ECE2035 **Spring Semester, 2015**

Project 4 – Image Manipulation

Assigned: Apr 6, 2015 *Due: Apr 17, 2015, 11:59pm*

Introduction. For this assignment we will use a custom graphics *Application Programming Interface* to manipulate an existing image, and to draw a simple animation of a moving wheel with spokes. There are three parts:

1. Invert an image.

   Your program is to load in an existing image of the Tech Tower called `TechTowerSign.jpg`. After reading in the image, create another window of the same size (width, height, and depth) of the original and transform the original image to be upside-down from the original image. The original image and the inverted image are shown below:

2. Convert an image from 32-bit RGB color values to 8-bit gray scale black and white images. The original image is the `TechTowerSign.jpg` (same as part 1). To do the conversion, simply compute the average of the three color components (red, green, blue) and store that in the 8-bit pixel array.

3. Draw an animated wagon wheel.

   Create a blank image of 512 by 512, 32 bit colors. The image should do the following:

   (a) Draw a circle centered at width() / 2 and height() / 2, with a radius of 200 pixels. The color for the circle should be black.

   (b) Draw spokes from the center point of the circle to the circle border every 30 degrees (12 spokes total). Each spoke should be either solid red, solid green, or solid blue depeding on the angle and the algorithm given below.

   (c) Animate the rotation of the wheel at 24 frames per second. For each update rotate each spoke by 12 degrees.

   (d) The color of the spoke should cycle between red, green, and blue. Pick an arbitrary starting point (one of the spokes) and use red, for the next spoke use gree, and the next use blue. Repeat the cycling of colors until the last spoke is drawn.

The Graphical API To visualize your algorithm, we have provided a simplified interface to the *Qt* graphics library. This library is common across Linux and Windows platforms, so the same interface should work on most platforms. It has been compiled and tested on deepthought, so we are confident it works properly.

The API is defined in `gtDisplay.h`, and implemted in `gtDisplay.cc` and `gtWidget.cc`. You should not change any of these. Skeleton code is given in gtMain.cc, and you should implement your code in `gtMain.cc`

The define API for gtDisplay is as follows. See comments in the `gtDisplay.h` file fo rmore details.

(a) `gtDisplayinit`. All *gtDisplay* applications require an initialization by calling `gtDisplayinit`.

(b) `gtCreateBlankWindow` A new blank window can be created by calling `gtCreateBlankWindow`. The width, height, and depth of the new window are specified by function arguments.

(c) `gtLoadImage`. A new window displaying an image displayed can be created by calling `gtLoadImage`, specifying the file name of an image file. The image file must be in the same subdirectory as you `gtMain` program.

(d) `gtSaveImage`. The contents of a window can be saved to a file by calling `gtSaveImage`. The first argument is the window pointer and the sedind argument is the file name for the image rile.

(e) `gtGet8BitPixels`. Once an image is loaded or a blank window is created, the `gtGet8BitPixels` function returns a pointer to the pixel array. This array is an array of `uint8_t`, 8–bit gray scale. Zero is solid black, 255 is solid white, with linear interpolation in between for various shades of gray. This of course assumes the window was created with 8-bit pixel depth.

(f) `gtGet8BitPixels`. Once an image is loaded or a blank window is created the `gtGet32BitPixels` function returns a pointer to the pixel array. This array is an array of `uint32_t`, 24–bit colors. This of course assumes the window was created with 32-bit pixel depth. There are several functions described below for managing the red, green, and blue components of a color pixel.

(g) `gtUpdate`. A call to `gtUpdate` will copy all pixels from the in-memory pixel array (using the `gtGet` functions above) to the graphics hardware that the user sees. The entire pixel array is copied.

(h) `gtUpdate(gtWindow, x, y, w, h)`. This function also copies pixels from the in–memory pixel array to the graphics card, but it differes from the above in that only a subset of pixels are copied. The x, y, w, and h paramters specify the upper left corner (x and y) and the width/height (w, h) of the rectangle to copy.

(i) `gtGetWidth()` returns the width of the specified window.

(j) `gtGetHeight()` returns the height of the specified window.

(k) `gtGetDepth()` returns the depth of the specified window.

(l) `gtUpdateRate` specifies the maximum update rate in frames per second. This is use to "slow down" animations so the user can actually see things moving. If not specified, the updates are "as fast as possible" and are often too fast to be seen by the user.

(m) `gtRun()` should be called after all drawing is complete. It will not exit until all windows are closed.

(n) `gtMakeRGB()` creates a `uint32_t` color value from the three components red, green, and blue¿.

(o) `gtGetRed()` returns the red component of a 32–bit color value.

(p) `gtGetGreen()` returns the green component of a 32–bit color value.

(q) `gtGetBlue()` returns the blue component of a 32–bit color value.

Copying the Project Skeletons

1. Log into `deepthought19.cc.gatech.edu` using `ssh` and your prism log-in name.
2. Copy the files from the ECE2035 user account using the following command:

   ```
   /usr/bin/rsync -avu /nethome/ECE2035/ImageManipulation .
   ```

   Be sure to notice the period at the end of the above command.
3. Change your working directory to `ImageManipulation`

   ```
   cd ImageManipulation
   ```
4. Copy the provided `gtMain-skeleton.cc` to `gtMain.cc` as follows:

   ```
   cp gtMain-skeleton.cc gtMain.cc
   ```
5. Edit the `gtMain.cc` program to include your image manipulation solution.

Turning in your Project. The system administrator for the *deepthought* cluster has created a script that you are to use to turn in your project. The script is called `riley-turnin` and is found in `/usr/local/bin`, which should be in the search path for everyone. From your **home directory** (not the ImageManipulation subdirectory), enter:

`riley-turnin ImageManipulation`.

This automatically copies everything in your `ImageManipulation` directory to a place that the TA's can access (and grade) it.