Model

-모델을 실제 로딩, 변환 코드를 생성할 것

-Model 클래스의 구조

|  |
| --- |
| class Model  {  public:  /\* 함수 \*/  Model(char \*path)  {  loadModel(path);  }  void Draw(Shader shader);  private:  /\* Model 데이터 \*/  vector<Mesh> meshes;  string directory;  /\* 함수 \*/  void loadModel(string path);  void processNode(aiNode \*node, const aiScene \*scene);  Mesh processMesh(aiMesh \*mesh, const aiScene \*scene);  vector<Texture> loadMaterialTextures(aiMaterial \*mat, aiTextureType type,  string typeName);  }; |

-Draw 함수

|  |
| --- |
| void Draw(Shader shader)  {  for(unsigned int i = 0; i < meshes.size(); i++)  meshes[i].Draw(shader);  } |

Importing a 3D model into OpenGL

-Assimp의 해더파일

|  |
| --- |
| #include <assimp/Importer.hpp>  #include <assimp/scene.h>  #include <assimp/postprocess.h> |

-Importer객체 선언

|  |
| --- |
| Assimp::Importer importer;  const aiScene \*scene = importer.ReadFile(path, aiProcess\_Triangulate |  aiProcess\_FlipUVs); |

-ReadFile함수를 호출(파일경로, 전처리옵션들)

-전처리 옵션들에는 여러가지 있음

-aiProcess\_Triangulate : 삼각형으로 이루어 지지 않았다면, 모든 도형을 삼각형으로 변환

-aiProcess\_FlipUVs : 텍스처 좌표를 y축으로 뒤집음

-aiProcess\_Gennormals : 각 vertex들에 대한 법선을 생성

-aiProcess\_SplitLargeMeshed : 큰 mesh들을 작은 mesh로 나눔

-aiProcess\_OptimizeMeshes : 여러 mesh들을 큰 mesh로 합침

-<http://assimp.sourceforge.net/lib_html/postprocess_8h.html>

-완성된 loadModel함수

|  |
| --- |
| void loadModel(string path)  {  Assimp::Importer import;  const aiScene \*scene = import.ReadFile(path, aiProcess\_Triangulate  | aiProcess\_FlipUVs);    if(!scene || scene->mFlags & AI\_SCENE\_FLAGS\_INCOMPLETE  || !scene->mRootNode)  {  cout << "ERROR::ASSIMP::" << import.GetErrorString() << endl;  return;  }  directory = path.substr(0, path.find\_last\_of('/'));  processNode(scene->mRootNode, scene);  } |

-각 노드는 Mesh Index들의 모음을 가지고 있음

-각 index는 scene 객체 내부의 특정한 mesh를 가리킴

-mesh index들을 얻고, mesh를 얻고, mesh를 처리하고 나서 각 노드의 자식노드에 작업을 반복하는 식으로 처리

-processNode함수

|  |
| --- |
| void processNode(aiNode \*node, const aiScene \*scene)  {  // 노드의 모든 mesh들을 처리(만약 있다면)  for(unsigned int i = 0; i < node->mNumMeshes; i++)  {  aiMesh \*mesh = scene->mMeshes[node->mMeshes[i]];  meshes.push\_back(processMesh(mesh, scene));  }  // 그런 다음 각 자식들에게도 동일하게 적용  for(unsigned int i = 0; i < node->mNumChildren; i++)  {  processNode(node->mChildren[i], scene);  }  } |

Assimp to Mesh

-mesh들의 관련된 속성들에 접근하여 우리만의 객체에 저장하는 것

-processMesh 함수의 구조

|  |
| --- |
| Mesh processMesh(aiMesh \*mesh, const aiScene \*scene)  {  vector<Vertex> vertices;  vector<unsigned int> indices;  vector<Texture> textures;  for(unsigned int i = 0; i < mesh->mNumVertices; i++)  {  Vertex vertex;  // vertex 위치, 법선, 텍스처 좌표를 처리  ...  vertices.push\_back(vertex);  }  // indices 처리  ...  // material 처리  if(mesh->mMaterialIndex >= 0)  {  ...  }  return Mesh(vertices, indices, textures);  } |

-Mesh를 처리하는 3단계

-1. 모든 vertex 얻기

-2. mesh의 indices 얻기

-3. 연관된 Material 데이터 얻기

-1. vertex 얻기

|  |
| --- |
| glm::vec3 vector;  vector.x = mesh->mVertices[i].x;  vector.y = mesh->mVertices[i].y;  vector.z = mesh->mVertices[i].z;  vertex.Position = vector; |

-법선을 위한 작업도 동일함

|  |
| --- |
| vector.x = mesh->mNormals[i].x;  vector.y = mesh->mNormals[i].y;  vector.z = mesh->mNormals[i].z;  vertex.Normal = vector; |

-텍스처 좌표가 있는가 확인

|  |
| --- |
| if(mesh->mTextureCoords[0]) // mesh가 텍스처 좌표를 가지고 있는가?  {  glm::vec2 vec;  vec.x = mesh->mTextureCoords[0][i].x;  vec.y = mesh->mTextureCoords[0][i].y;  vertex.TexCoords = vec;  }  else  vertex.TexCoords = glm::vec2(0.0f, 0.0f); |

-이제 이것들을 vertices vector의 끝에 삽입할 수 있음

Indices

-Assimp의 인터페이스는 각 mesh들이 face 배열을 가지도록 정의함

-face들은 하나의 primitive를 나타냄

-현재 사용하고 있는 primitive는 삼각형

-face는 어떤 순서로 vertex를 그려야 하는지 정의하는 indices를 가지고 있어서 indices를 가져옴

|  |
| --- |
| for(unsigned int i = 0; i < mesh->mNumFaces; i++)  {  aiFace face = mesh->mFaces[i];  for(unsigned int j = 0; j < face.mNumIndices; j++)  indices.push\_back(face.mIndices[j]);  } |

-루프가 끝나면 glDrawElements 함수를 통해 그리기를 위한 vertex, index 데이터가 설정됨

Material

-mesh는 material의 index값만 가지고 있음

-실제 material을 가지기 위해서는 scene의 mMaterial 배열을 인덱싱 해야함

-mesh의 material index는 mMaterialIndex 속성에 설정되어있음

|  |
| --- |
| if(mesh->mMaterialIndex >= 0)  {  aiMaterial \*material = scene->mMaterials[mesh->mMaterialIndex];  vector<Texture> diffuseMaps = loadMaterialTextures(material,  aiTextureType\_DIFFUSE,  "texture\_diffuse");  textures.insert(textures.end(), diffuseMaps.begin(), diffuseMaps.end());  vector<Texture> specularMaps = loadMaterialTextures(material,  aiTextureType\_SPECULAR,  "texture\_specular");  textures.insert(textures.end(), specularMaps.begin(), specularMaps.end());  } |

-loadMaterialTextures 함수

|  |
| --- |
| vector<Texture> loadMaterialTextures(aiMaterial \*mat, aiTextureType type, string typeName)  {  vector<Texture> textures;  for(unsigned int i = 0; i < mat->GetTextureCount(type); i++)  {  aiString str;  mat->GetTexture(type, i, &str);  Texture texture;  texture.id = TextureFromFile(str.C\_Str(), directory);  texture.type = typeName;  texture.path = str;  textures.push\_back(texture);  }  return textures;  } |

An optimization 최적화

-최적화 작업이 남아있음

-mesh들에 여러 텍스처들을 사용할 것임, 이때, 여러 텍스처가 재사용될 것임

-따라서 이미 load가 끝난 텍스처일 경우, 그것을 사용하게 만들면 좋음

-model코드에 변형을 줄 것임, Texture struct

|  |
| --- |
| struct Texture {  unsigned int id;  string type;  string path; // 다른 텍스처와 비교하기 위해 텍스처의 경로를 저장  }; |

-모델 class 위에 private 변수로 선언된 vector에 불러온 텍스처를 저장

|  |
| --- |
| vector<Texture> textures\_loaded; |

-loadMaterialTextures 함수를 변경

|  |
| --- |
| vector<Texture> loadMaterialTextures(aiMaterial \*mat, aiTextureType type, string typeName)  {  vector<Texture> textures;  for(unsigned int i = 0; i < mat->GetTextureCount(type); i++)  {  aiString str;  mat->GetTexture(type, i, &str);  bool skip = false;  for(unsigned int j = 0; j < textures\_loaded.size(); j++)  {  if(std::strcmp(textures\_loaded[j].path.data(), str.C\_Str()) == 0)  {  textures.push\_back(textures\_loaded[j]);  skip = true;  break;  }  }  if(!skip)  { // 텍스처가 이미 불러와져있지 않다면 불러옵니다.  Texture texture;  texture.id = TextureFromFile(str.C\_Str(), directory);  texture.type = typeName;  texture.path = str.C\_Str();  textures.push\_back(texture);  textures\_loaded.push\_back(texture); // 불러온 텍스처를 삽입합니다.  }  }  return textures;  } |

