khw7096 Update osl.md cfb156a 9 days ago

1 contributor

185 lines (152 sloc) | 8.5 KB

# Open Shanding Language

대부분의 3D소프트웨어에서 쉐이딩 작업 쉽게 몇몇 조작을 통해서 진행할수 있습니다. 이 방법은 많이 사용되며 굉장히 쉬운 방법입니다.

이번 시간에 알아보는 방법은 코딩을 통해서 쉐이딩 작업을 해보는 것 입니다. Open Shading Language는 최초 소니이미지 픽쳐스에서 아놀드 렌더러에 사용하기 위해서 만들어진 쉐이딩 렝귀지 입니다.

쉐이더를 만들기 위한 언어입니다. 많은 렌더러들이 지원하며 렌더러마다 조금씩 문법의 차이는 있습니다. 쉐이더를 작성할 때 조금 수고 스럽더라도 OSL로 작성하면 Asset 라이브러리화 하기 쉽습니다. 조직이 다른 렌더러로 갈아타더라도 쉐이더를 재활용할 수 있기 때문에 Shading Asset의 가치도 높습니다.

지원하는 렌더러는 아래와 같습니다.

- Pixar: PhotoRealistic RenderMan RIS
- Autodesk/SolidAngle: Arnold
- DNA Research: 3Delight (Katana Default Renderer)
- Chaos Group: V-Ray
- Autodesk: 3DS Max 2019
- Blender/Cycles
- Sony Pictures Imageworks: in-house "Arnold" renderer
- Isotropix: Clarisse
- Autodesk Beast
- Opensource Renderer Appleseed
- Animal Logic: Glimpse renderer
- Image Engine: Gaffer (for expressions and deformers)
- Ubisoft motion picture group's proprietary renderer
- 마야 : 기본 렌더러 : Arnold, Plug-ins : Vray, Renderman을 사용할 수 있습니다.

## 강점

그냥 3D 소프트웨어로 작업을 할 때 효율성도 좋지않기도 하고 이 언어 자체를 이용해서 작업을 할 필요는 없습니다. 내부에서 지원하는 도구를 이용해서 쉐이더를 작성하면 되니까요. 하지만 회사 전체에 쉐이더 시스템을 에셋으로 구축하고 텍스쳐 라이브러리를 구성하고 바로 렌더가능하도록 모든 리소스를 빌드하고 싶다면 상황은 달라집니다. 대부분 렌더러가 OSL 을 지원하기 때문에 OSL 베이스로 쉐이더 에셋을 구축해두면 미래적으로 경쟁력 있는 조직, 회사가 될 수 있습니다. OSL을 알아두면 나중에 렌더맨을 공부할 때 도움이 됩니다. 굉장히 프로세스가 비슷합니다.

룩뎁이 끝나고 모든 쉐이더를 Lock 걸어야 할 때 실수로 아티스트가 옵션을 건드리지 않도록 할 때도 활용할 수 있습니다.

- 회사 쉐이더 시스템을 구축할 때
- 렌더러 전환에 대비할 때
- 아티스트가 실수로 옵션을 건드리지 않았으면 할 때
- 수치가 굉장히 많고 미묘한 값으로 결과가 크게 달라지는 쉐이더
- 기본 쉐이더를 꺼내서 항상 반복적인 셋팅을 해야하는 경우

## 언어구성

보통 아래 형태의 언어를 띕니다. C++ 코드와 비슷하게 생겼습니다.

아래는 OSL 파일의 한 예입니다.

```
#include <stdosl.h>

shader TDdiffuse_ramp(
        normal Normal = N,
        color Color1 = color(0.235205, 0.8, 0.025),
        color Color2 = color(0.0, 0.8, 0.0),
        color Color3 = color(0.0, 0.0, 0.8),
        color Color4 = 0.1,
        color Color5 = 0.2,
        color Color6 = 0.3,
        color Color7 = 0.4,
        color Color8 = 0.5,
        output closure color BSDF = 0 )
{
    color Color[8] = {Color1, Color2, Color3, Color4, Color5, Color6, Color7, Color8};

    BSDF = diffuse_ramp(Normal, Color);
}
```

## Unreal

아직 지원하고 있지 않습니다. 아래 URL에서 해당 사항에 대해서 Discussion은 올라와 있습니다.
https://forums.unrealengine.com/development-discussion/rendering/36083-osl-support

## 명령어

gaffer를 설치하면 내부에 이미 osl 명령어가 존재합니다. osl 명령어를 사용하기 위해서 gaffer가 설치된 경로의 LD_LIBRARY_PATH를 .bashrc에 설정할 필요가 있습니다. Renderman Non커머셜을 설치해도 빌드된 명령어를 이용할 수 있습니다. 렌더맨이 설치된 경로의 /bin 디렉토리에는 오픈소스 명령어도 많이 존재합니다. 렌더맨을 제외한 나머지 명령어는 오픈소스이기 때문에 활용가능합니다.

### oslc

osl 컴파일러 컴파일러 입니다. osl 파일을 이용해서 oso 파일을 생성하면 렌더러가 바로 사용할 수 있습니다. 일반적으로 렌더러는 이 과정을 자동으로 많이 해줍니다.

```
$ oslc input.osl
Compiled noise.osl -> noise.oso
$ ls
input.osl input.oso
```

### oslinfo

터미널에서 oso 파일 정보를 출력합니다.

쉐이더 로딩타임 측정

```
$ oslinfo --runstats noise.oso
0.00605843 sec for noise.oso
```

### osltoargs

oso파일을 xml로 출력합니다.

```
$ osltoargs noise.oso -o output.xml
```

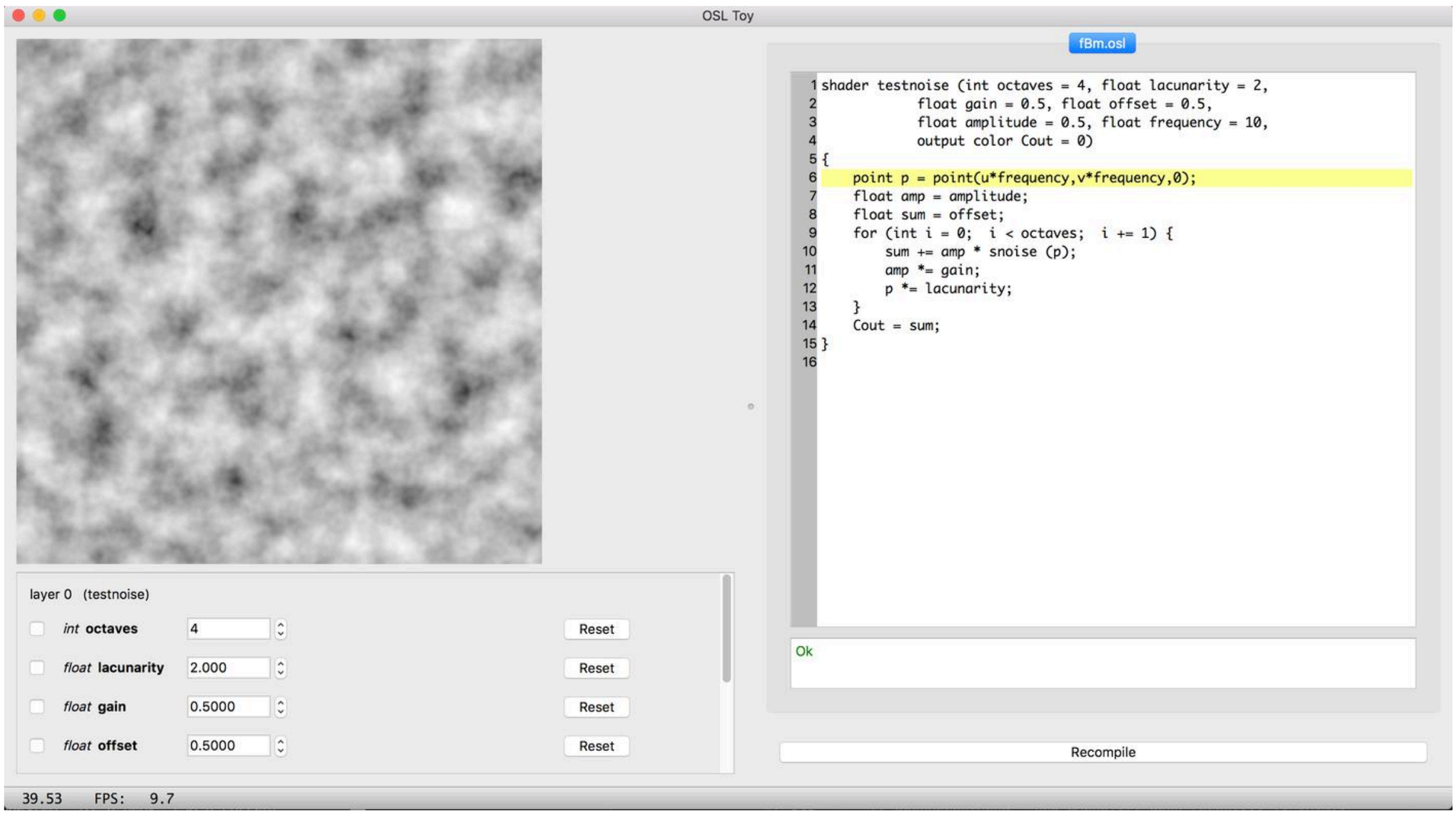
쉐이더의 옵션을 터미널에서 볼 수 있습니다.

```
$ oslinfo noise.oso
shader "noise"
float Time  1
point Point nodefault
output float Cell  0
output color Perlin  [ 0.8 0.8 0.8 ]
output color UPerlin  [ 0.8 0.8 0.8 ]
output color Simplex  [ 0.8 0.8 0.8 ]
output color USimplex  [ 0.8 0.8 0.8 ]
```

### osltoy

osl 코드를 인터렉티브하게 보여주는 툴입니다.

```
$ osltoy input.osl
```




## 실습

OSL을 작성, 적용하기 위해서는 3D 소프트웨어가 필요합니다. Blender는 오픈소스이면서 OSL을 지원합니다. Blender 환경에서 OSL을 작성하고 기본적인 렌더링을 해보겠습니다.

## 컴파일정보(준비중)

- 소스코드 및 사용된 프로젝트 : https://github.com/imageworks/OpenShadingLanguage
- 컴파일정보 : https://github.com/imageworks/OpenShadingLanguage/blob/master/INSTALL.md

```
# yum install llvm-toolset-7
```

```
$ cd ~/app
$ git clone https://github.com/imageworks/OpenShadingLanguage.git OSL_src
$ mkdir OSL_build
$ mkdir OSL
$ cd OSL_build
$ scl enable llvm-toolset-7 bash
```

```
$ ~/app/cmake-3.13.2/bin/cmake ../OSL_src -DOPENEXR_INCLUDE_PATH=$HOME/app/openexr/include -
DOPENEXR_LIBRARY_PATH=$HOME/app/openexr/lib -DCMAKE_INSTALL_PREFIX=$HOME/app/OSL -
DILMBASE_INCLUDE_PATH=$HOME/app/IlmBase -DOPENIMAGEIO_INCLUDE_DIR=$HOME/app/OpenImageIO/include -
DOPENIMAGEIO_ROOT_DIR=$HOME/app/OpenImageIO

$ make OPENEXR_HOME=$HOME/app/openexr OPENIMAGEIO_INCLUDE_DIR=$HOME/app/OpenImageIO/include
```

에러

```
-- Project build dir   = /home/woong/app/OSL_build
-- Project install dir = /home/woong/app/OSL
-- platform = linux64
-- CMAKE_CXX_COMPILER is /bin/c++
-- CMAKE_CXX_COMPILER_ID is GNU
-- Building for C++11
-- Setting Namespace to: OSL_v1_11
-- Using OpenImageIO 2.1.0
CMake Error at /home/woong/app/cmake-3.13.2/share/cmake-
3.13/Modules/FindPackageHandleStandardArgs.cmake:137 (message):
  Could NOT find LLVM: Found unsuitable version "", but required is at least
  "4.0" (found )
Call Stack (most recent call first):
  /home/woong/app/cmake-3.13.2/share/cmake-3.13/Modules/FindPackageHandleStandardArgs.cmake:376
(_FPHSA_FAILURE_MESSAGE)
  src/cmake/modules/FindLLVM.cmake:113 (find_package_handle_standard_args)
  src/cmake/externalpackages.cmake:70 (find_package)
  CMakeLists.txt:128 (include)
```

## Reference

- https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwi5kfuTuYHfAhWBF4gKHaDGAVgQFjAAegQIChAC&url=https%3A%2F%2Fraw.githubusercontent.com%2Fimageworks%2FOpenShadingLanguage%2Fmaster%2Fsrc%2Fdoc%2Fosl-languagespec.pdf&usg=AOvVaw0fnZDAj-almK7unV7NKApA
- https://blendersushi.blogspot.com/2013/10/osl-basic-functions.html
- https://www.youtube.com/watch?v=9CYDi8h0SuE
- http://thhube.github.io/tutorials/osl/osl.html
- https://www.youtube.com/watch?v=sEqQFZkVVEE
- MaterialX : http://www.cgchannel.com/2017/07/lucasfilm-and-ilm-open-source-materialx/
- https://www.shadertoy.com
- Vray OSL 강좌 : http://help.chaosgroup.com/vray/help/200R1/examples_vrayosl.htm
- Skin : https://docs.sharktacos.com/vray/osl.html
- 블렌더 소스코드를 받으면 내부에서 활용하기 위해서 작성된 OSL 코드 전체를 볼 수 있습니다.