A Simple Radiosity Model

Ryan Gunther

rtcgunth

I.D 95803907

December 5, 1995

CS488/688

Final Project

The purpose of this project was to implement a very simplistic radiosity model. I feel that I have successfully met all of my goals and satisfied all of my stated objectives.

Topics

- Generating very simple meshes.
- Progressive Refinement method.
- Simple calculation of Form Factors.
- Gouraud Shading.

Statement

Radiosity is the rate at which energy leaves a surface. It is the sum of the rates at which a surface emits energy and reflects or transmits it from that surface to other surfaces. The radiosity method is a modeling of visible light that models light interreflections between diffuse surfaces. It assumes that all surfaces in the environment are ideal diffuse reflectors which means that the reflectance does not depend on the outgoing direction. Thus radiosity models are view-independent. Radiosity models allow any surface to emit light so light sources are modeled with an area and they are treated no differently than any other surface in the environment.

To implement a Radiosity model we break up each surface into a finite number of patches. Thus the entire environment will contain some finite number of surface patches. We consider each surface patch to be an opaque Lambertian diffuse emitter and reflector and for each patch we calculate it's radiosity with the following equation:

$$B_i = E_i + \rho_i \sum_{1 \leq j \leq n} B_j F_{j-i} \frac{A_j}{A_i}$$

, where B_i and B_j are the radiosities of patches i and j, E_i is the rate at which light is emitted from patch i, ρ_i is patch $i's$ reflectivity and F_{j-i} is the form factor which specifies the fraction of energy leaving patch j that arrives at patch i, taking into account the shape and relative orientation of both patches and the presence of any obstructing patches. A_i and A_j are the areas of patches i and

j. Therefore, finding the radiosity of each patch involves solving a set of simultaneous equations, however it is possible to approximate the algorithm's results incrementally using a method called Progressive Refinement. This procedure continually selects the patch with the greatest unshot radiosity and shoots it's energy towards all other patches. It then updates the radiosity and unshot radiosity of each patch in the environment and selects the next shooting patch. It terminates when the greatest unshot energy reaches some pre-defined minimum.

Radiosity is a very interesting and challenging subject. It has the restriction that it is limited to environments containing only diffuse surfaces and computing Form-factors is both difficult and time consuming. One advantage it has over Ray-tracing is that because it only considers diffuse surfaces it is view-independent. Another nice feature of the Radiosity model is that it creates images with noticeable color bleeding. Color bleeding is the effect that diffuse reflection has between adjacent surfaces. It causes diffuse surfaces to be tinged with the colors of other diffuse surfaces that they reflect. Another aspect that adds to the difficulty of implementing a good Radiosity model is that surfaces must be subdivided very carefully in order to get a realistic looking image.

Goals

I hope to implement a very simple Radiosity model for this project. The goals that I am aiming to achieve are the following:

I will limit my model to rectangular surfaces only. Thus, my first goal will be to subdivide a rectangular surface into a number of patches and create an efficient data structure to hold the information for all of the surfaces in the environment. This data structure will have to efficiently store information for each patch in the surface. This information includes the vertices of each patch as well as the reflectance and radiosity values for each patch.

My next goal will to be implement a progressive refinement method very similar to the one described by Cohen, Chen, Wallace, and Greenberg[1]. Initially, I will assume a convex environment where all surfaces are in full view of one another and there are no shadows. This simplifies things because there is no need to do any visibility testing. I will calculate Form-Factors using the formula proposed by Baum, Rushmeier, and Winget[2]. This formula will be described in more detail in the Technical Outline section of this proposal.

My third goal will be to modify the Gn ray-tracer I created in assignment 4. I will need to modify it so that when it determines visible surfaces it is able to figure out which patch of the surface is intersected and then use the radiosity value for that patch to determine surface color. Hopefully, this will give me the first bit of encouraging feedback from my project. I should be able to get crude images generated with my radiosity model.

The fourth goal I hope to achieve is the ability to test for visibility when computing my Form-Factors. To do this I will cast four rays from the emitter patch to the receiver patch and see how many reach the receiver without intersecting another surface. I will then weight my Form-Factor according to the amount of the receiver patch that is visible. Next, I will add an ambient correction factor to demonstrate the intermediate results of the radiosity model.

My fifth goal will be to create some scenes that are able to demonstrate some of the effects that my radiosity model is able to create as well as some of the shortcomings that it has. I will also make some scenes that illustrate the differences created by adding visibility checking to my model.

Hopefully, I will be able to explain the problems and limitations of my model and suggest ways in which I could improve it given more time. I will try to write my code in such a way that it will be practical to implement additional advanced features in the future.

My final goal will be to hand the project in on time and get a good night's sleep.

Communication

Input: Slightly modified Gn script

Interaction: None

Output: .ppm image files

Modules

I have used a modified version of Gn so I use all of the standard Gn modules. I have also used the following modules.

radiosity.c contains my main radiosity routine

patches.c contains code to find patch vertices and center points

formfactor.c contains code to calculate formfactors between surfaces

callbacks.c contains the code for the modifed Gn Ray-Tracer, including such added features as gouraud shading

Technical Outline

Here I will explain in more detail how I plan to meet all of my goals and implement the features of my project. As I stated previously, I only plan to handle rectangular surfaces. Thus, I will modify Gn to read in my rectangular surfaces along with the information associated with each surface such as the reflectance of the surface. I will not implement hierarchical modeling and I will specify all surfaces in World coordinates so that I don't have to worry about transformations. To create the patches for each surface I will just subdivide each surface uniformly, allowing as input a value for the size of the subdivision. I will have to put considerable thought into developing a data structure to hold all of the information for the surfaces. This data structure will have to hold values for the radiosities of each patch and it will also have to store all of the vertices of the patches. It will have to be easy to determine which vertices belong to which patch. A clever design for this data structure is crucial to quick computation of the radiosity values.

I will implement the progressive refinement algorithm described by Cohen, Shenchang, Wallace, and Greenberg[1] (page 79) except that I will not use the hemicube method to compute the Form-factors. To compute the Form-Factors I will use the method given by Baum, Rushmeier, and

Winget[2] (page 330). They give a formula for calculating the Form-factor from a point to a surface. It is:

$$F_{dA_j A_i} = \frac{1}{2\pi} \sum_{g \in G_i} N_j \bullet \Gamma_g$$

where: G_i is the set of edges in surface i

N_j is the surface normal for the differential surface j

Γ is a vector with magnitude equal to the the angle gamma (in radians) illustrated in the diagram below, and direction given by the cross product of the vectors R_g and R_{g+1} as illustrated below. I will implement visibility testing by shooting a ray to each vertex of the receiving patch

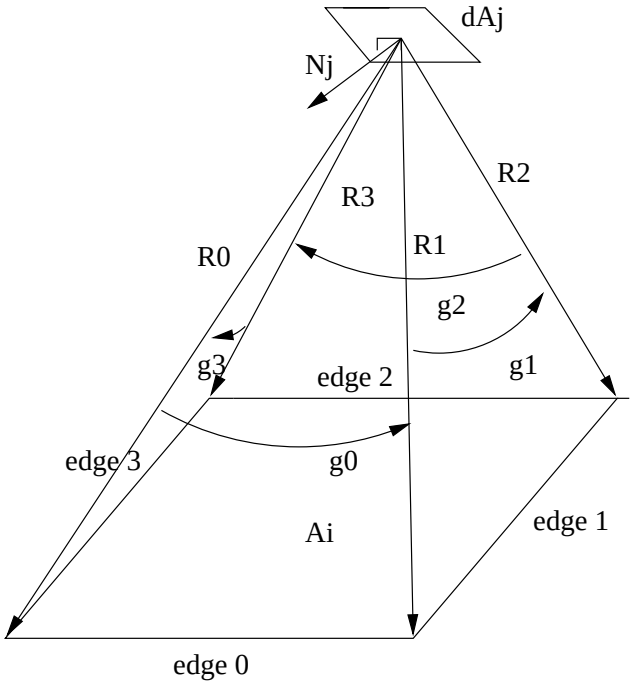

Figure 0.1:
Form-Factor diagram.

and multiplying the value I get for my Form-factor by the ratio:

[num rays that don't intersect a surface between the emitter and receiver divided by four]

I will try to implement this algorithm in such a way that it is possible to increase the number of rays which can be shot. The algorithm for the Ray-tracer I implemented in assignment 4 will have to be modified so that it can determine which patch of a surface is intersected. The formulas for computing the radiosity and unshot radiosity values are given by Cohen in his progressive refinement algorithm. He also gives a formula for computing the ambient light term which I will use to generate intermediate images.

Milestones

My first task will be to read the literature on radiosity more throughly and make sure I have a complete understanding of all the issues involved. This will help me to avoid potential pitfalls.

Next, I will come up with a data structure that stores all of the patch information for each surface. This data structure should store the vertices for each patch in such a way that it is easy to figure out which vertices correspond to a particular patch. I would like to have this done by Wednesday, November 22.

Next I will modify Gn to allow me to read in square surfaces, subdivide them uniformly, and store them in my data structure. I would like to have this done by Saturday, November 25.

Then I will implement the progressive refinement algorithm without visibility testing and modify my Ray-Tracer so that it can create a .ppm image file from the radiosity information I have stored for each patch of a surface. At this point I will create some simple scenes. I would like to have this done by Wednesday, November 29.

Then I will add visibility testing to my model and create some more interesting scenes to compare against the ones generated without visibility testing. I should have this done by Friday, December 1.

Finally, I will add the ambient light factor to generate intermediate images and I will construct some scenes which demonstrate the shortcomings and limitations of my model. This should be done by Sunday, December 3. This will leave me a day to write up my documentation and do some final testing. (Not much time!)

Organization

All files will be found in the subdirectories under the `A5` directory. The subdirectories and files are indicated in the following.

radiosity contains all of the source code for my project

data contains all of the Gn scene scripts I create

doc contains all of the documentation for my project

exe contains the executable for my project

Documentation

The file `doc/README` gives a brief installation guide. The file `doc/MANUAL` is a (not too lengthy) user's guide to running the programme. The file `doc/REPORT` is a short technical report on the principles, algorithms, and background literature on the project.

Implementation

Data Structures

To implement radiosity as my final project I expanded upon the Gn data structures. To create a scene you can still write standard Gn scripts with a few minor modifications that I have implemented. I created a new model geometry called **GnRecSurface**. This is my geometry for a rectangular surface, which is the only type I support for my radiosity project. It has the following structure:

GnRadPatch subdivision[100][100];

GnVector3D surfacenormal;

GnPoint3D surfacepoints[3];

It's surface can be uniformly subdivided up to a maximum of 100x100 patches. It also contains the surface normal and the three points defining the surface.

A surface patch has the following structure:

GnRadiosity B *total radiosity values*

GnRadiosity deltaB *unshot radiosity values*

double unshotvalue *total unshot radiosity*

GnColour roe *light reflexivity for the patch*

GnVector3D patchnormal *patch normal*

GnRadiosity estimate *estimate for radiosity with ambient term added*

Progressive Refinement Algorithm

I implemented a slight variation of the progressive refinement algorithm developed by Cohen, Shebchang, Wallace, and Greenberg[1]. My version is as follows:

Step 1 Initialize unshot and total radiosity of each patch to its emission value.

Step 2 Set a convergent factor to a supplied user specification, or default to 0.1. The algorithm will terminate when no patch has an unshot radiosity greater than that of the convergent factor.

Step 3 Cycle through all of the patches to find the patch with the highest unshot radiosity.

Step 4 Cycle through each of the remaining patches in the scene, calculate the formfactor between the shooting and the current receiving patch. Then shoot of rays to determine if the shooting and receiving patch are visible to one another. The user can specify anywhere between 0 and 4 rays to test for visibility. Weight the formfactor by the number of rays which do not

intersect a surface between the receiver and shooter. Then transfer radiosity to the receiving patch based on the form factor and the reflectivity of the patch. The formula to do this was stated previously as was the formula for calculating the form factor.

Step 5 Repeat steps 3 and 4 until the convergent factor is reached.

The user also has the option to render intermediate images or to add an ambient estimation factor in order to get better looking intermediate images. To render intermediate images I simply increment a counter each time a new shooter is found and when the counter reaches a specified number, I call my ray tracing routine and render an intermediate image. I render it with the estimated values if the user has supplied a parameter which puts on the ambient correction factor.

To calculate the estimated ambient factor I use the following algorithm given by Cohen, Shebchang, Wallace, and Greenberg[1]. It is as follows:

The form-factor from any patch i to patch j can be approximated as the fraction of the total area of the environment taken up by the area of patch j. Thus an estimated form-factor for each patch can be calculated as:

$$F_{*j} \approx \frac{A_j}{\sum_{j=1}^{n} A_j}$$

An average reflectivity for the environment can be computed as an area weighted average of the patch reflectivities:

$$\rho_{ave} = \frac{\sum_{i=1}^{n} \rho_i A_i}{\sum_{i=1}^{n} A_i}$$

For any unit energy sent into the environment, ρ_{ave} will on average be reflected, and some of that will be reflected, etc.. Thus an overall interreflection factor R is simply the geometric sum:

$$R = 1 + \rho_{ave} + \rho_{ave}^2 + \rho_{ave}^3 + \cdots = \frac{1}{1 - \rho_{ave}}$$

From these assumptions we can derive an Ambient radiosity which is simply the area average of the radiosity which has not yet been shot by formfactor computation times the reflection factor R.

$$Ambient = R \sum j = 1^n (\Delta B_j F_{*j}$$

Then whenever I render the scene I derive an estimate for the radisoity of each patch with the following formula:

$$B_i^{'} = B_i + \rho_i Ambient$$

I only use this estimate for display purposes, it has no influence on the final image.

The progressive refinement algorithm is in my module **radiosity.c** .

Form-Factor Algorithm

The algorithm I used to calculate Form-Factors was already stated in my **technical outline**. The algorithm I use to test for visibility was also given there. I have attempted to speed up this algorithm in the following way. I shoot one ray at a time and test it against all of the surfaces in the scene(not against each patch). If it intersects a surface, I store this surface and that is the first surface I test my next ray against. It is likely that all four rays intersect against the same surface so this will speed things up a little bit. I also test against this surface with the first ray of my next patch because it is likely that adjacent patches will also intersect with the same surface. I allow the user to specify how many rays are to be shot for visibility testing. They can specify from 0 to 4 rays. One ray is shot from each vertex of the recieving patch to the center of the shooting patch if four rays are specified. All of my Form-Factor algorithms are in the module formfactor.c. Thus, it would be easy to implement a different method for computing form-factors, such as the hemicube method, at some point in the future. I have also seperated the code for computing form-factors from the code for computing visibility so it would be easy to implement a different algorithm for determing visibility without changing the form-factor algorithm.

Ray-tracing Algorithm and Gouraud shading

I have just trimmed down my ray tracer from assignment A4. It now just checks for intersections with rectangular surfaces and determines which surface is closest. When it intersects a surface it uses the u and v vectors of the surface to determine which patch of the surface has been intersected. Than I have written a routine to determine the vertices of this patch and I calculate the radiosity of each vertex. I do this by taking the average radiosity of the four patches surrounding a vertex. Then I use Gouraud shading to linearly interpolate intensity values across the patch.

Gnscript

Gnscripts can be written in the normal fashion, except for the following modifications I have made:

Addition 1 I have created a new model geometry, which is the only one that my radiosity program can handle. It is can be specified by giving *recsurface* as the geometry for a model. A list of three points defining the surface must follow this option along with a parameter called subdivide and a list of two integers which specify the amount of subdivision for this surface. The first integer specifies subdivision along the u direction and the second one specifies subdivision along the v direction.

Addition 2 To specify that a radiosity lighting model is being implemented, a paramater called *radiosity* must be supplied with a material name as well as the reflectance of the material. Reflectance is specified with the paramater *reflect* followed by a list of three doubles in the range from 0.0 to 1.0. If the object is to be a light source, an *emission* paramater must also be specified followed by a list of three doubles. As well, if you wish a material to be rendered with Gouraud shading, you must include the paramater *shade* followed by *gouraud*.

Addition 3 When specifing a camera object for the scene, there are now some additional paramaters you can specify. You can include an iterations paramater, *iteration*, followed by an integer which will cause an intermediate image to be rendered after each number of specified passes through the progressive refinement loop. This is useful because it enables you to watch the radiosity algorithm in progress. The files it creates will have the name supplied, with an integer appended to it that identifies how far along in the algorithm the program was at the time the image was rendered. An *addambient* paramater can also be specified. If it is given, then any intermediate images generated will have the estimated ambient factor added to the radiosities of the patches. A *shadows* paramater can also be included followed by an integer between 0 and 4. This specifies the number of shadow rays to be shot for visibility testing. 0 means that no visibility testing is performed. The default value is 4.

Manual

The program was compiled and runs on an SGI. The main executable is called **gnash_ray**. I have not had time to implement any real error-handling if the user enters invalid scripts. I assume that a correct script has been run and I do not guarantee that any of the Gn functionally except that required for my implementation of radiosity will work.

The image files are output as .ppm files. I have already discussed all of the algorithms and data structures I use in the above sections.

Some sample image files which demonstrate some aspects of my radiosity algorithm are in the directory `data`. They demonstrate the following things:

Uniform subdivision The image *Demo1.ppm0* demonstrates that I have sucessfully implemented uniform subdivision of rectangular surfaces. It also demonstrates that I have implemented visibility testing. It shows that just implementing uniform subdivision won't produce very good images. They appear blocky and shadow boundries are too rigid. You don'y get a nice soft shadow effect.

Gouraud shading The image *Demo0.ppm0* illustrates the fantastic effect that implementing Gouraud shading has. This image is exactly the same as the previous one, except for the fact that Gouraud shading has been implemented. You now no longer have this blocky effect and there are nice soft shadow boundries.

Shadow Rays The image *Demo5.ppm0* demonstrates what happens when only one shadow ray is cast as opposed to four shadow rays. There is a much larger shadow on the right hand side because only one ray is being shot so there is less degree in the range that a shadow can have.

Colour Bleeding The image *Demo10.ppm* demonstrates the aspect of colour bleeding. The wall on the left is a white wall, but it has a redish, blueish, and greenish tinge to it. This is because the other walls in the room reflect some of their colour onto the white wall.

Mach bands The image *Demo10.ppm* also demonstrates one of the short comings of uniform subdivision. If you look closely at the white wall you will notice bright streaks on the wall. This is a result of uniform subdivision and would be less noticable if I subdivided the wall more.

Progressive Refinement The image *Demo15* demonstrates the progressive refinement algorithm by showing various intermediate images. Each image occurs after 40 more iterations of the progressive refinement loop.

Ambient light factor The image *Demo16* demonstrates the addition of an ambient light estimate. A comparison is made between images with the ambient light estimate and without the ambient light estimate. You can see that initially, the images with the ambient light estimate are much closer to the final image.

Uniform Subdivision and Shadows The image *Demo20.ppm0* demonstrates how uniform subdivision fails to handle shadows properly when one surface is extremely close to another surface.

Sources

In `src`. The command `make` will compile my project and produce an executable called gnash_ray.

Executable

My executable is named `exe/gnash_ray`.

Data Files

My image files are in `data`

Gn Script files

My script files are in `exe`

Graduate Report

This brief report contains the material necessary for my tenth objective.

As you could see from some of my sample images, one of the biggest problems with my implementation of radiosity is the fact that I uniformly subdivide each surface into square patches. This causes artificats such as the previously mentioned Mach bands. Mach bands are caused because of first-derivative discontinuties across patch boundries. These discontinuties are caused becuase usually the rate of change of the radiosity function is different on both sides of a patch edge and our eye is very sensitive to these kinds of changes.

There are several techniques you can use that are an improvement over uniform subdivision. Substructuring is one approach that can be taken. In this approach you can treat patches differently depending on whether or not they are the emitter of energy or the reciever of energy. Each patch can be considered as an element itself when it is the emitter or it can be broken down into smaller pieces when it is the reciever. Then you can compute the radiosity for each element of a patch and the patch

radiosity can be obtained as an area-weighted average of it's element radiosities. This algorithm gives a much more accurate radiosity solution because intensity variations are represented inside the patches by the small surface elements. However, this method has the drawback of requiring more storage and the algorithm takes much more time becomes there are more elements to take into consideration.

Another technique that can be used is that of adaptive mesh refinement. The idea behind this approach is that a denser mesh of elements is needed in areas where the illumination changes radiply, since the assumption of a constant radiosity per element means that the radiosity function across a surface must be treated as a piecewise constant function. Therefore, we want to try to have the mesh follow the distribution of light in the environment, with a higher density of surface elements in places where the illumination changes rapidly. However, it is unrealistic to suggest that the user should try to guess where these rapid changes in illumination might occur so there should be some way to generate the surface mesh automatically by having the program look at the properties of the environment. Sillion gives two approaches that can be taken in doing this.[4](page 71).

The first approach involves predicting where important illumination features, such as shadow boundries, fall on surfaces. The mesh could than be based on the predicted location of these features.

Sillion suggests that a simplier idea would be to have the mesh evolve as the simulation progressed. This means it would generate more elements in areas where a rapid change in the radiosity function is detected. Thus, the program would have to constantly evaluate the radiosity solution and make new meshing decisions. New elements would be generated in areas where the intensity of light changed rapidly, but no new patches would ever be introduced into the mesh. Sillion says that the rate of change in radiosity across a surface element could be estimated by looking at the radiosity of neighbouring elements. A new mesh could than be generated by subdividing all elements for which the gradient exceeds some predefined threshold. Sillion also says that more elaborate schemes can be developed such as computing radiosity values at the vertices of elements as a weighted average of the neighbouring element's radiosities. This method means that the varying level of subdivision of neighbouring elements can be accounted for because a very small element that shares a vertex with a large element will have more influence on that vertex's radiosity. Vertex radiosity values can than be used to estimate the radiosity gradient.

Another method that has been suggested to improve the radiosity solution is to remove the arbitrary seperation of patches and elements and replace them by a continuum of hierarchial elements. Radiosity exchanges can than be computed between the various levels of the hierarchy and for any pair of surfaces an appropriate subdivision level on both surfaces could be determined and used to compute the energy exchange. This method is fairly involved and the details are beyond the scope of this report.

Finally, there are various other more accurate ways to compute form-factors than the analytical method I used. One such method is the hemi-cube method. The idea here is to center a cube around the point from which form factors are to be computed. Then discretize the faces of the hemi-cube into a number of square cells for which a delta form factor can be computed analytically. Than each face of the hemi-cube is used as a projecton surface to determine visibility by using what is called an item buffer. This is an array of patch identifiers, one for each hemi-cube cell, that records the

visible surface for the direction of each cell. It is constructed using the z-buffer strategy where all surfaces are projected in turn onto the face of the cube. This determines the closest surface patch visible through each cell. The greater hemi-cube resolution you have, the greater your accuracy. Than the form-factors to all surfaces can be computed at once and each hemi-cube cell contributes it's delta-form-factor to the form-factor with it's visible patch. Than the form-factor to a given patch can be obtained by summing up all of the delta-form factors corresponding to all the cells covered by the projection of the patch.

In conclusion, Radiosity is a very complex topic which is currently a very active area of research. The algorithms I used were very simplistic, but I now have a definite feel for what radiosity is all about. This has been a very rewarding project.

0.1 Bibliography

[1]Cohen, Michael F., Shenchang Eric Chen., Wallace R. John, Greenberg P. Donald, "A Progressive Refinement Approach to Fast Radiosity Image Generation," *Computer Graphics* (SIGGRAPH '88 Proceedings)22pp. 75-84(1988).

[2]Baum, R. Daniel, Rushmeier E. Holly, Winget M. James, "Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors" *Computer Graphics* (SIGGRAPH '89 Proceedings)23pp. 325-334(1989).

[3]Foley, vanDam, Feiner, Hughes, *Computer Graphics Principles and Practice* Addison-Wesley(1990).

[4]Sillion X. Francois, Puech Claude, *Radiosity and Global Illumination* Morgan Kaufmann(1994).

[5]Glassner, S. Andrew, *Principles of Digital Image Synthesis* Volume Two. Morgan Kaufmann(1995).

[6]Watt Alan, Watt Mark, *Advanced Animation and Rendering Techniques Theory and Practice* Addison-Wesley(1992).

[7]Hearn, Donald, Baker, M. Pauline, *Computer Graphics* Prentice Hall(1994).

[8]Kwok, Bernard, *Analysis of Radiosity Techniques in Computer Graphics* (Thesis) York University(May 1992).

Objectives: Project

Name: Ryan Gunther

UserID: rtcgunth

Student ID: 95803907

___ Objective 1.Scene is designed and square surfaces are uniformly sub-divided into patches.

___ Objective 2.A well designed data structure is created which allows efficient retrieval of patches and vertices for a surface which has been uniformly subdivided.

___ Objective 3.The progressive refinement algorithm has been successfully implemented.(i.e. a method for determining which patch is the shooter and an algorithm for updating the radiosities of all recieving patches is in place. This does not include Form-Factor calculation)

___ Objective 4.Calculation of Form-Factors without visibility testing.

___ Objective 5.Calculation of Form-Factors with visibility testing in place.

___ Objective 6.Modification of ray-tracer to create image files using computed radiosity values of surface patches.

___ Objective 7.Addition of an ambient correction factor to generate intermediate images of my radiosity model.

___ Objective 8.Creation of images which clearly demonstrate the difference between my radiosity model with visibility testing as opposed to my model without visibility testing.

___ Objective 9.Creation of images which clearly show that the meshing I use(i.e. uniform) is inadequate.

___ Objective 10.A short written explanation addressing the problems of my model and how my software design could be modified to address further radiosity techniques known to treat these problems.

Declaration:

I have read the statements regarding cheating in the CS488/688 course handouts. I affirm with my signature that I have worked out my own solution to this assignment, and the code I am handing in is my own.

Signature: