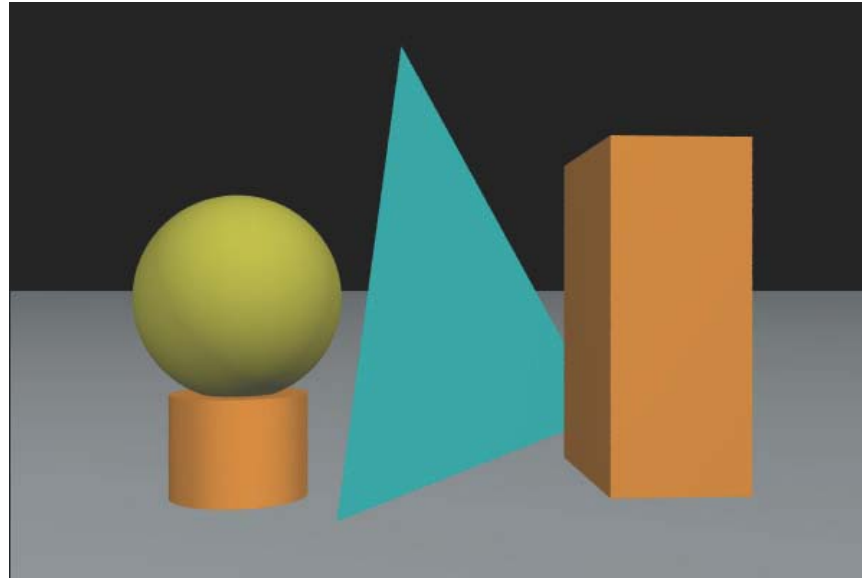




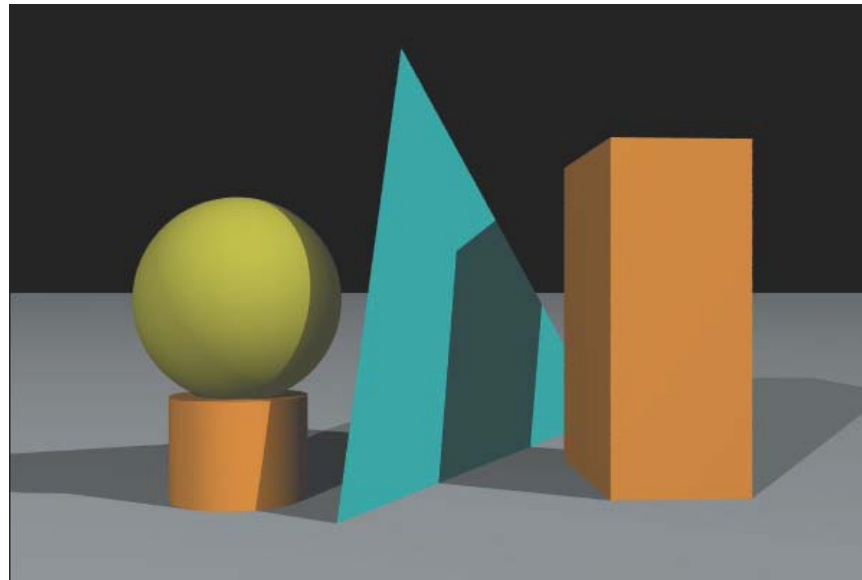
**CS 563 Advanced Topics in
Computer Graphics**
Shadows

by Sam Song

- Introduction
- Definitions
- Implementation
- Costs
- Results

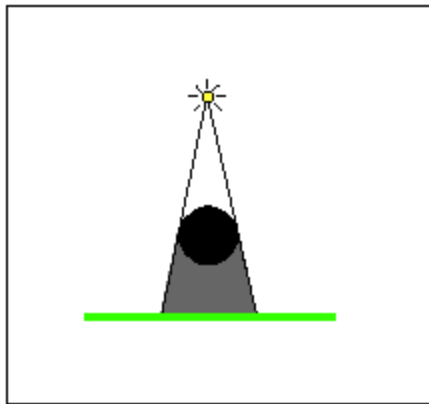


- Why do we need shadows?
 - Where are the objects relative to the plane?
 - How many light sources are there?
 - What type of light / direction
 - What are the relative sizes of the objects?

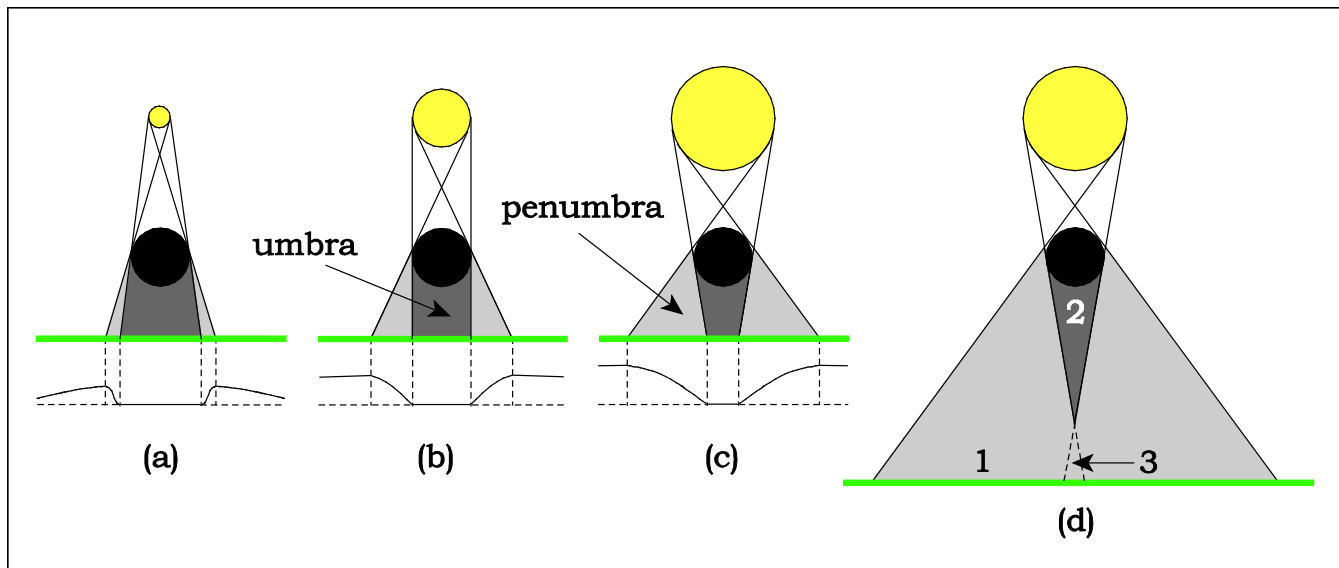


- Shadows give us important visual clues
 - Objects are on or penetrating plane
 - There are two light sources
 - Able to estimate relative sizes
- Shadows are everywhere and expected for realism

- Shadows
 - Part of the scene that is blocked from a light source by some object
 - 3 dimensional space – only rendered on object surfaces
- Point and direction lights
 - Hard edged shadows



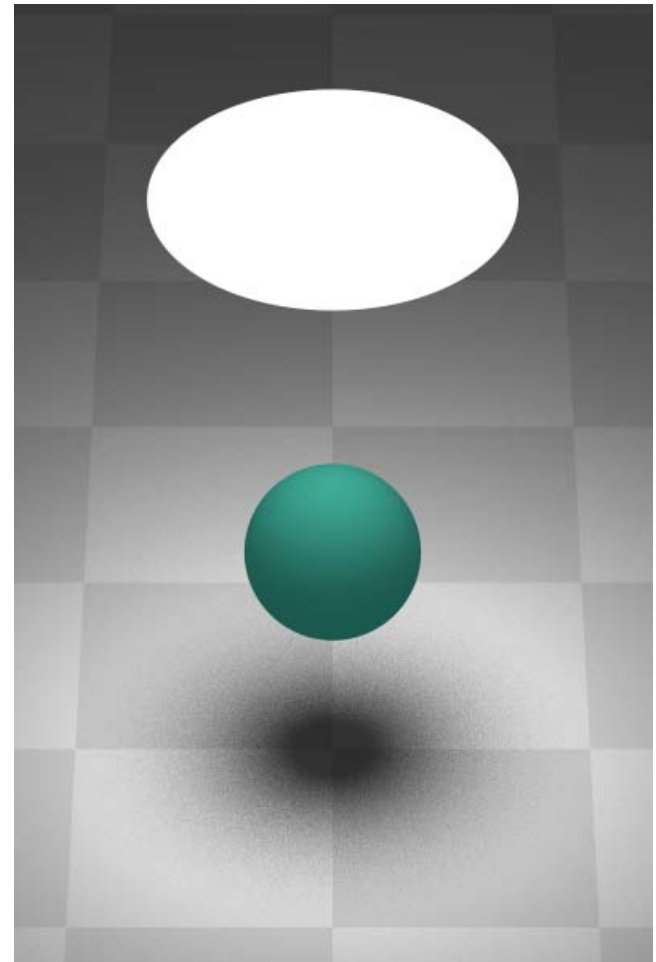
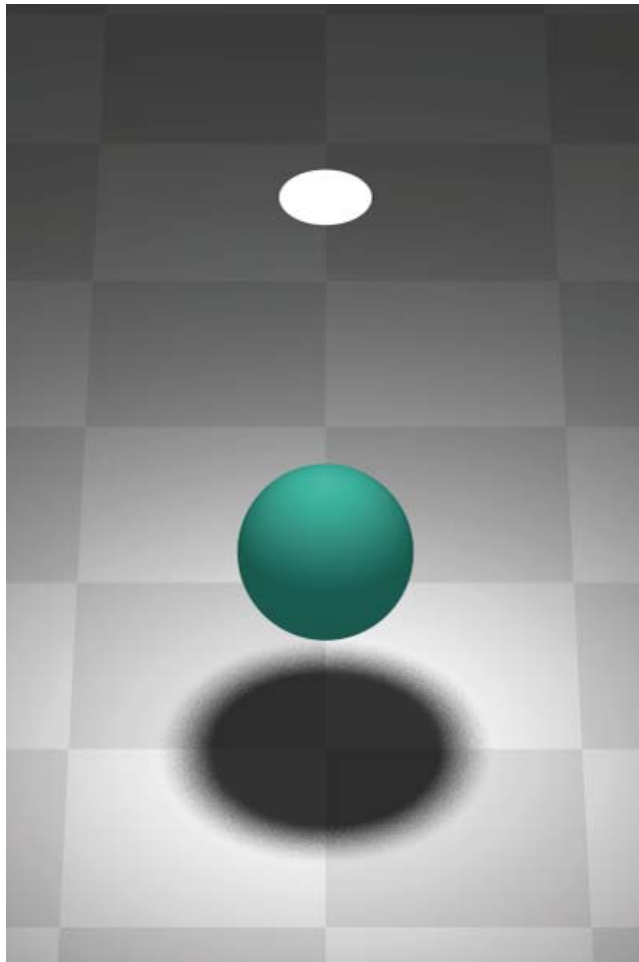
- Real lights have finite surfaces
 - Partially blocked by objects



- Umbra: no direct illumination from light source
- Penumbra: partial illumination from light source

Definitions

- Soft shadows
 - no sharp boundaries



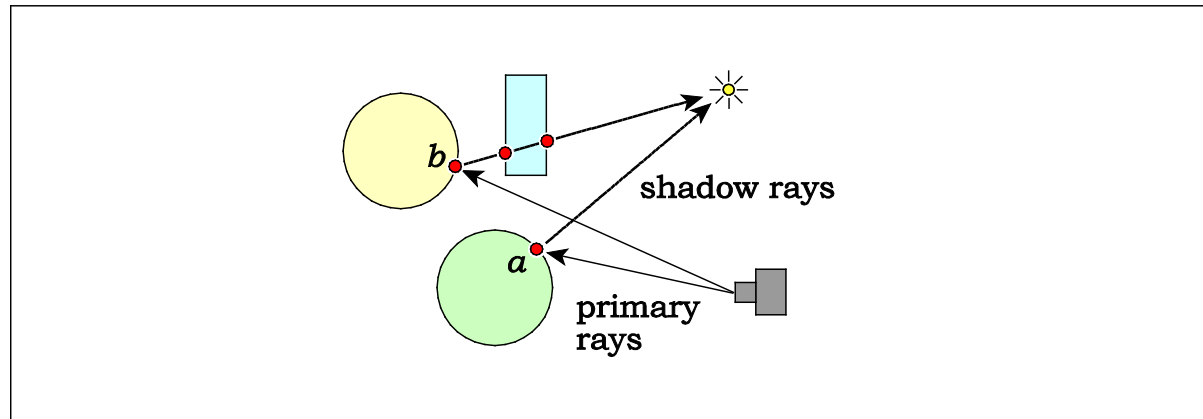
- How do we implement shadows for point lights?
- Equation for reflected radiance from the light

$$L_o(p, w_o) = f_r(p, l(p), w_o) l_s c_l V(p, l_p) \cos \Theta_l,$$

- We need to evaluate the visibility function
 - $V(p, l_p)$
 - p = hit point
 - l_p = light location
 - We need to know if a hit point is in shadow from a point light

- Shadow Ray

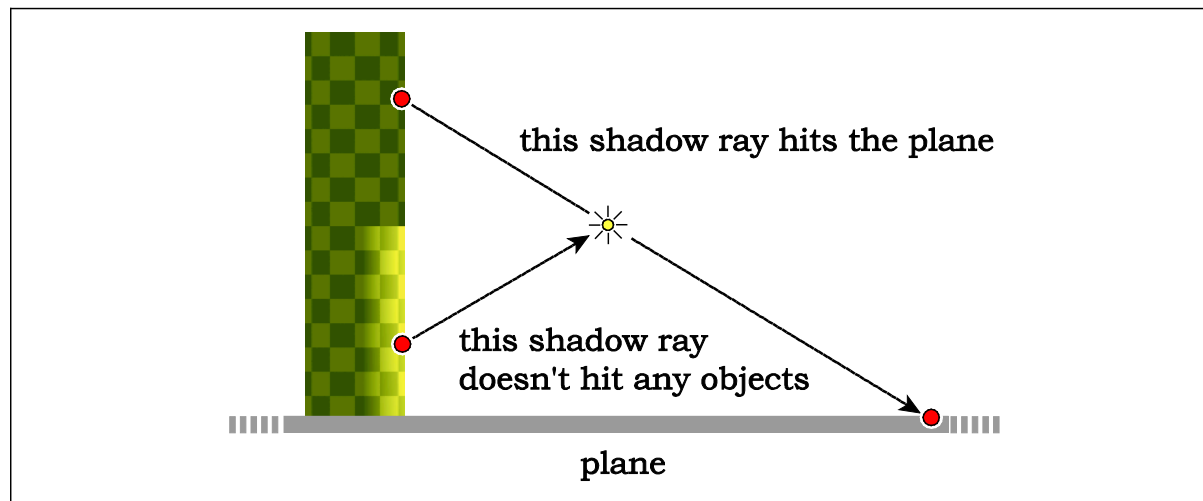
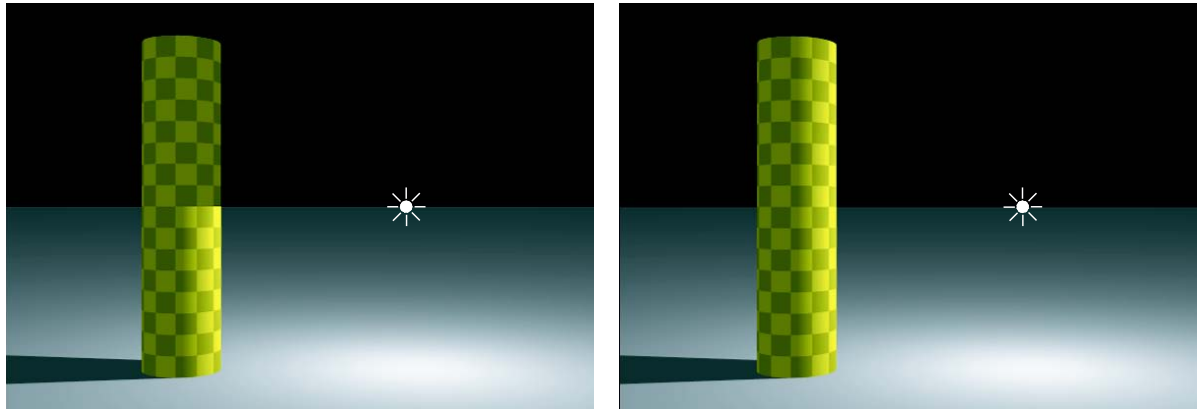
- Shoot a ray from the hit point to the point light and check if the ray hits anything in-between



- Hit point *a* shaded with ambient and direct illumination
- Hit point *b* shaded with only ambient illumination

Implementation

- Check if shadow rays intersect with objects to determine if a hit point is in shadow



- Origin of the Shadow Ray is the hit point p of the primary ray
- Direction of the Shadow Ray is towards the light point l_p
 - $d = (l_p - p) / ||p_l - p ||$
- Shadow Ray intersection is only valid if the hit point is between origin and light
 - $t < \text{distance}$

- Epsilon

- Constant $\epsilon > 0$ in hit functions

- $\epsilon < t$

```
bool
```

```
Plane::shadow_hit(const Ray& ray, float& tmin) const {
```

```
    float t = (a - ray.o) * n / (ray.d * n);
```

```
    if (t > kEpsilon) {
```

```
        tmin = t;
```

```
        return (true);
```

```
    }
```

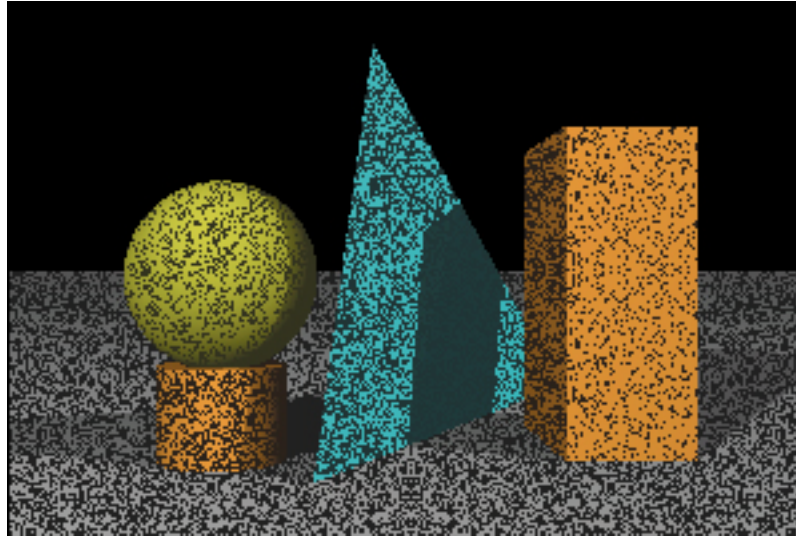
```
    else
```

```
        return (false);
```

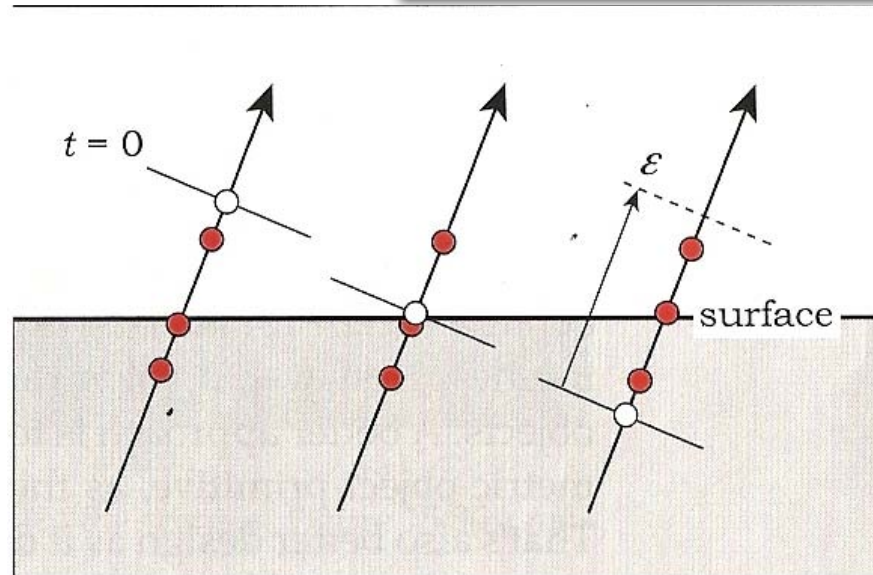
```
}
```

- Why do we need Epsilon?

Implementation



- $\varepsilon = 0$
- Salt-and-pepper noise
- Random self shadowing

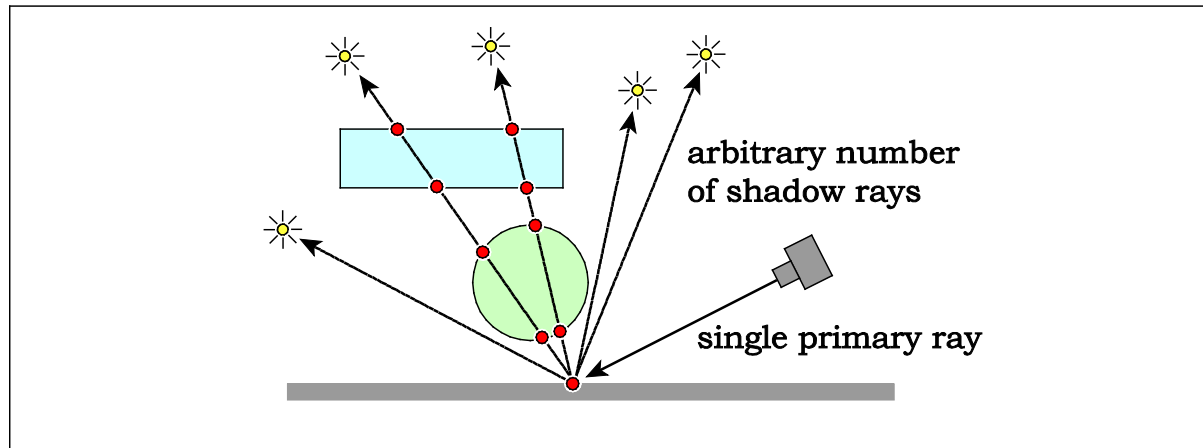


- Hit points can be above or below the surface of the object
 - Finite numerical precision
 - Origin of shadow ray may be inside or outside surface
 - $t=0$ for the hit function of a shadow ray may not return the intersection with the surface

Implementation

- The value of t can randomly be small positive or negative values even for a plane surface
- Positive t value cause the shadow ray to return true and self shadow
- To stop self shadowing
 - ϵ larger than the largest possible t value that could return true
 - Constant ϵ value for each geometric object
 - plane uses $\epsilon = 0.00001$
 - Global constant could create artifacts
 - Does consider object sizes
 - Reflected & transmitted rays originate on objects surfaces



- Why are shadows expensive to render?



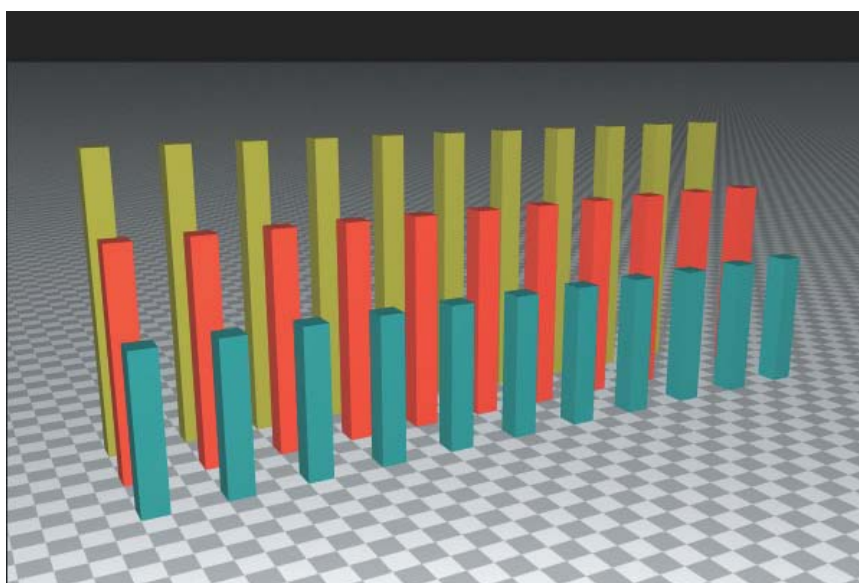
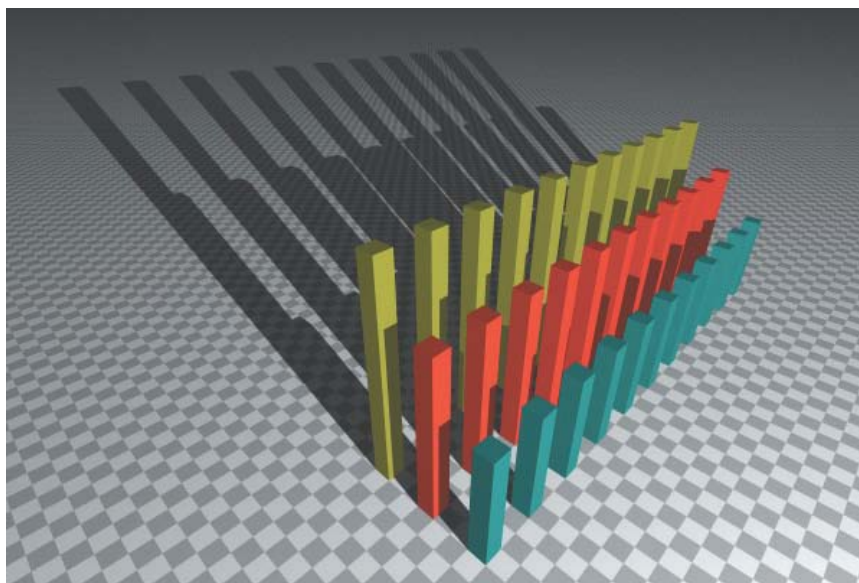
- Each hit point requires a shadow ray to each point light
- Each shadow ray needs to check intersection with objects in the scene

- How can we cheapen the cost?
- Allow shadowing options
 - Lights & Objects optionally cast shadows
 - Materials optionally have shadows cast on them
 - If shading inside transparent objects
 - Not physically correct but allows for flexibility to get an image to look right
 - Allows for quicker rendering to test new features

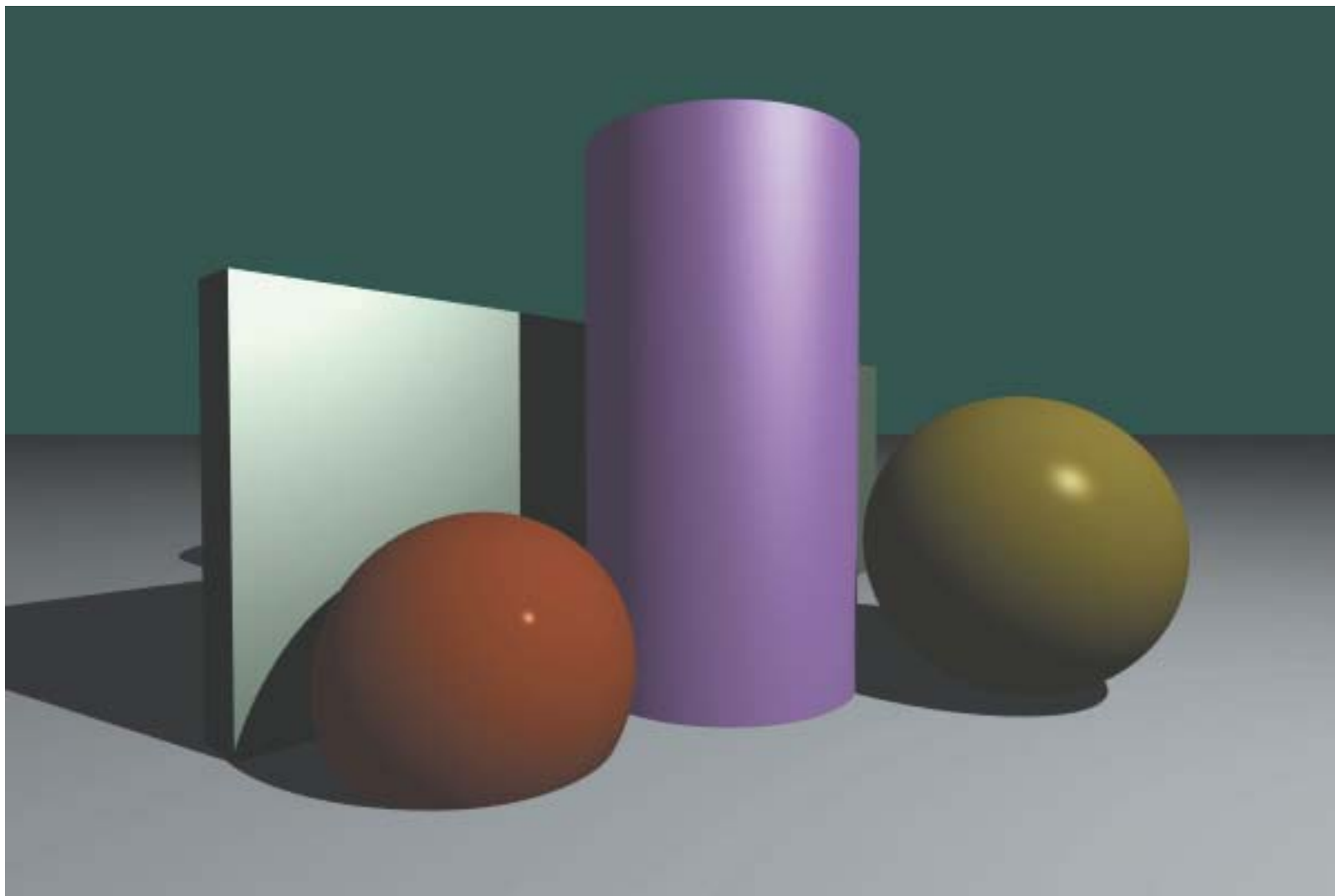
- Shadow rays can potentially check all of the objects in the scene, but we can stop if we find a hit point
- Create a shadow_hit function to replace the standard hit function
 - Standard hit returns an unneeded normal
 - Helps for more complicated objects
 - Not much work, shadow hit function requires removing shading code from the standard hit function
 - Increases code size

- 
- 
- Only create shadow rays if the surface hit point of the primary ray is facing towards the light point $(n \cdot w_0) > 0$
 - Caching
 - Remember which object a shadow ray intersects
 - Use that object to test first
 - May intersect again because shadow rays point to the same location

Results



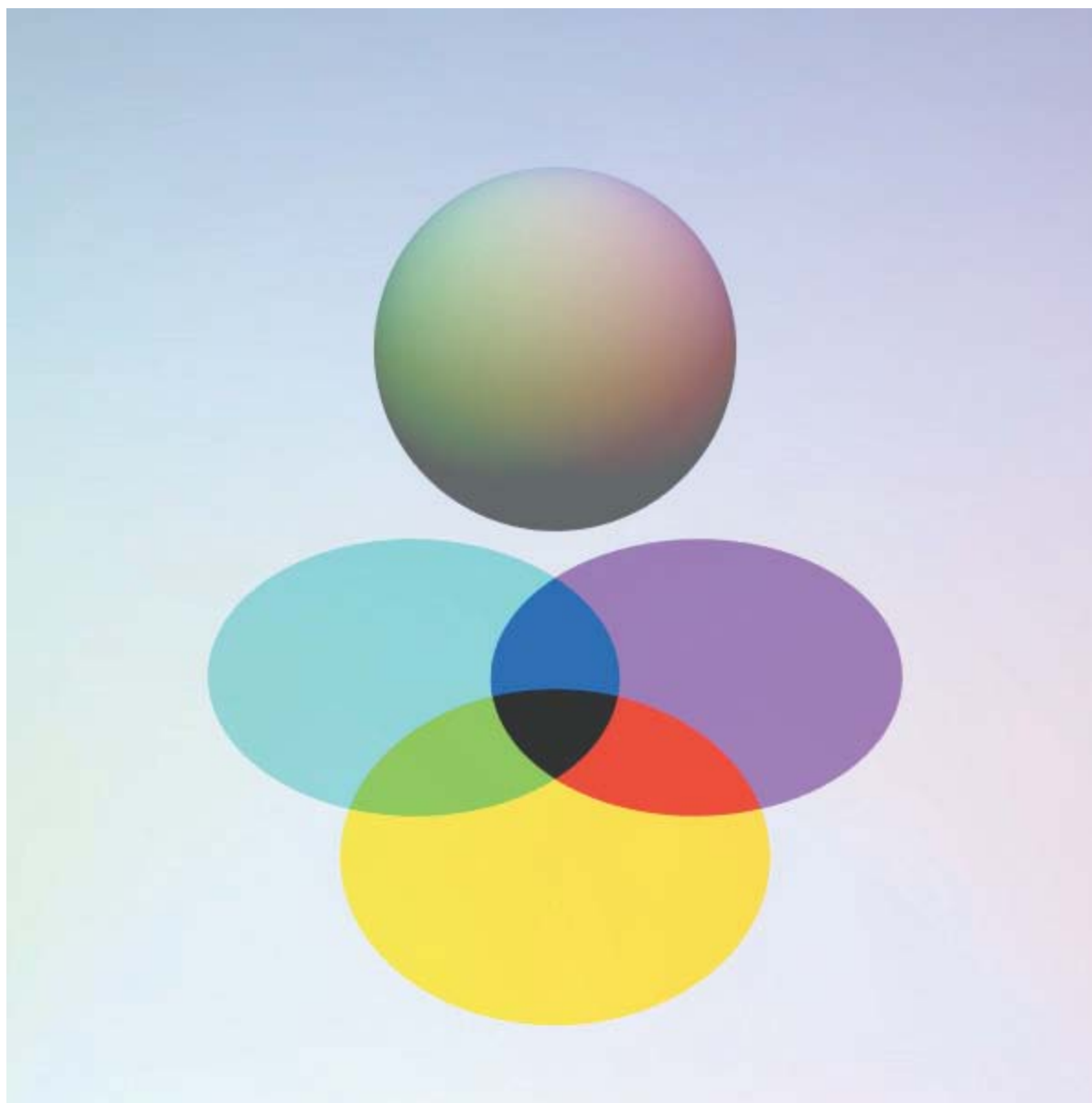
Results



Results



Results



- Questions?

- <http://www.raytracegroundup.com>
- Suffern, Kevin (2007). Ray Tracing from the Ground up. Pp. 197-216 Wellesley, MA: A K Peters, Ltd.
- <http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtracewr.htm>
- <http://www.groovyvis.com/other/raytracing/shadows.html>