**2018180020 박재우(클라이언트) 7주차 기록**

1. 모델이 포함한 정보를 저장할 객체를 만들어야 한다. (텍스쳐, 머터리얼, 애니메이션 등등..)
2. 모델이 포함한 정보를 읽을 함수를 만들어야 한다.
3. 모델을 렌더링할 때 사용할 쉐이더를 만들어야 한다.
4. 모델이 가지고 있는 리소스를 쉐이더로 넘겨줘야 한다.

**모델 띄우기 2번**

모델을 불러오기 위해 bin파일을 읽어야 한다. bin파일 안에는 모델의 메쉬, 머터리얼, 본 구조, 스키닝, 애니메이션 등 여러 데이터가 들어있다. 우선 메쉬와 머터리얼을 불러오는 기능을 먼저 만든다.

우선 함수를 만들기 전에 우리가 사용하는 bin파일의 구조를 먼저 알아야 한다.

모델은 계층 구조로 구성 되어있고 “<데이터이름>: ~~~ </데이터이름>“ 처럼 표현되며 이름에 해당하는 데이터가 ~~~ 부분에 기술 되어있다.

예시 )

<Frame>: 12 0 FKRoot\_M

<Transform>: 0 0 -1.972152E-32 0 -1.235894E-30 0 1 1 1 0 -1.078521E-32 0 1

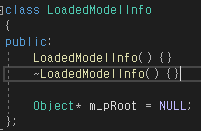
<TransformMatrix>: 1 0 2.157042E-32 0 0 1 0 0 -2.157042E-32 0 1 0 0 0 -1.972152E-32 1

<Children>: 0

</Frame>

이 예시 같은 경우는 Frame의 정보들이 <Frame>:과 </Frame> 사이에 기술 되어있고 Frame의 이름 Transform, 행렬, 자식의 정보를 확인 할 수 있다.

파일의 구조는 크게 Hierarchy와 Animation 두가지로 나뉘어져 있다. 따라서 먼저 파일을 열고 두가지 정보를 나누어 읽는 부분을 만든다. 함수는 Object 클래스의 멤버함수로 만든다.

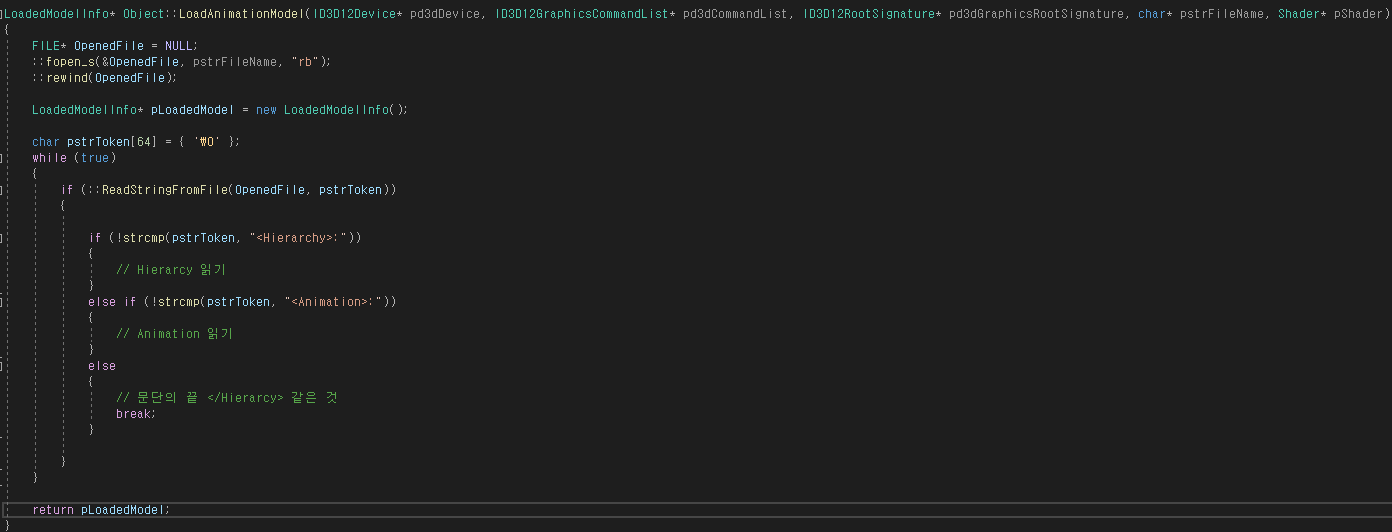


파일을 읽어와 해당 데이터를 임시로 저장할 객체를 만들었다. 루트에 해당하는 객체를 m\_pRoot에 저장하고 이 객체에 자식과 형제들을 붙여 나갈 예정이다. 또 추가로 필요한 데이터가 생긴다면 그 때 추가로 넣어줄 것이다.

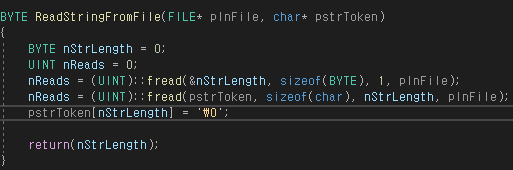


Object 객체에 LoadAnimationModel이라는 함수를 만들었다. 형식은 위에서 만들어준 LoadedModelInfo이고 static으로 만든 이유는 객체 내에서 함수를 부르는 것이 아니라 외부에서 직접 불러서 반환 값을 대입해주는 방식으로 모델을 로드하기 때문이다.

이 함수에서 해주는 일은 <Hieraracy>부분과 <Animation> 부분을 나누어 로드한다. 최종적으로 모두 로드가 된 모델 데이터를 반환한다.



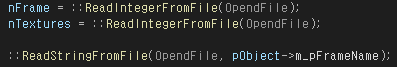
LoadAnimation의 함수는 위와 같이 짰고 파일을 열어 파일 포인터의 위치를 처음으로 설정한다. ReadStringFromFile이란 함수로 한 토큰씩 읽는다.



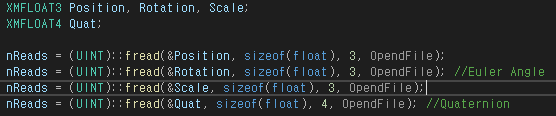
읽은 토큰을 strcmp로 비교해 Hierachy인지 Animation인지 판단하고 각각 맞는 데이터를 불러오는 함수를 부른다. 해당 함수들도 비슷한 구조로 데이터를 읽는다.

다음은 Hierarchy를 읽는 함수를 만든다. Hierarchy 계층 아래엔 Frame이 있고 그 밑에 Transform, TransformMatrix, Mesh, SkinningInfo, Materials, Children 이 있다. Mesh와 SkinningInfo 밑에 또 다른 계층들이 존재하므로 해당 데이터를 읽는 함수를 추가로 만들어 줘야 하고 Children은 밑에 새로운 Frame이 등장하므로 Hierarchy함수를 다시 부르는 재귀적인 구조가 되어야한다.

<Frame>: 문구나 나온 후 숫자 두개와 문자열이 입력되는데 첫번째 숫자는 프레임의 번호이고 두번째 번호는 해당 프레임이 가지는 텍스쳐의 개수이다. 마지막 문자열은 해당 프레임의 이름이다. 따라서 숫자를 두 번 읽고 문자열을 한 번 읽으면 Frame의 정보를 얻을 수 있다.



다음은 Transfrom이다. <Transform>: 이 후 13개의 실수가 입력되고 앞에서부터 3개씩 각각 Position, Rotation, Scale 값을 의미한다. 마지막 4개의 실수는 쿼터니언의 데이터이다.



위와 같이 데이터를 담을 수 있는 변수를 선언해 받아온다. 하지만 실제로 이 값은 쓰이지 않기 때문에 받아만 놓고 따로 저장을 해두지는 않는다.

다음은 TransformMatirx이고 이름처럼 행렬이 입력된다. 하지만 월드 행렬을 의미하지는 않는다.



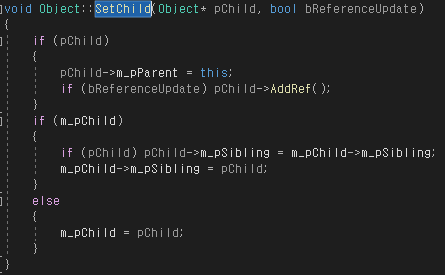
다음은 Mesh, SkinningInfo, Material, Children이 남았는데 4개 모두 하위 계층을 갖고있을 가능성이 있기 때문에 해당 데이터를 로드하는 부분을 만들어야 한다.

먼저 Children은 하위계층에 Frame이 등장한다. Frame의 데이터를 읽는 함수는 지금 만들고 있으므로 자기 자신을 다시 불러주면 된다.



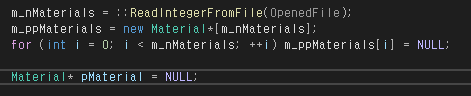
그 전에 오타가난 OpendFile을 OpenedFile 로 이름을 바꿔줬다.

Children 문구 이후 자식이 몇이 있는지 입력되므로 먼저 nChildren에 입력 받은 후 입력 받은 수만큼 LoadHierarchy 함수를 부른다. 이 때 pChild에 읽은 데이터를 저장하고 SetChild함수로 자식으로 연결한다.



SetChild 함수의 구조는 위와 같다. 먼저 입력 받은 pChild가 유효한지 확인 후 유효하다면 pChild의 부모를 자기 자신으로 설정한다. 입력 받은 bool값으로 참조를 늘려줄지 말지를 정하는데 기본값은 false이다. 두번째로 자기 자신의 자식인 m\_pChild가 있는지를 검사한다. 만약 없다면 받아온 pChild를 자기자식으로 하면 되고 만약 이미 자식이 존재 한다면 입력받은 pChild의 형제를 현재의 자식의 형제로 한 후 pChild를 m\_pChild의 형제로 만들면 자식과 형제들이 쭉 이어지게 된다. 해당 방식으로 부모와 자식과 형제가 계층(트리) 구조로 저장할 수 있게 된다.

다음은 Material을 불러오는 부분을 만든다. <Materials>: 가 나온 후 정수가 입력되는데 이 정수는 Material의 개수이고 개수에 맞는 <Material>: 문이 나온다. 이 <Material>: 부분을 읽을 함수를 만들어야 한다. 우선 Object 클래스에 Material 객체를 멤버로 갖고 있도록 만들어 준다. Material은 2개 이상일 수 있으니 이중 포인터로 만든다.

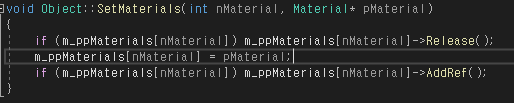


<Materials>: 다음의 정수를 읽어야 하므로 ReadInegerFromFile함수를 이용해 Material의 개수를 얻은 뒤 개수만큼 Material 객체를 생성해 준다. 생성한 Material들은 전부 NULL로 초기화한다.

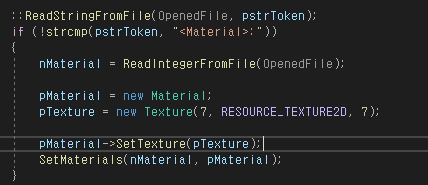
마지막 줄의 pMaterial은 이후 동적으로 생성될 Material의 포인터를 저장해두기 위한 변수이다.

Material의 데이터는 AlbedoColor, EmissiveColor, SpecularColor, Glossiness, Smoothness, Metallic, SpecularHighlight, GlossyReflection, AlbedoMap, SpecularMap, NormalMap, MetallicMap, EmissionMap, DetailAlbedoMap, DetailNormalMap 으로 총 15가지가 있다. 이중 끝에 Map이 들어가있는 7개를 제외한 나머지 8개의 데이터는 float값이고 이중에서도 Color가 들어가는 데이터는 float 4개의 데이터이다. Map이 들어가있는 7개의 데이터는 텍스쳐이고 텍스쳐 파일의 이름이 적혀있다. 따라서 저번주에 만들어 놓은 LoadTextureFromFile 함수를 여기서 사용하게 된다.

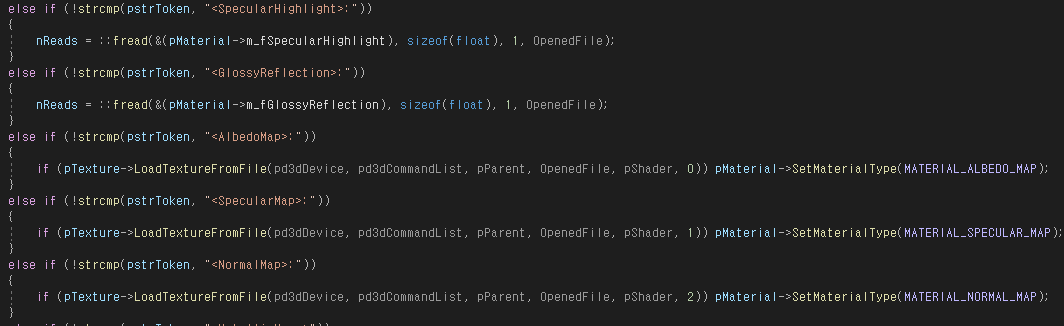
If else문으로 각 경우를 나누어 해당되는 데이터를 불러올 수 있도록 한다. 데이터를 저장할 수 있도록 해당 변수들을 Material 클래스에 만들어 준 뒤 fread 첫번째 인자에 만든 변수를 넣어 직접 받아올 수 있도록 한다. 우선 <Material>: 이 나왔을 경우 Material 객체를 생성하고 Material의 인덱스 번호를 받아 동일한 인덱스에 연결한다. 연결을 위한 SetMateials 함수는 다음과 같다.



다음은 Material을 생성해 SetMaterials 함수로 연결해 주는 작업이다.



Texture 생성자의 첫번째 인자는 텍스쳐의 개수이고, 2번째 인자는 리소스 타입, 3번째 인자는 루트파라미터 개수이다. 읽는 데이터에서 AlbedoMap, SpecularMap, NormalMap, MetallicMap, EmissionMap, DetailAlbedoMap, DetailNormalMap 총 7가지의 텍스쳐 종류가 있기 때문에 7을 넣어줬고 어떤 텍스쳐가 있을지 어떤 텍스쳐가 없을지 알 수 없기 때문에 일단 7개 전부 생성을 해준다. 루트파라미터도 텍스쳐마다 따로 설정을 해주어야 하기에 똑같이 7을 넣어준다. 이렇게 생성한 텍스쳐를 SetTexture 함수로 Material에 넣어주고 SetMaterials로 오브젝트와 연결한다. 그 후 나오는 문구에 맞게 데이터를 읽어서 저장해 준다. 다음은 데이터를 읽는 if 문의 일부분 이다.



텍스쳐를 읽는 부분은 마지막에 SetMaterialType 함수로 타입을 지정해 주는데 이 타입을 이용해 나중에 루트 파라미터에 값을 넘겨줄 때 어떤 타입인지를 보고 해당하는 루트 파라미터에 값을 넘겨준다.

**캐릭터 능력치 구상**

플레이어블 캐릭터는 전사와 마법사 두 명이고 몬스터는 근접, 원거리, 돌진 세 가지의 유형과 보스가 있다.

필요한 능력치는 HP, MP, 방어력, 공격력 정도

MP는 플레이어블 캐릭터에만 존재하며 100으로 고정한다.

HP는 플레이어블 캐릭터는 전부 100으로 하고 근접 몬스터도 100, 원거리 몬스터 80 돌진 몬스터 150으로 한다. HP는 밸런스 문제이기 때문에 이후 게임이 완성되었을 때쯤 조절해야 한다. 우선 이정도 수치로..

문제는 방어력과 공격력인데. 단순 빼기로 계산을 하면 만약 공격력이 방어력보다 적을 경우 -가 되어 문제가 발생할 수 있다. 적당한 비율로 데미지를 산출할 수 있는 수식을 만들어 봐야할 듯하다. 공격력이 커지면 결과도 커지고 방어력이 커지면 결과는 작아지는 수식이 필요하다. 그러면서 결과는 공격력과 방어력의 범위를 크게 벗어나지 않는 값이어야 하고 항상 양수 이어야 한다.

해당 수식을 사용하면 공격력이 커지면 값이 커지고 방어력이 커지면 값이 작아지며 양수의 결과만 얻을 수 있다. 방어력이 만약 -100이라면 문제가 되지만 우리 게임에서 방어력이 음수가 되는 일은 절대 없으므로 그냥 사용해도 괜찮을 것이다. 방어력 계수를 조정하여 밸런스 조절도 용이 할 것으로 생각된다. 방어력 계수가 100으로 했을 때의 수식은 실제로 “리그 오브 레전드” 라는 게임에서 사용 중인 피해량 계산 수식이다. 리그 오브 레전드는 공격력과 방어력이 거의 100 단위이고 체력도 1000단위로 계산되기 때문에 방어력 계수가 100이지만 우리 게임은 체력이 100이고 공격력은 10~20 언저리 정도로 생각하고 있기 때문에 방어력 계수를 그에 맞게 조절할 필요가 있다.

수식은 이렇게 하는데 수치를 어느 정도로 해야 할지는 팀원과 상의를 해봐야 할 것 같다. 몬스터를 대략 몇 번정도의 공격으로 쓰러뜨릴 수 있어야 할지, 몬스터의 공격을 몇 번 받아야 쓰러질 지 정해야 한다.