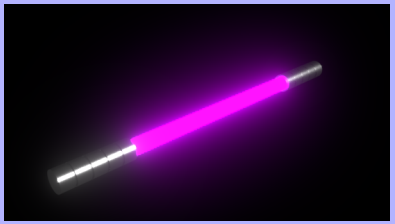
Alpha

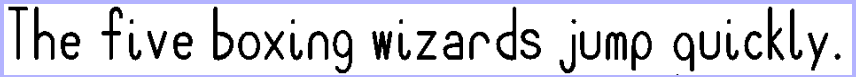
Motivation

- Thus far, we've always drawn some color without regard to what's already on the screen
- But many interesting effects require that we *blend* a new color with one that's already on the screen



Problem

- This is especially important when rendering text
- Example: Consider this text:



Zoom

- Consider a zoomed-in view
- Notice grey along the boundaries. Very important to making smooth appearance

Blending

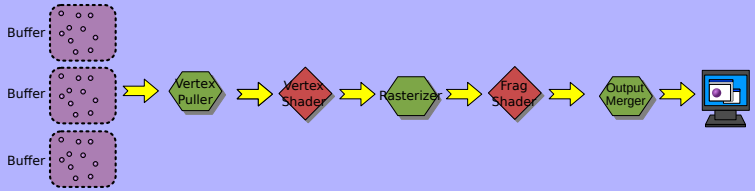
- We need to blend pixels with the background
- GPU has facility to do this: *alpha blending*

Definition

- Alpha = Transparency
  - 0 = Perfectly transparent
  - 1 = Totally opaque
- Blending = process of combining new fragments with what's already on screen
- Source: Incoming fragment
- Destination: Existing fragment

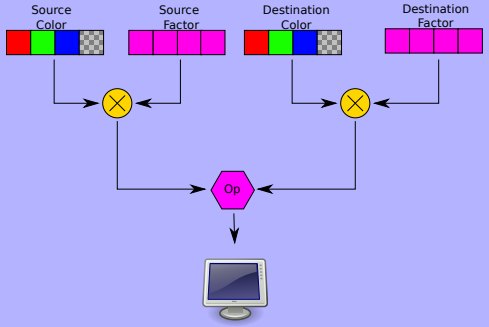
Pipeline

- Output merger stage: Handles blending



Blending

- Colors have four components: r,g,b,α
- Usually, we use α to control blending
- *How* alpha does this is configurable
- We have a few knobs to tweak:
  - Source factor
  - Destination factor
  - Combination Op



Factor

- Some of the more common choices for the factors:

Factor	Input
GL_ZERO	0
GL_ONE	1
GL_SRC_ALPHA	source alpha
GL_ONE_MINUS_SRC_ALPHA	1-source alpha

Op

- Our choices for the op:

Op	Action
GL_FUNC_ADD	src + dest
GL_FUNC_SUBTRACT	src - dest
GL_FUNC_REVERSE_SUBTRACT	dest - src
GL_MIN	min(src,dest)
GL_MAX	max(src,dest)

- We nearly always use GL_FUNC_ADD (the default)

API

- glEnable(GL_BLEND)
- glDisable(GL_BLEND)
- glBlendFunc(source factor, dest factor)
- glBlendEquation(op)
  - We rarely need this one

Application

- How do we use?
  - Hardware doesn't care about meaning
  - Just does whatever math we tell it
- So *we* attach meaning to alpha
- Question: How do we define alpha?
- Two methods
  - Conventional
  - Premultiplied

Conventional α

- Philosophy: Alpha tells transparency; RGB tells color
  - RGB governs color of object itself + how much it transmits of each wavelength
- To blend things:
  output = src_rgb * src_alpha + (1-src_alpha) * dest_rgb
- Can alter either color or transparency without changing the other
  - Ex: Fade out: Keep lowering alpha; keep RGB the same
- Note: Perfectly transparent has *infinitely* many ways to specify
  - (0,0,0,0), (1,0,0,0), (0,1,0,0), (0,0.5,1,0), (1.0,1.0,0.3,0), ...

Conventional α

- glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
- glBlendEquation(GL_FUNC_ADD)
  - This is the default equation, so we don't need to set that one explicitly

Premultiplied α

- Philosophy:
  - RGB = How much color does source *add* to pixel
  - Alpha = Controls how much source *transmits*
- Only one color for fully transparent object: (0,0,0,0)
  - Object neither *adds* any color to scene nor *blocks* any color behind it
- To make object fade out: Must reduce alpha AND reduce RGB

Premultiplied α

- glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA)
- Again, blend equation can be left at its default of ADD

Example

- Interactive demo: Link

Overlay

- Conventional alpha can show outlines when sprites rendered (as textured quads)
- Premultiplied alpha helps prevent this.
- Demo: Link

Why

- Why do the outlines occur in classical alpha?
  - Opaque squares are (1,0,0,1) or (0,0,1,1)
  - What should transparent squares be?
    - Could be (1,0,0,0) or (0,0,1,0) or (0,0,0,0) or (1,1,1,0)
  - When we use linear filtering for texture, the transparent color “bleeds” into the nontransparent
    - Ex: If we used (0,0,0,0) for the transparent squares, we get (0.5,0,0,0.5) where the red fades to transparent
    - If we used (1,0,0,0) for transparent, then the red is fine, but the blue gives (0.5,0,0.5,0.5), which is purple
- Premultiplied alpha avoids this: We use (0,0,0,0) for transparent
  - As we move off red squares, we get (0.75,0,0,0), then (0.5,0,0,0), ... (0,0,0,0).

Converting

- Most graphics packages use classical alpha
  - I.e., image editors
- Can convert conventional $\rightarrow$ premultiplied
  - r'=r*a; g'=g*a; b'=b*a; a'=a;
  - Can precompute this as a preprocess

Other Uses

- Other useful blend modes:
- glBlendFunc(GL_ONE, GL_ZERO)
  - Same as no blending at all!
  - Seems useless...
  - But sometimes we want to render part of scene with no blending while other parts do use blending

Other Uses

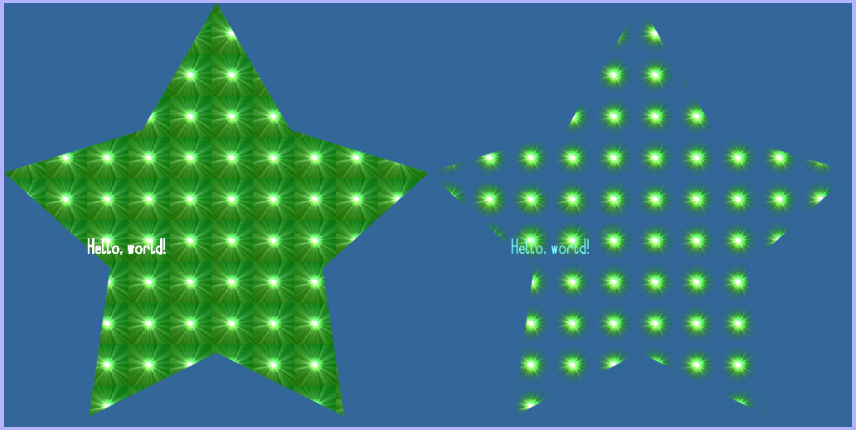
- Other useful blend modes:
- glBlendFunc(GL_SRC_ALPHA, GL_ONE)
  - Additive blending
  - Often used for sparks or similar

Text

- Example: Suppose we're rendering text
- Enable blending (with glEnable)
- We probably want classical α here
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
- These can be done in setup()

Rendering

- Example: Rendering text without and with α blending



Analysis

- The text looks much better!
- The star changes its appearance because the nova texture has α

Assignment

- When a bullet has traveled for a certain amount of time, make it explode in a shower of sparks. Have the shower of sparks fade away.
  - The sparks should have α transparency and be textured so they look good
  - Discuss: Particle systems
  - Discuss: Point sprites
  - Discuss: Using the VS to create a stateless particle system
- For bonus [+33%], instead of the bullets exploding after a certain amount of lifetime, make them explode when they hit an enemy. Have the enemy fade away too.
  - Discuss: Collision detection

Bibliography

- S. Hargreaves. Premultiplied alpha. http://blogs.msdn.com/b/shawnhar/archive/2009/11/06/premultiplied-alpha.aspx
- S. Hargreaves. Texture filtering: Alpha cutouts. http://blogs.msdn.com/b/shawnhar/archive/2009/11/02/texture-filtering-alpha-cutouts.aspx
- D. Schreiner, et al. *The OpenGL Programming Guide*.
- Tom Forsyth. Premultiplied alpha. http://home.comcast.net/~tom_forsyth/blog.wiki.html#[[Premultiplied%20alpha]]

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org