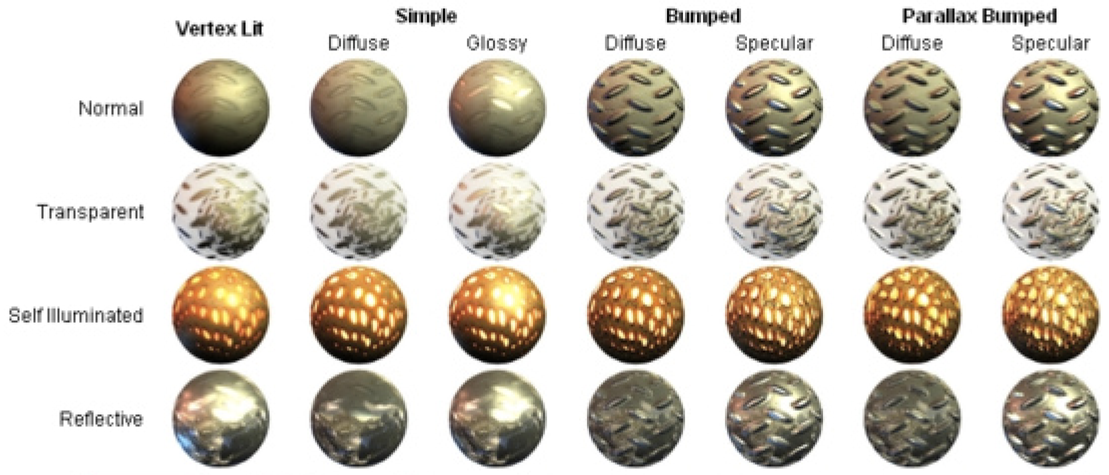
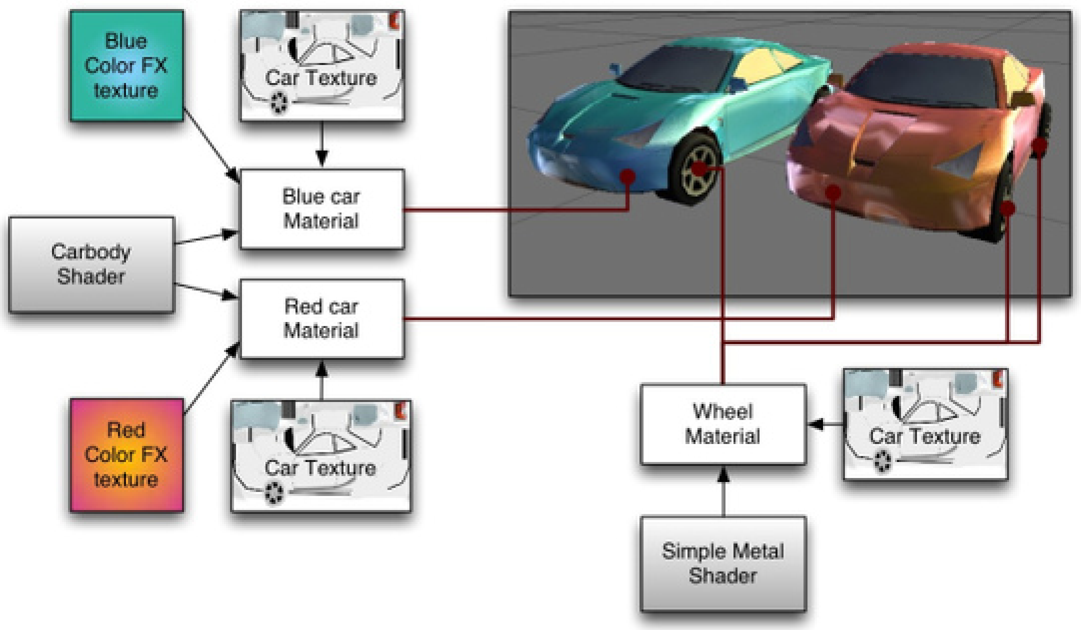
Graphics shaders

Mike Hergaarden
January 2011, VU Amsterdam





Abstract
Shaders are the key to finalizing any image rendering, be it computer games, images or movies. Shaders have greatly improved the output of computer generated media; from photo-realistic computer generated humans to more vivid cartoon movies. This paper introduces shaders by discussing the theory and practice.

Introduction

A shader is a piece of code that is executed on the Graphics Processing Unit (GPU), usually found on a graphics card, to manipulate an image before it is drawn to the screen. Shaders allow for various kinds of rendering effect, ranging from adding an X-Ray view to adding cartoony outlines to normal output.

The history of shaders starts at LucasFilm in the early 1980's. LucasFilm hired graphics programmers to computerize the special effects industry [1]. This proved a success for the film/ rendering industry, especially at Pixars Toy Story movie launch in 1995. RenderMan introduced the notion of Shaders;

> *"The Renderman Shading Language allows material definitions of surfaces to be described in not only a simple manner, but also highly complex and custom manner using a C like language. Using this method as opposed to a pre-defined set of materials allows for complex procedural textures, new shading models and programmable lighting. Another thing that sets the renderers based on the RISpec apart from many other renderers, is the ability to output arbitrary variables as an image—surface normals, separate lighting passes and pretty much anything else can be output from the renderer in one pass."* [1]

The term shader was first only used to refer to "pixel shaders", but soon enough new uses of shaders were such as vertex shaders and geometry shaders were introduced, making the term shaders more general. Shaders required the addition of the programmable pipeline and later on required another change in videocard hardware. These developments will be further examined in the following sections.

From the fixed-function pipeline to the programmable pipeline

Before exploring the latest shader types and possibilities it is important to provide some more background on shaders itself. To this end we'll look at the graphics pipeline process to put shaders into perspective.

In the 1980s developing computer graphics was a pain; every hardware needed it's own custom software. To this end OpenGL and DirectX were introduced in 1992 respectively 1995. OpenGL and DirectX provide an easy to use graphics API to provide (hardware) abstraction. Then in 1995 the first consumer videocard was introduced. The graphic APIs in combination with the new hardware made possible real time (3D) graphics and games. There was not enough support for shaders yet. At this time the graphics APIs and video hardware only offered the fixed-function pipeline (FFP) as the one and only graphics processing pipeline. The FFP allowed for various customization of the rendering process. However the possibilities are predefined and therefore limited. The FFP did not allow for custom algorithms. Shader a-like effects are possible using the FFP, but only within the predefined constraints. Examples are setting fog color, lightning etc. For specific image adjustments such as we want to do using shaders, custom access to the rendering process is required.

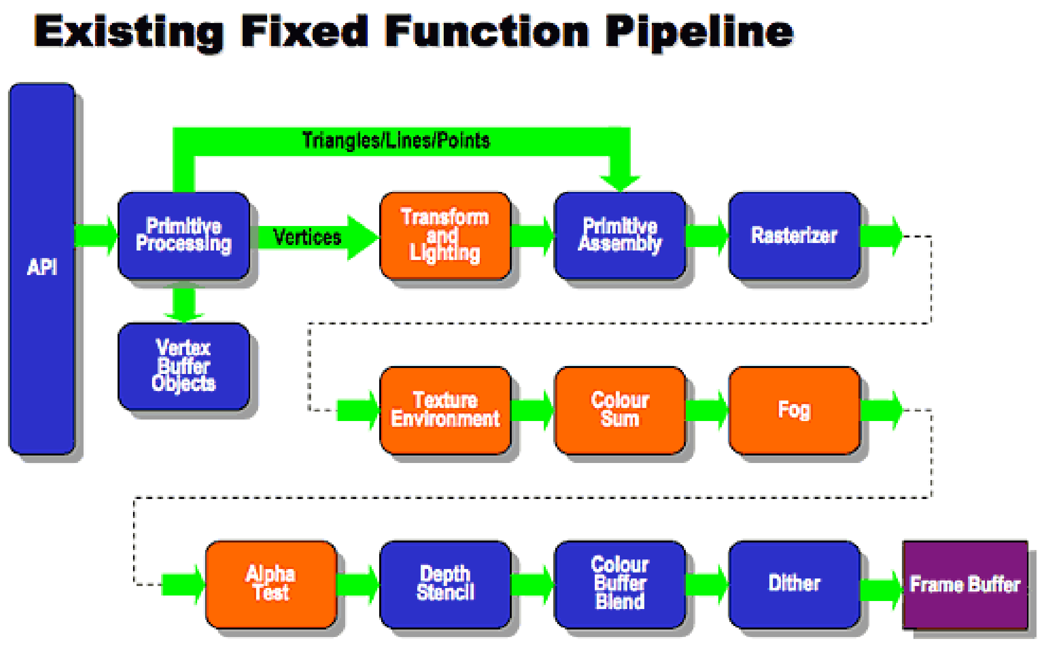


Figure 1: FFP in OpenGL ES 2.0 (http://www.khronos.org/opengles/2_X/)

In 2001 the GeForce3/Radeon 8500 introduced the first limited programmability in the graphics pipeline and the Radeon 9700/GeForce FX brought full programmability in 2002 [2]. This is called the programmable pipeline (PP), see figure 2. The orange boxes in figure 1 and 2 clearly show the difference between the FFP and the PP. The previously fixed-functions are replaced with general phases in which you can modify the data however you want; The PP allows you to do whatever you can program. You're in total control of every vertex and pixel on the screen and by now also geometry. However, with more power comes more responsibility. The PP required a lot more work as well. Shader languages, which we'll discuss in a later section, help to address this problem.

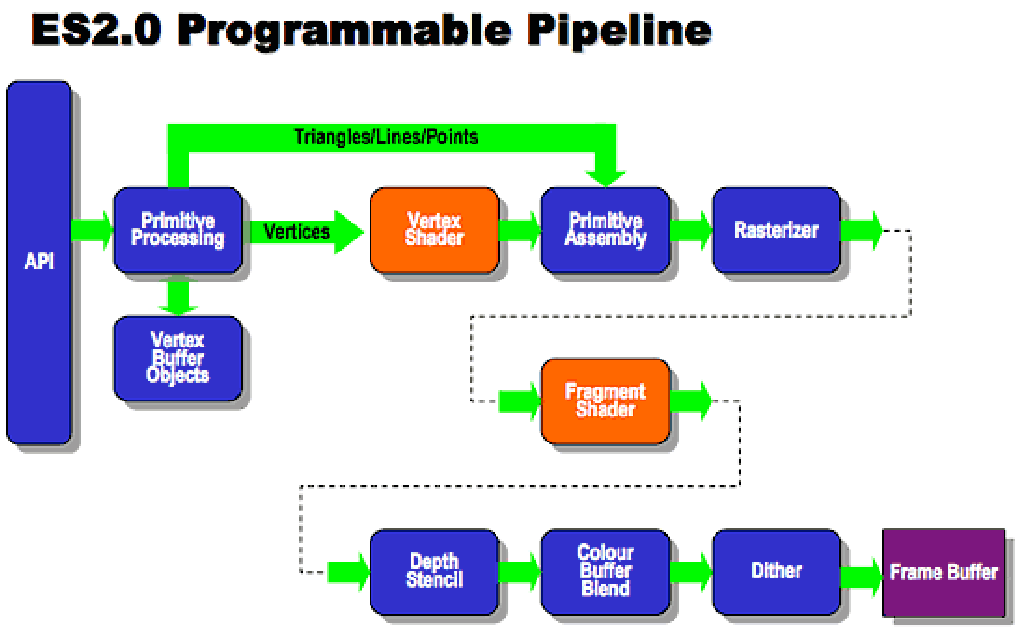
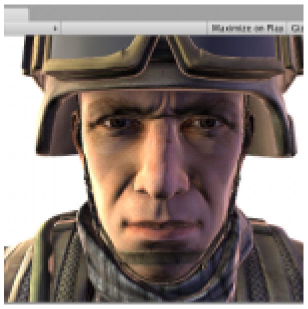


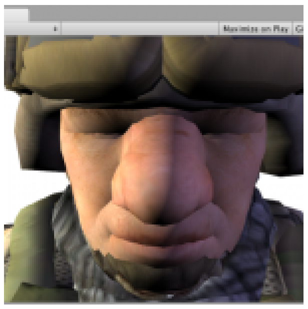
Figure 2: PP in OpenGL ES 2.0 (http://www.khronos.org/opengles/2_X/)

Types of shaders

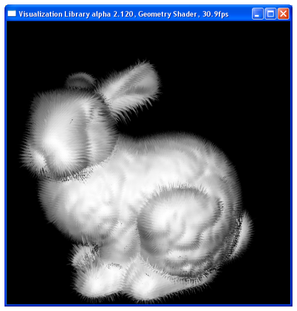
Originally shaders only performed pixel operations; taking a single “flat” image, operations are done
based on it’s pixel. this is what we now call pixel shaders. The term shader is now a more general term to describe any of the following shaders;



Pixel shaders(DirectX) / Fragment shaders(OpenGL)
Calculate the color of individual pixels. The image on the left has various applied effects such as rim lightning and normal mapping [6]. Note that DirectX calls this pixel shaders and OpenGL calls this fragment shaders.



Vertex shaders
Modifies a vertices position, color and texture coordinates. To the left an example is shown where the vertices are moved along their normals [6]. Note that vertices cannot be added, that would fall under the geometry shader.



Geometry shaders
The newest of the three, can modify vertices. I.e. procedurally generated geometry. An example is shown on the left, where the bunnys has polygonal fur genereated by the geometry shader [7].

Unified shader model

While the previously mentioned shader types are seperate entities in OpenGL and DirectX, a “quite recent” (2006) change by nVidia introduces the Unified shader model hereby merging all shader types into one [8][9]. This unified shader model consists of two parts: A unified shading architecture and a unified shader model.

In DirectX the Unified shader model is called “Shader model 4.0”, in OpenGL it’s simply “Unified shader model” . The instructions for all three types of shaders are now merged into one instruction set for all shaders. To this end, the hardware had to undergo some changes as well. The hardware used to have separate units for vertex processing, pixel shading etc. To accomodate for the unified shader model a single shader core and processors were developed. Any of the independent processors can handle any type of shading operation [14]. This hardware change allows for more flexible workload distribution as it’s perfectly fine to dedicate all processors to pixel shading when no vertex shading is required.

Shader languages

Originally shaders were written in assembly, but as with any other programming fields it became clear that a higher level language was required. The three major shader languages that have been greatly adopted are:

- HLSL (DirectX)
    - Outputs a DirectX shader program.
    - Latest version as of January 2011: Shader model 5 (http://msdn.microsoft.com/directx)
    - Syntax similar to Cg
- GLSL (OpenGL)
    - Outputs a OpenGL shader program.
    - Latest version as of January 2011: OpenGL 4.1 (http://www.opengl.org/registry/#apispecs)
- Cg (nVidia)
    - Outputs a DirectX or OpenGL shader program
    - Latest version as of January 2011: Cg 3.0 (http://developer.nvidia.com/page/cg_main.html)
    - Syntax similar to HLSL

The Cg and HLSL syntax is very similar since nVidia and Microsoft co-developed these languages. Cg does have extra function to support GLSL. None of these language is ultimately the best as it depends on the targeted platform(s) and the specific features/ performance you need.

Concluding

The programmable pipeline allowed shaders to fully change rendering output instead of the limited options the fixed-function pipeline had to offer. There are three types of shaders; Vertex, Pixel and Geometry shaders. These three shaders have been unified in the Unified shader model. To program shaders three languages are available; HLSL, GLSL and Cg.
The basic theory is hereby

Shaders in practice

For a simple demonstration of a shader we look at an Cg shader, or actually, we will look at an CgFX "effect". Cg shaders are not perfectly transferrable between applications as application specific details are always embedded in Cg shaders. CgFX solves this. An effect—in the CgFX use of the term—is a shading algorithm and everything needed to utilize it that can be authored independently from the 3D application that ultimately renders the effect [1]. DirectX has it's HLSL equivalent namely Directx Fx's

Below is the code for a simple shader that colors a vertix green. As you can figure from the name, Cg code is very similar to C.

```
// This is C2E1v_green from "The Cg Tutorial" (Addison-Wesley, ISBN
// 0321194969) by Randima Fernando and Mark J. Kilgard.  See page 38.
struct C2E1v_Output {
  float4 position : POSITION;
  float3 color    : COLOR;
};

C2E1v_Output C2E1v_green(float2 position : POSITION)
{
  C2E1v_Output OUT;
  OUT.position = float4(position,0,1);
  OUT.color = float3(0,1,0);
  return OUT;
}
```

To start with shaders, AMDs RenderMonkey and nVidias FX composer areessential tools. Which one to use is a matter of personal taste. The tools can be downloaded here:

http://developer.nvidia.com/object/fx_composer_home.html
http://developer.amd.com/archive/gpu/rendermonkey/pages/default.aspx

[1]http://developer.download.nvidia.com/cg/Cg_3.0/CgFX_bumpdemo_Tutorial.pdf

Conclusion

Shader technology has matured the last few years, both the software and hardware. However this does not mean we'll see less new developments from now on.
Shaders are an essential piece of the rendering process to finish an image/movie or game. In games shaders even allow new gameplay, think of X-rays or applications of geometry shading.
Related fields that undergo a lot of movement are parralel (shader) processing, moving processes from the CPU to the GPU (GPGPU). As hardware keeps improving we will only see more and more uses of shaders appearing, especially on mobile devices where the hardware support is still limited.

References

Comments will be added per reference

[1] RenderMan shader history
http://wiki.cgsociety.org/index.php/Renderman

[2] The Fixed-Function pipeline
http://www.cis.upenn.edu/~suvenkat/700/lectures/2/Lecture2.ppt

[3] Introduction to Shaders
http://gamedevelopment.com/blogs/AtulSharma/20090318/937/Introduction_to_Shaders.php
simple intro to shaders..

[4] How things work: How GPUs work. DAvid Luebke, NVIDIA Research, Greg Humphreys, University of Virginia.
http://www.cs.virginia.edu/~gfx/papers/pdfs/59_HowThingsWork.pdf

[5] A Crash Course in HLSL, Matt Christian
www.insidegamer.org/presentations/CrashCourseInHLSL.ppt

[6] Unity 3 technology – Surface Shaders
http://blogs.unity3d.com/2010/07/17/unity-3-technology-surface-shaders/

[7] Visualization Library; A lightweight C++ OpenGL middleware for 2D/3D graphics.
http://www.visualizationlibrary.com/jetcms/node/4

[8] Common-Shader Core (DirectX HLSL)
http://msdn.microsoft.com/en-us/library/bb509580(VS.85).aspx

[9] NVidia's new GPU Architecture
http://www.extremetech.com/article2/0,1697,2053309,00.asp
Highlighting NVidia's new Unified shader model GPU Architecture

[10] Shaders for Game Programmers and Artists - SEBASTIEN ST-LAURENT
Great shader introduction book

[11] Shader visualisations, easy shader tutorial
http://jerome.jouvie.free.fr/shaders/Shaders.php

[12] Introduction to Shaders, Marco Benvegnù
http://www.benve.org/Download/Introduction%20to%20Shaders.pdf

[13] Exploiting the Shader Model 4.0 Architecture
http://researchweb.iiit.ac.in/~skp/papers/gfxSM4.pdf

[14] GeForce 8 introduction FAQ
“What does unified shader core mean?”
http://www.nvidia.com/object/geforce_8600_8500_faq.html

Appendix A: Shading tools

RenderMonkey
by ATI
http://developer.amd.com/archive/gpu/rendermonkey/pages/default.aspx

Cg Toolkit
by nVidia
http://developer.nvidia.com/object/cg_toolkit.html

FX Composer
by Microsoft & nVidia
http://developer.nvidia.com/object/fx_composer_home.html

Appendix B: Shaders in Unity

This appendix lists some pointers on how to get started with shaders in Unity (http://www.Unity3d.com). Unity developed it's own shader language which they cal "ShaderLab". They made their own wrapper to allow seamless integration in Unitys (i.e. using the built in materials). For the "real" shader operations ShaderLab includes Cg/HLSL code. GLSL is also supported but since this will not compile for Cg/HLSL it is not advised to be used unless you know you're only targetting GLSL platforms.

Hereby an example of a shader that makes an object entirely white and enables lambert lightning.

```
Shader "Example/Diffuse Simple" {
  SubShader {
    Tags { "RenderType" = "Opaque" }
    CGPROGRAM
    #pragma surface surf Lambert
    struct Input {
       float4 color : COLOR;
    };
    void surf (Input IN, inout SurfaceOutput o) {
       o.Albedo = 1;
    }
    ENDCG
  }
  Fallback "Diffuse"
 }
```

http://unity3d.com/support/documentation/Components/SL-SurfaceShaderExamples.html

Video tutorials:
http://unity3d.com/support/resources/unite-presentations/shader-programming-course
http://unity3d.com/support/resources/unite-presentations/shader-tricks-in-game-production

Shader examples:
http://unity3d.com/support/resources/example-projects/shader-replacement

Documentation:
http://unity3d.com/support/documentation/Manual/Shaders.html
http://unity3d.com/support/documentation/ScriptReference/Shader.html
http://unity3d.com/support/documentation/Components/SL-Reference.html

Built-in shader describitions:
http://unity3d.com/support/documentation/Components/Built-in%20Shader%20Guide.html

About Unity rendering:
http://unity3d.com/support/documentation/Components/Rendering-Tech.html