Advanced OpenGL (2)

GPU Programming

2022학년도
2학기

ⓒ nvidia

Framebuffers

Framebuffers

- Framebuffer 는 아래 버퍼들의 결합
  - 컬러 값들을 작성하는 color buffer
  - 깊이 정보를 작성하기 위한 depth buffer
  - 특정한 조건에 의해 해당 fragment들을 폐기하는 stencil buffer

- 기본 framebuffer는 윈도우 창을 생성할 때 함께 만들어짐 (GLFW가 생성)

- 새로운 framebuffer를 직접 만들어 여러가지 효과를 생성 가능

Creating a Framebuffer

- Framebuffer object (FBO) 생성
  - OpenGL의 다른 객체들과 유사하게, glGenFramebuffers() 함수 사용

```
unsigned int fbo;
glGenFramebuffers(1, &fbo);
```

- 이를 바인딩하면 framebuffer를 활성화시킬 수 있음

```
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
```

  - 이후 나오는 모든 framebuffer 읽기 및 쓰기 명령이 현재 바인딩된 framebuffer에 영향을 미침
  - GL_FRAMEBUFFER 대신 GL_READ_FRAMEBUFFER나 GL_DRAW_FRAMEBUFFER 타겟에도 바인딩 가능
    → 특별히 읽기와 쓰기를 구분해야 하는 경우

- 완전한(complete) framebuffer는 아래 요구사항을 만족해야 만들어짐
  - 최소 하나의 buffer(color, depth 혹은 stencil buffer)를 첨부해야 함
  - 그 중 적어도 하나는 color buffer여야 함
  - 모든 첨부된 buffer들(attachments)은 메모리가 할당되어 완전(complete)해야 함
  - 각 buffer들은 MSAA (multi-sample anti-aliasing)을 위한 샘플의 개수가 같아야 함

Creating a Framebuffer

- Framebuffer 첨부가 완전한지는 아래의 명령어로 확인

```
if(glCheckFramebufferStatus(GL_FRAMEBUFFER) == GL_FRAMEBUFFER_COMPLETE)
  // execute victory dance
```

- 이후 모든 렌더링 작업들은 이제 현재 바인딩된 framebuffer에 첨부된 버퍼에 그리게 됨
  - 이는 기본 framebuffer가 아니므로 윈도우 창의 출력에 영향을 주지 않음 (off-screen rendering)

- 다시 메인 윈도우창에 렌더링을 하기 위해서는 0번 바인딩을 통해 기본 framebuffer를 활성화

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

- 모든 framebuffer 작업을 완료하면 framebuffer 객체 제거 `glDeleteFramebuffers(1, &fbo);`

- 결과물이 쓰여질 attachments는 textures 또는 renderbuffer objects 형태로 사용 가능
  - 렌더링 결과로 만들어진 이미지를 어떠한 형태로 접근할 것인가에 따라 달라짐

Texture Attachments

- 텍스처를 첨부하면, 렌더링 작업의 결과가 텍스처 이미지로 저장되므로 shader에서 쉽게 사용 가능
- Framebuffer에 첨부할 텍스처 생성 방법
  - glTexImage2D() 함수에서 텍스처의 크기는 스크린 해상도와 똑같이 지정
  - glTexImage2D() 함수의 맨 마지막 data 파라미터 값을 NULL로 설정 (텍스처를 읽지 않고 쓰겠다는 의미)

```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 800, 600, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);

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

- 전체 화면을 스크린과 다른 크기의 텍스처에 렌더링하고 싶은 경우,
  glViewport()를 framebuffer에 렌더링하기 전에 호출하여 텍스처의 새로운 크기를 인자로 넘김

Texture Attachments

- 텍스처를 framebuffer에 첨부하는 방법 – glFramebufferTexture2D() 함수 사용

```
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture, 0);
```

  - target: 텍스처를 첨부할 타겟 framebuffer (draw, read 혹은 둘 다)
  - attachment: attachment의 유형. GL_COLOR_ATTACHMENTi(i는 숫자), GL_DEPTH_ATTACHMENT, GL_STENCIL_ATTACHMENT, 또는 GL_DEPTH_STENCIL_ATTACHMENT.
  - textarget: 첨부하기 원하는 텍스처의 유형. GL_TEXTURE_2D 또는 cube map의 각 면
  - texture: 첨부할 실제 텍스처
  - level: Mipmap 레벨 (0은 base level)

- 24비트 깊이 + 8비트 스텐실 버퍼를 첨부하는 예

```
glTexImage2D(
  GL_TEXTURE_2D, 0, GL_DEPTH24_STENCIL8, 800, 600, 0,
  GL_DEPTH_STENCIL, GL_UNSIGNED_INT_24_8, NULL
);

glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_TEXTURE_2D, texture, 0);
```

Renderbuffer Object Attachments

- Renderbuffer
  - Byte, 정수, 픽셀 등으로 이루어진 배열이 저장된 실제 buffer
  - 렌더 데이터를 특정 텍스처 포맷으로 변환할 필요가 없이 버퍼에 곧바로 저장하므로 속도가 빠름 (glfwSwapBuffers() 함수 또한 renderbuffer 객체로 구현)
  - 단, 텍스처와 달리 attachment의 데이터를 직접 셰이더를 통해 읽을 수는 없음 (glReadPixels()을 통해 느린 속도로 CPU-side로 읽는 것은 가능)

- RBO (renderbuffer object)의 생성 및 바인딩은 FBO와 유사

```
unsigned int rbo;
glGenRenderbuffers(1, &rbo);
```

```
glBindRenderbuffer(GL_RENDERBUFFER, rbo);
```

- RBO에 저장될 버퍼 설정
  - 24비트 깊이 + 8비트 스텐실 버퍼 객체를 만드는 예

```
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH24_STENCIL8, 800, 600);
```

Renderbuffer Object Attachments

- RBO를 framebuffer에 첨부

```
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER, rbo);
```

- 그럼 어떨 때 텍스처를 첨부하고 어떨 때 renderbuffer를 첨부하나요?
  - 특정 버퍼에서 데이터를 절대 sample할 필요가 없다면 → renderbuffer 사용 (성능에 이득일 수 있음)
  - 특정 버퍼에서 언젠가 컬러값과 같은 데이터를 sample해야 한다면 → 텍스처 사용

Rendering to a Texture

- FBO 객체에 첨부된 color 텍스처에 장면을 렌더링하고,
  이를 다시 화면을 가득 채운 render quad에 렌더링하는 에
  - 시각적 출력은 별도 framebuffer 없이 그렸을 때와 동일하나, 이후 별도의 후처리(post-processing) 가능
- FBO 생성 후 바인딩

```
unsigned int framebuffer;
glGenFramebuffers(1, &framebuffer);
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
```

- 컬러 버퍼 텍스처 생성 후 첨부

```
// generate texture
unsigned int textureColorbuffer;
glGenTextures(1, &textureColorbuffer);
glBindTexture(GL_TEXTURE_2D, textureColorbuffer);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 800, 600, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glBindTexture(GL_TEXTURE_2D, 0);

// attach it to currently bound framebuffer object
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, textureColorbuffer, 0);
```

Rendering to a Texture

- 깊이/스텐실 RBO 생성 후 첨부

```
unsigned int rbo;
glGenRenderbuffers(1, &rbo);
glBindRenderbuffer(GL_RENDERBUFFER, rbo);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH24_STENCIL8, 800, 600);
glBindRenderbuffer(GL_RENDERBUFFER, 0);
```

```
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER, rbo);
```

- FBO가 완전한지 확인한 후, 언바인딩을 통해 잘못된 framebuffer에 대한 렌더링 방지

```
if(glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
        std::cout << "ERROR::FRAMEBUFFER:: Framebuffer is not complete!" << std::endl;
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

- 이제 FBO를 바인딩하는것 만으로, 기본 framebuffer가 아닌 다른 framebuffer에 렌더링 가능
  - 이후 모든 렌더링 명령들은 현재 바인딩된 framebuffer에 영향을 미침

Rendering to a Texture

- 하나의 텍스처에 장면을 그리기 위한 단계
  - 활성화된 framebuffer로 바인딩된 새로운 framebuffer에, 평상시처럼 장면을 렌더링
  - 기본 framebuffer를 바인딩
  - 전체 화면에 맞게 늘린 사각형(render quad)을 그리고,
    여기에 입힐 텍스처로 새로운 framebuffer의 color buffer를 사용

- 셰이더 코드

VS

```
#version 330 core
layout (location = 0) in vec2 aPos;
layout (location = 1) in vec2 aTexCoords;

out vec2 TexCoords;

void main()
{
    gl_Position = vec4(aPos.x, aPos.y, 0.0, 1.0);
    TexCoords = aTexCoords;
}
```

FS

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoords;

uniform sampler2D screenTexture;

void main()
{
    FragColor = texture(screenTexture, TexCoords);
}
```

Rendering to a Texture

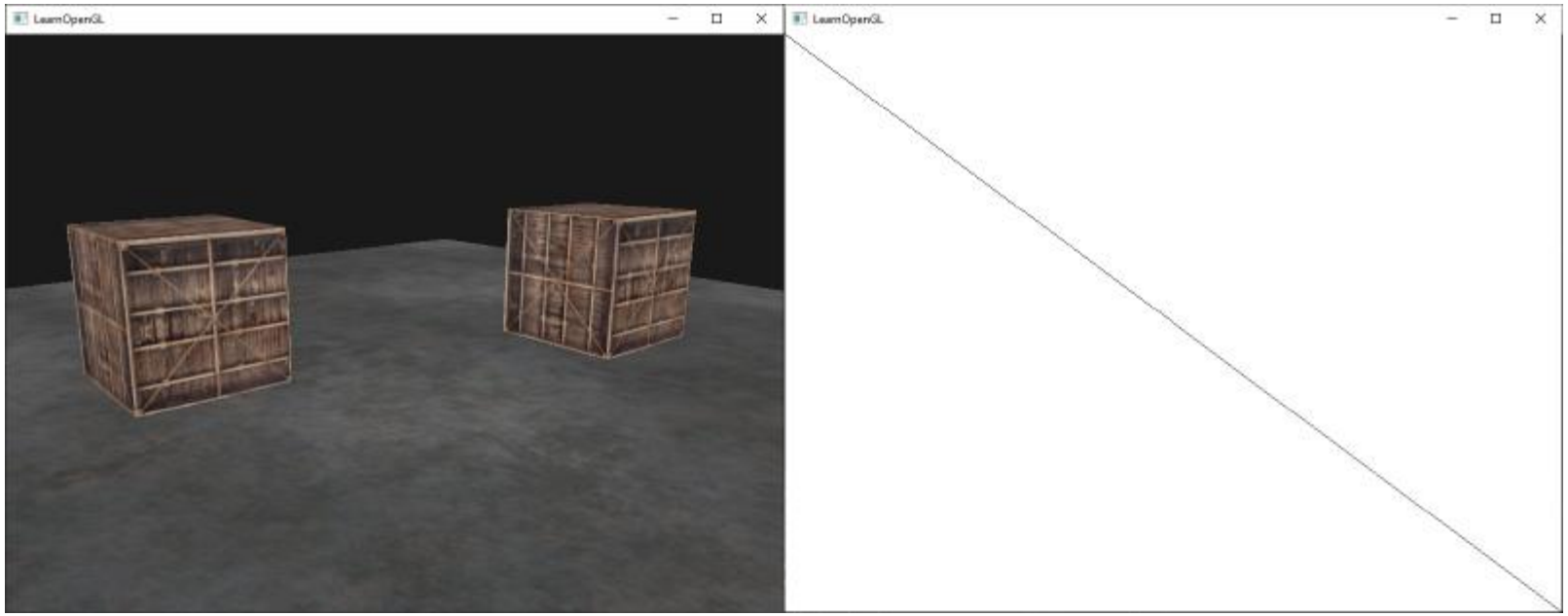
- Render quad를 그리기 위한 VAO 설정

CPU

```
// first pass
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // we're not using the stencil buffer now
glEnable(GL_DEPTH_TEST);
DrawScene();

// second pass
glBindFramebuffer(GL_FRAMEBUFFER, 0); // back to default
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

screenShader.use();
glBindVertexArray(quadVAO);
glDisable(GL_DEPTH_TEST);
glBindTexture(GL_TEXTURE_2D, textureColorbuffer);
glDrawArrays(GL_TRIANGLES, 0, 6);
```

Rendering to a Texture

- 결과 화면
  - Depth testing 챕터에서 본 것과 결과 동일 (좌)
  - 하지만 wireframe을 그리도록 하면 단지 삼각형 두 개만 그려진 것을 확인 가능 (우)



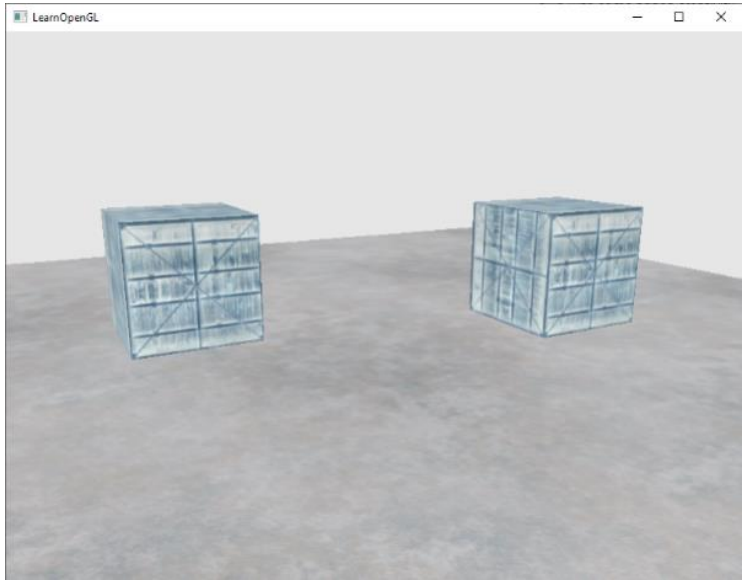
- 전체 코드
  - Code Viewer. Source code: src/4.advanced_opengl/5.1.framebuffers/framebuffers.cpp (learnopengl.com)

Post-processing

- 이제 텍스처 데이터를 조작하여 흥미로운 후처리(post-processing) 효과를 생성 가능

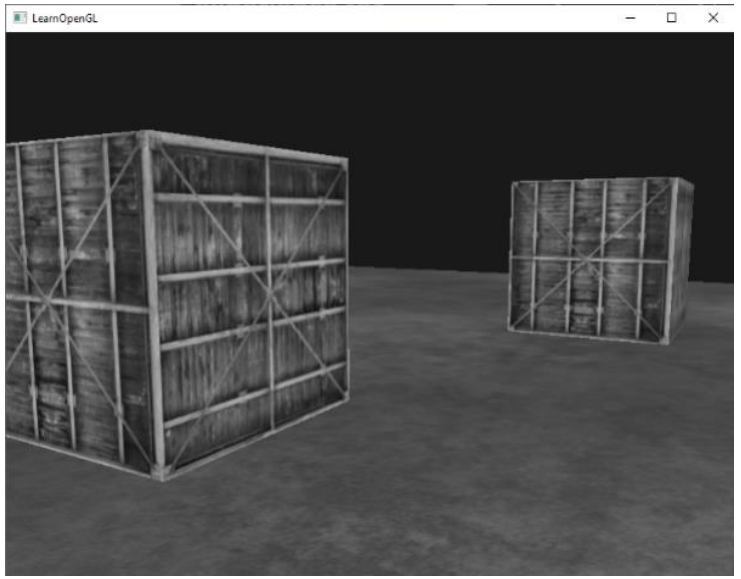
- 반전 (Inversion)

```
void main()                                                    FS
{
    FragColor = vec4(vec3(1.0 - texture(screenTexture, TexCoords)), 1.0);
}
```



Post-processing

- 흑백 (grayscale)
  - 사람의 눈은 녹색→빨강→파랑 순으로 민감하게 반응
  - 이를 이용하여, RGB 값에 각각 다른 weight를 부여하여 더하면 컬러를 흑백으로 만들 수 있음



FS

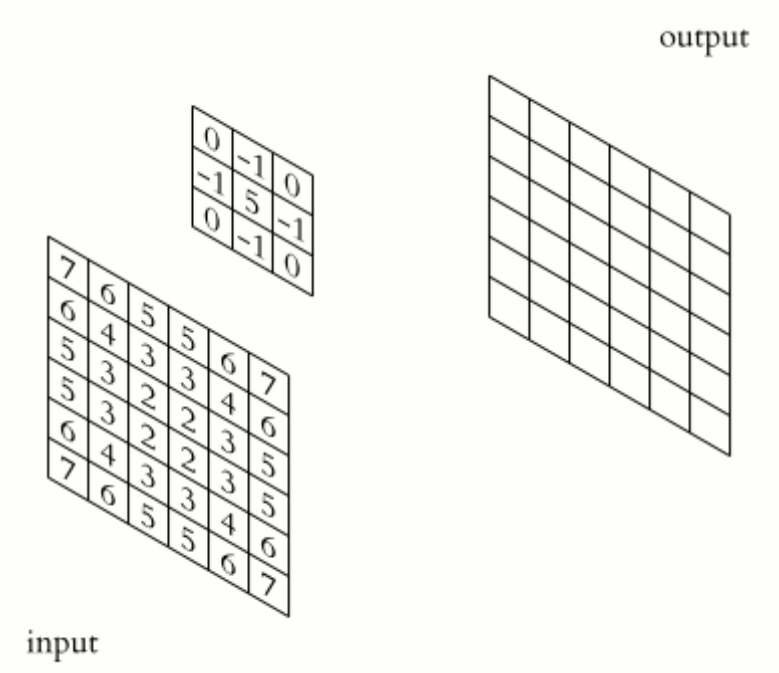
```
void main()
{
    FragColor = texture(screenTexture, TexCoords);
    float average = 0.2126 * FragColor.r + 0.7152 * FragColor.g + 0.0722 * FragColor.b;
    FragColor = vec4(average, average, average, 1.0);
}
```

Post-processing

- 텍스처 이미지에 대한 post-processing의 장점
  - 텍스처의 다른 부분으로부터 컬러 값을 sample할 수 있음
  - 예: 현재 좌표 주변의 작은 영역의 값을 가져와 결합

- Kernel effects
  - Kernel (또는 convolution matrix)는 현재 픽셀에 convolution(합성곱)을 수행할 때 쓰이는 행렬임
  - 커널 행렬의 값을 현재 픽셀을 중심으로 한 주변 픽셀들에 곱한 후, 이를 모두 더해 현재 픽셀값에 반영
  - 딥러닝에서 많이 쓰이는 CNN(convolutional neural networks)도 이러한 원리를 반영하여 특징을 추출
  - 오른쪽 예시는 8개 주변부 픽셀에 2를 곱하고, 현재 픽셀에 -15를 곱하는 커널 행렬
  - 커널 행렬의 모든 값을 더하면 보통 1이 나옴 (1이 아니면 결과 텍스처가 전체적으로 어두워지거나 밝아지게 됨)

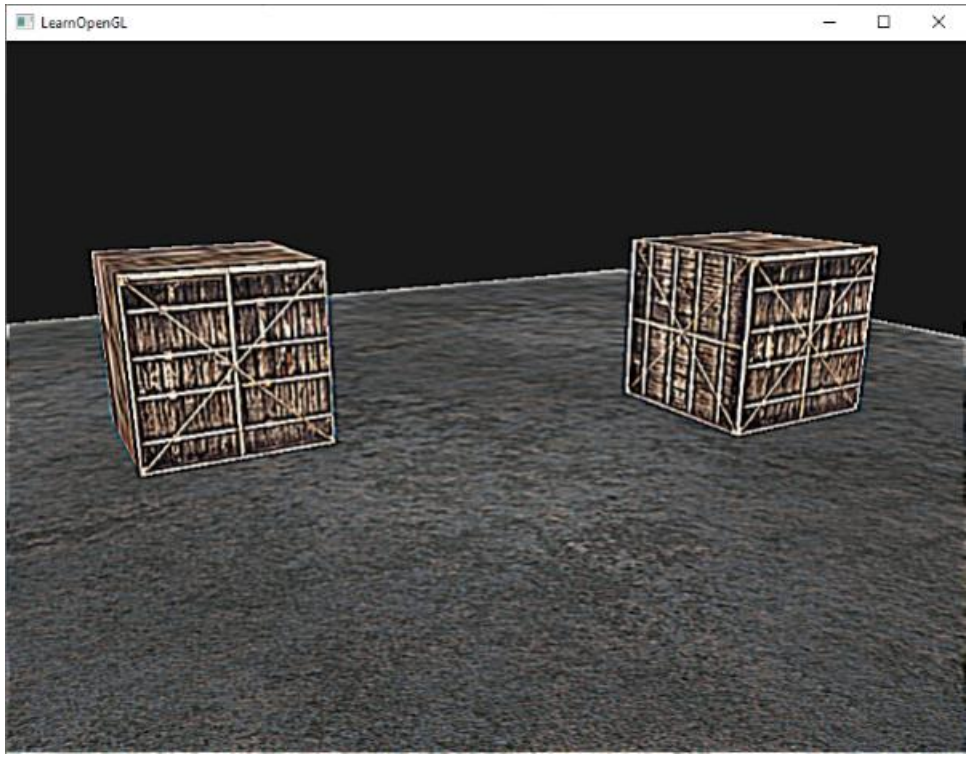
$$\begin{bmatrix} 2 & 2 & 2 \\ 2 & -15 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$



Kernel (image processing) - Wikipedia

Post-processing

- Kernel effects (cont.)
  - 3x3 sharpen kernel의 적용 예
  - 약에 취해서 보는 것 같은 (?) 오묘한 효과



FS

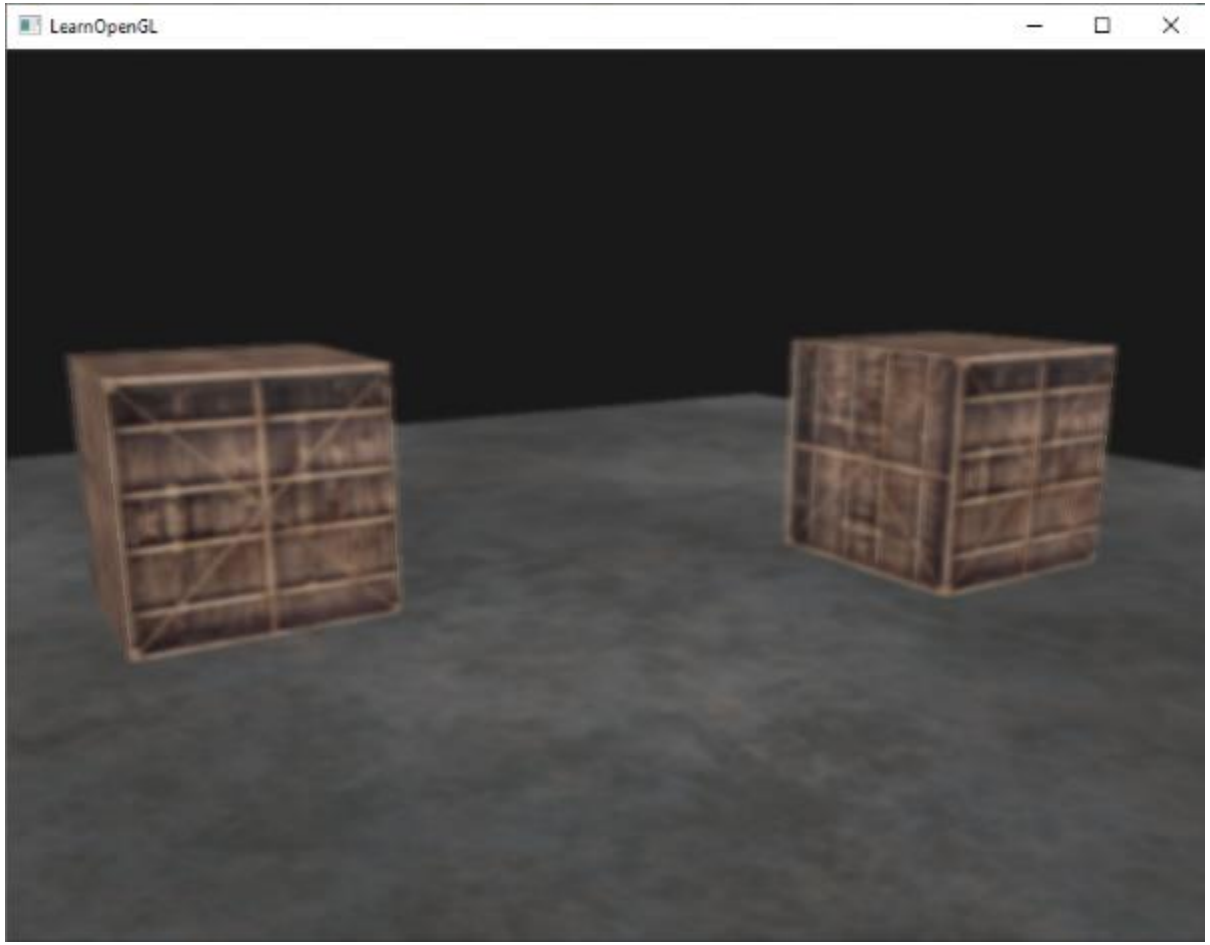
```
const float offset = 1.0 / 300.0;

void main()
{
    vec2 offsets[9] = vec2[](
        vec2(-offset,  offset), // top-left
        vec2( 0.0f,    offset), // top-center
        vec2( offset,  offset), // top-right
        vec2(-offset,  0.0f),   // center-left
        vec2( 0.0f,    0.0f),   // center-center
        vec2( offset,  0.0f),   // center-right
        vec2(-offset, -offset), // bottom-left
        vec2( 0.0f,   -offset), // bottom-center
        vec2( offset, -offset)  // bottom-right
    );

    float kernel[9] = float[](
        -1, -1, -1,
        -1,  9, -1,
        -1, -1, -1
    );

    vec3 sampleTex[9];
    for(int i = 0; i < 9; i++)
    {
        sampleTex[i] = vec3(texture(screenTexture, TexCoords.st + offsets[i]));
    }
    vec3 col = vec3(0.0);
    for(int i = 0; i < 9; i++)
        col += sampleTex[i] * kernel[i];

    FragColor = vec4(col, 1.0);
}
```

Post-processing

- Blur (부드럽게/흐릿하게)
  - Kernel을 조작하여 blur 효과도 낼 수 있음
  - 현재 좌표를 중심으로, 주변부 픽셀을 위치에 비례한 가중치에 따라 섞음 (Gaussian blur)
  - 플레이어가 술에 취하거나 안경을 벗은 경우, 나중에 소개할 bloom 효과, 또는 초점이 안 맞는 영역에 대해 사용 가능

```
float kernel[9] = float[](
    1.0 / 16, 2.0 / 16, 1.0 / 16,
    2.0 / 16, 4.0 / 16, 2.0 / 16,
    1.0 / 16, 2.0 / 16, 1.0 / 16
);
```
FS

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} /16$$

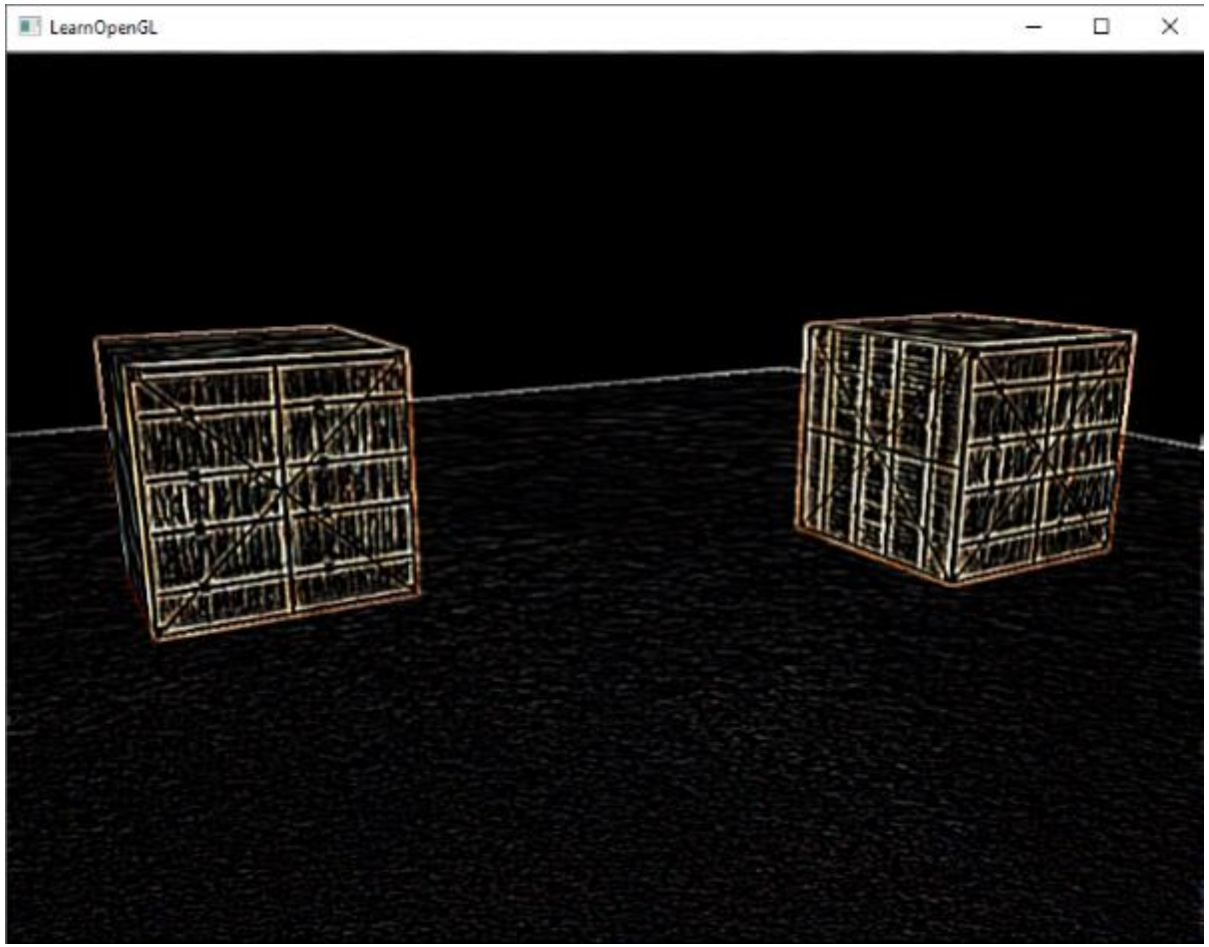


Post-processing

- Edge detection
  - 모서리를 하이라이트하고 나머지들은 어둡게 (sharpen과 형태가 유사하지만 부호가 반대)

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- 그 외에도 커널 조작에 따라 포토샵과 같은 이미지 에디터에서 제공하는 다양한 효과를 생성 가능
  - 이러한 도구들도 빠른 속도를 위해 이미지 프로세싱시 GPU를 사용하는 경향에 있음

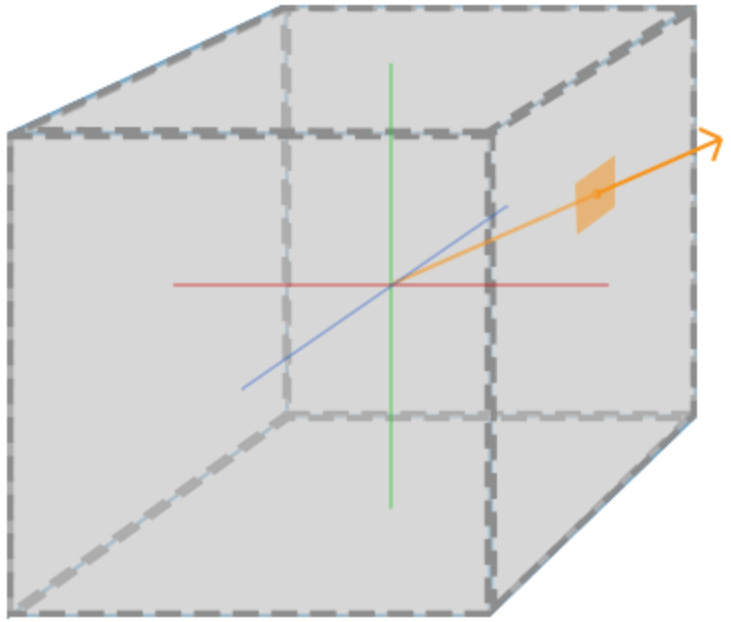


ⓒ nvidia

Cubemaps

Cubemaps

- Cubemap
  - 기본적으로 6면체(cube)의 각 면을 형성하는 2D 텍스처들을 포함하고 있는 텍스처
  - 1x1x1 큐브에서 방향 벡터를 사용하여 indexing 및 sampling 가능
  - 방향 벡터의 크기와 상관 없이,
    OpenGL에서는 이 방향과 맞닿는 해당 텍셀을 검색하여 반환



- Cubemap이 입혀진 cube를 렌더링시
  - 방향 벡터는 cube의 (보간된) vertex 위치와 유사
  - 만약 이 큐브가 원점에 존재한다면,
    이 큐브의 실제 위치 벡터들을 사용하여 cubemap을 샘플링 가능
  - 이 큐브의 (보간된) vertex 위치를 텍스처 좌표로 얻을 수 있음
  - 그 결과 cubemap의 적절한 각 face(면)를 접근할 수 있는
    텍스처 좌표를 얻을 수 있음

Creating a Cubemap

- 텍스처 생성 및 텍스처 타겟으로의 바인딩

```
unsigned int textureID;
glGenTextures(1, &textureID);
glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
```

- Cubemap의 각 면마다 glTexImage2D() 함수를 6번 호출
  - 6개의 특별한 텍스처 타겟 사용
    (다른 OpenGL enum과 같이, 각 target 매크로는 선형 증가)

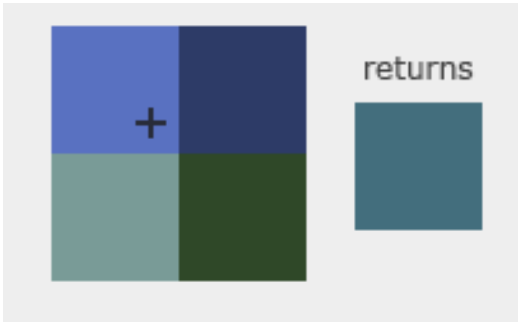
Texture target	Orientation
GL_TEXTURE_CUBE_MAP_POSITIVE_X	Right
GL_TEXTURE_CUBE_MAP_NEGATIVE_X	Left
GL_TEXTURE_CUBE_MAP_POSITIVE_Y	Top
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y	Bottom
GL_TEXTURE_CUBE_MAP_POSITIVE_Z	Back
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z	Front

```
int width, height, nrChannels;
unsigned char *data;
for(unsigned int i = 0; i < textures_faces.size(); i++)
{
    data = stbi_load(textures_faces[i].c_str(), &width, &height, &nrChannels, 0);
    glTexImage2D(
        GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
        0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data
    );
}
```

Creating a Cubemap

- Wrapping과 filtering 모드 지정
  - GL_TEXTURE_WRAP_R이 들어간 이유는 텍스처의 3번째 차원에 대한 wrapping 처리를 위함

```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
```



GL_LINEAR

- 이후 이 cubemap을 사용할 object를 그리기 전에, 해당 텍스처 유닛을 활성화하고 cubemap을 바인딩

Creating a Cubemap

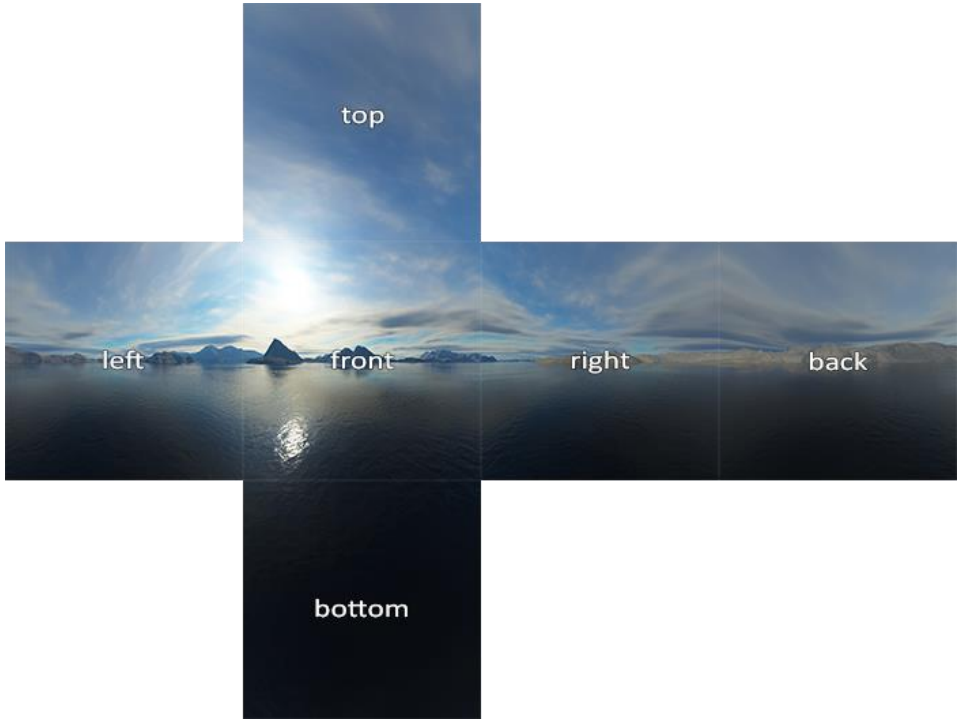
- Fragment shader 내부에서는 다른 샘플러 타입인 samplerCube를 사용해야 함
  - texture 함수를 사용하여 샘플링을 하지만, vec2 대신에 vec3의 방향 벡터 사용

```
in vec3 textureDir; // direction vector representing a 3D texture coordinate
uniform samplerCube cubemap; // cubemap texture sampler

void main()
{
    FragColor = texture(cubemap, textureDir);
}
```
FS

Skybox

- Skybox
  - 전체 scene을 둘러싸고 주변 환경에 대한 6개 이미지를 가지고 있는 큰 큐브
  - 플레이어가 실제로 그 환경(산, 구름, 별이 빛나는 밤하늘 등) 안에 있는 듯한 착각을 들게 함

엘더스크롤 3의 스크린샷



skybox 큐브맵
https://learnopengl.com/img/textures/skybox.zip

Loading a Skybox

```
unsigned int loadCubemap(vector<std::string> faces)
{
    unsigned int textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);

    int width, height, nrChannels;
    for (unsigned int i = 0; i < faces.size(); i++)
    {
        unsigned char *data = stbi_load(faces[i].c_str(), &width, &height, &nrChannels, 0);
        if (data)
        {
            glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
                         0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data
            );
            stbi_image_free(data);
        }
        else
        {
            std::cout << "Cubemap tex failed to load at path: " << faces[i] << std::endl;
            stbi_image_free(data);
        }
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);

    return textureID;
}
```

- 각 면의 파일을 vector에 넣어 전달

```
vector<std::string> faces;
{
    "right.jpg",
    "left.jpg",
    "top.jpg",
    "bottom.jpg",
    "front.jpg",
    "back.jpg"
};
unsigned int cubemapTexture = loadCubemap(faces);
```

Displaying a Skybox

- 큐브를 위한 또 다른 VAO, VBO 및 vertex set 필요

- 큐브의 위치를 텍스처 좌표로 사용하여 큐브맵 샘플링 가능
  - 큐브가 원점에 위치해 있을 때, 각 위치 벡터들은 원점으로부터의 방향 벡터와 동일
  - 따라서 텍스처 좌표 대신 오직 위치 벡터만을 VS에서 FS에 제공하면 됨

VS
```
#version 330 core
layout (location = 0) in vec3 aPos;

out vec3 TexCoords;

uniform mat4 projection;
uniform mat4 view;

void main()
{
    TexCoords = aPos;
    gl_Position = projection * view * vec4(aPos, 1.0);
}
```

FS
```
#version 330 core
out vec4 FragColor;

in vec3 TexCoords;

uniform samplerCube skybox;

void main()
{
    FragColor = texture(skybox, TexCoords);
}
```

CPU
```
float skyboxVertices[] = {
    // positions
    -1.0f,  1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    -1.0f,  1.0f, -1.0f,
     1.0f,  1.0f, -1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
     1.0f, -1.0f,  1.0f
};
```

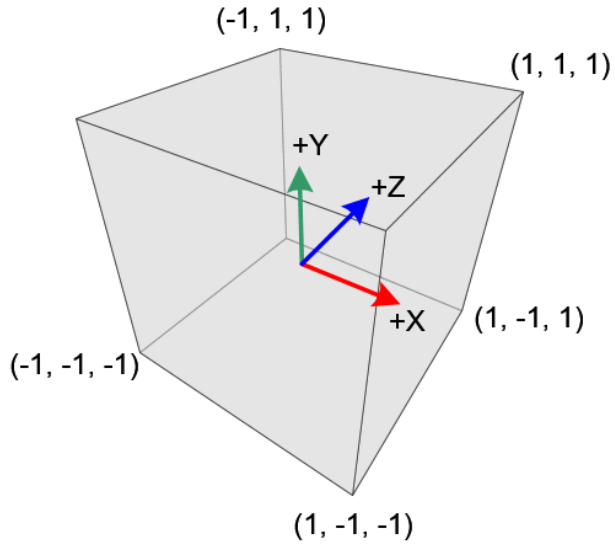
An Optimization

- Skybox를 먼저 렌더링하면, 앞의 물체에 의해 가려지는 영역의 fragment shader 계산이 버려지게 됨
- Skybox를 맨 마지막에 렌더링하면, early z-test를 통과한 fragment만 렌더링 가능!
- gl_Position의 z 요소를 w 로 바꿔주면, perspective division에 의해 NDC의 z값이 1.0 (최대 깊이 값)이 됨

```
void main()
{
    TexCoords = aPos;
    vec4 pos = projection * view * vec4(aPos, 1.0);
    gl_Position = pos.xyww;
}
```

VS

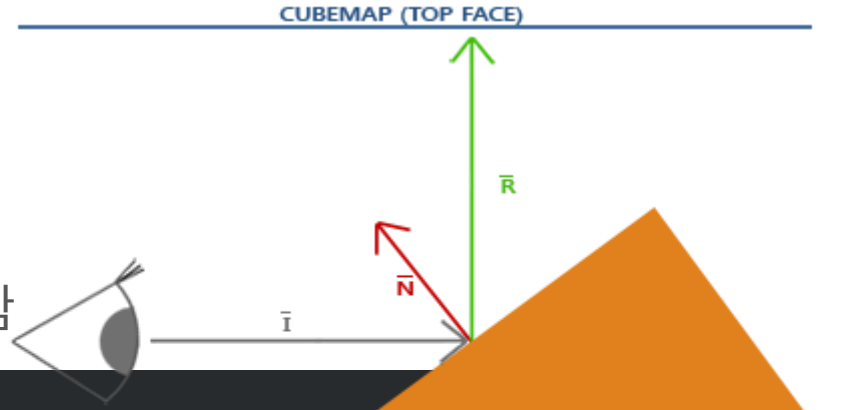
- 깊이 함수 (depth function)은
  GL_LESS 대신에 GL_LEQUAL로 설정
  - Skybox 영역이 깊이 검사를 통과해야 하기 때문



OpenGL Projection Matrix (songho.ca)

Environment Mapping - Reflection

- 환경 매핑(environment mapping)
  - Cubemap이 매핑된 환경 object를 이용해,
    특정 물체에 빛을 반사 또는 굴절시키는 효과를 주는 방법
- 반사 (Reflection)
  - 시점의 각도에 따라 주변을 반사
  - Diffuse/specular lighting 때와 유사하게, 입사벡터 I로 방향벡터 R을 구함



VS

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 Normal;
out vec3 Position;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    Normal = mat3(transpose(inverse(model))) * aNormal;
    Position = vec3(model * vec4(aPos, 1.0));
    gl_Position = projection * view * vec4(Position, 1.0);
}
```

FS

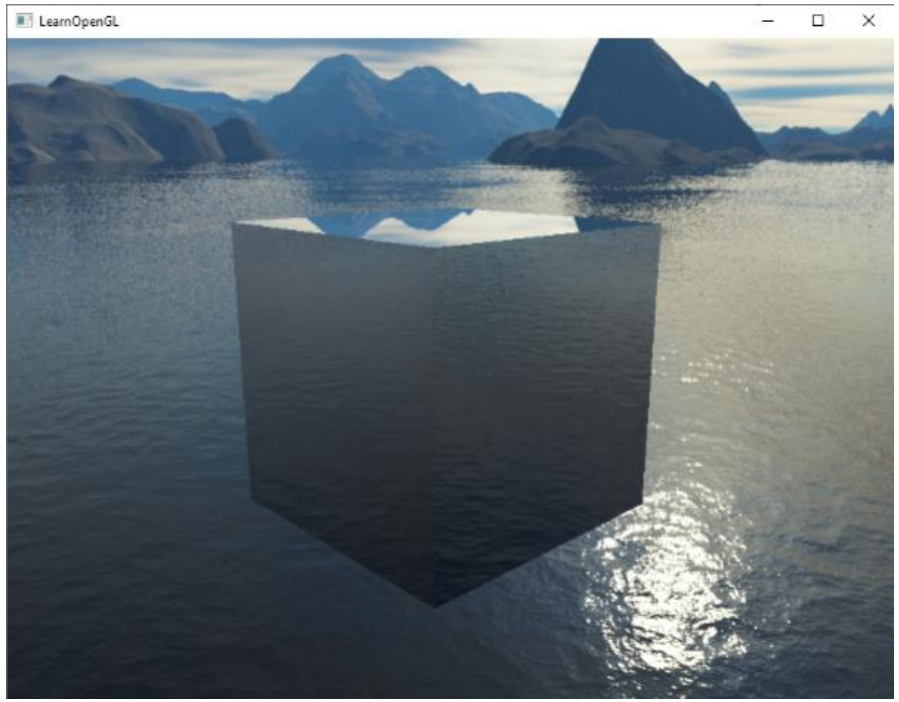
```
#version 330 core
out vec4 FragColor;

in vec3 Normal;
in vec3 Position;

uniform vec3 cameraPos;
uniform samplerCube skybox;

void main()
{
    vec3 I = normalize(Position - cameraPos);
    vec3 R = reflect(I, normalize(Normal));
    FragColor = vec4(texture(skybox, R).rgb, 1.0);
}
```

Environment Mapping - Reflection

- 큐브VAO와 큐브맵 바인딩 후 렌더링

```
glBindVertexArray(cubeVAO);                                    CPU
glBindTexture(GL_TEXTURE_CUBE_MAP, skyboxTexture);
glDrawArrays(GL_TRIANGLES, 0, 36);
```

- 큐브 렌더링 결과



- Assimp를 이용하면,
  큐브가 아닌 다른 모델도 사용 가능
  - 흡사 크롬 재질과 같은 느낌

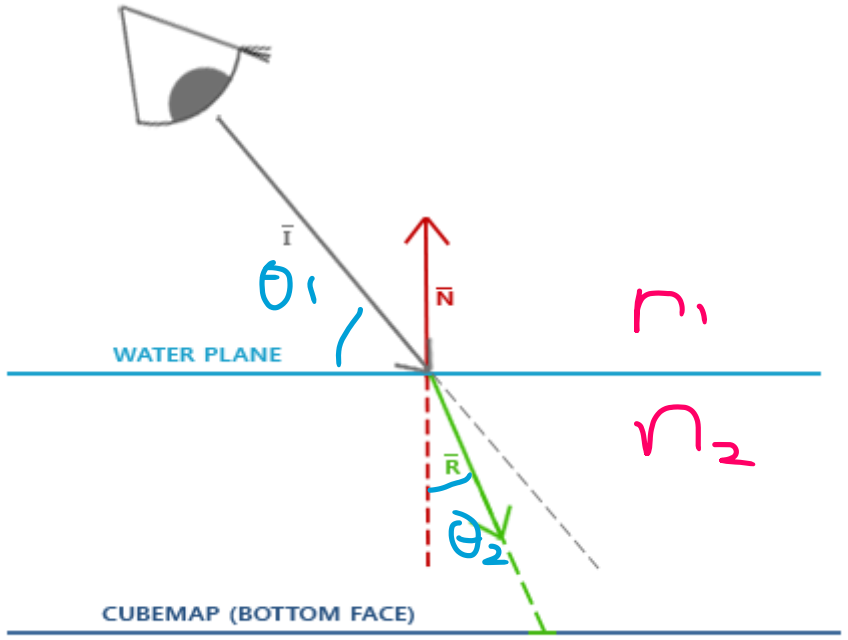
  - 일부 영역만 반사 재질을 부여하기 위해,
    별도의 reflection maps도 사용 가능

Environment Mapping - Refraction

- 굴절 (Refraction)
  - 재질의 변화에 따라 빛의 방향이 달라지는 현상
  - 빛이 직선으로 통과하지 않고 휘어지는 물과 같은 표면에서 흔히 볼 수 있음

- 스넬의 법칙 (Snell's law)
  - 굴절률(refraction index)이 n1과 n2인 재질이 맞닿아 있고, 빛의 경로가 휜 정도가 빛의 입사 평면 상에서 θ1과 θ2라면,

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \ .$$

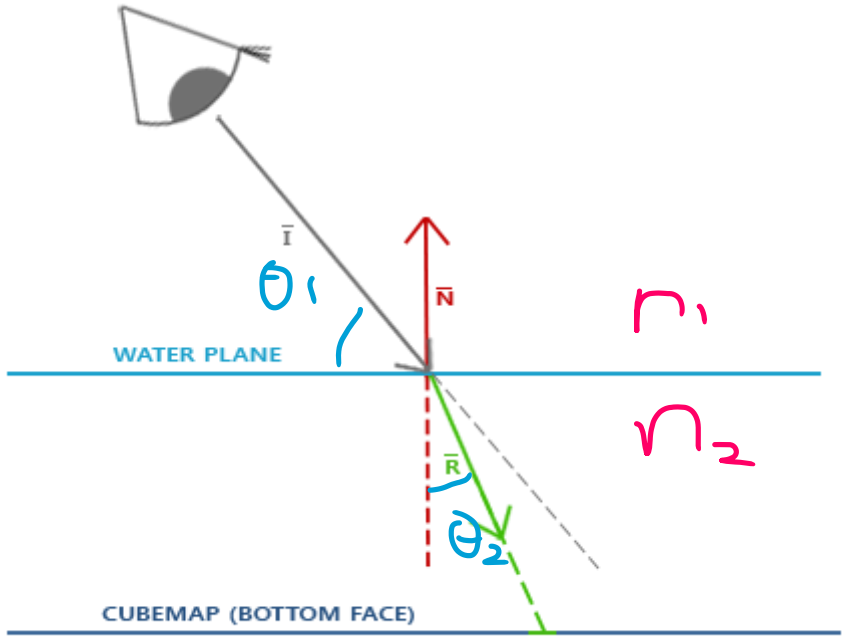


Environment Mapping - Refraction

- 공기에서 유리로 빛이 향하는 경우의 FS 코드
  - refract()함수에 I, N, 굴절률을 입력하여 굴절벡터 R 계산

```
void main()                                                    FS
{
    float ratio = 1.00 / 1.52;
    vec3 I = normalize(Position - cameraPos);
    vec3 R = refract(I, normalize(Normal), ratio);
    FragColor = vec4(texture(skybox, R).rgb, 1.0);
}
```

Material	Refractive index
Air	1.00
Water	1.33
Ice	1.309
Glass	1.52
Diamond	2.42



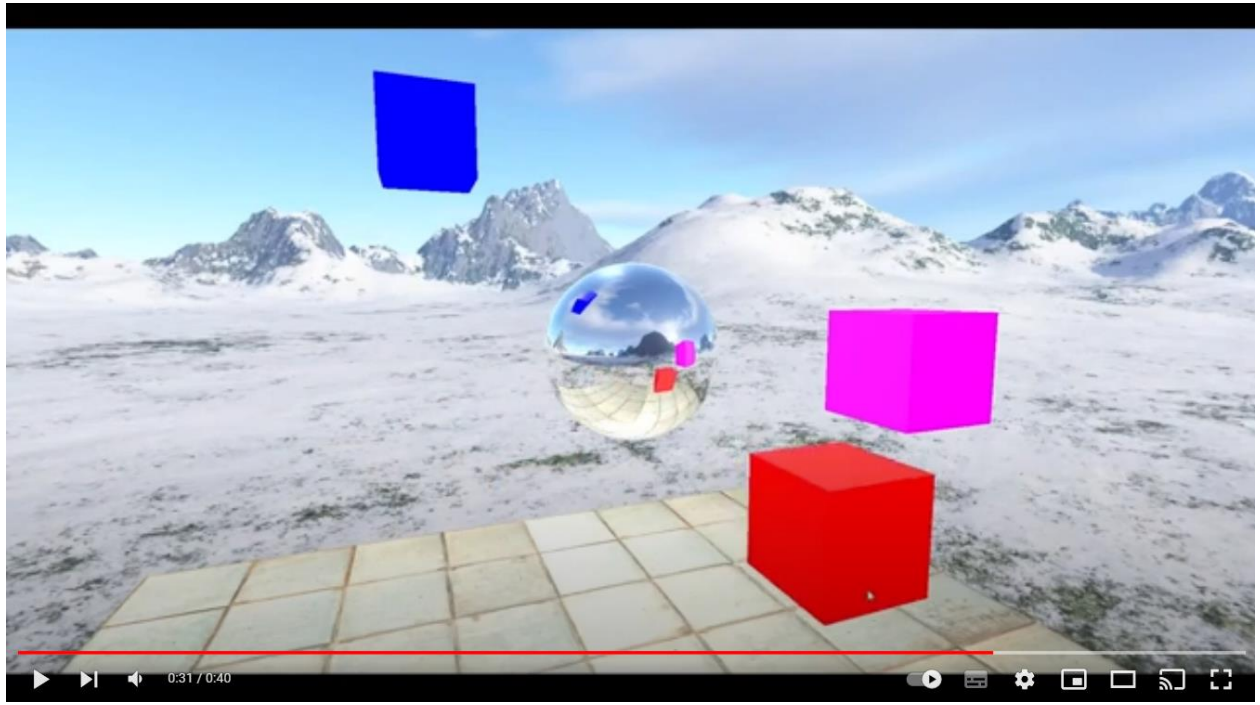
Environment Mapping - Refraction

- 굴절 결과

- 좀 더 정확한 결과를 위해서는,
  빛이 물체에 들어올 때 뿐 아니라 물체로부터 나갈 때에도 굴절이 되어야 함

Dynamic Environment Maps

- 정적인 큐브맵을 읽어들이는 대신, 큐브맵을 동적으로 생성하는 것도 가능
  - Framebuffer를 사용하여 object를 기준으로 6개의 다른 각도에 해당하는 scene을 렌더링하여 cubemap에 저장
  - 이를 현실적인 반사 및 굴절 면을 생성하기 위해 사용 가능 (주변의 움직이는 것들도 제대로 반사/굴절됨)
  - Object당 6번의 렌더링 필요 → 큰 성능 패널티 가능성



Dynamic Environment Cubemap DirectX11 - YouTube

ⓒ nvidia

마무리

마무리

- Advanced OpenGL의 두 번째 시간으로, 아래와 같은 내용을 살펴보았습니다.
  - Framebuffers
  - Cube mapping

- 다음 시간에는 아래 실습을 수행할 예정입니다.
  - 첫번째 기반 코드(LearnOpenGL 4.5.1)에서, 여러가지 post-processing 효과 추가
  - 두번째 기반 코드(LearnOpenGL 4.6.1)에서,
    아래 코드를 참조하여 environment mapping 효과 적용 (반사 및 굴절)
    Code Viewer. Source code:
    src/4.advanced_opengl/6.2.cubemaps_environment_mapping/cubemaps_environment_mapping.cpp
    (learnopengl.com)