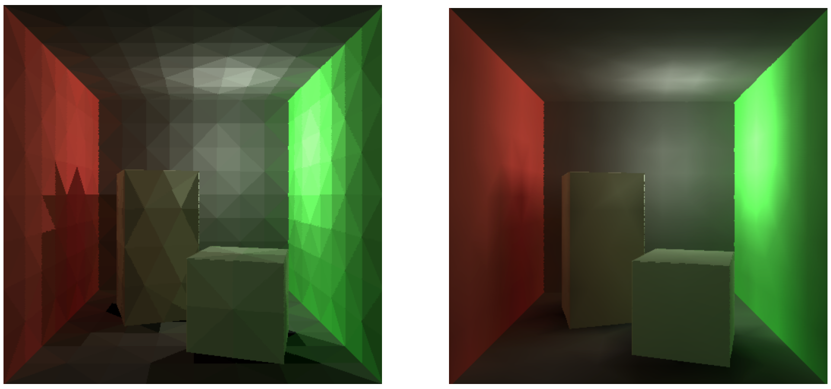

(a) 784 patches, no interpolation (b) 784 patches interpolated

Figure 6: Interpolation of patches

This form of interpolation can lead to Mach banding [2], where edges of adjacent triangles are perceived as being brighter resulting in a visible pattern across the surface. This can be seen across the back wall and ceiling in Figure 6b (not to be confused with the blocky shadow on the wall to the left which is due to the only shadow coming from radiosity coupled with low patch density). While there are ways to reduce this [2], it is not as noticeable in combination with per-pixel direct lighting. Another problem is found in the edge case where one vertex in a triangle is surrounded by patches that receives little or no light, such as the vertices that are underneath the boxes in the scene. This results in one vertex having a much darker color than the other two in the triangle during color averaging, which in turn results in the interpolation across that triangle not being correct in relation to the lighting of the room (see Figure 7). I have not found a good solution to this, but one possible solution could be to split the patches in a more dynamic way around connected geometry, and compensate by perhaps lowering the density over the larger surfaces (such as in the middle of each wall). This would in turn require a more accurate per-patch calculation of direct light to avoid the issues with low patch density mentioned previously.

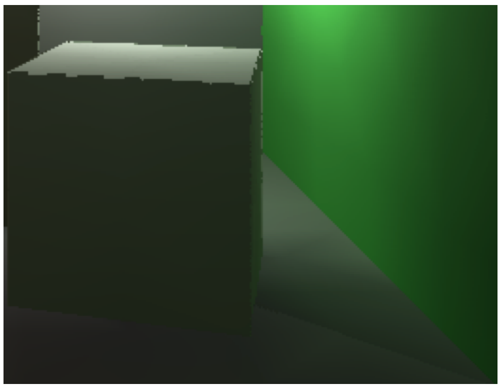
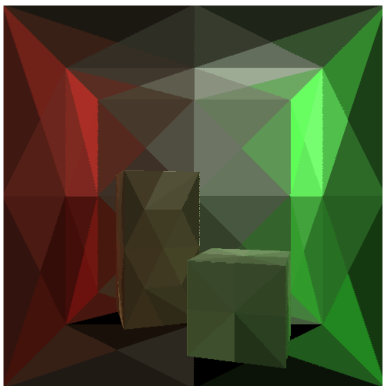


Figure 7: Error in interpolation

5 Results

Below are the results presented in order of amount of patches. 100 iterations of Gauss-Seidel relaxation was used for each patch density.


(a) Indirect light only

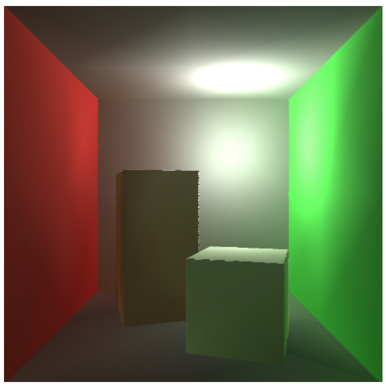
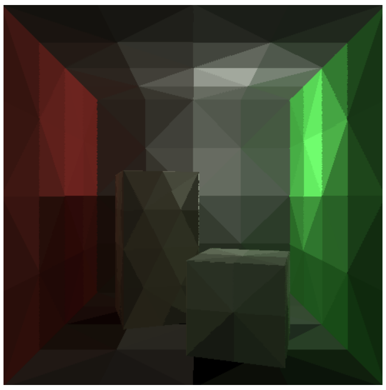

(b) Final result

Figure 8: 192 patches


(a) Indirect light only

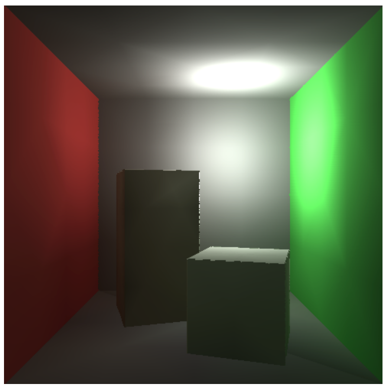
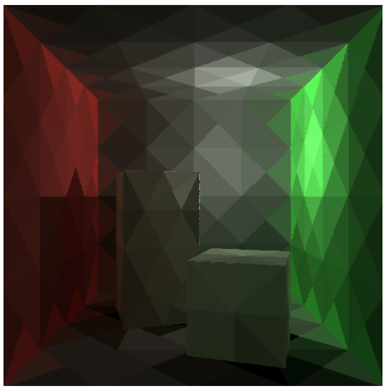

(b) Final result

Figure 9: 304 patches


(a) Indirect light only

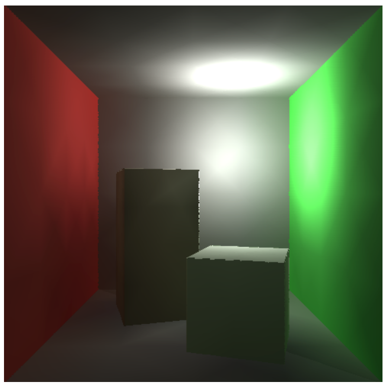
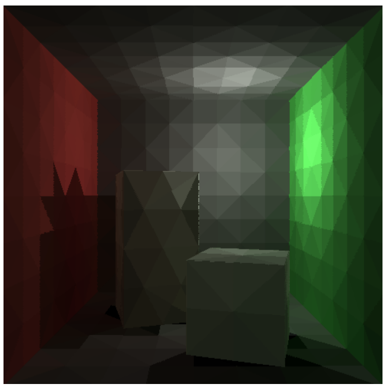

(b) Final result

Figure 10: 464 patches


(a) Indirect light only

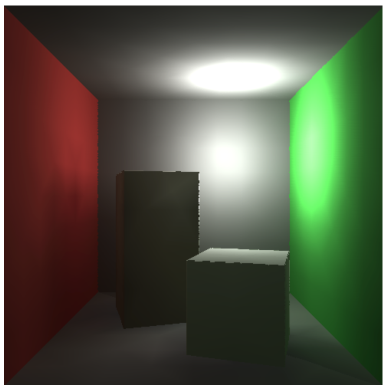
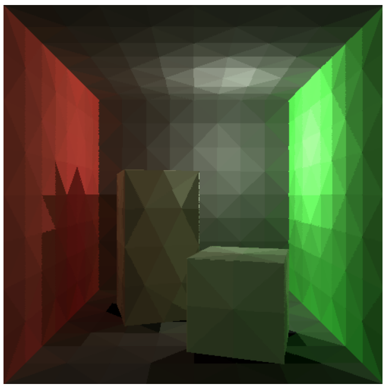

(b) Final result

Figure 11: 784 patches


(a) Indirect light only

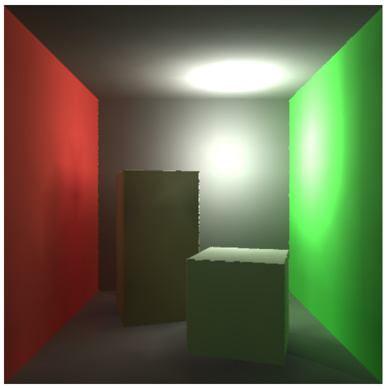

(b) Final result

Figure 12: 784 patches, higher albedo values

6 Discussion

It is clear that patch density has a big impact on the final result. While I was fairly certain this would be the case from the start, I more expected it to be a matter of jagged edges and noticeable patterns when instead it impacts the entire lighting of the scene. The increased albedo seen in Figure 12 as opposed to Figure 11 (mainly on the red and green walls) makes a subtle yet very noticeable improvement in the scene. Using similar albedo values with less patches would result in both the red and green walls glowing towards the back wall (as hinted in Figure 8 using 192 patches). This is in large due to individual patches in each corner getting a too bright first emission which is then pronounced with further indirect lighting. The problem would likely be somewhat remedied by taking the average initial direct light of each vertex of the patch instead of using the direct light of the incircle center for the whole patch. Had the Monte Carlo-approach been implemented for the $V(\overline{x}, \overline{y})$-function, it would perhaps also proven useful for calculating the initial direct light for each patch. Another possible solution to this problem could be a more dynamic approach to subdividing the surfaces, where more complex geometry would generate a higher density of patches. In this case, it could mean generating more patches towards the corners while the central areas of each wall could do with lesser patches. A caveat here would be that it would require extra work to discern whether the area would need higher patch density due to shadows being cast over it (as is the case with the left wall in the scene).

Since this implementation only handles a static scene, all calculations are done once during initialization and thus have no actual impact on performance besides patches adding to polygon complexity. I have because of this not studied the overhead or differences in frame times with radiosity enabled. This would be more relevant had the relaxations been carried out over several frames to enable dynamic lighting as mention by Burenius [3]. That said, the form factor calculations are in dire need of some form of hierarchical tree structure for the vertices. At six iterations of surface divison (ramping up the triangle count from 30 to 784 and thus $\frac{784^2}{2}$ calculations), deciding the form factors take minutes to perform on my Macbook even though the scene is about as simple as it gets. Culling, on the other hand, can not be used to slim down the calculations since all patches affect each other. Do note that my intention was not to make an optimal implementation, but rather to focus on the concept of radiosity and really understand how it works on a basic level. In this regard, I consider the project a success and am more than happy with the result.

7 Perceptual Study

After having participated in Jack Shabos perceptual study on groups and crowds, a few ideas came to mind. The obvious first choice for a perceptual study, which I mentioned in the project specification, would be to render a selection of 2 - 3 scenes, have each scene represented several times with different patch densities and amounts of relaxation and then have the user rate the perceived quality of the rendered light. The scenes would preferably be constructed in such a way that the lighting of the scene would take center stage, i.e. dramatic placement of point-lights, and then shown in a shuffled order to "mask" patch density. It would be important to make sure that each scene would render with the same frame times to avoid offsetting the users perception due to perceived lag. With enough data this study could be useful in finding a good middle ground between performance and correctness. Since it is already known that Gauss-Seidel relaxation converges quickly, perhaps only a few iterations will suffice for the end-user? Likewise, maybe the end-users perceive no difference in quality between 200 and 700 patches for a given scene, making the added overhead of additional patch calculations redundant? As an example, each user could answer questions rating from 1 to 10 such as these:

- Overall quality
- Is the light natural?
- Are the patches visible?
- Are other artifacts present?

If enough data is gathered using this study, one should be able to find some form of correlation between patch density, relaxation amount and perceived quality which then can be used to weigh quality and correctness against performance impact. This could further be studied in relation to various forms of geometry.

8 Project Summary

I spent most of the project reading Burenius thesis in detail, mapping out the radiosity equation in my mind and understanding how it works. Unlike other courses I've had so far at KTH, this course really gave me the freedom to try my hand at picking a subject, reading up on it and implementing it all on my own. While the literature used for this project was a thesis and not a research paper, I still feel more confident now to pick up a paper and make a working (but perhaps slow) implementation of it. I also find joy in the fact that I know myself where most of the improvements can be made, and that the state of my implementation is largely governed by time constraints more than lack of knowledge. As a follow up to the project, I'm thinking of converting my solution into using the GLU-library and OpenGL-shaders. In particular, I'd be interested in using some form of stencil shadows as a preprocessing step to somehow split surfaces into patches along direct light shadows. This would probably remedy the spots from shadows caused by low patch density without having to increase the scene complexity as much. Overall, I'm very pleased with the project and what I've learned.

References

[1] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis.* Ed. by Michael F. Cohen, and John R. Wallace. Boston: Morgan Kaufmann, 1993. ISBN: 978-0-12-178270-2. DOI: `https://doi.org/10.1016/B978-0-08-051567-0.50007-5`.

[2] Rui Bastos, Michael Goslin, and Hansong Zhang. "Efficient Radiosity Rendering Using Textures and Bicubic Reconstruction". In: *Proceedings of the 1997 Symposium on Interactive 3D Graphics.* I3D '97. Providence, Rhode Island, USA: ACM, 1997, 71–ff. ISBN: 0-89791-884-3. DOI: `10.1145/253284.253309`. URL: `http://doi.acm.org/10.1145/253284.253309`.

[3] Magnus Burenius. "Real-time radiosity : Real-time Global Illumination of a Static Scene with Dynamic Lights using Hierarchical Radiosity". 2009.