Dynamics and Animation

Prerequisites

A knowledge of modeling, viewing, lighting, and on the roles of parameters in setting up modeling and viewing.

Introduction

This is an unusual chapter, because it includes few figures because the topic is motion, and we cannot readily capture motion in a written document. It would be possible to include movie files in a purely electronic document, of course, but these notes are intended for print. Perhaps future versions of this will include inline movies, or at least pre-compiled executables with animation, but for now you must work with the code segments that we provide and see the execution on your own systems.

Computer animation is a very large topic, and there are many books and courses on the subject. We cannot hope to cover the topic in any depth in a first course in computer graphics, and indeed the toolkits needed for a great deal of computer animation are major objects of study and skill development in themselves. Instead we will focus on relatively simple animations that illustrate something about the kind of models and images we have been creating in this course, with an emphasis on scientific areas.

Animation is thought of as presenting a sequence of frames, or individual images, rapidly enough to achieve the sense that the objects in the frames are moving smoothly. There are two kinds of animation — real-time animation, or animation in which each frame is presented by the program while it is running, and frame-at-a-time animation, or animation that is assembled by rendering the individual frames and assembling them into a viewable format (possibly through film or video in a separate production process). This chapter focuses more on frame-at-a-time animation with a goal of achieving real-time animation. The two share the problems of defining how models, lighting, and viewing change over time, but frame-at-a-time animation tends to focus on much more detailed modeling and much more sophisticated rendering while real-time animation tends to focus on simpler time-varying information in order to get refresh rates that are high enough to convey the variation that is desired. While real-time animation may not be as realistic as frame-at-a-time animation because simpler modeling and rendering are used or images may be provided at a slower rate, it can be very effective in conveying an idea and can be especially effective if the user can interact with the animation program as it is running.

As with everything else in this course, the real question is visual communication, and there are some special vocabularies and techniques in using animation for communication. This module does not try to cover this in any depth, but we suggest that you spend some time looking at successful animations and trying to discover for yourself what makes them succeed. To start, we suggest that you focus on clarity and simplicity, and work hard to create a focus on the particular ideas you want to communicate.

Definitions

Animation is the process of creating a sequence of images so that the viewer’s eye will see them as occurring in a smooth motion sequence. The motion sequence can illustrate the relationship between things, can show processes for assembling objects, can allow you to design a sequence of ways to present information, or can allow a user to see a scene from a variety of viewpoints that you can design.
There are many ways to design an animation sequence, but a good place to start is to model your scene using parameters to control features of the model. When you use parameters — variables that you can manipulate in your program — to control positions of objects, positions or properties of lights, shapes or relationships or objects, colors, texture coordinates, or other key points in your model, you can change the values of the parameters with time to change the view you present your audience as the program runs. This allows you to emphasize special features of any of these facets of your model and to communicate those features to your audience.

In defining your modeling in terms of parameters, we need to recall that there are really only three parts to a scene as described by a scene graph. One is the geometry of the scene, where we could use parameters to define the geometry of the scene itself; one example of this could be a function surface, where the function includes a parameter that varies with time, such as \( z = \cos(x^2 + y^2 + t) \). Another is the transformations in the scene, where we could use parameters to define the rotation, translation, or scaling of objects in the scene; one example of this could be moving an object in space by a translation with horizontal component \( t \) and vertical component \( (2-t)^2 \), which would give the object a parabolic path. A third is the appearance of an object in a scene, where a surface might have an alpha color component of \( (1-t) \) to change it from opaque at time 0 to transparent at time 1, allowing the user to see through the surface to whatever lies below it. These are straightforward kinds of applications of parametric modeling and should pose no problem in setting up a model.

One way to design an animation sequence is by explicitly controlling the change in the parameters above through your code. This is called *procedural* animation, and it works well for simple animations where you may have only a few parameters that define the sequence (although what defines “a few” may depend on your system and your goals in the sequence). Another way to design an animation sequence is through key frames, or particular images that you want to appear at particular times in the animation display, and this is called *keyframe* animation. Again, each key frame can be defined in terms of a set of parameters that control the display, but instead of controlling the parameters programmatically, the parameters are interpolated between the values at the key frames.

Probably the simplest approach to animation is to define your entire scene in terms of a single parameter, and to update that parameter each time you generate a new frame. You could think of the parameter as time and think of your animation in terms of a model changing with time. This is probably a natural approach when you are working with scientific problems, where time plays an active role in much of the modeling — think of how much of science deals with change per unit time. If you know how long it will take to generate your scene, you can even change a time parameter by that amount for each frame so that the viewer will see the frames at a rate that approximates the real-time behavior of the system you are modeling.

Another meaning for the parameter could be frame number, the sequence number of the particular image you are computing in the set of frames that will make up the animation sequence. If you are dealing with animation that you will record and playback at a known rate (usually 24 or 30 frames per second) then you can translate the frame number into a time parameter, but the difference in names for the parameter reflects a difference in thinking, because you will not be concerned about how long it takes to generate a frame, simply where the frame is in the sequence you are building.

A key concept in generating animations in real time is the *frame rate* — the rate at which you are able to generate new images in the animation sequence. As we noted above, the frame rate will probably be lower for highly-detailed generated frames than it would be for similar frames that were pre-computed and saved in digital or analog video, but there’s one other difference: frame rates may not be constant for real-time generated images. This points out the challenge of doing your own animations and the need to be sure your animations carry the communication you need.
However, the frame rate can be controlled exactly, no matter how complex the images in the individual frames, if you create your own video “hardcopy” of your animation. See the hardcopy chapter for more details on this.

Keyframe animation

When you do a keyframe animation, you specify certain frames as key frames that the animation must produce and you calculate the rest of the frames so that they move smoothly from one key frame to another. The key frames are specified by frame numbers, so these are the parameter you use, as described above.

In cartoon-type animation, it is common for the key frames to be fully-developed drawings, and for the rest of the frames to be generated by a process called “tweening” — generating the frames between the keys. In that case, there are artists who generate the in-between frames by re-drawing the elements of the key frames as they would appear in the motion between key frames. However, we are creating images by programming, so we must start with models instead of drawings. Our key frames will have whatever parameters are needed to define the images, then, and we will create our in-between frames by interpolating those parameters.

Our animation may be thought of as a collection of animation sequences (in the movies, these are thought of as shots), each of which uses the same basic parts, or components, of objects, transformations, lights, etc. For any sequence, then, we will have the same components and the same set of parameters for these components, and the parameters will vary as we go through the sequence. For the purposes of the animation, the components themselves are not important; we need to focus on the set of parameters and on the ways these parameters are changed. With a keyframe animation, the parameters are set when the key frames are defined, and are interpolated in some fashion in moving between the frames.

As an example of this, consider a model that is defined in a way that allows us to identify the parameters that control its behavior. Let us define ...

In order to discuss the ways we can change the parameters to achieve a smooth motion from one key frame to another, we need to introduce some notation. If we consider the set of parameters as a single vector \( \mathbf{P} = <a, b, c, \ldots, n> \), then we can consider the set of parameters at any given frame \( M \) as \( \mathbf{P}_M = <a_M, b_M, c_M, \ldots, n_M> \). In doing a segment of a keyframe animation sequence starting with frame \( K \) and going to frame \( L \), then, we must interpolate \( \mathbf{P}_K = <a_K, b_K, c_K, \ldots, n_K> \) and \( \mathbf{P}_L = <a_L, b_L, c_L, \ldots, n_L> \), the values of the parameter vectors at these two frames.

A first approach to this interpolation would be to use a linear interpolation of the parameter values. So if we the number of frames between these key frames, including these frames, is \( C = L - K \), we would have \( p_i = (i * (p_K) + (C-i) * p_L) / C \) for each parameter \( p \) and each integer \( i \) between \( L \) and \( K \). If we let \( t = i / C \), we could re-phrase this as \( p_i = (t * (p_K) + (1-t) * p_L) \), a more familiar way to express pure linear interpolation. This is a straightforward calculation and would produce smoothly-changing parameter values which should translate into smooth motion between the key frames.

Key frame motion is probably more complex than this simple first approach would recognize, however. In fact, we not only want smooth motion between two key frames, but we want the motion from before a key frame to blend smoothly with the motion after that key frame. The linear interpolation discussed above will not accomplish that, however; instead, we need to use a more general interpolation approach. One approach is to start the motion from the starting point more...
slowly, move more quickly in the middle, and slow down the ending part of the interpolation so that we stop changing the parameter (and hence stop any motion in the image) just as we reach the final frame of the sequence. In Figure 12.X, we see a comparison of simple linear interpolation on the left with a gradual startup/slowdown interpolation on the right. The right-hand figure shows a sinusoidal curve that we could readily describe by 
\[ t = 0.5(1-\cos(f/\pi)) \]
where in both cases we use 
\[ t = i/C \]
as in the paragraph above, so that we are at frame \(K\) when \(t = 0\) and frame \(L\) when \(t = 1\).

![Figure 12.X: two interpolation curves; linear (left) and sinusoidal (right)](image)

In spite of our adjustment to move through the key frames slowly, we still have the problem that a parameter can provide motion (or another effect) in one direction up to a key frame, and then that motion or effect can go off in an entirely different direction when it leaves the key frame and goes to another one. That is, our motion is not yet smooth as it goes through a key frame. To achieve this, we will need to provide a more sophisticated kind of interpolation.

If we consider the quadratic and cubic interpolations from the mathematical fundamentals in an early chapter, we see that there are interpolations among a number of points that meet some of the points. We need to find such an interpolation that allows our interpolation to meet each keyframe exactly and that moves smoothly among a set of keyframes, and in order to do this we need to be able to interpolate the parameters for the frames in a way that meets exactly the parameters for the keyframes and that moves smoothly between values of the parameters. From the various kinds of interpolations available, we would chose the Catmull-Rom interpolation described in the chapter on spline modeling, which gives us the kind of interpolation shown in the second row of Figure 12.Y and that contrasts with the point-to-point interpolation shown in the first row of that figure.

![Figure 12.Y: moving in and out of a keyframe (left to right follows time). Top row: using a two-point interpolation; bottom row: using a multi-point interpolation](image)
Building an animation

The communication you are developing with your animation is very similar to the communication that a director might want to use in a film. Film has developed a rich vocabulary of techniques that will give particular kinds of information, and animations for scientific communication can benefit from thinking about issues in cinematic terms. If you plan to do this kind of work extensively, you should study the techniques of professional animators. Books on animations will show you many, many more things you can do to improve your planning and execution. Initially, you may want to keep your animations simple and hold a fixed eyepoint and fixed lights, allowing only the parts of the model that move with time to move in your frames. However, just as the cinema discovered the value of a moving camera in a moving scene when they invented the traveling shot, the camera boom, and the hand-held walking camera, you may find that you get the best effects by combining a moving viewpoint with moving objects. Experiment with your model and try out different combinations to see what tells your story best.

A word to the wise...

Designing the communication in an animation is quite different from the same task for a single scene, and it takes some extra experience to get it really right. Among the techniques used in professional animation is the storyboard — a layout of the overall animation that says what will happen when as the program executes, and what each of your shots is intended to communicate to the viewer.

Some examples

Moving objects in your model

Since animation involves motion, one approach to animation is to move individual things in your model. The example in the source file movingcube.c takes a mathematical approach to defining the position of a cube, moving it around in space. As we suggested above, the key to this is the animate() function that we illustrate below:

```c
void animate(void)
{
    #define deltaTime 0.05
    // define position for the cube by modeling time-based behavior
    aTime += deltaTime; if (aTime > 2.0*PI) aTime -= 2.0*PI;
    cubex = sin(2.0*aTime);
    cubey = cos(3.0*aTime);
    cubez = cos(aTime);
    glutPostRedisplay();
}
```

This function sets the values of three variables that are later used with a `glTranslate*`(...) function to position the cube in space. You could set variable orientation, size, or other properties of your objects as well with similar processes.

Moving parts of objects in your model

Just as we moved whole objects above, you could move individual parts of a hierarchial model. You could change the relative positions, relative angles, or relative sizes by using variables when you define your model, and then changing the values of those variables in the idle callback. You can even get more sophisticated and change colors or transparency when those help you tell the
story you are trying to get across with your images. The code below increments the parameter \( t \) and then uses that parameter to define a variable that is, in turn, used to set a rotation angle to wiggle the ears of the rabbit head in the file `rabbitears.c`.

```c
void animate(void)
{
    #define twopi 6.28318
    t += 0.1;
    if (t > twopi) t -= twopi;
    wiggle = cos(t);
    glutPostRedisplay();
}
```

Moving the eye point or the view frame in your model

Another kind of animation is provided by providing a controlled motion around a scene to get a sense of the full model and examine particular parts from particular locations. This motion can be fully scripted or it can be under user control, though of course the latter is more difficult. In this example, the eye moves from in front of a cube to behind a cube, always looking at the center of the cube, but a more complex (and interesting) effect would have been achieved if the eye path were defined through an evaluator with specified control points. This question may be revisited when we look at evaluators and splines.

```c
void display( void )
{
    // Use a variable for the viewpoint, and move it around ...
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt( ep.x, ep.y, ep.z, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    ...
}

void animate(void)
{
    GLfloat numsteps = 100.0, direction[3] = {0.0, 0.0, -20.0};
    if (ep.z < -10.0) whichway = -1.0;
    if (ep.z > 10.0)  whichway = 1.0;
    ep.z += whichway*direction[2]/numsteps;
    glutPostRedisplay();
}
```

As you travel, you need to control not only the position of the eye point, but also the entire viewing environment — in simple terms, the entire `gluLookAt(...)` parameter list. So not only the eye point, but also the view reference point and the up vector must be considered in creating an effective moving viewpoint. Further, as you move around you will sometimes find yourself moving nearer to objects or farther from them. This means you will have the opportunity to use level-of-detail techniques to control how those objects are presented to the viewer while you keep the frame rate as high as possible. There’s a lot of work here to do everything right, but you can make a good start much more easily.

Changing features of your model

There are many other special features of your models and displays that you can change with time to create the particular communication you want. Among them, you can change colors, properties of
your lights, transparency, clipping planes, fog, texture maps, granularity of your model, and so on. Almost anything that can be defined with a variable instead of a constant can be changed by changing the model.

In the particular example for this technique, we will change the size and transparency of the display of one kind of atom in a molecule, as we show in Figure 12.1. The change in the image is driven by a parameter \( t \) that is changed in the \texttt{idle} callback, and the parameter in turn gives a sinusoidal change in the size and transparency parameters for the image. This will allow us to put a visual emphasis on this kind of atom so that a user could see where that kind of atom fits into the molecule. This is just a small start on the overall kinds of things you could choose to animate to put an emphasis on a part of your model.

```c
void molecule(void)
{
    ...
    j = atoms[i].colindex; // index of color for atom i
    for (k=0; k<4; k++)
    { // copy atomColors[j], adjust alpha by alphaMult
        myColor[k] = atomColors[j][k];
    }
    if (j==CARBON) myColor[3] += alphaAdd;
    glMaterialfv(..., myColor);
    glTranslatef(...);
    if (j==CARBON)
        gluSphere(atomSphere,sizeMult*ANGTOAU(atomSizes[j]),GRAIN,GRAIN);
    else
        gluSphere(atomSphere,ANGTOAU(atomSizes[j]),GRAIN,GRAIN);
    glPopMatrix();
    ...
    }
    ...
    void animate(void)
    {
        t += 0.1; if (t > 2.0*M_PI) t -= 2.0*M_PI;
        sizeMult = (1.0+0.5*sin(t));
        alphaAdd = 0.2*cos(t);
        glutPostRedisplay();
    }
```

Figure 12.1: molecule with carbon expanded (left) or contracted (right)
Some points to consider when doing animations with OpenGL

There are some things about OpenGL that you need to understand more fully when you move your eyepoint than you when you simply create a single image. The viewing transformation is part of the overall modeling transformation, and it needs to be done at the right place if you are going to use parameters to define your view. In the display() function in the viewcube.c example, you will note that the modeling transformation is set to the identity, the glutLookAt(...) function is called, the resulting transformation is saved for future use, and then the rotation processes are called. This keeps the viewing transformation from changing the position of objects in the model and thus keeps the animation looking right.

Finally, be careful when you use texture maps in animations. There is always the possibility of aliasing with texture maps, and when they are animated the aliasing can cause strange-looking behavior in the texture rendering. Some effort in antialiasing textures is particularly important in animating them.

Code examples

As we noted above, and as the code examples show, animation control is primarily managed by changing parameters of your scene in the callback for the idle event. We have seen several of these examples to control several aspects of the model, the scene, and the display. You should experiment as widely as you can to see what kind of things you can control and how you can control them in order to become fluent in using animation as a communication tool.

References

You should look at videos of graphics work to get a fuller understanding of what you can do with this tool. In general, though, you want to avoid high-end entertainment-focused animations and look at informational animations — presentations of scientific or technical work are idea. But when you look at video, you are probably looking at presentation-level animations, work that is done to impress others. Most of the work you can do in a first course is more likely personal-level or peer-level animation: work that is done to explore an idea yourself or to share with a friend or colleague. So don’t think you can match the quality of the videos you watch; try to find the key communication ideas and learn from them.