마지막달

조영환○, 김태현, 양영현
한국공학대학교 게임공학과
jkyu6554@tukorea.ac.kr,{dece1225,1223sam}@naver.com

요 약

밀물과 썰물에 따른 환경의 변화를 주제로 하는 서바이벌 샌드박스 게임 플레이를 제작하였다. 밀물과 썰물 사이클을 이용해 제한 시간 내 다양한 활동을 할 수 있도록 기획하였고, 흐름에 따라 환경의 난이도가 증가하도록 설계하였다. 이를 통해 자원 파밍과 기지 건설 과정을 즐길 수 있도록 개발하였고, 벡터 그래픽을 활용해 사실주의 그래픽과 차별화된 아트 스타일을 구현하였다.

차별화 된 벡터 아트 스타일을 최대한으로 끌어내며 서바이벌 게임의 최적화 기법을 연구한다.

1. 서 론

1.1 제작 목적

지구를 향해 달이 다가오는 상황을 주제로 삼은 과학 유튜브 동영상에서 영감을 얻어 시간의 흐름에 따라 증가하는 난이도와 밀물 썰물로 나뉘는 환경과 공간의 변화를 게임으로 보여주고자 한다.

1.2 게임 기획 의도 및 주요 특성

벡터 그래픽을 이용한 아트스타일을 구현하고, 이를 통해 Pixel 아트, Voxel 아트와 유사하게 사실주의 그래픽과 차별화되는 아트스타일을 게임의 세일즈 포인트로서 사용하고자 한다.

제한 시간동안 밀물과 썰물을 통해 자원을 파밍 하거나 제작을 할 수 있고 밀물,썰물 사이클을 제작하여 파밍시간, 제작시간을 구분하여 게임의 진행을 나타내려고 한다.

1.3 플랫폼

PC 게임

1.4 장르

서바이벌 샌드박스 게임

1.5 시점

1인칭

2. 개발 내용

2.1 핵심 아트스타일

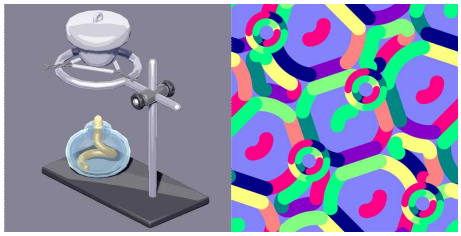
게임의 핵심 아트스타일은 Kurzgesagt 채널의 벡터 그래픽 애니메이션 스타일이다.

이 아트스타일은 벡터 그래픽의 끝이 둥근 Stroke 도형을 위주로 형태와 명암을 그리며, 도형의 외곽에 Stroke로 표현하는 Stroke 하이라이트가 특징이다.

[그림 1]

이를 달성하기 위해 오브젝트를 모델링 할 때, Stoke 도형과 유사한 3차원 Spline 도형

을 위주로 모델링 해 벡터 아트의 실루엣을 구현하였고, 노말 맵과 같은 텍스처 맵을 제작할 때 벡터 그래픽 프로그램을 이용하였다.



[그림 2], [그림 3]

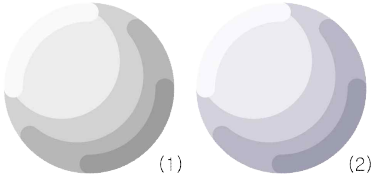
2.2.1 Cel Shading

광원 방향과 노말 벡터 값의 Dot Product 연산값에 Half Lambert 연산을 수행해 명암 값을 구한 뒤, Cel 개수의 곱을 정수화 한 뒤 Cel 개수로 다시 나눠 명암 색상 수를 Cel 개수 만큼 제한하는 Cel 쉐이딩을 제작하였다.

[그림 4]

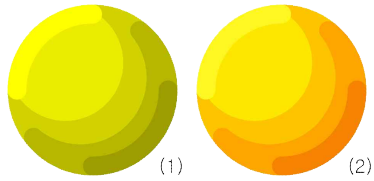
2.2.2 Grayscale 명암 처리의 문제점

광원 연산으로 구해진 명암의 Grayscale 밝기 값을 기본 색의 RGB 값에 곱하는 방식의 쉐이딩을 사용할 시 채도가 감소하는 문제가 발생할 수 있다.(그림 5.1) 따라서 백색을 채색할 때 낮은 명도에서 청색 값을 증가시키는 것과 같이 밝기 감소에 따라 높은 채도의 색으로 채색할 필요가 있다.(그림 5.2)



[그림 5]

특히 노란색은 다른 색상에 비해 명도 감소로 인한 채도 감소가 두드러지고, 낮은 채도에서 녹색에 가깝게 인식될 수 있다. (그림 6.1) 따라서 황색 계열, 특히 황금 같은 색상을 의도하고자 한다면 낮은 명도에서 적색값을 늘려 주황색에 가깝게 변하는 특수한 채색이 필요하다. (그림 6.1)



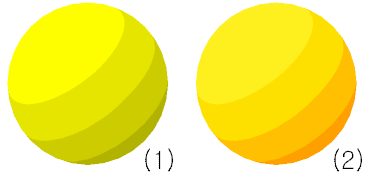
[그림 6]

2.2.3 Look-Up 텍스처 기반 쉐이딩

따라서 미리 명도에 따라 미리 지정한 색상 스트립으로 이루어진 Look-Up 텍스처를 제작하였다.(그림 7) Look-Up 텍스처 범위는 명암 값의 두배 길이로 제작해 그림자, Ambient Occlusion으로 인한 명도 감소 효과, Rim Lighting, 반사 하이라이트로 인한 명도 증가 효과에 따른 색상을 지정하였다.

[그림 7]

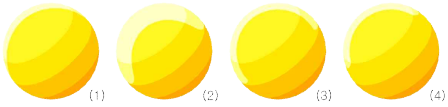
반복 유형을 Clamp로 설정한 후 명암 값을 UV 채널의 G값으로 Look-Up 텍스처를 불러오는 것으로 사전 지정 색상으로 채색되는 쉐이더를 제작하였다.(그림 8.1) [1] 기능을 구현하기 위해 walll4542 블로그를 참조하였다.



[그림 8]

2.3.1 Rim Lighting과 Stroke 하이라이트

노말 벡터 값과 시야 방향 값의 Dot Product 연산의 반전 값과 명암 값의 곱을 정수화 하여 Rim 라이팅 효과를 제작하였다.(그림 9.1) 이후 Dot Product 연산 값에 제곱 연산을 한 뒤 반전하면 벡터 아트의 Stroke 하이라이트와 유사한 형태가 된다는 것을 발견(그림 9.2), Dot Product 연산을 정수화 후 곱해 Rim 라이팅을 외곽으로 제한해 Stoke 하이라이트 효과를 완성하였다.(그림 9.3) 이후 명암 효과의 강도를 조절해 Stroke 하이라이트의 길이를 조절 가능하게 쉐이더를 제작하였다.(그림 9.4)



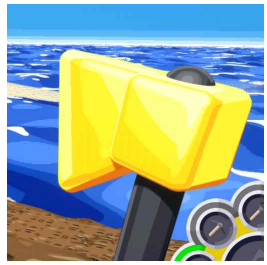
[그림 9]

2.3.2 반사 하이라이트와 인게임 적용

광원 방향과 노말 벡터를 유니티 반사 노드 연산을 한 후 시야 방향 값과 Dot Product 연산 값으로 하이라이트 반사 효과를 제작하였다.

[그림 10]

Look-Up 텍스처 참조, Rim 라이팅, 하이라이트 반사 효과가 적용된 쉐이더를 제작, Look-Up Material 쉐이더로 이름 붙이고 단색이고, 디테일 요소가 불필요한 오브젝트에 적용하였다.



[그림 11]

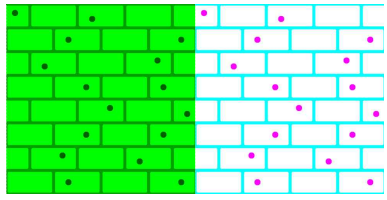
2.4.1 Look-Up Texture Mask

Look-Up Material 쉐이더는 한 종류의 색상만 적용 가능하기 때문에 다양한 종류의 색이 필요한 오브젝트에는 전용 쉐이더를 제작해 제작할 필요가 발생하였다.

이를 해결하기 위해 색상 마스크 맵을 제작해 UV의 R값으로 적용해 다색의 Look-Up 텍스쳐 쉐이더 구현을 시도하였다.(그림 12)

하지만 단색의 색상 마스크 맵은 안티 에일리어싱 과정에서 양 극의 색상이 만나는 지점에서 중간 값의 색상이 들어가는 문제가 발생하였다.

이를 해결하기 위해 RGBA 채널에 각각 색상 마스크를 적용해 기본 색상과 RGBA 마스크로 최대 5종류의 색상을 표현 가능한 다색 Look-Up 쉐이더를 구현하였다.(그림 13)

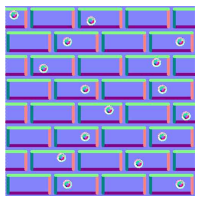


[그림 12], [그림 13]

2.4.2 Normal Map, HOST Map

마스크 적용을 이용한 색상 수에는 한계가 있기 때문에 텍스처의 디테일은 노말맵과 Ambient Occlusion 맵을 이용해 적용하도록 결정하였다. 노말맵 제작 과정에서도 중간값 문제가 발생해 RGB 채널에는 0, 0.5, 1 세

가지 값만 가지도록 제작하고, 알파 채널 값에 노말맵의 강도 값를 지정하였다.(그림 14)

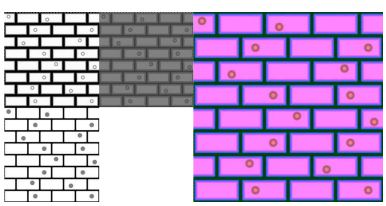


[그림 14]

노말맵, Look-Up Texture Mask 맵을 제외한 기타 맵들은 하나의 텍스처 파일의 RGBA 값으로 합쳐 단일 파일로 관리하였다.

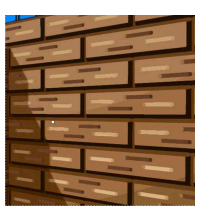
HOST = Height, Occlusion-Emmision, Specular, Transparant(그림 15)

Occlusion-Emmision 맵은 명도를 낮추는 효과와 높이는 효과를 128 값을 기준으로 판정하도록 하는 단일 맵으로 병합하였고, 적은 색상 수로 인한 디테일 부재를 대체하기 위해 같은 색상의 다른 밝기 값 (얼룩 등)을 표현에도 활용하였다.



[그림 15], [그림 16]

노말맵, HOST맵, LU 마스크맵을 적용한 Look-Up Textured 쉐이더를 적용해 빌딩, 지형 등 텍스처 디테일이 필요한 오브젝트에 적용하였다.(그림 17)

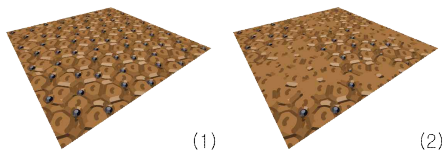


[그림 17]

2.4.3 무작위 지형 패턴 효과

Look-Up Textured 쉐이더로 지형을 표현할 때 동일한 형태의 텍스처를 단순 타일링하면 모습이 패턴화되어 유사한 형태가 반복되는 문제가 발생하였다.(그림 18.1)

이를 해결하기 위해 한 텍스처를 4등분해 각 조각에 랜덤 값 노이즈를 적용하고 그 값에 따라 최대 4가지 종류의 텍스처 Array중 하나를 적용해 무작위 패턴을 가지는 쉐이더를 제작하였다.(그림 18.2) 이러한 노이즈 가중치 기반 무작위 패턴 알고리즘은 절차적 오브젝트 배치에도 응용하였다.



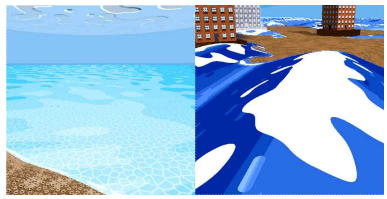
[그림 18]

2.5.1 바다 쉐이더

바다 쉐이더는 LUT 쉐이더를 기반으로 벡터 그래픽 스타일에 어울리는 파도 형태를 제작하는 것을 목적으로 하였다.

벡터 아트는 끝이 둥근 Stroke 도형의 특징상 뾰족한 돌출점이 부재한 특징을 가지고 있어 파도의 형태를 sin 함수 그래프 형상을 그대로 유지하도록 설계하였다.

게임의 진행에 따라 파도의 강도가 강해지고, 게임플레이에 변화가 생기기 때문에 파도 쉐이더 또한 이를 반영할 수 있도록 파도 강도에 따라 여러 요소가 변화하도록 제작하였다.



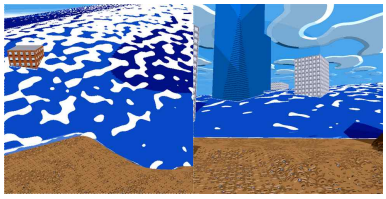
[그림 19], [그림 20]

2.5.2 파도 Vertex Shader

파도의 강도가 1 이하이면, 진폭만 증가하고, 파도의 강도가 1 이상이면 진폭과 파장이 같이 증가해 1 이상의 강도에서 sin(x) 그래프와 같은 형상을 유지하고자 하였다.

이를 달성하기 위해 파도의 강도를 a로 삼아 a<=1 일 때 a(sin(x)), a>1 일 때 a(sin(x/a))를 따르는 함수를 적용하였다.

이를 통해 매우 큰 강도에서 파도의 파장이 길어져 플레이어가 활동 가능한 넓은 저지대와 이를 덮치는 거대한 파고의 해일의 형태를 표현하고자 하였다.



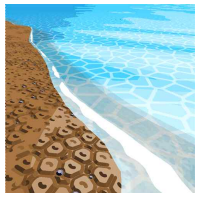
[그림 21], [그림 22]

2.5.3 파도 Fragment Shader

Dept 테스트로 연안 지역을 반투명하도록 제작하였고, 강도 증가에 따라 불투명 하게 하여 후반부 해저 탐사를 제한하고자 하였다.

Dept 테스트로 오브젝트 사이 접촉 지점을 구하고 펄린 노이즈를 적용해 접촉 지점에 거품 효과를 제작하였고, 높은 파고 부분에도 추가로 거품 효과를 적용하였다.

보로노이 노이즈로 제작된 텍스처로 코스틱 효과를 적용하고, 강도 값이 증가하면 코스틱 효과를 감소하도록 제작하였다.



[그림 23]

2.6 오브젝트(노드, RPOI, 장애물) 배치

2.6.1 오브젝트 배치를 불규칙적 노이즈를 이용하여 오브젝트들을 플레이할 때마다 랜덤한 위치를 가지게 된다.

```
public static float[,] GenerateIrregularNoiseMap(int mapWidth, int mapHeight,
    int seed, float scale, float irregularity, Vector2 offset)
{
    float[,] noiseMap = new float[mapWidth, mapHeight];

    System.Random prng = new System.Random(seed);
    offset = new Vector2(prng.Next(-100000, 100000)
            + offset.x, prng.Next(-100000, 100000) + offset.y);

    if (scale <= 0)
    {
        scale = 0.0001f;
    }

    float halfWidth = mapWidth / 2f;
    float halfHeight = mapHeight / 2f;

    for (int y = 0; y < mapHeight; y++)
    {
        for (int x = 0; x < mapWidth; x++) // Fixed: mapWidth instead of mapHeight
        {
            // Generate random noise value
            float randomValue = (float)prng.NextDouble();

            // Add offset
            float sampleX = (x - halfWidth + offset.x) / scale;
            float sampleY = (y - halfHeight + offset.y) / scale;

            // Add the random noise to the noise map
            noiseMap[x, y] = randomValue;
        }
    }
    return noiseMap;
```

[그림 24]불규칙 노이즈 코드

맵의 중심을 기준으로 샘플링을 진행하도록 해, 오프셋이 적용된 좌표로 이동하고,

맵의 좌표 (x, y)를 scale과 offset을 반영하여 정규화 된 위치로 변환하며,

noiseMap[x, y] = randomValue; 를 통해 (x, y) 위치의 노이즈 값으로 randomValue를 사용하여, 무작위 값을 0과 1 사이의 값으로 저장한다.

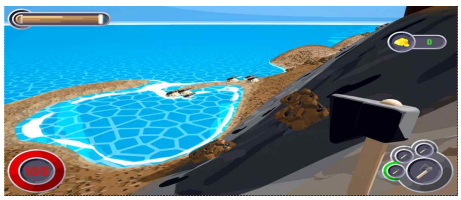
2.6.2 노드가 생성될 때 높낮이를 확인하여 이에 따라 다른 종류의 노드들이 생성되도록 하였다. 새로운 지역이 열림에 있어서 수면 높낮이에 따라 다르게 생성되어야 하기 때문이다.

```
GameObject prefabToPlace = null;
float heightY = hit.point.y;
if (heightY > NodedirtYMin && heightY <= NodedirtYMax)
{
    IsSand = false;
    position = new Vector3(x - (int)(width / 2f), heightY, y - (int)(height / 2f));
    // heightThreshold에 따라 다른 브러쉬 배열 선택
    if (noiseMap[x, y] > NodedirtThreshold1)
    {
        prefabToPlace = dirtlowNodePrefabs[prng.Next(dirtlowNodePrefabs.Length)];
    }
    else if (noiseMap[x, y] > NodedirtThreshold2)
    {
        prefabToPlace = dirtmediumNodePrefabs[prng.Next(dirtmediumNodePrefabs.Length)];
    }
    else if (noiseMap[x, y] > NodedirtThreshold3)
    {
        prefabToPlace = dirthighNodePrefabs[prng.Next(dirthighNodePrefabs.Length)];
    }
}
```

[그림 25] 높낮이를 확인하여 다른 노드들이 생성되는 코드

각각의 노드의 정해진 Threshold 값에 따라 노이즈 높이 오프셋과 비교하여 특정 높이 범위에 따라 prefab을 dirt와 sand로 구분하

여 각각에 맞게 배치한다.



[그림 26] dirt와 sand로 나뉘어진 오브젝트

노드 오브젝트가 지정된 범위 내에 여러 개 군집되도록 하였다.

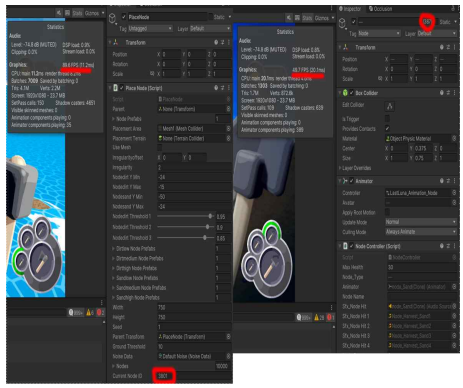
[그림 27] 군집된 오브젝트

2.7 최적화

Photon은 멀티플레이를 지원해주는 SDK이다. Photon에서 동기화를 하려면 Photon View와 Photon Animator View를 사용하여 동기화를 시켜줘야 한다. 문제는 Photon에서 Photon View의 개수가 한정적으로 지원을 해준다.

이를 해결 하기 위해서 부모 오브젝트에 동기화를 처리하는 소스 코드를 적용하여 오브젝트의 개수를 증가하고 메모리를 줄이는 효과를 얻었다.

유니티 Animator는 많은 GPU 자원을 사용한다. Animator을 실행하지 않아도 GPU 자원을 소모하게 되는데 이를 플레이어의 Boundary를 지정하여 플레이어의 Boundary 안에 들어오게 되면 Animator를 활성화 시켜 GPU 자원을 줄이는 효과를 얻었다.



[그림 28] 최적화 적용전(좌),최적화 적용후(우)

2. 아트 작업

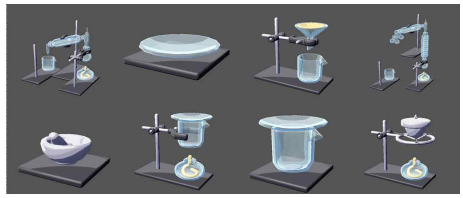
자원 아이템 – 6개

[그림 30] 자원 모델링

도구 아이템 3개

[그림 31] 도구 모델링

자동화 건물 – 8개

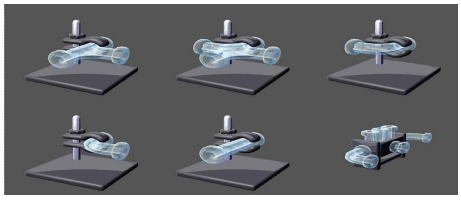


[그림 32] 자동화 건물 모델링

배경 오브젝트 모델링 – 5개

[그림 33] 배경 오브젝트 모델링

파이프 모델링 - 6개



[그림 34] 파이프 모델링

3. 결 론

3.1 장단점

3.1.1 장점

벡터 아트 그래픽을 이용하여 타 게임과 차별화된 아트 스타일을 보여줄 수 있다.

쉐이딩 색상이 사전에 지정되어 아티스트가 의도한 색감의 게임을 보여줄 수 있다.

3.1.2 단점

외부 자원 파밍 컨텐츠로 획득 가능한 자원의 종류와 유형이 한정되어 있어 외부 탐사 게임플레이 요소가 미흡하다.

쉐이더가 Directional Lights에만 의존해 점광원과 광원 색상에 영향을 받지 않아 표현 가능한 범위가 제약된다.

3.2 향후 추가 개발 또는 업그레이드 내용이 필요한 부분 등 서술.

절차적 생성을 이용하여 새로운 게임마다 지형 맵을 변화시키고자 하였으나 지형 크기 조절에 문제가 있어 플레이 가능 지역이 제한되어 Terrain 시스템으로 제작된 게임 맵을 사용할 수 밖에 없었다.

쉐이더가 광원의 색상과 밝기, 점광원에 따라 명암 처리가 이루어지도록 작업하고, 모델과 애니메이션을 개선해 시각적 단조로움을 개선할 필요가 있다.

튜토리얼을 제작하여 항후 게임의 이해도를 높여야한다.

4. 부 록

첨부된 LastMoon.exe 파일을 실행

4.1 게임 실행 환경:

다이렉트 11 이상 지원하는 컴퓨터

권장사양 CPU: 인텔 i7 8700,라이젠 3600x 이상 RAM 16GB 이상 GPU 1660super 이상 급의 컴퓨터

4.2 게임 진행

게임 시작시 닉네임,최대 인원수를 설정하고 랜덤 매칭을 눌러 시작

[그림 35] 게임 시작 화면

[그림 36] 매치메이킹 화면

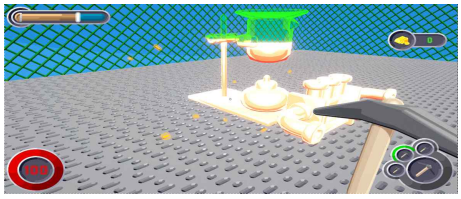
게임을 플레이

[그림 37] 게임플레이 화면

파도가 썰물일 때 자원을 파밍하고 밀물일 때 자동화 기지를 건설하면서 더 높은 빌딩으로 가기 위해 자원을 많이 수집

[그림 38] 자원 파밍 화면

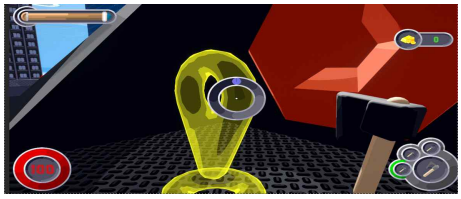
[그림 39] 자동 화 기지 UI



[그림 40] 자동화 기지 건설

자원을 모아서 e키를 눌러 벙커에 들어가게 되면 게임은 종료, 엔딩 영상이 나온 후 게임 결과가 나온다.

돌아가기 버튼을 누르면 게임 시작 화면으로 복귀한다.



[그림 41] 탈출 화면

[그림 42] 게임 엔딩 화면

[그림 43] 게임 결과 화면

4.4 문제 해결

4.4.1 개발된 게임의 제약사항

멀티플레이를 위한 동일한 네트워크에서 실행

4.4.2 문제 해결을 위한 연락처

01051899116 조영환
01026546199 김태현
01064951760 양영현

5. 참고문헌

[1]“(UnityShader) 12 URP Shader Graph”,낭낭하게 개발하기,2020년 2월 24일. 2024년 6월 20일 접속, https://walll4542.wixsite.com/watchthis/post/unityshader-12-urp-shader-graph