Colour

OpenGL

Learning Outcomes

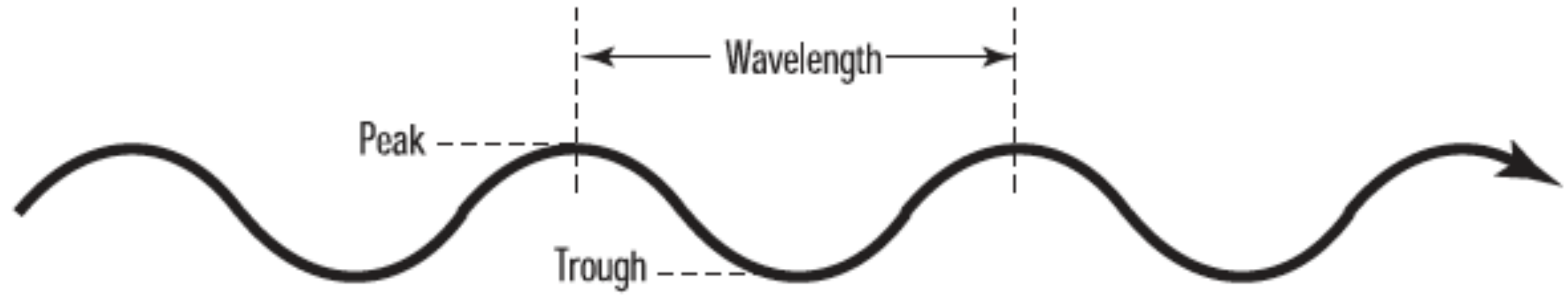
- Have a general understanding of the principles surrounding the colour model in OpenGL

- Have seen the OpenGL colour Cube in action, and appreciate the shading model.

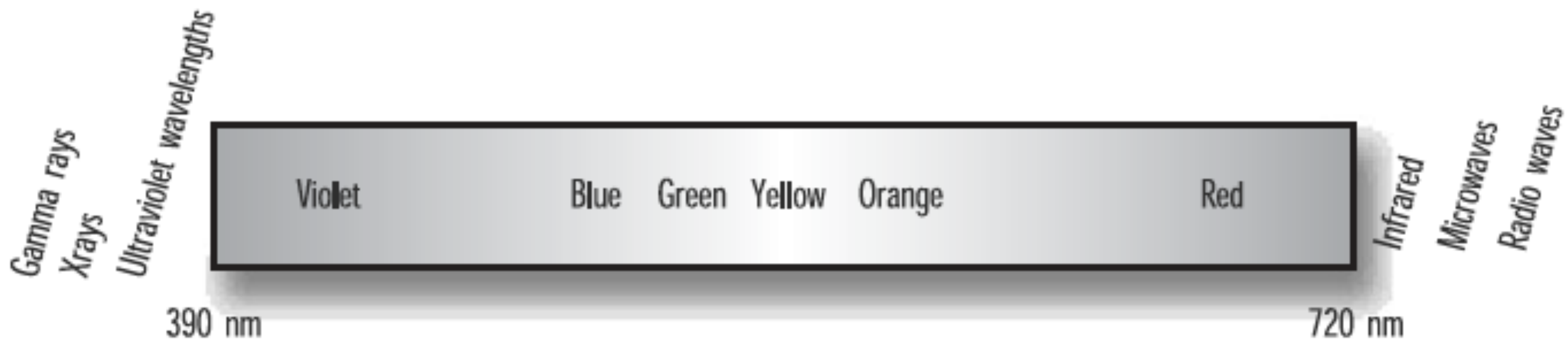
- Understand how glColour operates, and in particular the effect of glColour on polygon rendering with SMOOTH shading enabled

Colour & Light

- Color is simply a wavelength of light that is visible to the human eye.



- Wavelengths of visible light range from 390 nanometers (one billionth of a meter) for violet light to 720 nanometers for red light - called the visible spectrum

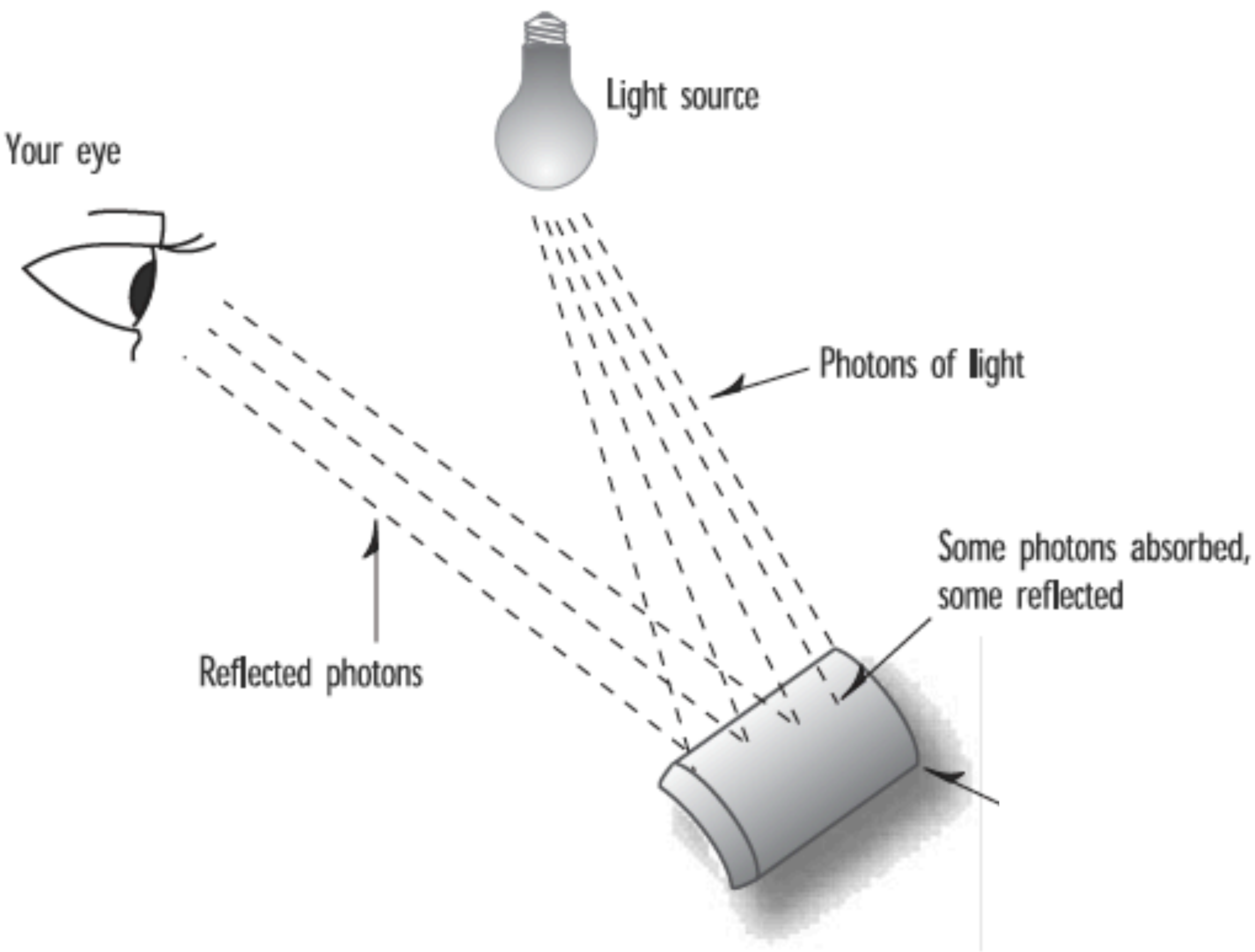


Colour & Reflection

- A white object reflects all wavelengths of colors evenly, and a black object absorbs all wavelengths evenly.
- Considering light as a particle - any given object when illuminated by a light source is struck by photons.
- The reflection of photons from an object depends on the kinds of atoms, the number of each kind, and the arrangement of atoms (and their electrons) in the object

Materials

- Some photons are reflected and some are absorbed (the absorbed photons are usually converted to heat)
- Any given material or mixture of materials reflects more of some wavelengths than others



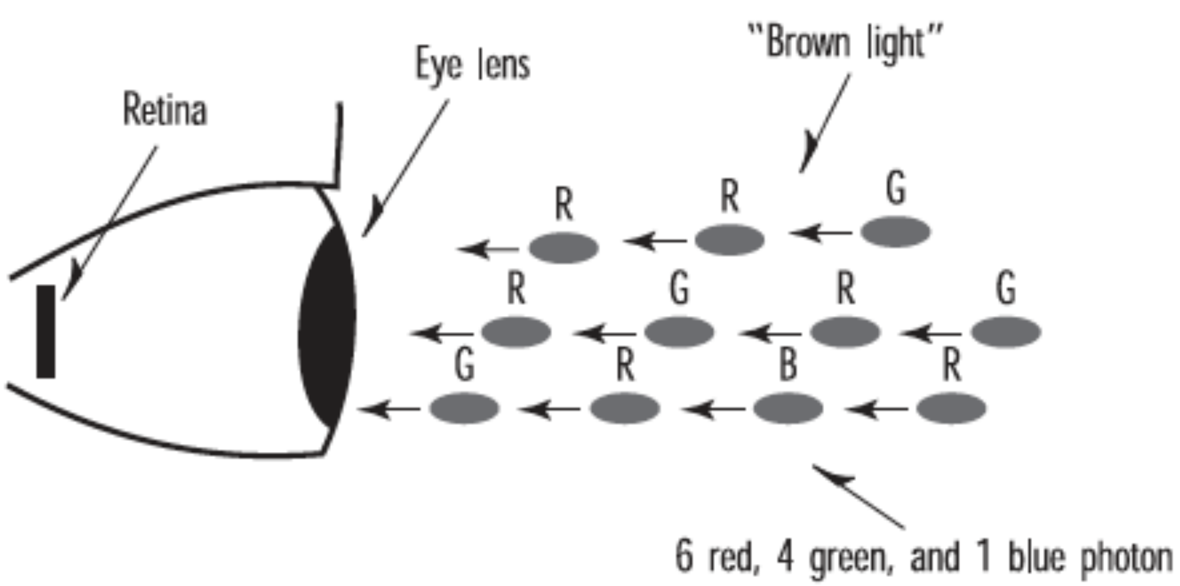
The Eye

- The eye has three kinds of cone cells. All of them respond to photons, but each kind responds most to a particular wavelength.

- One is more excited by photons that have reddish wavelengths; one, by green wavelengths; and one, by blue wavelengths.

- A combination of different wavelengths of various intensities will yield a mix of colors.

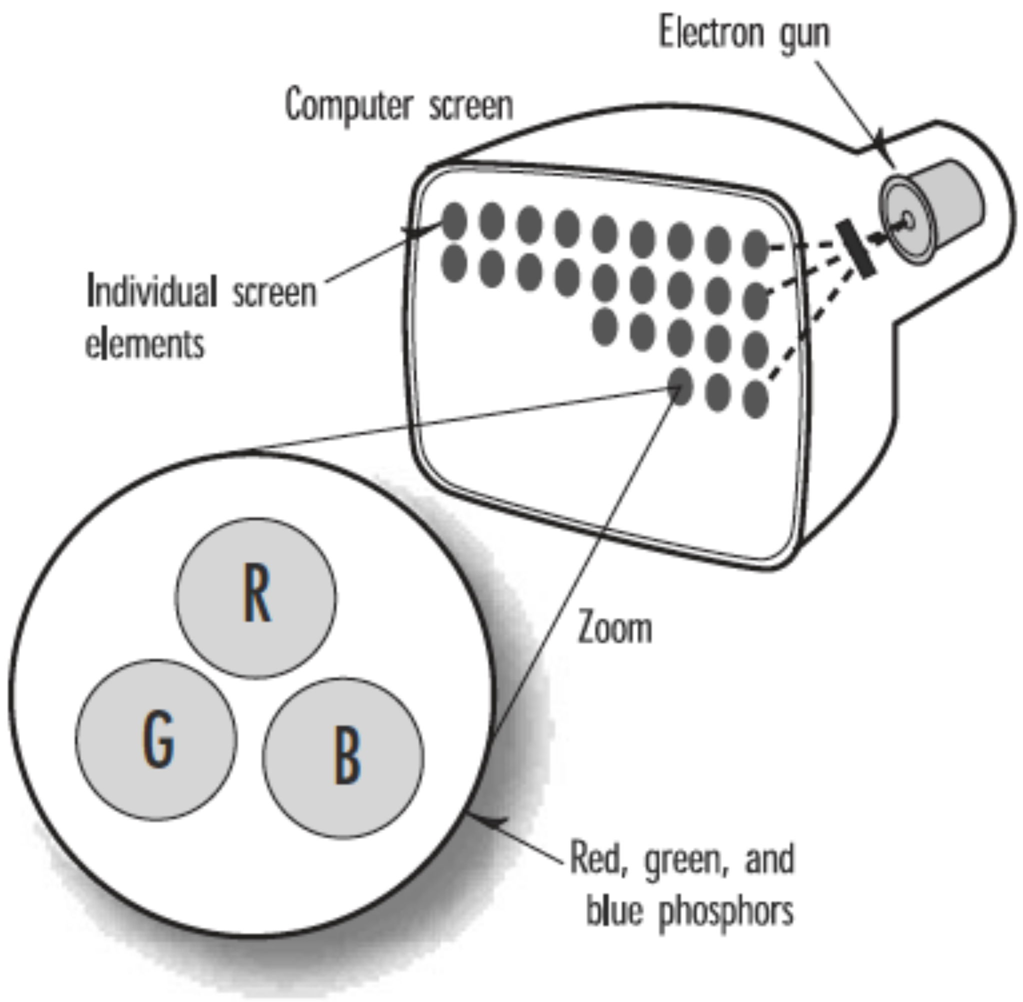
- All wavelengths equally represented thus are perceived as white, and no light of any wavelength is black.



Screens

- Each pixel on your LCD screen has a light behind it and three very small computer-controlled polarized (red, green, and blue) filters.

- Basic LCD technology is based on the polarization of light, and blocking that light with the LCD material electronically



Graphics Hardware: Resolution

- 960-by-640 (iphone) up to 1,900-1,200 (this mac) or more.
- Well-written graphics applications display the same approximate image regardless of screen resolution.
- The user should automatically be able to see more and sharper details as the resolution increases.

Graphics Hardware: Colour Depth

- **Colour Components: Red, Green, Blue**
- An increase in available colors improve the clarity of the resulting image.
- **4 bits per colour component = 12 bits**, rounded to 16 bits to align with machine word size
  - Supports 65,536 different colors, and consumes less memory for the color buffer than the higher bit depth modes.
  - Many graphics applications have very noticeable visual artifacts (usually in color gradations) at this color depth.

Graphics Hardware: Colour Depth

- **8 bits per colour component** - 24 and usually rounded to 32 bit display modes
  - Allows more than 16 million colors onscreen at a time.
  - 8 bits per Red, Green and Blue “Channel” = 24
  - + 8 bits for “Alpha” component - used in some operations to simulate transparency and other effects.

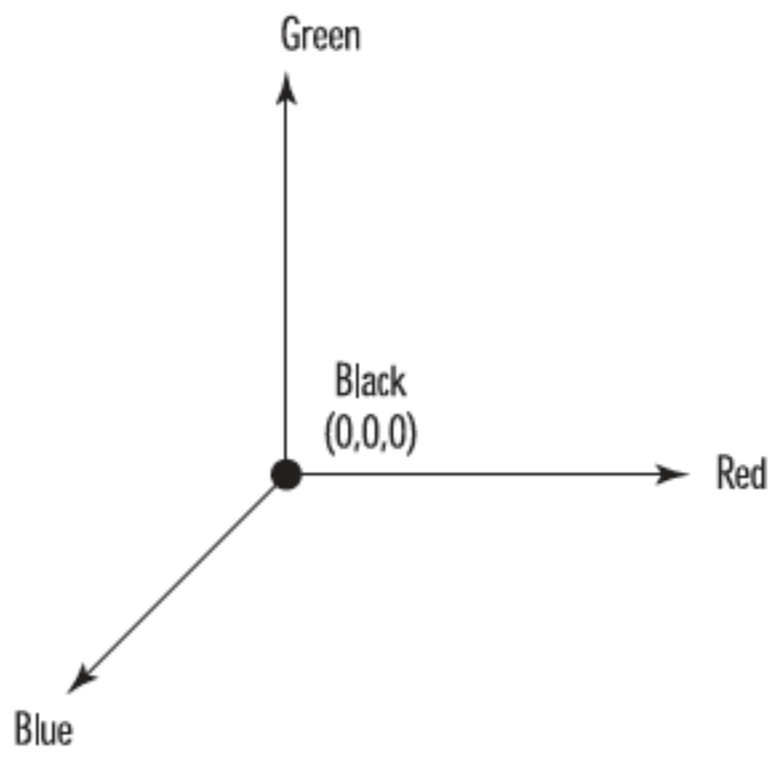
Colour in OpenGL

- Color is specified by three positive color values, can be modeled as a volume called the RGB colorspace

- The red, green, and blue coordinates are specified just like x, y, and z coordinates.

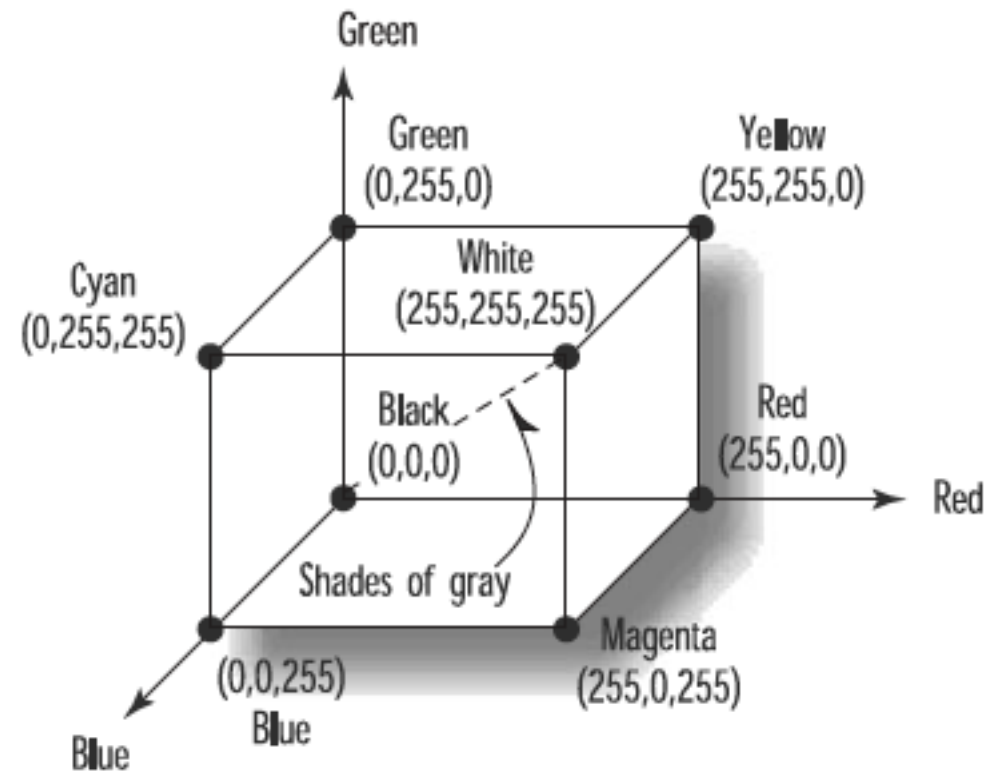
- At the origin (0,0,0), the relative intensity of each component is zero, and the resulting color is black.

- With 8 bits for each component, so 255 along the axis represents full saturation of that component.



Colour Cube

- We then end up with a cube measuring 255 on each side.
- The corner directly opposite black, where the concentrations are (0,0,0), is white, with relative concentrations of (255,255,255).
- At full saturation (255) from the origin along each axis lie the pure colors of red, green, and blue.
- This “color cube” contains all the possible colors, either on the surface of the cube or within the interior of the cube.
- Eg all possible shades of gray between black and white lie internally on the diagonal line between the corner at (0,0,0) and the corner at (255,255,255).



glColour function

```
void glColorNT(red, green, blue, alpha);
```

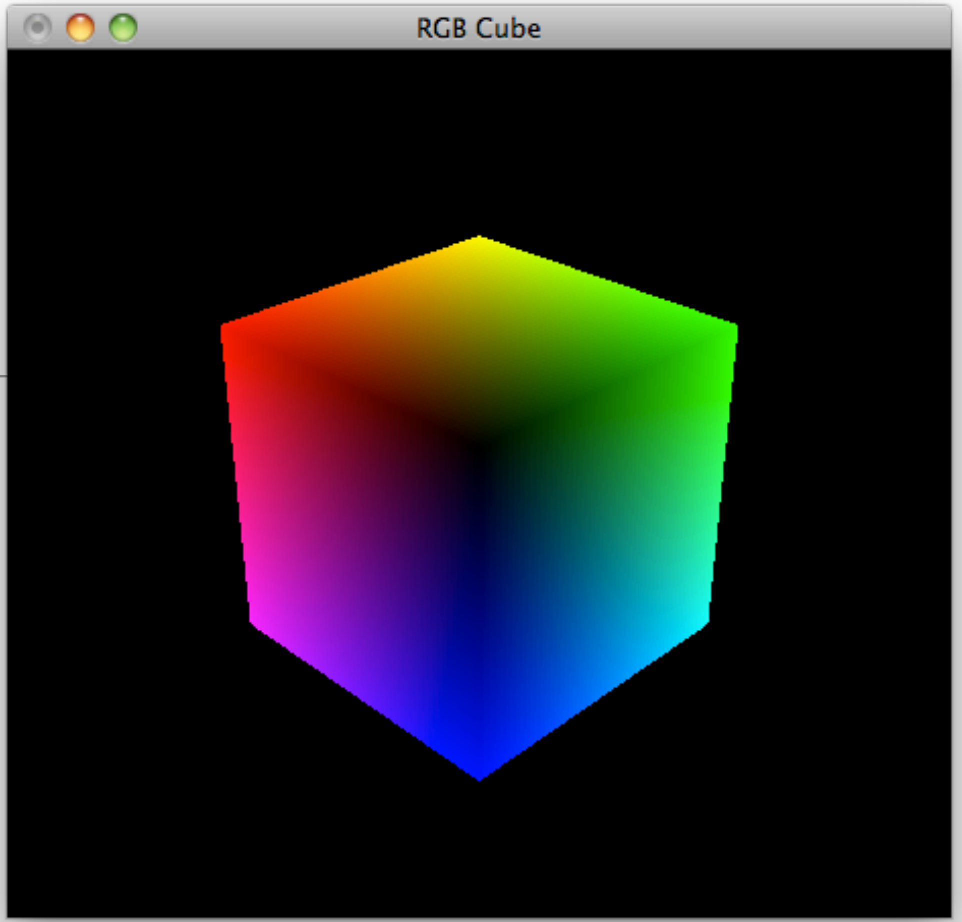
- N = number of parameters
  - 3 RGB
  - 4 RGBA (alpha)
- T = Type
  - b, d, f, i, s, ub, ui, or us for byte, double, float, integer, short, unsigned byte, unsigned integer, and unsigned short
- Another version of the function has a v appended
  - to the end; this version takes an array that contains the arguments (the v stands for vectored)

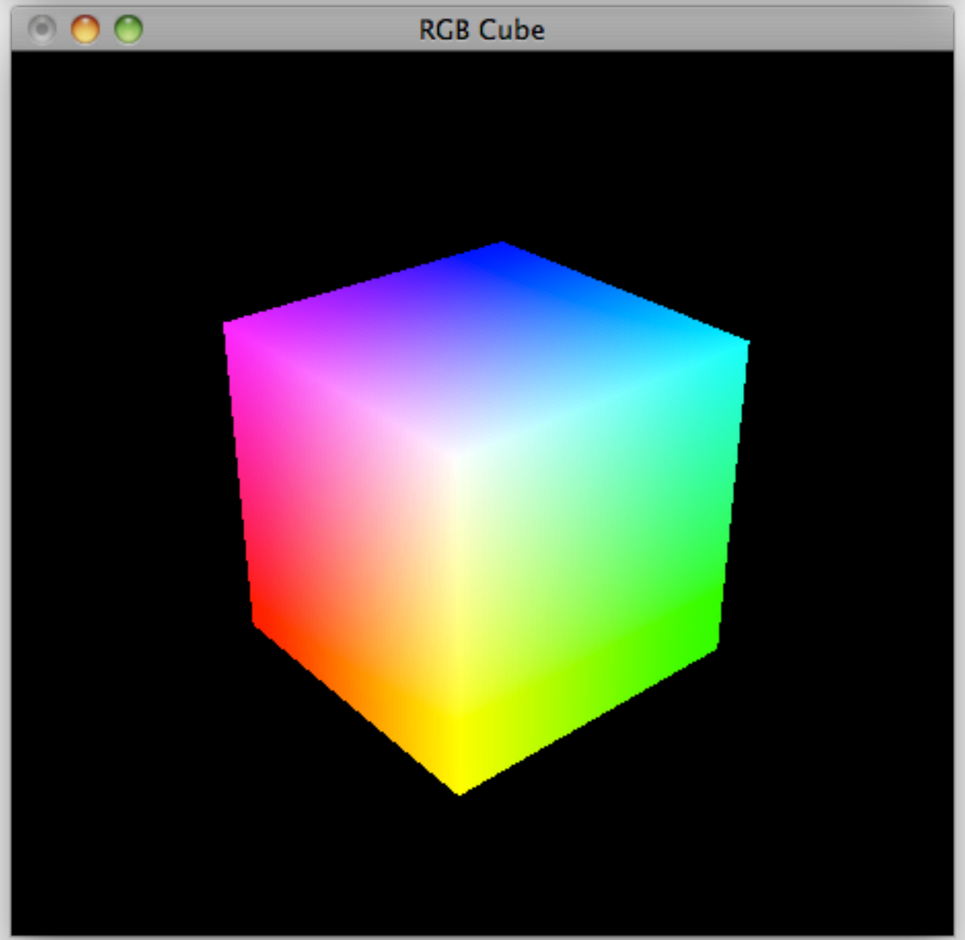
glColor3f

- Most OpenGL programs that you'll see use glColor3f and specify the intensity of each component as 0.0 for none or 1.0 for full intensity.
- Internally, OpenGL represents color values as floating-point values.
- As higher resolution floating point color buffers evolve using floats will be more faithfully represented by the color hardware.

Colour Cube

- The surface of this cube shows the color variations from black on one corner to white on the opposite corner.
- Red, green, and blue are present on their corners 255 units from black.
- Additionally, the colors yellow, cyan, and magenta have corners showing the combination of the other three primary colors





Colour Cube Code

- Draw 6 QUADS
- Each Quad will specify appropriate colour at the corners

```
void SetupRC()
{
  // Black background
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

  glEnable( GL_DEPTH_TEST);
  glShadeModel( GL_SMOOTH);
}
```

```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  //...

  glBegin( GL_QUADS);
    frontFace();
    backFace();
    topFace();
    bottomFace();
    leftFace();
    rightFace();
  glEnd();

  //...
  glutSwapBuffers();
}
```

Verbose Version

```
void leftFace()
{
  // White
  glColor3f(1.0f, 1.0f, 1.0f);
  glVertex3f(50.0f, 50.0f, 50.0f);

  // Cyan
  glColor3f(0.0f, 1.0f, 1.0f);
  glVertex3f(50.0f, 50.0f, -50.0f);

  // Green
  glColor3f(0.0f, 1.0f, 0.0f);
  glVertex3f(50.0f, -50.0f, -50.0f);

  // Yellow
  glColor3f(1.0f, 1.0f, 0.0f);
  glVertex3f(50.0f, -50.0f, 50.0f);
}

void rightFace()
{
  // Magenta
  glColor3f(1.0f, 0.0f, 1.0f);
  glVertex3f(-50.0f, 50.0f, 50.0f);

  // Blue
  glColor3f(0.0f, 0.0f, 1.0f);
  glVertex3f(-50.0f, 50.0f, -50.0f);

  // Black
  glColor3f(0.0f, 0.0f, 0.0f);
  glVertex3f(-50.0f, -50.0f, -50.0f);

  // Red
  glColor3f(1.0f, 0.0f, 0.0f);
  glVertex3f(-50.0f, -50.0f, 50.0f);
}
```

```
void topFace()
{
 // Cyan
 glColor3f(0.0f, 1.0f, 1.0f);
 glVertex3f(50.0f, 50.0f, -50.0f);

 // White
 glColor3f(1.0f, 1.0f, 1.0f);
 glVertex3f(50.0f, 50.0f, 50.0f);

 // Magenta
 glColor3f(1.0f, 0.0f, 1.0f);
 glVertex3f(-50.0f, 50.0f, 50.0f);

 // Blue
 glColor3f(0.0f, 0.0f, 1.0f);
 glVertex3f(-50.0f, 50.0f, -50.0f);
}

void bottomFace()
{
  // Green
  glColor3f(0.0f, 1.0f, 0.0f);
  glVertex3f(50.0f, -50.0f, -50.0f);

  // Yellow
  glColor3f(1.0f, 1.0f, 0.0f);
  glVertex3f(50.0f, -50.0f, 50.0f);

  // Red
  glColor3f(1.0f, 0.0f, 0.0f);
  glVertex3f(-50.0f, -50.0f, 50.0f);

  // Black
  glColor3f(0.0f, 0.0f, 0.0f);
  glVertex3f(-50.0f, -50.0f, -50.0f);
}
```

```
void frontFace()
{
  // White
  glColor3f(255, 255, 255);
  glVertex3f(50.0f, 50.0f, 50.0f);

  // Yellow
  glColor3f(255, 255, 0);
  glVertex3f(50.0f, -50.0f, 50.0f);

  // Red
  glColor3f(255, 0, 0);
  glVertex3f(-50.0f, -50.0f, 50.0f);

  // Magenta
  glColor3f(255, 0, 255);
  glVertex3f(-50.0f, 50.0f, 50.0f);
}

void backFace()
{
  // Cyan
  glColor3f(0.0f, 1.0f, 1.0f);
  glVertex3f(50.0f, 50.0f, -50.0f);

  // Green
  glColor3f(0.0f, 1.0f, 0.0f);
  glVertex3f(50.0f, -50.0f, -50.0f);

  // Black
  glColor3f(0.0f, 0.0f, 0.0f);
  glVertex3f(-50.0f, -50.0f, -50.0f);

  // Blue
  glColor3f(0.0f, 0.0f, 1.0f);
  glVertex3f(-50.0f, 50.0f, -50.0f);
}
```

Colour Class

```
struct Color
{
  float R;
  float G;
  float B;
  float A;

  static Color White;
  static Color Yellow;
  static Color Red;
  static Color Magenta;
  static Color Cyan;
  static Color Green;
  static Color Black;
  static Color Blue;


  Color();
  Color(float r, float g, float b, float a=1.0f);
  Color(int r, int g, int b, int a=255);

  void render();
   void renderClear();
};
```

```
Color Color::Black   (0,     0,    0);
Color Color::Blue    (0,     0,  255);
Color Color::Green   (0,   255,    0);
Color Color::Cyan    (0,   255,  255);
Color Color::Red     (255,   0,    0);
Color Color::Magenta (255,   0,  255);
Color Color::Yellow  (255, 255,    0);
Color Color::White   (255, 255,  255);

Color::Color()
{
  R = G = B = A = 1.0f;
}

Color::Color(float r, float g, float b, float a)
{
  R = r;
  G = g;
  B = b;
  A = a;
}

Color::Color(int r, int g, int b, int a)
{
  R = (float) r / 255.0f;
  G = (float) g / 255.0f;
  B = (float) b / 255.0f;
  A = (float) a / 255.0f;
}

void Color::render()
{
  glColor4f(R,G,B,A);
}

void Color::renderClear()
{
  glClearColor(R,G,B, 1.0f);
}
```

Use Our World Framework

```
struct Quad : public Geometry
{
  void render()
  {
    glBegin( GL_QUADS);
      frontFace();
      backFace();
      topFace();
      bottomFace();
      leftFace();
      rightFace();
     glEnd();
  }
};
```

```
int main(int argc, char* argv[])
{
  theWorld.setCmdlineParams(&argc, argv);
  theWorld.initialize(800,800, "Color Cube");

  Quad *quad = new Quad();
  theWorld.add(quad);

  theWorld.setProjection(new Perspective(35, Range(1,1000), 500));

  theWorld.start();
  return 0;
}
```

To Make it Rotate

- Make quad a Global Variable
- In specialKeyPress - use the rotation function we already have implemented for assignment 1 solution.

```
struct Quad : public Geometry
{
  void render()
  {
    glRotatef(angle, rotationAxis.X, rotationAxis.Y, rotationAxis.Z);
    glBegin( GL_QUADS);
      frontFace();
      backFace();
      topFace();
      bottomFace();
      leftFace();
      rightFace();
     glEnd();
  }
};
```

```
Quad *quad;

void World::specialKeypress(int key)
{
  if (key == GLUT_KEY_UP)
    quad->rotate(1, Vector3::UnitX);
  if (key == GLUT_KEY_DOWN)
    quad->rotate(-1, Vector3::UnitX);
  if (key == GLUT_KEY_LEFT)
    quad->rotate(1, Vector3::UnitY);
  if (key == GLUT_KEY_RIGHT)
    quad->rotate(-1, Vector3::UnitY);

   glutPostRedisplay();
}
```

Colour Cube Specification

```
Color colours[][6] =
{
  {Color::White,   Color::Yellow, Color::Red,     Color::Magenta},
  {Color::Cyan,    Color::Green,  Color::Black,   Color::Blue},
  {Color::Cyan,    Color::White,  Color::Magenta, Color::Blue},
  {Color::Green,   Color::Yellow, Color::Red,     Color::Black},
  {Color::White,   Color::Cyan,   Color::Green,   Color::Yellow},
  {Color::Magenta, Color::Blue,   Color::Black,   Color::Red}
};
Vector3 vertices[][6] =
{
  {Vector3(50.0f, 50.0f, 50.0f),   Vector3(50.0f, -50.0f, 50.0f),  Vector3(-50.0f, -50.0f, 50.0f),  Vector3(-50.0f, 50.0f, 50.0f)  },
  {Vector3(50.0f, 50.0f, -50.0f),  Vector3(50.0f, -50.0f, -50.0f), Vector3(-50.0f, -50.0f, -50.0f), Vector3(-50.0f, 50.0f, -50.0f) },
  {Vector3(50.0f, 50.0f, -50.0f),  Vector3(50.0f, 50.0f, 50.0f),   Vector3(-50.0f, 50.0f, 50.0f),   Vector3(-50.0f, 50.0f, -50.0f) },
  {Vector3(50.0f, -50.0f, -50.0f), Vector3(50.0f, -50.0f, 50.0f),  Vector3(-50.0f, -50.0f, 50.0f),  Vector3(-50.0f, -50.0f, -50.0f)},
  {Vector3(50.0f, 50.0f, 50.0f),   Vector3(50.0f, 50.0f, -50.0f),  Vector3(50.0f, -50.0f, -50.0f),  Vector3(50.0f, -50.0f, 50.0f)  },
  {Vector3(-50.0f, 50.0f, 50.0f),  Vector3(-50.0f, 50.0f, -50.0f), Vector3(-50.0f, -50.0f, -50.0f), Vector3(-50.0f, -50.0f, 50.0f) }
 };
```

Rendering the Cube

```
void drawFace(Color colours[], Vector3 vertices[])
{
  for (int i=0; i<4; i++)
  {
    colours[i].render();
    vertices[i].render();
  }
}
```

```
  glBegin( GL_QUADS);
  for (int i=0; i<6; i++)
  {
    drawFace(colours[i], vertices[i]);
  }
  glEnd();
```

glColour Definition

- Working definition for glColor: *sets the current color that is used for all vertices drawn after the call.*

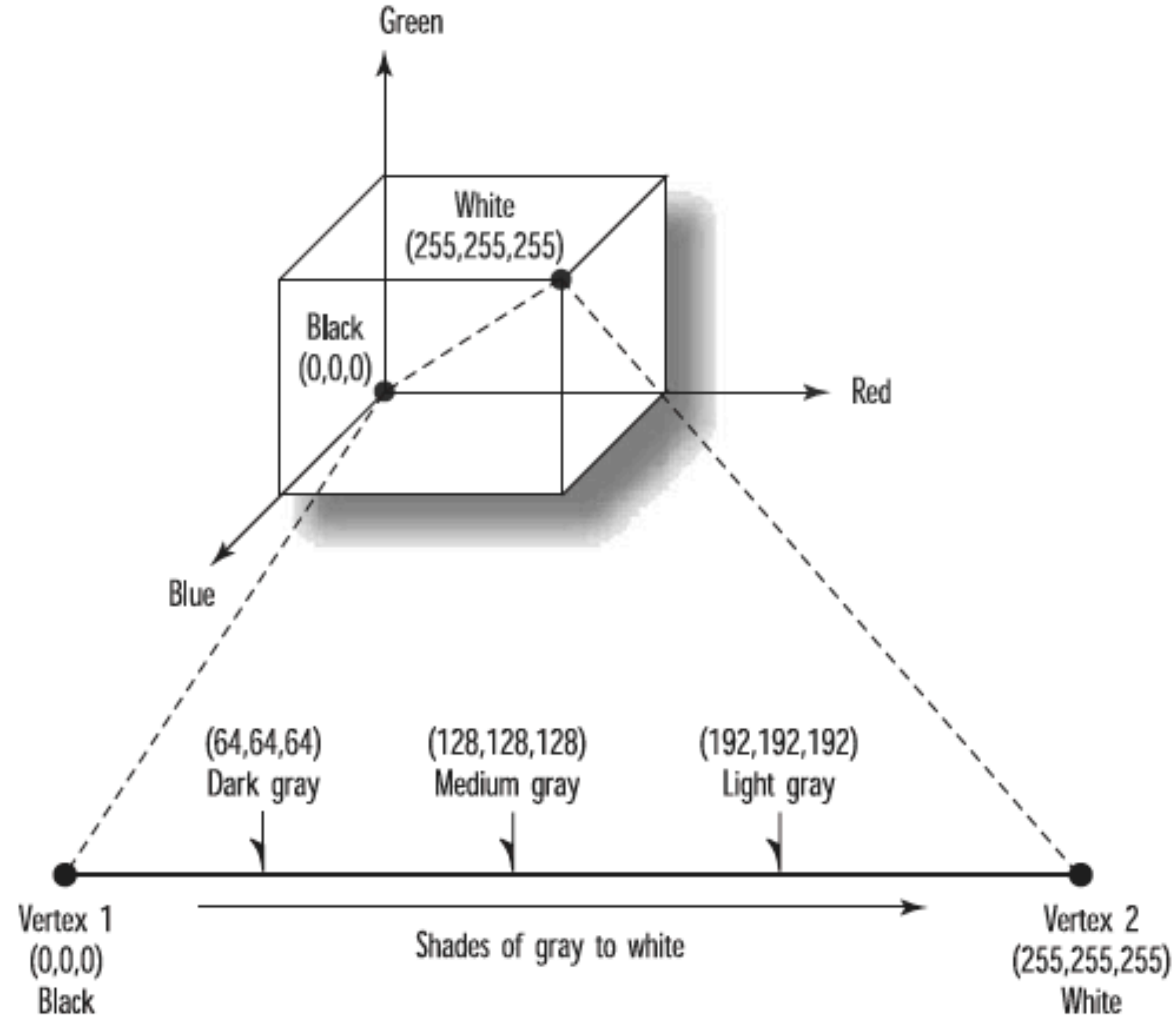
- If we specify a different color for each vertex of a primitive (point, line, or polygon), what color is the interior?

- For Points: A point has only one vertex, and whatever color you specify for that vertex is the resulting color for that point

glColor & Lines

- A line, however, has two vertices, and each can be set to a different color.
- The color of the line depends on the shading model. Shading is simply defined as the smooth transition from one color to the next.
- Any two points in the RGB colorspace can be connected by a straight line.
- Smooth shading causes the colors along the line to vary as they do through the color cube from one color point to the other.

- Can do shading mathematically by finding the equation of the line connecting two points in the three-dimensional RGB colorspace.
- Then you can simply loop through from one end of the line to the other, retrieving coordinates along the way to provide the color of each pixel on the screen.
- OpenGL implements this algorithm via GL_SMOOTH shading

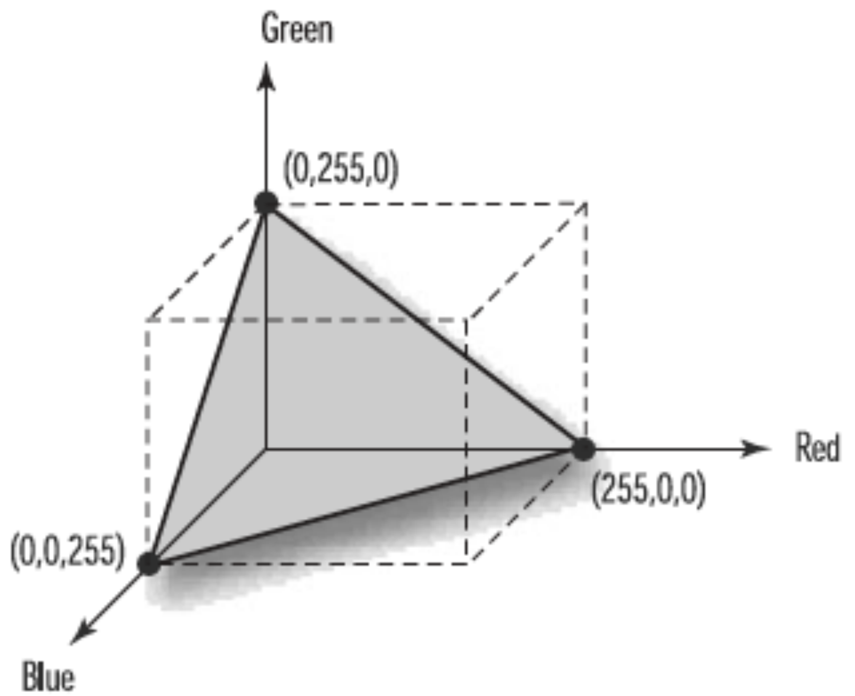



```
void setupRC()
{
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

  glEnable( GL_DEPTH_TEST);
  glShadeModel( GL_SMOOTH);
}
```

Polygon Shading

- More complex for polygons.
- E.g. A triangle can also be represented as a plane within the color cube.
- Draw a triangle with each vertex at full saturation for the red, green, and blue color components.



Triangle Class

- Add colour to our triangle class:

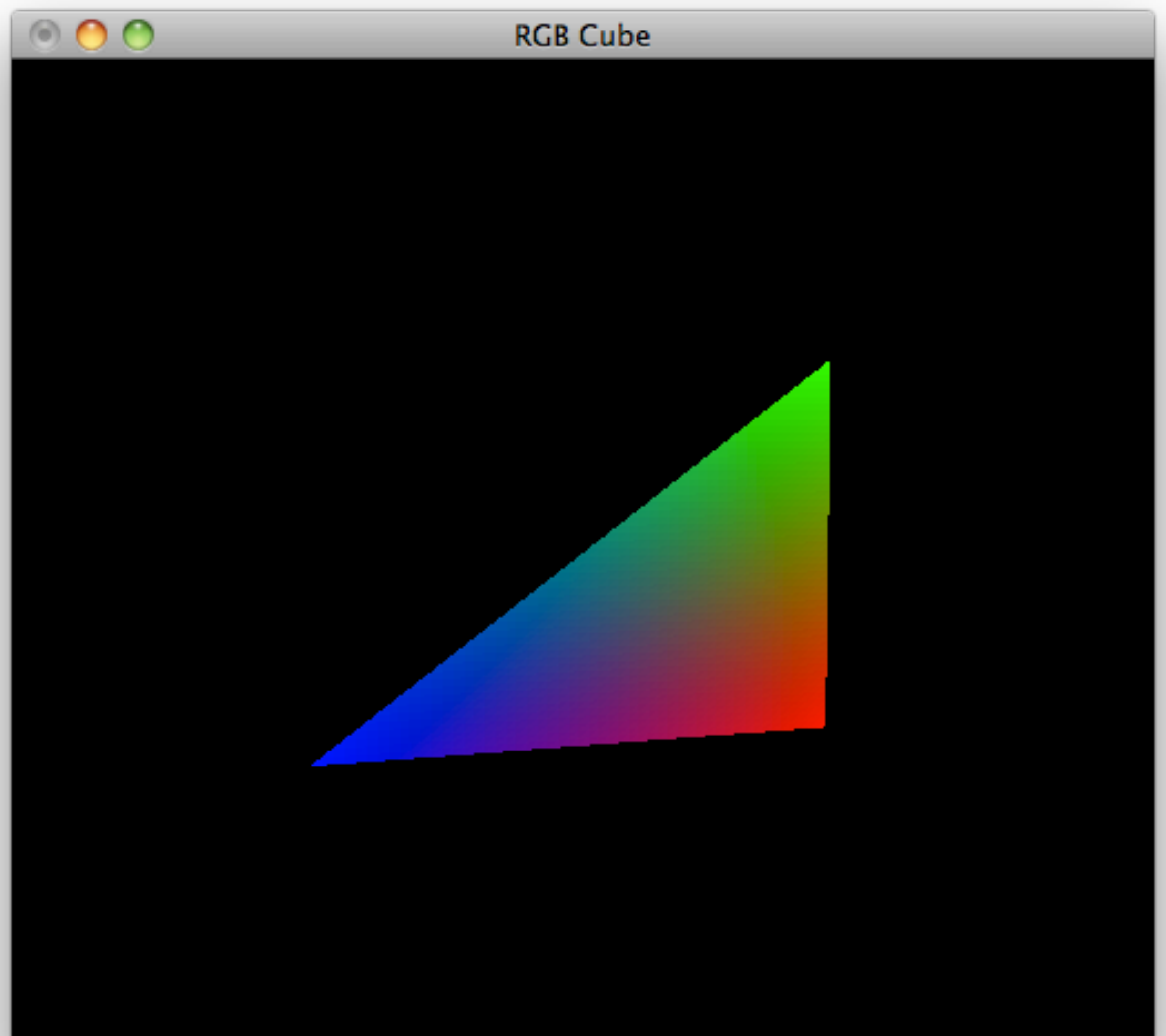
```
struct Triangle : public Geometry
{
  Vector3 p1, p2, p3;
  Color  c1, c2, c3;

  Triangle(std::istream& is);
  Triangle(Vector3 p1, Vector3 p2, Vector3 p3);
  Triangle(Vector3 p1, Vector3 p2, Vector3 p3,
           Color    c1, Color   c2, Color   c3);
  void render();
};
```

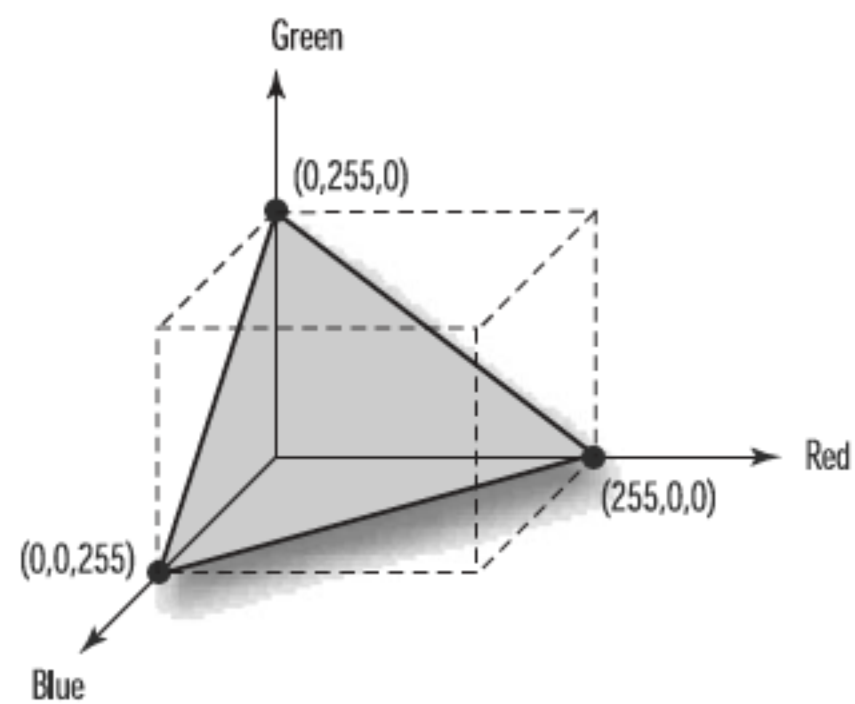
```
void Triangle::render()
{
  glBegin( GL_TRIANGLES);
    c1.render();
    p1.render();
    c2.render();
    p2.render();
    c3.render();
    p3.render();
  glEnd();
}
```

Smooth Shading Triangle

```
glShadeModel( GL_SMOOTH);
Triangle t (Vector3(-50.0f, -50.0f,   50.0f), Vector3(50.0f, -50.0f, -50.0f), Vector3(50.0f, 50.0f, -50.0f),
             Color::Blue,                       Color::Red,                        Color::Green);
t.render();
```

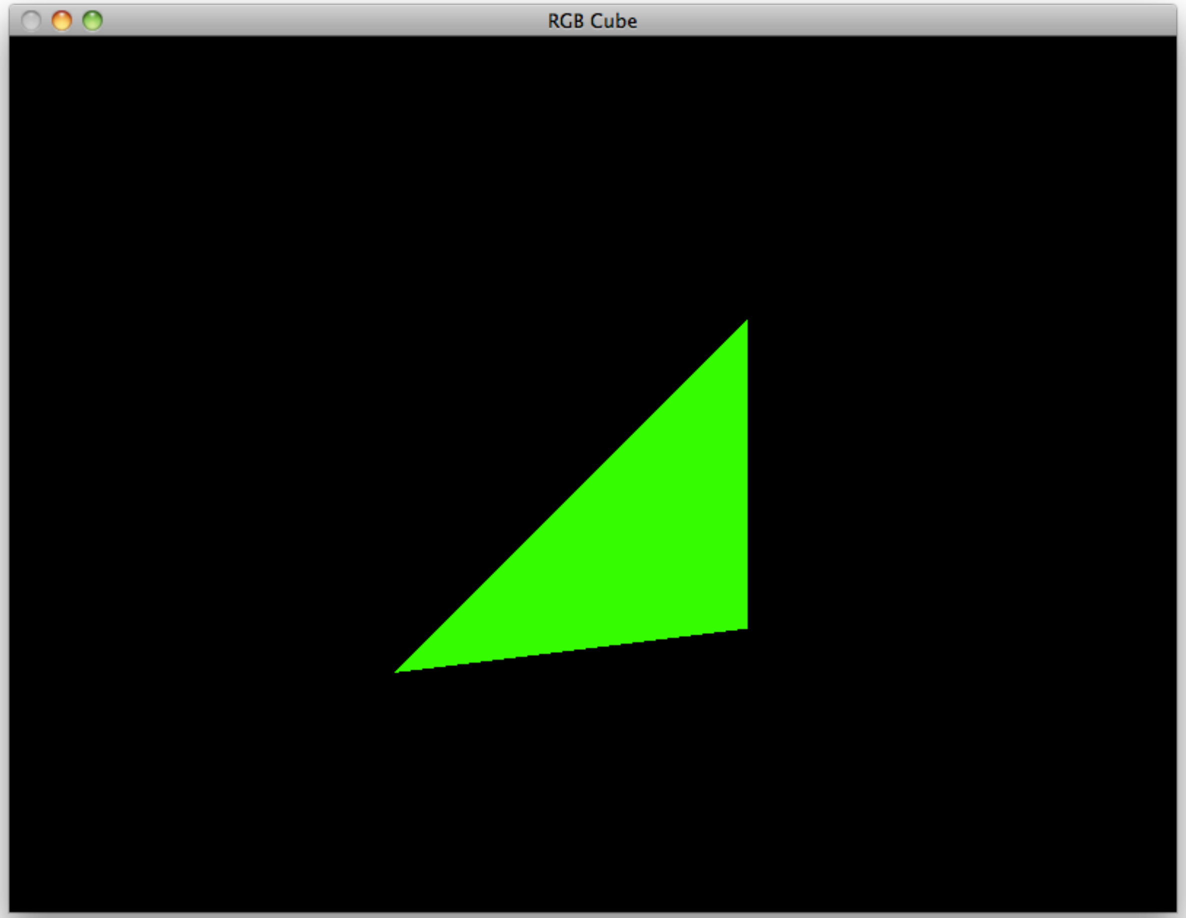


- Because smooth shading is specified, the interior of the triangle is shaded to provide a smooth transition between each corner



Flat Shading Model

```
glShadeModel( GL_FLAT);
Triangle t (Vector3(-50.0f, -50.0f,   50.0f), Vector3(50.0f, -50.0f, -50.0f), Vector3(50.0f, 50.0f, -50.0f),
              Color::Blue,                    Color::Red,                        Color::Green);
t.render();
```



- Flat shading means that no shading calculations are performed on the interior of primitives.
- Generally, with flat shading, the color of the primitive's interior is the color that was specified for the last vertex.
- The only exception is for a GL_POLYGON primitive, in which case the color is that of the first vertex.