2D OpenGL Transformations

OpenGL transformation commands set up a 4 by 4 transformation matrix for all transformations. Therefore, the transformation looks like this:

\[
\begin{bmatrix}
Q_x \\
Q_y \\
0 \\
1
\end{bmatrix} =
\begin{bmatrix}
m_{11} & m_{12} & 0 & t_x \\
m_{21} & m_{22} & 0 & t_y \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
P_x \\
P_y \\
0 \\
1
\end{bmatrix}
\]

As mentioned previously the transformation matrix, \( M \), is normally created using one or more of the following OpenGL function calls:

Translation (in the x or y directions):
\[
\text{glTranslatef}(t_x, t_y, 0.0);
\]
Rotation (\( \theta \) about the z-axis):
\[
\text{glRotatef}(\theta, 0.0, 0.0, 1.0);
\]
Scaling (in the x or y directions):
\[
\text{glScalef}(x, y, 1.0);
\]
Shearing: there is no specific GL command; use a combination of scaling and rotation.

Combining Transformations

We can combine two or more transformations and compactly define them using a single matrix. Consider the following rectangular object:

Its points are defined in the local or object coord system as a set of points starting from the origin and proceeding in a CCW direction as follows:

OBJ: \{(0,0), (2, 0), (2, 1), (0, 1)\}

- Suppose that we translate the object by 1 unit in the x-direction and 1 unit in the y-direction. Denote this transformation by \( T \).
- We can express this transformation as:
  \[
P'_{\text{OBJ}} = T P_{\text{OBJ}}
\]
Now suppose that we rotate the object by 45° about the origin (z-axis) and denote this transformation by R.

- We can express this overall transformation as:
  \[ P''_{OBJ} = R P'_{OBJ} \]
  \[ = R T P_{OBJ} \]

- Note that all of the transformations were described with respect to a fixed set of axes, namely the origin.

Now, if we perform the transformations in reverse order starting with the rotation:

- We can express the rotation transformation as:
  \[ P'_{OBJ} = R P_{OBJ} \]

- Then, after performing the translation we get:
  \[ P''_{OBJ} = T P'_{OBJ} \]
  \[ = T R P_{OBJ} \]

Notice that the result is not the same; matrix composition is non-commutative:

\[ R T \neq T R \]

Also, there are 2 ways of looking at these transformations; either in world coords or in object coords. If we are describing all of our transformations w.r.t. a fixed set of axes, then the transformations are written down from right to left, as in the example above.

If we are describing all of our transformations in terms of a local (object) coordinate system, they should be ordered from left to right.

Both views are correct, but often it is easier to think in terms moving with a local coordinate system, especially when displaying hierarchical objects that are relative to one another.