DirectX Shader
- HLSL

305890
2009년 봄학기
6/3/2009
박경신

셰이더 (Shader)

- 완전 프로그래밍 가능한 작은 커스텀 프로그램
  - 우리가 직접 작성한 커스텀 셰이더 프로그램으로 고정 기능 파이프라인(fixed function pipeline)을 대체
  - *"programmable pipeline"*
- 버텍스(VS)와 픽셀 셰이더(PS)
  - 그래픽 카드의 GPU(graphics processing unit)에서 실행되는 프로그램
    - VS – 고정 기능 파이프라인의 변환과 조명 단계를 대체
    - PS – 각 픽셀의 래스터라이즈 과정 수행
- 더 이상 고정된 작업에 제한될 필요 없음
  - 구현 가능한 그래픽 효과의 범위가 엄청나게 넓어질 수 있음

고수준 셰이딩 언어

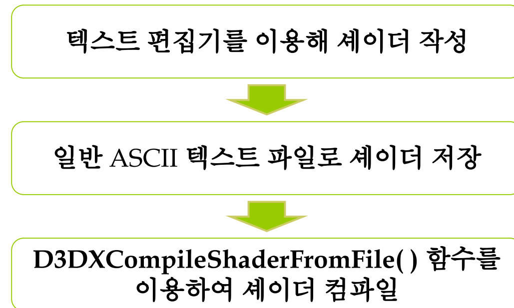
- 셰이더 프로그램 작성을 위한 언어
  - DirectX 8.x : 저수준 셰이더 어셈블리 언어
  - DirectX 9 : 고수준 셰이딩 언어 (HLSL : High-Level Shading Language)
- 이점
  - 생산성 향상
  - 가독성 향상
  - 종종 효율적인 코드 생산
  - 다른 버전의 셰이더 코드 컴파일 가능
- C/C++ 구문과 유사 – 배우기 쉬움
- REF 장치 – 셰이더를 지원하지 않는 그래픽 카드에서 셰이더 예제를 실행해 볼 수 있음

Overview

- **HLSL** 셰이더 프로그램을 작성하고 컴파일 하는 방법
- 응용 프로그램과 셰이더 프로그램 간의 데이터 교환 방법
- HLSL의 구문(syntax), 형(types), 내장 함수(built-in functions)

HLSL 셰이더 프로그램 작성하기

- 응용 프로그램의 코드와 셰이더 프로그램의 코드 분리
  - 모듈화나 편리함을 추구
- 과정



Sample: Transform.txt

```
//////////////////////////////////////////////////////////////////////////
//
// File: transform.txt
//
//////////////////////////////////////////////////////////////////////////

// Globals

// Global variable to store a combined view and projection  transformation matrix.
// We initialize this variable from the application.
matrix ViewProjMatrix;

// Initialize a global blue color vector.
vector Blue = {0.0f, 0.0f, 1.0f, 1.0f};

// Structures

// Input structure describes the vertex that is input into the shader.
// Here the input vertex contains a position component only.
struct VS_INPUT
{
    vector position  : POSITION;
};
```

Sample: Transform.txt

```
// Output structure describes the vertex that is output from the shader.
// Here the output vertex contains a position and color component.
struct VS_OUTPUT
{
    vector position : POSITION;
    vector diffuse  : COLOR;
};

// Main Entry Point, observe the main function  receives a copy of the input vertex
// through its parameter and returns a copy of the output vertex it computes.
VS_OUTPUT Main(VS_INPUT input)
{
    // zero out members of output
    VS_OUTPUT output = (VS_OUTPUT)0;

    // transform to view space and project
    output.position  = mul(input.position, ViewProjMatrix);

    // set vertex diffuse color to blue
    output.diffuse = Blue;

    return output;
}
```

Globals (전역 변수)

```
matrix ViewProjMatrix;
vector Blue = {0.0f, 0.0f, 1.0f, 1.0f};
```

- **ViewProjMatrix**
  - "*matrix*" 형 – HLSL에 내장된 4x4 행렬 형
  - 뷰(view)와 투영(projection)이 결합된 행렬을 보관
  - 응용 프로그램 소스 코드에서 변수가 초기화 됨 – 셰이더에서 아님
- **Blue**
  - "*vector*" 형 – 내장된 4차원 벡터 형
  - 파란색 (RGBA 컬러 벡터)

입력과 출력 구조체

```
struct VS_INPUT
{
   vector position : POSITION;
};
struct VS_OUTPUT
{
   vector position : POSITION;
   vector diffuse  : COLOR;
};
```

- 버텍스 셰이더가 입력하고 출력하는 버텍스 데이터의 구조체
  - 입력(VS_INPUT) – 위치 성분
  - 출력(VS_OUTPUT) – 위치와 컬러 성분
- 콜론 구문 (colon syntax) – "의미"(*semantic*)
  - 변수의 이용법을 지정 – 유연한 버텍스 포멧 (FVF)과 유사

진입점 함수 (Entry Point Function)

```
VS_OUTPUT Main(VS_INPUT input)
{
   VS_OUTPUT output = (VS_OUTPUT)0;
   output.position = mul(input.position, ViewProjMatrix);
   output.diffuse = Blue;
   return output;
}
```

- Main ( )
  - 함수 이름은 고정된 것이 아님 – 올바른 형식의 이름은 모두 가능
  - 입력 구조체 인자를 가져야 함
  - 출력 구조체 인스턴스를 리턴해야 함
- 진입점 함수의 본체
  - 입력 버텍스를 이용해 출력 버텍스를 계산하는 역할
  - **mul** ( ) : 내장 함수 (벡터-행렬 곱, 행렬-행렬 곱)

Note: 입력과 출력 구조체

- 반드시 이용해야 하는 것은 아님

```
float4 Main(in float2 base : TEXCOORD0,
            in float2 spot : TEXCOORD1,
            in float2 text : TEXCOORD2) : COLOR
{
   . . .
}
```

II

```
struct INPUT
{
   float2 base : TEXCOORD0;
   float2 spot : TEXCOORD1;
   float2 text : TEXCOORD2;
};
struct OUTPUT
{
   float4 c : COLOR;
};
OUTPUT Main(INPUT input)
{
   . . .
}
```

HLSL 셰이더의 컴파일

- 상수 테이블 (Constant Table)
  - 셰이더의 변수를 보관
- ID3DXConstantTable 인터페이스
  - 응용 프로그램이 셰이더의 상수 테이블에 접근하여 셰이더 소스 코드 내의 변수 값을 지정
- 상수로의 핸들 얻기
  - D3DXHANDLE – 셰이더의 변수를 참조하기 위한 통로

```
D3DXHANDLE   ID3DXConstantTable::GetConstantByName (
          D3DXHANDLE hConstant,
          LPCSTR pName
);
```

- **hConstant** – 변수가 존재하는 부모 구조체의 핸들
- **pName** – 셰이더 소스 코드 내의 변수 이름

HLSL 셰이더의 컴파일

- 상수 값 설정하기
  - ID3DXConstantTable::SetXXX 메서드

```
HRESULT   ID3DXConstantTable::SetXXX (
          LPDIRECT3DDEVICE9 pDevice,
          D3DXHANDLE hConstant,
          XXX value
);
```

- **pDevice** – 장치로의 포인터
- **hConstant** – 참조하려는 변수의 핸들
- **value** – 지정하려는 변수의 값

- 상수의 디폴트 값 지정하기
  - ID3DXConstantTable::SetDefaults 메서드

```
HRESULT ID3DXConstantTable::SetDefaults (
          LPDIRECT3DDEVICE9 pDevice
);
```

셋업 과정에서 한번만 호출!!!

Sample: 상수 값 설정하기

SetBool	bool b = true; ConstTable->SetBool(Device, handle, b);
SetBoolArray	bool b[3] = {true, false, true}; ConstTable->SetBoolArray(Device, handle, b, 3);
SetFloat	float f = 3.14f; ConstTable->SetFloat(Device, handle, f);
SetFloatArray	float f[2] = {1.0f, 2.0f}; ConstTable->SetFloatArray(Device, handle, f, 2);
SetInt	int x = 4; ConstTable->SetInt(Device, handle, x);
SetIntArray	int x[4] = {1, 2, 3, 4} ConstTalbe->SetIntArray(Device, handle, x, 4);
SetMatrix	D3DXMATRIX M(...); ConstTable->SetMatrix(Device, handle, &M);
SetMatrixArray	D3DXMATRIX M[4]; // ... Initialize matrices ConstTable->SetMatrixArray(Device, handle, M, 4);
SetMatrixPointerArray	D3DXMATRIX *M[4]; // ... Allocate and initialize matrix pointers ConstTable->SetMatrixPointerArray(Device, handle, M, 4);

Sample: 상수 값 설정하기

SetMatrixTranspose	D3DXMATRIX M(...); D3DXMatrixTranspose(&M, &M); ConstTable->SetMatrixTranspose(Device, handle, &M);
SetMatrixTransposeArray	D3DXMATRIX M[4]; // ... Initialize matrices and transpose them ConstTable->SetMatrixTransposeArray(Device, handle, M, 4);
SetMatrixTransposePointerArray	D3DXMATRIX *M[4]; // ... Allocate, initialize matrix pointers and transpose them ConstTable->SetMatrixTransposePointerArray(Device, handle, M, 4);
SetVector	D3DXVECTOR4 v(1.0f, 2.0f, 3.0f, 4.0f); ConstTable->SetVector(Device, handle, &v);
SetVectorArray	D3DXVECTOR4 v[3]; // ... Initialize vectors ConstTable->SetVectorArray(Device, handle, v, 3);
SetValue	D3DXMATRIX M(...); ConstTable->SetValue(Device, handle, (void*)&M, sizeof(M));

HLSL 셰이더의 컴파일

- 텍스트 파일에 저장된 셰이더를 컴파일

```
HRESULT ID3DXCompileShaderFromFile (
          LPCSTR                     pSrcFile,
          CONST D3DXMACRO*           pDefines,
          LPD3DXINCLUDE              pInclude,
          LPCSTR                     pFunctionName,
          LPCSTR                     pTarget,
          DWORD                      Flags,
          LPD3DXBUFFER*              ppShader,
          LPD3DXBUFFER*              ppErrorMsgs,
          LPD3DXCONSTANTTABLE*       ppConstantTable
);
```

- **pScrFile** – 셰이더 소스 코드를 포함한 텍스트 파일의 이름
- **pFunctionName** – 진입점 함수의 이름
- **pTarget** – 컴파일할 셰이더 버전
- **Flags** – 선택적인 컴파일링 플래그
- **ppShader** – 버텍스/픽셀 셰이더를 만드는 다른 함수에 전달하는 인자로 이용

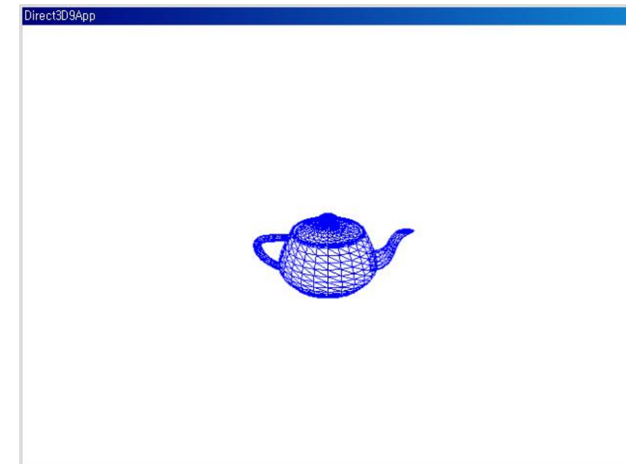
Sample: Compiling a Shader

```
// Compile shader.
ID3DXConstantTable* TransformConstantTable = 0;
ID3DXBuffer* shader      = 0;
ID3DXBuffer* errorBuffer = 0;

hr = D3DXCompileShaderFromFile(
    "transform.txt",
    0,
    0,
    "Main",  // entry point function name
    "vs_1_1",// shader version to compile to
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &TransformConstantTable);

// output any error messages
if( errorBuffer ) {
    ::MessageBox(0, (char*)errorBuffer->GetBufferPointer(), 0, 0);
    d3d::Release<ID3DXBuffer*>(errorBuffer);
}
if(FAILED(hr)) {
    ::MessageBox(0, "D3DXCreateEffectFromFile() - FAILED", 0, 0);
    return false;
}
```

Sample: Transform



Creating a Vertex Shader and Getting the Handles

```
// Create a vertex shader
hr = Device->CreateVertexShader(
                    (DWORD*)shader->GetBufferPointer(),
                    &TransformShader);

if(FAILED(hr)) {
    ::MessageBox(0, "CreateVertexShader - FAILED", 0, 0);
    return false;
}
d3d::Release<ID3DXBuffer*>(shader);

// Get Handles
TransformViewProjHandle = TransformConstantTable->GetConstantByName(
                                                    0,
                                                    "ViewProjMatrix");

// Set shader constants:
TransformConstantTable->SetDefaults(Device);
```

Setting the Shader and its Constants

```
D3DXVECTOR3 position( cosf(angle) * 10.0f, height, sinf(angle) * 10.0f );
D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);
D3DXMATRIX V;
D3DXMatrixLookAtLH(&V, &position, &target, &up);

// Combine view and projection transformations
D3DXMATRIX ViewProj = V * ProjMatrix;

TransformConstantTable->SetMatrix(
                    Device,
                    TransformViewProjHandle,
                    &ViewProj);

// Render
Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
Device->BeginScene();
Device->SetVertexShader(TransformShader);
Teapot->DrawSubset(0);
Device->EndScene();
Device->Present(0, 0, 0, 0);
```

Variable Types (변수 형)

- 스칼라 형 (Scalar Types)
- 벡터 형 (Vector Types)
- 행렬 형 (Matrix Ttypes)
- 배열 (Arrays)
- 구조체 (Structures)
- **typedef** 키워드
- 변수 접두어 (Variable Prefixes)

스칼라 형

- **bool** – **true** 혹은 **false** 값
- **int** – 32-비트 부호 정수
- **half** – 16-비트 부동 소수점 수
- **float** – 32-비트 부동 소수점 수
- **double** – 64-비트 부종 소수점 수

NOTE : 일부 플랫폼에서는 int, half, double 등이 지원되지 않을 수 있다. → float을 이용해 애플레이트 한다.

벡터 형

- **vector** – **float** 형 4차원 벡터
- **vector<T,n>** – **n**차원 **T** 형 벡터
  - **n** – 1에서 4 내의 정수
  - **T** – 스칼라 형
  - 예) **vector<double,2> vec2;**
- 벡터 형의 접근 방법
  - 예) i^{th} 요소 :

```
vec[i] = 2.0f;
vec.x = vec.r = 1.0f;
vec.y = vec.g = 2.0f;
vec.z = vec.b = 3.0f;
vec.w = vec.a = 4.0f;
```

벡터 형

- 미리 정의된 형

```
float2 vec2;
float3 vec3;
float4 vec4;
```

- 벡터 형의 복사
  - *swizzles* – 순서에 구애 받지 않고 복사를 수행

```
vector u1 = {1.0f, 2.0f, 3.0f, 4.0f};
vector v1 = {0.0f, 0.0f, 5.0f, 6.0f};

v1 = u1.xyyw;    // v1 = {1.0f, 2.0f, 2.0f, 4.0f};

vector u2 = {1.0f, 2.0f, 3.0f, 4.0f};
vector v2 = {0.0f, 0.0f, 5.0f, 6.0f};

v2.xy = u2;      // v2 = {1.0f, 2.0f, 5.0f, 6.0f};
```

행렬 형

- **matrix** – 4x4 **float** 형 행렬
- **matrix<T,m,n>** – **m** x **n T** 형 행렬
  - **m, n** – 1에서 4 내의 정수
  - **T** – 스칼라 형
  - 예) **matrix<int,2,2> m2x2;**
- **m** x **n** 행렬을 정의하기 위한 다른 방법

```
float2x2 mat2x2;
float3x3 mat3x3;
float4x4 mat4x4;
float2x4 mat2x4;

int2x2 i2x2;
int3x3 i3x3;
int4x4 i4x4;
```

행렬 형

- 행렬 형의 접근 방법
  - 예) *ij^th^* 요소 : **M[i][j] = value;**
    - 1-기반 :
```
M._11 = M._12 = M._13 = M._14 = 0.0f;
M._21 = M._22 = M._23 = M._24 = 0.0f;
M._31 = M._32 = M._33 = M._34 = 0.0f;
M._41 = M._42 = M._43 = M._44 = 0.0f;
```
    - 0-기반 :
```
M._m00 = M._m01 = M._m02 = M._m03 = 0.0f;
M._m10 = M._m11 = M._m12 = M._m13 = 0.0f;
M._m20 = M._m21 = M._m22 = M._m23 = 0.0f;
M._m30 = M._m31 = M._m32 = M._m33 = 0.0f;
```
  - 예) *i^th^* 행 벡터 :
```
vector ithRow = M[i];
```

Note: 초기화

- 벡터 형
  - 예)
```
vector u = {0.6f, 0.3f, 1.0f, 1.0f};
vector v = {1.0f, 5.0f, 0.2f, 1.0f};
```
    혹은
```
vector u = vector(0.6f, 0.3f, 1.0f, 1.0f);
vector v = vector(1.0f, 5.0f, 0.2f, 1.0f);
```
- 행렬 형
  - 예)
```
float2x2 f2x2 = float2x2(1.0f, 2.0f, 3.0f, 4.0f);
int2x2 m = {1, 2, 3, 4};
```
- 그 외
  - 예)
```
int n = int(5);
int a = {5};
float3 x = float3(0.0f, 0.0f, 0.0f);
```

배열과 구조체

- C++과 비슷한 구문을 이용해 특정한 형의 배열을 선언
  - 예)
```
float M[4][4];
half p[4];
vector v[12];
```
- C++에서와 동일한 방법으로 구조체 정의
  - 멤버 함수는 가질 수 없음
  - 예)
```
struct MyStruct
{
    matrix T;
    vector n;
    float f;
    int x;
    bool b;
};

MyStruct s;      // instantiate
s.f = 5.0f;      // member access
```

typedef 키워드와 변수 접두어

- typedef 키워드는 C++에서와 정확하게 같음
  - 예)
    ```
    typedef vector<float,3> point;
    typedef const float CFLOAT;
    typedef float point2[2];
    ```
  - vector<float,3> myPoint; 대신 point myPoint; 사용가능
- 변수 선언의 접두어

static	- 전역 변수: 셰이더 외부에서 변수 접근할 수 없음 - 지역 변수: C++에서의 정적 지역 변수와 같은 의미
uniform	셰이더 외부(응용 프로그램)에서 초기화됨
extern	셰이더 외부에서 변수 접근 가능 (static이 아닌 전역 변수는 디폴트로 extern)
shared	다수의 효과에 공유될 변수
volatile	자주 수정될 변수
const	C++에서와 같은 의미

키워드

- HLSL가 정의하는 키워드 목록:

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				

- 현재는 이용되지 않지만 나중에 이용할 수 있도록 예약된 키워드 목록:

auto	break	case	catch	char	class
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			

문 (Statements)

- 기본적인 프로그램 흐름 – C++의 문과 매우 비슷
  - return 문:
    ```
    return (expression);
    ```
  - if 와 if…else 문:
    ```
    if( condition )
    {
        statement(s);
    }
    ```
    ```
    if( condition )
    {
        statement(s);
    }
    else
    {
        statement(s);
    }
    ```
  - for 문:
    ```
    for(initial; condition; increment)
    {
        statement(s);
    }
    ```
  - while 와 do…while 문:
    ```
    while( condition )
    {
        statement(s);
    }
    ```
    ```
    do
    {
        statement(s);
    }while( condition );
    ```

형 변환 (Casting)

- 매우 유연한 형 변환 체제
  - C/C++에서의 구문과 동일
  - 예)
    ```
    float f = 5.0f;
    matrix m = (matrix)f;
    ```
- 지원되는 형 변환에 대한 좀더 자세한 정보
  - DirectX SDK 문서 참조

연산자 (Operators)

- C++와 비슷한 연산자 체계

[]	.	>	<	<=	>=
!=	==	!	&&	!!	?:
+	+=	-	-=	*	*=
/	/=	%	%=	++	--
=	()	,			

- 예외 – 나머지(%) 연산자
  - 정수와 부동 소수점 형 모두에 이용 가능
  - 연산자의 오른쪽, 왼쪽 값이 반드시 같은 부호여야 함 (예를 들어, 양쪽 값이 모두 양수이거나 음수)

연산자 (Operators)

- 예외 – 각각의 성분에 대해 연산자가 적용됨
  - 내장되어 있는 벡터와 행렬 형이 여러 개의 성분을 가질 수 있기 때문
  - 예)

```
vector u = { 1.0f, 0.0f, -3.0f, 1.0f};
vector v = {-4.0f, 2.0f,  1.0f, 0.0f};

vector sum = u + v;        // sum=(-3.0f,2.0f,-2.0f,1.0f)
sum++;                     // sum=(-2.0f,3.0f,-1.0f,2.0f)
vector u1 = { 1.0f, 0.0f, -3.0f, 1.0f};
vector v1 = {-4.0f, 2.0f,  1.0f, 0.0f};

vector p = u1 * v1;        // p=(-4.0f,0.0f,-3.0f,0.0f)
vector u2 = { 1.0f, 0.0f, -3.0f, 1.0f};
vector v2 = {-4.0f, 0.0f,  1.0f, 1.0f};

vector b = (u2 == v2);     // b=(false,true,false,true)
```

연산자 (Operators)

- 이항 연산에 따른 변수 승급
  - 왼쪽과 오른쪽 피연산자의 형이 다른 경우:
  - ➔ 낮은 형을 갖는 피연산자가 승급 (형 변환)
    - 예) **int x; half y; (x + y);**
      - ➔ **x** 가 **half** 형으로 승급
  - 왼쪽과 오른쪽 피연산자의 크기가 다른 경우:
  - ➔ 작은 크기를 갖는 피연산자가 승급 (형 변환)
    - 예) **float x; float3 y; (x + y);**
      - ➔ **x** 가 **float3** 형, 즉 **x = (x, x, x)**로 스칼라-벡터 형 변환 정의에 따라 승급
  - 형 변환이 정의되어 있지 않은 경우, 승급 역시 정의되지 않음
    - 예) **float2** 에서 **float3** 로의 승급은 불가능

사용자 정의 함수

- HLSL 함수들의 특징
  - C++과 비슷한 구문을 사용
  - 인자는 항상 값으로 전달 (call-by-value)
  - 재귀는 지원되지 않음
  - 함수는 항상 인라인(inline)으로 쓰여짐
- 부가적인 키워드(**in, out, inout**)의 사용 예

```
bool foo(in const bool b, out int r1, inout float r2)
{
  if( b )
  {
         r1 = 5;
  }
  else
  {
         r1 = 1;
  }
  r2 = r2 * r2 * r2;
  return true;
}
```

사용자 정의 함수

- 부가적인 키워드: **in, out, inout**
  - **in** – 함수가 시작되기 전에 인수를 인자로 복사할 것임을 지정 (인자들은 디폴트로 **in**임)

```
float square(in float x)
{
   return x * x;
}
```

상등

```
float square(float x)
{
   return x * x;
}
```

  - **out** – 함수가 리턴할 때 인자를 인수에 복사할 것임을 지정

```
void square(in float x, out float y)
{
   y = x * x;
}
```

  - **inout** – 인자가 **in** 과 **out** 모두로 이용됨을 표기

```
void square(inout float x)
{
   x = x * x;
}
```

내장 함수들

- 3차원 그래픽에 유용한 내장 함수들을 풍부하게 제공

abs(x)	ceil(x)	clamp(x,a,b)	cos(x)
cross(u,v)	degrees(x)	determinant(M)	distance(u,v)
dot(u,v)	floor(x)	length(v)	lerp(u,v,t)
log(x)	log10(x)	log2(x)	max(x,y)
min(x,y)	mul(M,N)	normalize(v)	pow(b,n)
radians(x)	reflect(v,n)	refract(v,n,eta)	rsqrt(x)
saturate(x)	sin(x)	sincos(in x,out s,out c)	sqrt(x)
tan(x)	transpose(M)		

  - 대부분의 함수들이 내장된 형들과 작업할 수 있도록 오버로드 됨
    - 예) **abs**(x) – 모든 스칼라 형에 대해 오버로드 됨
      cross(u,v) – 모든 형(int, float, double,...)의 3차원 벡터에
      lerp(u,v,t) – 스칼라와 2D, 3D, 4D 벡터에 대해 오버로드

내장 함수들

- 예)

```
float x = sin(1.0f);    // sine of 1.0f radian
float y = sqrt(4.0f);   // square root of 4

vector u = {1.0f,  2.0f, -3.0f, 0.0f};
vector v = {3.0f, -1.0f,  0.0f, 2.0f};
float s = dot(u,v);     // dot product of u and v

float3 i = {1.0f, 0.0f, 0.0f};
float3 j = {0.0f, 1.0f, 0.0f};
float3 k = cross(i,j);            // cross product of u and v

matrix<float,2,2> M = {1.0f, 2.0f, 3.0f, 4.0f};
matrix<float,2,2> T = transpose(M);    // transpose of M
```

NOTE: 스칼라에만 동작하는 함수에 스칼라가 아닌 형을 전달하면, 함수는 성분 단위로 동작한다.

- 예)

```
float3 v = {0.0f, 0.0f, 0.0f};
v = cos(v);          // v = (cos(v.x),cos(v.y),cos(v.z))
```