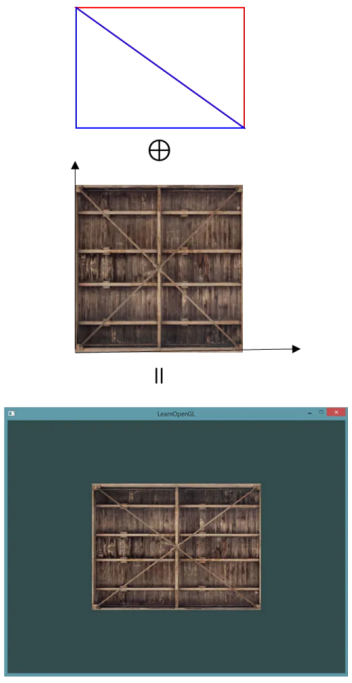
OpenGL-GLSL Textures Basics

Prepare Texture Data

```
Shader ourShader("4.1.texture.vs", "4.1.texture.fs");

// set up vertex data (and buffer(s)) and configure vertex attributes
// ------------------------------------------------------------------
float vertices[] = {
    // positions          // colors           // texture coords
     0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f, // top right
     0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f, // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f, // bottom left
    -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f  // top left
};
unsigned int indices[] = {
    0, 1, 3, // first triangle
    1, 2, 3  // second triangle
};
unsigned int VBO, VAO, EBO;
```



Texture Mapping 과정

1. Parametric coordinate (u, v) 정의
2. Texture coordinate (s, t) 정의
3. World coordinates → Parametric coordinates 변환 (역함수 이용)
4. Parametric coordinates → Texture coordinates 변환 (정규화)

> 💡 OpenGL에서는 **Parametric Coordinate** 과 **Texture Coordinate** 변환 과정을 생략하고,
> 사용자가 직접 **Texture Coordinate** 를 정의해야 함

Texture object 생성 & Bind

```
// load and create a texture
// -------------------------
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture); // all upcoming GL_TEXTURE_2D operations now have effect on this texture objec
// set the texture wrapping parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);    // set texture wrapping to GL_REPEAT (default wrap
```

Texture object 생성

Generate a texture object and bind

생성된 Texture object 를 2D Texture 로 Binding

1. Image 를 저장할 **Texture object 생성**
2. Texture object 에 **Texture 정보를 Binding**

> 💡 OpenGL 상태 머신은 이후의 모든 텍스처 관련 명령을 이 텍스처 객체에 적용함

Texture Image 를 Texture object 에 Load

```
#include <stb_image.h>
```

```
int width, height, nrChannels;
// The FileSystem::getPath(...) is part of the GitHub repository so we can find files on any IDE/platform; replace it
unsigned char *data = stbi_load(FileSystem::getPath("resources/textures/container.jpg").c_str(), &width, &height, &nrC
if (data)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
```

Mipmap 자동 생성

Load image and generate texture

1. **이미지 load**
   - 까다로운 작업임 (이미지 타입이 다양함, Texture format 에 맞게 불러와야 함)

     ⇒ **stbi** 함수를 이용하여 이미지를 쉽게 불러올 수 있음

> 💡 **stb_image.h** : public domain image loader
> (jpg, png, tga, bmp, psd, gif, hdr, pic, pnm)

2. **Texture object 에 이미지 데이터를 load**

Mipmap

- 원본 이미지의 Resolution 을 절반씩 줄여 (Pixel 하나가 될 때까지) 저장한 계층 구조

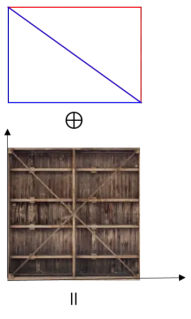
> 💡
> - 만드는 이유: 일반적으로, 기존 이미지의 Resolution 과 Rasterization 된 Target object 의 Resolution 이 같을 수 없음. Mipmap에서 **Target surface resolution 과 가장 비슷한 이미지**를 골라 Texture mapping 을 함
> - Mipmap 의 목적: Resolution 에 맞지 않는 이미지를 샘플 했을 때 원하는 결과가 나오지 못함. 이렇듯, 고해상도 텍스처가 낮은 해상도 표면에 매핑될 때 성능 및 품질을 개선

Texture Wrapping + Texture Filtering

```
// set the texture wrapping parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);    // set texture wrapping to GL_REPEAT (default wrap
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// set texture filtering parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

Texture Wrapping

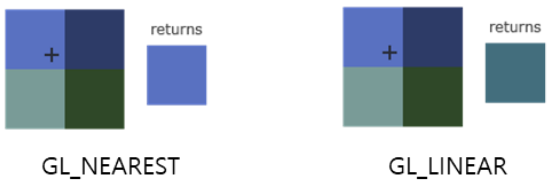
> 💡 보통, Texture space 가 0~1에 normalize 하게 존재함.
> ⇒ 만약, **Texture coordinate 가 0~1 범위를 벗어난 경우**, 해당 구간을 어떻게 나타낼 것인지?



- 옵션
  - `GL_REPEAT`: 반복 복제
  - `GL_MIRRORRED_REPEAT`: flip 반복 복제
  - `GL_CLAMP_TO_EDGE`: 가장자리 픽셀을 늘림
  - `GL_CLAMP_TO_BORDER`: 경계 바깥은 특정 색으로 채움

Texture Filtering

> 💡 stbi 함수를 쓰더라도, Texture 자체는 이미지이기에 유한한 Texel 로 구성되어 있음.
>
> 사용자가 정의한 Texture coordinate (s, t) 는 임의의 continuous 한 소수값으로 정의되는 게 대부분임 → 정확히 하나의 Texel 과 일치하지 않음.
>
> ⇒ 실제 Texel 은 4개인데, 사용자가 정의한 (s, t) 좌표값 중 (0.4, 0.6) 에 해당하는 점은 어떤 Texture 를 사용할 것인지?



옵션

- `GL_NEATEST`: 해당 점에서 가장 가까운 Texture 를 사용
- `GL_LINEAR`: 해당 점으로부터 각 Texel 까지의 거리에 따라 Color Interpolation

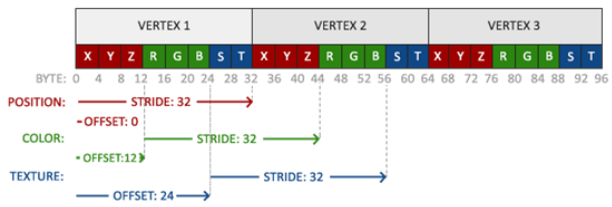
  ⇒ 스무스한 Texture mapping 가능

Configure Texture Attributes

```
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(vertices), vertices, GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

// position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
// color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
// texture coord attribute
```

```
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);
```

Texture (s, t) 정보까지 담아 정의



1. **Vertex shader**
   - Texture Coordinate 를 **Fragment shader 로 넘겨주는 역할**

   > 💡 Rasterization 후엔, 수많은 픽셀이 생겨날 텐데, 사용자가 정의한 Texture coordinate 는 4개뿐임.
   >
   > ⇒ **Vertex 와 Fragment** (Rasterization 된 Pixel) **사이 Interpolation** 하여 각 Fragment 에 전달

2. **Fragment shader**
   - 전달 받은 Texture Coordinate 와 Texture object 를 통해 **실제 Texture Mapping 이 일어나는 곳**
3. **Texture object 와 Shader 연결**
   - 이제 실제 Texture 가 저장되어 있는 곳을 알아야 Mapping 을 할 수 있으니, 그에 대한 정보도 필요
   - OpenGL은 `sampler2D` 타입을 사용해 Texture object 와 Shader 를 연결
     - `sampler2D` **: 2차원 Texture 타입**
     - **OpenGL host 에서 uniform 으로 전달**

> 💡 Single Texture vs. Multiple Textures
>
> - Single Texture: 지금처럼 Texture 를 하나만 쓰는 경우. Shader에 연결할 텍스처 슬롯은 항상 0(`GL_TEXTURE0`)로 고정되어, 별도의 설정 없이 Shader 에서 바로 사용 가능 → 그래서 내부 uniform sampler2D texture1 로 전달 가능한것
> - Multiple Textures: 각각의 Texture 를 다른 텍스처 슬롯에 Binding 해야 함. 이때, Shader 에서 각 텍스처를 식별할 수 있도록 `glUniform1i()` 를 사용해 슬롯 번호를 전달
>
> We need to call
> `glUniform1i(glGetUniformLocation(ourShader.ID, "texture1"), 0);`
> unless you use only a default(single) texture.