

A Programmable Pipeline for Graphics Hardware

by
Marc Olano

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
1998

Approved by:

Advisor: Anselmo Lastra

Reader: Steven Molnar

Reader: Jan Prins

Gary Bishop

Henry Fuchs

© 1998

Marc Olano

ABSTRACT

MARC OLANO: A Programmable Pipeline for Graphics Hardware

(Under the direction of Anselmo Lastra)

This dissertation demonstrates user-written procedures on an interactive graphics machine. Procedural shading is a proven technique for off-line rendering, and has been effectively used for years in commercials and movies. During most of that time, polygon-per-second performance has been a major focus for graphics hardware development. In the last few years, we have seen increased attention on surface shading quality for graphics hardware, principally image-based texture mapping. Today, even low-end PC products include support for image textures. The PixelFlow graphics machine demonstrates that techniques like procedural shading are possible at interactive rates. PixelFlow is the first machine to run, at real-time rates of 30 frames per second, procedural shaders written in a high-level shading language.

A graphics machine like PixelFlow is a large and complex device. An abstract pipeline is presented to model the operation of this and other interactive graphics machines. Each stage of the pipeline corresponds to one type of procedure that can be written by the graphics programmer. Through the abstract pipeline, the user can write shading procedures or procedures for other graphics tasks without needing to know the details of the machine architecture. We also provide a special-purpose language for writing some of these procedures. The special-purpose language hides more details of the machine implementation while enabling optimizations that make execution of the procedures possible.

ACKNOWLEDGMENTS

I would like to thank my advisor, Anselmo Lastra, for guidance, when it was necessary; for pressure, when it was necessary; and for freedom, when it was necessary. I would also like to thank my readers, Steve Molnar and Jan Prins for their comments, and the rest of my committee, Gary Bishop and Henry Fuchs.

I would like to thank Voicu Popescu and Rob Wheeler for their great work on the pfman compiler. While they both did many things, I would like to make a special acknowledgment of Voicu's work on the memory optimizer and Rob's work on the code generator and an early version of the fixed-point sizing code. I would also like to thank Yulan Wang, Peter McMurry and Greg Pruet for their work on the shading and math support libraries. Jon Leech deserves much of the credit for ensuring that the API extensions fit into the OpenGL framework.

For their time, effort, patience, and enthusiasm, I thank everyone who used pfman, Alexandra Bokinsky, Chun-Fa Chang, Arthur Gregory, Arun Helser, Sang-Uok Kum, Renee Maheshwari, and Chris Wynn.

Finally, while I have only given individual mention to a handful of the people who worked on PixelFlow, they all deserve thanks. This dissertation would not have been possible without them. Those not already mentioned are Dan Aliaga, Greg Allen, Peter Arnold, John Austin, Randy Bailey, Tony Bennett, Vern Chi, Jon Cohen, David Ellsworth, Nick England, John Eyles, Trey Greer, Larry Harrold, Fred Heaton, Justin Heinecke, Rich Holloway, Kurtis Keller, Paul Keller, Mike Keshk, Lawrence Kesteloot, Paul Layne, Jon McAllister, Carl Mueller, Mike Myers, Bryon Nordquist, John Poulton, Krish Ponamgi, Chris Practico, Jiang Qian, Brad Ritter, Durward Rogers, Doug Schiff, Jason Smith, Mel Snyder, Steve Tell, Herman Towles, Jim Weaver, and Lee Westover.

TABLE OF CONTENTS

1 INTRODUCTION.....	1
1.1. Thesis Statement	2
1.2. Procedural techniques.....	2
1.3. Abstract pipeline	4
1.4. Procedure language.....	5
1.5. Demonstration.....	6
1.6. Organization.....	7
2 ABSTRACT INTERFACE.....	9
2.1. Prior systems	9
2.1.1. Testbeds	10
2.1.2. User-programmable systems	12
2.2. Device independence	13
2.2.1. Using a “shading” language	14
2.2.2. Abstract models	15
2.3. Abstract Pipeline	16
2.4. Pipeline stages.....	17
2.4.1. Modeling	17
2.4.2. Transformation.....	19
2.4.3. Primitives	22
2.4.4. Interpolate.....	24

2.4.5. Shading	24
2.4.6. Lighting.....	29
2.4.7. Volume and Atmospheric Effects.....	30
2.4.8. Image Warping and Filtering.....	30
2.5. Maps	31
2.6. Using the pipeline	33
3 IMPLEMENTATION OVERVIEW	34
3.1. High-level view	34
3.1.1. Applying the abstract pipeline	35
3.1.2. Parameter manager	37
3.2. Low-level view	37
3.3. PixelFlow node.....	39
3.3.1. Compiler target.....	40
3.4. Machine suitability.....	41
4 SURFACE SHADING	42
4.1. Pfman language.....	43
4.1.1. Data	43
4.1.2. Functions.....	46
4.1.3. Expressions.....	49
4.1.4. Statements.....	51
4.1.5. Antialiasing	53
4.2. Application interface.....	53
4.2.1. Shading parameters.....	54
4.2.2. Shader instances	55
4.2.3. Lights	56

4.3. Memory optimizations	57
4.3.1. Paging.....	58
4.3.2. Uniform and varying	58
4.3.3. Fixed point	59
4.3.4. Memory allocation	61
4.3.5. Memory allocation method.....	62
4.3.6. Memory allocation results	64
4.4. Bandwidth optimizations	65
4.4.1. Shader-specific maps	66
4.4.2. Bound parameters	66
4.4.3. Explicit shader parameters.....	67
4.5. Execution optimizations	68
4.5.1. Deferred shading.....	68
4.5.2. Fixed point	69
4.5.3. Math functions.....	70
4.5.4. Combined execution	73
4.5.5. Cached instruction streams.....	76
4.5.6. Identification of active shaders	78
4.6. Stages in shading.....	79
5 PRIMITIVES AND INTERPOLATION	81
5.1. Historical perspectives	83
5.2. Creating primitives	86
5.2.1. Graphics pipeline.....	86
5.2.2. Transformation.....	88
5.2.3. Interpolation	89

5.3. Application interface for arbitrary primitives	90
5.4. Language details.....	92
5.4.1. The pixel-centric view.....	93
5.4.2. Parameter types	94
5.4.3. Subdivision to other primitives	95
5.4.4. Direct rendering.....	96
5.4.5. Interpolator functions.....	97
5.5. Implementation	98
5.5.1. Primitive bounds.....	98
5.5.2. Interpolator functions.....	99
5.5.3. Code efficiency	100
5.5.4. Buffer space	102
5.5.5. Data organization	104
5.6. Demonstration of procedural primitives	105
6 USER EXPERIENCES	106
6.1. Results.....	106
6.2. Lessons.....	108
6.2.1. Early users.....	108
6.2.2. Research project	109
6.2.3. Machine details	109
6.2.4. Similarity to RenderMan.....	110
6.2.5. Development cycle	110
6.2.6. Learning curve.....	111
6.2.7. Application program interface	111
7 CONCLUSION	113

7.1. Conclusions and future research	114
7.1.1. Models	114
7.1.2. Transformations	115
7.1.3. Primitives/Interpolators.....	116
7.1.4. Shading/Lighting	116
7.1.5. Fog and atmospheric effects	118
7.1.6. Image warping	118
7.1.7. Other areas.....	118
7.2. Contributions	119
APPENDIX A: SAMPLE SHADERS.....	121
A.1. Surface shaders.....	121
A.1.1. Simple brick	121
A.1.2. Full brick shader	123
A.1.3. Rippled reflection.....	128
A.1.4. Environment map	133
A.1.5. Mandelbrot and Julia sets	135
A.1.6. Wood	138
A.2. Light.....	140
A.2.1. Light through a window	141
A.3. Primitives.....	143
A.3.1. Simple direct rendered primitive	143
A.3.2. Basic triangle without clipping.....	144
A.3.3. Subdivision primitive.....	146
REFERENCES.....	148

LIST OF FIGURES

FIGURE 1.1. SIZE AND SHAPE PARAMETERS FOR BRICK SHADER	3
FIGURE 1.2. PORTION OF CODE FOR A SIMPLE BRICK SHADER	3
FIGURE 1.3. BRICK SHADER IMAGES.....	4
FIGURE 1.4. EXAMPLES OF SHADERS.	4
FIGURE 2.1. ABSTRACT PIPELINE FOR PROCEDURAL RENDERING.	16
FIGURE 2.2. MODEL STAGE.	17
FIGURE 2.3. TRANSFORM STAGE.	19
FIGURE 2.4. BEND DEFORMATION IN 2D.....	20
FIGURE 2.5. BICUBIC FREE FORM DEFORMATION IN 2D.....	20
FIGURE 2.6. PRIMITIVE STAGE.....	22
FIGURE 2.7. INTERPOLATION STAGE.	24
FIGURE 2.8. SHADING STAGE.	24
FIGURE 2.9. LIGHT STAGE.	29
FIGURE 2.10. ATMOSPHERIC STAGE.....	30
FIGURE 2.11. WARPING STAGE.....	30
FIGURE 2.12. TYPICAL MAP.	31
FIGURE 3.1. SIMPLIFIED VIEW OF PIXELFLOW SYSTEM.....	35
FIGURE 3.2. PROCEDURE PIPELINE	35
FIGURE 3.3. PIXELFLOW MACHINE ORGANIZATION.	38
FIGURE 3.4. SIMPLE BLOCK DIAGRAM OF A PIXELFLOW NODE.....	39
FIGURE 4.1. CODE FOR A SIMPLE LIGHT.	48
FIGURE 4.2. PFMAN OPERATOR PRECEDENCE	50
FIGURE 4.3. USE OF THE ILLUMINANCE CONSTRUCT.....	52

FIGURE 4.4. TYPICAL OpenGL CODE FOR A VERTEX.	54
FIGURE 4.5. INSTANCES OF A BRICK SURFACE SHADER.....	55
FIGURE 4.6. APPLICATION CODE TO CREATE A SHADER INSTANCE.	56
FIGURE 4.7. FIXED POINT VS. FLOATING POINT COMPARISON.	59
FIGURE 4.8. EXAMPLE OF FIXED POINT SIZE DETERMINATION.....	60
FIGURE 4.9. EXAMPLE OF SSA ANALYSIS.....	62
FIGURE 4.10. GENERATED CODE	63
FIGURE 4.11. SHADER MEMORY USAGE IN BYTES.	64
FIGURE 4.12. EXAMPLE SURFACE SHADERS.....	64
FIGURE 4.13. SHADER EXECUTION TIME AND MEMORY.....	65
FIGURE 4.14. FIXED POINT AND FLOATING POINT EXECUTION TIMES.....	69
FIGURE 4.15. NATURAL LOG FUNCTION OVER THE APPROXIMATION DOMAIN.....	71
FIGURE 4.16. RELATIVE ERROR IN NATURAL LOG APPROXIMATION.	71
FIGURE 4.17. SIMD EXECUTION TIME FOR FLOATING POINT MATH FUNCTIONS.....	72
FIGURE 4.18. OUTLINE OF A TYPICAL SURFACE SHADER.....	74
FIGURE 4.19. INTERLEAVING OF SURFACE SHADERS AND LIGHTS.....	74
FIGURE 4.20. COMPARISON OF PIXELFLOW SOFTWARE STAGES AND ABSTRACT STAGES ...	79
FIGURE 5.1. EXAMPLES OF SEVERAL PRIMITIVE TYPES.	81
FIGURE 5.2. SOME TYPICAL GRAPHICS PIPELINES.	86
FIGURE 5.3. PIPELINE MODIFIED FOR PROCEDURAL PRIMITIVES.....	87
FIGURE 5.4. EXAMPLE OF PRIMITIVE-INDEPENDENT INTERPOLATION.....	90
FIGURE 5.5. OpenGL CODE FOR A TRIANGLE.	91
FIGURE 5.6. API CODE DEMONSTRATING GLRASTPARAMEXT.....	91
FIGURE 5.7. EXAMPLE OF A RASTERIZER USING GLSEQUENCEPOINTEXT.....	92
FIGURE 5.8. EXAMPLE PER-PRIMITIVE PARAMETERS (RADIUS, CENTER, AND AXIS).....	95

FIGURE 5.9. PRIMITIVE USING SUBDIVISION TO TRIANGLES.	95
FIGURE 5.10. PRIMITIVES USING DIRECT RENDERING.	96
FIGURE 5.11. PSEUDO-CODE FOR A TRIANGLE RASTERIZER.	97
FIGURE 5.12. A SIMPLE INTERPOLATOR FUNCTION.....	98
FIGURE 5.13. COMPARISON OF FIXED AND FLOATING POINT TYPES.....	100
FIGURE 5.14. CODE GENERATED BY THE PFMAN COMPILER.	101
FIGURE 6.1. BOWLING IMAGES BY ARTHUR GREGORY.....	106
FIGURE 6.2. NANOMANIPULATOR SHADERS.....	107
FIGURE 6.3. MORE NANOMANIPULATOR SHADERS.	108