Shader Creation Case Study: Is Visual Programming The Future?

Marcus Nygren, marny568@student.liu.se
TNM084 Procedural images, M.Sc. Media Technology,
Department of Science and Technology, Linköping University

5 februari 2015

1 Introduction

Programming procedural textures are becoming increasingly common, in part thanks to faster GPU's [2]. This raises the question what method to use when creating a procedural texture. Today a designer wants to use a graphical user interface (GUI) while a programmer wants to write code. Written by a media technology student, this report investigates if intuitive and powerful tools for visual programming can change that.

In a case study, the same visual result has been sought after by (1) coding in GLSL, (2) designing in Unity 5.0's Standard Shader, and (3) programming visually using Shader Forge. The three methods are highly modern and fully supported in Unity, which is the reason why that game engine has been chosen.

The report starts by highlighting the specific problems programmers and designers face when creating shaders when coding and using a GUI respectively. A short background of visual programming is aslo given. In chapter 4, modern tools are chosen that correspons to the three different methods of creating shaders.

The rest of the report focuses on the actual case study. The work preparing for the case study is presented, in order to varify that the results of the case study can be reliable. Then, the work process and result is presented. In the discussion, the experienced advantages and disadvantages of each method are explained. Lastly, conclusions and predictions about the state of shader programming are laid out.

2 The art of creating shaders

Most commonly a designer has more visual talent than technical expertise, and a programmer has more technical competence than visual talent. But as consumers expectations increase on technical *and* artistic excellence, the demand for a silver bullet that blends the two worlds increases. Could a node-based shader editor be the answer?

2.1 About coding

Shader coding is complicated. It takes a lot of boilerplate code, you have very limited error handling, and it is often a complex task learning how to code your own shaders even for experienced programmers. However, since the execution can run on the GPU shaders can be very advanced, and can produce visually stunning and technically very advanced results.

2.2 About using a GUI

Graphical user interfaces makes it easy for people independently of coding expertise to be able to build shaders with immediate visual feedback. But compared to programming shaders, the operations possible to manipulate the shader is much more limited. But for most materials in a game production, programming might not be neccessary due to the nature of the materials. There are however other problems as well.

Because an interface limits what the user can modify, shaders created across different projects can unintentionally share the same look and feel. Professionals and consumers may be able to identify that several projects use the same engine, due to materials sharing similiar characteristics and limitations.

2.3 Visual programming

An ideal visual node-based shader editors allows you to program your own shaders using a visual and intuitive environment. The node structure allows you to connect different programming blocks into a fully functioning program, producing e.g. a shader. There are several examples of modern shader editors. They have existed for quite some time in modelling software like 3Ds Max, Autodesk Maya and Unreal Engine. It is still however more common producing shaders with code or a graphical user interface. In this report, we take a look at one of the newest tools called Shader Forge.

3 Modern techniques in Unity

The previous chapter presented three different ways of creating shaders: via writing code, using a graphical user interface, and visual programming. All of these techniques have modern equivalents in Unity, which is the most common game engine in the industry [13]. Below is a brief description and motivation for why the following programming languages and software are relevant for the case study.

3.1 OpenGL and GLSL

OpenGL is decribed as the "the industry standard for high performance graphics" [7]. GLSL is the language used when writing shaders within the OpenGL graphic pipeline. The OpenGL Shading Language, or GLSL, is based on the syntax of the C programming language and is designed "to give developers more direct control of the graphics pipeline" (Wikipedia, 2015).

3.2 Shader Forge in Unity

Shader Forge is a visual node-based shader editor for Unity first released in 2013 [4], inspired by its Unreal Engine equivalent. Its goal is to make "the science behind the magic of shader creation accessible to artists" [5]. In August 2014, Shader Forge won Unity's Tech Achievement category during the annual Unity Awards [11].

3.3 Unity 5.0's Standard Shader

For comparison, the same result has been tried in the Pre-Order Beta of Unity 5.0. "The Standard Shader" is the new default visual editor for creating materials. It is also a highlighed feature of Unity 5.0.

The promise of The Standard Shader is that it "can be used to make such a wide variety of materials it's easily possible that this one shader can make every material in a given project."

4 Pre-study

In order to make an inbiased case study, the first part of my work was to research and get thorough hands-on experience with the Standard Shader and Shader Forge. The goal was to match the amount of experience and theory gained about GLSL in the course TNM084 Procedural Images, including the two laborations (circa 6 hours per laboration). I did not have exessive knowledge about GLSL before the course. Most of the work was undoubtably spent learning Shader Forge, translating a WebGL halftone shader into a node-based shader.

4.1 Unity 5.0's Standard Shader

Since Unity 5.0 is still in a Pre-Order Beta, the documentation is extremely sparse and not a lot of support is available. The current available resource for learning about the concepts of the Standard Shader is the official beginner tutorial (9 minutes) [9]. I mostly tried the tool for myself so I got an understanding for what level of custimization was possible. The more advanced runthroughs available online focuses on Physics Based Shading (PBS). The advantage of PBS was tried by reconstructing some of the scenes from a talk during the Unite 2014 conference [10].

In Unity, PBS is descibed as a technique to simulating realistic interactions between materials and light. PBS has only recently become possible in real-time graphics. The largest benefits coming in scenes where lighting and materials need to be displayed intuitively and realistically. [8]

As the case study focused solely on a single texture, the importance of PBS when creating procedural textures (independent of method) can not be justifiable concluded in this report. This should be considered in the result and discussion, when comparing using the Standard Shader with the other shader creation tools.

4.2 Shader Forge

To get to the same level of knowledge in Shader Forge, I first read the documentation [6] about the different nodes. Then, I looked at the talk from Unite 2014 [5]. After this, I started creating my own shaders from scratch, and modifying existing ones, in order to get familiar with the tool and thinking.

The ultimate training was trying to translate a WebGL halftone shader [3] step by step into Shader Forge. It took a big effort trying my best to translate all code into nodes, especially since I wanted ot explore how to implement algorithms in Shader Forge versus using the built-in code node. The result can be seen in figure 1. Due to lack of good documentation, it took a lot of time and it is still unclear how the author wants you to use Shader Forge in more complex situations like the one tried. It was great but painful training for experiencing the advantages and disadvantages of visual programming using Shader Forge. I learned a lot, and got increased fluency in the program, being transferrably also in doing more simple shaders.

One of the things that caused the most frustration was when there was not an obvious couterpart to a data structure. Instead of a matrix node, Shader Forge wants you to use a predefined node like "Rotator". Instead of a vector node, Shader Forge wants you to append values to each other. This works and is probably a choice in favor of artists, but as a programmer it is an unneccesary souce of frustration. With a bit of creativity, it is almost always possible to find a way around a problem in Shader Forge, howerver.

Another thing that takes time to wrap your head about is in what order to connect blocks. When creating a long expression, you need to be careful so that nodes are connected in the right order. When translating the halftone shader code into nodes, it rarely worked translating the code into nodes in the way you read the code. It is similar to when you are calculating a mathematical expression with a lot of parentheses: you work your way out and make the connections between parantheses as a last step.

During the training, I got increasingly more fond of the tool. I got quicker, but also more aware of its limitation and when coding would have been preferrable. Since there was no built-in smoosthep node, I started by doing one in nodes. What is a very simple expression to write in code, took an hour to do graphically because every mathematical operation needed a new node. Then, I found that there was a code node. It was to hard to understand how it worked at first, but then I could replace my 15 nodes with calling the build-it HLSL function smoothstep.

Using the code node was frustrating at first because of the lack of documenation. Nowhere could I find that the code has to be written in HLSL and not GLSL, so until I discovered that, I got compilation errors for what I thought was no reason, and I was not shown any error explanations.

5 Reaching the desired end result

The aim was to reproduce the shader result achieved by Linnea Mellblom and me during Lab 2 of TNM084 Procedural textures, see figure 2.

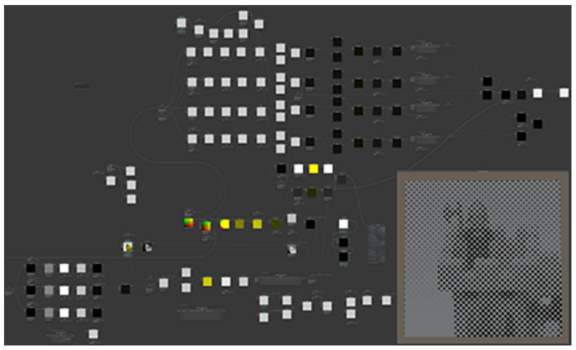


Figure 1: Halftone shader in Shader Forge. The image displays all nodes used to reproduce step 1-10 of Gustavsson's tutorial, including CMYK color recreation. The dots you see have variable radiuses based on a 2D texture, with anti-aliasing implemented.
The nodes for Step 8 did not produce the right visual result, which is why the end result is not in color. A lot of trial and error has been done to get it to work, but it seems like Shader Forge has a bug when combining certain nodes with the code node.

5.1 Shader Forge

It was surprisingly effortless implementing all the basic components: a 2D texture, connecting the alpha channel to a vertex offset, and replacing the original water with a noise node which was animated in a similiar fashion to the GLSL counterpart. The main difference is the complexity of the noise: the noise node of Shader Forge is a very naive implementation, whereas Gustavsson's simplex noise code was used in the GLSL version. This is painfully apperent. As a result, I decided to replace the node node with a code node. Instead of needing to take the

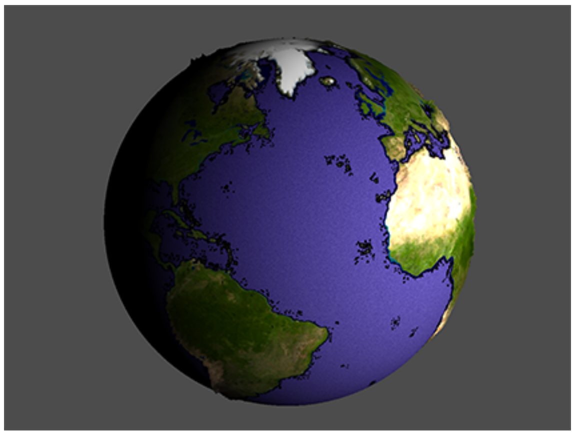


Figure 2: Earth produced in GLSL, with height map, and water produced via procedural noise.

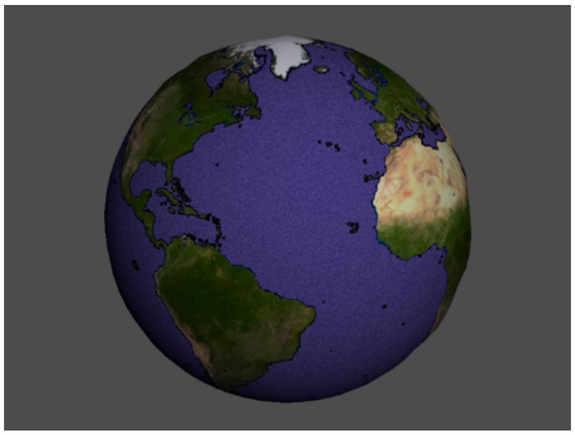


Figure 3: Earth produced in Shader Forge, with height map, and water produced via procedural noise.

time to translate the simplex noise code from GLSL to HLSL (the only supported language in Shader Forge), I used existing code from the Shader Forge wiki, FBM noise [1]. This is still a very naive and limited implementation of noise compared to Perlin or Simplex noise, and it's painfully apperent in the result, see figure 3. The same problem was apparent when producing the halftone shader in chapter 4.1.2.

5.2 The Standard Shader

The Standard Shader makes it effortless to import a 2D texture, however the rest of the visual core elements are very hard to get. Either, we need to prepare the effects via scripting and then modify the material via the Standard Shader, or we are limited by the customizations the shader editor allows us to do. For example, there is no displacement map, only a height map. Using the alpha channel from the 2D texture where the height information is stored, thus produces a strange visual result. The 2D texture remains stationary and the mapping us pushing the texture outward visually.

Replacing the water from the texture with another color is not possible without modifying the image beforehand in a image editing software such as Adobe Photoshop. The Standard Shader does have support for secondary maps, but even so it would not have been possible to use animated noise to simlate water.

Using only the Standard Shader, the most similar result obtained was the 2D texture with the height map having only a minimal impact, somewhat simulating a 3D emboss. See result in figure 4, and the custimization that the Standard Shader allows in figure 5. It is obvious that for the desired visual result, the Standard Shader did not provide the functionality needed.

6 Discussion

6.1 Traditional shader programming

While programming the shader, it is smooth when everything goes as planned. What slows

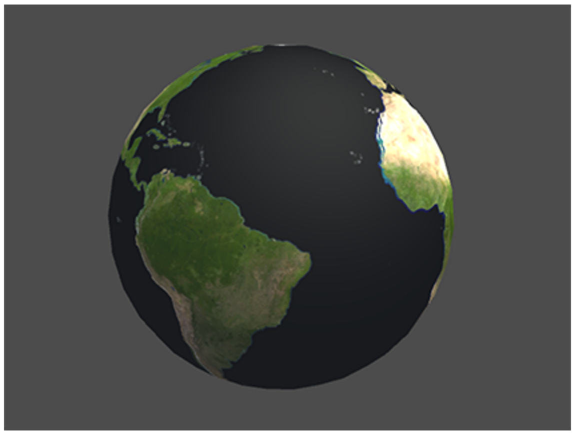
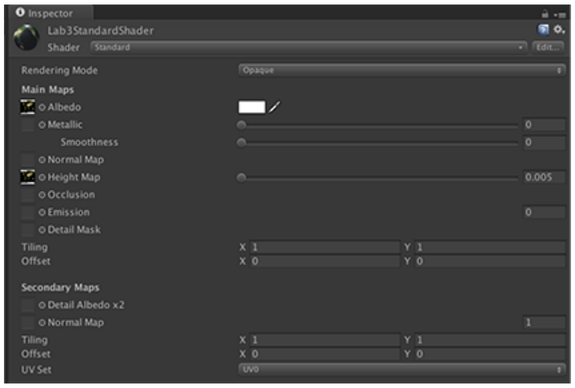


Figure 4: Result using the Standard Shader in Unity 5.0.



```
Inspector
Lab3StandardShader
Shader  Standard                    Edit...
Rendering Mode      Opaque
Main Maps
  Albedo
  Metallic                          0
      Smoothness                    0
  Normal Map
  Height Map                        0.005
  Occlusion
  Emission                          0
  Detail Mask
Tiling              X 1       Y 1
Offset              X 0       Y 0

Secondary Maps
  Detail Albedo x2
  Normal Map                        1
Tiling              X 1       Y 1
Offset              X 0       Y 0
UV Set              UV0
```

Figure 5: Custimizations using the Standard Shader in Unity 5.0.

down programming is that errors often occur in the code, and especially with GLSL it can sometimes be hard to find what the error comes from. Is it a syntax error, a technical detail that has been misunderstood, or something else? It is hard to get into shader programming with syntax and knowing what is needed to get a certain result. It is a lot of trial and error to get your code to work as expected. Also, when programming shaders you want to spend the maximum amout of time writing the defining part of the shader, but you take quite some time writing code for dependencies in order to make the shader work. In Shader Forge, you only have to program the defining parts, and Shader Forge assumes your dependencies.

There are some territories where traditional programming is still superior to visual programming. There are no limits as to what you can do, and if the shader consists of a lot of algorithms, this is build more easily in code than graphically.

I like that my knowledge from shader programming is transferrable to visual programming. Is it faster or slower writing shaders with code? It depends on the user's experience with shader programming and what you are going to build.

6.2 Unity 5's Standard Shader

The Standard Shader is marketed as "artist-driven shading technology". The workflow is designed to be as pared back and intuitive as possible. Therein lies also its biggest problem. Nothing may summerize Unity 5.0's Standard Shader as accurately as their own words: "Use it across 95% of the materials in your game" [12]. The Standard Shader can produce stunning procedural textures, but only within the scope that Unity has deemed most appropiate. It is evident from the case study that the Standard Shader will not change the behaviour when a programmer chooses to use a material editor and when he or she decides to code.

A lot of functionality is missed in the Standard Shader, for example being allowed to have procedural noise and being able to change it's parameters, having a bump map option, etcetera.

6.3 Shader Forge

The Shader Forge node tree is clean, provides a great overview without the need of scrolling, and is quite easily understandable even by a programmer not familiar with Shader Forge. A designer might be confused by the purpose of some of the technical nodes (like "Lerp" for linear interpolation). Thanks to seeing the visual output after each node connection, the designer can still get an immediate feeling for how all nodes contribute to the end result however. Thanks to this, the designer can be confident daring to change the input for the Lerp node, for example slowing down the time of the water animating, etcetera. This is percieved as a big strenght, compared to coding. See figure 6 to see the node overview in Shader Forge.

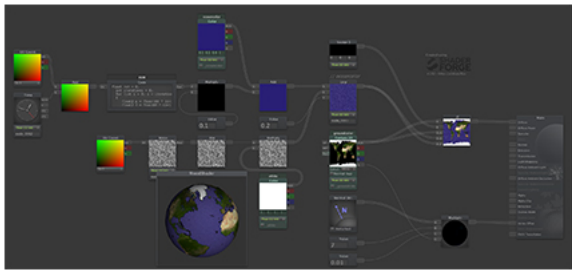


Figure 6: Node tree for the earth texture in Shader Forge.

Also, if the code node had not been used, there would have been no problems typically associated with shader programming like syntax errors or linking files and functions. It is very obvious when used with Shader Forge what nodes expects what input.

In summary, it was a surprisingly fun and quick process imitating the result of the GLSL example, and the result is almost as visually appealing (the limited noise function prevents the water to look equally realistic). Since no algorithms were present that didn't contribute immediately to the visual result, Shader Forge fitted the project very well.

7 Conclusion

As a programmer, I appreciate Shader Forge's effort to appeal to programmers and artists alike. When Unity says "High-End Shaders For All" about the Standard Shader, they mean "High-End Shaders for Artists", while Shader Forge means "Shaders For Artists and Programmers".

How should you combine code and graphical nodes when creating shaders? This is something I now can answer. A node with lack of customization (e.g. the noise node) is a good reason to go from a node to coding. Going from code to nodes is good when you have use for visual feedback step-by-step (for example, step 1 in the halftone shader tutorial described in Chapter 4.2. Code is superior during purely mathematical operations (for example anti-aliasing), where visual feedback is not necessary and helpful until the end of the algorithm.

I would like to continue to use Shader Forge in the future. However, there are some limitations that need to be fixed. The integration between using nodes and using code in Shader Forge needs to be seemless and intuitive, and it should be easier to get an overview in a large project. In summary, this is what is needed in order for Shader Forge to be useful for professional and larger projects:

1. A much improved code node - support linking to HLSL/GLSL files, and have a good compiler that gives error messages (right now, everything breaks if something is wrong, with no other feedback)

2. The ability to pack together nodes into maps or another kind of abstraction, so larger node projects can still keep a good overview and doesn't feel messy.

With these suggestions implemented, Shader Forge would be powerful enough to have more strenghts than weaknessess over traditional programming when creating shaders. The tool shows a lot of potential, and it open doors for artists and programmers alike.

In summary, a node-based shader editor can be powerful enough to replace traditional programming, as long as there is good support for custom code when visual programming is not preferrable. Shader Forge is a very good tool, and does takes away some of the most annoying aspects of programming shaders.

References

[1] Jason Booth. Procedural noise. `http://acegikmo.com/shaderforge/wiki/index.php?title=Procedural_Noise`, January 2015.

[2] Stefan Gustavson and Ian McEwan. Procedural gpu shading ready for use. 2014.

[3] Stefan Gustavsson. Webgl halftone shader: a step-by-step tutorial. `http://webstaff.itn.liu.se/~stegu/webglshadertutorial/shadertutorial.html`, January 2015.

[4] Joachim Holmer. [release] shader forge - a node-based shader editor for unity. `http://www.polycount.com/forum/showthread.php?t=123439`, July 2013.

[5] Joachim Holmer. Unite 2014 - shader forge. `http://youtu.be/WMHpBpjWUlY`, August 2014.

[6] Joachim Holmer. Shader forge nodes. `http://acegikmo.com/shaderforge/nodes/`, January 2015.

[7] OpenGL. Opengl - the industry standard for high performance graphics. `https://www.opengl.org`, January 2015.

[8] Unity. Physically based shading in unity 5: A primer. `http://blogs.unity3d.com/2014/10/29/physically-based-shading-in-unity-5-a-primer/`, October 2014.

[9] Unity. The standard shader. `http://unity3d.com/learn/tutorials/modules/beginner/5-pre-order-beta/standard-shader`, October 2014.

[10] Unity. Unite 2014 - mastering physically based shading. `https://www.youtube.com/watch?v=eoXb-f_pNag`, August 2014.

[11] Unity. Unity 2014 awards. `https://unity3d.com/awards/2014/winners`, December 2014.

[12] Unity. High-end shaders for all. `http://unity3d.com/5`, January 2015.

[13] Unity. Unity - fast facts. `http://unity3d.com/public-relations`, January 2015.