Buffers

Objectives
- Introduce additional OpenGL buffers
- Learn to read and write buffers
- Learn to use blending

Buffer
Define a buffer by its spatial resolution \((n \times m)\) and its depth \(k\), the number of bits/pixel

OpenGL Frame Buffer

OpenGL Buffers
- Color buffers can be displayed
  - Front
  - Back
  - Auxiliary
  - Overlay
- Depth
- Accumulation
  - High resolution buffer
- Stencil
  - Holds masks
Writing in Buffers

- Conceptually, we can consider all of memory as a large two-dimensional array of pixels
- We read and write rectangular block of pixels
  - Bit block transfer (bitblt) operations
- The frame buffer is part of this memory

Writing Model

Read destination pixel before writing source

Writing Modes

- Source and destination bits are combined bitwise
- 16 possible functions (one per column in table)

XOR mode

- Recall from Chapter 3 that we can use XOR by enabling logic operations and selecting the XOR write mode
- XOR is especially useful for swapping blocks of memory such as menus that are stored off screen
  
  If S represents screen and M represents a menu
  
  the sequence

  \[ S \leftarrow S \oplus M \]

  \[ M \leftarrow S \oplus M \]

  \[ S \leftarrow S \oplus M \]

  swaps the S and M
The Pixel Pipeline

- OpenGL has a separate pipeline for pixels
  - Writing pixels involves
    - Moving pixels from processor memory to the frame buffer
    - Format conversions
    - Mapping, Lookups, Tests
  - Reading pixels
    - Format conversion

Buffer Selection

- OpenGL can draw into or read from any of the color buffers (front, back, auxiliary)
- Default to the back buffer
- Change with `glDrawBuffer` and `glReadBuffer`
- Note that format of the pixels in the frame buffer is different from that of processor memory and these two types of memory reside in different places
  - Need packing and unpacking
  - Drawing and reading can be slow

Raster Position

- OpenGL maintains a raster position as part of the state
- Set by `glRasterPos*()`
  - `glRasterPos3f(x, y, z);`
- The raster position is a geometric entity
  - Passes through geometric pipeline
  - Eventually yields a 2D position in screen coordinates
  - This position in the frame buffer is where the next raster primitive is drawn

Bitmaps

- OpenGL treats 1-bit pixels (bitmaps) differently than multi-bit pixels (pixelmaps)
- Bitmaps are masks which determine if the corresponding pixel in the frame buffer is drawn with the present raster color
  - 0 ⇒ color unchanged
  - 1 ⇒ color changed based on writing mode
- Bitmaps are useful for raster text
  - `GLUT_BIT_MAP_8_BY_13`
**Raster Color**

- Same as drawing color set by `glColor*()`
- Fixed by last call to `glRasterPos*()`

```c
    glColor3f(1.0, 0.0, 0.0);
    glRasterPos3f(x, y, z);
    glColor3f(0.0, 0.0, 1.0);
    glBitmap(......
    glBegin(GL_LINES);
    glVertex3f(.....)
```

- Geometry drawn in blue
- Ones in bitmap use a drawing color of red

---

**Drawing Bitmaps**

`glBitmap(width, height, x0, y0, xi, yi, bitmap)`

- Offset from raster position
- Increments in raster position after bitmap drawn

**Example: Checker Board**

```c
    GLubyte wb[2] = {0 x 00, 0 x ff};
    GLubyte check[512];
    int i, j;
    for(i=0; i<64; i++) for (j=0; j<64, j++)
        check[i*8+] = wb[(i/8+j)%2];
    glBitmap( 64, 64, 0.0, 0.0, 0.0, 0.0, check);
```
Pixel Maps

• OpenGL works with rectangular arrays of pixels called pixel maps or images
• Pixels are in one byte (8 bit) chunks
  - Luminance (gray scale) images 1 byte/pixel
  - RGB 3 bytes/pixel
• Three functions
  - Draw pixels: processor memory to frame buffer
  - Read pixels: frame buffer to processor memory
  - Copy pixels: frame buffer to frame buffer

OpenGL Pixel Functions

```c
#include <GL/gl.h>

GLubyte myimage[512][512][3];

// Draw pixels
glDrawPixels(0, 0, 512, 512, GL_RGB,
             GL_UNSIGNED_BYTE, myimage);

// Read pixels
glReadPixels(x, y, width, height, format, type,
             myimage);
```

Image Formats

• We often work with images in a standard format (JPEG, TIFF, GIF)
• How do we read/write such images with OpenGL?
• No support in OpenGL
  - OpenGL knows nothing of image formats
  - Some code available on Web
  - Can write readers/writers for some simple formats in OpenGL

Displaying a PPM Image

• PPM is a very simple format
• Each image file consists of a header followed by all the pixel data
• Header

```
P6
# comment 1
# comment 2
...
#comment n
rows columns maxvalue
pixels
```
Reading the Header

FILE *fd;
int k, nm;
char c;
int i;
char b[100];
float s;
int red, green, blue;
printf("enter file name\n");
scanf("%s", b);
fd = fopen(b, "r");
scanf(fd,"%[^
]",b);
if(b[0] != 'P' || b[1] != '6') {
  printf("%s is not a PPM file!\n", b);
  exit(0);
}
printf("%s is a PPM file\n",b);

Reading the Header (cont)

fscanf(fd, "%c", &c);
while(c == '\#') {
  fscanf(fd, "%[^
]", b);
  printf("%s\n",b);
  fscanf(fd, "%c", &c);
}
ungetc(c,fd);

skip over comments by
looking for # in first column

Reading the Data

fscanf(fd, "%d %d %d", &n, &m, &k);
printf("%d rows  %d columns  max value= %d\n",n,m,k);

nm = n*m;
image=malloc(3*sizeof(GLuint)*nm);
s=255./k;
scale factor
for(i=0;i<nm;i++) {
  fscanf(fd,"%d %d %d",&red, &green, &blue );
  image[3*nm-3*i-3]=red;
  image[3*nm-3*i-2]=green;
  image[3*nm-3*i-1]=blue;
}

Scaling the Image Data

We can scale the image in the pipeline

glPixelTransferf(GL_RED_SCALE, s);
glPixelTransferf(GL_GREEN_SCALE, s);
glPixelTransferf(GL_BLUE_SCALE, s);

We may have to swap bytes when we go from
processor memory to the frame buffer depending on
the processor. If so we can use

glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
The display callback

```c
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRasterPos2i(0,0);
    glDrawPixels(n,m,GL_RGB,
                GL_UNSIGNED_INT, image);
    glFlush();
}
```