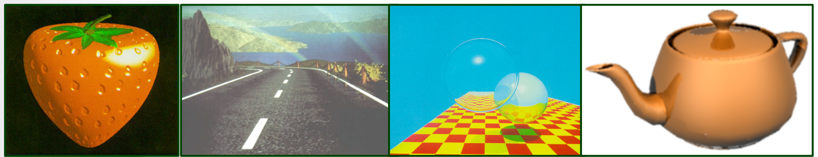


Chap. 7
Illumination-based Shading

Ensino de Informática (3326) - 4º ano, 2º semestre
Engenharia Electrotécnica (2287) - 5º ano, 2º semestre
Engenharia Informática (2852) - 4º ano, 2º semestre

Lighting Review

- Lighting Models
  - Ambient
    - Normals don’t matter
  - Lambert/Diffuse
    - Angle between surface normal and light
  - Phong/Specular
    - Surface normal, light, and viewpoint

Applying Illumination

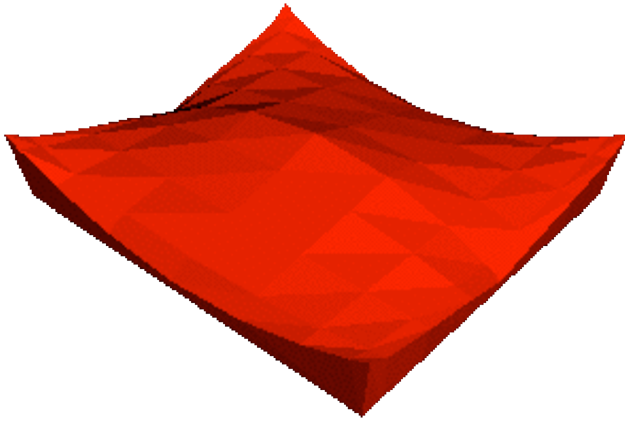
- We now have an direct illumination model for ***a single point*** on a surface
- Assuming that our surface is defined as a mesh of polygonal facets, ***which points should we use?***
  - Computing these models for every point that is displayed is expensive
  - Normals may not be explicitly stated for every point
- Keep in mind:
  - It's a fairly expensive calculation
  - Several possible answers, each with different implications for the visual quality of the result

Shading Models

- Several options:
  - **Flat** shading
  - **Gouraud** shading (interpolation)
  - **Phong** shading (interpolation)
- New hardware does per-pixel programmable shading!

Flat (or Constant) Shading

- The simplest approach, ***flat shading***, calculates illumination at a single point for each polygon.
  - OpenGL uses one of the vertices
- The illumination intensity (color) is the <u>same</u> for all points of each polygon.



- Advantages:
  - Fast - one shading value computation per polygon
- Disadvantages:
  - Inaccurate
  - Artifacts: Discontinuities at polygon boundaries

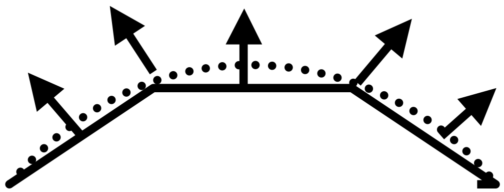
Is flat shading realistic for faceted object?

NO!

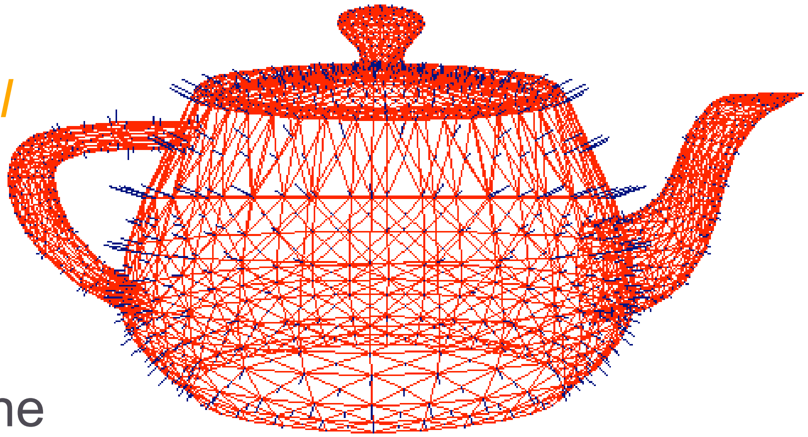
- For point sources, the direction to light varies across the facet
- For specular reflectance, direction to eye varies across the facet

Flat Shading

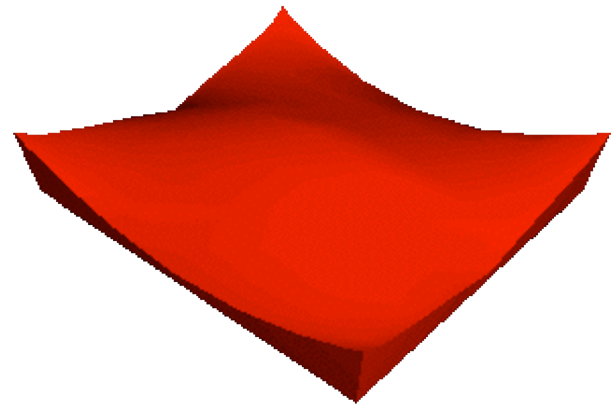
- We can refine it a bit by evaluating the Phong lighting model at each pixel of each polygon, but the result is still clearly faceted:



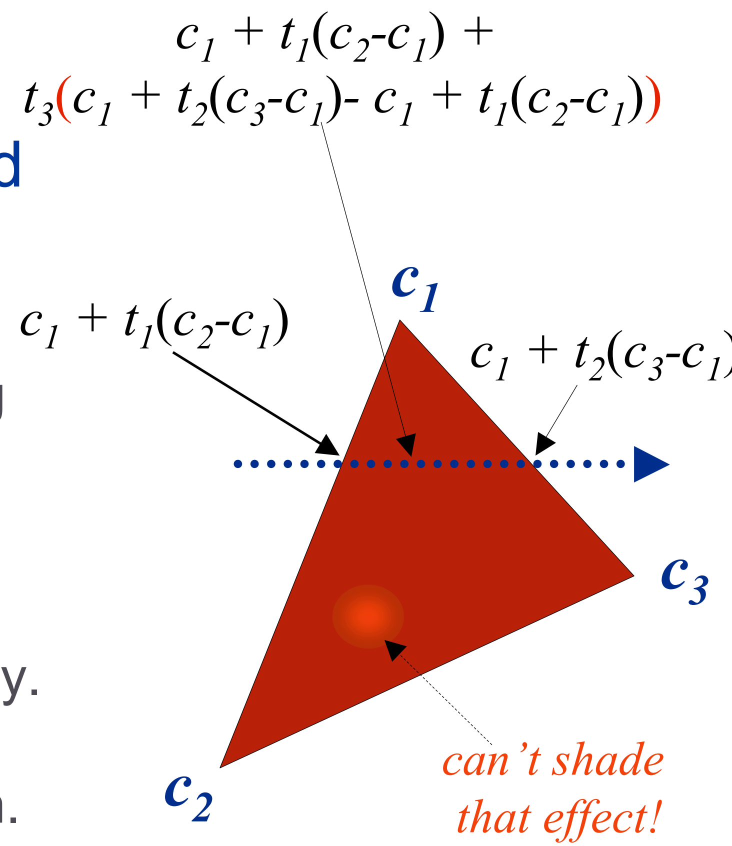
- To get smoother-looking surfaces we use ***vertex normals*** at each vertex
  - Usually different from facet normal
  - Used *only* for shading
  - Think of as a better approximation of the *real* surface that the polygons approximate
- Vertex normals may be
  - Provided with the model
  - Approximated by averaging the normals of the facets that share the vertex



Gouraud Shading



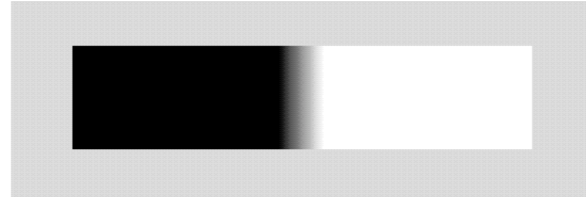
- It directly illuminates or shades each vertex by using its location and normal.
- It linearly **interpolates** the resulting colors over faces: along bounding edges first, and then along scanlines in its interior.
- Advantages:
  - Fast - incremental calculations when rasterizing
  - Much smoother - use one normal per shared vertex to get continuity between faces
- Disadvantages:
  - Still inaccurate. Polygons appear dull and chalky.
  - It tends to eliminate the specular component. If included, it will be averaged over entire polygon.
  - Mach banding.

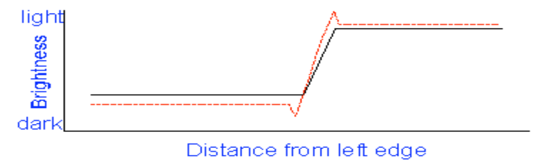


http://www.markschenk.com/various/machband.html

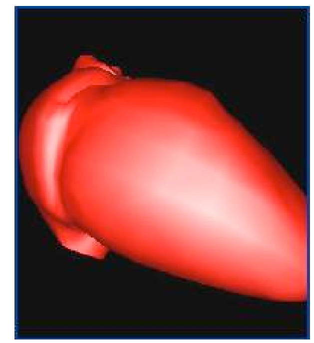
Gouraud Shading: Mach banding

- Artifact at discontinuities in intensity or intensity slope.
- The Mach banding describes an effect where the human mind subconsciously increase the contrast between two surfaces with different luminance.
- The difference between two colors is more pronounced when they are side by side and the boundary is smooth.
- This emphasizes boundaries between colors, even if the color difference is small.
- Rough boundaries are “averaged” by our vision system to give smooth variation



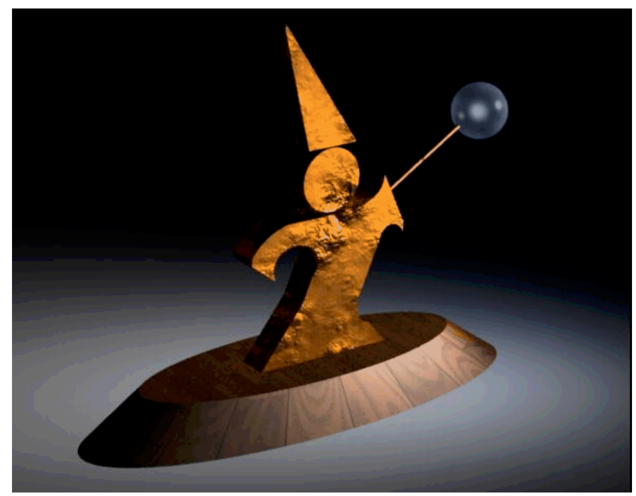


flat shading



Gouraud shading

banded along edges



floor appears banded

OpenGL shading

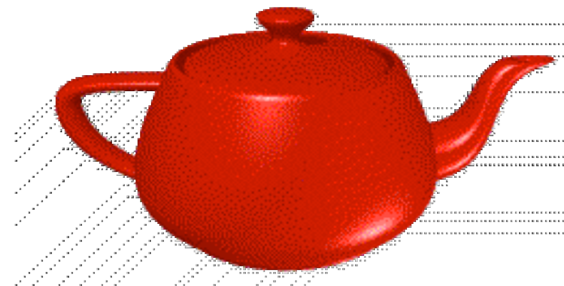
- OpenGL defines two particular shading models:
  - Controls how colors are assigned to *pixels*
  - **Gouraud shading**:*interpolates* between the colors at the *vertices* (the default)

```
glShadeModel(GL_SMOOTH)
```

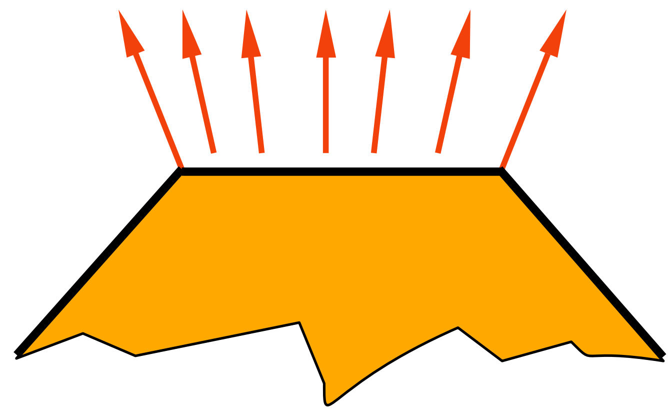
  - **Flat shading**: uses a *constant* color across the polygon

```
glShadeModel(GL_FLAT)
```

Phong Shading

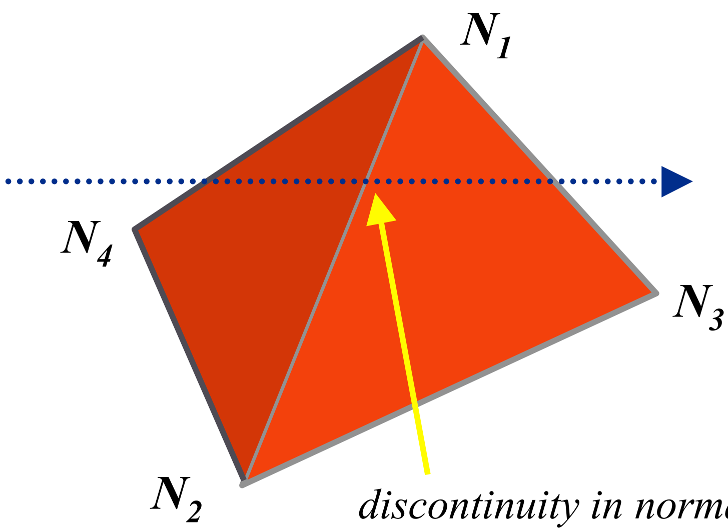


- *Phong shading* is not the same as Phong lighting, though they are sometimes mixed up
  - Phong lighting: the empirical model we've been discussing to calculate illumination at a point on a surface
  - Phong shading: linearly interpolates the surface normals across the facet, applying the Phong lighting model at every pixel
- Advantages:
  - Usually very smooth-looking results
  - High quality, narrow specularities
- Disadvantages:
  - But, considerably more expensive
  - Still an approximation for most surfaces



Phong Shading

- Linearly interpolate the vertex normals
  - Compute lighting equations at each pixel
  - Can use specular component
  - Note that normals are used to compute diffuse and specular terms



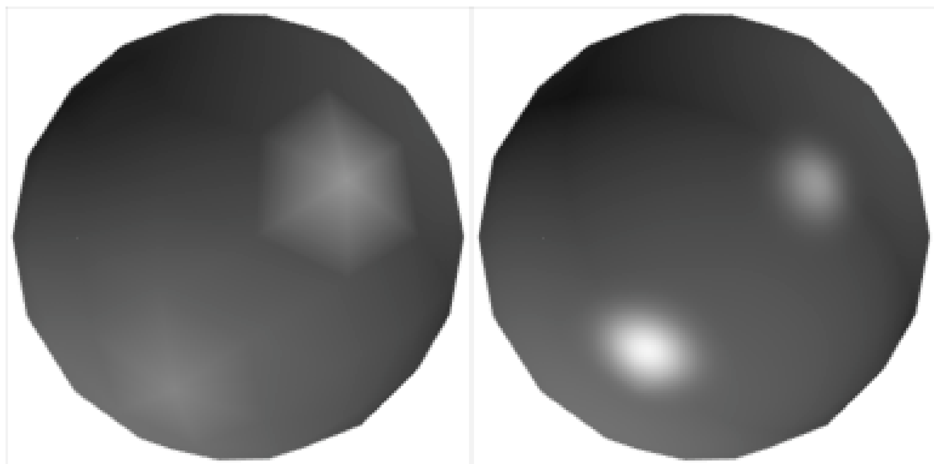
$$I_{total} = K_A I_A + \sum_{i=1}^{\#\,lights} I_i \left(K_D \left(\vec{N} \cdot \vec{L}_i \right) + K_S \left(\vec{V} \cdot \vec{R}_i \right)^n \right)$$

Shortcomings of Shading

- Polygonal silhouettes remain
- Perspective distortion
- Interpolation dependent on the polygon orientation
- Problems at shared vertices
- Bad vertex averaging

Shortcomings of Shading

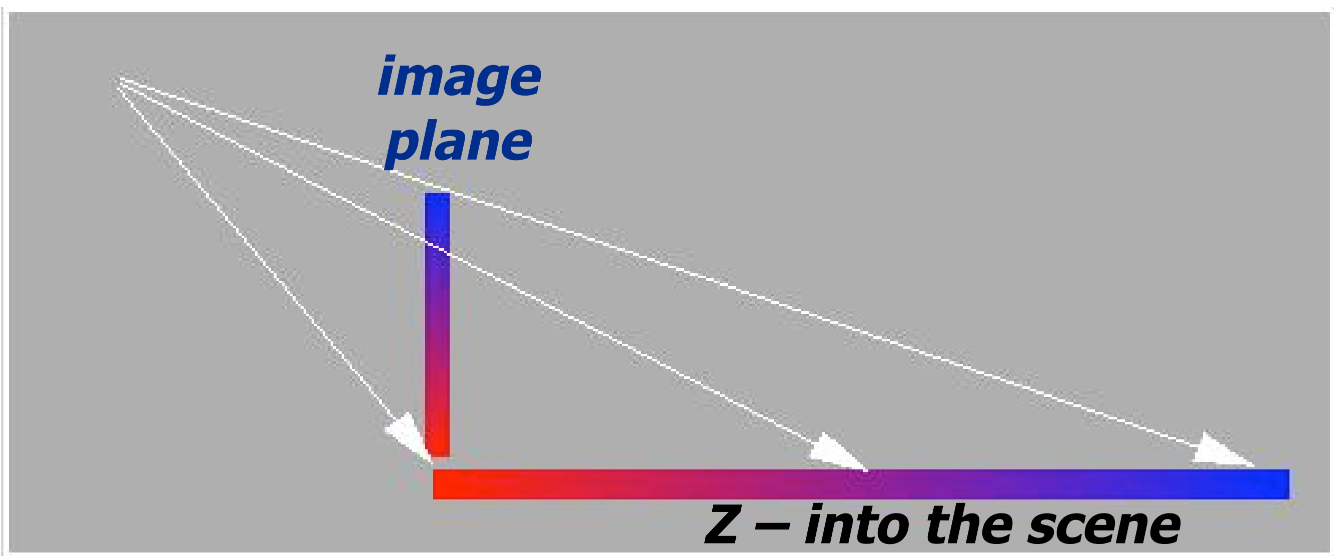
- Polygonal silhouettes remain



Gouraud *Phong*

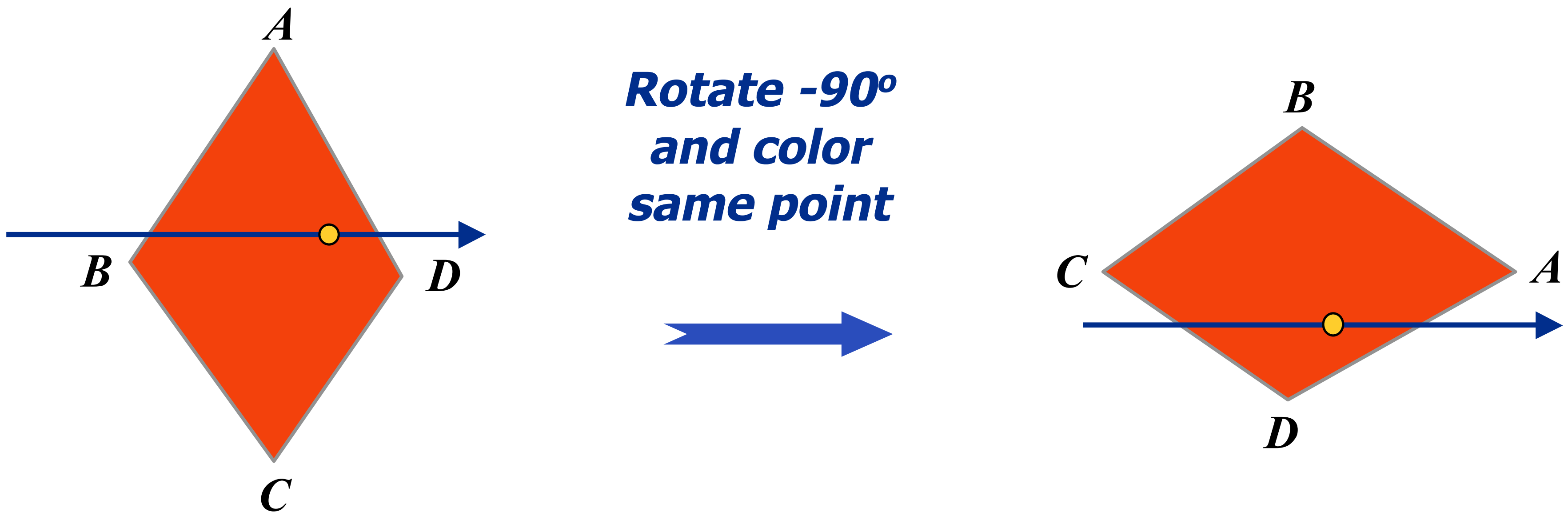
Shortcomings of Shading

- Perspective distortion
  - Note that linear interpolation in screen space does not align with linear interpolation in world space.
  - Break up large polygons with many smaller ones to reduce distortion.



Shortcomings of Shading

- Interpolation dependent on the polygon orientation

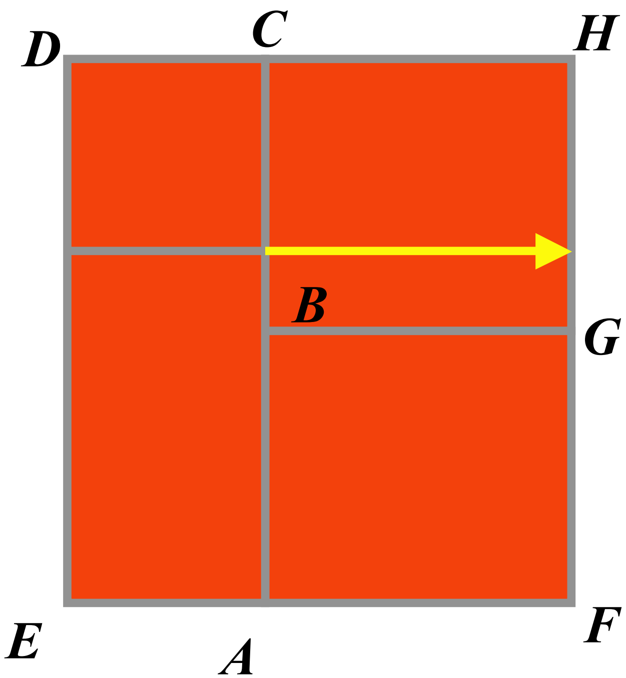


Interpolate between AB and AD

Interpolate between CD and AD

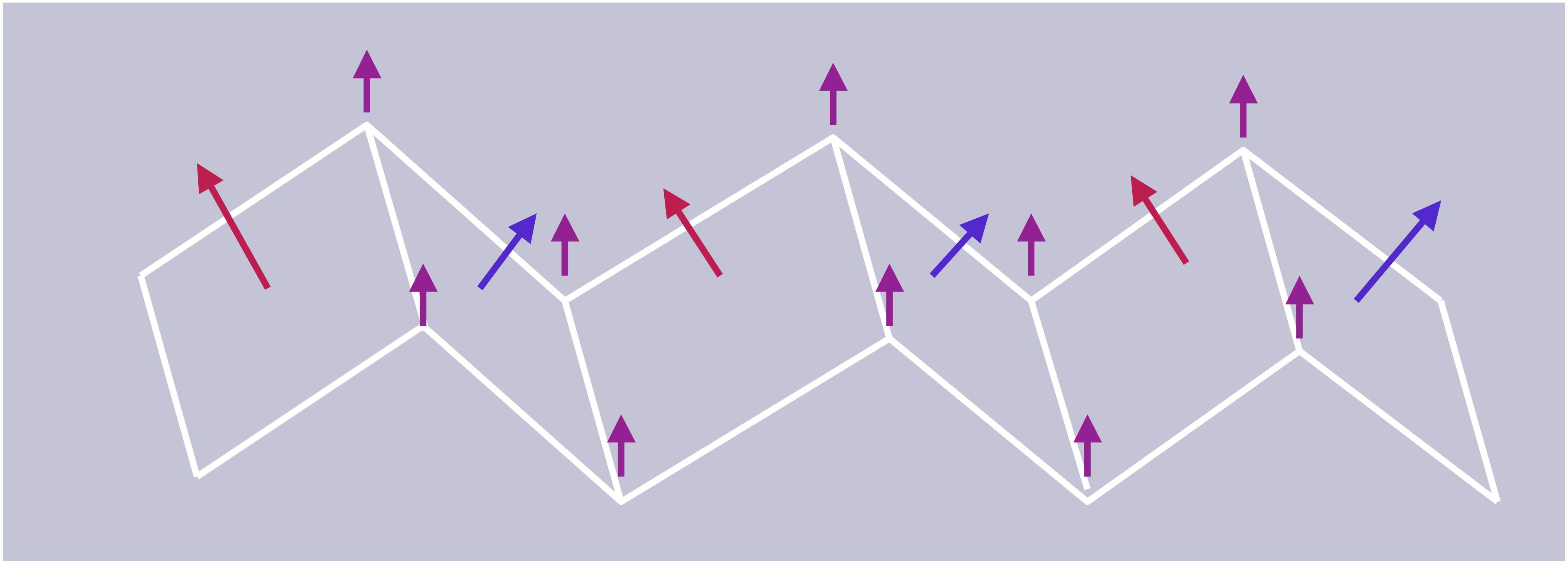
Shortcomings of Shading

- Problems at shared vertices
  - Example aside:
    - The vertex B is shared by the two rectangles on the right, but not by the one on the left
    - The first portion of the scanline is interpolated between DE and AC
    - The second portion of the scanline is interpolated between BC and GH
    - A large discontinuity could arise



Shortcomings of Shading

- Bad vertex averaging



Shading Models (Direct lighting)

summary

- Flat Shading
  - Compute Phong lighting once for entire polygon
- Gouraud Shading
  - Compute Phong lighting at the vertices and interpolate lighting values across polygon
- Phong Shading
  - Compute averaged vertex normals
  - Interpolate normals across polygon and perform Phong lighting across polygon

Current Generation of Shaders

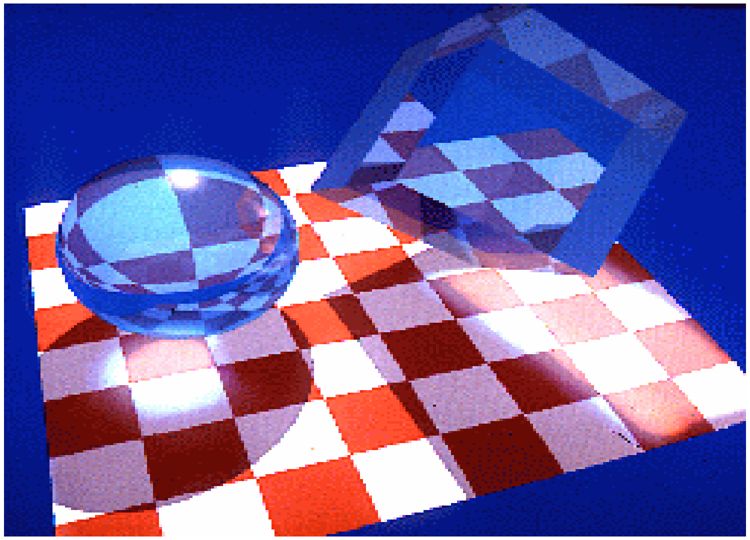
- Current hardware allows you to break from the standard illumination model
- *Programmable Vertex Shaders* allow you to write a small program that determines how the color of a vertex is computed
  - Your program has access to the surface normal and position, plus anything else you care to give it (like the light)
  - You can add, subtract, take dot products, and so on

Current Generation of Shaders

- We have only touched on the complexities of illuminating surfaces
  - The common model is hopelessly inadequate for accurate lighting (but it's fast and simple)
- Consider two sub-problems of illumination
  - Where does the light go? *Light transport*
  - What happens at surfaces? *Reflectance models*
- Other algorithms address the transport or the reflectance problem, or both
  - Much later in class, or a separate course

Overview: lighting-based models

- Direct Illumination
  - Emission at light sources
  - Scattering at surfaces
- Global Illumination
  - Shadows
  - Refractions
  - Inter-object reflections



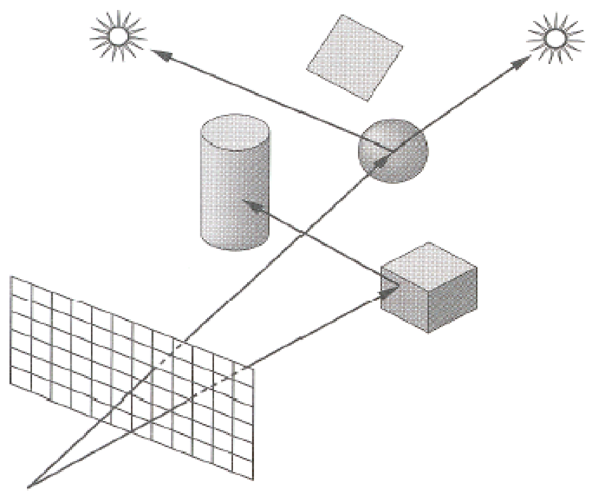
Global Illumination

- We've glossed over how light really works
- And we will continue to do so…
- One step better

- Global Illumination
  - The notion that a point is illuminated by more than light from local lights; it is illuminated by all the emitters and reflectors in the global scene

Shadows

- Shadow terms tell which light sources are blocked
  - Cast ray towards each light source L_i
  - S_i = 0 if ray is blocked, S_i = 1 otherwise

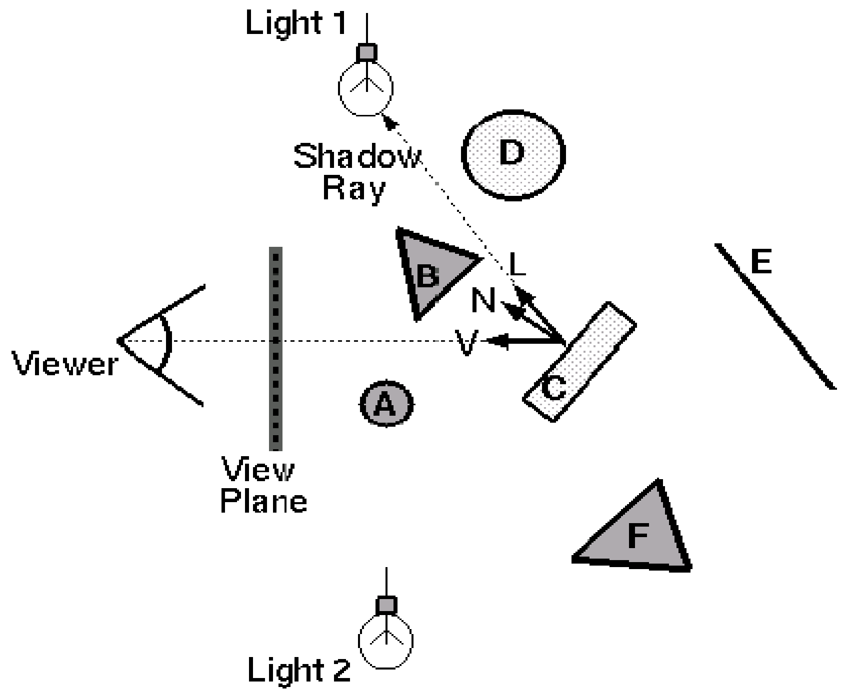


Shadow Term

$$I = I_E + K_A I_A + \sum_{i=1}^{\#\,lights} [K_D(\vec{N}\cdot\vec{L}) + K_S(\vec{V}\cdot\vec{R})^n] S_i I_i$$

Ray Casting

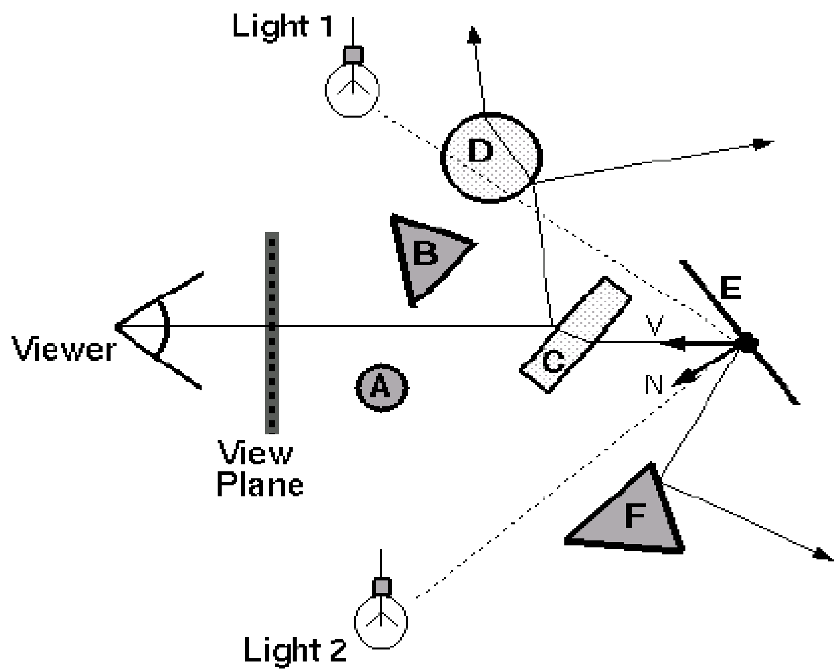
- Trace **primary rays** from camera
  - Direct illumination from unblocked lights only



$$I = I_E + K_A I_A + \sum_{i=1}^{\#\, lights} [K_D\left(\vec{N}\cdot\vec{L}\right) + K_S\left(\vec{V}\cdot\vec{R}\right)^n] S_i I_i$$

Recursive Ray Tracing

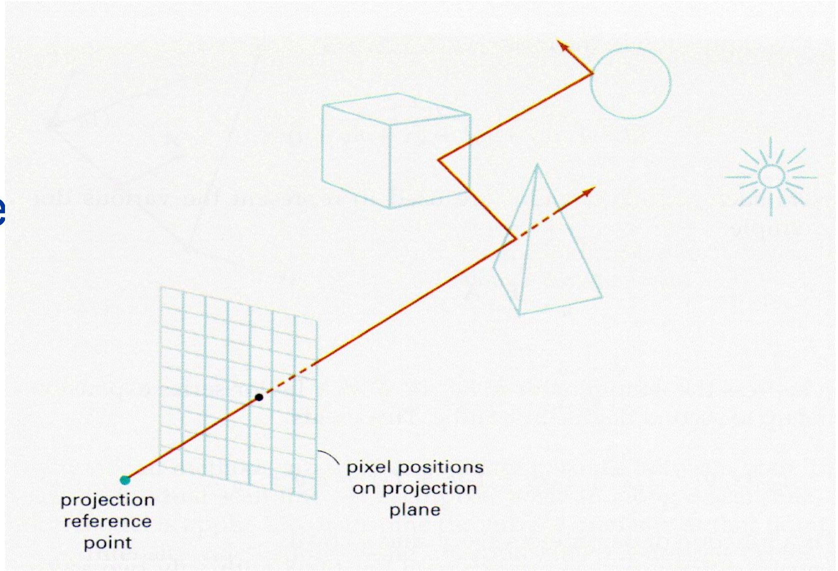
- Also trace **secondary rays** from hit surfaces
  - Global illumination from **mirror reflection** and **transparency**

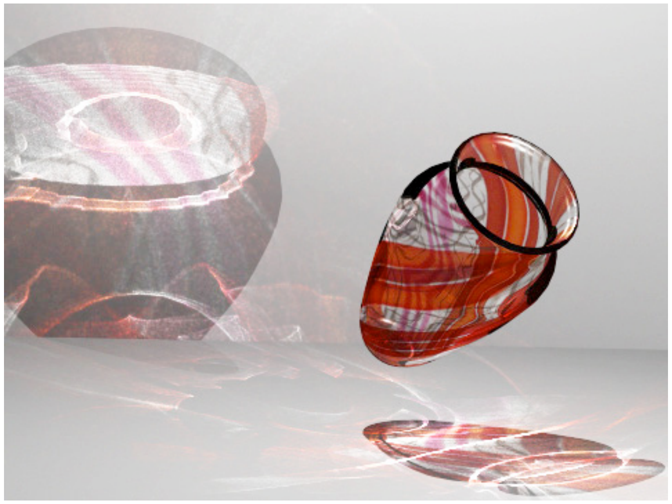


$$I = I_E + K_A I_A + \sum_{i=1}^{\#\,lights} [K_D(\vec{N}\cdot\vec{L}) + K_S(\vec{V}\cdot\vec{R})^n]S_i I_i + K_S I_R + K_T I_T$$

Recursive Ray Tracing: overview

- **Primary rays**. Cast a ray from the viewer's eye through each pixel, and then from intersected object to light sources and determine shadow/lighting conditions
- **Secondary rays**. Also spawn secondary rays
  - Reflection rays and refraction rays
  - Use surface normal as guide (angle of incidence equals angle of reflection)
  - If another object is hit, determine the light it illuminates by recursing through ray tracing
- Stop recursing when:
  - ray fails to intersect an object
  - user-specified maximum depth is reached
  - system runs out of memory



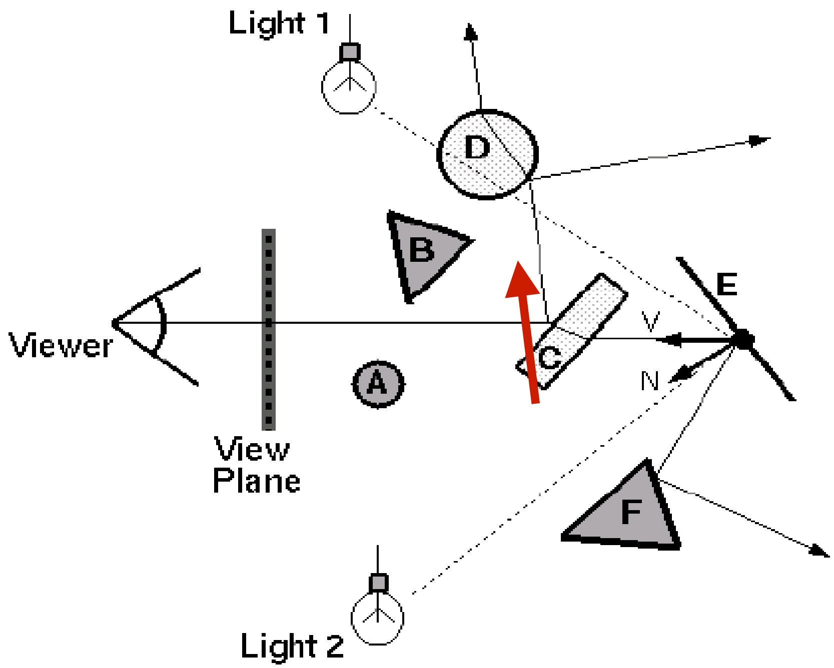


Recursive Ray Tracing

- Stop recursing when:
  - ray fails to intersect an object
  - user-specified maximum depth is reached
  - system runs out of memory
- Common numerical accuracy error
  - Spawn secondary ray from intersection point
  - Secondary ray intersects another polygon on same object

Mirror Reflection

- Trace secondary ray in direction of mirror reflection
  - Evaluate radiance along secondary ray and include it into illumination model

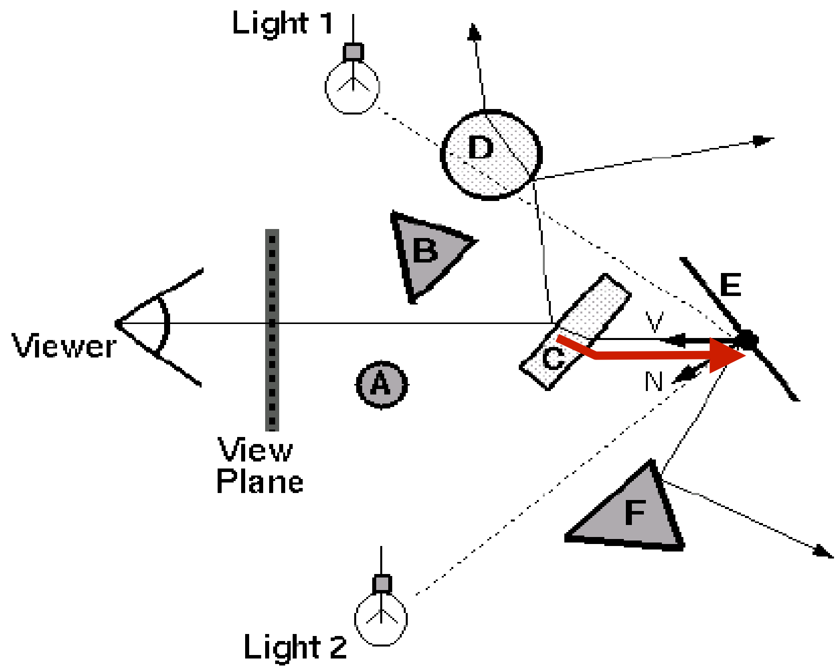


Radiance for mirror reflection ray

$$I = I_E + K_A I_A + \sum_{i=1}^{\#\, lights} [K_D(\vec{N}\cdot\vec{L}) + K_S(\vec{V}\cdot\vec{R})^n] S_i I_i + K_S I_R + K_T I_T$$

Transparency

- Trace secondary ray in direction of refraction
  - Evaluate radiance along secondary ray and include it into illumination model

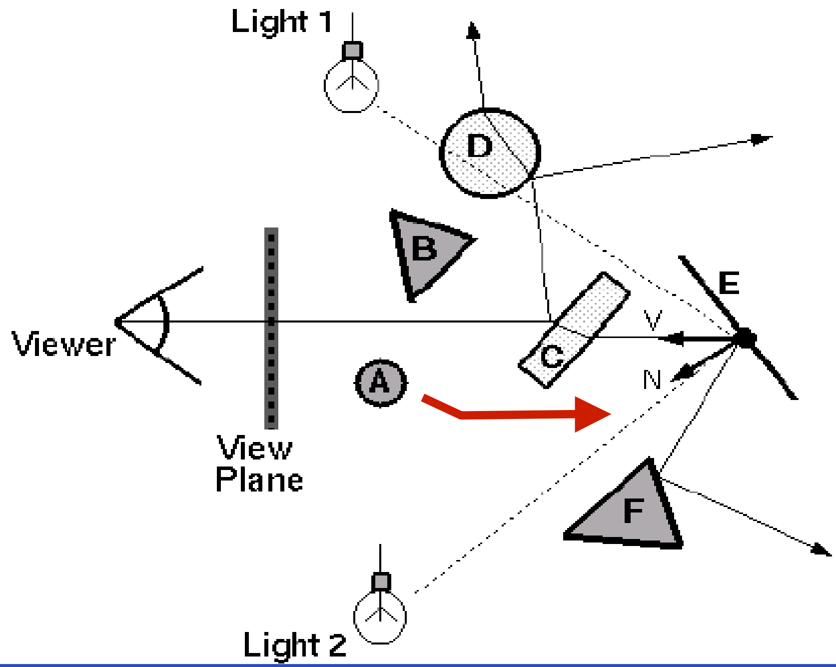


Radiance for refraction ray

$$I = I_E + K_A I_A + \sum_{i=1}^{\#\,lights} [K_D(\vec{N}\cdot\vec{L}) + K_S(\vec{V}\cdot\vec{R})^n] S_i I_i + K_S I_R + K_T I_T$$

Transparency

- Transparency coefficient is fraction transmitted
  - K_T = 1 if object is translucent, K_T = 0 if object is opaque
  - 0 < K_T < 1 if object is semi-translucent

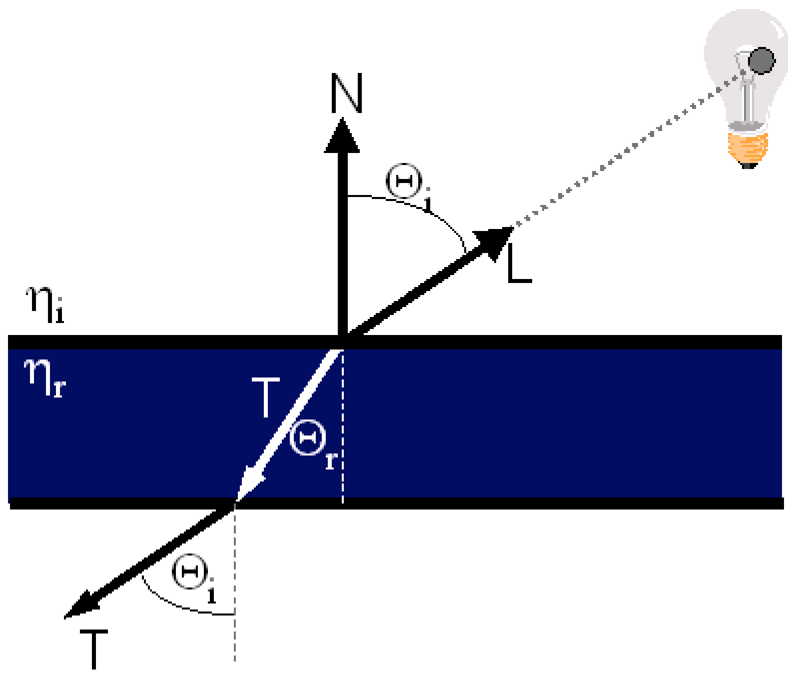


Transparency Coefficient

$$I = I_E + K_A I_A + \sum_{i=1}^{\# lights} [K_D(\vec{N}\cdot\vec{L}) + K_S(\vec{V}\cdot\vec{R})^n]S_i I_i + K_S I_R + K_T I_T$$

Refractive Transparency

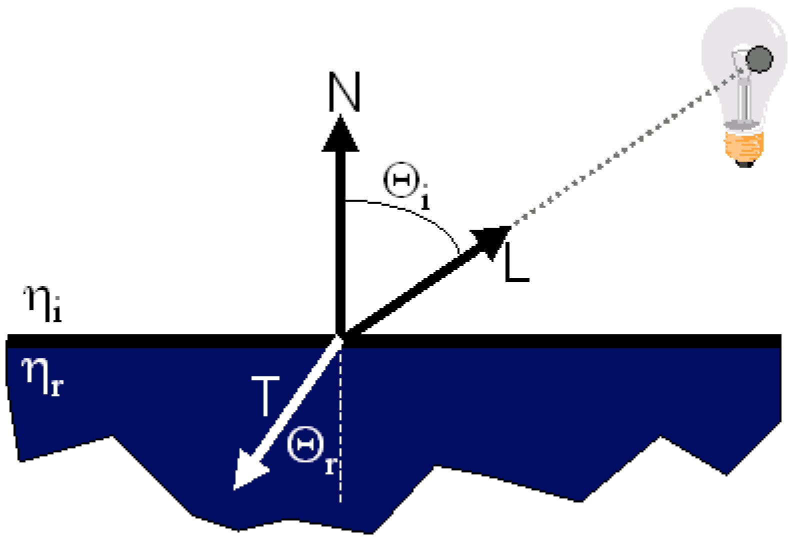
- For thin surfaces, can ignore change in direction
  - Assume light travels straight through surface



$$\boxed{T \cong -L}$$

Refractive Transparency

- For solid objects, apply Snell's Law:
  - $\eta_r \sin \Theta_r = \eta_i \sin \Theta_i$



$$T = \left(\frac{\eta_i}{\eta_r}\cos\Theta_i - \cos\Theta_r\right)N - \frac{\eta_i}{\eta_r}L$$

Radiosity

- Ray tracing models specular reflection and refractive transparency, but still uses an ambient term to account for other lighting effects
- Radiosity is the rate at which energy is emitted or reflected by a surface
- By conserving light energy in a volume, these radiosity effects can be traced

Summary

- Direct Illumination-based Shading
  - Ray casting
    - Usually use simple analytic approximations for light source emission and surface reflectance
- Indirect illumination-based Shading
  - Recursive ray tracing
    - Incorporate shadows, mirror reflections, and pure refractions
  - Radiosity
    - Use energy conservative law.

FIM