

게 임 엔진

김현

# 목차

1. 서론

2. 관련 연구

3. 게임 엔진

4. 실험 및 고찰

5. 결론 및 향후 과제

6. References

## 2. 관련 연구

현재 대부분의 게임은 가장 많이 상용화된 언리얼 엔진, 유니티 엔진으로 게임을 제작하고 있다. 빠르고 쉽게 게임을 구현할 수 있고 PC뿐만 아니라 모바일, 콘솔 등 다양한 플랫폼에서도 사용이 가능하다. 하지만 상용화된 엔진을 사용하지 않고 자체적인 엔진을 사용하는 회사도 있다. '검은 사막'을 제작한 '펄 어비스'란 회사가 대표적이다. '펄 어비스' 대표 김대일의 인터뷰 중 이런 말을 했다. “상용화 된 엔진을 구매하여 사용할 경우 그만큼 개발시간을 단축시킴과 동시에 이미 관련 엔진에서 자주 나오는 버그들이 알려져 있어서 수정하기 쉬우며 관련 자료들도 있기에 개발을 하는 데 용이하나, 만들고자 하는 게임을 원하는 대로 100% 전부 만들기에는 부족함이 있다.” 게임 엔진의 개발은 이러한 이점이 있기 때문에 다른 회사의 엔진을 사용하기 보단 자신의 엔진을 사용하는 것이 더 효율적일 수 도 있다.



1. <https://unity3d.com/kr>

상용화 게임 엔진인 유니티 엔진과 언리얼 엔진 로고



2. <https://www.epicgames.com/ko>



3. [www.pearlabyss.com](http://www.pearlabyss.com)

'검은 사막'을 제작한 회사 '펄 어비스'

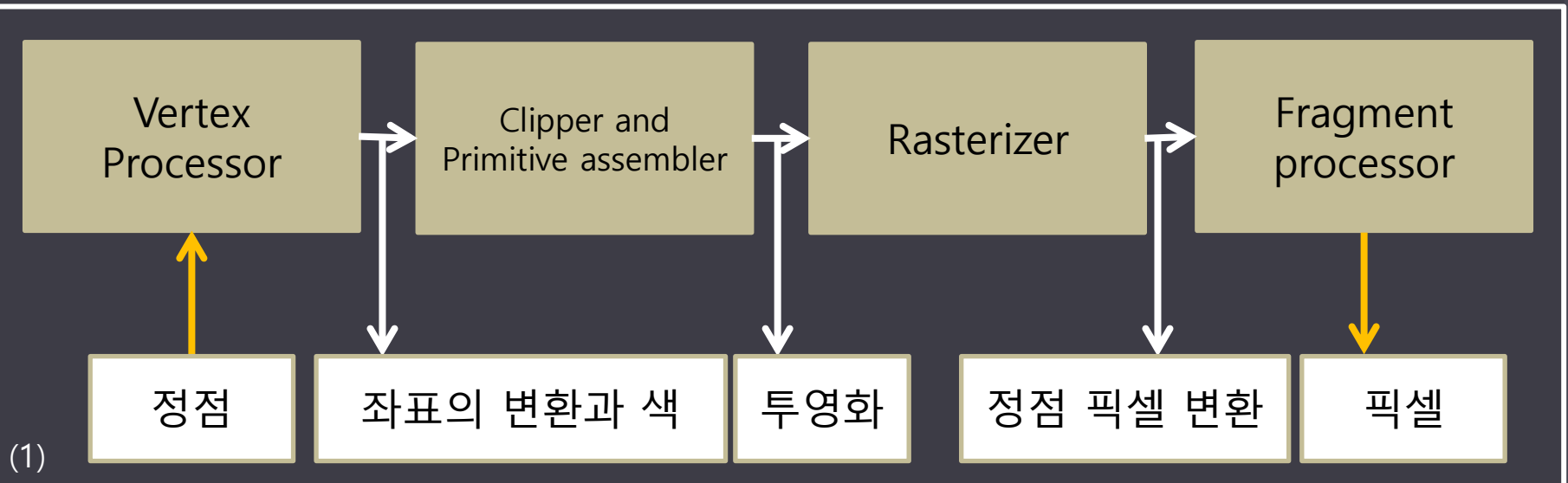
진행한 프로젝트는 게임 엔진을 직접 구현해보는 프로젝트이다. 상용화된 엔진으로 직접 게임을 제작 할 수도 있지만, 어떤 방식으로 구현되어 있는지도 모르는 엔진을 사용하는 것 보단, 직접적으로 게임 엔진을 제작해 봄으로써 어떤 방식으로 게임 엔진이 구현되는지 알고 습득하는 것이 훗날 게임 제작에 더 효과적일 것이다.

## 2. 관련 연구

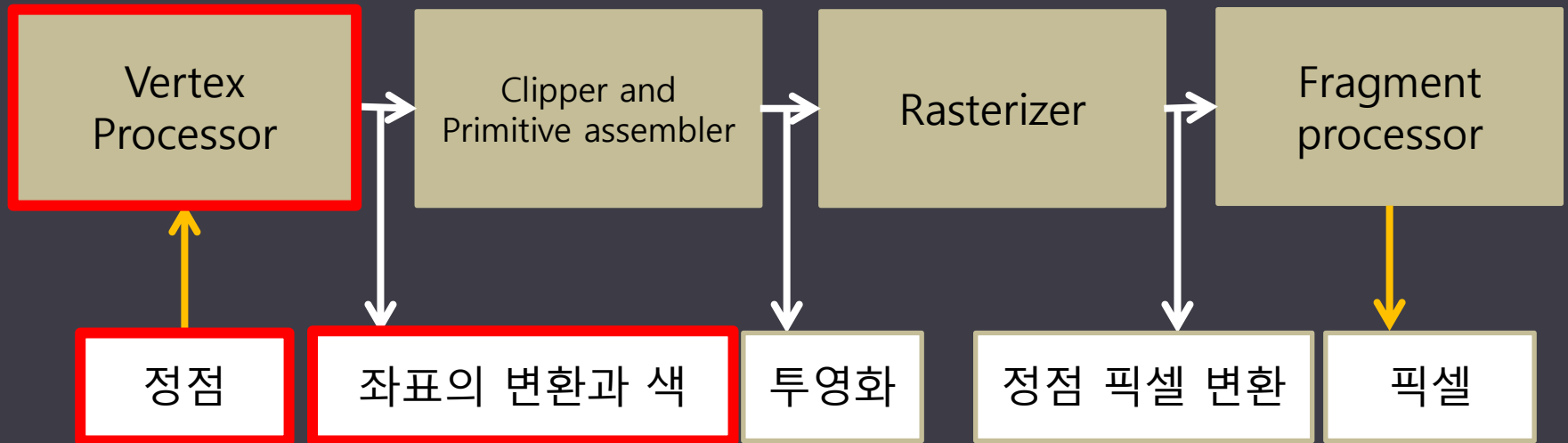
### 2.1. OpenGL

OpenGL은 3D 그래픽 애플리케이션을 개발하기 위한 환경으로 프로그래밍 언어나 소프트웨어가 아닌 3D 애플리케이션을 위한 API( 운영체제나 응용프로그램 사이의 통신에 사용되는 언어나 메시지 형식)이다. 즉, 시각적으로 근접한 2차원 또는 3차원 그래픽 프로그램 개발 API이다.

OpenGL은 기본적으로 컴퓨터 그래픽스 파이프 라인을 따라간다.



## 2. 관련 연구



### VertexProcessor

좌표 변환에 대한 수행과 색상을 결정한다. 정점 셰이더라고 불리는데, 3D 데이터 자체이다. (x,y,z 좌표, 노말 값, uv 값 등) 각 정점의 위치를 찾아주고 변환하는 역할을 한다. 색은 픽셀 셰이더가 맡고 있다. 정점의 색을 결정하는데 화면에 출력될 색상을 결정한다.

좌표 변환은 모두 행렬을 통해 변환이 이루어진다.

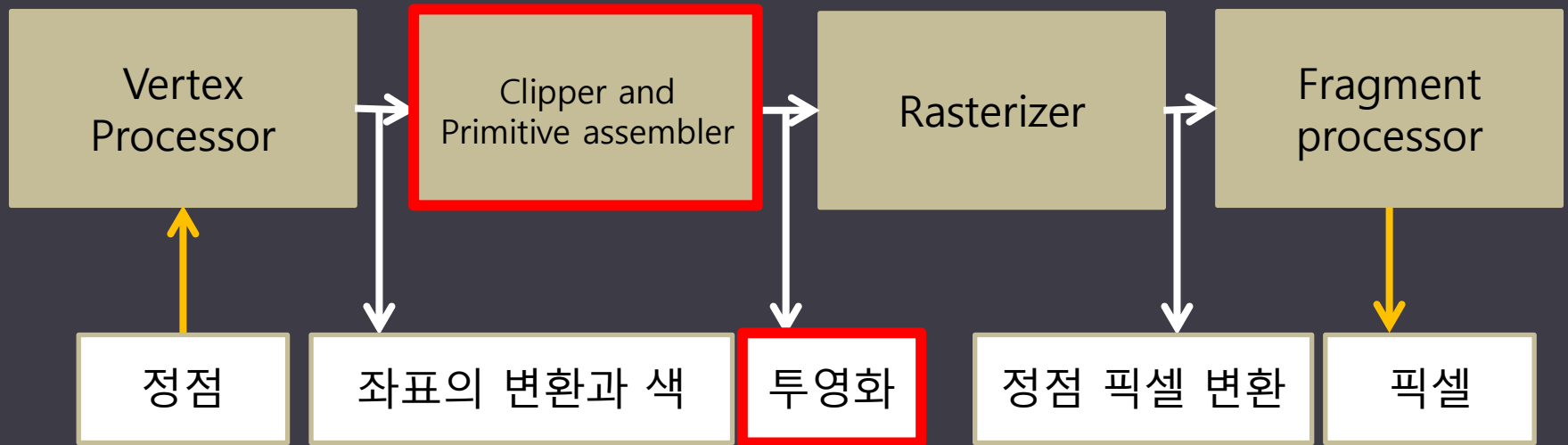
사각형이 이동하면 사각형의 정점들이 동일하게 이동한다.  
해당 하는 변환 행렬을 T라 하고 X,Y축으로 이동한 양을 각각  $T_x, T_y$ 라 하면 두 점 사이의 관계는 이렇하다.

$$P' = T + P$$

$\begin{bmatrix} X' \\ Y' \end{bmatrix}$	$=$	$\begin{bmatrix} T_x \\ T_y \end{bmatrix}$	$+$	$\begin{bmatrix} X \\ Y \end{bmatrix}$
--	-----	--	-----	--

(2)

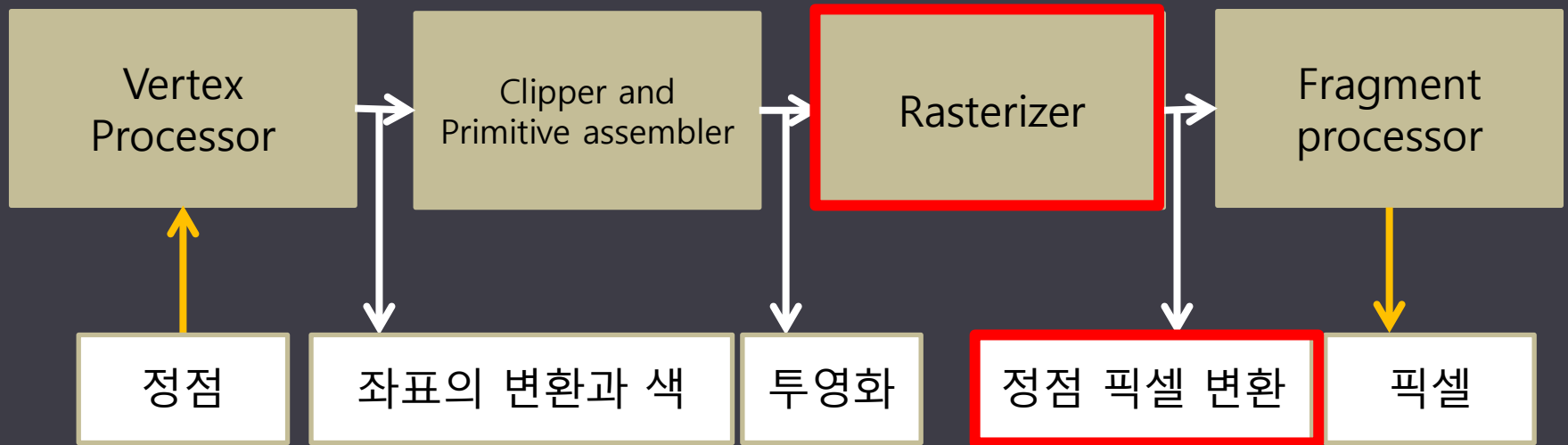
## 2. 관련 연구



### Clipper and Primitive assembler

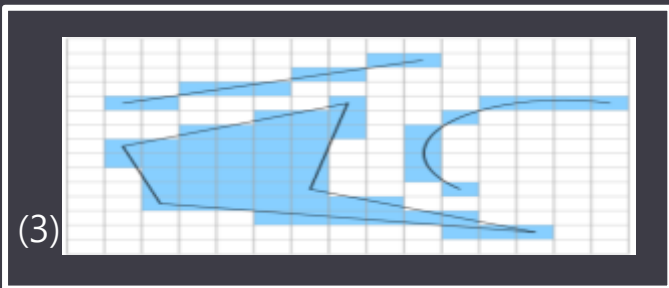
Cripper은 깎아 내린다는 뜻이고, Primitive assembler은 기본요소를 조립한다는 뜻이다. 스크린에 출력되는 모든 그래픽스는 2D로 투영변환 되어 출력하기 때문에 투영될 정점들을 필요 없는 부분을 깎아 내리는 것 ( 보이지 않는 부분을 지운다 ) 을 Cripper의 역할, 투영된 정점을 조립 하는 것을 Primitive assembler가 맡아서 수행한다.

## 2. 관련 연구



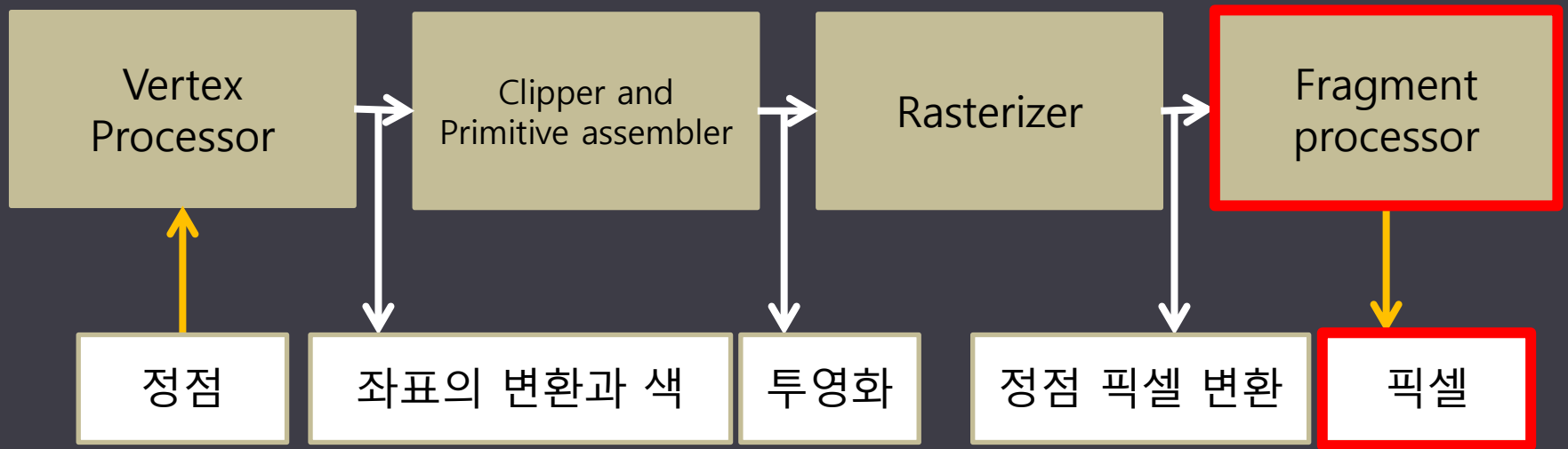
### Rasterizer

레스터화는 입력된 이미지를 출력 가능한 형태로 전환한다. 투영화 된 이미지는 아직 정점으로 표현 되어 있을 뿐 픽셀로 변환되어 있지 않다. 이 정점을 프레임 버퍼 안에 있는 픽셀로 변환하는 과정을 수행한다.



스크린 위에 정점들이 각각의 해당하는 투영된 좌표 (스크린 좌표) 에 픽셀이 그려진 그림이다.

## 2. 관련 연구



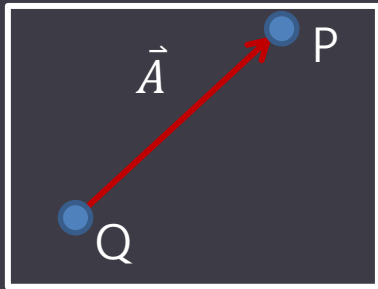
Fragment processor

프레임 버퍼 안에 있는 픽셀을 스크린에 출력하는 역할을 한다.



### 2.2. 그래픽스 요소

그래픽스는 기본적으로 벡터, 스칼라, 점 세 가지 요소를 통해서 기술된다. 벡터는 크기와 방향, 스칼라는 크기, 점은 위치를 가지고 있다. 벡터와 스칼라, 점으로 선분을 표현 할 때



다음과 같이 표현 할 수 있다. Q와 P는 두 점,  $\vec{A}$ 는 방향과 크기를 가지고 있는 벡터이다. 여기서 스칼라는 선분의 길이를 나타낸다.

기하학적 객체는 세 기저 벡터 ( 좌표계 ) 위에 표현 되며 어파인 공간( 기존 벡터 공간에서 위치를 표현하기 위해 점을 더한 공간 )에서의 연산을 통해 오브젝트가 그려진다.

세 요소는 기하학적 객체를 그릴 뿐 아니라 내적을 구해 반사 벡터, 속도 벡터, 힘 벡터 같은 물리적 값을 구할 수 있고, 외적을 통해 새로운 벡터를 생성하여 법선 벡터로도 사용한다.

## 2. 관련 연구

그래픽스는 세 기본요소를 사용하여 행렬을 만들어 사용한다. 모든 변환 ( 이동, 회전, 크기 등)을 행렬의 곱을 통해 변환하는데, 행렬을 사용하는 이유는 모든 변환을 한 행렬로 표현 할 수 있기 때문이다. 회전, 이동, 크기 변환은 다음과 같이 행렬로 표현한다.

● X축회전행렬	● Y축회전행렬	● Z축회전행렬	● 이동변환행렬	● 크기변환행렬
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos & \sin & 0 \\ 0 & -\sin & \cos & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos 0 & -\sin 0 & 0 \\ 0 & 1 & 0 & 0 \\ \sin 0 & \cos 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos & \sin & 0 & 0 \\ -\sin & \cos & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$	$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

(4)

정점 P(1,1,2,1)가 (0,0,3,1) 만큼 이동하고, x축으로 90도 회전 한 뒤, 다시 (1,0,0,1) 만큼 이동한다면 행렬은 다음과 같이 표현 한다.

$$\begin{array}{c}
 \text{이동} \\
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}
 \end{array}
 \begin{array}{c}
 \text{회전} \\
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
 \end{array}
 \begin{array}{c}
 \text{이동} \\
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix}
 \end{array}
 =
 \begin{array}{c}
 \text{최종 변환 행렬} \\
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 3 & 1 \end{bmatrix}
 \end{array}$$
  

$$\begin{array}{c}
 \text{정점 P'} = \\
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 3 & 1 \end{bmatrix}
 \end{array}
 \begin{array}{c}
 \text{정점 P} \\
 [1 \quad 1 \quad 2 \quad 1]
 \end{array}
 =
 \begin{array}{c}
 \text{정점 P'} \\
 [2 \quad -2 \quad 4 \quad 1]
 \end{array}$$

## 2. 관련 연구

객체는 기본적으로 자신의 현재 변환 행렬( CTM : Current Transformation Matrix)를 가진다.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix} = \overset{\text{CTM}}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 3 & 1 \end{bmatrix}} \overset{\text{로컬좌표}}{[1 \ 1 \ 2 \ 1]} = \overset{\text{월드좌표}}{[2 \ -2 \ 4 \ 1]}$$

전 페이지에서 봤던 이동, 회전, 이동 행렬 결과 값이 CTM이다. 자신의 CTM과 자신의 로컬 좌표를 곱하면 자신의 위치인 월드 좌표가 나온다.

일반적으로 정적 모델인 경우에는 월드좌표만 출력하면 제대로 된 오브젝트가 출력되지만 옳은 방법이 아니다. 각 객체의 오브젝트는 자신만의 로컬 좌표계가 있으며 로컬 좌표계를 사용해야 올바른 계층구조 모델을 출력할 수 있다. 계층 구조가 존재하지 않는 오브젝트( 부모가 없는 오브젝트 )는 월드 좌표가 로컬 좌표가 된다.

## 2. 관련 연구

그래픽스에는 여러 좌표계가 존재한다. 월드 좌표계, 로컬 좌표계, 시점 좌표계, 투영 좌표계 등 모든 정점은 이러한 좌표계의 여러 변환을 통해 출력된다. 월드 좌표를 로컬 좌표로 변환해야 한다면,

**로컬 정점 = (자신의 CTM)<sup>-1</sup> \* 월드 정점**

**월드 정점 = 자신의 CTM \* 로컬 정점**

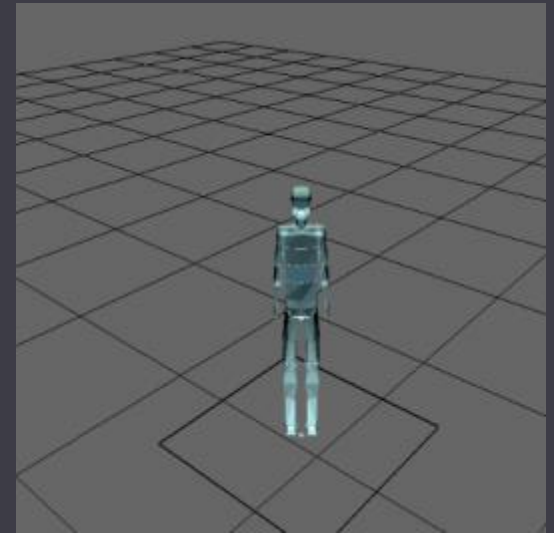
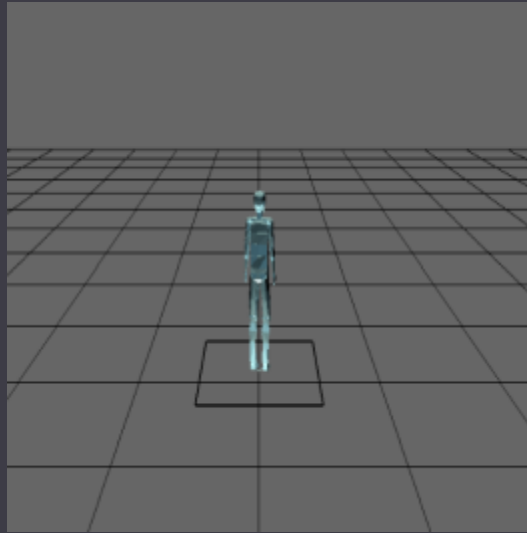
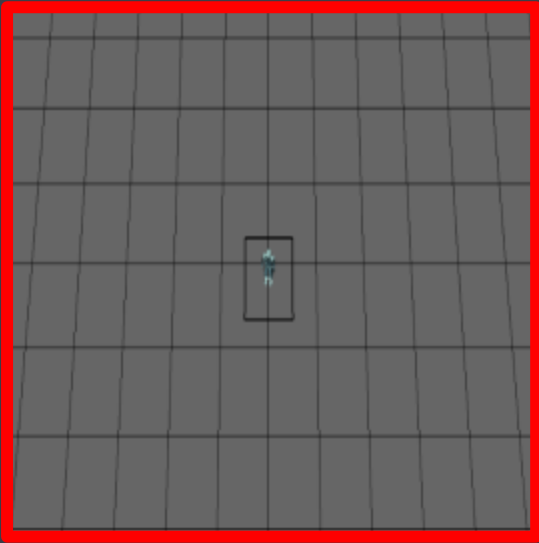
카메라 좌표계도 마찬가지로

**카메라에서 보이는 정점 = (카메라 CTM)<sup>-1</sup> \* 월드 정점**

이와 같이 모든 정점은 여러 변환을 통해서 출력 된다.

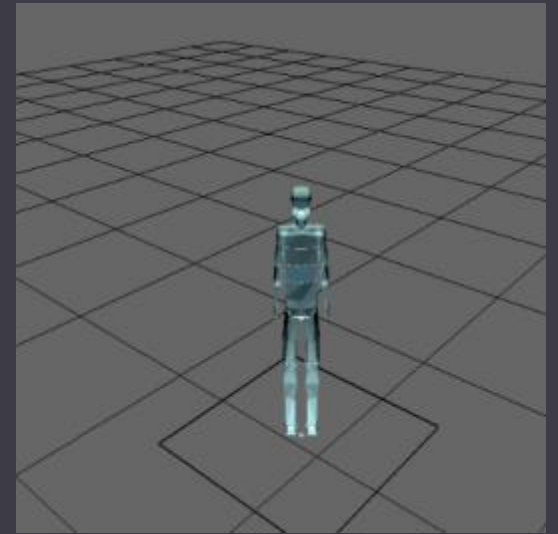
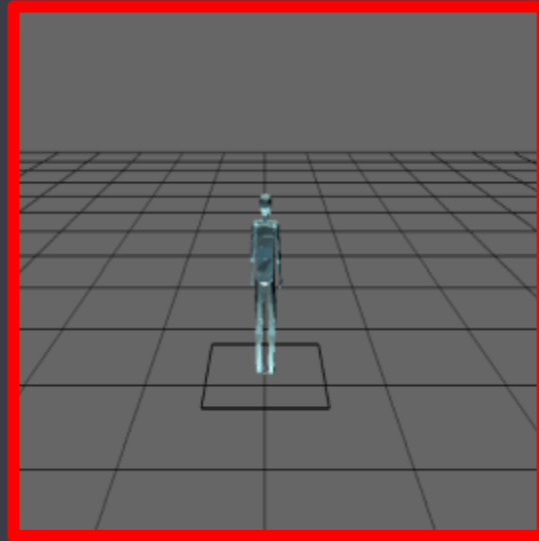
### 3. 게임 엔진

#### 3.1 엔진 구현



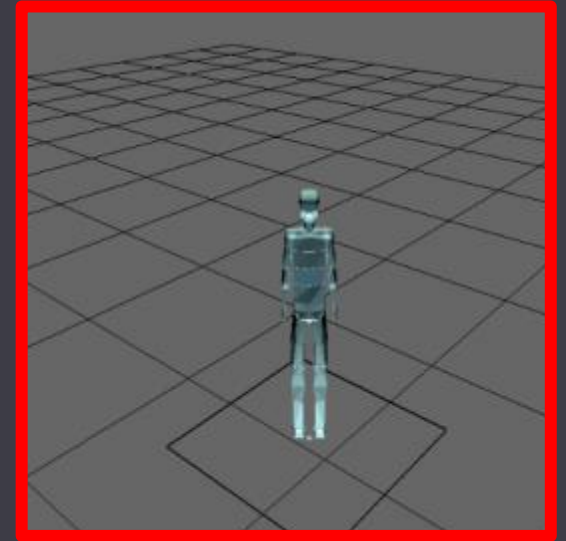
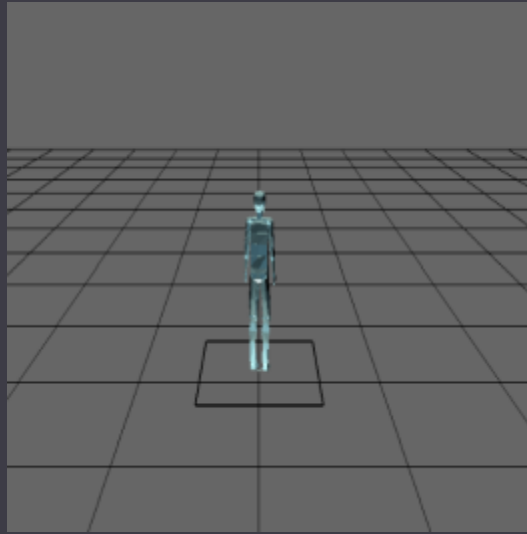
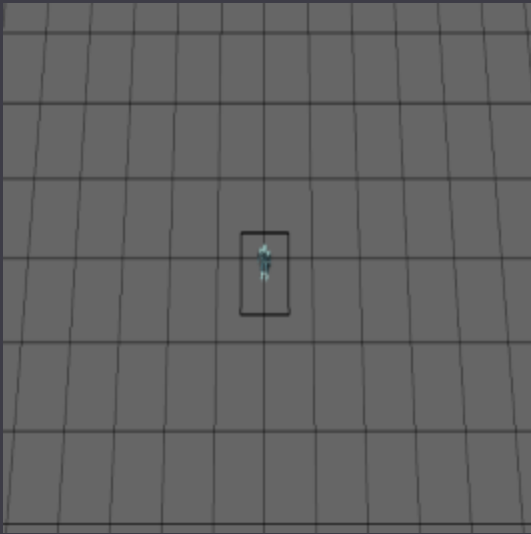
1번을 누르면 카메라 시점을 변경 할 수 있다. Y축 높이에서 밑으로 바라보는 시야 이다. 마우스 휠로 높이를 지정하고 클릭 & 드래그로 위치를 변경 할 수 있다.

### 3. 게임 엔진



다시 1번을 누르면 다음 과 같은 그림으로 변경이 된다. 카메라 시점은 Z축을 이동하면서 오브젝트가 움직이면 자동으로 카메라가 오브젝트와 좌표계(그리드)를 전체적으로 보여준다. 마우스 클릭 & 드래그로 위 아래를 바라 볼 수 있다.

### 3. 게임 엔진

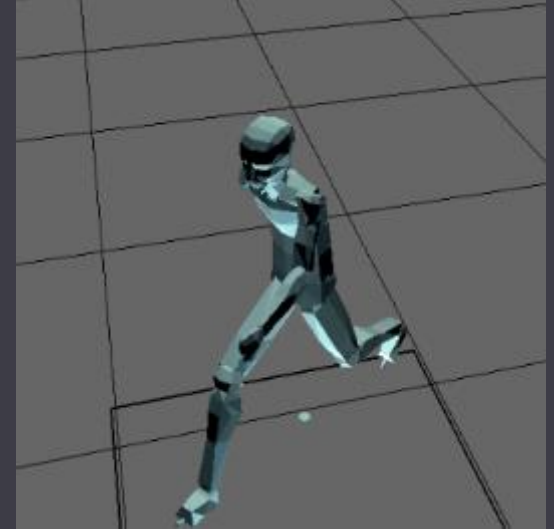
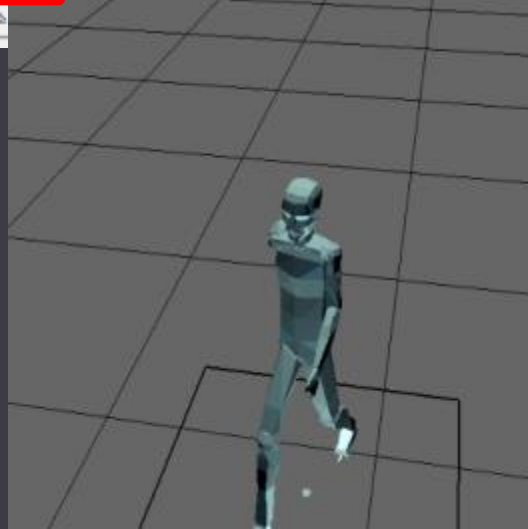
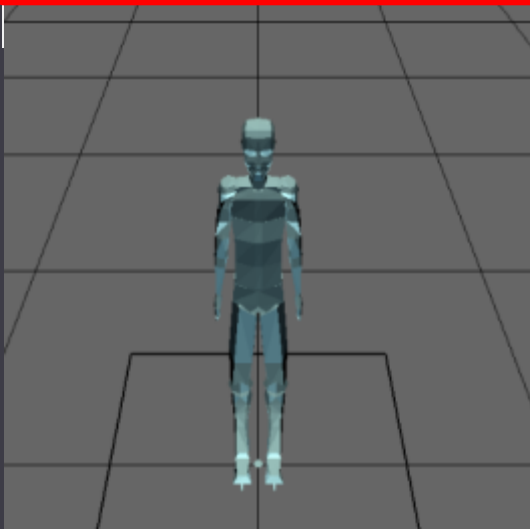


다시 1번을 누르면 다음 과 같은 그림으로 변경이 된다. FPS 게임 처럼 오브젝트를 따라다니며 카메라가 이동한다. 마우스 클릭 & 드래그로 위 아래를 바라 볼 수 있다. 마우스 휠로 줌, 앞과 뒤를 변경 할 수 있다.

### 3. 게임 엔진

오브젝트01.ASE		
오브젝트01_애니메이션01.ASE	추가	삭제
오브젝트01_애니메이션02.ASE	추가	삭제
오브젝트01_애니메이션04.ASE	추가	삭제

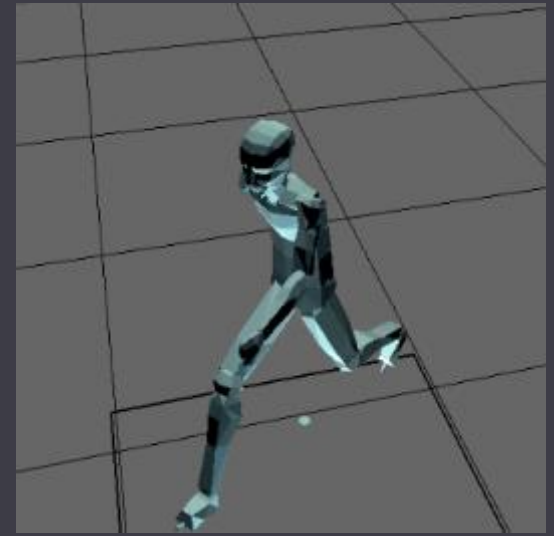
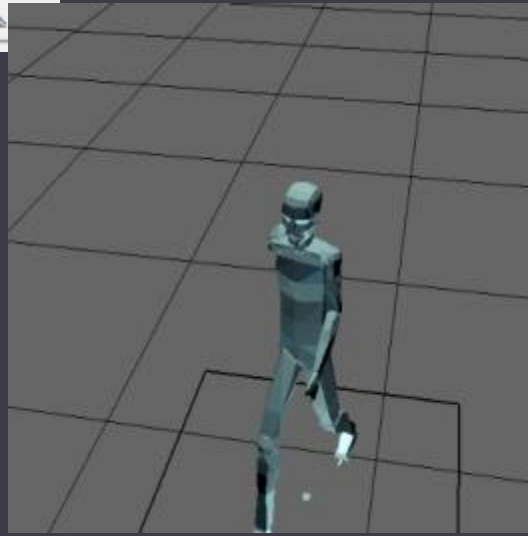
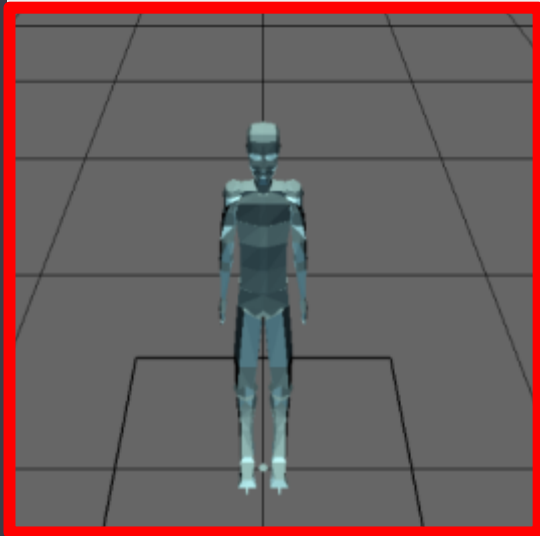
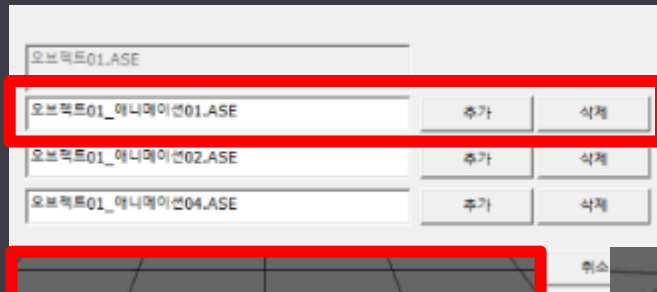
취소



오브젝트와 애니메이션을 추가하는 다이얼로그이다. 오브젝트는 하나, 애니메이션은 3개를 추가 할 수 있다. 애니메이션을 추가하거나 삭제하면 전체적으로 초기화가 되며 오브젝트는 원점으로 다시 이동한다.

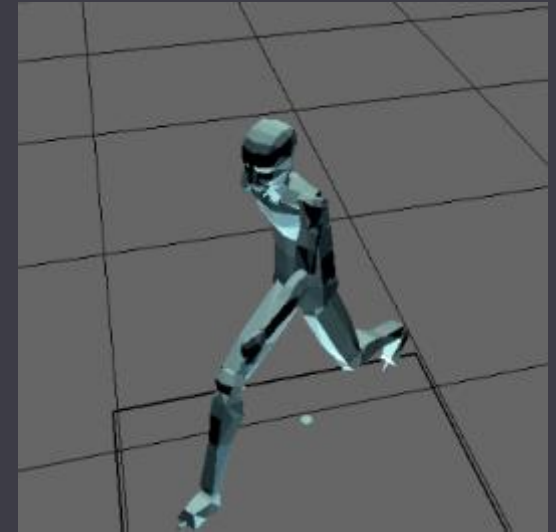
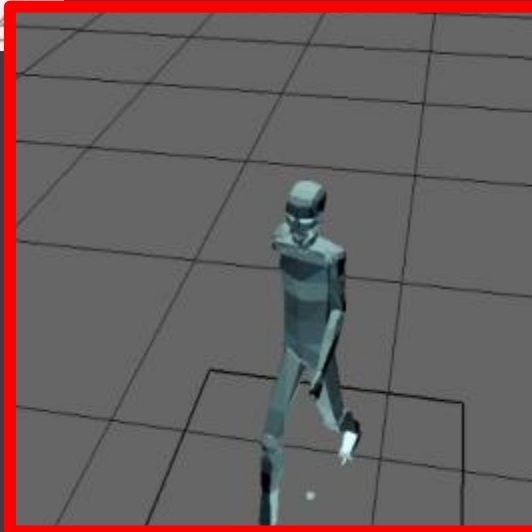
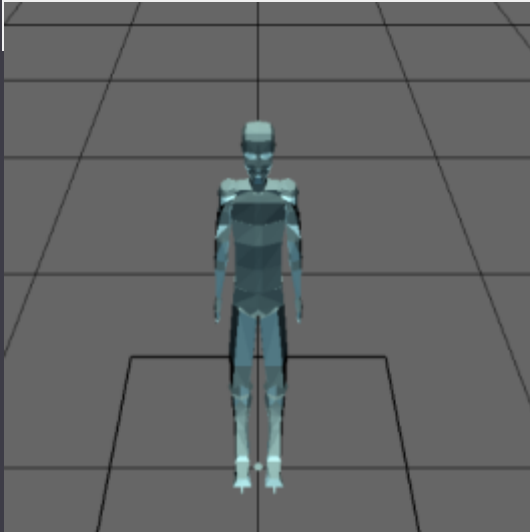
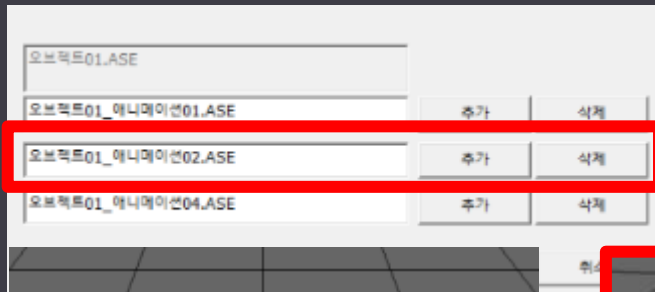


### 3. 게임 엔진



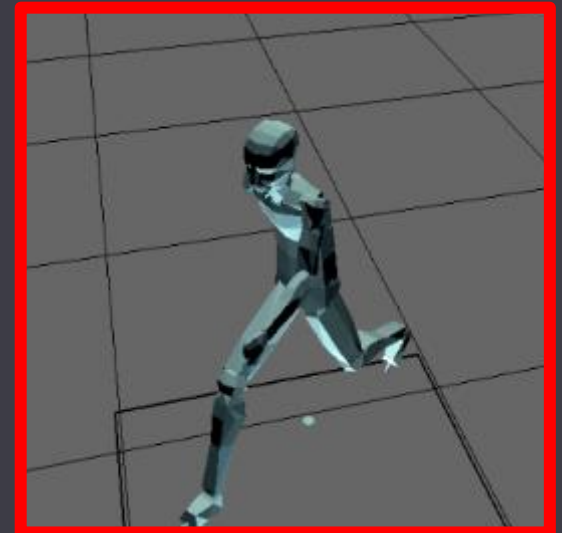
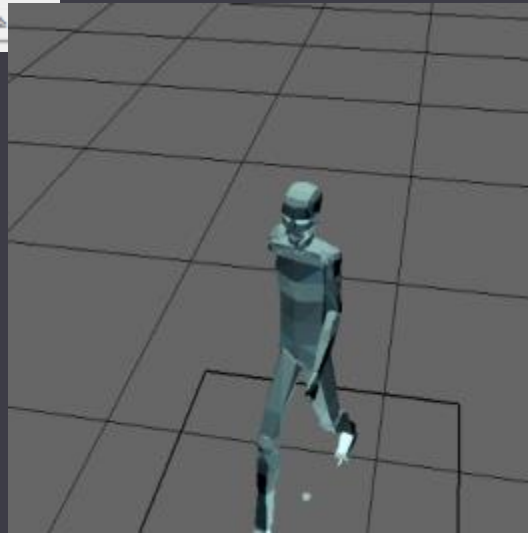
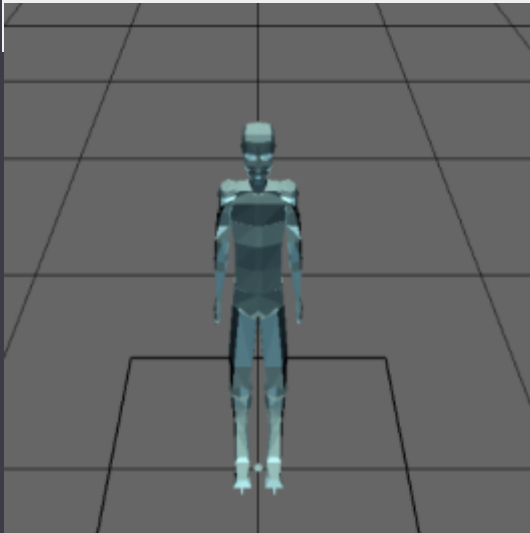
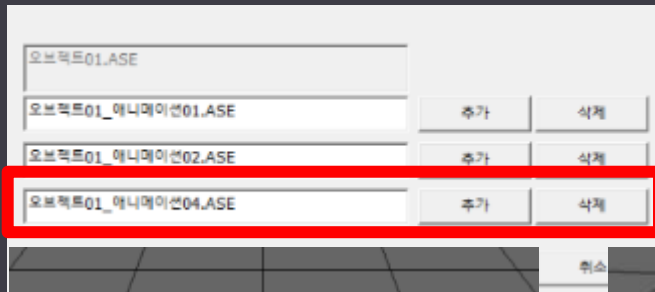
1번 모션은 서있는 모션 ( 숨쉬는 모션) 이다.

### 3. 게임 엔진



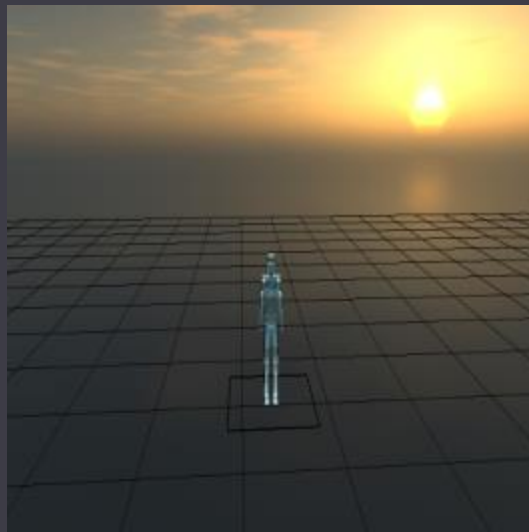
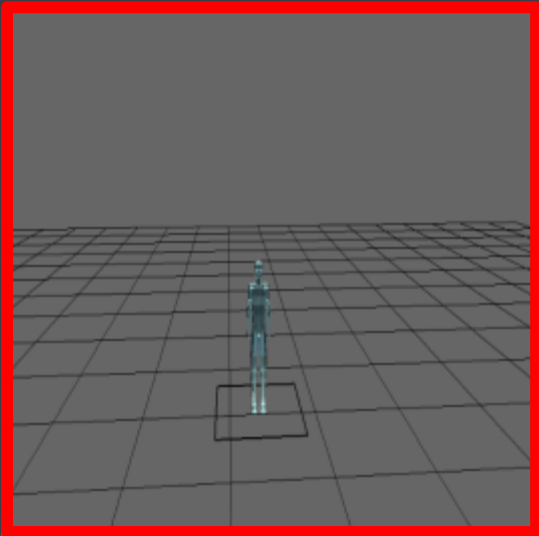
2번은 걷기 모션이다. 키보드 방향키로 움직인다.

### 3. 게임 엔진



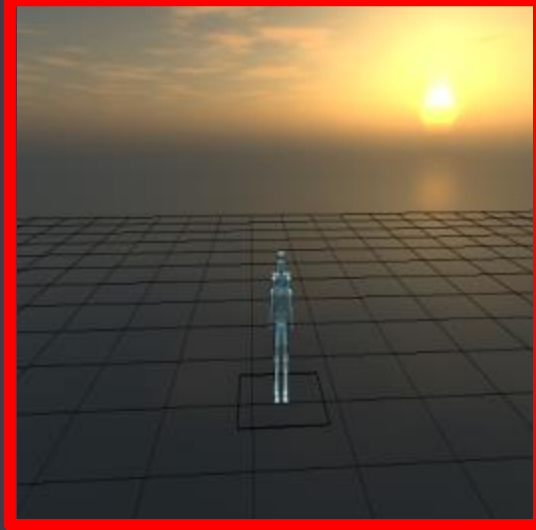
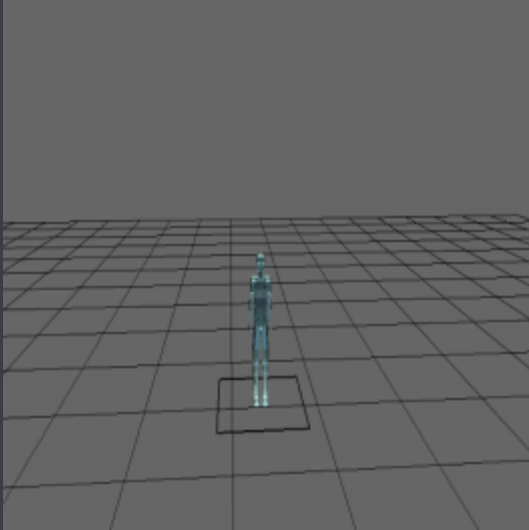
3번은 뛰기 모션이다. Ctrl를 누르면 뛰기로 변경 할 수 있다.

### 3. 게임 엔진



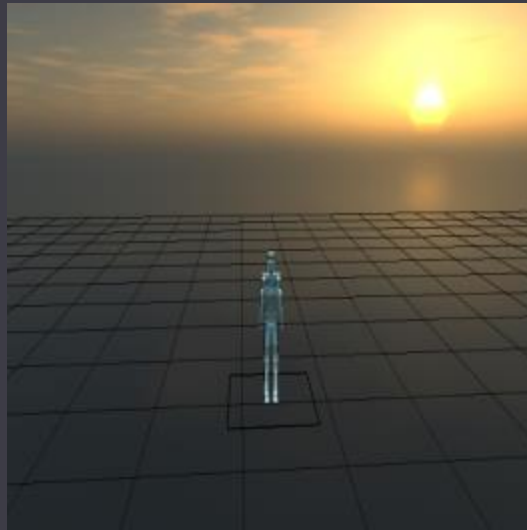
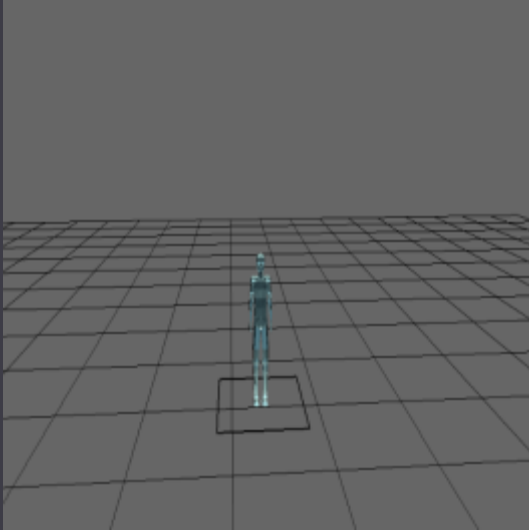
스카이 박스가 없는 상태이다.

### 3. 게임 엔진



2번을 누르면 스카이 박스가 생성된다.

### 3. 게임 엔진



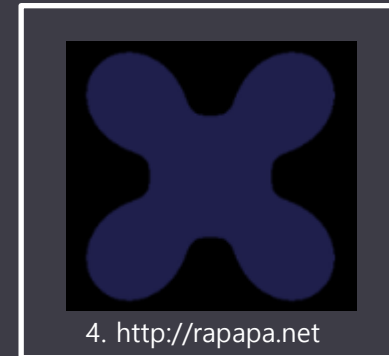
3번을 누르면 좌표계(그리드)가 사라진다.

### 3. 게임 엔진

#### 3.2 구현 내용

3DsMax 에서 제작한 오브젝트를 ASE ( Ascii Export )파일로 출력 한다. ASE파일은 3D 데이터가 텍스트 형식으로 저장되어 있다. 크게 재질, 노드, 애니메이션 세 부분으로 나눈다.

```
MATERIAL_LIST {  
  *MATERIAL_COUNT 1  
  *MATERIAL 0 {  
    *MATERIAL_NAME "01 - Default"  
    *MATERIAL_CLASS "Poutress"  
    *MATERIAL_AMBIENT 0.0000 0.0000 0.0000  
    *MATERIAL_DIFFUSE 0.5020 0.5765 0.5255  
    *MATERIAL_SPECULAR 0.3098 0.5412 0.6392  
    *MATERIAL_SHINE 0.5400  
    *MATERIAL_SHINESTRENGTH 16.2900  
    *MATERIAL_TRANSPARENCY 0.0000  
    *MATERIAL_WIRESIZE 1.0000  
  }  
}
```

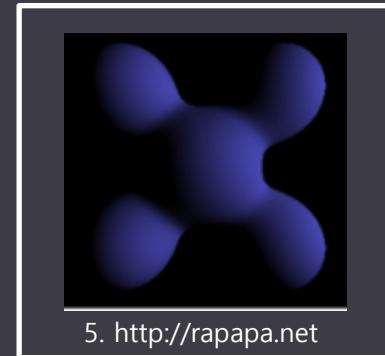


해당 부분은 재질 부분이다.

Ambient 주변광 이다. 특정한 방향 없이 주변을 덮고 있는 빛이다. 어느 쪽으로 회전하던 간에 일정한 밝기 와 색상으로 표현된다.

### 3. 게임 엔진

```
MATERIAL_LIST {  
  *MATERIAL_COUNT 1  
  *MATERIAL 0 {  
    *MATERIAL_NAME "01 - Default"  
    *MATERIAL_CLASS "Raytrace"  
    *MATERIAL_AMBIENT 0.0000 0.0000 0.0000  
    *MATERIAL_DIFFUSE 0.5020 0.5765 0.5255  
    *MATERIAL_SPECULAR 0.3098 0.5412 0.6392  
    *MATERIAL_SHINE 0.5400  
    *MATERIAL_SHINESTRENGTH 16.2900  
    *MATERIAL_TRANSPARENCY 0.0000  
    *MATERIAL_WIRESIZE 1.0000  
  }  
}
```



Diffuse는 분산광 이다. 일정한 방향으로 들어와서 물체의 표면에 여러 방향으로 분산되는 빛이다. 빛을 받는 부분은 밝게 빛나고 그렇지 않는 부분은 빛을 받는 부분 보다 빛을 적게 받는다.



### 3. 게임 엔진

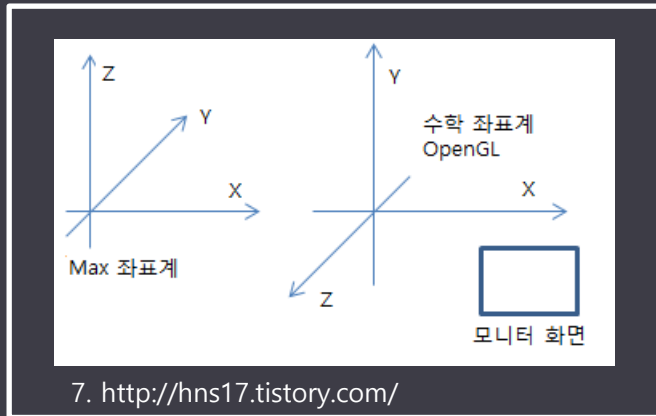
```
MATERIAL_LIST {  
  *MATERIAL_COUNT 1  
  *MATERIAL 0 {  
    *MATERIAL_NAME "01 - Default"  
    *MATERIAL_CLASS "Raytrace"  
    *MATERIAL_AMBIENT 0.0000 0.0000 0.0000  
    *MATERIAL_DIFFUSE 0.5020 0.5765 0.5255  
    *MATERIAL_SPECULAR 0.3098 0.5412 0.6392  
    *MATERIAL_SHINE 0.3400  
    *MATERIAL_SHINESTRENGTH 16.2900  
    *MATERIAL_TRANSPARENCY 0.0000  
    *MATERIAL_WIRESIZE 1.0000  
  }  
}
```



Specular는 반사광 이다. 특정한 방향으로 들어와 한 방향으로 완전히 반사되는 빛이다. 강한 반사광은 물체 표면에 밝은 점을 비춘다.

### 3. 게임 엔진

노드의 CTM 부분이다. 월드 CTM으로 저장되어 있고 3DsMAX의 좌표계는 오른손 좌표계를 사용하지만 Z축이 위를 향하고 있기 때문에 OpenGL에 맞게 좌표축을 변경해야 한다.



```
*GEOMOBJECT {
  *NODE_NAME "Bip001"
  *NODE_TM {
    *NODE_NAME "Bip001"
    *INHERIT_POS 0 0 0
    *INHERIT_ROT 0 0 0
    *INHERIT_SCL 1 1 1
    *TM_ROW0 0.0000 -1.0000 0.0000
    *TM_ROW1 1.0000 0.0000 0.0000
    *TM_ROW2 0.0000 0.0000 1.0000
    *TM_ROW3 -0.0000 0.0000 5.8608
    *TM_POS -0.0000 0.0000 5.8608
    *TM_ROTAXIS -0.0000 -0.0000 1.0000
    *TM_ROTANGLE 1.5708
    *TM_SCALE 1.0000 1.0000 1.0000
    *TM_SCALEAXIS 0.0000 0.0000 0.0000
    *TM_SCALEAXISANG 0.0000
  }
}
```


TM_ROW0	0.0000	-1.0000	0.0000
TM_ROW1	1.0000	0.0000	0.0000
TM_ROW2	0.0000	0.0000	1.0000
TM_ROW3	-0.0000	0.0000	5.8608

변환 :  $X=X$  ,  $Y=Z$  ,  $Z=-Y$

이와 같이 변환 한다.

### 3. 게임 엔진

노드의 버텍스와 페이스 부분이다. 마찬가지로 축 변환을 해야한다.




H_VERTEX_LIST				
*MESH_VERTEX	0	0.0000	-0.5046	5.8924
*MESH_VERTEX	1	0.0000	-0.5160	6.3993
*MESH_VERTEX	2	-0.0000	0.6617	6.0757
*MESH_VERTEX	3	0.0000	0.5166	6.3993
*MESH_VERTEX	4	0.1041	0.0040	5.3935
*MESH_VERTEX	5	0.2918	-0.4175	5.6332
*MESH_VERTEX	6	0.4186	0.5953	5.7472
*MESH_VERTEX	7	0.7735	-0.2839	5.8996
*MESH_VERTEX	8	0.7200	-0.2678	6.4204
*MESH_VERTEX	9	0.7200	0.2684	6.4204
*MESH_VERTEX	10	0.6726	0.4767	6.0829
*MESH_VERTEX	11	0.6927	0.0485	5.6858

*MESH_FACE	0:	A:	7	B:	1	C:	0
*MESH_FACE	1:	A:	1	B:	7	C:	8
*MESH_FACE	2:	A:	2	B:	9	C:	10
*MESH_FACE	3:	A:	9	B:	2	C:	3
*MESH_FACE	4:	A:	10	B:	8	C:	7
*MESH_FACE	5:	A:	8	B:	10	C:	9
*MESH_FACE	6:	A:	4	B:	11	C:	5
*MESH_FACE	7:	A:	4	B:	6	C:	11
*MESH_FACE	8:	A:	5	B:	11	C:	7
*MESH_FACE	9:	A:	10	B:	11	C:	6
*MESH_FACE	10:	A:	7	B:	11	C:	10


0번 페이스의 A : 7, B : 1, C : 0 의 뜻은 메쉬 형태로 A에는 7번 버텍스, B에는 1번 버텍스, C에는 0번 버텍스로 이루어진 메쉬이다.

### 3. 게임 엔진

애니메이션의 이동, 회전 애니메이션 부분이다. 마찬가지로 좌표 변환을 해야 한다.



```
*NODE_NAME "Bip001"
*CONTROL_POS_TRACK {
  *CONTROL_POS_SAMPLE 0 -0.0000 0.0000 5.8608
  *CONTROL_POS_SAMPLE 800 -0.0000 0.0000 5.8472
  *CONTROL_POS_SAMPLE 1600 -0.0000 0.0000 5.8322
  *CONTROL_POS_SAMPLE 2400 -0.0000 0.0000 5.8200
  *CONTROL_POS_SAMPLE 3200 -0.0000 0.0000 5.8150
  *CONTROL_POS_SAMPLE 4000 -0.0000 0.0000 5.8200
  *CONTROL_POS_SAMPLE 4800 -0.0000 0.0000 5.8322
  *CONTROL_POS_SAMPLE 5600 -0.0000 0.0000 5.8472
  *CONTROL_POS_SAMPLE 6400 -0.0000 0.0000 5.8608
```



```
*CONTROL_ROT_TRACK {
  *CONTROL_ROT_SAMPLE 800 0.0000 0.0000 -1.0000 0.1350
  *CONTROL_ROT_SAMPLE 1600 -0.0000 0.0000 -1.0000 0.0914
  *CONTROL_ROT_SAMPLE 2400 0.0000 -0.0000 1.0000 0.0516
  *CONTROL_ROT_SAMPLE 3200 -0.0000 -0.0000 1.0000 0.0712
  *CONTROL_ROT_SAMPLE 4000 -0.0000 0.0000 -1.0000 0.0383
  *CONTROL_ROT_SAMPLE 4800 0.0000 0.0000 -1.0000 0.0319
  *CONTROL_ROT_SAMPLE 5600 -0.0000 -0.0000 1.0000 0.0717
  *CONTROL_ROT_SAMPLE 6400 -0.0000 -0.0000 1.0000 0.1022
```

회전 애니메이션은  $x, y, z, w$ 로 이루어져 있다.  $x, y, z$  축 값과  $w$ 의 라디안 값으로 이루어져 있다. 애니메이션을 구현 할 때 보통 게임에선 축 회전 행렬 3개를 사용하기 보다는 사원수(쿼터니언)을 사용한다. 행렬에 비해 연산속도가 빠르고 메모리가 적기 때문에 사용한다.

행렬 – float 16개 , 곱셈  $16 * 16$   
사원수 – float 4개 , float 곱셈 16

### 3. 게임 엔진

회전 애니메이션 정보를 가지고 쿼터니언 값으로 변환 한다.

사원수.x =  $\sin(w/2.0) * x$ ;  
사원수.y =  $\sin(w/2.0) * y$ ;  
사원수.z =  $\sin(w/2.0) * z$ ;  
사원수.w =  $\cos(w/2.0)$ ;

```
vec4 AParser::quaternion(vec4 &a){  
    vec4 b = { 0.f, 0.f, 0.f, 1.f };  
    b[0] = cosf(a[0] / 2.0f); // w  
    b[1] = sinf(a[0] / 2.0f) * a[1]; // x  
    b[2] = sinf(a[0] / 2.0f) * a[2]; // y  
    b[3] = sinf(a[0] / 2.0f) * a[3]; // z  
    return b;  
}
```

회전 애니메이션은 중첩된 값을 구해야 한다. Y축으로 10도를 회전하고 20도를 회전하면 총 회전 각도는 20도가 아닌 30도이다. 중첩된 값을 얻기 위해 사원수 곱을 한다.

사원수.x'' = ( Q.x \* P.x ) - ( Q.y \* P.y ) - ( Q.z \* P.z ) - ( Q.w \* P.w);  
사원수.y'' = ( Q.x \* P.y ) - ( Q.y \* P.x ) - ( Q.z \* P.w ) - ( Q.w \* P.z);  
사원수.z'' = ( Q.x \* P.z ) - ( Q.y \* P.w ) - ( Q.z \* P.x ) - ( Q.w \* P.y);  
사원수.w'' = ( Q.x \* P.w ) - ( Q.y \* P.z ) - ( Q.z \* P.y ) - ( Q.w \* P.x);

```
vec4 AParser::quaternionMultiply(vec4 &a, vec4 &b){  
    vec4 c = { 0.f, 0.f, 0.f, 1.f };  
    c[0] = (b[0] * a[0]) - (b[1] * a[1]) - (b[2] * a[2]) - (b[3] * a[3]);  
    c[1] = (b[0] * a[1]) + (b[1] * a[0]) + (b[2] * a[3]) - (b[3] * a[2]);  
    c[2] = (b[0] * a[2]) - (b[1] * a[3]) + (b[2] * a[0]) + (b[3] * a[1]);  
    c[3] = (b[0] * a[3]) + (b[1] * a[2]) - (b[2] * a[1]) + (b[3] * a[0]);  
    return c;  
}
```

누적된 사원수를 행렬로 변환한다.

```
rot[0][0] = 1 - (2 * pow(r[i].rot[2], 2)) - (2 * pow(r[i].rot[3], 2));
rot[1][0] = (2 * (r[i].rot[1] * r[i].rot[2])) + (2 * (r[i].rot[0] * r[i].rot[3]));
rot[2][0] = (2 * (r[i].rot[1] * r[i].rot[3])) - (2 * (r[i].rot[0] * r[i].rot[2]));
rot[3][0] = 0;

rot[0][1] = (2 * (r[i].rot[1] * r[i].rot[2])) - (2 * (r[i].rot[0] * r[i].rot[3]));
rot[1][1] = 1 - (2 * pow(r[i].rot[1], 2)) - (2 * pow(r[i].rot[3], 2));
rot[2][1] = (2 * (r[i].rot[2] * r[i].rot[3])) + (2 * (r[i].rot[0] * r[i].rot[1]));
rot[3][1] = 0;

rot[0][2] = (2 * (r[i].rot[1] * r[i].rot[3])) + (2 * (r[i].rot[0] * r[i].rot[2]));
rot[1][2] = (2 * (r[i].rot[2] * r[i].rot[3])) - (2 * (r[i].rot[0] * r[i].rot[1]));
rot[2][2] = 1 - (2 * pow(r[i].rot[1], 2)) - (2 * pow(r[i].rot[2], 2));
rot[3][2] = 0;

rot[0][3] = 0;
rot[1][3] = 0;
rot[2][3] = 0;
rot[3][3] = 1;
```

$$\begin{pmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) & 0 \\ 2(q_x q_y + q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_w q_x) & 0 \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & 1 - 2(q_x^2 + q_y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

### 3. 게임 엔진

누적된 사원수를 행렬로 변환한다.

*CONTROL_ROT_SAMPLE 800	0.0000	0.0000	-1.0000	0.1350
*CONTROL_ROT_SAMPLE 1600	-0.0000	0.0000	-1.0000	0.0914
*CONTROL_ROT_SAMPLE 2400	0.0000	-0.0000	1.0000	0.0516
*CONTROL_ROT_SAMPLE 3200	-0.0000	-0.0000	1.0000	0.0712
*CONTROL_ROT_SAMPLE 4000	-0.0000	0.0000	-1.0000	0.0383
*CONTROL_ROT_SAMPLE 4800	0.0000	0.0000	-1.0000	0.0319
*CONTROL_ROT_SAMPLE 5600	-0.0000	-0.0000	1.0000	0.0717
*CONTROL_ROT_SAMPLE 6400	-0.0000	-0.0000	1.0000	0.1022

두 프레임의 회전 행렬을 구한다고 하면, 현재 프레임 ( 빨간색 )의 사원수 값을 구한다.

w : 0.1350 , x : 0.0000 , y : -1.0000 , z : 0.0000

사원수.x =  $\sin(0.1350/2.0) * 0.0000 = 0$

사원수.y =  $\sin(0.1350/2.0) * -1.0000 = -0.0674$

사원수.z =  $\sin(0.1350/2.0) * 0.0000 = 0$

사원수.w =  $\cos(0.1350/2.0) = 0.9977$

### 3. 게임 엔진

현재 프레임의 사원수 값은

$$\text{사원수.x} = \sin(0.1350/2.0) * 0.0000 = 0$$

$$\text{사원수.y} = \sin(0.1350/2.0) * -1.0000 = -0.0674$$

$$\text{사원수.z} = \sin(0.1350/2.0) * 0.0000 = 0$$

$$\text{사원수.w} = \cos(0.1350/2.0) = 0.9977$$

1번 프레임은 중첩된 값이 아니기 때문에 그대로 행렬 변환을 한다.

$$\begin{bmatrix} 1-2(-0.0674_y^2+0_z^2) & 2(0_x-0.0674_y-0.9977_w0_z) & 2(0_x0_z-0.9977_w-0.0674_y) & 0 \\ 2(0_x0_y+0.9977_w0_z) & 1-2(0_x^2+0_z^2) & 2(-0.0674_y0_z-0.9977_w0_x) & 0 \\ 2(0_x0_z+0.9977_w-0.0674_y) & 2(-0.0674_y0_z+0.9977_w0_x) & 1-2(0_x^2+-0.0674_y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.9909 & 0 & 0.1344 & 0 \\ 0 & 1 & 0 & 0 \\ -0.1344 & 0 & 0.9909 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



### 3. 게임 엔진

```
-----
*CONTROL_ROT_SAMPLE 800 0.0000 0.0000 -1.0000 0.1350
*CONTROL_ROT_SAMPLE 1600 -0.0000 0.0000 -1.0000 0.0914
*CONTROL_ROT_SAMPLE 2400 0.0000 -0.0000 1.0000 0.0516
*CONTROL_ROT_SAMPLE 3200 -0.0000 -0.0000 1.0000 0.0712
*CONTROL_ROT_SAMPLE 4000 -0.0000 0.0000 -1.0000 0.0383
*CONTROL_ROT_SAMPLE 4800 0.0000 0.0000 -1.0000 0.0319
*CONTROL_ROT_SAMPLE 5600 -0.0000 -0.0000 1.0000 0.0717
*CONTROL_ROT_SAMPLE 6400 -0.0000 -0.0000 1.0000 0.1022
```

다음 프레임( 노란색 ) 사원수 값을 구한다.

w : 0.0914 , x : 0.0000 , y : -1.0000 , z : 0.0000

$$\text{사원수.x} = \sin(0.0914 / 2.0) * 0.0000 = 0$$

$$\text{사원수.y} = \sin(0.0914 / 2.0) * -1.0000 = -0.0456$$

$$\text{사원수.z} = \sin(0.0914 / 2.0) * 0.0000 = 0$$

$$\text{사원수.w} = \cos(0.0914 / 2.0) = 0.9989$$

### 3. 게임 엔진

1번 프레임 사원수에 중첩하여 사용한다.

1번 프레임 사원수

$$\text{사원수}.x = \sin(0.1350/2.0) * 0.0000 = 0$$

$$\text{사원수}.y = \sin(0.1350/2.0) * -1.0000 = -0.0674$$

$$\text{사원수}.z = \sin(0.1350/2.0) * 0.0000 = 0$$

$$\text{사원수}.w = \cos(0.1350/2.0) = 0.9977$$

2번 프레임 사원수

$$\text{사원수}.x = \sin(0.0914 /2.0) * 0.0000 = 0$$

$$\text{사원수}.y = \sin(0.0914 /2.0) * -1.0000 = -0.0456$$

$$\text{사원수}.z = \sin(0.0914 /2.0) * 0.0000 = 0$$

$$\text{사원수}.w = \cos(0.0914 /2.0) = 0.9989$$

중첩된 결과 ( 2번 프레임 사원수 )

$$\text{사원수}.x'' = ( 0.x * 0.x ) - (-0.0674.y * -0.0456.y ) - (0.z * 0.z ) - (0.9977.w * 0.9989.w) = -0.9996$$

$$\text{사원수}.y'' = ( 0.x * -0.0456.y ) - (0.0674.y * 0.x ) - (0.z * 0.9989.w) - (0.9977.w * 0.z) = 0$$

$$\text{사원수}.z'' = ( 0.x * 0.z ) - ((-0.0674.y * 0.9989.w) - (0.z * 0.x ) - (0.9977.w * -0.0456.y ) = 0.1128$$

$$\text{사원수}.w'' = ( 0.x * 0.9989.w) - ((-0.0674.y * 0.z ) - (0.z * -0.0456.y ) - (0.9977.w * 0.x) = 0$$

### 3. 게임 엔진

2번 프레임의 행렬을 구한다.

$$\text{사원수}.x'' = (0.x * 0.x) - (-0.0674.y * -0.0456.y) - (0.z * 0.z) - (0.9977.w * 0.9989.w) = -0.9996$$

$$\text{사원수}.y'' = (0.x * -0.0456.y) - (0.0674.y * 0.x) - (0.z * 0.9989.w) - (0.9977.w * 0.z) = 0$$

$$\text{사원수}.z'' = (0.x * 0.z) - ((-0.0674.y * 0.9989.w) - (0.z * 0.x) - (0.9977.w * -0.0456.y) = 0.1128$$

$$\text{사원수}.w'' = (0.x * 0.9989.w) - ((-0.0674.y * 0.z) - (0.z * -0.0456.y) - (0.9977.w * 0.x) = 0$$

$$\begin{bmatrix} 1-2(0_y^2+0.1128_z^2) & 2(-0.9996_x0_y-0_w0.1128_z) & 2(-0.9996_x0.1128_z-0_w0_y) & 0 \\ 2(-0.9996_x0_y+0_w0_z) & 1-2(-0.9996_x^2+0.1128_z^2) & 2(0_y0_z-0_w0_x) & 0 \\ 2(-0.9996_x0.1128_z+0_w0_y) & 2(0_y0.1128_z+0_w-0.9996_x) & 1-2(-0.9996_x^2+0_y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.9745 & 0 & -0.2255 & 0 \\ 0 & -0.9856 & 0 & 0 \\ -0.2255 & 0 & -0.9984 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

이와 같은 방법으로 회전 행렬을 구한다.

### 3. 게임 엔진

프레임 간의 보간을 한다.

```
*NODE_NAME "BIP001"
*CONTROL_POS_TRACK {
  *CONTROL_POS_SAMPLE 0 -0.0000 0.0000 5.8608
  *CONTROL_POS_SAMPLE 800 -0.0000 0.0000 5.8472
  *CONTROL_POS_SAMPLE 1600 -0.0000 0.0000 5.8322
  *CONTROL_POS_SAMPLE 2400 -0.0000 0.0000 5.8200
  *CONTROL_POS_SAMPLE 3200 -0.0000 0.0000 5.8150
  *CONTROL_POS_SAMPLE 4000 -0.0000 0.0000 5.8200
  *CONTROL_POS_SAMPLE 4800 -0.0000 0.0000 5.8322
  *CONTROL_POS_SAMPLE 5600 -0.0000 0.0000 5.8472
  *CONTROL_POS_SAMPLE 6400 -0.0000 0.0000 5.8608
```

```
*CONTROL_ROT_TRACK {
  *CONTROL_ROT_SAMPLE 800 0.0000 0.0000 -1.0000 0.1350
  *CONTROL_ROT_SAMPLE 1600 -0.0000 0.0000 -1.0000 0.0914
  *CONTROL_ROT_SAMPLE 2400 0.0000 -0.0000 1.0000 0.0516
  *CONTROL_ROT_SAMPLE 3200 -0.0000 -0.0000 1.0000 0.0712
  *CONTROL_ROT_SAMPLE 4000 -0.0000 0.0000 -1.0000 0.0383
  *CONTROL_ROT_SAMPLE 4800 0.0000 0.0000 -1.0000 0.0319
  *CONTROL_ROT_SAMPLE 5600 -0.0000 -0.0000 1.0000 0.0717
  *CONTROL_ROT_SAMPLE 6400 -0.0000 -0.0000 1.0000 0.1022
```

160 tick = 1프레임 이며 5프레임 단위로 애니메이션 정보가 저장되어 있다. 사이 값인 1~4 프레임의 값을 보간 한다.

보간 방법으로는 선형 보간을 사용한다.

```
t = 0.0;
while () {
  0번 프레임.x * (1.0 - t) + ( 1번 프레임.x * t )
  t += 0.05;
  if ( t == 1.0 ) break; }
```

```
rot[0] = (r[i].rot[0] * (1.0 - tc)) + (fos * (r[i + 1].rot[0] * tc));
rot[1] = (r[i].rot[1] * (1.0 - tc)) + (fos * (r[i + 1].rot[1] * tc));
rot[2] = (r[i].rot[2] * (1.0 - tc)) + (fos * (r[i + 1].rot[2] * tc));
rot[3] = (r[i].rot[3] * (1.0 - tc)) + (fos * (r[i + 1].rot[3] * tc));
```

### 3. 게임 엔진

프레임 간의 보간을 한다.

```
*NODE_NAME "BIP001"  
*CONTROL_POS_TRACK {  
  *CONTROL_POS_SAMPLE 0 -0.0000 0.0000 5.8608  
  *CONTROL_POS_SAMPLE 800 -0.0000 0.0000 5.8472  
  *CONTROL_POS_SAMPLE 1600 -0.0000 0.0000 5.8322  
  *CONTROL_POS_SAMPLE 2400 -0.0000 0.0000 5.8200  
  *CONTROL_POS_SAMPLE 3200 -0.0000 0.0000 5.8150  
  *CONTROL_POS_SAMPLE 4000 -0.0000 0.0000 5.8200  
  *CONTROL_POS_SAMPLE 4800 -0.0000 0.0000 5.8322  
  *CONTROL_POS_SAMPLE 5600 -0.0000 0.0000 5.8472  
  *CONTROL_POS_SAMPLE 6400 -0.0000 0.0000 5.8608  
}
```



0 tick( 0번 프레임)과 800 tick ( 5번 프레임)의 사이의 1~4 프레임 값을 구한다면,

$t = 0.2$  // 프레임 사이의 개수를 구하기 위한 변수.

1번 프레임. $z = 5.8608(0 \text{ tick}) * (1.0 - t) + ( 5.8472( 800 \text{ tick} ) * t) = 5.8580$

$t += t$  //  $t = 0.4$

2번 프레임. $x = 5.8608(0 \text{ tick}) * (1.0 - t) + ( 5.8472( 800 \text{ tick} ) * t) = 5.8553$

$t == 1.0 \text{ break ;}$  //  $t$ 가 1.0과 같으면 정지 , 총 4번이 반복되어 계산된다.

### 3. 게임 엔진

위와 같은 정보들로 3D 오브젝트를 그릴 수 있다.

먼저, 좌표를 로컬 좌표로 변환 한다.

$$\text{로컬 정점} = (\text{자신의 CTM})^{-1} * \text{월드 정점}$$

두 번째로 CTM을 로컬 행렬로 변환한다.

$$\text{로컬 행렬} = (\text{부모 CTM})^{-1} * (\text{자신의 CTM})$$

세 번째로 애니메이션 행렬을 만든다.

$$\text{애니메이션 행렬} = (\text{이동 행렬}) * (\text{회전 행렬}) * (\text{크기 행렬})$$

마지막으로 최종 행렬을 만든다.

$$\text{최종 행렬} = \text{로컬 행렬} * \text{애니메이션 행렬} * \text{부모의 최종 행렬}$$

$$\text{월드 좌표} = \text{최종 행렬} * \text{로컬 좌표}$$

### 3. 게임 엔진

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ 월드 CTM}$$

$$[0 \quad 0 \quad 6.0714 \quad 1] \text{ 월드 버텍스}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5.8608 & 1 \end{bmatrix} \text{ 이동 행렬}$$

세가지로 최종 행렬을 구한다면,

로컬 정점 = (자신의 CTM)<sup>-1</sup> \* 월드 정점

$$[0 \quad 0 \quad 6.0714 \quad 1] \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [0 \quad 0 \quad 6.071400 \quad 1]$$

```
for (int i = 0; i < msize; i++){
    for (int j = 0; j < mesh->node[i]->vertex.size(); j++){
        mesh->node[i]->vertex[j].v = mesh->node[i]->ctm.inctm * mesh->node[i]->vertex[j].v;
    }
}
```

로컬 행렬 = (부모 CTM)<sup>-1</sup> \* (자신의 CTM)

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5.8608 & 1 \end{bmatrix}$$
 부모가 없기 때문에 자신이 로컬 행렬이다.

```
if (parentid == -1) ctm.localctm = ctm.ctm;  
else if (parentid > -1) ctm.localctm = mesh->node[parentid]->ctm.inctm * ctm.ctm;
```

부모가 없으면 자신의 CTM이 로컬 CTM이 되고  
부모가 있다면 부모 행렬을 곱하여 로컬 행렬을 구한다.



애니 행렬 = (이동 행렬) \* (회전 행렬) \* (크기 행렬)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5.8608 & 1 \end{bmatrix}$$

이동 행렬만 존재하기 때문에 이동 행렬이 애니 행렬이다.

```
mat4 res = (p[i].pos * r[j].rot);
```

### 3. 게임 엔진

최종 행렬 = 로컬 행렬 \* 애니 행렬 \* 부모의 최종 행렬

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5.8608 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5.8608 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5.860800 & 1 \end{bmatrix}$$

부모 행렬이 없기 때문에 로컬 행렬과 애니메이션 행렬만 연산한다.

```
c = animation->ani[parentid]->last[i].result * s[i].result;
```

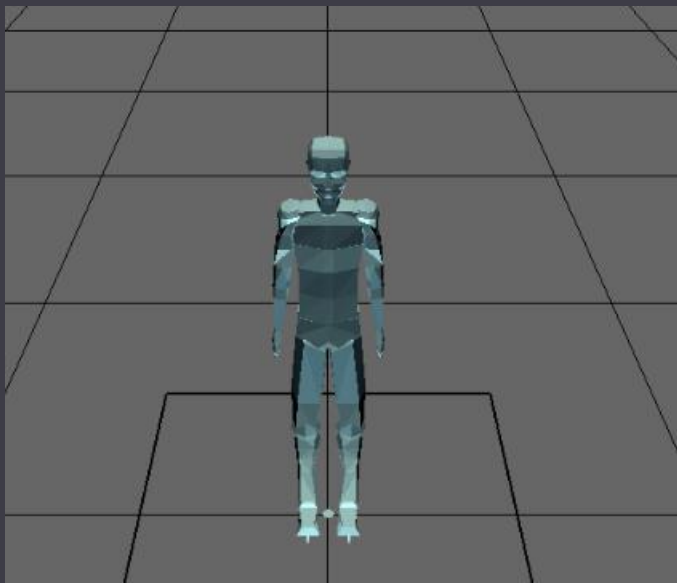
부모가 있고, 없고, 애니메이션이 있고, 없고를 판단해서 상황 별로 연산을 하도록 한다.

### 3. 게임 엔진

월드 좌표 = 최종 행렬 \* 로컬 좌표

```
v1 = pos_ * (this->animation[index].ani[j]->last[frame_s].result) * (this->mesh->node[j]->vertex[faceA].v);  
v2 = pos_ * (this->animation[index].ani[j]->last[frame_s].result) * (this->mesh->node[j]->vertex[faceB].v);  
v3 = pos_ * (this->animation[index].ani[j]->last[frame_s].result) * (this->mesh->node[j]->vertex[faceC].v);
```

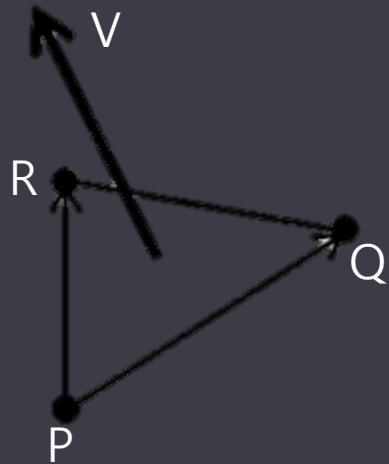
오브젝트 출력



$$\begin{bmatrix} 0 & 0 & 6.071400 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5.860800 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0 & 11.9342 & 1 \end{bmatrix}$$

### 3. 게임 엔진

오브젝트에 조명을 주기 위해 노말 값을 설정해야 한다. ASE파일에 노말 값이 명시 되어 있긴 하지만 좋은 외관을 보여주지 못하기 때문에 직접 계산하여 노말 값을 구한다. 세 점으로 만들어지는 평면에 법선 벡터( 외적 )을 구한다.



$$\text{벡터 } PQ = Q - P = (2,4,6) - (0,0,0) = (2,4,6)$$

$$\text{벡터 } PR = R - P = (-1,2,7) - (0,0,0) = (-1,2,7)$$

$$V = PQ \times PR$$

$$V = V.x = ( PQ.y * PR.z ) - ( PQ.z * PR.y )$$

$$V.y = ( PQ.z * PR.x ) - ( PQ.x * PR.z )$$

$$V.z = ( PQ.x * PR.y ) - ( PQ.y * PR.x )$$

$$V = V.x = ( 4 * 7 - 6 * 2 )$$

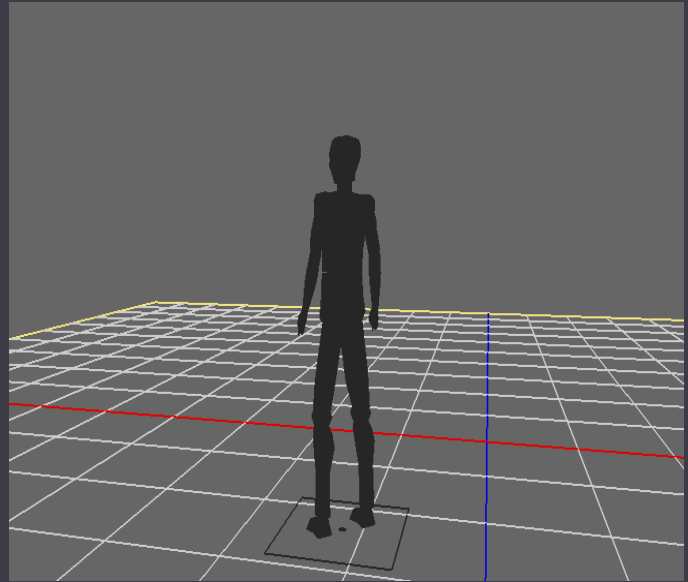
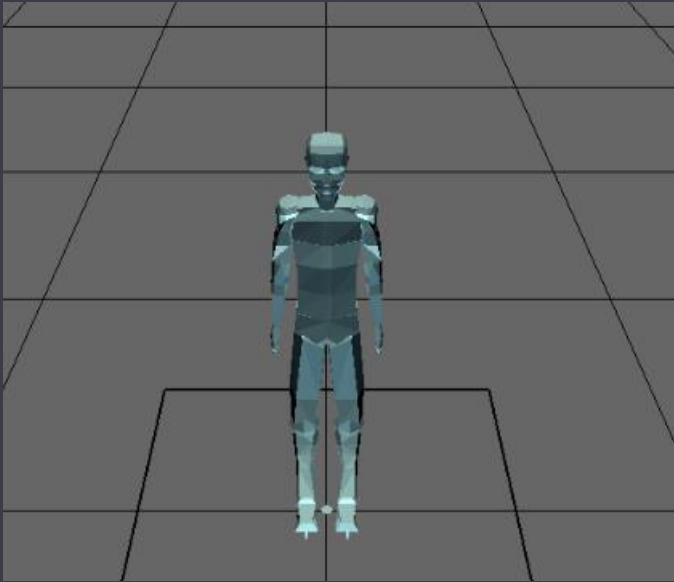
$$V.y = ( 6 * (-1) - 2 * 7 )$$

$$V.z = ( 2 * 2 - 4 * (-1) )$$

$$= (16, -20, 8)$$

```
ax = v1[0] - v2[0];  
ay = v1[1] - v2[1];  
az = v1[2] - v2[2];  
  
bx = v3[0] - v2[0];  
by = v3[1] - v2[1];  
bz = v3[2] - v2[2];  
  
nx = (ay * bz) - (az * by);  
ny = (az * bx) - (ax * bz);  
nz = (ax * by) - (ay * bx);  
f1 = 1 / (sqrt(nx * nx + ny * ny + nz * nz));  
nx = nx * f1;  
ny = ny * f1;  
nz = nz * f1;
```

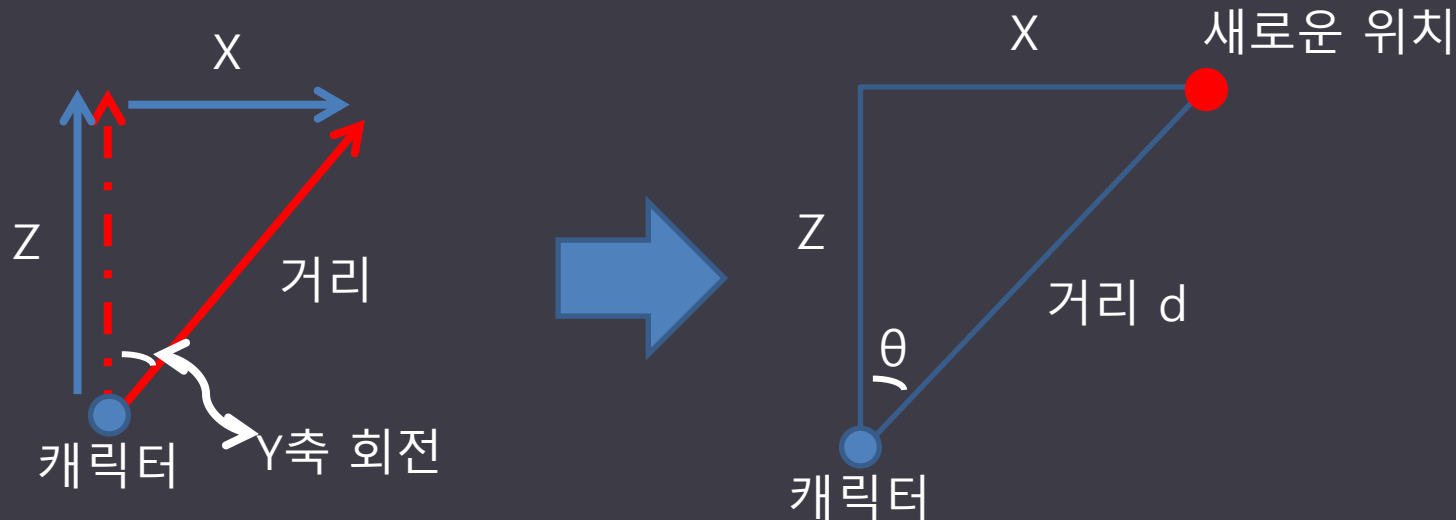
### 3. 게임 엔진



노멀 값 유무 차이다. 노멀 값이 없는 경우 완전 2D같은 이미지를 보여주지만 노멀 값이 있는 경우 입체감을 보여준다.

### 3. 게임 엔진

게임에서 가장 중요한 부분은 이동과 시점이다. 먼저 이동과 방향 구현 내용이다. 이동과 방향은 삼각함수를 사용하여 구현 하였다.



$$X = d * \sin(\theta)$$
$$Z = d * \cos(\theta)$$

```
GLfloat disX = (sin(radians(rotY))) * dist[2];  
GLfloat disZ = (cos(radians(rotY))) * dist[2];  
glTranslatef(disX, dist[1], disZ);
```

### 3. 게임 엔진

카메라 시점은 카메라 자신의 위치와 시점 방향을 지정해야 한다. 카메라 자신의 위치는 현재 캐릭터의 위치 - X, Z축 이다.

**카메라 위치.X = 캐릭터 위치.x - (d \* sin( $\theta$ ))**

**카메라 위치.Z = 캐릭터 위치.z - (d \* cos( $\theta$ ))**

카메라의 시점은 캐릭터를 향한다.

**카메라 시점.x = 캐릭터 위치.x**

**카메라 시점.z = 캐릭터 위치.z**

```
GLfloat disZ = (cos(radians(st.rotation))) * (dm));  
Camera camera((st.pos[0] / 10) - disX, 2, (st.pos[2] / 10) - disZ, (st.pos[0] / 10), dL / 5, (st.pos[2] / 10));
```

### 3. 게임 엔진

```
void CEngin_05Dlg::GLRenderScene(){ // 랜더링 함수

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //현재 버퍼를 비워줌
glLoadIdentity(); // 단위 행렬로 초기화

if (viewtrans == 0)Camera camera(dX, dY, dZ + 3, dX, 0, dZ); // 카메라 시점 1
else if (viewtrans == 1){
GLfloat disZ = ((st.pos[2] / 10) - 5);
Camera camera(0, 2, disZ, st.pos[0] / 10, dL / 5, st.pos[2] / 10); // 카메라 시점2
}
else if (viewtrans == 2){
GLfloat disX = (sin(radians(st.rot[1]))) * ((dW));
GLfloat disZ = (cos(radians(st.rot[1]))) * ((dW));
Camera camera((st.pos[0] / 10) - disX, 2, (st.pos[2] / 10) - disZ, (st.pos[0] / 10), dL / 5, (st.pos[2] / 10));
} // 카메라 시점3

if (girdtrans)Grid grid(0.1f, 1.f);
glColor3f(1.0f, 1.0f, 1.0f);
if ( skytrans) sky->Draw_Skybox(0, 0, 0, 100, 100, 100); // 스카이 박스 출력
glPushMatrix();
glScalef(0.1, 0.1, 0.1);
if (strcmp("W0", mesh->file))st = OBJ->object(frame, dist, rotY, animationindex); // 오브젝트 출력
Box box(st.pos[0], st.pos[2]);
glPopMatrix();
SwapBuffers(m_hdc);
InvalidateRect(NULL, FALSE);
}
```



### 3. 게임 엔진

```
if (s.size() > 0 && parentid == -1){// 애니 O 부모 X
last.assign(s.begin(), s.end()); // 애니메이션 O 부모 X , 최종 행렬 구하기
}
else if (s.size() == 0 && parentid == -1){// 애니 x 부모 x

for (int i = 0; i <= animation->wholeFrame; i++){
last.push_back(ResultMat(c.localctm, (i * animation->unitTick))); // 애니메이션 X 부모 X , 최종 행렬 구하기
}
}
else if (s.size() > 0 && parentid > -1){ // 애니 O 부모 O
mat4 c;

for (int i = 0; i < s.size(); i++){

c = animation->ani[parentid]->last[i].result * s[i].result; // 애니메이션 O 부모 O , 최종 행렬 구하기

last.push_back(ResultMat(c, i));
}
}

else if (s.size() == 0 && parentid > -1){ // 애니 X 부모 O
mat4 d;
for (int i = 0; i <= animation->wholeFrame; i++){
d = animation->ani[parentid]->last[i].result * c.localctm; // 애니메이션 X 부모 O , 최종 행렬 구하기

last.push_back(ResultMat(d, i));
}
```

### 3. 게임 엔진

```
GLint pid = mesh->node[j]->parentid;
if (strcmp("W0", animation[index].file)){
v1 = pos_ * (this->animation[index].ani[j]->last[frame_s].result) * (this->mesh->node[j]->vertex[faceA].v); // 월드 좌표 연산
v2 = pos_ * (this->animation[index].ani[j]->last[frame_s].result) * (this->mesh->node[j]->vertex[faceB].v);
v3 = pos_ * (this->animation[index].ani[j]->last[frame_s].result) * (this->mesh->node[j]->vertex[faceC].v);
}
else {
v1 = pos_ * mesh->node[j]->ctm.ctm * (this->mesh->node[j]->vertex[faceA].v);
v2 = pos_ * mesh->node[j]->ctm.ctm * (this->mesh->node[j]->vertex[faceB].v);
v3 = pos_ * mesh->node[j]->ctm.ctm * (this->mesh->node[j]->vertex[faceC].v);
}

vec3 n = normal(v1, v2, v3); // 노말 값 연산

glShadeModel(GL_FLAT);
glBegin(GL_TRIANGLES);
glNormal3f(n[0], n[1], n[2]);
glVertex3f(v1[0], v1[1], v1[2]);
glVertex3f(v2[0], v2[1], v2[2]);
glVertex3f(v3[0], v3[1], v3[2]);
glEnd();

for (int j = 0; j < msize; j++){
for (int i = 0; i <= this->mesh->wholeframe; i++){
if (strcmp("W0", animation[0].file)) this->animation[0].ani[j]->last[i].result = mR * this->animation[0].ani[j]->last[i].result; // 키보드 회전, 이동 좌표 연산
if (strcmp("W0", animation[1].file)) this->animation[1].ani[j]->last[i].result = mR * this->animation[1].ani[j]->last[i].result;
if (strcmp("W0", animation[2].file)) this->animation[2].ani[j]->last[i].result = mR * this->animation[2].ani[j]->last[i].result;
}
//if (this->animation[index].wholeFrame == 0){
mesh->node[j]->ctm.ctm = mR * mesh->node[j]->ctm.ctm; // 애니메이션이 없는 경우 ctm으로 애니메이션 연산
//}
pos_ = mP * pos_; // 이동 좌표는 나중에 계산 ( 회전을 0,0,0 원점에서 회전 한 뒤 이동 연산 시작 )
}
```

## 4. 실험 및 고찰

오브젝트를 불러오고 움직이고 애니메이션을 추가 할 수 있는 단순한 엔진이지만 직접 구현해 봄으로써 캐릭터가 어떻게 움직이고 애니메이션이 어떻게 돌아가는지, 게임이 어떤 원리로 구현 되는지 알 수 있다.

다른 게임 엔진을 한번 봐야 한다. 특히 렌더링 부분을 봐야 한다. 현재 진행한 프로젝트는 렌더링 속도가 너무 느리다. 렌더링 속도가 느린 이유는 렌더링 함수에 반복문이 많다. 렌더링 함수는 그림을 그리기 위해서 계속 돌아가는데, 그곳에 반복문이 계속 돌아가기 때문에 속도가 늦는 것 같다. 반복문을 줄이기 위해서 최대한 머리를 쓰며 코딩을 했지만 여전히 느리다. 다른 게임들을 보고 어떤 방법으로 렌더링을 하는지 찾아 봐야 한다.

애니메이션 추가에서 CTM 문제가 있다. 현재는 애니메이션을 추가하면 어쩔 수 없이 모든 행렬 ( 메쉬, 애니메이션 )을 초기화 한다. 애니메이션 마다 CTM이 다르기 때문에 중간에 삭제하거나 추가하면 CTM이 엉켜버리기 때문인데, 추가하거나 삭제할 때 마다 모든 행렬을 초기화 하기 보다는 삭제한 행렬이나 추가한 행렬만 초기화 하도록 변경해야 한다. 각각의 애니메이션 행렬을 독립적으로 사용하면 될 것 같긴 한데 아직 구현 하지 못했다.

## 5. 결론 및 향후 과제

아직 전체적으로 프로젝트는 미완성이기 때문에 계속 만들어 가야 한다. 가장 먼저 추가해야 할 부분은 충돌 처리 부분이다. 현재 버전 코드 말고 다른 버전 코드에는 박스도 랜덤으로 생성 되도록 했고 충돌처리를 하도록 구현을 했는데 정확하지가 않다. 방향이 맞지 않는 현상이 발생해서 많이 미흡하다.

충돌처리가 완료 되면 렌더링 속도를 높이도록 하고 박스 대신 다른 오브젝트를 출력할 생각이다. 오브젝트 생성 방법으로는 마우스 드래그로 오브젝트를 그리드 안에 생성하는 방법으로 생성할 생각이다. 현재 제작한 공격 모션으로 오브젝트를 공격하면 오브젝트는 죽는 모션을 하며 사라지는 것을 구현할 생각이다.

아직 미완성 프로젝트이기 때문에 계속 개발할 예정이다.

## 6. References

1. <https://unity3d.com/kr>
2. <https://www.epicgames.com/ko>
3. [www.pearlabyss.com](http://www.pearlabyss.com)
4. <http://rapapa.net>
5. <http://rapapa.net>
6. <http://rapapa.net>
7. <http://hns17.tistory.com/>

- (1) 참고서적 : Interactive Computer Graphics –Edward Angel
- (2) 참고서적 : OpenGL로 배우는 3차원 컴퓨터 그래픽스 -주우식
- (3) 참고서적 : OpenGL로 배우는 3차원 컴퓨터 그래픽스 -주우식
- (4) 참고서적 : OpenGL로 배우는 3차원 컴퓨터 그래픽스 -주우식

Q & A