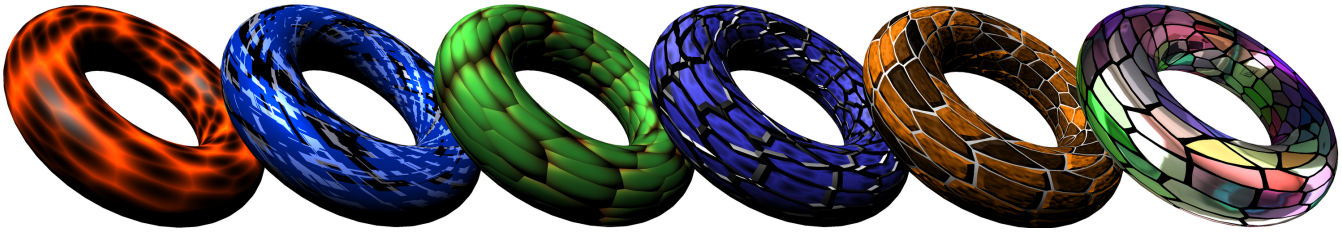
# Worley Cellular Textures in Sh

Bryan Chan Michael McCool

Computer Graphics Lab, School of Computer Science, University of Waterloo



(a) $F_1$, $\ell^2$ (b) $F_4$, $\ell^1$ (c) $F_1$, $\ell^2$ (d) $F_2 - F_1$, $\ell^\infty$ (e) $F_2 - F_1$, $\ell^2$ (f) $F_2 - F_1$, $\ell^2$

Procedural texturing is a memory-efficient method for improving surface realism. One of the most interesting such techniques is Worley's cellular texture basis function [Worley 1996]. We have adapted the original algorithm to a single-pass pixel shader on current GPUs that lets us interactively manipulate parameters and generate dynamic textures.

## 1 Adapting the Algorithm

We adapted the Worley basis functions to run efficiently on the GPU while maintaining the flavor of the original technique. To generate the cellular texture basis functions, we need to randomly distribute a set of *feature points*. Then the first basis, $F_1(p)$ is defined as the distance from $p$ to the closest feature point. In the same manner, we can define $F_n$ to be the distance to the $n^{th}$-nearest feature point.

To generate feature points, the original version placed points using a Poisson distribution. We chose instead to use a jittered grid, with exactly one feature point in each cell. Although this reduces the isotropy of the result, it guarantees a limited search radius. For example, to find the four nearest feature points to a point $p$, we only need to search the cell holding $p$ and the immediately adjacent cells. This gives a maximum of $3^d$ cells when working in $\Re^d$. Since this is a small number for low dimensions, we can generate feature points near $p$ on the fly, without precomputation.

With nearby feature points, we need the k-nearest neighbors to calculate functions up to $F_k$. There are many available methods, but with such a small number of points, the fastest is a simple sort. We took advantage of the parallel nature of fragment shader units to build a comparator sorting network.

Our implementation uses the Sh metaprogramming language [McCool et al. 2002], an open-source C++ matrix/vector library that can record sequences of operations as GPU shader programs. For more information see the website [McCool and Du Toit 2003] and book [McCool and Du Toit 2004].

To simplify the code for storing and sorting feature points, we added support in Sh for tuples with more than 4 components. These long tuples expose the same operations as normal tuples, so this allows us to specify a comparator network simply by using swizzles (permutations on tuple components).

## 2 Examples

A variety of images produced by our implementation are shown above illustrating color mappings in (a), (b), bump mapping using basis function gradients in (c), (d), and tile coloring using random cellnoise in (e). The stain glass example in (f) uses shader algebra operations [McCool et al. 2004] to perturb the parameters of an existing glass shader.

On a Geforce 6800 at 512×512 resolution, all of the examples ran interactively, ranging from 14 frames per second for the stone tile shader to over 200fps for the simple color mapping.

## 3 Conclusion

We have demonstrated that Worley basis functions can be implemented concisely and efficiently using Sh. However, many unexplored avenues remain.

One of the problems with the current approach is the quality of the noise used to jitter feature points. Texture-based noise requires expensive lookups, so we are working towards a procedural noise function with acceptable randomness. An integer hash function implemented in hardware would be equally welcome.

## References

McCool, M., and Du Toit, S. 2003. Sh web site. *http://libsh.sourceforge.net*.

McCool, M., and Du Toit, S. 2004. *Metaprogramming GPUs with Sh*. AK Peters, Ltd.

McCool, M., Qin, Z., and Popa, T. S. 2002. Shader metaprogramming. In *Proc. Graphics Hardware 2002*, Eurographics Association, 57–68.

McCool, M., Du Toit, S., Popa, T., Chan, B., and Moule, K. 2004. Shader algebra. *ACM Trans. on Graphics (Proc. SIGGRAPH)*.

Worley, S. 1996. A cellular texture basis function. In *Proc. of SIGGRAPH 1996*, ACM Press, 291–294.