openGL 텍스처 매핑

텍스처 매핑
블렌딩

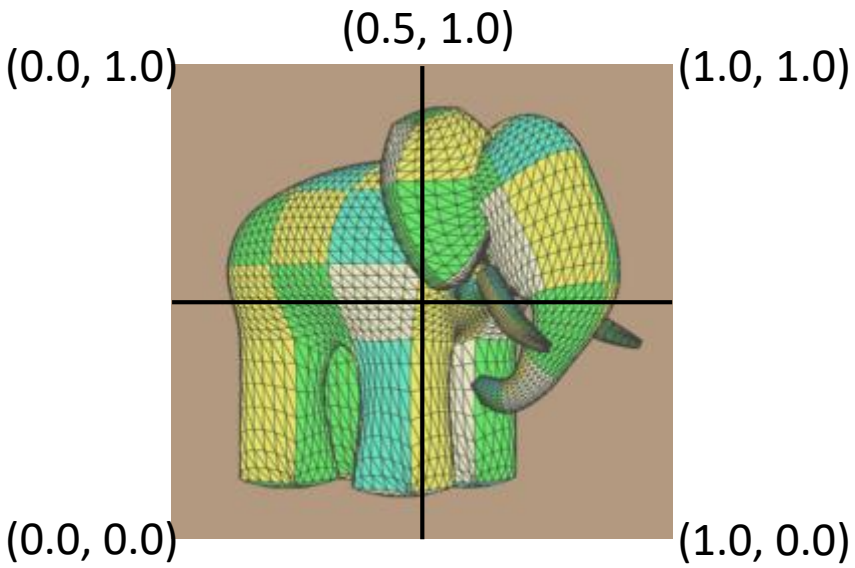
텍스처 매핑

- Modern openGL에서 텍스처 매핑은 프래그먼트 세이딩에서 sampler로 구현됨
  - 샘플러 (sampler): 텍스쳐 접근이 가능하는 특별한 타입들로 텍스쳐 값(texel)에 접근하는데 사용된다.
    - 샘플러는 GLSL에서 접근 가능한 텍스처를 나타내는 변수로 uniform 타입으로 선언한다.
    - 샘플러는 텍스처 색을 반환한다.
    - 텍스쳐 샘플링을 위한 데이터 타입:
      - sampler1D - 1D 텍스쳐을 위한 샘플러
      - sampler2D - 2D 텍스쳐를 위한 샘플러
      - sampler3D - 3D 텍스쳐를 위한 샘플러

- 텍스처 매핑을 위해서
  - 텍스처 생성
    - 이미지를 파일에서 읽기
    - 텍스처에 할당
  - 텍스처 매핑 방법 정의
    - 랩핑, 필터링 방법 등을 정의
    - 텍스처 정의
  - 각 정점에 텍스처 좌표 할당
    - 객체의 각 정점에 텍스처 좌표를 지정

텍스처 매핑

- 텍스처 좌표 범위:
  - 2차원 이미지: 각 x와 y축에서 [0.0, 1.0]



- 응용 프로그램: 각 꼭지점에 대응하는 텍스처 좌표값 설정 후 속성 중 한 개로 버텍스 세이더에 전달
  - 텍스처 매핑을 위한 속성들 설정
- 버텍스 세이더: 해당 좌표값을 프래그먼트 세이더에 전달
- 프래그먼트 세이더: 모든 텍스처 좌표를 각 프래그먼트에 보간

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 타깃
      - 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, GL_TEXTURE_1D_ARRAY…
    - 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

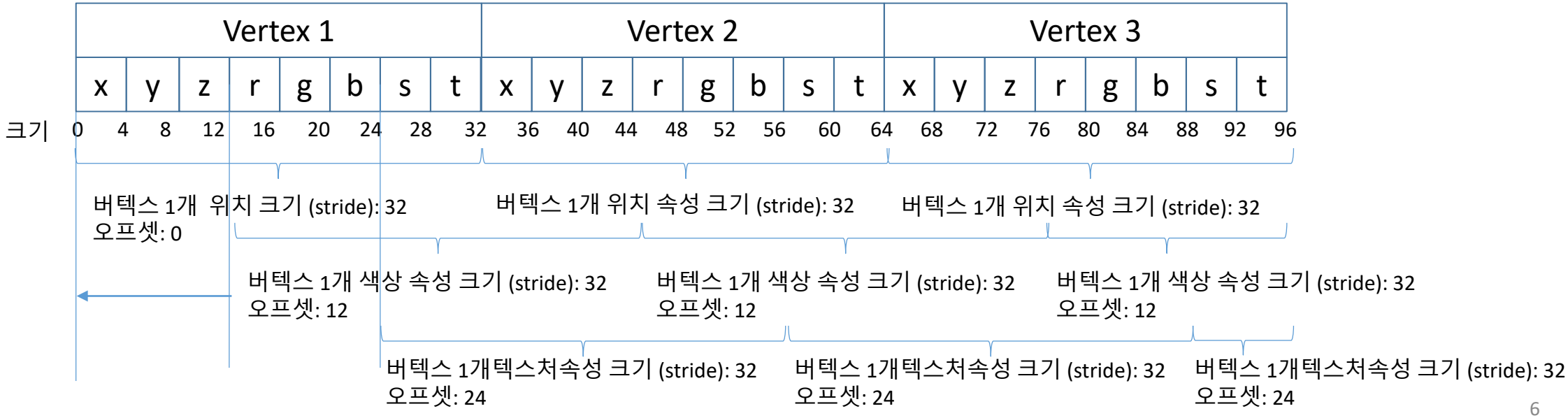
- 텍스처 파라미터 설정
  - void glTexParameteri (GLenum target, GLenum pname, GLfloat param);
    - Target: GL_TEXTURE_1D, GL_TEXTURE_2D
    - Pname: 설정할 텍스처 파라미터
      - GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER: 텍스처 필터링을 위한 파라미터 설정
      - GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T
    - Param: pname의 스칼라 값
      - 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
      - GL_TEXTURE_WRAP_S(S축 랩핑), GL_TEXTURE_WRAP_T(T축 랩핑)인 경우:
        - GL_REPEAT: 필요한 경우 텍스처 이미지가 반복
        - GL_CLAMP: 경계 픽셀이 나타난다

텍스처 적용하기

- 좌표값

```
float vertexData[] = {
    // 위치                  // 컬러                  // 텍스처 좌표
    0.5f,  0.5f, 0.0f,       1.0f, 0.0f, 0.0f,       1.0f, 1.0f,  // 우측 상단
    0.5f, -0.5f, 0.0f,       0.0f, 1.0f, 0.0f,       1.0f, 0.0f,  // 우측 하단
   -0.5f, -0.5f, 0.0f,       0.0f, 0.0f, 1.0f,       0.0f, 0.0f,  // 좌측 하단
   -0.5f,  0.5f, 0.0f,       1.0f, 1.0f, 0.0f,       0.0f, 1.0f   // 좌측 상단
};
```

- 버텍스 포맷:



텍스처 적용하기

- 메인 프로그램
  - 버텍스 속성 읽기: 위치, 색상, 텍스처 좌표값
  - Vertex buffer에 속성 저장하기

```
float vertexData[] = {
   // 위치            // 컬러                    // 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,
   };
```

```
void initBuffer () {
        unsigned int VBO, VAO, EBO;
        glGenVertexArrays (1, &VAO);
        glGenBuffers (1, &VBO);
        glGenBuffers (1, &EBO);

        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);                  //--- 위치 속성
        glEnableVertexAttribArray (0);
        glVertexAttribPointer (1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));  //--- 색상 속성
        glEnableVertexAttribArray (1);
        glVertexAttribPointer (2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));  //--- 텍스처 좌표 속성
        glEnableVertexAttribArray (2);
}
```

텍스처 적용하기

- 메인 프로그램

```
void InitTexture () {
    unsigned int texture;
    BITMAP *bmp;
    glGenTextures (1, &texture);                                          //--- 텍스처 생성
    glBindTexture (GL_TEXTURE_2D, texture);                               //--- 텍스처 바인딩

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);         //--- 현재 바인딩된 텍스처 객체의 속성 설정하기
    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);            //--- 텍스처로 사용할 비트맵 이미지 로드하기

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);       //---텍스처 이미지 정의
}

void DrawScene () {
    glUseProgram (shaderProgram);
    glBindVertexArray (VAO);
    glBindTexture (GL_TEXTURE_2D, texture);
    glDrawArrays (GL_TRIANLES, 0, 6);
}
```

텍스처 적용하기

- 버텍스 세이더

```
#version 330 core
layout (location = 0) in vec3 vPos;                    //--- 위치
layout (location = 1) in vec3 vColor;                  //--- 색상
layout (location = 2) in vec2 vTexCoord;               //--- 텍스처 좌표

out vec3 outColor;
out vec2 TexCoord;

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

텍스처 적용하기

- 프래그먼트 세이더

```
#version 330 core
out vec4 FragColor;

in vec3 outColor;
In vec2 TexCoord;

uniform sampler2D outTexture;                    //--- 텍스처 인덱스: 0번부터 설정

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

        //--- 색상과 혼합하기 위해서는
        // FragColor = texture (outTexture, TexCoord) * vec4 (myColor, 1.0);
}
```

  - gvec4 texture (gsampler2D sampler, vec2 p);
    - 텍스처로부터 텍셀을 검색
    - sampler: 텍스처 샘플러
    - p: 텍스처 좌표

여러 텍스처 사용하기

- 하나의 세이더에서 여러 텍스처를 사용하려면
  - 여러 샘플러 유니폼을 만든다.
  - 각각 다른 텍스처 유닛을 참조하도록 한다.
  - 텍스처 유닛:
    - 텍스처 위치로 기본 텍스처 유닛은 0
    - 세이더에서 하나 이상의 텍스처를 사용할 수 있도록 해준다.
    - glActiveTexture 함수에 텍스처 유닛을 전달하여 호출하여 텍스처 유닛을 활성화한다.
  - 세이더에서 sampler uniform이 각각의 텍스처 유닛을 참조하도록 해야 한다.
    - 유니폼 함수를 사용하여 응용 코드에서 직접 sampler uniform값을 설정할 수 있다.
      - 위치 찾기: glGetUniforLocation () 함수 사용
      - 값 설정: glUniform1i () 함수 사용 (샘플러 변수가 세이더 내에서 실제로 정수로 읽혀지는 값은 아니지만, 해당 텍스처 유닛을 설정하는 목적하에 정수 유니폼처럼 다룰 수 있다)
  - 오픈지엘은 최소 16개의 텍스처 유닛을 가지고 있다
    - GL_TEXTURE0 ~ GL_TEXTURE 15
    - 순서대로 선언되어 있으므로 GL_TEXTURE0 + 5 같은 형식으로 접근 가능

함수 프로토타입

- 함수 프로토타입
  - void glActiveTexture (GLenum texture);
    - 활성화할 텍스처 유닛을 선택한다.
    - Texture: 활성화할 텍스처

여러 텍스처 사용하기

- 사용 예) 여러 텍스처 사용하기
  - 텍스처 속성 설정하기

```
float vertexData[] = {
    // 위치              // 컬러              // 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,
    };

float vertexData2[] = {
     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
    };
```

```
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);                    //--- 위치 속성
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));  //--- 색상 속성
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));  // 텍스처 좌표 속성
        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);                    //--- 위치 속성
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));  //--- 색상 속성
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));  // 텍스처 좌표 속성
        glEnableVertexAttribArray(2);
}
```

여러 텍스처 사용하기

- 응용 프로그램

```
void InitTexture ()
{
        unsigned int texture1, texture2;

        //--- texture 1
        glGenTextures (1, &texture1);
        glBindTexture (GL_TEXTURE_2D, texture1);
        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 (“texture1.bmp”, &bmp);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

        //--- texture 2
        glGenTextures (1, &texture2);
        glBindTexture (GL_TEXTURE_2D, texture2);
        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 (“texture2.bmp”, &bmp);
        glTexImage2D (GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
}
```

여러 텍스처 사용하기

- 텍스처 바인딩

```
void drawScene ()
{
    glUseProgram (shaderProgram);

    glBindVertexArray (VAO[0]);                 //--- 첫 번째 폴리곤
    glActiveTexture (GL_TEXTURE0);
    glBindTexture (GL_TEXTURE_2D, textures1);   //--- texture1을 사용하여 폴리곤을 그린다.
    glDrawArrays (GL_TRIANGLES, 0, 3);

    glBindVertexArray (VAO[1]);                 //--- 두 번째 폴리곤
    glActiveTexture (GL_TEXTURE0);
    glBindTexture (GL_TEXTURE_2D, textures2);   //--- texture2를 사용하여 폴리곤을 그린다.
    glDrawArrays (GL_TRIANGLES, 0, 3);
}
```

여러 텍스처 사용하기

- 버텍스 세이더

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec2 aTexCoord;

out vec3 ourColor;
out vec2 TexCoord;

void main()
{
	gl_Position = vec4(aPos, 1.0);
	ourColor = aColor;
	TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

- 프래그먼트 세이더

```
#version 330 core
out vec4 FragColor;

in vec3 ourColor;
in vec2 TexCoord;

// texture samplers
uniform sampler2D texture1;

void main()
{
	FragColor = texture(texture1, TexCoord);
}
```

여러 텍스처를 동시에 사용하기

- 여러 텍스처를 한 개의 세이더에서 동시에 사용하기

```
void InitTexture ()
{
        …
        int tLocation_1 = glUniformLocation (shaderProgram, “texture1”);
        glUniform1i (tLocation_1, 0);
        int tLocation_2 = glUniformLocation (shaderProgram, “texture2”);
        glUniform1i (tLocation_2, 1);
}

void drawScene ()
{
        glUseProgram (shaderProgram);
        glBindVertexArray(VAO[0]);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture1);
        glActiveTexture(GL_TEXTURE1);
        glBindTexture(GL_TEXTURE_2D, texture2);
        glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

여러 텍스처를 동시에 사용하기

- 프래그먼트 세이더

```
#version 330 core
out vec4 FragColor;

in vec3 outColor;
In vec2 TexCoord;

uniform sampler2D texture1;                    //--- 텍스처 1
uniform sampler2D texture2;                    //--- 텍스처 2

void main()
{
        FragColor = (texture (texture1, TexCoord) +  texture (texture2, TexCoord) ) / 2.0;          //--- 2개의 텍스처를 동시에 사용
}
```

이미지 파일 로드: bmp 이미지

```
GLubyte * LoadDIBitmap (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
  - 한 개의 헤더 파일 (stb_image.h)
  - 프로젝트가 있는 폴더에 저장

  - 메인 프로그램에 헤더파일 추가

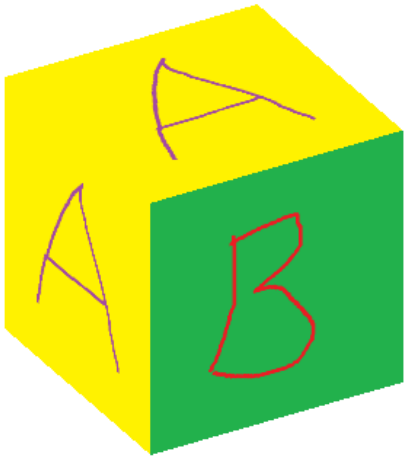
```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

  - 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);
```

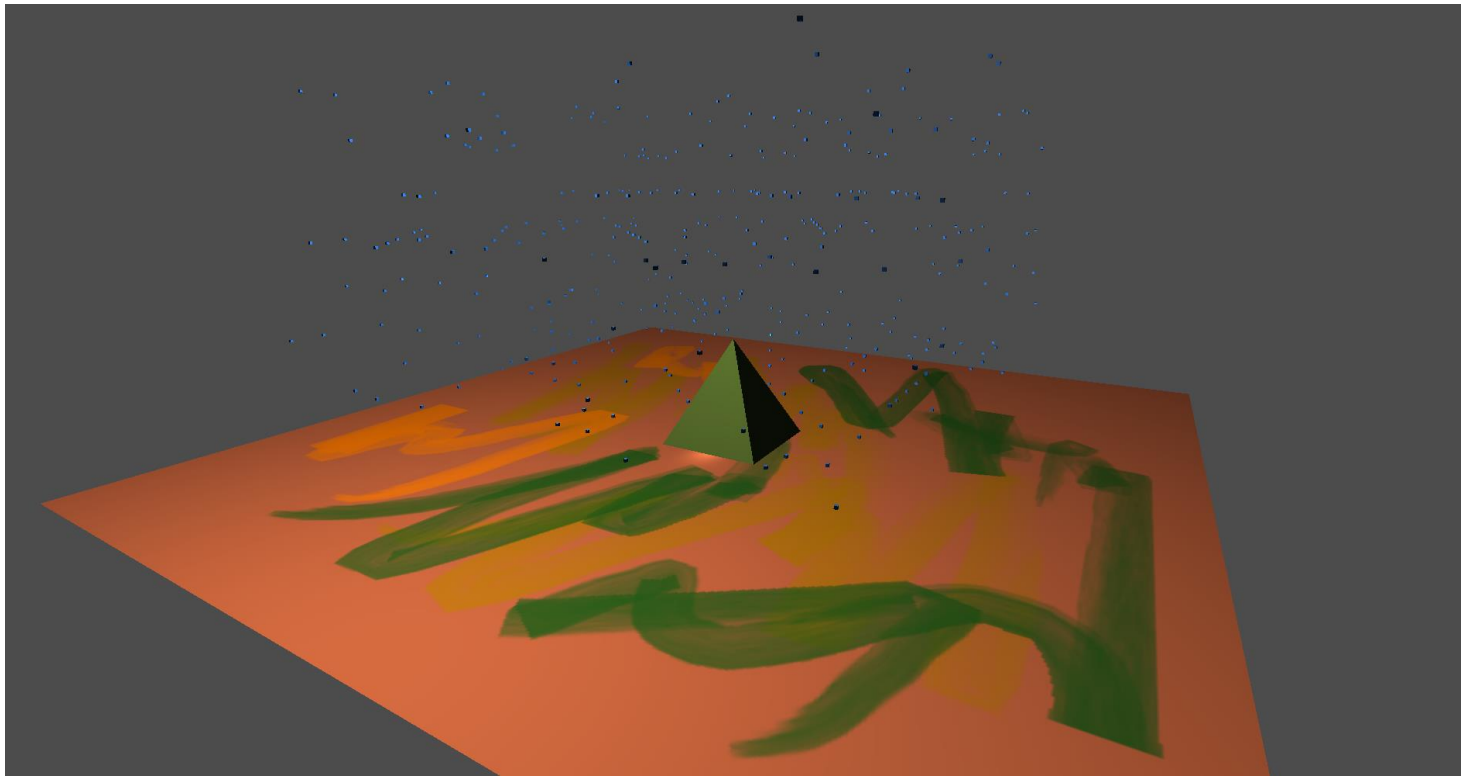
실습 22

- 육면체와 사각뿔에 (실습 15) 텍스처 입히기
  - 육면체의 각 면과 사각뿔의 각 면에 다른 텍스처를 입혀본다. (최소 3개의 다른 텍스처 사용)



실습 23

- 눈내리는 실습 (실습 9 또는 실습 15 또는 실습 21)에 텍스처를 입힌다.
  - 바닥에 텍스처 입히기
  - 중앙의 피라미드에 텍스처 입히기



블렌딩

- Blending
  - 두 가지 색상을 섞어서 그리는 기능
  - 투명도 조절
    - RGBA색상에서 A값을 조절하여 투명한 효과를 넣는다.
      - Alpha 값이 1.0: 완전 불투명, alpha 값이 0.0: 완전 투명
  - openGL의 블렌딩 방정식
    - 원본 색상 (세이더에서 생성한 값)에 원본 인자값을 곱하고 프레임 버퍼의 색상에 대상인자를 곱한 다음에, 그 곱의 결과들을 원하는 블렌딩 공식을 사용하여 혼합
    - $C_{result} = C_{source} * F_{source} + C_{destination} * F_{destination}$
      - C_{source} : 원본 컬러 색상 (객체의 원본 컬러 색상)
      - F_{source} : 원본 인자값
      - $C_{destination}$: 프레임 버퍼 컬러 색상 (컬러 버퍼에 현재 저장된 컬러 색상)
      - $F_{destination}$: 대상 인자값

블렌딩

- 기능 활성화:
  - **glEnable (GL_BLEND);**
    - 블렌딩 기능을 활성화한다.
  - **glDisable (GL_BLEND);**
    - 블렌딩 기능을 비활성화 한다.

블렌딩

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 컬러벡트 Cdestination 로 설정	A_d
GL_ONE_MINUS_DST_COLOR	$(1,1,1) – (R_d, G_d, B_d)$	지수를 1-Cdestination 로 설정	$1 – A_d$
GL_SRC_ALPHA	(A_s, A_s, A_s)	지수를 원본 컬러 벡터 Csource의 알파값으로 설정	A_s
GL_ONE_MINUS_SRC_ALPHA	$(1,1,1) – (A_s, A_s, A_s)$	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

함수 프로토타입

- 블렌딩 함수
  - void glBlendFunc (GLenum sFactor GLenum dFactor)
    - sFactor: 들어오는 픽셀 값에 적용하는 값
    - dFactor: 컬러 버퍼내에 저장되어 있는색상 값에 적용 (목적지의 위치에 있던 화소값)

  - void glBlendFuncSeparate (GLenum srcRGB, GLenum dstRGB, GLenum srcAlpha, GLenum dstAlpha);
    - RGB와 alpha 채널을 다른 옵션으로 따로 설정 가능
    - srcRGB: red, blue, green 블렌딩 값 (초기는 GL_ONE)
    - dstRGB: red, blue, green 목적 블렌딩 값 (초기는 GL_ZERO)
    - srcAlpha: alpha 블렌딩 값 (초기는 GL_ONE)
    - dstAlpha: alpha 목적 블렌딩 값 (초기는 GL_ZERO)

  - void glBlendColor (GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);
    - GL_BLEND_COLOR의 블렌딩 색상을 설정

블렌딩

- 알파 블렌딩 효과 내기
  - glBlend(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 $C_d = (0.5, 1, 1, 1)$이고 소스 색상값 $C_s = (1, 0, 1, 0.3)$
    - $R = 0.3 * R_s + (1 – 0.3) * R_d = 0.3*1 + 0.7*0.5 = 0.65$
    - $G = 0.3 * G_s + (1 – 0.3) * G_d = 0.3*0 + 0.7*1 = 0.7$
    - $B = 0.3 * B_s + (1 – 0.3) * B_d = 0.3*1 + 0.7*1 = 1.0$
    - $A = 0.3 * A_s + (1 – 0.3) * A_d = 0.3*0.3 + 0.7*1 = 0.79$
- 반투명 효과를 얻으려면
  - 불투명한 오브젝트를 먼저 그린다.
  - 투명한 오브젝트들을 정렬한다.
  - 투명한 오브젝트들을 정렬한 순서대로 그린다. (먼 것 먼저)

- 사용 예)

```
glEnable (GL_BLEND);
glBlendColor (0.2f, 0.5, 0.7f, 0.5f);

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

실습 24

- 눈내리는 실습 (실습 21 또는 실습 23)에 추가하기
  - 화면에 육면체를 임의의 위치에 5개 이상 그린다.
    - 그 육면체는 투명 효과를 넣는다.
  - 중앙의 피라미드를 그린다.
  - 키보드를 입력
    - s/S: 눈이 내린다/멈춘다.
    - m/M: 조명을 on/off
    - y/Y: 전체 화면이 y축을 중심으로 회전/멈춘다.