Assignment 3

Deadline:	Hand in by 5pm on Friday 31st May 2024
Evaluation:	20 marks – which is 20% of your final grade
Late Submission:	2 marks off per day late
Work:	This assignment is to be done **individually** – your submission may be checked for plagiarism against other assignments and against Internet repositories. If you adapt material from the internet you must acknowledge your source.
Purpose:	Develop a parallel implementation of a simulation.

Problem to solve:
A reference (sequential) implementation of an ***n-body simulator*** is available on the course stream site. This program simulates the motion of a number of bodies (planets, particles etc) with an attractive force between each pair of particles. Your task is to convert this program to a multi-threaded implementation with the purpose of improving performance.

Description:

You may use whichever multi-threading library/API you choose to convert the simulator to a multi-threaded implementation intended for a multi-core PC. The current implementation is written in C/C++ but you may write your new implementation in a different programming language if you so choose.

You are free to re-arrange the computation however you see fit; however, the simulator **must** compute the force between each pair of particles (*n^2* operations) and perform the calculations using double precision floating-point values. Do **not** attempt to convert the program to an *n log(n)* algorithm or similar approximate method.

The state of a body in an ***n-body*** simulation consists of a *position*, *velocity*, *mass* and *radius*.

```
struct body {
   vec2 pos;
   vec2 vel;
   double mass;
   double radius;
   ...
};
```

The force between each pair of bodies can be calculated by iterating through every pair.

```
// for each body
for(int i = 0; i < N; ++i) {
   // for each following body
   for(int j = i+1; j < N; ++j) {
      // Calculate force between bodies
      ...
   }
}
```

(continued over)

The force between two bodies is calculated from their relative *positions* and *masses* and accumulated in an array of *accelerations*.

```
vec2 dx = bodies[i].pos - bodies[j].pos;      // Difference in position
vec2 u = normalise(dx);                       // Normalised difference in position
double d2 = length2(dx);                      // Distance squared
if(d2 > min2) {                               // If greater than minimum distance
   double x = smoothstep(min2, 2 * min2, d2); // Smoothing factor
   double f = -G*bodies[i].mass*bodies[j].mass / d2; // Force
   acc[i] += (u * f / bodies[i].mass) * x;           // Accumulate acceleration
   acc[j] -= (u * f / bodies[j].mass) * x;           // Accumulate acceleration
}
```

Finally, the new state (*position* and *velocity*) of each body can be calculated from the old state, the accumulated *acceleration* and the timestep *dt*.

```
// For each body
for(int i = 0; i < N; ++i) {
   bodies[i].pos += bodies[i].vel * dt; // Update Position
   bodies[i].vel += acc[i] * dt;        // Update Velocity
}
```

Details:

The sequential implementation can be compiled into an executable that has a real-time graphics display or a version with no real-time graphics but that writes the final positions and an image to file. The real-time graphics version can be compiled by including the flag `-DGRAPHICS` and linking to the SFML libraries (available here https://www.sfml-dev.org/).

Requirements:

- You must provide full details of the language and multi-threaded API you used to develop the implementation (including any version information and any specific flags).
- Your submission will be marked based on its performance and how effectively it parallelises the program. The performance will be benchmarked on a multi-core machine with a large number of particles (>= *5000*).
- The particle positions produced by your implementation (written to the output file) should be almost the same as the original sequential version. Be aware that there may be slight differences due to floating-point errors (see assignment video for discussion on expected accuracy).

(continued over)

You **must** follow the next two specifications in **each and every assignment for this course**

1. Place the following comments at the top of your program code and **provide the appropriate information**:

```
/* Family Name, Given Name, Student ID, Assignment number, 159.341
*/
/* explain what the program is doing . . . */
```

2. Ensure that your `main` function uses `printf` to print this information to the console. You might use code like:

```
int main( int argc, char * argv[] ){
   printf( "----------------------------------------" );
   printf( "  159.341 Assignment 3 Semester 1 2023  " );
   printf( "  Submitted by: Rick Deckard, 20191187  " );
   printf( "----------------------------------------" );
   ...
}
```

Hand-in: Submit **your program and documentation (a zip file is acceptable)** electronically through the form on the stream site.

Marks will be allocated for: correctness, fitness of purpose, sensible use of data structures and algorithms, utility, style, use of sensible **comments and program documentation**, and general elegance. Good comments will help me to award you marks even if your code is not quite perfect.

If you have any questions about this assignment, please ask the lecturer.