Megan Yociss

An Exploration of Constrained Centroidal Voronoi Tessellations for Fast Biological Modelling

Background and Motivation

The author was inspired by a recent development in the field of biology to study methods for computing constrained centroidal Voronoi tessellations (CCVTs) on arbitrary surfaces. Gomez-Galvez et al. proposed that the distribution of and geometric relationships between cells in some types of epithelial tissue can be modelled using CCVTs of the two surfaces of a sheet formed by these cells (1). They discovered that an important property of such tissues is demonstrated by the relationship between the CCVTs on both surfaces. A CCVT is first computed for the basal surface which the biological cells, represented computationally by Voronoi cells, are embedded in. Then the centroid of each Voronoi cell is projected to an apical surface, i.e. the surface at the top of the cells, which may be represented as having some arbitrary distance from the basal surface in its normal direction. A Voronoi diagram is computed for this surface using the projected centroids as Voronoi sites. Gomez-Galvez et al. found that depending on the distance assigned from the basal to the apical surfaces, some percentage of simulated cells contained different neighbors at the different surfaces. They were able to confirm that this geometric model represents some natural tissues accurately.

This model was demonstrated in the context of tissues with homogeneous cell distribution. Sanchez-Gutierrez et al. previously demonstrated that computational modelling of tissues using a Voronoi framework is generalizable to tissues with different cell distributions (2). They state that "as long as packed cells present a balance of forces within a tissue, they will be under a physical constraint that limits its organization", and use Voronoi diagrams to predict the count and area on a surface of different types of polygons. Their work challenged the widely held assumption that probabilistic cell proliferation mechanisms are required to explain the cellular organization of tissues. However, their computation of Voronoi diagrams assumes a flat plane as the surface embedding biological cells, while Gomez-Galvez et al. extend the method to find Voronoi diagrams on tubular and spheroidal surfaces.

Since Gomez-Galvez et al. state that there is a relationship between the reorganization of cells between the apical and basal surfaces, and the large-scale reshaping of epithelial tissue, the author inferred the utility of a generalizable framework for computing Voronoi diagrams on surfaces. Additionally, according to common biological knowledge, it seems that some tissues would require a catastrophic number of Voronoi sites to be adequately represented. Input from biologists would obviously be required in order to create a mathematically realistic, informative and useful model of a biological tissue, but the exploration of fast Voronoi diagram and CCVT computation on arbitrary surfaces is the motivation behind the rest of this research.

GPU-based computation of CCVTs

The jump flood algorithm for discrete Voronoi diagram computation is an attractive solution to the potential problems inherent in a large number of Voronoi sites, as its time complexity does not depend on the number of sites. Additionally, it is known to be O(log(*n*)) where the diagram is computed on a grid of *n* by *n* cells. (In reality the grid need not be square, and *n* will be the maximum value of its two dimensions.) The jump flood algorithm uses the following simple mechanism to assign each grid cell to a Voronoi site:

procedure 1:

- initialize a texture where each grid cell corresponding to a Voronoi site has as its own grid position x as its red channel, grid position y as its green channel, and some value that is unique per site as its blue channel. All other cells in the grid store some values indicated that they are not associated with a Voronoi site.

- at every step of size *step_size*, a grid cell performs checks on its eight neighbors at distance *step_size* in each cardinal and diagonal directions. For each neighbor, it obtains the values stored in their red and green channels. If a grid cell is not associated with a Voronoi site but its neighbor is, it takes on its neighbor's red, green and blue values. If a grid cell is associated with a Voronoi site but its neighbor is not, it does not update its color values. It determines its distance from the location (x, y) stored in its own red and green channels, and compares that distance to its distance to the location stored by its neighbor. If its distance to the location stored by its own label is greater than its distance to the location stored by its neighbor, it writes the red, green and blue values stored by its neighbor to the output texture buffer.

Rong et al. present extensions to the jump flood algorithm to compute a CCVT on a rectangular texture. They demonstrate that Lloyd's algorithm for CCVT computation can be implemented on the GPU. In a discrete domain, integrating a Voronoi region to find the centroid reduces to determining the average x and y coordinates of grid cells in that region. Additionally, a grid cell may be assigned a density, giving it more or less weight in the centroid computation. The author infers from the writing of Sanchez-Gutierrez et al. that different density distributions may be utilized to simulate different geometric formations of cells in a biological tissue.

Rong et al. state that the centroid computation can be done on the GPU by using a vertex shader program to translate all vertices with the same Voronoi site label to the same pixel position (3). Using the brief description provided by Rong et al. as guidance, the author worked out the following procedure that takes a texture containing a Voronoi diagram output by procedure 1 and outputs a new texture that can be used as input to procedure 1 in the next iteration:

procedure 2:

- initialize a set of points where each point has a location corresponding to a grid cell in the Voronoi texture.
- Initialize a clip space having its default pixel color equal to the “unlabeled” value used in procedure 1
- use the Voronoi texture as input to a vertex shader that handles each point in the previously initialized set. The vertex shader will determine a location in clip space for a point based on the Voronoi site id (stored in the blue channel of the input texture) such that all points having the same site id will be translated to the same pixel. The vertex shader also assigns a color to each point where the red and green channels contain that point’s x and y locations on the grid, and the blue channel contain the value 1.0.
- set the flag glBlendFunc such that additive blending of colors will be performed, meaning that if multiple vertices occupy the same pixel, the color computed for the pixel will be the sum of their individual colors. This means that a pixel containing all vertices for a particular Voronoi site will have the sum total of their x and y location values in its red and green channels, and the total number of grid cells associated with this Voronoi site in its blue channel
- render the resulting computed clip space to a texture
- in another fragment shader, which uses this texture as input, each grid cell checks every specific grid cell in the input texture known to contain Voronoi site information. If the red and green values of the specifically identified information cell, divided by the blue value, are equal to the location of this grid cell, this grid cell becomes a Voronoi site and stores its own location and a unique site id. Otherwise, it is given the unlabeled color value.

The author was able to implement Lloyd’s algorithm for CCVT computation on the GPU as presented by Rong et al. by utilizing the three.js library’s general-purpose GPU computation framework and writing shader programs implementing the procedures described. In order to more closely emulate the procedure followed by Gomez-Galvez et al., the author made some additional modifications in order to compute the CCVT for a cylindrical surface, rather than a two-dimensional plane: cylindrical distance was used to assign grid cells to Voronoi sites, and the location of a grid cell in the centroid computation was determined in relation to its Voronoi site rather than its absolute location on the grid. This code can be found at https://github.com/myociss/ccvt-gpu.

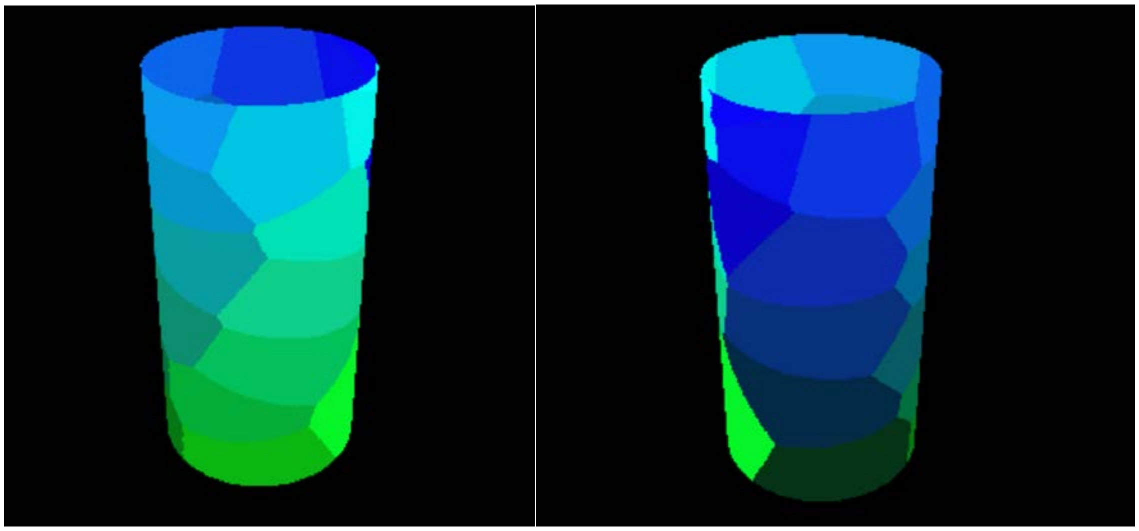

Figure 1: different views of a CCVT computed on a cylindrical surface

Generalization to arbitrary surfaces

A surface must be parameterized to a rectangular domain so that the GPU algorithms, which can only operate on rectangular textures, may be executed. While a cylindrical surface has an intuitive parameterization to a rectangular domain, most surfaces do not. Such a parameterization is called a geometry image. Gu et al. state that arbitrary surfaces cannot be mapped to a rectangular domain unless they have the same topology as a disk, and that a triangle mesh must be cut into a disk before it can be parameterized over a rectangular domain (4). Although algorithms exist to automatically generate a geometry image from an arbitrary triangle mesh, such as the one presented by Gu et al., it seems likely that most biological domains of interest could be represented by a triangle mesh of genus zero. The author found an interesting paper discussing simple automatic spherical parameterization for genus zero meshes. The usage of a spherical conformal parameterization would eliminate the need to cut a mesh. Additionally, spherical Voronoi diagrams are known to be computable on the GPU (5).

Spherical conformal mapping

A spherical conformal map of a mesh is considered to be centered if it has the following property: suppose the boundary of a sphere where to contain one point (having a mass attribute distributed uniformly or otherwise) per point on a defined triangle mesh. If the sphere is centered, it will not rotate on a pivot point located at its center, if gravity is allowed to act on the sphere in its natural way. Baden et all propose a method to obtain a centered spherical mapping from a mesh (6). An initial conformal mapping of a mesh to a sphere centered at the origin can be obtained in some way (perhaps by Gaussian mapping). Then a vector µ pointing at the center of mass of the sphere is computed. µ is given by the sum of the sphere's surface area contained in a mapped triangle multiplied by that triangle's centroid, approximated as the average of its three vertices as mapped to the sphere. If the magnitude of µ is within some user-defined tolerance to the origin, the algorithm terminates. Otherwise, the Jacobian matrix with respect to the center of mass is determined. The Jacobian is then inverted and multiplied by µ to obtain the inversion center *c*. For each vertex v as mapped to the sphere, the inversion is applied: each vertex is moved to the coordinates given by the equation *c* + (1 - |*c*|**2) * (v + *c*)/|v + *c*|**2.

A completely different approach: the heat flow algorithm for geodesic distance computation

The heat flow algorithm, proposed by Crane et al., simulates a Poisson problem to determine approximate geodesic distances (7). The authors state that their method is particularly useful for applications that "require repeated distance queries on a fixed geometric domain". A Voronoi diagram on a discrete manifold can be most simply obtained by computing the distances between each mesh vertex and each Voronoi site and assigning each vertex to its closest site, and this method requires only *s* matrix operations to determine the distance from each vertex to each site, where *s* is the number of sites. The authors also state that the solution of a Poisson problem has guaranteed subquadratic complexity, and that the complexity tends to be closer to linear in practice.

The heat flow method simulates the application of heat to a source vertex or set of source vertices and the diffusion of the heat to the rest of the mesh over time. Crane et al. state that heat diffusion on a surface can be imagined “a large collection of hot particles taking random walks starting at [the source vertex or vertices]: any particle that reaches a distance point [on the mesh] after a small time *t* has had little time to deviate from the shortest possible path”. Therefore, there is a direct relationship between the number of “heat particles” at a vertex and its geodesic distance from the source.

First, the heat flow at each vertex is simulated by solving the equation (*A – tLc)u* = *u_start*, where *u_start* is a vector containing 1 at *u_start*[*i*] if vertex *i* is a heat source vertex and 0 otherwise, *A* is a square diagonal matrix containing the area of each vertex (equal to the total area of its incident triangles divided by 3), *t* represents the amount of time the heat has to diffuse (experimentally determined by Crane et al. to be adequately represented by the mean edge length of the mesh), and *Lc* is the cotangent matrix of the mesh. The cotangent matrix *Lc* is a square matrix of size |V| by |V|, where V is the set of vertices in the mesh. The value *Lc*[i, j] is equal to half the sum of the cotangents of the two angles opposite the edge between vertices i and j. The cotangent matrix is not only intrinsic to the mesh but also positive definite, meaning that it can be pre-factored using Cholesky decomposition for use in all geodesic distance calculations on the mesh. Solving this system of equations gives a matrix *u* where *u*[i] contains the heat flow to vertex i from all its neighbors at time *t*.

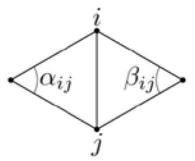

Figure 2: geometric representation of Lc[i, j]

Next the gradient of the heat flow in each triangle of the mesh can be computed. The gradient of a triangle *face* is given by the heat flow at vertex i, multiplied by the cross product of the triangle edge opposite i and its normal vector, for each vertex i in *face*. The integrated divergence of this gradient is then computed at each vertex.

The divergence of a gradient field, intuitively, gives a value for each point in the field demonstrating how “outgoing” that point is, i.e. how much it is acting like a source or acting like a sink. Therefore, vertices that contain more heat particles will be more source-like and vertices that contain fewer heat particles will be more sink-like. A second matrix equation must be solved here: a vector containing the integrated divergence for each vertex can be obtained by calculating the sum of the dot products of each edge containing that vertex with the gradients at the faces containing those edges. Since a Poisson equation is given as the Laplacian operation on a matrix phi = the divergence of some vector field X, the equation *Lc*(phi)=*b*, where *b* contains the integrated divergence at each vertex.

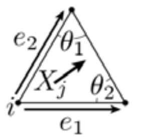

Figure 3: geometric representation of a component of the divergence at vertex i

Experiments and Discussion

The heat flow method has recently been added to the Computational Geometry Algorithms Library. The author utilized this feature to demonstrate the computation of Voronoi diagrams on triangle meshes, obtaining the following images for a mesh containing 8896 vertices and 17788 triangles, and a Voronoi diagram with 14 sites. This mesh was obtained using the custom level set example code from the pygalmesh library with an angle bound of 30, and radius and distance bounds of 0.03. Computation time was slightly less than one second per Voronoi site.

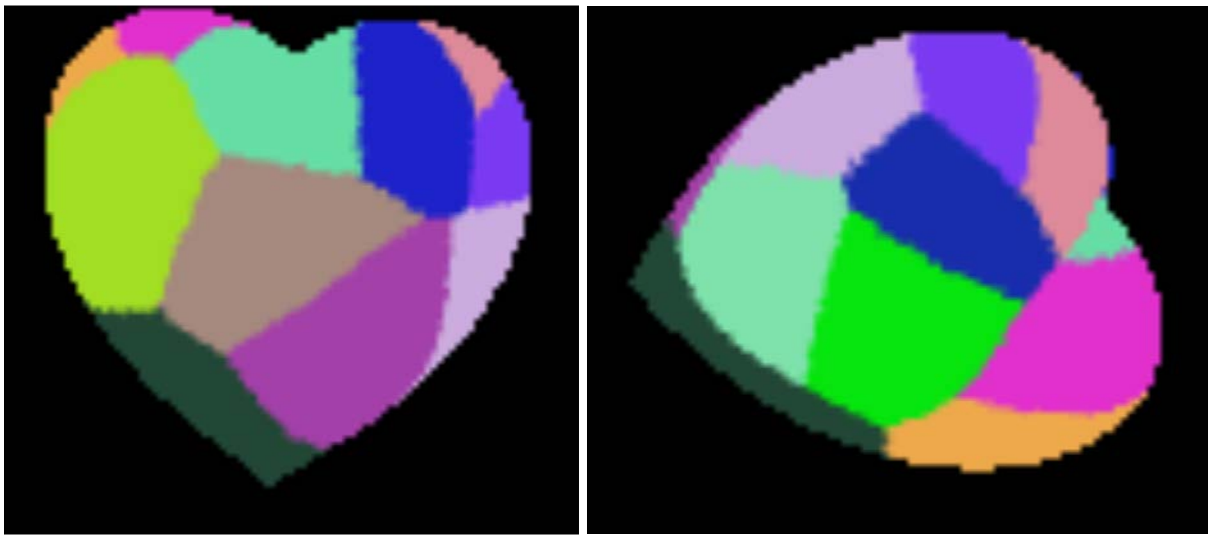
Figure 4: views of 3d mesh with vertex colors assigned using the geodesic distances from randomly selected Voronoi sites. The mesh appears flat because a flat material shader was used; however, two different views are provided to illustrate its three-dimensional properties

The heat flow algorithm is clearly a good candidate for the computation of a geodesic Voronoi diagram, but the CGAL implementation may not be the best choice to use if one wishes to obtain a centroidal Voronoi diagram: in order to obtain the new centroid for a Voronoi cluster, the geodesic distances between all points in the cluster must be computed individually, which essentially means solving a matrix for each individual vertex on the surface.

The author proposes that the new centroid of a Voronoi cluster may be quickly computable using a modified heat method, however: it seems reasonable that if all vertices contained in a Voronoi cluster are considered heat sources and are also allowed to receive additional heat, and heat is allowed to diffuse, the mesh vertex with the shortest average geodesic distance to the other vertices in the cluster will be the one that receives the most heat. Therefore, it would not be necessary to compute all geodesic distances directly. This would require the values computed for an intermediate step of the geodesic distance computation, since the geodesic distance from all vertices in a Voronoi cluster to that cluster as a region will naturally be zero.

One obstacle to this proposed modification is that it seems that a vertex cannot be both a heat source and a sink if “infinite” heat is applied to all source vertices (represented by the fact that source vertices have value 1 in *u_start* if they are sources and 0 otherwise). Therefore, a vertex must be able to gain heat as it diffuses in addition to emitting it. This may be as simple as modifying *u_start* to contain nonbinary values (i.e. values between 0 and 1) to represent vertices as both sources and sinks, but a more detailed method may in fact be required. If a method to compute the heat at each vertex with each vertex acting as both a source and a sink is possible, then finding the new Voronoi site for each cluster can be done by solving one matrix per Voronoi site.

Alternatively, each edge in the mesh could be iterated over to determine if it contains a boundary between Voronoi clusters, which would occur if one edge vertex had a different label than the other. For each Voronoi cluster, a list of neighboring vertices could be compiled and used as a heat source region. It seems that the Voronoi cluster vertex with the longest geodesic distance to the region consisting of the cluster's neighbors would be the approximate centroid of the cluster.

Conclusion

This research is meant to be an exploration of possible methods that could be used for quick computation of CCVTs on arbitrary surfaces. In order to build a generalized simulation of biological tissue, input from biologists would be required. The methods presented here and other methods would need to be implemented for comparison purposes. However, the existence of computational Voronoi-based tissue modelling seems to imply that a general framework for such simulations could be of use to the biology community.

References

1. Gómez-Gálvez, Pedro, et al. "Scutoids Are a Geometrical Solution to Three-Dimensional Packing of Epithelia." *Nature Communications*, vol. 9, no. 1, 2018, doi:10.1038/s41467-018-05376-1.
2. Sanchez-Gutierrez, D., et al. "Fundamental Physical Cellular Constraints Drive Self-Organization of Tissues." *The EMBO Journal*, vol. 35, no. 1, 2015, pp. 77–88., doi:10.15252/embj.201592374.
3. Rong, Guodong, et al. "GPU-Assisted Computation of Centroidal Voronoi Tessellation." *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 3, 2011, pp. 345–356., doi:10.1109/tvcg.2010.53.
4. Gu, Xianfeng, et al. "Geometry Images." *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '02*, 2002, doi:10.1145/566570.566589.

5. Wang, L., et al. “A GPU-Based Algorithm for the Generation of Spherical Voronoi Diagram in QTM Mode.” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-4/W2, 2013, pp. 45–50., doi:10.5194/isprsarchives-xl-4-w2-45-2013.
6. Baden, Alex et al. “Möbius Registration.” *Comput. Graph. Forum* 37 (2018): 211-220.
7. Crane, Keenan, et al. “Geodesics in Heat.” ACM Transactions on Graphics, vol. 32, no. 5, 2013, pp. 1–11., doi:10.1145/2516971.2516977.