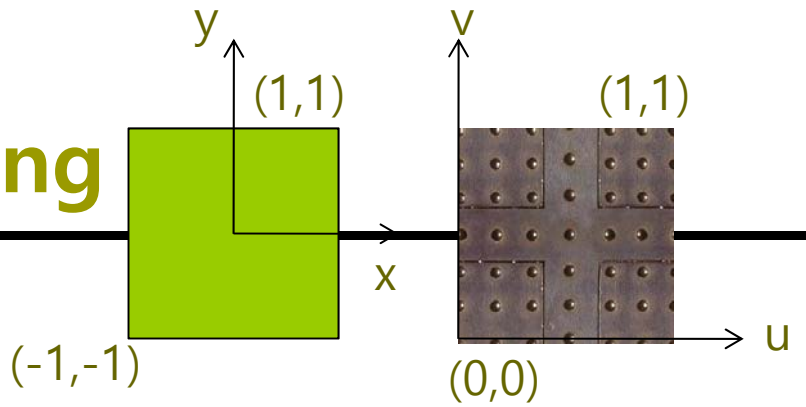
Blending

527970
Fall 2020
11/26/2020
Kyoung Shin Park
Computer Engineering
Dankook University

Single-Pass Multitexturing




```
void SetMultitextureSquareData() { // ..
glGenBuffers(4, &vbo[0]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec3), &quadVertices[0], GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec3), &quadNormals[0], GL_STATIC_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec2), &quadTextureCoords[0], GL_STATIC_DRAW
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, vbo[3]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec2), &quadTextureCoords[0], GL_STATIC_DRAW
glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(3);
}
```

Single-Pass Multitexturing

- Bind and enable two 2D multitextures to draw a quad

```
// stage 0 activate
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture0);
// stage 1 activate
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texture1);
// draw multitexture square
drawSquare();
// texture disabled
glBindTexture(GL_TEXTURE_2D, 0);
```

Single-Pass Multitexturing

- GLSL fragment shader

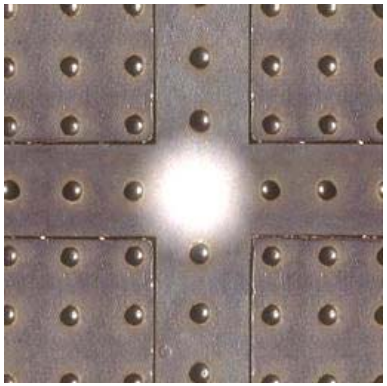
```
uniform sampler2D gTextureSampler1, gTextureSampler2;
uniform int gModulate;

// Material properties
if (gModulate == 1)
    MaterialDiffuseColor = texture2D(gTextureSampler1, TexCoordPass0).rgba *
    texture2D(gTextureSampler2, TexCoordPass1).rgba;
else
    MaterialDiffuseColor = texture2D(gTextureSampler1, TexCoordPass0).rgba +
    texture2D(gTextureSampler2, TexCoordPass1).rgba;
vec4 MaterialAmbientColor = gMaterialAmbientColor * MaterialDiffuseColor;
vec4 MaterialSpecularColor = gMaterialSpecularColor;
```

Multipass Multitexturing

- Rendering the same object multiple times using different modes
  - For example, after drawing an object normally for a lightmap effect, the same object is drawn again using a blending function.

```
// first pass – normal drawing using normal texture
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureID1);
drawSquare();
// second pass – blend the lightmap texture with the original texture
glDepthFunc(GL_LEQUAL); // accept co-planar fragments
glEnable(GL_BLEND);
glBlendFunc(GL_ONE, GL_ONE); // Add Blending
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureID2);
drawSquare();
glDepthFunc(GL_LESS);
glDisable(GL_BLEND);
```



Alpha Channel

- Alpha Channel Model
  - Porter & Duff's "Compositing Digital Images", SIGGRAPH'84
- RGBA – alpha is the 4^{th} 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

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

  $R = SourceFactor * R_s + DestinationFactor * R_d$

  $G = SourceFactor * G_s + DestinationFactor * G_d$

  $B = SourceFactor * B_s + DestinationFactor * B_d$

  - Source color (R_s, G_s, B_s) is the color of object
  - Destination color (R_d, G_d, B_d) is the color in the framebuffer.
  - SourceFactor, DestinationFactor are specified with glBlendFunc().
- glBlendFunc() function
  - Alpha Blending

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

GLSL Alpha Blending

```
Gvec4 result = vec4(gl_FragColor.a) * gl_FragColor + vec4(1.0 -
   gl_FragColor.a) * pixel_color;
```

Blending

- Alpha Blending makes the object appear transparent.

  Alpha blending = $A_s * C_s + (1 - A_s) * C_d$

```
// alpha blending – determin the transparency of object to be drawn by alpha
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_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

Blending Functions

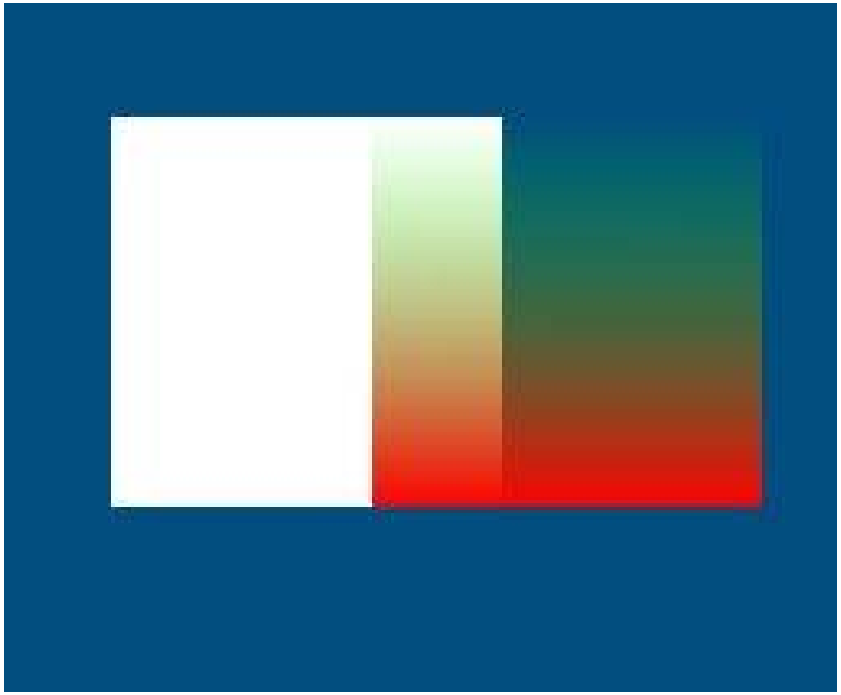
Factor name	Computed Factor
GL_ZERO	vec4(0.0)
GL_ONE	vec4(1.0)
GL_SRC_ALPHA	vec4(gl_FragColor.a)
GL_ONE_MINUS_SRC_ALPHA	vec4(1.0 – gl_FragColor.a)
GL_DST_ALPHA	pixel_color.a
GL_ONE_MINUS_DST_ALPHA	vec4(1.0 – pixel_color.a)
GL_CONSTANT_ALPHA	vec4(color.a)
GL_ONE_MINUS_CONSTANT_ALPHA	vec4(1.0 – color.a)
GL_SRC_COLOR	gl_FragColor
GL_ONE_MINUS_SRC_COLOR	vec4(1.0) - gl_FragColor
GL_DST_COLOR	pixel_color
GL_ONE_MINUS_DST_COLOR	vec4(1.0) - pixel_color
GL_CONSTANT_COLOR	color
GL_ONE_MINUS_CONSTANT_COLOR	vec4(1.0) – color

OpenGL Blending

- Enable blending
  - glEnable(GL_BLEND)
- Define the blending function
  - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
- Add the alpha in [0, 1]
  - RGBA vec4(1, 0, 0, 0.5) // transparency 50% red
- Or, use an RGBA textured image containing alpha

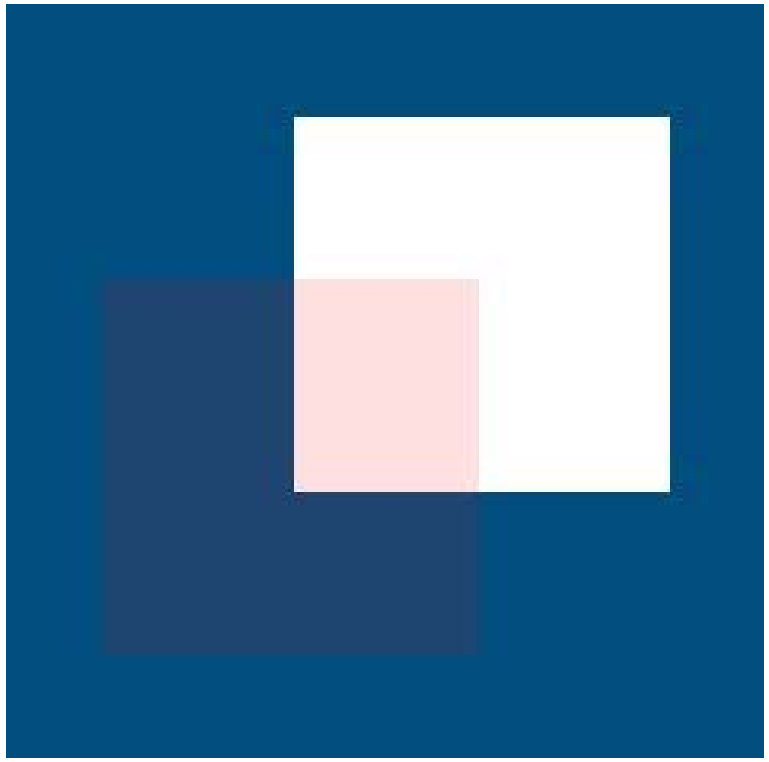
Smooth-shaded Alpha

- 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.



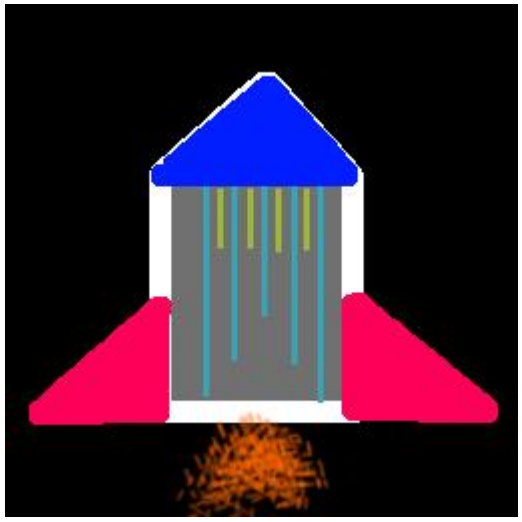
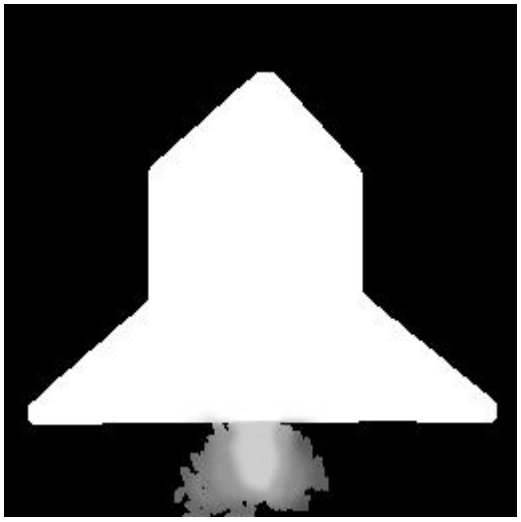
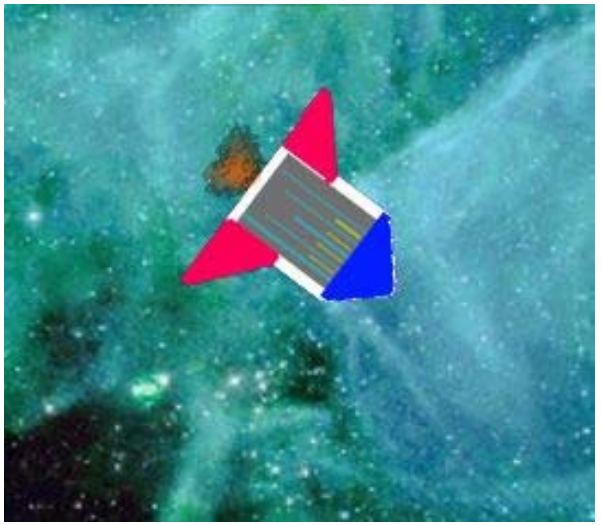
Time-Varying Alpha

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



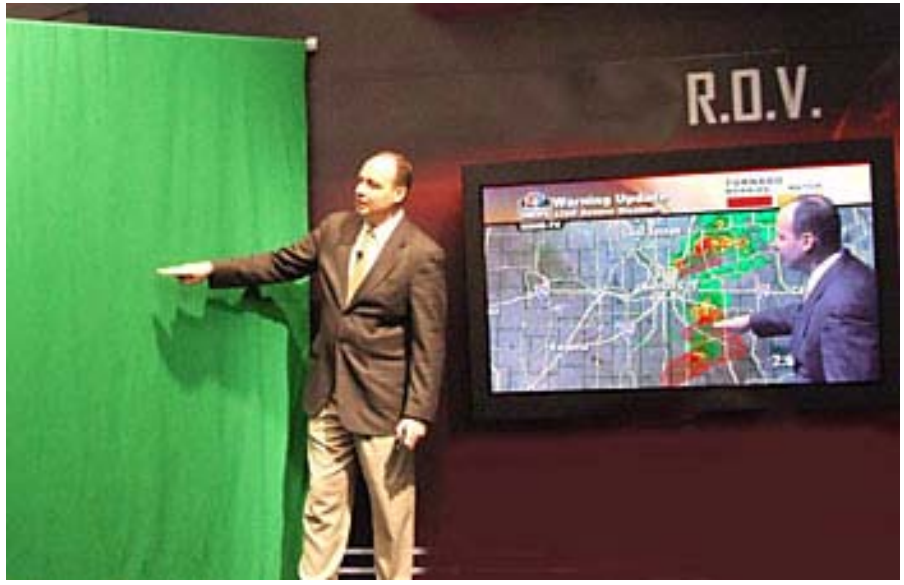
Texture Alpha

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

Chroma Keying

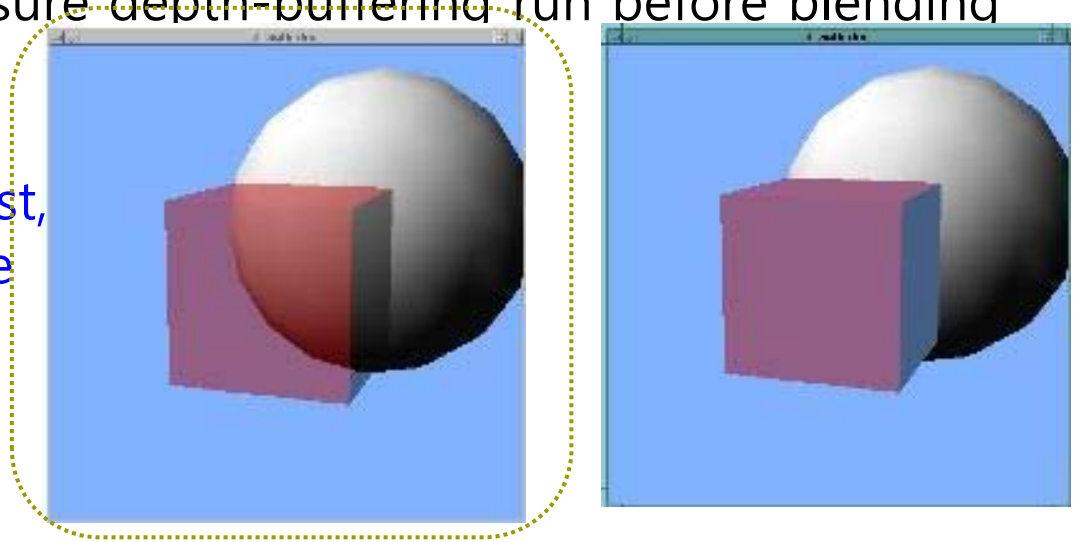
- Often used in film or video production.
- One example of chroma keying is the ㅋsynthesis of images of live actors and graphical weather information in a weather caster's TV broadcasting.
- Use the background color as an alpha value.



Blending & Drawing Order

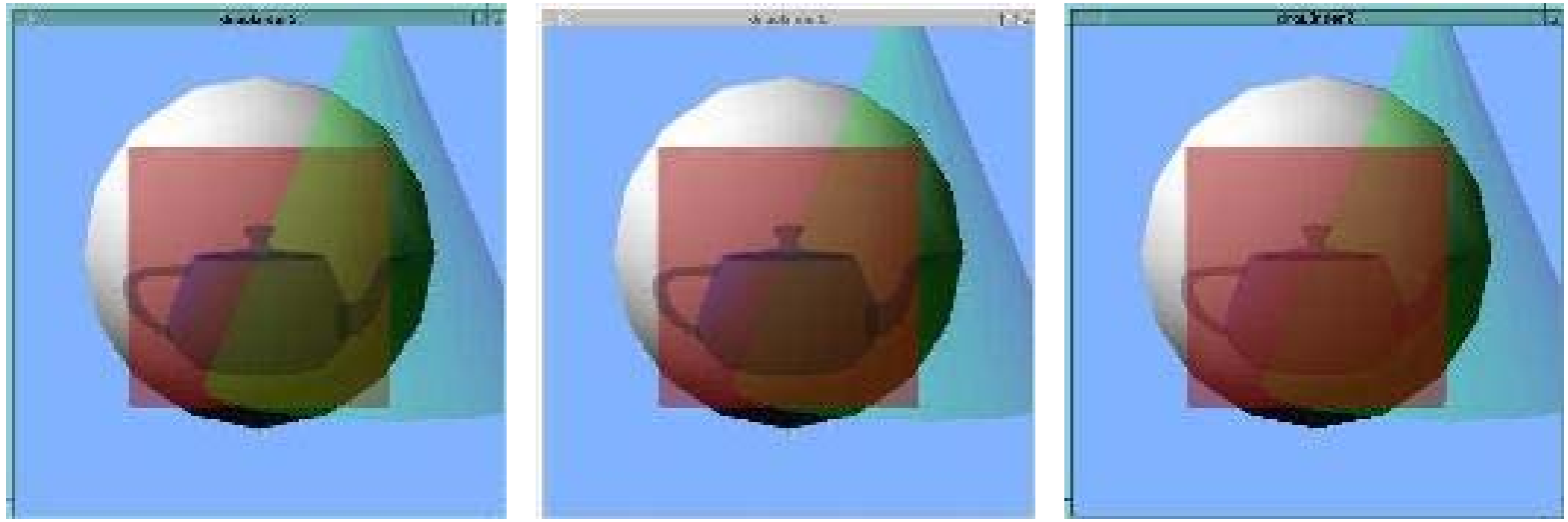
- For blending, the **drawing order** of the object to be drawn and the previously drawn object is important.
  - It acts as the source color (the color of object to be drawn) and the destination color (the color of the framebuffer already drawn) of the blending function.
- If you want to draw transparent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

Draw sphere first,
Then draw cube



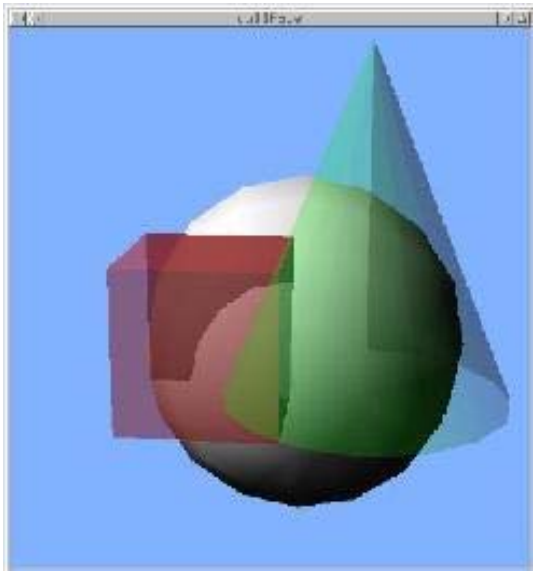
Blending & Drawing Order

- 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.
  - glDepthMask(GL_FALSE) makes the depth buffer read-only.

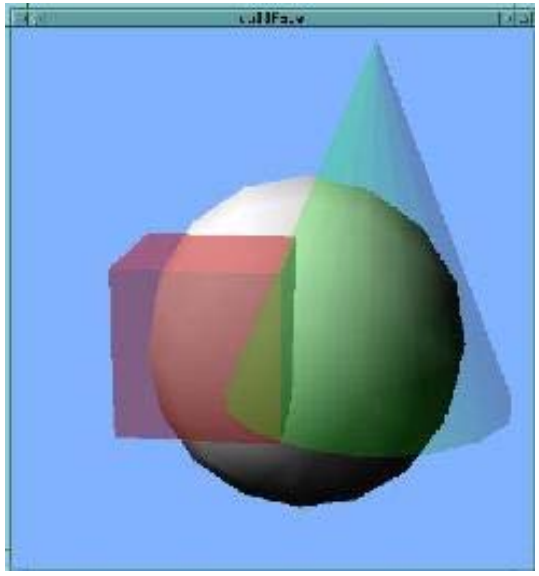


Backface Culling

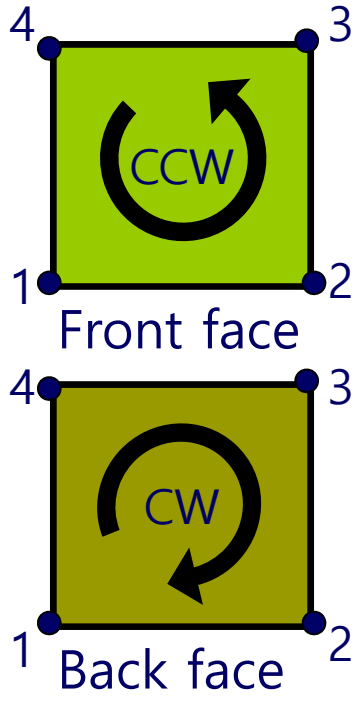
- When drawing transparent objects, enable backface culling.
  - Transparent objects usually have a rear view.
  - glEnable(GL_CULL_FACE) prevents drawing the backface of an object.



glDisable(GL_CULL_FACE)



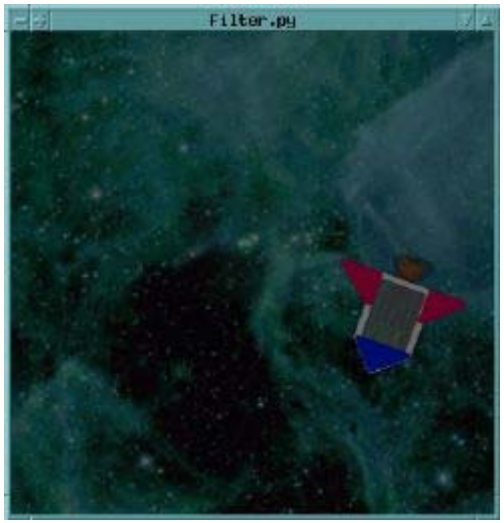
glEnable(GL_CULL_FACE)



Filtering

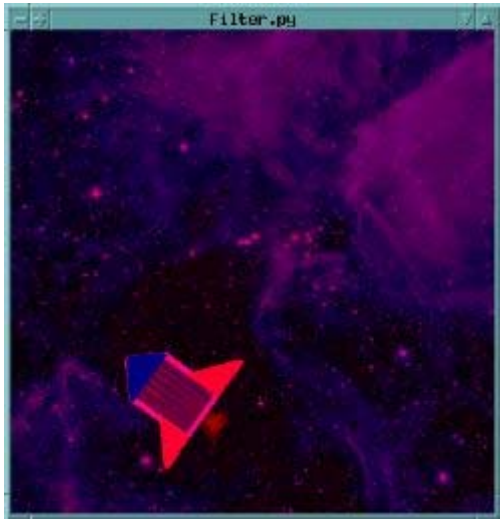
- Blending can be used for effects that filter out the color of the entire scene.
  - Draw a rectangle with the size of the entire screen and pply a blending function.

```
// darken the whole scene
glBlendFunc(GL_ZERO, GL_SRC_ALPHA)
```

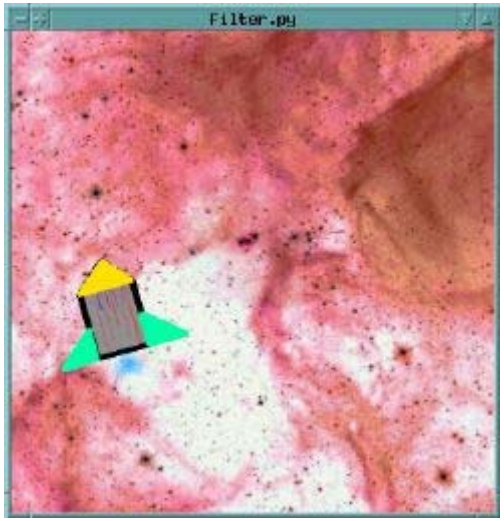


Filtering

```
// Make the scene the color you want (purple)
glBlendFunc(GL_ZERO, GL_SRC_COLOR);
glColor4f(1.0, 0.0, 0.5, 1.0);
```




```
// Inverse the color of the entire scene
glBlendFunc(GL_ONE_MINUS_DST_COLOR, GL_ZERO);
glColor4f(1.0, 1.0, 1.0, 1.0)
```



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.
  - Fog is supported in OpenGL. 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.

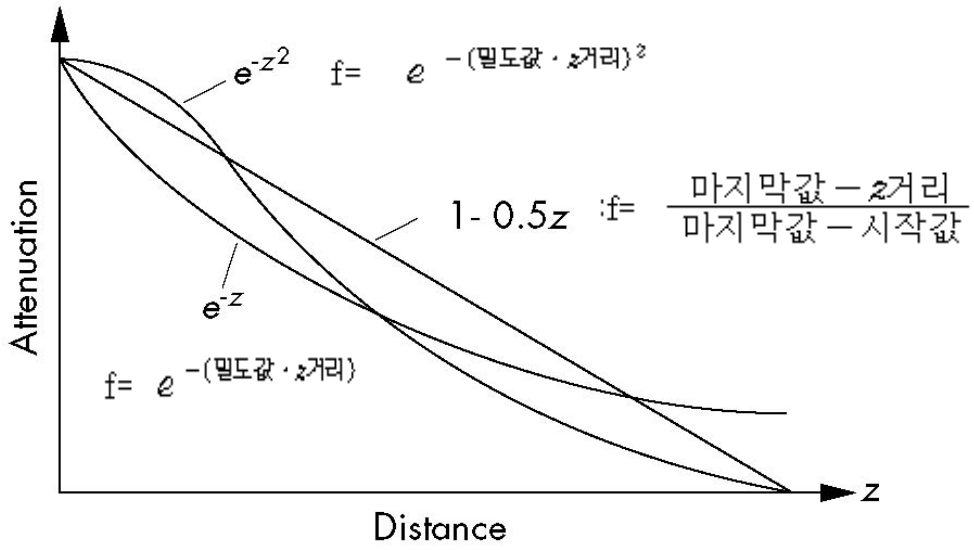
Fog

- Fog blending function

  finalColor = FogFactor * fragmentColor + (1 – FogFactor) * fogColor

- Fog function
  - void glFogifv(Glenum pname, TYPE param)

- Fog factor
  - Exponential
  - Gaussian
  - Linear (depth cueing)



OpenGL Fog

- OpenGL 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.

```
GLfloat fcolor[4] = {1.0, 1.0, 1.0, 1.0}:
glEnable(GL_FOG);
glFogi(GL_FOG_MODE, GL_LINEAR);
glFogf(GL_FOG_START, 5.0);
glFogf(GL_FOG_END, 40.0);
glFogfv(GL_FOG, fcolor);
```

OpenGL Fog Mode

- Fog factor
  - Linear
    - glFogi(GL_FOG_MODE, GL_LINEAR);
    - GL_FOG_START, GL_FOG_END
  - Exponential
    - glFogi(GL_FOG_MODE, GL_EXP);
    - GL_FOG_DENSITY
  - Gaussian
    - glFogi(GL_FOG_MODE, GL_EXP2);
    - GL_FOG_DENSITY

```
GLfloat fcolor[4] = {1.0, 1.0, 1.0, 1.0}:
glEnable(GL_FOG);
glFogi(GL_FOG_MODE, GL_EXP);
glFogf(GL_FOG_DENSITY, 0.5);
glFogfv(GL_FOG, fcolor);
```

OpenGL Fog Mode

```
struct FogParameters {
vec4 vFogColor; // Fog color
float fStart; float fEnd; // This is only for linear fog
float fDensity; // For exp and exp2 equation
int iEquation; // 0 = linear, 1 = exp, 2 = exp2
};
float getFogFactor(FogParameters params, float fFogCoord) {
    float fResult = 0.0;
    if(params.iEquation == 0)
        fResult = (params.fEnd-fFogCoord)/(params.fEnd-params.fStart);
    else if(params.iEquation == 1)
        fResult = exp(-params.fDensity*fFogCoord);
    else if(params.iEquation == 2)
        fResult = exp(-pow(params.fDensity*fFogCoord, 2.0));
    fResult = 1.0-clamp(fResult, 0.0, 1.0);
    return fResult;
}
```