Gaze Contingent GPU Shader Complexity Reduction

by

Luke Paireepinart

Algorithm Design and Analysis

Texas State University

Fall 20009

Contents

[Team 3](#_Toc248253429)

[Responsibilities 3](#_Toc248253430)

[Time Spent 3](#_Toc248253431)

[Coding 3](#_Toc248253432)

[Researching 3](#_Toc248253433)

[Debugging 4](#_Toc248253434)

[Design 4](#_Toc248253435)

[Report Writing 4](#_Toc248253436)

[Software 4](#_Toc248253437)

[Main Engine 4](#_Toc248253438)

[Shading Language 5](#_Toc248253439)

[Installation 5](#_Toc248253440)

[Running the example 5](#_Toc248253441)

[Alternate Running Method 5](#_Toc248253442)

[Hardware 6](#_Toc248253443)

[My Desktop 6](#_Toc248253444)

[Project Implementation 6](#_Toc248253445)

[Low Average Framerates 6](#_Toc248253446)

[Algorithm for Applying the Visual Sensitivity Function 7](#_Toc248253447)

[Why it works 8](#_Toc248253448)

[Difficulties Encountered 8](#_Toc248253449)

[References 9](#_Toc248253450)

[Referenced Directly In Document 9](#_Toc248253451)

[Consulted for Research 9](#_Toc248253452)

# Team

The team for this project was composed solely of Luke Paireepinart.

# Responsibilities

All responsibilities were undertaken by Luke Paireepinart.

# Time Spent

## Coding

Approximately 3-5 hours were spent actually writing the original source code for the project.

## Researching

Before the project I had no idea what was involved in writing shaders for GPUs, or even what shaders were. I spent approximately 3 weeks at the beginning of the project just researching shaders and learning the shading language CG. It took another week to learn how to use Panda3D and then I was able to implement the project fairly quickly. I estimate actual research time to around 30 hours, this includes writing test applications and reading voluminous amounts of documentation.

## Debugging

Debugging was fairly time-consuming, the shading language CG is difficult to work with. Panda3D gives you syntax errors when compiling CG but you are responsible for semantic errors, such as errors in variable access, and I had many of these initially. I would say about 8 extra hours were spent actually debugging the code, but it was spread out over many days with long processes of introspection trying to figure out what the code was doing wrong on an abstract level.

## Design

Perhaps 30 minutes were spent on design. The code is heavily based on a previous codebase (the tutorial for Panda3D) so there were no major design considerations. The only issue was where to include the supplementary files such as the models and textures.

## Report Writing

The presentation took about 7 hours to put together. This report is fairly supplemental to the presentation; most of the key information was given in the presentation. This report took 2 hours to write.

# Software

## Main Engine

I used the wonderful Panda3D engine. It is open-source, provided by Carnegie Mellon and Disney, and written entirely in C++. It has a great Python framework for actually implementing the games, but the engine itself is entirely C++ and therefore very fast. Also Panda3D is one of the only ways to get very easy GPU Shader support in Python (which is the language I wished to use for the project.)

## Shading Language

The shaders for the GPU were written in the cross-compatible CG library. HLSL is only available to Direct3D and GLSL only works with OpenGL, but CG works with both.

## Installation

The project requires only that you install Panda3D. You can download Panda3D from here[1]. Once you download it simply run the installer and install it wherever you would like. Then extract the source code zip file to wherever you would like it to be located.

## Running the example

Start a command prompt (by going to Start -> Run and typing “cmd” and hitting enter). Then you use the ‘dir’ command (which shows your current directory) and the ‘cd’ command (which changes directory) to navigate to the folder you unzipped to earlier. Once you’re there, you simply type ppython main.py to run it. ‘ppython’ is a special implementation of Python provided for use by Panda3D and so the program should run fine even if you don’t have regular Python installed (it’s included with Panda3D.) Once you run it you will be presented with a rotating screen with a normal map applied to the whole screen. Simply press up to enable the gaze-contingent area and press ‘down’ to disable it. .

## Alternate Running Method

I have uploaded a standalone installer to the website located at [2]. However, the hosting site will delete files at random intervals depending on usage, so do not expect for it to be there past 12/20/2009. I can be contacted if you would like me to provide the file directly, but it is 30 MB and thus is too large for most e-mail services to receive.

# Hardware

## My Desktop

My processor is an Intel Core2 Quad Q6600 which has a default clock speed of 2.4 ghz per core. My motherboard is an Asus P5N-D (which contains the NVIDIA 750i northbridge).

The computer also contains 4 gb of DDR2, and an ASUS version of NVIDIA’s GeForce 8800 GT graphics card. (note the 8800 and 9800 are the same card.) Hard drives are two standard 1TB 7200 RPM 32MB cache drives. The operating system is Windows XP with Service Pack 3 installed. The Panda version used is Panda3D 1.6.2.

# Project Implementation

## Low Average Framerates

The project requires that there is initially a slowdown in the frames per second so that there can be an increase at the end of the project. I attempted to do this by running at a high resolution and applying a normal map to the entire world, but the graphics card was far too fast for any of this to have an effect with a single shader. The only way to cause a noticeable slowdown (anything greater than 60 frames a second is not noticeable because my monitors only update at 60 frames per second) would be to have applied multiple shaders on every object in the scene, which was out of the scope of this project (I did not have time to create my own shaders, I had to modify existing shaders, and I could not find a source for quality, free shaders that could be modified for the project. Most shaders are proprietary and the code is not available.) So the framerate usually runs between 700 and 750 frames per second at 1680x1050 (the highest resolution my monitors support.)

## Algorithm for Applying the Visual Sensitivity Function

After attempting to apply the visual sensitivity function for a while, I realized that, for the shader I was using, there is not a way to slowly lower the applicability of the sensitivity. Instead, when rendering I check whether a pixel is within the sensitivity area, and if it’s not, the effect is bypassed (there is still implementation required for the bypass, because a fragment shader must always output a color of a pixel.) Some other shaders may have visual sensitivity application, but for most shaders, the time it would take to do an ‘intermediate’ calculation would be the same as the original calculation, and thus you would get no speedup. For example, suppose the intended effect is a vertex shader that distorts a mesh of ocean texture to make it look like waves. If the visual sensitivity function determines how rapidly the waves undulate, or the amplitude of the undulation, then the same calculations are required, they just have a fractional effect on output. To truly save computational time the effect must just not be applied at all.

Some effects WOULD benefit from the sensitivity function. For example, there are lighting algorithms that you can specify a depth to which they examine a reflection, where a further depth will result in better clarity of the image. This function could have the number of iterations specified by the visual sensitivity, but I could not find any free source of a shader that performed this operation and I am not skilled enough at shader programming to reproduce it.

## Why it works

For the longest time (until documenting the source code) I thought that the code was slowed down by the branch that was taken in the shader, but I realized that the reason I had a framerate decrease (which is a bad thing) whenever the effect was enabled was due to how the mouse position was passed to the shaders.

So in the end, for the normal map, the rendering speed was not enough to cause a significant improvement in frame rate. I believe with a more involved effect the rendering speed increase will be obvious, however.

# Difficulties Encountered

The biggest difficulty encountered was getting the screen location accessible in the fragment shader in CG. The vertex shader passes the location of each fragment shader to the fragment shader, but it is difficult to access the value. The workaround I used was to pass the position in a separate variable. I copied the old value into a new one in the vertex shader before sending. Otherwise the fragment shader would not perform correctly.

Another difficulty I encountered was that the Python is actually slower than the GPU rendering of the scene, so the camera rotation tasks and the mouse position passing are actually the limiting factor in my example. If I had enough time I would have used a much much more complicated shader so that the GPU became the bottleneck, but I ran out of time and had too many other obligations.

# References

## Referenced Directly In Document

[1] Panda3D download site, <<http://www.panda3d.org/download.php>>, accessed 12/10/2009.

[2] MegaUpload of installer, <<http://www.megaupload.com/?d=V5GTLNAM>>, accessed 12/10/2009.

## Consulted for Research

[1] CG User’s Manual, <<http://developer.nvidia.com/object/cg_tutorial_home.html>>, accessed 12/10/2009.

[2] Panda3D Documentation, <<http://www.panda3d.org/apiref.php?page=classes>>, accessed 12/10/2009.

[3] Explanation of Bumpmapping, <<http://en.wikipedia.org/wiki/Bump_mapping>>, accessed 12/10/2009.

[4] Explanation of Normal Mapping, <<http://en.wikipedia.org/wiki/Normal_mapping>>, accessed 12/10/2009.

[5] Nvidia’s Developer Resources, <<http://developer.nvidia.com/page/home.html>>, accessed 12/10/2009.

[6] IRC channel for Panda3D, <irc://irc.freenode.net | channel #Panda3D>, accessed 12/10/2009.

[7] NVIDIA GPU Programming Guide, <<http://developer.nvidia.com/object/gpu_programming_guide.html>>, accessed 12/10/2009.