UNIVERSITY OF DERBY

5CM507 Graphics

Lecture 03 Modelling and Viewing

Dr Youbing Zhao

October 7, 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 in a Scene

All the World's a Stage

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

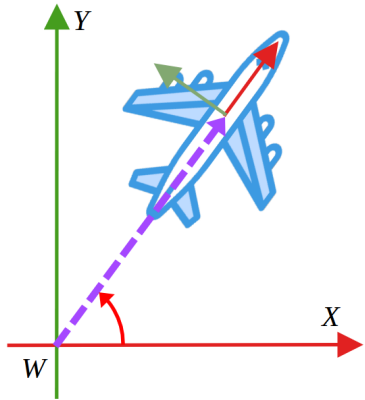
Notation

- 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, ${}^{A}_{B}\mathbf{M}$ is 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

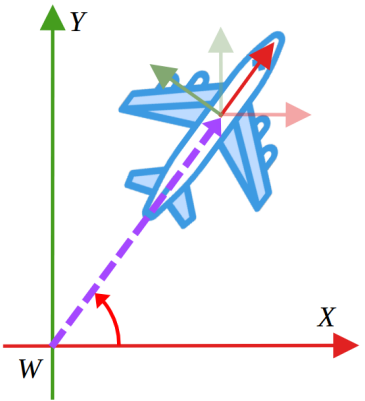
All the World's a Stage

To add a 3D model directly into the world space, we apply a series of operations

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

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

As scaling is only applied once at the beginning, we are going to ignore **S** in the future. The key is **T** and **R**.



The object and the world space

Position and Orient Models in the World Space

An Example

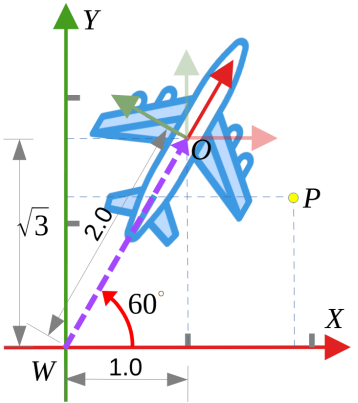
$$ {}^{W}_{O}\mathbf{M} = {}^{W}_{\tilde{O}}\mathbf{M}\,{}^{\tilde{O}}_{O}\mathbf{M} = {}^{W}\mathbf{TR} $$

$$ {}^{W}_{O}\mathbf{M} = \mathbf{T}(1, \sqrt{3}, 0)\mathbf{R_z}(60^\circ) = $$

$$ \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & \sqrt{3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 0 \\ \frac{\sqrt{3}}{2} & \frac{\sqrt{1}}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 1 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 & \sqrt{3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

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

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



A **TR** example

Derive the Order of Transforms

A Frame Centred Top-down Approach

Case 1 : Position an aircraft model at (x, y) with ${}^{W}\mathbf{T}(\mathbf{x}, \mathbf{y})$

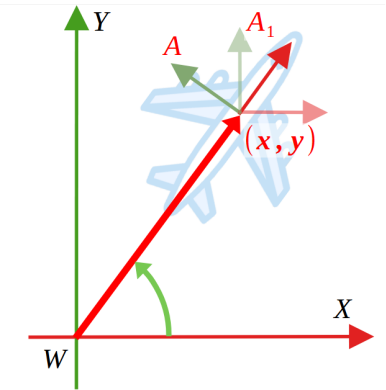
${}^{W}_{A}\mathbf{M}$: transform coordinates from frame Object to World

Think in Opposite: Align World with Object

Intermediate frame A_1 : parallel to World, origin at (x, y):

1. Translate frame W to A_1: ${}^{W}_{A_1}\mathbf{M} = {}^{W}\mathbf{T}(\mathbf{x}, \mathbf{y})$
2. Rotate frame A_1 to A: ${}^{A_1}_{A}\mathbf{M} = \mathbf{R}$
3. Concatenate from left to right

$$ {}^{W}_{A}\mathbf{M} = {}^{W}_{A_1}\mathbf{M}\,{}^{A_1}_{A}\mathbf{M} = {}^{W}\mathbf{T}(\mathbf{x}, \mathbf{y})\mathbf{R} $$



TR order

Derive the Order of Transforms

A Frame Centred Top-down Approach

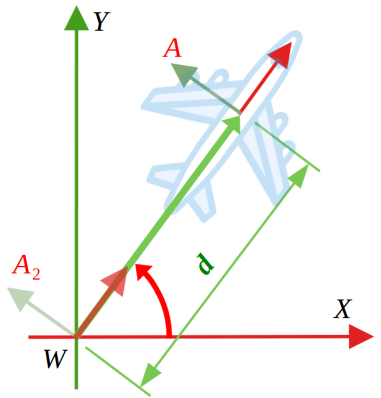
Case 2 : An aircraft fly forward a distance of d with $\mathbf{T}(\mathbf{d}, \mathbf{0})$

Think in Opposite: Align World with Object

Intermediate frame A_2 : parallel to Object, origin at $(0, 0)$:

1. Rotate frame W to A_2: ${}^{W}_{A_2}\mathbf{M} = \mathbf{R}$
2. Translate frame A_2 to A: ${}^{A}_{A_2}\mathbf{M} = \mathbf{T}(\mathbf{d}, \mathbf{0})$
3. Concatenate from left to right

$$ {}^{W}_{A_2}\mathbf{M} = {}^{W}_{A_2}\mathbf{M}\,{}^{A_2}_{A}\mathbf{M} = \mathbf{RT}(\mathbf{d}, \mathbf{0}) $$



RT order

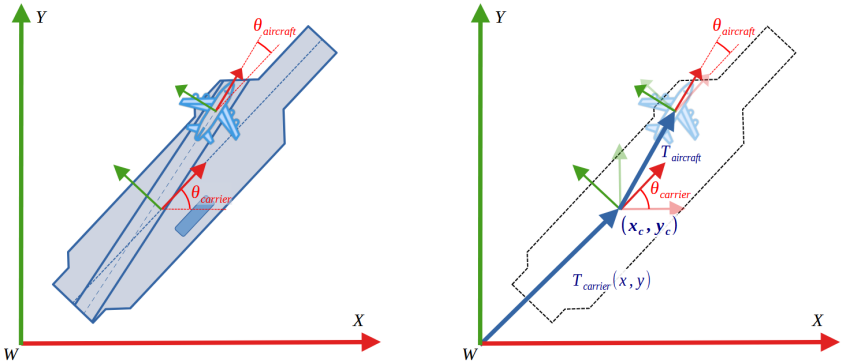
Hierarchical Transforms

Hierarchical Transforms

Add One Level

World $\Rightarrow$ **Carrier** $\Rightarrow$ **Aircraft**

$$ {}^{W}_{A}\mathbf{M} = {}^{W}_{C}\mathbf{M}\,{}^{C}_{A}\mathbf{M} = \left({}^{W}_{C}\mathbf{T}(\mathbf{x},\mathbf{y})\mathbf{R_C}\right)\left({}^{C}_{A}\mathbf{TR_A}\right) $$



Two level transform

Hierarchical Transforms

Question

Problem : Carrier moves forward a distance d

Key : d is with the Carrier frame

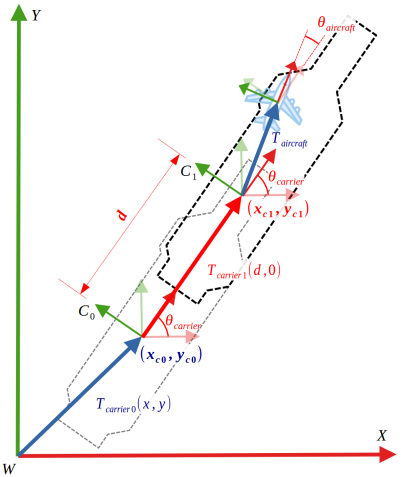
Add a Frame c_1 at time t_1

$$ {}^{C_0}_{C_1}\mathbf{M} = {}^{C_0}_{C_1}\mathbf{T}(d,0,0) = {}^{C_0}_{C_1}\mathbf{T_x}(\mathbf{d}) $$

World $\Rightarrow$ **Carrier@**t_0 $\Rightarrow$ **Carrier@**t_1 $\Rightarrow$ **Aircraft**

Now we add this transform in the middle

$$ {}^{W}_{A}\mathbf{M} = {}^{W}_{C_0}\mathbf{M}\,{}^{C_0}_{C_1}\mathbf{M}\,{}^{C_1}_{A}\mathbf{M} = \left({}^{W}_{C}\mathbf{TR_C}\right){}^{C_0}_{C_1}\mathbf{T_x}(\mathbf{d})\left({}^{C_1}_{A}\mathbf{TR_A}\right) $$



Hierarchical Transforms

Articulated Robot Arms

Joint: rotate in its local frame with θ_i

Link (arm): length $L_i \Rightarrow \mathbf{T}_x(L_i)$ in its joint frame

Problem : $(\theta_i, L_i) \Rightarrow$ gripper position

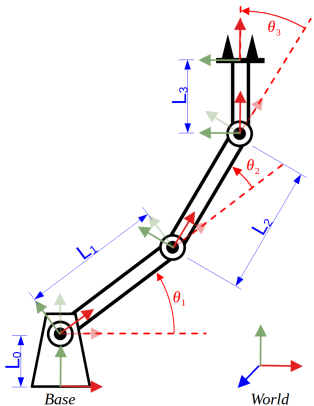
Key: $\mathbf{T}_x(L_i)$ relative to the local joint frame

$$\textbf{Transform at Joint i}: {}^{i}_{i+1}\mathbf{M} = \mathbf{R}(\theta_i)\mathbf{T}_x(L_i)$$

$$\textbf{World} \Rightarrow \textbf{Base} \Rightarrow \textbf{Joint}_1 \Rightarrow \textbf{Joint}_2 \Rightarrow \textbf{Joint}_3 \Rightarrow \textbf{Gripper}$$

$${}^{W}_{G}\mathbf{M} = {}^{W}_{B}\mathbf{M}\,{}^{B}_{1}\mathbf{M}\,{}^{1}_{2}\mathbf{M}\,{}^{2}_{3}\mathbf{M}\,{}^{3}_{G}\mathbf{M}$$

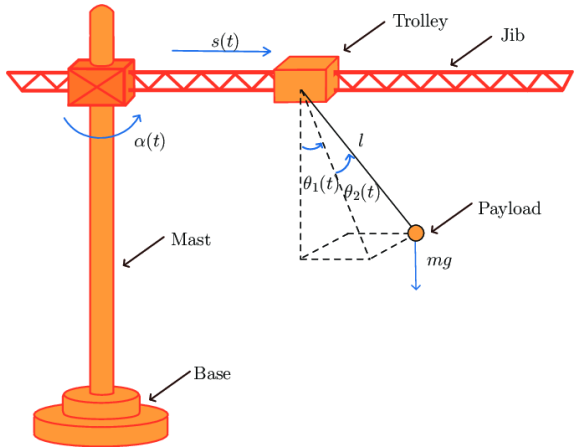
$$= \mathbf{T}_{Base}\mathbf{T}_y(L_0)\Big(\mathbf{R}(\theta_1)\mathbf{T}_x(L_1)\Big)\Big(\mathbf{R}(\theta_2)\mathbf{T}_x(L_2)\Big)\Big(\mathbf{R}(\theta_3)\mathbf{T}_x(L_3)\Big)$$



A three link planar robot arm

Our Turn

How to obtain the **Trolley** position of the tower crane in **World** space ?

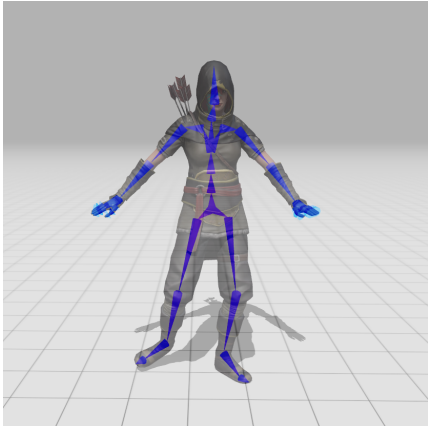


Hierarchical Transforms

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.

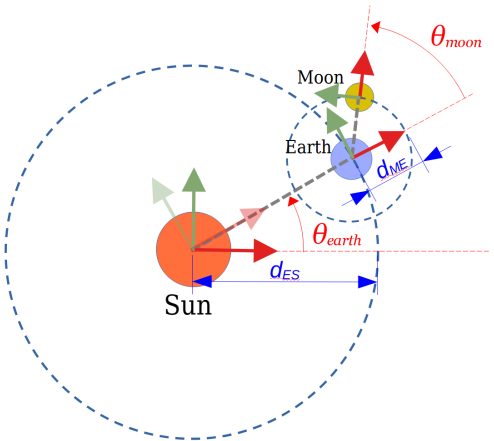


[Basel]

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
   $${}^{S}_{Ec}\mathbf{M} = \mathbf{R}_{Ec\_rev}\mathbf{T}_{ES}$$
2. Earth center frame => Moon center frame:
   $${}^{S}_{Mc}\mathbf{M} = \mathbf{R}_{Ec\_rev}\mathbf{T}_{ES}\mathbf{R}_{Mc\_rev}\mathbf{T}_{ME}$$
3. Moon's surface rotation relative to the centre (actually no rotation as its rotation period equals its revolution period):
   $${}^{S}_{M}\mathbf{M} = \mathbf{R}_{Ec\_rev}\mathbf{T}_{ES}\mathbf{R}_{Mc\_rev}\mathbf{T}_{ME}\mathbf{R}_{M\_rot}$$



Sun-Earth-Moon System

A hierarchical transform tree

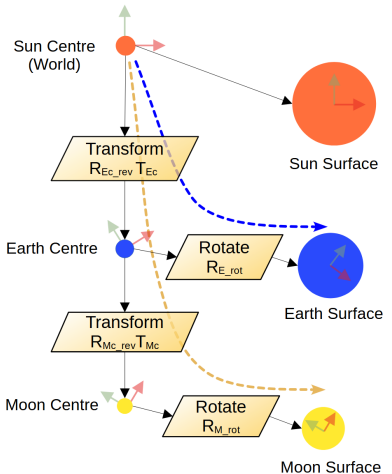
We can organise the transforms in a hierarchical tree structure and write the transforms in a top-down order.

- Earth surface to Sun

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

- Moon surface to Sun

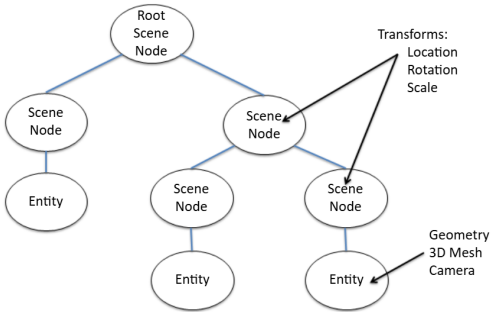
  $${}^{S}_{M}\mathbf{M} = {}^{S}_{Ec}\mathbf{M}\,{}^{Ec}_{Mc}\mathbf{M}\,{}^{Mc}_{M}\mathbf{M} = \mathbf{R}_{E\_rev}\mathbf{T}_{Earth}\mathbf{R}_{M\_rev}\mathbf{T}_{ME}\mathbf{R}_{M\_rot}$$



Hierarchical Modelling - Scene Graph

Scene Graph

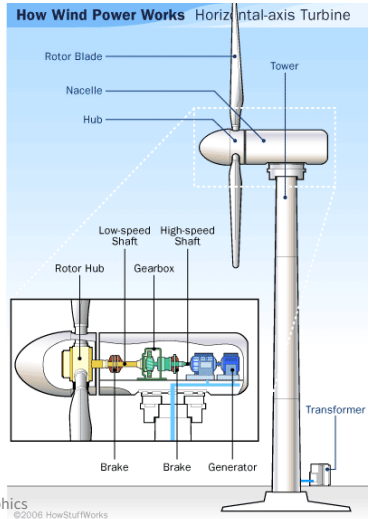
- A hierarchical tree structure of a model or scene, includes
  - Geometry
  - Transform
  - Material, Texture
  - Camera, Light Source ...
  - Group
- Transforms and Geometries can either be nodes, or properties attached to a node
- Used by game engines, modelling software, high-level graphics libs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

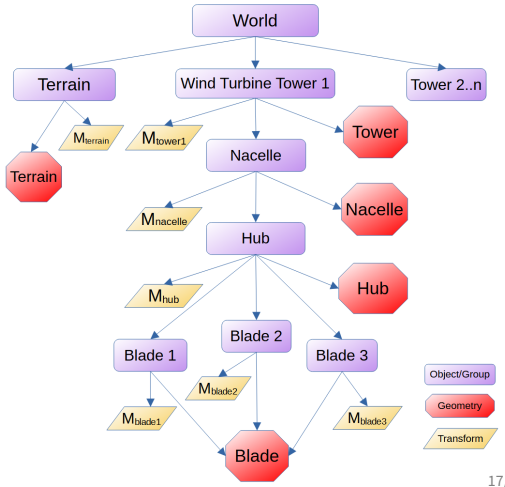


A concept diagram of Ogre3D scenegraph

Traverse a Scene Graph of a Wind Farm

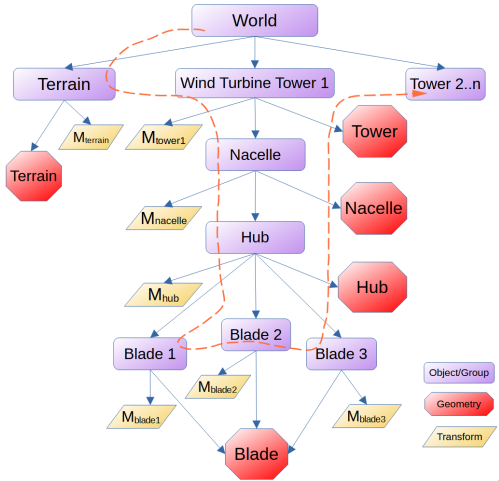
- Drawing a scene is equivalent to traverse its scene graph (tree) .





A Matrix Stack Analogy of Traversing a Scene Graph

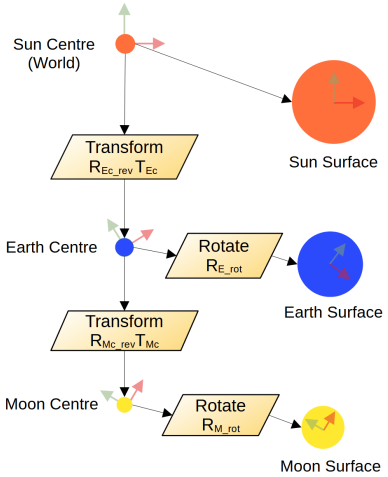
- Recursive traversal of the scene graph applies transforms like an implicit matrix stack.



Object Oriented Design - Inheritance or Composition

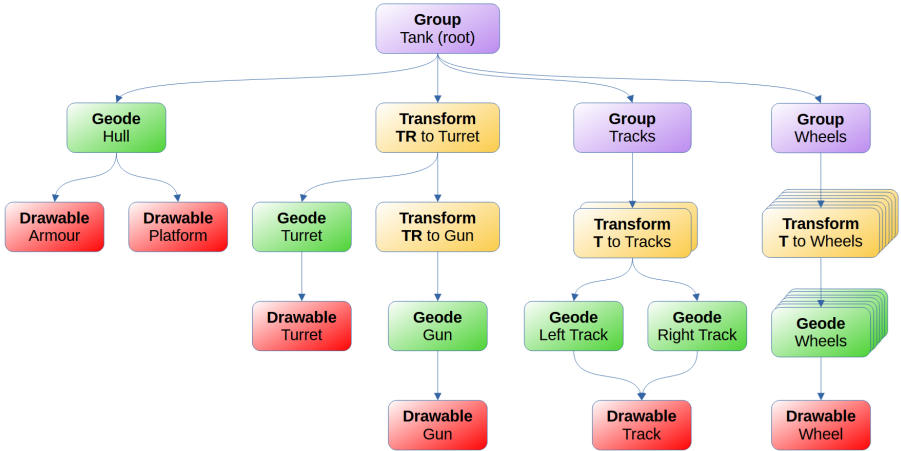
- Inheritance-based
  - Transform is a type of Node
  - A Transform Node can have other Node children, such as shapes
  - OpenSceneGraph, Java3D, OpenInventor ...
  - More used by scene graph libraries
- Composition-based
  - A Node has a Transform
  - e.g. Unreal, Unity, Blender; OGRE3D, Three.js ...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

Ask AI or check APIs for more.



Scene Graph - an OpenSceneGraph example

A Tank model



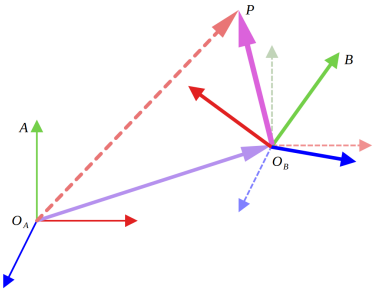
An example scene graph of a tank model in OpenSceneGraph

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 $\mathbf{T}^{-1}(t_x, t_y, t_z) = \mathbf{T}(-t_x, -t_y, -t_z)$.

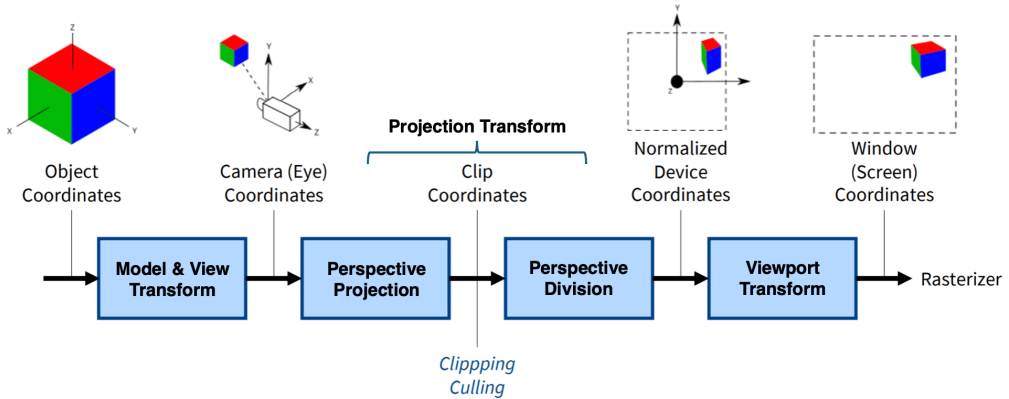
$$
{}^B_A\mathbf{M} = {}^A_B\mathbf{M}^{-1} = ({}^A_B\mathbf{T}\,{}^A_B\mathbf{R})^{-1} = {}^A_B\mathbf{R}^{-1}\,{}^A_B\mathbf{T}^{-1} = {}^A_B\mathbf{R}^T\,{}^A_B\mathbf{T}^{-1}
$$

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

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

Model-View Transformation

Modelling, Viewing and Projection (MVP) chain



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

Viewing Transform

Position and Orient the Camera

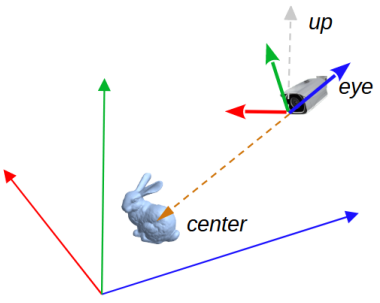
World space $\Rightarrow$ **Camera** space

Camera frame (Right-handed):

- Origin (Eye) in the world space
- Major axes described in world space
  - Can be derived from `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

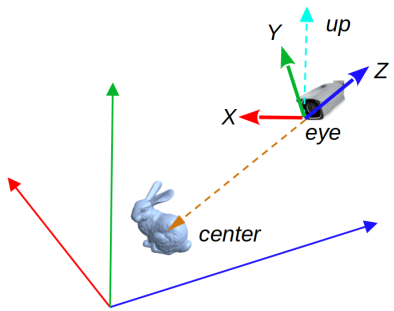
The LookAt function specifies the major axes of the camera space described in the world space

```
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

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



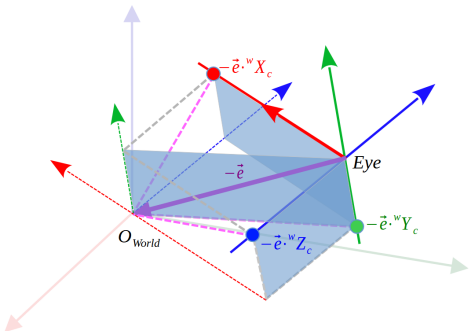
Viewing Transform

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
- Frame Centred Top-down Derivation of the Transform Order
- Hierarchical Transformation
- Hierarchical Modelling : Scene Graph
- Viewing Transformation

Questions?

You are welcome to take the poll !