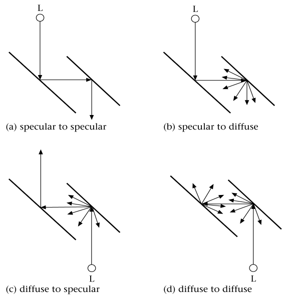
Ray-tracing

Overview

- *Recursive Ray Tracing*
- *Shadow Feelers*
- *Snell's Law for Refraction*
- *When to stop!*

Recap: Light Transport

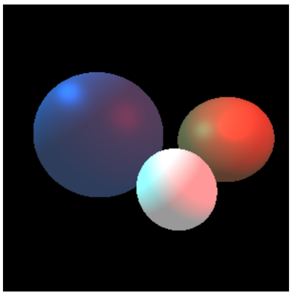


Recap: Local Illumination

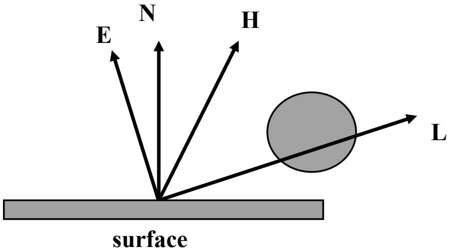
$$I_r = k_a I_a + \sum_{j=1}^{M} I_{i,j}\left(k_d\left(n \cdot l_j\right) + k_s\left(h_j \cdot n\right)^m\right)$$

- Ambient, diffuse & specular components
- The sum is over the specular and diffuse components for each light

Recap: Result of Ray Casting



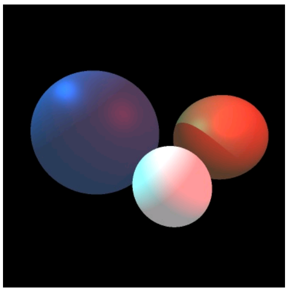
Correcting for Non-Visible Lights



$$I_r = k_a I_a + \sum_{j=1}^{M} S_j I_{i,j} \left(k_d \left(n \cdot l_j \right) + k_s \left(h_j \cdot n \right)^n \right)$$

- Where Sj is the result of intersecting the ray L with the scene objects
- Note consider your intersection points along the ray L carefully
  - Hint – they might be beyond the light!

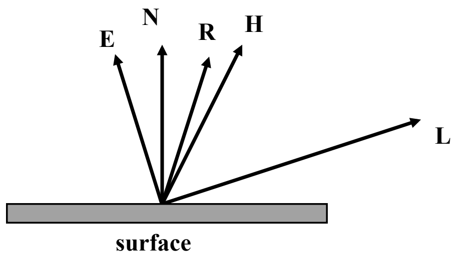
Result of Shadow Feeler



Recursive Ray-Tracing

- We can simulate specular-specular transmission elegantly by recursing and casting secondary rays from the intersection points
- We must obviously chose a termination depth to cope with multiple reflections

Introducing Reflection



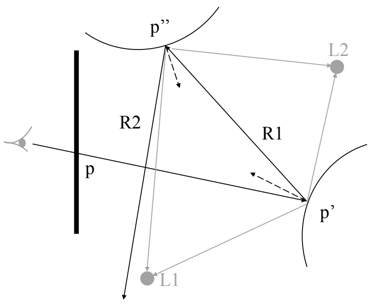
- *Where* $R = -E + 2(N \cdot E)N$

Computing Reflectance

$$I_r = I_{local} + k_r I_{r'}$$

- *Where I_{local} is computed as before*
- *Ray r' is formed from intersection point and the direction R and is cast into the scene as before*

Recursive Ray Tracing

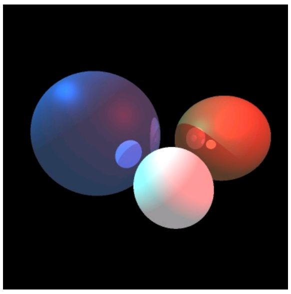


Pseudo Code

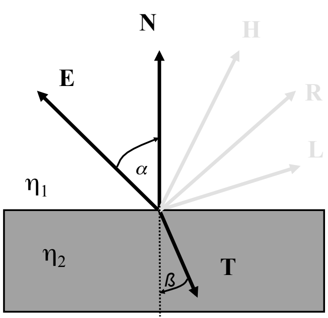
```
Color RayTrace(Point p, Vector direction, int depth) {
   Point pd    /* Intersection point */
   Boolean intersection
   if (depth > MAX) return Black
   intersect(p,direction, &pd, &intersection)
   if (!intersection) return Background
   I_local = k_a I_a + I_p.v.(k_d(n.l) + k_s.(h.n)^m)
   return I_local + k_r*RayTrace(pd, R, depth+1)
}
```

Normally $k_r = k_s$

Result of Recursion



Perfect Specular Transmission



Snell's Law

$$\frac{\sin\alpha}{\sin\beta} = \frac{\eta_2}{\eta_1}$$

η *is index of refraction*

Using Snell's Law

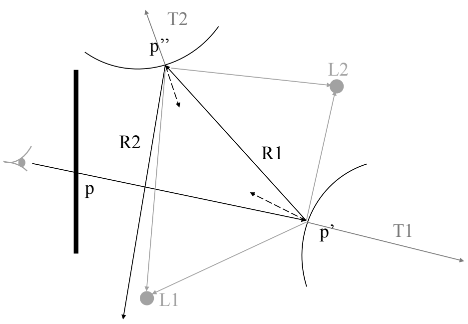
$$\frac{\sin\alpha}{\sin\beta} = \frac{\eta_2}{\eta_1} = \eta_{21}$$

- *Using this law it is possible to show that:*

$$T = -\eta_{12}E + N\left(\eta_{12}\cdot\cos\alpha - \sqrt{1+\eta_{12}^2\cdot\left(\cos^2\alpha - 1\right)}\right)$$

- *Note that if the root is negative then total internal reflection has occurred and you just reflect the vector as normal*

Recursive Ray Tracing Including Transparent Objects

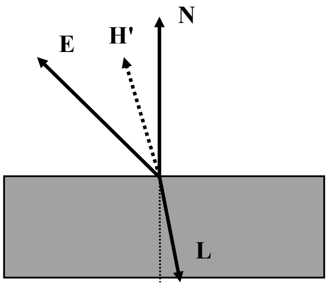


New Pseudo Code

```
Color RayTrace(Point p, Vector D, int depth) {
   Point pd    /* Intersection point */
   Boolean intersection
   if (depth > MAX) return Black
   intersect(p,direction, &pd, &intersection)
   if (!intersection) return Background
   I_local = k_a I_a + I_p.v.(k_d(n.l) + k_s.(h.n)^m)
   return I_local + k_r*RayTrace(pd, R, depth+1) +
          k_t*RayTrace(pd, T, depth+1)
}
```

Direct Specular Transmission

- *A transparent surface can be illuminated from behind and this should be calculated in I_{local}*

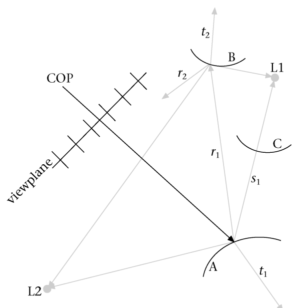


Calculating H′

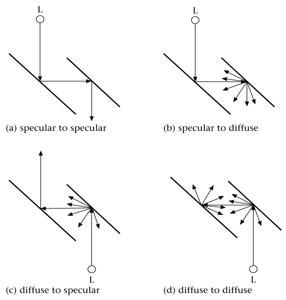
$$H' = \frac{E - \frac{\eta_2}{\eta_1} L}{\frac{\eta_2}{\eta_1} - 1}$$

- *Use H‘ instead of H in specular term*

Putting Everything Together



Discussion – What Can't We Simulate?



Remark

- Specular and transmission only
  - What should be added to consider diffuse reflection?
- Why it's expensive
  - Intersection of rays with polygons (90%)
- How to reduce the cost?
  - Reduce the number of rays
  - Reduce the cost on each ray
    - First check with bounding box of the object
    - Methods to sort the scene and make it faster

Summary

- Recursive ray tracing is a good simulation of specular reflections
- We've seen how the ray-casting can be extended to include shadows, reflections and transparent surfaces
- However this is a very slow process and still misses some types of effect!