

Visual Shader Editor

2007.06.23

김성익

noerror@hitel.net

<http://www.gamecode.org>

Spirit of Flame
3D RealTime Graphics Programming Study

Kasa

개요

- 셰이더의 발전과 대중화
- HLSL 코드 작성의 어려움
 - 활용도가 높아짐에 따라 다수의 셰이더 코드를 관리하기가 힘들어짐
 - => 동적인 셰이더 코드 조합식 방식으로 접근

개요

- 아티스트들이 접근하기 어려움
=> 아티스트를 위한 HLSL 도구 등장
- 차세대 엔진의 기본 구성 요소

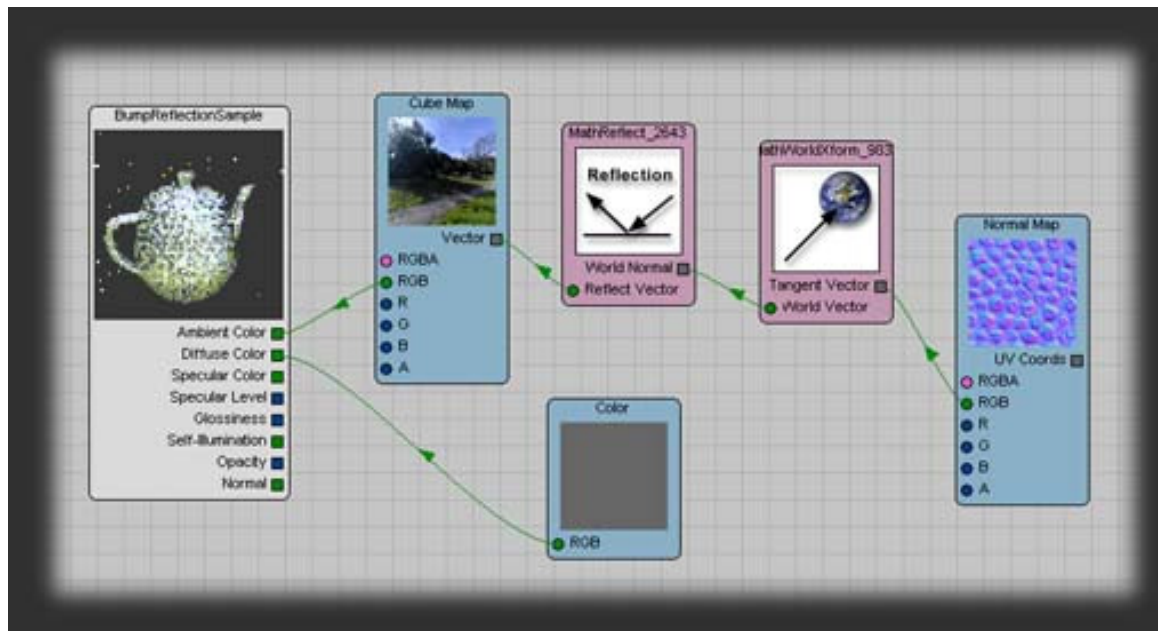
그래프 인터페이스

- 다양한 셰이더 코드를 쉽게 생성
- 아티스트들이 작성 가능
- 직관적이고 편리함

- 도입 사례
 - ShaderFX
 - Unreal3
 - Project Offset

ShaderFX

- Autodesk 3ds Max 플러그인
- 아티스트를 위한 셰이더 에디터



Spirit of Flame
3D RealTime Graphics Programming Study

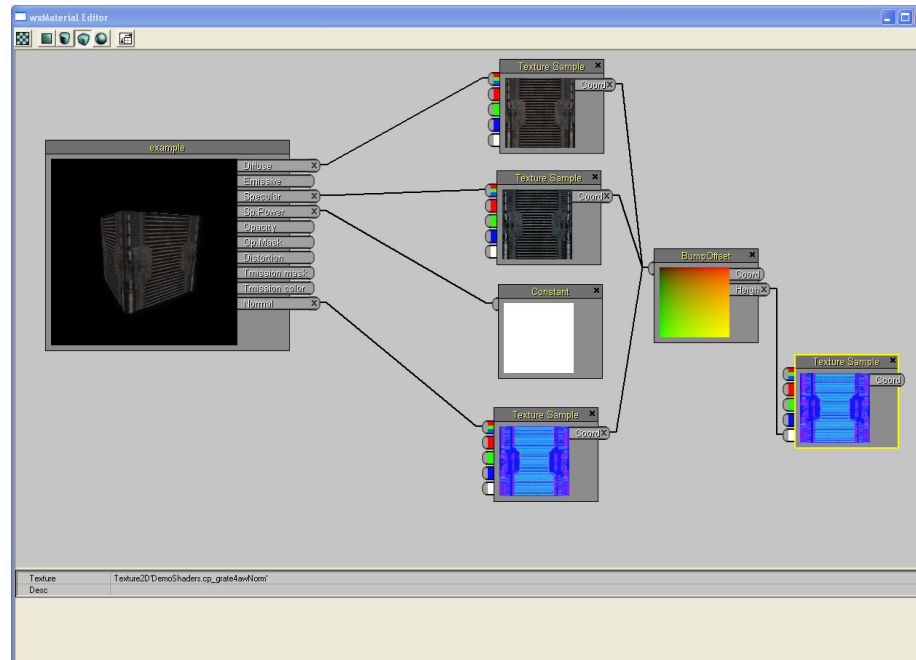
Kasa

ShaderFX

- 실시간 확인 가능
- FX 파일로 export 가능
 - 일반 엔진에서 사용 가능
- 확장 가능
- 저렴
- 명령어 단위의 노드들
- (참고동영상)
- <http://www.lumonix.net/>

Unreal3

- 언리얼3 엔진 전용
- <http://www.unrealtechnology.com/>
- (참고영상)

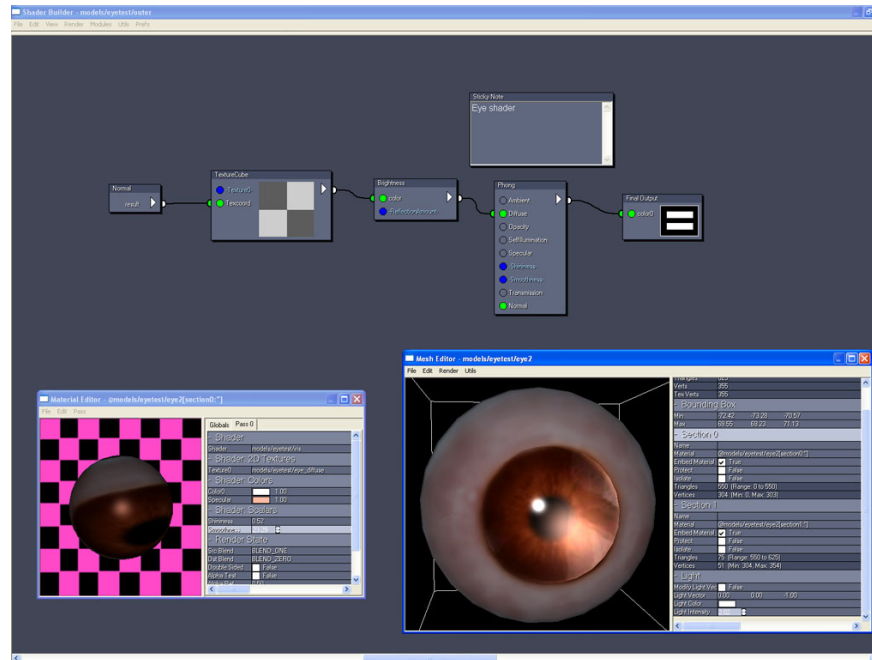


Spirit of Flame
3D RealTime Graphics Programming Study

Kasa

Project Offset

- 엔진 전용
- <http://www.projectoffset.com>
- (참고영상)



Spirit of Flame
3D RealTime Graphics Programming Study

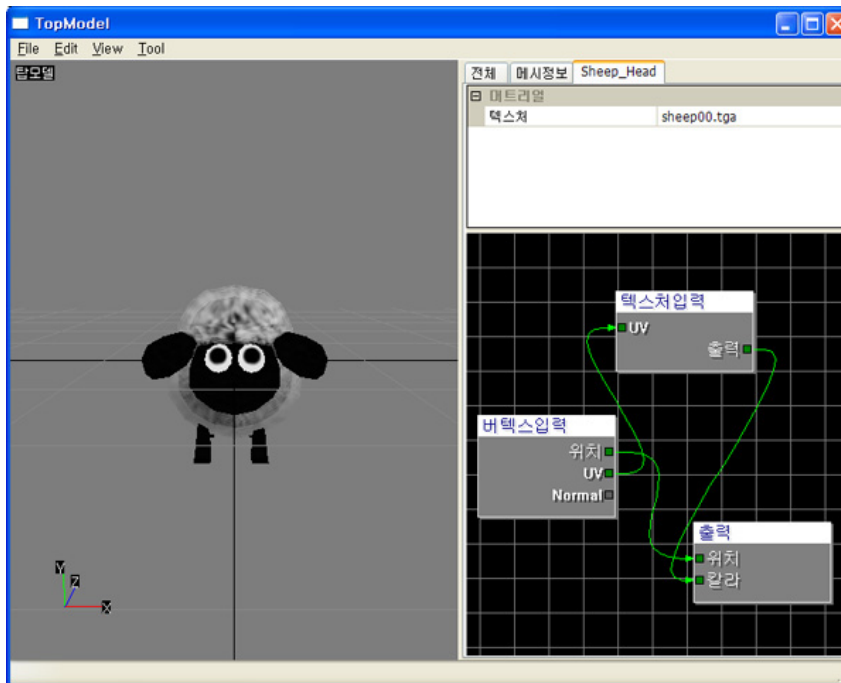
Kasa

셰이더 에디터 구현

- 자체 엔진을 위한 inhouse 툴 사례 소개
- (universal graph편집 사용한 shader edit)
- 사전준비
 - 노드 베이스의 그래프를 렌더링할 툴 킷 작성 (개발 초기 단계임)
 - hlsl shader 코드 적용 가능한 그래픽 엔진 작성 (개발 초기 단계임)
 - xml 기반의 데이터 엔진

셰이더 에디터 구현 이슈

- 기본 이슈
 - HLSL 함수의 구성 분석
 - 그래프화 (노드화)



```

float4x4 wvp : WorldViewProjection;
texture TextureMap_1234;
sampler2D TextureMap_1234Sampler = sampler_state
{
    Texture = <TextureMap_1234>;
};
struct a2v
{
    float4 position      : POSITION;
    float2 texCoord     : TEXCOORD0;
};
struct v2f
{
    float4 position      : POSITION;
    float2 texCoord     : TEXCOORD0;
};
v2f v(a2v In)
{
    v2f Out = (v2f)0;

    Out.position = mul(In.position, wvp);
    Out.texCoord = In.texCoord;
    return Out;
}
float4 p(v2f In) : COLOR
{
    float4 Out;
    Out = tex2D(TextureMap_1234Sampler, In.texCoord.xy);
    return Out;
}
technique Complete
{
    pass
    {
        VertexShader = compile vs_1_1 v();
        PixelShader = compile ps_2_0 p();
    }
}

```

- 베이스가 될만한
기본 셰이더 코드

```

%<shader.header>

%<shader.function>

struct a2v
{
    %<shader.vertexinput>
};

struct v2f
{
    %<shader.pixelinput>;
};

v2f v(a2v In)
{
    v2f Out = (v2f)0;
    %<shader.vertexcode>
    return Out;
}

float4 p(v2f In) : COLOR
{
    float4 Out;
    %<shader.pixelcode>
    return Out;
}

technique Complete
{
    pass sheep
    {
        VertexShader = compile vs_1_1 v();
        PixelShader = compile ps_2_0 p();
    }
}

```

- HLSL 코드는 공통적으로 다음과 6 부분으로 구성됨

float4x4 wvp : WorldViewProjection;

버텍스입력

```
texture TextureMap_1234;

sampler2D TextureMap_1234Sampler = sampler_state
{
    Texture = <TextureMap_1234>;
};

struct a2v
{
    float4 position : POSITION;
    float2 texCoord : TEXCOORD0;
};

struct v2f
{
    float4 position : POSITION;
    float2 texCoord : TEXCOORD0;
};

v2f v(a2v In)
{
    v2f Out = (v2f)0;

    Out.position = mul(In.position, wvp);
    Out.texCoord = In.texCoord;
    return Out;
}

float4 p(v2f In) : COLOR
{
    float4 Out;
    Out = tex2D(TextureMap_1234Sampler, In.texCoord.xy);
    return Out;
}

technique Complete
{
    pass
    {
        VertexShader = compile vs_1_1 v();
        PixelShader = compile ps_2_0 p();
    }
}
```

- 버텍스 입력과 관련 있는 부분

```
float4x4 wvp : WorldViewProjection;
```

```
texture TextureMap_1234;  
sampler2D TextureMap_1234Sampler = sampler_state  
{  
    Texture = <TextureMap_1234>;  
};
```

```
struct a2v  
{  
    float4 position      : POSITION;  
    float2 texCoord      : TEXCOORD0;  
};
```

텍스처 출력

```
struct v2f  
{  
    float4 position      : POSITION;  
    float2 texCoord      : TEXCOORD0;  
};
```

```
v2f v(a2v In)  
{  
    v2f Out = (v2f)0;  
  
    Out.position = mul(In.position, wvp);  
    Out.texCoord = In.texCoord;  
    return Out;  
}
```

```
float4 p(v2f In) : COLOR  
{  
    float4 Out;  
    Out = tex2D(TextureMap_1234Sampler, In.texCoord.xy);  
    return Out;  
}
```

```
technique Complete  
{  
    pass  
    {  
        VertexShader = compile vs_1_1 v();  
        PixelShader = compile ps_2_0 p();  
    }  
}
```

텍스처 출력과 관련 있는 부분

- 버텍스 셰이더에 입력이 픽셀 셰이더에 출력이 있음
- 버텍스 셰이더 출력 데이터를 통해서 전달

```

float4x4 wvp : WorldViewProjection;
texture TextureMap_1234;
sampler2D TextureMap_1234Sampler = sampler_state
{
    Texture = <TextureMap_1234>;
};
struct a2v
{
    float4 position      : POSITION;
    float2 texCoord      : TEXCOORD0;
};
struct v2f
{
    float4 position      : POSITION;
    float2 texCoord      : TEXCOORD0;
};
v2f v(a2v In)
{
    v2f Out = (v2f)0;

    Out.position = mul(In.position, wvp);
    Out.texCoord = In.texCoord;
    return Out;
}
float4 p(v2f In) : COLOR
{
    float4 Out;
    Out = tex2D(TextureMap_1234Sampler, In.texCoord.xy);
    return Out;
}
technique Complete
{
    pass
    {
        VertexShader = compile vs_1_1 v();
        PixelShader = compile ps_2_0 p();
    }
}

```

출력

- 출력과 관련 있는 부분
 - 최종 버텍스 값도 출력 함수 노드에 포함

HLSL 특성

- 실제 셰이더 코드는 컴파일 되어 생성
- HLSL 단계에서는 최적화 불필요
- 아래 두 코드 동일한 결과

```
float4 p(v2f In) : COLOR
{
    float4 Out;
    float4 TextureMap_1234 = tex2D(TextureMap_1234Sampler, In.texCoord.xy);
    float4 color_rgba = TextureMap_1234;
    Out = color_rgba;
    return Out;
}
```

```
float4 p(v2f In) : COLOR
{
    float4 Out;
    Out = tex2D(TextureMap_1234Sampler, In.texCoord.xy);
    return Out;
}
```

HLSL 특성

- 중간 변수를 활용 가능

```
v2f Out = (v2f)0;
```

```
Out.position = mul(In.position, wvp);  
Out.texCoord = In.texCoord;  
return Out;
```

```
v2f Out = (v2f)0;
```

```
float4 tt1 = mul(In.position, wvp);  
float2 tt2 = In.texCoord;
```

```
Out.position = tt1;  
Out.texCoord = tt2;  
return Out;
```

```
float4x4 wvp : WorldViewProjection;
```

```
texture TextureMap_1234;
```

```
sampler2D TextureMap_1234Sampler = sampler_state  
{  
    Texture = <TextureMap_1234>;  
};
```

```
struct a2v
```

```
{  
    float4 position : POSITION;  
    float2 texCoord : TEXCOORD0;  
};
```

```
struct v2f
```

```
{  
    float4 position : POSITION;  
    float2 texCoord : TEXCOORD0;  
};
```

```
v2f v(a2v In)
```

```
{  
    v2f Out = (v2f)0;  
    float4 ttt1 = mul(In.position, wvp);  
    float4 ttt2 = In.texCoord;  
  
    Out.position = ttt1;  
    Out.texCoord = ttt2;  
    return Out;  
}
```

```
float4 p(v2f In) : COLOR
```

```
{  
    float4 Out;  
    float2 ttt3 = In.texCoord.xy;  
    float4 ttt4 = tex2D(TextureMap_1234Sampler, ttt3);  
    Out = ttt4;  
    return Out;  
}
```

```
technique Complete
```

```
{  
    pass  
    {  
        VertexShader = compile vs_1_1 v();  
        PixelShader = compile ps_2_0 p();  
    }  
}
```

- 중간 변수를 사용해서 다시 정리해본 셰이더 코드
- 각 함수 분리 가능

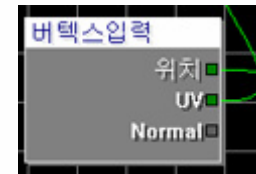
그래프화

- 노드
 - 각 노드는 서브 노드 input node, out node 를 가짐
 - 고유 id 를 가지고 있음
(중간 변수 등에 사용됨)

노드 구성

- 입력 노드 구성

```
<input title="입력" enum="0">
  <node text="위치" nodetype="output" id="1" format="float4" shadertype="vs"></node>
  <node text="UV" nodetype="output" id="2" format="float2" shadertype="vs"></node>
  <node text="NORMAL" nodetype="output" id="3" format="float3" shadertype="vs"></node>
  <shader>
    <header>
      float4x4 wvp : WorldViewProjection;
    </header>
    <vertexinput>
      float4 position : POSITION;
      float3 normal : NORMAL;
      float2 texCoord : TEXCOORD0;
    </vertexinput>
    <vertexcode>
      float4 %1 = mul(In.position, wvp);
      float3 %3 = mul(In.normal, wvp);
      float2 %2 = In.texCoord;
    </vertexcode>
  </shader>
</input>
```

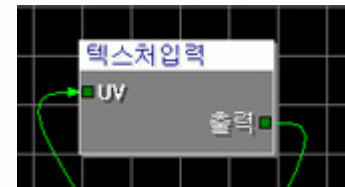


노드 구성

- 텍스처 출력 노드 구

```
<texture title="텍스처입력">
  <texture filter="TGA(+.tga)|+.dds" type="filename" text="텍스처" />
  <node text="UV" nodetype="input" id="1" format="float2" shadertype="vs"></node>
  <node text="출력" nodetype="output" id="2" format="float4" shadertype="ps"></node>
  <shader>
    <header>
      texture %id_texture;

      sampler2D %id_textureSampler = sampler_state
      {
        Texture = &lt;%id_texture&gt;;
      };
    </header>
    <pixelinput>
      float2 texCoord : TEXCOORD0;
    </pixelinput>
    <vertexcode>
      Out.texCoord = %1;
    </vertexcode>
    <pixelcode>
      float4 %idcolor = tex2D(%id_textureSampler, In.texCoord.xy);
      float4 %2 = %idcolor;
    </pixelcode>
  </shader>
</texture>
```



노드 구성

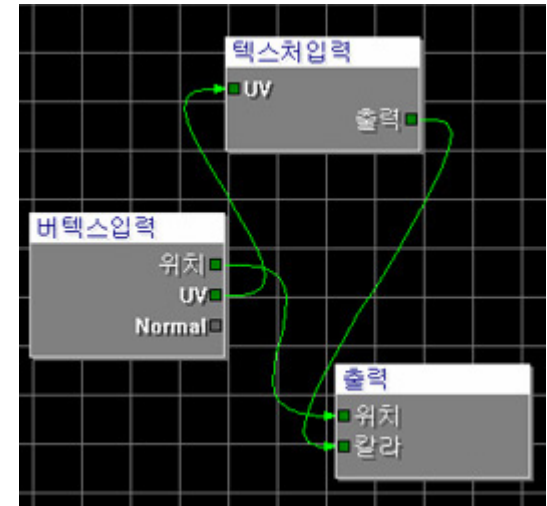
- 출력 노드 구성

```
<output title="출력" enum="0" endpoint="1">
  <node text="위치" nodetype="input" id="1" format="float4" shader="vs"></node>
  <node text="칼라" nodetype="input" id="2" format="float4" shader="ps"></node>
  <shader>
    <pixelinput>
      float4 position : POSITION;
    </pixelinput>
    <vertexcode>
      Out.position = %1;
    </vertexcode>
    <pixelcode>
      Out = %2;
    </pixelcode>
  </shader>
</output>
```



그래프화

- 링크
 - Output node -> input node
 - 노트 연결 제한
 - 같은 데이터형
 - VS->VS, PS->PS
 - 노드간 임시 변수를 사용해 전달

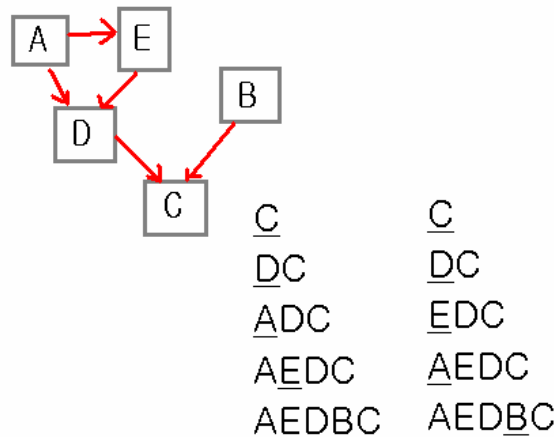


조합

- 각 노드별로 input node, output node를 정의하고 필요한 셰이더 코드를 작성
- 연결 순서에 맞도록 정렬 후 코드 생성

노드 정렬

- 최종 노드에서 출발
- 리스트에 있는 제일 앞 노드에서 추가 안된 입력 노드를 사이에 추가
- 노드의 순서는 모든 노드에 대해서 (출력노드를 가진 노드 => 입력노드를 가진 노드)의 순서가 된다



셰이더 코드 조합

- 순서대로 정해진 코드를 조합
- %id 는 노드 고유 값으로 치환
- %(nodeid) 도 고유 값으로 치환

- 함수 부는 중복해서 더하지 않음

```

float4x4 wvp : WorldViewProjection;
texture texture41_texture;
sampler2D texture41_textureSampler = sampler_state
{
    Texture = <texture41_texture>;
};

struct a2v
{
    float4 position : POSITION;
    float2 texCoord : TEXCOORD0;
};

struct v2f
{
    float4 position : POSITION;
    float2 texCoord : TEXCOORD0;
};

v2f v(a2v In)
{
    v2f Out = (v2f)0;

    float4 input221 = mul(In.position, wvp);
    float2 input222 = In.texCoord;

    Out.texCoord = input222;
    Out.position = input221;

    return Out;
}

float4 p(v2f In) : COLOR
{
    float4 Out;

    float4 texture41_color = tex2D(texture41_textureSampler, In.texCoord.xy);
    float4 texture412 = texture41_color;

    Out = texture412;

    return Out;
}

technique Complete
{
    pass sheep
    {
        VertexShader = compile vs_1_1 v();
        PixelShader = compile ps_2_0 p();
    }
}

```

- 합성 결과
- (데모시연)

기타

- n개의 출력 (RGB, RGBA, R, G, B)
- 데이터 캐스팅
- 생략 가능한 입력 노드
- 멀티패스
- 셰이더 노드 추가

- 오브젝트 - 매트리얼 - 셰이더

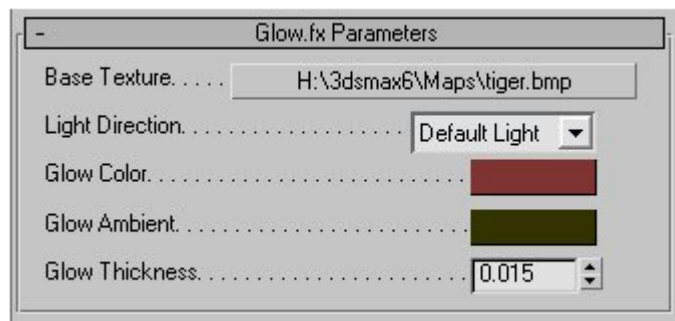
기타

- 생성한 FX파일 3ds MAX 에서 import
- http://sparks.discreet.com/knowledgebase/techdocs/searchable/techdoc_DXMaterialFormat/DxMaterial_Effect_format.htm

```
texture Tex0
<
    string name = "tiger.bmp";
    string UIName = "Base Texture";
    string UIType = "bitmap";
>;
// light direction (view space)
float3 LightDir : Direction <
    string UIName = "Light Direction";
    string Object = "TargetLight";
> = {-0.577, -0.577, 0.577};

// glow parameters
float4 GlowColor <
    string UIName = "Glow Color";
    string UIType = "MaxSwatch";
> = float4(0.5f, 0.2f, 0.2f, 1.0f);

float4 GlowAmbient <
    string UIName = "Glow Ambient";
    string UIType = "ColorSwatch";
> = float4(0.2f, 0.2f, 0.0f, 0.0f);
```



기타

- HLSL과 엔진 연동 이슈
segmentation, annotation
- 셰이더 공유
- 다수의 technique (그림자 렌더링 등)

질문

Spirit of Flame
3D RealTime Graphics Programming Study

Kasa