Radiosity Project

Alejandro Armenta

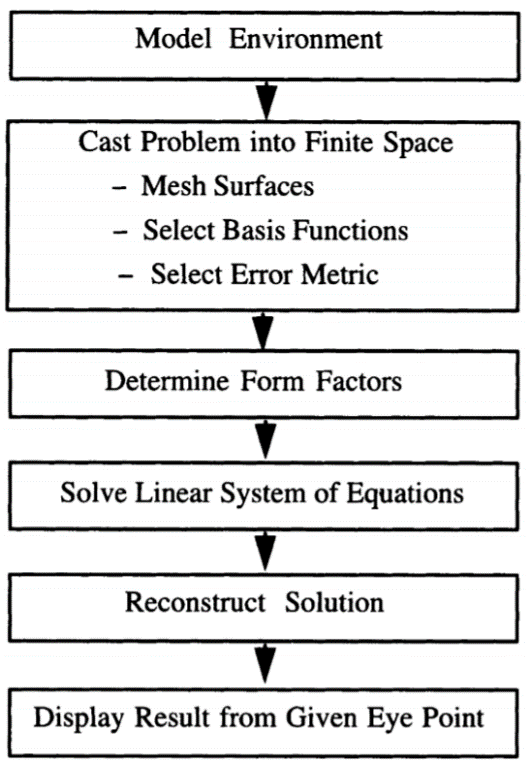
# The Radiosity Model

The radiosity model is defined by a Fredholm Integral equation of the second kind:

|  |  |  |
| --- | --- | --- |
|  |  | () |

This equation represents the stable distribution of light within environment and needs to be solved to solve the global illumination problem.

These are the steps I followed to solve the equation:



To solve the problem, I used the finite element method to discretize my radiosity model into a sum of easier polynomial functions of the form:

|  |  |  |
| --- | --- | --- |
|  |  | () |

The basis functions I used are constant basis functions which are 1 within element and 0 otherwise.

# Galerkin Form of Weighted Residuals.

I needed to define an error metric to approximate my radiosity model, I chose the Galerkin Form of weighted residuals to minimize the error as an average across element surface, so:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Expanding :

|  |  |  |
| --- | --- | --- |
|  |  | () |

The Galerkin Formulation selects the same basis functions used to approximate the radiosity function as the weighting functions, thus:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Finally expanding :

|  |  |  |
| --- | --- | --- |
|  |  | () |

Grouping terms:

|  |  |  |
| --- | --- | --- |
|  |  | () |

This is the system of equations to solve. This system of equations can be expressed in matrix form as:

|  |  |  |
| --- | --- | --- |
|  |  | () |

The entries of K are given by:

|  |  |  |
| --- | --- | --- |
|  |  | () |

And the entries in E are:

|  |  |  |
| --- | --- | --- |
|  |  | () |

We can simplify the above equation by assuming constant basis functions, and constant reflectivities and emissivities within element.

|  |  |  |
| --- | --- | --- |
|  |  | () |

Where is the Kronecker Delta function:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Then:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Where is the average area emission value for element .

Then we can change the integral, to integrate over element areas only:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Making these substitutions in equation 3.20:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Dividing through and moving emission term to the right side gives:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Which is:

|  |  |  |
| --- | --- | --- |
|  |  | () |

# The Form-Factor

The form factor ( in the last equation), specifies the fraction of the energy leaving one surface which lands in another.

By definition the sum of all the form-factors from a particular point or patch is equal to unity.

For non-occluded environments, the form factor for one differential area to another is given by:

|  |  |  |
| --- | --- | --- |
|  |  | () |

By integrating over area j, the form factor from a finite area (or patch) to a differential area can be expressed:

|  |  |  |
| --- | --- | --- |
|  |  | () |

The form factor between finite surfaces (patches) is defined as the area average and is thus:

|  |  |  |
| --- | --- | --- |
|  |  | () |

This expression for the form-factor does not account for the possibility of occluding objects hiding all or part from one patch from another. The complete form is then:

|  |  |  |
| --- | --- | --- |
|  |  | () |

The function takes one If differential area can see differential area and zero otherwise.

This double area integral is difficult to solve analytically, so I used the Hemi-Cube algorithm to compute form factors between surfaces (patches).

# The Hemi-Cube Algorithm

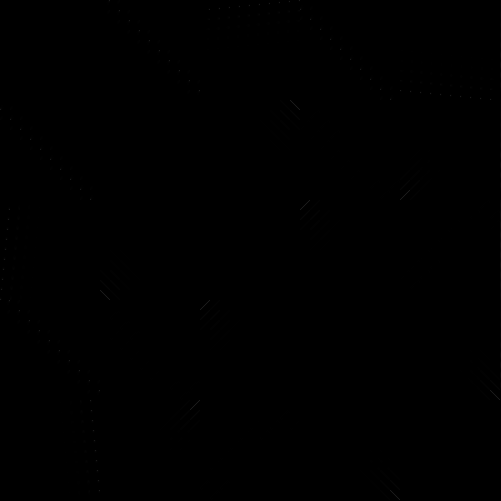
If the distance between two patches is large compared to their size, and they are not partially occluded from one another, it can be seen that the integrand from the inner integral remains almost constant. In that case the effect of the outer integral is simply a multiplication by one, and finding a solution to the inner integral will provide a good approximation for the form factor.

I set a hemi-cube around the normal in patch and discretized its area into small pixels (100 x 100 per face). The hemicube contains 5 faces, 1 full face and 4 half-faces. The area of each pixel determines the multiple to multiply the form factor by, so the form factor for each pixel can be calculated as:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Each value goes into a matrix of form factors. (e.x. if we are on element and we see element through pixel then value goes into row column of Form Factor Matrix.)

Making this calculation for all elements in scene, we get filled a full n by n Form Factor Matrix, the resulting matrix can be seen in the next image (NOTE: it is possible that the values cannot be seen because they are relatively low values, but that is good because the sum of each row needs to be less than one to make the system converge):



# The Energy Balance solution

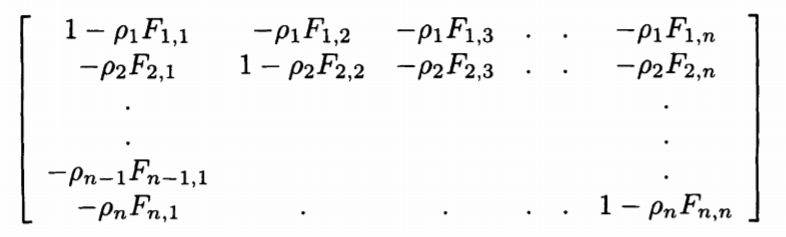
Now having the form-factor matrix we can solve the system of linear equations:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Where K is defined to be:

|  |  |  |
| --- | --- | --- |
|  |  | () |

M is the identity matrix; P is a diagonal matrix with reflectivities on its diagonal and F is the matrix of form factors. Thus, K looks like:



Using the form factor matrix computed before we can do matrix operations to compute which leads to this matrix:



The matrix K is diagonally dominant; thus, it is well suited to iterative methods as Gauss-Seidel, which is the method I used to solve the system of linear equations.

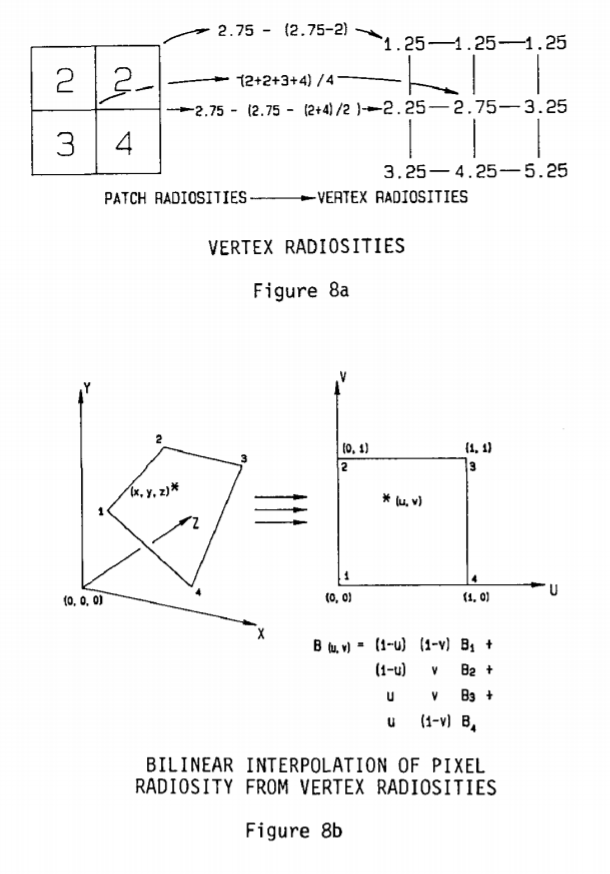
The Gauss-Seidel Method is really fast with K matrix (only one iteration). Now to make it converge the spectral radius of K must be less than 1. Therefore, the largest absolute value for an eigenvalue of K must be less than one.

# Stimuli-Dependent

There is a system of linear equations per stimuli required, so for the radiosity code I solved three systems of linear equations (red, green and blue).

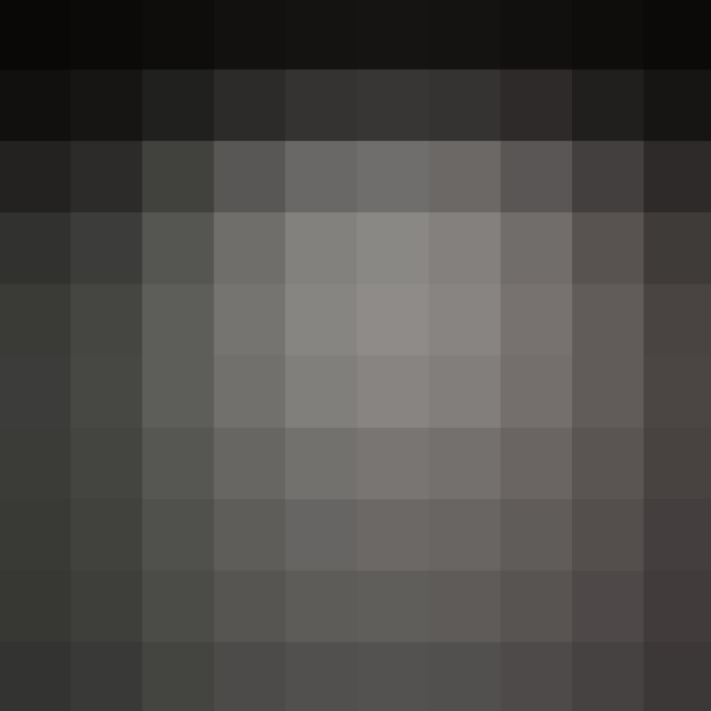
# Rendering

Once the radiosity values are known, I pass that information to vertex colors for each quadrilateral as shown in the next image:

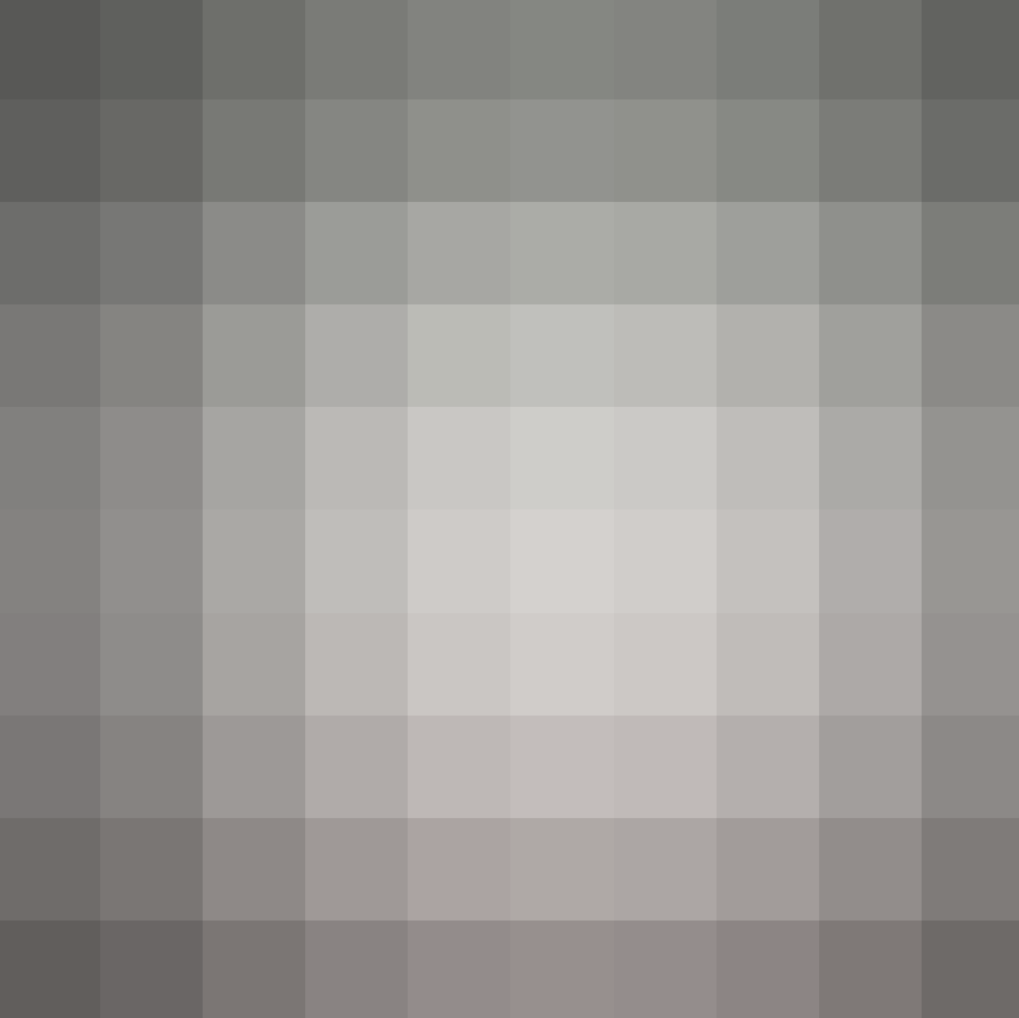


The vertex values are used to bilinearly interpolate vertex colors when the rendering pass requests a color from the scene in a specific direction.

These are vertex colors for face xy with z = 0 without interpolation. I only render Upper Left Vertex Color for each Quad in this case):

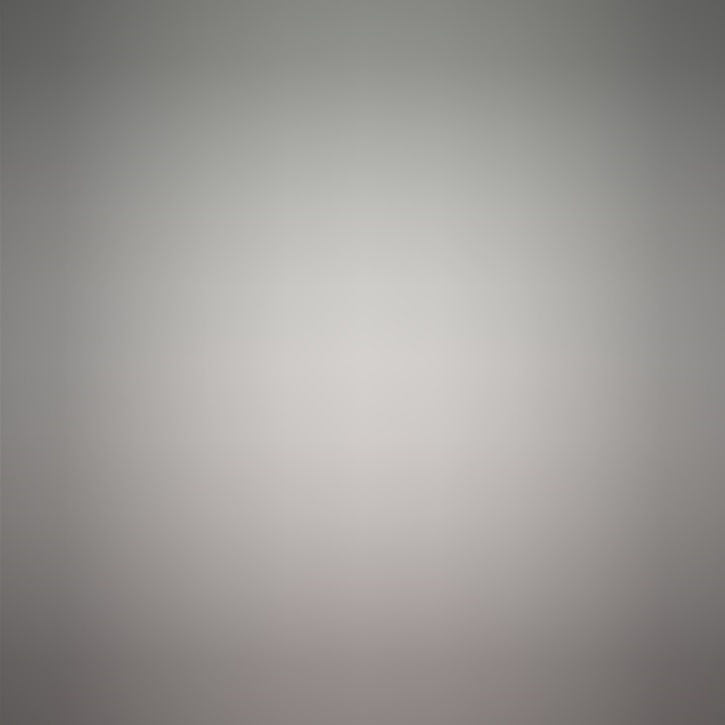


And this is face xz with y equal 0:

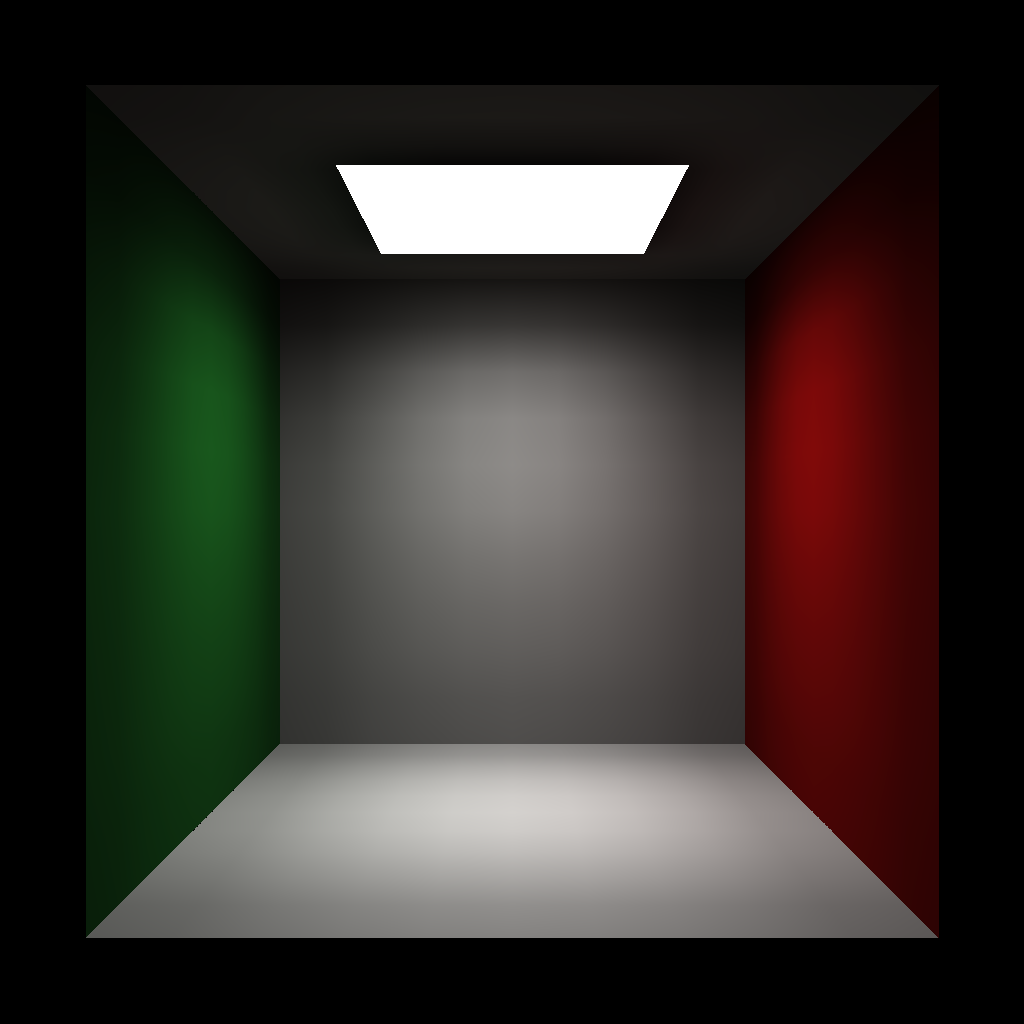


Now with interpolation respectively:





The radiosity solution is view-independent, therefore changing the view position does not need a new computation of radiosities nor form factors.



The Mesh is generated procedurally, for this image I used 10 elements per face side, each face side is 10 units long.

Bibliography:

1. M.F. Cohen and J.R. Wallace. Radiosity and Realistic Image Synthesis. Boston, MA: Academic Press Professional, 1993.