Transpose Matrix

Rows become columns and columns become rows

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n1} & \ldots & a_{nm} \end{bmatrix} \quad A^T = \begin{bmatrix} a_{11} & a_{21} & \ldots & a_{m1} \\ a_{12} & a_{22} & \ldots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \ldots & a_{mn} \end{bmatrix}$$
Matrices A and B have these dimensions:

\[
\begin{bmatrix}
  r \\
  c
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
  s \\
  d
\end{bmatrix}
\]

Matrices A and B can be multiplied if: \( c = s \)

The resulting matrix will have the dimensions: \( r \times d \)
Computation: \( A \times B = C \)

\[
A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad [2 \times 2]
\]

\[
B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} \quad [2 \times 3]
\]

\[
C = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & a_{21}b_{13} + a_{22}b_{23} \end{bmatrix} \quad [2 \times 3]
\]
Computation: $A \times B = C$

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 2 \end{bmatrix}$$

$A$ and $B$ can be multiplied

$$C = \begin{bmatrix} 2*1+3*1=5 & 2*1+3*0=2 & 2*1+3*2=8 \\ 1*1+1*1=2 & 1*1+1*0=1 & 1*1+1*2=3 \\ 1*1+0*1=1 & 1*1+0*0=1 & 1*1+0*2=1 \end{bmatrix} = \begin{bmatrix} 5 & 2 & 8 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$[3 \times 3]$$
Computation: $B \times A = D$

\[
A = \begin{bmatrix} 2 & 3 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 2 \end{bmatrix}
\]

\[
D = \begin{bmatrix} 1 \times 2 + 1 \times 1 + 1 \times 1 = 4 \\ 1 \times 3 + 1 \times 1 + 1 \times 0 = 4 \\ 1 \times 2 + 0 \times 1 + 2 \times 1 = 4 \\ 1 \times 3 + 0 \times 1 + 2 \times 0 = 3 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 4 & 3 \end{bmatrix}
\]

$D \neq C$
How to ...

- Establish your position in the scene
- Position objects within the scene
- Scale objects
- Establish a perspective transformation
- Moving around in OpenGL using a camera
## Terminology

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewing</td>
<td>Specifies the location of the viewer or camera</td>
</tr>
<tr>
<td>Modeling</td>
<td>Moves objects around the scene</td>
</tr>
<tr>
<td>ModelView</td>
<td>Describes the duality of viewing and modeling transformations</td>
</tr>
<tr>
<td>Projection</td>
<td>Clips and sizes the viewing volume</td>
</tr>
<tr>
<td>Viewport</td>
<td>Scales the final output to the window</td>
</tr>
</tbody>
</table>
Eye Coordinates

- Viewpoint of the observer
- They are a **virtual** fixed coordinate system
- Used as a common frame of reference
Viewing Transformations

- Like placing and pointing the camera at the scene
- To be specified before any other transformation
- By default, the point of observation is at the origin, looking down the negative Z-axis.

What about objects drawn with positive Z values?
Modeling Transformation

- Manipulate the model and the particular objects within it
- **Move** objects into place, **rotate** them, and **scale** them
- These are the most common modeling transformations
Final appearance of an object depends greatly on the order of transformations.
ModelView Duality

- In fact, viewing and modeling transformations are the same
- Induce the same effects on the final appearance of the scene
- The distinctions is made as a convenience for the programmer
glm::ortho(left, right, bottom, top, -near, -far)
Glm::frustum(left, right, bottom, top, near, far)

glm::perspective(GLdouble fov, GLdouble aspectRatio, GLdouble zNear, GLdouble zFar)
Projection Transformations

- Applied after the modelview transformation
- Defines the **viewing volume** and the **clipping planes**
- Specifies how a finished scene is projected to the screen
Orthographic Vs. Perspective

- **In an Orthographic projection:**
  - objects are mapped directly on the 2D screen using parallel lines
  - No matter how far away something is

- **In Perspective projection:**
  - Scenes are more realistic
  - Distant objects appear smaller than nearby objects of the same size
  - Parallel lines in 3D space do not always appear parallel
OpenGL projection model uses *eye coordinates*

- the “eye” is located at the origin
- looking down the -z axis

Projection matrices use a six-plane model:

- near (image) plane and far (infinite) plane
- both are distances from the eye (positive values)
- enclosing planes
- top & bottom, left & right
The Pipeline - Recall

Original vertex data $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix}$ → Modelview matrix $\begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}$ → Transformed eye coordinates → Projection matrix $\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix}$ → Clip coordinates → Perspective division $\begin{bmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{bmatrix}$ → Normalized device coordinates

Viewport transformation $\begin{bmatrix} \end{bmatrix}$ → Window coordinates
3D Transformations

- A vertex is transformed by 4×4 matrices
- all affine operations are matrix multiplications
- all matrices are stored column-major in OpenGL
- this is opposite of what “C” programmers expect

- matrices are always post-multiplied
- product of matrix and vector is \( \mathbf{M} \mathbf{\vec{v}} \)

\[
\mathbf{M} = \begin{bmatrix}
  m_0 & m_4 & m_8 & m_{12} \\
  m_1 & m_5 & m_9 & m_{13} \\
  m_2 & m_6 & m_{10} & m_{14} \\
  m_3 & m_7 & m_{11} & m_{15}
\end{bmatrix}
\]
Camera Management

Glm::lookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz)
Current Matrix is part of the OpenGL state
Options are:
- MODELVIEW – changes, rotate, translate the model
- PROJECTION – changes, translate, rotate the camera
To specify which one is the “Current Matrix”:

```c
glMatrixMode(...)
```

Subsequent matrix operations affect the specified matrix
One of these matrices could be changed in a given time
The default matrix is the identity matrix:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

We can change the current matrix to the identity matrix any time use:

\[\text{glm::mat4}(1);\]
Translation

\texttt{mat4 glm::translate(mat4 m, vec3 v)}

This function takes as parameters the amount to translate along the x, y, and z directions

\[
\begin{pmatrix}
1 & 0 & 0 & dx \\
0 & 1 & 0 & dy \\
0 & 0 & 1 & dz \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\texttt{glTranslatef(dx, dy, dz)}
Translation

Move the origin to a new location

\[ T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \]
Rotation

\texttt{mat4 glm::rotate(mat4 m, float angle, vec3 v)}

This function performs a rotation around the vector specified by the x, y, and z arguments. The \textit{angle} of rotation is in the counterclockwise direction.

A rotation around an \textbf{arbitrary} axis could be done.

\begin{align*}
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \alpha & -\sin \alpha & 0 \\
0 & \sin \alpha & \cos \alpha & 0 \\
0 & 0 & 0 & 1
\end{pmatrix} & \begin{pmatrix}
\cos \alpha & 0 & \sin \alpha & 0 \\
0 & 1 & 0 & 0 \\
-\sin \alpha & 0 & \cos \alpha & 0 \\
0 & 0 & 0 & 1
\end{pmatrix} & \begin{pmatrix}
\cos \alpha & -\sin \alpha & 0 & 0 \\
\sin \alpha & \cos \alpha & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\end{align*}

\texttt{glRotate(\alpha,1,0,0) - X} \quad \texttt{glRotate(\alpha,0,1,0) - Y} \quad \texttt{glRotate(\alpha,0,0,1) - Z}
Rotation

Rotate coordinate system about an axis in space

Note, there’s a translation applied here to make things easier to see.
Scaling

\texttt{mat4 glm::scale(mat4 m, vec3 v)}

Multiplies the x, y and z values by the scaling factors specified. Scaling does not have to be uniform.
Scale

Stretch, mirror or decimate a coordinate direction

\[
S(s_x, s_y, s_z) = \begin{pmatrix}
    s_x & 0 & 0 & 0 \\
    0 & s_y & 0 & 0 \\
    0 & 0 & s_z & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]

Note, there’s a translation applied here to make things easier to see.
The idea of Identity Matrix

- Matrix operations are cumulative
- Erroneous artifacts may occur to the scene
- You reset the origin by loading the modelview matrix with the identity matrix

```c
translatef(vec3(0.0f, 10.0f, 0.0f));

glutSolidSphere(1.0f, 32, 32);

translatef(vec3(10.0f, 0.0f, 0.0f));

glutSolidSphere(1.0f)
```
The idea of Identity Matrix – (cont’)

glMatrixMode(GL_MODELVIEW)
glLoadIdentity();

glTranslatef(0.0f, 10.0f, 0.0f);
glutSolidSphere(1.0f, 32, 32);

glLoadIdentity();

glTranslatef(10.0f, 0.0f, 0.0f);
glutSolidSphere(1.0f)
Resetting the modelview matrix to identity before placing the every object is not always desirable.

To save the current transformation state,

OpenGL maintains a matrix stack for both the modelview and projection matrices.

- `glPushMatrix()` – *push* the current state matrix to the stack.
- `glPopMatrix()` – *pop* the top matrix out of the stack, and *replace* the current state matrix with it.
The idea of matrix stack – (cont’)

```c
render_car()
{
    glTranslatef(c_x,c_y,c_z);
    render_car_body();
    glPushMatrix();
    glTranslatef(f_x,f_y,f_z);
    glPushMatrix();
    glTranslatef(l_x,l_y,l_z);
    render_wheel();
    glPopMatrix();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(r_x,r_y,r_z);
    render_wheel();
    glPopMatrix();
    glPopMatrix();
}
```

See Code Example 4
Euler Angles

- Euler Angles, definition:
  - A set of three angles used to describe the orientation of a reference frame in 3D space

- Draws on the following observation:
  - You can **align (superimpose) the global reference frame to any arbitrary reference frame** through a sequence of THREE rotation operations

- In picture: start from blue RF, end in the red RF after three rotations

- The sequence of three rotations that we’ll consider is
  - About axes Z then X then Z again (called the 3-1-3 sequence)
  - We’ll denote the three Euler Angles by $\phi$, $\theta$, and $\psi$, respectively
Euler matrices

\[ A_1 = \begin{bmatrix}
\cos \phi & -\sin \phi & 0 \\
\sin \phi & \cos \phi & 0 \\
0 & 0 & 1
\end{bmatrix} \]

Rotate around z axis

\[ A_2 = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \theta & -\sin \theta \\
0 & \sin \theta & \cos \theta
\end{bmatrix} \]

Rotate around new x axis

\[ A_3 = \begin{bmatrix}
\cos \psi & -\sin \psi & 0 \\
\sin \psi & \cos \psi & 0 \\
0 & 0 & 1
\end{bmatrix} \]

Rotate around new z axis.
Expressing $A$ using Euler Angles

~ Putting it All Together ~

- Using the expression of the matrices $A_1$, $A_2$, $A_3$, one gets for the expression of the orientation matrix $A$

$$A = A_1 A_2 A_3 = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Carry out multiplications to get

$$A = \begin{bmatrix} \cos \psi \cos \phi - \cos \theta \sin \phi \sin \psi & -\sin \psi \cos \phi - \cos \theta \sin \phi \cos \psi & \sin \theta \sin \phi \\ \cos \psi \sin \phi + \cos \theta \cos \phi \sin \psi & -\sin \psi \sin \phi + \cos \theta \cos \phi \cos \psi & -\sin \theta \cos \phi \\ \sin \theta \sin \psi & \sin \theta \cos \psi & \cos \theta \end{bmatrix}$$
Building a scene with Multiple Objects

- The camera is the first object to be located in (0,0,0)
- You can draw each object in scene coordinate but then you have to calculate rotation and position by yourself.
- The better option is to draw each object when its center is on the origin and afterward use rotation and translation to move it to its position in the scene.
- Remember: Rotation of OpenGL is always around the origin!
Tips for scene building

- Remember that rotations and translations are accumulated
- Rotating change your axis system
- Build for each object (include scene) its own “modelview” matrix
- Remember that the order of transformation affects the result.