셰이더 기본

빨강 셰이더

* 정점의 위치 변환.
* 입력된 로컬 정점에 월드~투영행렬을 적용한다.

// 입력데이터, 하나의 정점 정보를 담는 구조체. 여기서는 위치값 하나.

struct VS\_INPUT

{

float4 mPosition : POSITION; // 정점데이터에서 위치 정보를 가져와 mPosition에 대입하라.

};

// 출력데이터, 픽셀의 위치를 찾기 위해. 위치 변환 결과를 래스터라이저에 전달.

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

};

float4x4 gWorldMatrix; // 월드행렬

float4x4 gViewMatrix; // 뷰행렬

float4x4 gProjMatrix; // 투영행렬

VS\_OUTPUT vs\_main(VS\_INPUT input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(input.mPosition, gWorldMatrix); // 로컬\* 월드

Output.mPosition = mul(Output.mPosition, gViewMatrix); // 로컬\* 월드\* 뷰

Output.mPosition = mul(Output.mPosition, gProjMatrix); // 로컬\* 월드\* 뷰\* 투영

return Output; // 픽셀셰이더에 전달. 정확히는 래스터라이저.

}

텍스처 매핑

* 정점의 UV 좌표를 픽셀셰이더에 전달.
* 픽셀에 대응하는 텍스처의 텍셀을 구한다.

struct VS\_INPUT

{

float4 mPosition : POSITION;

float2 mTexCoord : TEXCOORD0; // uv 좌표값.

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float2 mTexCoord : TEXCOORD0;

};

float4x4 gWorldMatrix;

float4x4 gViewMatrix;

float4x4 gProjMatrix;

// 정점위치와 uv 좌표매핑.

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(Input.mPosition, gWorldMatrix);

Output.mPosition = mul(Output.mPosition, gViewMatrix);

Output.mPosition = mul(Output.mPosition, gProjMatrix);

Output.mTexCoord = Input.mTexCoord; // 정점에 입력된uv 좌표매핑.

return Output;

}

sampler2D DiffuseSampler; // 2D 텍스처 개체. 보통 텍스처샘플러 라고 함. sampler1D, sampler3D, samplerCUBE 등이있음.

struct PS\_INPUT

{

float2 mTexCoord : TEXCOORD0;

};

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float albedo = tex2D(DiffuseSampler, Input.mTexCoord); // 텍셀을 구함.

return albedo.rgba; // 텍셀 반환.

// 벡터swizzle.

// albedo.bgra 또는 albedo.rrra 와 같이 순서를 바꿔가며 벡터의 성분에 접근할 수 있다.

}

난반사광(Diffuse Light)

* 광원의 위치에 따른 정점별 밝기 정도를 표현.
* 밝기 = 절대값(dot(광원벡터, 정점노말))

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float4 mDiffuse : TEXCOORD1;

};

float4x4 gWorldMatrix;

float4x4 gViewMatrix;

float4x4 gProjMatrix;

float4 gWorldLightPosition;

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(Input.mPosition, gWorldMatrix);

float3 lightDir = Output.mPosition.xyz - gWorldLightPosition.xyz; // 벡터(광원-> 정점)

lightDir = normalize(lightDir);

Output.mPosition = mul(Output.mPosition, gViewMatrix);

Output.mPosition = mul(Output.mPosition, gProjMatrix);

float3 worldNormal = mul(Input.mNormal, (float3x3)gWorldMatrix); // 정점노말. 월드변환(lightDir 이 월드 기준이기 때문에).

worldNormal = normalize(worldNormal);

Output.mDiffuse = dot(-lightDir, worldNormal); // 내적은 cos값(-1 ~ 1). 이 값이 밝기의 정도를 나타낸다.

return Output;

}

struct PS\_INPUT

{

float3 mDiffuse : TEXCOORD1;

};

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float3 diffuse = saturate(Input.mDiffuse); // 절대값으로 변환. 음수값은 0으로 처리한다(검은색).

return float4(diffuse, 1);

}

정반사광(Specular Light)

* 하이라이트 표현.
* 난반사광에 밝기를 더한다.
* 하이라이트 정점 값 = 절대값(dot(반사광 벡터, 시점벡터))의 제곱승수(약 20).
* 제곱승수를 많이 하면 더 밝아진다.

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float3 mDiffuse : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 mReflection : TEXCOORD3;

};

float4x4 gWorldMatrix;

float4x4 gViewMatrix;

float4x4 gProjectionMatrix;

float4 gWorldLightPosition;

float4 gWorldCameraPosition;

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(Input.mPosition, gWorldMatrix);

// 입사광.

float3 lightDir = Output.mPosition.xyz - gWorldLightPosition.xyz;

lightDir = normalize(lightDir);

Output.mPosition = mul(Output.mPosition, gViewMatrix);

Output.mPosition = mul(Output.mPosition, gProjectionMatrix);

// 정점 노말.

float3 worldNormal = mul(Input.mNormal, (float3x3)gWorldMatrix);

worldNormal = normalize(worldNormal);

Output.mDiffuse = dot(-lightDir, worldNormal);

// 반사광. 픽셀셰이더에서 이 반사광과 시점의 내적값으로 밝기의 정도를 더 표현한다.

Output.mReflection = reflection(lightDir, worldNormal);

return Output;

}

struct PS\_INPUT

{

float3 mDiffuse : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 mReflection : TEXCOORD3;

};

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float3 diffuse = saturate(Input.mDiffuse); // diffuse.

float3 reflection = normalize(Input.mReflection);

float3 viewDir = normalize(Input.mViewDir);

float3 specular = 0;

if (diffuse.x > 0)

{

// 반사광과 시점의 내적. 제곱승수. 제곱승수가 클수록 표현의 강도가 쎄진다.

specular = saturate(dot(reflection, -viewDir));

specular = pow(specular, 20.0f);

}

// 기본 환경광.

float3 ambient = float3(0.1f, 0.1f, 0.1f);

return float4(ambient + diffuse + specular, 1);

}

디퓨즈/스페큘라 매핑

* 난반사 및 정반사광에 텍스처를 입힌다.
* 디퓨즈맵으로 색을 입힌다.
* 스페큘라맵으로 빛이 반사하는 정도를 계산한다(좀 더 매끄러운 표현을 위해).

float4x4 gWorldMatrix;

float4x4 gViewMatrix;

float4x4 gProjectionMatrix;

float4 gWorldLightPosition;

float4 gWorldCameraPosition;

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

float2 mUV : TEXCOORD0;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float2 mUV : TEXCOORD0;

float3 mDiffuse : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 mReflection : TEXCOORD3;

};

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(Input.mPosition, gWorldMatrix);

// 입사광

float3 lightDir = Output.mPosition.xyz - gWorldLightPosition.xyz;

lightDir = normalize(lightDir);

float3 viewDir = normalize(Output.mPosition.xyz - gWorldCameraPosition.xyz);

Output.mViewDir = viewDir;

Output.mPosition = mul(Output.mPosition, gViewMatrix);

Output.mPosition = mul(Output.mPosition, gProjectionMatrix);

float3 worldNormal = mul(Input.mNormal, (float3x3)gWorldMatrix);

worldNormal = normalize(worldNormal);

// 난반사값.

Output.mDiffuse = dot(-lightDir, worldNormal);

// 반사광

Output.mReflection = reflect(lightDir, worldNormal);

// UV값. 텍스처의 컬러(텍셀값)를 얻기 위해.

Output.mUV = Input.mUV;

return Output;

}

struct PS\_INPUT

{

float2 mUV : TEXCOORD0;

float3 mDiffuse : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 mReflection : TEXCOORD3;

};

sampler2D DiffuseSampler; // 디퓨즈 텍스처 개체.

sampler2D SpecularSampler; // 스페큘라 텍스처 개체.

float3 gLightColor;

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float4 albedo = tex2D(DiffuseSampler, Input.mUV); // 현재 픽셀이 반사하는 색깔.

// diffuse = 빛의색상 \* 픽셀이 반사하는 색깔 \* 난반사광의양.

float3 diffuse = gLightColor \* albedo.rgb \* saturate(Input.mDiffuse);

float3 reflection = normalize(Input.mReflection);

float3 viewDir = normalize(Input.mViewDir);

float3 specular = 0;

if (diffuse.x > 0)

{

// 정반사광을 구한다.

specular = saturate(dot(reflection, -viewDir));

specular = pow(specular, 20.0f);

float4 specularIntensity = tex2D(SpecularSampler, Input.mUV); // 정반사광의 세기는 스페큘러 텍스처의 텍셀값으로 한다.

specular \*= specularIntensity.rgb \* gLightColor; // 최종 정반사광의 값.

}

float3 ambient = float3(0.1f, 0.1f, 0.1f) \* albedo;

return float4(ambient + diffuse + specular, 1); // 최종색 = 환경광 + 난반사광 + 정반사광.

}

툰 셰이더

* 디퓨즈 값을 계단식으로 표현하면 만화 같은 표현이 가능하다.
* 0 ~ 1 사이의 디퓨즈 값을 0, 0.2, 0.4, 0.6, 0.8, 1 로 강제 변환.
* ceil() 함수를 이용한다.

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float3 mDiffuse : TEXCOORD1;

};

float4x4 gWorldViewProjMatrix; // 월드 ~ 투영행렬. 정점위치 계산 시 한 번만 곱하도록.

float4x4 gInvWorldMatrix;

float4 gWorldLightPosition;

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(Input.mPosition, gWorldViewProjMatrix);

float3 objectLightPosition = mul(gWorldLightPosition, gInvWorldMatrix);

float3 lightDir = normalize(Input.mPosition.xyz - objectLightPosition);

Output.mDiffuse = dot(-lightDir, normalize(Input.mNormal));

return Output;

}

float3 gSurfaceColor;

struct PS\_INPUT

{

float3 mDiffuse : TEXCOORD1;

};

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float3 diffuse = saturate(Input.mDiffuse);

// 위의 diffuse가 0 ~ 1 사이의 값을 가진다.

// 5를 곱하면 0 ~ 5

// ceil() 적용하면 값은 0, 1, 2, 3, 4, 5 중하나.

// 5로 나누면 0, 0.2, 0.4, 0.6, 0.8, 1

diffuse = ceil(diffuse \* 5) / 5.0f;

return float4(gSurfaceColor \* diffuse.xyz, 1); // 배경값을적용하여리턴.

}

법선매핑

* 물체의 질감을 사실적으로 표현하는 트릭.
* 디퓨즈/스페큘라 매핑과의 유일한 차이점은 노말에 있음.
* 정점 노말을 법선맵(텍스처)의 노말로 대체하여 사용한다.
* 법선맵의 노말은 기준좌표가 없기 때문에 접선공간(정점공간)을 이용.
* 접선공간 = Z(정점노말), X(UV 좌표의 U), Y(Z와X의 외적)
* 노말 = 접선공간 행렬 \* 법선맵 노말

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

float3 mTangent : TANGENT;

float3 mBinormal : BINORMAL;

float2 mUV : TEXCOORD0;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float2 mUV : TEXCOORD0;

float3 mLightDir : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 T : TEXCOORD3;

float3 B : TEXCOORD4;

float3 N : TEXCOORD5;

};

float4x4 gWorldMatrix;

float4x4 gWorldViewProjMatrix;

float4 gWorldLightPosition;

float4 gWorldCameraPosition;

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(Input.mPosition, gWorldViewProjMatrix);

Output.mUV = Input.mUV;

float4 worldPosition = mul(Input.mPosition, gWorldMatrix);

float3 lightDir = worldPosition.xyz - gWorldLightPosition;

Output.mLightDir = normalize(lightDir);

float3 viewDir = normalize(worldPosition.xyz - gWorldCameraPosition);

Output.mViewDir = viewDir;

// 접선공간 벡터들.

float3 worldNormal = mul(Input.mNormal, (float3x3)gWorldMatrix);

Output.N = normalize(worldNormal);

float3 worldTangent = mul(Input.mTangent, (float3x3)gWorldMatrix);

Output.T = normalize(worldTangent);

float3 worldBinormal = mul(Input.mBinormal, (float3x3)gWorldMatrix);

Output.B = normalize(worldBinormal);

// 이전과 다르게 이곳에서 diffuse를 구하지 않는다.

// 이유는 diffuse를 구하기 위해서는 정점노말과 입사광이 필요한데

// 여기서는 굴곡질감을 표현하기 위해 정점노말을 바로 사용하지 않고

// 법선맵을 이용하여 접선공간의 노말을 사용하기 때문이다.

// 이것은 픽셀셰이더에서 구현한다.

return Output;

}

struct PS\_INPUT

{

float2 mUV : TEXCOORD0;

float3 mLightDir : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 T : TEXCOORD3;

float3 B : TEXCOORD4;

float3 N : TEXCOORD5;

};

sampler2D DiffuseSampler;

sampler2D SpecularSampler;

sampler2D NormalSampler;

float gLightColor;

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float3 tangentNormal = tex2D(NormalSampler, Input.mUV).xyz;

// 접선노말값 복원.

// 텍스처에서 저장하는 픽셀 하나의 색상은 rgba 이다.

// 각각의값은 0 ~ 1 사이의 값으로 저장할 수 밖에 없기 때문에

// 저장시 -1 ~ 1 의 값을 다음의 공식을 이용해서 저장했다.

// - 법선맵RGB = 법선백터XYZ \* 0.5 + 0.5

// - 법선벡터XYZ = 법선맵RGB \* 2 -1

tangentNormal = normalize(tangentNormal \* 2 - 1);

float3x3 TBN = float3x3(normalize(Input.T), normalize(Input.B), normalize(Input.N));

TBN = transpose(TBN); // 열기준 행렬으로 변환. 직교행렬의 전치행렬은 역행렬이다. 벡터= 열기준행렬\* 벡터, 벡터= 벡터\* 행기준행렬

float3 worldNormal = mul(TBN, tangentNormal); // 이제 이 값을 노말로 사용.

float4 albedo = tex2D(DiffuseSampler, Input.mUV); // 색상.

float3 lightDir = normalize(Input.mLightDir);

float3 diffuse = saturate(dot(worldNormal, -lightDir)); // diffuse 값.

diffuse = gLightColor \* albedo.rgb \* diffuse; // 최종diffuse 값.

// 정반사광(스페큘라)을 더하여 하이라이트 효과를 넣는다.

float3 specular = 0;

if (diffuse.x > 0)

{

float3 reflection = reflect(lightDir, worldNormal);

float3 viewDir = normalize(Input.mViewDir);

specular = saturate(dot(reflection, -viewDir));

specular = pow(specular, 20.0f);

float4 speculatIntensity = tex2D(SpecularSampler, Input.mUV);

specular \*= specularIntensity.rgb \* gLightColor;

}

float ambient = float3(0.1f, 0.1f, 0.1f) \* albedo;

return float4(ambient + diffuse + specular, 1);

}

환경매핑

* 물체 표면에 주변 환경을 반사하는 표현.
* 반사 대상인 물체의 입방체 텍스처를 준비한다(보통은 스카이박스 텍스처).
* 반사 벡터를 이용하여 입방체 텍스처의 텍셀을 구한다(texCUBE() 함수 이용).
* 텍셀값을 더한다. 끝.

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

float3 mTangent : TANGENT;

float3 mBinormal : BINORMAL;

float2 mUV : TEXCOORD0;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float2 mUV : TEXCOORD0;

float3 mLightDir : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 T : TEXCOORD3;

float3 B : TEXCOORD4;

float3 N : TEXCOORD5;

};

float4x4 gWorldMatrix;

float4x4 gWorldViewProjMatrix;

float4 gWorldLightPosition;

float4 gWorldCameraPosition;

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

Output.mPosition = mul(Input.mPosition, gWorldViewProjMatrix);

Output.mUV = Input.mUV;

float4 worldPosition = mul(Input.mPosition, gWorldMatrix);

float3 lightDir = worldPosition.xyz - gWorldLightPosition;

Output.mLightDir = normalize(lightDir);

float3 viewDir = normalize(worldPosition.xyz - gWorldCameraPosition);

Output.mViewDir = viewDir;

float3 worldNormal = mul(Input.mNormal, (float3x3)gWorldMatrix);

Output.N = normalize(worldNormal);

float3 worldTangent = mul(Input.mTangent, (float3x3)gWorldMatrix);

Output.T = normalize(worldTangent);

float3 worldBinormal = mul(Input.mBinormal, (float3x3)gWorldMatrix);

Output.B = normalize(worldBinormal);

return Output;

}

struct PS\_INPUT

{

float2 mUV : TEXCOORD0;

float3 mLightDir : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 T : TEXCOORD3;

float3 B : TEXCOORD4;

float3 N : TEXCOORD5;

};

sampler2D DiffuseSampler;

sampler2D SpecularSampler;

sampler2D NormalSampler;

samplerCUBE EnvironmentSampler; // 입방체 텍스처.

float gLightColor;

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float3 tangentNormal = tex2D(NormalSampler, Input.mUV).xyz;

tangentNormal = normalize(tangentNormal \* 2 - 1);

float3x3 TBN = float3x3(normalize(Input.T), normalize(Input.B), normalize(Input.N));

TBN = transpose(TBN);

float3 worldNormal = mul(TBN, tangentNormal);

float4 albedo = tex2D(DiffuseSampler, Input.mUV);

float3 lightDir = normalize(Input.mLightDir);

float3 diffuse = saturate(dot(worldNormal, -lightDir));

diffuse = gLightColor \* albedo.rgb \* diffuse;

float3 specular = 0;

if (diffuse.x > 0)

{

float3 reflection = reflect(lightDir, worldNormal);

float3 viewDir = normalize(Input.mViewDir);

specular = saturate(dot(reflection, -viewDir));

specular = pow(specular, 20.0f);

float4 speculatIntensity = tex2D(SpecularSampler, Input.mUV);

specular \*= specularIntensity.rgb \* gLightColor;

}

float3 viewReflect = reflect(viewDir, worldNormal);

// 추가할건 이거 한 가지 뿐이다.

// 반사벡터를 이용하여 입방체 텍스처의 텍셀값을 구하여 더하면 된다.

float3 environment = texCUBE(EnvironmentSampler, viewReflect).rgb;

float ambient = float3(0.1f, 0.1f, 0.1f) \* albedo;

return float4(ambient + diffuse + specular + environment \* 0.5f, 1);

}

UV 애니메이션

* 물이 흐르는 듯한 사실감있는 표현.
* 텍스처를 스크롤링한다.
* 정점 위치를 임의로 변경한다(cos() 함수 이용).

float4x4 gWorldMatrix;

float4x4 gViewMatrix;

float4x4 gProjMatrix;

float4 gWorldLightPosition;

float4 gWorldCameraPosition;

float gTime;

float gWaveHeight;

float gSpeed;

float gWaveFrequency;

float gUVSpeed;

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

float2 mUV : TEXCOORD0;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float2 mUV : TEXCOORD0;

float3 mDiffuse : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 mReflection : TEXCOORD3;

};

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

// 정점이 위 아래로 울렁이는 애니메이션을 표현.

// cos() 함수는 -1 ~ 1 범위에 있기 때문에 적절한 요소를 줘서 활용하면 정점이 범위 내에서 움직이게 할 수 있음.

// 사실적인 표현을 위해서는 정점별로 울렁이는 정도가 달라야 한다.

// 모든 정점이 동일한 정도로 변화한다면 하나의 물체가 위치가 이동하는 정도로밖에 보이지 않을것임.

// UV값은 정점별로 다르기 때문에 정점 위치 변화에 대한 요소로 활용할 수 있음.

// 여기에 속도에 대한 요소 정도만 적절하게 넣어주면 좀 더 사실적인 표현이 가능함.

float cosTime = gWaveHeight \* cos(gTime \* gSpeed + Input.mUV.x \* gWaveFrequency);

Input.mPosition.y += cosTime;

Output.mPosition = mul(Input.mPosition, gWorldMatrix);

float3 lightDir = Output.mPosition.xyz - gWorldLightPosition.xyz;

lightDir = normalize(lightDir);

float3 viewDir = normalize(Output.mPosition.xyz - gWorldCameraPosition.xyz);

Output.mViewDir = viewDir;

Output.mPosition = mul(Output.mPosition, gViewMatrix);

Output.mPosition = mul(Output.mPosition, gProjMatrix);

float3 worldNormal = mul(Input.mNormal, (float3x3)gWorldMatrix);

worldNormal = normalize(worldNormal);

Output.mDiffuse = dot(-lightDir, worldNormal);

Output.mReflection = reflection(lightDir, worldNormal);

// 텍스처 스크롤링 표현. U값만 스크롤링 한다.

Output.mUV = Input.mUV + float2(gTime \* gUVSpeed, 0);

return Output;

}

struct PS\_INPUT

{

float2 mUV : TEXCOORD0;

float3 mDiffuse : TEXCOORD1;

float3 mViewDir : TEXCOORD2;

float3 mReflection : TEXCOORD3;

};

sampler2D DiffuseSampler;

sampler2D SpecularSampler;

float3 gLightColor;

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float4 albedo = tex2D(DiffuseSampler, Input.mUV);

float3 diffuse = gLightColor \* albedo.rgb \* saturate(Input.mDiffuse);

float3 reflection = normalize(Input.mReflection);

float3 viewDir = normalize(Input.mViewDir);

float3 specular = 0;

if (diffuse.x > 0)

{

specular = saturate(dot(reflection, -viewDir));

specular = pow(specular, 20.0f);

float4 speculatIntensity = tex2D(SpecularSampler, Input.mUV);

specular \*= specularIntensity.rgb \* gLightColor;

}

float3 ambient = float3(0.1f, 0.1f, 0.1f) \* albedo;

return float4(ambient + diffuse + specular, 1);

}

그림자매핑

1. 그림자 생성단계(셰도우맵) – pass0
   1. 렌더링 결과(빛을 가로막는 첫 번째 물체의 깊이)를 저장할 렌더타깃(후면버퍼가 아닌 텍스처)을 정해 준다.
   2. 카메라를 광원의 위치에 두고 물체들을 그린다.
   3. 픽셀셰이더에서 빛으로부터 현재 픽셀까지의 깊이를 반환한다.

struct VS\_INPUT

{

float4 mPosition : POSITION;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float4 mClipPosition : TEXCOORD1;

};

float4x4 gWorldMatrix; // 월드

float4x4 gViewMatrix\_Light; // 뷰 – 라이트

float4x4 gProjMatrix\_Light; // 투영 – 라이트

float4 gWorldLightPosition; // 라이트 위치

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

float4x4 ViewMatrix\_Light = gViewMatrix\_Light;

// 렌더몽키에서 테스트 하기 위한 코드.

// 실제게임에서는 라이트뷰가 전역으로 세팅될 것임.

//float3 dirZ = -normalize(gWorldLightPosition.xyz);

//float3 up = float3(0, 1, 0);

//float3 dirX = cross(up, dirZ);

//float3 dirY = cross(dirZ, dirX);

//ViewMatrix\_Light = float4x4(

// float4(dirX, - dot(gWorldLightPosition.xyz, dirX)),

// float4(dirY, - dot(gWorldLightPosition.xyz, dirY)),

// float4(dirZ, - dot(gWorldLightPosition.xyz, dirZ)),

// float4(0, 0, 0, 1)

// );

// 라이트시점으로위치변환.

Output.mPosition = mul(Input.mPosition, gWorldMatrix);

Output.mPosition = mul(Output.mPosition, ViewMatrix\_Light);

Output.mPosition = mul(Output.mPosition, gProjMatrix\_Light);

Output.mClipPosition = Output.mPosition;

return Output;

}

struct PS\_INPUT

{

float4 mClipPosition : TEXCOORD1;

};

float4 ps\_main(PS\_INPUT Input) : COLOR

{

// 깊이값을 구한다.

// 위치벡터에 원근투영행렬을 곱하면 w성분은 1이 아니게 된다(동차좌표).

// 동차좌표에 있는 값을 화면상의 좌표값으로 사용하려면 w 값을 1로 바꿔줘야 한다(그래야 원근이 표현된다).

float depth = Input.mClipPosition.z / Input.mClipPosition.w;

return float4(depth.xxx, 1);

}

1. 그림자 적용단계 – pass1
   1. 렌더링 결과(일반 장면 렌더링)를 화면(후면버퍼)에 저장한다.
   2. 카메라를 눈의 위치에 두고 물체들을 그린다.
   3. 빛으로부터 현재 픽셀까지의 깊이를 그림자맵에 담겨있는 결과와 비교한다.
   4. 현재 깊이가 그림자맵의 깊이보다 크면(현재 깊이 > 그림자맵 깊이) 그림자를 씌운다.

struct VS\_INPUT

{

float4 mPosition : POSITION;

float3 mNormal : NORMAL;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float4 mClipPosition : TEXCOORD1;

float mDiffuse : TEXCOORD2;

};

float4x4 gWorldMatrix; // 월드행렬

float4x4 gViewMatrix\_Light; // 뷰행렬- 카메라

float4x4 gProjMatrix\_Light; // 투영행렬- 카메라

float4 gWorldLightPosition;

float4x4 gViewProjectionMatrix; // 사용자-시점 뷰행렬

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

float4x4 ViewMatrix\_Light = gViewMatrix\_Light;

// 렌더몽키에서 테스트하기 위한코드.

// 실제 게임에서는 라이트뷰가 전역으로 세팅될 것임.

//float3 dirZ = -normalize(gWorldLightPosition.xyz);

//float3 up = float3(0, 1, 0);

//float3 dirX = cross(up, dirZ);

//float3 dirY = cross(dirZ, dirX);

//ViewMatrix\_Light = float4x4(

// float4(dirX, - dot(gWorldLightPosition.xyz, dirX)),

// float4(dirY, - dot(gWorldLightPosition.xyz, dirY)),

// float4(dirZ, - dot(gWorldLightPosition.xyz, dirZ)),

// float4(0, 0, 0, 1)

// );

// 화면에 그릴 정점 위치 계산- 사용자 시점.

float4 worldPosition = mul(Input.mPosition, gWorldMatrix);

Output.mPosition = mul(worldPosition, gViewProjectionMatrix);

// 그림자맵 사용을 위한 정점 위치 계산- 라이트 시점.

Output.mClipPosition = mul(worldPosition, ViewMatrix\_Light);

Output.mClipPosition = mul(Output.mClipPosition, gProjMatrix\_Light);

// 디퓨즈값.

float3 lightDir = normalize(worldPosition.xyz - gWorldLightPosition.xyz);

float3 worldNormal = normalize(mul(Input.mNormal, (float3x3)gWorldMatrix));

Output.mDiffuse = dot(-lightDir, worldNormal);

return Output;

}

sampler2D ShadowSampler; // 쉐도우맵.

float4 gObjectColor;

struct PS\_INPUT

{

float4 mClipPosition : TEXCOORD1;

float mDiffuse : TEXCOORD2;

};

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float3 rgb = saturate(Input.mDiffuse) \* gObjectColor;

// 현재 깊이값.

float currentDepth = Input.mClipPosition.z / Input.mClipPosition.w;

// 그림자맵의 깊이값.

// 그림자맵의 좌표범위는 UV = (0,0) ~ (1,1)

// 투영공간 좌표계로 생각하면 XY = (-1,1) ~ (1,-1)

// XY -> UV 좌표변환:

// u = x \* 0.5 + 0.5;

// v = -y \* 0.5 + 0.5

float2 uv = Input.mClipPosition.xy / Input.mClipPosition.w; // 이 위치는 투영행렬까지 곱해진 위치임.

uv.y = -uv.y;

uv = uv \* 0.5 + 0.5;

float shadowDepth = tex2D(ShadowSampler, uv).r; // R32F 포맷의 텍스처를 사용했기 때문.

// 현재 깊이가 더 깊으면 그림자를 씌운다.

if (currentDepth > shadowDepth + 0.0000125f)

{

rgb \*= 0.5f; // 현재 색상에서 2배 어둡게.

}

return (float4(rgb, 1.0f));

}

Post Processing(사후처리)

* 화면에 렌더링할 이미지를 2D 이미지로(렌더타깃) 저장.
* 2D 이미지를 픽셀셰이더에 불러와서 특정 효과를 입힌다.
* 흑백 효과, 세피아 효과 등에 유용.

struct VS\_INPUT

{

float4 mPosition : POSITION;

float2 mUV : TEXCOORD0;

};

struct VS\_OUTPUT

{

float4 mPosition : POSITION;

float2 mUV : TEXCOORD0;

};

// 포스트프로세싱.

// 화면에 렌더링할 결과를2D 텍스처(렌더타깃)에 렌더링하고

// 이 렌더타깃을 화면 출력 시 특별한 효과를 처리하기 위해(흑백효과, 세피아효과등)

// 화면크기사각형의(투영공간좌표를가진) 정점4개를 입력받아 픽셀셰이더를 호출한다.

// 픽셀셰이더에서 특정효과를 처리하면 된다.

VS\_OUTPUT vs\_main(VS\_INPUT Input)

{

VS\_OUTPUT Output;

// 입력받은 정점은 이미 투영공간정점이기 때문에( (-1,1) ~ (1,-1) )

// v0 = (-1, 1, 0)

// v1 = (-1, -1, 0)

// v2 = (1, 1, 0)

// v3 = (1, -1, 0)

// 변환없이 바로 픽셀셰이더에 전달한다.

Output.mPosition = Input.mPosition;

Output.mUV = Input.mUV;

return Output;

}

struct PS\_INPUT

{

float2 mUV : TEXCOORD0;

};

sampler2D SceneSampler; // 렌더타깃 텍스처.

float4 ps\_main(PS\_INPUT Input) : COLOR

{

// 여기서는 특별한 처리 없이 렌더타깃의 원본을 렌더링한다.

float4 tex = tex2D(SceneSampler, Input.mUV);

return tex;

}

흑백셰이더(Grayscale)

* 포스트 프로세싱 절차를 따른다.
* 픽셀셰이더에서 컬러값을 흑백으로 변환한다.
* 변환 방법은 아래 코드에서 확인.

struct PS\_INPUT

{

float2 mUV : TEXCOORD0;

};

sampler2D SceneSampler; // 렌더타깃텍스처.

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float4 tex = tex2D(SceneSampler, Input.mUV);

// 컬러이미지를 흑백으로 변환하는방법

// <http://en.wikipedia.org/wiki/Grayscale>

// tex.rgb = tex.r \* 0.3 + tex.g \* 0.59 + tex.b \* 0.11

tex.rgb = dot(tex.rgb, float3(0.3, 0.59, 0.11));

return tex;

}

세피아 셰이더

* 세피아 톤 : 약간 붉그스름하게 갈색 빛이 돌게함.
* 변환 공식은 아래 코드 주석 참조.

struct PS\_INPUT

{

float2 mUV : TEXCOORD0;

};

sampler2D SceneSampler; // 렌더타깃 텍스처.

float4 ps\_main(PS\_INPUT Input) : COLOR

{

float4 tex = tex2D(SceneSampler, Input.mUV);

// 마이크로소프트 권장 변환 공식.

// R = (R \* 0.393 + G \* 0.769 + B \* 0.189)

// G = (R \* 0.349 + G \* 0.686 + B \* 0.168)

// B = (R \* 0.272 + G \* 0.534 + B \* 0.131)

float4 sepia;

sepia.a = tex.a;

sepia.r = dot(tex.rgb, float3(0.393f, 0.769f, 0.189f));

sepia.g = dot(tex.rgb, float3(0.349f, 0.686f, 0.168f));

sepia.b = dot(tex.rgb, float3(0.272f, 0.534f, 0.131f));

return sepia;

}