

## Transformations (9/12/6)

*Lecturer: James Kuffner**Scribes: Rosen Diankov*

The OpenGL matrix stack is processed from top to bottom. The most recently pushed transformation by `glRotate`, `glScale`, `glTranslate`, etc becomes the new top of the stack. The stack is actually two levels in that transformations can be grouped by `glPushMatrix` and `glPopMatrix`. Basically, `glPushMatrix` can be regarded as a marker somewhere in the stack. When called, this marker gets placed on the stack. When `glPopMatrix` is called, it will keep on popping transformations off the stack until it reaches the last pushed matrix. Imagine that you have a group of objects that share the same rotation, but have different translations. Then ideally we should apply the rotation only once and the code will look like:

- `glMatrixMode(GL_MODELVIEW);` *change the mode to model view*
- `glRotatef(degrees, x, y, z);` *set the rotation*
- `glPushMatrix();` *set a marker*
- `glLoadIdentity();` *reset the pushed matrix*
- `glTranslatef(-1, 0, 0);` *translate to coordinates (-1,0,0)*
- `DrawObject();`
- `glPopMatrix();` *remove the translation*
- `glTranslatef(1, 0, 0);` *translate to (1,0,0)*
- `DrawObject();`

Usually rotations rotate objects around the center. To rotate an object around an arbitrary point  $(P_x, P_y, P_z)$  by a rotation  $R$ : first the object has to be translated so that the point of rotation is in the center, then rotated, then translated back.

$$\begin{pmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} R \begin{pmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} X = X'$$

In OpenGL this will look like:

- `glMatrixMode(GL_MODELVIEW);`
- `glPushMatrix();`
- `glLoadIdentity();`
- `glTranslatef( $P_x, P_y, P_z$ );`

- `glRotatef(degrees, x, y, z);`
- `glTranslatef(-Px, -Py, -Pz);`

Note that a scaling matrix generated by `glScalef(sx, sy, sz)` scales around the x, y, and z axes of the current transformation. Therefore a scale before a rotation is different than a scale after a rotation. You can think of the column vectors of a transformation from local to world space as the x, y, and z axis vectors of the local coordinate system in world coordinates.

The types of transformations we're interested in are: rigid (translation and rotation), affine (rigid along with scaling and shearing), and projective. They are all subgroups of linear transformations. In general, a linear transformation  $T$  is such that  $T(x_1 + x_2) = T(x_1) + T(x_2)$ , and  $T(\alpha x_1) = \alpha T(x_1)$  for any vectors  $x_1$  and  $x_2$ . It can be proven that all linear transformations can be represented by matrices with matrix multiplication. However, lengths and angles between two vectors are only preserved for rigid transformations. Parallel lines are preserved between any linear transformation.

Any 2D point is represented as  $(x, y, 1)$  where the last coordinate is the homogeneous coordinate  $w$ . We are only interested in the points on the  $w = 1$  plane. Any point in the 3D space  $(x, y, w)$  is equivalent to  $(\frac{x}{w}, \frac{y}{w}, 1)$ , where the division represents the projection of the point to the  $w = 1$  plane. These two points are equivalent and the division happens implicitly once real 2D coordinates are needed. For 4x4 matrices in 3D space, manipulating the  $w$  coordinate correctly will allow for perspective transformations that project all geometry to the screen plane such that farther objects are smaller than closer objects.

Matrix transformations can be easily composed with each other. For example a point transformed by  $M_1$  then by  $M_2$  is the same as the point transformed by  $M_2M_1$ . An object is usually composed of many points, so when transforming its points to the screen, it is best to compute only one matrix that is the composition of all the smaller transformations.