Program 9 3D Computer Animation

Write a program that generates an animated scene showing a three-dimensional wireframe object moving through space.

This breaks down into three main tasks: building a model of the object (a set of points connected by lines); designing an animation sequence consisting of translation, scaling, and rotation operations; and implementing the basic graphics routines required to transform the object as specified by the animation sequence.

Use `/u/cs126/progs/animation.h` and `animation.c` to draw the object. These routines can be used to draw lines from a viewpoint be at (0,0,0) in a "lefthanded" coordinate system, looking in the Z direction (the X axis is to the right, and the Y axis is up). For your object to be visible, it must be in the positive Z direction and with |X| and |Y| coordinates less than Z. It is OK for some coordinates to sometimes go outside these limits but they will be "offscreen" when they do.

The animation is a sequence of frames where you draw all the lines in the object, moving it slightly for each frame. The basic structure of your program is simple: call `printprologue` at the beginning, `printepilog` at the end, `newframe` every time you want to "move" the object (redraw it in a slightly different position), and `drawline` for every line in the object, in every frame. Between frames, you make calculations to figure out precisely how the object is to be moved, as described below.

To move the object, you should maintain a translation matrix, a rotation matrix, and a scaling matrix. A single transformation matrix is computed for each frame by concatenating these, then the position of every line in the object can be computed by multiplying the coordinate vector for the endpoints by the transformation matrix.

Summary of graphics primitives: It is convenient to use “homogeneous coordinates” with 4 coordinates for 3D space, and 4x4 matrices. One data structure is defined for you: — struct Point3dPlus double x; double y; double z; double w; — You should always initialize the w coordinate to 1.0. The following transformation matrices are used for translation, scaling and rotation about the X axis:

\[
T = \begin{pmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & t_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\quad
S = \begin{pmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & s_z & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\quad
R_x(\theta) = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & \sin \theta & 0 \\
0 & -\sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

The corresponding rotation matrix for Y is obtained by exchanging the first two rows and columns of \(R_x(\theta)\); for Z by exchanging the first and third rows.
and columns. To translate a point from \((0, 0, 0, 1)\) to \((t_x, t_y, t_z, 1)\), multiply it by the matrix \(T(t_x, t_y, t_z)\) and to scale a point by the scaling factors \((s_x, s_y, s_z)\), multiply it by the matrix \(S(s_x, s_y, s_z)\). Similarly, multiplying a point by one of the rotation matrices gives the coordinates of the point after the rotation. For example, a ninety degree rotation about the \(x\) axis corresponds to using \(R_x(\pi/2)\), which takes \((x, y, z, 1)\) into \((x, z, -y, 1)\) since \(\sin(\pi/2) = 1\) and \(\cos(\pi/2) = 0\).

To effect the animation, you will maintain the five matrices above to represent the current “state” of the position and orientation of the object. For each frame, your program will update one or more of the matrices and compute the transformation matrix by multiplying the five matrices together:

\[
\begin{pmatrix}
M & M & M & M \\
M & M & M & M \\
M & M & M & M \\
0 & 0 & 0 & 1
\end{pmatrix} =
\begin{pmatrix}
1 & 0 & 0 & T \\
0 & 1 & 0 & T \\
0 & 0 & 1 & T \\
0 & 0 & 0 & 1
\end{pmatrix}\begin{pmatrix}
R & R & R & 0 \\
R & R & R & 0 \\
R & R & R & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}\begin{pmatrix}
S & 0 & 0 & 0 \\
0 & S & 0 & 0 \\
0 & 0 & S & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Here the middle matrix represents the product of the three rotation matrices. Then each point in the object can be transformed to give the coordinates for its current position by the computation

\[
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix} =
\begin{pmatrix}
M & M & M & M \\
M & M & M & M \\
M & M & M & M \\
0 & 0 & 0 & 1
\end{pmatrix}\begin{pmatrix}
x_0 \\
y_0 \\
z_0 \\
1
\end{pmatrix}
\]

To “draw” the object, do this transformation for the endpoints of each line in the object, then call \texttt{drawline} using the transformed endpoints.

Run \texttt{/u/cs126/progs/animate} to get an idea of what you’re working towards. Start with a relatively simple object, like a cube and with a relatively simple transformation, like spinning around or translating down the \(Z\) axis. Then add more transformations, one at a time, with some idea in your mind about what will happen before running it. Then design a more complex object. Finally, enter into a design cycle of modifying the object and the animation sequence to get the effect that you want.

You may find it convenient to separate out the object design from animation by writing a separate program that generates point coordinates for objects. A few examples of objects may be found in \texttt{/u/cs126/data/obj*}. The format for these is just a sequence of points, with the convention that successive points are to be connected by lines to draw the object. You may wish to choose a more complicated format.
For debugging purposes, you will want to write `printpoint` and `void printmatrix` routines. Use these to make sure that your basic transformations are sound before even looking at a display. The most difficult part of debugging such a program usually is to “find” the initial object, as a bug may place it out of view or too small to be seen. Once you can see the object, you’re off and running toward designing an interesting animation.

There are two options for producing the final output: the default X animation is easiest for debugging, but inconvenient for complex scenes. Running with no arguments produces Postscript output that can be cut and stapled together to produce a “flipbook” that gives an animation by riffling the pages. You may choose to hand in a flipbook or an X animation or both.

**Extra Credit:**

Two extra credit points will be given for creativity above and beyond the basics. Use your imagination, and experiment with different effects. This might involve an interesting object design (model some real object, or have a program generate some mathematical or recursive surface); interesting motion or object distortion, multiple objects; or some other effect (for example, make an object grow or shrink by adding or deleting lines during the animation).

Due: Friday, May 7 (/u/cs126/submit before 11:59PM on 5/6)