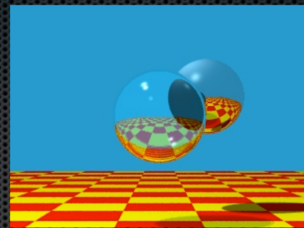
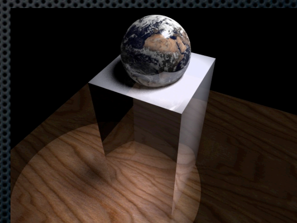


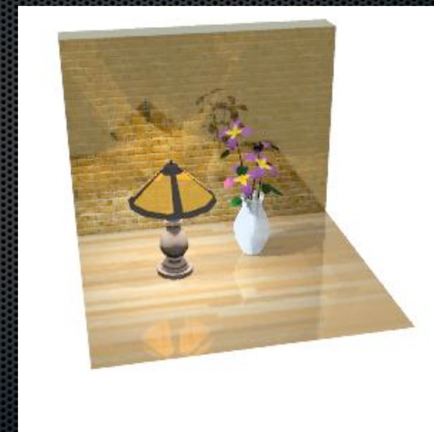
Ray tracing

(yet another example of recursion)



The problem

- Want to generate synthetic pictures with reflection, refraction, and shadows.

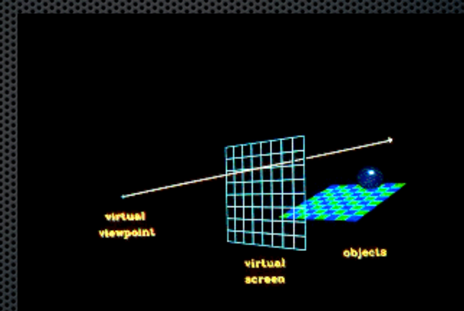


Basic approach

- Trace 'light rays' from the eye through a screen and follow them

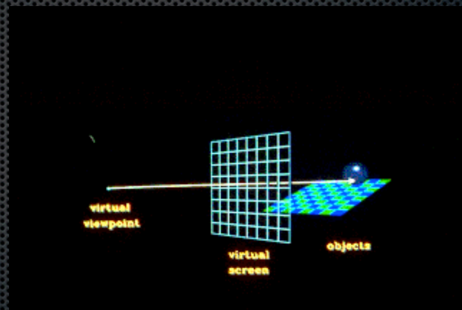
Trace the ray...

- Sometimes the ray misses all objects



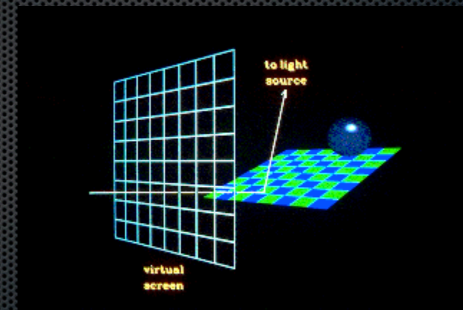
Trace the ray...

- Sometimes the ray misses hits an object



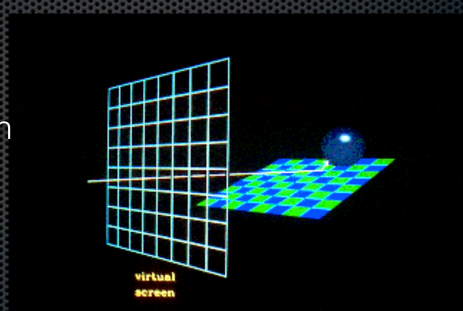
Trace the ray...

- If it hits an object, we want to know if the object is in shadow



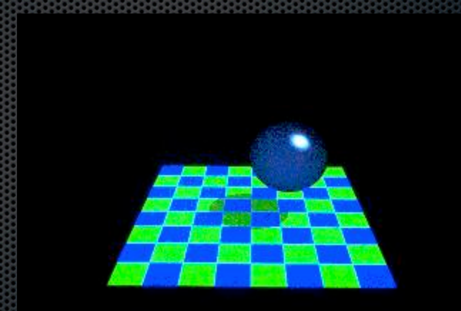
Trace the ray...

- If the shadow ray hits an object, then we are in shadow.



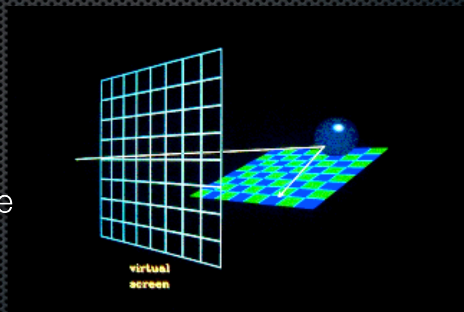
Trace the ray...

- And if its in shadow, ignore the effect of that light in colouring that spot.



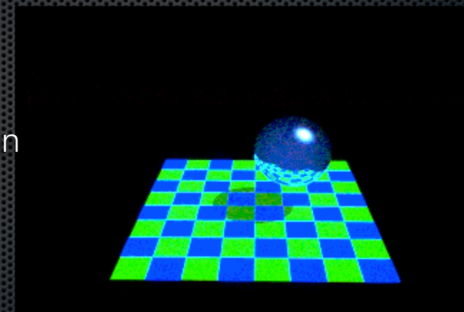
Trace the ray...

- When the ray hits an object, generate a reflected ray to determine what might be reflected in the surface



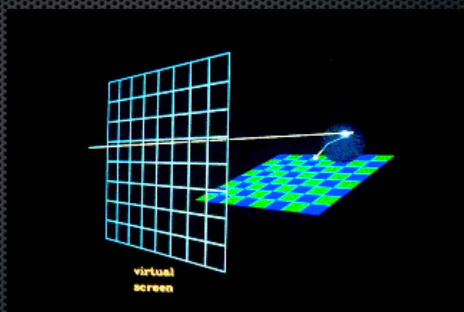
Trace the ray...

- This gets us reflections in reflective surfaces



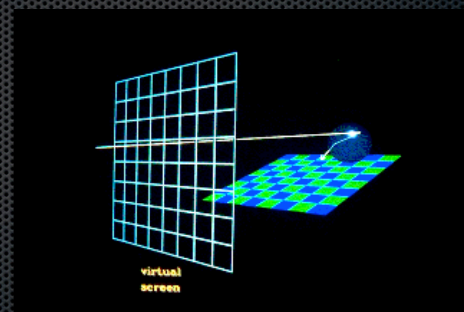
Trace the ray...

- If the object is transparent, then trace the ray through the object.



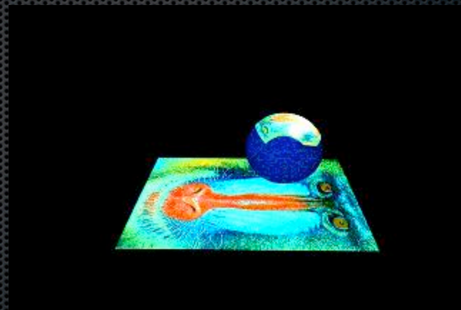
Trace the ray...

- If the object is transparent, then trace the ray through the object.



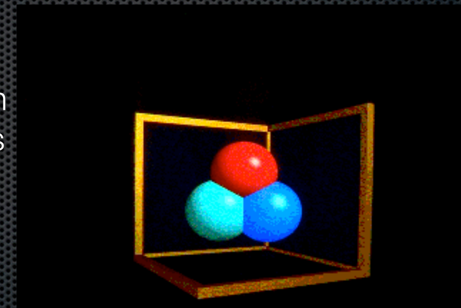
Trace the ray...

- So now we have simulated refraction



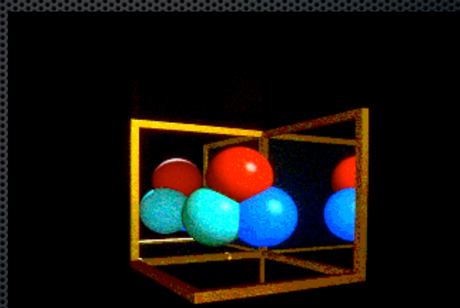
Trace the ray...

- But what happens when we have multiple mirrors (reflections within reflections).



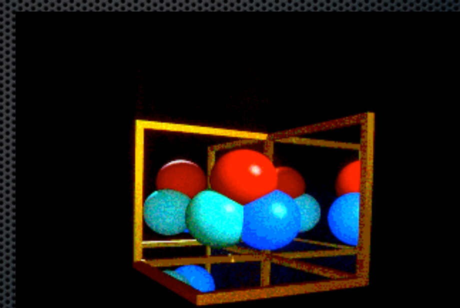
Trace the ray...

- But what happens when we have multiple mirrors (reflections within reflections).

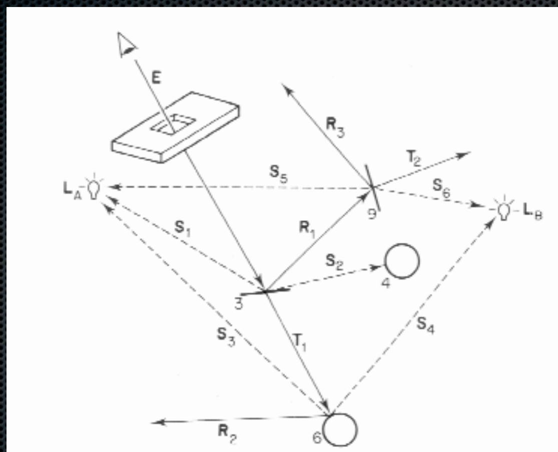


Trace the ray...

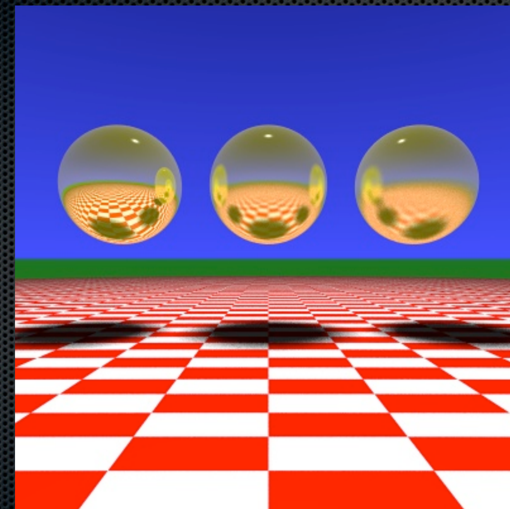
- But what happens when we have multiple mirrors (and reflections within reflections within reflections).



The math...

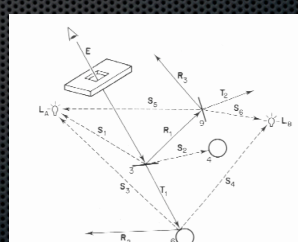


Surface properties



```
function Raytrace(E, D) returns Colour {
    nearest_t = infinity
    nearest_object = NULL
    for each object {
        find t, the smallest, non-negative real solution of the ray/object
                intersection equation
        if t exists {
            if t < nearest_t {
                nearest_t = t
                nearest_object = current object
            }
        }
    }
    colour = black
    if nearest_object exists {
        find normal vector, N, at intersection point
        if object is reflective {
            reflected_colour=Raytrace(intersection point, reflection vector)
            colour += reflection_coeff * reflected_colour ;
        }
        if object is refractive {
            refracted_colour=Raytrace(intersection point, refracted vector)
            colour += refraction_coeff * refracted_colour ;
        }
        for each light {
            if shadow_ray(intersection point, light position) returns No_Shadow {
                calculate light's colour contribution by doing the illumination calculations
                        using D, N, the current light, and the object properties
                colour += light's colour contribution
            }
        }
    }
    return colour
}
```

```
function shadow_ray(point1, point2) returns Shadow or No_Shadow {
    ray defined with E=point1, D=point2-point1
    nearest_t = infinity
    nearest_object = NULL
    for each object {
        find t, the smallest, non-negative real solution of the ray/object
                intersection equation
        if t exists {
            if t < nearest_t {
                nearest_t = t
            }
        }
    }
    if t <1 return Shadow
    else return No_Shadow
}
```



Ray tracing...

Base case?

- Ray hits nothing...
- Total amount of energy for this ray falls below some threshold
- Out of system resources

Recursive case

- Ray hits a surface
  - Reflected ray
  - Refracted ray