Modeling the World as a Mesh of Springs

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

Mike Bailey
mjb@cs.oregonstate.edu

Weight
Solving for Motion where there is a Spring

\[ F_{\text{spring}} = -k (y - D_h) \]

\[ \Delta V = \sum_{m} \Delta F = \frac{-W - k (y - D_h)}{m} \Delta t \]

Computing Forces in 1D

Generalize by using indices:

\[ F_{i-1} = k (Y_{i-1} - Y_i - D_h) \]

\[ F_{i+1} = k (Y_{i+1} - Y_i - D_h) \]

From the Physics Notes:

What does a Second Order solution look like in a Program?

```cpp
void AdvanceOneTimeStep() {
    GetDerivs(State, Derivatives1);
    State2.t = State.t + \Delta t;
    State2.x = State.x + Derivatives1.vx * \Delta t;
    State2.vx = State.vx + Derivatives1.ax * \Delta t;
    GetDerivs(State2, Derivatives2);
    float aavg = (Derivatives1.ax + Derivatives2.ax) / 2.;
    float vavg = (Derivatives1.vx + Derivatives2.vx) / 2.;
    State.x = State.x + vavg * \Delta t;
    State.vx = State.vx + aavg * \Delta t;
    State.t = State.t + \Delta t;
}
```
Solve for Each State as a Whole, not as Individual Links: Do it This Way

Correct Second Order solution:

```c
for( int i = 0; i < NUMLINKS; i++ )
{
    GetOneBodysDerivs( array, node, float *vxi, float *vyi, float *axi, float *ayi );
}
```

Get all the velocities and accelerations first.

Get all the velocities and accelerations first.

Apply all the velocities and accelerations first.

Apply all the velocities and accelerations first.

Incorrect Second Order solution:

```c
for( int i = 0; i < NUMLINKS; i++ )
{
    GetOneBodysDerivs( links[i], array[i], float *vxi, float *vyi, float *axi, float *ayi );
}
```

Changes the state before we are done getting the derivatives!

Get all the velocities and accelerations first.

Get all the velocities and accelerations first.

Apply all the velocities and accelerations first.

Apply all the velocities and accelerations first.

C Extensions for Array Notion (CEAN) make it look cleaner, and possibly more efficient:

```c
for( int i = 0; i < NUMLINKS; i++ )
{
    TmpLinks[i] = Links[i];
    TmpLinks[i].x = Links[i].x + DT * vx1[i];
    TmpLinks[i].y = Links[i].y + DT * vy1[i];
}
```

```c
for( int i = 0; i < NUMLINKS; i++ )
{
    Links[i] = Links[i] + DT * ( vy1[i] + vy2[i] ) / 2.;
    Links[i].x = Links[i].x + DT * ( vx1[i] + vx2[i] ) / 2.;
```

Don't do it this Way!

```c
if( node < NUMLINKS-1 )
{
    yp = array[node+1].y - array[node].y;
    xp = array[node+1].x - array[node].x;
    length = sqrt( xp*xp + yp*yp );
    yp /= length;
    xp /= length;
    stretch = length - D0;
    force = K * stretch;
}
```

```c
sumfx -= Cd * array[node].vx; // damping
sumfx = Cd * array[node].vx;
sumy = sumy / Mass;
```
Simulating a String

Less Damping

First Order Instability

Placing a Physical Barrier in the Scene

if( DoCircle )
{
  for( int i = 0; i < NUMLINKS; i++ )
  {
    float dx = Links[i].x - CIRCX;
    float dy = Links[i].y - CIRCY;
    float rsqd = dx*dx + dy*dy;
    if( rsqd < CIRCR*CIRCR )
    {
      float r = sqrt( rsqd );
      dx /= r;
      dy /= r;
      Links[i].x = CIRCX + CIRCR * dx;
      Links[i].y = CIRCY + CIRCR * dy;
      Links[i].vx *= dy;
      Links[i].vy *= -dx;
    }
  }
}

Placing a Physical Barrier in the Scene

Vector from circle center to the lumped mass

If the lumped mass is inside the circle

Unit vector from circle center to the lumped mass.

dx = x0-x

dy = y0-y

Push the lumped mass from inside the circle to the circle's surface

Keep just the tangential velocity

Modeling Cloth

• • •
Modeling Cloth

Cloth Example

Cloth Examples

Cloth Example

Cloth Example with Blender
We Can Also use this Same Method to Model and Analyze Rigid Objects

A Bridge on Top of Loose Soil