Textures

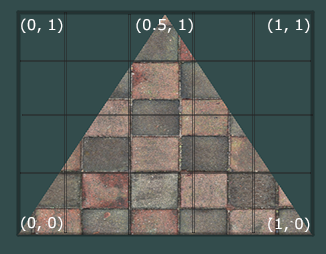
-각 vertex마다 컬러 attribute를 필요로 하기 때문에 많은 vertex를 사용할 경우 상당한 오버헤드를 가져옴

-이를 방지하기 위해 textures를 씀

-Textures는 2D를 주로 사용하나, 1D, 3D texture도 있음

-삼각형에 texture를 매핑하기 위해 vertex에 texture의 어느 부분이 해당하는지 알려줘야함

-texture coordinate 텍스처 좌표 필요

-texture는 0~1 사이의 x, y좌표를 가지고있음

-각 텍스처 좌표

|  |
| --- |
| float texCoords[] = {  0.0f, 0.0f, // 좌측 하단  1.0f, 0.0f, // 우측 하단  0.5f, 1.0f // 꼭대기  }; |

Texture Wrapping

-0~1 밖의 좌표를 지정할 경우

-GL\_REPEAT : 기본, 이미지를 반복함

-GL\_MIRRORED : 반복할 때 마다 이미지를 뒤집음

-GL\_CLAMP\_TO\_EDGE : 큰 좌표가 1로 고정되어 가장자리 패턴이 늘어남

\_GL\_CLAMP\_TO\_BORDER : 사용자가 지정한 색으로 늘어남

-사용법

|  |
| --- |
| glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_MIRRORED\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_MIRRORED\_REPEAT); |

-glTexParameteri(2D텍스처를 알림, S축(S, T, R축이있음 X, Y, Z와 동일), wrapping옵션)

-GL\_CLAMP\_TO\_BORDER사용할 경우 색을 추가 해줘야함

|  |
| --- |
| float borderColor[] = { 1.0f, 1.0f, 0.0f, 1.0f };  glTexParameterfv(GL\_TEXTURE\_2D, GL\_TEXTURE\_BORDER\_COLOR, borderColor); |

Texture Filtering

-텍스처 좌표는 실수 값이 될 수 있음

-OpenGL은 좌표를 매핑할 텍스처 픽셀(텍셀 이라고도 불림)을 찾아야함

-GL\_NEAREST(enarest neighbor filtering) : 가까운 픽셀 선택

-GL\_LINEAR((bi)linear filtering) : 이웃한 픽셀에서 보간된 값을 가져와 근사치를 가져옴

|  |  |
| --- | --- |
| GL\_NEAREST | GL\_LINEAR |
|  |  |

-texture을 확대, 축소 할 때 filtering을 선택할 수 있음

-축소에는 nearest, 확대할 때는 linear을 사용할 수 있음

|  |
| --- |
| glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR); |

Mipmaps

-넓은 공간에 수천 개의 오브젝트가 있고, 각각에 텍스처가 있는 경우

-시점에 멀리 있는 오브젝트가 고해상도 텍스처가 사용된 물체는 오브젝트가 멀리 있기에 약간의 fragment만 생성함, 그러나 고해상도임으로 텍스처 색상 선택하기 어려움을 가짐

-작은 물체에 고해상도 텍스처를 사용할 경우도 마찬가지

-메모리 낭비는 물론, 결함이 생길 수 있음

-이러한 문제를 해결하기 위해 mipmaps를 사용

-기본 텍스처보다 2배작은 텍스처 이미지를 mipmaps이라 함

-특정 거리를 넘으면 거리에 적합한 mipmap 텍스처를 사용

-Mipmaps 필터링 방법

-GL\_NEAREST\_MIPMAP\_NEAREST : nearest 보간법으로 mipmap을 필터링 텍스처 샘플링도 nearest 보간법을 사용

-GL\_LINEAR\_MIPMAP\_NEAREST : nearest 보간법으로 mipmap을 필터링 텍스처 샘플링은 linear 보간법을 사용

-GL\_NEAREST\_MIPMAP\_LINEAR : linear 보간법으로 mipmap을 필터링 텍스처 샘플링은 nearest 보간법을 사용

-GL\_LINEAR\_MIPMAP\_LINEAR : linear 보간법으로 mipmap을 필터링하고 텍스처 샘플링도 linear 보간법을 사용

-필터링 코드

|  |
| --- |
| glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR); |

-주의해야할 점이 MAG(확대)에서는 mipmap을 사용하지 않음, 이 경우 오류코드를 생성

-GL\_INVALID\_ENUM

stb\_image.h

-Sean Barrett의 싱글 헤더 이미지 로드 라이브러리

-헤더파일을 관련된 정의 소스코드만 포함되게 효과적으로 cpp파일로 제작

|  |
| --- |
| #define STB\_IMAGE\_IMPLEMENTATION  #include "stb\_image.h" |

-이미지 로드방법

|  |
| --- |
| int width, height, nrChannels;  unsigned char \*data = stbi\_load("container.jpg", &width, &height, &nrChannels, 0); |

-stbi\_load(이미지경로, 너비, 높이, 컬러채널수, 0)

Generating a texture

-객체생성, ID로 참조

|  |
| --- |
| unsigned int texture;  glGenTextures(1, &texture); |

-바인딩

|  |
| --- |
| glBindTexture(GL\_TEXTURE\_2D, texture); |

-텍스쳐생성

|  |
| --- |
| glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, width, height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, data);  glGenerateMipmap(GL\_TEXTURE\_2D); |

-glTexImage2D(텍스처 타겟, mipmap레벨, 텍스처 포멧, 너비, 높이, 0, 원본 이미지의 포멧 데이터타입, 실제 이미지 데이터)

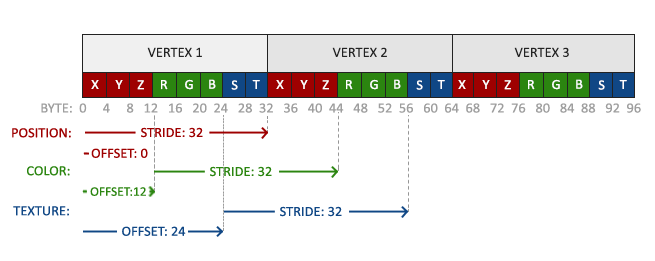
-텍스처와 mipmap을 생성한 뒤에는 이미지 메모리를 반환

|  |
| --- |
| stbi\_image\_free(data); |

Applying texutres

-vertex데이터에 texture좌표 추가

|  |
| --- |
| float vertices[] = {  // 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  }; |

-코드 수정

|  |
| --- |
| glVertexAttribPointer(2, 2, GL\_FLOAT, GL\_FALSE, 8 \* sizeof(float), (void\*)(6 \* sizeof(float)));  glEnableVertexAttribArray(2); |

-vertex shader source 수정

|  |
| --- |
| #version 460 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 = aTexCoord;  } |

-Fragment shader source 수정

|  |
| --- |
| #version 460 core  out vec4 FragColor;    in vec3 ourColor;  in vec2 TexCoord;  uniform sampler2D ourTexture;  void main()  {  FragColor = texture(ourTexture, TexCoord);  } |

-fragment shader에 텍스처 개체를 전달하는 방법

-GLSL에는 텍스처 객체에 대한 데이터타입인 sampler가 있음 뒤에 1D, 2D, 3D가 붙음

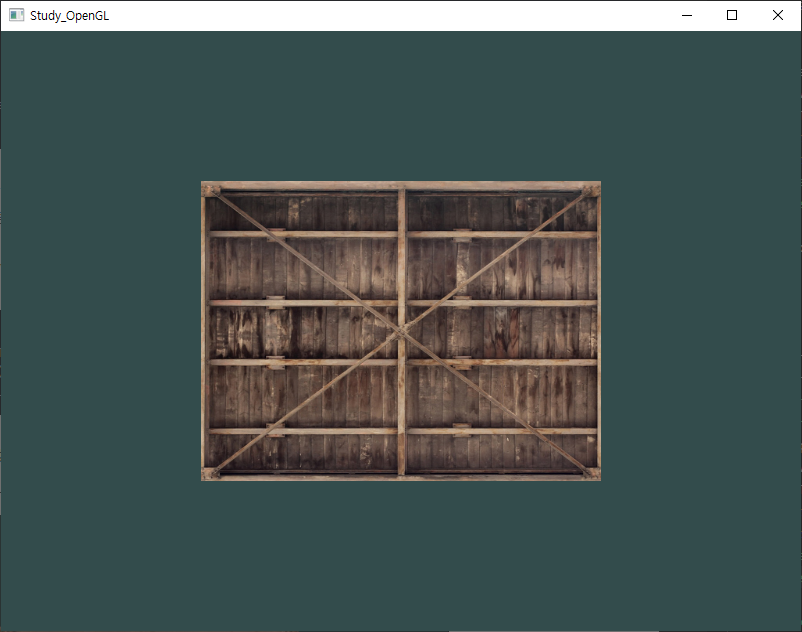
-텍스처 색을 샘플링하기 위해서 texture함수를 사용함

-texture(샘플러, 텍스처좌표)

-텍스처 바인딩

|  |
| --- |
| glBindTexture(GL\_TEXTURE\_2D, texture);  glBindVertexArray(VAO);  glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0); |

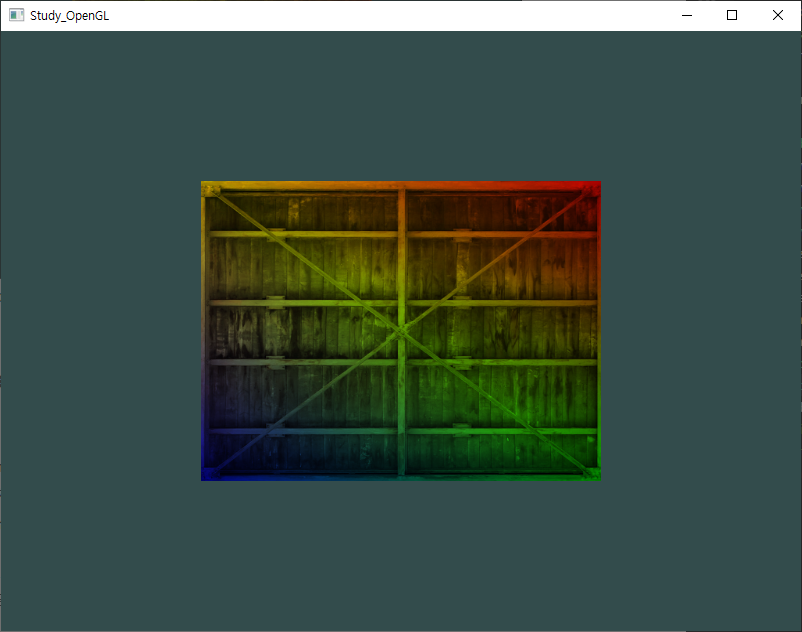
-결과



-텍스처와 vertex Color을 혼합하는 법

-fragment shader에서 텍스처컬러 \* vertex컬러 하면 됨

|  |
| --- |
| FragColor = texture(ourTexture, TexCoord) \* vec4(ourColor, 1.0); |



Texture Units

-unifrom으로 할당하지 않은 sampler가 왜 uniform인가?

-glUniform1i 함수를 사용해 텍스처 sampler에 위치값을 할당해 fragment shader에서 동시에 여러 텍스처들을 설정할 수 있음

-texture unit의 주 목적 : shader에서 하나 이상의 텍스처를 사용할 수 있도록 하는 것

-sampler에 텍스처 유닛을 할당, 유닛을 활성화 하면 여러 텍스처를 동시에 바인딩이 됨

|  |
| --- |
| glActiveTexture(GL\_TEXTURE0); // 텍스처를 바인딩하기 전에 먼저 텍스처 유닛을 활성화  glBindTexture(GL\_TEXTURE\_2D, texture); |

-GL\_TEXTURE0~15 까지 쓸 수 있음 / GL\_TEXTURE0+7 처럼 접근이 가능함

-fragment shader 수정

|  |
| --- |
| #version 460 core  ...  uniform sampler2D texture1;  uniform sampler2D texture2;  void main()  {  FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), 0.2);  } |

-뒤에 0.2는 첫 번째 텍스처를 0.8, 두 번째 텍스처를 0.2만큼 리턴해 혼합

-두 번째 텍스처 불러오기

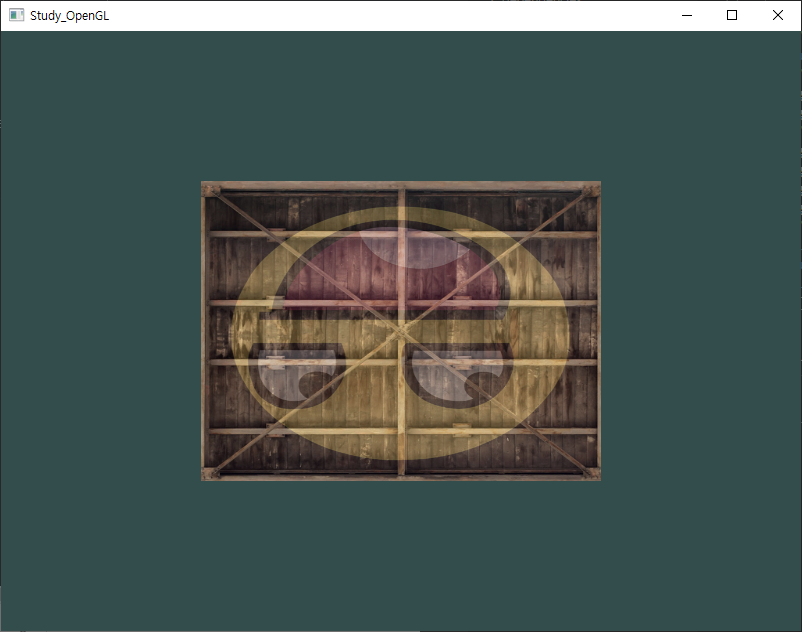
|  |
| --- |
| unsigned char \*data = stbi\_load("awesomeface.png", &width, &height, &nrChannels, 0);  if (data)  {  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, width, height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, data);  glGenerateMipmap(GL\_TEXTURE\_2D);  } |

-랜더링 수정

|  |
| --- |
| glActiveTexture(GL\_TEXTURE0);  glBindTexture(GL\_TEXTURE\_2D, texture1);  glActiveTexture(GL\_TEXTURE1);  glBindTexture(GL\_TEXTURE\_2D, texture2);  glBindVertexArray(VAO);  glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0); |

-glUniform1i를 사용해 각 sampler을 설정, shader sampler가 속하는 텍스처 유닛이 어떤 것인지 알려줌(루프 들어가기 전에 설정)

|  |
| --- |
| ourShader.use(); // uniform을 설정하기 전에 shader를 활성화해야 한다는 것을 잊지마세요!  glUniform1i(glGetUniformLocation(ourShader.ID, "texture1"), 0); // 직접 설정  ourShader.setInt("texture2", 1); // 혹은 shader 클래스를 활용    while(...)  {  [...]  } |

-2번째 텍스처가 뒤집어져서 나옴

-OpenGL은 y 0.0을 이미지의 아래쪽으로 인식, 대부분의 이미지는 위쪽으로 인식

-이미지를 로드하기전에 뒤집는법

|  |
| --- |
| stbi\_set\_flip\_vertically\_on\_load(true); |

