Simple Keyframe Animation

Approaches to Animation
1. Motion Capture (“MoCap”)
2. Using the laws of physics (we’ll use this in the spring-based motion)
3. Using functional (target-driven) animation (we’ll use this in collision avoidance)
4. Using keyframing

Keyframing

Keyframing involves creating certain key positions for the objects in the scene, and then the program later interpolating the animation frames in between the key frames.

In hand-drawn animation, the key frames are developed by the senior animators, and the in-between frames are developed by the junior animators.

In our case, you are going to be the senior animator, and the computer will do the in-betweening.

But, first we need to look into the mathematics of smooth curves …

Bézier Curves: the Derivation

One parametric line:

Note: we are not actually going to use Bézier curves for the animation, but they are a good place to start to understand how smooth curves work.
Bézier Curves: the Derivation

Three parametric lines, blended:

\[ P(u) = (1-u)^3P_0 + 3(1-u)^2uP_1 + 3(1-u)u^2P_2 + u^3P_3 \]

\[ P(u) = (1-u)^3P_0 + 3(1-u)^2uP_1 + 3(1-u)u^2P_2 + u^3P_3 \]

\[ P(u) = (1-u)^3P_0 + 3(1-u)^2uP_1 + 3(1-u)u^2P_2 + u^3P_3 \]

So How Do Smooth Curves Work in Computer Graphics?

A Small Amount of Input Change Results in a Large Amount of Output Change

The General Form of Cubic Curves

\[ P(t) = At + Bt^1 + Ct^2 + Dt^3 \]

In this form, you need to determine 4 quantities (A, B, C, D) in order to use the equation. That means you have to provide 4 pieces of information. In the Bézier curve, this happens by specifying the 4 points.

Rearranging gives A, B, C, and D for the Bézier curve:

\[ A = P_0 \]
\[ B = -3P_0 + 3P_1 \]
\[ C = 3P_0 - 6P_1 + 3P_2 \]
\[ D = -P_0 + 3P_1 - 3P_2 + P_3 \]

Another Approach: Coons (also called Hermite) Cubic Curves

Another approach to specifying the 4 pieces of information would be to give a start point, an end point, a start parametric slope, and an end parametric slope.

Note the question is: are there other ways to control A, B, C, and D?
Now, Let's Apply this to the Y Translation of a Keyframe Animation

To make this simple to use, our goal is to just specify the keyframe values, not the slopes. We will let the computer compute the slopes for us, which will then result in being able to compute the in-between frames.

Many Professional Animation Packages Make You Sculpt the Slopes (but we won’t . . .)

Blender:

The “Y vs. Frame” Curve Looks Like This

Getting the Two End Slopes

To get the slope at a keyframe point, draw a line between one keyframe back from that one and one keyframe ahead

Do This Same Thing for the X, Y, and Z Translations and the X, Y, and Z Rotations

Instead of Key Frames, I Like Specifying Key Times Better

And, so, I created a C++ class to do it all for you

```cpp
class Keytimes:
    void AddTimeValue(float time, float value);
    float GetFirstTime();
    float GetLastTime();
    int GetNumKeytimes();
    float GetValue(float time);
    void PrintTimeValues();
```

Class Keytimes:

- void AddTimeValue(float time, float value);
- float GetFirstTime();
- float GetLastTime();
- int GetNumKeytimes();
- float GetValue(float time);
- void PrintTimeValues();

Instead of Key Frames, I Like Specifying Key Times Better

```c
int main( int argc, char *argv[] )
{
    Xpos.AddTimeValue( 0.0, 0.000 );
    Xpos.AddTimeValue( 2.0, 0.333 );
    Xpos.AddTimeValue( 1.0, 3.142 );
    fprintf( stderr, "Time runs from %8.3f to %8.3f
", Xpos.GetFirstTime(), Xpos.GetLastTime() );
    for( float t = 0.; t <= 2.01; t += 0.1 )
    {
        float v = Xpos.GetValue( t );
        fprintf( stderr, "%8.3f	%8.3f
", t, v );
    }
}
```

Instead of Key Frames, I Like Specifying Key Times Better

```c
#define MSEC        10000 // i.e., 10 seconds
Keytimes Xpos, Ypos, Zpos;
Keytimes ThetaX, ThetaY, ThetaZ;

if( AnimationIsOn )
{
    // # 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( Xpos.GetValue( nowTime ), Ypos.GetValue( nowTime ), Zpos.GetValue( nowTime ) );
    glRotatef( ThetaX.GetValue( nowTime ), 1., 0., 0. );
    glRotatef( ThetaY.GetValue( nowTime ), 0., 1., 0. );
    glRotatef( ThetaZ.GetValue( nowTime ), 0., 0., 1. );
    /* draw the object */
    glPopMatrix( );
}
```

Using the System Clock in Display() for Timing

```c
#define MSEC 10000 // i.e., 10 seconds
Keytimes Xpos, Ypos, Zpos, Keytimes ThetaX, ThetaY, ThetaZ;

#define AnimationsOn 1
if( AnimationsOn )
{
    // # 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( Xpos.GetValue( nowTime ), Ypos.GetValue( nowTime ), Zpos.GetValue( nowTime ) );
    glRotatef( ThetaX.GetValue( nowTime ), 1., 0., 0. );
    glRotatef( ThetaY.GetValue( nowTime ), 0., 1., 0. );
    glRotatef( ThetaZ.GetValue( nowTime ), 0., 0., 1. );
    /* draw the object */
    glPopMatrix( );
}
```