Shaders

-GPU에서 동작하는 작은 프로그램

-Graphic pipeline의 특정 부분을 맡아서 실행

-기본적인 의미로 입력값을 출력값으로 변환시키는 프로그램

GLSL

-C언어와 비슷하게 생김

-Shader의 구조

|  |
| --- |
| #version 460  in type in\_variable\_name;  in type in\_variable\_name;  out type out\_variable\_name;    uniform type uniform\_name;    void main()  {  // 입력 값을 처리하고 그래픽 작업을 합니다.  ...  // 처리된 것을 출력 변수로 출력합니다.  out\_variable\_name = weird\_stuff\_we\_processed;  } |

-입력 변수는 vertex attribute 라고 부름

-선언할 수 있는 최대 개수가 정해져 있음, 최소 16개까지는 보장함

-최대 개수는 GL\_MAX\_VERTEX\_ATRRIBS로 알 수 있음

|  |
| --- |
| int nrAttributes;  glGetIntegerv(GL\_MAX\_VERTEX\_ATTRIBS, &nrAttributes);  std::cout << "Maximum nr of vertex attributes supported: " << nrAttributes << std::endl; |

Types

-GLSL은 int, float, double, uint, bool의 데이터 타입을 가지고있음

-vector, matrices의 컨테이너 타입도 가지고있음

Vectors

-GLSL의 백터의 형식

-vecn : n개의 float ex) vec3 : 3개의 float을 가진 vector

-bvecn : n개의 bool

-ivecn : n개의 integer

-uvecn : n개의 unsigned integer

-dvecn : n개의 double

-백터의 요소 접근 : vec.x 와 같이 접근 ( .x .y .z .w)

-swizzling이라는 유연한 요소 선택을 할 수 있음

-swizzling

|  |
| --- |
| vec2 someVec;  vec4 differentVec = someVec.xyxx;  vec3 anotherVec = differentVec.zyw;  vec4 otherVec = someVec.xxxx + anotherVec.yxzy; |

-새로운 vector을 만들 때, 원래 있던 vector의 요소들로 만들 수 있음

|  |
| --- |
| vec2 vect = vec2(0.5, 0.7);  vec4 result = vec4(vect, 0.0, 0.0);  vec4 otherResult = vec4(result.xyz, 1.0); |

Ins and outs

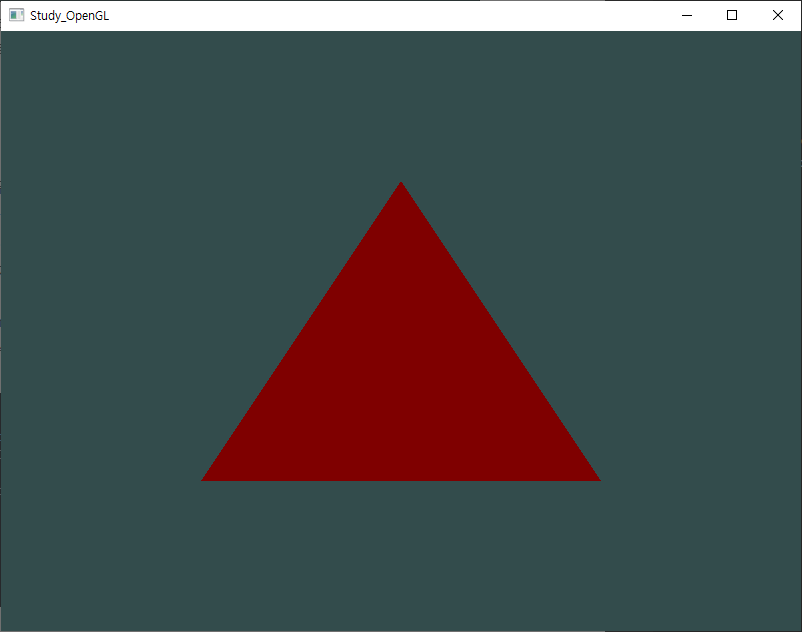
-Shader마다 입력, 출력이 필요함

-데이터를 한 shader에서 다른 shader로 보내고 싶다면, 넘기는 shader에서 출력을 선언해야하고, 받는 shader에선 입력을 선언해야함

-양쪽의 타입과 이름이 같으면 OpenGL은 두 변수를 연결시킴

-Vertex Shader와 Fragment Shader을 변경

|  |
| --- |
| #version 460 core  layout (location = 0) in vec3 aPos; // 위치 변수는 attribute position 0을 가집니다.    out vec4 vertexColor; // fragment shader를 위한 컬러 출력을 지정  void main()  {  gl\_Position = vec4(aPos, 1.0); // vec4의 생성자에 vec3를 직접적으로 주었습니다.  vertexColor = vec4(0.5, 0.0, 0.0, 1.0); // 출력 변수에 짙은 빨간색을 설정했습니다.  } |
| #version 460 core  out vec4 FragColor;    in vec4 vertexColor; // Vertex Shader로 부터 받은 입력 변수(똑같은 이름, 똑같은 타입)  void main()  {  FragColor = vertexColor;  } |



Uniforms

-CPU위의 응용프로그램에서 GPU위의 Shader로 데이터를 전달하는 방법

-uniform global : shader program객체에서 고유한 변수, shader program에서 모든단계의 shader에서 접근이 가능함, uniform변수에 어떤 값을 설정하던, 리셋을 하거나 업데이트하기 전까지 숫자를 유지(전역변수)

-uniform 선언방법 (Fragment shader source)

|  |
| --- |
| #version 460 core  out vec4 FragColor;    uniform vec4 ourColor; // OpenGL 코드에서 이 변수를 설정  void main()  {  FragColor = ourColor;  } |

-uniform을 선언했는데, GLSL코드에서 사용하지 않는다면 컴파일러는 자동으로 uniform변수를 삭제해 버그를 일으킬 수 있음

-uniform의 index/location을 알아낸 후, 값을 수정함

-시간에 따라 색이 변하는 코드를 삽입

|  |
| --- |
| float timeValue = glfwGetTime();  float greenValue = (sin(timeValue) / 2.0f) + 0.5f;  int vertexColorLocation = glGetUniformLocation(shaderProgram, "ourColor");  glUseProgram(shaderProgram);  glUniform4f(vertexColorLocation, 0.0f, greenValue, 0.0f, 1.0f); |

-값을 수정할 때, Program을 사용해야 함(glUseProgram 호출)

-glUniformㅁㅁ : ㅁㅁ에 설정하고 싶은 uniform 타입을 나타냄

-f : float

-i : integer

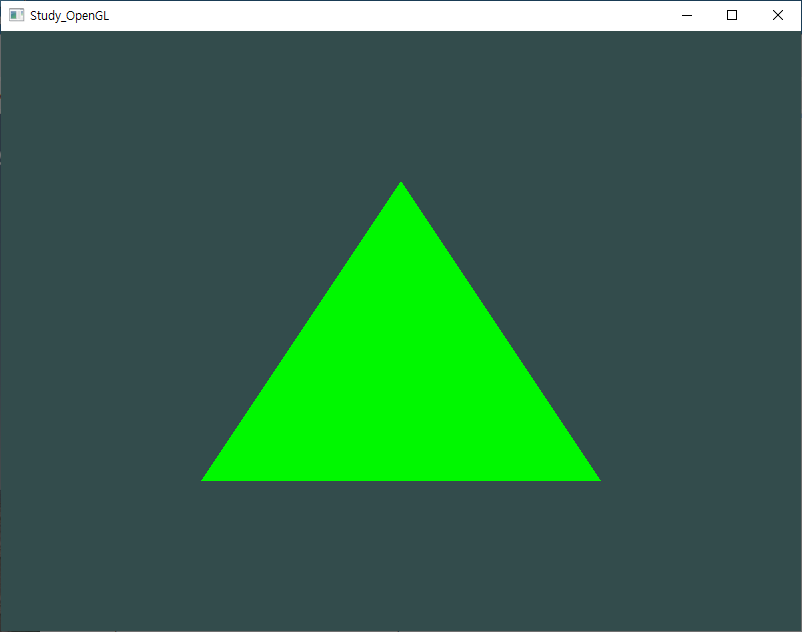
-ui : unsigned integer

-nf : n개의 float

-fv : float타입의 vector/array

-루프가 돌때마다 삼각형 색을 변경시킴

|  |
| --- |
| while(!glfwWindowShouldClose(window))  {  // 입력  processInput(window);  // 렌더링  // colorbuffer 비우기  glClearColor(0.2f, 0.3f, 0.3f, 1.0f);  glClear(GL\_COLOR\_BUFFER\_BIT);  // shader를 활성화  glUseProgram(shaderProgram);    // uniform 컬러 수정  float timeValue = glfwGetTime();  float greenValue = sin(timeValue) / 2.0f + 0.5f;  int vertexColorLocation = glGetUniformLocation(shaderProgram, "ourColor");  glUniform4f(vertexColorLocation, 0.0f, greenValue, 0.0f, 1.0f);  // 삼각형 렌더링  glBindVertexArray(VAO);  glDrawArrays(GL\_TRIANGLES, 0, 3);    // 버퍼 교체, IO 이벤트 처리  glfwSwapBuffers(window);  glfwPollEvents();  } |



More attributes!

-vertices 추가

|  |
| --- |
| float vertices[] = {  // 위치 // 컬러  0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, // 우측 하단  -0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, // 좌측 하단  0.0f, 0.5f, 0.0f, 0.0f, 0.0f, 1.0f // 위  }; |

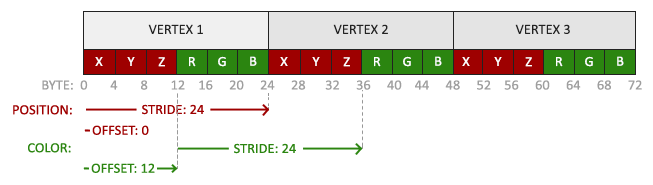
-vertex shader source 변경

|  |
| --- |
| #version 460 core  layout (location = 0) in vec3 aPos; // 위치 변수는 attribute position 0을 가짐  layout (location = 1) in vec3 aColor; // 컬러 변수는 attribute position 1을 가짐    out vec3 ourColor; // 컬러를 fragment shader로 출력  void main()  {  gl\_Position = vec4(aPos, 1.0);  ourColor = aColor; // vertex data로부터 가져오 컬러 입력을 ourColor에 설정  } |

-fragment shader source변경

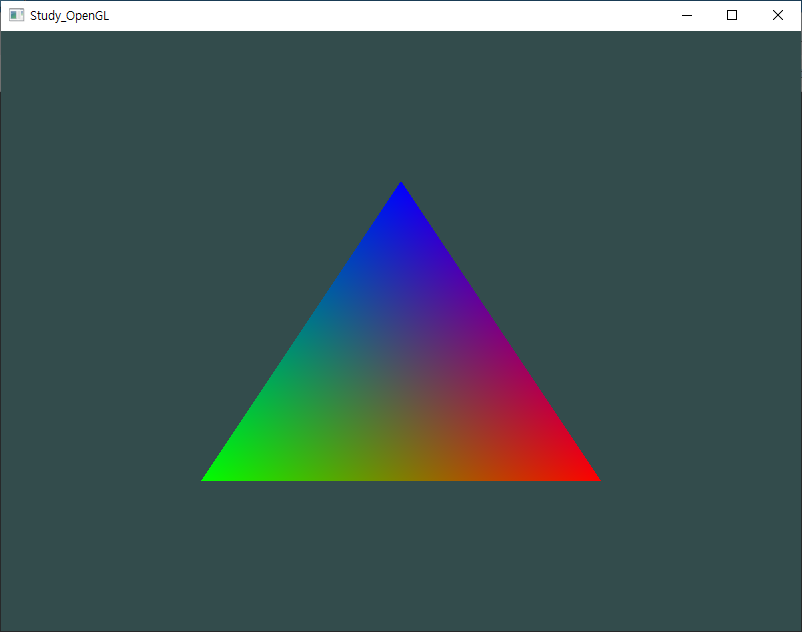
|  |
| --- |
| #version 460 core  out vec4 FragColor;  in vec3 ourColor;    void main()  {  FragColor = vec4(ourColor, 1.0);  } |

-VBO를 수정해야함 –변경한 데이터



-vertex 형식 수정

|  |
| --- |
| // 위치 attribute  glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(float), (void\*)0);  glEnableVertexAttribArray(0); //layout 0  // 컬러 attribute  glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(float), (void\*)(3\* sizeof(float)));  glEnableVertexAttribArray(1); //layout 1 |



-정점의 색은 3개의 색(파란색, 녹색, 빨간색)인데 이렇게 나오는 이유

fragment interpolation

-각 위치들을 기반으로 fragment shader의 모든 입력 변수를 보간(interpolate)함

-사이의 값을 계산해서 적용시킴

Our own shader class

-shader를 프로그램 하나에 작성하고, 컴파일하기엔 불편한 점이 있음

-따로 저장하고 프로그램에서 읽어오는 방식으로 바꿈

-shader class

|  |
| --- |
| #ifndef SHADER\_H  #define SHADER\_H  #include <glad/glad.h> // 필요한 모든 OpenGL의 헤더파일을 가져오기 위해 glad를 포함    #include <string>  #include <fstream>  #include <sstream>  #include <iostream>    class Shader  {  public:  // program ID  unsigned int ID;    // 생성자는 shader를 읽고 생성  Shader(const GLchar\* vertexPath, const GLchar\* fragmentPath);  // shader를 활성화하고 사용  void use();  // Uniform 유틸리티 함수들  void setBool(const std::string &name, bool value) const;  void setInt(const std::string &name, int value) const;  void setFloat(const std::string &name, float value) const;  };    #endif |

-Shader class는 Shader Program의 ID를 가지고있음

-생성자는 vertex shader, fragment shader의 소스코드 파일 경로가 필요

-use()는 shader program을 활성화함

-setㅁㅁ함수는 uniform location을 확인 값을 설정함

Reading from file

-shader class에 파일내용을 string에 읽기 위해 c++filestream을 사용하ㅏㅁ

|  |
| --- |
| Shader(const char\* vertexPath, const char\* fragmentPath)  {  // 1. 파일 경로를 통해 vertex/fragment shader 소스 코드를 검색합니다.  std::string vertexCode;  std::string fragmentCode;  std::ifstream vShaderFile;  std::ifstream fShaderFile;  // ifstream 객체들이 예외를 던질 수 있도록 합니다.  vShaderFile.exceptions (std::ifstream::failbit | std::ifstream::badbit);  fShaderFile.exceptions (std::ifstream::failbit | std::ifstream::badbit);  try  {  // 파일 열기  vShaderFile.open(vertexPath);  fShaderFile.open(fragmentPath);  std::stringstream vShaderStream, fShaderStream;  // stream에 파일의 버퍼 내용을 읽기  vShaderStream << vShaderFile.rdbuf();  fShaderStream << fShaderFile.rdbuf();  // 파일 핸들러 닫기  vShaderFile.close();  fShaderFile.close();  // stream을 string으로 변환  vertexCode = vShaderStream.str();  fragmentCode = fShaderStream.str();  }  catch(std::ifstream::failure e)  {  std::cout << "ERROR::SHADER::FILE\_NOT\_SUCCESFULLY\_READ" << std::endl;  }  const char\* vShaderCode = vertexCode.c\_str();  const char\* fShaderCode = fragmentCode.c\_str();  [...] |

-후 shader을 컴파일, 링크함

|  |
| --- |
| // 2. shader 컴파일  unsigned int vertex, fragment;  int success;  char infoLog[512];    // vertex Shader  vertex = glCreateShader(GL\_VERTEX\_SHADER);  glShaderSource(vertex, 1, &vShaderCode, NULL);  glCompileShader(vertex);  // 오류가 발생한다면 컴파일 오류를 출력  glGetShaderiv(vertex, GL\_COMPILE\_STATUS, &success);  if(!success)  {  glGetShaderInfoLog(vertex, 512, NULL, infoLog);  std::cout << "ERROR::SHADER::VERTEX::COMPILATION\_FAILED\n" << infoLog << std::endl;  };  // Fragment shader도 마찬가지  [...]  // shader Program  ID = glCreateProgram();  glAttachShader(ID, vertex);  glAttachShader(ID, fragment);  glLinkProgram(ID);  // 오류가 발생한다면 링킹 오류를 출력  glGetProgramiv(ID, GL\_LINK\_STATUS, &success);  if(!success)  {  glGetProgramInfoLog(ID, 512, NULL, infoLog);  std::cout << "ERROR::SHADER::PROGRAM::LINKING\_FAILED\n" << infoLog << std::endl;  }    // program 내부에서 shader들이 링크 완료되었다면 이제 필요 없으므로 shader들을 삭제  glDeleteShader(vertex);  glDeleteShader(fragment); |

-use함수

|  |
| --- |
| void use()  {  glUseProgram(ID);  } |

-uniform세팅 함수

|  |
| --- |
| void setBool(const std::string &name, bool value) const  {  glUniform1i(glGetUniformLocation(ID, name.c\_str()), (int)value);  }  void setInt(const std::string &name, int value) const  {  glUniform1i(glGetUniformLocation(ID, name.c\_str()), value);  }  void setFloat(const std::string &name, float value) const  {  glUniform1f(glGetUniformLocation(ID, name.c\_str()), value);  } |

-메인 프로그램에서 사용할 코드

|  |
| --- |
| Shader ourShader("path/to/shaders/shader.vs", "path/to/shaders/shader.fs");  ...  while(...)  {  ourShader.use();  ourShader.setFloat("someUniform", 1.0f);  DrawStuff();  } |

-shader.vs : Vertex Shader Source를 저장한 파일

-shader.fs : Fragment Shader Source를 저장한 파일

////////////////////////////////////////////////////////////////

-shader.h는 폴더에 src/shader.h

-vertex/fragment shader source는 src/shaders/에 있음

-shader.h를 따로 사용한 source code는 src/6\_Shaders\_Use\_Class에 구현