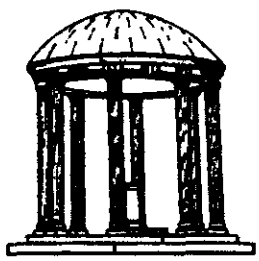
Progressive Radiosity Using Hemispheres

TR89-002

February 1989

Jack Goldfeather



The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

Progressive Radiosity Using Hemispheres

by Jack Goldfeather

February 13, 1989

INTRODUCTION

In these notes, I will outline an approach to progressive radiosity using a hemisphere rather than a hemicube. A hemisphere is the correct geometrical figure to use for calculating form factors, but requires non-linear mathematical calcuations. A hemicube has been shown to be a good geometrical approximation and one which requires only linear calculations for scan conversion and z-buffering – the two important steps in the computation of the form factors. This has been especially important in an era in which fast hardware exists for doing linear calculations, but not higher order calculations. There is, however, a computational disadvantage to hemicube calculations, namely, the data base needs to be transformed, scan converted, and z-buffered once for each face of the hemicube – a total of 5 times. In summary,

	ADVANTAGE	DISADVANTAGE
HEMICUBE	Linear Calculations	Patches processed 5 times
HEMISPHERE	Patches processed once	Non-Linear Calculations

The Pixel-planes5 system promises to reduce the disadvantage of the hemisphere by providing a fast non-linear calculating device, namely the quadratic expression evaluator (QEE). I will outline in the next section how the QEE can be used to compute the form factors based on projection of the patches onto a hemisphere (assuming the patches are planar convex polygons). In the last section, I will discuss why this approach seems likely to provide significant speedup over the hemicube approach in pxpl5.

I. COMPUTING FORM FACTORS USING A HEMISPHERE

The traditional hemicube approach is

(1) Compute (as a pre-process) the Δ form factors for a hemicube
(2) Enable a rectangular xy-buffer for each side of the hemicube.
(3) Scan convert patches in the buffer.
(4) Z-buffer the patches in the buffer.
(5) Store at each (x,y) in the buffer the ID number of the closest patch.
(6) Use the ID numbers and the Δ form factors to compute form factors.

I describe below how each of (1), (2), (3), and (4) need to be modified for a hemisphere.

Δ form factors for a hemisphere

A hemisphere is placed on a planar polygon so that its polar axis coincides with the plane's surface normal. The Δ form factors for a hemisphere are computed by subdividing the hemisphere's surface into patches P_{ij} and computing:

$$\Delta F_{ij} = \frac{Area(P_{ij})\cos(\theta_{ij})}{\pi}$$

where θ_{ij} is the angle that P_{ij} makes with the polar axis. If P_{ij} projects to region R_{ij} in the xy-plane then

$$Area(P_{ij}) = \int\int_{R_{ij}} \frac{1}{\sqrt{1-x^2-y^2}} dxdy$$

The choice of coordinate system for the hemisphere subdivision is crucial and must be based on the complexity of the projections of edges and faces onto the surface of the hemisphere. A spherical coordinate subdivision of the hemisphere while simplifying the integral calculation, suffers from the problem that lines projected onto the surface of the hemisphere will involve trig functions. A cartesian coordinate subdivision creates a much harder integral, but projected lines will be quadratic functions. For this reason, we subdivide the hemisphere using $\Delta x \Delta y$ regions in the plane. This creates 2 difficulties:

(1) A subdivision of the circular region

$$\{(x,y)|x^2+y^2 \leq 1\}$$

into small rectangles of area $\Delta x \Delta y$ omits small non-rectangular regions near the boundary of the circle.

(2) The surface area integral

$$\int_{x_{i-1}}^{x_i}\int_{y_{j-1}}^{y_j} \frac{1}{\sqrt{1-x^2-y^2}} dydx$$

cannot be evaluated in closed form so it needs to be approximated.

Since the Δ form factors are computed once off-line, (2) is not a problem. Standard numerical techniques can be used to compute the integral to any accuracy desired. The problem of the missing pieces in (1) is negligible since they occur near the base of the hemisphere where multiplication by $\cos(\theta_{ij})$ makes the delta form factor ΔF_{ij} practically 0.

Enabling the xy-Buffer

A circular xy-buffer needs to be used for the hemisphere. This can be achieved especially easily in pxpl5 by enabling a square buffer and then disabling those pixels outside a circle.

Scan Converting Planar Polygonal Patches

As discussed above, we choose a parametrization (u, v) of the hemisphere so that edges of polygons in (x, y, z) space project to quadratics in (u, v) space. Specifically,

$$(*) \qquad x = \rho u, \quad y = \rho v, \quad z = \rho\sqrt{1 - u^2 - v^2}.$$

where $\rho = \sqrt{x^2 + y^2 + z^2}$ is the distance to the origin.

Geometrically, this takes the point (x, y, z), projects it radially onto the surface of the unit sphere, and then projects it downward onto the xy-plane. Each edge of the polygon determines a plane passing through the edge and the origin, which we will call the cutting plane. The cutting plane intersects the sphere in a great circle, and this great circle when projected down onto the xy-plane is an ellipse centered at the origin. Further, the cutting plane intersects the xy plane in a line which we will call the cutting line. We need to find the equation of this line and the equation of the ellipse in uv coordinates.

The cutting plane has equation

$$(**) \qquad Ax + By + Cz = 0$$

since it passes through the origin. The cutting line is found by setting $z = 0$. Using $(*)$, we express this in uv coordinates as

$$Au + Bv = 0.$$

We find the equation of the ellipse by using $(*)$ in the plane equation to obtain

$$A\rho u + B\rho v + C\rho\sqrt{1 - u^2 - v^2} = 0.$$

Dividing through by ρ and rearranging terms we get

$$C\sqrt{1 - u^2 - v^2} = -(Au + Bv).$$

Squaring both sides and rearranging we get

$$Q(u, v) = (A^2 + C^2)u^2 + 2ABuv + (B^2 + C^2)v^2 - C^2 = 0.$$

The cutting line divides the ellipse E into 2 parts, EPOS and ENEG, which can be thought of geometrically as the parts of the projecting great circle which lie above and below the xy plane. The arc EPOS divides the circular uv disk into two pieces. We will describe below how to use 2 one-bit flags to find which piece a given pixel (u, v) is in.

We must be able to determine EPOS and this can be by solving $(**)$ for z:

$$z = \frac{Ax + By}{-C} = \rho\frac{Au + Bv}{-C}$$

If we choose the plane normal so that $C < 0$ and observe that ρ is always positive then we see that the sign of z is the same as the sign of $L(u, v) = Au + Bv$.

The scan conversion process described below is based on a uv frame buffer centered at $(0,0)$ and ranging between -1 and 1. A frame buffer that has $(0,0)$ in the lower left corner and ranges between 0 and N needs to have everything scaled. That is, $u = as + b$, $v = at + b$, for appropriately chosen a and b. Plugging these in for u and v everywhere moves all equations into st space consistant with pixel addresses. To simplify the following discussion, I will describe everything in uv coordinates.

The scan conversion process begins by setting ENABLE for all pixels in the uv buffer. For each edge a vertex of the polygon not on the edge is chosen and converted to (u_0, v_0) using $(*)$. A one bit flag POS is used to determine which pixels are on the positive side of $Au + Bv$ as discussed above. A second one-bit flag INSIDE is used to determine whether a pixel is inside of the ellipse. The linear expression $L(u,v) = Au + Bv$ is broadcast and is used to set the value of POS at each pixel. The quadratic expression $Q(u,v)$ is broadcast and is used to set the value of INSIDE at each pixel. There are 2 cases to consider, depending on the values of $L(u_0, v_0)$ and $Q(u_0, v_0)$.

CASE 1 $L(u_0, v_0) > 0$ and $Q(u_0, v_0) > 0$. That is, the third vertex lies on the positive side of the cutting line, and is outside the ellipse. Then a pixel is disabled if INSIDE OR not(POS). This geometrically equ

CASE 2 All other possibilities. Then a pixel is disabled if not(INSIDE) AND POS.

Geometrically, this is equivalent to disabling pixels on one side or the other of EPOS.

Z-Buffering Patches

The distance from the center of the hemisphere to a point (x,y,z) is given by $\rho = \sqrt{x^2 + y^2 + z^2}$. If the plane equation of the polygon is:

$$Ax + By + Cz + D = 0$$

then using the reparametrization of $(*)$ it becomes:

$$A\rho u + B\rho v + C\rho\sqrt{1 - u^2 - v^2} + D = 0$$

which when solved for ρ becomes:

$$\rho = \frac{1}{(\frac{-A}{D})u + (\frac{-B}{D})v + (\frac{-C}{D})\sqrt{1 - u^2 - v^2}}$$

Since we only need to know relative distances, we can use

$$\lambda = \frac{-1}{\rho} = (\frac{A}{D})u + (\frac{B}{D})v + (\frac{C}{D})\sqrt{1 - u^2 - v^2}$$

which monotonically increases as ρ monotonically increases. The quadratic expression evaluator cannot evaluate this expression directly. There are 2 possible approaches to resolve this.

APPROXIMATION OF Z. The expression $\sqrt{1-u^2-v^2}$ can be approximated using a Taylor series expansion by:

$$1-\frac{1}{2}(u^2+v^2).$$

The expression for λ can then be approximated by the quadratic expression

$$\gamma = (\frac{A}{D})u + (\frac{B}{D})v + (\frac{C}{D})(1-\frac{1}{2}(u^2+v^2))$$

If we let

$$d(u,v) = \sqrt{1-u^2-v^2} - (1-\frac{1}{2}(u^2+v^2))$$

then

$$\lambda_1 - \lambda_2 = \gamma_1 - \gamma_2 + d(u,v)(\frac{C_1}{D_1} - \frac{C_2}{D_2}) = \Delta\gamma + DIFF$$

Hence an error can occur only if $\Delta\gamma$ and $DIFF$ are of opposite sign *and* $|\Delta\gamma| < |DIFF|$. Preliminary tests show that this approximation rarely produces incorrect choices about which value is larger.

STORING $\sqrt{1-u^2-v^2}$. The expression $\sqrt{1-u^2-v^2}$ can be stored at each pixel (u,v) and the correct z value can then be computed by performing a few arithmetic operations at each pixel.

II. ESTIMATE OF SPEEDUP IN PXPL5

In this section I estimate the cost of doing progressive radiosity in Pxpl5 based on hemicubes and hemispheres. Suppose the scene is divided into M polygon patches and N "brightest" patches are to be used. I divide the calculation into several steps:

(1) Each broadcast of the M primitives for scan conversion, z-buffering, and ID storage at the pixels is roughly comparable to the cost of complete rendering. Using the estimate of 10^6 polygons per second, each time the polygonal database is broadcast, it will take $\frac{M}{1,000,000}$ seconds. For hemicubes, the database needs to be broadcast 5 times for each bright patch, once for each of the 5 faces. The hemisphere, on the other hand, requires only 1 broadcast of the data base for each bright patch.

(2) The polygon ID numbers need to be returned from the renderers to the Graphics Processors for computation of form factors, etc. Assuming a renderer size of 128×128, and a transmission rate of 80 million words per second (4 channels on the ring), this cost is $\frac{128^2}{80,000,000}$ seconds for each broadcast of the database.

In summary:

Hemicube	$5(\frac{MN}{1,000,000} + \frac{128^2}{80,000,000})$	seconds
Hemisphere	$(\frac{MN}{1,000,000} + \frac{128^2}{80,000,000})$	seconds

If the bottleneck is the transmission of data to the renderers (and not the GP's computing coefficients) this indicates that a 5-fold speedup might be attained.