2020-1

[CSED451] Computer Graphics

Term project
Dammit Doll Simulator

Team: 지곡동 피주먹

20192913 컴퓨터공학과 김기홍 (rrgg985)

20202889 창의 IT 융합공학과 이다진 (dj1122)

20202755 컴퓨터공학과 이영규 (lee0gyu)

목차

1 프로젝트 개요

1.1 프로젝트 설명

*소년이 잘못하면 소년원에 가는데, 대학생이 잘못하면 대학원*에 간다는 우스갯소리가 있다. 대학원생들이 불확실한 미래, 금전적 어려움, 교수님 및 연구실 구성원들 간의 사소한 마찰 등으로 인해 정신적 스트레스를 많이 받는다는 뜻이다. 만화영화 <짱구는 못말려>의 등장인물 유리는 억압된 분노를 토끼인형을 때림으로써 풀곤 한다[1]. 2000년 대에 유행했던 플래시 게임 <컴퓨터 부수기>는 시각적으로 컴퓨터가 부숴지는 모습만으로도 마치 실제 컴퓨터를 부수는 듯한 쾌감을 전달할 수 있음을 보여주었다[2]. 이처럼 본 팀은 사용자의 입력에 따라 인형이 물리적으로 망가지는 모습을 시각적으로 구현하여 제공함으로써 대학원생들의 스트레스 해소에 도움을 주고자 한다.

[Fig. 1] (좌) 만화영화 <짱구는 못말려>에서 유리가 토끼 인형을 때리는 모습.

[Fig 2.] (우) 플래시 게임 <컴퓨터 부수기>.
컴퓨터를 클릭할 때마다 주먹이 날아와 컴퓨터를 때리면서 컴퓨터가 조금씩 파손된다.

1.2 프로젝트 주요 기능

본 프로젝트에서는 다음과 같은 기능을 구현하였다.

1) 던지기

사용자는 space bar + 마우스 좌클릭을 통하여 인형을 원하는 방향으로 던질 수 있다. 인형 뿐만 아니라 주위 환경에 있는 물건들도 던질 수 있다.

2) 자르기

사용자는 space bar + 마우스 우클릭을 통하여 인형을 원하는 각도로 자를 수 있다. 인형을 자를 땐 인형 안에 들어가있는 각종 재질들이 튀어나온다. 인형 뿐만 아니라 주위 환경에 있는 물건들도 자를 수 있다.

3) 인형을 구성하는 각종 재질들

본 프로젝트에 사용된 인형은 다양한 재질들로 구성되어 있다. 주된 재질은 점탄성 물질, 이른바 젤리이며 내부는 고체(구슬)와 액체(혈액)가 특정 영역을 이루고 있다.

4) 시점 이동

본 프로젝트에서는 1 인칭 뷰를 구현하였다. 그래서 키보드와 마우스를 통해 실제 FPS 게임과 같은 조작감을 실감할 수 있다.

5) Sound

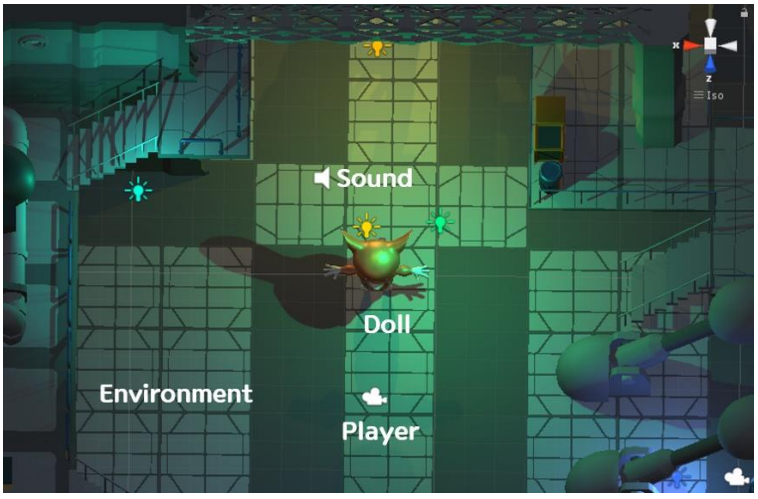
사용자가 인형을 포함하여 object 들과 interaction 을 할 때 각종 효과음이 실행되도록 하였다. 이에 더불어 분위기를 긴장시키는 bgm 도 추가하였다.

1.3 프로젝트 개발 환경 및 실행 방법

본 프로젝트는 Window 10, Unity version 2018.4.23f1 하에서 개발하였다. 하지만 실제 실행되는 것은 별도의 exe 파일이기 때문에 Window 환경에서는 큰 제약 없이 실행시킬 수 있다. 다만 Mac OSX 환경에서는 실행이 제한된다.
키보드 w, s, a, d 키를 이용하여 전, 후, 좌, 우를 이동할 수 있으며 마우스를 이용하여방향을 조정할 수 있다. 물체를 던질 때에는 space bar 를 누른 상태에서 던질 물체를 마우스 좌클릭한 다음 던지고 싶은 방향으로 던지면 된다. 물체를 자를 때에는 space bar 를 누른 상태에서 자를 물체에 마우스 우클릭으로 슬래쉬해주면 된다.

2 프로젝트 설계

아래는 본 프로젝트의 주요 구성 요소들이다.



[fig 3. 프로젝트 주요 구성 요소들]

항목별로 살펴보면 다음과 같다.

2.1 Doll

Doll 은 Player 와 가장 많은 interaction 이 있는 object 다. Doll 내부 물질들을 제외하고는 project 내에서 유일하게 deformable 한 object 다. Doll 자체는 하나의 GameObject 이며 Doll 형태의 Mesh 와 Texture 를 입혔다. Doll 의 mesh 는 mixamo 에서 제공하는 무료 .fbx 파일을 import 하여 사용하였다.

[fig 4. Doll[3]]

Doll 의 동작과 효과를 정의하기 위해 여러 가지 Script 와 속성들, 그리고 하위 object 들이 사용되었다. 주요 항목들을 살펴보면 다음과 같다.

1) Ripple Deformer Script

Doll은 기본적으로 젤리로 구성되어 있다. 젤리 마다 갖고 있는 특성은 다르지만 기본적으로 다른 물체와의 충돌 시에 약간의 찰랑거림과 탄성력이 발생한다. 이를 Doll에서도 표현할 수 있도록 해주는 C# script가 Ripple Deformer Script다. 기본적인 원리는 frequency와 phase를 주어 이에 맞는 파동을 발생시킨다. 이것을 object의 각 축에 반영하여 모양이 변하게끔 보여주는 것이다. 자세한 원리는 4장에 기술하였다.

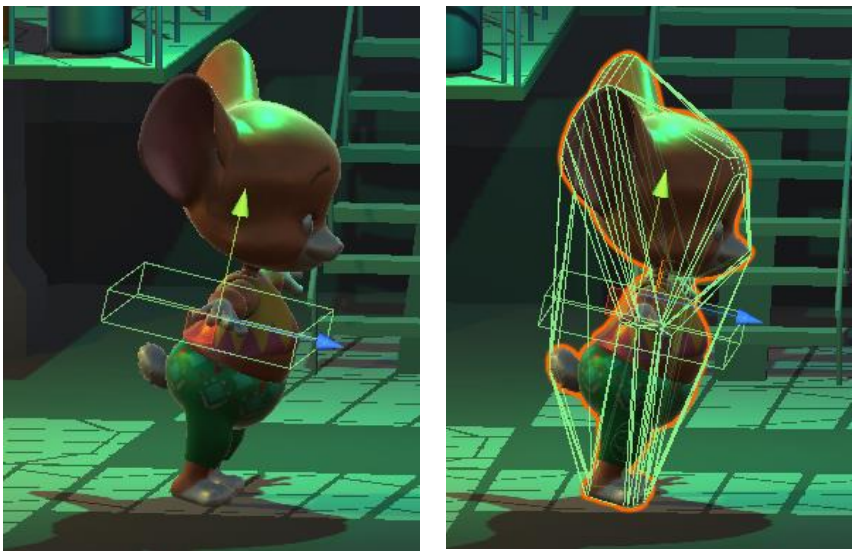
2) Splitable Script

Doll을 비롯하여 Environment에 존재하는 일부 object들은 Player에 의해 잘려나갈 수 있다. 이때 잘려나갈 수 있는 상태를 지정해주는 Script가 필요한데 그 역

할을 하는 것이 Splitable Script다. 기본적인 원리는 space bar + 마우스 우클릭을 한 시점의 좌표와 slash 이후 우클릭을 뗀 지점 그리고 slash하는 동안 raycast를 통해 check한 collider가 어느 object의 것인지 확인 후 잘라낸다. 자세한 원리는 4장에 기술하였다.

3) 신체부위 objects

Doll은 자르는 신체부위에 따라 발생하는 event가 조금씩 차이가 있다. 팔, 다리를 자르면 구슬이 퍼져 나오는 event가, 머리를 자르면 이에 액체가 흘러나오는 event가 발생한다. Doll의 경우 하나의 거대한 collider를 갖기 때문에 각 영역별로 세분화된 collider가 필요했다. 각 신체 부위에 붙은 object들은 이를 위한 것이다. 이로써 신체부위 object와 mouse raycast간 collision으로 인형 내의 material을 발생시킬 수 있다. 동시에 Doll의 collider에서도 hit하게 되므로 Doll 자르기 효과까지 보존할 수 있다. 다만 mesh가 존재하면 원치 않은 visual effect가 생길 수 있기 때문에 mesh없이 collision 영역만 설정하였다.



[fig 4. 신체부위 object 중 right arm의 collider(좌), 신체부위 object 및 Doll의 collider(우)]

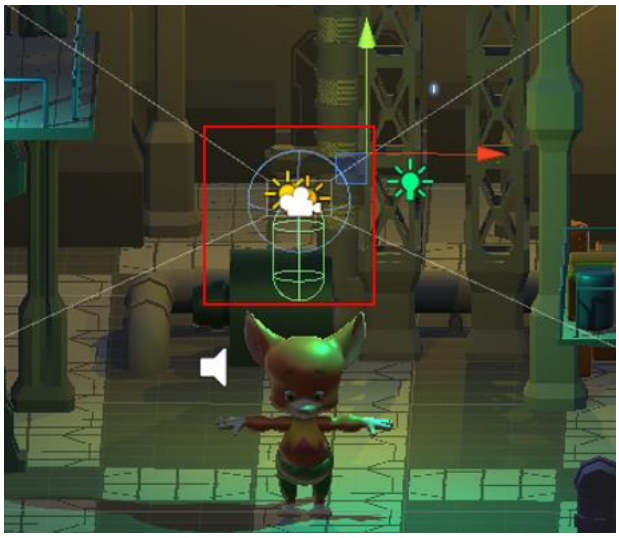
4) 각종 material

Doll은 여러 material로 구성된 복합 object이다. 본체는 점탄성 물질 중 하나인 젤리, 팔다리 등의 각종 관절에는 고체 물질 중 하나인 구슬, 그리고 머리는 액체 중 하나인 혈액으로 이루어져있다. 젤리는 본체 자체가 젤리인 만큼 Doll을 던지거나 다른 object와의 충돌을 발생시켰을 때 약간의 찰랑거림과 탄성력이 발생하는 것으로 구현하였다. Player가 해당 위치를 잘랐을 때 발생한다. material의 생성

방법에 대한 자세한 설명은 4에 기술하였다.

2.2 Player

Player는 1인칭 FPS게임처럼 사용자의 시점을 대변하는 역할을 한다. 더불어 Doll 및 기타 Environment의 object와 interaction하는 주체이다. 그래서 카메라와 interaction 관련 Script가 Player의 주요 구성 항목들이다.



[fig 5. Player(in red box)]

Player 의 동작과 효과를 정의하기 위해 여러 가지 Script 와 속성들, 그리고 하위 object 가 사용되었다. 주요 항목들을 살펴보면 다음과 같다.

1) Camera

본 프로젝트는 1인칭인 만큼 Camera를 Player에 장착하고 이를 키보드 및 마우스 컨트롤로 조작할 수 있도록 하였다. 카메라 내부에는 Mouse Motor Script를 통해 mouse 회전에 따라 camera를 회전한다. 그리고 Mouse Cursor Script를 통해 interaction 유형별로 마우스 커서 모양을 달리 하였다. interaction 유형은 일반, 잡기(던지기), 자르기가 있다.

2) Camera Line Splitter Script

PRG나 FPS 게임에서 무기를 들고 휘두르면 그 잔상이 남는 효과를 자주 확인할 수 있다. 본 프로젝트에서도 상기 게임만큼의 효과는 아니지만 slash가 어느 구간

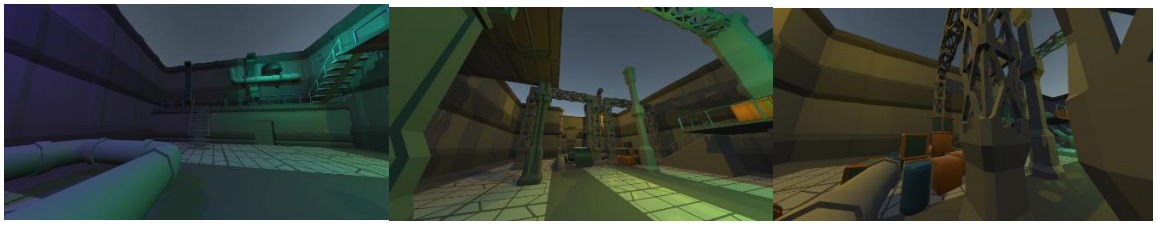
만큼 되는지 사용자가 확인할 필요가 있다 판단하였다. 그래서 slash의 궤적을 남기기로 하였고 그 역할을 하는 것이 Camera Line Splitter다. C# Script이며 마우스 우클릭을 누른 좌표를 시작으로 우클릭을 뗀 좌표까지의 궤적을 사용자가 지정한 색깔로 그려준다.

3) Drag Rigidbody Script

Player는 Doll 및 기타 object를 자르기 뿐만 아니라 잡아서 던질 수도 있다. 잡아서 던지는 효과를 구현한 Script가 Drag Rigidbody다. 마우스 좌클릭을 했을 때 카메라가 보고 있는 시점에서 Raycast에 닿은 rigidbody를 확인한다. 그렇게 Raycast에 닿은 object를 Coroutine을 활성화하여 drag 가능토록 한다. 던지기는 잡기가 이루어진 상태에서 원하는 포지션으로 마우스를 통해 던지는 모션을 취해주면 된다. 그러면 mouse의 방향에 맞게 ray cast가 활성화되고 좌클릭을 뗀 순간의 좌표를 읽어 해당 방향으로 잡고 있던 object를 던진다.

2.3 Environment

게임에서 Player와 기타 캐릭터들만큼 중요한 것이 배경과 그 안에 존재하는 각종 object들이다. 본 프로젝트는 희생자가 될 Doll의 입장에서 다소 음산한 분위기를 연출할 수 있는 Envrionment를 구성하였다. 전체적인 틀은 Asset store에서 import하였지만 배치나 조명, collider와 같은 중요 요소들은 직접 구성하였다.



[fig 6. Environment[4]]

Player 의 동작과 효과를 정의하기 위해 여러 가지 Script 와 속성들, 그리고 하위 object 가 사용되었다. 주요 항목들을 살펴보면 다음과 같다.

1) SceneLights(object)

조명은 Environment의 분위기를 연출하는 가장 중요한 요소이다. SceneLight는 총 7개의 개별 Light로 구성된 Light 집합체이다. 각 Light가 발산하는 빛의 범위, 색 그리고 세기까지 전부 다르다. 그리고 각 Light들을 적절한 위치에 배치하여 각 지점별로 합성되는 빛의 종류를 다르게 하였다. 이를 통해 Environment내의 각

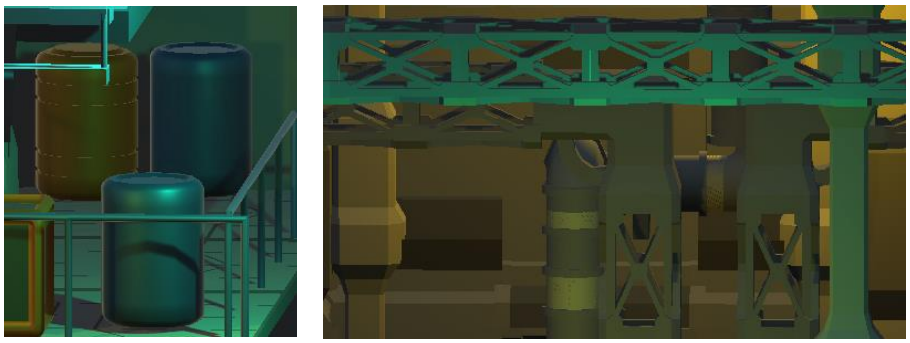
지점 별로 다른 느낌의 조명을 구현하였다.

2) Floors, Walls(object)

Doll 이외에도 Environment의 일부 object들은 던지기 및 자르기가 가능하다. 어느 경우에서든 충돌 효과로 인해 지정된 Environment를 벗어날 수 있다. 던지기는 목표 지점을 Environment 영역 밖으로 설정하면 되고, 자르기의 경우 자를 때의 force를 크게 주면 된다. 하지만 object가 지정 영역을 벗어나면 Envirionment를 구성한 의미가 없어지고, 시각적으로도 좋지 않은 결과를 불러온다. 그래서 Environment를 감싸는 Floors와 Walls를 만들고 각각에 collider를 부여하여 object들이 범위를 벗어나지 않게끔 하였다.

3) 각종 object들

Floors와 Walls를 제외하고도 Environment에는 다양한 object가 존재한다. 계단과 기둥과 같이 시설물로써 쓰인 object가 있는가 하면 박스나 드럼통같이 던지고 자르기가 가능한 object가 있다. 이와 같은 object들은 Doll에게만 interaction을 주려고 했던 기존 계획과는 달리 고안된 object들이다. 제작 진행 중 주제가 스트레스 해소이니 만큼 주변 object들과도 interaction을 할 수 있으면 좋겠다는 의견이 나왔고 이를 반영하였다. 다만 모든 object와 interaction을 하면 다소 현실성이 떨어지는 만큼 기둥, 계단과 같은 시설물 관련 object와는 interaction을 하지 않는다.

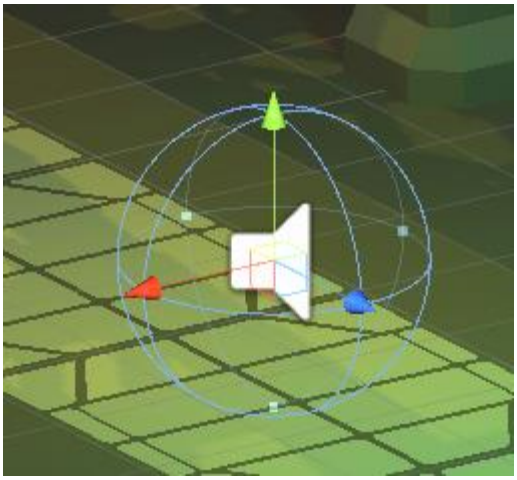


[fig 7. interaction 가능 object인 드럼통과(좌) 불가 object인 철근(우)]

2.4 Sound

그래픽스에서는 시각적 효과를 잘 살려내는 것이 중요하다. 이에 반해 청각적 요소는 중요도면에서는 뒤쳐지지만 시각적 효과를 극대화하는 중요한 역할을 한다. 음산한 bgm을 통해 배경을 더 음산하게 보이는 효과를 준다. 또한 Doll이나 내부 material들

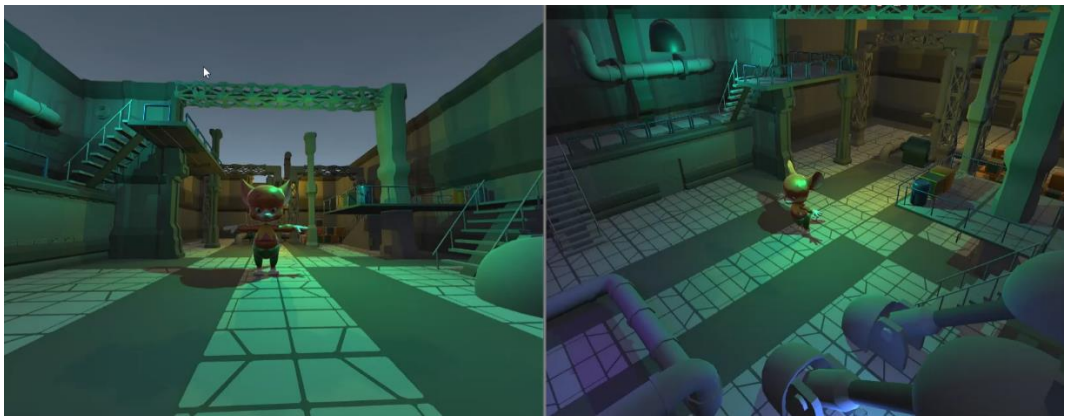
에 대해 collision 및 emission이 발생했을 때 보이지 않던 material들의 존재를 암시하거나 material들에게 더욱 실감나는 효과를 주는 역할을 한다. Sound는 bgm을 포함하여, 구슬, 젤리, 혈액 emission effect sound, 그리고 던지기와 자르기까지 총 6개가 있다. Sound object는 Environment내에 있지만 실행 화면 상에서는 드러나지 않는다.



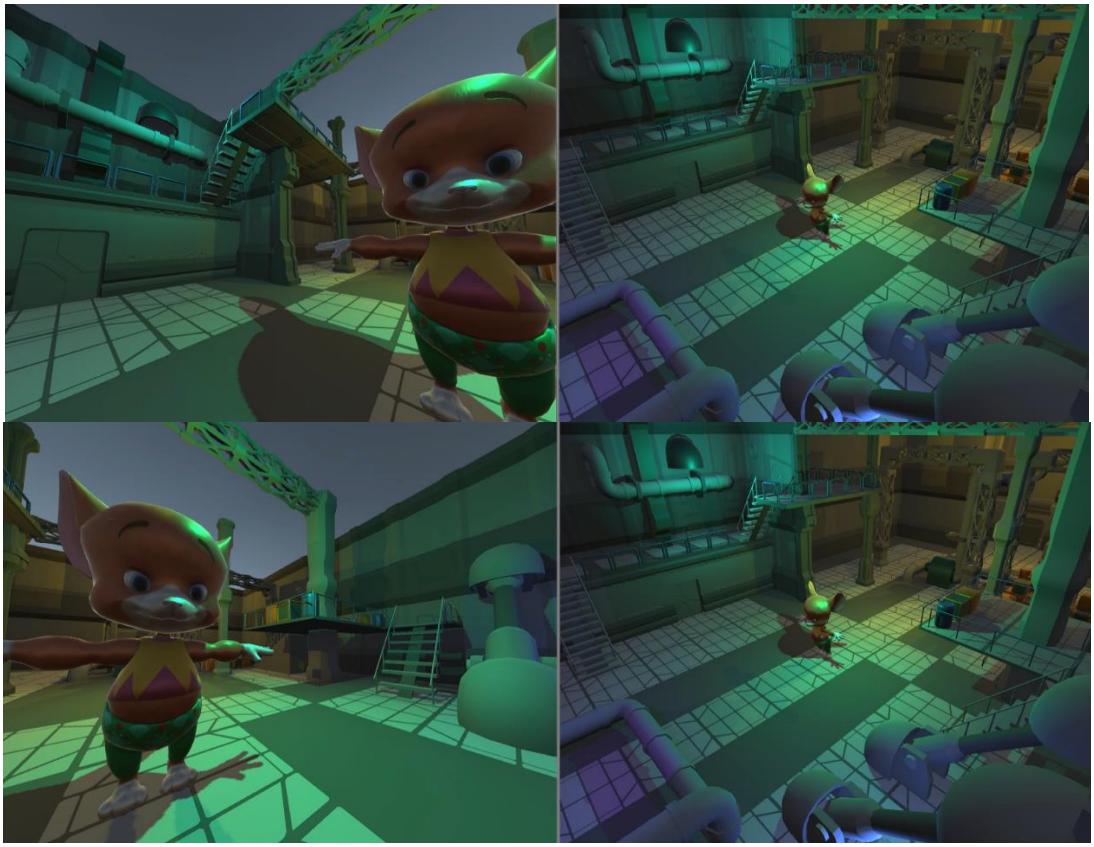
[fig 8. Sound Object]

3 실행화면

모든 실행화면에서 왼쪽은 1 인칭 시점, 오른쪽은 3 인칭 시점이다.



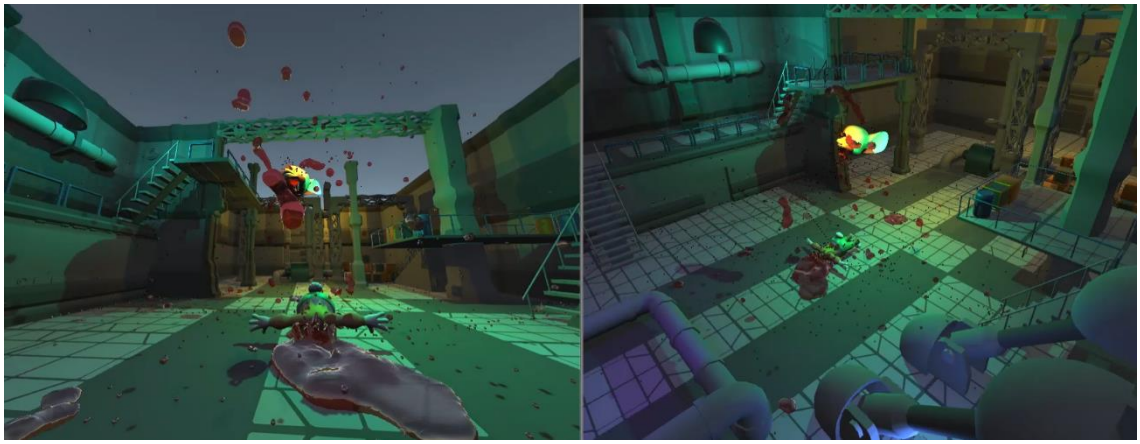
[fig 9. 초기 화면]

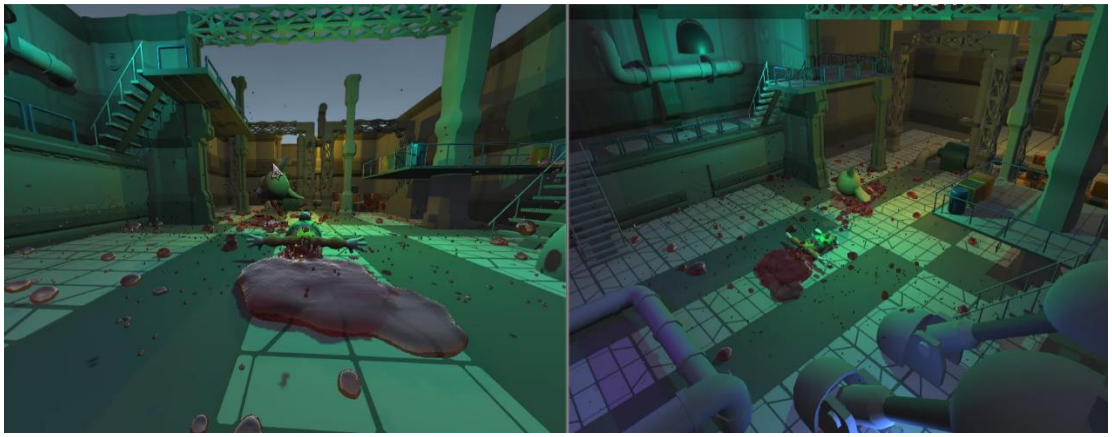


[fig 10. Player의 이동 및 시점 변환]

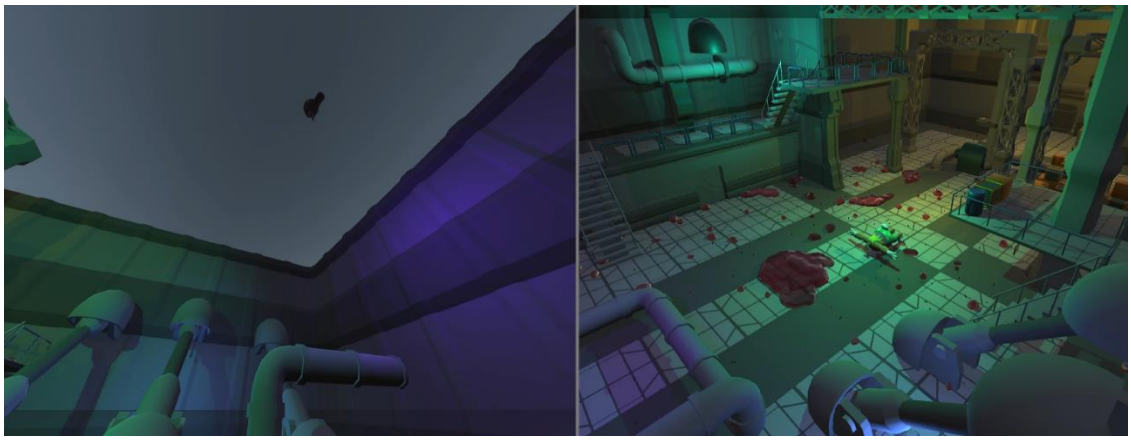
[fig 11. 허공에 대고 자르기]

[fig 12. Doll의 머리 자르기]

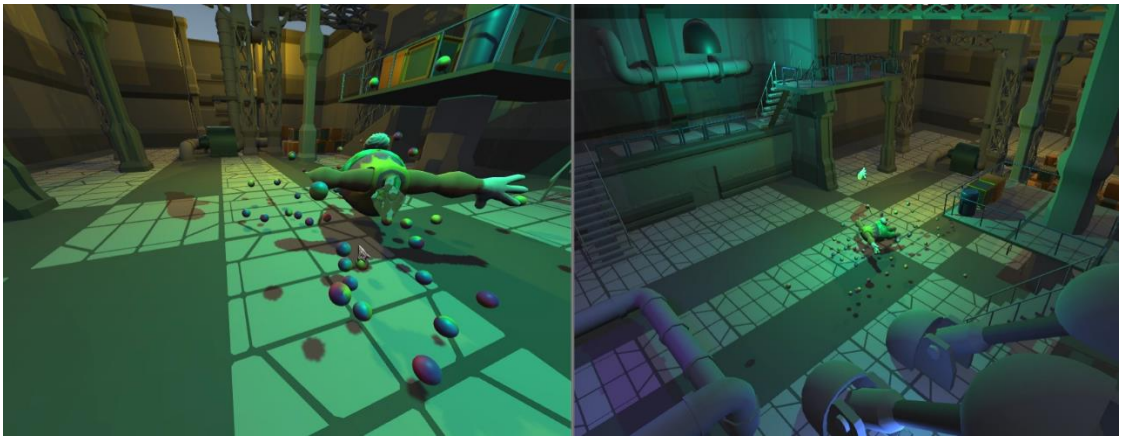




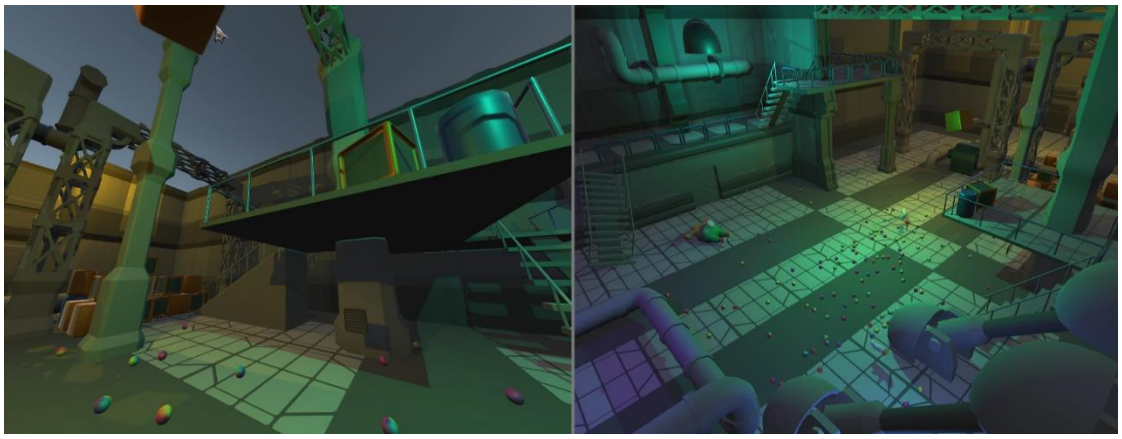
[fig 13. Doll의 머리를 자른 후의 모습들]



[fig 14. 잘린 머리 던지기]



[fig 15. Doll의 팔 자르기 및 효과]



[fig 16. 기타 Object 던지기]

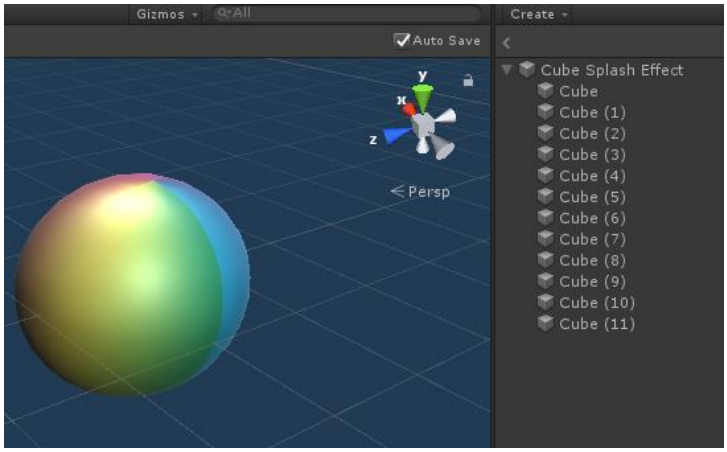
4 토론

4.1 Material

본 프로젝트에서 사용된 material은 고체(구슬), 점탄성 물질(젤리), 액체(혈액) 총 3가지다. 시나리오에 맞게 구현함과 동시에 각자의 특성을 최대한 고려하고자 하였다. 그 결과 각 material별로 구현 방법이 조금씩 달라졌다.

1) 구슬

고체 매질로 선택한 구슬은 특정 event가 발생하면 mesh가 생성되는 방식을 이용하여 구현하였다. 여기서 mesh는 무작위로 생성되는 것이 아니라 미리 지정한 물리적 특성을 갖는 prefab을 통해 생성된다.



[fig 17. 구슬의 prefab]

구슬은 Player가 Doll의 특정 부위를 잘랐을 때 해당 부위에서 발생한다. Unity의 함수 중 Instantiate를 사용하여 준비된 prefab이 튀어나오는 것 같은 효과를 만들어냈다. 다만 단순히 Instantiate만 해서는 안 된다. Instantiate 하기 전에 두 가지 문제를 우선 해결해야 한다.

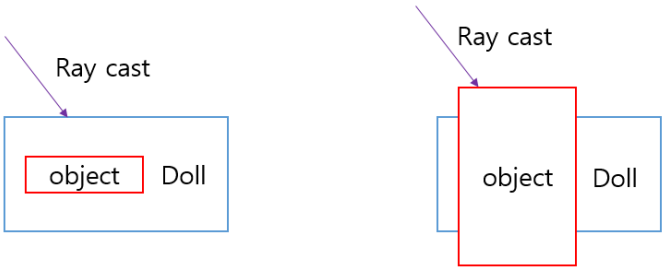
첫째는 방향의 문제다. Instantiate는 지정한 object가 생성될 기준과 방향을 지정해주어야 한다. 그렇지 않으면 object가 생성되지 않는다. 본 프로젝트에서 Doll은 언제 어느 위치에 있으며 어떤 방향을 보게 될지 모른다. 하지만 구슬이 발생되는 Doll의 신체 object들은 Doll의 하위 object들이기 때문에 Doll의 transform이 반영된다. 즉 Doll이 Translate(3.0, 0.0, 0.0)하면 object들도 그만큼 Translate한다. 같은 방식으로 Rotate(90.0, 0.0, 0.0)하면 object들도 그만큼 Rotate한다. 이 점을 이용하여 신체 object들의 초기 orientation을 잘 설정하고, Instantiate의 기준과 방향을 각각 object의 position과 rotation으로 설정하였다. 그러면 Doll이 어느 position과 rotation을 갖든 간에 그 결과가 각 object들에게 반영이 되고 이는 Instantiate에도 반영이 된다. 그 결과 원하는 방향으로 구슬 effect를 emission할 수 있었다.

```
Instantiate(splashEffect, transform.position, transform.rotation);
```

[fig 18. 구슬의 Instantiate]

둘째는 발생 조건의 문제다. 구슬은 자르기 이벤트가 신체 object에게 영향을 주어야 발생한다. 다시 말하면, slash의 raycast가 object의 collider를 hit해야 한다. 이 뿐만 아니라, slash의 동작을 따라야 한다. 즉, space bar를 누른 상태에서 마우스 우클릭을 하고 slash를 한 다음 마우스 우클릭을 뗀 시점에서 발생해야 한다. 어느 한 조건이라도 만족하지 않으면 구슬 effect가 발생해서는 안 된다. 이

를 해결하기 위해 2.1에서 기술한 바와 같이 신체 object의 collider를 넓혔다. Doll에 가려지는 영역을 갖고 있다면 slash를 아무리 한다 한들 raycast가 collider를 hit할 수가 없다. 그래서 collider를 넓혀서 hit할 수 있도록 한 것이다. 물론 object의 collider가 넓어지면 그만큼 Doll의 collider를 가리게 되지만 Doll의 collider영역이 본래 넓게 설정되어 있기 때문에 영향은 미미한 편이다.



[fig 19. collider 영역에 따른 Ray cast hit의 차이]

collider 문제가 해결된 이후엔 2개의 flag를 이용하였다. 우클릭을 누른 상태에서 object를 지나치면 첫 번째 flag를 true로 변경한다. 이후 우클릭을 떼면 두 번째 flag를 true로 변경한다. 이렇게 함으로써 한 번만 emission이 되게끔 하였다.

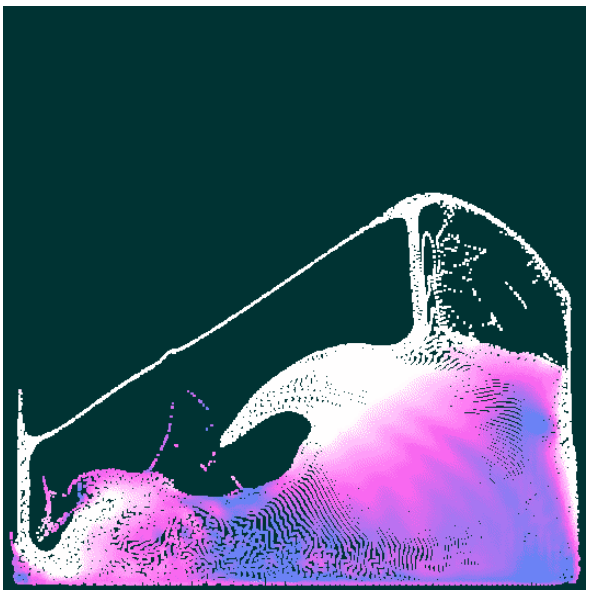
```
RaycastHit hit;
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
if (Physics.Raycast(ray, out hit, 1000f))
{
    if (Input.GetMouseButton(1))
    {
        if (hit.collider.tag == "bead")
        {
            slashed = true;
        }
    }

    if (Input.GetMouseButtonUp(1) && slashed && !emitted)
    {
        SoundManager.instance.PlaySE("beadSound");
        Instantiate(splashEffect, transform.position, transform.rotation);
        emitted = true;
    }
}
```

[fig 20. 구슬 effect의 과정]

2) 젤리

점탄성 물질로 선택한 젤리는 초기 계획과 최종 구현 방향이 많이 달라졌다. 초기에는 Material Point Method(이하 MPM)를 이용하여 구현하려 하였다. MPM은 고체, 액체, 기체 등 다양한 연속체의 움직임을 시뮬레이션하는 데 사용되는 수치적 기법이다. 한 연속체는 material point라고 하는 많은 수의 라그랑지안 요소들로 기술되는데, 이 material point들은 변형 gradient트 항을 계산하는 데에 사용되는 back ground mesh/grid로 둘러싸여 있다. Background mesh가 있음에도 불구하고

high deformation tangling이나 advection errors와 같은 mesh기반 단점들이 나타나지 않는다는 장점이 존재하여 채택하였다.

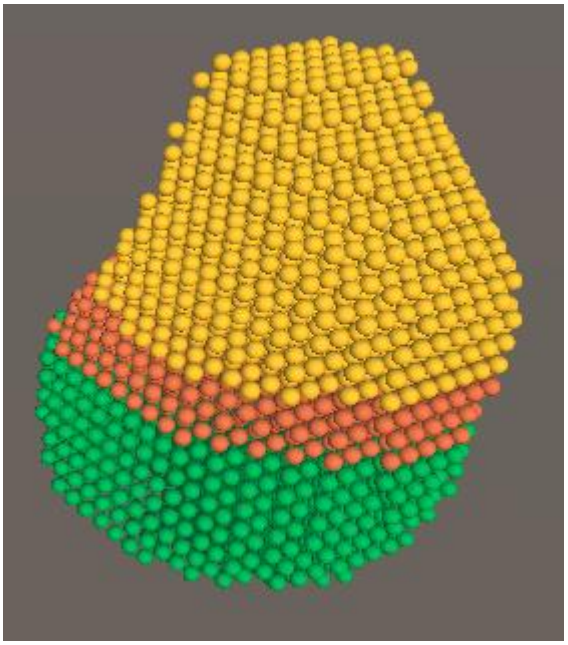


[fig n. MPM의 구현 예시[5]]

결과적으로 MPM을 채택하지 않은 이유는 두 가지가 있었다. 우선 MPM이 2D기반이다. 하지만 프로젝트는 3D 환경에서 구현되었기 때문에 2D 기반인 MPM을 사용하는 데에는 제약이 컸다. 하지만 2D로 정의되어 있던 내용들을 전부 3D화 하는 데에는 성공하였다. 실제로 Doll의 몸통 부분에 맞는 젤리를 만들기도 하였다.

[fig 21. MPM을 이용하여 구현한 Doll의 몸통형 젤리]

두 번째 이유는 계획했던 시각적 효과들을 반영할 수가 없었다. fig21 만 보아도 알 수 있듯이 실제 3D 모델과는 다소 큰 괴리감이 느껴진다. point based 라 하나의 mesh 라기보다는 point 들이 모여있는 것처럼 보였다.



[fig 22. MPM을 이용하여 구현한 Doll의 몸통형 젤리 확대]

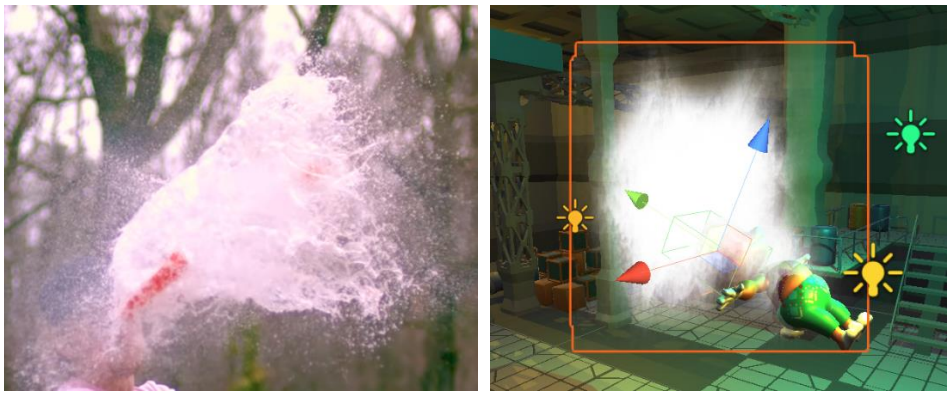
겉모습에서 유발되는 문제뿐만 아니라 interaction 면에서도 문제가 있었다. 프로젝트 특성상 mesh 를 절단해야 하는데 MPM 은 그것이 불가하였다. 더군다나 몸통과 다른 material 로 채울 계획이었던 팔, 다리, 머리는 MPM 을 이용하여 구현할 계획이 아니었다. 위와 같은 이유로 MPM 은 본 프로젝트에 적합하지 않다는 판단 하에 다른 방향으로 구현하기로 하였다. 그것이 Mesh Deformation[6]이다.

Mesh Deformation 의 원리는 비교적 단순하다. 직관적으로 설명하자면 object 에 진동을 일으켜 형태가 변하는 것처럼 보이게 하는 것이다. Inspector 를 통해 원하는 frequency 와 phase, time 등 진동에 필요한 parameter 들을 입력한다. 이때 넣어준 값에 따라 파동이 달라지기 때문에 형태가 얼마나 변하는지도 달라진다. 이렇게 생성된 파동을 각 x, y, z 축에 적절한 연산을 취한다. 그러면 서로 맞물리고 엇갈리면서 object 가 마치 찰랑거리는 효과가 나타나게 된다.

3) 혈액(blood)

액체로 선정한 혈액의 경우 초기와 달라진 점이 2가지가 있다. 첫째는 액체의 종류다. 본래는 일반적인 물을 구현하려 하였다. 그러나 개발 과정 중 분위기 연출을 위해 다른 액체를 이용하자는 의견이 나왔고 그 결과, 음산한 분위기와 잘 어울리는 혈액으로 종류를 변경하였다.

두 번째는 구현 방법이다. 초기에는 Particle System을 이용하려 구현하려 하였고, 어느정도 완성도도 갖추었다. 다만 이 방법이 최종까지 채택되지 않은 이유는 Environment와의 context를 일관하는 데에 다소 이질감이 든다는 점이었다. 초기에는 Doll의 머리가 잘렸을 때 액체의 effect를 어떻게 주어야할지 의문이었다. 그래서 물풍선을 잘랐을 때의 effect를 참고하여 Particle을 구현하였다.



[fig 23. 물풍선을 칼로 잘랐을 때의 효과[7](좌)와 초기 액체 Particle effect(우)]

프로젝트에서 Doll의 머리 자르기는 영상에서처럼 강하게 내려 자르는 행동과는 다소 거리가 멀다. 그래서 이를 반영하여 구현한 Particle System도 실제 시나리오와의 거리감이 다소 있었다. 하지만 별다른 대안이 없었기에 Particle System을 계속 발전시키기로 하였다. 그러던 중 후반부에 액체를 시뮬레이션할 수 있는 다른 방법을 찾아내어 방향을 바꾸기로 하였다. 그것이 NDVIA에서 제공하는 라이브러리인 Flex다.

[fig 24. NVDIA Flex[8]]

NVDIA Flex는 Particle을 기반으로하여 실시간 visual effect를 simulation하는 라이브러리의 일종이다. 특히 deformable object나 액체, particle와 같이 정해진 형태

가 없는 object를 표현하기에 적합한 라이브러리다. 단점이 있다면 NVDIA기반이라 Mac OSX에서는 사용이 불가하다.

결과적으로 액체의 경우 Flex를 이용하여 구현하였다. 다만 Flex가 모든 것을 제공해주는 것은 아니었다. 그렇기에 액체의 흐름, 방향, particle의 사이즈와 액체 자체의 재질 등 물리적 및 겉보기 특성은 직접 구성해주어야 했다. 그렇게 구성하던 중 좀비가 다수 출현하는 게임을 모티브로 하여 혈액이 흘러 나오는 효과를 구현하였다. 시나리오 맥락과 Environment 및 visual effect를 고려하였을 때 물풍선보다 양질의 효과를 보였기에 최종적으로 이를 채택하였다.

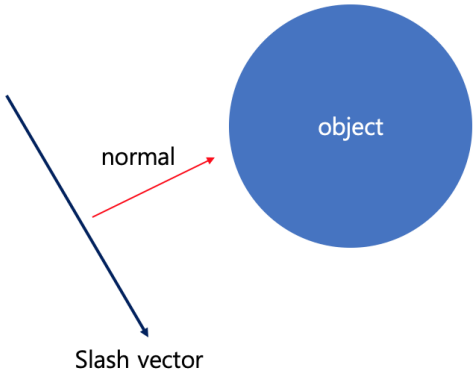
[fig 25. 혈액 분출 효과]

4.2 split

프로젝트 내에서 Player가 타 object들과 할 수 있는 interaction 중 가장 중요한 것이 '자르기'다. 자르기를 통해 Doll 내의 숨겨진 material을 확인할 수 있고 그에 따른 분출 효과도 발생시킬 수 있다.

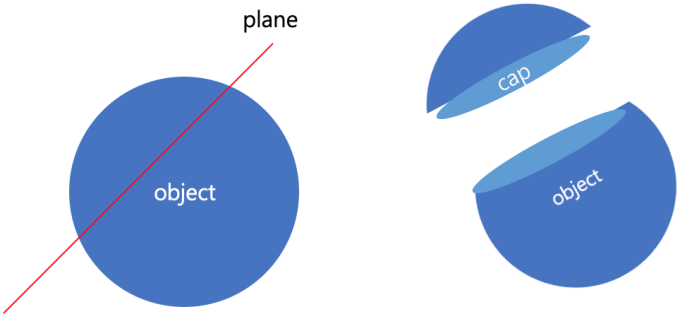
Unity의 Object들은 대부분 mesh로 구성되어 있다. 즉 Object를 자르려면 mesh를 자를 수 있어야 한다. 현실세계에서 물체를 자르듯이 object를 자를 수는 없다. 그래서 새로운 접근법이 필요한데 바로 상위 mesh로부터 하위 mesh를 생성해내는 것이다.

과정은 다음과 같다. 우선 Player가 우클릭으로 그려낸 궤적의 시작 point와 끝 point의 좌표를 구한다. 두 좌표를 양 끝으로하는 벡터를 구하고 이 벡터의 normal에 ray를 적용하여 collision이 발생하는 object 존재 여부를 확인한다. 이때 object는 rigidbody가 적용되어 있어야 한다.



[fig 26. split 대상 object 추적]

다음으로 normal vector와 slash의 normal을 normal vector로 갖는 plane을 생성한다. 이 plane은 object와 normal vector가 만나는 지점에서 생성된다. 그러면 object를 가로지르는 임의의 plane이 생긴다. 이 평면을 기준으로 양옆에 있는 mesh의 vertices를 별도의 변수에 저장한다. 이 vertices들을 이용하여 새로운 mesh들을 생성해낸다. 그리고 이들로 구성된 새로운 object를 만드는 것이 split의 원리다. 여기까지하면 원래 object에서 잘려나간 면이 비어있게 된다. 그래서 새로운 mesh를 만드는 과정에서 plane의 normal과 vertices를 정보를 이용하여 비어있는 부분을 메꾸는 vertices(cap)를 별도로 생성해낸다. 그리고 이것을 새로운 mesh를 이루는 vertices에 넣어준다. 이렇게 해서 잘린 후 새롭게 생긴 object들의 mesh를 만들어낸다.



[fig 27. split]

4.3 Mesh Renderer vs Skinned Mesh Renderer

Unity에서는 Mesh를 렌더링할 때 크게 2가지 방법을 사용한다. 첫째는 Mesh Render-

er 이고 둘째는 Skinned Mesh Renderer다. 전자는 정적인 mesh에 texture를 입힐 때 주로 사용되는 방식이라면 후자는 관절 및 유동성을 가진 mesh를 표현할 때 주로 사용되는 방식이다.

초기에는 젤리로 이루어진 Doll을 표현함과 동시에 ragdoll의 관절까지 나타내기 위해 skinned mesh renderer를 채택하였다. 확실히 Mesh renderer와 비교해서 skinned mesh renderer가 유동성 있는 object와 이에 걸맞는 interaction을 잘 표현하였다. 하지만 split 과정에서 큰 문제가 발생하였다. Skinned mesh renderer는 이미 구현이 완료된 split에 적용할 수가 없었다. Split의 경우 일반 mesh renderer를 기반으로 작성된 것이기에 skinned mesh renderer에서는 작동하지 않았다.

```
MeshFilter[] meshFilters = go.GetComponentsInChildren<MeshFilter>();
for (int j = 0; j < _meshContainerStatic.Length; j++)
{
    Renderer renderer = meshFilters[j].GetComponent<Renderer>();
    if (ForceNoBatching)
    {
        renderer.materials = renderer.materials;
    }
```

[fig 28. mesh renderer를 이용하는 split script]

이미 split은 잘 작동하는 시점에서 skinned mesh renderer를 split하기 위해 기존 split을 수정하기에 부담이 다소 컸다. 그래서 mesh renderer를 쓰는 object라도 유연함을 갖게 하는 방법을 찾기로 했고 그 결과가 deformable object였다.

5 결론

5.1 물질 특성 구현하기

본 프로젝트는 다양한 물질의 특성을 구현하는 것에 전부였다고 해도 과언이 아닐 만큼 큰 비중을 차지하였다. 점탄성과 액체는 난이도가 높을 것으로 예상했으나 실제로는 그 이상이었고, 비교적 쉬울 것으로 예상됐던 고체도 물리적 효과를 고려하는 순간 난이도가 제법 올라갔다. 단순히 각 물질의 느낌만 어느정도 내는 선에 그쳤다면 해법을 찾는 데에 많은 시간을 기울이지는 않았을 것이다. 하지만 본 프로젝트가 Computer Graphics 관련 프로젝트이고 그만큼 visual effect에 많은 공을 들여야 하는 만큼 최대한 realistic하게 구현하는 데에 많은 시간을 투자하였다.

과거의 누군가가 고민했던 내용들이나, 라이브러리들을 참고하면서 구현을 하였는데 팀원 모두가 만족할 수 있는 realistic한 결과물을 내기 위해서는 결국 직접 상당량의 코드를 수정하고 추가하여야 했다. 특히 물리적 특성이 준비된 환경과 시나리오 상에서 잘 작동하게끔 하는 것이 큰 난제였다. 그래도 납득할만한 visual effect를 갖춘 것에 만족하는 바이다.

5.2 Unity

본 프로젝트는 Unity를 기반으로 진행되었다. 대중적인 게임 엔진이면서 다양한 Computer Graphics 효과를 낼 수 있으며, 여러모로 쓰임새가 좋기에 채택하였다. 확실히 Scene을 구성하거나, 충돌 효과 texture등 OpenGL에서는 구현하는 데에 시간이 다소 걸리거나, 구현하더라도 속도나 성능 면에서 만족스럽지 않은 부분들이 Unity 상에서는 말끔히 해결되었다. Mesh 하나를 만들더라도 vertex를 직접 구성하는 노력을 덜어도 되는 면이 큰 장점이었다.

다만 익숙하지 않은 언어인 C# 사용한다는 것과 Unity 자체가 익숙하지 않다는 것이 단점이었다. 그래서 초반에는 Unity 사용법 숙지하는 데에도 많은 시간이 소요되었다. 또한 버전 갱신이 자주 되는 편이라 버전 체크에도 신경을 기울여야 한다. 무엇보다도 문제가 생겼을 때 검색을 하는데, Android나 iOS 그리고 node.js와 달리 원하는 답변을 찾기가 어려웠다.

상기 언급한 내용들이 본 프로젝트에서 Unity를 사용함으로써 진행에 방해가 되거나 문제가 생긴 부분들이었다. 하지만, 결과적으로 새로운 IDE를 사용하고 익혔다는 점은 이후 큰 이점으로 다가올 것이라 생각한다.

6 개선 방향

6.1 collider 겹침 문제

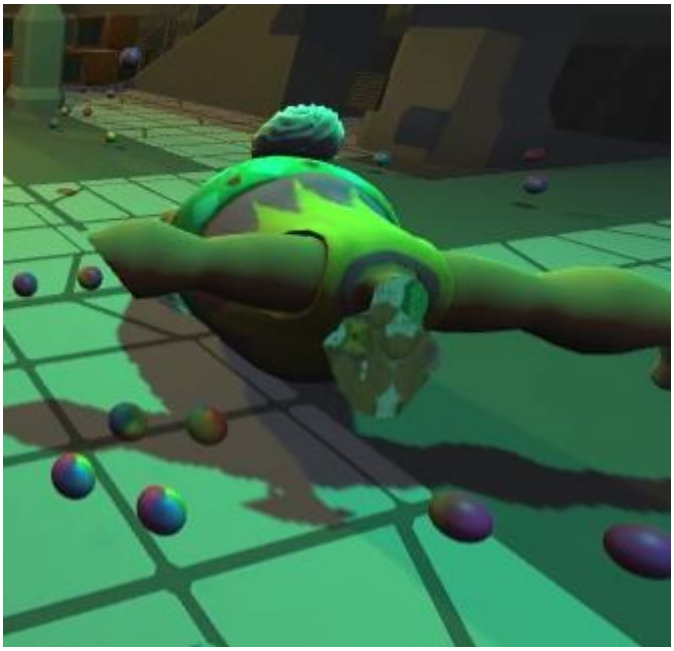
앞서 언급했듯, Doll 자체의 collider와 신체부위 object의 collider가 겹치는 문제가 있었다. 정확히는 object의 collider가 Doll에게 가려지는 문제였다. 본 프로젝트에서는 신체부위의 collider를 조금 확장함으로써 해당 문제를 해결하였다. 하지만 이보다 collider영역을 작게 잡더라도 event 발생은 확실히 일어나는 방법이 있을 것이라 생각한다. 본 프로젝트를 진행하는 동안에 발견하지는 못했지만 차후 이와 비슷한 개발을 할 때에는 찾아서 그 해결방법을 적용할 수 있었으면 한다.

6.2 절단면

현재 사용하고 있는 split의 한 가지 문제점은 바로 절단면이다. object를 절단했을 때 절만면의 texture가 원래 object에 입혔던 texture가 연장되는 문제가 발생한다. 이를 해결하고자 cap에 해당하는 부분의 mesh를 따로 계산하여 이 부분만의 texture를 적용하려 하였다. 이 경우 2가지 문제가 발생하였다.

첫째는 cap만의 mesh를 따로 구하고 그 부분에 대한 새로운 object도 생성할 수 있었다. 다만 cap만 생성되고 그 외의 mesh는 생성되지 않았다. Mesh filter가 하나의 큰 mesh 덩어리만 인식할 수 있는 구조가 원인인 듯하였다.

둘째는 어떻게든 cap과 이외의 부분을 분리했다 하더라도 cap에 원하는 texture를 적용할 수가 없었다. 정확히는 지정 texture를 적용할 수는 있는데 기존 object에 적용한 texture와 겹쳤다. 현재 split 구조 상으로는 두 문제를 동시에 해결할 수는 없었다. 차후 이와 비슷한 프로젝트를 하게 된다면 이 부분 역시 짚고 넘어갈 필요가 있어 보인다.



[fig 29. mesh renderer를 이용하는 split script]

6.3 마우스 커서

대부분의 FPS게임은 사용자가 조작하는 캐릭터의 전면부, 특히 팔 부분과 무기는 어느정도 보인다. 이는 넓은 맵을 사용하는 게임의 및 시점의 특성을 고려한 UX다.

[fig 30. 온라인 FPS 게임 오버워치]

다시 말해서 게임 내의 마우스가 캐릭터 자체가 되는 것이다. 즉, 마우스 컨트롤을 통해 잡고 던지는 액션이 캐릭터를 통해 이루어져서 2D에 비해 실감나는 interaction이 가능하다.

본 프로젝트에서 마우스는 2D 이미지를 사용하였다. 기본 상태, 잡기 그리고 자르기까지 3가지 상황에 대해 커서의 모양이 달라지다. 실제 FPS 게임에서 캐릭터의 행동에 따라 달라지는 느낌을 구현하고자 한 것이다. 하지만 2D인만큼 FPS게임처럼 캐릭터가 직접 잡고 던지거나 자르는 느낌은 다소 부족하다. Player와 다른 Object간 interaction이 중요한 프로젝트인만큼 이 부분은 차후 개선이 필요한 부분으로 생각한다.

[fig 31. 본 프로젝트의 마우스 커서, 왼쪽부터 기본, 잡기, 자르기]

7 팀원 역할 및 기여도

	김기홍(33.3%)	이다진(33.3%)	이영규(33.3%)
역할	-관련 자료 조사 -인형 내용물 구현 -인형 3D 모델 구현 -인형 텍스쳐 구현	-관련 자료 조사 -인형 내용물 구현 -인형 조작 공간 및 인터페이스 구현 -던지기 효과 구현	-관련 자료 조사 -인형 내용물 구현 -자르기 효과 구현

8 참고 문헌

[1] 투니버스 짱구는 못말려 7기

[2] http://fg.gameangel.com/10100000000001447

[3] https://www.mixamo.com/#/?page=1&query=mouse

[4] https://assetstore.unity.com/packages/3d/environments/sci-fi/snaps-prototype-sci-fi-industrial-136759

[5] https://nialltl.neocities.org/articles/mpm_guide.html

[6] https://assetstore.unity.com/packages/tools/modeling/mesh-and-object-deformers-for-unity-3d-81427

[7] https://www.youtube.com/watch?v=OIroNhbQ0XI&t=174s

[8] https://developer.nvidia.com/flex

기타 reference

mesh split : https://github.com/DanniSchou/MeshSplitting

drag : https://pastebin.com/j3DWqe3R

sound : https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116#content

https://assetstore.unity.com/packages/audio/sound-fx/foley/water-splash-pack-14039

mouse cursor : https://assetstore.unity.com/packages/2d/textures-materials/basic-rpg-cursors-139404