Global Illumination

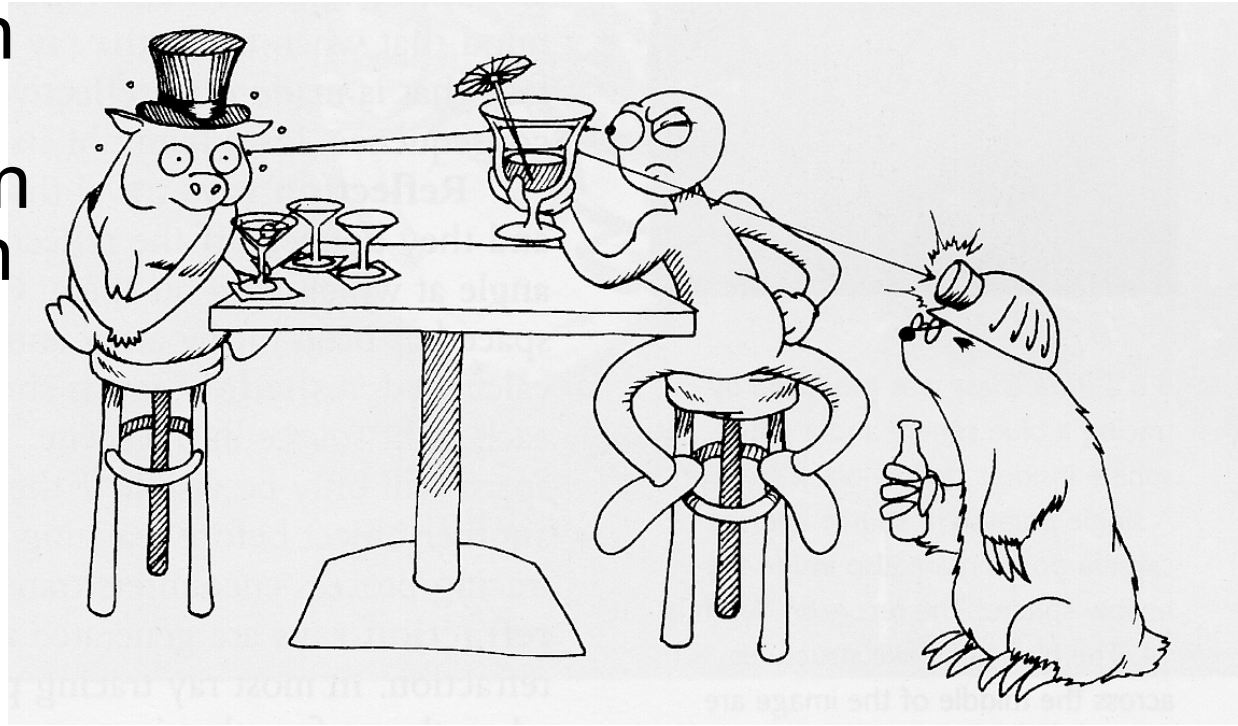
Ray Tracing

- Great at handling reflection, refraction
- Ultra-realistic look

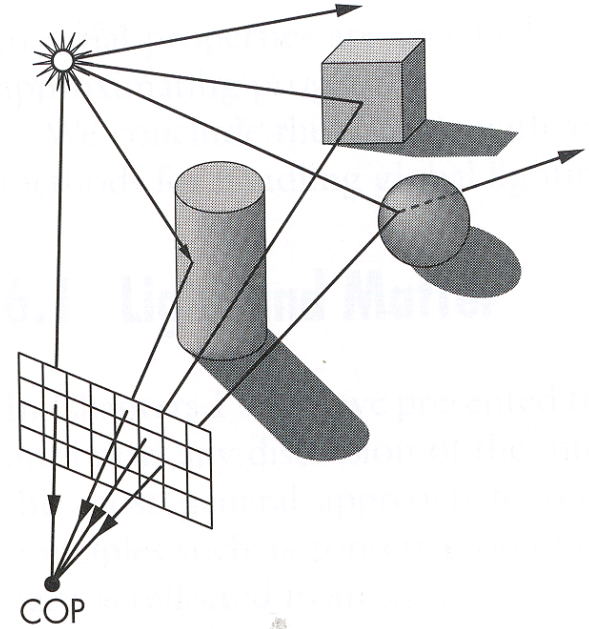
http://www.oyonale.com/modeles.php?lang=en&page=40

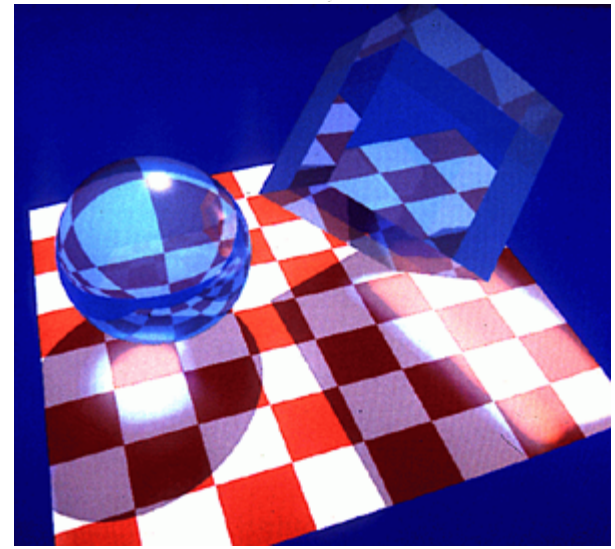
Ray Tracing

- Trace the rays backward starting at eye position to see where it came from
- Forward tracing from lights would mostly trace rays that do not enter camera
- When ray intersects an object, determine illumination contribution then trace more rays in the reflective and refractive direction

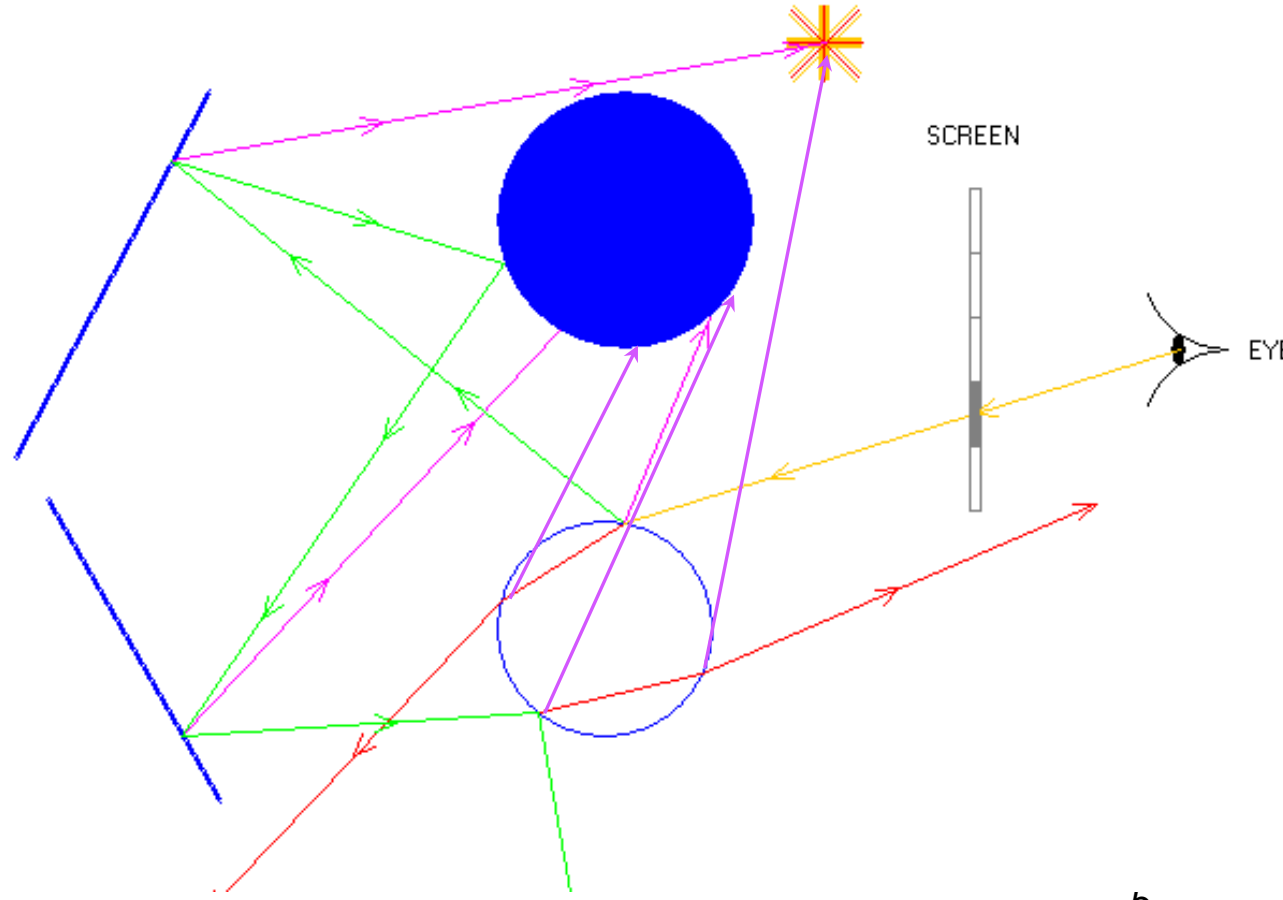


- Rays are shot from the eye position through the pixels into the scene
- Ray tracing accomplishes
  - Global reflection and transmission
  - Detects visible surfaces
  - Identifies shadow areas
  - Transparency effects
  - Does perspective transformation
- Expensive!

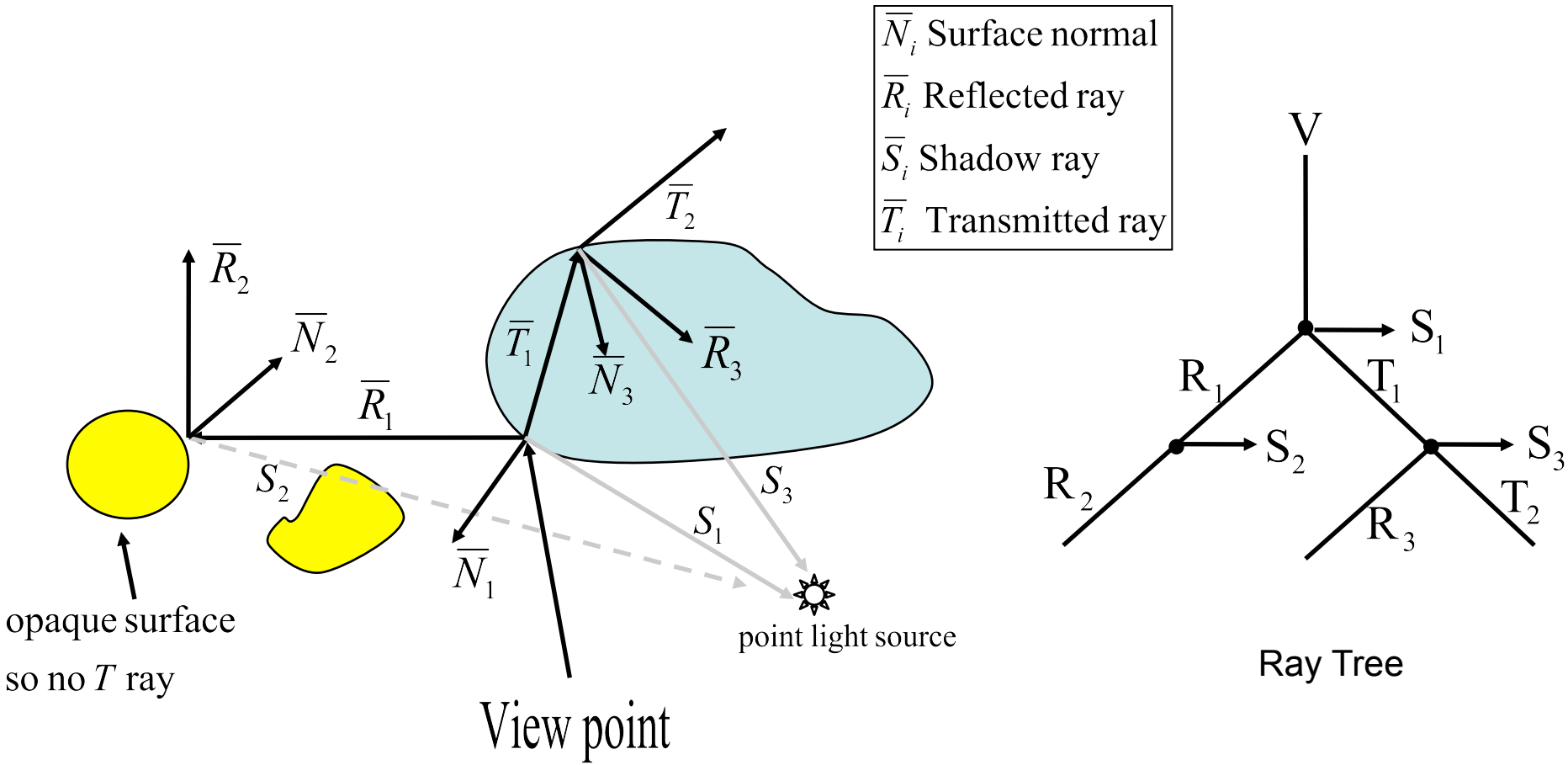




- Yellow - Starting ray
- Purple - Shadow check rays
- Red - Refraction rays
- Green - Reflection Rays



Recursive rays example



Ray/Sphere Intersection

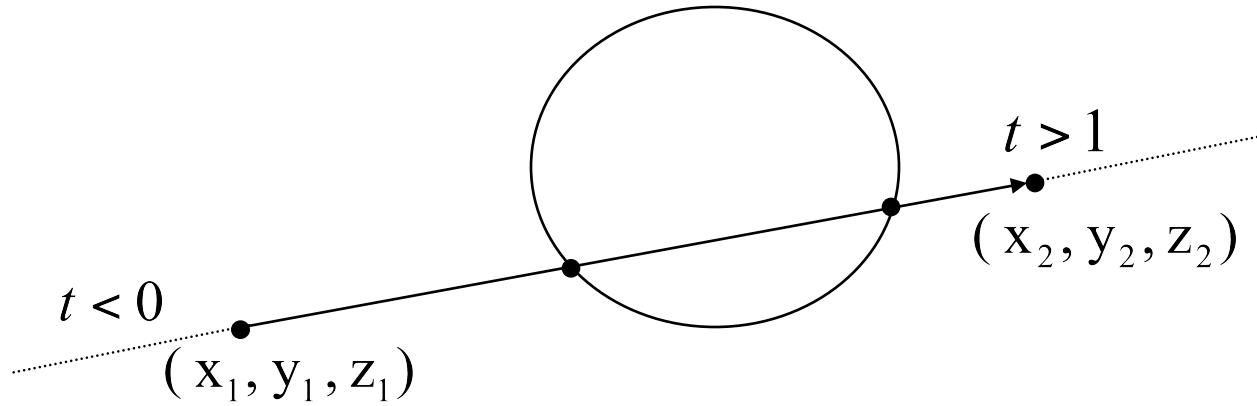
– Ray from (x1, y1, z1) to (x2, y2, z2)

$$x = x_1 + (x_2 - x_1)t = x_1 + it$$

$$y = y_1 + (y_2 - y_1)t = y_1 + jt$$

$$z = z_1 + (z_2 - z_1)t = z_1 + kt$$

$$\mathbf{p} = \mathbf{p}_0 + t\mathbf{u}$$



– Sphere center at (l,m,n) radius r

$$(x-l)^2+(y-m)^2+(z-n)^2=r^2$$

– Substitute $x = x_1 + it \quad y = y_1 + jt \quad z = z_1 + kt$

  • quadratic equation of form $at^2+bt+c=0$

– Find the root (t), substitute into line equation to get intersection point

– Normal at intersection point: $N = (\frac{x_i - l}{r}, \frac{y_i - m}{r}, \frac{z_i - n}{r})$

Ray-Plane Intersection

- Solve for ray-plane intersection
- Do point-in-polygon test to see if point is inside

Plane equation $\mathbf{n} \bullet \mathbf{p} = -d$

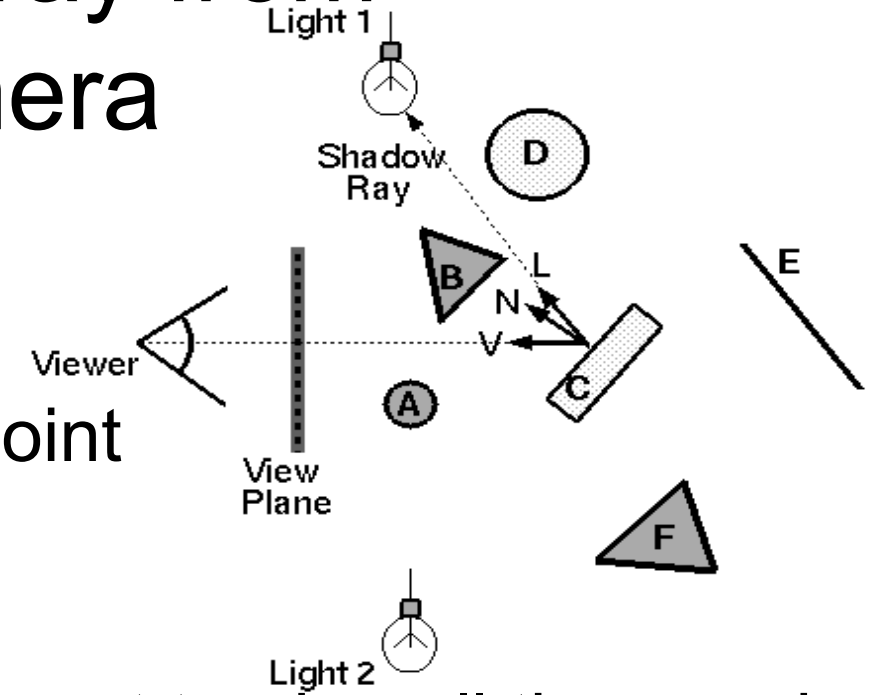
Ray Equation $\mathbf{p} = \mathbf{p}_0 + t\mathbf{u}$

Sub ray into plane $\mathbf{n} \bullet (\mathbf{p}_0 + t\mathbf{u}) = -d$

$$\mathbf{n} \bullet \mathbf{p}_0 + (\mathbf{n} \bullet \mathbf{u})t = -d$$

$$t = -\frac{d + \mathbf{n} \bullet \mathbf{p}_0}{\mathbf{n} \bullet \mathbf{u}}$$

Primary ray from camera

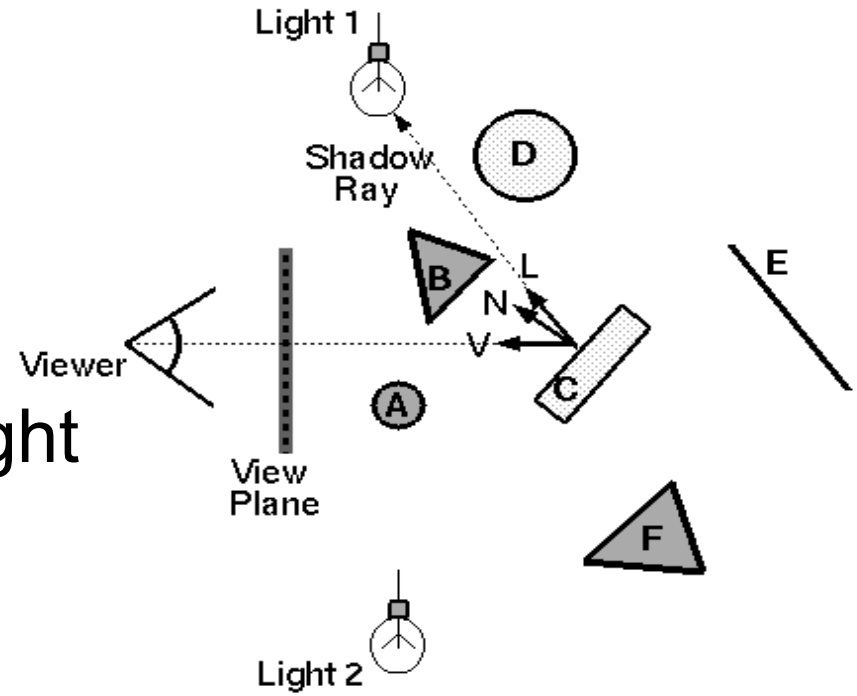


- Direct illumination from unblocked lights only
- S_L determines whether point is in shadow
- Ambient term used as a fudge factor since we are not tracing all the rays in the environment

$$I = I_E + K_A I_A + \sum_L (K_D (N \bullet L) + K_S (V \bullet R)^n) S_L I_L$$

Shadows

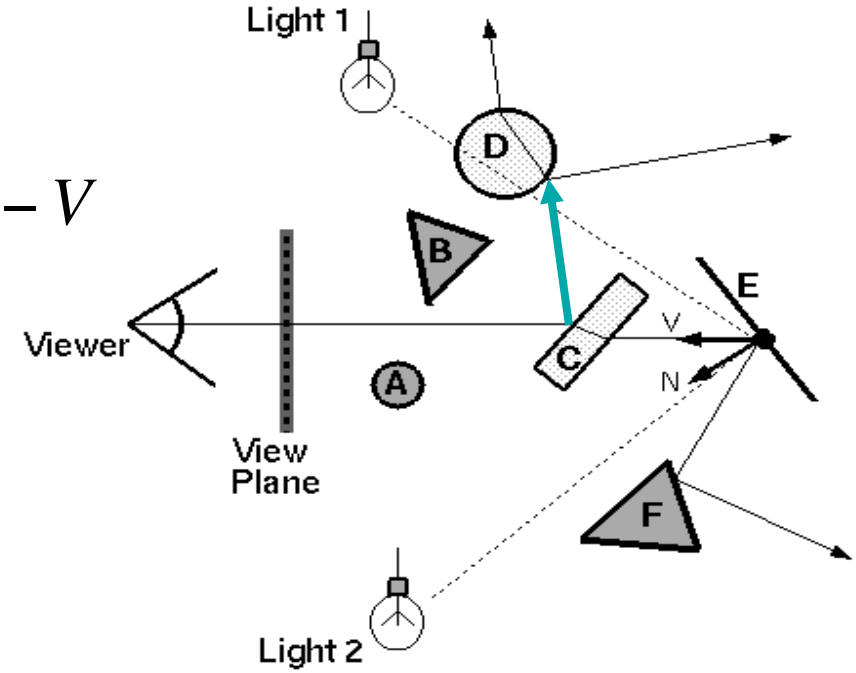
- Shadow term tells if light sources are blocked
- Cast ray towards each light source L
- $S_i = 0$ if ray is blocked, $S_i = 1$ otherwise
- $0 < S_i < 1$ → soft shadows (hack)



$$I = I_E + K_A I_A + \sum_L (K_D (N \bullet L) + K_S (V \bullet R)^n) S_L I_L$$

Mirror reflections

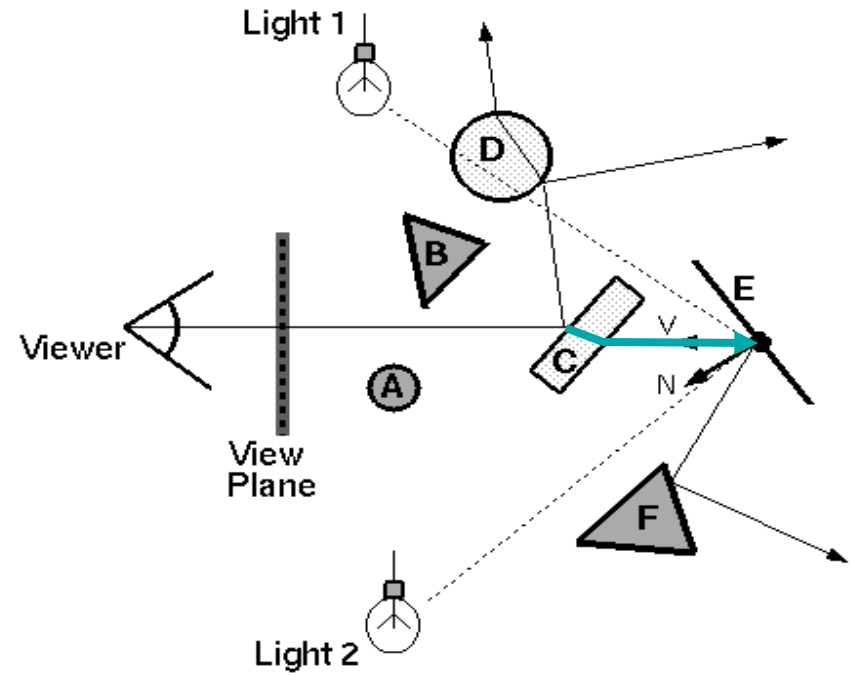
- Trace secondary ray in mirror direction: $R = 2N(N \bullet V) - V$
- Reflection coefficient K_R is fraction reflected
- Evaluate radiance along secondary ray I_R (recursively) and include it into illumination model



$$I = I_E + K_A I_A + \sum_L (K_D (N \bullet L) + K_S (V \bullet R)^n) S_L I_L + K_R I_R + K_T I_T$$

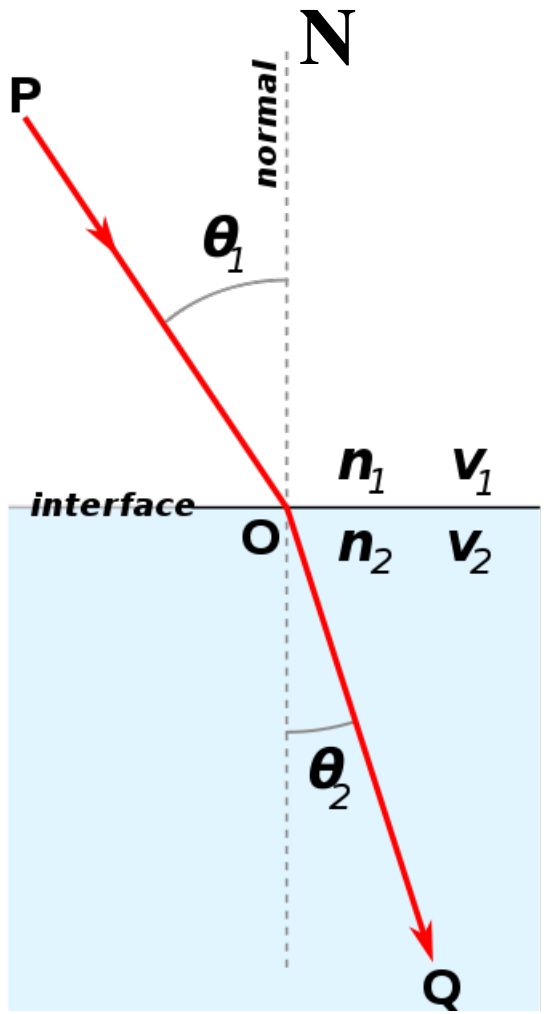
Transparency

- Trace secondary ray in direction of refraction
- Evaluate radiance along refracted ray (recursively) and include it into illumination model
- In the illustration, the internal refraction is ignored (c is thin)



$$I = I_E + K_A I_A + \sum_L (K_D (N \bullet L) + K_S (V \bullet R)^n) S_L I_L + K_R I_R + K_T I_T$$

Refraction



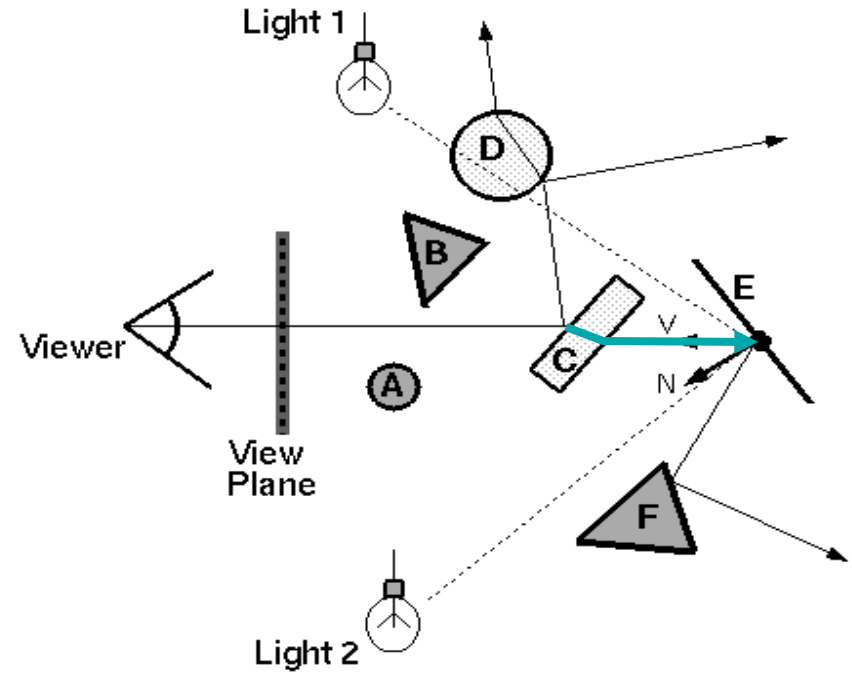
Snell's Law: $\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{\eta_2}{\eta_1}$

$$\cos\theta_1 = l \bullet n$$

$$\cos\theta_2 = \sqrt{1 - \left(\frac{\eta_1}{\eta_2}\right)^2 (1 - \cos^2\theta_1)}$$

$$\mathbf{Q} = \frac{\eta_1}{\eta_2}\mathbf{P} - (\cos\theta_2 - \frac{\eta_1}{\eta_2}\cos\theta_1)\mathbf{N}$$

- Transparency coefficient K_T is fraction transmitted
- $K_T = 1$ for transparent object, $K_T = 0$ for opaque
- $0 < K_T < 1$ for object that is semi-transparent



Transparency Coefficient

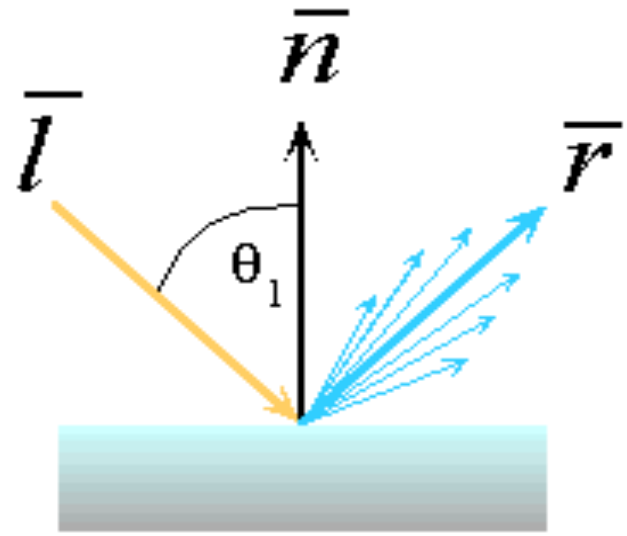
$$I = I_E + K_A I_A + \sum_L (K_D (N \bullet L) + K_S (V \bullet R)^n) S_L I_L + K_R I_R + K_T I_T$$

Illumination at each intersection (recap)

- At each intersection, calculate the intensity given by the equation
- I_r and I_t are calculated recursively
- At each recursion contribution to final illumination of pixel is less and less (why?)
- Recursion terminates at specified depth

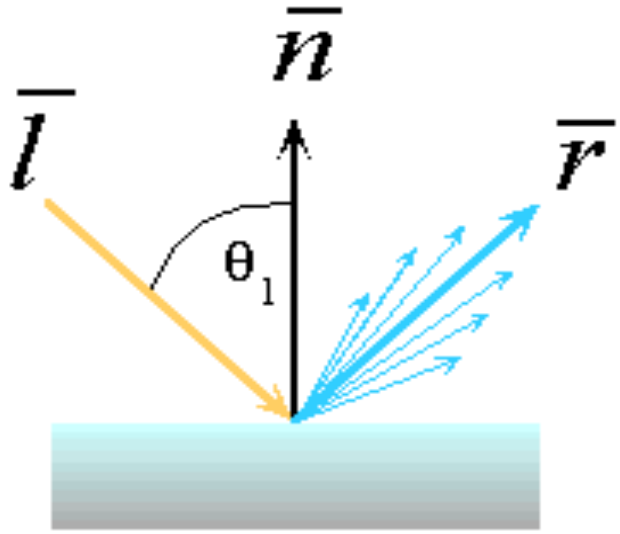
Problems with basic Ray tracing

– Best for ideal specular surfaces (mirror) since trace only one ray in the reflective (and/or refractive) direction
  - Consider the semi-rough surface
  - Light going out in the r direction did not come just from the l direction
  - Must integrate (add) contributions from all directions meaning shoot a number of rays in the reflective direction



Distributed Ray tracing

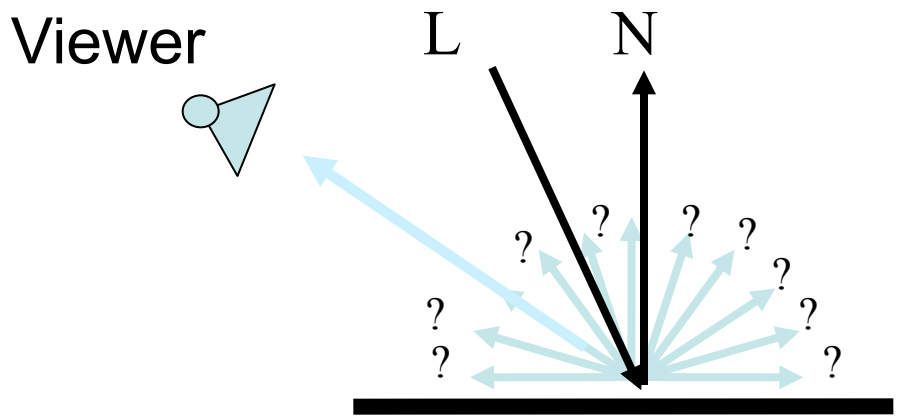
- Shoot rays in all directions and combine results according to weight given by the specularity distribution (known as BRDF)
- Can be done by using numerical integration method called Monte Carlo method
- Notice the soft shadows in the image: shoot many rays toward the area light source
- The integration can be done in other dimensions (like time-motion blur)



Radiosity

– Diffuse surfaces reflect the same in every direction

– $I_{diffuse} = k_d I_{light} \cos \theta = k_d I_{light} (N \bullet L)$



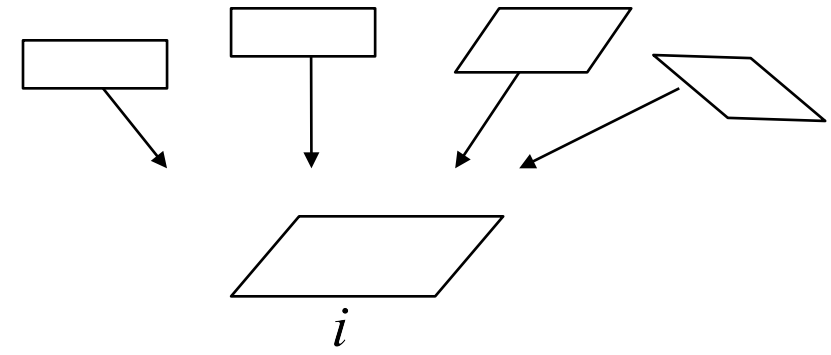
http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_1.htm

http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_3.htm

Radiosity

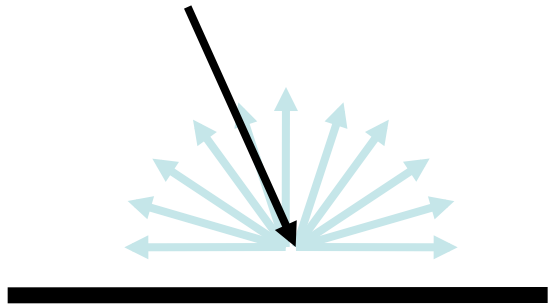
- Divide surfaces into small "patches"
- Calculate light emitted from each patch
- Patches may not be uniform. Areas of large illumination change will need more "detailed" solution

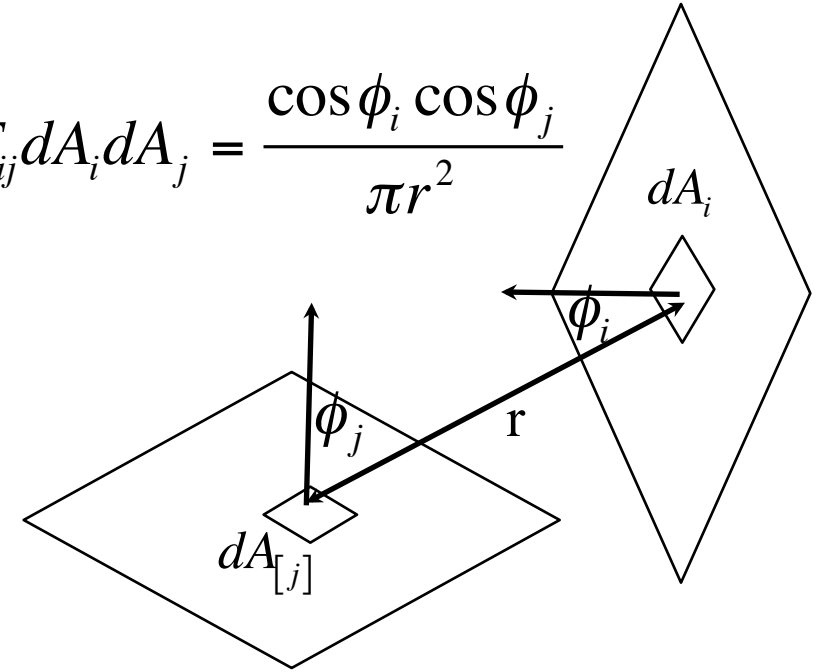
– For each patch write a Radiosity Equation:
  Energy out = Energy emitted by the patch + Energy from all other patches that is reflected (!absorbed) by this patch
– This is a mutually recursive equation
– Solve simultaneous equation



– To make things simple, assume that there is only one energy intensity in all directions for each patch (Lambertian reflectance)

Form Factor between 2 patches: $F_{ij} dA_i dA_j = \frac{\cos\phi_i \cos\phi_j}{\pi r^2}$





Radiosity Equation

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

B_i, B_j – radiosity of i, j

E_i - rate at which light is emitted by i

ρ_i - i's reflectivity(assume constant Lambertian)

$F_{j\text{-}i}$ - form factor - fraction of energy leaving patch j that arrives at patch i (shape, orientation of patches, obstruction)

A_i, A_j - area of i, j

$B_j F_{j\text{-}i}$ – amount of light leaving unit area A_j reaching all of A_i

multiply by $\frac{A_j}{A_i}$ to get light leaving all of A_j reaching unit area of A_i

Radiosity Equation

$$\mathbf{B}_i = \mathbf{E}_i + \rho_i \sum_{1 \le j \le n} \mathbf{B}_j \mathbf{F}_{ji}$$

$\mathbf{B}_i, \mathbf{B}_j$ – radiosity of patchs i,j

$\mathbf{E}_i$ – rate at which light is emitted by i

ρ_i – i's reflectivity (assume constant Lambertian)

$\mathbf{F}_{ji}$ – form factor - fraction of energy leaving patch j that arrives at patch i (shape, patch orientation, obstruction)

$\mathbf{B}_j \mathbf{F}_{ji}$ – amount of light leaving unit area A_j reaching all of A_i

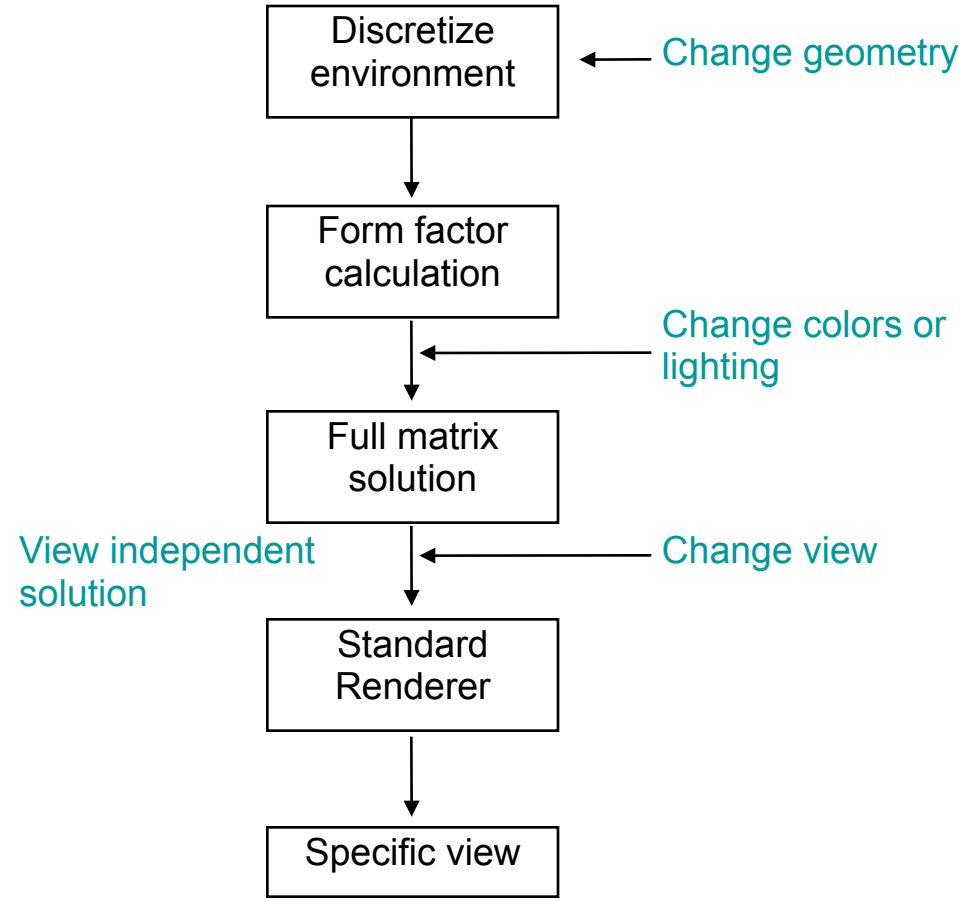
Solve simultaneously

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

- Known: ρ, F, E; solve for B at particular wavelengths
- Gauss-Seidel iteration
- For planes or convex patches, $F_{i\text{-}i} = 0$
- Gouraud or Phong renderer used for view-dependent solution

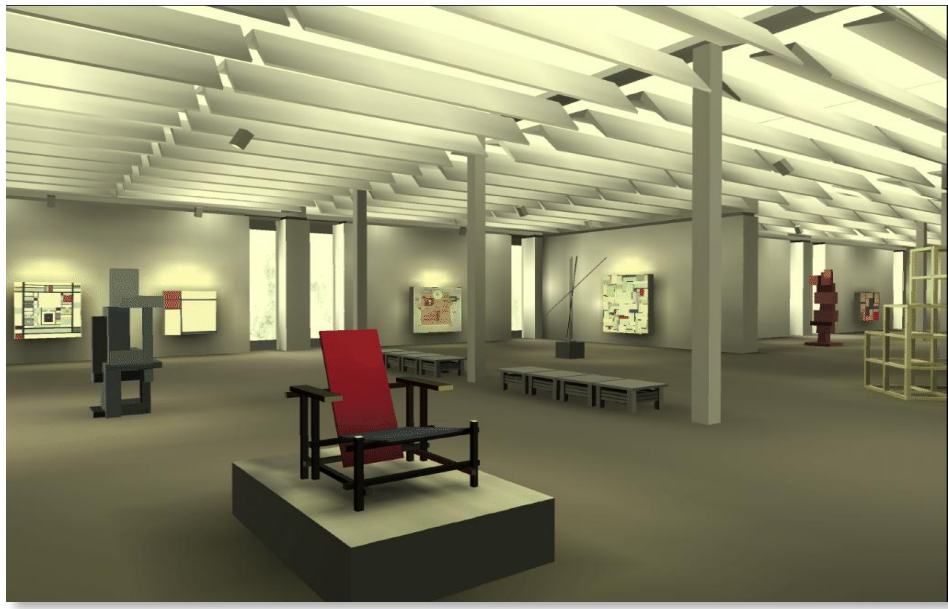
Radiosity rendering pipeline

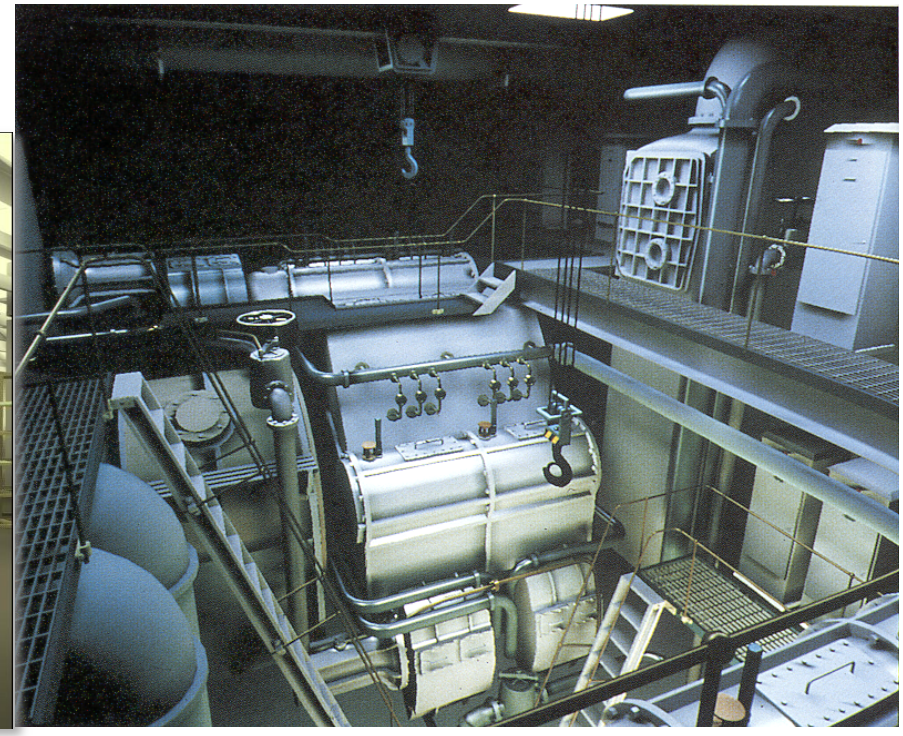
- Up to full matrix solution, viewpoint independent
- Standard renderer is basically Gouraud rendering with the illuminations given by the matrix solution
- This means that once the radiosities are calculated, the scene can be traversed in real time



Examples of Radiosity rendering

- Early one on the left (1988 Cornell), later one on the right
- Notice the nice soft shadows
- No specular surfaces



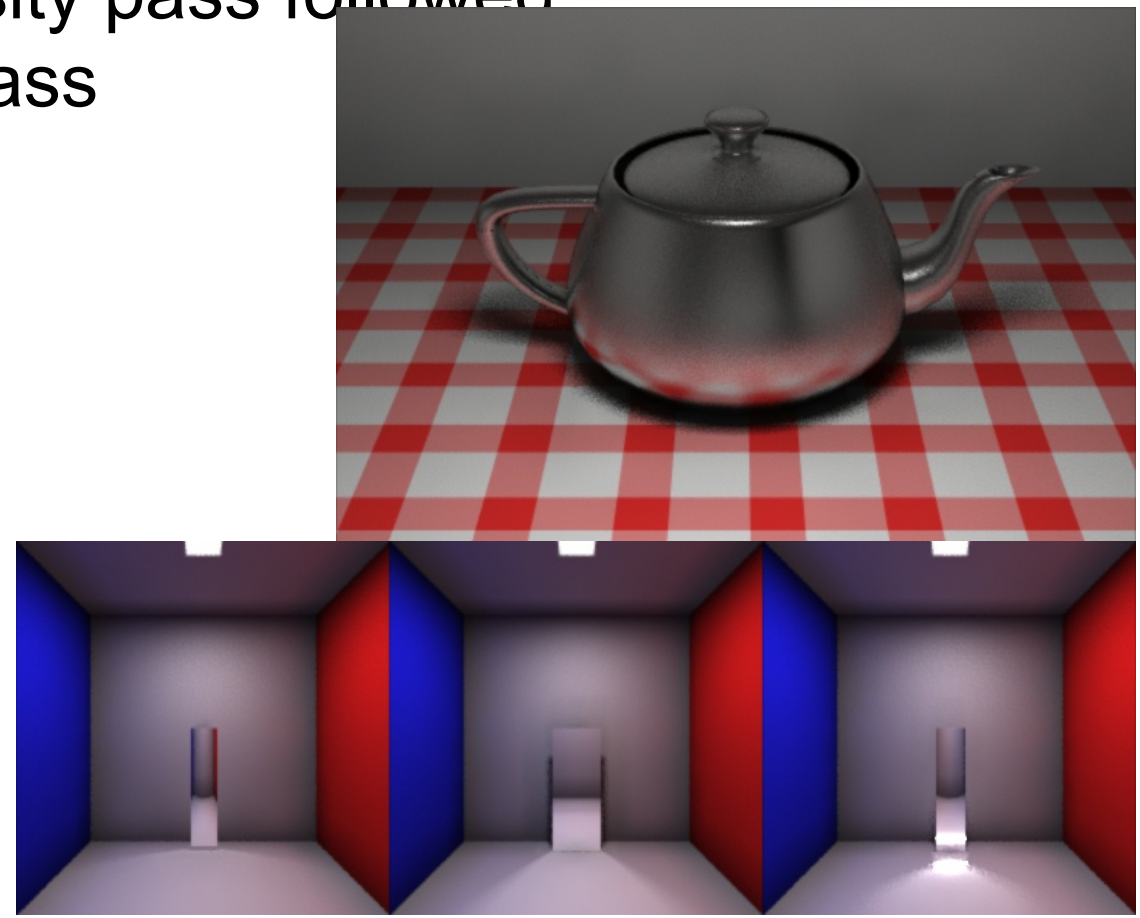


Ray tracing and Radiosity

- Both are global illumination algorithms
- Ray tracing is viewpoint dependent
- Radiosity is viewpoint independent
  - At least the important first part, up to the full matrix solution
- Ray tracing does specular (mirror-like) surfaces well
- Radiosity does perfectly diffuse surfaces well

Combining Ray tracing and Radiosity

– First do the radiosity pass followed by a ray tracing pass

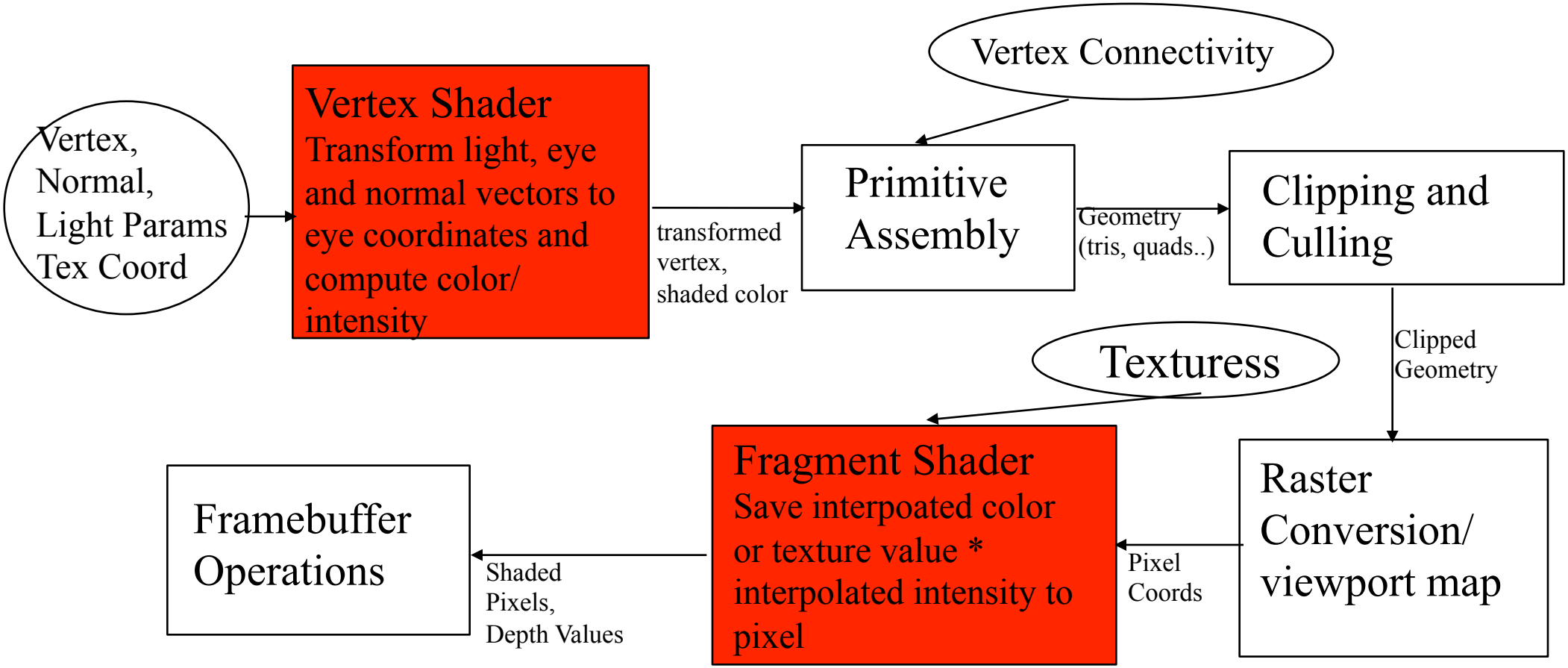


Review of all rendering pipelines

– Gouraud shading
– Phong shading
– Radiosity
– Ray tracing

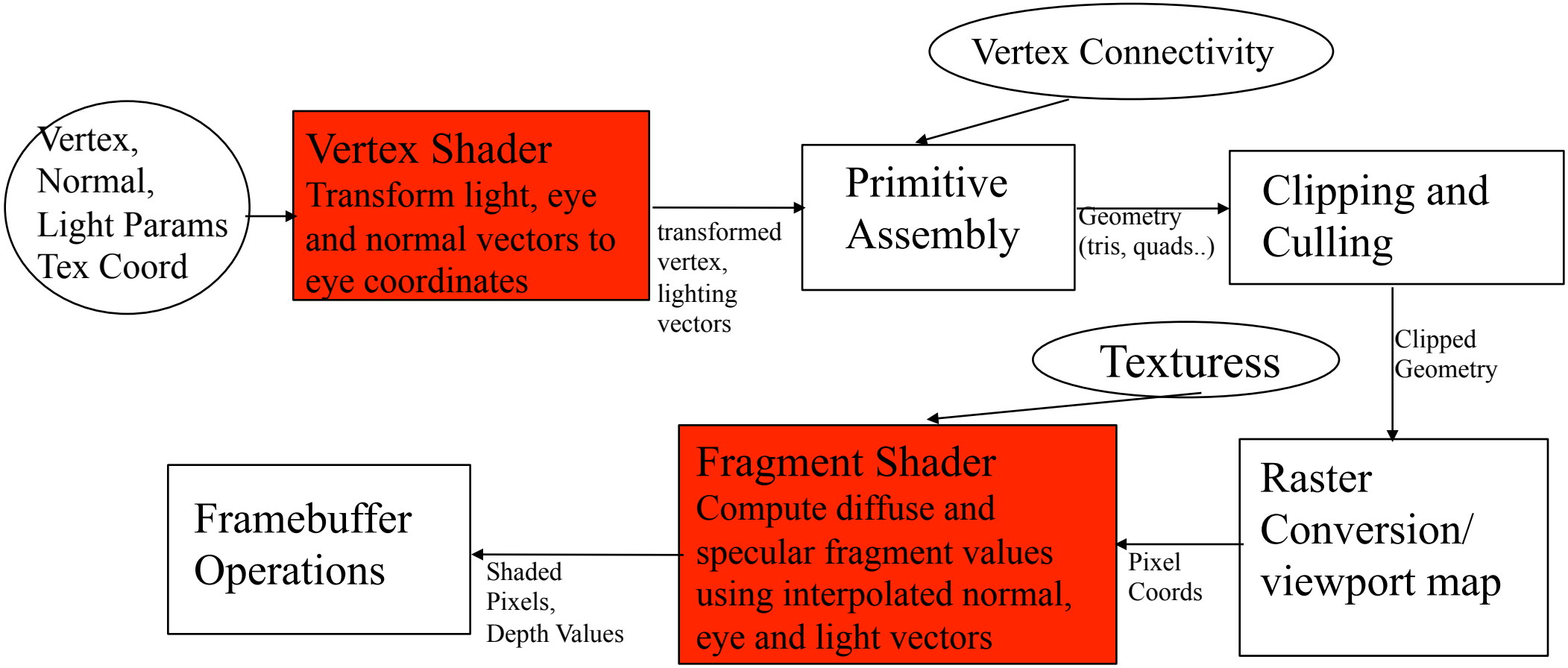
Gouraud shading

- Illumination done at vertices
- Vertex values interpolated at pixels



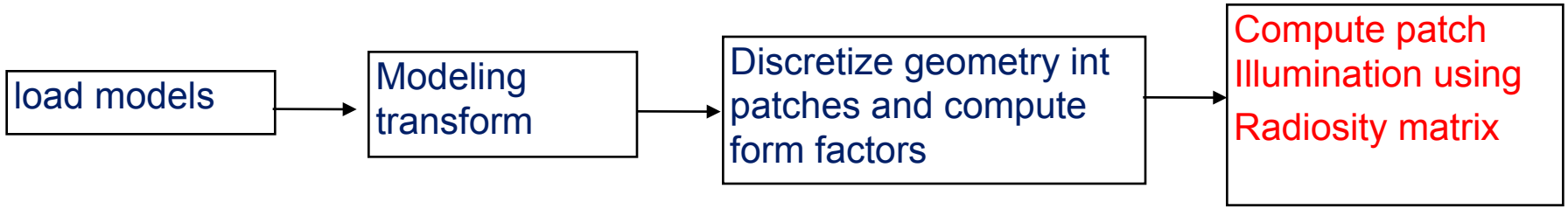
Phong shading

– Illumination calculation done during scan conversion - faster since there are more pixels than vertices

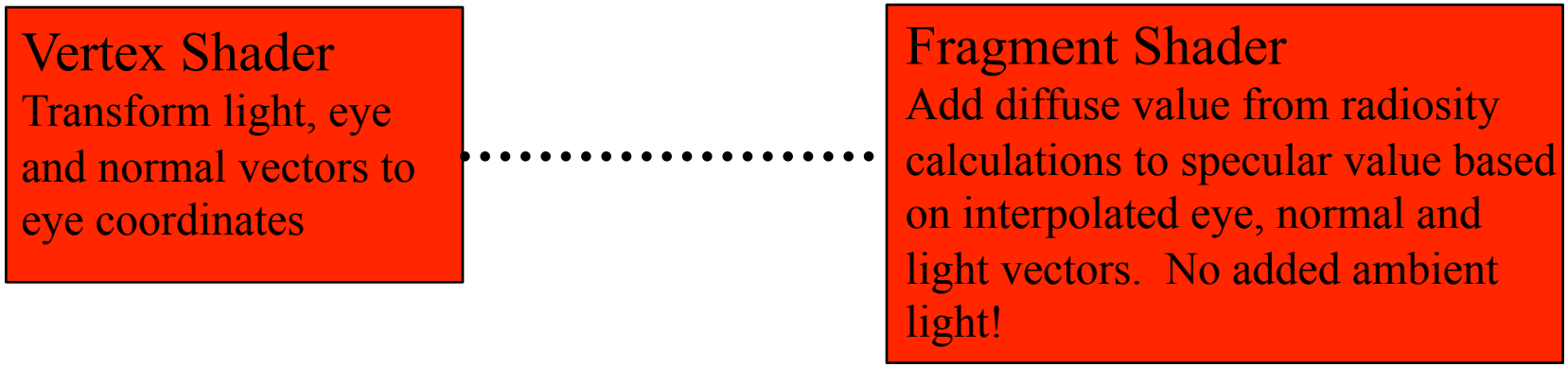


Radiosity, Phong Shading

– Radiosity computes diffuse illumination that we store at vertices or in textures



– Add radiosity-computed values to Phong shading by using the values in place of diffuse and ambient terms



Ray tracing

- Ray tracing includes:
  - Hidden surface removal
  - Illumination
  - Rasterization
  - Perspective x-form

Next: Shaders

OpenGL Shading language specification: - See the specification for version GLSL 3.3 at the API specification page:

http://www.opengl.org/registry/#apispecs

http://www.opengl.org/registry/doc/GLSLangSpec.3.30.6.clean.pdf

Lighthouse tutorial. Look at first 13 pages - OpenGL Setup and Communication

http://www.lighthouse3d.com/opengl/glsl/

A good book (but not required)

http://www.3dshaders.com/home/