# 라이팅 구조

* Lambert(램버트)
  + 환경광만 적용. Specular 없음.
  + SurfaceOutput 구조체 사용.
  + 출력.



* + 셰이더
    - *#pragma surface surf Lambert*
    - *SurfaceOutput 구조체 사용.*

*shader “Custom/Lambert”*

*{*

*Properties*

*{*

*tex (“tex”, 2D) = “white” {}*

*}*

*SubShader*

*{*

*Tags { “RenderType”=”Opaque” }*

*CGPROGRAM*

*#pragma surface surf Lambert*

*sampler2D tex;*

*struct Input*

*{*

*float2 uv;*

*};*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(tex, IN.uv);*

*o.Albedo = c.rgb;*

*o.Alpha = c.a;*

*}*

*ENDCG;*

*}*

*FallBack “Diffuse”*

*}*

* Blinn Phong(블린 퐁)
  + Specular 공식 사용.
  + Phong 을 라이트하게 구현.
  + SurfaceOutput 구조체 사용.

struct SurfaceOutput

{

half3 Albedo; // 기본 색상.

half3 Normal; // 노말맵.

half3 Emission; // 빛의 영향을 받지 않는 색상.

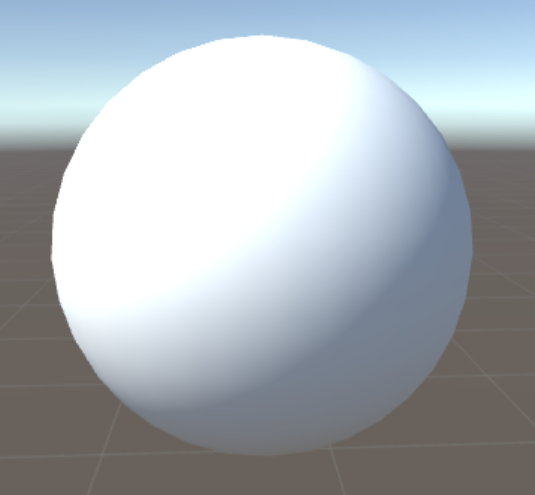
half3 Specular; // 하이라이트 수치.

half3 Gloss; // Specular 의 강도.

half3 Alpha; // 알파.

}

* + 출력



* + 셰이더

*shader “Custom/BlinnPhong”*

*{*

*Properties*

*{*

*tex (“tex”, 2D) = “white” {}*

*specColor(“Specular Color”, color) = (1,1,1,1);*

*}*

*SubShader*

*{*

*Tags { “RenderType”=”Opaque” }*

*CGPROGRAM*

*#pragma surface surf BlinnPhong*

*sampler2D tex;*

*struct Input*

*{*

*float2 uv;*

*};*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(tex, IN.uv);*

*o.Albedo = c.rgb;*

*o.Specular = 0.5;*

*o.Gloss = 1;*

*o.Alpha = c.a;*

*}*

*ENDCG;*

*}*

*FallBack “Diffuse”*

*}*

* Standard(스탠다드)
  + Unity5 부터 지원.
  + 물리 기반 렌더링 구조.
  + SurfaceOutputStandard 구조체와 SurfaceOutputStandardSpecular 구조체 사용.

struct SurfaceOutputStandard

{

fixed3 Albedo; // 기본 색상.

fixed3 Normal; // 노말맵.

fixed3 Emission; // 빛의 영향을 받지 않는 색상.

half Metallic; // 금속 재질.

half Smoothness; // 재질의 거친 정도.

half Occlusion; // 차폐되어 어둡게 되는 강도.

half Alpha; // 알파.

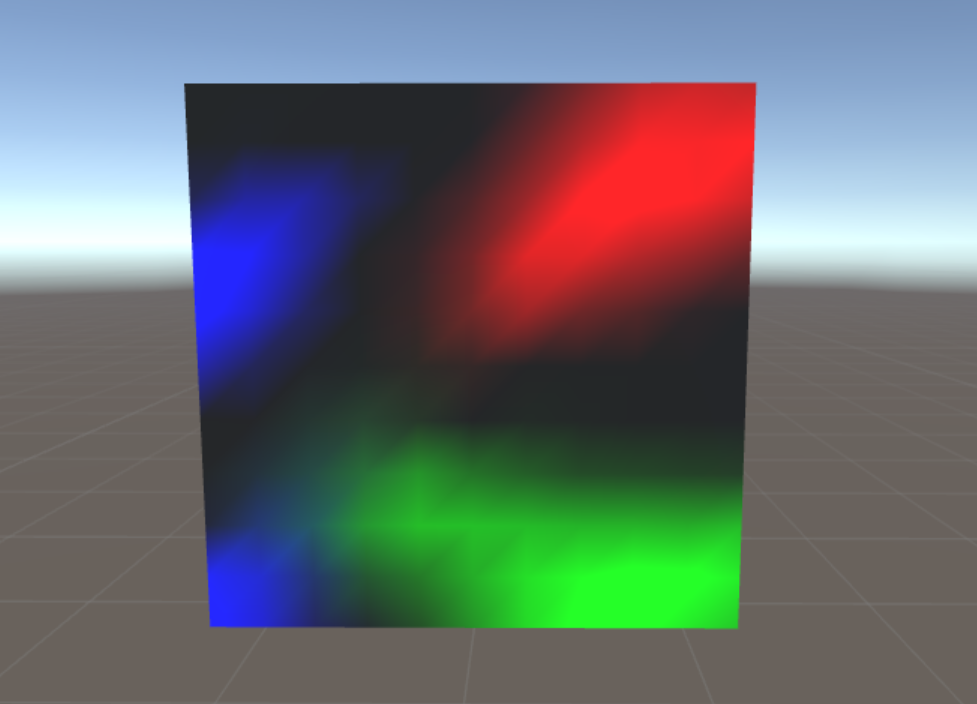
}

* + 아래 셰이더 효과들은 모두 스탠다드 기반 렌더링임.

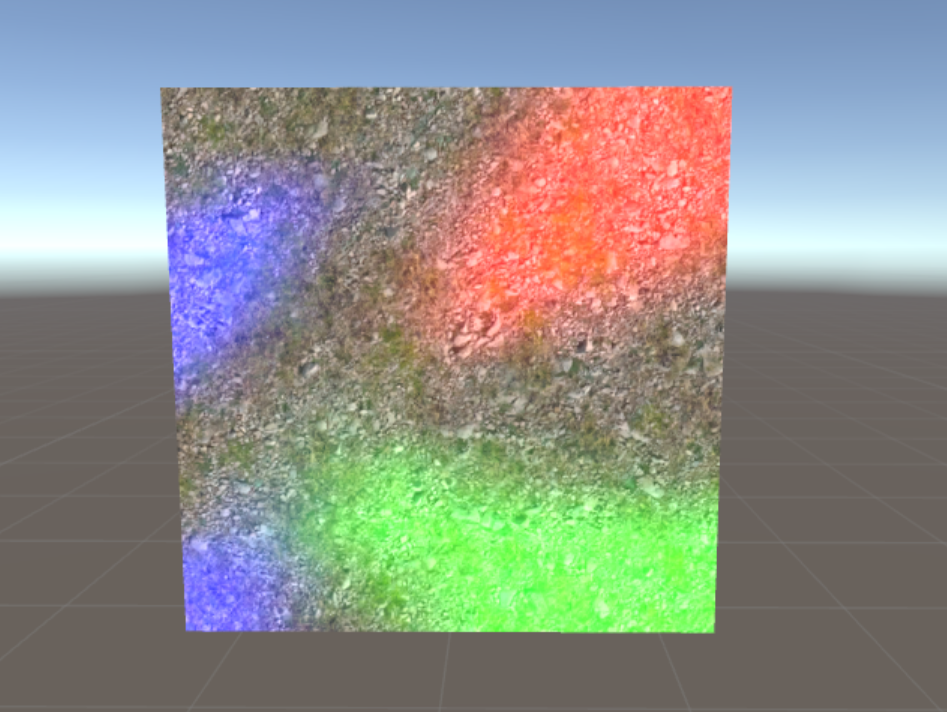
# 버텍스 컬러 이용

정점 셰이더의 입력으로 들어오는 정점의 요소는 위치, 텍스처 UV, 색상, 노말, 탄젠트 노말 등이 있을 수 있다. 이 중 색상 요소는 메시를 그릴 때 대부분 사용하지 않지만(텍스처로 바르기때문) 라이트맵, 앰비언트 오클루전 등 저렴한 특수 효과를 낼 수 있기 때문에 중요하다.

* 메시에 버텍스 컬러 적용.
  + 유니티는 직접적으로 지원하지 않는다(언리얼 엔진은 지원함).
  + 3ds max 에서 작업.
* 버텍스 컬러 원본 출력.

정점에 색상을 직접 입힌 사각형(텍스처 적용 아님).

* 텍스처 적용 렌더링.



단순히 텍스처의 텍셀과 정점 컬러를 더한 모습이다. 이런 식으로 정점 색을 혼합하면 라이트맵 등의 효과를 구현할 수 있다.

* 셰이더 구현
  + 버텍스 컬러 매터리얼 및 셰이더 생성.
  + 매터리얼 : 텍스처 적용.
  + 셰이더 : 버텍스 컬러와 텍셀 혼합.

*shader “Custom/VertexColor”*

*{*

*Properties*

*{*

*tex (“Albedo”, 2D) = “white” {}*

*}*

*SubShader*

*{*

*Tags { “RenderType” = “Opaque” }*

*CGPROGRAM*

*#pragma surface surf Standard noambient*

*sampler2D tex;*

*struct Input*

*{*

*float2 uv;*

*float4 color : COLOR;*

*}*

*void surf(Input IN, inout SurfaceOutputStandard o)*

*{*

*fixed4 texel = tex2D(tex, IN.uv);*

***// 텍셀과 버텍스 컬러를 더함.***

*o.Albedo = texel.rgb + IN.color.rgb;*

*o.Alpha = texel.a;*

*}*

*ENDCG*

*}*

*FallBack “Diffuse”*

*}*

# 금속 재질 & 스페큘라

* SurfaceOutputStandard 구조체

SurfaceOutputStandard구조체 내에 금속 재질 과 스페큘라 요소가 정의되어 있다.

struct SurfaceOutputStandard

{

fixed3 Albedo;

fixed3 Normal;

fixed3 Emission;

half Metallic; // Metallic. 금속 재질 효과.

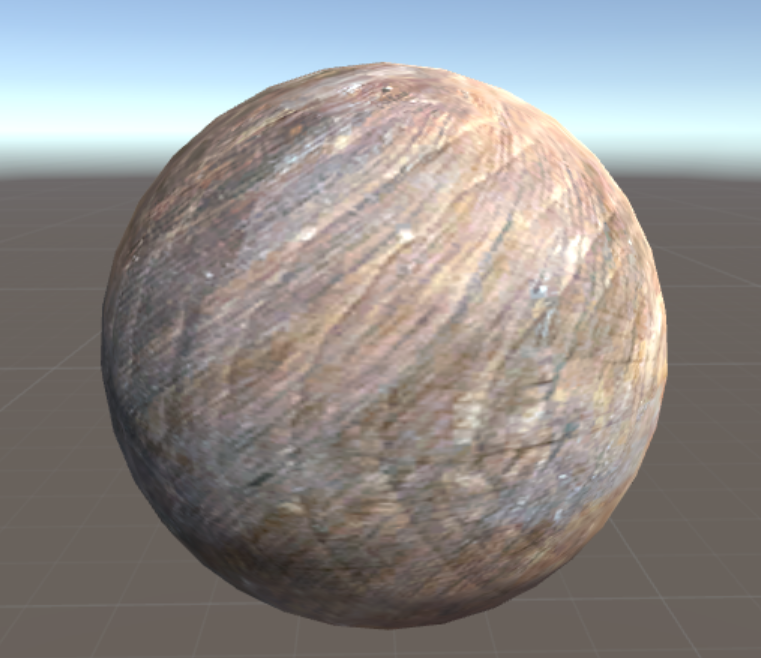
half Smothness; // Specular. 정반사.

half Occlusion;

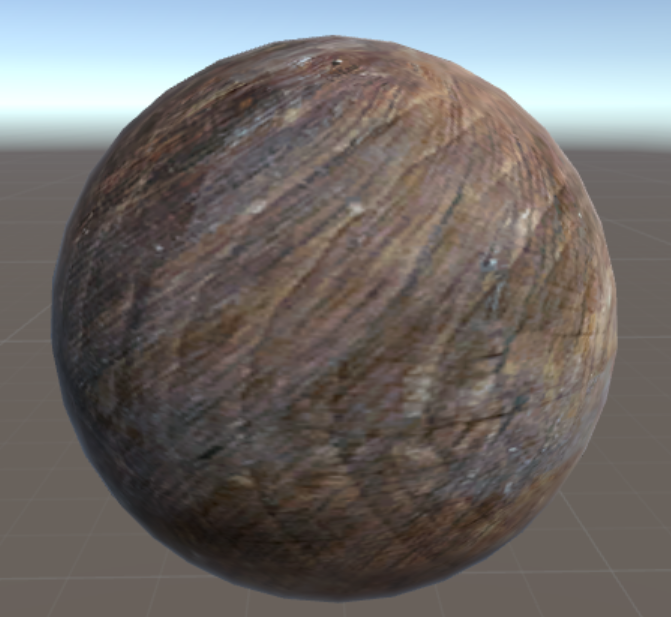
half Alpha;

}

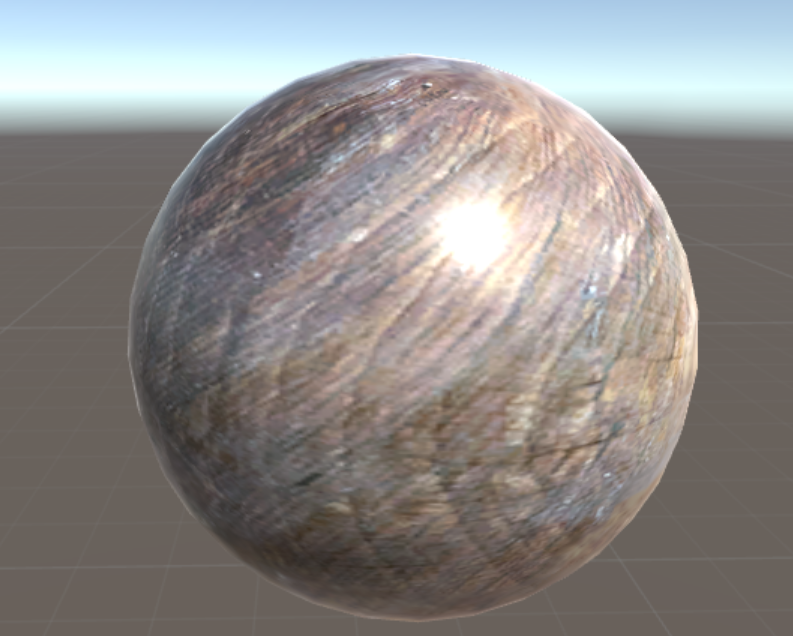
* 두 변수 값을 조절하면 각각의 효과를 낼 수 있음.
  + 원본 이미지



* + metallic 값을 1에 가깝게 할 경우



* + smoothness 값을 1로 가깝게 할 경우.



* 셰이더 적용.

*shader “Custom/Surface”*

*{*

*Properties*

*{*

*tex (“Albedo”, 2D) = “white” {}*

*metallic (“Metallic”, Range(0, 1)) = 0*

*smothness (“Specular, Range(0, 1)”) = 0*

*}*

*SubShader*

*{*

*Tags { “RenderType” = “Opaque” }*

*CGPROGRAM*

*#pragma surface surf Standard noambient*

*sampler2D tex;*

*float metallic;*

*float smothness;*

*struct Input*

*{*

*float2 uv;*

*}*

*void surf(Input IN, inout SurfaceOutputStandard o)*

*{*

*fixed4 texel = tex2D(tex, IN.uv);*

*o.Albedo = texel.rgb;*

*o.metallic = metallic;* ***// 1에 가까울수록 금속효과 큼.***

*o.Smothness = smothness;* ***// 1에 가까울수록 정반사효과 큼.***

*o.Alpha = texel.a;*

*}*

*ENDCG*

*}*

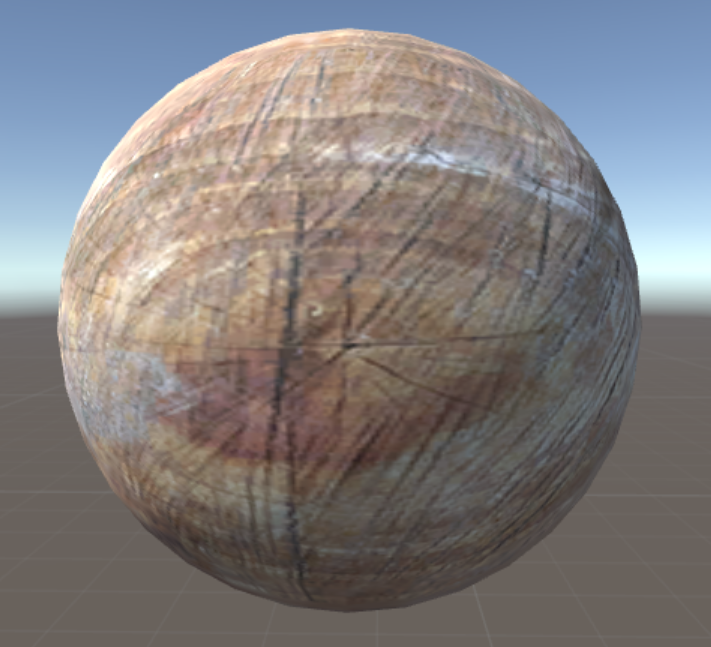
*FallBack “Diffuse”*

*}*

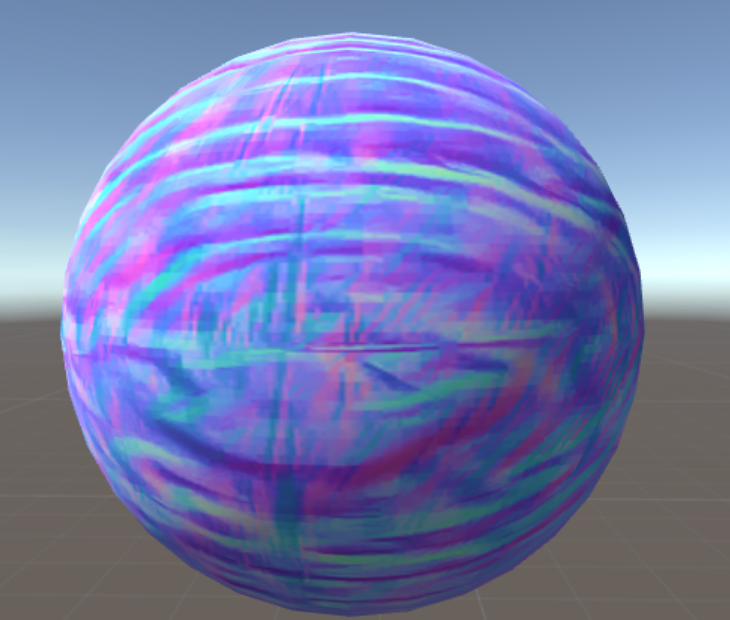
# 노말맵

노말 텍스처를 사용하여 음영을 표현.

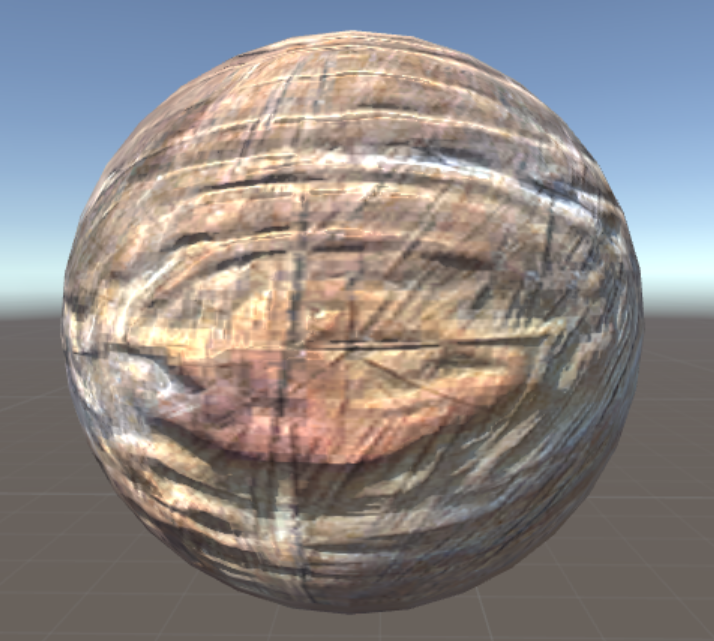
* 노말맵 생성 방법
  + ZBrush, Mudbox 와 같은 Sculpting 툴을 이용.
  + 3ds max 에서 폴리곤 모델링 후 Render To Texture 기능 이용하여 추출.
  + Crazybump 툴 이용 : 원본 이미지의 음영을 기반으로 자동으로 간단하게 추출.
  + 유니티 내장 기능 이용.
    - 일반 텍스처 선택, Inspector 에서 Texture type 을 Normalmap으로 변경.
    - Create from GrayScale 을 선택하고, Bumpiness 나 Filtering 을 적절히 조절.
    - Apply 를 누르면 일반 텍스처가 NormalMap 으로 추출됨.
  + diffuse 텍스처만 출력.



* + 노말맵 이미지만 직접 출력



* + 노말맵 적용.



* 셰이더 적용.
  + Properties 에서 노말맵 변수 생성.
  + 노말맵 텍셀값 추출. DxTnm 파일 포맷의 경우 UnpackNormal 함수 이용 추출.
  + SurfaceOutputStandard 구조체의 Normal 변수에 노말맵 텍셀값 대입.

*shader “Custom/NormalMap”*

*{*

*Properties*

*{*

*tex (“Albedo”, 2D) = “white” {}*

*normalmap (“NormalMap”, 2D) = “bump” {}* ***// 노말맵 속성 준비.***

*}*

*SubShader*

*{*

*Tags { “RenderType” = “Opaque” }*

*CGPROGRAM*

*#pragma surface surf Standard noambient*

*sampler2D tex;*

*sampler2D normalmap;* ***// 노말맵 텍스처.***

*struct Input*

*{*

*float2 uv;*

*float2 uv\_normalmap;* ***// 노말맵 텍스처 uv.***

*}*

*void surf(Input IN, inout SurfaceOutputStandard o)*

*{*

*fixed4 texel = tex2D(tex, IN.uv);*

*o.Albedo = texel.rgb;*

*o.Alpha = texel.a;*

*o.Normal = UnpackNormal(tex2D (normalmap, IN.uv\_normalmap));* ***// 노말맵 텍셀값을 언팩하여 대입.***

*}*

*ENDCG*

*}*

*FallBack “Diffuse”*

*}*

# 오클루전

환경광이 닿지 못하는 매우 구석진 부분의 추가적인 음영을 표현.

* 오클루전 맵 생성 방법
  + 툴로 추출 가능.

# 램버트 라이트 커스텀

가장 간단한 라이팅 구현으로 라이트 벡터와 노말 벡터의 내적으로 음영을 표현.

* 커스텀 셰이더 적용 전 원본.



* 커스텀 셰이더 생성
  + 빨간색으로 출력.



*Shader "Custom/CustomLight"*

*{*

*Properties*

*{*

*\_MainTex("Albedo (RGB)", 2D) = "white" {}*

*}*

*SubShader*

*{*

*Tags{ "RenderType" = "Opaque" }*

*LOD 200*

*CGPROGRAM*

*#pragma surface surf Custom* ***// 커스텀 셰이더 함수 이름 설정.***

*#pragma target 3.0*

*sampler2D \_MainTex;*

*struct Input*

*{*

*float2 uv\_MainTex;*

*};*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

*o.Albedo = c.rgb;*

*o.Alpha = c.a;*

*}*

***// 커스텀 셰이더 함수***

*float4 LightingCustom(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*return float4(1, 0, 0, 1);* ***// 빨간색으로 출력.***

*}*

*ENDCG*

*}*

*FallBack "Diffuse"*

*}*

* Lambert 라이트 연산.
  + Normal 과 lightDir 의 내적값을 음영으로 처리. LightingCustom 함수를 아래와 같이 변경.

*float4 LightingTest(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*float NdotL = saturate(dot(s.Normal, lightDir));* ***// 0 ~ 1.***

*return NdotL;*

*}*

* + 출력 결과.



* 노말맵 적용
  + 셰이더 코드

*Shader "Custom/CustomLight"*

*{*

*Properties*

*{*

*\_MainTex("Albedo (RGB)", 2D) = "white" {}*

*\_BumpMap(“NormalMap”, 2D) = “bump” {}*

*}*

*SubShader*

*{*

*Tags{ "RenderType" = "Opaque" }*

*LOD 200*

*CGPROGRAM*

*#pragma surface surf Custom* ***// 커스텀 셰이더 함수 이름 설정.***

*#pragma target 3.0*

*sampler2D \_MainTex;*

*sampler2D \_BumpMap;*

*struct Input*

*{*

*float2 uv\_MainTex;*

*float2 uv\_BumpMap;*

*};*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

*o.Albedo = c.rgb;*

*o.Normal = UnpackNormal(tex2D(\_BumpMap, IN.uv\_BumpMap));*

*o.Alpha = c.a;*

*}*

***// 커스텀 셰이더 함수***

*float4 LightingCustom(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*float NdotL = saturate(dot(s.Normal, lightDir));* ***// 0 ~ 1.***

*return NdotL;*

*}*

*ENDCG*

*}*

*FallBack "Diffuse"*

*}*

* + 출력 결과



* 하프 램버트
  + 밸브(Valve)의 논문인 Shading in Valve’s Source Engine 에 발표된 Lambert 라이팅의 수정 공식.
  + 기존 Lambert 라이트는 cos 그래프 특성상 밝다가 너무 갑자기 검게 음영이 떨어지는 단점이 있었음.
  + 단순히 내적값에 0.5 를 곱하고 다시 0.5를 더하기만 하면 됨.

*float4 LightingCustom(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*float NdotL = dot(s.Normal, lightDir) \* 0.5 + 0.5;*

*return NdotL;*

*}*

* + 출력 결과.



* 하프 램버트 완성
  + 소스 코드

*float4 LightingCustom(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*float NdotL = dot(s.Normal, lightDir) \* 0.5 + 0.5;*

*float4 final;*

*final.rgb = NdotL \* s.Albedo \* \_LightColor0.rgb \* atten;*

*final.a = s.Alpha;*

*return final;*

*}*

* + \_LightColor0.rgb : 내장 변수. 조명의 색상이나 강도를 표현.
  + atten : 빛의 감쇠 현상을 시뮬레이트.
  + 출력 결과.



# 림 라이트

물체가 역광에 있을 때 외곽선이 밝아 보이는 현상을 구현.

* 원리
  + 카메라 시각에서 메시의 외곽선에 가까운 노말 벡터일수록 수직임(내적이 0에 가까워짐).
  + 두 벡터의 각도 차이가 벌어질수록 흰색이 되도록 함(1 – 내적값).
  + 흰 테두리를 얇게 만들어 줌(지수 함수 이용).
* 구현

*Shader "Custom/rim"*

*{*

*Properties*

*{*

*\_MainTex("Albedo (RGB)", 2D) = "white" {}*

*\_BumpMap("NormalMap", 2D) = "bump" {}*

*\_RimColor("RimColor", Color) = (1,1,1,1)*

*\_RimPower("RimPower", Range(1, 10)) = 3*

*}*

*SubShader*

*{*

*Tags{ "RenderType" = "Opaque" }*

*LOD 200*

*CGPROGRAM*

*#pragma surface surf Lambert*

*#pragma target 3.0*

*sampler2D \_MainTex;*

*sampler2D \_BumpMap;*

*float4 \_RimColor;*

*float \_RimPower;*

*struct Input*

*{*

*float2 uv\_MainTex;*

*float2 uv\_BumpMap;*

*float3 viewDir;*

*};*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

*o.Albedo = c.rgb;*

*o.Normal = UnpackNormal(tex2D(\_BumpMap, IN.uv\_BumpMap));*

***// 노말 벡터와 뷰 벡터 내적.***

*float rim =* ***saturate(dot(o.Normal, IN.viewDir));***

***// 지수 분포 적용.***

*o.Emission =* ***pow(1 - rim, \_RimPower) \* \_RimColor.rgb;***

*o.Alpha = c.a;*

*}*

*ENDCG*

*}*

*FallBack "Diffuse"*

*}*

* 결과



# 홀로그램

림 라이트 기반으로 알파블렌딩을 이용하면 홀로그램 형상을 출력할 수 있음.

* 원리
  + 림 라이트를 구한다.
  + 림 라이트 값을 알파로 적용한다.

. 외곽 부분을 제외한 나머지 부분은 어두우므로(0에 가까움) 이 값을 알파로 적용하면 투명해지는 효과를 노림.

* 구현

*Shader "Custom/hologram"*

*{*

*Properties*

*{*

*\_MainTex("Albedo (RGB)", 2D) = "white" {}*

*}*

*SubShader{*

*Tags{ "RenderType" = "Transparent" "Queue"="Transparent"}*

*LOD 200*

*CGPROGRAM*

*#pragma surface surf Lambert noambient alpha:fade*

*#pragma target 3.0*

*sampler2D \_MainTex;*

*struct Input*

*{*

*float2 uv\_MainTex;*

*float3 viewDir;*

*};*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

*o.Emission = float3(0,1,0);* ***// 녹색으로 출력.***

*float rim = saturate(dot(o.Normal, IN.viewDir));*

*rim = pow(1-rim, 3);*

***o.Alpha = rim; // 알파값을 림 라이트 값으로 적용. 투명 효과.***

*}*

*ENDCG*

*}*

*FallBack "Diffuse"*

*}*

* 결과



# 블린 퐁(BLINN PHONG)

램버트 라이트 효과에 하이라이트(스페큘라) 효과를 추가적으로 구현. 퐁 라이트 모델보다 연산이 라이트 하다. 반사벡터를 구할 필요가 없다.

* 원리
  + 하프 벡터 구하기 : 라이트 벡터와 뷰 벡터의 중간 벡터.
  + 기본 음영 값 구하기 : 하프 벡터와 노말 벡터의 내적.
  + 하이라이트 값을 구하여 기본 음영 값에 적용.
* 코드

*Shader "Custom/Blinn-Phong"*

*{*

*Properties*

*{*

*\_MainTex ("Albedo (RGB)", 2D) = "white" {}*

*\_BumpMap ("NormapMap", 2D) = "bump" {}*

*}*

*SubShader {*

*Tags { "RenderType"="Opaque" }*

*LOD 200*

*CGPROGRAM*

*#pragma surface surf Blinn noambient*

*#pragma target 3.0*

*sampler2D \_MainTex;*

*sampler2D \_BumpMap;*

*struct Input*

*{*

*float2 uv\_MainTex;*

*float2 uv\_BumpMap;*

*};*

*UNITY\_INSTANCING\_BUFFER\_START(Props)*

*UNITY\_INSTANCING\_BUFFER\_END(Props)*

*void surf (Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D (\_MainTex, IN.uv\_MainTex);*

*o.Albedo = c.rgb;*

*o.Normal = UnpackNormal(tex2D(\_BumpMap, IN.uv\_BumpMap));*

*o.Alpha = c.a;*

*}*

*float4 LightingBlinn(SurfaceOutput s, float3 lightDir, float3 viewDir, float atten)*

*{*

*float4 final;*

***// Lambert.***

*float3 diffuse;*

*float NdotL = saturate(dot(s.Normal, lightDir));*

*diffuse = NdotL \* s.Albedo \* \_LightColor0.rgb \* atten;*

***// Specular.***

***// 1. 하프 벡터 구하기.***

*float3 halfVector =* ***normalize(lightDir + viewDir);***

***// 2. 하프 벡터와 노말 벡터의 내적 값을 스페큘라 값으로.***

***// 100 제곱 처리로 스페큘라 강도 처리.***

*float specular =* ***saturate(dot(halfVector, s.Normal));***

*specular =* ***pow(specular, 100);***

*// Lambert + Specular.*

*final.rgb = diffuse.rgb + specular;*

*final.a = s.Alpha;*

*return final;*

*}*

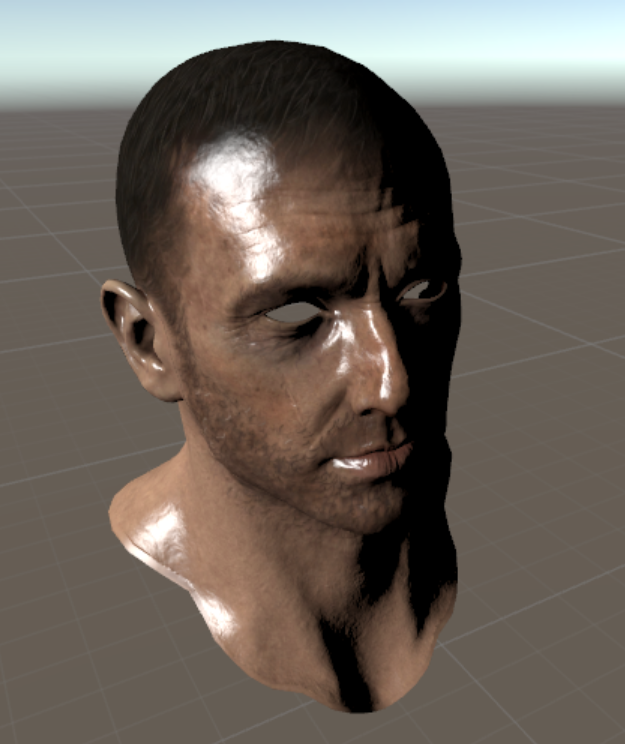
*ENDCG*

*}*

*FallBack "Diffuse"*

*}*

* 결과



# 스페큘라 맵

위 블린 퐁(Blinn-Phong) 모델의 하이라이트 부분을 텍스처를 이용하여 좀 더 세부적으로 묘사.

* 원리
  + 스페큘라 맵 적용.
  + 스페큘라 맵의 알파값을 SurfaceOutput 구조체의 \_Gloss 에 대입.
  + 커스텀 함수 내에서 구한 스페큘라 값에 \_Gloss 값을 곱함.
* 코드

*Shader "Custom/Blinn-Phong"*

*{*

*Properties*

*{*

*\_MainTex("Albedo (RGB)", 2D) = "white" {}*

*\_BumpMap("NormapMap", 2D) = "bump" {}*

***\_SpecularMap("Gloss Tex", 2D) = "white" {}***

*}*

*SubShader{*

*Tags{ "RenderType" = "Opaque" }*

*LOD 200*

*CGPROGRAM*

*#pragma surface surf Blinn noambient*

*#pragma target 3.0*

*sampler2D \_MainTex;*

*sampler2D \_BumpMap;*

***sampler2D \_SpecularMap;***

*struct Input*

*{*

*float2 uv\_MainTex;*

*float2 uv\_BumpMap;*

*float2 uv\_SpecularMap;*

*};*

*UNITY\_INSTANCING\_BUFFER\_START(Props)*

*UNITY\_INSTANCING\_BUFFER\_END(Props)*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

***// 스페큘라맵의 uv 위치에 해당하는 텍셀.***

***fixed4 s = tex2D(\_SpecularMap, IN.uv\_SpecularMap);***

*o.Albedo = c.rgb;*

*o.Normal = UnpackNormal(tex2D(\_BumpMap, IN.uv\_BumpMap));*

***o.Gloss = s.a; // 스페큘라맵의 알파값 대입.***

*o.Alpha = c.a;*

*}*

*float4 LightingBlinn(SurfaceOutput s, float3 lightDir, float3 viewDir, float atten)*

*{*

*float4 final;*

*// Lambert.*

*float3 diffuse;*

*float NdotL = saturate(dot(s.Normal, lightDir));*

*diffuse = NdotL \* s.Albedo \* \_LightColor0.rgb \* atten;*

*// Specular.*

*float3 halfVector = normalize(lightDir + viewDir);*

*float specular = saturate(dot(halfVector, s.Normal));*

***// 스페큘라 맵 알파값을 기존 스페큘라 값에 적용.***

***specular = pow(specular, 100) \* s.Gloss;***

*// Lambert + Specular.*

*final.rgb = diffuse.rgb + specular;*

*final.a = s.Alpha;*

*return final;*

*}*

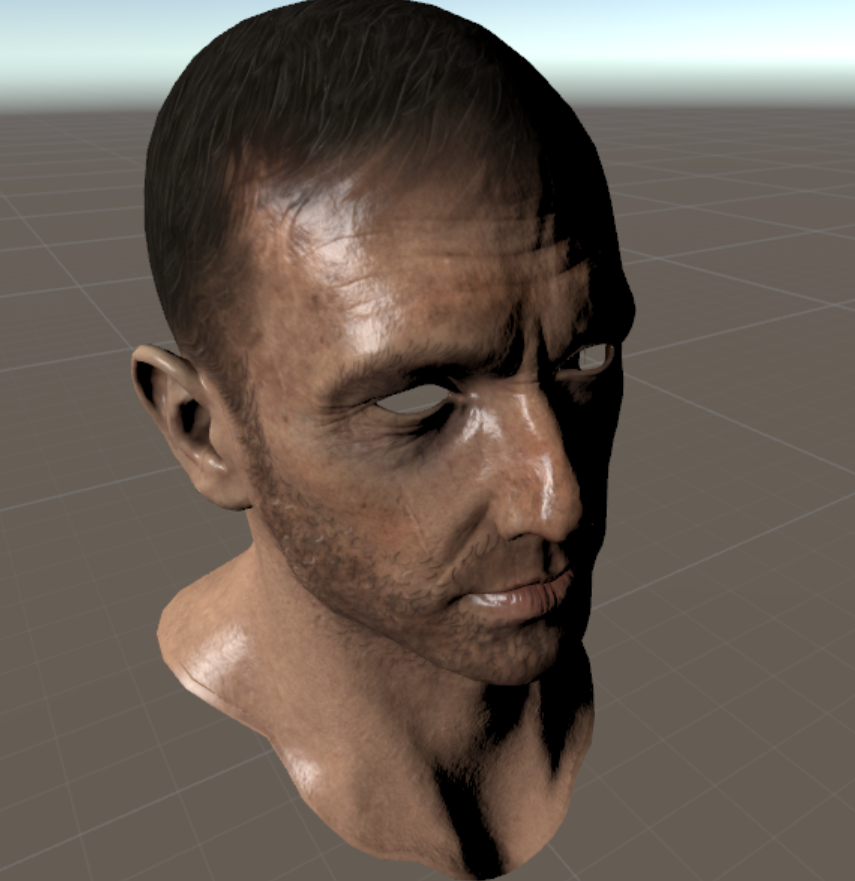
*ENDCG*

*}*

*FallBack "Diffuse"*

*}*

* 결과



# 2pass 외곽선

메시를 두 번 렌더링하여 외곽선 출력.

* 원리
  + 여러 패스 렌더링 방법 : CGPROGRAM ~ ENDCG 구문을 여러 번 반복.
  + 첫 번째 메시 렌더링 : 외곽선
    - 면을 뒤집는다 : cull front
    - 버텍스 셰이더 이용 정점을 입력 노말 방향으로 적절히 위치 이동.
    - 외곽선 색상 출력 : 커스텀 셰이더 이용 색상 리턴.
  + 두 번째 메시 렌더링 : 원본 메시 출력.
    - 면을 원래대로 출력 : cull back.
    - 원래 메시의 디퓨즈 그대로 출력.
* 코드

*Shader "Custom/2pass"*

*{*

*Properties*

*{*

*\_MainTex ("Albedo (RGB)", 2D) = "white" {}*

*}*

*SubShader*

*{*

*Tags { "RenderType"="Opaque" }*

*LOD 200*

***// 1st pass.***

***cull front // 면을 뒤집어 출력. 외곽선 효과 위해.***

***CGPROGRAM***

***// vertex:vert <= vert 함수명의 버텍스 셰이더.***

*#pragma surface surf NoLight* ***vertex:vert*** *noshadow noambient*

*#pragma target 3.0*

*sampler2D \_MainTex;*

***// 정점을 노말 방향으로 적절히 이동하여 크게 스케일링 한다.***

*void vert(inout appdata\_full v)*

*{*

*v.vertex.xyz = v.vertex.xyz + v.normal.xyz \* 0.01;*

*}*

*struct Input*

*{*

*float4 color:COLOR;*

*};*

*void surf (Input IN, inout SurfaceOutput o)*

*{*

***// 픽셀 셰이더는 아무 것도 하지 않음.***

*}*

*float4 LightingNoLight(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*return float4(0, 0, 0, 1);* ***// 검은색으로 외곽선 출력.***

*}*

***ENDCG***

***// 2nd pass. 기존대로 메시 출력.***

***cull back // 원래대로 컬링.***

***CGPROGRAM***

*#pragma surface surf Lambert*

*#pragma target 3.0*

*sampler2D \_MainTex;*

*struct Input*

*{*

*float2 uv\_MainTex;*

*};*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

*o.Albedo = c.rgb;*

*o.Alpha = c.a;*

*}*

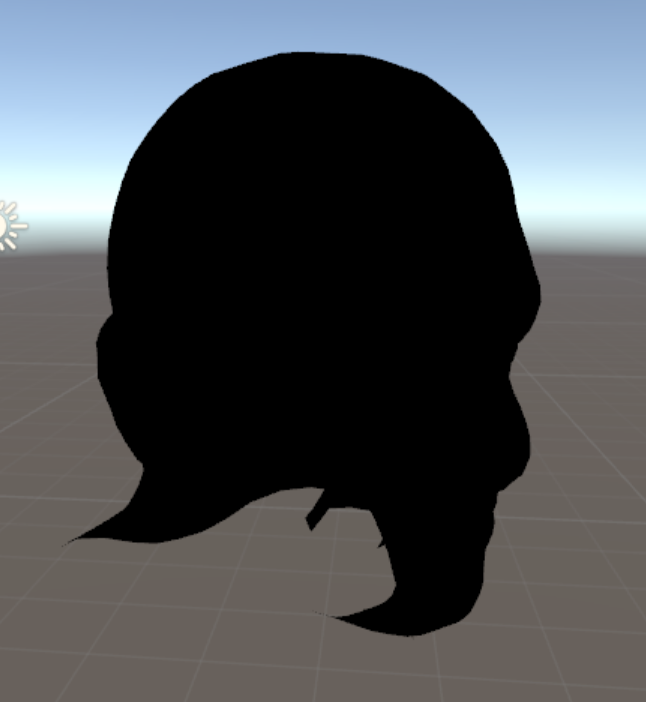
***ENDCG***

*}*

*FallBack "Diffuse"*

*}*

* 출력
  + 첫 번째 패스만 렌더링



* + 두 번째 패스만 렌더링



* + 최종 출력 : 두 개 패스 렌더링.



# 카툰 렌더링

만화에서 볼 수 있는 끊어지는 음영을 구현.

* 원리
  + 노말벡터와 라이트벡터의 내적값을 구한다.
  + 내적값을 계단화하여 출력.
* 코드

*float4 LightingToon(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*float NdotL = dot(s.Normal, lightDir) \* 0.5f + 0.5f;*

***// 계단화. ceil 함수를 사용해도 좋다.***

*if (NdotL > 0.7f)*

*{*

*NdotL = 1.0f;*

*}*

*else*

*{*

*NdotL = 0.3f;*

*}*

*float4 final;*

*final.rgb = s.Albedo \* NdotL \* \_LightColor0.rgb;*

*final.a = s.Alpha;*

*return final;*

*}*

* 출력

# Fresnel 외곽선

2pass 외곽선 구현은 성능상에 단점이 있다. 1pass로 외곽선을 구현해본다.

* 원리
  + Fresnel 연산을 한다.
  + 이 연산값을 계단화하여 음영을 만든다.
* 코드

*float4 LightingToon(SurfaceOutput s, float3 lightDir, float3 viewDir, float atten)*

*{*

*float NdotL = dot(s.Normal, lightDir) \* 0.5f + 0.5f;*

*if (NdotL > 0.7f)*

*{*

*NdotL = 1.0f;*

*}*

*else*

*{*

*NdotL = 0.3f;*

*}*

***// Fresnel 연산 결과값을 계단화하여 음영 처리.***

*float rim = abs(dot(s.Normal, viewDir));*

*if (rim > 0.3f)*

*{*

*rim = 1.0f;*

*}*

*else*

*{*

*rim = -1.0f;*

*}*

*float4 final;*

*final.rgb = s.Albedo \* NdotL \* \_LightColor0.rgb;*

*final.a = s.Alpha;*

*return rim;*

*}*

* 출력

# diffuse warping

Illustrative Rendering in Team Fortress 2- Valve 에서 발표한 팀 포트리스 2 의 기법. 빛 계산으로 쓰는 공식인 노말과 라이트 벡터의 내적을 텍스처 UV 로 활용하여 해당 텍스처의 색상을 출력. 예를 들어 특정 색상의 그라데이션 출력을 원한다면 빛을 받는 강도에 따라 그라데이션 텍스처의 UV 에 해당하는 색상을 출력하면 됨. 응용에 따라 실사, 비실사 렌더링 모두 가능.

* 원리
  + 노말과 라이트 벡터의 내적을 구함(-1 ~ 1).
  + 하프 램버트 이용 내적값을 0 ~ 1 로 적용.
  + 텍스처 준비(RampTex).
  + RampTex 의 UV 를 내적값으로 이용 diffuse 적용.
* 코드

*shader “Custom/warp”*

*{*

*Properties*

*{*

*\_MainTex(“Albedo (RGB)”, 2D) = “white” {}*

*\_BumpMap(“Normal (RGB), 2D”) = “bump” {}*

*\_RampTex(“RampTex”, 2D) = “white” {}*

*}*

*SubShader*

*{*

*Tags {“RenderType”=”Opaque”}*

*CGPROGRAM*

*#pragma surface surf warp noambient*

*sampler2D \_MainTex*

*sampler2D \_BumpMap*

*sampler2D \_RampTex* ***// warp 텍스처 준비.***

*struct Input*

*{*

*float2 uv\_MainTex;*

*float2 uv\_BumpMap;*

***// warp 텍스처의 UV 는 직접 계산하여 사용할 것이기 때문에 여기서는***

***// 필요없음.***

*}*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

*o.Normal = UnpackNormal(tex2D(\_BumpMap, IN.uv\_BumpMap));*

*o.Albedo = c.rgb;*

*o.Alpha = c.a;*

*}*

*float4 Lightingwarp(SurfaceOutput s, float3 lightDir, float atten)*

*{*

*float NdotL = dot(s.Normal, lightDir) \* 0.5f + 0.5f;*

***// 내적값(0~1) 을 warp 텍스처 UV 의 U 로 처리하여 diffuse 값 뽑아냄.***

*float4 ramp = tex2D(\_RampTex, float2(NdotL, 0.5f));*

*float4 final;*

*final.rgb = s.Albedo.rgb \* ramp.rgb;*

*final.a = s.Alpha;*

*return final;*

*}*

*}*

*}*

* 출력

# 리플렉션(큐브맵)

레이트레이싱 등의 기술을 사용하여 반사를 표현하면 매우 현실적인 반사를 구현할 수 있지만 연산이 매우 무거워 실제로 적용하기에는 무리가 있다. 가벼운 연산을 위해 트릭을 사용하는데 큐브맵에 diffuse 를 넣고 반사 영역을 계산하여 출력한다.

* 원리
  + 큐브맵을 준비한다.
  + 입력 구조체에 반사 벡터 준비.
  + texCUBE 함수로 diffuse 를 구한다.
  + diffuse 값을 Emission 에 대입.

. Emission 은 빛의 영향을 받지 않는다. 반사는 밝거나 어두워지는 것이 아닌 단순히 반사하는 것일 뿐이므로 Emission 에 대입해야 한다.

* 코드

*shader “Custom/Reflection”*

*{*

*Properties*

*{*

*\_MainTex(“Albedo (RGB)”, 2D) = “white” {}*

*\_Cube(“Cubemap”,* ***Cube****) =* ***“”*** *{}*

*}*

*SubShader*

*{*

*Tags {“RenderType”=”Opaque”}*

*CGPROGRAM*

*#pragma surface surf noambient*

*sampler2D \_MainTex;*

***samplerCUBE \_Cube; // 큐브맵 준비.***

*struct Input*

*{*

*float2 uv\_MainTex;*

***float3 worldRefl; // 반사 벡터.***

*}*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

***float4 re = texCUBE(\_Cube, IN.worldRefl);***

*o.Albedo = 0;*

***// 반사는 빛의 영향을 받아 밝거나 어두워지는 것이 아닌 단순히 반사***

***// 하는 것일 뿐이므로 Albedo 에 적용하지 않고 빛의 영향을 받지 않는***

***// Emission 에 넣어야 한다.***

***o.Emission = re.rgb;***

*o.Alpha = c.a;*

*}*

*ENDCG*

*}*

*FallBack “Diffuse”*

*}*

* 출력



# 리플렉션(큐브맵+노말맵)

반사효과와 동시에 노말맵 효과도 출력한다.

* 원리
  + 노말맵 적용(o.Normal = …).
  + 큐브맵의 UV 구하기.
    - WorldReflectionVector() 함수 사용하여 uv 뽑기.
    - o.Normal 은 탄젠트 공간 벡터임. 월드좌표계의 픽셀 노말로 뽑기 위해서 입력 구조체의 반사벡터(float3 worldRefl) 에 INTERNAL\_DATA 키워드 적용.
* 코드

*shader “Custom/Reflection”*

*{*

*Properties*

*{*

*\_MainTex(“Albedo (RGB)”, 2D) = “white” {}*

***\_BumpMap(“NormalMap”, 2D) = “bump” {}***

***\_Cube(“Cubemap”, Cube) = “” {}***

*}*

*SubShader*

*{*

*Tags {“RenderType”=”Opaque”}*

*CGPROGRAM*

*#pragma surface surf noambient*

*sampler2D \_MainTex;*

***sampler2D \_BumpMap; // 노말맵 준비.***

***samplerCUBE \_Cube; // 큐브맵 준비.***

*struct Input*

*{*

*float2 uv\_MainTex;*

***float2 uv\_BumpMap; // 노말맵 uv***

***// 반사벡터를 INTERNAL\_DATA 키워드를 사용하여 월드 좌표계의***

***// 픽셀 노말로 변형해서 사용.***

***float3 worldRefl;***

***INTERNAL\_DATA***

*}*

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*o.Normal = UnpackNormal(tex2D(\_BumpMap, IN.uv\_BumpMap));*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

***// NormalMap이 적용된 월드좌표계의 픽셀 노말을 뽑아서***

***// 큐브맵의 UV 로 사용.***

*float4 re =* ***texCUBE(\_Cube, WorldReflectionVector(IN. o.Normal));***

*o.Albedo = c.rgb \* 0.5f;*

***o.Emission = re.rgb \* 0.5f;***

*o.Alpha = c.a;;*

*}*

*ENDCG*

*}*

*FallBack “Diffuse”*

*}*

* 출력



# 리플렉션(큐브맵+노말맵+마스크맵)

* 원리
  + 마스크맵 적용.
    - 마스크맵은 r 채널에 흑백 강도로 저장되어 있음.
  + 마스크맵 r 채널을 이용하여 diffuse 와 반사의 강도를 조절함.
* 코드

*void surf(Input IN, inout SurfaceOutput o)*

*{*

*o.Normal = UnpackNormal(tex2D(\_BumpMap, IN.uv\_BumpMap));*

*fixed4 c = tex2D(\_MainTex, IN.uv\_MainTex);*

*float4 re = texCUBE(\_Cube, WorldReflectionVector(IN, o.Normal));*

***float4 m = tex2D(\_MaskMap, IN.uv\_MaskMap);***

*o.Albedo =* ***c.rgb \* (1-m.r); // 마스크맵으로 diffuse 강도 조절.***

*o.Emission =* ***re.rgb \* m.r; // 마스크맵으로 Emission 강도 조절.***

*o.Alpha = c.a;*

*}*