View Frustum Clipping

Don Fussell
Computer Science Department
The University of Texas at Austin
A Simplified Graphics Pipeline

1. Application
2. Vertex batching & assembly
3. Triangle assembly
4. Triangle clipping
5. NDC to window space
6. Triangle rasterization
7. Fragment shading
8. Depth testing
9. Color update
10. Framebuffer

University of Texas at Austin    CS354 - Computer Graphics     Don Fussell
A few more steps expanded

1. **Application**
   - Vertex batching & assembly
     - Vertex transformation
     - Lighting
       - Texture coordinate generation
     - View frustum clipping
     - Perspective divide
     - User defined clipping
     - NDC to window space
     - Back face culling
     - Triangle rasterization
     - Triangle assembly
   - Fragment shading
     - Depth testing
       - Depth buffer
     - Color update
       - Framebuffer

University of Texas at Austin    CS354 - Computer Graphics    Don Fussell
Conceptual Vertex Transformation

```
glomer* API commands
object-space coordinates (x_o, y_o, z_o, w_o)

Modelview matrix
eye-space coordinates (x_e, y_e, z_e, w_e)

User-defined clip planes
client eye-space coordinates (x_e, y_e, z_e, w_e)

Projection matrix
clip-space coordinates (x_c, y_c, z_c, w_c)

View-frustum clip planes
clipped clip-space coordinates (x_c, y_c, z_c, w_c)

Perspective division

Viewport + Depth Range transformation
normalized device coordinates (NDC) (x_n, y_n, z_n, 1/w_c)

window-space coordinates (x_w, y_w, z_w, 1/w_c)
to primitive rasterization
```
OpenGL Perspective

```c
glFrustum(left, right, bottom, top, near, far)
```
Consider a simple perspective with the COP at the origin, the near clipping plane at $z = -\text{near}$, and a 90 degree field of view determined by the planes $x = \pm z, y = \pm z$.
Generalization

\[
N = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \alpha & \beta \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\]

after perspective division, the point \((x, y, z, 1)\) goes to

\[
x' = \frac{x}{z} \\
y' = \frac{y}{z} \\
z' = -\left(\frac{\alpha + \beta}{z}\right)
\]

which projects orthogonally to the desired point regardless of \(\alpha\) and \(\beta\)
Picking $\alpha$ and $\beta$

If we pick

\[
\alpha = -\frac{f + n}{f - n} \quad \beta = -\frac{2nf}{f - n}
\]

the near plane is mapped to $z = -1$
the far plane is mapped to $z = 1$
and the sides are mapped to $x = \pm 1, y = \pm 1$

Hence the new clipping volume is the default clipping volume

\[
\mathbf{N} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -\frac{f + n}{f - n} & -\frac{2nf}{f - n} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]
General Perspective Frustum

\[ x' = x + \frac{l+r}{2n}z \]
\[ y' = y + \frac{t+b}{2n}z \]
\[ z' = z \]

\[ H = \begin{bmatrix}
1 & 0 & \frac{l+r}{2n} & 0 \\
0 & 1 & \frac{t+b}{2n} & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \]

Step 1: Shear to center on \(-z\) axis
General Perspective Frustum

\[
x' = \frac{2n}{r-l} x \\
y' = \frac{2n}{t-b} y \\
z' = z
\]

\[
S = \begin{bmatrix}
\frac{2n}{r-l} & 0 & 0 & 0 \\
0 & \frac{2n}{t-b} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Step 2: Scale so boundary slopes are \( \pm 1 \)
Normalization Transformation

original clipping volume

original object

COP

new clipping volume

distorted object projects correctly

University of Texas at Austin    CS354 - Computer Graphics    Don Fussell
The normalization in `glFrustum` requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume. Finally, the perspective matrix results in needing only a final orthogonal transformation.

\[ P = NSH \]

our previously defined perspective matrix
shear and scale
Normalization

- Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume.
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping.
Oblique Projections

- The OpenGL projection functions cannot produce general parallel projections such as

However if we look at the example of the cube it appears that the cube has been sheared

- Oblique Projection = Shear + Orthogonal Projection
General Shear

![Diagram of General Shear with top and side views showing clipping planes and object transformation.](image)
Shear Matrix

$xy$ shear ($z$ values unchanged)

$$H(\theta, \phi) = \begin{bmatrix} 1 & 0 & -\cot \theta & 0 \\ 0 & 1 & -\cot \phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection matrix

$$P = M_{\text{orth}} H(\theta, \phi)$$

General case:

$$P = M_{\text{orth}} \text{STH}(\theta, \phi)$$
Equivalency
The projection matrix $P = STH$ transforms the original clipping volume to the default clipping volume.
Using Field of View

- With `glFrustum` it is often difficult to get the desired view.
- `gluPerspective(fovy, aspect, near, far)` often provides a better interface.

![Diagram showing front plane and aspect ratio](image)

```
front plane
aspect = w/h
```
\textbf{OpenGL Perspective}

- \texttt{glFrustum} allows for an unsymmetric viewing frustum (although \texttt{gluPerspective} does not)

\[
\begin{align*}
\begin{pmatrix} x_{\text{min}} & y_{\text{min}} & z_{\text{max}} \end{pmatrix} & \quad \text{COP} \\
\begin{pmatrix} x_{\text{max}} & y_{\text{max}} & z_{\text{max}} \end{pmatrix} &
\end{align*}
\]
Frustum Transform

- **Prototype**
  - \( \text{glFrustum}(\text{GLfloat} \text{ left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far}) \)

- **Post-concatenates a frustum matrix**

\[
\begin{bmatrix}
2n & 0 & \frac{r+l}{r-l} & 0 \\
\frac{2n}{r-l} & \frac{2n}{t-b} & \frac{r-l}{t-b} & 0 \\
0 & 0 & -\frac{(f+n)}{f-n} & -2fn \\
0 & 0 & 1 & f-n \\
0 & 0 & 0 & 0
\end{bmatrix}
\]
glFrustum Matrix

- Projection specification
  - glLoadIdentity();
  - glFrustum(-4, +4, -3, +3, 5, 80)
  - left=-4, right=4, bottom=-3, top=3, near=5, far=80

- Matrix

\[
\begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{r-l}{t-b} & 0 \\
0 & 0 & \frac{-t+b}{t-b} & 0 \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
= 
\begin{bmatrix}
\frac{5}{4} & 0 & 0 & 0 \\
0 & \frac{5}{3} & 0 & 0 \\
0 & 0 & \frac{-85}{75} & \frac{800}{75} \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\]

symmetric left/right & top/bottom so zero

-Z axis
glFrustum Example

Consider

- \texttt{glLoadIdentity();}
- \texttt{glFrustum(-30, 30, -20, 20, 1, 1000)}
- left=-30, right=30, bottom=-20, top=20, near=1, far=1000

Matrix

\[
\begin{bmatrix}
\frac{2n}{r - l} & 0 & \frac{r + l}{r - l} & 0 \\
0 & \frac{2n}{t - b} & \frac{r - l}{t - b} & 0 \\
0 & 0 & -(f + n) & -2fn \\
0 & 0 & f - n & f - n
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
\frac{1}{30} & 0 & -1001 & -2000 \\
0 & \frac{1}{20} & -999 & -999 \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

Symmetric left/right & top/bottom so zero.
**glOrtho and glFrustum**

- These OpenGL commands provide a parameterized transform mapping eye space into the “clip cube”
- Each command
  - `glOrtho` is orthographic
  - `glFrustum` is single-point perspective
Handedness of Coordinate Systems

- **When**
  - Object coordinate system is right-handed,
  - Modelview transform is generated from one or more of the commands `glTranslate`, `glRotate`, and `glScale` with positive scaling values,
  - Projection transform is loaded with `glLoadIdentity` followed by exactly one of `glOrtho` or `glFrustum`,
  - Near value specified for `glDepthRange` is less than the far value;

- **Then**
  - Eye coordinate system is right-handed
  - Clip, NDC, and window coordinate systems are left-handed
Conventional OpenGL Handedness

- **Right-handed**
  - Object space
  - Eye space

- **Left-handed**
  - Clip space
  - Normalized Device Coordinate (NDC) space
  - Window space

In eye space, eye is “looking down” the negative Z axis

Positive depth is further from viewer
**Affine Frustum Clip Equations**

- The idea of a $[-1,+1]^3$ view frustum cube
  - Regions outside this cube get clipped
  - Regions inside the cube get rasterized

- Equations
  - $-1 \leq x_c \leq +1$
  - $-1 \leq y_c \leq +1$
  - $-1 \leq z_c \leq +1$
Projective Frustum Clip Equations

- Generalizes clip cube as a projective space
  - Uses \((x_c, y_c, z_c, w_c)\) clip-space coordinates

- Equations
  - \(-w_c \leq x_c \leq +w_c\)
  - \(-w_c \leq y_c \leq +w_c\)
  - \(-w_c \leq z_c \leq +w_c\)

- Notice
  - Impossible for \(w_c < 0\) to survive clipping
  - Interpretation: \(w_c\) is distance in front of the eye
    - So negative \(w_c\) values are "behind your head"
NDC Space Clip Cube

Post-perspective divide puts the region surviving clipping within the $[-1,+1]^3$
Clip Space Clip Cube

Constraints

\begin{align*}
x_{\text{min}} &= -w \\
x_{\text{max}} &= w \\
y_{\text{min}} &= -w \\
y_{\text{max}} &= w \\
z_{\text{min}} &= -w \\
z_{\text{max}} &= w \\
w &> 0
\end{align*}

Pre-perspective divide puts the region surviving clipping within

\(-w \leq x \leq w, \ -w \leq y \leq w, \ -w \leq z \leq w\)
Window Space Clip Cube

Assuming `glViewport(x, y, w, h)` and `glDepthRange(zNear, zFar)`

Constraints
- \( w > 0 \)
- \( h > 0 \)
- \( 0 \leq z_{\text{Near}} \leq 1 \)
- \( 0 \leq z_{\text{Far}} \leq 1 \)

University of Texas at Austin    CS354 - Computer Graphics    Don Fussell
Perspective Divide

- Divide clip-space \((x,y,z)\) by clip-space \(w\)
- To get Normalized Device Coordinate (NDC) space
- Means reciprocal operation is done once
  - And done after clipping
  - Minimizes division by zero concern

\[
\begin{bmatrix}
  x_n \\
  y_n \\
  z_n
\end{bmatrix} = \begin{bmatrix}
  x_c / w_c \\
  y_c / w_c \\
  z_c / w_c
\end{bmatrix}
\]
Transform All Box Corners

- Consider
  - `glLoadIdentity();`
  - `glOrtho(-20, 30, 10, 60, 15, -25);`
    - l=-20, r=30, b=10, t=50, n=15, f=-25
  - Eight box corners: (-20,10,-15), (-20,10,25), (-20, 50,-15), (-20, 50,-25), (30,10,-15),  (30,10,25),  (30,50,-15),     (30,50,25)
- Transform each corner by the 4x4 matrix
  \[
  \begin{bmatrix}
  1 & 0 & 0 & 1 \\
  25 & 0 & 0 & -\frac{1}{5} \\
  0 & 1 & 0 & -\frac{3}{2} \\
  20 & 0 & 1 & -\frac{1}{4} \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
  \[
  \begin{bmatrix}
  -20 & -20 & -20 & -20 & 30 & 30 & 30 & 30 \\
  10 & 10 & 50 & 50 & 10 & 10 & 50 & 50 \\
  1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
  \end{bmatrix}
  \]

Keep in mind: looking down the negative Z axis... so Z box coordinates are negative n (-15) and negative f (+25)

8 corners in column vector (position) form
### Box Corners in Clip Space

8 "eye space" corners in column vector form:

\[
\begin{bmatrix}
\frac{1}{25} & 0 & 0 & -\frac{1}{5} \\
0 & \frac{1}{20} & 0 & -\frac{3}{2} \\
0 & 0 & \frac{1}{20} & -\frac{1}{4} \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
-20 & -20 & -20 & -20 & 30 & 30 & 30 & 30 \\
10 & 10 & 50 & 50 & 10 & 10 & 50 & 50 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
-1 & -1 & -1 & -1 & +1 & +1 & +1 & +1 \\
-1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 \\
-1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

Result is "corners" of clip space (and NDC) clip cube.
Transform All Box Corners

*keep in mind:* looking down
the negative Z axis... so Z box coordinates are
negative n (-1) and
negative f (-1000)

- Consider
  - `glLoadIdentity();`
  - `glFrustum(-30, 30, -20, 20, 1, 1000)`
    - left=-30, right=30, bottom=-20, top=20, near=1, far=1000
- Eight box corners: (-30,-20,-1), (-30,-20,-1000), (-30, 20,-1), (-30, 20,-1000),
  (30,10,-1), (30,10,-1000), (30,50,-1), (30,50,-1000)

- Transform each corner by the 4x4 matrix

\[
\begin{bmatrix}
\frac{1}{30} & 0 & 0 & 0 \\
0 & \frac{1}{20} & 0 & 0 \\
0 & 0 & -1001 & -\frac{2000}{999} \\
0 & 0 & -\frac{999}{999} & 0 \\
\end{bmatrix}
\begin{bmatrix}
-30 & -30000 & -30 & -30000 & 30 & 30000 & 30 & 30000 \\
-20 & -20000 & 20 & 20000 & -20 & -20000 & 20 & 20000 \\
-1 & -1000 & -1 & -1000 & -1 & -1000 & -1 & -1000 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\]

near far near far near far near far near far
Box Corners in Clip Space

8 “eye space” corners in column vector form

\[
\begin{bmatrix}
\frac{1}{30} & 0 & 0 & 0 \\
0 & \frac{1}{20} & 0 & 0 \\
0 & 0 & -\frac{1001}{999} & -\frac{2000}{999} \\
0 & 0 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
\frac{-30}{30} & -\frac{30000}{30} & -\frac{30000}{30} & \frac{30}{30} & \frac{30000}{30} & \frac{30}{30} & \frac{30000}{30} \\
\frac{-20}{20} & -\frac{20000}{20} & \frac{20000}{20} & -\frac{20}{20} & -\frac{20000}{20} & \frac{20}{20} & -\frac{20000}{20} \\
\frac{-1}{1} & -\frac{1000}{1} & -\frac{1000}{1} & -\frac{1}{1} & -\frac{1000}{1} & -\frac{1}{1} & -\frac{1000}{1} \\
\frac{1}{1} & \frac{1}{1} & \frac{1}{1} & \frac{1}{1} & \frac{1}{1} & \frac{1}{1} & \frac{1}{1}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
-1 & -1000 & -1 & -1000 & +1 & +1000 & +1 & +1000 \\
-1 & -1000 & +1 & +1000 & -1 & -1000 & +1 & +1000 \\
-1 & +1000 & -1 & +1000 & -1 & +1000 & -1 & +1000 \\
+1 & +1000 & +1 & +1000 & +1 & +1000 & +1 & +1000
\end{bmatrix}
\]
Box Corners in NDC Space

- Perform perspective divide

\[
\begin{pmatrix}
-1 & -1000 & -1 & -1000 & +1 & +1000 & +1 & +1000 \\
-1 & -1000 & +1 & +1000 & -1 & -1000 & +1 & +1000 \\
-1 & +1000 & -1 & +1000 & -1 & +1000 & -1 & +1000 \\
+1 & +1000 & +1 & +1000 & +1 & +1000 & +1 & +1000 \\
\end{pmatrix}
\]

\[
= \begin{pmatrix}
-1 & -1 & -1 & -1 & +1 & +1 & +1 & +1 \\
-1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 \\
-1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 \\
+1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\
\end{pmatrix}
\]

W component is 1 (at near plane) or 1/1000 (at far plane)
Z component is always -1 (assuming W=1 eye-space positions)
Eye Space and NDC Space

- "behind the eye"
- "between eye and near clip plane"
- "rendered (visible) region"
- "beyond the far clip plane"

[Eric Lengyel]
Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if \( z_1 > z_2 \) in the original clipping volume then for the transformed points \( z_1' > z_2' \).

Thus hidden surface removal works if we first apply the normalization transformation.

However, the formula \( z' = -(\alpha + \beta/z) \) implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small.
Why do we do it this way?

- Normalization allows for a single pipeline for both perspective and orthogonal viewing.
- We stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading.
- We simplify clipping.
We stay in four-dimensional homogeneous coordinates through both the modelview and projection transformations.

- Both these transformations are nonsingular.
- Default to identity matrices (orthogonal view).

Normalization lets us clip against simple cube regardless of type of projection.

Delay final projection until end.

- Important for hidden-surface removal to retain depth information as long as possible.
Viewport and Depth Range

- Prototypes
  - `glViewport(GLint vx, GLint vy, GLsizei vw, GLsizei vh)`
  - `glDepthRange(GLclampd n, GLclampd f)`

- Equations
  - Maps NDC space to window space

\[
\begin{bmatrix}
\frac{v_w}{2} x_n + \left( \frac{v_x + \frac{v_w}{2}}{2} \right) \\
\frac{v_h}{2} y_n + \left( \frac{v_y + \frac{v_h}{2}}{2} \right) \\
\frac{f - n}{2} z_n + \frac{f + n}{2}
\end{bmatrix}
\]
Next Lecture

- Modelview Transformations