### Alpha Channel

“Color” traditionally consists of R, G, & B. Can also specify a fourth value: \textit{opacity}

- Measure of “opaqueness” (opposite of “transparency”)
- Stored in the “alpha channel”
  - \( \text{Alpha} = 0 \) means “no opacity” (completely transparent)
  - \( \text{Alpha} = 1 \) means “fully opaque” (no transparency)

```
R  G  B  A
8  8  8  8
```

"32-bit color"

### Blending (Compositing)

Writing new pixels to the frame buffer:
- "Source" – the new pixel
- "Destination" – the pixel already in the buffer

Source & Destination colors can be \textit{blended}

- Many different “blending equations” can be used
  - \textit{add}, \textit{subtract}, \textit{min}, \textit{max}, ...
- Equations can apply different “scale factors” to source & dest

### Blending Control

```c
void glBlendEquation(GLenum mode);
void glBlendFunc(GLenum srcFactor, GLenum destFactor);
```
Compositing Example 1

```
glBlendFunc(GL_ONE, GL_ZERO)
```
- Source pixel RGBA scaled by [1 1 1 1]
- Destination pixel RGBA scaled by [0 0 0 0]
- Pixels added together (default `glBlendEquation()`) 
  - Result?
    - Framebuffer = source color (the default)

*note: assumes `glBlendEquation(GL_FUNC_ADD)`*

Compositing Example 2

```
glBlendFunc(GL_ONE, GL_ONE)
```
- Source pixel scaled by [1 1 1 1]
- Destination pixel scaled by [1 1 1 1]
- Pixels added together (default `glBlendEquation()`) 
  - Suppose:
    - Src = [0, 255, 0, 128] //semi-transp. green
    - Dest = [255, 0, 0, 128] //semi-transp. red
  - Result?
    - Framebuffer = [255, 255, 0, 255] //solid yellow!

Compositing Example 3

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```
- Source pixel scaled by its alpha value (opacity)
- Destination pixel scaled by (1 - srcAlpha) [source transparency]
- Pixels added together (default `glBlendEquation()`) 
  - Suppose:
    - Src = [255, 0, 0, 192] // red, 75% opaque
    - Dest = [0, 255, 0, 255] // green, completely opaque
  - Result?
    - `srcPixel*srcAlpha` = [ (255*0.75), (0*0.75), (0*0.75), (0.75*0.75) ] = [192, 0, 0, 144] // partially unsaturated red
    - `destPixel*(1-srcAlpha)` = [ (0*0.25), (255*0.25), (0*0.25), (255*0.25) ] = [0, 64, 0, 64] // heavily unsaturated green
    - Framebuffer = [192, 64, 0, 208] //red with some green

Drawing Order

- HSR (normally) completely determines “visibility”
- With blending, more than one thing is visible
- Output order matters when blending
- Reason: determines what is “src” & “dest”
  - Reversing these produces different results
- Solution approaches:
  - Separate object lists (opaque & transparent)
    - Draw all opaques 1st, then transparents
  - Depth-sort all objects, draw from back to front ("Painter’s Algorithm")
    - Slow; doesn’t deal with intersecting objects

Multi-Pass compositing

can fill in back faces:

http://www.alecjacobson.com/weblog/?p=2750