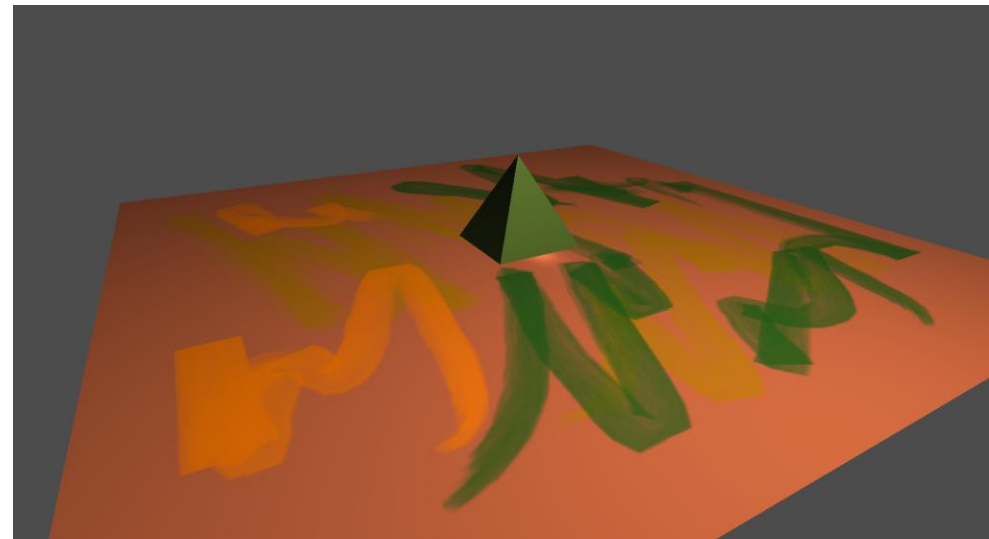


# OpenGL 텍스처 매핑

2022년 2학기

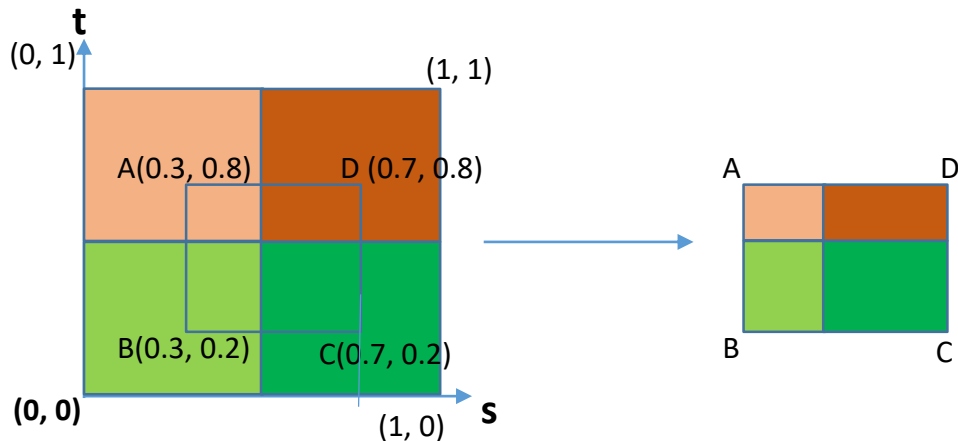
# 텍스처 매핑

- 텍스처 (Texture)
  - 객체에 세부 정보를 추가하는데 사용되는 이미지. 대개 2차원 이미지
- 텍스처 매핑 (Texture mapping)
  - 물체의 표면에 텍스처를 입히는 것
- 텍스처 매핑을 위해서
  - 1) 텍스처 생성
    - 이미지를 파일에서 읽기
    - 텍스처 생성
  - 2) 텍스처 정의
    - 래핑, 필터링 방법 등 텍스처 파라미터 정의
    - 객체의 각 정점에 텍스처 좌표를 지정
    - 텍스처 정의
  - 3) 텍스처 매핑하기
    - 텍스처를 객체에 바인드하기
    - 객체 그리기



# 텍스처 매핑

- Modern OpenGL에서 텍스처 매핑은 프래그먼트 셰이딩에서 sampler로 구현됨
  - 샘플러 (sampler): 텍스처 값(texel)에 접근하는데 사용되는 텍스처 타입
    - 샘플러는 GLSL에서 접근 가능한 텍스처를 나타내는 변수로 uniform 타입으로 선언한다.
    - 샘플러는 텍스처 색을 반환한다.
    - 텍스처 샘플링을 위한 데이터 타입:
      - sampler1D : 1D 텍스처를 위한 샘플러
      - sampler2D : 2D 텍스처를 위한 샘플러
      - sampler3D : 3D 텍스처를 위한 샘플러
  - 샘플링 (sampling): 텍스처 좌표를 사용하여 텍스처 컬러를 가져오는 것
- 텍스처 좌표 범위:
  - 2차원 이미지: 각 x축(s축)과 y축(t축)에서 [0.0, 1.0]



# 텍스처 매핑

- 텍스처 매핑 구현하기
  - 응용 프로그램:
    - 텍스처 생성하고 텍스처 파라미터 설정
    - 각 꼭지점에 대응하는 텍스처 좌표값 설정 후 버텍스 속성 중 한 개로 버텍스 셰이더에 전달
  - 버텍스 셰이더:
    - 텍스처 좌표값을 프래그먼트 셰이더에 전달
  - 프래그먼트 셰이더:
    - 모든 텍스처 좌표를 각 프래그먼트에 보간하여 프레임 버퍼로 전달

# 함수 프로토타입

- 텍스처 생성

- void **glGenTextures** (GLsizei n, GLuint \*textures);
  - n: 생성할 텍스처 개수
  - textures: 텍스처 이름

- 텍스처 바인딩

- void **glBindTexture** ( GLenum target, GLuint texture);
  - target: 텍스처 타겟
    - GL\_TEXTURE\_1D, GL\_TEXTURE\_2D...
  - texture: 텍스처 이름

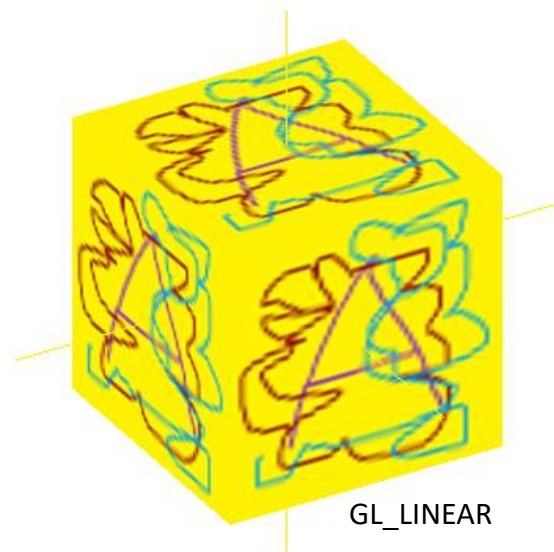
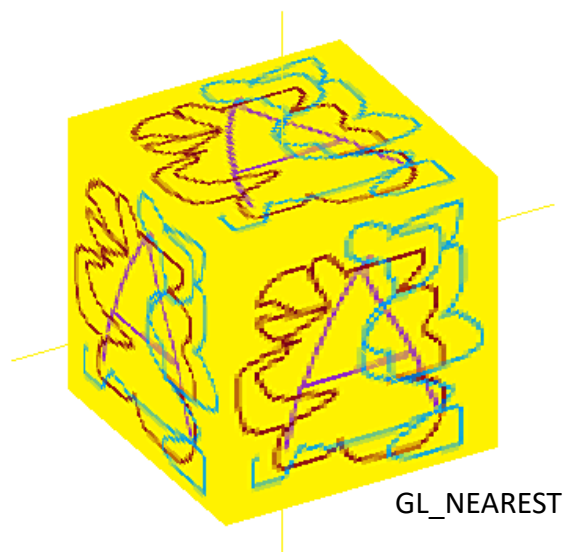
- 텍스처 이미지 정의

- void **glTexImage2D** (GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid \* data);
  - target: GL\_TEXTURE\_2D
  - level: 텍스처 이미지의 상세한 정도 (0으로 설정)
  - internalformat: 각 픽셀에 사용할 컬러 수 (1 ~ 4까지 중 RGB 이면 3, RGBA면 4)
  - width: 텍스처의 폭 (2의 지수승)
  - height: 텍스처 높이(2의 지수승)
  - border: 경계 픽셀 수 (0으로 설정)
  - format: 픽셀 데이터에 대한 포맷 (GL\_RED, GL\_RG, GL\_RGB, GL\_BGR, GL\_RGBA, GL\_BGRA...)
  - type: 각 픽셀 데이터에 대한 데이터 타입 (GL\_UNSIGNED\_BYTE, GL\_BYTE, GL\_UNSIGNED\_SHORT, GL\_SHORT, GL\_UNSIGNED\_INT...)
  - data: 메모리의 실제 픽셀 데이터 값

# 함수 프로토타입

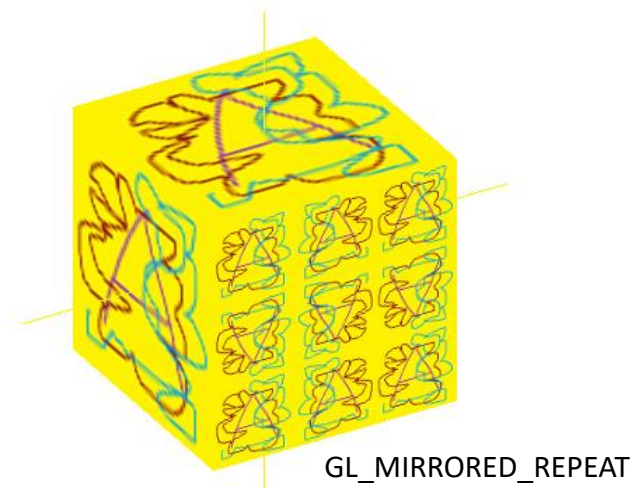
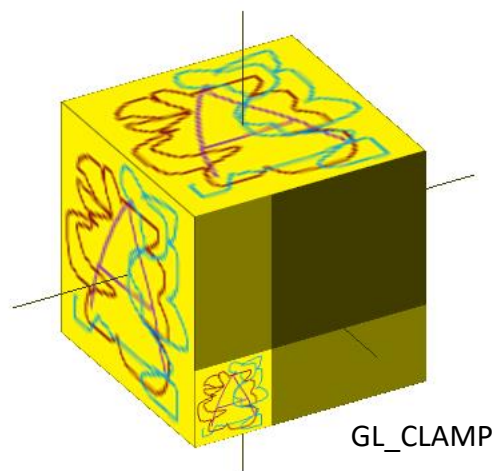
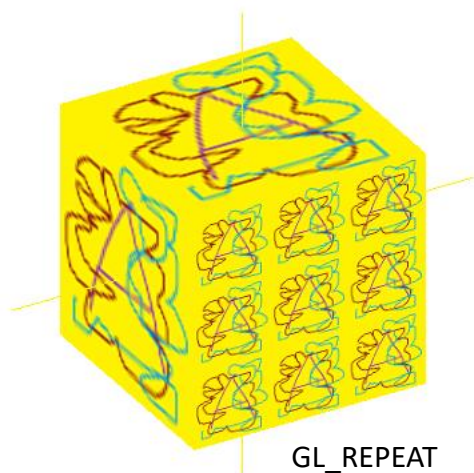
- 텍스처 파라미터 설정

- void **glTexParameter**i (GLenum target, GLenum pname, GLfloat param);
  - target: GL\_TEXTURE\_1D, GL\_TEXTURE\_2D
  - pname: 설정할 텍스처 파라미터
    - GL\_TEXTURE\_MIN\_FILTER, GL\_TEXTURE\_MAG\_FILTER: 텍스처 필터링을 위한 파라미터 설정
    - GL\_TEXTURE\_WRAP\_S, GL\_TEXTURE\_WRAP\_T: 텍스처 랩핑 파라미터 설정
  - param: pname의 스칼라 값
    - GL\_TEXTURE\_MIN\_FILTER (축소 필터), GL\_TEXTURE\_MAG\_FILTER (확대 필터)인 경우:
      - GL\_NEAREST: 가장 가까운 nearest-neighbor 필터링 (픽셀의 근사치 사용) – 선명한 이미지 결과
      - GL\_LINEAR: 이웃한 텍셀의 선형 보간값 – 더 부드러운 결과



# 함수 프로토타입

- void **glTexParameteri** (GLenum target, GLenum pname, GLfloat param);
  - param: pname의 스칼라 값
    - GL\_TEXTURE\_WRAP\_S(S축 래핑), GL\_TEXTURE\_WRAP\_T(T축 래핑)인 경우:
      - GL\_REPEAT: 필요한 경우 텍스처 이미지가 반복
      - GL\_CLAMP: 경계 픽셀이 나타난다
      - GL\_MIRRORED\_REPEAT: 이미지가 뒤집혀서 반복됨



# 함수 프로토타입

- 텍스처 컬러 샘플링하기 (GLSL함수)
    - `gvec4 texture` (`gsampler2D sampler, vec2 p`);
      - GLSL함수로 프래그먼트 셰이더에서 사용
      - 설정한 텍스처 파라미터를 사용하여 해당 컬러값을 샘플링한다.
        - `sampler`: 텍스처 샘플러
        - `p`: 텍스처 좌표
  - 텍스처 유닛:
    - 텍스처의 위치로 셰이더에서 하나 이상의 텍스처를 사용할 수 있도록 해준다.
    - 기본 텍스처 유닛은 0
      - `GL_TEXTURE0, GL_TEXTURE1...`
        - 0번부터 순서대로 선언되어 있다. (`GL_TEXTURE0 + 5` 같은 형식으로 접근 가능)
      - 사용 가능한 최대 개수: `glGetIntegerv` 함수를 사용하여 최대 사용 개수 확인 가능
- ```
int num;  
glGetIntegerv (GL_MAX_TEXTURE_IMAGE_UNITS, &num);
```



# 1) 텍스처 적용하기

## • 좌표값

```
float vertexData[] = {
```

```
    //--- 위치
```

```
    -0.5f, -0.5f, 0.5f,
    0.5f, -0.5f, 0.5f,
    0.5f, 0.5f, 0.5f,
```

```
    //--- 노말
```

```
    0.0, 0.0, 1.0,
    0.0, 0.0, 1.0,
    0.0, 0.0, 1.0,
```

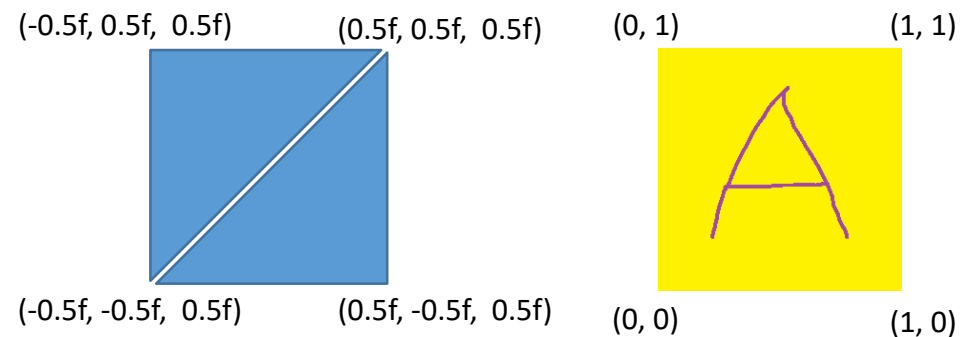
```
    //--- 텍스처 좌표
```

```
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
```

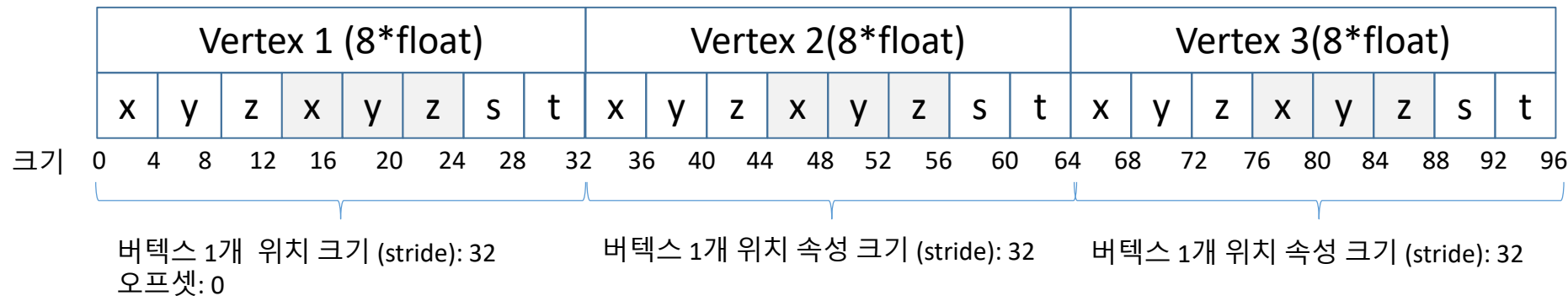
```
    0.5f, 0.5f, 0.5f,
    -0.5f, 0.5f, 0.5f,
    -0.5f, -0.5f, 0.5f,
```

```
    0.0, 0.0, 1.0,
    0.0, 0.0, 1.0,
    0.0, 0.0, 1.0,
```

```
    1.0, 1.0,
    0.0, 1.0,
    0.0, 0.0    };
```



## • 버텍스 포맷:



# 1) 텍스처 적용하기

- 메인 프로그램

- 버텍스 속성 읽기: 위치, 노말값, 텍스처 좌표값
- Vertex buffer에 속성 저장하기

```
void initBuffer () {  
    unsigned int VBO, VAO;  
    glGenVertexArrays (1, &VAO);  
    glGenBuffers (1, &VBO);  
  
    glBindVertexArray (VAO);  
    glBindBuffer (GL_ARRAY_BUFFER, VBO);  
    glBufferData (GL_ARRAY_BUFFER, sizeof (vertexData), vertexData, GL_STATIC_DRAW);  
  
    glVertexAttribPointer (0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);  
    glEnableVertexAttribArray (0);  
    glVertexAttribPointer (1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));  
    glEnableVertexAttribArray (1);  
    glVertexAttribPointer (2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));  
    glEnableVertexAttribArray (2);  
}
```

```
float vertexData[] = {  
    //--- 위치          //--- 노말          //--- 텍스처 좌표  
    -0.5f, -0.5f, 0.5f,    0.0, 0.0, 1.0,    0.0, 0.0,  
    0.5f, -0.5f, 0.5f,    0.0, 0.0, 1.0,    1.0, 0.0,  
    0.5f, 0.5f, 0.5f,     0.0, 0.0, 1.0,    1.0, 1.0,  
  
    0.5f, 0.5f, 0.5f,     0.0, 0.0, 1.0,    1.0, 1.0,  
    -0.5f, 0.5f, 0.5f,    0.0, 0.0, 1.0,    0.0, 1.0,  
    -0.5f, -0.5f, 0.5f,   0.0, 0.0, 1.0,    0.0, 0.0    };
```

//--- 위치 속성

//--- 노말값 속성

//--- 텍스처 좌표 속성

# 1) 텍스처 적용하기

- 메인 프로그램

```
void InitTexture ()
```

```
{
```

```
    unsigned int texture;
```

```
    BITMAP *bmp;
```

```
    glGenTextures (1, &texture);
```

```
    glBindTexture (GL_TEXTURE_2D, texture);
```

```
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

```
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

```
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

```
    unsigned char *data = LoadDIBitmap ( "texture.bmp", &bmp);
```

```
    glTexImage2D (GL_TEXTURE_2D, 0, 3, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
```

```
}
```

```
void DrawScene ()
```

```
{
```

```
    glUseProgram (shaderProgram);
```

```
    glBindVertexArray (VAO);
```

```
    glBindTexture (GL_TEXTURE_2D, texture);
```

```
    glDrawArrays (GL_TRIANGLES, 0, 6);
```

```
}
```

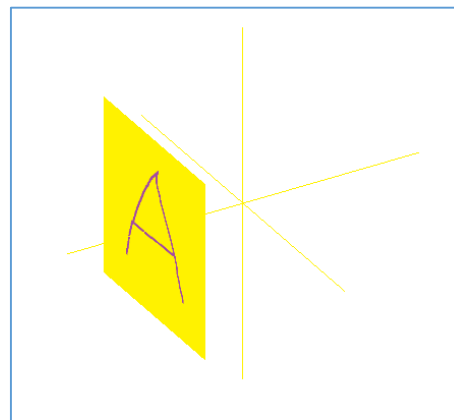
//--- 텍스처 생성

//--- 텍스처 바인딩

//--- 현재 바인딩된 텍스처의 파라미터 설정하기

//--- 텍스처로 사용할 비트맵 이미지 로드하기

//--- 텍스처 이미지 정의



# 1) 텍스처 적용하기

- 버텍스 셰이더

```
#version 330 core
```

```
layout (location = 0) in vec3 vPos;           //--- 위치
layout (location = 1) in vec3 vNormal;        //--- 노말값
layout (location = 2) in vec2 vTexCoord;      //--- 텍스처 좌표
```

```
out vec3 outNormal;
out vec2 TexCoord;
```

```
void main()
{
    gl_Position = vec4(aPos, 1.0);
    outNormal = vNormal;           //--- 노말값 전달
    TexCoord = vTexCoord;         //--- 텍스처 좌표 전달
}
```

- 프래그먼트 셰이더

```
#version 330 core
```

```
out vec4 FragColor;
```

```
in vec3 outNormal;
In vec2 TexCoord;
```

```
uniform sampler2D outTexture;           //--- 텍스처 샘플러
```

```
void main()
{
    FragColor = texture(outTexture, TexCoord);

    //--- 조명을 설정하는 경우:
    //--- FragColor = vec4 (result, 1.0f);
    //--- FragColor = texture(outTexture, TexCoord) * FragColor;
}
```

## 2) 여러 텍스처 사용하기

- 여러 텍스처 사용하기: 한 번에 한 개의 텍스처 사용하기

- 메인 프로그램: 텍스처 속성 설정하기

```
void initBuffer () {  
    unsigned int VBO[2], VAO[2];  
    glGenVertexArrays(2, VAO);  
    glGenBuffers(2, VBO);  
  
    //--- 첫 번째 버텍스 데이터  
    glBindVertexArray(VAO[0]);  
    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);  
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexData), vertexData, GL_STATIC_DRAW);  
  
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);  
    glEnableVertexAttribArray(0);  
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));  
    glEnableVertexAttribArray(1);  
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));  
    glEnableVertexAttribArray(2);  
  
    //--- 두 번째 버텍스 데이터  
    glBindVertexArray(VAO[1]);  
    glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);  
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexData2), vertexData2, GL_STATIC_DRAW);  
  
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);  
    glEnableVertexAttribArray(0);  
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));  
    glEnableVertexAttribArray(1);  
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));  
    glEnableVertexAttribArray(2);  
}
```

```
float vertexData[] = {  
    //--- 위치      //--- 색상      //--- 텍스처 좌표  
    -0.5f, -0.5f, 0.5f,      0.0, 1.0, 0.0,      0.0, 0.0,  
    0.5f, -0.5f, 0.5f,      0.0, 1.0, 0.0,      1.0, 0.0,  
    0.5f, 0.5f, 0.5f,       0.0, 1.0, 0.0,      1.0, 1.0,  
    0.5f, 0.5f, 0.5f,       0.0, 1.0, 0.0,      1.0, 1.0,  
    -0.5f, 0.5f, 0.5f,      0.0, 1.0, 0.0,      0.0, 1.0,  
    -0.5f, -0.5f, 0.5f,     0.0, 1.0, 0.0,      0.0, 0.0};
```

```
float vertexData2[] = {  
    //--- 위치      //--- 색상      //--- 텍스처 좌표  
    -0.5f, -0.5f, -0.5f,     0.0, 1.0, 0.0,      1.0, 0.0,  
    0.5f, -0.5f, -0.5f,     0.0, 1.0, 0.0,      0.0, 1.0,  
    0.5f, 0.5f, -0.5f,      0.0, 1.0, 0.0,      0.0, 0.0,  
    0.5f, 0.5f, -0.5f,      0.0, 1.0, 0.0,      0.0, 1.0,  
    -0.5f, 0.5f, -0.5f,     0.0, 1.0, 0.0,      1.0, 0.0,  
    -0.5f, -0.5f, -0.5f,    0.0, 1.0, 0.0,      1.0, 1.0};
```

//--- 위치 속성

//--- 색상 속성

//--- 텍스처 좌표 속성

//--- 위치 속성

//--- 색상 속성

//--- 텍스처 좌표 속성

## 2) 여러 텍스처 사용하기

- 텍스처 속성 설정

```
void InitTexture ()
{
    unsigned int texture[2];

    //--- texture 1
    glGenTextures (2, &texture);
    glBindTexture (GL_TEXTURE_2D, texture[0]);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    unsigned char *data1 = loadDIBitmap ("A.bmp", &bmp);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

    //--- texture 2
    glBindTexture (GL_TEXTURE_2D, texture[1]);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    unsigned char *data2 = loadDIBitmap ("B.bmp", &bmp);
    glTexImage2D (GL_TEXTURE_2D, 0, 3, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data2);

}
```

## 2) 여러 텍스처 사용하기

- 텍스처 바인딩

```
void drawScene ()  
{
```

```
    glUseProgram (shaderProgram);
```

```
    glBindVertexArray (VAO[0]);
```

```
    glBindTexture (GL_TEXTURE_2D, textures[0]);
```

```
    glDrawArrays (GL_TRIANGLES, 0, 6);
```

```
    glBindVertexArray (VAO[1]);
```

```
    glBindTexture (GL_TEXTURE_2D, textures[1]);
```

```
    glDrawArrays (GL_TRIANGLES, 0, 6);
```

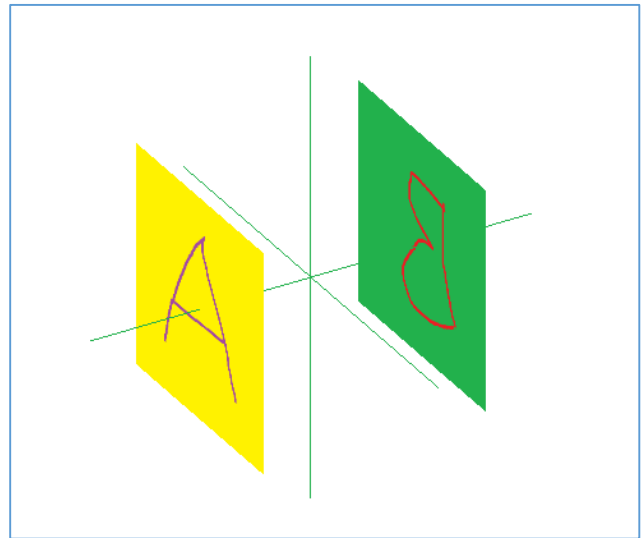
```
}
```

//--- 첫 번째 폴리곤

//--- texture[0]을 사용하여 폴리곤을 그린다.

//--- 두 번째 폴리곤

//--- texture[1]를 사용하여 폴리곤을 그린다.



## 2) 여러 텍스처 사용하기

- 버텍스 셰이더

```
#version 330 core
```

```
layout (location = 0) in vec3 vPos;  
layout (location = 1) in vec3 vColor;  
layout (location = 2) in vec2 vTexCoord;
```

```
out vec3 ourColor;  
out vec2 TexCoord;
```

```
void main()  
{  
    gl_Position = vec4(vPos, 1.0);  
    ourColor = vColor;  
    TexCoord = vTexCoord;  
}
```

- 프래그먼트 셰이더

```
#version 330 core
```

```
out vec4 FragColor;
```

```
in vec3 ourColor;  
in vec2 TexCoord;
```

```
uniform sampler2D outTexture;           //--- 텍스처 샘플러
```

```
void main()  
{  
    FragColor = texture (outTexture, TexCoord);  
}
```



# 텍스처 유닛 (Texture Unit)

- 텍스처 유닛:
  - 텍스처 위치로 셰이더에서 하나 이상의 텍스처를 사용할 수 있도록 해준다.
  - 기본 텍스처 유닛은 0
    - GL\_TEXTURE0, GL\_TEXTURE1...
      - 0번부터 순서대로 선언되어 있다. (GL\_TEXTURE0 + 5 같은 형식으로 접근 가능)
    - 사용 가능한 최대 개수: glGetIntegerv 함수를 사용하여 최대 사용 개수 확인 가능

```
int num;  
glGetIntegerv (GL_MAX_TEXTURE_IMAGE_UNITS, &num);
```
- 하나의 셰이더에서 여러 텍스처를 사용할 수 있다.
  - 여러 샘플러 유니폼을 만든다.
  - 각각의 샘플러는 다른 텍스처 유닛을 참조하도록 한다.
    - **glActiveTexture** 함수 사용: 인자로 텍스처 유닛을 전달하여 해당 텍스처 유닛을 활성화
  - 셰이더에서 sampler uniform이 각각의 텍스처 유닛을 참조하도록 한다.
    - 유니폼 함수를 사용하여 응용 코드에서 직접 sampler uniform값을 설정할 수 있다.
      - 위치 찾기: **glGetUniformLocation** () 함수 사용하여 텍스처 유닛의 위치를 가져온다.
      - 값 설정: **glUniform1i** () 함수 사용하여 유닛 번호를 설정한다.
        - 샘플러 변수가 셰이더 내에서 실제로 정수로 읽혀지는 값은 아니지만, 해당 텍스처 유닛을 설정하는 목적 하에 정수 유니폼처럼 다룰 수 있다.

# 함수 프로토타입

- 텍스처 유닛 활성화
  - void **glActiveTexture** (GLenum texture);
    - 한 개 이상의 텍스처를 사용할 때, 사용할 텍스처를 활성화
      - texture: 활성화할 텍스처 유닛
  - 텍스처 유닛: 텍스처의 위치

### 3) 여러 텍스처를 동시에 사용하기

- 여러 텍스처를 한 개의 셰이더에서 동시에 사용하기

```
void InitTexture ()
{
    unsigned int texture[2];
    glGenTextures (2, &texture);

    //--- texture[0]
    glBindTexture (GL_TEXTURE_2D, texture[0]);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    unsigned char *data1 = loadDIBitmap ("A.bmp", &bmp);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

    //--- texture[1]
    glBindTexture (GL_TEXTURE_2D, texture[1]);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    unsigned char *data2 = loadDIBitmap ("B.bmp", &bmp);
    glTexImage2D (GL_TEXTURE_2D, 0, 3, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data2);

    glUseProgram (shaderProgram);
    int tLocation1 = glGetUniformLocation (shaderProgram, "outTexture1");
    glUniform1i (tLocation1, 0);

    int tLocation2 = glGetUniformLocation (shaderProgram, "outTexture2");
    glUniform1i (tLocation2, 1);
}
```

//--- outTexture1 유니폼 샘플러의 위치를 가져옴  
//--- 샘플러를 0번 유닛으로 설정

//--- outTexture2 유니폼 샘플러의 위치를 가져옴  
//--- 샘플러를 1번 유닛으로 설정

### 3) 여러 텍스처를 동시에 사용하기

- 메인 프로그램

```
void drawScene ()
```

```
{
```

```
    glUseProgram (shaderProgram);
```

```
    glActiveTexture(GL_TEXTURE0);
```

```
    glBindTexture(GL_TEXTURE_2D, texture[0]);
```

```
    glActiveTexture(GL_TEXTURE1);
```

```
    glBindTexture(GL_TEXTURE_2D, texture[1]);
```

```
    glBindVertexArray(VAO[0]);
```

```
    glDrawArrays(GL_TRIANGLES, 0, 6);
```

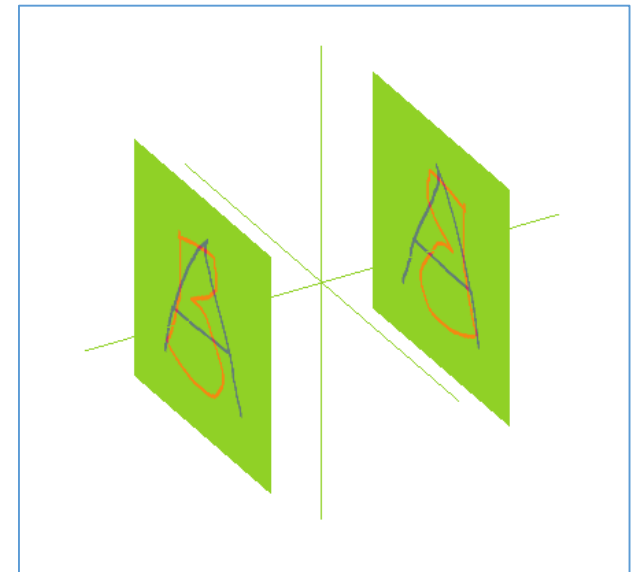
```
    glBindVertexArray(VAO[1]);
```

```
    glDrawArrays(GL_TRIANGLES, 0, 6);
```

```
}
```

```
//--- 유닛 0을 활성화  
//--- texture[0]을 바인딩
```

```
//--- 유닛 1을 활성화  
//--- texture[1]을 바인딩
```



### 3) 여러 텍스처를 동시에 사용하기

- 프래그먼트 셰이더

```
#version 330 core
```

```
out vec4 FragColor;
```

```
in vec3 outColor;
```

```
In vec2 TexCoord;
```

```
uniform sampler2D outTexture 1;          //--- texture sampler outTexture1
```

```
uniform sampler2D outTexture 2;          //--- texture sampler outTexture2
```

```
void main()
```

```
{
```

```
    FragColor = (texture (outTexture1, TexCoord) + texture (outTexture2, TexCoord) ) / 2.0;  //--- 2개의 텍스처를 동시에 사용: 두 개의 텍스처 혼합
```

```
    //--- GLSL 지원 mix 함수
```

```
    //--- genType mix (genType x, genType y, genType a);
```

```
    //--- x와 y 값으로 선형보간한다.
```

```
    //--- 선형 보간 식:  $x * (1-a) + y * a$ 
```

```
    //--- FragColor = mix(texture(Texture1, TexCoord), texture(Texture2, TexCoord), 0.8);
```

```
}
```

# 이미지 파일 로드: bmp 이미지

```
GLubyte * LoadBmp (const char *filename, BITMAPINFO **info)
{
    FILE *fp;
    GLubyte *bits;
    int bitsize, infosize;
    BITMAPFILEHEADER header;

    //--- 바이너리 읽기 모드로 파일을 연다
    if ( (fp = fopen (filename, "rb")) == NULL )
        return NULL;

    //--- 비트맵 파일 헤더를 읽는다.
    if ( fread (&header, sizeof(BITMAPFILEHEADER), 1, fp) < 1 ) {
        fclose(fp);
        return NULL;
    }

    //--- 파일이 BMP 파일인지 확인한다.
    if ( header.bfType != 'MB' ) {
        fclose(fp);
        return NULL;
    }

    //--- BITMAPINFOHEADER 위치로 간다.
    infosize = header.bfOffBits - sizeof (BITMAPFILEHEADER);

    //--- 비트맵 이미지 데이터를 넣을 메모리 할당을 한다.
    if ( (*info = (BITMAPINFO *)malloc(infosize)) == NULL ) {
        fclose(fp);
        return NULL;
    }
}
```

```
//--- 비트맵 인포 헤더를 읽는다.
if ( fread (*info, 1, infosize, fp) < (unsigned int)infosize ) {
    free (*info);
    fclose(fp);
    return NULL;
}

//--- 비트맵의 크기 설정
if ( (bitsize = (*info)->bmiHeader.biSizeImage) == 0 )
    bitsize = ( (*info)->bmiHeader.biWidth *
                (*info)->bmiHeader.biBitCount+7) / 8.0 *
                abs((*info)->bmiHeader.biHeight);

//--- 비트맵의 크기만큼 메모리를 할당한다.
if ( (bits = (unsigned char *)malloc(bitsize) ) == NULL ) {
    free (*info);
    fclose(fp);
    return NULL;
}

//--- 비트맵 데이터를 bit(GLubyte 타입)에 저장한다.
if ( fread(bits, 1, bitsize, fp) < (unsigned int)bitsize ) {
    free (*info); free (bits);
    fclose(fp);
    return NULL;
}

fclose (fp);
return bits;
}
```

# 이미지 파일 로드

- Sean Barrett의 stb\_image.h 라이브러리 사용 가능

- 다운 받기: [https://github.com/nothings/stb/blob/master/stb\\_image.h](https://github.com/nothings/stb/blob/master/stb_image.h)
  - bmp, png, jpg, tga 파일 등을 읽을 수 있음
- 한 개의 헤더 파일 (stb\_image.h)
- 프로젝트가 있는 폴더에 저장

- 메인 프로그램에 헤더파일 추가

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

- 이미지 파일 읽기

```
int widthImage, heightImage, numberOfChannel;
```

```
stbi_set_flip_vertically_on_load (true);    //-- 이미지가 거꾸로 읽힌다면 추가
```

```
unsigned char* data = stbi_load ("A.png", &widthImage, &heightImage, & numberOfChannel, 0);
```

```
...
```

```
stbi_image_free (data);
```

# 블렌딩

- Blending

- 두 가지 색상을 섞어서 그리는 기능
- 투명도 조절
  - 색상 속성에 알파값을 추가한다.
    - 기존의 RGB 값에 A값을 추가하여 RGBA 값으로 사용한다.
  - RGBA색상에서 A값을 조절하여 투명한 효과를 넣는다.
    - Alpha 값이 1.0: 완전 불투명, alpha 값이 0.0: 완전 투명

- openGL의 블렌딩 방정식

- 프레임 버퍼의 색과 물체의 색을 합성함.
- 블렌딩 방정식:
  - $C_{result} = C_{source} * F_{source} + C_{destination} * F_{destination}$
  - 소스값
    - $C_{source}$ : 소스 색 (물체의 색상)
    - $F_{source}$ : 소스 컬러의 블렌딩 값으로 glBlendFunc 함수로 지정
  - 데스티네이션값
    - $C_{destination}$ : 프레임 버퍼에 있는 색
    - $F_{destination}$ : 프레임 버퍼에 적용할 블렌딩 값으로 glBlendFunc 함수로 지정



# 블렌딩

- 기능 활성화:
  - **glEnable (GL\_BLEND);**
    - 블렌딩 기능을 활성화한다.
  - **glDisable (GL\_BLEND);**
    - 블렌딩 기능을 비활성화 한다.
- 블렌딩 함수
  - void **glBlendFunc** (GLenum sFactor, GLenum dFactor)
    - sFactor: 들어오는 픽셀 값 (소스값)에 적용하는 값
    - dFactor: 프레임 버퍼 내에 저장되어 있는 색상 값에 적용하는 값

| Factor                        | Blend Factor Value          | 설명                                    | Alpha Blend Factor |
|-------------------------------|-----------------------------|---------------------------------------|--------------------|
| GL_ZERO                       | (0,0,0)                     | 지수를 0으로 설정                            | 0                  |
| GL_ONE                        | (1,1,1)                     | 지수를 1로 설정                             | 1                  |
| GL_SRC_COLOR                  | ( $R_s, G_s, B_s$ )         | 지수의 원본컬러벡터 Csource로 설정                | $A_s$              |
| GL_ONE_MINUS_SRC_COLOR        | $(1,1,1) - (R_s, G_s, B_s)$ | 지수를 1-Csource로 설정                     | $1 - A_s$          |
| GL_DST_COLOR                  | ( $R_d, G_d, B_d$ )         | 지수를 목적 컬러벡터 Cdestination 로 설정         | $A_d$              |
| GL_ONE_MINUS_DST_COLOR        | $(1,1,1) - (R_d, G_d, B_d)$ | 지수를 1-Cdestination 로 설정               | $1 - A_d$          |
| <b>GL_SRC_ALPHA</b>           | ( $A_s, A_s, A_s$ )         | 지수를 원본 컬러 벡터 Csource의 알파값으로 설정        | $A_s$              |
| <b>GL_ONE_MINUS_SRC_ALPHA</b> | $(1,1,1) - (A_s, A_s, A_s)$ | 지수를 1-원본 컬러 벡터 Csource의 알파값으로 설정      | $1 - A_s$          |
| GL_DST_ALPHA                  | ( $A_d, A_d, A_d$ )         | 지수를 목적 컬러벡터 Cdestination 의 알파값으로 설정   | $A_d$              |
| GL_ONE_MINUS_DST_ALPHA        | $(1,1,1) - (A_d, A_d, A_d)$ | 지수를 1-목적 컬러벡터 Cdestination 의 알파값으로 설정 | $1 - A_d$          |
| GL_CONSTANT_COLOR             | ( $R_c, G_c, B_c$ )         | 지수를 상수 컬러벡터로 설정                       | $A_c$              |
| GL_ONE_MINUS_CONSTANT_COLOR   | $(1,1,1) - (R_c, G_c, B_c)$ | 지수를 1-상수 컬러벡터로 설정                     | $1 - A_c$          |
| GL_CONSTANT_ALPHA             | ( $A_c, A_c, A_c$ )         | 지수를 상수컬러벡터의 알파값으로 설정                  | $A_c$              |
| GL_ONE_MINUS_CONSTANT_ALPHA   | $(1,1,1) - (A_c, A_c, A_c)$ | 지수를 1-상수 컬러벡터 알파값으로 설정                | $1 - A_c$          |

- ( $R_s, G_s, B_s, A_s$ ): 물체의 원본 색상, ( $R_d, G_d, B_d, A_d$ ): 프레임 버퍼에 있는 색상, ( $R_c, G_c, B_c, A_c$ ): 상수 블렌딩 색상

# 블렌딩

- 알파 블렌딩 효과 내기
  - `glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`로 설정
    - $R = A_s * R_s + (1 - A_s) * R_d$
    - $G = A_s * G_s + (1 - A_s) * G_d$
    - $B = A_s * B_s + (1 - A_s) * B_d$
    - $A = A_s * A_s + (1 - A_s) * A_d$
  - 예) 대상 색상값  $C_d = (0.5, 1.0, 1.0, 1.0)$ 이고 소스 색상값  $C_s = (1.0, 0.0, 1.0, 0.3)$ 
    - $R = 0.3 * R_s + (1 - 0.3) * R_d = 0.3*1 + 0.7*0.5 = 0.65$
    - $G = 0.3 * G_s + (1 - 0.3) * G_d = 0.3*0 + 0.7*1 = 0.7$
    - $B = 0.3 * B_s + (1 - 0.3) * B_d = 0.3*1 + 0.7*1 = 1.0$
    - $A = 0.3 * A_s + (1 - 0.3) * A_d = 0.3*0.3 + 0.7*1 = 0.79$
- 반투명 효과를 얻으려면
  - 불투명한 오브젝트를 먼저 그린다.
  - 투명한 오브젝트들을 정렬하여 먼 순서대로 그린다.

# 블렌딩

- 사용 예)

```
void drawScene ()
{
    glClearColor(0.3f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //--- 필요한 불투명 객체 그리기
    ...

    //--- 투명 객체 그리기
    glEnable (GL_BLEND);

    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glDrawArrays (GL_TRIANGLES, 0, 6);

    //--- 블렌딩 해제
    glDisable (GL_BLEND);
}

void InitBuffer ()
{
    ...
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexData), vertexData, GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(float), 0);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(float), (void*)(7 * sizeof(float)));
    glEnableVertexAttribArray(2);
}
```

```
float vertexData[] = {
    //--- 위치      //--- 색상      //--- 노말값
    -0.5f, -0.5f, 0.5f, 0.0, 1.0, 0.0, 0.5, 0.0, 0.0, 1.0,
    0.5f, -0.5f, 0.5f, 0.0, 1.0, 0.0, 0.5, 1.0, 0.0, 1.0,
    0.5f, 0.5f, 0.5f, 0.0, 1.0, 0.0, 0.5, 1.0, 1.0, 1.0,

    0.5f, 0.5f, 0.5f, 0.0, 1.0, 0.0, 0.5, 1.0, 1.0, 1.0,
    -0.5f, 0.5f, 0.5f, 0.0, 1.0, 0.0, 0.5, 0.0, 1.0, 1.0,
    -0.5f, -0.5f, 0.5f, 0.0, 1.0, 0.0, 0.5, 0.0, 0.0, 1.0 };
```

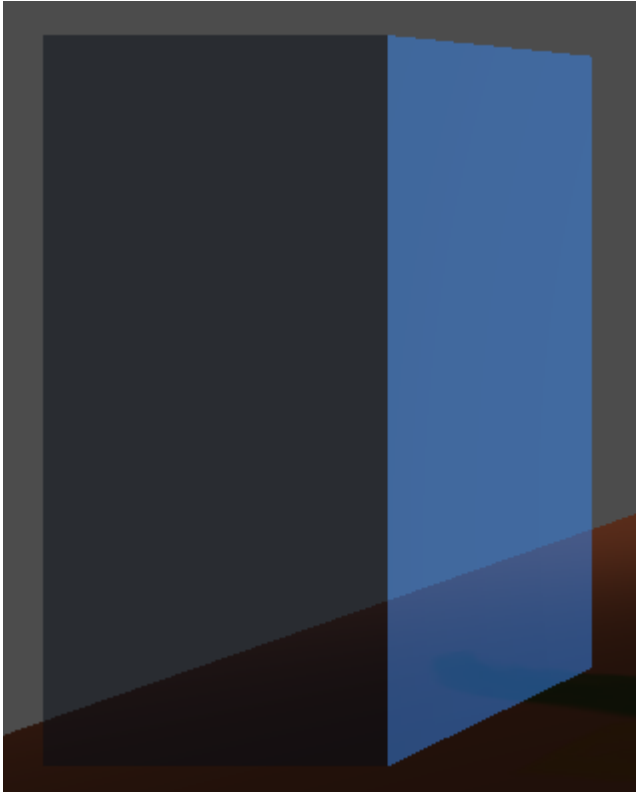
//--- 위치

//--- 색상: RGBA

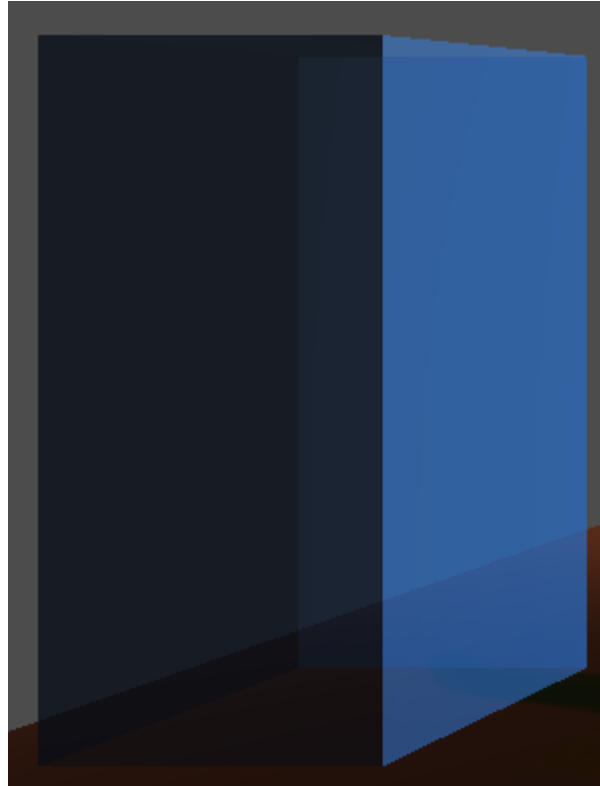
//--- 노말값

# 블렌딩

- 뒷면 제거:
  - glEnable (GL\_CULL\_FACE) 사용/미 사용



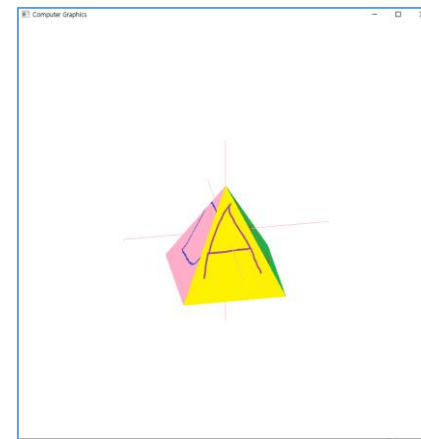
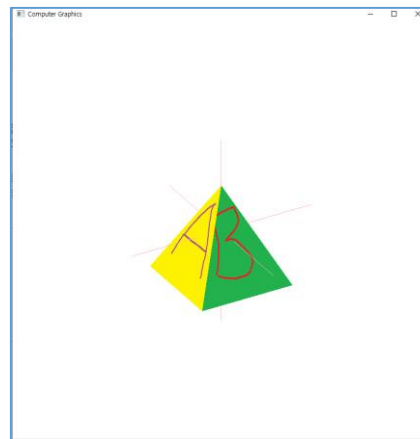
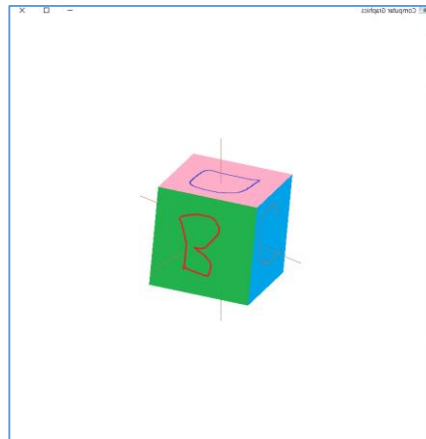
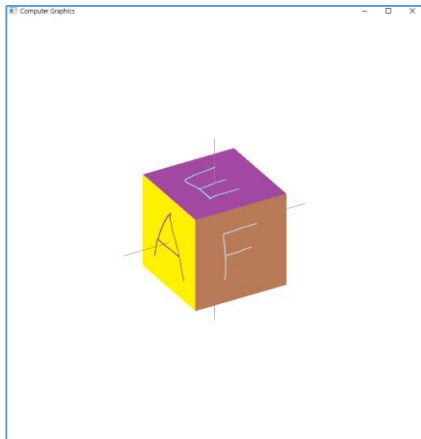
CULL\_FACE 를 한 경우



CULL\_FACE를 하지 않은 경우

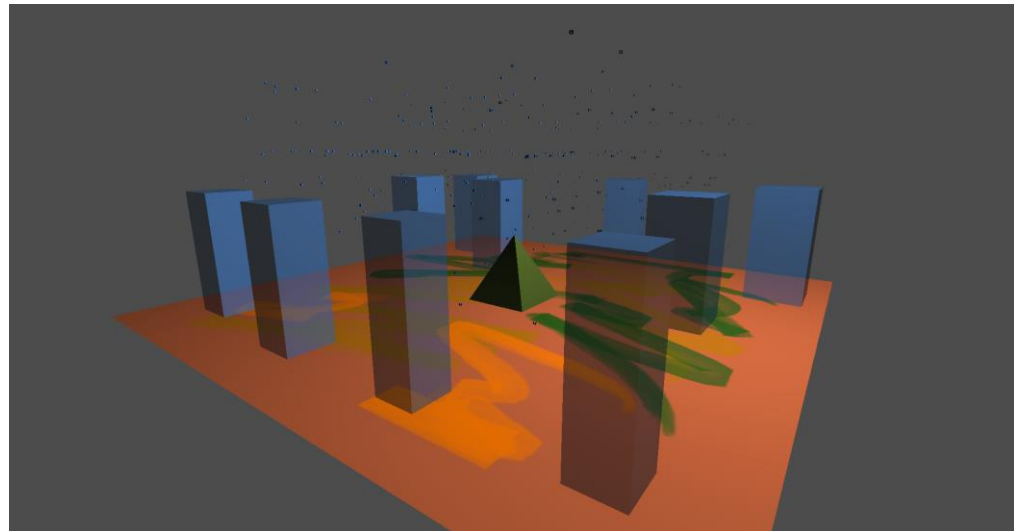
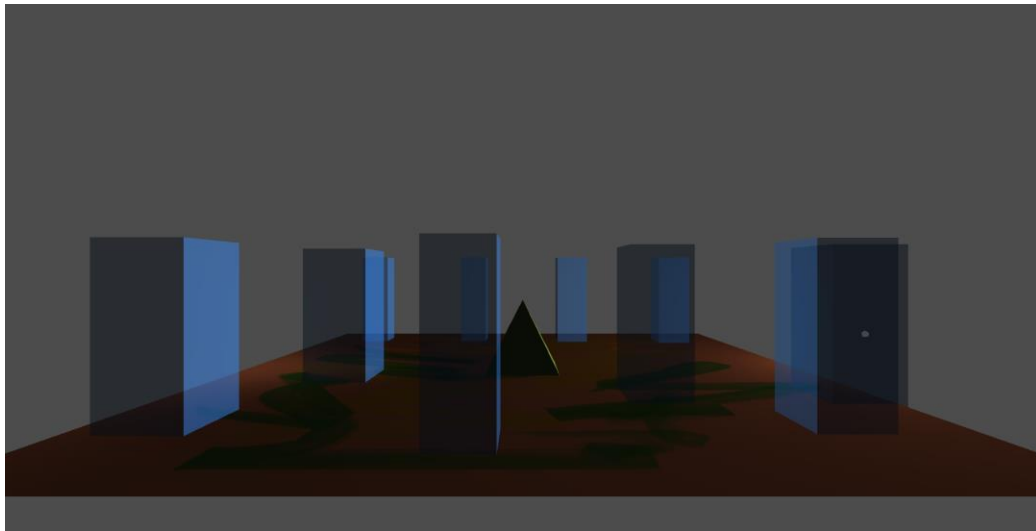
# 실습 26

- 육면체와 사각뿔 (실습 14)에 텍스처 입히기
  - 육면체의 각 면과 사각뿔의 각 면에 다른 텍스처를 입혀본다.
  - 키보드 명령:
    - c: 육면체
    - p: 사각뿔
    - x/X: x축을 중심으로 회전
    - y/Y: y축을 중심으로 회전
    - s: 초기화



## 실습 27

- 눈내리는 실습 (실습 25)에 추가하기
  - 화면에 다양한 크기의 육면체를 임의의 위치에 5개 이상 그린다.
    - 육면체들은 반투명효과를 넣는다.
  - 중앙에 피라미드를 그린다.
  - 키보드 입력
    - m/M: 조명을 켜다/끄다.
    - s/S: 눈이 내린다/멈춘다. (건물과의 충돌체크는 안하고, 바닥과의 충돌체크 함)
    - y/Y: 전체 화면이 y축을 중심으로 회전/멈춘다.



# 실습 28

- 육면체 실습 (실습 26)에 배경 이미지 넣기
  - 육면체를 그리고 육면체에 각각 다른 이미지를 넣는다.
  - 배경 이미지를 넣는다.
    - 배경 이미지 넣기:
      - 변환을 적용하지 않은 화면에 화면 크기의 평면을 화면 맨 뒤 (z축으로 맨 뒤쪽)에 그리고 평면에 배경을 넣는다.
      - 배경 이미지에는 변환을 적용하지 않는다.
  - 실습 26의 키보드 명령어는 그대로 실행하도록 한다.





# 이번 주에는

- 이번주에는
  - 텍스처 매핑
  - 블렌딩