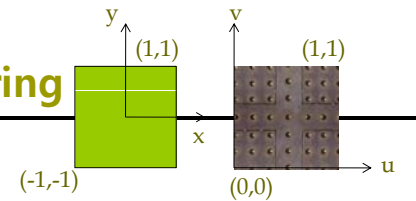


Blending

321190
2014년 봄학기
5/22/2014
박경신

Single-Pass Multitexturing



```
void SetMultitexturSquareData() { // 중간생략..
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

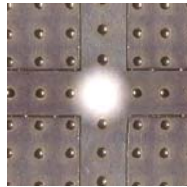
```
uniform sampler2D gTextureSampler1, gTextureSampler2;
uniform int gModulate;

// 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

- 같은 물체를 다른 모드를 사용하여 여러 번 렌더링하는 것
 - 예를 들어, 라이트맵(lightmap) 효과를 위해 물체를 정상적으로 그리고 난 후 블렌딩 함수를 사용하여 같은 물체를 다시 한번 그려줌

```
// first pass - 일반 텍스처를 사용하여 정상적으로 그림
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureID1);
drawSquare();
// second pass - 라이트맵 텍스처와 원래 텍스처를 블렌딩함
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 - alpha는 4번째 색으로 불투명도(opacity of color) 조절에 사용함
 - 불투명도 (opacity)는 얼마나 많은 빛이 면을 관통하는가의 척도임
 - 투명도 (transparency)는 1 - alpha로 주어짐
 - Alpha=1.0 - 완전히 불투명
 - Alpha=0.5 - 반투명
 - Alpha=0.0 - 완전히 투명

Blending

- 프레임 버퍼의 색과 물체의 색을 합성함
- 일반적인 블렌딩 공식
$$R = \text{SourceFactor} * R_s + \text{DestinationFactor} * R_d$$
$$G = \text{SourceFactor} * G_s + \text{DestinationFactor} * G_d$$
$$B = \text{SourceFactor} * B_s + \text{DestinationFactor} * B_d$$
 - Source color (R_s, G_s, B_s)는 물체의 색
 - Destination color (R_d, G_d, B_d)는 프레임버퍼에 있는 색
 - SourceFactor, DestinationFactor는 glBlendFunc() 함수로 지정함
- glBlendFunc()함수에서 쓰이는 블렌딩 공식
 - 알파 블렌딩의 경우
$$\text{glBlendFunc}(\text{GL_SRC_ALPHA}, \text{GL_ONE_MINUS_SRC_ALPHA});$$
GLSL 알파 블렌딩
$$\text{Vec4 result} = \text{vec4}(\text{gl_FragColor.a}) * \text{gl_FragColor} + \text{vec4}(1.0 - \text{gl_FragColor.a}) * \text{pixel_color};$$

Blending

- 알파 블렌딩 - 물체의 색을 투명하게 나타나게 함
$$\text{Alpha blending} = A_s * C_s + (1 - A_s) * C_d$$

```
// alpha blending - alpha에 의해 그리고자 하는 물체의 투명도 결정
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

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

대상 색상 값 $C_d = \text{vec4}(0.5, 1, 1, 1)$
소스 색상 값 $C_s = \text{vec4}(1, 0, 1, 0.3)$

- ```
// 즉, 소스 alpha = 0.3
■ $R = 0.3 * R_s + 0.7 * R_d$
■ $G = 0.3 * G_s + 0.7 * G_d$
■ $B = 0.3 * B_s + 0.7 * B_d$
■ $A = 0.3 * A_s + 0.7 * A_d$
```

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

## Blending Functions

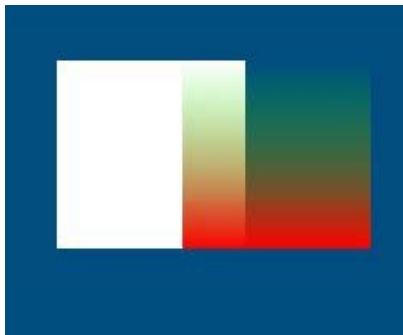
| Factor name                 | Computed Factor            |
|-----------------------------|----------------------------|
| GL_ZERO                     | vec4(0.0)                  |
| GL_ONE                      | vec4(1.0)                  |
| GL_SRC_ALPHA                | vec4(gl_FragColor.a)       |
| 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

- 블렌딩 활성화
  - glEnable(GL\_BLEND)
- 블렌딩 함수 정의
  - glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)
- [0, 1]영역의 알파값 추가
  - RGBA vec4(1, 0, 0, 0.5) // transparency 50% red
- 혹은 알파가 있는 RGBA 텍스처 이미지를 사용함

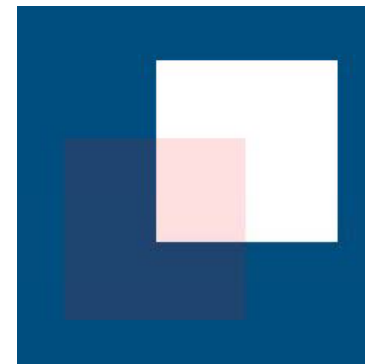
## Smooth-shaded Alpha

- R,G,B 색들과 마찬가지로 응용프로그램에서 각각의 픽셀에 대한 Alpha 값을 제어 할 수 있음
  - 만약 알파값이 각 정점에 다르게 지정되어 있으면, 알파값도 보간되어 나타남 - 그래서 부드러운 면 (soft edge)을 형성할 수 있음



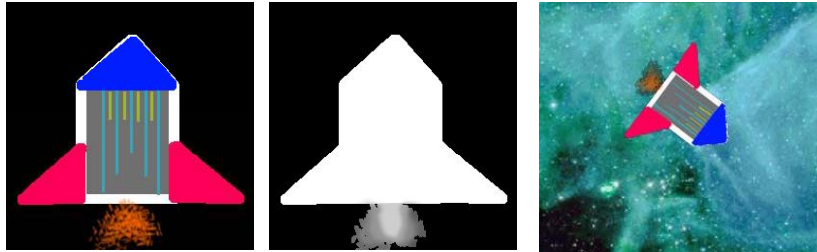
## Time-Varying Alpha

- 시간의 경과에 따라 알파값을 변하게 주어 fade-in 또는 fade-out 효과를 줄 수 있음



## Texture Alpha

- RGBA 4채널 텍스처 이미지를 사용하여 보다 복잡한 형체를 간단한 기하객체를 사용하여 구성할 수 있음



## Chroma Keying

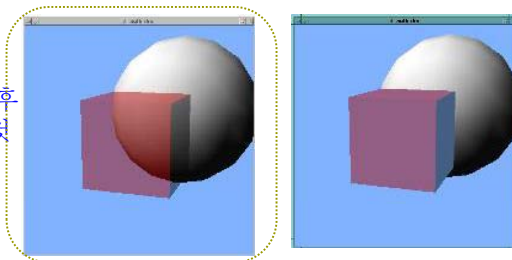
- 영화나 비디오 프로덕션에서 많이 사용
- 크로마키잉의 단적인 예로, 기상 캐스터의 TV 날씨 방송에서 실시간 액터 (live actor)의 이미지와 그래픽적인 날씨 정보의 합성을 들 수 있음
- 배경색을 찾아서 그 값을 알파값으로 지정하여 사용함



## Blending & Drawing Order

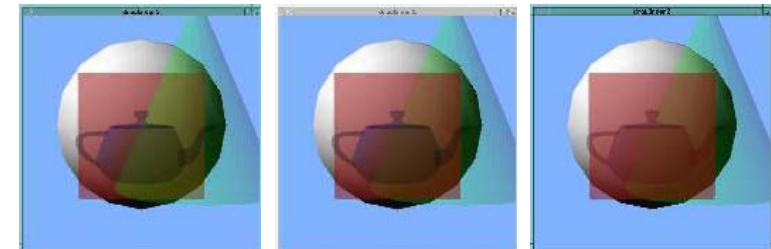
- 블렌딩은 현재 그리고자 하는 물체와 이전에 그려진 물체의 그림 그리는 순서(drawing order)가 중요함
  - 블렌딩 함수의 source color(현재 그리고자 하는 물체의 색)와 destination color (이미 그려진 프레임버퍼의 색)로 작용함
- 만약 투명한 물체와 불투명한 물체를 같이 그리고자 한다면, **불투명부터 먼저 그린 후에 투명한 것을 그릴 것**
  - Depth-buffering이 블렌딩 전에 실행되도록 함

구를 먼저 그린 후  
입방체를 그릴 것



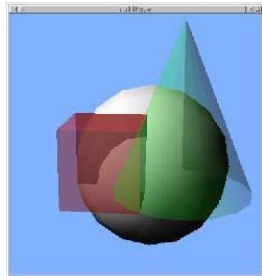
## Blending & Drawing Order

- 만약 여러 개의 투명한 물체를 같이 그리고자 한다면, 전향 순서 (back-to-front order)로 그릴 것
  - 이 순서는 카메라의 위치에 의해서 달라질 수 있음
- 여러 개의 투명한 물체를 같이 그릴 때, 서로를 가리는 현상 (occlusion)을 막기 위해서 depth mask를 비활성화함
  - glDepthMask(GL\_FALSE)를 통하여 깊이버퍼를 read-only로 만듦

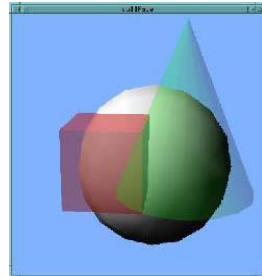


## Backface Culling

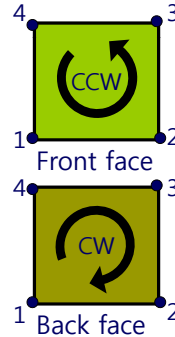
- 투명한 물체를 그릴 때는 후면 추리기 (backface culling)를 활성화할 것
  - 투명한 물체는 일반적으로 후면이 보이게 됨
  - glEnable(GL\_CULL\_FACE)는 물체의 후면 (backface)를 그리는 것을 막아줌



glDisable(GL\_CULL\_FACE)



glEnable(GL\_CULL\_FACE)

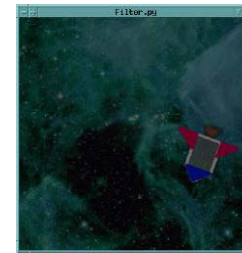


## Filtering

- 블렌딩은 전체 장면의 색을 필터링하는 효과에 사용될 수 있음
  - 전체 화면의 크기를 가진 사각형을 그리고 블렌딩 함수를 적용함

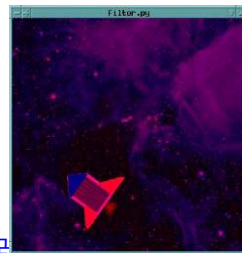
// 전체 장면을 어둡게 함

```
glBlendFunc(GL_ZERO, GL_SRC_ALPHA)
```

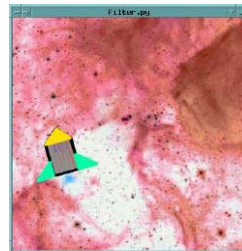


## Filtering

```
// 전체 장면에 원하는 색 (즉, 보라색)으로 만듦
glBlendFunc(GL_ZERO, GL_SRC_COLOR);
glColor4f(1.0, 0.0, 0.5, 1.0);
```



```
// 전체 장면의 색을 보색(inverted color)으로 바꿈
glBlendFunc(GL_ONE_MINUS_DST_COLOR, GL_ZERO);
glColor4f(1.0, 1.0, 1.0, 1.0)
```



## Fog

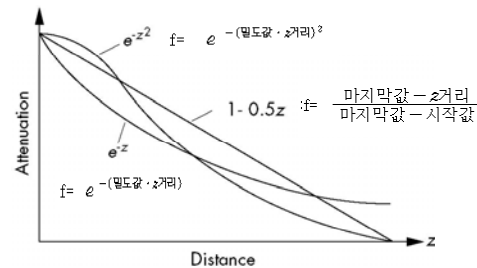
- 연무 효과 (fog effect)
  - 깊이에 의존적인 색으로 블렌딩함으로써 물체와 관측자 사이의 부분적인 반투명 공간의 느낌을 생성함
  - Fog를 컴퓨터 그래픽스에서 구현하려면 관측점에서 멀리있는 물체를 작고 희미하게 보이도록 표현함
  - OpenGL에서 Fog를 지원하는데, 연무 효과를 적용하는 시점은 좌표변화, 광원 설정, 텍스처 매핑 등의 그리기 과정에서 제일 마지막에 수행함

## Fog

- 연무 블렌딩 함수  
 $finalColor = FogFactor * fragmentColor + (1 - FogFactor) * fogColor$

- 연무 함수
  - void glFogfv(Glenum pname, TYPE param)

- 연무계수 (fog factor)
  - Exponential
  - Gaussian
  - Linear (depth cueing)



## OpenGL Fog

- OpenGL에서 연무효과는 연무 색과 단편 (fragment)의 색이 합성되는 것임. 합성의 정도는 렌더링될 단편과 관측자와의 거리 함수로 계산됨.

```
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

- 연무계수 (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;
}
```