Experimenting with Radiosity

Zizhuang Yang
Computer Science
Carnegie Mellon University
zizhuang@andrew.cmu.edu

Matthew Kaemmerer
Computer Science
Carnegie Mellon University
mkaemmer@andrew.cmu.edu

Abstract

We go over our attempt to implement a distributable version of a global illumination model in radiosity. This paper will touch on the challenges encountered in our efforts to implement radiosity, to use CUDA on top of the implementation, and to figure out a schema that can be used to export the radiosity data to multiple viewers.

1 Introduction

Radiosity is one of the more commonly used approached towards global illumination. By emitting radiation from light sources and having the scene transfer radiation amongst other surfaces in the scene repeatedly until convergence, the radiosity algorithm mimics real-world phenomena to produce a scene. As with all global illumination algorithms, the data provided by the radiosity algorithm is useful for all viewpoints, making it a highly distributable process - it does the calculation, and any number of viewers may use the results of the calculation to render their clients' perspectives.

For the purposes of project 7, we tried to first implement radiosity for correctness, and then attempted to optimize its speed (though the use of the GPU amongst other methods) and display the viability of multiple viewers utilizing the same result output.

2 The Algorithm

We attempted to implement a matrix radiosity algorithm that seeks to transform the scene into a large linear system and then solve the incident energy within each patch decisively. This puts most of the computation into the pre-computation phase, which is ideal for distributive algorithms where viewers may come in at any time - a server can have done the pre-computation long ago and be able to serve information to a new client immediately. However, the pre-computation time is overwhelming with increasing scene complexity, as is quadratic to the number of objects in the scene.

The general procedure of radiosity is as follows: First, you partition every object in a scene into small 'patches', which will be the fundamental unit of transmission in the radiosity algorithm. These patches span anything, from object planes to light sources themselves. Once that is done, a large **form factor** matrix is computed. Form factors are essentially coefficients that express the likelihood for any given patch in the scene to transmit radiation to another patch. For instance, two patches directly facing each other have a much higher form factor than two patches on the same wall or two patches with another patch directly between them. This is the heavy pre-computation part of the algorithm, and makes up the quadratic runtime of the algorithm.

Then, we solve for $M\mathbf{x} = \mathbf{b}$ where M is the form factor matrix, $\mathbf{x}$ is our unknown, and $\mathbf{b}$ is the emission state of each patch. The emission state is positive for light sources and 0 otherwise. Solving for $\mathbf{x}$ then provides a convergence solution to the incident energy of each patch in the scene. In our algorithm, this is currently done through Jacobi iterations. Though the number of iterations used to get sufficiently close to convergence is high, the processing time is nonetheless insignificant compared to the time expended by form factor calculations.

Finally, once the linear system is solved to obtain the incident energy for every patch, the resultant data can used for rendering from any viewpoint.

3 The Challenges

At first, we used raycasting as a rendering method, which was prohibitively slow as each ray cast had to check against every patch created in the scene. This method was quickly switched out for the usage of OpenGL to render the planes as textures instead, which allowed us to not be reliant on the number of pixels rendered and instead be linear on the number of patches in the scene.

The ongoing challenge for our implementation is the quadratic runtime of the form factor calculation, which heavily dominates the radiosity runtime. We have not implemented a solution to this challenge yet, though the usage of spatial data structures is on the table.

As always, some the challenges we've faced up to now consisted of simply hunting down bugs when seeing unsatisfying results (for instance, a scene that's wholly dark due to coefficients being 0 or wholly light due to coefficients being `nan`). As always, liberal use of `printf` and some of `gdb` helped us find out where uninitialized variables or incorrect indexing led us astray.

The biggest challenge was really dealing with the time constraint. Due to our respective Carnival obligations and following exams, we basically had to learn and implement radiosity within a span of 7 hours. Oops.

4 Measurement Results

As of now, our code's pre-computation time for a scene with 600 polygons over several runs averaged around 46 seconds, which is suboptimal. The computation of the form factor matrix takes up almost all of this time, standing out as the clear performance bottleneck. Note here that

However, after this computation is complete, the rendering speed is not reliant on further calculations - all of the patches contain all the information necessary for scene correctness, so the rest of the work can be done by clients in real time. Assuming that the client keeps data of where the objects are, the only data that the client requires is the resultant incident energy (helpfully stored as a color), which involves transmission of a mere four bytes for every object in the scene. Otherwise, the client must be sent the entire rendered screen. For this reason, this solution can scale to as many clients as bandwidth allows - the number of agents does not affect the computation complexity beyond any avatars or objects that those agents may represent.

5 Future Work

Since this prototype was quickly hashed out, there is a lot to be done. Here are some notes on steps forward that can be taken.

Textures & Smoothing

Due to the nature of the radiosity algorithm, it is generally infeasible to compute radiosity per-pixel (and indeed, without knowing the camera position ahead of time, this is truly impossible). As a compromise, we subdivide all surfaces in the scene into small patches and compute energy transfer per patch. One improvement we plan to make on this strategy is to store radiosity information to textures and blur each of these textures to create a smooth result.

Multiple Cameras

In order to evaluate the benefits one could get from computing radiosity on the cloud, we will implement a second (and possibly third or fourth) camera. This will help showcase the view-independent nature of the calculation.

Dynamic Scenes

With a static scene, it is unneccesary to recompute radiosity per-frame. Instead, we can precompute radiosity and continue to use a stored result. We will generalize this to scenes with moving light-sources and recompute as necessary.