CSE 413
(Computer Graphics)
Ray Tracing

Iyolita Islam

Department of Computer Science and Engineering
Military Institute of Science and Technology

Last Updated: November 15, 2020

Unintentional Mistakes

Best efforts have been exercised in order to keep the slides error-free, the preparer does not assume any responsibility for any unintentional mistakes. The text books must be consulted by the user to check veracity of the information presented.

Outline

Ray Tracing/ Ray Casting

- Ray tracing determines the visibility of surfaces by tracing imaginary rays of light from the viewer's eye to the object in the scene.
- It is a prototypical image precision algorithm.
- A center of projection (the viewer's eye) and the window on an arbitrary view plane are selected.
- The window may be assumed to be divided into regular grid which can correspond to the pixels of the image.
- For each pixel of the window an eye ray is fired from the COP through the pixel center into the scene.

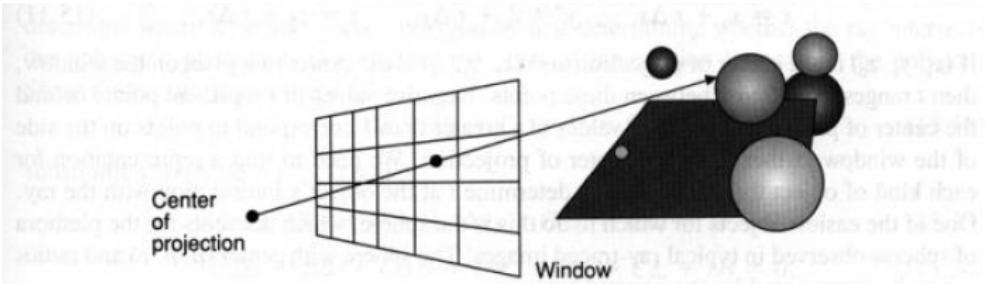

Fig. 15.55 A ray is fired from the center of projection through each pixel to which the window maps, to determine the closest object intersected.

Pinhole Camera

- Using a pinhole camera, a closed box with a tiny hole in the center of the front panel, we can take a perfect picture.
- The hole is so small that only one light ray passing through it can strike a particular point on the negative, which is mounted on the inside of the back panel.
- As light rays from across the scene expose the negative by hitting their respective target precisely, a sharp image depicting the scene emerges.

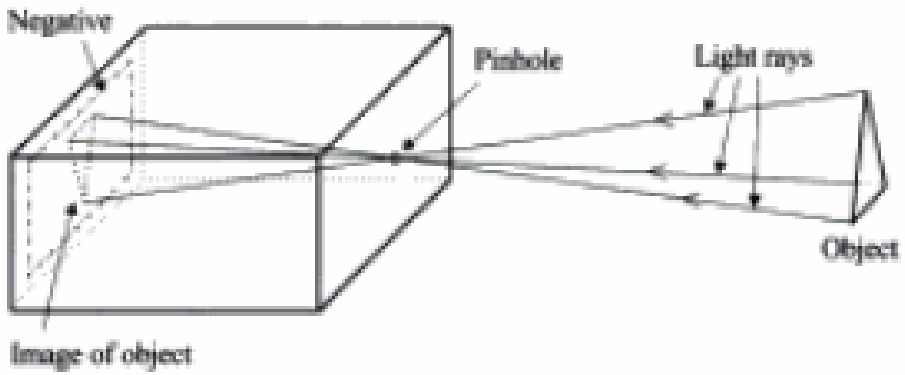

Fig. 12-1 A pinhole camera.

Ray Tracing Algorithm

```
select center of projection and window on viewplane;
for (each scan line in image) {
    for (each pixel in scan line) {
        determine ray from center of projection through pixel;
        for (each object in scene) {
            if (object is intersected and is closest considered thus far)
                record intersection and object name;
        }
        set pixel's color to that at closest object intersection;
    }
}
```

Fig. 15.56 Pseudocode for a simple ray tracer.

Ray Tracing Algorithm is used to determine:

- Visible Surface
- Shadow
- Reflection
- Refraction

Type of Ray Tracing Algorithm

Three types of ray tracing algorithm:

- Forward Ray Tracing:
  - Rays from light source bounce of objects before reaching the camera.
  - Computational wastage
- Backward Ray Tracing:
  - Track only those rays that finally made it to the camera.
- Hybrid Ray Tracing:
  Combination of forward and backward ray tracing:
  - Backward Ray tracing + rays to light source
  - backward ray tracing + forward ray tracing + storage on surface.
  - backward ray tracing + forward ray tracing + connecting lines.

Type of Ray Tracing Algorithm

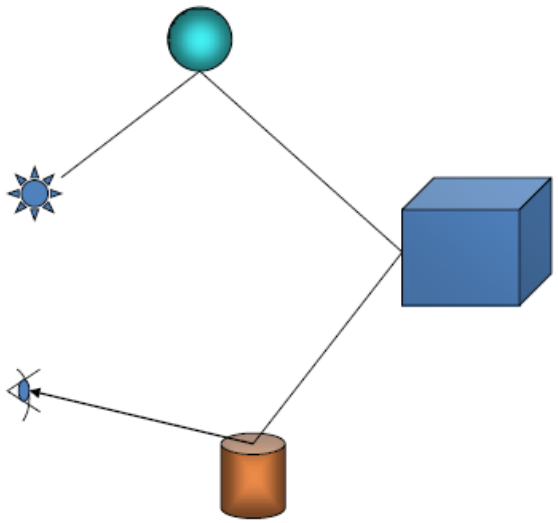

Figure: Forward Ray Tracing

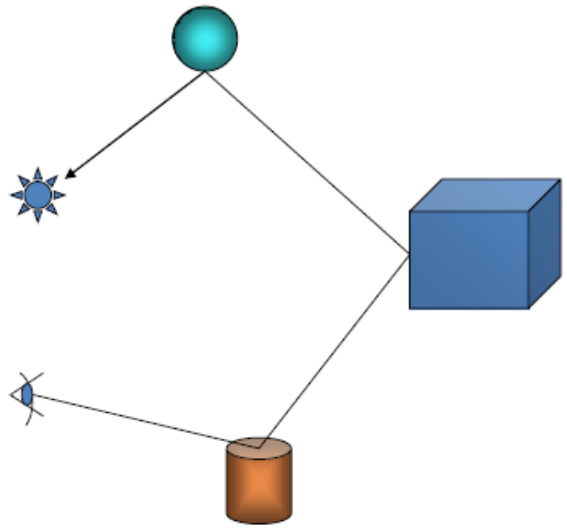

Figure: Backward Ray Tracing

Ray Tracing Algorithm Example

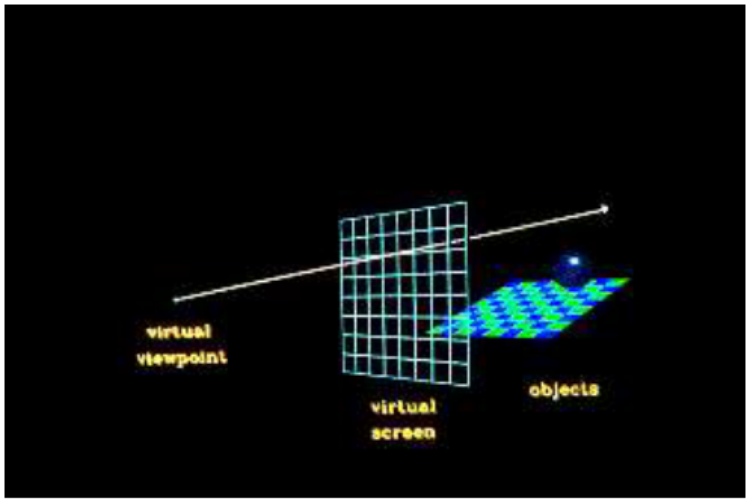

Figure: Sometimes a ray misses all objects

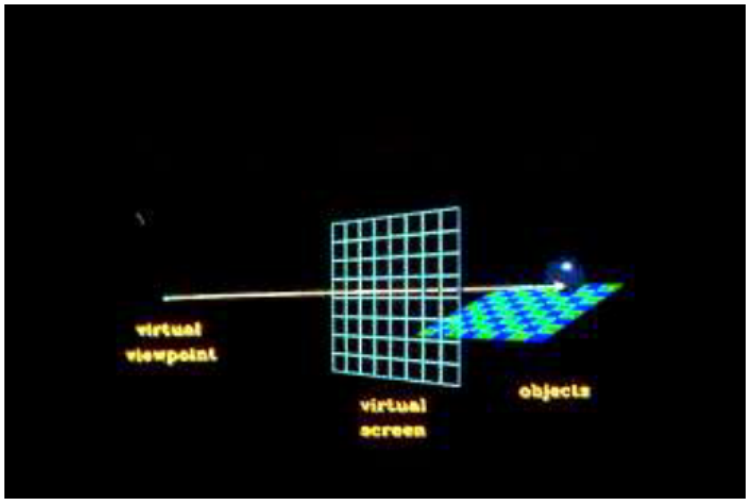

Figure: Sometimes a ray hit an object

Ray Tracing Algorithm Example

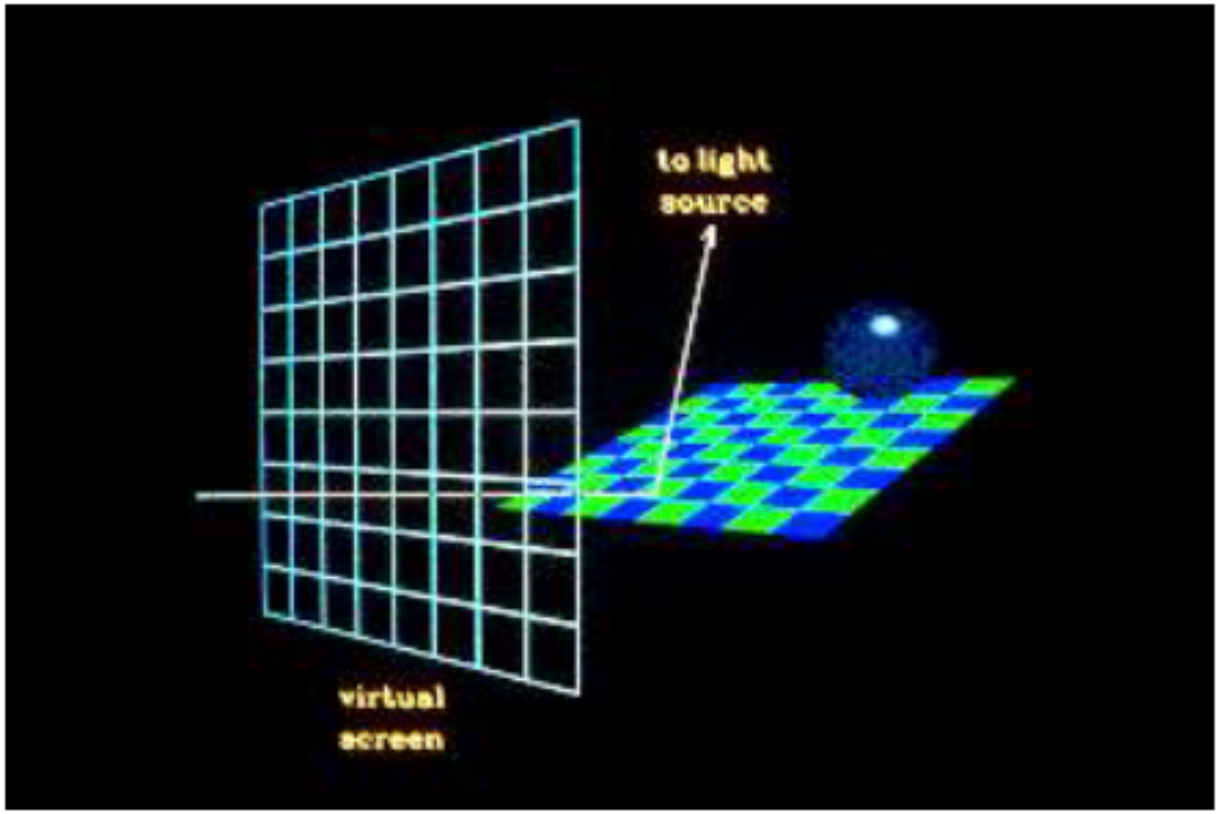

Figure: Rays reflect to the light source

Ray Tracing Algorithm Example

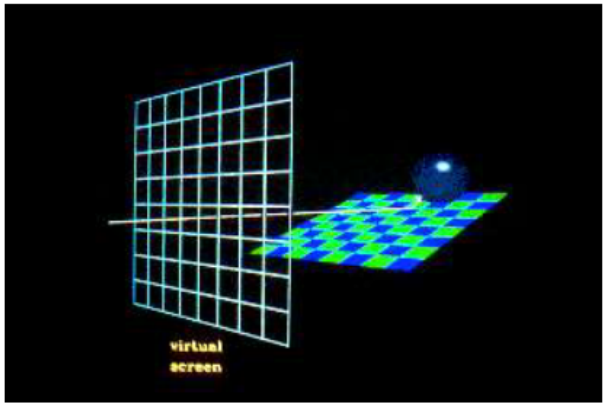

Figure: Rays intersect another object (First intersection point in shadow of the second object)

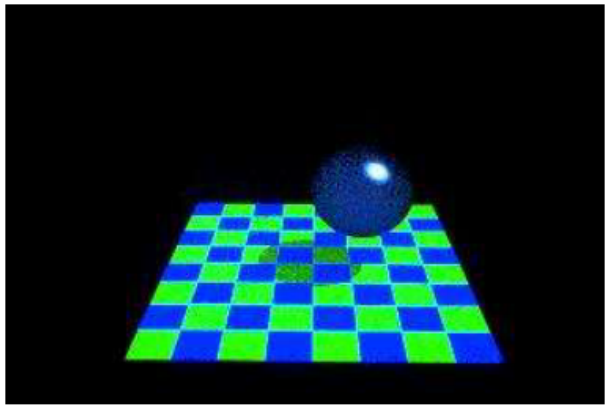

Figure: Output image of this case

Ray Tracing Algorithm - Reflection

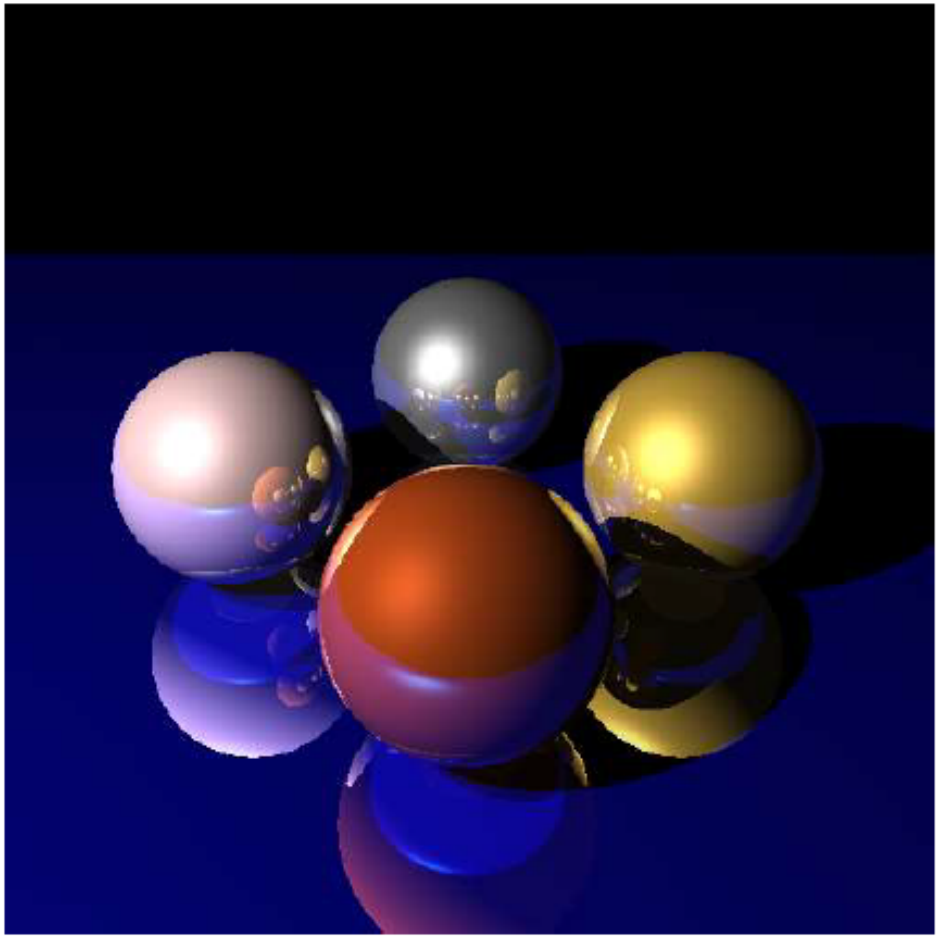

Figure: Image with all the reflecting information

Computing Intersection

- The task of determining the intersection of a ray with an object is very important for any ray tracer.
- For this, parametric representation of a vector is used.
- Each point (x, y, z) along the ray from (x_0, y_0, z_0) to (x_1, y_1, z_1) is defined by some value t such that
  $x = x_0 + t(x_1 - x_0)$,
  $y = y_0 + t(y_1 - y_0)$,
  $z = z_0 + t(z_1 - z_0)$
- We define, $\Delta x = x_1 - x_0$, $\Delta y = y_1 - y_0$, $\Delta z = z_1 - z_0$
- $x = x_0 + t\Delta x$,
  $y = y_0 + t\Delta y$,
  $z = z_0 + t\Delta z$ ... ... (i)

Computing Intersection

- If (x_0, y_0, z_0) is the center of projection and (x_1, y_1, z_1) is the center of a pixel on the window, then t ranges from 0 to 1 between these points.
- Negative values of t represent points behind the COP, whereas the values of t greater than 1 correspond to the points on the side of the window farther from the COP.
- We need to find a representation for each kind of object that enables us to determine t at the object's intersection with the ray.
- We can consider a sphere with center at (a, b, c) and radius r, as an object and its equation:
  $(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$ ... ... (ii)

Computing Intersection

- The intersection can be found by expanding (i) and substituting values of x, y and z from (ii) to yield
  $x^2 - 2ax + a^2 + y^2 - 2by + b^2 + z^2 - 2cz + c^2 = r^2$
- $(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2$
  $+(y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + b^2$
  $+(z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2$
- ${x_0}^2 + 2x_0\Delta xt + \Delta x^2t^2 - 2ax_0 - 2a\Delta xt + a^2$
  $+{y_0}^2 + 2y_0\Delta yt + \Delta y^2t^2 - 2by_0 - 2b\Delta yt + b^2$
  $+{z_0}^2 + 2z_0\Delta zt + \Delta z^2t^2 - 2cz_0 - 2c\Delta zt + c^2 = r^2$
- $(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2$
  $+2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)]$
  $+({x_0}^2 - 2ax_0 + a^2 + {y_0}^2 - 2by_0 + b^2 + {z_0}^2 - 2cz_0 + c^2) - r^2 = 0$
- $(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2$
  $+2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)]$
  $+(x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$ ... ... (iii)

Computing Intersection

- (iii) is a quadratic equation in t, with coefficients expressed entirely in constants derived from the sphere and ray equations, so it can be solved using quadratic formula:
  - If there are no real roots, then the ray and the sphere doesn't intersect.
  - If there is one real root, then the ray grazes the sphere.
  - Otherwise, the roots are the points of intersection with the sphere; the one that yields the smallest positive t is the closest.
- It is also useful to normalize the ray so that the distance from (x_0, y_0, z_0) to (x_1, y_1, z_1) is 1.
- This gives a value of t that measures distance in WC units and simplifies the intersection calculation.since the coefficient of t^2 in (iii) becomes 1.

Computing Intersection

- We must determine the surface normal at the point of intersection in order to shade the surface.
- This is particularly easy for sphere since the (unnormalized) normal is the vector from the center to the point of intersection: The sphere with the center (a, b, c) has a surface normal $(\frac{x-a}{r}, \frac{y-b}{r}, \frac{z-c}{r})$ at the point of intersection (x, y,z).
- Finding the intersection of a ray with a polygon somewhat more difficult.
- We can determine where a ray intersects a polygon by determining whether the ray intersect the polygon's plane at first and then whether the point of intersection lies within the polygon.

Computing Intersection (Polygon)

- Since the equation of the plane is : Ax + By + Cz + D = 0 ... ... (iv)
- Substituting from (iii),
  $A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0$
  $\Rightarrow t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0$
  $\Rightarrow t = -\frac{Ax_0+By_0+Cz_0+D}{A\Delta x+B\Delta y+C\Delta z}$ ... ... (v)
  - If the denominator is 0, the plane and the ray are parallel and don't intersect.
  - Otherwise, the ray and the plane intersect.
- To determine if the point lies within the polygon, we can project the polygon and point orthographically onto one of the three planes defining the coordinate system (Fig 15.57).

Computing Intersection (Polygon)

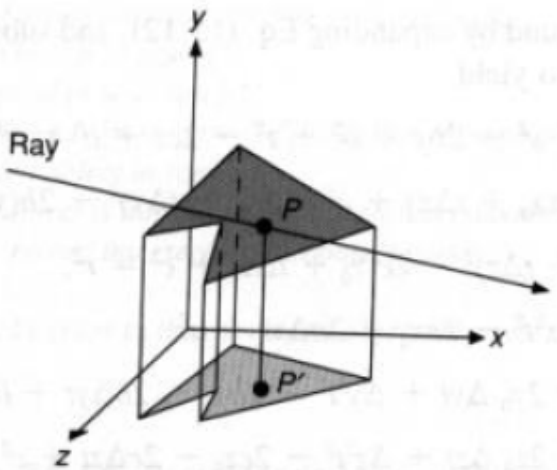

Fig. 15.57 Determining whether a ray intersects a polygon. The polygon and the ray's point of intersection p with the polygon's plane are projected onto one of the three planes defining the coordinate system. Projected point p' is tested for containment within the projected polygon.

- Selecting the axis along which to project that yields the largest projection will help to obtain most accurate result .
- This corresponds to the coordinate whose coefficient in the polygon's plane equation has the largest absolute value.
- The orthographic projection is accomplished by dropping this coordinate from the polygon vertices and from the point.

Efficiency Considerations

- The ray tracing algorithm intersects each of the rays from the eye with each of the eyes in the scene.
- A 1024 X 1024 image of 100 objects will require 100M intersection calculations.
- To improve the efficiency of ray tracing we can-
    - speed up individual intersection calculations
    - avoid intersection calculation entirely

Optimizing Intersection Calculations

- Many of the terms in the equations for object-ray intersection contains expression that are constant either throughout an image or for a particular ray.
- These can be computed in advance as for example, the orthographic projection of a polygon onto a plane.
- if rays are transformed to lie along the z axis, the same transformation can be applied to each candidate object so that any intersection occurs at $x = y = 0$.
- This simplifies the intersection calculation and allows the closest object to be determined by a z sort.
- The intersection point be transformed back for use in shading calculations via the inverse transformation.

Optimizing Intersection Calculations

- Bounding volumes provide a particularly attractive way to decrease the amount of time spent on intersection calculations.
- An object that is relatively expensive to test for intersection may be enclosed in a bounding volume whose intersection test is less expensive, such as a sphere, ellipsoid, or rectangular solid.
- The object doesn't need to be tested is the ray fails to intersect with its bounding volume.
- Using a bounding volume that is a convex polyhedron formed by the intersection of a set of infinite slabs, each of which is defined by a pair of parallel planes that bound the object.
- Figure 15.58(a) shows in 2D an object bounded by 4 slabs (defined by pair of parallel lines), and by their intersection.

Optimizing Intersection Calculations

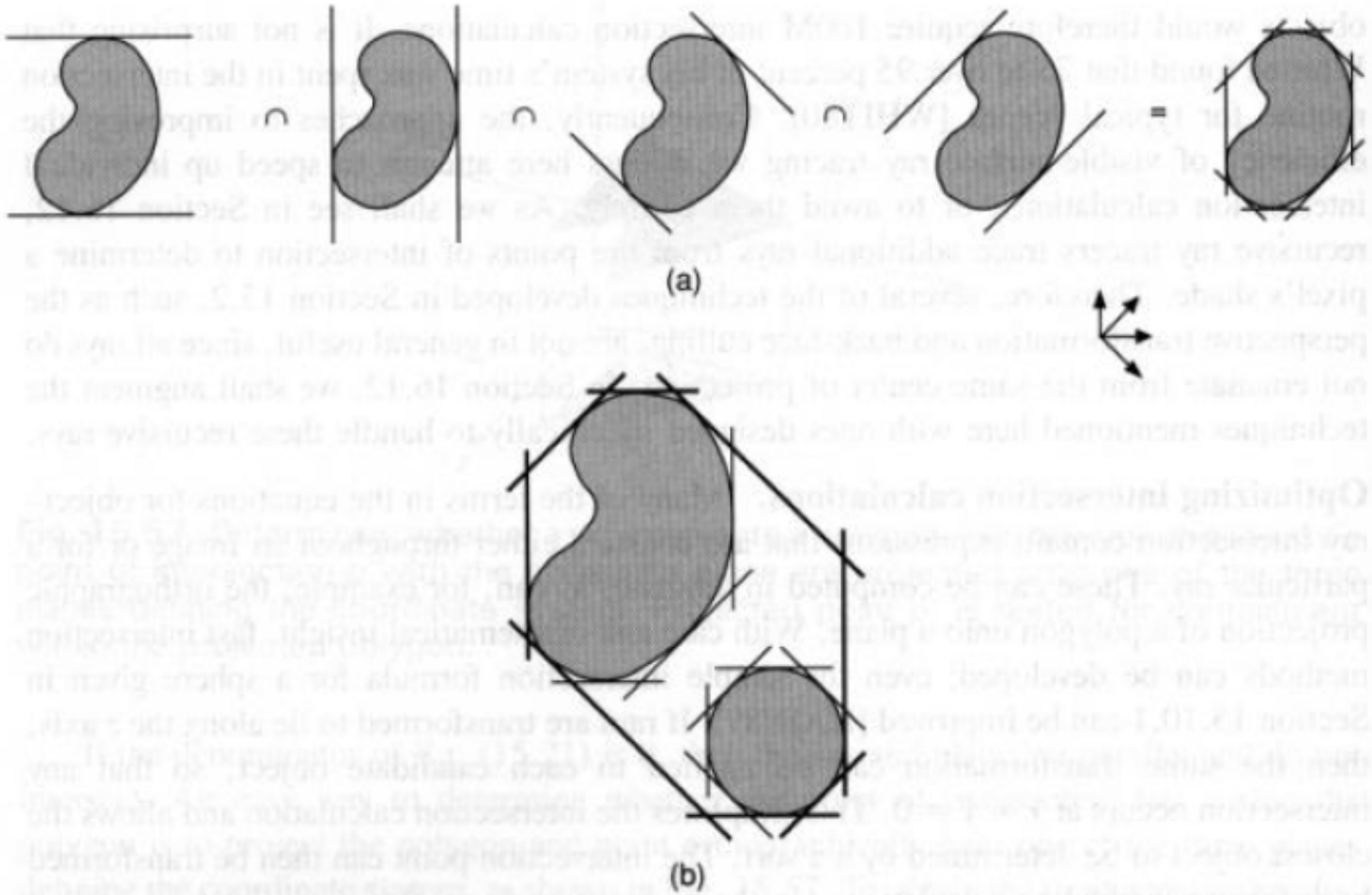

Fig. 15.58 Bounds formed intersection of slabs. (a) Object bounded by a fixed set of parameterized slabs. (b) The bounding volume of two bounding volumes.

Optimizing Intersection Calculations

- Thus, each slab is represented by equation (iv), where A, B, and C are constant, and D is either D_{min} or D_{max}.
- If the same set of parameterized slabs are used to bound all the objects, each bound can be described compactly by the D_{min} and D_{max} of each of its slabs.
- A ray is intersected with a bound by considering one slab at a time.
- The intersection of a ray with a slab can be computed using equation (v) for each of the slabs planes, producing near and far values of t.

Optimizing Intersection Calculations

- Using the same set of parameterized slabs for all bounds, however, allows us to simplify equation (v) as:
  $t = \frac{S+D}{T}$
  where, $S = Ax_0 + By_0 + Cz_0$
  $T = \frac{-1}{A\Delta x + B\Delta y + C\Delta z}$
- Both S and T can be calculated once for a given ray and parameterized slab.

Optimizing Intersection Calculations

- Since each bound is an intersection of slabs, the intersection of the ray with an entire bound is just the intersection of the ray's intersections with each of the bound's slabs.
- This can be computed by taking the maximum of the near values of t and the minimum of the far values of t.
- In order to detect null intersections quickly, the maximum near and minimum far values of t for a bound can be updated as each of its slabs is processed, and the processing of the bound terminated if the former ever exceeds the latter.

Avoiding Intersection Calculations

- Ideally each ray should be tested for intersection only with objects that it actually intersects.
- Furthermore, in many cases we would like each ray to be tested against only that object whose intersection with the ray is the closest to the ray's origin.
- There is a variety of techniques that attempt to approximate the goal by preprocessing the environment to partition rays and objects into equivalence classes to help limit the number intersections that need to be performed.
- This technique includes two approaches:
    - Hierarchies
    - Spatial Partitioning

Hierarchies

- Although bounding volumes do not by themselves determine the order or the frequency of intersection tests, bounding volumes may be organized in nested hierarchies with objects at the leaves and internal nodes that bound their children.
- For example, a bounding volume for a set of Kay-Kajiya bounding volumes can be computed by taking each pair of planes the minimum D_{min} and the maximum D_{max} of the values for each child volume (Fig 15.58(b)).
- A child volume is guaranteed not to intersect with a ray if its parent does not.
- Thus, if intersection tests begin with the root, many branches of the hierarchy (and hence many objects) may be trivially rejected.

Traverse the Hierarchies

A simple method to traverse a hierarchy:

```
void HIER_traverse (ray r, node n)
{
    if (r intersects bounding volume of n)
        if (n is a leaf)
            intersect r with object of n;
        else
            for (each child c of n)
                HIER_traverse(r, c);
}
```

Spatial Partitioning

- Bounding volume hierarchies organize the objects bottom-up; in contrast, spatial partitioning subdivides the space top-down.
- The bounding box of the scene is calculated first.
- In one approach, the bounding box is then divided into regular grid of equal-sized extents (Fig 15.61).
- Each partition is associated with a list of objects it contains either wholly or in part.
- The lists are filled by assigning each object to the one or more partitions that contain it.
- Now, a ray needs to be intersected with only those objects that are contained within the partitions through which it passes (Fig 15.62).

Spatial Partitioning

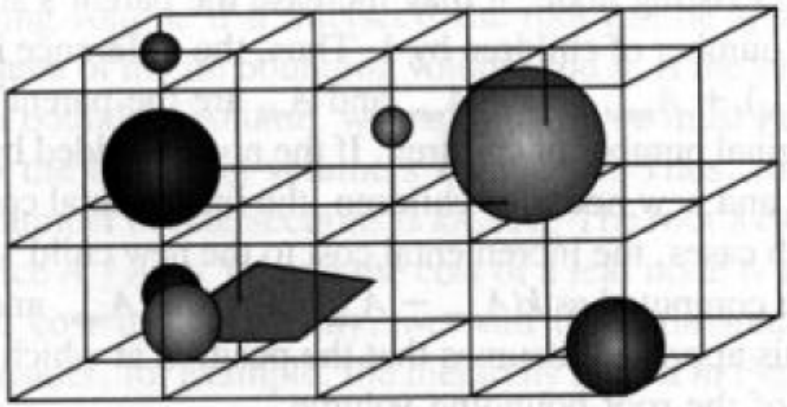

Fig. 15.61 The scene is partitioned into a regular grid of equal-sized volumes.

Spatial Partitioning

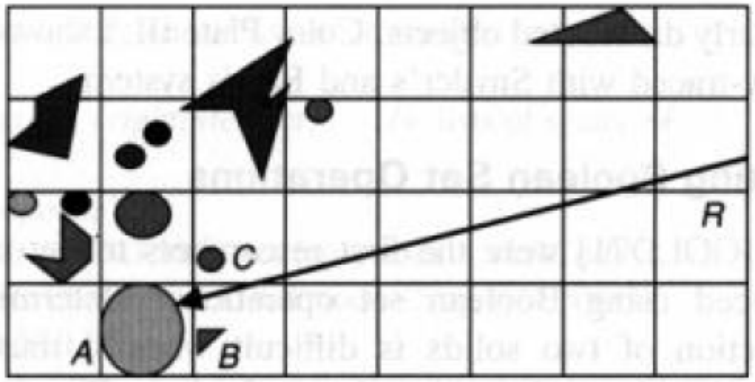

Fig. 15.62 Spatial partitioning. Ray *R* needs to be intersected with only objects *A*, *B*, and *C*, since the other partitions through which it passes are empty.

Spatial Partitioning

- The partitions can be examined in the order in which the ray passes through them; thus, as soon as a partition is found in which there is an intersection, no more partitions need to be inspected.
- We must consider all the remaining objects in the partition, to determine the one whose intersection is closest.
- Since the partitions follow a regular grid, each successive partition lying along a ray may be calculated using a 3D version of the line-drawing algorithm.

End of Slides