Viewing and Modeling
Computer Viewing

Three aspects of viewing process:

- Position camera (model-view matrix)
- Selecting a lens (projection matrix)
- Clipping (view volume)
Computer Viewing

Three aspects of viewing process:

- Position camera (model-view matrix)
- Selecting a lens (projection matrix)
- Clipping (view volume)

We discussed projection and NDC last time...
Computer Viewing

Three aspects of viewing process:

- Position camera (model-view matrix)
- Selecting a lens (projection matrix)
- Clipping (view volume)

We’ll discuss object and world space this time!
World and Camera Frames

• Base frame in OpenGL is world frame
• Use view matrix to change world representation to camera representation
• Fixed pipeline OpenGL treated model and view matrices as single (model-view) matrix
Model (Object) Coordinates

- Consider this bunny model...
- Each tri has relative position to the other tris
- Must define **space** in which all tris exist
World Coordinates

Now consider this scene...

Aside: Motunui Island (Disney’s Moana) has over 15 billion primitives (90 million unique quads and 5 million curves). A still frame of the base scene is 44.8GB + 23.6GB of animation data.
Model Matrix

- Unique to each model
- Used to position the model and its tris in world coordinates
OpenGL Camera

Initial representation:

- Object and camera frames are the same (model-view matrix is identity)
- Camera located at origin
- Camera points in negative Z direction
- Default view volume is centered at origin with side lengths of 2 (NDC)
Changing the View

How to change visible objects?
Moving Camera Frame

Move the camera in the positive Z direction (translate camera frame)
Move objects in the negative Z direction (translate world frame)

…Which is better?
Moving Camera Frame

Move the camera in the positive Z direction (translate camera frame)
Move objects in the negative Z direction (translate world frame)

...they’re equivalent!
View Matrix

• All vertices defined relative to the camera
• Therefore world moves relative to camera

Consider:

```
glm::mat4 ViewMatrix =
  glm::translate(0.f, 0.f, -14.f);
```

What is this doing?
Translation in View Space

\[
\begin{bmatrix}
1 & 0 & 0 & x \\
0 & 1 & 0 & y \\
0 & 0 & 1 & z \\
0 & 0 & 0 & 0
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & -14 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Point at (0, 0, 0) moves to (0, 0, -14)
Remember!

In graphics, everything is relative

SO WHAT I TOLD YOU WAS TRUE...

FROM A CERTAIN POINT OF VIEW
Remember!

In graphics, EVERYTHING IS RELATIVE
General Camera Motion

Position camera using translations and rotations
- Move camera to origin (T)
- Rotate camera (R)
- $MV = RT$
A Better Viewing Matrix

“Look at” Transform:
Construct an affine 4x4 matrix to map world space into camera space

What do we need to know about the camera’s placement in the world to construct this?
glm::lookAt

Defines:
• Camera position
• Camera target
• Camera up

Returns:
• View matrix
lookAt Algorithm

In order to define view coordinate system:

• Z axis (forward vector) = normalize(at - eye)
• X axis (left vector) = normalize(up x Z)
• Y axis (up vector) = normalize(X x Z)

What happens if Z or up are zero length?
What happens if Z and up are coincident?
Why Recompute Up?

The given \textbf{up} vector is not necessarily perpendicular to \textbf{forward} vector.

Actual \textbf{up} vector will be orthogonal to \textbf{left} and \textbf{forward} vectors.
OpenGL’s Internal lookAt Matrix

\[
\begin{bmatrix}
X_x & X_y & X_z & 0 \\
Y_x & Y_y & Y_z & 0 \\
-Z_x & -Z_y & -Z_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Note: Z (i.e. look direction) is made negative to negate OpenGL’s default of looking down the -Z axis.
Combining Model-View-Projection

$$\text{glm::mat4 MVPmatrix = projection*view*model;}$$

\[
\begin{bmatrix}
1.250 & 0 & 0 & 0 \\
0 & 1.667 & 0 & 0 \\
0 & 0 & -1.333 & -10.667 \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & -14 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
0.9107 & -0.2440 & 0.3333 & 0 \\
0.3333 & 0.9107 & -0.2440 & 0 \\
-0.2440 & 0.3333 & 0.9107 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

projection  view  model

Remember: matrix multiplication is associative but not commutative:

$$A(BC) = (AB)C \quad ABC \neq CBA$$
A Note About Matrices

• OpenGL uses column-major notation (DirectX uses row-major notation)
  • Note that layout in memory is separate from this!

• OpenGL uses post-multiplication (and yes, DirectX uses pre-multiplication)

• OpenGL transforms are therefore multiplied in “reverse” order of application:
  e.g. \( p' = P \times V \times M \times p \)
OpenGL Tutorial

Look through:

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

https://learnopengl.com/Getting-started/Coordinate-Systems
References