3DKIT

A set of objects for displaying wireframe 3D graphics.

Daniel Mark Gessel
5 Roylencroft Ln.
Rose Valley, PA 19065
gessel@cs.swarthmore.edu untill Sept. 1990

Introduction

I have set out to create an example of how the AppKit's hierarchical View structure can be adapted for displaying 3D graphics. You will find the code for the classes I have created below. It will be submitted, along with a copy of this article, and an example program created through Interface Builder, to various ftp sites.

Please feel free to contact me at the above addresses about suggestions for additions, modifications, successes and failures you have had with this kit, as well as any bugs that may have crept in. I will submit this code along with an example created in Interface Builder to various archive sites. If interest warrants, I will keep the kit up to date with suggestions and modifications, to the best of my ability.

Core Structure

The file `3D.h` declares the type `vector3D`. It is used throughout the kit as a primitive type.

There are three core objects for this kit.

Context3D

This is a subclass of View. It holds unique information that is vital for the presentation of 3D graphics on screen. In this simple case, it holds the distance from the viewer's eye to the picture plane, which the screen represents, in the instance variable *`pictureDistance`*. The conversion of 3D coordinates to 2D coordinates is dependent on this distance. It also holds distance to a z clipping plane. Any drawing that would be closer to the eye than *clippingDistance* is not drawn.

The negative z axis is into the screen, the x axis is toward the right, and the y axis is upward. This makes the coordinate system right handed. The clipping and picture planes are represented as distances, however. This means that the picture plane has the implicit formula 0=-*z*-*`pictureDistance`*. Simalarly for the clipping plane.

View3D

This is a subclass of object. It is not a responder, since this 3D Kit assumes that the 3D objects drawn by this kit are purely graphical. This is not necessarilly how we would like it to be, but, because this kit is slow, and only implements drawing routines for wireframe images, making a Responder3D class seemes unnecessary. All drawing is done by View3D subclasses, using `moveto:`, `lineto:` and `polygon:howMany:` messages sent to `self` or directly to `superview`, where `superview` has a function identical to the superview in the View class. Sending these messages directly to the `superview` will avoid the View3D's own transformation object. This may be useful at times.

Transformation3D

This is also a subclass of object. It implements the methods `operateOn:` and `operateOn:howMany:` such that they do nothing. The methods are to be overwritten (as in the subclass Linear3D) to perform transformations on vector3Ds, pointers to which are passed as arguments to these methods. Linear3D implements many useful transformations, including rotations, translations and combinations of these. There is a `transformation` instance variable in the View3D class. This can be assigned to an instance of a subclass of Transformation3D, which will automatically be sent `operateOn:` messages with the vector3Ds which are passed to the drawing messages mentioned above.

Comments

There is the Linear3D (mentioned above), a Transformation3D subclass, and Cube, a View3D which draws a simple unit cube centered at the origin. The code is commented, with instructions on how to use the classes in the interface files.

The drawing as implemented is slow. There is no attempt at optimization, just to create a simple example of object oriented 3D drawing in a style similar to the AppKit.