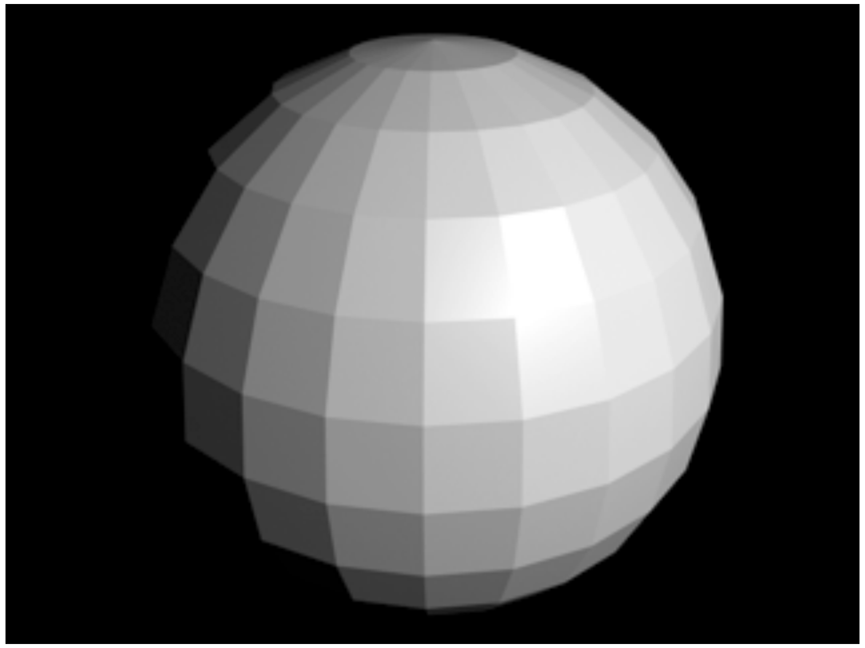


Flat shading (Phong Model)

Monash University • Clayton's School of Information Technology

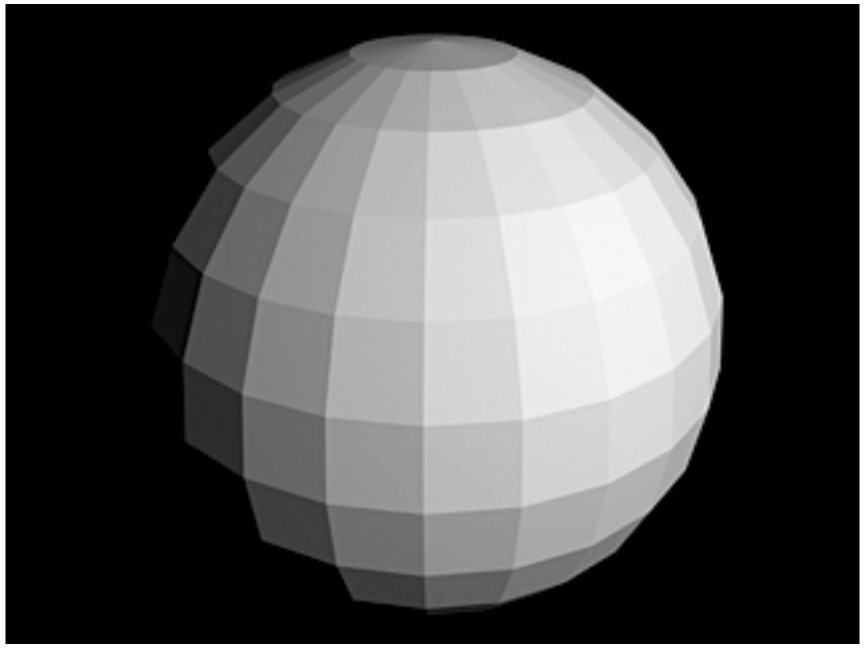
CSE3313 Computer Graphics

Lecture 23: Polygonal Shading and Global Illumination

Polygonal Shading

Reference: Angle 6.5

- The shape of the surface at any point is described by the normal at that point.
- Representing a curved surface by a mesh of flat, polygonal faces is efficient with respect to hidden surface elimination and scan conversion, but it means every point on the same face gets the same normal.
- If we only consider ambient light and diffuse reflection and the light source is far away, we might say that $\mathbf{L}$ is constant for all points on the plane and calculate one intensity/colour for each surface element – **constant shading**.

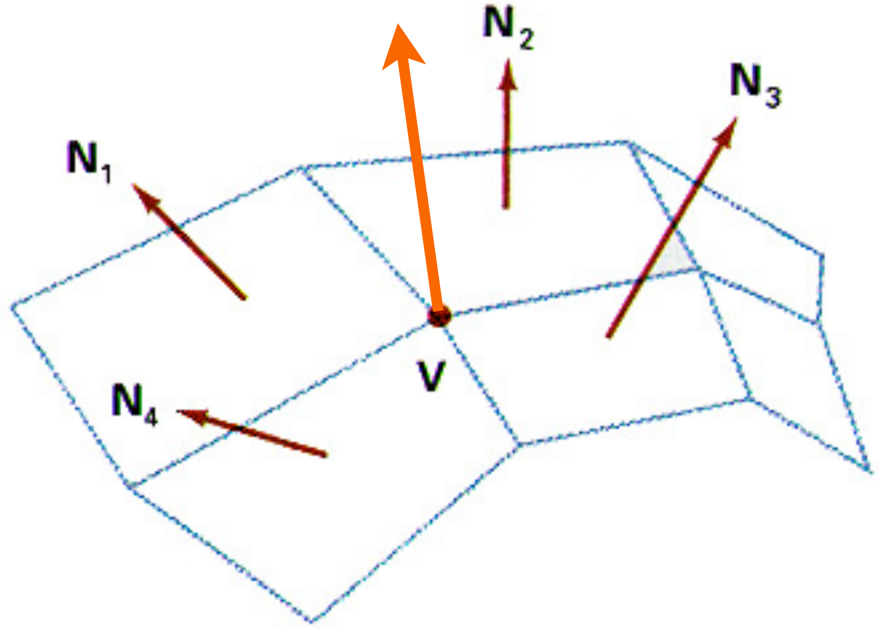


Constant or Flat Shading across Polygons

Polygonal Shading (cont.)

- Problems with constant shading:
  - specular reflections;
  - abrupt changes at polygon boundaries get emphasized by the human visual system.
- With a polygonal mesh a unique normal may not exist at the boundaries of polygons. We can create a normal for a vertex by averaging the unit normals of the faces that meet at that vertex:

$$\mathbf{N}_v = \frac{\mathbf{N}_1 + \mathbf{N}_2 + \mathbf{N}_3 + \mathbf{N}_4}{4}$$



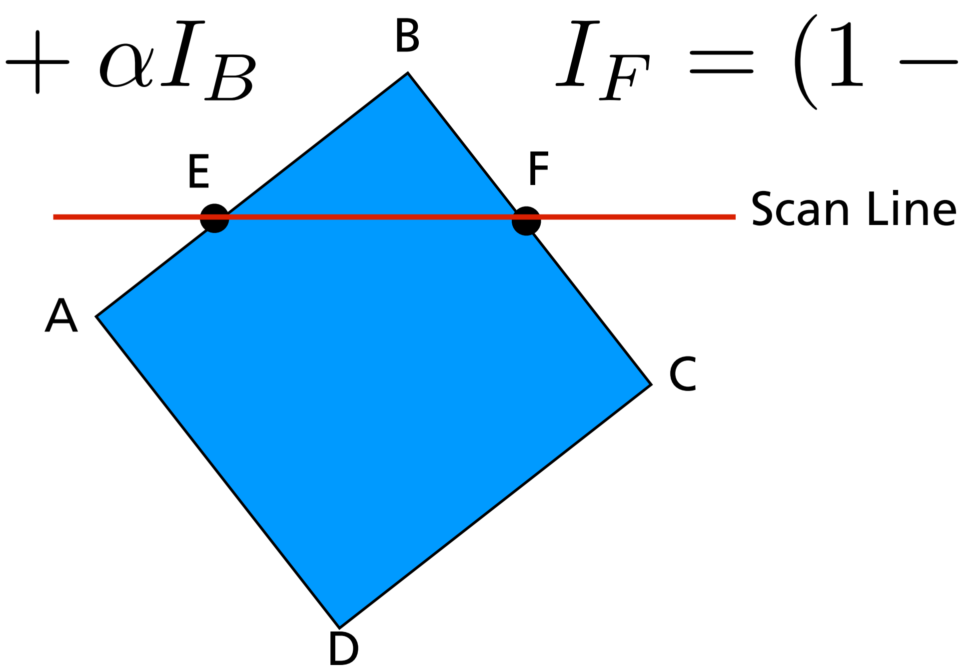
$$N_v = \frac{\sum_{i=1}^{n} N_i}{n}$$

The normal vector at vertex **V** is calculated as the average of the surface normals for each polygon sharing that vertex.

Vertex Normals and Gouraud Shading

- The normal along an edge can be calculated by interpolating the normal at each endpoint of the edge since the endpoints are vertices.
- In **Gouraud shading**, intensities at edges are linearly interpolated to derive intensities at pixels within a face. Interpolation is normally across a scanline and the scanline hidden surface removal algorithm can be generalised to include this interpolation.

$$I_E = (1-\alpha)I_A + \alpha I_B \qquad I_F = (1-\beta)I_B + \beta I_C$$

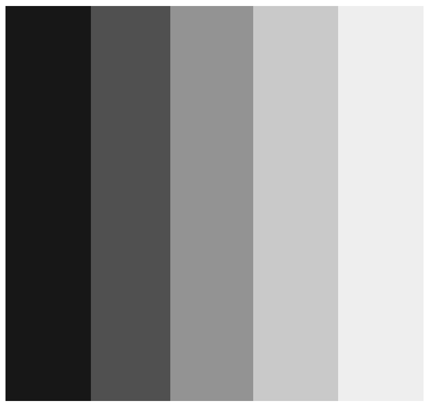


Gouraud Shading (cont.)

The intensities along EF get calculated by:

$$I(\gamma) = (1 - \gamma) I_E + \gamma I_F$$

- Gouraud shading still has some problems:
  - If the polygons are large;
  - If specular reflections need to be included;
  - Intensity differential is not continuous at edges (Mach banding).
- Gouraud shading can be carried out relatively efficiently since the intensities can be calculated incrementally across a scanline.



Step Chart

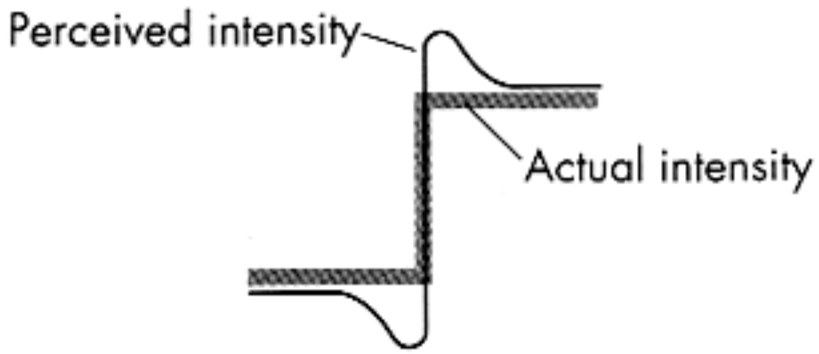


Figure 6.29 Perceived and actual intensities at an edge.

Phong Shading

- An improvement on Gouraud shading which is significantly more computationally expensive is **Phong interpolation shading**.
- With Phong shading, the normal is interpolated across a scanline with the full shading model being applied to every pixel in the polygon.
- The normals across the scanline can be calculated using linear interpolation and computed incrementally:

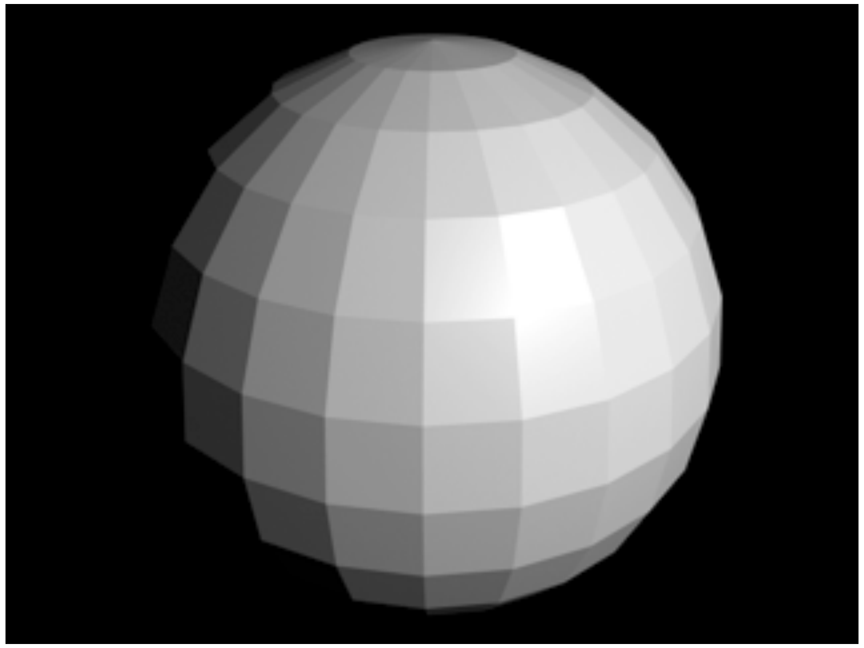
$$N_E = (1-\alpha)N_A + \alpha N_B$$
$$N_F = (1-\beta)N_B + \beta N_C$$

while the normals along EF get calculated by:

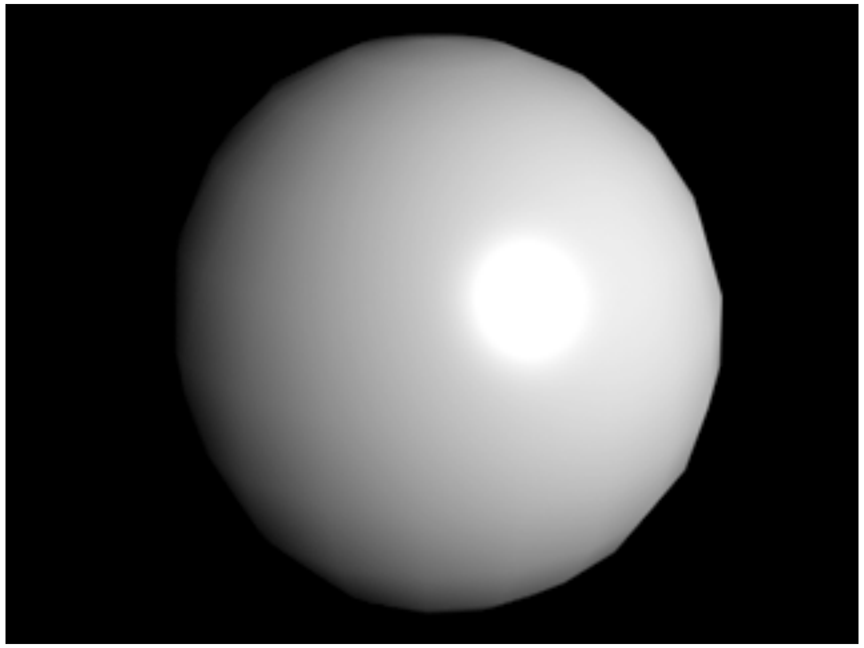
$$N(\gamma) = (1-\gamma)N_E + \gamma N_F$$

Phong shading can allow for specular reflections and Mach banding does not occur.

Local Illumination Models

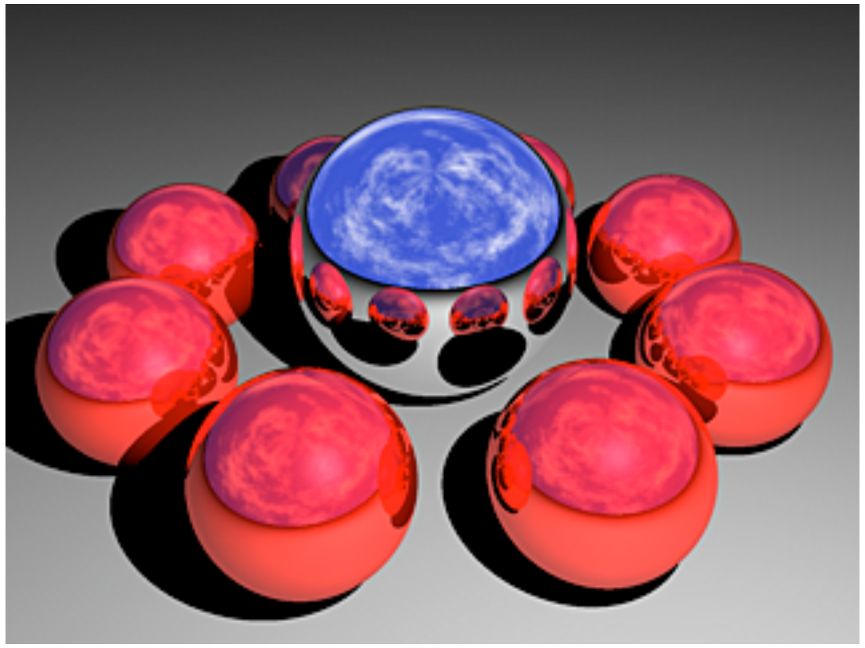


Flat shading (Phong Model)

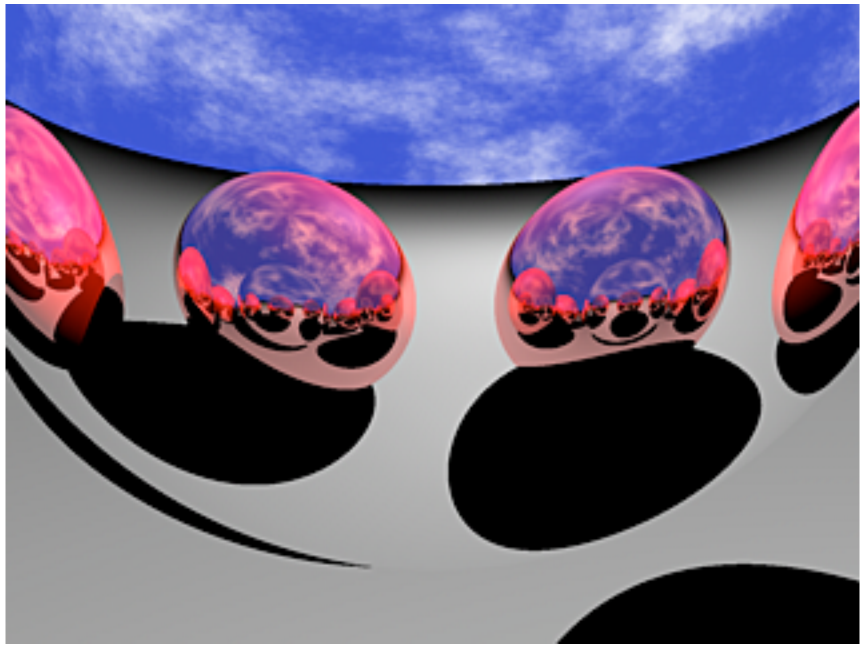


Phong Interpolation (Phong Model)

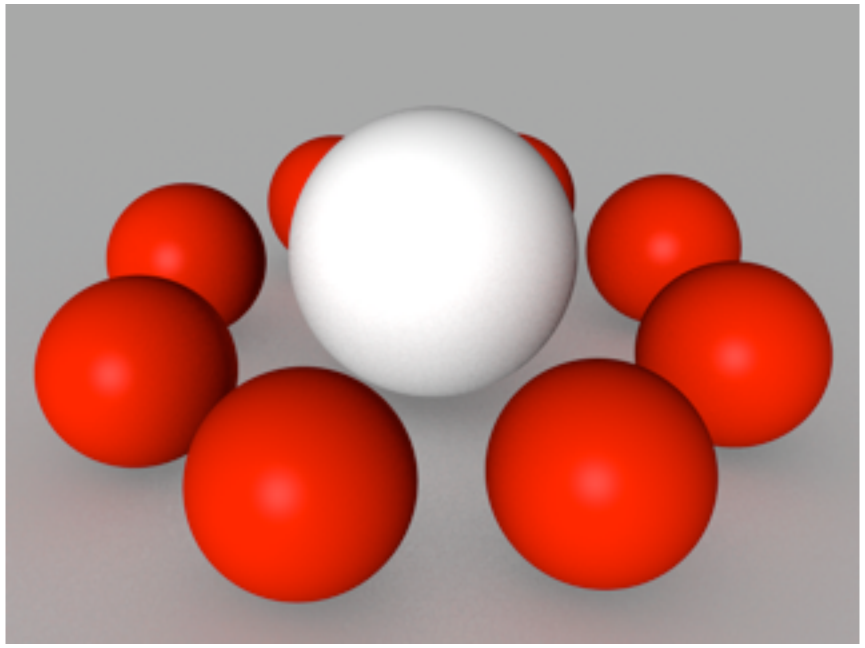
Global Illumination Models



Ray Tracing



Ray Tracing (close up)



Radiosity

Ray Tracing

- An extension of the *ray casting* algorithm, one of the earliest image-based methods of HSR.
- Based on principles of geometric optics (physics)
- Ray tracing determines visibility, calculates shadows, reflection, refraction, transparency, even does the perspective transform.
- Extensions include:
  - Anti-alising enhancements
  - Camera-like effects: depth-of-field and focusing effects, motion blur
  - Partitioning algorithms to deal with complex scenes

Ray Tracing

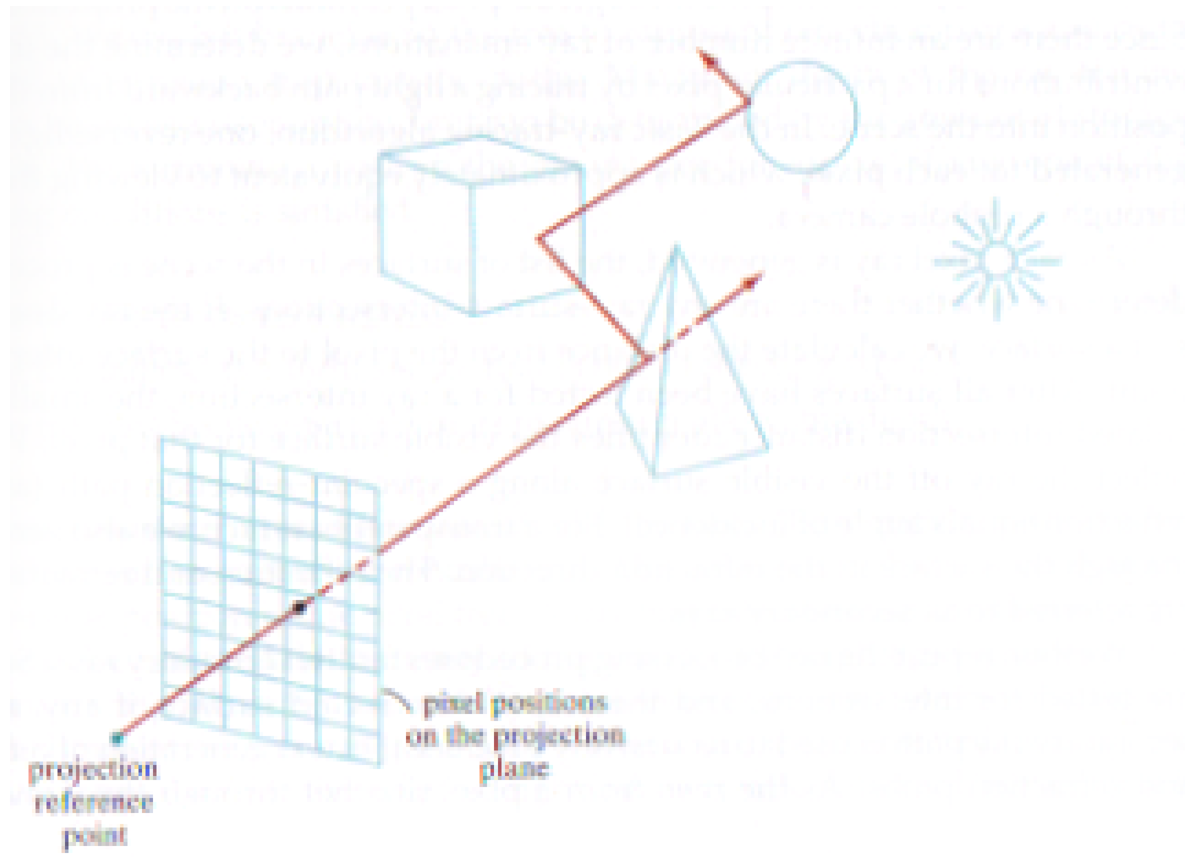


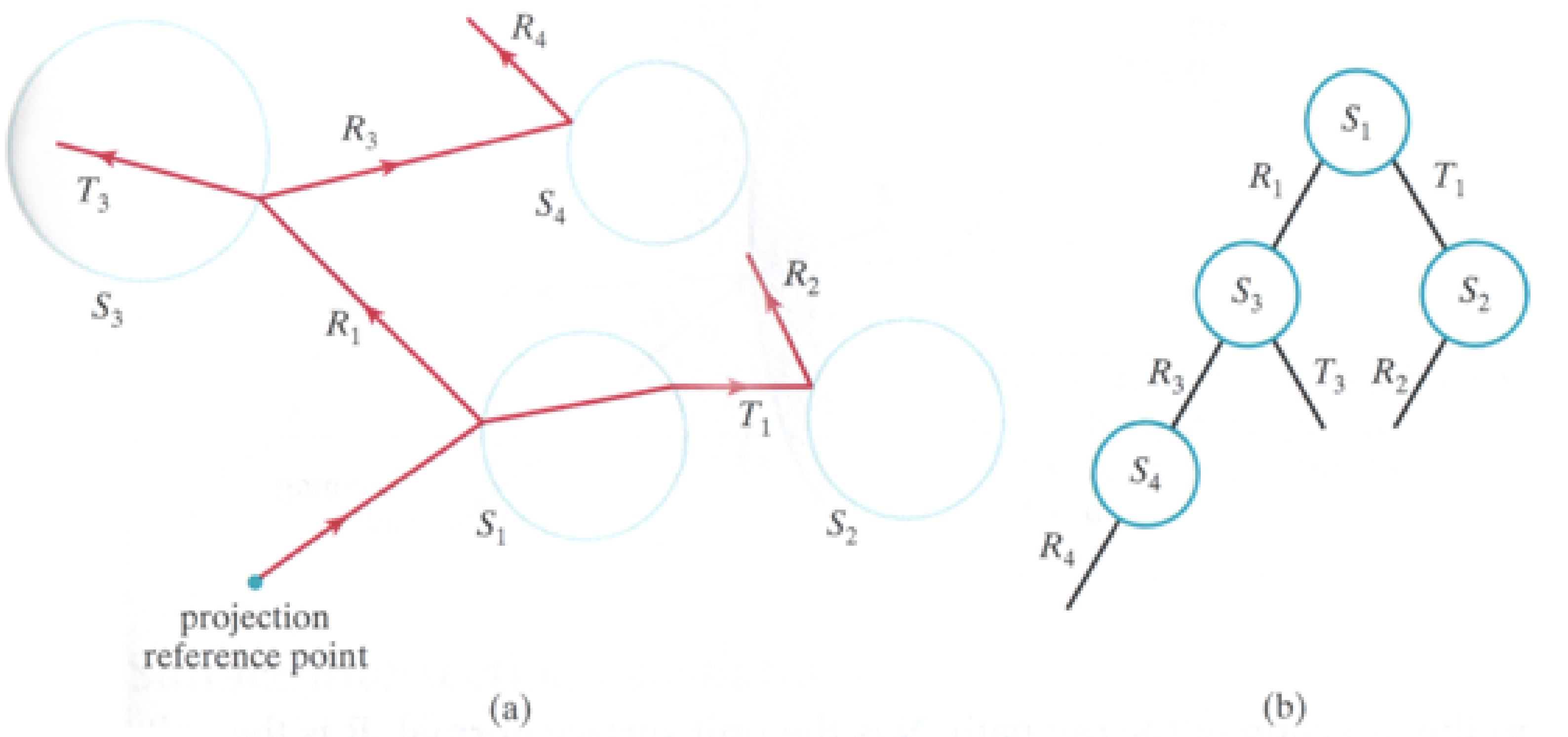
FIGURE 10-52 Multiple reflection and transmission paths for a ray from the projection reference point through a pixel position and on into a scene containing several objects.

- We assume pixel positions on the *x-y* plane and a centre of projection (projection reference point) on the *z* axis.
- Rays beginning at the COP are passed through each pixel position and tested for intersection with objects in the scene.
- Further rays may be spawned as a result of transmission, shadow or reflection.
- Each ray returns its contribution to the pixel intensity.

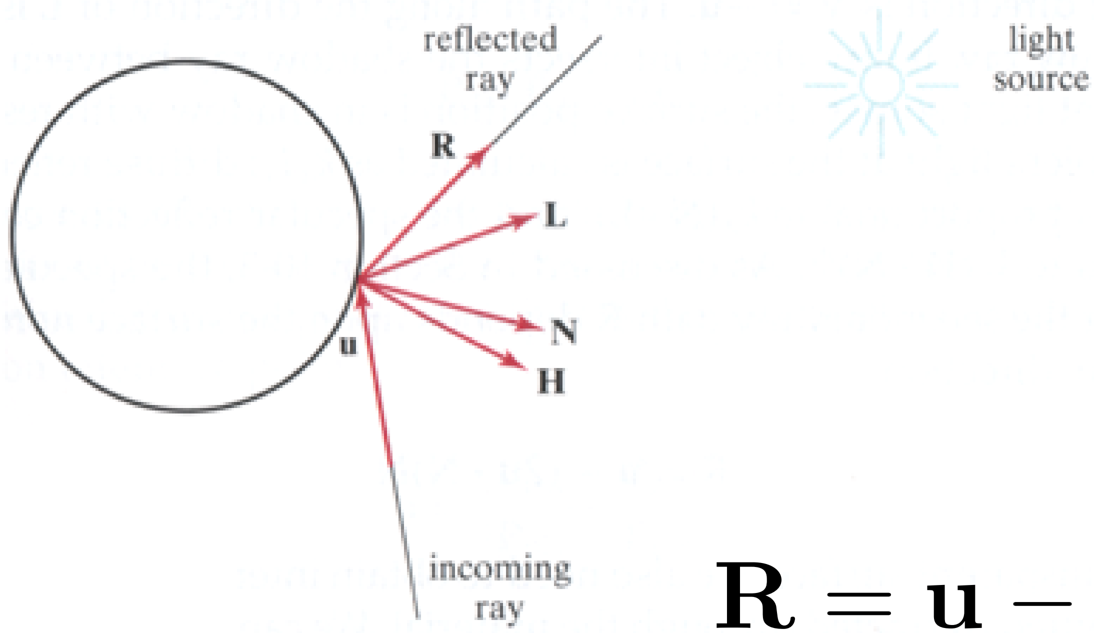
Ray Tracing (cont.)

- One primary “reverse” ray is generated for each pixel in the image (a bit like a “pinhole” camera).
- If an intersection occurs it means a surface is visible to that pixel.
- Shading can be performed at the intersection point, using a Phong model for local shading + contributions from spawned rays (if any).
- Secondary rays: reflection, refraction, transmission, shadow (trace ray from intersection point to the light source).
- Secondary rays may intersect with other surfaces in the scene (even ones outside the viewing frustum!). These intersections may spawn further rays as a result of reflection or refraction for example.
- A binary *ray-tracing tree* can be built based on these intersections. Traversing the tree returns the final illumination for the pixel.

Ray Tracing (cont)



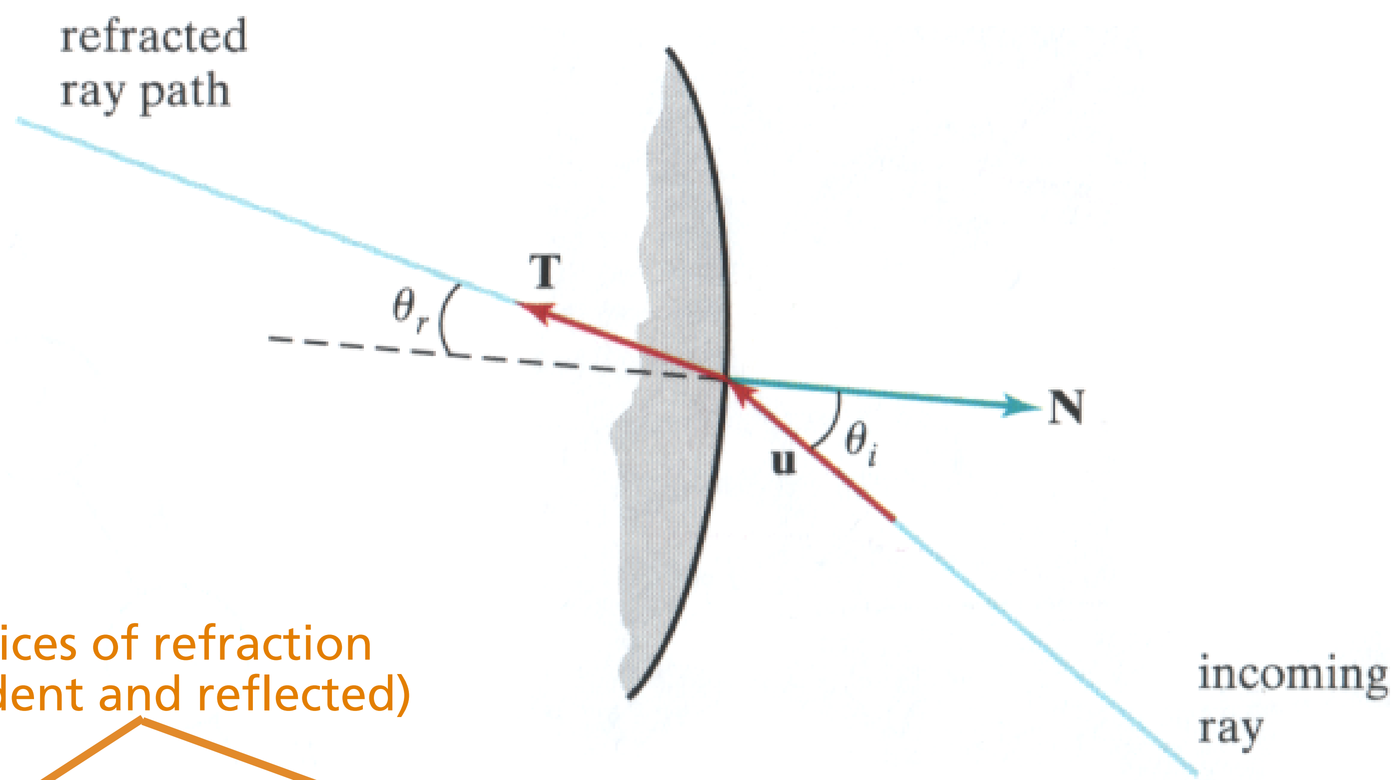
Reflection and Refraction paths for a pixel ray traveling through a scene are shown in (a) and the corresponding binary ray-tracing tree given in (b)



Unit vectors at the surface of an object intersected by an incoming ray along direction **u**.

$$\mathbf{R} = \mathbf{u} - (2\mathbf{u} \cdot \mathbf{N})\mathbf{N}$$

Ray Refraction



Refracted ray-transmission path **T** through a transparent material.

indices of refraction (incident and reflected)

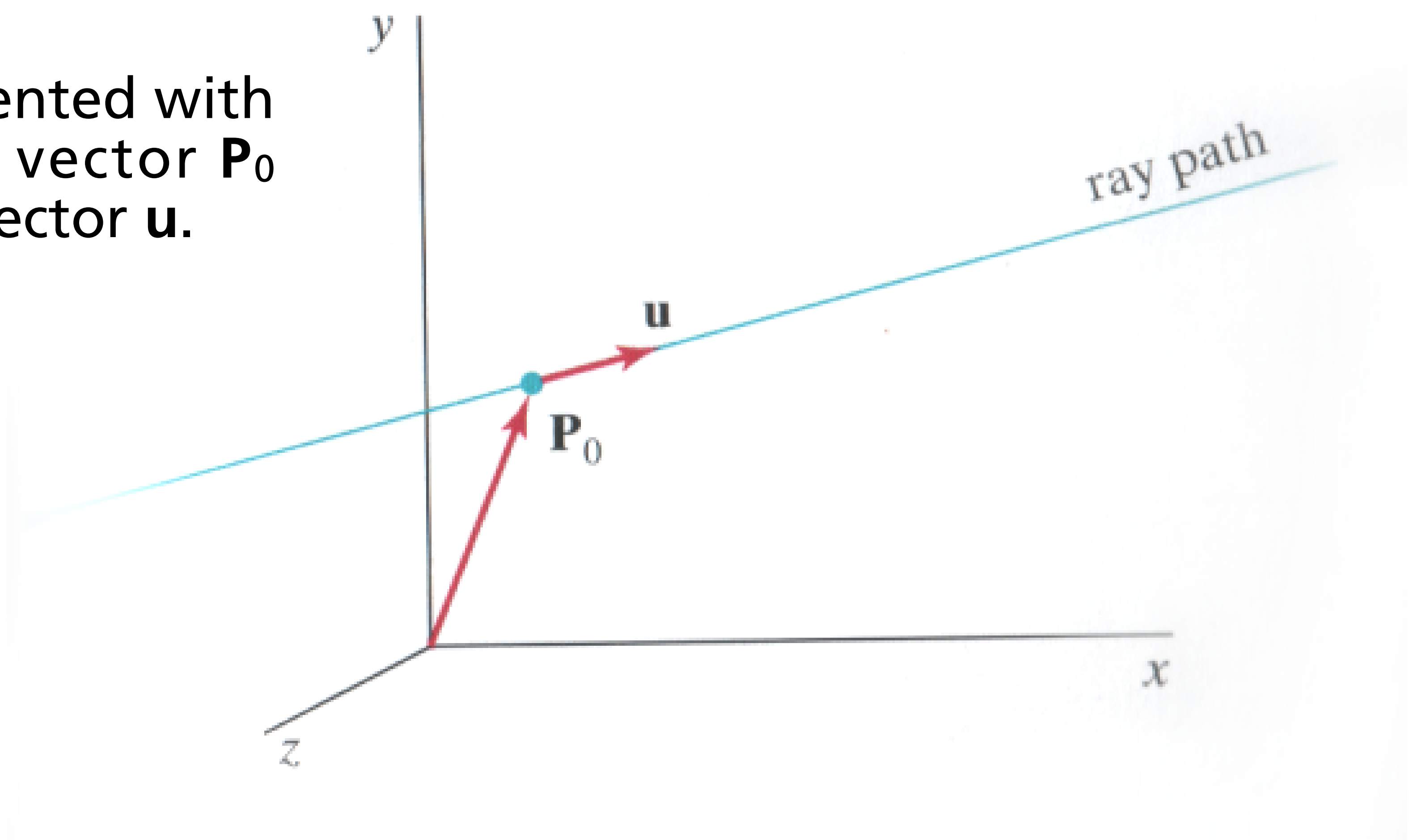
$$\mathbf{T} = \frac{\eta_i}{\eta_r}\mathbf{u} - (\cos\theta_r - \frac{\eta_i}{\eta_r}\cos\theta_i)\mathbf{N}$$

by Snell's law:

$$\cos\theta_r = \sqrt{1 - (\frac{\eta_i}{\eta_r})^2(1 - \cos^2\theta_i)}$$

Ray Representation

A ray can be represented with an initial-position vector $\mathbf{P}_0$ and unit direction vector **u**.



point on ray

distance along the ray

The ray-equation:

$$\mathbf{P} = \mathbf{P}_0 + s\mathbf{u}$$

- **u** can be calculated by creating a unit vector formed from the different from the current pixel centre and the COP

Ray-Surface Intersections

- Ray-surface intersection algorithms have been devised for many geometric primitives. Here we will look at ray-sphere intersection testing.

- We assume a sphere of radius *r* and centre position **P**c. Any point **P** on the surface satisfies the *sphere equation:*

$$|\mathbf{P} - \mathbf{P}_c|^2 - r^2 = 0$$

- Substitute the ray equation for **P:**

$$|\mathbf{P}_0 - s\mathbf{u} - \mathbf{P}_c|^2 - r^2 = 0$$

- Expanding we obtain the quadratic equation:

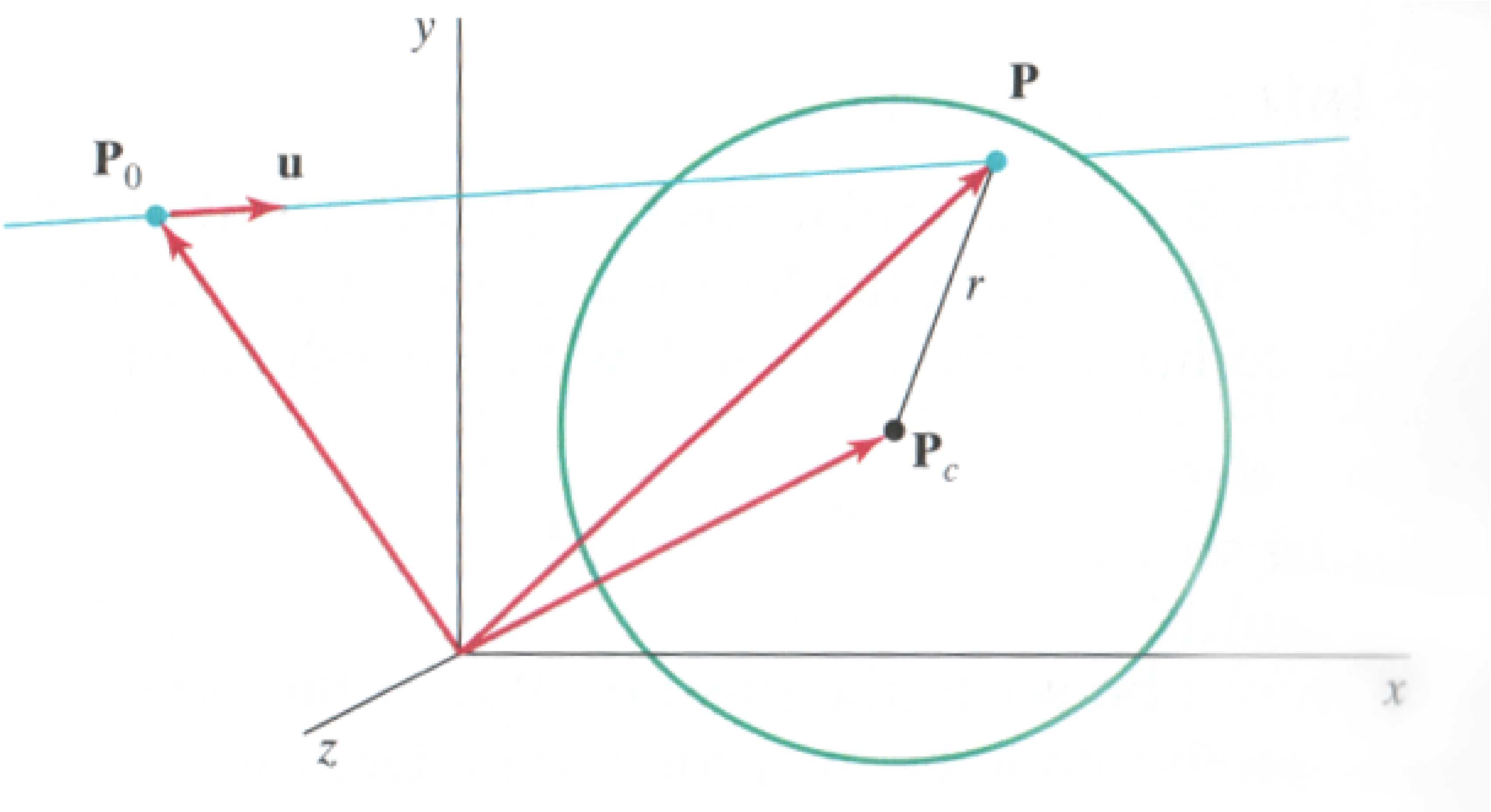
$$s^2 - 2(\mathbf{u} \cdot \Delta\mathbf{P})s + (|\Delta\mathbf{P}|^2 - r^2) = 0 \qquad \text{where } \Delta\mathbf{P} = \mathbf{P}_c - \mathbf{P}_0$$

- solving gives...

Ray-Sphere Intersections (cont.)

$$s = \mathbf{u} \cdot \Delta\mathbf{P} \pm \sqrt{(\mathbf{u} \cdot \Delta\mathbf{P}) - |\Delta\mathbf{P}|^2 + r^2}$$

- If the discriminant is negative, the ray does not intersect (or is behind $\mathbf{P}_0$
- For a non-negative discriminant, the intersection point is calculated from the smaller of the two values (the “front” side of the sphere)



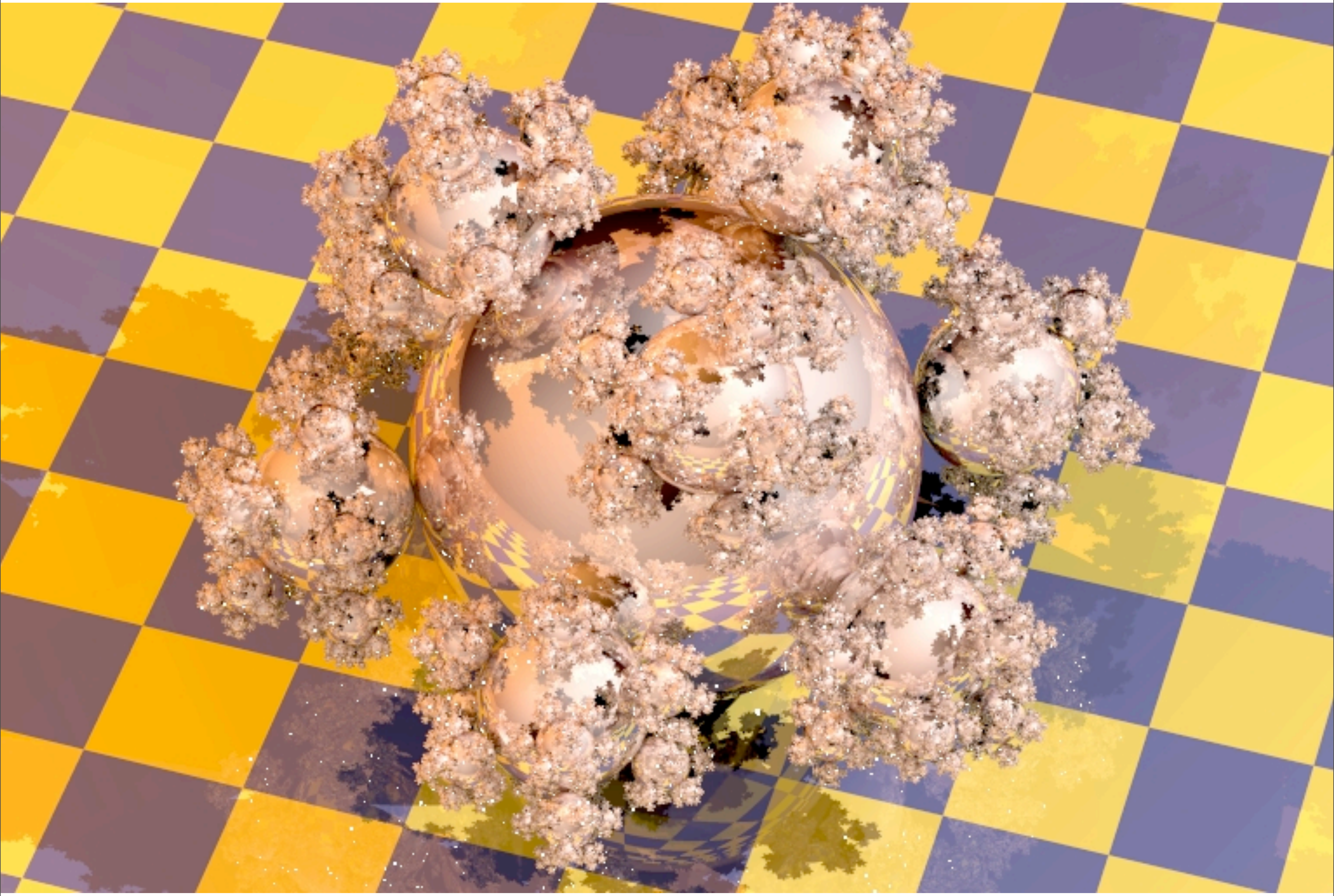
Ray-Sphere Optimisation

- The ray-sphere intersection test can be optimised.
- The intersection test is susceptible to round-off errors for very small spheres or spheres far from the initial ray position. i.e. if:

$$r^2 \ll |\Delta \mathbf{P}|^2$$

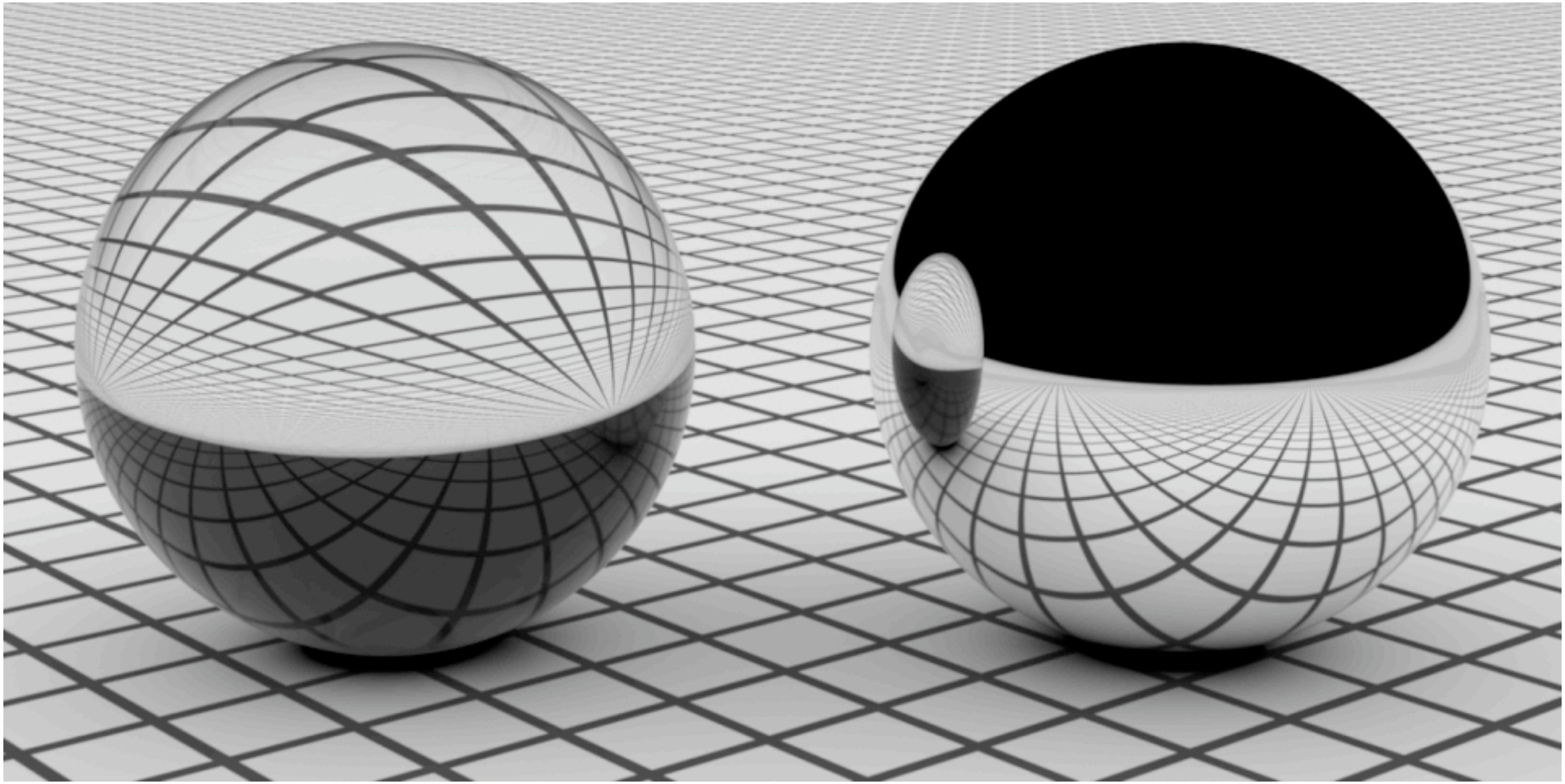
- The r^2 term may be lost due to the large size of $|\Delta \mathbf{P}|^2$
- This can be avoided by rearranging the calculation for distance *s*:

$$s = \mathbf{u} \cdot \Delta \mathbf{P} \pm \sqrt{r^2 - |\Delta \mathbf{P} - (\mathbf{u} \cdot \Delta \mathbf{P})\mathbf{u}|^2}$$



Distributed Ray Tracing

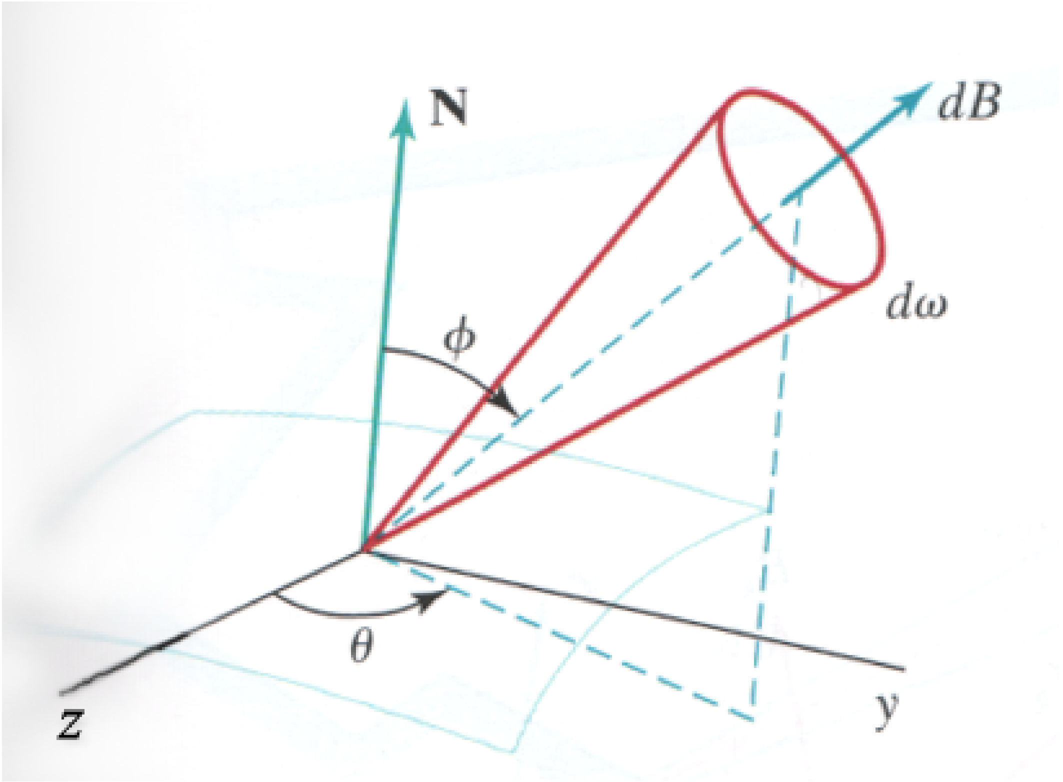
Cook, R. L, T. Porter and L. Carpenter (1984), Distributed Ray Tracing, *Computer Graphics* (SIGGRAPH '84 Proceedings), pp. 137-145 (http://doi.acm.org/10.1145/800031.808590)



Radiosity

- Radiosity methods use physical methods to model radient-energy transfers between surfaces in a scene.
- The basic radiosity model treats surfaces as small, opaque, ideal diffuse reflectors. For a given point on the surface, we measure the incoming energy from all other surfaces.
- The *radiant energy transfer* form a surface *dB* is the visible radiant flux emanating from the surface point in the direction given by angles θ and ϕ within the differential solid angle *d*ω per unit time, per unit surface area.

$$I = \frac{dB}{d\omega \cos \phi}$$



Radiosity (cont.)

- Assume the surface is an ideal diffuse reflector (*I* is constant in all directions). *dB/dω* is proportional to the projected surface area.
- To obtain the total rate of energy radiation from the surface point we sum the radiation over all directions (a hemisphere centered on the surface point) $$B = \int_{hemi} dB$$
- *I* is constant for a perfect diffuser so the radiant flux, *B* is:

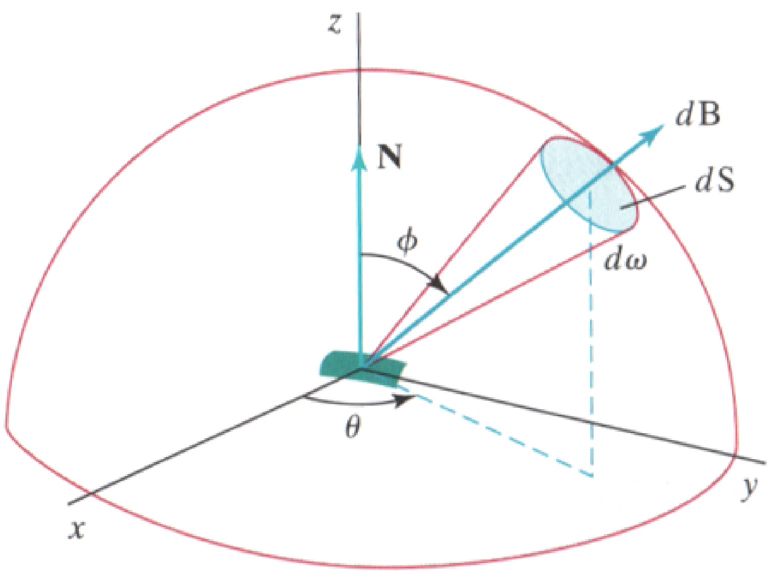
$$B = I \int_{hemi} \cos\phi \, d\omega$$

since $d\omega = \frac{dS}{r^2} = \sin\phi \, d\phi \, d\theta$

$$B = I \int_0^{2\pi} \int_0^{\pi/2} \cos\phi \sin\phi \, d\phi \, d\theta = I\pi$$

Basic Radiosity Model

- Surfaces in the enclosed scene are one of:
  - (i) reflectors
  - (ii) emitters (light source)
  - (iii) combination of (i) and (ii)



Incident energy parameter

Total rate of radiant energy leaving surface *j* per unit area

$$H_k = \sum_j B_j F_{jk}$$

Summed over all surfaces in the enclosure

Form factor, fraction of radiant energy from surface *j* that reaches surface *k* ($F_{kk} = 0$)

Radiosity Equation

Rate of energy emitted from surface *k* per unit area (watts/m^2) – light source

$$B_k = E_k + \rho_k H_k$$

Radiant energy from surface *k*

Diffuse reflection coefficient for surface *k* (like k_d)

Basic Radiosity Model (cont.)

- For a scene with *n* surfaces we need to solve the simultaneous radiosity equations. i.e.:

$$(1 - \rho_k F_{kk}) B_k - \rho_k \sum_{j \neq k} B_j F_j k = E_k \qquad k = 1, 2, 3, \cdots, n$$

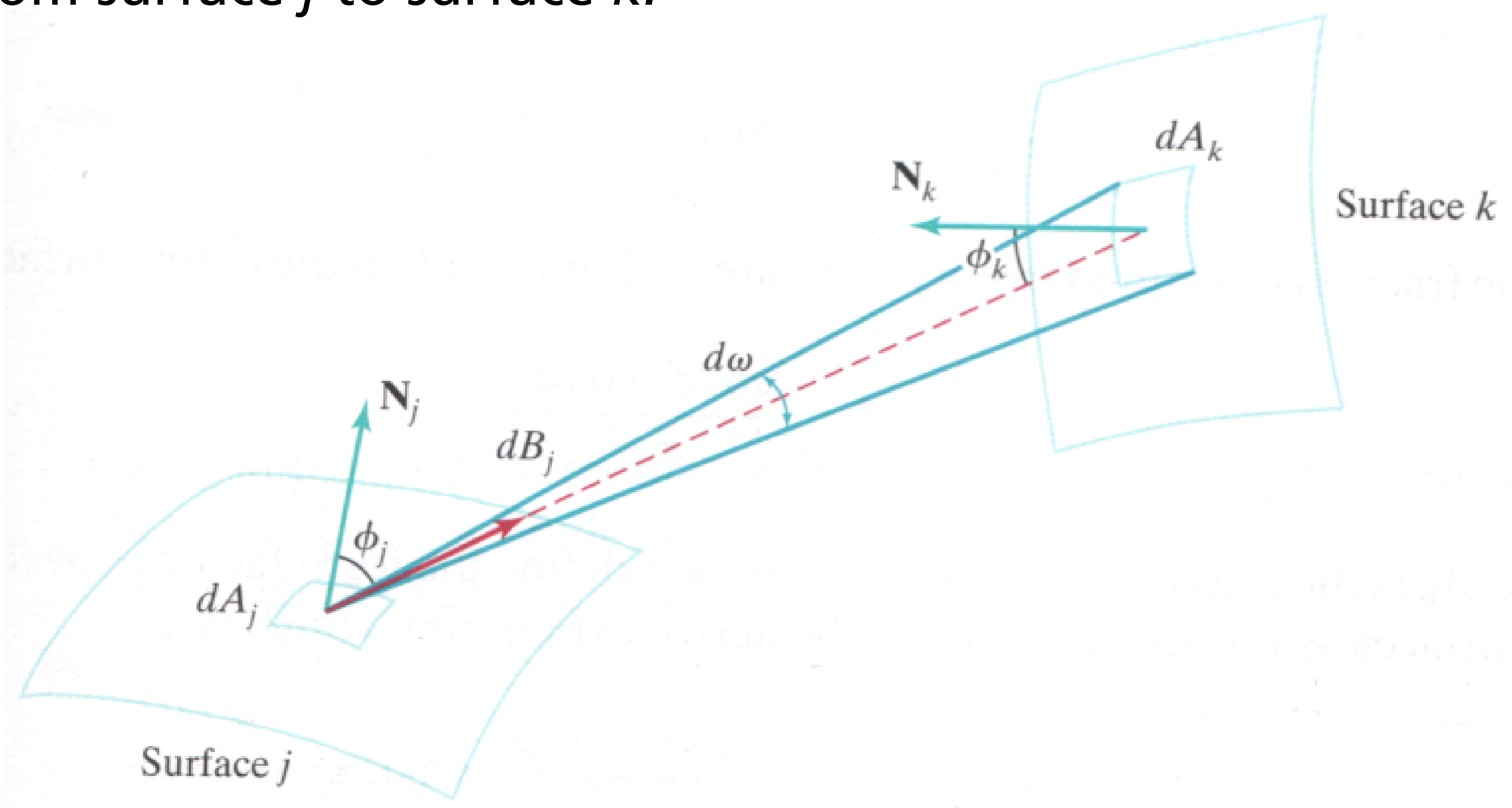
or

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

- Intensity values are calculated by dividing B_k by π

Form Factor Calculation

- Form factors F_{jk} are calculated by considering the energy transfer from surface j to surface k.



$$
\begin{aligned}
F_{dA_j,dA_k} &= \frac{\text{energy incident on } dA_k}{\text{total energy leaving } dA_j} \\
&= \frac{I_j \cos\phi_j \cos\phi_k dA_j dA_k}{r^2} \cdot \frac{1}{B_j dA_j}
\end{aligned}
$$

Form Factor Calculation (cont.)

$$F_{jk} = \frac{1}{A_j} \int_{surf_j} \int_{surf_k} \frac{\cos\phi_j \cos\phi_k}{\pi r^2} dA_k dA_j$$

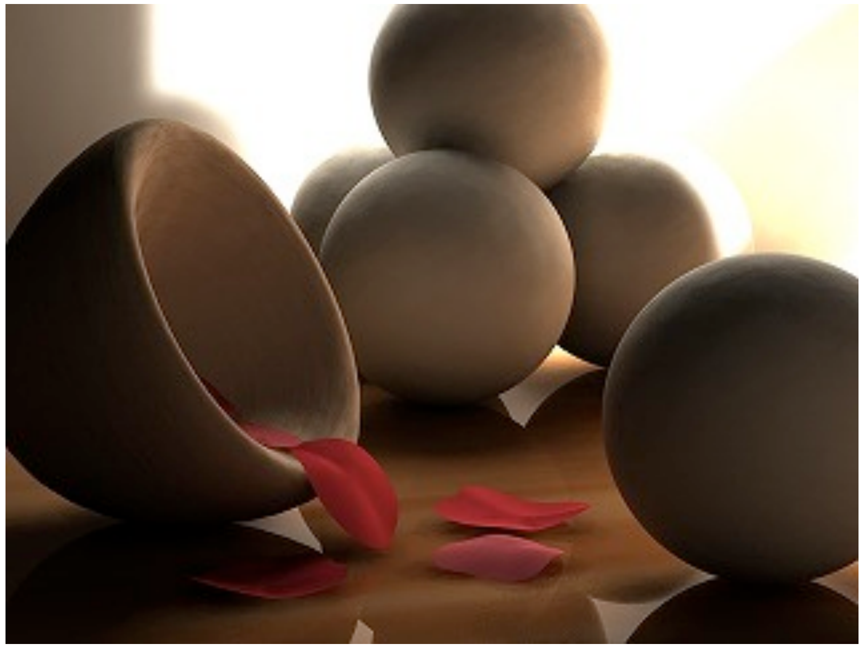
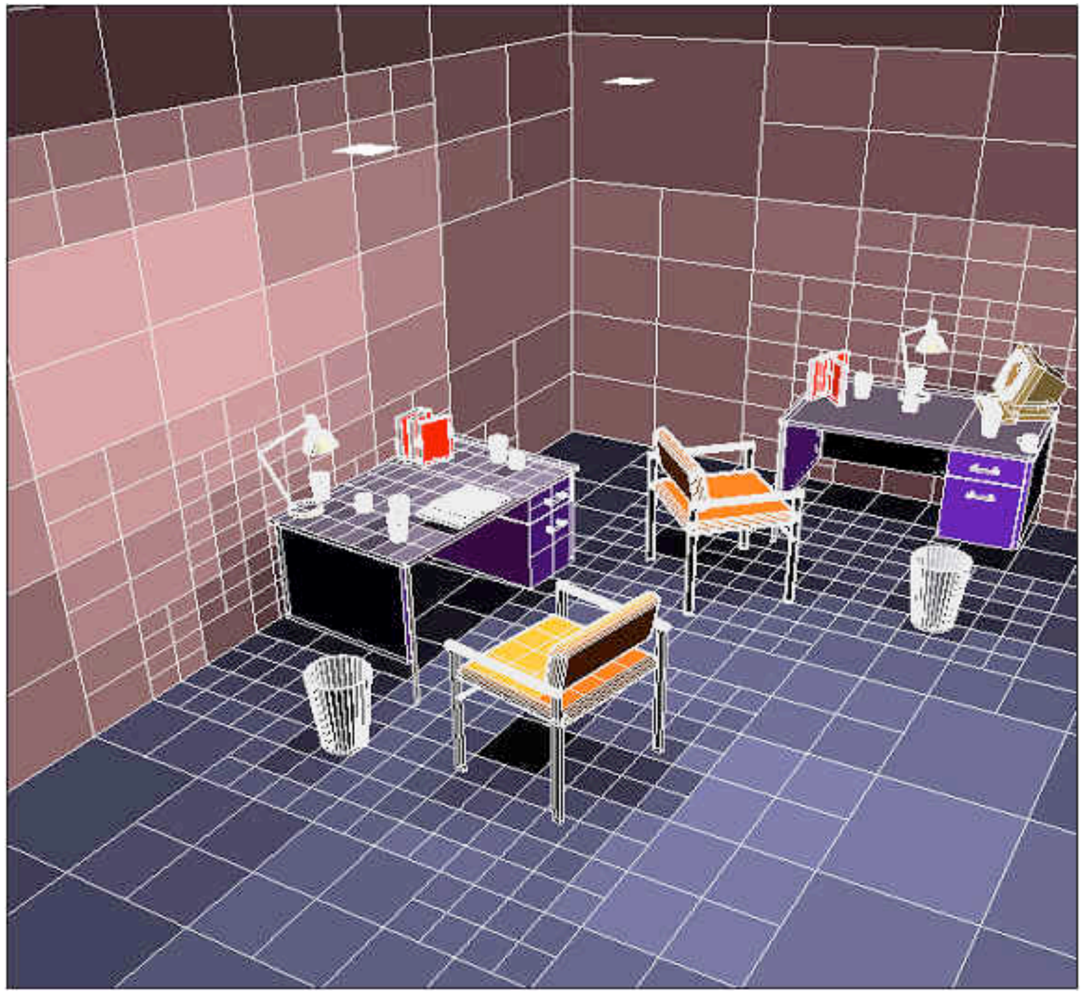
- This equation can be evaluated using numerical integration methods, with the following conditions:

- $\sum_{k=1}^{n} F_{jk} = 1, \quad \text{for all } k \quad \text{(conservation of energy)}$
- $A_j F_{jk} = A_k Fkj \quad \text{(uniform light reflection)}$
- $F_{jj} = 0, \quad \text{for all } j \quad \text{(assuming only planar or convex surface patches)}$

- Form Factor calculation can be speeded up using the *hemicube* method, which approximates the hemisphere integration with a cube of linear surface patches.

- For more information see Hearn & Baker, Section 10-12.



More Fun

- Pharr, M. and G. Humphreys (2004): Physically-based Rendering: From Theory to Implementation, Morgan Kaufmann. http://pbrt.org/ – advanced open source renderer with book written in literate programming style (available in the library)
- POVRay - www.povray.org
- Rayshade - graphics.stanford.edu/%7Ecek/rayshade/
- See the subject “Course Resources” page for more...