Blending

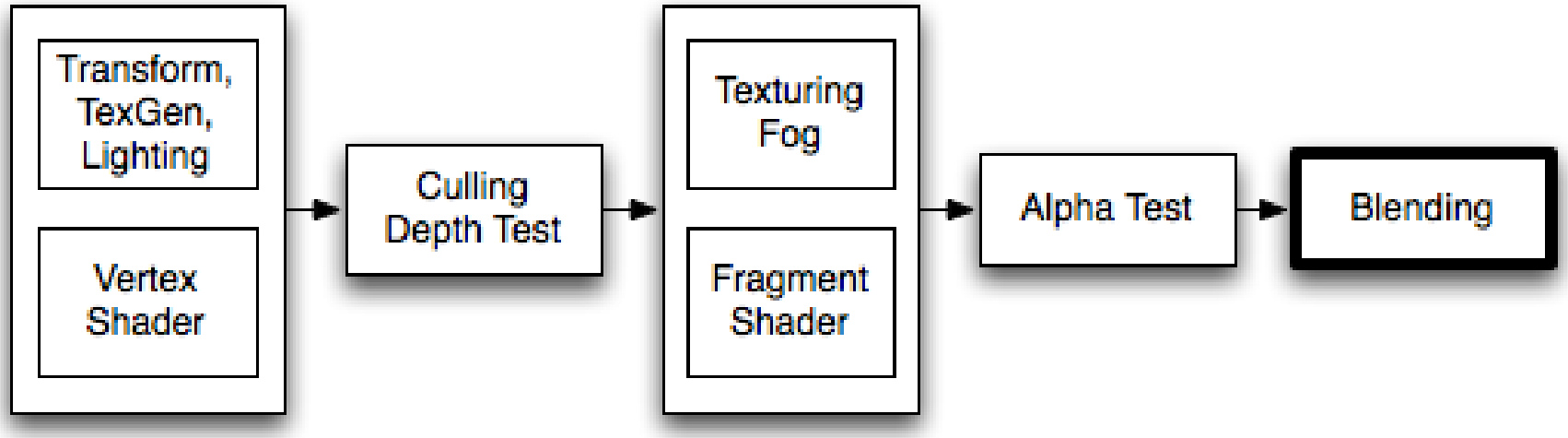
Fall 2024
11/28/2024
Kyoung Shin Park
Computer Engineering
Dankook University

Alpha Channel

- Alpha Channel Model
  - Porter & Duff's "Compositing Digital Images", SIGGRAPH'84
- RGBA – alpha is the 4th color and is used to adjust the opacity of color.
  - Opacity is a measure of how much light passes through a surface.
  - Alpha=1.0 – completely opaque
  - Alpha=0.5 – translucent
  - Alpha=0.0 – completely transparent

Blending

- In Unity, when graphics are rendered, after all shaders have executed and all textures have been applied, the pixels are written to the screen. How they are combined with what is already there is controlled by the **Blend** command.



https://docs.unity3d.com/kr/current/Manual/SL-Blend.html

Blending

- Blend the color of framebuffer and the color of object
- Blending equation

  FinalValue = srcFactor * srcColor BlendOp DstFactor * dstColor
  - SourceValue is the value output by the fragment shader.
  - DestinationValue is the value already in the destination buffer.
  - SourceFactor, DestinationFactor are specified with Blend command.
  - If the BlendOp command is used, the blending operation is set to that value. Otherwise, **the blending operation defaults** to **Add**.
    - If the blending operation is **Add**, **Sub**, **RevSub**, **Min**, or **Max**, the GPU multiplies the value of the output of the fragment shader by the source factor.
    - If the blending operation is **Add**, **Sub**, **RevSub**, **Min**, or **Max**, the GPU multiplies the value that is already in the render target by the destination factor.

BlendOp

- The BlendOp command sets the blending operation used by the Blend command.
- Example syntax:
  - BlendOp Sub // the subtract blending operation

BlendOp	Description
Add	Add source and destination together
Sub	Subtract destination from source.
RebSub	Subtract source from destination.
Min	Use the smaller of source and destination.
Max	Use the larger of source and destination.
LogicalClear	Logical operation: Clear
…	…

Blend Syntax

Example Syntax	Function
Blend Off	Disables blending for the default render target. This is **the default value**.
Blend 1 Off	As above, but for a given render target. (1)
Blend One Zero	Enables blending for the default render target. Sets blend factors for RGBA values.
Blend 1 One Zero	As above, but for a given render target. (1)
Blend One Zero, Zero One	Enables blending the default render target. Sets separate blend factors for RGB and alpha values. (2)
Blend 1 One Zero, Zero One	As above, but for a given render target. (1) (2)

Blend Factor

SrcFactor/DstFactor	Function
One	1 Use this to use the value of the source or the destination color.
Zero	0 Use this to remove either the source or the destination values.
SrcColor	SrcColor
DstColor	DstColor
DstAlpha	DstAlpha
OneMinusSrcColor	(1 – SrcColor)
OneMinusSrcAlpha	(1 – SrcAlpha)
OneMinusDstColor	(1 – DstColor)
OneMinusDstAlpha	(1 – DstAlpha)

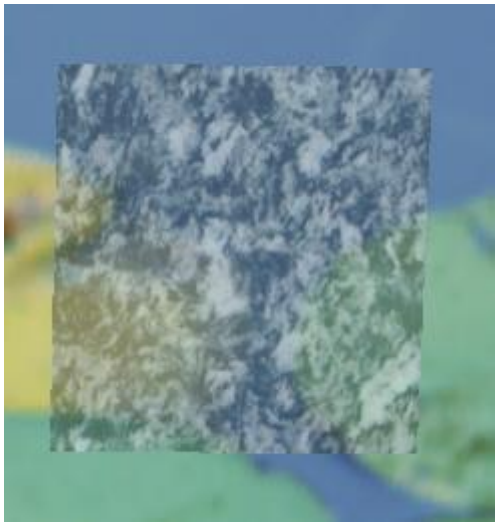
Common Blend Types

Example Syntax	Blending
Blend SrcAlpha OneMinusSrcAlpha	Alpha blending
Blend One OneMinusSrcAlpha	Premultiplied alpha blending
Blend One One	Additive blending
Blend OneMinusDstColor One	Soft additive
Blend DstColor Zero	Multiplicative
Blend DstColor SrcColor	2x multiplicative

Blend Filtering

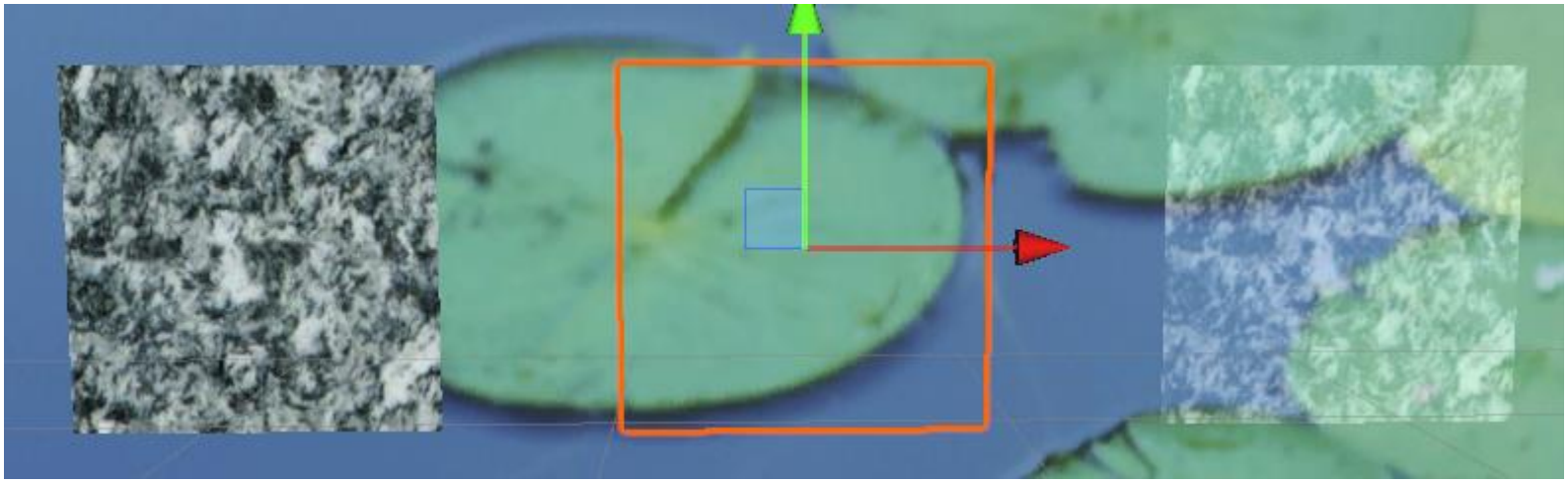
- Blending can be used for effects that filter out the color of the entire scene.
  - Draw a rectangle with the size of the entire screen and apply a blending function.

```
// alpha blending (Cs * As + Cd * (1-As))
Blend SrcAlpha OneMinusSrcAlpha
```



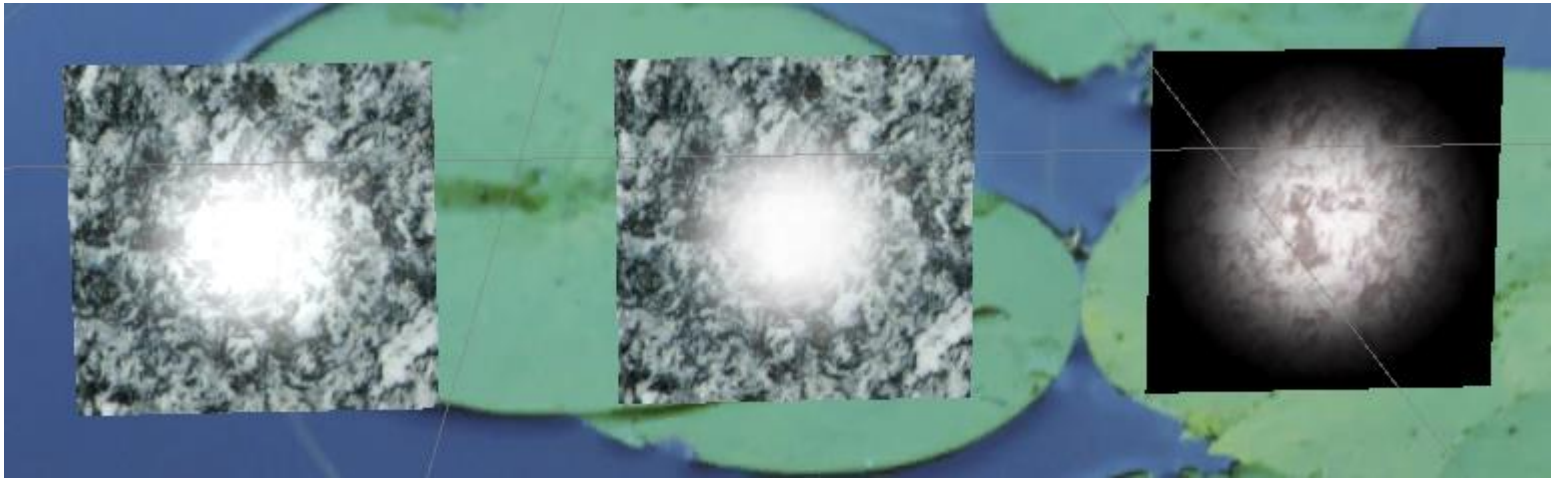
Blend Filtering

```
// no blending (Cs * 1 + Cd * 0)
Blend One Zero
// draw background (Cs * 0 + Cd * 1)
Blend Zero One
// Brighten the entire scene
Blend SrcAlpha One
```



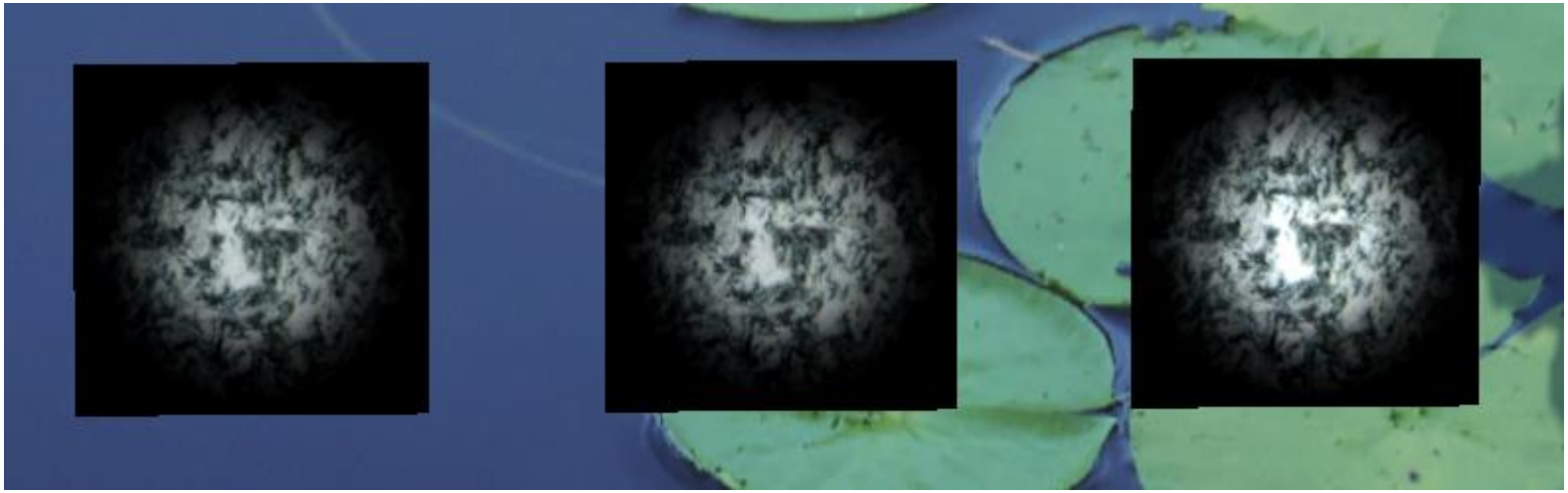
Blend Filtering

```
// additive blending (Cs * 1 + Cd * 1)
Blend One One
// soft additive blending (Cs * (1 - Cd) + Cd * 1)
Blend OneMinusDstColor One
// invert the color of the entire scene (Cs * (1 - Cd) + Cd * 0)
Blend OneMinusDstColor Zero
```



Blend Filtering

```
// multiplicative blending (Cs * Cd + Cd * 0)
Blend DstColor Zero
// multiplicative blending (Cs * 0 + Cd * Cs)
Blend Zero SrcColor
// 2x multiplicative blending (Cs * Cd + Cd * Cs)
Blend DstColor SrcColor
```



Blend Filtering

```
Shader "Custom/OneOne" { // OneOne.shader
    Properties      {
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
    }
    SubShader      {
        Tags { "RenderType"="Transparent" "Queue"="Transparent" }

        Pass          {
            ZWrite Off
            Blend One One // additive blending (Cs * 1 + Cd * 1)
            SetTexture [_MainTex]
            {
                Combine texture * previous
            }
        }
    }
}
```

Blending

- **Alpha Blending** makes the object appear transparent.
  Alpha blending = As * Cs + (1 - As) * Cd

// alpha blending – determine the transparency of object to be drawn by alpha

- $R = A_s * R_s + (1 - A_s) * R_d$
- $G = A_s * G_s + (1 - A_s) * G_d$
- $B = A_s * B_s + (1 - A_s) * B_d$
- $A = A_s * A_s + (1 - A_s) * A_d$

// source alpha = 0.3

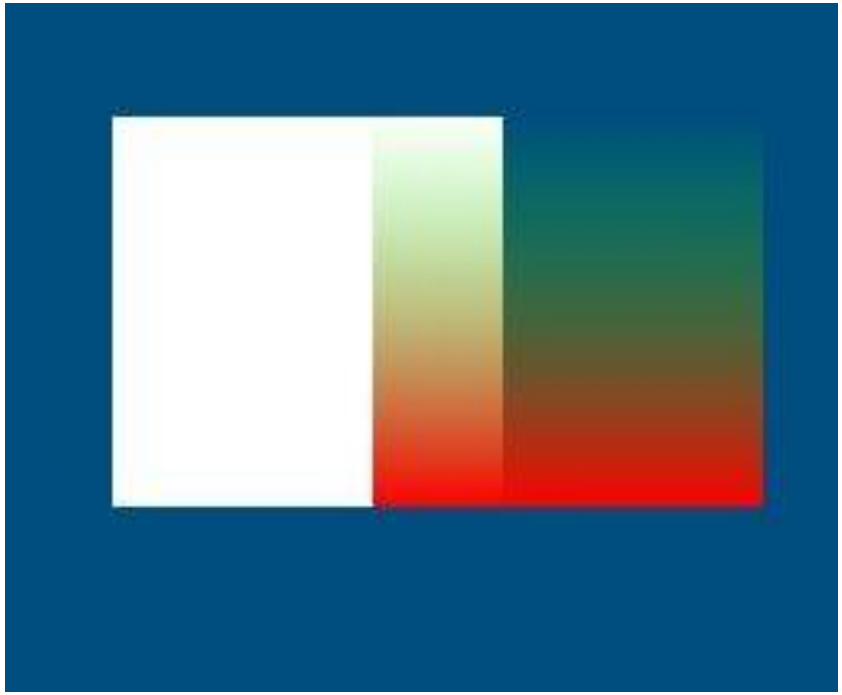
- $R = 0.3 * R_s + 0.7 * R_d$
- $G = 0.3 * G_s + 0.7 * G_d$
- $B = 0.3 * B_s + 0.7 * B_d$
- $A = 0.3 * A_s + 0.7 * A_d$

Dst color C_d = vec4(0.5, 1, 1, 1)
Src color C_s = vec4(1, 0, 1, 0.3)

R = 0.3*1 + 0.7*0.5 = 0.65
G = 0.3*0 + 0.7*1 = 0.7
B = 0.3*1 + 0.7*1 = 1
A = 0.3*0.3 + 0.7*1 = 0.79

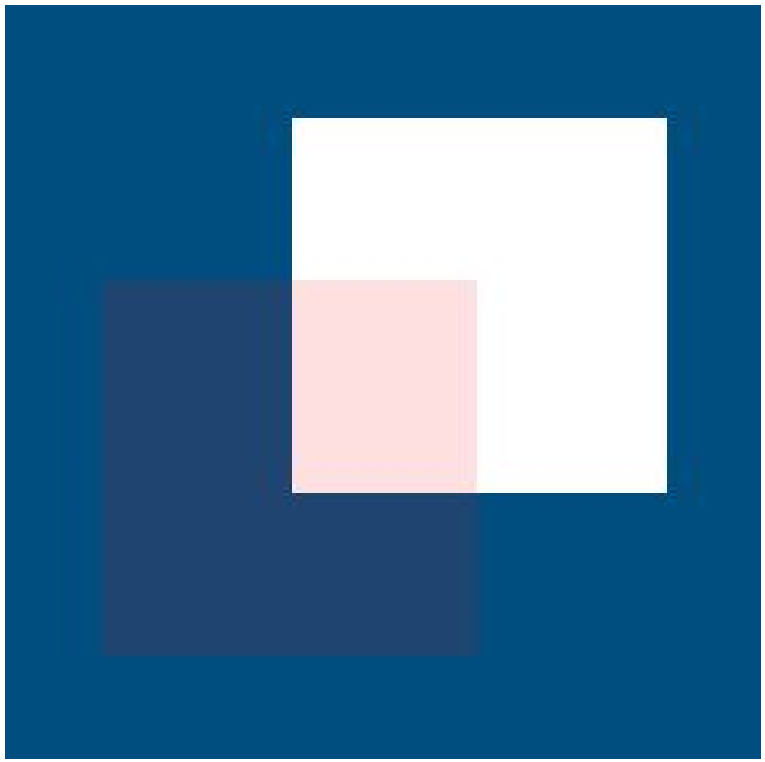
Smooth-shaded Alpha

- Like RGB colors, you can control the alpha value for each pixel in the application program.
  - If the alpha value is specified differently for each vertex, the alpha value is also interpolated – so, it can form a soft edge.



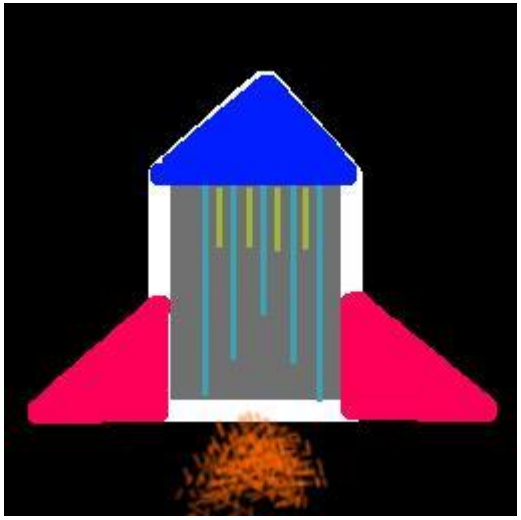
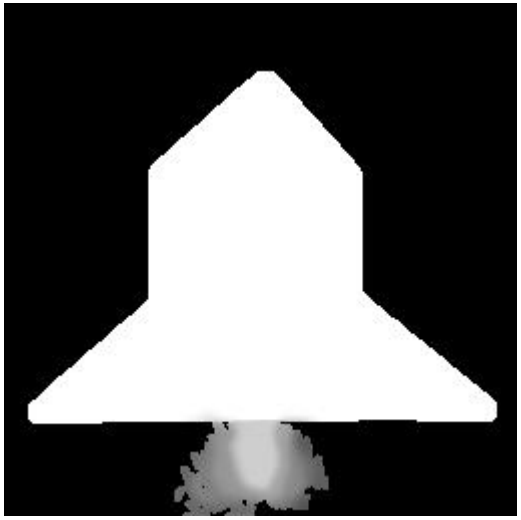
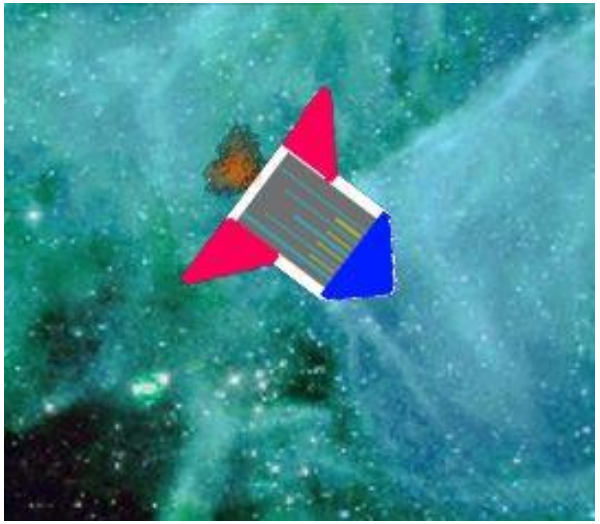
Time-Varying Alpha

- Changing the alpha value over time gives a fade-in or fade-out effect.



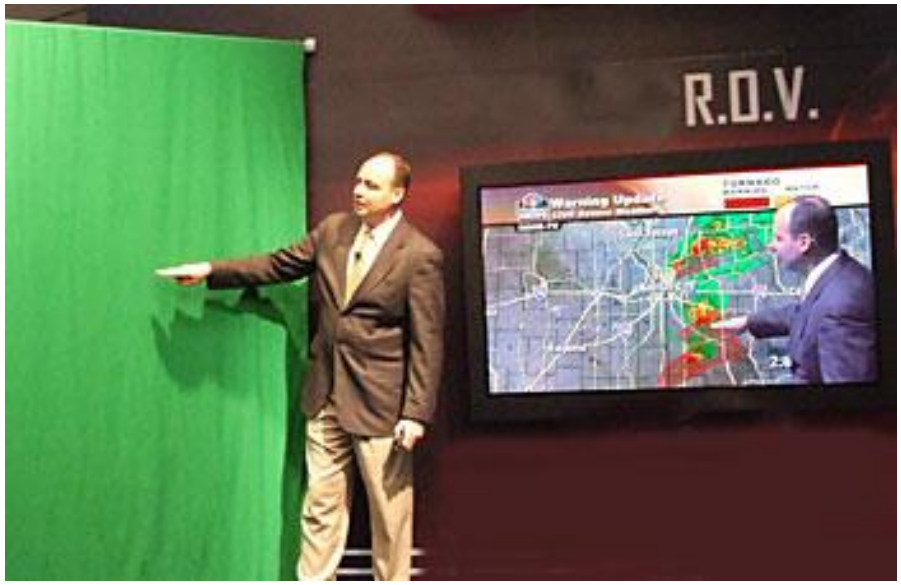
Texture Alpha

- Using RGBA 4-channel texture images, more complex shapes can be constructed on a simple geometric object.

Chroma Keying

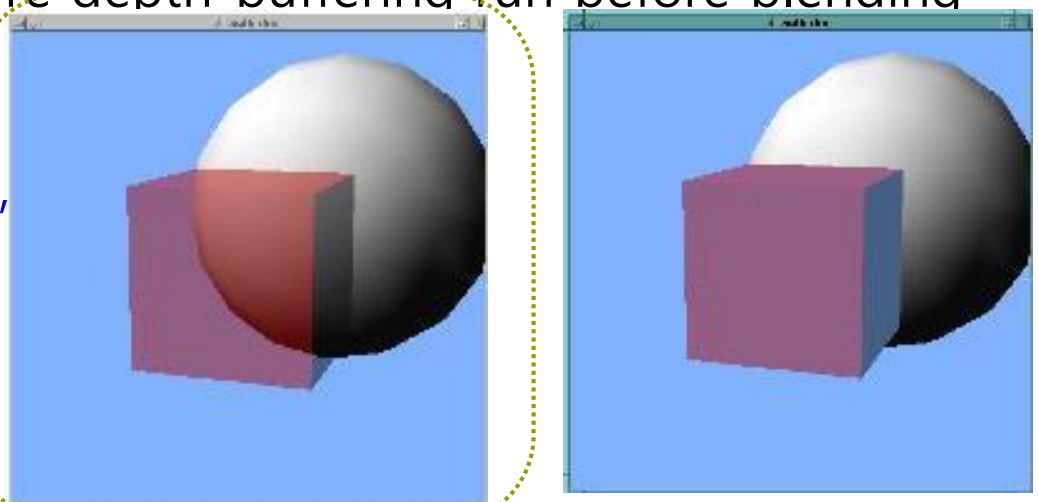
- Often used in film or video production.
- One example of chroma keying is the synthesis of images of live actors and graphical weather information in a weather caster's TV broadcasting.
- Use the background color as an alpha value.



Blending & Drawing Order

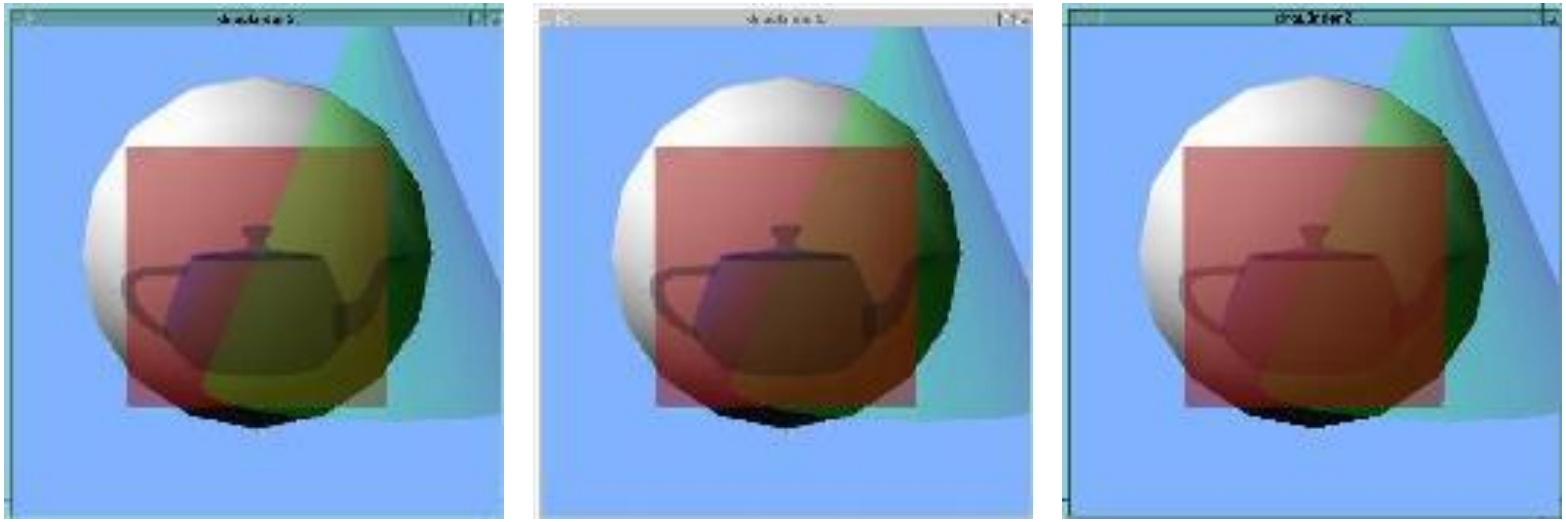
- For blending, the **drawing order** of the object to be drawn and the previously drawn object is important.
  - It acts as the source color (the color of object to be drawn) and the destination color (the color of the framebuffer already drawn) of the blending function.
- If you want to draw transparent and opaque object together, **draw opaque first and then transparent**.
  - Make sure depth-buffering run before blending

Draw sphere first,
Then draw cube



Blending & Drawing Order

- If you want to draw **multiple transparent objects** together, draw them in **back-to-front order**.
  - This order may vary depending on the location of camera.
- When drawing multiple transparent objects together, disable the depth mask to prevent occlusion.
  - Makes the depth buffer read-only.



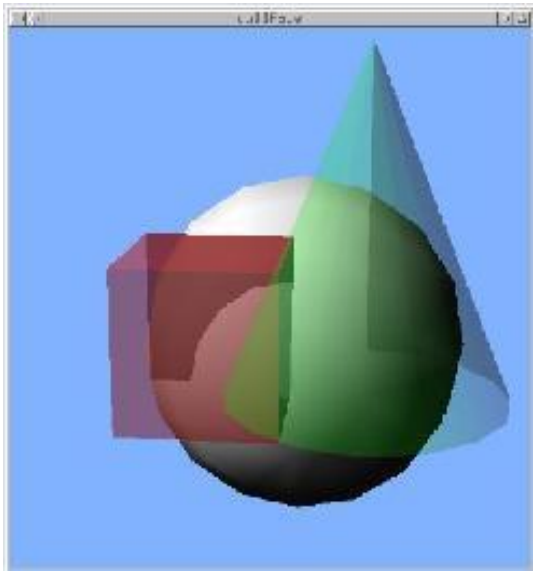
Render Queues

- Unity sorts objects into groups called **render queues**, which it renders in the following order.

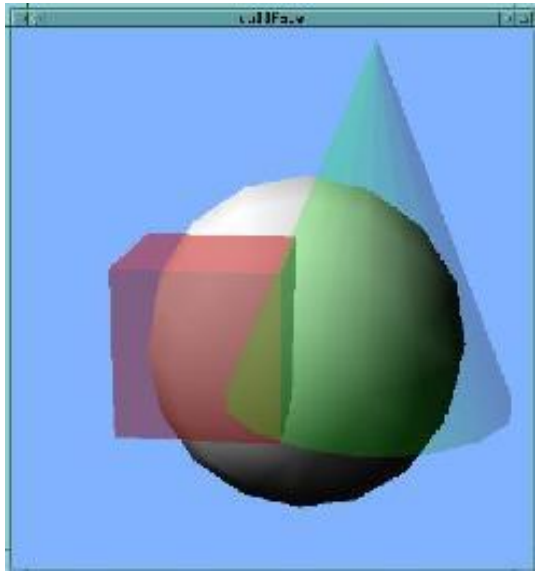
Name	Index	Description
Background	1000	Use this queue for anything that should be drawn in the background of your **Scene**
Geometry	**2000**	Use this queue for **opaque** geometry. This is the **default** queue.
AlphaTest	2450	Use this queue for alpha tested geometry. This is after the **Geometry** queue because it's more efficient to render alpha-tested objects after all solid ones are drawn.
Transparent	**3000**	Use this queue for anything **alpha-blended**; i.e. **shaders** that don't write to the **depth buffer**. Examples include glass, or particle effects.
Overlay	4000	Use this queue for effects that are rendered on top of everything else, such as **lens flares**.

Backface Culling

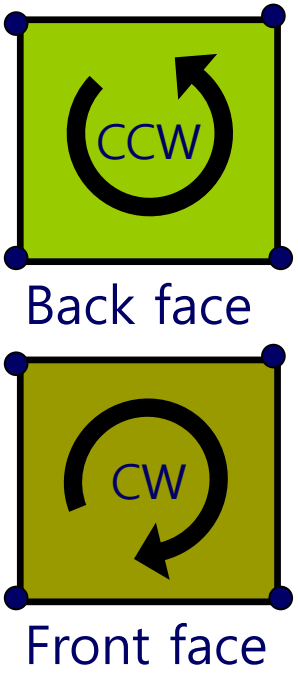
- When drawing transparent objects, enable backface culling.
  - Transparent objects usually have a rear view.
  - Backface culling prevents drawing the backface of an object.



No backface culling



Backface culling



Fog

- Fog effect
  - By blending with a depth-dependent color, it creates the feeling of a partially translucent space between the object and the observer.
  - To implement Fog in computer graphics, objects distant from the viewpoint are rendered small and fuzzy.
  - The point of time to apply the haze effect is performed last in the drawing process such as coordinate change, light source setting, and texture mapping.

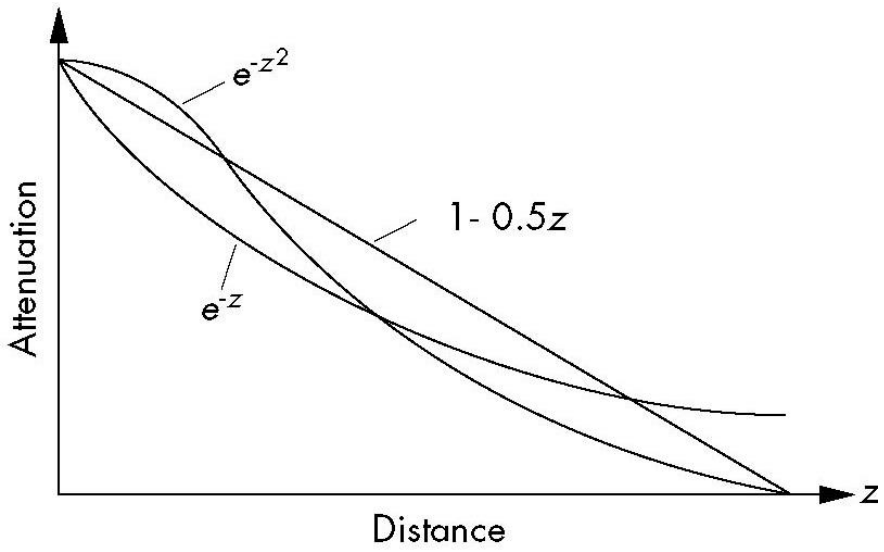
Example of “cheating” at atmospheric effects using global fog.
https://docs.unity3d.com/530/Documentation/Manual/script-GlobalFog.html

Fog

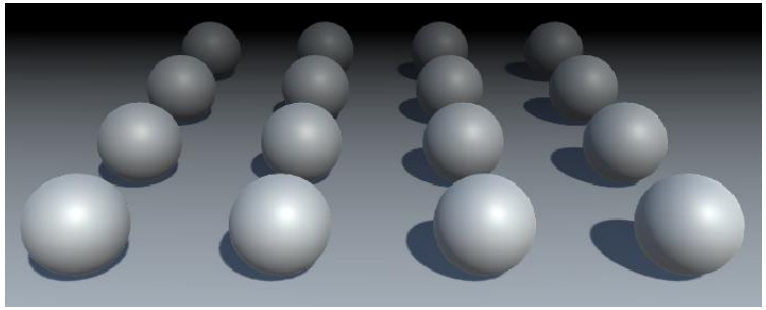
- Fog Mode
  - Linear (depth cueing)
    - The linear fog factor is computed with the function $f = \frac{E - c}{E - S}$ where c is the fog coordinate and S (the start) and E (the end).
  - Exponential, which is a more realistic approximation of fog
    - The exponential fog factor uses the function $f = \frac{1}{2^{cd}} = 2^{-cd}$ Where d is the fog's density factor.
  - Exponential Squared
    - It uses the function

      $f = \frac{1}{2^{(cd)^2}} = 2^{-(cd)^2}$

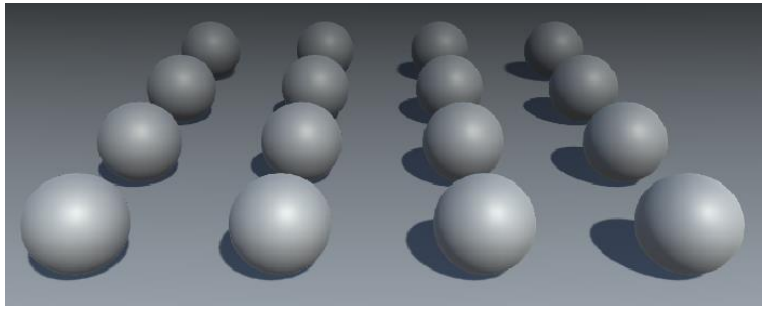
      which results in less fog at close range, but Increases quicker.



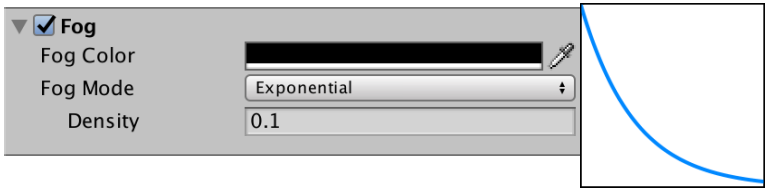
Fog

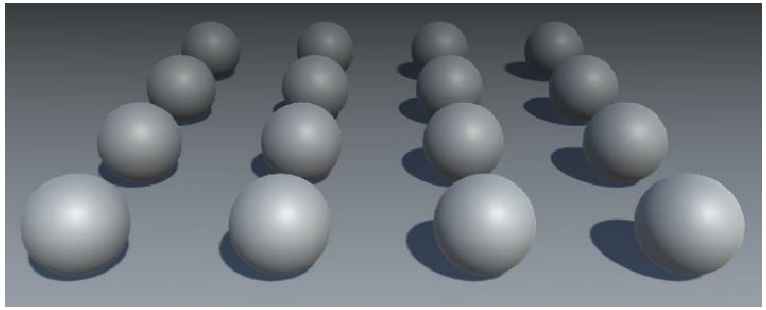


Linear fog

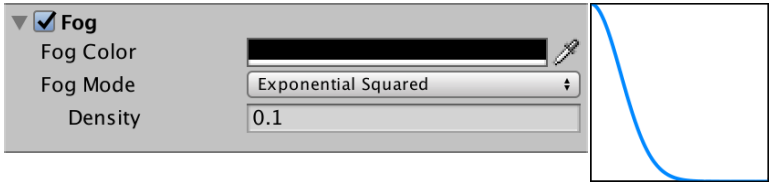


Exponential fog





Exponential squared fog



https://catlikecoding.com/unity/tutorials/rendering/part-14/

Fog

- Fog effect is the blending of the fog color and the color of a fragment. The degree of blending is calculated as a function of the distance between the fragment to be rendered and the viewer.