Setting up the view

1 Overview

This document describes a way of setting up the modelview and projection matrices so that a 3D model shows up, its center is typically roughly at the center of the window and the size is reasonable (not too small, not too large).

2 Normalizing the model

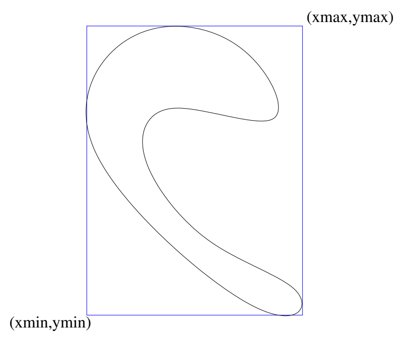

Figure 1: Bounding box in 2D (object shown in black, bounding box in blue)

A simple (albeit not necessarily the best) method of estimating where the center of the object uses the concept of the bounding box. The bounding box is the smallest axis-oriented rectangular parallelepiped containing the model. It is easy to see that the bounding box is the set of all points (x, y, z) such that $x_{min} \leq x \leq x_{max}$, $y_{min} \leq y \leq y_{max}$ and $z_{min} \leq z \leq z_{max}$, where $?_{min}$ and $?_{max}$ are the minimum and the maximum ?-coordinates of the model for $? \in \{x, y, z\}$. Thus it is very easy to compute (just go over all vertices keeping track of minimum and maximum value of each coordinate). The bounding box extends from $(x_{min}, y_{min}, z_{min})$ to $(x_{max}, y_{max}, z_{max})$ (see Figure 1 for a 2D counterpart).

The bounding box carries information about the center and size of the model. Here (for some models this is far from optimal) by the center of the model we'll mean the center of its bounding box or

$$C = (\frac{x_{min} + x_{max}}{2}, \frac{y_{min} + y_{max}}{2}, \frac{z_{min} + z_{max}}{2}).$$

By the size of the model we'll mean the maximum dimension of its bounding box, i.e.

$$s = \max\{x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min}\}.$$

We can take advantage of this and 'normalize' the model, i.e. put its center at the origin and scale it uniformly so that the largest magnitude of a vertex coordinate is 1 (i.e. so that the smallest axis-oriented cube centered at the origin is the cube extending from -1 to 1 in every dimension). This can be done by applying two transformations:

Translate by -C

Scale uniformly by $\frac{2}{s}$.

Note that we want to scale uniformly here to preserve the shape of the object. Also, we scale by $2/s$ because the edge length of the cube exending from -1 to 1 in x, y and z is 2.

3 Projection

The projection matrix cannot be thought of in isolation from the modelview matrix: they have to work together to produce a good image. What we'll do here is build upon the `glm::perspective` function, that construct the projection matrix for a camera that points in the $-Z$ direction and has a programmer-provided field of view (Figure 2).

Obviously, trying to render the normalized model with this projection matrix is not a good idea – since the viewpoint (origin) is at the center of the normalized model. But translating the model in the direction of $-Z$ axis will work (see Figure). We need to translate so that the bounding box fits tightly between the planes defined by the field of view, as shown in Figure 2.

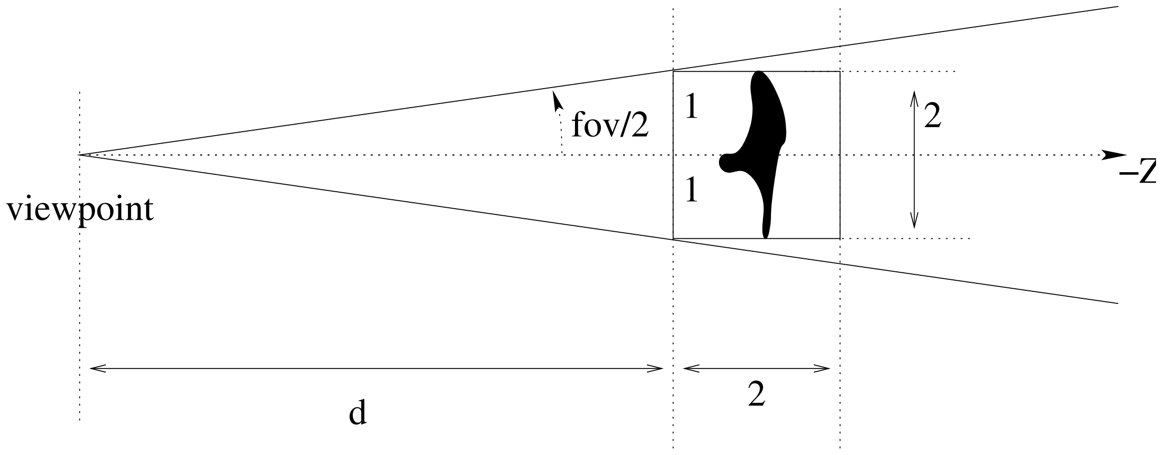

Figure 2: 2D Section (in either xz or yz plane) of the perspective projection, assuming the aspect ratio of 1, i.e. square window. **fov** is the field of view. The square box is a section of cube of size 2 (the cube centered at the origin containing the normalized model translated in the $-Z$ direction). The black blob is the section of the model, translated by some amount in the $-Z$ direction.

If the field of view is α then $1/d = \tan(\alpha/2)$ (see Figure) and therefore $d = 1/\tan(\alpha/2)$. Thus, the translation that needs to be applied to the normalized model is $(0, 0, -(d+1))$. The '+1' is needed because we need to move the center of the bounding box from the origin to $(0, 0, -(d+1))$.

Now, one more thing that the `glm::perspective` call requires is information on front and back clipping planes, i.e. how far they are from the viewpoint (origin). It is obvious from Figure 2 that the front plane can be put d away from the viewpoint and the back plane $d+2$ away. However, we'll want to rotate the object around its center $(0, 0, -(d+1))$ using the trackball interface. In the worst case, the diagonal of the bounding cube of the object may stretch along the $z-$axis. and therefore the $z-$ coordinates of the vertices may potentially be as far as $\sqrt{3}$ (half of the length of the diagonal of the cube) away from the $z-$ coordinate of the center of the cube, i.e. $d+1$. Thus, the optimal front and back clipping planes are $d+1-\sqrt{3}$ and $d+1+\sqrt{3}$ away from the origin. Of course, you don't need to be optimal here and putting the planes at $d-1$ and $d+3$ will do.

4 Conclusion

Here is how to get good modelview and projection matrices. First, pick a reasonable field of view α (something like 10-20 degrees).

Use `glm::perspective` functions to compute the projection matrix. You'll need to send this matrix to a uniform variable (see sample code and project description for details). Use α as the field of view and 1 as the aspect ratio (we'll render into square window). Place the front and back clipping planes as described in the previous section, e.g. at $d-1$ and $d+3$ where $d = 1/\tan(\alpha/2)$.

As the modelview matrix, use the superposition of the following transformations:

Translate by -C

Scale uniformly by $\frac{2}{s}$.

Rotate (controlled by the trackball interface)

Translate by $(0, 0, -(d+1))$

In order to zoom in or out, you can change the field of view of the projection matrix. Don't move or scale the model (this is not what you do with a camera!). However, when zooming in or out, don't recompute the front or back clipping planes or modelview matrix for the new field of view (otherwise, your zoom won't work since you would be simultanously, say, decreasing the field of view and moving the object away).