1 Programming

In this assignment you will animate a mud-like particle system with frictional collisions.

The main program `simulate_particles` will take a configuration text file as input, which includes the following lines in order:

- The length of the simulation in seconds
- The frame rate in frames per second
- The mass m of each particle
- The spring stiffness k
- The restlength r
- The interaction radius R
- The spring damping d
- The boundary friction coefficient μ

and in addition the name of a collision geometry file (in .OBJ format) and a format string (e.g. `particles%04d`) for the text files containing particle data. Optionally the program also takes the frame number to begin the simulation at (the default value is zero, but "restart" capability is something very useful to have in general, so it's a good idea to design it in from the start).

The text file containing particle data has the following format:

- n: the number of particles (a positive integer)
- x_0, y_0, z_0: the initial position of particle 0
- u_0, v_0, w_0: the initial velocity of particle 0
- ...
- $x_{n-1}, y_{n-1}, z_{n-1}$
- $u_{n-1}, v_{n-1}, w_{n-1}$

The program will output the positions of the particles at each frame in a separate text file of the same format.

The force on particle i due to particle j is:

$$F_{ij} = \begin{cases} \left(1 - \frac{|\vec{x}_i - \vec{x}_j|}{R}\right)\left[-k\left(\frac{|\vec{x}_i - \vec{x}_j|}{r} - 1\right) - d\frac{(\vec{v}_i - \vec{v}_j)\cdot(\vec{x}_i - \vec{x}_j)}{|\vec{x}_i - \vec{x}_j| r}\right]\frac{\vec{x}_i - \vec{x}_j}{|\vec{x}_i - \vec{x}_j|} & : \quad |\vec{x}_i - \vec{x}_j| < R \\ 0 & : \quad |\vec{x}_i - \vec{x}_j| \geq R \end{cases}$$

The total force on a particle is the sum of the forces from other particles along with gravity. Your program will compute the motion of the particles using Symplectic Euler. Note that a critical part of this is determining a stable time step size (see the analysis section). In addition, particles collide inelastically with friction against the specified unmoving triangle mesh geometry.

I have provided some code to start you off, in the directory `simulate_particles`. See the README for instructions on what code you need to add, how to compile, etc.

There is also a directory `make_blobbies`, containing a simple program to construct a triangle mesh that smoothly wraps around the particles (technically: runs the marching tetrahedra algorithm on a blobby isosurface). See the README there for more information. You can use this project to create .OBJ geometry files for rendering. You do not need to add code here.

The third directory, `view_obj`, is a generic animation viewer for .OBJ format files. You can use this to quickly visualize the output from the blobby output. See the README in that directory for instructions on how to use it. You do not need to add any code here: this is purely for your convenience in debugging and analyzing what's going on. One particularly useful feature is that you can export a RenderMan RIB file containing a description of the current camera view, for use in your final rendering.

Finally, there is a directory `obj_to_rib`, which contains a simple project for converting .OBJ files to RenderMan RIB output. This will help you construct the RIB files for each frame to render your final animation. I have also included an example RIB file that you can use as a guide for producing output.

2 Analysis

(1) Figure out a reasonable stable time step size for Symplectic Euler applied to the system you are implementing above. Assume that the program will find the number of nearby particles for each particle in the system (this will play a role in your time step estimate).

(2) Prove that RK4 applied to $dx/dt = -x$ is never oscillatory no matter how large the time step is (even if it is going unstable). Hint: use Taylor's remainder theorem.

(3) Prove that even in the general nonlinear case $dx/dt = f(x)$, trapezoidal rule ($x_{n+1} = x_n + \frac{1}{2}\Delta t(f(x_n) + f(x_{n+1}))$) and midpoint rule ($y_{n+1} = y_n + \Delta t f(\frac{1}{2}y_n + \frac{1}{2}y_{n+1})$) have the same stability, assuming a fixed time step Δt. That is, one method grows exponentially if and only if the other method grows exponentially. Hint: look at $z_{n+1/2} = \frac{1}{2}(y_n + y_{n+1})$.

3 Handing It In

You need to hand in the code you write for `simulate_particles` and `render_particles`, an animation you generate from your code (note that you will need to come up with your own particle system constants, collision geometry, and initial conditions), and answers to the questions in the analysis section.

To hand in the animation you generate for an assignment, put it on the web and email me the URL.

To hand in the programming part of an assignment, tar and gzip the directory containing your source code (but not output data, object files or executables!). If you have changed how the code is compiled or run, or sone anything you want to comment on, include a README file. Then email me the .tar.gz file as an attachment.

To hand in the written part of an assignment, either email it to me (preferably in plain text or PDF), give it to me in person, or slide it under my office door.

I understand it is sometimes difficult to turn in assignments on time due to unforeseen emergencies. If you have a good reason you can't turn something in on time, please contact me as soon as possible about it and we can work something out. Otherwise, there will be a 20% per day late penalty, starting when I come to my office in the morning after the due date.