Advanced Modelling, Rendering and Animation
Coursework II

Tim Weyrich and James Tompkin

March 23, 2010

Assignment due: **12:00 noon 9th April 2010**

Framework

We provide a simple GLUT/OpenGL framework to run and visualise a particle and boids simulation, as described in the lecture. The framework should compile on Windows, Linux and MacOS X. (For MacOS X: you'll have to add "-lgl and -lglut" to the Makefile as denoted in the Makefile comment.)

Part I: Particle Simulation

1. The framework provides an (incomplete) C struct `Particle` that contains the physical state of a particle for a particle simulation. Part of this state should be a creation time stamp (double creationTimeStamp) denoted in seconds since program start. Organise all particles in your system in the provided global `ps.particles`.
2. A small GLUT application framework is provided for you. The `idle()` function receives program control at regular intervals. The `display()` function draws to the screen, and is notified to draw at the end of the `idle()` function. Two mouse functions are provided: `mouse()` and `mouseActiveMotion()`. `mouse()` is called whenever the mouse is clicked, and `mouseActiveMotion()` is called whenever the mouse is dragged.
3. Use the `idle()` function to call a particle simulation: this simulation will be a function `simulate()` that simulates the time duration between the previous and the current call to `simulate()` and updates the particles' states accordingly. (The stub`simulate()` shows how the provided function `myclock()` can be used to obtain the seconds since the first call to `myclock()`).

4. Implement this `simulate()`:

    a) The particle simulation should create particles at the centre of the screen at a constant rate, with a constant velocity in the direction of an initial vector. The only force acting on the particles shall be gravity (defined as a vector in `ParticleSystem`).

    b) Use the "explicit Euler" integration scheme described in the lecture.

    c) Depending on the time passed since the last invocation of simulate(), perform multiple integration steps before returning.

    d) Particles older than 5 seconds shall be removed from the simulation (check creation date).

5. Each particle shall be assigned an intensity of 1.0 at the beginning. (The display should show all particles at their intensities.) Reduce the intensities over time, so that a particle's intensity reaches 0.0 after 5 seconds. A particle whose intensity reached zero shall be removed from the simulation. (This can be done very easily with the deque data type used for ps.particles).

6. Both gravity and the initial velocity vector should be controllable with the mouse. The magnitude and direction of both these vectors should be controllable.

7. Additional marks are available for creative particle effects — blending, motion blur, etc.

Total marks: 40 + 10 additional.

Part II: Boids

1. Using the same framework, implement a Boids solution. Fill in the function, `simulateBoids()`, which should be togglable by some key (use the `keyboard` function to control the `boids` variable).

2. Some elements of the simulation from Part I are no longer needed: boids should not decay after 5 seconds, and the number of boids in the simulation will likely be less.

3. Boids are simulated using three functions as explained in the notes: separation, alignment, and cohesion. Obstacle avoidance should also be included. **Clarification: obstacles should be visualised on the screen. Note that a disc or sphere obstacle is sufficient for this part.** Implement these functions as you see fit.

4. Realistic sensing (within some radius) should also be included.

5. Allow the direction of your boids to be controlled by using the mouse.

6. Additional marks are available for improved boid rendering (e.g., as birds ;-), and for experimenting with creative boid behaviour. **Clarification: additional marks are also available for implementing object avoidance of more complicated shapes.**

Total marks: 40 + 10 additional.

Please write a short report on your work, detailing how you solved each part of the coursework. Include all relevant code along with screenshots to demonstrate your solution. Make sure that your report shows examples of all the required simulation effects. Submit all of it electronically, as described on the web page.

Good luck!