Interactive Geometry in 3D

Jacob Welsh
TJHSST Computer Systems Lab Senior Research Project
2007-2008

April 7, 2008

Abstract

The goal of this project is to write a program that allows its user to create and manipulate a complex system of geometric objects in space. From a few basic object types, interesting and useful constructions can be built. This could be useful for education, mathematical or scientific research or visualization, or just for fun.

Keywords: Euclidean geometry, human-computer interaction, educational computing, scientific visualization

1 Background

For a while there has been software for computer assisted design (CAD), which utilizes a few basic shapes and techniques such as snapping and numeric entry to create precise, polished diagrams of a product that can then be used in its manufacturing.

A similar sort of program is used for 3D modeling, in which the user constructs polygon meshes in three dimensions: freehand; with snapping; and numerically. My program aims to be more focused on geometric objects and dynamic preservation of their relationships as some are manipulated. The leading example of this is a commercial program called The Geometer's Sketchpad. Its interface is rather clunky, and it is limited to two dimensions. However, the fact that it is possible to build primitive 3D constructions in it illustrates the power behind the idea of geometric construction. The basic

philosophy for the user interface of my program comes from the modeling program Blender and the text editor VI.

A somewhat different and interesting approach is taken by the SKETCH project of Zeleznik, Herndon and Hughes of Brown University from the mid-1990's, and the commercial program SketchUp. With these programs, the goal is quick, informal visualization of a scene from the user's imagination, much like a pencil-and-paper sketch in three dimensions. SKETCH is particularly noteworthy for making extensive use of mouse gestures for determining how to interpret the drawn lines. My program will aim for a simultaneous use of mouse and keyboard for maximum flexibility and efficiency.

2 Project Development

In the first quarter I worked on the basic structure of the program. A variety of functions were developed abstracting common graphics and mathematical routines. SDL and OpenGL were used as the underlying graphics libraries, but the core routines of the program are ignorant of that, calling routines that deal with geometric objects – points, lines, and the space that contains them. The original conception was that two parallel display backends would be implemented: one using OpenGL for full 3D drawing and performance, and software-only SDL routines to offer more limited support for computers lacking OpenGL. This approach was soon abandoned due to the complexity and confusion it added to the prototype program, but may eventually be added back. Another of my early ideas was that all objects in the scene would be linked in a multidirectional tree, which would allow only the necessary dependent objects to be recalculated when their parents moved. Again, this approach had to be greatly simplified for the sake of getting a working prototype off the ground.

Second quarter saw the various components of the program come together in a functional way. After extensively considering the data structures representing the geometry, I settled on a linked list of all the objects in the scene, with pointers to parent objects when necessary for correct drawing and calculation. This linked list is not actually global to the program, but rather contained in a "space" superstructure. In addition to providing a handy place for storing information like 3D projection paramaters, a space completely contains a scene. This allows future features such as multiple spaces displayed in separate tabs or panes. With the data structures layed

out properly, I created wrapper functions for their memory management and linking, enabling core code to be easy to write as well as comprehend. From the programmer's standpoint, objects such as points and lines can be added to the scene and assigned locations with ease.

Once the code was functioning properly, it became possible to focus on the implementation of user interface. The first task was to be able to easily select desired objects from a potentially dense scene. The mouse pointer should not have to exactly touch the desired object; rather, the program should select the most reasonable object given the position of the pointer. For selecting points, this is done by simply finding the point closest to the pointer location. However, since the scene is in three dimensions, the distance in question is actually from the point to the ray through the mouse pointer from the viewing point, and can be found through vector algebra. A similar principle applies to objects other than points, but it was not clear whether such rules would work well in combination to find the most desired object.

To resolve this doubt I employed some field theory. A point can be thought of as emitting a spherical field, decreasing in strength with distance from the point. Similarly, a line segment emits a cylindrical field with spherical end caps, again representing the shortest distance to a given point in space. Thus, for a certain location of the mouse pointer, the desired selection was the object whose field was strongest at that location. This has a flaw though; consider a line segment with endpoints. The distance to the segment is often equal to the distance to one endpoint. This can be resolved by scaling down the strength of the segment's field. Distant locations will select one of the endpoints, but sufficiently close locations can still select the segment itself.

The third quarter was focused on further development of the program's user interface. Multiple interaction modes were established, to be switched through using the keyboard. In Point mode, the user simply clicks to create points; in Line mode each click creates an endpoint of a line segment. When drawing lines, Snap mode can be enabled, causing lines to attach to existing points rather than create new ones. This allows a beginning step toward the heirarchically related objects that are the goal of the program. In Selection mode, points can be selected based on their distance from the mouse pointer as described above, and moved around simply by dragging the mouse. Since the lines are defined only by their parent points, they are always redrawn in the updated position.

3 Results

The current program can create points and line segments that connect to each other. The point closest to the mouse pointer can be highlighted, using the selection algorithm. While the current demonstration appears in two dimensions, all the code is scalable to three, as will become evident once the user interface is developed enough to perform rotations of the space.

The most interesting aspect of the program, however, will come with the implementation of constrained objects, such as a point lying on a line segment or point of intersection, and the addition of circles. With this, principles of Euclidean geometry can be dynamically illustrated, and more sophisticated features will follow.