

3D Visualization Engine

– 2008 졸업작품 최종 보고서 –

2008. 10. 09

성명	백민기
소속	전자통신컴퓨터공학부
학년	4
학번	2000005867
지도교수	박종일 (인)

전자전기컴퓨터공학부
한양대학교

목차

2	서론	4
3	본론	6
4	결과	26
5	참고문헌	27

그림목차

그림 1 Iterative Process.....	4
그림 2 TodEngine Architecture	6
그림 3 Object System의 Class Diagram.....	7
그림 4 Exception 관련 클래스	8
그림 5 TodEngine Resource 지정 방식 : URI(Universal Resource Identifier)	10
그림 6 Scene System의 Class Diagram.....	12
그림 7 Far Cry의 HDR Rendering 예제	13
그림 8 RenderPath를 이용한 HDR Rendering	14
그림 9 HDR Rendering Post Processing 과정	14
그림 10 TodEngine의 HDR Rendering	15
그림 11 T-Junction 문제를 해결하기 위한 Connector	16
그림 12 SLOD Terrain Rendering.....	16
그림 13 T-Junction 문제	17
그림 14 T-Junction Crack 해결을 위한 8개의 Connector	17
그림 15 Connector를 이용한 T-Junction Crack 문제 해결	18
그림 16 SLOD Terrain Rendering.....	18
그림 17 Gaussian을 이용한 2차원 Kernel.....	20
그림 18 Terrain Editor	20
그림 19 Node Explorer	21
그림 20 Node Explorer Popup Menu	22
그림 21 Property Grid	23
그림 22 Node Creator.....	24
그림 23 Command Console.....	25

1 서론

1.1. 연구목적

Software를 개발하는 것은 가장 복잡하고 어려운 일들 중에 하나라고 생각한다. 소규모의 Software를 개발하는 프로젝트도 종종 실패를 경험한다. 특히, Game Software개발은 창의적인 컨텐츠 제작이 핵심 개발 능력으로 요구되는데, 이는 잦은 기획의도 변경과 빠른 프로토타이핑이 가능해야 되기 때문이다. 하지만 이는 Software의 설계에 있어 문제를 가져오게 하여 결국 시스템의 Reliability 와 Maintainability, Performance 등의 주요 Software 측정 Matrix에 나쁜 영향을 준다.

졸업작품에서는 Tool-Oriented Development라는 개발 방법론을 소개하고 이 방법론의 결과물인 TodEngine 과 TodEditor를 개발하여 방법론의 성공여부를 알아보도록 할 것이다.

1.2. 기존연구

Waterfall Process에 대안인 Iterative 한 Project 진행 Process가 연구되어 있다. Iterative Process에서는 시스템을 모듈화 하여 각 모듈을 요구사항, 분석&설계, 개발, 테스팅을 한 주기로 삼아 반복적으로 개발해나가는 방법이다.

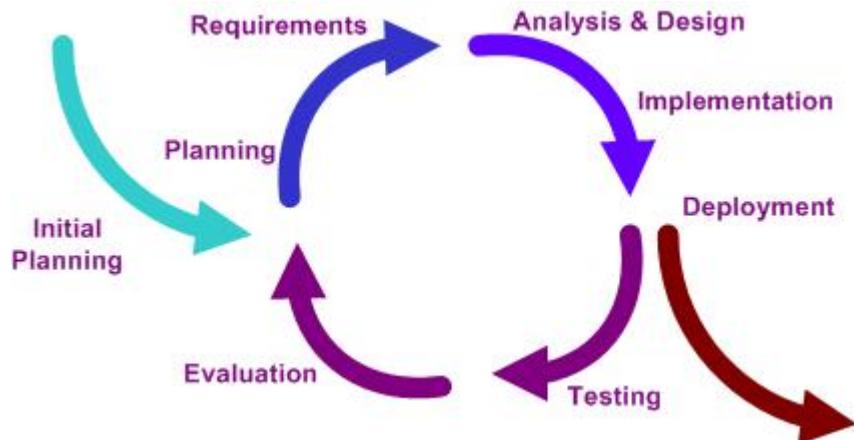


그림 1 Iterative Process

Software Prototyping는 미개척의 새로운 기술을 시험하거나 개발 가능 여부를 확인하고자 할 때 사용하는 방법이다. Game Software Development에서는 Prototyping을 사용하여 게임의 성공여부를 가려낼 수도 있다.

Test-Driven Development는 코드를 만들 때 사용하는 방법이다. 먼저 테스트케이스를 작성하고 그 테스트케이스를 통과하도록 코드를 작성하는 것이다. 이 방법은 프로그램의 결과를 보장해주고 Refactoring할 때 안전한 역할을 해준다.

1.3. 동기

평소 3D Engine, 특히 Game Engine/Framework 제작에 대한 관심이 많았다. 2D Engine 제작 경험은 꽤 되지만 3D Engine에 대한 기술적인 경험이 적었기에 이번 졸업작품을 통해서 개인적으로 3D Engine에서 구현해 보고픈 주제에 대해 공부하고 제작해 보고 싶어 이 주제를 선택하게 되었다. 또한 8년 남짓 게임회사에서 근무하면서 쌓은 노하우를 바탕으로 게임개발을 어떻게 하면 더욱 쉽고 빠르게 개발 할 수 있을까에 대해서 많은 고찰과 연구를 해왔는데, 그 결과가 Tool-Oriented Development이다. 이 컨셉을 바탕으로 개발된 실체가 TodEngine/TodEditor이다. 그리고 이러한 컨셉은 오랜 기간을 두고 개발을 진행해야 그 진가가 발휘되므로, 졸업작품에서는 이러한 TodEngine을 바탕으로 한 3D 기술을 구현할 것이다.

2 본론

2.1. Architecture

TodEngine의 Architecture는 크게 4부분으로 나뉘어 있다. Core에는 Engine의 가장 기본적인 기능이 Module 별로 나뉘어 있다. Core를 사용하여 Game Engine을 구성하는 부분이 Engine이다. Core에서는 Scripting Interface를 제공하는 Sub System이 있는데 이를 통해서 Script Language와 통신하게 된다. 그리고 Engine은 Blackbox Extending을 가능하게 하도록 Plug-ins을 만들 수 있는 다수의 Interface들이 존재한다. 또한 Core에서도 이러한 Plug-ins을 만들수 있는 기반을 제공한다. 이렇게 만들어진 기반에서 TodEditor가 실행된다. TodEditor는 Python과 wxPython 라이브러리를 사용하여 제작되었다. Scripting Interface를 통해서 TodEngine과 통신하게 된다.

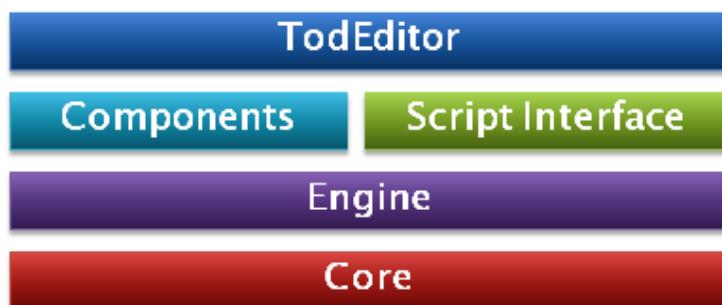


그림 2 TodEngine Architecture

2.2. Core

2.2.1 Object System

TodEngine의 많은 기능을 제공해 주는 Background를 만든다. Object System은 Modular Programming의 핵심을 이룬다. Object System은 Reflection Architecture의 기반이 된다. Kernel은 Engine의 Built-in Module과 Plug-in Module에서 제공되는 Object의 Meta Information을 저장한다. 그리고 Kernel에 요청을 하면 해당 Object의 instance가 반환된다.

TodEngine Reflection Architectural Pattern

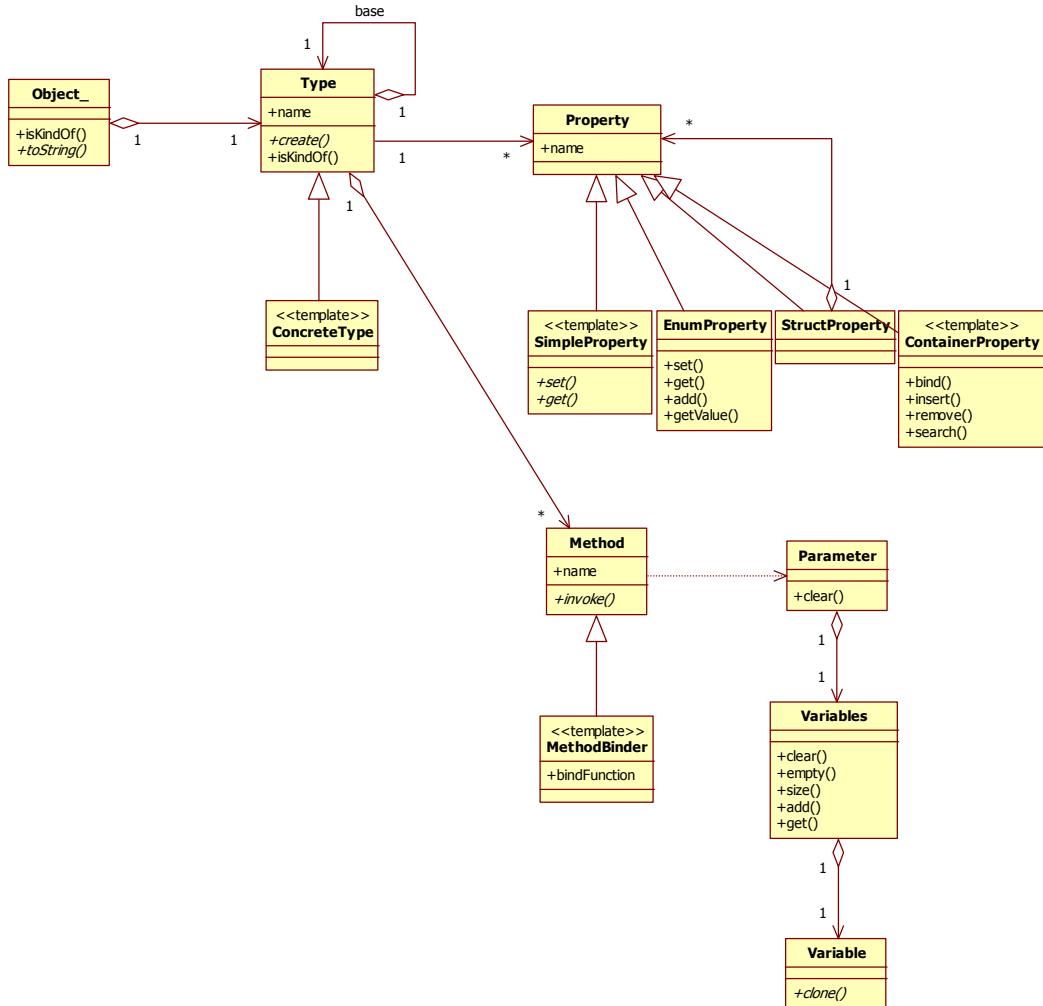


그림 3 Object System의 Class Diagram

Object의 Meta Information은 Property 와 Method로 구현된다. 이 클래스에 Object 의 Meta Information 이 저장된다. 또한 Method 는 Scripting Interface 와 관련된다.

2.2.2 Exception Policy

Engine에서는 상태에 따른 각종 에러처리를 명확히 해줘야만 한다. 그리고 그 에러나 예외가 발생하면 어떤 예외가 발생했는지 Engine 사용자에게 명확히 전달해야 한다. 그래야 디버깅이 빨라질 수 있다.

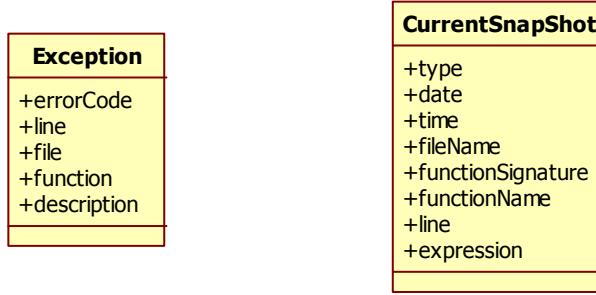


그림 4 Exception 관련 클래스

TodEngine에서 Exception이 발생하면 현재 상태를 CurrentSnapShot에 담는다. 그리고 이 객체를 throw하게 된다. 이러한 예외를 던지는 행위는 코드상에서 한 줄로 처리할 수 있게 되어 있다.

2.2.3 NOH(Named Object Hierarchy)

Object System과 함께 TodEngine의 가장 큰 특징 중의 하나는 NOH(Named Object Hierarchy)이다. 물론 이 개념은 Object System Based이며 System 전체에 녹아있다. 각 Object는 이름이 존재하며 이 이름을 기반으로 UNIX의 File System처럼 탐색이 가능하다. 이러한 특징은 Scripting Interface를 만드는데 주된 역할을 한다. C/C++언어의 특징인 pointer는 다른 언어에는 없지만 string은 기본언어기능으로 존재해야만 하기 때문에 NOH가 유용하게 쓰인다. 또한 NOH는 Engine에서 Object Management기능을 수행한다. Kernel은 Root Object를 hold 한다. Root Object 하위로 Object 탐색이 가능하다. 이러한 특징 때문에 Singleton Pattern의 특징을 고스란히 갖고 있다.

2.2.4 Reflection

Object System에서도 설명했지만 TodEngine의 Object System은 Reflection Architectural Pattern을 따르고 있다. 완벽한 것은 아니지만 성능과 기능상의 중간을 취하고 있다. JAVA의 Reflection처럼 Language에서 제공되지는 않지만 C++에서도 어느 정도의 Reflection 기능을 빠르게 지원할 수 있도록 하였다.

Property들은 set/get method의 함수 포인터를 binding하는 것으로 구현된다. 기본적인 형태뿐 아니라 struct, enum 등도 binding이 가능하도록 하였다. 그리고 container를 binding하는 것으로

Recursive 한 Property Binding 이 가능하다.

Method는 binding function을 binding 하는 것으로 구현된다. 하나의 Method는 하나의 binding function을 갖고 있다. 그리고 이러한 binding function은 객체마다 고유의 namespace 상에 존재하며 native command abstraction을 갖는다. 그렇기 때문에 어떠한 스크립트 언어로 확장되어도 동일한 형태로 binding 이 가능하다. TodEngine 에서는 기본적으로 Python Language 에 대한 확장이 구현되어있다.(Python Embedding)

2.2.5 Scripting Interface

Reflection 의 Method binding 기능으로 Native Command Abstraction가 존재한다. 이러한 Abstraction 은 다양한 언어의 특징을 취하여 추상화하여 만들어져 있다. 모든 procedure call 은 Command 이름, Input Argument, Return Value로 구성된다. 이러한 특징을 Native Command Abstraction에 녹여 고유한 Scripting Interface 가 구성된다. 그러므로 Lua, Ruby 등의 Script Language 로 확장되어도 문제없이 동일한 Interface 로 Command 호출이 가능하다.

2.2.6 Serializer

NOH 와 Reflection 조합의 또 다른 응용은 Serialization이다. NOH 는 XML은 homogeneous하다. NOH 상의 각 Object 는 XML 의 Element에 대응되고 Object 의 Property들은 Attribute에 대응된다. 그러므로 특정 Object를 Root로 하위의 Node들을 순회하며 각 Object의 Property를 XML Attribute로 만들어주면 하나의 XML문서가 만들어진다. 이러한 과정을 XML Serialization 이라고 한다. 반대로 XML문서로부터 NOH의 Node들을 reconstruction하는 과정을 XML Deserialization이라고 한다.

이 응용과 같이 Reflection Architecture는 객체의 저장/불러오기에 있어서도 강력한 성능을 발휘한다.

2.2.7 Resource System

Engine에서 처리할 일 중 하나는 Resource Management 이다. Software 내에서 사용되는 다양한 종류의 자원을 편리하게 지정하고 관리할 수 있어야 한다.

TodEngine 에서는 Resource System이 있어서 Resource 를 관리

하게 된다. 하나의 Resource 는 Package 단위로 관리하게 되는데 이러한 Package는 보안을 위해 하나의 파일로 archive된다. 또한 개발 상의 편의를 위해 기존의 Win32 파일 시스템을 그대로 사용하기도 하는데, 이 때문에 Resource 를 unique 하게 지정할 수 있는 interface가 필요하다.

TodEngine 의 Resource 지정은 URI(Universal Resource Identifier) 로 지정한다.

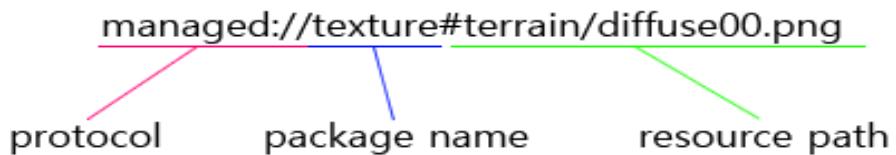


그림 5 TodEngine Resource 지정 방식 : URI(Universal Resource Identifier)

위와 같이 지정하면 Resource System으로 관리되는 Resource를 리턴받을 수 있다. 이러한 URI 는 Archive되었을 때와 그렇지 않을 때, 구분이 없다. Protocol은 http:// 또는 zip:// 등등이 될 수 있다.

2.2.8 Modular Programming

Core에서 지원되는 가장 큰 특징 하나는 Modular Programming이 가능하다는 것이다. Module은 크게 Built-in Module 과 Dynamic Module 이 있다. Built-in Module 은 Core 에서 기본적으로 제공하는 기능적 Class 들의 집합이고, Dynamic Module은 End-User가 제작한 Custom DLL/LIB 파일이다. Core는 이러한 Module을 On-Demand Loading하게 되고 이러한 모듈의 생성과 사용에 대한 Transparency가 보장된다. 그리고 이러한 Dynamic Module들이 확장 가능하도록 설계되어 있기 때문에 각 Module별로 개발이 가능한 기반이 제공된다.

2.3. Engine

2.3.1 Graphic System

Engine Layer에서 제공되는 기본적인 System 은 Graphic System 이다. 이 프로젝트는 기본적으로 3D Visualization을 표현하고 또한 여러 3D Technologies를 구현해야 하기 때문에 이 System은 중요하다. Graphic System은 Renderer와 그 구성요소에 대한 Abstract Interface만을 제공한다. 이에 대한 Implementation은 Dynamic

Module로 제공될 것이다. TodEngine에서는 기본적으로 *d3d9graphics* Dynamic Module을 제공한다. *d3d9graphics* 는 DirectX 9.0c API 를 사용하여 Renderer를 구현한 모듈이다. 이와 같이 Graphic System의 interface를 Module구현으로 제공할 수 있으며 이는 Whitebox extending 을 제공한다.

Graphic System 은 FFP(Fixed Function Pipeline) 대신 Shader Centric한 설계를 하였다. 모든 Object는 Shader기반 Rendering을 한다.

2.3.2 Scene System

Scene System은 TodEngine의 Graphic System을 사용하고, 전체 Rendering System의 중심이 된다. 화면에 Rendering되는 Scene은 다수의 SceneNode의 조합으로 이루어지게 된다. 이는 Scene Graph의 동작과 매우 흡사하다.

SceneNode는 최상위 클래스이고, 이 클래스를 상속받은 클래스는 Scene Graph에 의해서 관리된다. 이렇게 SceneNode를 상속받는 클래스에는 Object의 Rendering에 필요한 정보를 차례로 추가하게 되는데, 이는 계속되는 상속에 의해서 처리된다. SceneNode – TransformNode – AbstractShaderNode – ShaderNode – MeshNode 의 클래스 계층구조는 하나의 Mesh를 Rendering 하기 위한 모든 정보를 hold한다. 이렇게 해서 얻어진 정보를 통해 Renderer 는 해당 Object를 Rendering 한다.

위와 같은 상속구조는 SceneNode의 확장을 할 수 있도록 한다. 예를 들면, Shader를 사용하고 Transform정보를 가지고 있는 Solid – Leaf BSP(Binary Space Partitioning) Tree를 만들경우 ShaderNode 를 상속하면 이러한 새로운 Node를 만들어 낼 수 있다.

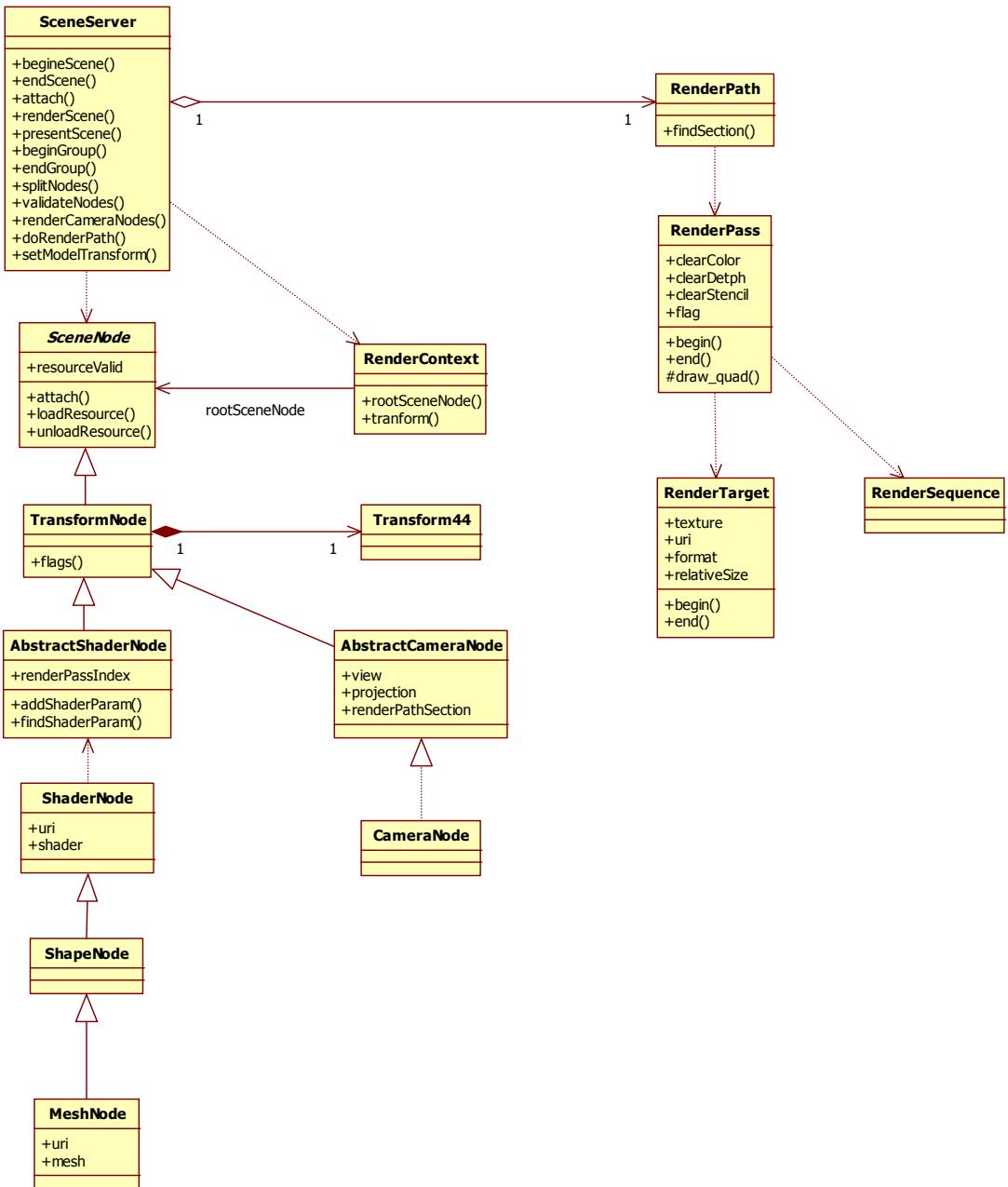


그림 6 Scene System의 Class Diagram

2.3.3 Render Path

Render Path 는 Scene 이 Rendering 되는 과정을 제어해주는 Sub System 이다. Render Path를 사용하면 Post Effect Processing 을 매우 쉽게 제어할 수 있다. Render Path 에는 여러 개의 Render Target 을 사용할 수 있으며, 이 Render Target을 Shader를 사용하여 조작하여 최종 Scene을 Composition할 수 있게 한다.

2.4. 3D Technologies

2.4.1 HDR(High Dynamic Range) Rendering

Computer에서 색을 표현하는 기본적인 방법 중 RGBA 표현방법이 있다. 이는 각 색 channel을 0~255사이의 값으로 표현하고 각 channel이 모두 255라면 흰색, 모두 0이라면 검은색을 표현한다. 이는 LDR(Low Dynamic Range) Rendering으로 이전의 Computer Graphics에서 쓰이는 방법이다.

하지만 사람이 실제로 환경을 볼 때는 0~255사이의 값을 갖지 않고 그 보다 더 큰 에너지를 받아 들일 수도 있기 때문에 빛에 대해서 over exposure될 수 있다. 이러한 표현을 가능하게 하는 Rendering 기술이 HDR(High Dynamic Range) 이다.



그림 7 Far Cry의 HDR Rendering 예제

위 그림은 게임 Far Cry에서의 HDR Rendering을 보여준다. 오른쪽 그림을 보면 빛이 다른 surface에 영향을 주지 않는다. 하지만 왼쪽 그림은 여러 Post Effect Processing 처리를 통해 보다 자연스럽고 현실적인 빛 처리를 보여주고 있다.

TodEngine에서는 Render Path subsystem 을 사용하여 이러한 처리를 구현하였다. Render Path를 사용하여 Post Processing을 거쳐 최종 scene 을 rendering 하는 개략적인 과정은 다음과 같다.

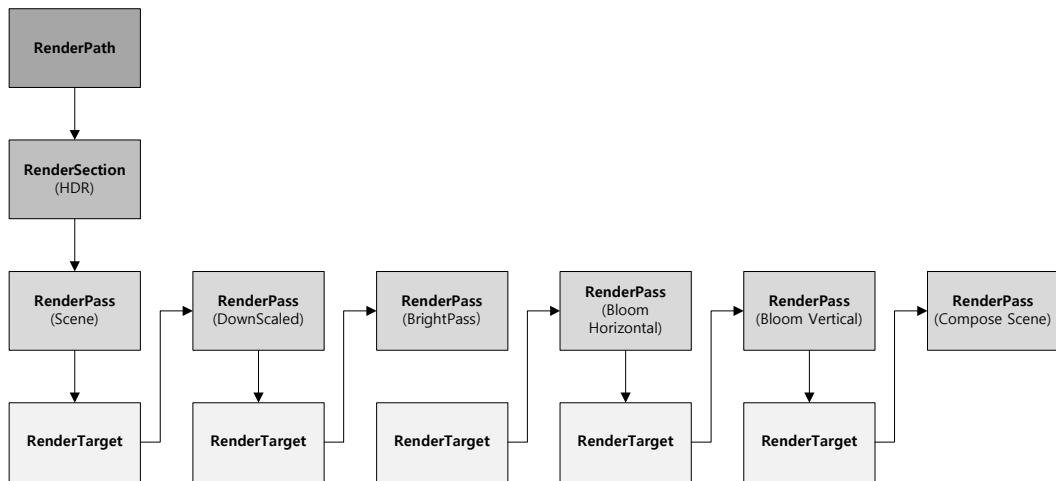


그림 8 RenderPath를 이용한 HDR Rendering

다음은 각 RenderPass에서 RenderTarget에 그려지는 HDR Rendering Post Processing 처리 모습을 나타내었다.



그림 9 HDR Rendering Post Processing 과정

Original Scene을 Down Scaled하면 renderer 자체 내에서 축소될 때 사용되는 filter를 이용하여 texel들이 뭉개진다. 이를 Bright Pass Filter를 거치면 Scene내의 밝은 부분만 pass되어 Render Target에 저장된다. 이를 Gaussian Blur(Gaussian Filter) Horizontal/Vertical과 함께 Bloom Scaled값을 각 texel에 곱하면 Bloomed Scene이 만들어진다. 이를 최종적으로 Original Scene과 Compose하면 Final

Scene이 생성된다.

TodEngine에서는 기본적으로 HDR Rendering을 수행하도록 되어 있다.

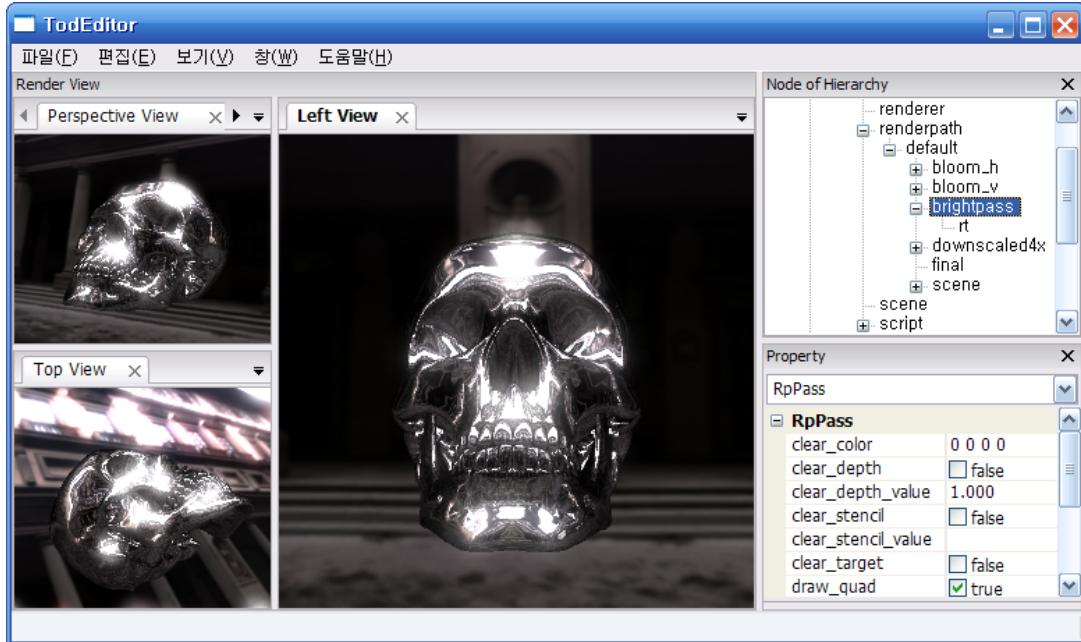


그림 10 TodEngine의 HDR Rendering

2.4.2 Terrain

2.4.2.1 SLOD(Static Level of Detail)

Continuous Level of Detail(CLOD) 관련 terrain LOD는 Video card로 전송되는 vertex 개수를 최적의 상태로 만들어주긴 하지만 CPU usage 가 많기 때문에, 잘 사용되지 않는다. 왜냐하면 GPU 의 발전으로 이러한 알고리즘을 사용하여 vertex 를 최적으로 만들어주기보다 GPU를 최대한 활용하여 CPU 자원을 다른 처리를 위해 양보하는 것이 좋기 때문이다. 그래서 TOD Engine 에서는 요즘 추세에 맞추어 SLOD 를 구현하였다. 다음에 기회가 된다면 CLOD 알고리즘 중 하나인 ROAM(*Real-time Optimally-Adapting Meshes*) 를 이용한 terrain rendering 에 도전해보고 싶다.

TodEngine에서 구현한 SLOD 알고리즘은 Chunked LOD^[2]의 변형이다. 하나의 height map으로부터 만들어진 Vertex Buffer 를 만든다., 이 Vertex Buffer를 일정한 간격으로 나눈 (보통 33x33) tile 로 나눈다. 이러한 tile 별로 각 LOD 별

Index Buffer 를 만들고, 렌더링 할때 이를 이용하여 LOD 를 구현한다. 이때, T-Junction 문제가 발생하는데, 이는 tile 옆의 LOD 를 참조하여 적절하게 이어주는 connector 용 Index Buffer 를 사용한다.

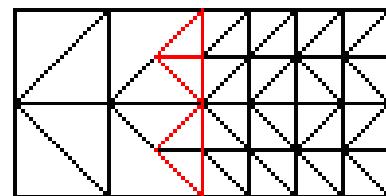


그림 11 T-Junction 문제를 해결하기 위한 Connector

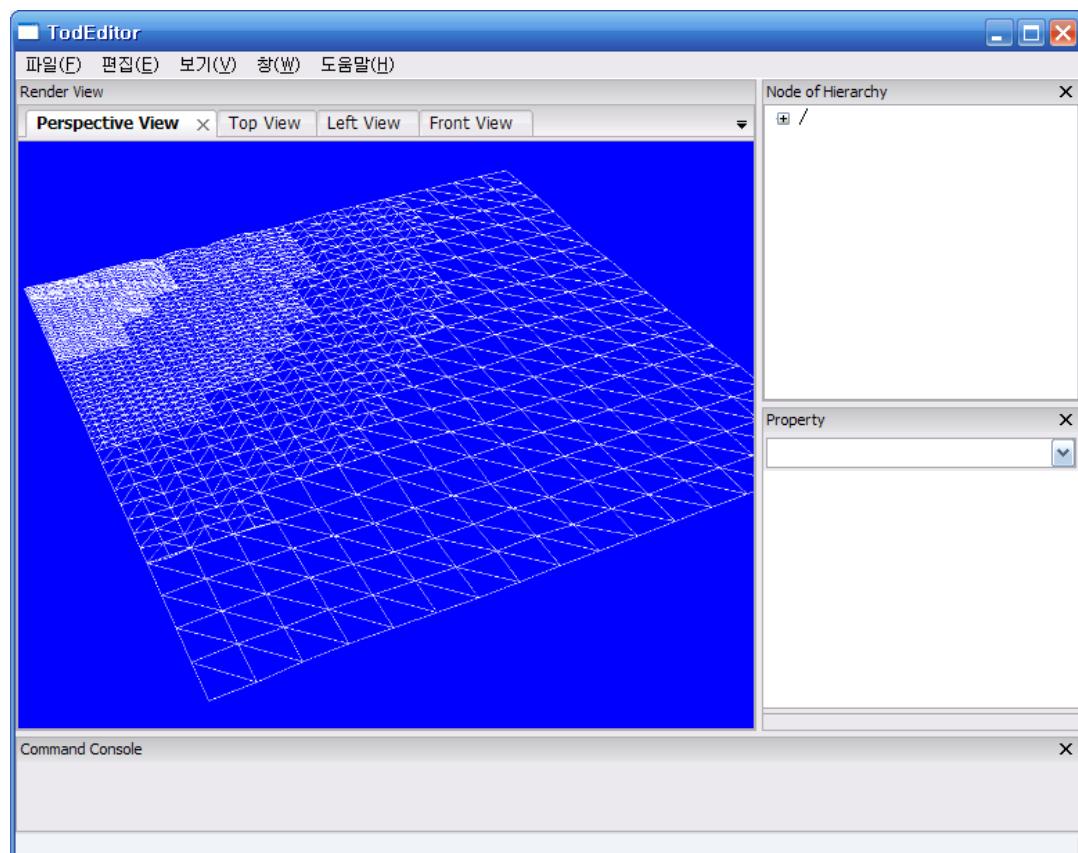


그림 12 SLOD Terrain Rendering

SLOD에서는 detail level 차이가 나는 타일과 타일 사이의 T-Junction에서 발생하는 crack문제를 해결해야만 한다.

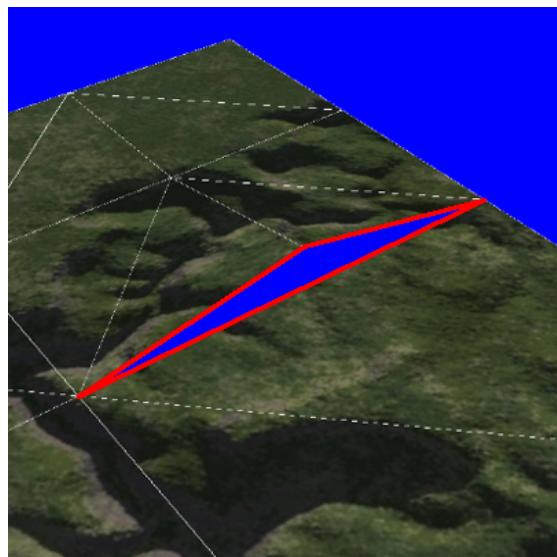


그림 13 T-Junction 문제

위 그림은 레벨차이가 나는 타일과 타일 사이에 생긴 crack의 전형적인 모습을 보여준다. Crack을 해결하기 위해서는 다음과 같은 레벨차이를 이어주는 connector용 타일이 8개 필요하다. 초기의 SLOD에서는 connector용 타일이 16개가 제작되어야 하는데, 이는 레벨이 큰 쪽에서 작은 쪽으로 메워주었기 때문이지만, 레벨이 낮은 쪽에서 큰 쪽으로 메워주면 다음과 같은 connector 8개만 있으면 된다.

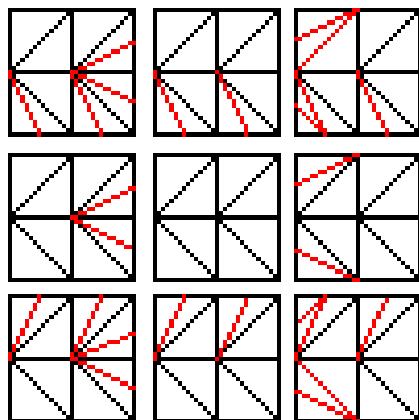


그림 14 T-Junction Crack 해결을 위한 8개의 Connector

이러한 connector가 마련되면 top/left/right/bottom/top-left/top-right/bottom-left/bottom-right를 참조하여 상황에 맞는 적절한 connector를 사용하여 렌더링 해주면 된다.

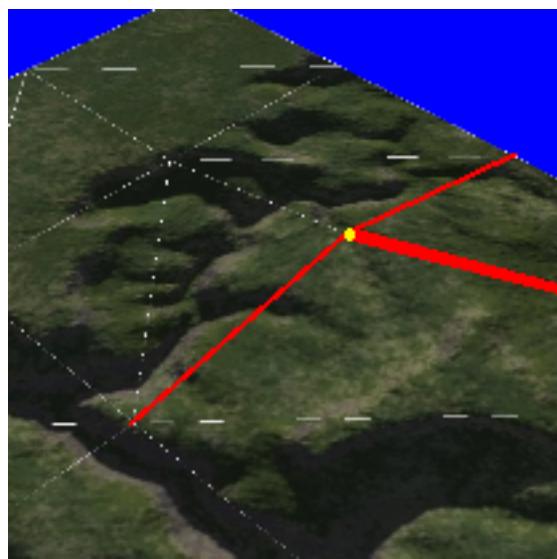


그림 15 Connector를 이용한 T-Junction Crack 문제 해결

위 그림은 Connector를 이용하여 crack을 해결한 Terrain을 렌더링한 모습을 나타낸 것이다.

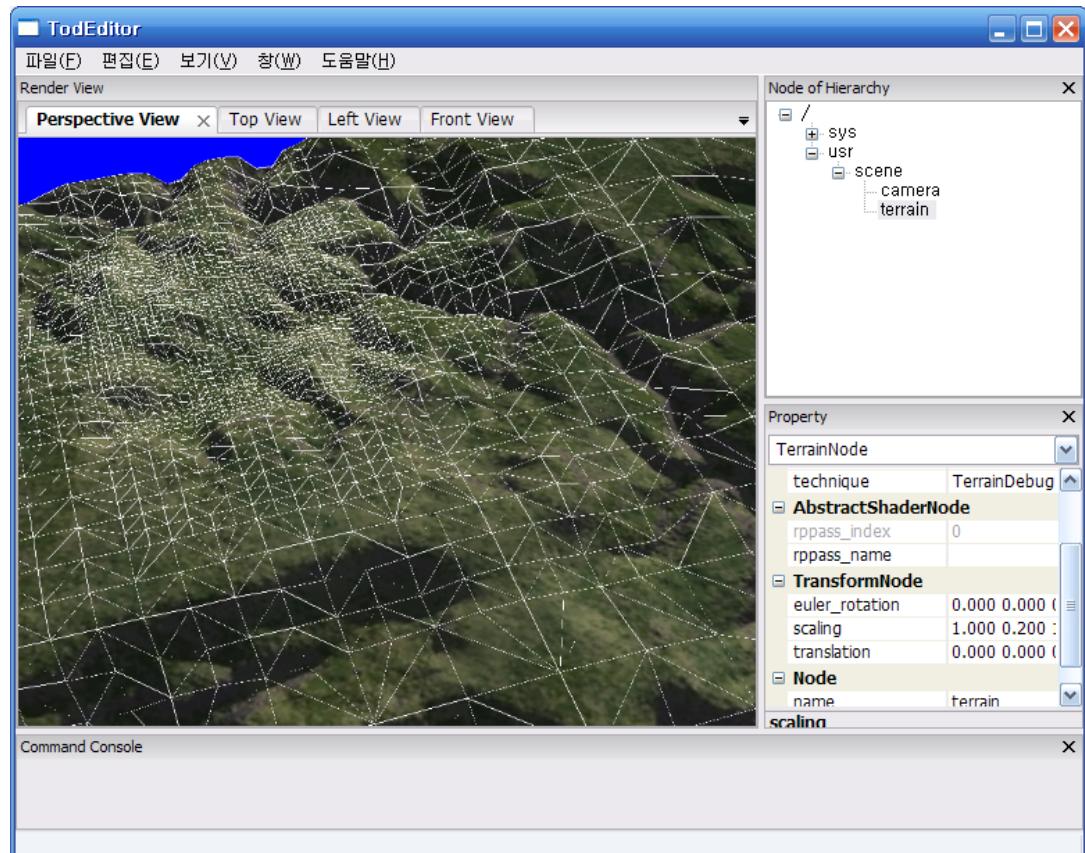


그림 16 SLOD Terrain Rendering

2.4.2.2 Terrain Editor

Terrain Editor는 SLOD Terrain Rendering을 이용하여 Terrain 을 실시간으로 편집할 수 있도록 하는 TodEditor의 Plug-in Editor이다. Terrain Editor 를 구현하기 위해서 Object Picking 기술과 Gaussian Kernel을 사용하였다.

2.4.2.2.1 Object Picking

Object Picking이란 마우스 등으로 화면의 특정 위치를 지정하면 그 위치에 해당하는 World Space에 있는 객체를 선택할 수 있는 기술이다. World Transform, View Transform, Projection Transform 변환을 거쳐 화면에 그려진 객체를 picking하는 것은 이 Transform들을 역순으로 거쳐 올라갈 필요가 있다. 하지만 Projection Transform 하는 순간 w로 값들이 나눠지기 때문에, picking 할 때 위치를 지정하면 Look at vector 방향의 Line이 생긴다.(Ray) 이 Ray를 따라 객체를 검색하다가 Eye Point와의 Distance를 비교하여 가장 작은 객체가 picking이 된 객체가 된다. TodEngine에서 구현된 Picking 알고리즘은 폴리곤 단위까지 검출이 가능 하다.

2.4.2.2.2 Gaussian Kernel

Object Picking된 Terrain의 한 지점을 중심으로 주변의 Vertex들의 y값을 올리고 내리고 하면 Terrain 편집이 된다. 이때 필요한 것이 Gaussian Kernel 이다. Terrain 편집에 쓰일 Gaussian Kernel은 2차원이다.

$$2D(\quad ; \quad) = \frac{1}{2} e^{-\frac{x^2+y^2}{2}}$$

이 공식에서 를 높이면 값들이 전체적으로 sharpness해지고 낮추면 softness 해진다. 이렇게 수치조작을 통해 자유롭게 Gaussian Kernel을 만들 수 있기 때문에 Terrain Editor에서는 이를 반영하기 위한 GUI가 구성되어 있다.

위 공식을 이용하여 중심점을 기준으로 Gaussian Kernel 을 구하면 된다. 이렇게 구해진 Gaussian Kernel은 2차원 array로 구성된다. 이렇게 구성된 array를 유효범위의

Vertex 의 y값에 더해주면 지형을 올릴 수 있고, 빼주면 지형을 내릴 수 있다.

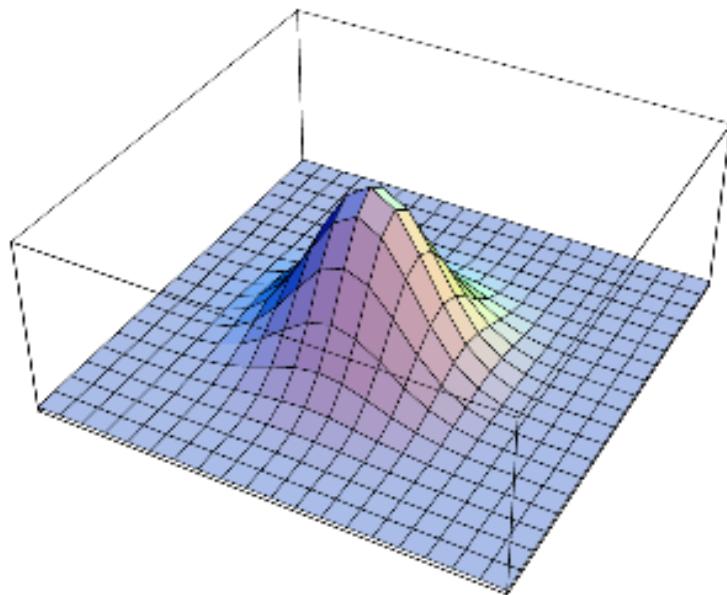


그림 17 Gaussian을 이용한 2차원 Kernel

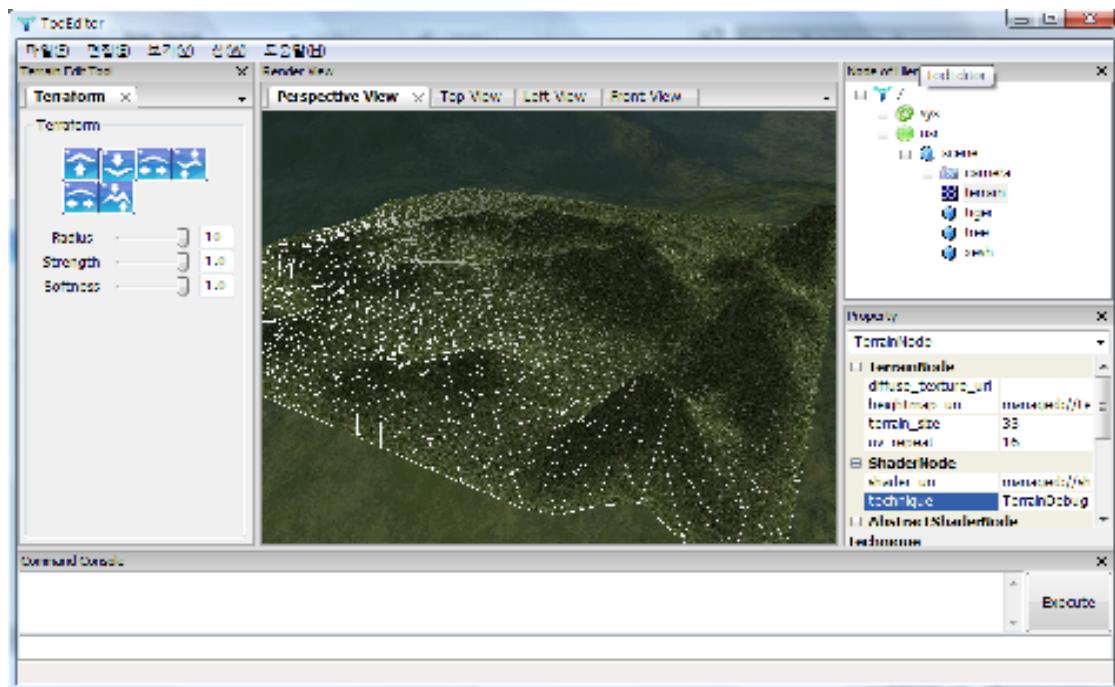


그림 18 Terrain Editor

2.5. TodEditor

2.5.1 Introduction

TodEditor는 TodEngine기반으로 만들어진 In-Game Editor이다. TodEngine과 TodEditor는 공통의 Core와 Engine code를 공유하기 때문에 TodEngine의 기능이 발전하면 TodEditor의 기능이 발전한다. 또한 그 반대로 가능하다.

TodEditor의 모든 코드는 Python Language로 작성되어있고 Python의 풍부한 라이브러리를 그대로 사용할 수 있다. 그렇기 때문에 TodEditor의 GUI는 wxPython toolkit을 사용해서 만들어졌다. wxPython은 강력한 GUI toolkit인 wxWidgets의 Python 버전이다.

TodEditor는 여러 에디터가 모여 전체를 이루며, 그 기반이 되는 에디터는 Node Explorer와 Property Grid이다. 에디터를 쉽게 확장할 수 있도록 하기 위해 각 Node별로 Plug-in Editor를 만들 수 있으며 Node Explorer에서 곧바로 실행이 가능하다.

2.5.2 Node Explorer

Node Explorer는 TodEditor의 기본적인 에디터 중 하나이다.

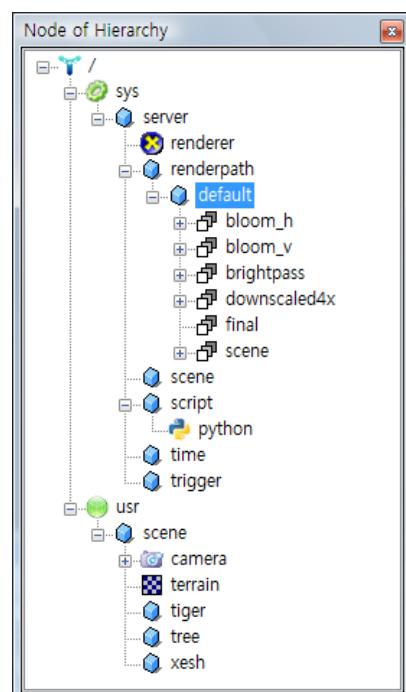


그림 19 Node Explorer

Node Explorer는 Kernel에서 생성되어있는 NOH의 모든 객체를

동적으로 탐색해 볼 수 있는 에디터이다. 이 에디터를 시작점으로 하여 Editor의 모든 기능이 동작된다. 화면에 보여지는 Node는 실시간으로 생성/삭제/편집이 가능하다.

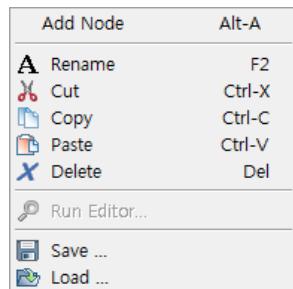


그림 20 Node Explorer Popup Menu

Node Explorer 상에 보이는 Node에 마우스 오른쪽 클릭을 하면 Popup Menu가 뜬다. 이 메뉴에서 해당 Node에서 동작 가능한 모든 명령이 표시된다. 하위에 Node를 생성하거나, 이름을 바꾸거나 복사, 붙이기 등이 가능하다. 이 Node를 기점으로 Serialization을 하려면 Save를 누르고, Deserialization 하여 XML 파일로부터 하위 Node를 reconstruction 하고 싶다면 Load를 클릭한다. 또 하나의 중요한 특징은 이 Node에 특수한 Plug-in Editor가 존재하면 Run Editor...를 선택함으로써 에디터를 실행 할 수 있다. Terrain Editor는 이 기능을 활용하여 확장된 Editor이다.

2.5.3 Property Grid

Property Grid는 Node Explorer에서 선택된 객체의 속성을 편집할 수 있는 기능을 가진 Editor이며 TodEditor의 기본 에디터 중 하나이다.

Property Grid 는 Reflection 기능을 이용해 선택된 객체의 Property 리스트를 받는다. 그리고 그 리스트를 Editor의 화면에 일괄적으로 표시하고, 사용자로부터 편집기를 통해 입력을 받으면 그 변경된 값을 객체의 새로운 Property 값으로 설정한다.

TodEngine의 Object System에는 Reflection Architecture가 적용되어 있고 C++기본으로는 지원해 주지 않는 향상된 RTTI(Run-time Type Information)이 탑재되어 있다. Property Grid 는 이 RTTI를 바탕으로 class 상속구조를 에디터상에 반영한다. 다음 그림에는 이러한 상속구조를 반영한 Property Grid의 모습을 보여준다.

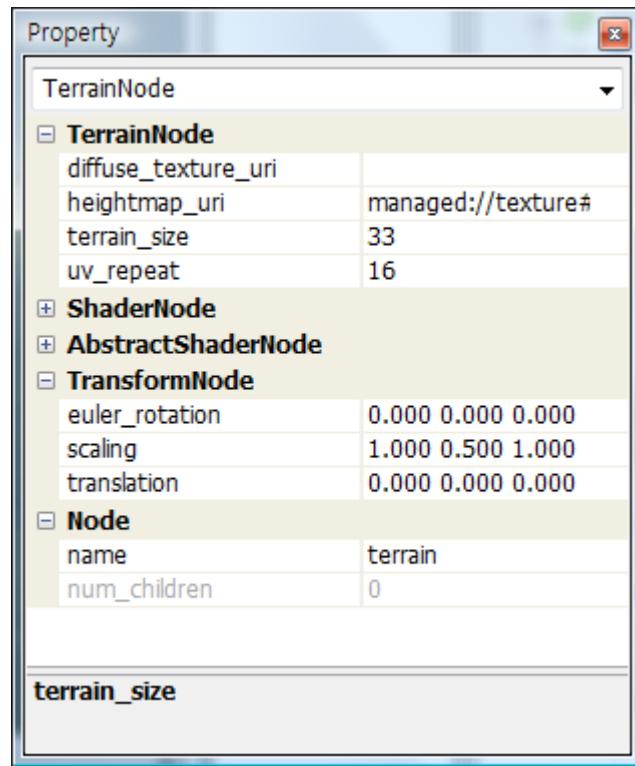


그림 21 Property Grid

위 그림처럼 TerrainNode 는 ShaderNode를 상속받았고 ShaderNode 는 AbstractShaderNode—TransformNode—Node 에 이르는 class 상속 구조를 보여준다. 그리고 각 클래스에 속한 Property들을 표현하고 있다. Property 이름 옆의 값들은 Property의 현재 설정된 값을 표현하고, 이 값을 사용자가 임의로 변경할 수 있다. 이렇게 변경이 되면 객체는 Property 값이 바뀌고 바뀐 내용이 Scene에 관련이 있다면 Scene View에 즉시 반영된다.

o) Property Grid는 Node Explorer와 함께 Tool Editor의 큰 축을 이루는 Editor이며, 사용자와 가장 많은 interaction이 일어나는 에디터 중 하나가 된다.

2.5.4 Node Creator

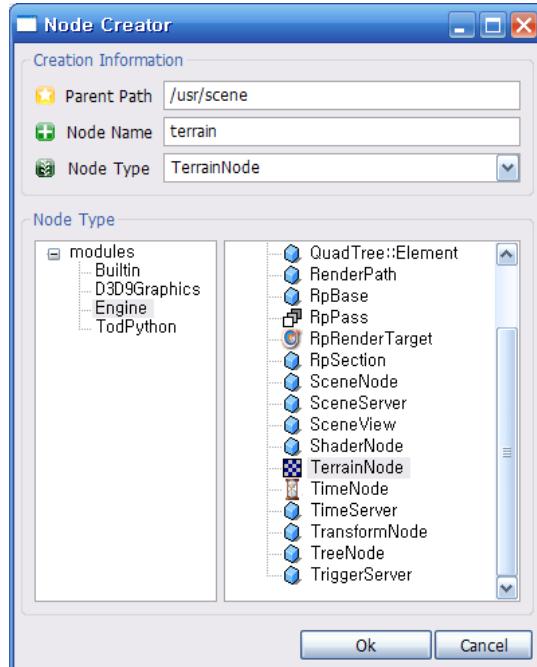


그림 22 Node Creator

Node Creator는 Node Explorer에서 Add Node를 선택했을 때 나오는 대화상자이다. 이 대화상자에서는 새로운 Node를 만드는데 필요한 기본적인 interface를 제공한다. 새로운 Node의 타입을 고르고 이름을 입력하면 새로운 Node를 만드는데 필요한 모든 정보가 수집된다.

Node Creator에서 Node Type 부분은 Modular Programming⁹ 가능한 TodEngine의 기능을 보여주고 있다. 각 Module¹⁰이 표시되고 이 Module에서 제공되는 객체 타입이 오른쪽에 나타난다. TodEngine을 사용하는 프로그래머는 이렇게 Module을 만들면 Node Creator에서는 이를 자동적으로 인식해서 표시해준다. TodEditor 사용자는 이렇게 만들어진 객체를 생성할 수 있게 된다.

2.5.5 Pluggable Editor

Pluggable Editor는 Node Explorer에서 객체를 선택한 뒤 Run Editor...를 선택하면 특수한 Editor가 뜨게 되는데 이러한 기능을 Pluggable Editor라고 한다. 이는 새롭게 만들어지는 객체 타입에 대한 고유한 Editor를 Rapid하게 개발할 수 있는 방법을 제공한다. 이렇게 만들어지는 Editor는 TodEditor를 사용하는 중간에 실시간으로 만들어 질 수 있기 때문에 Editor의 빠른 개발이 가능하다.

2.5.6 Scene View

Scene View는 Graphic System과 Scene System에 의해서 Rendering 된 결과물이 표시되는 곳이다. Scene View는 여러 인스턴스가 생성될 수 있으며, 각기 다른 Camera Object를 지정할 수 있다. 이렇게 서로 다른 위치에서 같은 Scene을 Rendering할 수 있는 기능은 여러 장점들을 제공한다. 예를 들면 실제 연출을 위한 Camera와 에디터를 위한 Camera가 동시에 설정될 수 있기 때문에 편집이 용이해 진다.

Scene View는 NOH 상의 /usr/scene 하위의 모든 SceneNode들이 Scene System에 의해 처리되어 화면에 표시된다. 그렇기 때문에 Property Grid로 특정 객체의 Property가 변경되면 그 결과는 즉시 Scene View에 반영된다. 이러한 특징을 이용하여 interactive한 편집이 가능하다.

2.5.7 Command Console

TodEngine 은 Architecture상 Scripting Interface가 존재한다. 그렇기 때문에 Script를 통해서 객체 생성/삭제/변경이 가능하다. Python 으로 TodEditor 전체를 만들어진 것이 그 결과라고 볼 수 있다. Command Console은 사용자에서 Scripting Interface를 직접 노출하는 에디터이다. Command Console에서 Python Language 명령을 사용하면 그 명령은 Python Interpreter에 의해 해석되고 TodEngine과 적절한 통신을 수행하게 된다.

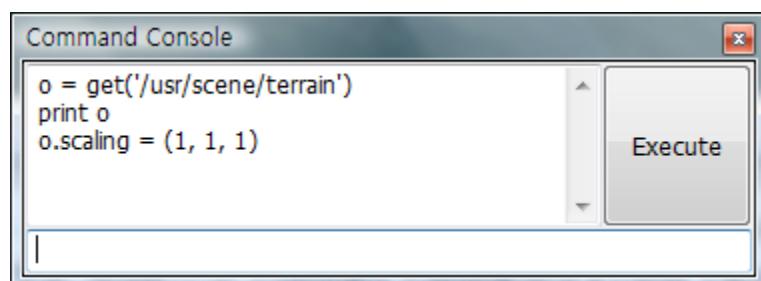


그림 23 Command Console

3 결과

이번 졸업 작품을 진행하면서 처음에는 의욕적으로 4개의 3D Technologies를 구현하려 하였다. 결과적으로 2개의 3D Technologies를 구현하는 것으로 마무리 짓는 것은 매우 아쉬운 일이긴 하다. 하지만 3D Technologies 이외에 3D Engine이라면 갖추어야 할 대부분의 기반 시스템을 구현하였다는 것이 아쉬움을 달래 주었다.

Tool-Oriented Development는 이름만 틀릴 뿐 이미 Commercial Game Engine 시장에서 대세로 자리잡고 있다고 생각한다. 최고의 게임 엔진 중 하나인 Unreal Engine에도 TodEngine과 유사한 방법의 게임 제작 패러다임을 제시하고 있다. 이러한 면에서 TodEngine의 구현은 개인적으로 이러한 Game Engine에 대한 기술 축적으로서도 상당한 가치를 지니고 있다고 생각한다. 또한 지난 8년간 게임 제작에 대한 노하우의 결정판이라고 생각한다. 앞으로 개선할 사항은 많지만 그러한 것은 Detail에 지나지 않는다. 중요한 것은 개발 Process와 생산성 향상을 위한 Concept 등이다. 특히 Software 공학의 영원한 난제인 생산성 향상 부분을 Tool-Oriented Development에서 어느 정도 풀어냈다고 생각한다. Tool을 빨리 만드는 방법으로 이러한 생산성 위기를 극복할 수 있다고 생각한다.

Smart programmer are Tool programmer
- Tom Forsyth

4 참고문헌

Books

- wxWidget Documentation
- Python Documentation
- ShaderX 3, 4, 5
- Real-time Terrain Rendering with C++
- 3D Game Programming
- Game Programming Gems 1, 4, 5
- Game Institute – Game Programming with DirectX Module I, II
- Thatcher Ulrich – Rendering Massive Terrains using Chunked Level of Detail Control

Web

- Iterative Process
http://en.wikipedia.org/wiki/Iterative_and_incremental_development
- Gaussian Kernel
<http://www.stat.wisc.edu/~mchung/teaching/MIA/reading/diffusion.gaussian.kernel.pdf.pdf>

Open Source Engine

- The Nebula Device 2
- Ogre (Object Oriented Game Rendering Engine)