Homework #2: Hello assimp

컴퓨터 그래픽스
국민대학교 소프트웨어학부

Abstract

본 과제에서는 Open Asset Import Library (Assimp)를 사용하여 간단한 모델 파일을 불러와서 OpenGL을 이용해 렌더링한다.

1 Open Asset Import Library

Open Asset Import Library (assimp)는 다양한 3D model의 데이터들을 assimp가 생성한 데이터 구조로 불러옴으로써 3DS, BLEND, DAE/Collada, FBX, OBJ 등을 포함한 대부분의 file format을 불러올 수 있다. assimp를 통해 model을 불러오면 assimp의 데이터 구조로 변환된다. File format의 유형과 관계없이 assimp의 데이터 구조로 import되기 때문에 모든 file format에 대해서 동일한 프로그램을 구성할 수 있다.
assimp 공식문서를 통해 더 자세한 내용을 확인할 수 있다.

2 assimp 설치 (ubuntu 16.04)

```
$ sudo apt-get install libassimp-dev assimp-utils
```

해당 명령어를 통해 assimp library API를 다운로드 할 수 있으며 assimp library API를 include하여 사용할 수 있다.
assimp 공식 github를 참조하여 assimp library API header file들을 확인할 수 있다.

2.1 assimp library API 사용법

assimp에서 기본적으로 사용되는 기능들의 헤더 파일들을 가져오게 된다.

```
#include <assimp/cimport.h>
#include <assimp/scene.h>           // Output data structure
#include <assimp/postprocess.h>     // Post processing flags
```

assimp를 사용한 코드를 컴파일하기 해서는 assimp 라이브러리 파일를 링크 시켜주어야 한다. Makefile에서 링커에 -lassimp 옵션으로 assimp 라이브러리 파일을 링크시키도록 한다.

```
HEADERS =
SOURCES = main.cpp
CC = g++
CFLAGS = -std=c++11
LDFLAGS = -lGL -lGLEW -lglfw -lassimp
EXECUTABLE = helloassimp
RM = rm -rf

all: $(SOURCES) $(HEADERS)
     $(CC) $(CFLAGS) -o $(EXECUTABLE) $(SOURCES) $(LDFLAGS)
```

2.2 파일 불러오기

```
bool load_asset(const std::string& filename)
{
  scene = aiImportFile(filename.c_str(), aiProcessPreset_TargetRealtime_MaxQuality);
  if(scene)
  {return true;}
  else
  {return false;}
}
```

- load_asset 함수
- file을 입력으로 주는 함수이며 함수내의 scene은 cimport.h header file로부터 file을 import를 할 수 있게 해준다. scene에 file이 import가 된다면 해당함수는 true를 return해주고, import 되지 않았다면 false를 return해준다.

2.3 mesh 정보 확인

읽어온 file의 mesh에 대한 정보들을 확인한다.

```
void print_scene_info(const aiScene* scene)
{
  std::cout << "num meshes in the scene " << scene->mNumMeshes << std::endl;
  for (int i=0; i < scene->mNumMeshes; ++i)
  {
    const aiMesh* mesh = scene->mMeshes[i];
    print_mesh_info(mesh);
  }
}
```

- 읽어온 file의 mesh에 대한 정보를 얻어오는 코드이다.
- scene의 'mNumMeshes'는 scene의 mesh의 갯수를 출력해준다.
- assimp의 데이터 구조를 보면 scene은 mesh에 대한 정보를 갖고 있다. 이를 통해 'aiMesh* mesh'에 해당 scene의 mesh의 갯수에 맞게끔 mesh에 대한 정보를 저장한다.
- 'print_mesh_info()' 함수를 호출하여 mesh 내에 vertex에 대한 정보들을 확인한다.

2.4 vertex의 position & color & face 정보 확인

file로 부터 읽어온 vertex의 position과 color와 face정보들을 확인한다.

```
void print_mesh_info(const aiMesh* mesh)
{
  std::cout << "num vertices: " << mesh->mNumVertices << std::endl;
  for (int i = 0; i < mesh->mNumVertices; ++i}
  {
    aiVector3D vertex = mesh->mVertices[i];
    std::cout<<"vertex ("<<vertex.x<<","<<vertex.y<<","<< vertex.z <<")"<< std::endl;
  }

  if (mesh->mColors[0] != NULL)
  {
    aiColor4D color = mesh->mColors[0][i];
    std::cout<<"color ("<<color.r<<","<<color.g<<","<<color.b<<","<<color.a<<")"<<std::endl;
  }

  std::cout << "num faces " << mesh->mNumFaces << std::endl;
  for (int i=0; i < mesh->mNumFaces; ++i)
  {
    const aiFace& face = mesh->mFaces[i];

    if(face.mNumIndices == 1)
    {
      std::cout << "  point" << std::endl;
```

```
    }
    else if(face.mNumIndices == 2)
    {
      std::cout << "  line" << std::endl;
    }
    else if(face.mNumIndices == 3)
    {
      std::cout << "  triangle" << std::endl;
    }
    else
    {
      std::cout << " polygon" << std::endl;
    }

    for (int j=0; j < face.mNumIndices; ++j)
    {
      std::cout << "    index " << face.mIndices[j] << std::endl;
    }
  }
}
```

- mesh의 'mNumVertices'를 통해 각 mesh마다 vertex의 갯수를 알 수 있다.
- assimp의 데이터 구조를 보면 mesh는 vertex에 대한 정보를 갖고 있으며, 이를 통해 'aiVector3D vertex'에 각 vertex에 대한 position 정보를 저장하고 mesh에 color가 있다면 'aiVector4D color'에 각 vertex에 대한 color를 저장한다.

- mesh의 'mNumFaces'를 통해 각 mesh마다 몇개의 face를 갖는지 알 수 있다.
- face는 primitive (점 선, 삼각형, ...)를 형성하기위한 vertex들의 index 정보를 갖고 있다. assimp는 vertex와 index들이 분리되어 있기 때문에 index buffer를 통해 랜더링을 쉽게 할 수 있다.
- 'aiFace& face'에 mesh의 face정보를 저장하고, face의 index의 갯수가 1, 2, 3, ... 에 따라서 해당 face가 어떠한 형태를 갖는지 알 수 있다.
- index의 값들을 'face.mIndices[]'에 저장되어 index에 대한 값들을 알 수 있다.

3 ASSIMP 데이터 구조

assimp의 데이터 구조는 다음과 같다.

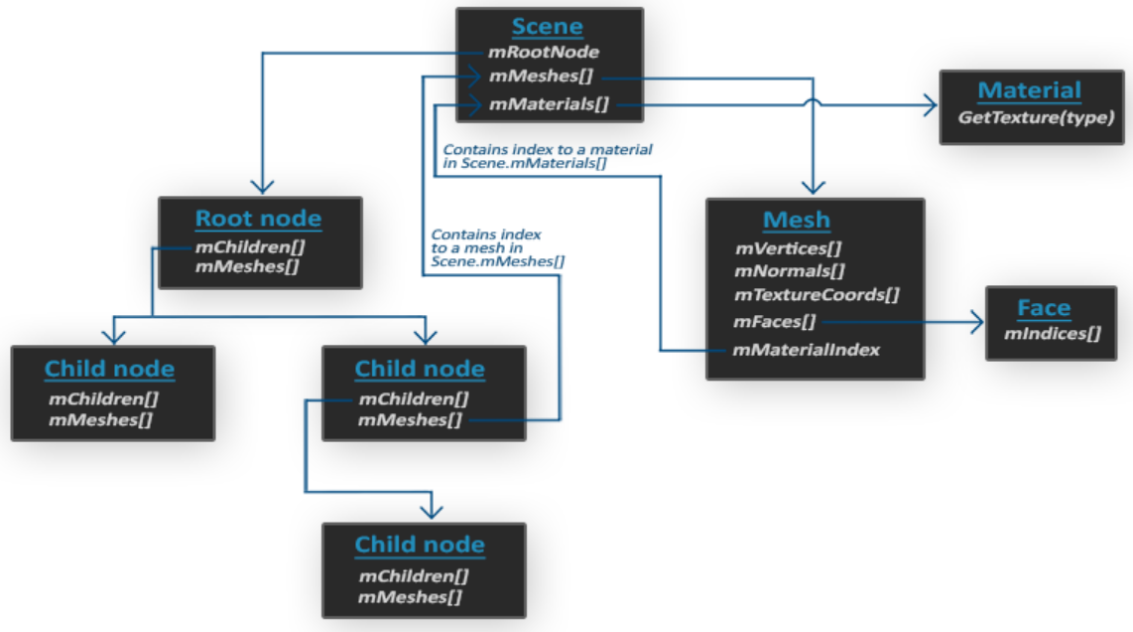
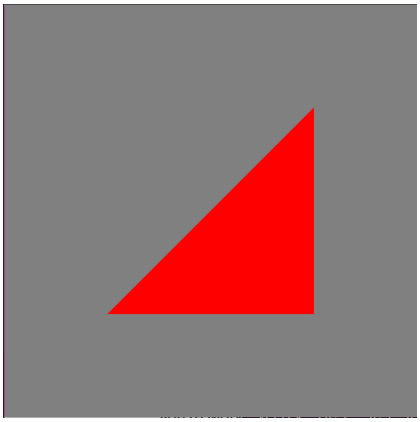


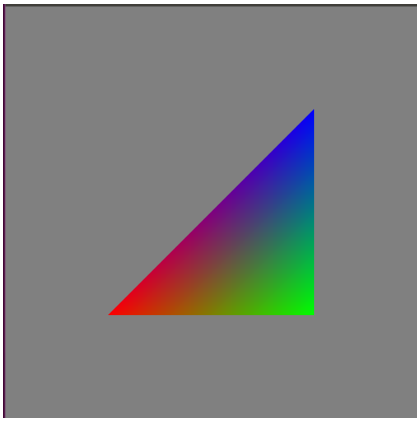
Fig. 1: assimp data structure

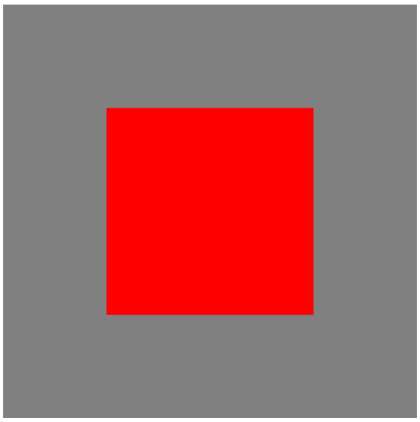
- scene 객체는 root node, mesh, material에 대한 정보들을 갖고 있다.
- scene의 root node는 child node들을 포함할 수 있고 그 child node들은 또 다른 child node들을 포함할 수 있다. scene 객체의 mMeshes[] 배열 안의 데이터를 가리키는 index 정보들의 집합을 가지고 있다. root node의 mMeshes[] 배열은 실제 mesh 객체들을 가지고 있고 일반 node의 mMeshes[] 배열에 들어있는 값은 오직 scene의 mesh 배열에 대한 index 정보들만 가지고 있다.
- mesh 객체는 랜더링하는 데에 관련한 모든 데이터를 포함하고 있다. 객체를 구성하는 vertex positions, normal vectors, texture coordinates, faces, material 등을 포함한다.
- mesh는 여러개의 face들을 가진다. face는 객체의 primitive (삼각형, 사격형, 점 등)을 나타낸다. face는 primitive를 형성하기 위한 vetex들의 index 정보들을 가지고 있다. assimp는 vertex들과 index들이 분리되어 있기 때문에 index buffer를 통해 랜더링하는 것을 쉽게 만들어 준다.
- mesh는 material 객체를 갖는다. material 객체는 material의 attribute를 얻기 위한 함수들을 관리한다.

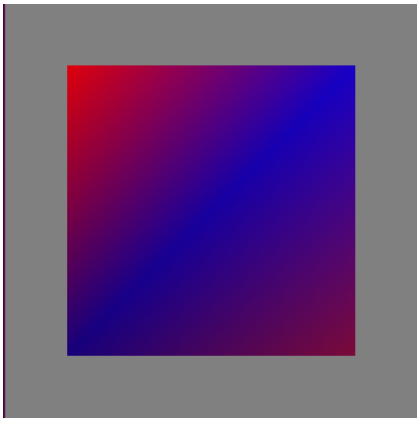
4 실행 방법 및 결과

실행 방법으로 makefile을 활용하여 컴파일을 진행한 후, 실행 결과로 assimp library API를 활용하여 model을 불러온 후, 해당 모델을 화면에 띄우는 것을 목표로한다. 아래의 사진은 실행 결과의 예이다.
반드시 "./helloassimp [filename]" 형식으로 실행시킬 것!!!


triangle_with_reds.ply


triangle_with_vertexcolors.ply


rect_with_reds.ply


cube_with_vertexcolors.ply


bunny.ply


face.ply

5 과제 제출 방법(매우 중요!!!)

- 본 과제는 개인과제이며, 각자 자신의 코드를 완성하도록 한다.
- 공지된 마감 시간까지 과제 코드를 가상대학에 업로드하도록 한다.
- 과제 코드는 **Ubuntu 16.04 LTS 환경에서 make 명령으로 컴파일 가능**하도록 작성한다.
- 과제 코드는 다음의 파일들은 하나의 압축파일로 묶어 **tar.gz** 파일 형식이나 표준 **zip** 파일 형식으로만 제출하도록 한다. 이 때, 압축파일의 이름은 반드시 'OOOOOOOO_HW02.tar.gz(OOOOOOOO은 자신의 학번)'과 같이 자신의 학번이 드러나도록 제출한다.

  1) 소스코드 및 리소스 파일들
  2) Makefile

- 과제에 관한 질문은 오피스 아워를 활용하도록 한다. 교육조교(teaching assisant, TA)에게 메일로 약속시간을 정한 후, zoom 등을 활용한 방법을 통해 물어보는 것도 매우 권장하는 방법이다.