Buffers

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico
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 (or precision) \(k\), 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

![Diagram showing writing into frame buffer from source memory]
Writing Model

Read destination pixel before writing source
Bit Writing Modes

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

<table>
<thead>
<tr>
<th>s</th>
<th>d</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

replace

XOR

OR
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 ← S ⊕ M
    M ← S ⊕ M
    S ← S ⊕ 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
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
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
Bitmaps

• OpenGL treats 1-bit pixels (*bitmaps*) differently from multi-bit pixels (*pixelmaps*)

• Bitmaps are masks that 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 font: *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

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

- `width`: width of the bitmap
- `height`: height of the bitmap
- `x0, y0`: first raster position
- `xi, yi`: offset from raster position
- `bitmap`: the bitmap to be drawn

- **offset from raster position**

- **increments in raster position**

- **first raster position**

- **second raster position**

- **second raster position after bitmap drawn**
GLubyte \texttt{wb}[2] = \{0 x 00, 0 x ff\};
GLubyte \texttt{check}[512];
int \texttt{i, j};
for(\texttt{i=0; i<64; i++}) for (\texttt{j=0; j<64, j++})
\texttt{check[i*8+j]} = \texttt{wb[(i/8+j)\%2]};

\texttt{glBitmap( 64, 64, 0.0, 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

\[ \text{glReadPixels}(x, y, \text{width}, \text{height}, \text{format}, \text{type}, \text{myimage}) \]

\text{start pixel in frame buffer} \quad \text{size} \quad \text{type of pixels} \quad \text{type of image} \quad \text{pointer to processor memory}

\begin{verbatim}
GLubyte myimage[512][512][3];
glReadPixels(0,0, 512, 512, GL_RGB, GL_UNSIGNED_BYTE, myimage);
\end{verbatim}

\[ \text{glDrawPixels}(\text{width}, \text{height}, \text{format}, \text{type}, \text{myimage}) \]

\text{starts at raster position}
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

    P3
    # comment 1
    # comment 2
    .
    #comment n
    rows columns maxvalue
    pixels
Reading the Header

```c
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");
fscanf(fd,"%[^\n] ",b);
if(b[0]!='P'|| b[1] != '3'){
    printf("%s is not a PPM file!\n", b);
    exit(0);
}
printf("%s is a PPM file\n",b);
```

check for “P3” in first line
fscanf(fd, "%c", &c);
while(c == '#')
{
    fscanf(fd, "^[^\n] ", b);
    printf("%s\n", b);
    fscanf(fd, "%c", &c);
}
ungetc(c, fd);

skip over comments by looking for # in first column
Reading the Data

```c
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

```c
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

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