Fundamentals of Computer Graphics

Lecture 9. Ray tracing

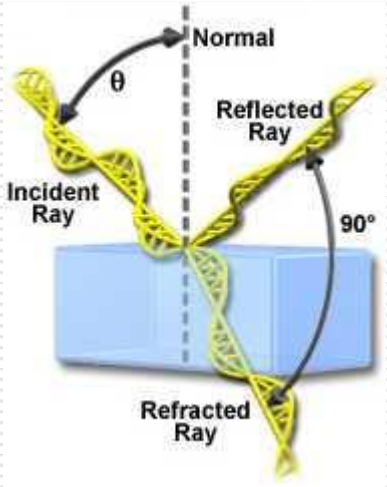
Yong-Jin Liu
liuyongjin@tsinghua.edu.cn

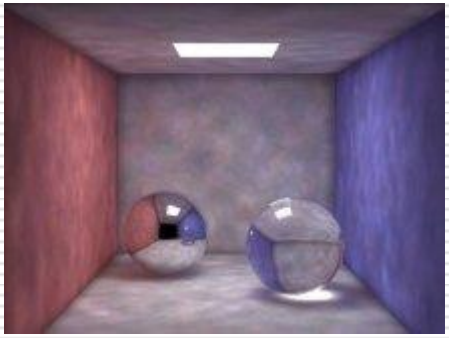
Outline

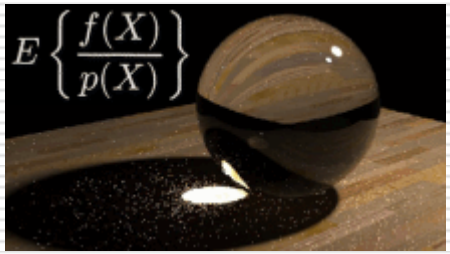
Next week --- Public holiday

- 4th May (next Monday)
- There will be no class

Trade-off: speed vs. accuracy







Simple lighting
heuristic rules

Global
illumination

Physics
Optics

Computer
Science

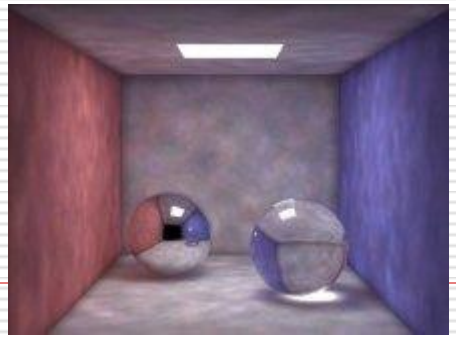
Realistic(Accurate)
Very time consuming

Fast
Real time

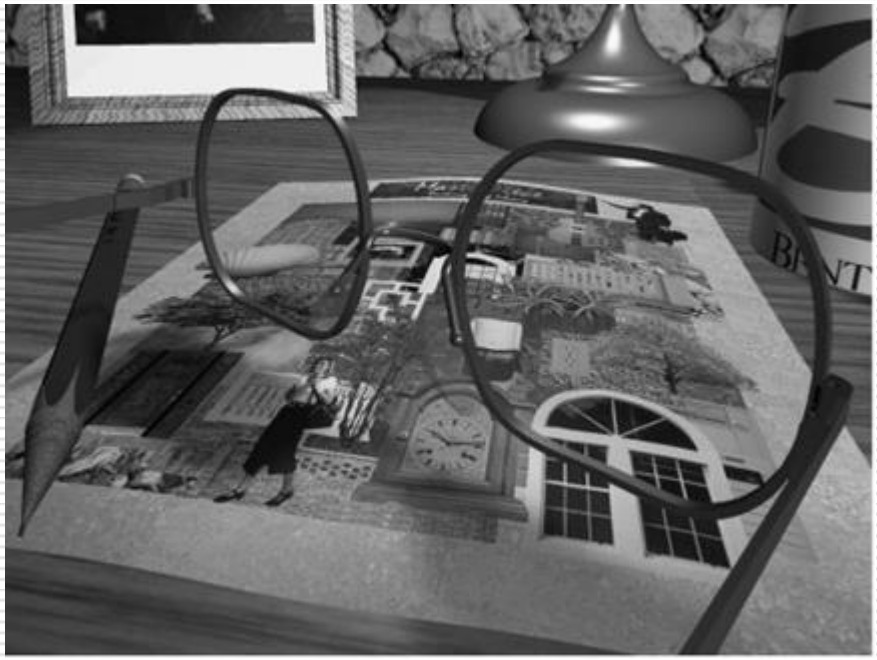
Outline

Ray tracing

- Mathematics and algorithms
- Create highly realistic images, which include transparency and refraction of light



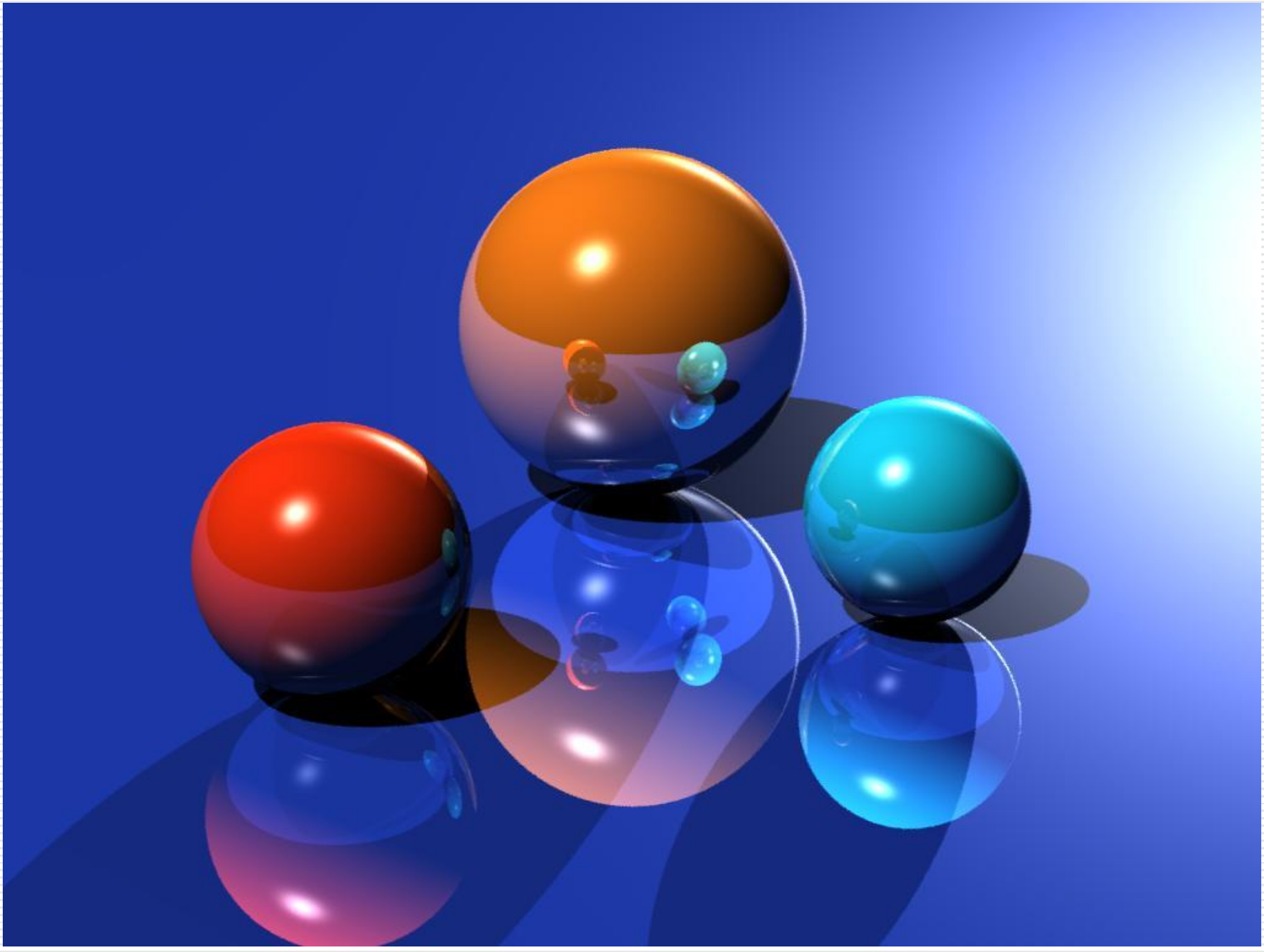
Local vs Global Illuminations



OpenGL model

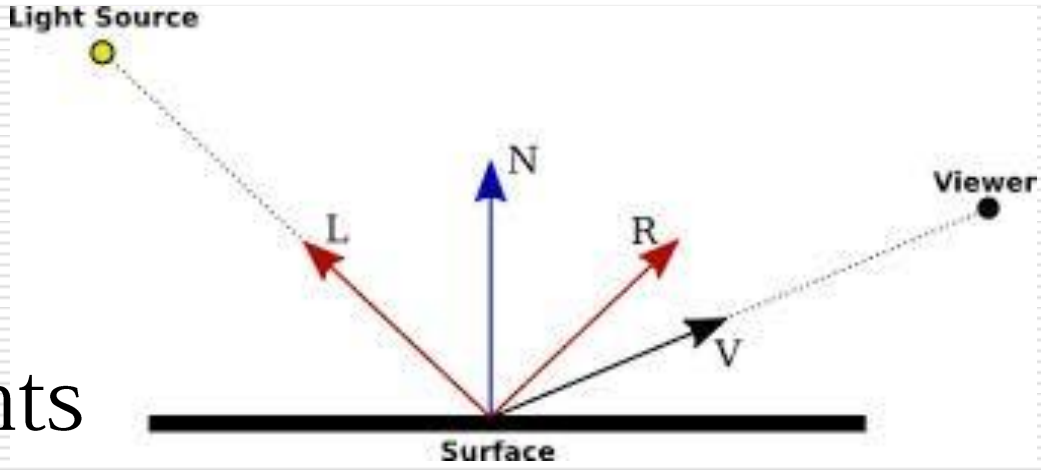
Ray tracing

Specular, reflective and transparent components



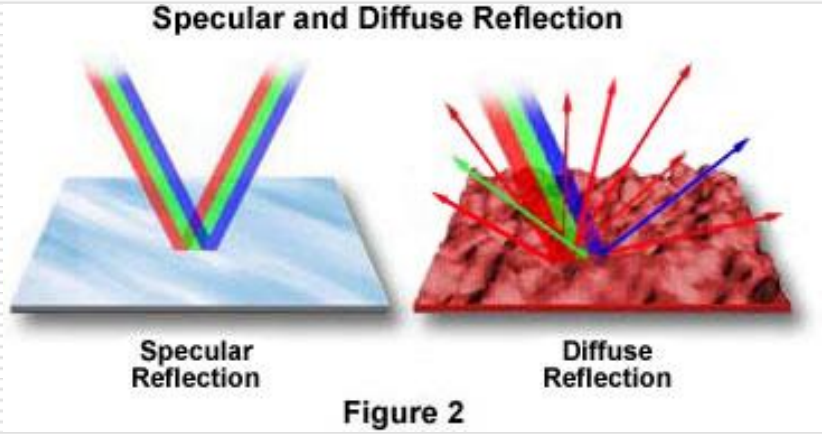
OpenGL light model

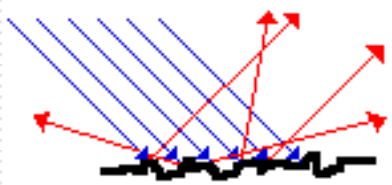
- Light source
  - Point light
  - Direction light
- Three components
  - Diffuse light
  - Specular light
  - Ambient

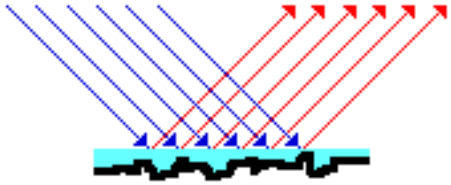


OpenGL light model

- Three components
  - Specular light
    The intensity along the reflected light direction decided by the law of reflection is maximum

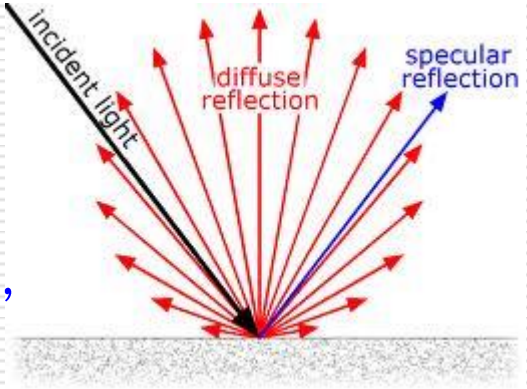



A dry asphalt roadway diffuses incident light.


When wet, water fills in the crevices, resulting in specular reflection and a glare.

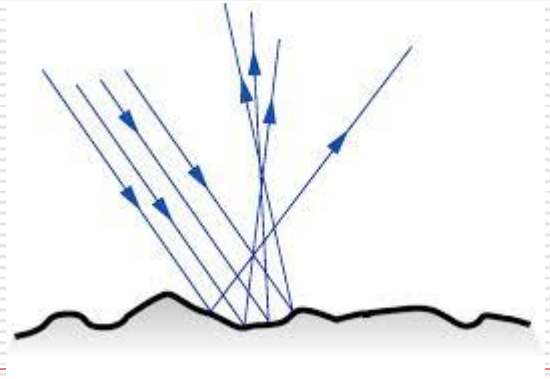
OpenGL light model

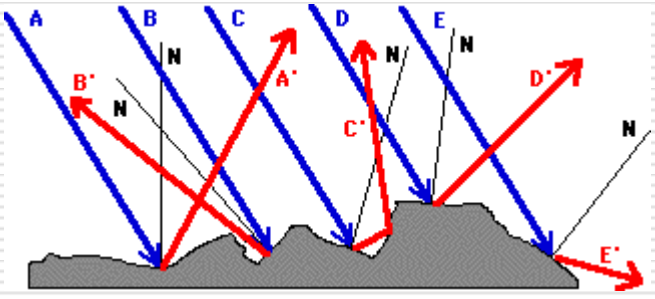
- Three components
  - Diffuse light



Diffuse light evenly spreads in all directions, has nothing to do with view point

  - Specular light

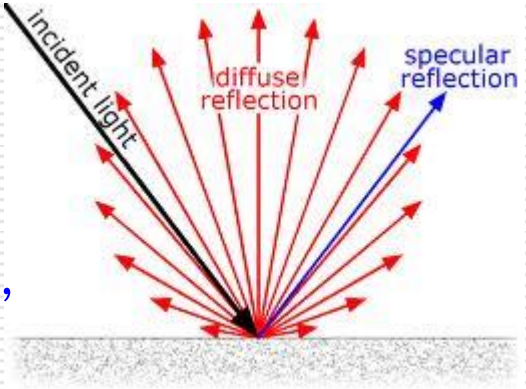




OpenGL light model

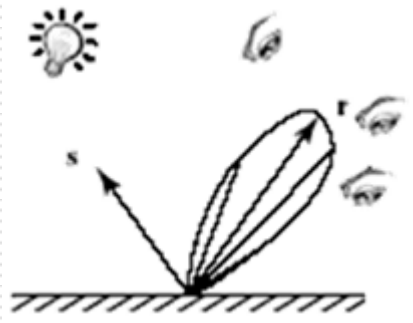
- Three components
  - Diffuse light

Diffuse light evenly spreads in all directions, has nothing to do with view point



  - Specular light

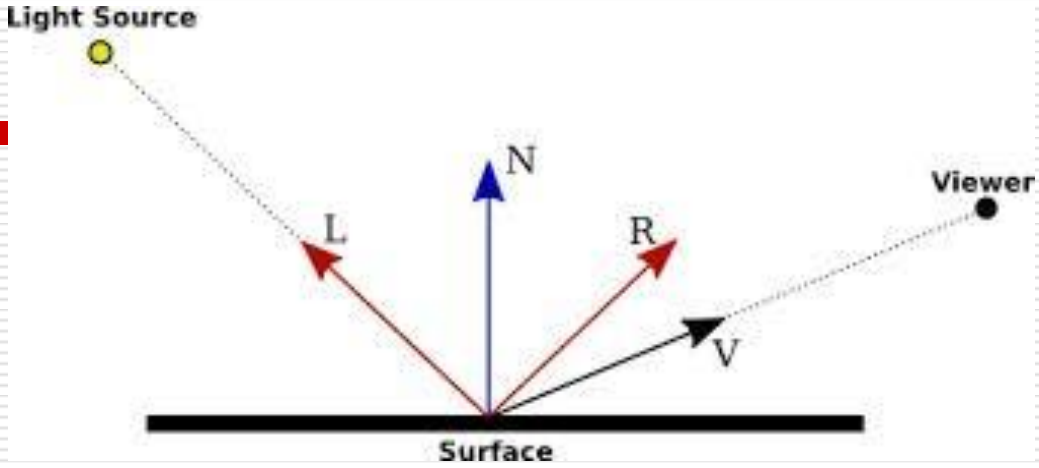
    The intensity along the reflected light direction decided by the law of reflection is maximum



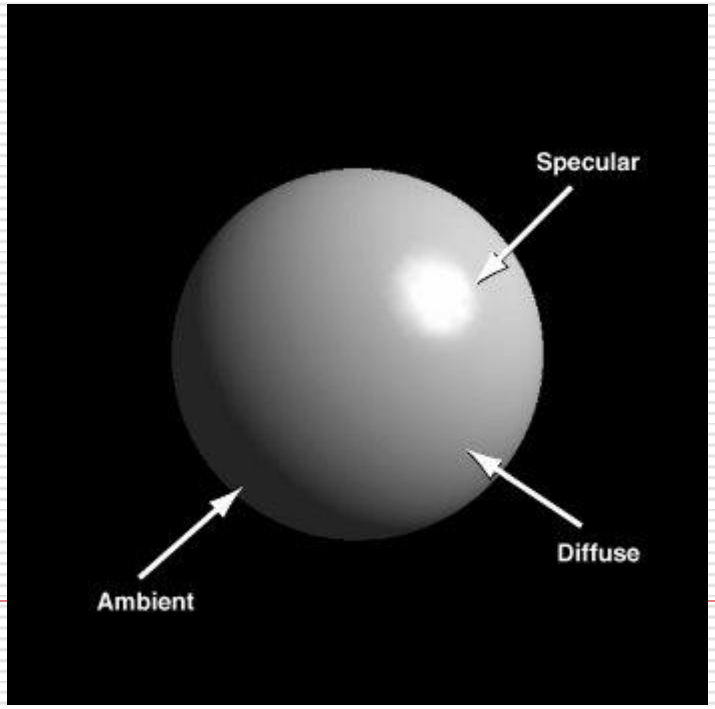
  - Ambient light

    Approximate global light effect roughly

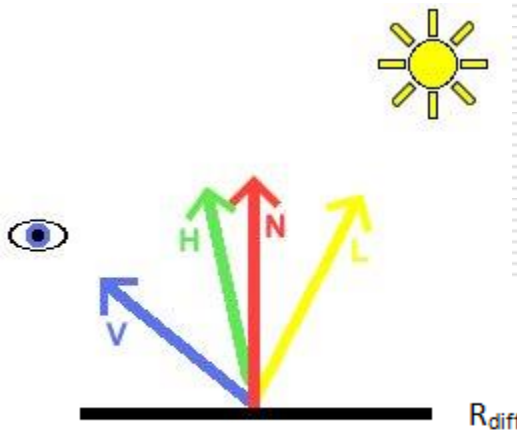
OpenGL light model

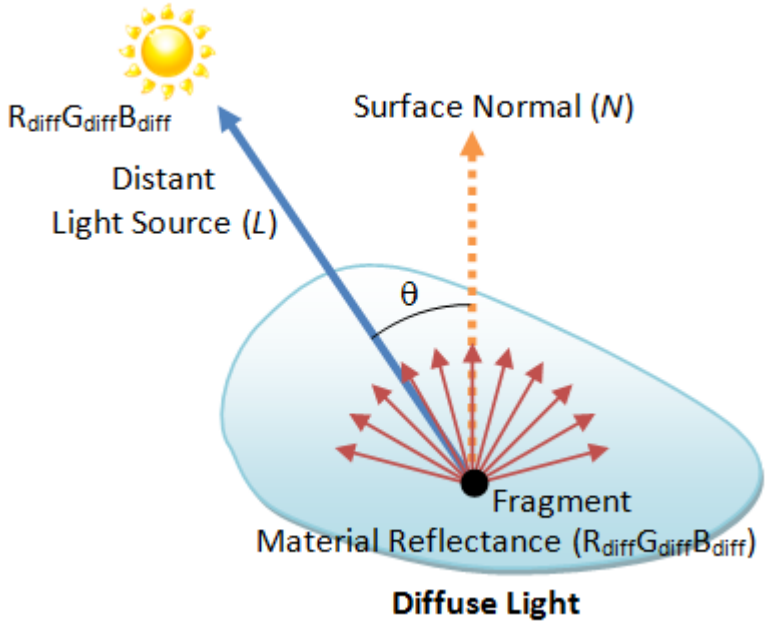


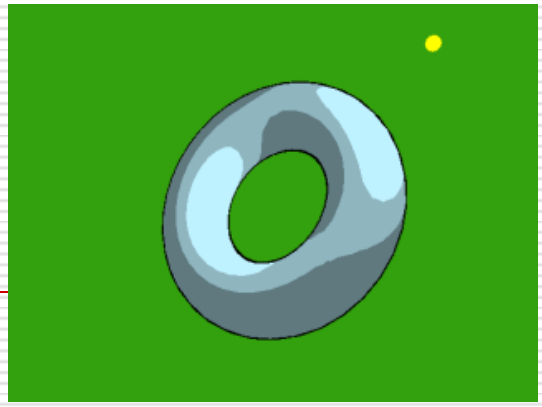
- Light source
  - Point light
  - Direction light
- Three components
  - Diffuse light
  - Specular light
  - Ambient light

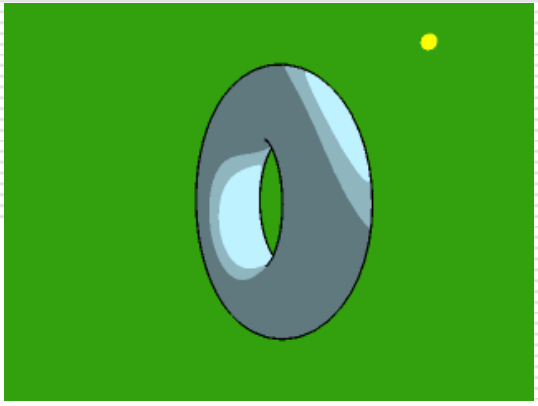


OpenGL light model

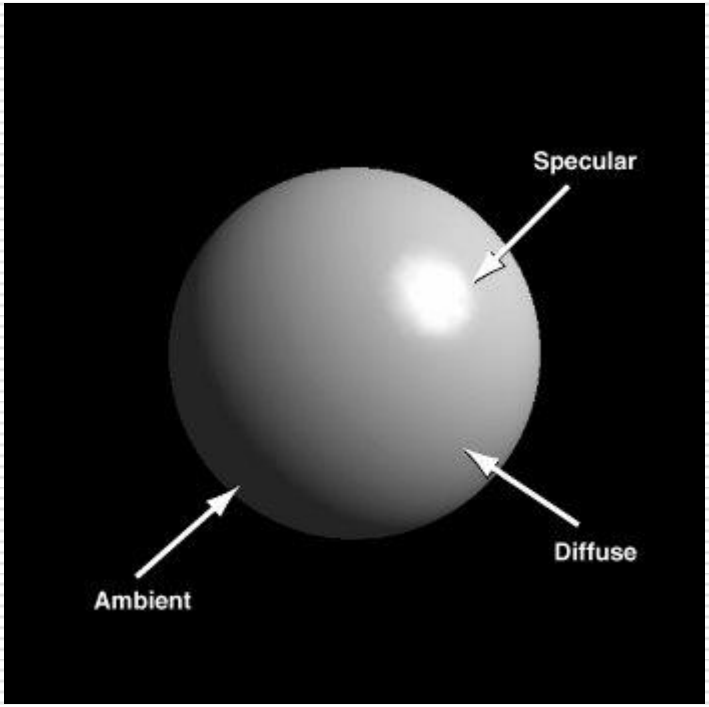


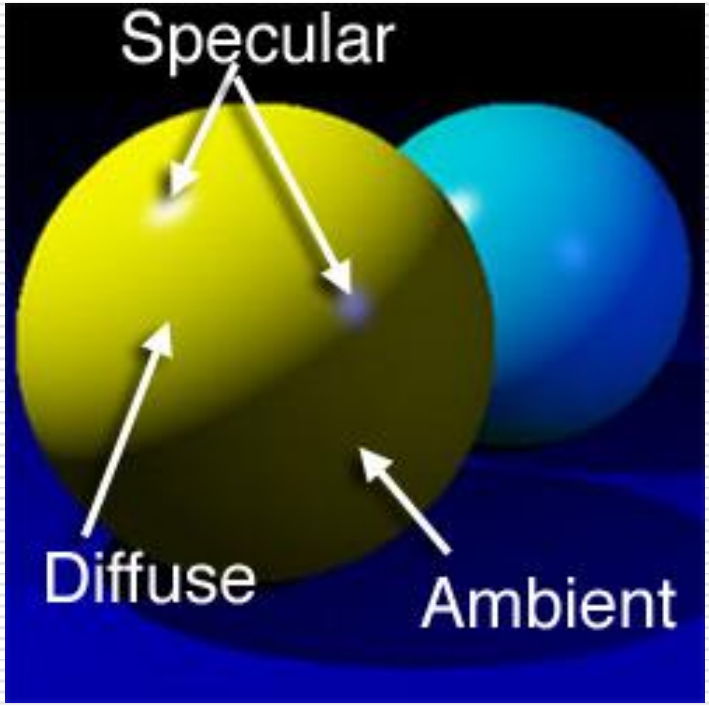


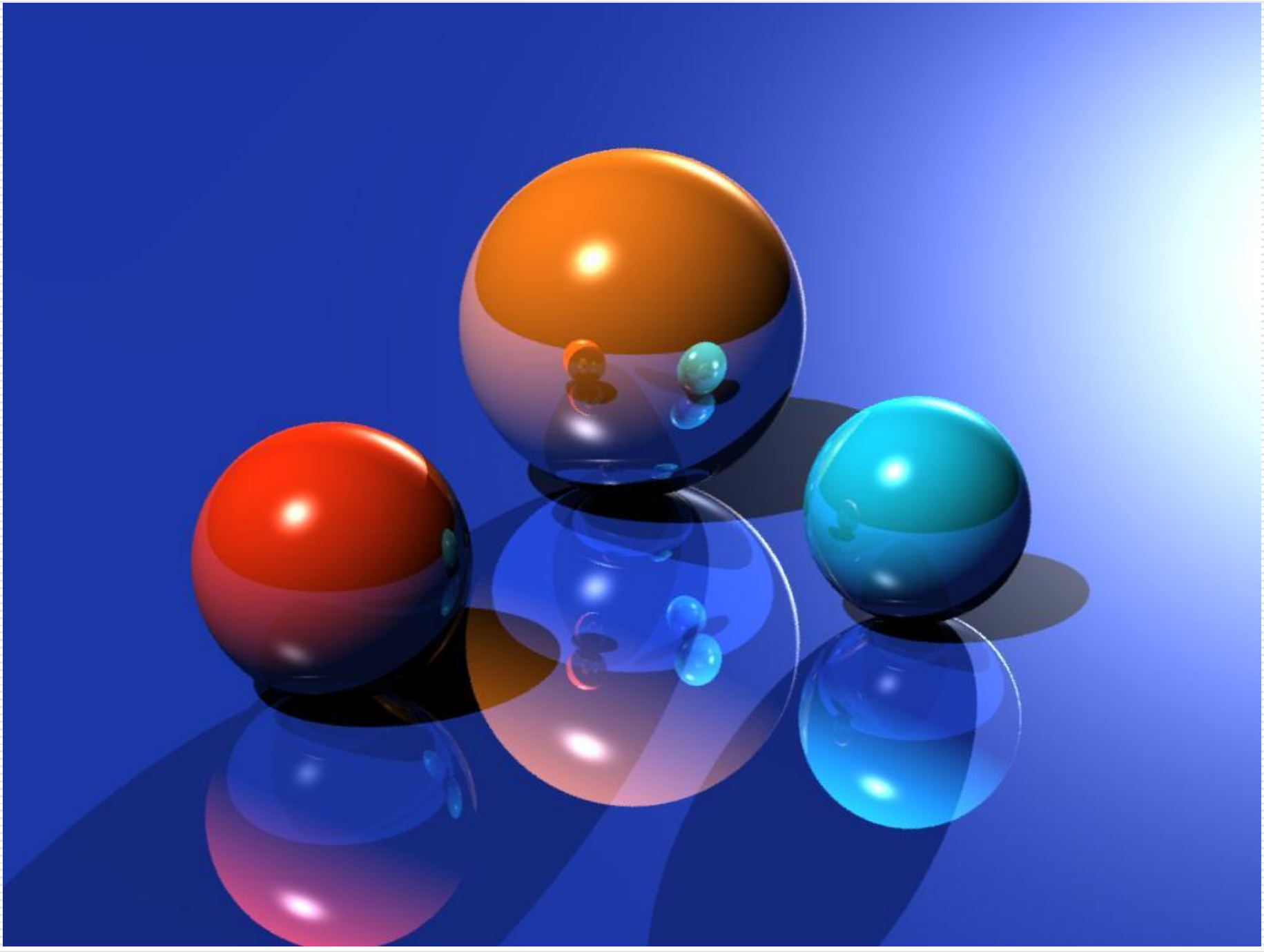




OpenGL light model







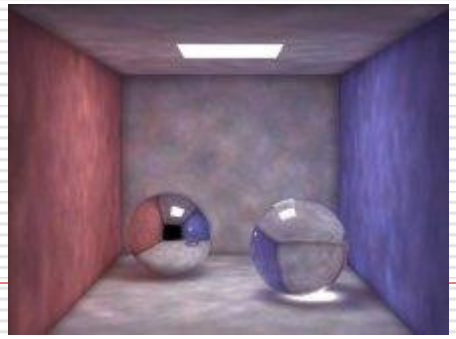
Scenes that OpenGL cannot render

Scenes that OpenGL cannot render

Outline

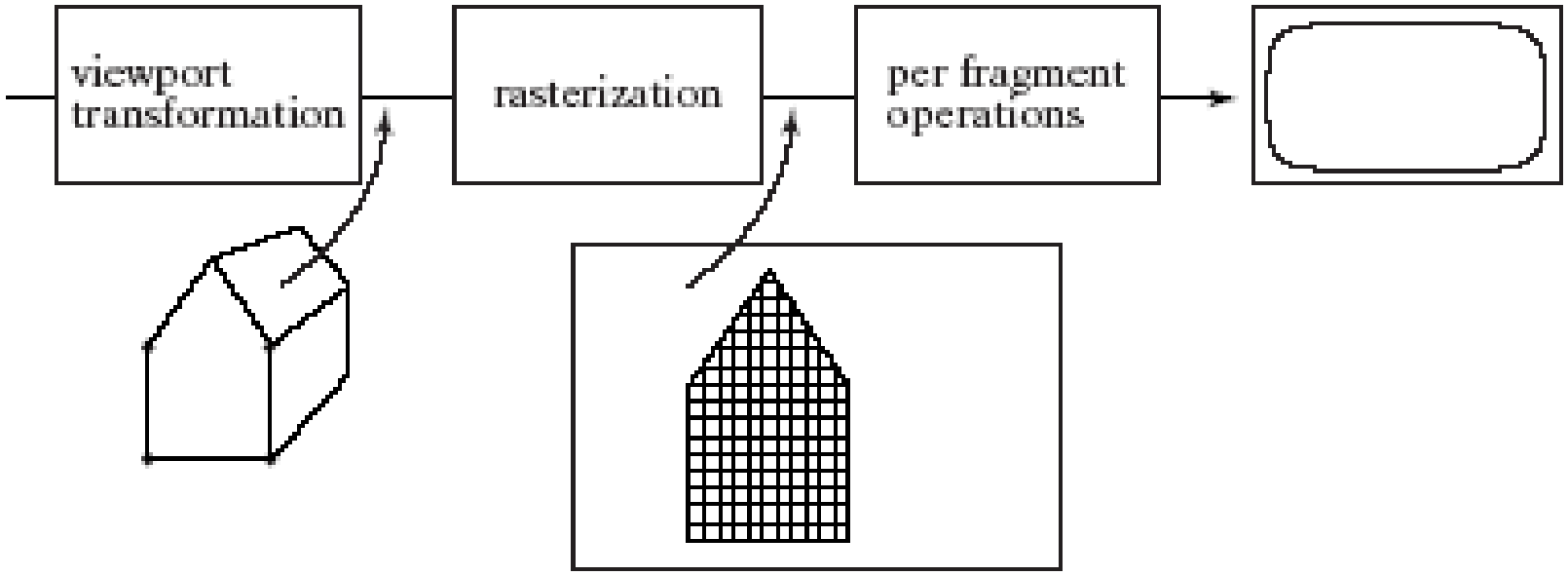
Ray tracing

- Mathematics and algorithms
- Create highly realistic images, which include transparency and refraction of light



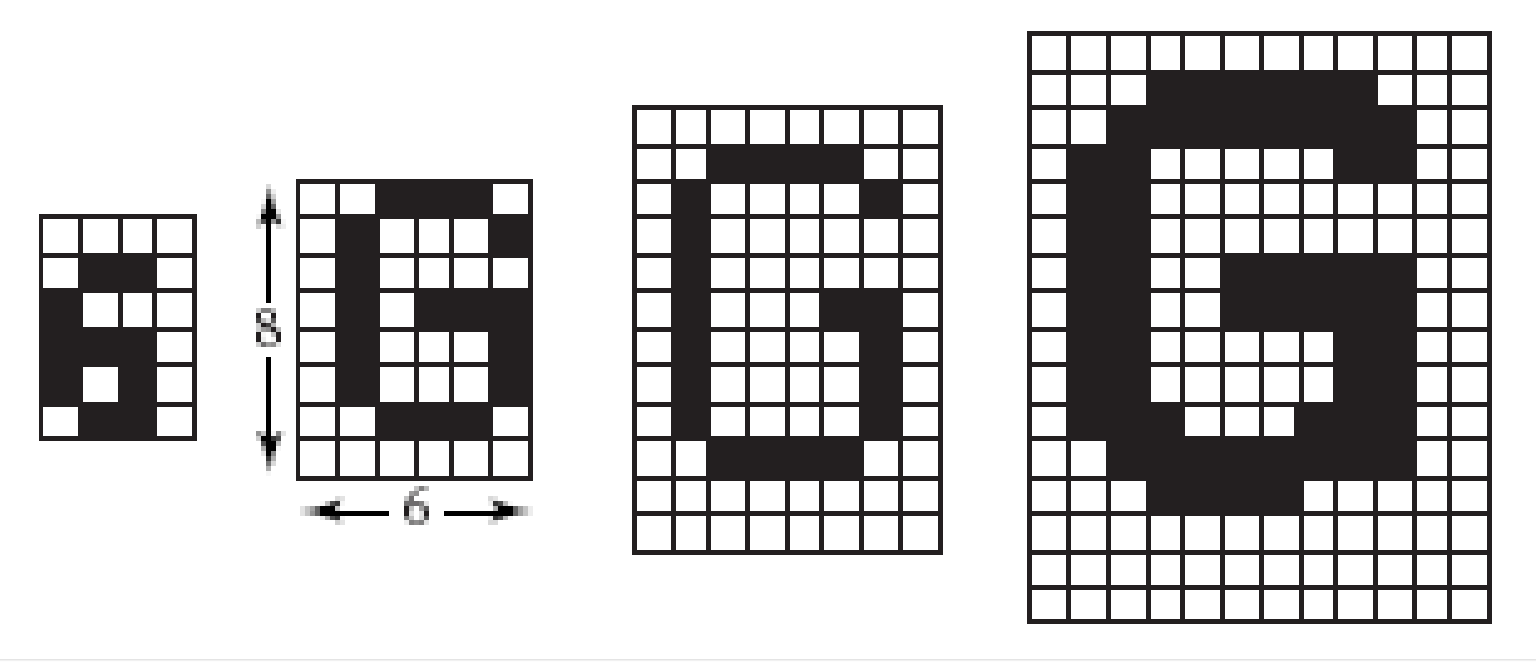
Rasterization

- Determines individual pixel values
- Rasterization in the GL graphics pipeline



Rasterization

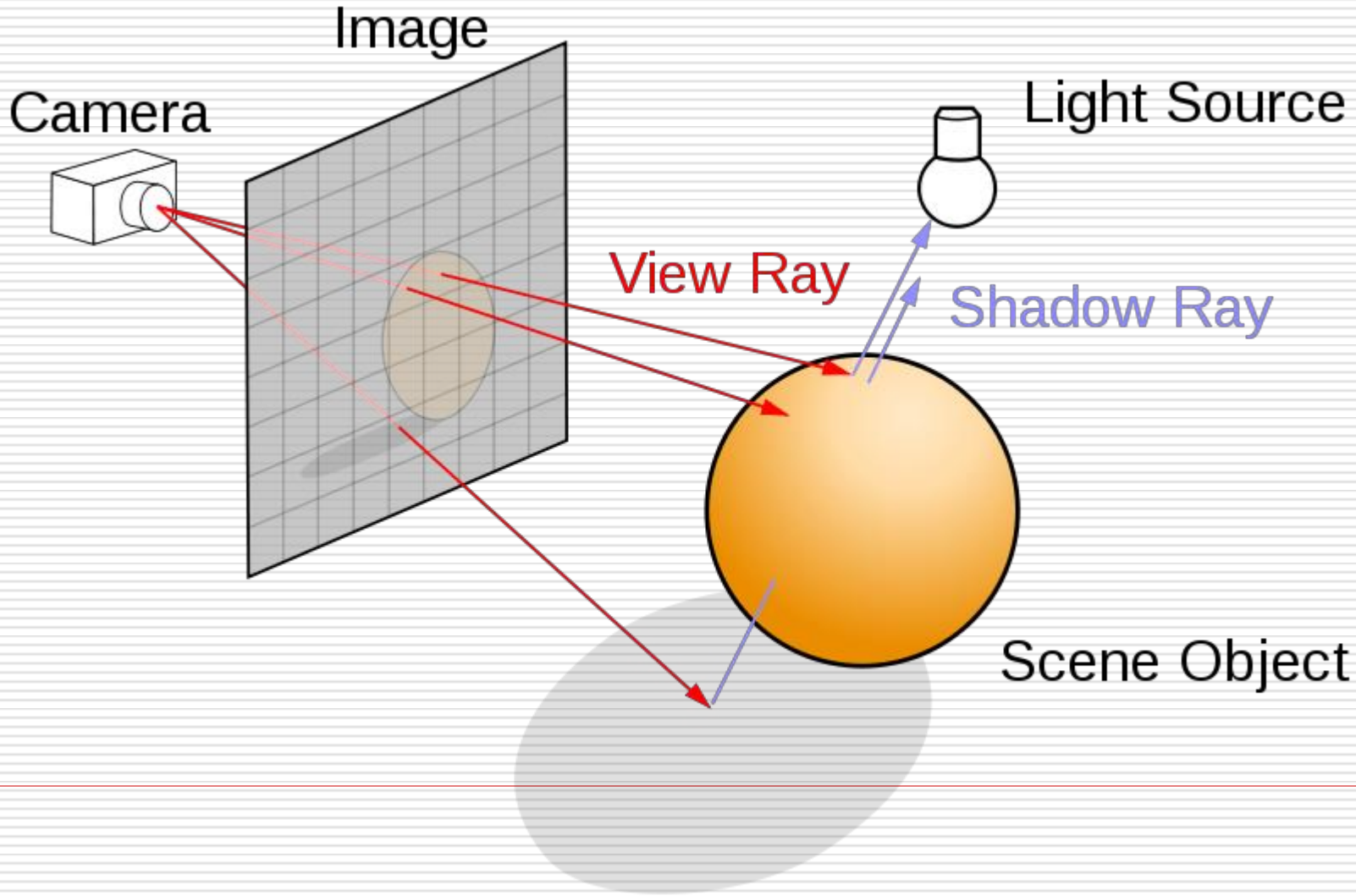
- Determine individual pixel values



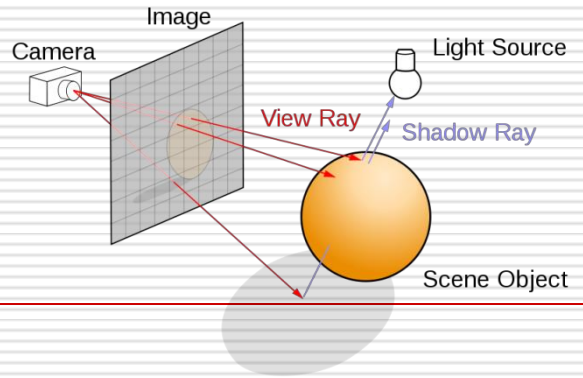
Ray Tracing

- We know how to render scenes composed of polygonal meshes, including shading models that represent—at least approximately—how light reflects from the surface of a polygon
- Ray tracing thinks of the frame buffer as an array of pixels positioned in space with the eye looking through it into the scene
- For each pixel in the frame buffer we ask: What does the eye see through this pixel?

Ray Tracing



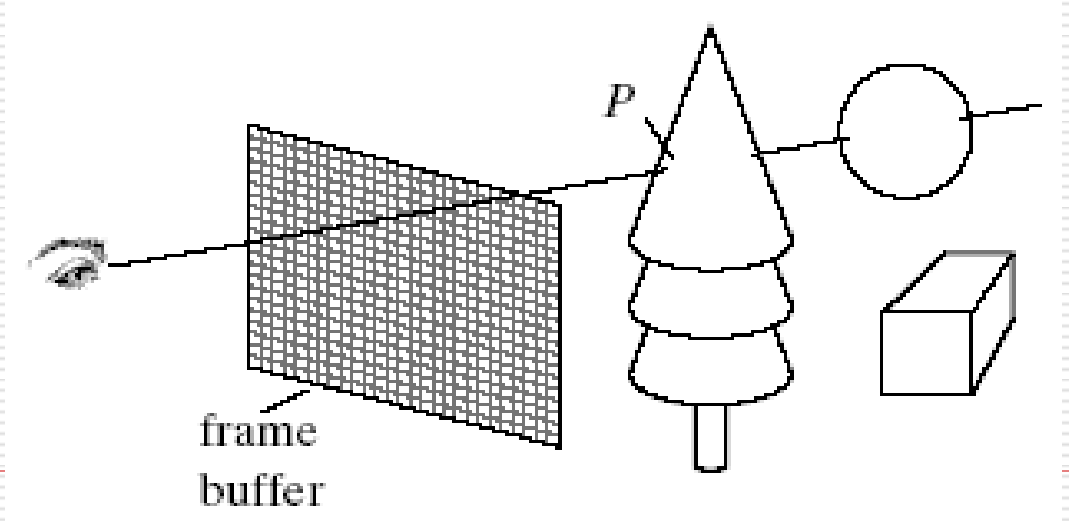
Ray Tracing



- We know how to render scenes composed of polygonal meshes, including shading models that represent—at least approximately—how light reflects from the surface of a polygon
- Ray tracing thinks of the frame buffer as an array of pixels positioned in space with the eye looking through it into the scene
- For each pixel in the frame buffer we ask: What does the eye see through this pixel?

Ray Tracing

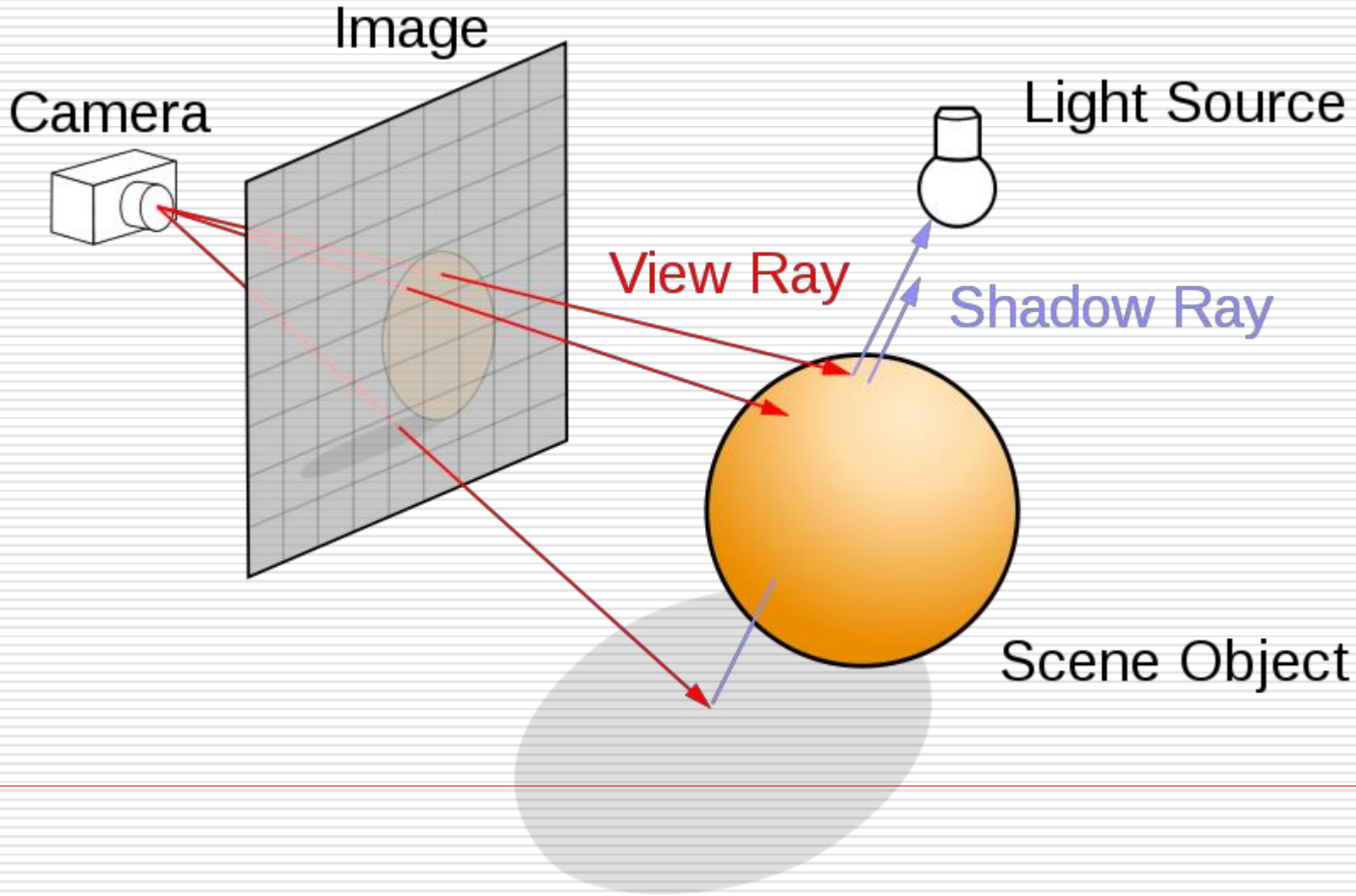
- We think of a ray of light arriving at the eye (or more importantly on the viewport) through this pixel from some point P in the scene. The color of the pixel is set to that of the light that emanates along the ray from point P in the scene



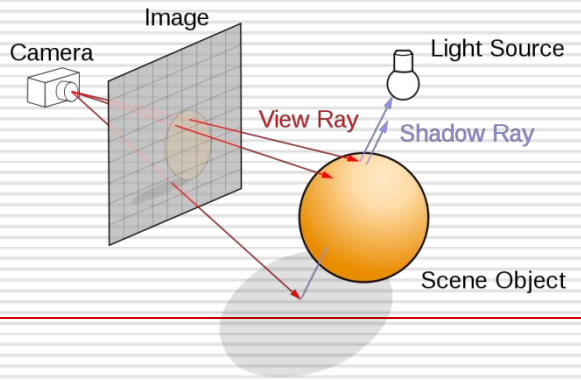
Ray Tracing --- tracing in an inverse way

- In practice, ray tracing follows a ray from the eye through the pixel center and out into the scene.
- Its path is tested against each object in the scene to see which object (if any) it hits first and at what point
- A parameter t is used to associate the ray's position with our abstract notion of time so that we can say "the ray is here now" or "the ray reaches this point first."

Ray Tracing

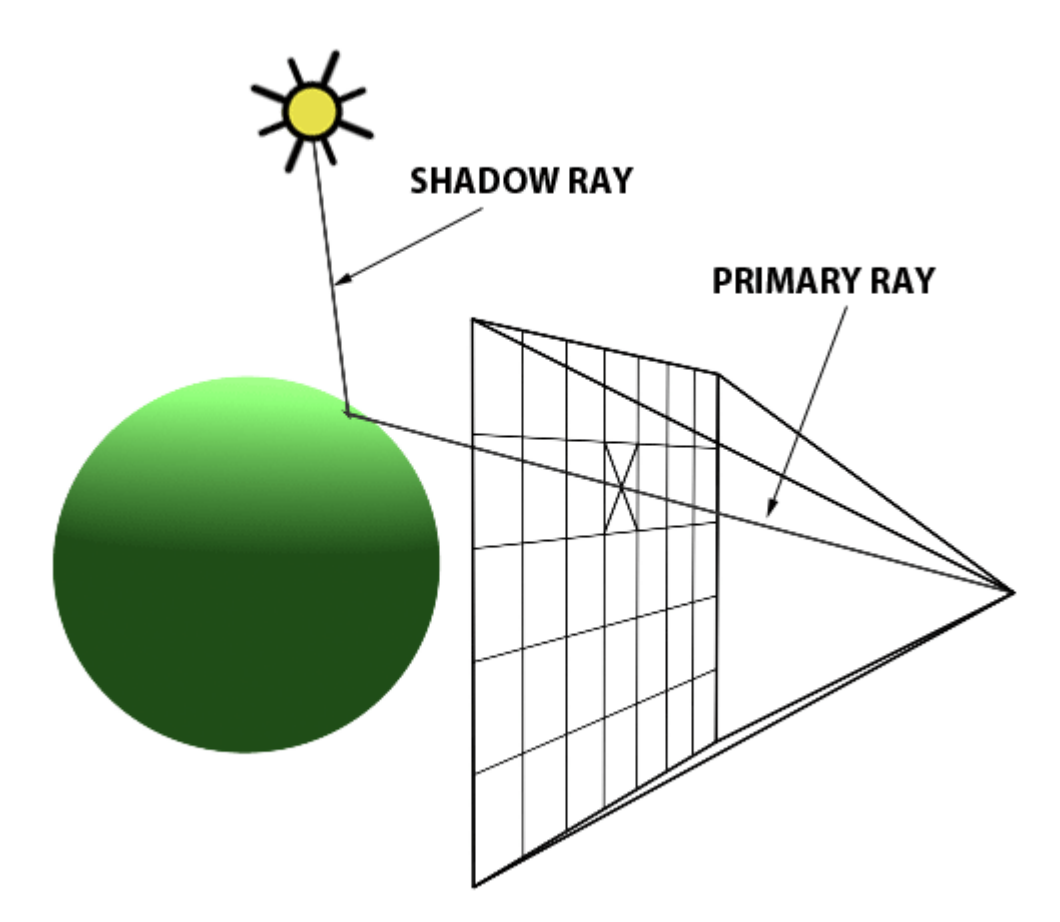


Ray Tracing

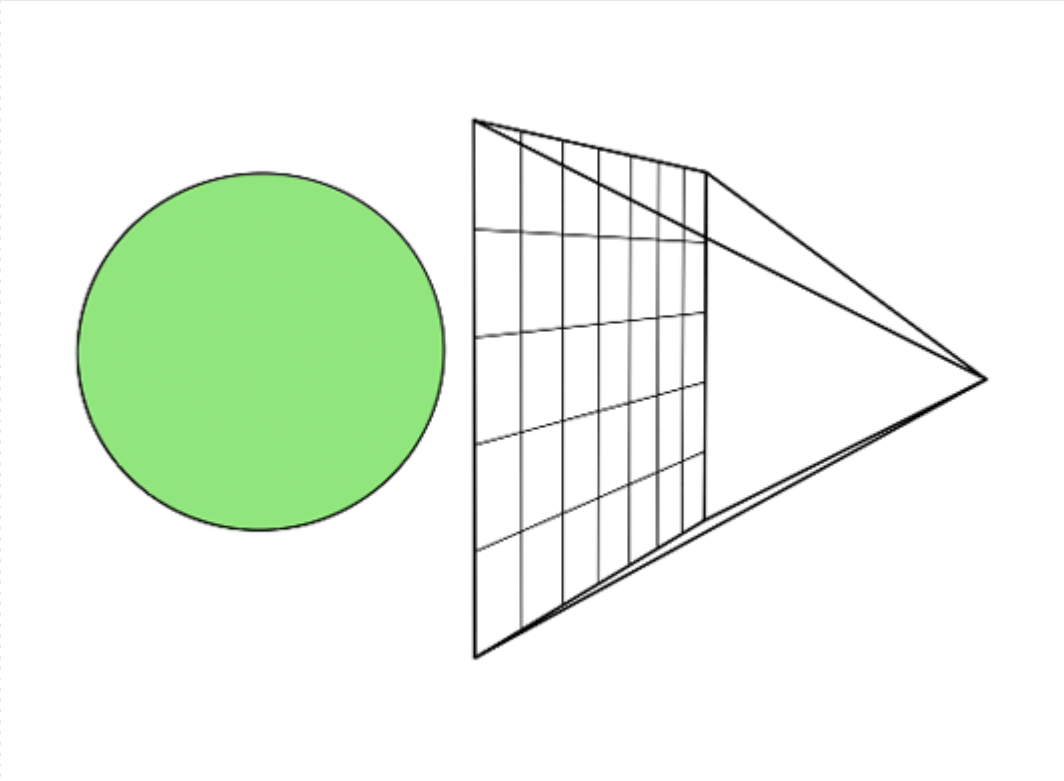


- Ray tracing automatically solves the *hidden surface removal problem*, since the first surface hit by the ray is the closest object to the eye. More remote surfaces are ignored.
- Using a description of light sources in the scene, the same shading model as before is applied to the first hit point, and the ambient, diffuse, and specular components of light are computed
- The resulting color is then displayed at the pixel.

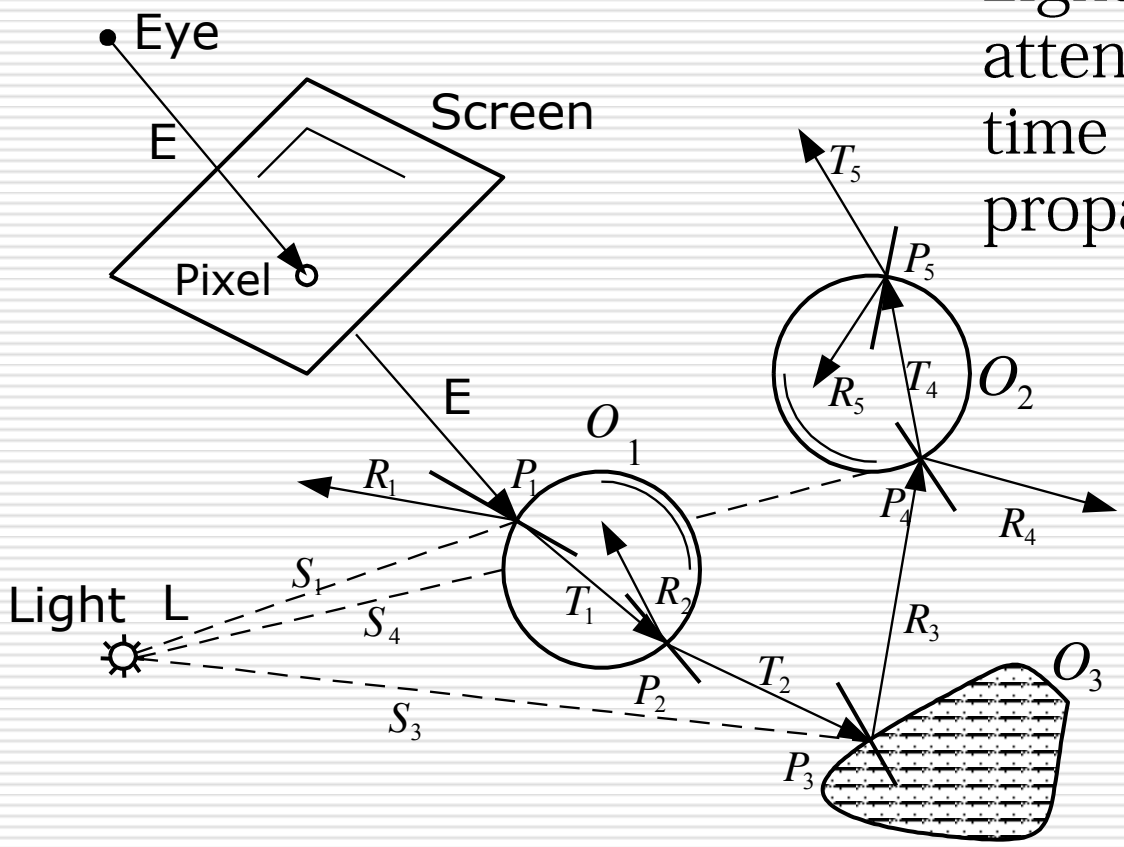
Hidden surface removal



Parallel computation

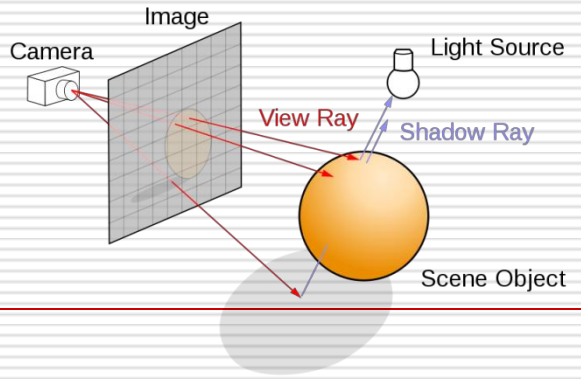


Recursive process



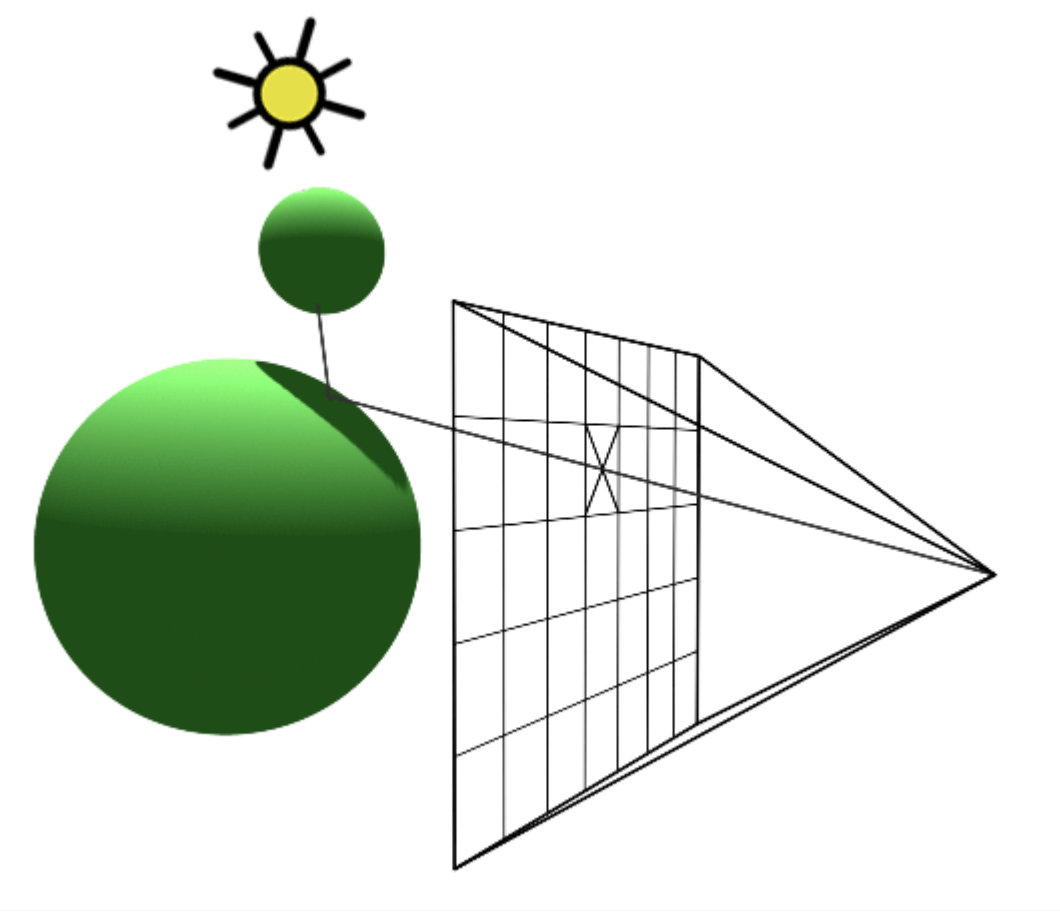
Light intensity attenuates every time during propagation

Ray Tracing

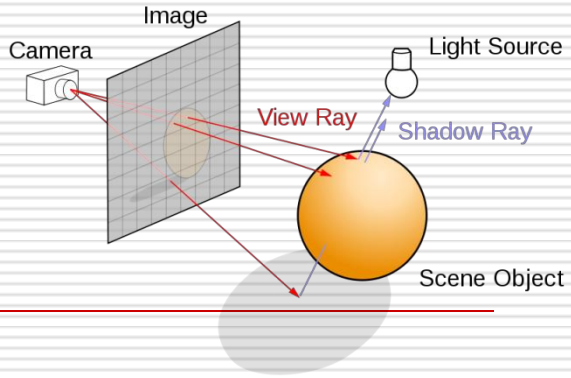


- Because the path of a ray is traced through the scene, effects such as shadowing, reflection, and refraction are easy to incorporate.
- Ray tracing can also work with a richer class of geometric objects than polygon meshes. Solid objects are constructed out of various geometric primitives such as spheres, cones, and cylinders
  - Each shape is represented exactly through a mathematical expression; it is not approximated as a mesh.

Shadowing, reflection, and refraction



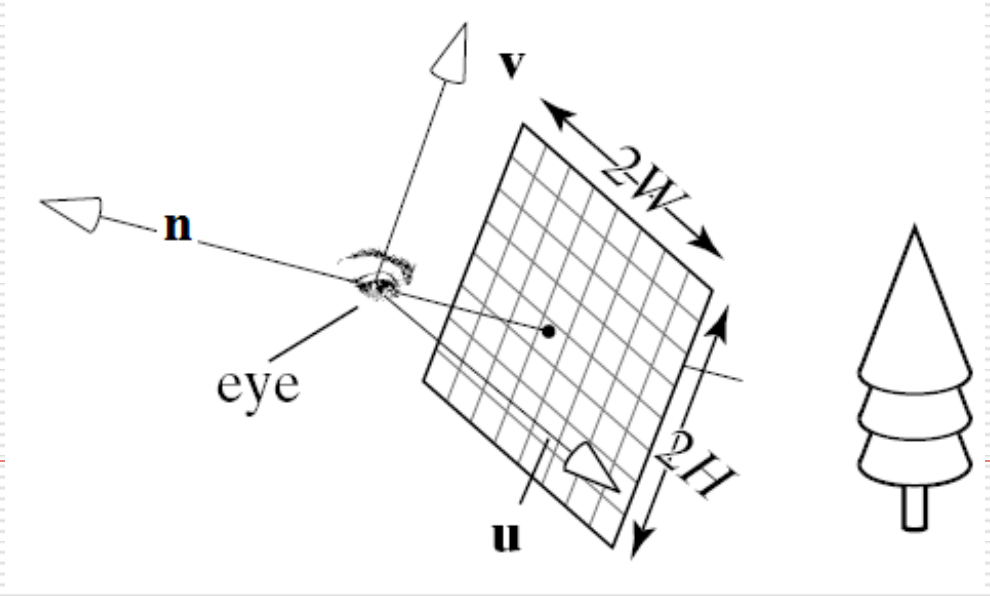
Geometry of Ray Tracing



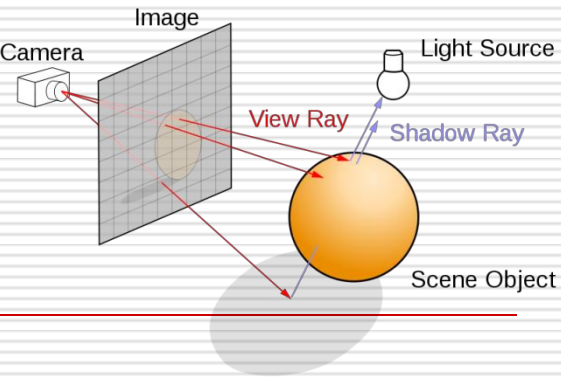
- In order to trace rays, we need a convenient representation for the ray (a parametric representation will be the most serviceable) that passes through a particular pixel.
- By way of warning, a large number of parameters will be necessary in order to describe exactly the geometry of the ray tracing process, which may seem overwhelming at first

Geometry of Ray Tracing (2)

- The same camera model used as before.
- Its eye is at point *eye*, and the axes of the camera are along the vectors **u**, **v**, and **n**.
- The near plane lies at distance N in front of the eye, and the frame buffer lies in the near plane

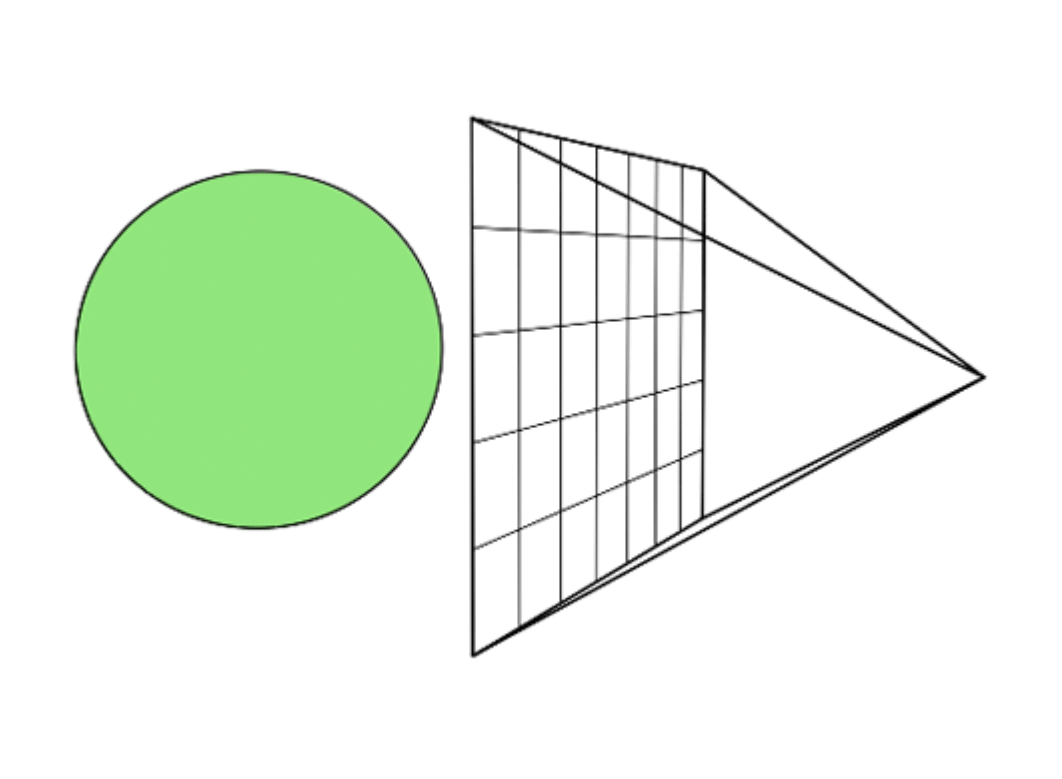


Overview of Ray Tracing



- The scene to be ray traced is inhabited by various geometric objects and light sources, each having a specified shape, size, and position
- The camera is created
- Then for each pixel in turn we construct a ray that starts at the eye and passes through the lower left corner of the pixel. This involves simply evaluating the direction **dir***rc* for the *rc*-th ray

Overview of Ray Tracing



Pseudocode for Ray Tracing

<*define objects, light sources and camera in the scene* >

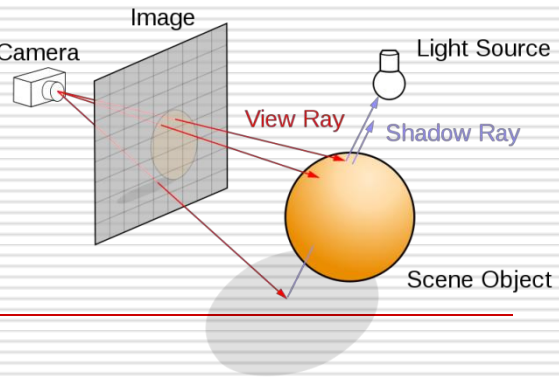
```
for (int r = 0; r < nRows; r++)
  for (int c = 0; c < nCols; c++)
  { < 1. Build the rc-th ray >
    < 2. Find all intersections of rc-th ray with objects >
    < 3. Find  intersection that lies closest to and in front of the eye >
    < 4. Compute the hit point where the ray hits this object, and the normal vector at that point >
    < 5. Find the color of the light returning to the eye along the ray from the point of intersection >
    < 6. Place the color in the rc-th pixel. > }
```

Pseudocode for Ray Tracing

<*define objects, light sources and camera in the scene*>

```
for (int r = 0; r < nRows; r++)
  for (int c = 0; c < nCols; c++)
  { < 1. Build the rc-th ray >
    < 2. Find all intersections of rc-th ray with objects >
    < 3. Find  intersection that lies closest to and in front
of the eye >
    < 4. Compute the hit point where the ray hits this
object, and the normal vector at that point >
    < 5. Find the color of the light returning to the eye
along the ray from the point of intersection >
    < 6. Place the color in the rc-th pixel. > }
```

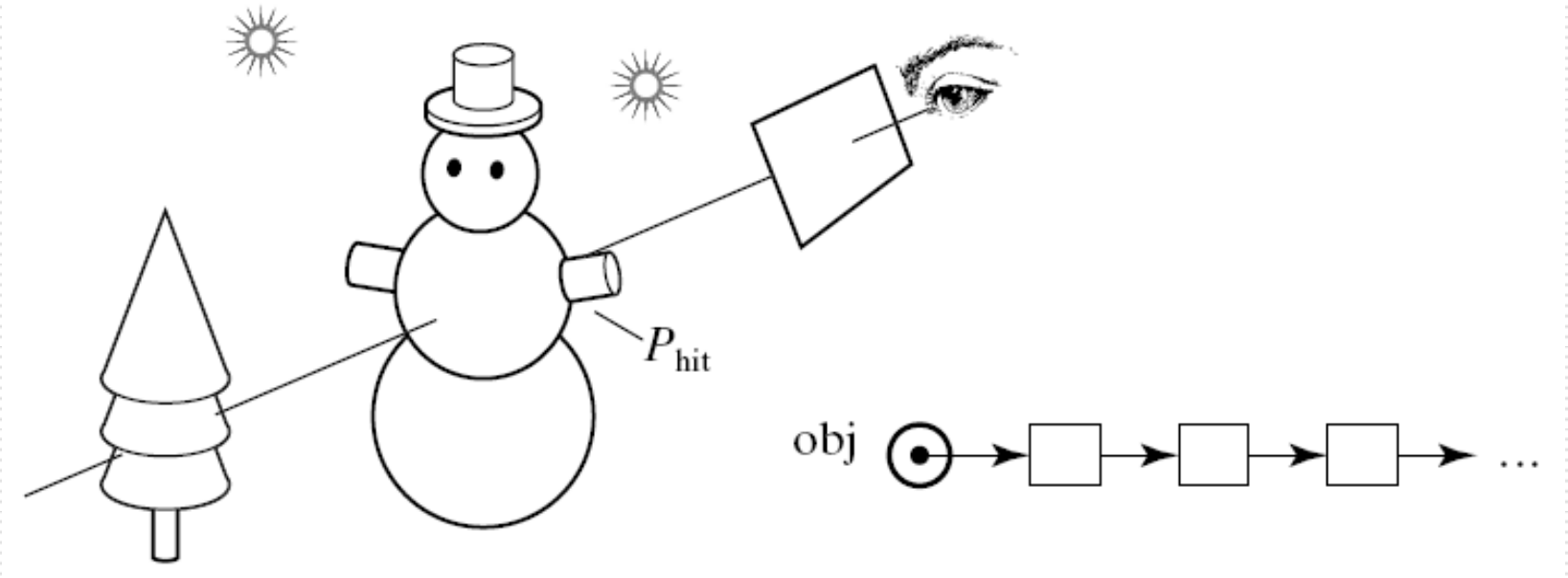
Overview of Ray Tracing



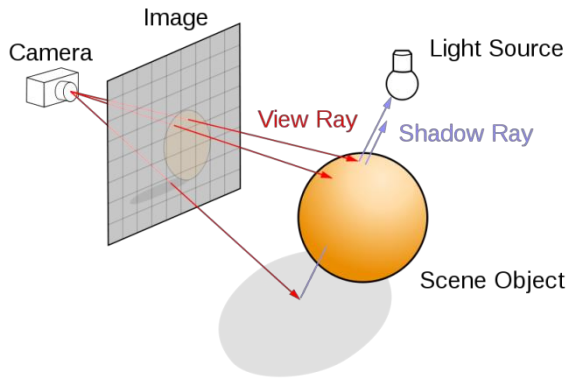
- We first find whether the *rc*-th ray intersects each object in the list, and if so, we note the "hit time" - the value of *t* at which the ray *r*(*t*) coincides with the object's surface.
- When all objects have been tested, the object with the smallest hit time is the closest to the eye
- The location of the hit point on the object is then found, along with the normal vector to this object's surface at the hit point.
- The color of the light that reflects off this object, in the direction of the eye, is then computed and stored in the pixel

Overview of Ray Tracing

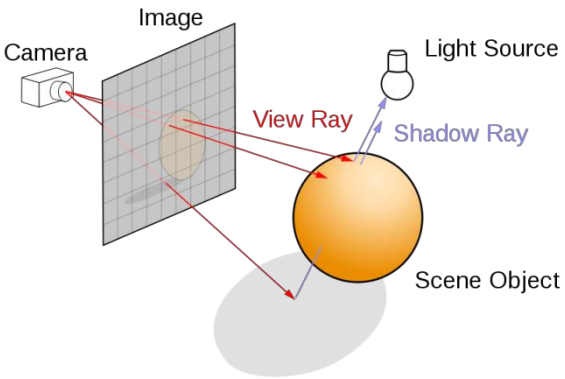
- A simple example scene consisting of some cylinders and spheres, three cones, and two light sources



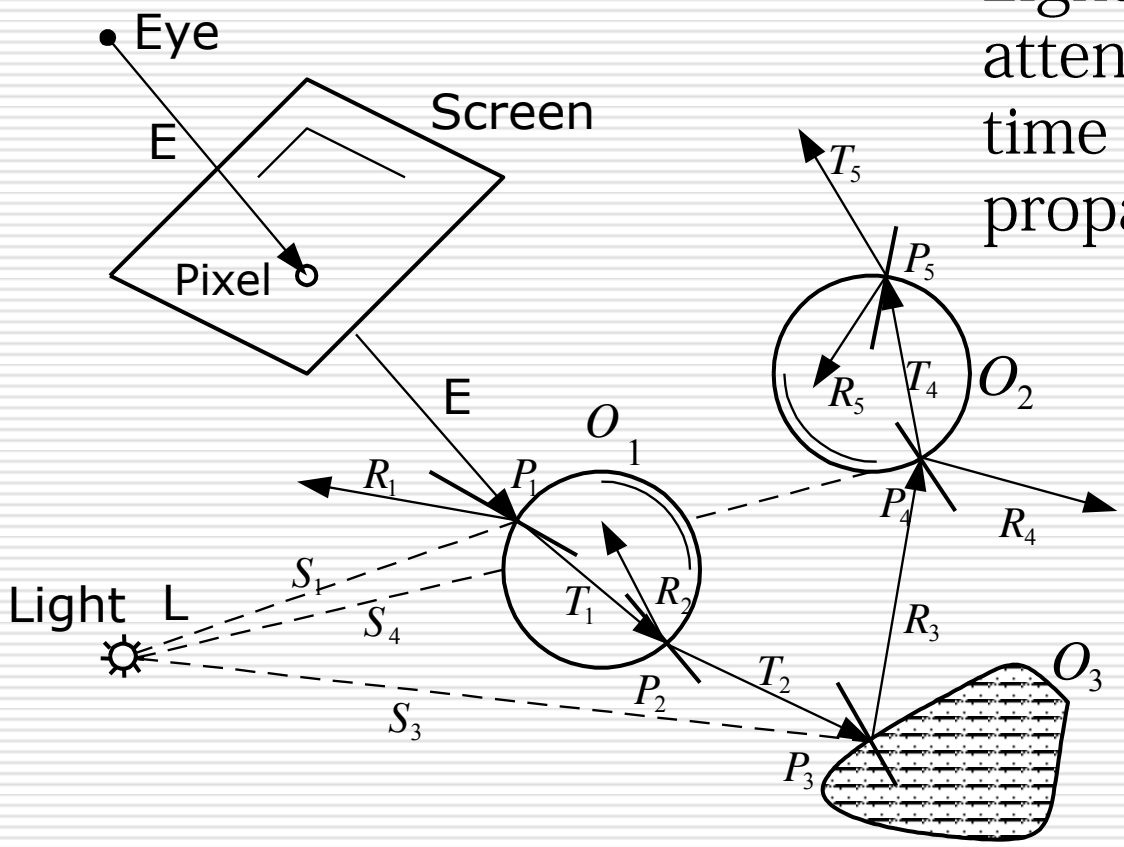
```
for (int j = 0; j < imageHeight; ++j) {
    for (int i = 0; i < imageWidth; ++i) {
    // compute primary ray direction
    Ray primRay;
    computePrimRay(i, j, &primRay);
    // shoot prim ray in the scene and search for intersection
    Point pHit;
    Normal nHit;
    float minDist = INFINITY;
    Object object = NULL;
    for (int k = 0; k < objects.size(); ++k) {
        if (Intersect(objects[k], primRay, &pHit, &nHit)) {
           float distance = Distance(eyePosition, pHit);
           if (distance < minDistance) {
              object = objects[i];
              minDistance = distance;
              // update min distance } } }
        if (object != NULL) {
          // compute illumination Ray
        }
        if (!isInShadow) pixels[i][j] = object->color * light.brightness;
        else pixels[i][j] = 0;
    }
}
```



```
for (int j = 0; j < imageHeight; ++j) {
    for (int i = 0; i < imageWidth; ++i) {
    // compute primary ray direction
    … … …
    for (int k = 0; k < objects.size(); ++k) {
        if (Intersect(objects[k], primRay, &pHit, &nHit)) {
        … … …
        }
        if (object != NULL) {
           // compute illumination
           Ray shadowRay;
           shadowRay.direction = lightPosition - pHit;
           bool isShadow = false;
           for (int k = 0; k < objects.size(); ++k) {
              if (Intersect(objects[k], shadowRay)) {
                 isInShadow = true;
                 break;
              }
           }
        }
       if (!isInShadow) pixels[i][j] = object->color * light.brightness;
       else pixels[i][j] = 0;
    }
}
```



Recursive process



Light intensity attenuates every time during propagation

Adding Surface Texture

- We can incorporate texturing into a ray tracer.

Adding Surface Texture

- We can incorporate texturing into a ray tracer.
  - The texture ***tex*** can modulate the ambient and diffuse reflection coefficients, so that

$$I = tex(x, y, z) \times \left(I_a \rho_a + I_d \rho_d \left(u_s \bullet u_n \right) + I_s \rho_s \left(u_b \bullet u_n \right)^f \right)$$

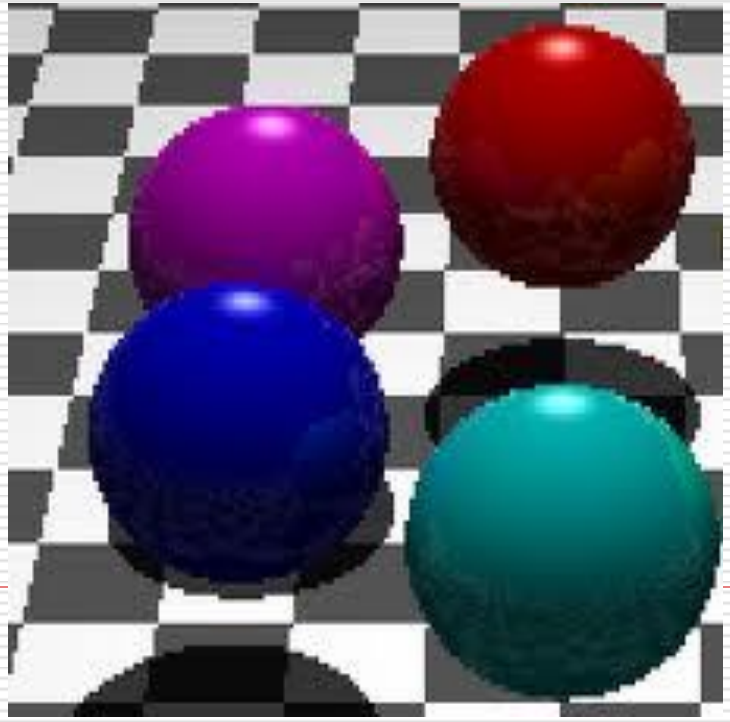
  - where **texture(x, y, z)** is evaluated at the hit point (x, y, z) of the ray (most common use of texture)

Adding Surface Texture

- We can incorporate texturing into a ray tracer.

Anti-Aliasing Ray Tracings

- Ray tracing is inherently a point sampling process - taking discrete looks at a scene along individual rays



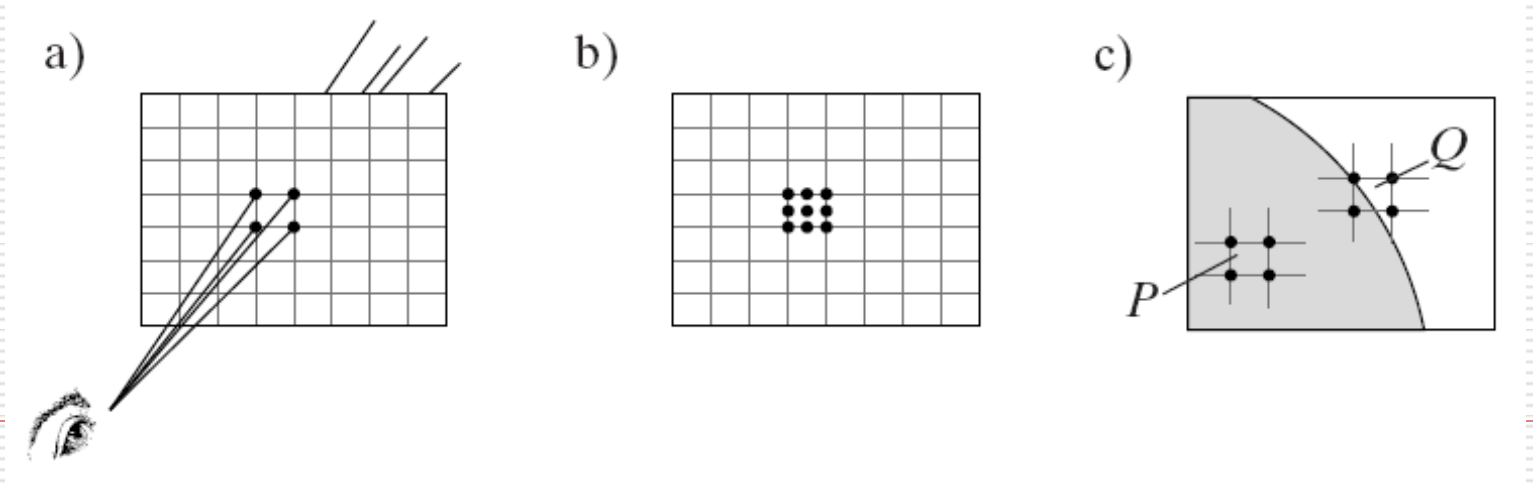
Anti-Aliasing Ray Tracings

- Aliasing effects often degrade the quality of ray traced images and can be reduced by sampling a scene at more points, often called supersampling.
- Several rays per pixel are traced into the scene, and the intensities that are returned along the ray are averaged. This is, of course, costly in execution time.

Anti-Aliasing Ray Tracings

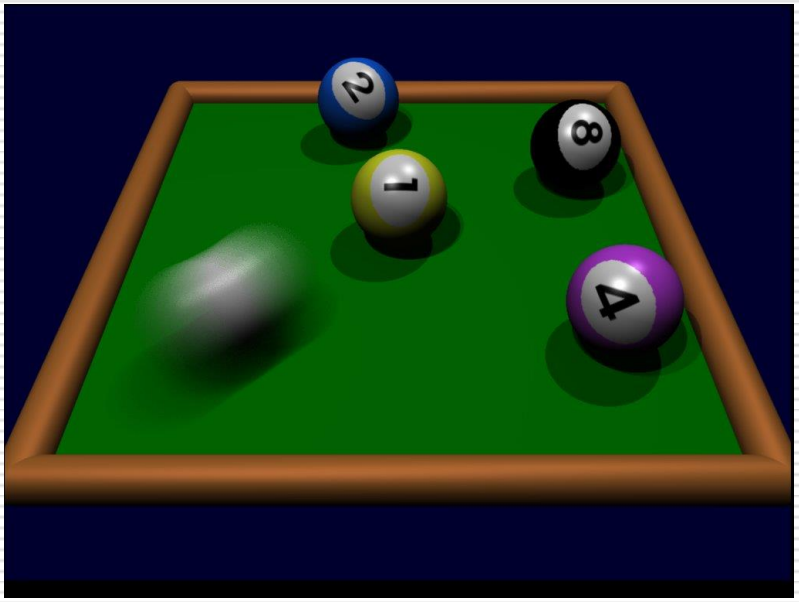
Supersampling.

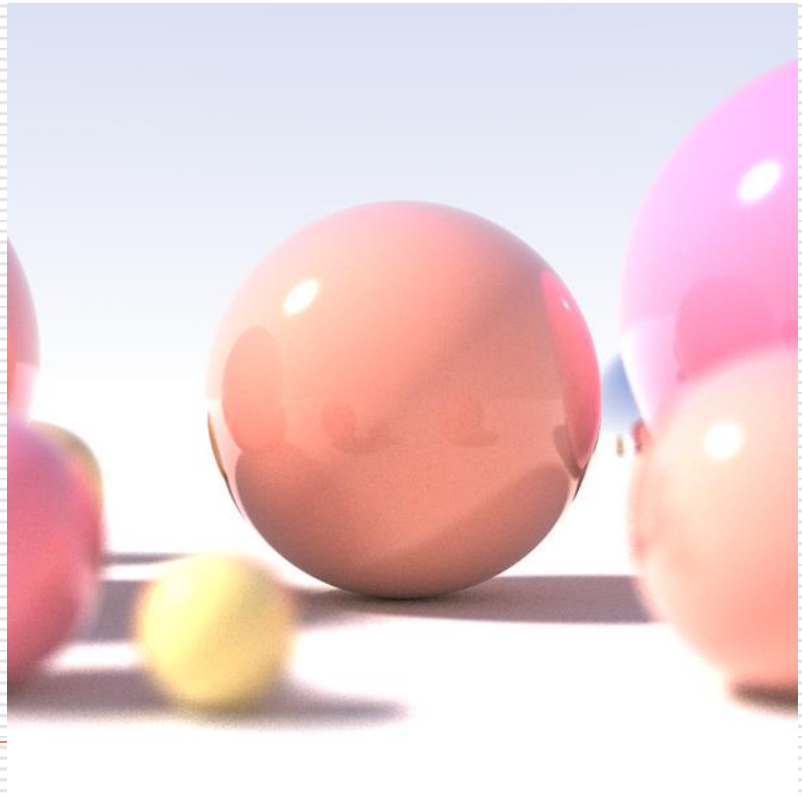
- Several rays per pixel are traced into the scene, and the intensities that are returned along the ray are averaged. This is, of course, costly in execution time.



Anti-Aliasing Ray Tracings

- The figure shows the effect of jittering on a billiard ball



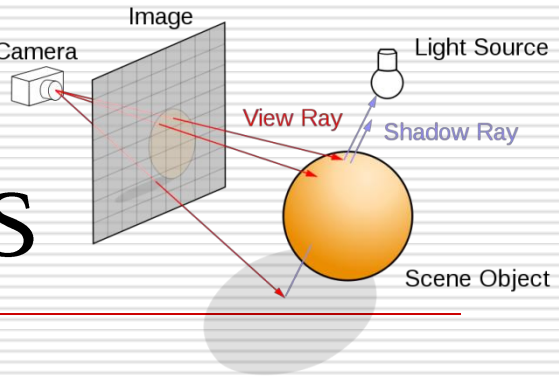


Organizing a Ray Tracer

A ray tracer has to deal with several different interacting objects

- Rays that emanate from the camera and migrate through the scene
- The scene may contains many geometric objects and light sources.
- The color seen at the hit point depends on the combination of the ambient, diffuse, and specular light contributions

Speed up using extents

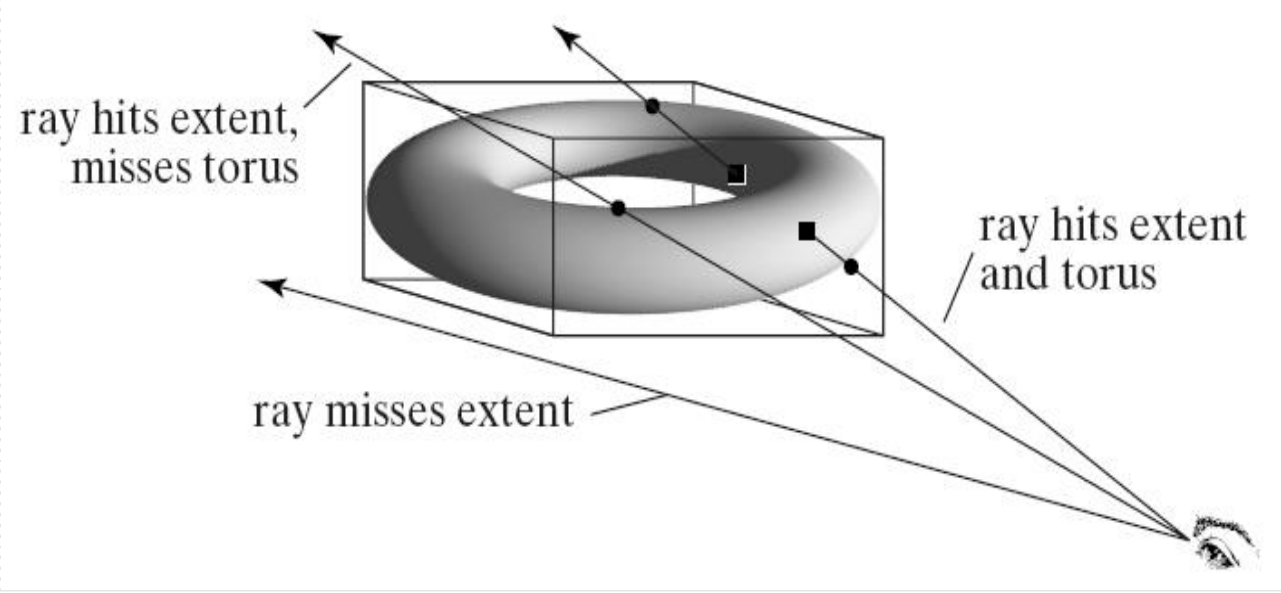


Ray tracing is very repetitive, performing the same set of operations again and again for a very large number of rays

- Each ray must be intersected with every object, amounting to an enormous number of intersection calculations
- Matters will get much worse when we incorporate shadows, reflections, and refractions
- The use of extents can speed up the ray tracing process significantly

Speed up using extents

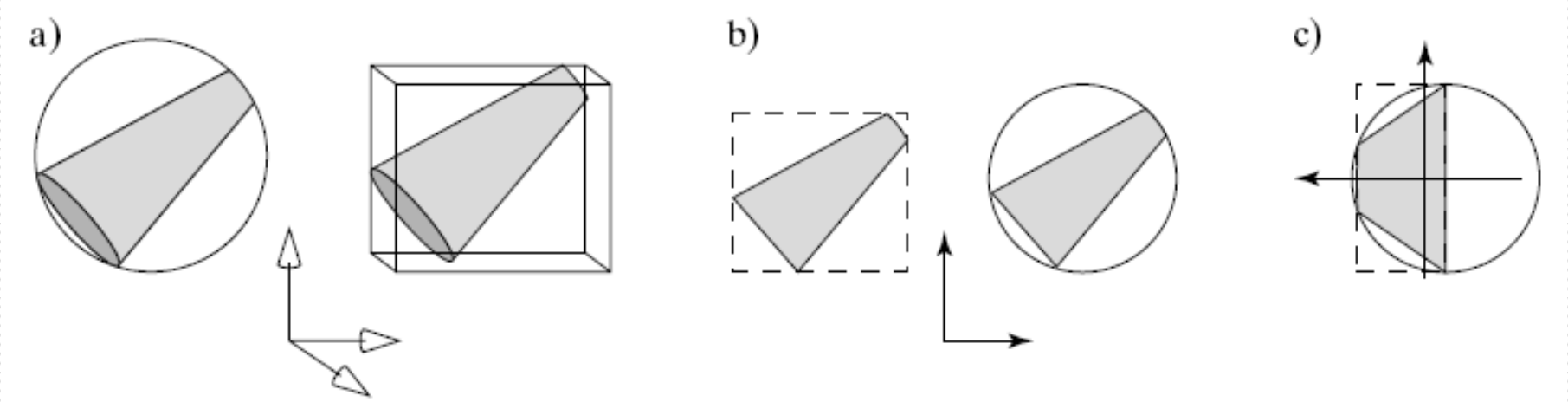
An **extent** of an object is a shape that encloses that object



Speed up using extents

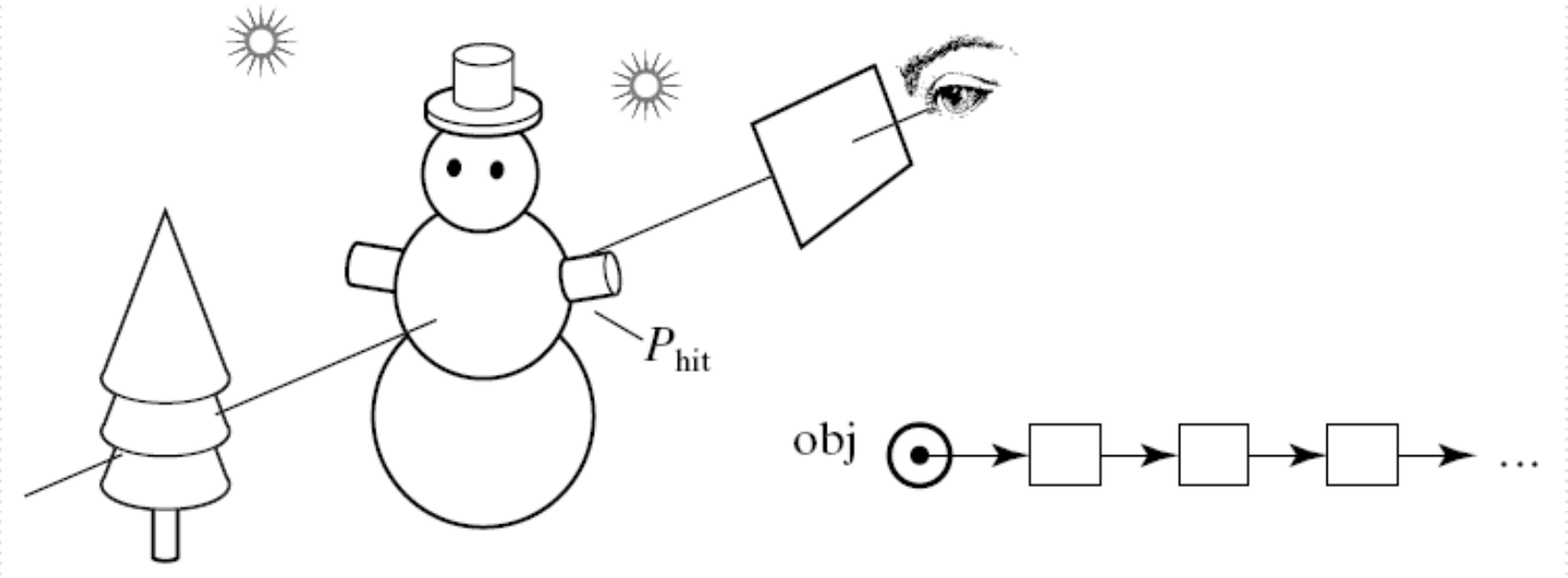
Extent : a shape that encloses that object

- It accelerates the ray tracing process by quickly revealing when the current ray could not possibly hit a particular object
  - The notion is that if a ray misses the extent, it must perforce miss the object



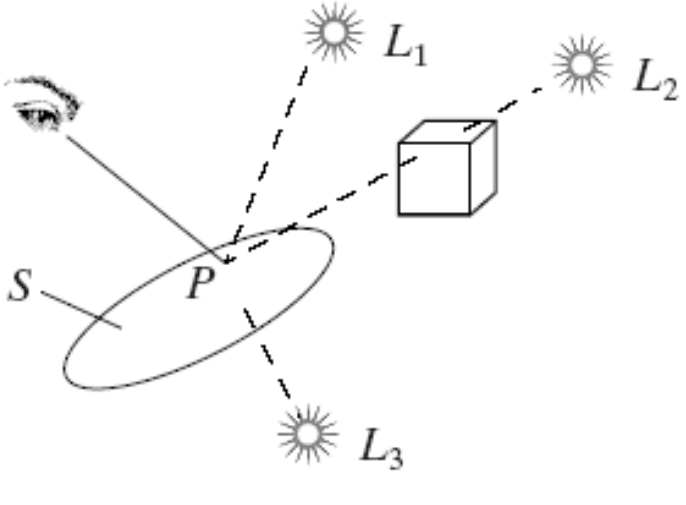
Overview of Ray Tracing

- A simple example scene consisting of some cylinders and spheres, three cones, and two light sources

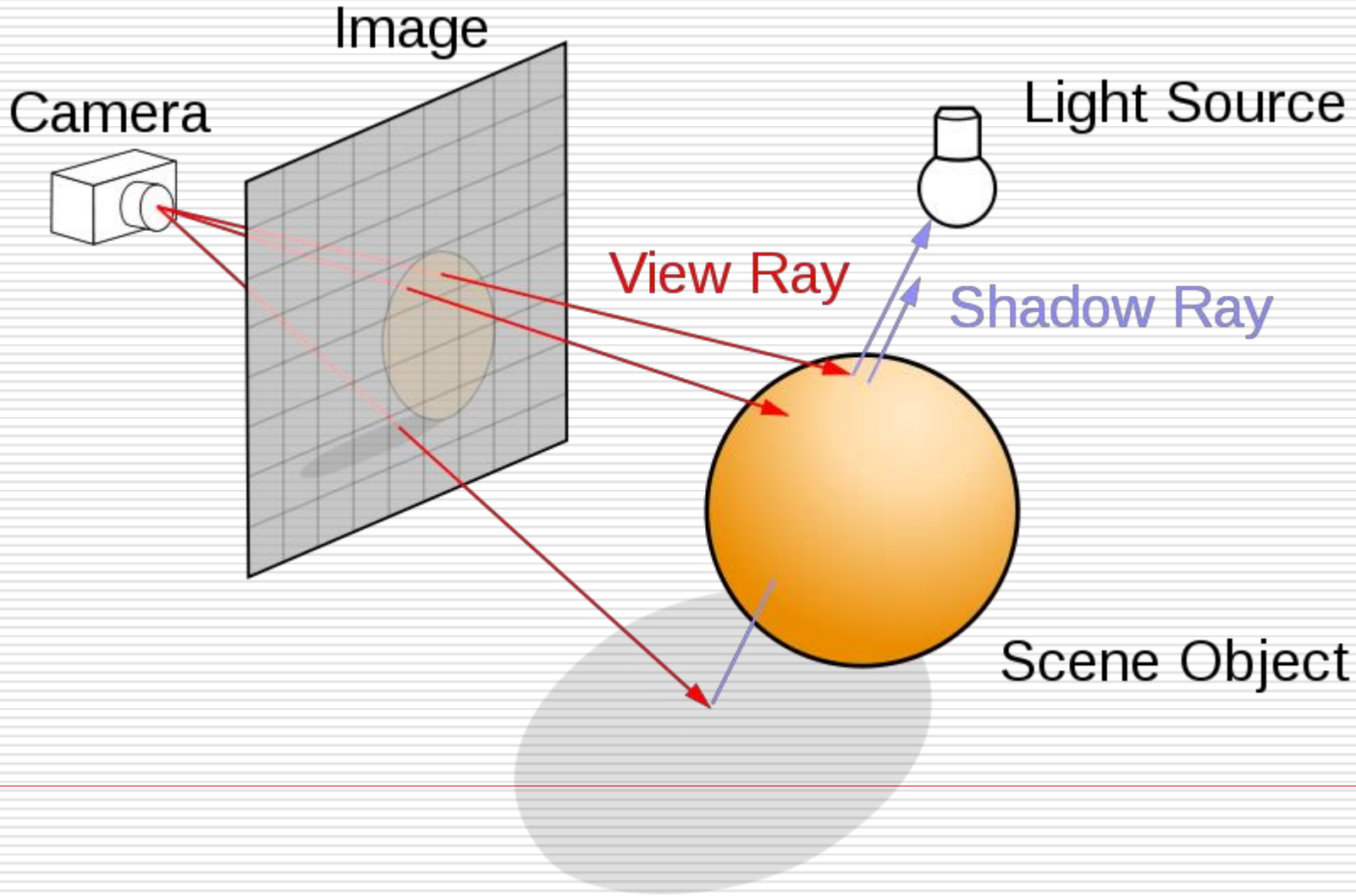


Shadows

- Ray tracing produces shadows easily.
- Shadow classification
  - Point P can see source L_1, so P is not in shadow with respect to L_1.
  - But P *is* in the shadow of the cube with respect to source L_2. Further, the hit object itself hides the source L_3 from P; this is called self-shadowing



Ray Tracing

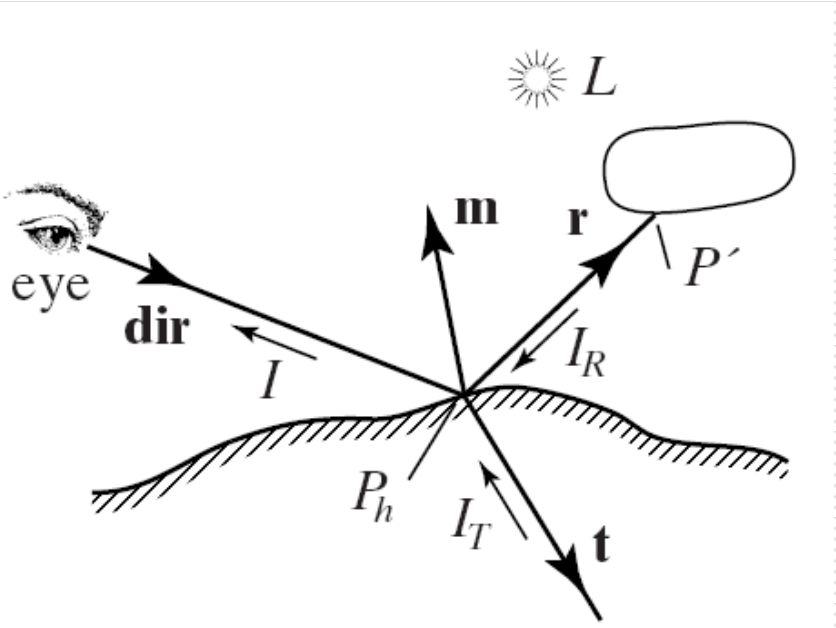


Reflections and transparency

- One of the great strengths of the ray tracing method is the ease with which it can handle both the reflection and the refraction of light
  - This allows one to build scenes of exquisite realism containing mirrors, fish bowls, lenses, and the like
- Each of these processes requires the spawning and tracing of additional rays

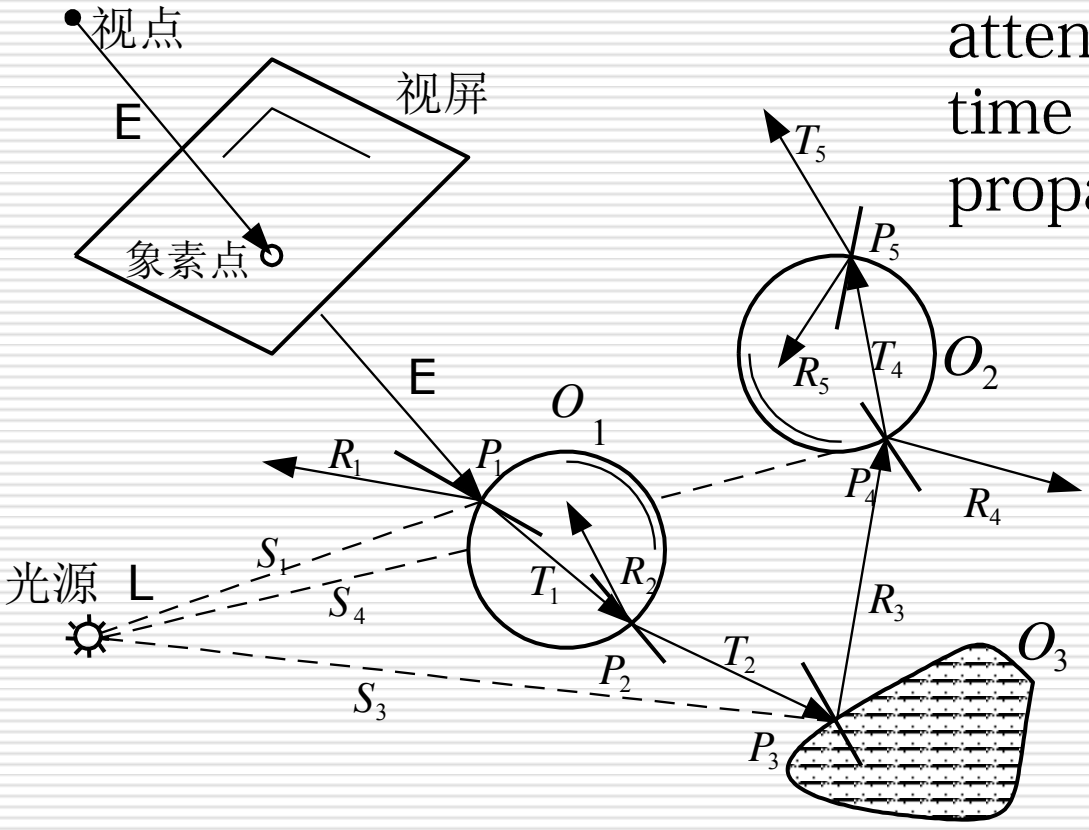
Reflections and transparency

- The figure shows a ray emanating from the eye in the direction **dir** and hitting a surface at the point P_h.
- The figure is in 2D, which is acceptable because the nature of reflection and refraction causes all vectors to lie in the same plane
- All formulas we develop operate in 3D.



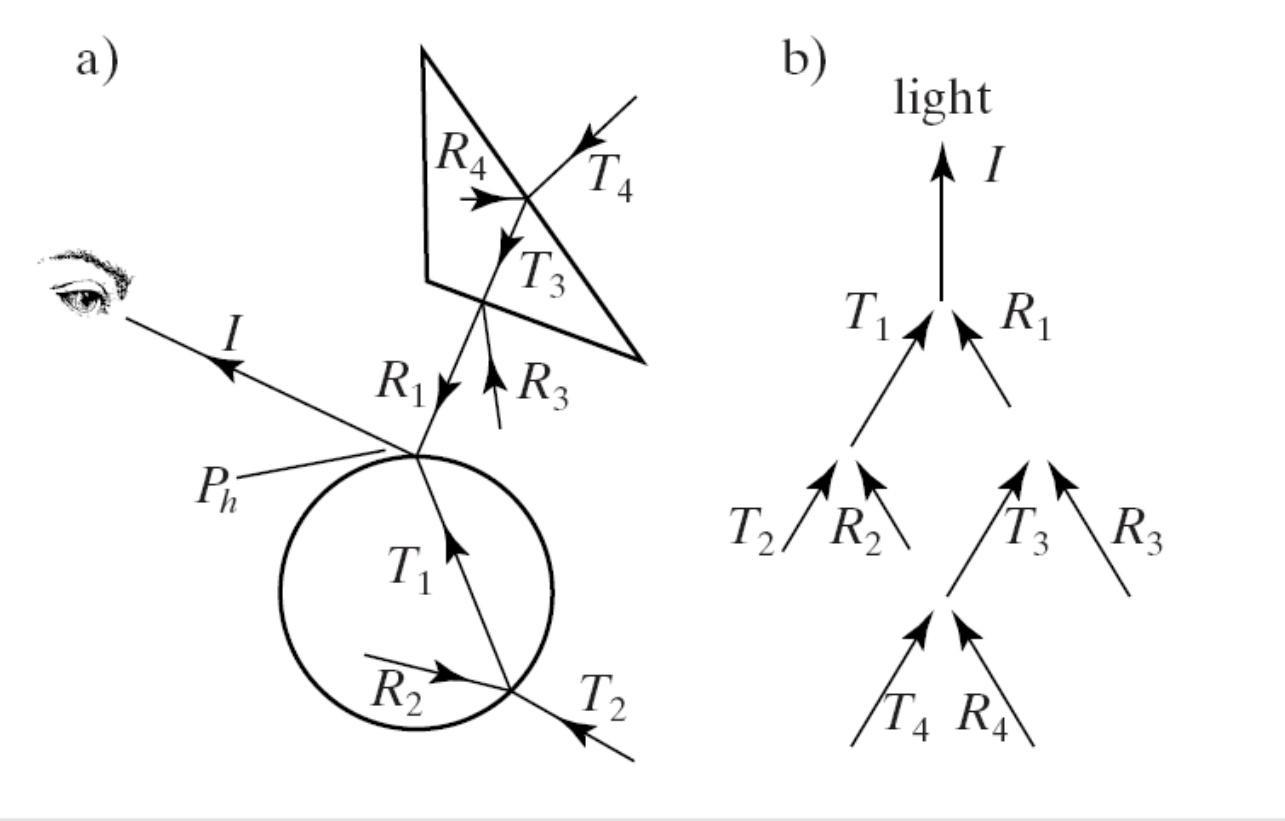
Recursive process

Light intensity attenuates every time during propagation



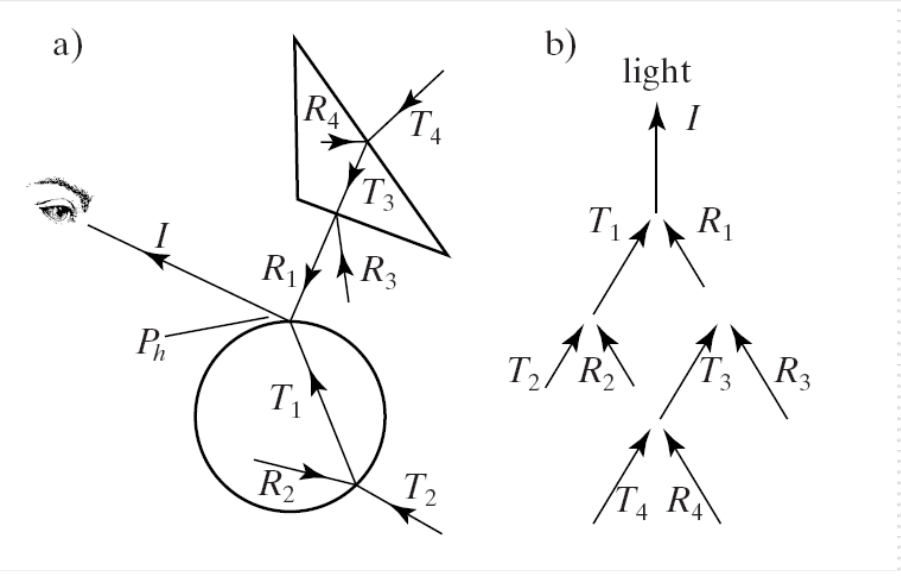
Reflections and Transparency

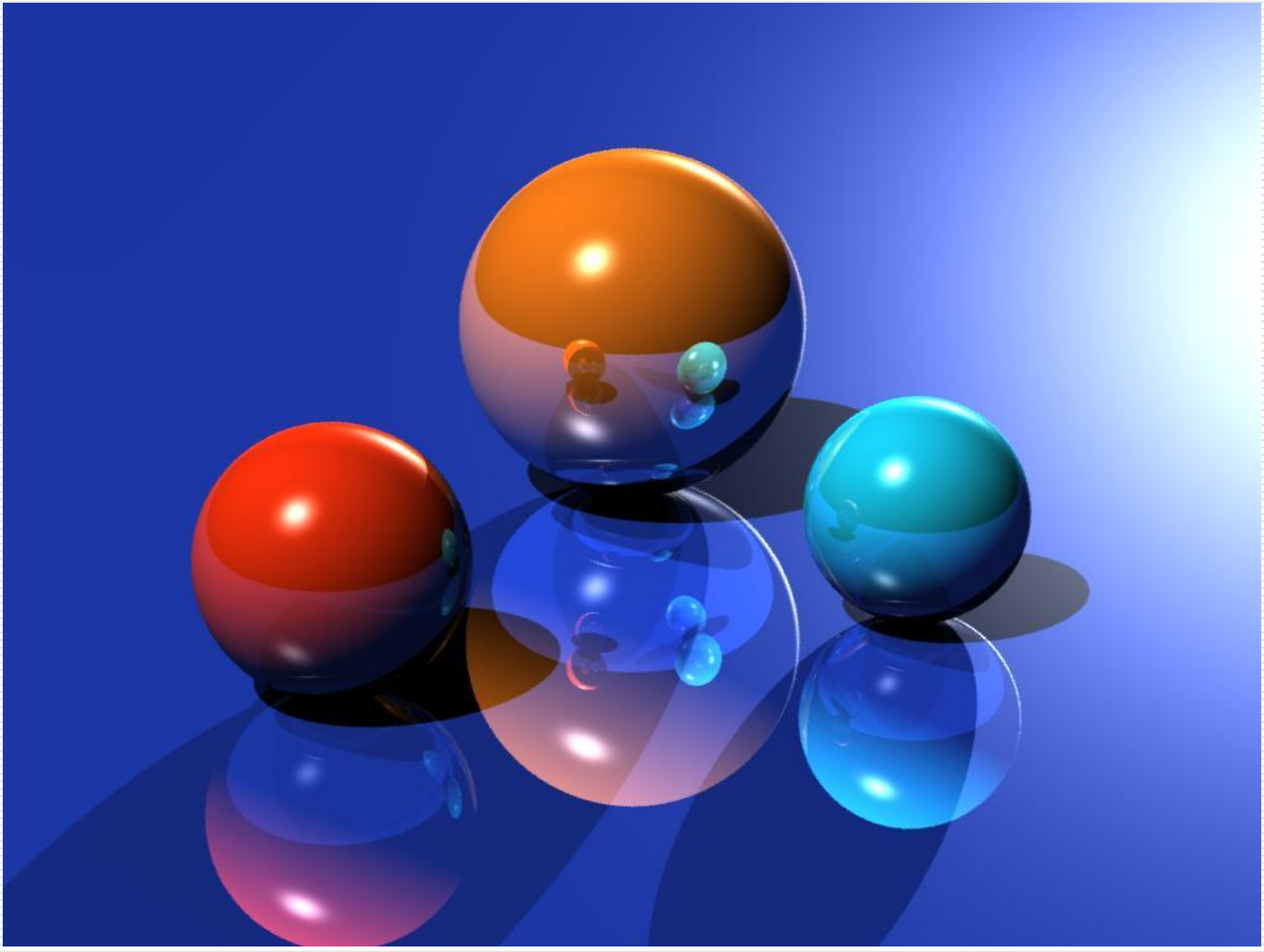
- The tree of light rays; local components are not shown



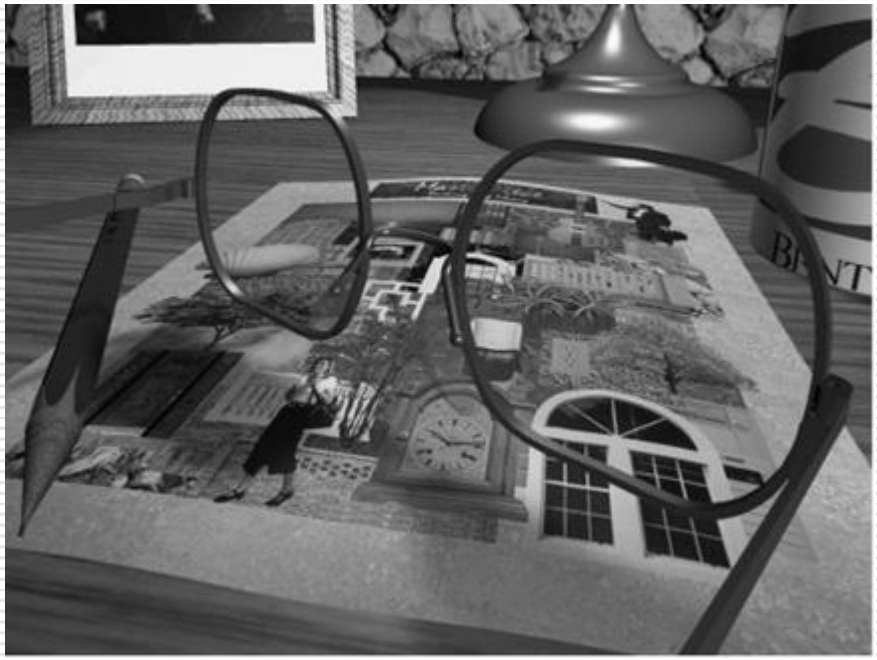
Reflections and Transparency

- The tree of light rays; local components are not shown
- The light intensity contributed at each level is decayed when the level is increased.





Local vs Global Illuminations



OpenGL model

Ray tracing

Specular, reflective and transparent components

End.
Thanks