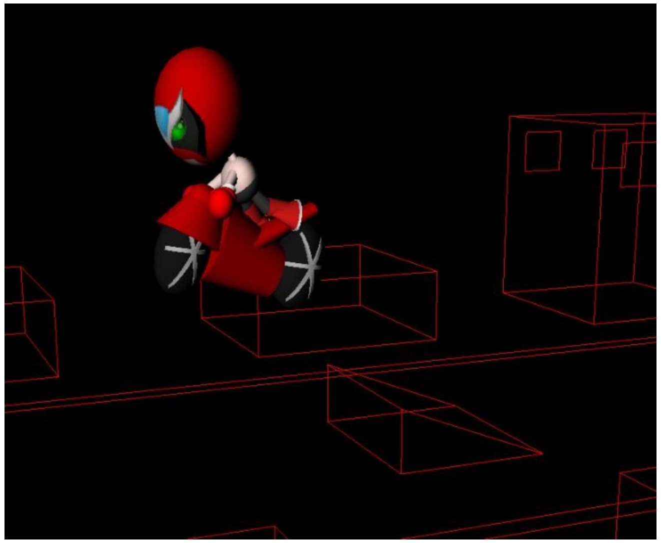
Animator Help Session



Assigned: Tuesday, February 25th

Due: Sunday, March 9th

Project TA: Mason Remy

Agenda

- Introduction
- Curve implementation

  - Requirements

  - What are all those vectors?

  - Where should I put things?
- Particle System

  - Requirements

  - What should I implement?

  - Suggestions

  - Cool forces

Agenda

- Extra credit ideas
- Animating well vs. well… just animating
  - Timing!
  - Music
  - Light
  - Lasseter's animation principles
- Creating your artifact
  - Selecting a curve type
  - Compositing

Introduction

- How to integrate with my model?
  - just replace sample.cpp with your model
  - better to do this sooner than later
- Modeler View vs. Curves View
- Graph Widget Interface

Requirements

- Bezier curve
  - can linearly interpolate in cases where there are not enough control points (< 4 or for the last couple in your set)
- B-spline
- Catmull-Rom
  - curve must be a function!
  - sample solution is not perfectly correct; you must do at least as well as sample.

Curve Implementation

What are all those vectors?

In any specific curveEvaluator class

- ptvCtrlPts: a collection of control points that you specify in the curve editor
- ptvEvaluatedCurvePts: a collection of evaluated curve points that you return from the function calculated using the curve type's formulas
- fAniLength: maximum time that a curve is defined
- bWrap: a flag indicating whether or not the curve should be wrapped

Curve Implementation

Vector Reference

For those of you not familiar with C++ vectors:

- A vector is a standard C++ data structure that acts like a dynamically resizable array (like an ArrayList in Java)
- A nice reference for vectors (one of many online): http://www.cplusplus.com/reference/stl/vector/
- Vectors often use iterators, which you can read about with some simple examples here: http://www.cppreference.com/wiki/stl/iterators

Where should I put things?

- Create curve evaluator classes for each that inherit from CurveEvaluator
  - Bezier
  - B-spline
  - Catmull-Rom
- In GraphWidget class
  - Change the skeleton to call your new constructors in the GraphWidget class.
  - Right now all the UI is set up for your new curve types, but they all call the constructor for the LinearCurveEvaluator class.

Particle System

Requirements

- Particle System class
  - Should have pointers to all particles and a marching variable (time) for simulation
  - If you have two separate simulations (say, cloth sim and particles that respond to viscous drag) you may want to make that distinction here (as well as in your force and particle implementation)
- Solver
  - In the skeleton, this actually exists within the Particle System class
- Particles

Requirements

- Two distinct forces
  - Distinct may mean forces whose values are calculated with different equations (gravity and drag are distinct because gravity eq is of form f=ma, where drag is defined in terms of a drag coefficient and velocity)
  - Alternatively (and better): distinct may mean that one force is a unary force and another is a n-ary force or spatially driven force
- Collision detection
  - With one primitive of your choice
  - Restitution coefficient must be slider controlled

Particle System

What should I implement?

- Canonical components
  - Constructor
  - Destructor
  - etc
- Simulation functions
  - `drawParticles()`
  - `startSimulation()`
  - `computeForcesAndUpdateParticles()`
  - `stopSimulation()`

Particle System

What should I implement?

- Particle struct or class
  - you may have several of these if you have multiple types of simulations
  - If this is the case, take advantage of inheritance
- Force class
  - An elegant implementation would include a generic Force class and a variety of distinct forces that inherit from it

Note: *I stress inheritance because it will make your implementation easier (and less messy) and in general will make your life easier. If the TA that grades your project must look at your code, clean object oriented code is a great headache-prevention-tool.*

Embedding in your hierarchy

- Need to find World Coordinates of Particles

  - Model View Matrix

  - Inverse Camera Transformation

  - Generate world coordinate for particles by undoing camera transformations to the point you want to launch from.

  - Note the provided pseudo-code and getModelViewMatrix() on the animator project page
- Euler Method
- Hooking up your particle System

Cool Forces

- Particles in a lattice
  - Cloth simulation
  - Deformable objects
- Flocking
  - Will require multiple forces:
    - Attractive force that affects far away particles
    - Repulsive force that affects nearby particles
    - What else?

Extra Credit ideas

- Tension control for Catmull-Rom
  - Interpolating splines are cool because keyframing the parameter values that you want is more intuitive…
  - But what about the time instances not keyed? Without control of curve tension, you may not get the parameter values that you would really like, and your animation could suffer
- Allow control points to have C^0, C^1, or C^2 continuity
  - This can be **VERY** helpful in creating a good animation
- Initialize particle velocity with velocity of model hierarchy node to which it is attached
  - This is fairly easy and will make your particle system noticeably more realistic

Extra Credit ideas

- Billboarding
  - Adding support for sprites (billboarding) can DRASTICALLY increase the aesthetic quality of your simulation
  - Additional benefit: very easy to ‘skin’ particles and make multiple instance of same particle look unique
- Baking
  - A must for complicated simulations or for particle systems with a lot of particles

Extra Credit ideas

- Better collision detection
- Better forces
- Lens flare
  - Most animator artifacts suffer from lack of realistic looking lighting
  - Ideally, this problem would be solved with ray tracing or photon mapping
  - Since these are probably not options, lens flare is an alternative way to give the impression of interesting lighting

Animating well
vs.
well… just animating

- Above all else, keep it simple:
  - You have limited time
  - You have a limited program (well, unless you implement a lot of bells and whistles)
  - If you make realistic goals, then meet them, you can use the extra time to add more shots and eye candy.
  - Complicated is not necessarily better

Animating well
vs.
well… just animating

- Have a plan
  - Though it seems simple, it's a lot easier to plan and iterate your animation on paper
  - Even if you don't draw, it's much easier to sketch out your key poses on paper, then implement them in animator
  - If you decide a pose or shot doesn't work on paper, you can make a new one in a couple minutes. If you find out a shot doesn't work after it's rendered, you have to get rid of potentially hours worth of work

Animating well
vs.
well… just animating

- Timing!
  - Timing is VERY, VERY important
  - Consider timing before you bother to get specific about joint rotations or object positions
  - Don’t forget you can change the animation length of your shots

Animating well

- Music!
  - Sound and music can greatly enhance the cohesion of your artifact
  - If your artifact idea includes a theme or stylization, it can be very effective to time your animation with events in the theme music.

Animating well

- Light!
  - Like sound, light is very important compositionally
  - Anything you can do to be creative with lighting will help

Light vs. No light

Animating well

- Use the animation principles
  - See John Lasseter's article on animation principles
  - See the lecture notes on animation principles
  - Remember, well animated grey models are a lot more entertaining than poorly animated complicated ones. That's why the first animated shorts ever (The Adventures of Andre and Wally B, Knick Knack, etc.) are still entertaining

Creating Your Artifact

- Choice of curve types
  - Bezier Curves
    - Recall the animation of a bouncing ball
    - When the ball hits the ground, the curve describing its position should have a C^1 discontinuity
    - Without C^1 discontinuity here, the animation will look wrong
  - Catmull-Rom is usually the preferred curve choice…
    - but unless your project supports the option to add C^1 discontinuity at will, you might find yourself trying to fight the Catmull-Rom to create pauses and other timing goodies

Creating Your Artifact

- Compositing
  - Recommended that you break your intended artifact up into shorter clips combining them all in the end.
  - This will make your life easier for many reasons:
    - Splitting up work is straightforward
    - Changing camera angles is GOOD for a composition
    - You can incrementally complete your artifact
  - Adobe Premiere
    - Play around with it, check the website for some details on how to use it. The user interface is pretty intuitive.

Any Questions?

Email the staff!

Cse457-staff@cs.washington.edu

Post on the discussion board (linked from the project page)