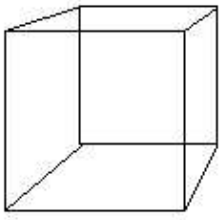
CSCI 1300 — Introduction to Computing
Professor Michael Mozer
Fall 2003

Assigned: November 4, 2003
Due: November 18, 2003

Wire Frame 3-D Graphics

In this exercise, you'll design and implement a program that displays a three dimensional "wire frame" cube and lets you rotate, shrink, or expand it. A wire frame figure is one that has only edges and vertices (corners) and not faces. Thus, you don't have to worry about what to do when some part of a figure is hidden by some other part. Here is a picture of a wire-frame cube:



A wire-frame cube is simply a collection of edges. Drawing the cube means drawing the edges. An edge is simply a line drawn between two points, so you can represent an edge as two points. A point in 3-dimensional space has three coordinates, (x, y, z).

1. Program requirements

(a) The program should read a description of the cube from a file. The description should specify the set of edges in the cube, and the vertices (points) that make up the edges. The specification is in three dimensional space. Although the assignment is to display a cube, your program should be general enough that you could replace the data points in this file and display another figure, such as a pyramid. You do not need to do error checking on the contents of the file; assume that it contains valid data. No knowledge of the figure may be built into your program (e.g., the number of edges or vertices), except you may specify an upper-limit on the number of edges and vertices permitted.

(b) The program should be able to rotate the cube around the x, y or z axes, zoom in or out, and and display the resulting figure. (We explain how to do this below.)

(c) Allow the user to specify repeatedly which of these operations to perform. The effect should be that the user can freely play with the cube, viewing it from different angles and zooming in or out on it. The following keys should be understood by your program:

up-arrow key : rotate the cube -5° around the x axis

down-arrow key : rotate the cube +5° around the x axis

left-arrow key : rotate the cube -5° around the y axis

right-arrow key : rotate the cube +5° around the y axis

< key : rotate the cube +5° around the z axis

> key : rotate the cube -5° around the z axis

z key : zoom out by a factor of 0.9

Z key : zoom out by a factor of 1.1

To help you organize your work, I have broken the task into five phases, which I describe below. I recommend that you take your work to a course staff member after completing each phase, so that we can offer you advice and feedback and make sure that you are on the right track. Please begin work on this assignment immediately. It will require three weeks of steady work, not one or two all nighters.

2. The mathematics of graphics operations

The data and operations required for this program have convenient mathematical representations. You can represent a point in a three dimensional space by a vector with three components, one for the x coordinate, one for the y coordinate, and one for the z coordinate.

The operations of rotating, shrinking, and expanding can all be represented as multiplication of the vectors representing the points of a figure by appropriate matrices; in our case, these are 3×3 matrices. (If you aren't familiar with the terms "vector" and "matrix," a vector is nothing more than a one-dimensional array of numbers, and a matrix is a two-dimensional array of numbers.) If you consider a point in 3-D space as a 3-element vector and you multiply this vector by a 3×3 matrix, you get a new 3-element vector representing the transformed point.

Vector-matrix multiplication works like this. Suppose we are multiplying the vector (v_0, v_1, v_2) by the matrix

$$\begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{pmatrix}$$

To get the first coordinate of the result, line up the vector with the first column of the matrix, multiply the corresponding entries, and add:

$$v_0 \times m_{0,0} + v_1 \times m_{1,0} + v_2 \times m_{2,0}$$

To get the second coordinate, line up the vector with the second column of the matrix, and do the same thing:

$$v_0 \times m_{0,1} + v_1 \times m_{1,1} + v_2 \times m_{2,1}$$

Similarly, the third coordinate is

$$v_0 \times m_{0,2} + v_1 \times m_{1,2} + v_2 \times m_{2,2}$$

Or, shown all at once, we have:

$$\begin{pmatrix} v_0 & v_1 & v_2 \end{pmatrix} \times \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{pmatrix} = \begin{pmatrix} v_0 \times m_{0,0} + v_1 \times m_{1,0} + v_2 \times m_{2,0} \\ v_0 \times m_{0,1} + v_1 \times m_{1,1} + v_2 \times m_{2,1} \\ v_0 \times m_{0,2} + v_1 \times m_{1,2} + v_2 \times m_{2,2} \end{pmatrix}$$

We can perform any of the operations required by selecting the appropriate matrix. To shrink or expand the figure by a factor p, the matrix is

$$\begin{pmatrix} p & 0 & 0 \\ 0 & p & 0 \\ 0 & 0 & p \end{pmatrix}$$

If p is less than 1 this will shrink the figure; if p is greater than 1 it will expand it.

To rotate a figure by an angle a around the z axis use the matrix

$$\begin{pmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For rotation around the x axis use

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) \\ 0 & \sin(a) & \cos(a) \end{pmatrix}$$

For rotation around the y axis use

$$\begin{pmatrix} \cos(a) & 0 & -\sin(a) \\ 0 & 1 & 0 \\ \sin(a) & 0 & \cos(a) \end{pmatrix}$$

Note: Rotation will produce unexpected (though correct) results if the figure is not centered at the origin (0, 0, 0): the figure will move when you rotate it. Similarly if you expand a figure that is not centered at or near the origin it will move farther from the origin as well as getting bigger.

Now that you know how to manipulate the points of a figure, how do you get from there to drawing the figure as a whole? First, you need to know how to draw a three dimensional point on the two dimensional screen. The operation required is called a projection. There are many different projections possible. I suggest you use the simplest one, which is called orthogonal projection. All you do is plot a point using just its x and y coordinates and ignoring its z coordinate. To plot a line, you simply connect the projections of the two points that define the line. To plot a wire frame figure, you plot the lines corresponding to the edges of the figure.

3. Graphics hints

You are already familiar with the graphics libraries. For the current assignment, you will need to know about `moveto` and `lineto`, or possibly `line`, which we did in a class demo (see `pong1_multimike.cxx`).

Suppose you use `initwindow` to open a 500-by-500 pixel window. The pixel (screen) coordinates are somewhat awkward for this assignment. The point (0,0) is at the upper-left corner of the screen, and the y coordinate increases down, not up, the screen. A reasonable thing to do about this is to lay out your own more sensible (or natural) coordinate system, with (0,0) in the middle of the screen and y increasing up the screen and with a smaller range, something like –3 to 3 for the x coordinate instead of 0 to 499, and 3 to –3 for the y coordinate instead of 0 to 499. Let's call these the *sensible* coordinates.

To draw lines in terms of the sensible coordinate system, you need to write functions that convert from the sensible coordinate system into the pixel coordinates. It might be a good idea to write a function to do this conversion.

To transform a point (x_s, y_s) from the sensible coordinate system—where the upper left corner is $(-m, n)$ and the lower right corner is $(m, -n)$—to a point (x_p, y_p) in a pixel coordinate system—where the upper left is $(0, 0)$ to and the lower right is (a, b)—use the following: $x_p = \frac{a(x_s+m)}{2m}$ and $y_p = b - \frac{b(y_s+n)}{2n}$.

When you define your cube in sensible coordinates, be sure to make the center of the cube at (0, 0, 0). Otherwise your cube will swing around in a funny way as you rotate it, because your rotations are going to turn the cube around the origin instead of around the cube's center. For example, you might want to define the cube so that all coordinates of the corners are –1 or +1, as in (–/+1, –/+1, –/+1).

4. Four phase program implementation

It is daunting to face a large, unknown project. For this reason, I'm suggesting that you implement your program in four phases. Bring your work to a course staff member when you finish each phase to verify that you're on the right track. If you choose not to do this and you get in trouble, our sympathies will not be with you. If you wait until the last minute to start this program, you will also lose our sympathy. This program will require steady work over the three weeks available.

Try to use the suggestions given in the handout called "Program Design." Also, do a self-check using the check-list of that handout before coming to us for evaluation.

Phase 1: Specify several details of the program: (a) the organization of the input file containing the definition of the object, (b) the data structures you will use for representing the initial wire-frame object (i.e., the array declarations that will hold edges and/or vertices), (c) some general notion of how the user will interact with the program, and (d) pseudocode for major functions in your program, including the functions that draw the wire frame object and read the description of the object from a file.

Phase 2: Implement a version of the program that reads a description of the wire-frame object from a file and displays the object. If you are going to do allow the object to be specified in sensible coordinates, you will also need to write the code that performs the transformation from sensible coordinates to screen coordinates.

Phase 3: Augment the program from phase 2 to perform some fixed transformation, e.g., shrink the object or rotate it, and display the transformed object. One way to do this is to implement a general purpose function for multiplying vectors by 3×3 matrices. For this phase, you will not need to worry about the user interface, since the program will transform the object in a fixed way without any input from the user. You may wish to use the `setcolor` function to display the transformed object in a different color.

Phase 4: Implement the user interface, which will allow the user to manipulate the object by typing various commands.

5. Submitting the assignment

Your program is due by 9:00 p.m. on the due date. It must be submitted via webct. Late assignments will not be accepted. There will be no extensions even if the class web pages go down shortly before the deadline, so print out a hard copy of the assignment. In recitation on the day following the deadline, bring your code (via email or floppy or CDrom) to recitation and you will be asked to demonstrate it for the TA. Part of your grade will depend on this demonstration. The TA may also inspect your code and ask you to make small changes in the functionality of the program to verify that you indeed understand the code.