UNIVERSITY OF DERBY

5CM507 Graphics

Lecture 03 Modelling and Viewing

Dr Youbing Zhao

October 5, 2025

Last Week

- 2D & 3D Scaling, Translation
- Homogeneous Coordinates
- 2D Rotation
- 3D Rotation : Euler angles
- Composition of Transforms

Contents

Position and Orient Models

Position and Orient Models a the Scene

All the World's a Stage

[http://www.mobilegamespro.com/battle3dz/]

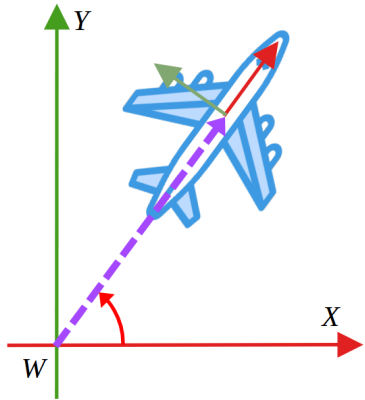
Notation

- Object/body frame/space : defines the model; axes rotate with the object (For simplicity, we assume all objects are already scaled)
- Global/world frame/space : contains all objects in the scene; root frame for the scene hierarchy
- Intermediate/local space/frame : parent and child frames in the scene hierarchy

Borrowed from robotics, ${}^{A}_{B}\mathbf{M}$ denotes a matrix transforms:

- coordinates from frame B to frame A ($B \Rightarrow A$)

Example: ${}^{W}_{O}\mathbf{R}$: *rotation*,Object $\Rightarrow$ *World*



The object and the world space

Position and Orient Models in the World Space

All the World's a Stage

1. Scale the model with $\mathbf{S}(s_x, s_y, s_z)$ in the object space.
2. Orient the model with a rotation $\mathbf{R}$ in the object space.
3. Position the object with a translation $\mathbf{T}$ relative to world frame.

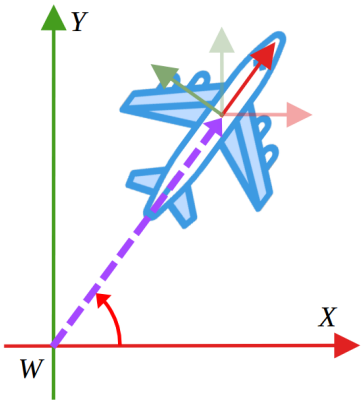
$$^{W}_{O}\mathbf{M} = \mathbf{TRS}$$

We interpret $^{A}_{B}\mathbf{M}$ as if we move and rotate frame **A** to fit frame **B**.
To derive the matrix order of **T** and **R**, we use an helper frame $\tilde{O}$ parallel to the world frame and located at the object frame origin:

1. Imagine to move the world frame to frame $\tilde{O}$: $^{W}_{\tilde{O}}\mathbf{M} = {}^{W}\mathbf{T}$
2. Imagine to rotate frame $\tilde{O}$ to the scaled object frame O_s: $^{\tilde{O}}_{O_s}\mathbf{M} = \mathbf{R}$

Finally, we concatenate them from left to right

$$^{W}_{O}\mathbf{M} = {}^{W}_{\tilde{O}}\mathbf{M}\ {}^{\tilde{O}}_{O_s}\mathbf{M}\ {}^{\tilde{O_s}}_{O}\mathbf{M} = {}^{W}\mathbf{TRS}$$



The object and the world space

Position and Orient Models in the World Space

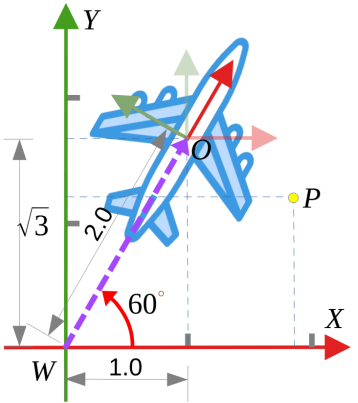
An example

$$ {}^{W}_{O}\mathbf{M} = {}^{W}_{\tilde{O}}\mathbf{M}\,{}^{\tilde{O}}_{O}\mathbf{M} = {}^{W}\mathbf{TR} $$

$$ {}^{W}_{O}\mathbf{M} = \mathbf{T}(1,\sqrt{3},0)\mathbf{R_z}(60^\circ) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & \sqrt{3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 0 \\ \frac{\sqrt{3}}{2} & \frac{\sqrt{1}}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 1 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 & \sqrt{3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{W}_{O}\mathbf{M}O(0,0,0) = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 1 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 & \sqrt{3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \sqrt{3} \\ 0 \\ 1 \end{bmatrix} $$

$$ {}^{W}_{O}\mathbf{M}P(0,-1,0) = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 1 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 & \sqrt{3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+\frac{\sqrt{3}}{2} \\ -\frac{1}{2}+\sqrt{3} \\ 0 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 1.87 \\ 1.23 \\ 0 \\ 1 \end{bmatrix} $$



An *TR* example

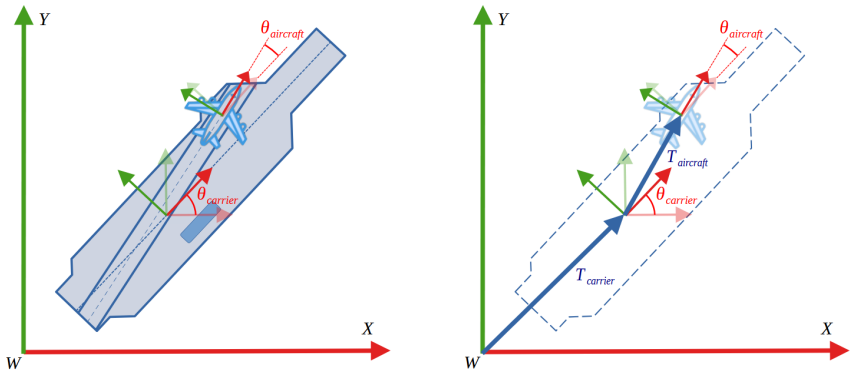
Hierarchical Transforms

Hierarchical Transforms

Add one object level

Our method can be extended to a hierarchy.

$$ {}^{W}_{a}\mathbf{M} = {}^{W}_{c}\mathbf{M}\,{}^{c}_{a}\mathbf{M} = {}^{W}_{c}\mathbf{TR}_{\mathbf{c}}\,{}^{c}_{a}\mathbf{TR}_{\mathbf{a}} $$



Two level transform

Hierarchical Transforms

Question

If in your game, the aircraft carrier moves a distance d along its heading direction at time t_1, what is the new transform of the aircraft?

The key difference is that the translation is relative to the carrier frame, not the world.

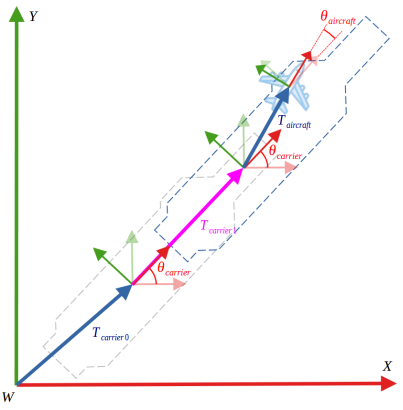
We have a new intermediate frame c_1 and

$$ {}^{c}_{c_1}\mathbf{M} = {}^{c}_{c_1}T(d,0,0) = {}^{c}_{c_1}T_x(d) $$

Now we add this transform in the middle

$$ {}^{W}_{a}\mathbf{M} = {}^{W}_{c}\mathbf{M}\,{}^{c}_{c_1}\mathbf{M}\,{}^{c_1}_{a}\mathbf{M} = {}^{W}_{c}\mathbf{TR_c}\,{}^{c}_{c_1}T_x(d)\,{}^{c_1}_{a}\mathbf{TR_a} $$

If we read from left to right, it still follows the world frame => intermediate frames => object frame order.



Hierarchical Transforms

Robot arms

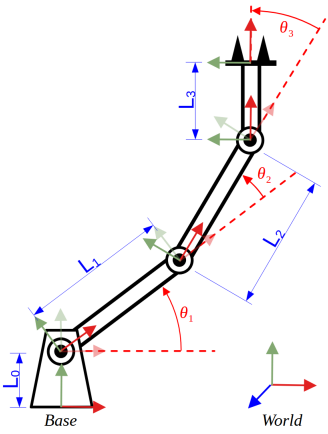
Problem : with a set of joint angles and arm length (θ_i, L_i), calculate the position of the gripper.

- Joint (rotation), each joint is associated with a local frame and normally rotates about one axis
- Link (arm, translation)

Translation of robot arms is also relative to local frames. Again, we can write the transforms from left to right down from world frame to the gripper via joint frames:

$$ {}^{W}_{g}\mathbf{M} = T_{base} T_y(L_0) R(\theta_1) T_x(L_1) R(\theta_2) T_x(L_2) R(\theta_3) T_x(L_3) $$

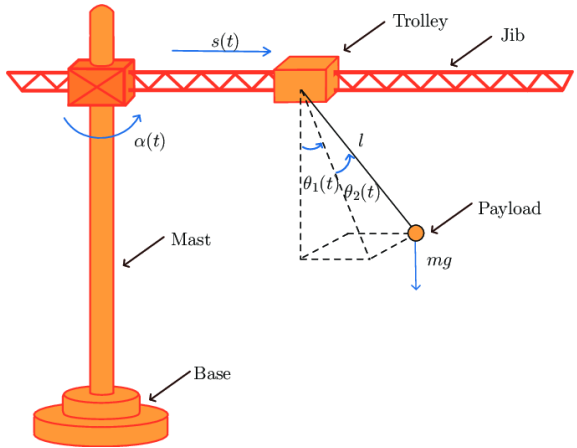
How about adding rotation for the base or gripper ?



A three link planar robot arm

Our turn

How to compute the trolley position of the tower crane in the world space ?

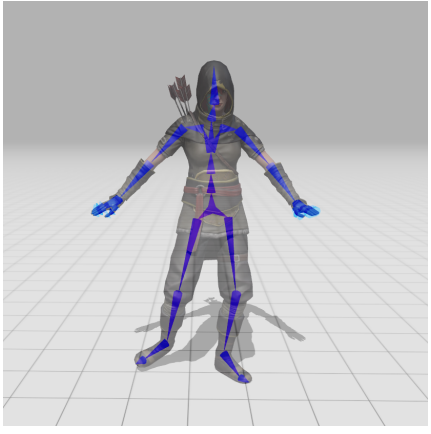


Hierarchical Transforms

Skeletal animation

- Character rigging: manipulate skeleton joints to define poses
  - Pose : $\{\theta_i(t)\}$
  - Joint : degree-of-freedom ≥ 1
- Forward Kinematics : calculate the pose of every bone
- Skinning: binding a 3D mesh to the skeleton

To cover it in the next term.



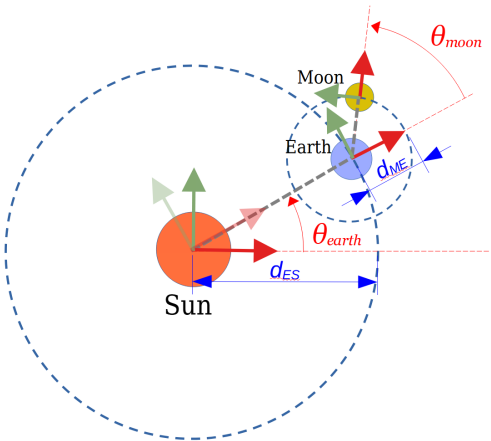
[Basel]

Sun-Earth-Moon System

Assume all orbits are circles and no tilts. Rotation transform include revolution and self-rotation.

To calculate the transform of the moon surface.

1. Sun centre frame => Earth center frame
   $${}^{S}_{Ec}\mathbf{M} = R_{Ec\_rev}T_{ES}$$
2. Earth center frame => Moon center frame:
   $${}^{S}_{Mc}\mathbf{M} = R_{Ec\_rev}T_{ES}R_{Mc\_rev}T_{ME}$$
3. Moon's surface rotation relative to the centre (actually no rotation as its rotation period equals its revolution period):
   $${}^{S}_{M}\mathbf{M} = R_{Ec\_rev}T_{ES}R_{Mc\_rev}T_{ME}R_{M\_rot}$$



Sun-Earth-Moon System

A hierarchical transform tree

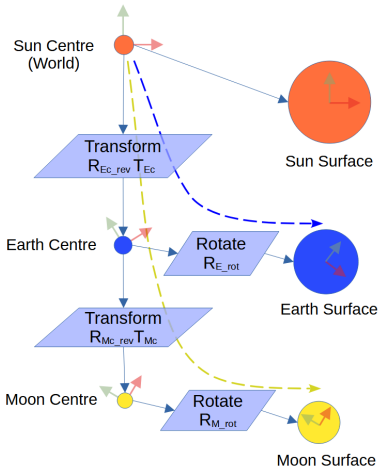
We can organise the transforms in a hierarchical tree structure and write the transforms in a top-down order.

- Earth surface to Sun

  $${}^{S}_{e}\mathbf{M} = {}^{S}_{Ec}\mathbf{M} R_{E\_rot} = R_{E\_rev} T_{ES} R_{E\_rot}$$

- Moon surface to Sun

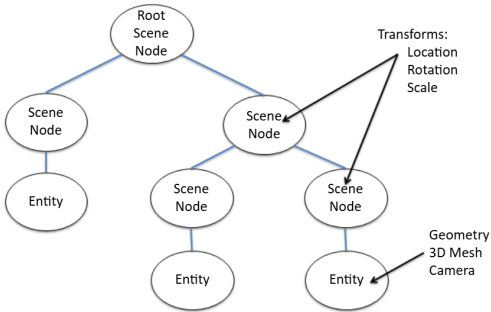
  $${}^{S}_{M}\mathbf{M} = {}^{S}_{Ec}\mathbf{M}\ {}^{Ec}_{Mc}\mathbf{M}\ {}^{Mc}_{M}\mathbf{M} = R_{E\_rev} T_{Earth} R_{M\_rev} T_{ME} R_{M\_rot}$$



Hierarchical Modelling - Scene Graph

Scene Graph

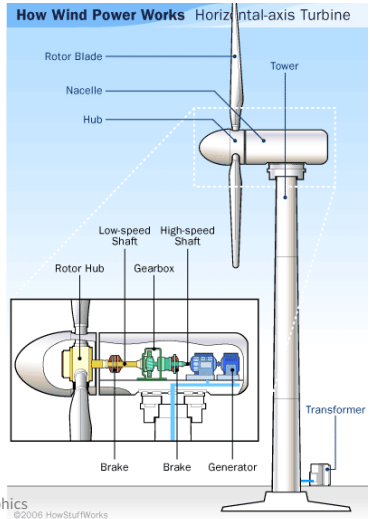
- A hierarchical tree structure of a model or scene, includes
  - Geometry
  - Transform
  - Material, Texture
  - Camera, Light Source ...
  - Group
- Transforms and Geometries can either be nodes, or properties attached to a node
- Used by game engines, modelling software, high-level graphics libs
  - Unreal, Unity
  - OpenSceneGraph/VulcanSceneGraph
  - OGRE3d, Three.js, Java3D ...

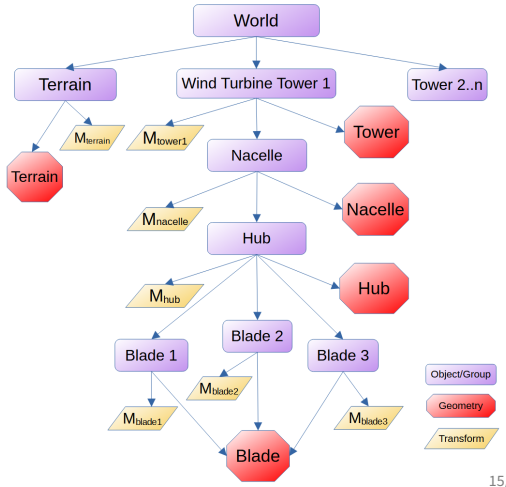


A concept diagram of Ogre3D scenegraph

Traverse a Scene Graph of a Wind Farm

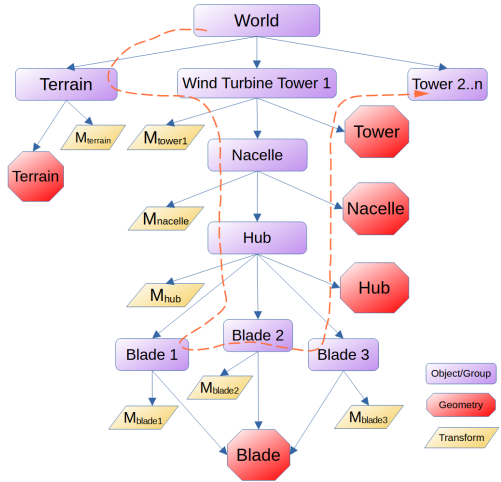
- Drawing a scene is equivalent to traverse its scene graph (tree) .





Traverse a Scene Graph with a Matrix Stack

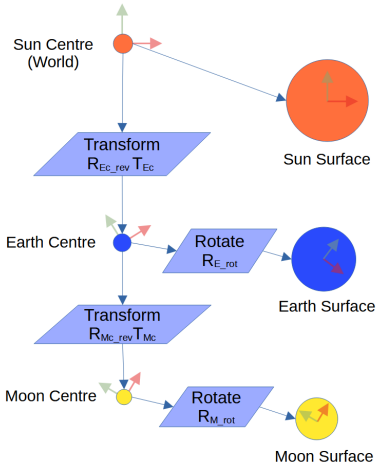
- Use a matrix stack to keep track of current transformations.



Object Oriented Design - Inheritance or Composition

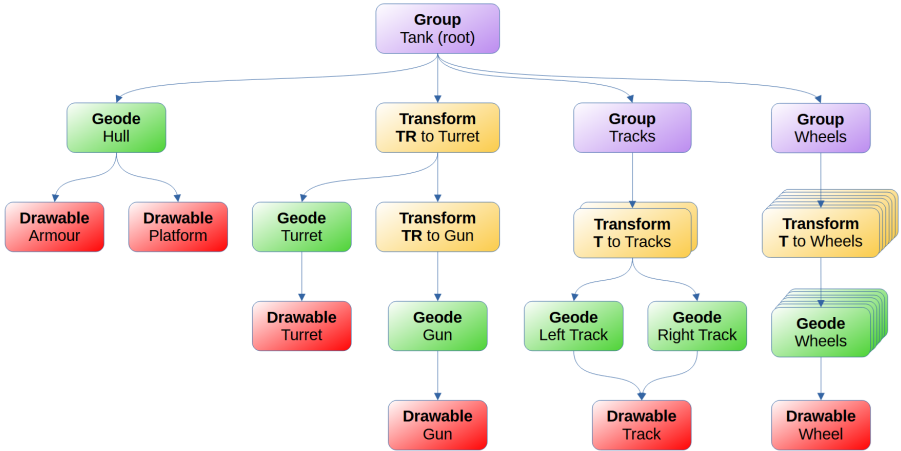
- Inheritance-based
  - Transform is a type of Node
  - A Transform Node can have other Node children, such as shapes
  - OpenSceneGraph, Java3D, OpenInventor ...
  - More used by scene graph libraries
- Composition-based
  - A Node has a Transform
  - e.g. Unreal, Unity, Blender; OGRE3D, Three.js ...
  - More used in modelling

Ask AI or check APIs for more.



Scene Graph - an OpenSceneGraph example

A Tank model



An example scene graph of a tank model in OpenSceneGraph

Change Reference Frames (for derivation of the view matrix, can be skipped)

Direct Change of Reference Frames

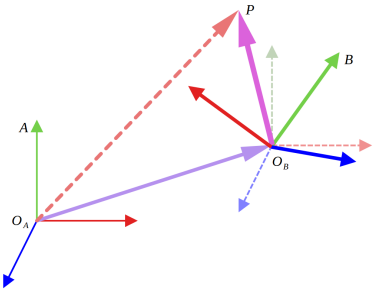
Transform from frame B to frame A: ${}^A_B\mathbf{M} = \mathbf{TR}$, can be obtained as if:

1. Translate frame A's origin to frame B's origin : ${}^A\mathbf{T}$
2. Rotate the intermediate frame to align with frame B : ${}^A_B\mathbf{R}$

According to linear algebra, the rotation ${}^A_B\mathbf{R}$ takes the axis vectors of frame B- ${}^A\vec{X}_B, {}^A\vec{Y}_B, {}^A\vec{Z}_B$ - as column vectors.

Frame B's origin, ${}^AO_B$, described in frame A, forms the translation.

$$ {}^A_B\mathbf{R} = \begin{pmatrix} {}^AX_B^x & {}^AY_B^x & {}^AZ_B^x & 0 \\ {}^AX_B^y & {}^AY_B^y & {}^AZ_B^y & 0 \\ {}^AX_B^z & {}^AY_B^z & {}^AZ_B^z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, {}^A\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & {}^AO_B^x \\ 0 & 1 & 0 & {}^AO_B^y \\ 0 & 0 & 1 & {}^AO_B^z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$



Converting frame $B \Rightarrow A$, looks like overlapping **T** and **R**

$$ {}^A_B\mathbf{M} = {}^A_B\mathbf{T}\,{}^A_B\mathbf{R} = \begin{pmatrix} 1 & 0 & 0 & {}^AO_B^x \\ 0 & 1 & 0 & {}^AO_B^y \\ 0 & 0 & 1 & {}^AO_B^z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^AX_B^x & {}^AY_B^x & {}^AZ_B^x & 0 \\ {}^AX_B^y & {}^AY_B^y & {}^AZ_B^y & 0 \\ {}^AX_B^z & {}^AY_B^z & {}^AZ_B^z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} {}^AX_B^x & {}^AY_B^x & {}^AZ_B^x & {}^AO_B^x \\ {}^AX_B^y & {}^AY_B^y & {}^AZ_B^y & {}^AO_B^y \\ {}^AX_B^z & {}^AY_B^z & {}^AZ_B^z & {}^AO_B^z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Direct Change of Reference Frame

From linear algebra : matrix converts $A \Rightarrow B$ is the inverse matrix which converts B to A.

The inverse of a rotation matrix is its transpose : $\mathbf{R^{-1}} = \mathbf{R^T}$

The inverse of a translation $\mathbf{T}^{-1}(t_x, t_y, t_z) = \mathbf{T}(-t_x, -t_y, -t_z)$.

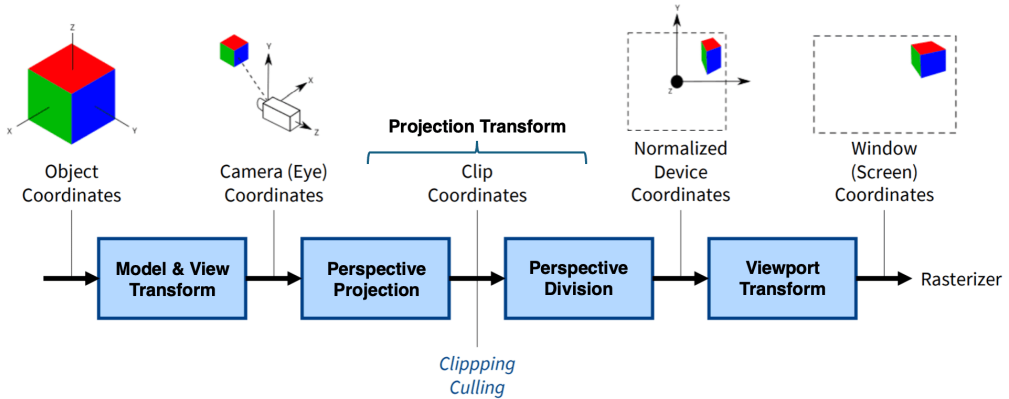
$$
{}^B_A\mathbf{M} = {}^A_B\mathbf{M}^{-1} = ({}^A_B\mathbf{T}\,{}^A_B\mathbf{R})^{-1} = {}^A_B\mathbf{R}^{-1}\,{}^A_B\mathbf{T}^{-1} = {}^A_B\mathbf{R}^T\,{}^A_B\mathbf{T}^{-1}
$$

$$
= \begin{pmatrix} {}^A\chi^x_B & {}^A\chi^y_B & {}^A\chi^z_B & 0 \\ {}^A\gamma^x_B & {}^A\gamma^y_B & {}^A\gamma^z_B & 0 \\ {}^A z^x_B & {}^A z^y_B & {}^A z^z_B & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -{}^A O^x_B \\ 0 & 1 & 0 & -{}^A O^y_B \\ 0 & 0 & 1 & -{}^A O^z_B \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
= \begin{pmatrix} {}^A\chi^x_B & {}^A\chi^y_B & {}^A\chi^z_B & -{}^A\vec{O}^x_B\,{}^A\chi^x_B - {}^A\vec{O}^y_B\,{}^A\chi^y_B - {}^A\vec{O}^z_B\,{}^A\chi^z_B \\ {}^A\gamma^x_B & {}^A\gamma^y_B & {}^A\gamma^z_B & -{}^A\vec{O}^x_B\,{}^A\gamma^x_B - {}^A\vec{O}^y_B\,{}^A\gamma^y_B - {}^A\vec{O}^z_B\,{}^A\gamma^z_B \\ {}^A z^x_B & {}^A z^y_B & {}^A z^z_B & -{}^A\vec{O}^x_B\,{}^A z^x_B - {}^A\vec{O}^y_B\,{}^A z^y_B - {}^A\vec{O}^z_B\,{}^A z^z_B \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} {}^A\chi^x_B & {}^A\chi^y_B & {}^A\chi^z_B & -{}^A\vec{O}_B \cdot {}^A\vec{\chi}_B \\ {}^A\gamma^x_B & {}^A\gamma^y_B & {}^A\gamma^z_B & -{}^A\vec{O}_B \cdot {}^A\vec{\gamma}_B \\ {}^A z^x_B & {}^A z^y_B & {}^A z^z_B & -{}^A\vec{O}_B \cdot {}^A\vec{z}_B \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Model-View Transformation

Modelling, Viewing and Projection (MVP) chain



$$\mathbf{H} = \mathbf{PVM}$$

Viewing Transform

Position and Orient the Camera

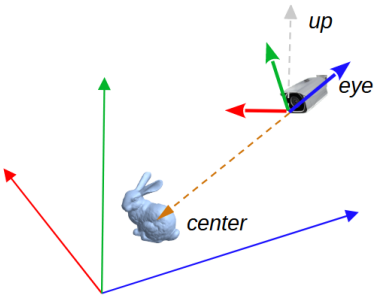
Frame Transformation: World space ⇒ Camera space For projection

Camera Space (Right-handed):

- Eye position described in the world space
- Major axes of described in the world space
  - Can be derived from the LookAt function

Method :

- Use knowledge of Frame Transformation
- Use Camera frame axes as row vectors to build a viewing matrix



The Camera Space

The LookAt function

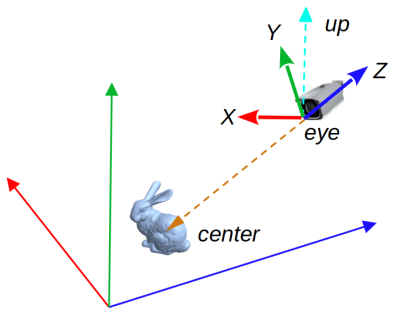
The LookAt function specifies the major axes of the camera space described in the world space

```
glm::mat4 glm::lookAt(glm::vec3 eye, glm::vec3 center, glm::vec3 up);
```

The `LookAt` function defines a view transform.

- *eye* : the location of the eye/camera
- *center* : the looking-at centre point, defines the camera LookAt (Z) direction
- $\vec{up}$: the up vector, defines the camera YZ plane

$$\mathbf{Z} = \frac{\mathbf{e}-\mathbf{c}}{\|\mathbf{e}-\mathbf{c}\|}, \mathbf{X} = \frac{\mathbf{u}\times\mathbf{Z}}{\|\mathbf{u}\times\mathbf{Z}\|}, \mathbf{Y} = \mathbf{Z}\times\mathbf{X}$$



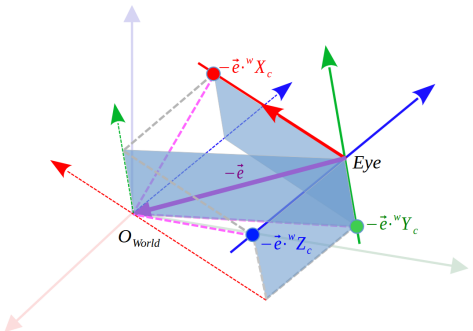
Viewing Transform

The view matrix

Applying the frame transform formula, we have

$$\mathbf{V} = {}^{c}_{w}\mathbf{M} = \begin{pmatrix} {}^{w}X_c^x & {}^{w}X_c^y & {}^{w}X_c^z & -\vec{e}\cdot{}^{w}\vec{X}_c \\ {}^{w}Y_c^x & {}^{w}Y_c^y & {}^{w}Y_c^z & -\vec{e}\cdot{}^{w}\vec{Y}_c \\ {}^{w}Z_c^x & {}^{w}Z_c^y & {}^{w}Z_c^z & -\vec{e}\cdot{}^{w}\vec{Z}_c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$(-\vec{e}\cdot{}^{w}\vec{X}_c, -\vec{e}\cdot{}^{w}\vec{Y}_c, -\vec{e}\cdot{}^{w}\vec{Z}_c)$ is the projection of the translation vector in the camera space.



Summary

- Position and Orient Models
- Hierarchical Transformation
- Hierarchical Modelling - Scene Graph
- Viewing Transformation

Questions?

You are welcome to take the poll !