Computer Graphics
CSC 418/2504
Patricio Simari
October 26, 2011

Some slides courtesy of Patrick Coleman
Some figures courtesy of Peter Shirley,
Culling

Back faces

Bounding Volumes

google images
Visibility Review

Backface Culling

Clipping

Shape Occlusion
BSP Tree Review

\[ f_1(p) \leq 0 \]

\[ f_1(p) > 0 \]
Drawing a BSP Tree

Principle: Draw the opposite side of the eye first.

\[ f_1(e) \leq 0 \]

draw right subtree

draw \( S_1 \)

draw left subtree

\[ f_1(e) > 0 \]

draw left subtree

draw \( S_1 \)

draw right right subtree
class Node():
    Shape shape
    Node left
    Node right

class Tree():
    Node root
    draw()
    add()

Tree.draw(n, e):
    if empty(node):
        return
    if f_n(e) <= 0:
        self.draw(n.right)
        draw n.shape
        self.draw(n.left)
    else:
        self.draw(n.left)
        draw n.shape
        self.draw(n.right)
Creating a BSP Tree

Given a shape sequence \([S_1, S_2, ..., S_n]\)

tree.root = node(S_1)
for i in [2, n]:
  tree.add(S_i, tree.root)

<table>
<thead>
<tr>
<th>Split s</th>
<th>Add left</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (f_n(p) \leq 0 \ \forall p \in s):</td>
<td></td>
</tr>
<tr>
<td>if n.left is None:</td>
<td></td>
</tr>
<tr>
<td>node.left = Node(s)</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>self.add(s, node.left)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Add right</th>
</tr>
</thead>
<tbody>
<tr>
<td>else if (f_n(p) \geq 0 \ \forall p \in s):</td>
</tr>
<tr>
<td>if n.right is None:</td>
</tr>
<tr>
<td>node.right = Node(s)</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>self.add(s, node.right)</td>
</tr>
</tbody>
</table>

else:
split s into s_a and s_b s.t.:
\[ f_n(p) \geq 0 \ \forall p \in s_b \]
\[ f_n(p) \leq 0 \ \forall p \in s_a \]
self.add(s_a, n)
self.add(s_b, n)
Creating a BSP Tree

Tree.add(s, n):

- if $f_n(p) \leq 0 \forall p \in s$:
  - if n.left is None:
    - node.left = Node(s)
  - else
    - self.add(s, node.left)
- else if $f_n(p) \geq 0 \forall p \in s$:
  - if n.right is None:
    - node.right = Node(s)
  - else
    - self.add(s, node.right)
- else:
  - split s into s_a and s_b s.t.:
    - $f_n(p) \geq 0 \forall p \in s_b$
    - $f_n(p) \leq 0 \forall p \in s_a$
  - self.add(s_a, n)
  - self.add(s_b, n)
Scan Conversion

Discretize a polygon in image space to the image samples (pixels)
Scan Conversion

Discretize a polygon in image space to the image samples (pixels)
Triangle Scan Conversion

for each pixel \((x, y)\):
  if \((x, y)\) inside triangle:
    \(\text{image}(x, y) = \text{compute color}\)
Barycentric Coordinates

$$\mathbf{p} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c}$$

$$\alpha + \beta + \gamma = 1$$

**Inside test:**

$$\alpha \geq 0, \beta \geq 0, \gamma \geq 0$$
Computing Barycentric Coordinates

\[ \mathbf{p} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c} \]
\[ \alpha + \beta + \gamma = 1 \]

\[ \alpha = \frac{\text{area}(A)}{\text{area}(\Delta)} \]
\[ \beta = \frac{\text{area}(B)}{\text{area}(\Delta)} \]
\[ \gamma = \frac{\text{area}(C)}{\text{area}(\Delta)} \]
Triangle Scan Conversion

Efficiency: Bound the triangle.

for y in [y_min, y_max]:
    for x in [x_min, x_max]:
        if (x, y) inside triangle:
            image(x, y) = compute color

There are even more efficient approaches.
There are algorithms for general polygons.
Z-Buffer

Handle shape occlusion during scan conversion.
Z-Buffer

Handle shape occlusion during scan conversion.
Z-Buffer

Array of Color
Initialize to Black

Array of $z$
Initialize to $\infty$
Scan Conversion with the Z-Buffer

for each pixel (x, y):
    if (x, y) inside triangle:
        z = z(x, y)
        if z < zBuffer(x, y):
            zBuffer(x, y) = z
            image(x, y) = compute color
Rasterization: All Together

Define Geometry

Object Space

World Space

Camera Space

Backface Culling

BSP Tree

Scan Conversion

Z-Buffer

Clipping to the CVV

Appearance Model Evaluation

\[ M_{OW} \]

\[ M_{WC} \]

\[ M_{CN} \]

\[ M_{NI} \]
Rasterization

Efficient

Complicated

Ideas show up in many other places

Ray Tracing

Slow

Much Simpler

Ideas show up in many other places
Ray Tracing

for each pixel:
  for each object:
    if the object is visible in the pixel:
      compute the color and draw it
Ray Tracing

Image parameterized by $(u, v)$
Ray Tracing
Ray Tracing
Ray Tracing
Ray Tracing
Ray Tracing

Questions:
• Constructing Rays
• Calculating intersections
• Determining the closest intersection
• Transforming objects
Ray construction

\[ p(t) = e + t(s - e) \]
Implicit surface

- ray: \( p(t) = e + t(s - e) \)
- surface: \( f(p) = 0 \)
- plug in value
  \[ f(p) = 0 \]
  \[ \Rightarrow f(e + t(s - e)) = 0 \]
- solve for \( t \)
- if solution exists then ray hits and plugging the value back in \( p(t) \) will give the point of intersection
Parametric surface

- \( \mathbf{e} = (e_x, e_y, e_z) \)
- \( \mathbf{d} = \mathbf{s} - \mathbf{e} = (d_x, d_y, d_z) \)
- \( f(u,v) = (f_x(u,v), f_y(u,v), f_z(u,v)) \)
- System of 3 equations in 3 unknowns:
  \[
  \begin{align*}
  e_x + td_x &= f_x(u,v) \\
  e_y + td_y &= f_y(u,v) \\
  e_z + td_z &= f_z(u,v)
  \end{align*}
  \]
- Solve for \( t, u, v \). If solution exists, ray hits and plugging values back gives point of intersection.
Instancing

1. Scale
2. Rotate
3. Move
Instancing

points $Mp$ on circle

ray $M^{-1}a + tM^{-1}b$

points $p$ on circle
Instancing

```cpp
instance::hit(ray a + t b, real t0, real t1, hit-record rec)
ray r' = M⁻¹ a + t M⁻¹ b
if (base-object → hit(r', t0, t1, rec)) then
  rec.n = (M⁻¹)ᵀ rec.n
  return true
else
  return false
```