Introduction to Programming Photoshop Filters

Athanasios Thanos Mantas
Darren Lehoux

January 2008

Abstract: This paper will take a look at developing and programming a filter plug-in for Adobe Photoshop Creative Suite 2 using Microsoft Windows. Given that not much has been set out for amateurs to effectively develop their own Photoshop filters, this paper will handle introductory concepts. Focus will be on getting the basics to work, and understanding how the files in the Photoshop CS2 Software Development Kit interact with each other, so beginners can see their code working as soon as possible.

1 Introduction

When it comes to photo editing software, Adobe Photoshop is the first program that comes to mind. As such, it should be no surprise that when we decided to program a filter plug-in for an industry standard, we chose to develop for Adobe's most successful version of Photoshop: Adobe Photoshop Creative Suite 2.

The main purpose of this section is to inform the reader of how to acquire a Photoshop CS2 SDK, and how to get it working. File interaction will be an important topic covered, as will filter plug-in programming. While there are many ways to go about creating image filters, the focus will be on how it is done in Photoshop CS2. Keep in mind that examples used will be done with respect to the Dissolve filter that is provided in the SDK.

1.1 Where to Find the SDK

In order to start working, an official Adobe Photoshop SDK is required. If used for academic or non-profit use, the SDK will be free. The official Adobe Photoshop SDK can be found at http://www.adobe.com/devnet/photoshop/. From here, simply register for an account, and explain why you wish to access the SDK. After all your information has been confirmed, you'll have the full SDK complete with sample code, include files, and documentation[1]. This SDK, along with Microsoft Visual Studio .NET and Adobe Photoshop CS2, is all that is needed in order to start developing filter plug-ins.

The big hurtle that beginner plug-in programmers face is the fact that the Photoshop SDK is made by developers for developers. That is, the information given in the SDK is not very user friendly. It takes many hours of reverse-engineering, debugging, and trial-and-error in order for someone unfamiliar with the environment to understand the features of the Adobe's SDK.

1.2 Preliminaries

Most SDKs on the market are very large and daunting; unfortunately the Photoshop SDK is no exception. However, after getting into it, there are a lot of things that can safely be abstracted from the programmer's point of view.

The first order of business is to figure out exactly how to get a "plug-in file" and where to place it. After everything is said and done, the output will be a MyFilter.8BF file. When Photoshop starts up, it loads all files that are in the Plug-Ins folder, including its subfolders. It is because of that start-up procedure that the .8BF file should be placed in the "C:\Program Files\Adobe\Adobe Photoshop CS2\Plug-Ins\Filters" folder, or wherever the "Plug-Ins\Filters" folder is located.

In order to make a filter, it is best to take one of the example filters that Adobe has provided, and modify it as necessary. This is done because going through all the mandatory start-up required to create a plug-in that is purely "your own" is much too time consuming for a project that will not be sold to the public. By using sample code, and abstracting advanced elements away from the programmer, the problem at hand is simplified greatly.

1.3 Getting Started

As unintuitive as it sounds, you simply can't just open up a solution for the filter you wish to edit and start changing code. Instead, what you have to do is go to the folder: C:\Adobe Photoshop CS2 SDK\samplecode\masterprojects, and open the BuildAll.sln file that is located there. This is where you are given access to almost all the code for the filter plug-in.

In the Solution Explorer, there is a list of every example plug-in inside the SDK. Upon cascading the solution tree, is the main code that accompanies each filter. From here the programmer can edit everything about the filter's functionality. In the Resource View, there is a similar cascading resource tree for each filter. By fully expanding any of the filter's resource tree and opening the dialog file (i.e. DISSOLVEDIALOG); you can see what the filter interface will look like. Although the environment has a very Visual Basic feel to it, modifying the filter interface from here will only result in errors during the compilation process. Instead, you can modify the filter's interface if you go to the folder "C: \Adobe Photoshop CS2 SDK\samplecode\filter\dissolve\win" and open up dissolve.rc in a basic text editor. Even though changing the filter's interface via text manipulation is cumbersome, it does allow for greater variation. Despite the fact that this method seems cryptic, it is not nearly as complicated as other interface programming tools, such as Java's SWING interface.

In order to make the .8BF file discussed earlier, simply right-click on the plug-in of your choice, and click "build". This will compile all the files in the plug-in, and create a .8BF file in the "C: \Adobe Photoshop CS2 SDK\samplecode\Output\win\debug" folder, assuming there are no errors in the code[2]. After placing the .8BF in the appropriate folder, as discussed above, a restart of Photoshop is all that is needed to see the newly created filter hard at work.

2 The Photoshop SDK Environment

Adobe created its own file system in order to make plug-in development as simple as possible for commercial developers. While veterans find the file system easy to use, beginners will find the file system quite confusing. The key is to only concern oneself with the key information in the SDK. The two main parts of a filter are: the Plug-In Property List (PiPL) with its accompanying .RC file, and the C source code that is used to describe the filter's functionality.

2.1 Plug-In Property List (PiPL)

The essence of a PiPL is that Adobe needed an extendable and flexible data structure that would store all of a plug-in's metadata, that is to say, the PiPL keeps the data about the data the plug-in will use[1]. For use in the Windows environment PiPLs are stored in Windows Resource Files, files with an extension of .rc, where Macintosh Resource Files have the .r extension.

Since Photoshop's clientele are mostly Macintosh users, the PiPL structure as been tailored to their needs. As such, it is recommended that .rc files be developed in the Macintosh "Rez" language. However, the problem with this is the output is a Macintosh Resource File. In order to convert the Macintosh .r file into a Windows .rc file, a program called "CNVTPIPL.EXE" should be used. This program is given in the Windows version of the Photoshop SDK to allow developers to create plug-ins for both Windows and OSX platforms.

In addition to a Property List the .rc file will contain information about the window that pops up when the filter is selected by a user, assuming that there is a need for a window, of course. From here, everything about the filter interface can be changed. As is the norm with other interface programming, each line corresponds to a widget on the interface. Each widget has attributes that are used to describe how it looks. For example, look at the following line of code:

```
RTEXT          "Dissolve Colour:",6,80,24,55,8,NOT WS_GROUP
```

This code means there will be a regular text area with the phrase "Dissolve Colour:" in it. The unique identifier associated with it is 6. The text area is located at an (x, y) location of (80, 24) relative to the upper-left corner of the pop-up window associated with the dissolve filter. The text area is going to be 55 pixels long and 8 pixels tall. One of the optional attributes associated with it is that it will not be associated with the Radio Button group, as it is simply a heading for the radio button choices.

As shown, it may not be the most intuitive approach, but it allows the programmer to tune every aspect of the interface to the finest of grains.

2.2 Filter Programming

One of the most important concepts Photoshop employs when it uses filter plug-ins is the idea of "rectangles". The idea is that there is a main rectangle that is the size of the image. The filter may be passed that rectangle to process, or it may be given a smaller

rectangle that is a sub-rectangle of the image rectangle[2]. This is well and good, but the true usefulness comes when the user selects a section of the image with the marquee tool, and then applies the filter. Photoshop will take a sub-rectangle that encompasses the selected section of the image, and when the filter is applied to the sub-rectangle, Photoshop masks any part that is not within the marquee area. This way, the algorithm for the filter does not have to change depending on what shape the filter will be applied to.

Back when Photoshop was first developed, short ints were used to determine the height and width of the image. However, as technology has evolved, higher resolution images are being used that exceed 0..32,767 range of the short int. Photoshop has a way to support images that have a height or width greater then 32.767 pixels, but unless the plug-in is going to be commercialized, any efforts to support Big Image Files will see little return on investment.

In order to effectively standardize filter plug-in code, all filter source code must satisfy certain requirements. As such, all filters have seven mandatory functions that have a fixed definition standardized by Adobe. However, a filter can have as many local functions as necessary. Each filter must have these functions because Photoshop allows scripting, and if all utilities are run the same way, it simplifies the scripting process greatly. The functions required for all filters are: PluginMain, DoAbout, DoParameters, DoPrepare, DoStart, DoContinue, and finally DoFinish[1].

PluginMain is where the filter starts all of its processing. The input variables include the image, what command to execute, a handle that is maintained between filter calls, and a place to store an error code, if necessary. It is in PluginMain that any useful data that the filter will use should be stored into global memory for future use. This is also the place where the other six essential functions are called, using a switch statement. DoAbout, DoParameters, DoPrepare, DoStart, DoContinue, and DoFinish are all functions that have to be included, and must take in no parameters. As their names state, it is fairly obvious what each one does.

DoAbout simply has information about what the filter does. In most cases, this function is empty. DoParameters ensures that the information the filter is trying to access is in fact accessible. Similarly, DoPrepare initializes all data structures such that the optimal amount of memory will be used by the filter. DoStart is the real meat of the filter. It is here that the filter will probably call most of its local functions in order to filter the image properly. DoContinue is usually not implemented in most filters. It is where the filter ends up if there would be in the middle of a multiple-stage filter. For example, if the user is able to select the first parameter for a filter, and then select the second parameter for the filter before the filter is finished, then the DoContinue function would be implemented. However, since there are not many filters like that, it is usually void of any meaningful code. Finally DoFinish wraps up the filter, freeing any memory that would cause memory leaks, writing any scripting parameters, and writing to Photoshop's registry

3 COSC 3P98 Filters

When developing our Photoshop filters, we chose to have a Psychedelic Theme to most of our filters. With the exception of the Burn Filter, every other filter has a similar theme to it.

3.1 Burn Filter

The Burn filter is a modification of the Invert filter that is provided in the SDK, and is the only filter that does not follow the Psychedelic theme. Like the Invert filter, Burn is a raw filter. That is to say it is applied once per activation, and there are no parameters or interface that accompanies the filter. Burn simply takes the colour at every pixel, and multiplies it by 0.88. This effectively diminishes the intensity of all the pixels to be filtered by 12%. The reason this filter has been kept, despite it not following the theme of the other filters is that it was also a filter that was coded in Assignment 1. As it turns out, the method of Burning an image in Photoshop is the same as Burning one using FreeImage.

3.2 Funky Invert Filter

Another Invert modification, Funky Invert was developed using stress testing methodologies. Each pixel of the rectangle to be filtered has its colour multiplied by 1.8. This multiplication causes many of the pixels to over-flow out of their colour boundaries. The filter was designed to see how Photoshop would handle such an error, be it a crash, negating the filter entirely, or simply ignoring the error. As it turns out, what Photoshop does reflects its C programming, and it just takes whatever value it has in that memory location, and treats it like a colour, as if it was still in the range. The result is this rather strange filter.

3.3 Our Dissolve Filter

The Dissolve example given in the SDK was the simplest version of a filter that had an interface the user could modify. As such, it quickly became the focus of interest, as modifiable filters can be very powerful. The Our Dissolve filter modifies the Dissolve interface slightly, and adds new psychedelic colours for the user to select. The goal was to have the Percentage of Dissolved Image variable a Sliding Bar, instead of a Text Entry Box, unfortunately there was little support for Slide Bars in the SDK, and online.

3.4 Psychedelic Filter

The Psychedelic filter was the main goal of the project. Taking what was known about how Photoshop works, and using the other filters as a guide, a truly Psychedelic style filter was born. Taking in three parameters from the user, a new colour for each pixel was calculated. As such, the interface had to be modified in order to provide the information that was needed. The main function that is used to calculate pixel colour is as follows: pixelcolor = ((pixelcolor^2) / param1) - param2 + param3;

The result was checked to ensure that it remained within the colour range of the objects. In addition to calculating what colour each pixel would receive, the code that updates the parameters, as well as the preview window had to be customized as well.

4 Conclusions

Since there are not many guides to Photoshop Filter programming are made available, and even less that are applicable to the Windows environment, hopefully this paper as cleared up some issues with how beginners are able to create their own useable filters. While this information is not useful for making a commercially available product, for both research and common interest, the SDK has been simplified to the point of usability.

Bibliography

1. Adobe Photoshop Documentation. Adobe, 20 December, 2007 <C:\Adobe Photoshop CS2 SDK\documentation>

2. Photoshop – Graphics – The Computer Science Agora. Prof. John C. Hart, University of Illinois. 20 December, 2007 <https://agora.cs.uiuc.edu/display/graphics/Photoshop>.