1. Build scene
2. Projection transform
3. Clip
4. Perspective division
5. Viewport transform
6. Scan convert

User described primitives
Object coordinates

\[ v_0 = (0,0,0), \quad v_1 = (1,0,0), \quad v_2 = (1,1,0), \quad v_3 = (0,1,0) \]

\[ v_0 = (0,0,0,1), \quad v_1 = (1,0,0,1), \quad v_2 = (1,1,0,1), \quad v_3 = (0,1,0,1) \]
primitives

- points
- line segments
- polygons

build scene

User described modeling transforms
World coordinates

Vertex in world coordinates: \( M_{W}v \)

modeling transforms

- scale
- rotate
- translate

build scene

User described lights
(Usually)
World coordinates

Lights

- Ambient
- Directional
- Point
- Spot

Exercise (teams of 2)

- Vertex in object coordinates: \( (1,1,2) \)
- Scale by 2 in x
- Translate by 3 in x and -4 in z
- What is result?
- Write sequence of modeling transform
- Multiply to get composite transform
- Multiply to get vertex in world coordinates
- Put your matrix on the board
build scene

User defined viewpoint
View coordinates:
vertex in view coordinates: \( M_v \)
lights in view coordinates: \( M_l \)

view in world coordinates

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

world ↔ view coordinates

translate & rotate: \( M_V = M_B M_T \)

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

\( M_T \): translate by \( (p_x, p_y, p_z) \)

right \( r \)
up \( u \)
toward \( t \)

rotation
\( r = M_V (1,0,0)^T \)
\( u = M_V (0,1,0)^T \)
\( t = M_V (0,0,-1)^T \)
Exercise cont.

- The viewer is at (4,0,-2) looking in the (1,0,0) direction. Up is (0,1,0).
- What is our vertex in view coordinates?
- Write the translation and rotation matrices needed to convert to viewpoint coordinates. (You should be able to compute the inverse matrix by inspection.)
- Multiply to create the composite view transform.
- Multiply to get point in view coordinates.
- Write your view transform on the board.

build scene

vertex in view coordinates: $M_{WV}$
lights in view coordinates: $M_v$, $M_d$

geometric primitives

object coordinates: $v$

world coordinates: $M_W v$

view coordinates: $M_{WV}$

description of vertex

description of vertex in world

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: $M_vp$, $M_d$

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!

Application Programmer

```
Primitive in object coordinates

 glBegin(gTriangles);
 glVertexf(-,-,-);
 glVertexf(-,-,-);
 glVertexf(-,-,-);
 glEnd();
```

see Woo for details!

Application Programmer

```
Primitive in world coordinates

 glScalef(2,1,1)
 glTranslatef(10,0,0);
 glBegin(gTriangles);
 glVertexf(-,-,-);
 glVertexf(-,-,-);
 glVertexf(-,-,-);
 glEnd();
```

see Woo for details!
Application Programmer I

Primitive in viewpoint coordinates

Viewpoint Transforms

- glScalef(1.0, 1.0, 1.0)
- glTranslatef(0.0, 0.0, 0.0)
- glBegin(gTriangles);
- glVertexf(-1.0, -1.0, -1.0);
- glVertexf(1.0, 1.0, 1.0);
- glVertexf(-1.0, -1.0, 1.0);
- glEnd();

Application Programmer II

Primitive in viewpoint coordinates

viewpoint description

- gluLookAt(-1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)
- glScalef(1.0, 1.0, 1.0)
- glTranslatef(0.0, 0.0, 0.0)
- glBegin(gTriangles);
- glVertexf(-1.0, -1.0, -1.0);
- glVertexf(1.0, 1.0, 1.0);
- glVertexf(-1.0, -1.0, 1.0);
- glEnd();

see Woo for details!

Modelview Matrix

- glLoadIdentity()
- gluLookAt(-1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)
- glPushMatrix()
- glScalef(1.0, 1.0, 1.0)
- glTranslatef(0.0, 0.0, 0.0)
- glBegin(gTriangles);
- glVertexf(-1.0, -1.0, -1.0);
- glVertexf(1.0, 1.0, 1.0);
- glVertexf(-1.0, -1.0, 1.0);
- glEnd();
- glPopMatrix();

Graphics pipeline

1. Build scene
2. Projection transform
3. Clip
4. Perspective division
5. Viewport transform
6. Scan convert

User's description of scene

1. Build scene
2. Projection transform
3. Clip
4. Perspective division
5. Viewport transform
6. Scan convert
graphics pipeline – with a twist

1. Build scene
2. Projection transform
3. Clip
4. Perspective division
5. Viewport transform
6. Scan convert

This really is part of the projection step.

Projection/Perspective Division

1. Compute projected 2d vertices in canonical world
2. Compute relative depth info

Projection mode

- Orthographic projection
- Perspective projection

orthographic projection

\[(x,y,z,1) \rightarrow (x',y',z',1)\]

where \((x',y')\) is the vertex projected into a canonical 2x2 view window and \(z'\) is vertex's relative depth based on a canonical 2x2x2 world

NOTE: for orthographic all we do is convert to the canonical world
1. translate by \((-\frac{r+l}{2}, -\frac{t+b}{2}, \frac{f+n}{2})\)
2. scale by \(\left(\frac{2}{r-l}, \frac{2}{t-b}, \frac{2}{f-n}\right)\)
3. scale by \((0,0,-1)\)

**Canonical world transform**

\[M_{C} = \begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\
0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\
0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
orthographic projection matrix

\[
M_p = \begin{pmatrix}
\frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\
0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\
0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

orthographic projection

\[M_p : (x,y,z,1) \rightarrow (x',y',z',1)\]

where \((x',y')\) is the vertex projected into a canonical 2x2 view window and \(z'\) is vertex's relative depth based on a canonical 2x2x2 world

Projection mode

- Orthographic projection
- Perspective projection

Projection/Perspective Division

1. Compute projected 2D vertices in canonical world
2. Compute relative depth info

Perspective mode

1. Compute projected 2D vertices
2. Preserve relative depth info
3. Convert to canonical world

Perspective view volume

The frustum centerline is aligned with the z axis.
perspective view volume (frustum)

in view coordinates

view window

viewpoint

(0,0,0)+z

(r,t,-n)

perspective projection

(xp,yp,-n)

(0,0,0)

2a. compute projected 2d vertices: (-xn/z, -yn/z)

2b. preserve relative depth information: z

Mp1 = ?

what is wrong with this picture

\[
\begin{pmatrix}
-n/z & 0 & 0 & 0 \\
0 & -n/z & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
= 
\begin{pmatrix}
-xn/z \\
-yn/z \\
z \\
1
\end{pmatrix}
\]

perspective Mp1

graphics pipeline
perspective $M_{p_i}$

A little trick!

\[
\begin{pmatrix}
  n & 0 & 0 & 0 \\
  0 & n & 0 & 0 \\
  0 & 0 & a & b \\
  0 & 0 & -1 & 0
\end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix} = \begin{pmatrix}
  nx \\
  ny \\
  az + b \\
  -z
\end{pmatrix}
\]

We'll specify $a$ & $b$ in a moment.

Remember scale factor - $z$ in w-component.

Later we'll normalize by w-component of vertex.

Later is step 4

This is where the relative depth info will be.

-b/z is (almost) as good as $z$ for any nonzero constant $b$ so let $a=0$ and $b=\text{fn}$.

If the vertex has been clipped then $z$ takes on values in $[-n,-f]$.

-fn/z takes on values in $[n,f]$.

(We switched to right-handed coordinates... we'll fix this in a minute.)
perspective projection

\[(x,y,z) \rightarrow (-xn/z, \ -yn/z, \ -fn/z)\]

depth-dependent scale

perspective projection

1. \(M_{p1}\)
2. perspective division
3. orthographic \(M_{p2}\)

this would work

alternative

1. \(M_{p1}\)
2. orthographic \(M_{p2}\)
3. perspective division

does this work?

sure – perspective division is multiplication by a scalar
(albeit a special one)

perspective projection matrix

\[
\begin{pmatrix}
\frac{2}{r-l} & 0 & 0 & -(r+l)/(r-l) \\
0 & \frac{2}{t-b} & 0 & -(t+b)/(t-b) \\
0 & 0 & -\frac{2}{f-n} & -(f+n)/(f-n) \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]

except we’re back in left-handed coordinates

projection matrix: \(M_p\)

\[
\begin{pmatrix}
\frac{-2n}{r-l} & 0 & (r+l)/(r-l) & 0 \\
0 & -\frac{2n}{t-b} & (t+b)/(t-b) & 0 \\
0 & 0 & -(f+n)/(f-n) & -2fn/(f-n) \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]
geometric primitives

- object coordinates: \( v \) — description of vertex
- world coordinates: \( M_w v \) — situated in world
- view coordinates: \( M_M v \) — description of vertex in world as seen from a particular viewpoint
- clip coordinates: \( M_M M_w v \) — description of vertex seen from viewpoint in a normalized world

Application programmer

Define projection mode and view volume.
Example:
```c
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(\( \theta \), \( \rho \), near, far);  
glMatrixMode(GL_MODELVIEW);
```

\( \theta \) = \( \omega \) = \( \frac{2}{h} \times \tan(\frac{\theta}{2}) \)

Application programmer

Define projection mode and view volume.
Example:
```c
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(\( \theta \), \( \rho \), near, far);  
glMatrixMode(GL_MODELVIEW);
```

```c
glFrustum(...)  
or  
glOrtho(...)
```

viewport transformation

(right-handed)

\((x,y,z,1)\) projected 2d coordinates

display coordinates
viewport transformation

specifies how the projected world is mapped to the display window

viewport transformation

display window: set by end-user

aspect ratio may be different

viewport transform

2D projected coordinates

display coordinates

We'll call this $M_V$

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

clip coordinates: $M_pM_vM_wv$
description of vertex seen from viewpoint in a normalized world

display coordinate $M_pM_vM_wv$
description of a vertex in display coordinates
Application Programmer

- Object coordinates: \( v \)
- World coordinates: \( M_wv \)
- View coordinates: \( M_vM_wv \)
- Clip coordinates: \( M_pM_vM_wv \)
- Display coordinate: \( M_dM_pM_vM_wv \)

Description of vertex

- Object → world specified by transforms in scene graph
- View transform specified where the viewer (view volume) located/oriented
- Projection transform specified by description of the view volume
- Viewport transform specified by viewport command

Initialization

- Create window (use GLUT)
- Define projection mode and view volume
- Define viewport transform

Display function: draws scene

Display Function:
- Clear frame and depth buffers
- Load Identity (default is Modelview mode)
- Load view transform
- Traverse scene graph
- Swap buffers

Push Matrix()
- Local Transforms
- Draw local model
- Visit children
- Pop Matrix()
- Return to parent node

Resize

1. May change view volume
2. Change viewport
Now do it!