Realistic Water

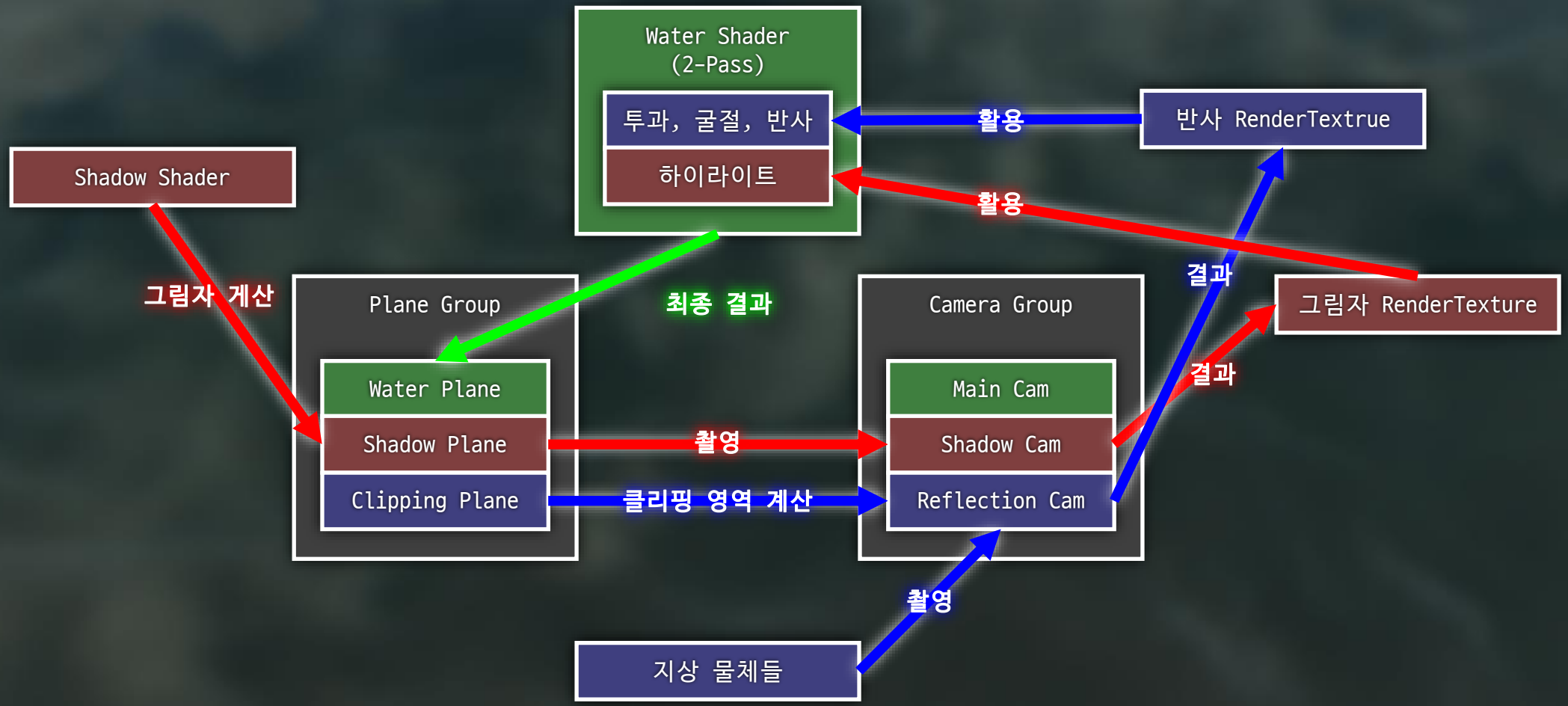
소프트웨어응용학부 201704044 백문하

프로젝트 개요

물은 투명하면서 표면의 파동에 따라 상이 굴절되고, 주변 풍경을 반사하는 특징을 가지고 있다.

여러 기술이 사용되기 때문에 유니티 엔진에서 범용적으로 쓸 수 있는 물 표면 셰이더를 만들어 보기로 했다.

프로젝트 구조

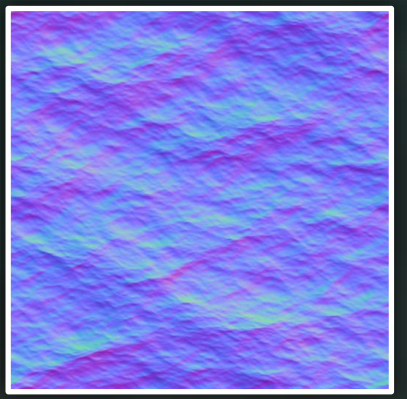

Water Shader
(2-Pass)
투과, 굴절, 반사
하이라이트
활용
반사 RenderTextrue
활용
Shadow Shader
그림자 계산
최종 결과
Plane Group
Water Plane
Shadow Plane
Clipping Plane
촬영
클리핑 영역 계산
Camera Group
Main Cam
Shadow Cam
Reflection Cam
결과
결과
그림자 RenderTexture
촬영
지상 물체들

사용 기술 설명 - 수면 노멀 맵핑

물결을 표현하기 위해 무한하게 이어지는 수면 노멀 맵 이미지들을 구했다.

UnpackNormal()로 노멀 벡터를 얻을 때, 노멀 텍스처의 UV 값에
_Time.x와 파도의 속도값을 곱한 값을 더하여 흐르게 만든다.
그렇게 얻은 노멀 벡터와 위를 향하는 벡터 (0, 1, 0)을
사용자가 설정하는 파도의 세기로
lerp() 하여 파도의 강도도 설정하게 만든다.

4가지 방향으로 이동하는 파도의 노멀값을 모두 더한 뒤,
4로 나누어 최종적인 노멀 벡터를 얻는다.



노멀 벡터는 반사와 투시 등 나머지 파트에서 활용한다.

사용 기술 설명 - 반사 하이라이트

반사 하이라이트를 구현하기 위해 블린 퐁 방식을 이용하였다.

라이팅 벡터와 뷰 벡터를 더하고 정규화하는 것으로 하프 벡터를 구한다.

하프 벡터를 노멀 벡터와 내적하여 하이라이트를 구하고, 거듭제곱 연산으로 하이라이트 영역을 좁게 만들었다.

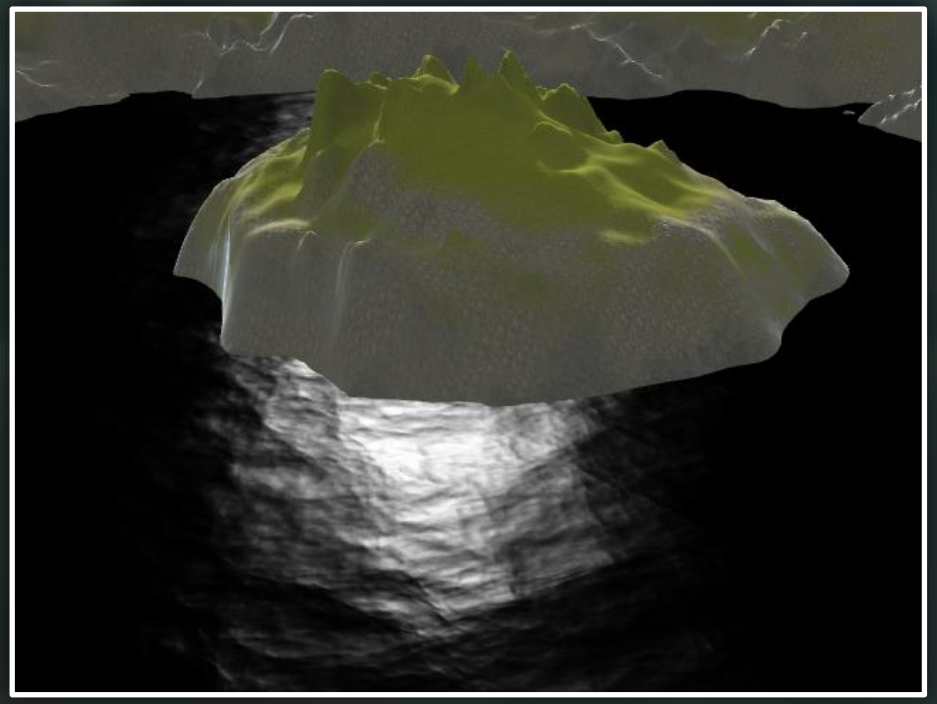
반사 하이라이트는 투과 및 배경 반사와는 상관 없이 그려지므로,

2-Pass 방식으로 덧그리게 만들었다.

```
float4 LightingWaterSpecHL(SurfaceOutput s, float3 lightDir, float3 viewDir, float atten) {
    float3 halfVec = normalize(lightDir + viewDir);
    float spec = pow(saturate(dot(halfVec, s.Normal)), 200);
    float4 final;
    final.rgb = 1;
    final.a = spec * s.Alpha;
    return final;
}
```

단, 이것만 구현하면 오른쪽처럼 그림자로 가려진 곳에도

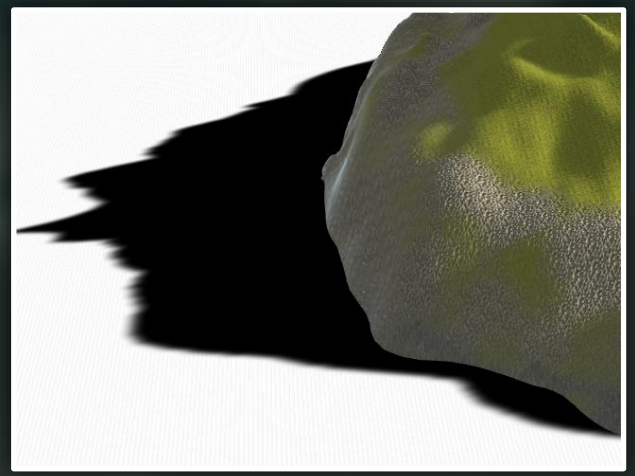
하이라이트가 나타나는 문제점이 존재한다.

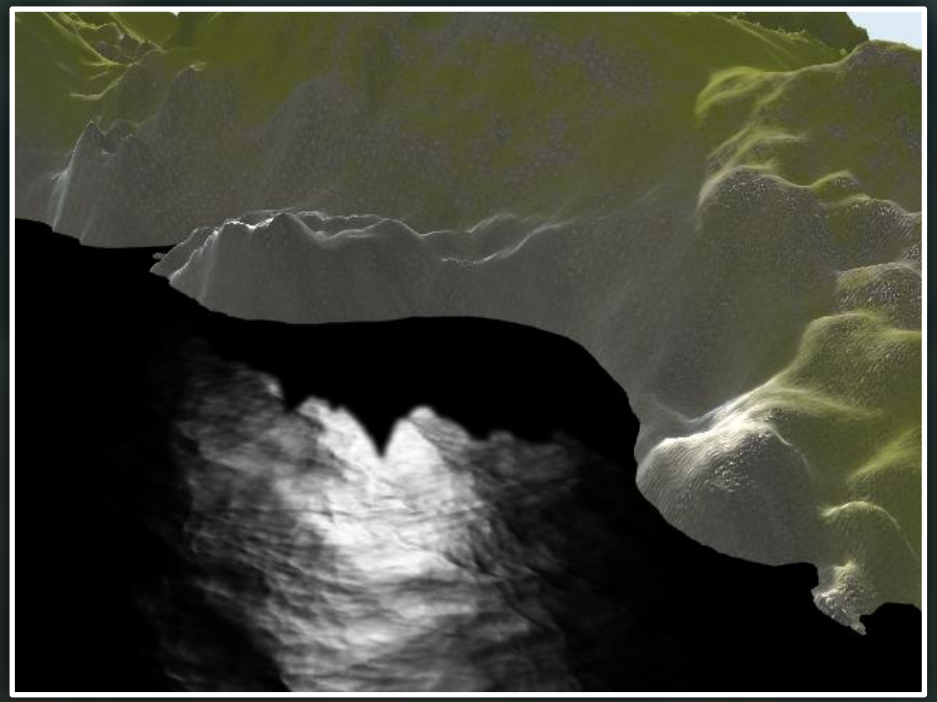


사용 기술 설명 - 하이라이트를 그림자로 가리기

그림자 촬영을 위한 카메라 2와 평면을 하나 더 설치한다. 카메라 2는 메인 카메라와 위치와 각도를 일치시킨다.
해당 평면은 메인 카메라에 보이지 않게 설정하고, 카메라 2의 결과는 RenderTexture로 출력시킨다.
별도의 버텍스/프래그먼트 셰이더를 이용하는데,
버텍스 출력인 v2f 구조체에 SHADOW_COORDS(2)를 추가하고 frag에서 LIGHT_ATTENUATION()를 이용하면,
그림자가 비치는 표면을 알아낼 수 있다.
다시 앞의 블린 퐁 셰이더로 돌아가서, RenderTexture 값으로
그림자가 있는 영역만 하이라이트의 투명도를 조정했다.

```
float4 frag(v2f i) : COLOR {
    float atten = LIGHT_ATTENUATION(i);
    float3 color = atten;
    return float4(color, 1);
}
ENDCG
```

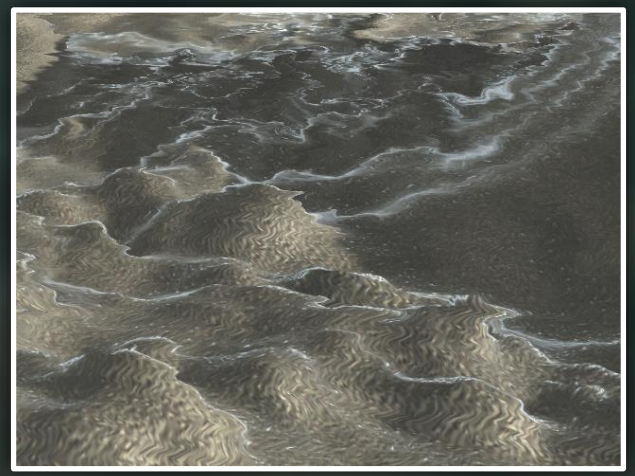


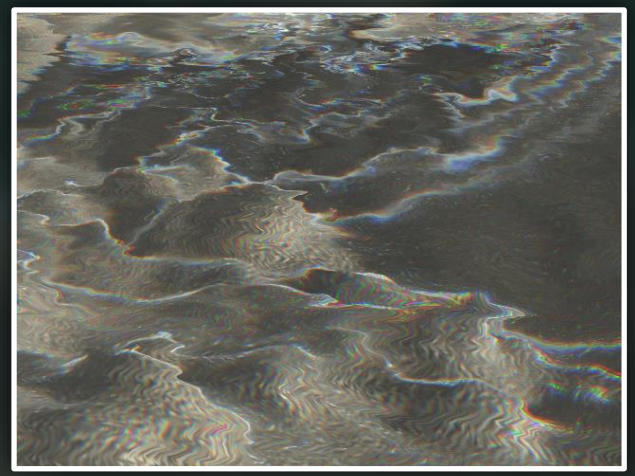


사용 기술 설명 - 물 투과 및 왜곡

Grabpass {} 를 이용하면 그리기 전 영역을 가져올 수 있고, 스크린 UV와 함께 투명한 물체를 표현할 수 있다.
수면의 노멀 벡터값을 UV 값에 더해준다면 물 아래의 상이 왜곡되어 보인다.

이 때, 배경을 RGB의 세 채널로 나누어
UV에 한번 더 노멀 벡터를 약간씩 더하거나 빼 주면,
빛이 무지갯빛 느낌으로 흩어지는 프리즘 효과를 구현할 수 있다.



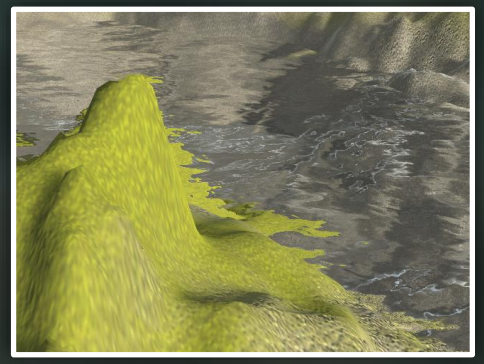


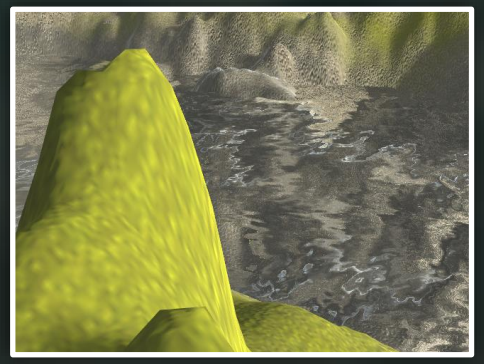
```
float underwater_r = tex2D(
    _BackgroundTexture,
    float2(
        jitter_uv.x + n_prism.r,
        jitter_uv.y
    )
).r;
float underwater_g = tex2D(
    _BackgroundTexture,
    float2(
        jitter_uv.x + n_prism.r / 2,
        jitter_uv.y + n_prism.g / 2
    )
).g;
float underwater_b = tex2D(
    _BackgroundTexture,
    float2(
        jitter_uv.x,
        jitter_uv.y + n_prism.g
    )
).b;
```

사용 기술 설명 - 높이를 이용한 왜곡 보정

물의 왜곡은 수면 아래의 화면만 활용하는 것이 아니기 때문에 왜곡 시 수면 위의 부분도 함께 흐트러진다.

_CameraDepthTexture로 메인 카메라의 깊이 텍스처를 참조하여 바닥까지의 거리를 알 수 있다.
또한 UNITY_Z_0_FAR_FROM_CLIPSPACE()로 표면까지의 거리를 알아낼 수 있다.
이 두 가지로 수면부터 바닥까지의 거리를 잴 수 있다.
이 때, 수면까지의 거리가 음수라면 수면 위에 있는 것이므로 왜곡을 수행하지 않게 만들면 문제가 해결된다.
jitter_uv는 노멀을 적용한 스크린 UV인데, 이를 원래의 스크린 UV로 되돌린다. 프리즘 효과도 비활성화한다.





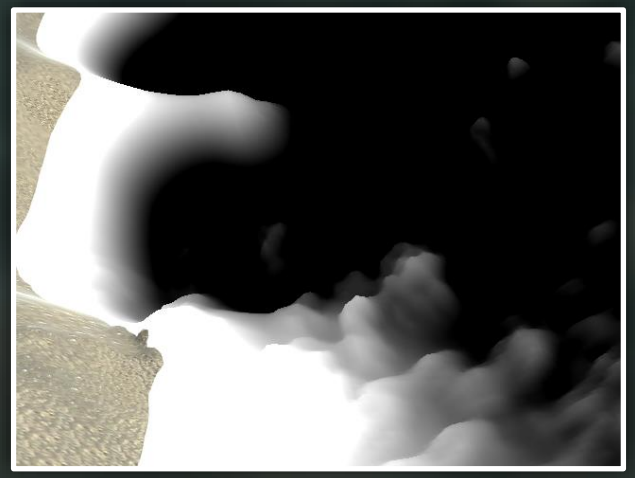
```
float background_depth = LinearEyeDepth(SAMPLE_DEPTH_TEXTURE(_CameraDepthTexture, jitter_uv));
float surface_depth = UNITY_Z_0_FAR_FROM_CLIPSPACE(IN.screenPos.z);
float depth_difference = background_depth - surface_depth;
float fog = 1 - saturate(depth_difference * _WaterFog1 - _WaterFog2);

if (depth_difference < 0) {
    jitter_uv = depth_uv;
    n_prism = 0;
    background_depth = LinearEyeDepth(SAMPLE_DEPTH_TEXTURE(_CameraDepthTexture, jitter_uv));
    depth_difference = background_depth - surface_depth;
    fog = 1 - saturate(depth_difference * _WaterFog1 - _WaterFog2);
}
```

사용 기술 설명 - 깊이에 따른 물의 색상 변화

앞에서 구한 수면으로부터 바닥까지의 거리를 이용해 깊이에 따른 물의 색상 농도를 표현할 수 있다.
깊이값에 사용자가 정할 수 있는 임의의 값을 곱하고 빼서
색이 생기는 시작점과 색이 짙어지는 기울기를 정하는 농도값을 구한다.

물 색의 RGB값과 배경을 농도값으로 lerp() 한 뒤,
그 결과와 배경을 물 색의 알파값으로 다시 lerp()하여 반투명한 물의 농도를 표현한다.



```
fog = 1 - saturate(depth_difference * _WaterFog1 - _WaterFog2);
```

```
float3 underwater_with_fog = lerp(_Color.rgb, underwater, fog);
underwater = lerp(underwater, underwater_with_fog, _Color.a);
```

사용 기술 설명 - 탁수에 그림자 적용

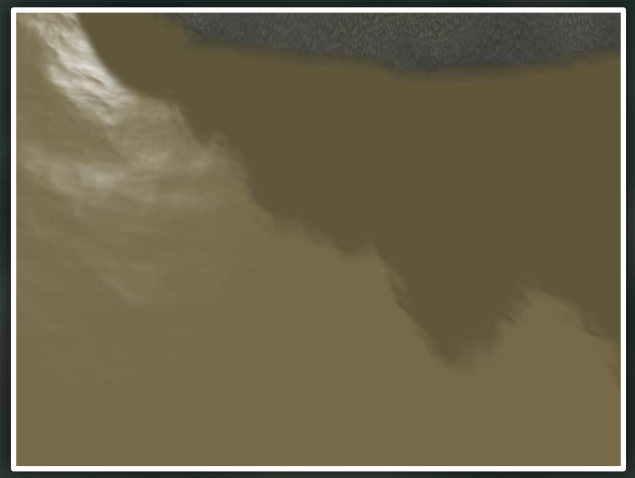
앞의 물의 농도와 색깔을 응용하면 흐린 강물이나 하수, 흙탕물 등 탁수를 구현할 수도 있다.

그런데 흙탕물 등 물이 흐릴 경우 위에 그림자가 드리워진다.

이를 구현하기 위해 그림자를 찍은 RenderTexture를 재활용하여

사용자가 그림자 값을 조정하면 그 값만큼 그림자가 드리워지도록 구현했다.

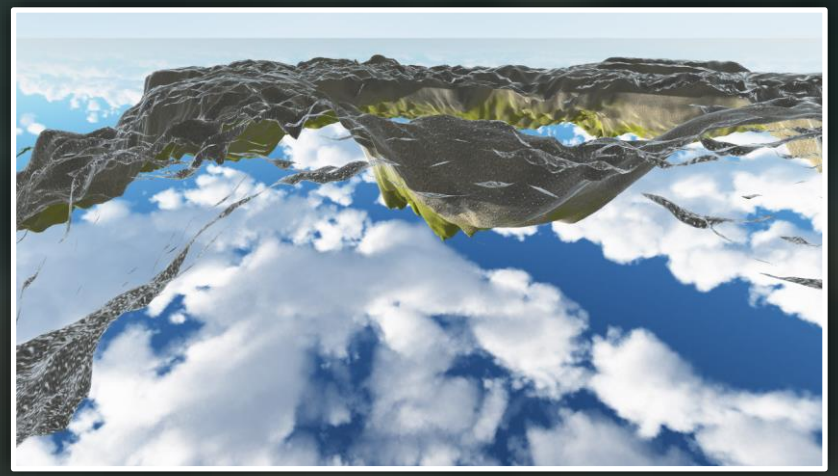
```
underwater = lerp(underwater, float3(0, 0, 0), (1 - tex2D(_ShadowRender, jitter_uv)) * (_ShadowOnWater));
```

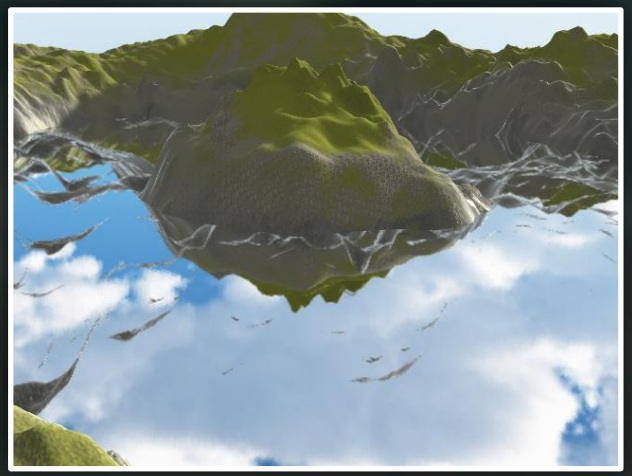


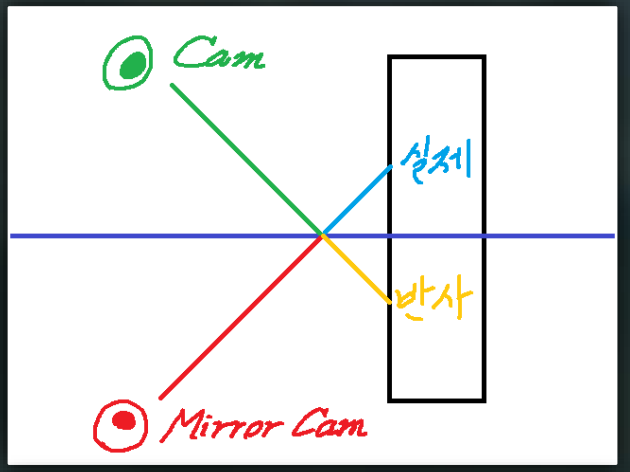
사용 기술 설명 - 배경 반사용 카메라

별도의 카메라를 만들어 수면 반대편에서 투사하면 실시간으로 반사된 장면을 얻을 수 있다.
그리고 그 결과를 별도의 RenderTexture에 출력하고 물 셰이더에서 이용하면 반사를 구현할 수 있다.

update 시 메인 카메라의 위치와 각도를 이용하여,
수면 정반대편에서 투사하도록 카메라 3의 위치와 각도를 조정한다.
그런데 수면 아래의 지형도 촬영되어 잘못된 결과가 나타나버린다.








```
transform.position = new Vector3(
    camera_Transform.position.x,
    -camera_Transform.position.y + 10,
    camera_Transform.position.z
);

transform.eulerAngles = new Vector3(
    -camera_Transform.eulerAngles.x,
    camera_Transform.eulerAngles.y,
    camera_Transform.eulerAngles.z + 180
);
```

사용 기술 설명 - 반사 카메라의 수면 아래 클리핑

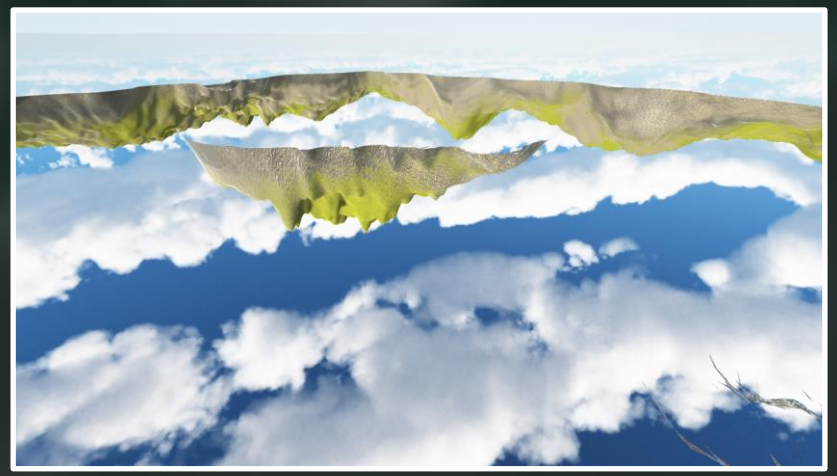
카메라에서 내용물을 그리지 않으려면, 카메라의 클리핑 영역을 설정하는 방법이 있다.

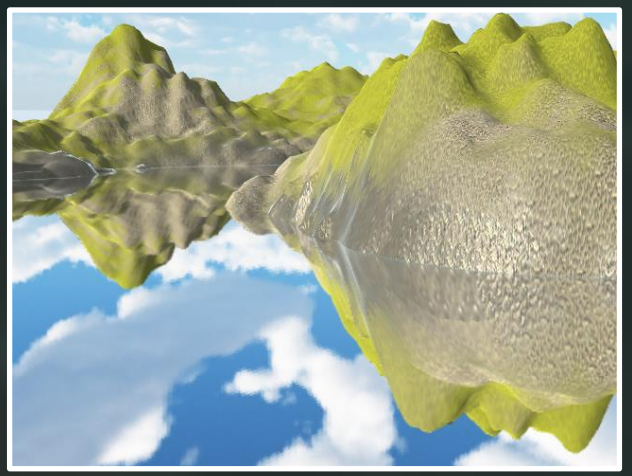
그런데 카메라의 클리핑 영역은 기본적으로 카메라와의 거리로 설정할 수 있다.

그러나, 유니티 엔진은 CalculateObliqueMatrix()를 이용해 비스듬한 영역의 클리핑 영역을 계산할 수 있다.

```
cam.ResetProjectionMatrix();
Plane plane = new Plane(clipping_plane.up, clipping_plane.position);
Vector4 camera_space_clip_plane = camera_space_plane(cam, clipping_plane.position, plane.normal);
cam.projectionMatrix = cam.CalculateObliqueMatrix(camera_space_clip_plane);
```

```
Vector4 camera_space_plane(
    Camera cam, Vector3 pos, Vector3 normal
) {
    Matrix4x4 mat = cam.worldToCameraMatrix;
    Vector3 c_pos = mat.MultiplyPoint(pos);
    Vector3 c_normal =
        mat.MultiplyVector(normal).normalized * -1;
    return new Vector4(
        c_normal.x,
        c_normal.y,
        c_normal.z,
        -Vector3.Dot(c_pos, c_normal)
    );
}
```

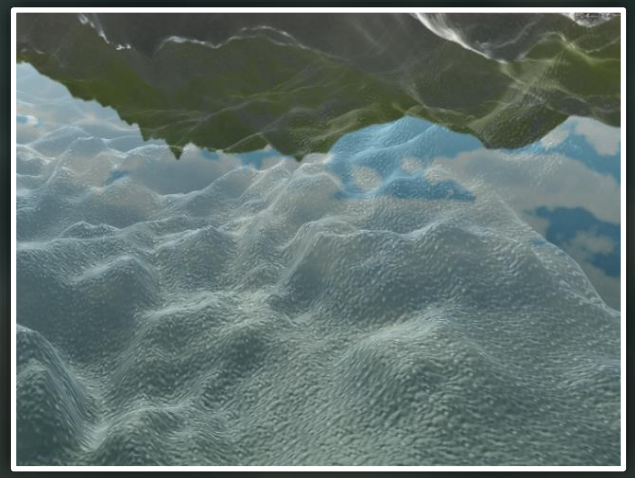


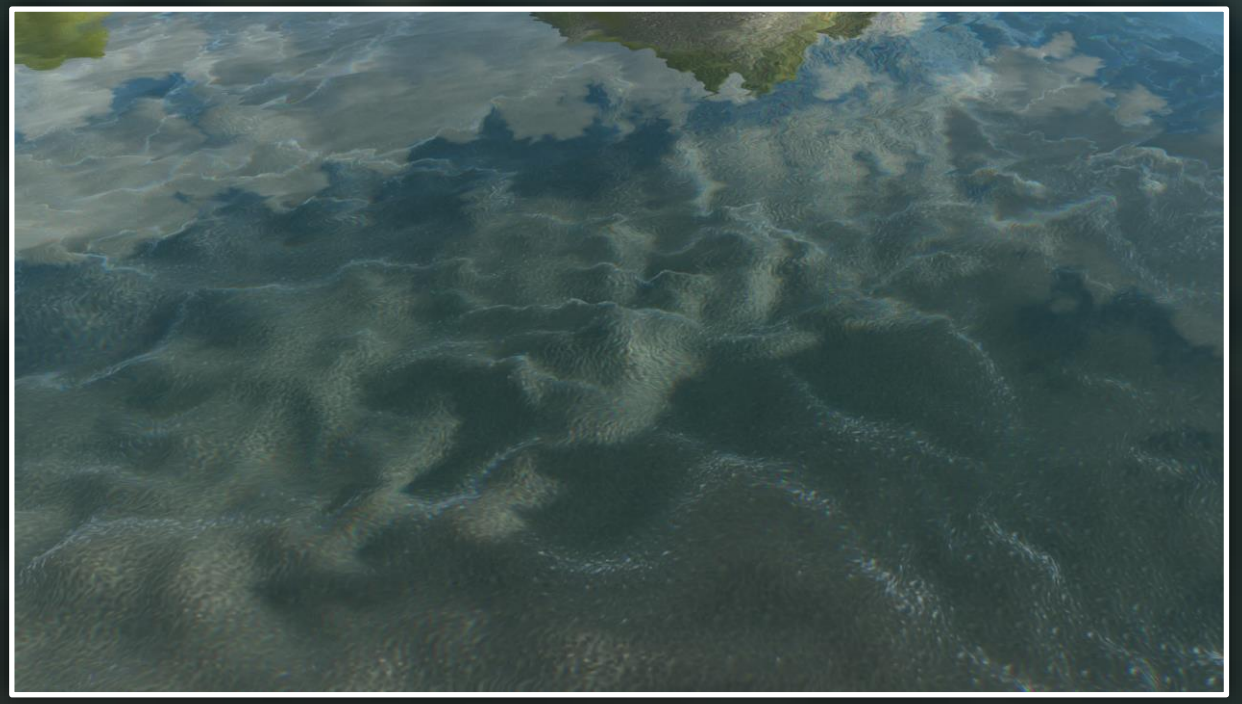


사용 기술 설명 - 프레넬을 이용한 투과와 반사

각도에 따라 투과와 반사되는 정도를 달리하기 위해 림 라이팅을 응용한다.
앞에서 얻어낸 굴절과 물의 색상이 적용된 배경과
반사된 상을 림 값으로 lerp()하여 구현한다.

```
float rim = pow(1 - abs(dot(IN.viewDir, n)), 3);
o.Emission = lerp(underwater, re.rgb, rim);
```

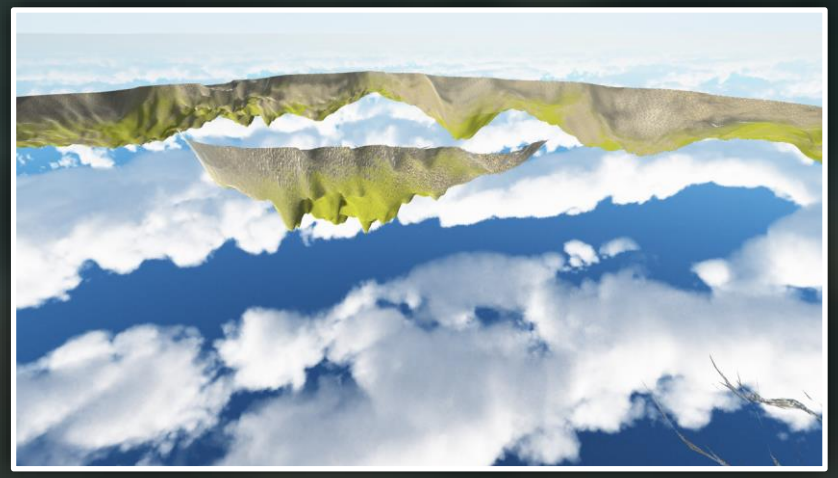


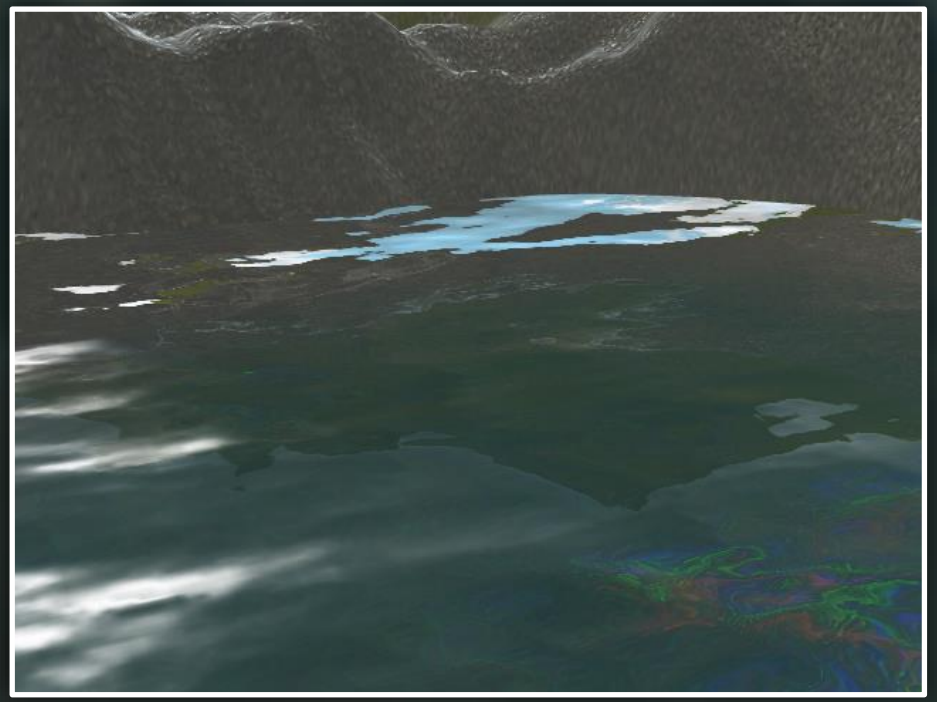


문제점

반사 카메라가 수면 아래를 반사하지 않게끔 만들 때,
카메라를 클리핑하는 방법을 사용했기 때문에 절단면은 뒷 풍경을 비추고 있다.
그런데 배경도 파도의 노멀 벡터에 의해 왜곡되므로
절단면의 뒷 풍경이 파도에 의해 모습을 보이게 되는 문제점이 존재한다.

뒷 풍경이 비슷한 지면이면 괜찮지만,
보통은 스카이박스가 노출되기 때문에 문제가 있다.





감사합니다.