,

Bridging Graphics

Lori Scarlatos, January 2007

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 computer graphics libraries are so complex that it takes a semester-long course to learn how to use them.

Bridging Graphics (BG) is a library of functions that enables beginning C++ programmers to quickly create programs that draw 2D graphics. Animation and response to the keyboard are also supported.

Using Bridging Graphics

BG can be used on any computer that has a C++ compiler and the OpenGL and GLUT[1] libraries on it. Bridging Graphics is distributed as a .zip file, and can be downloaded from your instructor's web site. There are 2 versions currently in distribution: one for windows, and one for unix. Both distributions contain the following files:

- **`bg.h`** - defines BG types and declares BG functions : DO NOT MODIFY
- **`bg.cpp`** - defines most BG functions declared in bg.h : DO NOT MODIFY
- **`bgEvents.cpp`** - defines BG callback functions : change the body of these functions to respond to key presses and change over time
- **`main.cpp`** - template for a BG main program : most modifications will be done here

To use BG, you must download the appropriate .zip file to your computer, and then extract all the files[2] before you begin working on them. Here's how to use the files on the two platforms.

BG for Unix

In the unix distribution, two additional files are included: **`Makefile`** and **`Make-config`**. These files are used to compile and link your BG program using g++ on the Sun computers in the WEB building[3]. To compile your program,

---

[1] OpenGL is distributed with most C/C++ development environments. Mesa libraries may also be used for BG. GLUT can be downloaded and installed for free; Open GLUT and freeglut may also be used for BG.

[2] You will need a program for un-zipping files, such as WinZip, on your computer. Ask your instructor if you are unsure of how to do this.

[3] These files link to the Mesa, GLUT, and Xwindows libraries on the ITS network. They work best on the new (grey, higher-numbered) Sun computers in the back of the lab.

,

1. Type **`make bg`** to compile bg.cpp and bgEvents.cpp.
2. Type **`make main`** to compile main program **`main.cpp`** and create an executable file named **`main`**.
3. Type **`main`** to run your program. Notice that you will need to move the mouse cursor over the graphics window to make the colors appear correctly.

BG for Windows

To develop your program using an IDE (such as Visual C++ or Bloodshed Dev C++) on a computer running Windows,

1. Start up your IDE. Use File -> Open to open your main program (**`main.cpp`**). Create a project for this program.
2. Add **`bg.cpp`** and **`bgEvents.cpp`** to your project.
3. Modify link settings so that OpenGL and GLUT libraries are linked in to your final executable file.

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`** and **`bgEvents.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. Clear the screen by calling **`bgClearScreen.`** If you like, you can select a background color by calling either **`bgSetBackgroundColor`** or **`bgSetBackgroundRGB`** ... otherwise, the background will be black. Clearing the screen is like getting a clean sheet of colored paper to work on; you cannot skip this step!
3. Draw your picture. This is done with a sequence of color selection and drawing functions, as described.
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. Call **`bgMainLoop`**. This opens the window, displays your picture, and loops forever so you have time to look at the picture. If you want your picture to change over time, you'll have to use event callbacks (described below).

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).

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.

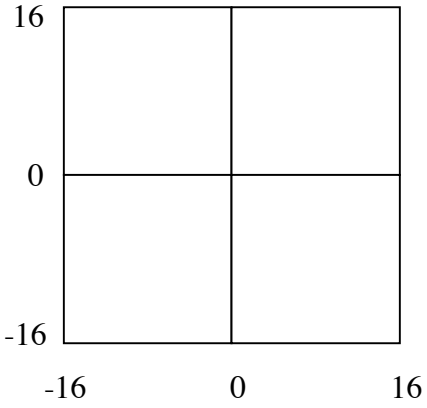


Figure 1. Coordinates of the BG_SQUARE window

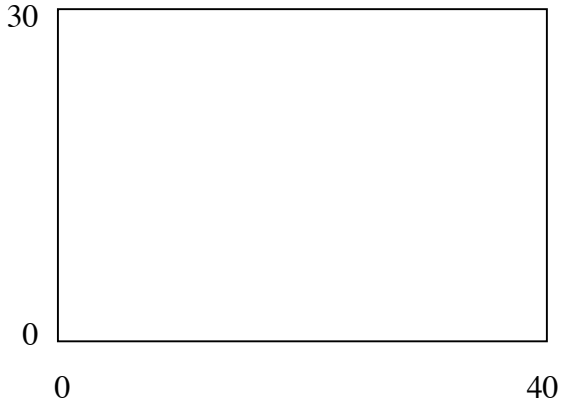


Figure 2. Coordinates of the BG_ RECTANGLE window

There are two possible windows you can create with BG. The default is a square window, shown in Figure 1. In this window, the origin of the coordinate system is in the center.

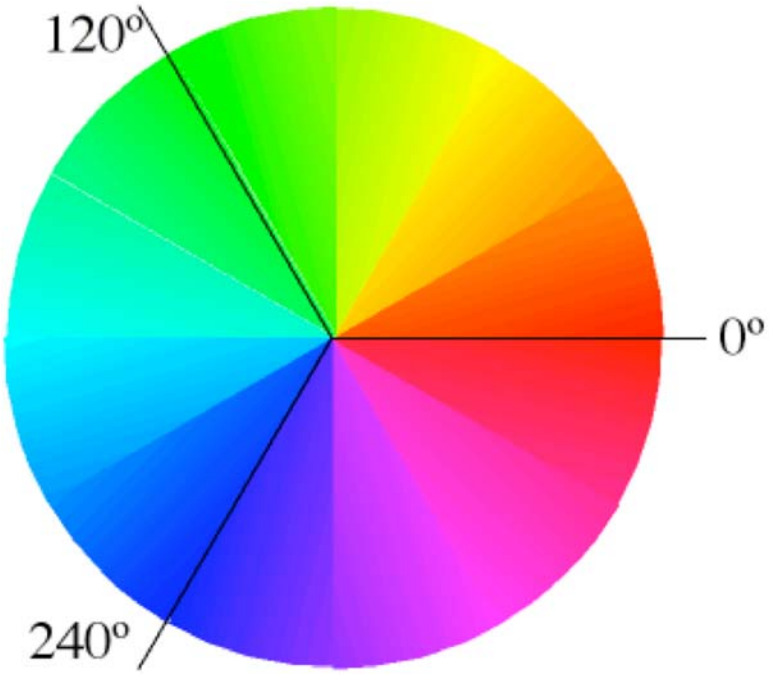
,

You can also create a rectangular window, as shown in Figure 2. In this window, the origin of the coordinate system is in the lower-left corner. In both cases, any points that are drawn to the left of the left edge or right of the right edge (or below the bottom edge or above the top edge) are not visible in the window. You can get these coordinate limits by calling the appropriate functions described below.

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.

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.[4] We can imagine hue as positions on a *color wheel*, as shown in Figure 3. 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 3. Hues on the Color Wheel

---

[4] 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.

,

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.

Images

An image is a two-dimensional array of color spots or *pixels*. You can think of it as being like a piece of graph paper where every square is filled in with exactly one color. Like the graph paper, an image has a certain number of *rows* and *columns*. Unlike the graph paper, an image also has a *depth*, which represents the amount of computer memory needed to store the color of each pixel.

With BG, an image is defined with a class called **`BGimage`**. This class has public methods to draw the image, as well as find the number of rows, columns, and depth. The array of pixels is also public, so you can change the values as you see fit.

There are two possible ways to create a BGimage object. One is to read an image in from a file. Image files can come in a variety of *formats*. 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, BG uses two simple (portable) formats: Portable Pixmap (`.ppm`) and Portable Greymap (`.pgm`).[5] When you read in an image from a file, you can decide whether the background should be transparent or not.[6]

You can also create a blank image that your program will fill in later. When you take this latter option, you need to specify the type of image to create. BG supports six different types of images:

---

[5] 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
```

You can also create PPM and PGM files using Photoshop.

[6] In BG, white is considered the background color; therefore all white pixels will become transparent.

,

- `BG_GREY_IMAGE`: Each pixel is represented by one byte, with values ranging from 0 (black) to 255 (white)
- `BG_HUE_IMAGE`: Each pixel is represented by one byte, with values representing half of the angle on the color circle, i.e. ranging from 0 to 180.
- `BG_COLOR_IMAGE`: Each pixel is represented by one byte, with each value representing one of BG's color values.
- `BG_RGB_IMAGE`: Each pixel is represented by three bytes, corresponding to proportions of Red, Green, and Blue. For each of these, values range from 0 (no color) to 255 (full color). The result is more than 16 million possible colors!
- `BG_RGBA_IMAGE`: Each pixel is represented by four bytes. In addition to the Red, Green, and Blue bytes, there is an Alpha (transparency) byte, with values ranging from 0 (transparent) to 255 (opaque).
- `BG_GREY_ALPHA_IMAGE`: Each pixel is represented by two bytes. In addition to the grey byte, there is an Alpha (transparency) byte, with values ranging from 0 (transparent) to 255 (opaque).

Event Callbacks

Sometimes you may want your picture to change, either over time (as in an animation) or in response to some event (like pressing a key on the keyboard). BG provides *event callbacks* to support these types of changes in your program. You must take the following steps to enable this capability:

1. Fill in the appropriate callback function found in the **`bgEvents.cpp`** file. This specifies what should be done when the event occurs. If you want the drawing to change, remember to clear the screen first (`bgClearScreen`) and update the display last (`bgUpdateDisplay`). Do not change the name of the function or the arguments, and do not remove the final call to `glutPostRedisplay`.
2. Enable the callback by calling `bgEnableCallback`. If you later want to disable the callback, call `bgDisableCallback`.

The following types of callbacks are supported:

- `BG_KEY`: The function `bgKeyFunction` is called when one of the printable characters (or ESC, DEL, or BACKSPACE) is pressed on the keyboard. The key pressed is the first argument returned.
- `BG_SPECIAL_KEY`: The function `bgSpecialKeyFunction` is called when one of the special keys is pressed on the keyboard. These include the arrow and function keys on the keyboard. The key pressed is the first argument returned.
- `BG_IDLE`: The function `bgIdleFunction` is called repeatedly while the system is idle (i.e. no keys are being pressed). Therefore, whatever you put in this function should represent *one frame* (or step) in the animation. You can slow down the animation with a call to `bgPause`; you can stop the animation with a call to `bgDisableCallback`.

,

Bridging Graphics Functions

All of the functions listed here are declared in `bg.h` and defined in `bg.cpp` and `bgEvents.cpp`. You must have all 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 square window with the following statement:

```
bgCreateWindow();
```

Alternatively, you can create a rectangular window with the following statement:

```
bgCreateWindow(BG_RECTANGLE);
```

Get Window Dimensions

Window dimensions refer to the Cartesian coordinates that define the window; they have nothing to do with the actual size of the window (in pixels) on the screen. Each of the following functions returns a floating point number.

```
bgGetWindowWidth();
bgGetWindowHeight();
bgGetWindowLeft();
bgGetWindowRight();
bgGetWindowBottom();
bgGetWindowTop();
```

Clear the Screen

Before you begin to draw anything, you must first clear the screen. If you haven't set the background color, the cleared screen will be black. This is accomplished with the following call:

```
bgClearScreen ();
```

Updating the Screen

All of the drawing functions are applied to a hidden (background) part of memory. To see what you have drawn, you must call the following function[7]:

```
bgUpdateDisplay ();
```

---

[7] Notice that ALL drawing functions must be sandwiched between a call to `bgClearScreen` and a call to `bgUpdateDisplay`. Otherwise, the drawing will not appear on the screen.

,

Hold the Window on Screen

The last call in your main program (before the return) must be:

```
bgMainLoop();
```

Without this, your graphics will never appear on the screen. This is an infinite loop, so any changes made to the screen must be done through the callbacks.

If you need to pause briefly in your animation, you can use the following statement. The floating point argument represents the number of seconds to pause.

```
bgPause(seconds);
```

Drawing Functions

Once the window is open, you can draw your picture. BG 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. If you haven't selected a color, the paint color will be white.

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). You can change the current pen position to (x, y) (without drawing anything) with a call to:

```
bgMoveTo (x, y);
```

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

```
bgDrawTo (x, y);
```

Notice that this will also update the current position to the new (x, y).

Draw a Point

You can draw a point at position (x, y) with the following statement. Notice that this will also change the current pen position to (x, y).

```
bgDrawPoint (x, y);
```

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 circle is defined 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);
```

Color Functions

Colors are selected with a variety of functions. Once you have selected a color, that color will always be used ... until your program makes a call to select another color.

Set the Paint Color

The paint color is the color used in all of the drawing functions above. There are five 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 following: 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.

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. If you would rather use integers ranging from 0 to 255 to specify the quantities of red, green, and blue, you can use the following call instead:

```
bgSetRGBbyte (red_value, green_value, blue_value);
```

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 following: 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.

Image Functions

Images are implemented in BG as a class. Image objects can either contain an image from a file, or have an empty image that your program fills in.

Create an Empty Image Object

You can create an empty image object (called `my_image` for this example) with `nrows` rows and `ncols` columns using:

```
BGimage my_image (nrows, ncols, type);
```

where `type` is either BG_GREY_IMAGE, BG_HUE_IMAGE, BG_COLOR_IMAGE, BG_RGB_IMAGE, BG_RGBA_IMAGE, or BG_GREY_ALPHA_IMAGE. As an alternative, you can use the following:

```
BGimage *my_image_ptr;
my_image_ptr = new BGimage (nrows, ncols, type);
```

Modifying the Image Array

The resulting object contains a public array (of type `unsigned char`) called `image`. Although the image looks two-dimensional to us, it is stored as a one-dimensional array. For an image with a depth of one byte, you can set the pixel in row `r` and column `c` to 0 with the following statements[8]:

```
index = ((r - 1) * ncols) + c;
my_image.image[index] = 0;
```

Create an Image Object from a File

Sometimes it is useful to start with a pre-existing image. You can create an image object (called `my_image` for this example) using a file (called `my_image.ppm` for this example) with the following call:

```
BGimage my_image ("my_image.ppm", true);
```

The second argument (`true`) indicates that white pixels in the image should be treated as background, and made transparent. This allows you to have irregular-shaped images. If the second argument is `false`, then the resulting image will appear rectangular.

Getting Image Dimensions

Once you have created an image object, you cannot change its height (rows), width (columns) or depth. However, you can find out what these values are by using the following methods (once again, for an image object called `my_image` in this example):

---

[8] In BG images (like OpenGL images) row 0 is the bottom row, and column 0 is the left column,

,

```
int height = my_image.rows();
int width = my_image.columns();
int depth = my_image.depth();
```

Drawing an Image

You can draw an image two different ways. In both cases, you will use the image object's draw method. If you wish the fill the screen with the image in my_image, call:

```
my_image.draw();
```

If, however, you want the image to be at a certain location (x, y) with a certain width (w), you can call:

```
my_image.draw(x, y, w);
```

Writing an Image

You can create a Portable Pixmap file showing whatever's on the screen using the following call:

```
bgWriteImage (filename);
```

Event Callbacks

Event callbacks allow the picture in your window to change while your program is running.

Enabling a Callback

When the callback is enabled, its corresponding function will be called whenever the corresponding event occurs. Each callback type must be enabled separately, as follows:

```
bgEnableCallback (callback_type);
```

where `callback_type` is either BG_KEY, BG_SPECIAL_KEY, or BG_IDLE.

Disabling a Callback

If you disable a callback, then the corresponding event will have no effect. For example, you may want to halt an animation. This is achieved with the following call:

```
bgDisableCallback (callback_type);
```