University of Scranton

Haptics Technology as a Physical Therapy Treatment Technique

A thesis submitted in partial

Fulfillment of the requirements for the degree of

MASTERS OF SCIENCE

In

SOFTWARE ENGINEERING

By

Bret Oplinger

Spring 2016

# Abstract

This project is being undertaken for Dr. Renee Hakim of the University of Scranton’s Physical Therapy department with Dr. Benjamin Bishop as my advisor. Dr. Hakim wanted someone to produce software to be used in a study to assess the benefits of haptics technology as a Physical Therapy tool in the treatment of a person recovering from carpal tunnel surgery. This software would task a user with practicing writing with the hope of regaining the desired functionality in their wrist and/or hand. There are two parts to this thesis: the development of the PTHaptics application and research into letter partitioning. The PTHaptics application maintains a three dimensional block with a word engraved in it. The user manipulates the stylus attached to the haptics device to interact with the application and traces the word inside the block. Presently, there is a grid that divides up the word to track user progress. This leads to inaccurate trace route through the word, so a replacement to the grid is desired. The letter partitioning research delves into finding and implementing an algorithm to divide a letter into polygons.

# Table of contents

[1 Abstract 2](#_Toc449559279)

[2 Table of contents 3](#_Toc449559280)

[3 Background 4](#_Toc449559281)

[3.1 Haptics 4](#_Toc449559282)

[3.2 Haptics device 4](#_Toc449559283)

[3.3 Physical therapy concepts 5](#_Toc449559284)

[3.4 OpenGL 5](#_Toc449559285)

[3.5 OpenHaptics toolkit 6](#_Toc449559286)

[3.6 OpenSCAD 6](#_Toc449559287)

[4 PTHaptics application 7](#_Toc449559288)

[4.1 Introduction 7](#_Toc449559289)

[4.2 Run through 7](#_Toc449559290)

[4.2.1 User selects one of three PTHaptics shortcuts 8](#_Toc449559291)

[4.2.2 User prompted for input 9](#_Toc449559292)

[4.2.3 Word blocks generated from user input 10](#_Toc449559293)

[4.2.4 User practices writing 11](#_Toc449559294)

[4.2.5 User completes the level 12](#_Toc449559295)

[4.3 Issues with the grid 13](#_Toc449559296)

[5 Letter partitioning 14](#_Toc449559297)

[5.1 Introduction 14](#_Toc449559298)

[5.2 Unsuccessful algorithms 15](#_Toc449559299)

[5.2.1 Art gallery problem 15](#_Toc449559300)

[5.2.1.1 What it is 15](#_Toc449559301)

[5.2.1.2 Why doesn’t it work for my project? 17](#_Toc449559302)

[5.2.2 Hertel-Mehlhorn’s partition algorithm 18](#_Toc449559303)

[5.2.2.1 What it is 18](#_Toc449559304)

[5.2.2.2 Why doesn’t it work for my project? 18](#_Toc449559305)

[5.3 Working algorithm 19](#_Toc449559306)

[5.3.1 Voronoi diagram 20](#_Toc449559307)

[5.3.1.1 Introduction 20](#_Toc449559308)

[5.3.1.2 Fortune’s algorithm 21](#_Toc449559309)

[5.3.2 Random sites 24](#_Toc449559310)

[5.3.3 Fitness function 25](#_Toc449559311)

[6 Conclusion 26](#_Toc449559312)

[7 Work cited 27](#_Toc449559313)

# Background

## Haptics

Computer haptics is defined as software associated with generating and rendering the touch and feel of virtual objects (i.e. computer graphics) [11].

## Haptics device

In the case of this project, the haptics device being used is a *PHANTOM Geomagic Touch*. As seen in Figure 1, it has a stylus attached to a robotic arm that is held inside a base. When a haptics application is active on a computer that the haptics device is connected to, the user would manipulate the haptics stylus and translate into something in the haptics application.

For instance, a movement of the stylus translates to a cursor moving in this 3D environment created by the application. When the cursor bumps into another object (i.e. a cube in space), the haptics device would give the user physical feedback for having the stylus’ cursor run into the cube.

**Figure 1:***PHANTOM Geomagic Touch* haptics device

## Physical therapy concepts

Because this project was produced at behest of the University’s Physical Therapy department, the concepts that supported the project requirements were brought forth by the domain expert, Dr. Hakim.

Robotic and virtual reality systems are used to train individuals with loss of motor functions. Based on principles of motor learning and neuroplasticity, task-oriented repetitive movements using these VR systems provide the experience, timing, motivation and attention necessary to improve motor function. Robotic devices can mimic some of the features of a therapist’s manual assistance, allowing patients to semi-autonomously practice their movement training. With respect to rehabilitation and recovery, Dr. Hakim described *neuroplasticity* as the ability of the nervous system to respond to intrinsic and extrinsic stimuli by reorganizing its structure, function and connections.

*Fitts’ law* predicts that the time required to rapidly move to a target area is a function of the ration between the distance to the target and the width of the target [6].

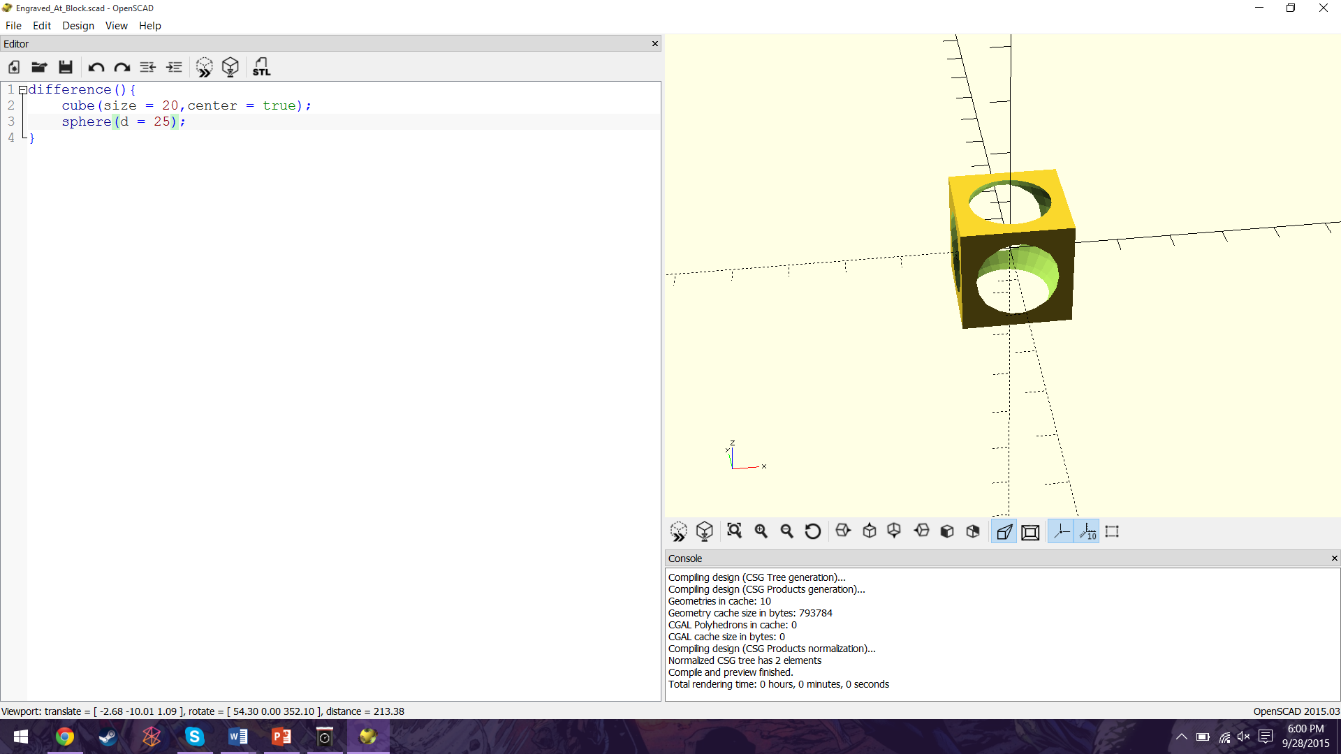
## OpenGL

*OpenGL* (Open Graphics Library) is a cross-language, multi-platform API for rendering 2D and 3D vector graphics. It is a very popular library to create graphics application because OpenGL is available on any computer with a graphics card and there are many wrapper libraries (such as GLUT) in a multitude of programming languages.

## OpenHaptics toolkit

The software library necessary to create applications with the PHANTOM Geomagic Touch device is the OpenHaptics Toolkit. The toolkit is provided by Geomagic for use only with Geomagic haptics device. From geomagic.com, the *OpenHaptics toolkit* enables “developers to add haptics and true 3D navigation to a broad range of applications including 3D design and modeling, medical, games, entertainment, visualization, and simulation” [3]. Because the toolkit is patterned after the OpenGL API, it allows for graphics programmers to easily integrate with OpenGL applications.

## OpenSCAD

OpenSCAD is a free software application for creating solid 3D CAD (or computer-aided design) objects. These objects are created by a 3D compiler based on a textual description language. Figure 2 provides an example of how one would generate a 3D model. In Figure 2, there is a simple script that renders the difference between a cube and a sphere.

**Figure 2:**OpenSCAD script example

# PTHaptics application

## Introduction

This application, as mentioned earlier, will be used in the physical therapy study administrated by Dr. Hakim to determine how well haptics technology might serve as a physical therapy technique. The user practices writing with the use of the haptics device and the PTHaptics application. The haptics device provides physical feedback as if one was writing with a pen on a table. The goal of a session is to trace out the word and not touch the sides of the word block in the lowest time possible (displayed in seconds). The user is provided physical feedback (force on the pen based on where the pen is running into the wall) and auditory feedback (a grating buzzer noise) when a side is touched in a similar vein as the Hasbro board game *Operation*.

## Run through

This section presents an example of a session of the PTHaptics application.

Quick run through:

1. User selects one of three PTHaptics shortcuts
2. User prompted for input
3. Word blocks generated from user input
4. User practices writing
5. User completes the level

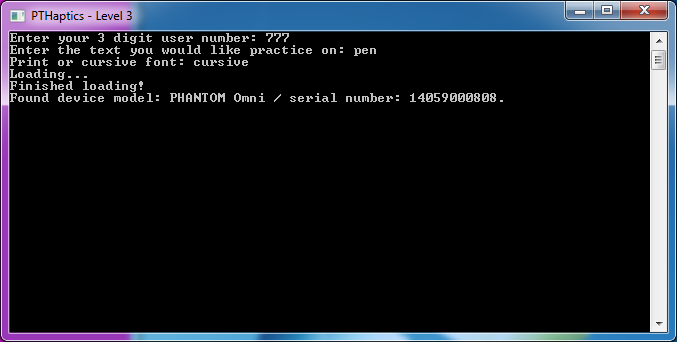
### User selects one of three PTHaptics shortcuts

On the computer where the haptics study is being performed, there are three desktop icons: “PTHaptics – Level 1”, “PTHaptics – Level 2”, and “PTHaptics – Level 3”. The user selects which level they would like to practice on by double clicking the appropriate icon. The levels differ on width of the word being practiced on. Each icon is a shortcut to the PTHaptics.exe. They differ in that a different run time variable is passed to PTHaptics. The run time variable is the wanted width of the word in the block. The first level has three times the width as the original word. The second level has two times the width as the original word. The third level has the width as the original word.

**Figure 3:**Shortcuts on desktop

### User prompted for input

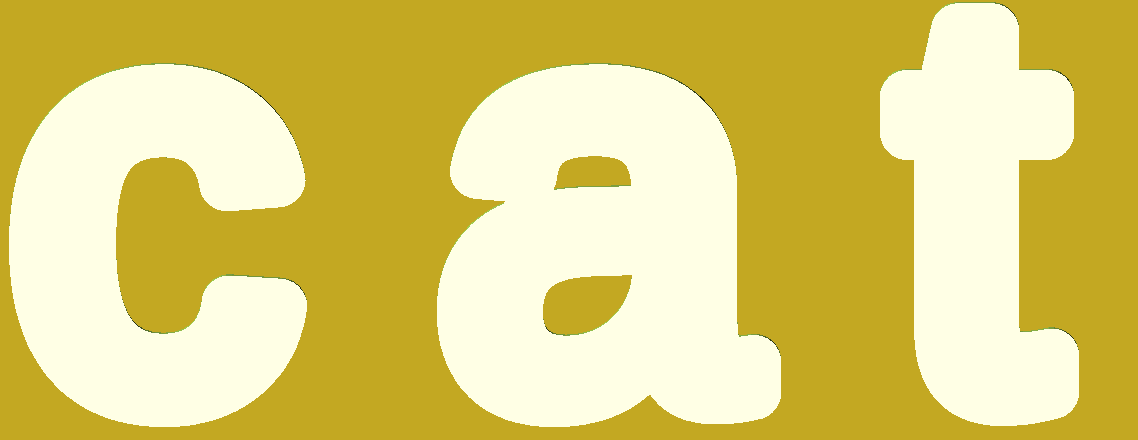
When the program is launched, the user is prompted to provide three different items: user number, word and font type. The user number consists of three digits and is the name of the file containing the user’s session information. The word is the string that the user will be practicing on. The user can then choose between cursive or print font types. The level is selected depending on which desktop shortcut the user selects, so only one level is loaded now.



**Figure 4:**Prompt with user input

### Word blocks generated from user input

Word text, font, and size are passed by a command line call to an OpenSCAD script. A solid block with a word-shaped hole cut out. There are three possible sizes for a word block. They correspond to the three levels the user can select on the desktop. The three sizes (standard size, twice times standard size and thrice times standard size) correspond to levels (3, 2, and 1). So, first level is the easiest with widest path and third level is the hardest level with the narrowest path.



Level 1 example

Level 2 example

Level 3 example

**Figure 5:**Word blocks for each level

### User practices writing

The user must then trace the word as best as they can without touching the side of the block (the purple part as seen in Figure 6). Their elapsed time and score is also displayed. The score corresponds to the number of the user’s collisions with the sides of the block. To indicate and keep track of a user’s progress, a seventeen by seven grid is superimposed on the block. Each time the user touches one of the squares in this grid with the haptics device, the currently touched grid square changes color and the total number of grid squares touched increases.



**Figure 6:**Beginning of a writing session

### User completes the level

When enough of the grid has been touched, the user completes the level. Along with this message and button, the session data (stylus-wall collisions, elapsed time, etc.) is saved in the file with their user number. The user is able to go through the three generated levels as many times as they want. To change user number, word, font type or level, the application needs to be restarted.

**Figure 7:**Completion of a session

## Issues with the grid

The grid presently used needs to be replaced by a better way to partition the word in the block. Currently, the author has to guess how many grid squares would be touched to consider a letter traced. In the current iteration of PTHaptics, 10 grid squares per letter need to be touched for the user to succeed. This does not depend on the size of the letter. With a three letter word, only 25 percent of the whole grid spanning the screen is used. The grid has to be this large to accommodate all possible letter sizes. Besides wasting performance drawing the useless 75% of the grid, there is another issue. Looking at Figure 8, it is possible for a user to succeed without filling in every grid square touching a letter. A third issue arises from constraints from the OpenHaptics toolkit. There is a limit to the number of objects in an OpenHaptics application. The 117 squares of the grid take up most of this allotment, so the squares have to be rather large. The goal of this research was to find a way to dynamically partition each letter of the word block. The next section “Letter Partitioning” speaks about the results of my research.

**Figure 8:**Issue with the grid

# Letter partitioning

## Introduction

There are two parts to this thesis: the development of the PTHaptics application and research into letter partitioning. The PTHaptics application maintains a three dimensional block with a word engraved in it. The user manipulates the stylus attached to the haptics device to interact with the application and traces the word inside the block. Presently, there is a grid that divides up the word to track user progress. This leads to inaccurate trace route through the word, so a replacement to the grid is desired. The goal of this research is to figure out a way to partition each letter of the word block into polygons. These partitions would need to be spaced evenly throughout each letter to ensure that a trace path would represent a natural path through the word. For instance, having a really small polygon in a corner of a letter that the user needs to go out of their way to touch would be less than ideal. In a similar vein, having too many polygon partitions in a letter would cause a user to trace a route through the letter that painstakingly includes these tiny polygons. Each of these polygonal partitions would then be the haptically interactive objects that would track the user’s progress in tracing the word.

During this research, two algorithms almost fit my purposes, but both had some properties that are undesirable. Luckily, there was an algorithm that met my needs.

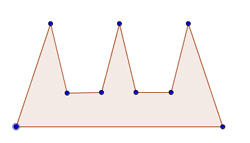
## Unsuccessful algorithms

### Art gallery problem

#### What it is

This problem was proposed in 1973 by Victor Klee. It asked for the minimum number of guards sufficient to see every point of the interior of an n-vertex simple polygon. In this art gallery, each guard would be placed on a vertex that makes up this art gallery/simple polygon. A simple polygon is a simply-connected closed region whose boundary consists of a finite set of line segments. Two points in a polygon are said to be mutually visible if the line segment joining these two points lies inside the polygon.

Václav Chvátal in 1975 showed that n/3 guards are always sufficient for any n-vertex simple polygon. He demonstrated the validity of that statement with a mathematical proof. Steve Fisk in 1978 provided a simple proof using graph coloring which can be seen in Figures 9[4].

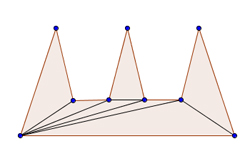


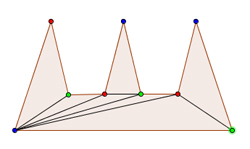
Step 1: Find a simple polygon

Here number of vertices equals 9.

Step 2: Triangulate the polygon.

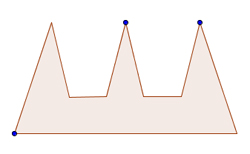
In other words, connect the vertices of your polygon by straight lines in such a way that the polygon is divided up into triangles.





Step 3: Color the vertices

Color the vertices of each triangle three different colors.



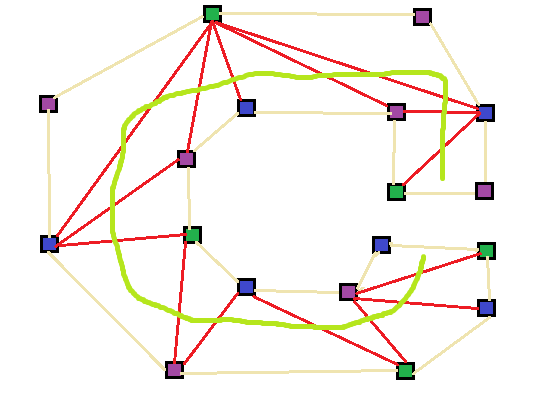
**Figure 9:**Art gallery problem steps

Step 4: Guard selection.

Pick a color (one of the three you colored your vertices with) and the vertices of that color are now the art gallery’s guards.

#### Why doesn’t it work for my project?

If the art gallery problem was used to implement the letter partitioning, it would produce too many subdivisions. These polygons would not match the trace path. In Figure 10, the green line represents the trace path.



**Figure 10:**Paper art gallery implementation

### Hertel-Mehlhorn’s partition algorithm

#### What it is

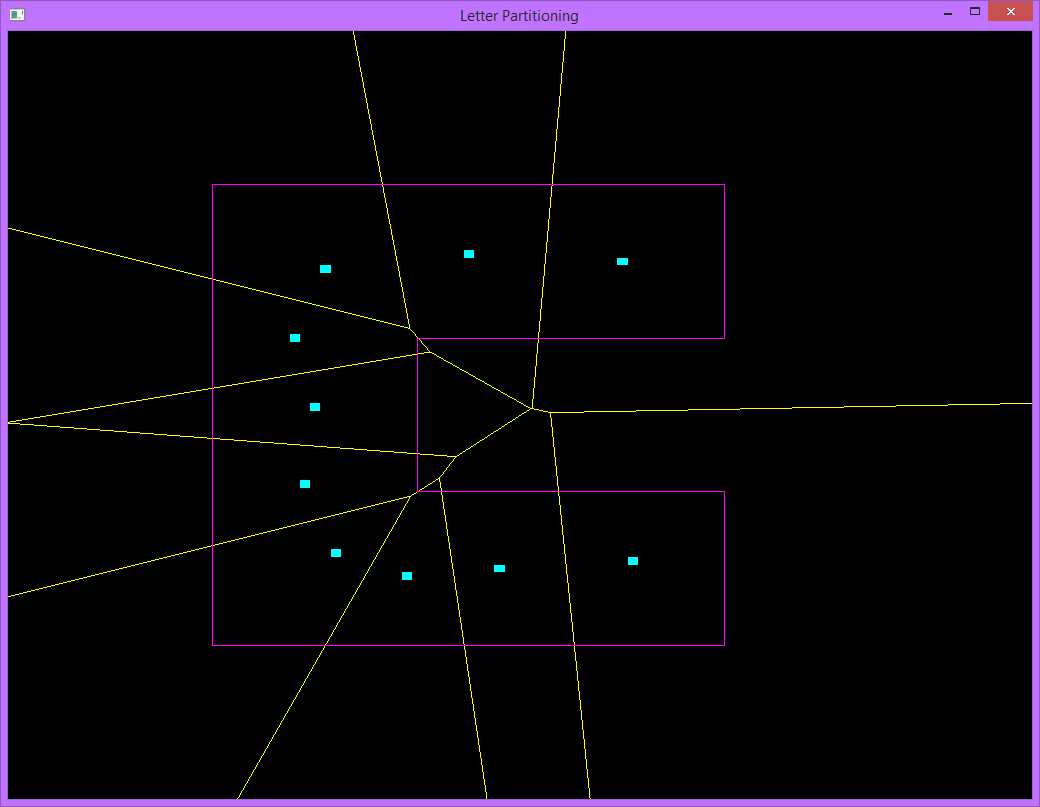
Hertel-Mehlhorn’s algorithm partitions a polygon with diagonals. When considering a convex partition of a polygon by diagonals, a diagonal is essential for a vertex if the removal of it results in the vertex no longer being convex. If a diagonal is not essential for either endpoint, it is considered inessential. Hertel-Mehlhorn’s algorithm can be simplified to: start with a triangulation of the polygon, remove an inessential diagonal, and then repeat [9].

#### Why doesn’t it work for my project?

On the surface level, this algorithm sounds like it fits my needs perfectly. It works on complicated polygons, even ones with holes. Sadly, the algorithm has an input of the polygon with vertices in order (either counter-clockwise or clockwise). At present, it is unsure how one would extract from a 3D model file just the vertices for a singular letter in the word block.

## Working algorithm

I make use of Voronoi diagrams to partition letters. I place random points inside the letter (in the case of Figure 11, the lower case letter c). These random points are selected based on a fitness function where points far from walls of the letter and other points are desirable. The Voronoi diagram is generated based on these points. Figure 11 demonstrates the fruits of this research.



**Figure 11:**Voronoi diagram partitioning a letter

### Voronoi diagram

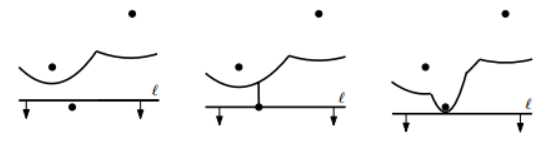
#### Introduction

Named after Georgy Voronoi, a Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. These points are called seeds, sites or generators and are specified beforehand. Voronoi cells are regions that for each site there is a region consisting of all points closer to that site than to any other.

Voronoi edges and vertices always need to meet certain requirements. Inner Voronoi edges need to have two Voronoi vertices. Outer Voronoi edges usually have just one Voronoi vertex and go off into infinity. For the purpose of my project and so it can be drawn using OpenGL, the Voronoi edges are bounded in a box. A point *q* lies on a Voronoi edge between two sites if and only if the largest empty circle centered at *q* touches only those two sites. So, a Voronoi edge is made up of points equidistant from two sites. A point *q* is a Voronoi vertex if and only if the largest empty circle centered at *q* touches at least three sites [1].

Voronoi diagrams have a number of applications. For instance, one of the earliest uses of a Voronoi diagram was by John Snow to study the 1854 Broad Street cholera outbreak in SoHo, England. He showed the correlation between residential areas on a map of Central London of residents who were using a specific water pump and the areas with the most deaths due to the outbreak (so each water pump is a site and the resulting Voronoi cell was the area of the city that made use of that pump).

#### Fortune’s algorithm

A popular and fast (timing-wise) algorithm to construct a Voronoi diagram is called Fortune’s algorithm. It was originally published by Steven Fortune in 1986 in his paper “A sweepline algorithm for Voronoi diagrams.” The algorithm keeps track of both a sweep line and a beach line. The sweep line is a horizontal line that sweeps the set of sites from top to bottom. The beach line is not a line, but composed of pieces of parabolas. For each site above the sweep line, you can define a parabola equidistant from that site and the sweep line. A new arc/parabola is added to the beach line when the sweep line encounters a site. As the sweep line progresses downward, when two parabolas cross, this traces out the edges of the Voronoi diagram. The place where two parabolic arcs meet are called breakpoints and they lie on a Voronoi diagram edge. The arcs flatten out as a sweep line moves down. Voronoi vertices are identified when two breakpoints meet/fuse [1].

**Figure 12:**Sweepline moving down and adding to the beachline [1]

This algorithm makes use of three different data structures: one to keep track of the current state of the Voronoi diagram, one to store the current state of the beach line and one that has the current state of the sweep line. The state of the Voronoi diagram is stored in a doubly connected edge list or DCEL. A DCEL is a set of faces, half-edges and vertices. Each half edge has a reference to one face on one side of the edge and one vertex on one end of the edge. Also, each half edge has a next and previous pointers which point towards the next or previous half edge on the face [3]. A binary search tree is used to keep track of break points and arcs currently on the beach line. The internal nodes represent break points between two arcs and the leaf nodes represent arcs where each arc is in turn represented by the site that generated it. A priority queue is the data structure that traces the sweep line’s downward progress. Events are stored in the priority queue. There are two types of events: site and circle events. Site events are when the sweep line encounters a new site. Circle events are when the sweep line encounters the bottom of an empty circle touching three or more sites (here is where what was mentioned about how to encounter Voronoi vertices). The events are prioritized by y-coordinate.

Here’s the pseudo-code [1]:

Algorithm VORONOIDIAGRAM(P)

Input. A set P := {p1,..., pn} of point sites in the plane.

Output. The Voronoi diagram Vor(P) given inside a bounding box in a doubly connected edge list D.

1. Initialize the event queue Q with all site events, initialize an empty status

structure T and an empty doubly-connected edge list D.

2. while Q is not empty

3. do Remove the event with largest y-coordinate from Q.

4. if the event is a site event, occurring at site pi

5. then HANDLESITEEVENT(pi)

6. else HANDLECIRCLEEVENT(γ), where γ is the leaf of T representing the arc that will disappear

HANDLESITEEVENT(pi)

1. If T is empty, insert pi into it (so that T consists of a single leaf storing pi)and return. Otherwise, continue with steps 2– 5.

2. Search in T for the arc α vertically above pi. If the leaf representing α has a pointer to a circle event in Q, then this circle event is a false alarm and it must be deleted from Q.

3. Replace the leaf of T that represents α with a subtree having three leaves. The middle leaf stores the new site pi and the other two leaves store the site pj that was originally stored with α. Store the tuples pj, pi and pi, pj representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on T if necessary.

4. Create new half-edge records in the Voronoi diagram structure for the edge separating V(pi) and V(pj), which will be traced out by the two new breakpoints.

5. Check the triple of consecutive arcs where the new arc for pi is the left arc to see if the breakpoints converge. If so, insert the circle event into Q and add pointers between the node in T and the node in Q. Do the same for the triple where the new arc is the right arc.

HANDLECIRCLEEVENT(γ)

1. Delete the leaf γ that represents the disappearing arc α from T. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on T if necessary. Delete all circle events involving α from Q; these can be found using the pointers from the predecessor and the successor of γ in T. (The circle event where α is the middle arc is currently being handled, and has already been deleted from Q.)

2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list D storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoint of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end at the vertex.

3. Check the new triple of consecutive arcs that has the former left neighbor of α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into Q. and set pointers between the new circle event in Q and the corresponding leaf of T. Do the same for the triple where the former right neighbor is the middle arc.

### Random sites

Because one needs sites to construct a Voronoi diagram and it could be any letter that needs to be partitioned, so random sites are used. The goal is to just have sites inside the letter, so the resulting Voronoi diagram would be divided into parts. To do this, a random point is selected that at least exists in the bounding box around the letter and check if it is inside the letter. If it is, the point is added to the list holding all the generated sites. If it does not, another point is tried. This process is iterated a number of times equal to the number of wanted sites.

Here’s the code used to check if a point is in a letter [2]:

bool isPointInLetter(VPoint\* p) {

int numVerts = polygon.size();

bool pointIsInside = false;

for (int i = 0, j = 1; i < numVerts - 1; i = j++) {

if (((polygon[i]->y > p->y) != (polygon[j]->y > p->y))) {

if (p->x < (polygon[j]->x - polygon[i]->x) \* (p->y - polygon[i]->y) /(polygon[j]->y - polygon[i]->y) + polygon[i]->x) {

pointIsInside = !pointIsInside;

}

}

}

return pointIsInside;

}

### Fitness function

With randomly generated sites in some letter, a Voronoi diagram is constructed that does partition the letter. These randomly spaced sites in the letter can cause partitions to be less than ideal (like too many divisions closer together). It is ideal when a site is far as possible from the walls of the letter and from other sites. To guarantee that practice writing sessions on a particular letter are as smooth as possible, partitions should be spaced evenly through a letter, so sites being in the middle of a letter should be emphasized more heavily than maximizing distance to points. To get generated points that meet the above criteria, a fitness function is used.

A fitness function is a type of objective function that is used to evaluate how close a solution is to achieving some goal. When a site is generated, a fitness value is given that is the minimum distance to the nearest wall times a coefficient of three plus the minimum distance to the nearest point. The minimum distance to the nearest wall for an optimal point would be larger (i.e. far away from the wall) than a not optimal point. The minimum distance to the nearest site would also be larger for an optimal point. The fitness equation is (Coefficientwall \* distancenearest wall) + distance nearest point where Coefficientwall is equal to three.

Ten sites is desired for this Voronoi Diagram. From trying different number of sites, more than ten sites results in too many partitions which results in less than ideal trace routs. So for each of those, a thousand sample points are generated and calculate the fitness value for each. A thousand sample points are used because 10000 points impacts performance and one hundred points would not provide a large enough pool. The point with the greatest fitness value is returned.

# Conclusion

The current state of the PTHaptics application and the research into letter partitioning has been discussed in previous sections. The next step of my project (to be taken up by the person Dr. Bishop selects) would be to bring the letter partitioning into the PTHaptics application. To accomplish this, several items would need to be completed. The vertices for each possible letter a user can enter would need to be hard coded, so when the user enters a word, a word vertices data structure can be populated. The level the user selects would require different operations applied to the word vertices data structure. The next implementation would have to manually apply the Minkowski sum to the word to increase the letter width depending on which level the user selects. The easiest way to think of Minkowski sum is that a circle of some size (30, 20 and 10 for levels 1, 2 and 3 respectively) is dragged around the edge of each letter which dramatically increases the size. After this operation is applied, a Voronoi diagram is generated for each letter to subdivide them and the word vertices data structure would then be sent to OpenSCAD, so it can be rendered to the screen. The Voronoi diagram cells might need to be placed into 3D model files to allow for easy communications with the OpenHaptics library.

# Work cited

1. De Berg, Mark. "Computational Geometry." *Computational Geometry* (2008): 1-17. 2008. Web. 14 Apr. 2016.
2. Franklin, W. Randolph. "Point Inclusion in Polygon Test." PNPOLY. Rensselaer Polytechnic Institute, 2003. Web. 20 Apr. 2016.
3. Fischer, Leonardo. "The DCEL Data Structure: A C++ Implementation." Leonardofischer.com, 5 Nov. 2011. Web. 14 Apr. 2016.
4. Kaul, Hemanshu. "The Art Gallery Problem." *Bulletin of the Associates in Fine Arts at Yale University* 10.2 (1941): 1-3. Web. 14 Apr. 2016.
5. Lien, Jyh-Ming, and Nancy M. Amato. "Approximate Convex Decomposition of Polygons ∗." (n.d.): n. pag. Texas A&M University. Web. 14 Apr. 2016.
6. MacKenzie, Scott. "Fitts' Law as a Research and Design Tool in Human-Computer Interaction." Human-Computer Interaction (1992): 91-139. Web. 20 Apr. 2016.
7. Miu, Allen. "Voronoi Diagrams." *SpringerReference* (n.d.): n. pag. *Voronoi Diagrams*. MIT, 30
8. "OpenHaptics Toolkit Programmer's Guide." SensAble Technology, Inc (n.d.): n. pag. 2008. Web. 20 Apr. 2016.Sept. 2003. Web. 14 Apr. 2016.
9. O'Rourke, Joseph. "Computational Geometry in C." *Google Books*. Cambridge University Press, 1998. Web. 14 Apr. 2016.
10. Sacrist´an, Vera. "TRIANGULATING POLYGONS." (n.d.): n. pag. Universitat Polit`ecnica De Catalunya. Web. 14 Apr. 2016.
11. Srinivasan, Mandayam A. "What Is Haptics." Laboratory for Human and Machine Haptics: The Touch Lab (n.d.): n. pag. Massachusetts Institute of Technology. Web. 20 Apr. 2016.