Assignment 2 – Computer Animation and Visualisation

Deadline – 27 March, 4pm **Submit to floyd.m.chitalu@ed.ac.uk**
Format: <StudentID_FirstName_LastName>.zip

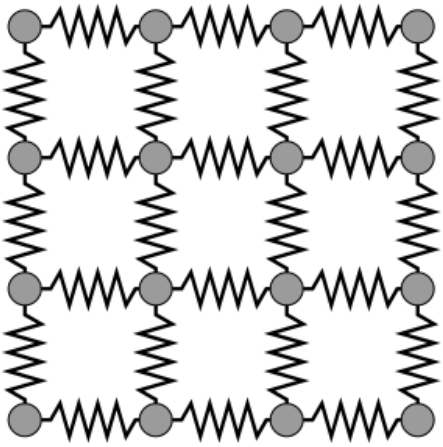
In this assignment, you will work with the C++ programming language to learn about implementing a mass-spring system, and a soft-body simulation using the Finite Element Method (FEM).

The final assignment should be submitted as a .zip file which contains only the source files of your project. Do not include pre-compiled binaries unless deemed necessary. For non-coding tasks, please submit an additional file that contains your answers.

1. Extract the contents of the project .zip file from the course website into a directory of your choosing on your machine.

   The instructions for building and running the template can be found in the README, which is a simple text file describing basic setup steps.

TUTORIAL



A mass spring system (shown above) is described by the following equation:

$$\mathbf{f}_p = \left[k_s \left(\frac{\|\mathbf{x}_q - \mathbf{x}_p\|}{r} - 1 \right) + k_d \left(\frac{(\mathbf{v}_q - \mathbf{v}_p) \cdot (\mathbf{x}_q - \mathbf{x}_p)}{r\|\mathbf{x}_q - \mathbf{x}_p\|} \right) \right] \frac{\mathbf{x}_q - \mathbf{x}_p}{\|\mathbf{x}_q - \mathbf{x}_p\|}$$

2. What exactly does this equation describe? Give details about each term on the right-hand-side.

3. How does k_s affect computational performance and the stability of a simulation? Describe a possible solution to mitigate this problem.

4. Describe is the physical interpretation of k_d *?*

TASK 1 – Implementing a mass-spring system

4. Navigate to the project directory and open a file called “mass_spring.cpp”. Implement a mass spring system using the algorithm provided below. A template file has been provided to help you get started. You are required to implement the missing code fragments which have been marked with “TODO”.

Refer to the README for instructions on how to build your executable.

a) Implement code to apply gravity to each particle

```
1: for Particle p : particles do
2:     p.frc = 0
3:     p.frc += p.mass*gravity
4: end for
```

b) Implement code to evaluate spring forces

```
5: for Spring s : springs do
6:     Vec3 d = particles[s.j].pos - particles[s.i].pos
7:     double l = mag(d)
8:     Vec3 v = particles[s.j].vel - particles[s.i].vel
9:     Vec3 frc = (k_s*((l / s.r) - 1.0) + k_d*dot(v / s.r, d / l)) * (d / l)
10:    particles[s.i].frc += frc
11:    particles[s.j].frc -= frc
12: end for
```

c) Implement numerical integration (explicit Euler).

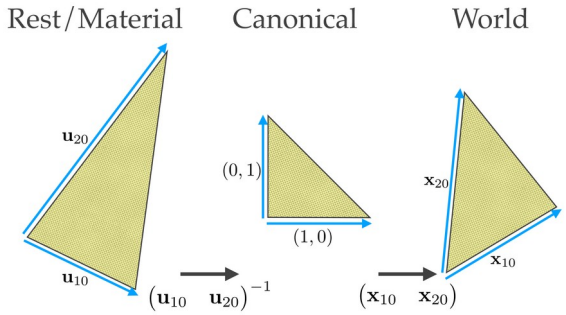
```
13: for Particle p : particles do
14:     p.vel += dt*(p.frc / p.mass)
15:     p.pos += dt*(p.vel)
16: end for
```

TASK 2 – Implementing soft-body physics using FEM

5. For this task, you are required to implement a so-called “volumetric deformable object” simulation using the finite element method. A template is provided to help you get started: Navigate to the project folder and locate a file called “fem.cpp”. Within this file, you are required to implement the missing code fragments which have been marked “TODO”.

Hint: Refer to your lecture notes for details on how to implement each required code fragment.

The following algorithm provides you with an idea of how to go about completing this task. Be aware that this is only guiding aide (the notation is also a little different from the source code – we use capital “X” to name coordinate variables of material/rest shape instead on “u”).




```
for Particle p : particles do
    p.frc = 0
    p.frc += p.mass*gravity
end for
for Element e : elements do
    Matrix3x3 F = Matrix3x3 (x1-x0, x2-x0, x3-x0) * inverse(Matrix3x3(u1-u0, u2-u0, u3-u0))
    PolarDecomp (F, Q, Ftilde)
    Matrix3x3 strain = 1/2 * (Ftilde + transpose (Ftilde)) - I
    Matrix3x3 stress = lambda * trace(strain) * I + 2 * mu * strain
    for i = 0 to 3 do
        particles[e.node[i]].frc += Q * stress * e.normal[i]
    end for
end for
for Particle p : particles do
    p.vel += dt*(p.frc / p.mass)
    p.pos += dt*(p.vel)
end for
```

Bonus marks will be awarded to those who are able to 1) calculate stress using corotational strain (i.e. by implementing polar decomposition), and 2) implement another numerical integration scheme other than the one provided.