Computer Graphics: Geometry and Simulation
Coursework 3: Debugging Geometry and Simulation

Oliver Jones (s2153980)

The code listings in this report highlight the changes made to the codebase. Pink sections have been removed from the original code, while green ones have been added.

1 As-Rigid-as-Possible Deformation

The bugs in this section were all found by simply running the code and observing the incorrect output. Figure 1 shows the progression of changes as the bugs are fixed. The leftmost image shows the original output, the second shows the output after the first bug is fixed, and so on. The rightmost image shows the final output after all bugs have been fixed.

To find each bug, I carefully examined the code and compared it to the algorithm described in Lecture 15. This highlighted three bugs, the fixes for which can be seen in Listings 1 through 4. To assess the correctness of these fixes, I ran the code again and observed the output. Once the output matched the expected output, I was confident that the bugs had been fixed.

The changes shown in Listing 4 include some optimisations which do not affect the behaviour of the code, while the bug fix itself is shown in Listing 3.

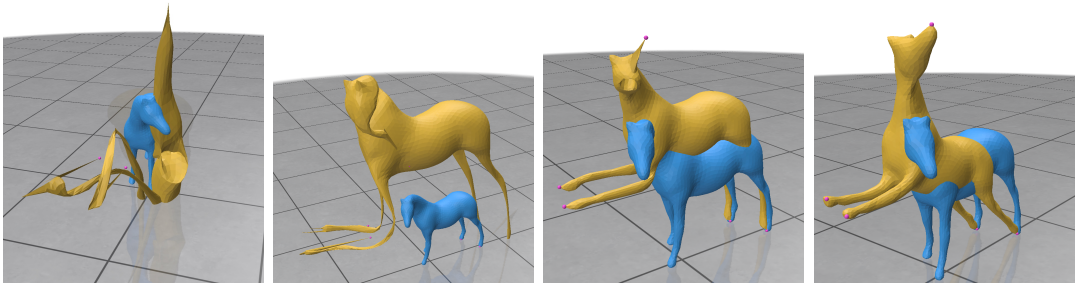

Figure 1: The progression of changes as the bugs are fixed.

1.1 Incorrect Rotation

```
g.row(j) = (origV.row(E(j, 0 1)) - origV.row(E(j, 1 0))) * REdge;
```

Listing 1: The `0` and `1` have been swapped.

1.2 Missing W

```
MatrixXd rhs = d0I.transpose() * W * (g - d0B * constPositions);
```

Listing 2: The `* W` was missing.

1.3 Incorrect Multiplication

```
S = Q P.transpose() * P Q;
```

Listing 3: `P` and `Q` have been swapped.

```
for (int k = 0; k < oneRings[j].size(); k++) {
    P.row(k) = origV.row(oneRings[j][k]) - origV.row(j);
    P.row(k) = origV.row(oneRings[j][k]) - origV.row(j);
    Q.row(k) = currV.row(oneRings[j][k]) - currV.row(j);
    Q.row(k) = currV.row(oneRings[j][k]) - currV.row(j);
}
    S = Q P.transpose() * P Q;
// Several lines ommitted for brevity
    R[j] = currR;
}
```

Listing 4: Two redundant lines have been removed, and the `for` loop has been closed much earlier. The bug fix shown in Listing 3 has also been applied.

2 Multi-body Rigid Simulation

The first bug in this section was found by running the code and observing the incorrect output. It was obvious that the forces between the spheres were reversed, and it was easy to find the bug by examining the code. Two possible fixes were identified, and are shown in Listings 5 and 6. The next two bugs were less obvious in the output, but were found by carefully examining the code.

The bug fix in Listing 7 was a clear mathematical error. To check for a collision between two spheres, the distance between their centres must be less than the sum of their radii, i.e. $d < 2r$. However, as this calculation is checking the square of the distance against the squares of the radii, the comparison should be $d^2 < 4r^2$, not $d^2 < 2r^2$.

The final bug was a simple comparison error, as shown in Listing 8. To see the effect of this bug, I changed the value of `CRCoeff` from `0.0` to `1.0`. This made it clear that the comparison was incorrect, as the spheres would not bounce when they hit the ground.

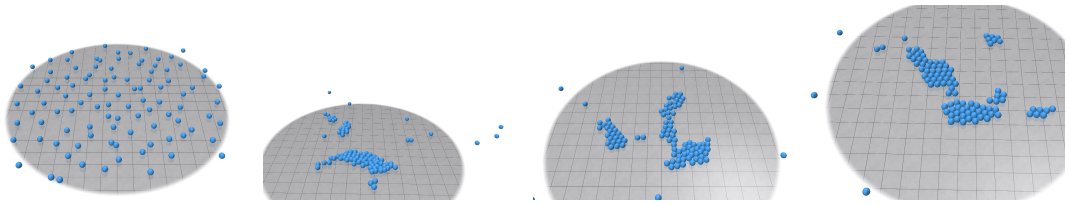

Figure 2: The progression of changes as the bugs are fixed.

2.1 Reversed Pairwise Gravity

```
forces.row(i).array() = forces.row(i) - + forceDirection *
   gravityConstant / sqrDistance;
forces.row(j).array() = forces.row(j) + - forceDirection *
   gravityConstant / sqrDistance;
```

Listing 5: The `+` and `-` have been swapped.

```
RowVector3d forceDirection = spherePoses.row( j i ) - spherePoses.
   row( i j );
```

Listing 6: The `i` and `j` have been swapped.

2.2 Incorrect Squared Radius Multiplier

```
if (sqrDistance < 2 4 * sqrRadius)
```

Listing 7: The `2` has been replaced with `4`.

2.3 Incorrect Zero Comparison

```
if (sphereVelocities(i, 1) > < 0.0)
```

Listing 8: The `>` has been swapped with `<`.