Chapter 11

OpenGL – A 3D Graphics API

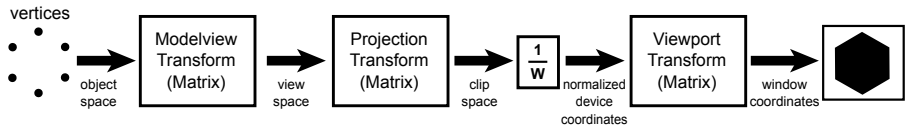
11.1 Pipeline architecture

You can think of OpenGL as a pipeline, with processing stations along the way, that takes in the vertices of your model (i.e points in 3D object space) and transforms them into screen coordinates. The basic pipeline is organized as shown below.

The 3D coordinates of vertices enter on the left, and undergo a series of transformations and operations. The operations include:

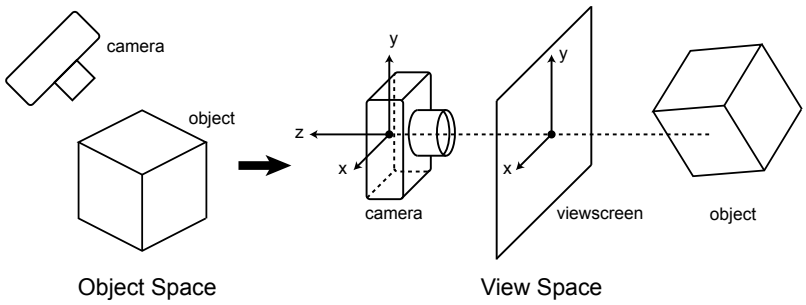
1. Clipping away vertices outside of the view volume,
2. Determining which surfaces are visible and which are hidden (occluded) behind other surfaces,
3. *Rasterizing* polygons (i.e. filling in the space between polygon edges), and drawing lines,
4. Shading and texturing of the pixels making up the polygonal faces,
5. Performing a number of image processing operations such as blurring and compositing.

The transformations are described below.

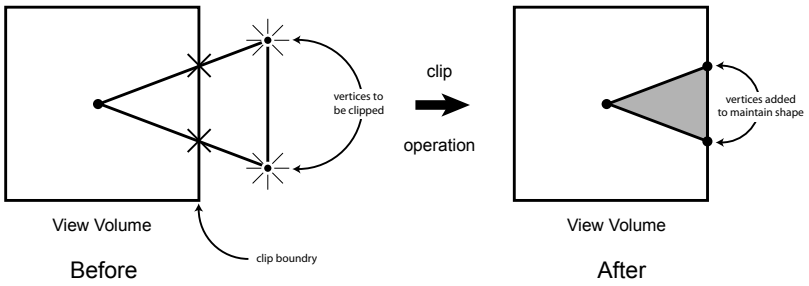


11.2 Transformations in the OpenGL pipeline

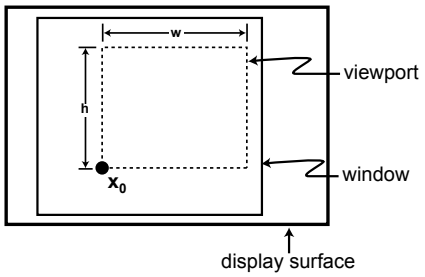
The *Modelview Matrix* transforms object coordinates into the viewing (or camera) coordinate system, where the camera viewpoint is at the origin $(0, 0, 0)$, the view direction is aligned with the negative z axis, and the view screen is perpendicular to the view direction, so that the x and y axes are parallel to the plane of the view screen. An example Modelview transform is shown in the figure below. The steps implied in the transform are first translating the camera viewpoint to the origin and then rotating the camera to properly align it with the x, y, and z coordinate axes. Note: This is equivalent to translating the entire scene in the opposite direction, and then rotating the scene about the origin in the opposite direction, which is what is actually done in OpenGL (i.e. the vertices of the model are transformed).



The *Projection Matrix* transforms the vertices in view coordinates into the canonical view volume (a cube of sides $2 \times 2 \times 2$, centered at the origin, and aligned with the 3 coordinate axes). Typically, this will be either by an orthographic projection or a perspective projection. This transform includes multiplication by the projection transformation matrix followed by a normalization of each vertex, calculated by dividing each vertex by its own w coordinate. We say that the vertices thus transformed are in *Normalized Device Coordinates* as they are now independent of any display or camera parameters. During this process, OpenGL discards any vertices not in the view volume by clipping them to the sides of the view volume, as shown below. The integrity of edges and polygon faces is maintained by adding a new vertex wherever the view volume clips an edge. Thus, after conversion to Normalized Device Coordinates, and clipping, all remaining vertices have coordinates in the range $-1 \leq x, y, z \leq 1$.



Finally, the *Viewport Matrix* transforms vertices into window coordinates. After this transformation, everything is in pixel coordinates relative to the lower left corner of the window on the display. As shown to the right, the viewport is a rectangular region on the window defined by the position $\mathbf{x}_0$ of its lower left corner, its width w and its height h. The Viewport Matrix performs a scale of the canonical view volume (in Normalized Device Coordinates) by $w/2$ in the horizontal direction, and by $h/2$ in the vertical direction, transforming the front face of the volume from a 2×2 square to a $w \times h$ rectangle. This is followed by a translation of the front lower-lefthand corner $(-w/2, -h/2, -1)$ to the position $\mathbf{x}_0$. In this space, all the rasterization and shading operations are performed, and each resulting pixel is drawn to the display.



11.3 OpenGL API calls affecting the pipeline transforms

The OpenGL API defines a set of procedure calls that are used to update the three matrices in the pipeline. These are described below.

11.3.1 Modelview Matrix

One selects which matrix is to be affected by matrix operations in OpenGL, by a call to the procedure `glMatrixMode()`. This procedure takes one parameter, which is a symbol indicating which matrix to affect. Once this call is made,

the indicated matrix will remain the active matrix until another call is made to `glMatrixMode()`.

The Modelview Matrix is selected via a call to

```
glMatrixMode(GL_MODELVIEW);
```

The Modelview Matrix is used both to transform the vertices of the various objects in the scene from their own object coordinates into scene (world) coordinates, and to position the camera within the scene. The commands typically used to affect the Modelview matrix are:

`glLoadIdentity();` – to initialize the matrix to the identity matrix

`glTranslatef(`$\triangle x$`,` $\triangle y$`,` $\triangle z$`);` – to translate by the vector $\begin{bmatrix} \triangle x \\ \triangle y \\ \triangle z \end{bmatrix}$,

`glScalef(`s_x`,` s_y`,` s_z`);` – to scale by the scale factors s_x, s_y, and s_z,

`glRotatef(`θ`,` u_x`,` u_y`,` u_z`);` – to rotate θ degrees about the axis $\mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ u_x \end{bmatrix}$.

Note that the `f` following `glTranslate_()`, `glScale_()` or `glRotate_()` indicates that parameters are floats. To use doubles, use `d` or to use integers use `i` instead.

The translate, scale and rotate calls each create a matrix to do the indicated transform, and multiply it onto the right hand side of the currently selected matrix (usually the Modelview Matrix). Therefore, the sequence:

`glLoadIdentity();` $\Longrightarrow M = I$
`glRotatef(30, 1, 0, 0);` $\Longrightarrow M = IR_{30,x}$
`glTranslatef(-5, -5, -5);` $\Longrightarrow M = IR_{30,x}T_{-5,-5,-5}$

results in a Modelview matrix M, which when multiplied by a vertex $\mathbf{x}$, yields a transformed vertex

$$\mathbf{x}' = M\mathbf{x}$$

that is first translated by -5 in each direction, and then rotated by 30° about the x axis.

11.3.2 Projection Matrix

The call

```
glMatrixMode(GL_PROJECTION);
```

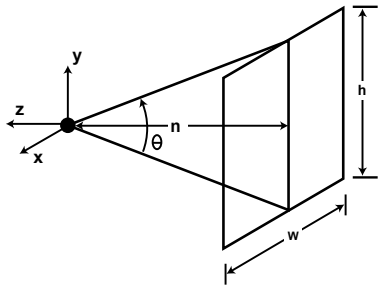
tells OpenGL that calls that affect matrices should be applied to the Projection Matrix. The calls that are especially applicable to the Projection Matrix are:

`GlLoadIdentity();` – initializes the matrix to the identity matrix.

`GlOrtho(l, r, b, t, n, f);` – multiplies an orthographic projection matrix onto the right side of the matrix.

`GlFrustum(l, r, b, t, n, f);` – multiplies a perspective projection matrix onto the right side of the matrix.

In these calls, the parameters `l`, `r`, `b`, `t` specify the left, right, bottom, and top coordinates of the viewscreen; while `n` and `f` are the distances, along the view direction, of the near and far clipping planes. The near clipping plane corresponds with the viewplane, and `n` corresponds with the focal length of the camera. The far clipping plane is a distance, beyond which, the camera ignores anything in the scene.



Alternatively, the call

```
gluPerspective(theta, aspect, n, f);
```

can be used in place of `glFrustum()` to specify a perspective projection. The parameter `theta` is the vertical camera viewing angle in degrees, and `aspect` is the aspect ratio of the window (i.e. its width divided by its height). The figure to the left shows the relationship between the parameters and the perspective view volume. If we let α stand for `aspect` and θ stand for `theta` the relationships between the width w and height h of the window are given by

$$h = 2n \tan \theta/2,$$
$$w = \alpha h.$$

.

For example, a camera with view screen centered along the view direction would have its projection transform set up by:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
glOrtho(-w/2, w/2, -h/2, h/2, n, f);
```

for orthographic projection, or

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
glFrustum(-w/2, w/2, -h/2, h/2, n, f);
```

for perspective projection.

If OpenGL is being used strictly for 2D viewing, the call

```
gluOrtho2d(l, r, b, t);
```

can be used to create an orthographic projection, whose view plane is at the origin and that is of zero depth, so that all z coordinates must be 0 for vertices to be visible (i.e only the x, y coordinates matter).

11.3.3 Viewport

The viewport is set by a single call to

```
glViewport(x0, y0, w, h);
```

This causes the viewport matrix to be constructed, scaling the normalized device coordinates so that the width becomes `w` and the height becomes `h`, and translating the lower left corner to (`x0`, `y0`, 0).