cs155 - z sweedyk

graphics pipeline systems

who's who

end user

application program

graphics pipeline

who's who for today

user

graphics pipeline

user defined scene description

• models
• lights
• view (eye/camera)

graphics pipeline

1. Build scene
2. Clip
3. Project
4. Scan convert

graphics pipeline 1

1. build scene
2. clip

3. project

4. scan convert

vertices in view plane

frame buffer

1. Build scene
2. Clip
3. Project
4. Scan convert

1. Build scene
2. Clip
3. Project
4. Scan convert

vertex operations

User described primitives
Object coordinates

v₀ = (0,0,0), v₁ = (1,0,0), v₂ = (1,1,0), v₃ = (0,1,0)
build scene

pipeline representation is in homogeneous coordinates

\[ \begin{align*}
  v_0 &= (0,0,0,1), \\
  v_1 &= (1,0,0,1), \\
  v_2 &= (1,1,0,1), \\
  v_3 &= (0,1,0,1)
\end{align*} \]

primitives

- points
- line segments
- polygons

modeling transforms

- scale
- rotate
- translate

Lights

- Ambient
- Directional
- Point
- Spot
build scene

- User-defined viewpoint
- View coordinates
- Vertex in view coordinates: $M_vM_wv$
- Lights in view coordinates: $M_l$, $M_d$

view in world coordinates

- User-defined viewpoint
- View coordinates
- Vertex in view coordinates: $M_vM_wv$
- Lights in view coordinates: $M_l$, $M_d$

world ↔ view coordinates

- Translate & rotate: $M_v = M_wM_T$
- Right: $r$
- Up: $u$
- Toward: $t$
- Rotation:
  - $M_r = (1,0,0)^T$
  - $M_u = (0,1,0)^T$
  - $M_t = (0,0,-1)^T$

world ↔ view coordinates

- Translate by $-p_x, -p_y, -p_z$
- Right: $r$
- Up: $u$
- Toward: $t$
- Rotation:
  - $r = M_w(1,0,0)^T$
  - $u = M_w(0,1,0)^T$
  - $t = M_w(0,0,-1)^T$
build scene

vertex in view coordinates: $M_vM_wv$
lights in view coordinates: $M_vp, M_vd$

geometric primitives

object coordinates: $v$
description of vertex
world coordinates: $M_wv$
description of vertex situated in world
view coordinates: $M_vM_wv$
description of vertex in world as seen from a particular viewpoint

lights

world coordinates: $p, d$
description of light position/direction in world
view coordinates: $M_vp, M_vd$
description of light position/direction in world as seen from a particular viewpoint
note: $M_vd$ is shorthand for the "multiply vector" operation we've used before!

graphics pipeline

1. Build scene
2. Clip
3. Project
4. Scan convert
Done

graphics pipeline 2

user specified view volume
when to clip?

1. Build scene
2. Clip
3. Project
4. Scan convert

2d clipping
3d clipping
2d clipping
3d clipping
easy 3d clipping
scissoring

when to clip?

1. Build scene
2. Clip
3. Project
4. Scan convert

3d clipping

this is most general ... we'll say more about the other approaches later

graphics pipeline 2

user specified view volume

projection type & view volume

orthographic

perspective

view window

orthographic

perspective

view window specification

view window (in view coordinates):
axes aligned rectangle
- in view plane \(z = z_{\text{near}}\)
- centered at \(z\) axis
- with boundaries \(x = x_{\text{left}}, x = x_{\text{right}}, y = y_{\text{bottom}}, y = y_{\text{top}}\)
orthographic view volume

axes aligned parallelepiped

perspective view volume (frustum)

frustum centerline aligned with z_{view} axis

3d clipping

eliminate "outside" primitive

orthographic

perspective

bounding plane description

1. point on plane
2. inward-pointing normal

orthographic bounding planes
point on plane

\((x_{\text{left}}, y_{\text{top}}, z_{\text{near}})\)

inward-pointing normal

inward pointing normal \(n: <1,0,0>\)

perspective bounding planes

\(z = z_{\text{near}}\)

\(z = z_{\text{far}}\)

perspective bounding planes

view volume bounding plane

plane containing

\((0,0,0),\)

\((x_{\text{left}}, y_{\text{bottom}}, z_{\text{near}}),\)

\((x_{\text{left}}, y_{\text{top}}, z_{\text{near}})\)

inward-pointing normals

plane containing

\((0,0,0),\)

\((x_{\text{left}}, y_{\text{bottom}}, z_{\text{near}}),\)

\((x_{\text{left}}, y_{\text{top}}, z_{\text{near}})\)

with inward pointing normal \(n = w \times v\)

\(n = w \times v\)
clipping plane

clipping plane specification:
- point q on the plane
- inward pointing normal n

3d clipping

given a clipping plane and a graphics primitive
return “in-side primitive”

vertex clipping

p is in with respect to the clipping plane iff n \cdot v \geq 0 where
- n is the inward facing normal
- v is the vector from q to p

3d clipping

- vertex clipping
- line segment clipping
- polygon clipping

line segment clipping

use test for vertex clipping

Classify endpoint p₀ & p₁
Case: p₀ & p₁ in ____
Case: p₀ & p₁ out ____
Case: p₀ in & p₁ out ____
Case: p₀ out & p₁ in ____

p₀
p₁
line segment clipping

Case $p_0$ & $p_1$ in:
return $(p_0, p_1)$

Case $p_0$ & $p_1$ out:
return null

Case $p_0$ in & $p_1$ out:
return $(p_0, p')$

Case: $p_0$ out & $p_1$ in:
return $(p', p_1)$
do you know how to compute $p'$?

Color at $p'$:
interpolate

out-code optimization

eliminate unnecessary intersection computations

Endpoint $p$ has out-code $b_0b_1…b_5$:
- $b_i=0$ if $p$ is inside plane $i$
- $b_i=1$ else
out-code optimization

- compute endpoint out-codes B and B'
- if $B \land B' \neq 0$ return ________
- if $B \lor B' = 1$ return ________
- else clip against plane j where the $i^{th}$
  ____________________________
- restart test

3d clipping

- vertex clipping
- line clipping
- polygon clipping

polygon clipping

1. classify vertices

2. compute intersection
   points of intersecting edges
   & write out new polygon

polygon clipping

• if $v_0$ is in then write $v_0$
• for $i=1\ldots n-1$
  - case $v_i$ & $v_{i+1}$ in: write $v_{i+1}$
  - case $v_i$ & $v_{i+1}$ out: do nothing
  - case $v_i$ in and $v_{i+1}$ out: write intersection point
  - case $v_i$ out and $v_{i+1}$ in: write intersection point
  and $v_{i+1}$
  indices taken modulo n

example

$v_0$ out: do nothing
example

\[ \begin{align*} v_0 & \text{ & } v_1 \text{ out: do nothing} \\ v_0 & \text{ & } v_3 \text{ in: write } v, v_3 \end{align*} \]

example

\[ \begin{align*} v_1 \text{ & } v_2 \text{ out: do nothing} \\ v_3 & \text{ in & } v_0 \text{ out: write } v' \end{align*} \]

day polygon clipping

- if \( v_0 \) is in then write \( v_0 \)
- for \( i = 1 \ldots n-1 \)
  - case \( v_i \) & \( v_{i+1} \) in: write \( v_{i+1} \)
  - case \( v_i \) & \( v_{i+1} \) out: do nothing
  - case \( v_i \) in and \( v_{i+1} \) out: write intersection point
  - case \( v_i \) out and \( v_{i+1} \) in: write intersection point and \( v_{i+1} \)
    interpolate along edge to find color at intersection point

day graphics pipeline

1. Build scene
2. Clip
3. Project
4. Scan convert

Done