A Framework for GPU Accelerated GIMP Plugins

Ajitav Sahoo (08305011)
Prashant Borole (08305022)
Sriram Kashyap (08305028)

November 12, 2008

Contents

1 Introduction

1.1 Motivation

Existing GIMP filters are driven by Scheme scripts/C programs which perform image processing on the CPU. Each filter is started as a separate process by GIMP. Therefore, although GIMP itself supports multiprocessors (for its inbuilt tools), the plugins use a single CPU core.

Most mid-range desktop machines nowadays have a decent graphics card. Graphics cards are built for inherently parallel tasks like image processing, and this gives us an edge in terms of processing speed. Proprietary solutions like Adobe Photoshop already have plugins that use the graphics hardware. Even though GIMP is the most popular open source image processing solution, it does not have a GPU based processing solution.

1.2 Problem Statement

We propose a framework that will allow users to write GIMP Plugins that perform processing on a Graphics Card (GPU). The plugins developed using our framework will run on a machine with graphics card (OpenGL 1.2+ compliant) with Pixel shader v2.0 support.

This framework has the following components:

- A gimp plug-in which acts as an abstraction between GIMP and the graphics card. This plugin will dynamically load compiled GPU shader modules, and execute them on images provided by GIMP
- A library of utilities that each shader module will be linked against
- A repository of shader modules that can be loaded by the plugin to perform image processing
- A UI framework to enable users to configure various shader parameters

2 System Implementation

2.0.1 Work Done

- Developed a plugin interface to interact with the GIMP core
- Developed a program to transfer buffers between the system memory and GPU memory, i.e. loading a texture to the graphics card
- Developed pixel shaders to be applied on the image
- Building Shader Modules to load these pixel shaders and run them on GIMP images
- We didn't modify any existing GIMP code, but implemented parts of it (using plugins) to run it on parallel hardware
- Total Time Spent: 50 hours
- The main framework consists of about 600 lines of code
- Each shader module is in itself about 150 - 250 lines of code
- We have implemented brighten, gaussian blur, emboss and whirl shaders

2.1 Design Overview

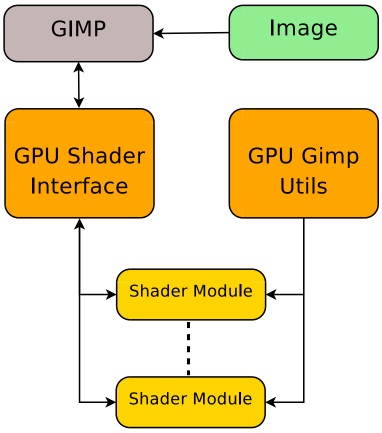

Figure 1: System overview

The system consists of 3 main parts:

- The GIMP Shader Interface
- The GPU GIMP Utilities
- Shader Modules

The GIMP Shader interface interacts with GIMP as well as the shader modules to provide an abstraction between them.

The GPU GIMP Utilities are compiled as a single library which provides useful OpenGL functionality to the Shader Modules.

The Shader Modules execute programs on a graphics card. These programs operate on the image data provided by the GIMP Shader Interface, thus performing image processing on the graphics card.

2.2 Detailed Design

2.2.1 GIMP Shader Interface

The GIMP Shader Interface is a layer of abstraction that is present between the Shader Modules and GIMP. It provides a way for shader modules to seamlessly interact with GIMP. This includes obtaining input images, shader program parameters, and displaying the final result on screen.

The GIMP Shader interface performs the following functions:

- Identify the various Shader Modules that are available
- Register these Shader Modules with GIMP. This means that each module will appear as a Menu Item in GIMP
- Dynamically load the required Shader Module when the user clicks on the corresponding GIMP Menu item
- Query the Shader Module to find out what parameters it requires
- Dynamically construct a GUI to read these parameters from the user
- Return these parameters to the Shader Module
- Convert the drawable(image) provided by GIMP into a format viable for processing on the graphics card
- Call routines from the GPU Gimp Utilities module, which will initialise OpenGL and load the texture into GPU memory
- Call the 'run' routine of the Shader Module to initiate processing
- Once the Shader Module returns, convert the resultant data into a format readable by GIMP
- Refresh GIMP image buffers to display the result

Notes: The Shader Interface is the program that is recognised by GIMP as a Plugin. The Shader Interface is compiled with GIMP headers (gimptool) and placed in the plug-ins directory of GIMP. This means that the Shader Modules are not directly seen by GIMP. The Shader Interface is responsible for telling GIMP about the various Modules available and loading the Shader Modules when required.

2.2.2 GPU GIMP Utilities

The GPU GIMP utilities are a set of functions compiled into a library that is used by both the GIMP Shader Interface, and by the Shader Modules. These utilities handle OpenGL initialisation and loading and storing textures(images) to and from graphics memory. These utilities also contain definitions common to the Shader Interface and the Shader Modules.

The GPU GIMP Utilities provide the following functionality:

- Initialise OpenGL
- Create an off-screen rendering framebuffer object (we do not render to a window, unlike most other OpenGL applications)
- Set up a floating point texture in video memory
- Write data from main memory to video memory
- Read data from video memory to main memory
- Check for any errors in the OpenGL state
- Provide error checking functionality for GLSL (GL Shader Language) programs and shaders
- Clean up after the Shader Module has completed processing
- Defines the PluginParam structure that is used to exchange parameters between the GIMP Shader Interface and the Shader Modules

Notes: The GPU GIMP utilities are compiled into a library and placed in the /usr/lib (default) directory. Thus, developers writing a Shader Module have to link their program with this library (GPUGimpUtils). This library is also critical to the functioning of the Shader Interface. When GIMP invokes the Shader Interface, it should be able to locate this library using the standard library paths.

2.2.3 Shader Modules

A shader module is a program which processes textures(images) on the graphics card. Each shader module contains set of shader programs (fragment shaders). These fragments are GPU instructions written in a language like GLSL (GL Shading Language). Fragment shaders are programs that operate like kernels on each pixel of a texture. This means that the instructions in the shader are executed for each pixel in the texture. A graphics card executes such fragment shaders in parallel on multiple pixels. This is where we obtain the speedup in comparision with CPU based image processing.

Our framework does not overly restrict the structure of a Shader Module. It only requires that each Shader Module implement a set of well defined interfaces. These interfaces are described below:

- A function to exchange shader parameter information with the GIMP Shader Interface
- A function to initiate processing of the texture passed to the Shader Module

In addition to these functions, the shader designer will have to write code to perform these general tasks:

- Management of read and write buffers in graphics memory
- Attaching textures to frame buffer objects
- Loading (or defining inline), the GLSL program to be executed
- Compiling and linking the GLSL program
- Passing parameters to the GLSL program
- Rendering a quad using the texture and the GLSL program

2.3 Dependencies

We have developed the framework using:

- C programming for interface and communicating with GIMP core
- OpenGL1.2 ARB extensions for image processing on the GPU
- GLSL programming for implementing pixel shaders

Our application has the following code dependencies:

- GIMP Headers: The developer libraries & headers for GIMP
- GL: The OpenGL Libraries
- GLUT: OpenGL Utility Toolkit
- GLEW: OpenGL Extension Wrangler Library
- libdl: Dynamic Linking Library

Hardware dependency:
We assume that the user has a Graphics Card with OpenGL support and Pixel Shader 2.0 Support.

3 Results

3.1 Testing

- To check the correctness of the output, we compare the outputs of the existing GIMP filters and the output of our filter
- We also compare the performance of our filters with the the original filters
- CPU: Pentium D 2.8 GHz (Family 9 Model 4 Step 7)
- GPU: NVidia 8800 GS (Mem: 128 bit DDR3@1048MHz, GPU@699MHz (G92 A2), Pixel Shader@1728 MHz)

3.2 Correctness

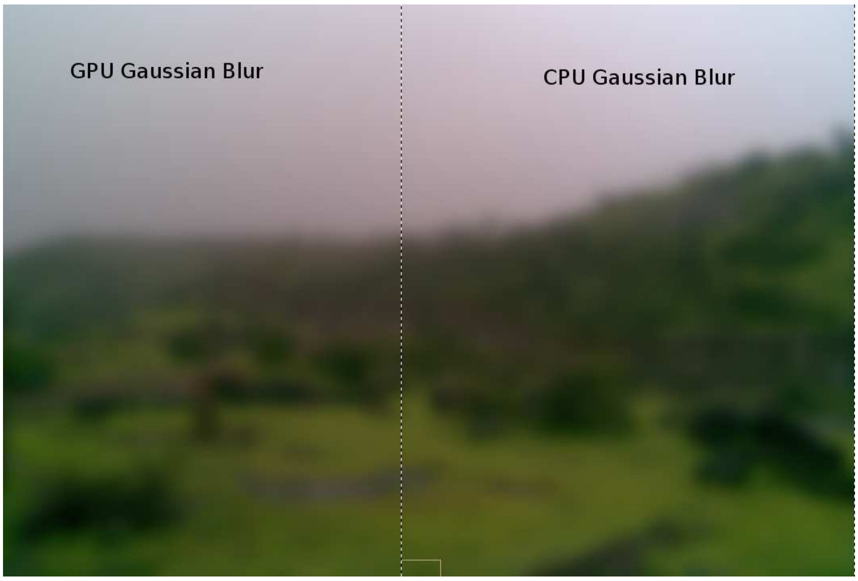

Figure 2: Comparision of GPU and CPU blur output

3.3 Performance

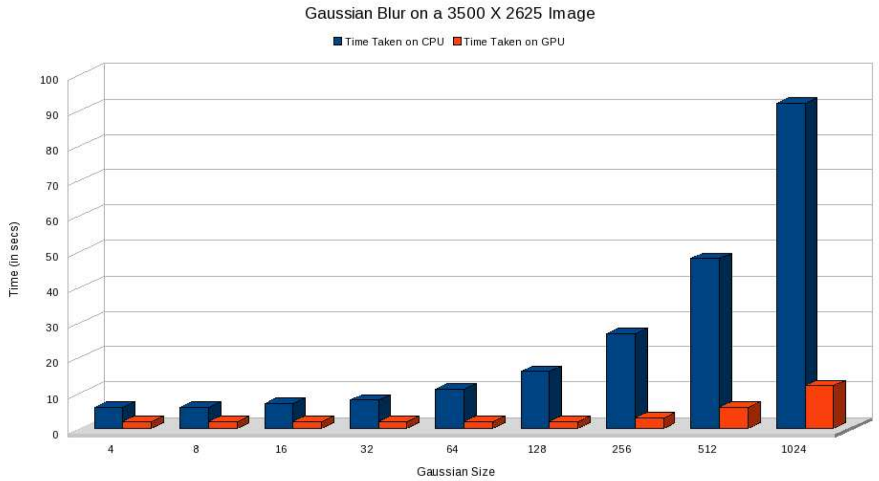

Figure 3: Time comparision for various blur filter sizes

3.4 Code Profiling

We have performed profiling on our code to identify regions of bottleneck

2

Process:	Emboss	Brighten:0	Blur:3	Blur:500	Blur:1000
GIMP Initialisation	252	235	253	251	237
Data Conversion1	614	613	617	614	615
GL Initialisation	313	1057	299	328	312
Plugin Run	176	185	195	206	206
Transfer from Texture	701	550	593	4739	9052
Data Conversion2	238	238	236	238	233
GIMP Refresh	690	740	684	689	662

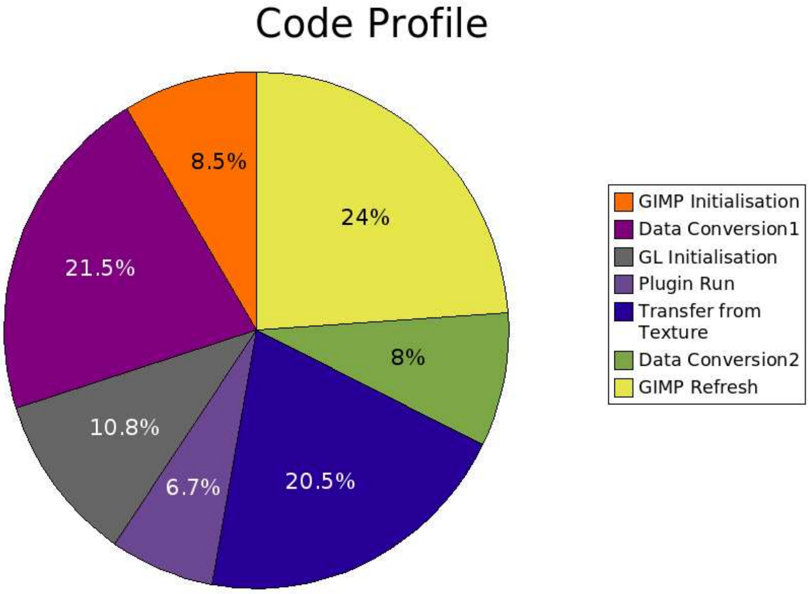

Figure 4: Code Profile for Blur-radius 3

4 Analysis and Future Work

- To the best of our knowledge, this is the only framework that implements GIMP filters on GPU
- We have developed a flexible framework for executing shaders on GIMP
- The Shader Module designer need not be concerned with GIMP internals, or OpenGL initialisation
- The framework can be easily extended to allow multiple shader modules to be executed sequentially
- The process of writing shaders for Graphics hardware is non-trivial, as GPU has several code structure limitations. Thus a lot more work can be done by professional shader designers, using our framework
- We plan to upload this project on Source Forge after some code clean up