ODE / Creature Contest Help Session

Winter 2009

Outline

- ODE and GLUT libraries
- How the skeleton code works
  - If you were going to adapt any skeleton code for any graphics applications in the future… you could probably start with this instead of the other hairy assignments from this class
- ODE basics
- Project requirements

Open Dynamics Engine (ODE)

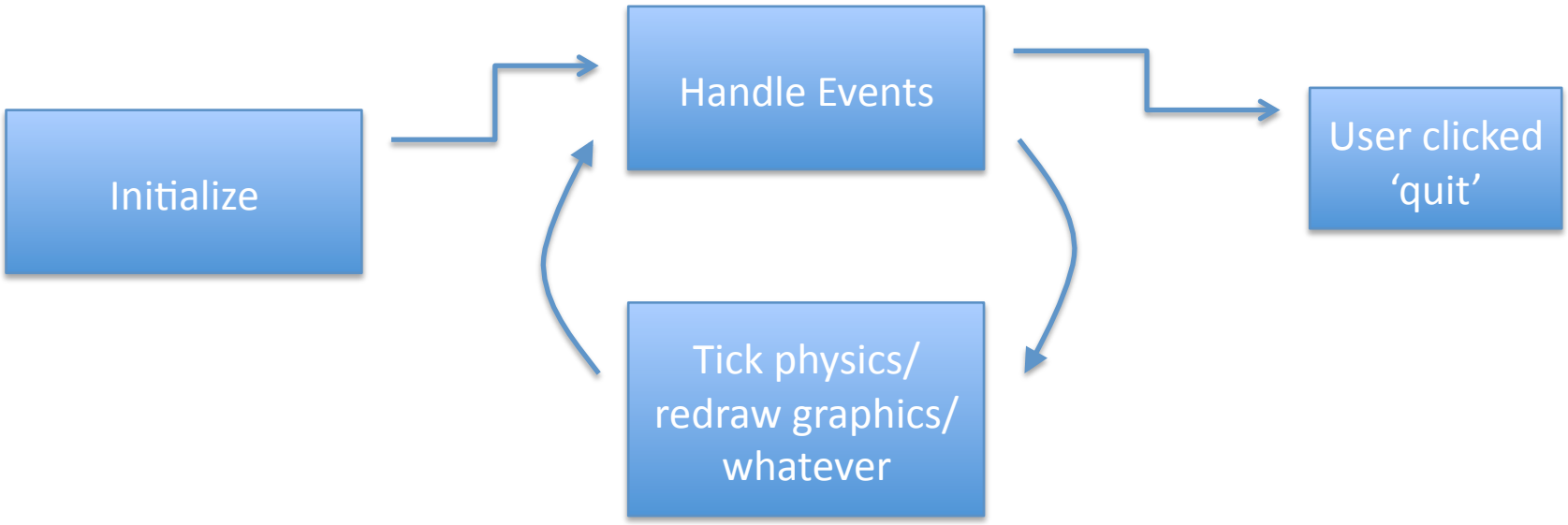
- http://ode.org
- Library for simulating articulated rigid body dynamics
- Fast, robust, built-in collision detection
- Articulated structure = rigid bodies of various shapes connected together with various kinds of joints

OpenGL Utility Toolkit (GLUT)

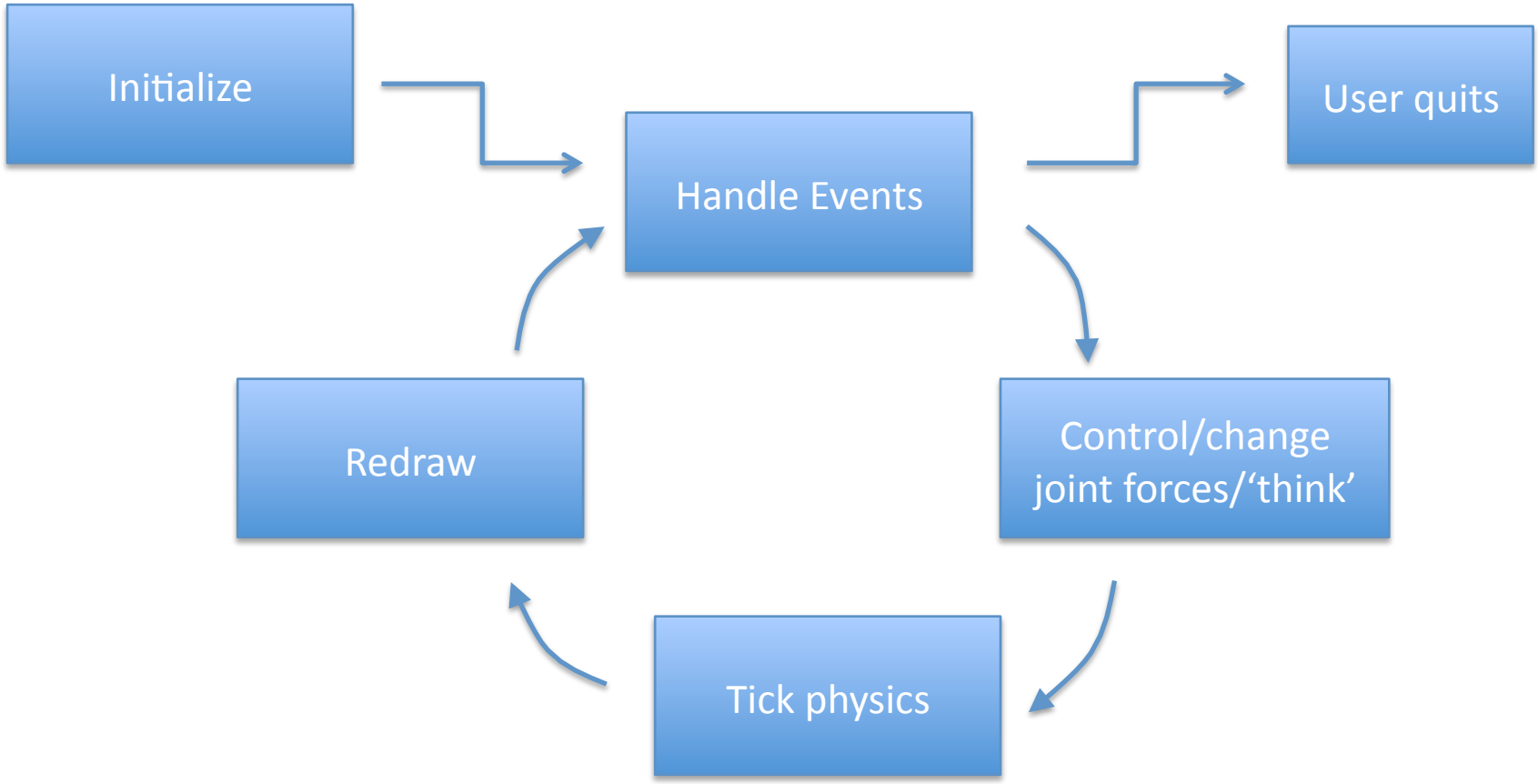
- http://www.opengl.org/resources/libraries/glut/
- Library for writing OpenGL programs
  - Makes it easy to write an application that opens a window and displays stuff
- Callback driven event processing
- Handle events from input devices: mouse, keyboard, etc.
- Kind of like FLTK, minus GUI

Program Flow

- Both GLUT and ODE work like this:
  - Set up some stuff, start main loop (handle events, draw/simulate, repeat)



Creature Contest Program Flow



Nitty gritty application details

- The graphics loop is the main loop
- After starting glutMainLoop(), no longer have direct control
- Need a physics simulation loop, too
- glutIdleFunc called when not drawing anything
  - Update physics here (dWorldStep())
  - Force redraw
- glutDisplayFunc called to redraw

Class: CreatureWorld

- Sets up graphics, physics, everything else
- Contains:
  - Main()
  - 1 Creature object
  - 1 Terrain object
- Init graphics: GLUT window, mouse and keyboard callbacks
- Init physics: create ODE simulation environment, set gravity to (0,0,-9.9), populate according to terrain and creature build() functions
- Init control: Start creature thought process

CreatureWorld (cont.)

- Each ‘tick’
  - Tick control: creature.think(), up to you to implement
  - Tick physics: dWorldStep()
  - Redraw: glutPostRedisplay()
- Drawing function
  - Camera controls (camera happens to follow a user-specified position on creature, you can change this)
  - Terrain responsible for drawing itself: terrain.draw()
  - Creature responsible for drawing itself: creature.draw()

Class: Terrain

- Ground
- Obstacles/ramps on ground
- Height map
- Build(): define the terrain, put it in physics simulation
- Draw(): describe how to draw it, can include colors and textures

Class: Creature

- Articulated rigid-body structure held together with joints
- Also contains some ‘brains’ for deciding how to move and control joints
- YOU have to build it, draw it, and make a controller for it:
  - build(), draw(), start_thinking(), think()

ODE Basics

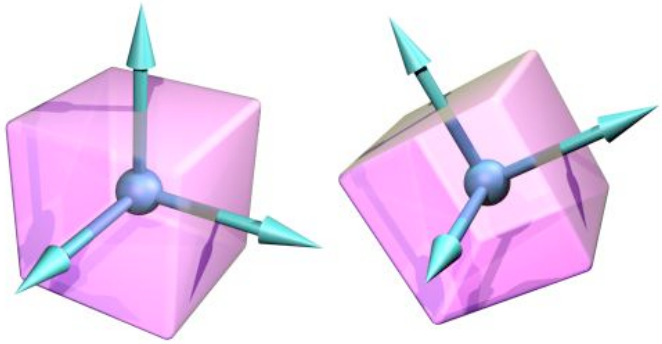
- Read the manual: http://opende.sourceforge.net/wiki/index.php/Manual

Some Terms

- World: ‘a dynamics world’
  - Think masses, forces, movement
- Space: ‘a collision space’
  - Think 3d shapes colliding, intersecting
  - Collisions generate *forces* which have an effect on the *world*
- Body: a rigid body living in the world
- Geom: geometry (for collision), need a geom to spatially describe a body if you want it to collide with anything
- Joint: a constraint between two *bodies*

Rigid Body

- A body has
  - Position (of its center of mass)
  - Mass, inertia matrix describing distribution of mass
  - Orientation
  - Linear velocity
  - Angular velocity
  - NO PHYSICAL SHAPE!
    - Unless you also define a geom
  - Lives in the WORLD!
    - Moves, has forces applied to it, may be connected to other bodies with joints



Geom

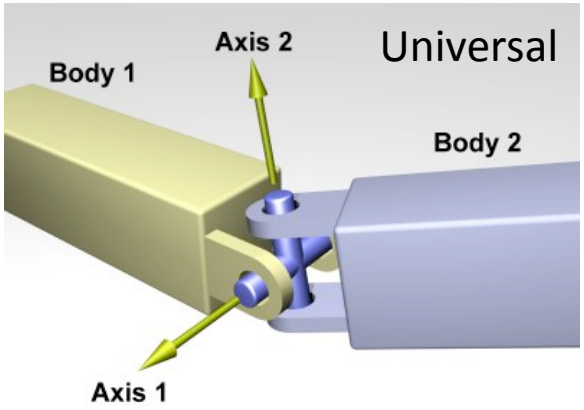
- A geom has
  - Shape (box, cylinder, etc)
  - Possibly a body associated with it
  - Lives in the SPACE!
- Each limb of your creature will be a body plus a geom
  - Define a geom in the same shape and position as the body's mass (see code for example)
  - Once attached to a body, a geom will have same position/orientation

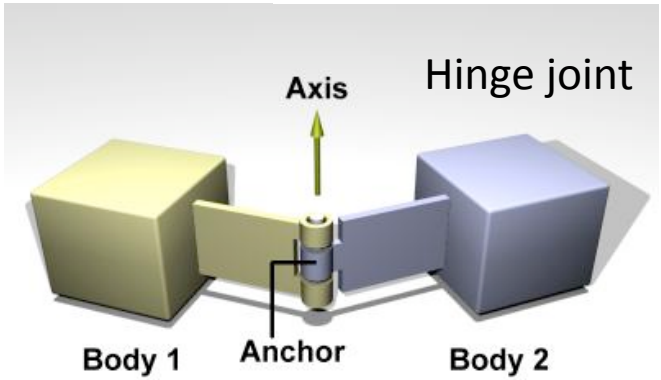
Geom (cont.)

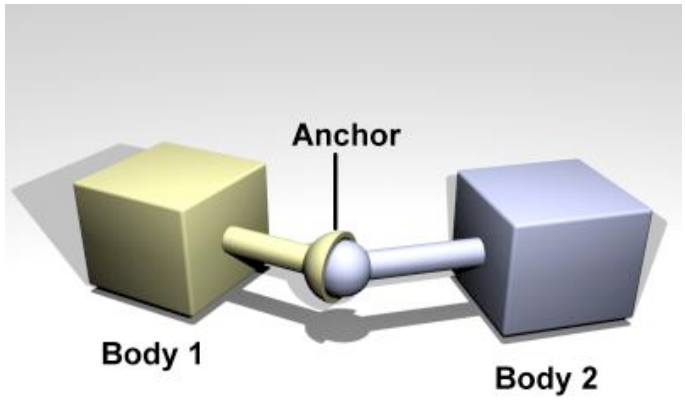
- In starter code:
  - geoms are all box shaped
  - put in a vector of geoms for drawing and manipulating
  - draw function loops over each boxy geom, gets position and scale factor, draws it, and draws its shadow
    - If you have differently shaped geoms, you should handle keep track of them and draw them differently
- Creature class includes function to turn geom/ body position into transformation matrix for easy drawing

Joint

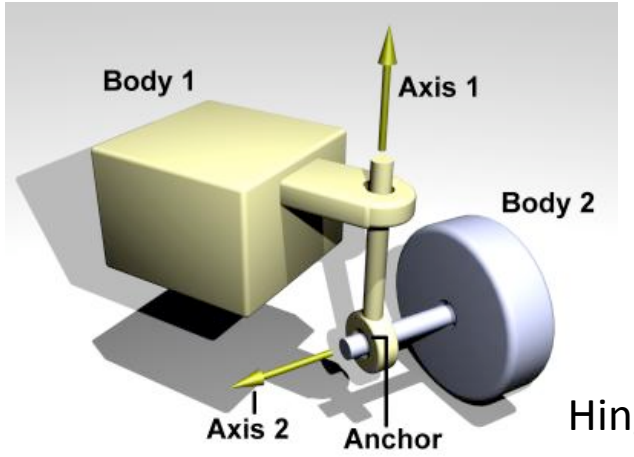
- Connects two bodies
- Lives in the WORLD
- Lots of kinds…







Ball and Socket



Hinge 2

Controlling Joints

- Read the manual, experiment, look at examples… I'm learning this along with you
- Motors
  - specify a velocity and the maximum force you want to use to achieve this velocity
- Stops
  - Prevent joints from moving along whole range of motion (eg. Your knee doesn't bend backwards)
  - Limit forces when using motors
- Motion parameters (dParamVel, dParamLoStop, dParamFMax are a few)
- Set force/torque directly

Collision Detection

- Already set up for you (look at cb_near for more details)
- Internal check to see which objects in the space are colliding
  - create contact points at each collision
  - create a joint (between bodies) at each contact point
  - force colliding bodies away from each other
  - destroy contact joints before next time step

General things you might end up tweaking

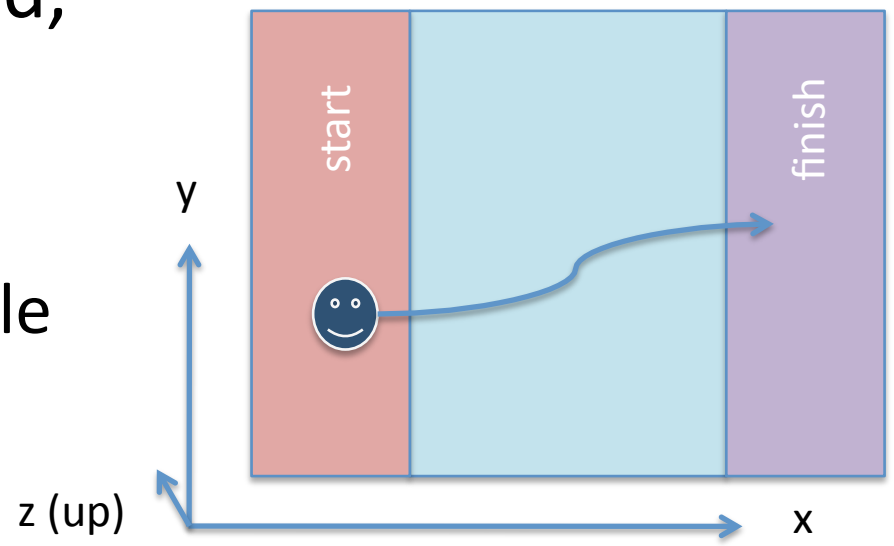
- Read the manual for different ways to tune your simulation
- dWorldStep vs dWorldQuickStep
  - QuickStep useful when your simulation has >100s of objects… less accurate and stable
  - Should probably use constant physics time step but starter code tries to keep constant animation rate… tweak this if your simulations have stability problems
- Error Reduction Parameter (erp)
- Constraint Force Mixing (cfm)
- Friction… maybe skeleton code doesn't have it

Putting it together

- Now you know the basics to give your creature joints and limbs!
- Gravity and collision simulations will happen automatically
- You need to define how the creature moves
- No animation key frames here, have to do it all in code…

Actual Project Requirements

- Make a creature that walks from one zone, across the playing field, to the finish zone.
  - Start: x<0
  - Finish: x>100? (variable course width, to be determined, tell to creature at start up)



Vague… grading… criteria…

- I'm not sure how to invent new assignments
- Have fun!! Do something cool!
  - Hopefully the time and effort you put into it will be apparent
- Get a creature that can stumble across a flat terrain
- Make a new terrain (ahh, you don't have to put these shapes into the World, just into the Space)
- Describe your design and controller in a short but intelligent-sounding webpage write-up, include some graphics and/or animations
- Bonus points for dressing up your creature with faces/colors/ textures/etc. (as long as it also can get from start to finish)
- I should make an actual webpage for this project and post this information there

Constraints

- Minimum torso size: 1x1x1
- Maximum torso size: 10x10x10
- Minimum density: not sure yet
- Minimum leg length/width ratio: not sure yet, can't have infinitely thin legs
- Maximum torque and velocity: not sure yet
- These are to keep you from 'cheating', but the creative ways you could cheat might be interesting

Basically…

- I could pick random numbers, but maybe that's not the point
- Just make something that walks, hops, or slithers and doesn't really cheat
- "How do I know if I'm cheating?"
  - Does it touch the ground a few times between start and finish area
- Speed, style, creativity will be judged subjectively by the class…

Check out these blocky creatures

- Karl Sims
  - Evolved Virtual Creatures (SIGGRAPH 1994)
  - http://www.karlsims.com/evolved-virtual-creatures.html