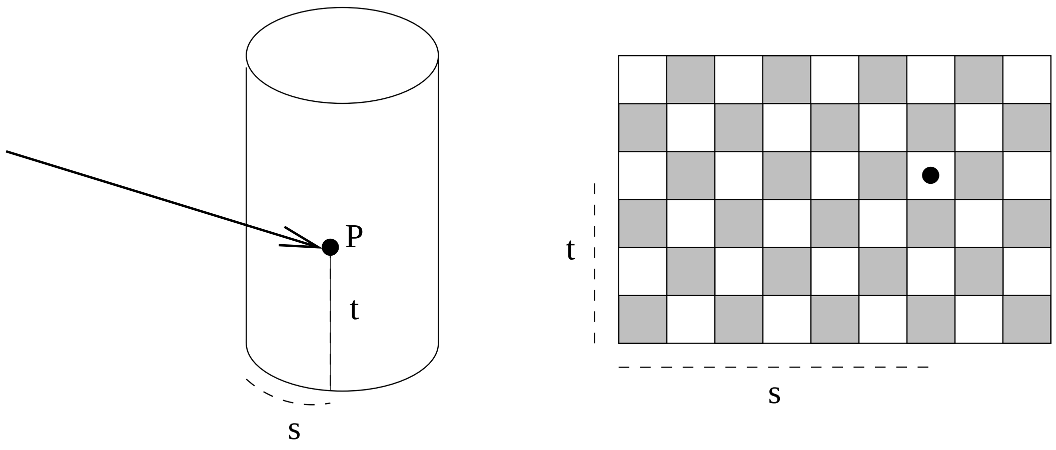
Texture Mapping

Set the diffuse and specular surface properties using a **texture map**:

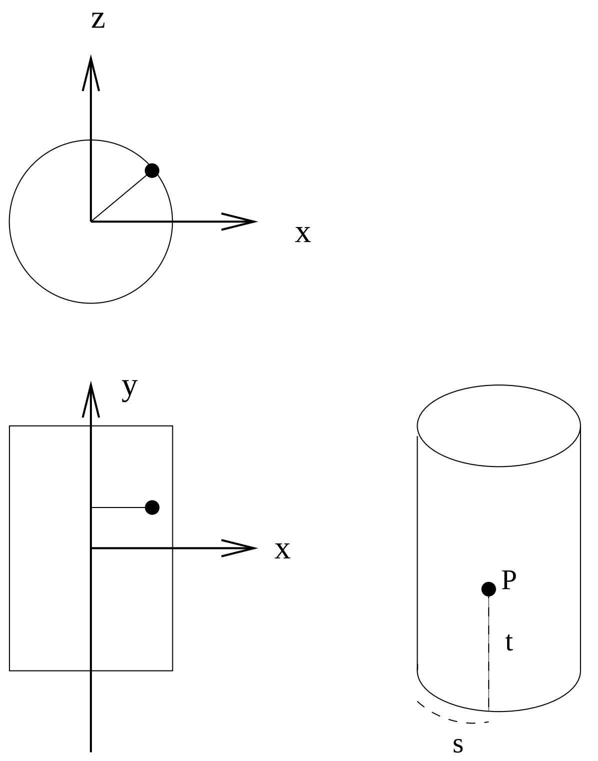


- **Ray strikes cylinder at P.**
- **Map P to parametric coordinates (s, t).**
  - s coordinate is angle in radians from $x = 0$.
  - t coordinate is height from the bottom.
- **Look up surface properties in texture map at location (s, t).**

Parametric Coordinates for Objects

- **Construct parametric equation for the surface of the object.**
  This relates (x, y, z) and (s, t).
- **Solve equations for (s, t) given (x, y, z).**

Parametric Equation for Cylinders



Parametric Equation:

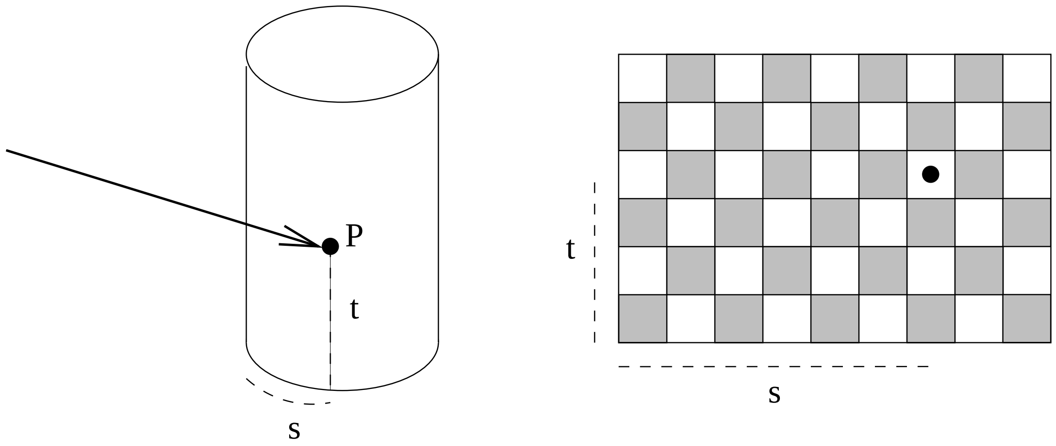
$$
\begin{aligned}
x &= \cos 2\pi s \\
y &= 2t - 1 \\
z &= \sin 2\pi s
\end{aligned}
$$

(s, t) range between 0 and 1.

Solve for (s, t):

$$
\begin{aligned}
s &= \frac{1}{2\pi} \arccos x \\
t &= (y + 1)/2
\end{aligned}
$$

Computing The Texture



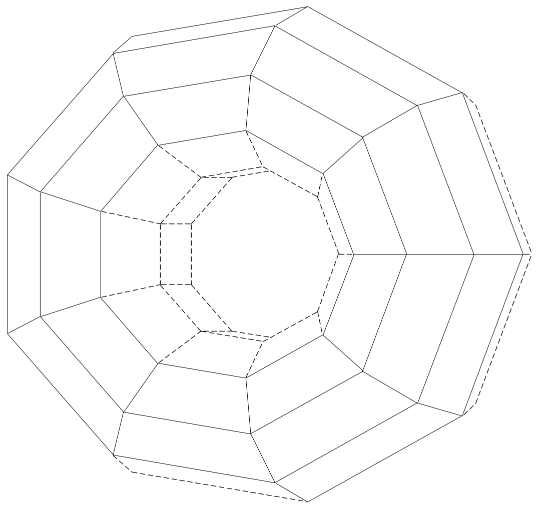
Given an intersection point $P = (x, y, z)$, we want to compute (s, t):

- Apply arbitrary-to-unit transformation.
- $s = \frac{1}{2\pi} \arccos x$
- $t = (y + 1)/2$

A unit square texture map will be mapped onto the side of the cylinder.

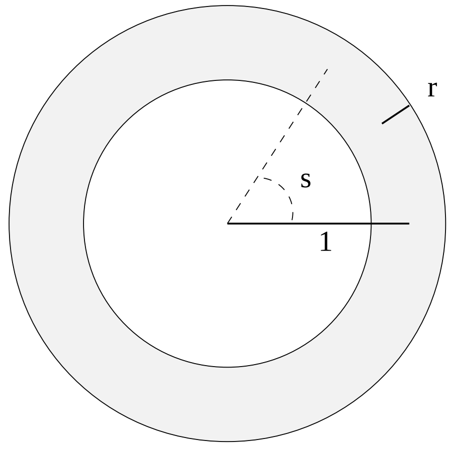
To avoid distorting the texture, we may wish to scale s and t.

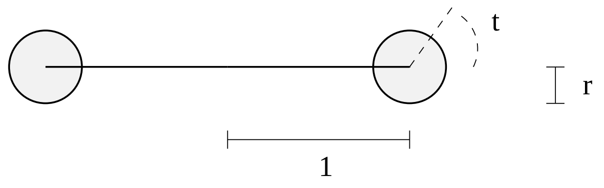
Parametric Coordinates for Toruses



- **Parameter s will be angle taken from center of hole.**
- **Parameter t will be angle taken from center of ring.**

Parametric Equation for Torus

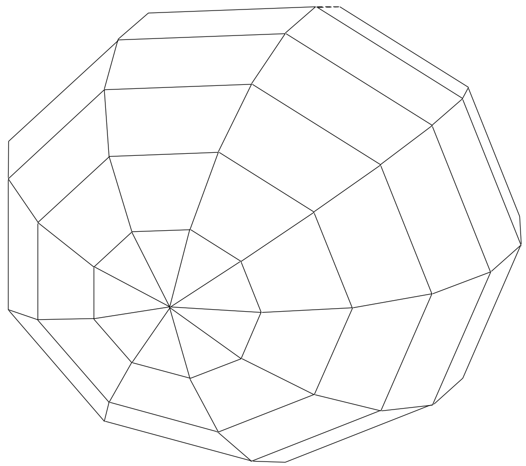




$$
\begin{aligned}
x &= (1 + r\cos 2\pi t)\cos 2\pi s \\
y &= r \sin 2\pi t \\
z &= (1 + r\cos 2\pi t)\sin 2\pi s
\end{aligned}
$$

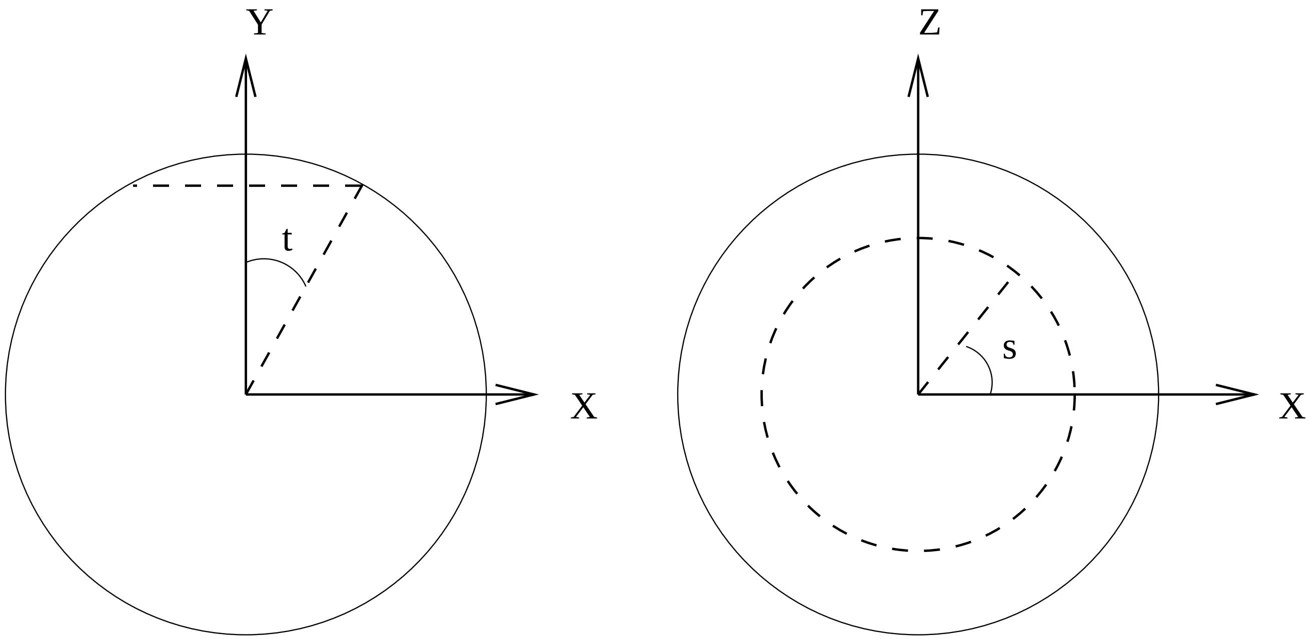
$$
\begin{aligned}
t &= \frac{\arcsin(y/r)}{2\pi} \\
s &= \frac{\arccos(x/(1 + r\cos 2\pi t))}{2\pi}
\end{aligned}
$$

Parametric Coordinates for Spheres



- **Parameter s will be longitude.**
- **Parameter t will be latitude.**

Parameteric Equation for Sphere



$$
\begin{aligned}
x &= \sin \pi t \cos 2\pi s \\
y &= \cos \pi t \\
z &= \sin \pi t \sin 2\pi s
\end{aligned}
$$

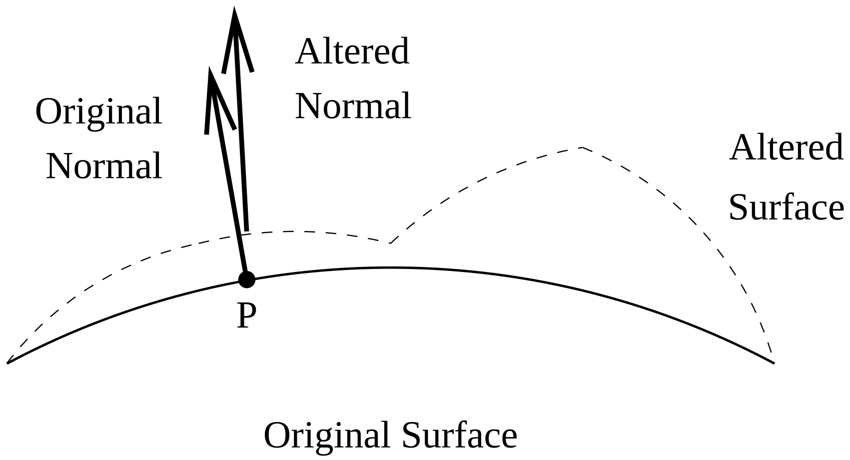
$$
\begin{aligned}
t &= \frac{1}{\pi} \arccos y \\
s &= \frac{1}{2\pi} \arccos\left(\frac{x}{\sin \pi t}\right)
\end{aligned}
$$

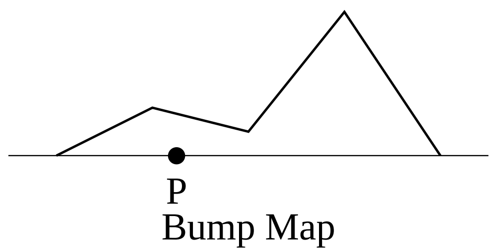
Bump Mapping

Texture mapping changes the color, diffuse, and specular reflection properties, but it does not change the surface normal.

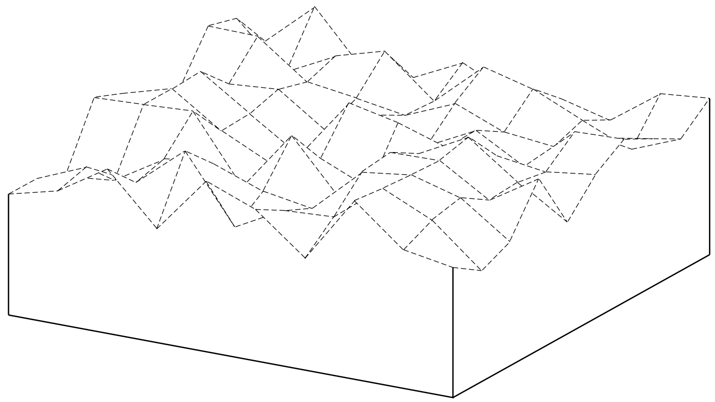
The goal of bump mapping is to perturb the surface normal slightly.

- **Determine where ray hits surface**
- **Map bumps onto surface at hit point**
- **Compute new surface normal at hit point**





Generating a Bump Map



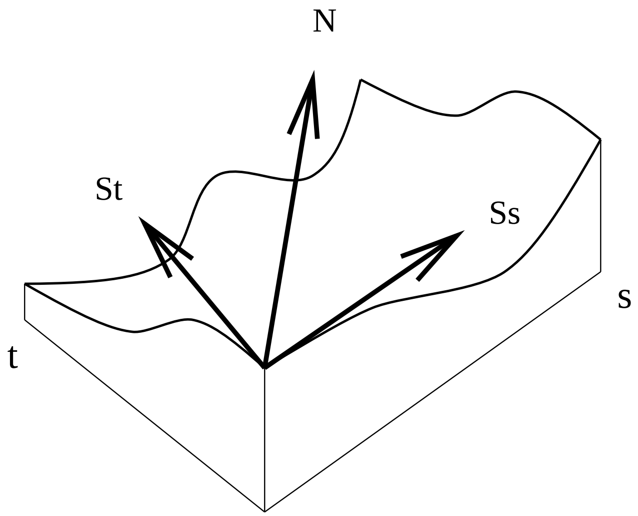
- **Random Bump Map.**
  Choose small random values on a grid—simulates a rough surface such as plaster, concrete, orange.
- **Regular Bump Map.**
  Systematically generate values on a grid—simulates the skin of a strawberry, surface ridges on manufactured surface.
- **Scanned Bump Map.**
  Bump maps can be scanned from real objects using a 3-D range scanner.
  Image intensity values from a scanned image can be converted into elevations. (works well for raised lettering.)

Computing Surface Normal for a Parametric Surface

Let $(x, y, z) = \mathbf{S}(s, t)$ be a parametric surface.

To compute its surface normal, we can compute

$$
\begin{aligned}
\mathbf{S}_s(s,t) &= \frac{\partial \mathbf{S}(s,t)}{\partial s} \\
\mathbf{S}_t(s,t) &= \frac{\partial \mathbf{S}(s,t)}{\partial t} \\
\mathbf{N}(s,t) &= \mathbf{S}_t(s,t) \times \mathbf{S}_s(s,t) \\
\mathbf{n}(s,t) &= \mathbf{N}(s,t)/\|\mathbf{N}(s,t)\|
\end{aligned}
$$

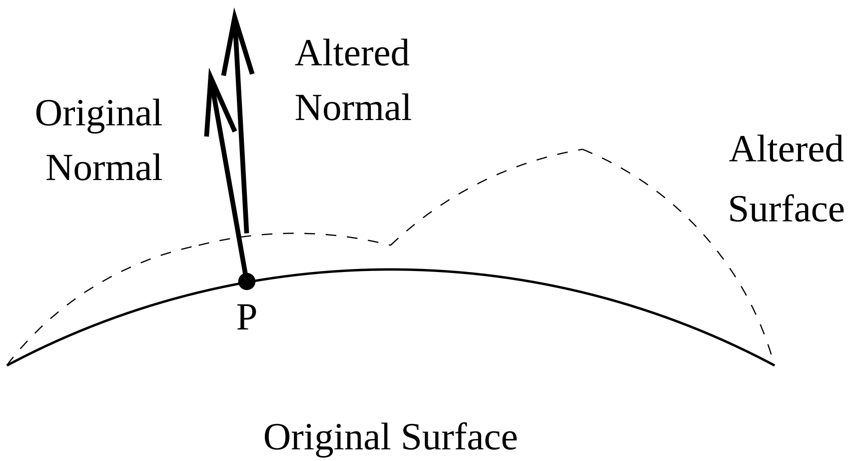


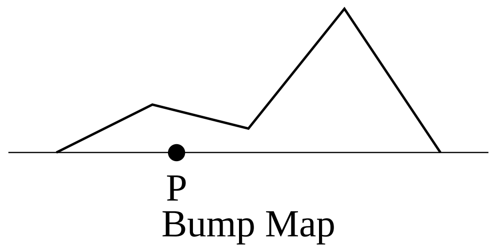
Computing Altered Surface

Let $B(s,t)$ be the bump map value.

We add this amount in the direction normal to $\mathbf{S}(s,t)$:

$$\mathbf{S}'(s,t) = \mathbf{S}(s,t) + B(s,t) \cdot \mathbf{n}(s,t)$$





Computing Altered Surface Normal

Altered surface is

$$\mathbf{S}'(s,t) = \mathbf{S}(s,t) + B(s,t)\cdot \mathbf{n}(s,t)$$

Compute partial derivatives:

$$\begin{aligned}
\mathbf{S}'_s &= \frac{\partial \mathbf{S}'(s,t)}{\partial s} = \mathbf{S}_s(s,t) + B_s(s,t)\cdot \mathbf{n}(s,t) + \mathbf{n}_s(s,t)\cdot B(s,t) \\
\mathbf{S}'_t &= \frac{\partial \mathbf{S}'(s,t)}{\partial t} = \mathbf{S}_t(s,t) + B_t(s,t)\cdot \mathbf{n}(s,t) + \mathbf{n}_t(s,t)\cdot B(s,t)
\end{aligned}$$

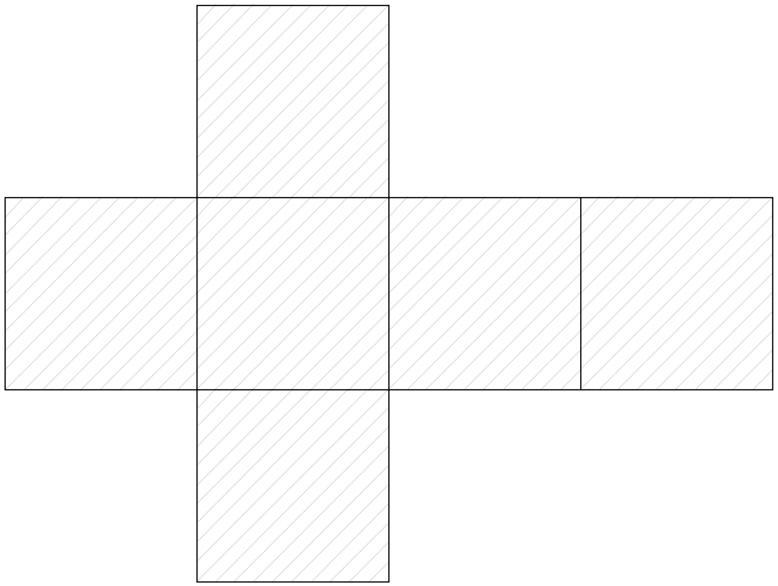
We ignore the last term in each partial derivative and take the cross-product.
Note that $(A+B)\times(C+D) = A\times C + A\times D + B\times C + B\times D$:

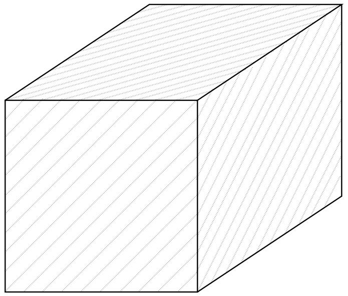
$$\begin{aligned}
\mathbf{N}'(s,t) &= \mathbf{S}'_t(s,t)\times\mathbf{S}'_s(s,t) \\
&= [\mathbf{S}_t(s,t) + B_t(s,t)\cdot\mathbf{n}(s,t)]\times[\mathbf{S}_s(s,t) + B_s(s,t)\cdot\mathbf{n}(s,t)] \\
&= \mathbf{S}_t(s,t)\times\mathbf{S}_s(s,t) + \mathbf{S}_t(s,t)\times(B_s(s,t)\cdot\mathbf{n}(s,t)) + \\
&\quad (B_t(s,t)\cdot\mathbf{n}(s,t))\times\mathbf{S}_s(s,t) + B_s(s,t)B_t(s,t)(\mathbf{n}(s,t)\times\mathbf{n}(s,t)) \\
&= \mathbf{N}(s,t) + B_s(s,t)(\mathbf{S}_t(s,t)\times\mathbf{n}(s,t)) + B_t(s,t)(\mathbf{S}_s(s,t)\times\mathbf{n}(s,t))
\end{aligned}$$

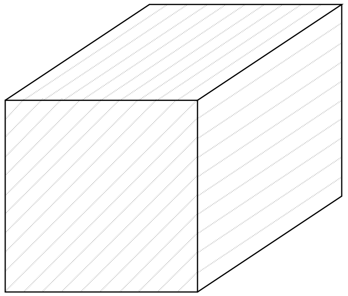
Note that we don't actually compute the altered surface: Only its normal!

Solid Texture

For wood, granite, marble, and other natural materials, simply pasting a texture onto the exterior of the object does not give the desired results:

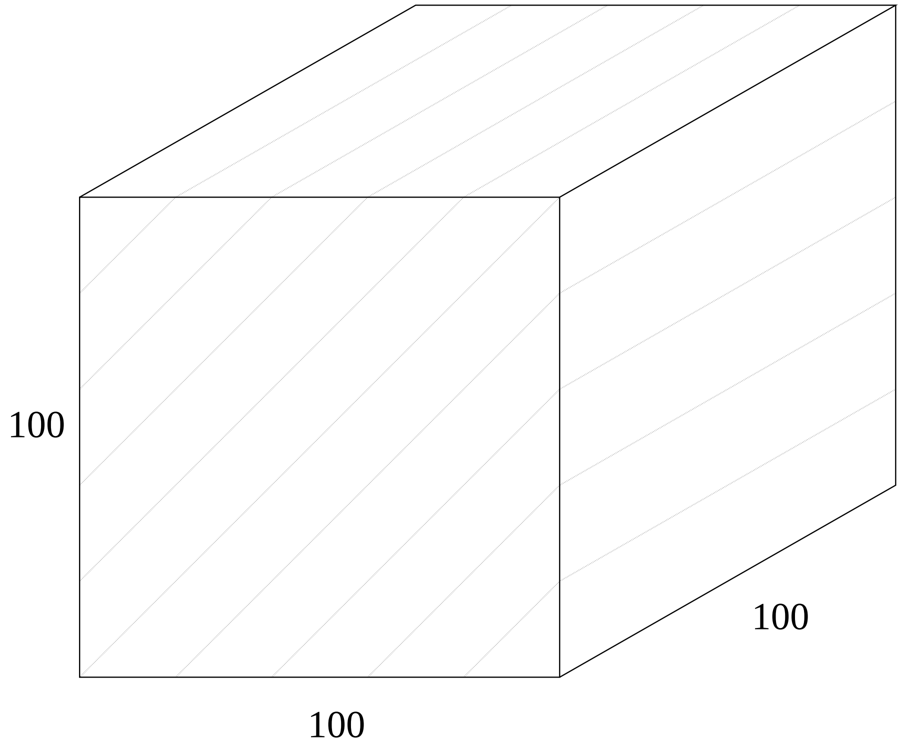






Solid Texture Maps

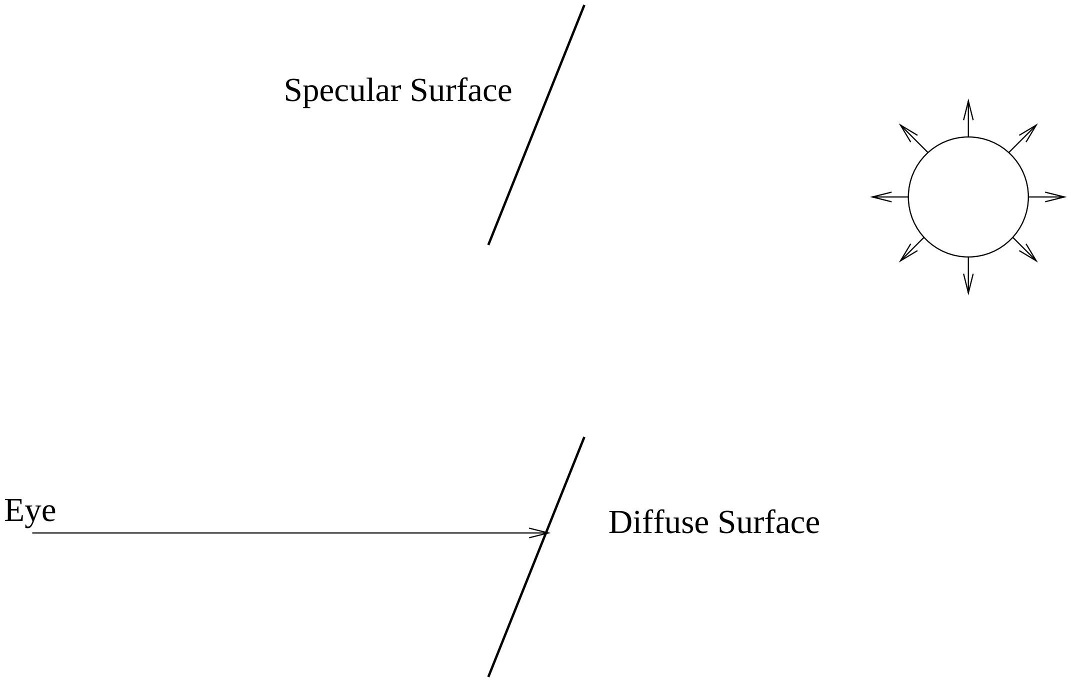
The solution is to construct a 3-D solid texture cube. This is a 3-D array, where each cell contains (R,G,B) values. We imagine that a given 3-D solid object is then "carved" out of this 3-D cube.



We translate and rotate (but do not rescale) the solid object so that it is completely enclosed inside this texture cube. Then we map the hit point and take the average of the 7 RGB values nearest the hit point (or some smoother interpolation).

Radiosity

Ray tracing can't handle the following situation:

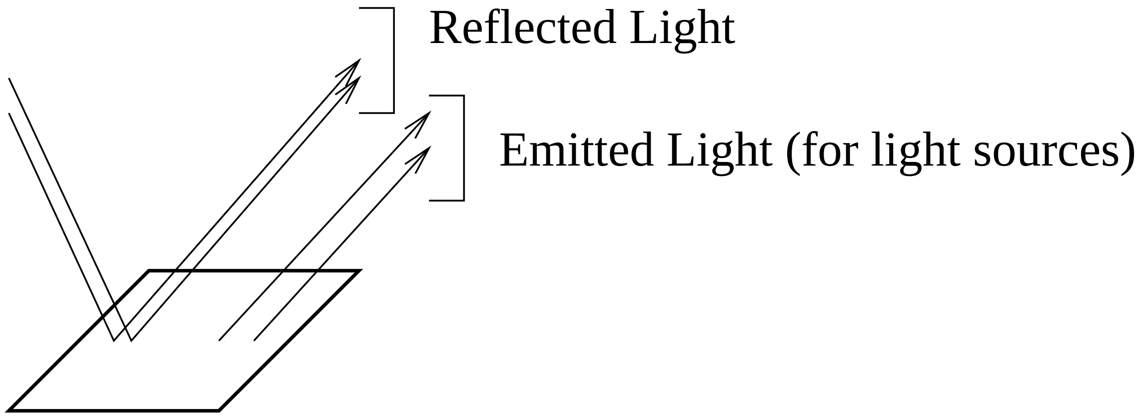


In general, ray-tracing only considers the flow of light along a few rays.

The goal of radiosity methods is to follow the flow of light in all directions simultaneously.

Radiosity (2)

- Divide each surface in the scene (including light sources) into small patches.
- Set up a system of simultaneous radiosity equations:
  B_i = emitted light + reflected light ($\sum_j B_j$).
- Solve the system to compute RGB of each patch.
- Render scene using any method that supports Gouraud shading.



Radiosity (3)

Radiosity = out-going power per unit area (Watt/meter^2)

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

where:

$$\begin{aligned}
B_i &= \text{Radiosity of patch } i \\
B_j &= \text{Radiosity of patch } j \\
E_i &= \text{Light emitted by patch } i \\
\rho_i &= \text{Fraction of incident light that is reflected} \\
F_{j-i} &= \text{“Form factor”: the faction of energy leaving } j \text{ that arrives at } i \\
A_i &= \text{Area of patch } i \\
A_j &= \text{Area of patch } j
\end{aligned}$$

The **form factor** takes into account the shape and relative orientation of the two patches, and the presence of any obstructing patches.

Radiosity (4)

The form factor is symmetric:

$$A_i F_{i-j} = A_j F_{j-i}$$

This simplifies the equation to

$$B_i = E_i + \rho_i \sum_j B_j F_{i-j}$$

or, rearranging terms,

$$B_i - \rho_i \sum_j B_j F_{i-j} = E_i$$

which can be viewed as restating the conservation of energy.

This gives us the following system of simultaneous equations:

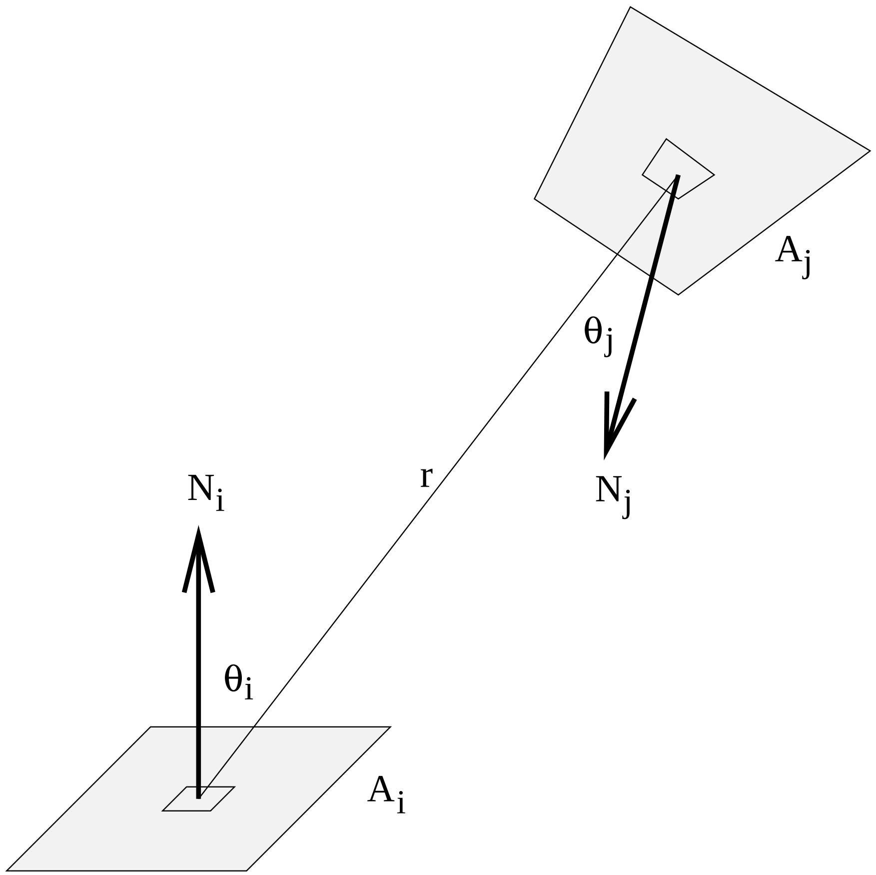
$$\begin{bmatrix} 1-\rho_1 F_{1-1} & -\rho_1 F_{1-2} & \cdots & -\rho_1 F_{1-n} \\ \rho_2 F_{2-1} & 1-\rho_2 F_{2-2} & \cdots & -\rho_2 F_{2-n} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ \rho_n F_{n-1} & -\rho_n F_{n-2} & \cdots & 1-\rho_n F_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ . \\ . \\ . \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ . \\ . \\ . \\ E_n \end{bmatrix}$$

Given the E_i and F_{i-j}, we can solve for the B_i by Gaussian elimination.

Computing the Form Factors

The expensive part of this process is computing the form factors, F_{i-j}. We must compute a form factor for every pair of surface patches.

$$dF_{di-dj} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij} dA_j \qquad H_{ij} = \begin{cases} 1 & \text{if } dA_j \text{ is visible from } dA_i \\ 0 & \text{otherwise} \end{cases}$$



Integrating the Differential Form Factor to get the Form Factor

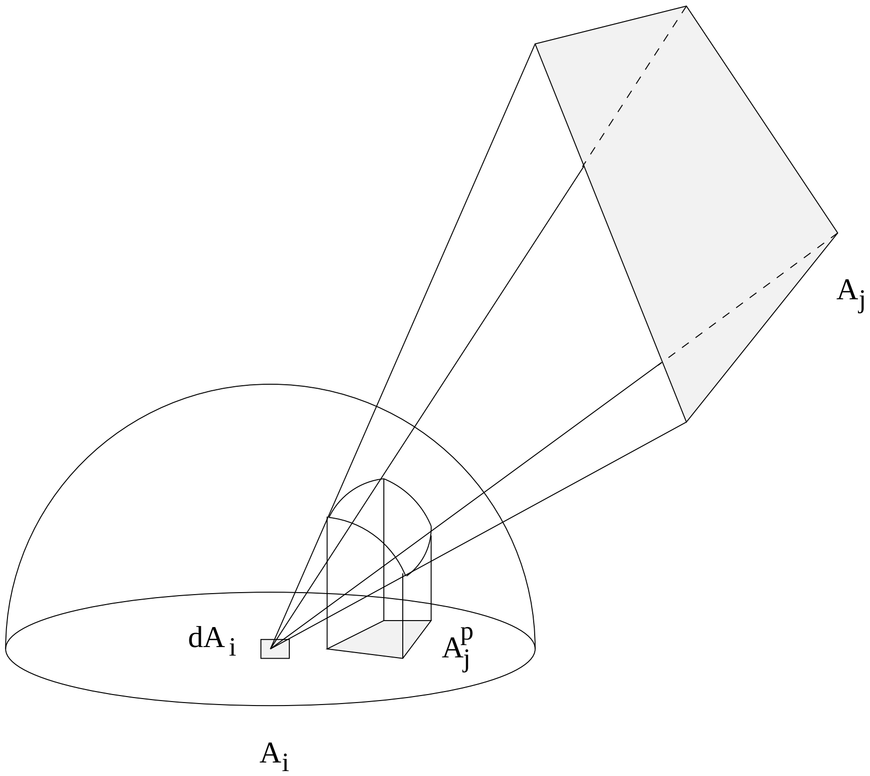
To determine the total energy reaching dA_i from A_j, we must integrate over A_j:

$$F_{di-j} = \int\limits_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j$$

To determine the energy per unit area of A_i, we must average over A_i:

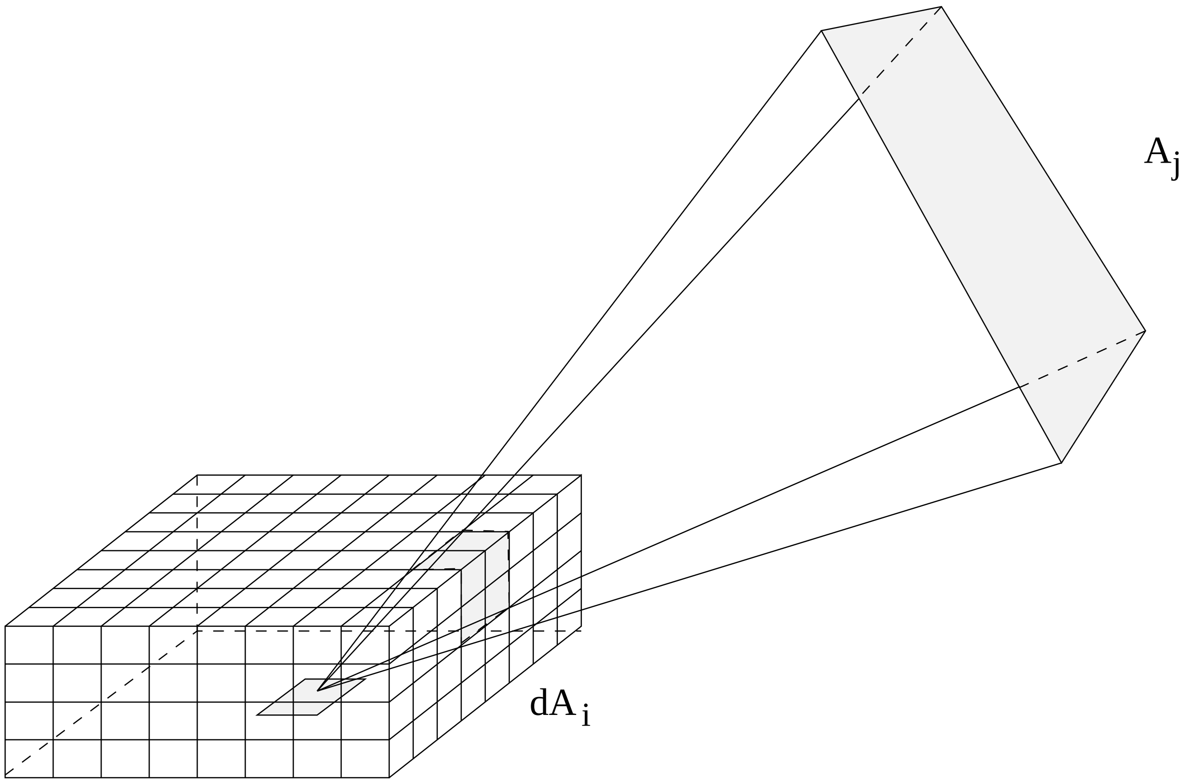
$$F_{i-j} = \frac{1}{A_i} \int\limits_{A_i} \int\limits_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j dA_i$$

A Geometrical Interpretation



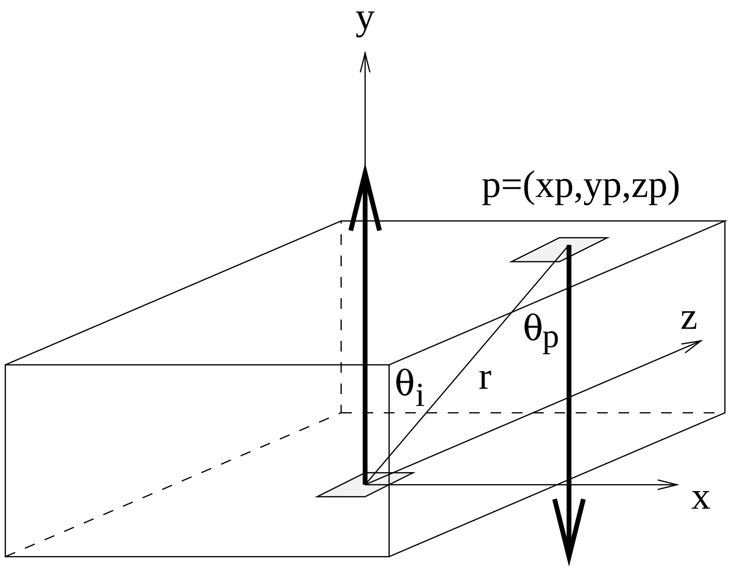
- **Construct unit hemisphere over patch A_i.**
- **Project (perspective) the visible parts of A_j onto this sphere.**
- **Project (orthogonal) the resulting spherical patch onto the base of the hemisphere to obtain A_j^p**
- **Form factor $F_{di-j} = A_j^p/\pi$.**

Approximating the Hemisphere with a Hemicube



- Position unit hemicube over patch.
- Project A_j onto the cells of the hemicube.
- Sum the contribution ΔF_p of each projected cell on hemicube to approximate F_{di-j}.
- Assuming r is much larger than patch sizes, F_{di-j} is a good approximation of F_{i-j}.

Approximating the Hemisphere with a Hemicube (2)



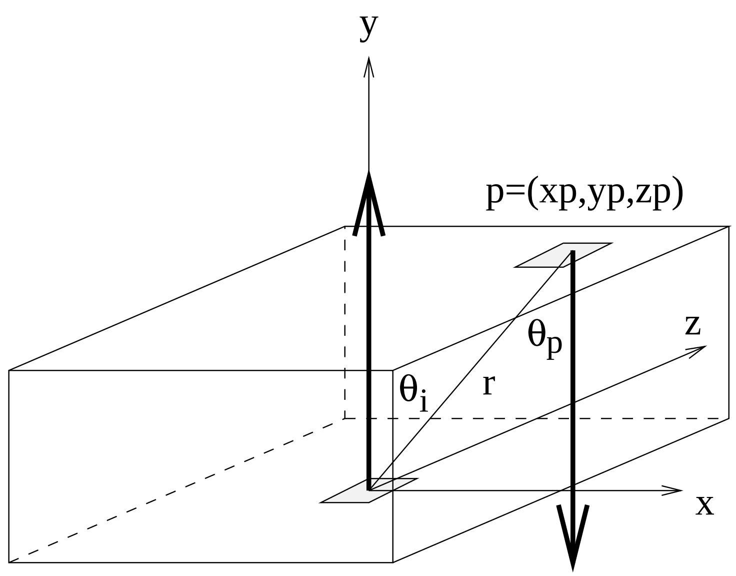
The contribution of each hemicube cell is

$$\Delta F_p = \frac{\cos\theta_i\theta_p}{\pi r^2}\Delta A$$

where:

$$\begin{aligned} r &= \text{Length of line connecting patch center to cell} \\ \theta_i &= \text{Angle between patch normal and line} \\ \theta_p &= \text{Angle between inward cell normal and line} \\ \Delta A &= \text{Area of cell} \end{aligned}$$

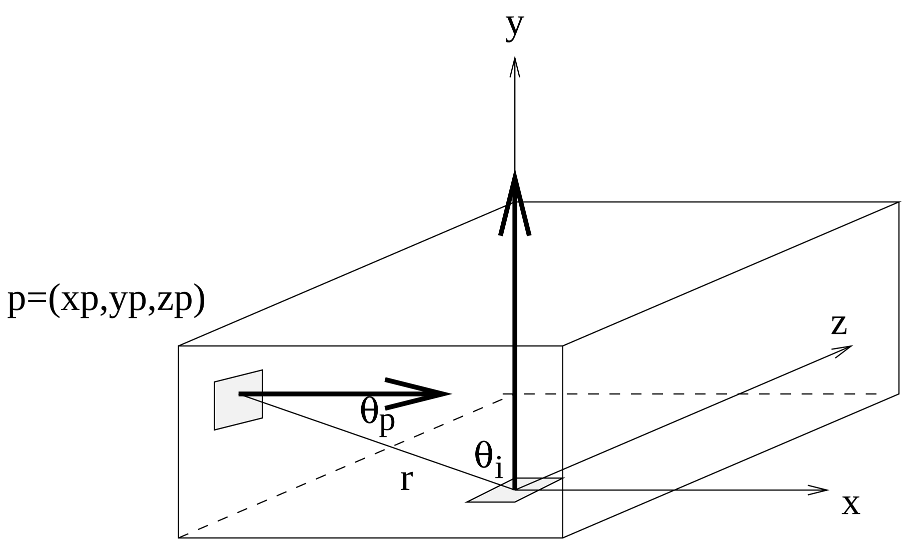
Approximating the Hemisphere with a Hemicube (3)



For cells on the top, the coordinates of P are $(x_p, 1, z_p)$

$$
\begin{aligned}
r &= \sqrt{x_p^2 + 1 + z_p^2} \\
\cos\theta_i &= (0,1,0) \cdot \frac{1}{r}(x_p, 1, z_p) = \frac{1}{r} \\
\cos\theta_p &= (0,-1,0) \cdot \frac{1}{r}(-x_p, -1, -z_p) = \frac{1}{r} \\
\Delta F_p &= \frac{\cos\theta_i \cos\theta_p}{\pi r^2}\Delta A = \frac{\frac{1}{r}\frac{1}{r}}{\pi r^2}\Delta A \\
&= \frac{1}{\pi r^4}\Delta A = \frac{1}{\pi(x_p^2 + 1 + z_p^2)^2}\Delta A
\end{aligned}
$$

Approximating the Hemisphere with a Hemicube (4)

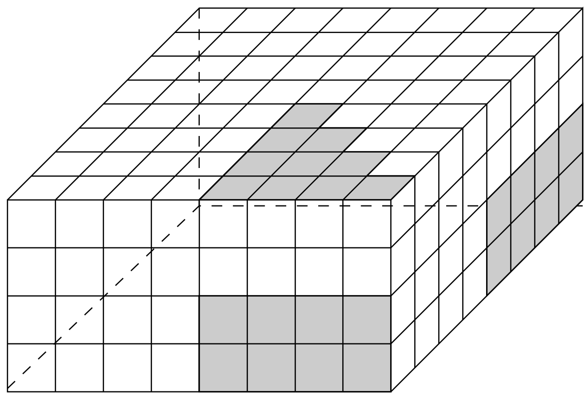


For cells on the left side, the coordiantes of P are $(-1, y_p, z_p)$

$$
\begin{aligned}
r &= \sqrt{1 + y_p^2 + z_p^2} \\
\cos\theta_i &= (0,1,0) \cdot \frac{1}{r}(-1, y_p, z_p) = \frac{y_p}{r} \\
\cos\theta_p &= (1,0,0) \cdot \frac{1}{r}(1, -y_p, -z_p) = \frac{1}{r} \\
\Delta F_p &= \frac{\cos\theta_i \cos\theta_p}{\pi r^2}\Delta A = \frac{\frac{y_p}{r}\frac{1}{r}}{\pi r^2}\Delta A \\
&= \frac{y_p}{\pi(1 + y_p^2 + z_p^2)^2}\Delta A
\end{aligned}
$$

Precomputing the Form Factor Contributions F_p

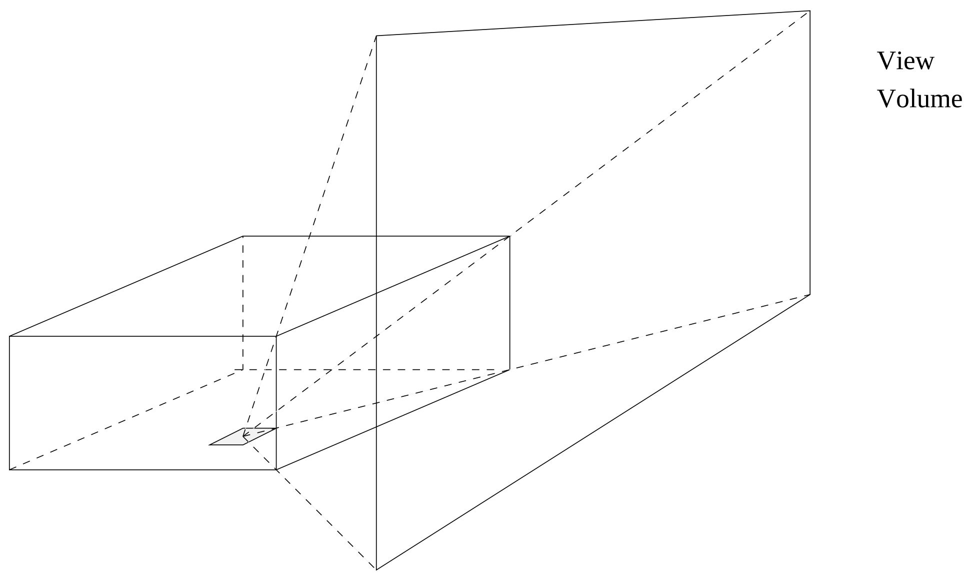
We only need to compute the F_p values once. We can reduce the amount of work by exploiting symmetries.



Item Buffering

```
const int HemiCubeSide = 100;
patch_t * ItemBuf[HemiCubeSide][HemiCubeSide];
float ZBuf[HemiCubeSide][HemiCubeSide];
void InitBufs()
{
  for (int i = 0; i < HemiCubeSide; i++) {
    for (int j = 0; j < HemiCubeSide; j++) {
      ItemBuf[i][j] = NULL;
      ZBuf[i][j] = 1.0;
    }
  }
}
void setpix_item(int x, int y, float z, patch_t * item)
{
  if (ZBuf[x][y] > z) { ZBuf[x][y] = z; ItemBuf[x][y] = item; }
}
```

Efficiently Computing the Form Factors



Consider a single patch and one side of the hemicube. We can treat the center of the patch as if it were the eyepoint, and the side of the hemicube as if it were the view screen. We define an item buffer for the side. Then we render the scene (as in Z buffering), but storing the item buffer information instead of the RGB values.

Finally, we can loop through the item buffer and compute the form factors.

Overall Flow

for each patch i

    for each hemicube side s $\{$

        define item buffer

        render scene using item buffer (skip patches $j < i$)

    $\}$

    for each cell c $\{$

        determine which patch j projected on c

        add its contribution to F_{i-j}

    $\}$

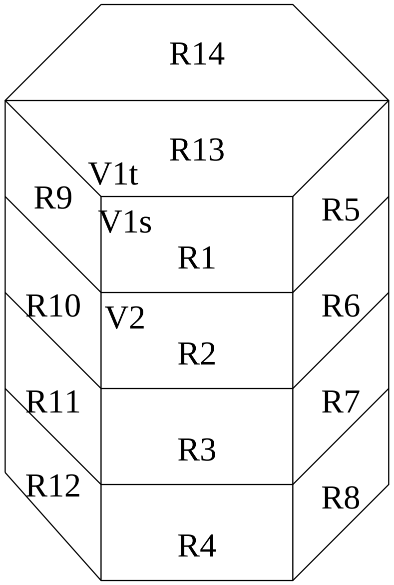
$\}$

solve radiosity equations for red, green, and blue

render scene using depth sorting with interpolated shading

Interpolated Shading for Radiosity

If the patches are fairly large, we need to interpolate the radiosity values to get smooth shading.



- **Compute vertex radiosities (R,G,B)**
- **Apply Gouraud Shading**

$$
\begin{aligned}
R_{V_2} &= \frac{R_1 + R_2 + R_9 + R_{10}}{4} \\
R_{V_{1s}} &= \frac{R_1 + R_9}{2} \\
R_{V_{1t}} &= R_3
\end{aligned}
$$