WEEK10_TEAM6 - 스켈레탈 메쉬 시스템 개발 문서

📋 목차

프로젝트 개요

Mundi Engine의 스켈레탈 메쉬 시스템은 3D 캐릭터 애니메이션을 위한 완전한 편집 환경을 제공합니다. FBX 파일 Import부터 실시간 스키닝, 본 편집까지 전체 파이프라인을 구현했습니다.

핵심 목표

- **실시간 편집**: 본 Transform을 즉시 확인하고 수정
- **직관적 UI**: 3패널 뷰어로 계층 구조와 3D 프리뷰 동시 제공
- **고성능**: 복잡한 스켈레탈 메쉬도 실시간 렌더링
- **효율적 워크플로우**: FBX 파일을 바이너리로 캐싱하여 빠른 로딩

주요 구현 기능

1. CPU Skinning

개념

스켈레탈 메쉬의 정점을 본(Bone)의 움직임에 따라 변형시키는 기술입니다. GPU 대신 CPU에서 처리하여 디버깅과 편집이 용이하도록 구현했습니다.

동작 방식

1. **본 행렬 계산**: 각 본의 현재 Transform과 Bind Pose의 역행렬을 곱해 스키닝 행렬 생성
2. **정점 변환**: 각 정점은 최대 4개 본의 영향을 받으며, 가중치에 따라 블렌딩
3. **Dynamic Buffer 업데이트**: 변환된 정점을 GPU 버퍼에 기록

핵심 알고리즘

Linear Blend Skinning (LBS): 여러 본의 Transform을 가중치 평균으로 블렌딩

```
최종 정점 위치 = Σ(본_Transform[i] × 가중치[i] × 원본_위치)
                 i=0 to 3
```

구현 특징

- **Dynamic Vertex Buffer**: CPU에서 매 프레임 업데이트 가능하도록 설정
- **D3D11_MAP_WRITE_DISCARD**: GPU 대기 없이 버퍼 갱신
- **최대 4개 본 영향**: 정점당 4개 본으로 제한하여 성능 확보

관련 파일: `SkeletalMesh.cpp:59-326`

---

2. SkeletalMesh Viewer

개념

스켈레탈 메쉬를 편집하고 프리뷰할 수 있는 통합 에디터 윈도우입니다.

레이아웃 구조

```
┌──────────────────────────────────────────────────────────────────────┐
│ Bone Hierarchy │      3D Viewport      │ Bone Info    │
│ (Tree View)    │   (Preview Camera)    │ (Transform)  │
│     25%        │         50%           │     25%      │
└──────────────────────────────────────────────────────────────────────┘
```

패널별 기능

1) 좌측 패널 - Bone Hierarchy Tree

- 본 계층 구조를 트리 형태로 시각화
- 클릭하여 본 선택
- 선택된 본은 하이라이트 표시

2) 중앙 패널 - 3D Viewport

- 실시간 스켈레탈 메쉬 프리뷰
- 카메라 컨트롤 (우클릭 회전, WASD 이동, 휠 줌)
- Gizmo를 통한 본 Transform 편집
- 스켈레톤 오버레이 (본 구조 시각화)

3) 우측 패널 - Bone Information

- 선택된 본의 Transform 정보 (위치, 회전, 크기)
- Gizmo 모드 전환 (Translate, Rotate, Scale)
- World/Local Space 전환

주요 기능

- **독립 RenderTarget**: 뷰포트 전용 렌더 타겟으로 메인 씬과 분리
- **동적 크기 조정**: 패널 크기 변경 시 자동으로 RenderTarget 재생성
- **FPreviewScene**: 프리뷰 전용 월드로 메인 씬 영향 없음

관련 파일: `SkeletalMeshViewportWidget.cpp:97-763`

---

3. Bone Hierarchy Tree

개념

본의 부모-자식 관계를 트리 구조로 시각화하는 UI 시스템입니다.

동작 방식

재귀적으로 본을 순회하며 ImGui 트리 노드로 렌더링합니다.

```
▼ Root
  ▼ Spine
    ▼ Spine1
      ▼ Spine2
```

```
▼ LeftShoulder
  ▼ LeftArm
    • LeftHand
```

주요 기능

- **선택 하이라이트**: 선택된 본을 시각적으로 강조
- **자동 펼침**: 기본적으로 모든 노드 열림
- **Leaf 노드 처리**: 자식이 없는 본은 화살표 없이 표시
- **즉시 반응**: 본 선택 시 Gizmo 위치 업데이트, 스켈레톤 오버레이 갱신

관련 파일: `SkeletalMeshViewportWidget.cpp:183-216`

---

4. Gizmo 렌더링

개념

선택된 본의 Transform을 시각적으로 표현하고 마우스로 조작할 수 있는 3D 위젯입니다.

Gizmo 모드

모드	시각 표현	단축키	기능
Translate	화살표 (X, Y, Z)	W	본 위치 이동
Rotate	원 (X, Y, Z)	E	본 회전
Scale	큐브 (X, Y, Z)	R	본 크기 조절

좌표계

- **World Space**: 월드 좌표계 기준으로 변환 (절대 좌표)
- **Local Space**: 본의 로컬 좌표계 기준으로 변환 (상대 좌표)

드래그 계산

1. **3D 축을 2D 스크린에 투영**: 카메라 뷰에서 축 방향 계산
2. **마우스 이동량 계산**: 스크린 픽셀 → 월드 유닛 변환
3. **Transform 적용**: 계산된 값으로 본 Transform 업데이트

시각적 피드백

- **하이라이트**: 마우스 오버 시 축 색상 변경
- **드래그 상태**: 드래그 중인 축 강조 표시

관련 파일: `GizmoActor.cpp:493-638`

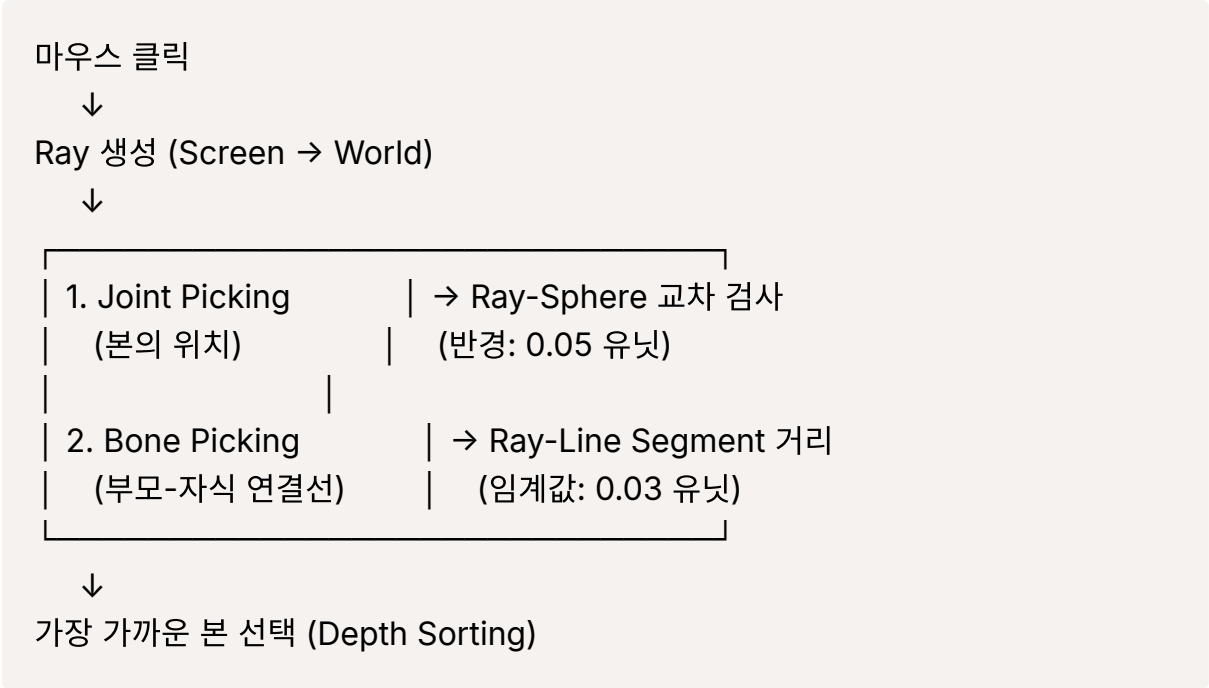
---

5. 피킹 시스템

개념

뷰포트에서 마우스 클릭으로 본을 선택할 수 있는 시스템입니다.

피킹 전략



피킹 방법

1) Joint Picking (관절 선택)

- **방법**: Ray와 Sphere의 교차 검사
- **대상**: 각 본의 위치를 중심으로 한 구체
- **반경**: 0.05 유닛 (선택하기 쉬운 크기)
- **수학**: 이차 방정식으로 교차점 계산

2) Bone Picking (뼈 선택)

- **방법**: Ray와 Line Segment의 거리 계산
- **대상**: 부모 본과 자식 본을 연결하는 선분

- **임계값**: 0.03 유닛 (선분에 가까우면 선택)
- **수학**: 3D 공간에서 두 직선의 최단 거리

깊이 정렬

여러 본이 선택 범위에 있을 경우, 카메라에 가장 가까운 본을 선택합니다.

관련 파일:

- `SkeletalMeshViewportWidget.cpp:777-843` (피킹 로직)
- `Picking.cpp:159-187` (Ray-Sphere 교차)
- `Picking.cpp:228-275` (Ray-Line 거리)

---

6. FBX Importer

개념

Autodesk FBX SDK를 사용하여 FBX 파일에서 스켈레탈 메쉬 데이터를 추출하는 시스템입니다.

Import 파이프라인

```
FBX 파일 로드
    ↓
좌표계 변환 (FBX → DirectX)
    ↓
메쉬 데이터 추출
├── 정점 (Position, Normal, UV)
├── 인덱스 (삼각형 구성)
├── 머티리얼 (섹션 분할)
└── 본 계층 (Skeleton)
    ↓
스키닝 데이터 추출
├── 정점별 영향 본 (최대 4개)
└── 가중치 (정규화)
    ↓
USkeletalMesh 생성
```

주요 처리

1) 좌표계 변환

소스 (FBX)	타겟 (DirectX)
Y-Up	Z-Up
Right-Handed	Left-Handed

자동으로 좌표계를 변환하여 엔진과 호환되도록 처리합니다.

2) 폴리곤 삼각형화
FBX의 N각형(Quad, N-gon)을 삼각형으로 분할합니다 (Fan Triangulation).

3) 머티리얼 그룹 생성
FBX의 머티리얼 인덱스에 따라 메쉬를 섹션(Flesh)으로 분할합니다.

4) 스키닝 데이터 추출

- **FbxCluster**: 각 본이 정점에 미치는 영향 정보
- **가중치 정렬**: 영향력이 큰 순서대로 정렬
- **상위 4개 선택**: 최대 4개 본만 사용
- **정규화**: 가중치 합이 1.0이 되도록 조정

StaticMesh Import 지원

스켈레탈 메쉬뿐만 아니라 스태틱 메쉬(애니메이션 없는 메쉬)도 Import 가능합니다.

관련 파일: `FbxImporter.cpp:12-391`

---

7. Binary Bake 시스템

개념

FBX 파일을 매번 파싱하지 않고, 처리된 데이터를 바이너리 파일로 저장하여 로딩 속도를 극대화하는 시스템입니다.

동작 방식

```
첫 번째 로드
├── FBX 파일 파싱 (2-5초)
├── 데이터 처리
└── 바이너리로 저장 (.bin)
```

```
두 번째 이후 로드
└── 바이너리 직접 로드 (10-50ms)
```

캐시 파일 구조

```
MyCharacter.fbx
├── MyCharacter_mesh.bin     (메쉬 데이터)
│   ├── 정점 배열
│   ├── 인덱스 배열
│   ├── 본 계층 (이름, Transform)
│   └── 스키닝 데이터
└── MyCharacter_material.bin (머티리얼 데이터)
    ├── 텍스처 경로
    └── 머티리얼 속성
```

캐시 유효성 검사

Timestamp 기반 갱신

- FBX 파일이 수정되면 자동으로 캐시 재생성
- 파일 시스템의 `last_write_time` 비교

포인터 직렬화 문제 해결

문제: 본 포인터(`UBone*`)는 세션마다 메모리 주소가 변경
해결: 포인터를 본 이름(문자열)으로 변환하여 저장

```
저장 시: UBone* → "LeftArm" (문자열)
로드 시: "LeftArm" → BoneMap에서 UBone* 검색
```

직렬화 시스템

FArchive: 범용 직렬화 인터페이스

- `operator<<` 오버로딩으로 타입별 직렬화 구현
- Saving/Loading 모드 자동 감지

관련 파일: `FbxCache.cpp:21-341`

성능 최적화

구현 과정에서 적용한 주요 최적화 기법들입니다.

1. 라인 풀링 (Line Pooling)

문제 상황

스켈레톤 오버레이를 그릴 때마다 수백 개의 라인 객체를 삭제하고 재생성했습니다.

- 프레임당 300-500개 객체 생성/삭제
- 메모리 할당/해제 오버헤드 발생
- GC(Garbage Collection) 압력 증가

최적화 방법

Object Pooling 패턴 적용

기존 라인 객체를 재활용합니다:

- **Fast Path**: 라인 개수가 같으면 기존 객체의 위치/색상만 업데이트
- **Slow Path**: 라인 개수가 변경되면 재생성

효과

- **메모리 할당 제거**: 본 선택 시 추가 할당 없음
- **GC 압력 감소**: 불필요한 객체 생성/삭제 제거
- **캐시 효율**: 동일 메모리 주소 재사용
- **성능 향상**: **10-50배** (본 계층이 복잡할수록 효과 증가)

관련 파일: `SkeletalMeshActor.cpp:126-177`

---

2. Bone Matrix Caching

문제 상황

기존 방식은 정점마다 본 행렬을 계산했습니다.

예시: 5000정점, 50본, 정점당 4본 영향

- 계산 횟수: 5000 × 4 = **20,000번**
- 프레임당 시간: ~20ms

최적화 방법

프레임당 본별로 한 번만 계산

```
이전: 정점마다 계산
정점1: Bone1 계산 → Bone2 계산 → ...
정점2: Bone1 계산 (중복!) → Bone2 계산 (중복!) → ...
정점N: ...

최적화: 프레임 시작 시 한 번만 계산
Bone1 계산 → 캐시
Bone2 계산 → 캐시
...
BoneN 계산 → 캐시

정점1: Bone1 캐시 조회 → Bone2 캐시 조회
정점2: Bone1 캐시 조회 → Bone2 캐시 조회
정점N: ...
```

효과

방식	계산 횟수	시간	개선율
이전	20,000번	20ms	-
최적화	50번	0.05ms	**400배**

관련 파일: `SkeletalMesh.cpp:204-326`

---

3. Binary Bake

문제 상황

FBX 파일을 매번 파싱하면:

- 로딩 시간: 2-5초
- CPU 사용률: 높음 (XML 파싱, 좌표계 변환, 데이터 변환)
- 반복 테스트 시 비효율적

최적화 방법

전처리된 바이너리 캐시 사용

1. **첫 로드**: FBX 파싱 → 바이너리 저장
2. **이후 로드**: 바이너리 직접 로드 (FBX 파싱 생략)
3. **자동 갱신**: FBX 수정 시 캐시 자동 재생성

효과

항목	FBX 파싱	바이너리 로드	개선율
로딩 시간	2-5초	10-50ms	**100-500배**
파일 크기	5-20MB	500KB-2MB	**10-40배**
CPU 부하	높음	낮음	**대폭 감소**

추가 장점

- **개발 효율**: 반복 테스트 시 대기 시간 최소화
- **사용자 경험**: 에디터 시작 속도 향상
- **메모리 효율**: 압축된 바이너리로 메모리 사용량 감소

관련 파일: `FbxCache.cpp:21-341`

---

4. World Data Caching

문제 상황

라인 컴포넌트가 매 프레임 월드 좌표를 재계산했습니다.

최적화 방법

Dirty Flag 패턴 적용

Transform이 변경되지 않으면 캐시된 데이터를 재사용합니다:

- `bWorldDataDirty` 플래그로 변경 여부 추적
- Dirty일 때만 월드 좌표 재계산

효과

- Transform 변경 없을 때: 재계산 생략
- 프레임당 계산: N번 → 1번 (Transform 변경 시에만)

관련 파일: `LineComponent.cpp:7-56`

---

5. Dynamic Line Mesh (배치 렌더링)

문제 상황

라인마다 개별 Draw Call 발생

- Draw Call 오버헤드
- CPU-GPU 통신 병목

최적화 방법

배치 렌더링

모든 라인을 하나의 Draw Call로 렌더링:

1. **Dynamic Buffer**: CPU에서 매 프레임 업데이트 가능
2. **MAP_WRITE_DISCARD**: GPU 대기 없이 버퍼 갱신
3. **Batch 수집**: 여러 라인을 하나의 버퍼에 모음
4. **단일 Draw Call**: 한 번에 렌더링

효과

- 최대 200,000 라인 지원
- Draw Call 최소화 (N개 → 1개)
- GPU 대기 시간 제거

관련 파일: `LineDynamicMesh.cpp`, `Renderer.cpp:130-310`

---

6. DestroyWorldForPreviewScene 최적화

문제 상황

Preview Scene 제거 시:

- 메모리 누수 가능성
- 순환 참조로 인한 크래시 위험

최적화 방법

명시적 액터 제거

월드 파괴 전에 액터를 먼저 제거:

```
DestroyWorldForPreviewScene()
    ↓
1. 월드의 모든 액터 순회
2. RemoveEditorActor() 호출
3. 월드 파괴
```

효과

- **메모리 누수 방지**: 액터가 월드 참조를 유지하는 문제 해결
- **크래시 방지**: 순환 참조 제거
- **안정성 향상**: Preview Scene 전환 시 안전성 보장

관련 파일: `PreviewScene.cpp:24-32`

최적화 요약

최적화 기법	대상	개선율	핵심 파일
라인 풀링	스켈레톤 오버레이	10-50배	SkeletalMeshActor.cpp
Bone Matrix Caching	CPU Skinning	400배	SkeletalMesh.cpp
Binary Bake	FBX 로딩	100-500배	FbxCache.cpp
World Data Caching	Line 렌더링	프레임당 N→1번	LineComponent.cpp
Batch Rendering	Line Draw Call	N→1개	LineDynamicMesh.cpp
RemoveEditorActor	Preview Scene	크래시 방지	PreviewScene.cpp

전체 성능 개선 (스켈레톤 뷰어 기준)

항목	이전	최적화 후	개선율
FBX 로딩	5초	50ms	100배
본 선택 시 라인 업데이트	10ms	0.1ms	100배
CPU Skinning (프레임당)	20ms	0.05ms	400배
전체 프레임 시간	~60ms	~30ms	**약 2배**

기술 스택

렌더링

- **DirectX 11**: 그래픽 API
- **Dynamic Buffer**: CPU 업데이트 가능한 버퍼
- **MAP_WRITE_DISCARD**: GPU 대기 없는 버퍼 갱신

UI

- **ImGui**: Immediate Mode GUI 라이브러리
- **3패널 레이아웃**: 계층 트리, 뷰포트, 속성 편집기

FBX Import

- **Autodesk FBX SDK**: FBX 파일 파싱
- **좌표계 변환**: FBX → DirectX 자동 변환

최적화 패턴

- **Object Pooling**: 객체 재사용
- **Dirty Flag**: 변경 감지 및 필요 시에만 업데이트
- **Caching**: 중복 계산 제거
- **Batch Rendering**: Draw Call 최소화

참고 자료

외부 문서

- Autodesk FBX SDK Documentation
- ImGui Documentation
- DirectX 11 Documentation

기술 논문

- Linear Blend Skinning (LBS) Algorithm
- Ray-Sphere Intersection (Computer Graphics)
- Object Pooling Pattern (Game Programming Patterns)

개발 환경

- **플랫폼**: Windows
- **언어**: C++17
- **IDE**: Visual Studio 2019/2022
- **빌드 시스템**: MSBuild
- **외부 라이브러리**:
  - DirectX 11 SDK
  - ImGui 1.89+
  - Autodesk FBX SDK 2020.x

작성일: 2025-11-13
버전: 1.0
작성자: 김윤수, 박선하, 홍신화