Bridging Graphics

Lori Scarlatos, September 2006

Introduction

Computer graphics have one clear advantage over other types of programs: with computer graphics, you can literally see whether your program is working or not. This is great for beginning programmers, who need that immediate feedback. Yet most graphics libraries are so complex that it takes a semester-long graphics course to learn how to use them. Bridging Graphics is a library of functions that enables beginning C++ programmers to quickly create programs that draw 2D graphics.

Using Bridging Graphics

A Bridging Graphics template project (BG.zip) can be downloaded from the web site (http://acc4.its.brooklyn.cuny.edu/~lscarlat/BG/BG.zip). When you open the project, you will notice that there are four files in it: BGtemplate.dev, bg.h, bg.cpp, and main.cpp. This project was created for the Bloodshed development environment on Windows machines. The following instructions assume that you are also using Bloodshed on a Windows machine.[1]

Here's how to modify this template project to create your own Bridging Graphics project:

1. Click on the link and choose to save BG.zip to your disk.
2. Double-click on BG.zip to open it.[2] Choose to extract all files. This will create a folder named BG. You can rename (and relocate) this folder as you see fit.
3. Start up Bloodshed.[3] Use File -> Open to open up the BGtemplate.dev project.[4]
4. Add instructions to the main.cpp file to draw your graphics. You should also rename main.cpp to something more appropriate. *Do NOT make any changes to bg.h or bg.cpp unless you know what you are doing!*
5. Compile and run your program to see what you drew. You can close the graphics window (and end the program) by clicking on the "X" in the corner of the window, or pressing the ESC key.

---

[1] You can use the Bridging Graphics functions in another development environment by simply importing bg.h and bg.cpp into your project. Your project must also link to the OpenGL library on your system. Please note that functions in bg.cpp currently rely on windows functions, and therefore cannot be ported to other operating systems without making major changes to it.

[2] This assumes that your system has a program for un-zipping files, such as WinZip.

[3] See the Bloodshed startup manual for more information on how to install and use the Bloodshed IDE.

[4] Alternatively, you can start a new project. Choose File -> New -> Project from the menu bar. A popup window will appear; select Multimedia, and then OpenGL. You will need to remove the default file from that project (use Project -> Remove) and then add in bg.cpp, bg.h, and main.cpp from the BG folder (using Project -> Insert).

Graphics Overview

Imagine you are in elementary school, and your teacher asks you to paint a picture at an easel. First you must ask for an easel to work at. Then you can ask for a piece of paper to paint on. The teacher will give you any color paper you want, but that paper is a fixed size and you can only have one sheet at a time. Then you can start painting your picture. Initially you have a pot of white paint to work with. You also have a wide range of other colors to choose from, but you must ask for a color before you can use it. Again, you can only have one pot of paint at a time. The paint is opaque: everything that you draw covers up whatever was in that spot before. When you are done with your painting, you must show it to your teacher.

If you were to write down all the steps that you took to paint your picture, what would that list of steps look like? It would look like a BG program!

Steps in your BG Program

Bridging Graphics (BG) defines functions for drawing the graphics, and constants representing graphics options. You may notice that all of the functions start with *bg* and all of the constants start with *BG*. This is a *naming convention*. To use this these functions in your project, bg.cpp must be part of your project. You also need to include bg.h in your main program file with the following line:

#include "bg.h"

BG was designed to support the creation of *procedural* programs: the first command is done first, the second command is done second, and so on. Your BG program will tell the computer exactly what to do, step by step. Here is the sequence of commands that your BG program must use:

1. Create a graphics window to draw in, by calling `bgCreateWindow`. This is like getting an easel to work at.
2. Select a background color (by calling either `bgSetBackgroundColor` or `bgSetBackgroundRGB`) and then clear the screen (by calling `bgClearScreen`). This is like getting a clean sheet of colored paper to work on. If you skip this step, it will be like painting directly on the easel ... and you don't know what might be there!
3. Draw your picture. This is done with a sequence of color selection and drawing functions (see below).
4. Update the display (by calling `bgUpdateDisplay`) so you can see what you drew. This is like showing your finished picture to your teacher or parent. If you forget this step, the window will remain blank.
5. Pause (or freeze) the screen (by calling `bgPause`) so you have time to look at the picture. If you skip this step, the program will end and your picture will disappear immediately.
6. Destroy the graphics window by calling `bgDestroyWindow`. There's no good analogy for this; it's just good programming practice.

The Drawing Surface

When you tell the computer how to draw something, you will do this by specifying locations (on the drawing surface) using *2D Cartesian coordinates*. You may recall that the Cartesian coordinate system has two *axes*: *X* (with values increasing to the right, decreasing to the left) and *Y* (with values increasing up and decreasing down). The coordinate system also has an *origin*, where the two axes cross one another. Any point in the coordinate system can be specified by a pair of values representing measurement along the X axis followed by measurement along the Y axis. The origin is the point with coordinates (0, 0).

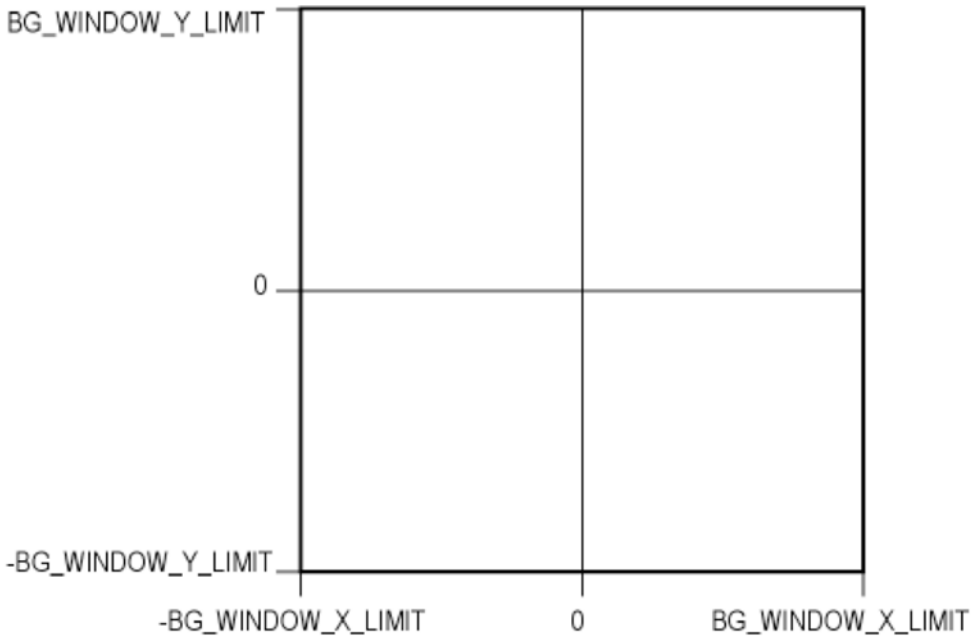


Figure 1. Coordinates of the BG window

In math class, you may have also learned that the coordinate system is infinite. In computer graphics, however, we can only see part of the coordinate system. The part of the coordinate system that we see is viewed through a *window* ... just like looking through a window in your home. As shown in Figure 1, the origin (seen in a BG window) is always in the center. Any points that are left of the left edge (-BG_WINDOW_X_LIMIT) or right of the right edge (BG_WINDOW_X_LIMIT) are not visible in the window. Likewise, any points that are below the bottom edge (-BG_WINDOW_Y_LIMIT) or above the top edge (BG_WINDOW_Y_LIMIT) are not visible in the window.[5]

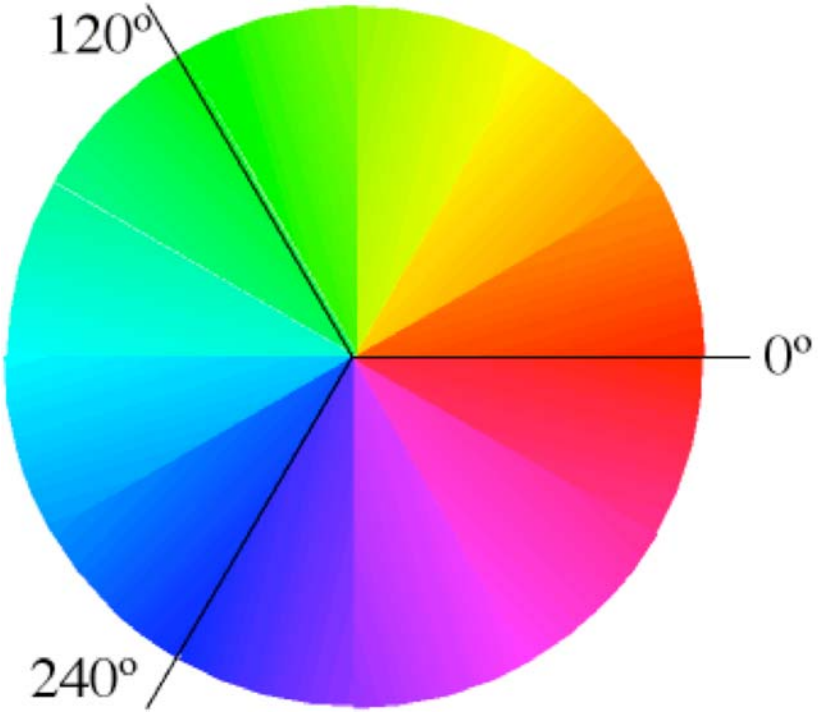
Colors

BG allows you to specify the color of the background (paper) and the color that you are working with (paint). Once you select a color, that color is used for all of your drawing until you change the color. If you do not specify any colors, the background will be black and the drawing color will be white.

---

[5] If you look inside bg.h, you will see that BG_WINDOW_X_LIMIT = BG_WINDOW_Y_LIMIT = 16.

When you go to the store to buy a can of paint, you have to ask for the color by its name. Each brand of paint has different colors with different names. You must use the precise name of the color for the brand you want, or else the salesman won't know what to give you. Just like a brand of paint, BG has its own set of named colors that you can refer to. These colors are BG_BLACK, BG_WHITE, BG_GREY, BG_RED, BG_GREEN, BG_BLUE, BG_YELLOW, BG_CYAN, BG_MAGENTA, BG_ORANGE, BG_PINK, BG_BROWN, and BG_PURPLE.



BG also allows you to select a color based on its *hue*. When we say that something is green or reddish-orange, we are referring to the hue of the color.[6] We can imagine hue as positions on a *color wheel*, as shown in Figure 2. Then, each color on the wheel can be referenced by an angle. Red is 0º (or 360º); green is 120º; and blue is 240º. BG allows you to specify 360 different hues by their angle.

Figure 2. Hues on the Color Wheel

Sometimes, however, we don't want a color picture; we want to paint in shades of grey instead. BG allows you to paint with a seemingly infinite range of values by using a *grey scale*. In the grey scale, 0 is black, 1 is white, and every grey value is a fraction in-between.

Finally, BG allows you to specify the full range of visible colors using RGB (red-green-blue) values. RGB colors are frequently used in computer graphics because they correspond directly to how colors are made on the screen. However, they are non-intuitive to use. If you want to use RGB colors in your program, look at the Internet or the color palette in a program like Photoshop to get the RGB values you want.

---

[6] When we refer to a color as "bright green" or "dull orange", we are referring to the *saturation* of the color. When we say that the color is "light pink" or "dark purple", we are referring to the *value* of the color. BG does not support variations in saturation and value.

Bridging Graphics Functions

All of the functions listed here are declared in bg.h and defined in bg.cpp. You must have both of these files in your project if you want to use them. You must also tell the compiler to include bg.h with the following:

#include "bg.h"

Window Functions

Create a New Window

The first thing you should do in your program is to create a window to draw in. Create a window with the following statement:

bgCreateWindow();

Hold the Window on Screen

After you have drawn something in the window, you will probably want to keep it on the screen for awhile. If you don't, your program will end and your picture will disappear almost immediately. Hold the window on the screen with the following statement:

bgPause();

Close the Window

The last thing you should do in your program (before the return statement) is to destroy the window you created. Destroy that window with the following statement:

bgDestroyWindow();

Color Functions

Set the Background Color

There are two different ways that you can choose a background color. If you want to use a named color, use the following statement:

bgSetBackgroundColor (color_name);

where *color_name* is one of the BG named colors: BG_BLACK, BG_WHITE, BG_GREY, BG_RED, BG_GREEN, BG_BLUE, BG_YELLOW, BG_CYAN, BG_MAGENTA, BG_ORANGE, BG_PINK, BG_BROWN, or BG_PURPLE.

The other way to specify a background color is by calling:

bgSetBackgroundRGB (red_value, green_value, blue_value);

where *red_value*, *green_value*, and *blue_value* are all floating point numbers from 0 to 1, which represents the intensity of each component.

Set the Paint Color

There are four different ways that you can choose a paint (foreground) color. If you want to use a named color, use the following statement:

bgSetColor (color_name);

where *color_name* is one of the BG named colors: BG_BLACK, BG_WHITE, BG_GREY, BG_RED, BG_GREEN, BG_BLUE, BG_YELLOW, BG_CYAN, BG_MAGENTA, BG_ORANGE, BG_PINK, BG_BROWN, or BG_PURPLE.

You may also specify a color by a hue value, which corresponds to an angle on the color wheel. Do this by calling:

bgSetHue (hue_value);

The hue_value should be an integer, ranging from 0 to 360.

If you want to paint in shades of grey, set the paint color by calling:

bgSetGrey (grey_value);

The grey_value is a floating point number, ranging from 0 to 1, where 0 is black and 1 is white.

Finally, you can access the full range of colors by calling:

bgSetRGB (red_value, green_value, blue_value);

where *red_value*, *green_value*, and *blue_value* are all floating point numbers from 0 to 1, which represents the intensity of each component.

Drawing Functions

Next, you will want to draw your picture. You can draw points, lines, rectangles, triangles, and circles. You must draw these by specifying floating-point coordinates in the window. Remember, if you specify coordinates outside the window, they will not be seen in the window. Whatever you choose to draw, it is drawn with the paint color that you chose most recently.

Clear the Screen

Before you begin to draw anything, you must first clear the screen. Otherwise, you will be painting on whatever was already there. This is accomplished with the following call:

bgClearScreen ();

Draw a Point

You can draw a point at position (x, y) with the following statement:

bgDrawPoint (x, y);

Draw a Line

A line is draw from the current pen position to a new pen position. Initially, the current pen position is at (0, 0). Calling **bgDrawPoint** changes the current pen position. You can also change the current pen position to (x, y) (without drawing anything) with a call to:

bgMoveTo (x, y);

Then, you can draw a point from the current position to (x, y) with a call to:

bgDrawTo (x, y);

Notice that this will also update the current position.

Draw a Rectangle

Rectangles (or boxes) are drawn with their edges parallel to the X and Y axes of the coordinate system. A rectangle is specified by its lower-left corner (x1, y1) and its upper-right corner (x2, y2). A rectangle is drawn with the following statement:

bgDrawBox (x1, y1, x2, y2);

Draw a Triangle

Triangles are defined by 3 points: (x1, y1), (x2, y2), and (x3, y3). Notice that any polygonal shape can be created with one or more triangles. A triangle is drawn with the following function call:

bgDrawTriangle (x1, y1, x2, y2, x3, y3);

Draw a Circle

A circles is specified by its center point (x1, y1) and its radius (all floating point values). A circle is drawn with the following function call:

bgDrawCircle (x1, y1, radius);

Updating the Screen

All of the drawing functions are applied to a hidden (background) part of memory known as the *screen buffer*. To see what you have drawn, you will have to move this screen buffer to the foreground. This is accomplished with the following statement:

bgUpdateDisplay ();

Image Functions

Images are 2D arrays of pixel values. Every image has a height (number of rows) and a width (number of columns). Pixels may be accessed individually by specifying their row and column.

Most images that we see are stored in some particular *format*. Some examples are GIF, JPEG, PGM, PICT, TIFF, and BMP. The format specifies what information about the image goes where. Many image files are also compressed. Reading and writing an image file in a specific format is typically a non-trivial task. For this reason, several simple (portable) formats have been specified, including PGM and PPM.[7]

---

[7] If you want to create a PGM file from a GIF or JPEG image, or create a GIF or JPEG version of a PPM image, there are several utilities on the UNIX machines that you can use.

To convert a JPEG file (named image.jpg) to a PGM file, type
```
jpegtopnm image.jpg > image.pgm
```
To convert a GIF file (named image.gif) to a PPM file, type
```
giftopnm image.gif > image.pgm
```

To convert a PPM file (named image.ppm) to a JPEG file, type
```
pnmtojpeg image.ppm > image.jpg
```
To convert a PPM file (named image.ppm) to a GIF file, type
```
pnmtogif image.ppm > image.gif
```

BG includes a few simple functions for reading PGM (grey) files and writing PPM (color) files.

Read in an Image

Sometimes it is useful to start with a pre-existing image. The following function reads in an image from a file called filename. The variables width (an integer), height (an integer), and image (a pointer to unsigned char) will be filled in by the function, so you need to pass pointers to the function. The function itself returns TRUE (1) if the file was read successfully, FALSE (0) otherwise. Assuming that an integer variable called result has been defined, the call looks like this:

result = bgReadImage (filename, &width, &height, &image);

Writing an Image

You can write out a copy of your picture with a call to:

bgWriteImage (filename);

Drawing an Image

You can draw an array on your graphics screen 2 different ways. In both cases, you must specify the number of rows and columns in the image. A grey image, with pixel values (representing grey values) stored as unsigned char (ranging from 0..255) can be drawn with a call to:

bgDrawGreyImage (rows, columns, image);

A hue image, with pixel values (representing hue values) stored as integers (ranging from 0..360) can be drawn with a call to:

bgDrawHueImage (rows, columns, image);