JACOBS
UNIVERSITY

Efficient Collision Detection in a Scene with Static and Dynamic Objects (and its Application to Chewing Simulation)

by

Muhammad Omer Saeed

Bachelor Thesis in Computer Science

Prof. Dr. Lars Linsen
Supervisor

Date of Submission: May 16, 2014

Jacobs University — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicates resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Signature Place, Date

Abstract

The importance of computer graphics lies in its applications. In engineering applications (e.g. automotive and aerospace), the ability to quickly visualize newly designed shapes is indispensible. Before the advent of computer graphics, designers built expensive prototypes and time-consuming clay models. Now, designers can interactively view and modify models of their shapes using a computer.

Medical imaging is another application where computer graphics has proven valuable. Recent advances in imaging technology such as computer tomography and magnetic resonance imaging allow physicians to take 3D X-rays of the human body. Interactive computer graphics allows the physician to interpret this large volume of data in new and useful ways.

What is common in all the applications of computer graphics is the notion of reality in the simulations. If the computer graphics do not potray the real world, they cannot be successfully used in the applications described. One of an important feature of reality in computer graphics is that objects do not pass through each other i.e, there is some sort of response when two or more objects collide. This part of the simulation is known as the collision detection mechanism and is computationally expensive as the collision detection has to be performed at each rendering of the frame of the simulation. Thus, the efficiency and the accuracy of the collision detection mechanism is not only essential but is required.

Many methods have been proposed for the collision detection mechanism in computer graphics. These include the distance field computations of the scene, breaking the scene into a tree and then following that branch of the tree which is potentially involved in the collision, creating bounding volume hierarchy of the scene to reduce the number of operations for checking the collision and more.

This paper describes an approach of how a variety of methods can be used together to provide an efficient collision detection mechanism in a scene containing static and dynamically moving objects.

Contents

1 Introduction

1.1 Collision detection

Collision detection is a geometric problem. In a scene with two or more objects with given configuration, the problem is to determine if the objects intersect with each other or not. In addition to determining whether two objects have collided, collision detection system also report the set of intersecting points. If there there is not a collision , the collision detection systems can report the distance to the nearest object. A collision detection system is composed of a choice of object representation in the form of a data structure and a choice of a detection algorithm which reports a collision when the distance between two objects falls under a certain threshold.

1.2 Problem definition

This thesis explores the efficient collision detection algorithm with respect to the chewing simulation of the human teeth model and the gummy bear substrate. The human teeth model consists of the upper jaw and the lower jaw. The whole scene of the simulation is scaled and contained in a unit cube which is placed at the center. The scene configuration is shown in the figure below.

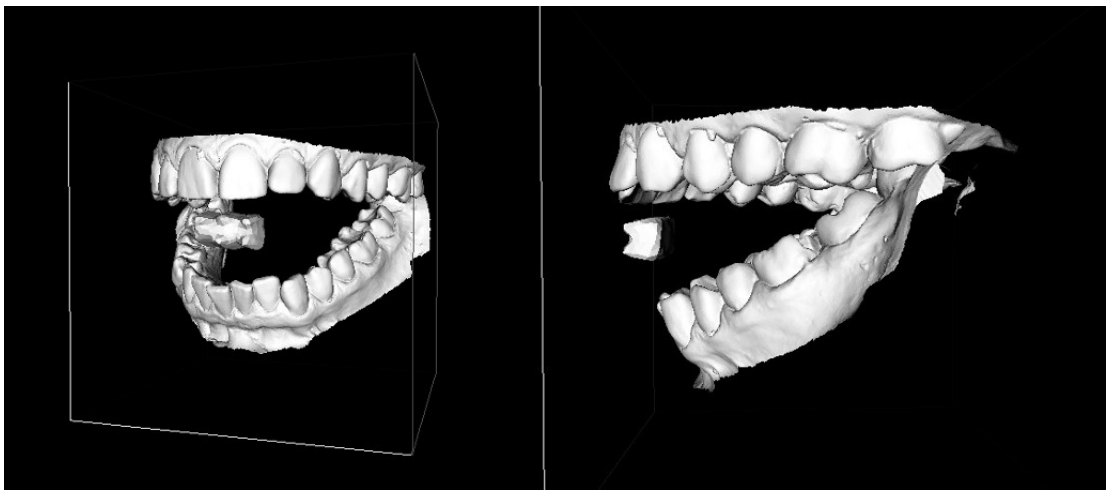

Figure 1: Scene configuration of the chewing simulation from different angles

The upper jaw is static i.e, it does not change position or orientation during the simulation. The lower jaw is dynamic and rotates from 0 to -30 degrees and back along the x-axis.The gummy bear substrate is also dynamic and it can change its position as well as its orientation during the simulation. The task is to provide an efficient collision detection mechanism for the simulation to be real time i.e, without any lag.

2 Background and literature review

2.1 Distance fields

A distance field (also known as a distance map) is a gridded structure where each cell in the grid represents the shortest distance from that cell to the nearest polygon or object. In other words, it is a label for each cell that has a value of a nearest object with a desired property defined by the map. The distance field has been found to be a useful construction within the areas of computer vision, physics, and computer graphics.

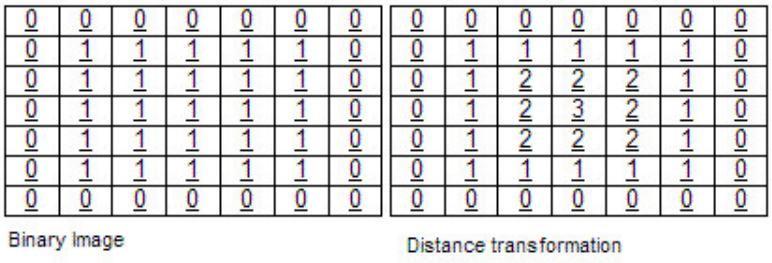

Figure 2: Left: Binary image, Right: distance to the nearest obstacle pixel in form of a distance map
Source: http://en.wikipedia.org/wiki/File:Distance_Transformation.gif

Distance fields can be used as a mechanism to provide rapid collision detection in computer graphics. For this purpose we use the signed distance field for the polygonal mesh. A positive distance value represents the point exterior to the mesh and a negative distance value represents a point interior to the mesh [5].

Mathematically the distance field D of a mesh surface S can be represented by a scalar function:

$$D : \mathbb{R}^3 \rightarrow \mathbb{R}$$
$$\text{where, } D(p) = min_{q \in S}\{|p - q|\}, \forall p \in \mathbb{R}^3$$

If the mesh surface is closed and the distance field is signed, then the sign function of the distance field for any point in the field can be defined as:

$$sgn(p) = \begin{cases} \text{-1} & \text{if p is inside the surface} \\ 1 & \text{if p is outside the surface} \end{cases}$$

The distance to the closest object in the distance field can be calculated in various ways which depends on the application of the distance field. Some of the common ones are[6]:

- Manhattan distance
- Eucledian distance
- Chebyshev distance

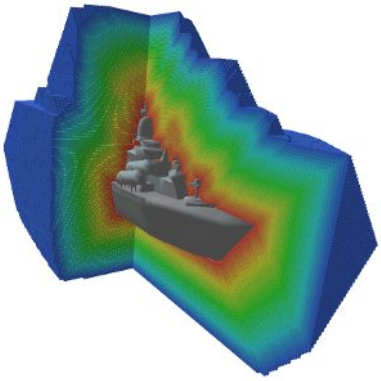

Figure 3: Visualization of a distance field, the changing colors represent the increasing distance from the object.
Source:
http://lgdv.cs.fau.de/uploads//publications/images/distanceTransform_Teaser.jpg

2.2 Bounding volume hierarchies

A bounding volume hierarchy is a tree structure on a set of geometric objects. All geometric objects are wrapped in bounding volumes that form the leaf nodes of the tree. These nodes are then grouped as small sets and enclosed within larger bounding volumes. These, in turn, are also grouped and enclosed within other larger bounding volumes in a recursive fashion, eventually resulting in a tree structure with a single bounding volume at the top of the tree. Bounding volume hierarchies are used to support several operations on sets of geometric objects efficiently. Bounding volume hierarchies are generally referred to as BVH[10].

Some of the desirable characteristics of BVH are:

- efficient intersection test
- tight fittng of the complex objects
- effecient generation and update in case of object transformation
- memory efficiency

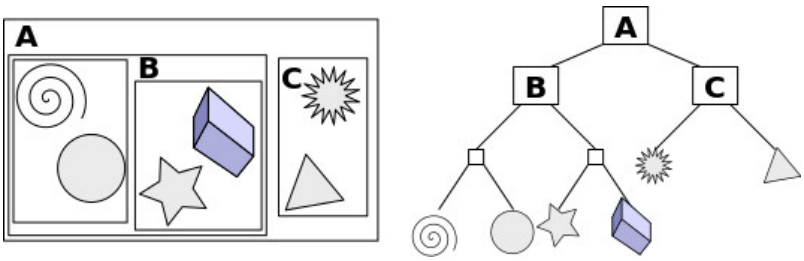

Figure 4: Complex geometry is bounded by simple rectangles.
Source: http://upload.wikimedia.org/wikipedia/commons/2/2a/Example_of_bounding_volume_hierarchy.svg

Bounding volume hierarchies are widely used for collision detection purposes. BVH allow the early rejection of large amount of complex gemoetry by simple intersection tests. If a complex geometry is contained inside a sphere, a simple check can be made to check if there is an intersection with the sphere. If there is an intersection we need to do the test with bounding volumes lower in the heirarchy otherwise we can reject the possibility of collision for the whole complex geometry thus saving computational time.

2.3 KD - Tree

A Kd - tree is a binary tree data structure which is used to store the geometry of a 3D scene . The Kd tree can be built starting from the cell that spans the whole 3D scene, then iteratively partitioning the current cell into two subcells by an optimal cut orthogonal to the coordinate axes. The optimal cut is made in such a way that the geometery is splitted into two halves. This process is carried again and the recursion is stopped when the geometery per cell is below a threshold or the maximum depth of the hierarchy is reached [3].

The following figure shows the sections of a cube partitioned while generating a Kd - tree.

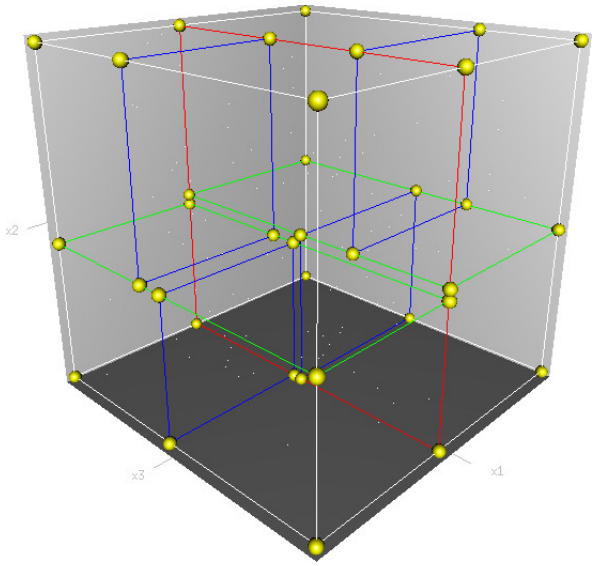

Figure 5: Iterations of creating a kd tree from complex geometry in a cube
Source: http://upload.wikimedia.org/wikipedia/commons/b/b6/3dtree.png

Kd trees are widely used for collision detection in computer graphics. The main idea is to start with the root node and then test which node will possibily collide, the nodes are followed until we reach the leaf node. The geometery at the leaf node is then tested for the collision detection and in this way a lot of geometery is discarded early and does not have to be checked with the colliding object.

3 Investigation and implementation

3.1 Calculation of the distance fields for the teeth model

The algorithm for fast calculation of distance fields was proposed by Arnulph, Gerrit and Clemens [1]. The algorithm takes the triangular mesh and the face normal for each triangle as an input to calculate the distance field. We describe our approach based on this algorithm.

The distance field gird is stored as a 3D array of points in our approach. Initially the distance at each point of the grid to the closest object is set to a maximum value.

For every triangle of the mesh, a prism is calculated by moving the vertices of the triangle in the positive and negative direction of the face normal by an amount ϵ. The value of ϵ depends on the object against which we want to check the collision. Our susbstrate object is a gummy bear inside a bounding volume hierarchy, so we take ϵ as the radius of the root bounding sphere of this hierarchy so that we can detect the collision when the bounding sphere enters the collision possible region. For each calculated Prism, an axis aligned integer bounding box is calculated. After calculating the bounding box, for all the points of the grid which fall inside this bounding box, the shortest distance to the triangle is calculated. For calculating the shortest distance between a point and a triangle, please refer to the 3.2 section of this paper. If the calculated distance is less than the distance already stored at that point, the distance at that point is updated to the new value.

As our mesh has a closed surface, we also need to calculate the sign function at each point which would tell us whether the point is inside or outside the mesh. The sign at each point is calculated from the sign of the inner product of the face normal of the triangle and the direction vector from the point to the triangle. Thus, the triangle plane divides the bounding box volume into a positive and a negative part[7].

The complexity of this algorithm is $O(nm)$ where n is the number of triangles in the triangular mesh and m is the number of points in the grid.

The pseudo code of the algorithm is as follows:

Algorithm 1 Fast distance field calculation

```
1:  procedure CalculateDistanceField(Triangular_mesh M, Distance_field F)
2:      F ← + ∞
3:      ε ← Radius of the root bounding sphere of the substrate
4:      for all Triangle t ∈ M do
5:          calculate Prism P ← Prism(t , t.face_normal , ε)
6:          Compute Axis aligned Bounding box B_P ← BoundingBox(P)
7:          for all p ∈ B_p ∩ F do
8:              distance ← PointToTriangleDistance(p,t)
9:              if distance < abs(p.distance) then
10:                 p.distance ← distance · sign(p)
11:         end for
12:     end for
```

The following figures show the sign function result of applying this method to our teeth

model:

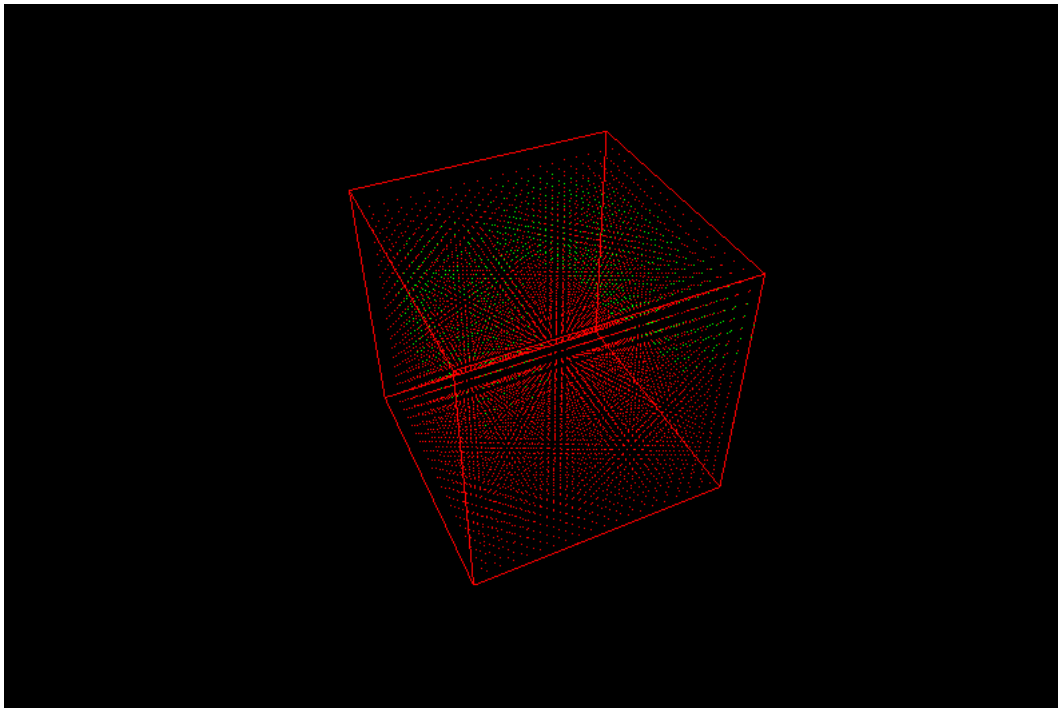

Figure 6: Distance field of the upper jaw. The green points are inside the mesh and the red points are outside

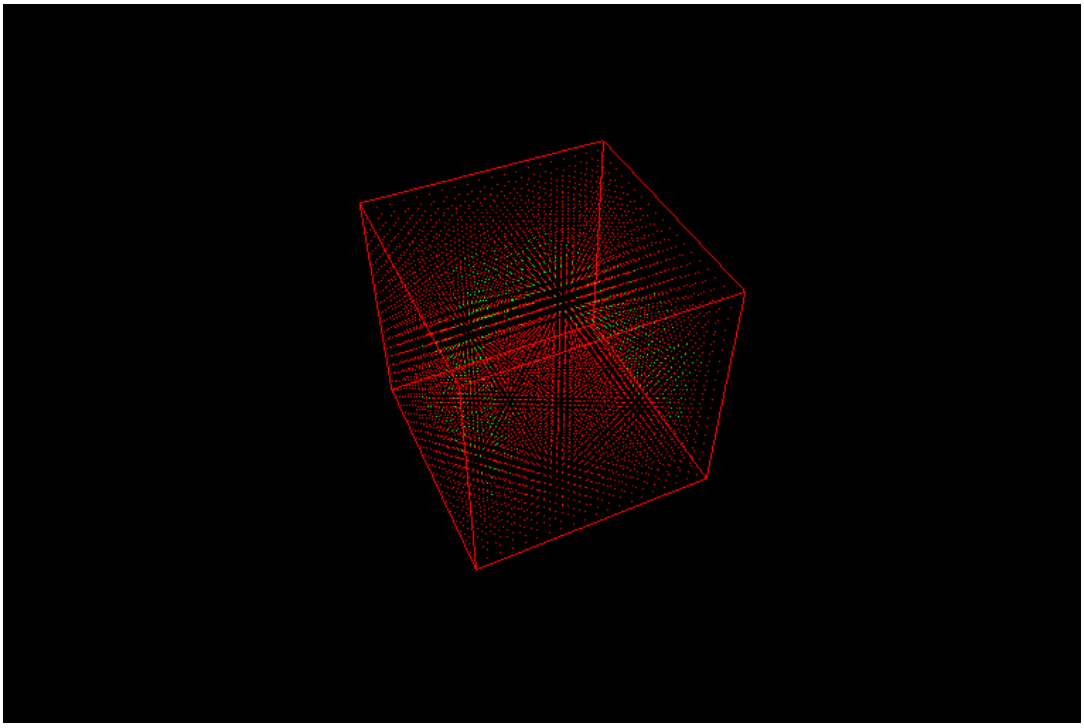

Figure 7: Distance field of the lower jaw. The green points are inside the mesh and the red points are outside

3.2 Calculation of the distance between a point and a triangle

The method for calculating the distance between a point and a triangle in 3D was proposed by Eberly [2].

We are looking to calculate the minimum distance between a point $\mathbf{P}$ and a Triangle

$$\mathbf{T}(s,t) = B + sE_0 + tE_1 \text{ for } (s,t) \in D = \{(s,t) : s \in [0,1], t \in [0,1], s+t \leq 1\}$$

The minimum distance is calculated by locating the values $(\bar{s}, \bar{t}) \in D$ corresponding to the point on the triangle closest to $\mathbf{P}$.

The squared-distance function for any point on the triangle to $\mathbf{P}$, Q(s,t), is quadratic in s and t.

$$Q(s,t) = as^2 + 2bst + ct^2 + 2ds + 2et + f$$

Where a = $\mathbf{E_0}.\mathbf{E_0}$, b = $\mathbf{E_0}.\mathbf{E_1}$, c = $\mathbf{E_1}.\mathbf{E_1}$, d = $\mathbf{E_0}.(\mathbf{B}-\mathbf{P})$ and f = $(\mathbf{B}-\mathbf{P}).(\mathbf{B}-\mathbf{P})$.

The gradient of Q is zero only when,

$$\bar{s} = (be - cd)/(ac - b^2)$$

$$\bar{t} = (bd - ae)/(ac - b^2).$$

If $(\bar{s}, \bar{t}) \in D$ then we have found the minuimum of Q thus our answer. Otherwise the minimum must occur on some boundary of the triangle. The following figure shows different regions of the boundaries:

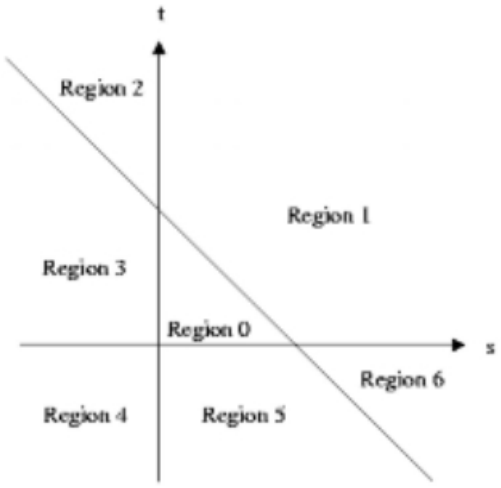

Figure 8: Partitioning of the st-plane by triangle domain D

If $(\bar{s}, \bar{t}) \in D$, we belong to region 0 and we have found the parameters to calculate the minimum distance of the point to the triangle. Otherwise, we are in one of the other regions. The algorithm presented handles all the cases, for further reading on this method and how all the other regions are handled, please refer to the original paper.

.

Let V1, V2 and V3 be the three vertices of the Triangle, the algorithm is as the following:

Algorithm 2 Fast point to triangle distance in 3D

```
1: procedure POINTTOTRIANGLEDISTANCE(POINT P, TRIANGLE T)
2:     distance = CalculateDistance(p, t.V1, t.V2, t.V3)
3:     return distance
```

For the *CalculateDistance* method, please refer to section 6.1 in Appendix.

3.3 Bounding volume hierarchy for the gummy bear substrate

The substrate against which we need to check the collison has 231 vertices and 458 faces. If we go through each of the triangle and try to find an intersection during the collision test, it could be computationally expensive. We need to discard as much geometry as possible before making any intersection tests. For this purpose we calculate the bounding volume hierarchy for the gummy bear substrate.

We calculate the axis aligned bounding box for the gummy bear. The axis aligned bounding box is calculated by finding the maximum and the minimum coordinate of the gummy bear. The axis aligned bounding box is then placed inside a sphere which is calculated such that the box fits tightly in the sphere.

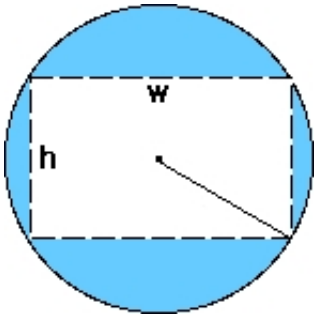

Figure 9: Tightly fitted axis aligned box inside the sphere shown in 2D

The radius of the sphere is found by calculating the distance from the center of the box to any of the vertices. The sphere is then placed at the ceter of the box with this radius.

The axis aligned box is further divided into two parts, the box can be either divided using a horizontal or vertical plane depending on the which is longer in length. The sphere is then again calculated for each of the bounding boxes. The hierarchy can be calculated as deep as desired recursively. The bounding volume hierarchy thus builds a binary tree [4][11][9].

For our project we chose to calculate until depth 3 as at the leaf of the hierarchy we already are left with upto 30-40 triangles.

The reason why we are placing the axis aligned bounding box is a bounding sphere is that using the sphere we can check very cheaply if there is a possibility of a collision or not. Once we find there is a collision, we know that all the geometry is contained inside the axis aligned bounding box and we can proceed with our further checks.

Suppose that a sphere has radius r. We check the distance from the nearest geometry at center of the sphere from the calculated distance field. Suppose that we fidn this distance as d. Then there is a possible collision with the substrate if,

$$d + r \leq 0$$

If this condition is not met, we can immediately know that there is no possibility of collision with the substrate but the value $d + r$ gives us the distance of the bounding sphere to the nearest object.

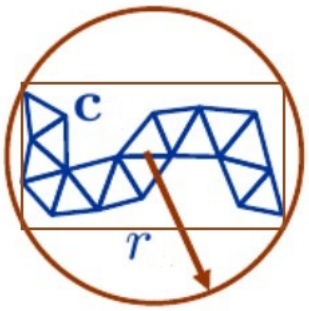

Figure 10: Complex geometry in an axis aligned box further bounded by a sphere with radius r

In the other case if the condition is met, we know there is a possibility of collision and we need to go deeper in the hierarchy, checking intersection with the next smaller sphere and we do this until we reach the leaf of the hierarchy tree, at this stage we can perform a ray triangle intersection test with the geometry at the leaf and find the intersection coordinates.

3.4 Rapid collision detection

After calculating the distance field of the teeth model and the bounding volume hierarchy for the substrate, what is left is to be able to use these to provide an efficient way to check the collision of the substrate against the teeth model.

3.4.1 Collision check against the upper jaw

The upper jaw is a static object i.e, it does not change its position or orientaion during the simulation. Our $\mathbf{checkCollision}$ method takes three arguments. The $\mathbf{point}$ at which the collision is to be checked , the 3D array containing the stored $\mathbf{distance\ field}$ and the $\mathbf{resolution}$ of the distance field. As the distance field is a discrete field, the main challenge here is to map the point that we have to check to the nearest point in the distance field from which then we can check the stored distance at that point.

As our scene is centered at origin and is kept inside a cube of unity length, we can achieve this by following this block of code. The idea is to use the distance between two points in the grid to find an offset in the distance field closest to the point.

Let X,Y and Z be the three values of the coordinates of the point.

1: **procedure** CHECKCOLLISION(POINT $point$, DISTANCE_FIELD $Field$, INTEGER $resolution$)
2: $x = (int)((point.X + 0.5)$*resolution)
3: $y = (int)((point.Y + 0.5)$*resolution)
4: $z = (int)((point.Z + 0.5)$*resolution)

The integer values x, y and z are the mapped value to the distance field grid, this point is marked as a cross on the left end vertex of a cell of the distance field grid in the figure below:

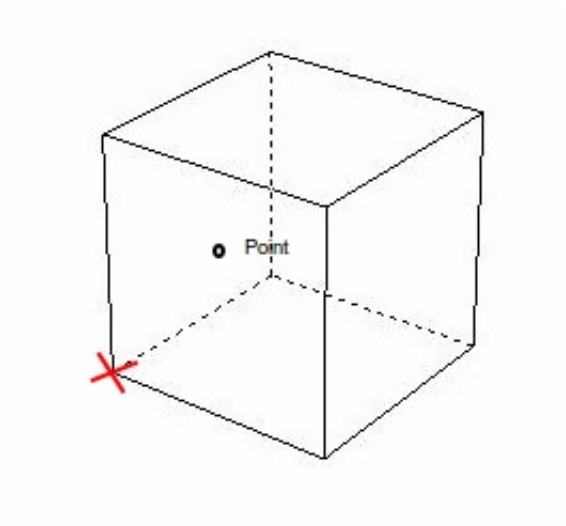

Figure 11: A single cell of the distance field grid showing the mapped coordinate and the original point position.

Now that we have the mapped coordinates to the distance field grid, we can find the distance values at all the other vertices of the grid cell containing this point. Finding the final distance value at the point is now trivial, we can use two methods to achieve this depending on our application. These are as follows.

- using tri linear interpolation to find the distance at the point inside the cell using the values at the vertices of the cell
- using the least of the stored distances at the vertices of the cell

As our teeth model is irregular, we use the second method of choosing the least distance of the vertices of the cell to check for the intersections, as it will give us better results.

After finding the distance at this point, as described in the previous section, we can conclude that whether there is a possibility of a collision or not. If there is a possibility of collision, we go deeper in the hierarchy until we reach the leaf. We can then perform the ray triangle test to find the interection points with the geometry.

3.4.2 Collision detection against the lower jaw

The lower jaw is a dynamic object i.e, it can change orientation during the chewing simulation. It rotates around the x-axis from 0 to -30 degrees and back. Although the lower jaw is changing its orientation during the simulation run, the distance field calculated for the lower jaw is only valid for the initial position of the lower jaw.

We can find a turn around for this situation, instead of rotating the whole distance field at each frame which can be computationally costly, we can rotate the point which we want to check the distance at in the opposite direction to that of which the lower jaw has rotated from the initial position. We can use this transformed point with the distance field as described in 3.4.1 to check for the collision.

4 Results and comparisons

All the measurements in this section are done on an Intel Core i5-2410M CPU @ 2.30GHz x 4 with 4 GB RAM.

4.1 Performance of the distance field calculation method

The distance field is calculated for the upper jaw and the lower jaw at different resolutions and the time taken is measured. The time taken to calculate the distance field depends on the resolution of the field and number of vertices and faces the geometry has. The following table shows the the statistics for the geometry for the upper and the lower jaw.

	Number of vertices	Number of faces
Upper jaw	126449	251650
Lower jaw	163032	323998

The following table shows the time taken to calculate the distance field at different resolutions for the upper and the lower jaw.

Resolution	Time taken for upper jaw $\backslash s$	Time taken for lower jaw $\backslash s$
5x5x5	7.5	9.7
10x10x10	53.3	71.7
15x15x15	180	236
20x20x20	430	534.05
25x25x25	840	920
30x30x30	1440.92	1550.73
35x35x35	2316.09	2455.21

The data is visualized and the graph is as follows.

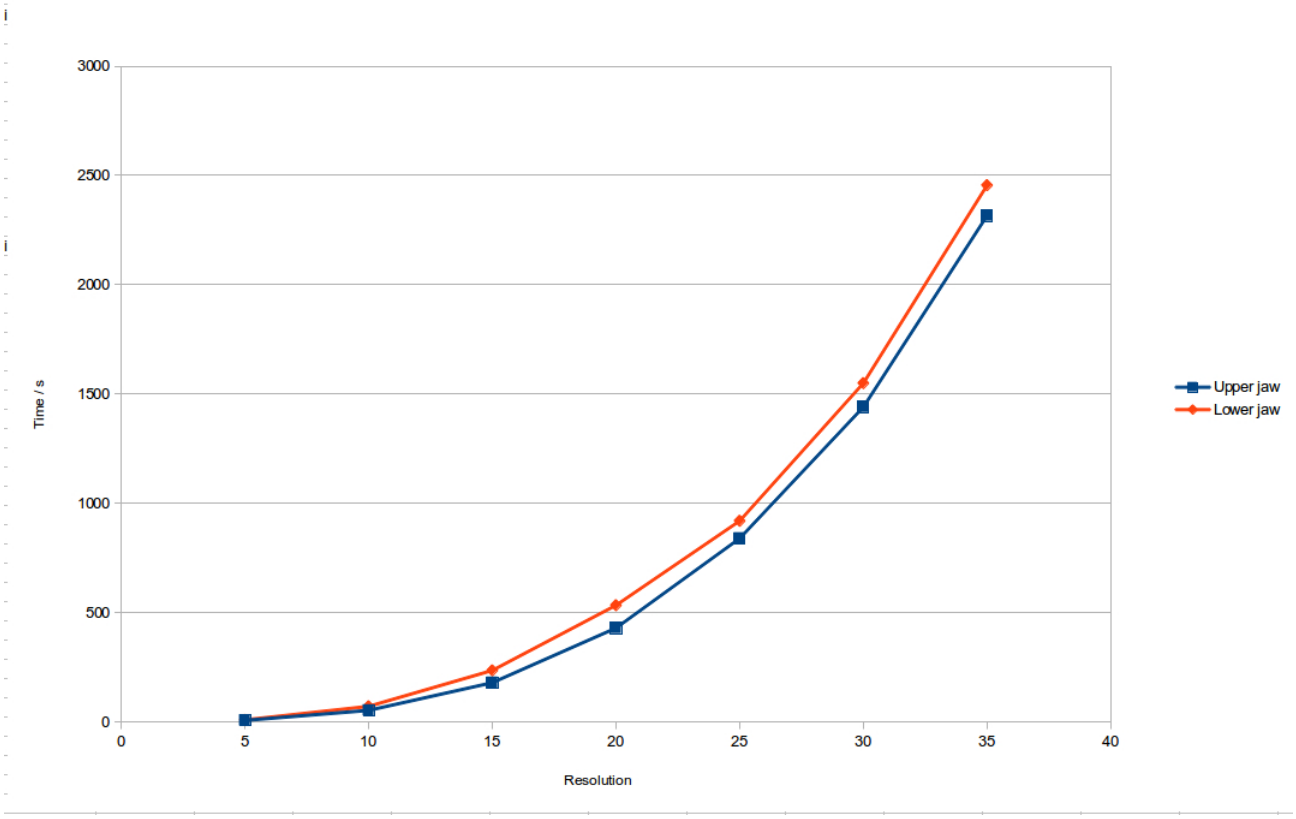

Figure 12: Comparison of the distance field calculation times for the upper and the lower jaw.

The accuracy of the collision detection increases with increase in the resolution of the distance field. It can be inferred from the data that there is a cubic relation between the time taken to calculate the distance and the resolution. The calculation of the distance field is a pre processing step and it need to be calculated once and then stored in the memory so a higher resolution distance field is desired even though with increase in the time taken.

4.2 A comparison between the distance field and some other methods

To make analysis of the proposed collision detection mechanism, it is important to test it against some of the other methods that are widely used for collision detection. The chewing simulation of our teeth model already has two methods implemented for the collision detection purpose. These are the simple ray triangle intersection tests and the collision detection using the kd tree. We will compare our distance field method with both of these methods in the following sections.

4.2.1 Distance field versus simple ray to triangle intersection test method

The ray to triangle intersection method was tested in a simple run of the simulation. At each iteration of the simulation the number of intersections and the total time taken for the intersections was recorded. The average time per intersection was then computed for each iteration. The results are shown in the table below.

Iteration	Number of intersections	Total time taken / s	Average time per intersection / s
1	2	2.61	1.305
2	3	2.67	0.755667
3	9	2.219	0.246556
4	16	2.5	0.15625
5	20	2.594	0.1297
6	23	2.537	0.110304
7	28	2.612	0.0932857
8	33	2.33	0.0706061
9	37	2.597	0.0701892
10	52	2.408	0.0463077
11	56	2.423	0.0432679
12	63	2.391	0.0379524
13	63	2.359	0.0374444
14	61	0.375	0.0031454
15	60	0.375	0.00625
16	64	0.39	0.00609375

The data was then plotted with number of iterations on the x-axis and the average time taken per collision on the y-axis.

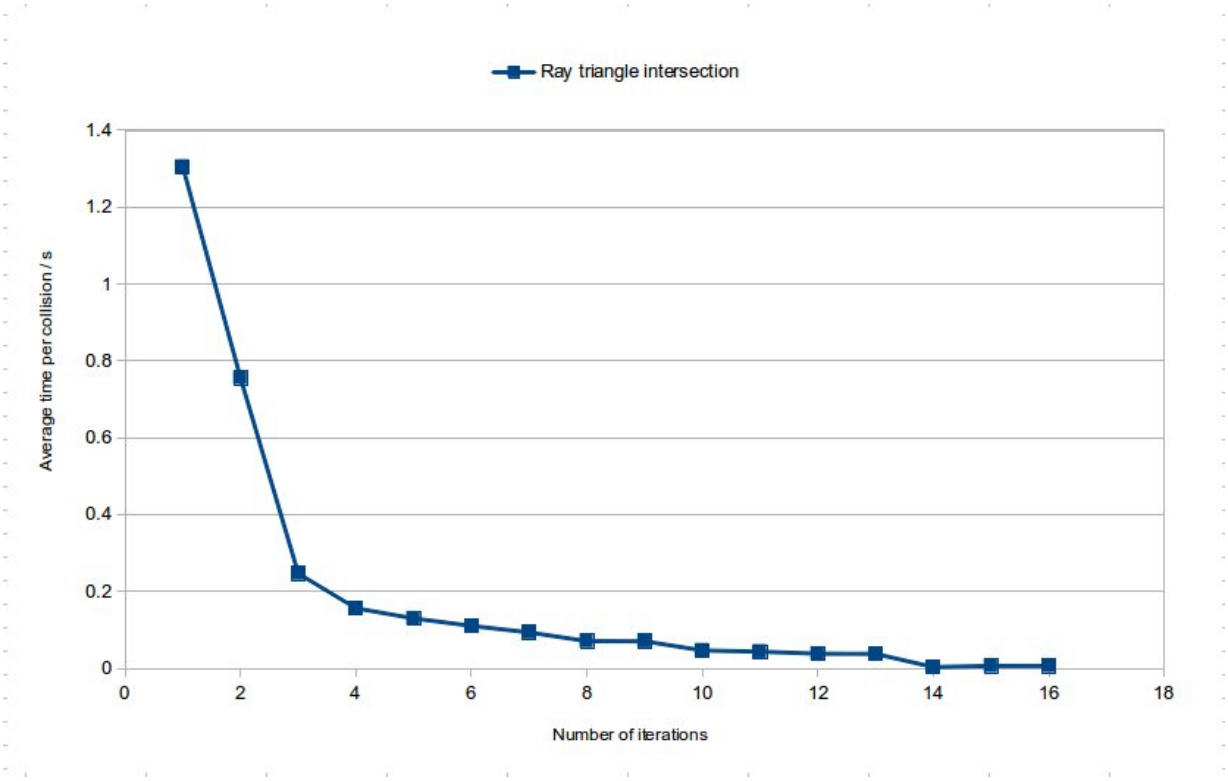

Figure 13: Ray triangle intersection approach

As visible from the plot, the ray triangle intersection approach is very expensive in the start when the number of intersections are low but when the number of intersections start to increase the average time per intersection decreases significantly and becomes stable.

The distance field method was tested in the similar manner. The distance field was calculated with resolution 25^3. And the bounding volume heirarchy was calculated with depth 3 for the substrate. The results are shown in the table below.

Iteration	Number of intersections	Total time taken / s	Average time per intersection / s
1	2	0.00579	0.002895
2	7	0.015505	0.002215
3	11	0.021857	0.001987
4	18	0.039186	0.002177
5	21	0.049665	0.002365
6	25	0.049675	0.001987
7	36	0.06606	0.001835
8	40	0.12144	0.003036
9	43	0.093267	0.002169
10	45	0.109845	0.002441
11	48	0.15432	0.003215
12	51	0.110007	0.002157
13	51	0.113373	0.002223
14	53	0.1165311	0.0021987
15	55	0.10931	0.0019875
16	57	0.14478	0.002541

The data was then plotted with number of iterations on the x-axis and the average time taken per collision on the y-axis.

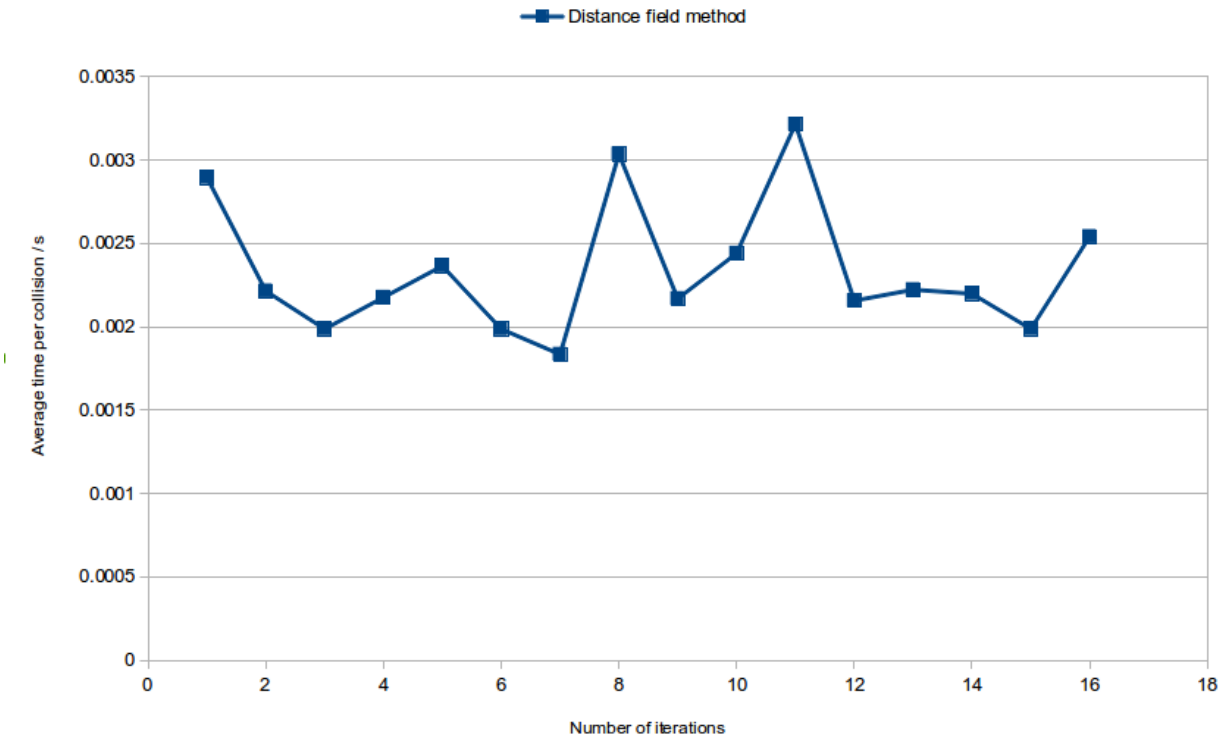

Figure 14: Distance field approach

The plot shows that the average time per intersection for the distance field method generally stays between $0.002s$ and $0.003s$. It was observed that whenever the substrate collided with the lower jaw, the average time per collision increased. This can be explained as according to our collision detection mechanism, for the lower jaw the bounding volume heirarchy has to be rotated before it can be checked against the distance field, these extra operations cause the average time per intersection to increase.

To compare the two methods, they were plotted on a single axis and the result is as follows.

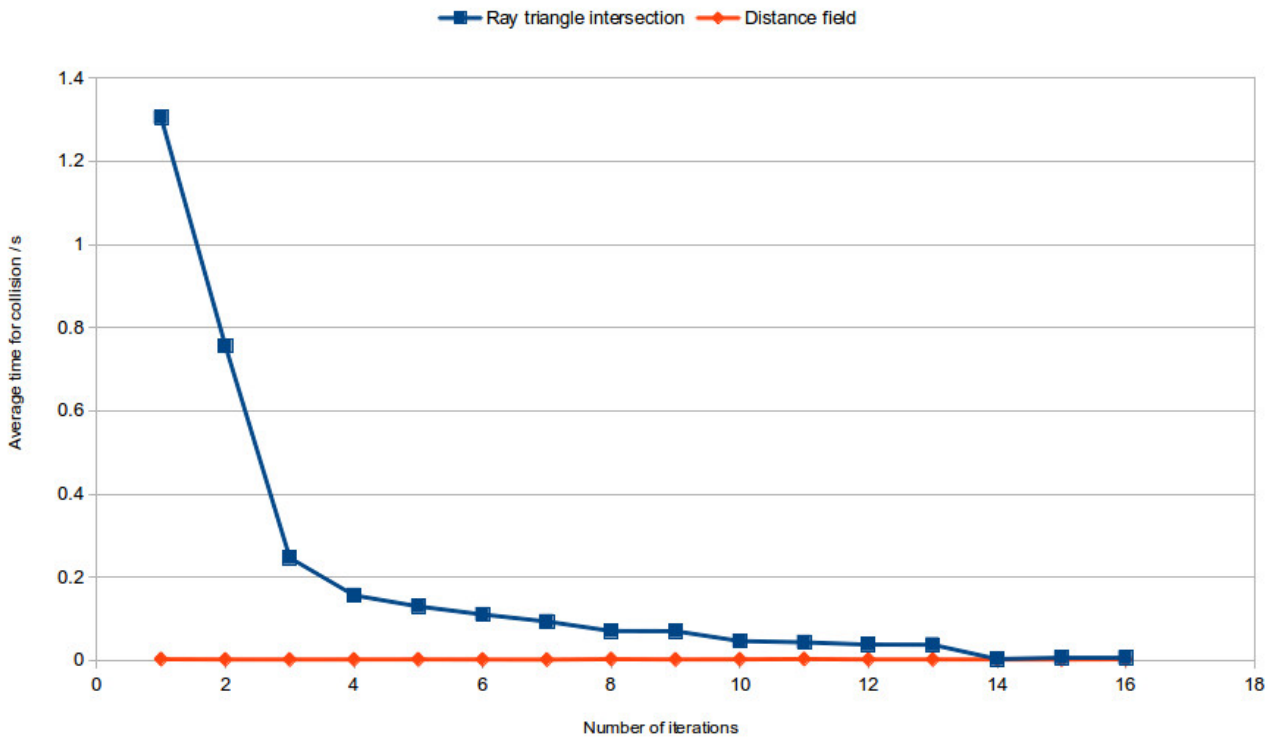

Figure 15: Comparison of both of the approaches

The first thing to note is that the distance field method is much more stable as compared to the ray triangle intersection method. The ray triangle intersection method moves towards stability as the number of intersections increase but no such trend was seen in the distance field method. The average time taken per intersection for the distance field method was generally lower than the ray intersection method.

4.2.2 Distance field versus collision detection using the kd-tree

The kd tree was used to check for collision detections in the chewing simulation scene. At each iteration of the simulation, the number of intersections were recorded and the total time taken to detect the intersections. The average time per collision was then computed. The results are shown in the table below.

Iteration	Number of intersections	Total time taken / s	Average time per intersection / s
1	2	0.016	0.008
2	3	0.016	0.00533333
3	9	0.047	0.00522222
4	16	0.093	0.0058125
5	20	0.125	0.00625
6	23	0.156	0.00678261
7	28	0.172	0.00614286
8	33	0.219	0.00663636
9	37	0.235	0.00635135
10	52	0.313	0.00601923
11	56	0.344	0.00614286
12	63	0.39	0.00619048
13	66	0.406	0.00615152
14	61	0.375	0.00614754
15	60	0.375	0.00625
16	64	0.39	0.00609375

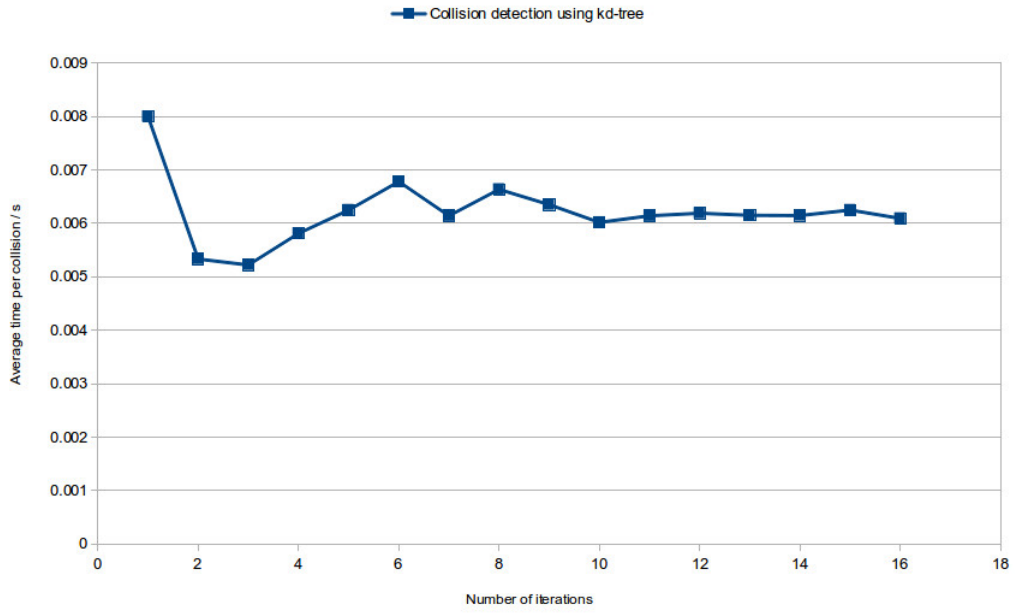

Figure 16: Collision detection using the kd-tree

The graph shows that the average time to detect intersection becomes stable quickly, the value settles around $0.006s$.

The data from kd tree method was compared with the distance field method from the previous section. The graph is as follows.

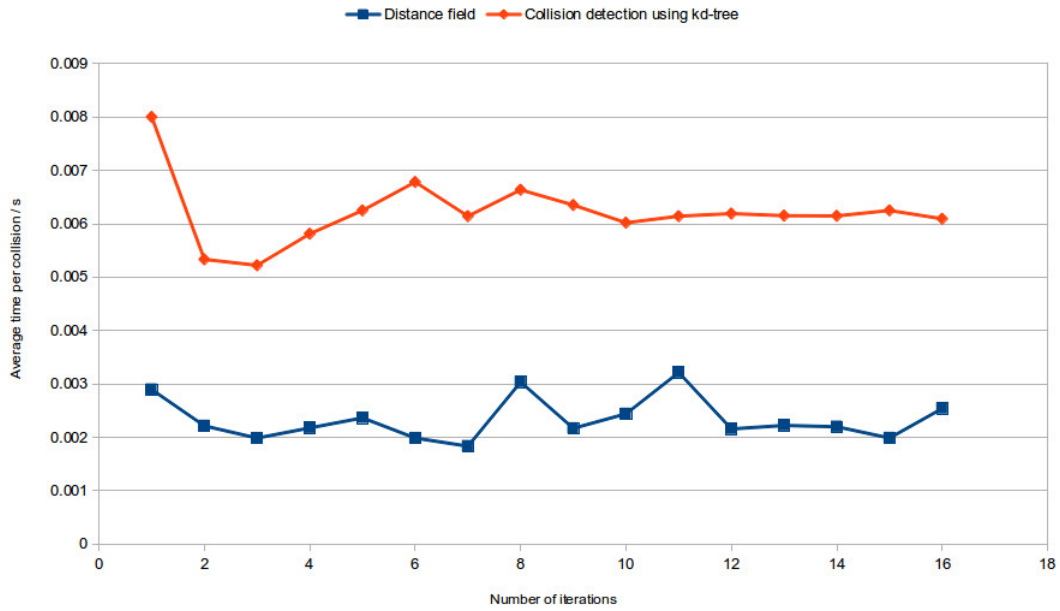

Figure 17: Kd tree approach versus the distance field approach

It can be noted from the comparison that the average time per intersection of the distance field method always remains lower than that of the kd tree method. The kd tree method shows some unstability in the start but as the number of intersections increase, the values become stable.

4.3 Performance of the collision detection depending on the depth of the bounding volume heirarchy

The performance of our collision detection algorithm is also dependant on the depth of the bounding volume hierarchy of the substrate. To investigate the result of changing the depth of the hierarchy on the performance of the collision detection, the simulation was done with different hierarchy depths and the average time taken to detect a collision was recorded. The results are shown in the table below.

Heirarchy depth	Average time taken per collision / s
0	0.00513
1	0.00411
2	0.00323
3	0.002547
4	0.002236
5	0.002672
6	0.003154

The data is plotted below with depth on the x axis and the average time per collision on the y axis.

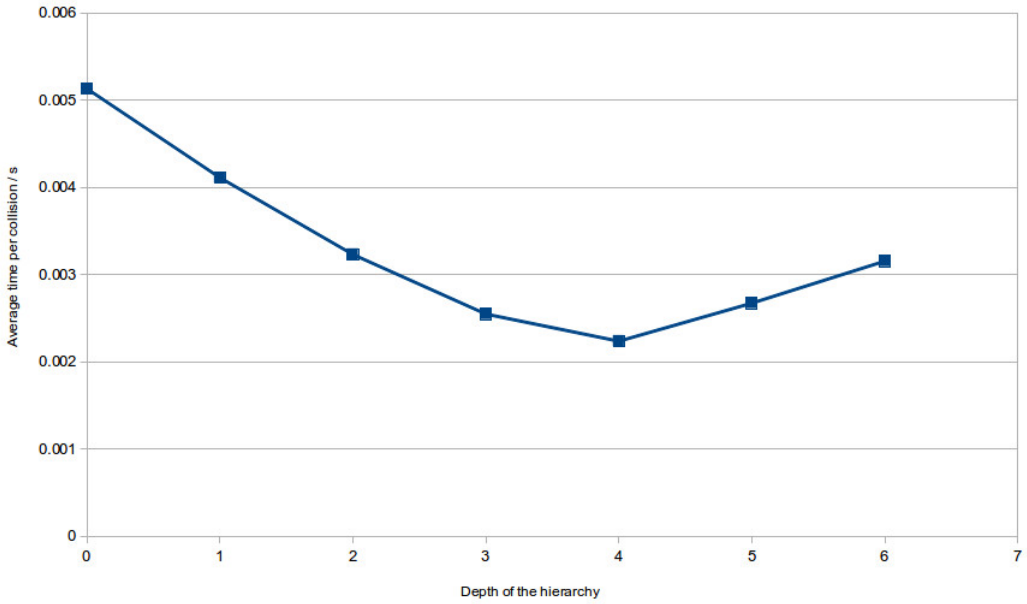

Figure 18: Average time per collision versus bounding volume hierarchy depth

As the depth of the hierarhy increases, the average time per collision decreases until depth 4. After this depth the average time per collision increases. This can be explained as the bounding volume hierarchy has to be rotated to check the collision for the lower jaw, increasing the depth makes it more expensive to rotate the hierarchy. This compensates the less time for checking the collision for the reduced geometry per node of the hierarchy and the average time per collision actually increases.

5 Conclusion

The purpose of this thesis was to explore an efficient collision detection mechanism which could allow the human teeth model chewing simulation in real time. The paper describes how distance fields can be used to provide fast collision detection. We built upon the fast calculation of distance fields method proposed by by Arnulph, Gerrit and Clemens [1] and saw how it can be used to calculate the distance fields for our human teeth model. We also explored the method for fast calculation of minimum distance from a point to a triangle [2].

The bounding volume hierarchies were discussed and we saw how the hierarchies can be constructed and used to discard large amount of complex geometry during the collision detection.

In the end we compared the distance field approach for collision detection with the simple ray - triangle intersections method without any tree and with the collision detection using the kd - tree. We saw that the distance field method on average takes less time to detect a collision than the other two methods. We also observed that using the distance field method, the average collision detection time with the lower jaw is higher than that of the

upper jaw due to the extra operations of rotating the bounding volume hierarchy into the local coordinates of the lower jaw.

Although, the distance field gives us a better average collision detection time, the memory consumption of this method is also of interest. We have to store the distance field in a 3D array and also store the bounding volume hierarchy for the substrate. Thus, the memory consumption of this method is higher than the ray triangle intersection and the kd tree method. Memory is the trade off we have to make for the speed of the collision detection in the distance field method.

We also compared the performance of the distance field method at different depths of the bounding volume hierarchy of the gummy bear substrate. It was observed that as we increased the hierarchy depth from 0 to 4 the performance of the method improved but after this when we increased the depth, the performance actually started to decrease. This behaviour was because of the extra operations for rotating the bounding volume hierarchy to test the collision with the lower jaw.

6 Appendix

6.1 Point to triangle distance pseudo code

```
1: procedure CalculateDistance(Point P, Point V1, Point V2, Point V3)
2:     E0 = V2 − V1
3:     E1 = V3 − V1
4:     D = V1 − P
5:
6:     a = E0.E0
7:     b = E0.E1
8:     c = E1.E1
9:     d = E0.D
10:    e = E1.D
11:    f = D.D
12:
13:    det = a ∗ c − b ∗ b
14:    s = b ∗ e − c ∗ d
15:    t = b ∗ d − a ∗ e
16:
17:    if (s + t ≤ det) then
18:        if (s < 0) then
19:            if (t < 0) then
20:                // Region 4
21:            else
22:                // Region 3
23:        else if (t < 0) then
24:            // Region 5
25:        else
26:            // Region 0
27:    else
28:        if (s < 0) then
29:            // Region 2
30:        else if (t < 0) then
31:            // Region 6
32:        else
33:            // Region 1
```

The code for Region 0:

```
1: invDet = 1/det
2: s∗ = invDet
3: t∗ = invDet
```

The code for Region 1:

```
1: numer = c + e − b − d
```

2: **if** $(numer \leq 0.0)$ **then**
3: $\quad s = 0.0$
4: **else**
5: $\quad denom = a - 2 * b + c$
6: $\quad s = (numer \geq denom ? 1.0 : numer/denom)$
7: $t = 1 - s$

The code for Region 3 ; Region 5 is similar:

1: $s = 0$
2: $t = (e \geq 0 ? 0 : (-e \geq c ? 1 : -e/c))$

The code for Region 2 ; Region 4 and 6 are similar:

1: tmp0 = B+D
2: tmp1 = C+E
3: **if** $(tmp1 > tmp0)$ **then**
4: $\quad$ numer = tmp1-tmp0
5: $\quad$ denom = A-2*B+C
6: $\quad$ s = (numer $\geq$ denom ? 1 : numer / denom)
7: $\quad$ t = 1-s
8: **else**
9: $\quad$ s = 0
10: $\quad$ t = (tmp1 $\leq$ 0 ? 1 : (E $\geq$ 0 ? 0 : -E/C));

The final block of code is as follows:

1: $sq\_distance = as^2 + 2bst + ct^2 + 2ds + 2et + f$
2: **return** $squareRoot(sq\_distance)$

References

[1] Arnulph Fuhrmann, Gerrit Sobottka, Clemens Gro. *Distance Fields for Rapid Collision Detection in Physically Based Modeling* , University of Bonn. 2003.

[2] David Eberly. *Distance Between Point and Triangle in 3D*. Geometric Tools, LLC. 1999.

[3] Laurent Poirrier.*An efficient space partitioning technique based on linear kd-trees for collision culling*. 2012.

[4] Hamzah Asyrani Sulaiman, Abdullah Bade. *The Construction of Balanced Bounding-Volume Hierarchies using Spatial Object Median Splitting Method for Collision Detection*.2011.

[5] Athanasios Vogiannou, Michael G. Strintzis. *Distance maps for collision detection of deformable models*. Informatics and Telematics Institute, CERTH.

[6] Meyer, M. Debunne, G. Desburn ,and Barr , A. H. Interactive animation of cloth-like objects for virtual reality. The Journal of Visualization and Computer Animation 12 (May 2001), 112.

[7] Aans, H., and J. A. Brentzen. 2003. *Pseudo-normals for Signed Distance Computation*. In Proceedings of Vision, Modeling, and Visualization 2003.

[8] Stefan Gottschalk. *Collision queries using oriented bounding boxes*. Chapel Hill. 2000.

[9] Vam Den Bergen , G. Efficient collision detection of complex deformable models using aabb trees. Journal of Graphics Tools 2, 4(1997), 114.

[10] Herman J. Haverkort. *Introduction to bounding volume hierarchies*. 2004.

[11] Pablo Bauszat, Martin Eisemann and Marcus Magnor. *The Minimal Bounding Volume Hierarchy*. TU Braunschweig. 2010.

[12] Thomas Larsson, Tomas Akenine-Moller. *Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models*. 2013.