Computing Centroidal Voronoi Tessellations on the GPU

Evan F. Bollig, Advisor: Gordon Erlebacher

Department of Scientific Computing, Florida State University

bollig@scs.fsu.edu

Abstract

IN the last decade, commodity Graphics Processing Units (GPUs) specialized for 2D and 3D graphics have dramatically changed in purpose from purely video game hardware to something akin to their general purpose counterpart, the CPU. Currently capable of near teraflop speeds with gigabytes of on-board memory, GPUs have transformed from accessory hardware for a young generation to general purpose coprocessors for scientific computation.

This poster presents results from the first (to our knowledge) implementation to compute centroidal Voronoi tessellations of manifolds entirely on the GPU. To complete these tasks, a highly efficient flooding algorithm is used to produce the Voronoi tessellation, while a regularized sampling approach is employed to compute centroids of Voronoi regions. We consider simple surfaces (2-manifolds) of the form $f(u,v) \rightarrow (u,v,z(u,v))$ partitioned according to Euclidean-based metrics, with the generating points updated by a deterministic Lloyd's method.

1. Definitions

Definition 1 *$\{V_i\}_{i=1}^k$ is a tessellation of the bounded open set $\Omega \subseteq \mathbb{R}^N$ if the following are satisfied:*

- *$V_i \cap V_j = \emptyset$ for $i \neq j$*
- *$\cup_{i=1}^k \overline{V_i} = \overline{\Omega}$*

Definition 2 *Given a set of seeds $\{s_i\}_{i=1}^k \in \Omega$, the set $\{V_i\}_{i=1}^k$ is Voronoi tessellation of Ω if*

$$V_i = \{x \in \Omega \quad | \quad dist(x,s_i) < dist(x,s_j) \text{ for } i,j = 1,..,k \text{ and } i \neq j\}$$

Note that we call V_i a Voronoi region of Ω.

Definition 3 *Given the tessellation $\{V_i\}_{i=1}^k$ of $\Omega \subseteq \mathbb{R}^N$ and a general density function $\rho(x)$, the mass centroid (center of mass) of each region V_i is*

$$z_i = \frac{\int_{V_i} x\rho(x)dx}{\int_{V_i} \rho(x)dx}.$$

If

$$z_i = s_i \text{ for } i = 1,...,k,$$

then $\{V_i\}_{i=1}^k$ form a centroidal Voronoi tessellation (CVT)[1], which minimizes the energy

$$F(\{z_i, V_i\}_{i=1}^k) = \sum_{i=1}^k \int_{V_i} \rho(x)\ dist(x,z_i)^2 dx$$

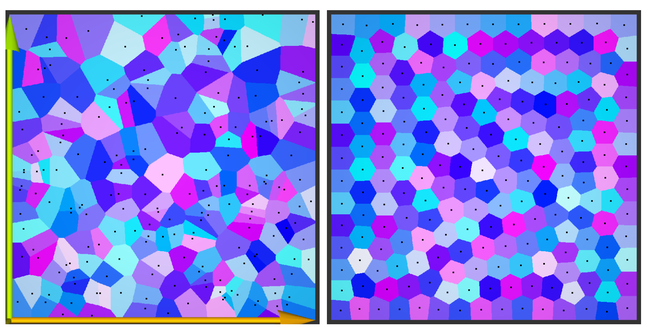

Figure 1: *(left) Voronoi vs. (right) Centroidal Voronoi Tessellation.*

Note:

- For the manifold $\mathcal{M}$ centroids may not lie on surface
- The projection of z_i (z_i^c) is the energy minimizer on $\mathcal{M}$ [2]

Definition 4 *When*

$$z_i^c = s_i \text{ for } i = 1,...,k$$

the tessellation is a constrained centroidal Voronoi tessellation (CCVT)

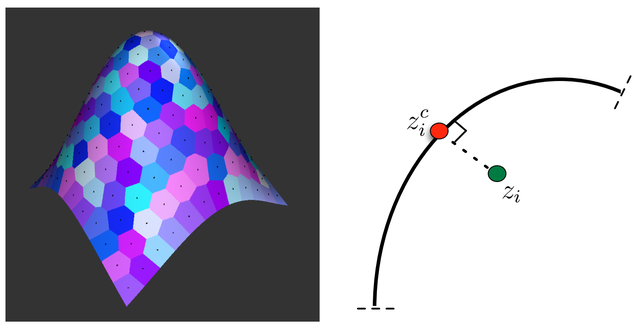

Figure 2: *(left) Constrained Centroidal Voronoi Tessellation of Manifold $f(u,v) = (u,v,e^{-(2u-1)^2-(2v-1)^2})$. (right) Centroids are "constrained" (projected) to the surface at each iteration of Lloyd's method.*

2. Why Compute on GPUs?

RECENT years have shown tremendous growth in GPU vs. CPU performance. At the same time, the new hardware has been joined by increasingly powerful graphics programming languages for general purpose computation. The latest language, CUDA, allows development of parallel kernels in C without required knowledge of graphics APIs like OpenGL.

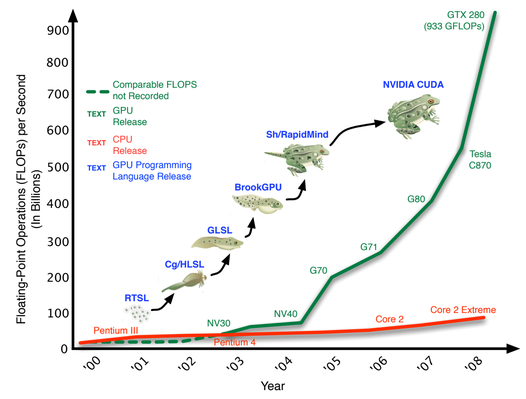

Figure 3: *Approximate GPU vs CPU performance growth and language releases over time. The dramatic growth in performance demands consideration of GPU as a computational co-processor*

3. GPU Computing

PROGRAMMING for GPUs requires knowledge of the hierarchical memory, scope of data sharing, and penalties incurred by accesses at each level.

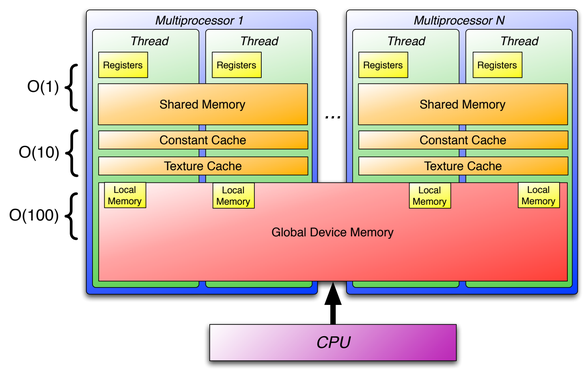

Figure 4: *GPUs have multiple levels of memory with increasing access penalties. Communication with the CPU is more costly than global device memory access and should be avoided.*

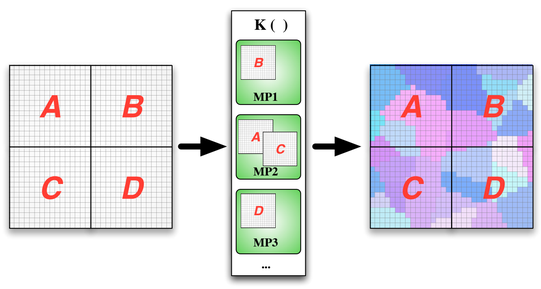

Figure 5: *Problems are decomposed into blocks of threads. Multiprocessors on the GPU apply kernels (programs) to blocks of threads in parallel.*

4. Algorithms

GENERATING CCVTs requires an iterative algorithm: Lloyd's Method.

1. Generate Voronoi tessellation using current seeds
2. Calculate centroids of each Voronoi region
3. Project centroids onto $\mathcal{M}$
4. Update current seeds to the projected centroids

Repeat 1-4 until stopping criterion is satisfied (i.e., energy is minimized or maximum iteration is reached).

Algorithm 1: *Deterministic Lloyd's Method.*

We generate the Voronoi tessellation with a fast, O(N^2log(N)) (N is one-sided domain resolution), propagation method called Jump Flooding (Variant 2 + JFA)[3].

Centroids are calculated using a custom algorithm that uniformly samples Voronoi regions with masking in 16×16 samples we call tiles. A dynamic edge scaling parameter (independently variable for each region) controls the spacing between samples according to the percentage of threads masked on the previous iteration. The complexity of this algorithm is O(t^2S) where t is number of tiles in one direction and S is number of seeds.

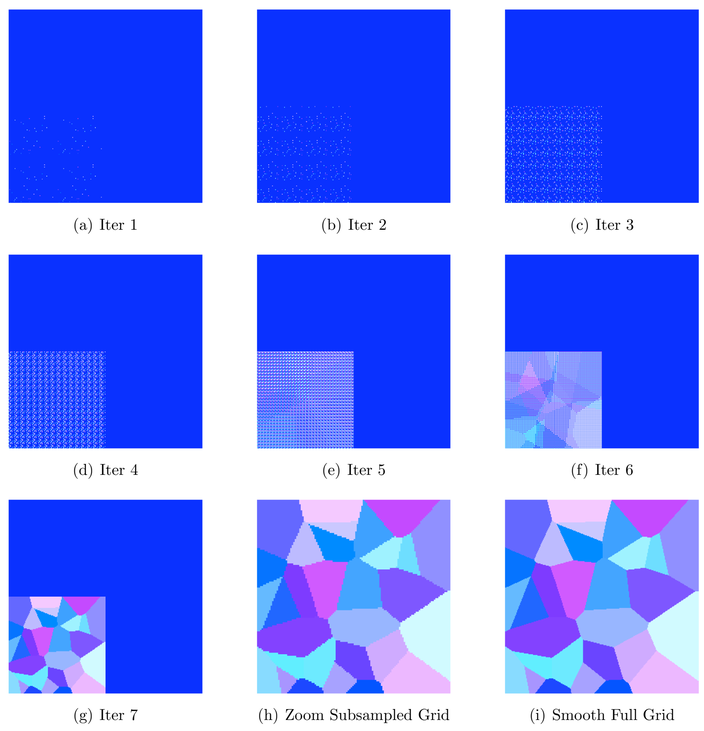

Figure 6: *Jump Flooding Variant 2 + JFA. A Voronoi tessellation is formed in $log(\frac{N}{2})$ passes by GPU then scaled to twice the resolution and corrected with a final pass.*

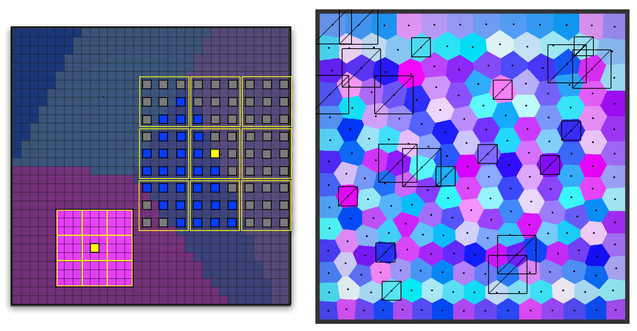

Figure 7: *(left) Uniform sampling of Voronoi regions with masking. (right) Sample spacing is independently variable for each region.*

5. Results

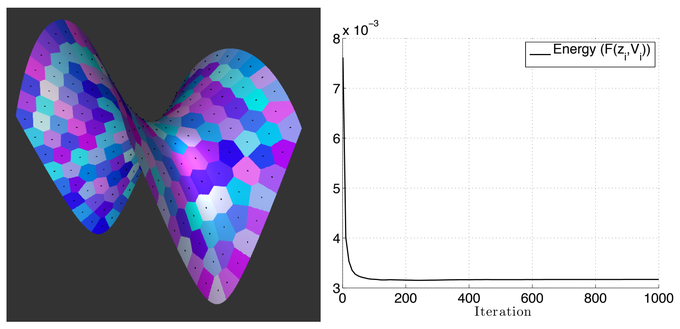

Figure 8: $f(u,v) = (u,v,\sqrt{1.1-(2u-1)^2-(2v-1)^2})$*: 1000 iterations,* 1024×1024 *resolution,* 3×3 *tiles and dynamic edge scaling.* $F(\{z_i,V_i\}) = 3.17 * 10^{-3}$

We compare our centroid calculation to the standard histogram algorithm used for centroid calculation on the CPU. Increasing resolutions have no effect on centroid calculation. With 4×4 tiles, the GPU is faster for 2000 seeds or less.

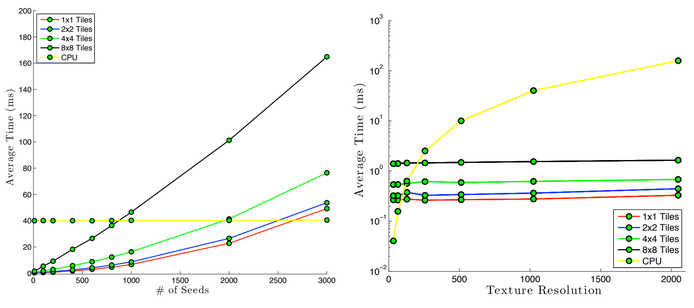

Figure 9: *Average times for (left) varying number of seeds (1024×1024 resolution) and (right) varying texture resolution (30 seeds) for our centroid calculation algorithm.*

References

[1] DU, Q., FABER, V., AND GUNZBURGER, M. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev. 41*, 4 (1999), 637–676.

[2] DU, Q., GUNZBURGER, M. D., AND JU, L. Constrained centroidal voronoi tessellations for surfaces. *SIAM J. Sci. Comput. 24*, 5 (2003), 1488–1506.

[3] RONG, G., AND TAN, T.-S. Variants of jump flooding algorithm for computing discrete voronoi diagrams. In *ISVD '07: Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 176–181.