FINAL PROJECT

Due: Friday, June 15, 2018, 11:55 pm

Your objective for the final project is to develop a set of MATLAB codes that will allow you to study fluid behavior in 2D using smoothed particle hydrodynamics (SPH) on a spatially-hashed domain. The forces driving a single particle in your simulation are relatively simple, but the combined behavior of the entire particle system can model some amazingly complicated fluid behaviors. Review the lecture slides and check this document frequently for any updates. A few notes and guidelines on the project:

1. **Data Structures.**

   Before you start calculating any fluid properties, you'll need to be *very* comfortable navigating two key data structures: (1) the particles in your system, $k = 1, 2, ..., N$ and (2) the bins which contain them, $z = 1, 2, ..., \text{numBins}$. Every particle in your grid should have *at least* the following set of properties:

Name	**Description/Example**
Position (x)	`1x2` vector, `[x, y]`
Velocity (v)	`1x2` vector, `[v_x, v_y]`
Force (f)	`1x2` vector, `[f_x, f_y]`
Density (ρ)	Scalar
Neighbors	Vector containing ID's of all nodes within distance h of this node

   The bin structure is simpler by comparison, requiring only that every bin store a list of all the particle ID's contained in that bin and a list of all bins adjacent to that bin:

Name	**Description/Example**
Particle IDs	Vector containing ID's of all particles within this bin
Adjacent Bins	Vector containing ID's of all bins adjacent to this bin

   Since this set of properties will be associated with *every* particle and bin, structured variable arrays should be used to keep things organized. You're free to incorporate additional fieldnames and values as you see fit, but you *must* incorporate at least one structured variable array in your solution.

2. **Initializing Fluid Particles.**

   Begin by assigning each of the particles, $k = 1, 2, ..., N$, in your simulation a position on the 2D grid. An equally-spaced rectangular array is a good starting point, but is by no means the only layout you can construct. Depending on the simulation you're designing, you can choose to set the initial velocity to zero (fluid starts from rest) or non-zero (e.g., a horizontal jet of water, $v_x = 1$). After initializing each particle, we'll set the properties of the fluid being simulated.

Symbol	Definition	Value
ρ_0	Rest density	1000
m	Mass of single particle	ρ_0/N
κ	Stiffness constant	100
μ	Viscosity coefficient	0.1

You'll need to experiment with different values to find what works for your setup, but the above values work relatively well for a fluid moving under gravity given a sufficiently small timestep size. Note that the mass of each particle is a constant, so we set only a single value to be used in every calculation. By setting this mass value with the calculation ρ_0/N, we're assuming the fluid is initially at its rest density provided the particles cover roughly a unit area.

Lastly, we set the smoothing radius, h, which will govern the strength and spread of all the particle-particle interactions. You'll need to experiment to find a smoothing radius that includes just enough neighbor particles for a stable, accurate simulation, but a good rule of thumb is to set h just small enough to satisfy that ≤ 8 neighbor particles are identified in the most particle-dense region of your fluid.

3. **Initializing Bins.**

   For each bin in your simulation, $z = 1, 2, ..., \text{numBins}$, construct a vector that contains the bin IDs of every adjacent bins. Since this information doesn't change during the course of the simulation (the position of each bin is a constant) it only needs to be computed once at the beginning of your method. Next, calculate the bin number for every particle in your simulation using the hash function below:

$$\text{binNum}_k = \left(\left\lceil \frac{x_k}{\Delta x} \right\rceil - 1 \right) N_Y + \left\lceil \frac{y_{max} - y_k}{\Delta y} \right\rceil \qquad \text{for } k = 1, 2, ..., N \tag{1}$$

   where $\lceil ... \rceil$ indicates rounding the bracketed quantity up to the nearest integer. As in Homework 8, h, the smoothing radius, will be used along with x_{max} and y_{max} (the domain size you set) to generate the integer number of bins in both the x- and y-directions, N_X and N_Y, respectively. Depending on your choice of h, the *actual* bin dimensions, Δx and Δy, may be larger than requested (remember, the value h controls the *minimum* width and height).

   We'll use this classification to construct a list of the particle ID's contained within each bin. For example, `bin(1).particleIDs = [3, 8, 11]` indicates that particles 3, 8, and 11 are located within bin 1. Since the fluid particles move during the course of the simulation, the particle lists for each bin will need to be re-built after each timestep.

4. **Identifying Neighbors.**

   Now that both structures are initialized, we can build the list of neighbors for every particle. Because we're using spatial hashing, we only need to calculate the distance between particles if they are located in the same or adjacent bins. For all the particles in a given bin, we compare them against all the particles in the same bin *and* all the particles in every adjacent bin. If the distance is less that our smoothing radius, h, we add the corresponding particle ID as a neighbor. This process is laid out in the pseudocode below.

```
for $z$ = all bins
    Get all particles in bin z;
    if bin $z$ is not empty
        Get vector of bin IDs adjacent to bin z;
        for $w$ = [z, adjacentBins]
            Get all particles in bin w;
            for $k$ = all particles in bin $z$
                for $j$ = all particles in bin $w$
                    dist = $||\boldsymbol{x}_k - \boldsymbol{x}_j||$;
                    if dist $< h$ and $k \neq j$
                        Particles k and j are neighbors;
                    end
                end
            end
        end
    end
end
```

where the $||...||$ notation denotes the Euclidean norm of a vector (e.g., $||\boldsymbol{x}|| = \sqrt{x_1^2 + x_2^2}$). The index z can be thought of as the current bin and w as the comparison bins. Note that w takes on the value of z itself, which ensures that we include the distances between same-bin particles in our identification of neighbors.

5. **Calculating Density.**

   Now that the list of neighbors is calculated for each particle, we can calculate the fluid density represented at each point. The fluid density at particle k, ρ_k, depends on the neighbor particles' mass and the separation distances as shown below

$$\rho_k = \frac{4m}{\pi h^2} + \frac{4m}{\pi h^8} \sum_{j=\text{neigh}_k} (h^2 - ||x_k - x_j||^2)^3 \tag{2}$$

   where the first term on the right-hand side of Equation 2 is due to particle k's self mass and the terms in the summation are due to the k's neighbors.

6. **Calculating Forces.**

   We'll now make use of both the stored density values and the list of neighbors to calculate the total force on particle k, $\boldsymbol{f}_k$. Each particle, $k = 1, 2, ..., N$, is affected by the sum of the fluid force contributions from neighbors *plus* any external forces like gravity, wind, etc.

$$\boldsymbol{f}_k = \boldsymbol{f}_{\text{external}} + \sum_{j=\text{neigh}_k} \boldsymbol{f}_{k,j} \tag{3}$$

   where the fluid force on particle k due to neighboring particle j is given by the following expression

$$\boldsymbol{f}_{k,j} = \frac{m}{\pi h^4 \rho_j}(1 - q_{k,j}) \left[15\kappa(\rho_k + \rho_j - 2\rho_0)\frac{1 - q_{k,j}}{q_{k,j}}(\boldsymbol{x}_k - \boldsymbol{x}_j) - 40\mu(\boldsymbol{v}_k - \boldsymbol{v}_j) \right] \tag{4}$$

where $q_{k,j} = ||\boldsymbol{x}_k - \boldsymbol{x}_j||/h$ and κ, ρ_0, and μ are the stiffness constant, rest density, and viscosity coefficient, respectively. Since a particle cannot exert pressure or viscous effects on itself, the calculation of $\boldsymbol{f}_k$ does not include a "self" term as we saw in the density calculation.

Note: Since the force on particle j due to k, $\boldsymbol{f}_{j,k}$, involves mostly symmetric terms compared to $\boldsymbol{f}_{k,j}$, it's possible to avoid a completely separate calculation by simply adjusting for the ρ_j term in the denominator: $\boldsymbol{f}_{j,k} = -(\rho_j/\rho_k)\boldsymbol{f}_{k,j}$.

7. **Update Kinematics.**

   Once the total force acting on each particle is known, its acceleration, velocity, and position can be calculated using the following kinematics equations

$$\frac{d\boldsymbol{v}_k}{dt} = \frac{\boldsymbol{f}_k}{\rho_k} \tag{5}$$
$$\frac{d\boldsymbol{x}_k}{dt} = \boldsymbol{v}_k \tag{6}$$

   We'll apply the same semi-implicit time stepping scheme we used in Homework 4. This means we'll calculate the new velocity using *explicit* Euler and the new position using *implicit* Euler (i.e., the right-hand side of Equation 6 should use the newly-calculated velocity produced by evaluating Equation 5 explicitly).

   Boundary conditions should be implemented immediately after adjusting the position of each particle. If the updated x and/or y position of a particle exceeds the limits of the boundary walls you specify at $x = 0$, $x =$ xMax, $y = 0$, or $y =$ yMax, we'll update its position and velocity in the following way

$$\boldsymbol{x}_k^d = 2(\text{boundary}) - \boldsymbol{x}_k^d \tag{7}$$
$$\boldsymbol{v}_k^d = -\beta \boldsymbol{v}_k^d \tag{8}$$

   where d is either a 1 or 2 indicating the first or second component of the position and velocity vectors depending on whether the collision was with a vertical or horizontal wall, respectively. Conceptually, this approach reflects the particle onto the correct side of the domain and reverses its velocity with an amount of damping you control (e.g., $\beta = 0.75$).

   Note: Only the position and velocity components *normal* to the orientation of the boundary receive this update. For example, if a particle crosses the vertical wall at $x =$ xMax, then $x_k = 2(\text{xMax}) - x_k$ and $v_x = -\beta v_x$ (reflected), while $y_k = y_k$ and $v_y = v_y$ (unchanged).

8. **Visualizing Results.**

   A significant portion of your final project grade rests in how you present your results, so spend some time making your results look nice! At a minimum, your visualization must: (a) clearly show the location of each particle and (b) clearly show the location of any relevant boundaries. You must create movie files to submit alongside your report code *in addition to the still images used in your report*. You should submit *at least* the following results in your report (saved images plus a brief explanation) to showcase your method.

   - A simple "dam break" simulation, where the particles are initially positioned to cover roughly half the domain and then fall and spread due to a gravitational force. The purpose of this case is to show that your method works for a relatively simple case where the expected behavior is known in advance before further complicating the problem. The exact simulation parameters aren't critical; as long as you can point to key values and behaviors in this result to defend the correctness of your SPH code, you've got a good verification result.

- A complicated setup of your own design. This can include additional geometric boundaries and/or obstacles, irregular particle initializations (water balloons, fountains, jets, etc.), multiple fluid interactions, and non-constant external forces. Experiment with different grid sizes and fluid constants to see what your simulation can really do!

These results can, of course, be supplemented with any examples you find interesting or necessary in supporting key points in your report's discussion. Spend some time answering the following questions with your new simulation tool: **(a)** In your own words, how do the values of of κ and μ influence your simulation in terms of observed fluid behavior and numerical stability? **(b)** How does your simulation handle the mixing of two different fluids with different rest densities (e.g., `particles(1:N/2)` have $\rho_0 = 1000$ while `particles(N/2+1:N)` have $\rho_0 = 2000$)? **(c)** In detailed words or pseudocode in the report only, how would simulate a fluid inside a *circular* domain centered at $(0, 0)$ with size R_{max}? Make sure you address the spatial-hashing scheme you would use, the test for particles crossing a boundary, and the boundary reflection update.

Extra Credit: Open Boundary Condition Water Tunnel. In the core method outlined above, we're limited to testing fluids contained inside solid boundaries. In this extension, you'll remove one or more of the reflective boundaries to simulate a steady stream of fluid particles acting as a water tunnel. You'll first need to modify your algorithm to allow particles to move *past* the grid limits and exit the domain. Particles that are sufficiently outside the domain (a test you design based on position and velocity) should be removed from the particles structure so they're no longer considered. Meanwhile, new particles should be seeded upstream so that the incoming flow is uninterrupted. Although the exact value of N changes, this deletion/insertion procedure means the total number of particles is effectively capped. Design a solid obstacle to test in the flow by adding tightly-packed, fixed-position fluid particles in an arrangement of your choice.

Note: If you attempt the extra credit, be sure to include a result or two in your report highlighting how it works!

Using the following naming convention, `fp_UID.zip`, `fp_UID.pdf`, `fp_UID.m`, where `UID` is your nine-digit University ID number, submit **two** separate files to the CCLE course website: (1) a .pdf of your written report and (2) a .zip file containing all of the MATLAB files written for the assignment. Remember to use good coding practices by keeping your code organized, choosing suitable variable names, and commenting where applicable. Any of your MATLAB .m files should contain a few comment lines at the top to provide the name of the script, a brief description of the function of the script, and your name and UID.