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

Double-buffered Color Framebuffers

Depth-Buffer

Update

Refresh

Front

Back

Video Driver
glutSwapBuffers()  

// swap the double-buffered framebuffers:  

`glutSwapBuffers();`

`glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);`

`glDrawBuffer(GL_BACK);`

---

You draw into here

This is called the **update**

"swap buffers" changes the role of the two framebuffers

The monitor displays from here

This is called the **refresh**

You draw into here

The monitor displays from here
The Video Driver

The viewer sees the contents of the **front framebuffer**
The Video Driver

• N *refreshes/second* (typically, N is between 50 and 100)

• 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
The Framebuffer: Integer Color Storage

# Bits/color  # Intensities per color
8  \(2^8 = 256\)  “Typical”
10 \(2^{10} = 1024\)  High Dynamic Range (HDR)
12 \(2^{12} = 4096\)

# Bits/pixel  Total colors:
24 \(2^{24} = 16.7\) M
30 \(2^{30} = 1\) B
36 \(2^{36} = 69\) 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

# Bits / Z: 32
Total Z Values: $2^{32} = 4B$
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 #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. It sucked to have to do things this way.
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 when the depth of the incoming pixel is closer to the viewer than the pixel that is already there.
Depth Buffer

**Incoming RGBZ**

- **Z**
- **B**
- **G**
- **R**

**Existing RGBZ**

- **Z**
- **B**
- **G**
- **R**

**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