Fast Capacity Constrained Voronoi Tessellation

Hongwei Li* Diego Nehab† Li-Yi Wei† Pedro V. Sander* Chi-Wing Fu‡

*Hong Kong UST †Microsoft Research ‡Nanyang Tech. Univ.

Abstract

Lloyd relaxation is widely employed to generate point distribution for a variety of applications in computer graphics, computer vision, and image processing. However, Lloyd relaxation has several known issues that could harm distribution quality, among which the tendency to settle into semi-regular structures being the most severe. Recently, Balzer et al. [2009a] addressed this issue via a variation of Lloyd relaxation, termed capacity constraint Voronoi tessellation (CCVT). CCVT has superior quality than Lloyd relaxation, but could run orders of magnitude slower. Given the importance of Lloyd relaxation, CCVT has the potential to become a widely adopted replacement, but its slow computation could be a hindrance. In this paper, we present fast CCVT, an accelerated version of the original CCVT method. Our acceleration is composed of different methods ranging from high level parallelization and complexity reduction to low level speed optimizations. Our aggregated accelerations are orders of magnitude faster than the original method proposed by [Balzer et al. 2009a] (and $10\times$ faster than a previous accelerated implementation of the same technique) while maintaining excellent distribution quality and scaling very well as the number of points increase. We provide additional analysis for the properties of CCVT.

Keywords: capacity constrained, point distribution, Voronoi tessellation, relaxation, blue noise, sampling, stippling

1 Introduction

Lloyd's method [Lloyd 1982; Lloyd 1983] has been widely adopted in a variety of applications in computer graphics, computer vision, and image processing. Some typical scenarios include sampling, stippling, triangulation, coding, and compression [Balzer et al. 2009a; Gersho and Gray 1991]. In computer graphics, Lloyd's method is commonly employed to generate point distribution with blue noise property, i.e. points that are randomly and uniformly distributed and yet lack any regular structure that shows up as bias in the Fourier spectrum [Lagae and Dutré 2008]. However, despite its widespread usage, point distribution produced by Lloyd's method might exhibit certain regularity, causing quality problems for the intended applications.

Recently, Balzer et al. [2009a] presented an extension of Lloyd's method, termed capacity constrained Voronoi tessellation (CCVT), that is devoid of this regularity issue. Due to the importance of Lloyd's method, CCVT has the potential to become a popular replacement for many applications. Unfortunately, CCVT is computationally much more expensive than the original Lloyd's method, a potential hindrance for widespread adoption of this improved method.

previous CCVT, 342 seconds our acceleration, 36 seconds

Figure 1: Fast capacity constrained Voronoi tessellation. Shown here are two stippling results each in 16384 sites with capacity 256 points-per-site. Our method significantly accelerates the CCVT method [Balzer et al. 2009a] without compromising quality. (The timing for previous CCVT is obtained via unpublished accelerations of [Balzer et al. 2009a] in [Balzer et al. 2009b]; without these the performance will be even slower. See Table 1 for details.)

To address this performance issue, we present a fast capacity constrained Voronoi tessellation (CCVT) algorithm. Our method is extended from CCVT [Balzer et al. 2009a], but we make several algorithmic innovations that removes several performance bottlenecks of the original method. Our acceleration is composed of several different algorithms, ranging from high level parallelization and complexity reduction to low-level performance optimizations. Our fast method is able to achieve significant speedups over the original CCVT algorithm without sacrificing point distribution quality. The performance improvement by our method could facilitate the adoption of CCVT by a variety of applications.

2 Previous Work

Sampling is a core component in a variety of applications in computer graphics, vision, and image processing. Although different applications may favor different sampling methods, blue noise sampling has been one of the most popular [Lloyd 1982; Cook 1986; Mitchell 1987; McCool and Fiume 1992; Ostromoukhov et al. 2004; Jones 2006; Dunbar and Humphreys 2006; Kopf et al. 2006; Ostromoukhov 2007; Bridson 2007; White et al. 2007; Wei 2008; Fu and Zhou 2008; Balzer et al. 2009a; Cline et al. 2009]. In a nutshell, blue noise sampling produces samples that are randomly and uniformly distributed in the spatial domain, and lack low-frequency energy as well as structural bias in the frequency domain. In essence, blue noise sampling replaces low frequency bias by high frequency noise, a visually less objectionable artifact [Yellott 1983].

A blue noise sampling can be produced by one of two main approaches, relaxation [Lloyd 1982] with explicit sample count, and dart throwing [Cook 1986] with explicit sample spacing. Most of the other techniques are more or less derived from or based on these two main methods. Depending on the particular application needs, either relaxation or dart throwing would be more suitable.

Dart throwing generates samples one by one by drawing a trial sample uniformly randomly from a given sample domain. If the trial

sample is at least a minimum distance away from all existing samples, it is accepted. Otherwise, it is rejected. Since dart throwing keeps samples stationary once they are accepted, it does not tend to get trapped into local minimums and thus usually exhibits excellent blue noise properties. However, dart throwing cannot guarantee a fixed sample count, and the process can be slow as many trials can be rejected. The speed issue has been addressed by several recent techniques such as [Jones 2006; Dunbar and Humphreys 2006; White et al. 2007; Wei 2008], making dart throwing a practical technique for blue noise sampling.

Relaxation starts with a given set of samples and moves them around to so that they are as far away from each other as possible. Thus, unlike dart throwing which allows explicit control of sample spacing but not sample count, relaxation offers the exact opposite features. However, relaxation has the tendency to get trapped into local minimums, producing samples with harmful regular structures. Balzer et al. [2009a] provides an ingenious solution based on capacity constraints, eliminating this regular structure problem. However, the proposed method can be quite slow, hampering its potentially wide adoption. Unfortunately, even though related acceleration techniques exist for traditional relaxation (see e.g. [Hoff et al. 1999; Rong and Tan 2006]), they are not applicable to [Balzer et al. 2009a] due to its use of capacity constraints. Accelerating [Balzer et al. 2009a] is our main focus.

3 Background

Here, we provide a brief background description on Lloyd relaxation [Lloyd 1982] and the CCVT extension by [Balzer et al. 2009a]. Let S be a set of point samples called sites (in the jargon of [Balzer et al. 2009a]) whose distribution we wish to optimize for. The uniformity of S can be measured by the following energy function:

$$\mathbf{E}(S, \mathcal{V}) = \sum_i \int_{V_i} \rho(p) \left|p - s_i\right|^2 dp \tag{1}$$

where $\mathcal{V}$ is the Voronoi tessellation generated from S, V_i the Voronoi region corresponding to site $s_i \in S$, p a sample point in the domain Ω, and ρ a non-negative density function defined over Ω. Lloyd relaxation [Lloyd 1982] is a common method to minimize this energy function, iterating between the following two steps until meeting some termination criterion:

Voronoi generate the Voronoi tessellation $\mathcal{V}$ from the sample set S

Centroid move each site $s_i \in S$ to the centroid m_i of the corresponding Voronoi region $V_i \in \mathcal{V}$, i.e.

$$m_i = \frac{\int_{V_i} \rho(p) p dp}{\int_{V_i} \rho(p) dp} \tag{2}$$

Lloyd relaxation can be implemented in either a discrete or a continuous sample domain Ω, whereas the former represents the sample space Ω via a collection of discrete points P while the latter is without such discretization. In this paper, we focus on the discrete formulation, as it is conceptually simpler and easier to formulate, especially for high dimensional Ω for which a continuous method could be tricky to implement. Balzer et al. [2009a] also focused on the discrete setting. Furthermore, as will be detailed later, a discrete formulation also allows us to treat both uniform and adaptive sampling via the same algorithm and implementation, by simply varying the input point distribution.

Thus, in a discrete setting, Lloyd relaxation performs the **Voronoi** step by finding, for each discrete point $p \in \Omega$, the site $s(p)$ that is closest to p among all sites in S:

$$s(p) = \arg\min_{s \in S} |p - s|^2 \tag{3}$$

The Voronoi region V_i for site s_i is then defined as the collection of all points p whose site affiliation is s_i:

$$V_i = \bigcup \{p, s(p) = s_i\} \tag{4}$$

As shown, Lloyd relaxation is a very simple and elegant algorithm, and both the Voronoi and centroid computation steps can be accelerated and/or parallelized (see e.g. [Hoff et al. 1999; Rong and Tan 2006]). However, Lloyd relaxation is also known for its tendency to settle into a undesirable regular structure. Balzer et al. [2009a] observed that this problem can be fixed by enforcing a constant *capacity constraint* for all the Voronoi regions. Specifically, the capacity c of each site s_i is defined as the weighted sum over its Voronoi region:

$$c(s_i) = \sum_{p \in V_i} \rho(p) \tag{5}$$

Then, instead of Equation 4, the **Voronoi** step is performed in a way so that the capacity for each site remains invariant. This is achieved by initializing a random site to each point so that the capacity $c(s_i)$ of each site s_i meets the target value and remains so during the **Voronoi** step by allowing change of site affiliation of a point only by swapping pairs of points belonging to different sites. To ensure that such swapping indeed improves distribution quality, the operation is performed only if it reduces the energy in Equation 1. This entire process is summarized in Program 1. Balzer et al. [2009a] demonstrated that their CCVT method can significantly improve the site distribution quality. However, their method requires sequential examination of sorted list of point pairs stored in a heap data structure, a potentially slow process.

Let n be the total number of sites and m the total number of points. Then the time complexity per iteration of Lloyd relaxation is $O(m \log n)$ for a discrete space implementation, and for CCVT $O(n^2 + nm \log \frac{m}{n})$ [Balzer et al. 2009a]. Given that to obtain quality results we usually need $m \gg n$, the latter can be simplified to $O(nm \log \frac{m}{n})$.

4 Complexity Reduction

In this section, we describe high-level ideas that reduce the computational complexity from the original CCVT method in [Balzer et al. 2009a]. In Section 5, we describe more detailed low-level optimizations that do not necessarily reduce high-level time complexity but nonetheless provide speedups. In Section 6 we analyze the performance impacts of each of these methods. For easy reference and comparison, we have summarized our algorithm in Program 2.

4.1 Median site swap

As shown in Program 1, the most time consuming part of the original CCVT algorithm is in swapping site affiliations among the heap points. Let n be the total number of sites and m the total number of points. Then the time complexity of swapping between 2 sites will be $O(\frac{m}{n} \log \frac{m}{n})$ where $\frac{m}{n}$ is the average number of points per site (under the usual situation of equal capacity among all sites). Since $\frac{m}{n}$ needs to be sufficiently large (at least 256; see Section 6) to produce good results, this could be a significant performance bottleneck, especially considering that the pair-wise swapping needs to be conducted for all sites pairs, bringing the total complexity to

```
function V ← CCVT(S, P, C)
  // input: sites set S, points set P, and capacity constraints C
  //        where Σ_{s_i ∈ S} c(s_i) = |P|.
  // output: the Voronoi set V
  V ← Initialization(S, P, C)
  do
    stable ← true
    Υ ← SelectSitePairs(S)
    foreach pair of sites (s_i, s_j) ∈ Υ
      changed ← SiteSwap(s_i, s_j)
      if(changed = true) stable ← false
    end
  while(stable = false)
  return V

function V ← Initialization(S, P, C)
  V ← random assignment from P to S, preserving C
  return V

function Υ ← SelectSitePairs(S)
  return {(s_i, s_j); s_i, s_j ∈ S, i < j}

function changed ← SiteSwap(s_i, s_j)
  changed ← false
  // build the two heaps H_i and H_j
  H_i ← H_j ← ∅
  foreach point p_i with s(p_i) = s_i
    insert p_i into H_i with key |p_i − s_i|^2 − |p_i − s_j|^2
  end
  foreach point p_j with s(p_j) = s_j
    insert p_j into H_j with key |p_j − s_j|^2 − |p_j − s_i|^2
  end
  // swap site affiliations
  while |H_i| > 0 and |H_j| > 0 and max(H_i) + max(H_j) > 0
    p_i ← arg max(H_i); p_j ← arg max(H_j)
    s(p_i) ← s_j; s(p_j) ← s_i
    remove p_i/p_j from H_i/H_j
    changed ← true
  end
  return changed
```

Program 1: The original CCVT method by [Balzer et al. 2009a].

$O(n^2 \times \frac{m}{n} \log \frac{m}{n}) = O(mn \log \frac{m}{n})$. Here, we present a site swapping algorithm that has linear time complexity $O(\frac{m}{n})$ per site pair, bring down the total time complexity to $O(mn)$.

Our method is as follows. For a given pair of sites s_i and s_j, we seek the optimal assignment of points between them. Conceptually, we can first move all points to site s_i, and decide which points to move to site s_j. Each time we move a point p from site s_i to site s_j, the change in energy is

$$\Delta\mathbf{E}(p, s_i \to s_j) = |p - s_j|^2 - |p - s_i|^2 \tag{6}$$

Given that we have to move exactly $c(s_j)$ points from s_i to s_j to preserve the capacity constraints, the question now becomes which $c(s_j)$ out of a total $c(s_i) + c(s_j)$ points to move in order to minimize the above energy term. This can be achieved by splitting the points between s_i and s_j at the median point τ whose energy $\Delta\mathbf{E}(\tau, s_i \to s_j)$ lies on the median among all $c(s_i) + c(s_j)$ points (see Program 3). Since the median can be found in $O(\frac{m}{n})$ time (e.g. see quick selection algorithm by Hoare), our algorithms achieves the intended time complexity.

```
function V ← FastCCVT(S, P, C)
  V ← Initialization(S, P, C)
  do
    stable ← true
    Υ̂ ← SelectSitePairClusters(S)
    changed ← false
    foreach cluster Υ ∈ Υ̂
      parallel foreach site pair (s_i, s_j) ∈ Υ
        changed += SiteSwap(s_i, s_j)
      parallel end
    end
    if(changed = true) stable ← false
  while(stable = false)
  return V

function V ← Initialization(S, P, C)
  // Program 6

function Υ̂ ← SelectSitePairClusters(S)
  // Program 5 + bounding circle pruning in Section 5.1

function changed ← SiteSwap(s_i, s_j)
  // Program 3 +
  // median cut pruning and point distance pruning in Section 5.2
```

Program 2: Our fast CCVT method.

```
function changed ← SiteSwap(s_i, s_j)
  P_ij ← ⋃ {p, s(p) = s_i or s_j}
  compute {ΔE(p, s_i → s_j), p ∈ P_ij} according to Equation 6
  find τ so c(s_j) points ∈ P_ij have ΔE(p, s_i → s_j) < ΔE(τ, s_i → s_j)
  foreach p ∈ P_ij
    if ΔE(p, s_i → s_j) < ΔE(τ, s_i → s_j)
      s(p) ← s_j
    else
      s(p) ← s_i
    end
  end
  return τ changed from last time
```

Program 3: Our median site swap method.

4.2 Site pair selection

Another major performance bottleneck in Program 1 is the need to swap points for every possible pair of sites. Intuitively, this could be overkill, as not all pair of sites will end up swapping points, e.g. those that are geographically far away or those whose site assignments have already stabilized.

To address this issue, we present a general strategy to reduce the number of site pairs that need to be considered for the swap operation. To do so, we propose to alternate between two processing stages: a full stage which considers all pairs, and a cached state which only considers a subset of the pairs which have actually swapped points in the last iteration.

Full stage This stage follows the traditional approach by considering all n^2 pairs of sites. Sites that swap pairs in the most recent iteration are entered in a cache. When the size of the cache reaches a small factor of the number of sites (e.g., $10n$), then we transition to the cached stage.

Cached stage This stage only considers the set of pairs that are in the cache. The algorithm switches back to the full stage after k

(e.g., 5) iterations or convergence of *all* cached pairs.

function $\Upsilon \leftarrow$ SelectSitePairs(S)
  $\Upsilon \leftarrow \{(s_i, s_j); i < j, s_i\ s_j$ swapped points last iteration$\}$ // *cached stage*
  if Υ is empty // *first time or convergence of previously cached* Υ
    $\Upsilon \leftarrow \{(s_i, s_j); s_i, s_j \in S, i < j\}$ // *full stage*
  end
  return Υ

Program 4: Our site pair selection method.

4.3 Parallel site swap

Another main acceleration is that for pairs of sites that do not share common sites, their swap operations can be conducted in parallel. In order to maximize processing and minimize synchronization delays, we arrange the site pairs such that they are grouped in large clusters of independent site pairs.

Full stage For the full stage, in which all pairs are considered, we can optimally partition in clusters of $\frac{n}{2}$ pairs as shown in Program 5. Thus, we maximize parallelization with up to $\frac{n}{2}$ processors.

function $\widehat{\Upsilon} \leftarrow$ SelectSitePairClusters(S)
  $\widehat{\Upsilon} \leftarrow$ SelectSitePairClustersCachedStage(S)
  if Υ is empty
    $\widehat{\Upsilon} \leftarrow$ SelectSitePairClustersFullStage(S)
  end
  return $\widehat{\Upsilon}$

function $\widehat{\Upsilon} \leftarrow$ SelectSitePairClustersFullStage(S)
  // *k: the kth site pair cluster.* $k \in [0, n-1]$
  $\Upsilon \leftarrow \emptyset$
  for($i \leftarrow 0, j \leftarrow k$ - 1; $i < j; i \leftarrow i+1, j \leftarrow j-1$)
    $\Upsilon \leftarrow \Upsilon \bigcup (s_i, s_j)$
  end
  for($i \leftarrow k, j \leftarrow n$ -1; $i < j; i \leftarrow i+1, j \leftarrow j-1$)
    $\Upsilon \leftarrow \Upsilon \bigcup (s_i, s_j)$
  end
  return $\{\Upsilon\}$

function $\widehat{\Upsilon} \leftarrow$ SelectSitePairClustersCachedStage(S)
  $\widehat{\Upsilon} \leftarrow \emptyset$
  $\Upsilon_{init} \leftarrow \{(s_i, s_j); i < j, s_i\ s_j$ swapped points last iteration$\}$
  foreach pair $(s_i, s_j) \in \Upsilon_{init}$
    added $\leftarrow$ **false**
    foreach $\Upsilon \in \widehat{\Upsilon}$ in increasing size
      if neither s_i nor s_j belongs to any pair $\in \Upsilon$ // *no conflict*
        $\Upsilon \leftarrow \Upsilon \bigcup (s_i, s_j)$
        added $\leftarrow$ **true**
        break
      end
    end
    if not added // *conflict with all existing clusters*
      $\widehat{\Upsilon} \leftarrow \widehat{\Upsilon} \bigcup \{(s_i, s_j)\}$ // *start a new cluster*
    end
  end
  return $\widehat{\Upsilon}$

Program 5: Our site pair cluster method for parallel computation.

In the cached stage, we only have a subset of the pairs. Furthermore, we do not know the distribution or number of occurrences of each site within these pairs. We therefore propose a fast clustering algorithm that maximizes the size of the smallest cluster, thus distributing the pairs as evenly as possible. To that end, we employ a simple greedy heuristic. The algorithm greedily adds a pair at a time to one of the clusters until all pairs have been added. Each candidate pair joins the smallest non-conflicting cluster, i.e., the smallest cluster whose pairs do not share any sites with the candidate pair. If the pair to be added is conflicted with all clusters, it then starts a new cluster. Note that an upper bound on the number of clusters created by this algorithm is $d + 1$, where d is the number of pairwise conflicts of the highest conflicting pair. This is easily shown since for any candidate pair with d or fewer conflicts, there must be at least one out of $d + 1$ clusters that does not contain a pair that conflicts to the candidate pair. Figure 2 shows that this approach results in an relatively even distribution of pairs. Note that nearly all clusters in the 2048-site simulation have between 525 and 675 pairs.

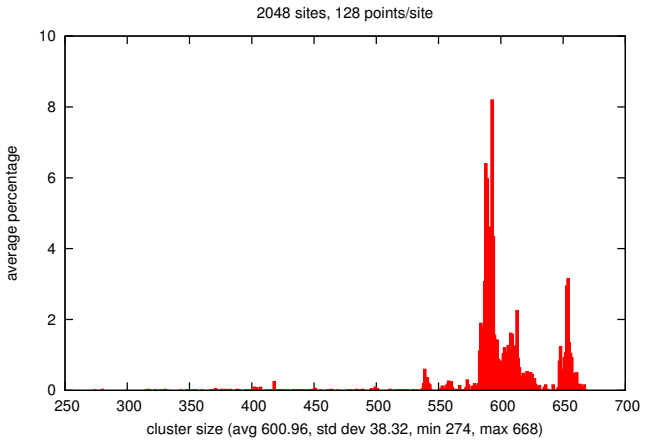

Figure 2: Histogram of the distribution of pairs among clusters. The total number of sites is 2048 with 128 points-per-site in this experiment. The average cluster size is 600.96 with minimum 274, maximum 668, and standard deviation 38.32.

This clustering is vital for the practical use of the caching technique as evidenced by our results (see Section 6).

5 Performance Optimization

Here, we describe more detailed low-level optimizations we have performed in addition to the high level ideas we presented in Section 4.

5.1 Previous CCVT optimizations

We first describe several optimizations that were deduced from the source code in [Balzer et al. 2009b] and provably conform to the original technique, but not published elsewhere.

Coherent initialization While Balzer et al. [2009a] have shown that a random point-to-site initialization is sufficient to produce good output, it is not particularly necessary. This optimization replaces the random initialization with a more *coherent* that not only obeys capacity constraints as in [Balzer et al. 2009a] but also attempts select sites that are as close to the points as possible just like traditional Lloyd method. The algorithm is summarized in Program 6. This coherent initialization still reaches a local energy minimum (for Equation 1), maintains good quality, and is significantly faster.

function $\mathcal{V} \leftarrow$ Initialization(S, P, C)
  // $\mathcal{V} \leftarrow$ *coherent assignment from* P *to* S*, preserving* C
  $S' \leftarrow S$ // *the set of sites with capacity not yet filled*
  foreach $V_i \in \mathcal{V}$
    $V_i \leftarrow \emptyset$
  end
  foreach $p \in P$
    $s_i \leftarrow \arg\min_{s \in S'} |s - p|$
    $s(p) \leftarrow s_i$
    $V_i \leftarrow V_i \bigcup p$
    if $|V_i| \geq c(s_i)$ // s_i *has reached its capacity*
      remove s_i from S'
    end
  end
  return $\mathcal{V}$

Program 6: The coherent initialization algorithm.

Bounding circle pruning In Section 4.2 we have presented a general strategy to prune the potential site pairs to consider for point swap based entirely on caching past behaviors. However, intuitively, there are more specialized pruning methods that could also be helpful. Here, we describe one possible strategy that is based on the distance between pairs of sites. Intuitively, when two sites are sufficiently far away from each other, they are unlikely to exchange points with each other and could thus be pruned.

More specifically, we first compute a bounding circle with radius r_i centered at each site $s_i \in S$ so that it contains all points assigned to s_i. Then, we can prune each pair of sites (s_i, s_j) whose bounding circles do not intersect, i.e. $r_i + r_j < |s_i - s_j|$. To see why this will work, let us compute the energy change due to exchanging points p_i and p_j between sites s_i and s_j whose bounding circles do not intersect:

$$\begin{aligned}\Delta\mathbf{E}(p_i, p_j) &= |s_i - p_j|^2 + |s_j - p_i|^2 - |s_i - p_i|^2 - |s_j - p_j|^2 \\ &> |s_i - p_j|^2 + |s_j - p_i|^2 - r_i^2 - r_j^2 \\ &> (|s_i - s_j| - r_j)^2 + (|s_i - s_j| - r_i)^2 - r_i^2 - r_j^2 \\ &= 2|s_i - s_j|^2 - 2|s_i - s_j|(r_i + r_j) \\ &= 2|s_i - s_j|\big(|s_i - s_j| - (r_i + r_j)\big) \\ &> 0 \end{aligned} \tag{7}$$

Therefore, if $|s_i - s_j| > r_i + r_j$, swapping two points will always increase the energy and we can skip the pair (s_i, s_j).

5.2 Our optimizations

Here we describe further performance enhancements beyond those in Section 4 and 5.1.

Median cut pruning When considering swapping between two sites s_i and s_j we first compute $\Delta\mathbf{E}(p, s_i \rightarrow s_j)$ for all points belonging to s_i and s_j (see Equation 6 and Section 4.1). If $\max\left(\Delta\mathbf{E}(p_j, s_i \rightarrow s_j), p_j \in s_j\right) < \min\left(\Delta\mathbf{E}(p_i, s_i \rightarrow s_j), p_i \in s_i\right)$, then the points are already optimally allocated between these two sites and the median computation can be skipped. For efficiency, we track the max and min numbers while performing the median swap (Section 4.1) without re-computing all the $\Delta\mathbf{E}$ values.

Point distance pruning This optimization prunes the list of points that need to be processed by the median computation. Consider the energy change due to exchanging points p_i and p_j between sites s_i and s_j in Figure 3:

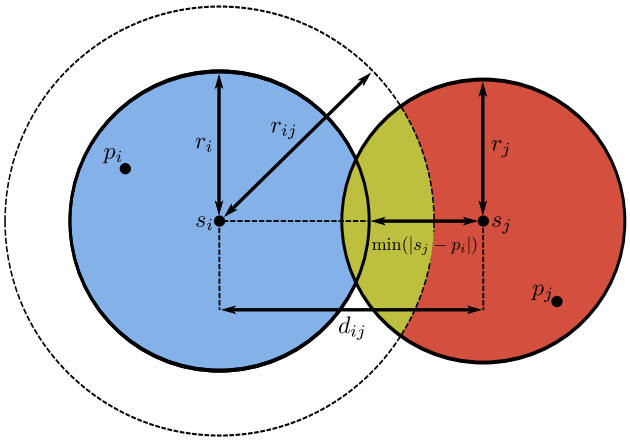

Figure 3: Point distance pruning. Only the points within r_{ij} need to be considered for swapping from s_j to s_i. Thus p_j need not be considered and can be omitted from median computation.

$$\begin{aligned}\Delta\mathbf{E}(p_i, p_j) &= |s_i - p_j|^2 + |s_j - p_i|^2 - |s_i - p_i|^2 - |s_j - p_j|^2 \\ &> |s_i - p_j|^2 + |s_j - p_i|^2 - r_i^2 - r_j^2 \\ &> |s_i - p_j|^2 + \min(|s_j - p_i|^2) - r_i^2 - r_j^2 \end{aligned} \tag{8}$$

where

$$\begin{aligned}\min(|s_i - p_j|^2) &= \max(0, (|s_i - s_j| - r_j)^2) \\ \min(|s_j - p_i|^2) &= \max(0, (|s_i - s_j| - r_i)^2) \end{aligned} \tag{9}$$

From this we see that, if $|s_i - p_j|^2 > r_{ij}^2 = r_i^2 + r_j^2 - \min(|s_j - p_i|^2)$ for some $p_j \in s_j$, then no matter where p_j is, exchanging it with any $p_i \in s_i$ will increase the energy. Therefore, when considering points p_j from site s_j to exchange with site s_i, ignore those for which $|s_i - p_j|^2 > r_{ij}^2$. Conversely, when considering points p_i from site s_i to exchange with site s_j, ignore those for which $|s_j - p_i|^2 > r_{ji}^2 = r_i^2 + r_j^2 - \min(|s_i - p_j|^2)$. Note that since all quantities on the right hand side of these inequalities are site-wise constants and thus only need to be computed once per site, making these pruning tests quite efficient.

Multilevel optimization This optimization starts the simulation with a smaller number of points m. Upon convergence, we increase m by a factor of k using the same sampling pattern as the one used for the initial point set. We repeat this process until we reach the final desired number of points in our optimization.

6 Results

Here, we analyze the quality and performance of our proposed methods.

Quality Being a variant of Lloyd relaxation, the quality of CCVT depends on the initial point density and distribution. Since [Balzer et al. 2009a] does not contain a full analysis on these input parameters, we provide one here based on the spectrum analysis as detailed in [Lagae and Dutré 2008]. A reliable spectrum analysis requires multiple runs of inputs with identical parameters and sufficiently large number of samples. This can be a computationally demanding task, and thus could benefit from our fast CCVT method.

Our analysis results are shown in Figure 4. There, we perform CCVT relaxation on input points with different density (16, 64, 256, and 1024 points per site $\frac{m}{n}$) and distribution (square grid, hexagonal grid, and white noise). We pick these four point distributions because they are all common sampling patterns, are easy to implement and can be generated efficiently in large quantities.

As expected, the output quality improves with increasing point density $\frac{m}{n}$. When this ratio approaches ∞ the discrete-space algorithms will behaves like a continuous-space one, at the expense of heavier computation. However, empirically we have found that setting $\frac{m}{n}$ to at least 256 will be enough to produce good output quality. When $\frac{m}{n}$ is too small (e.g. 16), the outputs may exhibit visible artifacts as demonstrated in the spectrum results. Under such low point density, the white noise distribution tends to produce noisier results while square and hexagonal grids can exhibit structural biases. As the point density increases, the white noise result becomes less noisy but the structural biases in square and hexagonal grids tend to remain there. Thus, we recommend using a white noise point distribution with density at least 256 points-per-site for best quality/performance tradeoffs.

Performance Table 1 contrasts the performance of our algorithm with prior methods for varying numbers of points m and sites n. First, one can clearly see that Lloyd relaxation is extremely efficient for all examples. The original [Balzer et al. 2009a] algorithm, on the other hand, while providing better quality, it is much slower. Our complexity-reduced algorithm as presented in Section 4 but without optimizations in Section 5, is over two times faster than the original approach. Once we add the bounding circle and coherent initialization optimizations, both approaches become significantly faster, with ours still edging the original method. The point distance pruning optimization gives an additional improvement except for the left-most configuration with the smallest number of sites×points. The multi-level optimization is only advantageous in situations where there are proportionally few points (i.e., a low $\frac{m}{n}$ ratio).

Finally, when parallelizing the pairwise computation using OpenMP, we can achieve a significant speedup *only* if we perform the two-stage cache-based optimization that uses our heuristic to partition site pairs into clusters.

The incorporation of caching without parallelization does not yield significant improvements, while parallelizing only the full stage actually slows down the algorithm, since the pairs have highly varying running times due to the early pruning when the other optimizations are performed. On the other hand, when caching is performed, each of the cached pairs require a more uniform (and significant) amount of point swapping work, thus significantly taking advantage of parallelization.

Also note that our algorithm scales very well. With a very large number of sites and points (16384 and 256), we achieve speedups of $10\times$ in our parallelized caching code, when compared to the optimized earlier results. These results are orders of magnitude faster than the original [Balzer et al. 2009a] technique.

# sites	1024	4096	4096	16384
# points	256	256	1024	256
Lloyd	1.67	7.66	32.08	37.38
original CCVT	473.72	N/A	N/A	N/A
+ bounding circle				
+ coherent init	4.89	36.70	292.33	342.34
our method (in Section 4)	176.96	N/A	N/A	N/A
+ bounding circle				
+ coherent init	2.18	24.71	181.59	280.37
+ point dist prune	2.20	18.85	169.75	257.78
+ multi-level	1.96	22.68	149.48	340.78
+ OpenMP	7.25	51.55	262.28	510.48
+ 2-stage pair caching	1.53	7.28	44.69	35.75

Table 1: Performance of three relaxation algorithms. We measured the running time (in seconds) of three algorithms (Lloyd, original CCVT in [Balzer et al. 2009a], and our fast CCVT) on different combinations of # sites and points. We also break down the performance impact for each added acceleration technique. N/A indicates entries that are too slow for measurement. All performance numbers are measured on a PC with Windows 7 + Intel Core i7 920 (quad core) @ 2.67GH + 9GB RAM.

7 Concluding Remarks

We present fast CCVT, an acceleration of the original CCVT algorithm in [Balzer et al. 2009a], which is a higher quality yet much slower alternative to Lloyd relaxation [Lloyd 1982]. Given the importance of Lloyd relaxation and its known quality issue, we hope our contribution could help popularize the adoption of CCVT by more applications.

Due to the presence of both sequential and parallel computations in the current CCVT algorithm, we have mainly investigated accelerations for multi-core CPUs. A potential future direction is to explore possibility for a pure GPU or a combined CPU-GPU implementation for further performance improvements.

All accelerations we have investigated in this paper do not change the fundamental algorithmic behavior of the original CCVT method. However, this might not be a necessary requirement, as the ultimate goal is to produce point sets with uniform blue noise distribution. A potential future work is to alter the method more fundamentally by changing the algorithmic behavior while maintaining the output spatial and spectrum quality.

References

BALZER, M., SCHLOMER, T., AND DEUSSEN, O. 2009. Capacity-constrained point distributions: A variant of lloyd's method. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Papers*.

BALZER, M., SCHLOMER, T., AND DEUSSEN, O., 2009. CCVT C++ code. `http://code.google.com/p/ccvt/`.

BRIDSON, R. 2007. Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007: Sketches & Applications*.

CLINE, D., JESCHKE, S., RAZDAN, A., WHITE, K., AND WONKA, P. 2009. Dart throwing on surfaces. In *EGSR '09*.

COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1, 51–72.

DUNBAR, D., AND HUMPHREYS, G. 2006. A spatial data structure for fast poisson-disk sample generation. *ACM Transactions on Graphics 25*, 3, 503–508.

FU, Y., AND ZHOU, B. 2008. Direct sampling on surfaces for high quality remeshing. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, 115–124.

GERSHO, A., AND GRAY, R. M. 1991. *Vector quantization and signal compression*. Kluwer Academic Publishers.

HOFF, III, K. E., KEYSER, J., LIN, M., MANOCHA, D., AND CULVER, T. 1999. Fast computation of generalized voronoi diagrams using graphics hardware. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 277–286.

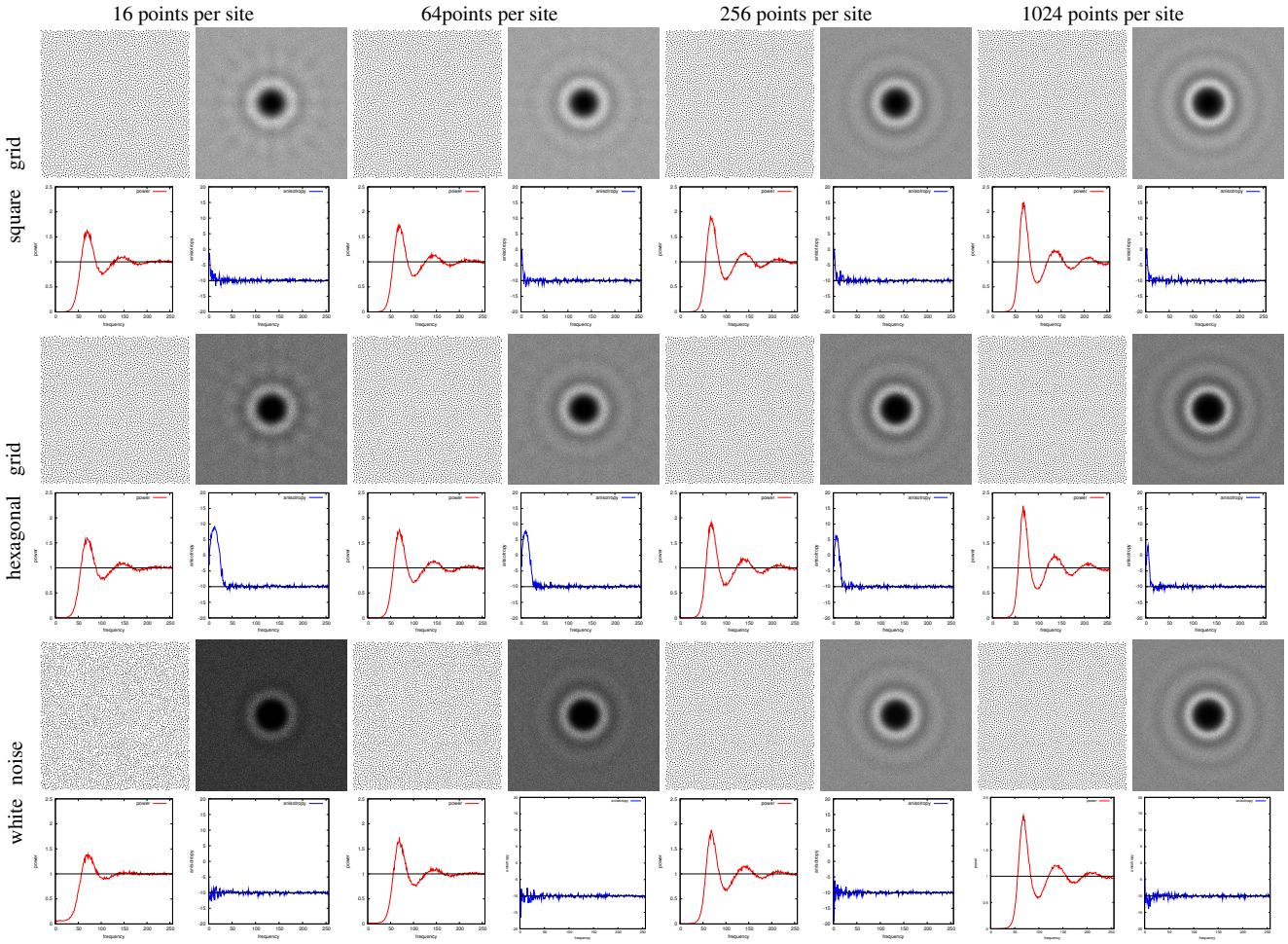

Figure 4: Output site distribution quality with different point density and distribution. The former is layed-out in the horizontal direction while the latter the vertical direction. The number of sites is 4096 for all cases. Shown in each group of 4 images are the point distribution, the power spectrum image, the radial mean, and radial variance/anisotropy [Lagae and Dutré 2008].

JONES, T. R. 2006. Efficient generation of poisson-disk sampling patterns. *journal of graphics tools 11*, 2, 27–36.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive wang tiles for real-time blue noise. *ACM Transactions on Graphics 25*, 3, 509–518.

LAGAE, A., AND DUTRÉ, P. 2008. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum 27*, 1, 114–129.

LLOYD, S. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2, 129–137.

LLOYD, S. 1983. An optimization approach to relaxation labeling algorithms. *Image and Vision Computing 1*, 2.

MCCOOL, M., AND FIUME, E. 1992. Hierarchical poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92*, 94–105.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, ACM, 65–72.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics 23*, 3, 488–495.

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. *ACM Transactions on Graphics 26*, 3, 78.

RONG, G., AND TAN, T.-S. 2006. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, 109–116.

WEI, L.-Y. 2008. Parallel poisson disk sampling. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Papers*, 20:1–9.

WHITE, K., CLINE, D., AND EGBERT, P. 2007. Poisson disk point sets by hierarchical dart throwing. In *Symposium on Interactive Ray Tracing*, 129–132.

YELLOTT, J. I. J. 1983. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science 221*, 382–385.