Review of Thursday

- Vectors and points are two/three-dimensional objects
- Operations on vectors (dot product, cross product, …)
- Matrices are linear transformations of vectors

\[
\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} xa + yb \\ xc + yd \end{pmatrix}
\]

- Objects are transformed by applying transformation matrix to its vertices (coordinate transformation)
Review of Thursday

- Objects are transformed by applying transformation matrix to its vertices (coordinate transformation)

\[
\begin{bmatrix}
2 & 0 \\
0 & 3 \\
\end{bmatrix} \cdot 
\begin{bmatrix}
1 \\
1 \\
\end{bmatrix} = 
\begin{bmatrix}
2 \\
3 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 0 \\
0 & 3 \\
\end{bmatrix} \cdot 
\begin{bmatrix}
3 \\
3 \\
\end{bmatrix} = 
\begin{bmatrix}
6 \\
9 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 0 \\
0 & 3 \\
\end{bmatrix} \cdot 
\begin{bmatrix}
4 \\
1 \\
\end{bmatrix} = 
\begin{bmatrix}
8 \\
3 \\
\end{bmatrix}
\]

\[\text{det}(M) = 6\]
A Brief Review of Linear Algebra

• Rotation

\[
\begin{pmatrix}
\cos \alpha & -\sin \alpha \\
\sin \alpha & \cos \alpha
\end{pmatrix}
\]

counterclockwise (positive direction of rotation)

Determinant:

\[
\cos \alpha \cdot \cos \alpha - \sin \alpha \cdot (-\sin \alpha) = 1
\]

Inverse

\[
M^{-1} = M^T
\]
A Brief Review of Linear Algebra

- Constructing a rotation matrix

\[
M(1, 0)^T = (\cos \alpha, \sin \alpha)^T \\
M(0, 1)^T = (-\sin \alpha, \cos \alpha)^T
\]

\[
(a, b)^T = a(1, 0)^T + b(0, 1)^T
\]

\[
M(a, b)^T = aM(1, 0)^T + bM(0, 1)^T \quad \text{(linearity of } M)\]

\[
M(a, b)^T = (a \cos \alpha - b \sin \alpha, a \sin \alpha + b \cos \alpha)^T
\]

\[
\begin{pmatrix}
\cos \alpha & -\sin \alpha \\
\sin \alpha & \cos \alpha
\end{pmatrix}
\]

columns correspond to axes of new coordinate frame
A Brief Review of Linear Algebra

- Shear transformation (shearing)

\[
\begin{pmatrix}
1 & 0 \\
0.5 & 1
\end{pmatrix}
\]

displacement along direction proportional to (signed) distance to a line

- Reflection

\[
\begin{pmatrix}
-1 & 0 \\
0 & 1
\end{pmatrix}
\]
A Brief Review of Linear Algebra

- Applying multiple transformations in sequence

\[ M_{scal} \cdot M_{rot} \cdot v = v' \]

\[ (M_{scal} \cdot M_{rot}) \cdot v = v' \]

Order of operations is important!
Notes on Matrices in OpenGL

- OpenGL <3.0 includes matrix operations and stack

```c
glMatrixMode(...)
glLoadIdentity(...)  
glMultMatrix(…)      
glPushMatrix()       //push current matrix to top of stack  
glPopMatrix()        //take matrix from top of stack  
...                  
```

Assignment 2: You may use glLoadIdentity, glMultMatrix or pass the matrices to the vertex shader as *uniform* variables (do not use glTranslate, glRotate).

- Matrix operations and stack deprecated in OpenGL 3.0+
  - Implement own transformation matrices
  - Pass to vertex shader (OpenGL/GLSL: *uniform* variables)
Notes on Matrices in OpenGL

• OpenGL <3.0 vertex shader example

```c
void main(void)
{
    gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex;
}
```

• OpenGL 3.0+ vertex shader example

```c
attribute vec3 in_Position;  //vertex attribute is part of VBO/vertex array
uniform mat4 m_modelview;    //custom uniform variable for modelview matrix
uniform mat4 m_projection;   //custom uniform variable for projection matrix

void main(void)
{
    gl_Position = m_projection * m_modelview * vec4(in_Position,1.0);
}
```
Homogeneous Coordinates

• Problems so far:
  • No translation
  • No explicit notion of local and global coordinates
  • No distinction between points and vectors

Homogeneous coordinates solve these problems.
Homogeneous Coordinates

- Distinguishing points and vectors

\[ v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 \]

In world frame **vectors** are defined by a local coordinate frame.

\[ P = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + P_0 \]

In world frame **points** are defined by a local coordinate frame **and origin**.
Homogeneous Coordinates

- Distinguishing points and vectors

Local coordinate system as rows in matrix

\[
\begin{pmatrix}
v_1 \\
v_2 \\
v_3 \\
P_0
\end{pmatrix}
\]

\[
v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3
\]

\[
v = (\alpha_1, \alpha_2, \alpha_3, 0) \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix}
\]

\[
P = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + P_0
\]

\[
P = (\alpha_1, \alpha_2, \alpha_3, 1) \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix}
\]
Transformation in Homogeneous Coordinates

- Affine transformation now represented by 4x4 matrix

\[
A = \begin{pmatrix}
\alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\
\alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\
\alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

A transforms points \( p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \) and vectors \( v = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} \)
Translation

\[ P' = P + v \]

\[ P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad v = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} x + v_x \\ y + v_y \\ z + v_z \\ 1 \end{pmatrix} \]

\[ P' = \begin{pmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot P \]

translation matrix \( T \)
Transformation in Homogeneous Coordinates

- Example coordinate transformation (scaling and translation)

\[
v_{world} = T(p)S(0.5, 0.5, 0.5) \cdot v_{local} = \begin{pmatrix} 0.5 & 0 & 0 & p_1 \\ 0 & 0.5 & 0 & p_2 \\ 0 & 0 & 0.5 & p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot v_{local}
\]
Rotation in 3D

\[
R_z(\alpha) = \begin{pmatrix}
\cos(\alpha) & -\sin(\alpha) & 0 & 0 \\
\sin(\alpha) & \cos(\alpha) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

upper left sub-matrix identical to 2D matrix

\[
R_x(\alpha) = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(\alpha) & -\sin(\alpha) & 0 \\
0 & \sin(\alpha) & \cos(\alpha) & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
inversion:

\[
R(\alpha)^{-1} = R(-\alpha)
\]

\[
\cos(-\alpha) = \cos(\alpha)
\]

\[
\sin(-\alpha) = -\sin(\alpha)
\]

\[
R(\alpha)^{-1} = R(\alpha)^T
\]
Rotation: Fixed Point

- Rotation always performed around origin
- Changing the fixed point requires translations:

\[
\begin{align*}
v_{\text{world}} &= T(p)R(z)T(-p) \cdot v_{\text{local}} \\
&= \begin{pmatrix}
\cos(\alpha) & -\sin(\alpha) & 0 & p_1 - p_1 \cos(\alpha) + p_2 \sin(\alpha) \\
\sin(\alpha) & \cos(\alpha) & 0 & p_2 - p_1 \sin(\alpha) - p_2 \cos(\alpha) \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix} \cdot v_{\text{local}}
\end{align*}
\]
Arbitrary Rotation

• Arbitrary rotations can be expressed as successive rotations around coordinate axes

\[ R = R_x(\alpha)R_y(\beta)R_z(\gamma) \]

• Decomposition is not unique

\[ R_z(90^\circ)R_y(90^\circ) = R_y(90^\circ)R_x(-90^\circ) \]
Rotation Around Arbitrary Axis

- Rotation around arbitrary axis \( v \) by \( \alpha \):
  - Move fixed point (e.g., center of object) to origin: \( T(-p) \)
  - Rotate axis of rotation, such that it aligns with z-axis: \( R_y(\beta_2)R_x(\beta_1) \)
  - Rotate around z-axis by \( \alpha \): \( R_z(\alpha) \)
  - Undo alignment rotation: \( R_x(-\beta_1)R_y(-\beta_2) \)
  - Undo translation: \( T(p) \)

\[
A = T(p) R_x(-\beta_1) R_y(-\beta_2) R_z(\alpha) R_y(\beta_2) R_x(\beta_1) T(-p)
\]

‘Only’ need to find \( \beta_1, \beta_2 \)
Rotation Around Arbitrary Axis

A sequence of two rotations can align any vector with the z-axis.
Rotation Around Arbitrary Axis

• $\beta_1, \beta_2$ do not need to be computed explicitly

$$d = \sqrt{v_y^2 + v_z^2}$$

$$R_x(\beta_1) = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & v_z/d & -v_y/d & 0 \\
0 & v_y/d & v_z/d & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}$$

$$R_y(\beta_2) = \begin{pmatrix}
d & 0 & -v_x & 0 \\
0 & 1 & 0 & 0 \\
v_x & 0 & d & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}$$
Homogeneous Coordinates

• How to interpret a transformation matrix

\[
\mathbf{v}_{\text{world}} = \begin{pmatrix}
0.5 & 0 & 0 & p_1 \\
0 & 0.5 & 0 & p_2 \\
0 & 0 & 0.5 & p_3 \\
0 & 0 & 0 & 1
\end{pmatrix} \cdot \mathbf{v}_{\text{local}}
\]

Local coordinate frame (source) expressed in global coordinates (target).

Inverse matrix reverses transformation.
Review of Tuesday

- Transformation matrix in homogeneous coordinates

\[
v_{world} = \begin{pmatrix} 0.5 & 0 & 0 & p_1 \\ 0 & 0.5 & 0 & p_2 \\ 0 & 0 & 0.5 & p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot v_{local}
\]

Local coordinate frame (source) expressed in global coordinates (target).

Transforms points \( p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \) and vectors \( v = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} \)
Review of Tuesday

• Origin of current coordinate system is fixed point

Identical transformation matrix has different effects based on relative position of the object and coordinate origin.
• More complex operations can be performed by concatenating transformations. For example:

\[ v_{world} = T(p)R_z(\alpha)T(-p) \cdot v_{local} \]

Rotation around arbitrary point (2D)

\[ A = T(p)R_x(-\beta_1)R_y(-\beta_2)R_z(\alpha)R_y(\beta_2)R_x(\beta_1)T(-p) \]

Rotation around arbitrary axis (3D)
Sequences of Transformations

- How do we interpret a sequence of transformation matrices?
- We can read transformations from right to left or from left to right

- Reading from right to left: Interpret operations as being performed in (fixed) world coordinate system.

- Reading from left to right: Interpret operations as being performed in (dynamic) object coordinate system.
The vertex processor transforms vertices by applying transformation of *current transformation matrix*

```plaintext
setTransformationMatrix(matrix1)
renderObject(object1)  //matrix1 is active

setTransformationMatrix(matrix2)
renderObject(object2)  //matrix2 is active
```

Pseudo code

Caution when using OpenGL matrix operations (OpenGL <3.0): OpenGL does matrix **post-multiplication** as opposed to **pre-multiplication**.
Coordinate Transformations - Summary

• Homogeneous coordinates allow us to
  • Incorporate local coordinate system origins (translation)
  • Distinguish points and vectors
  • Do a full transform between coordinate systems

• Important notions
  • Points are different from vectors (cf. vertices and normals)
  • Order of transformations matters
  • Rotation and translation are rigid-body transformations
  • Programming: be aware of row-major vs. column major matrices
Chapter 4 - Summary

- Object geometry (vertices and connectivity) is passed to the graphics pipeline together with transformation operations.

- Transformation operations are represented by matrices.

- Objects are transformed by applying the transformation to each of its vertices (vertex processor does this in parallel).

- The transformed objects need to be projected into a 2D coordinate system and mapped to the screen (“Where do objects end up on screen?” - next chapter).