Weighted Linde–Buzo–Gray Stippling

OLIVER DEUSSEN*, University of Konstanz and SIAT Shenzhen
MARC SPICKER*, University of Konstanz
QIAN ZHENG, University of Konstanz and Shenzhen University

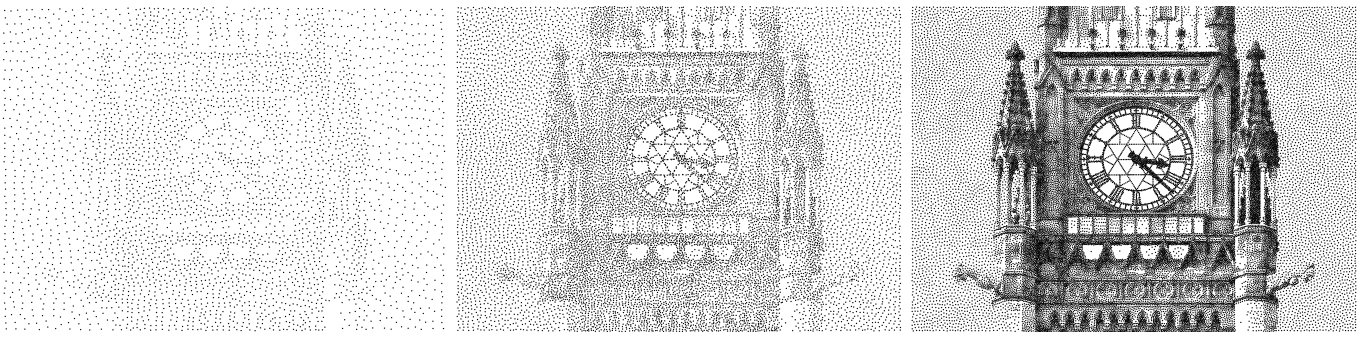


Fig. 1. Starting from a single random point, our dynamic illustration method creates high-quality stipple drawings with a small number of iterations (left to right: 12, 14, and 18 iterations). In the example shown above we end up with 36k points of constant size.

We propose an adaptive version of Lloyd's optimization method that distributes points based on Voronoi diagrams. Our inspiration is the Linde–Buzo–Gray–Algorithm in vector quantization, which dynamically splits Voronoi cells until a desired number of representative vectors is reached. We reformulate this algorithm by splitting and merging Voronoi cells based on their size, greyscale level, or variance of an underlying input image. The proposed method automatically adapts to various constraints and, in contrast to previous work, requires no good initial point distribution or prior knowledge about the final number of points. Compared to weighted Voronoi stippling the convergence rate is much higher and the spectral and spatial properties are superior. Further, because points are created based on local operations, coherent stipple animations can be produced. Our method is also able to produce good quality point sets in other fields, such as remeshing of geometry, based on local geometric features such as curvature.

CCS Concepts: • **Computing methodologies → Computer graphics**; **Non-photorealistic rendering**; **Image processing**;

Additional Key Words and Phrases: Lloyd optimization, Voronoi Diagram, Linde–Buzo–Gray-Algorithm, Stippling, Sampling, Remeshing

ACM Reference Format:
Oliver Deussen, Marc Spicker, and Qian Zheng. 2017. Weighted Linde–Buzo–Gray Stippling. *ACM Trans. Graph.* 36, 6, Article 233 (November 2017), 12 pages.
https://doi.org/10.1145/3130800.3130819

*Joint first authors; email: {oliver.deussen,marc.spicker}@uni.kn

0730-0301/2017/11-ART233 $15.00
https://doi.org/10.1145/3130800.3130819

1 INTRODUCTION

In many computer graphics applications, points are to be distributed randomly, but with almost uniform point-to-point distances. Examples are sampling sets for rendering, halftoning applications in print technology, refinement of meshes in geometry, as well as artistic rendering methods such as stippling. A widely used method to create such point sets is based on an optimization method using Voronoi diagrams, proposed by Stuart Lloyd [1982] and thus later called Lloyd's algorithm. The method takes a given point set, computes its Voronoi diagram and moves each point to the centroid of its Voronoi cell. This is repeated until the points are well distributed. Deussen et al. [2000] and later Secord [2002] used it for producing stipple patterns, and a number of methods for generating sampling patterns have since taken advantage of it (see Section 2).

Despite its various applications, a number of problems are known for Lloyd's algorithm and weighted Voronoi stippling. First, there is no clear mechanism to stop the iteration—it will finally end up in hexagonal subsets (cf. Balzer et al. [2009]) which are distracting for visual applications. Second, a good initial point distribution has to be given. In absence of such a set, the method converges extremely slowly because points have to be moved by local optimization over large distances to be grouped where a higher density is needed.

In 1980, notably two years before Lloyd presented his work, researchers in Vector Quantization proposed a method that dynamically adapts a set of vectors based on properties of their Voronoi Diagram. The Linde–Buzo–Gray (LBG)-Algorithm [1980] represents a given set of training vectors with a small set of exemplars that are computed by repeatedly moving them to the centroids of their Voronoi cells. Based on an initially given vector (the centroid of the whole training set), its Voronoi cell is computed, the vector is moved in the same way as Lloyd's algorithm moves points, then the cell is split and the vectors are moved again. This is repeated until a given number of representative vectors is reached (see Figure 3).

We extend this algorithm to also allow points to be merged, creating a dynamic method that adapts easily and precisely to given density functions, allows varying dot sizes, enables time-coherent stipple animations and can also be applied in remeshing of geometry. As shown in Figure 1, starting from a single initial random point, our method generates new points until a wanted density (here the greyscale values of an input image) is reached. The method splits a Voronoi cell when the point density has to be increased and merges neighboring cells, i.e. deletes corresponding points, when it is too high. A hysteresis function is used to stabilize the iteration. This allows different target distributions to be quickly reached, even from a single initial point, and to locally adapt to given constraints such as wanted size of Voronoi cells or needed point density (see Figure 2).

After reviewing related works, we describe the method and evaluate its precision as well as performance and compare it to state-of-the-art methods in stippling and sampling. To demonstrate its applicability beyond 2D problems, we implemented a dynamic remeshing method for geometry based on the algorithm.

2 RELATED WORK

In 1980, Linde, Buzo, and Gray [1980] developed their algorithm for vector quantization. The algorithm is known for creating optimal codebooks with wanted sizes and has many applications in image compression and data reduction [Joshi et al. 2014]. Two years later, Lloyd published his Voronoi-based optimization algorithm [Lloyd 1982] that can be seen as a variant of *k-means* clustering. Both algorithms are used in many applications within computer science, but it took some time until they were introduced to graphics. Pelleg and Moore [2000] refined k-means in their *x-means* clustering method by using a variant of LBG-clustering that splits regions in dependency to a scoring function that evaluates both the original cell and the two potential new cells. If the score is higher for the new cells, splitting is done.

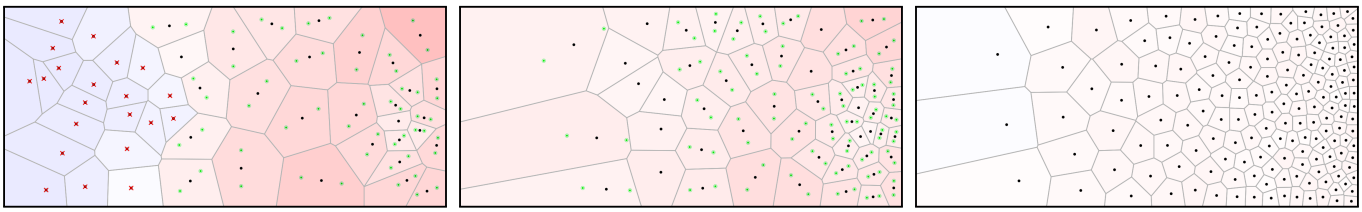
Stippling. Deussen et al. [2000] use Lloyd's method to create point patterns that look similar to human stipple artworks. An initial point set is created using a halftoning method and stipple dots are moved within an interactive system because the original Lloyd method does not account for local variations of the point density. Secord [2002] enhances their scheme by adding weights to Lloyd's algorithm. Given the right initial distribution (he uses rejection sampling) this allows local stipple variations to be recreated faithfully. One year earlier, Hausner [2001] created tilings with different tile sizes by way of an algorithm that implicitly also defines a weighted variant of Lloyd's algorithm by representing tiles with 3D objects of different slope in depth for computing the Voronoi diagram. Hiller et al. [2003] present stippling techniques that are capable of using simple shapes instead of only dots. This is done by creating Voronoi diagrams of these objects and by rotating in addition to just moving them.

Other researchers use different approaches to create stipple drawings. Mould et al. [2007] transform an image into a regular graph and use the gradient magnitude as edge weights. The authors apply Dijkstra's algorithm to place stipples while marching along edges of this graph. Starting from a single point, edge weights are summed up and every time the sum exceeds a given threshold, a new stipple is added. While this produces interesting results, it is a complex solution that is not very efficient for large point sets. Kim et al. [2008] let stipple patterns follow image features. This is accomplished by constraining Lloyd's algorithm via parallel offset lines obtained from a distance field of image features such as outlines. Kim et al. [2009] employ texture synthesis to create stipple textures from artists, which produces images with a more natural look. Martín et al. [2010] use a halftoning method for creating initial stipple sets and later merge nearby stipples to reach a more uniform distribution. They focus on the stipples' shape and texture to produce an artistically looking output. Li and Mould [2011] try to retain image structures instead of just tonal values. This is achieved by processing pixels in a priority order while focusing on extremal values similar to error-diffusion [Floyd and Steinberg 1976].

Sampling. Early works on sampling created Poisson disk point sets where points are distributed with constrained point-to-point distances (cf. McCool and Fiume [1992]). Such sets are good for sampling, but do not resemble stipple drawings created by a human. Centroidal Voronoi Tesselations (CVT), the final outcome of Lloyd's method, were introduced in the same publication, but due to their hexagonal substructures they are not optimal for sampling. Balzer et al. [2009] address this problem: their Capacity Constrained Voronoi Tessellation (CCVT) constrains Voronoi areas in their size during iteration and in this way avoid regular subpatterns. Xu et al. [2011] optimize a capacity constrained Delaunay triangulation instead of Voronoi areas to improve the quality of the point sets. This can be computed much faster.

De Goes et al. [2012] formulate the computation of a CCVT as an optimal transport problem. This enables them to enforce capacity constraints exactly via constrained minimization in the space of power diagrams. Their method produces well suited point sets for sampling and is able to create varying densities. It is, however, not dynamic and needs approximately two minutes for distributing 40k points. More recently, Xin et al. [2016] improve the speed of this method by an order of magnitude by restricting kernel functions to squared Euclidean distances. Chen et al. [2012] propose a variational framework for producing Blue Noise samples in 2D and on geometry with a variant of CCVT. Their combined energy term allows sets to adapt very fast to different density levels. Vanderhaeghe et al. [2007] propose the idea of adding and removing samples in the context of stroke-based rendering. Their point sets retain a good distribution in 2D, follow the motion of an underlying scene, and only very few points need to be added or removed between frames. The LBG algorithm is also used for compression in global illumination, specifically in the case of precomputed radiance transfer techniques [Sloan et al. 2003; Tsai and Shih 2006]. Here the vectors correspond to surface lighting samples and the clusters exploit local coherence of the illumination.

Geometry. Researchers modify vertex sets by using Lloyd's method for purposes such as remeshing of geometry. On surfaces, the so-called Restricted Voronoi Diagram (RVD) has to be computed. While early methods such as from Alliez et al. [2005] and Valette et al. [2008] compute only approximate RVDs for remeshing, more recent methods improve the quality by computing exact RVDs [Ahmed et al. 2016; Yan et al. 2009]. The RVDs are then used to perform Lloyd's



(a) (b) (c)

Fig. 2. LBG-Stippling of a greyscale ramp (see also Figure 6) with color-coded size error in each cell: (a) initial point distribution with Voronoi cells. On the left, only some points are needed, thus cells are too small (blue) and will be merged (points will be deleted); on the right, many points are needed, thus cells are too large (red) and therefore will be split (new positions of points in green); (b) set after two iterations; (c) set after six iterations.

algorithm for vertex smoothing and improving the mesh quality, but none of the above methods use a dynamic version of this method.

Different strategies exist for vertex sets with varying density: Surazhsky et al. [2003] insert vertices at places with high curvature and then use Lloyd's method for moving points to reach a smooth mesh. Luo et al. [2014] create mesh vertices for FEM tests by using Lloyd's method, then perform a Delaunay triangulation, splitting triangles according to the needed density and use the updated set for their simulations. Tournois et al. [2009] interleave Delauney refinement and Optimal Delauney Triangulation, which is a variant of CVT.

Some dynamic methods are used to update various point processes: Klingner et al. [2008] propose a system that schedules removal, insertion, and relaxation steps to generate isotropic distributions of vertices. For meshing implicit surfaces, Meyer [2008] employs dynamic particle systems. Here, an energy function is defined and points are dynamically inserted or deleted to match this energy on a local level. Adaptive fluid simulations adapt point distributions with time coherence [Adams et al. 2007; Ando et al. 2013; Springel 2010]. None of these systems, however, use a unified method based on splitting and merging Voronoi cells.

3 METHOD

Our method combines two main components: weighted Voronoi stippling as an extension of Lloyd's algorithm and a variant of the Linde–Buzo–Gray algorithm. We describe both components below in compact form. For a more detailed description of the LBG-algorithm we refer to Joshi et al. [2014], and weighted Voronoi stippling is described in more detail by Secord [2002].

3.1 Lloyd's Algorithm

For a given point set $\mathbf{P}$, Lloyd's algorithm first computes its Voronoi diagram, which is a tessellation of space, into a set of Voronoi cells $\mathbf{V}$ associated to each point p_i. The Voronoi cell V_i of a point p_i contains all points of the defining space that are closer to p_i than to any other point of $\mathbf{P}$ (usually with respect to Euclidean distance). For a fast computation of Voronoi diagrams see Okabe et al. [1992] and Berg et al. [2008]. For 2D applications, Fortune's GPU algorithm [Fortune 1986] or Jump Flooding [Rong and Tan 2006] can be used. In all our experiments throughout this paper we used the GPU method due to its efficiency. After the Voronoi diagram is computed, all points p_i are moved to the centroids $\mathbf{C}_i$ of their Voronoi cells:

$$C_i = \frac{\int_A x\rho(x)dA}{\int_A \rho(x)dA},$$

where A is the region and $\rho(x)$ is a given density function, e.g. the greyscale values of an input image for weighted Voronoi stippling. After repeating this step for some iterations, the algorithm reaches what is called a Centroidal Voronoi Distribution (CVD), where all points are placed in the centroids of their Voronoi cells. Since such sets often incorporate hexagonal subpatterns, the iteration is usually stopped before convergence.

3.2 Linde–Buzo–Gray (LBG) Algorithm

The two-dimensional version of the Linde–Buzo–Gray Algorithm creates a point set $\mathbf{P}$ and associated Voronoi cells $\mathbf{V}$ based on a set of training points $\mathbf{T}$. A given point set $\mathbf{P}$ is moved using a few iterations of Lloyd's algorithm, this time by moving a point p_i to the centroid of all points $t_i \in \mathbf{T}$ contained in its current Voronoi cell V_i. The algorithm starts with a single point $\mathbf{P}^{(0)}$ and moves it to the centroid of $\mathbf{T}$ by applying Lloyd's method. Until a pre-defined number of points is reached, each point in $\mathbf{P}^{(i)}$ is replaced by two points that are moved apart slightly (splitting). Several steps of Lloyd's method are applied to create a CVD of $\mathbf{P}^{(i)}$ based on $\mathbf{T}$. This is repeated until $\mathbf{P}^{(n)}$ has the required size. Figure 3 shows the first steps of the algorithm[1].

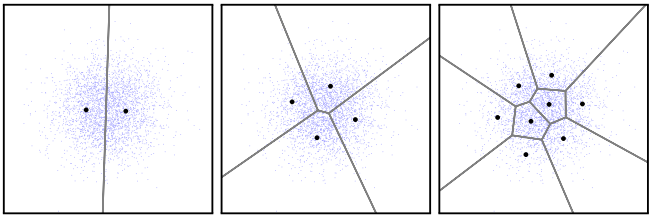
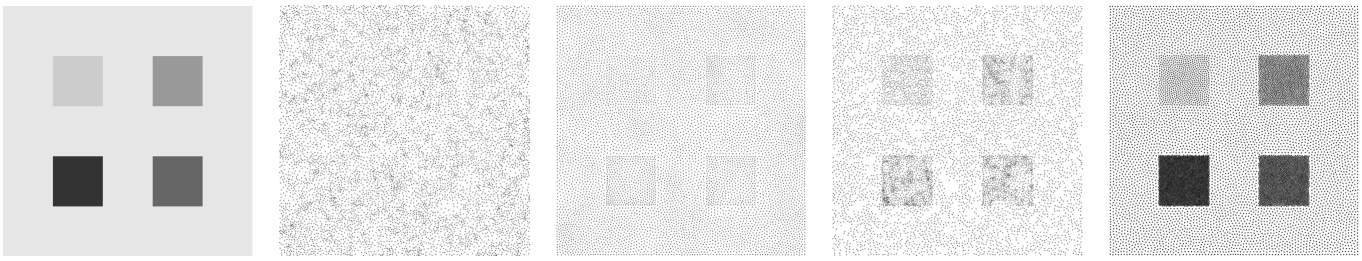


Fig. 3. Points created by Linde–Buzo–Gray algorithm and their respective Voronoi cells on a training data set (blue) after 1, 2, and 3 splitting steps.



(a) (b) (c) (d) (e)

Fig. 4. Approximation speed of LBG-Stippling showing faster convergence towards a good target approximation. Initially 8000 points are randomly distributed: (a) target image; (b),(c) result of weighted Lloyd stippling after 1 and 50 iterations; (d),(e) result of LBG-stippling after 1 and 50 iterations.

3.3 LBG-Stippling

We combine the LBG-Algorithm with weighted Voronoi stippling and this way create a dynamic stippling and point distribution method. We apply the LBG-Algorithm on top of a domain (an input image or a geometric shape) and start our method from a single point or a given point set. Cells are split until a wanted point density is reached, but in contrast to the LBG-algorithm we also merge cells when the density is too high. It has to be noted here that we do not really merge cells due to the corresponding computational overhead, but instead delete points and let subsequent iterations re-insert erroneously removed points. This, however, leads to the situation of Figure 2(a) where all points in the left region are deleted because the merge-criterion is given for each point. An analogous discussion of how many cells should be split and merged in each iteration can be found in Pelleg and Moore [2000]. Here, the authors suggest splitting half the centroids for which the respective indication is given. Our method, which splits and removes every candidate, represents an even more aggressive approach. While this results in a massive overshooting of the target function, the overall system is stable enough to still converge. It is known from such stable systems that overshooting even improves the approximation speed (cf. Cohen et al. [1993]).

For creating a stipple drawing, we assume that each point represents a given amount of ink (determined by its size). Thus, we know how much ink (or greyscale value in the input image) should be represented by its corresponding Voronoi cell because the stipple point stands for the respective image region. Splitting and merging of cells tries to achieve this required size. The handling of white areas is, similar to related methods such as weighted Voronoi stippling, a problem that we solve by assuming a minimal grey level.

Our algorithm has two main parameters: an upper threshold T_u that determines when a cell is to be split, and a lower threshold T_l for merging cells. These parameters describe a hysteresis function that is needed to stabilize the process and to avoid cells that were split in the previous iteration to be merged again. In each iteration we first calculate the Voronoi diagram. For each Voronoi cell we compute the sum of the contained greyscale values. If this value is below T_l, we remove the point. If the value is between T_l and T_u, we keep the point and move it to the centroid of its Voronoi cell; if it is above T_u, the cell is split (see Algorithm 1 and Figure 2). We do not split the cells' generator, but the centroid instead. For each cell the splitting direction is defined along its largest extent, with the displacement vector length depending on the cell size. Currently we estimate the displacement as $\frac{1}{2} \cdot r$, with r being the radius of the inscribing circle of the cell. With this form of displacement we try to distribute new points as evenly as possible, anticipating possible displacements of subsequent iterations. While random insertion of new points is also possible, here it takes more iterations until the same smoothness for the points is achieved. Splitting cells into more than two new cells is possible, but does not provide much better runtime results in practice and creates the problem of locally distributing the newly created points.

3.4 Convergence Rate

Compared to weighted Voronoi stippling, our method adapts much faster to a given density function. Figure 4(a) shows an input image that is to be approximated with an initial random point set (b). If

Algorithm 1 LBG stippling

```
1:  function LBG(T_l, T_u)
2:      stippleList ← initialize stipple list
3:      repeat
4:          vd ← computeVoronoiDiagram(stippleList)
5:          newStippleList ← empty list
6:          for all Voronoi cell vc ∈ vd do
7:              density ← vc.sumDensity
8:              if density < T_l then
9:                  // remove point
10:             else if density < T_u then
11:                 // keep point
12:                 newStippleList.add(vc.centroid)
13:             else
14:                 // split point
15:                 splitPoints ← splitCell(vc)
16:                 newStippleList.add(splitPoints)
17:             end if
18:         end for
19:         stippleList ← newStippleList
20:     until no splitting or merging
21: end function
```

[1] Adapted from http://www.data-compression.com/vq.html.

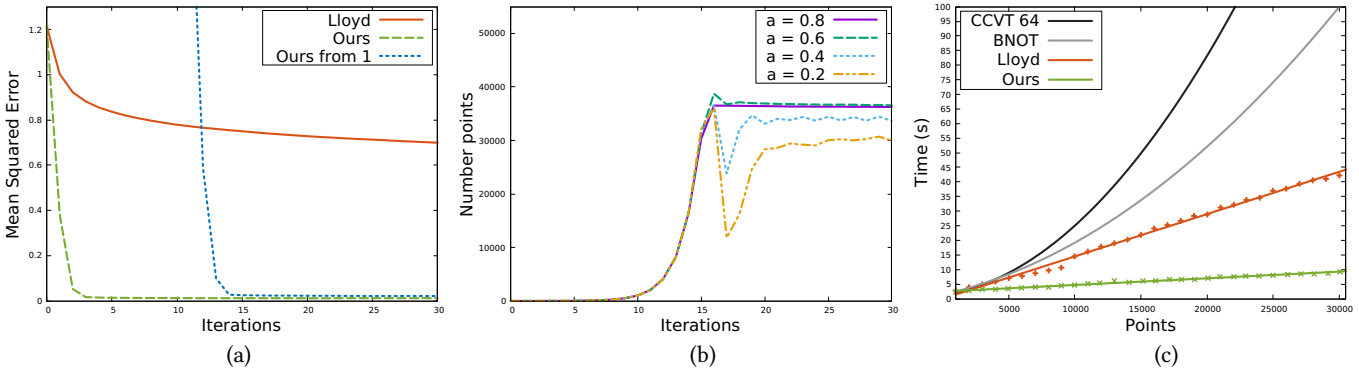


Fig. 5. (a) Error for density approximation in Figure 4: LBG-Stippling converges very fast, even from a single initial point. (b) Convergence rate for different hysteresis thresholds. The number of points after each iteration for the teaser image (Figure 1) are shown on the right, $a = T_u - T_l$. (c) Performance comparison on a uniform density distribution for an increasing number of points.

we apply weighted Voronoi stippling, the input is not represented well even after 50 iterations, whereas LBG-stippling creates a good distribution after only a few iterations. In Figure 5(a) we show the corresponding approximation error. We determined the error as the mean squared difference of the density in each Voronoi cell compared to the area of the corresponding circular point. LBG-stippling is displayed here in two variants: in the first variant the same random initial points are used as for weighted Voronoi stippling; in the second variant only a single initial point is given. Even in this case we reach a good approximation after only 15 iterations.

We also investigated the convergence rate of our algorithm for different settings; results for the teaser images are shown in Figure 5(b). The rate primarily depends on the hysteresis parameter $a = T_u - T_l$, which determines the range of 'acceptable' values for each Voronoi cell. If a is too small, the point set starts to oscillate and the number of iterations until convergence increases, or in the worst case the algorithm does not reach convergence at all. Dense areas have the tendency to undergo more changes because, due to our pixel-based approach, these areas are often under-sampled compared to areas with larger Voronoi cells.

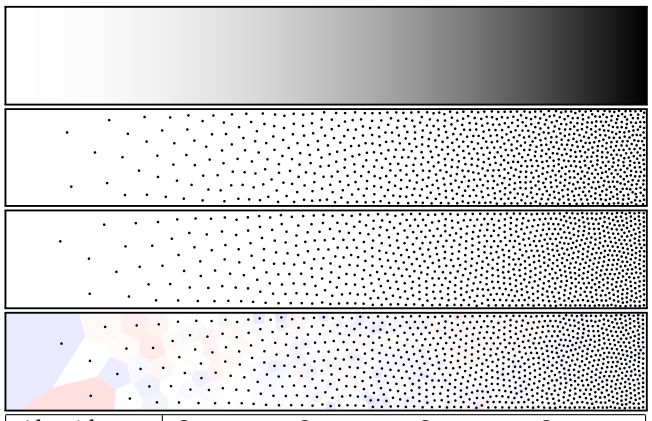
Choosing a larger a will result in fewer oscillations and fewer needed iterations, but will also decrease the quality of the resulting point set, especially in dark areas. Here points have the tendency to flow apart, which results in reduced contrast. The iteration can thus be optimized by changing a over time. A small hysteresis at the beginning might be used to adapt quickly to the wanted greyscale level. Later, the hysteresis interval is increased to stabilize the point set. Throughout the paper, we increased the hysteresis parameter linearly over time for all our results and the accompanying demo program, excluding the measurements for Figure 4. We will discuss the choice of hysteresis parameters and their effect on the approximation quality in the next subsection. The initial number of points does not play a major role because our splitting approach can create or delete points at an exponential rate. Choosing too many initial points is also not an issue because cell merging removes such points efficiently. In Figure 5(c) we compare the performance of our algorithm to that of Lloyd's method and two state-of-the-art blue noise generation methods—CCVT [Balzer et al. 2009] and BNOT [de Goes et al. 2012]—for a constant density function with an increasing number of points. The Lloyd relaxation stop criterion was set to $\delta = 0.75$ of overall movement with point coordinates $p_i \in [0,1]^2$, for our algorithm we set $a = 0.6$ and used 100 random starting points. For both our method and Lloyd's method, we used the same underlying GPU framework. Our algorithm is clearly faster than Lloyd relaxation due to the fact that we need far fewer iterations until convergence; however, a single iteration is slightly slower due to the splitting and merging overhead.

3.5 Approximation Quality

To measure the approximation quality of our algorithm, we try to reproduce different tonal values for a quadratic intensity ramp. Ideally, the relative amount of points in each quarter should represent the relative amount of ink (greyscale value) in the ramp, cf. Figure 6. A method such as BNOT allows such tonal values to be represented faithfully. Because we are not able to control the number of points in our sets directly, we pre-defined a point size resulting in a point set that approximates the given number reasonably well. In this case we ended up with 1000 points on average, which is the same number used by de Goes et al [2012].

Our system uses a hysteresis function to stabilize the overall process, thus it is quite likely that a bias is introduced. During our iteration it is not guaranteed that the cell sizes will automatically be in the middle between our thresholds T_u for splitting and T_l for merging, but rather they could be anywhere in between. For a symmetric hysteresis $T_u = 1 + a/2$, $T_l = 1 - a/2$ we measured the ratio between existing and desired sizes of the Voronoi cells and computed the average over all cells. Without bias this value should be 1; in our experiments we measured values in a range of $[1.02 - 1.06]$, depending on the number of points and initial conditions. Thus, LBG-stippling seems to have a tendency to underrepresent a wanted greyscale level.

Nevertheless, our method still approximates a given greyscale image faithfully. We compare the intensity approximation against BNOT in Figure 6. For obtaining our values, we averaged 100 stipple approximations. As in de Goes et al. [2012], we computed the relative point counts for the four quarters of the input ramp. While we



Algorithm	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Reference	1.55%	10.92%	29.69%	57.83%
BNOT [2012]	1.50%	10.90%	29.60%	57.90%
Ours	1.49%	10.76%	29.71%	58.03%

Fig. 6. Representation of a greyscale ramp: de Goes et al. [2012] set of 1000 points (second row), LBG stippling with close to 1000 points (third row) and a visualization of the relative approximation error (bottom, blue indicates a cell that is too small, red too large). The table shows relative ink density in each quarter of the input. Points are shrunk for better visual inspection.

approximate each quarter very well, it has to be noted that for very light areas, the Voronoi cells of the points are large and change drastically when points are inserted or deleted. Thus for a small point set as for the ramp, we have to use a relatively large tolerance a to reach a stable set.

In Figure 7, we compare spatial and spectral properties of the point sets from our algorithm ($a = 0.2$) to those produced by Lloyd's method, CCVT [Balzer et al. 2009] (1024 samples), Fattal [2011], and BNOT. From top to bottom: example point distribution, regularity visualizations by highlighting gaps and number of neighbors, mean periodogram, radial power spectra, and anisotropy. There are far fewer regularities compared to Lloyd's method, shown by the gap and neighbor visualizations. The spatial properties are much closer to that of CCVT and BNOT. This is also true for the spectral properties of our method: we get a flat spectrum in low and high frequencies, maintain high peaks at the characteristic frequencies, but have a slightly higher anisotropy at low frequencies. On the positive side we produce our point sets within seconds. The improved properties of our algorithm compared to Lloyd's method are mainly due to our hysteresis: since our stopping criterion is based on a range between T_l and T_u, our algorithm does not always converge to the global minimum, which would cause hexagonal substructures. Other algorithms, such as BNOT, use a post-processing step in order to remove points from such global minima and avoid regularities.

4 RESULTS

After describing the main properties of LBG-stippling, we now highlight some aspects of the method that distinguishes it from previous solutions. For all results we used pixel-based Voronoi diagrams implemented by a shader-version of Fortune's method [1986]. While this is extremely fast, it sometimes suffers in accuracy, especially for very high point densities with corresponding small Voronoi areas. To reduce this problem we use an offscreen buffer with a supersampling factor of at least two to three, which we found to be sufficient for common image sizes and numbers of points. We scale the input density map on which the integration is performed by this factor using linear interpolation.

All examples throughout this paper have been created on a computer equipped with an Intel Core i7–4790K CPU with 4.0 GHz and a Nvidia GTX 980 Ti GPU. Most of the computation time is needed for the calculation of the Voronoi diagram; measuring and updating the cells is very fast because all needed data is already produced during the Voronoi diagram calculation. For 15k points we achieve about 8 frames per second with an image size of 1200×1000 pixels and a supersampling factor of 2. Our iterations are slightly slower than Lloyd's method due to the splitting and merging, but far fewer iterations are needed to achieve a good stippled representation. Figure 4 shows an example of LBG stippling speed and Figure 5 shows convergence rates for different settings.

The algorithm of de Goes et al. [2012] (BNOT) and its optimization [Xin et al. 2016] with a speedup of factor 10 represent state-of-the-art methods for generating blue noise point distributions. In Figure 8 we compare our results to those of Xin et al. [2016]. From left to right we show the input image, their stippled result, and our result, both with 10k points[2]. While their method requires 146 seconds of computation time, ours only needs 7.7 seconds. Nevertheless, small structures are better preserved by LBG-stippling as indicated by the arrows.

Typically, weighted Voronoi stippling uses error diffusion to produce an initial set of points that are then optimized using Lloyd relaxation. In Figure 9(a) the initial set for an input image is given. The points after 10 relaxation steps are shown in the center (0.033s), and our result on the right (0.04s). All results contain approx. 2.5k points. The final results are visually similar, but our algorithm does not show the typical regularities present for the Lloyd relaxation.

4.1 Globally Varying Point Sizes

Our algorithm is capable of creating different abstraction levels by changing the required point size. Since both merge and split operations work based on comparing point sizes to Voronoi cell densities, we only have to change the required point size and the algorithm automatically determines the necessary number of points for representing the input image (see Figure 10). We simply have to multiply the split and merge thresholds T_u and T_l with the area of the corresponding point. As already mentioned, it is helpful during iterations to increase the hysteresis over time because otherwise the algorithm might converge slowly. Typically we go from an initial hysteresis of $[0.2 - 0.6]$ linearly up to $[0.6 - 1.0]$ over the course of 50 iterations. When parameters are changed during the iterations, we reset the hysteresis to its initial value because otherwise the algorithm might stick locally to undesired densities that do not optimally represent the greyscale level.

[2]Please note that we cannot precisely control the number of stipples, thus our shown solution contains 9,996 points.

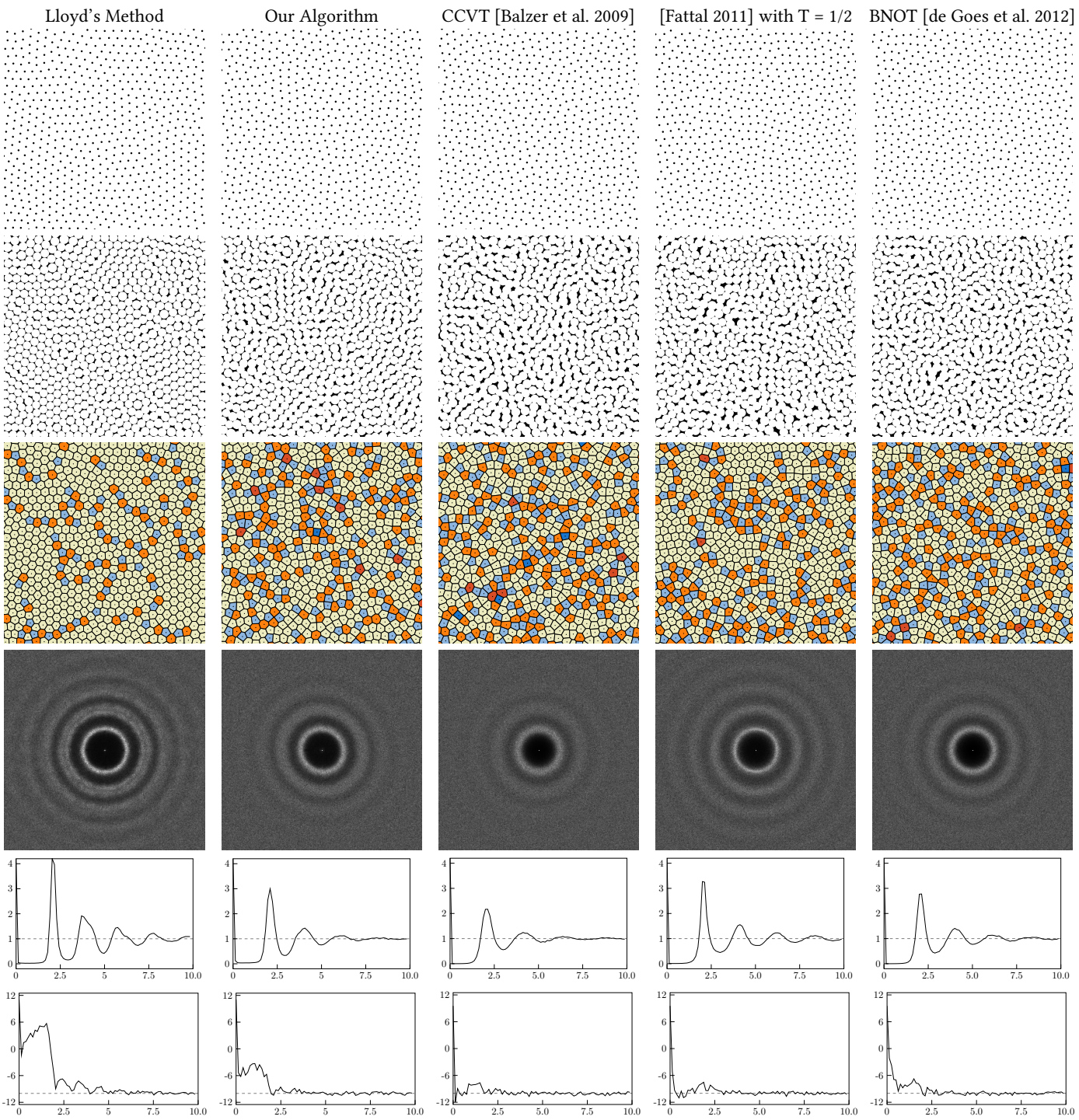


Fig. 7. Averaged point set properties from different algorithms for 10 realizations each with borders excluded from visualizations. Top row: example point distributions. Second row: white disks centered on points with radius according to mean neighbor distance to highlight gaps. Third row: color-coded visualization of the numbers of neighbors for each Voronoi cell. Fourth row: mean periodograms. Fifth and sixth row: radial power spectra and anisotropy (in dB over frequency).

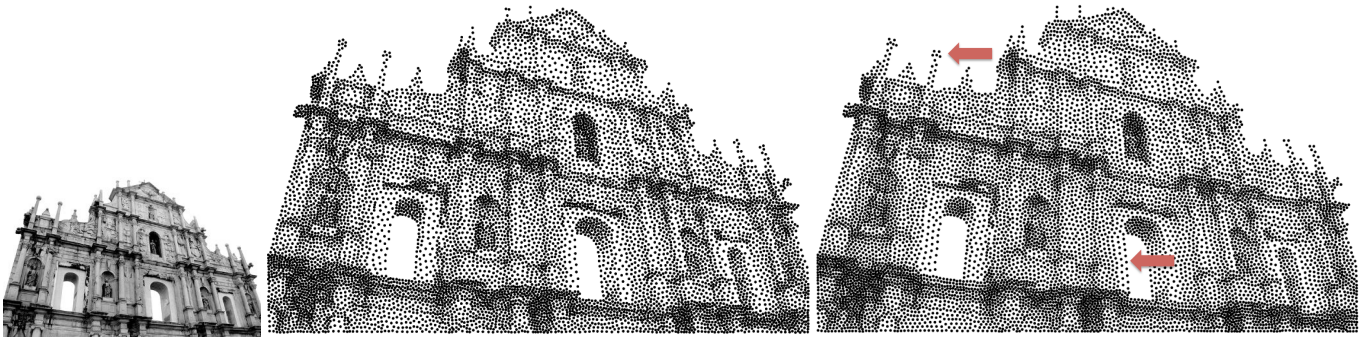


Fig. 8. Comparison of our technique to Xin et al. [2016]. From left to right: input image, their result, and our result (10k points each). While their result required 146 seconds of computation time, ours only needed 7.7 seconds. Our method represents details of the facade better, as indicated by the arrows.

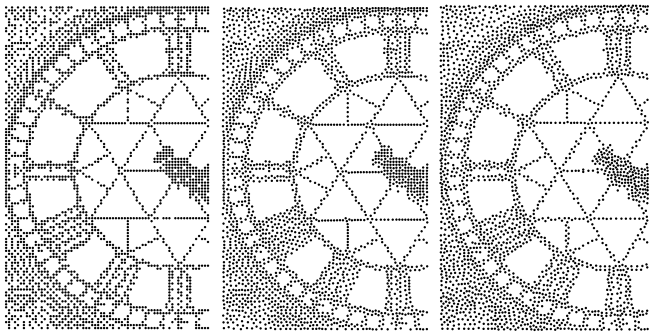


Fig. 9. (a) Point set created by Floyd Steinberg, (b) result of (a) followed by ten steps of conventional weighted Lloyd relaxation, (c) our result. Each result contains approx. 2.5k points, scaled down for better visual inspection.

The exponential point creation rate of LBG-stippling allows the method to double or halve the number of points and reach a new stable set within a few iterations (typically less than 10). Due to the implicit form of control, we are not able to represent an image instantly with a fixed number of points. To achieve this, the point size has to be adapted iteratively until the wanted number of points is reached. This, however, does not have to be seen only as a drawback: our method automatically determines the necessary number of points for a given dot size, which is not possible for comparable methods.

4.2 Locally Varying Point Sizes

In contrast to other methods, LBG-stippling can produce locally varying point sizes. We can determine the size of a point corresponding to a cell according to some of its properties and calculate the split and merge thresholds accordingly. The size of newly generated points can, for example, be related to their position or to attributes of the input image, such as the variance within a Voronoi cell or the local greyscale level. The idea behind using the variance is to represent small features (areas with high variance) by smaller points. In our experiments we use a point size s_i determined by:

$$s_i = lerp\left(\sqrt{\lambda / Var(V_i)}, P_{min}, P_{max}\right)$$

with $\lambda \in [0.05, 0.8]$ depending on the wanted style of the resulting stipple drawing and P_{min}, P_{max} being the minimal and maximal point size. Figure 11(a) shows an outdoor scene with $\lambda = 0.05, P_{min} = 2, P_{max} = 5$. For the face in (b) and other smooth objects $\lambda = 0.3$ and $P_{min} = 3, P_{max} = 9$ are appropriate settings.

We can also relate the point size proportionally to the greyscale level, allowing us to save points for representing dark areas. Stipple artists do the same and also use point variation to create more lively illustrations. For the relief in Figure 12, points in dark areas are three times larger than those in light areas. In this way, we reduce the overall number points from 60k for a constant size to 39k and at the same time represent dark areas with more contrast.

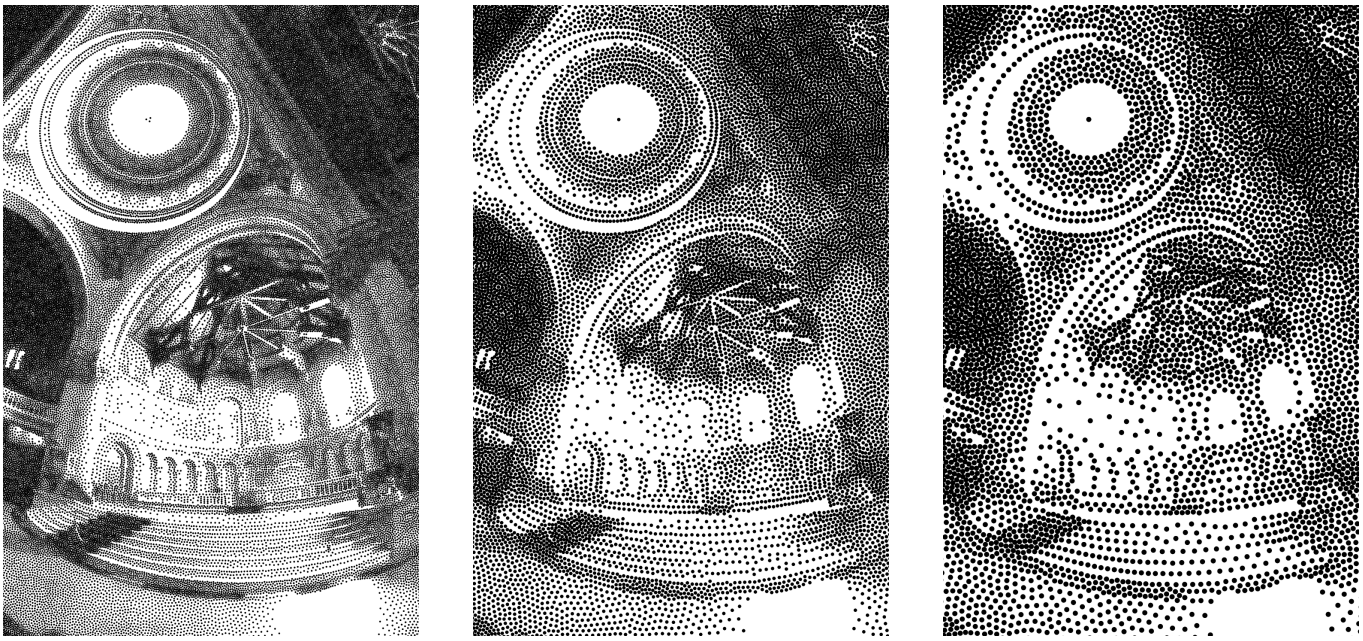
Finally, it is also possible to relate point sizes to geometric constraints. The female model in Figure 13(a) is rendered with radially increasing point sizes from her eye, which allows us to visually emphasize this area. In (b) close regions of the duck beak are rendered using larger points, which allows us the highlight the depth structure of the image.

4.3 Coherent Animations

Our algorithm works well for producing coherent animations. The accompanying video shows an animation in which we zoom into a Mandelbrot set. The first frame of the animation has been created with LBG-stippling and all successive frames use the previous points as the input point set. Since we know the optical flow in this case, we are able to avoid the shower door effect by moving the points of the previous frame with the flow and creating new points similar to Kass et al. [2011] only at places where the greyscale value changed. By doing so, only a few local changes are needed and the number of iterations can be kept small (typically 3-5).

5 APPLICATION: REMESHING

Our algorithm can be used beyond stippling and is not limited to the two-dimensional domain. As a short example we show that the method can be extended to 3D surfaces in order to produce high-quality meshes. Instead of 2D images, we now perform splitting and



(a) (b) (c)

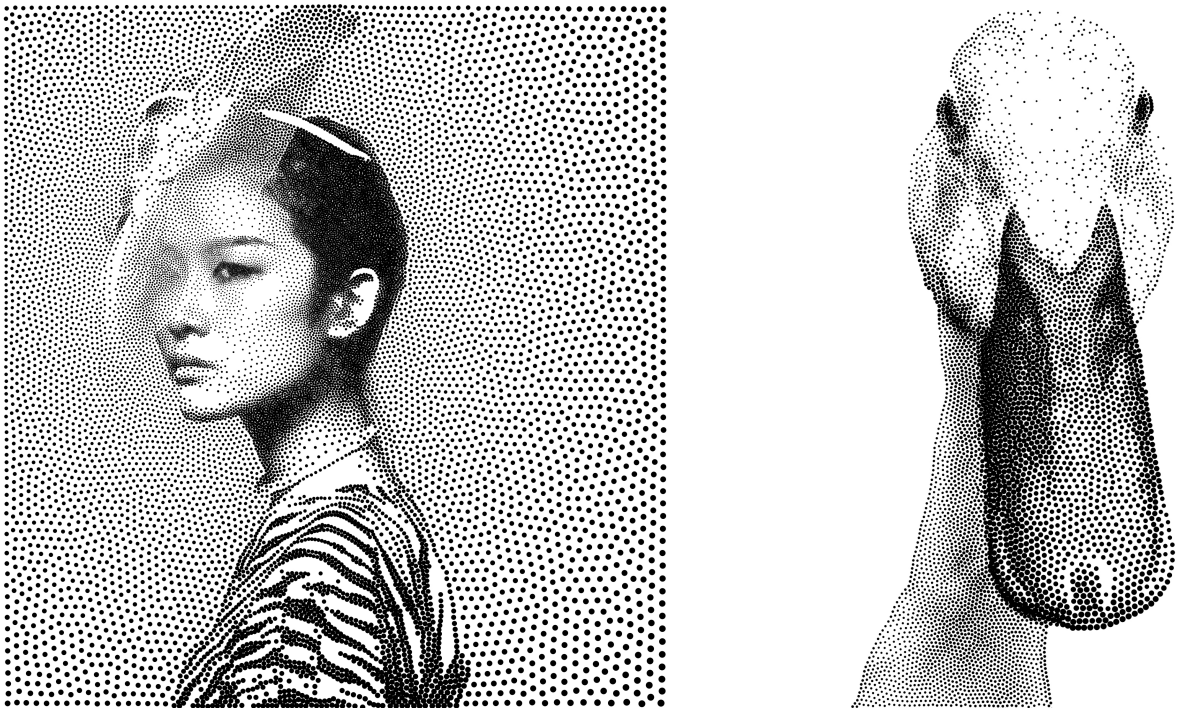
Fig. 10. Different point diameters used to represent an input image, creating different numbers of points and levels of abstraction: (a) 56k points (size 2), (b) 14k points (size 4), and (c) 6.3k points (size 6). Input image courtesy of P. Debevec.

(a) (b)

Fig. 11. Two stipple drawings with varying point sizes based on variance in the input image: (a) variable point size with $\lambda = 0.05$, $P_{min} = 2$, $P_{max} = 5$; (b) variable point size with $\lambda = 0.3$, $P_{min} = 3$, $P_{max} = 9$.

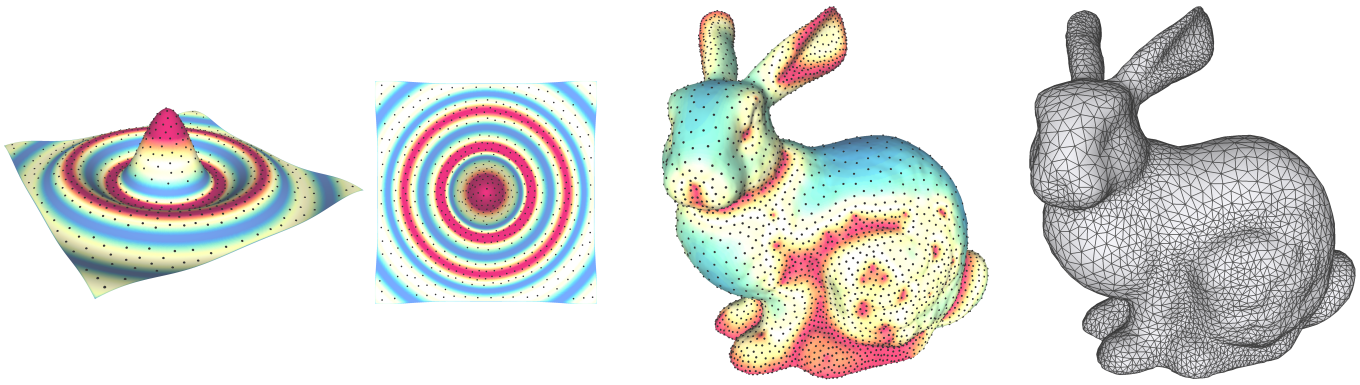
(a) (b)

Fig. 12. Point sizes in relation to the local greyscale level: (a) Egyptian Relief (Husband and Wife, Walters Art Museum), represented with 39k points from size 2 to 6; (b) upper image showing cutout from (a), lower image showing cutout from a relief with 60k points of constant size 3.



(a) (b)

Fig. 13. Point sizes in relation to other constraints: (a) radial increase from the eye of the model (original image by Bùi Linh Ngân, Flickr). In (b) the beak of the duck is rendered with points enlarged depending on their distance to the viewer.



(a) (b) (c) (d)

Fig. 14. Adaptive sampling of a height function and a geometric model with density defined by the color coded mean curvature: (a),(b) about 600 adaptively sampled points; (c) 5k adaptive points on the Stanford bunny; (d) remeshed result of (c).

merging operations on a surface that is represented by a triangular mesh. Because our method ultimately distributes vertices similar to Lloyd's algorithm—at least if cells are no longer split or merged—we are not able to surpass the quality produced by the most recent works on remeshing (cf. Ahmed et al. [2016]). Nevertheless, our method allows quick adaption to local geometric features such as curvature.

To compute the Voronoi diagram on a mesh, we adopt the method by Liu et al. [2011], which is based on exact geodesic distances instead of Euclidean distances. To approximate the area and density integral of each Voronoi cell, we use the discrete area and density of the vertices of the underlying mesh. To approximate the centroid of each cell, we use the method proposed by Wang et al. [2015] and iteratively move a base point to the centroid. In each iteration, we first project the cell to the planar tangent region of the current base point using geodesic polar coordinates [Melvær and Reimers 2012], and then update the point to the centroid of the projection. Once we found the centroid, we project the cell to the planar region for new splitting points.

We define the density function on the surface by local feature sizes or simply by the curvature. In the latter case, we use the mean squared curvature to compute the needed point density. Figures 14(a),(b) show adaptive sampling of a height function based on the mean curvature. Starting with 16 random initial points, our method converges after 20 iterations with about 600 points. Figures 14(c),(d) show an adaptive sampling of a bunny model with 5K points, and a remeshing result generated from these points by applying a mesh extraction algorithm [Yan et al. 2009]. We list the statistical properties of some input meshes in Table 1 according to Yan et al. [2014]: the values indicate that our method can compete with state-of-the-art methods (see Table 3 in their paper).

Table 1. Quality metrics for our adaptive remeshing algorithm based on dynamic Voronoi relaxation according to Yan et al. [2014].

Model	|X|	Q_{min}	Q_{avg}	θ_{min}	$\tilde{\theta}_{min}$	θ_{max}	$\theta_{<30^o}$	V_{567}	d_H
Bunny	4746	0.390	0.799	17.72	43.93	129.24	0.766	95.1	0.47
Kitten	4064	0.369	0.803	22.67	44.31	131.74	0.541	95.7	0.28
Homer	3512	0.394	0.799	21.14	44.04	128.60	0.897	95.0	0.51

6 CONCLUSION AND FUTURE WORK

We present a dynamic version of weighted Voronoi stippling that iteratively creates and removes points based on local properties of the Voronoi cells. The method adapts rapidly to any given input function, is able to produce stipple drawings with varying point sizes, can produce stable stipple animations, and can also be applied in remeshing of geometry. The spatial and spectral properties of our generated point sets are superior to those of classic Lloyd relaxation and comparable to those of CCVT and BNOT, despite the discrete GPU implementation of our algorithm. Due to the discretization we might introduce accuracy problems, especially in point-dense areas, in exchange for computation speed. Aside from the hysteresis parameter a and the optional variance mapping λ, no other parameters are required for the core method. Throughout this paper, however, we introduced some optional parameters for more control over the created output and to allow more artistic freedom.

Future works will include more advanced cell splitting criteria, local error diffusion between Voronoi cells and adaptive resolution models for density integration over Voronoi cells of different size. We will also further investigate whether it is possible to predict the convergence rate based on the hysteresis parameter. Due to the efficiency of the proposed method, we also want to create point distributions in 3D and higher dimensions, as well as adapting the method to other objects in addition to points.

ACKNOWLEDGEMENTS

The authors would like to thank the German Research Foundation (DFG) for financial support within project A04 of SFB/Transregio 161. We also want to thank the Guangdong Science and Technology Program (2015A030312015), Leading Talents of Guangdong Program (00201509), Shenzhen Innovation Program (JCYJ20151015151249564), and Natural Science Foundation of SZU (827-000196).

REFERENCES

B. Adams, M. Pauly, R. Keiser, and L. Guibas. 2007. Adaptively Sampled Particle Fluids. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. ACM, New York, NY, USA, Article 48. https://doi.org/10.1145/1275808.1276437

A. G. M. Ahmed, J. Guo, D. M. Yan, J. Y. Franceschi, X. Zhang, and O. Deussen. 2016. A Simple Push-Pull Algorithm for Blue-Noise Sampling. *IEEE Transactions on Visualization and Computer Graphics* PP, 99 (2016), 1–1. https://doi.org/10.1109/TVCG.2016.2641963

P. Alliez, E. de Verdière, O. Devillers, and M. Isenburg. 2005. Centroidal Voronoi diagrams for isotropic surface remeshing. *Graphical Models* 67, 3 (2005), 204 – 231. https://doi.org/10.1016/j.gmod.2004.06.007

R. Ando, N. Thürey, and C. Wojtan. 2013. Highly Adaptive Liquid Simulations on Tetrahedral Meshes. *ACM Trans. Graph.* 32, 4, Article 103 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2461982

M. Balzer, T. Schlömer, and O. Deussen. 2009. Capacity-constrained Point Distributions: A Variant of Lloyd's Method. *ACM Trans. Graph.* 28, 3, Article 86 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531392

M. Berg, O. Cheong, M. Kreveld, and M. Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag TELOS.

Z. Chen, Z. Yuan, Y. K. Choi, L. Liu, and W. Wang. 2012. Variational Blue Noise Sampling. *IEEE Transactions on Visualization and Computer Graphics* 18, 10 (Oct 2012), 1784–1796. https://doi.org/10.1109/TVCG.2012.94

M. Cohen, J. Wallace, and P. Hanrahan. 1993. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Inc., San Diego, CA, USA.

F. de Goes, K. Breeden, V. Ostromoukhov, and M. Desbrun. 2012. Blue Noise Through Optimal Transport. *ACM Trans. Graph.* 31, 6, Article 171 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366190

O. Deussen, S. Hiller, C. Van Overveld, and T. Strothotte. 2000. Floating Points: A Method for Computing Stipple Drawings. *Computer Graphics Forum* 19, 3 (2000), 41–50. https://doi.org/10.1111/1467-8659.00396

R. Fattal. 2011. Blue-noise Point Sampling Using Kernel Density Model. *ACM Trans. Graph.* 30, 4, Article 48 (July 2011), 12 pages. https://doi.org/10.1145/2010324.1964943

R. W. Floyd and L. Steinberg. 1976. An adaptive algorithm for spatial grey scale. In *Proceedings of the Society of Information Display*. 75–77.

S Fortune. 1986. A Sweepline Algorithm for Voronoi Diagrams. In *Proceedings of the Second Annual Symposium on Computational Geometry (SCG '86)*. ACM, New York, NY, USA, 313–322. https://doi.org/10.1145/10515.10549

A. Hausner. 2001. Simulating Decorative Mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 573–580. https://doi.org/10.1145/383259.383327

S. Hiller, H. Hellwig, and O. Deussen. 2003. Beyond Stippling - Methods for Distributing Objects on the Plane. *Computer Graphics Forum* 22, 3 (2003), 515–522. https://doi.org/10.1111/1467-8659.00699

M. A. Joshi, M. S. Raval, Y. H. Dandawate, K. R. Joshi, and S. P. Metkar. 2014. *Image and Video Compression: Fundamentals, Techniques, and Applications*. Chapman and Hall/CRC. https://doi.org/10.1201/b17738

M. Kass and D. Pesare. 2011. Coherent Noise for Non-photorealistic Rendering. *ACM Trans. Graph.* 30, 4, Article 30 (July 2011), 6 pages. https://doi.org/10.1145/2010324.1964925

D. Kim, M. Son, Y. Lee, H. Kang, and S. Lee. 2008. Feature-guided Image Stippling. *Computer Graphics Forum* 27, 4 (2008), 1209–1216. https://doi.org/10.1111/j.1467-8659.2008.01259.x

S. Kim, R. Maciejewski, T. Isenberg, W. Andrews, W. Chen, M. Sousa, and D. Ebert. 2009. Stippling by Example. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering (NPAR '09)*. ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/1572614.1572622

B. Klingner and J. Shewchuk. 2008. Aggressive tetrahedral mesh improvement. In *Proceedings of the 16th international meshing roundtable*. Springer, 3–23. https://doi.org/10.1007/978-3-540-75103-8_1

H. Li and D. Mould. 2011. Structure-preserving Stippling by Priority-based Error Diffusion. In *Proceedings of Graphics Interface 2011 (GI '11)*. Canadian Human-Computer Communications Society, 127–134. http://dl.acm.org/citation.cfm?id=1992917.1992938

Y. Linde, A. Buzo, and R. Gray. 1980. An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications* 28, 1 (Jan 1980), 84–95. https://doi.org/10.1109/TCOM.1980.1094577

Y. J. Liu, Z. Chen, and K. Tang. 2011. Construction of Iso-Contours, Bisectors, and Voronoi Diagrams on Triangulated Surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 8 (Aug 2011), 1502–1517. https://doi.org/10.1109/TPAMI.2010.221

S. Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theor.* 28, 2 (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489

L. Luo, Z. Jiang, H. Lu, D. Wei, K. Linghu, X Zhao, and D. Wu. 2014. Optimisation of Size-controllable Centroidal Voronoi Tessellation for FEM Simulation of Micro Forming Processes. *Procedia Engineering* 81 (2014), 2409 – 2414. https://doi.org/10.1016/j.proeng.2014.10.342

D. Martín, G. Arroyo, M. Luzón, and T. Isenberg. 2010. Example-based Stippling Using a Scale-dependent Grayscale Process. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering (NPAR '10)*. ACM, New York, NY, USA, 51–61. https://doi.org/10.1145/1809939.1809946

M. McCool and E. Fiume. 1992. Hierarchical Poisson Disk Sampling Distributions. In *Proceedings of the Conference on Graphics Interface '92*. Morgan Kaufmann Publishers Inc., 94–105.

E. Melvær and M. Reimers. 2012. Geodesic Polar Coordinates on Polygonal Meshes. *Computer Graphics Forum* 31, 8 (2012), 2423–2435. https://doi.org/10.1111/j.1467-8659.2012.03187.x

M. Meyer. 2008. *Dynamic Particles for Adaptive Sampling of Implicit Surfaces*. Ph.D. Dissertation. University of Utah.

D. Mould. 2007. Stipple Placement Using Distance in a Weighted Graph. In *Proceedings of the Third Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging (Computational Aesthetics'07)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 45–52. https://doi.org/10.2312/COMPAESTH/COMPAESTH07/045-052

A. Okabe, B. Boots, and K. Sugihara. 1992. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley & Sons.

D. Pelleg and A. Moore. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the 17th International Conference on Machine Learning (ICML '00)*. Morgan Kaufmann Publishers Inc., 727–734.

G. Rong and T. Tan. 2006. Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games (I3D '06)*. ACM, New York, NY, USA, 109–116. https://doi.org/10.1145/1111411.1111431

A. Secord. 2002. Weighted Voronoi Stippling. In *Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering (NPAR '02)*. ACM, New York, NY, USA, 37–43. https://doi.org/10.1145/508530.508537

P. Sloan, J. Hall, J. Hart, and J. Snyder. 2003. Clustered Principal Components for Precomputed Radiance Transfer. *ACM Trans. Graph.* 22, 3 (July 2003), 382–391. https://doi.org/10.1145/882262.882281

V. Springel. 2010. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. *Monthly Notices of the Royal Astronomical Society* 401, 2 (2010), 791. https://doi.org/10.1111/j.1365-2966.2009.15715.x

V. Surazhsky, P. Alliez, and G. Gotsman. 2003. *Isotropic Remeshing of Surfaces: a Local Parameterization Approach*. Research Report RR-4967. INRIA. https://hal.inria.fr/inria-00071612

J. Tournois, C. Wormser, P. Alliez, and M. Desbrun. 2009. Interleaving Delaunay Refinement and Optimization for Practical Isotropic Tetrahedron Mesh Generation. In *ACM SIGGRAPH 2009 Papers (SIGGRAPH '09)*. ACM, New York, NY, USA, Article 75, 9 pages. http://doi.acm.org/10.1145/1531326.1531381

T. Tsai and Z. Shih. 2006. All-frequency Precomputed Radiance Transfer Using Spherical Radial Basis Functions and Clustered Tensor Approximation. *ACM Trans. Graph.* 25, 3 (July 2006), 967–976. https://doi.org/10.1145/1141911.1141981

S. Valette, J. M. Chassery, and R. Prost. 2008. Generic Remeshing of 3D Triangular Meshes with Metric-Dependent Discrete Voronoi Diagrams. *IEEE Transactions on Visualization and Computer Graphics* 14, 2 (March 2008), 369–381. https://doi.org/10.1109/TVCG.2007.70430

D. Vanderhaeghe, P. Barla, J. Thollot, and F. Sillion. 2007. Dynamic Point Distribution for Stroke-based Rendering. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques (EGSR'07)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 139–146. https://doi.org/10.2312/EGWR/EGSR07/139-146

X. Wang, X. Ying, Y. Liu, S. Xin, W. Wang, X. Gu, W. Mueller-Wittig, and Y. He. 2015. Intrinsic computation of centroidal Voronoi tessellation (CVT) on meshes. *Computer-Aided Design* 58 (2015), 51 – 61. https://doi.org/10.1016/j.cad.2014.08.023 Solid and Physical Modeling 2014.

S. Xin, B. Lévy, Z. Chen, L. Chu, Y. Yu, C. Tu, and W. Wang. 2016. Centroidal Power Diagrams with Capacity Constraints: Computation, Applications, and Extension. *ACM Trans. Graph.* 35, 6, Article 244 (Nov. 2016), 12 pages. https://doi.org/10.1145/2980179.2982428

Y. Xu, L. Liu, C. Gotsman, and S. Gortler. 2011. Capacity-Constrained Delaunay Triangulation for point distributions. *Computers & Graphics* 35, 3 (2011), 510 – 516. https://doi.org/10.1016/j.cag.2011.03.031 Shape Modeling International (SMI) Conference 2011.

D. Yan, J. Guo, X. Jia, X. Zhang, and P. Wonka. 2014. Blue-Noise Remeshing with Farthest Point Optimization. *Computer Graphics Forum* 33, 5 (2014), 167–176. https://doi.org/10.1111/cgf.12442

D. Yan, B. Lévy, Y. Liu, F. Sun, and W. Wang. 2009. Isotropic Remeshing with Fast and Exact Computation of Restricted Voronoi Diagram. *Computer Graphics Forum* 28, 5 (2009), 1445–1454. https://doi.org/10.1111/j.1467-8659.2009.01521.x