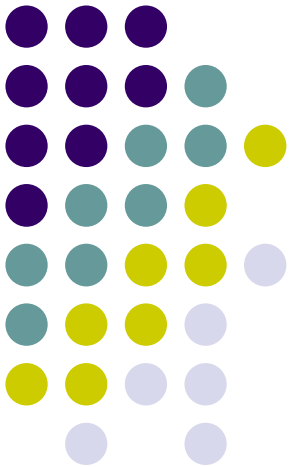
Computer Graphics (CS 543)
Lecture 12: Part 2
Ray Tracing (Part 1)

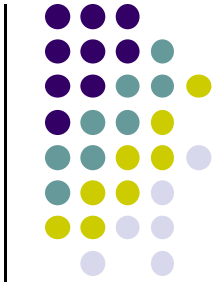
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

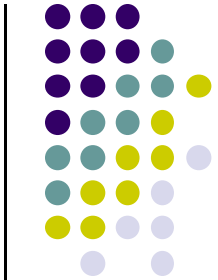


Raytracing



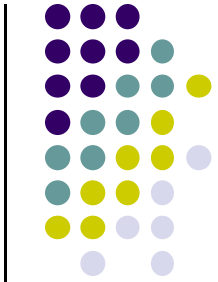
- Global illumination-based rendering method
- Simulates rays of light, natural lighting effects
- Because light path is traced, handles effects tough for openGL:
  - Shadows
  - Multiple inter-reflections
  - Transparency
  - Refraction
  - Texture mapping
- Newer variations… e.g. photon mapping (caustics, participating media, smoke)
- **Note:** raytracing can be semester graduate course
- Today: start with high-level description

Raytracing Uses



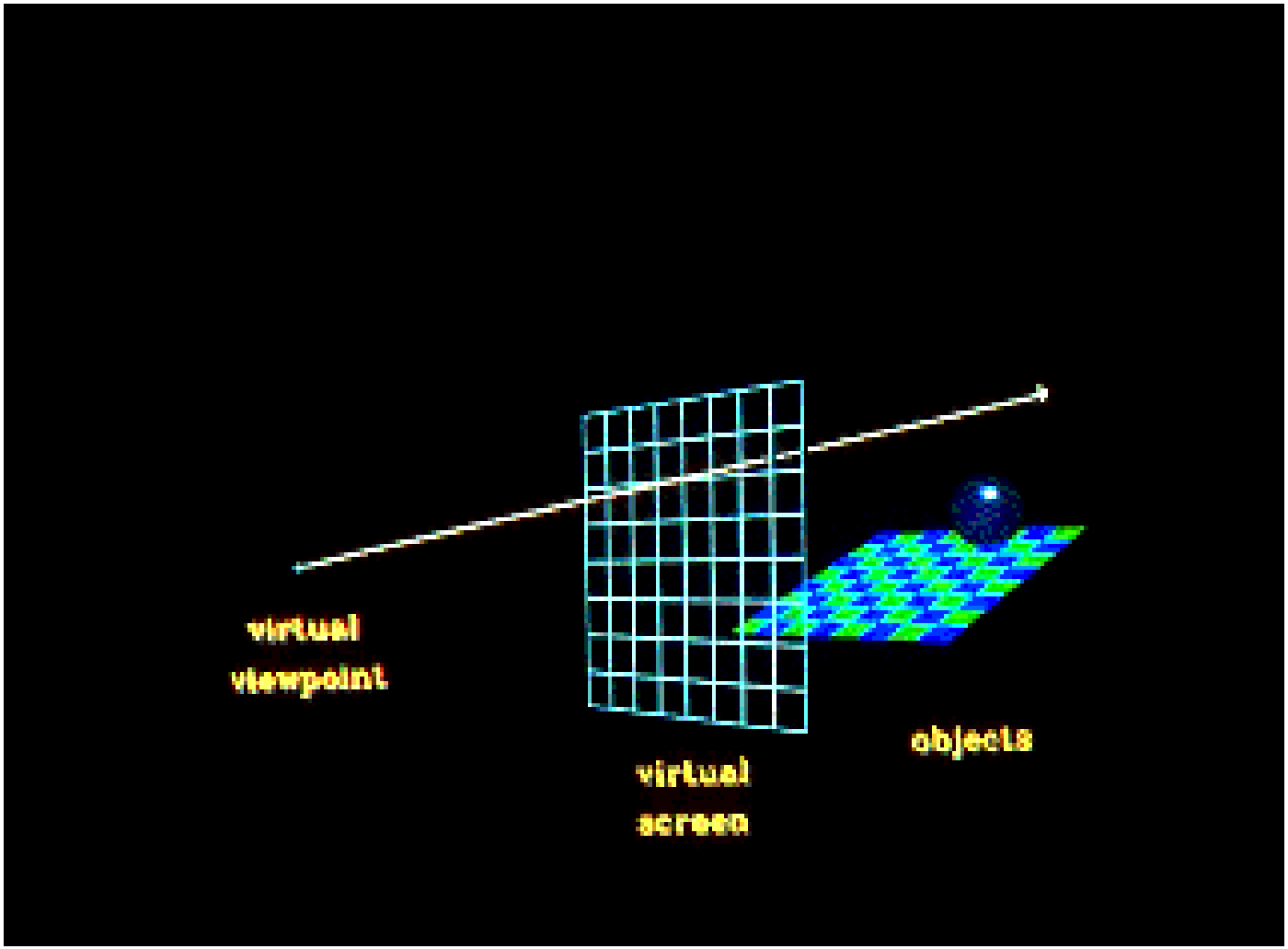
- Entertainment (movies, commercials)
- Games (pre-production)
- Simulation (e.g. military)
- Image: Internet Ray Tracing Contest Winner (April 2003)

How Raytracing Works

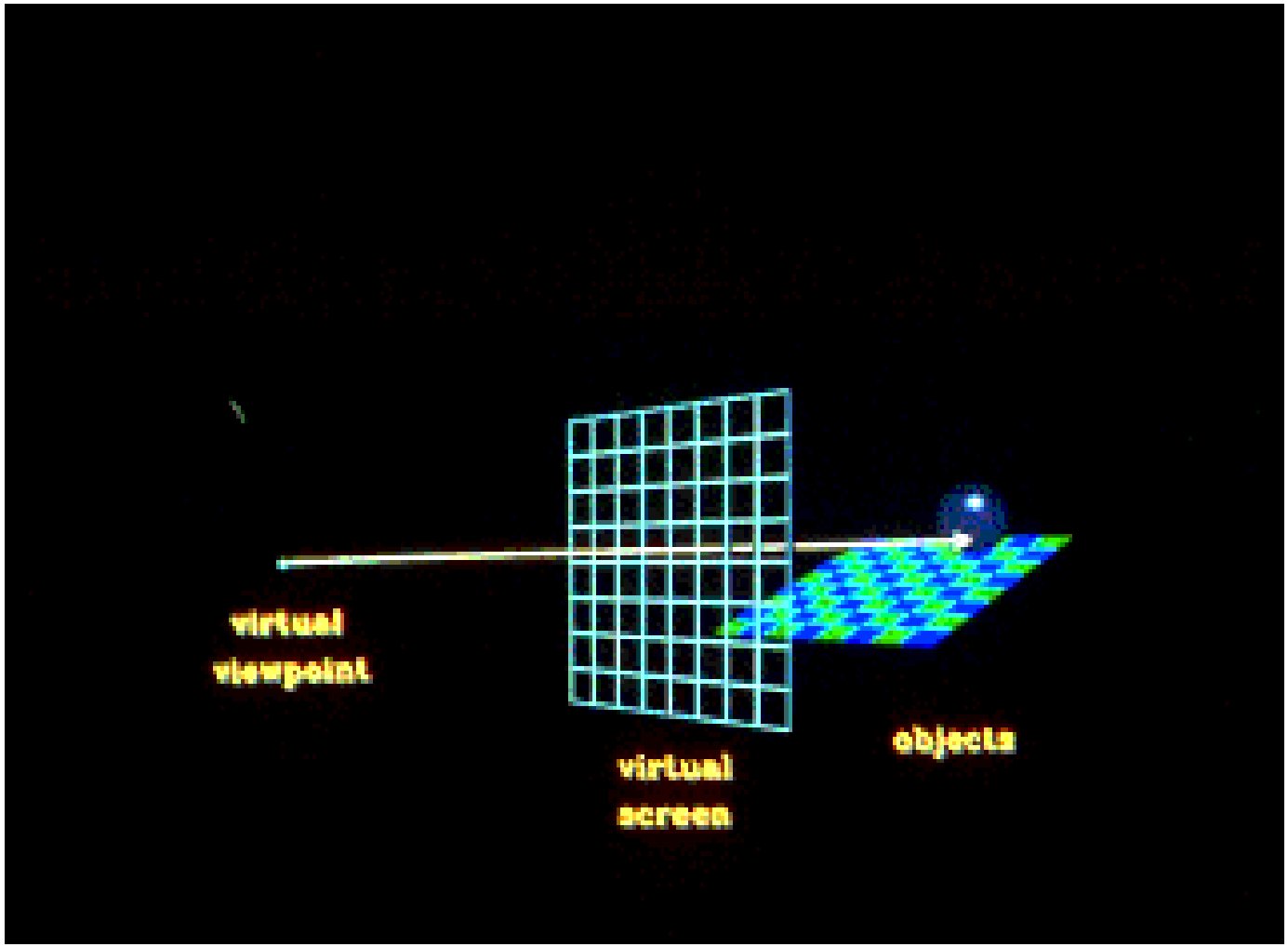


- OpenGL is object space rendering
  - start from world objects, rasterize them
- Ray tracing is image space method
  - Start from pixel, what do you see through this pixel?
- Looks through each pixel (e.g. 640 x 480)
- Determines what eye sees through pixel
- Basic idea:
  - Trace light rays: eye -> pixel (image plane) -> scene
  - If a ray intersect any scene object in this direction
    - Yes? render pixel using object color
    - No? it uses the background color
- Automatically solves hidden surface removal problem

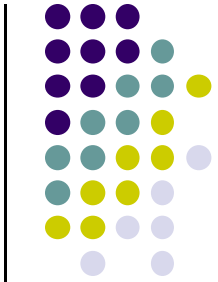
Case A: Ray misses all objects



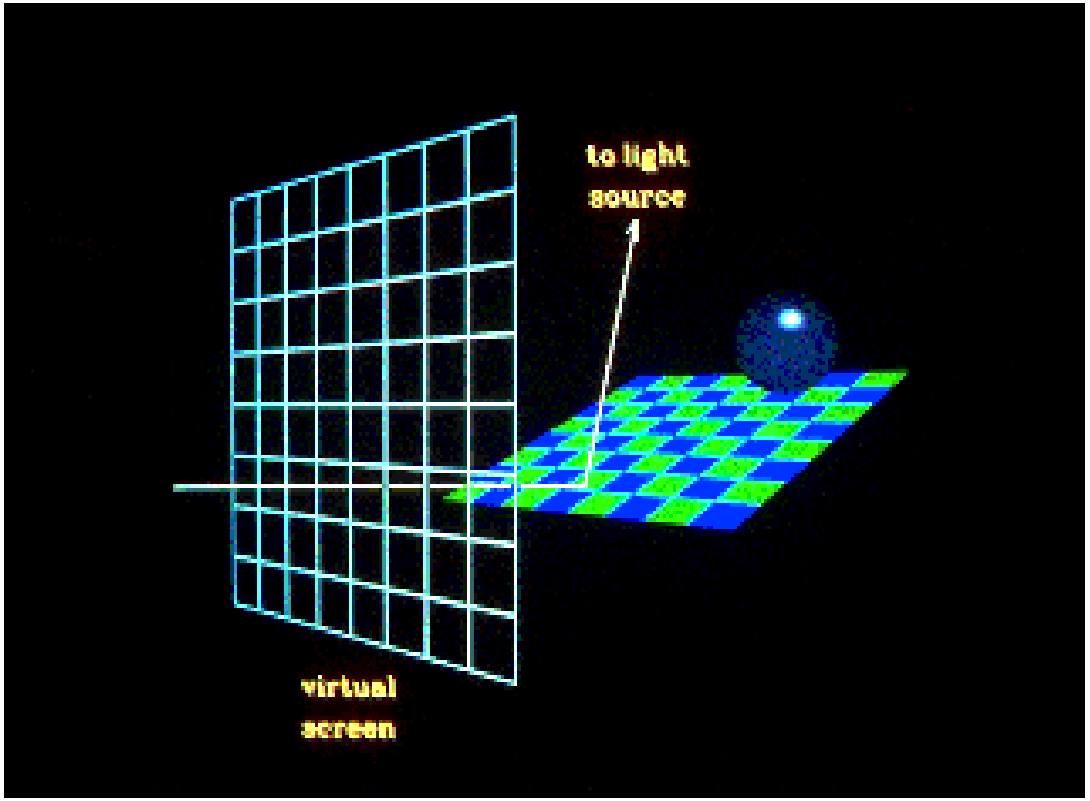
Case B: Ray hits an object

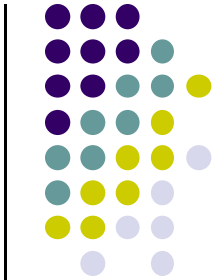


Case B: Ray hits an object



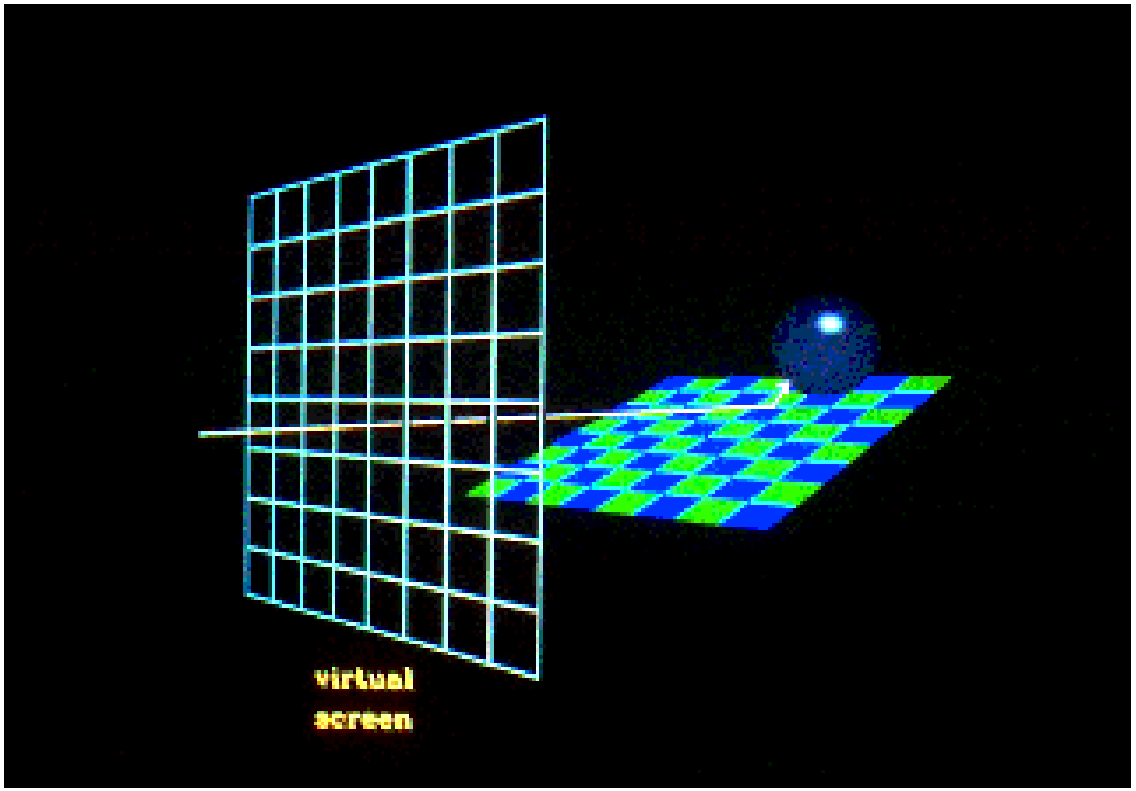
- **Ray hits object:** Check if hit point is in shadow, build secondary ray (shadow ray) towards light sources.





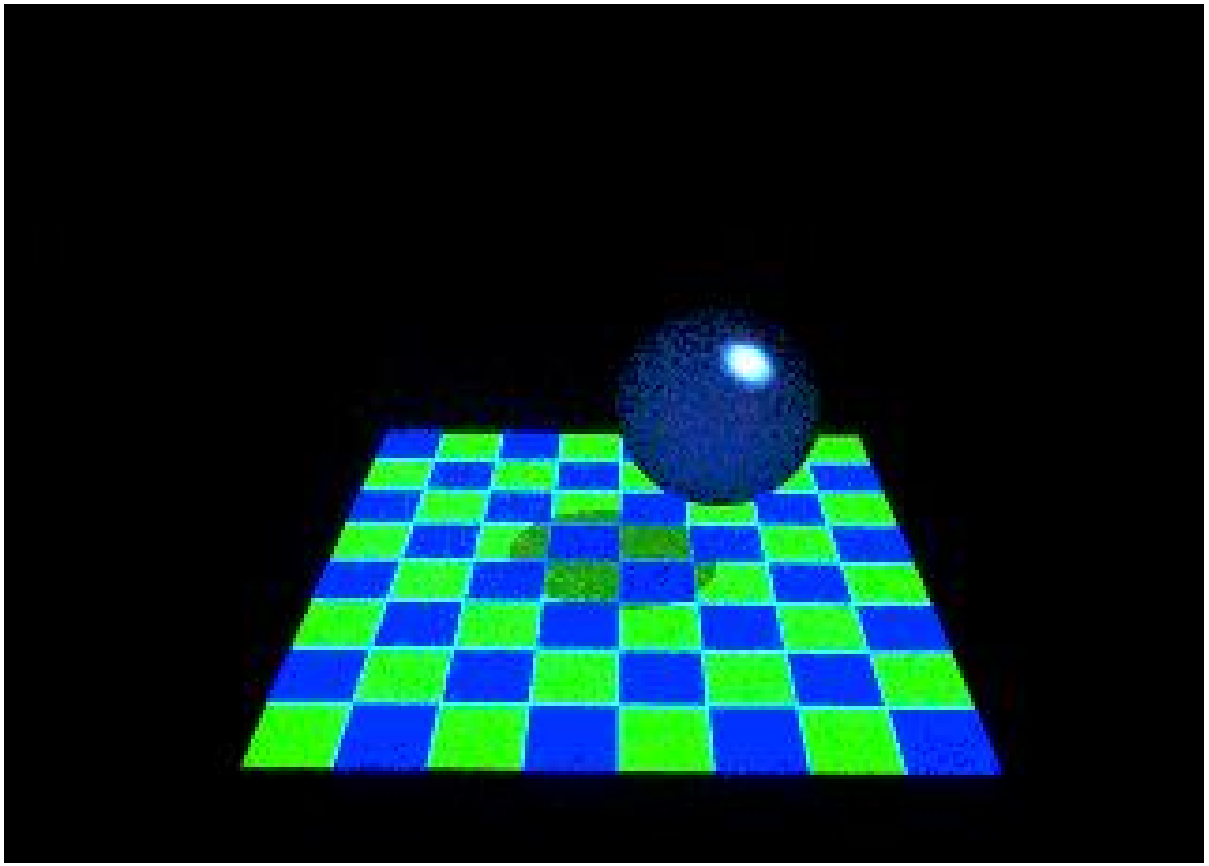
Case B: Ray hits an object

- If shadow ray hits another object before light source: first intersection point is in shadow of the second object. Otherwise, collect light contributions

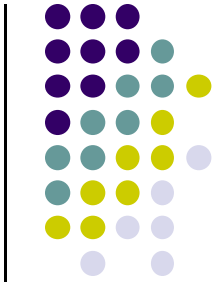


Case B: Ray hits an object

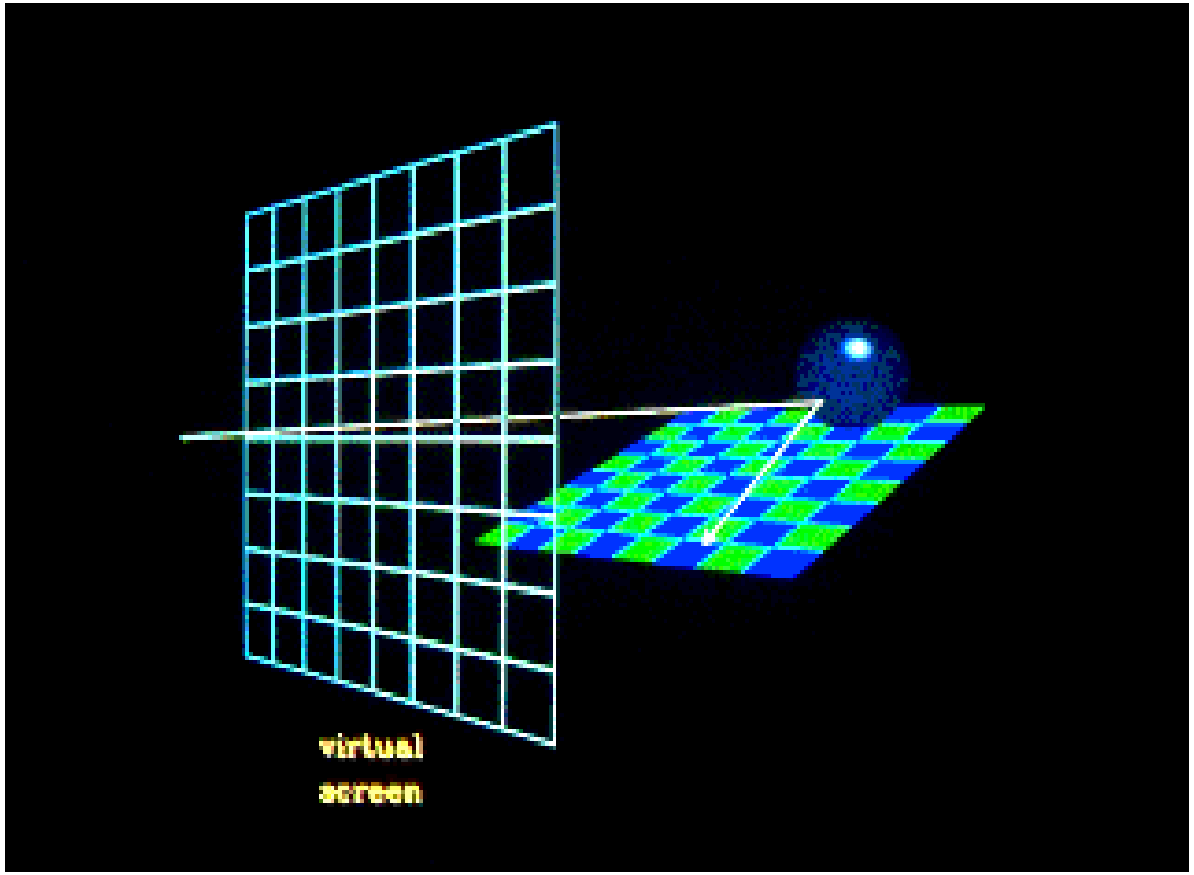
- First Intersection point in the shadow of the second object is the shadow area.



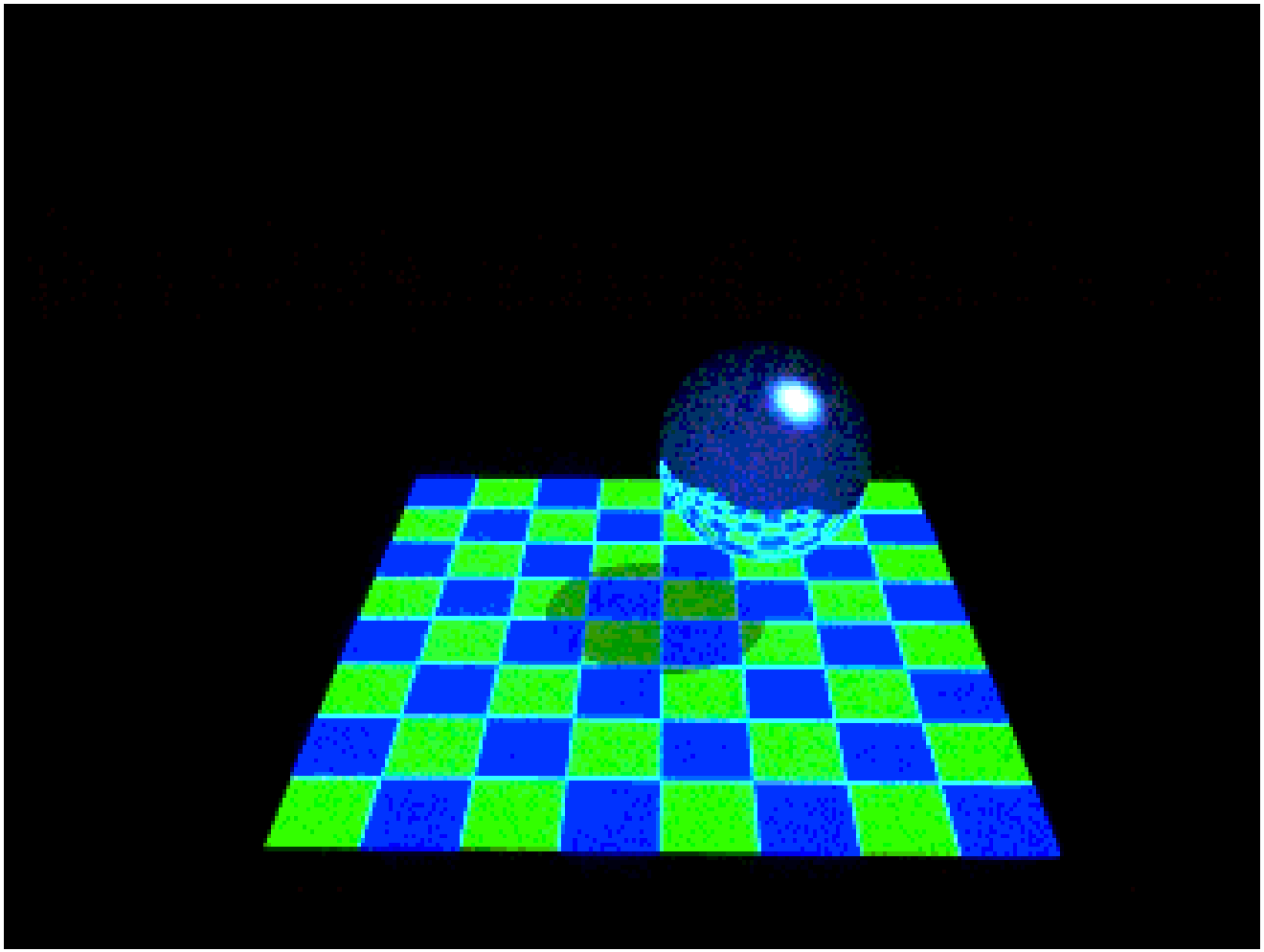
Reflected Ray



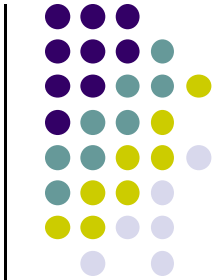
- When a ray hits an object, a reflected ray is generated which is tested against all of the objects in the scene.



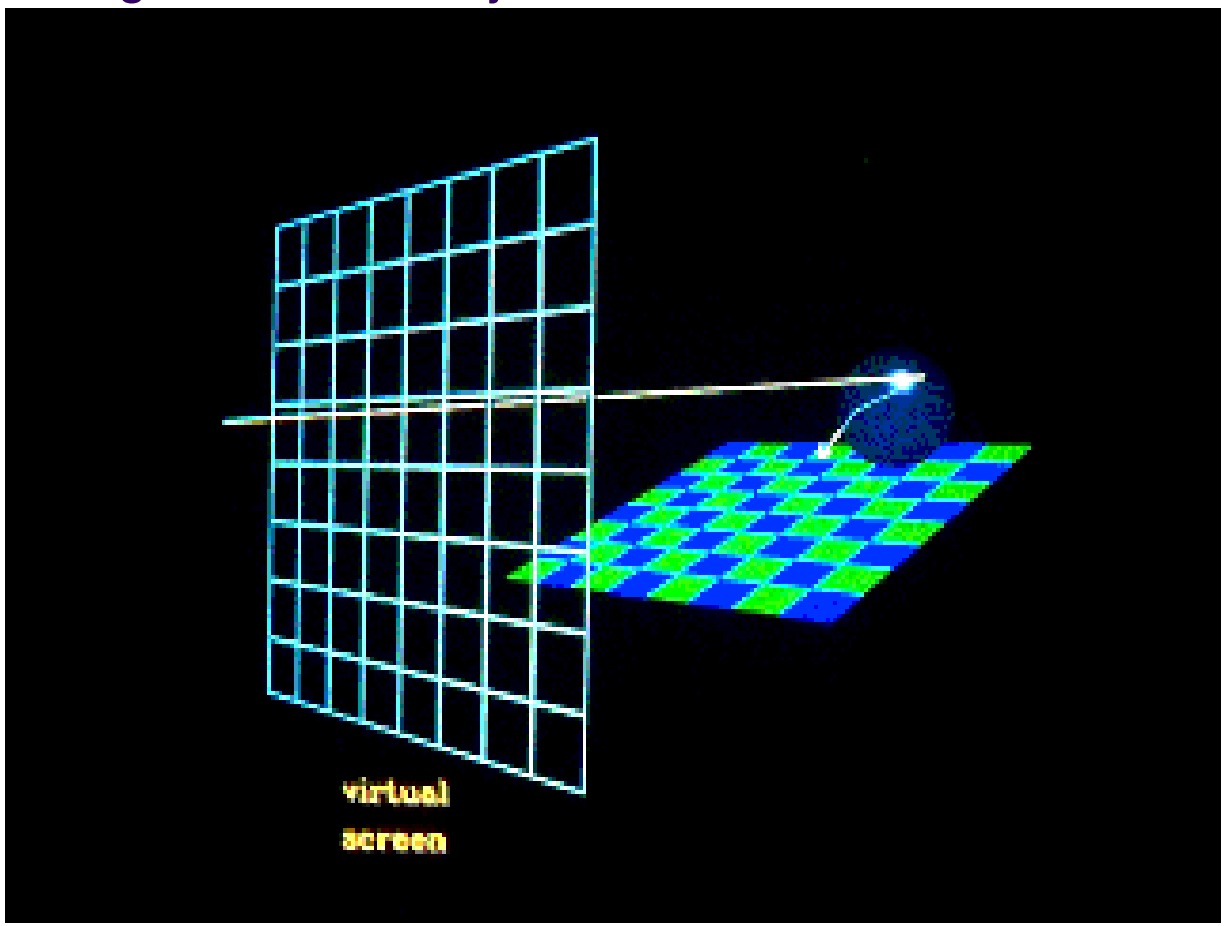
Reflection: Contribution from the reflected ray



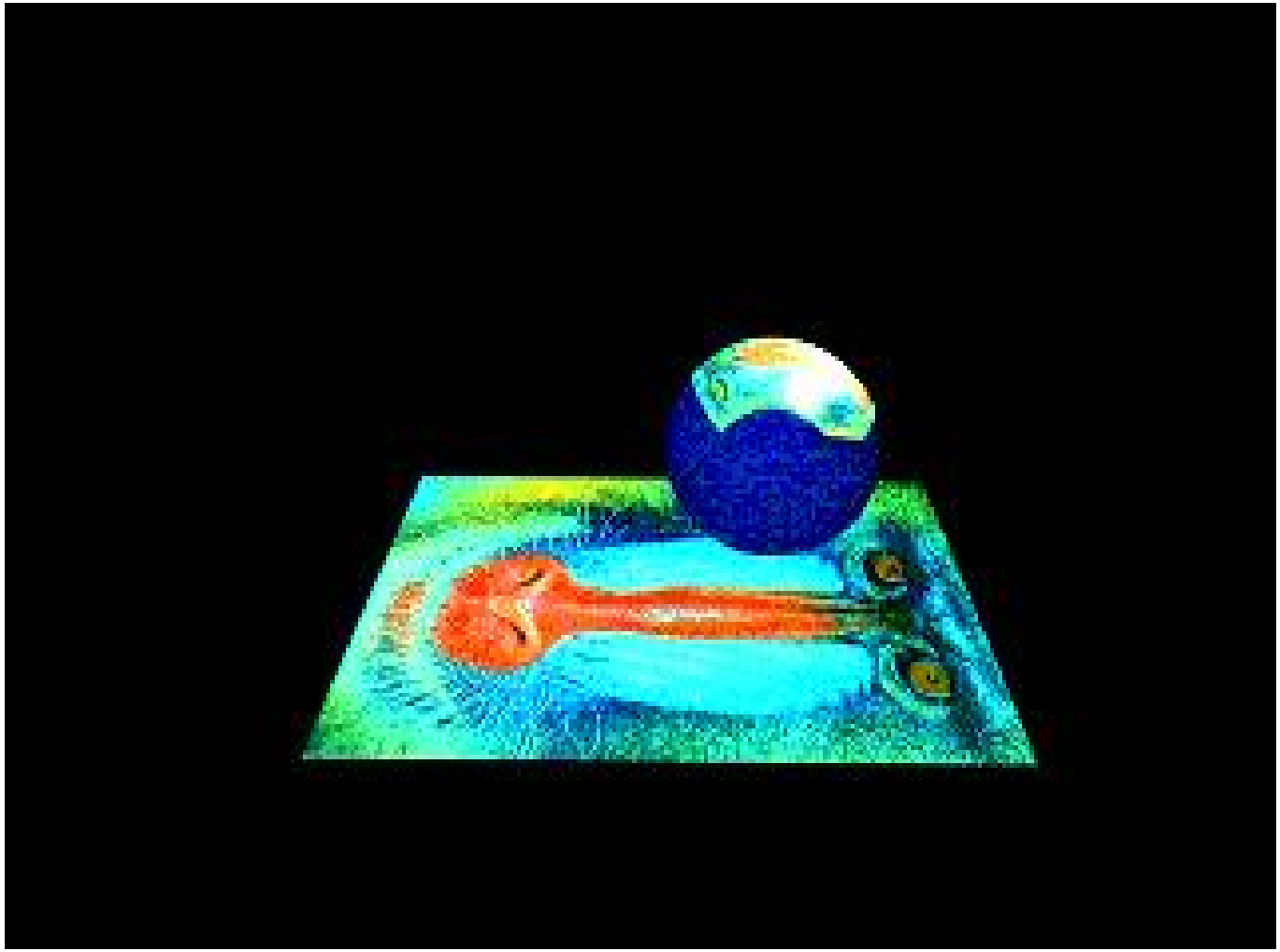
Transparency



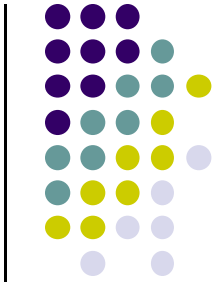
- If intersected object is transparent, transmitted ray is generated and tested against all the objects in the scene.



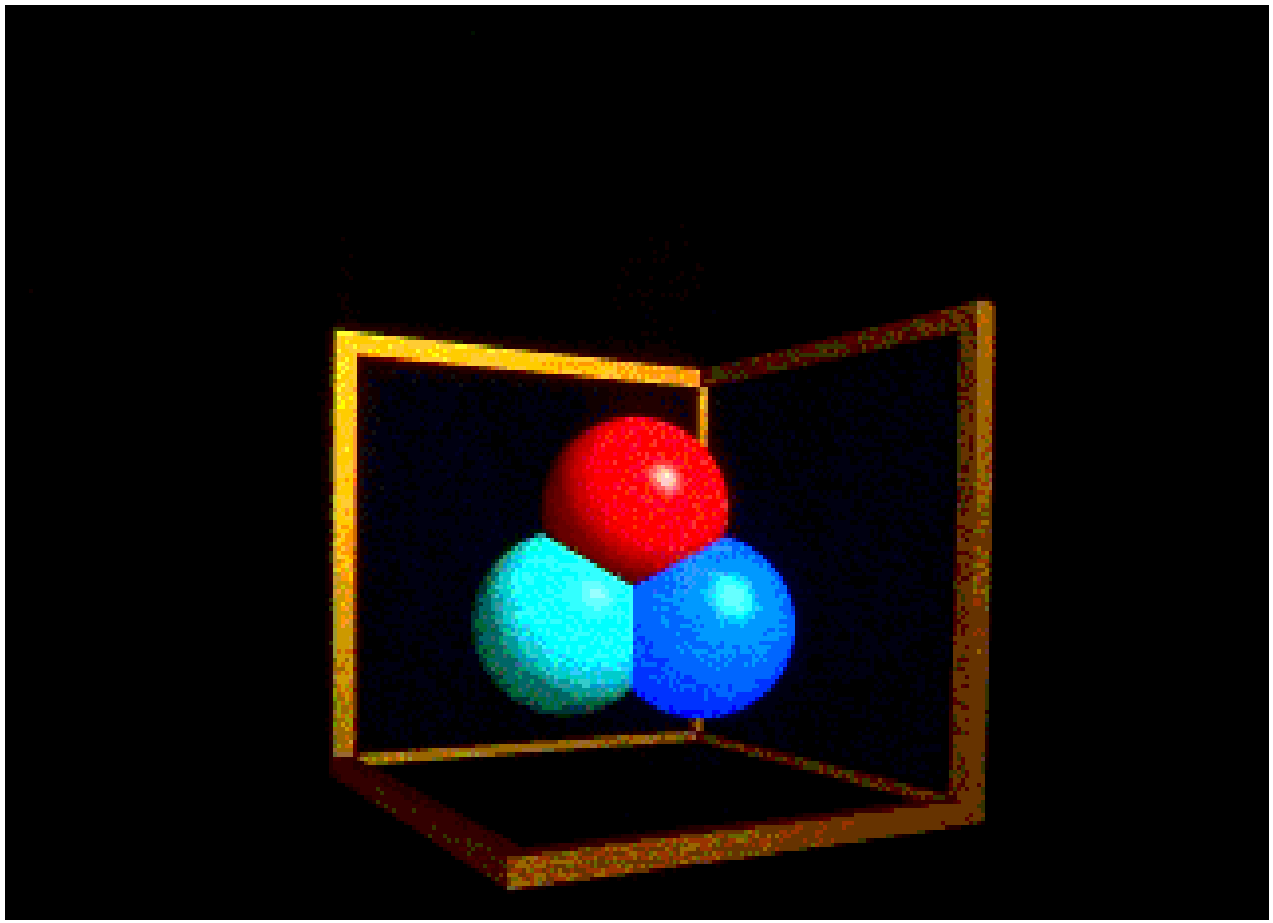
Transparency: Contribution from transmitted ray

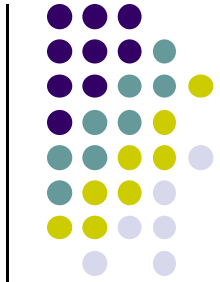


Reflected Ray: Recursion



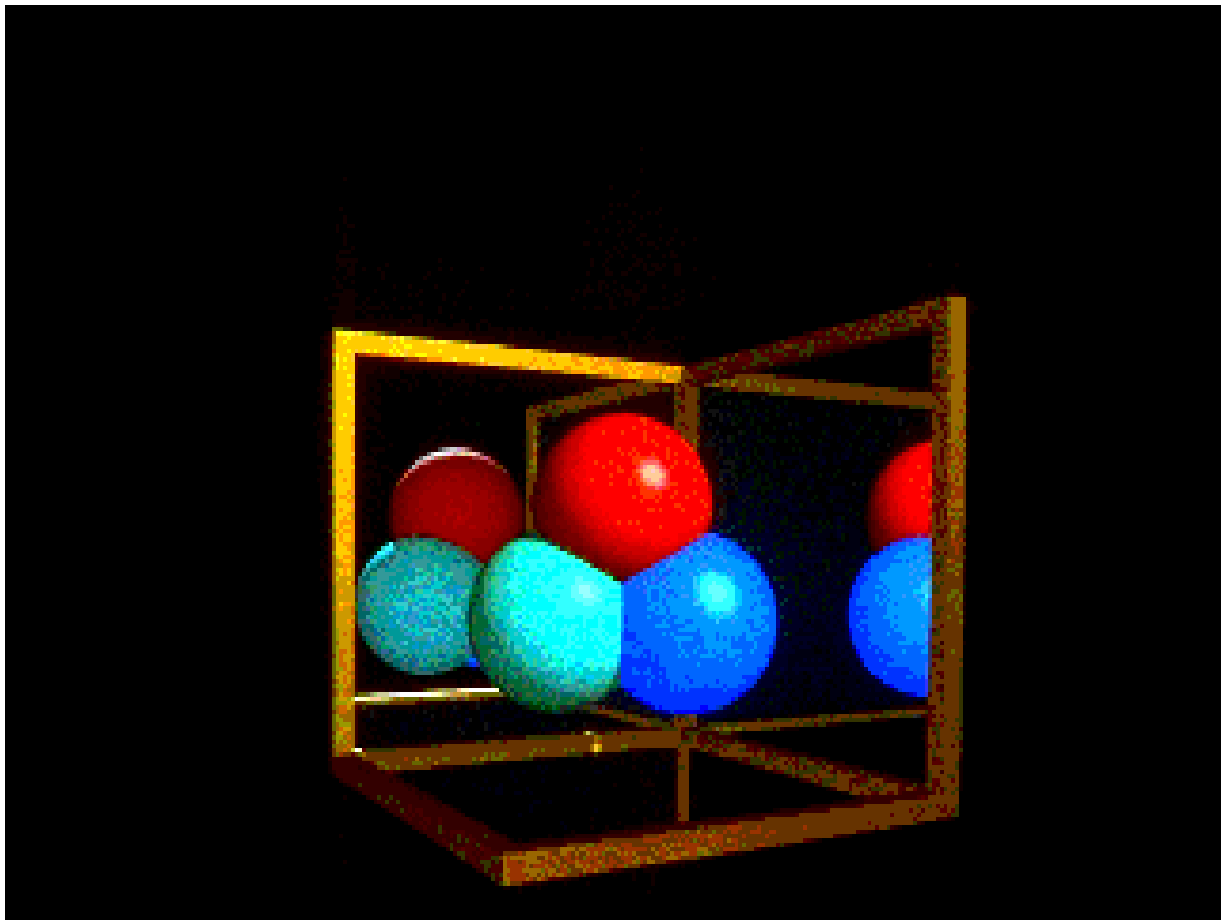
Reflected rays can generate other reflected rays that can generate other reflected rays, etc. Case A: *Scene with no reflection rays*

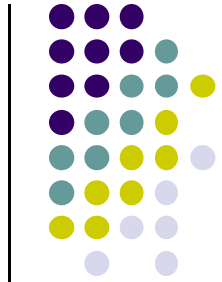




Reflected Ray: Recursion

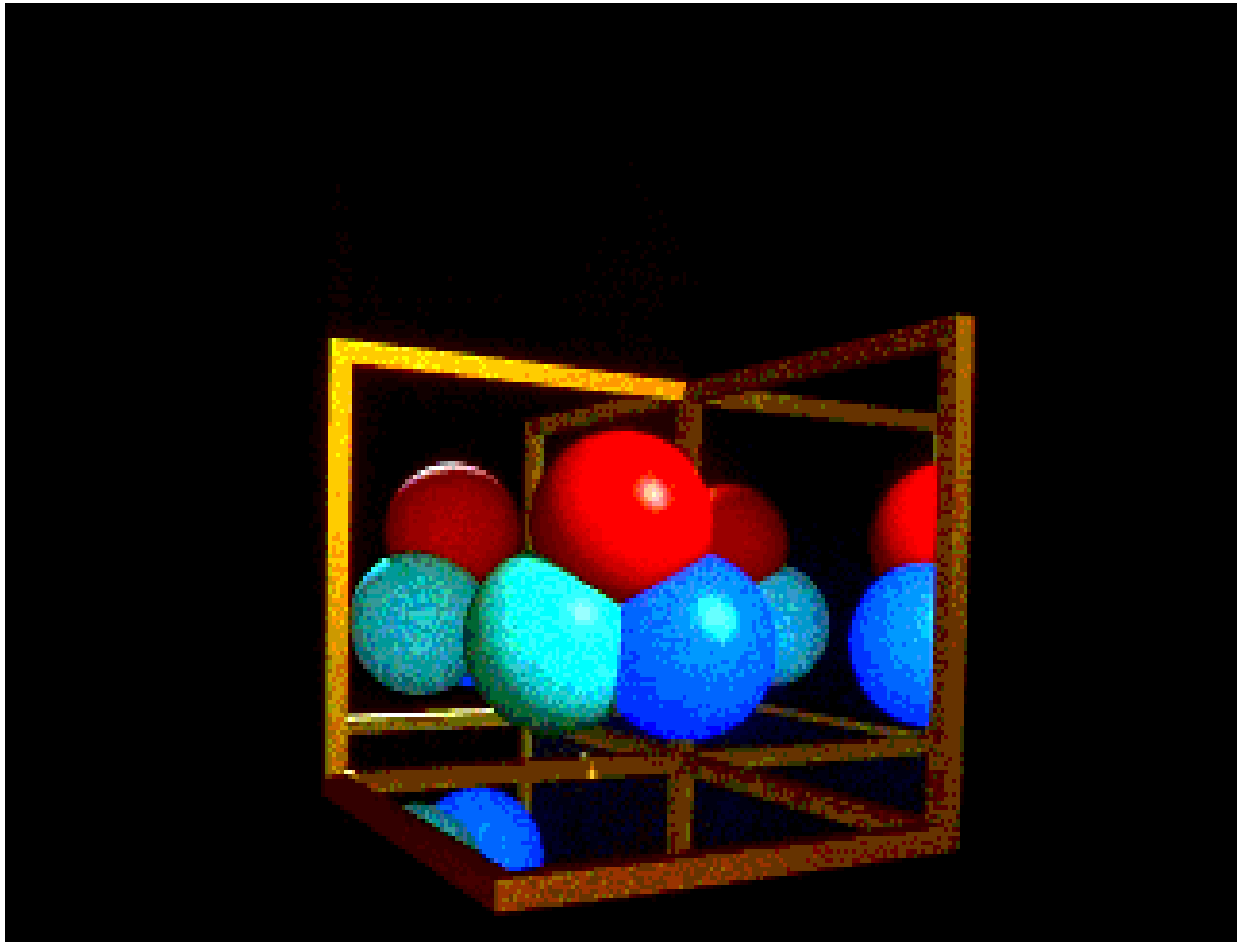
Case B: Scene with one layer of reflection





Reflected Ray: Recursion

Case C: Scene with two layers of reflection



Ray Tree

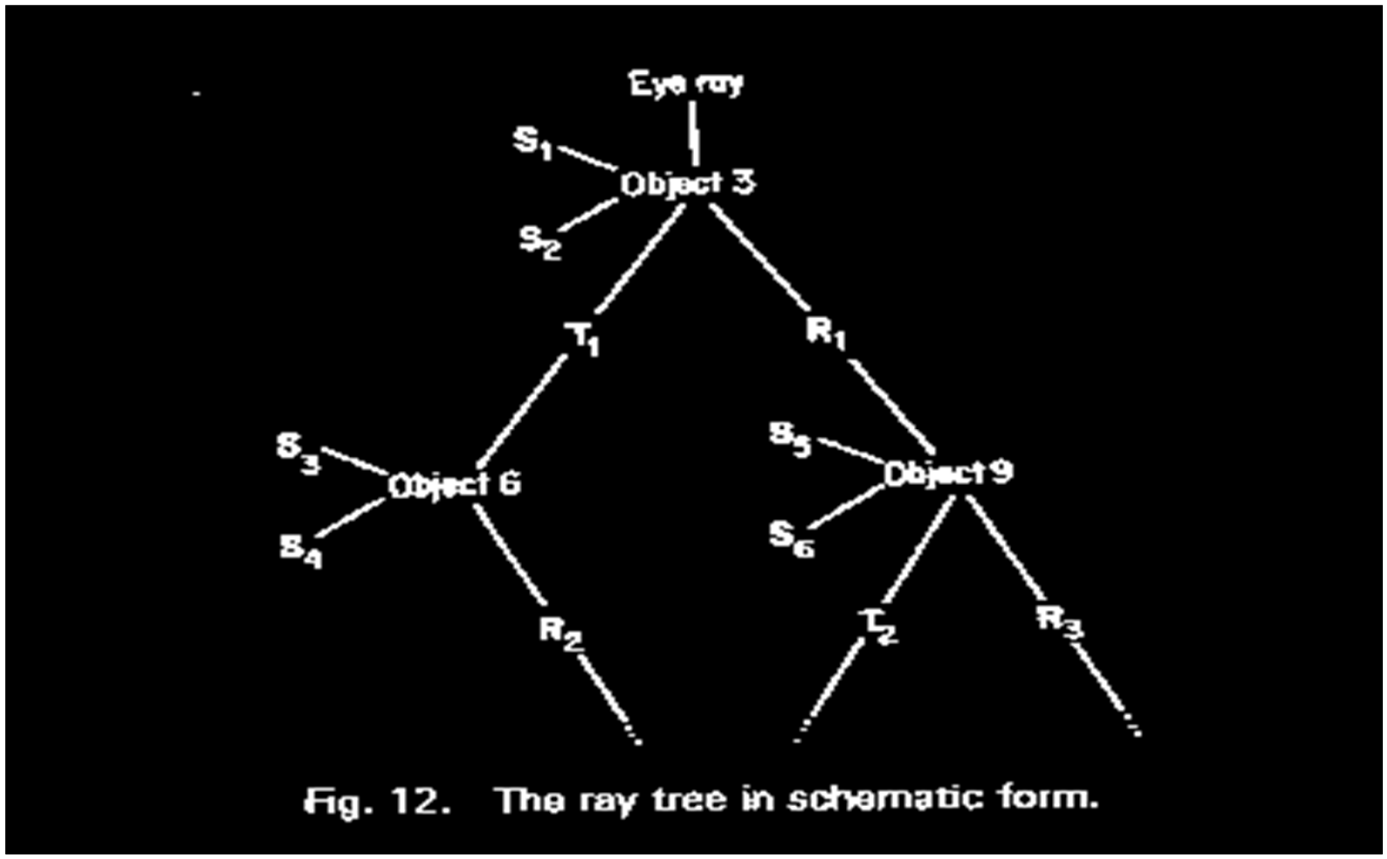
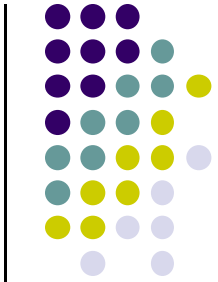


Fig. 12. The ray tree in schematic form.

- Reflective and/or transmitted rays are continually generated until ray leaves the scene without hitting any object or a preset recursion level has been reached.

Ray-Object Intersections



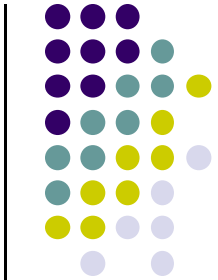
- So, express ray as equation (origin is eye, pixel determines direction)
- Define a ray as:

  `R0 = [x0, y0, z0]` - origin of ray

  `Rd = [xd, yd, zd]` - direction of ray
- then define parametric equation of ray:

  `R(t) = R0 + Rd * t` with t > 0.0
- Express all objects (sphere, cube, etc) mathematically
- Ray tracing idea:
  - put ray mathematical equation into object equation
  - determine if real solution exists.
  - Object with smallest hit time is object seen

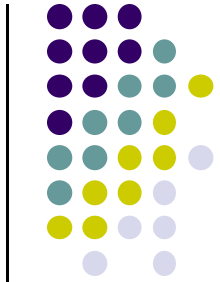
Ray-Object Intersections



- Dependent on parametric equations of object
  - Ray-Sphere Intersections
  - Ray-Plane Intersections
  - Ray-Polygon Intersections
  - Ray-Box Intersections
  - Ray-Quadric Intersections

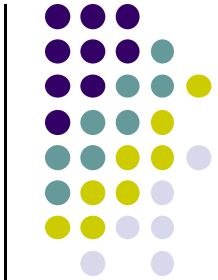
    (cylinders, cones, ellipsoids, paraboloids)

Writing a RayTracer



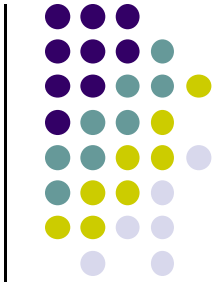
- The first step is to create the model of the objects
- One should **NOT** hardcode objects into the program, but instead use an input file.
- But for this simple ray tracer we shall hardcode our scenes into .cpp file
- Just two shapes: sphere, mesh
- The output image/file will consist of three intensity values (Red, Green, and Blue) for each pixel.

Accelerating Ray Tracing



- Ray Tracing is time-consuming because of intersection calculations
- Each intersection requires from a few (5-7) to many (15-20) floating point (fp) operations
- Example: for a scene with 100 objects and computed with a spatial resolution of 512 x 512, assuming 10 fp operations per object test there are about 250,000 X 100 X10 = 250,000,000 fps.
- Solutions:
  - Use faster machines
  - Use specialized hardware, especially parallel processors or graphics card
  - Speed up computations by using more efficient algorithms
  - Reduce the number of ray - object computations

Reducing Ray-Object Intersections

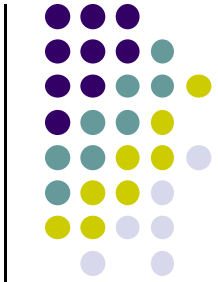


- Adaptive Depth Control: Stop generating reflected/transmitted rays when computed intensity becomes less than certain threshold.

- Bounding Volumes:
  - Enclose groups of objects in sets of hierarchical bounding volumes
  - First test for intersection with the bounding volume
  - Then only if there is an intersection, against the objects enclosed by the volume.

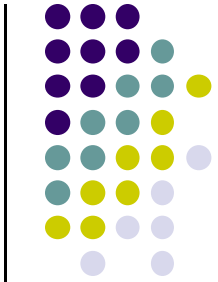
- First Hit Speed-Up: use modified Z-buffer algorithm to determine the first hit.

Writing a Ray Tracer



- Our approach:
  - Give arrangement of minimal ray tracer
  - Use that as template to explain process
- Minimal?
  - Yes! Basic framework
  - Just two object intersections
  - Minimal/no shading
- Paul Heckbert (CMU):
  - Ran ray tracing contest for years
  - Wrote ray tracer that fit on back of his business card

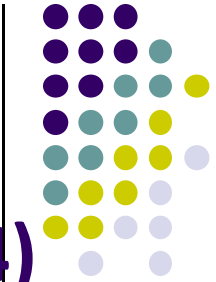
Pseudocode for Ray Tracer



- Basic idea

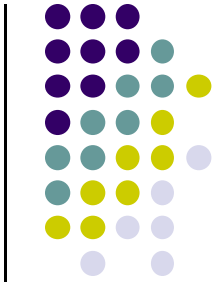
```
color Raytracer{
    for(each pixel direction){
      determine first object in this pixel direction
      calculate color shade
      return shade color
  }
}
```

More Detailed Ray Tracer Pseudocode (fig 12.4)

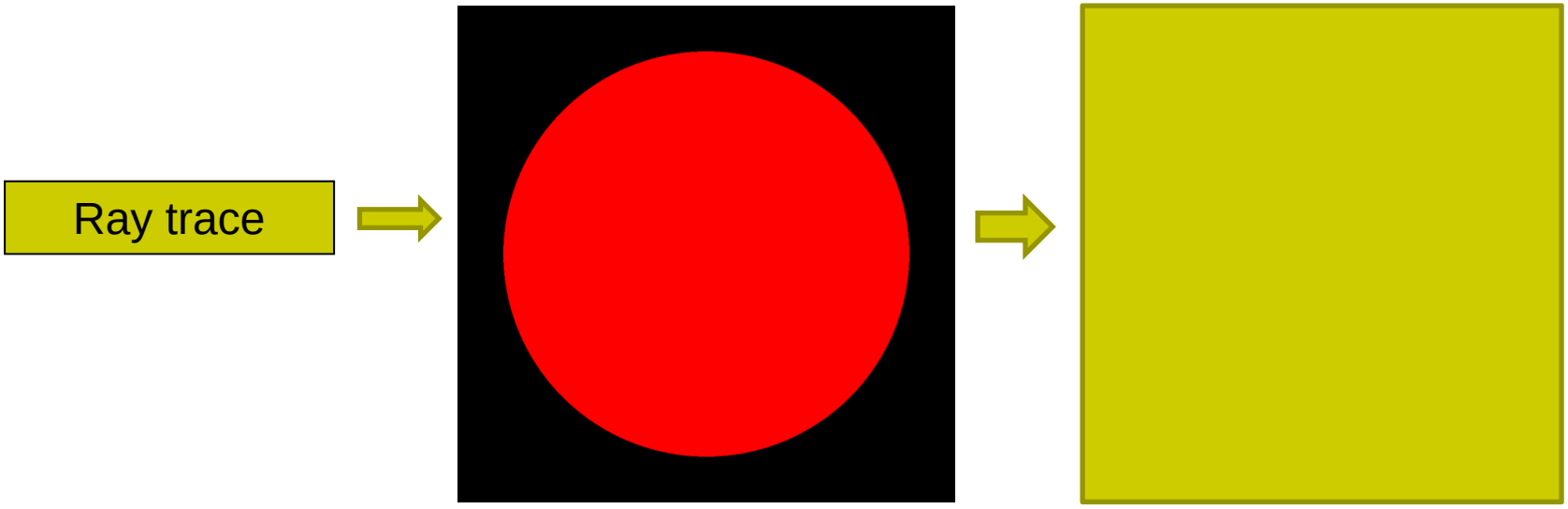


```
Define the objects and light sources in the scene
Set up the camera
For(int r = 0; r < nRows; r++){
  for(int c = 0; c < nCols; c++){
      1. Build the rc-th ray
      2. Find all object intersections with rc-th ray
      3. Identify closest object intersection
      4. Compute the "hit point" where the ray hits the
          object, and normal vector at that point
      5. Find  color of light to eye along ray
      6. Set rc-th pixel to this color
  }
}
```

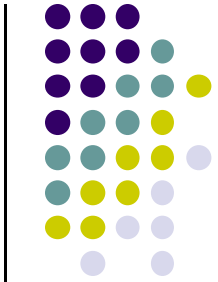
Rendering?



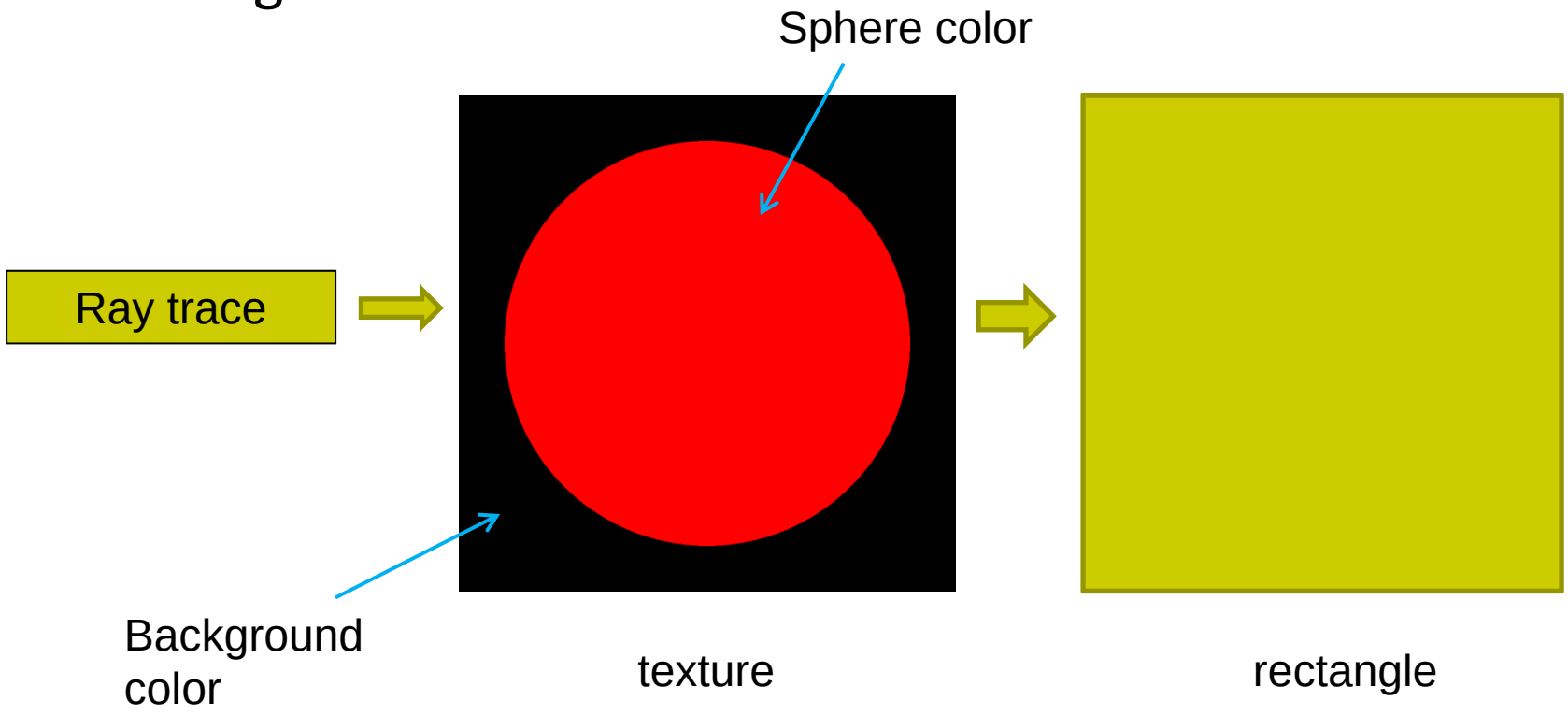
- Imagine big rectangle (2 triangles) facing camera
- Ray trace to texture image[N][N][3]
- Texture map image onto rectangle



Rendering?



- Declare two colors for texture
  - Sphere color
  - Background color

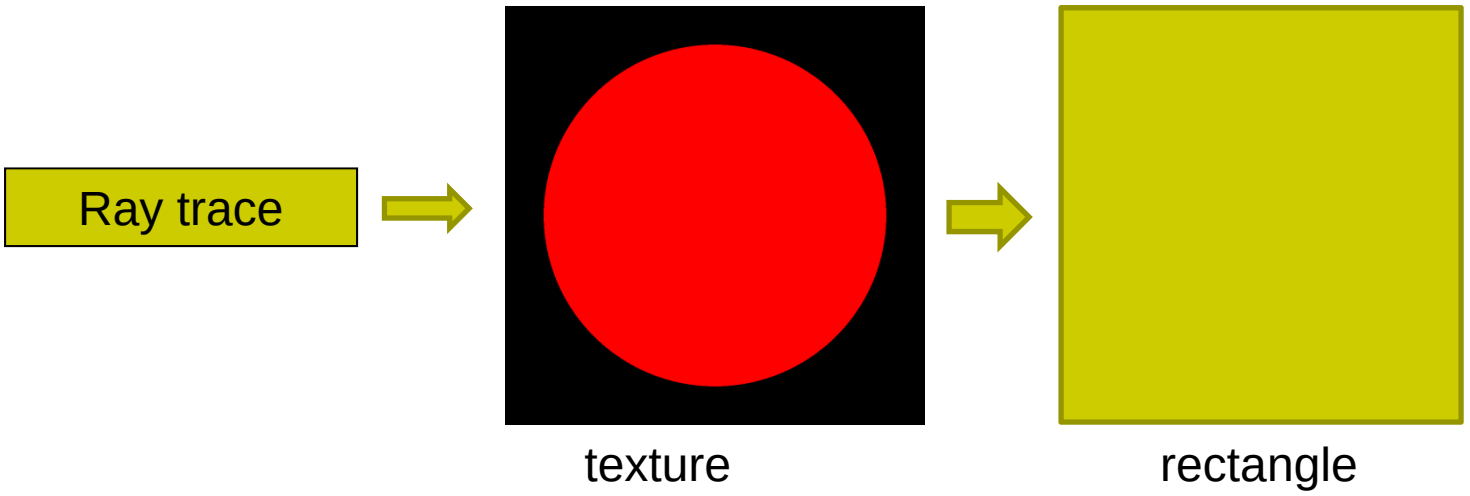


Map texture

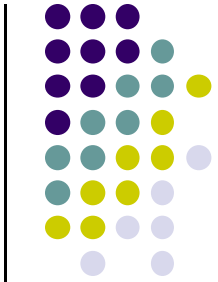
- Declare polygon as two triangles

```
point4 points[6]
= {point4(0.0, 0.0, 0.0, 1.0, point4(0.0, 1.0, 0.0, 1.0),
   point4(1.0, 1.0, 0.0, 1.0, point4(1.0, 1.0, 0.0, 1.0),
   point4(1.0, 0.0, 0.0, 1.0, point4(0.0, 0.0, 0.0, 1.0)}

Glfloat tex_coord[6][2] = {{0.0, 0.0}, {0.0, 1.0}, {1.0, 1.0},
                           {1.0, 1.0}, {1.0, 0.0}, {0.0, 0.0}},
```

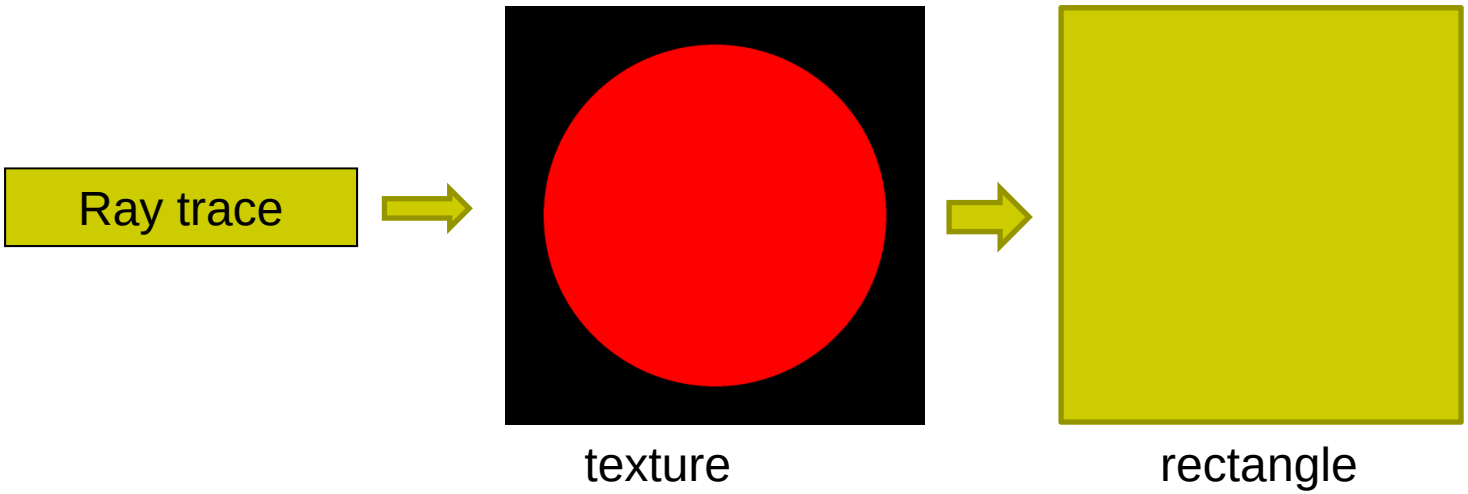


Rendering texture

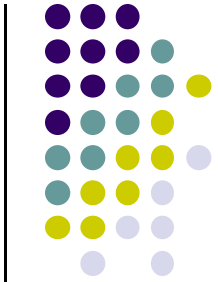


- Generate image using ray tracing
- Move image, rectangle to GPU
- Texture map as usual

(See section 9.8.5 of text. Mandelbrot example 9.2 on book website)

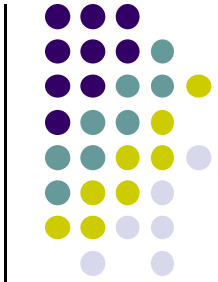


Ray Tracer Pseudocode



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

Setting RC-th pixel to Calculated Color

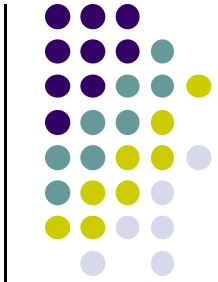


- Simply write into appropriate location of texture

```
image[i][j][0] = r;
image[i][j][0] = g;
image[i][j][0] = b;
```

- Note: r, g, b is either color of sphere if pixel covers sphere, or background color
- But ray tracing can take time.. minutes, days, weeks!! ☺?
- Use notion of blocksize to speedup ray tracing

References



- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition, Chapter 12