Week 2 : Dynamics & Numerical Methods

Goal : To write a simple physics simulation

Topics: Intro to ODEs, numerical methods for solving differential equations (using Euler Step, Runge-Kutta, Verlet Integration), 1D spring simulation, Particle Systems, cloth simulation, water simulation, spring-graph layout.

High level view

```
            put stuff on screen
           |                   |
via CPU (slower) ----------- via GPU (faster)
                                  |
             APIs (fixed)    Shaders (custom)
                 |              |
     DirectX OpenGL         CG GLSL HLSL

      C C++ C# ObjC Java Ruby Python Lua Scala, etc

Real-Time Frameworks: Processing, GameX, LuaAV,
  openFrameworks, Jitter, Unreal Engine, Ogre3D, etc

Other Applications: Maya, Adobe, Blender, etc, etc
```

super-simple example code

using Java bindings to openGL - The “windowing” methods are based on GLUT:

“initialize” – set up basic openGL state

“reshape” – set up projection matrix based on window size

“display” – set up modelview matrix and draw geometry

(If you aren’t going to resize the window, you can move the reshape functionality into the initialize method.)

In general, you set up the projection matrix one time (unless you are doing something fancy)

The display method is called every single frame, 60 fps.

In general, you reset the modelview matrix every single frame.

super-simple example code

1. clear the screen
2. reset the modelview matrix

the camera is now at the origin of the the coordinate system, so we move the coordinate system away from the camera to simulate camera movement.

3. position the camera (by moving/rotating the “drawing cursor”)
4. draw geometry, etc. in relation to the “drawing cursor”

repeat 60 times a second.

(show code)

Dynamics

Dynamics is the study of the effects of forces on the motion of objects.

In particular, a dynamic system models the time evolution of a physical process.

The process is generally governed by a set of intertwined equations for the relevant forces, such as gravity, velocity, acceleration, friction, wind resistance, angular momentum, etc.

The time variable is the “independent” variable, and the other variables are “dependent” variables that change with time...

Simple Examples

Dropping a rock off of a cliff:

need to model gravity, center of balance, air resistance, wind velocity

Pushing an object along a surface:

gravity, friction of object, friction of surface, angle of incline of surface, initial force

Firing a cannonball:

initial velocity, gravity, wind resistance, etc

Stretching and then letting go of a spring:

Spring elasticity/stiffness (spring constant), initial displacement position, dampening factor (so it doesn’t spring forever)

Simple Example #1: Gravity

Dropping a rock off of a cliff:

need to model gravity, other forces are less important

Force = mass * acceleration

Gravitational force = (G * m1 * m2) / r^2

Where

G = the gravitational constant,

m1 = the mass of the rock

m2 = the mass of the earth

r = the distance from the center of the rock to the center of the earth

Simple Example #1: Gravity

Then we can solve for the acceleration of the rock:

$F_G = m_1 a = G m_1 m_2 / r^2$

and m_1 cancels out, leaving:

$a = G m_2 / r^2$

Plugging in the gravitational constant and using the average radius of the earth, we get an approximate value for $a = 9.81 m / s^2$

Simple Example #1: Gravity

Knowing the initial position (the height) and the constant acceleration due to the gravitational force, we can use the appropriate kinematic equation to calculate the position of the rock at any time between when we first drop it and when we let it go. We of course have a “boundary condition” where we stop calculating when the rock hits the earth.

$p_t = p_i + v_i - ½(at^2)$ Kinematic equation

For instance, if we simply drop the rock from 100 meters, then after 3 seconds the rock is:

$100 + (0) - .5(9.81 * 3^2) = 55.85500$ meters above the earth.

Simple example #2: Spring

Stretching and then letting go of a spring:

need to model spring elasticity/stiffness (spring constant), initial displacement position, dampening factor (so it doesn't spring forever)

Force = mass * acceleration

Spring force = $-k(p_1 - p_0) - cv$ (Hooke's Law)

Where

k = the stiffness of the spring (based on the structural properties of the spring)

p_0 = the length of the spring at its equilibrium positions (at rest)

p_1 = the length of the spring when being stretched

c = a damping constant which models the reduction of velocity over time

v = the velocity

Simple example #2: Spring

Lets call the stretch distance x, so that our force equation is:

Spring force = -kx – cv

F = ma

a is the second derivative of the position, x

so, ma = -kx – cv

or, a = -k/m * x – c/m * v

Simple example #2: Spring

or, $a = k/m * x – c/m * v$

Now there happens to be an analytical solution to this, but instead we are going to use a numerical method to approximate it. In order to solve a 2nd order differential equation we need to split it into a series of 1st order equations

$dy/dt = v$

$dv/dt = -(k/m) * y - (c/m) * v = a$

That is, the change in position is based on the velocity, and the velocity is based on the acceleration.

Differential Equations

The spring system is what's called an Ordinary Differential Equation, or ODE.

Differential equations calculate the change in some quantity in relationship to another quantity.

In this case we are calculating the change in position with respect to time as well as the change in velocity with respect to time.

Differential Equations

How can we calculate the position of the spring?

Start with an initial condition for the position: eg, y = -10

Use our equations to determine the position after a small increment of time.

The change in position is based on the velocity,
and the velocity is based on the acceleration.

So we need to solve for the acceleration and the velocity "simultaneously",
that is, before we can update the position.

Euler's Step

$y_{n+1} = y_n + hf(t_n, y_n)$

Where

y_n is our current position

h is our time step

and $f(t_n, y_n)$ is our system of differential equations which solve the spring velocity for a particular y position and a particular time.

(draw example on board)

Euler’s Step

Has problems

- Expects the derivative at the current point is a good estimate of the derivative on the interval
- Approximation can drift off the actual function – adds energy to system!
- Worse farther from known values
- Especially bad when the system oscillates (springs, orbits, pendulums or when the time step gets large.

- Lousy for forces dependant on position
- Okay for forces dependant on velocity
- Bad for constant forces

Runge-Kutta methods

idea is to take samples from nearby the original points (t, y) and average the velocity derivates together.

RK2 takes 2 samples... RK4 takes 4 samples...

taking more samples doesn't significantly increase accuracy (at least not for our spring simulation)

RK4:

$y_{n+1} = y_n + 1/6(k_1 + 2k_2 + 2k_3 + k_4)$

Where:

$k_1 = hf(t_n, y_n)$

$k_2 = hf(t_n + h/2, y_n + k_1/2)$

$k_3 = hf(t_n + h/2, y_n + k_2/2)$

$k_4 = hf(t_n + h, y_n + k_3)$

(draw on board)

Runge-Kutta methods

RK4 works better for larger time steps

Tends to dampen energy

Expensive – requires many evaluations per time step

- Okay for forces dependant on position
- Okay for forces dependant on velocity
- Great for constant forces

Verlet Integration

Given y_n and v_n

First calculate

$y_1 = y_0 + hv_0$ (Euler step)

thereafter, use

$$y_{n+1} = 2y_n - y_{n-1} + h^2/m\, f(t_n, y_n)$$

Where:

m = mass of the object

note: verlet equation doesn't require keeping track of the velocity

Verlet Integration

Velocity-less scheme originally developed by Newton and first used in molecular dynamics

Uses position from previous time step

Very stable, but velocity estimated so potentially inaccurate

Good for particle systems

- Better for forces dependant on position (particles)
- Okay for forces dependant on velocity (friction, springs, etc)

Comparison

see demos for comparison of numerical methods...

code examples online...