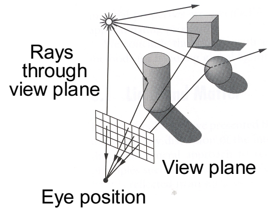
Ray Casting

Thomas Funkhouser
(covering for Finkelstein 9/27)
Princeton University
C0S 426, Fall 2001

3D Rendering

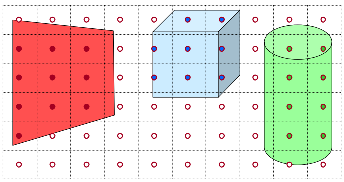
- The color of each pixel on the view plane depends on the radiance emanating from visible surfaces

Simplest method is ray casting



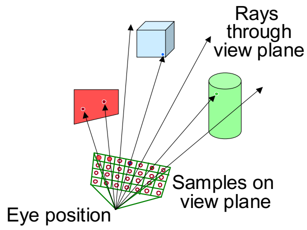
Ray Casting

- For each sample …
  - o Construct ray from eye position through view plane
  - o Find first surface intersected by ray through pixel
  - o Compute color sample based on surface radiance



Ray Casting

- For each sample …
  - o Construct ray from eye position through view plane
  - o Find first surface intersected by ray through pixel
  - o Compute color sample based on surface radiance



Ray Casting

- Simple implementation:

```
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

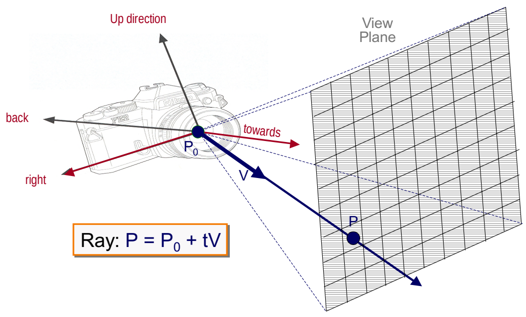

Ray Casting

- Simple implementation:

```
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

7

Constructing Ray Through a Pixel



Ray: $P = P_0 + tV$

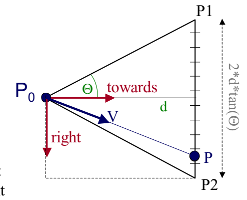
8

Constructing Ray Through a Pixel

- 2D Example

Θ = frustum half-angle
d = distance to view plane

right = towards x up



$P1 = P_0 + d*towards - d*\tan(\Theta)*right$
$P2 = P_0 + d*towards + d*\tan(\Theta)*right$

$P = P1 + (i/width + 0.5) * 2*d*\tan(\Theta)*right$
$V = (P - P_0) / \|P - P_0\|$

Ray: $P = P_0 + tV$

9

Ray Casting

- Simple implementation:

```
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

10

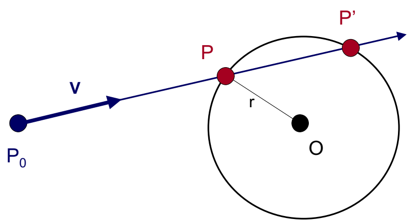
Ray-Scene Intersection

- Intersections with geometric primitives
  - o Sphere
  - o Triangle
  - o Groups of primitives (scene)
- Acceleration techniques
  - o Bounding volume hierarchies
  - o Spatial partitions
    - » Uniform grids
    - » Octrees
    - » BSP trees

11

Ray-Sphere Intersection

Ray: $P = P_0 + tV$
Sphere: $|P - O|^2 - r^2 = 0$



12

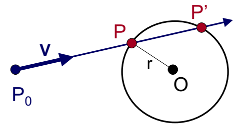
Ray-Sphere Intersection I

Ray: $P = P_0 + tV$
Sphere: $|P - O|^2 - r^2 = 0$

Algebraic Method

Substituting for P, we get:
$|\mathbf{P_0 + tV} - O|^2 - r^2 = 0$

Solve quadratic equation:
$at^2 + bt + c = 0$
where:
$a = 1$
$b = 2 V \bullet (P_0 - O)$
$c = |P_0 - C|^2 - r^2 = 0$



$P = P_0 + tV$

13

Ray-Sphere Intersection II

Ray: $P = P_0 + tV$
Sphere: $|P - O|^2 - r^2 = 0$

Geometric Method

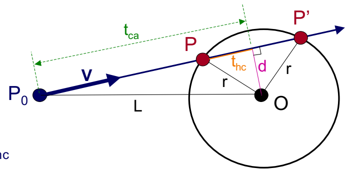
$L = O - P_0$

$t_{ca} = L \cdot V$
if ($t_{ca} < 0$) return 0

$d^2 = L \cdot L - t_{ca}^2$
if ($d^2 > r^2$) return 0

$t_{hc} = \text{sqrt}(r^2 - d^2)$
$t = t_{ca} - t_{hc}$ and $t_{ca} + t_{hc}$

$P = P_0 + tV$

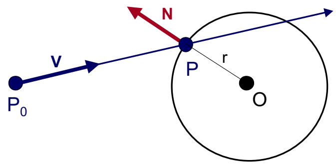


14

Ray-Sphere Intersection

- Need normal vector at intersection for lighting calculations

$N = (P - O) / ||P - O||$



15

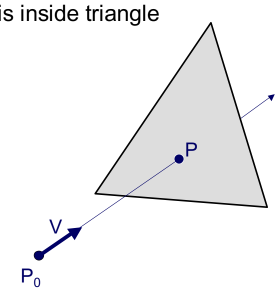
Ray-Scene Intersection

- Intersections with geometric primitives
  - o Sphere
  - » **Triangle**
  - o Groups of primitives (scene)
- Acceleration techniques
  - o Bounding volume hierarchies
  - o Spatial partitions
    - » Uniform grids
    - » Octrees
    - » BSP trees

16

Ray-Triangle Intersection

- First, intersect ray with plane
- Then, check if point is inside triangle



17

Ray-Plane Intersection

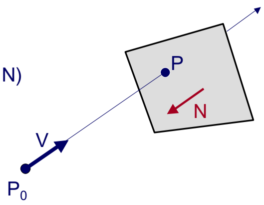
Ray: $P = P_0 + tV$
Plane: $P \cdot N + d = 0$

Algebraic Method

Substituting for P, we get:
$(\mathbf{P_0 + tV}) \cdot N + d = 0$

Solution:
$t = -(P_0 \cdot N + d) / (V \cdot N)$

$P = P_0 + tV$

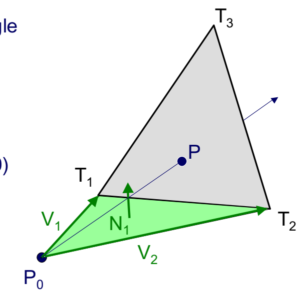


18

Ray-Triangle Intersection I

- Check if point is inside triangle algebraically

For each side of triangle
$V_1 = T_1 - P$
$V_2 = T_2 - P$
$N_1 = V_2 \times V_1$
Normalize N_1
$d_1 = -P_0 \cdot N_1$
if $((P \cdot N_1 + d_1) < 0)$
return FALSE;
end



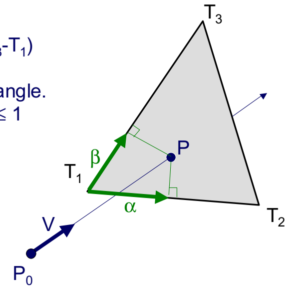
Ray-Triangle Intersection II

19

- Check if point is inside triangle parametrically

Compute α, β:
$P = \alpha (T_2-T_1) + \beta (T_3-T_1)$

Check if point inside triangle.
$0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$
$\alpha + \beta \leq 1$



Other Ray-Primitive Intersections

20

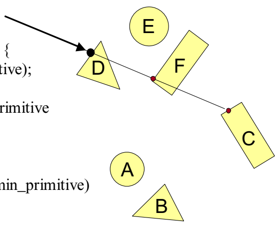
- Cone, cylinder, ellipsoid:
  - o Similar to sphere
- Box
  - o Intersect 3 front-facing planes, return closest
- Convex polygon
  - o Same as triangle (check point-in-polygon algebraically)
- Concave polygon
  - o Same plane intersection
  - o More complex point-in-polygon test

Ray-Scene Intersection

21

- Find intersection with front-most primitive in group

```
Intersection FindIntersection(Ray ray, Scene scene)
{
    min_t = infinity
    min_primitive = NULL
    For each primitive in scene {
        t = Intersect(ray, primitive);
        if (t < min_t) then
            min_primitive = primitive
            min_t = t
        }
    }
    return Intersection(min_t, min_primitive)
}
```



Ray-Scene Intersection

22

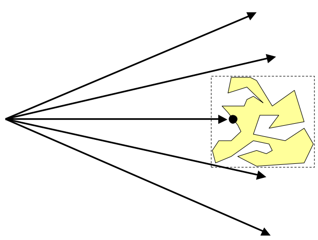
- Intersections with geometric primitives
  - o Sphere
  - o Triangle
  - o Groups of primitives (scene)

» Acceleration techniques
  - o Bounding volume hierarchies
  - o Spatial partitions
    - » Uniform grids
    - » Octrees
    - » BSP trees

Bounding Volumes

23

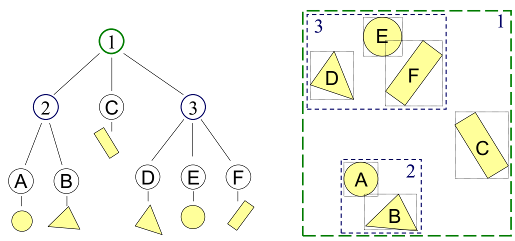
- Check for intersection with simple shape first
  - o If ray doesn't intersect bounding volume, then it doesn't intersect its contents



Bounding Volume Hierarchies I

24

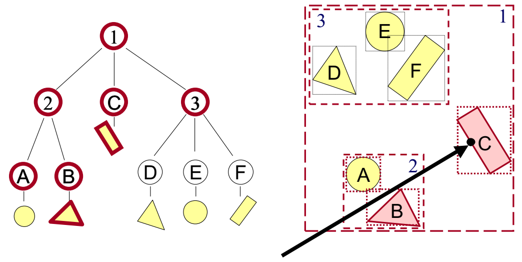
- Build hierarchy of bounding volumes
  - o Bounding volume of interior node contains all children



25

Bounding Volume Hierarchies

- Use hierarchy to accelerate ray intersections
  - o Intersect node contents only if hit bounding volume



26

Bounding Volume Hierarchies III

- Sort hits & detect early termination

```
FindIntersection(Ray ray, Node node)
{
      // Find intersections with child node bounding volumes
      ...
      // Sort intersections front to back
      ...
      // Process intersections (checking for early termination)
      min_t = infinity;
      for each intersected child i {
            if (min_t < bv_t[i]) break;
            shape_t = FindIntersection(ray, child);
            if (shape_t < min_t) { min_t = shape_t;}
      }
      return min_t;
}
```

27

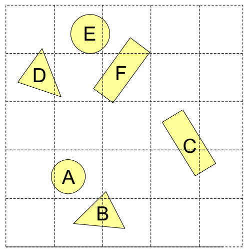
Ray-Scene Intersection

- Intersections with geometric primitives
  - o Sphere
  - o Triangle
  - o Groups of primitives (scene)
- » Acceleration techniques
  - o Bounding volume hierarchies
  - o Spatial partitions
    - » Uniform grids
    - » Octrees
    - » BSP trees

28

Uniform Grid

- Construct uniform grid over scene
  - o Index primitives according to overlaps with grid cells

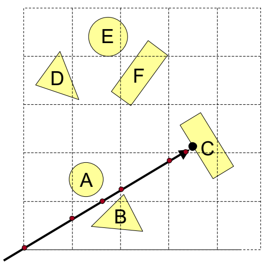


29

Uniform Grid

- Trace rays through grid cells
  - o Fast
  - o Incremental

Only check primitives
in intersected grid cells



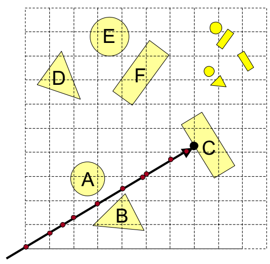
30

Uniform Grid

- Potential problem:
  - o How choose suitable grid resolution?

Too little benefit
if grid is too coarse

Too much cost
if grid is too fine



31

Ray-Scene Intersection

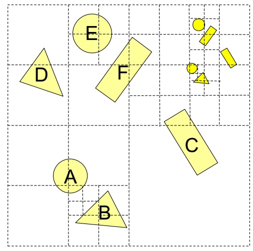
- Intersections with geometric primitives
  - o Sphere
  - o Triangle
  - o Groups of primitives (scene)
- » Acceleration techniques
  - o Bounding volume hierarchies
  - o Spatial partitions
    - » Uniform grids
    - » Octrees
    - » BSP trees

32

Octree

- Construct adaptive grid over scene
  - o Recursively subdivide box-shaped cells into 8 octants
  - o Index primitives by overlaps with cells

Generally fewer cells

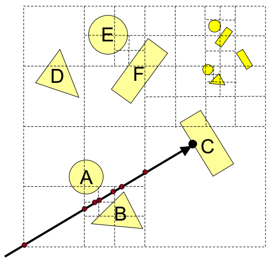


33

Octree

- Trace rays through neighbor cells
  - o Fewer cells
  - o More complex neighbor finding

Trade-off fewer cells for more expensive traversal



34

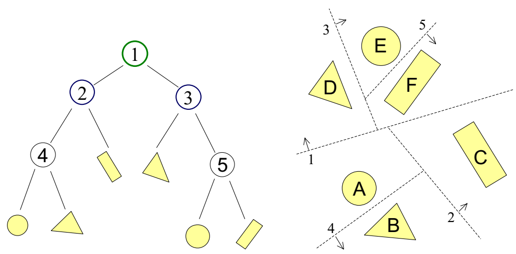
Ray-Scene Intersection

- Intersections with geometric primitives
  - o Sphere
  - o Triangle
  - o Groups of primitives (scene)
- » Acceleration techniques
  - o Bounding volume hierarchies
  - o Spatial partitions
    - » Uniform grids
    - » Octrees
    - » BSP trees

35

Binary Space Partition (BSP) Tree

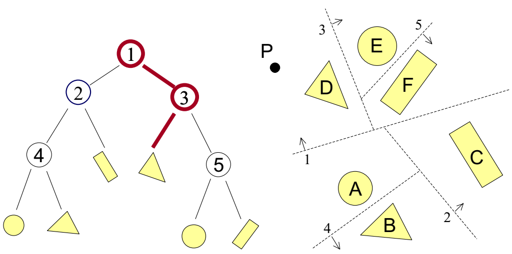
- Recursively partition space by planes
  - o Every cell is a convex polyhedron



36

Binary Space Partition (BSP) Tree

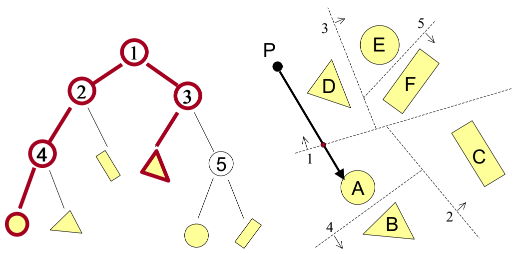
- Simple recursive algorithms
  - o Example: point finding



37

Binary Space Partition (BSP) Tree

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



38

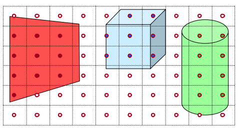
Binary Space Partition (BSP) Tree

```
RayTreeIntersect(Ray ray, Node node, double min, double max)
{
    if (Node is a leaf)
        return intersection of closest primitive in cell, or NULL if none
    else
        dist = distance of the ray point to split plane of node
        near_child = child of node that contains the origin of Ray
        far_child = other child of node
        if the interval to look is on near side
            return RayTreeIntersect(ray, near_child, min, max)
        else if the interval to look is on far side
            return RayTreeIntersect(ray, far_child, min, max)
        else if the interval to look is on both side
            if (RayTreeIntersect(ray, near_child, min, dist)) return …;
            else return RayTreeIntersect(ray, far_child, dist, max)
}
```

39

Other Accelerations

- Screen space coherence
  - o Check last hit first
  - o Beam tracing
  - o Pencil tracing
  - o Cone tracing



- Memory coherence
  - o Large scenes
- Parallelism
  - o Ray casting is “embarassingly parallelizable”
- etc.

40

Acceleration

- Intersection acceleration techniques are important
  - o Bounding volume hierarchies
  - o Spatial partitions
- General concepts
  - o Sort objects spatially
  - o Make trivial rejections quick
  - o Utilize coherence when possible

Expected time is sub-linear in number of primitives

41

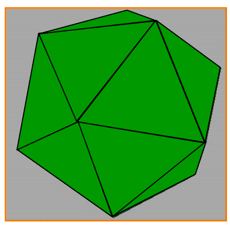
Summary

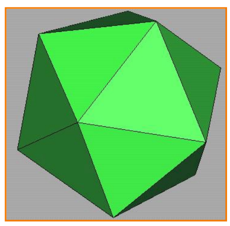
- Writing a simple ray casting renderer is easy
  - o Generate rays
  - o Intersection tests
  - o Lighting calculations

```
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

42

Next Time is Illumination!


Without Illumination


With Illumination