Computer Graphics Framebuffers

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

The Framebuffers

Video Driver

Update

Front

Back

Depth-Buffer

Double-buffered Color Framebuffers

Refresh

The Video Driver

The viewer sees the contents of the front framebuffer.

You draw into here

Front

The monitor displays from here

"swap buffers" changes the role of the two framebuffers

This is called the refresh

You draw into here

Back

The monitor displays from here

This is called the update

// swap the double-buffered framebuffers:

glutSwapBuffers();

glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );

glDrawBuffer( GL_BACK );

The monitor displays from here

You draw into here

Front

Back
The Video Driver

• **N refreshes/second** (N is between 50 and 120, 60 is common)

• The framebuffer contains the R,G,B that define the color at each pixel

• Because of the double-buffering, **Refresh** is asynchronous from **Update**, that is, the monitor gets refreshed at N (60) frames per second, no matter how fast or slowly you update the back buffer.

The Framebuffer Uses RGB Colors

- Red
- Yellow
- Green
- Magenta
- White
- Cyan
- Blue

The Framebuffer: Integer Color Storage

<table>
<thead>
<tr>
<th># Bits/color</th>
<th># Intensities per color</th>
<th>Total colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$2^8 = 256$</td>
<td>Typical</td>
</tr>
<tr>
<td>10</td>
<td>$2^{10} = 1024$</td>
<td>High Dynamic Range (HDR)</td>
</tr>
<tr>
<td>12</td>
<td>$2^{12} = 4096$</td>
<td></td>
</tr>
</tbody>
</table>

# Bits/pixel: Total colors:

- 24: $2^{24} = 16.7$ M
- 30: $2^{30} = 1$ B
- 36: $2^{26} = 69$ B

The Framebuffer: Floating Point Color Storage

• 16- or 32-bit floating point for each color component

Why so many bits?

Many modern algorithms do arithmetic on the framebuffer color components or treat the framebuffer color components as data. They need the extra precision during the arithmetic. However, the display system cannot produce all of those possible colors.
**Alpha values**

- Transparency per pixel
  - $\alpha = 0$. is invisible
  - $\alpha = 1$. is opaque
- Represented in 8-32 bits (integer or floating point)
- Alpha blending equation:

$$\text{Color} = \alpha C_1 + (1 - \alpha) C_2$$

$0.0 \leq \alpha \leq 1.0$

Note: this is really **blending**, not transparency!

---

**Z-buffer or Depth-Buffer**

- Used for hidden surface removal
- Holds the pixel’s depth in the 3D scene
- Typically is 32 bits
- Can be integer or floating point

Why do things in front look like they are really in front?

Your application code might draw this cube’s polygons in 1-2-3-4-5-6 order, but 1, 3, and 4 still need to look like they were drawn last:

**Solution #1**: Sort your polygons in 3D by depth and draw them back-to-front.

In this case 1-2-3-4-5-6 becomes 5-6-2-4-1-3.

This is called the **Painter’s Algorithm**. Once upon a time, we had to do things this way. It sucked even more than it sounds.

---

Why do things in front look like they are really in front?

Your application might draw this cube’s polygons in 1-2-3-4-5-6 order, but 1, 3, and 4 still need to look like they were drawn last:

**Solution #2**: Add an extension to the framebuffer to store the depth of each pixel. This is called a **Depth-buffer or Z-buffer**. Only allow pixel stores to take place when the depth of the incoming pixel is closer to the viewer than the pixel that is already there.
Incoming RGBZ from the application

Existing RGBZ in the framebuffer

Compare

Z_{incoming} closer to the viewer than Z_{existing}?

Yes

Allow RGBZ_{incoming} to overwrite RGBZ_{existing}

No

Do nothing

Why do things in front look like they are really in front?

With Depth Buffer

Without Depth Buffer