Numerical Computing 2014 Final Project: 2D Radiosity

Maximillian Dumas

May 21, 2014

Abstract

The field of computer graphics has been on a relentless pursuit towards graphical realism ever since its creation, with increasing horsepower and evolution of computer science and mathematical methods leading to ever-increasing fidelity and more ambitious immitations of reality. A significant subset of computer graphics algorithms are devoted to the subject of realistic lighting simulation, one of these algorithms being the radiosity algorithm. In the interest of pursuing realism, most of these algorithms are fundamentally based in three-dimensional space, but two-dimensional computer graphics are still widely used, even when simulating environments. In this article I propose an adaptation of the radiosity algorithm to suit this purpose. After introducing the topic of radiosity, I discuss the issues with the basic algorithm that make it unsuitable for immediate adaptation to two-dimensional problems, as well as why it is still a valid basis for a lighting simulation in a 2D abstract space. I explain the data structures that are required to represent the two-dimensional scene, and how they are analogous to those required by the standard algorithm. The various techniques and methods I attempted in implementing the various aspects of the algorithm comprise the majority of this article, concluding with analysis of the results of the completed implementation I created, and discussion of where further improvements in future implementations could be made.

1 Introduction

1.1 Radiosity Overview

The concept and mathematics of radiosity were originally developed in the 1950s in heat transfer engineering, as a way to measure the propagation of radiation throughout a space. In 1986 Greenberg, Cohen, and Torrance adapted the concept to create an algorithm which models the energy equilibrium balance within an enclosure, treating light as purely a radiant energy source.[2] The radiosity algorithm in its pure form accurately represents the perfectly diffuse reflection of light throughout a space. Diffuse in this case means that any and all light energy leaving a surface is equally distributed in all directions. The result is that, unlike more advanced methods such as ray-tracing, the radiosity algorithm is incapable of modeling light effects like caustics and non-diffuse

reflections. Additionally, due to the geometric requirements imposed on the scene by the algorithm, it is unsuitable for scenes which contain functional geometry such as NURBS. Also for these reasons radiosity is a poor choice for simulating point lights, spot lights, and directional lights. The algorithm does excel at representing the lighting of scenes which contain objects with matte, opaque surfaces which can be represented explicitly using polygons, and which exist as the exclusive sources of area light.

1.2 Representation

Representation in this case refers to the structure imposed on the scene by the algorithm. The primary operation performed on the scene by the algorithm is the subdivision of all surfaces in the scene into 'patches'. The algorithm divides the surfaces in the scene up into N pieces. For a patch i, its view-factor with all patches (including itself) is represented by the row of the N by N matrix $\boldsymbol{F}$, F_i. The view-factor (also known as a form-factor) is a unitless quantity expressing what fraction of the light leaving one patch will reach another. It is a reciprocal quantity; that is, $F_{ij} = F_{ji}$, so $\boldsymbol{F}$ is symmetrical. Additionally, this means that $F_{ii} = 1$, as any patch i is always completely visible to itself, and so $\boldsymbol{F}$ has 1's all down its diagonal.

The equation for the view-factor between two patches i and j is formally represented in the unoccluded case as the following:[4]

$$F_{ij} = F_{ji} = \frac{1}{A_i} \int_{x \in i} \int_{y \in j} \frac{\cos\theta \cos\theta'}{\pi r^2} dy dx \tag{1}$$

This is the double integral over all points $x \in i$ and $y \in j$ with respect to x and y, scaled according to the inverse square of their distance between one another, and the cosine products of the angles between their planar normals and the line segment connecting the two planes, finally divided by the area of the casting patch.

Although $\boldsymbol{F}$ is symmetric, and so only the lower triangular portion of it needs to be explicitly found, calculating the view-factor is still one of the most computationally expensive parts of the radiosity algorithm, and there have been many innovations in the methods for calculating the view factors since 1986. The Greenberg et al. paper calculates $\boldsymbol{F}$ using the hemicube method, which at a high level involves aggregating the incident light onto a patch by projecting a hemicube around that patch, and finding the average radiosity of the scene as viewed outward from the perspective of each face of the hemicube. Many practical implementations today do not explicitly calculate $\boldsymbol{F}$ due to the expense, and instead approximate the radiosity simply by using raycasting to sample the radiosities of the patches surrounding a given patch. I chose to use a hybrid method whereby I explicitly calculate $\boldsymbol{F}$ using raycasting, due to restrictions on the representation of my scene. For further details see the Scene Representation section.

1.3 Formulation

The discrete radiosity equation is originally described as the following:[2]

$$\beta_i = \underbrace{\varepsilon_i}_{\text{emitted}} + \underbrace{\rho_i \sum_{j=1}^{N} \beta_j F_{ij}}_{\text{reflected}} \quad \text{for } i = 1 \text{ to } N \tag{2}$$

Note: To avoid confusion, note that I use the Greek letters β and ε instead of B and E, contrary to what the Greenberg et al. paper and most sources use. I do this to avoid confusion with later variables involved in defining the scene. See the section on Scene Representation.

$\boldsymbol{\beta}$, $\boldsymbol{\varepsilon}$, and $\boldsymbol{\rho}$ are N-length vectors representing the radiosity, emissivity, and reflectivity values of the patches in the scene, respectively. The emissivity of a patch refers to the rate at which that surface is emitting light. Patches with non-zero emissivity are the only true sources of light in a scene. A patch's reflectivity refers to the fraction of light incident on that surface which is reflected back from the surface. Finally, the radiosity of a patch represents the total sum of the rates of emission and reflection, as expresssed in (2).

Note that (2) is an implicit equation, as it contains two separate terms of β. It still needs to be solved explicitly. There are several methods of doing this. The method which I used is explained in the Computing Radiosity section.

2 Adaptation of the Radiosity Algorithm

The radiosity algorithm is fundamentally attached to three-dimensional space, and so requires significant adaptation to be usable as a concept in two dimensions. The largest differences, of course, result from the difference in the structure of the scene in 3D versus 2D.

2.1 Representation

For purposes of illustration, let us consider a simple room to be our scene, with a flat floor, four walls, and a ceiling.

Representation of this scene in 2D is logically different than in 3D, because the most traditional way to visualize a 2D scene is as an orthogonal projection along an axis of some scene that might otherwise be three-dimensional, i.e. a 'side' view or a 'top' view. Thus a camera in a 2D space is necessarily viewing from 'outside' of an open scene; to visualize this room from a top view, for example, we would need to remove the roof. This creates two significant problems with the traditional radiosity algorithm: First is that the scene is open, so establishing an 'energy equilibrium' is no longer possible. Secondly, when this viewpoint is considered, there are only two kinds of surfaces - those that have normal vectors orthogonal to the camera, and those that face the camera directly. I refer to the former as 'edges' and the latter as 'spaces'.

The result is that the surfaces which cast light energy and the surfaces which recieve visible light become disjoint sets. Any light cast by a space will never hit any other space, as all the spaces are on a plane together and there is no ceiling to bounce that cast light off of. The light will only hit edges, which from a 2D perspective appear only as lines and have no visible surface from which to show the light that they recieved.

It quickly becomes apparent that edges have to be the subjects of the radiosity equation in this context. They are capable of facing one another and bouncing light to one another, and light emitted diffusely from them is capable of hitting our spaces, which are visible. While this has the benefit of drastically shrinking the set of surfaces we have to consider when calculating radiosity, a second stage of the algorithm also becomes necessary. In the traditional radiosity algorithm, calculating the radiosity for each energy-holding surface also calculates the illumination of each visible surface, as they are one in the same. In this algorithm, however, it becomes necessary to calculate the radiosity of the edges, and then disperse the energy from the edges onto the spaces.

2.2 Formulation

Due to the nature of the vectors formulating the radiosity equation in (2), little adaptation is required to this equation. In my solution I chose to reformulate it into a more concise vector format in the standard form for a linear system of equations:

$$(\boldsymbol{I} - \boldsymbol{\rho F})\boldsymbol{\beta} = \boldsymbol{\epsilon} \tag{3}$$

Note that $\boldsymbol{\rho F}$ is still a row-wise multiplication, so each ρ_i is multiplied to each element of the row F_i. Also significant is that this has become a linear system of equations with an explicit solution. Due to the nature of $\boldsymbol{F}$, the left-hand side matrix in (3) can be shown to be symmetric positive definite, and as a result we are free to use any number of solution methods, such as LU-factorization, Jacobi iteration, Gauss-Seidel, etc.

3 Implementation

3.1 Scene Representation

For my implementation of the radiosity algorithm I chose to primarily represent the scene using two data structures: The spaces of the scene are represented with a grid, and a the edges are represented with a graph.

The space and boundaries of the scene, which have constant square dimensions, are discretized into a regular grid of M by M spaces. This grid is referred to by the M^2-length vector $\boldsymbol{L}$, and each element of $\boldsymbol{L}$ contains information regarding the color of the grid space as 8-bit integer red, green, and blue components. $\boldsymbol{L}$ also contains an illumination factor for each element, by which the color of that space is scaled when displayed to reflect how much light is hitting it. The color values are predetermined by the user and the illumination factor is initalized to zero.

The geometry within in the scene is represented by a standard undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges which compose the graph. E is initially defined by the predetermined vertices in V, but edges are

subdivided further so that they do not exceed a maximum length. These subdivided edges are the one-dimensional analogue to the original algorithm's two-dimensional patched surfaces. Additional vertices are added as needed while the edges are subdivided. There are a total of N edges *after* subdivision. Edges are right-facing, meaning that for an edge (v, w), the normal of the edge points to the right when facing towards vertex w from vertex v. While an edge occludes light going in both directions through it, it will only reflect light approaching from its right side, and will only emit light from its right side. Thus to construct a solid wall, one edge is not sufficient; a rectangle of four edges with outward-facing normals is required.

E has associated vectors of magnitude N for the following: $\boldsymbol{\rho}$ represents the reflectivity of the edges, $\boldsymbol{\epsilon}$ represents the emissivity of the edges, and finally $\boldsymbol{\beta}$ represents the radiosity of the edges. The vectors are structured so that any edge i in E has reflectivity ρ_i, emissivity ϵ_i, and radiosity β_i. $\boldsymbol{\rho}$ and $\boldsymbol{\epsilon}$ are predefined. $\boldsymbol{\beta}$ is our unknown.

Due to the disparity in representation of two kinds of surfaces in the scene, I chose to have two view-factor matrices. The N by N matrix $\boldsymbol{F}$ is our edge-edge view-factor matrix, while the N by M^2 matrix $\boldsymbol{H}$ is our edge-space view-factor matrix.

For a given edge pair (i, j), I calculate F_{ij} in the following way:

$$F_{ij} = F_{ji} = \begin{cases} 1 & \text{if } i = j \\ \dfrac{A\omega}{A + r^2}\text{Vis}(i, j) & \text{if } i \neq j \text{ and } \omega > 0 \\ 0 & \text{if } \omega \leq 0 \end{cases} \tag{4}$$

$\omega = \cos\theta_i \cos\theta_j$, where θ_i is the angle between the normal vector of i and the line segment connecting the centers of i and j, and θ_j is the angle between the normal vector of j and the line segment connection the centers of j and i. This scales the view factors according to their respective angles to one another such that if they have parallel normals, their view factor with one another is zero.

r is the distance between the centers of the two edges.

A is the attenutation constant, which is the square of the distance at which the light energy emitted is half the magnitude as at the source. So for $A = 1$, two edges facing one another directly at distance 1 would each recieve half of the other's light energy.

$\text{Vis}(i, j)$ is the visibility function, which returns a scalar between 0 and 1 representing the fraction of i that is occluded by obstacles from j. This was accomplished in my implementation by uniformly sampling a set number of points on each edge, and organizing them into pairs by the order they were sampled. These points are joined with line segments, and then each of these segments are tested against all other edges u where $u \neq i$ and $u \neq j$ to see if the segment intersects u. If there is an intersection, that segment is considered blocked by an obstacle. Vis returns the fraction how many line segments have unobstructed paths between i and j versus how many total line segments there were.

H_{ij} is calculated in a similar way, where $i \in E$ and $j \in \boldsymbol{L}$:

$$F_{ij} = F_{ji} = \begin{cases} 1 & \text{if } i = j \\ \dfrac{A\cos\theta}{A + r^2}\text{Vis}(i, j) & \text{if } i \neq j \text{ and } \cos\theta > 0 \\ 0 & \text{if } \cos\theta \leq 0 \end{cases} \tag{5}$$

Since a space has no angular quantity, we need only calculate the cosine of the angle connecting i's center to j's center, $\cos\theta$, instead of the cosine product ω. Additionally, $\text{Vis}(i, j)$ is simpler, as we need only sample points along edge i and ensure their visibility to the center of space j. As we decrease the size of the spaces in the scene, this approximation becomes more and more accurate.

These two view-factor calculation functions are a blend of two existing functions: The $\frac{A}{A+r^2}$ term is a standard quadratic attentuation function for light sources.[3] The cosine product comes from our unobstructed view-factor equation in (1). The Vis function arose as a way to cleanly express the factor of occlusion on radiosity. The method of line intersection used for Vis is a derivative of the method proposed by Ronald Goldman in Graphics Gems.[1]

3.2 Computing Radiosity

Computing the actual radiosity is very simple once the view-factors have been calculated. We simply solve equation (3) for $\boldsymbol{\beta}$. As mentioned before, $(\boldsymbol{I} - \boldsymbol{\rho F})$ is a symmetric positive definite matrix, and is well-behaved for reasonable values of $\boldsymbol{\rho}$. Additionally, the small number of edges that can accurately define a scene means that the system of linear equations to solve is generally quite small. As a result, we are free to utilize any reasonably effective solution method to find our radiosity. I chose to use the LDLT method, as it takes advantage of the properties of the matrix, and is fast and accurate.

3.3 Illuminating the Scene

The illumination of a space j is also accomplished simply using the following equation:

$$I_j = \sum_{i=1}^{N} H_{ij}\beta_i \tag{6}$$

A space's illumination is the sum of the radiosities of all edges in the scene, scaled by the view-factors between the edges and that space. It becomes evident how large of a role the view-factors play in this equation: H_{ij} handles all changes in the illumination based on distance from the source, angle to the source, and occlusion by other obstacles, so that finding the actual illumination of any space is just a simple sum.

3.4 Displaying the Illuminated Scene

I use the Simple DirectMedia Layer library, or SDL for short, to display the results of the radiosity algorithm (see http://www.libsdl.org/). My implementation of the algorithm is not at all tied to the SDL method of displaying it, except perhaps in the way that color is stored for grid spaces, though this format is largely standard.

Display of the grid, at a high level, simply involves iterating through every element of $\boldsymbol{L}$ as a grid. I draw a filled rectangle of color given by L_j, scaled according to the illumination of L_j. The resulting color components are then capped at a max value of 255 to prevent artifacts resulting from 'blowout', where a given gridspace is lit by so

many sources or with such intensity that it appears completely white. The rectangle is of the unit grid size, which defined as the width of the screen divided by M, and has coordinates $(x * gridSize, y * gridSize)$, where $j = xM + y$.

Edges are displayed simply by iterating through every edge $i = (u, v)$ and drawing a line, shaded according to the radiosity β_i, from u to v.

4 Analysis

Level initialization, which involves the subdivision of the predefined edges into N smaller chunks, typically took about 0.1% of calculation time operating at approximately $O(|V|)$ time, with a large constant depending on the distance between vertex pairs. Graph initialization and vector initialization were instantaneous in all but extreme cases, as they are mostly trival operations operating in $O(N)$ time. Edge-edge view-factor initialization, despite operating in $O(N^3)$ time, was also instantaneous for most reasonable values of N, simply due to the fact that N is so relatively small.

Radiosity calculation for all scenes I tested was effectively instantaneous, taking at most a dozen or so milliseconds with nearly all solution methods I tested. With solution methods taking advantage of the properties of the matrix, calculation was faster than my tools could measure (<1 millisecond).

What took the vast majority of time, when calculated at full resolution with grid unit sizes of 1 pixel, was edge-space view-factor calculation. Tests had this step taking approximately 97% of total calculation time, at multiple orders of magnitude more expensive than the previous steps. This is because this calculation takes $O(N^2M^2)$ in its current implementation. As an example, in a small scene on a 640 pixel wide window with 12 edges, there are $640^2 = 409600$ spaces, which means $\boldsymbol{H}$ has $NM^2 = 4915200$ entries, and calculating $\boldsymbol{H}$ takes on the order of 60 million calculations. Compare this to calculating $\boldsymbol{F}$, which only takes on the order of 1700 calculations, and it is little wonder why edge-space view-factor calculation takes such a large proportion of the time.

That calculating $\boldsymbol{H}$ would be so expensive became apparent early on, and most of my efforts in this project were spent trying to minimize that cost. I experimented with a number of methods: I tried to use geometric reasoning to be able to immediately rule out large swaths of spaces from being visible for any given edge, but none of my attempts scaled correctly when trying to obtain higher resolution results. I tried using selective sampling methods based around the spaces themselves, where they would simply sample from the nearest, brightest edges, or just from any edges that uniformly dispersed rays intersected with, but these methods were not much faster, nor were they anywhere near as accurate as my current method. The largest problem with sampling methods is that every grid space must have some form of calculation performed on it, otherwise it simply remains dark; sampling selectively from edges to spaces was thus useless, unless I sampled all of them. The solution I came up with was the most effective I could create, which is able to quickly rule out many grid spaces using the cosine check, but it on average still needs to sample most spaces with rays. My solution does scale reasonably well, and most importantly it is very accurate as the number of rays cast against spaces per edge is increased. It produced well-gradiated, accurate

shadows with a high degree of fidelity, and correctly gives the appearance of a merged, cohesive visual model.

5 Conclusion

My adaptation of the radiosity algorithm from three-dimensional space to two-dimensional space was a success. It creates a visually accurate representation of the radiosity algorithm using the same fundamental equation but is purely two-dimensional in all of its internal and external representations. All portions of the algorithm scale very well, except for the edge-space view-factor calculations, and the solutions are stable for reasonable starting values.

Future work on this project would involve further optimization of view-factor calculation. I would look into using some sort of spatial hash, such a quadtree, to make checks for ray intersections faster and to allow stronger spatial reasoning about large occluded regions of space. I would also look into creating systems for more flexible scene definitions - right now all vertices and edges have to be hard-coded. Loading them from an image or at least a text file would be far more convenient. Additionally I would like to expand this algorithm to work on colored lights, which would essentially involve running the algorithm once per color-component, and I would like to re-arrange the algorithm so that the reflectivity, radiosity, and emissivity of the edges can be modified in real-time. Calculating the radiosity for changes in these values does not require recalculating view-factors, and so can be done nearly instantaneously. This would allow for interesting effects like animated lights that pulsate, or can be turned on and off.

PLEASE SEE THE ZIP FILE INCLUDED WITH THIS PAPER FOR THE ENTIRE PROJECT CODE AND EXAMPLE SCREENSHOTS.

References

[1] Goldman, Ronald. “Intersection of Two Lines in Three-Space.” *Graphics Gems*. By Andrew S. Glassner. Boston: Academic, 1990. 304. Print.

[2] Greenberg, Donald P., Michael F. Cohen, and Kenneth E. Torrance. ”Radiosity: A Method for Computing Global Illumination.” *The Visual Computer* 2.5 (1986): 291-97. JSTOR. Web. 20 Apr. 2014.

[3] ”Light Attentuation.” *BlenderWiki*. The Blender Foundation, 17 July 2013. Web. 21 Apr. 2014.

[4] Sillion, Francois X., and Claude Puech. *Radiosity and Global Illumination*. San Francisco, CA: Morgan Kaufmann, 1994. Print.