**2016**

**MIDAS Challenge**

**그래픽스 개발분야 – OpenGL Project 설명**

**1. Solution 구조**

1) bin

- debug / release 폴더 구분. \*.dll과 \*.exe 파일.

- assimp-vc110-mt.dll : obj 파일 관리 라이브러리

- opencv\_world310.dll : 이미지 관리 위한 OpenCV 라이브러리

2) glsl - glsl 쉐이더 소스

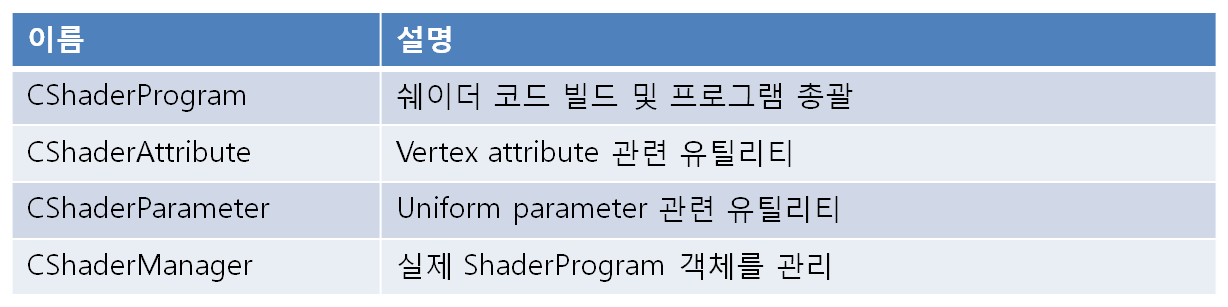
3) include – 추가 include 파일

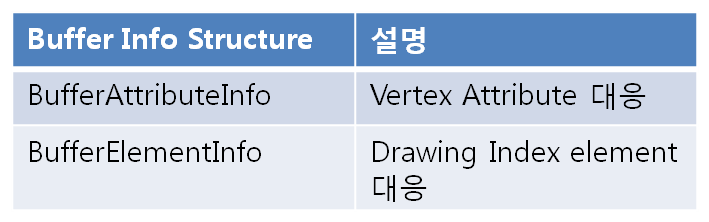
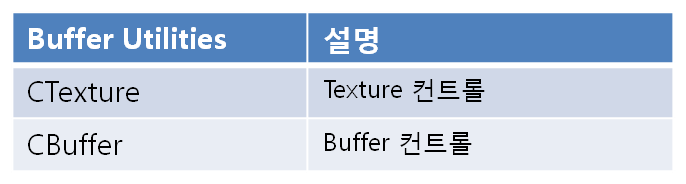
4) lib – 프로젝트 구성과 관계 없는 common lib과 구성에 따른 debug/release lib

5) models – obj 모델

6) src – 솔루션 파일과 MChallenge 프로젝트 폴더.

**2. Shader Utility**

****

<그림1. Shader Utility 개괄>

<그림 2. Shader Utility 구조>

**2.1. Shader Program의 생성 및 사용**

CShaderProgram 클래스는 Shader의 생성/삭제, Bind/Unbind, Compile, Link등을 수행하는 클래스 입니다. 사용법은 우선 GLCreate() 함수로 shader program 객체를 생성하고, bool GLAttachShader(GLenum target, const GLchar\* sFileName ) 함수로 \*.glsl 파일을 읽어 해당 쉐이더 소스를 컴파일 합니다. 모든 Shader 소스가 attach 된 후에는 GLLink() 함수를 호출하여 링크를 수행합니다. Link가 성공하면 GLLink() 함수 내부에서 GLBuildAttributeMap() 을 호출하는데, 이를 통하여 Vertex Attribute Location을 받아옵니다.

CShaderManager에서 실제로 상기한 내용들을 수행하며, CShaderManager 클래스는 이렇게 생성한 Shader Program을 개발자가 정의한 Shader Type을 키로 하는 맵 형태로 관리합니다.

본 예제에는 SHADERTYPE\_PHONG 이라는 Shader Type에 해당하는 CShaderProgram이 있습니다.

**2.2. VBO 의 생성 및 사용**

CBuffer 클래스는 여러 종류의 OpenGL 버퍼 데이터를 관리하기 위하여 사용되는 구조입니다. GLGenerate()를 통해 생성합니다. SetTarget() 함수로 직접 Target을 지정할 수 있습니다. SetAsAttribute()의 경우 target을 GL\_ARRAY\_BUFFER로, SetAsElement()의 경우는 GL\_ELEMENT\_ARRAY\_BUFFER 로 지정합니다.

GLData() 함수는 OpenGL Buffer에 실제 데이터를 올립니다. GLvoid\* data 인자에 데이터를 주면 데이터를 올리고, 해당 인자를 NULL로 전달하는 경우 메모리 영역만 미리 잡습니다. GLSubData()는 이렇게 잡힌 메모리 영역의 일부분에 데이터를 올립니다.

BufferInfo.hpp에 BufferAttribInfo와 BufferElementInfo 라는 구조체가 있습니다. BufferAttribInfo는 CBuffer로 정의한 VBO에서 Vertex Attribute Pointer 를 지정해주는 구조체입니다. BufferElementInfo는 CBuffer로 정의한 Index Buffer로 렌더링하기 위해 사용되는 구조체입니다.

정리하자면 CBuffer로는 VBO와 IBO(Index Buffer Object)를 생성할 수 있으며, VBO에 Vertex Attribute를 특정하기 위하여 BufferAttribInfo 구조체를, IBO의 index를 이용한 Render Call 을 수행하기 위하여 BefferElementInfo 구조체를 사용하게 됩니다.

렌더 데이터를 파일로부터 읽어서 구성하는 CSceneData 클래스의 MakeMeshVBO() 함수에서는 데데이터부터 VBO 정보를 구성하는 실제 예시가 있습니다.

**2.3. Shader Attribute, Shader Parameter, Texture**

Shader에서 입력으로 받는 stream data 는 보통 Attribute Location을 할당한 후에 해당 Location에 Vertex Attribute Array 연결하여 전달합니다. 이 과정은 때때로 상당히 긴 양의 코드를 필요로 하며 여러 shader의 개별 location들을 관리하는 일이 쉽지 않을 수 있습니다. CShaderAttribute 클래스는 이 과정을 객체화 하여 간단하게 관리합니다. GLInitAttrib() 함수를 통해 location을 할당하고 2.2. 의 VBO 파트에서 만든 BufferAttribInfo 구조체를 객체에 전달한 후 CShaderAttribute의 객체를 가진 ShaderProgram이 자신의 Attribute를 활성화 하는 순간 ( CShaderProgram::GLAttachAttributes() ) VBO의 각각의 Attribute들이 Shader와 성공적으로 연결됩니다.

CShaderParameter는 다양한 Uniform 값들을 제어합니다. ShaderParameter.hpp 에서는 CShaderParamFloat, CShaderParamVec3 등 다양한 상속 클래스들을 지원합니다.

텍스쳐는 CTexture 클래스를 통해 관리합니다. CTexture는 OpenGL에서 지원되는 Shader 함수들을 감싼 텍스쳐 객체입니다. 실제 Shader에서 Texture를 사용하는 경우는 CTexture 객체를 생성한 후 상기한 Shader의 CShaderParamTexture 에 CTexture를 할당함으로써 Shader에 texture를 전달할 수 있습니다. 다음은 색을 갖는 텍스쳐를 생성하여 shader parameter로 넘기는 예 입니다.

**// 텍스쳐 생성 및 로딩**

**cv::Mat decalTexImage; // image가 로드 된 openCV 데이터**

**CTexture\* pDecalTex = new CTexture(); // 텍스쳐 객체 생성**

**pDecalTex->GLGenerate(); // OpenGL TexGen**

**pDecalTex->GLAllocateAsImage2D( width, height, internalformat, format, type ); // 할당**

**pDecalTex->GLLoadImage2D( data ); // 데이터 채우기**

**pDecalTex->GLSetFilterLinear(); // Linear Filter**

**pDecalTex->GLSetWrap( GL\_REPEAT );**

**…**

**// shader 사용하는 렌더러 부분의 shader parameter,**

**// “decaltex”는 fragment shader내의 uniform sampler2D 이름.**

**CShaderParamTexture\* pParamDecalTex = NULL;**

**// 초기화 : Uniform Location 받아오기**

**pParamDecalTex = pShaderProgram->GLGetPArameter( SHP\_TEXTURE, “decaltex” );**

**// 값 전달**

**\*pParamDecakTex = \*pDecalTex;**

**// 실제 glUniformXXX 호출**

**pShaderProgram->GLAttach();**

**2.4. Shader가 빌드되어 렌더링 콜이 일어나는 전체 예시**

- CSceneData에서 VBO 및 Material 로드 및 데이터 빌드

- Shader Initialzation

- ShaderProgram 생성 및 Shader 컴파일, 링크

- ShaderAttribute와 ShaderParam 초기화

- Shader를 이용한 렌더링 과정.