Radiosity Assignment

Seth Rader
12/10/16

Note: This project is not complete. Some of this information is written as if this was a complete system. Details about the project in its current state is available in the post mortem.

1. Table of Contents

2. Description

This project simulates the effect of radiosity of light, which is the repeated reflection of light off diffuse surfaces, and forming the light simulation part of the image synthesis pipeline. The objective was to take a standard 3D scene, the “Cornell Box,” apply radiosity to it given information about the reflectance (the color and intensity of light that surfaces reflect) and initial exitance (light given off from a surface acting as a light source) of the objects in the scene. After lighting, the scene is rendered as an image as an additional stage.

3. Approach

To perform this job, I used two programs: one for radiosity calculations that takes a scene file and adds lighting information to it, and another that draws the scene that was processed by the radiosity program. Radiosity is performed using the “hemicube method,” wherein a scene composed of quadrilateral surfaces called “patches” is lit by repeatedly distributing light from each patch to other patches. Each patch uses a hemicube to determine how rays of light are cast from each patch, and how much of the total emitted light is distributed among the rays. Ray casting is performed using my own existing ray tracer, using pixel “colors” as array indices. For input of “raw” scene files, text files are used that contain information about each patches reflectance (color and intensity with which surfaces diffusely reflect light) and initial exitance (amount of light that is initially cast from surfaces).

4. Radiosity Program

rad *[scene file]*

Performs radiosity on raw scene with reflectance and initial exitance data, and outputs scene file with lit surfaces. This is represented as the first arrow in Fig. 1.

a. Input Format

The input is a plain-text file that consists of a two-line header, and a pair of lines for each patch. The first line contains camera information: world coordinates of the camera, a point for the camera to look at, and an “up” vector. The second line contains the dimensions of the image that is produced, hemicube resolution, and the number of lighting passes.

Camera position:	3D float vector
Look-at position:	3D float vector
Up vector:	3D float vector
Image dimensions (width, height):	2 ints
Hemicube dimensions (width, height):	2 ints
Number of lighting passes:	int

After that there are pairs of lines for the patches. The first line specifies the vertices for a quadrilateral patch in counter-clockwise order, and a normal vector. The second line determines the reflectance and initial exitance for the patches.

Quadrilateral:	4 vertices

Vertex 1:	3D float vector
Vertex 2:	3D float vector
Vertex 3:	3D float vector
Vertex 4:	3D float vector
Patch normal:	3D float normal
Initial exitance (RGB):	3 floats in range [0-1]
Reflectance (RGB):	3 floats in range [0-1]

Example file containing two patches:

```
278 273 -800 278 273 0 0 1 0
256 256 256 256 100
0.0 548.0 0.0 71.0 548.0 0.0 71.0 548.0 186.4 0.0 548.0 186.4 0.0 -1.0 0.0
 0.0 0.0 0.0 0.8 0.8 0.8
71.0 548.0 0.0 142.0 548.0 0.0 142.0 548.0 186.4 71.0 548.0 186.4 0.0 -1.0 0.0
 0.0 0.0 0.0 0.8 0.8 0.8
```

b. Output Format

The output format is the same as the input format, except that the final color for each patch is stored where the reflectance data would be. “Dummy” data is placed where initial exitance would be, used as padding; this allows the renderer to draw the initial, unprocessed scene as if it were lit at 100% on every surface.

5. Renderer

draw_patches *[lit scene file]*

Renders the scene to the screen, using provided image width and height parameters for the window size. All surfaces in the scene are lit as flat-shaded and diffuse. The input file format is the same as the output format for the radiosity process (see 2.a). This process is represented as the second arrow in Fig. 1.

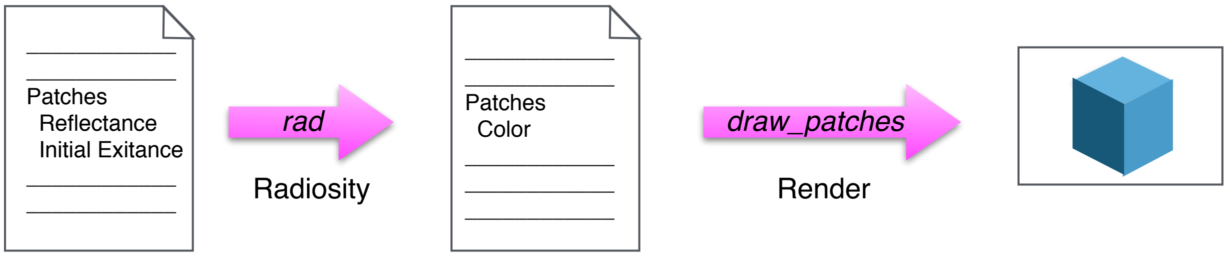

Fig. 1 Execution Pipeline

6. Radiosity Calculations

a. Definitions

exitance — Radiant power emitted by a surface per unit area. Ex. units: W/m2

form factor — Amount of light a patch receives from the surrounding environment.
initial exitance — Exitance that a surface starts with before applying radiosity calculations.
reflectance — The amount of light a surface reflects.

b. Process

First a file is read in for patch data. Patch data includes the geometry of a set of quadrilaterals, their normals, and reflectance and initial exitance given red, green, and blue values. Form factors are then calculated for five sides of a 256 x 256 x 128 cell hemicube. Each cell is given a value based on how much light can be received by the patch through that cell due to attenuation and Lambert's cosine law. Each patch is then assigned a collection indices or references to other, visible patches using a ray tracer, along with form factor scalars for each patch. Then light is collected for each patch a finite number of times. Starting with initial exitance, light is collected by the patches as sent and unsent exitance. In each stage, sent exitance is accumulated, while unsent exitance is depleted, until finally the sent exitance is stored as color values for the patches, and the data is written to file. Finally, the renderer draws the scene using diffusely lit, colored patches.

7. Post Mortem

Unfortunately the system is not complete. Partly, this is due to fatigue, and partly this is due to difficulties re-familiarizing myself with the system due to lack of documentation and clear design goals. However, certain mistakes were found that I had yet to fix.

One problem I noticed was my decision to use hashes of memory locations for patch information. This all started with an idea of mine to not simply record exitance values with each patch, but also all the other visible patches. This would speed up radiosity because the ray tracing pass would only need to be done once. However, hashing based memory addresses in C is not a good idea unless one knows how memory allocation is performed. This could lead to "gaps" in hash tables that are never written to, or having repeated hash collisions. Instead, I should have just indexed the patches by counting integers, as this would be entirely under my control.

8. Future Enhancements

Potential future developments could include addition of shaders, and reading/writing Quake .bsp files as part of the map compiling pipeline. Shaders could be implemented through interactions with rays cast from patches. Transmission effects could be implemented by marking some surfaces as translucent, multiplying a color value by the light value passing through it. Diffuse transmission could be implemented by allowing surfaces to be two-sided, with light adding some percentage of exitance to each side, so that light is transmitted in both directions.

It would also be possible to adapt this system to the Quake map compiling pipeline. This pipeline involves 3 steps: CSG-based binary space partitioning (qbsp), PVS calculations that determine which leaves are visible from each leaf (vis), and lighting (light); the radiosity calculations would be applied at the lighting stage. This would require reading and writing of Quake .bsp files, which is possible thanks to the availability of the Quake source code under the terms of the GPL. Part of this would involve adapting this program to using light maps, which are essentially textures containing lighting data that are multiplied by higher-detailed textures. Surfaces from an input .bsp file would be divided into patches, radiosity would be performed, then that data would be written to light maps for each surface. In addition, the textures would be used to determine reflectance values.