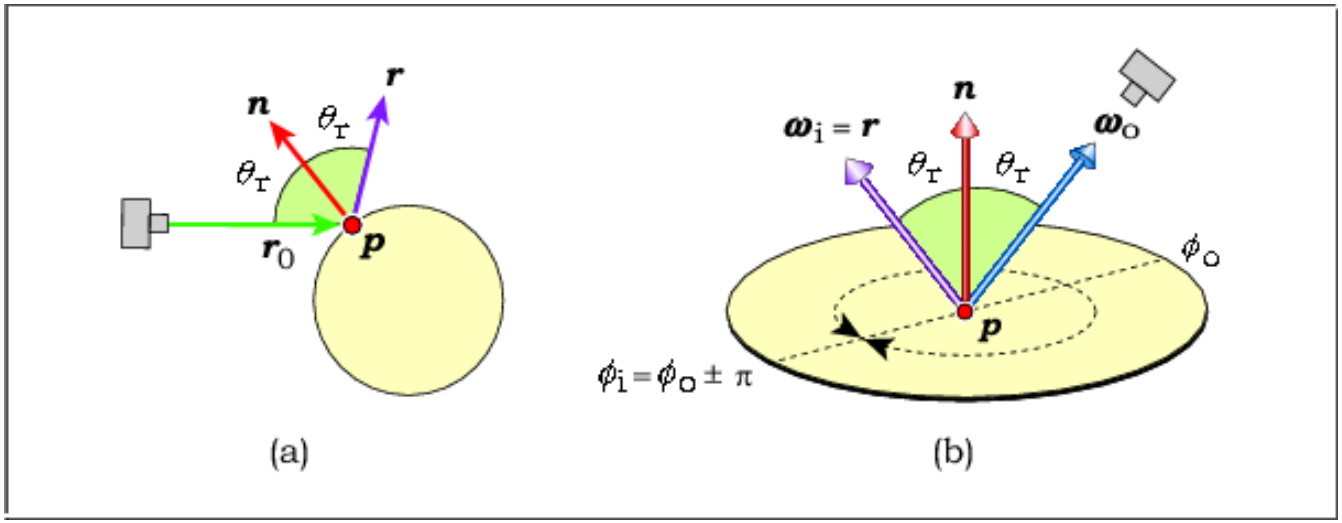
CS 563 Advanced Topics in Computer Graphics
Mirror Reflection

by Steve Olivieri

Reflection

- Reflections provide us with a way to model indirect illumination in ray traced images.
  - Some objects may appear more than once.
  - Objects the camera cannot see may appear.
- Perfect mirror reflection is the simplest model.



Mirror Reflection

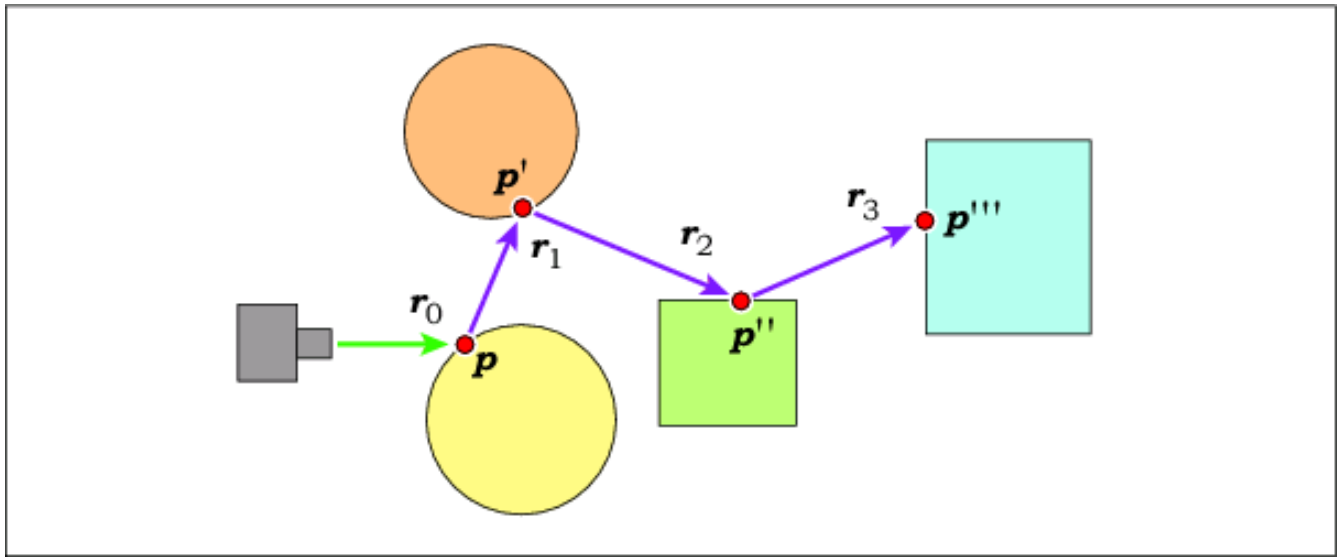
- Physically correct models involve complex integrals (e.g. Fresnel in Chapter 28).
- We can model mirror reflection with only two parameters.
  - Kr : reflection coefficient
  - Cr : reflection color

Multiple Bounces

- In a scene with multiple reflective surfaces, a ray may bounce more than once.
- Consider the reflected ray, ***r***, which bounces off of point ***p***.
  - Hit nothing, return the background color to ***p***.
  - Hit a light source, return L_e to ***p***.
  - Hit a non-reflective object, return the direct illumination at ***p'*** to ***p***.
  - Hit another reflective object, calculate the direct illumination at ***p'*** and then create reflected ray r_2. Accumulate illumination.

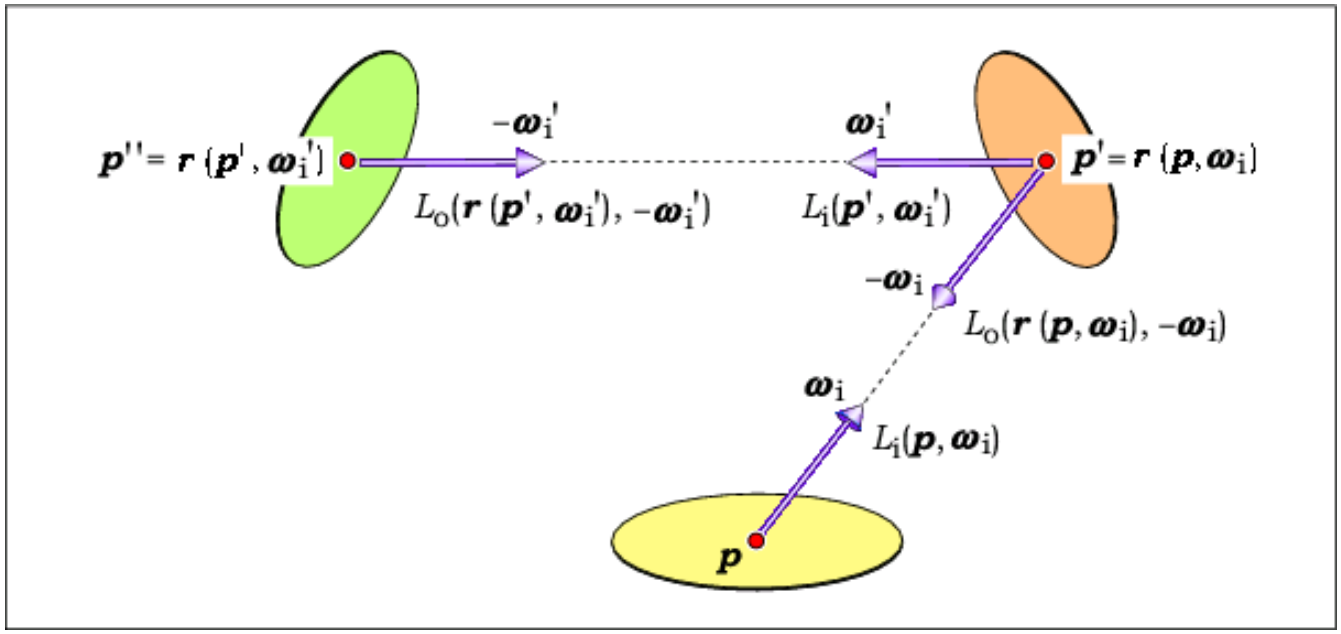
Multiple Bounces

One ray might bounce multiple times.



Multiple Bounces

One ray might bounce infinitely!



Whitted Tracer

- Developed by Turner Whitted of Bell Laboratories in 1980.
- Support for secondary rays in addition to primary and shadow rays.
- Recursive!
- No need to alter cameras or material shade functions.

Whitted Tracer

- Recall the current RayCast trace_ray() function.

```
RGBColor RayCast::trace_ray(Ray& ray) {
   ShadeRec sr(world_ptr->hit_objects(ray));

   if(sr.hit_an_object) {
        sr.ray = ray;
        return (sr.material_ptr->shade(sr));
   } else {
        return (world_ptr->background_color);
   }
}
```

Whitted Tracer

- Whitted: add a depth parameter!

```
RGBColor Whitted::trace_ray(Ray ray, int depth) {
   if(depth > world_ptr->vp.max_depth)
        return (black);

   ShadeRec sr(world_ptr->hit_objects(ray));

   if(sr.hit_an_object) {
        sr.depth = depth;
        sr.ray = ray;
        return (sr.material_ptr->shade(sr));
   } else {
        return (world_ptr->background_color);
   }
}
```

Reflective Material

- Where's the recursion?
- Create a reflective material that calls trace_ray() in its shade() function!
- "Recursion by stealth"

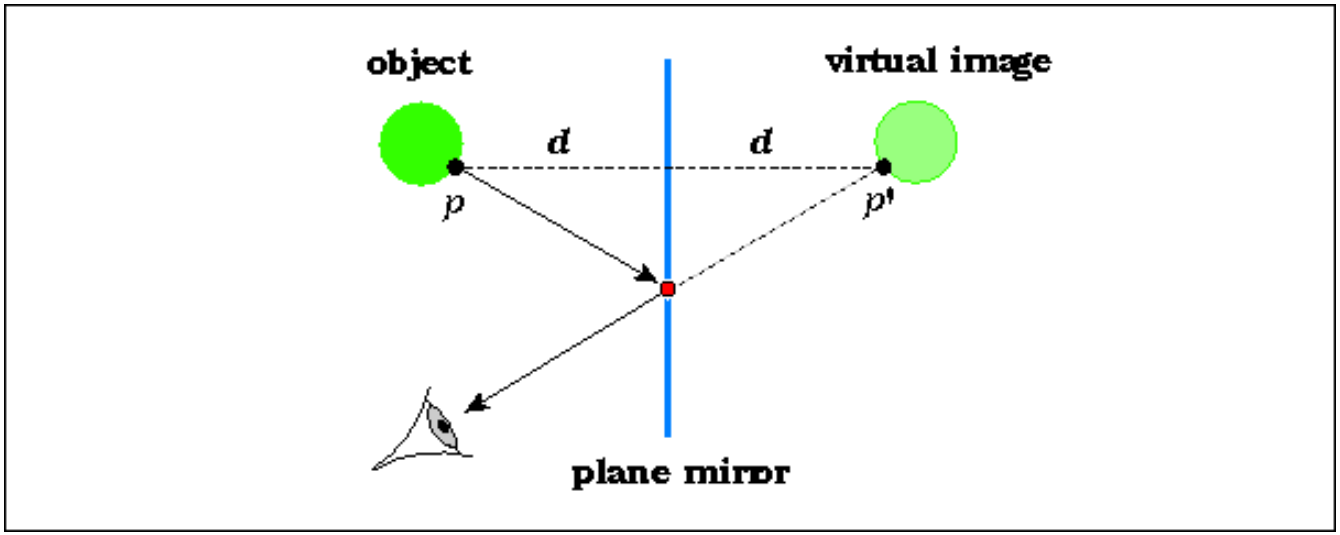
```
RGBColor Reflective::shade(ShadeRec& sr) {
   RGBColor L(Phong::shade(sr));

   Vector3D wo = -sr.ray.d, wi;
   RGBColor fr = reflective_brdf->sample_f(sr, wo, wi);
   Ray reflected_ray(sr.hit_point, wi);

   L += fr * sr.w.tracer_ptr->trace_ray(reflected_ray,
        sr.depth + 1) * (sr.normal * wi);
   return (L);
}
```

A few notes…

- Direct illumination models glossy reflection (Phong model), indirect illumination models perfect reflection.
- Mismatched coefficients – k_r should equal k_s!
- One can bypass the Phong material by setting $k_a = k_d = k_s = 0$.
  - Use a white color for a mirror
  - Use a non-white color for colorful reflective surface
  - These objects receive only indirect illumination

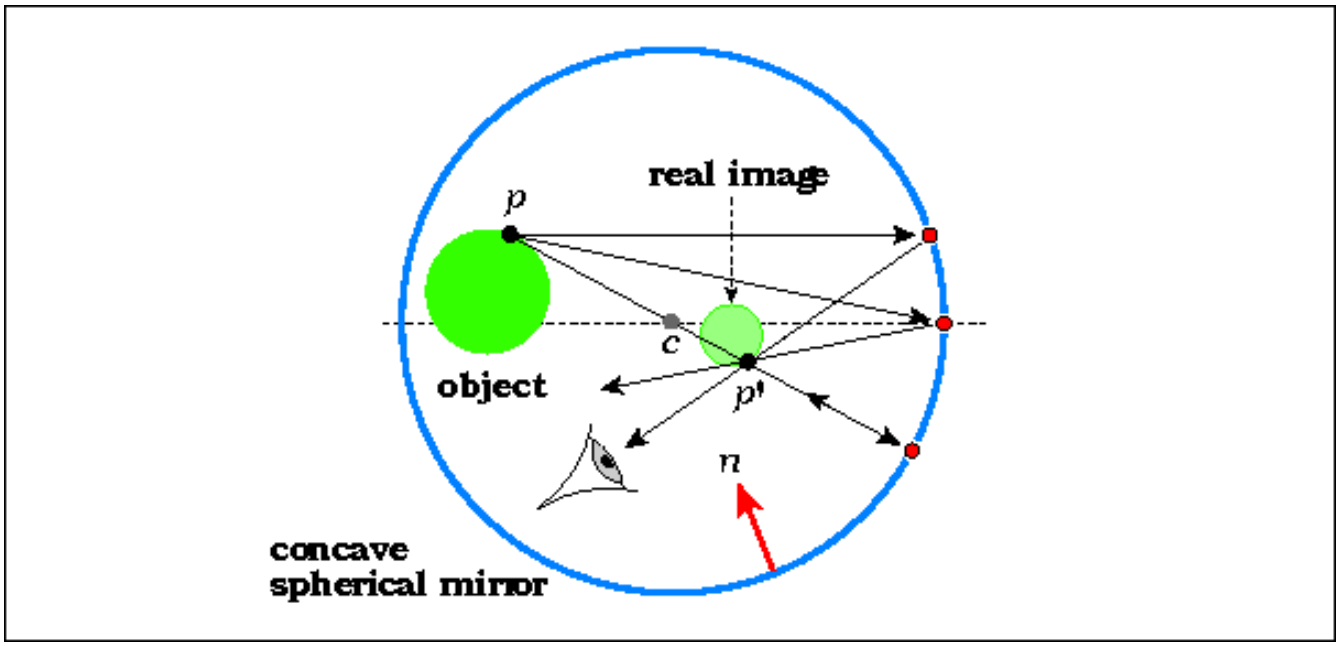
The Optics Connection

- There are two types of “images” in optics literature.
- Plane and convex mirrors form only virtual images.
  - No light comes from the image.
  - Light rays never actually touch the image, but appear to.



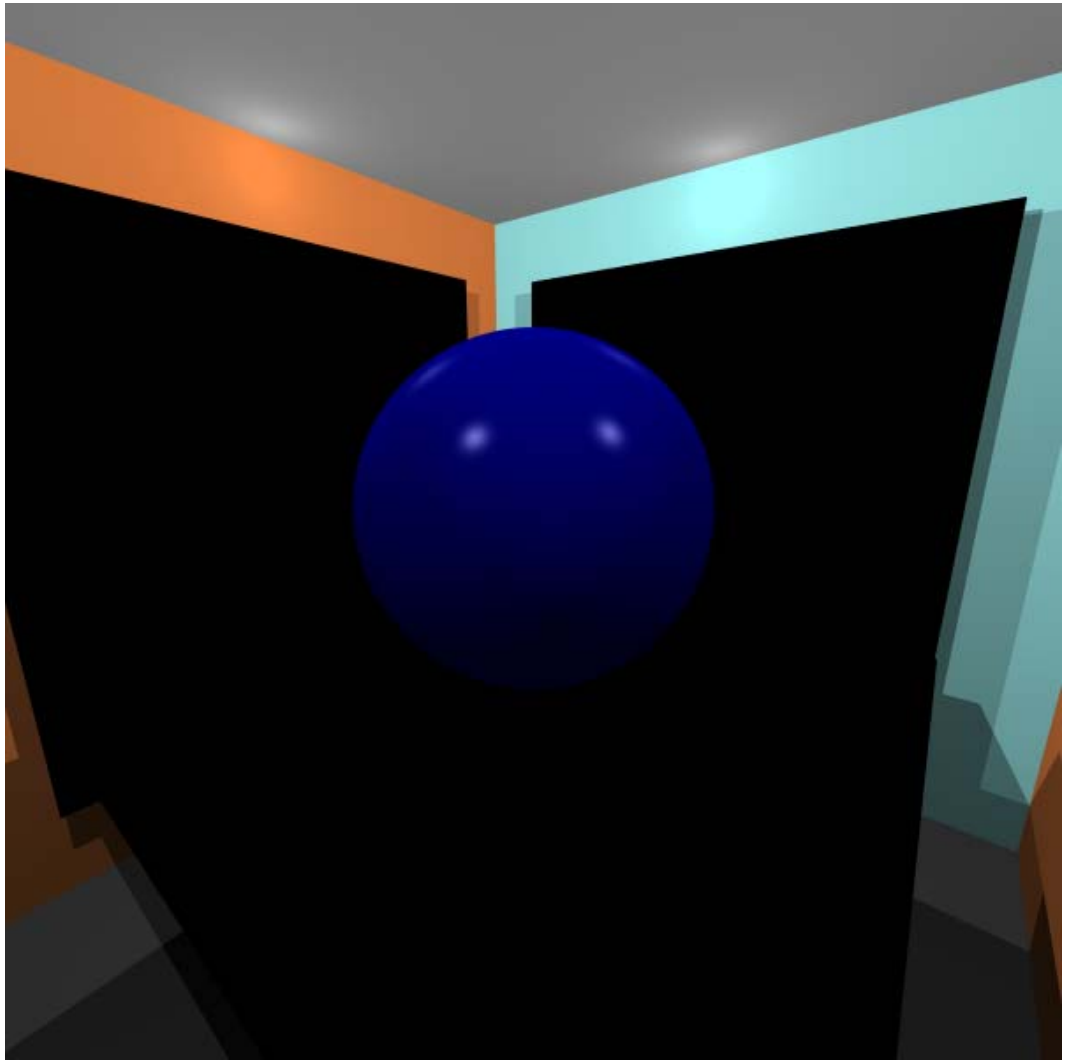
The Optics Connection

- Concave mirrors can also create real images.
- Lights rays actually pass through the image.



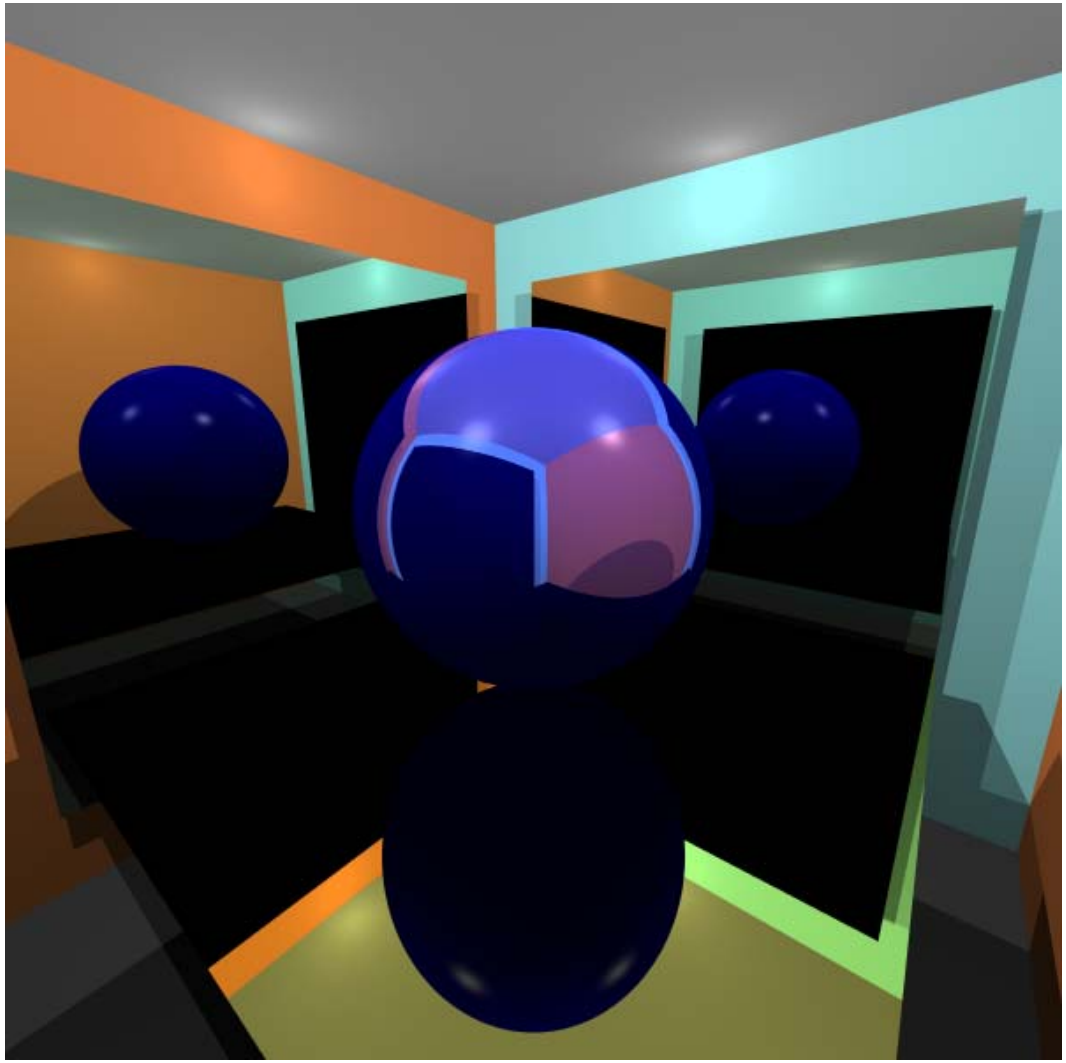
Hall of Mirrors

- Hall of Mirrors, depth = 0



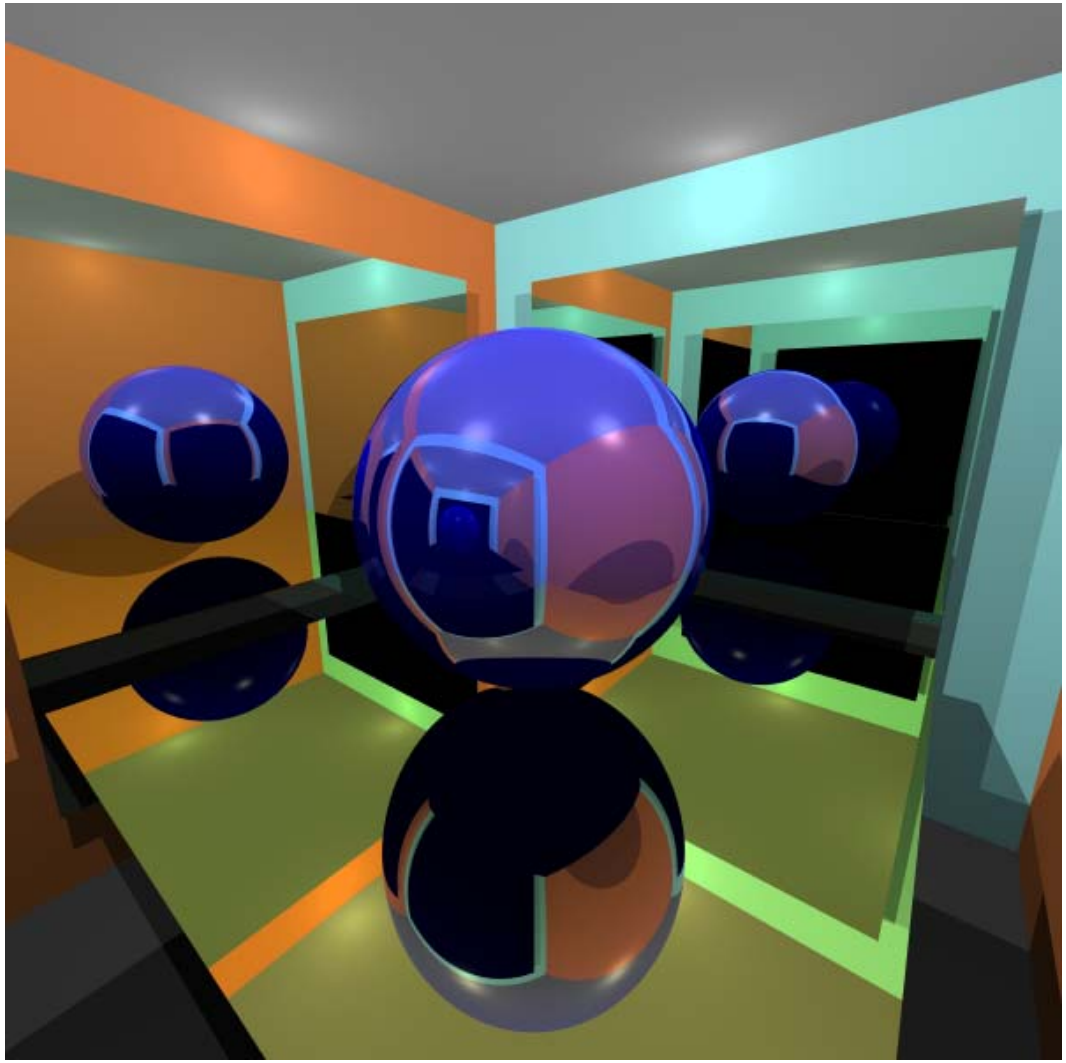
Hall of Mirrors

- Hall of Mirrors, depth = 1



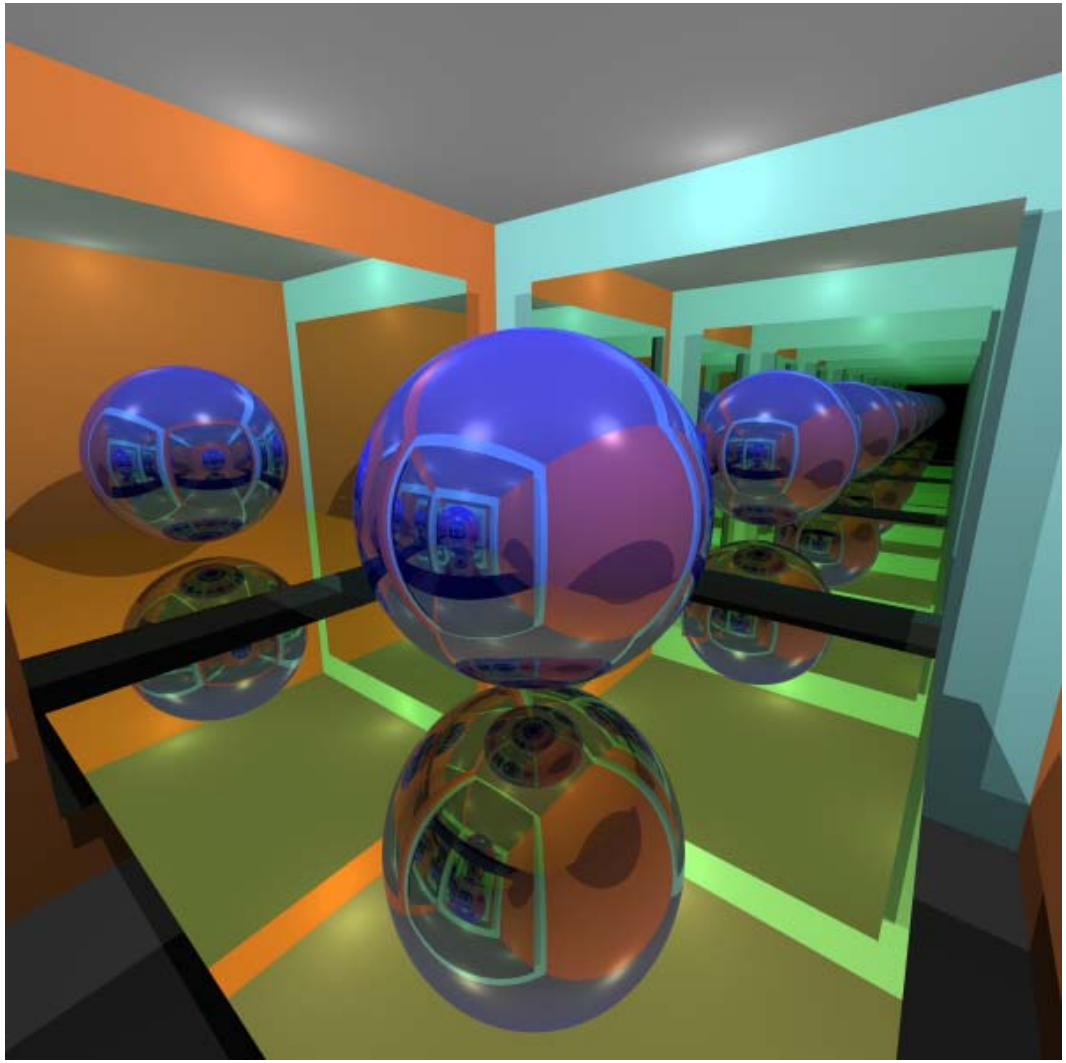
Hall of Mirrors

- Hall of Mirrors, depth = 2



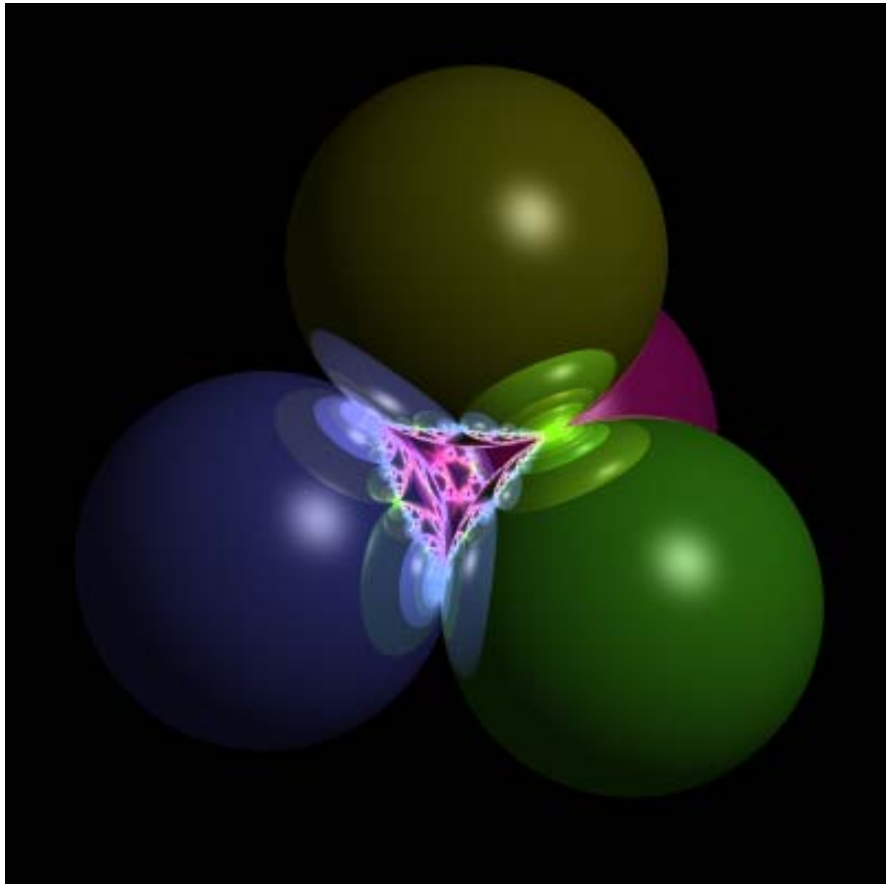
Hall of Mirrors

- Hall of Mirrors, depth = 10



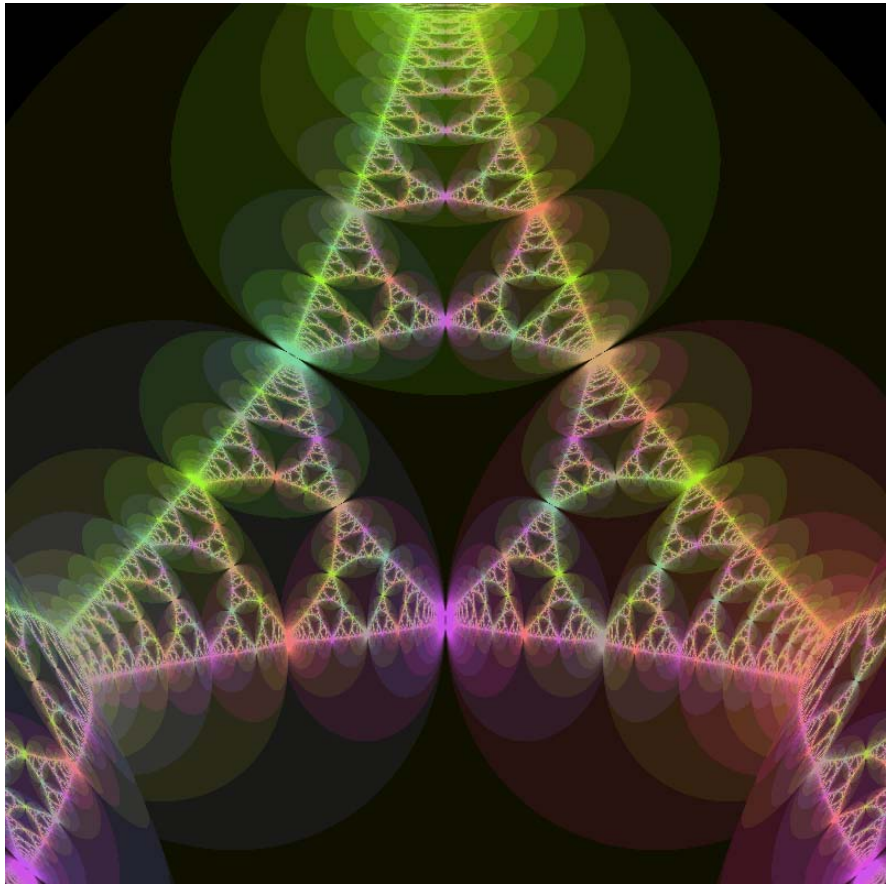
Four Orbs

- Four reflective Phong orbs, depth = 12



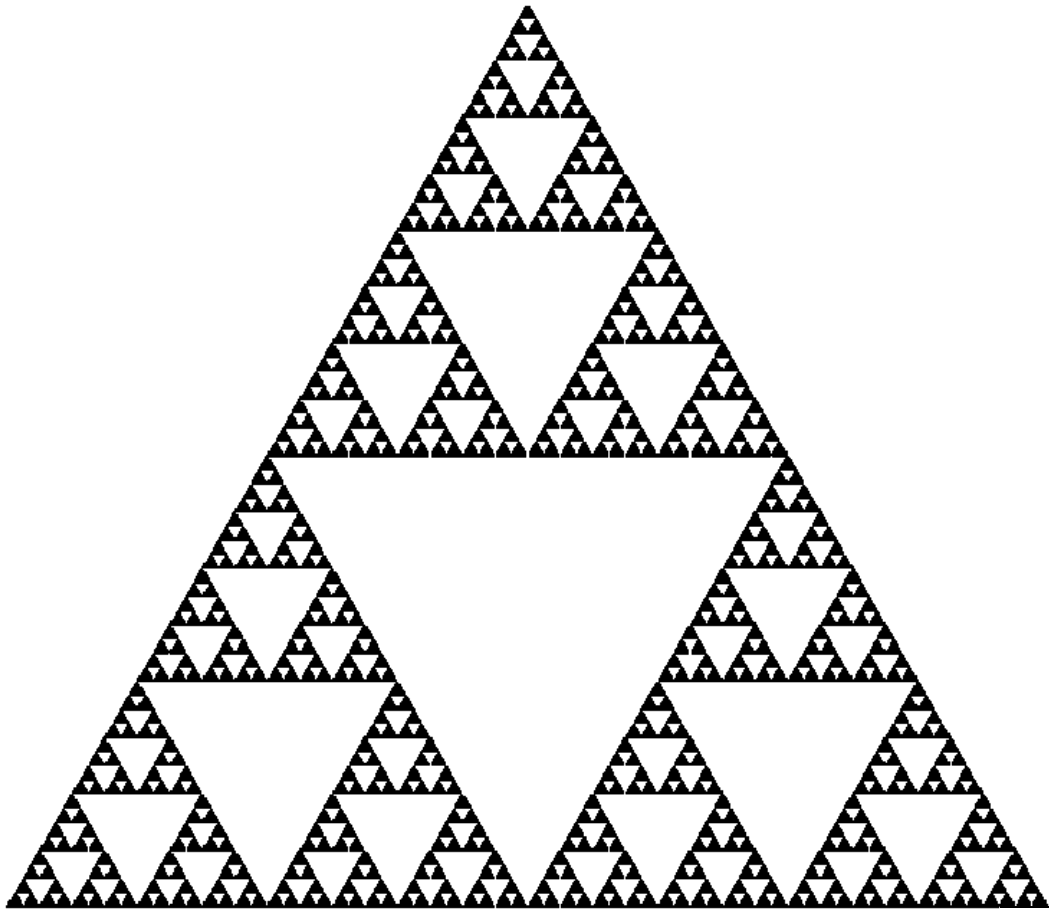
Four Orbs

- Zoomed in on four reflective Phong orbs, depth = 12



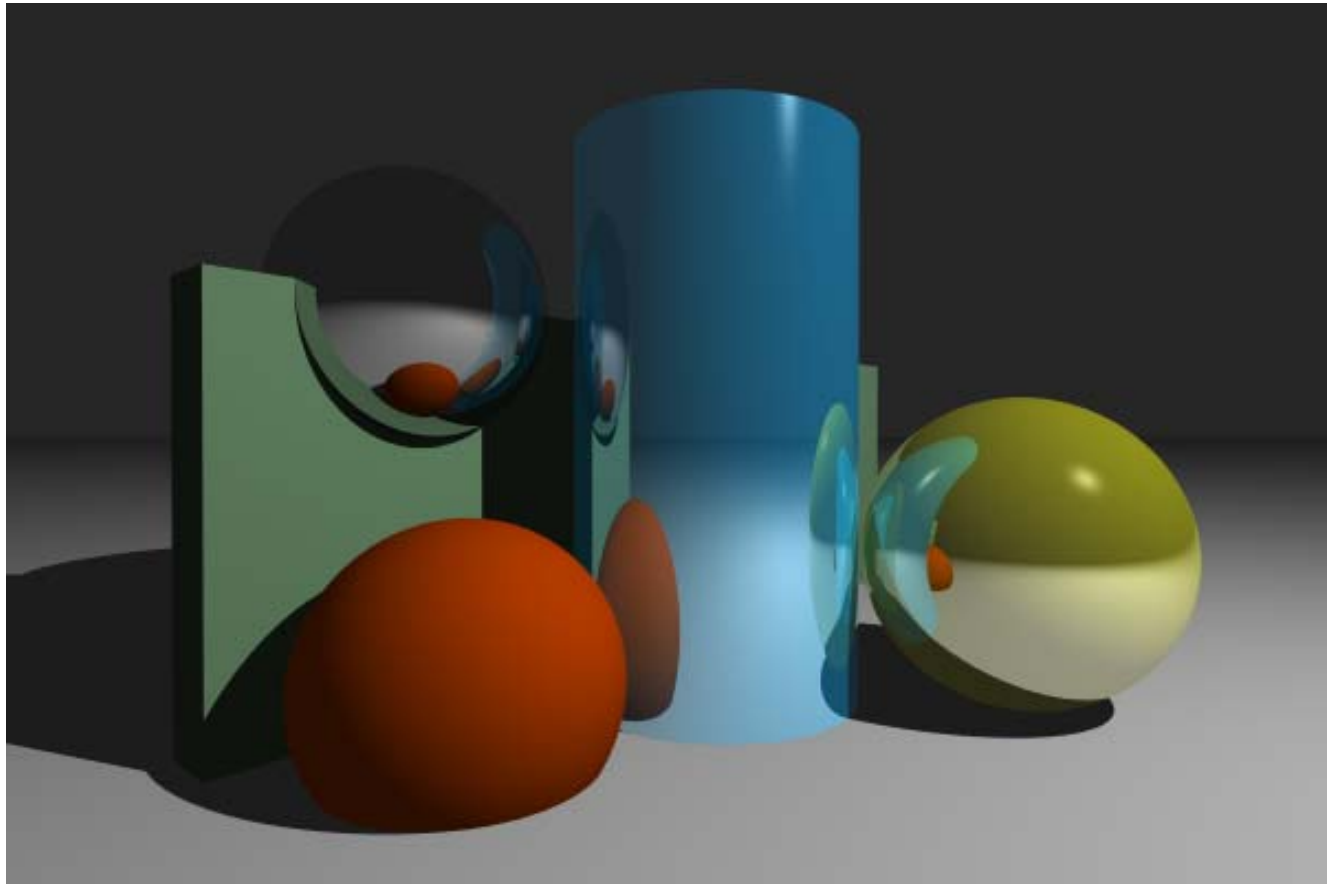
Problems

- Sierpinski Gasket



Reflective Shapes

- Reflective shapes, depth = 5



QUESTIONS?