Today’s class

- Geometric objects and transformations
Vector operations

- Review of vector operations needed for working in computer graphics
  - adding two vectors
  - subtracting two vectors
  - scaling a vector
  - magnitude of a vector
  - normalizing a vector
Linear combination of vectors

- Convex combinations (coefficients of linear combination are all non-negative and add up to 1)
- Important convex combination: \( p(t) = a(1-t) + b(t) \) where \( a \) and \( b \) are vectors and \( 0 \leq t \leq 1 \)
- If \( a \) and \( b \) are points, then \( p(t) = a(1-t) + b(t) \) is said to **interpolate** the points \( a \) and \( b \) \( (0 \leq t \leq 1) \)
Convex sets and hulls

- A **convex set** of points is a collection of points in which the line connecting any pair in the set lies entirely in the set.

- **Examples:**
  - circle
  - rectangle

- A **convex hull** is the smallest convex set that contains a specified collection of points.
Dot products

- $\cos \theta = \mathbf{u}_a \cdot \mathbf{u}_b$
- Vectors are orthogonal if dot product is 0
- Vectors are less than 90° apart if dot product is positive
- Vectors are more than 90° apart if dot product is negative
- Perpendicular projection of $\mathbf{a}$ onto $\mathbf{b}$ is $(\mathbf{a} \cdot \mathbf{u}_b) \mathbf{u}_b$
- Reflected ray is $\mathbf{a} - 2(\mathbf{a} \cdot \mathbf{u}_n) \mathbf{u}_n$ where $\mathbf{u}_n$ is a unit normal vector to the surface
Cross products

\[ \mathbf{u} = (u_x, u_y, u_z), \quad \mathbf{v} = (v_x, v_y, v_z) \]

\[ \mathbf{u} \times \mathbf{v} = \begin{vmatrix} x & y & z \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} \]

- \( \mathbf{u} \times \mathbf{v} \) is perpendicular to both \( \mathbf{u} \) and \( \mathbf{v} \)
- Orientation follows the right hand rule
- \( |\mathbf{u} \times \mathbf{v}| = |\mathbf{u}| |\mathbf{v}| \sin \theta \)
Polygons

- A polygon is defined by a set of vertices and edges.
- An interior angle of a polygon is an angle formed by two adjacent edges that is inside the boundary.
- A convex polygon has every interior angle < 180°.
- A concave polygon is a polygon that is not convex.
Identifying concave polygons

- Concave polygons will have at least one interior angle > 180°
- The sin of this angle will be negative
- Thus, the “magnitude” of the cross product of the two vectors (edges) forming this angle will be negative
- This will also manifest itself by some z components of the cross product being positive and others being negative
Splitting concave polygons

- Concave polygons present problems for computer graphics
- Splitting them up will get us convex polygons
- Define edge vector $\mathbf{E}_k$ as the vector $\mathbf{V}_{k+1} - \mathbf{V}_k$, where the $\mathbf{V}$'s are adjacent vertices
- Calculate the edge cross products in order counter-clockwise around the polygon
- If any $z$ component of a cross product is negative the polygon is concave and can be split along the line from the first edge in the cross product
Triangularization

- A convex polygon can easily be triangularized (sometimes also referred to as tessellated)
- Define three consecutive vertices of the polygon as a triangle
- Remove the middle vertex from the polygon’s vertex list
- Repeat with the modified list until only three vertices remain, forming the last triangle
Is a point inside a polygon?

- **Even-odd test:**
  - construct a line from point to a point known to be outside polygon
  - count # of intersections of line with boundary
  - if odd, point is inside
  - if crossing occurs at a vertex
    - look at endpoints of the edges that meet
    - if both endpoints are on same side of line segment then the vertex counts as an even number of crossings, otherwise it counts as an odd number
N-gons

- An n-gon is a regular polygon of n sides
Rosettes

- Rosettes are n-gons where each vertex has been joined to each other vertex.
A cube

- Consider a cube
- Pair off and come up with:
  - A set of vertices for the cube
  - A sequence of OpenGL statements to draw the cube
  - How many function calls are you making?
Vertex arrays

- Allow you to specify geometry with much fewer function calls
- For example, the cube can be drawn in just one function call
- Create an array of vertex data and use `glVertexPointer()` to refer to it
- Need to enable this with `glEnableClientState(GL_VERTEX_ARRAY);`
**glVertexPointer()**

- **Four parameters:**
  - **Size**: number of coordinates per vertex; must be 2, 3, or 4
  - **Type**: the data type of each coordinate in the array
  - **Stride**: the byte offset between consecutive vertices (frequently 0)
  - **Pointer**: a pointer to the first coordinate of the first vertex in the array
Index array

- Need a second array with indices into the vertex array to specify which vertices make up each face of your geometry
- `glDrawElements()` is then used to draw the geometry
glDrawElements() 

- Four parameters: 
  - *Mode*: specified what kind of geometric primitives to render 
  - *Count*: number of vertices in the index array to be rendered 
  - *Type*: the type of the values in the index array; must be one of `GL_UNSIGNED_BYTE`, `GL_UNSIGNED_SHORT`, or `GL_UNSIGNED_INT` 
  - *Indices*: a pointer to the array of indices into the vertex array
Example program

- `rotatingCube.c` demonstrates vertex arrays and introduces the idle callback function
A **transformation matrix** is a matrix that transforms a point to another point.

In 2-dimensions, this has the following algebraic form:

\[
\begin{bmatrix}
  a_{11} & a_{12} \\
  a_{21} & a_{22}
\end{bmatrix}
\begin{bmatrix}
  p_x \\
  p_y
\end{bmatrix}
= 
\begin{bmatrix}
  a_{11}p_x + a_{12}p_y \\
  a_{21}p_x + a_{22}p_y
\end{bmatrix}
\]
Uniform scaling

- Given a scale factor $s > 0$, a matrix which produces a uniform scaling is
  \[
  \begin{bmatrix}
  s & 0 \\
  0 & s \\
  \end{bmatrix}
  \]

- Example:
  \[
  \begin{bmatrix}
  s & 0 \\
  0 & s \\
  \end{bmatrix}
  \begin{bmatrix}
  p_x \\
  p_y \\
  \end{bmatrix} = \begin{bmatrix}
  sp_x \\
  sp_y \\
  \end{bmatrix}
  \]
Non-uniform scaling

- Given scale factors $s_x > 0$ and $s_y > 0$, a matrix which produces non-uniform scaling is

$$\begin{bmatrix}
  s_x & 0 \\
  0 & s_y
\end{bmatrix}$$

- Example:

$$\begin{bmatrix}
  s_x & 0 \\
  0 & s_y
\end{bmatrix}\begin{bmatrix}
  p_x \\
  p_y
\end{bmatrix} = \begin{bmatrix}
  s_x p_x \\
  s_y p_y
\end{bmatrix}$$
Comparison of scalings

- Uniform scaling preserves shape
- Non-uniform scaling introduces distortion
Two scalings in a row

- Two scalings in a row have a multiplicative effect
  - $P' = S_1 P$
  - $P'' = S_2 P' = S_2 S_1 P$

- Note the order of the scales appears to be reversed ($S_2$ before $S_1$), but it’s not
Composite transformation matrix

- The product of two or more transformation matrices is known as a composite transformation matrix

- For our two scalings we have

\[
\begin{bmatrix}
  s_{2x} & 0 \\
  0 & s_{2y}
\end{bmatrix}
\begin{bmatrix}
  s_{1x} & 0 \\
  0 & s_{1y}
\end{bmatrix}
= 
\begin{bmatrix}
  s_{2x}s_{1x} & 0 \\
  0 & s_{2y}s_{1y}
\end{bmatrix}
\]
A side effect

- Note that an object which does not have the origin inside it will be moved as well by a scaling
- We will see how to correct this effect shortly
Reflections

- Reflection through the $x$-axis negates the $y$ coordinate:
  \[
  \begin{bmatrix}
  1 & 0 \\
  0 & -1 \\
  \end{bmatrix}
  \]

- Reflection through the $y$-axis negates the $x$ coordinate:
  \[
  \begin{bmatrix}
  -1 & 0 \\
  0 & 1 \\
  \end{bmatrix}
  \]
Rotation

- Rotation transforms a point in a circular manner
- Positive angles imply a counter-clockwise rotation
The geometry of rotation

\[(x, y) = (r \cos \phi, r \sin \phi)\]

\[(x', y') = (r \cos (\phi + \theta), r \sin (\phi + \theta))\]
Rotation formula

- \((x, y) = (r \cos \phi, r \sin \phi)\)
- \((x', y') = (r \cos (\phi + \theta), r \sin (\phi + \theta))\)
- Recall sum of angle trig formulas:
  - \(\sin (\phi + \theta) = \sin \phi \cos \theta + \cos \phi \sin \theta\)
  - \(\cos (\phi + \theta) = \cos \phi \cos \theta - \sin \phi \sin \theta\)
- \(x' = r (\cos \phi \cos \theta - \sin \phi \sin \theta) = x \cos \theta - y \sin \theta\)
- \(y' = r (\sin \phi \cos \theta + \cos \phi \sin \theta) = y \cos \theta + x \sin \theta = x \sin \theta + y \cos \theta\)
Rotation matrix

Results from the previous slide give the matrix for rotation:

$$\begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta 
\end{bmatrix}$$

$$\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}$$
Two rotations in a row

- Two rotations in a row have an additive effect on the angles

\[ P' = R_1P \]

\[ P'' = R_2P' = R_2R_1P \]

\[
\begin{bmatrix}
\cos \theta_2 & -\sin \theta_2 \\
\sin \theta_2 & \cos \theta_2
\end{bmatrix}
\begin{bmatrix}
\cos \theta_1 & -\sin \theta_1 \\
\sin \theta_1 & \cos \theta_1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\cos \theta_2 \cos \theta_1 & -\sin \theta_2 \sin \theta_1 \\
\sin \theta_2 \cos \theta_1 & \sin \theta_2 \sin \theta_1
\end{bmatrix}
\begin{bmatrix}
\cos(\theta_2 + \theta_1) & -\sin(\theta_2 + \theta_1) \\
\sin(\theta_2 + \theta_1) & \cos(\theta_2 + \theta_1)
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\cos \theta_2 \cos \theta_1 & -\sin \theta_2 \sin \theta_1 + \sin \theta_2 \cos \theta_1 \\
\sin \theta_2 \cos \theta_1 + \cos \theta_2 \sin \theta_1 & \sin \theta_2 \sin \theta_1 + \cos \theta_2 \cos \theta_1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\cos \theta_2 \cos \theta_1 & -\sin \theta_2 \sin \theta_1 \\
\sin \theta_2 \cos \theta_1 & \sin \theta_2 \sin \theta_1 + \cos \theta_2 \cos \theta_1
\end{bmatrix}
\begin{bmatrix}
\cos \theta_1 & -\sin \theta_1 \\
\sin \theta_1 & \cos \theta_1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\cos(\theta_2 + \theta_1) & -\sin(\theta_2 + \theta_1) \\
\sin(\theta_2 + \theta_1) & \cos(\theta_2 + \theta_1)
\end{bmatrix}
\]
Reflection through $y=x$

- Rotate point $-45^\circ$ so that reflection line is $x$-axis
- Reflect through $x$-axis
- Rotate $45^\circ$ back
1. Rotate -45°

- Recall matrix to perform rotation:

\[
\begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta 
\end{bmatrix}
\]

- Fill in \( \theta = -45° \):

\[
\begin{bmatrix}
\cos(-45°) & -\sin(-45°) \\
\sin(-45°) & \cos(-45°) 
\end{bmatrix} = \begin{bmatrix}
.707 & .707 \\
-.707 & .707 
\end{bmatrix}
\]
2. Reflect through x-axis

- Recall matrix for x-axis reflection:

\[
\begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}
\]
3. Rotate back

- Recall matrix to perform rotation:

\[
\begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\]

- Fill in \( \theta = 45^\circ \):

\[
\begin{bmatrix}
\cos(45^\circ) & -\sin(45^\circ) \\
\sin(45^\circ) & \cos(45^\circ)
\end{bmatrix} = \begin{bmatrix}
.707 & -\.707 \\
.707 & .707
\end{bmatrix}
\]
Compute the total transformation

- Take each transformation matrix in the order the transformations need to be applied and multiply them together:

\[
\begin{bmatrix}
.707 & -.707 \\
.707 & .707
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}
\begin{bmatrix}
.707 & .707 \\
-.707 & .707
\end{bmatrix}
\]

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

- The result of reflecting a point through $y = x$ is the switching of the $x$ and $y$ coordinates of the point

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix} = \begin{bmatrix}
y \\
x
\end{bmatrix}
\]
Another transformation

Find the transformation that reflects a point through the line $y = 2$. 

\[
\begin{array}{c}
\circ \\
\hline
\circ \\
\hline
\circ \\
\hline
y = 2
\end{array}
\]
The steps

- Translate by -2 in the y direction
- Reflect through the x-axis
- Translate back
Translation

Translation is a transformation that results in a linear movement of a point

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  T_x \\
  T_y
\end{bmatrix}
+ \begin{bmatrix}
  x \\
  y
\end{bmatrix}
= \begin{bmatrix}
  T_x + x \\
  T_y + y
\end{bmatrix}
\]
A problem with translation

- Every other transformation we have looked at could be expressed using a $2 \times 2$ matrix and multiplying the point by the matrix.
- Translation cannot be expressed this way as we need to add fixed amounts to the $x$ and $y$ coordinates.
A solution

- So that translation behaves like our other transformations we want to express it using a matrix and multiplying the point by the matrix.
- Will need to add an extra coordinate to our point.
- This coordinate will have the value 1.
Translation matrix

- The matrix for doing a translation is

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

- Example:

\[
\begin{bmatrix}
1 & 0 & T_x \\
0 & 1 & T_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix} =
\begin{bmatrix}
x + T_x \\
y + T_y \\
1
\end{bmatrix}
\]
Homogeneous coordinates

- This new representation for our points (with the 1 in the extra coordinate) is known as **homogeneous coordinates**.
- There are times when the homogeneous coordinate is not 1; we’ll see this in perspective projections.
New versions of transformation matrices

- Homogeneous coordinates requires a change in the transformation matrices to accommodate the extra coordinate
- Matrices will now be $3 \times 3$
- Example: uniform scaling

$$
\begin{bmatrix}
   s & 0 & 0 \\
   0 & s & 0 \\
   0 & 0 & 1
\end{bmatrix}
$$
The transformations for our problem

- Translate by -2 in $y$
  \[ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \]

- Reflect through $x$-axis
  \[ B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

- Translate by +2 in $y$
  \[ C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \]
The composite transformation

The composite transformation for reflecting a point through the line $y=2$ is found by multiplying $\text{CBA}$:

$$
\begin{bmatrix}
1 & 0 & 0 \\
0 & -1 & 4 \\
0 & 0 & 1
\end{bmatrix}
$$
Rotations about an arbitrary point

- The rotations we have considered so far all occur around the origin.
- Suppose you had the following figure, which you would like to rotate about its center:
Rotation analysis

- If we applied the rotation transformation as we currently have it, the square would rotate about the origin
- This is not what we want
- We want the square to rotate in place
- Thus, we need to make the center of the square the point of rotation
Fixed points

- We call the point about which we want to rotate a **fixed point** as it will not change its position during the transformation.

- The same concept can be applied to scaling as well.
Steps for arbitrary rotation

- Translate the fixed point to the origin
  \[ A = \begin{bmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{bmatrix} \]

- Perform the rotation around the origin
  \[ B = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

- Translate the fixed point back to its position
  \[ C = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \]
Reflection through an arbitrary line

- Consider the following picture:

- What steps are necessary to perform the indicated reflection?
Arbitrary reflection transformation

- Sequence for arbitrary reflection:
  - Translate line by its $y$-intercept so that it goes through origin ($T$)
  - Rotate around origin so line lies on $x$-axis ($R$)
  - Reflect through $x$-axis ($X$)
  - Rotate back ($R^{-1}$)
  - Translate back ($T^{-1}$)

- Composite: $T^{-1}R^{-1}XRT$