Asia-pacific Journal of Convergent Research Interchange
Vol.9, No.9, September 30 (2023), pp.31-38
http://dx.doi.org/10.47116/apjcri.2023.09.03

A Study on Approximating Realism through Utilizing Physical Shader Optimization Techniques

물리적 쉐이더 최적화 기법을 활용한 사실감 근사값 추정에 관한 연구

Kyoung Min Lee[1], Stellar Choi[2], Ha Eun Jung[3] , Seong Geon Bae[4]

이경민[1], 최스텔라[2], 정하은[3], 배성근[5]

[1] Student, Dept of Software Application Virtual Reality Major, Kangnam University, South Korea, mininifanini@gmail.com

[2] Student, Dept of Software Application Virtual Reality Major, Kangnam University, South Korea, stellar010203@gmail.com

[3] Student, Dept of ICT Engineering Virtual Reality Major, Kangnam University, South Korea, codingha00@gmail.com

[4] Professor, Dept of ICT Engineering Virtual Reality Major, Kangnam University, South Korea, sgbae@kangnam.ac.kr

Corresponding author: Kyoung Min Lee

Abstract: Recently, various devices have been developed. It is because hardware used in devices like CPU, GPU has been improved. The demand for games that have high quality graphic is on rise in not only high-spec devices like PC or console but low-spec devices like mobile phone. Also, it is increasing to multi-platform development, especially in game. Existing rolling snow ball animation used in application for low-spec device employ a method of expanding the entire volume of a snow ball regardless of its movement direction. In real life, snow balls never increase their entire volume. They get volume considering their movement direction. In this paper propose a method the rolling snow ball animation looks more realistically than existing method. It is too complicated for real-time applications for a variety of devices to apply for physics related to snow. This paper utilizes the phenomenon with ease. Unity 3D is a multi-platform game engine. And it provides its own shader programming. It works efficiently to implement rolling snowball animation for multi-platform application because shader programming use hardware feature. In this paper, implement the snow ball animation using Unity 3D. As a result, it can offer to users realistic experience even if apply for low-spec devices.

Keywords: Unity Shader, Rolling Snow Ball, Collision Detection, UV Mapping, Spherical Coordinate System

요약: 최근 하드웨어 발전으로 인해 다양한 종류의 디바이스들이 발전됐다. 모바일 등 PC나 콘솔이 아닌 플랫폼 에서도 고성능 그래픽 게임에 대한 수요가 높아졌다. 이러한 이유로 멀티 플랫폼 개발이 증가하고 있다. 기존 모바일 게임 등에서 사용되는 눈덩이 굴리기 방식은 이동 방향과 관계없이 전체 부피가 커지는 방식이다. 제시하는 방법은 보다 사실적인 실시간 눈덩이 굴리기 시뮬레이션을 보여준다. 실제 눈에 적용되는 물리법칙은 매우 복잡하기 때문에

Received: May 19, 2023; 1st Review Result: June 24, 2023; 2nd Review Result: July 27, 2023
Accepted: August 25, 2023

다양한 플랫폼에서 실시간으로 구현하기 어렵다. 본 연구에서는 눈이 굴러갈 때 발생하는 현상을 간단하게 풀어 적용한 시뮬레이션 방법을 제안한다. 이로인해 멀티 플랫폼 콘텐츠에 적용할 수 있는 유니티 쉐이더를 이용한 실시간 눈덩이 굴리기 시뮬레이션을 제작할 수 있을 것이라 예상한다. 또한 저사양 디바이스에서도 보다 사실적인 경험을 제공할 수 있을 것이라 예상한다.

핵심어: 유니티 쉐이더, 눈덩이 굴림 시뮬레이션, 충돌처리, UV 텍스쳐 매핑, 구면좌표계

1. 서론

최근 반도체 성능이 향상 되어 하드웨어와 통합한 최적화 기법이 제시되었다. 계산량이 많은 고성능 그래픽용 컴퓨터는 개인 컴퓨터의 개발과 함께 발전하고 있다. 하드웨어의 발전은 저사양 모바일 기기도 포함된다. 다양한 기기에서 사실적인 컴퓨터 그래픽스 표현이 가능해졌다. 멀티 플랫폼 콘텐츠를 위한 개발의 중요성이 강조되고 있다. 인디 스튜디오를 제외한 스튜디오는 멀티플랫폼 게임을 더 많이 출시하는 추세다. 재정적인 측면에서 투자비용이 적기 때문이다[1]. 유니티는 모바일, VR기기, PC등에서 활용 가능한 멀티 플랫폼 개발 도구이다. 기존 실시간 어플리케이션의 눈덩이 굴리기 애니메이션은 과제를 수행하면 자동으로 커지는 형태로 표현된다. 실제 눈을 완전한 구 형태로 굴리기 위해서는 다양한 방향으로 눈을 굴려야 한다. 눈덩이가 커지는 원리는 눈덩이에 지면의 눈이 붙으며 부피가 늘어나는 것이다. 굴리는 방향이 일정하다면 구가 아닌 원통형으로 부피가 늘어난다. 구 형태의 물체가 굴러감으로써 부피가 커짐을 구현하기 위해서 새로운 방법을 제시할 필요가 있다. 이 시뮬레이션은 메쉬의 실시간 변형을 일으키기 때문에 저사양 모바일 기기의 어플리케이션에 적용하기 어렵다. 유니티의 GPGPU의 병렬 프로세싱은 저사양의 기기에서도 고성능의 애니메이션을 가능하게 한다. 눈은 입자로 이루어져 있다. 습도, 온도 등 기상환경에 따라 성질이 크게 변한다. 환경에 따라 입자의 모양도 변화한다. 눈은 시뮬레이션 하기 까다로운 물질 중 하나다. 기존 눈을 시뮬레이션하는 기술은 MPM(Material Point Method)이 있다. MPM은 물질을 입자 격자 형태로 나누어 표현함으로써 입자의 복잡한 운동과 변형을 정확하게 모델링할 수 있는 기법이다[2]. 본 연구에서는 눈의 완벽한 움직임이 아닌 멀티 플랫폼에서 사용할 수 있는 눈덩이의 커짐을 제시하고자 한다. 지면과 눈덩이가 닿는 순간 지면의 눈이 구형 물체에 붙어 부피가 커짐을 구현하기 위해서 충돌처리를 해야 한다. 지면과의 충돌 지점, 충돌 지점의 텍스쳐 UV 좌표를 활용하여 쉐이더 프로그래밍을 할 수 있다. 쉐이더 프로그래밍은 하드웨어의 가속을 받기 때문에 실시간 구현이 가능하다. 기존 유니티에서 사용하는 충돌처리 기법은 광선을 활용한 레이캐스팅 기법과 OnCollisionEnter() 등 콜라이더의 충돌 이벤트를 처리하는 콜백함수를 활용하는 방법이 있다. 눈덩이는 구체이기 때문에 텍스쳐를 입히는 과정에서 왜곡이 발생한다. 이는 텍스쳐의 좌표와 눈덩이 표면 버텍스 좌표의 관계를 이용하여 보완할 수 있다.

본 연구의 2장에서 기존 방식을 소개하고 본 연구에서 제안할 방법을 설명한다. 3장에서는 제안된 방법과 결과를 설명한다. 4장에서는 실험 및 결과를 확인한다. 5장에서는 향후 전개와 연구 방향에 관해 설명한다.

2. 유니티 충돌처리와 UV 매핑

레이 캐스팅은 광선의 시작 위치에서 발사한 광선에 충돌한 물체의 유무를 확인한다. 이때, 레이어 마스크를 이용하여 충돌을 감지할 물체 또는 충돌을 무시할 물체를 설정할 수 있다. Raycast.Hit의 TextureCoord 속성은 메시 콜라이더에만 있기 때문에 메시 콜라이더를 사용했다. 스플랫 맵은 데이터에 가까운 이미지다. 스플랫 맵은 지형에서 이끼, 젖음 등 특정 부분을 표현할 때 활용되고 있다. 레이 캐스트를 통해 얻은 UV 좌표를 스플랫 맵에 그려 자동차 바퀴 자국을 표현할 때 사용되기도 한다[3]. 본 연구에서는 지면과 충돌한 구체의 충돌지점 표면을 돌출시키기 위하여 사용한다. 매핑은 이차원 텍스쳐를 삼차원 물체에 입히는 과정이다. 매핑의 종류에는 큐브맵을 이용한 매핑, 프로젝션 매핑, 트라이플래너 매핑, UV 매핑이 있다. 큐브맵 매핑은 육면체 박스 물체에서 주로 사용한다. 프로젝션 매핑은 다양한 형태의 물체에 사용한다. 트라이플래너 매핑은 지형 구현에서 주로 사용한다. UV매핑은 가장 기본적인 매핑이라 할 수 있다. UV 매핑은 각 버텍스에 UV 좌표가 할당되는 매핑 방식이다. 본 연구는 UV 매핑을 사용한다 UV 좌표는 0~1 사이의 값으로 구성된 좌표다. 각 좌표에 해당하는 이미지 픽셀이 삼차원 물체를 감싸듯 매핑된다. 유니티에서는 이차원 UV 텍스쳐가 삼차원 물체에 매핑될 때 텍스쳐의 왜곡이 발생한다. 본 연구에서는 눈덩이 표현을 위해 구 물체를 사용했다. 유니티에서 어떠한 설정 없이 UV매핑을 사용했을 시 구 물체에서는 극점 부분에서 지그재그 왜곡이 발생한다. [그림 1]은 사각형 이차원 텍스쳐가 구에 매핑될 때 발생하는 현상을 보여준다.

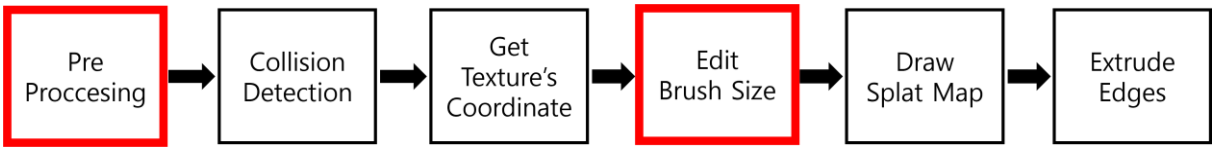
[그림 1] 구 극점에서 보이는 텍스쳐의 왜곡

[Fig. 1] The Distortion Seen at the Poles of Sphere

[그림 1]에서는 텍스쳐의 왜곡을 확실히 나타내기 위하여 격자형태의 텍스쳐를 제작해사용했다. 극점에서 보이는 좌표는 각각 (0,0), (1,0)으로 U축의 시작점과 끝점을 나타낸다. [그림 1]에서 텍스쳐가 지그재그로 매핑됨을 볼 수 있다. 극점에서 발생하는 텍스쳐의 왜곡에 의해 스플랫맵에 정확한 지점을 그리기 어렵다. 좌표가 정확히

지정되기 어렵기 때문이다. 해당 부분의 좌표가 매끄럽게 추출되지 않기 때문이다. 정확한 좌표를 얻기 위하여 이 왜곡을 완화해야 한다.

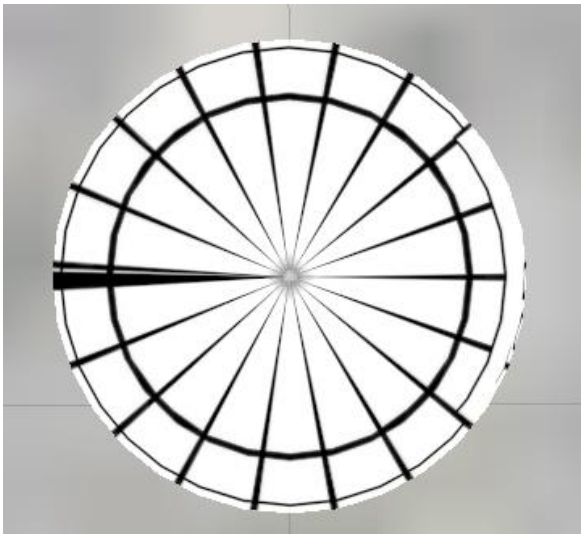
3. 제안된 왜곡 완화법



[그림 2] 시뮬레이션 구현 과정

[Fig. 2] The Process for Implementing Simulation

본 연구의 시뮬레이션은 [그림 2]와 같은 과정으로 구현했다. 기존 스플랫맵의 사용 방식은 4단계로 진행된다. 본 연구에서는 여기에 텍스쳐 전처리, 브러쉬 크기 수정 과정을 추가하여 구현한다. 텍스쳐 전처리 단계에서 구체 매핑시 발생하는 텍스쳐의 왜곡을 완화했다.



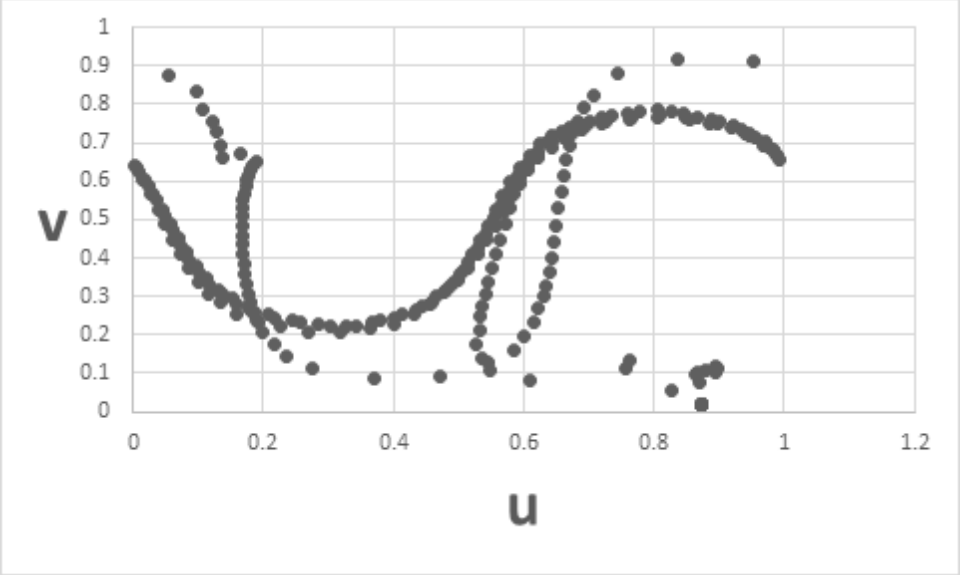
[그림 3] 텍스쳐 전처리 결과

[Fig. 3] The Result of Preprocessing

[그림 3]은 텍스쳐 전처리 과정으로 구의 극점 부근에서 발생하는 왜곡을 완화한 결과다. 충돌지점 좌표 추출에서 발생하는 왜곡을 보완할 수 있다. 이후 충돌을 감지하여 충돌지점 텍스쳐의 좌표를 추출한다. 추출된 좌표의 위치를 고려하여 브러쉬의 크기를 수정한다. 스플랫 맵에 해당 위치를 그린다. 스플랫 맵의 데이터가 있는 부분 엣지를 돌출한다. 레이 캐스팅 충돌 처리를 위하여 레이의 속성을 설정해야 한다. 본 연구에서 구는 회전하며 이동하기 때문에 광선의 시작 위치를 조정해야 한다. C 구의 중심에 해당하는 좌표이다.

$$P_{ray} = (C_x\ ,\ C_y - \alpha \times r,\ C_z\ ) \quad \text{식 (1)}$$

식 (1)은 기울임 없는 평면이라 가정했을 때 광선의 시작 위치를 계산하는 식이다. P_{ray}는 구의 중심에서 y축으로 $\alpha \times r$ 만큼 뺀 위치로 설정할 수 있다. 식 (1)을 통하여 지면과의 충돌을 감지하는 광선이 구를 따라다닐 수 있도록 하였다. 지면이 기울어져 있지 않을 때, 지면의 위치만 지정하면 유니티 내부 속성을 이용하여 직진하는 광선을 만들 수 있다. 기울임 없는 평면 지형에서는 광선의 시작 위치만 조절하여도 직진하는 속성을 사용하여 광선의 위치를 설정할 수 있다.



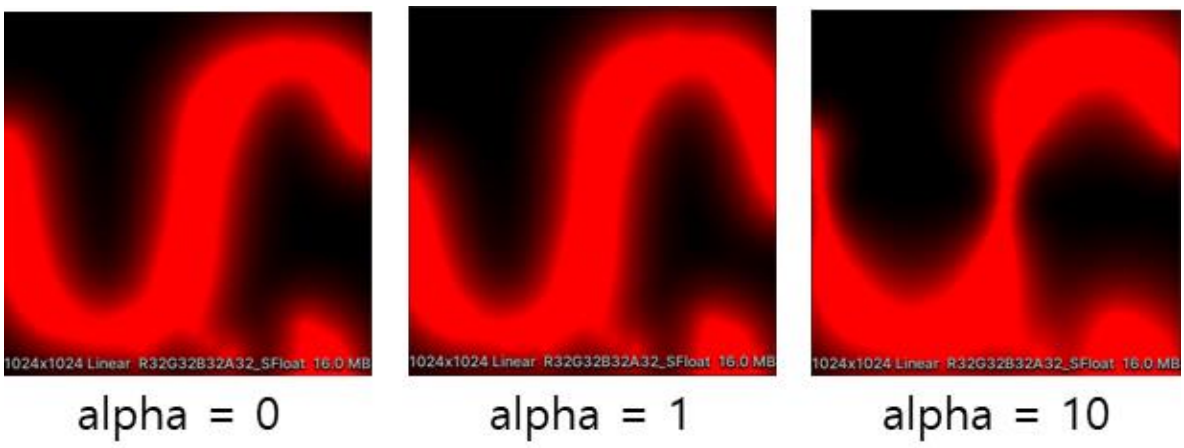
[그림 4] 구의 이동에 의한 UV좌표 이동

[Fig. 4] The UV Coordinate's Move by Sphere Position

[그림 4]는 구가 회전하며 직진할 때 지면과 충돌하는 UV 좌표의 궤도를 시각화한 것이다. 여기서 알 수 있듯이 구가 회전하며 이동할 때 지면과 충돌하는 지점의 UV 좌표는 구의 표면을 따라 이동한다. 구의 표면을 구면 좌표를 이용하여 매개 변수로 표현할 수 있다[4]. 반지름이 r인 구의 극점은 (x, y, z) = (0, r, 0) 혹은 (0,-r, 0)이다. v가 0 또는 1일 때 극점을 지난다. v 좌표가 0 또는 1에 가까울수록 텍스쳐의 수축이 발생한다. 멀어질수록 팽창이 일어난다. 본 연구에서는 구의 직교좌표계를 이용하여 스플랫 맵에 그려지는 브러쉬의 크기를 조정하였다.

$$B = B + \alpha cos(2\,\pi\ \times\ v) \quad \text{식 (2)}$$

식 (2)는 v 좌표에 따라 브러쉬의 크기를 변경하는 식이다. α로 가중치를 준다. 구 표면의 곡률에 맞추어 설정할 수 있다. 브러쉬의 크기가 조정되어 수축한 부분은 더 두껍게, 팽창된 부분은 더 얇게 그려지도록 한다. 곡면에 의한 텍스쳐의 수축과 팽창을 보완한다. 아래 그림은 가중치 α 값에 따른 스플랫 맵에 그려지는 곡선의 차이를 보여준다.



[그림 5] 스플랫 맵의 alpha값에 따른 변화

[Fig. 5] The Comparative Analysis of Splat Maps with Varying Alpha Values

[그림 5]의 가장 왼쪽 그림은 가중치가 0일 때, 중간 그림은 1일 때 그리고 가장 오른쪽 그림은 10일 때를 나타낸다. v 축의 변화에 따라 가운데 지점은 가장 얇게 그려진다. 반면 위아래 지점은 두껍게 그려진다. 스플랫 맵이 두껍게 그려지면서 돌출되는 부위가 넓어진다. 충돌지점이 극점으로 향할수록 텍스쳐가 좁아지는 점을 보완한다. 구의 구면좌표계를 활용하여 극점 부근의 우글거리는 왜곡을 완화할 수 있다. 극점에서 발생하는 왜곡을 완화함으로써 극점을 지날 때 UV 좌표를 더욱 정확하게 얻어올 수 있다.

4. 실험 및 결과

본 실험의 결과를 측정하기 위해 5인으로 구성된 평가단의 MOS 평가를 받았다.

[표 1] 파라미터

[Table 1] Parameters

	왜곡 보완 전	왜곡 보완 후
BrushStrength	0.15	0.15
BrushSize	0.7	0.7
Splat Amount	0.2	0.2
alpha		5

[표 1]은 MOS 테스트에 사용된 시뮬레이션의 파라미터를 나타낸다. 시뮬레이션에 사용된 PC는 인텔 Core i7-12700F CPU를 사용했다. GPU는 Nvidia의 GeForce RTX 3060을 사용했다. RAM은 삼성 DDR4 16GB PC4-25600을 2개 사용했다. 메인보드는 GIGABYTE B550M GAMING D4 모델을 사용했다. VRAM은 12288MB GDDR6이다. 사용한 소프트웨어로 운영체제는 Windows 11 Home, 시뮬레이션 플랫폼은 유니티 2021.3.11.f1을 사용했다. 테스트 평가단에게 왜곡 보완 전, 보완 후 각각의 시뮬레이션을 보여주고 [표

2] 기준에 맞춰 평가를 실시했다.

[표 2] MOS 스코어 기준

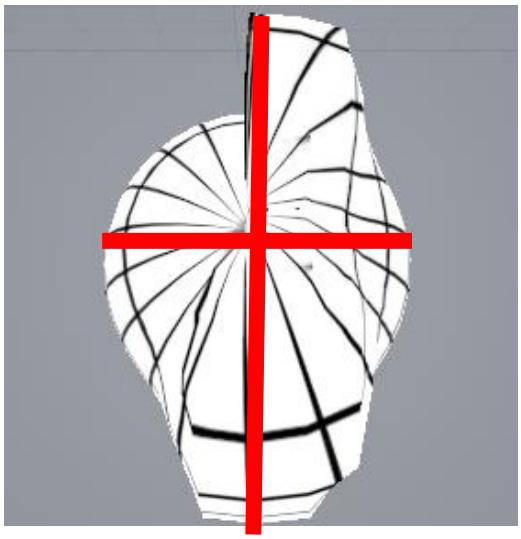
[Table 2] The Standard about MOS Socre

점수	품질	세부 내용
5	매우 좋음	이전보다 확실히 낫다
4	좋음	이전보다 낫다
3	보통	이전과 비슷하다
2	나쁨	이전보다 나쁘다
1	매우 나쁨	이전보다 확실히 나쁘다

테스트 결과 매우 좋음 2표, 좋음 3표로 총 점수 22점을 기록했다. “팽창 시 충돌 좌표가 지정되지 않아 발생했던 뚫림 현상이 완화되었다.”는 코멘트를 받았다.

5. 결론

기존 실시간 애플리케이션에서 구체를 굴려 크기를 키우는 애니메이션은 물리법칙을 무시한다. 기존 방식은 전체 부피가 팽창하는 방식이다. 실제로는 지면과 닿는 면에 지면의 물질이 붙으며 크기가 커지기 때문이다. 이러한 특성은 특히 모바일 게임에서 많이 발생한다. 모바일 환경 특성상 고사양의 그래픽을 재현하기 어렵기 때문이다. 눈에 관련한 애니메이션을 사실감있게 구현하는 것은 매우어렵다. 게임 등 장르 특성상 비현실적인 애니메이션이 더 잘 어울린다는 이유도 있다. 기존 과제를 수행하면 구의 전체 부피가 커지는 방식으로 구현된다. 본 연구의 방식을 사용하면 충돌과 마스킹 기법 등을 활용하여 실제와 유사한 구현을 한다.



[그림 6] 시뮬레이션 결과

[Fig. 6] The Result of Simulation

본 연구에서는 리지드 바디를 사용하여 눈이 회전하며 직진이동 할 수 있도록 했다. [그림 6]는 z축 방향으로 구가 회전하며 직진이동한 시뮬레이션 결과다. 구를 y축에서 바라보았다. x축보다 z축의 길이가 약 1.62배 길게 측정됐다. 즉, 이동 방향을 고려한 눈덩이의 불어남을 구현할 수 있다. 극점에서 특정 좌표가 지정되지 않는 문제가 남아있지만 이후 구와 평면 매핑의 관계 등으로 해결할 수 있을 것이라 예상한다.

본 연구를 바탕으로 다양한 기기에서 기존보다 실감 나는 표현을 구현할 수 있다. 이는 게임 콘텐츠 개발에 유용할 것으로 예상된다. 사용자는 보다 실감 나는 경험을 할 것이다. 다양한 쉐이더 기법과 물리 기반 시뮬레이션 기법을 결합하여 좀 더 자연스럽고 사실적인 표현을 할 수 있다. 최적화 기법을 결합하면 다양한 기기에서도 고성능의 애니메이션을 볼 수 있을 것이라 예상한다.

References

[1] The 2023 Unity Gaming Report, Unity, (2023)
Available from: https://create.unity.com/gaming-report

[2] Alex Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, Andrew Selle, A materail point method for snow simulation, ACM Transactions on Graphics, (2013), Vol.32, No.4, pp pp.1-10.
DOI: https://doi.org/10.1145/2461912.2461948

[3] Rogef Crawfis, Nelson Max, Texture Splats for 3D Scalar and Vector Field Visualization, IEEE Conference on Visualization, IEEE, (1993)
DOI: https://doi.org/10.1109/VISUAL.1993.398877

[4] Joo Woo Seok, Learning 3D Computer Graphics with OpenGL, Hanbit Academy, (2013)