2009, Copyright All rights reserved

3D Game Programming 41
- Vertex Shader

afewhee@gmail.com

0. 목차

GaLic. Heesung Oh
2009, Copyright All rights reserved

- Vertex Shader 기초
  - Shader 개요
  - Vertex Shader
  - Vertex Shader Coding 기초
- Vertex Shader 적용
  - Simple Vertex Shader
  - Transform
  - Texture Coordinate
  - Lighting
  - Fog
- Vertex Shader 응용
  - Cartoon Shading
  - Depth Encoding
  - Vertex Shader Effect 객체

1.1 Shader 개요

2009, Copyright All rights reserved

- Shader
  - Display Processor를 대상으로 한 명령어들의 집합 ➔ GPU의 렌더링 파이프 라인에 대한 Program

- 고정 함수 파이프 라인(Fixed-Function Pipe Line)과 비교
  - Fixed Pipe Line: 렌더링 머신의 그래픽 파이프 라인에 대한 상수 값을 함수를 통해서 설정, 출력 결과를 조정
  - 쉐이더: 고정 함수 파이프 라인에서 정점의 처리과정과 픽셀 처리 과정의 일부를 프로그래머가 저 수준(Low Level), 또는 고 수준(High Level) 언어를 이용해서 제어

- 쉐이더 종류
  - 정점 쉐이더: Vertex Shader ➔ (Programmable Vertex Processing)
    - 고정 함수 파이프 라인의 일부인 정점의 T&L(Transform and Lighting), Fog 등을 프로그램
  - 픽셀 쉐이더: Pixel Shader ➔ (Programmable Pixel Processing)
    - 픽셀 처리 과정의 Sampling(Filtering, Addressing), Texturing(Blending) 등을 프로그램

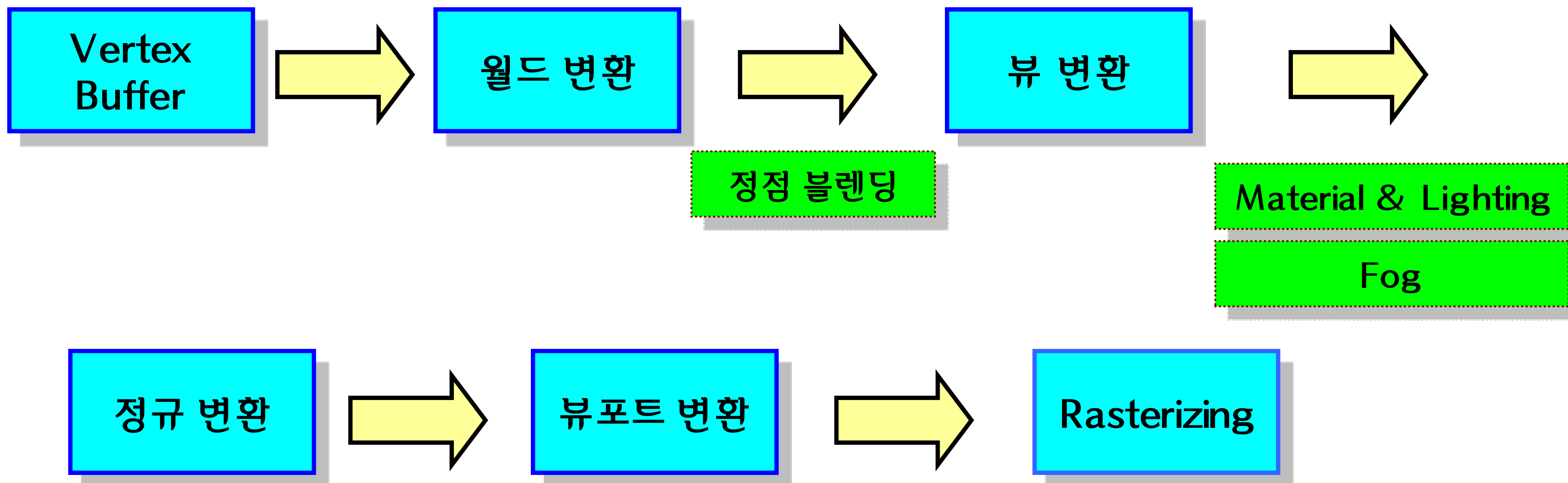
- 쉐이더 사용방법
  - 쉐이더(Shader): 저 수준 언어를 사용 ➔ Assembly 형식
  - HLSL(High Level Shading Language): 고 수준 언어를 사용 ➔ C 언어와 같은 고수준 형식
  - 혼용: HLSL은 저 수준 언어를 사용할 수 있음

GaLic. Heesung Oh
2009, Copyright All rights reserved

- Direct3D가 지원하는 파이프 라인 구조

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 고정 함수 파이프 라인 T&L



- Vertex Shader
  - 고정 함수 파이프 라인의 T&L을 프로그래밍

1.2 Vertex Shader

2009, Copyright All rights reserved

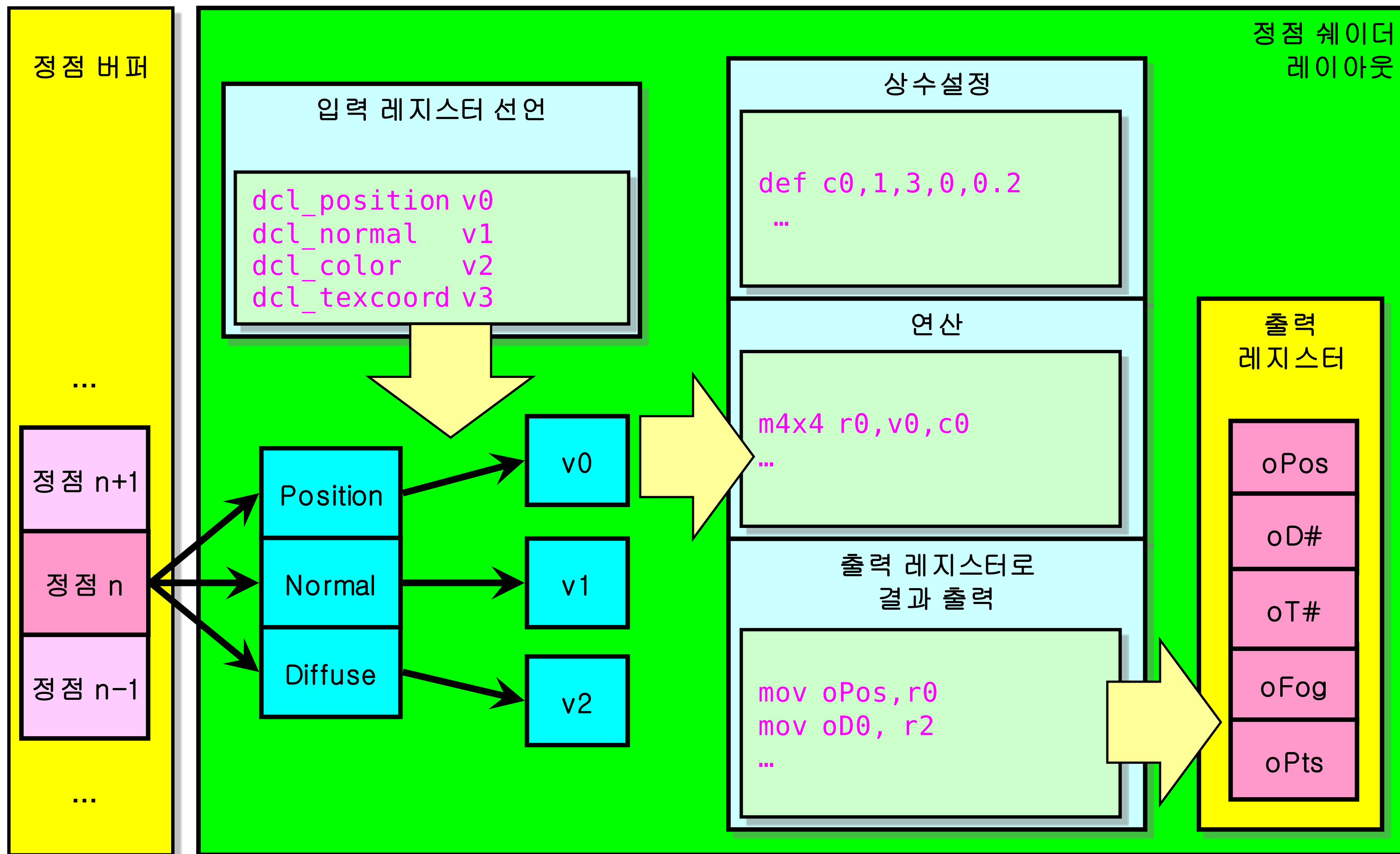
- Assembly 형식의 저 수준 정점 쉐이더 프로그램과 c언어의 Inline Assembly 예

```
; Simple Vertex Shader

vs_1_1

dcl_position    v0

dcl_color       v1


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

```
// In line Assembly
#include<stdio.h>
void main()
{
    char*  s1 = "Hello";
    char*  s2 = "World";
    char*  f  = "%s %s\n";
    _asm
    {
        mov eax, s2;
        push eax;
        mov ebx, s1;
        push ebx;
        mov ecx, f;
        push ecx;
        call printf;
        pop ecx;
        pop ebx;
        pop eax;
    }
}
```

GaLic.Heesung Oh
2009, Copyright All rights reserved

● 정점 쉐이더 레이아웃 (Vertex Shader Layout)



1.2 Vertex Shader

2009, Copyright All rights reserved

- 저 수준 정점 쉐이더 프로그램 방법
  - 쉐이더 명령어 버전 선언
    - vs_1_1 ➔ 정점 쉐이더 명령어 버전은 1.1
  - 입력 레지스터 선언(지정)
    - dcl_position v0 ➔ 입력 정점의 위치를 v0 레지스터로 선언(저장)
    - dcl_normal v1 ➔ 입력 정점 법선을 v1 레지스터로 선언(저장)
    - dcl_color v2 ➔ 정점의 diffuse 색상을 v2 레지스터로 선언(저장)
    - dcl_texcoord v3 ➔ 정점의 텍스처 좌표를 v3 레지스터로 선언(저장)
  - 상수 레지스터 설정
    - def c0, 1.0, 3.0, 0.0, 2
    - def c12 , 1.0, 0.0, 0.0, 1
  - 정점 처리에 대한 연산
    - 변환: m4x4 r0, v0, c0
    - 연산의 과정에서 상수 레지스터(c#)은 임시 변수로 사용 불가
    - 임시 변수는 임시 변수 레지스터(r#) 이용
  - 출력 레지스터에 결과 저장
    - mov oPos, r0
    - mov oDo, v2
    - mov oFog, r2 ...

GaLic.Heesung Oh
2009. Copyright All rights reserved

정점 쉐이더 이용 방법

쉐이더 코드와 객체 생성 단계

- 어셈블리 형식의 저 수준 쉐이더 코드를 Text로 작성
- 저 수준 쉐이더 코드 컴파일: D3DXAssembleShader()
- 정점 쉐이더 객체 생성: pDevice->CreateVertexShader()
- 정점 선언 객체 생성: pDevice->CreateVertexDeclaration()

렌더링 단계

- 렌더링 머신(장치: 디바이스)에게 정점 쉐이더 사용을 통보
- 장치에게 정점의 형식을 알 수 있도록 정점 선언 객체를 지정
- 쉐이더 상수 설정
- 렌더링
- 장치의 정점 쉐이더 사용과 정점 선언 해제

1.2 Vertex Shader

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 저 수준 정점 쉐이더 컴파일(Assemble)
  - 저급 언어인 어셈블리어 컴파일 담당은 어셈블러
  - 저수준 쉐이더 컴파일은 DirectX의 D3DXAssembleShader() 함수

```
Ex)
DWORD dwFlags = 0;
#if defined( _DEBUG ) || defined( DEBUG )
    dwFlags |= D3DXSHADER_DEBUG;
#endif

LPD3DXBUFFER    pShd    = NULL;
LPD3DXBUFFER    pErr    = NULL;
INT     iLen    = strlen(sShader);

hr = D3DXAssembleShader(sShader, iLen, NULL, NULL, dwFlags, &pShd, &pErr);

// Error
if ( FAILED(hr) )
{
    if(pErr)
    {
        MessageBox( GetActiveWindow(), (char*)pErr->GetBufferPointer(), "Err", 0);
        pErr->Release();
    }
    return -1;
}
```

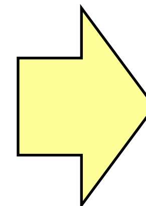
  - 문법 (Syntax) 에러 문자열은 D3DXAssembleShader() 함수의 에러 반환 객체의 버퍼에 대한 내용을 char형 포인터로 변환해서 사용
    - (char*)pErr->GetBufferPointer()

1.2 Vertex Shader

2009, Copyright All rights reserved

- 정점 쉐이더 객체 (IDirect3DVertexShader9)
  - 컴파일한 쉐이더 명령어를 사용하기 위한 객체
  - pDevice->CreateVertexShader(...);
- 정점 선언(Vertex Declaration: IDirect3DVertexDeclaration9) 객체
  - 정점의 구조를 전달 ➔ 고정 함수 파이프 라인의 SetFVF()와 비슷한 기능
  - 이 객체를 생성하기 위해 D3DVERTEXELEMENT9 구조체가 사용되는데 특별한 경우가 아니라면 고정 함수 파이프 라인의 FVF를 이용해서 구조체 내용을 채움 ➔ D3DXDeclarationFromFVF() 함수 이용

```
D3DVERTEXELEMENT9 decl[] =
{
   {0,  0, D3DDECLTYPE_FLOAT3,   D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0},
   {0, 12, D3DDECLTYPE_FLOAT3,   D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL,  0},
   {0, 24, D3DDECLTYPE_FLOAT2,   D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0},
   {0, 32, D3DDECLTYPE_FLOAT2,   D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 1},
   D3DDECL_END()
};
```



```
D3DVERTEXELEMENT9
vertex_decl[MAX_FVF_DECL_SIZE];

memset(vertex_decl, 0, sizeof(vertex_decl));

D3DXDeclaratorFromFVF(VtxD::FVF, vertex_decl);
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 정점 쉐이더 객체와 정점 선언 객체 생성 예

```
LPD3DXBUFFER   pShd   = NULL;
LPD3DXBUFFER   pErr   = NULL;

//Shader Compile
hr = D3DXAssembleShader(..., dwFlags, &pShd, &pErr);

if ( FAILED(hr) ) // Error
{
    if(pErr)
    {
        MessageBox( GetActiveWindow(), (char*)pErr->GetBufferPointer(), "Err", 0);
        pErr->Release();
    }
    return -1;
}

// Create Vertex Shader
hr = m_pDev->CreateVertexShader( (DWORD*)pShd->GetBufferPointer() , &m_pVs);

if ( FAILED(hr) ) // Error
    return -1;

pShd->Release();

// Create Vertex Declaration
D3DVERTEXELEMENT9 vertex_decl[MAX_FVF_DECL_SIZE]={0};

D3DXDeclaratorFromFVF(VtxD::FVF, vertex_decl);

if(FAILED(m_pDev->CreateVertexDeclaration( vertex_decl, &m_pFVF )))
    return -1;
```

1.2 Vertex Shader

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 렌더링 단계
  - 렌더링 머신(장치: 디바이스)에게 정점 쉐이더 사용을 통보
    - pDevice->SetVertexShader()
  - 장치에게 정점의 형식을 알 수 있도록 정점 선언 객체를 지정
    - pDevice->SetVertexDeclaration() == SetFVF()
  - 쉐이더 상수 설정
    - pDevice->SetVertexShaderConstantF(상수 레지스터 이름(시작), 변수 주소, 크기) ➔ 고정 함수 파이프 라인의 SetTransform(), SetLighting() 함수 등과 비슷하게 GPU의 상수 레지스터 설정
  - 렌더링
    - pDevice->DrawPrimitive()
  - 장치의 정점 쉐이더 사용과 정점 선언 해제
    - pDevice->SetVertexShader(NULL)
    - pDevice->SetVertexDelcaration(NULL)

1.2 Vertex Shader

2009, Copyright All rights reserved

- 상수 레지스터 설정 시 주의 사항
  - 상수 레지스터 범위 인지 : c0 ~ c95
  - c를 생략하고 숫자만 전달:
    - c10에 변수의 값을 설정할 경우 pDevice->SetVertexShaderConstantF(10, …, …)

  - 변수 주소: 대부분 float* 형으로 캐스팅

  - 크기: CPU는 자료의 처리가 INT형이 기본 이듯, 쉐이더는 float4(float 4개의 연속 메모리)가 기본
    - r, g, b, a 인 float형 4개를 전달할 때 크기는 1
    - x, y, z인 float형 3개를 전달할 때도 크기는 1
    - 행렬을 전달할 경우 행렬은 float4의 4개 이므로 크기는 4

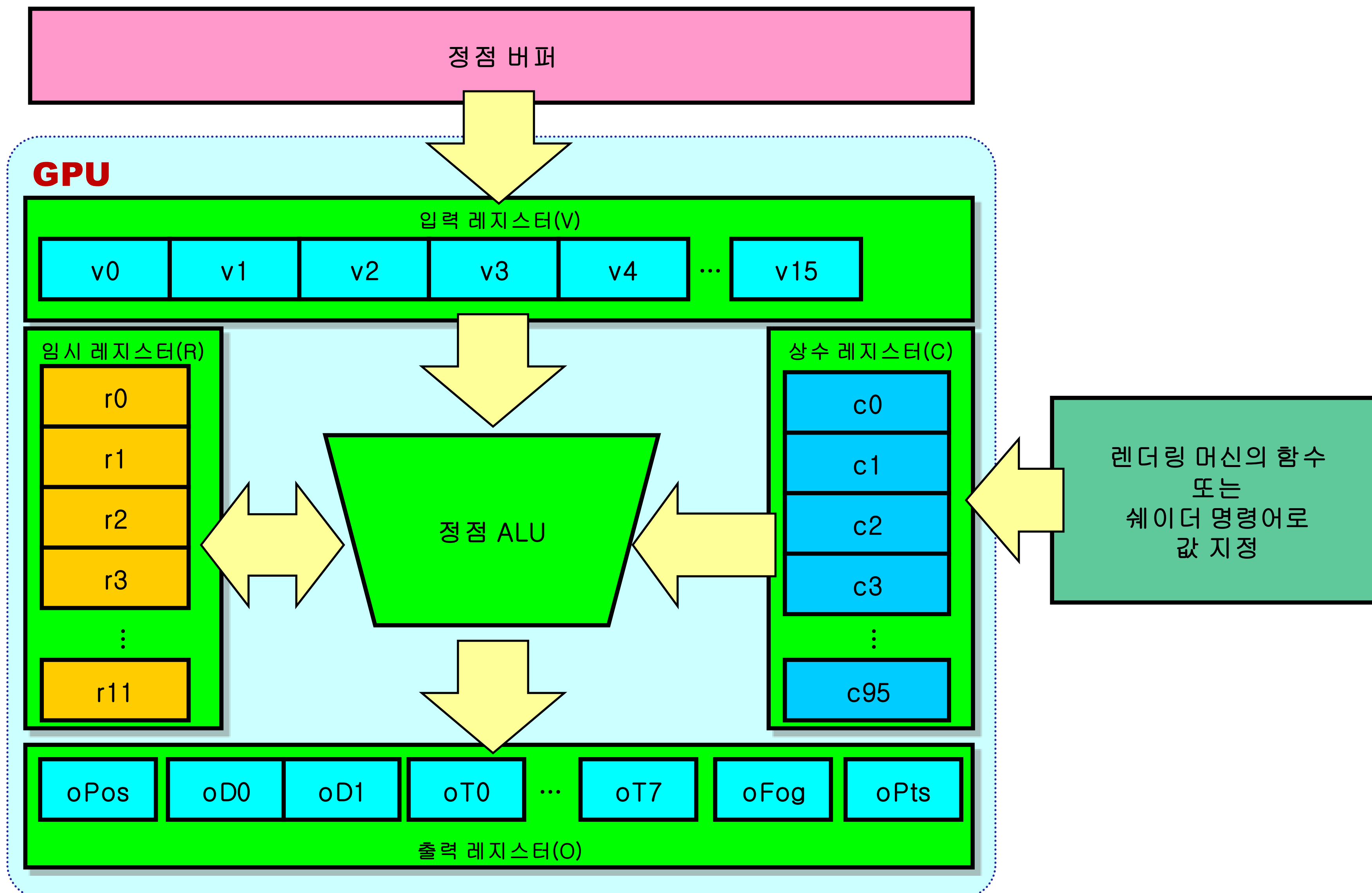
  - 상수 레지스터 사용 범위
    - 행렬을 사용할 때와 같이 크기를 1보다 큰 값을 전달하면 해당 상수 레지스터에서부터 크기 만큼 상수 레지스터 점유
    - Ex)
    - m_pDev->SetVertexShaderConstantF(10, (FLOAT*)&mtVP, 4);
      ➔ c10, c11, c12, c13을 mtVP 변수 값을 저장하는 데 사용

  - 행렬을 전달할 때 주의점
    - 쉐이더는 오른손 좌표계로 연산.
    - 변환에 사용되는 행렬은 전부 전치(Transpose) 함.
    - 여러 행렬의 곱을 연결할 경우 계산의 결과를 전달하는 것이 유리

      Ex)
```
D3DXMATRIX mtVP = mtWld * mtViw * mtPrj;
D3DXMatrixTranspose( &mtVP , &mtVP );
…
m_pDev->SetVertexShaderConstantF(10, (FLOAT*)&mtVP , 4);
```

GaLic.Heesung Oh
2009, Copyright All rights reserved

- **정점 쉐이더 가상 머신**



1.3 Vertex Shader Coding 기초

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 레지스터
  - 4 종류: 입력, 임시, 상수, 출력 레지스터
  - "레지스터+index" 가 이름
  - ex) v12 ➔"v12" 자체가 레지스터 이름.
- 입력 레지스터 (Input Register)
  - v로 시작.   v0 ~ v15;
  - 입력된 정점의 위치, 법선 벡터, Diffuse, Specular, 텍스처 좌표 등을 분해해서 저장
- 임시 레지스터 (Temporary Register)
  - r로 시작.   r0 ~ r12, ~ r32(vs_3_0)
  - 연산의 결과를 임시로 저장하는 데 사용
- 상수 레지스터 (Constant Float Register)
  - c로 시작.   c0 ~ c95, ~ c255(vs_2_0)
  - def: 상수 설정에 사용
  - 외부에서 사용자가 값을 설정할 수 있음 ➔pDev->SetVertexConstantF()
  - 정수 형 상수는 i로 시작 ➔ pDev->SetVertexConstantI()로 설정
  - bool 형 상수는 b로 시작 ➔ pDev->SetVertexConstantB()로 설정
- 출력 레지스터
  - o(영문자 small 'o')로 시작
  - oPos : 위치에 대한 출력 레지스터 ➔ 전체 쉐이더 코드에서 반드시 지정되어야 함
  - oD0, 0D1: 색상 레지스터. oD0 Diffuse, oD1-Specular 색상 출력 레지스터
  - oT0 ~ oT7: 텍스처 좌표 레지스터
  - oFog: 안개 효과에 대한 레지스터
  - oPts: 포인트 사이즈에 대한 레지스터
- 기타 레지스터
  - Address Register:
  - loop Counter Register: 루프의 반복 횟수를 저장하는 계수기

1.3 Vertex Shader Coding 기초 - 주요 명령어

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 쉐이더 버전:
  vs_1_1, vs_1_2, …

- 입력 레지스터 선언
  - dcl_usage ➔ usage: D3DDECLUSAGE 의 "D3DDECLUSAGE"를 제거한 것과 동일
  - dcl_position: Vertex Position
  - dcl_normal: Vertex Normal Vector
  - dcl_color
    - dcl_color0: Diffuse Color
    - dcl_color1: Specular Color
  - dcl_texcoord
    - dcl_texcoord{#}
    - Vertex Texture Coordinate

Ex)

```
dcl_position v0        ; 입력 레지스터 v0에 입력한 정점의 위치를 저장
dcl_normal v1          ; 입력 레지스터 v1에 정점의 법선 벡터 저장
dcl_color0 v2          ; 입력 레지스터 v2에 정점의 Diffuse 색상 저장
dcl_color1 v3          ; 입력 레지스터 v3에 정점의 Specular 색상 저장
dcl_texcoord0 v4       ; 입력 레지스터 v4에 정점의 0번째 텍스처 좌표 저장
dcl_texcoord1 v5       ; 입력 레지스터 v5에 정점의 1번째 텍스처 좌표 저장
```

1.3 Vertex Shader Coding 기초 - 주요 명령어

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 상수 레지스터(Constant Register) 설정
  - 종류: float, int, bool
  - 정의 : Def 상수 레지스터 이름, value 0, value 1, value 2, value 3

Ex)

```
def c0, -0.5, 0.5, 0, 0.6
def c1, 0,0,0,0 def c2, 1,1,1,1
```

  - 상수 레지스터는 외부에서 설정 가능
    - pDevice->SetVertexConstant{F|I|B}("c를 제외한 인덱스", 변수 주소, 크기);

Ex)

```
D3DXVECTOR3 p(x,y,z);
pDevice->SetVertexConstantF(10, (float*)&p, 1);
// c10에 float형 상수 설정
```

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 연산 명령어 문법
  - Assembly의 Mnemonic 형태
  - operator dest, source1, source2, source3
  - 모든 연산은 float4를 x, y, z, w 또는 r, g, b, a로 나누어서 연산
  - dest operand는 임시(r#), 출력(o…)레지스터만 가능
  - dest는 상수(c#), 입력(v#)는 불가
- 주석: ';' 또는 '//'

1.3 Vertex Shader Coding 기초 - 주요 명령어

2009, Copyright All rights reserved

- dp4: 4개 항에 대한 dot product
  - dp4 r2, r0, r1;
    r2.w=r0.x * r1.x + r0.y * r1.y + r0.z * r1.z + r0.w * r1.w;
    and r2.x=r2.y=r2.z=r2.w;

- dp3: 3개 항에 대한 dot product
  - r2, r0, r1;
    r2.w = r0.x * r1.x + r0.y * r1.y + r0.z * r1.z;
    and r2.x=r2.y=r2.z=r2.w;

- m4x4, m4x3: 행렬 벡터 연산 etc m3x2, m3x3
  - m4x4 r0.xyzw, r1, c0은 아래의 dp4를 4번 연산한 것과 동일
    dp4 r0.x, r1, c0
    dp4 r0.y, r1, c1
    dp4 r0.z, r1, c2
    dp4 r0.w, r1, c3

- m4x3 r0.xyz, r1, c0 은 아래의 dp4를 3번 연산한 것과 동일
  dp4 r0.x, r1, c0
  dp4 r0.y, r1, c1
  dp4 r0.z, r1, c2

1.3 Vertex Shader Coding 기초 - 주요 명령어

2009, Copyright All rights reserved

- mov: copy
  - mov r2, r0;
    r2.x = r0.x;
    r2.y = r0.y;
    r2.z = r0.z;
    r2.w = r0.w;

- max: 최대값
  - max r2, r0, r1
    r2.x=(r0.x >= r1.x) ? r0.x : r1.x;
    r2.y=(r0.y >= r1.y) ? r0.y : r1.y;
    r2.z=(r0.z >= r1.z) ? r0.z : r1.z;
    r2.w=(r0.w >= r1.w) ? r0.w : r1.w;

- min: 최소값
  - min r2, r0, r1
    r2.x=(r0.x < r1.x) ? r0.x : r1.x;
    r2.y=(r0.y < r1.y) ? r0.y : r1.y;
    r2.z=(r0.z < r1.z) ? r0.z : r1.z;
    r2.w=(r0.w < r1.w) ? r0.w : r1.w;

- sge: Greater or Equal to Second Source ←→ slt (Less Than)
  - sge r2, r0, r1
    r2.x = (r0.x >= r1.x) ? 1.0f : 0.0f;
    r2.y = (r0.y >= r1.y) ? 1.0f : 0.0f;
    r2.z = (r0.z >= r1.z) ? 1.0f : 0.0f;
    r2.w = (r0.w >= r1.w) ? 1.0f : 0.0f;

GaLic.Heesung Oh
2009, Copyright All rights reserved

- add: 덧셈
  - add r2, r0, r1
    ```
    r2.x = r0.x + r1.x;
    r2.y = r0.y + r1.y;
    r2.z = r0.z + r1.z;
    r2.w = r0.w + r1.w;
    ```

- sub: 뺄셈
  - sub r2, r0, r1
    ```
    r2.x = r0.x - r1.x
    r2.y = r0.y - r1.y
    r2.z = r0.z - r1.z
    r2.w = r0.w - r1.w
    ```

- mul: 곱셈
  - mul r2, r0, r1
    ```
    r2.x = r0.x * r1.x;
    r2.y = r0.y * r1.y;
    r2.z = r0.z * r1.z;
    r2.w = r0.w * r1.w;
    ```

- mad: 곱셈, 덧셈
  - mad r3, r0, r1, r2
    ```
    r3.x  = r0.x * r1.x  + r2.x;
    r3.y  = r0.y * r1.y  + r2.y;
    r3.z  = r0.z * r1.z  + r2.z;
    r3.w = r0.w * r1.w + r2.w;
    ```

1.3 Vertex Shader Coding 기초 - 주요 명령어

2009, Copyright All rights reserved

- pow: 멱수-승수 구하기(x^y). vs_2_0 이상
  - pow r2, r0, r1;
    r2 = pow(abs(r0), r1);
    r2 = exp(r1 * log(r0))

- exp; 2의 승수 2^x

```
def  c10, -1, 0.5, 0., 0.
mov  r0, c10
exp  r0.x, r0.x
exp  r0.y, r0.y
exp  r0.z, r0.z
exp  r0.w, r0.w
```

- log: 2의 지수 , log dst, src; src=0 -> dst = -FLT_MAX

```
def  c10, 4, 3., 2., 2.
mov  r0, c10
log  r0.x, r0.x
log  r0.y, r0.y
log  r0.z, r0.z
log  r0.w, r0.w
```

1.3 Vertex Shader Coding 기초 - 주요 명령어

GaLic.Heesung Oh
2009, Copyright All rights reserved

- frc : Fragment
  - frc dst, src;
    dst.x = src.x - floorf(src.x);
    dst.y = src.y - floorf(src.y);
    dst.z = src.z - floorf(src.z);
    dst.w = src.w - floorf(src.w);

- rcp : 역수
  - rcp dst, src;dst, src의 swizzle component 명시
  - src =0 -> dst = FLT_MAX;

```
def  c10, 4, 3., 2., 1.
mov  r0, c10
rcp  r0.x, r0.x
rcp  r0.y, r0.y
rcp  r0.z, r0.z
rcp  r0.w, r0.w
```

GaLic.Heesung Oh
2009, Copyright All rights reserved

- rsq : 제곱근의 역수
  - dst, src의 swizzle components 명시
  - src =0 이면 dst = FLT_MAX;

```
dp3 r3, r1, r2
rsq r3, r3.w

r3.x = r3.y = r3.z = r3.w = 1.f/sqrtf(r3.w);
```

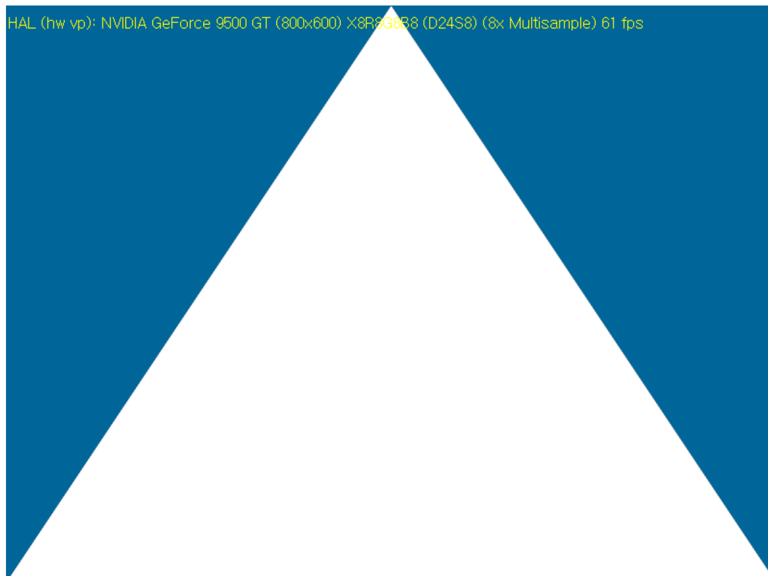
GaLic.Heesung Oh
2009, Copyright All rights reserved

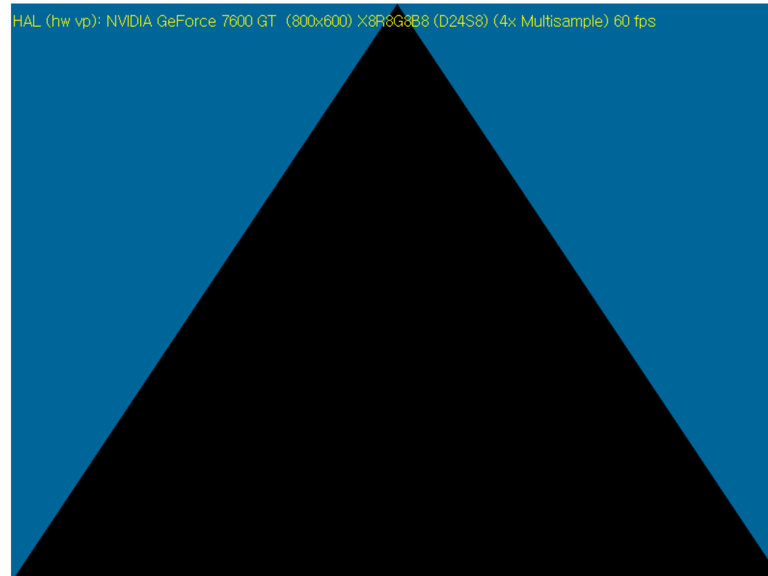
- 입력 받은 정점을 화면에 출력

```
vs_1_1              ; 정점 쉐이더 버전 1.1
dcl_position v0;  입력 레지스터 v0에 정점의 위치를 지정
mov oPos, v0  ;  v0에 적재된 위치 값을 출력 위치 레지스터에 복사
```

◆ // 해석
렌더링 파이프 라인의 정규 변환을 거치면 화면의 좌표는 [-1,1]로 정규화

위와 같은 쉐이더 코드에서 정점의 좌표는 [-1,-1,0] ~ [1,1,1] 범위에 있어야 렌더링 가능
색상이 지정되지 않으면 검정색으로 출력

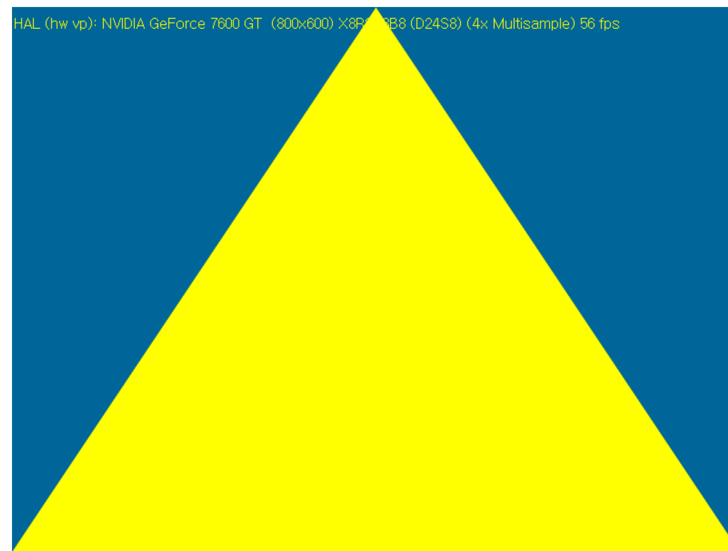




2.1 Simple Vertex Shader

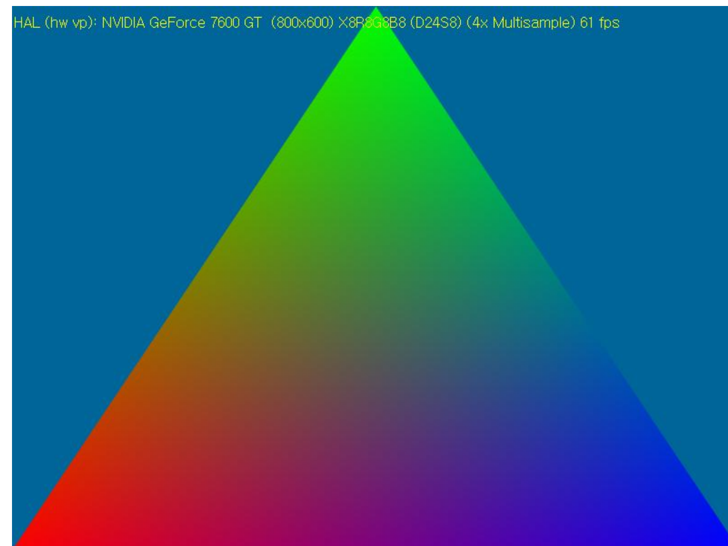
GaLic. Heesung Oh
2009, Copyright All rights reserved

● 쉐이더 코드에서 지정한 색상 출력




```
vs_1_1              ; 정점 쉐이더 버전 1.1
dcl_position v0     ; 입력 레지스터 v0에 정점의 위치를 지정
def c10, 1,1,0,1;   상수 레지스터 c10에 x,y,z,w 또는 r,g,b,a 값을 설정
mov oPos, v0        ; v0에 적재된 위치 값을 출력 위치 레지스터에 복사
mov oD0, c10        ; c10에 설정된 상수 값을 디퓨즈 출력 레지스터에 복사
```

● 정점의 색상을 출력




```
vs_1_1              ; 정점 쉐이더 버전 1.1
dcl_position v0     ; 정점 위치를 입력 레지스터 v0에 선언
dcl_color    v1     ; 정점 색상을 입력 레지스터 v1에 선언
mov oPos,    v0     ; v0에 적재된 위치 값을 출력 위치 레지스터에 복사
mov  oD0,    v1     ; v1에 적재된 정점의 Diffuse 값을 디퓨즈 출력 레지스터에 복사
```

2.2 Transform

2009, Copyright All rights reserved

- 쉐이더 파일 컴파일

```
DWORD dwFlags = 0;

#if defined( _DEBUG ) || defined( DEBUG )
    dwFlags |= D3DXSHADER_DEBUG;
#endif

LPD3DXBUFFER    pShd    = NULL;
LPD3DXBUFFER    pErr    = NULL;

hr = D3DXAssembleShaderFromFile(   "data/Shader.vsh" ,   NULL, NULL
                                  , dwFlags, &pShd, &pErr);

if ( FAILED(hr) )
{
    if(pErr)
    {
        MessageBox( hWnd, (char*)pErr->GetBufferPointer(), "Err", 0);
        pErr->Release();
    }
    return -1;
}
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

Geometry Transform Matrix의 설정

- 고정 함수 파이프 라인
  - pDevice->SetTransform(D3DTS_WORLD, &WorldTM);
  - pDevice->SetTransform(D3DTS_VIEW, &ViewTM);
  - pDevice->SetTransform(D3DTS_PROJECTION, &ProjectionTM);

쉐이더

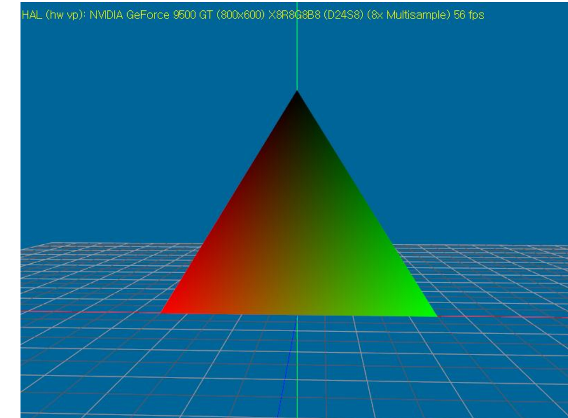
- 행렬의 크기: 4
- 인덱스: 쉐이더 코드에서의 상수 레지스터의 인덱스와 일치
- 행렬을 전치(Transpose)시킨 후 연결

- 상수 설정 예

```
D3DXMatrixTranspose(&matrix, &WorldTM);
pDevice->SetVertexShaderConstantF( 0, (float*)&matrix, 4);

D3DXMatrixTranspose(&matrix, &ViewTM);
pDevice->SetVertexShaderConstantF( 4, (float*)&matrix, 4);

D3DXMatrixTranspose(&matrix, &ProjectionTM);
pDevice->SetVertexShaderConstantF( 8, (float*)&matrix, 4);
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

● 상수 설정 함수와 쉐이더 코드 대응관계




```
vs_1_1                  // 쉐이더 버전
dcl_position v0         // 정점 위치를 v0에 선언
dcl_color    v1         // 정점 색상을 v1에 선언

def  c10, 1, 1, 0, 1    // 상수 설정

m4x4 oPos, v0, c0       // 정점 출력

mul  r0, v1, c10        // r0 = 정점 색상 * 내부 상수(c10) 곱
mul  r1, r0, c11        // r1 =  r0 * 외부 입력 상수(c11)곱
mov  oD0, r1
```

```
m_pDev->SetVertexShaderConstantF(
0, (FLOAT*)&mtVP  , 4);
```

```
m_pDev->SetVertexShaderConstantF(
10, (FLOAT*)&color , 1);
```

```
m_pDev->SetVertexShaderConstantF(
11, (FLOAT*)&color , 1);
```

외부에서 설정한 상수 값보다 쉐이더에서 설정한 상수 값이 우선

GaLic. Heesung Oh
2009, Copyright All rights reserved

- World, View, Projecton Transform

```
// 쉐이더 코드
// 상수 레지스터
//  c0~3: 월드 행렬
//  c4~7: 뷰 렬
//  c8~1: 투영 행렬

vs_1_1              // 쉐이더 버전
dcl_position    v0 // 위치 레지스터
dcl_color       v1  // 색상 레지스터

// 정점의 변환
mov r0, v0          // 정점 위치를 임시 레지스터에 복사
m4x4 r1, r0, c0     // 월드 행렬 변환 후 r1에 저장
m4x4 r2, r1, c4     // 뷰 행렬 변환 후 r2에 저장
m4x4 r3, r2, c8     // 투영 행렬 변환 후 r3에 저장
mov oPos, r3        // r3 값을 출력 레지스터에 저장

// 색상 출력
mov  oD0, v1        // 정점의 색상을 직접 복사
```

```
D3DXMatrixTranspose( &matrix , &mtWld );

pDev->SetVertexShaderConstantF(
0, (FLOAT*)&matrix , 4);
```

```
D3DXMatrixTranspose( &matrix , &mtViw );

pDev->SetVertexShaderConstantF(
4, (FLOAT*)&matrix , 4);
```

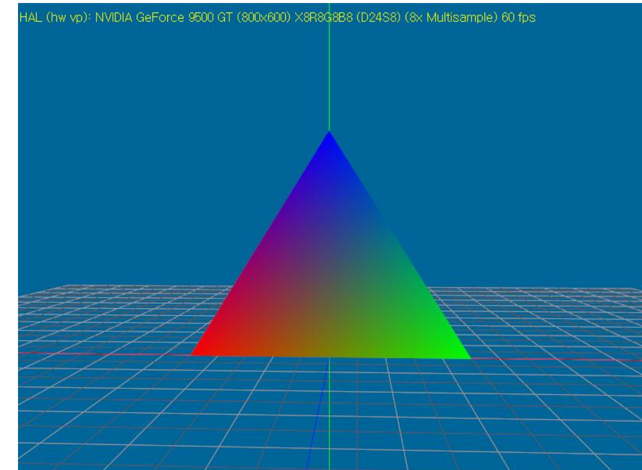
```
D3DXMatrixTranspose( &matrix , &mtPrj );

pDev->SetVertexShaderConstantF(
8, (FLOAT*)&matrix , 4);
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

● 변환에 대한 최적화

◆ 월드, 뷰, 투영 변환 행렬을 외부에서 연산 후 반영

```
// 쉐이더 코드

vs_1_1
dcl_position    v0
dcl_color       v1

// 정점의 변환
mov r0, v0
m4x4 r1, r0, c0
mov oPos, r1

// 색상 출력
mov  oD0, v1
```

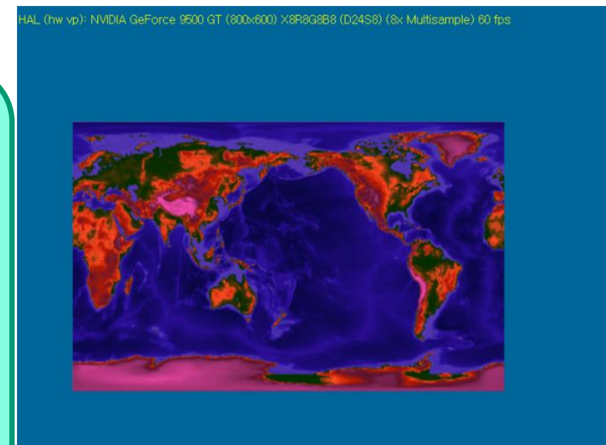


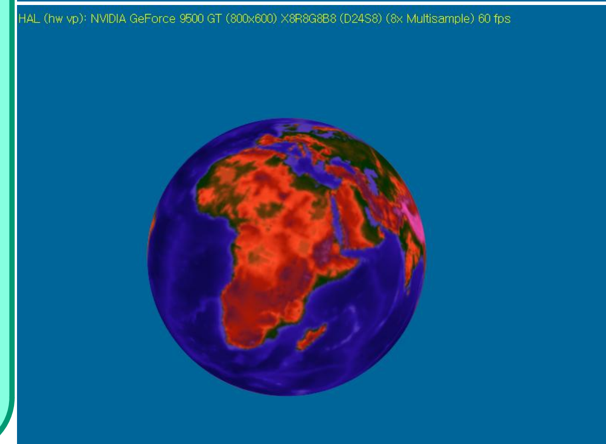

```
D3DXMATRIX  mtVP = mtWld * mtViw * mtPrj;

D3DXMatrixTranspose( &matrix , &mtVP );
pDev->SetVertexShaderConstantF(
 0,  (FLOAT*)&matrix , 4);
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 텍스처를 출력하기 위한 쉐이더 작성
  - 1. 정점의 텍스처 좌표에 대한 입력 레지스터 설정
  - 2. 텍스처 좌표변환(대부분 변환 없음)
  - 3. 출력 레지스터에 복사

```
// 정점의 색상과 텍스처 좌표

vs_1_1
dcl_position     v0
dcl_color        v2
dcl_texcoord     v3     // 정점 UV 좌표를 v3에 선언

m4x4 oPos,  v0,  c0  // 변환

mul  r0, v2,    c10        // ro = 정점의 색상 * 상수(c10)
mov  oD0, r0               // 출력 디퓨즈 색상

mov oT0, v3                // 출력 UV 좌표 = 입력 UV 좌표
```





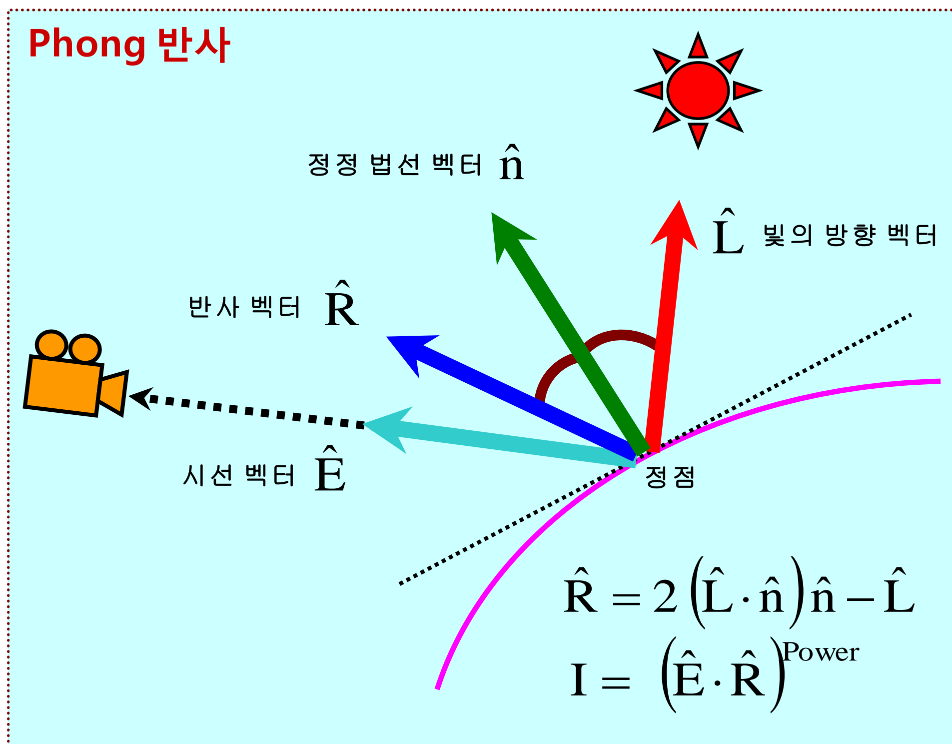
GaLic.Heesung Oh
2009, Copyright All rights reserved

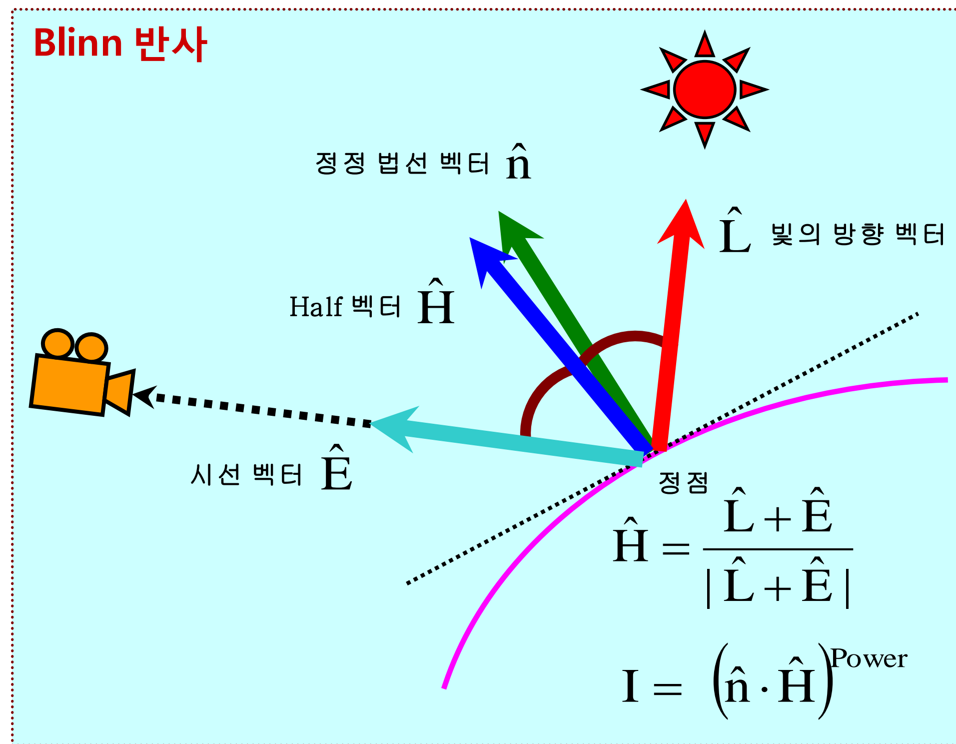
- 램버트 확산
  - 반사 세기 = Dot(정점의 법선(N), 빛의 방향(L))
  - 내적의 범위가 [-1, 1] 이므로 반사 세기 범위를 [0,1]로 변경
  - 반사 세기 = (1+ Dot(N, L))/2

```
vs_1_1                        // 쉐이더 버전
dcl_position  v0              // 정점 위치를 입력 레지스터 v0에 선언
dcl_normal    v1              // 정점 법선 벡터를 입력 레지스터 v1에 선언

def  c12, 1, .5, 0.1, 1.      // 정점과 빛의 내적을 위한 상수

m4x4 oPos, v0, c0             // 출력 위치 = 입력 위치 * c0에 입력된 행렬

m3x3 r0, v1,  c4              // 법선 벡터는 회전만 적용
dp3  r1, r0, -c8              // (-)Light 방향과 내적으로 정점의 밝기를 설정

add  r1,  r1, c12.x           // (Dot + 1)* 0.5로 범위를 [0,1]으로 조정
mul  r1,  r1, c12.y
max  r1, r1, c12.z
min  r1,  r1, c12.w

mul  oD0, r1, c10             // 최종 색상 = 정점의 밝기 * 빛의 색상
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

Phong and Blinn-Phong Reflection



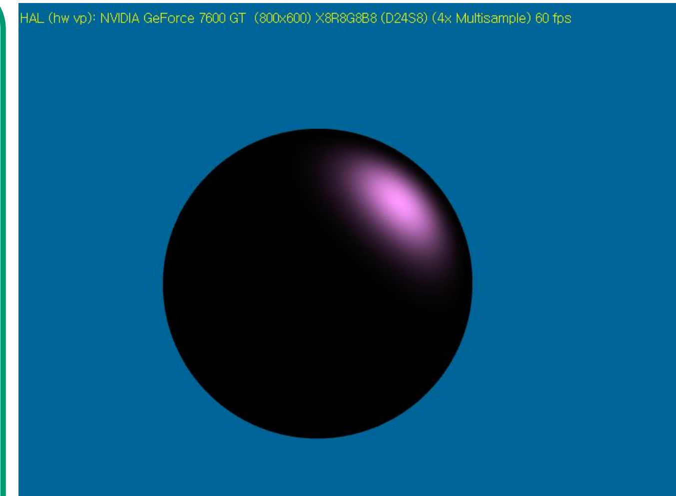


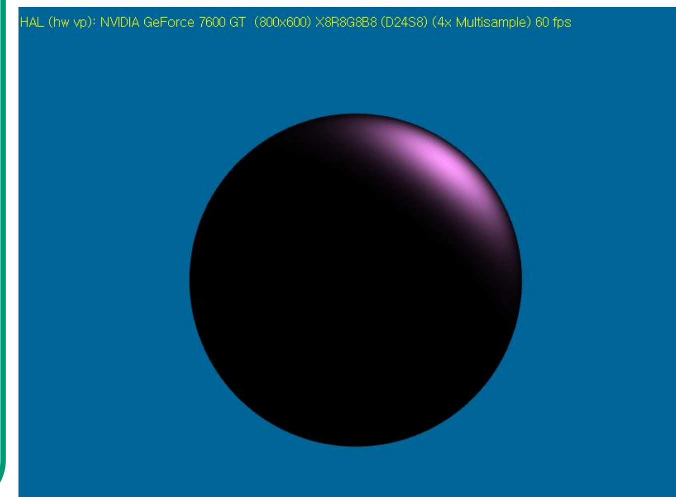
GaLic.Heesung Oh
2009, Copyright All rights reserved

Phong 반사

- 반사 세기 = Dot(E, R)^Power

```
...
// E
sub r1, c16, r0          // E = 카메라 위치 - 정점 위치

dp3 r2.x, r1, r1                  // 단위벡터로 만들기
rsq r2.xyzw, r2.xxxx              // 1/sqrt(Esq)
mul r4, r1, r2                    // Unit Vector

// Reflection Vector
dp3 r1, r3, -c8                   // dot(N, L)
add r1, r1, r1                    // * 2
mul r3.xyz, r3.xyz, r1.xxx        // * N
sub r3.xyz, r3.xyz, -c8.xyz       // -L

dp3 r1.x, r4, r3                  // dot(E, R)
...
// pow(r1, power)
log  r1.x, r1.x
mul r1.x, r1.x, c16.w
exp r1.x, r1.x
```



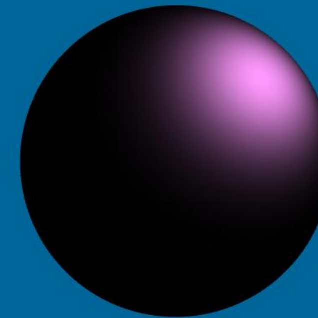


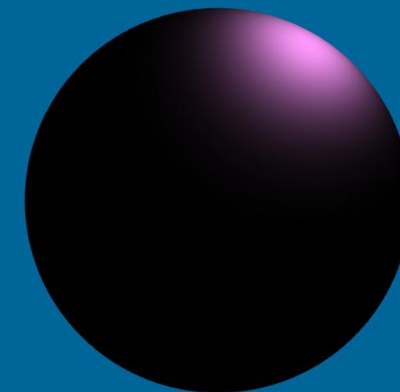
GaLic. Heesung Oh
2009, Copyright All rights reserved

Blinn-Phong 반사

- 반사 세기 = Dot(N, H)^Power

```
...
// E
sub r1, c16, r0          // E = 카메라 위치 - 정점 위치

dp3 r2.x, r1, r1                          // 단위벡터로 만들기
rsq r2.x, r2.x                            // 1/sqrt(Esq)
mul r4.xyzw, r1.xyzw, r2.wxxx             // Unit Vector

// Half Vector
add r1, r1, -c8                 // H = L + E
dp3 r2.x, r1, r1                // 단위 벡터로 만들기
rsq r2.x, r2.x
mul r1.xyz, r1.xyz, r2.xxx

dp3 r1.x, r1, r3                // dot(H, N)
...
// pow(r1, power)
log  r1.x, r1.x
mul r1.x, r1.x, c16.w
exp r1.x, r1.x
```





2.4 Lighting

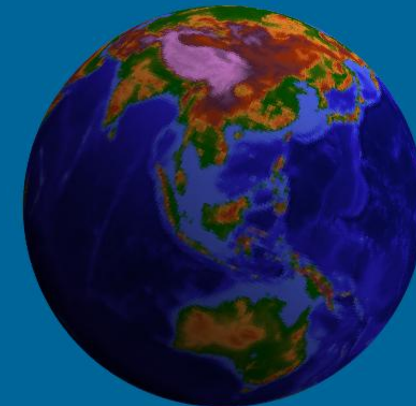
GaLic. Heesung Oh
2009, Copyright All rights reserved

- Lambert + Blinn-Phong 반사

GaLic.Heesung Oh
2009, Copyright All rights reserved

회전 물체에 대한 조명 적용

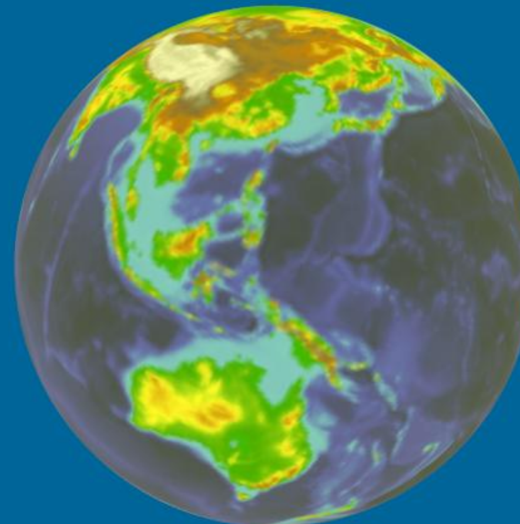
```
vs_1_1              // 쉐이더 버전
dcl_position  v0    // 정점 위치를 입력 레지스터 v0에 선언
dcl_normal    v1    // 정점 법선 벡터를 입력 레지스터 v1에 선언
dcl_color     v2    // 정점 색상을 입력 레지스터 v2에 선언
dcl_texcoord  v3    // 정점 UV 좌표를 입력 레지스터 v3에 선언

def c12, 1, .5, 0.1, 1. // 정점과 빛의 내적을 위한 상수

m4x4 oPos, v0, c0  // 출력 위치 = 입력 위치 * c0에 입력된 행렬

m3x3 r0, v1,  c4     // 법선 벡터는 회전만 적용
dp3  r1, r0, -c8     // Light -방향과 내적으로 정점의 밝기를 설정

add r1, r1, c12.x    // (Dot + 1)* 0.5로 범위를[0,1]로 조정
mul r1, r1, c12.y
max r1, r1, c12.z
min r1, r1, c12.w

mul  r0, r1, c10  // 색상 = 정점의 밝기 * 빛의 색상
mul  oD0, r0, v2 // 최종 색상 = 정점의 밝기 * 빛의 색상 * 정점의 색상

mov  oT0, v3     // 출력 UV 좌표 = 입력 UV 좌표
```



GaLic. Heesung Oh
2009, Copyright All rights reserved

- Range Fog
  - 색상 = 뷰 변환 후 정점의 z 값/(포그 끝 값 – 포그 시작 값)

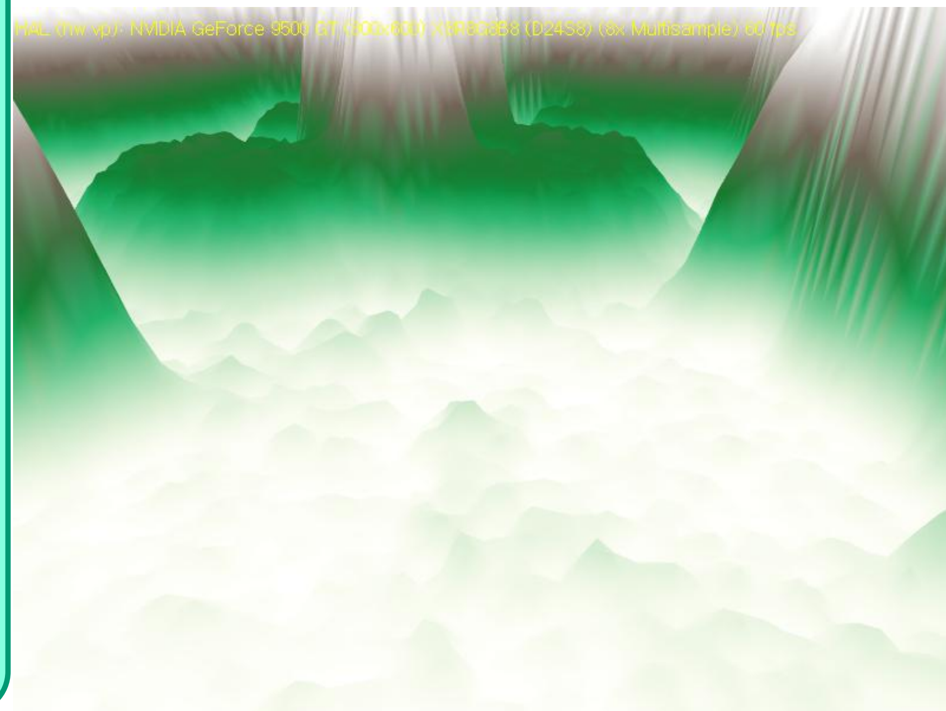
```
// Range Fog

vs_1_1

#define fgB    c9.x
#define fgE    c9.y
#define fgD    c9.z              // fgE-fgB
#define fgR    c9.w              // 1/(fgE-fgB)

dcl_position v0
dcl_texcoord v3

def c11, 0,0,1,1

// vertex processing
m4x4 oPos, v0, c0            // Output Position
mov oT0, v3                  // Output Texture
mov oD0, c10                 // Output Diffuse

// Process Fog
m4x4 r0, v0, c4              // transform vertices by world-view
sub r0.z, fgE, r0.z          // (fog end - distance)
mul r0.x, r0.z, fgR          // (end - distance)/(end - start)

max r0.x, c11.x, r0.x        // 음수 값 제거를 위해 0보다 크거나 같게 함
min r0.x, c11.z, r0.x        // 1 보다 큰 값들을 제거함
mov oFog, r0.x               // 포그 출력 레지스터에 저장
```



GaLic. Heesung Oh
2009, Copyright All rights reserved

- Layered Fog
  - 색상 = 정점의 높이

```
// Layered Fog

vs_1_1

#define fgB    c9.x
#define fgE    c9.y
#define fgD    c9.z              // fgE-fgB
#define fgR    c9.w              // 1/(fgE-fgB)

dcl_position v0
dcl_color    v2

def c11, 0,0,1,1

// vertex processing
m4x4 r0, v0, c0
mov oPos, r0                     // Output Position
mov oD0,  v2                     // Output Diffuse

// Process Fog
mul r0.x, v0.y, fgR              // (end - distance)/(end - start)

max r0.x, c11.x, r0.x            // 음수 값 제거를 위해 0보다 크거나 같게 함
min r0.x, c11.z, r0.x            // 1 보다 큰 값들을 제거함
mov oFog, r0.x                   // 포그 출력 레지스터에 저장
```



GaLic.Heesung Oh
2009, Copyright All rights reserved

- 툰(Cartoon) 쉐이딩
  - 반사의 세기를 양자화 시켜 단계적으로 처리
- 구현 방법
  - 연속적인 세기가 아닌 양자화된 텍스처 준비
  - 반사의 세기를 텍스처 좌표로 환원

```
// Toon Shading

vs_1_1

dcl_position v0
dcl_normal   v1    // 정점 법선 벡터 레지스터 선언 v1

def c12, 1.0, 0.5, 0.1, .9

m4x4 oPos, v0, c0       // 출력 위치 설정

m3x3 r0,  v1, c4        // 법선 벡터에 대한 변환
dp3 r1, r0, -c8         // 정점 밝기

// (Dot + 1)* 0.5로 설정해서 [0.1, .9]로 조정
add  r1,  r1, c12.x
mul  r1,  r1, c12.y
max r1,  r1, c12.z
min  r1,  r1, c12.w

mov oT0.x, r1           // 텍스 x 좌표 설정

mov oD0,   c9           // 색상 설정
```

GaLic.Heesung Oh
2009, Copyright All rights reserved

툰(Cartoon) 쉐이딩 테크닉

- 1차원 텍스처만 필요
  - D3DXCreateTexture(m_pDev , 512, 1 , 0, 0 , D3DFMT_X8R8G8B8, D3DPOOL_MANAGED , &m_pTxToon);

- 쉐이더에서 텍스처 좌표는 x만 설정 ➔y는 자동으로 0이 되어 명령어 사용을 줄임

```
mov oT0.x, r1
```

2009, Copyright All rights reserved

- 깊이 값 출력
  - 깊이 색상 = 정규변환.z * color

```
vs_1_1
dcl_position    v0

m4x4 r0, v0, c0 // 정점 변환
mov  oPos, r0    // 출력 위치 = r0, z=[0,1]

mul oD0, r0.z, c4.x // 디퓨즈 = 거리 * c.x
```

3.3 Vertex Shader Effect 객체

GaLic.Heesung Oh
2009, Copyright All rights reserved

Vertex Shader Interface

```
struct IShaderEffect
{
    virtual ~IShaderEffect(){};
    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(INT nRegister, D3DXMATRIX* v)=0;
    virtual INT  SetVector(INT nRegister, D3DXVECTOR4* v)=0;
    virtual INT  SetColor(INT nRegister, D3DXCOLOR* v)=0;
};
```

객체 생성 함수 정의

- int LcShd_CreateShaderFromFile(IShaderEffect** pData, void* pDevice, char* sFile);
- int LcShd_CreateShaderFromString(IShaderEffect** pData, void* pDevice, char* sString, int iLen);