Back to Title Page Previous Entry Next Entry→

July 4

Well, I've taken time out to watch the French film, "Chaos," about Algerian immigrants. Pretty good. Now, a dormir.

COLOR.

Yeah! De Colores!

This is a topic of some continual confusion, at least for me. Let's start with the idea of "primary colors." At http://www.liquisoft.com/colortheory.html , I read about the "color wheel":

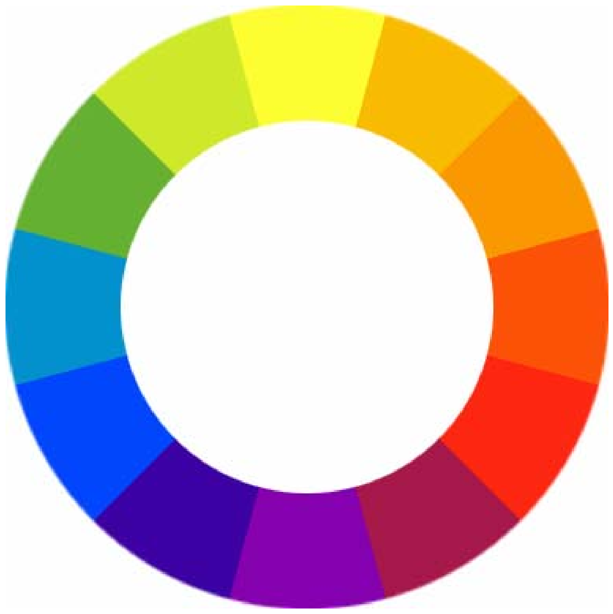
Primary Colors: Red, Yellow, Blue. These 3 colors are the base colors for every other color on the color wheel. This is why they're called "primary." When you mix two primaries together, you get a secondary color.
Also note the triangular positioning of the primary colors on the color wheel, and how the secondary colors are next to them.
Primary colors are useful for designs or art that needs to have a sense of urgency. Primary colors are the most vivid colors when placed next to eachother, which is why you'll notice that most fast food joints use primary colors in their logos, as it evokes speed.

The same web site gives a nice, brief description of RGB color:

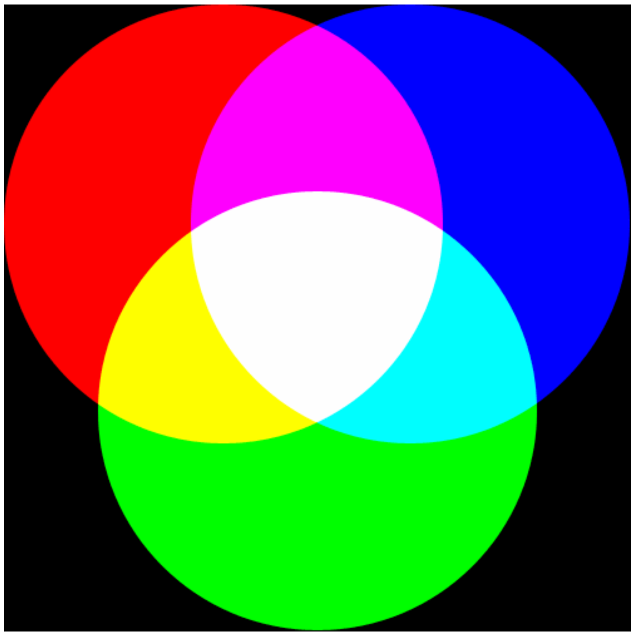
RGB Color: This is color based upon light. Your computer monitor and television use RGB. The name "RGB" stands for Red, Green, Blue, which are the 3 primaries (with green replacing yellow). By combining these 3 colors, any other color can be produced. Remember, this color method is only used with light sources; it does not apply to printing.

The difference between the color wheel and RGB seems to be pigment vs. light. When all pigments are mixed together, you get black. When all light wave frequencies are combined (in equal measure, I guess) you get white. That's a significant difference!

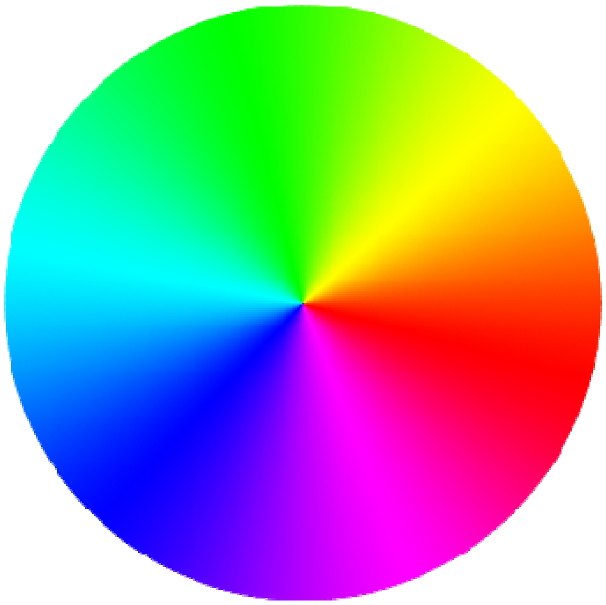
Wikipedia is an online collaborative encyclopedia with some really good stuff. At http://en.wikipedia.org/wiki/Color_theory you find all sorts of color wheels. The RYB (pigment) color wheel is shown with this lovely wheel showing 30 degree increments with secondary and tertiary colors.



Then there's the rgb picture.
The Wikipedia site with perfect 411 on this stuff. For instance, all the links below are very informative (stuff like the hex code of cyan is #00FFFF): "An **additive color system** involves light emitted directly from a source or illuminant of some sort. The additive reproduction process usually uses red, green and blue light to produce the other colors. Combining one of these additive primary colors with another in equal amounts produces the additive secondary colors cyan, magenta, and yellow. Combining all three primary colors in equal intensities produces white. Varying the luminosity of each light eventually reveals the full gamut of those 3 lights.



The 411 on how inkjet printers color has to do with the cmy wheel whose singularity at the center is striking!



"For printing purposes, the colors used are cyan, magenta, and yellow; this model is called the "CMY model". In the CMY model, black is created by mixing all colors, and white is the absence of any colors (assuming white paper). As colors are subtracted to produce black, this is also called the *subtractive* color model. A mix of Cyan, Magenta, and Yellow actually gives a muddy black so normally true black ink is used as well; when black is added, this color model is called the "CMYK model." More recently, it has been shown that the CMY color model is also more accurate for pigment-mixing."

http://en.wikipedia.org/wiki/RGB_color_model seems a pretty good straight source of info on the RGB model that OpenGl uses. Angel talks about parameters being floats in the range from 0.0 to 1.0 so that (1.0, 1.0, 1.0) is white, (1.0, 0.0, 0.0) is red and so forth. There is also RGBA mode, whose fourth parameter A (alpha) is for opacity (the default opacity value 1.0 is used if it is not specified). ALERT: The color precision of the display system may be less than that used to specify a color in glColor*().

The default background color is black (0.0,0.0,0.0) and the default drawing color is white (1.0, 1.0, 1.0), but these can be adjusted using

```
void glColor3{b i f d ub us ui}(TYPE r, TYPE g, TYPE b)
void glColor3{b i f d ub us ui}v(TYPE *color)
void glColor3{b i f d ub us ui}(TYPE r, TYPE g, TYPE b, TYPE a)
void glColor3{b i f d ub us ui}v(TYPE *color)
```

and

```
  void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)
```

specifies the color to clear with…with which to clear.

Specifying the drawing and clearing colors sets the current "color state." The color used to render an geometric object is the current drawing color. Binding colors to objects must be done with care, it seems.

DIMENSIONS
In OpenGL, 2D objects like the white square in the simple program of yesterday are actually 4D objects with the third (z) coordinate set to 0. Never mind about the 4th dimension for now. To simplify matters at first, well just draw to a rectangular canvas and not worry about the perspective of the virtual camera.

To create such a canvas (called the clipping window,) use

```
    void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)
```

Note that this function is in the glu library because it is a special case of the more general #D function `glOrtho()`.

You can draw outside the clipping window, but it will be "clipped out." ;-b

SUBWINDOWS
These are actually called viewports. They're little rectangular subsets of the clipping window- say a menu or whatever.

COORDINATES
There are the coordinates intrinsic to the window (called window or screen coordinates,) which is a simple count of pixels from the left (x) and pixels down from the top (y), but you can, of course, rescale to any coordinates you'd like, OpenGL will automatically convert to screen coordinates somehow. You need to know the size of the display window on the screen (`glutInitWindowSize()`) and how much the user wishes to display (`gluOrtho2D()`).

You need two matrices to perform the coordinate transformations in OpenGL: model-view and projection matrices. Hang on, we'll tackle these in detail soon enough. For now, be aware that `gluOrtho2D()` is used to set the projection matrix in the simple program we have so far. For example, to set up a two-dimensional licpping window with lower left corner (-1.0, -1.0) and upper right corner (1.0, 1.0), we can execute the functions

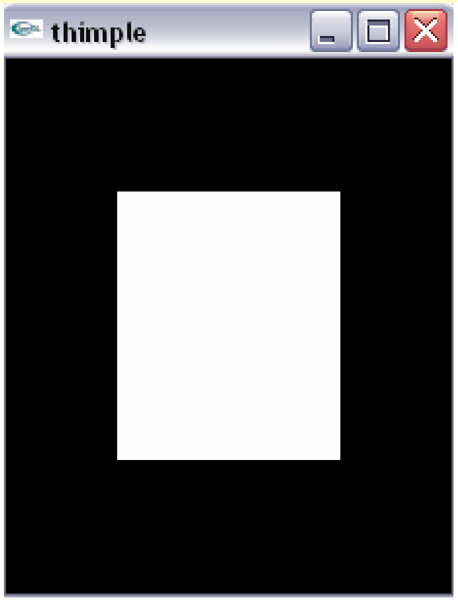
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1.0,-1.0,1.0,1.0);
```

Let's experiment with the new ideas so far. So I enter this code:

```
/* simple02 - From Angel's OpenGL Primer */

#include <GL/glut.h>   // This includes gl.h and glu.h

void display() {
      //clear window
      glClear(GL_COLOR_BUFFER_BIT);

      // draw unit square polygon
      glBegin(GL_POLYGON);
        glVertex2f(-0.5,-0.5);
        glVertex2f(-0.5,0.5);
        glVertex2f(0.5,0.5);
        glVertex2f(0.5,-0.5);
    glEnd();

      //flush GL buffers
      glFlush();
}

void init() {
      //set clear color to black
      glClearColor(0.0,0.0,0.0,0.0);

      //set fill color to white
      glColor3f(1.0,1.0,1.0);

      //set standard orthogonal (look straight at) clipping view
      //with cube of edge 2 centered at origin (default-could be omitted)

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(-1.0,1.0,-1.0,1.0);
}

int main(int argc, char** argv) {

      //init mode and open window in upper-left corner
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(0,0);
    glutCreateWindow("thimple");
      glutDisplayFunc(display);
      init();
```

```
        glutMainLoop();
}
```

Boring, but it does illustrate the major components of the programs we'll be writing.



1. There's a `main()` function to initialize GLUT, put a window on the screen and , identify the callback functions and enter the main loop.
2. An `init()` function to set state variables to initial values.
3. A display callback `display()` that describes the objects to display.
4. Other callbacks? We'll see how to deal with events soon enough.

The `main()` program in this model never needs to change much, depending on which callbacks and menus are needed for an application. The `init()` allows for a lot of detailed state information and desired parameters in one place, separate from the geometry (which is in the display callback) and from dynamics of animation and interaction (in the callbacks.)