3D Game Programming 43
-HLSL & ID3DXEffect

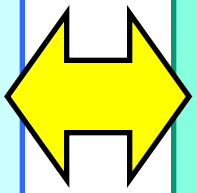
afewhee@gmail.com

- HLSL - 개요
- HLSL - 정점 쉐이더
- HLSL - 픽셀 쉐이더
- HLSL - 정점 + 픽셀 쉐이더
- ID3DXEffect

- 쉐이더 작성 방법
  - 저 수준: 어셈블리어 형식
  - 고 수준(HLSL): C언어와 같은 고급 언어로 작성. C언어에 익숙한 사람은 쉽게 적응

- 저 수준 언어와 HLSL 비교
  - 저 수준 언어는 간결
  - 고 수준 언어는 가속성과 문법이 우수 ➔ 유지 보수가 편리
  - 고 수준 언어는 정점 쉐이더, 픽셀 쉐이더를 혼용해서 사용할 수 있고, 부분적 컴파일 가능

```
// 저 수준 방식

vs_1_1

dcl_position    v0
dcl_color       v1

m4x4 oPos, v0,  c0  // 변환
mov  oD0, v1
```



```
//HLSL
float4x4 WorldViewProj;

struct SvsOut
{
    float4 Pos : POSITION;
    float4 Dif : COLOR0;
};

SvsOut VtxPrc(float3 Pos : POSITION, float4 Dif : COLOR0)
{
    SvsOut Out = (SvsOut)0;
    Out.Pos = mul(float4(Pos, 1), WorldViewProj);
    Out.Dif = Dif;
    return Out;
}
```

- 저 수준과 차이:
  - 쉐이더 작성 언어와 컴파일, 상수 설정만 다를 뿐 기본적인 객체들은 동일
  - 컴파일: D3DXCompileShader(), D3DXCompileShaderFromFile()
  - 필요 객체:
    - {정점|픽셀} 쉐이더 객체: 컴파일 한 쉐이더 코드를 로드
    - 정점 선언(자) 객체: 정점 스트림의 형식 제공
    - 상수 테이블 객체: 상수를 설정하기 위한 객체 ➔ 컴파일 할 때 반환 받음
- 장점
  - 응용프로그램
    - 행렬의 경우 전치가 필요 없음
    - 상수 테이블로 값을 설정할 때 float4이외에 다양한 크기와 타입의 상수들을 설정 가능
  - 코드
    - 저 수준에 비해 작성하기 쉬움 ➔ 문법이 편리
    - 여러 함수로 나누어서 작업이 가능 ➔ c 언어의 장점과 유사
- 단점
  - 고급 언어와 저급언어에서의 단점과 동일

- 준비 단계
  - 고 수준 언어 HLSL로 쉐이더 프로그램 작성
  - 쉐이더 프로그램 컴파일과 상수 테이블 객체 생성
    - D3DXCompileShader(..., &pTable)
  - 정점 쉐이더, 또는 픽셀 쉐이더 생성
    - pDevice->Create{Vertex|Pixel}Shader()
  - 정점 선언 객체 생성
    - pDevice->CreateVertexDeclaration()

- 렌더링
  - 장치에게 정점 쉐이더 객체 또는 픽셀 쉐이더 객체 사용 통보
    - pDevice->Set{Vertex|Pixel}Shader();
  - 정점 선언 객체 연결:
    - pDevice->SetVertexDeclaration();
  - 상수 테이블로 상수 설정:
    - pTable->SetMatrix(), pTable->SetVector();
  - 렌더링: DrawPrimitive();
  - 정점 쉐이더 또는 픽셀 쉐이더 사용 해제:
    - pDevice->Set{Vertex|Pixel}Shader(NULL);

- Keyword

asm	bool	compile	const	decl	do	double
else	extern	false	float	for	half	if
in	inline	inout	int	matrix	out	pass
pixelshader		return	sampler		shared	static
string		struct	technique		texture	true
typedef		uniform	vector		vertexshader	
void		volatile	while			

- 미래에 예약어로 지정될 단어들

auto	break	case	catch	char	class
compile	const	const_cast		continue	default
delete	dynamic_cast		enum	explicit	friend
goto	long	mutable		namespace	new
operator		private		protected	public
register	reinterpret_cast			short	signed
sizeof	static_cast			switch	template
this	throw	try		typename	union
unsigned				using	virtual

※ c++ Keyword와 유사

- 기본 자료 형(Data Type): Scalar, Vector, Matrix

- Scalar
  - bool
  - int -32bit int
  - half-16bit float
  - float-32bit float
  - double – 64 bit float

  - 장치에 따라 half, double은 지원 안될 수 있음

- Scalar Type 사용법
  - 선언: float f;
  - 선언 + 초기화: float f = 3.1f;

  - 배열: float f[3];
  - 배열 + 초기화 : float f[3] = {1.f, 2.f, 3.f};

- Vector
  - vector : float 형 4D == float4
  - vector<Type, Dimension>
  - float{#} : n 차원 float형. float2, float3, float4

- Vector 형 사용법
  - vector
    - vector v;
    - v.x=10; v.g=2.f;
  - vector<Type, Dimension>
    - double 형 2차원 벡터
    - 선언 또는 선언 + 초기화: vector<double, 4> v={1., 2., 3., 4.};
    - 성분으로 지정: v[2]= 3.4
  - float{#} 형
    - float4 v ={1,2,3,4}; float4 v=1.f;//v.x=v.y=v.z=v.w=1.f;
    - float4 v=float4(1,2,3,4); ← 생성자 float4()이용

    - float3 t;
    - float4 v=float4(t, 1.f);
    - float4 v=(1,2,3); // c언어와 동일하게 마지막 3이 채워짐. t.x = t.y = t.z = 3
    - float3 t=3; // t.x = t.y = t.z = 3;
  - swizzling 가능: 단, xyzw와 rgba의 혼용은 안됨, v.xg(X), v.yr(x)

- Matrix
  - matrix <Type, row, column>
  - int{row}x{column} : int2x2, int3x3, int4x4, int4x3
  - float{row}x{column} : float2x2, float3x3, float4x4, float4x3
  - {행|열} 우선 지정 : row_major, col_major
    - 연산의 결과에는 아무런 지장이 없음.
    - 행 우선 순위 방식으로 연산이 진행된다고 코드를 작성
  - ※HLSL의 행렬형은 지정하지 않으면 col_major 임
    - 저 수준 언어와 다르게 사전에 행렬 변수 값을 Transpose를 할 필요 없음
- 행렬 성분 접근
  - ._m{row}{col} ➔ ._m00 ~ ._m33
  - ._{row}{col} ➔ ._11 ~ ._44
  - .[{row}][{col}] ➔ mtTM[0][0] ~ mtTM[3][3]
- 행렬 초기화
  - float2x2 v={1,2,3,4} ➔ v._11=1, v._12=2, v._21=3, v._22 = 4;
  - float2x2 v=float2x2({1,2}, {3,4});
- 행렬 swizzling
  - _m{row}{col}, _{row}{col} 이 혼용 되지 않은 한 swizzling 가능. v._m00_11(X)

- Complex Data Type(혼합형): struct, user define, 텍스처 객체, sampler, shader 객체, …

- struct
  - "struct" 키워드 사용. C언어의 struct와 유사

```
struct Svertex
{
    float3 position;
    float3 normal;
};
```

  - 초기화 : struct Svertex v={ {1,2,3}, {4,5,6}};
  - 멤버 접근: "." operator 이용. v.position = float3(10,20,30);
    float3 t = v.position;

- user define
  - "typedef" 키워드 이용. c와 동일
  - typdef vector<float, 3> point; ➔ point v;

- 텍스처 객체: "texture" 키워드 이용. texture 타입, 변수 이름, 문자열 주해
  - texture tex0 <string name ="MyTexture.bmp">
  - 샘플러 객체는 텍스처 객체를 참조

- Sampler: 픽셀 processing의 샘플링 담당. 샘플러 스테이트, 텍스처, 샘플링 명령으로 구성
  - 단순한 형태: sampler SampDif; sampler 객체 SampDif

  - 샘플러 타입:
    sampler1D → tex1D(), sampler2D→tex2D(), sampler3D→tex3D, samplerCUBE→texCUBE() 함수의 sampling 진행

```
// 텍스처와 샘플러 따로 설정
texture tex0;

sampler3D MySampler = sampler_state
{
    Texture = NULL;
    MinFilter = LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};

// 이것도 가능
texture m_TxNor;
sampler SampNor = sampler_state
{
    Texture = <m_TxNor>;
    MinFilter = LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};
```


- 쉐이더 객체: 쉐이더가 Assemble 또는 컴파일 지정
  - ID3DXEffect 를 사용할 때 HLSL 안에서 HLSL 문법으로 쉐이더 객체를 지정하고 컴파일 방식을 결정
  - 정점 쉐이더 객체: "VertexShader vs"
  - 픽셀 쉐이더 객체: "PixelShader ps"
- 쉐이더 객체 선언 방법

```
VertexShader vs = asm {// 저수준 언어};
VertexShader vs = compile "쉐이더 버전" "대상 함수()";

PixelShader ps =
asm
{
    ps_2_0
    mov oC0, c0
}

PixelShader ps = compile ps_2_0 PixelProc();//PixelProc()함수를 ps_2_0으로 컴파일
```

- 주해 (Annotation)
  - ID3DXEffect 객체를 사용할 때 파라미터에 사용자 정보를 추가하는데 사용
  - 응용프로그램에서 Lookup에 대한 질의로 사용. HLSL자체에는 영향을 주지 않음
  - "<", ">" 안에 위치
  - 데이터 타입, 변수 이름, 등호, 데이터 값, ";" 으로 구성

Ex) texture t< string name="MyTexture.bmp";>;
- "string" 타입은 HLSL에 영향을 주지 않고 질의(Query)용도로 응용 프로그램에서 이용

```
// HLSL file
technique MyTech
<
    int MyInt = 12;
    string InputArray = "MyLookup";
>

// cpp
LPCSTR sName;
hAnnotation = m_pEffect->GetAnnotationByName(hTech, "InputArray");
m_pEffect->GetString( hAnnotation, &sName);
//"MyLookup" 문자열 주소 복사. ➔ 주해에서 InputArray 변수의 이름을 가져오기
```


- 기억 장소 분류(Storage Class)
  - Storage Class 수식어로 변수의 범위(Scope), 수명(life time)을 결정
  - 종류: static, extern, uniform, shard

- static(정적 변수)
  - C언어의 static과 유사
  - static 전역 변수: 응용 프로그램에서 접근 불가. HLSL 코드 내에서 유효
  - static 지역 변수: 함수가 종료되어도 값이 유지.

```
static float MyFloat = 10.f;
```

- extern(외부 변수)
  - 쉐이더 외부에 변수가 위치.
  - 응용 프로그램에서 수정 가능.
  - extern 수식어가 없는 전역 변수는 extern 변수.

```
extern float4x4  mtRotationMatrix;
float 4          vcLight;
```

- Uniform
  - 응용 프로그램에서만 수정 가능 ➔ 모든 영역에서 항상 같은 값을 유지

```
uniform float f = 10.f;        //HLSL 내부 코드에서는 값 변경 불가
// 응용 프로그램 API(Set{Vertex|Pixel}ShaderConstantF, or ID3DXConstantTable Interface)만 수정 가능
```

- Semantic:
  - 쉐이더 입력과 출력을 확인하고, 데이터의 출처와 역할에 대한 분명한 의미를 부여하기 위해 함수, 변수, 인수 뒤에 선택적으로 붙여서 서술
  - 종류
    - Vertex Shader Semantic
    - Pixel Shader Semantic
    - Explicit Register Binding-명시적 레지스터 바인딩
  - 방법
    - 변수, 인수, 함수 뒤에 ":" "SEMANTIC 이름"

    EX)

```
float4 pos: POSITION;
float4 MyFunc( float3 pos: POSITION) : COLOR{}
struct T {     float3 pos : POSITION;    };
```

  - 통상 대문자로 사용

- 정점 쉐이더 시멘틱
  - 데이터 출처에 대한 식별
  - 시멘틱 위치
    - 구조체 멤버 뒤
    - 함수의 인수 뒤
    - 함수 뒤
  - 종류: Vertex Shader Input Semantic, Vertex Shader Output Semantic

- 정점 쉐이더 입력 시멘틱
  - 정점 쉐이더의 입력 레지스터 지정
  - 저 수준 쉐이더의 dcl_과 대응
  - 종류
    - POSITION[n] - 정점의 위치
    - BLENDWEIGHT[n] - 정점 블렌딩 비중 값
    - BLENDINDICES[n] -정점 블렌딩 인덱스 값
    - NORMAL[n] - 정점 법선 벡터
    - PSIZE[n] - Point Size
    - COLOR[n] - 정점 Diffuse, Specular
    - TEXCOORD[n] - 텍스처 좌표
    - TANGENT[n] - 정점 접선 벡터
    - BINORMAL[n] - 정점 종법선 벡터
    - TESSFACTOR[n] - tessellation factor

- 정점 쉐이더 출력 시멘틱
  - 정점 쉐이더의 출력 레지스터 지정
  - 저 수준 명령어 oXXXX와 대응
    - POSITION - 정점의 출력 위치 → oPos
    - PSIZE - Point Size → oPts
    - FOG - 정점 포그 값 → oFog
    - COLOR[n] - Diffuse, Specular → oD0, oD1
    - TEXCOORD[n] - 텍스처 좌표 → oT0 ~ T7
  - 특히, 지정되지 않은 출력레지스터 중에서 픽셀 쉐이더로 데이터를 전달하기 위한 경우 TEXCOORD 출력 시멘틱을 가장 많이 사용

- 정점 쉐이더 시멘틱 사용 예-구조체

```
// 정점 입력에 대한 구조체
struct VS_INPUT
{
    float3 Pos : POSITION;
    float3 Nor : NORMAL;
    float3 Dif : COLOR;
    float2 Tex : TEXCOORD;
};

// 정점 출력에 대한 구조체
struct VS_OUTPUT
{
    float4 Pos : POSITION;
    float4 Dif : COLOR;
    float2 Tex : TEXCOORD0;
    float4 Nor : TEXCOORD7;  // 만약 정점 처리과정에서 출력을 픽셀 쉐이더에 전달할  때
                             // 시멘틱이 없는 경우에 texcoord#를 주로 사용
};
```

- HLSL 함수
  - 반환 타입
  - 함수 이름
  - 인수 목록-선택사항
  - 출력 시멘틱-선택사항
  - 주해-선택사항

- 함수의 인수 리스트에 입력과 출력이 지정되는 경우
  - 반환 타입은 void
  - 입력 레지스터 in, 출력 레지스터 out

```
void VtxPrc1( in   float4 vPos   : POSITION,        // 입력 레지스터 위치
              in   float2 vTex   : TEXCOORD0,       // 입력 레지스터 텍스처 좌표 0
              out  float4 oPos   : POSITION,        // 출력 레지스터 위치
              out  float2 oTex   : TEXCOORD0        // 출력 레지스터 텍스처 좌표 0
              )
{
    oPos = mul(vPos, g_WorldViewProj);
    oTex = vTex;
}
```

- 반환 타입, 인수에 시멘틱 부여, 출력 시멘틱 지정
  - 픽셀 쉐이더를 사용하는 경우에는 반환 타입은 색상이므로 최소한 float4 또는 float4가 포함된 구조체

```
float4 PxlPrc2(float2 vTex : TEXCOORD0) : COLOR0
{
    return tex2D(DiffuseSampler, vTex);
}
```

- 함수의 인수 값 변동을 불허할 경우 uniform 키워드 이용. 반환 타입은 사용자 지정 타입도 가능

- 픽셀 쉐이더 시멘틱-정점 쉐이더 시멘틱과 동일
  - 데이터 출처에 대한 식별
  - 시멘틱 위치
    - 구조체 멤버 뒤
    - 함수의 인수 뒤
    - 함수 뒤
  - 종류: Pixel Shader Input Semantic, Pixel Shader Output Semantic

- 픽셀 쉐이더 입력 시멘틱
  - 픽셀 쉐이더의 입력 레지스터 지정
  - Sampler 객체는 따로 지정
  - 종류
    - COLOR[n] - 정점 처리 과정에서 만들어진 Diffuse, Specular
    - TEXCOORD[n] - 텍스처 좌표

- 픽셀 쉐이더 출력 시멘틱
  - 픽셀 쉐이더의 출력 레지스터 지정
  - 저 수준 명령어 oXXXX와 대응
    - COLOR[n] - 색상 ➔oC0, Oc4. COLOR =COLOR0
    - TEXCOORD[n] - 텍스처 좌표
    - DEPTH[n] - 깊이 ➔ oDepth. DEPTH = DEPTH0

- Explicit Register Binding-명시적 레지스터 바인딩
  - Register 키워드와 저 수준 레지스터 이름 사용
  - 타입 + 변수 + ":" + register("저 수준 레지스터 이름");
  - Ex)
  - sampler SmpDif : register(s0); // Sampler 객체 SmpDif를 register s0에 바인딩
  - float4x4 mtWorld : register(c0); // 전역 변수 mtWorld를 상수 레지스터 c0에 바인딩
  - float4 vcLight : register(c10); // 전역 변수 vcLight를 상수 레지스터 c10에 바인딩

- HLSL 내장 함수를 적절히 사용하려면 쉐이더 버전이 2.0이상

- Math 함수
  - 삼각함수: sin, cos, tan, acos, asin, cot
  - 지수함수: exp: 자연대수 e의 승수, exp2(): 2의 승수, pow(x,y)=x^y
  - log함수: log, log2, log10
  - 벡터함수: dot(), cross(), distance(), length(), len(), sqrt(), rsqrt(), normalize() reflect(), refract()
  - 행렬함수: determinant(), transpose()
  - 이외: abs(), ceil(), floor(), round(), fmod(), lerp(), saturate()

- 샘플링 함수
  - tex{n}D(s,t) n차원 텍스처 sampler s로 t 좌표에 대한 색상 추출. texCUBE(s,t)
  - tex{n}Dproj(s,t): 정점의 깊이(z,w)를 저장한 텍스처에 대한 n차원 투영 샘플링. t= 4D임에 주의 샘플링 전에 t는 t.w로 나누어짐
  - tex{n}Dbias(s,t): 보정된 n차원 텍스처에 대한 Sampliing

- 정점 처리 함수 정의
  - 출력 레지스터 구조체로 반환
  - 함수의 입력은 구조체, 또는 자료형으로 처리

- 전역 변수 정의
  - 정점에 적용할 행렬 float4x4
  - 라이팅 float4
  - 색상 float4
- 샘플러 객체 정의 ← 픽셀 쉐이더
  - 텍스처 객체 정의
  - 샘플러의 필터링, 어드레싱 정의
- 입/출력 레지스터 정의
  - 구조체로 작성: 반드시 위치는 지정
  - 구조체 변수에 semantic 부여
- 정점 처리 함수 정의
  - 함수의 입력은 구조체, 또는 자료형으로 처리
  - 출력 레지스터 구조체로 반환
- 픽셀처리 함수 정의
  - 함수의 입력은 구조체, 또는 자료형으로 처리
  - 반환 타입 float4. 함수에 COLOR Semantic 사용
- Technique 정의 → ID3DXEffect를 사용할 때
  - pass 안에 정점 쉐이더, 픽셀 쉐이더 객체 생성 방법 지정
  - 가상 머신에 대한 렌더링 옵션 지정 ➔ D3DRENDERSTATETYPE 에 대응

2. HLSL - 정점 쉐이더

- HLSL
  - Input
    - 입력 없음 ➔ 의미 없음
    - Position 등 "dcl_"로 선언될 수 있는 입력 semantic에 대한 모든 것

```
float4 PxlPrc() : POSITION
{
   return  float4(1,1,1,1);
}
```

  - Output
    - 위치에 대한 출력 레지스터(oPos)에 대해서 반드시 필요
    - 출력 semantic에 대한 모든 것
  - extern 변수
    - 외부 프로그램에서 값을 설정할 수 있는 변수
    - 고정 함수 파이프라인과 비교했을 때 주로 변환 행렬, 빛의 방향, Material에 대한 색상 등을 정할 때 사용

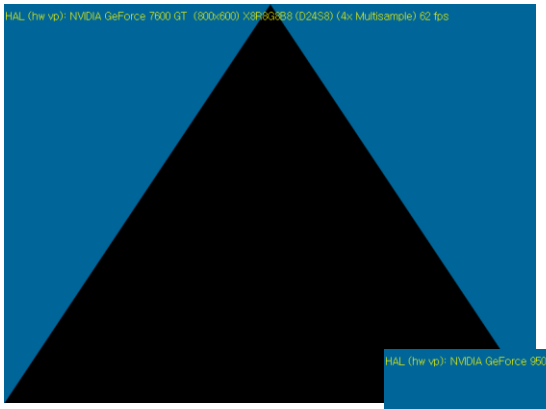
- 응용 프로그램
  - 상수 테이블
    - extern 변수 값 설정
  - 디폴트로 렌더링 머신의 일부 상태 값 활용

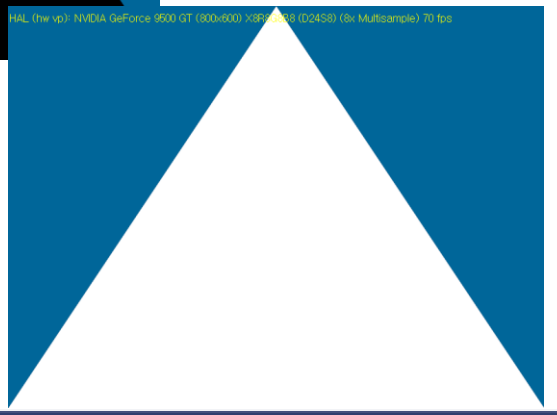
- 입력 받은 정점을 화면에 출력

```
FLOAT4 VtxPrc(float3 Pos : POSITION) : POSITION
{
    return Pos;
}
```

```
// HLSL Compile
hr = D3DXCompileShader(
    sHlsl
,   iLen
,   NULL
,   NULL
,   "VtxPrc"          // 쉐이더 실행 함수
,   "vs_1_1"          // 쉐이더 버전
,   dwFlags
,   &pShd
,   &pErr
,   NULL );
```

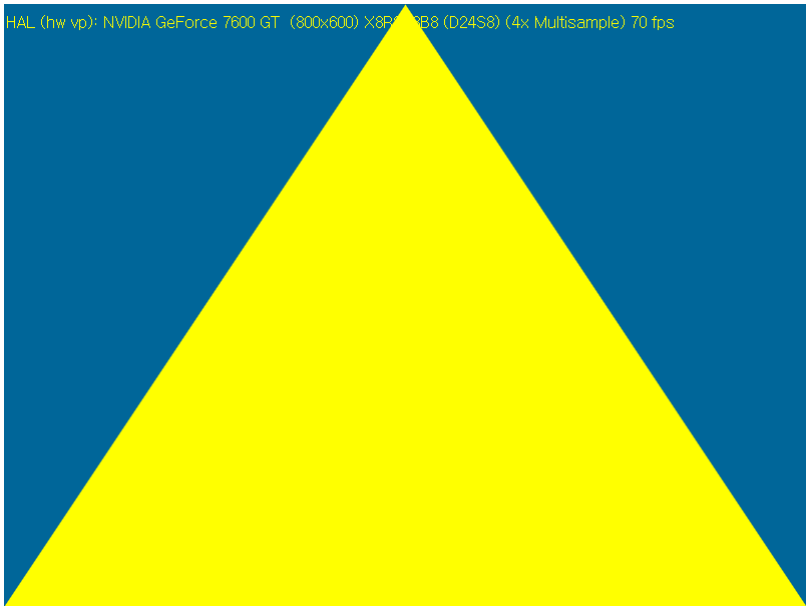
```
// HLSL Compile from File
D3DXCompileShaderFromFile()
```





상수 설정

```
// 전역 변수
float4  Dif  = float4(0,1,1,1);

void VtxPrc1(   in float3 iPos : POSITION
            ,   out float4 oPos : POSITION
            ,   out float4 oD0 : COLOR
)
{
    // 지역 변수
    float4  Dif  = float4(1,1,0,1);

    oPos = float4(iPos, 1);
    oD0  = Dif;
}
```

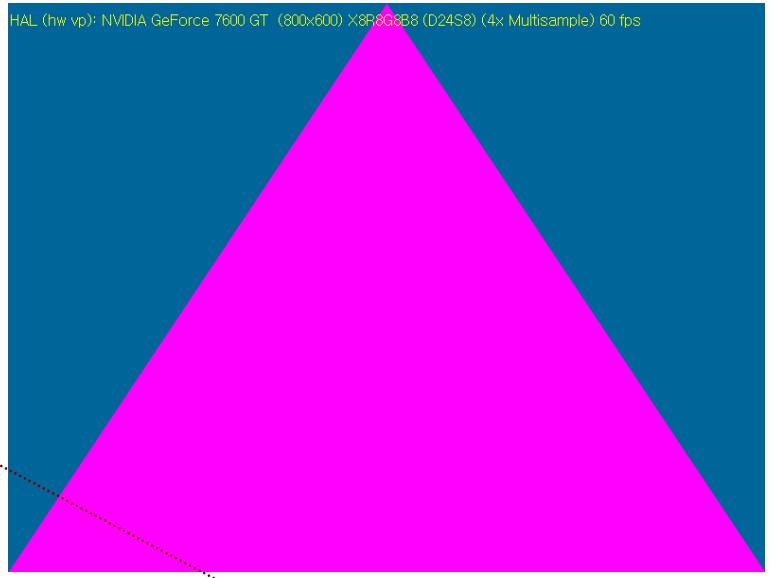


변수의 이름이 같으면 지역에서 선언한 변수가 우선 : scope는 c언어와 유사

● 상수 설정

storage class 키워드 없는 전역 변수: extern → 외부 프로그램에서 변경 가능

```
// 전역 변수
float4  Dif   = float4(0,1,1,1);

void VtxPrc2(   in float3 iPos : POSITION
            ,   out float4 oPos : POSITION
            ,   out float4 oD0 : COLOR
)
{
    // float4 Dif  = float4(1,1,0,1);
    oPos = float4(iPos, 1);
    oD0  = Dif;
}
```



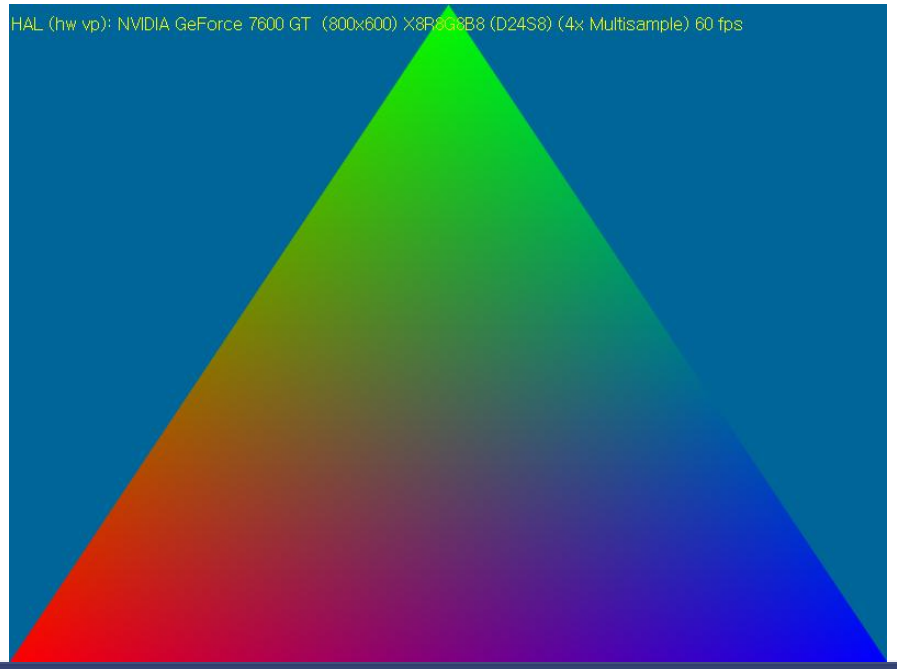

```
// 상수 테이블 반환
hr = D3DXCompileShaderFromFile
( ...,    &m_pTbl );
```

```
// 상수 레지스터 설정
D3DXCOLOR         color(1,0,1,1);
m_pTbl->SetVector(m_pDev, "Dif", (D3DXVECTOR4*)&color);
```

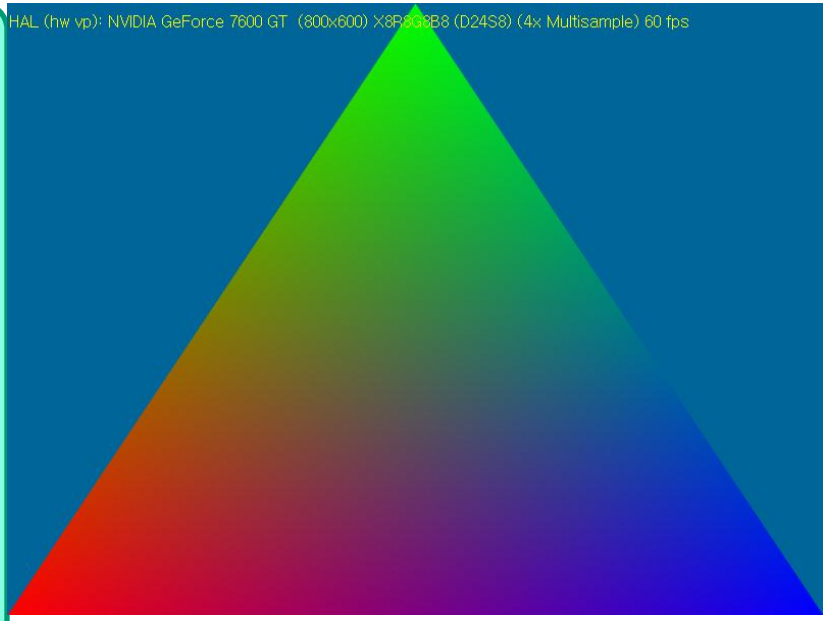
HLSL 변수가 extern 이면
외부 프로그램에서
"쉐이더 상수 테이블"
객체를 이용해서 변경 가능

- 변수는 extern
  - 변수를 전역에 설정. storage class 키워드를 생략하면 extern
- HLSL 코드를 컴파일 할 때 상수 테이블 객체를 반환
  - D3DXCompileShaderFromFile(..., &pConstTable);
- 상수 테이블로 변수 설정
  - 벡터: pConstTable->SetVector()
  - 행렬: pConstTable->SetMatrix(), SetFloat()
  - 기타: pConstTable->SetFloat(), SetFloatArray(), SetValue();
  - 색상: D3DXCOLOR 구조체 변수 주소를 D3DXVECTOR4*로 캐스팅해서 사용
    pConstTable->SetVector(...(D3DXVECTOR4*)&...)

```
void VtxPrc(    in float3 iPos : POSITION     // dcl_position
              , in float4 iDif : COLOR0       // dcl_color0
              , out float4 oPos: POSITION     // oPos
              , out float4 oDif: COLOR0       // oD0
)
{
   oPos = float4(iPos, 1);
   oDif = iDif;
}
```

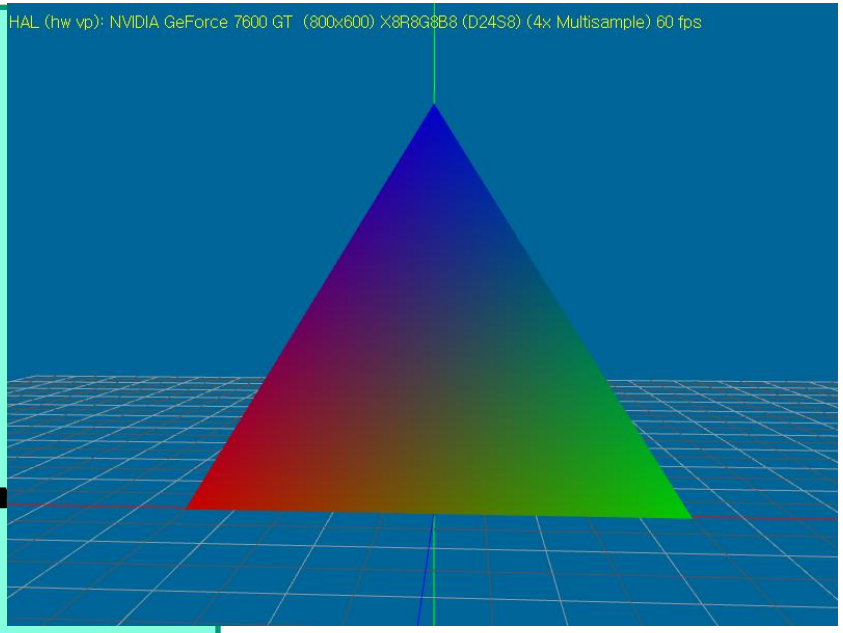


```
// For Vertex Processing Input
struct SvsIn
{
    float3 Pos : POSITION ;   // dcl_position
    float4 Dif : COLOR0    ;   // dcl_color0
};

// For Vertex Processing Output
struct SvsOut
{
    float4 Pos : POSITION ;   // oPos
    float4 Dif : COLOR0    ;   // oD0
};

SvsOut VtxPrc(SvsIn In)
{
    SvsOut Out = (SvsOut)0;     // Initialized to 0

    Out.Pos = float4(In.Pos, 1);
    Out.Dif = In.Dif;
    return Out;
}
```



정점 처리, 픽셀 처리에 대한 입력 또는 반환을 구조체로 하려면 semantic을 사용

```
// Output Vertex Processing
struct SvsOut
{
    float4 Pos : POSITION;  // oPos
    float4 Dif : COLOR0;    // oD0
};

// World * View * Projection Matrix
float4x4 WorldViewProj;

SvsOut VtxPrc(  float3 Pos: POSITION  // dcl_positio
              ,   float4 Dif: COLOR0    // dcl_color0
    )
{
    SvsOut Out = (SvsOut)0;          // Initialized to 0

    Out.Pos = mul(float4(Pos, 1), WorldViewProj);
    //Out.Dif = Dif*0.8f; // darken a little ;
    Out.Dif = Dif;

    return Out;
}
```

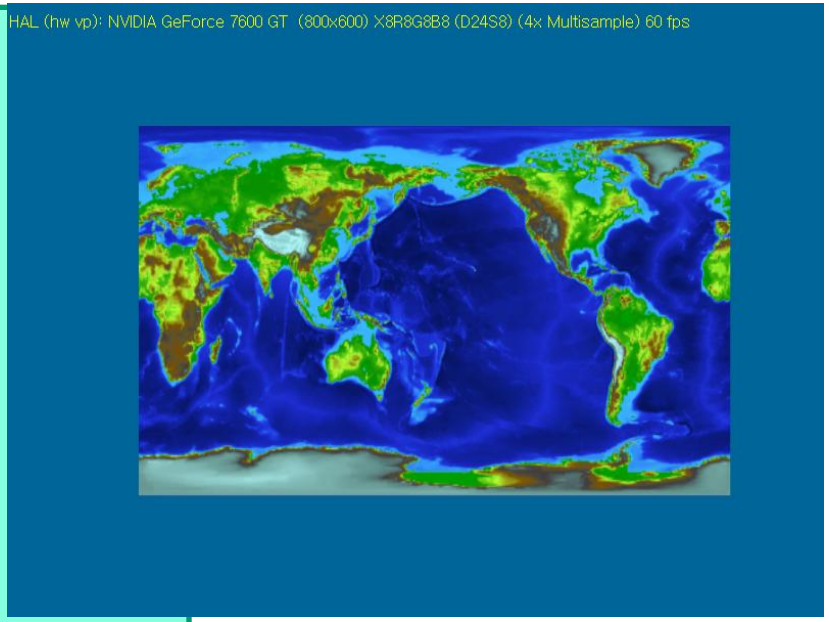


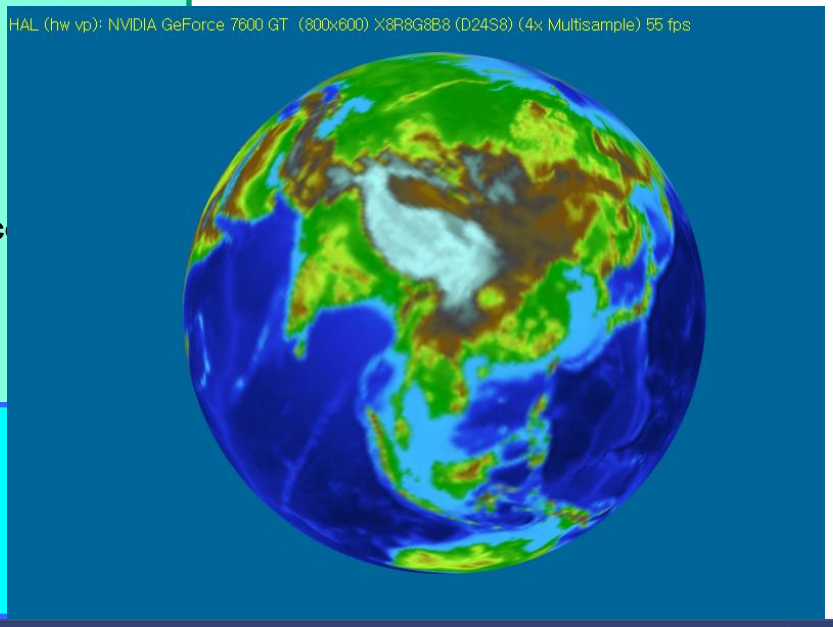
저 수준과 다르게 행렬을 전치(Transpose)할 필요가 없음

```
// 상수 연결: 상수 테이블 사용
m_pTbl->SetMatrix(m_pDev, "WorldViewProj", &(mtWld*mtViw*mtPrj));
```

```
// For Vertex Processing Output
struct SvsOut
{
    float4 Pos : POSITION   ;   // oPos
    float4 Dif : COLOR0     ;   // oD0
    float2 Tx0 : TEXCOORD0  ;   // oT0
};

// World * View * Projection Matrix
float4x4 m_mtWVP;

// Vertex Color
float4  m_vcColor;

SvsOut VtxPrc(  float3 Pos : POSITION      // dcl_position
              ,  float4 Dif : COLOR0       // dcl_color0
              ,  float4 Tx0 : TEXCOORD0 // dcl_texcoord0
){
    SvsOut Out = (SvsOut)0;                    // Initialized to 0

    Out.Pos = mul(float4(Pos, 1), m_mtWVP);  // Transform
    Out.Dif  = Dif*m_vcColor;                        // Diffuse * Color c
    Out.Tx0 = Tx0;

    return Out;
}
```

```
// Setup Constant Register
D3DXMATRIX mtWVP = mtWld * mtViw * mtPrj;
m_pTbl->SetMatrix(m_pDev, "m_mtWVP", &mtWVP);

D3DXCOLOR color(.7F, 1.f, 1.F, 1);
m_pTbl->SetVector(m_pDev, "m_vcColor", (D3DXVECTOR4*)&color);
```





- HLSL
  - 외부에서 설정할 변수는 extern으로 작성
  - 정점 처리 함수 작성

- 컴파일과 정점 쉐이더 객체 생성
  - 상수 테이블 객체 얻기
  - HLSL 코드를 컴파일 할 때 상수 테이블 객체를 반환: D3DXCompileShader(…, &pTable);
  - 정점 쉐이더 객체, 정점 선언 객체 생성

- 렌더링
  - 정점 쉐이더 사용 통보
  - 정점 쉐이더, 정점 선언 객체를 디바이스에 연결
  - 상수 테이블 객체로 HLSL 변수 설정
  - 렌더링
  - 정점 쉐이더, 정점 선언 객체 사용 해제

3. HLSL - 픽셀 쉐이더

- HLSL
  - Input: 0개 이상의 입력
  - Output: 반드시 필요
    - COLOR0 - float4 형
  - Diffuse color
  - Texture Coordinate
  - Texture
    - 텍스처 포인터
  - Sampler
    - 텍스처에서 uv 좌표를 이용해 추출
    - addressing, Filtering 타입, 방법 등 가능
    - 여러 개의 텍스처가 연결될 때 필요

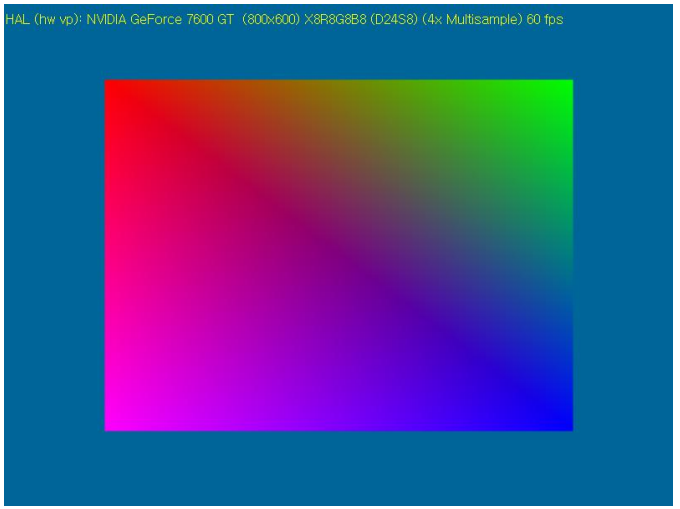
```
float4 PxlPrc() : COLOR0
{
   return  float4(1,0,1,1);
}
```

- 응용 프로그램
  - 상수 테이블
    - extern 변수 값 설정
  - 디폴트로 렌더링 머신의 상태 값 활용

● 색상 출력

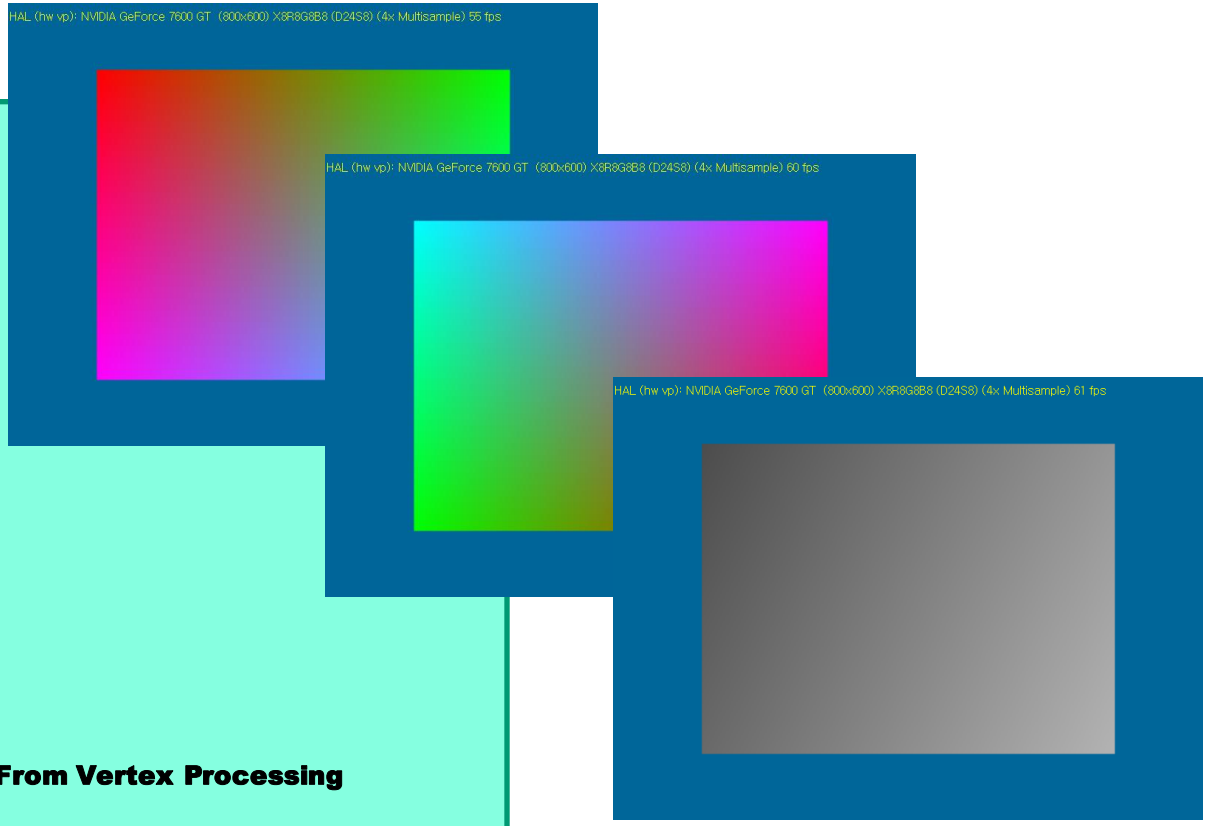
```
void PxlPrc( in float4 iDif : COLOR0 // From Vertex Processing
             out float4 oDif : COLOR0 // Output oC0
){
   oDif = iDif;
}
```

```
// HLSL Compile
hr = D3DXCompileShaderFromFile(
   "data/Shader.fx"
,   NULL
,   NULL
,   "PxlPrc"          // 쉐이더 실행 함수
,   "ps_1_1"          // 쉐이더 버전
,   dwFlags
,   &pShd
,   &pErr,    NULL );
```




```
m_pDev->SetPixelShader(m_pPs);                    // Pixel Shader 사용

m_pDev->SetFVF(VtxD::FVF);
m_pDev->DrawPrimitiveUP( D3DPT_TRIANGLEFAN, 2, m_pVtx, sizeof(VtxD));

m_pDev->SetPixelShader( NULL);                    // Pixel Shader 해제
```

- 반전, 단색화




```
// Pixel Processing Type
int g_nPxlPrc = 2;

// Color Inversion
float4  PxlInverse(float4 Input)
{
    return 1 - Input;
}
// Color Monotone
float4  PxlMonotone(float4 InColor)
{
    float4 Out = 0.f;
    float4 d = float4(0.299, 0.587, 0.114, 0);
    float4 t = InColor;
    Out = dot(d, t);
    Out.w = 1;
    return Out;
}

// Main Pixel Processing
float4 PxlPrc(  in float4 iDif : COLOR0     // From Vertex Processing
) : COLOR0
{
    float4 Out=float4(0,0,0,1);

    if(1== g_nPxlPrc)
        Out = PxlInverse(iDif);
    else if(2 == g_nPxlPrc)
        Out = PxlMonotone(iDif);
    else
        Out = iDif;

    return Out;
}
```

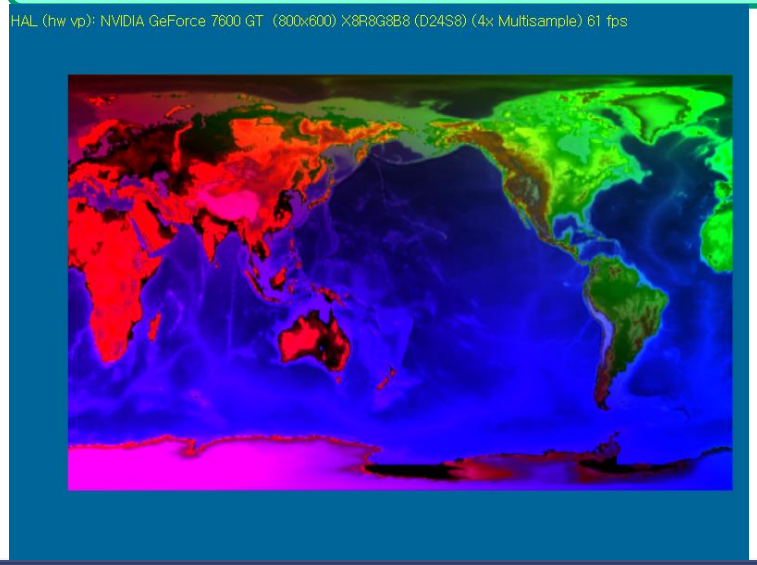
```
//조건 문을 사용하려면 ps_2_0 이상으로 컴파일
hr = D3DXCompileShaderFromFile(
...
,    "PxlPrc"              // 쉐이더 실행 함수
,    "ps_1_1"              // 쉐이더 버전
...);
```

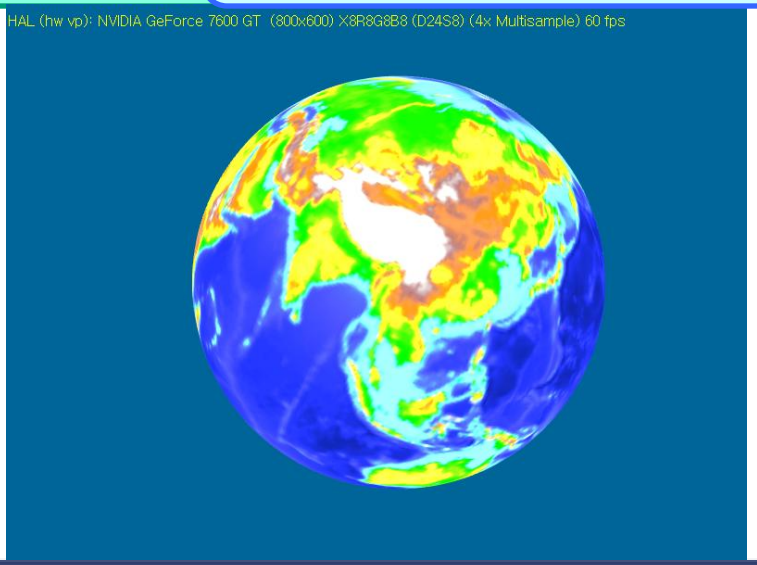
```
// 픽셀 연산에 대한 상수 연결
hr = m_pTbl->SetInt(m_pDev, "g_nPxlPrc", m_nType);
```

- Single Sampling

```
// sampler
sampler smpDif;

float4 PxlPrc(  in float4 Dif : COLOR0   // Diffuse From Vertex Processing
              ,  in float4 Tx0 : TEXCOORD0   // Texture Coord0 "
) : COLOR0
{
   float4  Out;
   Out = tex2D(smpDif, Tx0);  // Sampling Texture with Sampler
   Out *= (Dif*2);                  // Modulate2X Diffuse
   Out.w = 1;                       // Setup alpha = 1

   return Out;
}
```

// 샘플링 함수
tex2D(Sampler, Texture Coord)





```
// pDev->SetTexture에 해당하는 샘플러 ➔ 레지스터에 지정
sampler smp0 : register(s0); // pDev->SetTexture(0, ...)
sampler smp1 : register(s1); // pDev->SetTexture(1, ...)
```

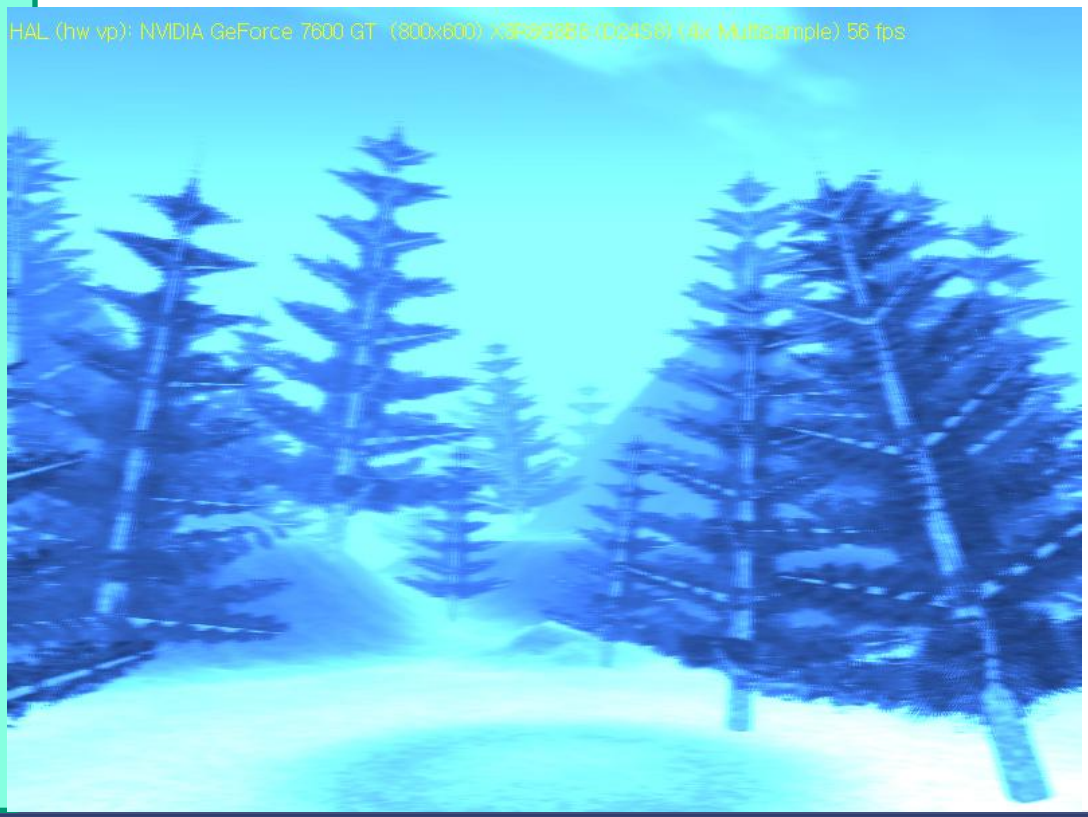
```
// Smapler state를 이용한 샘플러 객체 선언
sampler smpDif0 : register(s0) = sampler_state
{
    MinFilter = POINT;      // Filtering 설정
    MagFilter = POINT;
    MipFilter = POINT;
    AddressU = Wrap;        // Addressing 설정
    AddressV = Wrap;
};
// Smapler state를 이용한 샘플러 객체 선언
sampler smpDif1 : register(s1) = sampler_state
{
    MinFilter = NONE;
    MagFilter = NONE;
    MipFilter = NONE;
    AddressU = Wrap;
    AddressV = Wrap;
};
```

```
float4 PxlPrc(...) : COLOR0
{
    t0 = tex2D(smpDif0, Tx0);    // Sampling Texture
    t1 = tex2D(smpDif1, Tx0);    // Sampling Texture
    Out = t0 *.5 + t1 * 2.;
    return Out;
}
```

```
m_pDev->SetPixelShader(m_pPs);

m_pDev->SetTexture( 0, m_pTx0 );
m_pDev->SetTexture( 1, m_pTx1 );

m_pDev->DrawPrimitiveUP( ...);

m_pDev->SetTexture( 0, NULL);
m_pDev->SetTexture( 1, NULL);

m_pDev->SetPixelShader(NULL);
```

- Blurring 방법 ➔ 저 수준 픽셀 쉐이더 참조
  - 프로그래머가 저 수준 언어보다 편리하게 작성
  - 분기 문, Loop를 사용하려면 ps_2_0 이상 필요

```
sampler SampDif;

float4 PxlPrc(  float4 Dff  : COLOR0,
                float2 Tx0  : TEXCOORD0) : COLOR{
    float4 Out=0.f;
    //float4 Mono={0.299, 0.587, 0.114, 0.0};
    float4 Mono=float4(0.8, 0.9, 1., 0);
    for(int x=-4; x<=4; ++x)
    {
        float2 T    = Tx0;
        T.x +=(2.f * x)/1024.f;

        Out +=Dff * tex2D( SampDif, T ) * exp( (-x*x)/8.f);
    }
    Out /=5.f;
    Out.a = 1.f;
    float d = dot(Out, Mono);
    Out.r = d*0.5F;
    Out.g = d*1.F;
    Out.b = d*2.F;
    Out *= 1.5F;

    return Out;
}
```



- Procedural Texture :
  - 쉐이더 코드로 텍스처 픽셀 결정
  - 노이즈(Noise) 등과 같은 수학 함수로 텍스처를 생성할 때 주로 사용
  - D3DXFillTextureTX() 함수를 이용해서 텍스처 픽셀을 채움

```
//Texel Processing

float4 TxlPrc(float2  In : POSITION)  : COLOR0
{
   float4  Out;
   Out = float4(In.x, 0, 0, 0);
   ...
    return Out;
}
```

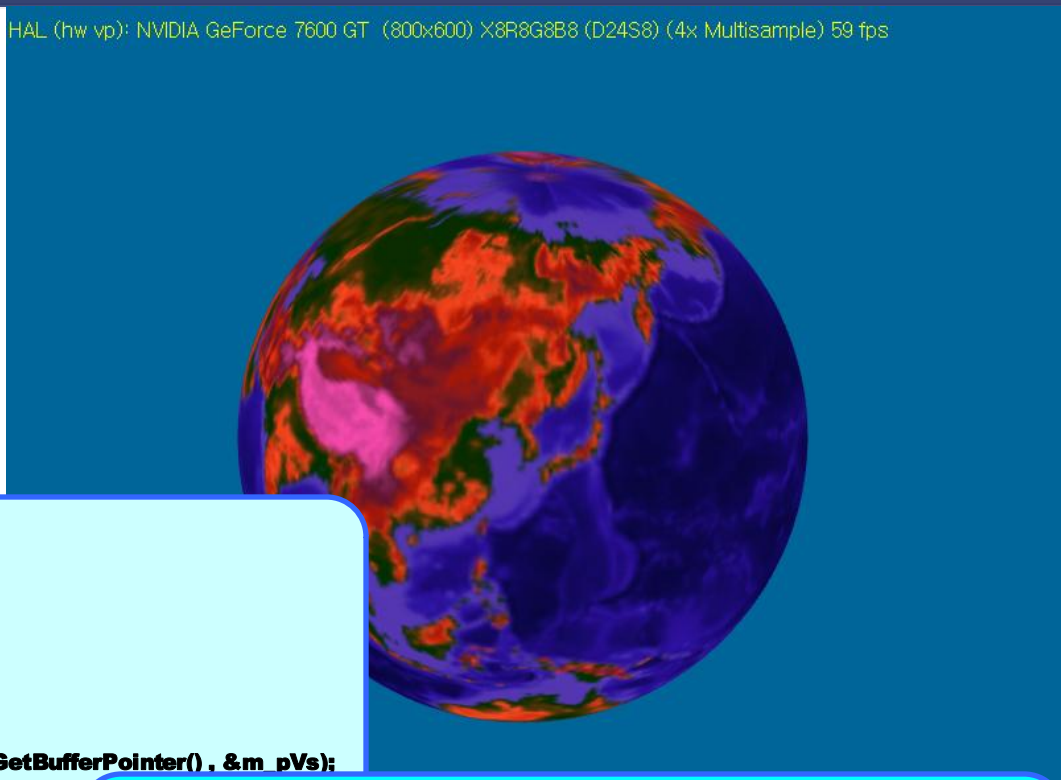
```
// HLSL Compile
hr = D3DXCompileShaderFromFile(
    "data/Shader.fx"
,    NULL
,    NULL
,    "TxlPrc"              // 쉐이더 실행 함수
,    "tx_1_0"              // 텍셀 버전
,    dwFlags
,    &pShd, &pErr, NULL );
```


```
//텍스처(procedural texture) 생성
D3DXCreateTexture(m_pDev, ..., &m_pTx);

// 텍셀을 이용 텍스처 픽셀 채우기
D3DXFillTextureTX(m_pTx, (DWORD*)pShd->GetBufferPointer(), NULL, 0);
```

4. HLSL - 정점 + 픽셀 쉐이더

- 코드
  - 하나의 파일에 정점 쉐이더, 픽셀 쉐이더 코드 작성 가능
  - 코드가 하나의 파일에 있어도 동작은 독립적임
- 쉐이더 객체
  - 정점 쉐이더, 픽셀 쉐이더 각각 컴파일
- 상수 테이블
  - 정점 쉐이더, 픽셀 쉐이더를 각각의 쉐이더에 따라 컴파일 할 때 상수 테이블 객체도 반환 받아 사용

● 혼용 예



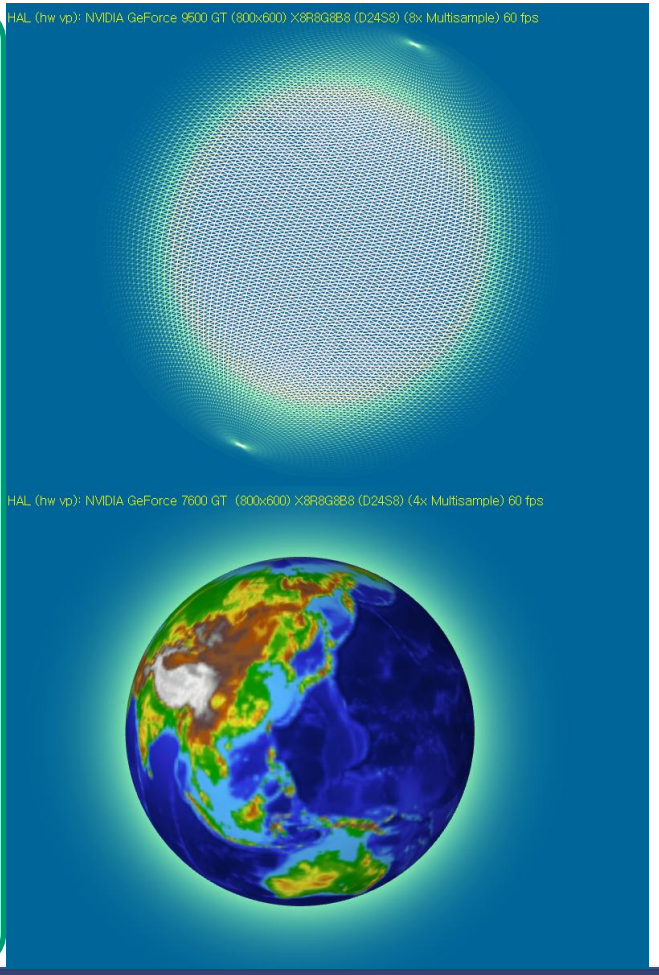

```
float4x4    g_WorldViewProj;
sampler     DiffuseSampler;

// 정점 프로세싱
void VtxPrc1(   in  float4 vPos  : POSITION,
                in  float2 vTex  : TEXCOORD0,
                out float4 oPos  : POSITION,
                out float2 oTex  : TEXCOORD0
                )
{
    oPos = mul(vPos, g_WorldViewProj);
    oTex = vTex;
}

float4  g_Diffuse={1,1,1,1

// 픽셀 프로세싱
void PxlPrc1(   in  float2 v
TEXCOORD0,
                out float4 oCol
{
    oCol = tex2D(DiffuseSa
    oCol *= g_Diffuse;
}
```

```
// 정점 쉐이더 컴파일
hr = D3DXCompileShaderFromFile( "data/hlsl.fx"
                              , NULL, NULL
                              , "VtxPrc1"
                              , "vs_1_1"
                              , dwFlags, &pShd
                              , &pErr, &m_pTbV
                              );
// 정점 쉐이더 생성
hr = m_pDev->CreateVertexShader( (DWORD*)pShd->GetBufferPointer() , &m_pVs);

// 픽셀 쉐이더 컴파일
hr = D3DXCompileShaderFromFile( "data/hlsl.fx"
                              , NULL, NULL
                              , "PxlPrc1"
                              , "ps_1_1"
                              , dwFlags, &pShd
                              , &pErr, &m_pTbP
                              );
// 픽셀 쉐이더 생성
hr = m_pDev->CreatePixelShader( (DWORD*)pShd->GetBuffer

// 정점 선언 개체 생성
D3DVERTEXELEMENT9 vertex_decl[MAX_FVF_DECL_SIZE]={
memset(vertex_decl, 0, sizeof(vertex_decl));
D3DXDeclaratorFromFVF(VtxDUV1::FVF, vertex_decl);
if(FAILED(m_pDev->CreateVertexDeclaration( vertex_decl, &
    return -1;
```

```
// Render
m_pDev->SetVertexShader(m_pVs);
m_pDev->SetPixelShader(m_pPs);

m_pDev->SetVertexDeclaration( m_pFVF );


// 버텍스 연산에 대한 상수 연결
m_pTbV->SetMatrix(m_pDev, "g_WorldViewProj", &(m_mtWld * mtViw * mtPrj));

D3DXCOLOR      color(1, 0.3F, 0.7F, 1);
// 픽셀 연산에 대한 상수 연결
m_pTbP->SetVector(m_pDev, "g_Diffuse", (D3DXVECTOR4*)&(color));

m_pDev->SetTexture( 0, m_pTx );
m_pDev->DrawPrimitiveUP( D3DPT_TRIANGLESTRIP, m_iNvx - 2, m_pVtx, sizeo
f(VtxDUV1));

m_pDev->SetVertexShader(NULL);
m_pDev->SetPixelShader(NULL);
```

- 혼용 예-Glow
  - 외각을 표현하는 오브젝트 위치 = 원본 오브젝트 위치 + 법선 벡터 * α
  - 원본 오브젝트의 법선과 원본 오브젝트의 중심 위치에서 카메라의 위치에 대한 단위 벡터의 내적으로 색상을 설정 ➔ 바깥쪽 또는 안쪽으로 밝아짐
  - CCW로 렌더링 ← culling CW

```
float  g_fThick;
float3 g_vcGlow = float3(0, 0, 1);                  // Glow axis

SVsOut VtxPrc1(     float4 Pos : POSITION     // 로컬위치좌표
             ,   float4 Nor : NORMAL           // 법선벡터
             ,   float2 Tx0 : TEXCOORD0        // 디퓨즈 맵
){
   SVsOut  Out = (SVsOut)0;

   float4 P = mul(Pos,  m_mtWld);
   float3 N = normalize(mul(Nor,  (float3x3)m_mtWld));  //법선 회전

   P +=float4(N,0) * g_fThick;
   P = mul(P,  m_mtViw);
   P = mul(P,  m_mtPrj);

   float4 G = float4(1.0f, 1.0f, 0.2f, 1.0f);
   float Power;
   Power  = dot(N, g_vcGlow)+1;
   Power *= 0.6F;
   Power = pow(Power, 4.);

   Out.Pos = P;                                  // 정점 위치 출력
   Out.Dff  = G * Power;
   Out.Tx0 = Tx0;                                // 디퓨즈 맵 좌표

   return Out;
}
```

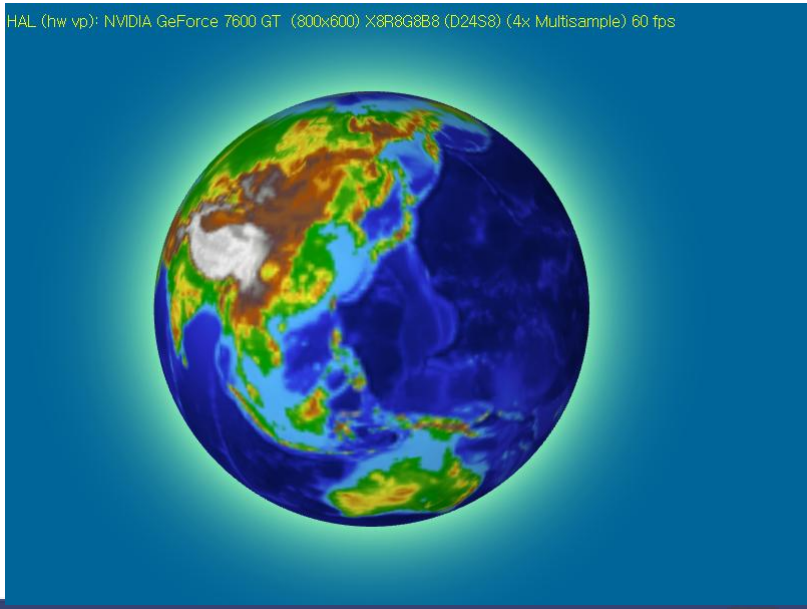


- HLSL에 대한 추상 클래스

```
struct IHlslEffect
{
    virtual ~IHlslEffect(){};
    virtual INT  Create(void* p1=NULL,void* p2=NULL,void* p3=NULL,void* p4=NULL)=0;
    virtual void Destroy()=0;
    virtual INT  Begin()=0;
    virtual INT  End()=0;
    virtual INT  SetupDecalarator(DWORD dFVF)=0;
    virtual INT  SetMatrix(char* sName, D3DXMATRIX* v)=0;
    virtual INT  SetVector(char* sName, D3DXVECTOR4* v)=0;
    virtual INT  SetColor(char* sName, D3DXCOLOR* v)=0;
    virtual INT  SetFloat(char* sName, FLOAT v) =0;
};
```

- 객체 생성 함수

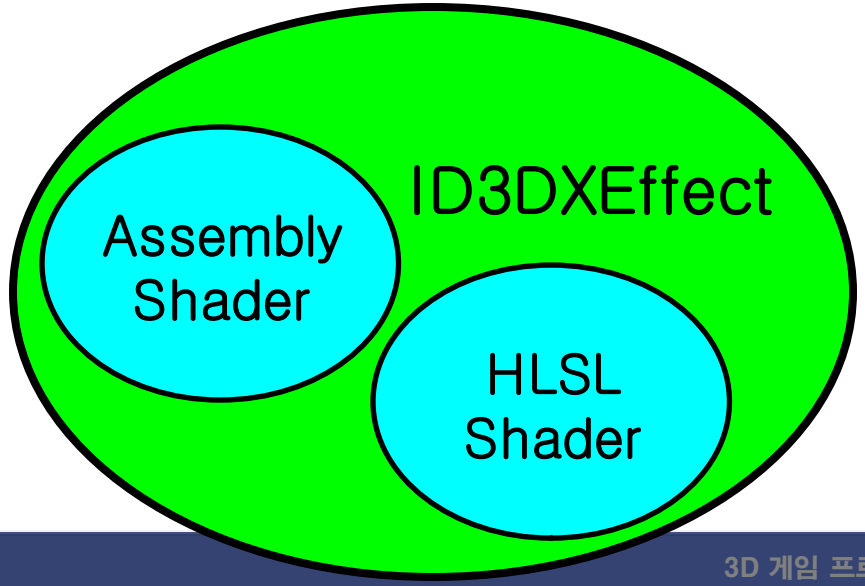
```
int LcHlsl_CreateShader(char* sCmd
, IHlslEffect** pData
, void* pDevice
, char* sFunction
                , char* sShaderMode
                , void* v1
                , void* v2 =NULL);
```



5. ID3DXEffect

- ID3DXEffect
  - 저 수준, 고 수준 쉐이더를 응용 프로그램에서 편리하게 사용하기 위해 정점 쉐이더, 픽셀 쉐이더, 상수 테이블 등을 혼합해서 만든 객체

- 특징
  - 정점 쉐이더, 픽셀 쉐이더 객체를 생성하지 않음 ← 정점 선언 객체는 필요
  - 저 수준, 고 수준 코드 모두 같은 함수로 컴파일
  - 여러 패스를 HLSL로 작성해서 지정해서 정점 쉐이더, 픽셀 쉐이더 처리 함수를 적절히 혼합해서 사용가능 ➔중복된 코드 줄어듦
  - 정점 쉐이더, 픽셀 쉐이더에 대한 전역 변수를 같이 사용
  - HLSL에서 제공하는 모든 기능 이외에 렌더링 상태 머신의 상수 값 등을 패스 안에서 설정 가능
  - 응용 프로그램에서 간결하게 사용



ID3DXEffect 예 - HLSL

```
// HLSL 코드

float4x4    m_mtWld;
...

// 정점 쉐이더 프로세스
SVsOut VtxPrc(  float3 Pos : POSITION,
                float4 Dif : COLOR0
             )
{
   SVsOut  Out = (SVsOut)0;
   ...
   return Out;
}


// 픽셀 쉐이더 프로세스
float4 PxlPrc(SVsOut  In) : COLOR
{
   float4 Out= In.Dff;
   ....
   return Out;
}

technique Tech0
{
   pass P0
   {
      // Shader 컴파일
      VertexShader = compile vs_1_1 VtxPrc();
      PixelShader  = compile ps_1_1 PxlPrc();
   }
}
```

```
// Effect Technique 지정
technique Tech0
{
   // 렌더링 패스 지정
   pass P0
   {
      // Shader 컴파일
      VertexShader = compile vs_1_1 VtxPrc();
      PixelShader  = compile ps_1_1 PxlPrc();
   }
}
```

ID3DXEffect 예 - 응용 프로그램

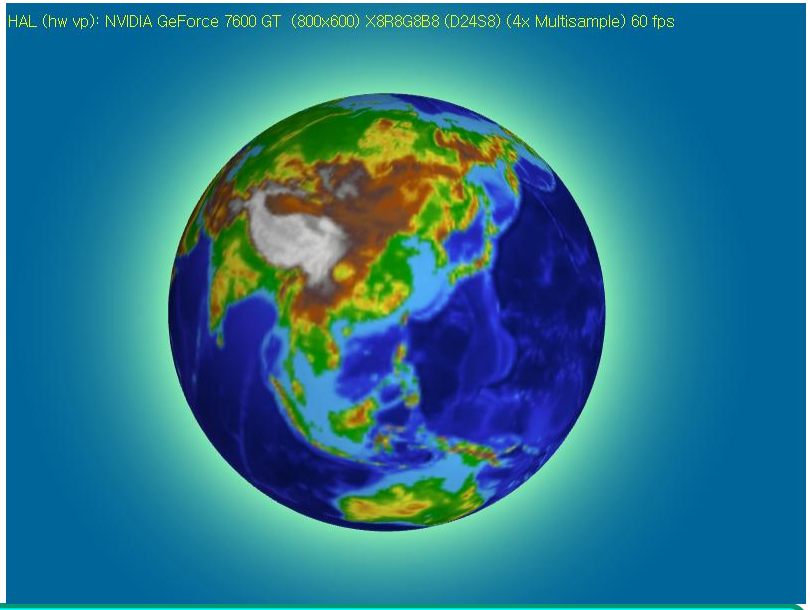
```
// 컴파일 + ID3DXEffect 객체 생성
D3DXCreateEffectFromFile(
    m_pDev
    , "data/hlsl.fx"
    , NULL
    , NULL
    , dwFlags
    , NULL
    , &m_pEft
    , &pErr);

// 정점 선언 객체 생성
D3DVERTEXELEMENT9
decl[MAX_FVF_DECL_SIZE]={0};
D3DXDeclaratorFromFVF(VtxD::FVF, decl);
m_pDev->CreateVertexDeclaration(decl, &m_pFVF)
```

```
// 상수 설정
m_pEft->SetMatrix("m_mtWld", &mtWld);
m_pEft->SetMatrix("m_mtViw", &mtViw);
m_pEft->SetMatrix("m_mtPrj", &mtPrj);
```

```
// 렌더링
m_pDev->SetVertexDeclaration(m_pFVF);
m_pEft->SetTechnique("Tech0");

UINT nPass=0;
m_pEft->Begin(&nPass, 0 );

for(UINT n=0; n<nPass; ++n)
{
    m_pEft->Pass(n);
    // m_pEft->BeginPass(n);// DX 2004이상
    m_pDev->DrawPrimitiveUP(...);
    // m_pEft->EndPass(n);
}

m_pEft->End();

// 정점, 픽셀 쉐이더 해제
m_pDev->SetVertexDeclaration(NULL);
m_pDev->SetVertexShader(NULL);
m_pDev->SetPixelShader(NULL);
```


여러 패스와 렌더링 상태 설정

```
technique Tech0
{
   pass P0
   {
      // 모델 렌더링
      LIGHTING    = FALSE;
      CULLMODE    = NONE;

      ALPHABLENDENABLE= False;
      ZWRITEENABLE    = TRUE;

      VertexShader = compile vs_1_1 VtxPrc0();
      PixelShader  = compile ps_1_1 PxlPrc0();
   }
   pass P1
   {
      // Glow 효과 렌더링
      LIGHTING    = FALSE;
      CULLMODE    = CW;

      ALPHABLENDENABLE= TRUE;
      SRCBLEND    = SRCALPHA;
      DESTBLEND   = DESTALPHA;
      ZWRITEENABLE    = FALSE;

      VertexShader = compile vs_1_1 VtxPrc1();
      PixelShader  = compile ps_1_1 PxlPrc1();
   }
}
```



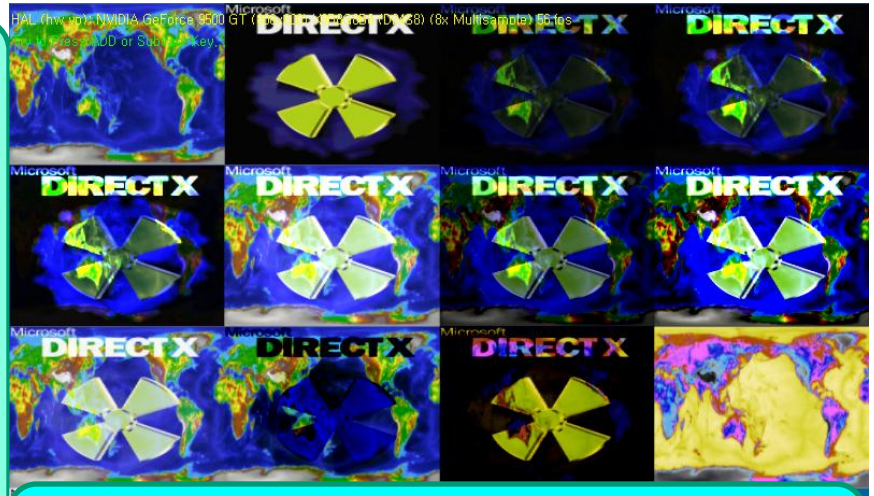
◆렌더링 머신의 상수 값은 대소문자 구분 없음

◆상태 머신 키워드:
고정 함수 파이프라인의 **"D3DRS_"**를 제외한 값

◆상태 머신 상수:
고정 함수 파이프라인 상수 "_" 이후

5. ID3DXEffect – Multi Texuture


```
// Multi Texturing Type
int     m_nMulti;

// 전역변수
texture m_TxDif0;
sampler SampDif0 = sampler_state
{
    Texture = <m_TxDif0>;
    MinFilter = LINEAR;
    MagFilter = LINEAR;
    MipFilter = LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};

texture m_TxDif1;
sampler SampDif1 = sampler_state
{
    Texture = <m_TxDif1>;
    MinFilter = LINEAR;
    MagFilter = LINEAR;
   ...
};

// 픽셀 쉐이더 프로세스
float4 PxlPrc0(float4 Tx0 : TEXCOORD0) : COLOR0
{
    float4 Out= 0;
    float4 t0 = tex2D( SampDif0, Tx0 );        // Sampling m_TxDif0
    float4 t1 = tex2D( SampDif1, Tx0 );        // Sampling m_TxDif1

    if(0 == m_nMulti)       Out = t0;
    else if(1 == m_nMulti)  Out = t1;
    else if(2 == m_nMulti)  Out = t0 * t1;          // Modulate
    else if(3 == m_nMulti)  Out = t0 * t1 * 2;      // Modulate 2x
...
    return Out;
}

technique Tech0
{
    pass P0
    {
        PixelShader  = compile ps_2_0 PxlPrc0();
    }
}
```




```
m_pDev->SetVertexDeclaration( m_pFVF);          // 정점선언

// 상수 연결
hr = m_pEft->SetTexture( "m_TxDif0", m_pTx0 );
hr = m_pEft->SetTexture( "m_TxDif1", m_pTx1 );

hr = m_pEft->SetTechnique("Tech0");
hr = m_pEft->Begin(NULL, 0 );
hr = m_pEft->Pass(0);

VtxDUV1 pVtx[4];
for(int i=0; i<=13; ++i)
{
    float   x = -1.f + float(i%4)/2.f;
    float   y =  1.f - float(i/4)/2.f;

    pVtx[0] = VtxDUV1(x + 0.f, y - 0.f,  0,  0, 0, D3DXCOLOR(1,0,0,1));
    pVtx[1] = VtxDUV1(x + .5f, y - 0.f,  0,  1, 0, D3DXCOLOR(0,1,0,1));
    pVtx[2] = VtxDUV1(x + .5f, y - .5f,  0,  1, 1, D3DXCOLOR(0,0,1,1));
    pVtx[3] = VtxDUV1(x+  0.f, y - .5f,  0,  0, 1, D3DXCOLOR(0,0,1,1));

    hr = m_pEft->SetInt("m_nMulti", i);
    hr = m_pDev->DrawPrimitiveUP( D3DPT_TRIANGLEFAN, 2, pVtx,
sizeof(VtxDUV1));
}
m_pEft->End();

m_pDev->SetVertexShader( NULL);
m_pDev->SetPixelShader( NULL);
```

5. ID3DXEffect - 정리

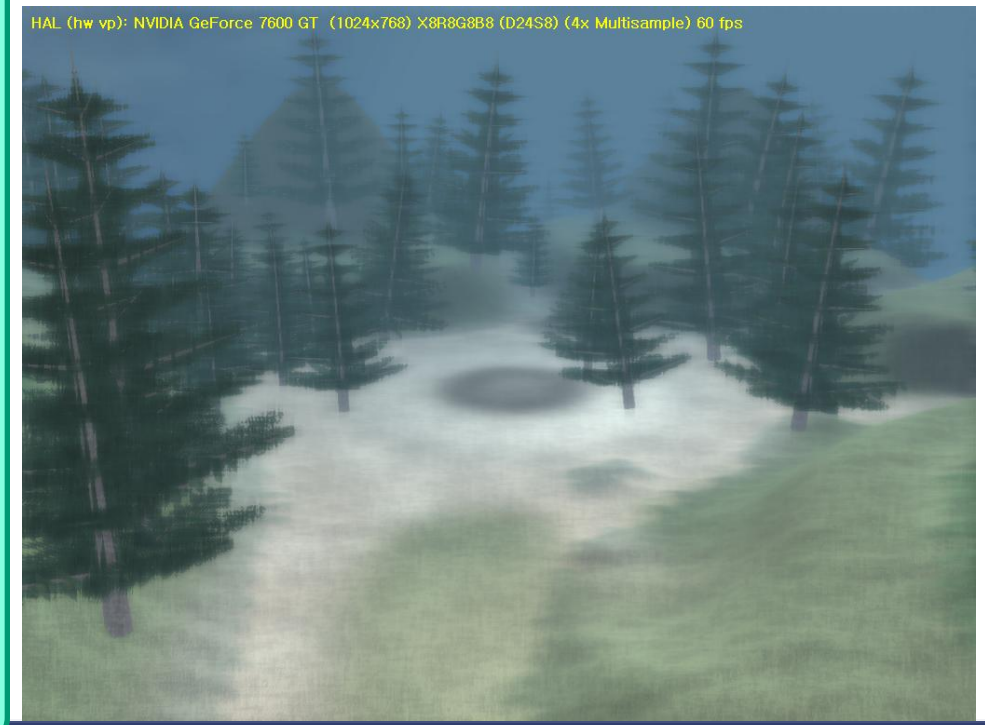
- HLSL
  - 저 수준 또는 고 수준 언어로 쉐이더 함수들을 작성
  - 텍스처의 경우 샘플러 정의를 통해서 Addressing, Filtering 지정
  - 테크닉(Technique)과 Pass 설정
    - Pass안에서 정점 쉐이더 객체, 픽셀 쉐이더 객체 컴파일 설정
    - 렌더링 상태 머신 상수 설정

- 렌더링 준비 단계
  - ID3DXEffect 객체 생성 ➔ D3DXCreateEffec…() 함수를 통해서 HLSL 코드를 컴파일 하고 ID3DXEffect 객체 생성
  - 정점 쉐이더 선언 객체 생성: pDev->CreateVertexShaderDeclaration();

- 렌더링
  - 정점 선언 객체 설정: pDevice->SetVertexDeclaration();
  - HLSL extern 변수 설정: pEffect->Set{Vector|Matrix|Texture|...}("전역 변수 이름", 설정 값 주소);
  - 테크닉 설정: pEffect->SetTechnique();
  - 패스 설정: pEffect->Pass();
  - 렌더링
  - 렌더링 해제:
    - pDev->SetVertexDeclaration(NULL);
    - pDev->SetVertexShader(NULL);
    - pDev->SetPixelShader(NULL);

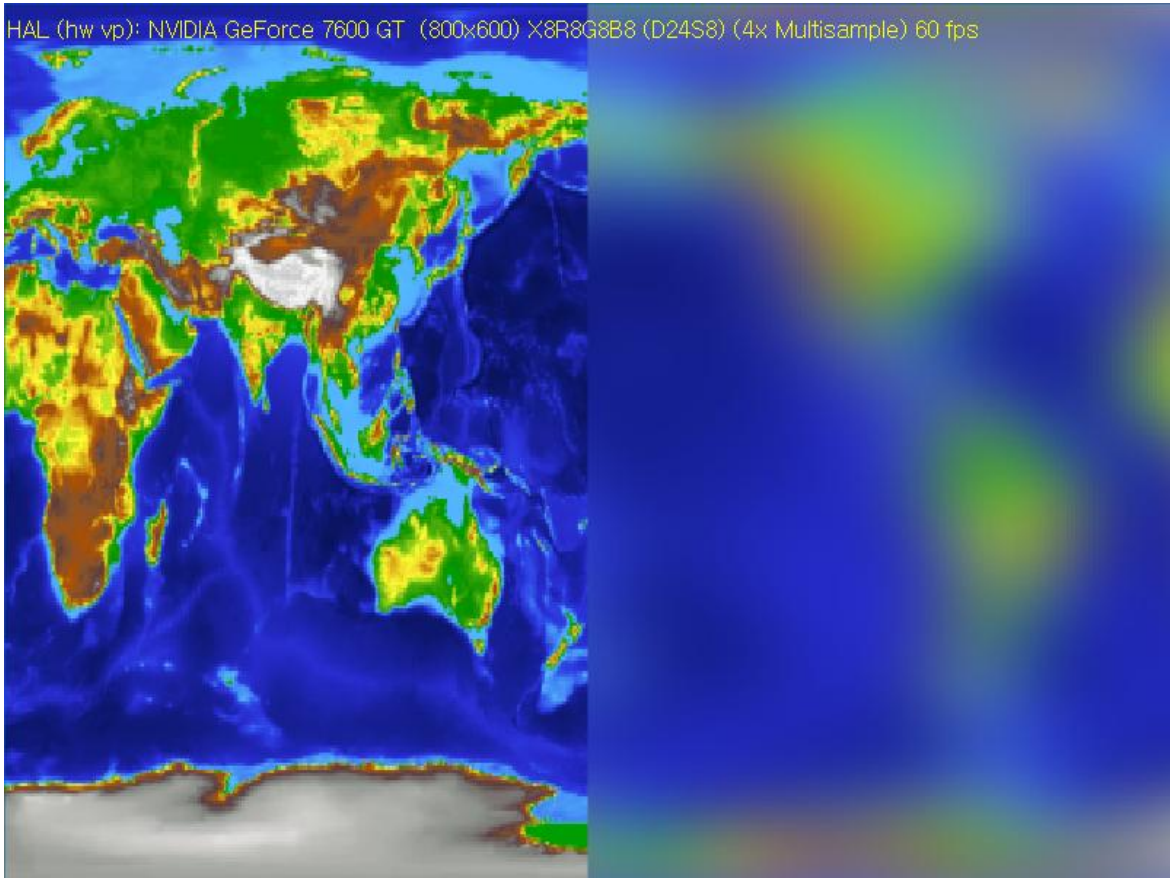
- Blur는 인접한 색상과 혼합이므로 단순하게 for문으로 텍스처 좌표를 이동하면서 인접한 색상을 얻고 이를 혼합
- 문제점: 명령어 slot의 제약이 있어 for 문의 범위가 커지면 쉐이더 컴파일에서 Error

```
float4 PxlPrc1( float4 Dif : COLOR0,
                float2 Tx0 : TEXCOORD0
                ) : COLOR
{
   int     i=0; int iMax=4; float4  Out= 0.f;
   for(i=-iMax; i<=iMax; ++i)  // 범위를 크게 하면 명령어 슬롯 제한 에러 발생
   {
      float2 Tx    = Tx0;
      Tx.x     += (i *g_fDev)/1024.f;
      float4  d = tex2D(sampTex, Tx);
      float    e = i*i;
      e = -e/16.0f;
      Out += d * 1.f* exp( e );
   }
   for(i=-iMax; i<=iMax; ++i)  // 범위를 크게 하면 명령어 슬롯제한 에러 발생
   {
      float2 Tx    = Tx0;
      Tx.y     += (i *g_fDev)/1024.f;
      float4  d = tex2D(sampTex, Tx);
      float    e = i*i;
      e = -e/16.0f;
      Out += d * 1.f* exp( e );
   }
   Out   *=0.3f;
   float    d = Out.r * 0.288 + Out.g * 0.588 + Out.b * 0.114f;
//  d    = pow(d, 1.f);
   d    *=.4f;
   Out.r    = d* 1.f; Out.g    = d* 1.f;  Out.b    = d* 1.f; Out.a     = 1.f;
   return Out;
}
```



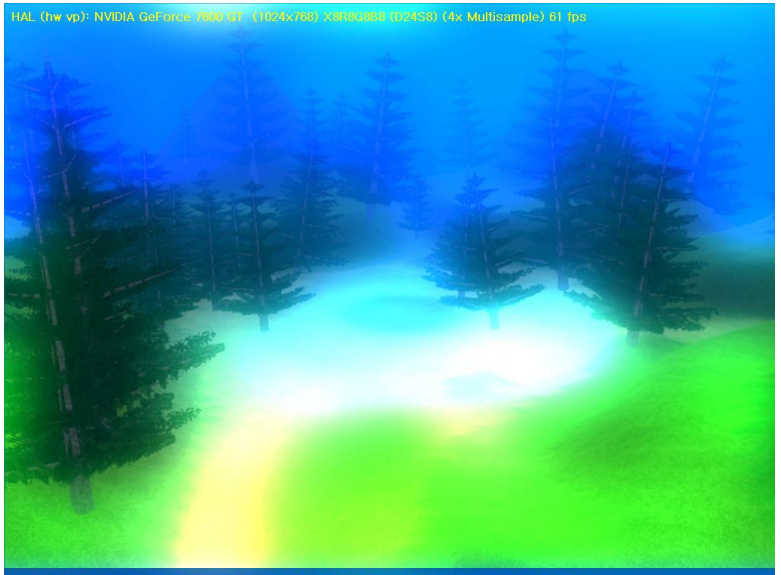
- 고정 함수에서 여러 번 렌더링 - HLSL의 패스(Pass)와 렌더링의 횟수가 많아지지만 HLSL에서 한 번에 처리하는 것 보다 효과적
- 렌더링 타깃의 해상도가 낮아도 잘 표현

```
SvsOut VtxProc0() ...

float4 PxlProc0(SvsOut In) : COLOR0...
float4 PxlBlurX(SvsOut In) : COLOR0...
float4 PxlBlurY(SvsOut In) : COLOR0...

technique Tech
{
   pass P0
   {
      VertexShader = compile vs_1_1 VtxProc0();
      PixelShader  = compile ps_2_0 PxlProc0();
   }

   // Blur X
   pass P1
   {
      VertexShader = compile vs_1_1 VtxProc0();
      PixelShader  = compile ps_2_0 PxlBlurX();
   }

   // Blur Y
   pass P2
   {
      VertexShader = compile vs_1_1 VtxProc0();
      PixelShader  = compile ps_2_0 PxlBlurY();
   }
};
```

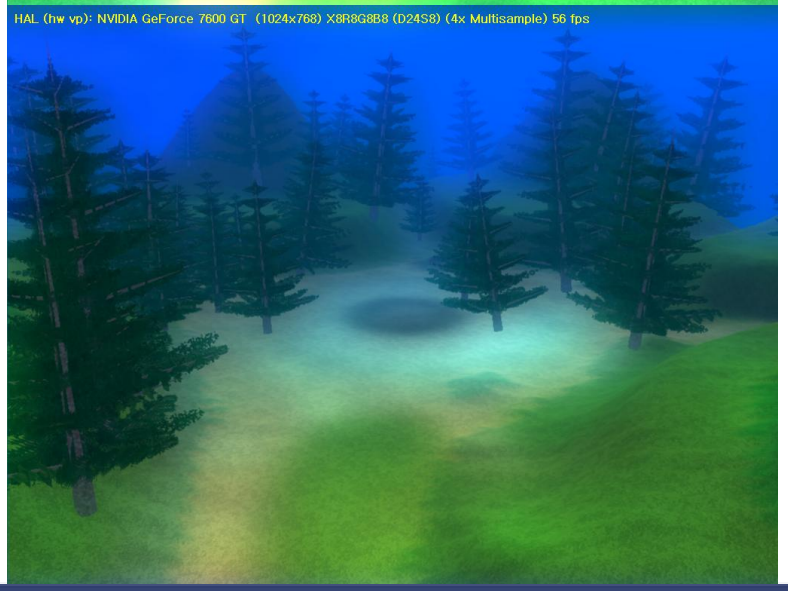


- 높이 맵에 적용

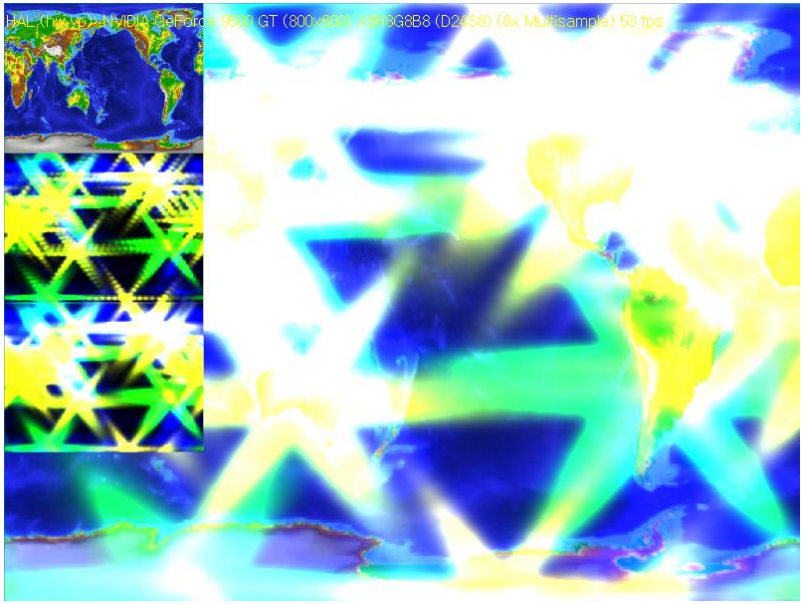
```
float4 PxlProc0(SvsOut In) : COLOR0
{
    float4  Out=0;
    float4 t0 = tex2D(smpDif, In.Tex);

    t0 = pow(t0,5)*50.f;
    Out = t0;
    Out.w = 1;

    return Out;
}
```



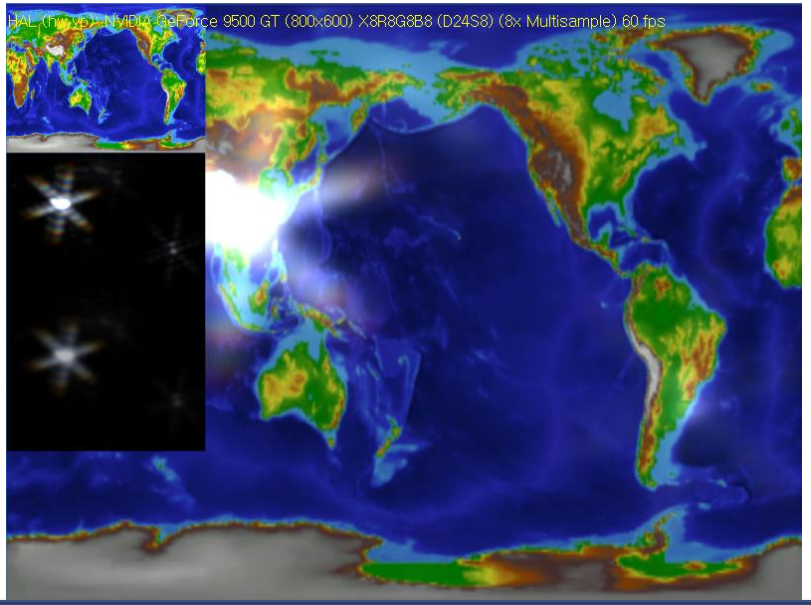

```
//렌더 타겟 사이즈
m_fTxW = 256;
m_fTxH = m_fTxW;

// HLSL 상수
static float fBgn = 7;
static float fIns=.115f;
static float fInc=1.2f;
```



```
// 고 휘도(luminance) 검출
float4 PxlProc0(SvsOut In) : COLOR0
{
    float4  Out=0;
    float4 t0 = tex2D(smpDif, In.Tex);

    t0 *= 1.8f;
    t0 = pow(t0,8);

    if(t0.r+t0.b+t0.b<2.f)
        t0.rgb=0.f;

    Out = t0;
    Out.w = 1;

    return Out;
}
```

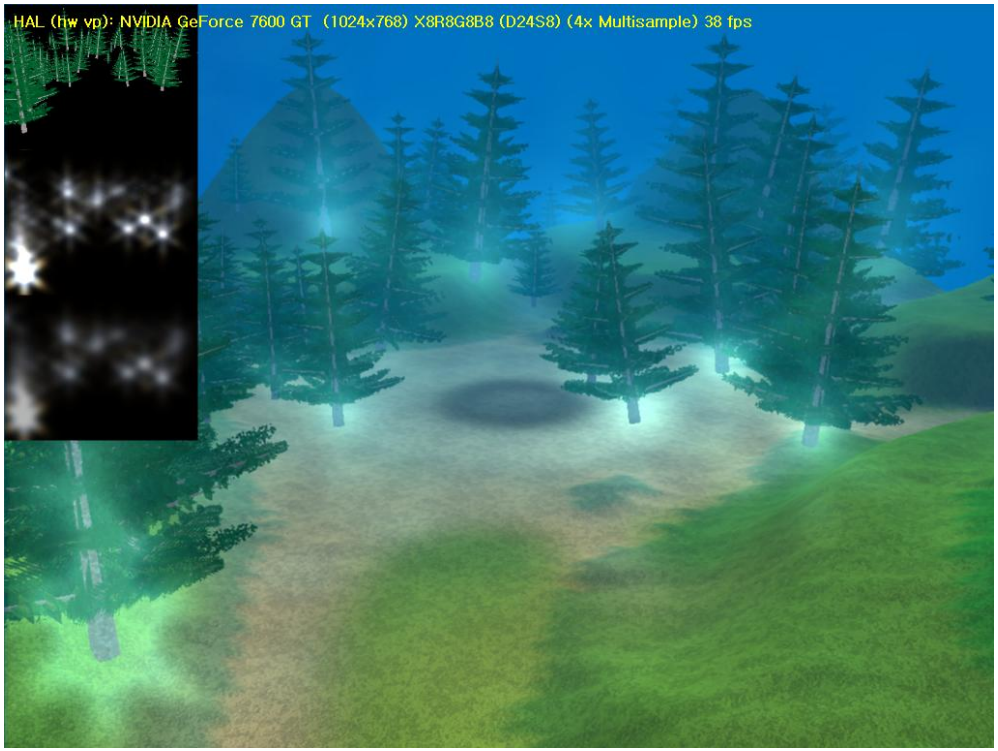


```
// Refract Red
float4 PxlStarR(SvsOut In) : COLOR0
{
    float4  Out=0;
    float   u = 0;
    float   v = 0;

    for(int i=0; i<MAX_SAMP;  ++i)
    {
        u = In.Tex.x + m_StarVal[i].x*1.25f;
        v = In.Tex.y + m_StarVal[i].y*1.25f;
        Out.r += tex2D(smpDif, float2(u,v)) * m_StarVal[i].z;
    }

    Out *=m_StarPow;
    Out.w = 1;
    return Out;
}
```



```
// Blur X
float4 PxlBlurX(SvsOut  In) : COLOR0
{
   float4  Out=0;
   float2  uv = 0;

   for(float i=-fBgn; i<=fBgn; i+=1)
   {
      uv = In.Tex + float2((i*fInc+1.5)/m_TexW, 0);
      Out += tex2D(smpDif, uv) * exp( -i*i * fDelta);
   }

   Out *=fIns;
   Out.w = 1;
   return Out;
}
```

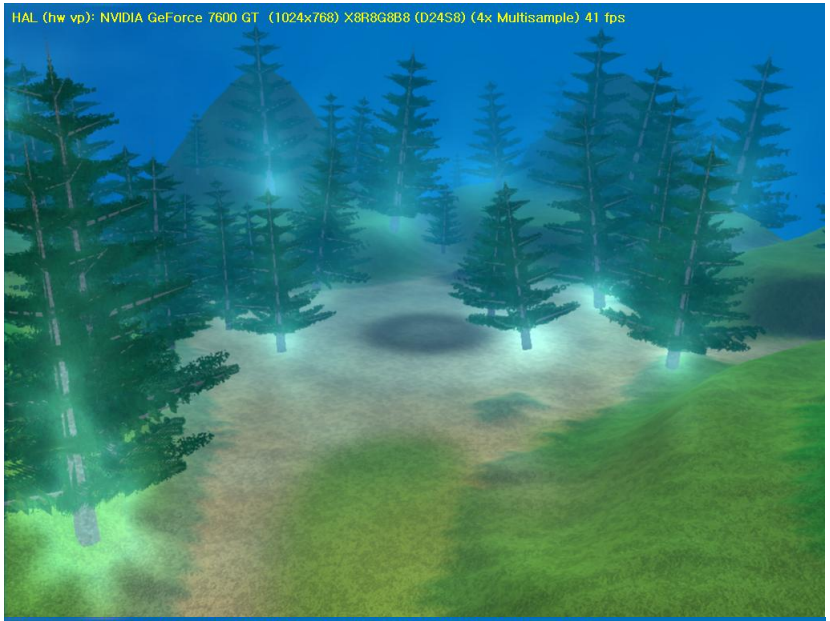
```
// Rendering Target
IrenderTarget*  m_pTrnd0;  // Rendering Target Texture for Scene
IrenderTarget*  m_pTrnd1;  // Rendering Target Texture for Scene

IrenderTarget*  m_pTrndX;  // Rendering Target Texture for Blur X
IrenderTarget*  m_pTrndY;  // Rendering Target Texture for Blur Y

IrenderTarget*  m_pTrndS;  // Cross Texture All
IrenderTarget*  m_pTrndC[6];    // Cross Texture

float         m_fTxW;           // Blur Texture Width
float         m_fTxH;           // Blur Texture Heigh
```

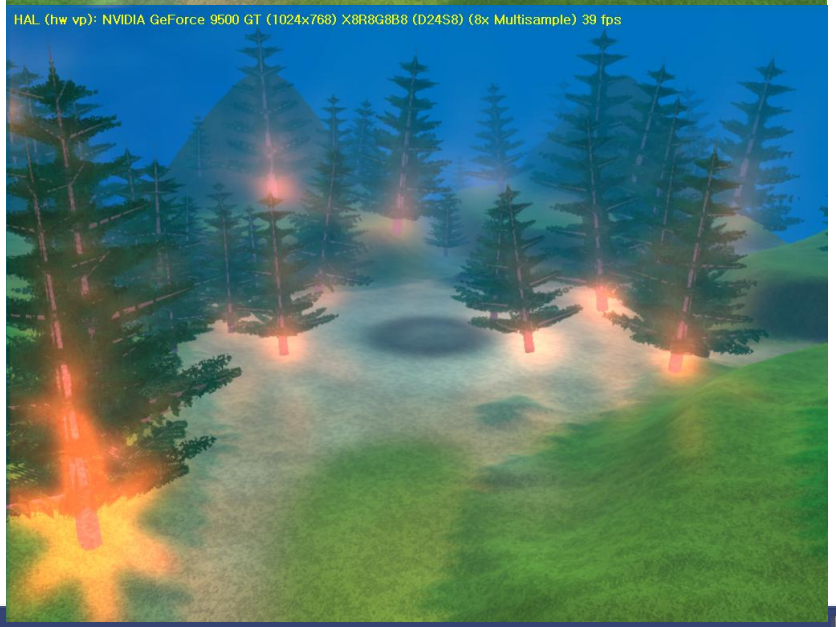


1. **Rendering Target**에 장면을 그린다.
2. 축소된 텍스처에 **Luminescence**가 강한 부분을 그린다.
3. 축소된 텍스처를 뭉갠다.
4. **Cross**를 만든다.
5. 모든 스타를 하나로 합친다.
6. 합친 텍스처 뭉개기
7. 장면 텍스처와 혼합

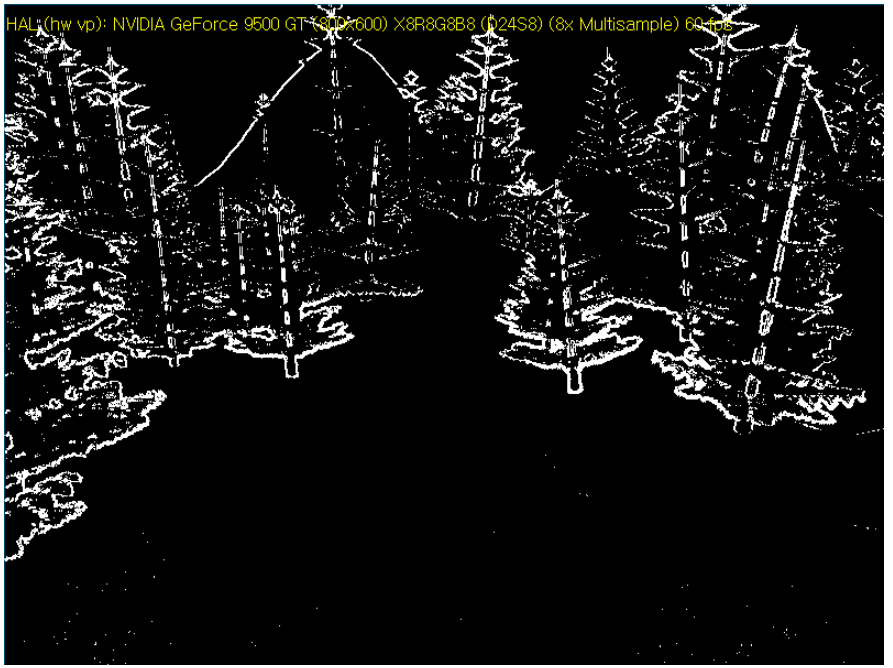
```
// Merge All Star...
float4 PxlStarAll(SvsOut In) : COLOR0
{
    float4  Out=0;
    Out += tex2D(s0, In.Tex);
    Out += tex2D(s1, In.Tex);
    Out += tex2D(s2, In.Tex);
    Out += tex2D(s3, In.Tex);
    Out += tex2D(s4, In.Tex);
    Out += tex2D(s5, In.Tex);

    Out = saturate(Out);
    Out.w = 1;
    return Out;
}
```

```
float4 PxlAll(SvsOut In) : COLOR0
{
    float4  Out=0;
    float4  t0 = tex2D(s0, In.Tex);
    float4  t1 = tex2D(s1, In.Tex);

    // Out = t0*1.5f + t1*.4f*float4(.6, 1., .8, 1);
    Out = t0*1.5f + t1*.4f*float4(2.6, 0.7, .2, 1);
    Out.w = 1;
    return Out;
}
```





```
float4 PxlProc(SvsOut In) : COLOR0
{
   float4  Out=float4(0,0,0,1);
   float2  uv;
   float3  txI;
   float3  txC;
   float D=0;

   uv  = In.Tex;
   txI = tex2D(smpDif, uv);

   for(int j=-1; j<=1; ++j)
   {
      for(int i=-1; i<=1; ++i)
      {
         if(!(0==i && 0==j))
         {
            uv = In.Tex + float2(i/m_TxW, j/m_TxH);
            txC = tex2D(smpDif, uv);
            D += distance(txI, txC);
         }
      }
   }

   if(D>0.2)
      Out = float4(1,1,1,1);

   Out.w = 1.f;
   return Out;
}
```



외곽선: 인접한 픽셀과 색상 차이가 크면 경계에 있는 픽셀

색상 차이 = distance(pixel1.rgb – pixel2.rgb);

※ 색상 rgb를 벡터 xyz로 계산