CS 450/550 -- Fall Quarter 2025

Project #4

100 Points

Due: November 3

Keytime Animation

This page was last updated: October 21, 2025

I created a summary document describing the steps in using KeyTime animation. It is called Keytime Steps. Check it out. It might help you!

Introduction

This project is about performing a keytime animation. You will be the Senior Animator and will choose key values to use in your animation. The computer will be the In-Betweener and will fill in interpolated values in-between your key values.

Learning Objective:

When you are done with this assignment, you will understand how to get your scene's objects to move in ways that you get to control, without having to write any equations or do any physics.

Instructions

1. Do a keytime animation of two 3D objects. You can pick what two objects you want to use, but they both need to have surface normals so they can be lit.

2. In this keytime animations, there need to be at least **9** quantities being animated, each with at least **6** keytimes within each animation.

3. The 9 quantities to be animated must be:
   3 quantities that animate the eye position, look-at position, or up-vector used in the gluLookAt( ) call.
   3 quantities that have something to do with translating, rotating, or scaling Object #1
   3 quantities that have something to do with translating, rotating, or scaling Object #2

4. Make the animation last 10 seconds. At time=10 seconds, bring each quantity back to its original time=0. value

5. Use lighting in the scene. The choice of what lighting parameters to use is up to you, but choose them so that it is obvious to us what your objects are doing.

6. The two objects to be displayed are your choice of any 3D geometry. As lighting is required, be sure those objects have surface normal vectors. All of the Osu* objects do. Most .obj files do, but not all.

7. Animate your quantities by smoothly interpolating them between the keytimes. Use the smooth interpolation C++ class that we discussed. For your convenience, the class methods are re-covered below.

Smooth Interpolation:

The smooth interpolation technique that you will be using employs the Coons cubic curve (end points and end slopes). But, rather than you having to specify all of the slopes, the technique makes a reasonable approximation of them based on the surrounding keytime information.

The Keytimes class methods that you will need are:

```
void    Init( );
void    AddTimeValue( float time, float value );
float   GetValue( float time );
```

Set the values like this:

```
// a defined constant:
#define MSEC    10000           // 10000 milliseconds = 10 seconds


// globals:
Keytimes        Xpos1, Xrot1;
GLuint          DL1, DL2;


// in InitGraphics( ):
        Xpos1.Init( );
        Xpos1.AddTimeValue(  0.0,  0.000 );
        Xpos1.AddTimeValue(  0.5,  2.718 );
        Xpos1.AddTimeValue(  2.0,  0.333 );
        Xpos1.AddTimeValue(  5.0,  3.142 );
        Xpos1.AddTimeValue(  8.0,  2.718 );
        Xpos1.AddTimeValue( 10.0,  0.000 );

        Xrot1.Init( );
        Xrot1.AddTimeValue(  0.0,  0.000 );
        . . .
        Xrot1.AddTimeValue( 10.0,  0.000 );

        Xpos2.Init( );
        . . .

. . .


// in InitLists( ):
        DL1 = glGenLists( 1 );
        glNewList( DL1, GL_COMPILE );
```

```
                OsuSphere( 1., 32, 32 );                // for example
        glEndList( );

        DL2 = LoadObjMtlFiles( (char *)"dino.obj" );    // for example

. . .

// in Animate( ):
        glutSetWindow( MainWindow );
        glutPostRedisplay( );

. . .

// in Display( ):
        // turn # msec into the cycle ( 0 - MSEC-1 ):
        int msec = glutGet( GLUT_ELAPSED_TIME )  %  MSEC;

        // turn that into a time in seconds:
        float nowTime = (float)msec  / 1000.;

        glPushMatrix( );
                glTranslatef( Xpos1.GetValue(nowTime),  0., 0. );
                glRotatef(    Xrot1.GetValue(nowTime),  1., 0., 0. );   // angle in degrees
                glCallList( DL1 );
        glPopMatrix( );

        glPushMatrix( );
                . . .
                glCallList( DL2 );
        glPopMatrix( );
```

A Debugging Suggestion:

One thing that has always helped Joe Graphics debug animation programs is to have a "freeze" option, toggled with the '**f**' key. This freezes the animation so you can really look at your object and see if it is really being drawn correctly. Remember what the current freeze status is with a boolean global variable:

```
// in the globals:
bool    Frozen;

. . .

// in Reset( ):
        Frozen = false;

. . .

// in Keyboard( ):
                case 'f':
                case 'F':
                        Frozen = ! Frozen;      // the exclamation point is the boolean "not" operator
                        if( Frozen )
                                glutIdleFunc( NULL );
                        else
                                glutIdleFunc( Animate );
                        break;
```

Turn-in:

Use Canvas to turn in your:

1. Your .cpp file
2. A short PDF report containing:
    - Your name
    - Your email address
    - Project number and title
    - A description of what you did to get the display you got
    - Tell us what your 9 animated quantities were.
    - A table showing your keytime values for each quantity. It is OK just to include the lines of code that set them.
    - A couple of cool-looking screen shots from your program.
    - The link to the Kaltura video demonstrating that your project does what the requirements ask for. If you can, we'd appreciate it if you'd narrate your video so that you can tell us what it is doing.
3. To see how to turn these files in to Canvas, go to our Project Notes noteset, and go the the slide labeled **How to Turn In a Project on Canvas.**
4. **Be sure that your video's permissions are set to *unlisted.***.
   The best place to set this is on the OSU Media Server.
5. A good way to test your video's permissions is to ask a friend to try to open the same video link that you are giving us.
6. The video doesn't have to be made with Kaltura. Any similar tool will do.

Grading:

Item	Points
Animation lasts 10 seconds	10
Lighting works	15
3 eye, look-at, or up-vector gluLookAt animation quantities	25
3 translation, rotation, or scale animation quantities for Object #1	25
3 translation, rotation, or scale animation quantities for Object #2	25
Potential Total	**100**