

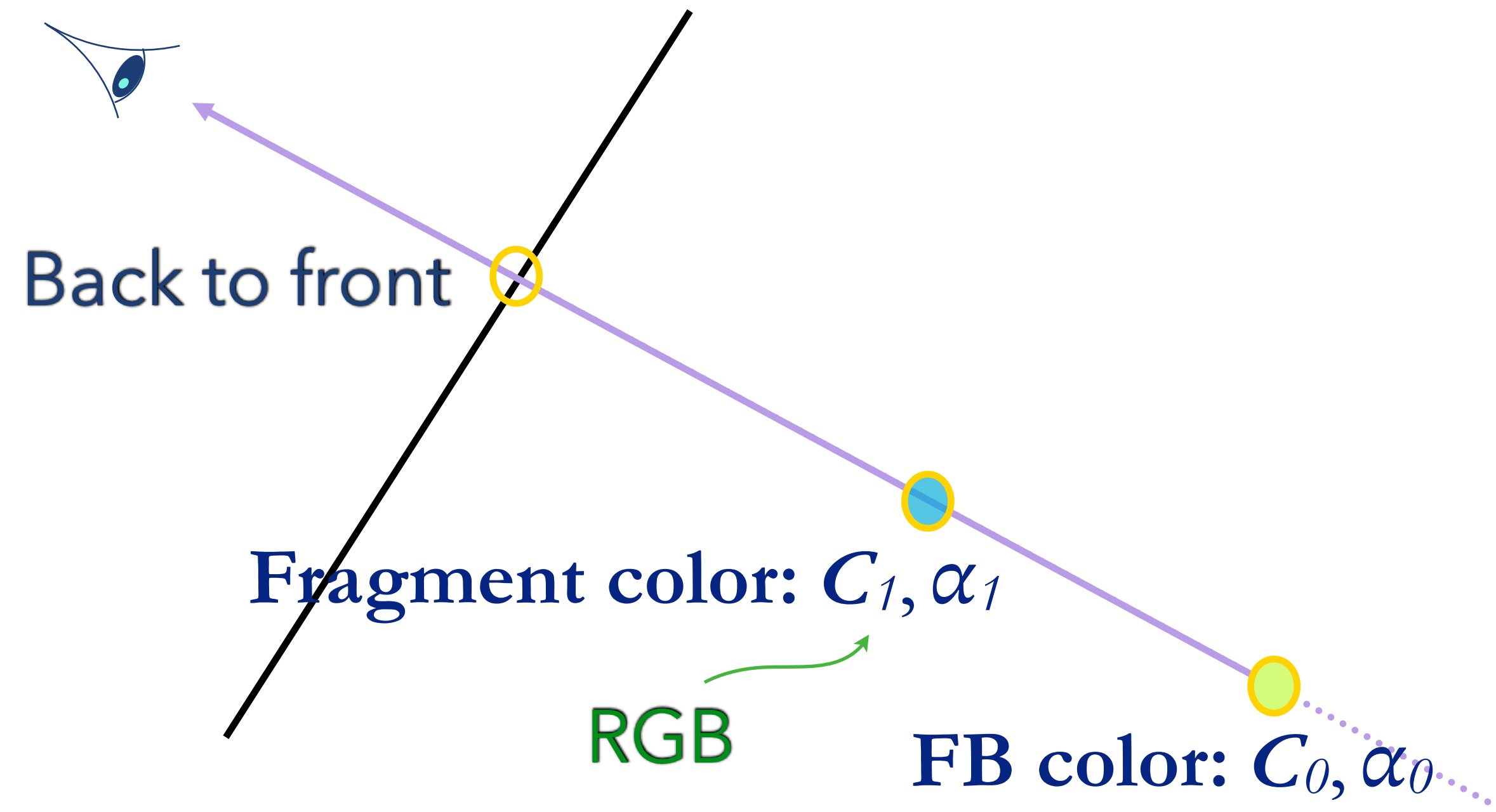
Computer Graphics

Transparency

Subodh Kumar, Dept of Computer Sc. & Engg., IIT Delhi



Blending



```
glBlendFunc(GL_SRC_ALPHA,  
            GL_ONE_MINUS_SRC_ALPHA)
```

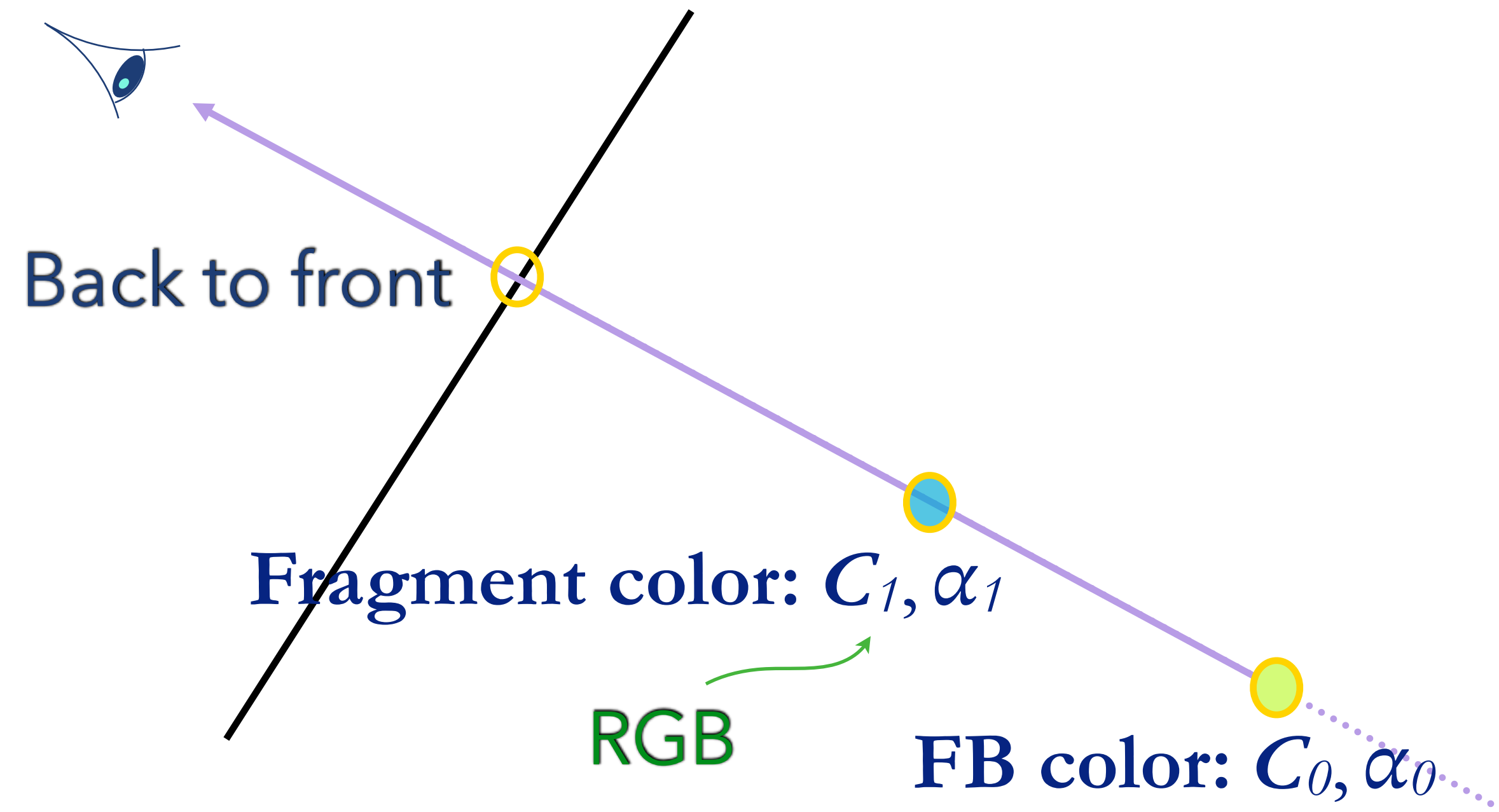


Mammen's Algorithm

- Allocate Buffer 1 and Buffer 2 Color (C1, C2) and Depth (Z1, Z2)
- Render opaque primitives into C1,Z1
- Set Z2 depth test to
 - ➡ pass for greater Z (farther fragment)
- Repeat
 - ➡ Clear C2 and Z2 (to closest Z, usually 0)
 - ➡ Rasterize all translucent primitives into C2+Z2
 - Write if Fragment-Z < Z1 (read from texture)
 - Fragment-Z in front of Z1
 - This test is in addition to the regular depth test
 - Blend C1 color to fragment-Color, write to C2
 - ➡ Swap (C1, Z1) and (C2, Z2)
- Until no pixel is updated



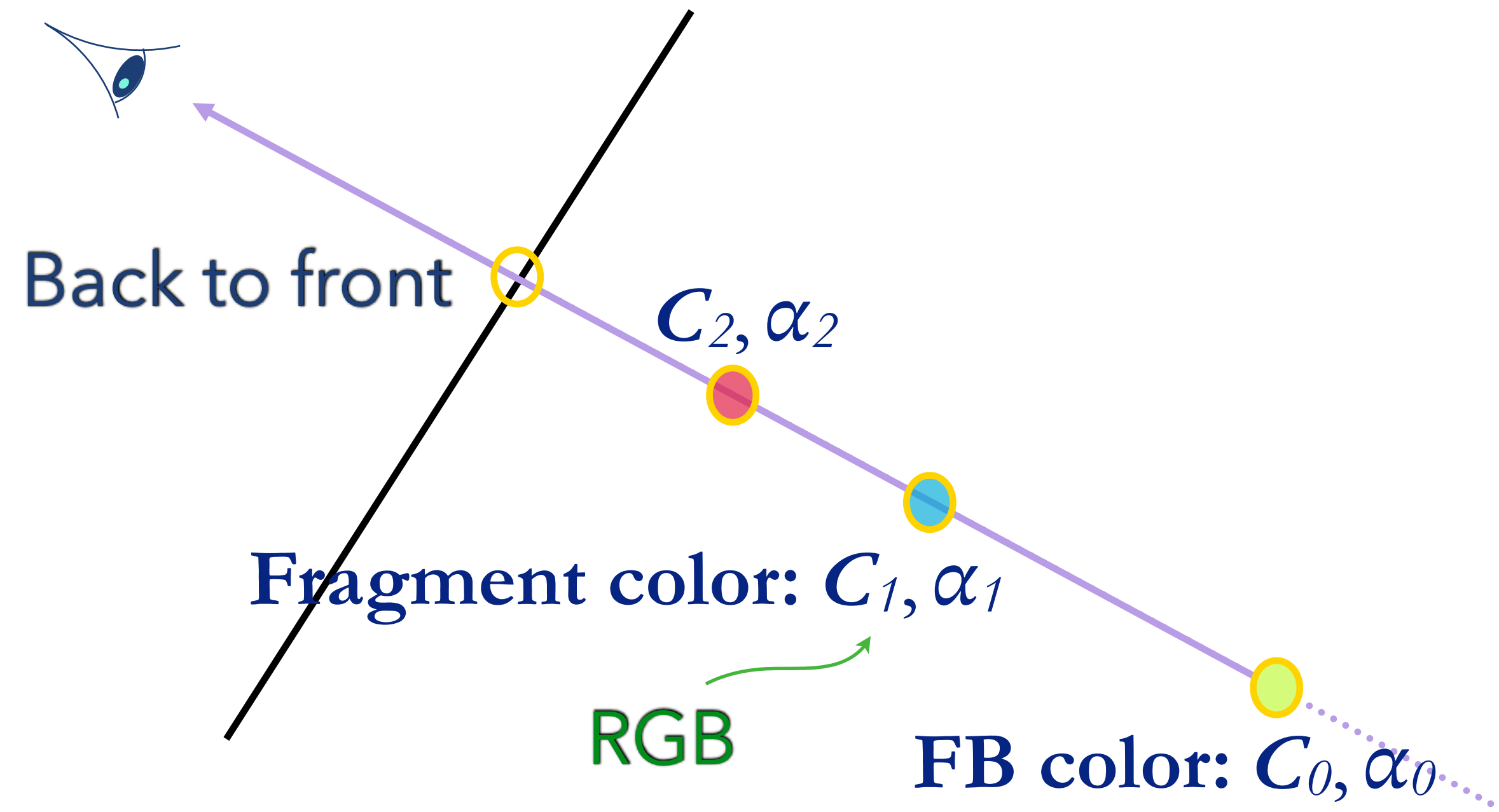
Blending Order



```
glBlendFunc(GL_SRC_ALPHA,  
            GL_ONE_MINUS_SRC_ALPHA)
```



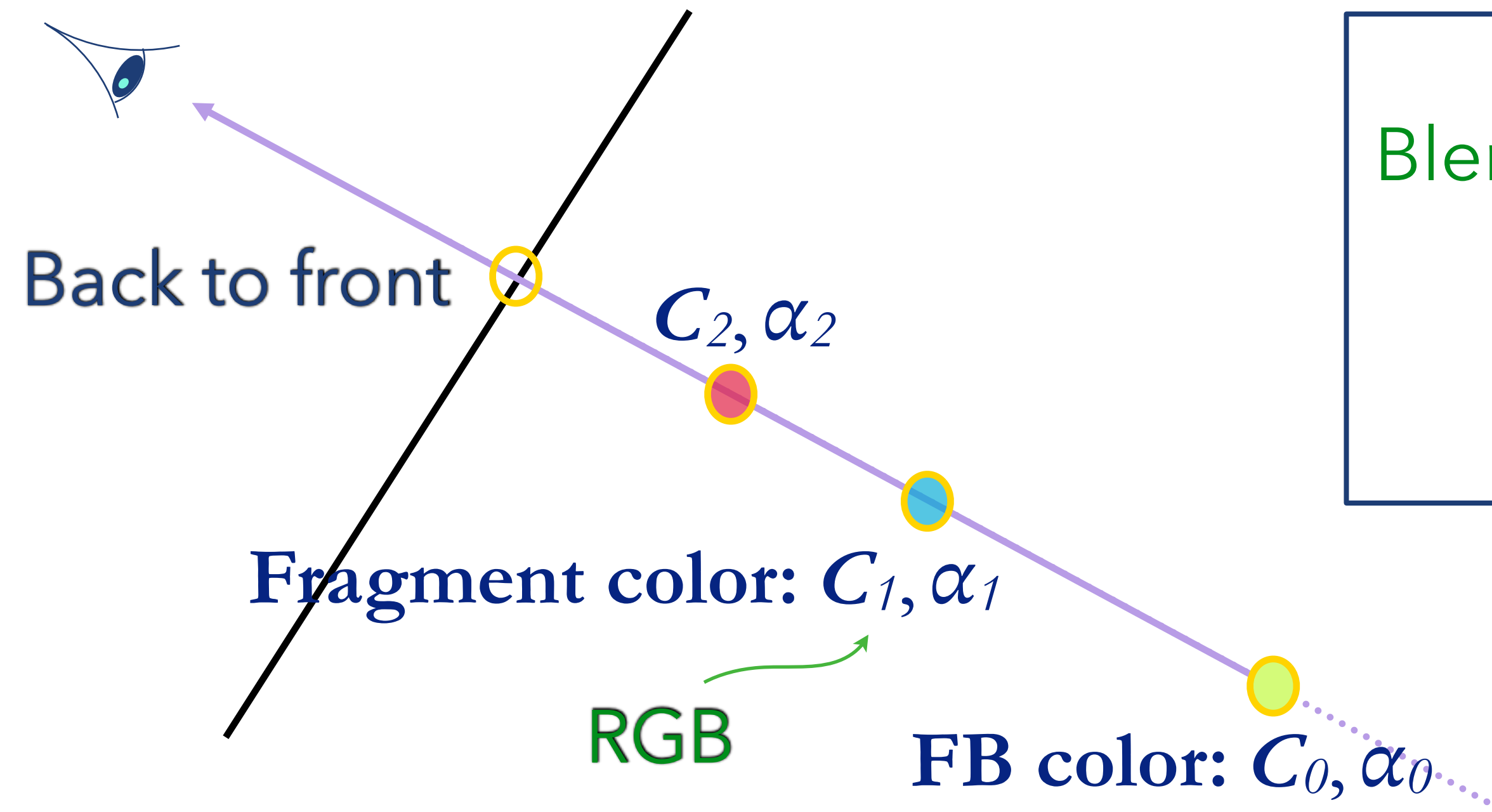
Blending Order



```
glBlendFunc(GL_SRC_ALPHA,  
            GL_ONE_MINUS_SRC_ALPHA)
```



Blending Order

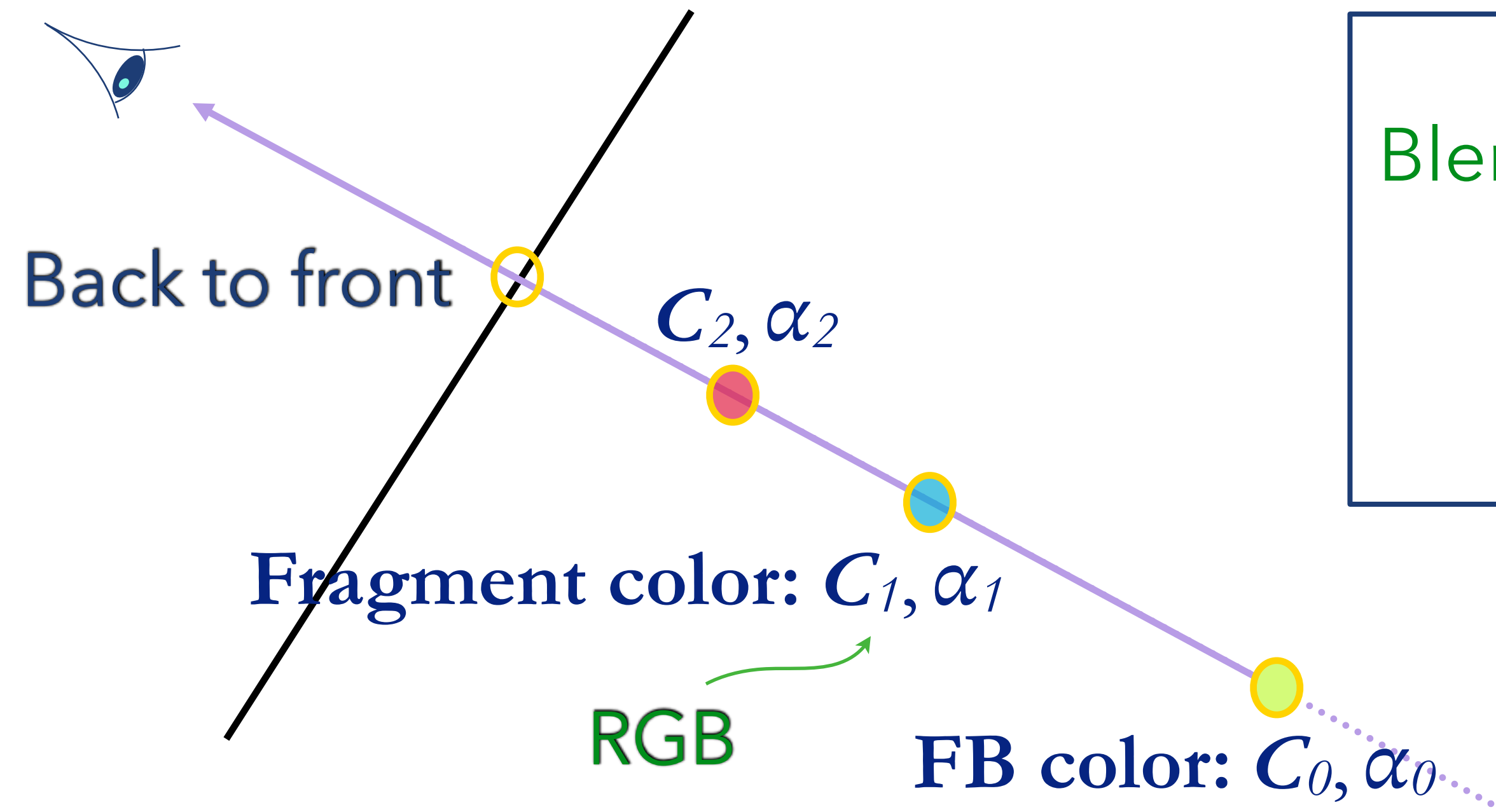


Back to front ordered
Blended color to FB (BG = 0,0,0,1)

`glBlendFunc(GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA)`



Blending Order



Back to front ordered

Blended color to FB (BG = 0,0,0,1)

$$\alpha_0 C_0$$

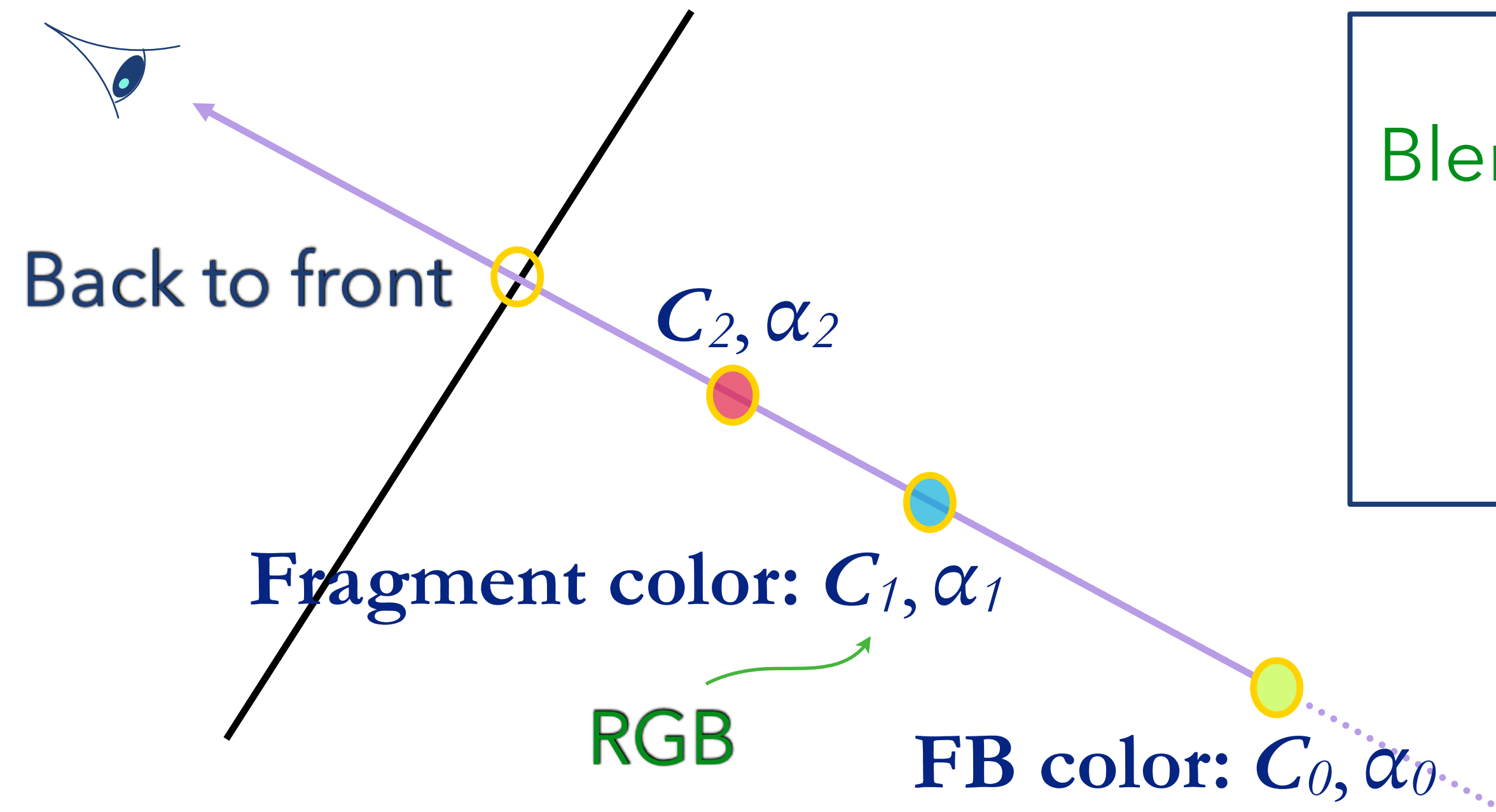
$$\alpha_1 C_1 + (1 - \alpha_1)(\alpha_0 C_0)$$

$$\alpha_2 C_2 + (1 - \alpha_2) [\alpha_1 C_1 + (1 - \alpha_1) \alpha_0 C_0]$$

`glBlendFunc(GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA)`



Blending Order



Back to front ordered

Blended color to FB (BG = 0,0,0,1)

$$\alpha_0 C_0$$

$$\alpha_1 C_1 + (1 - \alpha_1)(\alpha_0 C_0)$$

$$\alpha_2 C_2 + (1 - \alpha_2) [\alpha_1 C_1 + (1 - \alpha_1) \alpha_0 C_0]$$

`glBlendFunc(GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA)`

Front to back ordered

Blended color to FB: $(0,0,0) + 1.\alpha_2 C_2, 1.(1-\alpha_2)$

$$\alpha_2 C_2 + (1 - \alpha_2) \alpha_1 C_1, (1 - \alpha_1)(1 - \alpha_2)$$

$$[\alpha_2 C_2 + (1 - \alpha_2) \alpha_1 C_1] + (1 - \alpha_1)(1 - \alpha_2) \alpha_0 C_0, (1 - \alpha_0)(1 - \alpha_1)(1 - \alpha_2)$$



Front to Back Depth Peeling

Client

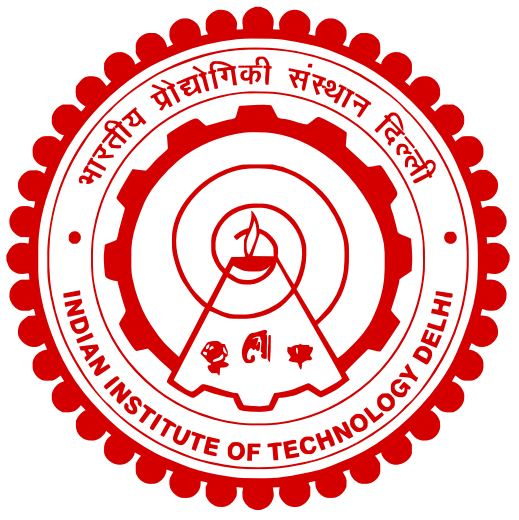
```
swap(curr, prev)
FBO DEPTH_ATTACHMENT, depthTex[curr]
Clear Depth
if not firstPass:
    glBindTexture(GL_TEXTURE_2D, depthTex[prev]);
```

Shader

```
vec2 uv = gl_FragCoord.xy / (w,h)
float z = gl_FragCoord.z

if not firstpass:
    float zPrev = texture(prevDepth, uv).r;
    if (z <= zPrev + 1e-6): discard
```

Disables Early-Z testing



Front to Back Depth Peeling

Client

```
swap(curr, prev)
FBO DEPTH_ATTACHMENT, depthTex[curr]
Clear Depth
if not firstPass:
    glBindTexture(GL_TEXTURE_2D, depthTex[prev]);
```

Shader

```
vec2 uv = gl_FragCoord.xy / (w,h)
float z = gl_FragCoord.z

if not firstpass:
    float zPrev = texture(prevDepth, uv).r;
    if (z <= zPrev + 1e-6): discard
```

Disables Early-Z testing

Optimizations:

- Add a Z-only pre-pass
- Dual peeling



Weighted Blended OIT

- Each transparent fragment contributes a fraction of its color
 - ➔ Deeper fragments contribute less

```
accum = 0, bg = 1  
weight = f(fragZ, color.a)*color.a  
accum.rgb += color.rgb*weight;  
accum.a += color.a;  
bg *= (1 - color.a);
```

clamp($1.0 / (1e-5 + \text{linearZ})$, 0.01, 10.0) //e.g.

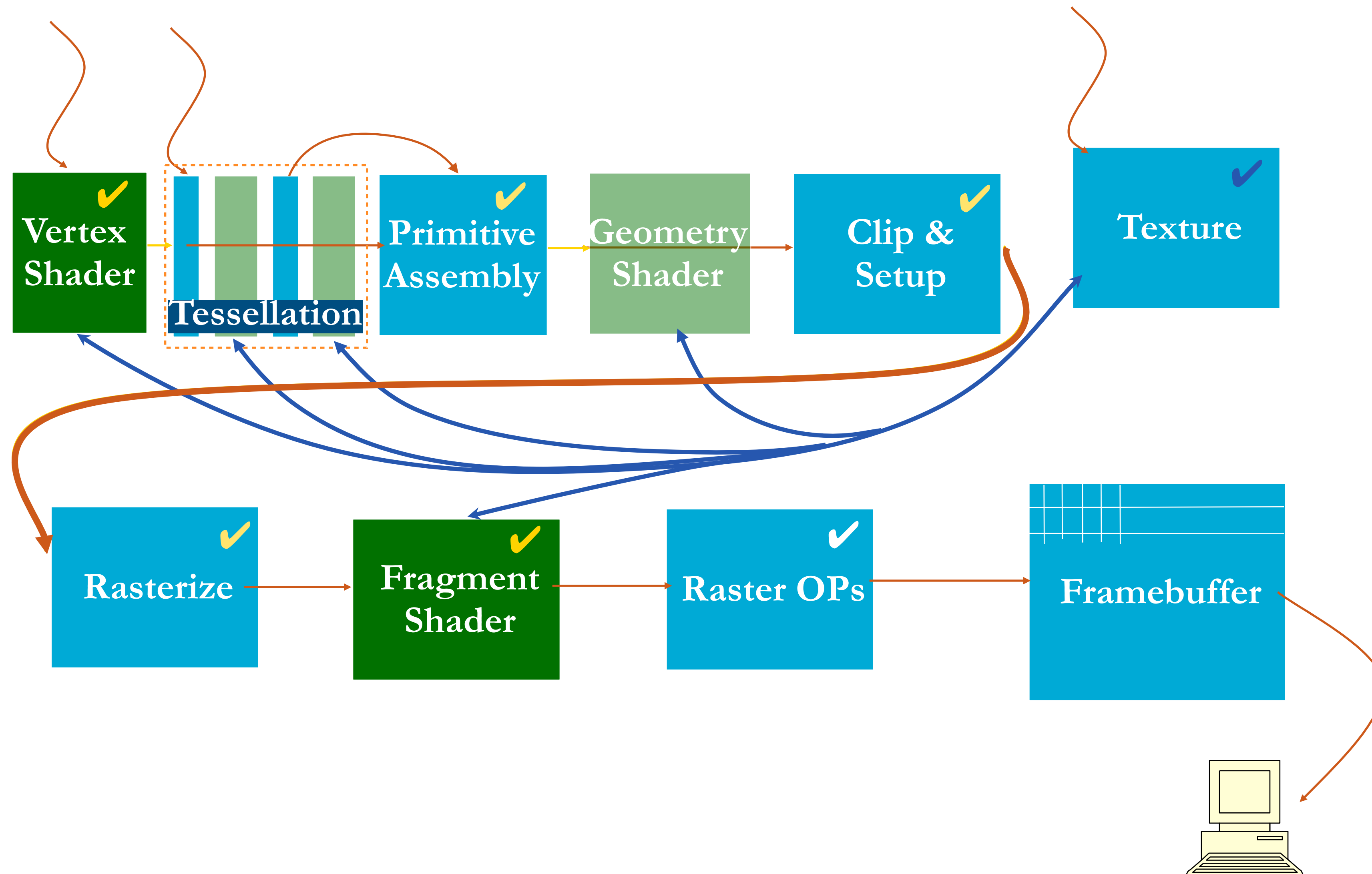
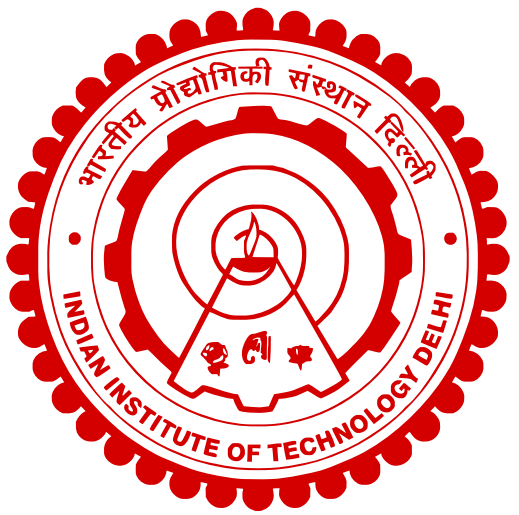
Or, clamp($\alpha^3 * (1.0 - \text{fragZ})^3 + 0.01$, 0.01, 1.0)

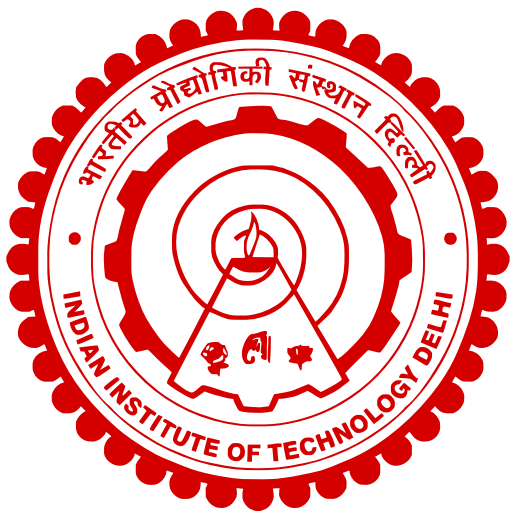
Later normalize:

```
rgb = accum.rgb / max(accum. $\alpha$ ,  $\epsilon$ );  
 $\alpha = 1 - \text{bg}$ 
```

How about Refraction?

Graphics Pipeline





Raster Operations

- Output is to frame-buffer
- Input may include frame-buffer data
 - ➡ Fixed function hardware close to the frame buffer memory can access it efficiently
 - ➡ Bringing that into the shader induces large latency
- Input includes FS output
 - ➡ R, G, B, Alpha
 - ➡ Z
 - ➡ Stencil: a constant number of bits per pixel (mainly fixed function)
- Inter-operate:
 - ➡ Effect Color write based on Alpha value
 - ➡ Effect Color write based on stencil
 - ➡ Effect Z write based on stencil

see "discard" in glsl



Z/Color-operations

- Conditional writes

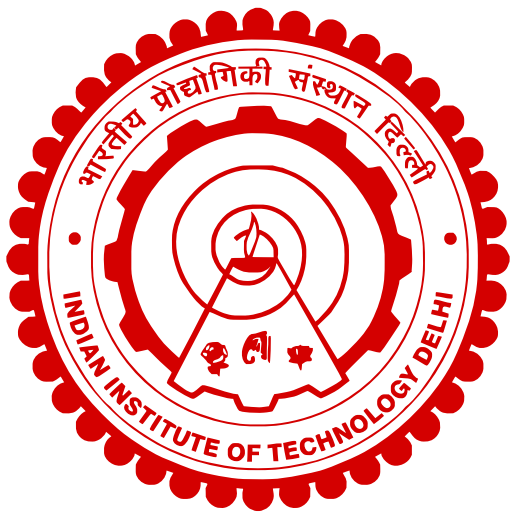
- ➡ Write masks: write if mask is 0 (or *vice versa*)
 - e.g., `glDepthMask(GL_TRUE); glColorMask(.); glStencilMask(.)`
 - Masks can be per component: R, G, B, Z, ..
- ➡ Stencil is mainly used for more complex per-pixel masking

- Arbitrary region ("clip")

- ➡ `glScissor(x, y, width, height)`

- FB Color modification

- ➡ Using Logic Operations
 - `glEnable(GL_COLOR_LOGIC_OP); glLogicOp(GL_XOR);`
 - $\Rightarrow \text{FB.color} = \text{FB.color} \text{ XOR } \text{FS.color}$
- ➡ By Blending
 - $\text{FB.color} = a * \text{FB.color} + b * \text{FS.color}$



Depth

- Z is just another variable to control FB writes
- Usually has the depth value of the fragment
 - ➔ Perspective-correct interpolation of $(1/Z)$
 - ➔ Normalized to range 0..1 (Viewport Z)
 - ➔ Other applications possible

`glEnable(GL_Depth_TEST); // Use depth test`

`glDepthFunc(GL_LEQUAL); // Write if  $FS.Z \leq FB.Z$`



Color and Alpha

- Arbitrary fourth component per pixel
 - ➔ RGB and A
 - ➔ A is 'just a parameter' for per-pixel operation
 - Or, $FB.G \leftarrow FB.G * FS.A + FS.G$
- Alpha can also be used to mask fragment write
 - ➔ `glEnable(GL_ALPHA_TEST); // Use alpha test`
 - ➔ `glAlphaFunc(GL_LEQUAL, 0.5); // write if  $FS.A \leq 0.5$`

Also see `GL_SAMPLE_ALPHA_TO_COVERAGE`



Stencil Buffer: S

- More general per pixel write enable

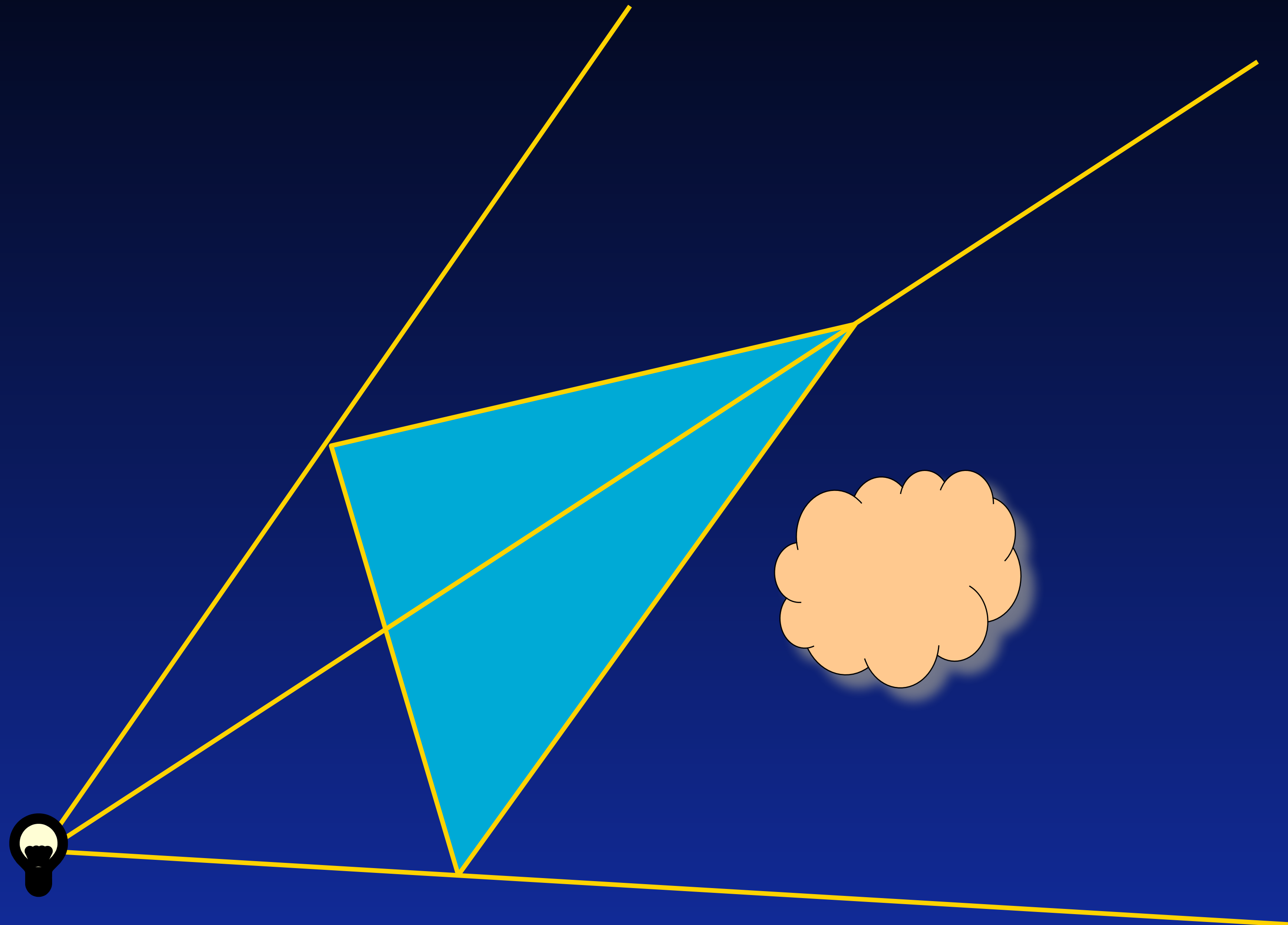
Also see: `glStencilFuncSeparate`

```
glEnable(GL_STENCIL_TEST);           // Stencil test
```

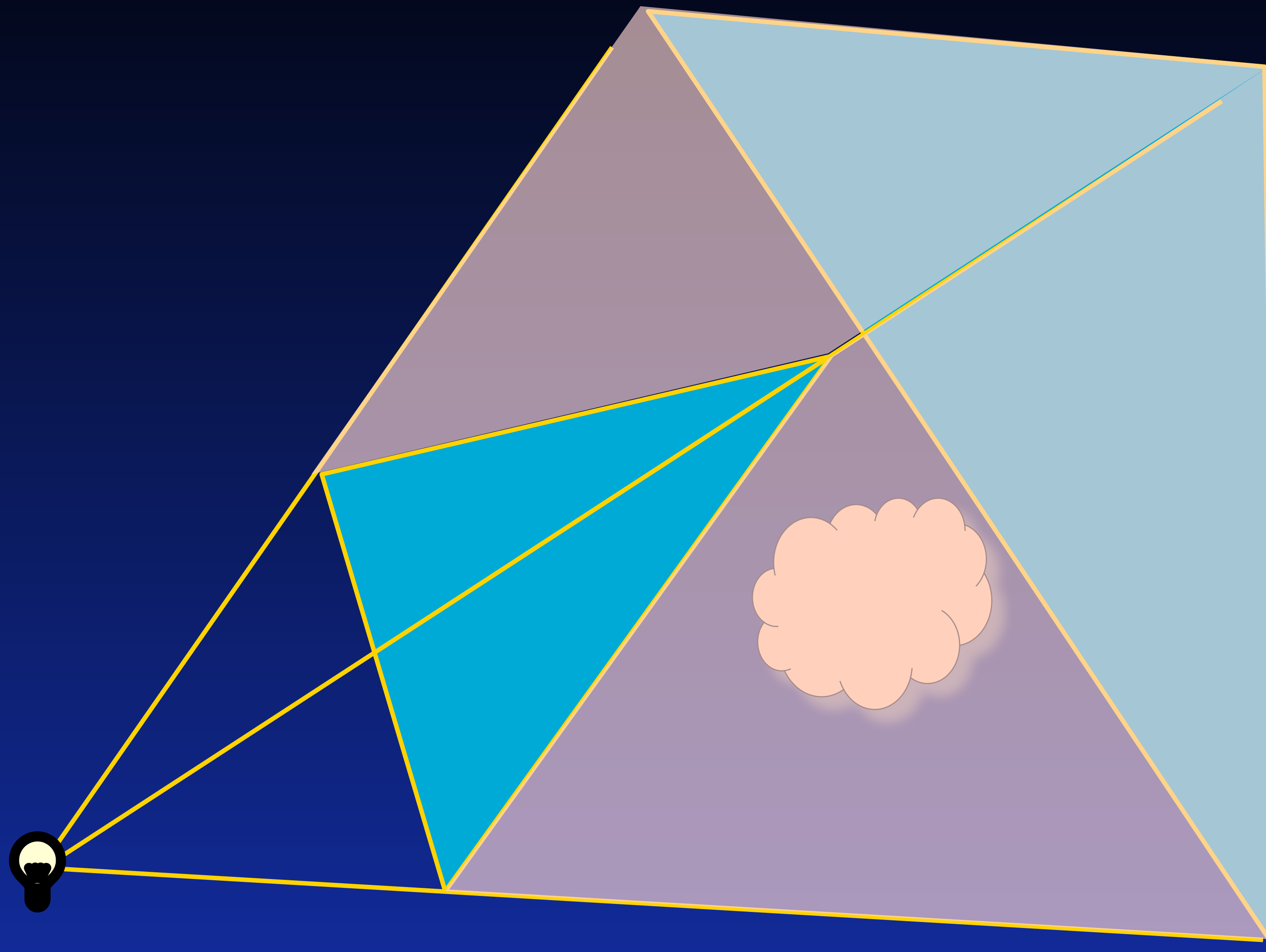
```
           function ref mask  
glStencilFunc(GL_LESS, 1, 0x1); // Write fragment if  $FB.S \& 0x1 < 1$ 
```

```
           sfail           zfail           zpass  
glStencilOp(GL_REPLACE, GL_INCR, GL_INVERT);  
           //  $FB.S \leftarrow FB.S$  fails? Ref :  $FS.Z$  passes?  $(\sim FB.S) : (FB.S + 1)$ 
```

Shadow Volumes



Shadow Volumes

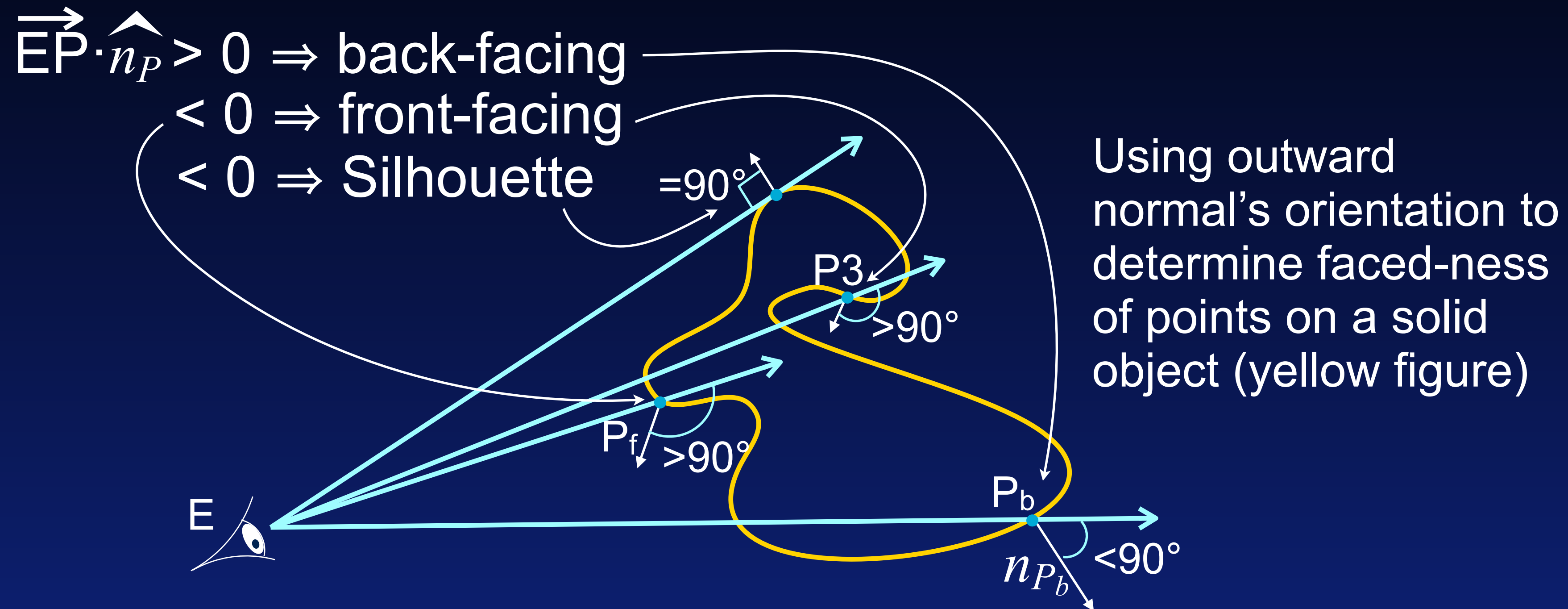


Stencil-based Shadow Volume Algorithm



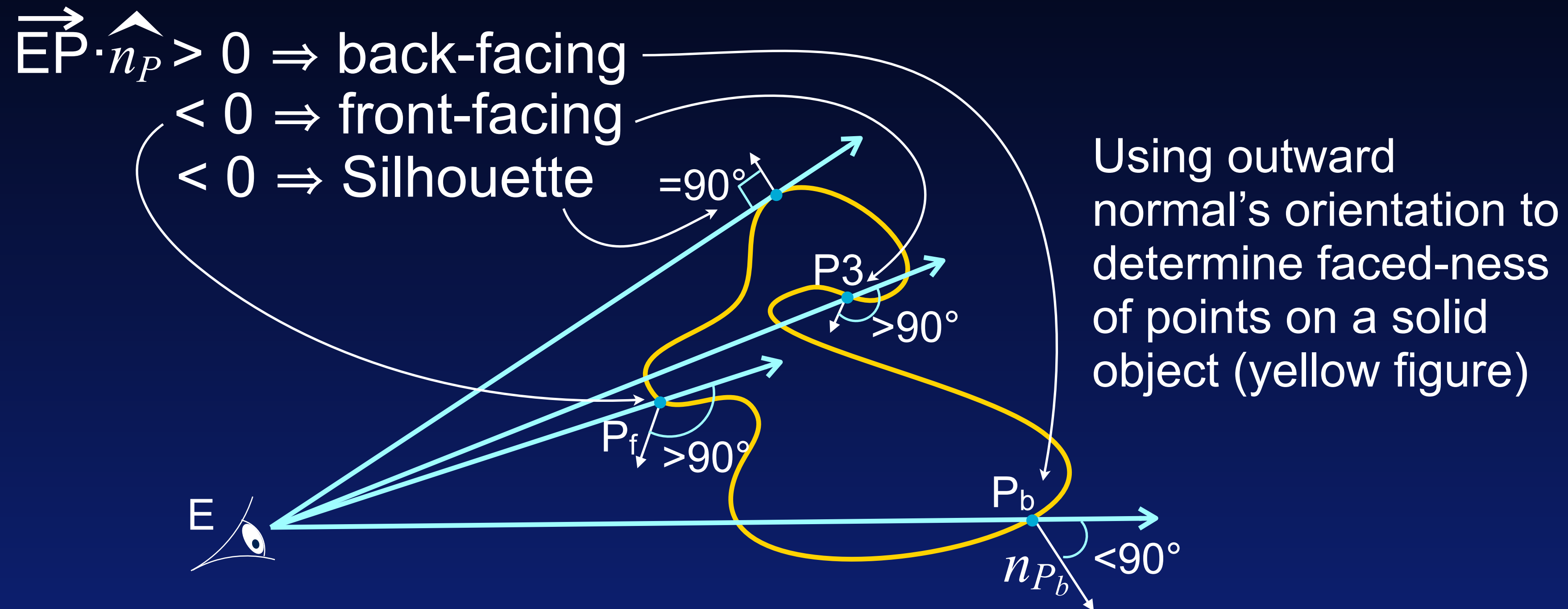
- Render the scene in shadow color
 - ➡ Maybe black; we want the Z-buffer
- For each light source
 - ➡ Using the depth from step 1, and shadow polygons, build stencil buffer, disabling scene-pixels in shadow
 - ▶ Use pre-built shadow polygons
 - ▶ Find silhouettes of occluders
 - ▶ Find far away points on the rays to silhouettes
 - ▶ Form the polygons, possibly triangulate
- Render the scene again, with lighting
 - ➡ Stencil buffer masks out shadowed areas

Front and Back Facing



- Outward-normal's direction is not the only way to determine faced-ness.
 - Sometime, normals are not provided, or they may be artificially created.
- A triangle's vertices can instead be oriented consistently.
 - For example, they may be clock-wise (cw) when viewed from exterior.
 - In that case, if they appear cw on screen, the eye is in the exterior.

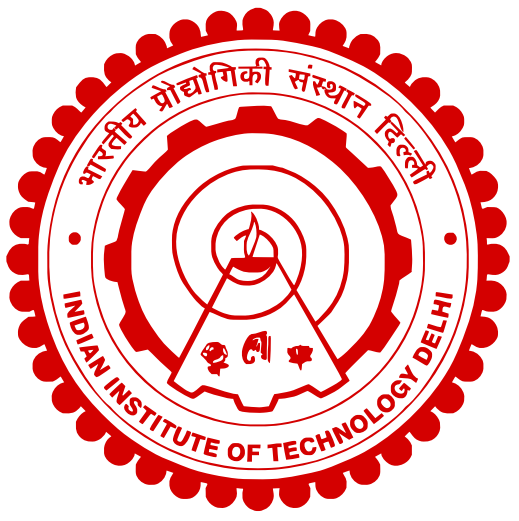
Front and Back Facing



OpenGL allows automatic discarding of back-facing primitives —
It is called culling.

These tests are customizable —

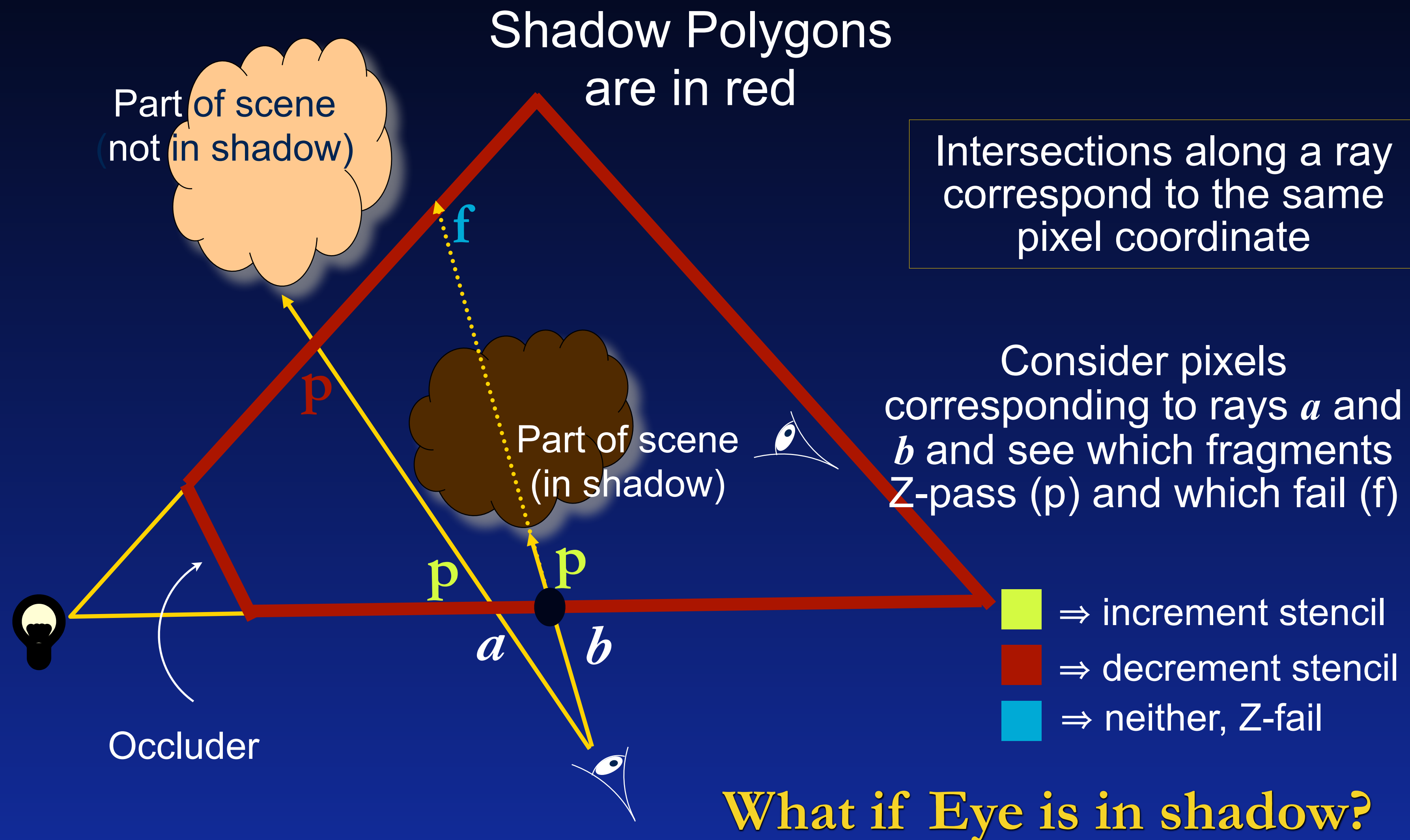
See `glFrontFace(..)`, `glCullFace(..)`, `gl_FrontFacing` (in FS)



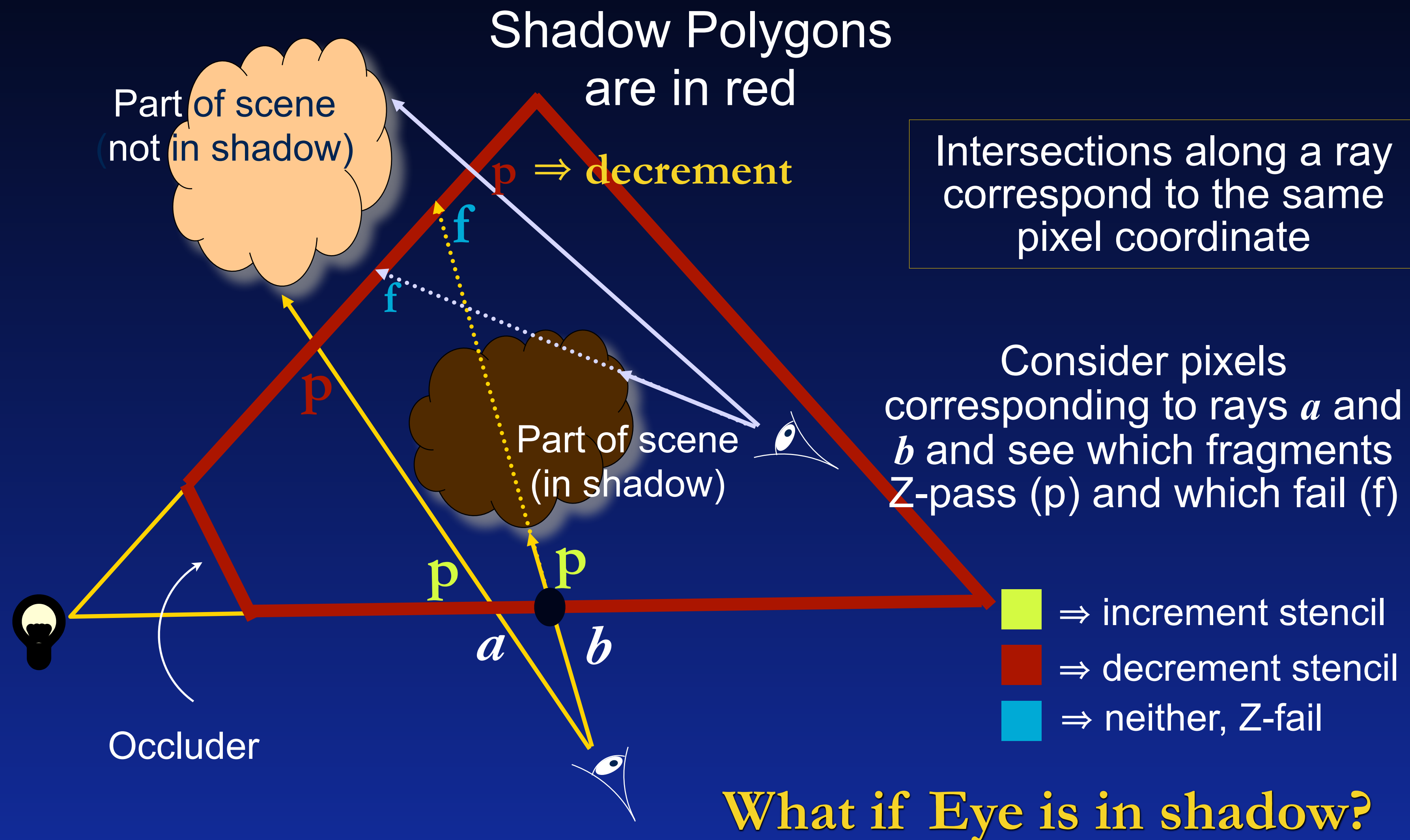
Stencil Generation

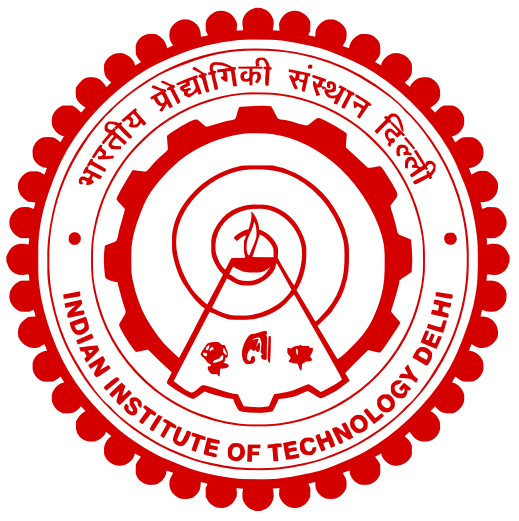
- Disable depth and color writes
- Set the stencil operation to increment on depth **pass** (counting shadow boundaries in front of the object)
- Draw the shadow volumes front faces
 - ➡ Enable back-face culling (*i.e.*, front drawing)
- Set the stencil operation to decrement on depth **pass**
- Draw the shadow volume back faces
 - ➡ Enable front-face culling

Stencil Shadow Volumes



Stencil Shadow Volumes

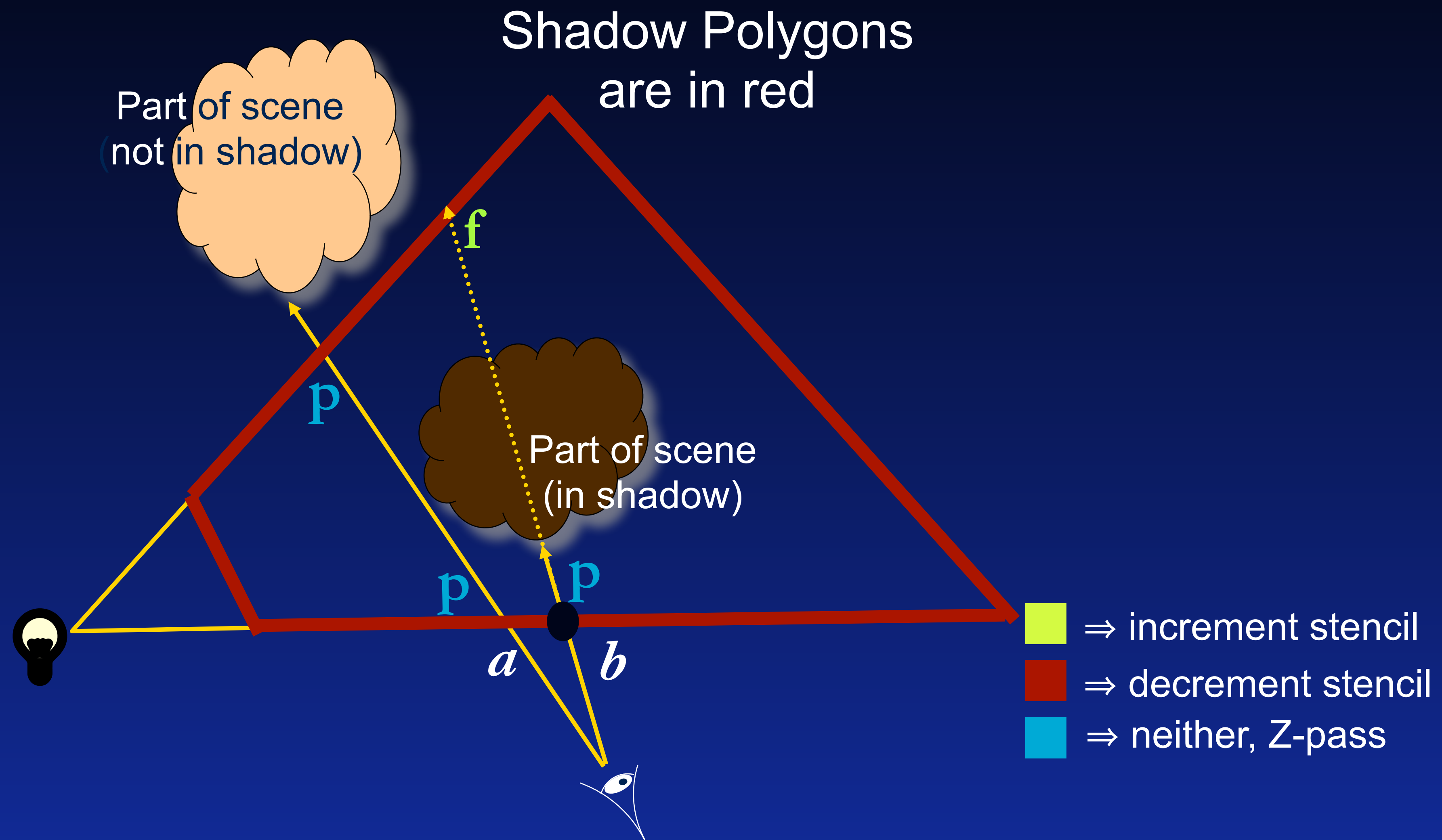




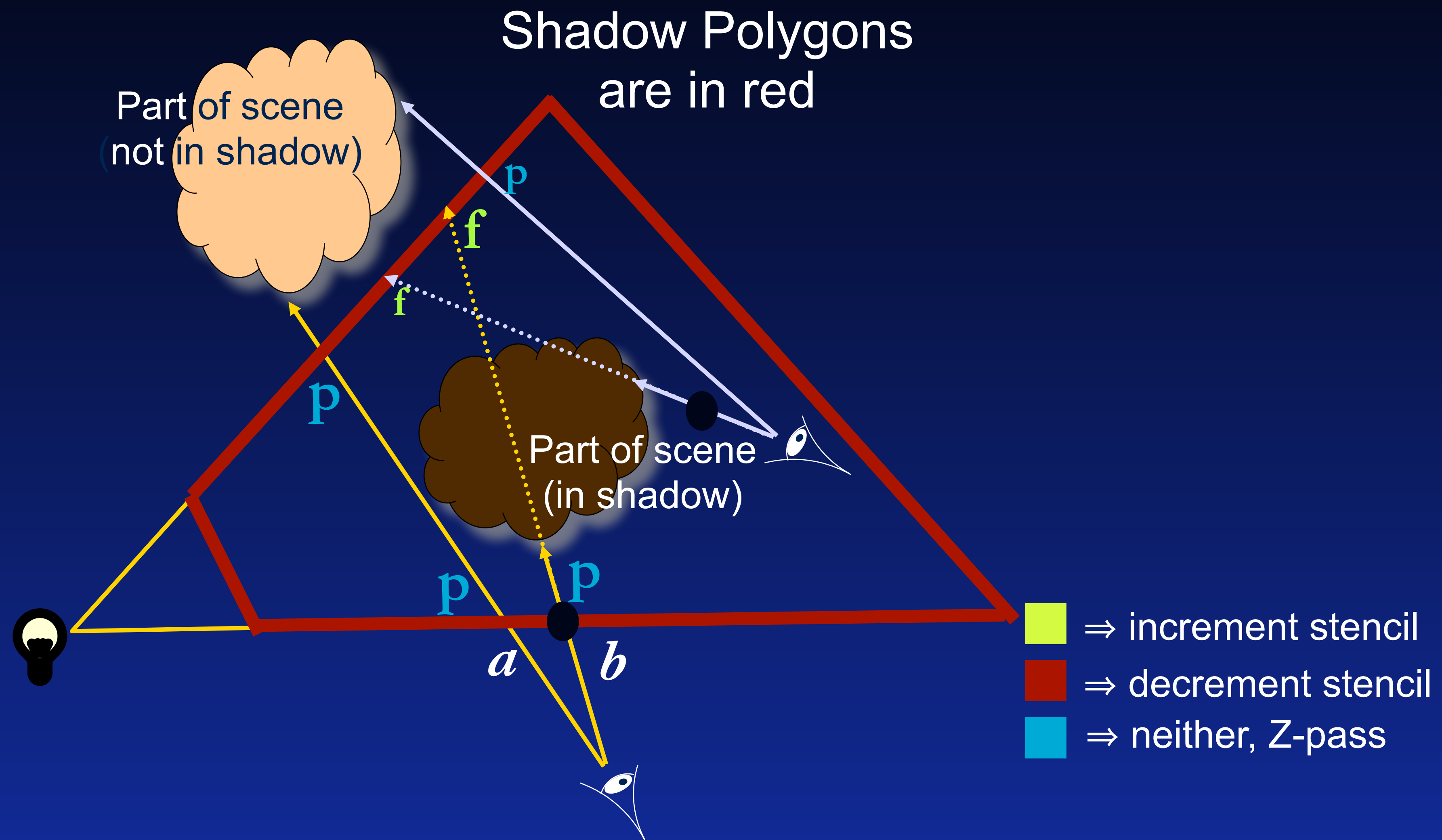
Reverse Stencil

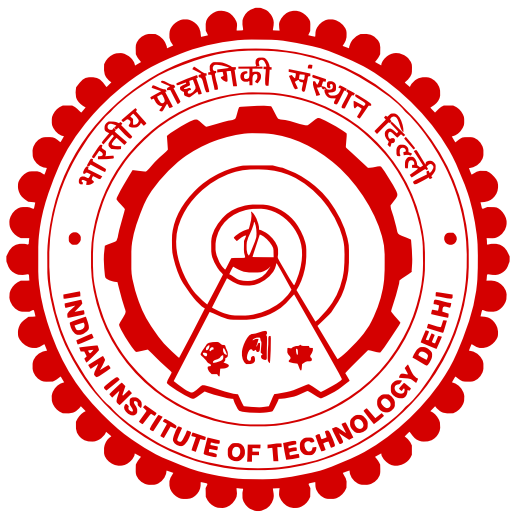
- Disable depth and color writes
- Set the stencil operation to increment on depth fail (counting shadow boundaries behind the object)
 - ➡ Render the shadow volumes' back faces
- Set the stencil operation to decrement on depth fail
 - ➡ Render the shadow volumes' front faces

Stencil Shadow Volumes



Stencil Shadow Volumes





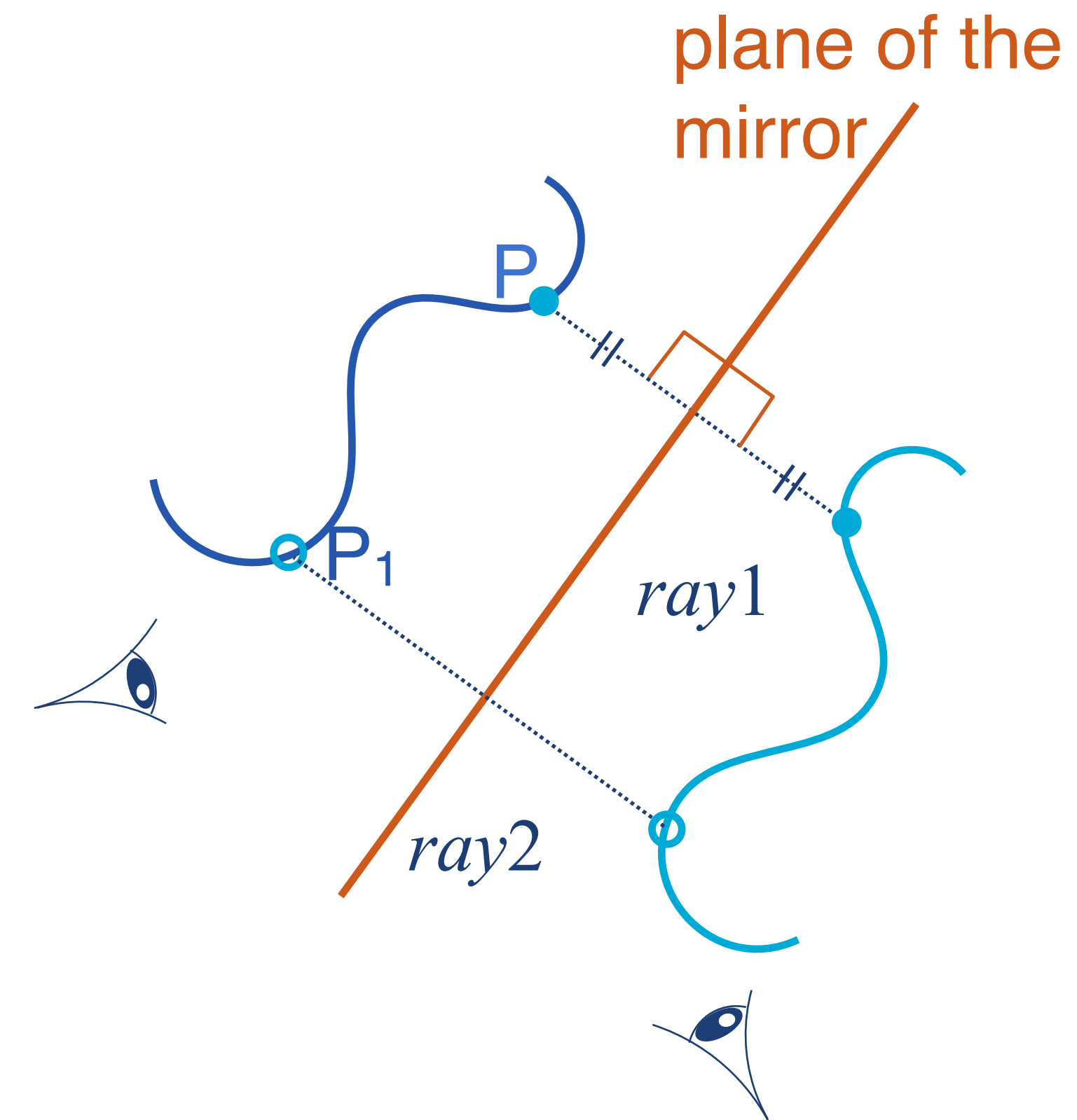
Stencil Generation

- Disable depth and color writes
- Set the stencil operation to increment on depth **pass** (counting shadow boundaries in front of the object)
- Draw the shadow volumes front faces
 - ➡ Enable back-face culling (*i.e.*, front drawing)
- Set the stencil operation to decrement on depth **pass**
- Draw the shadow volume back faces
 - ➡ Enable front-face culling



Reflection in Plane

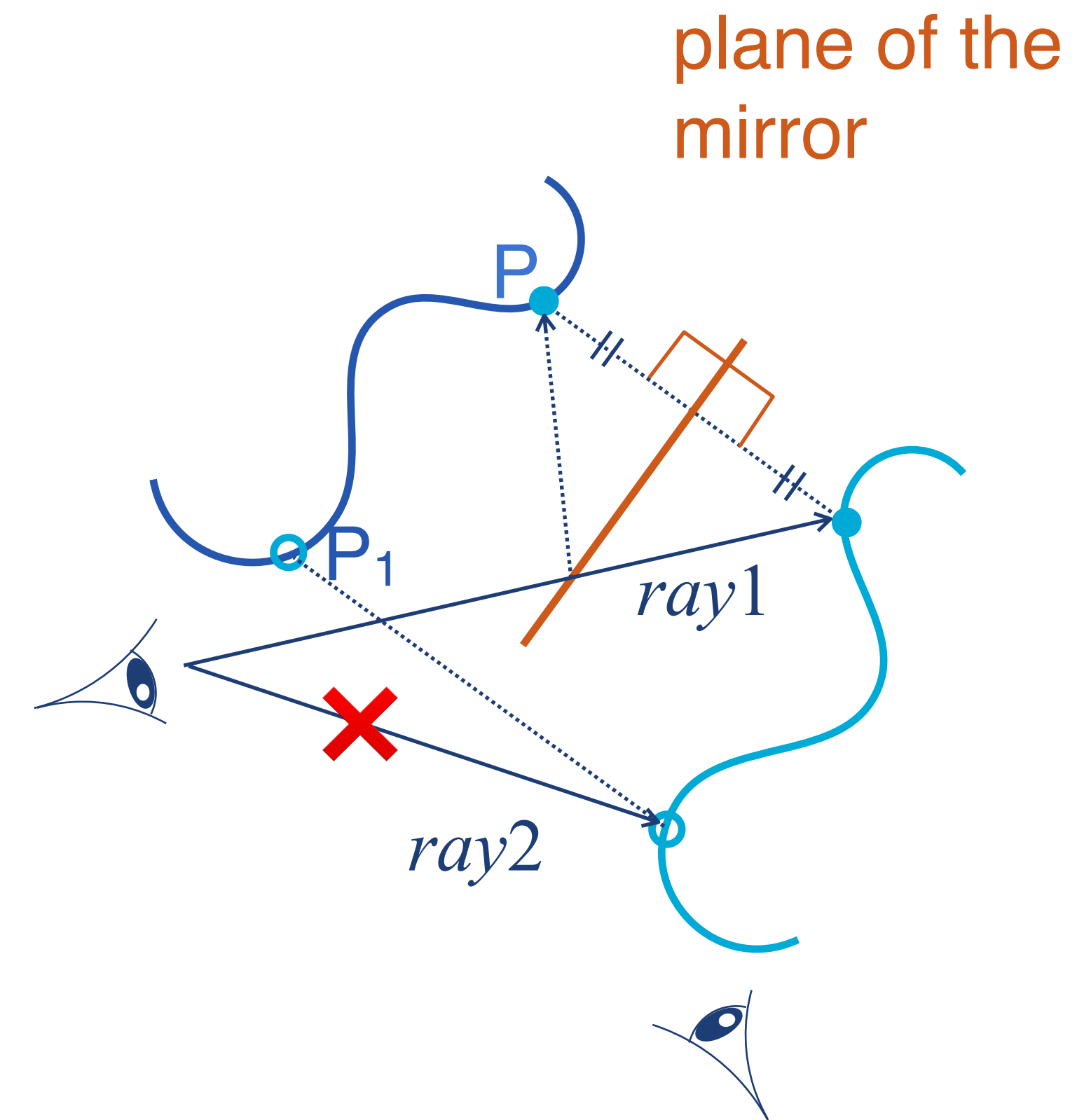
- Linear transformation
 - ➔ Review transformations
 - ➔ Specular reflection
 - Can be per fragment
- Two pass rendering
 - ➔ Render original geometry
 - ➔ Render reflected geometry
- But the mirror may be a triangle
 - ➔ Or a polygon
 - ➔ Assume designated reflectors





Reflection in Plane

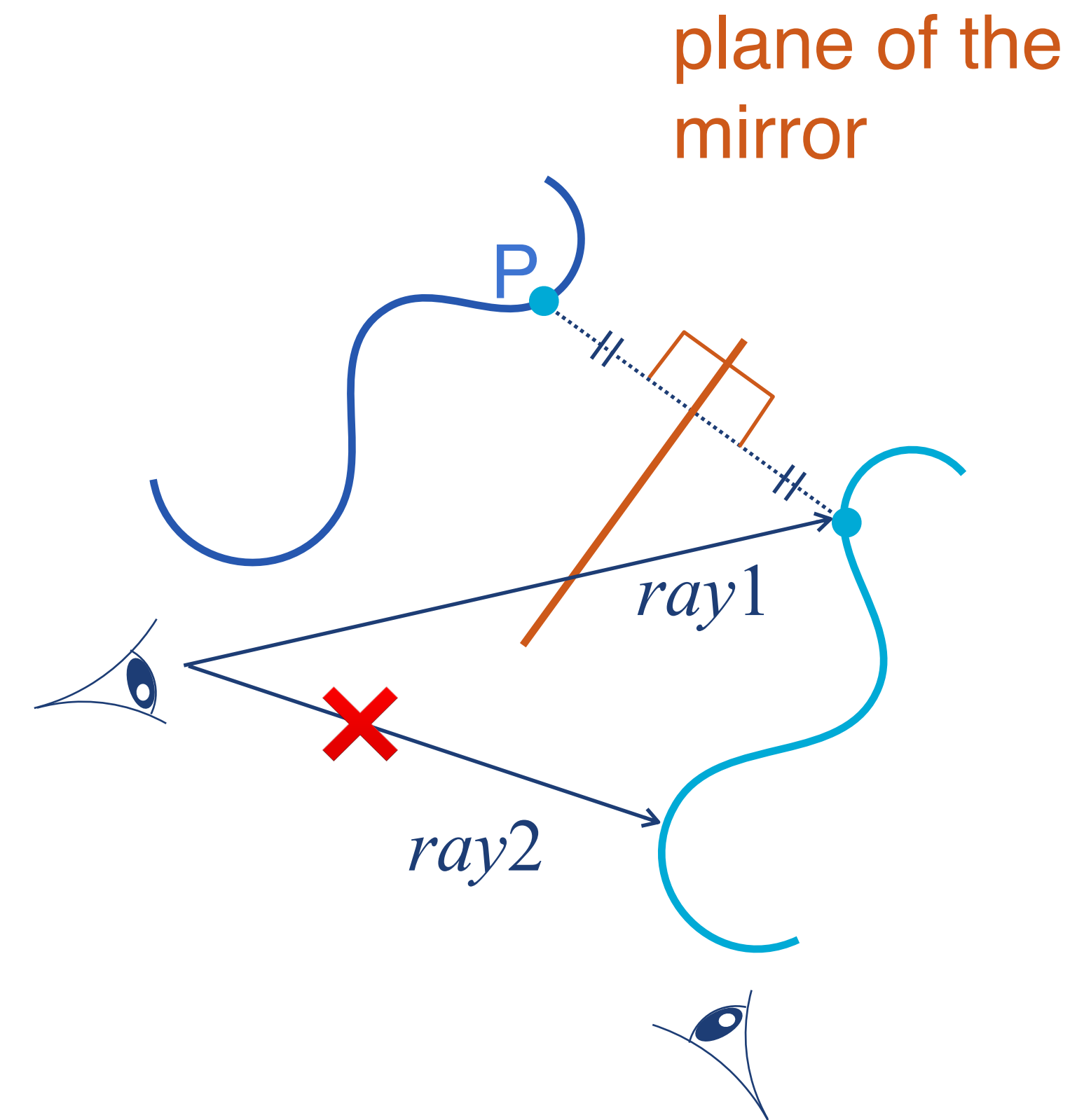
- Linear transformation
 - ➔ Review transformations
 - ➔ Specular reflection
 - Can be per fragment
- Two pass rendering
 - ➔ Render original geometry
 - ➔ Render reflected geometry
- But the mirror may be a triangle
 - ➔ Or a polygon
 - ➔ Assume designated reflectors

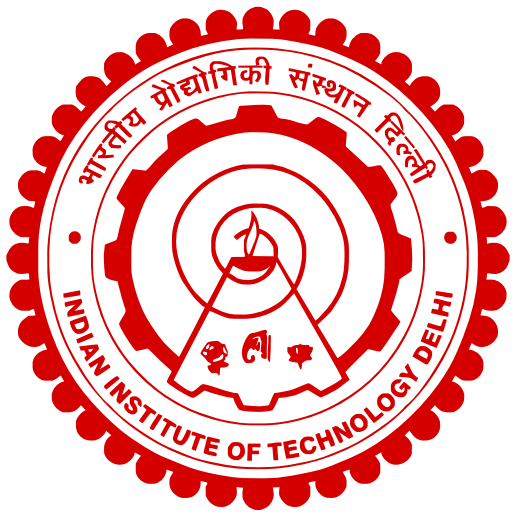




Reflection in Plane

- Linear transformation
 - ➔ Review transformations
 - ➔ Specular reflection
 - Can be per fragment
- Two pass rendering
 - ➔ Render original geometry
 - ➔ Render reflected geometry
- But the mirror may be a triangle
 - ➔ Or a polygon
 - ➔ Assume designated reflectors





Issues with Reflection

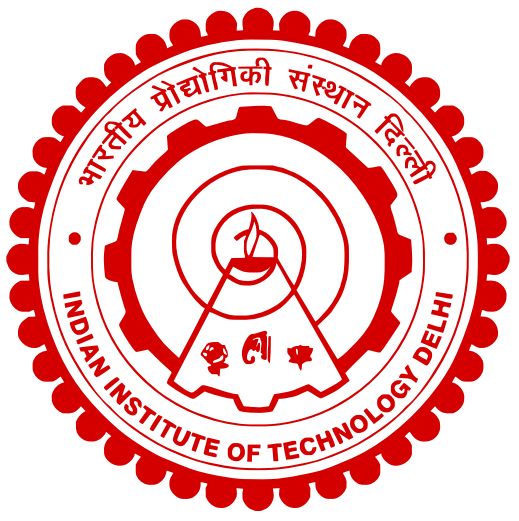
- No reflection drawing if Eye behind mirror
 - ➡ Client-side check
- Use `gl_ClipDistance` to clip against the mirror plane for the reflection's rendering
 - ➡ Object behind the mirror should not be reflected
- Invert `glFrontFace` To perform back-face culling
 - ➡ Polygon orientation changes on reflection
- Render the mirror after the reflections and before the objects (color to blend) for depth
 - ➡ Reflections behind the mirror should appear
 - ➡ Objects behind the mirror should not



Rendering Passes

1. Set up stenciling
2. Pre-multiply World matrix with Reflection matrix
3. Draw scene
 - ➡ These are the reflected objects
 - ▶ `glEnable(GL_CLIP_DISTANCE0); // client-side`
 - ▶ VS: `gl_ClipDistance[0] = dot(mirrorNormal, vertpos);`
`// In camera coordinates. Unit mirror normal`
4. Restore World matrix for VS
5. Draw mirror, write Z, disable color
 - ➡ `glColorMask( 0, 0, 0, 0 )`
 - ➡ Or set up alpha blending
6. Render rest of the scene (no mirror)
 - ➡ Restore `glColorMask(1, 1, 1, 1)`

Stencil Setup



1. Enable stenciling
2. Set stencil to always fail, but write 1 for each fragment (Color/Z also disabled)
3. Render the mirror normally
4. 1 in stencil means produce fragment, 0 means discard
5. Render scene with reflection matrix
6. Turn off stencil buffer
7. Render mirror again writing Z only
8. Render rest of the scene normally



Stencil Setup

```
glEnable(GL_STENCIL_TEST);  
glClearStencil( 0 );  
glClear( GL_STENCIL_BUFFER_BIT );
```

1. Enable stenciling
2. Set stencil to always fail, but write 1 for each fragment (Color/Z also disabled)
3. Render the mirror normally
4. 1 in stencil means produce fragment, 0 means discard
5. Render scene with reflection matrix
6. Turn off stencil buffer
7. Render mirror again writing Z only
8. Render rest of the scene normally



Stencil Setup

1. Enable stenciling
2. Set stencil to always fail, but write 1 for each fragment (Color/Z also disabled)
3. Render the mirror

```
glStencilFunc( GL_NEVER, 1, 1 ); // Always fails, Ref := 1  
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);  
// Write 1 on fail
```
4. 1 in stencil means mirror
5. Render scene with reflection matrix
6. Turn off stencil buffer
7. Render mirror again writing Z only
8. Render rest of the scene normally



Stencil Setup

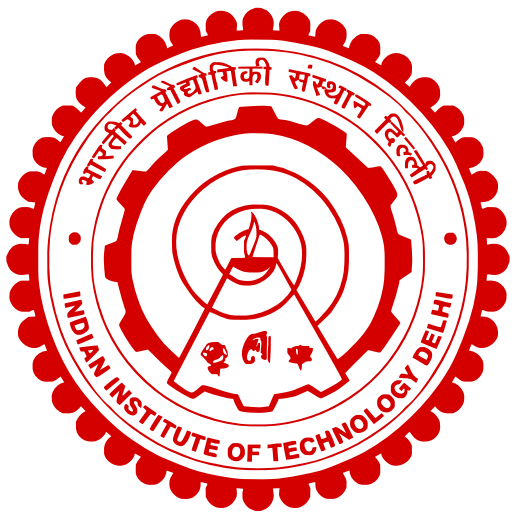
1. Enable stenciling
2. Set stencil to always fail, but write 1 for each fragment (Color/Z also disabled)
3. Render the mirror normally
4. 1 in stencil means produce fragment, 0 means discard
5. Render scene with reflection matrix

6. Turn off stencil buffer

```
glStencilFunc( GL_EQUAL, 1, 1 ); // Pass if SB == 1  
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);  
// Always retain old SB value
```

7. Render mirror again

8. Render rest of the scene normally

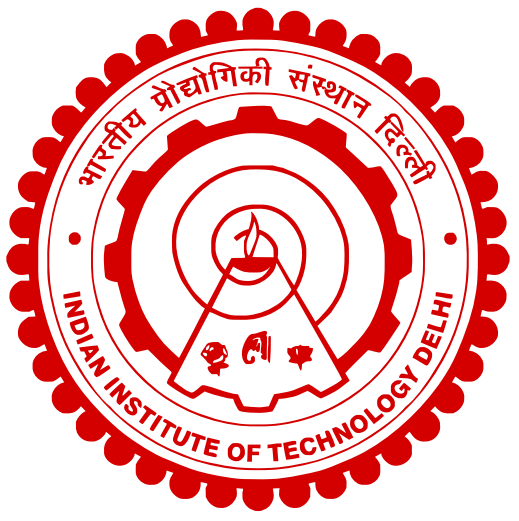


Stencil Setup

1. Enable stenciling
2. Set stencil to always fail, but write 1 for each fragment (Color/Z also disabled)
3. Render the mirror normally
4. 1 in stencil means produce fragment, 0 means discard
5. Render scene with reflection matrix
6. Turn off stencil buffer
7. Render mirror again writing Z only
8. Render rest of the scene normally

```
glDisable( GL_STENCIL_TEST );
```

Stencil Setup



1. Enable stenciling
2. Set stencil to always fail, but write 1 for each fragment (Color/Z also disabled)
3. Render the mirror normally
4. 1 in stencil means produce fragment, 0 means discard
5. Render scene with reflection matrix
6. Turn off stencil buffer
7. Render mirror again writing Z only
8. Render rest of the scene normally