Fundamental Concepts

- Transformations are critical to most graphics applications
- It’s important to understand how they work
- OpenGL maintains a "current transformation"
  - Vertices are sent through it for processing

Representation

- We’ll start with 2D transformations
- Most common transformations:
  - Translation – moving an object from one position to another
  - Scaling – changing the size of an object
  - Rotation – revolving an object around a pivot point
- We use a column vector representation for points
  - The point $P(x,y)$ is represented as:
    $$\begin{bmatrix} x \\ y \end{bmatrix}$$
- Operations on points thus become vector/matrix operations

Translation

- Implemented by addition
- Translation of $P(x,y)$ to $P'(x',y')$ is represented as
  $$P' = T + P$$
- We define the translation as
  $$x' = x + t_x \quad y' = y + t_y$$
- Representing these as vectors,
  $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Scaling

- Implemented with multiplication
- Unlike translation, scaling is relative to the origin
  - Translation is relative to the original position of the object
- Point $P(x,y)$ is scaled to $P'(x',y')$ with the products
  $$x' = s_x \cdot x \quad y' = s_y \cdot y$$
- In matrix form, $P' = S \cdot P$ is
  $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$
Rotation

• More complicated
• We rotate points through an angle $\theta$ about the origin
  - Positive angles rotate counter-clockwise
• Defined mathematically by
  
  \[
  x' = x \cdot \cos \theta - y \cdot \sin \theta \\
  y' = x \cdot \sin \theta + y \cdot \cos \theta
  \]

• In matrix form, $P' = R \cdot P$ is:

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

• Here is the result of rotating 45° about the origin:

Transformations

• Translation, Scaling, and Rotation are examples of affine transformations
  - Preserve parallelism of lines, but not lengths or angles
• Other affine transformations include reflection and shearing

Reflection and Shearing

• We reflect by inverting coordinate signs and/or reversing coordinate values:
  - Reflect around the y axis: $x' = -x$, $y' = y$
  - Reflect around the x axis: $x' = x$, $y' = -y$
  - Reflect around the positive diagonal ($m = 1$): $x' = y$, $y' = x$
  - Reflect around the negative diagonal ($m = -1$): $x' = -y$, $y' = -x$

• We shear by modifying each coordinate
  - Shear in y direction: $x' = x$, $y' = bx + y$
  - Shear in x direction: $y' = y$, $x' = x + ay$

Matrix Representations

• Here are the “big three” transformations, in matrix form:

\[
P' = T \cdot P \\
P' = S \cdot P \\
P' = R \cdot P
\]

• Unfortunately, translation is treated differently from rotation and scaling
  - Ideally, we want to treat them all the same way
    - This will allow us to easily combine different operations
• Solution: use **homogeneous coordinates**
  - First developed in the mid 1940s in geometry
  - Using homogeneous coordinates allows us to treat all three operations as multiplications
  - Basic modification: add a third coordinate to a point
  - The point \((x,y)\) becomes \((x,y,W)\)
  - Two sets of homogeneous coordinates \((x,y,W)\) and \((x',y',W')\) represent the same point if and only if \(x = nx, y = ny, W = nW\)
  - Examples: \((1,2,4)\), \((2,4,8)\), \((1,2,3)\)

• Using homogeneous coordinates allow us to use multiplication for translation:

  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} =
  \begin{bmatrix}
  1 & d_x & 0 \\
  0 & d_y & 0 \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

• Homogenized points form the plane defined by \(x = 0\)

• Normally, we use triples to represent points in 3-space
  - If we collect all triples \((tx,ty,tW)\) with \(t \neq 0\), we get a line

  \[
  \begin{bmatrix}
  x' \\
  y' \\
  W'
  \end{bmatrix} = \begin{bmatrix}
  t_x & 0 & 0 \\
  0 & t_y & 0 \\
  0 & 0 & t_W
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  W
  \end{bmatrix}
  \]

• We homogenize the point by dividing by \(W\)
  - This gives us the point \((x,y)\)
  - Thus, homogenized points form the plane defined by \(W = 1\)

• Points are now three-element vectors
  - Transformation matrices must now be 3x3
  - Transformation \(P' = T \cdot P\) becomes \(P' = T(d_x,d_y) \cdot P\)

• In matrix form:

  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} =
  \begin{bmatrix}
  1 & 0 & d_x \\
  0 & 1 & d_y \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

• Homogeneous Coords - Translation

• Homogeneous Coords - Scaling, Rotation

  \[
  P' = R(\theta) \cdot P
  \]

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

• Homogeneous Coords - Reflection

  - Around the \(x\) axis:
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} =
  \begin{bmatrix}
  1 & 0 & 0 \\
  0 & -1 & 0 \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

  - Around the \(y\) axis:
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} =
  \begin{bmatrix}
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

  - Around the diagonals:
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} =
  \begin{bmatrix}
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

• Homogeneous Coords - Shearing

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

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

• Shearing is a bit different:
Caution

- Note that some texts use row vectors rather than column vectors for points.
- They also alter the multiplication sequence to pre-multiply by row vectors rather than post-multiplying by column vectors:

\[ P \cdot M = P' \]

- We must perform a transposition to switch forms:

\[ (P \cdot M)^T = M^T \cdot P^T \]

Multiple Translations

- Intuitively, we know that translating a point twice is equivalent to a single, combined translation:

\[ P' = T(dx_1, dy_1) \cdot P \]

\[ P'' = T(dx_2, dy_2) \cdot P' \]

\[ P'' = T(dx_1 + dx_2, dy_1 + dy_2) \cdot P \]

Multiple Scaling

- The same is true for scaling (multiplicative, not additive):

\[ P' = S(sx_1, sy_1) \cdot P \]

\[ P'' = S(sx_2, sy_2) \cdot P' \]

\[ P'' = S(sx_1 \cdot sx_2, sy_1 \cdot sy_2) \cdot P \]

Multiple Rotation

- Rotation, like translation, is additive:

\[ P' = R(\theta) \cdot P \]

\[ P'' = R(\phi) \cdot P' \]

\[ P'' = R(\theta + \phi) \cdot P \]

Composition of Transforms - Rotation

- To translate point \( P(x,y) \) to the origin, we translate by the negative coordinates, \( T(-x,y) \).
- We then rotate this using \( R(\theta) \).
- Finally, we translate back by \( T(x,y) \).

The full sequence: \( T(x,y) \cdot R(\theta) \cdot T(-x,y) \)
Composition of Transforms – Scaling

- Scaling is also relative to the origin.
- To scale relative to an arbitrary point, we translate that point to the origin, scale by $S(s_x,s_y)$, and translate back.
- The full sequence: $T(x_1,y_1) \cdot S(s_x,s_y) \cdot T(-x_1,-y_1)$

$$
\begin{bmatrix}
1 & 0 & x_1 \\
0 & 1 & y_1 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -x_1 \\
0 & 1 & -y_1 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
s & 0 & x_1(1-s_x) \\
0 & s & y_1(1-s_y) \\
0 & 0 & 1
\end{bmatrix}
$$

Applications of Transformations

- So, where do we actually use these?
- Recall our window-to-viewport discussion.
  - Mapping the contents of the window to the viewport is often called the viewing transformation.
- We translate the window to the origin, scale it to the viewport size, and translate to the viewport position.
- Note that this also involves a coordinate system change.
  - World coordinates to view/screen coordinates

Efficiency

In general, composite matrices look like this:

$$
\begin{bmatrix}
r_{11} & r_{12} & d_x \\
r_{21} & r_{22} & d_y \\
0 & 0 & 1
\end{bmatrix}
$$

- Applying this to a point involves nine multiplications and six additions.
Efficiency

- We can cut this time significantly by noting the structure of the matrix
  - The bottom row, \([0 0 1]\), always yields 1 as the bottom coordinate of the resulting point
  - The rightmost column entries are always multiplied against 1, the bottom coordinate of the original point
- So, we can eliminate five multiplications and two additions:
  \[
  x' = r_{11} \cdot x + r_{12} \cdot y + d_x \\
  y' = r_{21} \cdot x + r_{22} \cdot y + d_y
  \]
- This is especially significant, given that we may be applying this to hundreds or thousands of vertices

Moving From 2D to 3D

- Not a lot of difference
- Again, we’ll use homogenized coordinates
  - \(P(x,y,z,W)\), with \(W \neq 0\)
- Add a \(z\) coordinate to our column vectors
- Add a fourth row and fourth column to our matrices
- Point \(P(x,y,z)\):
  \[
  \begin{bmatrix}
  1 \\
  0 \\
  0 \\
  0
  \end{bmatrix}
  \]

Standard Transformation Matrices

- Translation:
  \[
  \begin{bmatrix}
  1 & 0 & 0 & d_x \\
  0 & 1 & 0 & d_y \\
  0 & 0 & 1 & d_z \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
- Scaling:
  \[
  \begin{bmatrix}
  s_x & 0 & 0 & 0 \\
  0 & s_y & 0 & 0 \\
  0 & 0 & s_z & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
- We can verify these:
  \[
  T(d_x,d_y,d_z) \cdot [x,y,z,1]^T = [x+d_x, y+d_y, z+d_z, 1]^T \\
  T(s_x,s_y,s_z) \cdot [x,y,z,1]^T = [x \cdot s_x, y \cdot s_y, z \cdot s_z, 1]^T
  \]

Rotation

- Rotation matrix differs for each axis of rotation
- Consider our 3D coordinate system:
  - Called a right-handed coordinate system
  - Looking along the axis from positive coordinates to the origin, rotation is counterclockwise

Rotation Around Z

- 2D rotation is just rotation around the \(z\) axis:
  \[
  \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 & 0 \\
  \sin \theta & \cos \theta & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

Rotation Around X and Y

- Rotation around the \(x\) axis moves from \(y\) to \(z\):
  \[
  \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & \cos \theta & -\sin \theta & 0 \\
  0 & \sin \theta & \cos \theta & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
- Rotation around the \(y\) axis moves from \(z\) to \(x\):
  \[
  \begin{bmatrix}
  \cos \theta & 0 & \sin \theta & 0 \\
  0 & 1 & 0 & 0 \\
  -\sin \theta & 0 & \cos \theta & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
Composition of 3D Transformations

- Basically the same as in 2D
- Consider this diagram:

Our goal is to transform it so that it looks like this:

Composition of 3D Transformations

- This can be seen as a sequence of four transforms:
  - Translate $P_1$ to the origin
  - Rotate about $z$ so that $P_1P_2$ lies along the $z$ axis
  - Rotate about $x$ so that $P_1P_2$ lies in the $(y,z)$ plane
  - Rotate about $y$ so that $P_1P_3$ lies in the $(x,z)$ plane

Composition of 3D Transformations

- Translation to the origin is just $T(-x_1,-y_1,-z_1)$
  - $T(-x_1,-y_1,-z_1) \cdot P_1(x_1,y_1,z_1) = P'_1(0,0,0)$
  - $T(-x_1,-y_1,-z_1) \cdot P_2(x_2,y_2,z_2) = P'_2(x_2-x_1, y_2-y_1, z_2-z_1)$
  - $T(-x_1,-y_1,-z_1) \cdot P_3(x_3,y_3,z_3) = P'_3(x_3-x_1, y_3-y_1, z_3-z_1)$

Composition of 3D Transformations

- Determining rotations is a bit more involved
- We begin by projecting $P'_2$ onto the $(x,z)$ plane
  - Point $(x'_2,0,z'_2)$
  - Distance $D_1$ from origin
  - Angle from the $(x,y)$ plane is $\theta$
  - Angle of rotation around $y$ will be $\phi - 90$, which is just $\theta - 90$

Composition of 3D Transformations

- Our rotation is $R(\theta-90)$
  - Thus, $P_2'' = R_y(\theta-90) \cdot P_2'$

Composition of 3D Transformations

- Next, we do the same thing for the $x$ axis
- Our angle of rotation is $\phi$

Alternate view: when we apply a transformation, we’re not changing the coordinates of a point within a coordinate system
- Instead, we’re changing the coordinate system itself

Transforming Coordinate Systems

- Alternate view: when we apply a transformation, we’re not changing the coordinates of a point within a coordinate system
- Instead, we’re changing the coordinate system itself
Transforming Coordinate Systems

- We can apply multiple transformations:

![Image of coordinate systems](image)

**FIGURE 5.12** Transforming a coordinate system twice.

Transformations in OpenGL

- OpenGL maintains a number of matrices:

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x'' \\
  y'' \\
  z'' \\
  w''
\end{bmatrix}
\]

Perspective division → Viewport transformation → Window coordinates

![Image of OpenGL transformations](image)

**FIGURE 5.13** Elementary changes between coordinate systems.

- We switch between them with the `glMatrixMode()` routine.
- The most recent call to `glMatrixMode()` selects OpenGL’s current transformation (CT) matrix.
- Transformations are applied by postmultiplying CT by the desired transformation:

\[ CT = CT \cdot M \]

- More on how this works in the next set of notes.

![Image of OpenGL matrices](image)