600086-Lab-I Soft body physics

Concept

Implement two versions of a 2D soft body physics engine, one on the CPU and the other on the GPU.

Specification

The scene comprises of a 2D grid of N by M elements.

Each element is a quadrilateral with four corners made of points, and four edges made of springs / dampers

Each point has a mass (m), position (p_x, p_y) and velocity (v_x, v_y).

The springs / dampers create a constraint on the points, allowing the distance between them to expand or contract.

The algorithm is as follows:

1. Sum all forces acting on each point
   - 4 springs, one connected to each of the adjacent point masses (unless on the edge)
   - 1 damper, which resists movement
   - gravity
   - an external force
2. Calculate the acceleration (a_x, a_y) for each point using $F = m * a$, where F is the total forces acting on the point.
3. Calculate the new position (p_x, p_y) for each point using $p_{t+1} = p_t + v * dt + 0.5 * a * dt^2$, where dt is the time to render a single frame, v is the previous velocity and p_t is the previous position.
4. Calculate the new velocity of each point using $v = (p_t - p_{t-1})/dt$, where p_t is the current position and p_{t-1} is the previous position.

As this is a 2D simulation, the grid will be in the same plane as the screen.

If it is working correctly you should eventually see a stable grid on the screen.

springs

The forces between two adjacent points 1 and 2 can be calculated as:

$dx = p2_x - p1_x$

$dy = p2_y - p1_y$

$distance = sqrt(dx^2 + dy^2)$

$magnitude = springCoeff * (distance - springRelaxDistance)$

$springForce_x = magnitude * dx/distance$

$springForce_y = magnitude * dy/distance$

Hint: *Remember to apply the force to both points 1 and 2*

Initially try setting the $springRelaxDistance$ to your grid spacing. $springRelaxDistance = 1.0$ is a good starting point.

$springCoeff = 10.0$ is a reasonable starting point, but experiment to find a value that works for your simulation, once its working correctly

dampers

The damping force resists movement of point

$dampingForce_x = -v_x * dampingCoeff$

$dampingForce_y = -v_y * dampingCoeff$

$dampingCoeff = 0.03$ is a reasonable starting point, but experiment to find a value that works for your simulation, once its working correctly

gravity

Gravity only acts in the y direction

$gravityForce_x = 0.0$

$gravityForce_y = -9.81 * m$

Setting $m = 0.01$ is a good starting point for the mass

external

$externalForce_x = rand(-1..1) * externalMagnitude$

$externalForce_y = rand(-1..1) * externalMagnitude$

Gravity

Once you have a stable simulation, it is time to introduce gravity.

Use the ‘G’ key to toggle on and off gravity, as explained previously.

Gravity acts from the top of the screen to the bottom.

Fix the position of some or all of the points along the top row of the grid, to prevent the grid from falling off the bottom of the screen.

If you fix only the top two corners then the cloth with hang.

Interaction

By using the keyboard or mouse select a point mass within the hanging grid and apply a further force to disturb the equilibrium.

Algorithm Stability

The stability of these simple cloth simulations can be problematic. Initial values for all the constants have been provided. If you used these constants, and you have correctly implemented the algorithms then you should have a stable result.

It is also suggested that you limit or even fix your delta time to 0.01.

All units in this assessment are in metres, seconds, kilograms and newtons.

Parallel architecture

You are required to implement the simulation on two different architectures: CPU and GPU.

CPU implementation

The grid is to be divided up across a number of threads. Explore 3 possible options:

- number of threads equals core count
- number of threads is 2 x core count
- number of threads is 4 x core count

Increase the size of the grid ie M and N values, and plot the timings for each of the three simulations.

What conclusions can you draw from this experiment?

GPU implementation

The grid is to be divided up so that each mass is implemented on its own CUDA thread.

Alter the values of M and N and plot the results.

What conclusions can you draw from this experiment?

Visualization

The grid is to be visualised using either OpenGL or DirectX, where each spring/damper is rendered as a line.

HINT: You are likely to find that the visualisation slows down your simulation. If this is the case consider running your simulation loop multiple times before rendering the scene. e.g. calculate all the physics 10 times, before drawing once.

Implementation

CPU

A Rust implementation

GPU

A CUDA implementation. The choice of language from which to host CUDA is left open e.g. C++, C# or Python

Lab book entries

1. GPU design - Detail and reflect upon the design of your GPU implementation. Focus on the parallel aspects of your solution. Clearly explain how you have used threads, mutual exclusion and synchronization in your GPU implementation. (approx. 300-500 words)
2. CPU design - Detail and reflect upon the design of your CPU implementation. Focus on the parallel aspects of your solution. Clearly explain how you have used threads, mutual exclusion and synchronization in your CPU implementation. (approx. 300-500 words)
3. Performance metrics - Compare the performance of your GPU and CPU implementations, both WITH and WITHOUT graphics. Include a description of your performance benchmarks. (approx. 300-500 words)

Demonstration

You will be required to demonstrate your two solutions in FEN 177, running on the lab PCs

Submission

All submissions are via Canvas.

Software - CUDA

Your code, in the form of a Visual Studio solution, source code, and working executables/DLLs, along with any required assets.

Software - Rust

Your code, in the form of a Visual Studio Code, source code, and working executables, along with any required assets.

Video

A short narrated video showing both the CPU and GPU implementations

Lab book

The full lab book containing all labs, as a PDF.