Optimization and Analysis with Centroidal Voronoi Tessellations

Chad Brewbaker; Drake University
Daniel Wengerhoff; University of Iowa

August 10, 2001

Abstract

This paper discusses algorithms for computing and several applications of the Centroidal Voronoi Tessellation (CVT). Topics include: DNA Analysis, Stock Market Analysis, Image Compression, Sound Compression, and Higher Dimensional CVTs.

Contents

1 Introduction

Centroidal Voronoi Tessellations can be used in a broad range of applications. They can be easily visualized in two dimensions:

For example, suppose that a town wants to determine the optimal placement of mailboxes for its residents. Suppose also, that each resident will only use the mailbox closest to him/her. Then, corresponding to every mailbox, is a region consisting of the customers who use that mail box. The region that each mailbox draws from is a *Voronoi cell* and the mailbox is the *generator* of that cell. All of the cells together form a *Voronoi Tessellation.*

If the city planners want to determine these regions they have two choices. First, it is possible to explicitly (using an algorithm) calculate their regions. It is also possible to estimate these regions and their properties by polling a large number of people in the town to find out which mailbox they are closest to.

With an arbitrary placement of mailboxes some residents will have to walk a long way to get to the nearest mailbox. Now a group of residents who have this problem petition the town council to move the mailboxes to better locations so that the average distance everyone has to walk is minimized.

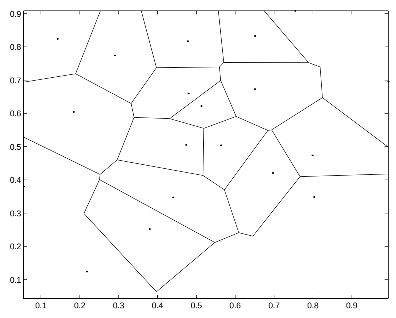

Figure 1: A Voronoi Tessellation with arbitrary cell generators. Notice that many of the Voronoi Regions are irregularly shaped

The council decides to find the average position of of the houses that use each mailbox (cell generator). If all mailboxes are then relocated those positions then the average distance people have to walk will decrease.

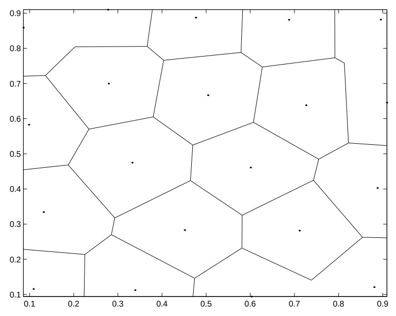

Figure 2: A Voronoi Tessellation after improving the cell generators 30 times.

If the process of repositioning mailboxes to the average location of their users is repeated many times, the mailboxes become both *centroids* and cell generators for their Voronoi cells. This mailbox layout is then a *Centroidal Voronoi Tessellation (CVT)*. Notice in figure 3, that the new generators, the dots, have moved to better (more central) locations. Also, notice that the shape of each Voronoi region is tending toward being a regular hexagon.

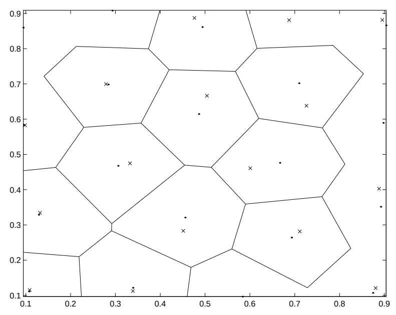

Figure 3: A Centroidal Voronoi Tessellation after improving the cell generators 100 times. The X's are where the cell generators were after 30 improvements and the dots are the cell generators after 100 improvements.

However, it is important not to limit the CVT to lower dimensions. This paper will discuss how CVTs can be used on data that is more relevant to problems at hand, and look at the CVT in higher dimensions.

2 Definitions

Voronoi Generator,Cell, and Tessellation:
Given a set of special points called *generators* and set of data, group the data into cells such that each data point is with the closest generator. These cells are called *Voronoi regions*, and all the cells together form a *Voronoi Tessellation.*

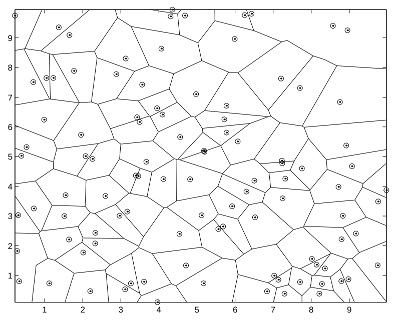

Figure 4: Voronoi Tessellation

Centroidal Voronoi Tessellation (CVT):

Centroidal means that for each Voronoi cell, the cell generator is also the centroid or average of all the data points in that cell.

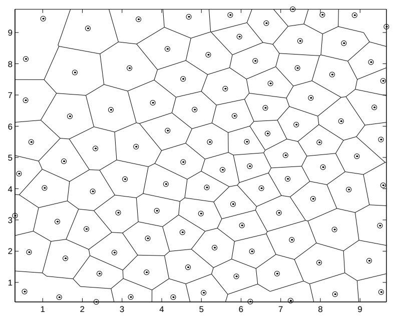

Figure 5: Centroidal Voronoi Tessellation

Delaunay Graph
Given the set of cell generators the *Delaunay graph* connects generators that are "neighbors" in the sense that their Voronoi cells share a common boundary.

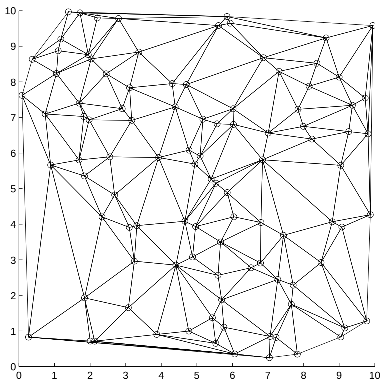

Figure 6: Delaunay Graph

3 Algorithm Outlines

CVT Algorithm
Given the region to make the CVT on (i.e. a box, sphere,...), choose some random points in the region to be the initial cell generators. Then, sample a number of random points from the region and find which cell generator each is closest to. For each set of points and corresponding cell generator find their average position in space. These averages are now the new set of cell generators. Repeat this process until a good enough (note: this is subjective and must be determined by the user) CVT is produced.

In pseudocode:

```
Pick K points in the region as initial cell generators
Loop until the CVT is good enough
    Loop for the number of points to be sampled
        Pick a random point in the region
        Find which cell generator it is closest to
        Add it to a "bucket" of points closest to that cell generator
    End Loop
    Find the average point for each "bucket"
    Assign the cell generators to be these new points
End Loop
```

Discrete CVT Algorithm
With the discrete CVT algorithm, only a finite number of data points are allowed to be used in the calculation. This means that each generator must come from the data set. Therefore, it is necessary to do the extra calculation of finding the closest point in the data set to the average for each "bucket".

```
Pick K points in the data set as initial cell generators
Loop until the Discrete CVT is good enough
    Loop for the number of points you want to sample
        Pick a random point in your data set
        Find which cell generator it is closest to
        Add it to "bucket" of points closest to that cell generator
    End Loop
    Find the average point for each "bucket"
    Find which point in the "bucket" is closest to this average
    Assign the cell generators to be these new points
End Loop
```

This algorithm is also known by other names such as Lloyd's Method and K-means. Also, it is important to note that sometimes "buckets" become empty. When this happens put a random point in the data set into the bucket.

Nearest Neighbor Search
This routine is used in the middle of the CVT algorithms to find which Voronoi generator a sample point is closest to. Querying every Voronoi generator would be inefficient. Instead, compute the Delaunay graph for your Voronoi generators. On average, this takes $n \log n + \lceil \frac{dimension}{2} \rceil$ [7] time. Then search the Delaunay graph each time you want to do a query. The query on the Delaunay graph takes around $n^{\frac{1}{dim}}$. For these routines we used QHull [5].

4 DNA Data Clustering

The first application we used to test the CVT algorithm was DNA data clustering. Daniel Ashlock and Nichole Leahy supplied some gene data from the plant Arabidopsis. Arabidopsis is in the mustard family, and has only 5 chromosomes. According to *http://www.arabidopsis.org* the genome was sequenced as of 2000, and now biologists are finishing up the work on mutations that exist, and what proteins each gene creates.

The data consisted of 180 plants going through seven experiments, where each experiment was done with two kinds of dye. This resulted in a data set of 180 fourteen dimensional points. These points were also euclidean normalized $(-1, 1)$ before computation.

On this data we ran four versions of the CVT algorithms:

Discrete CVT, euclidean distance
Discrete CVT, dissimilarity measure
Continuous CVT, euclidean distance
Continuous CVT, dissimilarity measure

In the discrete version only points in the data set were allowed to be Voronoi generators, but with the continuous version any number could be used.

The Euclidean distance measure for two vectors {A,B} was:

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2 + ...}$$

The dissimilarity distance measure was taking the dot product between the vectors to find the "angle of dissimilarity":

$$|A \cdot B|$$

The continuous ran faster than the discrete version because there wasn't the extra step of assigning Voronoi cell averages to points in the data set. However, this made the algorithm use Voronoi generators that could have nothing to do with the problem space.

"Similar" clusters are defined as the number of elements on the diagonal of a matrix formed by the comparison of two cluster sets and row/column sorted by a greedy search.

Cluster Comparisons		
algorithm	algorithm	similar elements out of 180
Discrete/Dissimilarity	Continuous/Euclidean	86
Discrete/Dissimilarity	Discrete/Euclidean	90
Discrete/Dissimilarity	Continuous/Dissimilarity	93
Continuous/Euclidean	Continuous/Dissimilarity	101
Continuous/Euclidean	Discrete/Euclidean	107
Discrete/Euclidean	Continuous/Dissimilarity	119

It is interesting to note that the Discrete/Euclidean was best matched with the Continuous/Dissimilarity results even though they were using different search spaces and distance metrics.

Also, we compared our results with another clustering program that Ashlock wrote:

Cluster Comparisons		
algorithm	algorithm	similar elements out of 180
Continuous/Dissimilarity	Ashlock's	71
Discrete/Euclidean	Ashlock's	79
Continuous/Euclidean	Ashlock's	94
Discrete/Dissimilarity	Ashlock's	95

This technique could be used by biologists to match gene expression data from multiple experiments to get a better idea of what conditions cause certain genes to be expressed. Each cluster could be studied to see what the data points have in common. Also, it is an area of open research as to what the best distance metric and search space is for various problems.

5 Stock Market Analysis

Stock market analysis was another application the CVT algorithm was tested for. Arnold Cowan and Rick Dark from the Iowa State Finance Department who were kind enough to supply a large sample of stocks from their database. The data consisted of 256 stocks over 250 trading days from January 2000 to December 2000, resulting in 256 points with 250 dimensions. The data was euclidean normalized $(-1, 1)$, but it was not adjusted for dividends and splits.

Here are four of the clusters formed using 20 Voronoi generators:

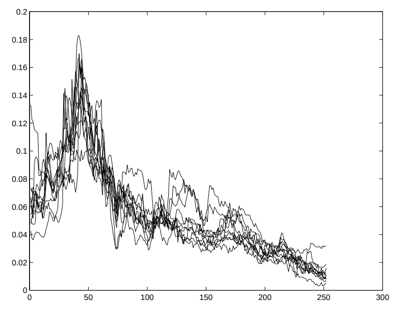

Figure 7: *AKLM, ECTL, DAVX, FIBR, NQLI, MSEL, GDC, SOF, VOXX, POCI, DWCH* This cluster seems to be tech stocks.

With 20 generators the stocks in similar industries seemed to cluster together. Also, it is interesting note is that one of the clusters was empty because the algorithm had not went through enough iterations on that particular run.

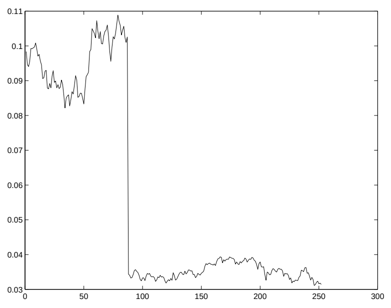

Figure 8: *GE* This cluster was an oddball because General Electric(GE) split three for one on May 8, 2000.

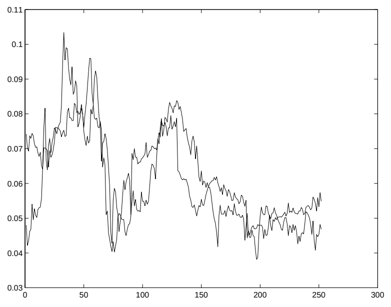

Figure 9: *REFR, MNTR* This is a cluster of high tech manufactures.

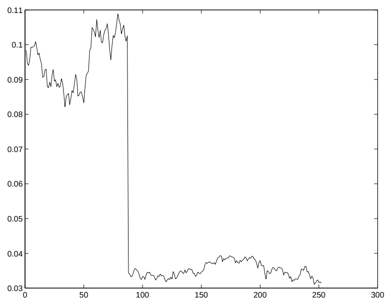

Figure 10: *ORCL, SCOR, PWEI, TRMB* This cluster had companies from different industries.

One possible use for this technique might be portfolio diversification. If the CVT algorithm is run on a large number of stocks and an investor notices that all of his stocks end up in the same cluster his portfolio might not be as diversified as originally thought.

Another use would be event analysis. The CVT algorithm could be used to cluster stock prices in the days following some event such as OPEC cutting oil production. This would help investors get an idea of which stocks were effected by this event.

Cowan mentioned that the CVT algorithm could also be used to study the difference in price fluctuations in high priced and low priced stocks. Do penny stocks have different price fluctuations than stocks with a larger price per share?

Dark mentioned that the CVT algorithm could be run on historical post Initial Public Offering(IPO) prices to create clusters of stocks that thrived and stocks that went belly up. What would the patterns be over three month, six month, one year, and five year time periods?

6 Image Compression

To begin with there are two kinds of compression: lossless and lossy. Lossless compression is the creation of a smaller file from an original one, so that all information in the original can be retrieved if desired. By contrast, in lossy compression a smaller file is created which may contain the essential information, but which has permanently lost some of the original information.

The CVT can be used for lossy image compression. For images lossy compression is okay because images contain lots of unimportant information. The images below are good examples of this. Notice how it is difficult to distinguish between the picture with 256 shades of gray and the picture with 64 shades.

Many digital image formats contain pixels that have a color component, and a coordinate of where they are located within a picture. Image compression can be achieved by reducing the the number of distinct color components stored, thus taking less bits to store the color component information of each pixel.

Grayscale images have one color component per pixel, thus making for a quick initial test. The first image tested was a simple .pgm file called FEEP because when viewed it is the word FEEP on a white background. Even though 16 shades of gray were possible, only five were actually used. After running it through the CVT program to compress it to two colors, the word became EEP. This occurred because the compression is lossy and the letter 'F' had a color very close to white. As a result, during the CVT program its color was assigned to be white while the other letters were assigned closer to black. After this success, a more complex image balloons.pgm[6] was tested. This picture originally had 256 shades of gray and was much larger than FEEP. The results of the compression are shown below. Notice that the picture using 16 shades of gray still has most of the features of the original picture. The main difference is that the compressed image has larger blocks of the same color. It has been shown in the paper by Gunzburger and Du[3] that a process called dithering can fix this problem, thus allowing for even more compression.

Next color images were tested. The color images we were encoded with a Red-Blue-Green (RGB) scheme where different intensities of the colors red, blue and green are stored for each pixel. This is great for CRT computer monitors (put a droplet of water on the screen of one to see why), but as far as color space it doesn't have a very good geometric representation. Therefore, the coordinates were converted to a YUV representation. In YUV the brightness/intensity of the image is stored as Y and then U and V are used to store light wavelengths. On a side note, looking at Y alone will produce a grayscale image. Performing the CVT on the YUV values and then converting back to RGB worked very well. Unfortunately the results can't be shown on paper, but they will be available on the web page.[9]

Here are the results on a grayscale image[6]. Notice how you loose color gradients as less and less Voronoi generators are used.

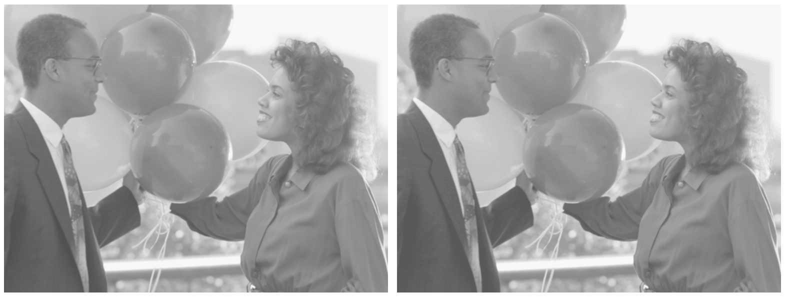

Figure 11: Original-256 shades, Compressed-64 shades

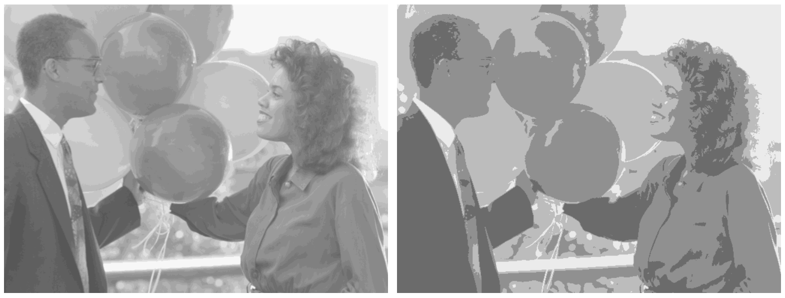

Figure 12: Compressed-16 shades, Compressed-4 shades

7 Audio Compression

After having success with the image compression, the next step was to try the CVT algorithm on audio compression The test sounds used were *.wav* files. These sound files contain audio samples over a time series. For testing two clips: *Woman.wav* [1]and *Thermo.wav*[2] were used. *Woman.wav* was a 9 second music clip that had 138 distinct amplitude values, and *Thermo.wav* was 3 second voice clip that had 247 distinct amplitude values. This meant that 2^8 values or 8 bits of data had to be stored for each sample. We ran each time series through the CVT to find generators that could closely approximate the sounds, thus reducing the number of bits that needed to be stored for each time sample.

There were some problems with picking the initial generators as random points in the sound files. The initial generators tended to be very similar. Doing a random distribution over the amplitude space seemed to work better. Also, it was important to ensure that zero amplitudes stayed zero so there wouldn't be constant noise.

Results were promising. The voice clip *Thermo.wav* could be recognized with 4 values, was clear with 16 values, and easy on the ears with 32 values. The music clip, Woman.wav, required a little more. Sixteen values was tolerable, and 64 was okay.

To hear the clips visit the webpage.[9]

Future uses of this technique might include doing a CVT of the frequency space for a sound file, and then taking each frequency band and doing another CVT on the amplitudes. Then, a lookup table could be stored at the start of a file describing the different generators in terms of their frequency and amplitude.

To really go whole hog you could cut down on the number of Voronoi generators in the amplitude dimension for waves that have a frequency not very audible to the human ear.

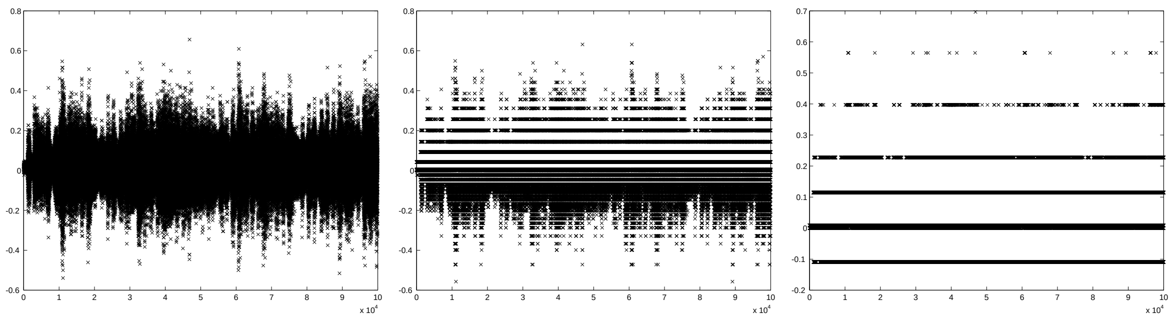

Figure 13: Original Clip, 64 sample values, 8 sample values

8 Higher Dimensional CVTs

After performing the CVT algorithm on 1 and 2 dimensions for some time, it is only natural to become curious about what a regular CVT might look like in higher dimensions. For this problem we performed the CVT algorithm on both a cube and a sphere to see if any recognizable shapes could be found.

For both runs we used 1000 generator points, and queried 1000 random sample points per generator for each iteration.

The cube did not work very well because the corners had too much influence on the Voronoi generators. The generators seemed to form cubes inside of cubes with noise in the middle.

However, the sphere seemed to give more interesting results. The cells seemed form shapes that looked like twelve sided regular polyhedra made up of pentagons (dodedcahedrons).

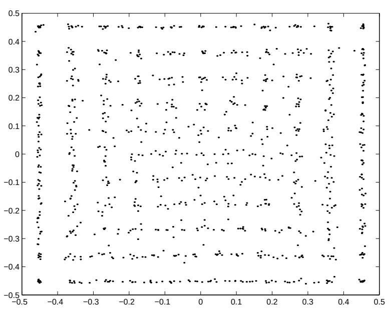
Figure 14: Looking down at the generators in the cube

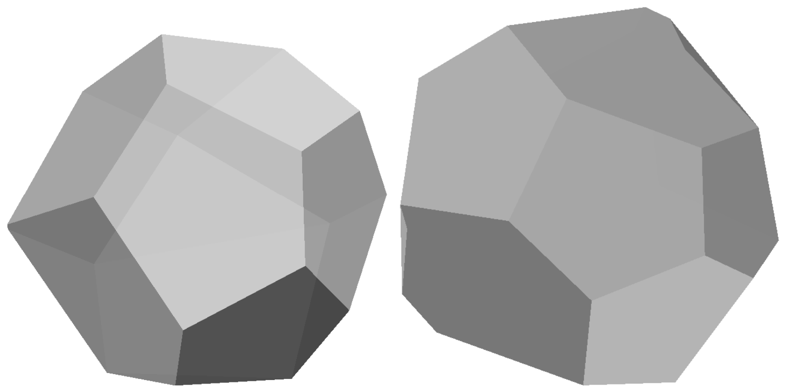
Figure 15: Centroidal Voronoi cells from the sphere

It would seem from these pictures that the regular CVT for 3 dimensions might be dodecahedron tessellations.

Then we began to wonder how many sides a regular CVT has for N dimensions. Our first guess was that it had to do with dimension combinations:

Conjecture 2		
Dimension	Sides	example
1	2	x^+, x^-
2	6	$x^+, x^-, y^+, y^-, xy^+, xy^-$
3	12	$x^+, x^-, y^+, y^-, z^+, z^-, xy^+, xy^-, xz^+, xz^-, yz^+, yz^-$
n	$2n + 2\binom{n}{2}$	$\ldots$

Curious, we checked out the sequence $\{2, 6, 12\}$ in the Online Encyclopedia of Integer Sequences[8]. We found that the sequence $2n + 2\binom{n}{2}$ is called the pronic (or heteromecic) numbers.

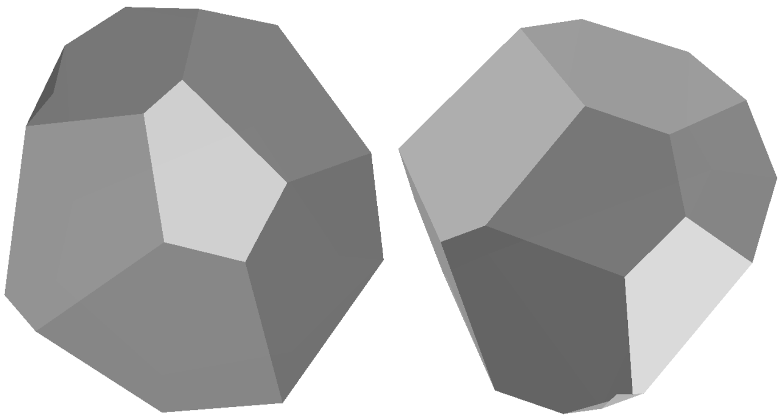
Figure 16: More Centroidal Voronoi cells from the sphere

Also, we came across the more probable answer: Kissing Numbers. A kissing number is the maximum number of hyperspheres that can touch a hypersphere in N dimensions. In one dimension a point can touch two points, in two dimensions a circle can touch six circles, and in three dimensions a sphere can touch 12 spheres.

The shape of a typical Voronoi cell in a CVT is a matter of some interest. Experimental results suggest that in two dimensions it is a hexagon and in three dimensions, perhaps a dodecahedron. However, we know of no proof, so this remains an open question.

9 Further Research Ideas

Other possible uses for this algorithm include:

- Modular structures: One might use the 3D CVT to design modular cargo cells such as fuel tanks for oil carriers and study their properties.

- Soap bubbles: If someone ran a 3D CVT of some clear container in a ray tracer like POVray he might observe some interesting soap bubble models.

- Decision Support Systems: The CVT of a data set could be used as guidance for a decision support system. If a distance metric can be defined so that groups of situations have a centroid, a nearest neighbor search with a new situation could be performed. Placing the new situation in the correct cluster would then provide the steps needed to deal with the new situation.

- Sports: Compute a CVT of where balls are hit or kicked to find a more optimal placement of athletes on the field.

- Database Design: Arrange data into a hierarchy of Centroidal Voronoi Tessellations to see what happens with query time.

There are also a few questions that can be looked in to:

- How does one compute the CVT on surfaces in 3D?
- Is there a feasible algorithm that always finds the centroids with the lowest energy?
- How do probabilistic methods compare with exact calculations in each iteration of the CVT algorithm?

10 Acknowledgements

We would like to thank Dan Ashlock, John Burkardt, Arnold Cowan, Rick Dark, Jason Grinblat, Max Gunzburger, Andrea Hamilton, Joe Koopmeiners, and Nichole Leahy for providing us data and support.

References

[1] Randy Bachman, Burton Cummings, Jim Kale, Garry Peterson, (The Guess Who), *American Woman*, January 1970.

[2] *The Simpsons*, Episode 135 *King-Size Homer*, first aired November 5, 1995.

[3] Max Gunzburger, Qiang Du, V. Faber *Centroidal Voronoi tessellations: applications and algorithms*; SIAM Review 41, 1999, 637-676; with Q. Du and V. Faber.

[4] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, *Spatial Tessellation: Concepts and Applications of Voronoi Diagrams*, 1992, John Wiley and Sons, ISBN 0-471-93430-5

[5] C. Bradford Barbeer, David P. Dobkin, Hannu Huhdanpaa, *The Quickhull Algorithm for Convex Hulls*, ACM Transactions on Mathematical Software, Volume 22, Number 4 (December 1996), pp. 469-483

[6] Howard Burdick, *Digital Imaging: Theory and Applications*, McGraw Hill, 1997

[7] Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 1998, Springer, 3-540-65620-0

[8] N. J. A. Sloane, *Online Encyclopedia of Integer Sequences*, http://www.research.att.com/ njas/sequences/

[9] ISU Mathematics REU 2001 website, http://www.math.iastate.edu/reu/2001/index.html