Collision Detection and Deformation of Soft Bodies

Rohan Motheram and Nathan Tribble

Abstract

This paper investigates simple methods for modeling soft body physics and deformations. Various objects were represented by a two dimensional surface mesh with a mass spring network that connects each vertex to every other vertex. Newton's laws of motion were implemented for each individual motion to compute the velocities and positions. An axis aligned bounding box tree was used to accelerate the detection of collision and basic velocity inversion was performed to compute the new velocities. Damping was also implemented to ensure that the models were stable. The results are semi-realistic with substantial increase in efficiency when computing collisions.

Key Words: deformation, soft-body, dynamics, collision

1 Introduction

1.1 Motivation

Rigid body dynamics simulations use the classical Newton's Laws of Motion to simulate forces and the motion of an object. However, one major assumption that is made is that the objects are perfectly rigid and never deform. In reality, no object is truly rigid and many objects have clear deformations when they come into contact with other objects. Soft Body Physics is an extension to the classical rigid body dynamics simulation to account for deformations in objects. The objects still follow the basic Newton's laws of motion, but deform when colliding with another object.

1.2 Recent Work

Much of the prior work in soft body physics is in modeling the deformations of the objects and also efficiently detecting collisions between multiple objects.

1.2.1 Deformation Model

Kenwright and Davison [1] present a simple method that models the entire object as a mass-spring system. Each vertex is treated as a simple point mass and is connected to every other vertex by a spring. This model only requires the surface mesh data and does not require any 3D meshing. Debunne [2] proposes an adaptive finite element method to model and compute deformations in objects. This method uses a stress and strain based approach along with the principle of minimum potential energy to find find the positions of all the nodes that minimize the overall strain energy in the model. This method offers a very high level of accuracy at the cost of being very difficult to implement and being very computationally expensive.

A unique approach presented by [3] computed the overall change in volume of the mesh after collision. The ideal gas law is then used to find the pressure distribution of the model. This approach is not considered in our implementation, but the simplification of the velocity computed presented in the paper is used to calculate velocity after collision.

1.2.2 Collision Detection

Classical Rigid body dynamics simulations use pre-computed data structures to determine when there is a collision. However, these data structures assume that the object never changes shape throughout the simulation. Once an object deforms, the data structure is no longer valid and needs to be updated. If the process of constructing the data structure is not efficient, the simulation can become very slow, especially when objects are continuously deforming. Therefore, data structures that can be constructed efficiently while still allowing for efficient collision detection are necessary. Larsson

[4] proposes the axis aligned bounding box tree. This data structure consists of a single bounding box that is split into multiple subvolumues. All of the geometric primitives that make up the object are assigned to a leaf node of the tree based off the centroid of the primitive. This paper also presents many different methods for tree construction and traversal, including top-down traversal and bottom-up traversal. When the triangles within two leaf nodes are checked for intersection, an efficient triangle intersection algorithm, proposed by Moller [5], allows for fast determination of whether to triangles intersect.

Figure 1: Post-Collision Response of two bunnies

2 Deformation Model

Given a 2D triangular mesh, a good deformation model should be able to efficiently compute deformations while also looking visually accurate. The deformations do not need to be numerically accurate but they should look visually convincing. Since we only have a 2D mesh of an object, our choices for modeling deformations were limited to just the simple mass spring approach, proposed by Kenwright [1]. This model does compromise on efficiency and visual accuracy, but it is the simplest to implement. Any other models researched would require a significant amount of time while also requiring a 3D mesh. Each vertex of the model is connected to an ideal, linear spring with a spring constant k. The force exerted by any individual spring connected to vertices v_1 and v_2 can be computed using Hooke's Law, presented in equation 1, where d_{new} is the new length of the spring, d_{old} is the original length of the spring, and $\hat{\mathbf{n}}_{v_1-v_2}$ is the normalized vector from v_1 to v_2.

$$F_s = -k(d_{new} - d_{old})\hat{\mathbf{n}}_{v_1-v_2} \tag{1}$$

3 Damping Correction

In continuous time, this ideal spring model does not lose energy over time, resulting in vertices oscillating infinitely once disturbed from equilibrium. However, our simulation is in discrete time, with objects being updated in small time steps. For this reason, the model becomes completely unstable due to solver inaccuracies. Figures 2 and 1 shows an example of such instability. Two bunnies collide and bounce off. Figure 1 shows the bunnies shortly after bouncing. Figure 2 shows the bunnies a long time after bouncing.

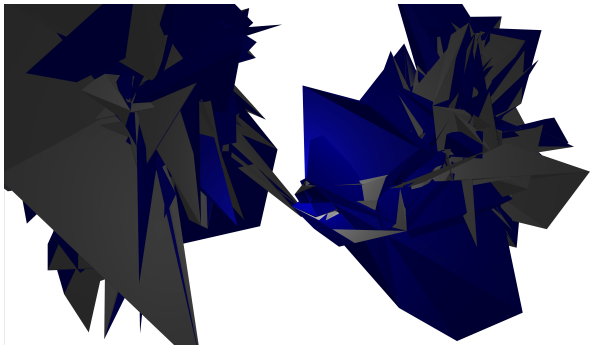

Figure 2: Post-Collision Response of two bunnies

This issue can be mitigated by introducing damping into the model. Each vertex of the object will experience a force that is directly proportional to the velocity of the vertex, denoted in equation 2

$$F_D = -c\mathbf{v} \tag{2}$$

This force always opposes the motion of the vertex and causes the motion to decay over time. However, the motion can still go unstable if the internal spring forces are too strong. For this reason, a sufficiently high enough damping constant c needs to be chosen such that the object will slowly return to its equilibrium position after being disturbed from its equilibrium position. However, the damping also cannot be too large, as it will reduce the visual appearance of deformations. Additionally, having a damping coefficient that is way too large can result in errors like the example shown in Figure 3.

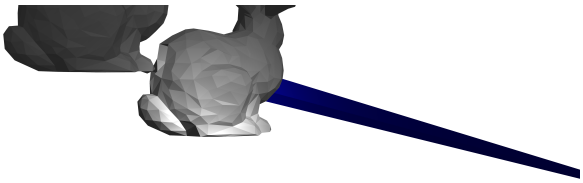

Figure 3: Post-Collision Response of two bunnies - High Damping

In this figure, a damping coefficient of 2000 was used to demonstrate the effects of excessive damping. When the twoo bunnies collide, the change in velocity results in an extremely large damping force that is enough to make the vertices of contact move far off-screen in just one time increment. This is an effect that occurs due to the simulation being in discrete time. In continuous time, the objects will decay, no matter how large the damping coefficient is.

Overall, the amount of damping that is necessary is heavily depended on the objects that is being simulation. Determining the correct damping to use for each object would require a large amount of experimentation, making it a very time consuming process. For this reason, an adaptive damping model was chosen, shown in equation 3.

$$F_D = -min(\frac{||\Sigma F_s||}{10}, 10) * c\mathbf{v} \tag{3}$$

This adaptive model multiples the damping force by the minimum of the magnitude of $F_s/10$, or 10. This allows for damping to be scaled as necessary when the net force experiences is larger, but also prevents the damping force by being scaled by more than a factor of 10 to prevent damping from becoming too large. The damping constant c is still present in the equation and is used to specify the general strength of damping in the simulation.

4 Collision Detection

In order for our simulation to run smoothly, we need a method to both determine if a collision is present and which vertices of a the objects are colliding. We chose to implement the axis aligned bounding box tree method proposed in Larsson [4].

For each object loaded into the scene, an AABB tree was constructed for that object's mesh. For simplicity each non leaf node always has two children, making the tree perfectly balanced. The root node of the tree consists of the bounding box that encloses the entire mesh when it is loaded into the scene. This root bounding box is then split into multiple hierarchy levels.

When splitting a node, the longest axis was split at the center of the bounding box. Two new nodes are created to represent the left and right half. The next hierarchy level is created by splitting all of the leaf nodes present in the tree.

After all the hierarchy levels are created, all of the triangles that make up the mesh are assigned to one of the leaf nodes in the tree, based off the centroid of the triangle. The resulting tree and triangle assignment forms the structure of the tree for the entire simulation. Since the tree is perfectly balanced and has $log(n)$ levels, the entire tree will ideally have $2n$ nodes, with the lowest level having n nodes. Constructing the entire tree will take $nlog(n)$ time.

During each time increment of the simulation, the bounding boxes of every node in the tree are updated to reflect the new positions of all of the primitives in the mesh. A node is updated by computing the bounding box contained by all of the triangles assigned to all the subsequent children of that node. The tree structure itself and the triangle assignment remains the same. For each level of the tree that needs to be updated, the simulation must loop through all of the triangles in the mesh. For a perfectly balanced tree (which is always the case for our simulation), there will be at most $log(n)$ levels in the tree, resulting in an $O(nlog(n))$ time complexity for updating the tree.

To check for collisions between two objects, the root node of both trees are checked for bounding box intersection. If the bounding boxes intersection, then all of the immediate children are checked with each other for collision. This process repeats recursively until two intersecting nodes are both leaf nodes. When both nodes are leaf nodes, all of the triangle in each node are tested for intersection. If any triangles intersect, all the vertices that form the triangles are added to a set of all intersection vertices.

Ideally, only small portions of two objects are intersecting with each other. For the case when one leaf node is barely intersecting a leaf node in another tree, checking for collision will take $O(logn)$ time. However, if an object fully contained within another objects and all vertices intersect, then checking for collisions will take $O(n^2)$ time. Assuming that proper corrections are performed (discussed in section 6) to ensure objects do not continue to overlap after a collision is detected, this scenario will not happen and therefore does not need to be accounted for.

For simplicity, this model assumed that all vertices part of an intersecting triangle are intersecting with the other object. This simplification does slightly reduce the visual accuracy of the deformations, but is acceptable for fine enough meshes.

5 Velocity Computation and Post-Collision Correction

For our simulation, we implemented a simplification of the velocity computation approach proposed by [3]. Instead of finding the normal and tangential components of the velocities, the the velocities for all colliding vertices were simply inverted and multiplied by an elasticity constant.

6 Post-Collision Correction

In real lift, impact occurs at the instant that two objects collide. However, our simulation is in discrete time and objects are checked for collisions in small time increments. For this reason, when a collision is detected between two objects, a small portion of one object may already be inside the other object. An example of this occurring with two intersecting bunnies is shown in figure 4, where a portion of the feet on the bunny on the left is inside the bunny on the right.

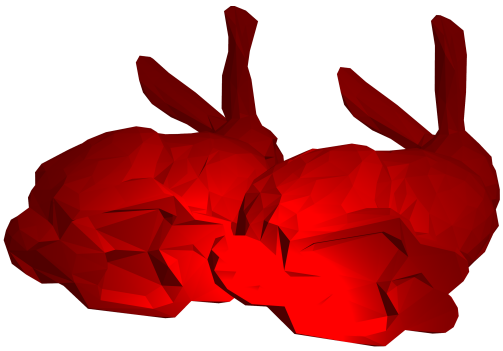

Figure 4: Overlapping of two bunnies

The major issues with this are that it slows down the simulation since more triangles have to be checked for intersection and also that velocity computation does not work properly if the elasticity is less than one. To account for this issue, the previous position of all vertices are tracked throughout the simulation. When a collision occurs, all of the intersecting vertices are backtracked to their previous position, before they were in collision with another object.

7 Newton's Laws of Motion

The total net force on any given mesh node is equal to the sum of the spring forces and the damping force, shown in equation 4

$$\Sigma F = m\frac{d^2x}{dt^2} = \Sigma F_s + F_D \tag{4}$$

This resulting differential equation can be solved using any numerical ODE solver. We chose Verner's 8th order Runge Kutta method to minimize the amount of instability in our simulation.

8 Implementation and Testing

For the implementaion, a basic OpenGL Renderer was used to visualize the various objects on screen. Mesh data was stored in separate files containing the vertex, edge, and face data. Currently, all test cases involve "hard-coding" which files to load and the different constants to use. All test cases were run on an Intel i9-11900K and an Nvidia GeForce RTX 3090 Graphics Card.

9 Results

For all test cases, a maximum tree depth of 12 was chosen and a damping constant of 1 was chosen along with an elasticity of 0.9.

9.1 Simple Sphere Collision

Our first test case consisted of two spheres with a uniform spring constant of 40, a mass of 1, a damping constant of 1 and moderate initial velocities. The two spheres move towards each other and collide. The resulting deformations from the collision are shown in figures 5 and 6.

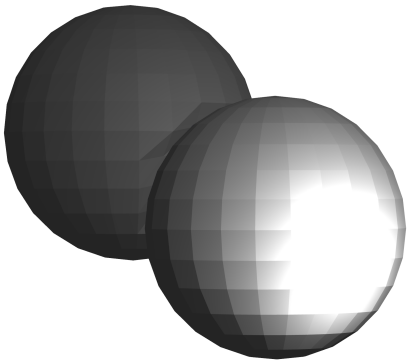

Figure 5: Deformation of Simple Spheres

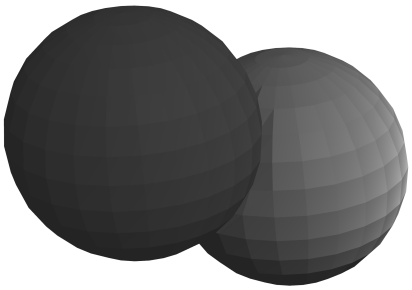

Figure 6: Deformation of Simple Spheres - Another View

Judging from the results, both spheres experience a similar amount of deformation after collision. The deformations also look visually convincing and the similar deformations make sense since both spheres have the same stiffness.

9.2 Sphere Collision - Differing stiffness

The second test case consisted of one large sphere with a uniform spring constant of 5, and a second sphere with a uniform spring constant of 100. Both spheres have a mass of 1, and a damping constant of 1. Similar to the first test case, both spheres start with a moderate initial velocity and move towards each other. The resulting deformations are shown in Figure 7.

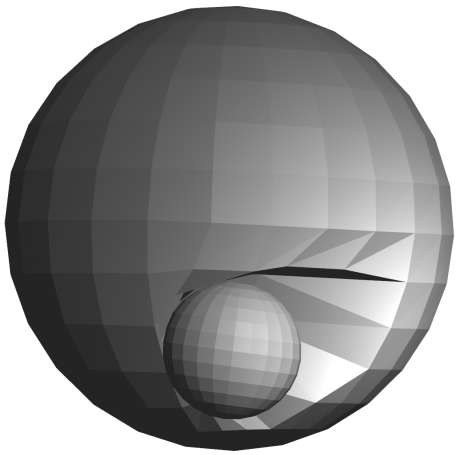

Figure 7: Deformation of large soft sphere

The results depict a substantial deformation for the large sphere and minimal deformation for the small sphere. The results accurately depict the differences is stiffness between the very soft large sphere and the very hard small sphere.

9.3 Sphere and Bunny Collision

Our third test case consisted of a sphere and a bunny with a uniform spring constant of 40, a mass of 1, and a damping constant of 1, with the same initial velocities as the previous two test cases. The deformations of the sphere and the bunny are shown in Figures 8 and 9, respectively

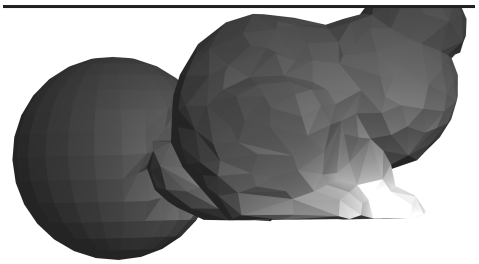

Figure 8: Deformation of the sphere

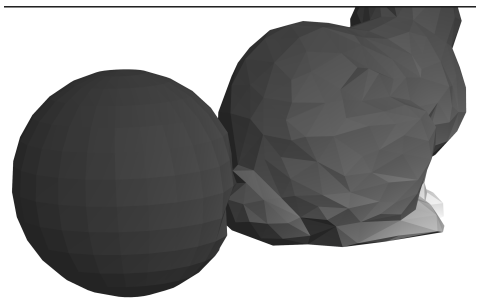

Figure 9: Deformation of the bunny

The results depict a somewhat realistic inwards deformation of the sphere and minor compression of the bunny's tail. The simulation was also able to run efficiently while the objects were colliding.

9.4 Bunny and Teapot Collision

Our fourth test case consisted of a bunny with an uniform spring constant of 40 and a teapot with a uniform spring constant of 10.

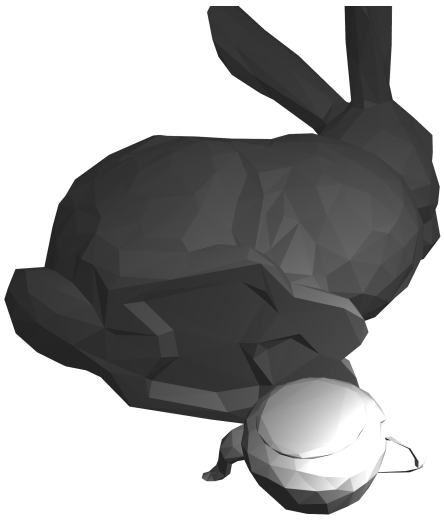

Figure 10: Bunny and Teapot - Before collision

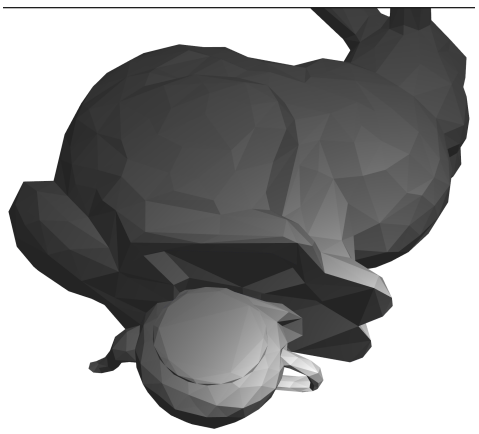

Figure 11: Bunny and Teapot - After Collision

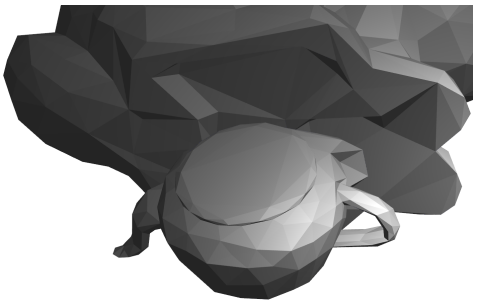

Figure 12: Bunny and Teapot - After Collision (Closer view)

The results indicate a small (but noticable) deformation for the small bunny and a larger deformation for the teapot, which correctly reflects the differences in spring constants for each material. However, the defotmation of the teapot is somewhat inaccurate since the vertices that initially contact the bunny are shifted to the right after the collision, likely due to the simplification in velocity computation. The bunny deformation looks plausible and the contacting vertices deform inwards, which is expected.

9.5 Multiple Objects

Our final test case involves three sphere with varying sizes. All spheres have a uniform spring constant of 40 and a damping coefficient of 1. The sphere on the top has a much higher initial velocity than the other two spheres. Figures 13, 14 and 15 show the initial position of the spheres, the deformation of the top sphere after colliding, and the deformation of the right sphere after colliding.

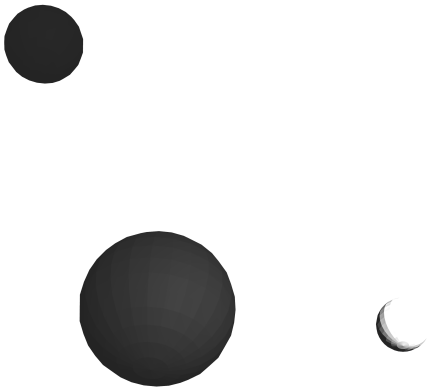

Figure 13: Initial position of spheres

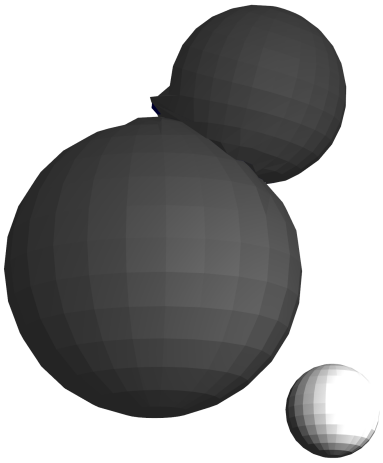

Figure 14: Deformation of top sphere

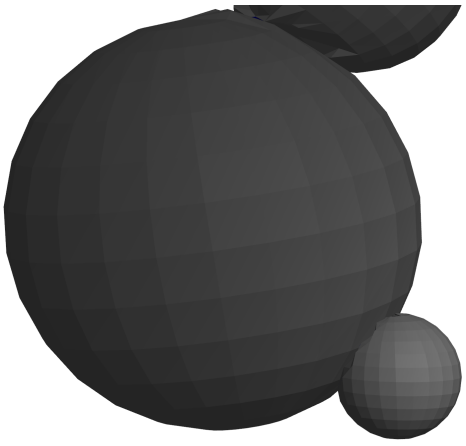

Figure 15: Deformation of right sphere

Judging from the results, the top sphere deforms in a way that is somewhat unrealistic. This is most likely due to the simplification that was made when computing velocity after impact. However, the large sphere deforms in a visually convincing manner when colliding with the right sphere. The simulation is able to run efficiently even with three objects that each have over 600 triangles.

10 Discussion

Overall, all of our results ran efficiently, with many of them having semi-realistic results. When running the simulation with a brute-force triangle to triangle intersection without an AABB tree (equivalent to having an AABB tree with just one root node), the simulation dropped to about one frame per second when any two objects got close to each other. The AABB tree almost entirely removed this slowdown experienced and still ran great even with more than two objects colliding at the same time. However, our test cases had at most 3 objects present at the same time and the efficiency could very possibly drop if a large amount of objects are being simulated at the same time. The velocity computation is somewhat inaccurate and is visibly clear for the last test case. However, the objects deform in a manner that is somewhat realistic when the impact velocities are reasonable.

11 Limitations and Future Work

Our simulation is intended to be a basic, preliminary implementation of soft body physics and deformation. Due to limitations in time, simplifications were made to the implementation.

11.1 Deformation Model

All of our objects in the simulation are represented by a 2D surface mesh. This is one of the biggest limiting factors that resulted in us choosing a simple mass spring system with each vertex connected to every other vertex. Any other deformation model would require a 3D mesh of the objects. Overall, this limitation does not allow us to represent how an inside of an object deforms, limiting the accuracy of this model. Future work would involve implementing full 3D triangulation of models along with implementing the voxelation technique, proposed by Kenwright [1].

Another limitation in our model is that all of the objects are assumed to have a uniform stiffness and elasticity. Additionally, the mass is also assumed to be the same for all mesh nodes as well. For future simulations, implementation and testing of variable-stiffness materials would also be investigated.

11.2 Collision Detection

Currently, our implementation uses a binary AABB Tree to determine collisions. Additionally, we only implemented a top down tree traversal to check for collisions. This implementation did work fine for all of our test cases, but could potentially become inefficient for scenarios involving many different objects colliding with each other simultaneously. The most objects present in our simulation was three. Future work would involve implementing an AABB tree with four or eight children and possibly implement a bottom-up tree traversal, as presented in [4]. Additionally, some of the better tree splitting algorithms could also be implememnted

12 Conclusion

Our project provides a preliminary experiment with various collision detection and impact computation methods. The result are sufficiently efficient, given our test cases. and also provide a good baseline for further improvement.s Future work would aim to improve the tree structure of the model while also implementing full 3D deformation models.

References

[1] B. Kenwright, R. Davison, and G. Morgan, "Real-time deformable soft-body simulation using distributed mass-spring approximations," in *CONTENT, The Third International Conference on Creative Content Technologies*, 2011.

[2] G. Debunne, M. Desbrun, M.-P. Cani, and A. Barr, "Adaptive simulation of soft bodies in real-time," in *Proceedings Computer Animation 2000*, pp. 15–20, IEEE, 2000.

[3] H.-H. P. C. C. North, "The annual sigrad conference special theme–real-time simulations november 20–21, 2003 umeå unversity, umeå, sweden, conference proceedings," pp. 29–34, 2003.

[4] T. Larsson and T. Akenine-Möller, "Collision detection for continuously deforming bodies.," in *Eurographics (Short Presentations)*, 2001.

[5] T. Möller, "A fast triangle-triangle intersection test," *Journal of graphics tools*, vol. 2, no. 2, pp. 25–30, 1997.

13 Appendix - Work Distribution

The major work in research and algorithm implementation was done by Rohan.

Testing and capturing results were done by Rohan

This project uses the base rendering code provided by Barb Cutler.

The sphere mesh files were created by Nathan.

Most parts of the presentation and report were written by Rohan, with minor edits done by Nathan.