Ray Tracing

Dr. Michael Laszlo
Nova Southeastern University
School of Computer and Information Sciences
2001

Topics

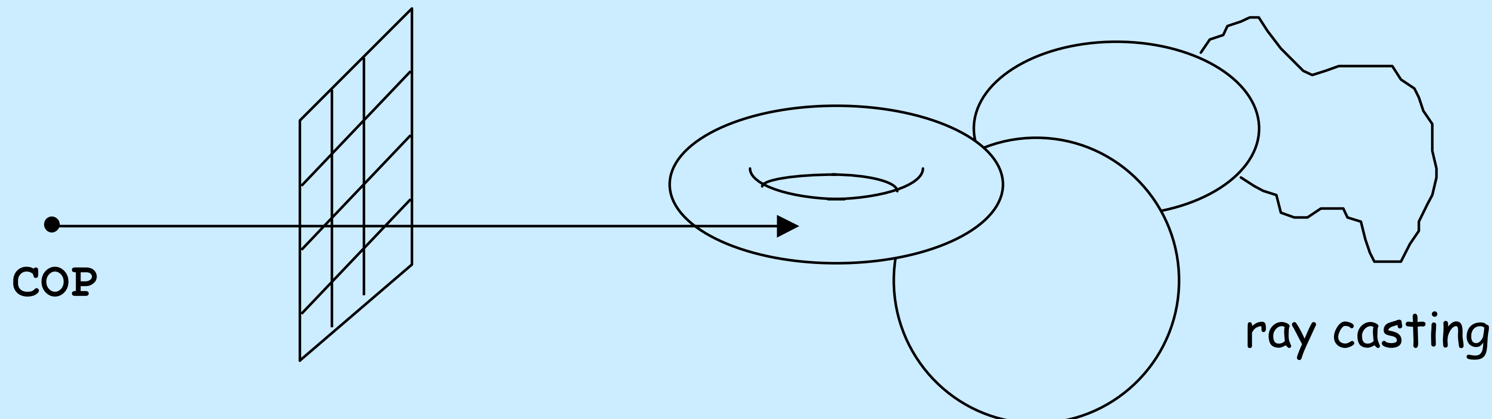
- What are ray casting and ray tracing?
- Computing the intersection of a ray with:
  - a sphere
  - a polygon
  - a constructive solid geometry (CSG) tree
- The standard illumination model
- A high-level implementation of ray tracing
- Techniques for accelerating ray tracing

Ray Casting and Ray Tracing

- Ray tracing is a technique by which rays are "cast" into the scene model to simulate the interaction of light with the scene. If no secondary rays are cast (due to the refraction and reflection of light), the technique is called ray casting.
- Primary rays are cast backwards, from the eyes into the scene, rather than from light source into the scene as occurs in the real world. This avoids casting rays that would not be seen and would not contribute to the image.

Ray Casting and Ray Tracing

- To ray cast, choose a center of projection (COP) in world coordinate space and, for each pixel in the image plane, cast a ray (called a primary ray) into the scene. The first object hit (if any) determines the color of the pixel. Ray casting is used primarily for hidden surface removal, and provides features such as shading, shadows, and texture mapping.
- To *ray trace*, secondary rays are also spawned: If the object hit by the primary ray is reflective, a secondary ray is cast; if the object hit is refractive, another secondary ray is cast. If these secondary rays hit objects, they may give rise to additional secondary rays. Ray tracing is used to render highly realistic images possessing such features as transparency and refraction, reflection, and shadows.



Ray Casting

```
rayCast() {
  select COP, image plane, and window on image plane;
  for each pixel p {
    determine ray r from COP through p;
    curobj = curint = background;
    for each object O in scene
      if O is intersected by r and is closer than curobj {
         curobj = O;
         curint = point of intersection;
      }
    set pixel p to color of point curint;
   }
}
```

- **Questions:**
  - **How do we determine the first point (if any) at which a ray intersects the objects in the scene?**
  - **What color do we paint the point of intersection when it has been found?**
- **The answers to these questions depend on the scene model.**

Intersecting a Ray with Objects

- The next several slides address the first question, that of determining the first point of intersection between a ray R and three different scene models:
  - spheres in space
  - polygons in space
  - constructive solid geometry (CSG) trees in space
- In each case, the ray R is parameterized by a positive real number t:

$$R(t) = p + qt \text{ for } t>0,$$
$$= (px,py,pz) + (qx,qy,qz)t$$

The ray R originates at the point $p=(px,py,pz)$. As the parameter t increases, we move along the direction vector $q=(qx,qy,qz)$. In general, we seek the smallest value $t>0$ for which the ray $R(t)$ intersects some object in the scene.

Intersecting a Ray with a Sphere

The ray R is given by

$R(t) = p + qt$ for $t >= 0$,

$= (px,py,pz) + (qx,qy,qz)t$

The sphere of radius r and center (a,b,c) is given by

$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$.

Substituting $px + t*qx$ for x,
$py + t*qy$ for y, and
$pz + t*qz$ for z, yields

$$(qx^2 + qy^2 + qz^2)t^2 +$$
$$2(qx(px-a) + qy(py-b) + qz(pz-c))t +$$
$$(px^2-2a*px+a^2 + py^2-2b*py+b^2 + pz^2-2c*pz+c^2 - r^2) = 0.$$

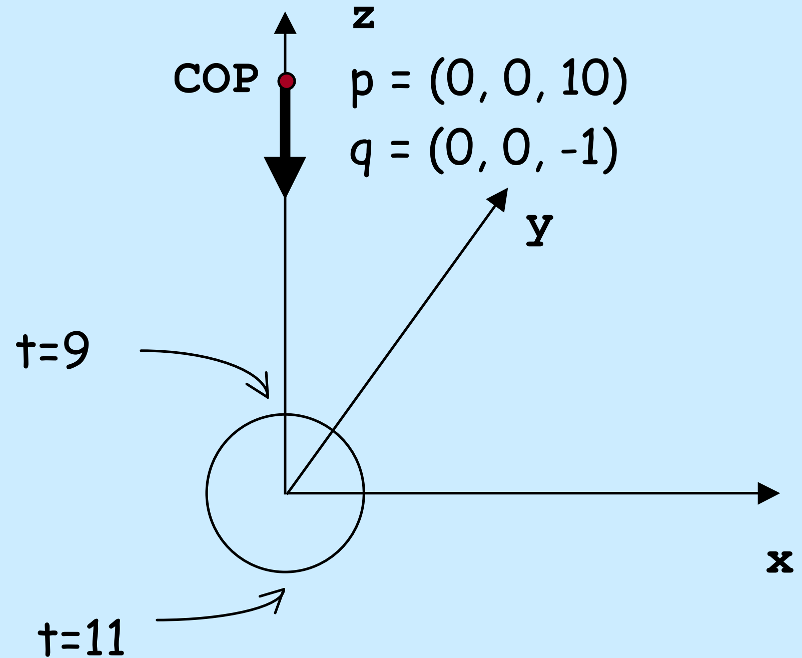
Since this is of the form $At^2 + Bt + C = 0$, we can use the quadratic formula to solve for t:

$t = (-B \pm sqrt(B2 -4AC)) / 2A$

Intersecting a Ray with a Sphere: An Example

$t^2 + 2*(-10)t + (100-1) = 0$
$t^2 - 20t + 99 = 0.$

$t = (20 \pm \text{sqrt}(400-396)) / 2$
$= (20 \pm 2) / 2$
$= 10 \pm 1$



a unit sphere
centered at the origin

Intersecting a Ray with a Polygon

The ray R is given by

$$R(t) = p + qt \text{ for } t >= 0,$$
$$= (px,py,pz) + (qx,qy,qz)t$$

The plane equation of the polygon is given by

$$Ax + By + Cz + D = 0.$$

Substituting px + t*qx for x,
py + t*qy for y, and
pz + t*qz for z, yields

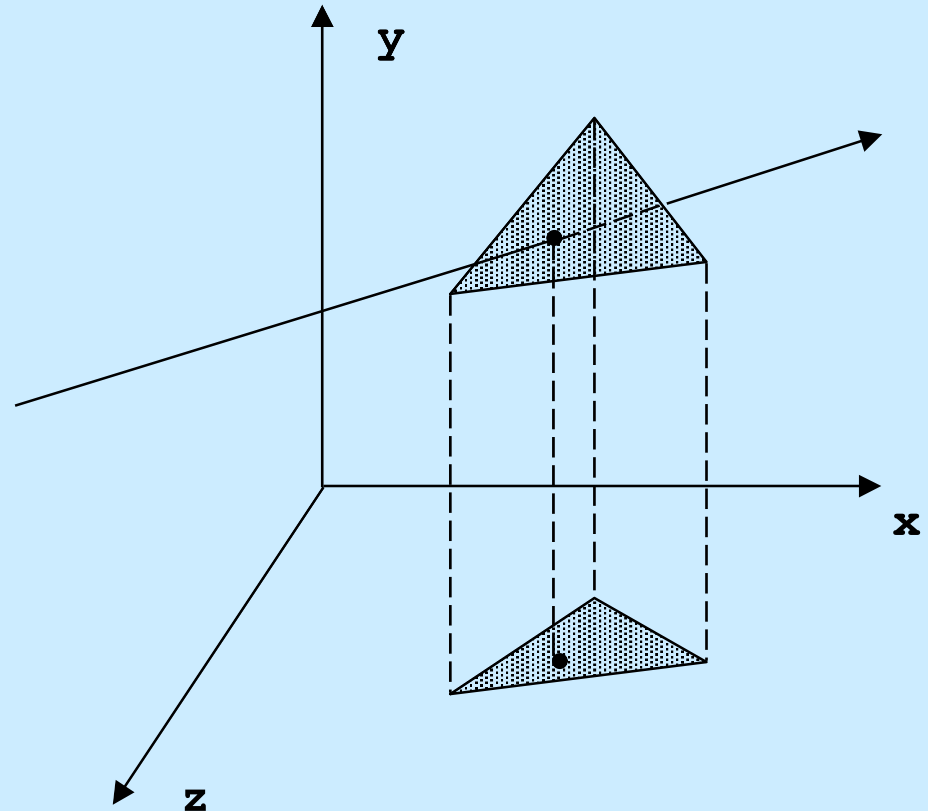
$$t = \frac{-(A*px + B*py + C*pz + D)}{(A*qx + B*qy + C*qz)}$$

If the denominator is 0, the plane and the ray are parallel, since the dot product
$(A,B,C) \cdot q = 0$.

If t<0, the ray does not intersect the plane.
Otherwise (t >= 0) the ray intersects the plane, and we must decide whether the ray intersects the polygon.

Intersecting a Ray with a Polygon

One way to decide whether ray *R* intersects polygon *P* is to project both *P* and the point *q* at which *R* crosses the plane of *P*, into one of the three major planes with which the polygon is not perpendicular. Let *q′* be the projection of *q*, and *P′* the projection of *P*. Then *q* lies in *P* if and only if *q′* lies in *P′*.



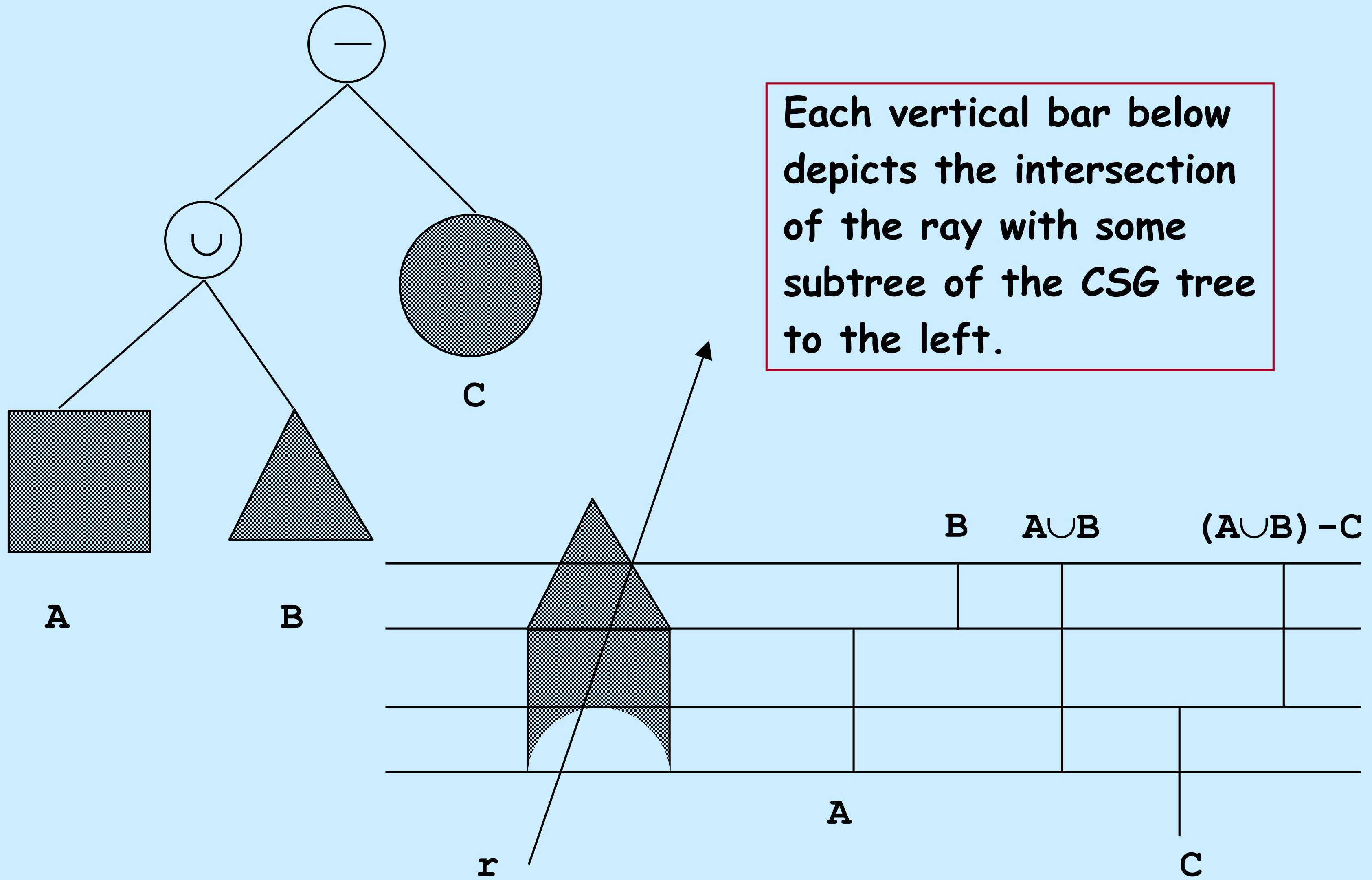
Intersecting a Ray with a CSG Tree

- To compute the intersection of ray *R* and CSG tree *T*, we perform an inorder traversal of *T*:
  - At each leaf node, we return the intersection of r and the primitive stored at the node.
  - At each internal node, we apply the operation stored at the node to the intervals yielded by the inorder traversal of its two subtrees, and return the result.
- The method, illustrated on the next slide, takes advantage of this relation:

  $$R \cap (A \; op \; B) = (R \cap A) \; op \; (R \cap B)$$

  where *op* stands for a Boolean operation.

Intersecting a Ray with a CSG Tree



Ray Tracing

C

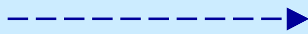
A

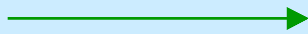
COP

image plane

B

light


primary ray


reflection ray

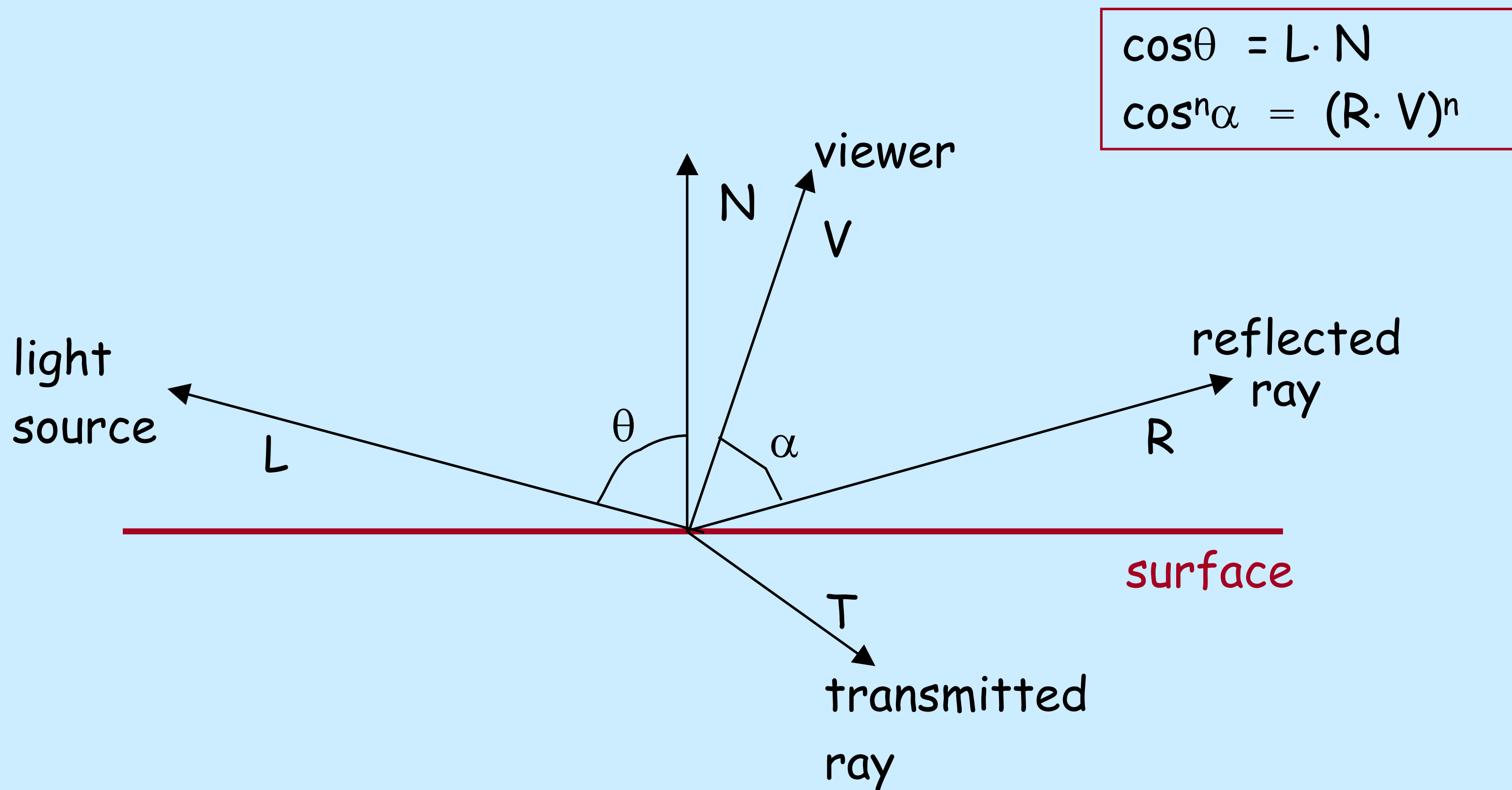
transmission ray

shadow ray

The Standard Illumination Model

- An illumination model determines the color of a ray. Since a pixel is painted the color of the primary ray that passes through it, the illumination model in force affects the ray traced image.
- Under the standard illumination model, the color of a ray derives from five factors:
  - Ambient light
  - Diffuse reflection
  - Specular reflection
  - Reflection (or refraction) of the scene
  - Transmission of the scene

The Standard Illumination Model



The Standard Illumination Model

- In the following formula, I denotes the intensity a point p on an object's surface, and k_a, k_d, k_s, n, k_t, and k_r are material properties:
  - N is the normal to the surface at p
  - R is the reflection vector
  - V is the vector toward the viewer
  - S_L = 1 if point p is not in shadow from light source L
= 0 if point p is in shadow from light source L
  - I_a is the intensity of ambient light
  - I_L is the intensity of light source L
  - I_t is the intensity of the contribution due to transparency
  - I_r is the intensity of the contribution due to reflection

$$I = k_a I_a + \sum_L S_L I_L \left(k_d (N \cdot L) + k_s (R \cdot V)^n\right) + k_t I_t + k_r I_r$$

Ray Tracing: An Implementation

- **In the following implementation, procedure *traceRay* traces a ray through the scene and returns the ray's color. Procedure *shadeRay* returns the color of a ray that has hit an object, spawning reflection and/or transmission secondary rays as necessary. Parameter *depth* limits the depth of the resulting ray tree.**

```
traceRay(Ray r, int depth) {
    if depth = 0 then return;
  determine closest intersection point p of ray
     with some object Obj;
  if Obj exists {
    compute normal N at poin;
    return shadeRay(obj, r, p, N, depth);
  } else
    return BACKGROUND_COLOR;
}
```

Ray Tracing: An Implementation

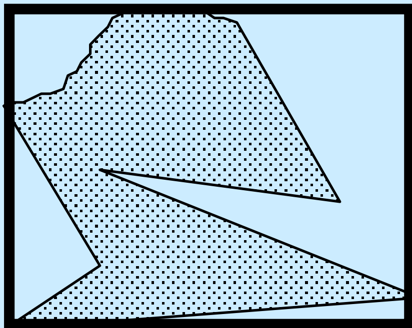
```
shadeRay(Object obj, Ray r, Point p, Vector N, int depth){
  color = ambient term;
  for each light source L {
    sray = shadow ray from point to light;
    if N · L > 0
      color = color + "specular and diffuse terms from
                       this light source";
  }
  if depth < MAXDEPTH {
    if Obj is reflective {
      rRay = ray from point p in reflection direction;
      rColor = traceRay(rRay, depth-1);
      color = color + kr*rColor;    // scale by refl. coeff.
    }
    if Obj is transparent and
       total internal refraction does not occur {
      tRay = ray from point p in refraction direction;
      tColor = traceRay(tRay, depth-1);
      color = color + kt*tColor;    // scaled by trans coef.
    }
  }
  return color;
}
```

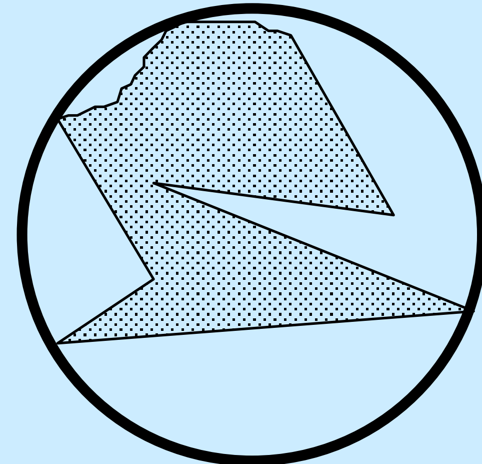
Accelerated Ray Tracing

- In ray tracing, most of the time (75–95%) is spent performing intersection calculations. There are two approaches to speeding things up:
  - Speed up intersection calculations. This approach depends on the scene model.
  - Perform fewer intersection calculations by avoiding unlikely candidates, objects which the ray cannot hit.
- There are numerous methods to avoid performing unnecessary intersection calculations. Preprocessing (prior to casting rays) is necessary for each of these methods. Although preprocessing tends to be expensive, its cost is amortized over the many rays that get cast. We will look at these acceleration methods:
  - bounding extents around objects
  - bounding extent hierarchies
  - uniform spatial partitioning
  - oct-tree spatial partitioning
  - item buffers
  - light buffers

Accelerated Ray Tracing: Bounding Extents

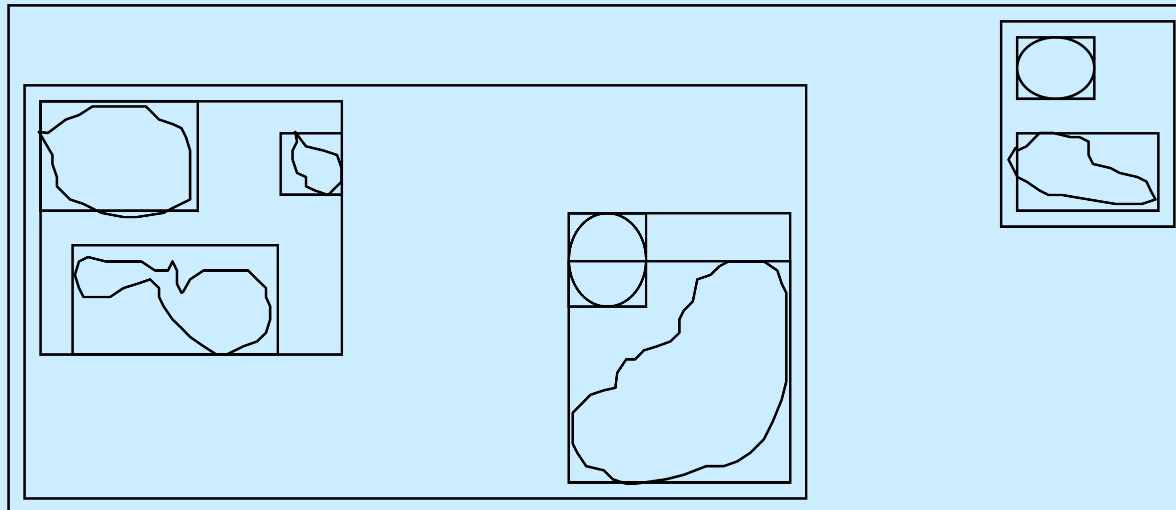
- For each object in the scene, a volume containing the object is precomputed. The volume is called a bounding extent; if it is a box whose edges are parallel to the major axes it is called a bounding box. To cast some ray *R* against an object, we test whether ray *R* intersects the object's bounding extent. If not, *R* cannot intersect the object. Alternatively, if *R* does intersect the bounding extent, ray R possibly intersects the object itself, so we calculate the intersection of *R* with the object.
- The advantage of using bounding extent is that the ray/extent intersection is cheaper to compute than the ray/object intersection. The tighter the bounding extent, the fewer "false positives" occur, but the more costly it may be to compute the ray/extent intersection.





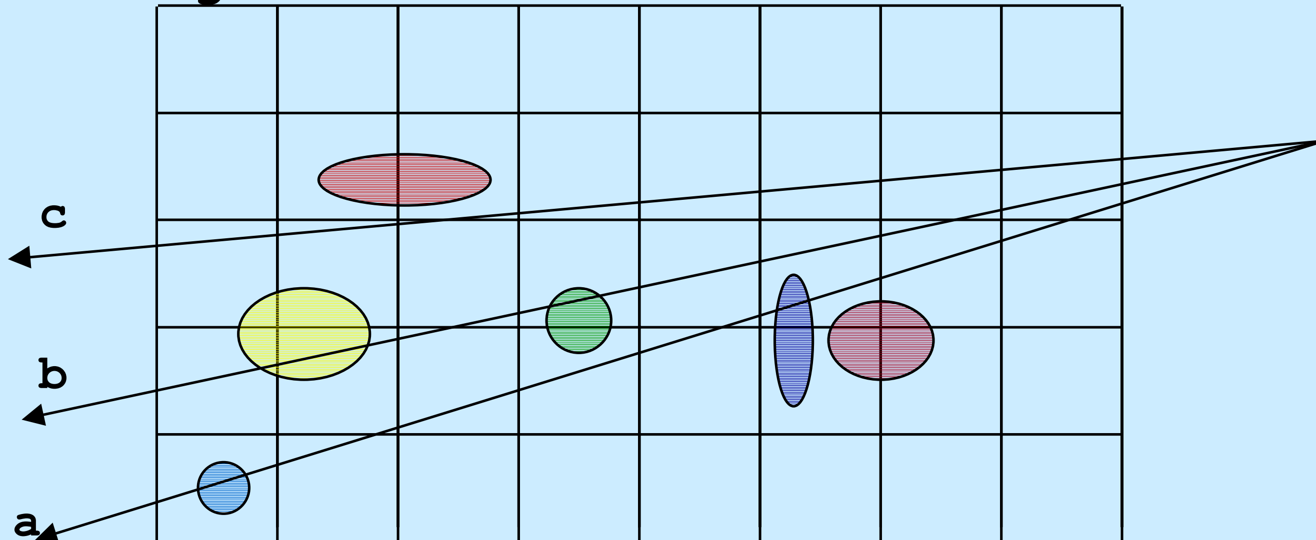
Accelerated Ray Tracing: Bounding Extent Hierarchies

- In a preprocessing phase, objects are enclosed in bounding extents, then the bounding extents are themselves enclosed in bounding extents based on clustering. The process continues until one bounding extent encloses the entire scene. What results is a hierarchy of bounding extents, with objects at the leaves.
- When casting a ray, one performs a bounded traversal of the tree. At each internal node *n*, if ray *R* intersects the bounding extent stored at *n*, then recur on node *n*'s subtrees; otherwise return. At each leaf node, compute the intersection between ray *R* and the object stored at the node.



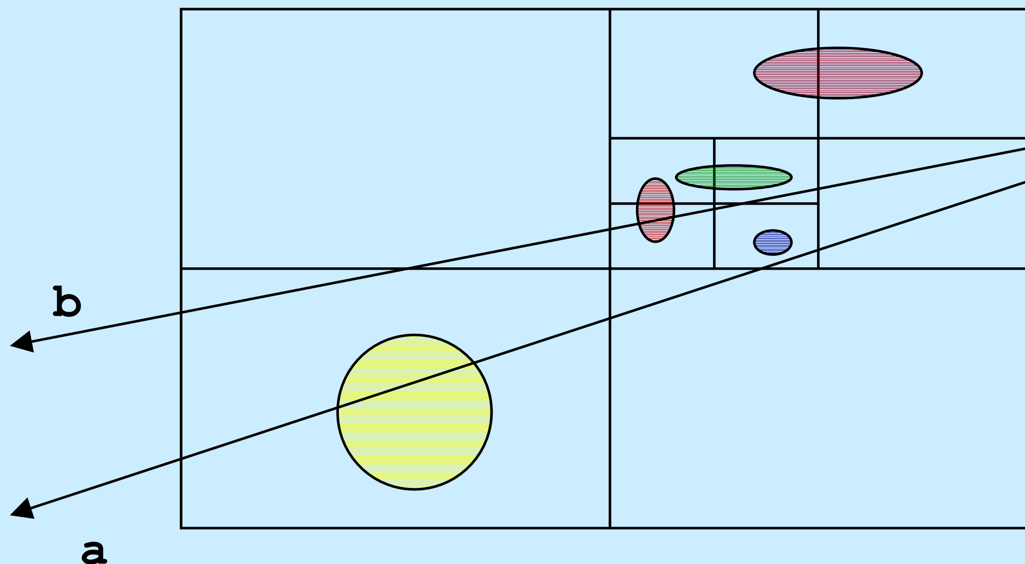
Accelerated Ray Tracing: Uniform Spatial Partitioning

- In a preprocessing phase, decompose the scene's bounding box into uniform cubes or voxels. For each voxel store a list of those scene objects that intersect the voxel (the objects are "dropped" into the voxels).
- When casting a ray *R*, propagate the ray from voxel to voxel. For each voxel encountered, test ray *R* against each objects stored in the voxel's list. If there is one or more intersection, return the closest point of intersection; otherwise propagate the ray to the next voxel. If ray *R* leaves the scene, *R* gets the background color.



Accelerated Ray Tracing: Oct-tree Spatial Partitioning

- The problem with uniform spatial partitioning is that that the objects in the scene may be non-uniformly distributed. If so, the voxel lists are empty where the scene is sparse and over-full where the scene is dense. The solution is to use an oct-tree partitioning that is sensitive to the distribution of objects.
- As a ray is propagated from octant to octant, it is intersected with each octant's list of objects. The process continues until the nearest object is hit or the background is reached.



In this example, partitioning occurs until every octant contains no more than two objects.

Accelerated Ray Tracing: Item Buffers

- **Item buffers** exploit the fact that primary rays are special: they emanate from the same point (i.e., the center of projection). An item buffer is used to avoid having to test for the intersection of primary rays with *any* objects in the scene. During preprocessing, an inexpensive visible surface algorithm (such as z-buffering) is applied. However, instead of creating an image in the frame buffer, the algorithm is used to fill an item buffer that has the same resolution as the frame buffer. Each pixel of the item buffer contains the identifier of the object of the closest object that projects to the pixel.
- When a primary ray is to be cast through pixel *p,* the ray is intersected with the item stored in pixel *p* of the frame buffer. There is no need to test the ray for intersection with any other objects in the scene.
- Secondary rays are cast in the usual way–the item buffer does not help with casting secondary rays.

Accelerated Ray Tracing: Light Buffers

- **Light buffers** exploit the fact that shadow rays are special: each is fired toward one of a small set of objects (the light sources). Light buffers speed up processing of shadow rays. A light buffer is a cube centered on a light source each of whose 6 faces is tiled with a regular square of grids. Each square determines a 4-sided pyramid whose apex is the light source. During preprocessing, a depth-ordered list is constructed for each pyramid, consisting of the set of objects that intersect the pyramid.
- When a shadow ray is cast to the light source, the list associated with the pyramid through which the ray passes is traversed to determine whether the ray's origin is in shadow.