# 3D 애니메이션 파일의 렌더링 레이어를 조절하기 위한 독립적인 렌더링 도구 개발

## Developing an Independent Rendering Tool to Control Render Layer in 3D Animation Files

**김기홍, 최철영, 채일진**
동서대학교 디지털콘텐츠학부

Ki-Hong Kim(khkim@gdsu.dongseo.ac.kr), Chul-Young Choi(freechoi@gdsu.dongseo.ac.kr)
Eel-Jin Chae(cinetree@gdsu.dongseo.ac.kr)

**요약**

2005년부터 장편 3D 애니메이션 제작의 세계적인 증가추세와 하드웨어와 소프트웨어의 기술적 진보가 시각적 표현의 한계치를 넘어설 정도가 되면서 전체 작업 공정 중에서 렌더링(rendering)이 차지하는 비중이 더욱 중요하게 되었다. 렌더링 직전의 최종 파일은 데이터의 크기가 증가하는 문제가 발생한다. 이것은 수정 작업 시에 대용량 파일을 불러오고 저장하는 불필요한 소요시간과, 수정할 부분 외에 전체적으로 다시 렌더링을 수행하게 되는 시간을 소모하게 만든다.

본 논문에서는 연구 방법으로 Maya API를 활용하여 렌더링 레이어들의 정보 수정 방법을 연구하고 이를 적용하여 렌더링을 실행하는 Tool을 구현하는 것이다.

연구결과, 상업적인 애니메이션 제작에서는 제작 일정의 준수는 매우 중요하다. 그러므로 수정 작업에서 효율을 극대화 할 수 있기 위하여 독립적인 렌더링 툴 기반의 레이어 렌더링 구현이 필요하다고 판단된다.

■ 중심어 : | 3D 애니메이션 | 렌더링 | 대용량 | 레이어 |

**Abstract**

Since 2005, the production of feature 3D animation has been growing up worldwide and technology of hardware and software has been advanced fully enough to get over the limitation of visual expression, so that rendering weight of the whole work process is getting more important. A final file in the state right before rendering increases. the data size gets bigger. Therefore, if such final file is rendered as it is, the production schedule may not be met. For, some needless time shall be spent to call and save a high-capacity file in amendment work and to execute the rendering entirely again in addition to amendment.

In this study, Maya API [1]was used to study the efficient information amendment method of extracting the rendering layers' files in order to develop a tool that executes rendering by applying such method.

As the result, compliance to production schedule that is important in animation production can be done through the independent rendering tool based layer rendering that can maximize the efficiency of amendment work.

■ keyword : | 3D Animation | Rendering | High-capacity | Layer |

* 본 연구는 지식경제부 및 정보통신연구진흥원의 해외교수요원초빙사업의 연구결과로 수행되었음. 과제관리번호: C1012-0801-0008
접수번호 : #080507-002
접수일자 : 2008년 05월 07일
심사완료일 : 2008년 07월 07일
교신저자 : 채일진, e-mail : cinetree@gdsu.dongseo.ac.kr

# I. 서 론

## 1. 연구배경과 목적

세계 애니메이션 산업은 디지털 기술의 발달로 인하여 성장을 거듭하고 있는데 그 중에서도 3D 애니메이션 산업분야의 급성장은 주목할 만하다. 최근 블록버스터 급 할리우드 애니메이션 제작이 줄을 있고 있는데 2005년에는 극장용 애니메이션 19편 중 12편이 3D 애니메이션으로 제작되었다[1)]

3D 애니메이션 산업의 성장세는 지속될 것으로 예상되며 3D 그래픽을 사용하는 게임 산업의 발전으로 그 규모는 더욱 성장할 것으로 보인다. IT 기술의 발전으로 컴퓨터 하드웨어의 성능이 좋아짐에도 불구하고 항상 다루어야 하는 작업 파일 크기의 양적 증가는 이를 압도하고 있다. 2~3년 전만 해도 하나의 Maya파일 크기가 100 MB(메가바이트, 이하 생략)이상이 되는 경우가 드물었던 것에 비해 최근 대다수가 100 MB 이상으로 파일자체의 오퍼레이션(Open, Save, Copy 등)에 소요되는 시간이 늘어났다. 이는 제작 일정이 정해져있는 상업적 애니메이션 제작에 있어서 큰 문제점으로 부각되었다[1][2]. 이를 해결하기 위한 연구도 진행되어 최근에는 다수의 대용량의 파일들을 본 3D 프로그램 없이도 렌더링 정보를 추출하고 수정하는 연구를 진행하여 애니메이션 작업 공정에 효율성을 많이 높일 수 있었다. 그러나 최근의 3D 애니메이션 작업 경향은 VFX(Visual Effects)를 위한 파티클 효과나 다이내믹 시뮬레이션의 사용이 증가함으로 제작과정이 복잡해지면서 수정작업이 증가함으로 레이어 렌더링의 필요성이 증가하고 있다[3]. 즉 파티클 효과의 수정 작업이 있을 때, 작업이 필요 없는 캐릭터나 배경마저 다시 렌더링을 한다면 중복 소모가 된다는 것이다. 잘못된 파티클 부분만 수정하고 파티클 레이어만 다시 렌더링을 하면 효율적인 수정 작업을 진행할 수 있다. 그러나 이러한 레이어 렌더링도 3D 프로그램(Maya)에서 작업을 하기에는 파일자체의 운용에 드는 시간이 부담이 된다. 그것을 피하기 위해 레이어 렌더링 작업도 독립적인 프로그램 안에서 레이어의 정보 추출과 수정 후 적용이 이루어져야 애니메이션 작업 공정 안에서 최대한의 효율을 높일 수 있다[4].

1)2006.8.세계 애니메이션 산업 동향(한국문화콘텐츠진흥원) 4P

## 2. 연구방법 및 범위

이를 위한 첫 번째 연구방법은 3D 파일에서 렌더 레이어 정보를 추출하는 것이고 둘째는 추출된 정보를 가공하여 네트워크 렌더링을 수행 하는 것이다. 본 연구 수행에서 사용되는 3D 파일의 포맷은 Maya를 사용하고 렌더링의 분산처리는 Alfred(RenderMan을 위한 분산처리 프로그램)를 활용 하는 것이므로 Alfred를 수행하기 위한 스크립트 파일을 생성하고 이를 Alfred가 수행하도록 하는 것이다.

첫 번째 렌더링 레이어 정보의 추출에는 Maya 파일 포맷을 활용 할 것 이므로 C++ 기반의 Maya API[1)]를 이용하여 파일의 내부 값을 추출 할 것이고 두 번째는 C#기반으로 추출된 정보로 분산 처리를 위한 Alfred Script를 만들어내 대용량 다수 파일의 렌더링을 실행하는 것으로 한다.

# II. 이론적 배경

## 1. Render Layer의 개요

일반적으로 TV나 영화에서 볼 수 있는 애니메이션들은 하나 이상의 Layer들의 구성으로 이루어져 있다. 보통은 캐릭터와 배경 등으로 나누어서 렌더링을 각각 실행 한 후 합성으로 각각의 Layer들을 하나의 이미지로 구성을 한다. 한 번에 렌더링을 한다면 합성 할 수 있는 시간을 절약 할 수 있지 않는가라는 생각을 할 수 있는데 일반적으로 완성된 이미지를 이루기 위해 여러 번의 수정을 하게 된다. 이때 각각 나누어진 Layer들을 수정하는 것이 크기도 가볍고 시간적으로 소모가 적기 때문에 Render Layer는 일반적인 방식이다[5].

Layer에서 렌더링 하는 경우, 사용자는 오브젝트들을 개별적으로 렌더링하게 된다. Layer 에서의 오브젝트 렌더링은 오브젝트들을 Layer들로 나누고, 각각의 Layer 에 서로 다른 이미지를 생성시키는 과정이다 . 예를 들어, 다음 씬은 두개의 분리된 요소들 - 전면부와

배경부로 렌더링 되어, 백그라운드 이미지와 합성되었다.

그림 1. Front Part Character

그림 2. Front Part Character Occlusion

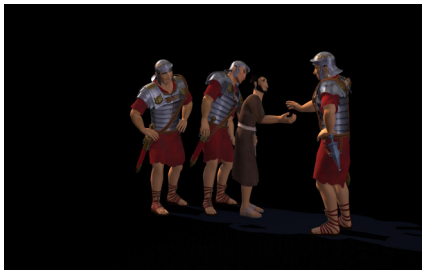


그림 3. Background Character

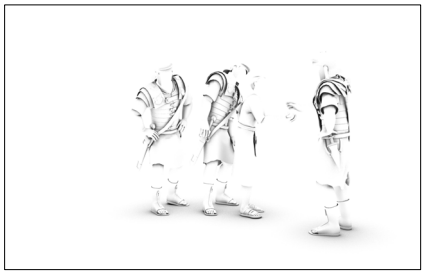


그림 4. Background Character Occlusion

그림 5. Background Image

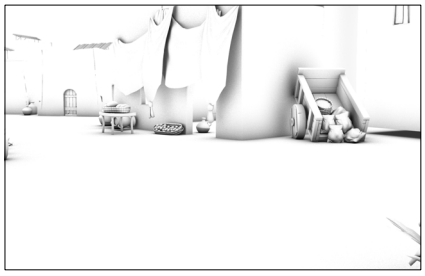


그림 6. Background Occlusion

그림 7. Render Layer Composed Image

○ Render Layer의 장점

Layer 와 패스를 통해 각각의 오브젝트를 렌더링 하는 작업은 계획을 수립할 때 많은 시간과 노력이 필요하다 . 하지만 이 방식엔 다음과 같은 장점들이 있다 .

• 보다 자유로운 수정

사용자는 개별 Layer를 서로 다른 렌더 옵션을 사용하여 렌더링 할 수 있다. 이후 사용자는 합성 어플리케이션에서 컬러를 수정하거나, 특별한 효과를 각 렌더링 된 Layer 에 추가시킬 수 있다.

• 변형의 유연성

과일들이 담겨있는 그릇과 바구니가 있는 탁자 씬이 있다면, 사용자는 전체 씬을 다시 렌더링하지 않고도,

도자기 그릇을 투명한 유리그릇으로 변화시킬 수 있다 .

● 렌더링의 효율성

예를 들어, 사용자는 오브젝트들이 거의 동작하지 않는, 전체 씬 또는 백그라운드를 제외하고, 대부분의 변화가 일어나는 전면 Layer 만을 빠르게 렌더링 할 수 있다. 사용자는 또한 큰 규모 장면의 작은 부분들을 렌더링 하여, 컴퓨터 메모리의 부하를 감소시킬 수 있다.

○ 합성을 위한 렌더링 일반 팁들

- 제작 과정 초기에 장면을 요소들로 분할하기 위한 계획을 세운다.
- 사용자가 렌더링 된 이미지들의 합성을 계획한다면, 장면의 백그라운드가 검정인지 확인하여야 한다.
- 합성기에 필요한 이미지 타입을 이해한다. 계획한 장면들이 미리 곱해졌는지 여부를 확인한다 .

○ Pre-multiplied 이미지들

이미지가 세 가지 컬러 채널 뿐 아니라, 알파채널을 포함하여 저장된 경우 알파 채널의 존재는 컬러 채널들을 일정수준 변화시킨다. 예를 들어 일반적으로 컬러 채널은 알파 채널의 값에 의해 곱해져서, 투명도 (Transparency) 를 표현하게 된다. 일부 합성기들 (Compositors-게임엔진과 같은-) 은 미리 곱해진 이미지를 사용할 수 있다. 그 밖의 합성기들은 분리된 이미지와 알파 정보를 필요로 한다. 특히 합성기가 백그라운드 컬러 데이터로부터 오브젝트 컬러 데이터를 분리하고자 하는 경우가 그러하다. 기본 설정에 의해, 일반적인 3D 프로그램은 이미지들의 값을 미리 곱한다. 하지만 사용자가 미리 곱하는 기능을 off 할 수도 있다.

○ Rendering 채널들

렌더링 과정에서 일반적인 3D 프로그램은 컬러 채널 (RGB), Mask 채널(RGBA), Depth 채널 (RGBZ) 또는 이 세 가지의 합성 (RGBAZ) 를 포함하는 이미지 파일을 생성시킨다. 일부 이미지 포맷들은 덧씌워진 Mask 또는 Depth 채널들을 포함하지 못한다. 이러한 경우 별도의 Mask 또는 Depth 파일을 생성한다. 기본 설정에 의해 세 가지 컬러 채널과 Mask 채널을 포함하는 이미지 파일을 생성한다. Mask 채널과 Depth 채널들은 주로 합성에 사용된다. 사용자는 렌더링 된 이미지 파일에 포함시킨 채널들의 타입을 컨트롤 할 수 있다. 예를 들어, 사용자가 렌더링 된 이미지들의 합성 계획이 없다면, 사용자는 렌더링 과정에서 Mask 또는 Depth 채널을 생성시킬 필요가 없다.

○ Mask 채널들

Mask 채널 ( 또는 알파 채널 ) 은 이미지가 불투명할지, 투명할지를 결정한다. 오브젝트의 불투명 부분은 백색이고, 반투명지역은 회색 , 투명 지역은 흑색이다. 합성 소프트웨어를 위해 이미지들을 Layer 에 넣는데 Mask 채널을 사용할 수 있다. 예를 들면, 사용자는 이미지의 Mask 채널을 Matte 로 사용하여( 배경이 없는 ) 오브젝트를 다른 이미지와 합성시키는데 사용할 수 있다.

○ Depth 채널들

Depth 채널 ( 또는 Z Depth, Z Buffer 채널 ) 은 이미지의 3D 정보를 제공한다. 이는 카메라로부터 오브젝트의 거리를 표시한다. Depth 채널은 합성 소프트웨어에 사용 된다. 예를 들어, 사용자는 Depth 채널을 사용하고 적절하게 맞물려 다수의 Layer 들을 정확하게 합성시킬 수 있다[6].

## Ⅲ. 대용량 파일을 위한 Render Layer 정보 추출과 적용

### 1. Layer 정보 추출을 위한 작업 흐름

작업의 흐름은 크게 몇 가지로 나눌 수 있다. 첫 번째로 필요한 과정은 3D 렌더링 전의 파일에서 기존에 파일이 가지고 있는 Render Layer 정보를 추출해야 한다. 이는 아래의 그림처럼 Maya API를 활용하여 Render Layer 정보를 추출하고 그 정보는 UI를 가지고 있는 Main 부분으로 넘어가고 Main에서 유저가 수정을 한 후 그 값을 다시 Maya API함수를 활용하여 3D 파일에 적용을 시킨다. 그 다음 과정으로 Alfred에서 Render를

실행 할 수 있도록 Script파일을 생성시키는 것이다. 이 Script파일에는 Render Layer별 Render Option의 값을 포함 하고 있다. 그다음 마지막 과정으로 생성된 Script를 이용하여 Alfred를 통하여 렌더링을 실행 시키는 것이다.

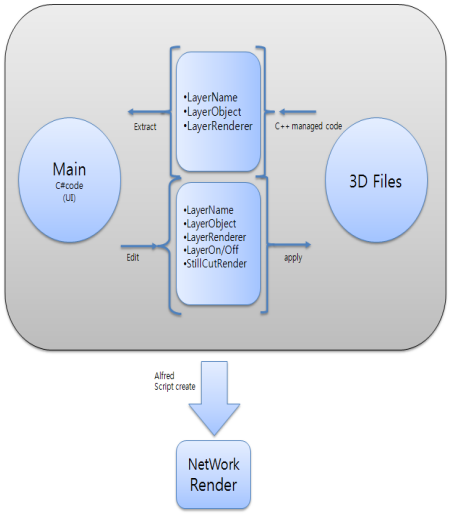



그림 8. 3D 렌더링을 위한 프로그램 흐름도

## 2. 정보 추출 방법

일반적으로 파일에서 정보를 가지고 오는 방법으로 첫 번째는 파일 포맷이 ASCII일 경우로 3D 프로그램 자체가 API를 지원하지 않는다면 Text로 된 내용을 한 줄씩 읽어서 원하는 정보를 추려 낼 수 있다. 이 경우 어떤 파일이든 쉽게 원하는 정보를 추출할 수 있지만 속도가 느려질 수 있다. 예를 들면 200MB의 Maya ASCII에서 정보를 추려 낸다면 아마도 몇 분은 걸릴 것이다. 주로 우리가 다루어야 할 파일들의 크기가 100MB~ 300MB가 되므로 이는 실용성이 없다. 또한 일반적으로 Maya 파일은 확장자가 mb인 Binary형식으로 구성되어 있음으로 Maya API를 활용함이 효율적이고 파일크기 자체도 Binary일 경우가 약 반 정도로 공간도 적게 차지한다.

○ 실용적인 정보 활용을 위한 C# 코드를 활용한 정보 추출

Render Layer를 위해 추출해야 될 정보의 대상이 되는 List는 다음과 같다.

**표 2. Maya API를 활용한 정보 추출 List(일반적인 렌더링을 위한 추출 항목(카메라와 기타 렌더링 옵션들)은 본 논문의 주요 연구 대상이 아님으로 언급하지 않도록 함)**

| Render Layer를 위한 추출 List |
|---|
| RenderLayer Name, RenderLayer별 Renderer, RenderLayer별 Objects, 전체 Objects |

일반적으로 Maya API를 활용하여 정보를 추출하는 방식은 Maya API의 함수를 활용하여 필요정보를 추출하지만 우리가 개발할 Main Controller가 C#기반이므로 Maya API로 정보를 추출 후 이들의 정보를 Managed 코드로 전환한 후 이를 다시 C# 코드가 읽을 수 있도록 해야 한다. 아래 그림은 일반적인 경우의 정보 추출 흐름이다.

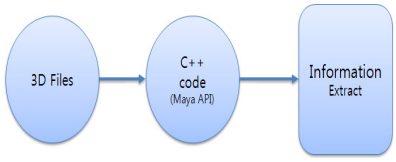



그림 9. 일반적인 정보추출

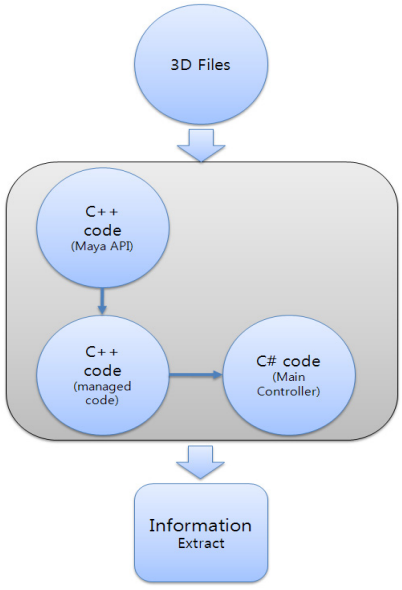



그림 10. C# code를 활용한 정보 추출

위 그림은 C++코드에서 Managed 코드로 전환 뒤 C#에서 사용될 수 있는 흐름을 보여 준다.

프로그래밍 내부 형식으로는 타임라인 상의 시작 키 프레임과 마지막 키 프레임의 정보 추출의 예를 본다면 다음과 같다.

▪ C++ Maya API코드

```
MayaApp::ReturnCode
MayaApp::mf__GetminmaxTime(char
*pszSceneFile, char    **ppszResult, String^&
returnString)
{
MStatus    stat;//status 변수 지정
MFileIO::newFile( true );// File New
MFileIO::open(pszSceneFile, NULL, true);//File Open
if (!stat) {
            MString    strError("Error: MFileIO::open():
            Failure");
            size_t        sizeOfBuffer = strError.length()
+           1;// 1 for '\0'
            *ppszResult = (char *) malloc(sizeOfBuffer);
            if (!*ppszResult) {
                        return
MayaApp::malloc__Failure;
                        }
            sprintf_s(*ppszResult, sizeOfBuffer, "%s",
            strError.asChar());
            return MayaApp::MFileIO_open__Failure;
            }// 파일의 오픈시 status를 리턴한다.
 MTime MinT = MAnimControl::animationStartTime();
//animation의 시작 부분을 지정한다.
 MTime MaxT = MAnimControl::animationEndTime();
// aniamtion의 끝부분을 지정한다.
 double temMinT = MinT.value(); //MTime 형식에서
double형식으로 전환
 double temMaxT = MaxT.value();//MTime 형식에서
double형식으로 전환
 returnString = temMinT.ToString()+"
"+temMaxT.ToString(); //string으로 전환 후 리턴 값을
만든다.
 return MayaApp::Ok;//status에 대한 리턴값
 } [7].
```

▪ Managed 코드

위의 코드처럼 Maya API를 활용하면 정보를 추출할 수가 있지만 C#에서 Maya API를 적용하기 위해선 Managed 코드로 전환하고 dll로 컴파일을 해야 사용이 가능하다. Min(Max) Time을 구하는 Managed 코드는 다음과 같다.

```
String^ ManagedMayaApp2::ExecuteGetminmax (
            String^ strSceneFile)
{
String^ returnString;
System::Text::UTF8Encoding ^utf8 = gcnew
System::Text::UTF8Encoding;
array<Byte> ^bytes = utf8->GetBytes(strSceneFile);
char *pszSceneFile = new char[bytes->Length + 1];
int i;
for (i = 0; i < bytes->Length; ++i) {
                        pszSceneFile[i] = bytes[i];}
            pszSceneFile[i] = '\0'
            char *pszResult = 0;
            MayaApp::ReturnCode rc =
            m__MayaApp->mf__GetminmaxTime
            (pszSceneFile, &pszResult, returnString);
            //여기서 기존의 함수를 지정한다.
            String^ strResult = gcnew
String(pszResult);
            free(pszResult);
            return returnString;
}
```

이 코드는 C#과 Maya API코드를 연결하는 역할을 하는 예를 “타임라인상의 시작 키프레임과 끝 프레임을 추출‘로 설명을 한 것이다. Render Layer에 대한 추출 방식도 같은 방식을 사용한다. 렌더 레이어에 대한 정보 추출도 이와 같다. 사용하는 Maya API의 함수를 렌더 레이어에 관한 함수 MFnRenderLayer ()를 활용한다..

## 3. Render Layer 정보 추출과 수정

○ Render Layer 추출 확인

일반적으로 대용량 파일들은 그 큰 사이즈의 용량 때문에 되도록 파일을 다루는 작업을 최소화 하는 것이 유리한데 그 이유는 시간적 소모가 적기 때문이다. 그러므로 한번 읽은 파일은 새로운 옵션 값을 적용하여 파일을 닫고 다시 열기 전에는 다시 읽을 필요가 없다. 즉 정보의 추출이 한 번에 모두 이루어지고 추후 필요할 때 이미 읽은 정보를 다시 가져오면 된다. 이런 이유로 추출의 확인이 필요하다. 아래 그림은 추출에 관한 작업 흐름을 보여준다.

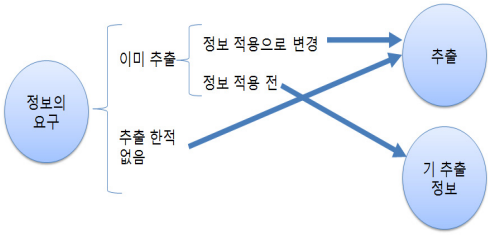



그림 11. 추출 확인

○ Render Layer 추출 함수

전에 추출한 적이 없는 경우 Render Layer 추출 함수를 적용하는데 필요한 정보는 존재하는 Render Layer의 이름과 각 Layer별 렌더러와 각 Layer가 포함된 Object이다. 이때 Object는 Geometry(mesh)와 Light가 해당이 된다. Layer 리스트는 자동적으로 “defaultRenderLayer”를 포함하는데 이는 일반적인 유저가 접근 할 수 없는 Render Layer임으로 이 정보는 추가하지 않도록 한다.

레이어별 렌더러의 정보는 Maya Soft와 MentalRay 두 가지로 한정하도록 한다. 초기에 Layer가 만들어지면 Maya Soft로 지정이 된다. 이후 그 Layer를 MentalRay로 지정하면 해당 Render Layer의 노드의 Source 방향으로 ”miDefaultOptions“ 노드가 첨부된다. ”miDefaultOptions“ 는 MentalRay에 관한 옵션 값을 지정하는 노드이다. 그러므로 노드가 있음으로 MentalRay 렌더러가 해당 레이어에 지정되어 있는지를 알 수 있다.

아래 그림은 초기 Render Layer 노드에 miDefaultOptions가 붙여진 모습이다.

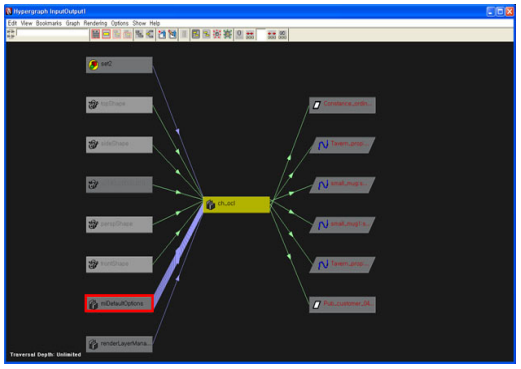


그림 12. miDefaultOptions 노드연결

Layer별 Mesh 정보는 추후 각각의 Mesh가 어느 Layer에 해당되는지 지정 혹은 재지정을 할 수 있음으로 꼭 필요한 정보이다. 그리고 전체 Mesh의 정보도 필요한데 수정 시 기존의 Layer별로 속한 Mesh외에 씬 안에 있는 다른 Mesh를 Layer에 포함 할 수도 있음으로 전체 Mesh의 정보도 필요하다.

○ Render Layer 정보의 수정

Render Layer의 정보의 수정에는 다음 4가지 요소들이 있다.

1. Renderer(Layer 별 /sw, mr )
2. Layer Objects(추가, 삭제)
3. Layer Activity(실제 렌더링을 수행 할 것인가?)
4. Layer 별 전체 프레임 혹은 한 프레임(still / All)

1, 3, 4는 실제 파일을 수정하지 않더라도 수행이 가능하다. 당연히 직접적으로 수정도 가능하지만 이럴 경우 실제 해당 파일을 수정과 저장을 하여야 함으로 그만큼 시간이 더 필요하게 된다. 그렇다면 실제 파일을 수정하지 않고 하는 방법은 renderer의 Command Line 명령을 이용하는 것이다.

render -r mr 해당파일path

위의 렌더링명령 뒤에 -r mr(MentalRay일 경우 / MayaSoft일경우는 sw)의 추가 명령으로 수행이 가능하다. Layer Activity는 Alfred script에서 해당 Layer의

Script를 작성하지 않는다면 렌더링이 되지 않는다. 그리고 Layer 별 전체 프레임 혹은 한 프레임도 Alfred Script에서 전체 프레임과 단하나의 프레임으로 렌더링 명령이 가능하다. 이렇게 Renderer Command Line명령 자체를 활용함으로 시간적 소모를 줄일 수 있다.

1 프레임부터 3 프레임까지 렌더링을 처리 하는 Alfred Script는 아래와 같다.

<소스코드1>

```
##AlfredToDo 3.0
Job -title {Stest3mmcam} -subtasks {
Task {Frame #1} -cmds {
        RemoteCmd {render -of iff -fnc 3 -s 1 -e 1
        -pad 3 -cam mmcam_testShape -x 320 -y
        240 -ard 1.33333 -eaa 3 -ss 1 -mss 8 -mvs
        1 -mvm 4 -pss 1 -ufil False -pft 2 -pfx 2.2
        -pfy 2.2 -ert False -rfl 1 -rfr 6 -sl 2 -g 1
        -premul True -premulthr 0 -preRender ""
        -postRender "" -preLayer "" -postLayer ""
        -preFrame "" -postFrame "" -rd
        Y:/kiTest/test4/ -im
        Stest3mmcammmcam_testShape
        Y:/kiTest/test4/test3mmcam.mb} -service
        {pixarRender}
        # Cmd {CMD /C DIR}}
Task {Frame #2} -cmds {
        RemoteCmd {render -of iff -fnc 3 -s 2 -e 2
        -pad 3 -cam mmcam_testShape -x 320 -y
        240 -ard 1.33333 -eaa 3 -ss 1 -mss 8 -mvs
        1 -mvm 4 -pss 1 -ufil False -pft 2 -pfx 2.2
        -pfy 2.2 -ert False -rfl 1 -rfr 6 -sl 2 -g 1
        -premul True -premulthr 0 -preRender ""
        -postRender "" -preLayer "" -postLayer ""
        -preFrame "" -postFrame "" -rd
        Y:/kiTest/test4/ -im
        Stest3mmcammmcam_testShape
        Y:/kiTest/test4/test3mmcam.mb} -service
        {pixarRender}
        # Cmd {CMD /C DIR}}
Task {Frame #3} -cmds {
        RemoteCmd {render -of iff -fnc 3 -s 3 -e 3
        -pad 3 -cam mmcam_testShape -x 320 -y
        240 -ard 1.33333 -eaa 3 -ss 1 -mss 8 -mvs
        1 -mvm 4 -pss 1 -ufil False -pft 2 -pfx 2.2
        -pfy 2.2 -ert False -rfl 1 -rfr 6 -sl 2 -g 1
        -premul True -premulthr 0 -preRender ""
        -postRender "" -preLayer "" -postLayer ""
        -preFrame "" -postFrame "" -rd
        Y:/kiTest/test4/ -im
        Stest3mmcammmcam_testShape
        Y:/kiTest/test4/test3mmcam.mb} -service
        {pixarRender}
        # Cmd {CMD /C DIR}}
} -cmds {
# Cmd {CMD /C DIR}          }
```

위 Alfred Script는 Job아래에 3개의 Task(프레임별 명령, 여기선 1프레임부터 3프레임으로 3개임)가 있고 이들 Task들은 RemoteCmd로 명령을 외부 렌더 팜에 보내는 역할을 한다. 그리고 RemoteCmd안에 있는 본 명령들은 Renderer들의 Command Line 명령에 해당한다.

Script는 프레임별로 렌더링 명령이 나누어져 있는 것을 볼 수 있다. 이는 분산 처리 시 각 CPU에게 렌더링 명령을 나누어 주기 위함이다.

Layer Objects(추가, 삭제)의 경우는 실재 파일을 수정 하지 않으면 불가능한 것으로 직접 수정 명령이 필요하다. 명령 수행 후 저장 명령도 같이 수행 되므로 큰 파일일수록 시간이 더 소요된다. 그러나 이경우도 최종 렌더링을 수행 시 한번만 수정과 저장이 필요함으로 그 전에 수정 사항은 UI에서만의 수정으로 하여 시간을 조금 더 줄일 수 있다. 즉 Objects의 추가, 삭제 경우마다 파일을 변경 저장하는 것이 아니라 최종 렌더링 시 한꺼번에 변경 저장한다. 그러므로 렌더 명령 수행 전에는 여러 번 변경하더라도 파일에 직접적으로 적용이 되지 않는다.

아래 그림에서처럼 중간 부분의 선택된 파일에서 정보를 추출하여 Layer이름과 각각의 Layer별 정보를 보여 주고 있다. 즉 중앙에 선택된 파일에 File_Info 버튼

을 누르면 파일이 갖고 있는 Layer들에 대한 정보와 그리고 각각의 Layer들에 해당되는 Object들의 리스트와 그 Layer의 정해진 Renderer에 대한 정보를 추출한다. 렌더링은 맨 앞에 보이는 Layer에 해당하는 체크 박스에 해당하는 Layer만 렌더링이 수행된다. 그다음 Still 항목은 애니메이션 전체 프레임을 렌더링 하지 않고 첫 장만 렌더링 할 수 있는데 주로 배경에 해당하는 Layer에 하당된다. 그 다음 항목은 Renderer의 전환에 해당된다. 상위의 Renderer 버튼을 누르면 선택된 Layer의 Renderer가 바뀌게 된다. 그 다음 항목은 선택된 Layer에 해당하는 Object를 보여준다. 이는 추가와 삭제가 가능하다. 그다음 항목은 파일에 있는 전체 object에 대한 리스트를 보여준다. 여기서 필요한 object를 추가 할 수 있다.

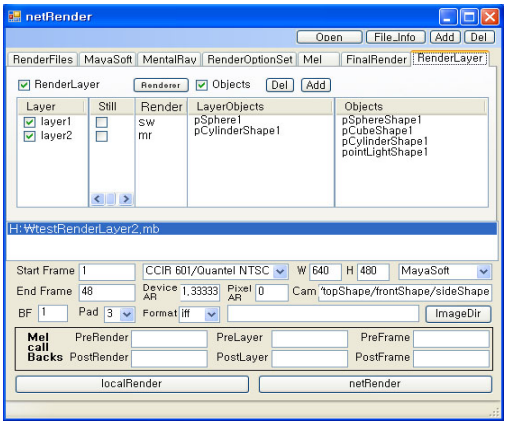



그림 13. Render Layer UI

## Ⅳ. 결 론

본 연구는 3D 애니메이션에서 최종 결과물을 생산해 내는 과정 즉 렌더링의 과정에서 빈번하게 발생하는 수정과 그것으로 발생하는 Data(파일)를 다루는 일의 손실과 수정 시 불필요한 부분의 재 렌더링 작업을 최대한 줄이기 위하여 3D 파일 자체를 열지 않고 Render Layer값을 수정함으로 작업의 효율을 극대화 할 수 있는 Software의 개발에 관한 연구이다. 일반적으로 Command Line 방식은 대 다수의 작업자들의 사용하지 않는 방식이다. 왜냐하면 각각의 파일에 대하여 미리 구체적인 정보를 알고 있어야 렌더링이 가능하고 심지어 Path 정보까지 완벽하게 알고 있어야 한다. 그리고 무엇보다도 Layer에 대한 정보의 추출과 변경은 Command Line으로 조정이 불가능하다. 수정작업에 있어서 기존의 파일이 무슨 값을 가지고 있는지를 알 수 있어야하고 이를 기반으로 변경도 할 수 있다. 기존의 렌더 정보 추출 및 수정에 관한 연구논문은 발견 할 수가 없었고 www.highend3d.com에서 2가지 유사 도구를 발견 할 수 있었다. 그러나 두 가지(hj BatchRender, Front BatchRender) 도구모두 기존 파일의 Batch Rendering의 개념이지 정보의 추출과 수정은 불가능하기 때문에 성능의 비교 대상이 될 수가 없다.

| | hj BatchRender, Front BatchRender | 연구결과 활용한 Tool 적용 |
|---|---|---|
| 독립적인 도구 | 아님(maya를 기반으로 수행되는 Mel Script) | 독립적인 도구임 |
| 정보 추출 | 불가능 | 가능 |
| 정보 수정 | 불가능 | 가능 |
| 동시 다수 파일 지원 | 불가능 | 가능 |
| Render Layer 지원 | 없음 | 지원 |

본 연구 결과를 일반적인 렌더링 작업과 비교를 하면 아래 표와 같다. 기준은 100개 씬 수정 작업, Open과 Save에 각각 2분 정도 소요되는 약 150MB의 씬으로 Character Layer 부분만 Render Layer값 수정 작업(약 5분소요)으로 비교를 하였다. 사용된 컴퓨터의 사양은 펜티엄 4 3.0, 2G Ram, GForce 7900, 200G Hard를 갖춘 컴퓨터를 사용하였다.

| | 일반적인 렌더링 방법 | Command Line방법 | 연구결과 활용한 Tool 적용 |
|---|---|---|---|
| 동일한 100개의 씬에 대해 각각 다른 값 적용 (character Render Layer만 수정) | Open 2분 *100개<br>Save 2분 *100개<br>수정 작업 5분 *100개<br>렌더링 시간 (1 still Cut) : 5분 *100개<br><br>총 1400분 소요 | 일반적인 파일의 렌더레이어 옵션의 변경 없이 그대로 렌더링은 가능하나 정보의 추출과 변경 후 렌더링은 불가능하다. | 수정 작업 5분 *100개<br>렌더링 시간 (1 still Cut) : 2분 *100개<br><br>총 700분 소요 |

애니메이션 제작과정의 불가피한 수정작업은 병목현상을 유발하고 제작관리는 심각한 어려움에 직면하게 된다. 결론적으로 제작일정과 예산을 정확히 관리해야 하는 상업적 애니메이션 프로젝트에서 대용량 다수의 데이타를 쉽게 다루는 기술의 필요성은 더욱 늘어날 것이다.

## 참 고 문 헌


[1] 김기홍 “다수의 대용량인 3D Render 파일을 위한 3D CG 소프트웨어와 독립적으로 Render Option 값 추출과 적용 가능한 Net-Work Rendering Tool 구현”, 한국애니메이션학회, 애니메이션 연구, Vol.3 No.1, 2007.

[2] 김현빈, 김기호, 김진서, 김해동, 박창준, 이지형, 정일권, 추상우, 이민기, "휜히보이는 디지털시네마", ETRI, 2006.

[3] 채일진, “애니메이션 효율적 공정관리를 위한 캐릭터 셋업”, 한국콘텐츠학회, 한국콘텐츠학회논문지, 7권, 4호, 2007.

[4] 한국전자통신연구원, 3D 애니메이션 기술/시장 보고서, 2000.

[5] Alias, "The Art of Maya," Alias System, 2005.

[6] Alias, "Maya 8.5 Help Documents," Alias System, 2007.

[7] David A.D. Gould, "Complete Maya Programming," 2003.

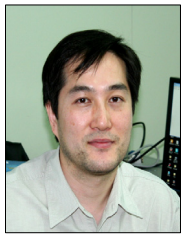

## 저 자 소 개


**김 기 홍(Ki-Hong Kim)** 정회원

- 1995년 2월 : 홍익대학교 화학공학과 (공학사)
- 2001년 2월 : CalArts Experimental Animation(MFA)
- 2002년 3월 ~ 현재 : 동서대학교 애니메이션학과 교수

<관심분야> : 컴퓨터그래픽, IT, 문화 콘텐츠

**최 철 영(Chul-Young Choi)** 정회원



- 1997년 8월 : 한림대학교 물리학과(학사)
- 2002년 5월 : Academy of Art Univ.(USA) Computer Arts(MFA)
- 2002년 5월 ~ 2004년 8월 : (주) 디지스팟, 애니메이션 팀장
- 2004년 9월 ~ 현재 : 동서대학교 디지털 콘텐츠학부 애니메이션 전공 교수

<관심분야> : 3D애니메이션, 모션캡쳐, 비쥬얼 이펙트, 문화 콘텐츠

**채 일 진(Eel-Jin Chae)** 정회원

- 1998년 2월 : 중앙대학교 사진학과(예술학사)
- 2001년 5월 : Academy of Art Univ.(USA) Motion Picture & Television(MFA)
- 2001년 6월 ~ 2002년 10월 : (주) 튜브 픽쳐스, 프로듀서
- 2002년 10월 ~ 2003년 10월 : (주)에픽스디지탈, 프로듀서
- 2003년 11월 ~ 2007년 2월 : (주)디씨디코리아, 대표
- 2004년 9월 ~ 현재 : 동서대학교 디지털콘텐츠학부 디지털영상제작전공 교수

<관심분야> : 멀티미디어, 소프트웨어, 문화콘텐츠, 기획, 마케팅