

# Partition of unity parametrics for texture synthesis

Jack Caron\*

David Mould†

School of Computer Science  
Carleton University

## ABSTRACT

Partition of unity parametrics (*PUPs*) are a recent framework designed for geometric modeling. We propose employing PUPs for procedural texture synthesis, taking advantage of the framework’s guarantees of high continuity and local support. Using PUPs to interpolate among data values distributed through the plane, the problem of texture synthesis can be approached from the perspective of point placement and attribute assignment. We present several alternative mechanisms for point distribution and demonstrate how the system is able to produce a variety of distinct classes of texture, including analogs to cellular texture, Perlin noise, and progressively-variant textures.

**Index Terms:** Computer Graphics [I.3.3]: Picture/Image Generation—Line and curve generation

Procedural texture has been a central aspect of content creation since the earliest days of texture. Probably the single most widely used texture synthesis tool is Perlin noise [10, 11]; however, many other techniques exist, including spot noise [14], cellular texture [17], reaction-diffusion texture [13, 16], and more. Modern approaches tend to be spectral techniques, such as wavelet noise [3] and Gabor noise [7], or nonparametric [4, 5].

At the same time, methods for point distribution are an ongoing area of research and have become very sophisticated. Methods for Poisson disk distributions abound, and recent methods such as capacity-constrained distributions [1] and differential domain distributions [15] allow high-quality sampling of arbitrary fields.

Point distributions and texture synthesis are linked by the notion of *discrete element textures* [6], where discrete textons are distributed over the plane in some fashion (in the cited work, by nonparametric synthesis based on a sample distribution). In our case, we begin with a discrete arrangement obtained by direct procedural methods but then use an interpolation scheme to create a continuous-tone texture.

This paper describes a proposal to employ partition of unity parametrics (PUPs) for texture synthesis. The PUPs framework establishes local, high-continuity interpolation, allowing texture synthesis tasks to become point distribution tasks. We have direct spatial control over texture features by manipulating point positions and values, as in conventional geometric modeling activities. We can also make use of known techniques for automatically distributing points, whether in Poisson disk [8, 9] or other distributions. PUPs-based textures are flexible and versatile, able to imitate Perlin noise, cellular textures, and other texture types. In addition, our technique is well-suited to nonstationary or progressive distributions, where the size and shape of texture elements vary spatially in controlled ways. We use the term *stationary* to refer to textures where the statistical properties do not vary over the image, and *nonstationary* when the properties vary with location.

\*e-mail: jcaron@connect.carleton.ca

†e-mail: mould@scs.carleton.ca

This paper is organized as follows. First, we discuss existing work in texture synthesis and point distribution, and we give an overview of PUPs. Next, we describe our texture synthesis algorithm: given an input point set and associated values, we create the corresponding continuous-tone texture. We next show a suite of textures generated by variations in point distribution and attribute assignment, demonstrating the versatility of the approach. Lastly, we conclude with some summarizing remarks and suggestions for possible future research directions.

## 1 RELATED WORK

The single most widely used texture synthesis primitive is Perlin noise [10, 11]. In Perlin noise, random orientations are set at lattice nodes and splines used to interpolate intermediate values. The ease of implementation, continuity, reproducibility, and ability to evaluate at arbitrary coordinates have been huge advantages for Perlin noise. Our proposed method is similar, but our use of the PUPs framework allows arbitrary interpolation functions and does not require a regular lattice; as we will see, the freedom to create an arbitrary graph allows progressive textures which would be difficult to create using Perlin noise, as well as avoiding lattice artifacts.

For biologically inspired textures such as spots and stripes, reaction-diffusion textures [13, 16] were proposed. Such textures have extremely high quality and the methods can synthesize nonstationary patterns, but have generally been deemed slow and difficult to control. Cellular texture [17] is an alternative for some animal markings such as giraffe spots, but the range of possible textures is more limited. Progressively-variant textures [19] use example-based techniques on a texton map to create textures that change gradually from one type to another, with intriguing results.

In recent years most research on texture synthesis has been based on nonparametric synthesis [5], where pixels or patches are copied in a structured way from an example texture to a destination. Such methods have achieved high quality, but have the drawback that the exemplar texture is needed. Another strand of recent work has been spectrally controlled noise such as wavelet noise [3] or Gabor noise [7]. These are texture synthesis primitives and allow a variety of textures to be created, and the proposed work is in the same spirit.

The proposed texture synthesis process is based on *partition of unity parametrics*, a recent meta-modeling framework proposed by Runions and Samavati [12]. In the version of the PUPs framework used here, a triangle mesh stores data at nodes; the values is interpolated across triangle faces using projection and normalization. Originally proposed for geometry, here we adapt it to texture synthesis; since the PUPs structure is already well suited to texture synthesis, we concentrate on showing how point distributions and point labels can be assigned in such a way as to create different types of texture. We show textures resembling Perlin noise, Worley noise, and reaction-diffusion spots, plus demonstrate how to create progressively-variant textures with controllable feature size and nonstationary structures. We discuss PUPs, and our modifications thereto, in detail in section 2.

## 2 ALGORITHM

PUPs uses a set of control points  $\mathcal{P} \subset \mathbb{R}^3$  to create an interpolated surface, which is a weighted sum of the control points. While

in PUPs the points stored no information beyond their geometry, points can be assigned additional attributes which are then interpolated: in our case, we store an orientation, used to adjust the interpolation weight, and a value, typically a scalar intensity value or a vector color value. Since we are creating 2D textures, our control points lie in  $\mathbb{R}^2$ ; without loss of generality, we created textures where  $\mathcal{P} \in [-1; 1]^2$ .

At a point in space  $u$ , the texture value is a weighted sum of the values at the nearby control points. The weight is constructed with reference to the edges of a graph connecting the control points; in our case, we used the Delaunay triangulation [2] (see fig. 2). A control point  $c$  is said to have a set of *axes*,  $\alpha_i$ , where each axis is an edge originating at  $c$  and terminating at its neighbour. The point  $u$  is projected onto each axis, and the projected length is used to determine an axial weight,  $A_{\alpha r}$ . The axial weights are given by cubic polynomial interpolants, decreasing from a maximum of 1 (exactly on the control point) to a minimum of zero (beyond the axis length); use of the cubic allows zero gradient at both ends, preventing discontinuities :

$$A_{\alpha r}(p) = (1 - p)^2(1 + 2p) \quad (1)$$

Finally, a given control point has a weight  $W_i$  (see fig. 1a), which is the product of the axial weights:

$$W_i(u) = \prod_{r=0}^p A_{\alpha r}(proj_{\alpha r}(u)). \quad (2)$$

Note that the projection operation  $proj_{\alpha r}$  returns a normalized projection in the range 0-1; the projected length is given as a fraction of the axis length, and clamped so that points beyond the axis end project to 1, while points behind the axis project to 0. It's also possible to control the smoothness of the intensity interpolation by changing the length of the axis (see fig. 4). Longer axes overlap and produce smoother outcomes, albeit with approximating interpolations rather than exact ones.

The texture value for a point in space is then the weighted sum of the attribute values at the nearby control points  $\mathcal{P}' \subseteq \mathcal{P}$ , normalized by the weights:

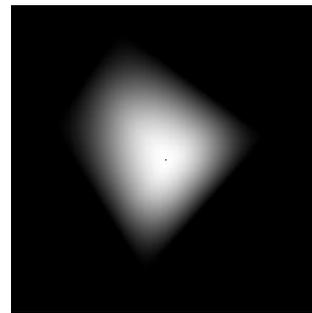
$$T(u) = \frac{\sum_{i=0}^{|\mathcal{P}'|} T_i W_i(u)}{\sum_{i=0}^{|\mathcal{P}'|} W_i(u)} \quad (3)$$

where  $T_i$  is a value, a scalar intensity or a color, stored with point  $p_i \in \mathcal{P}'$ .

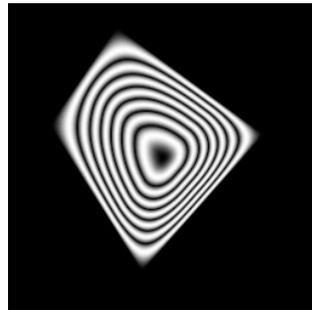
We can further adjust the weights using an *orientation* attribute associated with each control point. Orientation is a unit vector in  $\mathbb{R}^3$ , used to shift the weight of its point towards a preferred direction; call the orientation  $v$ . Suppose we are computing the shifted weight for a point  $p \in \mathbb{R}^2$  with respect to a control point at  $c \in \mathbb{R}^2$ . We first compute  $\eta = (p - c) \cdot v / |p - c|$ . Then we compute the shifted weight  $w'$ , using equation 4, where the original weight  $w \in [0; 1]$  is the weight of the point from equation 2 and  $\varepsilon \in [0; 1]$  is a parameter governing the strength of the shift and  $K = \varepsilon \eta$  (by default,  $\varepsilon = 0.5$ ). In equation 4, the exponent is in  $]0; 1]$  if  $K \geq 0$ , which increases the weight, and greater than 1 otherwise, which decreases the weight. The asymmetry of the exponent in the two parts of equation 4 allows more drastic changes on control points facing away from  $v$ . Figure 3 shows the effects of an orientation on the weight distribution.

$$w' = \begin{cases} w^{1.0-K} & \text{if } K \geq 0 \\ w^{\frac{1.0}{1.0+K}} & \text{otherwise} \end{cases} \quad (4)$$

We can adapt the intensity interpolation to produce basic cellular textures, similar to those of Worley [17]. Each control point is given a *label* attribute; pixels are given the label of the control point with the highest weight. Because of the way weights are computed, the



(a)



(b)

Figure 1: The (a) weight of a point (*brighter means more weight*) and (b) isovalues in its scalar field.

resulting cells approximate but are not exactly the same as Voronoi cells; in particular, they are not bounded by straight edges. We consider the appearance of the cells to be more organic and lively than Voronoi cells. Using orientations, we are able to curve the cell boundaries even further; this is illustrated in Figure 5.

### 3 TEXTURE VARIATIONS

In this section, we present algorithms to distribute control points and assign their attributes so as to create a wide variety of textures.

#### 3.1 Noise

Using the basic interpolation scheme over random data, we can create a smooth scalar field similar to Perlin noise. The details of the process are as follows. First, we place points over the plane in a Poisson disc distribution. Then, we randomly assign an intensity value to each point. We create a graph over the distribution by finding the Delaunay triangulation. Then, we construct the noise field by taking the PUPs interpolation as described in the previous section. Four octaves of texture akin to Perlin noise are shown in Figure 7.

Multiscale Perlin noise can be created by summing octaves of with different point densities, and examples are shown in Figure 6. These results use the equation

$$T(x) = \sum_{i=0}^{i=3} P_i(x) 2^{-i} \quad (5)$$

where  $P_i$  are Perlin-like textures with Poisson disk distributions of control points; the minimum interpoint distance is  $0.1 \times 2^{-i}$ . Note that we used separate textures for the different octaves, but because our textures are tileable, we could have used a single texture and changed the scale.

We can also reproduce Perlin turbulence [10], applying the folding transform  $T'(x) = 2|T(x) - 0.5|$  to each octave; intensity is in the range 0 to 1. The foldings of the individual octaves are shown

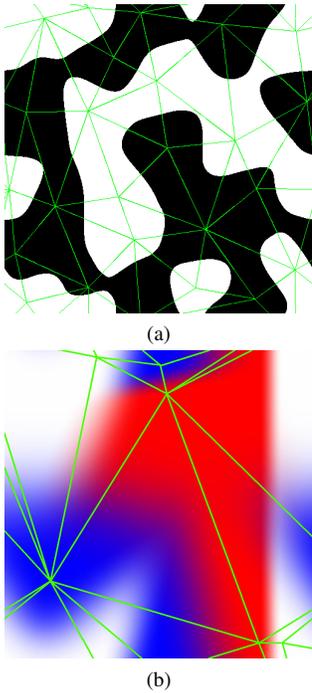


Figure 2: Delaunay triangulations of  $\mathcal{P}$ .

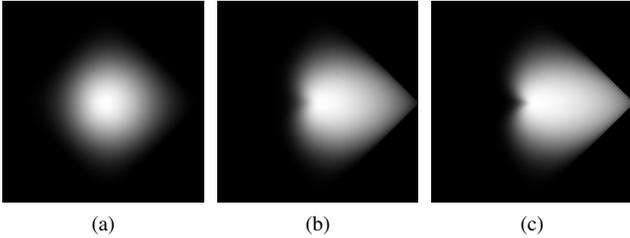


Figure 3: The weight shifting as the orientation is (a) almost aligned with the  $z$ -axis to be (c) almost aligned with the  $x$ -axis;  $\epsilon = 0.1$ .

in Figure 7 and the resulting turbulence texture appears in Figure 8. Unlike regular Perlin turbulence, there are no lattice artifacts: the Poisson disc control point distribution frees us from any such defects.

### 3.2 Labels

We previously discussed how to create cellular textures by organizing points into groups with common labels. Here, we demonstrate some of the different textures that can be achieved with this process.

Figure 9a shows a simple Voronoi-like cellular texture, obtained by assigning each point to a separate group. It is not identical to the Voronoi diagram; the use of orientation in the weight calculation disturbs the Voronoi boundaries. More interesting possibilities emerge when there are multiple points per group: Figure 9b shows a texture created by grouping points into clusters of three or four points. In this case, the regions are even further from Voronoi regions.

Multiscale cellular textures are another straightforward possibility. Two examples are shown in Figure 10. To obtain these, we created 4 layers of cellular textures of increasing density and summed them; higher densities have lower amplitude, as in multiscale Perlin noise.

An entirely different type of structure can be created by assigning

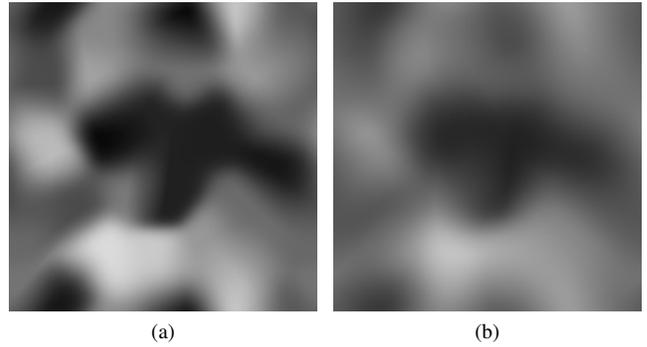


Figure 4: The same point set with (a) normal axis length and (b) axis twice as long.

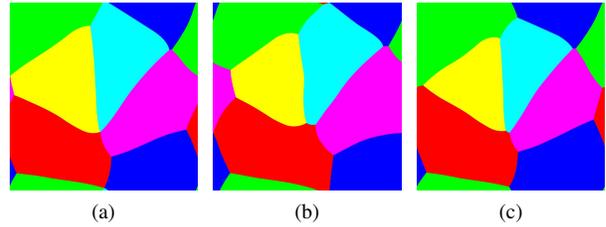


Figure 5: An (a) original point set, (b) the same point set with orientation all pointing to the right and (c) with all orientation pointing away from the center.

groups binary labels. In Figure 11 we see a texture resembling a stereotypical cow pattern, obtained by randomly assigning labels to points. The simplicity and unpredictability of this binary texture give it a distinctly organic character.

A more elaborate example with two groups is called the *space giraffe*, following the reaction-diffusion texture by Witkin and Kass [16]. The principal objective in crafting this texture was to control the irregular shape of the boundaries of the cells; a secondary objective was to prevent two cells from merging. The space giraffe texture is shown in Figure 12; we next describe the process used to create it.

We distribute cell centres with a Poisson disk distribution. For each centre, we then find its nearest neighbour; the cell's radius  $r$  is taken to be half the distance to the nearest neighbour. We then add further points surrounding the cell centre, as follows: choose a random number  $n$ , say between 10 and 20, and place pairs of points at angular intervals of  $2\pi/n$ . The points are placed at radii randomly chosen from the range 0 to  $r$ . At each angle, there is an inner point (smaller radius) and an outer point (larger radius); assign to the inner point the spot label and to the outer point the background label. Create a new triangulation on this new point set and apply the cellular texture generation process: the result is the space giraffe pattern, examples of which are shown in Figure 12.

More elaborate structures, and in particular nonstationary structures, can be obtained by using a reference image to decide group membership. In this variant, we distribute points using a Poisson disk distribution and for each of these points, we check the intensity of the corresponding pixel in the image. We select a random value, and if that value is less than the reference intensity, we assign that point to the white group, otherwise to the black group. Figure 13 shows images created with the approach; with this reference image, the only gray regions are near the region boundaries, so we get a few variations on the input texture with different feature sizes. Note that for this process to be effective, the reference image must be in grayscale.

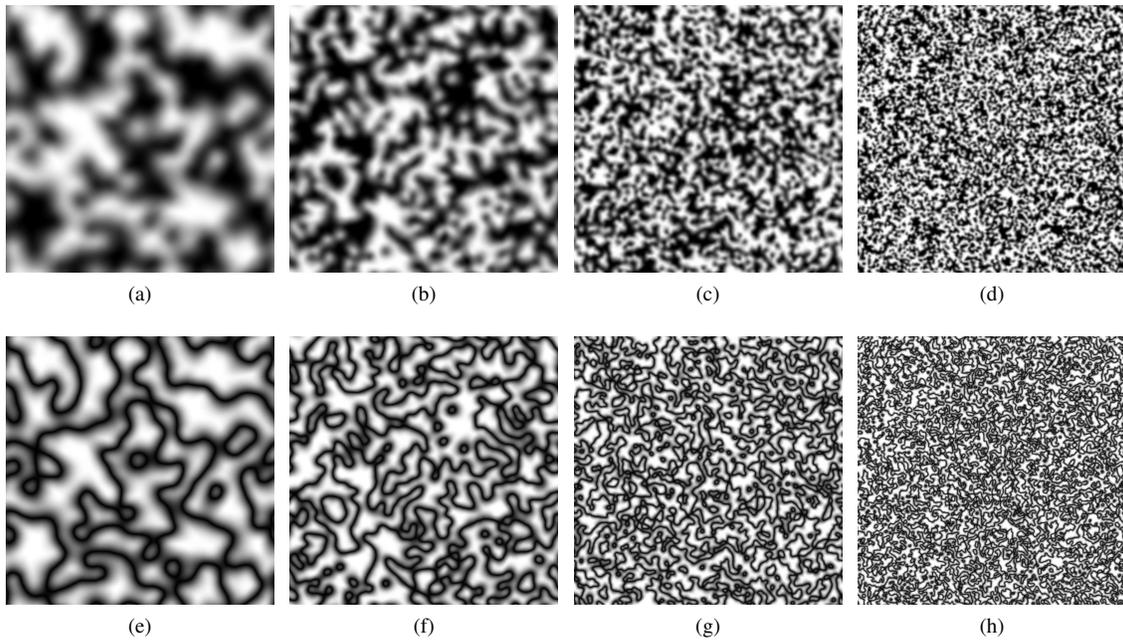


Figure 7: Multiple scales of Perlin-noise type textures, before and after folding.

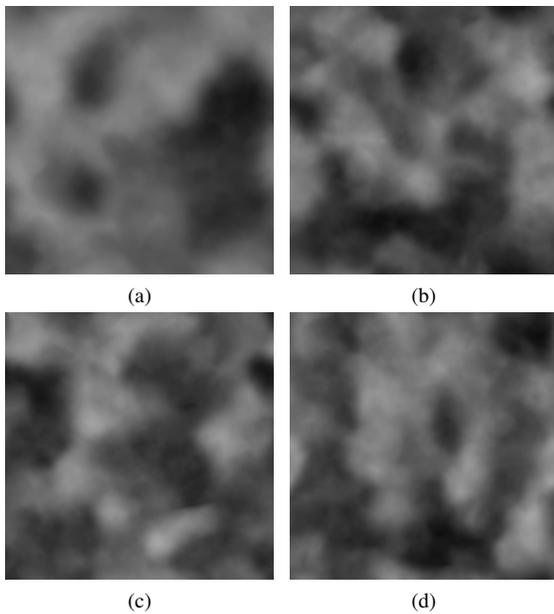


Figure 6: Multiscale Perlin noise textures (four octaves).

In another variant, we propose to use an iterative version of the previous process. The only difference is that we use the intensity interpolation to create the textures, ensuring that there are regions of intermediate intensity near region boundaries. At each iteration, we double the point density, ultimately creating a fractal boundary layer; four iterations are shown in Figure 14. Restricting the regions where control point labels are chosen randomly is a powerful technique, producing results that would not be straightforward to make using other existing texture synthesis techniques.

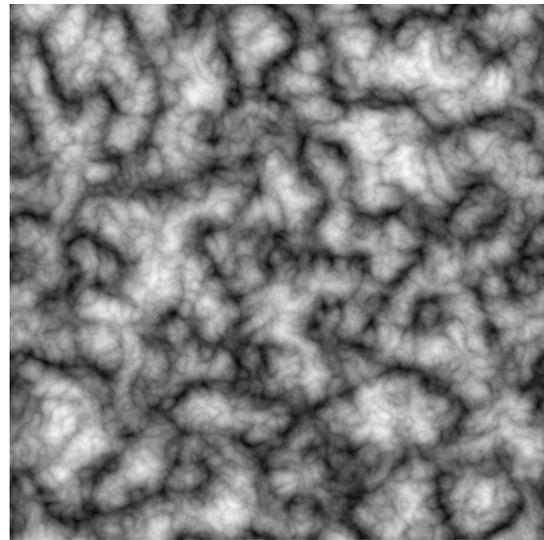


Figure 8: Perlin turbulence: sum of folded textures from fig. 7

### 3.3 Progression

One of the main strengths of our method is its ability to use non-stationary spatial distributions of points or spatially aware label assignments to produce progressively-varying textures in a systematic way. The simplest example is that of Figure 15. This image shows two examples of progressive cow textures, where the point density becomes greater as you move to the right. The same random binary labeling strategy is employed. The resulting texture shows an obvious but continuous decrease in feature size as the point density increases.

A more elaborate example appears in Figure 16, where again the point density increases left to right. Here, we have used the space giraffe process on an initial distribution. The method is robust enough to produce irregular spots smoothly varying in size.

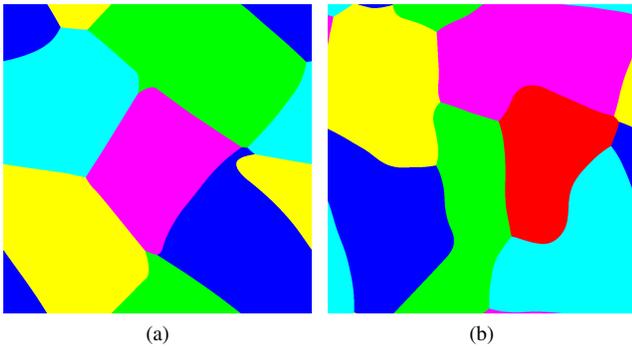


Figure 9: Cells made of (a) one point per cell and (b) a few points per cell.

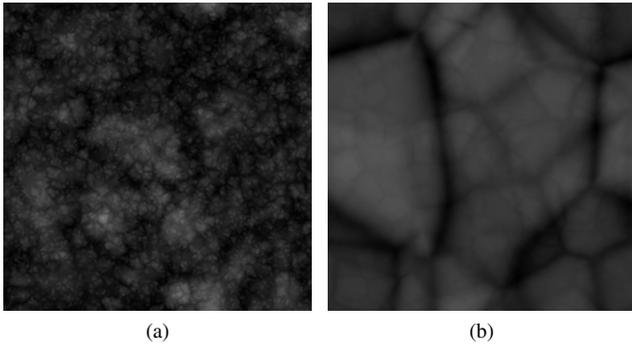


Figure 10: Noises with cells.

Changing the point distribution is one of the two mechanisms for synthesizing nonstationary textures. Even with a stationary distribution of control points, we can create nonstationary textures by applying labels using logic that depends on spatial location. For example, we can create a texture ramp by assigning black or white labels with a probability that depends on the point’s x-coordinate. See Figure 17 for an example. Of course, other distributions are also possible: this example only hints at the range of possibilities.

#### 4 DISCUSSION

The images in the previous section demonstrate some of the textures that can be generated using this method. We demonstrated analogs to Perlin noise, Worley noise, some biological structures, and some multiscale and progressive structures. The versatility of the method

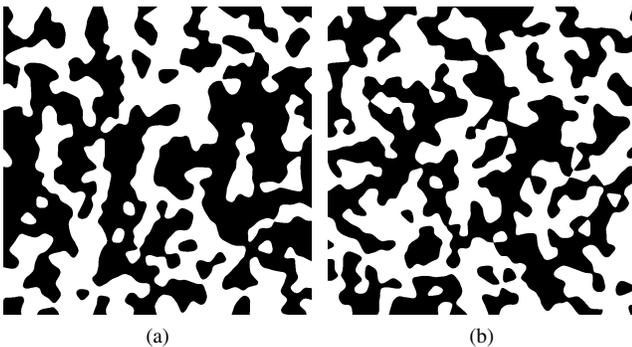


Figure 11: Two “cow texture” examples: Poisson disc distribution of control points with random assignment of binary labels.

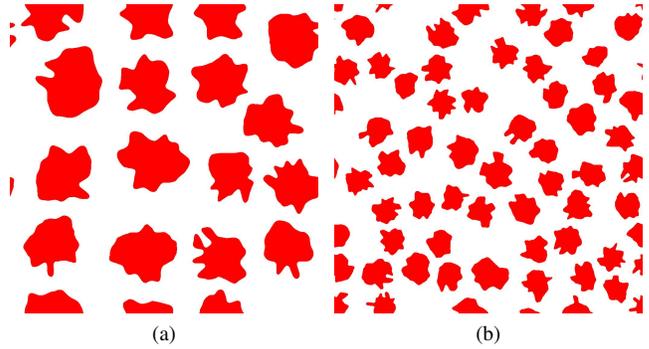


Figure 12: Space giraffe patterns.

is one of its chief strengths.

The PUPs framework allows arbitrary control net topology while maintaining local support of feature interpolation. We are able to exploit this in making progressively variant textures such as the cow textures in Figure 11, where the feature density varies spatially. An alternative approach for building progressive textures is to vary the attributes in a systematic way, e.g., as seen in Figure 15. The control points provide discrete locations where the attributes are set, and the PUPs interpolation propagates the values to the rest of the plane.

Figure 14 shows a process where the distribution of intensities on the control points are distributed according to the intensity in the previous image at the same location. At each iteration, more points are used (in our case, we doubled the number of points used in the previous image) allowing the creation of a sharper and more detailed image each time.

The approach we presented has some drawbacks, however. We have not much investigated color; while color can be interpolated using the same process we used to interpolate intensity or group membership, color assignment is a more challenging problem. In our current prototype implementation, optimized for ease of experimentation rather than for speed, rendering is slow: for a typical case, a  $1000 \times 1000$  texture with 700 control points, rendering requires approximately 30 seconds on a Quad-Core 1.3 MHz computer with 8 GB RAM.

We have cited the control net as an advantage of our method, but it has some disadvantages also. We used the Delaunay triangulation to provide our graph; small changes in the position of points can cause sudden changes in network topology, with correspondingly sudden changes in the texture, albeit only local. For static textures this is not an issue, but for manual editing of textures, or for animated textures, this would be a concern. Another drawback is the need to match the control net resolution to the feature resolution; arbitrary topology and local support allow us to do the matching adaptively, but very high-frequency features are still costly. Exploring different weight functions – for example, functions with oscillatory behaviour – might reveal a way to break that limitation.

#### 5 CONCLUSION AND FUTURE WORK

In this paper we described an adaptation of the “partition of unity parametrics” metamodelling framework for texture synthesis. We showed how to create textures reminiscent of classic textures such as Perlin noise and cellular texture, as well as novel textures including intricate patterns with multiscale boundaries. Because the PUPs interpolation method can propagate attribute values to the space between the control points, we can concentrate on procedurally distributing control points (for example, according to a Poisson distribution) and on assigning attributes to the control points (for example, randomly, according to spatial location or graph connectivity, or according to a reference image).

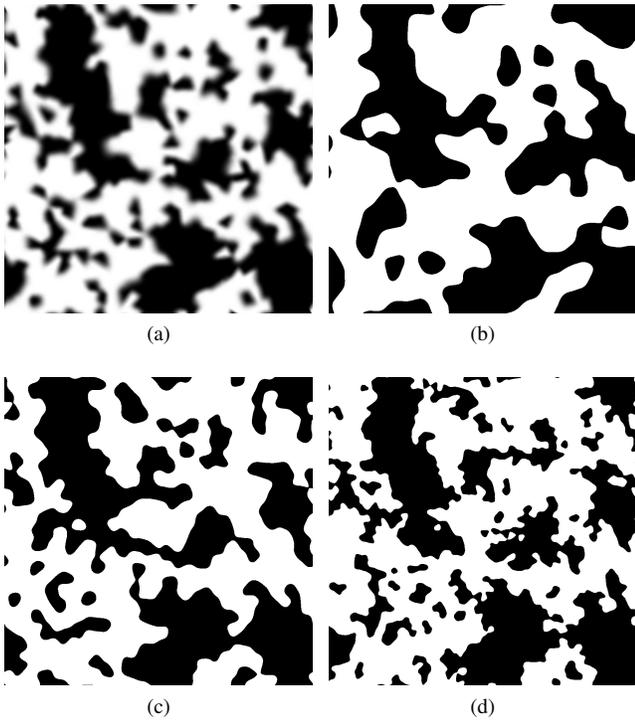


Figure 13: A reference image (*image (a)*) used to assign points to groups in images with different point densities (*images (b) (c) & (d)*).

The method is able to generate stationary textures, but is not restricted to doing so. Progressive textures, where the characteristics vary smoothly from one image location to another, are possible. Also, use of reference images allows us to create highly irregular textures; we also demonstrated an iterative point placement and attribute assignment process, producing fractal boundaries reminiscent of coastlines. The ability to mix large-scale and small-scale features is extremely powerful.

Opportunities for future work are numerous. The rendering speed can be dramatically improved with a GPU implementation. Further investigation along the lines we have already begun will reveal more types of patterns to be described procedurally. We are interested in trying other triangulations or even non-planar control net topologies, such as the Yao graph [18], and in performing direct synthesis over polygonal meshes.

Animated textures are a direction to which this method seems especially suited. The motion of the texture can be governed by the motion of the underlying discrete structure. The main difficulty here lies in smoothing out sudden changes in the texture arising from sudden changes in the control net topology.

Finally, we are optimistic about the usability of the approach for manual texture creation and editing. Building a tool for editing PUPs-based texture is another area of future work.

#### ACKNOWLEDGEMENTS

We would like to thank Adam Runions and Faramarz Samavati for helpful discussions about PUPs. Financial support for this work was provided by NSERC, GRAND, and Carleton University.

#### REFERENCES

- [1] M. Balzer, T. Schlömer, and O. Deussen. Capacity-constrained point distributions: a variant of Lloyd's method. *ACM Trans. Graph.*, 28(3):86:1–86:8, July 2009.

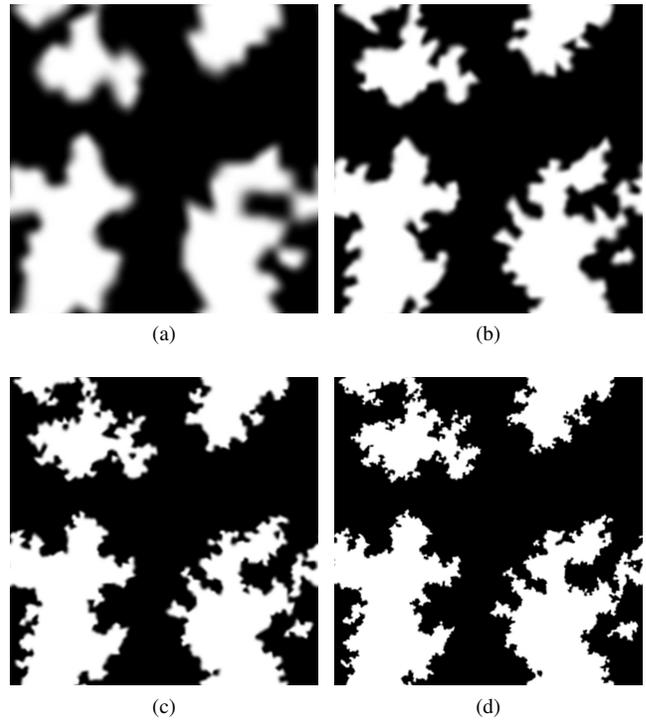


Figure 14: Iterative process to assign points to groups.

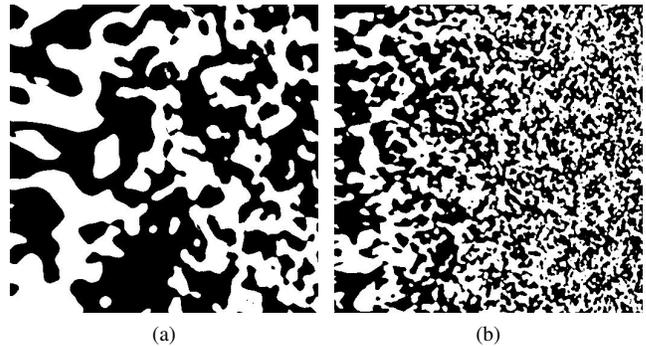


Figure 15: Progressive textures.

- [2] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.
- [3] R. L. Cook and T. DeRose. Wavelet noise. *ACM Trans. Graph.*, 24(3):803–811, July 2005.
- [4] A. A. Efros and W. T. Freeman. Image Quilting for Texture Synthesis and Transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.
- [5] A. A. Efros and T. K. Leung. Texture Synthesis by Non-Parametric Sampling. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2, ICCV '99*, pages 1033–, Washington, DC, USA, 1999. IEEE Computer Society.
- [6] T. Ijiri, R. Mech, T. Igarashi, and G. S. P. Miller. An Example-based Procedural System for Element Arrangement. *Comput. Graph. Forum*, 27(2):429–436, 2008.
- [7] A. Lagae, S. Lefebvre, G. Drettakis, and P. Dutré. Procedural noise using sparse Gabor convolution. *ACM Trans. Graph.*, 28(3):54:1–54:10, July 2009.
- [8] M. McCool and E. Fiume. Hierarchical Poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92*,

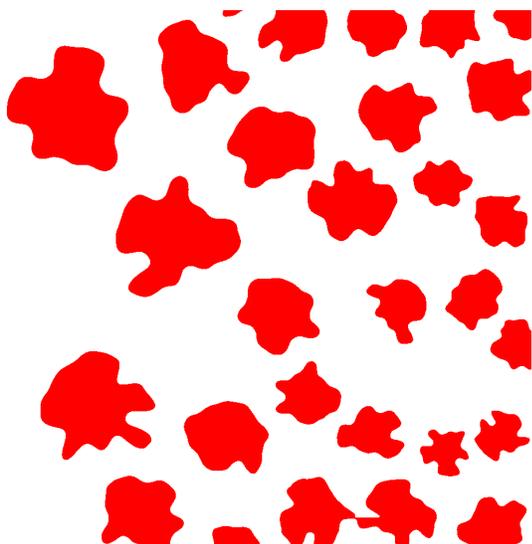


Figure 16: A progressive space giraffe.

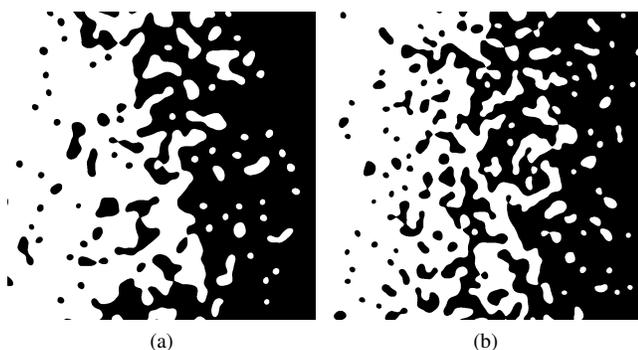


Figure 17: Points assigned to groups according to their location.

- pages 94–105, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.
- [9] D. P. Mitchell. Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph.*, 21(4):65–72, Aug. 1987.
- [10] K. Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985.
- [11] K. Perlin. Improving noise. *ACM Trans. Graph.*, 21(3):681–682, July 2002.
- [12] A. Runions and F. F. Samavati. Partition of unity parametrics: a framework for meta-modeling. *The Visual Computer*, 27(6-8):495–505, 2011.
- [13] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 289–298, New York, NY, USA, 1991. ACM.
- [14] J. J. van Wijk. Spot noise texture synthesis for data visualization. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 309–318, New York, NY, USA, 1991. ACM.
- [15] L.-Y. Wei and R. Wang. Differential domain analysis for non-uniform sampling. *ACM Trans. Graph.*, 30(4):50:1–50:10, Aug. 2011.
- [16] A. Witkin and M. Kass. Reaction-diffusion textures. *SIGGRAPH Comput. Graph.*, 25(4):299–308, July 1991.
- [17] S. Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 291–294, New York, NY, USA, 1996. ACM.
- [18] A. C.-C. Yao. On Constructing Minimum Spanning Trees in  $k$ -

Dimensional Spaces and Related Problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

- [19] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph.*, 22(3):295–302, July 2003.