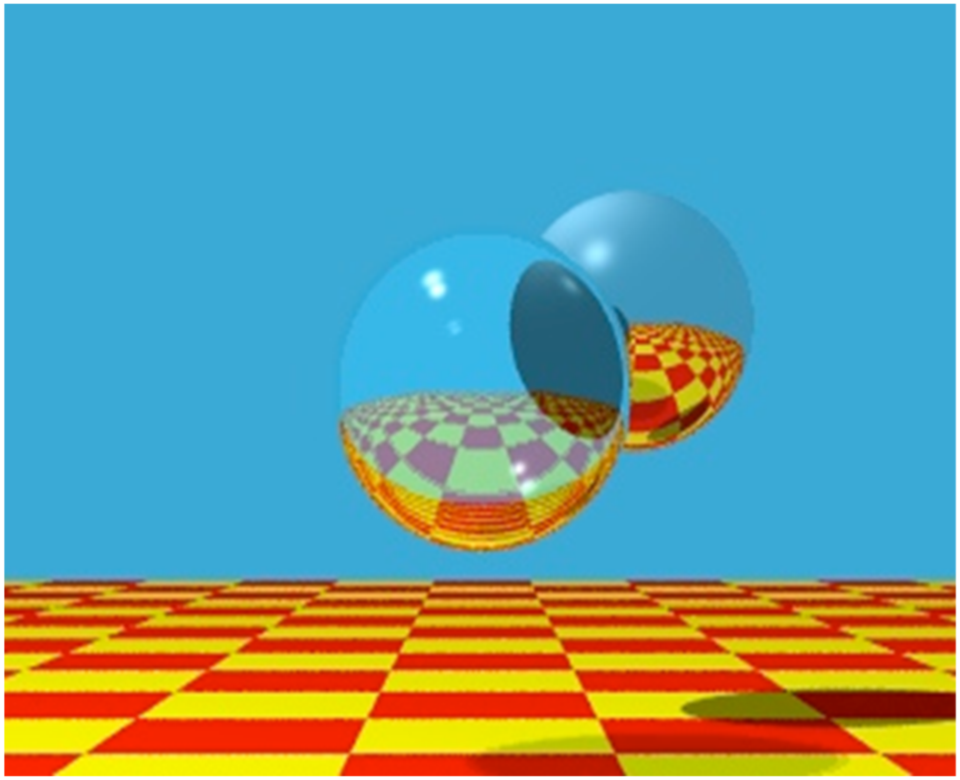
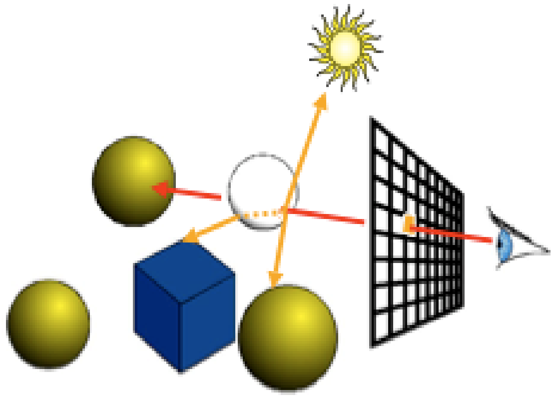
Interactive Computer Graphics: Lecture 11

Ray tracing (cont.)

Ray tracing - Summary

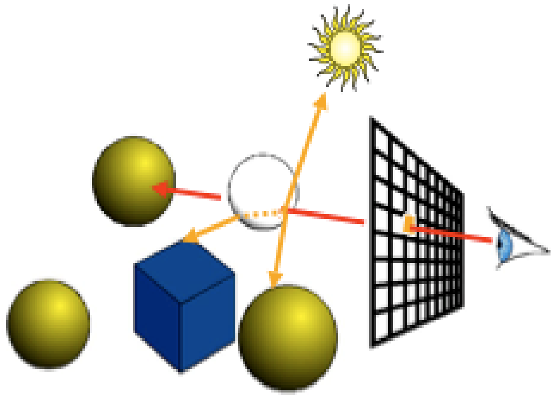


Ray tracing - Summary



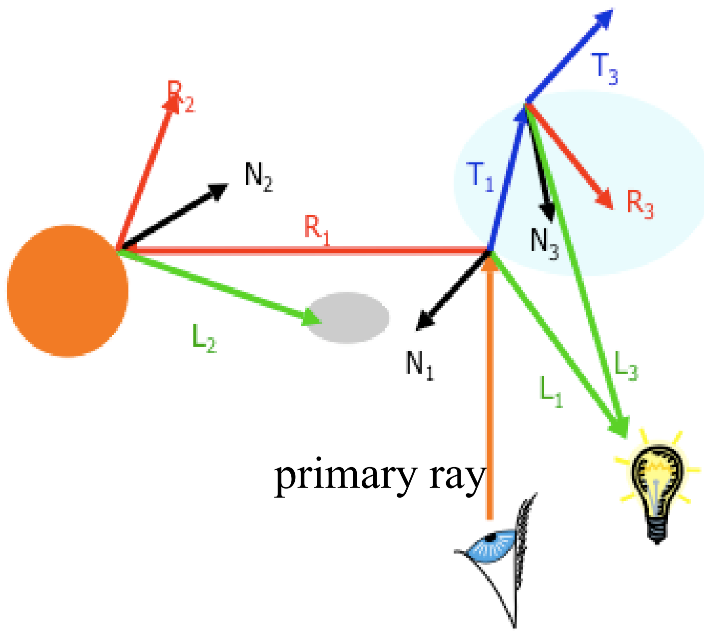
```
trace ray
      Intersect all objects
      color = ambient term
      For every light
            cast shadow ray
            col += local shading term
      If mirror
            col += k_refl * trace reflected ray
      If transparent
            col += k_trans * trace transmitted ray
```

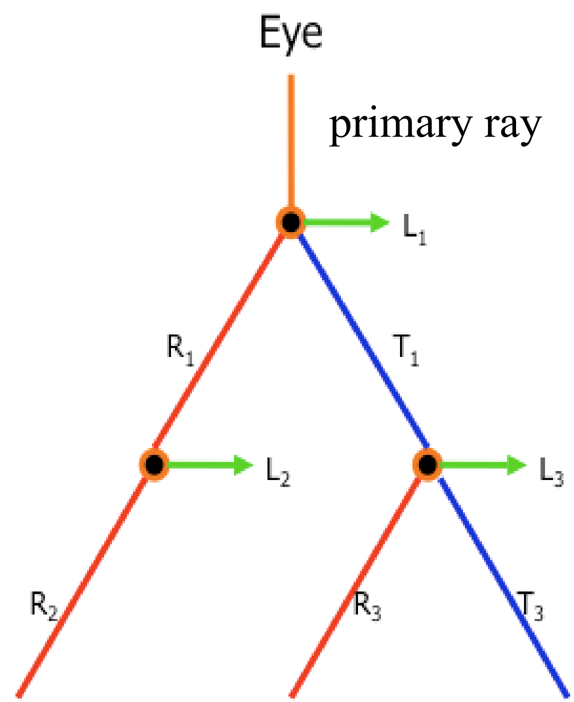
Ray tracing - Summary



```
trace ray
      Intersect all objects
      color = ambient term
      For every light
            cast shadow ray
            col += local shading term
      If mirror
            col += k_refl * trace reflected ray
      If transparent
            col += k_trans * trace transmitted ray
```

Ray tracing - Summary





R_i reflected ray

L_i shadow ray

T_i transmitted (refracted) ray

secondary rays

Graphi

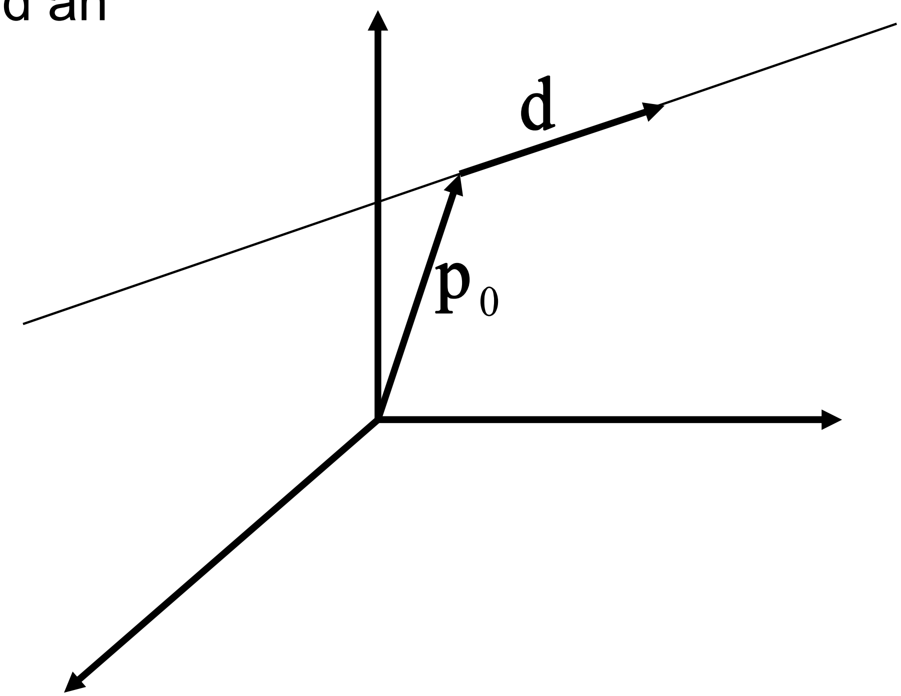
Intersection calculations

- For each ray we must calculate all possible intersections with each object inside the viewing volume
- For each ray we must find the nearest intersection point
- We can define our scene using
  - Solid models
    - sphere
    - cylinder
  - Surface models
    - plane
    - triangle
    - polygon

Rays

- Rays are parametric lines
- Rays can be defined an
  - origin $\mathbf{p_0}$
  - direction $\mathbf{d}$
- Equation of ray:

$$\mathbf{p}(\mu) = \mathbf{p}_0 + \mu\mathbf{d}$$



Ray tracing: Intersection calculations

- The coordinates of any point along each primary ray are given by:

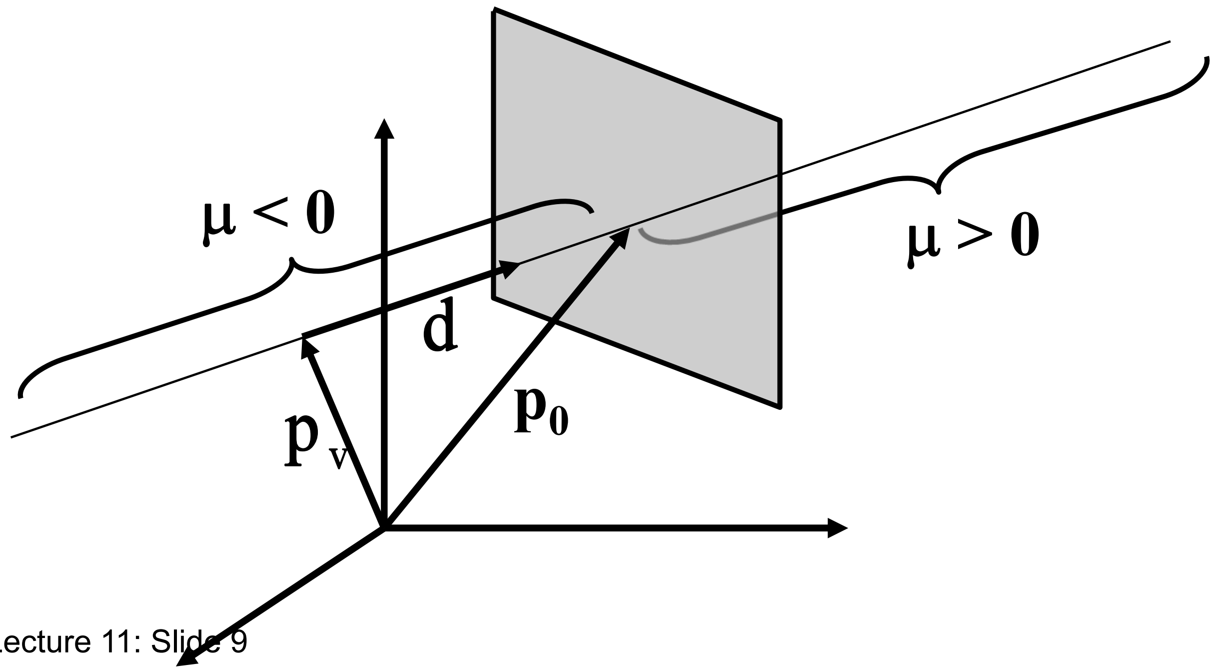
$$\mathbf{p} = \mathbf{p}_0 + \mu \mathbf{d}$$

  - $\mathbf{p_0}$ is the current pixel on the viewing plane.
  - $\mathbf{d}$ is the direction vector and can be obtained from the position of the pixel on the viewing plane $\mathbf{p_0}$ and the viewpoint $\mathbf{p_v}$:

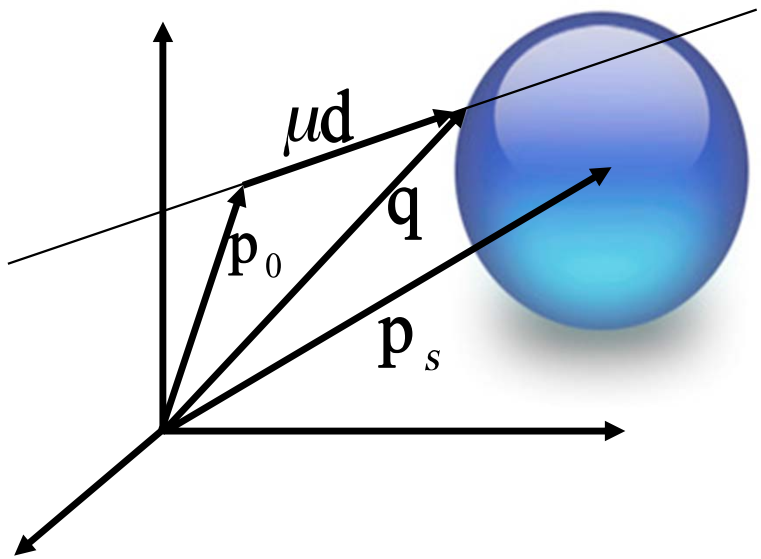
$$\mathbf{d} = \frac{\mathbf{p}_0 - \mathbf{p}_v}{|\mathbf{p}_0 - \mathbf{p}_v|}$$

Ray tracing: Intersection calculations

- The viewing ray can be parameterized by μ:
  - $\mu > 0$ denotes the part of the ray behind the viewing plane
  - $\mu < 0$ denotes the part of the ray in front of the viewing plane
  - For any visible intersection point $\mu > 0$



Intersection calculations: Spheres



- For any point on the surface of the sphere

$$\left|\mathbf{q}-\mathbf{p}_s\right|^2 - r^2 = 0$$

- where r is the radius of the sphere

Intersection calculations: Spheres

- To test whether a ray intersects a surface we can substitute for $\mathbf{q}$ using the ray equation:

$$\left|\mathbf{p}_0 + \mu\mathbf{d} - \mathbf{p}_s\right|^2 - r^2 = 0$$

- Setting $\Delta\mathbf{p} = \mathbf{p}_0 - \mathbf{p}_s$ and expanding the dot product produces the following quadratic equation:

$$\mu^2 + 2\mu(\mathbf{d} \cdot \Delta\mathbf{p}) + \left|\Delta\mathbf{p}\right|^2 - r^2 = 0$$

Intersection calculations: Spheres

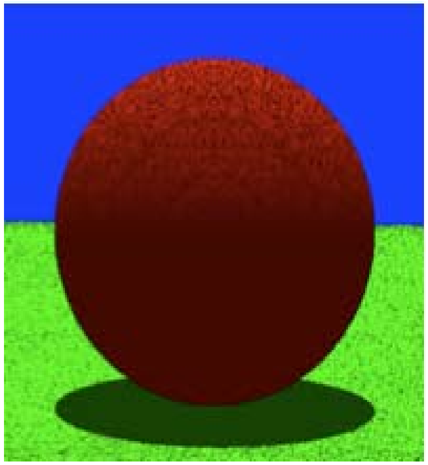
- The quadratic equation has the following solution:

$$\mu = -\mathbf{d} \cdot \Delta\mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \Delta\mathbf{p})^2 - |\Delta\mathbf{p}|^2 + r^2}$$

- Solutions:
  - if the quadratic equation has no solution, the ray does not intersect the sphere
  - if the quadratic equation has two solutions ($\mu_1 < \mu_2$):
    - μ_1 corresponds to the point at which the rays enters the sphere
    - μ_2 corresponds to the point at which the rays leaves the sphere

Precision Problems

- In ray tracing, the origin of (secondary) rays is often on the surface of objects
  - Theoretically, $\mu = 0$ for these rays
  - Practically, calculation imprecision creeps in, and the origin of the new ray is slightly beneath the surface
- Result: the surface area is shadowing itself



ε to the rescue ...

- Check if t is within some epsilon tolerance:
  - if abs(μ) < ε
    - point is on the sphere
  - else
    - point is inside/outside
  - Choose the ε tolerance empirically
- Move the intersection point by epsilon along the surface normal so it is outside of the object
- Check if point is inside/outside surface by checking the sign of the implicit (sphere etc.) equation

Intersection calculations: Cylinders

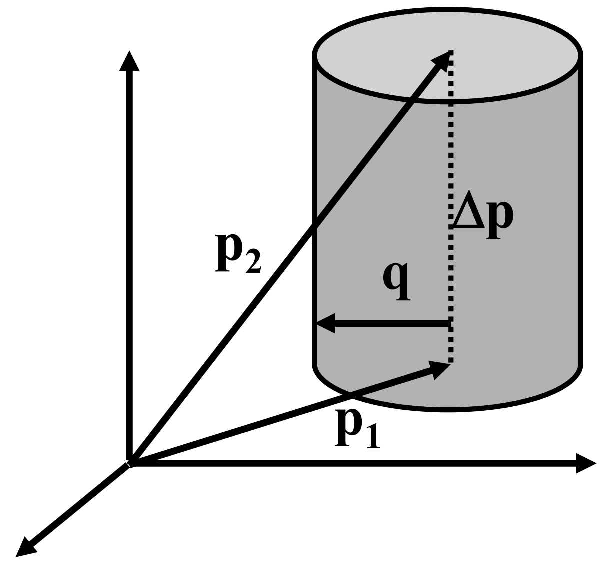
- A cylinder can be described by
  - a position vector $\mathbf{p}_1$ describing the first end point of the long axis of the cylinder
  - a position vector $\mathbf{p}_2$ describing the second end point of the long axis of the cylinder
  - a radius r
- The axis of the cylinder can be written as $\Delta\mathbf{p} = \mathbf{p}_1 - \mathbf{p}_2$ and can be parameterized by $0 \leq \alpha \leq 1$

Intersection calculations: Cylinders

- To calculate the intersection of the cylinder with the ray:

$$\mathbf{p}_1 + \alpha \Delta \mathbf{p} + \mathbf{q} = \mathbf{p}_0 + \mu \mathbf{d}$$

- Since $\mathbf{q} \cdot \Delta \mathbf{p} = 0$ we can write



$$\alpha(\Delta \mathbf{p} \cdot \Delta \mathbf{p}) = \mathbf{p}_0 \cdot \Delta \mathbf{p} + \mu \mathbf{d} \cdot \Delta \mathbf{p} - \mathbf{p}_1 \cdot \Delta \mathbf{p}$$

Intersection calculations: Cylinders

- Solving for α yields:

$$\alpha = \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}$$

- Substituting we obtain:

$$\mathbf{q} = \mathbf{p}_0 + \mu\mathbf{d} - \mathbf{p}_1 - \left(\frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}} \right) \Delta\mathbf{p}$$

Intersection calculations: Cylinders

- Using the fact that $\mathbf{q} \cdot \mathbf{q} = r^2$ we can use the same approach as before to the quadratic equation for μ:

$$r^2 = \left(\mathbf{p}_0 + \mu \mathbf{d} - \mathbf{p}_1 - \left(\frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu \mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}} \right) \Delta\mathbf{p} \right)^2$$

  - If the quadratic equation has no solution: **<u>no intersection</u>**
  - If the quadratic equation has two solutions: **<u>intersection</u>**

Intersection calculations: Cylinders

- Assuming that $\mu 1 \leq \mu 2$ we can determine two solutions:

$$\alpha_1 = \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu_1 \mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}$$

$$\alpha_2 = \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu_2 \mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}$$

- If the value of α_1 is between 0 and 1 the intersection is on the outside surface of the cylinder
- If the value of α_2 is between 0 and 1 the intersection is on the inside surface of the cylinder

Intersection calculations: Plane

- Objects are often described by geometric primitives such as
  - triangles
  - planar quads
  - planar polygons
- To test intersections of the ray with these primitives we must test whether the ray will intersect the plane defined by the primitive

Intersection calculations: Plane

- The intersection of a ray with a plane is given by

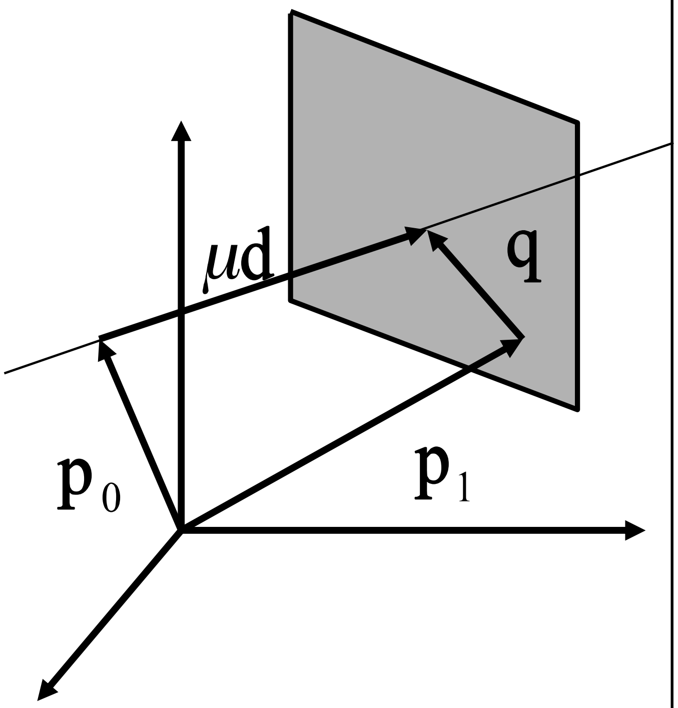
$$\mathrm{p}_1 + \mathrm{q} = \mathrm{p}_0 + \mu \mathrm{d}$$

- where $\mathbf{p}_1$ is a point in the plane. Subtracting $\mathbf{p}_1$ and multiplying with the normal of the plane $\mathbf{n}$ yields:

$$\mathrm{q} \cdot \mathrm{n} = 0 = (\mathrm{p}_0 - \mathrm{p}_1) \cdot \mathrm{n} + \mu \mathrm{d} \cdot \mathrm{n}$$

- Solving for μ yields:

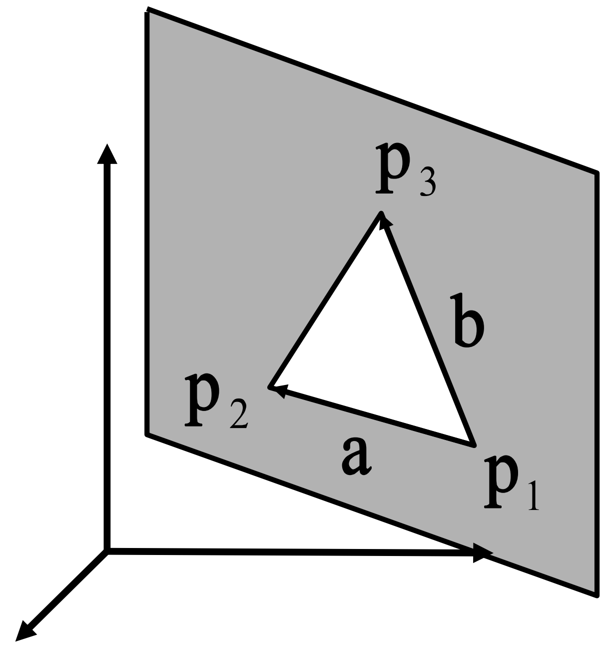
$$\mu = -\frac{(\mathrm{p}_0 - \mathrm{p}_1) \cdot \mathrm{n}}{\mathrm{d} \cdot \mathrm{n}}$$



Intersection calculations: Triangles

- To calculate intersections:
  - test whether triangle is front facing
  - test whether plane of triangle intersects ray
  - test whether intersection point is inside triangle
- If the triangle is front facing:

$$\mathbf{d} \cdot \mathbf{n} < 0$$



Intersection calculations: Triangles

- To test whether plane of triangle intersects ray
  - calculate equation of the plane using

$$\mathbf{p}_2 - \mathbf{p}_1 = \mathbf{a}$$

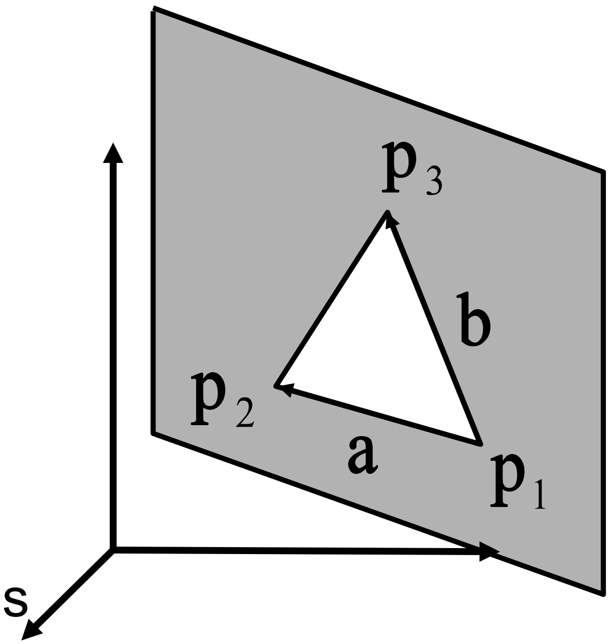
$$\mathbf{p}_3 - \mathbf{p}_1 = \mathbf{b}$$

  - calculate intersections with plane as before

$$\mathbf{n} = \mathbf{a} \times \mathbf{b}$$

- To test whether intersection point is inside triangle:

$$\mathbf{q} = \alpha\mathbf{a} + \beta\mathbf{b}$$



Intersection calculations: Triangles

- A point is inside the triangle if

$$0 \leq \alpha \leq 1$$
$$0 \leq \beta \leq 1$$
$$\alpha + \beta \leq 1$$

- Calculate α and β by taking the dot product with a and b:

$$\alpha = \frac{(\mathbf{b} \cdot \mathbf{b})(\mathbf{q} \cdot \mathbf{a}) - (\mathbf{a} \cdot \mathbf{b})(\mathbf{q} \cdot \mathbf{b})}{(\mathbf{a} \cdot \mathbf{a})(\mathbf{b} \cdot \mathbf{b}) - (\mathbf{a} \cdot \mathbf{b})^2}$$

$$\beta = \frac{\mathbf{q} \cdot \mathbf{b} - \alpha(\mathbf{a} \cdot \mathbf{b})}{\mathbf{b} \cdot \mathbf{b}}$$

Intersection calculations: Triangles

- algorithm by **Möller and Trumbore**:
- translate the origin of the ray and then change the base of that vector which yields parameter vector (t, u, v)
- t is the distance to the plane in which the triangle lies
- (u, v) are barycentric coordinates inside the triangle
- plane equation need not be computed on the fly nor be stored

Intersection calculations: Triangles

```
bool triangle_intersection( vec3 v1,
                           vec3 v2,
                           vec3 v3, // Triangle vertices
                           vec3 origin,  //Ray origin
                           vec3 ray_dir,  //Ray direction
                           float* out )

//Find vectors for two edges sharing v1
vec3 edge1 = v2-v1;
vec3 edge2 = v3-v1;

//Begin calculating determinant - also used to calculate u parameter
vec3 p = cross(ray_dir, edge2)

//if determinant is near zero, ray lies in plane of triangle or ray
//is parallel to plane of triangle
float det = dot(edge1, p);

if(det > -EPSILON && det < EPSILON)
return false;

float inv_det = 1.f / det;

//calculate distance from v1 to ray origin
vec3 t = origin-v1;
```


26

Intersection calculations: Triangles

```
//Calculate u parameter and test bound
float u = dot(t, p) * inv_det;

//The intersection lies outside of the triangle
if(u < 0.f || u > 1.f)
return false;

//Prepare to test v parameter
vec3 q = cross(t,edge1);

float v = dot(ray_dir, q) * inv_det;
//The intersection lies outside of the triangle
if(v < 0.f || u + v  > 1.f)
return false;

float mu = dot(edge2, q) * inv_det;

if(t > EPSILON) { //ray intersection
*out = mu;
return true;
```

Ray tracing: Pros and cons

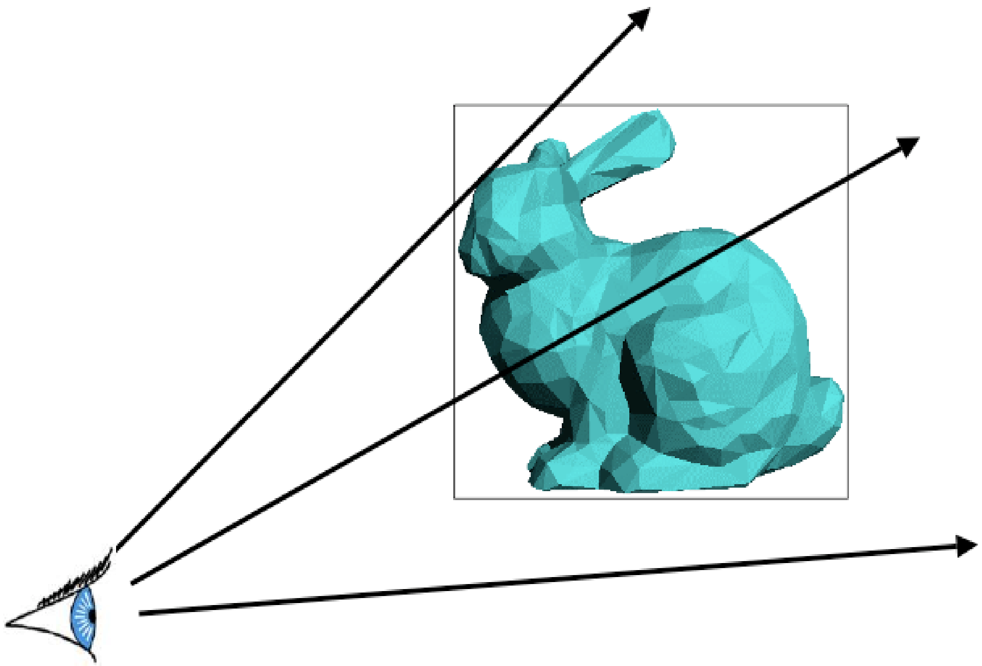
- Pros:
  - Easy to implement
  - Extends well to global illumination
    - shadows
    - reflections / refractions
    - multiple light bounces
    - atmospheric effects
- Cons:
  - Speed! (seconds per frame, not frames per second)

Speedup Techniques

- Why is ray tracing slow? How to improve?
  - Too many objects, too many rays
  - Reduce ray-object intersection tests
  - Many techniques!

Acceleration of Ray Casting

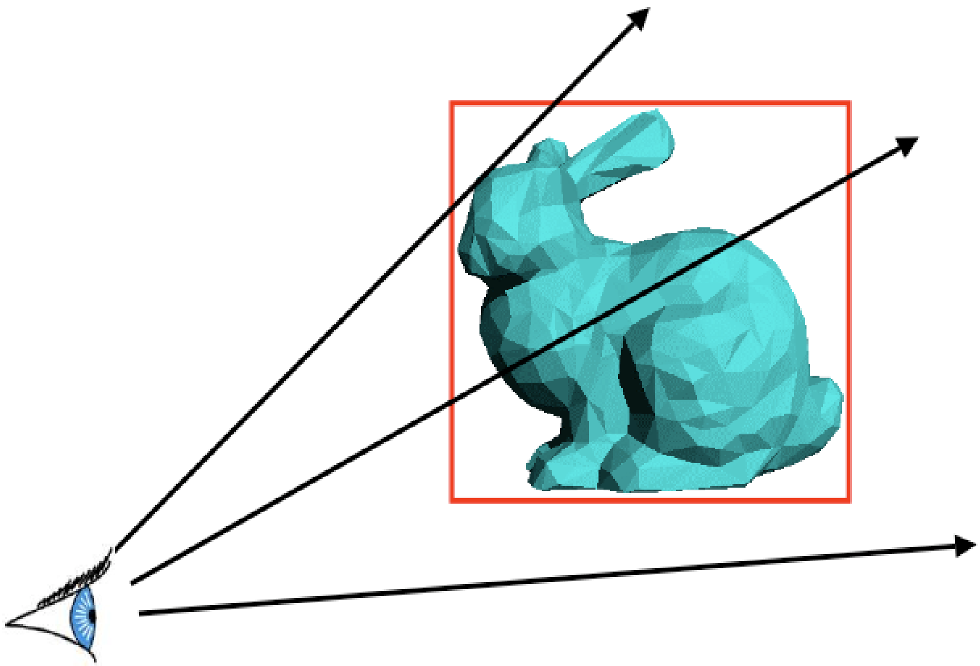
- Goal: Reduce the number of ray/primitive intersections



Graphics Lecture 11: Slide 30

Conservative Bounding Region

- First check for an intersection with a conservative bounding region
- Early reject

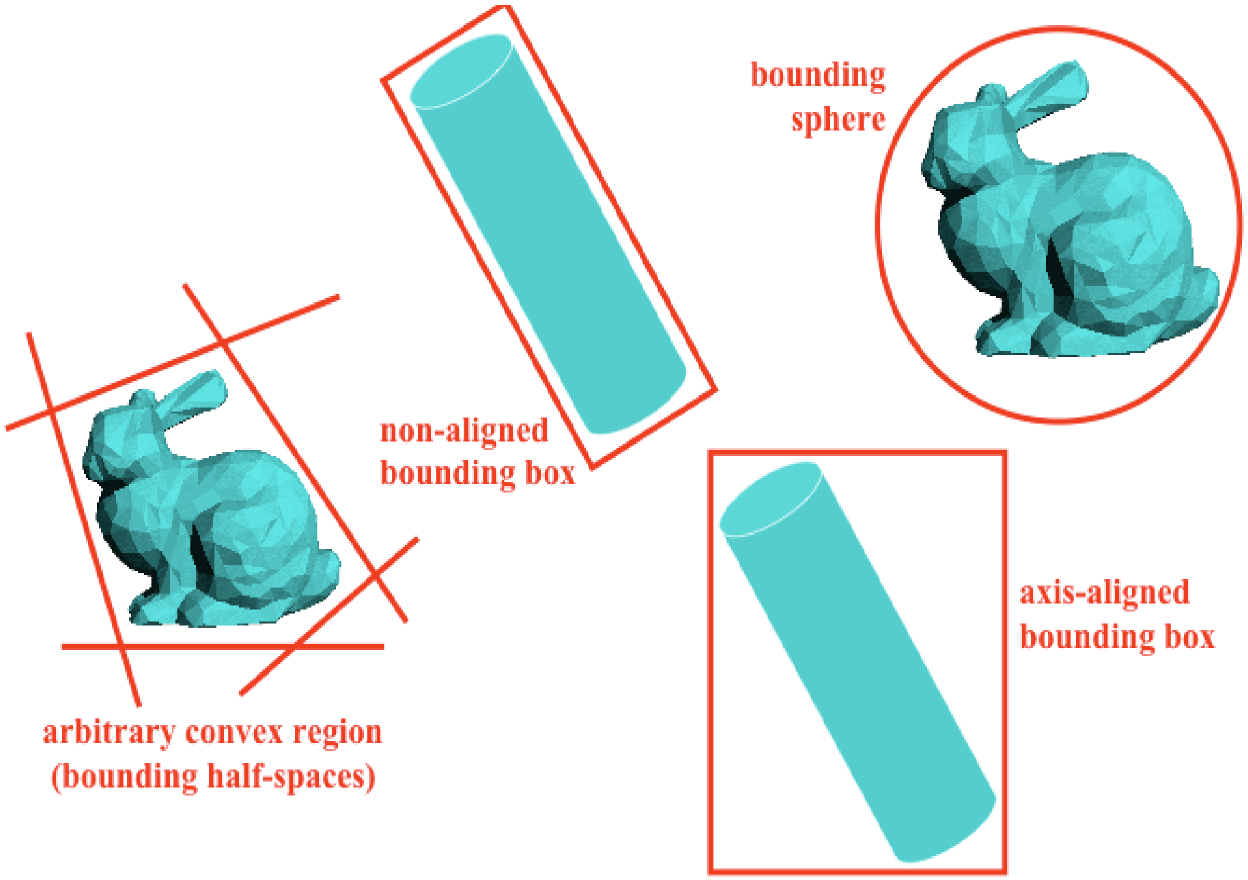


Graphics Lecture 11: Slide 31

Bounding Regions

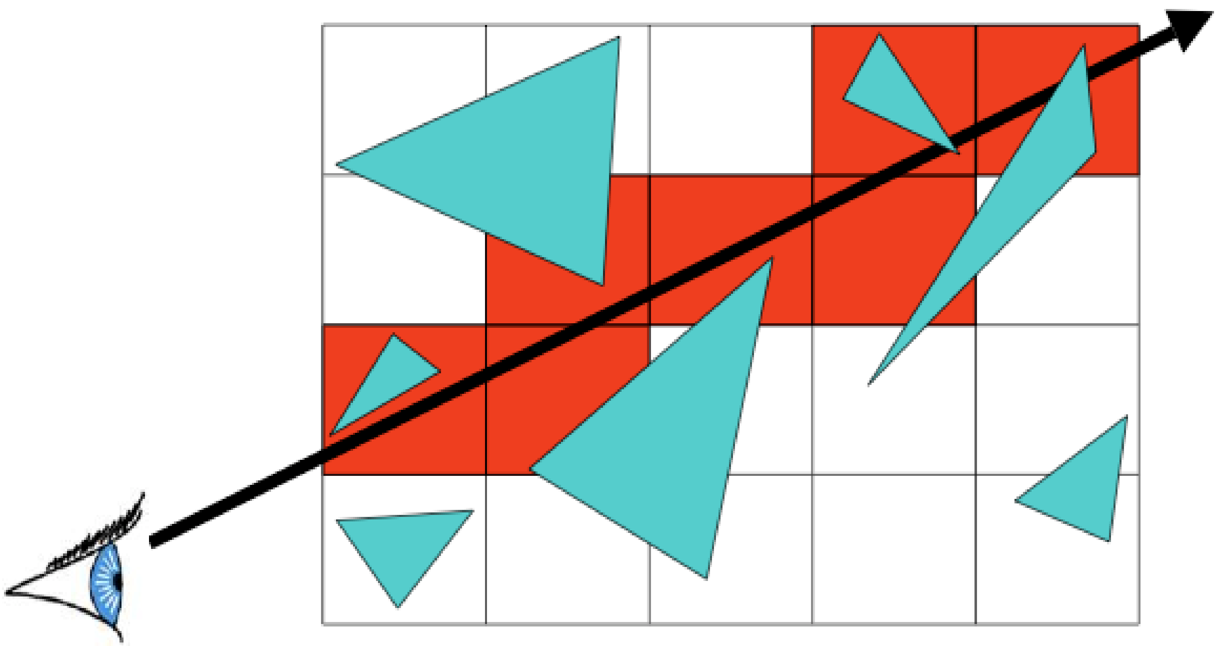
- What makes a good bounding region?

Conservative Bounding Regions



Graphics Lecture 11: Slide 33

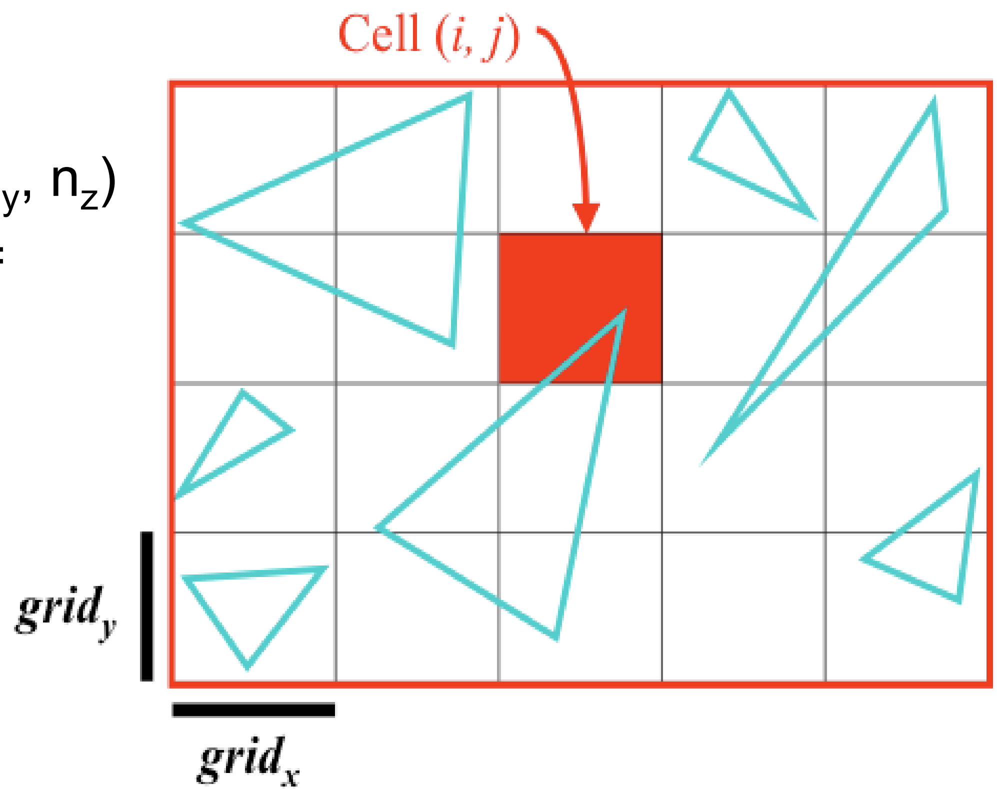
Regular Grid



34

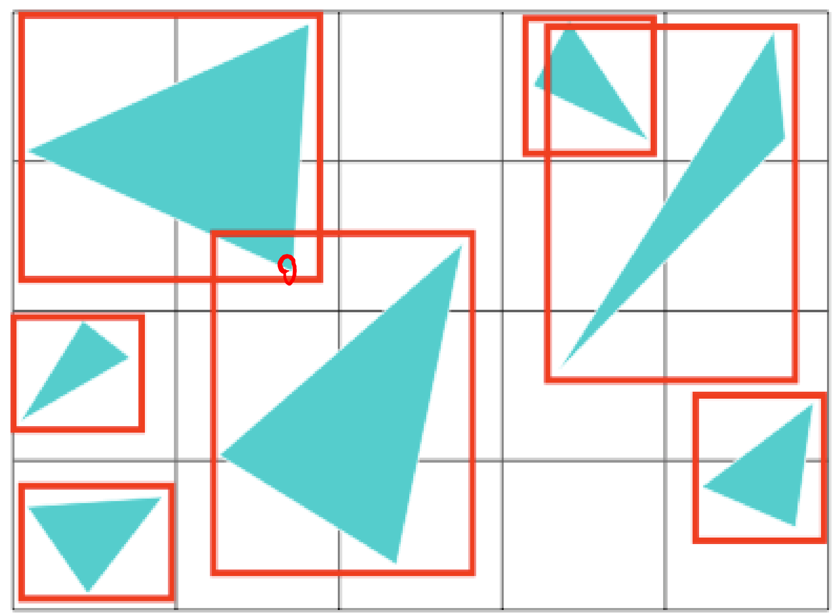
Create Grid

- Find bounding box of scene
- Choose grid resolution (n_x, n_y, n_z)
- $grid_x$ need not = $grid_y$



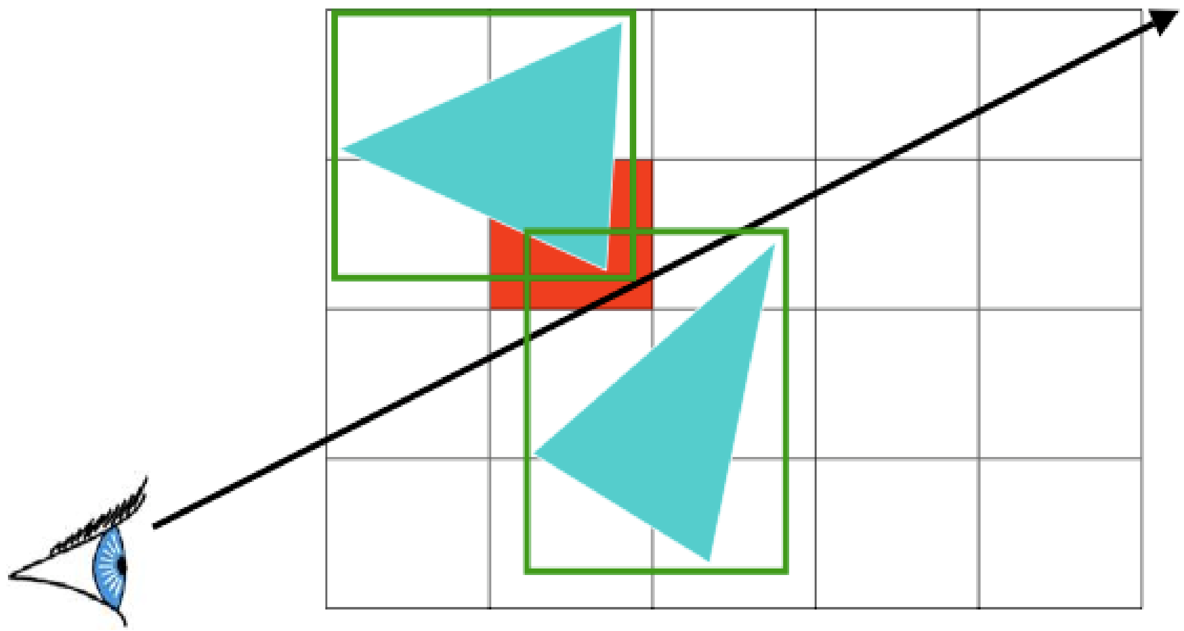
Insert Primitives into Grid

- Primitives that overlap multiple cells?
- Insert into multiple cells (use pointers)



For Each Cell Along a Ray

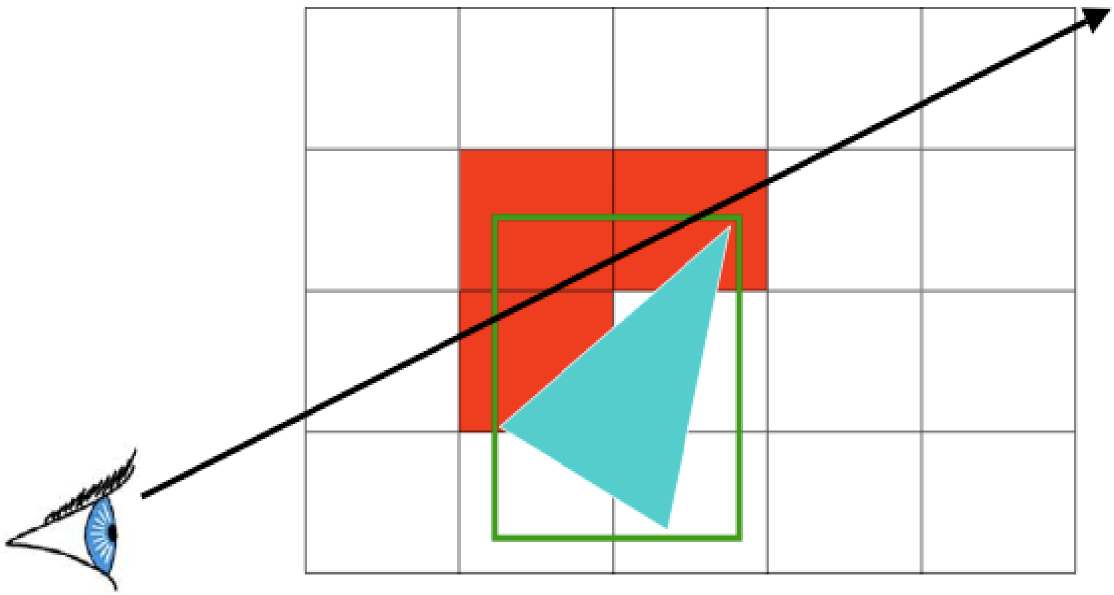
- Does the cell contain an intersection?
  - Yes: return closest intersection
  - No: continue



Graphics Lecture 11: Slide 37

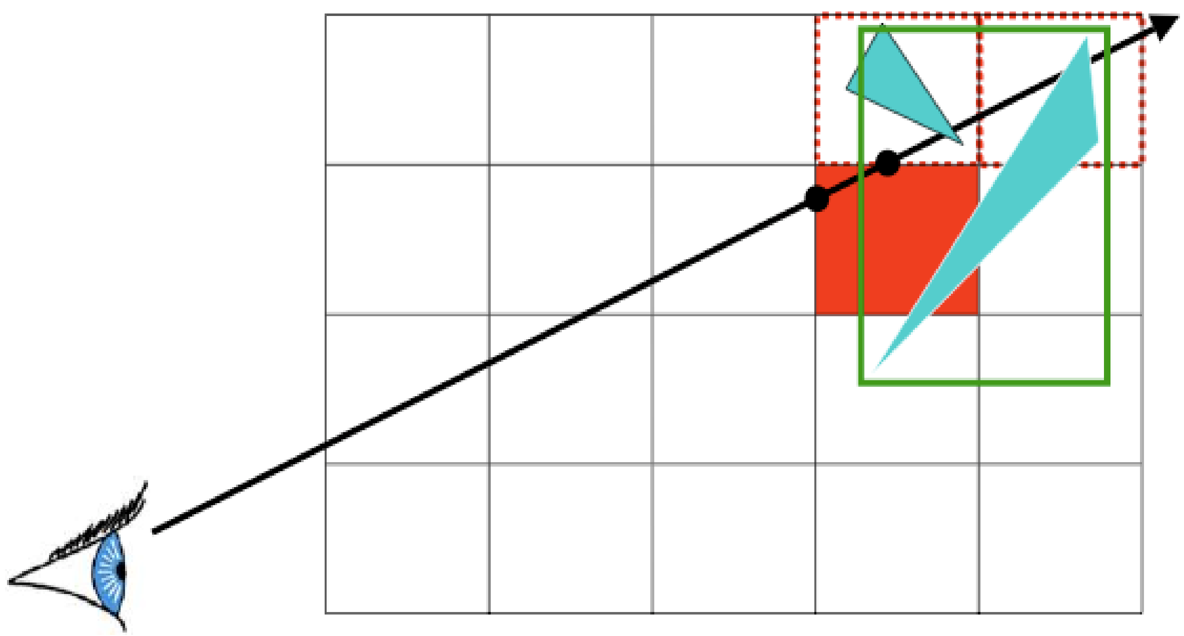
Preventing Repeated Computation

- Perform the computation once, "mark" the object
- Don't re-intersect marked objects



Don't Return Distant Intersections

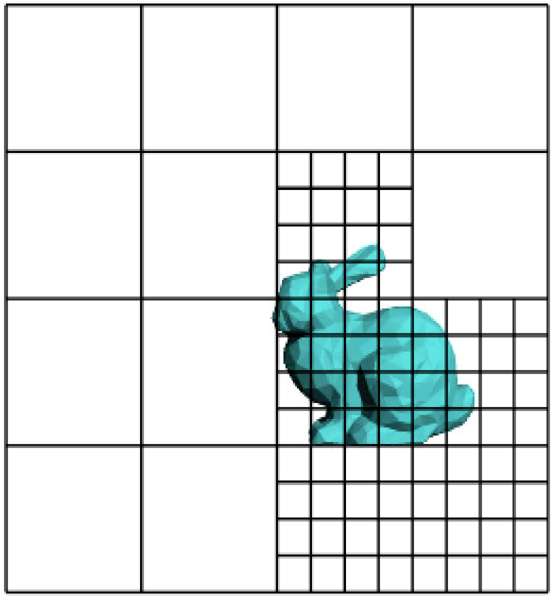
- If intersection t is not within the cell range, continue (there may be something closer)



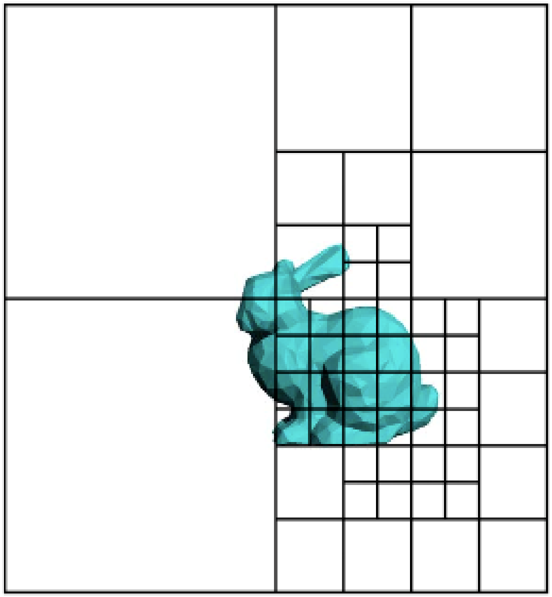
Graphics Lecture 11: Slide 39

Adaptive Grids

- Subdivide until each cell contains no more than n elements, or maximum depth d is reached



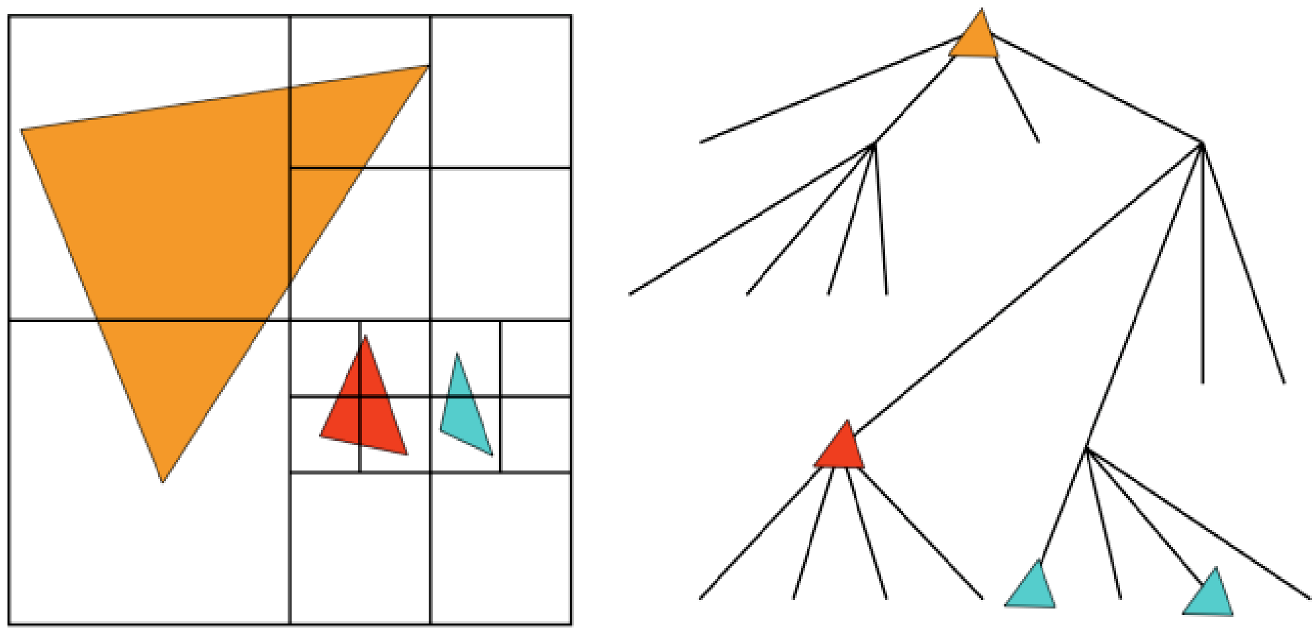
Nested Grids



Octree/(Quadtree)

Primitives in an Adaptive Grid

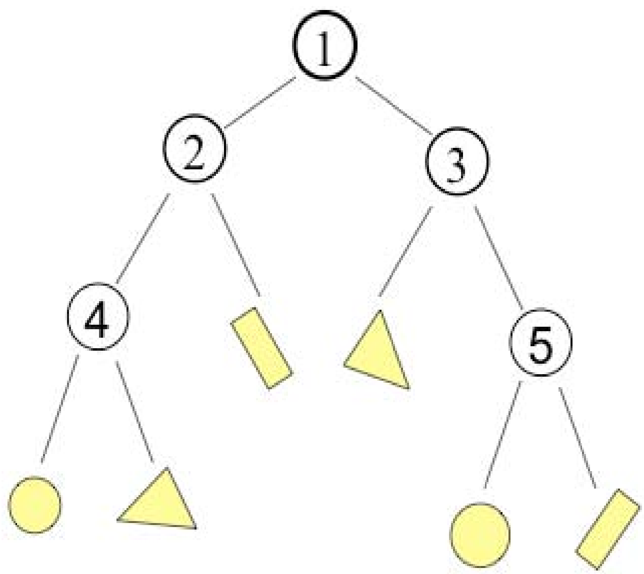
- Can live at intermediate levels, or be pushed to lowest level of grid

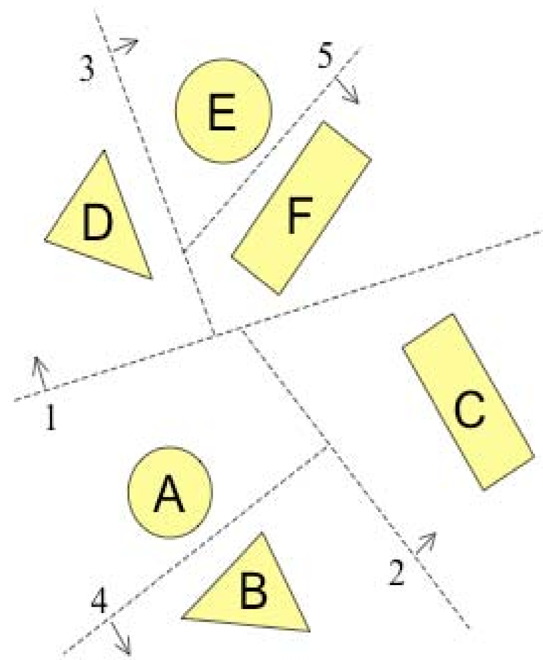


Octree/(Quadtree)

Binary Space Partition (BSP) Tree

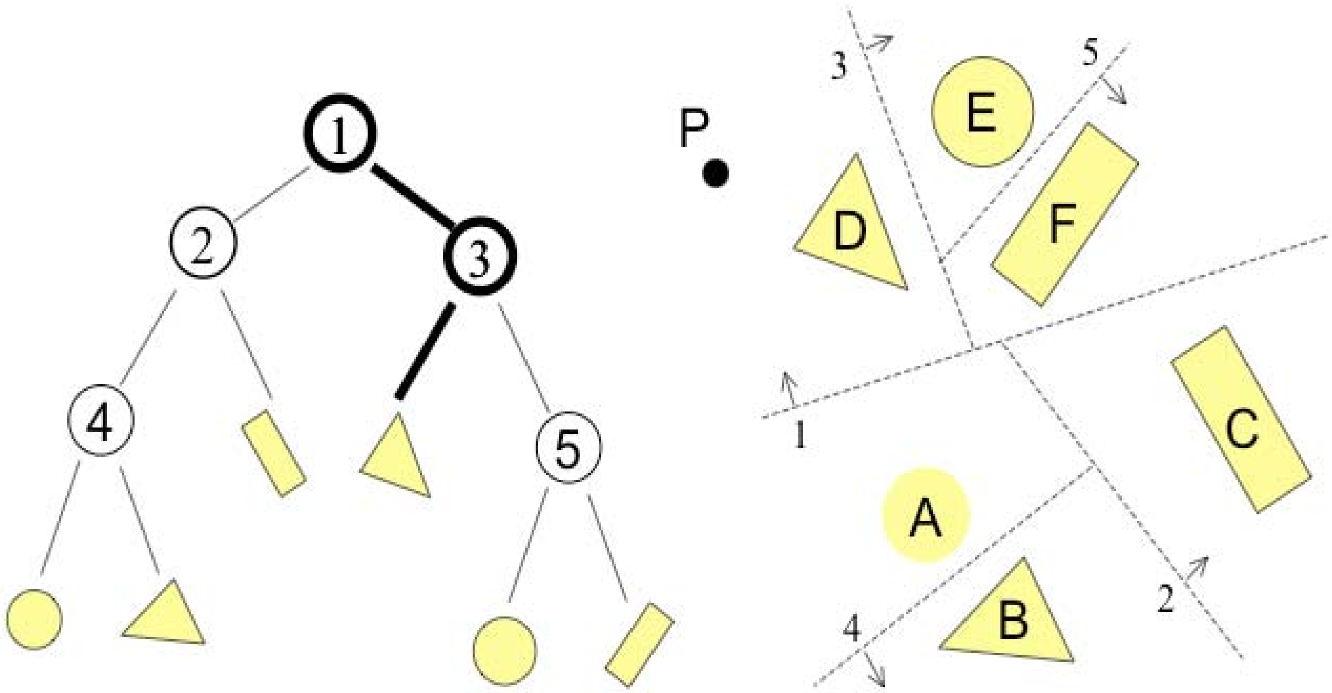
- Recursively partition space by planes
- Every cell is a convex polyhedron





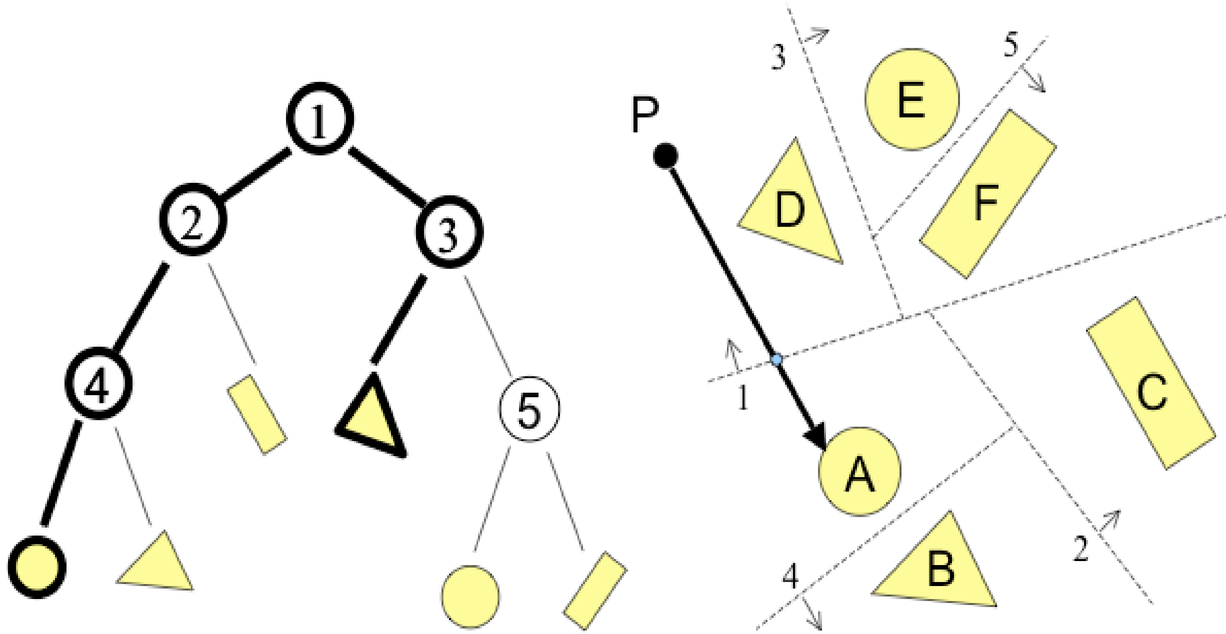
Binary Space Partition (BSP) Tree

- Simple recursive algorithms
- Example: point finding



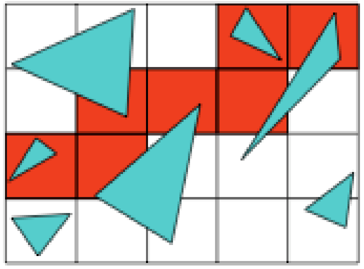
Binary Space Partition (BSP) Tree

- Trace rays by recursion on tree
  - BSP construction enables simple front-to-back traversal

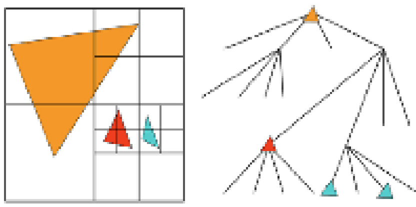


Grid Discussion

- Regular
  - \+ easy to construct
  - \+ easy to traverse
  - – may be only sparsely filled
  - – geometry may still be clumped



- Adaptive
  - \+ grid complexity matches geometric density
  - – more expensive to traverse (especially BSP tree)



Example

- https://www.youtube.com/watch?v=aKqxonOrl4Q
- https://www.youtube.com/watch?v=qlyzo9ll9Vw
- https://www.youtube.com/watch?v=AV279wThmVU
-