Computer Graphics Framebuffers

Mike Bailey
mjb@cs.oregonstate.edu

The Framebuffers

Double-buffered Color Framebuffers

Front

Back

Depth-Buffer

Update

Refresh

Video Driver
glutSwapBuffers( )

// swap the double-buffered framebuffers:

```c
glutSwapBuffers();
```

```c
glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
```

```c
glDrawBuffer( GL_BACK );
```

You draw into here

This is called the **update**

The video driver contains two framebuffers:

- **Front Framebuffer**
- **Back Framebuffer**

You draw into here

The monitor displays from here

“swap buffers” changes the role of the two framebuffers

This is called the **refresh**

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

The **Video Driver** contains the following components:

- **Double-buffered Color Framebuffers**
- **Depth-Buffer**

The video driver displays content from the **Front Framebuffer**.
The Video Driver

- N \textit{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, \textit{Refresh} is asynchronous from \textit{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

#### # Bits/color # Intensities per color

- 8 bits/color: $2^8 = 256$ ("Typical")
- 10 bits/color: $2^{10} = 1024$
- 12 bits/color: $2^{12} = 4096$

#### # Bits/pixel Total colors:

- 24 bits/pixel: $2^{24} = 16.7$ M colors
- 30 bits/pixel: $2^{30} = 1$ B colors
- 36 bits/pixel: $2^{36} = 69$ B colors

### 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  Total Z Values:
  32  \( 2^{32} = 4B \)
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.

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

Depth Buffer

Compare

Existing RGBZ in the framebuffer

Z\text{incoming} closer to the viewer than Z\text{existing}?

Yes

Allow RGBZ\text{incoming} to overwrite RGBZ\text{existing}

No

Do nothing

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

With Depth Buffer

Text That Moves

Text That Doesn’t

Without Depth Buffer

Text That Moves

Text That Doesn’t