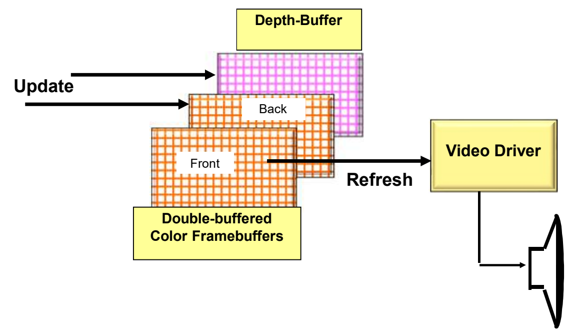
Computer Graphics Framebuffers

Mike Bailey

mjb@cs.oregonstate.edu

The Framebuffers

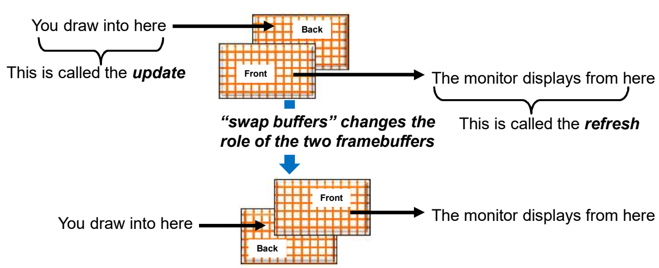


glutSwapBuffers( )

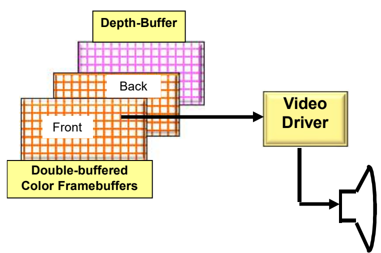
```
// swap the double-buffered framebuffers:

glutSwapBuffers( );
```

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

```
glDrawBuffer( GL_BACK );
```



The Video Driver

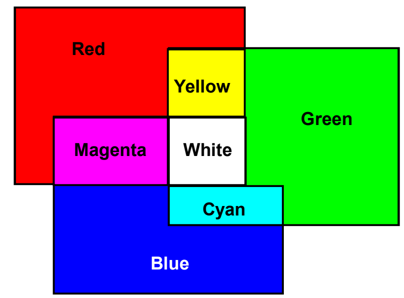


The viewer sees the contents of the **front framebuffer**

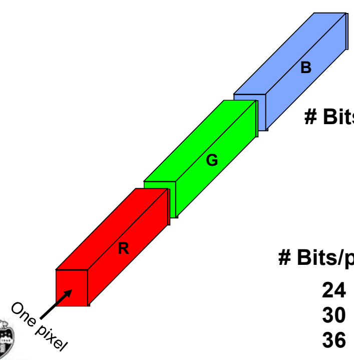
The Video Driver

- 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



The Framebuffer: Integer Color Storage

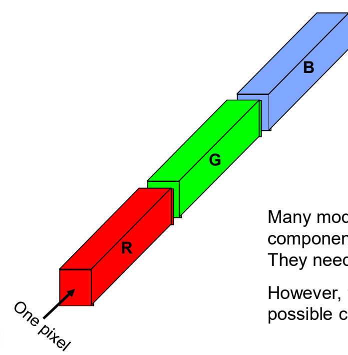


# Bits/color	# Intensities per color
8	2^8 = 256 ← "Typical"
10	2^{10} = 1024 ← High Dynamic Range (HDR)
12	2^{12} = 4096 ← High Dynamic Range (HDR)

# 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.

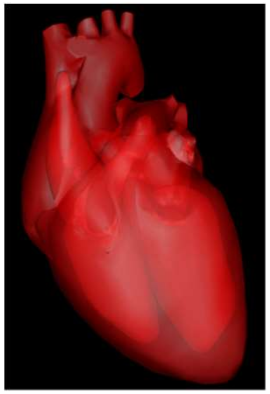
The Framebuffer

- ***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:

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

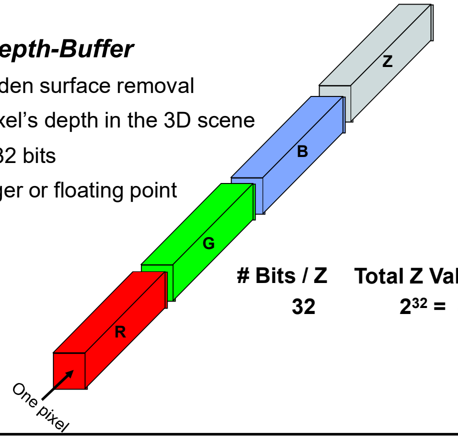
0.0 ≤ α ≤ 1.0

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



The Framebuffer

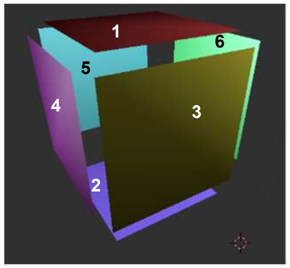
- ***Z-buffer or Depth-Buffer***
  - Used for hidden surface removal
  - Holds the pixel's depth in the 3D scene
  - Typically is 32 bits
  - Can be integer or floating point



# 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:



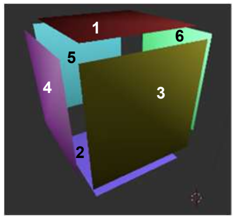
Solution #1: Sort your polygons in 3D by depth and draw them back-to-front.

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

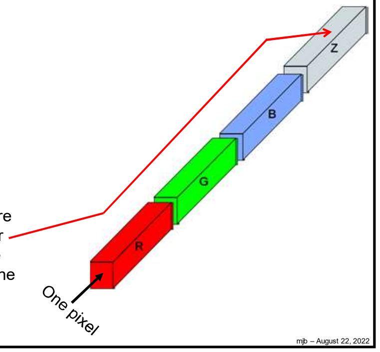
This is called the **Painter's Algorithm**. Once upon a time, we had to do things this way. It sucked even more than it sounds.

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

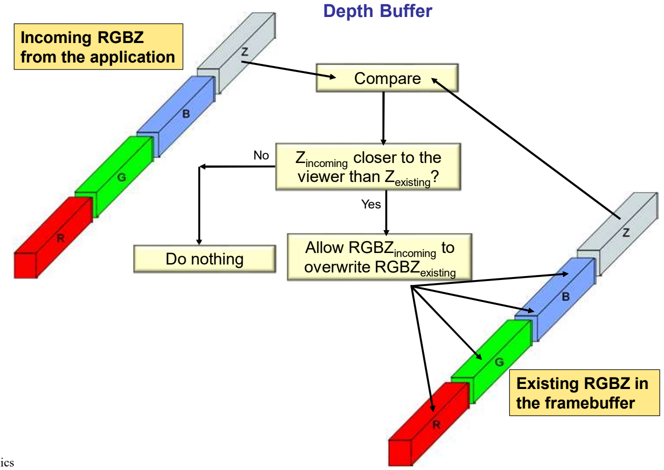
Your application might draw this cube's polygons in 1-2-3-4-5-6 order, but 1, 3, and 4 still need to look like they were drawn last:



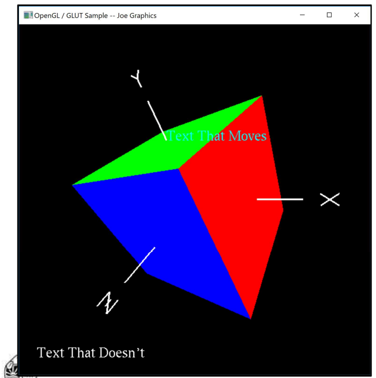
Solution #2: Add an extension to the framebuffer to store the depth of each pixel. This is called a **Depth-buffer** or **Z-buffer.** Only allow pixel stores to take place when the depth of the incoming pixel is closer to the viewer than the pixel that is already there.



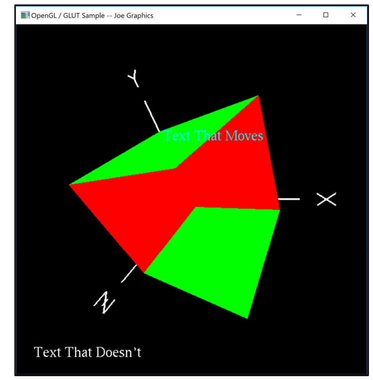
Depth Buffer



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



With Depth Buffer



Without Depth Buffer