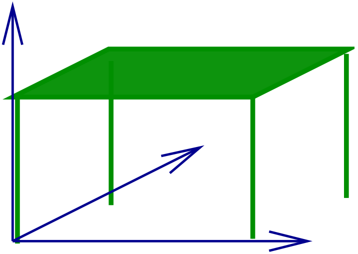
CSE251
Basics of Computer Graphics
Module: Graphics Pipeline

Avinash Sharma

Spring 2017

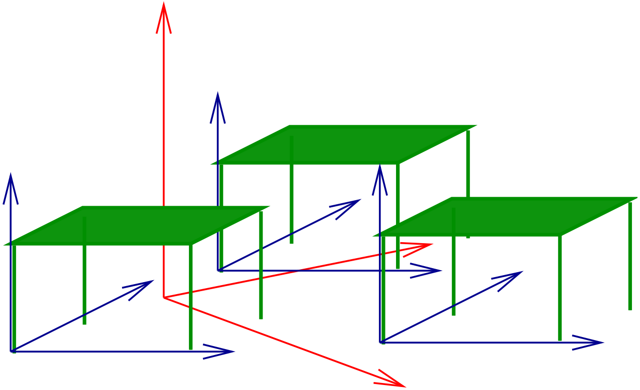
A Single Table

A table defined in its own coordinate system.

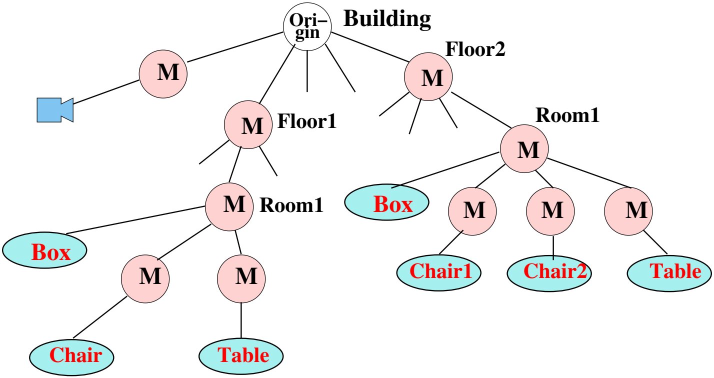


Many Tables in a Room

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

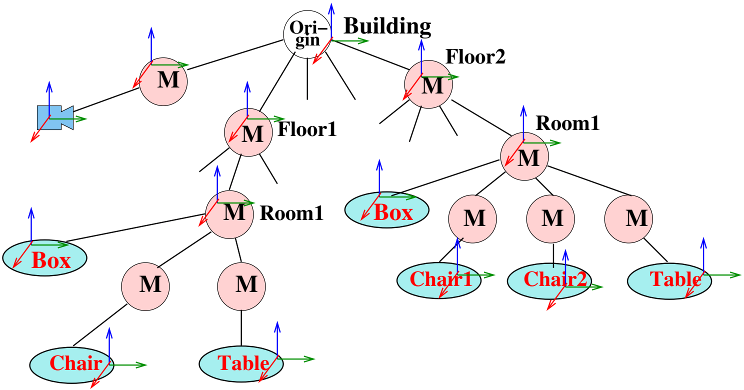


A Building Model



Hierarchical model with root representing whole scene.

A Building Model



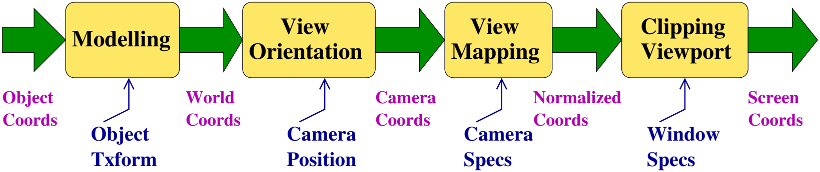
Each matrix **M** aligns parent frame to child frame

Different Coordinates

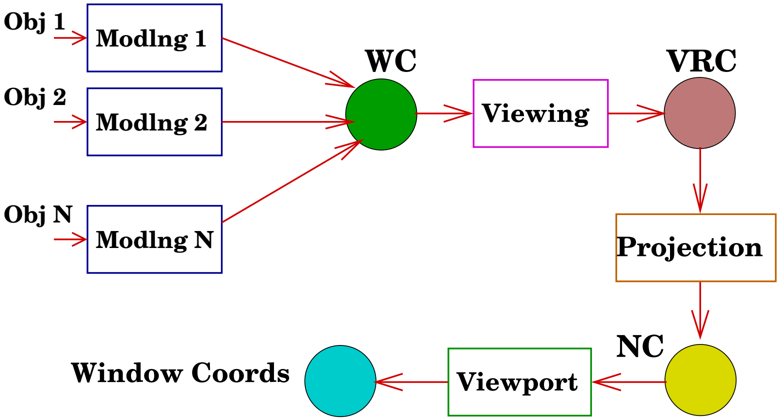
- **Object Reference:** Object is described in an internal coordinate frame called **ORC**.
- **World:** Common reference frame to describe different objects, called **WC**.
- **Camera/View Reference:** Describe with respect to the current camera position/orientation, called **VRC**
- Ultimately, **how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

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

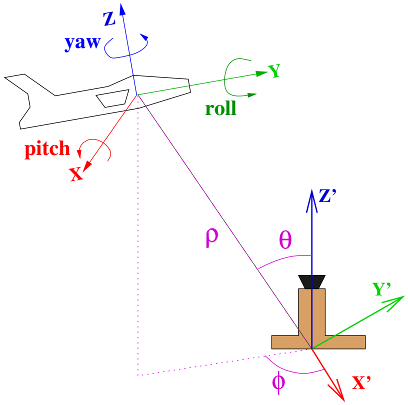
- Goal: Evaluate the coordinates of each point/line/triangle with respect to the camera

Modelling

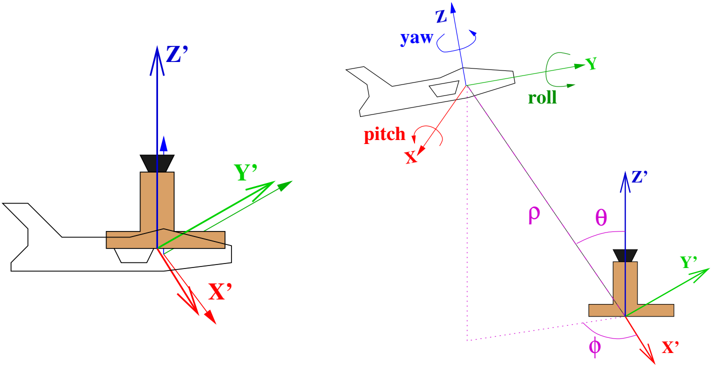
- Goal: Transform object coordinates to world coordinates.
- Method: Place ORC frame in the world coordinate frame.
- A single transformation matrix or **modelling matrix** with translation, rotation, scaling.
- A unit cube at origin can generate any cuboid using translation/rotation/scaling.
- Different 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: $\mathbf{T}(\rho, \theta, \phi)\ \mathbf{R_z}(\mathbf{y})\ \mathbf{R_x}(\mathbf{p})\ \mathbf{R_y}(\mathbf{r})$
- 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?

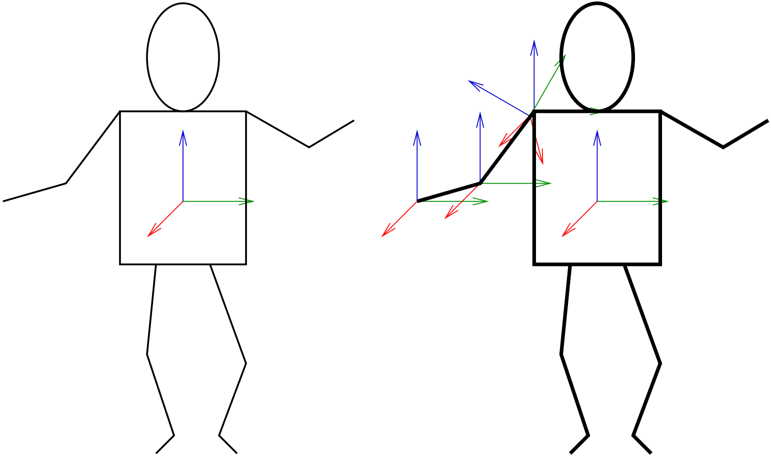
Why yaw, pitch, roll?

- 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!

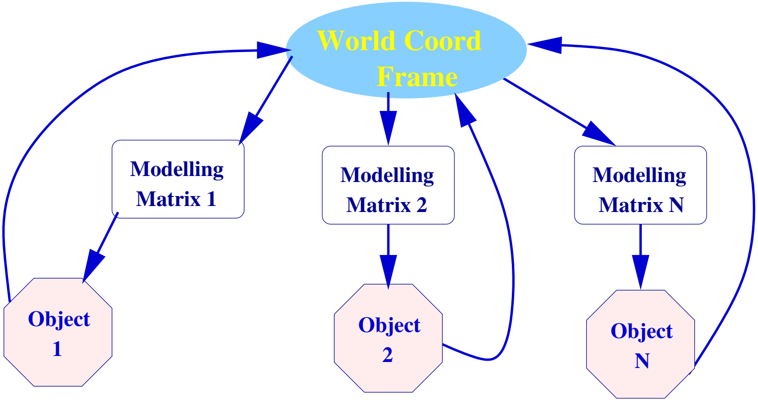
Hierarchy of Transformations

- 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.

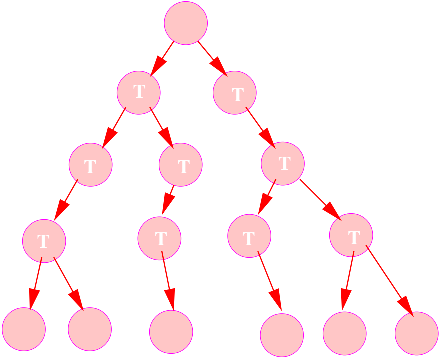
Humanoid Robot



Modelling Different Objects



Scene Graph



- Objects organized hierarchically with transforms.

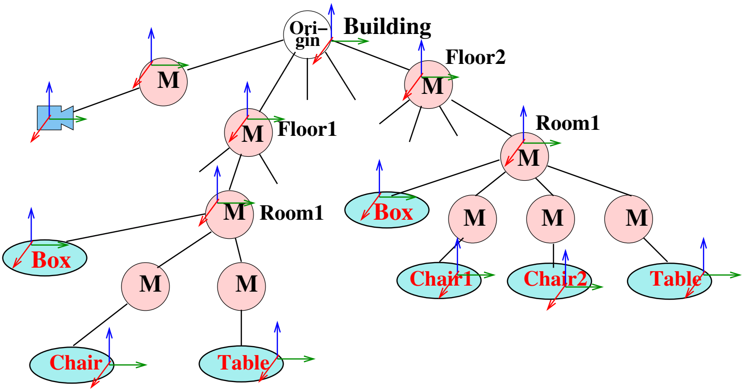
Modelling in OpenGL

- 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: $\mathbf{M = T\ R_Z(y)\ R_X(p)\ R_Y(r)}$

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



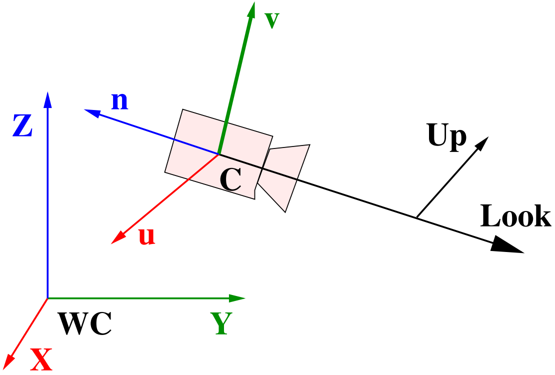
Each matrix **M** aligns parent frame to child frame

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: $\bar{\mathbf{v}} = \bar{\mathbf{n}} \times \bar{\mathbf{u}}$.

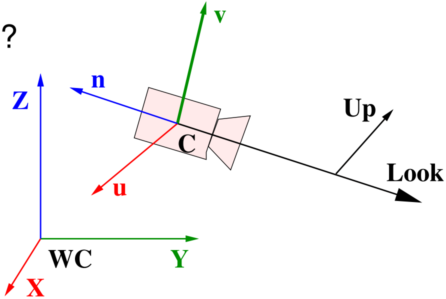
View Orientation Transformation

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

- What have we achieved?

$\mathbf{P}_{\mathbf{WC}} = \mathbf{A}\ \mathbf{P}_{\mathbf{VRC}}$ or

$\mathbf{P}_{\mathbf{VRC}} = \mathbf{A}\ \mathbf{P}_{\mathbf{WC}}$?



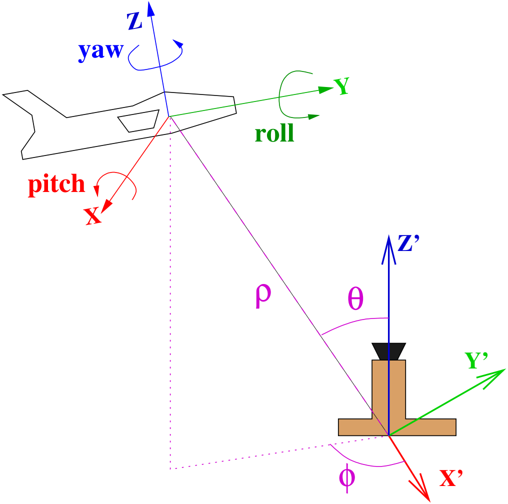
View Orientation Transformation

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

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

Viewing from the Aircraft

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



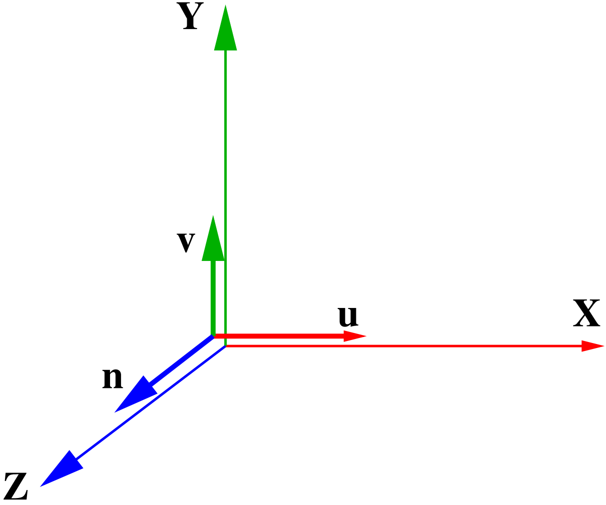
Aircraft in Polar World: Viewing

- 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

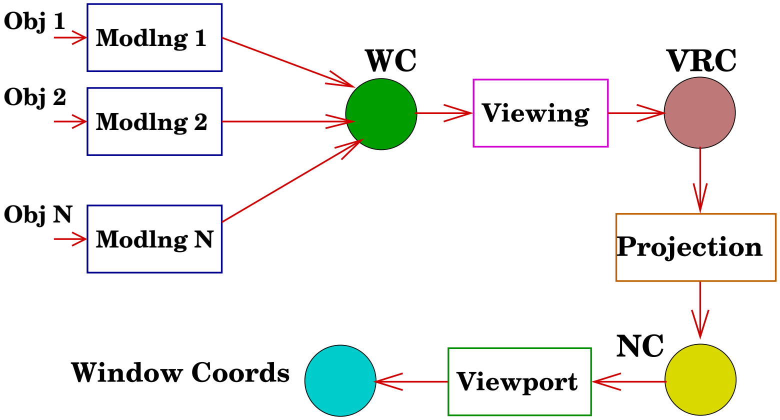


Modelling and Viewing matrices are Identity.

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 $\mathbf{P\ V\ M_i}$ 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

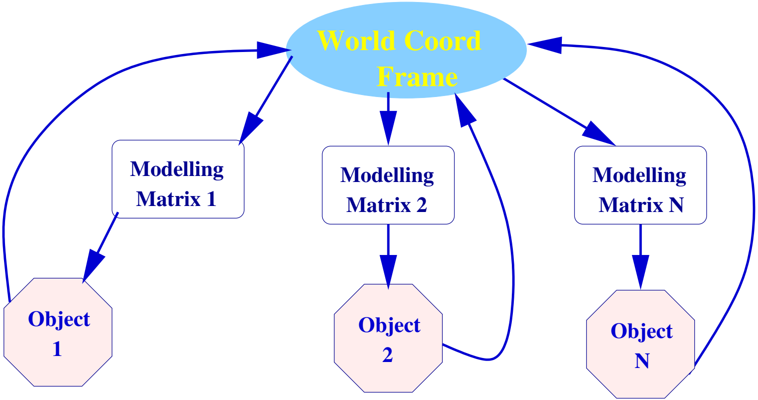
$\mathbf{Mat} = \mathbf{P}\ \mathbf{V}\ \mathbf{M}$ // from MVP matrix

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
- $\mathbf{P}_{\mathsf{VRC}} = \mathbf{V} \cdot \mathbf{M} \cdot \mathbf{P}_{\mathsf{ORC}}$