The Framebuffers

- Video Driver
- Double-buffered Color Framebuffers
- Depth-Buffer
- Update
- Refresh

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

- Red
- Yellow
- Green
- Blue
- Magenta
- White
- Cyan
### The Framebuffer: Integer Color Storage

**Number of bits/color and number of intensities per color**

<table>
<thead>
<tr>
<th>Bits/color</th>
<th># Intensities per color</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$2^8 = 256$</td>
</tr>
<tr>
<td>10</td>
<td>$2^{10} = 1024$</td>
</tr>
<tr>
<td>12</td>
<td>$2^{12} = 4096$</td>
</tr>
</tbody>
</table>

**Number of bits/pixel and total colors**

<table>
<thead>
<tr>
<th>Bits/pixel</th>
<th>Total Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>$2^{24} = 16.7$ M</td>
</tr>
<tr>
<td>30</td>
<td>$2^{30} = 1$ B</td>
</tr>
<tr>
<td>36</td>
<td>$2^{36} = 69$ B</td>
</tr>
</tbody>
</table>

### The Framebuffer: Floating Point Color Storage

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

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

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

### 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.

**Solution #2**: Add an extension to the framebuffer to store the depth of each pixel. This is called the Z-buffer or Depth-buffer. Only allow pixel stores when the depth of the incoming pixel is closer to the viewer than the pixel that is already there.
Compare $Z_{\text{incoming}}$ closer to the viewer than $Z_{\text{existing}}$?

Yes
No

Allow RGBZ$_{\text{incoming}}$ to overwrite RGBZ$_{\text{existing}}$?

Do nothing

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

With Depth Buffer
Without Depth Buffer