Computer Graphics labs

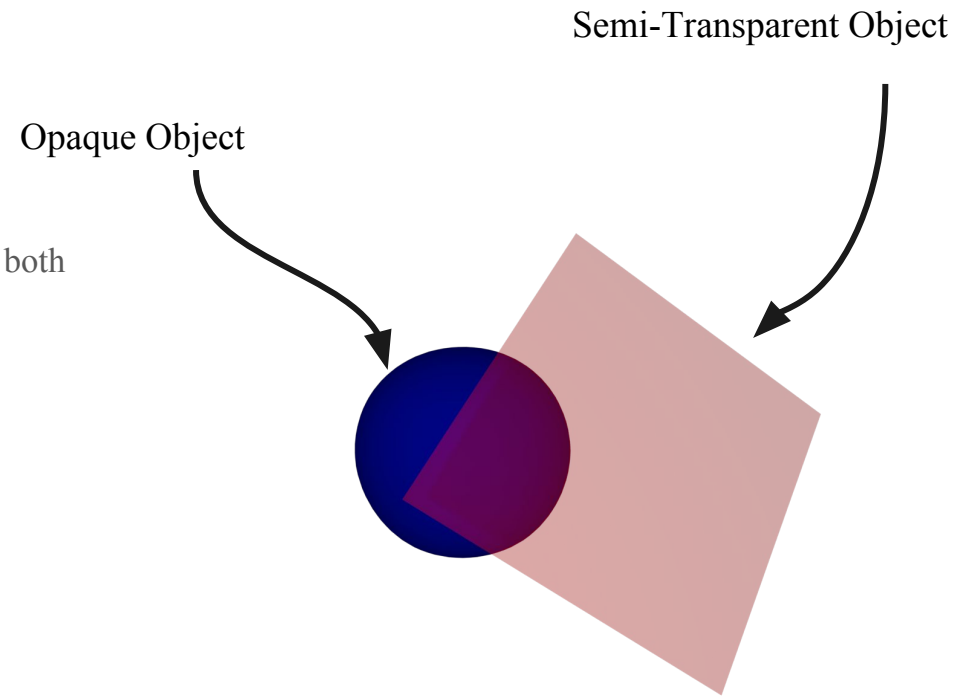
Lab 5 - Transparency and Blending

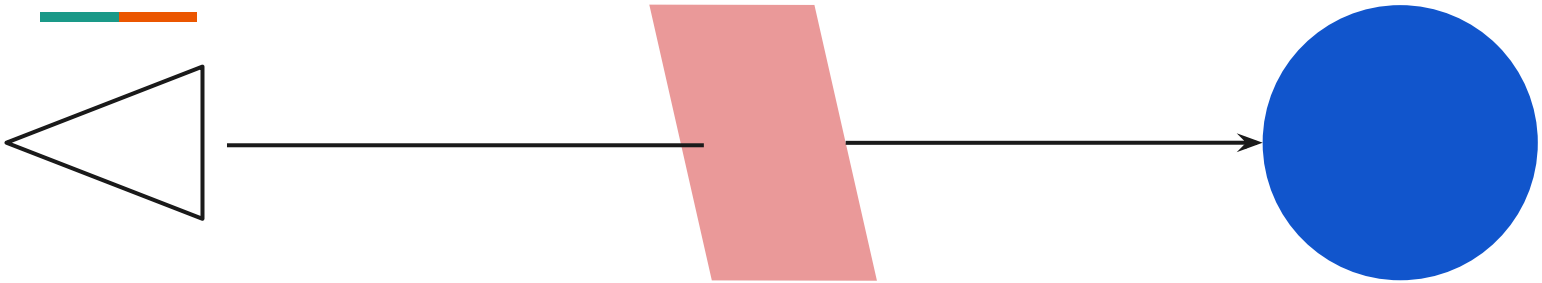
Up to this point we are dealing with opaque materials

Transparent Objects

Blending

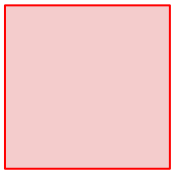
Transparency is like we want to see a bit of both objects.



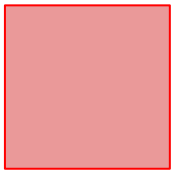


Full transparent

Alpha = 0



Alpha = 0.25



Alpha = 0.5

Alpha = 0.75

Fully opaque

Alpha = 1

Over Operator

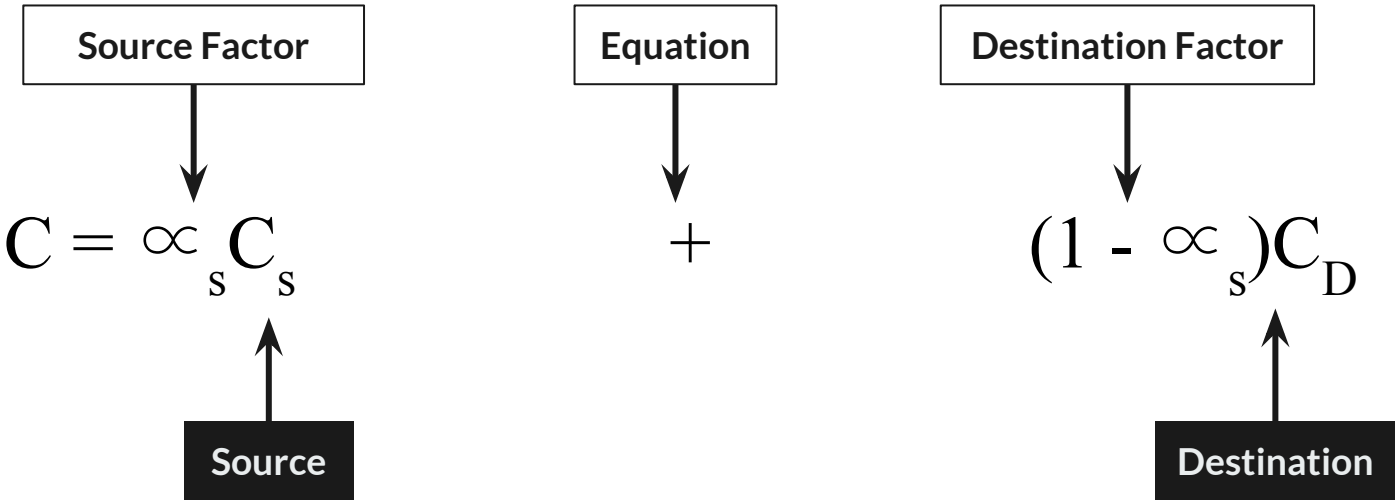
$$C = \propto_s C_s + (1 - \propto_s) C_D$$

$$C = C_D \qquad \text{When } \propto_s = 0$$

$$C = C_s \qquad \text{When } \propto_s = 1$$

Linear interpolation

OpenGL gives us the ability to control



Destination Factor

Source Factor

Could be:

GL_ZERO: ZERO
GL_ONE: ONE
GL_SRC_COLOR: SOURCE COLOR
GL_ONE_MINUS_SRC_COLOR: 1 - SOURCE COLOR
GL_DST_COLOR: DESTINATION COLOR
GL_ONE_MINUS_DST_COLOR: 1 -DESTINATION COLOR
GL_SRC_ALPHA: SOURCE ALPHA
GL_ONE_MINUS_SRC_ALPHA: 1 - SOURCE ALPHA
GL_DST_ALPHA: DESTINATION ALPHA
GL_ONE_MINUS_DST_ALPHA: 1 - DESTINATION ALPHA
GL_CONSTANT_COLOR: CONSTANT
GL_ONE_MINUS_CONSTANT_COLOR: 1 - CONSTANT COLOR
GL_CONSTANT_ALPHA: CONSTANT ALPHA
GL_ONE_MINUS_CONSTANT_ALPHA: 1 - CONSTANT ALPHA

Equation

Could be:

GL_FUNC_ADD: ADDITION
GL_FUNC_SUBTRACT: SUBTRACTION
GL_FUNC_REVERSE_SUBTRACT: REVERSE SUBTRACTION
GL_MIN: MIN
GL_MAX: MAX

The most common blending setup

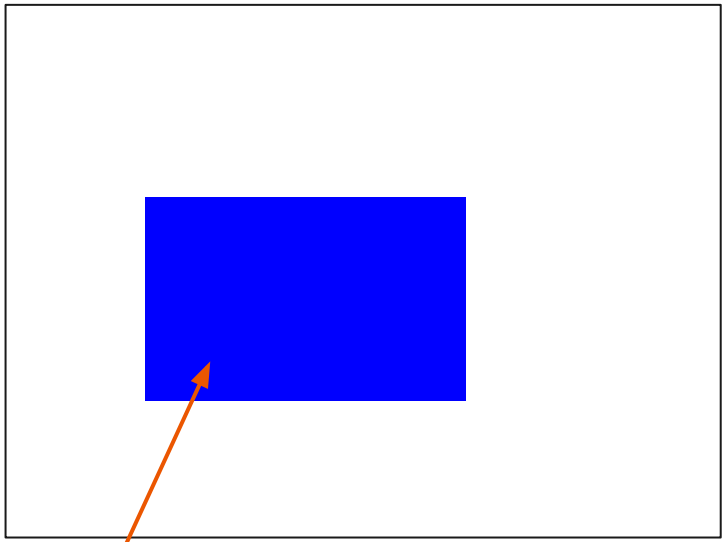
```
glEnable(GL_BLEND);
glBlendEquation(GL_FUNC_ADD);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Commands:

1. Draw Blue Rectangle (0, 0, 1, 0.5)
2. Draw Red Rectangle (1, 0, 0, 0.5)

Blending Setup:

- Equation: **GL_FUNC_ADD**
- Factors: **GL_SRC_ALPHA**, **GL_ONE_MINUS_SRC_ALPHA**



Source

(0, 0, 1, 0.5)

Destination

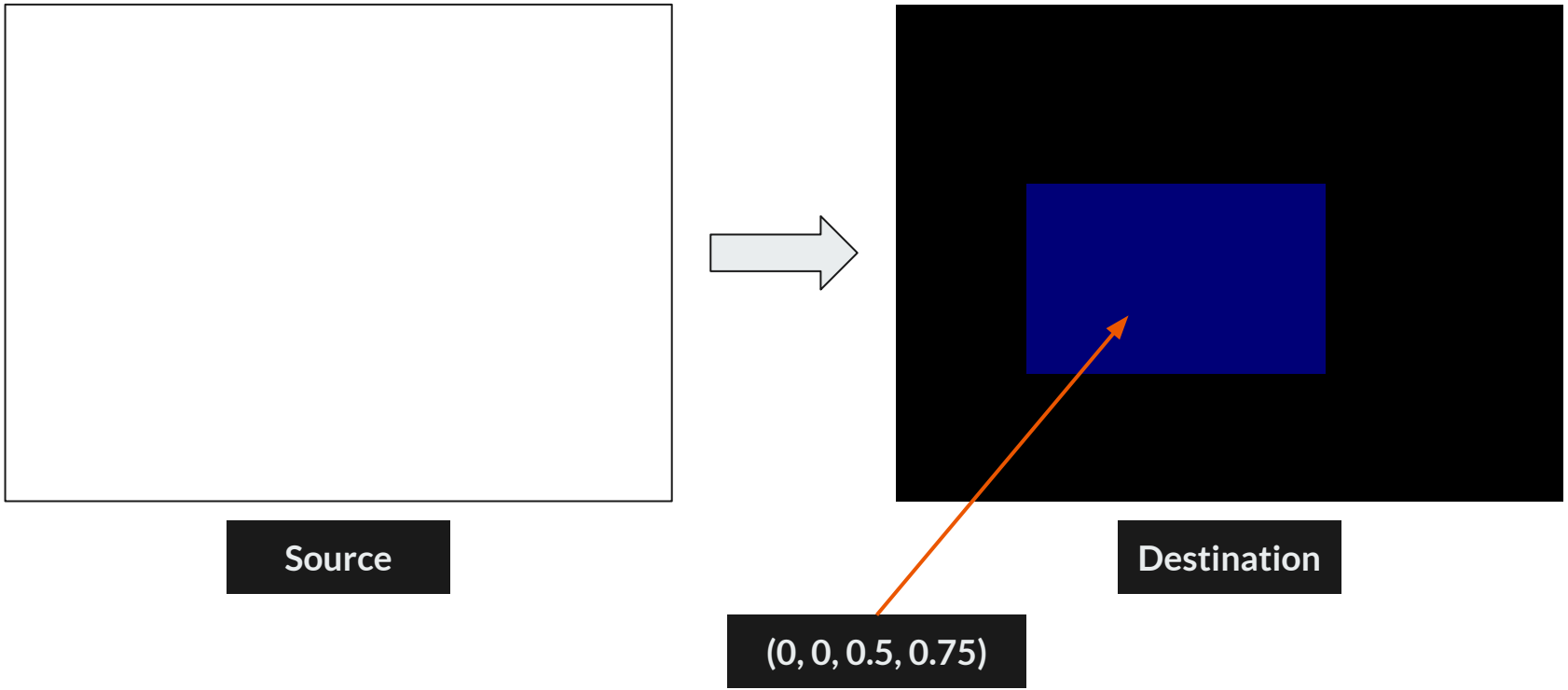
(0, 0, 0, 1)

Commands:

1. Draw Blue Rectangle (0, 0, 1, 0.5)
2. Draw Red Rectangle (1, 0, 0, 0.5)

Blending Setup:

- Equation: **GL_FUNC_ADD**
- Factors: **GL_SRC_ALPHA**, **GL_ONE_MINUS_SRC_ALPHA**

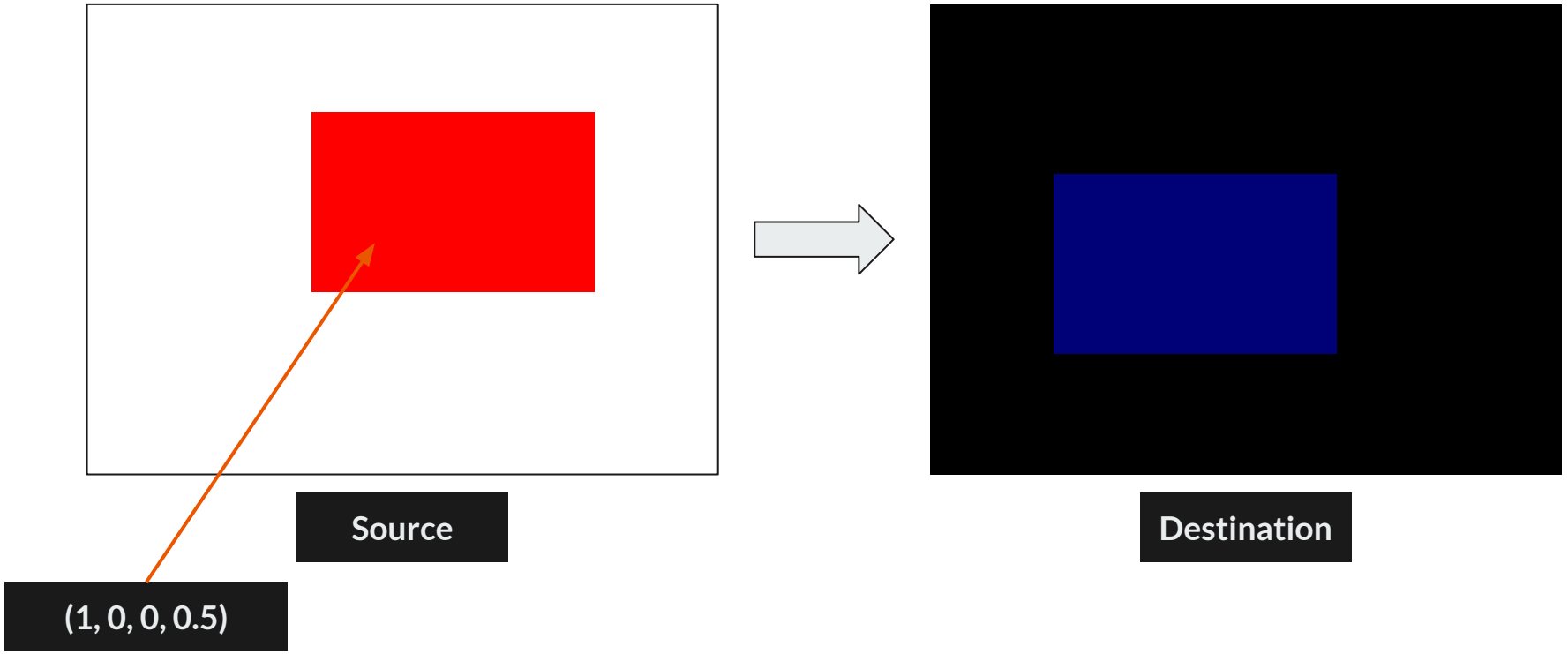


Commands:

1. Draw Blue Rectangle (0, 0, 1, 0.5)
   Draw Red Rectangle (1, 0, 0, 0.5)

Blending Setup:

- Equation: **GL_FUNC_ADD**
- Factors: **GL_SRC_ALPHA**, **GL_ONE_MINUS_SRC_ALPHA**

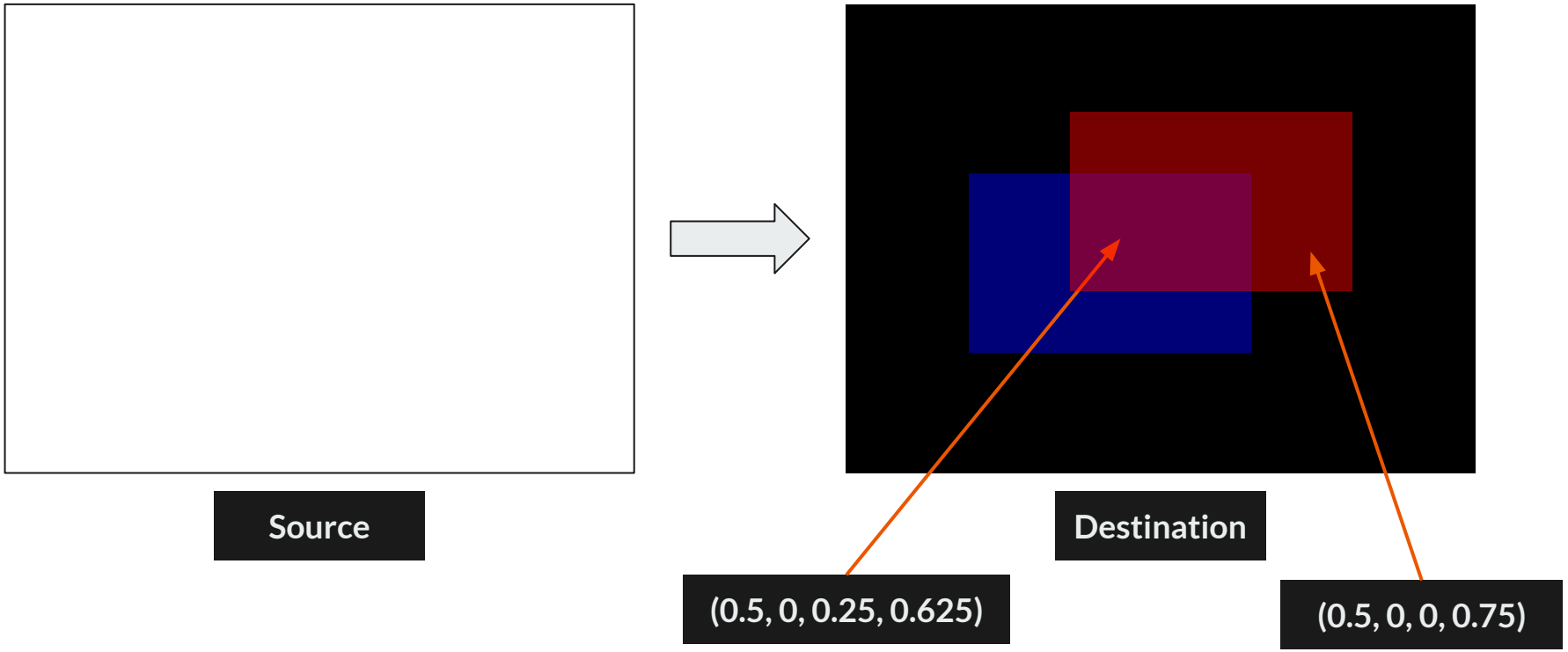


Commands:

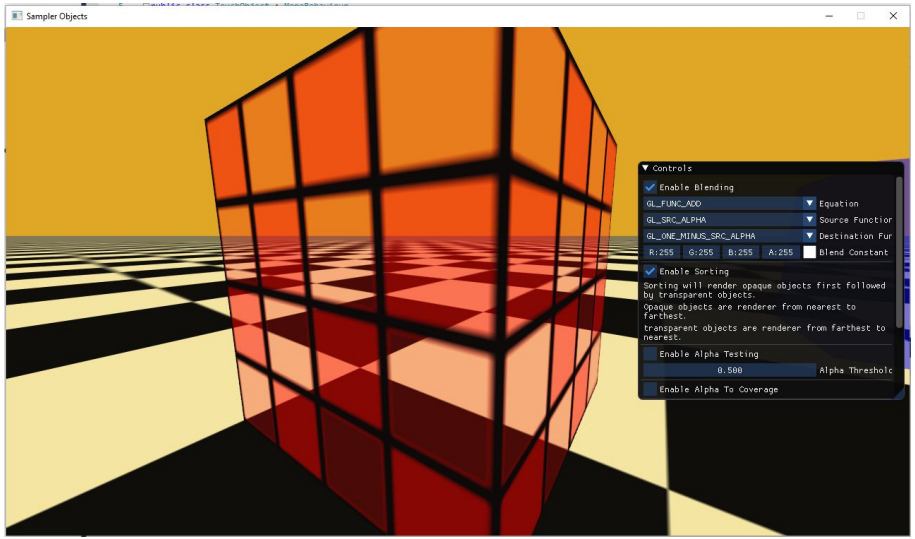
1. Draw Blue Rectangle (0, 0, 1, 0.5)
   Draw Red Rectangle (1, 0, 0, 0.5)

Blending Setup:

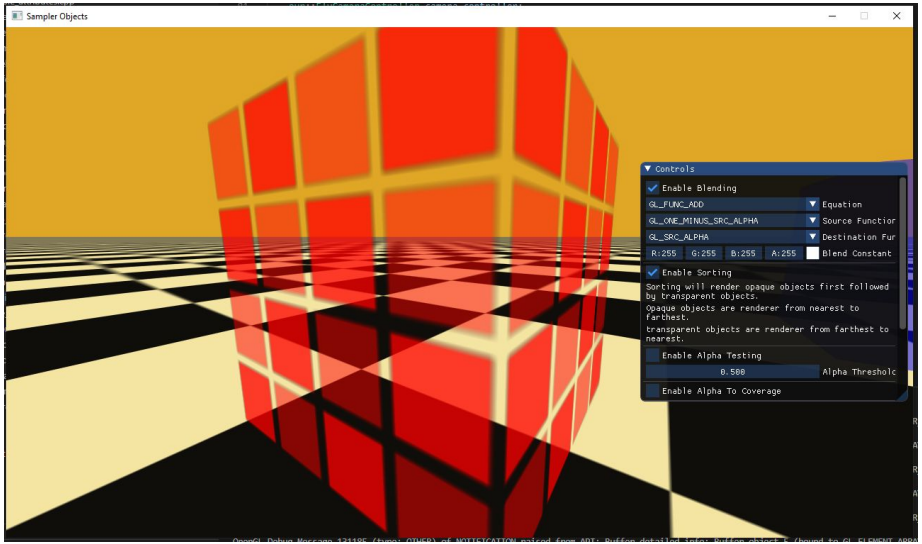
- Equation: **GL_FUNC_ADD**
- Factors: **GL_SRC_ALPHA**, **GL_ONE_MINUS_SRC_ALPHA**



(SOURCE ALPHA) **ADD** *(1 - SOURCE ALPHA)*



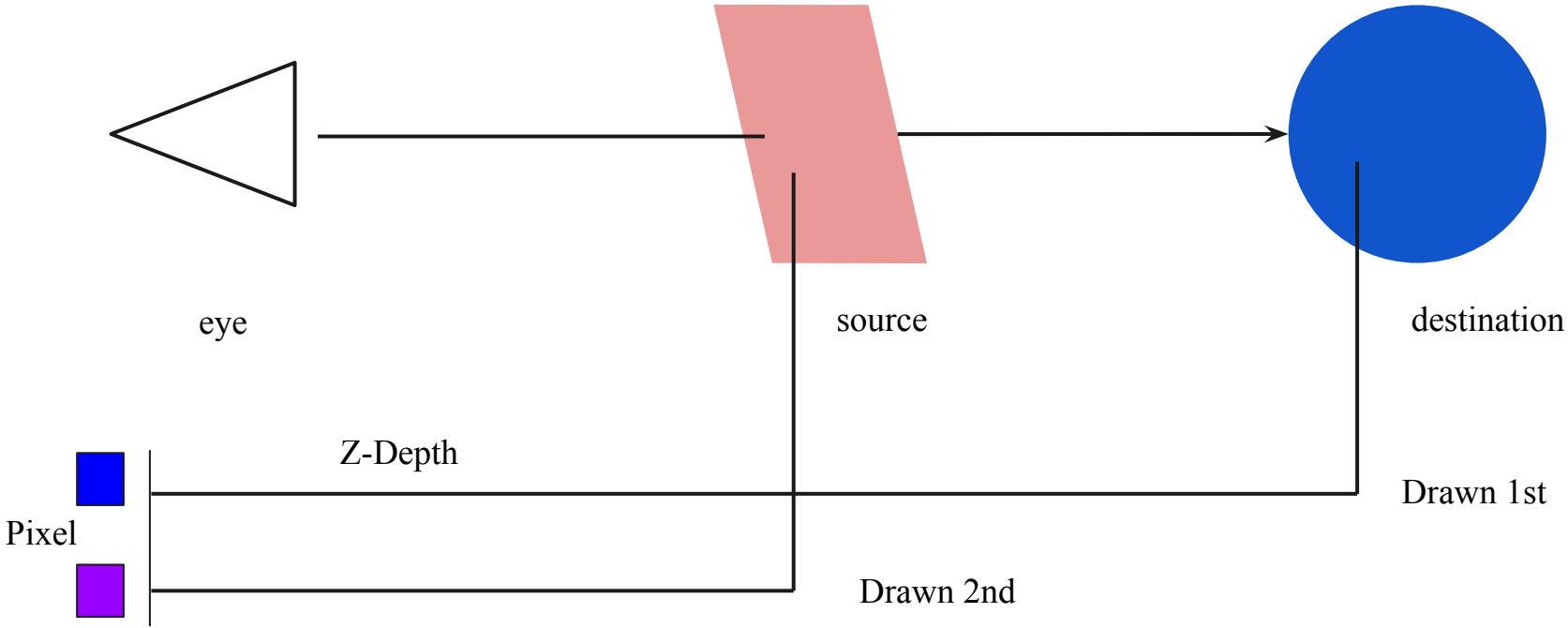
(1 - SOURCE ALPHA) **ADD** *(SOURCE ALPHA)*



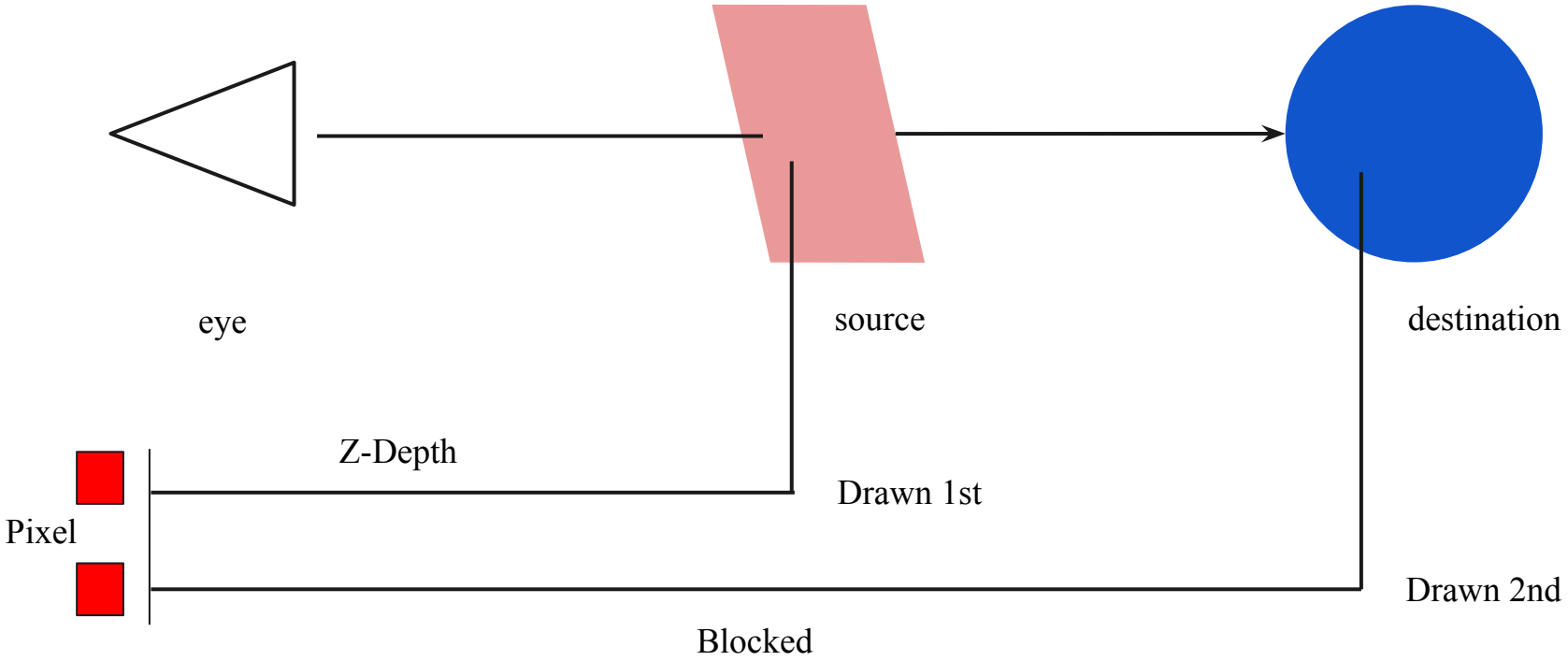
And many other combinations

Z-Buffer and transparency

Let us consider this drawing order scenario

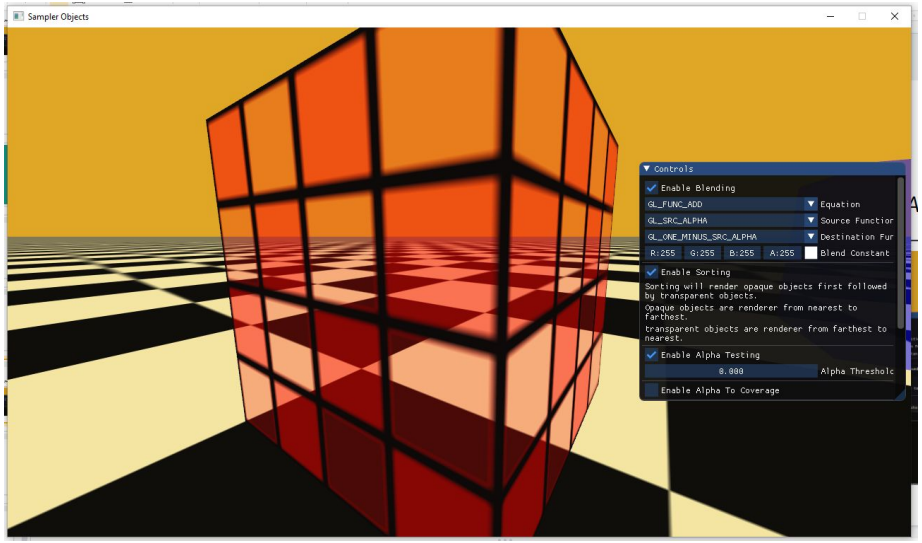


But what if


eye
source
destination
Z-Depth
Drawn 1st
Pixel
Drawn 2nd
Blocked

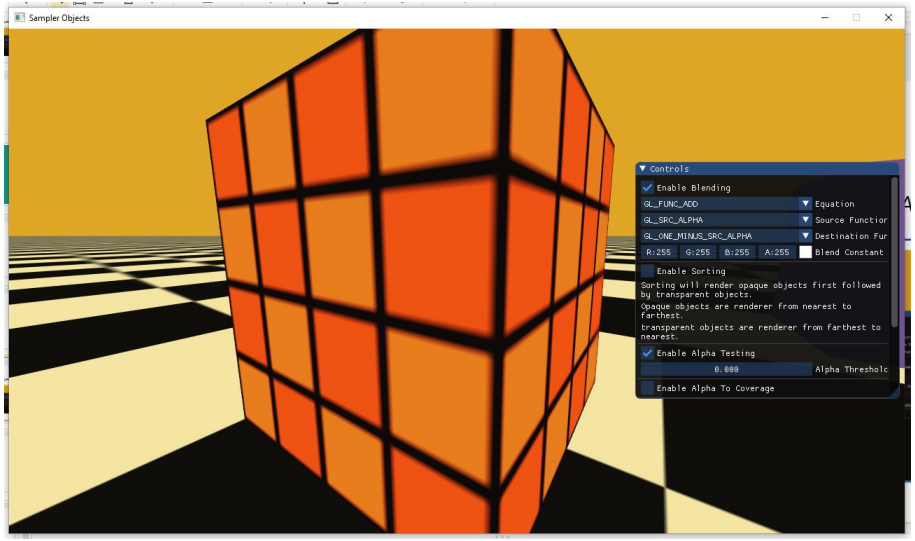
With sorting

The ground is drawn first to enable blending



Without sorting

The cube is drawn first blocking the ground



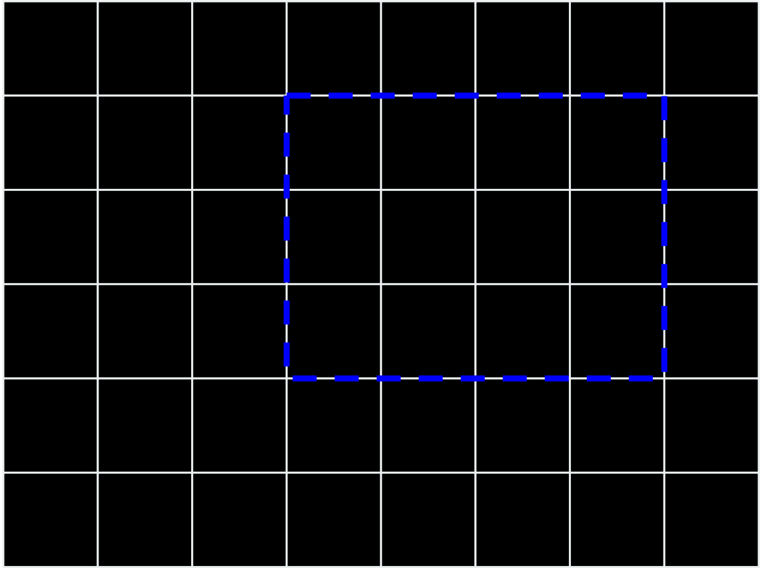
Commands:

Draw Blue Rectangle (0, 0, 1, 0.5) at Depth 0.9

2. Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.2
3. Draw Green Rectangle (0, 1, 0, 0.5) at Depth 0.5

1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

Depth Buffer



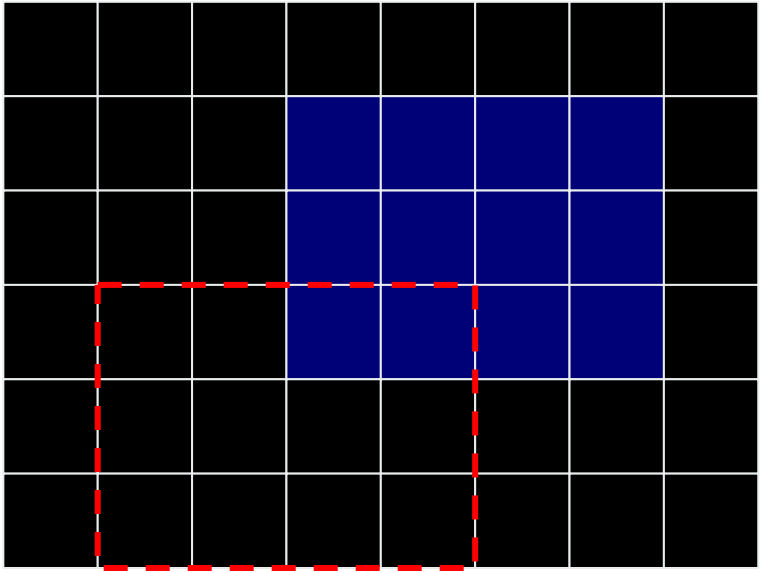
Color Buffer

Commands:

1. Draw Blue Rectangle (0, 0, 1, 0.5) at Depth 0.9
   Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.2
3. Draw Green Rectangle (0, 1, 0, 0.5) at Depth 0.5

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	0.9	0.9	0.9	0.9	1.0
1.0	1.0	1.0	0.9	0.9	0.9	0.9	1.0
1.0	1.0	1.0	0.9	0.9	0.9	0.9	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Depth Buffer



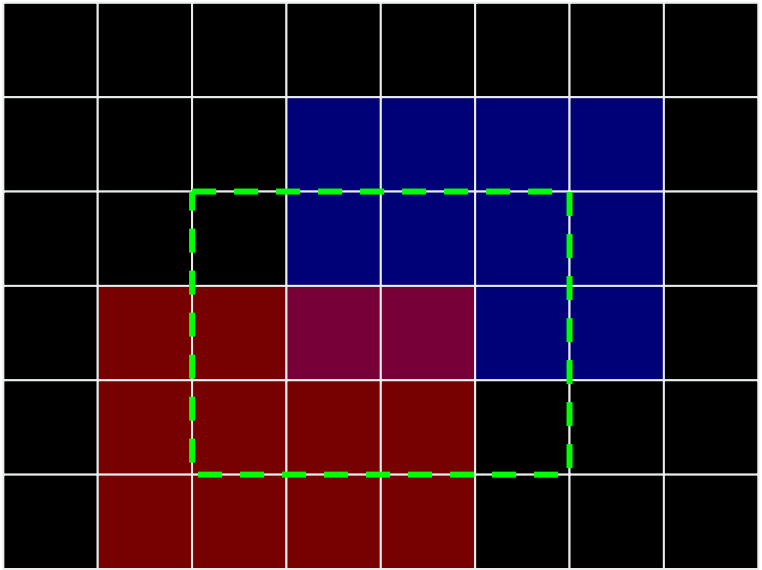
Color Buffer

Commands:

1. Draw Blue Rectangle (0, 0, 1, 0.5) at Depth 0.9
2. Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.2
   Draw Green Rectangle (0, 1, 0, 0.5) at Depth 0.5

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	0.9	0.9	0.9	0.9	1.0
1.0	1.0	1.0	0.9	0.9	0.9	0.9	1.0
1.0	0.2	0.2	0.2	0.2	0.9	0.9	1.0
1.0	0.2	0.2	0.2	0.2	1.0	1.0	1.0
1.0	0.2	0.2	0.2	0.2	1.0	1.0	1.0

Depth Buffer



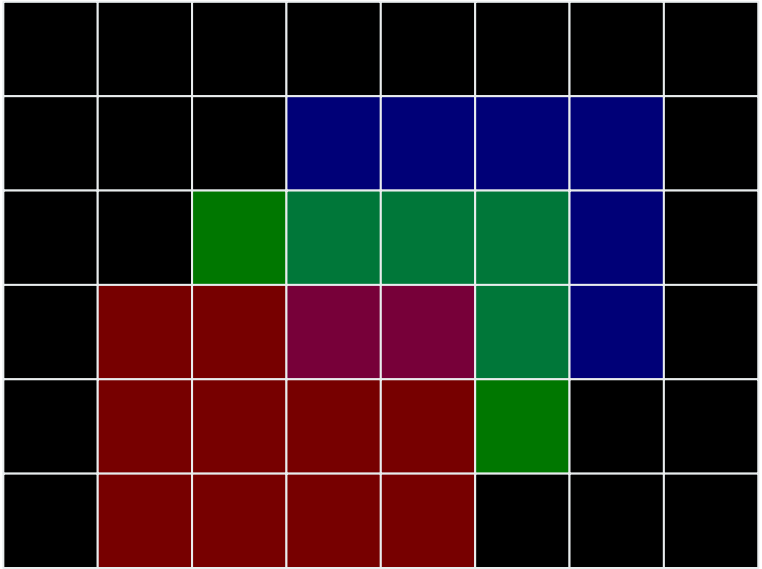
Color Buffer

Commands:

1. Draw Blue Rectangle (0, 0, 1, 0.5) at Depth 0.9
2. Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.2
3. Draw Green Rectangle (0, 1, 0, 0.5) at Depth 0.5

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	0.9	0.9	0.9	0.9	1.0
1.0	1.0	0.5	0.5	0.5	0.5	0.9	1.0
1.0	0.2	0.2	0.2	0.2	0.5	0.9	1.0
1.0	0.2	0.2	0.2	0.2	0.5	1.0	1.0
1.0	0.2	0.2	0.2	0.2	1.0	1.0	1.0

Depth Buffer



Color Buffer

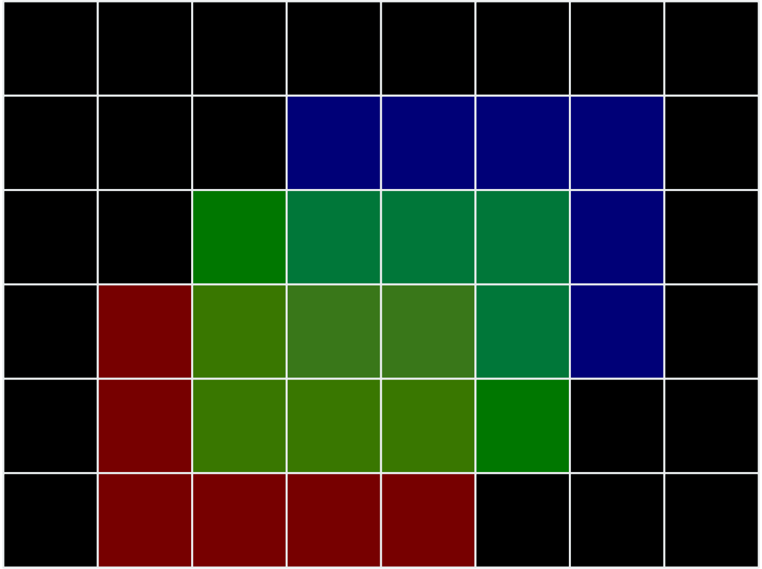
Commands: (**NO DEPTH TESTING**)

1. Draw Blue Rectangle (0, 0, 1, 0.5) at Depth 0.9
2. Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.2
3. Draw Green Rectangle (0, 1, 0, 0.5) at Depth 0.5

Incorrect Result (Order Matters)

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Depth Buffer



Color Buffer

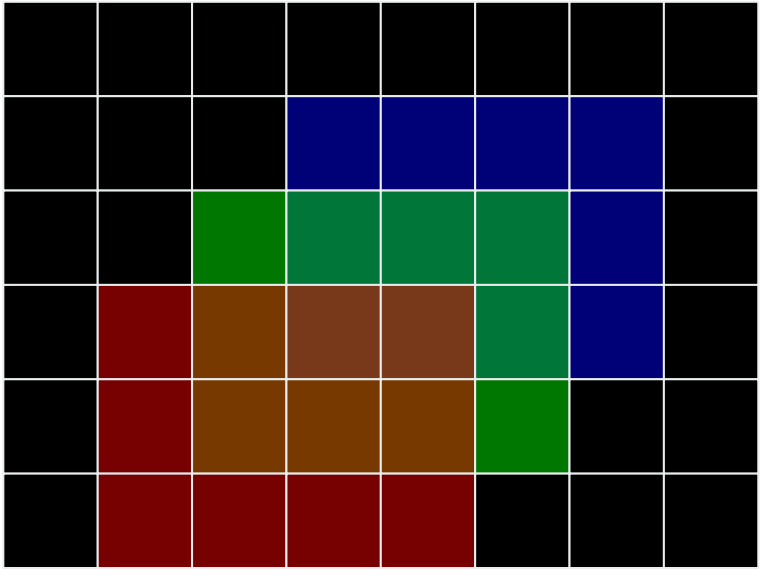
Commands **(Sorted)**:

1. Draw Blue Rectangle (0, 0, 1, 0.5) at Depth 0.9
2. Draw Green Rectangle (0, 1, 0, 0.5) at Depth 0.5
3. Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.2

Correct Result

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	0.9	0.9	0.9	0.9	1.0
1.0	1.0	0.5	0.5	0.5	0.5	0.9	1.0
1.0	0.2	0.2	0.2	0.2	0.5	0.9	1.0
1.0	0.2	0.2	0.2	0.2	0.5	1.0	1.0
1.0	0.2	0.2	0.2	0.2	1.0	1.0	1.0

Depth Buffer



Color Buffer

So now we are back to the Painter’s algorithm

Painter's Algorithm

Sort from Farthest to Nearest.

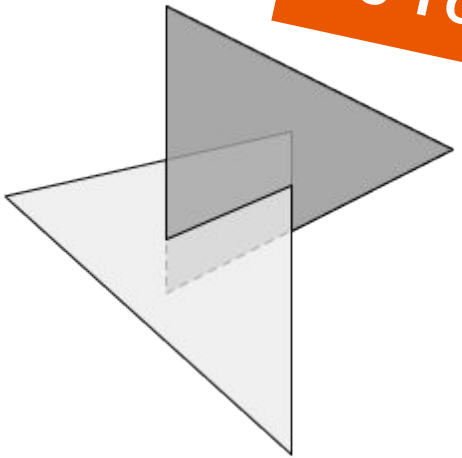
PROS:

- Memory Efficient.
- Still popular for transparent geometry till today.

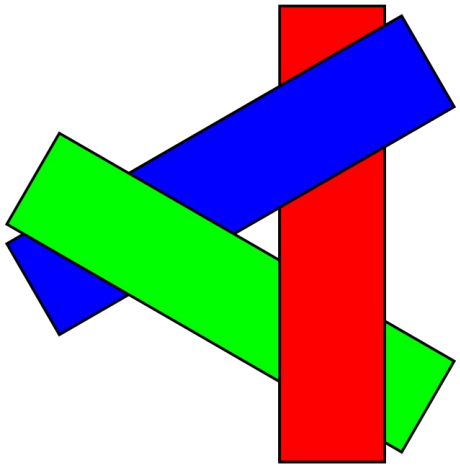
CONS:

- Can be computationally expensive.
- Fails for intersecting and cyclically-overlapping geometry.

Do You Remember?



Intersecting



Cyclical Overlapping

Common Handling of Transparency in Games

Draw Opaque Objects first and:

- Use Depth Buffer to resolve depth among each other.
- Prefer to draw to nearest to farthest to decrease overdraw. (Optional)

Then draw Transparent Objects and:

- Use Depth Buffer to resolve depth with the Opaque Objects.
- Strictly draw from farthest to nearest.

Other solutions

1. Use a blending setup that doesn't care about order (such as Multiplicative Blending) and disable depth testing. But it can only represent certain types of transparent objects.
2. Use depth peeling. [Expensive]
3. Use Order Independent Transparency via per-pixel linked-list sorting. [Expensive]
4. **Use Alpha Testing.** But fragments can either be fully opaque or fully transparent.
5. Use Screen-Door Transparency (Dithering). But it looks bad at low resolution and needs special handling for multiple layers of transparent objects.
6. Use Alpha to Coverage. But it only works when MSAA is enabled, can cause banding and needs special handling for multiple layers of transparent objects.
7. Stochastic Transparency. But it requires either MSAA or temporal AA.

Alpha testing

Another way to allow transparency

Is to discard pixels

Discard

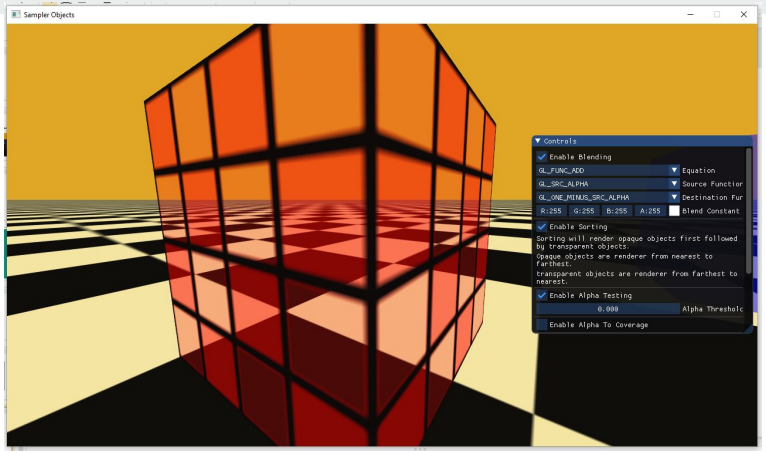
Discard pixels prevents them from being drawn.

By increasing the value of the threshold, the amount of drawn pixels decreases.

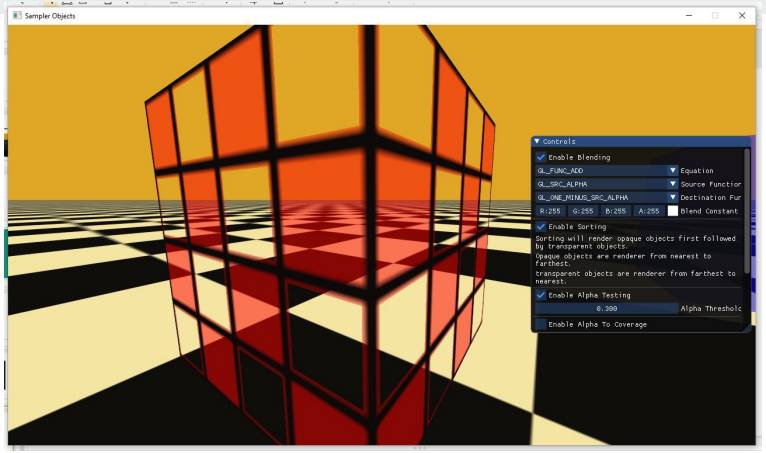
assets > shaders > ex25_blending > alpha_test.frag

```
1   #version 330 core
2
3   in Varyings {
4       vec4 color;
5       vec2 tex_coord;
6   } fsin;
7
8   uniform vec4 tint;
9   uniform sampler2D sampler;
10  uniform float alpha_threshold;
11
12  out vec4 frag_color;
13
14  void main() {
15      vec4 color = tint * fsin.color * texture(sampler, fsin.tex_coord);
16      if(color.a < alpha_threshold) discard;
17      frag_color = color;
18  }
19
```

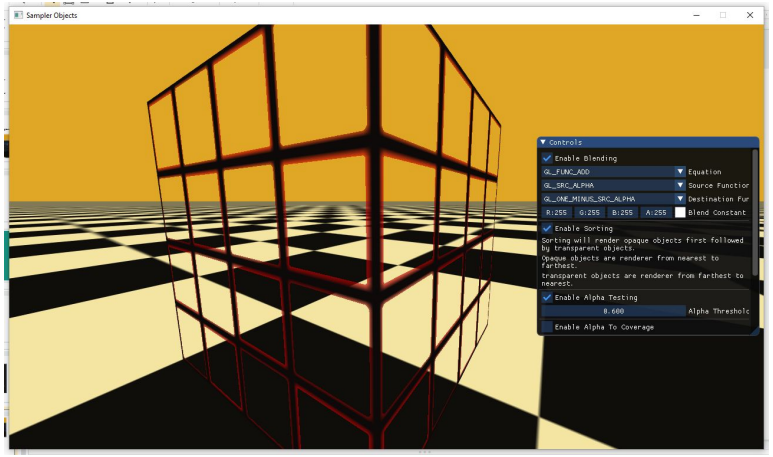
Threshold = 0



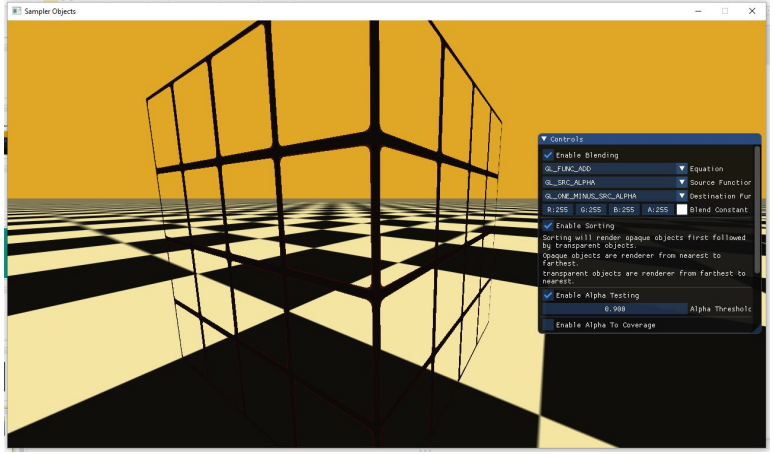
Threshold = 0.3



Threshold = 0.6



Threshold = 0.95



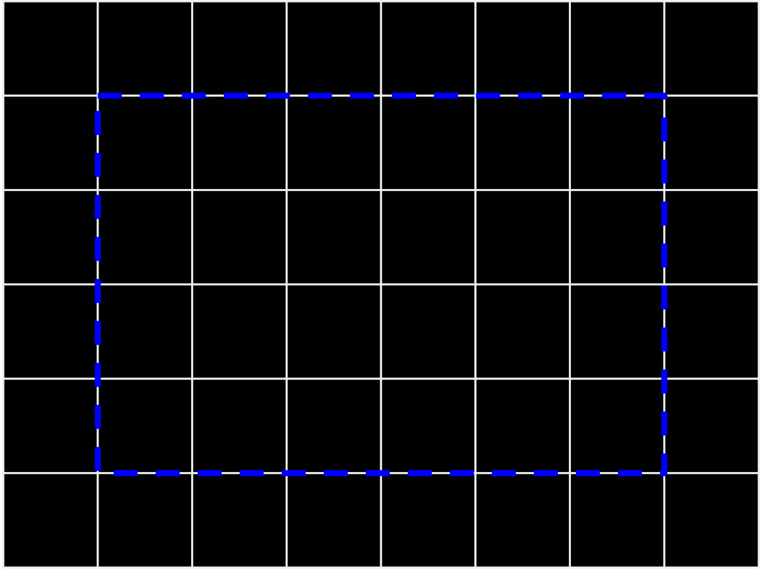
Commands:

Draw Alpha-Tested Checkerboard Rectangle (0, 0, 1, 1) & (0, 0, 1, 0) at Depth 0.2

2. Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.9

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Depth Buffer



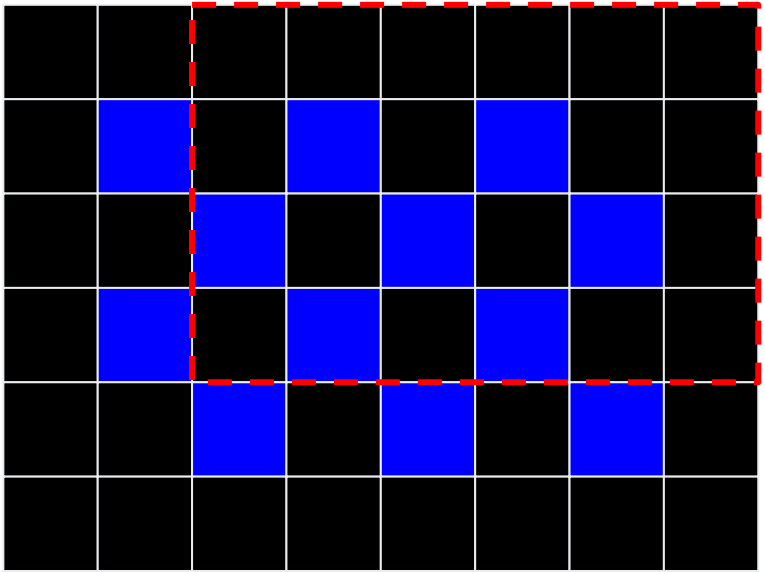
Color Buffer

Commands:

1. Draw Alpha-Tested Checkerboard Rectangle (0, 0, 1, 1) & (0, 0, 1, 0) at Depth 0.2
   Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.5

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	0.2	1.0	0.2	1.0	0.2	1.0	1.0
1.0	1.0	0.2	1.0	0.2	1.0	0.2	1.0
1.0	0.2	1.0	0.2	1.0	0.2	1.0	1.0
1.0	1.0	0.2	1.0	0.2	1.0	0.2	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Depth Buffer



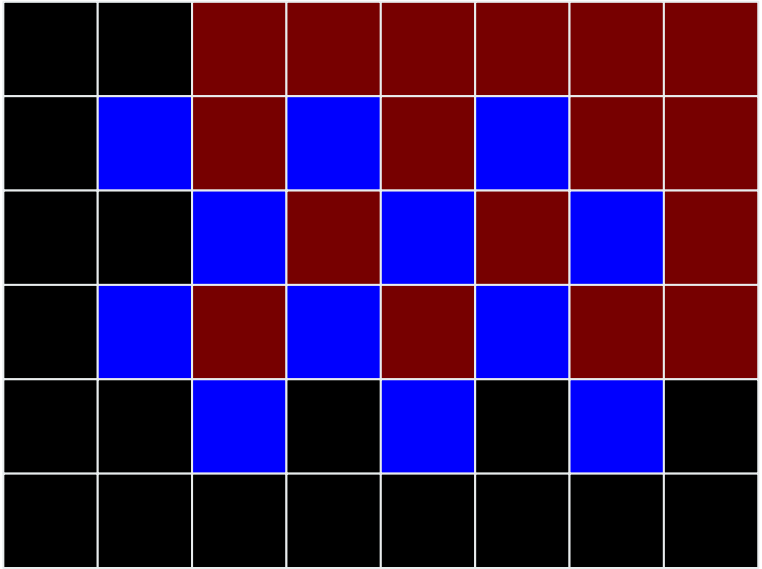
Color Buffer

Commands:

1. Draw Alpha-Tested Checkerboard Rectangle (0, 0, 1, 1) & (0, 0, 1, 0) at Depth 0.2
2. Draw Red Rectangle (1, 0, 0, 0.5) at Depth 0.5

1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.5
1.0	0.2	0.5	0.2	0.5	0.2	0.5	0.5
1.0	1.0	0.2	0.5	0.2	0.5	0.2	0.5
1.0	0.2	0.5	0.2	0.5	0.2	0.5	0.5
1.0	1.0	0.2	1.0	0.2	1.0	0.2	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Depth Buffer



Color Buffer

While transparency is simple as a concept and common in most applications, it is still a technical challenge till nowadays.

Thank you