

게임엔진

제4강 3D 모델 렌더링

한국산업기술대학교 이대현



학습 안내

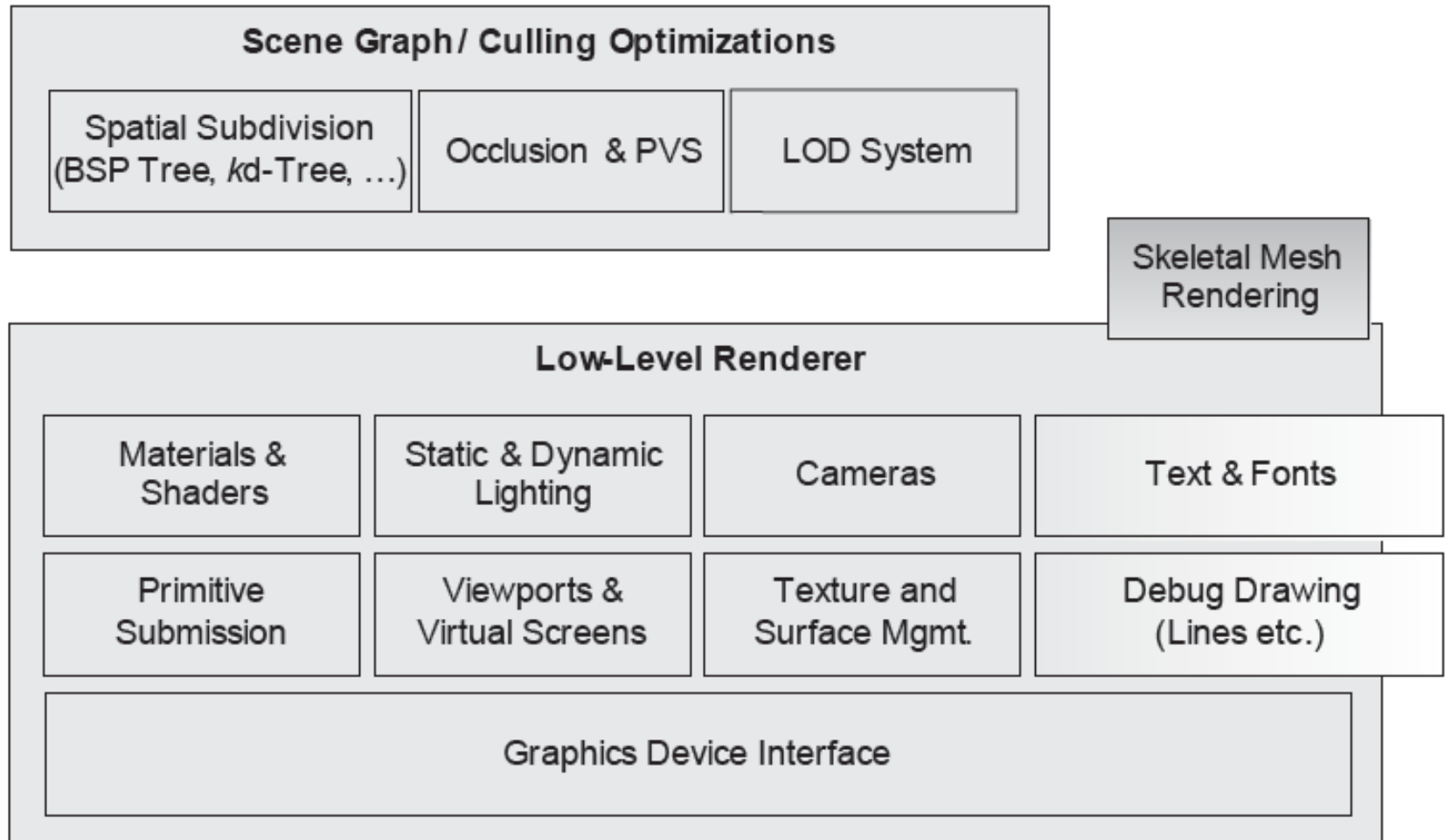
■ 학습 목표

- 3D 렌더링 과정을 이해한다.
- 오우거 엔진을 이용하여 3D 모델을 렌더링한다.

■ 학습 내용

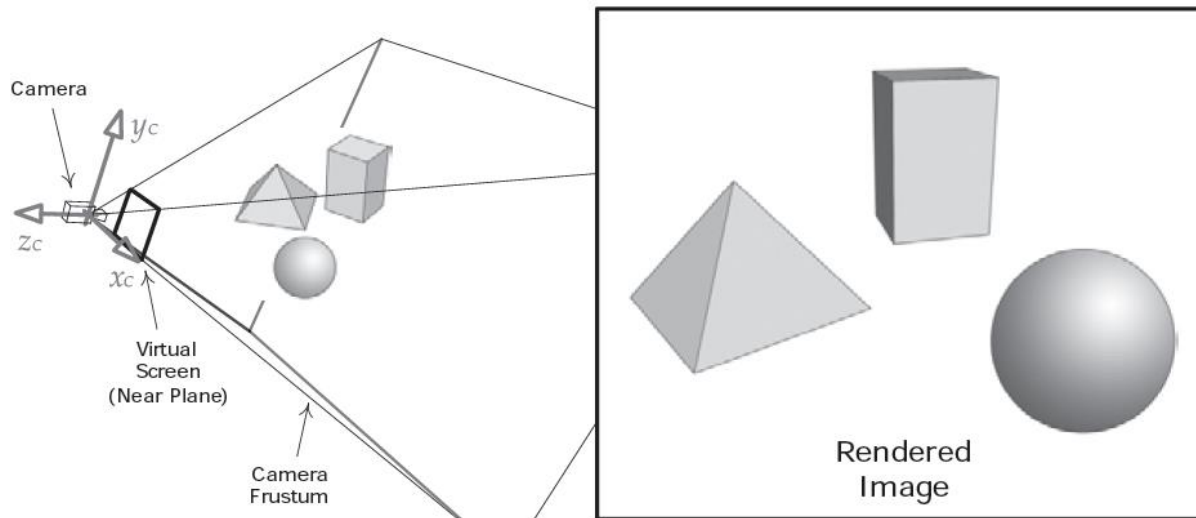
- 3D 렌더링 과정
- 오우거 렌더 윈도우의 생성
- 3D 모델의 렌더링
 - 3D 모델의 표현
 - 장면 노드
 - 3D 모델의 렌더링 과정

게임 엔진 아키텍처에서 렌더링 프로세스의 위치



3D 씬(Scene) 렌더링 프로세스 개관

- 3차원 표면으로 표시된 가상의 씬 구성
- 가상의 카메라로 씬을 바라보게 만듦
- 다양한 광원들이 정의됨
- 씬 내에 존재하는 표면들의 시각적인 속성이 계산/결정



씬의 구성

■ 실제 세계의 씬은 다양한 물체들로 구성

- 물체는 3D 공간에서 부피를 차지
- 단단한 물체(벽돌) vs. 형상이없는 물체(물, 연기)
- 불투명, 투명, 반투명

■ 컴퓨터그래픽에서 대부분의 물체는 “단단한 물체”로 가정.

3D 모델의 표현: 폴리곤과 메쉬

■ 폴리곤(Polygon)

- 3개의 점이 모이면 하나의 면(face)을 만들 수 있다. 이렇게 3개의 점으로 만들어진 삼각형을 폴리곤이라 한다.

■ 메쉬(Mesh)

- 폴리곤들이 모여서 하나의 3차원 물체를 만들게 되는데 이것을 메쉬라고 부른다. 다시 말해 메쉬는 폴리곤이 모여서 만들어진 3차원 공간의 객체(object)다.



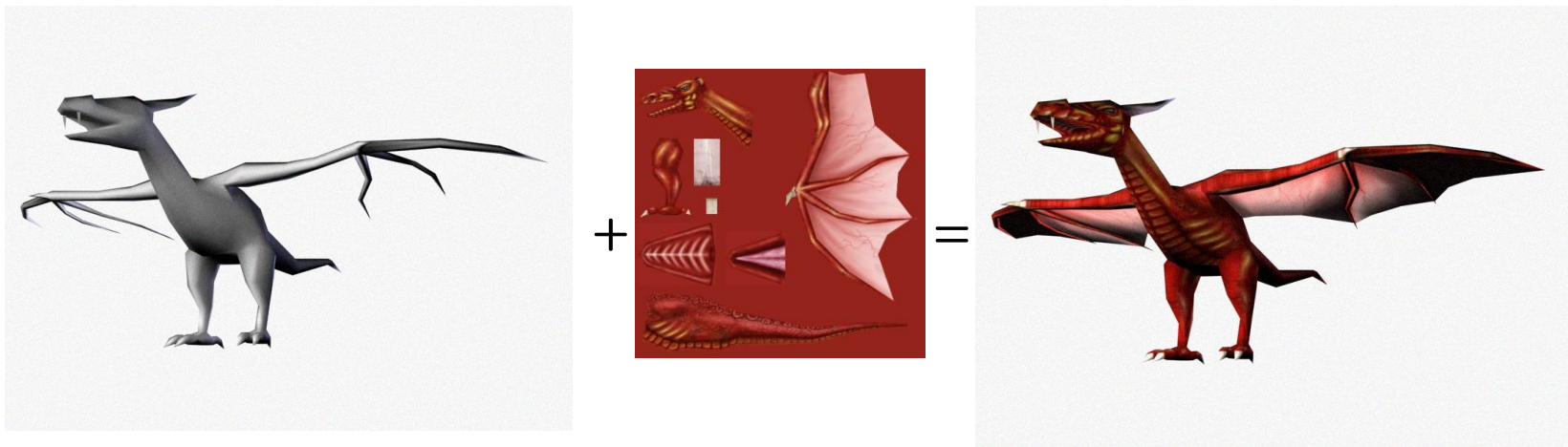
3D 모델의 표현: 텍스처 매핑

■ 텍스처

- 폴리곤만으로 화면에 물체를 나타내는 것은 제약을 많이 갖게 된다. 3차원 물체에 2차원의 이미지 (jpg, bmp, gif 등)를 입혀서 다양한 표현을 가능하게 해준다.
- 텍스처는 표면위의 점의 색상 계산을 오프라인으로 미리 계산해놓은 것이다. → 일일이 계산하는 것보다 훨씬 속도가 빠르다(근사 최적화).

■ 텍스처 매핑

- 3차원 물체에 텍스처를 입히는 것을 텍스처 매핑이라 하고, 매핑되는 2차원 이미지를 텍스처, 혹은 텍스처 맵이라고 부른다.



실습



MakeRenderWindow
렌더윈도우의 생성

- Git pull

main.cpp (1)



```
void go(void)
{
    // OGRE의 메인 루트 오브젝트 생성
    #if !defined(_DEBUG)
        mRoot = new Root("plugins.cfg", "ogre.cfg", "ogre.log");
    #else
        mRoot = new Root("plugins_d.cfg", "ogre.cfg", "ogre.log");
    #endif

    // 초기시작의 컨피규레이션 설정 - ogre.cfg 이용
    if (!mRoot->restoreConfig()) {
        if (!mRoot->showConfigDialog()) return;
    }

    // 렌더 윈도우의 생성
    mWindow = mRoot->initialise(true, "Make Render Window");

    // ESC key를 눌렀을 경우, 오우거 메인 렌더링 루프의 탈출을 처리
    size_t windowHnd = 0;
    std::ostringstream windowHndStr;
    OIS::ParamList pl;
    mWindow->getCustomAttribute("WINDOW", &windowHnd);
    windowHndStr << windowHnd;
    pl.insert(std::make_pair(std::string("WINDOW"), windowHndStr.str()));
    mInputManager = OIS::InputManager::createInputSystem(pl);
    mKeyboard = static_cast<OIS::Keyboard*>(mInputManager->createInputObject(OIS::OISKeyboard, false));
    mESCListener = new ESCListener(mKeyboard);
    mRoot->addFrameListener(mESCListener);
```



```
// 기본 씬매니저의 생성
```

```
mSceneMgr = mRoot->createSceneManager(ST_GENERIC);
```

```
// 카메라의 생성 및 설정
```

```
mCamera = mSceneMgr->createCamera("camera");
```

```
mCamera->setPosition(500.0f, 100.0f, 0.0f);
```

```
mCamera->lookAt(0.0f, 100.0f, 0.0f);
```

```
mCamera->setNearClipDistance(5.0f);
```

```
// 뷰포트의 생성 및 설정
```

```
mViewport = mWindow->addViewport(mCamera);
```

```
mViewport->setBackgroundColour(ColourValue(0.0f,0.0f,0.0f));
```

```
mCamera->setAspectRatio(Real(mViewport->getActualWidth()) /  
                        Real(mViewport->getActualHeight()));
```

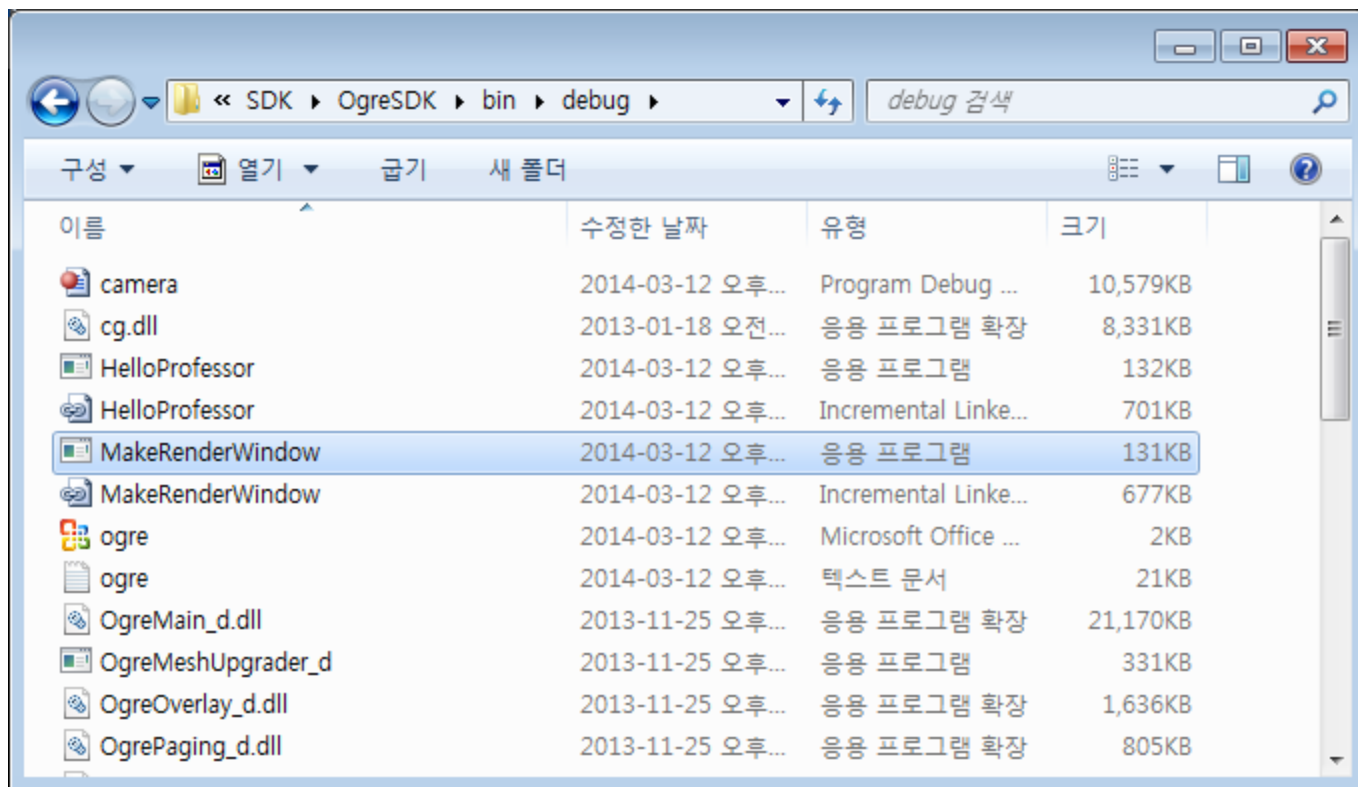
```
// 렌더링 루프의 실행
```

```
mRoot->startRendering();
```

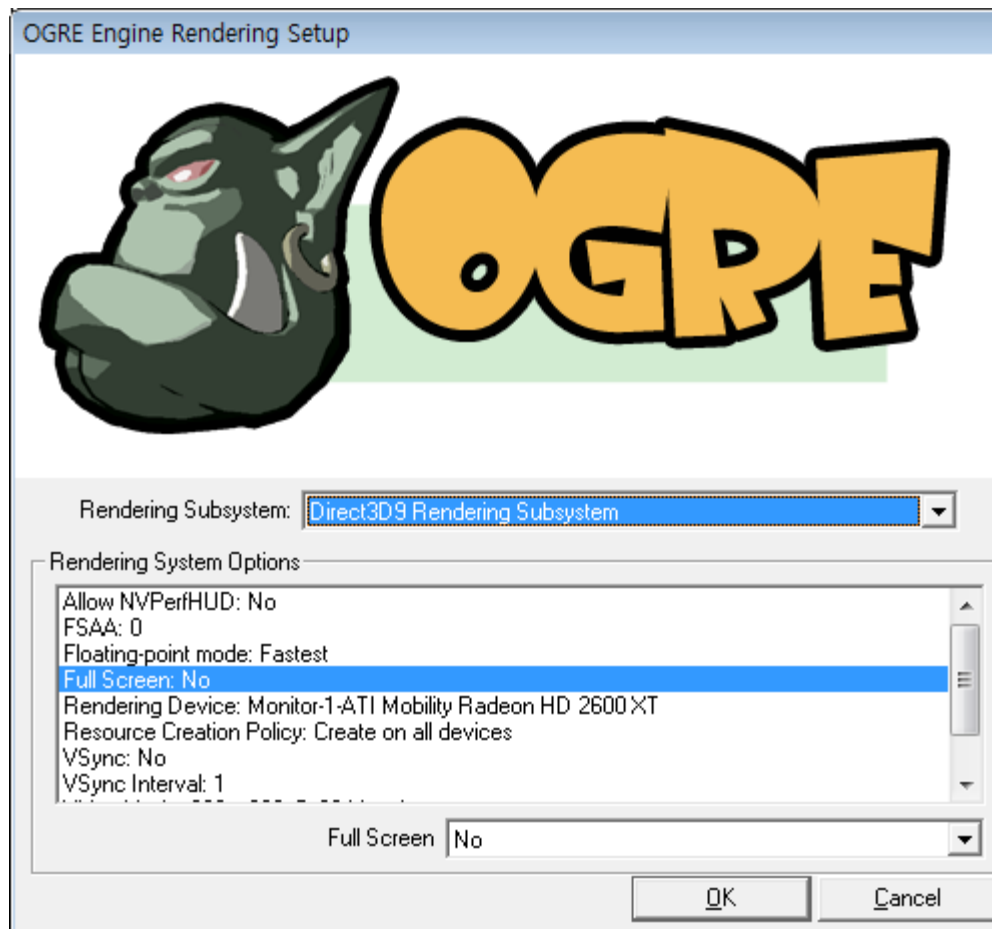
```
delete mRoot;
```

```
}
```

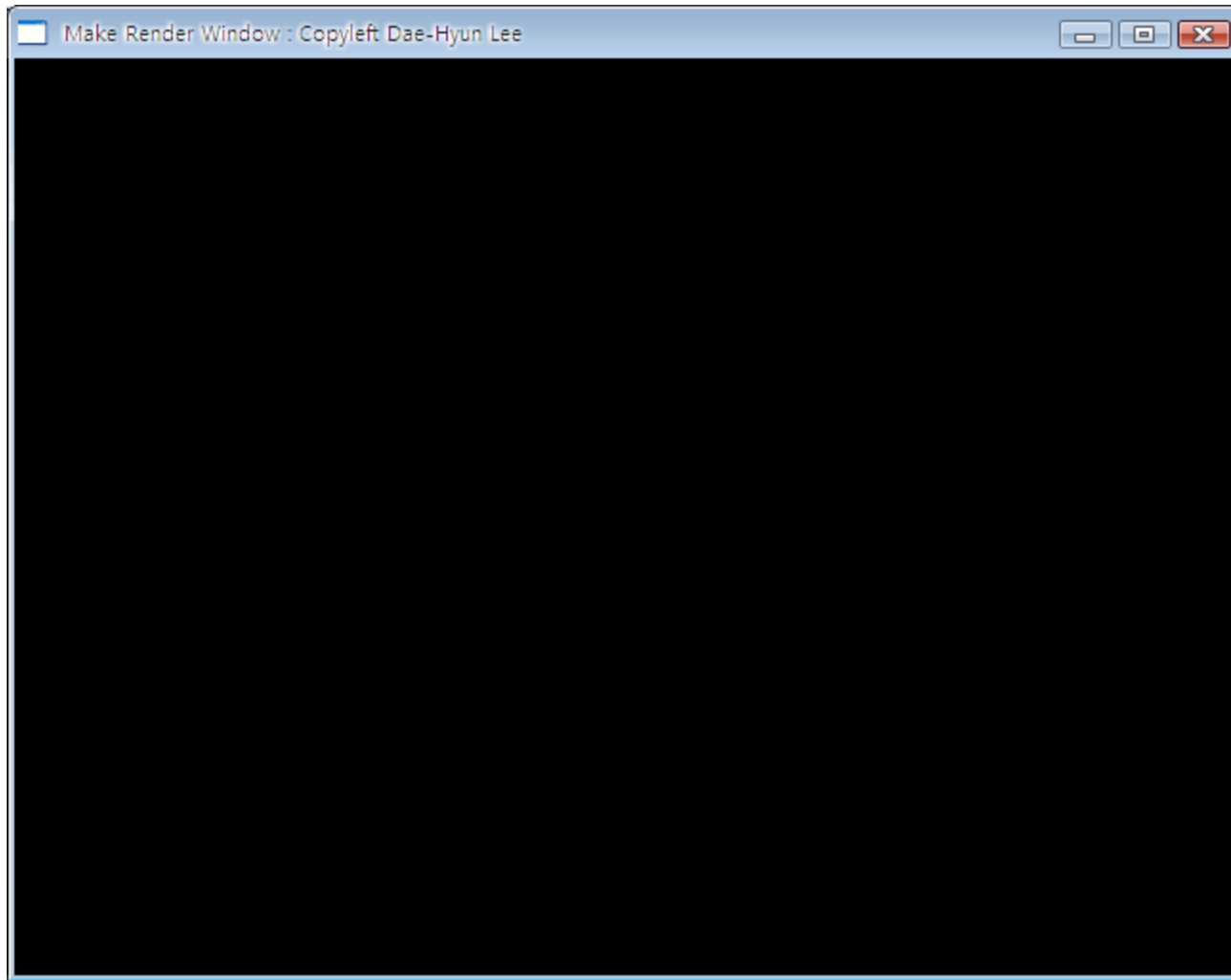
실행 폴더 안의 생성된 파일들



프로그램 실행 - Configuration Dialog 설정



실행 결과 - 아무것도 없는 빈화면!! ESC 로 중단할 수 있음.



Step #1: Root 객체의 생성

- 플러그인 프로그램들의 지정.

```
# Defines plugins to load

# Define plugin folder
PluginFolder=.

# Define plugins
Plugin=RenderSystem_Direct3D7
Plugin=RenderSystem_Direct3D9
Plugin=RenderSystem_GL
Plugin=Plugin_ParticleFX
Plugin=Plugin_BSPSceneManager
Plugin=Plugin_OctreeSceneManager
Plugin=Plugin_CgProgramManager
```

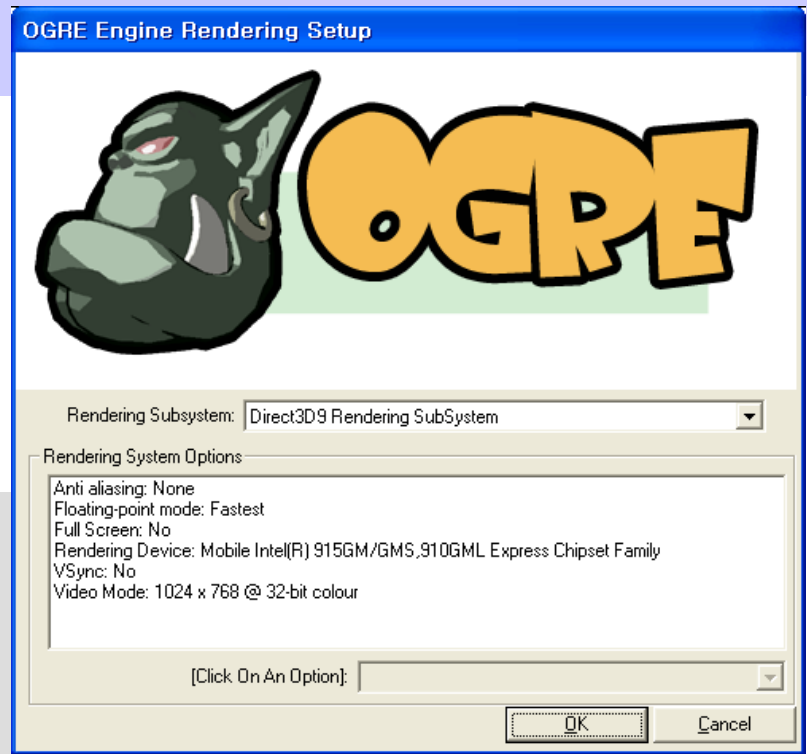
```
#if !defined(_DEBUG)
    mRoot = new Root("plugins.cfg", "ogre.cfg", "ogre.log");
#else
    mRoot = new Root("plugins_d.cfg", "ogre.cfg", "ogre.log");
#endif
```

- 오우거엔진의 초기 렌더링 옵션 설정에 대한 정보를 저장하는 파일.

Render System	Direct3D9 Rendering SubSystem
Anti aliasing	None
Floating-point mode	Fastest
Full Screen	No
Rendering Device	Mobile Intel(R) 915GM/GMS,910GML Express Chipset Family
VSync	No
Video Mode	640 x 480 @ 32-bit colour

Step #2: 컨피규레이션 설정

```
if (!mRoot->restoreConfig()) {  
    if (!mRoot->showConfigDialog())  
        return;  
}
```



- 사용자의 선택에 따른, *configuration*이 설정되고, 설정된 내용이 앞서 지정했던 “*ogre.cfg*” 파일에 저장된다.

Render System	Direct3D9 Rendering SubSystem
Anti aliasing	None
Floating-point mode	Fastest
Full Screen	No
Rendering Device	Mobile Intel(R) 915GM/GMS,910GML Express Chipset Family
VSync	No
Video Mode	1024x768 @ 32-bit colour

Step #3: 렌더 윈도우의 설정

■ RenderWindow* Root::initialise (

bool *autoCreateWindow*,

const String & *windowTitle* = "OGRE Render Window");

- *autoCreateWindow*: 현재 설정된 렌더 시스템을 기반으로 하여, 렌더 윈도우를 자동으로 생성함.
- *windowTitle*: 생성된 윈도우의 이름

```
mWindow = mRoot->initialise(true, "Make Render Window ");
```

Step #4: ESC 키를 처리하는 프레임 리스너 설정

```
size_t windowHnd = 0;
std::ostringstream windowHndStr;
OIS::ParamList pl;

mWindow->getCustomAttribute("WINDOW", &windowHnd);
windowHndStr << windowHnd;
pl.insert(std::make_pair(std::string("WINDOW"), windowHndStr.str()));

mInputManager = OIS::InputManager::createInputSystem(pl);
mKeyboard = static_cast<OIS::Keyboard*>(mInputManager-
    >createInputObject(OIS::OISKeyboard, false));
mESCListener = new ESCListener(mKeyboard);
mRoot->addFrameListener(mESCListener);
```

Step #5: 씬관리자(Scene Manager)의 생성

■ 씬관리자(Scene Manager)

- 화면 상에 보여지는 모든 객체들을 종합적으로 관리
- 카메라, 광원, 평면과 같은 것들도 장면 관리자의 관리 대상
- 장면 관리자의 종류
 - 옥트리(Octree: 8진 트리) 장면 관리자
 - 기본 장면 관리자
 - 대부분의 장면에 사용 가능
 - BSP(Binary Surface Partition) 장면 관리자
 - 건물 내부와 같이 벽과 복도 등으로 구성된 장면에 최적화된 성능.
 - 지형(Terrain) 장면 관리자
 - 정적 지형을 가지는 비교적 소규모의 장면에 적합
 - 고해상도의 지형에 적합

```
mSceneMgr = mRoot->createSceneManager(ST_GENERIC);
```

Step #6: 카메라의 생성 및 설정

```
mCamera = mSceneMgr->createCamera("camera");  
  
mCamera->setPosition(500.0f, 100.0f, 0.0f);  
  
mCamera->lookAt(0.0f, 100.0f, 0.0f);  
  
mCamera->setNearClipDistance(5.0f);
```

Step #7: 뷰포트의 생성 및 설정

```
mViewport = mWindow->addViewport(mCamera);  
  
mViewport->setBackgroundColour(ColourValue(0.0f,0.0f,0.0f));  
  
mCamera->setAspectRatio(Real(mViewport->getActualWidth()) /  
                        Real(mViewport->getActualHeight()));
```

Step #8: 렌더링 루프의 실행

```
mRoot->startRendering();
```



- 설정된 장면(Scene)을 카메라가 캡처한 내용을 뷰포트를 통해서 스크린으로 렌더링.
- 무한 반복되나, 프레임 리스너에 의해서 ESC키이가 눌린 경우 탈출한다.

Step #9: Root 객체의 해제

```
delete mRoot;
```

실습



HelloProfessor
3D 모델의 렌더링



```
void go(void)
{
    // ... 전략

    ResourceGroupManager::getSingleton().addResourceLocation("resource.zip", "Zip");
    ResourceGroupManager::getSingleton().initialiseAllResourceGroups();

    mSceneMgr->setAmbientLight(ColourValue(1.0f, 1.0f, 1.0f));

    Entity* daehyunEntity = mSceneMgr->createEntity("Daehyun", "DustinBody.mesh");

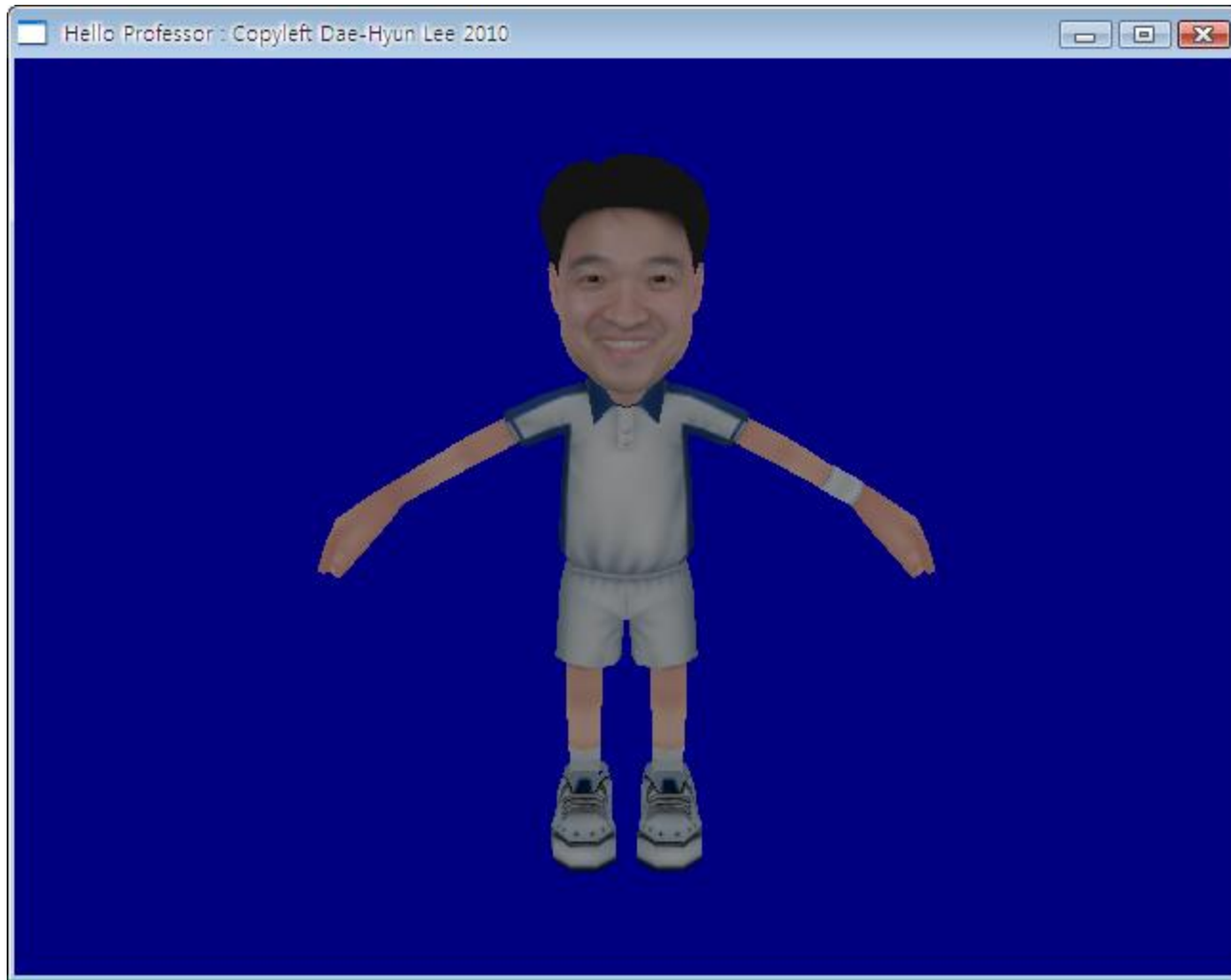
    SceneNode* daehyunNode = mSceneMgr->getRootSceneNode()->createChildSceneNode();
    daehyunNode->attachObject(daehyunEntity);

    mRoot->startRendering();

    mInputManager->destroyInputObject(mKeyboard);
    OIS::InputManager::destroyInputSystem(mInputManager);

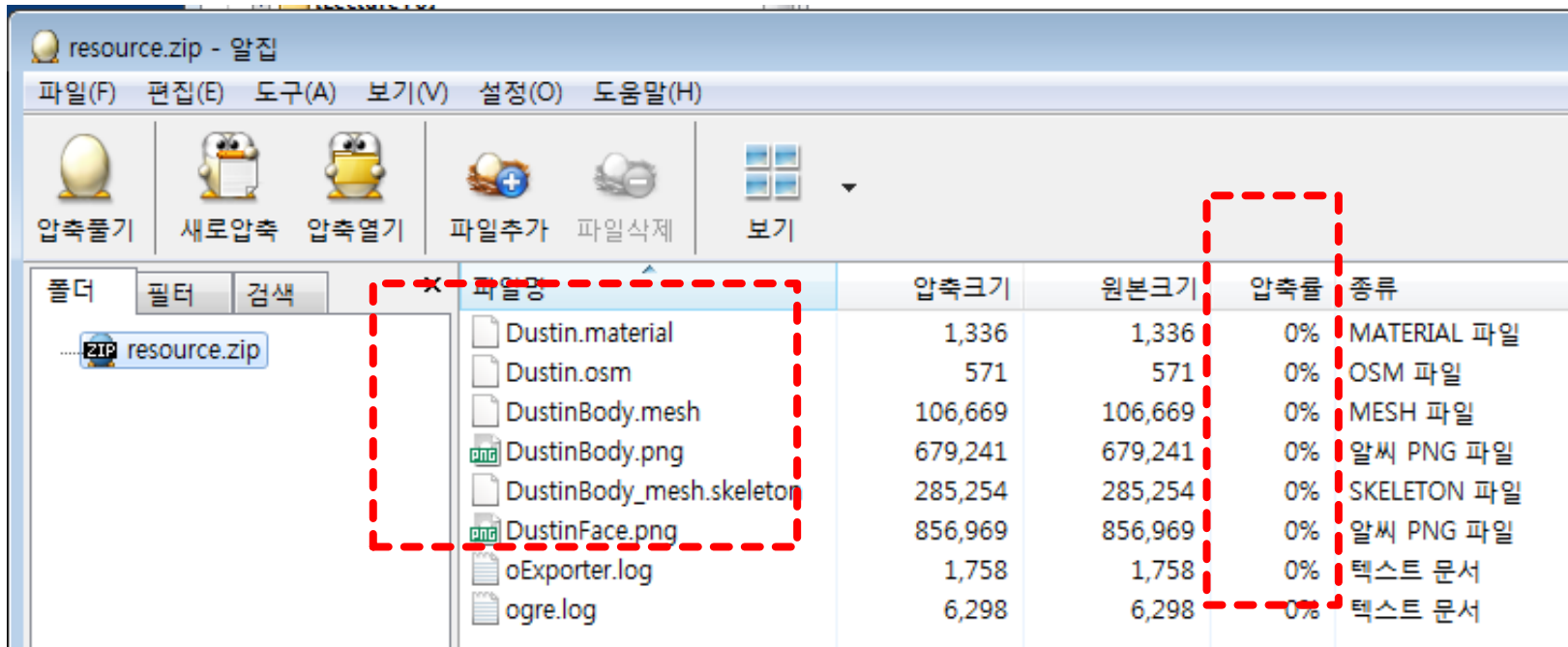
    // ... 후락
}
```

실행 화면 - Hello Professor



오우거의 3D 모델 파일

- OOO.mesh - 메쉬 데이터
- OOO.material - 메쉬의 재질 속성을 기술
- 그래픽 툴로 모델링한 후 OgreMax와 같은 export 툴을 이용하여, export 하게 됨.
- 리소스 압축시, 압축하지 않고, 그냥 합쳐놓는 것이 나중에 로딩 타임을 줄이는데 유리.



폴더	파일명	압축크기	원본크기	압축률	종류
.....ZIP resource.zip	Dustin.material	1,336	1,336	0%	MATERIAL 파일
	Dustin.osm	571	571	0%	OSM 파일
	DustinBody.mesh	106,669	106,669	0%	MESH 파일
	DustinBody.png	679,241	679,241	0%	알씨 PNG 파일
	DustinBody_mesh.skeleton	285,254	285,254	0%	SKELETON 파일
	DustinFace.png	856,969	856,969	0%	알씨 PNG 파일
	oExporter.log	1,758	1,758	0%	텍스트 문서
	ogre.log	6,298	6,298	0%	텍스트 문서

Resource.zip 의 내용

Step #1: 리소스 위치 지정

- 리소스 파일들의 경로 지정

```
ResourceGroupManager::getSingleton().addResourceLocation(
```

```
"resource.zip",
```

- 리소스 파일의 경로

```
"Zip");
```

- 리소스 타입: *FileSystem* 또는 *Zip*


```
ResourceGroupManager::getSingleton().initialiseAllResourceGroups();
```

- 사용될 모든 리소스들을 초기화

Step #2: 광원(Light)의 설정

- 물체를 보려면 빛이 있어야 한다 → 광원의 설정 필요
- 광원의 종류
 - 주변광(Ambient Light), 점 광원(Point Light), 방향성 광원(Directional Light), 점적 광원(Spot Light)
- 주변광(Ambient Light)
 - 모든 방향에서 빛의 세기가 동일한 광원.
 - 물체의 모든 부분이 균일하게 빛나게 됨.
 - 광원의 방향과 위치가 없으며, 색깔만 지정.

```
mSceneMgr->setAmbientLight(ColourValue(1.0f, 1.0f, 1.0f));
```



• $RGB=(1,1,1)$ 가장 밝은 흰색

Step #3: 엔터티의 생성

■ 엔터티(Entity)

- 장면 위에 표시(렌더링)되는 물체. Ex) 로봇, 물고기, 자동차, 캐릭터
- 지형 역시 엔터티로 간주됨.
- 광원, 입자, 카메라는 엔터티가 아님.

```
Entity* daehyunEntity = mSceneMgr->createEntity("Daehyun", "DustinBody.mesh");
```

- 엔터티의 이름.
- 모든 엔터티는 서로 다른 이름을 가짐.

- 엔터티에 사용될 메시의 파일 이름.
- 리소스 위치로 지정된 경로에 위치함.

Step #4: 씬노드의 생성

■ 씬노드(Scene Node)

- 장면 노드: 장면의 관리를 위한 트리 구조를 구성하는 기본 요소.
- 오우거 엔진은 엔터티와, 엔터티의 위치 및 방향 정보를 분리해서 처리함.
- 장면 위에 엔터티를 직접 배치할 수 없음.
- 간접적인 방법을 사용 → 장면 노드의 이용
- 엔터티, 카메라, 광원 등 모든 물체는 장면 노드에 담김.
- 물체의 위치 및 방향 정보를 담고 있음.
- 장면 관리자는 한 개의 최상위 장면 노드를 가짐.
- 장면 노드의 위치는 상위 장면 노드에 대해서 항상 상대적으로 값으로 표현.

- 씬노드의 생성.
- 모든 씬노드는 서로 다른 이름을 가짐.

- 현재 씬매니저의 최상위 노드를 획득.

```
SceneNode* daehyunNode = mSceneMgr->getRootSceneNode()->createChildSceneNode();
```


Step #5: 씬노드에 엔터티를 배치

```
daehyunNode->attachObject(daehyunEntity);
```



- 장면노드 *daehyunNode*에 엔터티 *daehyunEntity*를 배치

주요 멤버 함수

- void setVisible (bool visible)
 - 엔터티의 표시 여부 결정(true:보기 false:숨기기)
- bool isVisible (void)
 - 엔터티가 현재 표시 중인지를 확인.
- const String & getName (void)
 - 엔터티의 이름을 얻는다.
- SceneNode * getParentSceneNode (void)
 - 엔터티가 속해 있는 장면 노드를 얻는다.

■ 오우거 엔진의 렌더링 프로세스

□ 시스템 초기화

- Root 개체 생성
- 컨피규레이션 설정
- 렌더 윈도우 생성
- 리소스 위치 설정
- 프레임 리스너 생성

□ 씬 생성

- 씬 매니저 생성
- 카메라 및 뷰포트 생성
- 광원 생성
- 엔터티 생성
- 씬 노드 생성

□ 렌더링