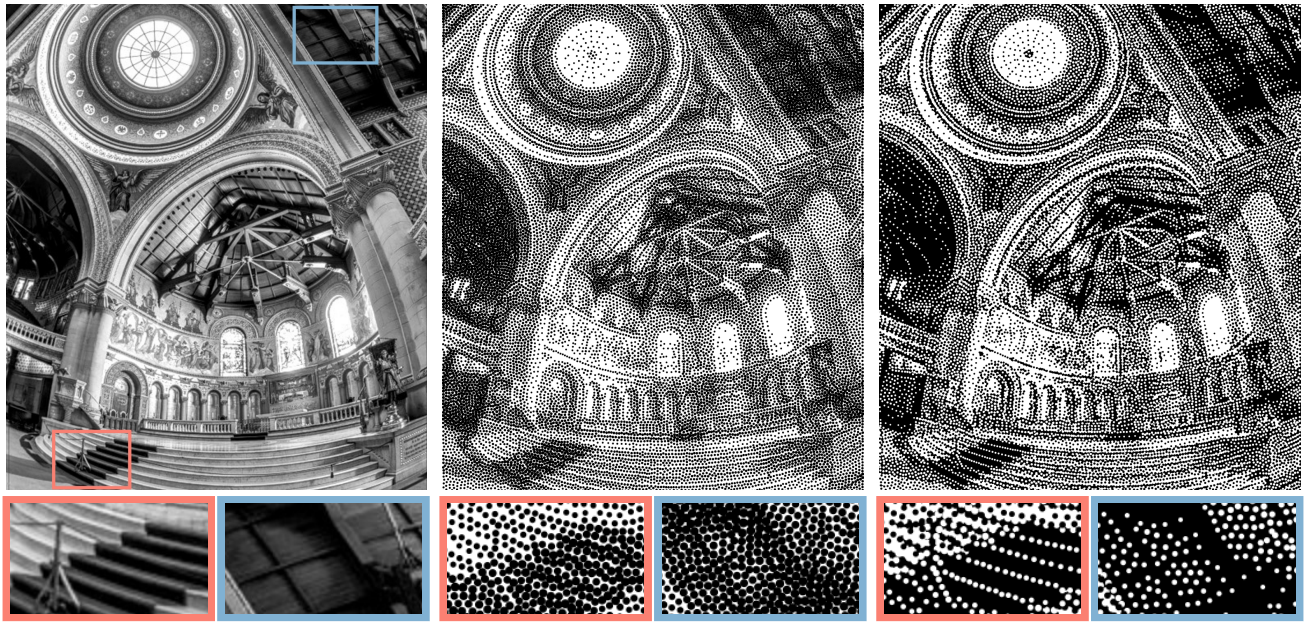
Multi-Class Inverted Stippling

CHRISTOPH SCHULZ*, University of Stuttgart, Germany
KIN CHUNG KWAN†*, University of Konstanz, Germany
MICHAEL BECHER, University of Stuttgart, Germany
DANIEL BAUMGARTNER, University of Stuttgart, Germany
GUIDO REINA, University of Stuttgart, Germany
OLIVER DEUSSEN, University of Konstanz, Germany
DANIEL WEISKOPF, University of Stuttgart, Germany



(a) Source (b) LBG [Deussen et al. 2017] (c) Our inverted stippling

Fig. 1. *Multi-class inverted stippling* uses black and white stipples to explicitly render positive and negative spaces. The cut-outs show that our method preserves fine details better than previous work and that the background is less noisy. Source image credit: "Memorial Church" by Paul Debevec. Used with permission.

We introduce *inverted stippling*, a method to mimic an inversion technique used by artists when performing stippling. To this end, we extend Linde-Buzo-Gray (LBG) stippling to multi-class LBG (MLBG) stippling with multiple layers. MLBG stippling couples the layers stochastically to optimize for per-layer and overall blue-noise properties. We propose a stipple-based filling method to generate solid color backgrounds for inverting areas. Our experiments demonstrate the effectiveness of MLBG in terms of reducing overlapping and intensity accuracy. In addition, we showcase MLBG with color stippling and dynamic multi-class blue-noise sampling, which is possible due to its support for temporal coherence.

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

Additional Key Words and Phrases: Stippling, Negative space, Voronoi Diagram, Linde–Buzo–Gray-Algorithm, Sampling

ACM Reference Format:
Christoph Schulz, Kin Chung Kwan, Michael Becher, Daniel Baumgartner, Guido Reina, Oliver Deussen, and Daniel Weiskopf. 2021. Multi-Class Inverted Stippling. *ACM Trans. Graph.* 40, 6, Article 245 (December 2021), 12 pages. https://doi.org/10.1145/3478513.3480534

*Joint first authors
†Corresponding Author.

Authors' addresses: Christoph Schulz, University of Stuttgart, Stuttgart, Germany, Christoph.Schulz@visus.uni-stuttgart.de; Kin Chung Kwan, University of Konstanz, Konstanz, Germany, kin-chung.kwan@uni-konstanz.de; Michael Becher, University of Stuttgart, Stuttgart, Germany, Michael.Becher@visus.uni-stuttgart.de; Daniel Baumgartner, st141786@stud.uni-stuttgart.de, University of Stuttgart, Stuttgart, Germany; Guido Reina, University of Stuttgart, Stuttgart, Germany, Guido.Reina@visus.uni-stuttgart.de; Oliver Deussen, University of Konstanz, Konstanz, Germany, Oliver.Deussen@uni-konstanz.de; Daniel Weiskopf, University of Stuttgart, Stuttgart, Germany, Daniel.Weiskopf@visus.uni-stuttgart.de.

0730-0301/2021/12-ART245 $15.00
https://doi.org/10.1145/3478513.3480534

1 INTRODUCTION

Stippling is an illustration technique that emulates shading using a large number of well-placed dots; it is heavily used by artists and illustrators. Previous research on computer-generated stipple illustrations (see the survey by Martín et al. [2017]) led to pleasing results, but the problem remains that dots often cannot represent structures in the areas of high stipple density well; unwanted patterns appear in these areas. In this paper, we improve the quality of these areas by mimicking a technique used by artists: *inversion*.

In dark areas, since gaps between stipples are more visually prominent than the stipples themselves, artists often distribute these gaps deliberately to convey the impression of fine details. They create the effect of negative space by filling an area with dense stipples for a solid black background and leaving white gaps to represent fine details. This creates an illusion of using both black and white stipples (Figure 2). Martín et al. [2011] also documented this inversion technique.

However, existing stippling methods [Balzer et al. 2009; Deussen et al. 2000, 2017; Hiller et al. 2003; Secord 2002] cannot achieve inversion, as they cannot control stipple gaps for negative space. To accomplish inversion in computer-generated stippling, we create stipples with the background color to explicitly "draw" the negative space. In other words, we stipple with both black stipples and white stipples. We also mimic the filling process of artists: we fill the background (i.e., gaps between stipples) to create a solid color background for the negative and positive space (Figure 1(c)). We call this method *inverted stippling*.

Inverted stippling can preserve fine details and dark intensities and involves two classes of stipples (positive and negative). This is challenging since existing methods only work on a single class. In this paper, we extend Linde-Buzo-Gray (LBG) stippling [Deussen et al. 2017] to support multiple classes. We name it *multi-class LBG* (MLBG) stippling. This method uses multiple layers to process individual stipple classes and couples the layers using a stochastic approach. By doing so, we maintain the blue-noise property for the individual stipple sets of each class (within-class) and for the combined set of all classes (cross-class). To further improve the quality of our results, we refine the rendering order of stipples to tackle overdraw and fill the remaining holes in the background with a subsequent stippling pass. This stippling pass ensures that our generated results are purely composed of stipples.

Multi-class stippling is flexible and therefore can readily achieve versatile stippling effects that go beyond traditional stippling such as color stippling (Figure 14), where different base colors are used. Our experiments show that inverted stippling preserves the density in dark areas well and generates clean stippling results. Our analyses demonstrate that our coupling design effectively reduces the overlap of stipples, and that our method improves the representation of high-density areas. We also showcase our multi-class approach using different applications such as object distribution and visualizations. Furthermore, MLBG stippling inherits the temporal coherence and adaptive stipple numbers from LBG stippling. Thus, it represents also a multi-class blue-noise sampling method that is temporally coherent and does not require any input of the number of sample points.

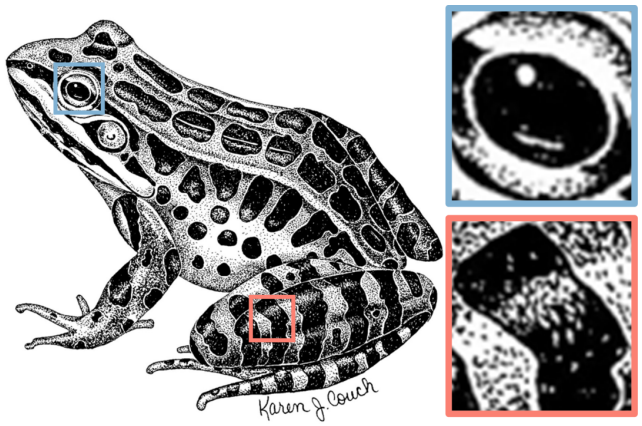


Fig. 2. Inversion in a famous stippling example. It contains illusions of "white stipples" in the image. Image credit: "Pickerel Frog" by Karen J. Couch. The image is in public domain.

Our main contributions are as follows: 1) We model a stippling technique from artists, inversion, for computational stippling. 2) We introduce MLBG stippling for multi-class blue-noise sampling with temporal coherence. 3) We present a stochastic layer coupling approach to maintain both within-class and cross-class blue noise properties.

2 RELATED WORK

Black-and-White Stippling. Deussen et al. [2000] presented a stippling method using Lloyd's algorithm [1982], a Voronoi-based optimization, to fill user-selected regions with stipples. Many works followed this idea. Secord [2002] added weights to incorporate intensity values from input images to guide stipple placement. Hiller et al. [2003] proposed using simple shapes instead of rounded stipples to perform stippling. Balzer et al. [2009] constrained all stipples to the intensity of the respective Voronoi cell. Deussen et al. [2017] added the idea of Linde-Buzo-Gray (LBG) optimization, adding splitting and merging operations, to Lloyd's optimization.

Other stippling works used different approaches. Kopf et al. [2006] proposed Wang tiles that contain predefined blue-noise point sets. Tiling these tiles creates stippled images. Mould [2007] transformed an image into a grid-shaped graph data structure and placed the stipples using Dijkstra's algorithm [1959]. Pang et al. [2008] used simulated annealing to generate halftoning images and measured the quality using structural similarity index measure (SSIM). Kim et al. [2009] formulated stippling as a texture synthesis problem using real artworks as texture examples. Martín et al. [2010; 2011] followed this idea. They generated resolution-dependent stipple images using an example-based halftoning technique with hand-drawn stipple dots examples. Li and Mould [2011] used error diffusion [Floyd and Steinberg 1976] to place stipples. They used a priority order to draw stipples in extreme regions first (i.e., both highest density and lowest density). Unlike our higher-density-first rendering order, their order is for minimizing error. Rendering order has zero impact in single-class stippling. Fattal [2011] modeled stipples with a kernel density function and fit these kernels into images to approximate the density. Ma et al. [2018] used incremental Voronoi sets (IVS)

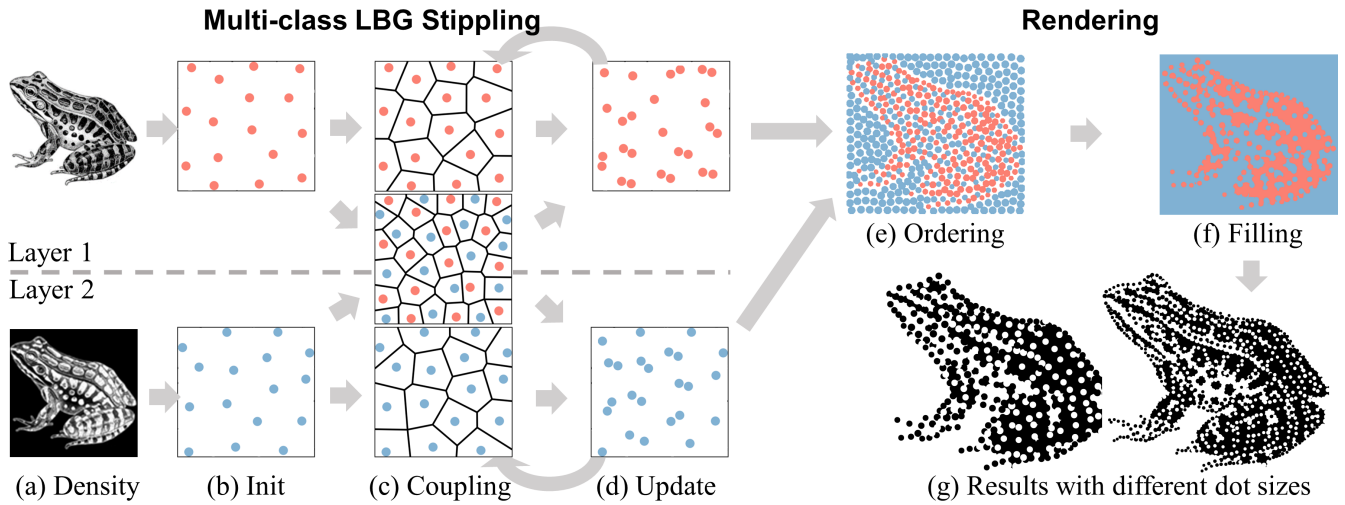


Fig. 3. Overview of our inverted stippling. Our MLBG method places stipples for multiple classes in different layers individually, and couple the layers during optimization. A rendering process for inverting backgrounds is then performed to determine rendering order and fill the remaining gaps. By varying stipple size, we can generate inverted stippling with different resolutions. Input image was created by SungYe Kim [Kim et al. 2009]. Used with permission.

to improve computing performance. Since it is hard to represent pure dark colors in stippling due to the gaps, Azami et al. [2019] presented a method that is analogous to ours. They combined stipples with filled polygon to produce pure dark areas. However, they filled all the gaps with black in this dark area and could not achieve negative space effects. Although there is a large number of stippling works, they mainly focus on black stipple layouts. None of them can explicitly control the stipple gaps to create negative space. Most importantly, they are single-class supported, making them not applicable for inverted stippling. For more detail, we refer readers to a comprehensive survey by Martín et al. [2017].

Color Stippling. Existing color stippling works [Houit and Nielsen 2011; Jang and Hong 2005; Ma et al. 2019] typically use black-and-white stippling methods followed by coloring each dot. Another straightforward approach uses error diffusion [Floyd and Steinberg 1976] to color the stipples from a black-and-white stippled result. Although these post-coloring methods support multiple colors, they cannot maintain the within-class blue-noise property of the points as being single-class methods. In inverted stippling, most of the time, only one class is visible in a region (Figure 1(c)). Using such post-coloring approaches causes uneven distribution of visible stipples in inverted stippling whereas our method does not.

Multi-class Sampling. Another kind of related work is multi-class blue-noise sampling. Wei [2010] generated multi-class blue-noise distributions using dart throwing [Cook 1986]. Chen et al. [2012] presented a variational capacity-constrained Voronoi tessellation. Jiang et al. [2015] used smoothed particle hydrodynamics (SPH) to compute particle distributions as fluid. Qin et al. [2017] presented another relaxation method using constrained Wasserstein barycenters. Ecormier-Nocca et al. [2019] use the pair correlation function (PCF) to measure the blue noise distributions property in gradient descent. Although these methods support multiple classes, they require an input number of samples. However, it is hard for users to determine the numbers of stipples when there are multiple classes of stipples. Thus, these methods are not suitable for inverted stippling. In contrast, our MLBG method has no requirements regarding a predefined number of samples and it is temporally coherent.

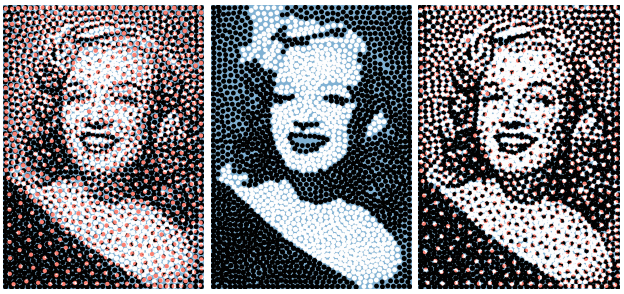
3 INVERTED STIPPLING

Inverted stippling uses black and white stipples (i.e., two classes) to explicitly "draw" negative space. To this end, we extend Linde-Buzo Gray (LBG) stippling [Deussen et al. 2017] to multi-class LBG (MLBG) stippling. We briefly introduce LBG stippling first and then describe our extension.

3.1 LBG Stippling

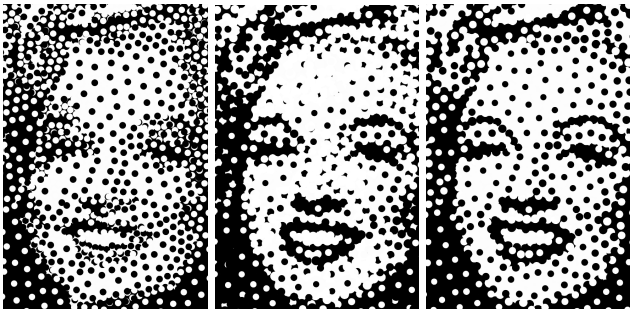
Given a target density map (i.e., image intensity) and a range of stipple sizes, LBG stippling starts with one or several randomly distributed stipples (one in our implementation) on the canvas. It calculates a Voronoi diagram from the stipples and determines the required stipple sizes to represent the target intensity within their Voronoi cells. All stipples are first moved toward the centroid of their Voronoi cells. This process is also known as Centroidal Voronoi Tessellation (CVT) in Lloyd's algorithm [1982]. CVT equalizes the distances between stipples and thus establishes the blue-noise property [Schlömer and Deussen 2011; Wei and Wang 2011]. Then, if a stipple is too large (i.e., excessive intensity) to represent the target intensity within its Voronoi cells, it is split into two. If it is too small (i.e., insufficient intensity), it is removed. The algorithm iterates until no more splitting or removal happens, or the maximum number of iterations is reached.

In contrast to Lloyd's method, LBG stippling does not require a predefined number of stipples. The algorithm can determine the number of stipples automatically. This characteristic makes LBG stippling more intuitive for users, as it is not easy to determine such a number for arbitrary images. However, LBG stippling supports only a single class of stipples.



(a) Independent (■ 18.4%, ■ 18.4%) (b) One-class (■ 0.0%, ■ 25.7%) (c) Ours (■ 9.3%, ■ 10.4%)

Fig. 4. Stippled image for two classes (black and white) using different approaches. The overlap areas (■) and gap areas (■) are highlighted. Percentages of overlap and gap areas for the bottom images are in brackets.



(a) Without coupling (b) Without ordering (c) Ours

Fig. 5. The influence of our processing steps on visual quality.

3.2 Multi-class LBG Extension

As mentioned above, for representing negative space, we need to extend LBG to MLBG stippling, which uses multiple layers. The *layers* are canvases of the same size, each for one class of stipples. For black-and-white inverted stippling, we need two layers: one creates positive space (black stipples), and one creates negative space (white stipples). In this case, we use the input image intensity and its inverted version as the density maps for each layer (Figure 3(a)).

One straightforward way would be to generate a stippled image for each layer independently using LBG stippling and then overlay the results. However, this independent approach cannot establish a cross-class blue-noise property (Figure 4(a)) and results in considerable overlaps with other layers (18.4%) due to missing inter-class correlation. Wei [2010] observed similar issues and concluded that generating classes together is more desirable. We can establish a cross-class blue-noise property by simply coupling all stipples together and processing them as one class. This, however, fails to create stipples in isolated regions (e.g., missing dots for eyebrows and teeth in Figure 4(b)) since stipples from different classes block each other. This problem is also known as the conflict of Voronoi diagrams [Qin et al. 2017]. Furthermore, it disregards the within-class blue-noise property of each class of stipples.

To overcome this issue, we introduce a stochastic optimization strategy: for each layer l, we compute a within-class Voronoi diagram V_l using the individual stipples in this layer. Also, we combine

ALGORITHM 1: Multi-class Linde-Buzo-Gray stippling

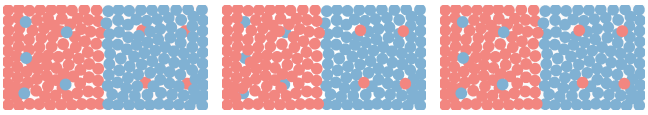
Input : Layers $l_{0\ldots n} \in L$
Range of stipple sizes $[r_0, r_1]$
Output: Set of all stipples for each layer $S_{0\ldots n}$

Initialize $S_{0\ldots n}$ with random positions
repeat
  $V_G \leftarrow$ Voronoi diagram of $S_{0\ldots n}$
  for each layer $l \in L$
    $V_l \leftarrow$ Voronoi diagram of S_l
    for each stipple $s \in S_l$
      $C_l \leftarrow$ centroid of Voronoi cell $V_l(s)$
      $C_G \leftarrow$ centroid of Voronoi cell $V_G(s)$
      Move s to either C_l or C_G based on probability P
      Choose a new size within $[r_0, r_1]$ for stipple s
      if insufficient intensity **then** split s in two
      if excessive intensity **then** remove s
until stable **or** maximum iterations reached

the stipples of all layers and compute a cross-class Voronoi diagram V_G (Figure 3(c)). With these two Voronoi diagrams, each stipple has two potential centroids: the within-class centroid C_l and the cross-class centroid C_G from V_l and V_G, respectively. Moving stipples to C_l enhances the within-class blue-noise characteristics, whereas moving to C_G enhances the cross-class blue-noise characteristics. Now, we randomly move each stipple to either C_l or C_G based on a probability P to balance within-class and cross-class blue-noise characteristics. Lastly, we split or remove the stipples as in LBG (Section 3.1). Putting it all together is our MLBG, shown in Algorithm 1.

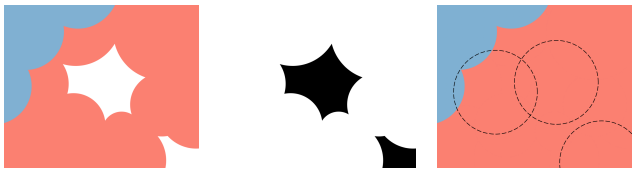
Concerning the probability P, we found in our experiments that an adaptive value based on stipple density works best. In inverted stippling, higher-density stipples often become invisible in our results due to the stipple rendering process (Sections 3.3 and 3.4). Viewers can only perceive the lower-density stipples (e.g., Figure 5(c)), the within-class blue-noise property of these stipples contributes most to the visual quality. Thus, lower-density stipples should have higher chances of moving toward their within-class centroids to increase within-class blue-noise characteristics. In contrast, higher-density stipples should move more often toward their cross-class centroids to keep away from lower-density stipples and reduce overlap. Specifically, we set the probability for a stipple to move toward its within-class centroid as $P = D - d_l$, where D is the sum of average densities of all layers within its Voronoi cell, and d_l is the average density of its layer l within its Voronoi cell. Here, the density is the intensity value of the provided density map. Note that $D = 1$ for black-and-white inverted stippling. We clamp P if it exceeds one.

Figure 5(a) shows an example of inverted stippling without stochastic coupling. The inter-class overlap generates many crescent-shaped artifacts. Our method suppresses these overlaps and thus results in better quality.



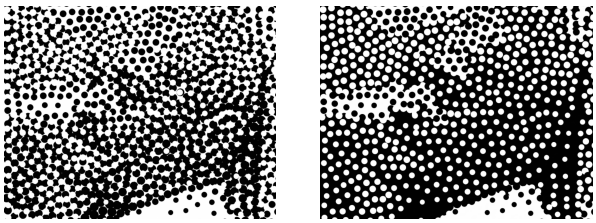
(a) Red first (b) Blue first (c) Ours

Fig. 6. (a) and (b) Some stipples drawn first are barely visible due to overdraw. (c) Our rendering approach avoids this issue.



(a) (b) (c)

Fig. 7. Our background-filling approach. (a) Gaps are formed by stipples. (b) We render the stipples in white on a black background to generate an intermediate density map. (c) Apply LBG stippling to place new stipples and draw them at the back of (a).



(a) Before filling (b) After filling

Fig. 8. The remaining gaps (a) make the stipples hard to represent solid background. Our filling (b) make the image cleaner. We recommend zooming in to identify the gaps.

3.3 Density-based Rendering Order

Once we have a stipple placement, we can render the stipples in any order. However, this may not generate a visually pleasing image due to overdraw (Figure 5(b)). Similar to transparency in traditional rendering, the drawing order of the stipples influences their visual quality.

Thus, we process our stippling results by establishing a better rendering order (Figure 3(e)): by observation, we found that it is preferable to place sparse stipples in front of densely distributed stipples (Figure 6). The rationale behind this is that the visual effect of overlaps at low-density stipples is visually more pronounced than that at high-density stipples. Therefore, we should avoid overdraw on the low-density stipples. To this end, we first sort all the stipples by their descending average density within their respective cell in the within-class Voronoi diagram. Higher-density stipples are drawn first and never overlap the lower-density stipples.

3.4 Background Filling

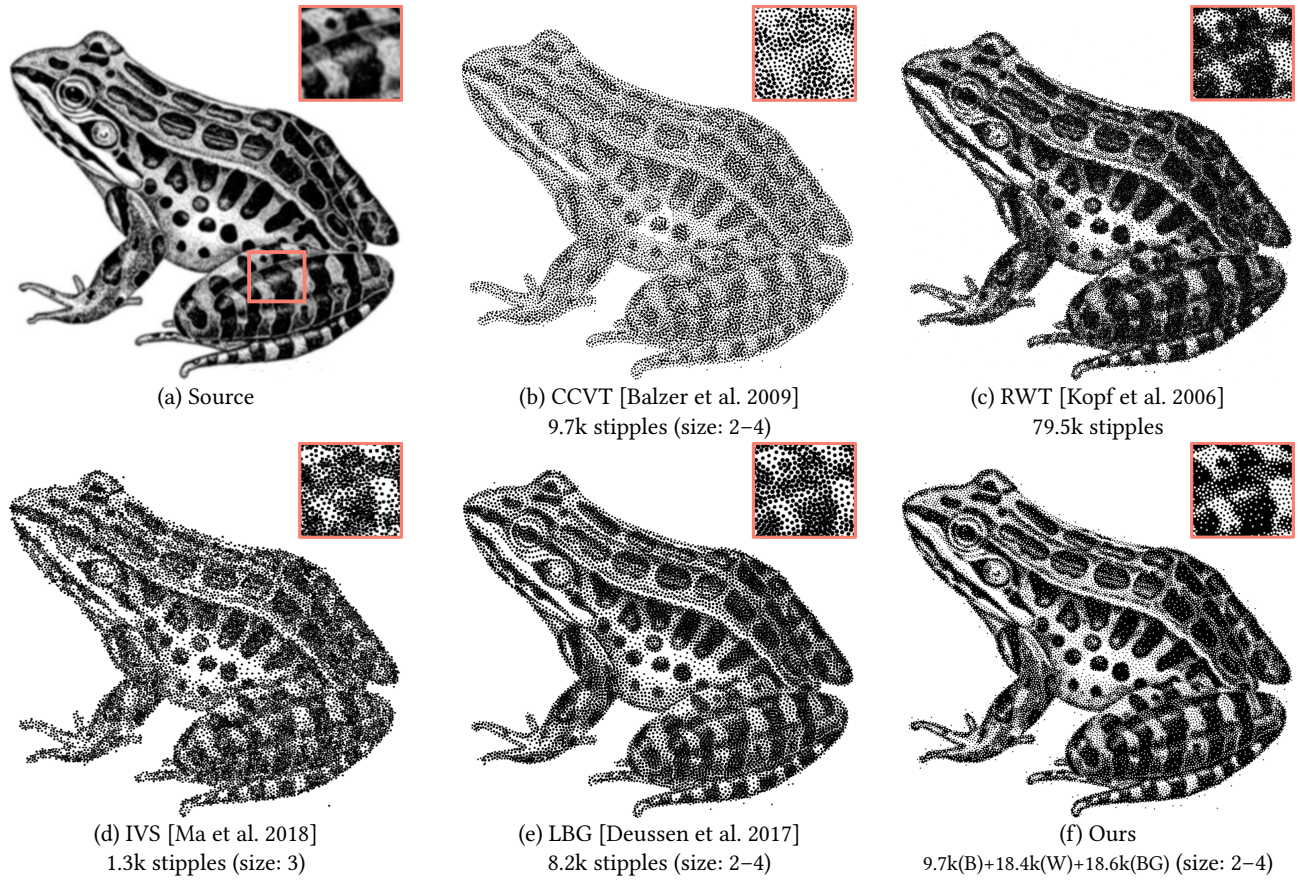
Our MLBG stippling already generates a good multi-class stippling distribution. However, similar to all existing stippling methods, it is hard to represent solid colors due to the remaining gaps between stipples (Figure 8(a)). Artists typically fill the canvas with dense stipples to create solid backgrounds. We mimic this technique and fill the remaining gaps between stipples for inverting backgrounds (Figure 3(f)). This could be done by filling the corresponding Voronoi areas. However, since stippling is a technique to create images using stipple primitives only, we want our inverted stippling to stay in this spirit—especially as users may require primitive representations, e.g., for engraving applications. Thus, we fill the gaps by adding new stipples (background stipples) below the existing stipples (foreground stipples), such that the background will not affect the layout by overlapping the foreground. Note that covering all the gaps with a minimum number of stipples can be NP-complete (depends on the primitive). Thus, we try our best to minimize the number of stipples rather than finding an optimal solution.

ALGORITHM 2: Background filling

Input : Set of foreground stipples S
Inputted image I and its inverted version I'
Maximum stipple size r
Output : Set of background stipples B

```
Gap mask G ← render S in white on black background
Copy S to B
Set the sizes of stipples in B as r
repeat
    V ← Voronoi diagram of B
    for each stipple s ∈ B
        Move s to centroid of V(s)
        if s cannot cover all gaps in V(s) then split s in two
        if V(s) cover no gap then remove s
        i ← total intensity of I in V(s)
        j ← total intensity of I' in V(s)
        if i ≤ j then
            Color of s ← black
        else
            Color of s ← white
until no split or remove happened
```

Our greedy approach for solving this polygon covering problem is another stippling pass (illustrated in Figure 7). Algorithm 2 shows our filling algorithm. We first render all foreground stipples in white (i.e., zero density) on a black background (i.e., highest density) to obtain a new density map (a gap mask). This allows us to mask out unnecessary areas. We then perform a single-class LBG stippling pass on this gap mask. We copy the existing foreground stipples to initialize the background stipples, as it is very likely that every foreground stipple requires one or more background stipples to cover all of its surrounding gaps. While this already gives good results, we modify the split/remove conditions for our filling purpose: If a Voronoi cell of a background stipple covers none of the black areas (i.e., zero total density), we remove the background stipple. If a background stipple cannot cover all the gaps in its cell (i.e., black areas remain outside the stipple ink), the stipple will be split into two. In our current implementation, the size of all background stipples is set to the maximum allowed size, which allows us to keep the number of background stipples low. Alternatively, the size can be varied depending on the application. Then, for each background

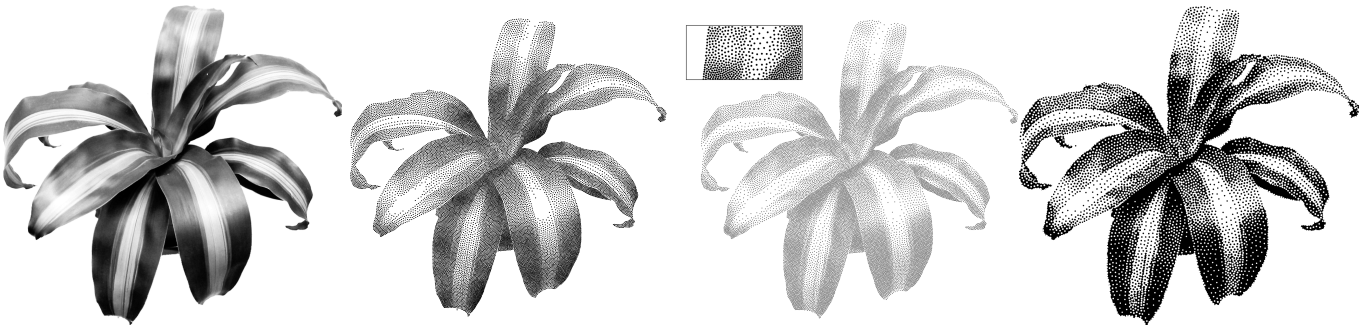


(a) Source
(b) CCVT [Balzer et al. 2009] 9.7k stipples (size: 2–4)
(c) RWT [Kopf et al. 2006] 79.5k stipples
(d) IVS [Ma et al. 2018] 1.3k stipples (size: 3)
(e) LBG [Deussen et al. 2017] 8.2k stipples (size: 2–4)
(f) Ours 9.7k(B)+18.4k(W)+18.6k(BG) (size: 2–4)

Fig. 9. Examples of black-and-white stippling using different selected methods. Note the differences in brightness and contrast. Image (b) was generated using StippleShop [Martín et al. 2017]. Images (c)-(e) were generated using the demo programs provided by Kopf et al. [2006], Ma et al. [2018], and Deussen et al. [2017], respectively. The labels (B), (W), and (BG) under our result (f) mean black stipples, white stipples, and background stipples, respectively. Source (a) was created by SungYe Kim [Kim et al. 2009]. Used with permission. More comparison images can be found in our supplemental material.

(a) Source
(b) RWT [Kopf et al. 2006]
(c) Ours

Fig. 10. Another comparison with RTW by Kopf et al. [2006]. Figures (a) and (b) originate from their work. Used with permission.



(a) Source (b) WCVD [Secord 2002] (c) CCVT [Balzer et al. 2009] (d) Ours

Fig. 11. Another example of black-and-white stippling. Figures (b) and (c) are from the respective papers. Source image credit: "Plant" by Adrian Secord. Used with permission.

(a) Source (b) CAS (c) Ours

Fig. 12. Comparison with CAS [Li and Mould 2011]. Source (a) originates from Mould [2007]. Figure (b) is from the CAS paper by Li and Mould [2011]. Used with permission.

Fig. 13. An example of our black-and-white inverted stippling and respective source images. The cat face features a fur texture and delicate whiskers. Source image credit: Pixabay user Alexas_Fotos. Used with permission.

stipple, we identify the layer with the highest density of the original density maps within its Voronoi diagram cell and use its color for the stipple. Finally, we insert these background stipples so that no stipple from the previous pass is overdrawn (Figure 8(b)).

According to our experiments, the first iteration of our filling can remove around 95% of the gaps, and only 0.1% remain after three iterations in most cases. Although we can sort background stipples as we did for the foreground, it does not have a noticeable impact on visual quality in our experiments. The potential reason for this can be that most background stipples are overdrawn and less obvious in small background areas.

Alternatively, when primitive representation is not required (e.g., rendering in pixel format), one can fill gaps using the layer color of the highest density within the gap. The difference is barely noticeable, but filling provides benefits regarding efficiency.

4 RESULTS

This section shows results and reports on our evaluations. Our CPU-based experiments ran on a PC with an AMD Ryzen 7 5800X CPU, whereas our GPU-based experiments ran on a PC with an Intel Core i7-6700 CPU and an NVIDIA GTX 1070 GPU. Our implementation was done in C++ and CUDA.

4.1 Result Images

Figures 1, 9, and 13 show examples of black-and-white inverted stippling. As demonstrated in Figure 1, original LBG stippling cannot represent dark regions well. The stippling looks brighter than the input because of white gaps in supposedly dark areas. In Figure 9, we compare our approach to other black-and-white stippling methods. We used StippleShop, a tool provided by Martín et al. [2017] for stippling benchmarks, to generate some of the comparison images. StippleShop is used for comparison in their additional material [Martín et al. 2017]. Note that using too many small stipples will lead to grayscale-like images, which is unwanted for stippling. It is hard to determine the stipple number for the existing stippling methods. Most of the methods cannot represent dark regions using ordinary parameter settings. Some methods can create solid color in dark regions by using a huge number of black stipples [Kim et al. 2009; Secord 2002]. However, they will also dim bright areas and cannot represent fine details in dark areas. To have a fair comparison, we tried our best to keep the stipples with similar density, and then carefully fine tune the parameters to obtain the best visual quality. Besides preserving the overall brightness, our result appears clearer than the other techniques since we can represent certain image features more directly due to inversion.

The sources in Figures 9, 10, 11, and 12 are the stippling examples also used in previous works. We included the comparisons with the images from the respective original paper. The stippled images from these other papers usually cannot well represent the original intensity. Please see our supplemental material for a more comprehensive comparison.

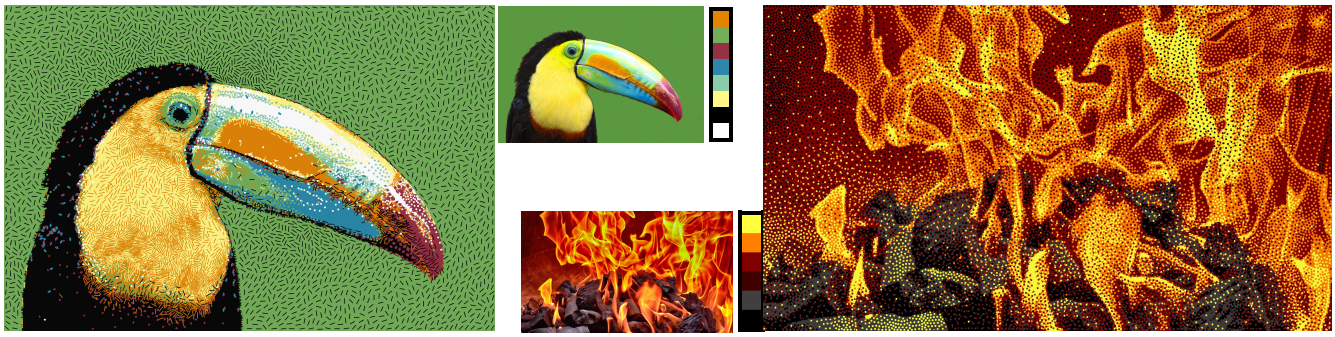
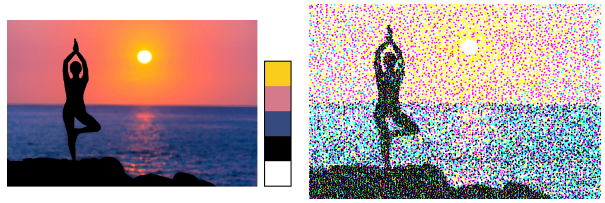


Fig. 14. Examples of color stippling results (left and right), source images, and used color palettes (middle). The fire uses circles only. The toucan exhibits a variety of non-traditional stipple shapes. We recommend zooming in to identify the various stipple effects. Source image credits: (Left) Tambako The Jaguar (https://www.flickr.com/photos/tambako/). (Right) Pixabay user Alexas_Fotos. Both used with permission.



(a) Source and selected colors (d) with CMYKW Wei [2010]

(b) with selected colors (e) with CMYKW LBG with error diffusion coloring.

(c) with selected colors (f) with CMYKW Our MLBG stippling

Fig. 15. Comparison of different methods and different colors for color stippling. Neither LBG with post-hoc error diffusion nor plain multi-class stippling generate clean results as our inverted stippling. Source image credit: Pexels user Cedric Lim Ah Tock. Used with permission.

Figure 13 shows more results. Our negative space in inverted stippling well preserves the structures of the cat whiskers. The white thin structures are hard to represent using pure black dots. Our inverted stippling can represent both bright objects and dark objects, which is challenging for traditional stippling. Moreover, the blurry edge at the background shows the smooth transition of different classes of stipples.

(a) Source (b) Without filter (c) With filter

Fig. 16. Small contributions of colors accumulate and result in noisy stipples. Source image credit: Pixnio user Milivojevic (CC0).

4.2 Color Stippling and Flexible Shapes

Besides black-and-white stippling, MLBG stippling allows us to create color stippling. For this, we need a reference color image and a user-provided color palette, which can be arbitrary. Alternatively, one could apply a palette extraction method to select colors automatically. Based on the colors provided, we decompose the color image into contribution layers using existing color decomposition methods [Tan et al. 2018]. These contribution layers are used as input density maps for our method. We use the same number of layers as the provided colors, as our MLBG naturally supports more than two classes. Unfortunately, non-obvious colors may have small contributions in some areas of the image. These small contribution values will accumulate and results in noisy stipples (Figure 16(b)). Although this problem also occurs in single-class stippling, the results are visually acceptable when the number of classes is small. For color stippling with more than two classes, this problem is more serious. To work around this issue, we clamp values that contribute less than a threshold (~10% in our case) and use a median filter to smooth the inputted image (Figure 16(c)).

Figure 14 shows color stippling results. We can determine the orientation of stipples by the second moment of their Voronoi cell or other data maps (e.g., vector field (Figure 22(b)), or edge tangent flow [Kang et al. 2007]) depending on the applications. It is worth noting that Reinert et al. [2013] recommended using a signed distance function (SDF) to compute Voronoi diagrams for primitives with variant sizes and shapes, as it reduces overlap. We followed

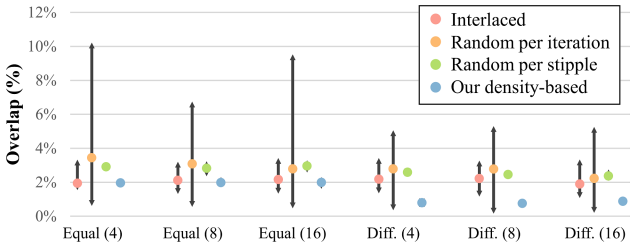


Fig. 17. Overlap for different coupling schemes, configurations, and stipple sizes (in brackets). The lines represent the min-max values, and the dots are their means.

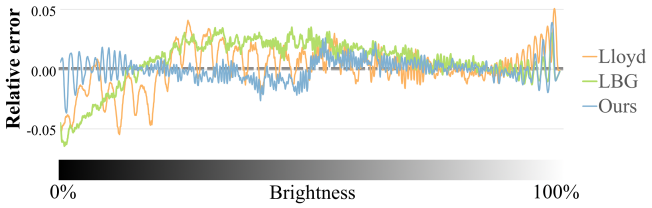


Fig. 18. Representation error between target density and achieved density: The zero line is ideal. Inverted stippling exhibits the smallest average error and the most balanced representation over the entire range of densities.

this recommendation and used SDF for our Voronoi diagrams. In Figure 15, we compare our inverted color stippling to other methods: LBG stippling with an additional post-hoc coloring using error diffusion approach, and the method of Wei [2010]. Manually selected colors and CMYKW colors are used for comparisons.

4.3 Layer-Coupling Scheme

We studied four layer-coupling schemes in terms of how they affect the quality of the results: 1) *Interlaced*: we consider the within-class Voronoi diagram in odd iterations and the inter-class one in even iterations. 2) *Random per iteration*: at the start of each iteration, we randomly chose within-class or inter-class Voronoi diagram, and all stipples consider the same Voronoi diagram in this iteration. 3) *Random per stipple*: similar to the previous case, except that we randomly chose within-class or inter-class for each stipple individually. 4) *Density-based*: selection as described in Section 3.2.

For this experiment, we use a 1024×1024 input image in two configurations, each consisting of two classes. One has equal densities, which are 0.33 ("Equal"). The other has unequal densities, which are 0.11 and 0.55 ("Diff."). An ideal result should show zero overlap and around 33% gap area. We ran MLBG stippling using the different schemes and different stipple sizes, each 40 times, and measured the inter-class overlap in pixels.

Figure 17 shows the statistics of the experimental results. The stipple size has nearly no impact on the overlap of the results. For the equal-density configuration ("Equal"), the average overlap for interlaced, random per iteration, random per stipple, and our density-based are 2.07%, 3.11%, 2.90%, and 1.98%, respectively. The density-based approach shows a lower overlap than the other schemes,and we found a significant improvement (paired t-tests $p < 0.05$) of the density-based scheme compared to random per iteration and random per stipple. Surprisingly, the interlaced scheme generates better results than these two as well. Although there is no significant improvement (repeated measures ANOVA ($F(2, 234) = 1.15, p = 0.32$)) between density-based and interlaced for the equal-density configuration, interlaced has a bias toward within-class or inter-class blue noise, depending on the last iteration being even or odd. Interlaced and random per iteration have higher ranges of overlap, which implies that they may stop in local optima.

For the case with different densities ("Diff."), we found a significant improvement (paired t-tests $p < 0.01$) from the density-based scheme to all the other schemes. Their average overlaps are 2.10%, 2.60%, 2.47%, and 0.81%, respectively. This shows that our density-based coupling scheme is effective in reducing overlaps.

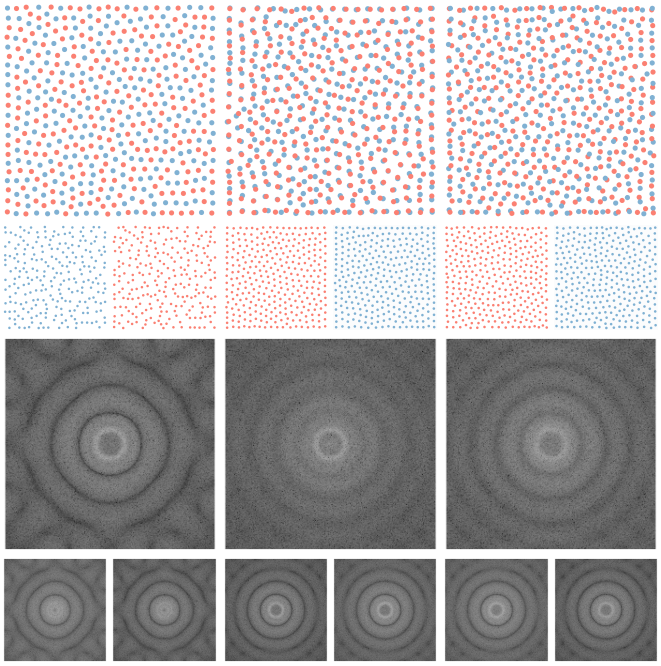
4.4 Density Representation

Existing stippling methods often cannot represent the density of an input image well. We demonstrate the ability of our method to do so by a quantitative experiment: A good method should accurately render all stipple densities with correct intensity (i.e., black-to-white ratio). Thus, we stipple a linear gradient with 2048×1600 pixels using different methods, and then examine the black-to-white pixel ratio of each column of the stippled results and compare it against the ground truth grayscale value. For a fair comparison, we set the number of stipples for Lloyd's method to the stipple number we automatically obtain using the LBG-based methods. The stipple size is 4 pixels for all methods. Figure 18 shows their relative errors between the ideal intensity and the measured intensity, where zero in the middle means no error. To improve readability, we smooth the measurements using a window of 32. In the dark region, the errors of all methods increased, while our inverted stippling has the lowest error in this region. It is also interesting to note that the error of our method has a smooth change in sign at the middle (50% brightness). This change shows where background colors are inverted. The absolute error is small in this region, and thus we can hide the transition of the backgrounds.

4.5 Blue-Noise Property

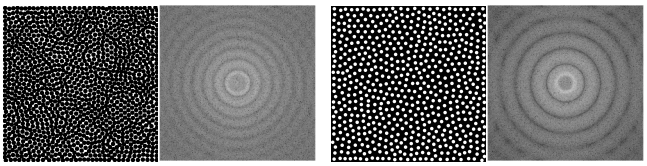
It is a community standard to visualize the quality of blue noise using Fourier analysis [Schlömer and Deussen 2011]. Following this standard, we stipple a uniform square with two classes, each at 25% density, using different approaches. Figure 19 compares the resulting images in the spatial and frequency domains. The post-coloring approach using error diffusion after LBG performs worst since it cannot maintain within-class blue noise. This results in uneven stippling for the individual classes. The independent approach shows blue-noise characteristics for the individual layers, but the inter-class blue noise has a low signal-to-noise ratio. Our stochastic coupling shows blue-noise characteristics for the individual layers, with a better inter-class blue noise than independent.

Figure 20 illustrates the blue-noise property of traditional stippling and inverted stippling in dark areas. The irregular small gaps in traditional stippling degrade the blue-noise property, whereas our inverted stippling maintains the blue-noise property better.



(a) Post-coloring (b) Independent (c) Ours

Fig. 19. Uniformly stippled squares in the spatial domain (top row) and frequency domain (bottom row). And the one of their individual layer. Different methods are compared. Ideal blue noise has a clear repeating ring (rings decay from inside to outside) and isotropic properties (low variation within the rings).



(a) Traditional (b) Inverted

Fig. 20. Comparison of blue noise between traditional stippling (a) and inverted stippling (b) using a uniform dark area.

4.6 Performance

Table 1 summarizes the information and timing statistics of our results shown in this paper. The number of gap stipples is usually larger than the one of black stipples and white stipples. If an image has a black background, this number will be much larger as we assume that the canvas is in white in our experiments. The CPU times and GPU times in the table represent the individual execution times using our CPU-based implementation and GPU-based implementation, respectively. In our CPU-based implementation, background filling takes 14% to 68% of the MLBG stippling execution time. As expected, the GPU implementation is faster than the CPU implementation and achieves a close-to interactive rate.

Here, we also evaluate the performance of MLBG stippling using GPU acceleration. We could not find a change in the convergence rate of our MLBG stippling compared to LGB stippling. Also, we could not find a correlation with density per layer. Thus, we will only consider total execution time for this discussion.

Table 1. The timing statistic of our MLBG. (B): Black, (W): White, (BG): Background, (S): Stippling, (R): Rendering.

Figure	1	9(f)	10(c)	11(c)	12(c)	13
Resolution	1.0k×1.5k	1.0k×1.0k	0.9k×0.9k	1.0k×1.0k	0.8k×0.8k	0.8k×0.8k
Stipple Size	2–3	2–4	2–4	2–4	2–3	2–4
#Stipple (B)	33,573	9,712	7,806	7,601	25,185	11,295
#Stipple (W)	31,392	18,434	14,975	16,992	20,885	14,975
#Stipple (BG)	88,565	18,683	12,067	11,836	53,740	24,409
CPU Time (S)	19.8s	12.3s	8.3s	9.3s	14.9s	4.8s
CPU Time (R)	7.8s	3.4s	2.4s	2.8s	7.4s	0.7s
GPU Time (S+R)	5.8s	1.7s	2.1s	3.1s	1.0s	2.1s

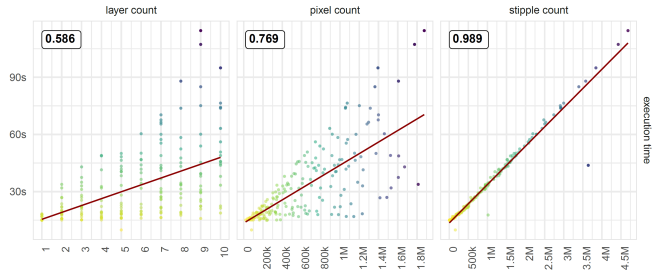


Fig. 21. Execution time correlated with layer count, pixel count, and stipple count. The Pearson index is shown in the upper left of each plot. The red line shows linear regression.

Figure 21 shows how our GPU-based implementation scales with layer count, image pixels, and stipples regarding the number of iterations and execution time. The number of layers has less influence than the number of pixels and the number of stipples. Outliers can be attributed to early termination.

5 DISCUSSION

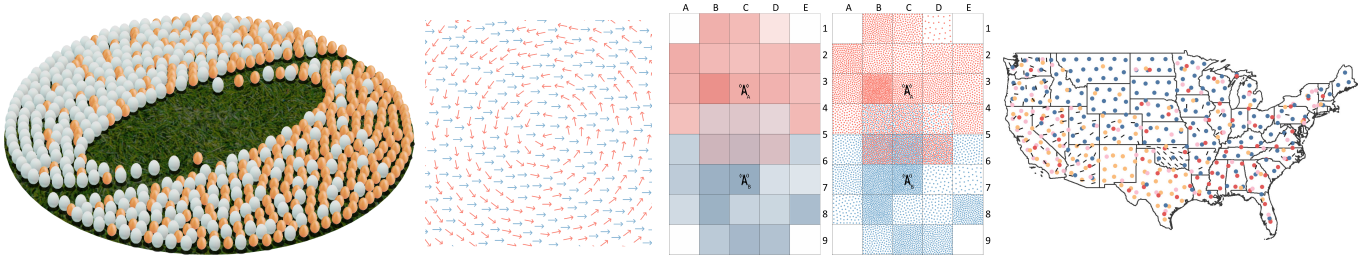
This section discusses several issues of our method: temporal coherence, applications, multi-class sampling, and transferability to other methods.

5.1 Temporal Coherence

LBG stippling is temporally coherent due to its relaxation process and the adaptive stipple number mechanism. MLBG stippling maintains all these properties and thus is temporally coherent as well.

To demonstrate this, we created a stippled animation with multi-class stippling (not inverted stippling). For each frame, we use stipples from the immediately preceding frame to initialize the stippling. Then, we run two to three iterations of MLBG stippling to update the scene. Note that we disabled our ordering process for every frame in animation as it is not temporally coherent. We only determine the rendering order of a stipple once when they are generated. Fortunately, the movement for stipples makes the ordering problems less obvious.

Our animation shows the tides of an island. All stipples move smoothly between the gaps of the others to avoid overlaps, and the stipples of the sea spawn toward the middle and cover the island without any sudden changes. This animated effect is problematic for existing stippling methods without an adaptive stippling number.



(a) Object distribution (b) Vector field visualization (c) Visualization with transparency (d) Multivariate demographic map

Fig. 22. Four applications for our method: (a) the method can be used for mixing object distributions (e.g., trees in forests, people in a crowd, artistic usages); (b) in scientific visualization, vector fields can be mixed and overlaid, and both fields are still visible; (c) in contrast to alpha blending (left), MLBG stippling (right) allows keeping colors and distribution variations in two overlaid classes; (d) we can visualize multivariate data by distributing blue-noise colored stipples on maps. We recommend zooming in to identify the details.

We refer the reader to the supplemental video for the animated effect.

5.2 Applications

To demonstrate the broad applicability of our MLBG stippling, we showcase four different examples: object distribution, vector field visualization, visualization with transparency, and multivariate demographic map.

One common application for multi-class sampling is object distribution. Given density maps and masks, our method can distribute objects to generate compelling images (Figure 22(a)) for artistic use. Another application is vector field visualization. Our method can create a joint visualization of multiple vector fields in one image by distributing multiple classes of blue-noise-distributed arrows and aligning the orientation of arrows with the field (Figure 22(b)).

Figure 22(c) shows an example that visualizes how well the population can access certain antennas on a map grid. Color transparency indicates the percentage of the population. In the overlap area, alpha blending (left image) will unavoidably introduce unwanted new colors (mostly gray). Since each color might have its meaning (i.e., class indicator), new colors are undesirable. Our stippling method (right image) preserves the original colors of each region. Moreover, only the dominant colors are visible in the overlap area with alpha blending, and hence the density information is lost. In contrast, the density information is clearly visible in our stippling result. Note that the perception of color-mixing is different for alpha blending and stippling. Thus, their color may be perceived differently.

Figure 22(d) shows another example of visualization of multivariate demographic map; the map shows the population of people (i.e., density) with different ethnicity (i.e., multi-class) in a country. Our method can clearly visualize the multi-class information in the same figure.

5.3 Multi-class Sampling

MLBG stippling belongs to the category of multi-class blue-noise sampling methods. Table 2 shows the differences between multi-class sampling methods from the literature; compared to the others, MLBG stippling is temporally coherent and does not require setting a predefined number of samples compared to the other methods. This is a major advantage since it is almost impossible for a user to set a number of samples per class, and frame by frame for animation. Dart throwing in Wei [2010] is inherently not temporally coherent. Although the other methods allow for gradual movement of samples, their fixed number of samples breaks temporal coherence regarding density. Hence, these methods are unsuitable for dynamic sampling, as the number of required samples in each frame can vary. To the best of our knowledge, MLBG stippling is the first multi-class blue-noise sampling method with temporal coherence. We did not compare them in detail as it is out of the scope of this paper, but leave a comparative study for future work.

Table 2. The difference of multi-class sampling methods. "Dart." for dart throwing. "Relax." for relaxation. "Ad." for adaptive. "Tem. Co." for temporal coherence.

Method	Type	#Stipple	Tem. Co.	GPU
Wei [2010]	Dart.	Fixed	No	Limited
Chen et al. [2012]	Relax.	Fixed	Partial	Full
Jiang et al. [2015]	Relax.	Fixed	Partial	Full
Qin et al. [2017]	Relax.	Fixed	Partial	Full
Ecormier-Nocca et al. [2019]	Relax.	Fixed	Partial	Full
Ours	Relax.	Ad.	Yes	Full

5.4 Transferability to Other Methods

Our MLBG stippling is an extension of LBG stippling. Its key idea is not limited to LBG but could extend to other stippling methods using CVT, such as Lloyd's method [1982]. We strongly believe that one can transfer our stochastic coupling approach to these methods. We can also apply our background-filling process to other stippling works, e.g., [Azami et al. 2019], and approximated coverage problems.

6 CONCLUSION

We have introduced a computational method for *inverted stippling* by mimicking an inversion technique used by artists. Inverted stippling uses both black stipples and white stipples for rendering positive space and negative space. Our extension to LBG stippling [Deussen et al. 2017], namely MLBG stippling, uses layers to obtain inverted stippling. In this context, we have proposed a stochastic layer coupling approach to achieve the blue-noise property for individual

layers and the combined layers. Finally, we have introduced a background-filling approach to achieve solid color by adding new stipples. These allow us to generate stippled images with better detail representation and better preservation of intensity in dark areas.

Limitations and Future Work. Similar to LBG, the discretization (e.g., rasterization) in our implementation introduces inaccuracies. Supersampling or a continuous optimization approach can improve accuracy but is also more expensive to compute. Although there are faster real-time stippling works, our inverted stippling still achieves interactive frame rates with parallel GPU computation. Further improving the performance could benefit the usability of our method for dynamic interactive cases. Our color stippling results highly depend on the quality of density map input; it truly represents the errors from the color decomposition, such as the non-uniform value in uniform background (Figure 14). Another limitation of our method is that stochastic layer coupling works well only when there are sufficient iterations. Inverted stippling may drive up the stipple number, as it needs to render the stipples from multiple classes and the background stipples. In our current implementation, we did not remove the invisible stipples (e.g., white stipples on white paper), which could be one solution. If there are only 2–3 iterations (e.g., due to performance constraints in animation), the resulting within-class and cross-class blue-noise properties are not fully balanced. In such cases, we recommend using the interlaced coupling scheme with at least two iterations. Lastly, our ordering step is not temporally coherent. We disable this step for dynamic cases at this time. A temporally coherent order for stippling could be interesting for future research. In the future, we are also interested in studying the perception quality of MLBG stippling in more detail, similar to studies by Maciejewski et al. [2008] and Spicker et al. [2017].

ACKNOWLEDGMENTS

The authors would like to thank Jochen Görtler and Angelika Knothe for the fruitful discussions. This research was funded by the *Deutsche Forschungsgemeinschaft* (DFG, German Research Foundation) — Project ID 251654672— TRR 161 (Project A01 and A04), and —Project ID 279064222— SFB 1244 (Project B05).

REFERENCES

Rosa Azami, Lars Doyle, and David Mould. 2019. Stipple Removal in Extreme-tone Regions. In *ACM/EG Expressive Symposium*. https://doi.org/10.2312/exp.20191083

Michael Balzer, Thomas Schlömer, and Oliver Deussen. 2009. Capacity-Constrained Point Distributions: A Variant of Lloyd's Method. *ACM Trans. Graph.* 28, 3 (2009), 1–8. https://doi.org/10.1145/1576246.1531392

Zhonggui Chen, Zhan Yuan, Yi-King Choi, Ligang Liu, and Wenping Wang. 2012. Variational Blue Noise Sampling. *IEEE Trans. Vis. Comput. Graph.* 18 (2012). https://doi.org/10.1109/TVCG.2012.94

Robert L. Cook. 1986. Stochastic Sampling in Computer Graphics. *ACM Trans. Graph.* 5, 1 (1986), 51–72. https://doi.org/10.1145/7529.8927

Oliver Deussen, Stefan Hiller, Cornelius Van Overveld, and Thomas Strothotte. 2000. Floating Points: A Method for Computing Stipple Drawings. *Comput. Graph. Forum* 19, 3 (2000), 41–50. https://doi.org/10.1111/1467-8659.00396

Oliver Deussen, Marc Spicker, and Qian Zheng. 2017. Weighted Linde-Buzo-Gray Stippling. *ACM Trans. Graph.* 36, 6 (2017), 1–12. https://doi.org/10.1145/3130800.3130819

Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1, 1 (1959), 269–271.

Pierre Ecormier-Nocca, Pooran Memari, James Gain, and Marie-Paule Cani. 2019. Accurate Synthesis of Multi-Class Disk Distributions. *Computer Graphics Forum* 38, 2 (2019). https://hal.inria.fr/hal-02064699

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

Robert W. Floyd and Louis Steinberg. 1976. An Adaptive Algorithm for Spatial Greyscale. *Proc. Soc. Inf. Disp.* 17, 2 (1976), 75–77.

Stefan Hiller, Heino Hellwig, and Oliver Deussen. 2003. Beyond Stippling - Methods for Distributing Objects on the Plane. *Comput. Graph. Forum* 22, 3 (2003), 515–522. https://doi.org/10.1111/1467-8659.00699

Thomas Houit and Frank Nielsen. 2011. Video Stippling. In *Adv. Concepts for Intel. Vis. Sys.* Springer, 384–395.

Seok Jang and Hyun-Ki Hong. 2005. Stippling Technique Based on Color Analysis. In *Adv. Multimedia Info. Proc. (PCM 2005)*. Springer, 782–793.

Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. 2015. Blue noise sampling using an SPH-based method. *ACM Trans. Graph.* 34, 6 (2015), 1–11. https://doi.org/10.1145/2816795.2818102

Henry Kang, Seungyong Lee, and Charles K. Chui. 2007. Coherent Line Drawing. In *Proc. Sym. on Non-Photorealistic Animation and Rendering (NPAR '07)*. ACM, New York, NY, USA, 43–50. https://doi.org/10.1145/1274871.1274878

Sung Ye Kim, Ross Maciejewski, Tobias Isenberg, William M. Andrews, Wei Chen, Mario Costa Sousa, and David S. Ebert. 2009. Stippling by Example. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering (NPAR '09)*. 41–50. https://doi.org/10.1145/1572614.1572622

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-Time Blue Noise. *ACM Trans. Graph.* 25, 3 (2006), 509–518. https://doi.org/10.1145/1141911.1141916

Hua Li and David Mould. 2011. Structure-preserving stippling by priority-based error diffusion. In *Proc. Graph. Interf. (GI 2011)*. Canadian Human-Computer Communications Society, 127–134.

Stuart Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theor.* 28, 2 (1982), 129–137.

Lei Ma, Yanyun Chen, Yinling Qian, and Hanqiu Sun. 2018. Incremental Voronoi Sets for Instant Stippling. *Vis. Comput.* 34, 6–8 (2018), 863–873. https://doi.org/10.1007/s00371-018-1541-7

Lei Ma, Hong Deng, Beibei Wang, Yanyun Chen, and Tamy Boubekeur. 2019. Real-Time Structure Aware Color Stippling. In *ACM SIGGRAPH 2019 Posters*. ACM, Article 92, 2 pages. https://doi.org/10.1145/3306214.3338606

Ross Maciejewski, Tobias Isenberg, William M Andrews, David S Ebert, Mario Costa Sousa, and Wei Chen. 2008. Measuring stipple aesthetics in hand-drawn and computer-generated images. *IEEE Comput. Graph. and Appl.* 28, 2 (2008), 62–74.

Domingo Martín, Germán Arroyo, M. Victoria Luzón, and Tobias Isenberg. 2010. Example-Based Stippling Using a Scale-Dependent Grayscale Process. In *Proc. Sym. on Non-Photorealistic Animation and Rendering (NPAR '10)*. ACM, 51–61. https://doi.org/10.1145/1809939.1809946

Domingo Martín, Germán Arroyo, Alejandro Rodríguez, and Tobias Isenberg. 2017. A Survey of Digital Stippling. *Comput. Graph.* 67 (2017), 24–44. https://doi.org/10.1016/j.cag.2017.05.001

Domingo Martín, Germán Arroyo, M. Victoria Luzón, and Tobias Isenberg. 2011. Scale-dependent and Example-based Grayscale Stippling. *Comput. Graph.* 35, 1 (2011), 160–174. https://doi.org/10.1016/j.cag.2010.11.006 Extended Papers from Non-Photorealistic Animation and Rendering.

David Mould. 2007. Stipple Placement Using Distance in a Weighted Graph. In *Proc. Eurographics (Computational Aesthetics'07)*. 45–52.

Wai-Man Pang, Yingge Qu, Tien-Tsin Wong, Daniel Cohen-Or, and Pheng-Ann Heng. 2008. Structure-aware halftoning. In *ACM SIGGRAPH 2008 papers*. 1–8.

Hongxing Qin, Yi Chen, Jinlong He, and Baoquan Chen. 2017. Wasserstein Blue Noise Sampling. *ACM Trans. Graph.* 36, 5, Article 168 (Oct. 2017), 13 pages. https://doi.org/10.1145/3119910

Bernhard Reinert, Tobias Ritschel, and Hans-Peter Seidel. 2013. Interactive By-Example Design of Artistic Packing Layouts. *ACM Trans. Graph.* 32, 6, Article 218 (2013), 7 pages. https://doi.org/10.1145/2508363.2508409

Thomas Schlömer and Oliver Deussen. 2011. Accurate Spectral Analysis of Two-Dimensional Point Sets. *J. Graph., GPU, & Game Tools* 15, 3 (2011), 152–160. https://doi.org/10.1080/2151237x.2011.609773

Adrian Secord. 2002. Weighted Voronoi Stippling. In *Proc. Sym. on Non-Photorealistic Animation and Rendering (NPAR '02)*. 37–43. https://doi.org/10.1145/508535.508537

Marc Spicker, Franz Hahn, Thomas Lindemeier, Dietmar Saupe, and Oliver Deussen. 2017. Quantifying Visual Abstraction Quality for Stipple Drawings. In *Proc. Sym. on Non-Photorealistic Animation and Rendering (NPAR '17)*. Article 8, 10 pages. https://doi.org/10.1145/3092919.3092923

Jianchao Tan, Jose Echevarria, and Yotam Gingold. 2018. Efficient Palette-based Decomposition and Recoloring of Images via RGBXY-space Geometry. *ACM Trans. Graph.* 37, 6, Article 262 (2018), 10 pages. https://doi.org/10.1145/3272127.3275054

Li-Yi Wei. 2010. Multi-Class Blue Noise Sampling. *ACM Trans. Graph.* 29, 4, Article 79 (2010), 8 pages. https://doi.org/10.1145/1778765.1778816

Li-Yi Wei and Rui Wang. 2011. Differential Domain Analysis for Non-Uniform Sampling. *ACM Trans. Graph.* 30, 4, Article 50 (2011), 10 pages. https://doi.org/10.1145/2010324.1964945