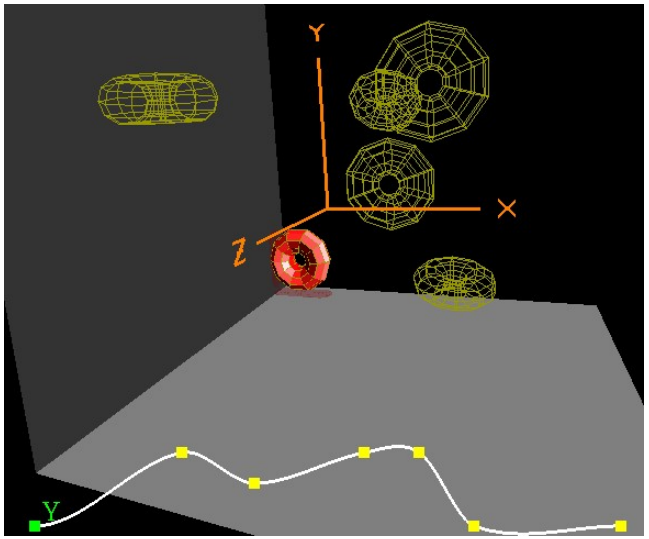
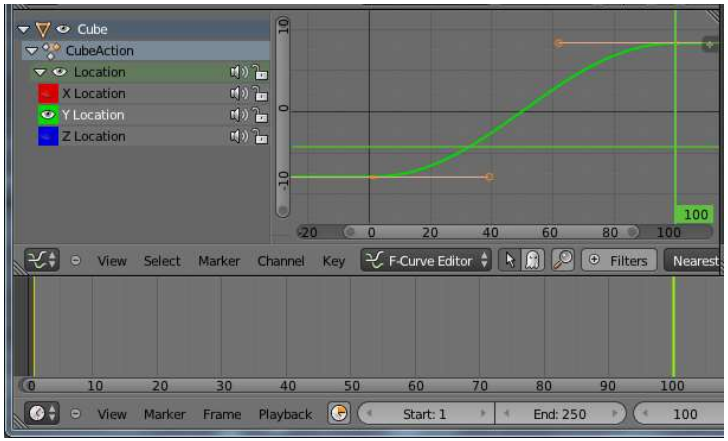
Simple Keyframe Animation



Oregon State University

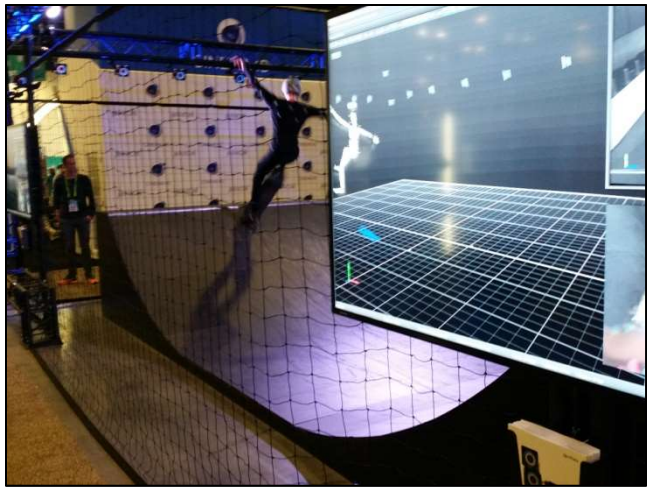
Mike Bailey

mjb@cs.oregonstate.edu



Oregon State University
Computer Graphics

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

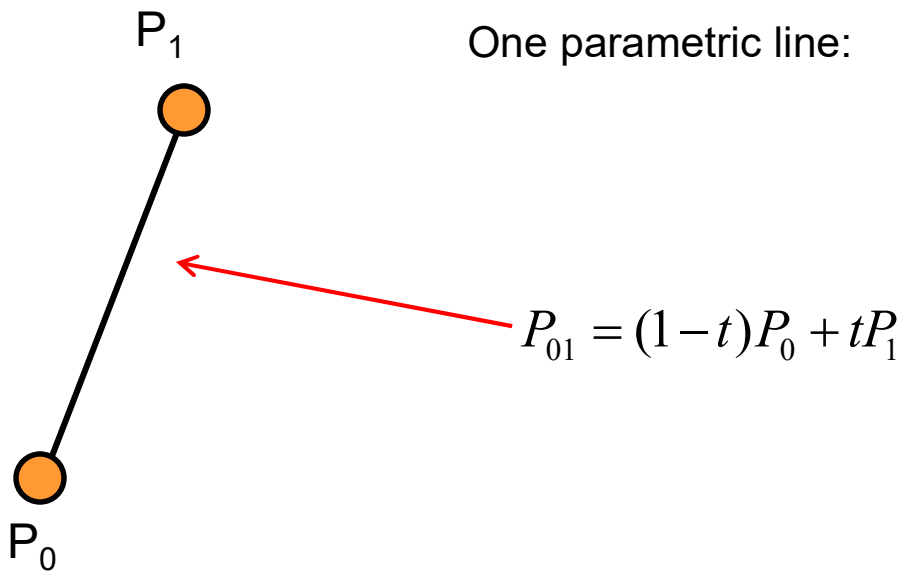
Keyframing involves creating certain *key* positions for the objects in the scene, and then the program later interpolating the animation frames *in between* the key frames.

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



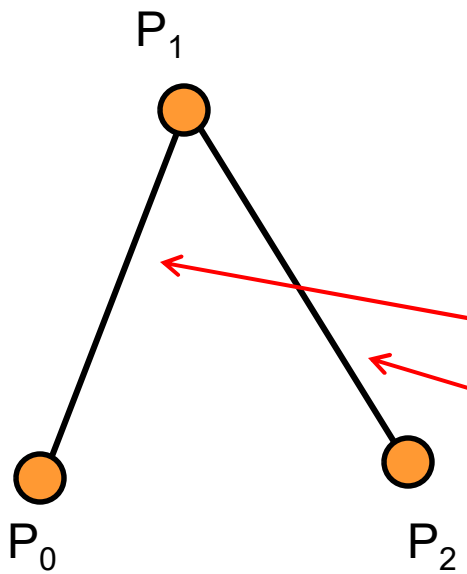
One parametric line:

$$P_{01} = (1-t)P_0 + tP_1$$

1 1

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



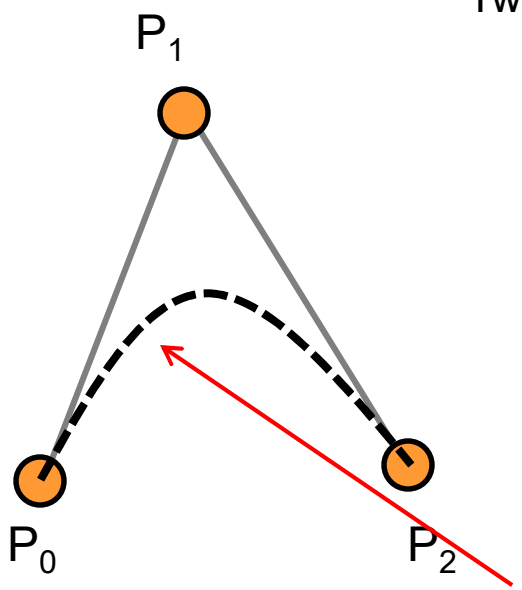
Two parametric lines:

$$P_{01} = (1-t)P_0 + tP_1$$

$$P_{12} = (1-t)P_1 + tP_2$$

1 1

Bézier Curves: the Derivation



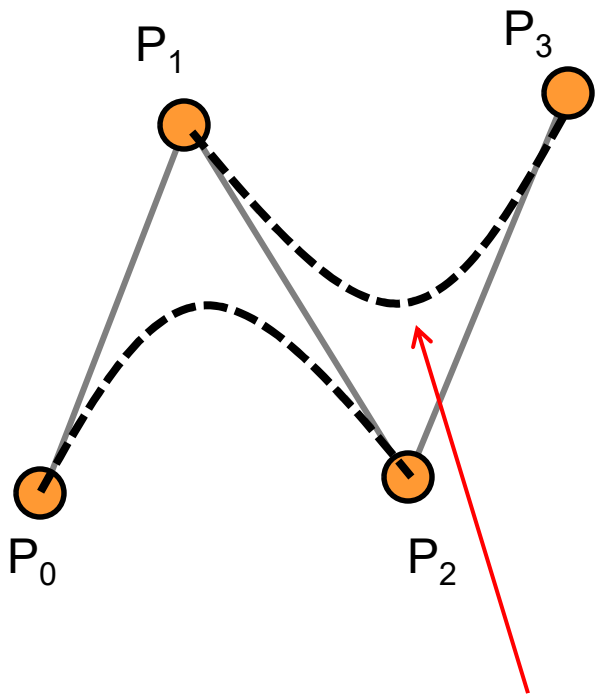
Two parametric lines, blended:

$$P_{01} = (1-t)P_0 + tP_1$$
$$P_{12} = (1-t)P_1 + tP_2$$

$$P_{012} = (1-t)P_{01} + tP_{12}$$
$$= (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

1 1
1 2 1

Bézier Curves: the Derivation



Three parametric lines, blended:

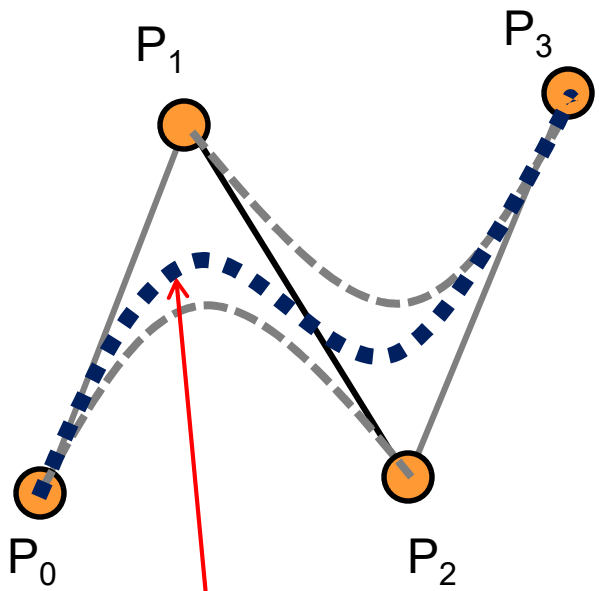
$$P_{01} = (1-t)P_0 + tP_1$$
$$P_{12} = (1-t)P_1 + tP_2$$

$$P_{012} = (1-t)P_{01} + tP_{12}$$
$$= (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

$$P_{123} = (1-t)P_{12} + tP_{23}$$
$$= (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3$$

1 1
1 2 1

Bézier Curves: the Derivation



Three parametric lines, blended:

$$P_{01} = (1-t)P_0 + tP_1$$
$$P_{12} = (1-t)P_1 + tP_2$$

$$P_{012} = (1-t)P_{01} + tP_{12}$$
$$= (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

$$\begin{array}{ccccccc} & & 1 & & 1 & & \\ & 1 & & 2 & & 1 & \\ 1 & & 3 & & 3 & & 1 \end{array}$$

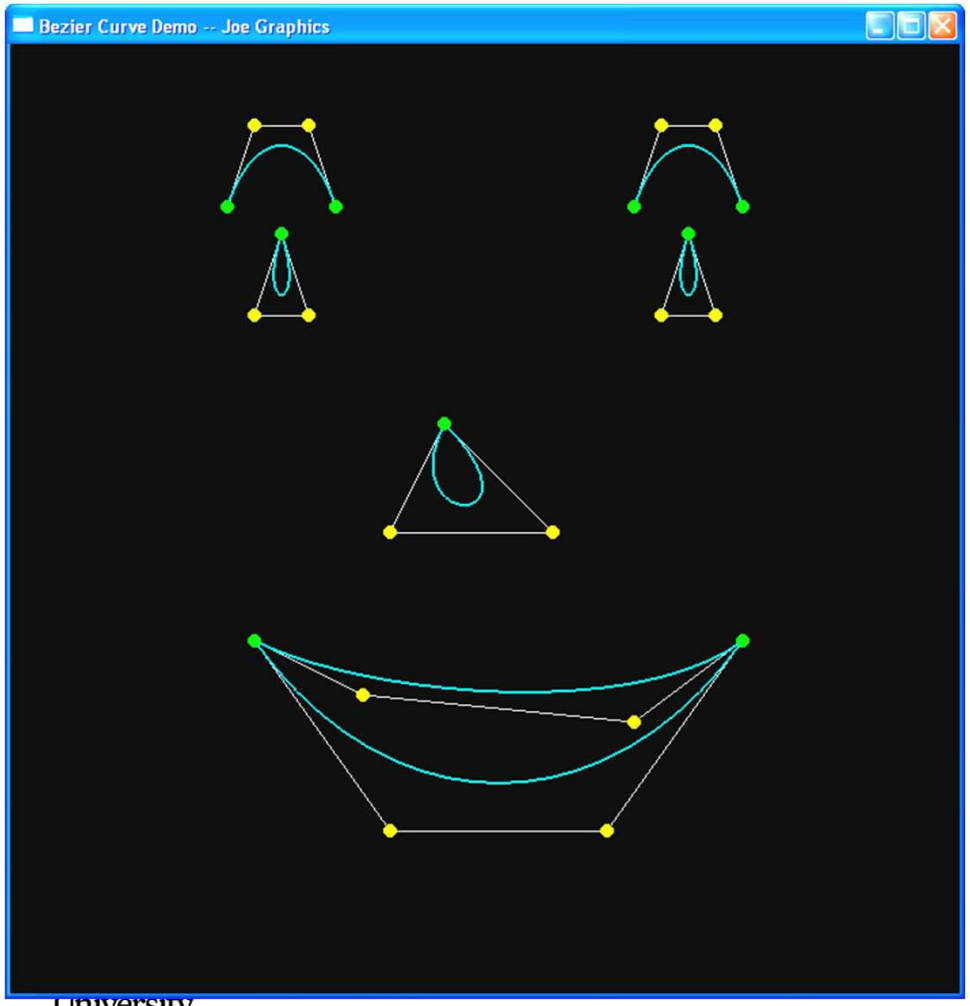
$$P_{123} = (1-t)P_{12} + tP_{23}$$
$$= (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3$$

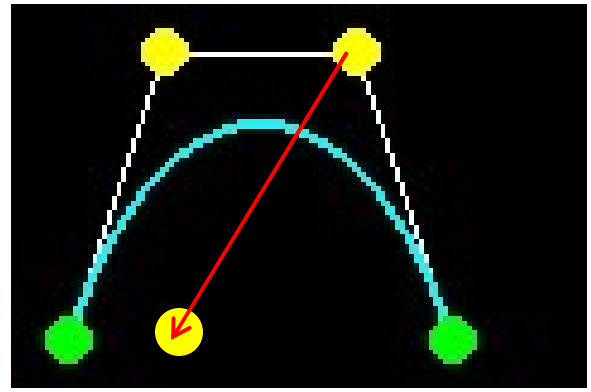
$$P_{0123} = (1-t)P_{012} + tP_{123}$$
$$= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

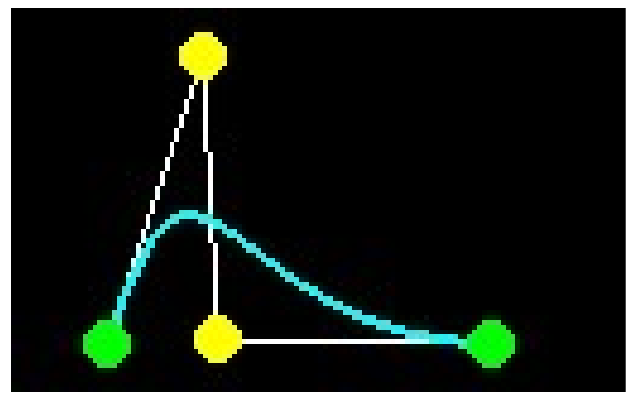
Bézier Curves: Drawing and Sculpting

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

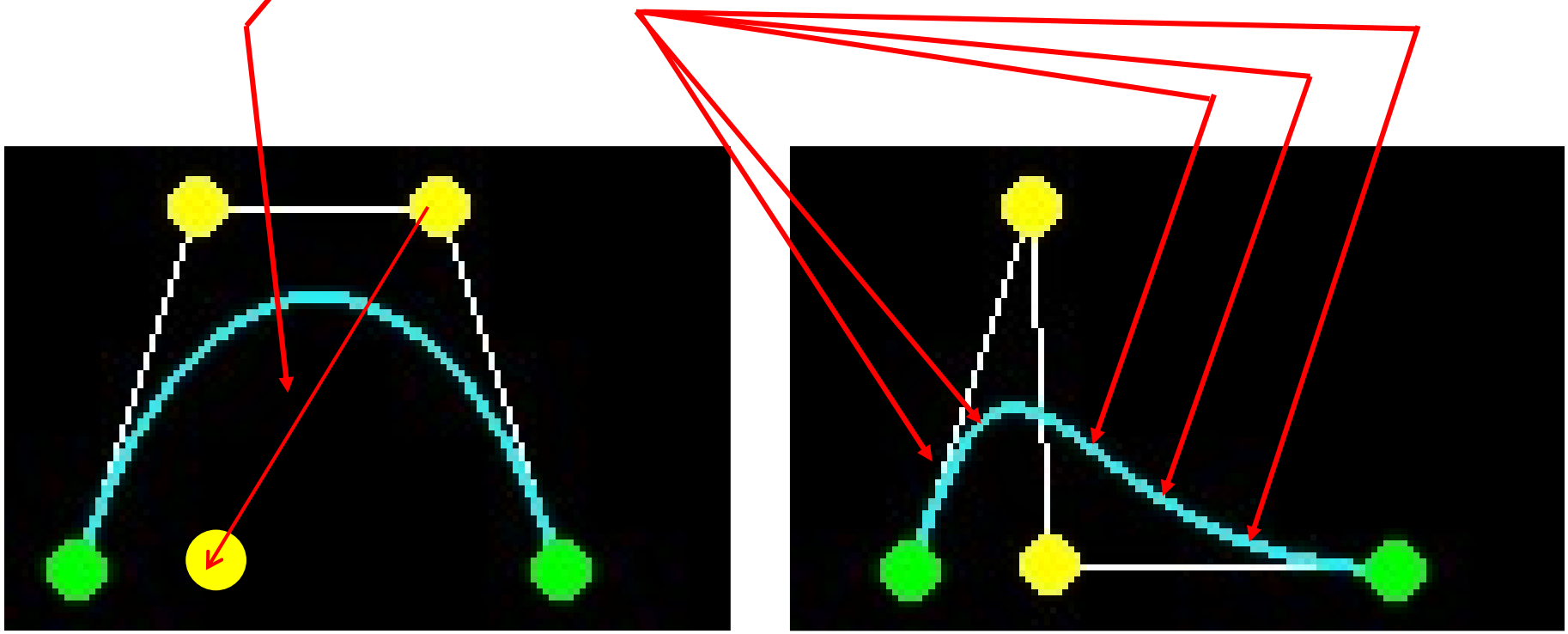
t = 0., .02, .04, .06, …, .98, 1.0







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

$$P(t) = A + Bt + Ct^2 + Dt^3$$

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.

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

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

$$A = P_0$$

$$B = -3P_0 + 3P_1$$

$$C = 3P_0 - 6P_1 + 3P_2$$

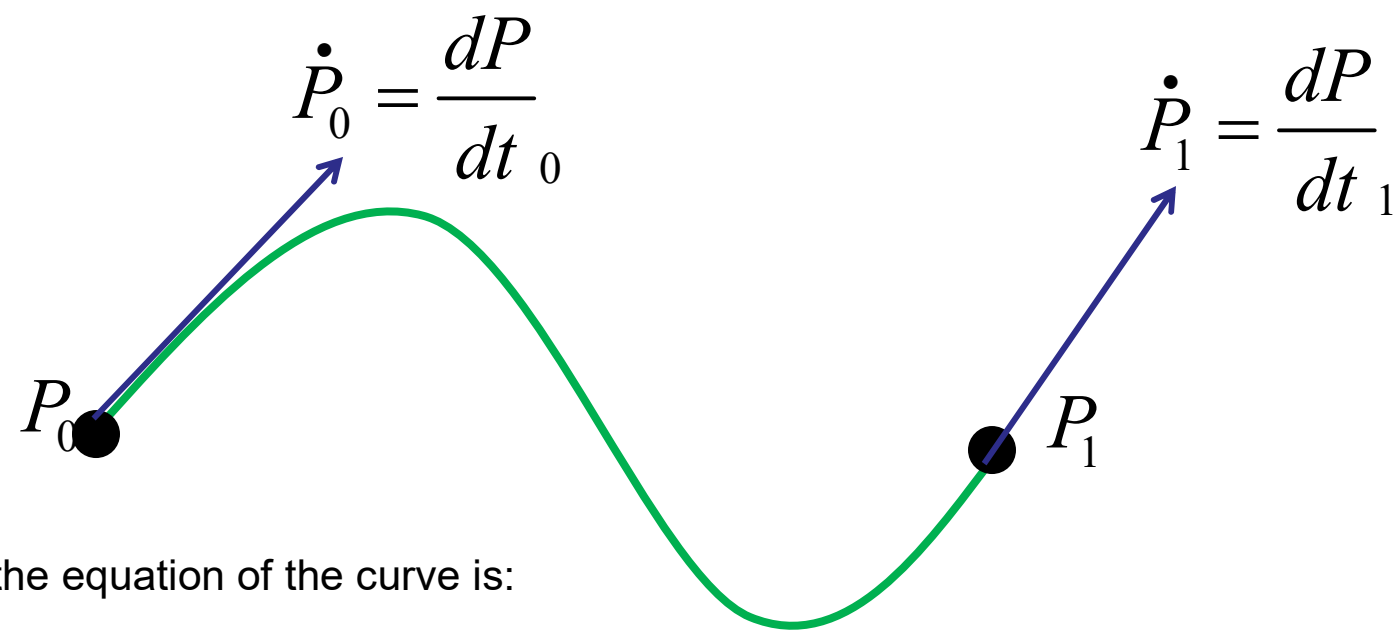
$$D = -P_0 + 3P_1 - 3P_2 + P_3$$

Oregon State University
Computer Graphics

Now the question is: are there other ways to control A, B, C, and D?

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.



If we do this, then the equation of the curve is:

$$P = A + Bt + Ct^2 + Dt^3$$

where:

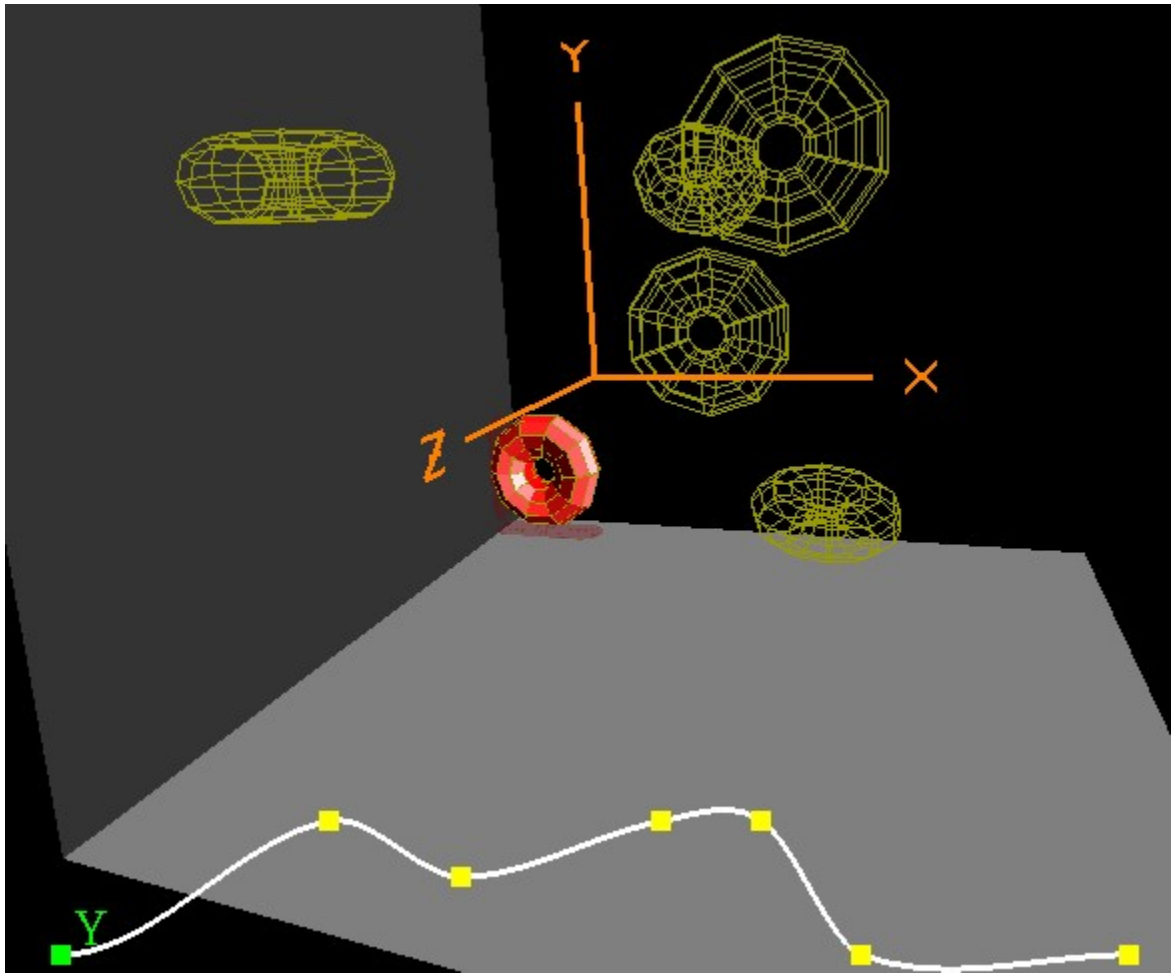
$$A = P_0$$

$$B = \dot{P}_0$$

$$C = -3P_0 + 3P_1 - 2\dot{P}_0 - \dot{P}_1$$

$$D = 2P_0 - 2P_1 + \dot{P}_0 + \dot{P}_1$$

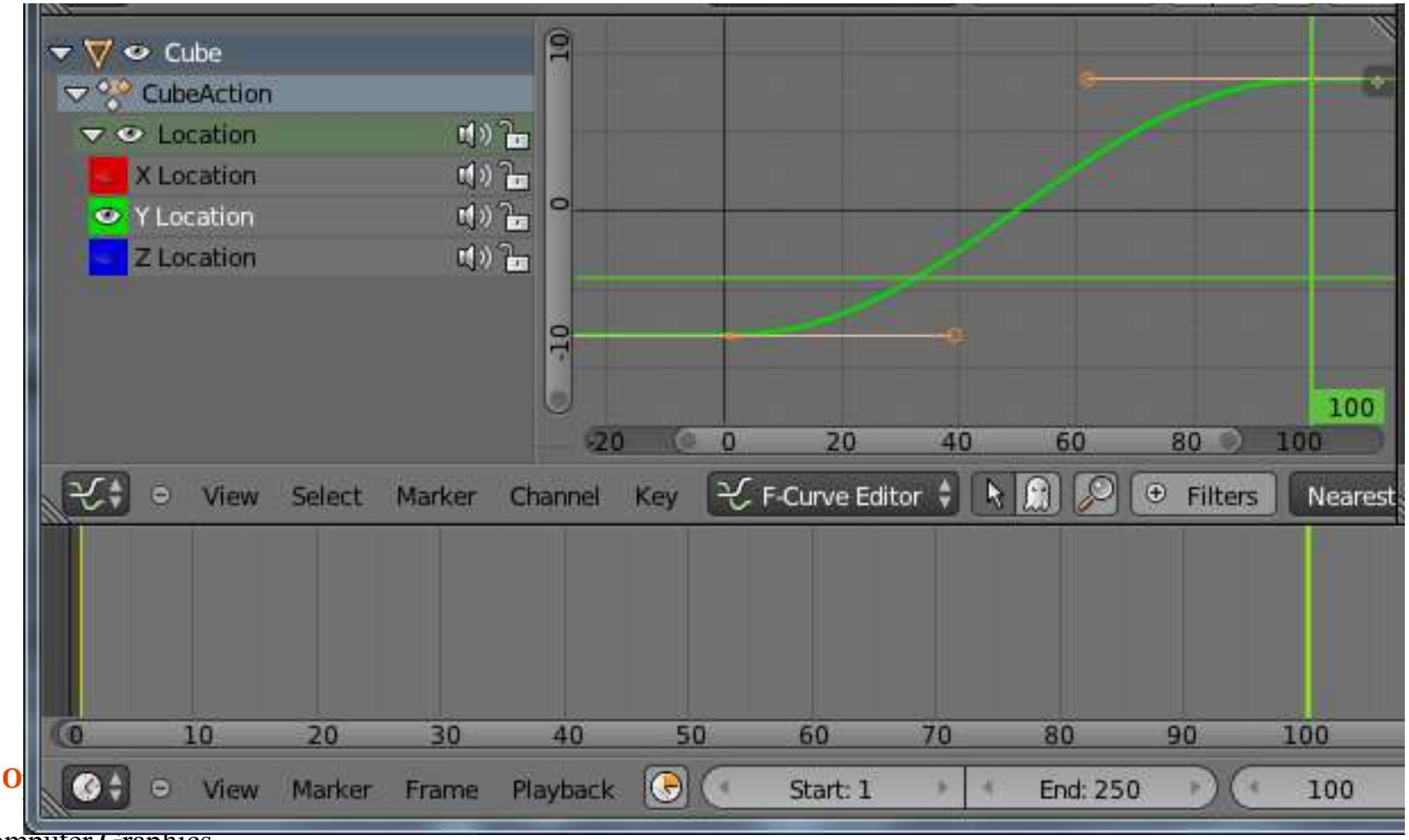
Now, Let's Apply this to the Y Translation of a Keyframe Animation



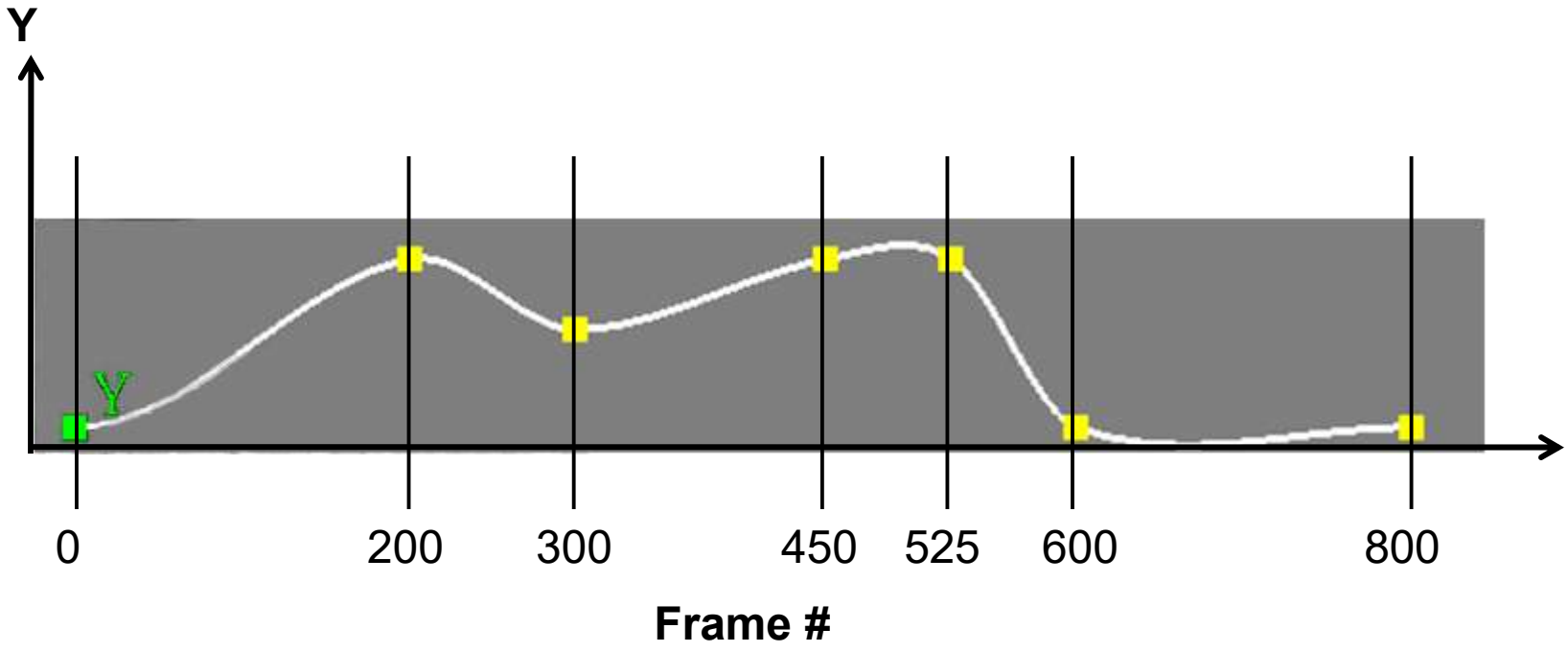
To make this simple to use, our goal is to just specify the keyframe *values*, not the *slopes*. We will let the computer compute the slopes for us, which will then result in being able to compute the in-between frames.

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

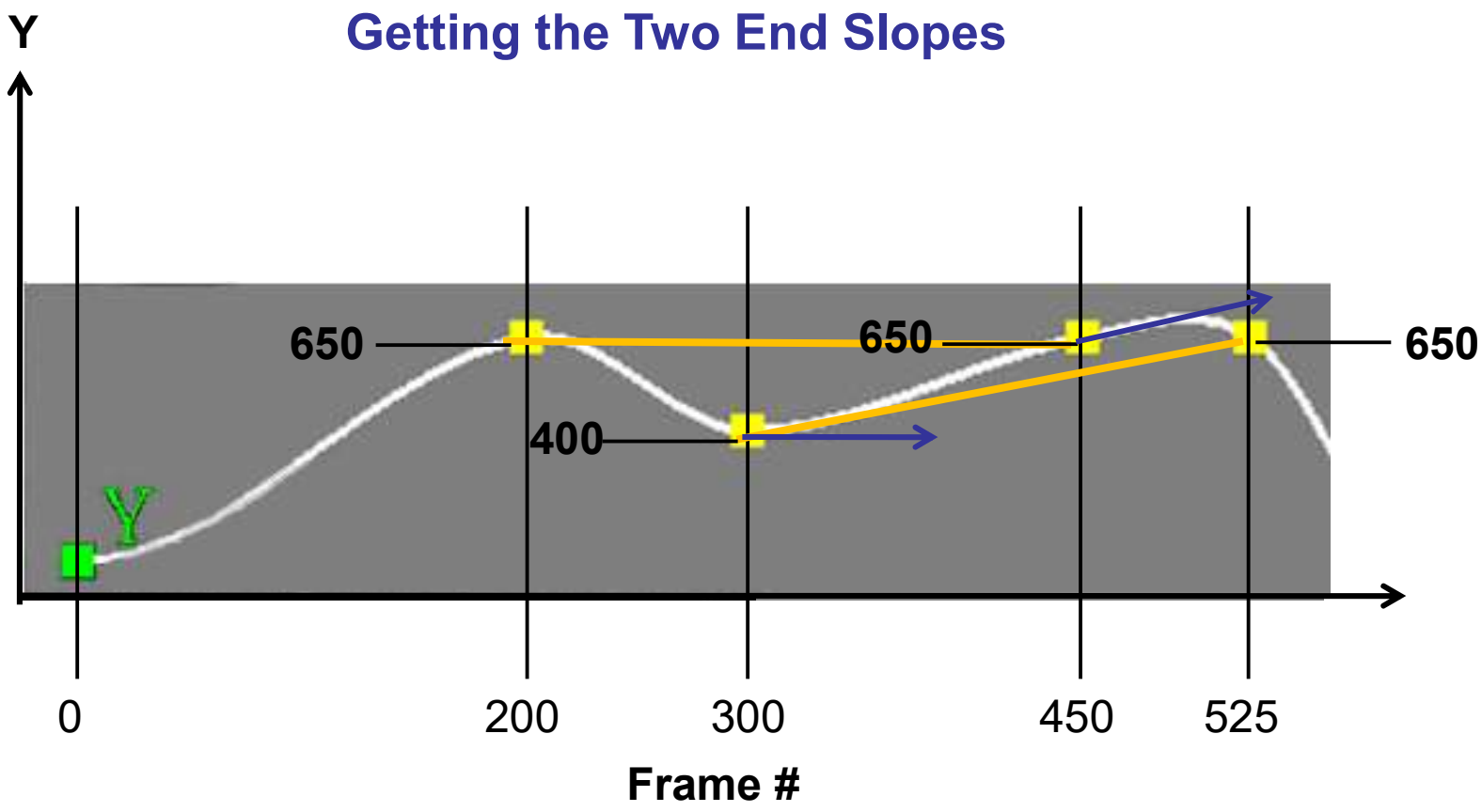
Blender:



The "Y vs. Frame" Curve Looks Like This

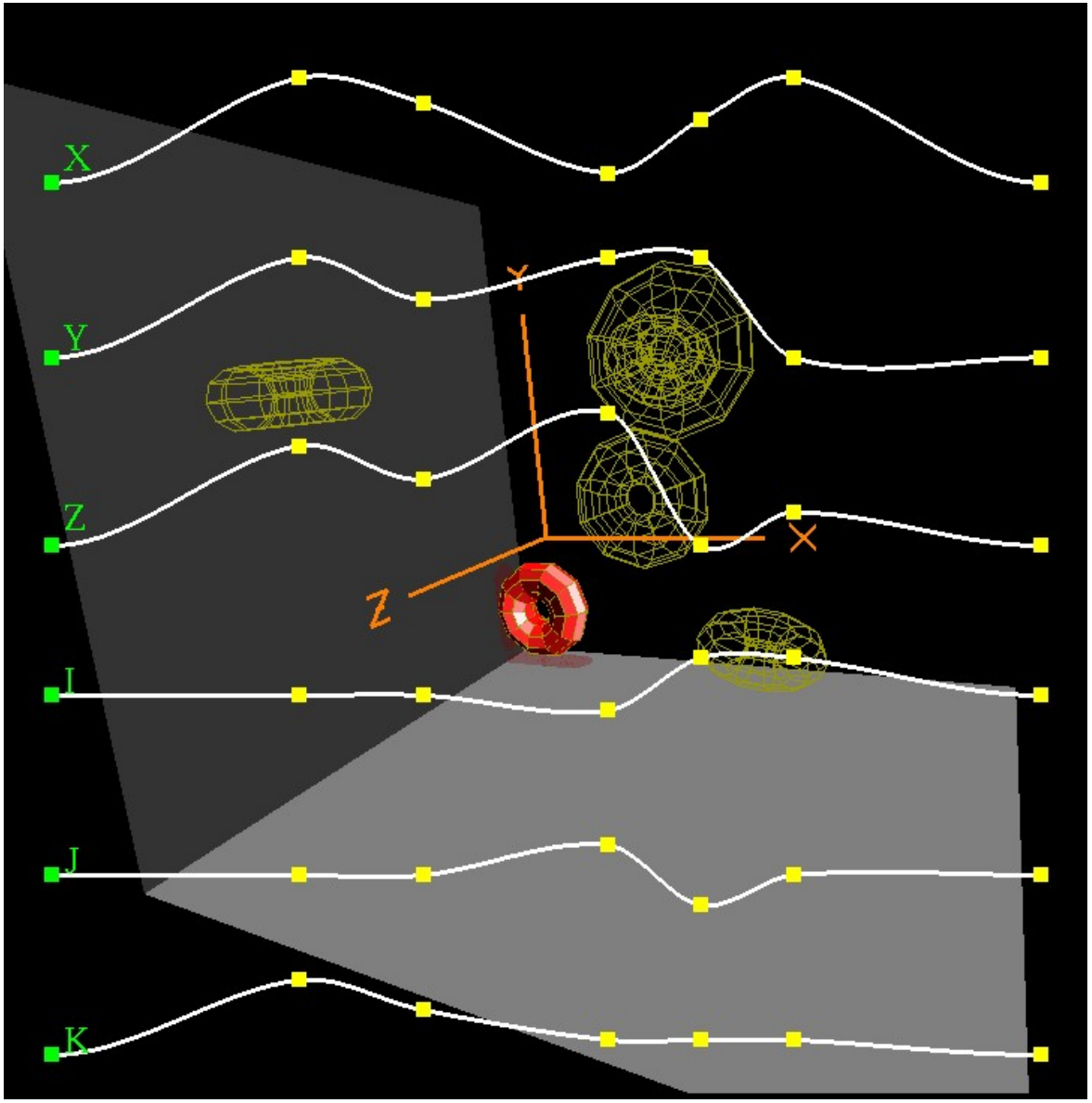


Getting the Two End Slopes



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



X

Y

Z

θ_x

θ_y

θ_z

Instead of Key *Frames*, I Like Specifying Key *Times* Better

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

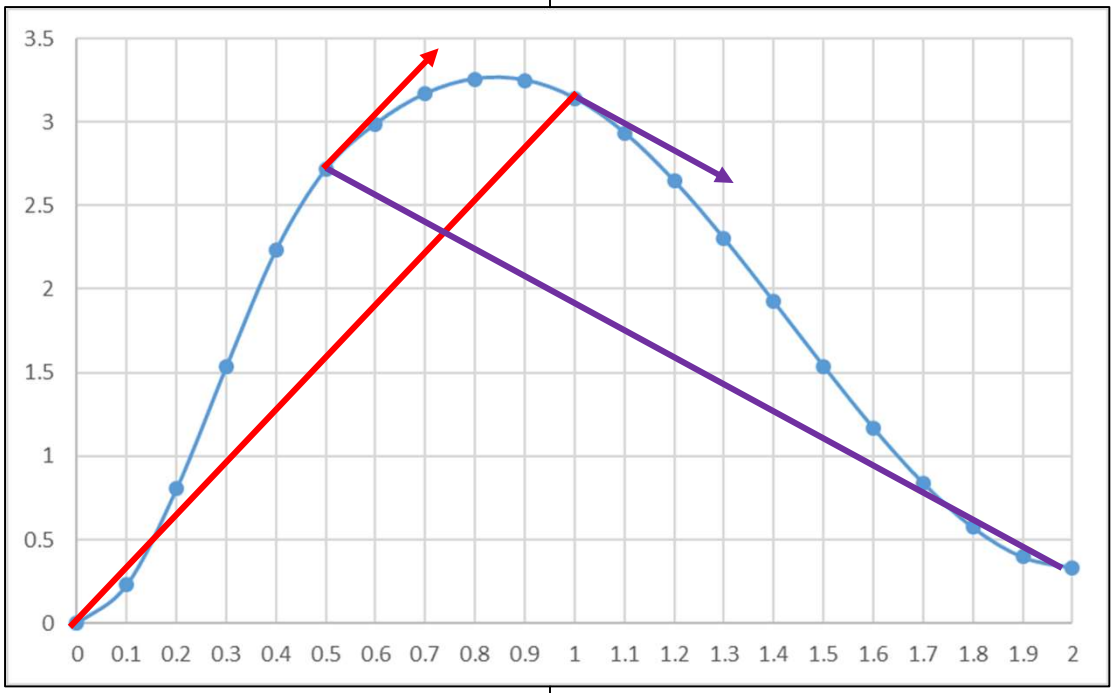
```
class Keytimes:

        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

```
Keytimes Xpos;

int
main( int argc, char *argv[ ] )
{
        Xpos.AddTimeValue(  0.0,  0.000 );
        Xpos.AddTimeValue(  2.0,  0.333 );
        Xpos.AddTimeValue(  1.0,  3.142 );
        Xpos.AddTimeValue(  0.5,  2.718 );
        fprintf( stderr, "%d time-value pairs:\n", Xpos.GetNumKeytimes( ) );
        Xpos.PrintTimeValues( );

        fprintf( stderr, "Time runs from %8.3f to %8.3f\n", Xpos.GetFirstTime( ), Xpos.GetLastTime( ) );

        for( float t = 0.; t <= 2.01; t += 0.1 )
        {
                float v = Xpos.GetValue( t );
                fprintf( stderr, "%8.3f\t%8.3f\n", t, v );
        }
}
```

Instead of Key *Frames*, I Like Specifying Key *Times* Better

```
( 0.00,  0.000)
( 0.00,  0.000)  ( 2.00,  0.333)
( 0.00,  0.000)  ( 1.00,  3.142)  ( 2.00,  0.333)
( 0.00,  0.000)  ( 0.50,  2.718)  ( 1.00,  3.142)  ( 2.00,  0.333)
4 time-value pairs
Time runs from   0.000 to   2.000
  0.000        0.000
  0.100        0.232
  0.200        0.806
  0.300        1.535
  0.400        2.234
  0.500        2.718
  0.600        2.989
  0.700        3.170
  0.800        3.258
  0.900        3.250
  1.000        3.142
  1.100        2.935
  1.200        2.646
  1.300        2.302
  1.400        1.924
  1.500        1.539
  1.600        1.169
  1.700        0.840
  1.800        0.574
  1.900        0.397
  2.000        0.333
```



Using the System Clock in Display( ) for Timing

```
#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( );
}
```

Number of msec in the animation cycle