Animation

Animation is the process of giving motion to your geometric models. Before animating, there are questions you need to ask first:

- Why am I doing this?
- Do I want the animation to obey the real laws of physics? Partially? Which elements?
- Am I willing to "fake" the physics to get the objects to want to move in a way that I tell it?
- Do I have specific key positions I want the objects to pass through no matter what?
- Do I want to simply record the motion of a real person, animal, etc., and then play it back?

Here's Some Code that Lets You Create DIY Keyframe Animations

```cpp
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. And, so, I created a C++ class to do it for you.

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

Here's Some Code that Lets You Create DIY Keyframe Animations

Instead of Key Frames, I like specifying Key Times better. And, so, I created a C++ class to do it 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

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

- the originally-specified key-time pairs

<table>
<thead>
<tr>
<th>Time (frames)</th>
<th>Value (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>(0.00, 0.000)</td>
</tr>
<tr>
<td>0.000</td>
<td>(2.00, 0.333)</td>
</tr>
<tr>
<td>0.000</td>
<td>(1.00, 0.333)</td>
</tr>
<tr>
<td>0.000</td>
<td>(0.00, 2.718)</td>
</tr>
<tr>
<td>0.000</td>
<td>(1.00, 0.142)</td>
</tr>
</tbody>
</table>

Time runs from 0.000 to 2.000

Using the System Clock in Display() for Timing

```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();
}
```

Number of msec in the animation cycle

Forward Kinematics: Transformation Hierarchies

Determine Object Locations?

Inverse Kinematics (IK):

Things Need to Move to a Particular Location – What Parameters Will Make Them Do That?

Of course, there will always be target locations that can never be reached. Think about that spot in the middle of your back that you can never scratch! 😊
Particle Systems: A Cross Between Modeling and Animation?

The basic process is:

- Emit
- Random Number Generator
- Display
- Update

Particle Systems Examples

- The Lion King (2019) -- Disney
A Particle System to Simulate Colliding Galaxies in Cosmic Voyage

Particles Don't Actually Have to Be "Particles"

In order to make this work, you need to supply physical properties such as mass, center of mass, moment of inertia, coefficients of friction, coefficients of restitution, etc.

Newton's second law:

\[ \text{force} = \text{mass} \times \text{acceleration} \]

or

\[ \ddot{x} = \text{acceleration} = \frac{\text{force}}{\text{mass}} \]

\[ x(T) = \int_{t_{\text{init}}}^{t} \ddot{x} \, dt \approx \sum \ddot{x} \Delta t \]

Animating using Rigid-body Physics

In Newtonian terms, the equation for the motion of a body as a function of time is:

\[ x(T) = \int_{t_{\text{init}}}^{t} \ddot{x} \, dt \approx \sum \ddot{x} \Delta t \]

\[ x(T) = \int_{t_{\text{init}}}^{t} \ddot{x} \, dt \approx \sum \ddot{x} \Delta t \]

Animating using Fluid Physics

\[ F = k (D - D_s) \]

This is known as Hooke's Law

Animating using the Physics of a Mesh of Springs

In order to make this work, you need to supply physical properties such as mass, center of mass, moment of inertia, coefficients of friction, coefficients of restitution, etc.

In order to make this work, you need to supply physical properties such as mass, center of mass, moment of inertia, coefficients of friction, coefficients of restitution, etc.
Functional Animation:
Make the Object Want to Move Towards a Goal Position

$$m\ddot{x} + c\dot{x} + kx = 0$$

Functional Animation:
While Making it Want to Move Away from all other Objects

$$mx = \sum F_{\text{repulsive}}$$

Total Goal – Make the Free Body Move Towards its Final Position
While Being Repelled by the Other Bodies

$$m\ddot{x} + c\dot{x} + kx = \sum F$$

Increasing the Stiffness

Stiffness = 3

Stiffness = 6

Increasing the Repulsion Coefficient

Repulse = 10

Repulse = 30
Motion Capture ("MoCap") as an Input for Animation

Natural Pose

Motion Capture is for Faces Too

Even Animals can be MoCapped

My cat would never have put up with this...

https://www.youtube.com/watch?v=zyq_LQrHpoo

Tron I –
They probably should have used physics, but didn’t

Card Trick

Rob Russ