JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 25, No. 5, pp. 1387-1397, May. 2024

Check for updates

언리얼 엔진 나이아가라 파티클 시뮬레이션을 이용한 대규모 메타버스 플랫폼 구현

정 광 무[1] · 정 일 권[2] · 김 진 술[3*]
[1]전남대학교 지능전자컴퓨터공학과 석사과정
[2]ETRI 초실감메타버스연구소 콘텐츠연구본부 본부장
[3*]전남대학교 지능전자컴퓨터공학과 교수

Implementation of a Massive Metaverse Platform using Niagara Particle Simulation in Unreal Engine

Kwang-Moo Chung[1] · Il-Kwon Jeong[2] · Jinsul Kim[3*]

[1]Master's Course, Department of Intelligent Electronics and Computer Engineering, Chonnam National University, Gwangju 61186, Korea
[2]Assistant Vice President, Hyper-Reality Metaverse Research Laboratory Content Research Division, Daejeon 34129, Korea
[3*]Professor, Department of Intelligent Electronics and Computer Engineering, Chonnam National University, Gwangju 61186, Korea

[요 약]

최근 코로나19의 유행으로 인해 디지털 기술이 빠르게 발전하고 있으며 메타버스와 같이 새롭게 떠오르고 있는 기술 분야의 연구가 활발히 진행되고 있다. 그러나 대규모 공연장 같은 플랫폼을 위한 메타버스 환경 구축을 위한 선행 연구가 존재하지 않고, 특히 대규모 렌더링 연산을 클라이언트 단에서 최적화하는 연구가 부족하다. 본 논문에서는 언리얼 엔진 5(Unreal Engine 5)라는 게임 개발 및 건축, 시뮬레이션 개발 엔진을 이용하여 클라이언트단에서 수만 명 이상의 인파를 안정적으로 시뮬레이션할 수 있는 메타버스 환경을 구축하는 방법을 제안했다. 언리얼 엔진 5에서 제공하는 나이아가라 파티클 시뮬레이션을 통해 메타버스 플랫폼 상의 플레이어들을 구현했으며, LoD 및 버텍스 애니메이션 기법을 이용하여 그래픽 자원을 최적화하였다. 그 결과, 안정적으로 60FPS를 상회하며 수십만 명 이상을 수용할 수 있었다.

[Abstract]

Due to the COVID-19 pandemic, digital technology is rapidly advancing, and research in emerging fields such as the metaverse is actively underway. However, no prior research examines the development of metaverse environments such as massive performance venues; especially, there is lack of research on optimizing client-side massive rendering. This study proposes a method for building an metaverse environment that can sustain crowds made up of thousands on the client side using Unreal Engine 5, which is an engine for game development, architecture, and simulation. We simulated a crowd of players on the metaverse platform through Niagara particle simulation provided by Unreal Engine 5 and optimized the graphic resources using the level of detail and vertex animation techniques. As a result, we managed to accommodate over 100,000 people while exceeding a frame rate of 60 fps.

색인어 : 언리얼 엔진, 메타버스, 나이아가라, 레벨 오브 디테일, 최적화
Keyword : Unreal Engine, Metaverse, Niagara, Level of Detail, Optimization

http://dx.doi.org/10.9728/dcs.2024.25.5.1387

Received 29 February 2024; **Revised** 18 March 2024
Accepted 29 March 2024

***Corresponding Author; Jinsul Kim**
Tel: ※ 개인정보 표시제한
E-mail: jsworld@jnu.ac.kr

Ⅰ. 서 론

코로나19의 급격한 전파로 인해 전 세계적으로 사회적 거리두기 및 재택근무 등의 새로운 환경이 활발하게 조성되는 사회적 현상이 발생하면서 디지털 기술은 우리의 삶에 더욱 긴밀하게 스며들게 되었다[1]. 이에 따라, 메타버스와 같은 가상세계에서의 사용자 간 상호작용과 사용자 경험이 더욱 중요해지고 있어 앞으로도 사람들은 메타버스에서 보내게 되는 시간이 점점 많아질 것으로 보이며 가상화폐 및 AI 등과 융합되어 디지털 기술 발전을 이끌어 나가는 주력 분야가 될 것으로 전망된다[2].

한편, 현재의 메타버스는 일부 게임 플랫폼까지 포함하여 메타버스 플랫폼이라고 부를 수 있을 정도로 메타버스의 범위에 대한 명확한 정의조차 아직 존재하지 않는 것처럼 여전히 개발의 여지가 많아 시장성이 있는 분야이기 때문에 하드웨어 및 소프트웨어 기술에 대한 투자가 활발히 이루어지고 있다[3]. 그러나 상용화되어 서비스 중인 대부분의 메타버스 플랫폼들의 경우 서버 네트워크 부하 및 클라이언트단의 그래픽 메모리 자원의 한계로 인해 한 공간 안에 수십 명에서 수백 명의 인원을 수용하는 것이 실질적 한계이다. 공연장이나 올림픽 경기장과 같은 대규모의 메타버스 플랫폼 혹은 수만 명 이상의 인파를 시뮬레이션해야 하는 상황이 올 경우 서버 및 클라이언트 모두에서 최적화가 이루어져야 한다.

서버 측의 네트워크 기술 연구에 비해 실무 중심인 클라이언트 측에서의 최적화 방법과 같은 기술의 연구가 부족하기 때문에 본 연구에서는 클라이언트측 최적화를 중심으로 하여 한 메타버스 공간 안에 수만 명 이상의 3D 캐릭터로 표현된 플레이어를 동시에 시뮬레이션하면서 모니터 화면이 업데이트되는 초당 프레임 수(FPS;frames per second)를 60 이상으로 유지할 수 있는 시뮬레이션 환경을 개발하는 것을 목표로 한다. 또한, 단순히 많은 수의 플레이어를 배치하는 데에 그치지 않고 한명 한명의 플레이어가 실제 공간 내에서 이동하고 있는 모습을 애니메이션으로 반영한 채로 대규모 인파를 무리 없이 시뮬레이션할 수 있을 정도로 최적화를 진행하여, 보다 현실감 있는 플랫폼을 구현하고자 한다. 본 시뮬레이션 플랫폼 개발에 사용된 최적화 기술을 여러 메타버스 플랫폼, 나아가 다양한 게임, 시뮬레이션 및 영화 업계에서도 활용할 수 있을 것으로 기대한다.

Ⅱ. 관련 연구 및 개념

2-1 3D 그래픽 렌더링 최적화

3D 컴퓨터 그래픽을 활용하는 문화콘텐츠를 제작하는 데에 있어서 다양한 최적화 방법이 존재하며, 상황에 맞게 적절한 최적화 기법을 활용하여 사용자 경험을 해치지 않으면서 최대한 원활한 동작을 목표로 하는 것이 중요하다. Kim은 다양한 종류와 형태의 디테일이 높은 하이폴리(Hi-Poly) 3D 오브젝트를 폴리곤 리덕션(Polygon Reduction)을 통해 폴리곤 수를 조절하여 최적화를 할 경우, 대체로 인공적인 하드서피스(Hard Surface) 물체보다 캐릭터와 같은 유기적인 오브젝트가 더 나은 결과를 보여준다고 분석하였다[4]. Lee 등은 게임 최적화 방법 중 하나인 Interior Mapping 기법을 적용해 폴리곤 개수, 리소스(resource) 크기, FPS 등으로 평가하여 3D 그래픽 처리 성능의 차이를 검증하였다[5].

2-2 LoD(Level of Detail)

본 연구에서는 범용적으로 쓰이는 게임 엔진에서 효과적으로 렌더링 최적화를 수행하고, 이를 정량적인 측정지표를 통해 수치화하기 위해 LoD(Level of Detail) 라는 개념을 사용한다.

3D 시뮬레이션 공간상에 있는 물체는 플레이어 카메라에 가까이 존재할 수도 있고 멀리 떨어진 곳에 존재할 수도 있으며, 멀리 떨어져 있을 경우 플레이어가 느끼게 되는 해당 물체의 세부 디테일의 양이 크게 감소하게 됨에도 불구하고 물체를 렌더링하는 데 사용되는 폴리곤의 개수는 그대로 유지되게 되어 렌더링 성능이 떨어지게 된다. 이 문제를 해결하기 위해 LoD 최적화 기법을 사용하여 3D 물체와 플레이어 카메라의 거리가 멀어질수록 해당 3D 물체를 렌더링하는 데에 필요한 폴리곤 수를 감소시켜 그래픽 자원의 부하를 줄일 수 있다. Aubel 등은 2000년도 혹은 그 이전의 시스템으로 LoD(Level of Detail) 최적화 기법을 활용하여 안정적인 실시간 24 FPS를 유지하는 동시에 120명의 애니메이션이 적용된 가상 캐릭터를 한 공간상에 시뮬레이션하는 것에 성공한 바 있다[6]. Wysopal 등은 AR 환경에서 사용자가 바라보는 각도 및 거리에 따라 UI 요소의 디테일이 변화하는 LoD를 적용하여 45명의 참가자에게 일련의 작업을 수행하는 실험을 진행하였는데 LoD를 적용한 환경에서 수행한 결과가 그렇지 않은 환경에서 수행한 결과에 비해 속도 면에서 상당한 향상이 이루어졌음을 보여 LoD의 도입이 어플리케이션의 사용성 측면을 개선할 수 있다는 것을 제시하였다[7].

2-3 Vertex Animation Texture

3D 공간상에 표현되는 가상 캐릭터의 애니메이션의 경우 CPU의 자원을 활용하여 처리되기 때문에 처리해야 할 애니메이션의 양이 많아질 경우, 효율적인 자원 관리 및 최적화를 하지 않으면 처리가 지연되어 전체적인 성능에 영향을 미칠 가능성이 있다.

일반적으로 캐릭터의 복잡한 움직임, 액체, 건물 붕괴, 깨짐 등의 시뮬레이션은 본(bone)을 이용한 리깅(rigging) 등을 통해 이루어지지만, 외부 3D 프로그램을 통해 제작된 시뮬레이션을 게임 엔진 등으로 옮겨 구현하고자 할 경우 서로 호환되지 않아 이동이 어려울 뿐만 아니라, 성공적으로 구현하여도

추가적인 하드웨어 처리 부담이 가중되게 된다. 이를 해결하기 위해 3D 오브젝트를 구성하고 있는 모든 정점(vertex)의 시간에 따른 이동 경로를 각각 2D 텍스처의 형태로 저장하여 애니메이션을 구현하는 방법이 버텍스 애니메이션 텍스처이다. Kang 등은 버텍스 애니메이션 텍스처 기법을 활용하여 애니메이션 처리로 인한 CPU의 보틀넥(bottleneck)을 해결하여 10만 마리 정도의 동물 떼의 이동을 시뮬레이션하였다[8].

본 연구에서는 LoD 및 버텍스 애니메이션 텍스처 기반의 3D 캐릭터 렌더링 최적화를 구현하기 위해 언리얼 엔진의 나이아가라 파티클 시뮬레이션 시스템을 택했다. 이에 대한 실험을 진행하고 결과를 도출하기 위해 3장에서는 언리얼 엔진 기반으로 최적화 성능을 측정하기 위한 실험 준비에 관한 내용을, 4장에서는 실질적으로 본 연구에서 제안하는 방법을, 5장에서는 시뮬레이션 결과에 대한 세부 분석을, 6장에서는 결론 및 향후의 연구 방향을 기술한다.

Ⅲ. 실험 준비

수만 명 이상을 동시에 시뮬레이션할 수 있는 환경을 구축하기 위해 언리얼 엔진 5를 메인 개발 환경으로 선택하였다. 전반적인 개발 환경을 제공하는 통합 엔진으로 게임 개발뿐만 아니라 영화 제작, 시뮬레이션, 건축 등의 다양한 분야에서 활용되고 있으며 최적화 면에서 뛰어난 성능을 보여주는 게임 엔진이다[9]. 언리얼 엔진 5 내에서 시뮬레이션에 사용될 캐릭터의 3D 모델을 제작하기 위해 브이로이드 스튜디오(VRoid Studio)[10]를 활용하여 가상의 캐릭터를 제작하여 임포트(import) 하였으며, LoD 도입에 의한 캐릭터 모델의 폴리곤 수 조절 및 버텍스 애니메이션을 구현하기 위해 블렌더(Blender)[11]라는 무료 3D 오픈소스 모델링 소프트웨어를 활용하였다.

구현에 사용된 메타버스 환경의 구축 및 시뮬레이션은 Intel Core i9 12900KS CPU, 128GB DDR5 4000MHz 4채널 메모리, NVIDIA GeForce RTX 3090Ti의 그래픽카드가 탑재된 시스템에서 이루어졌으며, 언리얼 엔진 버전 5.3.2에서 C++ 기반의 블루프린트 비주얼 스크립팅(Visual Scripting)을 이용하여 액터(actor), 모듈 등을 구성하였다.

그림 1은 시뮬레이션 환경을 구축하기 위한 과정을 순서도로 표현한 그림이다. 작업 과정은 크게 언리얼 엔진 5, 블렌더, 브이로이드 스튜디오 과정으로 나눌 수 있다. 시뮬레이션 과정을 작업하는 언리얼 엔진 5에서 시뮬레이션의 가장 기본이 되는 환경 구축을 진행한 후 캐릭터 모델링을 브이로이드 스튜디오를 통해 가상 3D 캐릭터를 제작하여 언리얼 엔진 5의 플레이어 캐릭터의 생김새 변경 및 애니메이션을 설정하는 데에 사용한다. 다음으로 새로 제작한 캐릭터에 적용된 애니메이션 및 LoD 데이터를 추출하여 블렌더에서 작업을 통해 각각의 LoD 레벨마다 버텍스 애니메이션이 적용된 캐릭터 모델을 제작한다. 마지막으로 생성된 모델들을 언리얼 엔진 5로 가져와 나이아가라 파티클 시뮬레이션에서 사용할 수 있도록 설정 후 사용자 정의 LoD 모듈을 제작하여 최적화 시뮬레이션 환경을 구축한다.

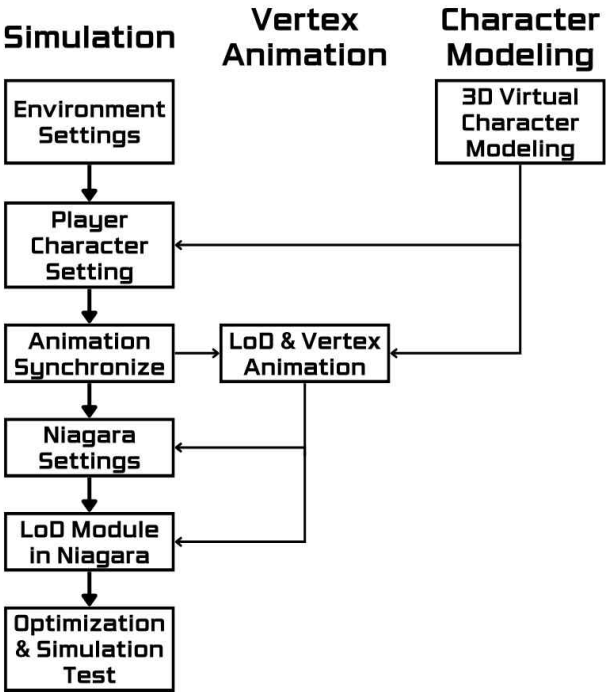

그림 1. 시뮬레이션 환경 구축 순서도
Fig. 1. Flow chart of simulation environment development

3-1 실험 환경 설정

언리얼 엔진 5를 이용해 대규모 공연장을 시뮬레이션할 수 있는 환경을 구축하기 위해 언리얼 엔진 버전 5.3.2를 채택하였다. 에픽게임즈 런처를 다운로드 후 에픽게임즈 계정을 통해 로그인을 마치면 라이브러리 탭에서 원하는 언리얼 엔진의 버전을 다운로드 할 수 있다[12]. 언리얼 엔진 버전 5.3.2의 설치를 마친 후 실행시켜 게임 탭의 삼인칭 템플릿을 선택한 후, 디폴트 설정으로 새 프로젝트를 생성하였다.

프로젝트 생성 시 기본으로 맵 상에 배치되어 있던 불필요

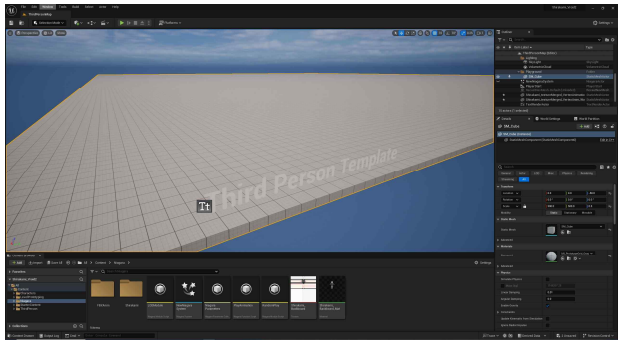

그림 2. 언리얼 엔진 5 에디터 내 시뮬레이션 환경
Fig. 2. Simulation environment in Unreal Engine 5

한 오브젝트들을 제거한 후 충분한 공간 확보를 위해 바닥 오브젝트의 스케일(scale)을 X:500 Y:500으로 확장하여 그림 2와 같은 시뮬레이션 기본 환경을 구축하였다.

3-2 플레이어 및 NPC 캐릭터 설정

언리얼 엔진 5에서 제공하는 기본 캐릭터 모델들을 활용해 시뮬레이션에 활용해도 무방하지만, 생김새가 미래형 공상과학 휴머노이드에 가까운 형태이기에 이번 연구에서는 조금 더 친근감 있는 메타버스 가상 환경에 최적화된 캐릭터 모델을 도입하는 동시에 모델의 임포트 단계부터 세부 설정 컨트롤을 원활하게 진행하기 위해 브이로이드 스튜디오를 통해 메인 캐릭터를 생성하였다.

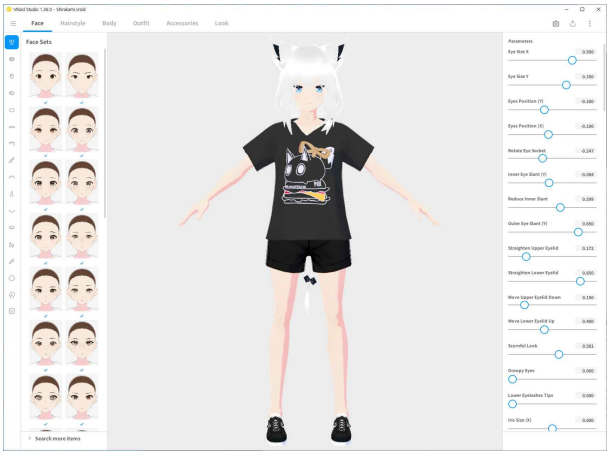

그림 3. 브이로이드 스튜디오를 이용한 캐릭터 3D 휴머노이드 모델 커스터마이징

Fig. 3. 3D Humanoid model customizing using VRoid studio

브이로이드 스튜디오는 그림 3과 같이 두상부터 발 끝까지 모든 요소를 직접 커스터마이징을 통해 원하는 형태의 캐릭터를 생성하여 다양한 목적으로 활용 가능한 무료 소프트웨어이다[13]. 작성한 3D 캐릭터를 VRM 확장자를 가진 파일로 익스포트(export)할 수 있으며, 이 확장자는 glTF2.0을 기반으로 하여 여러 프로그램에서 자유롭게 활용할 수 있다[14]. 익스포트된 VRM 확장자의 3D 캐릭터 모델을 언리얼 엔진 5에서 임포트하여 사용하기 위해서 VRM4U[15]라는 플러그인을 언리얼 엔진 5에 설치하여 해당 임포트 과정을 진행하였다.

임포트 과정 진행 후 언리얼 엔진 5 내의 콘텐츠 브라우저(contents browser)에 머티리얼 인스턴스(material instance), 스켈레탈 메시(skeletal mesh), 텍스처(texture) 등의 에셋(asset)들이 자동으로 생성되는 것을 확인할 수 있다. 제작한 캐릭터의 걷기, 달리기, 점프 등의 애니메이션을 생성하기 위해 언리얼 엔진 5에서 자체적으로 제공되는 IK(Inverse Kinematics) 리타기터(IK retargeter)를 통해 그림 4와 같이 언리얼 엔진 5의 기본 캐릭터 모델의 리깅(rigging) 정보와 브이로이드 스튜디오에서 제작하여 임포트한 캐릭터의 리깅 정보를 실시간으로 동기화하여 시뮬레이션하였다.

애니메이션 동기화까지 마친 캐릭터 모델을 플레이어 캐릭터로 설정하기 위해 현재 메인 플레이어 캐릭터 블루프린트(blueprint) 내의 좌측 계층 구조 중 스켈레탈 메시 컴포넌트(skeletal mesh component)의 하위 자식으로 다른 스켈레탈 메시 컴포넌트를 추가하여 해당 컴포넌트의 스켈레탈 메시 에셋(skeletal mesh asset) 슬롯에 제작한 캐릭터 모델을 추가한 후, 기존 스켈레탈 메시 컴포넌트의 렌더링 옵션에서 '표시' 항목의 체크를 해제하였다. 그 후 시뮬레이션 실행 시 플레이어의 모습이 정상적으로 반영되어 이동하거나 점프 등의 액션을 취할 시 실시간으로 애니메이션이 동기화되는 것을 확인하였다.

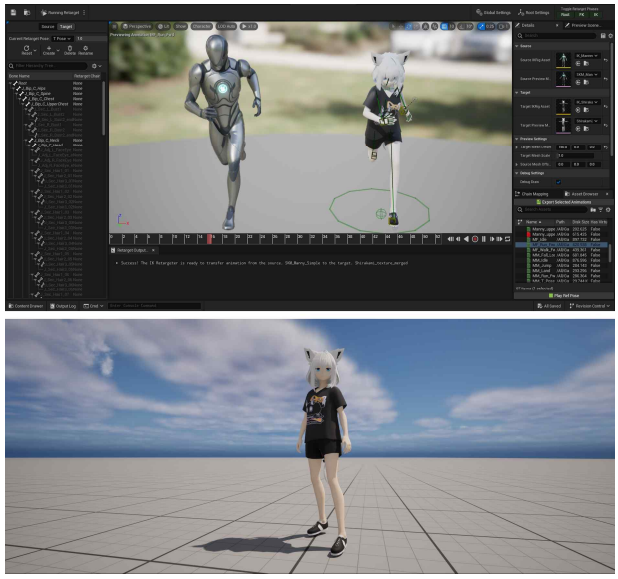

그림 4. 언리얼 엔진 5로 임포트된 브이로이드 스튜디오 커스텀 캐릭터(위: IK 리타기팅을 통해 기존 모델과 IK 릭 정보를 동기화한 모습, 아래: 맵에 배치된 최종 커스텀 플레이어 캐릭터의 모습)

Fig. 4. VRoid studio custom character imported in Unreal Engine 5 (top: IK rig information synchronized with existing model using IK retargeting, bottom: final custom player character placed on the map)

Ⅳ. 실 험

4-1 나이아가라 파티클 시뮬레이션

공연장이나 경기장과 같이 대규모의 관중들이 있는 메타버스 플랫폼을 구현하고자 할 경우, 통상적인 방법으로 단순히 맵 전체에 각각의 고유 객체로 배치하게 되면 최적화 측면 및 전체 통제 용이성 측면 모두 효율적이지 않기 때문에 VFX

(Visual Effects)를 통해 구현하였다.

언리얼 엔진에서는 VFX를 구현하기 위한 차세대 VFX 시스템인 나이아가라(Niagara) VFX 시스템을 제공한다. 이 나이아가라 시스템을 통한 폭죽, 눈, 번개 등과 같은 시각적 효과들의 구현은 기본적으로 이미터(emitter)에서 생성되는 파티클(particle)들을 여러 모듈(module)들을 통해 생김새에 변형을 가하거나 전체적인 흐름을 제어하는 방식으로 이루어진다. 나이아가라를 이용한 VFX 구현에 핵심이 되는 파티클들은 단순한 스프라이트(sprite) 형태부터 복잡한 3D 메시(3D mesh)까지 폭넓게 선택하여 사용할 수 있으며 최적화 관련 옵션들이 많아 관리하기 쉽고 커스텀 모듈들을 사용자가 직접 설계해 원하는 기능을 구현할 수도 있다[16].

언리얼 엔진 5에서 새로운 나이아가라 시스템 에셋을 만들어 맵 상의 원하는 위치에 배치해 파티클이 소환될 수 있도록 세팅한 후 그림 5의 좌측과 같이 시스템 내부의 이미터 업데이트, 파티클 스폰(spawn), 파티클 업데이트, 렌더(render) 항목에 필요한 모듈들을 추가하여 기본적인 메타버스 공간상 캐릭터들의 움직임을 구현하였다.

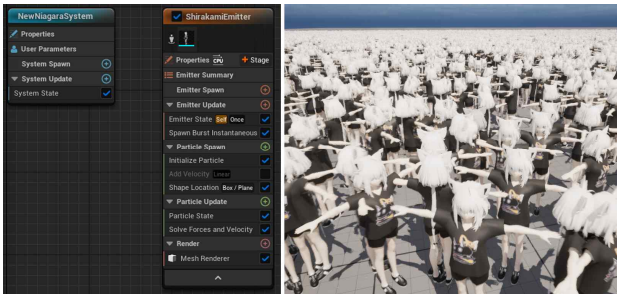

그림 5. 나이아가라 시스템의 파티클 이미터 설정과 맵에 랜덤으로 스폰된 캐릭터들

Fig. 5. Particle emitter settings in Niagara system and randomly spawned characters on the map

뷰포트(view port) 상에 표시되는 파티클들의 형태를 렌더 항목에서 새 렌더러(renderer)를 추가하여 결정할 수 있으며 3D 모델 오브젝트를 렌더하기 위해 메시 렌더러(mesh renderer)를 추가하여 원하는 모델을 선택할 수 있다. 그러나 현재 나이아가라 시스템은 메시 렌더러의 메시 오브젝트로 스태틱 메시(static mesh) 오브젝트만 선택이 가능하기 때문에 스켈레탈 메시로 구성된 캐릭터를 변형이 일어나지 않는 정적인 오브젝트로 정의된 스태틱 메시 오브젝트로 변형시켜 주어야 한다.

앞서 설명한 VRM 확장자 파일의 경우 glTF 기반으로 이루어져 있기 때문에 확장자를 .glb 혹은 .gltf로 변경하면 3D 모델 편집 소프트웨어인 블렌더로 임포트할 수 있다.

스태틱 메시 오브젝트는 폴리곤 및 텍스처 정보만을 필요로 하기 때문에 임포트한 모델에서 포즈(pose) 정보, 본(bone) 정보, 엠티(empty) 등을 제거해 준 후 남은 메시들을 모두 머지(merge) 작업을 통해 하나의 메시로 합쳐주어 그림 6과 같은 상태로 만들어 준 후 모델을 FBX 확장자를 가진 파일로 익스포트하여 언리얼 엔진을 포함한 대부분의 3D 프로그램에서 임포트할 수 있는 파일을 생성하였다. 캐릭터의 텍스처의 경우 블렌더의 'Unpack Resources' 기능을 통해 모델에 종속되어 있던 텍스처를 이미지 파일로 저장하였다.

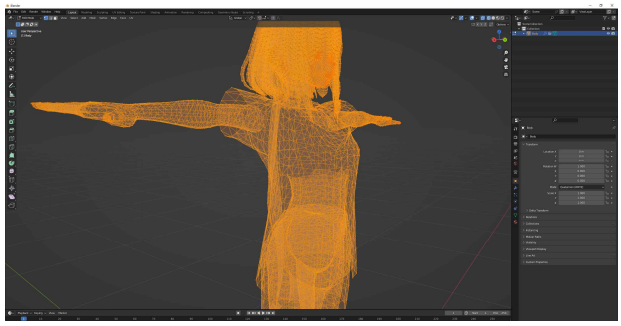

그림 6. 블랜더 3D 프로그램을 통해 스켈레탈 메시에서 스태틱 메시로 변형시킨 캐릭터의 모습

Fig. 6. Look of the character exchanged from skeletal mesh to static mesh using Blender 3D program

해당 FBX 파일을 언리얼 엔진 5로 임포트한 후 자동 생성된 기본 머티리얼 에셋에 추출한 텍스처 이미지를 연결하여 T자 형태의 포즈로 고정된 스태틱 메시 캐릭터 에셋을 완성하였다. 이 에셋을 앞서 생성한 나이아가라 시스템의 메시 렌더러에 추가한 후 시뮬레이션을 실행하면 그림 5의 우측과 같이 맵 상에서 수만 명의 캐릭터들이 돌아다니는 것과 같은 시각적 효과를 얻을 수 있다.

4-2 LoD 설정

현재까지의 진행으로 현실의 대규모 공연장에서 흔히 볼 수 있는 혼잡한 인파의 구현은 성공하였다. 그러나 파티클들이 밀집된 곳으로 시점을 이동하면 화면의 FPS가 20 이하로 하락하는 최적화 문제 및 캐릭터들이 전부 T자 포즈인 상태로 이동한다는 문제가 존재한다.

먼저 FPS 문제를 해결하기 위해 LoD(Level of Detail) 개념을 나이아가라 시뮬레이션에 도입하고자 한다. 플레이어와의 거리가 멀어질수록 오브젝트의 폴리곤 수를 낮춰 그래픽 렌더링의 부담을 덜어주는 LoD 최적화 기법을 나이아가라 시스템 내에서 커스텀 모듈을 설계하여 해당 기능을 수행할 수 있도록 하였다.

그림 7과 같이 커스텀 나이아가라 모듈 스크립트(script)를 구성하여 생성한 이미터의 파티클 업데이트 항목 아래에 추가하여 이미터가 사용자 정의 모듈을 사용하도록 하였다. 나이아가라 시스템의 이미터에는 여러 개의 렌더러가 사용될 수 있으며 각 렌더러마다 렌더러 비저빌리티(renderer visibility) 값을 설정할 수 있는 항목이 있어 해당 값과 일치하는 파티클들만이 해당 렌더러에 의해 표시되게 된다. 플레

이어 카메라의 위치값을 중심으로 거리 1500 이내에 존재하는 파티클들의 렌더러 비저빌리티 값을 0으로, 거리 1500 이상 8000 이내에 존재하는 파티클들은 1, 마지막으로 거리 8000보다 멀리 떨어진 파티클들은 2로 설정하도록 노드들을 배치하여 모듈을 구성하였다.

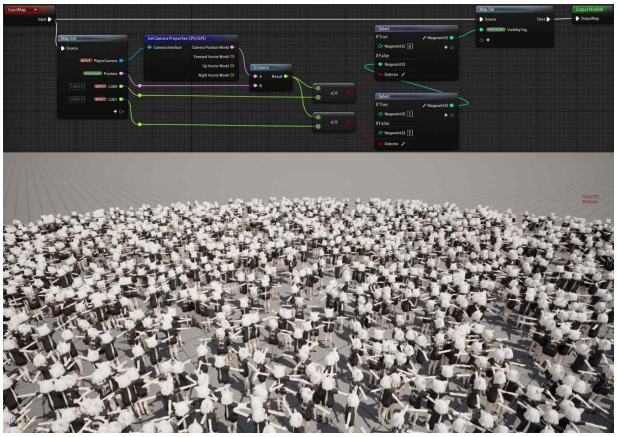

그림 7. 나이아가라 커스텀 모듈을 통해 LoD를 구현한 모습
Fig. 7. LoD implemented by Niagara custom module

4-3 버텍스 애니메이션 설정

나이아가라 시스템은 파티클 렌더러 3D오브젝트로 정적 메시만을 지원하므로 각각의 캐릭터 파티클들이 움직일 때 걷거나 뛰는 애니메이션을 도입하기 위해서는 스켈레탈 메시가 아닌 스태틱 메시에서 애니메이션을 구현할 수 있는 방법을 사용해야 한다. 버텍스 애니메이션은 키프레임 방식이 아닌 애니메이션 데이터를 2D 텍스처나 메시의 UV 자체에 저장하는 방식을 사용하여 불필요한 애니메이션 부하를 줄이면서 스태틱 메시에도 애니메이션을 적용할 수 있도록 해준다[17]. 버텍스 애니메이션 기능을 도입하기 위해 외부 프로그램을 통해 해당 텍스처를 생성하여 언리얼 엔진으로 임포트해 사용할 수 있다.

언리얼 엔진 5에서는 폴리곤의 개수를 조절하여 서로 다른 LoD 레벨을 생성하는 것이 가능하여 스켈레탈 메시 혹은 스태틱 메시 내의 에셋 디테일 탭에서 LoD 옵션을 설정해 원하는 LoD 레벨의 개수만큼 언리얼 엔진 5에서 자동으로 생성되게 할 수 있다. LoD 도입과 함께 버텍스 애니메이션 구현을 위해 캐릭터 모델의 LoD 개수를 원하는 수만큼 설정 후 생성하여 에셋 익스포트 기능을 통해 FBX 파일로 익스포트해서 블렌더에서 임포트하여 편집하였다.

캐릭터에게 원하는 애니메이션을 설정하기 위해 버텍스 애니메이션에 사용하고자 하는 애니메이션 시퀀스(sequence) 에셋을 언리얼 엔진 5 내에서 FBX 파일로 익스포트 후 블렌더로 임포트하여 링크 애니메이션 데이터(link animation data) 기능을 통해 미리 임포트 해둔 캐릭터와 원하는 애니메이션 데이터를 동기화하였다. 동기화한 애니메이션이 적용된 캐릭터 모델에 포함된 여러 LoD 레벨 중 원하는 레벨의 모델을 블렌더 버텍스 애니메이션 오픈소스 플러그인 Unreal Tools[18]를 사용하여 그림 8과 같이 언리얼 엔진 5 내에서 버텍스 애니메이션 구현에 필요한 Export Mesh, Normal map, Offset map을 생성하였다.

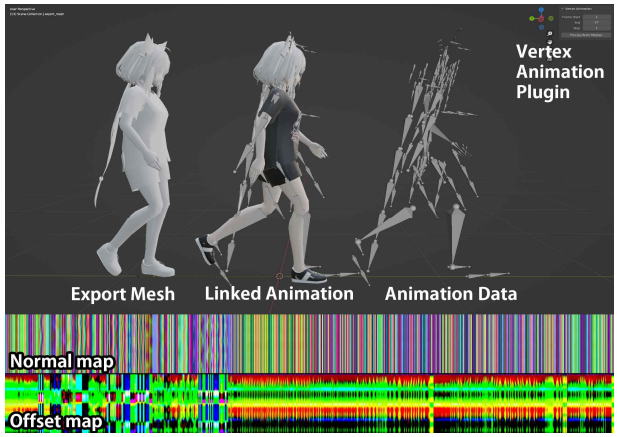

그림 8. 블랜더를 통해 생성한 버텍스 애니메이션 데이터
Fig. 8. Vertex Animation Data created in Blender

Export Mesh는 버텍스 애니메이션을 적용시키고자 하는 최종 출력 스태틱 메시이며, Normal map과 Offset map은 키프레임의 애니메이션 데이터가 2D 텍스처 형태로 변환된 것으로 언리얼 엔진 5 내에서 스태틱 메시에 버텍스 애니메이션을 적용시킬 때 사용된다. Export Mesh는 FBX 파일로, Normal map은 BMP 파일로, Offset map은 EXR 파일로 각각 블랜더 내에서 익스포트하여 언리얼 엔진 5로 임포트하였다.

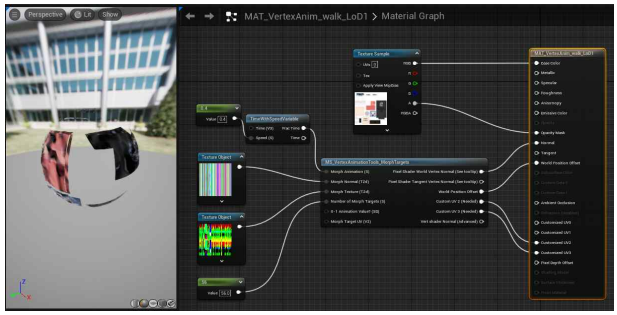

그림 9. 언리얼 엔진 5 내에서 머티리얼 노드를 사용하여 구현한 버텍스 애니메이션
Fig. 9. Vertex Animation implemented using material nodes in Unreal Engine 5

임포트한 두 종류의 2D 텍스처를 언리얼 엔진 5에서 제공하는 MS_VertexAnimationTools_MorphTargets 노드를 중심으로 그림 9와 같이 머티리얼을 구성하였다. 이 노드를 통해 최종 머티리얼 노드의 월드 포지션 오프셋(world position offset) 및 노멀(Normal) 값, 애니메이션 재생 속도 등을 설정하여 자연스러운 애니메이션이 되도록 조정하였다.

마지막으로 색상 설정 부분에 기존 모델에서 사용하던 텍스처를 그대로 가져와 베이스 컬러(base color) 및 오파시티 마스크(opacity mask) 항목에 사용할 수 있으며, 버텍스 애니메이션 관련 설정 항목들에는 영향을 미치지 않기 때문에 하나의 텍스처로 색 및 애니메이션 표현을 동시에 처리하게 되어 최적화 성능면에서 큰 이점으로 작용한다.

그림 10. 나이아가라 파티클 시스템의 캐릭터 파티클들에 성공적으로 버텍스 애니메이션이 적용된 모습
Fig. 10. Vertex animation successfully applied to character particles of the Niagara particle system

나이아가라 시스템 내 이미터의 렌더러에 할당되어 있는 기존 모델을 버텍스 애니메이션을 도입한 새 스태틱 메쉬로 변경시켜 준 후 시뮬레이션을 실행해 그림 10과 같이 모든 캐릭터 파티클들에 정상적으로 걷기 애니메이션이 추가된 것을 확인하였다. LoD 레벨이 더 낮은 캐릭터 모델들도 같은 방법으로 버텍스 애니메이션이 설정된 모델로 생성 후 임포트하여 이전 단계에서 구성한 나이아가라 시스템 내 다른 렌더러 비저빌리티 값을 갖는 렌더러에 각각 할당하여 거리별로 다른 LoD 레벨 캐릭터 모델이 렌더링될 수 있도록 하였다. LoD2의 경우 화면상에서 각 파티클 당 전체 면적이 수 픽셀 이내로밖에 표시되지 않는 원거리 범위에 존재하기 때문에 스프라이트 렌더러를 사용하여 중요도가 비교적 떨어지는 부분에 대한 그래픽 자원 낭비를 최소화하였다.

그림 11. LoD 및 버텍스 애니메이션이 모두 적용된 전체 시뮬레이션 환경
Fig. 11. Whole simulation environment with both LoD and vertex animation applied

최종적으로 그림 11과 같이 버텍스 애니메이션이 적용된 모든 캐릭터 하나하나가 실제로 메타버스 공간상을 걸어다니는 것과 같은 모습으로 움직일 수 있으면서 LoD를 도입함으로써 최적화 문제를 해결하여, 4만 명 이상의 플레이어를 동시에 시뮬레이션하여도 FPS가 안정적으로 60 이상을 유지하는 모습을 확인할 수 있었다.

V. 시뮬레이션 결과 및 분석

현재까지의 단계를 거쳐 완성된 시뮬레이션 환경을 통해 클라이언트 단에서 한 공간 안에 동시에 4만 명을 안정적으로 수용할 수 있게 되었지만, 설정값 변경을 통한 추가적인 최적화 가능성과 더불어 과도한 디테일 저하로 인한 불쾌감 발생 우려 등을 평가하기 위해 언리얼 엔진 5 내의 렌더링 설정, LoD의 거리, 나이아가라 파티클 시뮬레이션 설정 등의 수치 조정을 통해 플레이어 수에 따른 FPS를 측정하여 비교 및 분석을 진행하였다.

표 1은 시뮬레이션 환경을 구축하면서 사용자가 기호에 맞게 임의로 변경하여 적용할 수 있는 언리얼 엔진 5 내 값들 중 시뮬레이션 FPS에 직접적인 영향을 줄 가능성이 있는 값들을 본 연구에서 기본 설정값으로 정한 수치를 나타낸 표이다. LoD0 distance와 LoD1 distance는 나이아가라 파티클 시뮬레이션 내의 사용자 지정 모듈을 통한 LoD 구현에 쓰인 값으로, 플레이어로부터 어느 정도 떨어진 거리부터 해당 레벨의 LoD가 설정될지를 변경할 수 있다. Niagara Sim Target 설정에서는 나이아가라 파티클 시뮬레이션 시스템이 CPU 시뮬레이션으로 실행될지 GPU 시뮬레이션으로 실행될지 지정할 수 있으며 기본값은 CPU를 사용하여 시뮬레이션하도록 설정하였다. Niagara Object Detail – Cast Shadow 항목은 시뮬레이션 환경의 맵 상에 놓여있는 나이아가라 시스템 오브젝트 자체의 Detail 탭에서 변경할 수 있는 값으로 해당 오브젝트가 지면에 그림자를 드리울지를 설정할 수 있어 현실적인 시뮬레이션 환경 구축을 위해 기본값을 true로 설정하여 그림자가 지도록 하였다. 마지막으로 Niagara Particle Spawn Plane Size는 시뮬레이션에서 캐릭터로 표현된 파티클들이 소환되는 영역의 크기를 지정해 줄 수 있으며 기본값을 20,000 X 20,000의 평면으로 설정하였다.

표 1. 시뮬레이션 환경의 기본 설정값들
Table 1. Default values of simulation environment

Name	Default Value
LoD0 distance	1500
LoD1 distance	8000
Niagara Sim Target	CPUSim
Niagara Object Detail – Cast Shadow	true
Niagara Particle Spawn Plane Size	20,000 X 20,000

5-1 LoD distance에 따른 FPS

먼저 나머지 설정값은 고정시킨 후 LoD0 distance와 LoD1 distance 값의 조절을 통해 어느 정도의 성능 차이가 있는지, 디테일 저하로 인한 불쾌감이 느껴지지는 않는지 등

을 평가하여 본다. 표 2는 각각의 LoD 적용 거리별 FPS의 변화를 측정하기 위해 준비한 비교 집단들의 설정값이다. 기본값은 각각 1500과 8000으로 앞서 구축한 시뮬레이션 환경의 값이다. A와 B는 각각 LoD1 distance를 기본값으로 둔 채 플레이어와 가까운 LoD0 모델 및 LoD1 모델의 전환 경계인 LoD0 distance 값을 가깝게, 멀게 설정한 비교 집단이고 C와 D는 반대로 각각 LoD0 distance는 기본값으로 고정한 채 플레이어와 멀리 떨어져 있는 LoD1 모델 및 LoD2 모델 사이의 전환 경계인 LoD1 distance 값을 가깝게, 멀게 설정한 비교 집단이다. E와 F는 각각 LoD0 distance와 LoD1 distance를 동시에 플레이어에게서 가깝게, 멀게 설정한 비교 집단이다. 마지막으로 G는 LoD를 아예 적용시키지 않은 비교 집단이다.

그림 12는 각 비교 집단에 대해서 각각 나이아가라 시스템

표 2. LoD 거리 설정값에 따른 비교 집단
Table 2. Comparison groups according to LoD distance settings

Group	LoD0 distance	LoD1 distance
Default	1500	8000
A	100	8000
B	7900	8000
C	1500	1600
D	1500	16000
E	100	200
F	8000	16000
G	N/A	N/A

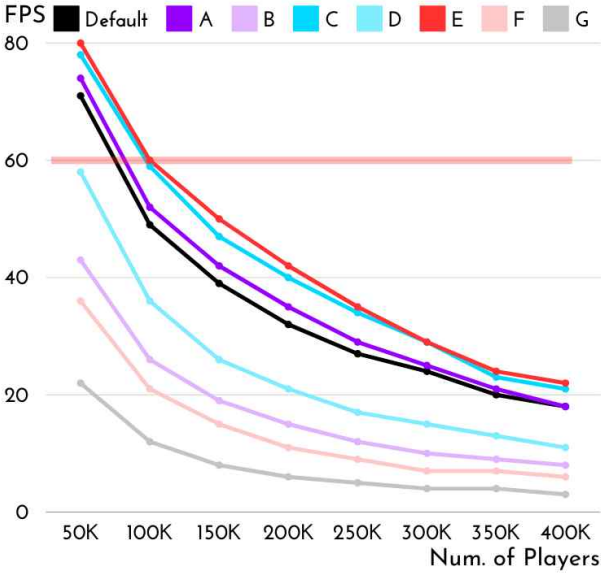

그림 12. LoD 거리 및 플레이어 수에 따른 시뮬레이션 FPS
Fig. 12. Simulation FPS by LoD distance and number of players

내 사용자 지정 모듈의 LoD0 distance 및 LoD1 distance 수치를 조절한 후 플레이어 수를 5만 명에서 시작하여 5만 명씩 늘려가며 총 40만 명까지 증가시켰을 때 화면에 표시된 실시간 시뮬레이션의 FPS를 기록한 그래프이다. 가로축은 맵 상에 소환된 총 플레이어 수를, 세로축은 FPS를 나타내며 본 연구에서 목표 FPS로 설정한 60FPS의 기준선을 빨간 가로선으로 표시하였다.

먼저 LoD 자체를 설정하지 않은 G에 비해 LoD를 기본값으로 적용한 Default의 경우 약 223%의 FPS 향상 효과를 보였다. 전체적으로 기본값에 비해 각각의 LoD 거리를 플레이어에게 가깝게 이동시킨 A, C, E의 경우, FPS 향상 효과가, 플레이어에게서 멀게 이동시킨 B, D, F의 경우, FPS가 하락하는 것을 확인할 수 있었다. E의 경우 LoD0 distance와 LoD1 distance 모두 플레이어에게 매우 근접한 수치로 설정하여 최대의 FPS 향상 효과를 얻어 10만 명의 플레이어를 동시에 시뮬레이션하는 상황에서도 60FPS의 기준을 달성하여 기본값 대비 약 15%의 향상 효과를 보였으나, 너무 낮게 설정된 LoD 거리로 인해 그림 13과 같이 플레이어와 상당히 근접한 파티클만 디테일이 가장 높은 LoD0 모델로 표시가 되고, 조금만 멀어져도 바로 눈에 띌 정도로 디테일이 저하되는 것을 볼 수 있다.

그림 13. 낮은 LoD 거리로 인한 디테일 저하
Fig. 13. Detail degradation due to low LoD distance

5-2 나이아가라 설정에 따른 FPS

LoD 거리에 따른 FPS 성능 차이를 살펴보았지만 각각의 LoD 거리를 최솟값으로 설정하여도 한 공간 안에 10만 명을 동시에 시뮬레이션하면서 60FPS의 기준을 간신히 달성하는 데에 그쳤다. 다음으로 파티클들의 LoD 거리를 기본값으로 고정시킨 후 나이아가라 시스템 자체의 설정값 조절을 통해 성능 차이를 평가해 본다. 표 3의 ①, ②, ③은 표 1에서 LoD 설정을 제외한 값들로, ①의 기본값은 CPU 시뮬레이션이며 GPU 시뮬레이션으로 수정할 수 있고, ②의 기본값은 그림자를 생성하는 true이며 그림자를 생성하지 않게 false로 변경할 수 있으며, ③의 기본값은 20,000 X 20,000의 크기에 파티클들을 소환하도록 설정하였으며 50,000 X 50,000 크기의 영역에 소환되도록 수정할 수 있다.

이 설정값들의 수정 사항 적용 여부로 기본값을 포함한 A부터 G까지의 비교 집단을 구성하였다. A, B, C는 각각의 수정 사항을 하나씩만 반영한 비교 집단이고, D, E, F는 두 개씩, 마지막으로 G는 세 가지의 수정 사항을 모두 반영한 비교 집단이다.

표 3. 나이아가라 설정값에 따른 비교 집단
Table 3. Comparison groups according to Niagara settings

Number	Parameter	Default	Modified
①	Sim Target	CPUSim	GPUCompute Sim
②	Cast Shadow	true	false
③	Spawn Plane Size	20,000X20,000	50,000X50,000
Group	Modification		
Default	N/A		
A	①		
B	②		
C	③		
D	① + ②		
E	② + ③		
F	① + ③		
G	① + ② + ③		

그림 14는 각 비교 집단에 대해서 각각 지정된 나이아가라 시스템의 설정 수정값들을 적용한 후 플레이어 수를 10만 명에서 시작해 10만 명씩 증가시켜 80만 명까지 증가시켰을 때 화면에 표시된 FPS를 기록한 그래프이다. 마찬가지로 가로축은 맵 상에 소환된 총 플레이어 수를, 세로축은 FPS를 나타내며 본 연구에서 목표 FPS로 설정한 60FPS의 기준선을 빨간 가로선으로 표시하였다.

나이아가라 시스템 설정 수정값들을 한 가지 이상 적용한

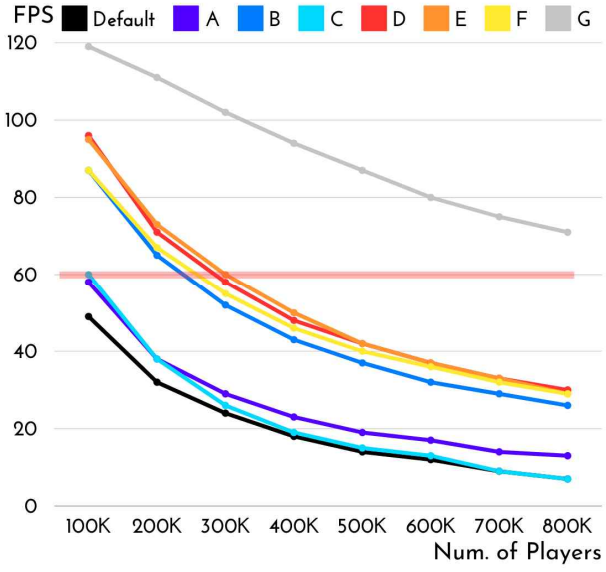

그림 14. 나이아가라 설정값 및 플레이어 수에 따른 시뮬레이션 FPS
Fig. 14. Simulation FPS by Niagara settings and number of players

시뮬레이션 환경은 대부분 기본값보다 나은 성능을 발휘하였다. 한 가지 수정 사항들만 반영한 A, B, C 중 B의 FPS 향상 폭이 거의 두 가지 수정값을 반영한 비교 집단에 준할 만큼 독보적으로 높은 것으로 보아 그림자 적용의 유무가 시뮬레이션 환경에 영향을 가장 많이 주는 것을 볼 수 있다. A를 통해서는 나이아가라 파티클 시뮬레이션을 GPU 시뮬레이션으로 할 경우, CPU보다 조금 더 나은 성능을 보인다는 것을 알 수 있었다. 마지막으로 C를 통해 파티클이 소환되는 영역의 크기는 플레이어의 수가 비교적 적을 때 가장 효과적인 것을 확인하였고 플레이어 수가 증가함에 따라 급격히 성능이 하락하는 모습을 보였다.

두 가지의 수정값을 적용한 비교 집단 D, E, F의 경우 대부분 한 가지 수정값만을 적용한 집단에 비해 훨씬 나은 FPS 향상을 보였지만 세 가지 수정값 중 어느 두 가지를 조합하여도 서로 크게 차이가 나지 않는 결과를 볼 수 있었다. 세 가지 수정값을 모두 적용한 G의 경우 플레이어 수가 80만 명까지 증가하였음에도 여전히 기준값인 60FPS 이상을 유지하는 모습을 보여 기본값 대비 914%의 성능 향상을 확인하였다.

Ⅵ. 결 론

본 논문에서는 대규모의 메타버스 플랫폼 개발에 문제가 되는 클라이언트 측 렌더링 자원 부족 문제 및 서버 측 트래픽 부하 문제 중 클라이언트 측 최적화 문제 해결 방안을 제시하였다. 브이로이드 스튜디오를 통해 커스터마이징한 캐릭터를 제작해 언리얼 엔진 5로 임포트하여 사용함으로써 관리의 용이성을 확보하였고, 언리얼 엔진 5에서 제공하는 나이아가라 VFX 시뮬레이션 시스템을 활용하여 대규모의 관중들을 시뮬레이션하였으며, Blender를 비롯한 외부 툴 및 플러그인들의 도움으로 LoD 및 버텍스 애니메이션 구현에 성공하여 최종적으로 목표로 했던 수만 명 이상의 플레이어를 시뮬레이션하면서 60 FPS 이상을 유지하는 조건을 성공적으로 만족시킬 수 있었다. 나아가 LoD의 거리 및 나이아가라 시스템의 렌더링 설정을 조정하여 FPS를 측정한 결과 크게 사용자 경험을 해치지 않는 선에서 80만 명에 준하는 플레이어들을 동시에 수용할 수 있는 환경을 구축하였다.

향후, 언리얼 엔진 5에서 제공하는 리플리케이트(replicate) 통신 방법[19]이나 외부 플러그인을 활용하는 TCP 통신 방법 등을 통해 서버와의 연동과 동시에 본 시뮬레이션과 거의 동일한 대규모 메타버스 플랫폼 환경을 경험할 수 있는 통신 환경을 구축하는 것을 목표로 할 계획이다. 캐릭터들의 이동 동작도 현재는 걷기 동작만 존재하며, 모든 캐릭터 파티클들의 애니메이션 재생 모습이 서로 완전히 동일하여 어색함이 느껴질 수 있는 문제가 존재하기 때문에 추후 연구를 통해 달리기, 점프, 감정 표현 등 다양한 애니메이션을 추가하여 무작위로 서로 다른 동작을 하도록 설계하는 방안을 모색하여 더욱 현장감

넘치는 시뮬레이션을 구현하고자 한다. 나이아가라의 사용자 정의 모듈 또한 추가적인 설계를 통해 실제 공연장에서 공연자의 신호에 맞춰서 이루어지는 단체 환호성이나 파도타기 혹은 월드컵 경기장에서의 단체 응원 동작과 같은 대규모 집단 동작에 대해 동일 사용자 경험을 메타버스 환경에서 제공할 수 있도록 개발을 진행할 계획이다.

감사의 글

이 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 지역지능화혁신인재양성사업(IITP-2024-00156287, 기여율: 50%)과 문화체육관광부 및 한국콘텐츠진흥원의 2022년도 문화기술 연구개발 사업으로 수행되었음(과제명 : 대형 공연장 규모의 실시간 양방향 메타버스 체험 플랫폼 기술 개발, 과제번호 : RS-2022-050002, 기여율: 50%)

참고문헌

[1] B.-K. Lee, “The Metaverse World and Our Future,” *The Korea Contents Association Review*, Vol. 19, No. 1, pp. 13-17, 2021.

[2] S. Seong, “A Study on R&D Trends and Propects of Metaverse,” *HCI 2008*, South Korea, pp. 1450-1457, 2008.

[3] H. S. Lim and C. W. Jung, “Effects of Meteverse Gaming Platforms: Focusing on Genre-Based Comparison,” *Journal of the Korea Contents Association*, Vol. 24, No. 1, pp. 209-219, January 2024. https://www.doi.org/10.5392/JKCA.2024.24.01.209

[4] Y.-J. Kim, “A Study on the Effect of Polygon Reduction on 3D Object Shape and Composition,” *Cartoon and Animation Studies*, No. 55, pp. 273-291, June 2019.

[5] J.-W. Lee and Y. Kim, “Rendering Performance Evaluation of 3D Games with Interior Mapping,” *Journal of Korea Game Society*, Vol. 19, No. 6, pp. 49-59, December 2019. https://www.doi.org/10.7583/JKGS.2019.19.6.49

[6] A. Aubel, R. Boulic, and D. Thalmann, “Real-time Display of Virtual Humans: Levels of Details and Impostors,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 10, No. 2, pp. 207-217, March 2000. https://www.doi.org/10.1109/76.825720

[7] A. Wysopal, V. Ross, J. Passananti, K. Yu, B. Huynh, and T. Höllerer, “Level-of-Detail AR: Dynamically Adjusting Augmented Reality Level of Detail Based on Visual Angle,” *IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, Shanghai, China, pp. 63-71, March 2023. https://www.doi.org/10.1109/VR55154.2023.00022

[8] I. Kang, Y.-I. Eom, and J. Han, “A Multi-resolution Technique for Real-time Animation of Large Crowds,” *Computer and Information Sciences – ISCIS*, Turkey, pp. 384-393, 2006. https://doi.org/10.1007/11902140_42

[9] Unreal Engine 5 [Internet]. Available: https://www.unrealengine.com/ko/unreal-engine-5

[10] N. Isozaki, S. Ishima, Y. Yamada, Y. Obuchi, R. Sato, and N. Shimizu, “VRoid Studio: A Tool for Making Anime-like 3D Characters Using Your Imagination,” in *Siggraph Asia 2021 Real-Time Live!*, Tokyo, Japan, December 2021. https://doi.org/10.1145/3478511.3491311

[11] Blender [Internet]. Available: https://www.blender.org/

[12] Epic Games Launcher Download [Internet]. Available: https://www.unrealengine.com/ko/download

[13] VRoid Studio [Internet]. Available: https://vroid.com/en/studio

[14] What is glTF? [Internet]. Available: https://www.khronos.org/gltf/

[15] VRM4U Plugin Download [Internet]. Available: https://github.com/ruyo/VRM4U

[16] Niagara VFX System [Internet]. Available: https://docs.unrealengine.com/4.27/ko/RenderingAndGraphics/Niagara/

[17] Vertex Animation Tool [Internet]. Available: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/Tools/VertexAnimationTool/

[18] Unreal Tools [Internet]. Available: https://github.com/JoshRBogart/unreal_tools?files=1

[19] Actor Replication [Internet]. Available: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Actors/

정광무(Kwang-Moo Chung)

2022년 : 전남대학교 컴퓨터정보통신공학과(학사)

2022년～현　재: 전남대학교 지능전자컴퓨터공학과 석사과정

※관심분야 : 컴퓨터 그래픽스(Computer Graphics), 메타버스(Metaverse), 디지털융합콘텐츠(Digital Convergence Content)) 등

정일권(Il-Kwon Jeong)

1992년 : Electrical Engineering from KAIST, Daejeon, South Korea(공학사)
1994년 : Electrical Engineering from KAIST, Daejeon, South Korea(공학석사)
1999년 : Electrical Engineering from KAIST, Daejeon, South Korea(공학박사)

1999년～현　재: ETRI 초실감메타버스연구소 콘텐츠연구본부 본부장

※관심분야 : hyper-realistic metaverse technology, five-sense emotional interaction, intelligent content understanding, etc.

김진술(Jinsul Kim)

2001년 : Computer Science from University of Utah, Salt Lake City,Utah, USA(공학사)
2005년 : 한국과학기술원 정보통신공학(공학석사)
2008년 : 한국과학기술원 정보통신공학(공학박사)

2005년～2008년: 한국전자통신연구원 IPTV 인프라 기술, 융·복합 방송/통신 분야 연구원
2009년～2012년: 나사렛대학교 멀티미디어학과 교수
2012년～현　재: 전남대학교 지능전자컴퓨터공학과 교수

※관심분야 : QoS/QoE 예측/분석/관리, 모바일 미디어 처리/통신, 클라우드 컴퓨팅 디지털 미디어 및 네트워크 지능기술