Issues In Introducing Micro-programmable Graphics Hardware Into the Animated Production Process

Yahya H. Mirza
Aurora Borealis Software
8502 166th Ave. NE
Redmond, WA, 98052 USA
yahya_mirza@hotmail.com

ABSTRACT

Today, a great opportunity is becoming available for content creators interested in near cinematic quality interactive 3D rendering. With the recent introduction of floating point calculations in graphics hardware, the functionality required to approach the capabilities of Pixar's Renderman shading language in real time are just now starting to become achievable. The mechanism being used to attain this goal is multi-pass rendering techniques enabled through compilation technology. A key player in this arena is Microsoft with their DirectX 9.0 API, which for the first time introduces many new real time capabilities which were once exclusively in the realm of high-end offline software renderers.

Keywords
Cinematic Productions, Micro-programmable graphics hardware, DX9, High Level Shading Language, Shader Effects, Procedural Rendering, Open GL2.0, Open GL Shading Language, C# and .NET.

"Delivery is not the problem, movies are not interactive today. Generating a high quality feature film in real time is not doable in the next 2-3 films."

Dana Batali

Pixar Animation Studios

1. INTRODUCTION

This paper will begin by taking the perspective of a computer animated production company interested in introducing micro-programmable graphics hardware into the animated production process. The aim will be to take a pragmatic view of how micro-programmable graphics hardware can actually be used practically, what some of the current limitations are, and what needs to be addressed to make micro-programmable solutions truly useful in a animated production setting. Throughout the paper concrete production scenarios will be used to illustrate the technical issues involved.

1st Int.Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing

February 6-8, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press
ISBN 80-903100-3-6

Figure 1. Aurora Borealis's "Elfin Song"

The aim will to explain the key technical issues involved in the micro-programmable approach to graphics. I will then illustrate how the DirectX API to a certain extent "virtualizes", these graphics hardware solutions, through a standard graphics instruction set. Finally, I will explore some issues that I believe will become important as graphics hardware evolves to more fully abstract the graphics pipeline.

2. A PRODUCTION SCENARIO

Production Elements

A typical computer animated production, can be broken down into its constituent elements. These elements may include the number of shots, the characters per shot including hero and non-hero characters, the scenes as well as the elements of the scene. Additionally, an animated production can have anywhere from 10 to 100 hero models and up to 1000 non-hero models. Furthermore a single character can have anywhere from 10 to 100 texture maps.

For example, In Disney's Dinasaur, there were 42 different characters belonging to 19 different dinosaur species. Each character model was built using 400-800 NURBS patches. To address the heavy weight nature of these models, Disney did a combination of modeling the essential features with geometry and then adding detail with Pixar's Photorealistic Renderman displacement shaders.

Production Authoring Issues

When creating animated productions, one must accept that all that matters is that the generated image looks good, not by what means that image was generated. Apodaca illustrates an important tradeoff in choosing a strategy to create a particular shot: "The question that art directors and modeling supervisors must answer is what's the cheapest way to get a look that is good enough, given the size, importance and screen time of the prop, and the relative complexity of one side's requirements on the other."

Today, in an animated production studio, the authoring process, like the rendering process is typically organized into a pipeline. Different individuals are responsible for different tasks. These tasks include modeling, texturing, lighting, and animation. In larger production companies, there generally are multiple individuals specializing in each task, thus requiring an army of individuals to complete a production or in some cases even a single shot. Consequently one must also realize that this army of "specialists" process used by the majority of the production studios is inherently costly.

Production Deployment Issues

Recently there has been much excitement in deploying an animated production over the Internet. Although eventually this scenario will be a reality, today few consider it a serious option for the near future. The fundamental roadblock is the bandwidth and memory requirements needed for the massive amounts of data that can be generated even for a single frame. For example on Toy Story 2, each frame required anywhere from 500 MB to 1GB of storage space. Perhaps minimizing the data created is a solution.

3. DATA AMPLIFICATION

A tangible opportunity that GPUs enable are the emphasis they place on tools for algorithmic or "procedural" modification and generation of content. While today's GPUs do not provide the data amplification required to handle a photorealistic computer animated production in real time, it is quite reasonable to expect this to change in the near future.

Proceduralizing Production Content

3.1.1 Procedural Geometry

Procedural geometry generation techniques have been utilized in computer animations for some time now. In the past their primary usage has been for creating secondary scene objects such as buildings, trees, rocks etc. [JAC95]. Recently, Reflex3D has created a procedural human modeler. Reflex3D's character models are generated from the inside out, and have parametrically driven descriptions for the skeleton, muscle, fatty tissue, skin, hair, and clothing.

To address the bandwidth issue discussed earlier, one approach could be to execute solutions such as Reflex3D directly on the GPU. In this scenario, rather than sending massive amounts of pre-generated geometry to the GPU, we just send a procedural description of a primary character to the GPU, and let the GPU demand generate and amplify the data as needed. To achieve this goal, the ability to generate vertices in a micro-coded GPU program is required.

3.1.2 Procedural Texturing

Many of the benefits of procedural geometry also apply to procedural texture generation. First and foremost is the compactness of a procedural texture representation, which can be KBs in size as opposed to MBs, which are required for images [EBE94]. Another important GPU capability, which could be considered an enabler for real-time cinematic productions, is the ability to algorithmically generate pixels. As in the procedural geometry scenario, having the ability to send a program to the GPU which when executed generates pixels would essentially demand generate texture values as needed.

3.1.3 Procedural Animation

A key aspect of Pixar's competitive advantage is due to their proprietary animation language ML [LEF90]. ML has been used for numerous productions such as Toy Story, Bugs Life, etc. as well as commercials including the "Listerine Arrows" [JAC95]. ML has numerous domain specific language constructs including a time-based articulated variable referred to as an avar. In the future, targeting a language like

ML to a modern GPU will not only increase productivity but also performance due to the higher data amplification capabilities enabled through ML.

4. THE DX9 VIRTUAL MACHINE

Execution Models

A fundamental challenge for a hardware accelerated graphics API is to enable developers to utilize the rapid advancements in 3D hardware, while allowing a certain amount of compatibility and uniformity across hardware solutions. In Direct X9, a programmer can either rely on a "fixed-function pipeline" or on an extensible "programmable pipeline".

4.1.1 Fixed Function Pipeline

The fixed-function pipeline itself relies on existing algorithms standardized by Microsoft Direct3D. These "fixed functions" are exposed through a set of enumeration values like OpenGL. This implies that the 'fixed-function" pipelines of both Direct3D and OpenGL utilize internal conceptual switch statements, where some of the cases corresponding to the enumeration values are hardware accelerated based on the capabilities of the graphics card on which the runtime relies.

4.1.2 Programmable Pipeline

The other more interesting approach to the hardware software co-evolution problem is what is referred to as the "programmable pipeline". In the programmable pipeline, the programmer, rather then picking a predefined enumeration value and asking Direct3D to perform the algorithm, can define their own algorithm and supply it to the Direct3D runtime, which can dynamically compile to whatever the underlying graphics hardware happens to be. In this case, the Direct3D runtime has a just-in-time (JIT) compiler, which is an explicit part of the hardware device driver.

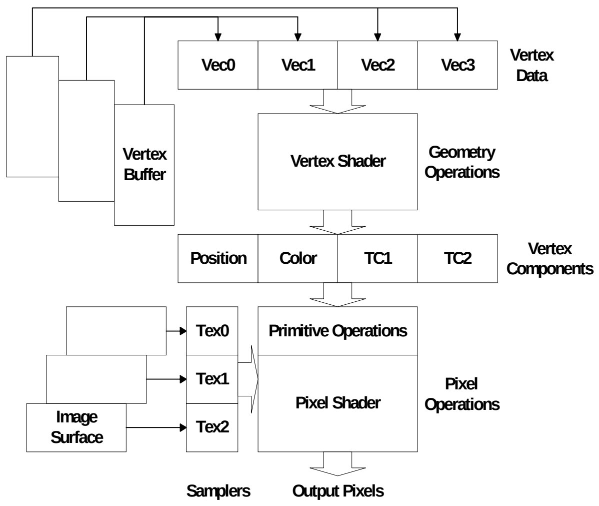

Figure 2 DX9 Programmable Architecture

Hardware vendors are responsible for providing a JIT compiler for their particular graphics hardware. Figure 1 illustrates the fundamental elements of the DX9 programmable architecture.

Virtual Processor Model

4.2.1 Vector. Register Processor Model

The Direct X processor model like preceding SIMD based vector processors is based on a vector register processor model. Although using a register model as opposed to a stack model places the complexity of register allocation to the language implementer, there are a few advantages. For example, since registers are a part of the GPU, data access in a register system can be much faster. Finally, register systems can have better code density since a memory address requires a larger representation then a register does.

Consequently the tradeoffs are very difficult to make. Often times it's a toss up, since the first thing many JIT compilers do is translate to an internal register format anyways. Although a register model was considered for .NET, a stack model was chosen to make building third party compilers for .NET easier[1].

4.2.2 Instruction Set Architecture

Currently the DirectX architecture includes two separate but related instruction sets explicitly designed for 3D graphics, one for vertex processing and the other for pixel processing. Microsoft has publicly stated that it is their intent to merge these two instruction sets in the near future.

Another interesting aspect of the DirectX instruction set is that like the .NET Common Intermediate Language, it is also typeless, with all values based on a IEEE float[4] vector. A typeless instruction set can reduces the total number of instructions dramatically. Instead of encoding the datatype into the instruction as in the JVM, in DirectX instructions operate on a float[4]. A GPUs memory limits makes packing instructions important. Making the instruction set typeless is one technique for increasing code density.

4.2.2 Extensibility Issues

As graphics hardware advances to support more colors, textures, and vertex streams, the DirectX virtual machine will need to evolve to support these new hardware features [Tay00]. Through the use of a programmable model, these new capabilities can then be accommodated through additional instructions, data inputs, as well as increasing the current resource environment a Direct X9 micro-coded program is currently constrained by. These constraints include

[1] Personal communication, George Bosworth co-architect Microsoft .NET Common Language Infrastructure.

elements of the virtual processor model such as the available registers, allowable max instruction count. Since the DirectX instruction set is encoded using DWORDs there is plenty of room for new instructions.

Algorithmic Logic Units

4.1.3 Vertex Shaders

Vertex shaders are micro-coded programs that execute on the GPU. Vertex shaders enable user programmable transform; lighting and texture coordinate modification algorithms. Functionally they replace the transform and lighting portion of the Direct3D fixed function pipeline.

Vertex shaders can be used for many things from custom lighting models to geometry deformation, to animation. For example, in the "Elfin Song" production from Figure 1, I am exploring a ubiquitous use of vertex shaders for things such as the movement of "Yvarinth's hair and clothes from the wind. The bulging of her muscles as she dances, and her shadow from the moonlight. Additional examples include the motion and deformation of the magical bubbles of light, the swaying of the grass from the wind as well as the movement of the aurora borealis across the sky.

Figure 3 illustrates the underlying vertex shader architecture. A vertex program can only modify the values of vertex components, one vertex at a time. A vertex shader cannot create vertices. A vertex shader can be used to modify the elements of a vertex such as position, normal, color, texture coordinates, etc.

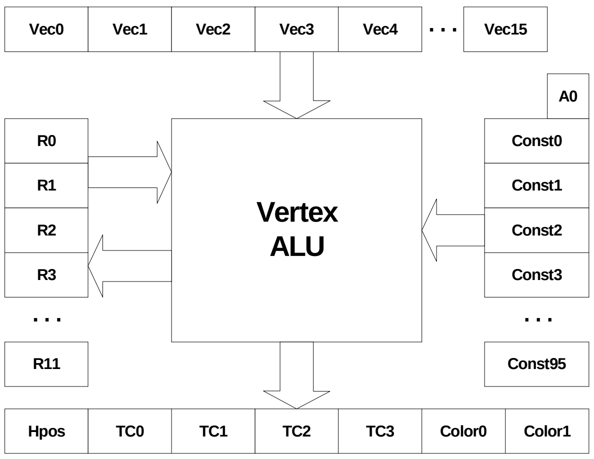

Figure 3. Vertex Shader Architecture

The vertex shader listing below declares the version followed by a position and texture coordinate register. The def instruction initializes the constant c4. Next vertices are transformed by the view and projection matrix. Finally the diffuse and texture colors are moved to the output color and texture registers.

```
vs_1_1
dcl_position v0
dcl_texcoord v8
def c4, 1, 1, 1, 1
m4x4 oPos, v0, c0
mov oD0, c4
mov oT0, v8
```

4.1.4 Pixel Shaders

Like vertex shaders, pixel shaders are also mico-coded programs that execute on the GPU. Pixel shaders enable per-pixel shading and lighting. Pixel shaders can take their input from data calculated by a vertex shader. Functionally pixel shaders replace the fixed function pixel-blending portion of the Direct3D pipeline.

Again, from Figure1, I am exploring the use of pixel shaders to create the aurora borealis effect. This effect will utilize the texture and alpha blending on several different aurora borealis shapes, which would then be blended together to create a real-time dynamic aurora borealis. Additionally, the shininess of "Yvarinth's" hair and eyebrows can be simulated with an anisotropic pixel shader.

Figure 4 illustrates the underlying pixel shader architecture. A pixel program can only modify the value of a single pixel, one pixel at a time. A pixel shader takes a texture coordinate and using the pixel shader program, coverts it into a color value. Since not all parts of the Direct3D pipeline is programmable, elements of pixel processing including stencil operations, fog blending and render target blending execute after a pixel shader executes.

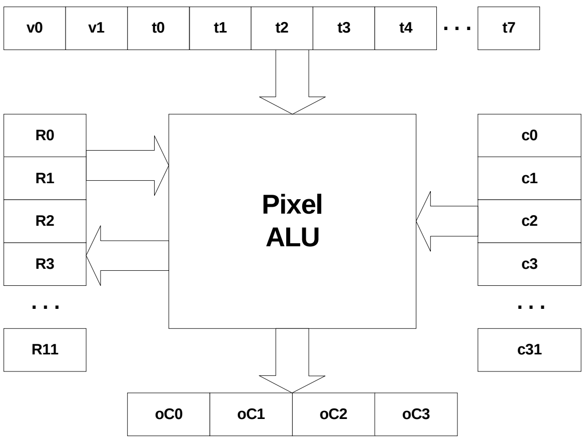

Figure 4. Pixel Shader Architecture

The pixel shader listing below declares the version. Next the texture registers t0 and t1 are loaded from stages 0 and 1. Then the texture t1 is moved into the output register r1. Finally, a linear interpolation is performed between to and r1 using the proportion defined in v0.

```
ps_1_1
tex t0
tex t1
mov r1, t1
lrp r0, v0, t0, r1
```

5. Future Challenges

Language Interoperability

A key aspect of the DirectX architecture is it's instruction set, as the DirectX instruction set stabilizes, a key opportunity that will arise will be the opportunity for production specific programming languages with their own higher level language constructs. The question that remains will be how do you interoperate amongst all these graphics modeling, shading, and animation languages?

Reuse

Currently the pervasive reuse model in shading languages is source based. This implies that as complexity of shaders increases, our shader reuse strategy of C, essentially cutting and pasting shader code will become a roadblock. In order to address code factoring issues more directly with object oriented programming techniques, we will need a polymorphic call instruction. Interestingly the HLSL has a keyword called virtual thus in the future we are sure to see a callVirtual instruction in the DirectX processor architecture.

Targeting Alternative Languages

As graphics hardware evolves to a more fully functional processor with it's own hardware stack and sufficient memory, languages that target that platform will also grow with complexity. Managing that complexity will be a key challenge. Languages that choose to target DirectX intermediate language would have to build their primitives and data structures on top of the DirectX instruction set.

6. CONCLUSION

In conclusion, given the fact that high-end computer animated productions are bandwidth limited, the fundamental problems that need to be addressed is better support for data amplification. The obvious follow on after vertex and pixel processing is to support a "primitive" processor that can procedurally generate geometry. Additionally we need to be able to procedurally generate textures as well.

7. ACKNOWLEDGMENTS

I would like to acknowledge Henry DaCosta for our recent exploration into Managed DirectX. Dana Batali from Pixar for insight into the production process. In addition, special thanks go to Jason Schellenberg of ATI developer relations for providing Radeon 9700 hardware, and Mark Lewin from Microsoft Research for providing travel sponsorship to present this paper. Finally my gratitude goes to Phillip Taylor formerly of Microsoft and now at ATI, for guiding me while I was learning DirectX.

8. REFERENCES

[Lef90] Leffler, Samuel J., Reeves, William T., and Ostby, Eben F., "The Menv Modeling and Animation Environment", 1990.

[Jac95] Jacob, Oren, Television Commercial Production at Pixar, Using Renderman in Animation Production, Siggraph 1995 Course.

[Pee00a] Peercy., Mark. S., Olano., Mark., Airey., John., and Ungar., Jeffrey., Interactive Multi-Pass Programmable Shading.

[Ref00b] Reflex3D Corporate Site, www.reflex3d.com

[Tay00] Taylor, Philip, "Programmable Shaders for DirectX 8.0", Microsoft Corporation, 2000.

[Pro01a] Proudfoot., K., Mark W. R., Svetoslav T., and Hanrahan, P., A Real-Time Procedural Shading System for Programmable Graphics Hardware, Siggraph Conference Proceedings, pp 159-170, 2001.

[Lin01b] Lindholm., E., Kilgard., M., and Moreton., Henry., A User-Programmable Vertex Engine. Siggraph Conference Proceedings, pp 149-158, 2001.

[Mal01] Mallinson, Dominic, "Benefits Of A Micro-programmable Graphics Architecture", Gamasutra, 2001. www.gamasutra.com/features/20010214/mallinson_01.htm

[Qin01c] Qung., Han Da., Tool Postmortem: Ubi Soft Entertainment's GL for Playstation 2., Gamasutra, pp 1-9, 2001.

[Pee02a] Peeper, Craig, Microsoft Meltdown UK Presentation: "DirectX High Level Shading Language", Microsoft Corporation, 2002.

[Pee02b] Peeper, Craig, Microsoft Meltdown UK Presentation: "DirectX Shader Management", Microsoft Corporation, 2002.

[Asa03] Asanovic, Krste, "Vector Processors", Department of Electrical Engineering and Computer Science, MIT, 2003.