










This branch is 20 commits ahead of ashima:master.

 Contribute

 stegu Link formatting	278b97f on Jan 8	 137 commits
 benchmark	Created warning about old code	6 years ago
 demo	Created warning about old code	6 years ago
 src	Merge pull request #9 from ashima/master	8 months ago
 webdemo	Update periodic.html	12 months ago
 LICENSE	Update LICENSE	6 years ago
 README.md	Link formatting	last month
 _config.yml	Set theme jekyll-theme-minimal	12 months ago

 README.md

Noise for GLSL 1.20

Note that newer, faster and more versatile versions of 2-D and 3-D simplex noise were published in 2022. They can be found on <https://github.com/stegu/psrdnoise/>. This repository is still relevant, though, because it has 4-D noise, classic Perlin noise and Worley noise, neither of which are in the new repository.

Many other noise implementations make heavy use of a texture lookup table and are texture bandwidth limited. The noise functions in this library, however, are completely self contained with no dependency on external data. While not quite as fast as texture-based implementations on typical current desktop GPUs, they are more scalable to massive parallelism and much more convenient to use, and they can make good use of unused ALU resources when run concurrently with a typical texture-intensive rendering.

2016-05-13: Ashima Arts is now defunct as a company, so I cloned this repository from ashima/webgl-noise to here.

This repository contains GLSL source code for Perlin noise in 2D, 3D and 4D, both the modern simplex versions and the classic versions, including periodic noise similar to the `pnnoise()` function in RenderMan SL. The functions have very good performance and no dependencies on external data. The code is open source, licensed under the terms of the OSI-approved and very permissive MIT license. For licensing details, please refer to the LICENSE file in the repository.

- Simplex noise functions are (C) Ashima Arts and Stefan Gustavson
- Classic noise functions are (C) Stefan Gustavson
- Cellular ("Worley") noise functions are (C) Stefan Gustavson
- The "psrdnoise" functions are (C) Stefan Gustavson

The simplex noise functions follow Ken Perlin's original idea, more clearly explained in Stefan Gustavson's paper "Simplex noise demystified" <http://www.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf> but without using any uniform arrays or texture lookups.

A scientific paper about this was accepted for publication in JGT, (Journal of Graphics Tools), and after a long delay it finally appeared in print in June 2012: <http://dx.doi.org/10.1080/2151237X.2012.649621>. You are also welcome to read our preprint on <http://www.itn.liu.se/~stegu/jgt2012/>, which is not locked up behind a paywall.

You can of course use the functions without reading any part of the article. They require no setup or external data, just the GLSL source provided here, and they should work on any current OpenGL platform, including OpenGL 2.1, OpenGL 3.x and 4.x, OpenGL ES 2.x and WebGL 1.0. They will also work in limited vertex shader environments where texture lookup is not available. In WebGL and OpenGL ES, you will need to use high precision (`precision highp float`) for any of the functions to work.

NOTE: You don't need a super fast GPU to use these functions. They are useful even on low end hardware, old hardware and embedded OpenGL ES hardware with low performance.

Live demo!

A WebGL live demo of many of the functions provided are here: <https://stegu.github.io/webgl-noise/webdemo/>

Dead demo

The repository also contains an old and, sadly, long outdated and abandoned C code for a cross-platform benchmark and a demo. To compile and run these programs, you need a desktop OS like Linux, Windows, MacOSX or some flavor of Unix with OpenGL support, and the GLFW library (<http://glfw.sourceforge.net>). Makefiles are provided for Linux, Windows and MacOS X. The demo is many years old and might be tricky to compile because it uses GLFW 2.x and old school OpenGL 2.1 calls (because that's what MacOS X required back in 2011), and I would advise against resurrecting the demo and benchmarking code. The CPU code is old and stale. However, the GLSL noise functions as such have aged well, and they are perfectly useful even in OpenGL 4.x, not only in WebGL.

List of functions

The GLSL source files and the noise functions that were originally published here were:

- **noise2D.glsl** - 2D simplex noise `float snoise(vec2 x)`. Very fast, very compact code.
- **noise3D.glsl** - 3D simplex noise `float snoise(vec3 x)`. Fast, compact code.
- **noise4D.glsl** - 4D simplex noise `float snoise(vec4 x)`. Reasonably fast, reasonably compact code.
- **classicnoise2D.glsl** - 2D classic Perlin noise `float cnoise(vec2 x)`, and a periodic variant `float pnnoise(vec2 x, vec2 period)`. Fast and compact code, only slightly slower than 2D simplex noise.
- **classicnoise3D.glsl** - 3D classic Perlin noise `float cnoise(vec3 x)`, and a periodic variant `float pnnoise(vec3 x, vec3 period)`. Reasonably fast but not very compact code. Not quite as fast as 3D simplex noise.
- **classicnoise4D.glsl** - 4D classic Perlin noise `float cnoise(vec4 x)`, and a periodic variant `float pnnoise(vec4 x, vec4 period)`. Only about half as fast as 4D simplex noise, and rather bloated code, but still useful.

The versions here are further optimized compared to the versions in the published JGT paper. Not by a lot, but they are somewhat faster still.

All the noise functions return values in the range -1 to 1, with a more or less bell-shaped distribution around a zero mean.

N-D return types

All classic and simplex noise functions above return a single float. For the equivalents of the GLSL native functions `noise2()`, `noise3()` and `noise4()` that are still unimplemented on every relevant platform, you need to call these noise functions several times with different constant offsets for the arguments, e.g. `vec3(snoise(P), snoise(P+17.0), snoise(P-43.0))`. People have different opinions on whether these offsets should be integers for the classic noise functions to match the spacing of the zeroes, so we have left that for you to decide for yourself. For most applications, the exact offsets don't really matter as long as they are not too small or too close to the noise lattice period (289 in our implementation).

Other noise implementations

The motivation for these functions not using any textures or arrays is ease of use, self-sufficiency and scalability for massively parallel execution. Noise is seldom used by itself, and these functions can make use of the often untapped ALU resources when run concurrently with traditional texture-intensive real time rendering tasks. However, if you are generating noise by itself and using a lot of it, the texture-intensive noise implementations this work was based on are still up to twice as fast on current generation desktop GPU hardware with lots of texture units and smart texture caching. You can find the original release of the old noise code on <http://www.itn.liu.se/~stegu/simplexnoise/GLSL-noise.zip> (Windows binary, Windows-dependent source code using legacy OpenGL). A cross platform benchmark comparing it to this new version is on <http://www.itn.liu.se/~stegu/simplexnoise/GLSL-noise-vs-noise.zip>. For the convenience of Windows users, a version with a precompiled Windows binary is in <http://www.itn.liu.se/~stegu/simplexnoise/GLSL-noise-vs-noise-Win32.zip>. Note that the demo is more than a decade old and uses long deprecated OpenGL functions on the CPU side. The GPU code is still useful, though.

Cellular (Worley) noise in GLSL

A few GLSL 1.20 compatible implementations of cellular noise ("Worley noise") are available in this repository, and also on <http://www.itn.liu.se/~stegu/GLSL-cellular/>. It is a straightforward but non-trivial implementation of previous ideas from software procedural shading, using the ideas for pseudo-random permutation from the computational Perlin noise originally presented here, with some optional shortcuts for situations where speed is more important than accuracy.

Tiling 2-D simplex noise with analytic derivatives and rotating gradients

In May 2016, another newly written variation on 2-D simplex noise was uploaded here, with eight separate but similar functions:

- **psrdnoise2D.glsl**

```
vec3 psrdnoise(vec2 pos, vec2 per, float rot)
```

```
vec3 psdnoise(vec2 pos, vec2 per)
```

```
float psrnoise(vec2 pos, vec2 per, float rot)
```

```
float psnoise(vec2 pos, vec2 per)
```

```
vec3 srdnoise(vec2 pos, float rot)
```

```
vec3 sdnoise(vec2 pos)
```

```
float srnoise(vec2 pos, float rot)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```






```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

```
float snoise(vec2 pos)
```

About

GLSL procedural noise functions compatible with WebGL

-  Readme
-  MIT License
-  289 stars
-  17 watching
-  254 forks