3D게임2 프로그래밍

과제2

게임공학과 3학년 2020182028 이규원

목차

1. **과제에 대한 목표**

* 스카이박스 오류 해결
* 클래스 구조 개선
* 빌보드 구현.

1. **구현 내용**

* 스카이박스 오류 해결
* 플레이어 헤더에 할당된 다른 오브젝트 분리
* 기하셰이더를 이용한 빌보드 기법

1. **구현 사진**

**1. 과제에 대한 목표**

**스카이박스 오류 해결**

* 스카이박스는 게임 환경에서 3D 공간을 둘러싼 배경을 구성하는 중요한 요소 중 하나지만, 과제1에서 스카이박스의 한쪽 면이 어둡게 표시되는 문제가 발생했고, 이를 수정하고자 했다.

**클래스 구조 개선**

* 플레이어 헤더에서 스카이박스, 지형코드 등이 포함되어 있어 각각 따로 입력을 받는 것이 어려워 문제가 발생했다.

앞으로 졸작과 같은 대형 프로젝트에 적용을 위해 가독성과 유지보수 측면에서 객체지향 프로그래밍 기법은 중요했고, 분리했다.

**빌보드 구현**

* 텍스처 기반의 나무와 같은 간단한 오브젝트들을 시각적 일관성을 유지하며, 리소스 사용을 최적화하는 빌보드 구현은 곧 작업할 졸업작품의 첫번째 스테이지인 초원 맵에 필요한 기법이다.

기하셰이더를 이용해 빌보드를 구현하고, 과제에선 랜덤 배치를 통해 지형을 꾸며질 요소로서, 졸업작품에선 맵의 경계선을 간접적으로 표현하는 요소로서 구현하고자 했다.

**2. 구현 내용**

**스카이박스 오류 해결**

초기에 스카이박스를 렌더링할 때 사용돈 코드는 다음과 같았다.

InitMatrix(CmdList, RENDER\_TYPE\_PERS);

Transform::Scale(ScaleMatrix, 500, 500, 500);

RenderMesh(CmdList, SkyboxMesh, SkyboxTex, ObjectShader, 1.0f);

이 코드는 스카이박스를 초기화하고, 크기를 확장해, 렌더링하는 단순한 구조였다. 하지만 한쪽 면이 어둡게 보이는 문제가 발생했고, 이는 다음 원인들로 인해 발생한 것으로 파악했다.

1. **텍스처 플리핑 설정 문제**

* RenderMesh 호출 시 텍스처가 올바르게 렌더링되지 않아, 면의 조명이 왜곡될 수 있었다.
* FlipTexture(CmdList, false)를 통해 텍스처가 각 면에 정확히 매핑되도록 했다.

1. **카메라 모드 문제**

* 플레이어 헤더에 구현하며 플레이어에 지정된 카메라 설정이 적절하지 않아, 조명이 왜곡될 수 있었다.
* 후술할 객체지향 구조로 변경하고, camera.SetToDefaultMode()카메라 세팅을 Default로 변경하는 코드를 구현해 오류를 해결했다.

**플레이어 헤더에 할당된 다른 오브젝트 분리**

초기 프로젝트 구조에서 플레이어 헤더에 대부분의 오브젝트를 관리했었다. 충돌처리 검사 등을 위해 다른 오브젝트를 참조하기 편했지만, 유지보수 및 확장성 관점에서 큰 장애물이 되었고, 각각 별도의 클래스로 분리해 각 객체의 역할을 명확히 정의하고자 했다.

1. **클래스 구조 개선**

* 초기 설계에서 지형, 스카이박스, 적 등의 오브젝트가 개별 헤더로 분리되지 않고, 하나의 클래스에 혼재되어 관리되었다.
* 오브젝트들을 독립적으로 관리하기 위해 별도의 헤더 파일로 분리하고, 이를 객체 지향 구조를 기반으로 효율적으로 관리하도록 설계했다.
* Skybox.h : 스카이박스의 크기, 위치, 텍스처, 렌더링 관리
* Terrain.h : 지형의 크기, 위치 관리
* Enemy.h : 적의 위치, 이동 패턴, 충돌 처리 관리
* 모든 오브젝트는 scene이란 테이블에 등록해서 관리한다. 이후 새로운 오브젝트를 추가할 때에도 scene에 등록하기만 하면 된다.
* 객체 지향적 구조 개선을 통해 각 클래스가 자신의 역할만 관리해 유지보수 및 확장성이 향상됐다.

1. **외부 헤더의 오브젝트 참조 개선**

* Control클래스가 Enemy 클래스를 직접적으로 의존하면 각 독립성이 낮아지고, 캡슐화 원칙을 위반한다고 판단했다.
* 객체 지향 원칙에 따라 외부 헤더 파일을 직접 참조하는 대신, Scene 테이블에서 객체를 동적으로 참조했다.
* auto enemy\_object = scene.Find("enemy")와 같이 사용하면, enemy.h를 직접 수정하지 않고도 정보를 가져올 수 있다.
* 기존 코드 수정 없이 적을 추가로 구현할 때도, scene테이블에서 기존과 같이 참조해도 된다.
* 위와 같이 의존성이 감소해 모듈화된 구조가 구현됐다.

**기하셰이더를 이용한 빌보드 구현**

나무와 같은 객체들을 복잡한 메쉬로 만들어서 많이 만들면 프레임 레이트 관리가 어렵다. 그래서 졸업작품에도 빌보드 시스템을 구현해 리소스 사용을 최적화하기 위해 구현하고자 한다. 텍스처가 항상 카메라를 향하도록 처리해 시각적 일관성을 유지하고자 한다.

* **구현 내용**
* Tree 클래스는 GameObject를 기반으로 설계됐다. 나무의 위치, 회전 텍스처 정보를 관리한다.
* 홈화면에서 인게임으로 넘어갈 때 <random>의 함수로 랜덤으로 좌표를 설정해 Tree클래스로 넘겨 해당 좌표로 변환해 배치시킨다
* MathUtil.cpp에 빌보드 전용 함수를 만들었다.

객체의 Up, Right, Look 백터와 현재위치, 카메라의 위치를 구하고, Mat4::LookAtLH를 호출해 변환하고 뷰 행렬값을 변환 행렬에 복사해 빌보드 오브젝트의 방향을 플레이어를 계속 바라보게 구현했다.

* 항상 플레이어를 바라보는 텍스처로 모든 방향을 구현해야 하는 3D 메쉬보다 크게 리소를 줄일 수 있어 객체를 많이 생성해도 메모리 사용에 큰 부담이 되지 않았다.

**3. 구현 사진**

- 빌보드를 이용해 만든 나무가 플레이어를 계속 바라보고 있음을 알 수 있다.

- 스카이박스의 조명이 정상적으로 출력 되는 것을 볼 수 있다.

야외, 하늘, 사람, 구름이(가) 표시된 사진

자동 생성된 설명