Model / Mesh

-Assimp로 불러온 보델은 Assimp 데이터 구조의 형식으로 저장되어짐

-이를 OpenGL이 이해할 수 있는 포멧으로 변환, 렌더링 해야함

-Mesh는 그려질 수 있는 하나의 독립체임

-Mesh가 필요하는 최소한의 요구사항

-vertex모음 / 위치벡터, 법선벡터, 텍스처 좌표 벡터

|  |
| --- |
| struct Vertex {  glm::vec3 Position;  glm::vec3 Normal;  glm::vec2 TexCoords;  }; |

-Texture struct에 텍스처 데이터를 저장하도록 코딩

|  |
| --- |
| struct Texture {  unsigned int id;  string type; // diffuse텍스처, specular텍스처  }; |

-Mesh class 생성

|  |
| --- |
| class Mesh {  public:  /\* Mesh 데이터 \*/  vector<Vertex> vertices;  vector<unsigned int> indices;  vector<Texture> textures;  /\* 함수 \*/  Mesh(vector<Vertex> vertices, vector<unsigned int> indices,  vector<Texture> textures);  void Draw(Shader shader);  private:  /\* 렌더 데이터 \*/  unsigned int VAO, VBO, EBO;  /\* 함수 \*/  void setupMesh();  }; |

-생성자에 데이터를 주고, setupMesh함수로 버퍼들을 초기화, Draw함수로 그림

-생성자 내부에서 setupMesh함수를 호출

|  |
| --- |
| Mesh(vector<Vertex> vertices, vector<unsigned int> indices, vector<Texture> textures)  {  this->vertices = vertices;  this->indices = indices;  this->textures = textures;  setupMesh();  } |

Initialization

-setupMesh함수로 렌더링에 사용할 수 있는 mesh데이터의 목록을 가질 수 있음

-적당한 버퍼를 설정, Vertex Attribute pointer을 통해 vertex shader layout을 지정해줘야 함

|  |
| --- |
| void setupMesh()  {  glGenVertexArrays(1, &VAO);  glGenBuffers(1, &VBO);  glGenBuffers(1, &EBO);    glBindVertexArray(VAO);  glBindBuffer(GL\_ARRAY\_BUFFER, VBO);  glBufferData(GL\_ARRAY\_BUFFER, vertices.size() \* sizeof(Vertex),  &vertices[0], GL\_STATIC\_DRAW);  glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, EBO);  glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, indices.size() \* sizeof(unsigned int),  &indices[0], GL\_STATIC\_DRAW);  // vertex positions  glEnableVertexAttribArray(0);  glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), (void\*)0);  // vertex normals  glEnableVertexAttribArray(1);  glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE,  sizeof(Vertex), (void\*)offsetof(Vertex, Normal));  // vertex texture coords  glEnableVertexAttribArray(2);  glVertexAttribPointer(2, 2, GL\_FLOAT, GL\_FALSE,  sizeof(Vertex), (void\*)offsetof(Vertex, TexCoords));  glBindVertexArray(0);  } |

-코드 중간에 float가 아니라 Veretx strcut가 들어감

-Struct의 속성들은 배열처럼 메모리의 위치가 순차적으로 저장되어짐

ex)

|  |
| --- |
| Vertex vertex;  vertex.Position = glm::vec3(0.2f, 0.4f, 0.6f);  vertex.Normal = glm::vec3(0.0f, 1.0f, 0.0f);  vertex.TexCoords = glm::vec2(1.0f, 0.0f);  // = [0.2f, 0.4f, 0.6f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f]; |

-따라서 Vertex Struct들을 buffer 데이터로 전달할 수 있음

|  |
| --- |
| glBufferData(GL\_ARRAY\_BUFFER, vertices.size() \* sizeof(Vertex),  vertices[0], GL\_STATIC\_DRAW); |

|  |
| --- |
| glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE,  sizeof(Vertex), (void\*)offsetof(Vertex, Normal)); |

Rendering

-Mesh를 렌더링하기 전에 텍스처를 바인딩 해야함

-각 diffuse, specular을 uniform으로 선언

|  |
| --- |
| uniform sampler2D texture\_diffuse1;  uniform sampler2D texture\_diffuse2;  uniform sampler2D texture\_diffuse3;  uniform sampler2D texture\_specular1;  uniform sampler2D texture\_specular2; |

-Draw 코드

|  |
| --- |
| void Draw(Shader shader)  {  unsigned int diffuseNr = 1;  unsigned int specularNr = 1;  for(unsigned int i = 0; i < textures.size(); i++)  {  glActiveTexture(GL\_TEXTURE0 + i); // 바인딩하기 전에 텍스처 유닛 활성화  // 텍스처 넘버(diffuse\_textureN 에서 N) 구하기  string number;  string name = textures[i].type;  if(name == "texture\_diffuse")  number = std::to\_string(diffuseNr++);  else if(name == "texture\_specular")  number = std::to\_string(specularNr++);  shader.setFloat(("material." + name + number).c\_str(), i);  glBindTexture(GL\_TEXTURE\_2D, textures[i].id);  }  glActiveTexture(GL\_TEXTURE0);  // mesh 그리기  glBindVertexArray(VAO);  glDrawElements(GL\_TRIANGLES, indices.size(), GL\_UNSIGNED\_INT, 0);  glBindVertexArray(0);  } |