

# Computer Graphics

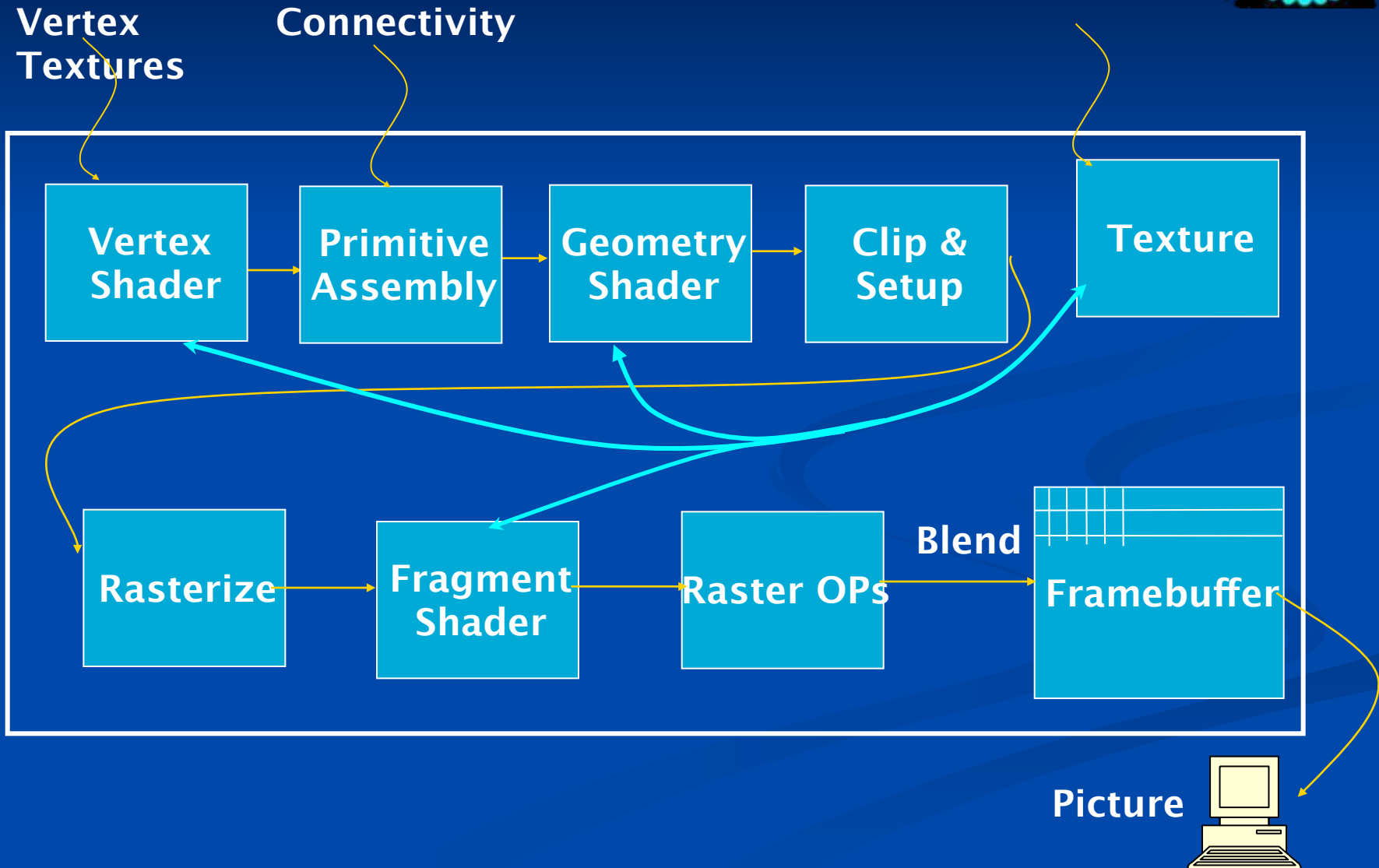
Subodh Kumar

Dept of Computer Sc. & Engg.

IIT Delhi



# Graphics Pipeline





# Raster Operations

- Z-buffer operations
- Color buffer operations
- Accumulation buffer operations
- Stencil buffer operations
- Note: a buffer is just that – a buffer
  - You bind it to be color, texture, etc.
  - So any buffer may be re-used as texture
    - Usually in a subsequent pass



# Z/Color-operations

- Read
- Write
  - Explicitly from the shader or implicitly
- Explicit shader reads not available (yet)
- Conditional Z-write allowed
- Write masks
  - `glDepthMask(GL_TRUE); glColorMask(.); glStencilMask(.)`
- Window “Clip”
  - `glScissor(x, y, width, height)`
- Color modify supported
  - Logic Ops
    - `glEnable(GL_COLOR_LOGIC_OP); glLogicOp(GL_XOR);`
  - Blend



# Color and Alpha

- Arbitrary fourth component
  - RGB + A
  - A is just a function parameter
    - E.g., Buffer R  $\leftarrow$  Buffer R - Fragment R,
    - Or, Buffer G  $\leftarrow$  Buffer G\*Fragment A + Fragment G
- Applies to each of  $\langle R, G, B \rangle$  normally
  - Any component can be masked out
- Can act as a 'Write Enable' as well

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_LEQUAL, 0.5)
```



# Stencil Buffer

- More general per pixel write enable

```
glEnable(GL_STENCIL_TEST);
```

```
glStencilFunc(GL_LESS, 1, 0x1); Function, Ref, Mask
```

```
glStencilOp(GL_REPLACE, GL_INCR, GL_INVERT);  
fail, zfail, zpass
```



# Stencil Buffer

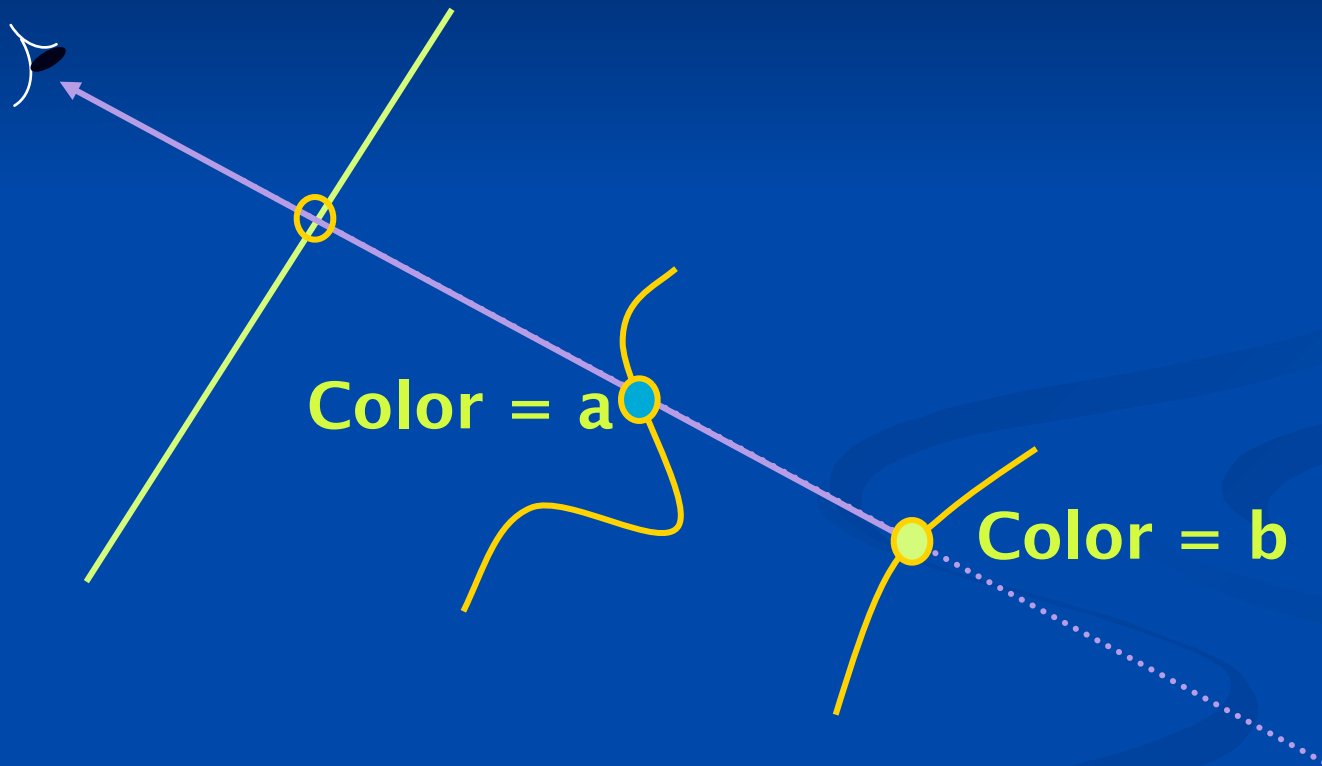
- Like color/z buffer; one entry per pixel
  - Traditionally a few bits of Z-buffer
- Stencil value also masks whether to render
- Render to Stencil
  - Stencil operation

void StencilFunc (enum func, int ref, uint mask )  
NEVER, ALWAYS, LESS, LEQUAL, EQUAL,  
GEQUAL, GREATER, or NOTEQUAL

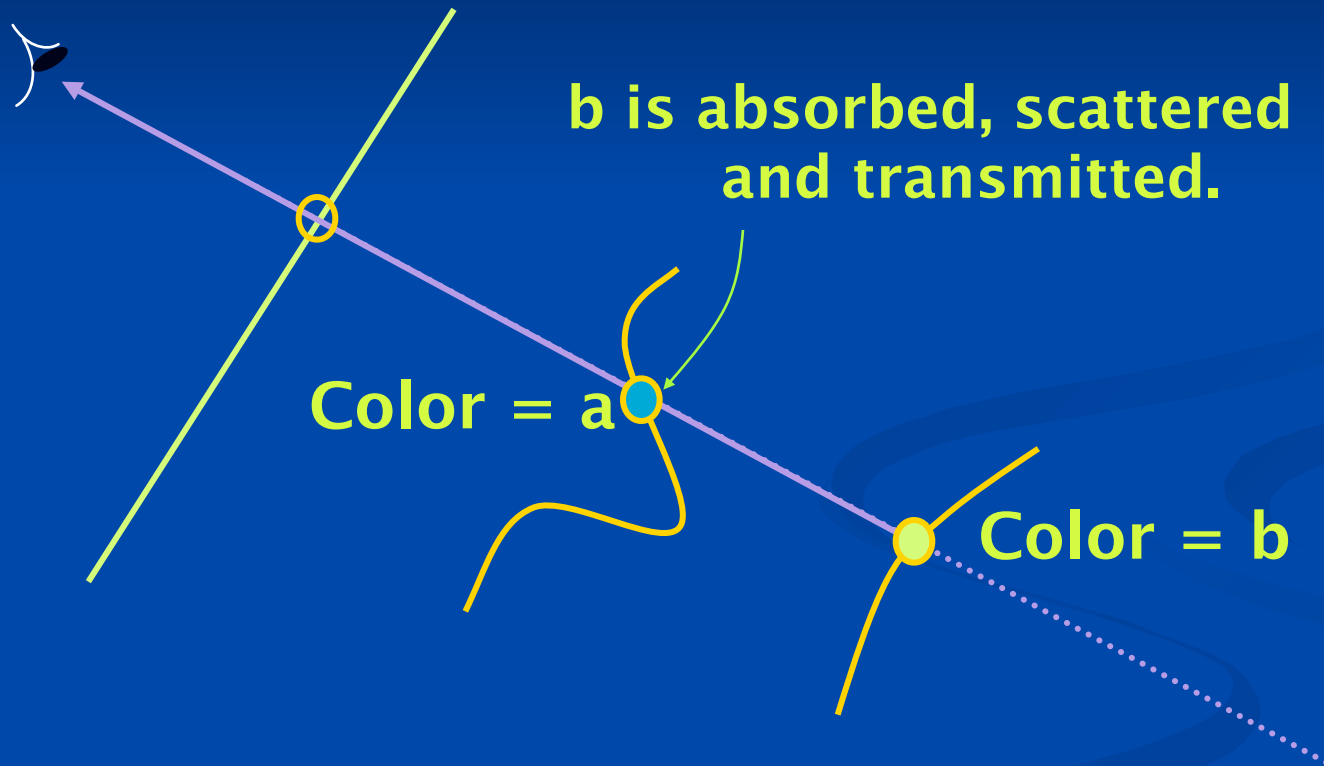
void StencilOp (enum sfail, enum zfail, enum zpass)  
KEEP, ZERO, REPLACE, INCR, DECR, INVERT

glEnable(GL\_STENCIL\_TEST)

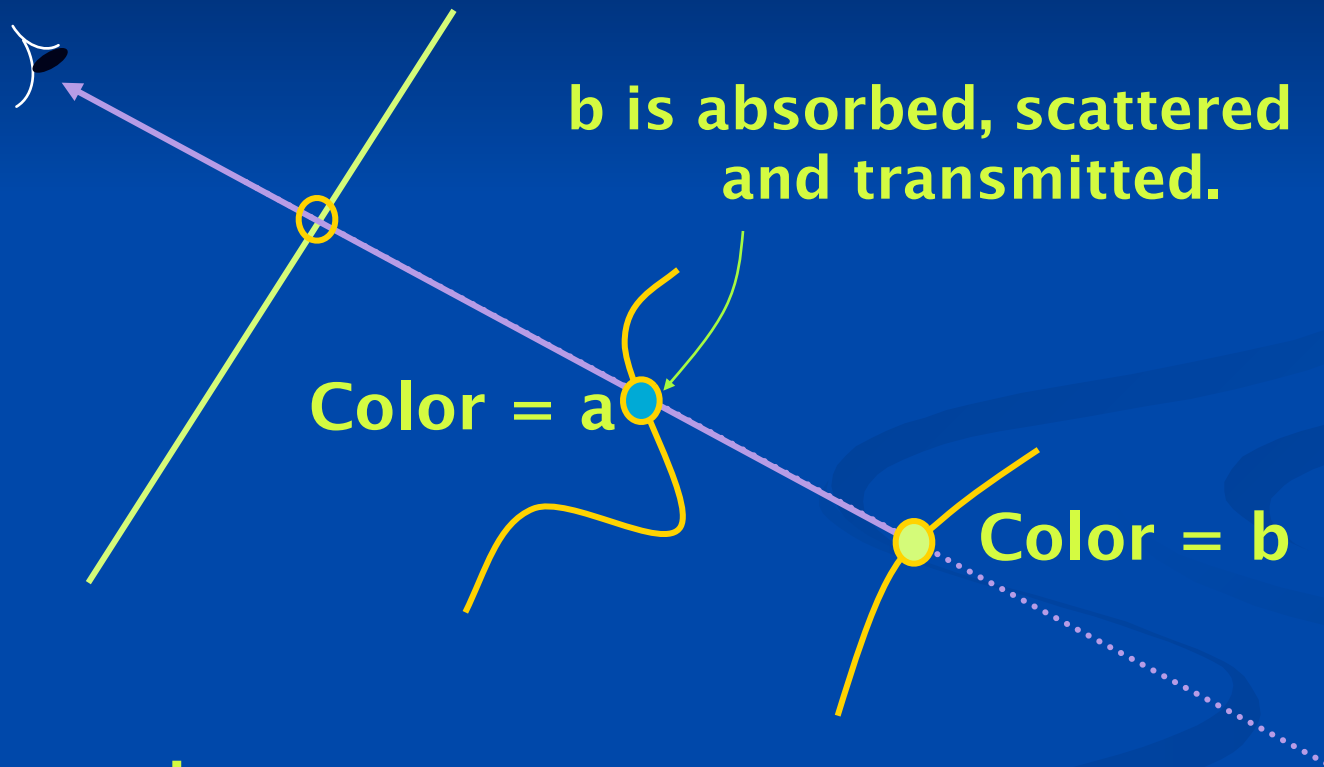
# Transparency → Blending



# Transparency $\rightarrow$ Blending



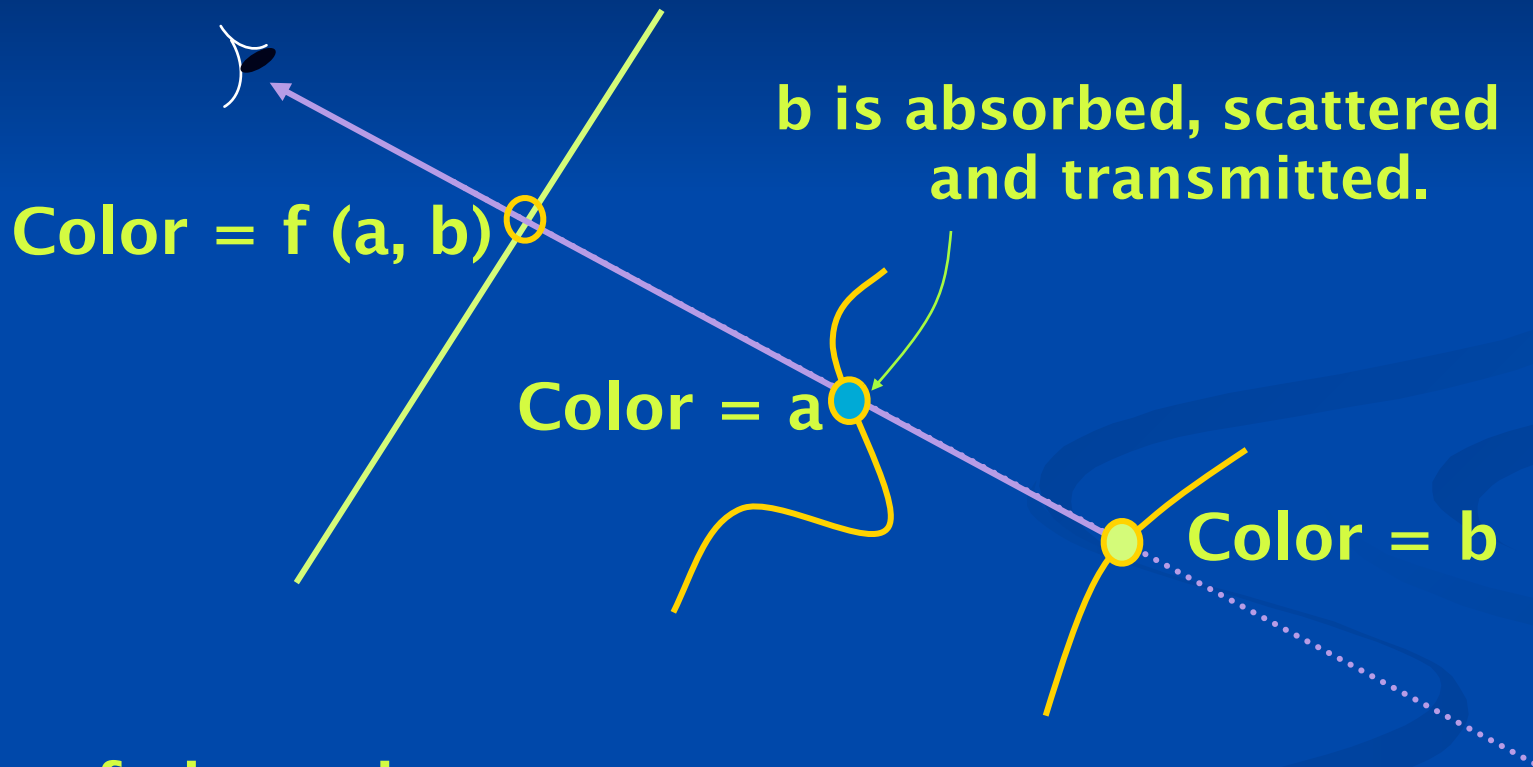
# Transparency $\rightarrow$ Blending



**f depends on:**

- Property of closer surface (translucence =  $\alpha$ )
- Depths of a and b

# Transparency $\rightarrow$ Blending

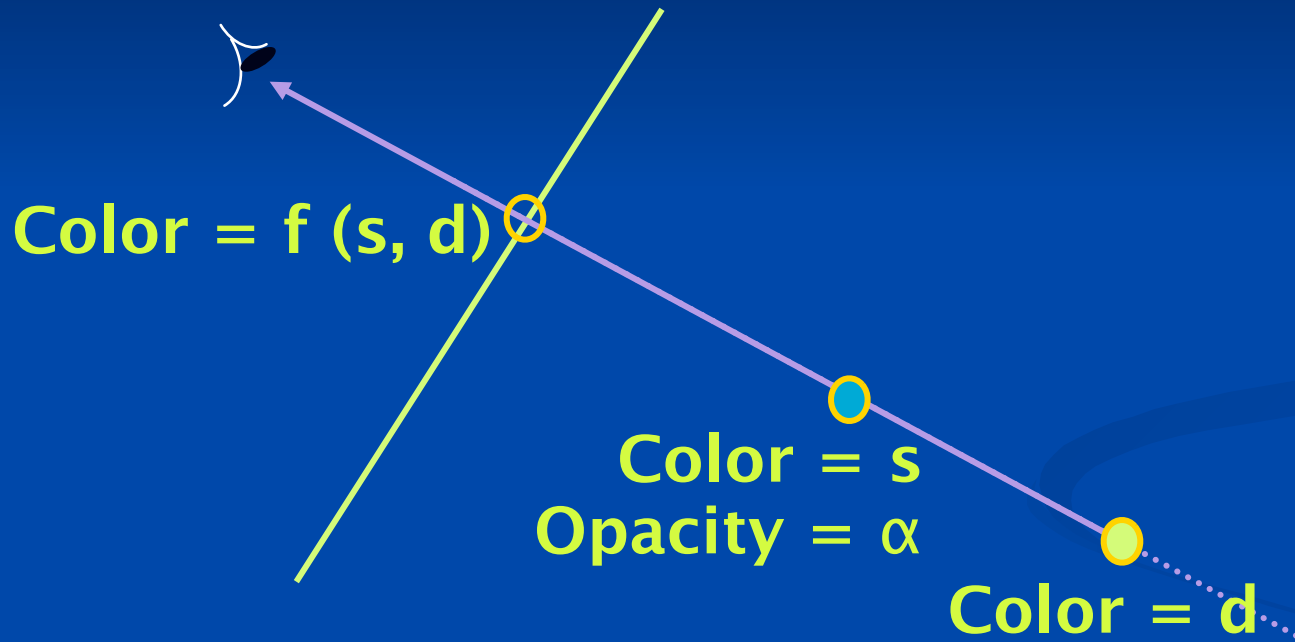


$f$  depends on:

- Property of closer surface (translucence =  $\alpha$ )
- Depths of  $a$  and  $b$

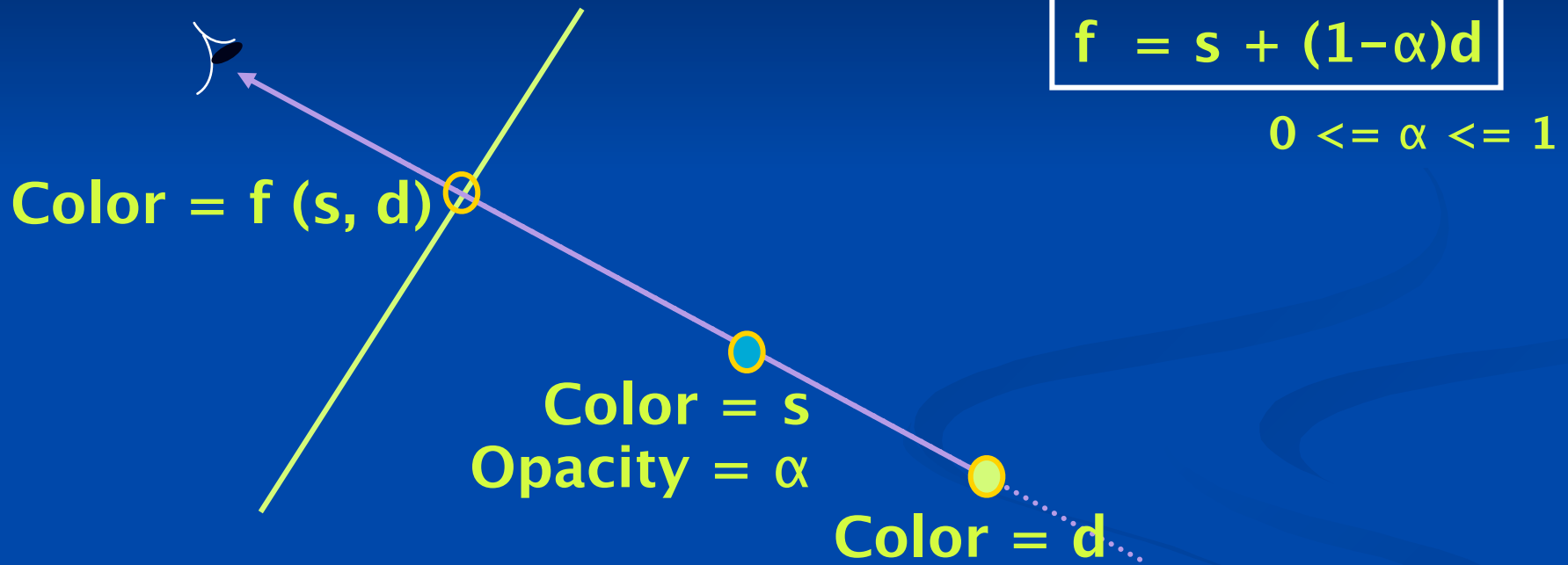


# Alpha Blending



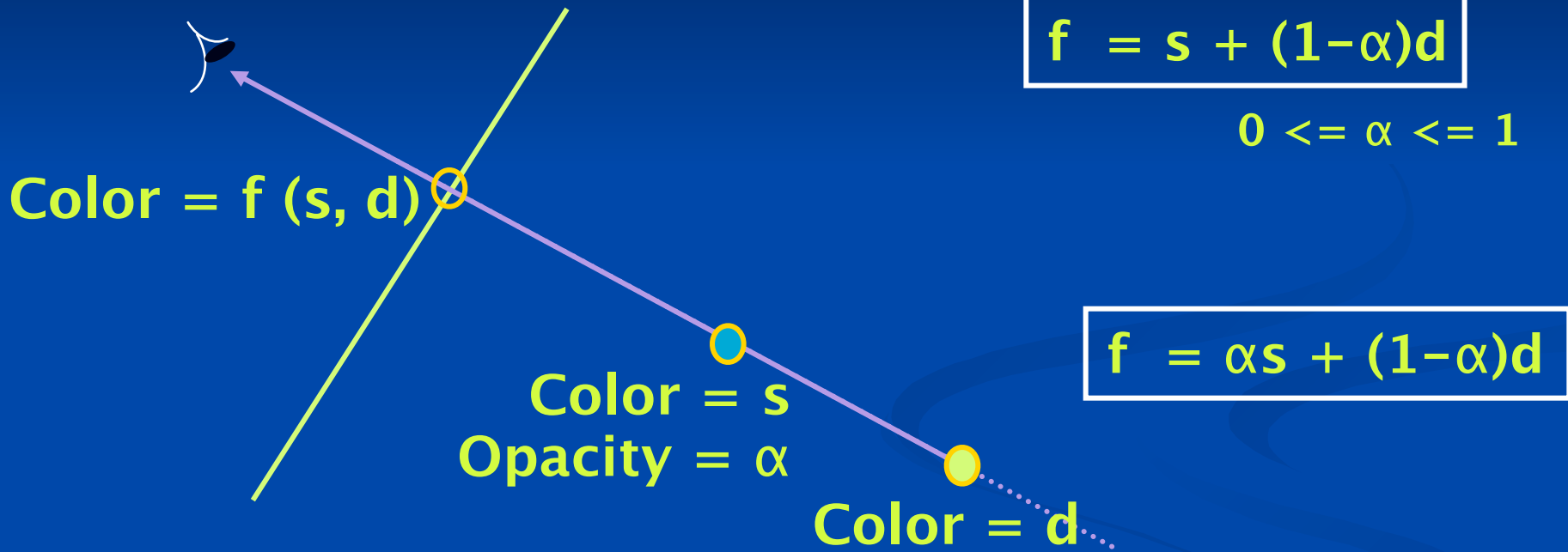


# Alpha Blending



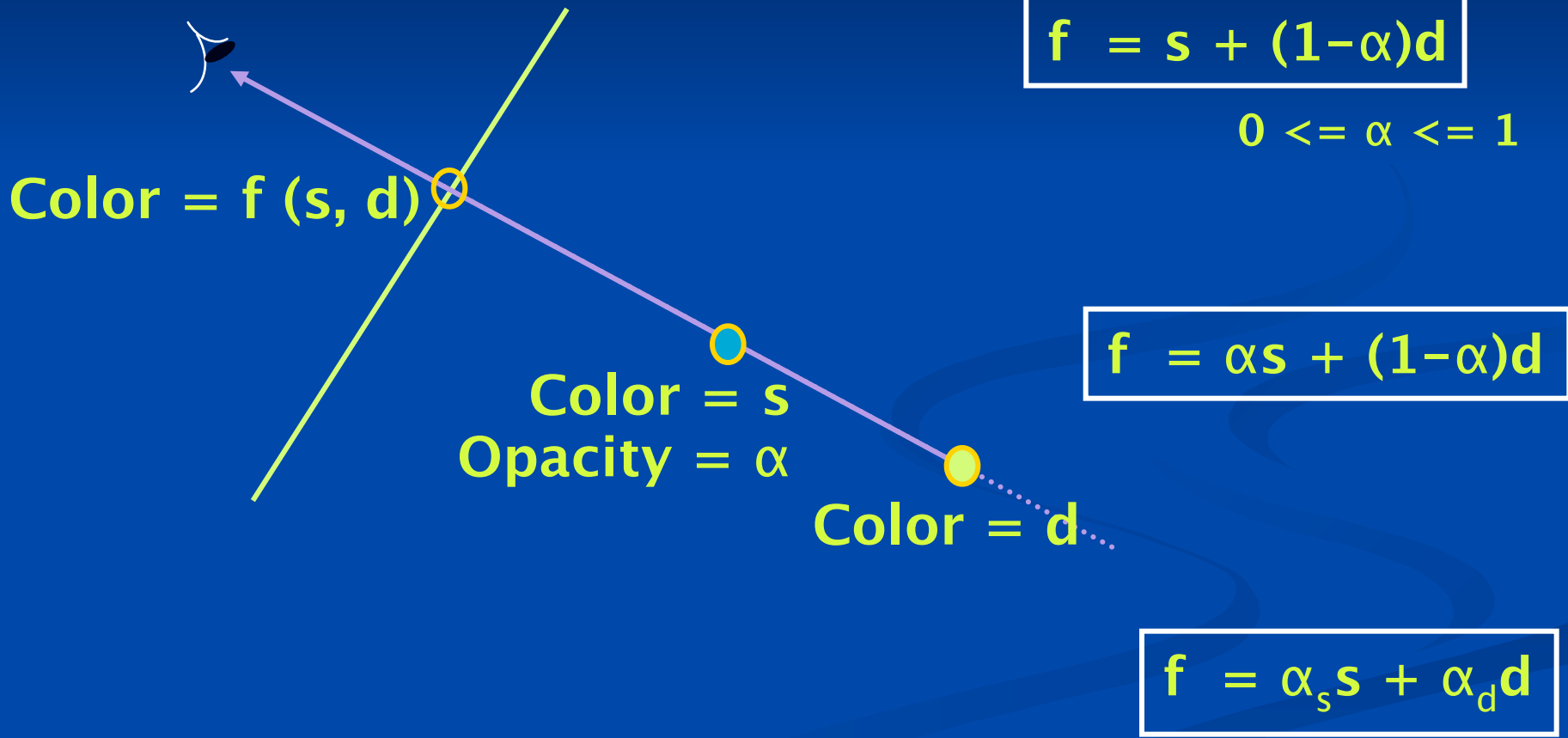


# Alpha Blending



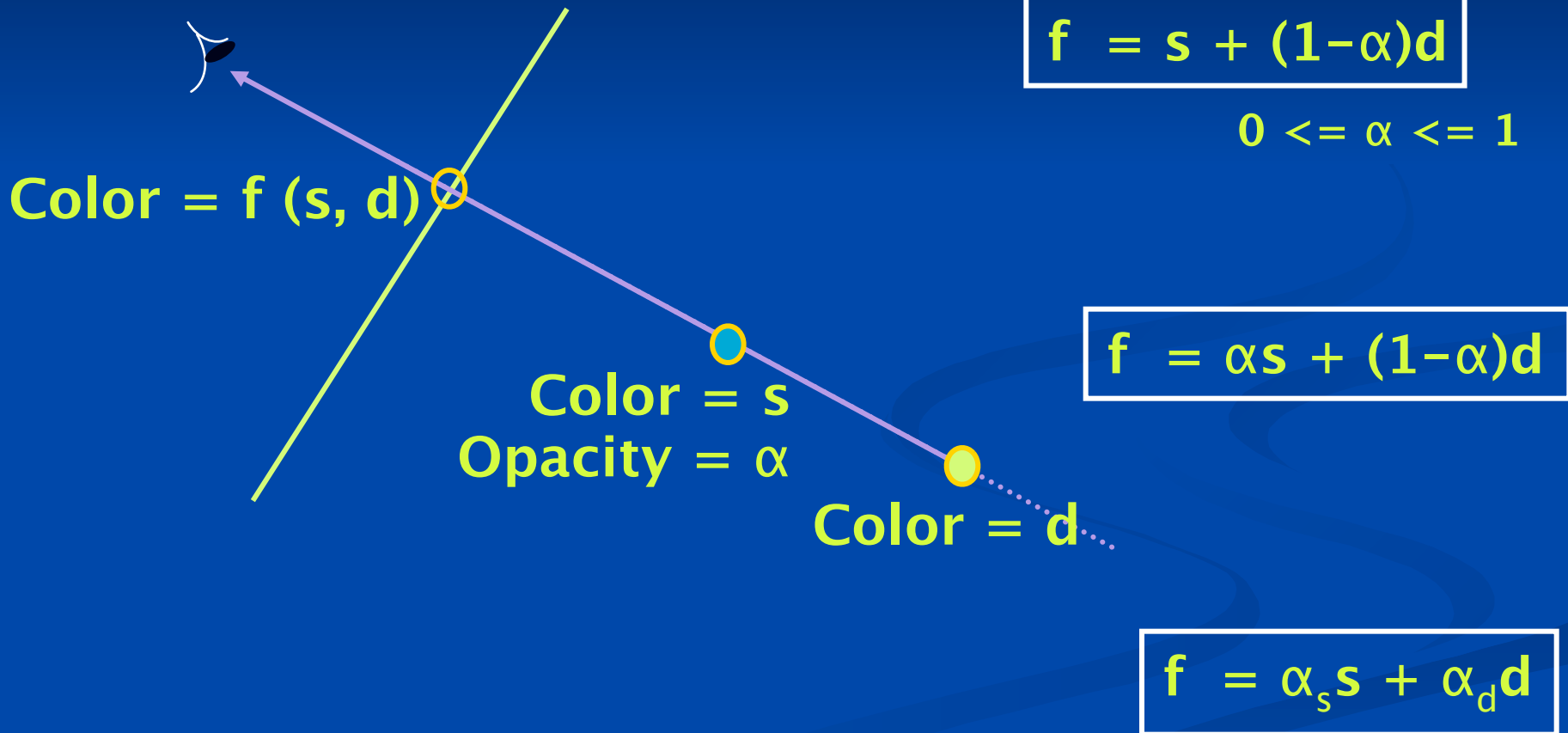


# Alpha Blending





# Alpha Blending



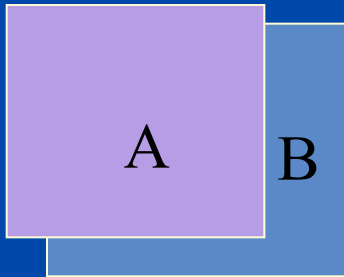
Ignoring light attenuation as before



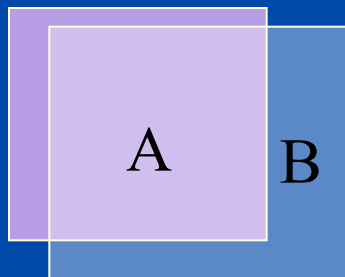
# Order Dependent

- $d' = \alpha_s s + (1-\alpha_s) d$

- Examples:  $\alpha_A = 1, \alpha_B = 0.4$

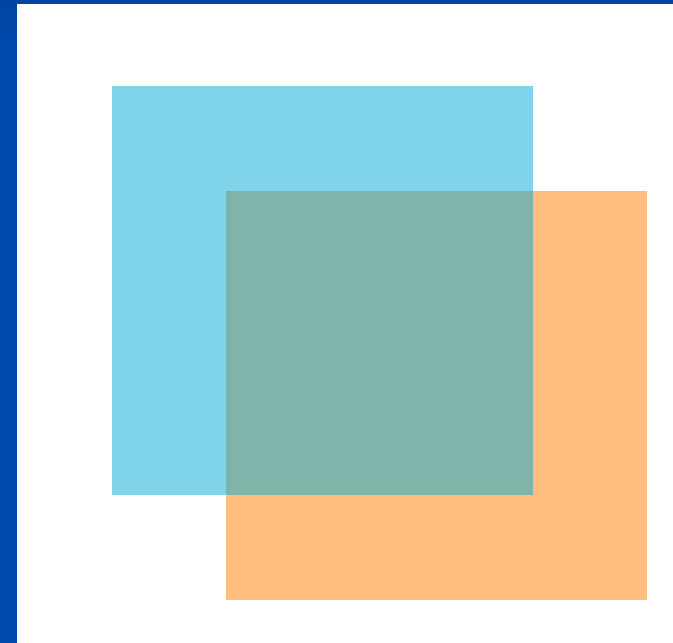
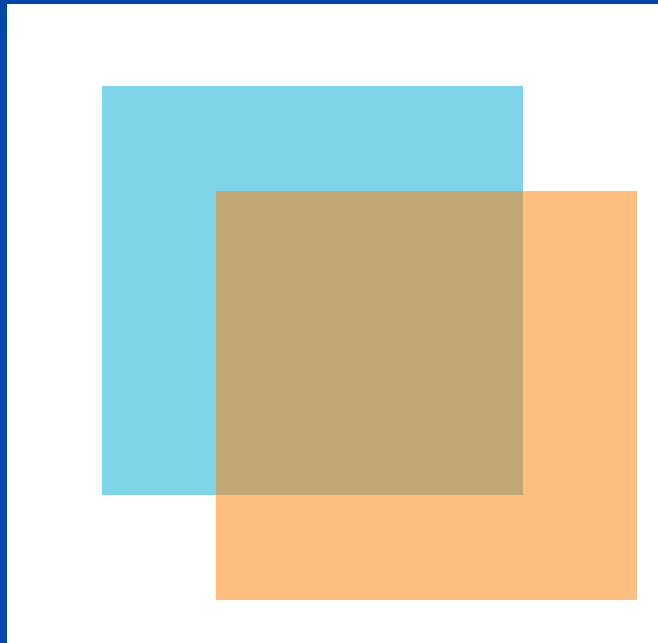


A over B:  
 $d' = 1 * C_A + (0) * C_B$



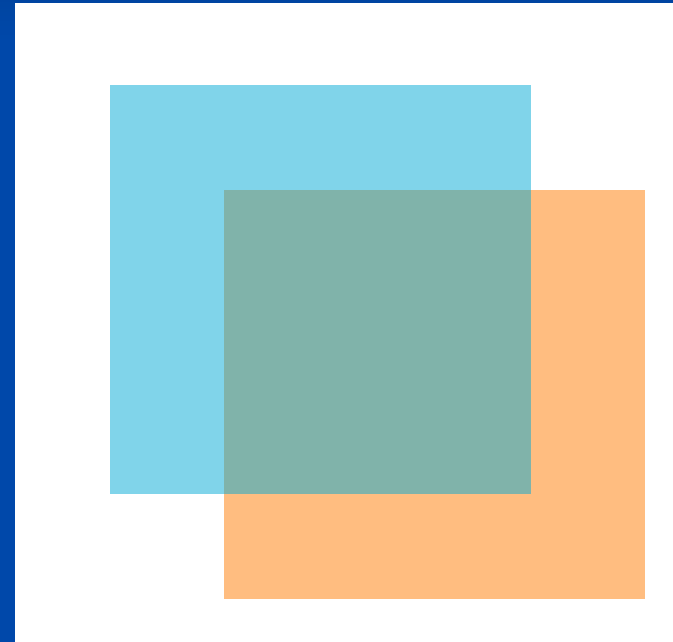
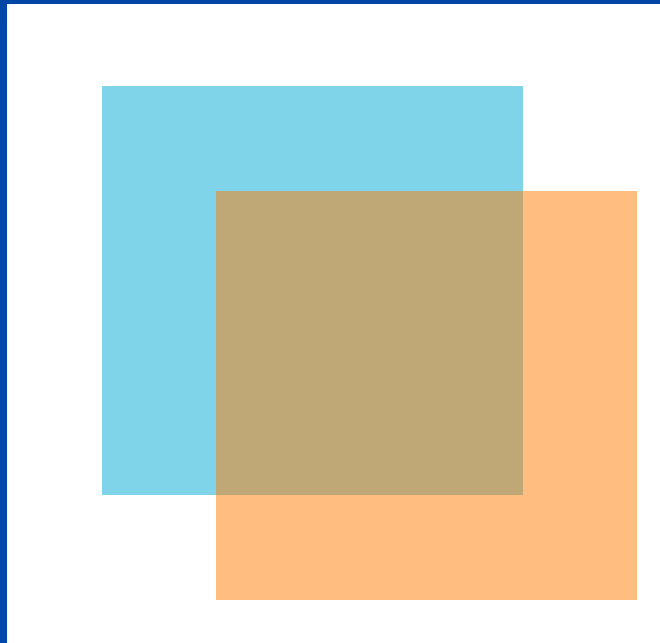
B over A:  
 $d' = 0.4 * C_B + (0.6) * C_A$

# Order Dependent





# Order Dependent

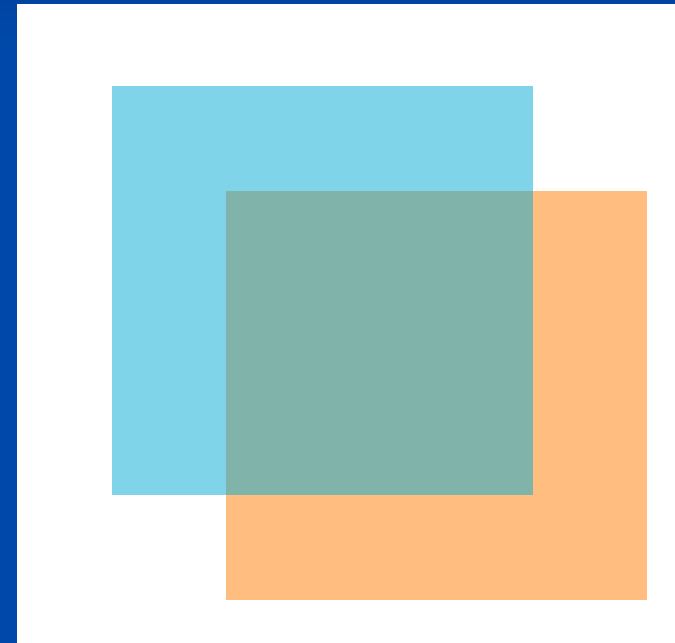
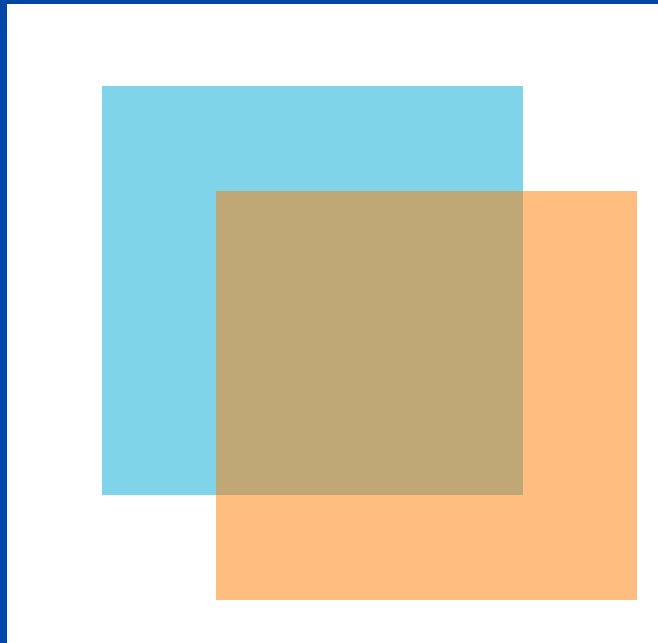


Here we blend the first quad with background



# Order Dependent

Even if each quad has an opacity of 0.5



Here we blend the first quad with background

# Alpha Textures



# Multiple depths per pixel



- **Sort back-to-front**
  - Cheap trick: Disable Z update, Blend color
  - Screen-door technique
- **A-buffer**
  - Multiple fragments per pixel
  - New hardware support needed
- **Multiple pass**
  - Each pass draws the next nearest at any given pixel



# Mammen's Algorithm

- Render all opaque primitives (into Buffer 1)
- Allocate second set of Color and Z buffers
- Set Z2 to be Overwritten by farther fragment
- Repeat
  - Clear Color2 and Z2 (to closest representable Z)
  - Rasterize all translucent primitives into Buffers2
    - If Fragment-Z behind Z2 but in front of Z1
  - Blend updates to Color2 into Color1
  - Merge Z2 into Z1 (closer of Z1 and Z2)
- Until no pixel is updated



# Depth Peeling (Front to Back)

n passes:

clear color buffer

For depth buffer 1 [Disabled for Pass 0]

Enabled depth test: Succeed if behind

Depth write disabled

Discards layers  
already saved

For depth buffer 2:

Clear depth buffer

Enable depth test: Succeed if in front

Finds nearest frag  
not discarded by 1

Enable depth writes

Render scene

save color buffer as the next layer

Swap the two depth buffers: Copy Depth2 to Texture

Two “depth buffers” are not supported, but depth textures are. Buffer1 is the texture. (May also use as a shadow-map, and update Alpha.)