OpenGL IV: Color Index Mode

Dale Rogerson
Microsoft Developer Network Technology Group

January 19, 1995

Click to open or copy the files in the EasyCI sample application for this technical article.

Click to open or copy the files in the GLlib DLL for this technical article.

Abstract

This article explores the Windows NT™ implementation of OpenGL™ color index mode. In color index mode, colors are specified as indexes into a palette instead of as levels of red, green, and blue. The EasyCI sample application (provided with this article) is a conversion of EasyGL that uses color index mode. EasyCI uses the GLlib.DLL, also included with this article.

Introduction

I still remember visiting Grandpa Albert every year before school started. Grandpa Albert and his twin brother Bartholomew, locally known as the Bert and Bart brothers, would take us kids fly fishing up on the North Fork of the Snoqualmie River. Now, Great-Uncle Bartholomew would take every fly that he owned with him. He had this large MyFishingBuddyPal tackle box in which he carried thousands of flies. He knew all of the flies and referred to them by name; for example, Super Humbler Bumbler Bee #2, Megamoth Pro, or Jake's Snake. Grandpa Albert, on the other hand, would study the weather conditions before he left and would select what he considered the 17 most appropriate flies for those conditions. He would arrange these flies on his hat. If you asked him which fly he was using, he would tell you the location of the fly on his hat. Grandpa Albert was no computer scientist, so this number was one-based.

Amazingly enough, Bart's method was similar to the RGBA color mode found in OpenGL™. Grandpa Albert's method was not unlike OpenGL's color index mode, because he picked from a palette of flies on his hat by index.

The previous OpenGL articles in this series (see bibliography) all used RGBA color mode. RGBA color mode uses the **glColor** command to specify colors by the levels of red, green, or blue they contain; for example:

```
glColor3d(1.0, 0.14, 0.6667) ; // purple people eater purple
```

In color index mode, the color is specified by its index into a palette of colors; for example:

```
glIndexs(37) ;
```

The command above picks the thirty-eighth color in the palette. (Because OpenGL was designed by computer scientists and not by Grandpa Albert, the index is zero-based.) The thirty-eighth color can be anything from mauve to Super Humbler Bumbler Bee yellow, depending on what you put in the palette. In practice, you don't use **glIndex** all that much (you use **glMaterial** instead); but we'll talk more about that later.

In this article, we'll examine the advantages and disadvantages of using color index mode. We'll also look at some of the differences between a program that uses color index mode and one that uses RGBA mode. The information in this article is based on my experience converting the EasyGL sample application to EasyCI, which uses color index mode. The changes I made to EasyGL and GLlib are discussed in the last section of the article. For information on EasyGL and GLlib, see my article "OpenGL III: Building an OpenGL C++ Class" in the MSDN Library.

Choosing Color Index Mode

RGBA mode and color index mode are mutually exclusive: You can use either mode, but not simultaneously. In the Windows NT™ implementation of OpenGL, you choose the mode by changing a field

in the pixel format descriptor and calling the **ChoosePixelFormat** and **SetPixelFormat** functions. Change the **iPixelType** field of the **PIXELFORMATDESCRIPTOR** structure to indicate which mode you would like to use. For RGBA mode, use:

```
pfd.iPixelType = PFD_TYPE_RGBA ;
```

For color index mode, use:

```
pfd.iPixelType = PFD_TYPE_COLORINDEX ;
```

In GLlib.DLL, the pixel format descriptor is set up in **CGL::Create**.

Once you set up a pixel format for a window, you cannot change it. Therefore, if you need to switch between the two modes, you will have to destroy and recreate the window. This information is buried in the documentation for **SetPixelFormat** in the Win32® Software Development Kit (SDK) for Windows NT.

Reasons for Choosing Color Index Mode

Chapter 5 of the *OpenGL Programming Guide* (hereafter called the "Red Book"; see the bibliography at the end of this article) explains some of the reasons for choosing color index mode instead of RGBA mode. In this section, I will give you my own reasons for using color index mode. My reasons are more specific to the Windows NT implementation than those given in the Red Book. Color index mode gives you:

- More control over the palette
- The ability to create identity palettes
- The ability to use false colors
- The ability to use palette animation
- Greater speed

More of what you want

One reason for using color index mode is to get the colors you want. If you are running on an 8-bits-per-pixel (bpp) device, you have a 256-color palette. As we discussed in the article "OpenGL II: Windows Palettes in RGBA Mode" in the MSDN Library, you have control over only 236 of the palette colors. To make matters worse, the Windows NT implementation of OpenGL requires that these colors be arranged in a 3-3-2 format. You can change these colors slightly by using gamma correction, but you are basically stuck with the 3-3-2 format. If your application needs 128 reds or 200 grays, you are out of luck.

In color index mode, the palette can have the 236 colors that you want. If you want 236 reds, you can have them. A scene rendered with color index mode can look more brilliant and alive than the same scene rendered with RGBA mode, because you can choose brighter colors from the color index mode palette.

The SYSPAL_NOSTATIC and PC_NOCOLLAPSE flags give you more control over the palette. In practice, however, these flags aren't very useful. For more information, read "The Palette Manager: How and Why" in the MSDN Library. Other good articles on palettes are listed in the bibliography at the end of this article.

Identity palettes

If you need to render an OpenGL scene on a bitmap and blt it to the screen, you might need to use color index mode to speed up the blting.

Fast blting is the key to fast animation. The key to fast blting is using identity palettes. The key to identity palettes? Well, read Nigel Thompson's book *Animation Techniques for Win32: A C++ Programmer's Guide to DIBs, Palettes, and Sprites*.

An identity palette is a logical palette that is identical to the system palette. A logical palette must have fewer than 236 colors (not including system colors) before it can aspire to be an identity palette. The first color in the logical palette that is not a system color is placed at (zero-based) index 10 in the system

palette.

The palette that we build when PFD_NEED_PALETTE is set cannot be an identity palette, because the location of the color within the palette is significant. Without an identity palette, blting is slow, because each pixel has to be translated from the logical palette to the system palette. If we use color index mode, we can choose almost any palette we want, so we can pick a palette that has a better chance of being an identity palette. I will explore this issue further in a future article on OpenGL and rendering on bitmaps.

False color

Some people claim that the main reason applications use color index mode is because they use color to illustrate another dimension. For example, a mapping program might use color to show the height of a particular area on the map. The mapping program might show areas between 1000 and 2000 feet as green, areas between 5000 and 10,000 feet as brown, and the top of Mount Tahoma as white.

An application could use color index mode to illustrate the pressure on an aircraft fuselage: The high pressure areas could be red, and the low pressure areas could be blue, with a whole range of colors in between. An aeronautical engineer could quickly glance at the three-dimensional (3-D) representation of the plane and determine the low-pressure locations instantly.

In the last two examples, we don't necessarily care what the colors are; we care only that each color is different. Color index mode fits the bill nicely. It is fairly easy to create a function **f(*x*)** that returns the index, where *x* is the pressure or height of a point. Note, however, that RGBA mode can do the same thing by using an array of colors.

Palette animation

Another use for color index mode is palette animation. You can change the colors on the screen instantly by changing the values in the palette. You do not have to repaint the scene, because the palette indexes in screen memory are still valid. The same index points to the new color in the palette.

Speed

Color index mode can be faster than RGBA mode, because calculations are performed on one color channel rather than three. However, the performance improvement is not exactly a factor of three, because you may need to perform tasks that are unique to color index mode. For example, lighting calculations in color index mode require a light intensity to be calculated from a weighted sum of the light's RGB components.

Reasons for Not Choosing Color Index Mode

The Red Book lists some of the reasons for not choosing color index mode. The main reason is that OpenGL is unable to perform much of its magic in this mode. For example, **glColorMaterial**, which EasyGL and GLEasy both use, is not available in color index mode. In fact, only GL_DIFFUSE and GL_SPECULAR of the ten **glLight** parameters are used in color index mode. Similarly, only GL_SHININESS and GL_COLOR_INDEXES of the seven **glMaterial** parameters affect color index mode.

GL_SHININESS

The *OpenGL Reference Manual* (also called the "Blue Book"; see the bibliography at the end of this article) states that GL_COLOR_INDEXES is the only **glMaterial** parameter that works in color index mode. This is incorrect. The GL_SHININESS parameter also works in color index mode. EasyGI uses the GL_SHININESS parameter. If you don't use GL_SHININESS, for example, if you comment out the following lines in **CScene::OnRender**:

```
GLfloat matShiny[] = { 120.0f };
glMaterialfv(GL_FRONT, GL_SHININESS, matShiny) ;
```

the objects will be rendered white instead of the desired colors. The GL_SHININESS parameter determines the size and brightness of a specular highlight. The larger the number, the smaller and brighter (more focused) the highlight. A GL_SHININESS value of 0 (the default) results in a large, unfocused highlight that obliterates the color of the object. Therefore, instead of using the default, we set this parameter to a reasonable value. (The maximum value for CL_COLOR_INDEXES is 128.)

See the section "Lighting in Color-Index Mode" in Chapter 6 of the Red Book for more information on lighting.

GL_COLOR_INDEXES

The EasyGI sample application used **glColorMaterial** to color objects. Because **glColorMaterial** does not work in color index mode, EasyCI had to use something different. Therefore, it uses the GL_COLOR_INDEXES parameter of **glMaterial**.

The GL_COLOR_INDEXES parameter is valid only for color index mode; RGBA mode does not use it. The following commands use the GL_COLOR_INDEXES parameter to color the pyramid purple:

```
GLfloat matPurple[] = {16.0f, 47.0f, 79.0f} ;
glMaterialfv(GL_FRONT, GL_COLOR_INDEXES, matPurple) ;
glCallList(Pyramid) ;
```

The **matPurple** array contains three numbers representing the indexes for the ambient, diffuse, and specular material reflectance:

- *Diffuse reflectance* determines the color of an object, so it is usually set to the color of the object.
- *Ambient reflectance* affects the color of an object when light is not directly shining on it (when it is illuminated only by the ambient lights). Ambient reflectance is usually set either to the same color as diffuse reflectance or to a darker color. It makes sense for the ambient reflectance to be darker because the object has less light shining on it.
- *Specular reflectance* determines the color of the object's highlights. A white specular reflectance is the color of the light shining on the object in RGBA mode. In color index mode, lights don't have colors; they only have intensities.

In the code example above, index 16 contains the pure ambient color, index 47 contains the pure diffuse color, and index 79 contains the pure specular color. If OpenGL used only these three indexes to shade objects, the results would be pitiful. Therefore, OpenGL uses colors between these indexes to shade objects. For smooth shading, the colors between the indexes should form a color ramp—that is, the colors should progress from the pure ambient color at index 16 in the example, to the pure diffuse color in index 47, and finally to the pure specular color in index 79.

To understand this, you need to know which colors the palette contains at these indexes and at the indexes in between.

The code below is from **CScene::OnCreatePaletteCI** in EasyCI. It fills up part of the palette with a purple, red, and green color ramp.

```
   // Purple ramp
   for (int i=0; i< 32; i++)
   {
      // 16 to 47 is a linear ramp from black to purple
      pPal->palPalEntry[16+i].peRed  = 255* i / 32 ;
      pPal->palPalEntry[16+i].peGreen= 0 ;
      pPal->palPalEntry[16+i].peBlue = 255* i / 32 ;

      // 48 to 79 is a linear ramp from purple to white
      pPal->palPalEntry[48+i].peRed  = 255 ;
      pPal->palPalEntry[48+i].peGreen= 255* i / 32 ;
      pPal->palPalEntry[48+i].peBlue = 255 ;
   }

   // Red ramp
   for (i=0; i< 32; i++)
   {
      // 80 to 111 is a linear ramp from black to red
      pPal->palPalEntry[80+i].peRed  = 255* i / 32 ;
      pPal->palPalEntry[80+i].peGreen= 0 ;
      pPal->palPalEntry[80+i].peBlue = 0 ;

      // 112 to 143 is a linear ramp from red to white
      pPal->palPalEntry[112+i].peRed  = 255 ;
      pPal->palPalEntry[112+i].peGreen= 255 * i / 32 ;
      pPal->palPalEntry[112+i].peBlue = 255 * i / 32 ;
   }

   // Green ramp
```

```
for (i=0; i< 32; i++)
{
   // 144 to 175 is a linear ramp from black to green
   pPal->palPalEntry[144+i].peRed  = 0;
   pPal->palPalEntry[144+i].peGreen= 255 * i / 32 ;
   pPal->palPalEntry[144+i].peBlue = 0 ;

   // 176 to 207 is a linear ramp from green to white
   pPal->palPalEntry[176+i].peRed  = 255 * i / 32  ;
   pPal->palPalEntry[176+i].peGreen= 255  ;
   pPal->palPalEntry[176+i].peBlue = 255 * i / 32 ;
}
```

Because I am such a nice guy, I created Figure 1 to illustrate what the above code puts into the palette.

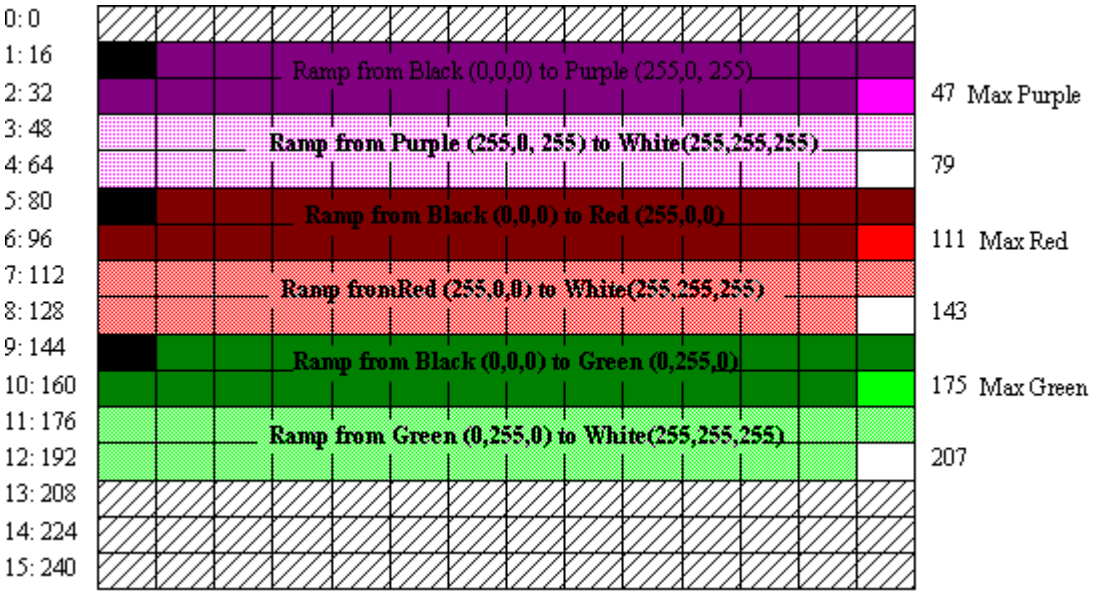

Figure 1. Palette filled in by CScene::OnCreatePaletteCI

Here's the code that makes an object red and green using the above palette:

```
GLfloat matRed[] = {80.0f, 111.0f, 143.0f} ;
glMaterialfv(GL_FRONT, GL_COLOR_INDEXES, matRed) ;
glCallList(Dodec) ;

GLfloat matGreen[] = {144.0f, 175.0f, 207.0f} ;
glMaterialfv(GL_FRONT, GL_COLOR_INDEXES, matGreen) ;
glCallList(Box) ;
```

In the example above, I used black for the pure ambient color, so an object that has only ambient light on it will appear black. The diffuse color of the object is maximum red (255, 0, 0), green (0, 255, 0), or purple (255, 0, 255). An object in a bright light will appear in this (pure red, green, or purple) color. A bright highlight is white (255, 255, 255). I used a smooth ramp to fill the indexes between these three key colors. For more information, see the Red Book, Chapter 6 (pages 192–194) for details. (In particular, see the "Advanced Topic" section, which explains the mathematics.)

Now, that's the code I used. The colors can be anything that you want them to be. The diffuse color can be anything that your heart desires. The ambient reflectance could be dark blue, dark green, dark purple, orange, or even white. The same is true for the specular reflectance. The color ramps can be any length. You can have the diffuse and ambient colors share the same index and have a 5-color ramp to the highlight color, if you desire. It's up to you—experiment to your heart's content.

Changes to EasyCI

To get EasyCI to use the color index mode instead of RGBA mode, I changed the classes in GLlib and some of the OpenGL code in EasyCI. Figure 2 illustrates the changes.

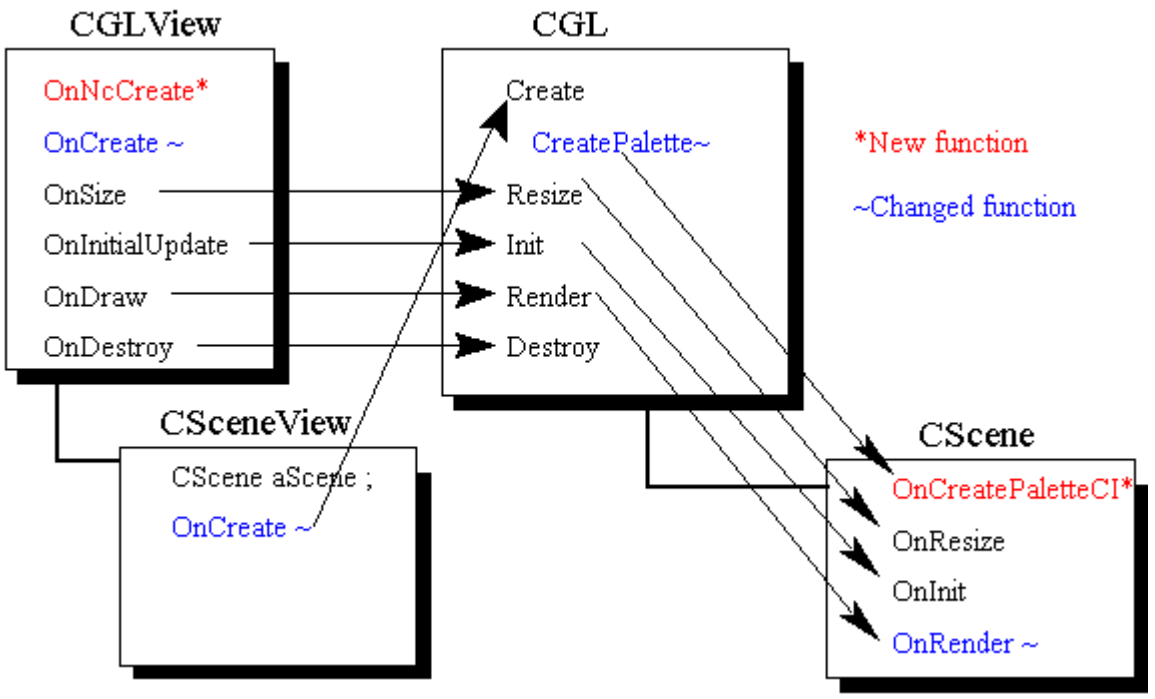

Figure 2. Changes and additions to GLlib and EasyCI

Changes to GLlib

Inside CGlib, I changed both **CGL** and **CGLView** to support color index mode. I made most of the Windows NT OpenGL implementation changes in **CGL**. I modified the EasyCI class **CScene** to meet OpenGL requirements for color index mode. **CScene::OnRender** includes the bulk of the OpenGL code and most of the OpenGL changes for color indexing.

First, we will examine the changes in **CGL**.

CGL Changes

In **CGL**, I changed both **CGL::Create** and **CGL::CreatePalette** and exported a new function, **CGL::OnCreatePaletteCI**, as shown in Figure 3.

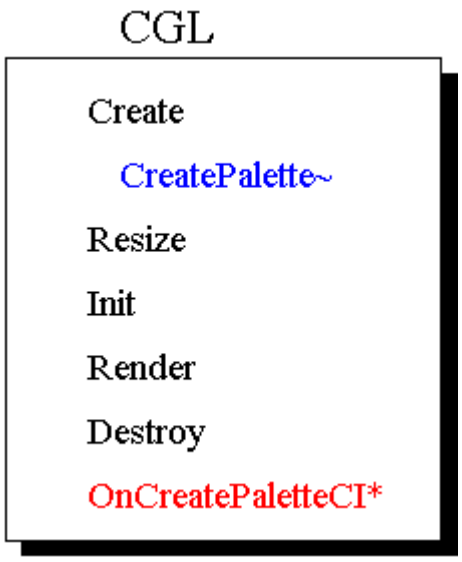

Figure 3. Changes to CGL

CGL::Create

The major change in **CGL::Create** is the addition of a new parameter, **iPixelType**, which is declared in CGL.H as follows:

```
   BOOL Create(CWnd* pWnd, int iPixelType = PFD_TYPE_RGBA) ;
```

iPixelType has a default value, so EasyGL didn't require any changes. In **CGL::Create**, I changed the following line:

```
pfd.iPixelType = PFD_TYPE_RGBA ;
```

to:

```
pfd.iPixelType = iPixelType ;
```

In EasyCI, **CSceneView::OnCreate** calls **CGL::Create** and passes it PFD_TYPE_COLORINDEX to set OpenGL to color index mode.

CGL::CreatePalette

Color index mode requires a palette. The whole point of color index mode is having a palette so you can specify palette indexes instead of palette colors. However, PFD_NEED_PALETTE is not set for color index mode. The previous version of **CGL::CreatePalette** included the line:

```
if (!(pfd.dwFlags & PFD_NEED_PALETTE)) return FALSE;
```

This means that a palette won't be created unless PFD_NEED_PALETTE is set. I replaced this line with the following code:

```
BOOL bColorIndex = (pfd.iPixelType & PFD_TYPE_COLORINDEX) ;
if ((pfd.dwFlags & PFD_NEED_PALETTE) || bColorIndex)
{
   // Need to create a palette
   LOGPALETTE* pPal = (LOGPALETTE*)malloc(sizeof(LOGPALETTE)
                      + 256 * sizeof(PALETTEENTRY));
   pPal->palVersion = 0x300 ;
   pPal->palNumEntries = 256 ;
   if (bColorIndex)
   {
      // Let user create palette for color index mode
      OnCreatePaletteCI(pPal) ;
   }
   else
   {
      // Create RGB palette for RGBA mode
      .
      .
      .
   }
}
```

With these changes, **CreatePalette** will create a palette for color index mode in addition to creating a palette when PFD_NEED_PALETTE is set. However, it is the application's job to set the palette to whatever it wants. Therefore, **CreatePalette** calls **CGL::OnCreatePaletteCI**.

CGL::OnCreatePaletteCI

CGL::OnCreatePaletteCI is declared and defined as follows:

```
virtual BOOL OnCreatePaletteCI(LOGPALETTE* pPal)
        { TRACE0("CGL::OnCreatePaletteCI\r\n"); return FALSE; }
```

CGL::OnCreatePaletteCI is implemented by classes derived from **CGL**, in a manner similar to **CGL::OnResize**, **CGL::OnInit**, and **CGL::OnRender**. **CGL::OnCreatePaletteCI** differs from these functions in that it is not a pure virtual function. I didn't make **OnCreatePaletteCI** a pure virtual function because only derived classes using color index mode need to implement it.

See the sections on GL_COLOR_INDEXES (earlier in this article) and **CScene::OnCreatePaletteCI** (in "EasyCI Changes," later in this article) for instructions on setting up the palette.

CGLView Changes

In **CGLView**, I modified **CGLView::OnCreate** and added **CGLView::OnNcCreate**, as illustrated in Figure 4.

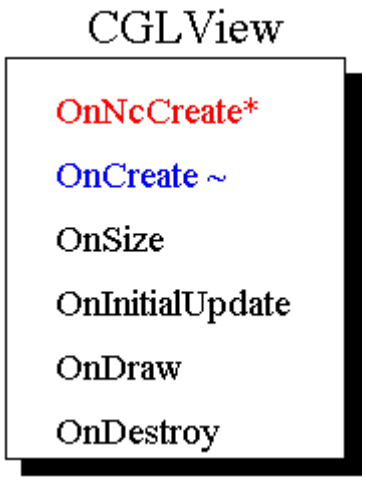

Figure 4. Changes to CGLView

I moved the following lines from **CGLView::OnCreate** to **CGLView::OnNcCreate**:

```
pGL = GetGLptr() ;
ASSERT(pGL) ;
```

I moved these lines to simplify using GLlib with color index mode. To use color index mode with GLlib, you must pass PFD_TYPE_COLORINDEX to **CGL::Create**. **CGLView::OnCreate** calls **CGL::Create** with the PFD_TYPE_RGBA parameter. We override **CGLView::OnCreate** to pass PFD_TYPE_COLORINDEX to **CGL::Create** in our **CGLView** derived class. Moving the above lines to **CGLView::OnNcCreate** means that we have less to remember when implementing the overridden version of **CGLView::OnCreate**.

For more information on the overridden version of **CGLView::Create**, see the explanation of **CSceneView::Create** in the "CSceneView Changes" section later in this article.

Changes to EasyCI

In creating EasyCI from EasyGL, I had to change functions in both the **CScene** class and the **CSceneView** class.

CScene Changes

I changed **CScene::OnRender** and added **CScene::OnCreatePaletteCI**, as shown in Figure 5.

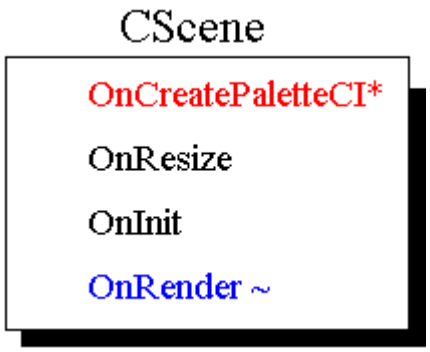

Figure 5. Changes to CScene

CScene::OnRender

To make it easier to detect the differences between the EasyGL and EasyCI versions of **CScene::OnRender**, I separated the new code from the old code with an **#if** / **#else** / **#endif** block in EasyCI:

```
#if COLORINDEX
    glMaterialFv(GL_FRONT, GL_COLOR_INDEXES, matPurple) ;
#else
    glColor3dv(purple);
#endif
```

The EasyCI version of **CScene::OnRender** differs from the EasyGL version in the following areas:

- I changed **glClearColor** to **glClearIndex**.
- I removed **glColorMaterial**. (It doesn't work in color index mode, so I replaced it with **glMaterialfv**, as described below.)
- I set the material's shininess with **glMaterialfv**, using the GL_SHININESS parameter.
- Instead of using **glColor**, I used **glMaterialfv** with the GL_COLOR_INDEXES parameter to set the color.

Refer to the "GL_SHININESS" and "GL_COLOR_INDEXES" sections earlier in this article for more information on these parameters.

CScene::OnCreatePaletteCI

The **CScene::OnCreatePaletteCI** virtual function is overridden by **CScene** so that it can create any palette it wants. The virtual function happens to create a purple ramp, a green ramp, and a red ramp. See the "GL_COLOR_INDEXES" section earlier in this article for more information.

CSceneView Changes

I implemented **CSceneView::OnCreate**, as shown in Figure 6, to pass PFD_TYPE_COLORINDEX to **CGL::Create**:

```
int CSceneView::OnCreate(LPCREATESTRUCT lpcs)
{
   if (CView::OnCreate(lpcs) == -1)
       return -1 ;

   BOOL bResult = pGL->Create(this, PFD_TYPE_COLORINDEX) ;
   if (bResult)
      return 0 ;
   else
      return -1 ;
}
```

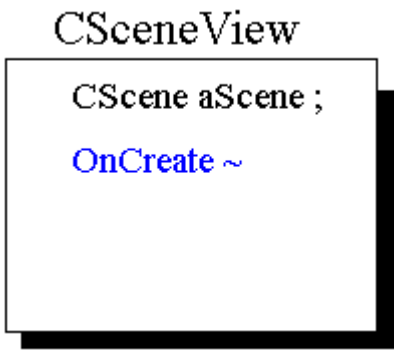

Figure 6. Change to CSceneView

Notice that we call **CView::OnCreate**, not **CGLView::OnCreate**.

Conclusion

Using color index mode is not more difficult than using RGBA mode, but it is slightly different. OpenGL code that uses RGBA mode can be difficult to convert to color index mode. This is especially true if the code uses many material properties, because color index mode doesn't support most of the material properties, and the properties that it does support are set differently. Converting an application from color index mode to RGBA can also be difficult.

Choosing color index mode instead of RGBA mode is a simple matter of changing one field in the pixel format descriptor. The application will also need to set up a palette that meets its specific requirements.

Bibliography

Sources of Information on OpenGL

Crain, Dennis. "Windows NT OpenGL: Getting Started." April 1994. (MSDN Library, Technical Articles)

Neider, Jackie, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1*. Reading, MA: Addison-Wesley, 1993. ISBN 0-201-63274-8. (This book is also known as the "Red Book".)

OpenGL Architecture Review Board. *OpenGL Reference Manual: The Official Reference Document for OpenGL, Release 1*. Reading, MA: Addison-Wesley, 1992. ISBN 0-201-63276-4. (This book is also known as the "Blue Book".)

Prosise, Jeff. "Advanced 3-D Graphics for Windows NT 3.5: Introducing the OpenGL Interface, Part I." *Microsoft Systems Journal* 9 (October 1994). (MSDN Library Archive Edition, Books and Periodicals)

Prosise, Jeff. "Advanced 3-D Graphics for Windows NT 3.5: The OpenGL Interface, Part II." *Microsoft Systems Journal* 9 (November 1994). (MSDN Library Archive Edition, Books and Periodicals)

Prosise, Jeff. "Understanding Modelview Transformations in OpenGL for Windows NT." *Microsoft Systems Journal* 10 (February 1995).

Rogerson, Dale. "OpenGL I: Quick Start.". December 1994. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL II: Windows Palettes in RGBA Mode." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL III: Building an OpenGL C++ Class." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL V: Translating Windows DIBs." February 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL VI: Rendering on DIBs with PFD_DRAW_TO_BITMAP." April 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL VII: Scratching the Surface of Texture Mapping." May 1995. (MSDN Library, Technical Articles)

Microsoft Win32 Software Development Kit (SDK) for Windows NT 3.5 *OpenGL Programmer's Reference*.

Sources of Information on Palettes

Gery, Ron. "Palette Awareness." April 1992. (MSDN Library, Technical Articles)

Gery, Ron. "The Palette Manager: How and Why." March 1992. (MSDN Library, Technical Articles)

Gery, Ron. "Using DIBs with Palettes." March 1992. (MSDN Library, Technical Articles)

Rodent, Herman. "Animation in Win32." February 1994. (MSDN Library, Technical Articles)

Rodent, Herman. "Animation in Windows." April 1993. (MSDN Library, Technical Articles)

Thompson, Nigel. *Animation Techniques for Win32*. Redmond, WA: Microsoft Press, 1995.