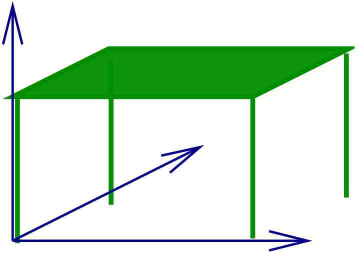
CS7.302
Basics of Computer Graphics
Module: Graphics Pipeline

Avinash Sharma

Spring 2021

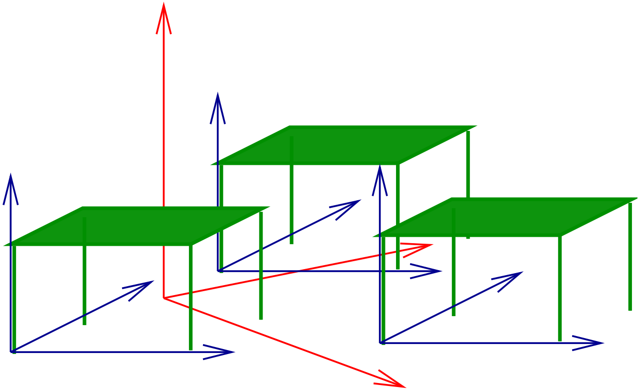
A Single Table

A table defined in its own coordinate system.

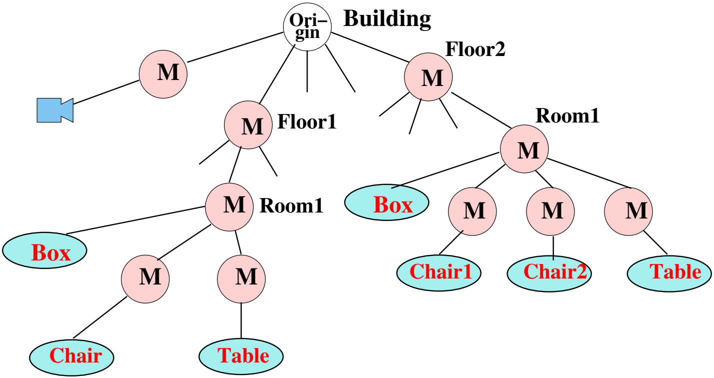


Many Tables in a Room

Place many tables in a **common world coord system**!

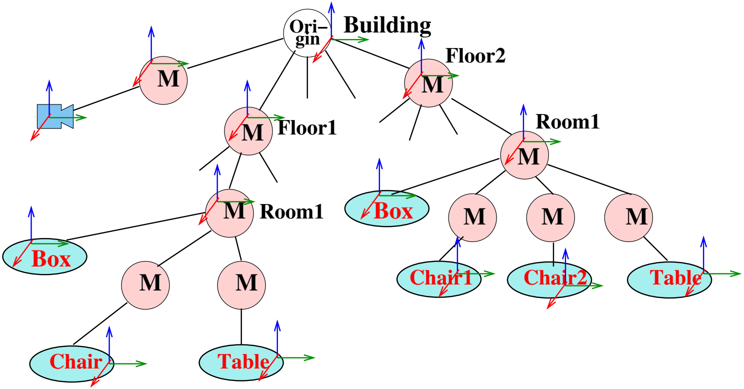


A Building Model



Hierarchical model with root representing whole scene.

A Building Model



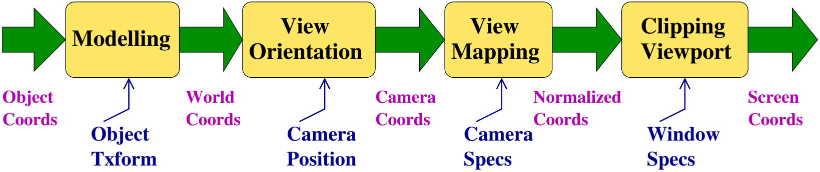
Each matrix **M** aligns parent frame to child frame

Different Coordinates

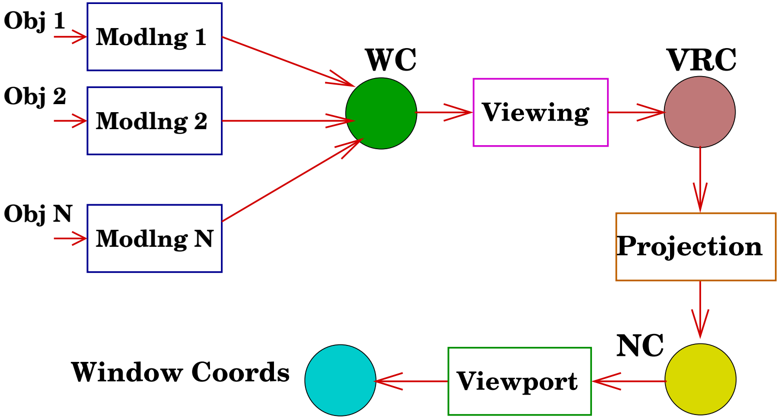
- **Object Reference:** Object is described in an internal coordinate frame called **ORC**.
- **World:** Common reference frame to describe different objects, called **WC**.
- **Camera/View Reference:** Describe with respect to the current camera position/orientation, called **VRC**
- Ultimately, **how the scene appears to the camera** determines the image produced
- Goal of Computer Graphics: describe the scene in the camera coordinate frame

3D Graphics Pipeline

- Objects are specified in their own coordinate system and placed in the world coordinate frame.
- Camera is also placed in the world coordinate frame.
- Camera-to-world geometry is first projected to normalized coordinates and then to screen.



3D Graphics: Block Diagram



Different Coordinates

- **Object Reference:** Object is described in an internal coordinate frame called **ORC**.
- **World:** Common reference frame to describe different objects, called **WC**.
- **Camera/View Reference:** Describe with respect to the current camera position/orientation, called **VRC**.
- **Normalized Projection:** A standard space from which projection is easy, called **NPC**.
- **Screen:** Coordinates in the output device space.

Transformations

- **Modelling:** Convert from object coordinates to world coordinates (ORC to WC).
- **View Orientation or Viewing:** From world coordinates to camera coordinates (WC to VRC).
- Simple coordinate transformations.
- **View Mapping or Projection:** From VRC to Normalized Coordinates (NC).
- **Viewport:** From NC to window coordinates.

Modelling and Viewing

- Transform points from object coordinates (ORC) to world coordinates (WC) to camera coordinates (VRC)
- A series of transformations for each object or point

$$\mathbf{P}_{\mathbf{VRC}} = \Big|_{\mathbf{VRC}} \mathbf{V} \Big|_{\mathbf{WC}} \mathbf{M} \Big|_{\mathbf{ORC}} \mathbf{P}_{\mathbf{ORC}}$$

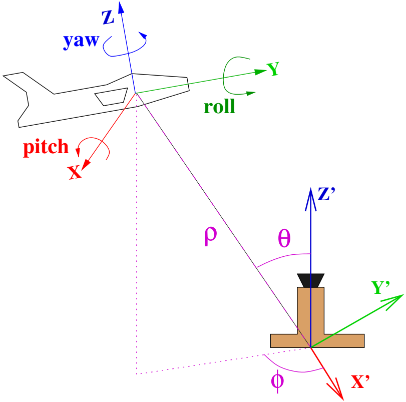
- Goal: Evaluate the coordinates of each point/line/triangle with respect to the camera

Modelling

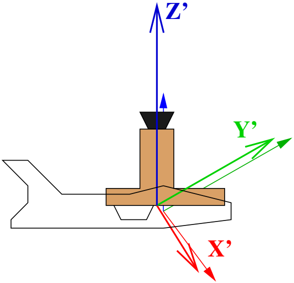
- Goal: Transform object coordinates to world coordinates.
- Method: Place ORC frame in the world coordinate frame.
- A single transformation matrix or **modelling matrix** with translation, rotation, scaling.
- A unit cube at origin can generate any cuboid using translation/rotation/scaling.
- Different objects have different modelling matrices.

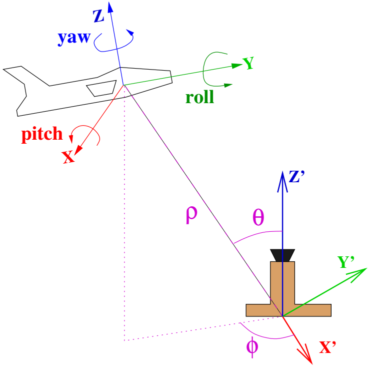
Example: Aircraft in a Polar World

- WC frame on ground, ORC frame on the aircraft.
- Controllers think in polar coordinates for position and roll-pitch-yaw for orientation.
- What are the modelling steps?



Example: Aircraft in a Polar World (cont.)





Start

at origin and move to new location

Aircraft in a Polar World

- Start with both axes aligned
- Translate to the location given by (ρ, θ, ϕ)
- Apply yaw, pitch, and roll: In which order ??

Aircraft in a Polar World

- Start with both axes aligned
- Translate to the location given by (ρ, θ, ϕ)
- Apply yaw, pitch, and roll in that order. (Why?)
- Coordinate axes undergoing transformation!
- Net effect: $\mathbf{T}(\rho, \theta, \phi)\ \mathbf{R_z}(\mathbf{y})\ \mathbf{R_x}(\mathbf{p})\ \mathbf{R_y}(\mathbf{r})$ yaw, pitch,roll
- What is $\mathbf{T}(\rho, \theta, \phi)$? Compute (x, y, z) and translate
- Alternate: Rotate to align aircraft's Z-axis to translation direction, translate by ρ and unrotate
  $\mathbf{T}(\rho, \theta, \phi) = \mathbf{R_z}(-\phi)\mathbf{R_y}(\theta)\mathbf{T}(\mathbf{0}, \mathbf{0}, \rho)\mathbf{R_y}(-\theta)\mathbf{R_z}(\phi)$

Why yaw, pitch, roll?

- Let **Y be East**, **X be South**, and **Z be Up**
- Consider a **pitch of 30 degrees** and a **yaw of 90 degrees**
- **Yaw** followed by **pitch**: what happens?
- **Pitch** followed by **yaw**: what happens?

Why yaw, pitch, roll?

- Let **Y be East**, **X be South**, and **Z be Up**
- Consider a **pitch of 30 degrees** and a **yaw of 90 degrees**
- **Yaw** followed by **pitch**: Flight going North, climbing 30^o
  - Flight goes from Hyderabad to Delhi, still climbing
- **Pitch** followed by **yaw**: what happens?

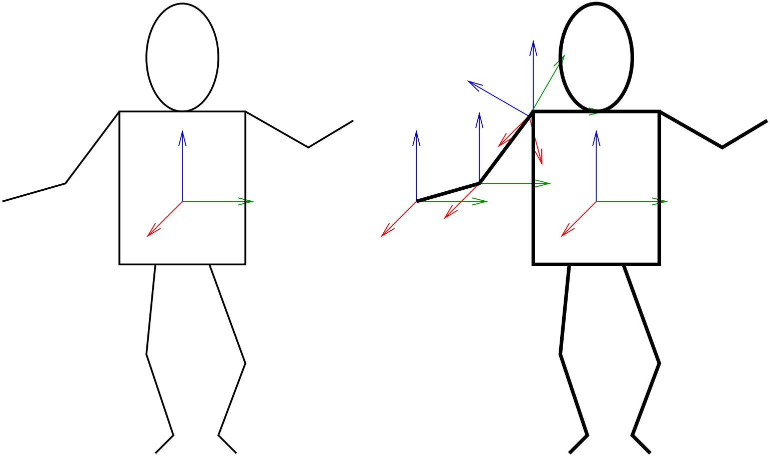
Why yaw, pitch, roll?

- Let **Y be East**, **X be South**, and **Z be Up**
- Consider a **pitch of 30 degrees** and a **yaw of 90 degrees**
- **Yaw** followed by **pitch**: Flight going North, climbing 30^o
  - Flight goes from Hyderabad to Delhi, still climbing.
  - (In reality, aircraft will also roll while turning left).
- **Pitch** followed by **yaw**.
  - Yaw happens in a different plane
  - Flight wont be climbing, but will have a roll!
  - Not what one wants!

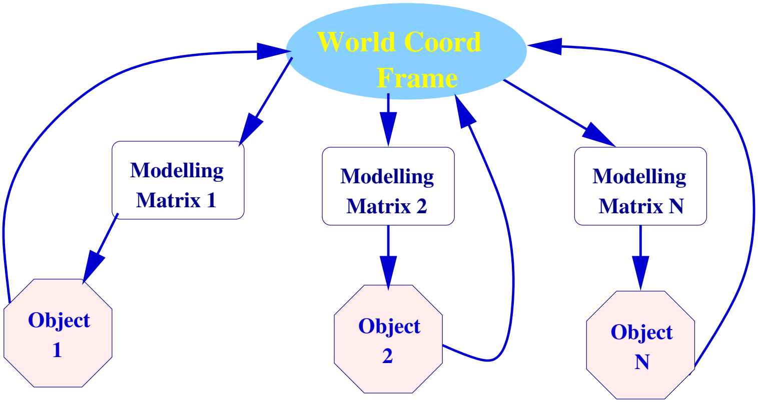
Hierarchy of Transformations

- A hierarchy of transformations needed to setup the world and the camera.
- A humanoid robot could have a coordinate frame on its body, another one on the shoulder, a third on the shoulder that moves with the upper arm, a fourth on the elbow, a fifth on the elbow that moves with the forearm, etc.
- Remember the wheel with an ant moving on its spoke!
- $\mathbf{M} = \mathbf{T}_1 \ \mathbf{T}_2 \ \mathbf{T}_3 \ \cdots$ captures the composite transform as a shift in coordinate frames.

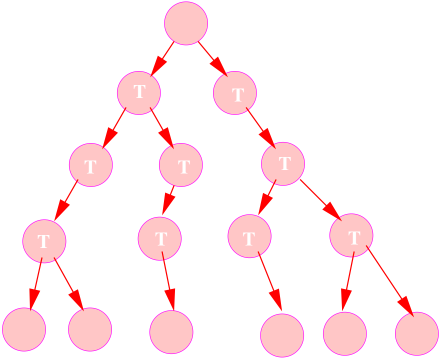
Humanoid Robot



Modelling Different Objects



Scene Graph



- Objects organized hierarchically with transforms.

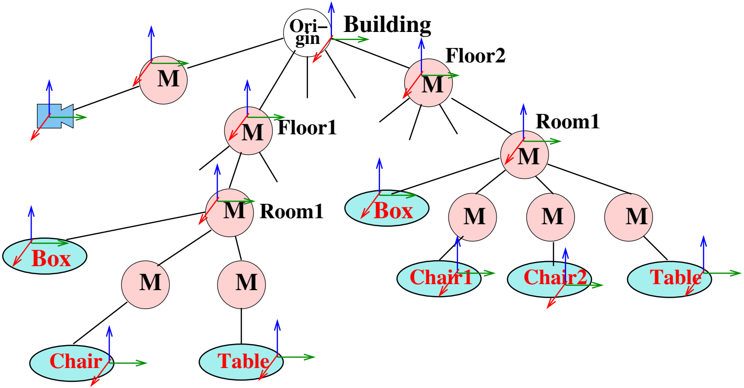
Modelling in OpenGL

- OpenGL 3.0 takes a single matrix that transforms object coordinates to normalized projection coordinates **directly**.
- You can devise separate **Projection**, **Viewing**, and **Modelling** matrices for ease of understanding
- Multiply them into $\mathbf{P\ V\ M}$ and send to the shader
- Shader transforms coordinates in the vertex array to projection/screen coordinates using this matrix
- Modelling matrix for the aircraft in polar coordinates:
  $\mathbf{M} = \mathbf{T}\ \mathbf{R_Z(y)}\ \mathbf{R_X(p)}\ \mathbf{R_Y(r)}$

View Orientation or Viewing

- Placing the camera in the world and orienting it right.
- Has 6 degrees of freedom: 3 for position and 3 for orientation.

Building: Scene Graph



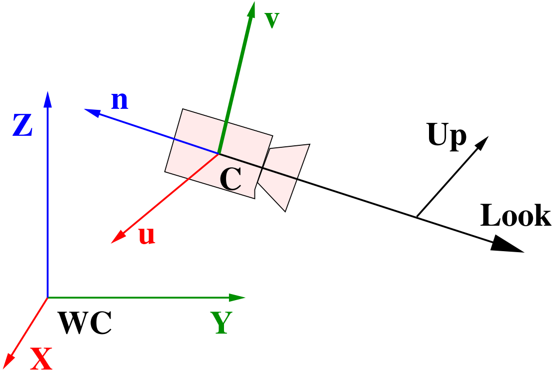
Each matrix **M** aligns parent frame to child frame

View Orientation or Viewing

- Placing the camera in the world and orienting it right.
- Has 6 degrees of freedom: 3 for position and 3 for orientation.
- Goal: Transform points expressed in WC to VRC.
- Let $\mathbf{u}, \mathbf{v}, \mathbf{n}$ be the VRC or camera coordinate axes
- Viewing Transformation can be specified in many ways.
- Commonly using: Camera location, Look point, and Up direction.

Viewing Specification

- Camera-center, Look-point and Up-vector specified in the world coordinates.



Transformation Steps

How do we align WC to VRC?

- Translate to $\mathbf{C} = (x, y, z)$.
- Rotate to align Z-axis to $-$(Look Vector) or $-\bar{\mathbf{L}}$
- Rotate to align Y-axis to Up.
- Translation is easy. How do we get the rotation matrix?
- Remember columns of the matrix give directions **to** which the axes rotate!!

Rotation

- Let $\bar{\mathbf{l}} = \bar{\mathbf{L}}/|\bar{\mathbf{L}}|$ and $\bar{\mathbf{t}} = \bar{\mathbf{U}}/|\bar{\mathbf{U}}|$ be the unit vectors in those directions.
- Third column of the matrix: $\bar{\mathbf{n}} = -\bar{\mathbf{l}}$.
- Up vector needn't be orthogonal to the look vector. The $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ vectors define the "vertical" plane. A **plane in the world** that projects to **a vertical line in the image**. Or the camera's $\mathbf{vn}$ plane.
- First column: $\bar{\mathbf{u}} = \bar{\mathbf{t}} \times \bar{\mathbf{n}}/|\bar{\mathbf{t}} \times \bar{\mathbf{n}}|$
- Second column: $\bar{\mathbf{v}} = \bar{\mathbf{n}} \times \bar{\mathbf{u}}$.

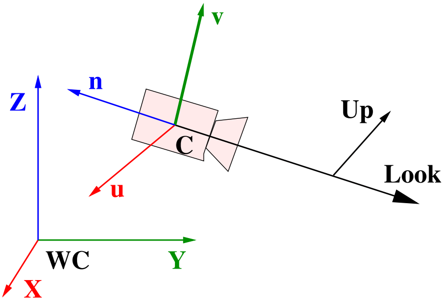
View Orientation Transformation

$$\mathbf{A} = \begin{bmatrix} & & & x \\ & \mathbf{I} & & y \\ & & & z \\ & \mathbf{0}^\mathsf{T} & & 1 \end{bmatrix} \begin{bmatrix} & & & 0 \\ \bar{\mathbf{u}} & \bar{\mathbf{v}} & \bar{\mathbf{n}} & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & & & x \\ \bar{\mathbf{u}} & \bar{\mathbf{v}} & \bar{\mathbf{n}} & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- What have we achieved?

  $\mathbf{P_{WC}} = \mathbf{A}\ \mathbf{P_{VRC}}$ or

  $\mathbf{P_{VRC}} = \mathbf{A}\ \mathbf{P_{WC}}$?



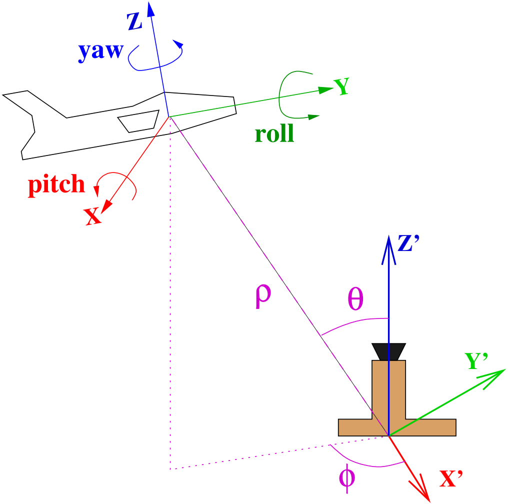
View Orientation Transformation

$$\mathbf{A} = \left[\begin{array}{cc} & x \\ \mathbf{I} & y \\ & z \\ \mathbf{0}^{\mathsf{T}} & 1 \end{array}\right] \left[\begin{array}{cccc} & & & 0 \\ \bar{\mathbf{u}} & \bar{\mathbf{v}} & \bar{\mathbf{n}} & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{cccc} & & & x \\ \bar{\mathbf{u}} & \bar{\mathbf{v}} & \bar{\mathbf{n}} & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{array}\right]$$

- We have achieved: $\mathbf{P}_{\mathbf{WC}} = \mathbf{A}\ \mathbf{P}_{\mathbf{VRC}}$.
- We need the reverse, everything to be in VRC
- Viewing transform: $\mathbf{V} = \mathbf{A}^{-1} = \mathbf{R}^{\mathsf{T}}\ \mathbf{T}(-\mathbf{C})$.

Viewing from the Aircraft

- Need to give the pilot's view from aircraft.
- What are the viewing steps?



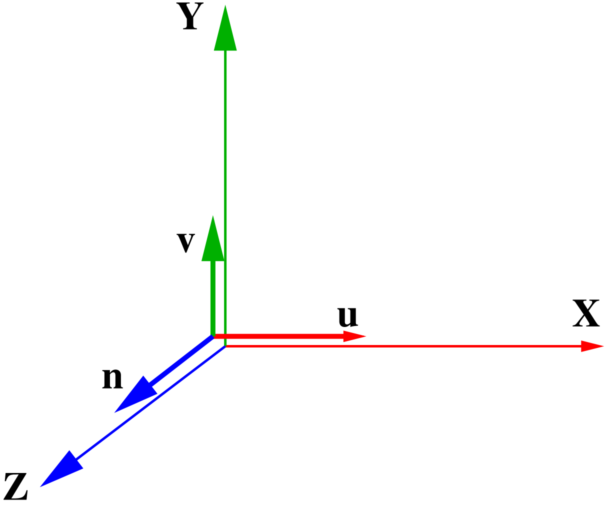
Aircraft in Polar World: Viewing

- Start with both axes aligned.
- Inverse of modelling or placing aircraft in WC
- Viewing transform: $\mathbf{R_y}(-\mathbf{r})\ \mathbf{R_x}(-\mathbf{p})\ \mathbf{R_z}(-\mathbf{y})\ \mathbf{T}^{-1}(\rho, \theta, \phi)$

Modelling and Viewing in OpenGL

- Modelling and Viewing are not truly independent.
- What ultimately matters is only the **relative geometry** between the camera and the object(s).
- What we want is the description of each point in VRC, with respect to the camera.
- It is convenient to think of each object being placed in a WC and then the WC being transformed to VRC.
- Thus, **each object has its** modelling matrix.
  The scene **has one** viewing matrix

When OpenGL Starts

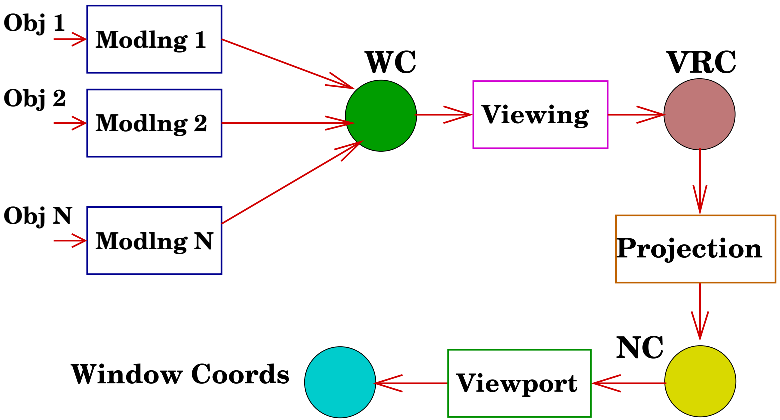


Modelling and Viewing matrices are Identity.

Setting up Objects and Camera

- WC is at VRC at start. First push it away to where WC should be. This is the Viewing Transformation matrix $\mathbf{V}$
- **Stay here** and draw objects in the scene one by one
  - Move to ORC of each object and draw its own model
  - Each object i has its Modelling Matrix $\mathbf{M_i}$
- Create matrix $\mathbf{P}\ \mathbf{V}\ \mathbf{M_i}$ and send to shader
  - Draw the object using description in its own frame

Block Diagram



Structure of an OpenGL Program

// Set projection matrix $\mathbf{P}$ (covered later)

// WC is aligned to VRC on start

// Camera is given by Pos & Orientation in WC

$\mathbf{V} = \mathbf{R}(-\mathbf{Orient})\ \mathbf{T}(-\mathbf{Pos})$ // WC moved away from VRC

// WC is set. Model each object with it as reference

// Draw object i with respect to WC

$\mathbf{M} = \mathbf{T}(\mathbf{i})\mathbf{R}(\mathbf{i})$ // Modelling matrix for object i

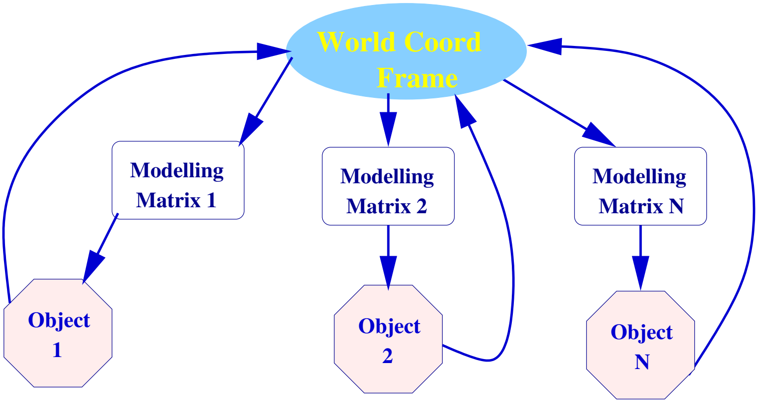
$\mathbf{Mat} = \mathbf{P}\ \mathbf{V}\ \mathbf{M}$ // from MVP matrix

send Mat to Shader // send to shader

`drawObject(i)` // Draw object polygon

// Start next object with respect to WC

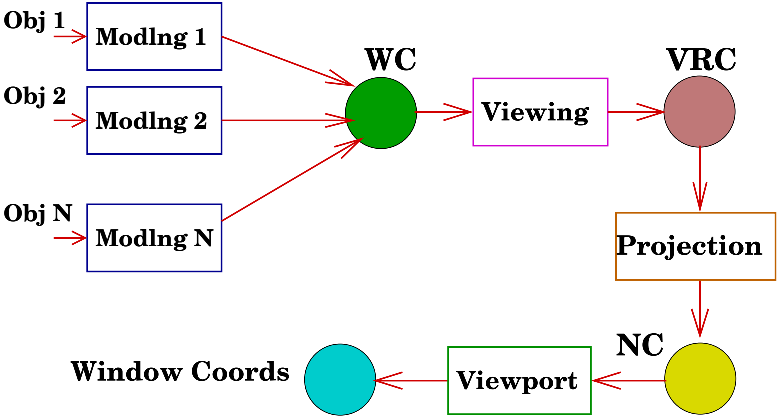
Modelling Different Objects



Modelling & Viewing: Summary

- Place objects in the world coordinate frame
- Place camera in the world coordinate frame
- Can compute object points in camera coordinate frame
- $\mathbf{P}_{\mathbf{VRC}} = \mathbf{V} \cdot \mathbf{M} \cdot \mathbf{P}_{\mathbf{ORC}}$

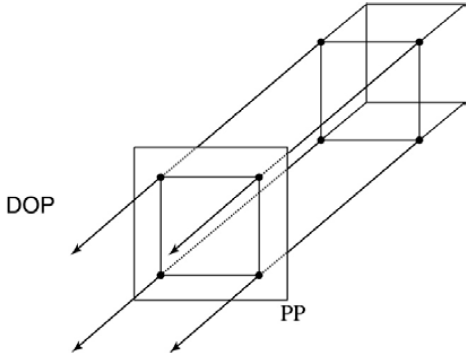
3D Graphics: Block Diagram



Projections

- Projection involves *projectors* starting from 3D points and hitting the 2D *projection plane*, forming the *image* of the point.
- Two types of projections.
- **Parallel projection**: Projectors are parallel to each other, all have the same *direction of projection* (**DOP**).

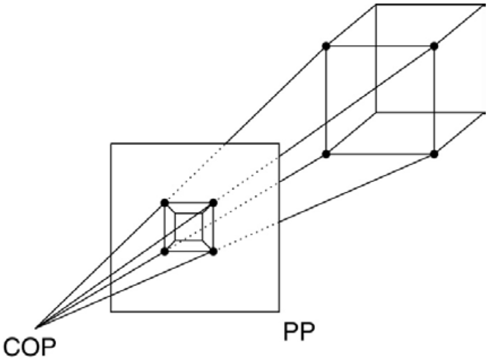
Projections (cont.)



Projections

- Projection involves *projectors* starting from 3D points and hitting the 2D *projection plane*, forming the *image* of the point.
- Two types of projections.
- **Parallel projection**: Projectors are parallel to each other, all have the same *direction of projection* (**DOP**).
- **Perspective projection**: All projects pass through a point in space called the *centre of projection* (**COP**).

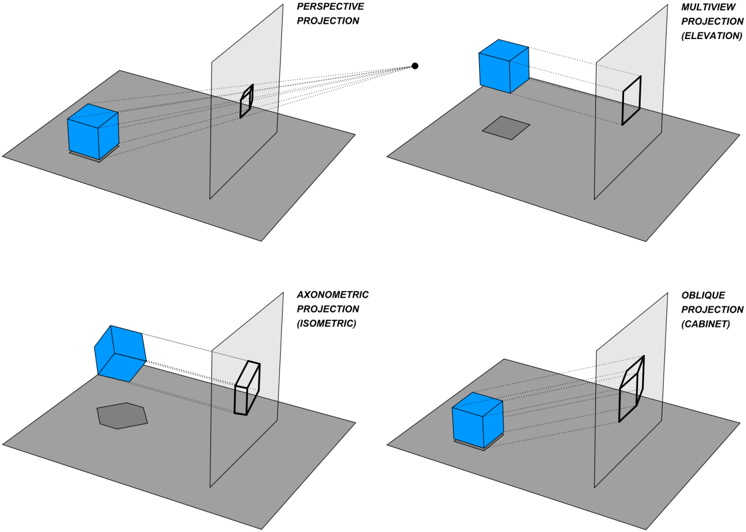
Projections (cont.)



Parallel Projections

- **Orthographic:** Projection plane is perpendicular to the direction of projection.
  - If direction of projection parallel to the axes: *plan, elevation, side elevation*.
  - If PP intersect all axels at equal distance : *isometric* projection.
- **Oblique:** Otherwise.
  - *Cavalier* when projectors make 45 degrees with the projection plane.
  - *Cabinet* when they make `arctan(2)` degrees with the projection plane.

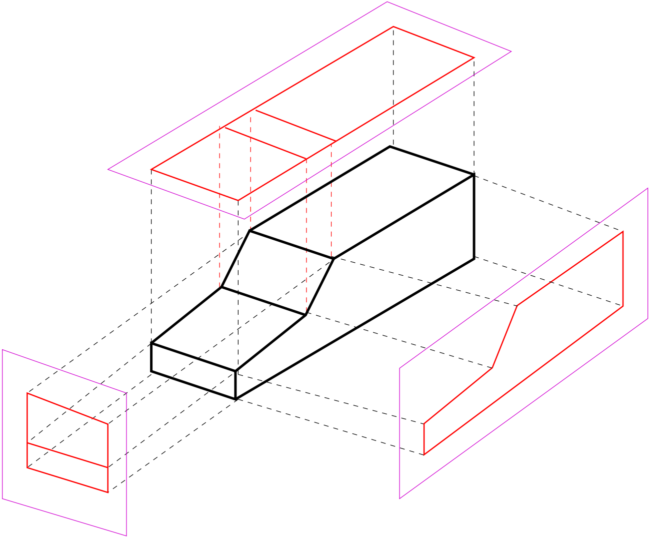
Parallel Projections (cont.)



Orthographic Projections

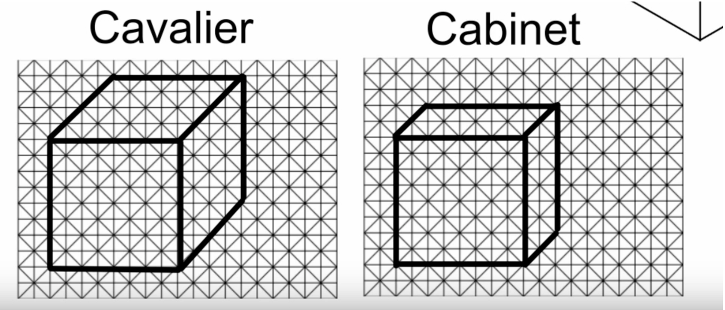
- Lengths parallel to the projection plane are preserved.
- Only direction of projection matters; distance from the point to the projection plane doesn't.
- Good approximation for a camera with a long focal length. (Orthographic with uniform scaling).
- Plan, elevation, side views etc.

Orthographic Projections (cont.)



Oblique Projections

- Cavalier: Length along the depth axis preserved.
- Cabinet: Length along depth axis halved. More realistic.



Orthographic Projection Equation

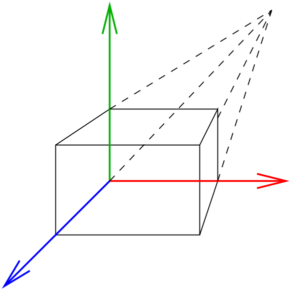
- Can be expressed as a matrix equation:

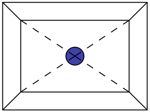
$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- If uniform scaling is involved, the top two 1's should be the scale factor.

Perspective Projections

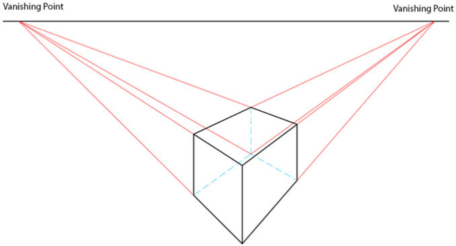
- Can be characterized by the number of **vanishing points**. (projections of points at infinity).
- Depends on the number of axes the projection plane intersects.
- 1-point, 2-point, and 3-point perspective projections.





Perspective Projections

- Can be characterized by the number of **vanishing points**. (projections of points at infinity
- Depends on the number of axe the projection plane intersects
- 1-point, 2-point, and 3-point perspective projections.



Perspective Projections 1-Point Example

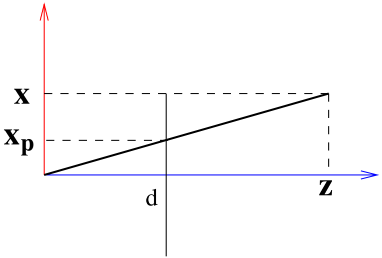
Perspective Projections 2-Point Example

Perspective Projections 3-Point Example

Geometry of Perspective Projection

- What is x_p, y_p, z_p?
- We know X, Z, and d.
- Remember similar triangles?

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

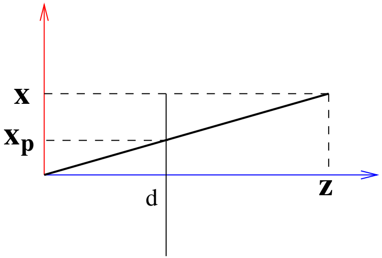


Geometry of Perspective Projection

- $\frac{x_p}{d} = \frac{X}{Z}, \quad \frac{y_p}{d} = \frac{Y}{Z}, \quad z_p = d.$
- In matrix form,

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

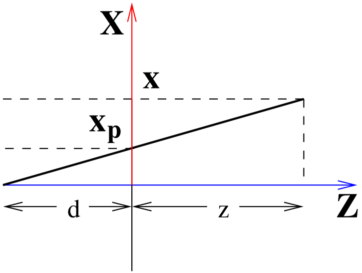
- Coordinates scaled down proportional to the depth or z values.



Another View

- Shift origin to lie on the projection plane, CoP at $(0, 0, -d)$, we get the matrix:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

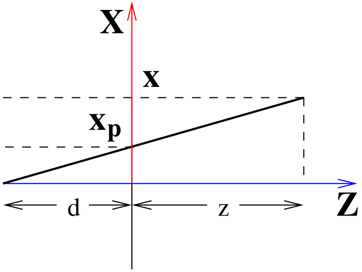


Another View

- Shift origin to lie on the projection plane, CoP at $(0, 0, -d)$, we get the matrix:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Orthographic Projection matrix is a special case when $d \to \infty$.



$$x_p = \frac{Xd}{Z+d} = \frac{X}{1+\frac{Z}{d}}, \quad z_p = 0$$

Projections: Summary

Perspective

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$d < \infty$$

Orthographic

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$d = \infty$$

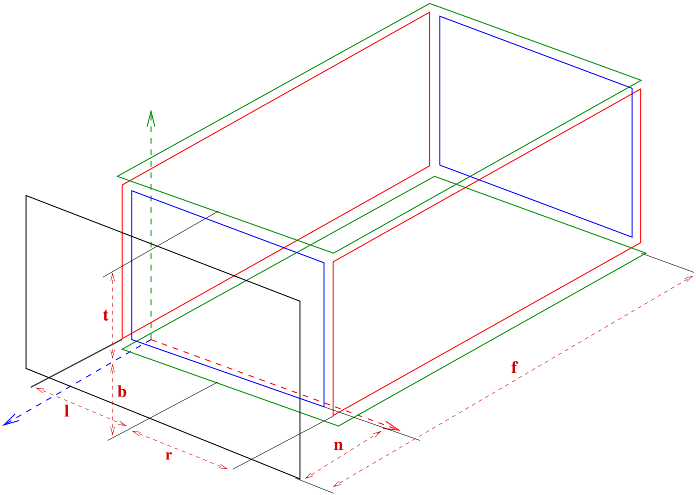
Volume of Visibility

- Cameras have finite fields of view in horizontal and vertical directions.
- What is the shape of its visible space?
- A cylinder for orthographic projections and a cone starting from the CoP for perspective may seem natural.
- Mathematics is difficult for cones; rectangular structures are easier!
- **View Volume:** The volume of potentially visible space.

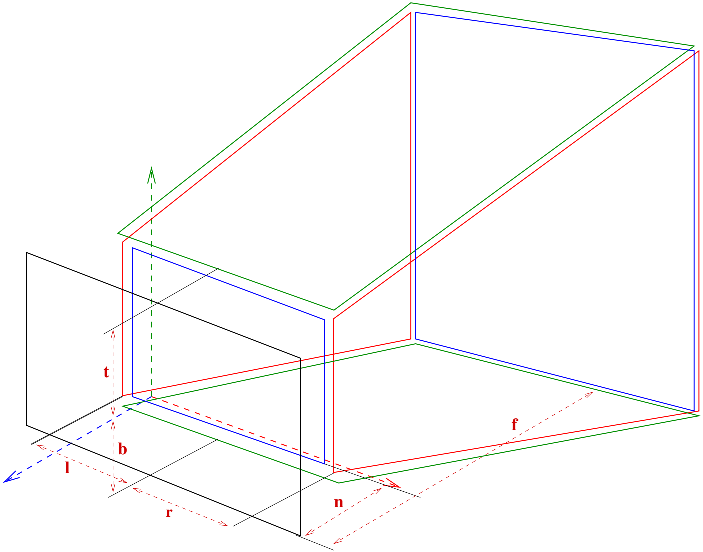
View Volume

- View volume is a cube for orthographic cameras and a (truncated) pyramid for perspective projections.
- 4 planes (left, right, top, bottom) define the view volume.
- Graphics cameras use 2 additional planes to limit visibility: **near & far!**
- Planes are specfied in VRC; they move with the camera

Orthographic View Volume

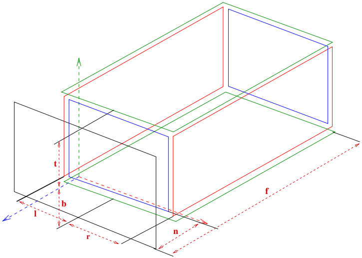


Perspective View Volume

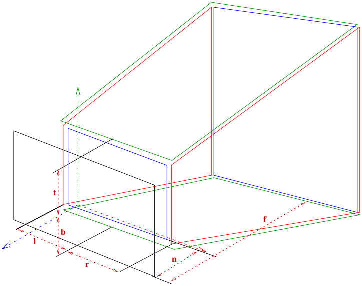


Orthographic & Perspective

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$



- View volume: Top, Bottom, Left, Right, Near, Far!

What About the Focal Length?

- An ideal pin-hole camera has the whole world in focus.
- Finite focal-length lenses introduce the effect of focus in real cameras.
- Even for them, the depth of field (region in focus) increases as the f-stop increases or the aperture gets smaller.
- Computer Graphics simulates ideal pin-hole cameras.
- Depth of field can be simulated by intentional blurring.

View Volume Specification

- View volume is specified by 6 planes:
  left, right, top, bottom, near, far. All values in VRC

  `glm::frustum()` or `glm::ortho()`

- left, right, top, bottom: signed distances.
  near, far: positive distances to planes.

- Needn't be symmetric!

Alternate Specification

- Symmetric view volumes: horizontal and vertical fields of view $\theta_{\mathbf{h}}, \quad \theta_{\mathbf{v}}$
- For symmetric perspective view volumes:

  $\tan \frac{\theta_h}{2} = \ldots?$

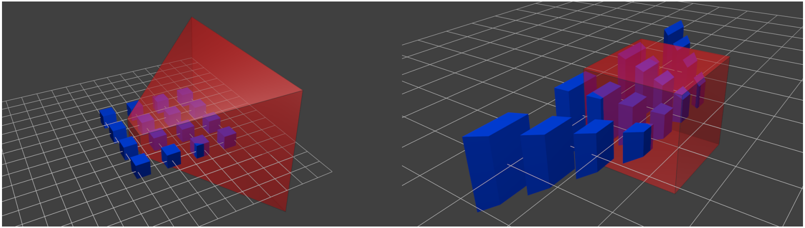
  $\tan \frac{\theta_v}{2} = \ldots?$
- Aspect ratio: top / right.

  Using `glm::perspective()`

Canonical View Volume

- Projection is not performed right away; instead, map the view volume to a cube of fixed dimensions, called the **canonical view volume** or a standard view volume
- A **normalizing matrix** performs this transformation.
- Why?
  - Easier to eliminate objects outside the view volume.
  - Orthographic & perspective aren't different.
  - The z-coordinates not thrown away. (Used later!)

Canonical View Volume: Visualization

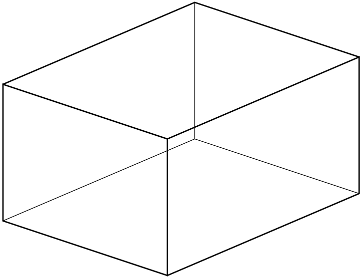


http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

Canonical View Volume: Dimensions

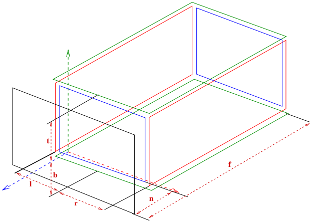
- OpenGL:

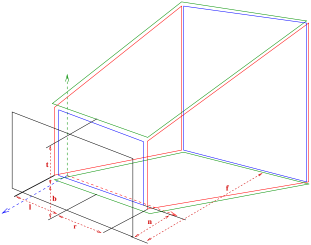
$$\begin{aligned}-1 &\leq x \leq 1\\ -1 &\leq y \leq 1\\ -1 &\leq z \leq 1\end{aligned}$$



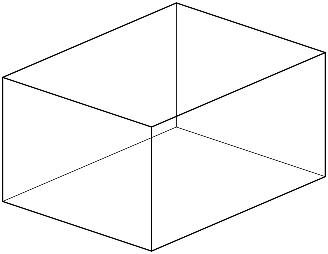
- Canonical view volume is the Orthographic View Volume, with appropriate scaling.

Orthographic & Perspective





To target view volume:



Orthographic Normalizing Matrix

- What are the side lengths on start?
- What are the side lengths at end?
- Whats the scale factor?
- Where is the origin at start? At end?
- How do we achieve that?

Orthographic Normalizing Matrix

- Lengths (right - left), (top - bottom) and (far - near) scaled to 2.
- Shift origin so as to range from -1 to +1.
- Matrix??

Orthographic Normalizing Matrix (cont.)

- Matrix:

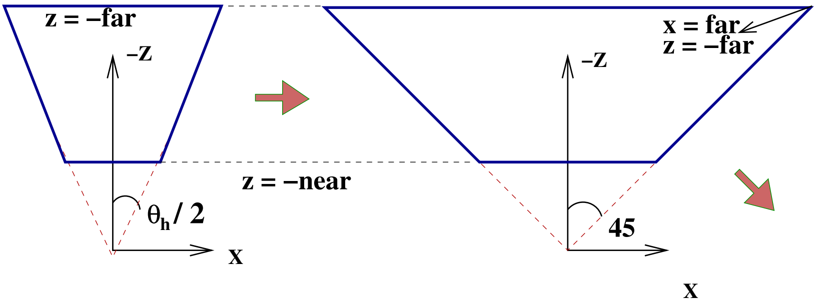
$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{far-near} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $(0, 0, -near)$ maps to $+1$ and $(0, 0, -far)$ maps to -1.
- Drop z and use the (x, y) coordinates as the (normalized) window coordinates.

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.
- $s_x = s_y = s_z = ??$

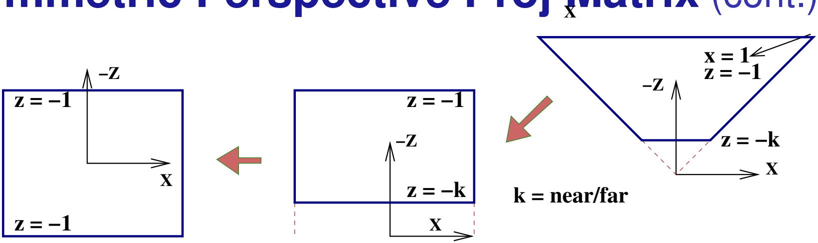
Symmetric Perspective Proj Matrix (cont.)



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 $s_x = s_y = s_z = \frac{1}{\mathbf{far}}$
- $$M_2M_1 = \begin{bmatrix} \frac{\cot\theta_h/2}{\mathbf{far}} & 0 & 0 & 0 \\ 0 & \frac{\cot\theta_v/2}{\mathbf{far}} & 0 & 0 \\ 0 & 0 & \frac{1}{\mathbf{far}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- 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.)



Symmetric Perspective Proj Matrix (cont.)

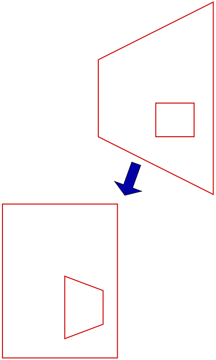
- Scale by to $-z$ to convert to a cube using a matrix with last row $[0 \quad 0 \quad -1 \quad 0]$.
- Simultaneously send third component to range $[-1, 1]$. $z = -k$ maps to 1 and $z = -1$ maps to -1.
- Scale z by $\frac{1+k}{1-k}$ and translate by $\frac{2k}{1-k}$.
- Perspective division: convert from homogeneous to cartesian by dividing by the last component.
- Keep third component for later use. Relative ordering needs to be preserved. The values are not important.

Perspective Normalizing Txform

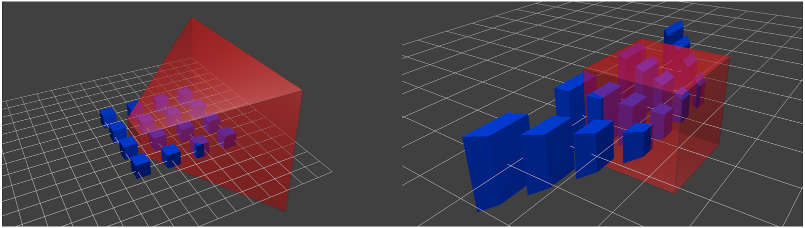
- Matrix M_3 for this step:

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

- $(x, y, -k)$ & $(x, y, -1)$ 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.