Computing Centroidal Voronoi Tessellation Using the GPU∗

Jiaqi Zheng†
jiaqi.zheng.ac@outlook.com
Harbin Engineering University

Tiow-Seng Tan
dcostants@nus.edu.sg
National University of Singapore

ABSTRACT
We propose a novel algorithm to compute centroidal Voronoi tessellation using the GPU. It is based on the iterative approach of Lloyd’s method while having good considerations to address the two major challenges of achieving fast convergence with few iterations, and at the same time achieving fast computation within each iteration. Our implementation of the algorithm can complete the computation for a large image in the order of hundreds of milliseconds and is faster than all prior work on a state-of-the-art GPU. As such, it is now easier to integrate centroidal Voronoi tessellations into interactive applications.

CCS CONCEPTS
• Theory of computation → Computational geometry; • Computing methodologies → Graphics processors.

KEYWORDS
GPGPU, Computational Geometry, Digital Geometry, Lloyd’s Method, Voronoi Diagram, PBA

ACM Reference Format:

1 INTRODUCTION
A centroidal Voronoi tessellation (CVT) is a special Voronoi tessellation whereby each site is the centroid (center of mass) of its Voronoi region with respect to a given positive density distribution. CVTs provide good point distributions and good partitions of the spaces. Their natural optimization properties make them very popular in diverse scientific and engineering applications. One popular example is to use CVTs to optimize meshes; see [Du and Wang 2006]. Other applications include, but is not limited to image processing, data analysis, and mobile sensing networks [Du et al. 1999, 2010].

For a given positive density distribution $\rho$ defined for each $z$ in the domain, the centroid of a Voronoi region $V_i$ is defined as:

$$c_i = \frac{\int_{V_i} z \rho(z) \, dz}{\int_{V_i} \rho(z) \, dz},$$

and the energy function is [Du et al. 1999]:

$$F(z_i k) = \sum_{k=1}^{k} \int_{V_i} \rho(z) \|z - z_i\|^2 \, dz,$$

where $\{z_i\} = \{z_1, z_2, \ldots, z_k\}$ is the set of sites in the domain and $\| \cdot \|$ denotes the Euclidean norm. The minimizer of the energy function necessarily forms a CVT. A CVT which corresponds to a local minimizer of $F$ is called a stable CVT, and a CVT that globally minimizes $F$ is called an optimal CVT [Liu et al. 2009].

In this paper, we discuss CVT in the digital domain. The centroid of a Voronoi region $V_i$ is then defined as:

$$c_i = \frac{\sum_{z \in V_i} z \rho(z)}{\sum_{z \in V_i} \rho(z)},$$

and the energy function is:

$$F(z_i k) = \sum_{i=1}^{k} \sum_{z \in V_i} \rho(z) \|z - z_i\|^2.$$

In 2D, the domain is a square $[-1, 1]^2$ represented as an image of $n \times n$ pixels (or resolution, or size), whereas in 3D, a cube $[-1, 1]^3$ of $n \times n \times n$ pixels (see Figure 1). Most algorithms for constructing CVTs are iterative in nature and compute stable CVTs. There are two main challenges: reducing the number of iterations to achieve fast convergence and speeding up the computation in each iteration. We address both challenges to construct a stable CVT using GPU.

Figure 1: The left picture shows (in a cross-sectional view of) the initial tessellation while the right shows the 3D CVT obtained by our algorithm gCVT on the domain $[-1, 1]^3$ with a constant density distribution.
Our contribution in this paper is a GPU algorithm to compute a stable CVT. It employs an over-relaxation scheme on top of Lloyd’s method to reduce the number of iterations, accompanied by a novel sub-sampling scheme to speedup the total computation. Within each iteration, the algorithm employs our improved implementation of the Parallel Banding Algorithm and the dimensional reduction centroid computation of Cao et al. [2010] to achieve further speedup. Our implementation of the algorithm in 2D and 3D using CUDA significantly outperforms all known CPU and GPU-based algorithms.

The rest of the paper is organized as follows. Section 2 outlines related work on constructing CVTs. Section 3 discusses our proposed algorithm with the experimental results presented in Section 4. Section 5 concludes the paper.

2 RELATED WORK
In this section, we review the main relevant and important algorithms. In general, the algorithms for constructing CVTs are classified into two categories: Newton-type numerical methods, and geometric-based algorithms represented by Lloyd’s method.

Liu et al. [2009] proved that the energy function is $C^2$ smooth in any convex domain in 2D or 3D with a $C^2$ smooth density distribution. This provides the necessary justification for developing efficient Newton-type methods for accelerating CVT computations [Du et al. 2010]. They developed a quasi-Newton method for computing CVTs and constrained CVTs based on the L-BFGS algorithm [Liu and Nocedal 1989]. Rong et al. [2011] implemented the L-BFGS algorithm on the GPU to compute CVTs on planes and surfaces. But their method is not done entirely on the GPU; it needs to use the CPU to compute the inverse Hessian matrix and the new sites. Fei et al. [2014] improved upon this by implementing a bound-constrained version of the L-BFGS algorithm on the GPU, termed L-BFGS-B. Our experience with these methods is that they can achieve (insignificantly) lower energy as compared to approaches based on Lloyd’s method but at the expense of much longer computation times.

Lloyd’s method is the most commonly used method to construct CVTs due to its simplicity and robustness. The energy associated with the Voronoi tessellation decreases monotonically from iteration to iteration until a stable CVT is reached [Du et al. 2006, 2010]. But the result of Lloyd’s method is influenced by the initialization of sites and convergence is generally slow. Xiao [2010] proposed an over-relaxation method to accelerate the convergence that we adopted in our work.

Vasconcelos et al. [2008] implemented Lloyd’s method on the GPU, but their algorithm on a 2D plane is not appropriate for non-uniform density distribution where each Voronoi region can possibly spread across a large part of the whole grid. Cao et al. [2010] proposed the first efficient GPU Lloyd algorithm to compute CVT by using their Parallel Banding Algorithm (PBA) to compute the Voronoi diagram needed in each iteration of Lloyd’s method. We note that Manduhu and Jones [2019] recently proposed a more efficient parallel algorithm to compute Voronoi diagram by adapting modern GPU features (warp operators) and modifying some computational steps in PBA. Their resulting algorithm is rather complex and requires many processors running concurrently to claim work efficiency in theory. We have attempted to replicate their method (as their work is not open source) with the help of the open source implementation of PBA for our problem here, but we did not have much success. Instead, we discovered a few simple tweaks that can be made to the implementation of PBA to gain good speedup. We call our version PBA$^+$ and its details are given in Section 3.3.

3 OUR PROPOSED ALGORITHM
Our proposed algorithm is as shown in Algorithm 2 (Section 3.2). Before reaching there, we first go through Lloyd’s method in Section 3.1 with our realization of the over-relaxation scheme [Xiao 2010] in GPU. The discussion in the following focuses on 2D, with some technical details about computing CVTs in 3D given in Section 3.5.

3.1 Over-relaxation Lloyd’s Method
Lloyd’s method is simple: it iterates between constructing a Voronoi diagram of sites in an image, and moving sites to the centroids until the convergent criterion is satisfied; see Algorithm 1. In general, the convergent criterion (Line 2) is satisfied when there are no significant changes in the locations of centroids in the current iteration from the last. The construction of the Voronoi diagram at Line 3 minimizes the energy function for a fixed set of sites, and moving the sites at Line 5 to the centroids calculated at Line 4 minimizes the energy function for a fixed tessellation [Moriguchi and Sugihara 2008]. Line 3 (and the same at Line 7) is discussed further in Section 3.3, and Line 4 in Section 3.4.

At Line 5, it is clear that moving sites to centroids is a reasonable attempt to reach the convergent criterion. But, it is often not the best answer until near the end of the convergent process. This is because, by doing so, Lloyd’s method still needs many more subsequent iterations to reach a stable CVT. Indeed, Xiao [2010] studied this issue mathematically and showed that it is good to consider moving each site as a linear combination of its current position and the centroid of its Voronoi region. That is, assume that $z_i^t$ is the current site and $c_i^t$ is the centroid of Voronoi region $V_i$, the

---

**Algorithm 1: Lloyd’s Method**

```
Algorithm 1: Lloyd’s Method

input : Ω, the domain of CVT
        $\rho$, a density distribution on Ω
        $k$, number of sites
output: $\{V_i\}_1^n$, the CVT on Ω

1. Initialize the sites $\{z_i\}_1^n$ on Ω
2. while the convergent criterion is not satisfied do
   3. Construct the Voronoi diagram $\{V_i\}_1^n$ for $\{z_i\}_1^n$
   4. Compute the centroid $c_i$ of each Voronoi region $V_i$
   5. Replace the sites $\{z_i\}_1^n$ with the centroids $\{c_i\}_1^n$
3. end
4. Construct the Voronoi diagram $\{V_i\}_1^n$ for $\{z_i\}_1^n$
5. return $\{V_i\}_1^n$
```

---
new position $z_{i}^{t+1}$ can be:

$$
  z_{i}^{t+1} = \omega \mathcal{L}_i^t + (1 - \omega) \overline{x}_i^t = \overline{x}_i^t + \omega (c_i^t - \overline{x}_i^t),
$$

where the parameter $\omega$ should lie between 0 and 2 to guarantee the convergence of the method. When $\omega = 1$, it is the original Lloyd’s method, and when $\omega > 1$ it is an over-relaxation with a bigger move than the normal one.

Line 5 realised with Equation (3) is an over-relaxation Lloyd’s method. Our experience shows that $\omega$ no larger than 2 is good for most situations to reduce the number of iterations needed as compared to the original Lloyd’s method. We reckon this suggests that taking a bigger step has a good chance to further reduce the energy. On the other hand, when the algorithm is close to reaching the convergent criterion, a large $\omega$ is not wise as it can cause sites to oscillate around some regions from iteration to iteration. Our experiments led us to the following good setting of $\omega$ related to the rate of reduction of energy $\nabla F$ (per iteration):

$$
  \omega = \min(2, 1 + \kappa \|\nabla F\|),
$$

where $\kappa$ is a constant to moderate $\|\nabla F\|$ by gradually shifting $\omega$ away from the largest value of 2, during over-relaxation, towards 1 as in the original Lloyd’s method. Note that this approach requires the computation of the energy function (Equation (2)) for an image, which can be done with a parallel prefix sum of all pixels in the image. Though it can be efficiently calculated, parallel prefix sum should be employed discreetly to amortize its cost to still benefit from adapting over-relaxation. A good balance we found experimentally is to do the computation of energy once in every 10 iterations.

For some applications, it may be good or appropriate to fix the maximum number of iterations $l_{\max}$ allowed to better control the total computation time. We thus have the following setting of $\omega$ to suggest:

$$
  \omega = 1 + \frac{l_{\max} - l}{l_{\max}},
$$

where $l$ is the current iteration number. That is, the value of $\omega$ decreases with the increase in iteration number. When $l$ reaches $l_{\max}$, we have $\omega = 1$ which is the same as Lloyd’s method and the program must terminate. Note that this approach does not require the calculation of the energy for an image.

Besides the above considerations on over-relaxation and tweaks to prior work of PBA appearing in our proposed algorithm, we have developed a sub-sampling scheme for use in computing CVTs to greatly reduce the total computation time as discussed next.

3.2 Sub-sampling Scheme

Our algorithm, called gCVT, wraps a sub-sampling scheme over the over-relaxation of Lloyd’s method; see Line 5 to Line 14 of Algorithm 2. The hope is that constructing a CVT in a 2D image, $\Omega_0$, can be done through constructing a series of CVTs progressively for a smaller image, $\Omega_\mu$, to a larger one, $\Omega_{\mu-1}$, by doubling the size till we reach $\Omega_0$. Specifically, the 2D domain $[-1, 1]^2$ is $\Omega_0$ with $n \times n$ pixels. Then, each pixel of $\Omega_\mu$ (on the same domain) is obtained from a region of $2^\mu \times 2^\mu$ pixels of $\Omega_0$, and is assigned a density equal to the average density of these pixels (Line 2). The construction of a CVT for $\Omega_\mu$ is generally a few times cheaper than that for $\Omega_{\mu-1}$, and the approximation to a solution of $\Omega_\mu$ can then be mapped into $\Omega_{\mu-1}$ and be a good approximation to repeat the next level of calculations till $\Omega_0$ is solved. In doing so, this sub-sampling scheme can solve the given problem in an overall shorter time than other methods working on the whole problem directly at the start.

In the following, we overload $\Omega_\mu$ to mean the digital Voronoi diagram, or CVT of $\Omega_\mu$ when the context is clear. We write simply $F(\Omega_\mu)$ to mean the same as $F(\{z_i^k\})$ with $\{z_i^k\}$ being the set of sites in $\Omega_\mu$. We use $F(\Omega_{\mu-1})$ to mean the energy function at some specific $\mu$th iteration during the computation.

Though the sub-sampling scheme may seem like a straightforward idea to deploy, we explain in the following the intricacies behind why it works and how we implement it in practice. The story centers around Figure 2. It shows the 3 energy curves for $\Omega_0 = 8K \times 8K$, $\Omega_1 = 4K \times 4K$, and $\Omega_2 = 2K \times 2K$ as generated by Lloyd’s method (Algorithm 1). Note that the energy value for a pixel in $\Omega_\mu$ has to be multiplied by $4^\mu$ to get to the same scale as that of $\Omega_0$. The interesting observations are: all curves overlap for some initial iterations (details in Section 3.2.1), then each pair diverges thereafter at some iteration (details in Section 3.2.2), and finally each one ends flat with energy inversely proportional to the image size (details in Section 3.2.3). The same observations are noticed with other smooth density distributions, just like the constant density distribution used here.

3.2.1 Overlapping in saving computation time. Just on the overlapping portion in Figure 2, CVT of $\Omega_\mu$ is a good approximation to that of $\Omega_{\mu-1}$ because $F(\Omega_\mu)$ and $F(\Omega_{\mu-1})$ are of similar values. When this is so, we can take advantage to save time in computing

### Algorithm 2: gCVT

**input**: $\Omega_0$, an image representing the domain $\rho_0$, a density distribution on $\Omega_0$ $k$, number of sites $\mu_{\max}$, the maximum value of $\mu$

**output**: $\{V_i\}_1^k$, the CVT on $\Omega_0$

1. for $\mu = 1$ to $\mu_{\max}$ do
2.  Initialize the density distribution $\rho_\mu$ on $\Omega_\mu$
3.  end
4.  Initialize the sites $\{z_i\}_1^k$ on $\Omega_{\mu_{\max}}$
5. for $\mu = \mu_{\max}$ to 0 do
6.  while the convergence criterion on $\mu_{\mu_{\max}}$ is not satisfied do
7.     Construct the Voronoi diagram $\{V_i\}_1^k$ for $\{z_i\}_1^k$
8.     Compute the centroid of each Voronoi region
9.     Update the sites $\{z_i\}_1^k$ using Equation (3)
10. end
11. if $\mu \neq 0$ then
12.     Map the sites $\{z_i\}_1^k$ to $\Omega_{\mu-1}$
13. end
14. end
15. Construct the Voronoi diagram $\{V_i\}_1^k$ for $\{z_i\}_1^k$
16. return $\{V_i\}_1^k$
consider overlaying the Voronoi diagrams of \( \Omega_i \) and \( \Omega_{i-1} \) allowing one to pick a very large \( \mu \) to perform computations on larger images.

3.2.2 Divergence in reducing number of iterations. With reference to Figure 2, the two curves of \( \Omega_{i} \) and \( \Omega_{i-1} \) diverge (i.e. no longer overlap) at some \( i \)th iteration. We call this the divergent point of \( \Omega_{i} \).

That is, \( \Omega_{i} \) can get worse in approximating \( \Omega_{i-1} \) at each additional iteration of computation. However, the computation with \( \Omega_{i} \) is still a few times (in general) cheaper to that with \( \Omega_{i-1} \). So, we may still enjoy the inexpensive computation to obtain some small reduction in energy. Specifically, the reduction in energy (per unit time) in using \( \Omega_{i} \) for computation must be higher than that when \( \Omega_{i-1} \) is used for us to continue to work with \( \Omega_{i} \). Otherwise, it is desirable to switch to using \( \Omega_{i-1} \).

Notice in Figure 2 that each energy curve of \( \Omega_{i} \) has a much lower rate of reduction after its divergent point. Recall in realizing the over-relaxation with Equation (4), we calculate the energy and thus rate of reduction in energy at some regular interval. With that, we can piggy back to switch (at Line 6 of Algorithm 2) to a larger image when \( \| \nabla F \| \) reaches some value, say 0.3 (in our experiments).

As an alternative, if one does not compute the energy function at some regular interval (such as when Equation (5) is employed), we have the following consideration to activate the switching from \( \Omega_{i} \) to \( \Omega_{i-1} \). When the sites in \( \Omega_{i} \) after each iteration is not far away from their intended new locations, moving the sites to their new locations will result in little change in the energy in the digital Voronoi diagram. Thus, one can carry out a switch when the maximum overall square distance between a site to its corresponding new location is no larger than \( 2 = (1^2 + 1^2) \) signifying that each site is less than 2 pixels away from the centroid of its Voronoi region.

The divergent points for successively larger images are at larger iteration numbers as shown in Figure 2. That is, when the divergent point for \( \Omega_{i} \) is at iteration \( i \) and that for \( \Omega_{i-1} \) at iteration \( j \), then \( i < j \). This is clear as \( \Omega_{i} \) is a better approximation for \( \Omega_{i-1} \) with more pixels in \( \Omega_{i} \) when \( \mu \) is smaller. Putting all these together, the algorithm (at Line 6 of Algorithm 2) doubles the image size at each switch to avoid unnecessary iterations that would otherwise make very little changes to the energy (flat part in Figure 2), and thus arriving at the final result faster.
3.2.3 Ending in monotonicity. Our sub-sampling scheme is an attempt to arrive at the lower envelop, \( F(\Omega^0) = \min_\mu F(\Omega^0_\mu) \), of the energy curves (as presented in Figure 2) in an inexpensive manner. In practical terms, we attempt to surf (search for an answer) along the high rate of reduction of energy for each image from a small size to the original one, and avoid iterations at those low rate of reduction in energy. In the following, we make it clear that energy values decrease monotonically along the whole computation.

The whole computation can be summarized in the following sequence:

\[
\begin{align*}
\Omega^0_{\mu_{\text{max}}} & \rightarrow \Omega^1_{\mu_{\text{max}}} \rightarrow \cdots \rightarrow \Omega^i_{\mu_{\text{max}}} & \text{switch} & \rightarrow \Omega^0_{\mu_{\text{max}}-1} \rightarrow \Omega^1_{\mu_{\text{max}}-1} \\
& \cdots \rightarrow \Omega^j_{\mu_{\text{max}}-1} & \text{switch} & \rightarrow \Omega^0_{\mu_{\text{max}}-2} \rightarrow \Omega^1_{\mu_{\text{max}}-2} \rightarrow \cdots \rightarrow \Omega^0_{\mu_{\text{min}}} \rightarrow \Omega^1_{\mu_{\text{min}}} \\
& \text{where switch} & \rightarrow \end{align*}
\]

indicates a switch from an image to one double its size. Before a switch to a larger image, the energy decreases monotonically according to the original Lloyd’s method (without employing the over-relaxation). A switch from \( \Omega^i_{\mu} \) to \( \Omega^i_{\mu-1} \) is such that \( F(\Omega^i_{\mu}) \) is of the same energy value if we view \( \Omega^0_{\mu_{\text{min}}} \) as the \( \mu_{\text{min}} \) resolution as \( \Omega^i_{\mu} \). Note that the Voronoi diagram for \( \Omega^i_{\mu} \) is not computed in Algorithm 2 at the point of switching even though the new locations for the sites are known from \( \Omega^i_{\mu_{\text{max}}} \) to \( \Omega^i_{\mu} \). So, \( F(\Omega^i_{\mu_{\text{max}}}) < F(\Omega^i_{\mu-1}) \) in the switch, and thus the sequence of energy in the whole computation is also strictly decreasing. From another perspective, the processing of \( \Omega^0_{\mu_{\text{max}}} \) through \( \Omega^0_{\mu_{\text{min}}} \) can be seen as an efficient way to perform initialization on the original image \( \Omega^0 \) before starting its computation.

In our implementation, we take a shortcut on the mapping at Line 12 of Algorithm 2. A pixel at \((x, y) \in \Omega^0_{\mu}\) corresponds to four pixels \((2x, 2y), (2x, 2y+1), (2x+1, 2y), \) and \((2x+1, 2y+1) \) in \(\Omega^0_{\mu-1}. \)

If this pixel is to be the new location of a site (that is, it is a centroid) for \(\Omega^i_{\mu} \), at the point of switching to \(\Omega^i_{\mu-1} \), then the site’s location is simply approximated at \((2x, 2y) \) in \(\Omega^0_{\mu-1} \), rather than doing an accurate calculation to decide on which of the four pixels is correct for the mapping. This may invalidate \( F(\Omega^i_{\mu}) = F(\Omega^0_{\mu-1}) \) in the above discussion, but we did not notice any obvious shortcomings when using this shortcut in practice.

3.3 Digital Voronoi Diagram Computation

PBA [Cao et al. 2010] efficiently computes a Voronoi diagram in an almost linear work manner. It maximizes the use of the computational power of GPU by dividing work into bands to increase the GPU occupancy rate. To compute a Voronoi diagram for a 2D image, PBA consists of three phases: Phase 1 (with parameter \( m_1 \)) computes the 1D Voronoi diagram for each row, which is finding the closest site \( S_{i,j} \) in row \( j \) for each pixel \((i, j) \); Phase 2 (with parameter \( m_2 \)) computes the proximate sites \( P_j \) for each column \( i \) from \( S_i = \{ S_{i,j} | S_{i,j} \neq \text{NULL}, j = 0, \ldots, n-1 \} \); Phase 3 (with parameter \( m_3 \)) computes the closest site for each pixel in column \( i \) using the proximate sites \( P_i \). Our work here, termed PBA\textsuperscript{+}, is adapted from the open source implementation of PBA. A marked difference is that PBA\textsuperscript{+} uses global memory instead of texture memory to store images. This allows us to work on large images with no noticeable difference in performance.

Similar to the original, threads in a warp process the data column by column in order to coalesce memory loads and stores. Note that Phase 1 processes the data in a row-major order, while Phase 2 and Phase 3 do so in a column-major order. Due to this, PBA requires 2 explicit transpose operations after Phase 1 and Phase 3. In PBA\textsuperscript{+}, the explicit operations are avoided by doing in-place transposition where required while writing the results from a previous phase to shared memory.

While processing each column \( i \) in Phase 2, the algorithm removes those sites in \( S_i \) whose Voronoi regions do not intersect with column \( i \). Consider three sites \( a, b, \) and \( c \) in \( S_i \), where \( a = (x_1, y_1) \), \( b = (x_2, y_2) \), \( c = (x_3, y_3) \) and \( y_1 < y_2 < y_3 \). Let \((i, u)\) denote the point on column \( i \) that is equidistant from \( a \) and \( b \), and \((i, v)\) denote another point on column \( i \) that is equidistant from \( b \) and \( c \). The Voronoi region of \( b \) does not intersect with column \( i \) if \( u > v \), because no pixel is closer to \( b \) than to \( a \) or \( c \). As such, we do not need to compute \( u \) and \( v \) exactly as done by PBA which requires floating point arithmetic. Instead, we use the following condition which can be computed using only integer operations:

\[
\begin{align*}
&[(y_2 - y_1)(y_1 + y_3) + (x_2 - x_1)(x_1 + x_2 - 2i)](y_3 - y_2) > \\
&[(y_3 - y_2)(y_2 + y_3) + (x_3 - x_2)(x_2 + x_3 - 2i)](y_2 - y_1).
\end{align*}
\]

In Phase 3, the sites in \( P_j \) are linked as a list in order of increasing \( y \)-coordinate where their Voronoi regions intersect with column \( i \) in the same order. A simple method is to process the pixels in increasing order of \( j \) from \( 0 \) to \( n-1 \) by maintaining a pointer pointing to the closest site of the current pixel. If the next site in the list is closer to the current pixel, then the pointer is updated to the next place. For each column, PBA uses \( m_3 \) threads to deal with \( m_3 \) consecutive pixels at a time. Each thread finds the closest site of the corresponding pixel. Once all the threads finish their work, they synchronize and move on to the next \( m_3 \) pixels. We have observed that this synchronization causes significant waste due to threads doing no work while waiting. To workaround this, our implementation PBA\textsuperscript{+} simply assigns the total work needed for a column \( i \) (with \( n \) total pixels) equally to \( m_3 \) threads at the start. That is, each \( k \)\textsuperscript{th} thread (for \( 0 \leq k < m_3 \)) is to process the pixels \((i, \alpha m_3 + k)\) sequentially, for \( \alpha = 0, 1, \ldots, \frac{n-1}{m_3} \), and independently from all the other threads.

3.4 Centroid Computation

Adopting [Cao et al. 2010], we use a dimension reduction method to compute the centroids. This process can be viewed as the reverse process of PBA where a \( d \)-dimensional Voronoi region of a site is evaluated at each pixel of its Voronoi region and stored to its projection in \((d-1)\)-dimension, and be repeated until \( d = 1 \) when all the values are added to the site. Specifically for the 2D case with a site \( s = (x_s, y_s) \) and its Voronoi region \( V_s \), we add up all the values \( z(p) \) and \( \rho(z) \) in \( V_s \) in row \( j \) to \((x_s, j)\). Then, all the needed partial sums for site \( s \) appear in the same column \( x_s \). Next, we sum up all these values in the 1D Voronoi region to \((x_s, y_s)\) and then compute the centroid of \( V_s \) according to Equation (1); see Figure 4.

Note that the above 2D to 1D projection over all Voronoi regions will not result in any entry in a column used by more than one site. Suppose on the contrary that we have site \((x, y_p)\) and site \((x, y_q)\) both using entry \((x, j)\) to store their partial sums; see Figure 5(a). Then, there is a \( j \) such that \( \|i - y_p\| = \|j - y_q\| \). That means there are pixels \( p' \) in Voronoi region \( V_p \) and \( q' \) in Voronoi region \( V_q \) both projected to \((x, j)\). However, \( p' \) is then of equal
The Voronoi region of a site is stored in the same column as the site. This results in better utilization of resources on the modern GPUs compared to the approach of [Cao et al. 2010] which uses one thread to process an entire column. For each band, we sweep pixels from top to bottom and maintain the sum of the previous values. If a pixel is encountered where the next pixel after it belongs to a different site, we add the current accumulated sum to the current pixel using an atomic operation and reset the sum to 0. If the pixel is at the end of a band, we also use atomic addition to accumulate the sum across bands for a site. Note that the atomic addition does not cause much penalty to performance. This is because the 1D Voronoi diagram of each site only borders with others at most once, and each Voronoi region crosses at most a small number of bands.

### 3.5 Three-dimensional gCVT

Our algorithm can be extended to 3D with the same structure as presented in Algorithm 2, with the following three notes.

First, for the sub-sampling scheme, one pixel in \( \Omega_p \) denotes eight pixels in \( \Omega_{p-1} \). The saving from processing \( \Omega_p \) instead of \( \Omega_{p-1} \) is even greater here than in the corresponding 2D case. Furthermore, the discussion in support of the use of the sub-sampling scheme in the 2D case remains valid in 3D and the over-relaxation with Equation (3) remains the same for relocating sites towards the centroids.

Second, PBA\(^+\) also works for computing 3D Voronoi diagrams. We first compute the 1D Voronoi diagram for each column \((i, j)\) by employing the two-pass sweeping along the z-axis. Note that there is already good occupancy rate for 3D here with one thread per \((i, j)\), and banding has diminishing advantage as the image size increases.

Next, we run Phase 2 and Phase 3 along the y-axis, after which the 2D Voronoi diagram for each plane \(i\) is constructed. Phase 2 and Phase 3 are repeated again along the x-axis to obtain the 3D Voronoi diagram for the whole grid. Note that transpose operations are needed to swap the \(x\) and \(y\) coordinate in each run of Phase 3 (that is, we employ a transpose for each plane along the z-axis), and each 3D coordinate is compressed into a single 32-bit integer to allow processing of larger 3D images.

Third, a Voronoi region in 3D for a cubical domain is a convex body. We could compute its centroid by reducing the values in its region to its projection on 2D, then 1D, then to a single value. Unfortunately, the 2D’s method of pre-computing prefix-sums is no longer feasible for 3D due to limited memory on the GPU. Thus, the reductions have to be computed directly. In practice, however, we find that it is better to skip the intermediate 1D projections by computing the final single values directly from the 2D projections via atomic additions, as the 2D projections are generally small enough that they can be processed without becoming a bottleneck in the whole process.

### 4 EXPERIMENTAL RESULTS

We implemented gCVT using CUDA programming model of NVIDIA [2019]. We used an Intel i7-8700 CPU with a GTX 1080 Ti graphics card with 11GB of video memory as the main machine for our experiments to obtain Table 1 and Table 2. We note that there are CPU work on computing CVT utilizing CGAL such as [Liu et al. 2009], but CPU approaches generally take at least an order of magnitude slower than GPU approaches reported here. We thus do not include CPU approaches in our experiments.
Algorithm

<table>
<thead>
<tr>
<th>Density Distribution</th>
<th>Algorithm</th>
<th>Energy Value</th>
<th>Number of Iterations</th>
<th>Time (msec)</th>
<th>Voronoi (msec)</th>
<th>Centroid (msec)</th>
<th>Others (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_1 = 1$</td>
<td>Lloyd's (PBA)</td>
<td>351.26</td>
<td>120</td>
<td>357</td>
<td>154</td>
<td>176</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>gCVT (PBA+)</td>
<td>344.88</td>
<td>100</td>
<td>139</td>
<td>75</td>
<td>50</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>L-BFGS-B (JFA)</td>
<td>340.22</td>
<td>140</td>
<td>2,080</td>
<td>697</td>
<td>992</td>
<td>391</td>
</tr>
<tr>
<td>$\rho_2 = \sin^2(\pi x) \sin^2(\pi y)$</td>
<td>Lloyd's (PBA)</td>
<td>58.72</td>
<td>180</td>
<td>510</td>
<td>224</td>
<td>246</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>gCVT (PBA+)</td>
<td>56.51</td>
<td>110</td>
<td>142</td>
<td>77</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>$\rho_3 = x^2 + y^2$</td>
<td>Lloyd's (PBA)</td>
<td>221.35</td>
<td>220</td>
<td>642</td>
<td>278</td>
<td>313</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>gCVT (PBA+)</td>
<td>206.24</td>
<td>180</td>
<td>288</td>
<td>153</td>
<td>107</td>
<td>28</td>
</tr>
<tr>
<td>$\rho_4 = e^{-20(x^2+y^2)} + 0.05 \sin^2(\pi x) \sin^2(\pi y)$</td>
<td>Lloyd's (PBA)</td>
<td>9.93</td>
<td>520</td>
<td>1,479</td>
<td>645</td>
<td>718</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>gCVT (PBA+)</td>
<td>9.87</td>
<td>320</td>
<td>603</td>
<td>320</td>
<td>225</td>
<td>58</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the performance on computing CVTs with 2,000 random sites on an image $\Omega_0 = 2K \times 2K$. The convergent criterion is satisfied when $\|\nabla F\| < 10^{-5}$ for Lloyd’s and gCVT, and $\|\nabla F\| < 10^{-3}$ for L-BFGS-B. We set $\mu_{\text{max}} = 2$ for sub-sampling. The Time column is the sum of the last 3 columns on computing digital Voronoi diagrams, centroids and others.

We have experimented with variants of gCVT, using Equation (4), on the density distributions found in the literature from a simple constant density distribution $\rho_1$ to a complex $\rho_4$ function among others as shown in Table 1. For sub-sampling, the rule of thumb is to choose $\mu_{\text{max}}$ such that $\Omega_{\text{max}}$ still has no less than 100 pixels on the average per Voronoi region. Indeed, for a 2K×2K image of 2,000 sites as used in Table 1, $\mu_{\text{max}} = 2$ from the rule of thumb is confirmed the best as shown in Figure 6(b).

Table 1 has only a single entry for L-BFGS-B [Fei et al. 2014]. This is because its original open source without modification could not handle the test cases other than $\rho_1$. Also, this version of L-BFGS-B uses JFA [Rong and Tan 2006] for computing digital Voronoi diagram and supersedes the earlier work by Rong et al. [2011] which can thus be omitted from comparison. Table 1 presents the results for image size of 2K×2K, though we have tested our implementation for image size up to 8K×8K. These results show that gCVT is capable of completing the computations within a second on sizable images even with complex density distributions. Our algorithm has achieved speedup of more than two times as compared to the best prior work of Lloyd’s method using PBA [Cao et al. 2010], while still arriving at energy levels comparable to all other methods.

Figure 7 shows the change in energy over the iterations. Note that the algorithm does not compute the energy every iteration during normal run. In particular, both Lloyd’s method and gCVT do the calculation of energy with parallel prefix sum [Harris 2007] once in every 10 iterations to determine the rate of reduction in energy. For gCVT, this is used to adjust the amount of over-relaxation in Equation (4) (with $\kappa = 10$) and to determine when to switch to larger image size for subsequent computation. Under careful examination, small spikes can be observed at various points in the otherwise smooth curves for gCVT. These indicate the iterations where switches occurred. In the experiment, gCVT has managed to achieve lower energy than Lloyd’s method for all cases, but is still higher than that of L-BFGS-B for the one comparable case.

Figure 8 shows the sample outputs by gCVT for $\rho_3$ and $\rho_4$.  

Figure 7: Energy curves recorded at each iteration for different density distributions.

Figure 6: For all density distributions as shown, sub-sampling can speedup the computation with a good choice of $\mu_{\text{max}} = 3$ in (a) and $\mu_{\text{max}} = 2$ in (b). On the other hand, it can also be counter-productive if $\mu_{\text{max}}$ is set too large such as $\mu_{\text{max}}$ to 4 and 3 in (a) and (b), respectively.

4.1 Overall Performance

We have experimented with gCVT, using Equation (4), on the density distributions found in the literature from a simple constant density distribution $\rho_1$ to a complex $\rho_4$ function among others as shown in Table 1. For sub-sampling, the rule of thumb is to choose $\mu_{\text{max}}$ such that $\Omega_{\text{max}}$ still has no less than 100 pixels on the average per Voronoi region. Indeed, for a 2K×2K image of 2,000 sites as used in Table 1, $\mu_{\text{max}} = 2$ from the rule of thumb is confirmed the best as shown in Figure 6(b).

Table 1 has only a single entry for L-BFGS-B [Fei et al. 2014]. This is because its original open source without modification could not handle the test cases other than $\rho_1$. Also, this version of L-BFGS-B uses JFA [Rong and Tan 2006] for computing digital Voronoi diagram and supersedes the earlier work by Rong et al. [2011] which can thus be omitted from comparison. Table 1 presents the results for image size of 2K×2K, though we have tested our implementation for image size up to 8K×8K. These results show that gCVT is capable of completing the computations within a second on sizable images even with complex density distributions. Our algorithm has achieved speedup of more than two times as compared to the best prior work of Lloyd’s method using PBA [Cao et al. 2010], while still arriving at energy levels comparable to all other methods.

Figure 7 shows the change in energy over the iterations. Note that the algorithm does not compute the energy every iteration during normal run. In particular, both Lloyd’s method and gCVT do the calculation of energy with parallel prefix sum [Harris 2007] once in every 10 iterations to determine the rate of reduction in energy. For gCVT, this is used to adjust the amount of over-relaxation in Equation (4) (with $\kappa = 10$) and to determine when to switch to larger image size for subsequent computation. Under careful examination, small spikes can be observed at various points in the otherwise smooth curves for gCVT. These indicate the iterations where switches occurred. In the experiment, gCVT has managed to achieve lower energy than Lloyd’s method for all cases, but is still higher than that of L-BFGS-B for the one comparable case.

Figure 8 shows the sample outputs by gCVT for $\rho_3$ and $\rho_4$.

4.2 Speedup Related to Features

We experimented with variants of gCVT to better understand the role of each feature we developed. As each variant requires a different number of iterations to converge, we use the average time taken per iteration to compare the performance. Figure 9 shows the speedup of our variants relative to Lloyd’s method. The importance of a feature with respect to the performance of gCVT is indicated by the magnitude of the drop in the performance recorded by the
Figure 8: 2D CVTs obtained by gCVT.

Figure 9: The speedup per iteration of our algorithm compared to Lloyd’s method on different image sizes having the same proportion of 0.06% of sites. We made variants of gCVT with certain features disabled:
- SS is without sub-sampling,
- PP without PBA* (i.e. use PBA),
- BC without banding in the centroid computation. For sub-sampling, all use $\mu_{\text{max}} = 2$.

Figure 10: The speedup of gCVT on different generations of GPUs using GTX 980 Ti as the base case. Take for example the image of size $2K \times 2K$, the parameters $(m_1, m_2, m_3, m_4)$ in gCVT working with the different GPUs are: (32, 16, 8, 32) for GTX 980 Ti, (32, 32, 8, 32) for GTX 1080 Ti, and (32, 32, 32, 32) for RTX 2080 Ti.

4.3 Performance Gain over GPU Generations

To understand how gCVT performs with the advances in GPU hardware, we ran our implementation (without modification) on 3 different generations of GPUs: GTX 980 Ti (2816 CUDA cores), GTX 1080 Ti (3584 CUDA cores), and RTX 2080 Ti (4352 cores). As our algorithm runs mainly on GPU, we assume that the differences caused by the slightly different CPUs hosting these GPUs are negligible.

4.4 Three-dimensional CVTs

There are generally very few details on the performance of algorithms computing 3D CVTs in existing works. In our experiments, we chose Lloyd’s method (with PBA) in 3D as our reference for comparison with gCVT in 3D. Besides gCVT incorporating both over-relaxation and one level of sub-sampling, we create two other variants: gCVT–OO is gCVT without over-relaxation, and gCVT–SS is gCVT without sub-sampling scheme. Note that there is no need to deploy banding in centroid computation as the occupancy rate in GPU is already high in 3D.

Table 2 shows the experimental results on a $512 \times 512 \times 512$ image. The difference in number of iterations for each case between gCVT and gCVT–OO shows that over-relaxation is effective in reducing the number of iterations, and the difference in time taken (and in time per iteration) between gCVT and gCVT–SS shows that sub-sampling is effective in reducing the total computation time. In all the experimented cases for 3D, gCVT is observed to perform better than Lloyd’s method by around two times or more.


Table 2: Comparison of the performance on computing CVTs with 10,000 random sites on an image $\Omega_0 = 512 \times 512 \times 512$. The convergent criterion is satisfied when $\|\nabla F\| < 10^{-3}$, and we use $\kappa = 10^{-4}$ in Equation (4). For sub-sampling, we have $\mu_{\text{max}} = 1$ so that the average number of pixels per Voronoi region is at least 1,000 (extended from the rationale in 2D).

<table>
<thead>
<tr>
<th>Density Distribution</th>
<th>Algorithm</th>
<th>Energy Value</th>
<th>Number of Iterations</th>
<th>Time (msec)</th>
<th>Voronoi (msec)</th>
<th>Centroids (msec)</th>
<th>Others (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_1 = 1$</td>
<td>Lloyd’s</td>
<td>286,297</td>
<td>60</td>
<td>5,926</td>
<td>2,603</td>
<td>2,965</td>
<td>358</td>
</tr>
<tr>
<td></td>
<td>gCVT</td>
<td>282,884</td>
<td>70</td>
<td>2,971</td>
<td>1,012</td>
<td>1,737</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td>gCVT-SS</td>
<td>286,281</td>
<td>80</td>
<td>4,587</td>
<td>1,590</td>
<td>2,669</td>
<td>337</td>
</tr>
<tr>
<td></td>
<td>gCVT-DO</td>
<td>281,236</td>
<td>40</td>
<td>3,371</td>
<td>1,150</td>
<td>1,981</td>
<td>240</td>
</tr>
<tr>
<td>$\rho_2 = \sin^2(\pi x) \sin^2(\pi y) \sin^2(\pi z)$</td>
<td>Lloyd’s</td>
<td>22,566</td>
<td>80</td>
<td>7,847</td>
<td>3,472</td>
<td>3,906</td>
<td>469</td>
</tr>
<tr>
<td></td>
<td>gCVT</td>
<td>21,835</td>
<td>70</td>
<td>2,906</td>
<td>985</td>
<td>1,705</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>gCVT-DO</td>
<td>22,572</td>
<td>100</td>
<td>5,402</td>
<td>1,835</td>
<td>3,176</td>
<td>391</td>
</tr>
<tr>
<td></td>
<td>gCVT-SS</td>
<td>21,671</td>
<td>50</td>
<td>4,094</td>
<td>1,362</td>
<td>2,439</td>
<td>293</td>
</tr>
<tr>
<td>$\rho_3 = x^2 + y^2 + z^2$</td>
<td>Lloyd’s</td>
<td>280,351</td>
<td>90</td>
<td>8,871</td>
<td>3,893</td>
<td>4,443</td>
<td>535</td>
</tr>
<tr>
<td></td>
<td>gCVT</td>
<td>273,528</td>
<td>80</td>
<td>3,825</td>
<td>1,318</td>
<td>2,228</td>
<td>279</td>
</tr>
<tr>
<td></td>
<td>gCVT-DO</td>
<td>280,432</td>
<td>100</td>
<td>6,295</td>
<td>2,182</td>
<td>3,661</td>
<td>452</td>
</tr>
<tr>
<td></td>
<td>gCVT-SS</td>
<td>269,972</td>
<td>50</td>
<td>4,216</td>
<td>1,440</td>
<td>2,477</td>
<td>299</td>
</tr>
<tr>
<td>$\rho_4 = e^{-20(x^2+y^2+z^2)} + 0.05 \sin^2(\pi x) \sin^2(\pi y) \sin^2(\pi z)$</td>
<td>Lloyd’s</td>
<td>2,260</td>
<td>190</td>
<td>18,220</td>
<td>8,202</td>
<td>8,932</td>
<td>1,086</td>
</tr>
<tr>
<td></td>
<td>gCVT</td>
<td>2,256</td>
<td>160</td>
<td>10,019</td>
<td>3,429</td>
<td>5,865</td>
<td>725</td>
</tr>
<tr>
<td></td>
<td>gCVT-DO</td>
<td>2,256</td>
<td>180</td>
<td>11,728</td>
<td>4,066</td>
<td>6,813</td>
<td>841</td>
</tr>
<tr>
<td></td>
<td>gCVT-SS</td>
<td>2,219</td>
<td>130</td>
<td>10,418</td>
<td>3,587</td>
<td>6,086</td>
<td>745</td>
</tr>
</tbody>
</table>

5 CONCLUDING REMARKS

This paper proposes gCVT, an efficient CVT algorithm that uses GPU, based on Lloyd’s method. It incorporates over-relaxation to speedup the convergence, and sub-sampling to greatly reduce the total computation time. Our experiments show that gCVT generally out-performs the next best method in 2D and 3D by around two times or better. It takes in the order of hundreds of milliseconds to compute CVT for reasonably large 2D images, and it can thus be used to build good tools for interactive applications. The paper also contributes to a better implementation of the PBA to compute digital Voronoi diagram. Our version performs about 40% better than the open source implementation of PBA, and is of independent interest to other applications using PBA.

This paper focuses on 2D and 3D computation of CVTs. Some considerations herein, such as the sub-sampling scheme, may still be useful for higher dimensions, though the amount of memory on current GPUs may be the limiting factor. Further investigation is necessary on the way to achieve practical use of the GPU for computing CVT for higher dimensions.

REFERENCES


