dorkShop: Paint with Shader

SHI Weili · 石伟力

Parsons School of Design

March 13th, 2016

In this dorkShop, we are going to explore the powerhouse of your computer—the Graphics Processing Unit (GPU).

By using **GLSL fragment shader** as your paint brush, you can take full advantage of the GPU's computational power to generate amazing visuals to enhance your work in various platforms such as openFrameworks, Processing, Unity, and WebGL.

Part 1 Fundamental Concepts
Examples of Shader Application

Part 2 Code Walk-through: Paint with Shader

Part 3 Shader in openFrameworks/Unity
Further Exploration

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages

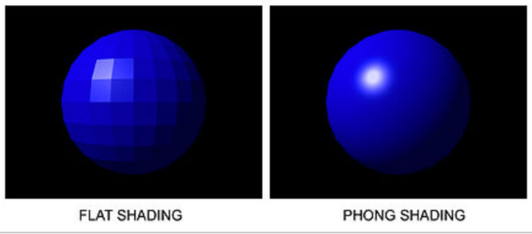
Shader

From Wikipedia, the free encyclopedia

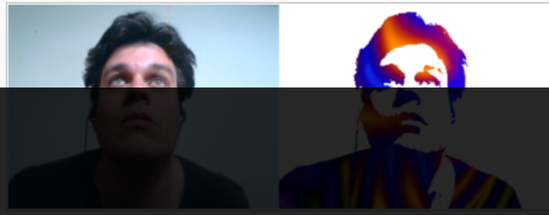
This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
(April 2014)

In the field of computer graphics, a **shader** is a computer program that is used to do shading: the production of appropriate levels of color within an image, or, in the modern era, also to produce special effects or do video post-processing. A definition in layman's terms might be given as "a program that tells a computer how to draw something in a specific and unique way".

Shaders calculate rendering effects on graphics hardware with a high degree of flexibility. Most shaders are coded for a graphics processing unit (GPU), though this is not a strict requirement. Shading languages are usually used to program the programmable GPU rendering pipeline, which has mostly superseded the fixed-function pipeline that allowed only common geometry transformation and pixel-shading functions; with shaders, customized effects can be used. The position, hue, saturation, brightness, and contrast of all pixels, vertices, or textures used to construct a final image can be altered on the fly, using algorithms defined in the shader language, and modified by external variables or textures introduced by the program calling the shader.

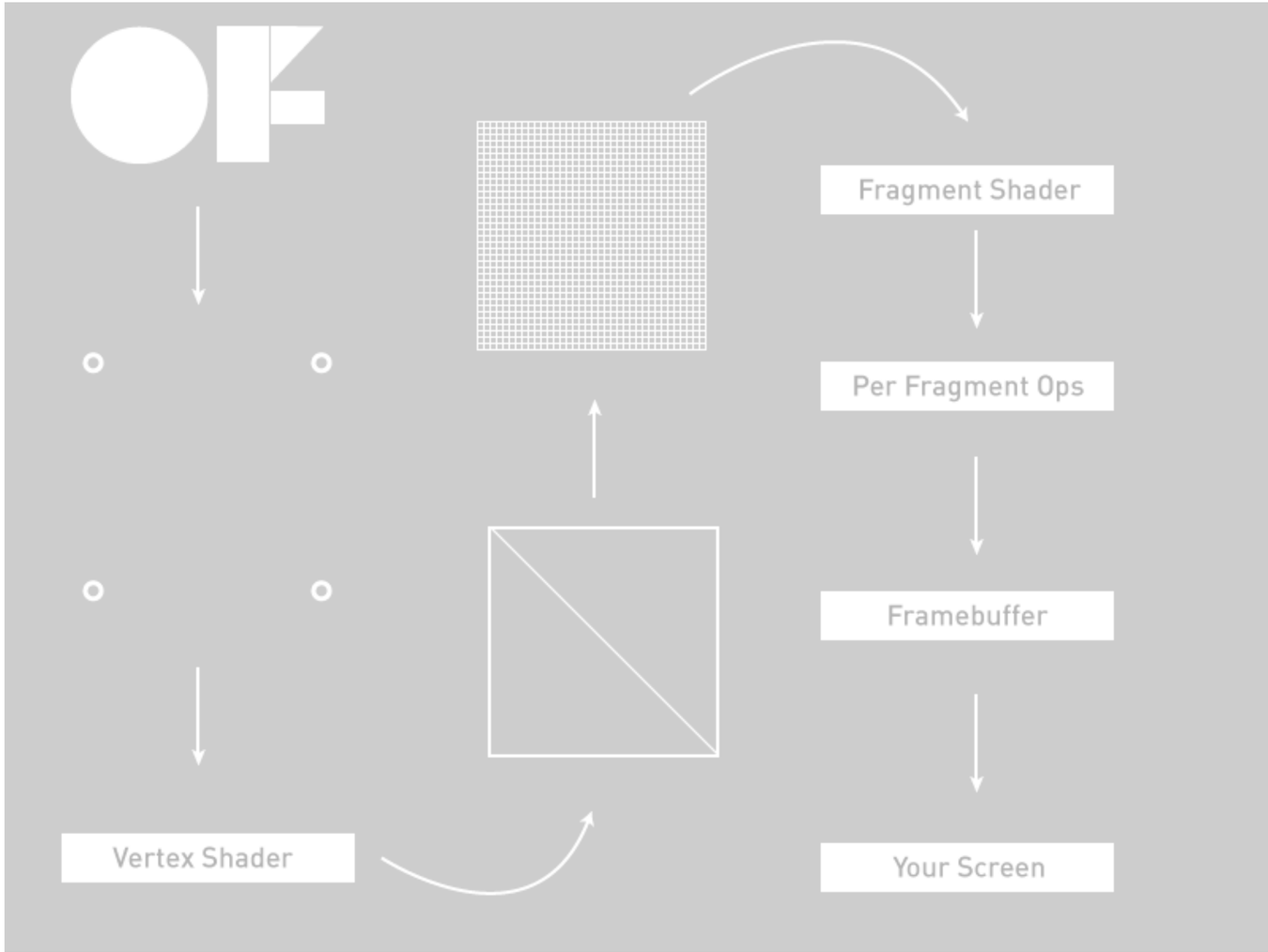


Shaders are most commonly used to produce lighting and shadow in 3D modeling. This image illustrates Phong shading, one of the first computer shading models ever developed



Shaders can also be used for

https://en.wikipedia.org/wiki/Shader

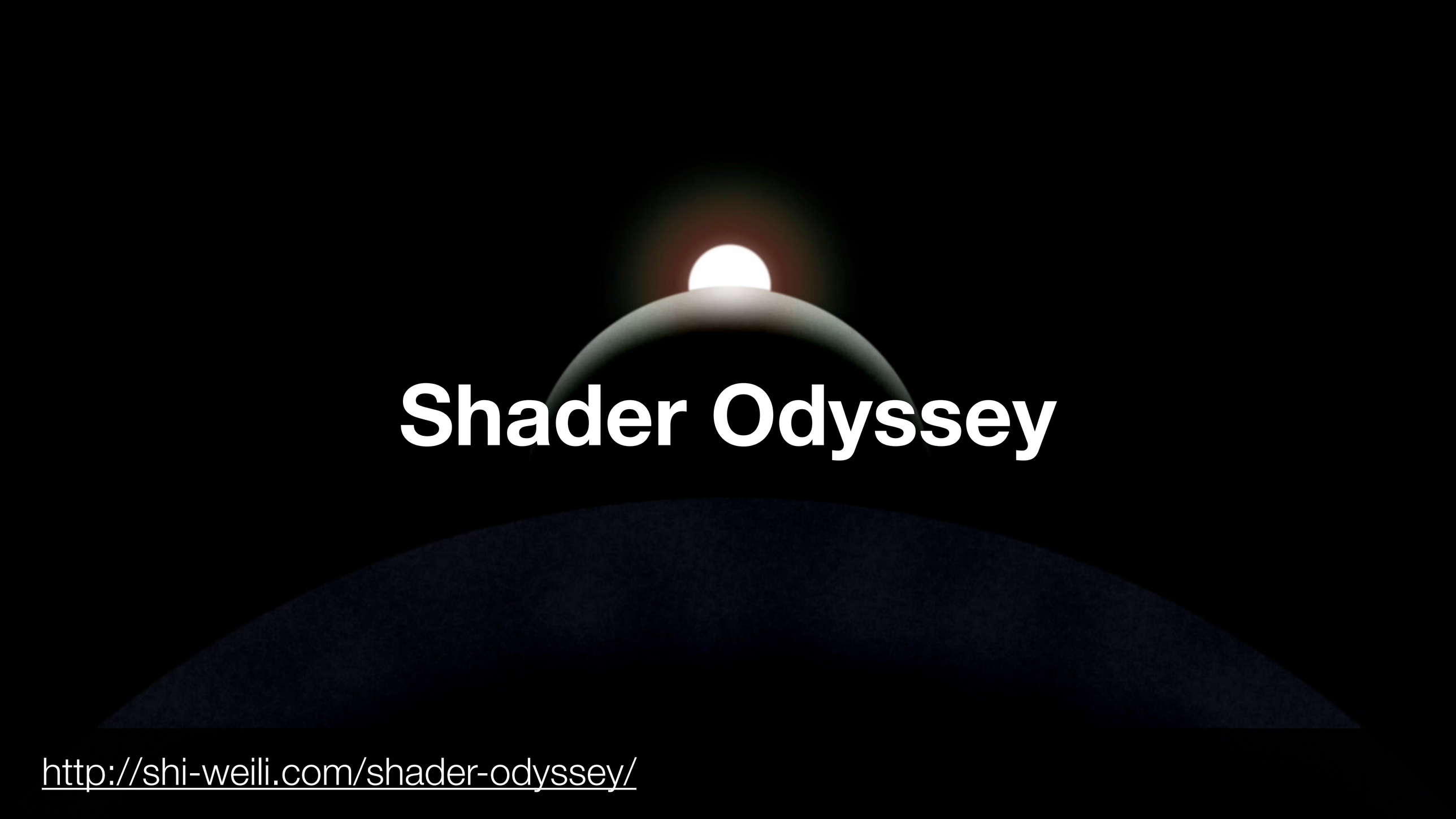


The Graphics Pipeline

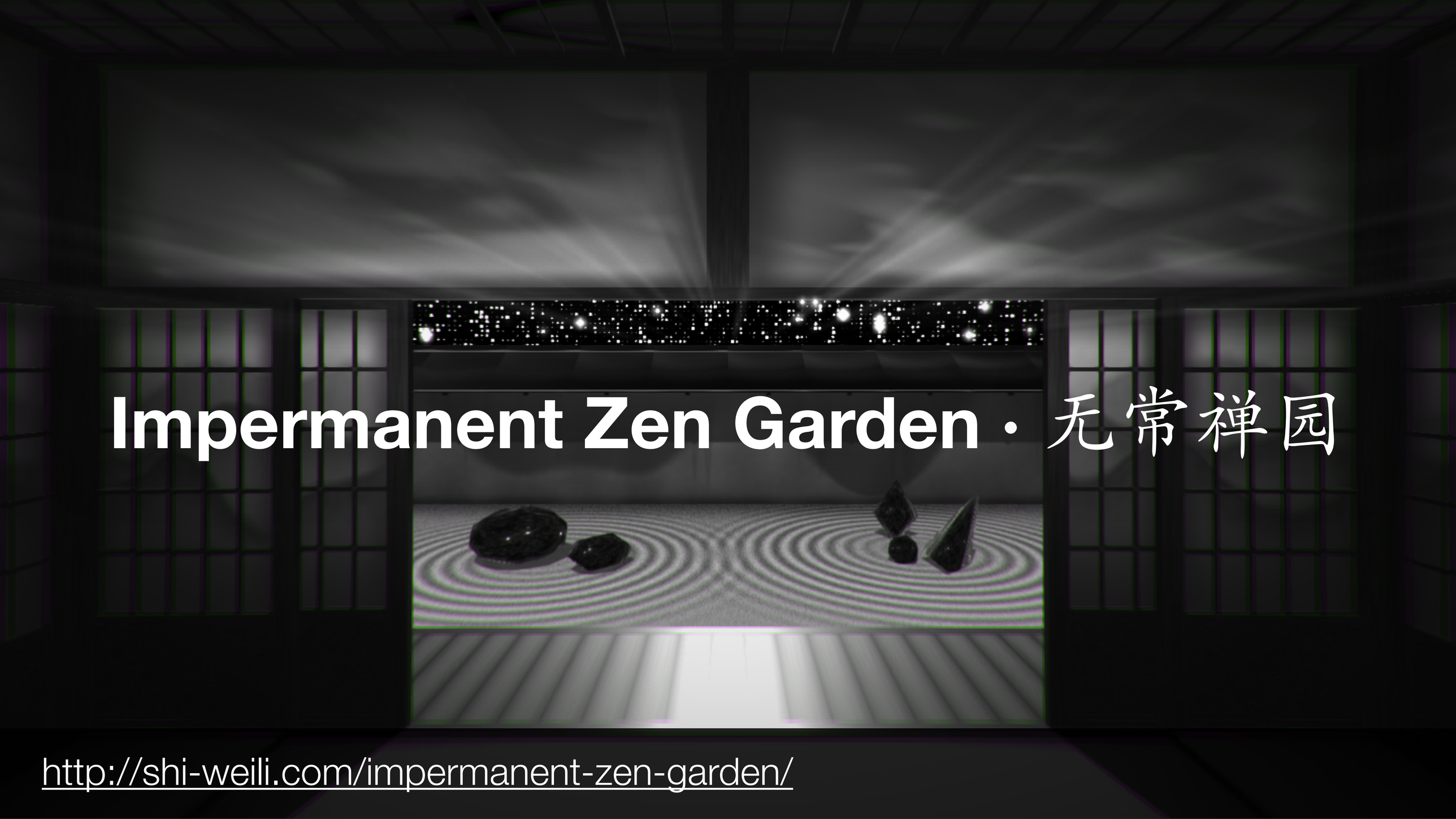
Walking from the upper left to the lower right we have:

1. Our OF application passing some vertex positions and texture coordinates to the graphics card
2. Our Vertex Shader.
3. Primitive Assembly setup of primitives, e.g. triangles, lines, and points
4. Rasterization: interpolation of data (colors, texture coordinates, other varying values) for all pixels
5. Our Fragment Shader
6. Per-pixel ops like discarding pixels for alpha, depth, and other reasons that would cause a pixel to not be drawn
7. Off to the framebuffer
8. Onto your screen

http://openframeworks.cc/ofBook/chapters/shaders.html

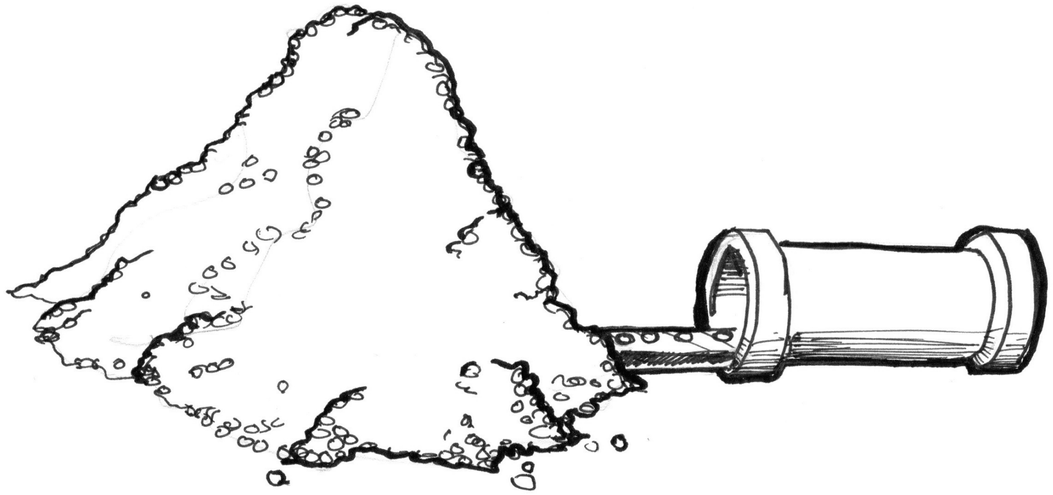


http://shi-weili.com/shader-odyssey/

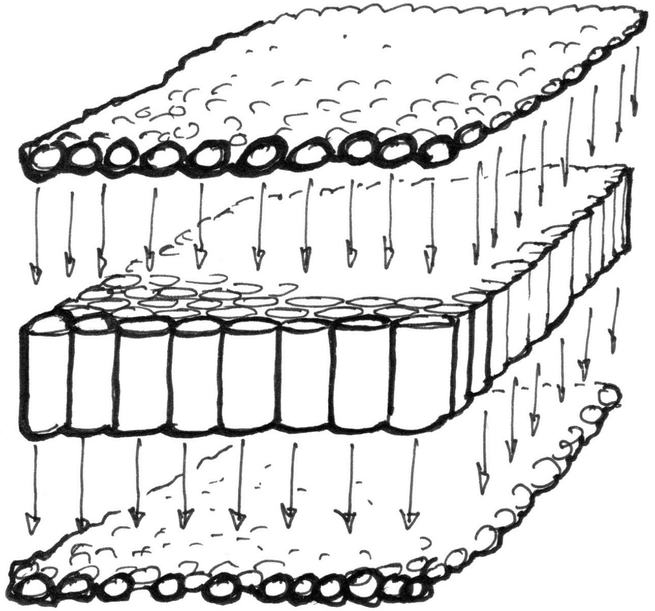


Impermanent Zen Garden · 无常禅园

http://shi-weili.com/impermanent-zen-garden/



CPU



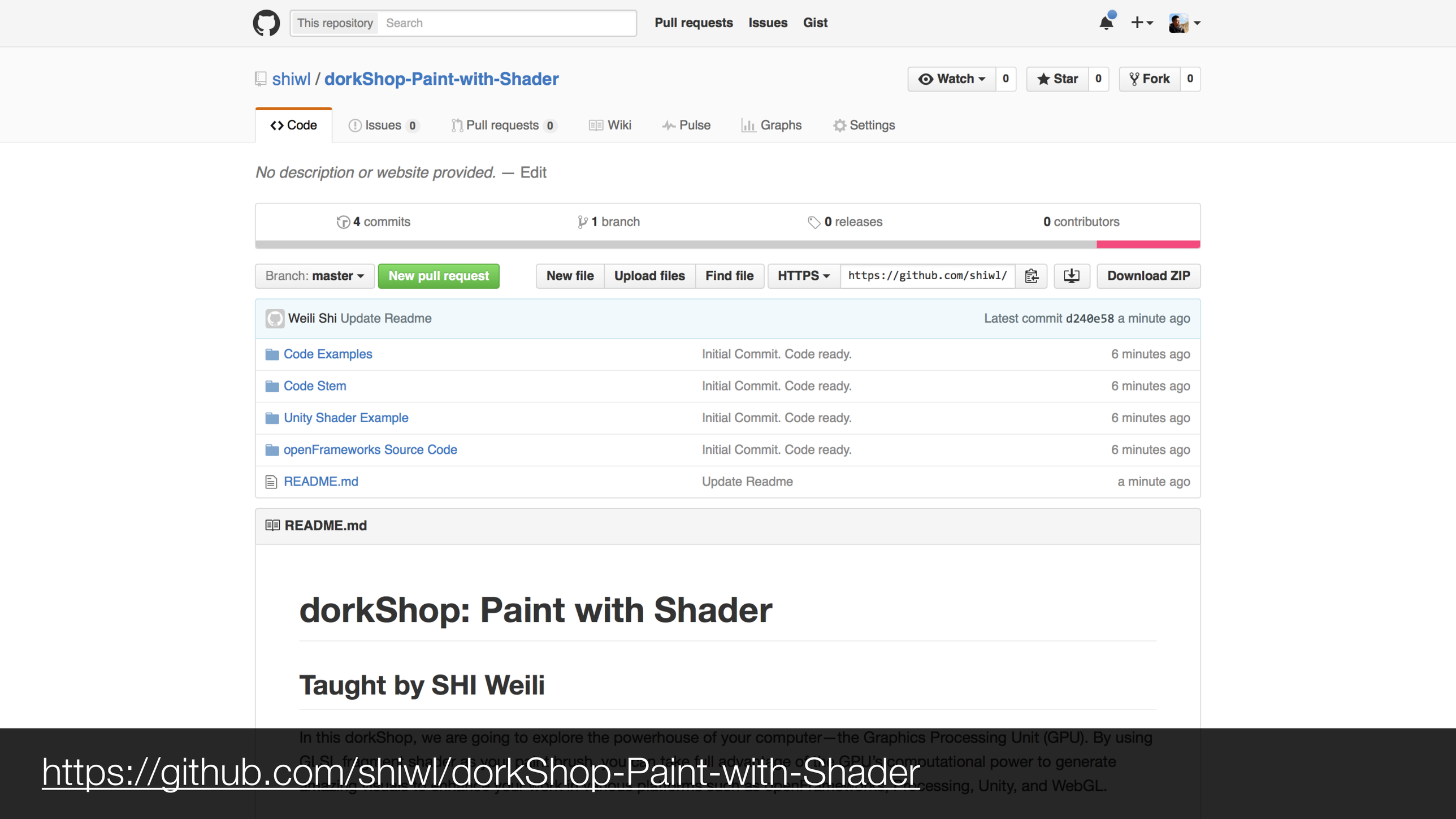
GPU

Image credit: **Patricio Gonzalez Vivo**, *The Book of Shaders*

We are going to walk through a few examples of GLSL fragment shader in a fast pace.

Our focus is on the drawing abilities of shader programming.

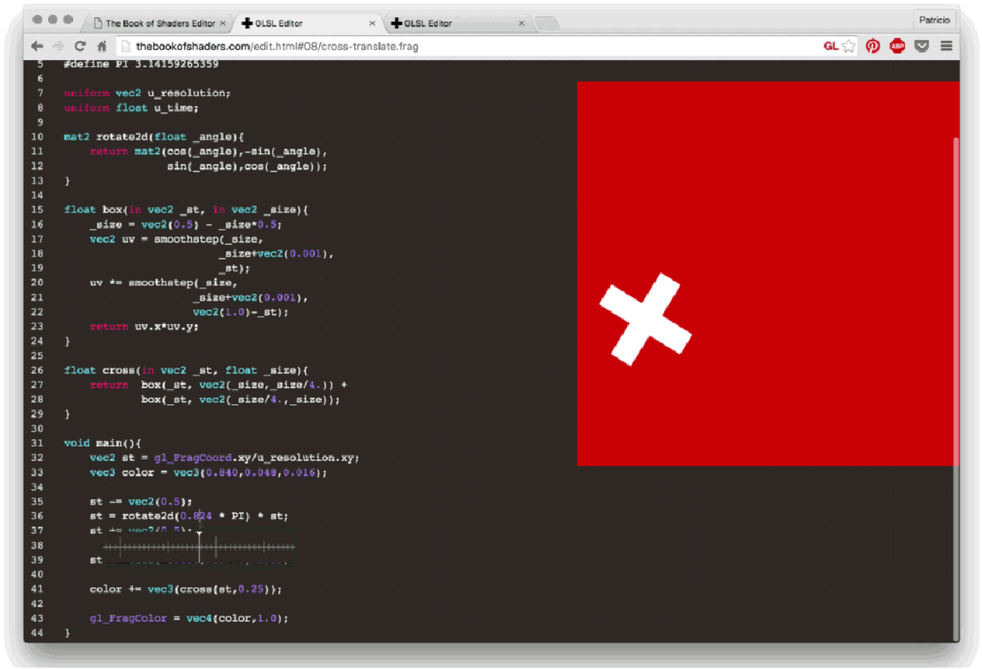
Keep in mind: **Do not try to understand and remember everything** in the code examples. The range of knowledge/techniques covered by these examples are normally taught in weeks. The aim of this dorkShop is to show you what unique thing shader can do, so that you know whether you want explore it further afterwards.



https://github.com/shiwl/dorkShop-Paint-with-Shader

Me Collaborations Tools Teaching

GlslEditor

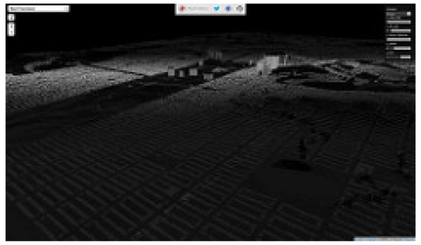


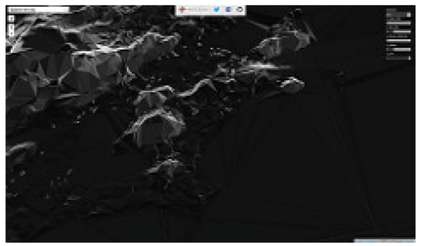
Donate

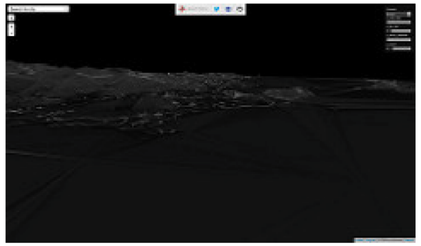
Friendly GLSL Shader editor based on **Codemirror** compatible with **glslViewer** (C++/OpenGL ES) and **glslCanvas** (JS/WebGL).

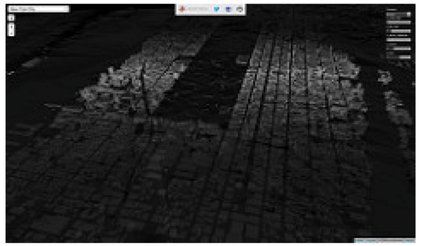
Was originaly develop to work as a embebed editor for **The Book of Shaders**. But now have grown as a stand alone Web app. Thanks to their compatibility with other apps of this ecosystems like **glslViewer** that runs in the RaspberryPi directly from console, **GlslEditor** interact with other projects like **OpenFrame.io** allowing the user to export the shaders to frames with only one button.

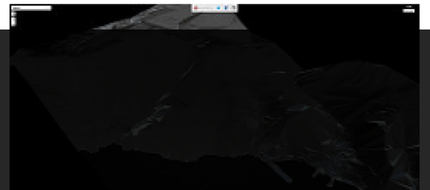
Now working on:











http://patriciogonzalezvivo.com/2016/glslEditor/

⊙ New ⇩ Open ⇧ Export

```
1   // Author:
2   // Title:
3
4   #ifdef GL_ES
5   precision mediump float;
6   #endif
7
8   uniform vec2 u_resolution;
9   uniform vec2 u_mouse;
10  uniform float u_time;
11
12  void main() {
13      vec2 st = gl_FragCoord.xy/u_resolution.xy;
14      st.x *= u_resolution.x/u_resolution.y;
15
16      st += vec2(.0);
17      vec3 color = vec3(1.);
18      color = vec3(st.x,st.y,abs(sin(u_time)));
19
20      gl_FragColor = vec4(color,1.0);
21  }
```

http://editor.thebookofshaders.com

Code Example 1

Hello World

- Every pixel counts!
- Uniforms

Code Example 2

Metaballs

- Distance Field
- Shaping Functions
  - step()
  - smoothstep()

Glossary

By theme

- TYPES

void bool int float bvec2 bvec3 bvec4 ivec2 ivec3 ivec4 vec2 vec3 vec4 mat2 mat3 mat4 sampler2D samplerCube struct

- QUALIFIERS

attribute const uniform varying precision highp mediump lowp in out inout

- BUILT-IN VARIABLES

gl_Position gl_PointSize gl_PointCoord gl_FrontFacing gl_FragCoord gl_FragColor

- BUILT-IN CONSTANTS

gl_MaxVertexAttribs gl_MaxVaryingVectors gl_MaxVertexTextureImageUnits gl_MaxCombinedTextureImageUnits gl_MaxTextureImageUnits gl_MaxFragmentUniformVectors gl_MaxDrawBuffers

- ANGLE & TRIGONOMETRY FUNCTIONS

radians() degrees() sin() cos() tan() asin() acos() atan()

- EXPONENTIAL FUNCTIONS

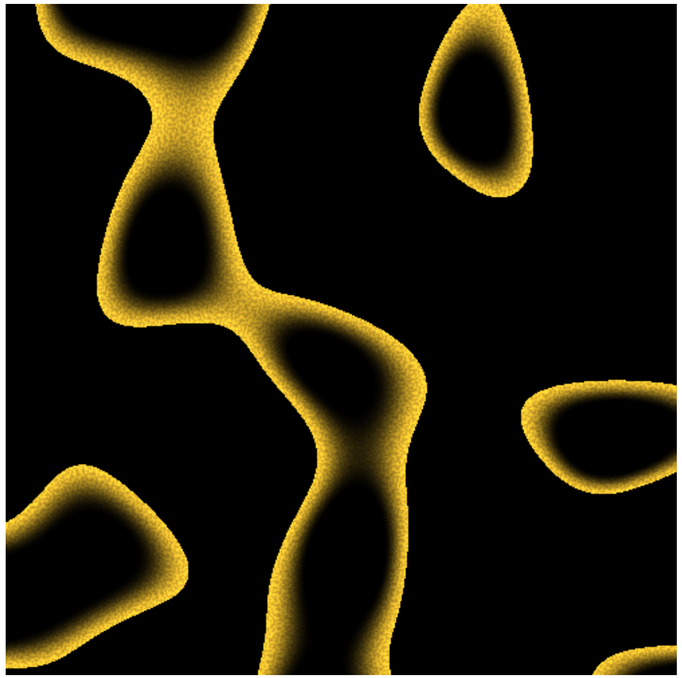
pow() exp() log() exp2() log2() sqrt() inversesqrt()

http://patriciogonzalezvivo.com/2015/thebookofshaders/glossary/?search=distance

Code Example 3

The Nature of Code

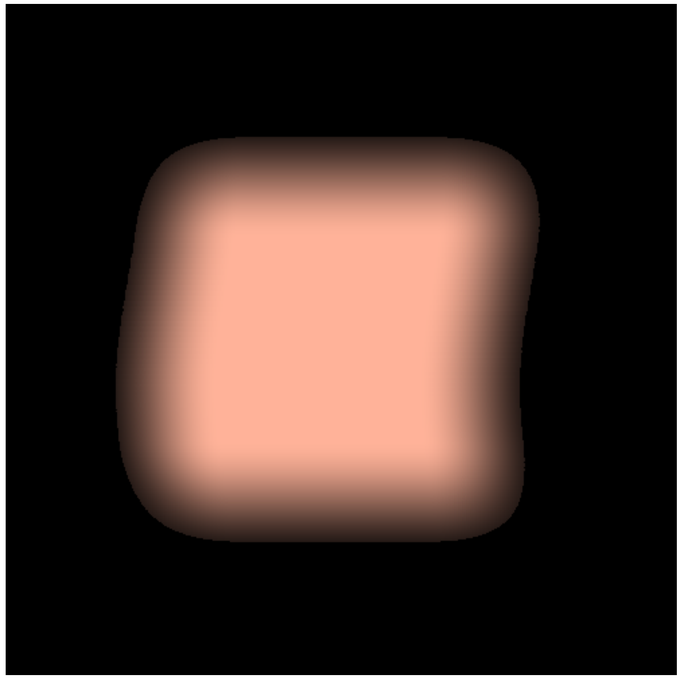
- Random
- Noise



Code Example 4

Jelly Square

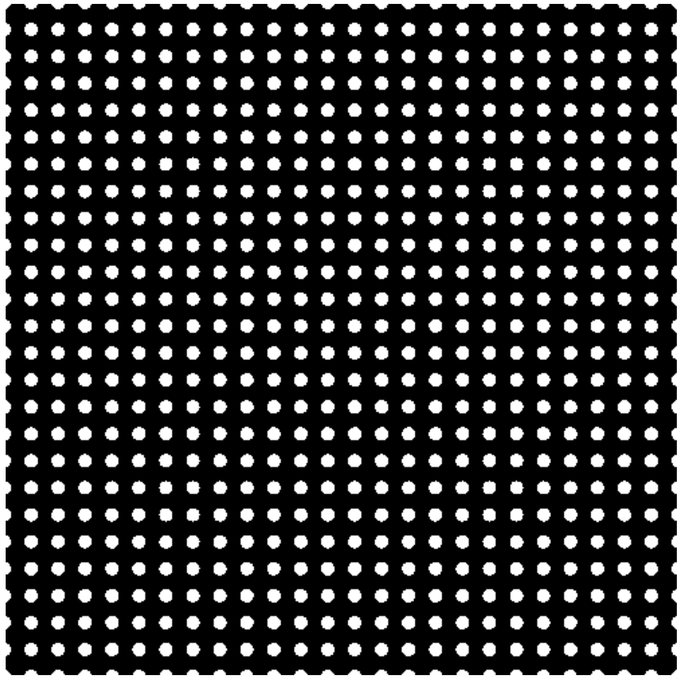
- Noise
- Shaping Functions
  - pow()



Code Example 5

From One to One Million

- Scaling



Application Example 1

openFrameworks

- Setting the uniforms

Introducing Shaders

by Lucasz Karluk, Joshua Noble, Jordi Puig

introducing

This tutorial comes in two parts: first, this thing, the HTML file and second, nine OF projects that progress along with this tutorial. You can find them in the example folder of your OF download, under `examples/shader` or on github. As you read along with this, you should check the code downloads as well because a lot of the code isn't here, but rather is there. You'll notice that all of those project folders have source code and a data folder that contains 3 different kinds of shaders: GL2, GL3, and GLES2. What these are and how they're different from one another will be described below.

You've probably heard somewhere about "shaders", those mystical things that let you render beautiful things at blazing speed. Shaders are not actually mystical but they are a little tricky to get started with at first because they're working with a different part of the computer than you're usually working with in your openFrameworks application: you're working on the Graphics Processing Unit (as opposed to the Central Processing Unit that you're usually working on in OF). To really get the hang of working with shaders, we need to get you a little background on what

http://openframeworks.cc/ofBook/chapters/shaders.html

Application Example 2

Unity 3D

- Cg Shader

The Cg Tutorial

The Cg Tutorial is now available, right here, online. You can purchase a beautifully printed version of this book, and others in the series, at a 30% discount courtesy of InformIT and Addison-Wesley.

Please visit our Recent Documents page to see all the latest whitepapers and conference presentations that can help you with your projects.

Appendix E. Cg Standard Library Functions

Cg provides a set of built-in functions and predefined structures with binding semantics to simplify GPU programming. These functions are similar in spirit to the C standard library, offering a convenient set of common functions. In many cases, the functions map to a single native GPU instruction, so they are executed very quickly. Of the functions that map to multiple native GPU instructions, you may expect the most useful to become more efficient in the near future.

Although you can write your own versions of specific functions for performance or precision reasons, it is generally wiser to use the Cg Standard Library functions when possible. The Standard Library functions will continue to be optimized for future GPUs; a program written today using these functions will automatically be optimized for the latest architectures at compile time. Additionally, the Standard Library provides a convenient unified interface for both vertex and fragment programs.

This appendix describes the contents of the Cg Standard Library, and is divided into the following five sections:

- **"Mathematical Functions"**
- **"Geometric Functions"**
- **"Texture Map Functions"**
- **"Derivative Functions"**
- **"Debugging Function"**

Where appropriate, functions are overloaded to support scalar and vector variants when the input and output types are the same.

E.1 Mathematical Functions

Table E-1 lists the mathematical functions that the Cg Standard Library provides. The table includes functions useful for trigonometry, exponentiation, rounding, and vector and matrix manipulations, among others. All functions work on scalars and vectors of all sizes, except where noted.

Table E-1. Mathematical Functions

Function	Description
`abs( x )`	Absolute value of ***x*** .
`acos( x )`	Arccosine of ***x*** in range [0, p], ***x*** in [−1, 1].
`all( x )`	Returns `true` if every component of ***x*** is not equal to 0. Returns `false` otherwise.
[illegible]	[illegible] equal to 0. Returns `false` otherwise.

Alan Zucconi

in programming

A gentle introduction to shaders in Unity3D

Part 1, Part 2, Part 3, Part 4, Part 5

Published

June 10, 2015

Tutorials
Unity
Shader
Python
Arduino
Machine Learning
Maths for Gamedev

We can safely say that Unity3D has made game development easier for a lot of people. Something where it still has a long way to go is, with no doubt, *shader coding*. Often surrounded by mystery, a shader is a program specifically made to run on a GPU. It is, ultimately, what draws the triangles of your 3D models. Learning how to code shaders is essential if you want to give a special look to your game. Unity3D also uses them for postprocessing, making them essential for 2D games as well. This series of posts will gently introduce you to shader coding, and is oriented to developers with little to no knowledge about shaders.

Introduction

The diagram below *loosely* represents the three different entities which plays a role in the rendering workflow of Unity3D:

http://www.alanzucconi.com/2015/06/10/a-gentle-introduction-to-shaders-in-unity3d/

Package Control

OpenGL sha

- Installation
- Browse
- Search
- Docs
- News
- Stats
- About
- Say Thanks

BROWSE

OpenGL Shading Language (GLSL)

by euler0 ST2/ST3

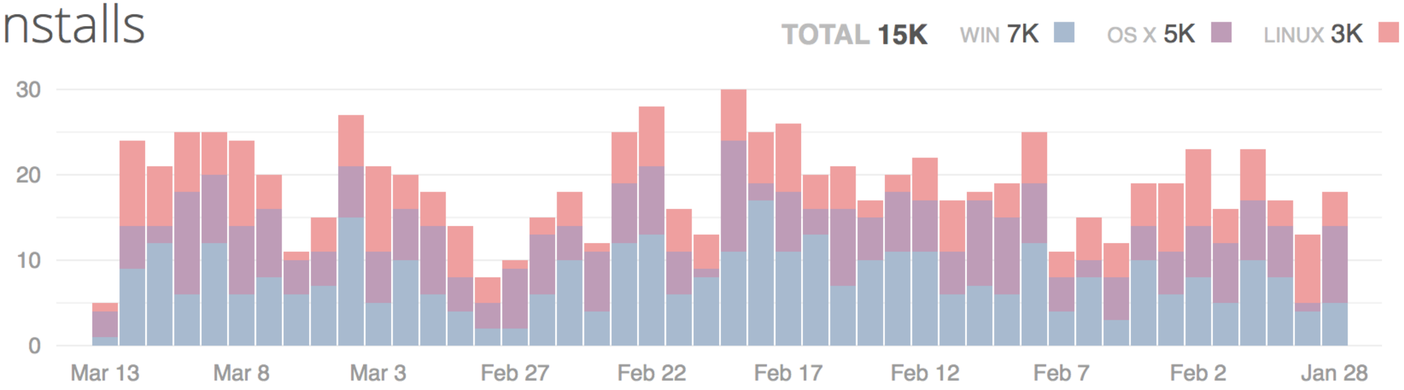
GLSL Syntax Highlighting for Sublime Text 2 & 3

LABELS language syntax

Details

VERSION 2015.09.13.13.53.36
HOMEPAGE github.com
ISSUES github.com
MODIFIED 6 months ago
LAST SEEN 7 minutes ago
FIRST SEEN 4 years ago

Installs



SUPPORTED VIA

Readme

SOURCE raw.githubusercontent.com

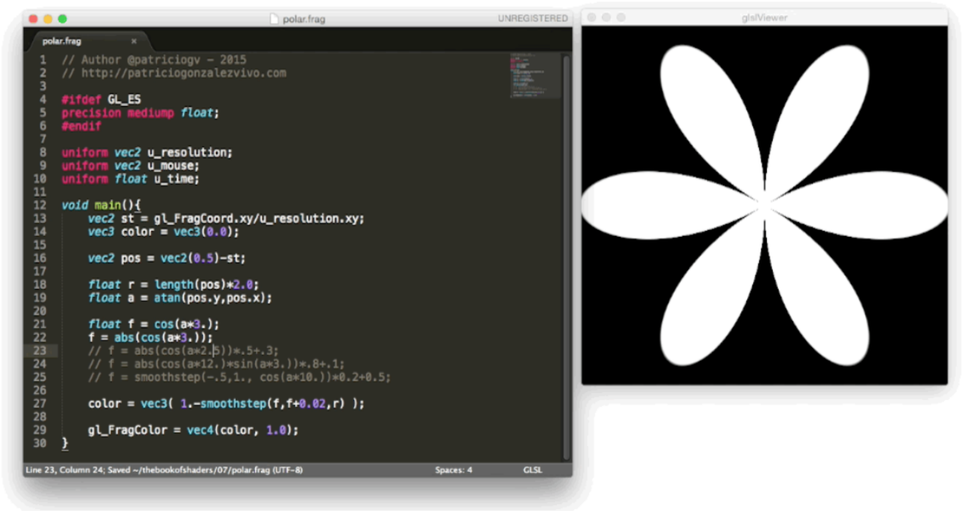
Sublime Text 2 & 3 - GLSL Package

How to Install

https://packagecontrol.io/packages/OpenGL%20Shading%20Language%20(GLSL)

2. Run `Package Control: Install Package` command, find and install `OpenGL Shading Language (GLSL)` package.

glslViewer



Donate

Live-coding console tool that renders GLSL Shaders. Every file you use (frag/vert shader, images and geometries) are watched for modification, so they can be updated on the fly.

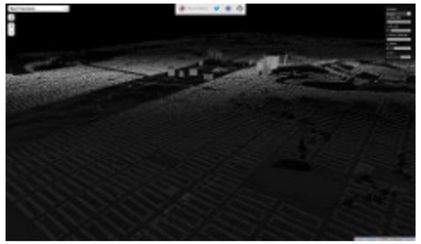
Install

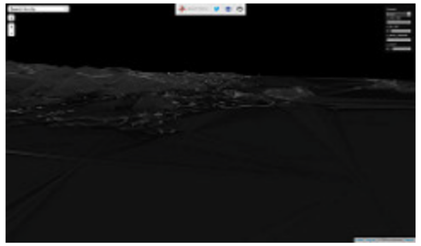
Installing in Ubuntu

You need to **install GLFW** then download the code, compile and install.

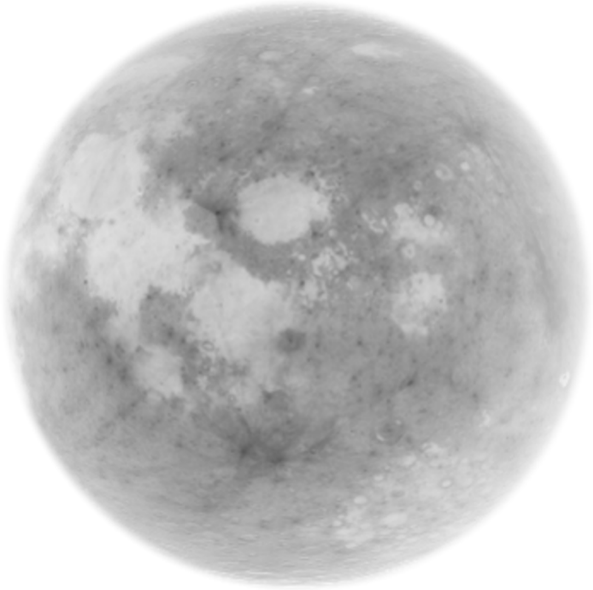
```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install git-core cmake xorg-dev libglu1-mesa-dev
cd glfw
cmake .
```

Now working on:





http://patriciogonzalezvivo.com/2015/glslViewer/



The Book of Shaders

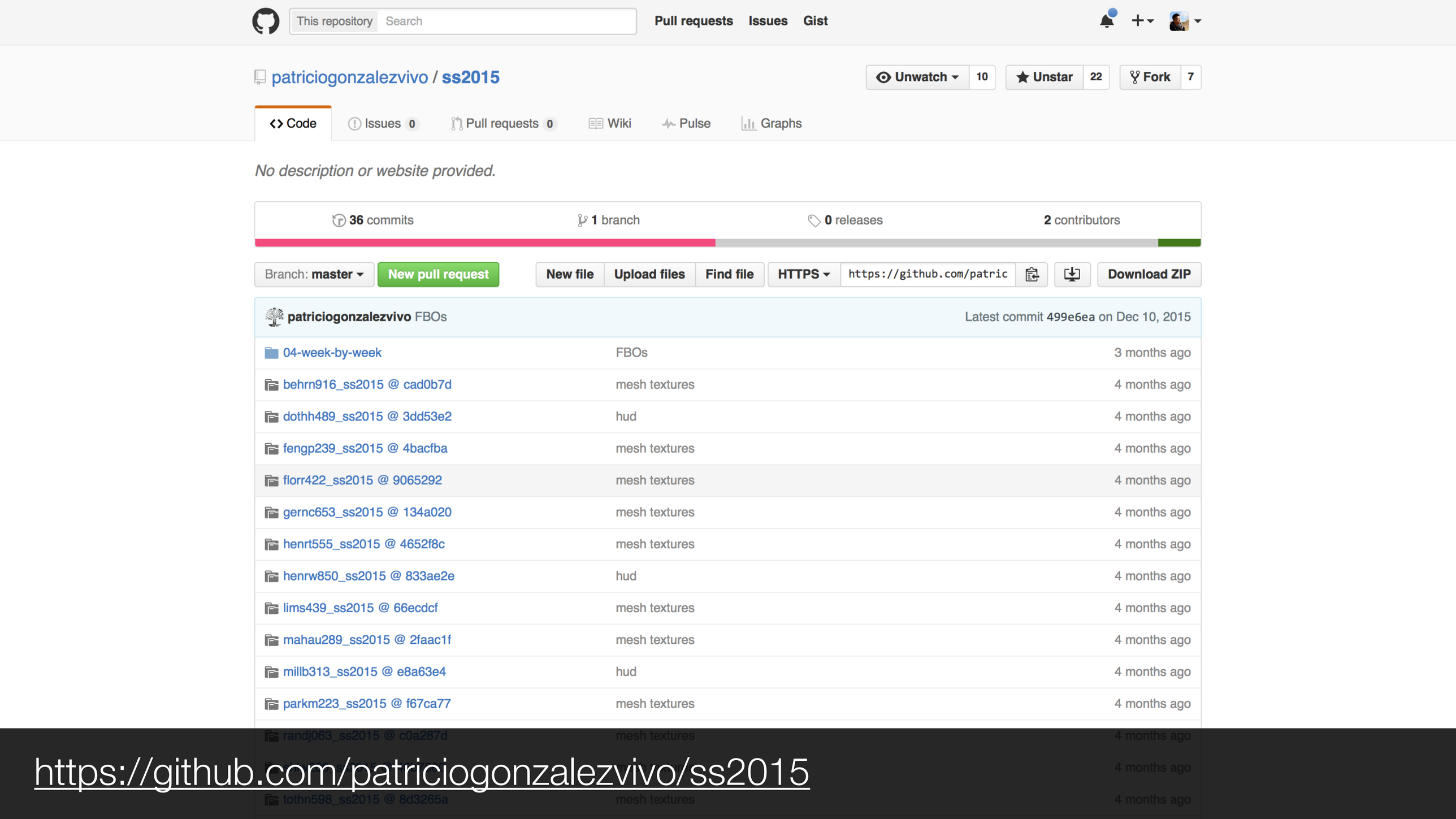
by Patricio Gonzalez Vivo

This is a gentle step-by-step guide through the abstract and complex universe of Fragment Shaders.

Contents

Donate

http://patriciogonzalezvivo.com/2015/thebookofshaders/



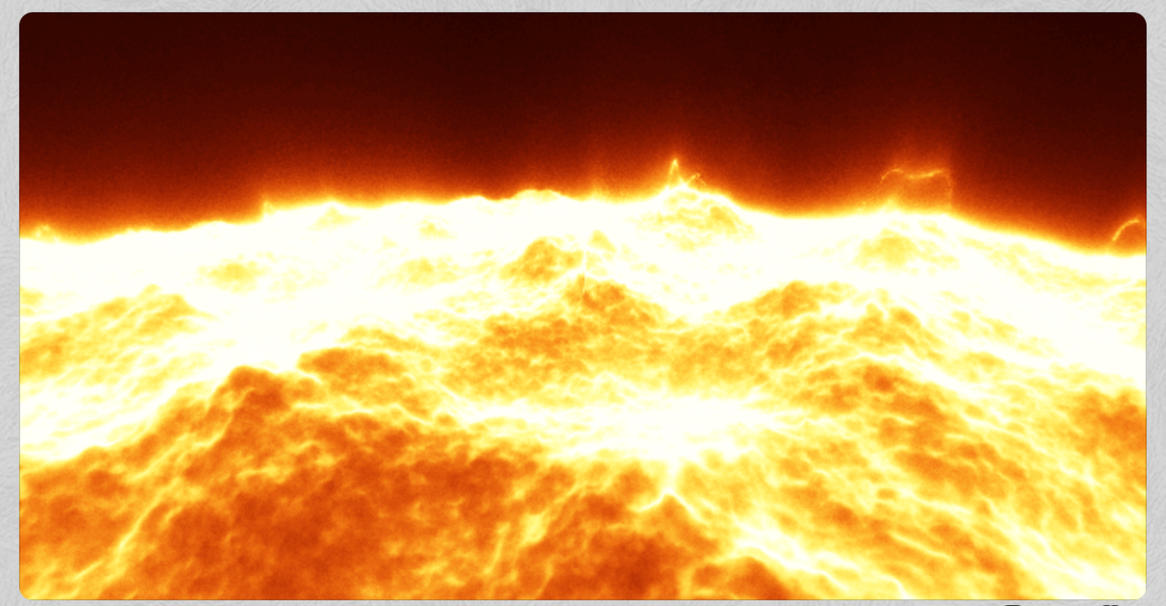
https://github.com/patriciogonzalezvivo/ss2015

Shadertoy

Search...

Browse Live New Shader Sign In

Shader of the Week

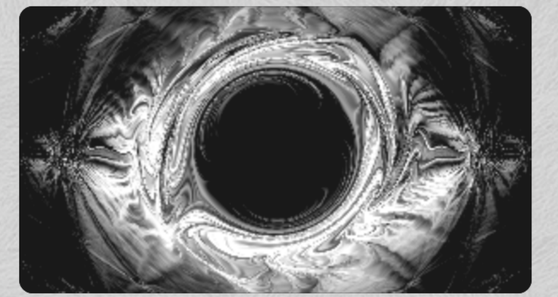


"Sun surface" by **Duke** 3135 85

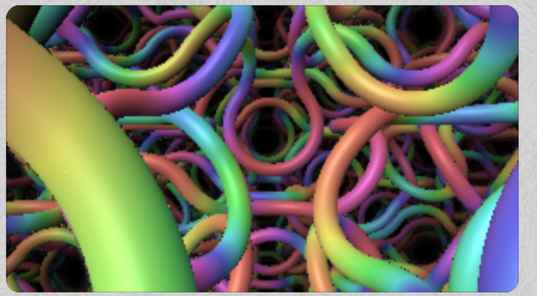
Build and **Share** your best shaders with the world and get **Inspired**

Latest contributions: **Shader Lesson #3, animation** by **hubbe** 1 hour ago, **Dancing Sun** by **quad_damage** 2 hours ago, **Plasma cube** by **patu** 3 hours ago, **Shader Lesson #2, functions** by **hubbe** 4 hours ago, **spectrum #1** by **davin** 5 hours ago

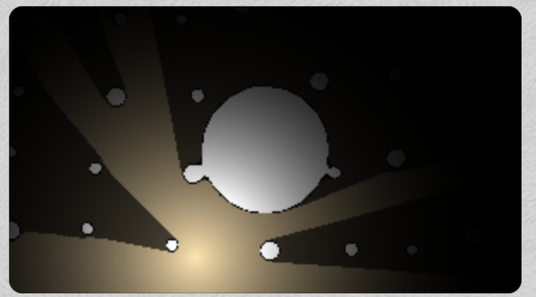
Featured shaders



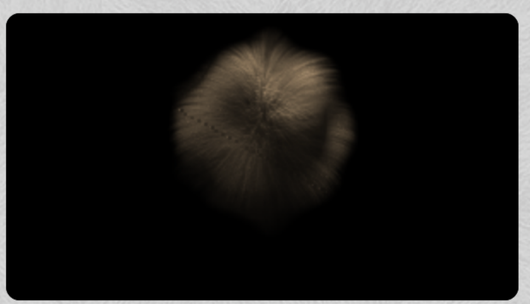
"Edge of the universe" by **fern**... 534 11



"rainbow spaghetti" by **mattz** 823 29



"2015/11/04" by **hughsk** 956 18



"Fur Distance Field" by **hat** 2614 48

https://www.shadertoy.com

Photo credit: **Megan Fox** in *Transformers* (2007)

Take full control of your Bumblebee!

Photo credit: **Barrett-Jackson**

dorkShop: Paint with Shader

SHI Weili · 石伟力

Parsons School of Design

March 13th, 2016