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

user defined scene description

• models
• lights
• view (eye/camera)
2. clip

3. project

vertices in view plane

frame buffer

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

User described primitives
Object coordinates

v₀ = (0,0,0), v₁ = (1,0,0), v₂ = (1,1,0), v₃ = (0,1,0)
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: $M_p, M_d$

view in world coordinates

view in world coordinates

up $u (u \perp t)$
toward $t$
right $r = t \times u$
viewpoint $p$

world and view coordinates

world ↔ view coordinates

translate & rotate

world ↔ view coordinates

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: \( M_p, M_d \) — description of light position/direction in world as seen from a particular viewpoint

Note: \( M_d \) 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?**

*Ease vs. pipeline efficiency*

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

2D clipping

3D clipping

Scissoring
graphics pipeline 2

projection type & view volume

user specified view volume

orthographic

perspective

view window

view window specification

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

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

project 3 default

orthographic view volume

axes aligned parallelepiped

viewpoint \((0,0,0)\)
project 3 default

perspective view volume (frustum)

project 3: default frustum

3d clipping

bounding plane description
orthographic bounding planes

point on plane

inward-pointing normal

perspective bounding planes

view volume bounding plane
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 \( n = w \times v \)

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 \( p \)
Line segment clipping

Use test for vertex clipping

Classify endpoint $p_0$ & $p_1$

Case: $p_0$ & $p_1$ in

Case: $p_0$ & $p_1$ out

Case: $p_0$ in & $p_1$ out

Case: $p_0$ out & $p_1$ in

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
out-code optimization

- assume k clipping planes
- endpoint p has out-code \( b_0, b_1, \ldots, b_{k-1} \)
- \( b_i = 0 \) if p is inside plane i
- \( b_i = 1 \) else

3d clipping

- vertex clipping
- line clipping
- polygon clipping

polygon clipping

1. classify vertices
2. compute intersection points of intersecting edges
   - 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 \( v_{i+1} \) out and \( v_i \) in: write intersection point
   and \( v_i \)
   - write out new polygon

indices taken modulo n
example

\[ v_0 \text{ out: do nothing} \]

\[ \begin{array}{c}
  v_3 \\
  v_2 \\
  v_1 \\
  v_0 \\
\end{array} \]

example

\[ v_0 \& v_1 \text{ out: do nothing} \]

\[ \begin{array}{c}
  v_3 \\
  v_2 \\
  v_1 \\
  v_0 \\
\end{array} \]

example

\[ v_1 \& v_2 \text{ out: do nothing} \]

\[ \begin{array}{c}
  v_3 \\
  v_2 \\
  v_1 \\
  v_0 \\
\end{array} \]

example

\[ v_2 \text{ out} \& v_3 \text{ in: write } v, v_3 \]

\[ \begin{array}{c}
  v_3 \\
  v_2 \\
  v_1 \\
  v_0 \\
\end{array} \]

example

\[ v_3 \text{ in} \& v_0 \text{ out: write } v' \]

\[ \begin{array}{c}
  v_3 \\
  v_2 \\
  v_1 \\
  v_0 \\
\end{array} \]

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 \( v_{i+1} \)

\text{interpolate along edge to find color at intersection point}