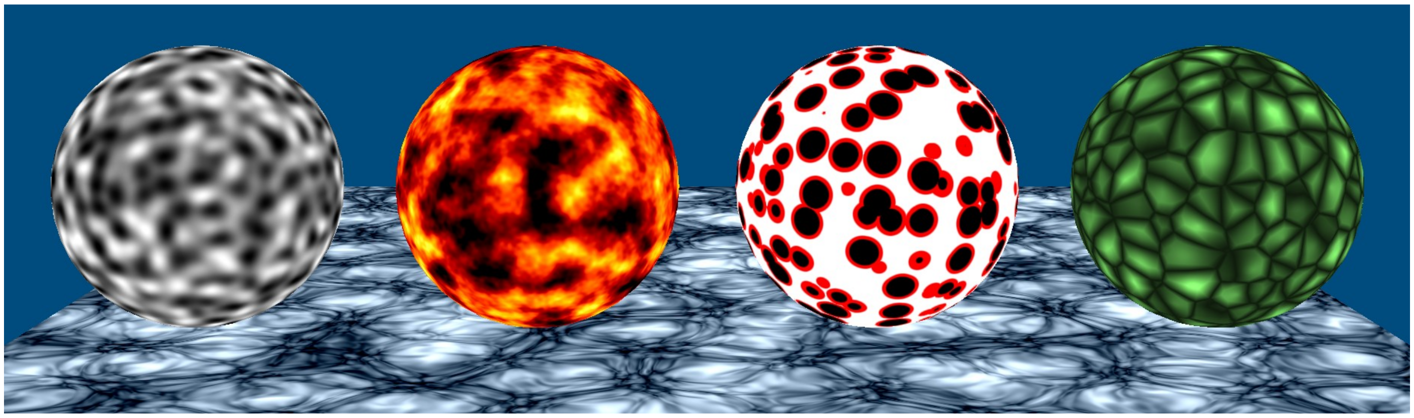
Procedural GPU Shading Ready for Use

Stefan Gustavson[1], Linköping University, Sweden and Ian McEwan[2], Ashima Research, USA



A selection of procedural patterns, generated entirely on the GPU without any texture accesses. The left two spheres use Perlin simplex noise by itself and in a fractal sum. The right two spheres use Worley cellular noise in different ways. The plane at the bottom shows Perlin and Neyret's "flow noise", with rotating gradients. All these shaders are animated, have analytic derivatives that are easy to compute, and are fast enough to be considered for routine use even on previous generation GPU hardware.

Procedural patterns have been a staple of software shading for decades. Perlin noise revolutionized the industry and won an Academy award for technical achievement. With the comparably recent introduction of programmable shading in GPU architectures, hardware accelerated procedural shading is now very straightforward and deserves to be considered a lot more than what seems to be current practice.

Very recent work by us, submitted for publication elsewhere, has provided open source GLSL implementations of many classic noise algorithms in the form of fast, self-contained functions [1]. To make the case for procedural shading, we will show live demos using this work of ours to create visually rich surface patterns. A cross platform demo, with full source code, to render the animated scene in the figure above is on:
http://www.itn.liu.se/~stegu/gpunoise/
More examples and demos will be presented during the talk. If you have a GLSL-capable laptop, bring it along.

Procedural shading has an inherent flexibility that cannot be matched by sampled texture images. The initial effort of writing a good procedural shader is more complicated than drawing a texture or editing a photographic image to suit your needs, but with procedural shaders, the pattern and the colors can be varied with a simple change of parameters. This allows extensive re-use in many circumstances, as well as fine tuning or even complete overhauls of the surface appearance very late in a production process. A procedural pattern allows for easy generation of a corresponding bump or normal map. Procedural patterns can be rendered at an arbitrary resolution without jagged edges or blurring in close-up views, which is particularly useful for real time applications where the viewpoint is often unrestricted. There are no problems with periodic tiling artifacts when a procedural texture is applied to a large area. Procedural shading also lifts the memory and tiling restrictions for 3D textures and animated patterns, and enables analytic anisotropic antialiasing.

While all these advantages have made procedural shading popular for offline rendering, real time applications have not yet adopted this practice. One obvious reason is that the GPU is a limited resource, and quality often has to be sacrificed for performance. However, recent developments have given us massive computing power even on typical consumer level GPUs, and with the massively parallel architectures that are employed, memory access has become a major bottleneck. A modern GPU has an abundance of texture units and uses caching strategies to reduce the number of accesses to global memory, but many real time applications now have an imbalance between texture bandwidth and processing bandwidth, to the extent where you can sometimes consider that "cycles are free", in the meaning that if there is a lot of texture access going on, computing instructions to augment the image based textures with procedural elements can often be executed in parallel to memory reads without any slowdown at all. Even on low end hardware for mobile devices, texture download and texture access both come at a considerable cost which can be alleviated by procedural texturing.

Procedural methods are not limited to fragment shading. With the ever increasing complexity of real time geometry and the recent introduction of GPU-hosted tesselation, tasks like surface displacements and ambient animations are best performed on the GPU. The tight interaction between procedural displacement shaders and procedural surface shaders have proven very fruitful for creating complex and impressive visuals in offline shading environments, and there is no reason to assume that real time shading would be fundamentally different in that respect.

For all these reasons, now is a good time to consider using more of the GPU power for procedural texturing.

[1] http://github.com/ashima/webgl-noise

1) stefan.gustavson@liu.se, 2) ijm@ashimaarts.com