**Projection ABC**

2. **Projection transform**
   - Linear transformation

2a. Compute projected 2d vertices
2b. Preserve relative depth info

---

**Graphics Pipeline**

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

- Why keep depth info?
- Need relative depth info

---

**This may seem like the way to do it**

- Need relative depth info
10/27/2003

scan conversion

1. Build scene
2. Projection transform

need relative depth

3. Scan convert

4. Perspective division
5. Viewport transform
6. Scan convert

graphics pipeline

user's description of scene

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

this may seem like the way to do it
but it doesn't eliminate the need to keep depth info

10/27/2003

linear transforms can be collapsed into 1

\[ v \rightarrow Mv \text{ where } M = M_p M_v M_w \]

remain fixed for all vertices in scene

10/27/2003

Projection ABC

linear transformation

1. Build scene
2. Projection transform

user's description of scene

4. Perspective division
5. Viewport transform
6. Scan convert

optimized for "canonical world"

10/27/2003

optimized by use of hierarchical coordinates & matrix stacks
2. Projection transform
   2a. Compute projected 2d vertices
   2b. Preserve relative depth info

   Linear transformation
   $\mathbf{u} \rightarrow \mathbf{M}_p \mathbf{M}_w \mathbf{u}$ or the shorthand
   $\mathbf{u} \rightarrow \mathbf{M}_p \mathbf{u}$

actually … there are 2 types of projection

- perspective
- orthographic

orthographic projection

perspective projection

actually … there are 2 types of projection

- perspective
- orthographic
orthographic view volume

axes aligned parallelepiped

view window

viewpoint (0,0,0)

z_{view}

orthographic projection

2a. compute projected 2d vertices: (x,y)
2b. preserve relative depth information: z

M_{p1} = I

1. translate by (-(r+l)/2, -(t+b)/2, (f+n)/2)
2. scale by (2/(r-l), 2/(t-b), 2/(f-n))

center at origin

left-handed

1. translate by (-c(1/2), -c(1/2), c(1/2))

scale to 2x2x2

left-handed

2. scale by (2/(r-l), 2/(t-b), 2/(f-n))
change handed-ness

1. scale by \((0,0,-1)\)
2. \(\{(r,l)/2, (t+b)/2, (f+n)/2\}\)

orthographic projection matrix

\[
M_{p2} = \begin{pmatrix}
\frac{2}{(r-l)} & 0 & 0 & -\frac{(r+l)}{(r-l)} \\
0 & \frac{2}{(t-b)} & 0 & -\frac{(t+b)}{(t-b)} \\
0 & 0 & -\frac{2}{(f-n)} & -\frac{(f+n)}{(f-n)} \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

orthographic projection

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

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

actually ... there are 2 types of projection

- perspective
- orthographic
Projection ABC

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

perspective view volume

frustum centerline aligned with z axis

perspective view volume (frustum)

view window

perspective projection

perspective projection

\[ \frac{x_p}{x} = \frac{-n}{z} \]

2a. Compute projected 2d vertices: \((-xn/z, -yn/z, -n)\)
2b. Preserve relative depth info: \(z\)
**perspective $M_{p_i}$**

**what is wrong with this picture**

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

$M_{p_i}$?

---

**graphics pipeline**

user's description of scene

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

linear transforms can be collapsed into 1

remain fixed for all vertices in scene

---

**perspective $M_{p_i}$**

**a little trick!**

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

we'll specify $a$ & $b$ in a moment

remember scale factor $-z$ in $w$-component

---

**graphics pipeline**

user's description of scene

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

later is step 4

---

**perspective $M_{p_i}$**

**a little trick!**

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

later we'll normalize by $w$-component of vertex

---

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 $b=fn!$
perspective $M_{p1}$

A little trick!

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f_n & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ fn \\ -z \end{pmatrix} \begin{pmatrix} -nx/z \\ -ny/z \\ -fn/z \\ 1 \end{pmatrix}$$

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

We switched to right-handed coordinates, we'll fix this in a minute.

Projection ABC

Linear transformation

2. Perspective transform

3. Perspective division

4. Compute projected 3D vertices

5. Convert to canonical world

Perspective projection

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

Depth dependent x,y scale

$z = z_0$ to $z = z_1$

Depth-dependent scale

Perspective projection

1. $M_{p1}$
2. Perspective division
3. Orthographic $M_{p2}$

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

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

except we're back in left-handed coordinates

projection matrix: \( M_P \)

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

graphic primitives

object coordinates: \( v \)

description of vertex

world coordinates: \( M_w v \)

description of vertex situated in world

view coordinates: \( M_v M_w v \)

description of vertex in world as seen from a particular viewpoint

clip coordinates:

\( M_p M_v M_w v \)

description of vertex seen from viewpoint in a normalized world

graphic pipeline

user's description of scene

1. Build scene

2. Projection transform

3. Clip

4. Perspective division

5. Scan convert

can we clip before perspective division?

yes – we’ll come back to this

user's description of scene

1. Build scene

2. Projection transform

3. Clip

4. Perspective division

5. Scan convert

\((x,y,z,w) \rightarrow (x/w,y/w,z/w,1)\)
viewport transformation

right-handed

$(-1, 1, -1)$

projected 2d coordinates

image coordinates

viewport transformation

view window

display window:
set by end-user

aspect ratio may be different

viewport transformation

view window

or this

viewport transformation

generates geometric primitives

object coordinates: $v$
world coordinates: $M_{wv}$
view coordinates: $M_{vmwv}$
clip coordinates: $M_{mpmvmwv}$
image coordinates

description of vertex

description of vertex situated in world

description of vertex as seen from a particular viewpoint

description of vertex seen from viewpoint in a normalized world

description of a vertex in image coordinates

2d projected
coordinates

image coordinates

We'll call this $M_{c}$
The graphics pipeline involves the following steps:

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

Let's break down each step:

- **Scene Description**: The user's description of the scene.
- **Projection Transform**: Converts 3D scene coordinates to 2D projection space.
- **Clip**: Clips objects that are outside the viewing volume.
- **Perspective Division**: Divides coordinates by depth to achieve perspective correctness.
- **Viewport Transform**: Transforms coordinates into the viewport's coordinate system.
- **Scan Convert**: Converts vertices into image coordinates and draws them into the frame buffer.

### Who's Who

- **End User**: The person using the application program.
- **Application Program**: The software that interacts with the graphics pipeline.
- **Graphics Pipeline**: The components that transform the scene into a drawable format.

### Application Program

- **Primitives & Modeling Transforms (M_w)**: Transforms objects into the world coordinate system.
- **Viewpoint (M_v)**: Places the viewpoint in the scene.
- **Projection Type & View Volume (M_p)**: Specifies the projection type and view volume.
- **Viewport Transformation (M_T)**: Transforms image coordinates into the viewport's coordinate system.

For next week, we'll return to clipping and scan conversion. For now, let's look at the pipeline transforms in OpenGL.