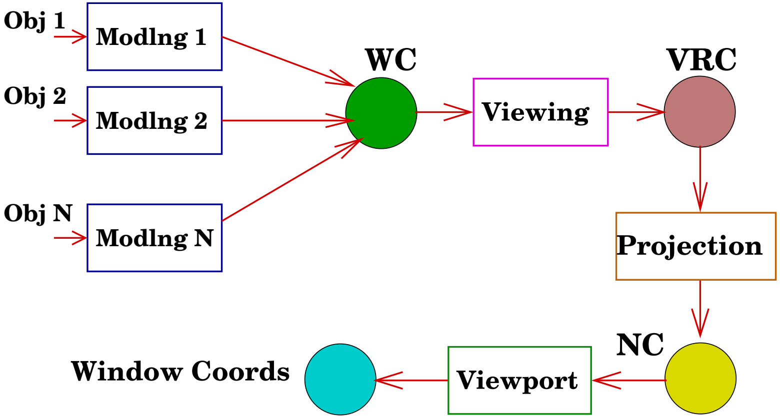
CSE251
Basics of Computer Graphics
Module: Graphics Pipeline

Avinash Sharma

Spring 2018

3D Graphics: Block Diagram



Perspective Normalizing Matrix

- General case: Match the vertices and solve!
  $(left, bottom, -near) \to (-1, -1, 1)$,
  $(l, t, -n) \to (-1, 1, 1)$, $(r, t, -n) \to (1, 1, 1)$,
  $(rf/n, bf/n, -f) \to (1, -1, -1)$,
  $(lf/n, tf/n, -f) \to (-1, 1, -1)$

Perspective Normalizing Matrix

- General case: Match the vertices and solve!
  $(left, bottom, -near) \rightarrow (-1, -1, 1)$,
  $(l, t, -n) \rightarrow (-1, 1, 1)$, $(r, t, -n) \rightarrow (1, 1, 1)$,
  $(rf/n, bf/n, -f) \rightarrow (1, -1, -1)$,
  $(lf/n, tf/n, -f) \rightarrow (-1, 1, -1)$

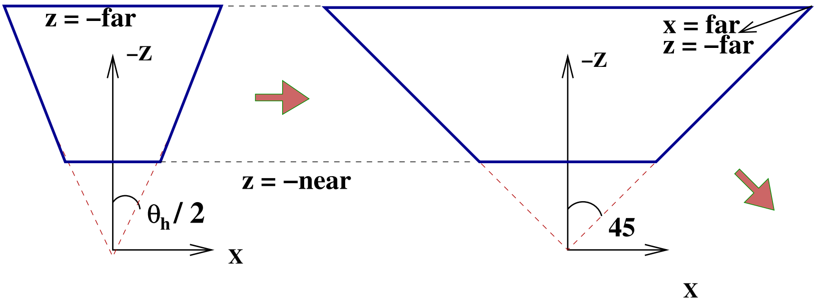
- Each match gives 3 equations in the 16 unknowns m_{ij}.
  (Only 15 unknowns upto scale!)

- Can solve for them given 5 point matchings.
  We have 8, hence easy!

Symmetric Perspective Proj Matrix

- More complicated than orthographic case, as a frustum has to be mapped to a cube. Do it in steps.
- First, scale the horizontal and vertical extents so that the vertical and horizontal fields of view are 90 degres.
- A scaling transformation with $s_x = ??, s_y = ??, s_z = 1$
- View volume is almost right except for a uniform scale.
- Next, scale uniformly so that the far plane is at -1. We will also have $-1 \leq x, y \leq 1$ at the far plane after this.
- $s_x = s_y = s_z = ??$

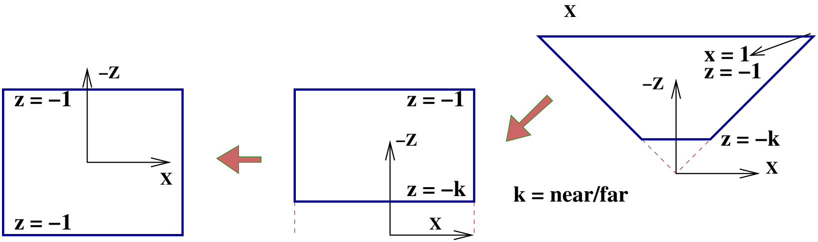
Symmetric Perspective Proj Matrix (cont.)



Symmetric Perspective Proj Matrix (cont.)

- M_1 with $s_x = \cot\frac{\theta_h}{2}, \quad s_y = \cot\frac{\theta_v}{2}, \quad s_z = 1$
- M_2 with $s_x = s_y = s_z = \frac{1}{\mathbf{far}}$
- $$M_2 M_1 = \begin{bmatrix} \frac{\cot\theta_h/2}{\mathbf{far}} & 0 & 0 & 0 \\ 0 & \frac{\cot\theta_v/2}{\mathbf{far}} & 0 & 0 \\ 0 & 0 & \frac{1}{\mathbf{far}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- The near plane is now at $\mathbf{k} = \frac{\mathbf{near}}{\mathbf{far}}$.
- View volume fits into the canonical view volume, but is still a frustum!

Symmetric Perspective Proj Matrix (cont.)



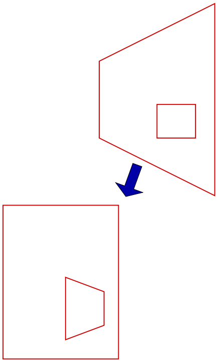
- Scale by to $-z$ to convert to a cube using a matrix with last row $[0 \quad 0 \quad -1 \quad 0]$.
- Simultaneously send third component to range $[-1, 1]$. $z = -k$ maps to 1 and $z = -1$ maps to -1.
- Scale z by $\frac{1+k}{1-k}$ and translate by $\frac{2k}{1-k}$.

Perspective Normalizing Txform

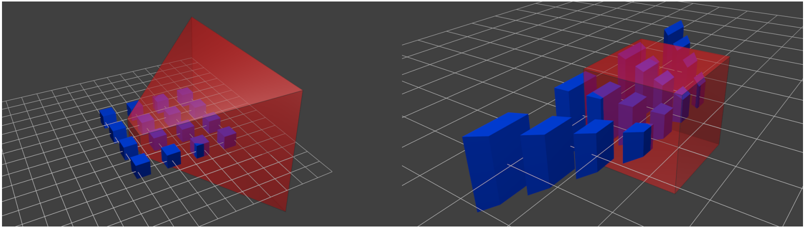
- Matrix M_3 for this step:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+k}{1-k} & \frac{2k}{1-k} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- $(x, y, -k)$ & $(x, y, -1)$ go to?
- Final matrix: $M = M_3 M_2 M_1$
- Frustum becomes a cube



Canonical View Volume: Visualization



http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

Final 2D Coordinates

$$(u, v, d) \equiv \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = M_3 M_2 M_1 \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\text{VRC}}$$

- **Perspective division:** Divide x', y' coordinates by the w to get the normalized coordinates (u, v). (z' maintains ordering and can be used without division.)
- The normalized d component has non-linear precision. Higher around the *near* plane and lower around the *far* plane due to the division by z.

OpenGL Normalizing/Perspective Matrix

- The projection matrix in OpenGL is given by

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & A & 0 \\ 0 & \frac{2n}{t-b} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where,

$$A = \frac{r+l}{r-l}, B = \frac{t+b}{t-b}, C = \frac{f+n}{f-n}, D = \frac{2nf}{f-n}$$

*Please refer to the support material for derivation.

Actual Projection

- We have already performed the perspective division.
- Projection involves simply dropping the z coordinate and scaling x-y to the viewport.
- Why go through with the z-coordinates?
- The ordering is preserved along the depth dimension.
  z values can be used for visibility determination.

Where is the Film?

- Turns out: **It does not matter**.
- A final scaling is in the viewport transformations.
- As long as the film is in front of the camera, we will see an upright image.
- Can consider the near plane as the film plane.

Viewport Txformation: To Window

- Image of size -1 to +1 in X and Y is ready. The **viewport** transformation maps it to the actual window on screen.
- From $[1, 1]$, map x and y to $[0, W]$ and $[0, H]$.
- First step: set sizes by scaling: $\mathbf{S}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})$.
  Next: Translate origin to South-West corner: $\mathbf{T}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})$
- Overall: $\mathbf{M} = \mathbf{T}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})\ \mathbf{S}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}}) = \begin{bmatrix} \frac{W}{2} & 0 & \frac{W}{2} \\ 0 & \frac{H}{2} & \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix}$

General Viewport Txform

- General command: `glViewport(l, b, r, t)`.
- Translate so the range of x, y is $0 \cdots 2$.
- Scale so x varies from 0 to $(r - l)$ and y varies from 0 to $(t - b)$.
- Translate so x range is l to r and y range is b to t.
- Matrix for this? $\mathbf{T}(l, b)\ \mathbf{S}(\frac{r-l}{2}, \frac{t-b}{2})\ \mathbf{T}(1, 1) = \begin{bmatrix} \frac{r-l}{2} & 0 & \frac{r+l}{2} \\ 0 & \frac{t-b}{2} & \frac{t+b}{2} \\ 0 & 0 & 1 \end{bmatrix}$
- $[-1 \quad -1 \quad 1]^{\mathsf{T}}$ maps to $[l \quad b \quad 1]^{\mathsf{T}}$.
- $[1 \quad 1 \quad 1]^{\mathsf{T}}$ maps to $[r \quad t \quad 1]^{\mathsf{T}}$.

(Point) Pipeline in Action

- Points are transformed from Object to World to Canonical to Window coordinates.
- Each 3D point maps to a pixel (i, j) in the window space.
- Lines are made out of two points. Triangles and polygons are made out of 3 or more points.

Recapitulation

- 3D Graphics additionally involves projecting the 3D world to the 2D image plane of the camera.
- Compute the 3D world with respect to the camera. Or compute the relative geometry first.
- This involves a series of rigid transfomrations. For complex objects/environments, each object or its part is described in its own coordinate system.
- **Modelling** places these different objects in the world coordinate system. This could involve a hierarchy of transforms for objects made up of complex parts.
- **View Orientation** computes the world in the VRC.

Recapitulation (cont.)

- Camera can be perspective or parallel (orthographic, oblique). 6 planes give the view volume and defines the camera.
- **View Mapping** involves mapping the world to a canonical view volume, which is an orthographic view volume. This **Normalizing Transformation** has different forms for parallel and perspective cameras.
- **Projection** and **Clipping** are easy to perform in the canonical view volume. An image with dimensions from -1 to +1 results.
- **Viewport** tranformation is the final step, involving a 2D scaling and translation to map to window coordinates that can be used to address the frame buffer.

Recapitulation (cont.)

- Given a description of the 3D world primitives, project each point to 2D to get 2D primitives. These can be scan-converted using standard algorithms.