CSC 195.01 2013S *Technologies for Mediascripting*

Class 10: GIMP Internals (2): Manipulating Pixels

Held: Thursday, 11 April 2013

Summary: We consider plug-ins that manipulate images at the pixel level.

Related Pages:

- EBoard.

Notes:

- My hazy brain missed one thing about plugins, which we'll cover today.
- We won't do return values until next class. (Generally, menu items don't have return values.)
- I'll be leaving out details of some approaches today. You can find additional info in parts II and III of http://developer.gimp.org/plug-ins.html.
- Assignment: Another plugin, this time one that deals with pixels in some interesting way.

Overview:

- Updating displays.
- Returning status from your plugin.
- Pixels.
- A Sample Plugin.

Topics Missing from the Previous Class

- I missed a few important (and not so important) topics in the previous class. So, let's catch up on those.
- You can often use `grep` to look for details in headers.
  - For example, the definition of the GimpParam struct is in `/usr/include/gimp-2.0/libgimp/gimp.h`.

Updating Displays

- How do you see what your operation has done? In MediaScheme, we use `context-update-displays!`. But the underlying PDB function is `gimp-displays-flush`.
- So, in C, we just write

```
gimp_displays_flush ();
```

Reporting on Success/Failure

- You may have noted that `run` is a `void` function.
- So, how do we indicate whether or not `run` succeeded?
- We do it with the `return_vals` parameter.
- That is, we need to make an array of return values, the first of which is an indication of the success or failure of our function.
- So, we do something like the following:

```
static GimpParam results[1];
results[0].type = GIMP_PDB_STATUS;
results[0].data.d_status = GIMP_PDB_SUCCESS;

*nreturn_vals = 1;
*return_vals = results;
```

- Note that the results have to be in static memory so that they are not lost when your procedure returns.
- Your other options for status are
  - GIMP_PDB_EXECUTION_ERROR
  - GIMP_PDB_CALLING_ERROR
  - GIMP_PDB_PASS_THROUGH
  - GIMP_PDB_CANCEL

Pixel-Based Operations

- There are times we want to do interesting things to images a pixel at a time.
  - `image-compute` involves computing a color for each pixel
  - Simple blurring involves averaging a pixel with the neighboring pixels
  - Most filters involve pixel-level operations.
- Note: You will need to deal with the underlying representation of pixels. In many cases, pixels are arrays of 8 bit values. Getting them right can take some work, though.
- The GIMP provides a host of ways to access pixels, some easy, some more complex.
  - Unfortunately, there seems to be an inverse relationship between complexity and speed.
  - Hence, while I'll give you an overview of approaches, I'll focus on the details of the most complex approach.
- Approach 1: Use PDB functions `gimp-drawable-get-pixel` and `gimp-drawable-set-pixel`
- Approach 2: Use the `GimpPixelRgn` struct (which you can get from a drawable id) along with `gimp_pixel_rgn_get_pixel` and `gimp_pixel_rgn_set_pixel`
- Approach 3: Use the `GimpPixelRgn` struct (which you can get from a drawable id) along with `gimp_pixel_rgn_get_row` and `gimp_pixel_rgn_set_row` (ro col, or rect)
- Approach 4: Use `GimpPixelRgn` struct, but let the GIMP tell you which portions tohandle.
- Note: When you're finished dealing with pixels, you need to deal with all sorts of fun cleanup. This includes

```
gimp_drawable_flush (drawable_id);
gimp_drawable_merge_shadow (drawable_id, TRUE);
gimp_drawable_update (drawable_id, left, top, width, height);
```

- So, let's get started.
- ...
- The iteration through tiles is somewhat strange.
  - We use `gimp_pixel_rgns_register` to start the iteration. That makes regions point to the first tile in their respective drawables.
  - We use `gimp_pixel_rgns_process` to advance them
  - Both functions return NULL when they are done.
  - So, something like

```
gpointer pr;
pr = gimp_pixel_rgns_register (2, &source_region, &target_region);
while (pr != NULL)
  {
    // Do work
    ...
    // Move on to the next tile
    pr = gimp_pixel_rgns_process (pr);
  } // while
```

- How do we iterate through the pixels in the region? We could use `gimp_pixel_rgn_get_pixel` and `gimp_pixel_rgn_set_pixel`. But that turns out to be slow.
- Instead, we need to step through the data structure.
