Light Issues in Computer Graphics

Presented by Saleema Amershi

- Light plays an important part in computer graphics for rendering *realistic* images.
- Using lighting models, we can simulate shading, reflection and refraction of light, comparable to what we see in the real world.

- Local illumination refers to direct interaction between *one* light source and *one* object surface.

- Global illumination refers to the interaction of light *between* all surfaces in a scene.
  - Responsible for shading
  - Reflection between surfaces
  - Refraction of surfaces

Local Illumination Models

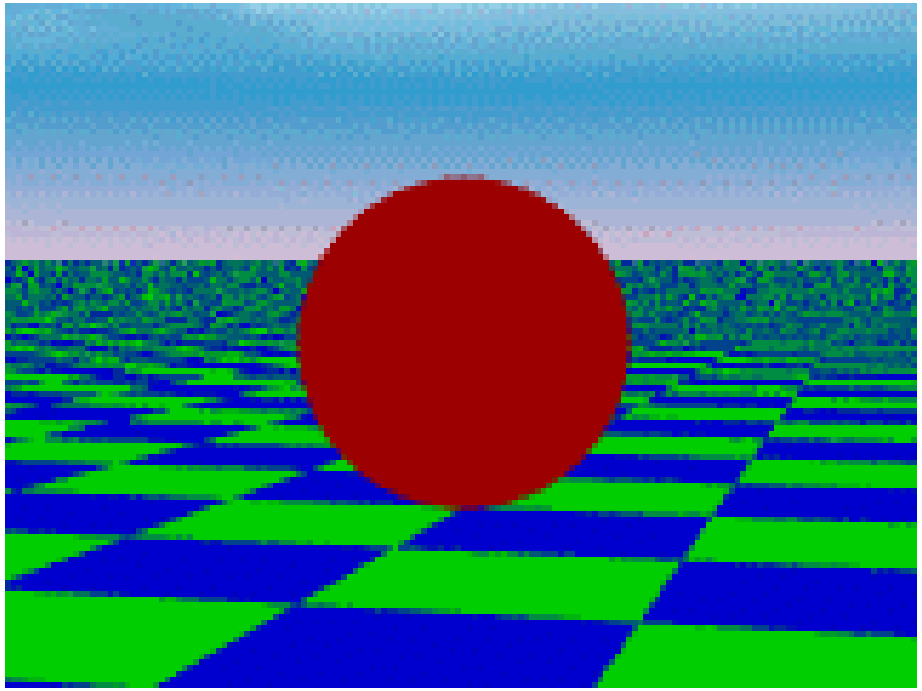
- In computer graphics, single object-light interaction is approximated through local illumination models.
- Basic model used is the Phong model which breaks local illumination into 3 components:
  - Ambient reflection
  - Diffuse reflection
  - Specular reflection
- For every point, or small surface area, of an object, we want to calculate the light due to these three components.

Ambient reflection

- Crude approximation to global effects of light.
- Accounts for the general brightness in a scene from light scattering in all directions from all surfaces.

- $I_{out} = k_{ambient} \cdot I_{ambient}$

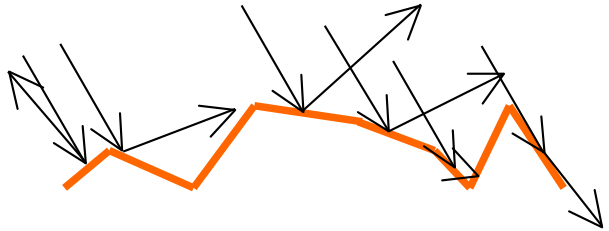
- I is the light intensity (power per unit area), or illumination.

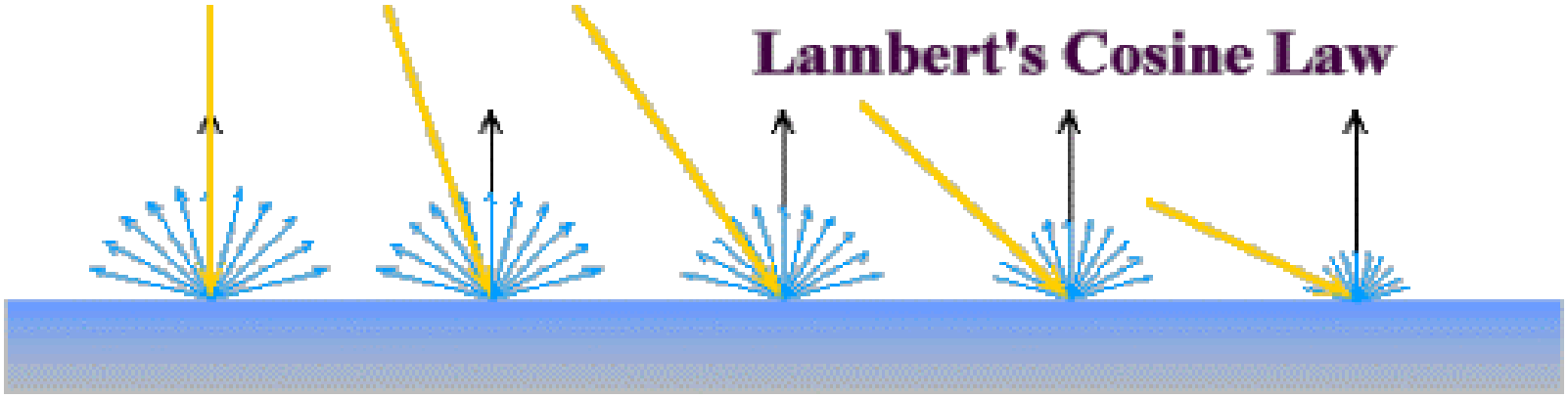


Diffuse Reflection

- All materials have diffuse properties, due to the ‘roughness’ of a surface at the microscopic level.

- Ideal diffuse reflection refers to light hitting a surface and then scattering evenly in all directions due to the surface ‘roughness’.





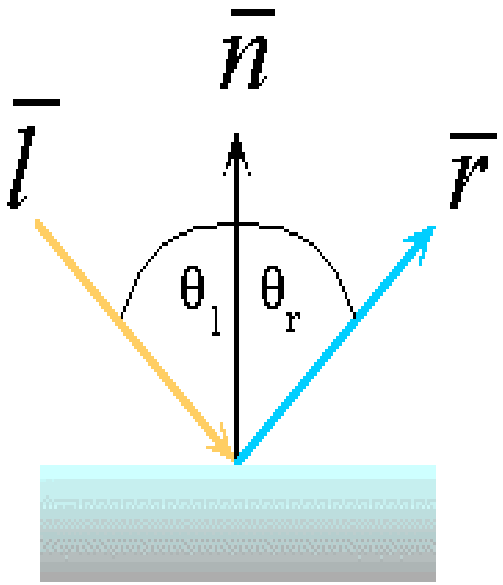
- Lambert said that the energy reflected off a surface from a light source is proportional to the cosine of the incident angle, *i*, between the light and the normal to the surface.

- I_{out} α $cos(i)$ or I_{out} α $\mathbf{n} \cdot \mathbf{l}$

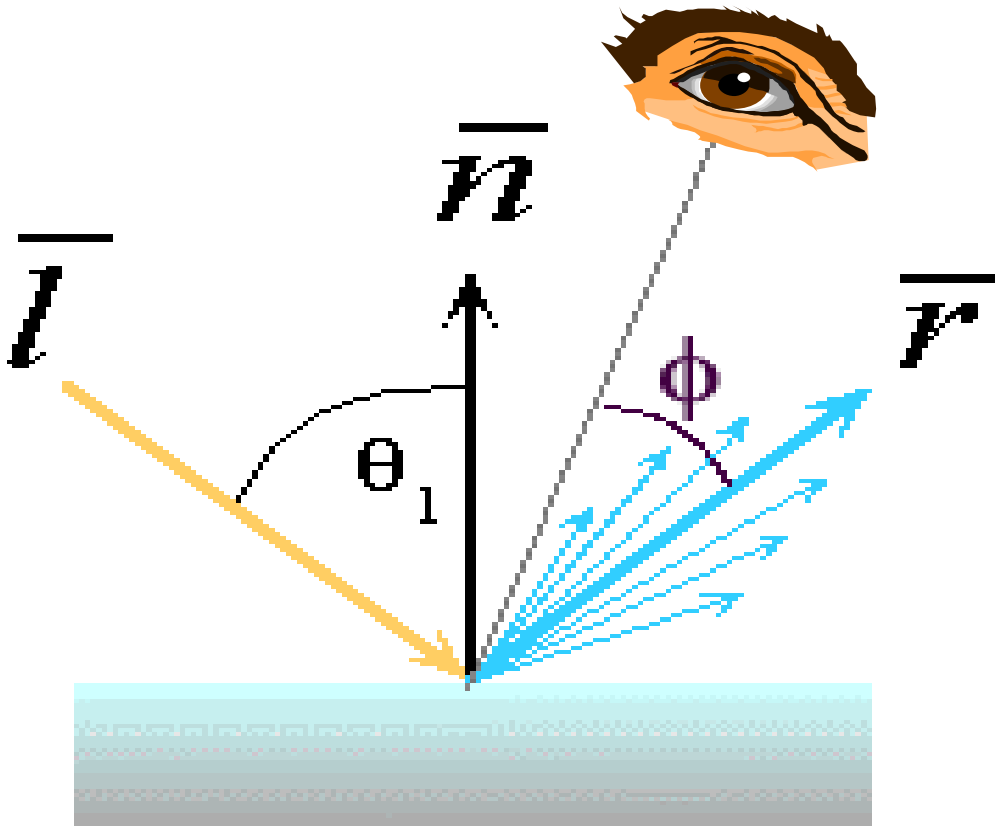
- So now we have
  - $I_{out} = k_{ambient} \cdot I_{ambient} + k_{diffuse} \cdot I_{light} \cdot \mathbf{n} \cdot \mathbf{l}$

Specular Reflection

- Shiny materials have specular properties, that give highlights from light sources.
- The highlights we see depends on our position relative to the surface from which the light is reflecting.
- For an ideal mirror, a perfectly reflected ray is symmetric with the incident ray about the normal.



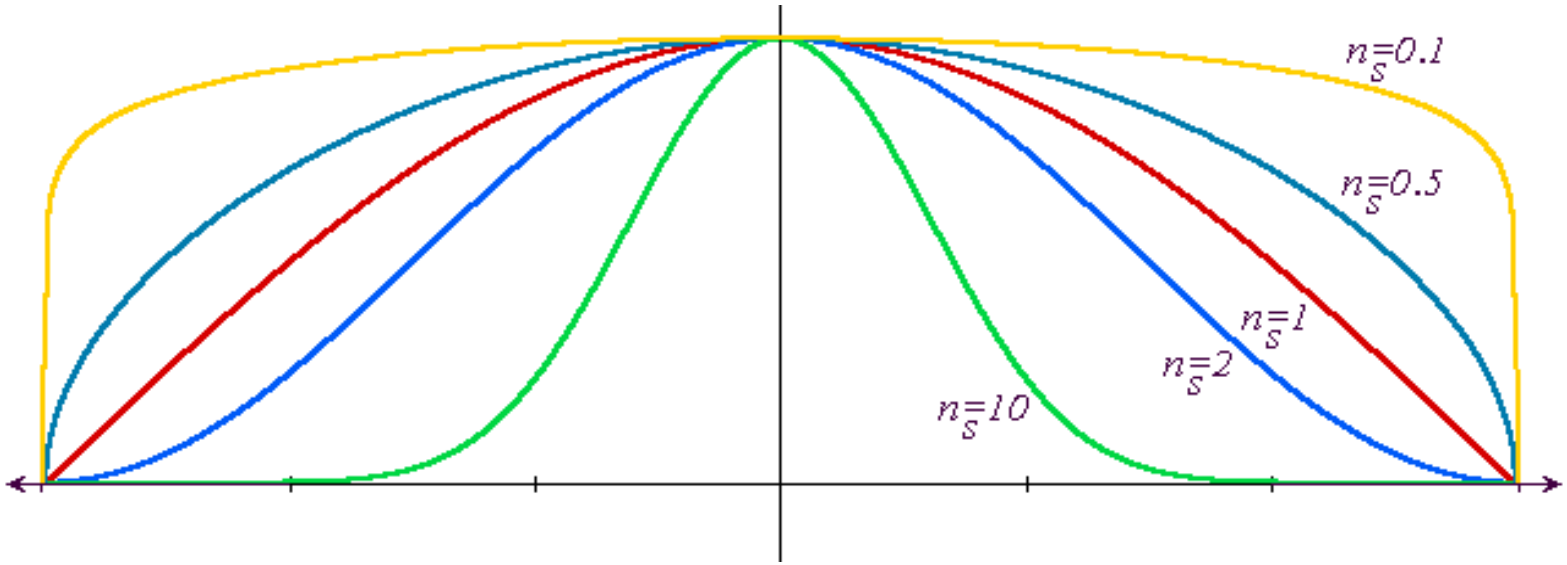
- But as before, surfaces are not perfectly smooth, so there will be variations around the ideal reflected ray.



- Phong modelled these variations through empirical observations.
- As a result we have:

$$I_{out} = k_{specular} \cdot I_{light} \cdot \cos^{s}(\theta)$$

- *s* is the shininess factor due to the surface material.



Phong Lighting Model

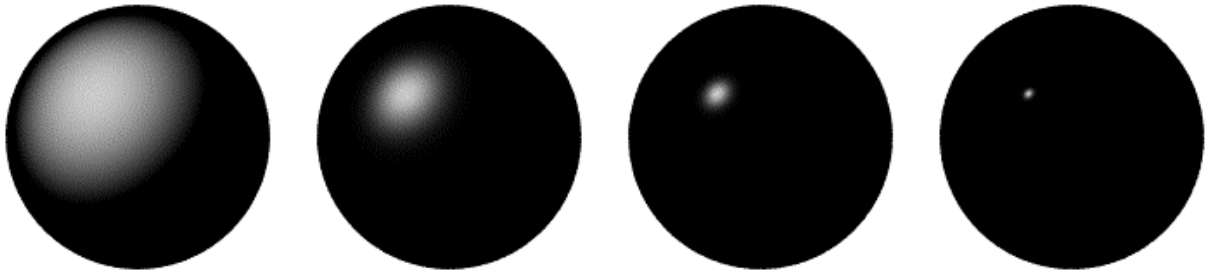
- Putting all these components together gives us:

$$I_{out} = k_{ambient} \cdot I_{ambient} + k_{diffuse} \cdot I_{light} \cdot (\mathbf{n} \cdot \mathbf{l}) + k_{specular} \cdot I_{light} \cdot (\boldsymbol{v} \cdot \mathbf{r})^{s}$$

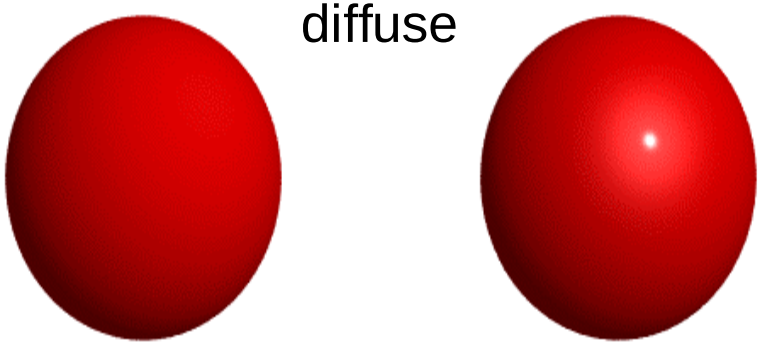
- In reality, however, we can have more then one light source reflecting light off of the same surface. This gives us:

$$I_{out} = k_{ambient} \cdot I_{ambient} + \sum I_{light} \cdot (k_{diffuse} \cdot (\mathbf{n} \cdot \mathbf{l}) + k_{specular} \cdot (\boldsymbol{v} \cdot \mathbf{r})^{s})$$

- Varying shininess coefficient in specular component:

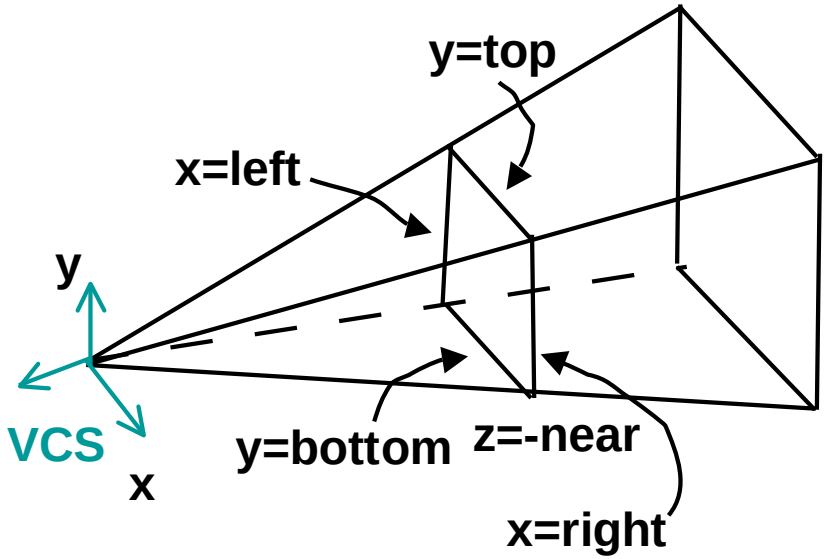


- Combining diffuse and specular lighting:

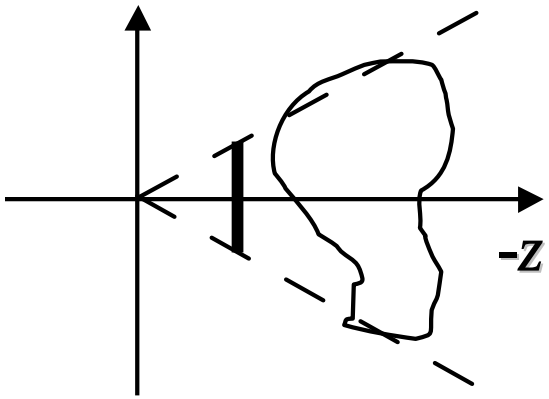


How do we use really use this?

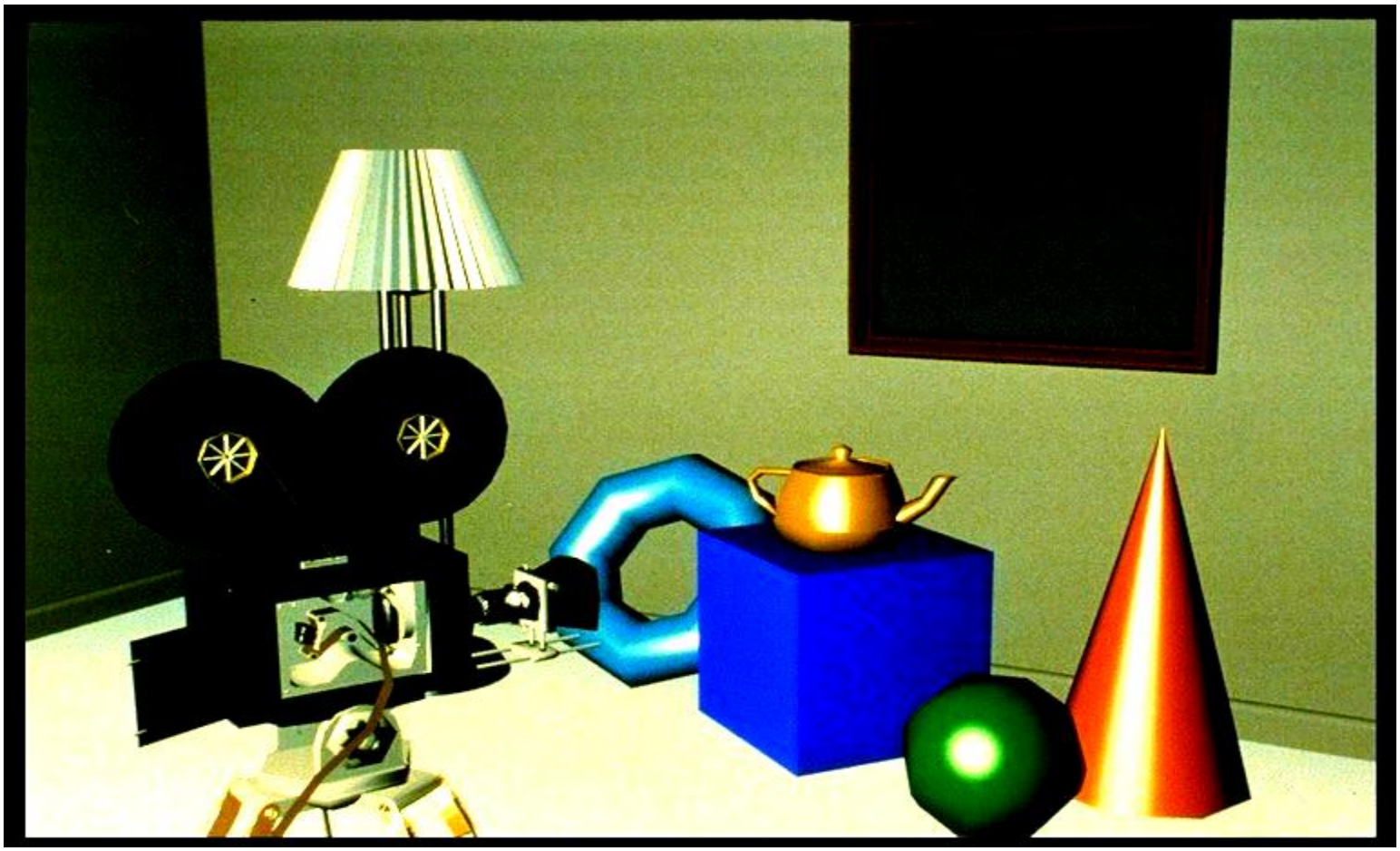
- Viewing frustrum



- Z-buffering and the image plane



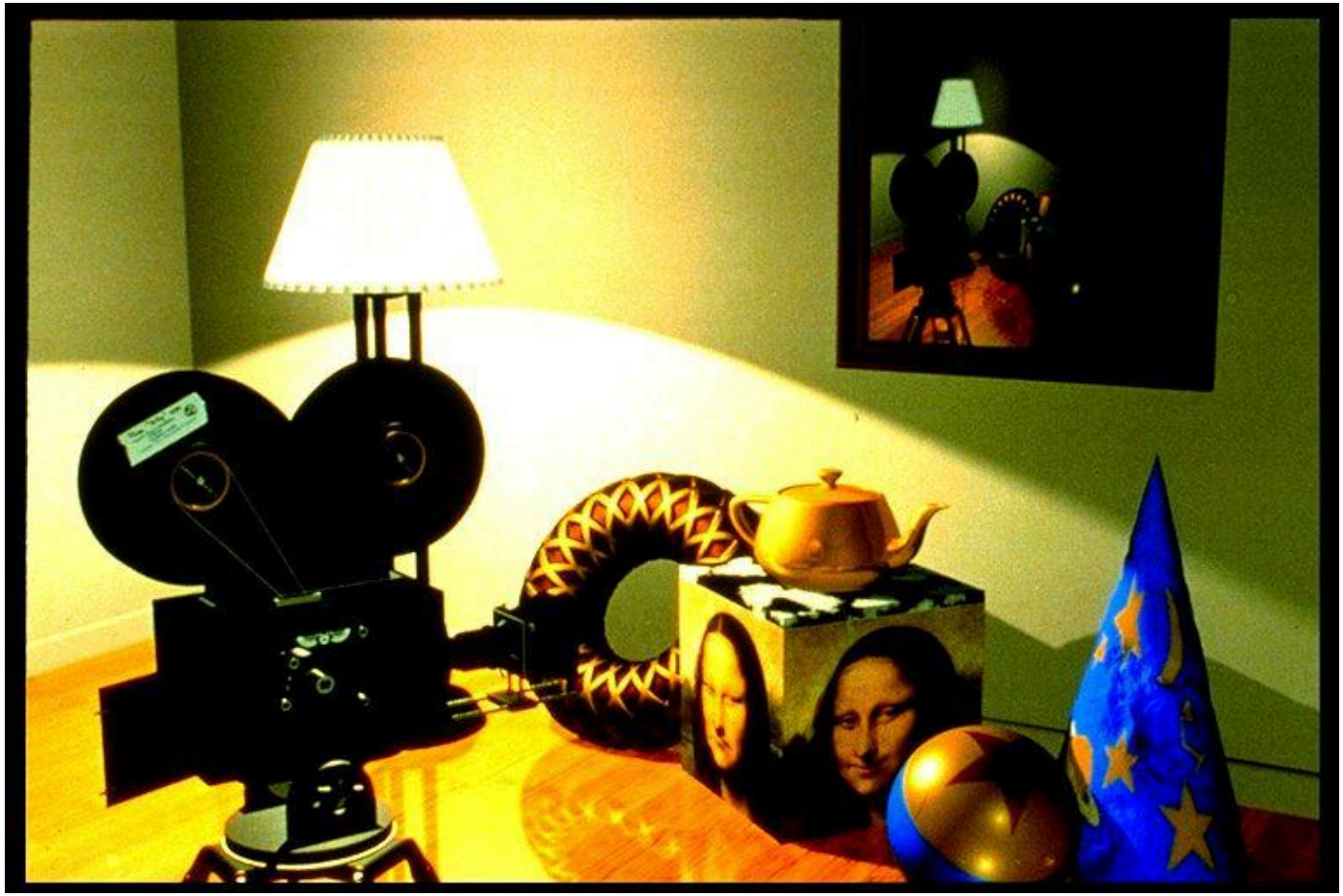
Example

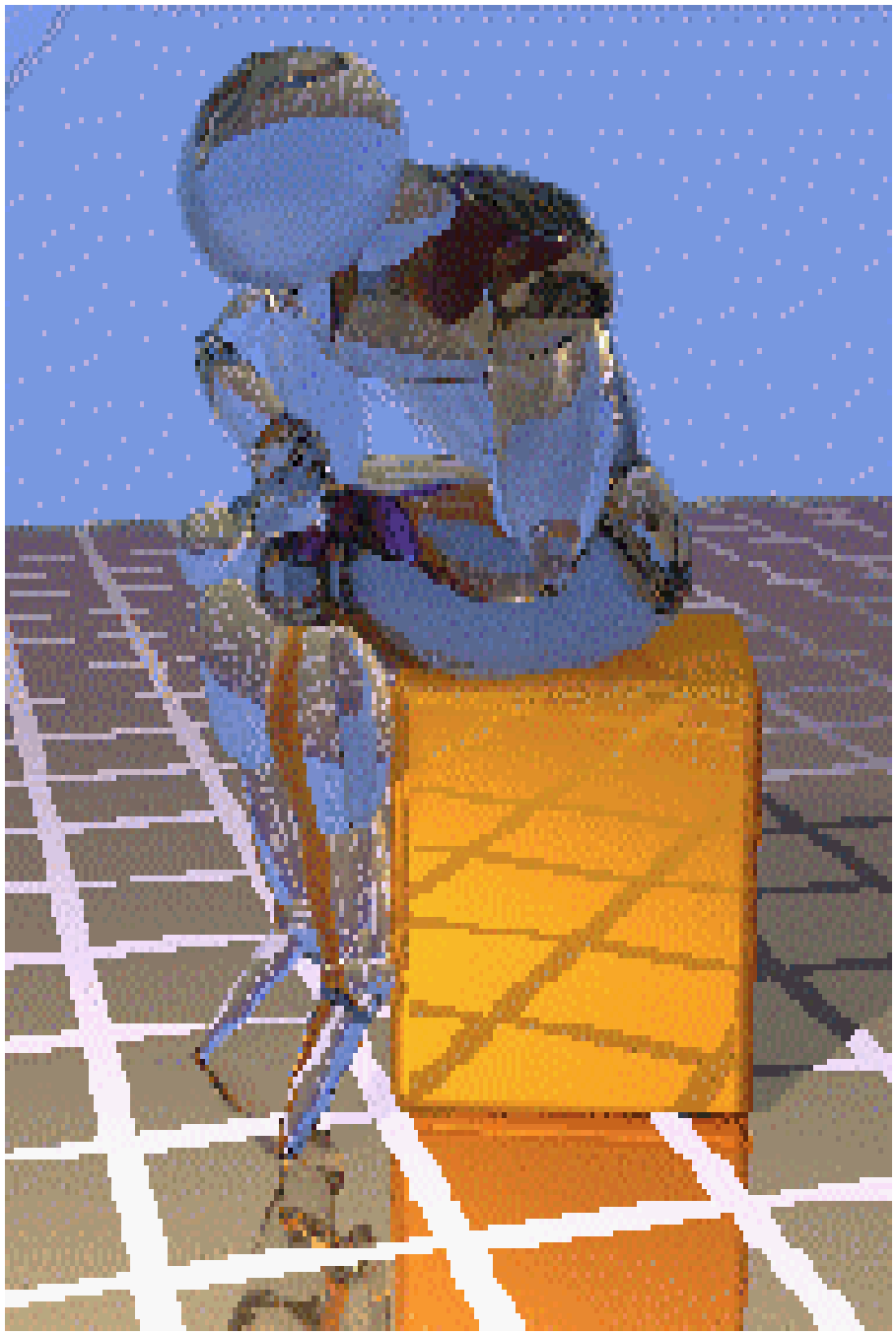


- No shadows
- No refractions or reflections

Ray Tracing!

- Better method, can show these realistic effects.





Ray Tracing Method

- Cast a ray from the eye (or the camera) through each pixel in the image plane, until the ray intersects an object.

- Calculate local illumination for this point using Phong model.

Calculating Intersections

- Largest computational overhead.
- Most objects are represented by collections of planar polygons.
- Intersections are between rays and planes.
- Implicit plane equation

$$F(P) = N \cdot P + D = 0$$

- Parametric ray equation

$$P(t) = P_a + t(P_b - P_a)$$

- Solve for t:

$$F(P(t)) = N\cdot(P_a + t(P_b - P_a) + D = 0$$

$$t =(-D -N\cdot P_a)/(N\cdot P_b - N\cdot P_a)$$

- Plug back into *P(t)* to get intersection point.
- Remember that calculating *t* is solving a quadratic. We want the first intersection of the ray with a surface, so take the smallest *t* value.

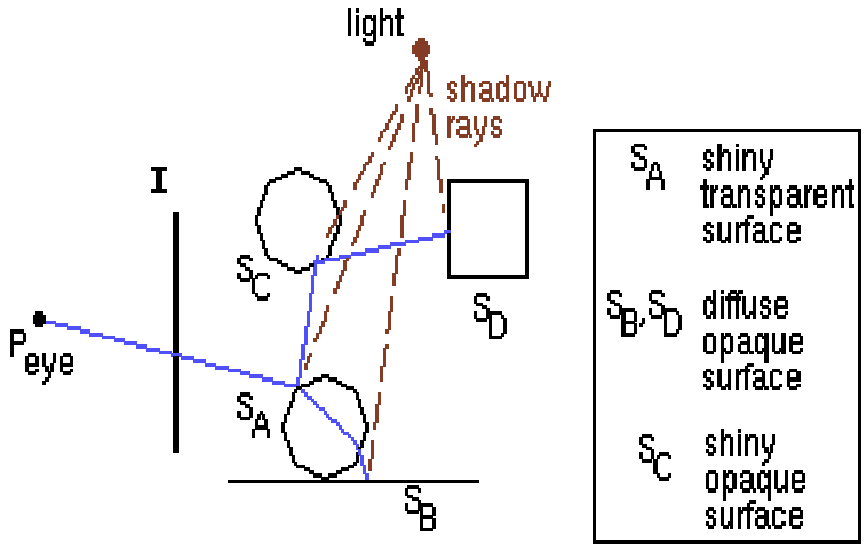
- After finding the intersection point, we need to actually see if this point lies within the polygon that described the plane.

- Use barycentric coordinates to test (not covered here).

- Try to avoid calculating intersections, by testing whether there actually will be an intersection before calculating it.

So Whats New?

- Need to do more then just calculate the local illumination for the point of intersection to make full use of ray tracing.
- Cast secondary reflection and refraction rays from point of intersections to see other effects.



Checking for Shadows

- For each light source in a scene, cast a ray from that light source to the intersection point we just calculated.
- If the ray hits an object before reaching the point, then ignore contributions from that light source.
- Add this to local illumination model:

$$I_{out} = k_{ambient} \cdot I_{ambient} + \sum b_{light} I_{light} \cdot (k_{diffuse} \cdot (\mathbf{n} \cdot \mathbf{l}) + k_{specular} \cdot (\boldsymbol{v} \cdot \mathbf{r})^{s})$$

- *b_{light} is 0 or 1 depending on whether light is obstructed or not.*

Refraction

- As we all know, lenses, glass, water and other translucent materials refract light.
- We can cast a secondary refraction ray from the intersection point if a material is translucent.

Snell's Law!

- Computer graphics uses Snell's law to compute the refracted ray, but in vector form.
- *Snell's Law:* $n_i \sin(i) = n_r \sin(r)$
- Vector form: $n_i (\mathbf{l} \times \mathbf{n}) = n_r (\mathbf{r} \times \mathbf{n})$
- *Solve for* ***r*** *(complicated derivation)*
  - $\mathbf{r} = n_i / n_r (\cos(i)) - \cos(r)\mathbf{n} - n_i / n_r \mathbf{l}$

    $= n_i / n_r (\mathbf{n} \cdot \mathbf{l}) -$

    $\sqrt{(1 - (n_i / n_r)^2 (1 - (\mathbf{n} \cdot \mathbf{l})^2)} * \mathbf{n} - n_i / n_r \mathbf{l}$

- After calculating the direction of the refracted ray, we can cast a secondary ray in this direction and recursively look for intersections and add illumination values from these other intersections to the illumination of the original intersections.
- Can do the same for reflected rays off of other surfaces by casting a ray in the direction of reflection as before:

$$\mathbf{r} = (2\mathbf{l} \cdot \mathbf{n})\mathbf{n} - \mathbf{l}$$

- These secondary illumination values will not have as much weight on the illumination of the pixel as the original illumination value, as intensity of light decreases as distance increases.

- Add a weighting factor to these secondary illumination values to account for this.

- Recurse from secondary intersections.

The Ray Tracing Algorithm

- *raytrace(ray from a pixel)*
  -calculate closest intersection
  -calculate local illumination //take shadows
  for intersection point //into account
  -reflected_component = raytrace(reflected_ray)
  // if an object surface has reflection properties (ie. is //not completely diffuse)
  -refracted_component = raytrace(refraced_ray)
  //if an object surface is transparent
  *-color_of_pixel = c1 * localcolor*
  *+ c2 * reflected_component*
  *+ c3 * refracted_component*

Cool Ray Traced Images


Ray Tracing