FBX 이해

자체 포맷을 FBX로 변환

SMD 형식에 기초한 M 포맷을 FBX로 변환한다. FBX Review로 볼 수 있도록 만든다.

가능성을 검토하는 정도로 하고 가능하면 구현을 시도한다.

스펙

구체적인 내용보다는 추상적으로 서술하고 실제 코드로 정리한다.

`FbxConvert file | folder`

- Mesh들
- Action들
- 본들

Mesh:

- Verices
- Normals
- TexCoords
- Triangles
- Texture File
- TextureScript
    - 텍스처 정보를 읽어서 지정

Action:

- animation을 의미
- Loop
- NumAnimationKeys
- Positions

Bone:

- Parent
- BoneMatrixes
  - 액션 개수 만큼 가짐
  - Position
  - 회전
    - Rotation
    - Quaternion
    - AngleQuaternion(Rotation, Quaternion) 으로 실행

Action과 Bone:

Bones[i]

Parent

BoneMatrix of an Action

Position [Action.NumAnimationKeys]

Bone

Action1

P1, P2, P3, ...

R1, R2, R3, ...

Action2

P1, P2, P3, ...

R1, R2, R3, ...

골격은 Bone 부모 자식 관계로 결정된다. 본의 위치는 특정 Action의 위치와 회전 값으로 결정된다.

Textures

Bitmap.GetTexture(filename)으로 가져온다.

크롬 쉐이더 효과, 알파 등 여러 효과를 지정한다.

BindTexture는 IndexTexture의 값으로 지정한다.

- glGenTexture와 텍스처 번호 지정은 로딩 시 하도록 되어 있다.

- 외부에서 encapsulation 안 되고 바로 지정한다.

TextureScript

- Bright
- HiddenMesh
- StreamMesh
- NoneBlendMesh
- ShadowMesh

1차 시도

- 매시
- 애님
- 텍스처

조사

- BMD::Open2() 로 모든 파일 변환 여부 확인
  - Open()을 사용하는 곳은 없다.
  - 간략한 XOR 암호화 사용
- SMD 남아있는 지?
  - 일부 사용하고 있는 듯 하나 추가 확인 필요

BMD::Open2()로 열어서 읽으면 된다. 여기서 읽은 데이터를 FBX로 쓰면 된다. FBX로 쓰는 과정이 간단하지는 않을 것이다.

구현

FBX SDK 사용

관련 자료 정리.

SDK Programming guide가 많은 내용을 제공하고 있어 특별한 어려움은 없다. Assimp의 FBX 로딩 기능이 스킨 매시와 재질을 포함하고 있으므로 참고한다.

Ex 1. 간단한 큐브 매시 만들기

- SDK 설치
  - VS 2015 버전
- FBX 파일 생성
- 뷰어로 확인
-

FBX\FBX SDK\2019.0\samples\UI Examples\CubeCreator

예제로 이미 있으니 이를 사용해서 Scene에 큐브를 만든다.

점점 매시랑 애님까지 갖는 구조로 확장한다.

튜토리얼의 내용을 함수 위주로 살핀다.

흐름은 다음과 같다.

- 카메라와 마커 추가
- 큐브 추가
  - 매시 추가
  - 텍스처 추가
  - 머티리얼 추가

매시 추가

CreateCubeMesh(FbxScene*, char* pName)

```
lMesh->InitControlPoints(24);
lMesh->GetControlPoints();
```

Q. 콘트롤 포인트는 무엇이고 어떻게 사용되는가?

콘트롤 포인트는 오토데스크의 용어로 보인다. 큐브 매시에서는 Vertex와 동일한 의미이다.

per-face vertices of a FbxMesh.

```
FbxGeometryElementNormal* l = lMesh->CreateElementNormal();
l->SetReferenceMode(FbxGeometryElement::eDirect);

l->GetDirectArray().Add(lNOrmalZPos);
... ;

// 각 콘트롤 포인트에 대해 노멀을 추가한다.
```

```
// Create UV for Diffuse channel.
FbxGeometryElementUV* lUVDiffuseElement = lMesh->CreateElementUV( "DiffuseUV");
FBX_ASSERT( lUVDiffuseElement != NULL);
lUVDiffuseElement->SetMappingMode(FbxGeometryElement::eByPolygonVertex);
lUVDiffuseElement->SetReferenceMode(FbxGeometryElement::eIndexToDirect);
```

위에서 diffuse 맵 지정을 준비한다.

```
//Now we have set the UVs as eIndexToDirect reference and in eByPolygonVertex  mapping mode
//we must update the size of the index array.
lUVDiffuseElement->GetIndexArray().SetCount(24);
```

Q. 위에서 왜 크기를 늘리고 루프를 돌 때 GetIndexArray.SetAt()의 의미는 뭔가?

```
// Control point index
lMesh->AddPolygon(lPolygonVertices[i*4 + j]);

// update the index array of the UVs that map the texture to the face
lUVDiffuseElement->GetIndexArray().SetAt(i*4+j, j);
```

UV 매핑은 폴리곤 버텍스 모드와 콘트롤 포인트 모드가 있다. (다른 것도 있는데 샘플에서 생략됨).

위는 FbxGeometryElement::eByPolygonVertex 모드로 UV를 지정하는 예시이다.

GetDirectArray()로 UV 좌표 4개를 지정하고 각 박스의 버텍스 인덱스 (순서)에 대해 사용할 Direct 배열의 값을 Index 배열에 지정한다. 즉, lUVDiffuseElemen.GetIndexArray[5] --> GetDirectArray[0]와 같이사용할 좌표를 매핑한다.

폴리곤을 만들 때 지정하는 경우 eByPolygonVertex 모드가 편리하다. 그리고, OpenGL 등에서 사용하는 방식이 이 모드이므로 이렇게 생각하는 게 편하다.

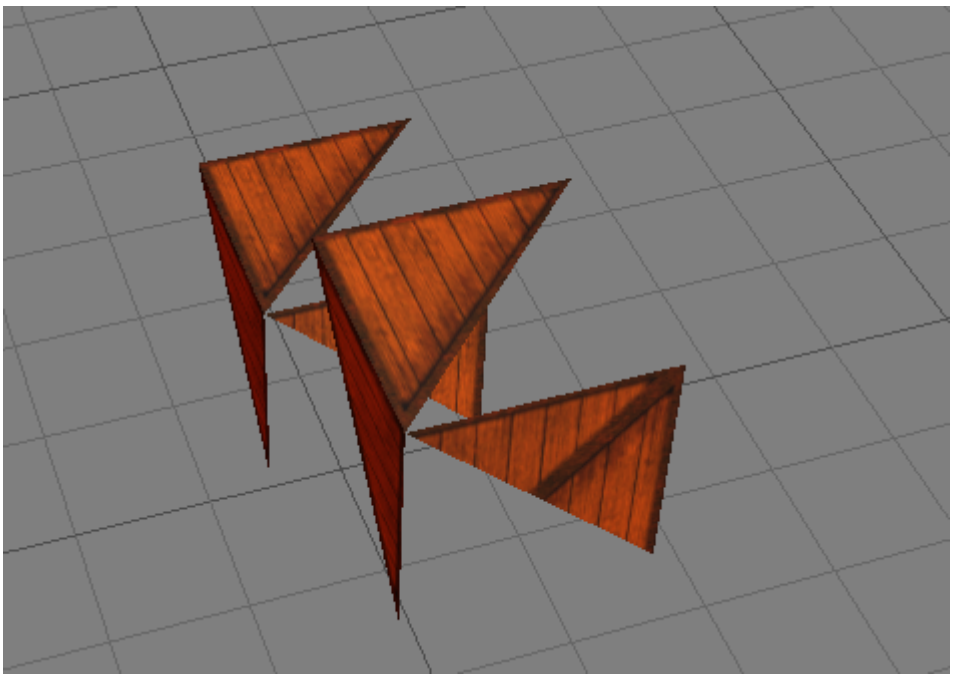
단지, DirectArray와 IndexArray가 UV에 대해서도 분리되어 있다는 점이 중요하다. 보통 렌더링 코드에서는 분리하지 않고 직접 사용한다. 이 기능도 eIndexToDirect 대신 eDirect 모드를 사용하면 배열의 기능을 직ㅓㅂ 사용할 수 있다.

Q. 아래 코드를 이해해야 한다.

```
// Create polygons. Assign texture and texture UV indices.
for(i = 0; i < 6; i++)
{
    // all faces of the cube have the same texture
    lMesh->BeginPolygon(-1, -1, -1, false);

    for(j = 0; j < 4; j++)
    {
```

```
            // Control point index
            lMesh->AddPolygon(lPolygonVertices[i*4 + j]);

            // update the index array of the UVs that map the texture to the face
            lUVDiffuseElement->GetIndexArray().SetAt(i*4+j, j);
        }

        lMesh->EndPolygon ();
    }
```

BeginPolygon과 EndPolygon 사이에 AddPolygon이 실행된다. 꼭지점을 추가한다. 0, 1, 2, 3 / 4, 5, 6, 7 과 같이 네 개의 점이 하나의 폴리곤이 된다. 즉, 사각형으로 각 면을 정의한다. 6면체이므로 6개의 폴리곤으로 매시가 구성된다.



위 코드에서 j=1부터 시작하면 한 면이 삼각형이 되어 비어 있는 육면체가 된다.

Ex 2. ViewScene FBX SDK 예제 분석

실행

- AnimationStack으로 애님 선택
- 스켈리톤?

휴머노이드 파일 실행. 완전한 소스라 규모가 큰 편으로 필요한 부분에 집중해서 보면 좋을 듯.

Ex 3. UE4 FBX Export 소스

FbxExporter::ExportSkeletalMesh() 함수

- ExportSkeletalMeshExport.cpp
  - ExportSkeletalMeshToFbx
  - ExportAnimSequence
  - CreateSkeleton
  - CreateMesh
  - BindMeshToSkeleton
  - CreateBindPose

CreateSkeleton

BoneNodes : List[FbxNode]

SkeletonAttribute : FbxSkeleton --> eLimNode 또는 eRoot

BoneNodes (Children)

CreateMesh

- ControlPoints
- Normal
- UV
  - Diffuse / Light
- Polygon
  - Index에서 추가
- Material
  - ExportMaterial 함수
  - Mesh 노드에 머티리얼 추가

ExportMaterial

MaterialInterface를 사용하여 저장.

FbxSurfacePhong 또는 FbxSurfaceLambert로 만들어서 export 진행.

BindMeshToSkeleton

```
FbxGeometry* MeshAttribute = (FbxGeometry*) MeshRootNode->GetNodeAttribute();
FbxSkin* Skin = FbxSkin::Create(Scene, "");

// 각 본에 대해
    FbxNode* BoneNode = BoneNodes[BoneIndex];

    // Create the deforming cluster
```

```
    FbxCluster *CurrentCluster = FbxCluster::Create(Scene,"");
    CurrentCluster->SetLink(BoneNode);
    CurrentCluster->SetLinkMode(FbxCluster::eTotalOne);

    // 본에 대한 가중치를 추가
    CurrentCluster->AddControlPointIndex(VertIndex, InfluenceWeight);

    ...

    Skin->AddCluster(CurrentCluster);

// 마지막에 스킨을 지정.
MeshAttribute->AddDeformer(Skin);
```

CreateBindPose

FbxPose를 만든다. SetIsBindPose(true)로 기본 포즈로 만들고 장면에 넣는다.

ExportAnimSequenceToFbx

Bone에서 Curve를 얻어온 다음에 각 커브에 애님 시퀀스 데이터를 지정한다.

FbxAnimStack이 하나만 있다. 여러 개를 같이 내보내는 방법은 없어 보인다. 어떻게 가능하지? 한 파일에 다 갖고 있을 필요는 없는데 그런 구조일 수도 있다.

Q. 여러 개의 애님 시퀀스를 export 하는 방법은 없는가?

Fbx는 여러 개의 FbxAnimStack을 가질 수 있는 구조이다. Fbx의 SDK 예제를 보면 휴머노이드에 대해 여러 개 의 애니메이션을 실행하는 걸 볼 수 있다. UE4는 스킨 매시와 애님 시퀀스가 분리된 구조로 여러 개의 애니메이션 선택을 지원한다.

현재 구현 코드 상으로는 하나의 애님 시퀀스만 본의 커브에 지정하는 방식으로 보인다.

Ex. 스킨 매시 처리

Fbx와 UE4, 자체 포맷 분석으로 어느 정도 이해했다.

- Skeleton (Bone)
  - 부모 자식
  - 변환 행렬
- 가중치
  - 영향 받는 본 인덱스와 가중치
  - 버텍스 별로 지정 (일반적)
- 애님 시퀀스
  - 애님 스택 / 액션

        - 다양한 이름으로 불림
    - Curve로 이루어짐
        - 고정 시간 간격
        - 시간 지정
- 애님 처리
    - 현재 애님 시퀀스의 커브를 가져와서
    - 각 본에 적용
    - 적용된 값에 따라 버텍스의 위치를 변경함

꼭지점에 적용할 때 가중치가 적용되는 본들의 최종 변환 행렬 값이 있어야 한다. 좌표계는 오브젝트의 로컬 좌표계이다. 애니메이션의 커브 값들이 로컬 좌표계 상에서 만들어진다. 그렇지 않으면 월드 상의 변환 행렬이 되기 때문이다.

애니메이션은 위치 값에 대해서만 적용되는 건 아니다. 색상이나 재질에 대해서도 지정할 수 있다. 시간을 이벤트로 보고 이벤트에 따라 행위에 따라 상태를 변경하는 방식은 항상 강력하다.

골격 쉐이더

https://www.khronos.org/opengl/wiki/Skeletal_Animation

- 노멀 맵을 고려하지 않은 애니메이션

http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html

- 위와 유사하나 assimp를 데이터를 사용한 CPU 코드와 쉐이더 코드를 포함
- 이 쪽이 상세한 설명을 포함
- 노멀 처리만 포함
    - 노멀 맵은 고려하지 않음
    - 이 쪽은 다른 영역