1. 3D 그래픽 기초(3D Graphics Fundamentals)

(8) DirectX Math

⑨ 삼각 함수

다음과 같이 스칼라 각도에 대한 삼각함수들이 제공된다. 함수 XMScalarCos()는 C-언어 라이브러리의 함수 cos()과 같다. 삼각함수들의 각도(value)는 라디안(Radian)을 사용함에 주의하라. 함수 XMScalarCosEst()는 코사인의 근사값을 반환하므로(실행 속도가 빠르다) 정확도가 요구되지 않는 경우 성능을 위하여 사용할 수 있다. sin(), tan(), acos() 등의 삼각함수들도 유사하게 제공된다.

```
float XMScalarCos(float value);
float XMScalarCosEst(float value);
float XMScalarACos(float value);
float XMScalarACosEst(float value);
float XMScalarASin(float value);
```

다음 함수 XMScalarSinCos()는 각도(value)에 대한 사인과 코사인 값을 반환한다.

```
void XMScalarSinCos(float *pSin, float *pCos, float value);
void XMScalarSinCosEst(float *pSin, float *pCos, float value);
```

다음 함수 XMScalarModAngle()는 각도(value)에 모듈러 연산(value % 2π)의 결과($-\pi$ ~ π)를 반환한다.

```
float XMScalarModAngle(float value);
```

다음 함수 XMScalarNearEqual()는 두 개의 실수(v1, v2) 값이 같은 값을 가지는 가를 반환한다. (|v1 - v2| ≤ epsilon)이면 v1과 v2가 같은 값을 갖는다고 판단한다.

```
bool XMScalarNearEqual(float v1, float v2, float epsilon);
```

다음과 같이 각도들의 벡터에 대한 삼각함수들이 제공된다. 함수 XMVectorCos()는 벡터 v의 각 요소의 코사인 값을 벡터 $(cos(v_x), cos(v_y), cos(v_z), cos(v_w))$로 반환한다.

```
XMVECTOR XMVectorCos(XMVECTOR v);
XMVECTOR XMVectorSin(XMVECTOR v);
XMVECTOR XMVectorACos(XMVECTOR v);
XMVECTOR XMVectorACosEst(XMVECTOR v);
void XMVectorSinCos(XMVECTOR *pSin, XMVECTOR *pCos, XMVECTOR v);
```

⑩ 벡터 산술(Arithmetic) 연산 함수

다음 함수 XMVectorAbs()는 벡터 $\mathbf{v} = (v_x, v_y, v_z, v_w)$의 각 요소의 절대 값을 벡터 $(|v_x|, |v_y|, |v_z|, |v_w|)$로 반환한다.

```
XMVECTOR XMVectorAbs(XMVECTOR v);
```

다음 함수 XMVectorAdd()는 벡터 $\mathbf{v}_1 = (x_1, y_1, z_1, w_1)$과 $\mathbf{v}_2 = (x_2, y_2, z_2, w_2)$의 요소별 합(덧셈) $(\mathbf{v}_1 + \mathbf{v}_2)$을 반환한다.

```
XMVECTOR XMVectorAdd(XMVECTOR v1, XMVECTOR v2);
```

$$\mathbf{v}_1 + \mathbf{v}_2 = (x_1 + x_2, y_1 + y_2, z_1 + z_2, w_1 + w_2)$$

다음 함수 XMVectorSubtract()는 벡터 $\mathbf{v}_1 = (x_1, y_1, z_1, w_1)$과 $\mathbf{v}_2 = (x_2, y_2, z_2, w_2)$의 요소별 차(뺄셈) $(\mathbf{v}_1 - \mathbf{v}_2)$을 반환한다.

```
XMVECTOR XMVectorSubtract(XMVECTOR v1, XMVECTOR v2);
```

$$\mathbf{v}_1 - \mathbf{v}_2 = (x_1 - x_2, y_1 - y_2, z_1 - z_2, w_1 - w_2)$$

다음 함수 XMVectorMultiply()는 벡터 $\mathbf{v}_1 = (x_1, y_1, z_1, w_1)$과 $\mathbf{v}_2 = (x_2, y_2, z_2, w_2)$의 요소별 곱(곱셈) $(\mathbf{v}_1 * \mathbf{v}_2)$을 반환한다.

```
XMVECTOR XMVectorMultiply(XMVECTOR v1, XMVECTOR v2);
```

$$\mathbf{v}_1 * \mathbf{v}_2 = (x_1 x_2, y_1 y_2, z_1 z_2, w_1 w_2)$$

다음 함수 XMVectorDivide()는 벡터 $\mathbf{v}_1 = (x_1, y_1, z_1, w_1)$과 $\mathbf{v}_2 = (x_2, y_2, z_2, w_2)$의 요소별 나눗셈 $(\mathbf{v}_1 \div \mathbf{v}_2)$을 반환한다.

```
XMVECTOR XMVectorDivide(XMVECTOR v1, XMVECTOR v2);
```

$$\mathbf{v}_1 \div \mathbf{v}_2 = (x_1 \div x_2, y_1 \div y_2, z_1 \div z_2, w_1 \div w_2)$$

다음 함수 XMVectorScale()는 벡터 $\mathbf{v} = (x, y, z, w)$와 실수 s의 곱셈(스칼라 곱셈)을 반환한다. XMVectorNegate()는 벡터 $\mathbf{v} = (x, y, z, w)$와 실수 -1의 곱셈을 반환한다.

```
XMVECTOR XMVectorScale(XMVECTOR v, float s);
XMVECTOR XMVectorNegate(XMVECTOR v);
```

$$s * \mathbf{v} = (sx, sy, sz, sw)$$

다음 함수 XMVectorSqrt()는 벡터 $\boldsymbol{v} = (x, y, z, w)$의 각 요소의 제곱근을 계산하여 벡터로 반환한다.

```
XMVECTOR XMVectorSqrt(XMVECTOR v);
XMVECTOR XMVectorSqrtEst(XMVECTOR v);
```

$$\sqrt{\boldsymbol{v}} = (\sqrt{x}, \sqrt{y}, \sqrt{z}, \sqrt{w})$$

다음 함수 XMVectorMultiplyAdd()는 벡터 $\boldsymbol{v}_1 = (x_1, y_1, z_1, w_1)$과 $\boldsymbol{v}_2 = (x_2, y_2, z_2, w_2)$의 요소별 곱(곱셈) $(\boldsymbol{v}_1 * \boldsymbol{v}_2)$을 벡터 $\boldsymbol{v}_3 = (x_3, y_3, z_3, w_3)$과 요소별 덧셈을 한 결과를 반환한다.

```
XMVECTOR XMVectorMultiplyAdd(XMVECTOR v1, XMVECTOR v2, XMVECTOR v3);
```

$$(\boldsymbol{v}_1 * \boldsymbol{v}_2) + \boldsymbol{v}_3 = (x_1x_2 + x_3, y_1y_2 + y_3, z_1z_2 + z_3, w_1w_2 + w_3)$$

다음 함수 XMVectorSaturate()는 벡터 $\boldsymbol{v} = (x, y, z, w)$의 각 요소의 값을 0과 1사이의 값이 되도록 만들어 벡터로 반환한다.

```
XMVECTOR XMVectorSaturate(XMVECTOR v);
```

$$\boldsymbol{s} = (min(max(v_x, 0), 1), min(max(v_y, 0), 1), min(max(v_z, 0), 1), min(max(v_w, 0), 1))$$

다음 함수 XMVectorAddAngles()는 각도를 나타내는 벡터 $\boldsymbol{v}_1 = (x_1, y_1, z_1, w_1)$과 $\boldsymbol{v}_2 = (x_2, y_2, z_2, w_2)$의 요소별 합(덧셈) $(\boldsymbol{v}_1 + \boldsymbol{v}_2)$을 반환한다.

```
XMVECTOR XMVectorAddAngles(XMVECTOR v1, XMVECTOR v2);
```

벡터 $\boldsymbol{v}_1$의 각 요소 θ는 $(-\pi \le \theta < \pi)$이고 벡터 $\boldsymbol{v}_2$의 각 요소 ϕ는 $(-2\pi \le \pi < 2\pi)$이어야 하며, 반환되는 벡터 $\boldsymbol{v}$의 각 요소 ρ는 $(-\pi \le \rho < \pi)$이다.

다음 함수 XMVectorClamp()는 벡터(v)의 각 요소의 값이 두 벡터(min, max)의 대응되는 요소 사이의 값이 되도록 바꾸어(클램핑하여) 벡터를 반환한다.

```
XMVECTOR XMVectorClamp(XMVECTOR v, XMVECTOR min, XMVECTOR max);
```

벡터 $\boldsymbol{min} = (min_x, min_y, min_z, min_w)$, 벡터 $\boldsymbol{max} = (max_x, max_y, max_z, max_w)$에 대하여 벡터 $\boldsymbol{v} = (v_x, v_y, v_z, v_w)$를 클램핑하여 반환되는 벡터 $\boldsymbol{r} = (r_x, r_y, r_z, r_w)$은 다음과 같다.

$$r_x = min(max(v_x, min_x), max_x)$$
$$r_y = min(max(v_y, min_y), max_y)$$
$$r_z = min(max(v_z, min_z), max_z)$$
$$r_w = min(max(v_w, min_w), max_w)$$

다음 함수 XMVectorCeiling()는 벡터(v)의 각 요소의 천장(Ceiling) 값을 계산하여 벡터로 반환한다. 함수 XMVectorFloor()는 벡터(v)의 각 요소의 바닥(Floor) 값을 계산하여 벡터로 반환한다.

```
XMVECTOR XMVectorCeiling(XMVECTOR v);
XMVECTOR XMVectorFloor(XMVECTOR v);
```

다음 함수 XMVectorMin()은 두 벡터(v1, v2)의 각 요소의 최소(min) 값을 계산하여 벡터로 반환한다. 함수 XMVectorMax()는 두 벡터(v1, v2)의 각 요소의 최대(max) 값을 계산하여 벡터로 반환한다.

```
XMVECTOR XMVectorMin(XMVECTOR v1, XMVECTOR v2);
XMVECTOR XMVectorMax(XMVECTOR v1, XMVECTOR v2);
```

벡터 $\boldsymbol{v}_1 = (x_1, y_1, z_1, w_1)$과 $\boldsymbol{v}_2 = (x_2, y_2, z_2, w_2)$의 최소 벡터 $\boldsymbol{m} = (m_x, m_y, m_z, m_w)$와 최대 벡터 $\boldsymbol{M} = (M_x, M_y, M_z, M_w)$은 다음과 같다.

$$m_x = min(x_1, x_2),\ m_y = min(y_1, y_2),\ m_z = min(z_1, z_2),\ m_w = min(w_1, w_2)$$

$$M_x = max(x_1, x_2),\ M_y = max(y_1, y_2),\ M_z = max(z_1, z_2),\ M_w = max(w_1, w_2)$$

다음 함수 XMVectorReciprocal()는 벡터 $\boldsymbol{v} = (x, y, z, w)$의 각 요소의 역수를 벡터로 반환한다.

```
XMVECTOR XMVectorReciprocal(XMVECTOR v);
XMVECTOR XMVectorReciprocalEst(XMVECTOR v);
```

$$\frac{1}{\boldsymbol{v}} = \left(\frac{1}{x}, \frac{1}{y}, \frac{1}{z}, \frac{1}{w}\right)$$

다음 함수 XMVectorReciprocalSqrt()는 벡터 $\boldsymbol{v} = (x, y, z, w)$의 각 요소의 제곱근의 역수를 벡터로 반환한다.

```
XMVECTOR XMVectorReciprocalSqrt(XMVECTOR v);
XMVECTOR XMVectorReciprocalSqrtEst(XMVECTOR v);
```

$$\frac{1}{\sqrt{\boldsymbol{v}}} = \left(\frac{1}{\sqrt{x}}, \frac{1}{\sqrt{y}}, \frac{1}{\sqrt{z}}, \frac{1}{\sqrt{w}}\right)$$

다음 함수 XMVectorRound()는 벡터 $\boldsymbol{v} = (x, y, z, w)$의 각 요소를 정수로 반올림하여 벡터로 반환한다. 함수 XMVectorTruncate()는 벡터 $\boldsymbol{v} = (x, y, z, w)$의 각 요소를 정수로 만들어(소수부분을 버림) 벡터로 반환한다.

```
XMVECTOR XMVectorRound(XMVECTOR v);
```

```
XMVECTOR XMVectorTruncate(XMVECTOR v);
```

다음 함수 XMVectorMod()은 벡터(v1)의 각 요소를 벡터(v2)의 대응되는 요소로 실수 나머지 값을 계산하여 벡터로 반환한다.

```
XMVECTOR XMVectorMod(XMVECTOR v1, XMVECTOR v2);
```

다음 함수 XMVectorModAngles()은 각도를 나타내는 벡터(v)의 각 요소를 2π로 나눈 나머지를 계산하여 벡터로 반환한다.

```
XMVECTOR XMVectorModAngles(XMVECTOR v);
```

다음 함수 XMVectorIsInfinite()는 벡터(v)의 각 요소가 $\pm\infty$인 가를 판단하여 벡터로 반환한다. 함수 XMVectorIsNaN()는 벡터(v)의 각 요소가 NaN인 가를 판단하여 벡터로 반환한다.

```
XMVECTOR XMVectorIsInfinite(XMVECTOR v);
XMVECTOR XMVectorIsNaN(XMVECTOR v);
```

⑪ 벡터 비교(Comparison) 연산 함수

다음 함수 XMVectorEqual()는 두 벡터(v1, v2)의 각 요소가 같은 가를 비교한 결과를 벡터로 반환한다.

```
XMVECTOR XMVectorEqual(XMVECTOR v1, XMVECTOR v2);
```

벡터 $\boldsymbol{v}_1=(x_1, y_1, z_1, w_1)$과 $\boldsymbol{v}_2=(x_2, y_2, z_2, w_2)$에 대하여 다음 계산의 결과를 벡터 $\boldsymbol{r}$로 반환한다.

$$r_x = (x_1 == x_2)\ ?\ \mathrm{0xffffffff} : 0$$
$$r_y = (y_1 == y_2)\ ?\ \mathrm{0xffffffff} : 0$$
$$r_z = (z_1 == z_2)\ ?\ \mathrm{0xffffffff} : 0$$
$$r_w = (w_1 == w_2)\ ?\ \mathrm{0xffffffff} : 0$$

다음 함수 XMVectorNearEqual()는 두 벡터(v1, v2)의 각 요소가 오차 범위 내에서 같은 가를 비교한 결과를 벡터로 반환한다.

```
XMVECTOR XMVectorNearEqual(XMVECTOR v1, XMVECTOR v2, XMVECTOR
epsilon);
```

다음 함수 XMVectorEqualR()는 XMVectorEqual()과 같은 검사를 수행한다. 반환 벡터의 모든 요소의 값이 0xffffffff이면 XM_CRMASK_CR6TRUE를 결과(pCR)로 반환하고, 모든 요소의 값이 0이면 XM_CRMASK_CR6FALSE를 결과(pCR)로 반환한다.

```
XMVECTOR XMVectorEqualR(uint32_t *pCR, XMVECTOR v1, XMVECTOR v2);
```

다음 함수 XMComparisonAllTrue()는 함수 XMVectorEqualR()의 호출에서 반환한 결과(pCR)가 XM_CRMASK_CR6TRUE이면 참을 반환한다. XMComparisonAllFalse()는 함수 XMVectorEqualR()의 호출에서 반환한 결과(pCR)가 XM_CRMASK_CR6FALSE이면 참을 반환한다. XMComparisonAnyTrue()는 함수 XMVectorEqualR()의 호출에서 반환한 결과(pCR)가 XM_CRMASK_CR6FALSE가 아니면 참을 반환한다. XMComparisonAnyFalse()는 함수 XMVectorEqualR()의 호출에서 반환한 결과(pCR)가 XM_CRMASK_CR6TRUE가 아니면 참을 반환한다.

```
bool XMComparisonAllTrue(uint32_t CR);
bool XMComparisonAllFalse(uint32_t CR);
bool XMComparisonAnyTrue(uint32_t CR);
bool XMComparisonAnyFalse(uint32_t CR);
```

다음 함수 XMVectorGreater()는 두 벡터(v1, v2)의 각 요소의 크기를 비교한 결과를 벡터로 반환한다.

```
XMVECTOR XMVectorGreater(XMVECTOR v1, XMVECTOR v2);
```

벡터 $\boldsymbol{v}_1 = (x_1, y_1, z_1, w_1)$과 $\boldsymbol{v}_2 = (x_2, y_2, z_2, w_2)$에 대하여 다음 계산의 결과를 벡터 $\boldsymbol{r}$로 반환한다.

$$r_x = (x_1 > x_2)\ ?\ \mathrm{0xffffffff} : 0$$

$$r_y = (y_1 > y_2)\ ?\ \mathrm{0xffffffff} : 0$$

$$r_z = (z_1 > z_2)\ ?\ \mathrm{0xffffffff} : 0$$

$$r_w = (w_1 > w_2)\ ?\ \mathrm{0xffffffff} : 0$$

두 벡터(v1, v2)의 각 요소의 크기를 비교하기 위한 다음의 함수들이 제공된다. 이 함수들의 의미와 사용법은 앞에서 설명한 것과 유사하다.

```
XMVECTOR XMVectorLess(XMVECTOR v1, XMVECTOR v2);
XMVECTOR XMVectorLessOrEqual(XMVECTOR v1, XMVECTOR v2);
XMVECTOR XMVectorGreaterOrEqual(XMVECTOR v1, XMVECTOR v2);

XMVECTOR XMVectorGreaterR(uint32_t *pCR, XMVECTOR v1, XMVECTOR v2);
XMVECTOR XMVectorLessR(uint32_t *pCR, XMVECTOR v1, XMVECTOR v2);
XMVECTOR XMVectorLessOrEqualR(uint32_t *pCR, XMVECTOR v1, XMVECTOR
v2);
XMVECTOR XMVectorGreaterOrEqualR(uint32_t *pCR, XMVECTOR v1,
XMVECTOR v2);
```

- 3차원 벡터 비교(Comparison) 함수

  3차원 벡터의 비교 함수는 위의 벡터 비교 함수와 유사하며, 벡터의 w요소를 제외한

나머지 요소들의 비교를 수행하는 것이 다르다.

다음 함수 XMVector3Equal()는 두 3차원 벡터(v1, v2)의 x-요소, y-요소, z-요소가 모두 같으면 참을 반환한다.

```
bool XMVector3Equal(XMVECTOR v1, XMVECTOR v2);
```

다음 함수 XMVector3EqualR()는 두 3차원 벡터(v1, v2)의 모든 요소가 같으면 XM_CRMASK_CR6TRUE를 반환하고, 모든 요소가 같지 않으면 XM_CRMASK_CR6FALSE를 반환한다. 적어도 하나의 요소가 같거나 또는 적어도 하나의 요소가 같지 않으면 0을 반환한다.

```
uint32_t XMVector3EqualR(XMVECTOR v1, XMVECTOR v2);
```

```
bool XMVector3NearEqual(XMVECTOR v1, XMVECTOR v2, XMVECTOR e);
bool XMVector3NotEqual(XMVECTOR v1, XMVECTOR v2);
bool XMVector3Greater(XMVECTOR v1, XMVECTOR v2);
uint32_t XMVector3GreaterR(XMVECTOR v1, XMVECTOR v2);
bool XMVector3GreaterOrEqual(XMVECTOR v1, XMVECTOR v2);
uint32_t XMVector3GreaterOrEqualR(XMVECTOR v1, XMVECTOR v2);
bool XMVector3Less(XMVECTOR v1, XMVECTOR v2);
bool XMVector3LessOrEqual(XMVECTOR v1, XMVECTOR v2);
bool XMVector3IsInfinite(XMVECTOR v);
bool XMVector3IsNaN(XMVECTOR v);
```

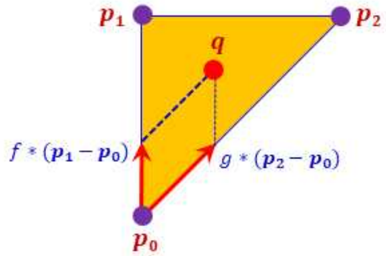
⑫ 벡터 기하(Geometric) 함수

다음 함수 XMVectorBaryCentric()는 삼각형의 세 꼭지점(p0, p1, p2)을 사용하여 삼각형을 포함하는 평면 위의 점을 반환한다. 평면 위의 점은 무게중심좌표(BaryCentric coordinates: f, g)를 사용하여 표현한다.

```
XMVECTOR XMVectorBaryCentric(XMVECTOR p0, p1, p2, float f, float g);
XMVECTOR XMVectorBaryCentricV(XMVECTOR p0, p1, p2, XMVECTOR f, g);
```

삼각형($\triangle p_0 p_1 p_2$)을 포함하는 평면 위의 점 q는 다음과 같이 매개변수 f와 g를 사용하여 다음과 같이 표현할 수 있다.

$$q = p_0 + f(p_1 - p_0) + g(p_2 - p_0)$$



$(f \geq 0), (g \geq 0), (f + g \leq 1)$이면 점 q는 삼각형 내부의 점이다.

$(f = 0), (g \geq 0), (f + g \leq 1)$이면 점 q는 선분 $\overline{p_0 p_2}$의 점이다.

$(f \geq 0), (g = 0), (f + g \leq 1)$이면 점 q는 선분 $\overline{p_0 p_1}$의 점이다.

$(f \geq 0), (g \geq 0), (f + g = 1)$이면 점 q는 선분 $\overline{p_1 p_2}$의 점이다.

다음 함수 XMVectorLerp()는 벡터(v0)와 벡터(v1)을 매개변수(t)로 선형 보간한 결과를 벡터로 반환한다.

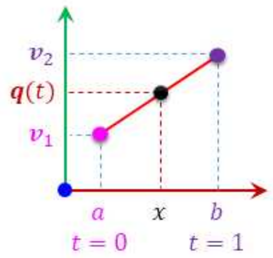
```
XMVECTOR XMVectorLerp(XMVECTOR v0, XMVECTOR v1, float t);
```

벡터 v_0와 벡터 v_1을 매개변수 t로 선형 보간(Linear Interpolation)한 결과는 다음과 같다.

$$q(t) = v_0 + t(v_1 - v_0) = (1 - t)v_0 + tv_1$$

벡터 v_0와 벡터 v_1을 선형 보간하는 것은 벡터 v_0와 벡터 v_1을 지나는 직선 위의 점을 구하는 것이다. 매개 변수 t는 다음과 같이 구할 수 있다.

$$t = \frac{x - a}{b - a}$$



다음 함수 XMVectorHermite()는 점(p0)과 접선의 기울기(tangent0), 점(p1)과 접선의 기울기(tangent1)을 사용하여 헤르밋(Hermite) 스플라인 보간을 수행하여 매개변수 t에 대한 곡선 위의 점을 반환한다.

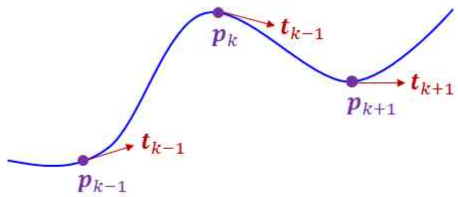
```
XMVECTOR XMVectorHermite(XMVECTOR p0, XMVECTOR tangent0, XMVECTOR
p1, XMVECTOR tangent1, float t);
```

두개의 제어점 p_0, p_1과 접선의 기울기 t_0, t_1를 사용한 헤르밋 스플라인 곡선 $q(t)$는 다음과 같다.

$$q(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)t_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)t_1$$

n개의 제어점(Control point)들의 집합 $\{p_0, p_1, \cdots, p_{n-1}, p_n\}$과 제어점에서 접선의 기울기(Tangent)들의 집합 $\{t_0, t_1, \cdots, t_{n-1}, t_n\}$이 주어질 때, 헤르밋 스플라인 곡선(Hermite Parametric Cubic Curves)은 모든 제어점들을 지나는 곡선이며, 제어점 p_k와 p_{k+1} 사이의 곡선은 다음과 같은 매개변수 $(0 \leq t \leq 1)$에 대한 3차 곡선이다.

$$q(t) = (2t^3 - 3t^2 + 1)p_k + (t^3 - 2t^2 + t)t_k + (-2t^3 + 3t^2)p_{k+1} + (t^3 - t^2)t_{k+1}$$

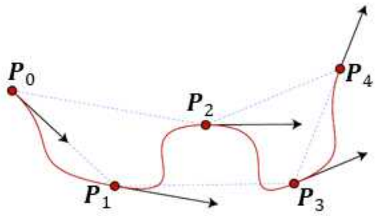


다음 함수 XMVectorCatmullRom()는 4개의 제어점(p0, p1, p2, p3)을 모두 지나는 매개변수 $(0 \le t \le 1)$에 대한 3차 매개변수 곡선 위의 점을 반환한다.

```
XMVECTOR XMVectorCatmullRom(XMVECTOR p0, XMVECTOR p1, XMVECTOR p2,
XMVECTOR p3, float t);
```

4개의 제어점 $\boldsymbol{p}_0$, $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{p}_3$을 지나는 CatmullRom 스플라인 곡선 $\boldsymbol{q}(t)$는 다음과 같다.

$$\boldsymbol{q}(t) = \left(-\frac{1}{2}t^3 + t^2 - \frac{1}{2}t\right)\boldsymbol{p}_0 + \left(\frac{3}{2}t^3 - \frac{5}{2}t^2 + 1\right)\boldsymbol{p}_1 + \left(-\frac{3}{2}t^3 + 2t^2 + \frac{1}{2}t\right)\boldsymbol{p}_2 + \left(\frac{1}{2}t^3 - \frac{1}{2}t^2\right)\boldsymbol{p}_3$$



n개의 제어점(Control point)들의 집합 $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_{n-1}, \boldsymbol{p}_n\}$이 주어질 때, CatmullRom 스플라인 곡선은 모든 제어점들을 지나는 곡선이며, 제어점 $\boldsymbol{p}_k$와 $\boldsymbol{p}_{k+1}$를 3차 곡선으로 표현한다. 각 제어점에서 이전 제어점과 다음 제어점을 연결한 선분을 접선으로 사용하여 곡선을 정의한다. 제어점 $\boldsymbol{p}_k$의 접선 $\boldsymbol{t}_k$은 다음과 같다.

$$\boldsymbol{t}_k = \frac{1}{2}(\boldsymbol{p}_{k+1} - \boldsymbol{p}_{k-1})$$

$\boldsymbol{p}_{k-1}$, $\boldsymbol{p}_k$, $\boldsymbol{p}_{k+1}$, $\boldsymbol{p}_{k+2}$를 사이의 곡선은 다음과 같은 매개변수 t에 대한 3차 곡선이다.

$$\boldsymbol{q}(t) = \left(-\frac{1}{2}t^3 + t^2 - \frac{1}{2}t\right)\boldsymbol{p}_0 + \left(\frac{3}{2}t^3 - \frac{5}{2}t^2 + 1\right)\boldsymbol{p}_1 + \left(-\frac{3}{2}t^3 + 2t^2 + \frac{1}{2}t\right)\boldsymbol{p}_2 + \left(\frac{1}{2}t^3 - \frac{1}{2}t^2\right)\boldsymbol{p}_3$$

다음 함수 XMVectorInBounds()는 점(v)이 바운드(bounds) 내부에 있는 가를 검사한 결과를 벡터로 반환한다.

```
XMVECTOR XMVectorInBounds(XMVECTOR v, XMVECTOR bounds);
```

벡터 $\boldsymbol{v} = (v_x, v_y, v_z, v_w)$와 벡터 $\boldsymbol{bounds} = (b_x, b_y, b_z, b_w)$에 대한 다음 계산의 결과를 벡터 $\boldsymbol{r}$로 반환한다.

$$r_x = (-b_x \le v_x \le b_x)\ ?\ 0\mathrm{xffffffff} : 0$$

$$r_y = (-b_y \le v_y \le b_y)\ ?\ 0\mathrm{xffffffff} : 0$$

$$r_z = (-b_z \le v_z \le b_z)\ ?\ 0\mathrm{xffffffff} : 0$$

$$r_w = (-b_w \leq v_w \leq b_w)\ ?\ 0xffffffff : 0$$

다음 함수 XMVectorInBoundsR()는 XMVectorInBounds()와 같은 검사를 수행한다. 반환 벡터의 모든 요소의 값이 0xffffffff이면 XM_CRMASK_CR6BOUNDS를 결과(pCR)로 반환한다.

```
XMVECTOR XMVectorInBoundsR(uint32_t *pCR, XMVECTOR v, XMVECTOR
bounds);
```

다음 함수 XMComparisonAllInBounds()는 함수 XMVectorInBoundsR()의 호출에서 반환한 결과(pCR)가 XM_CRMASK_CR6BOUNDS이면 참을 반환한다.

```
bool XMComparisonAllInBounds(uint32_t CR);
```

⑬ 3D 벡터 연산 함수

다음은 3차원 벡터 $\boldsymbol{v} = (v_x, v_y, v_z)$에 대한 벡터의 연산을 수행하는 함수들이다. 함수의 입력 벡터가 4차원 벡터(XMVECTOR) $\boldsymbol{v} = (v_x, v_y, v_z, v_w)$로 주어지면 3차원 벡터 $(v_x, v_y, v_z) = (v_x, v_y, v_z, 0)$로 바꾸어 연산을 수행하며, 4차원 벡터(XMVECTOR)를 반환한다. 벡터 연산의 결과가 3차원 벡터일 때, 반환되는 4차원 벡터의 w-요소는 무시하라. 벡터 연산의 결과가 스칼라 c일 때, 스칼라 c가 반환되는 4차원 벡터의 모든 요소에 반복되어 (c, c, c, c) 형태로 반환된다.

다음 함수 XMVector3AngleBetweenNormals()는 두 개의 3차원 법선 벡터(n1, n2)의 각도(라디안)를 반환한다. XMVector3AngleBetweenNormalsEst()는 각도의 근사값을 반환한다. 벡터(n1, n2)는 반드시 단위벡터이어야 한다.

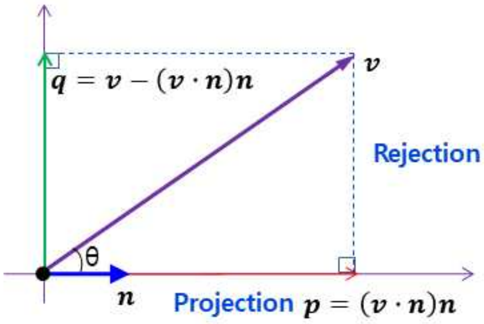
```
XMVECTOR XMVector3AngleBetweenNormals(XMVECTOR n1, XMVECTOR n2);
XMVECTOR XMVector3AngleBetweenNormalsEst(XMVECTOR n1, XMVECTOR n2);
```

다음 함수 XMVector3AngleBetweenVectors()는 두 개의 3차원 법선 벡터(v1, v2)의 각도(라디안)를 반환한다. 벡터(v1, v2)는 단위벡터가 아니어도 된다.

```
XMVECTOR XMVector3AngleBetweenVectors(XMVECTOR v1, XMVECTOR v2);
```

다음 함수 XMVector3ComponentsFromNormal()는 벡터(v)를 벡터(normal)에 평행한 벡터(parallel)와 수직인 벡터(perpendicular)로 분해한다(직교 투영).

```
void XMVector3ComponentsFromNormal(XMVECTOR *parallel,
*perpendicular, XMVECTOR v, XMVECTOR normal);
```



다음 함수 XMVector3ClampLength()는 벡터(v)의 길이가 최소(min)와 최대(max) 사이의 값이 되도록 벡터를 수정하여 반환한다.

```
XMVECTOR XMVector3ClampLength(XMVECTOR v, float min, float max);
```

다음 함수 XMVector3Normalize()은 3차원 벡터(v)를 정규화하여 단위벡터(Unit vector)를 반환한다. 3차원 벡터(v)의 크기가 0 또는 ∞이면 QNaN의 벡터를 반환한다. 함수 XMVector3NormalizeEst()는 단위벡터의 근사값을 반환한다.

```
XMVECTOR XMVector3Normalize(XMVECTOR v);
```

다음 함수 XMVector3Cross()는 두 개의 3차원 벡터(v1, v2)의 외적 연산의 결과를 반환한다.

```
XMVECTOR XMVector3Cross(XMVECTOR v1, XMVECTOR v2);
```

다음 함수 XMVector3Dot()는 두 개의 3차원 벡터(v1, v2)의 내적 연산의 결과를 반환한다.

```
XMVECTOR XMVector3Dot(XMVECTOR v1, XMVECTOR v2);
```

다음 함수 XMVector3Orthogonal()는 3차원 벡터(v)에 수직인 벡터를 반환한다.

```
XMVECTOR XMVector3Orthogonal(XMVECTOR v);
```

다음 함수 XMVector3Length()은 3차원 벡터(v)의 크기 l을 (l, l, l, l)의 형태로 반환한다. 함수 XMVector3LengthEst()는 3차원 벡터(v)의 근사(Estimated) 크기를 반환한다. 함수 XMVector3LengthSq()는 3차원 벡터(v)의 크기의 제곱을 반환한다. 함수 XMVector3ReciprocalLength()는 3차원 벡터(v)의 크기의 역수를 반환한다.

```
XMVECTOR XMVector3Length(XMVECTOR v);
XMVECTOR XMVector3LengthEst(XMVECTOR v);
XMVECTOR XMVector3LengthSq((XMVECTOR v);
XMVECTOR XMVector3ReciprocalLength(XMVECTOR v);
```

```
XMVECTOR XMVector3ReciprocalLengthEst(XMVECTOR v);
```

다음 함수 XMVector3LinePointDistance()는 두 개의 3차원 점 벡터(p1, p2)를 지나는 직선과 점(point) 사이의 최소 거리를 반환한다.

```
XMVECTOR XMVector3LinePointDistance(XMVECTOR p1, XMVECTOR p2,
XMVECTOR point);
```

다음 함수 XMVector3Reflect()는 법선 벡터가 $\boldsymbol{n}$인 평면에 대하여 3차원 입사 벡터($\boldsymbol{v}$)의 반사 벡터 $\boldsymbol{r}$을 반환한다($\boldsymbol{r} = \boldsymbol{v} - 2(\boldsymbol{v} \cdot \boldsymbol{n})\boldsymbol{n}$).

```
XMVECTOR XMVector3Reflect(XMVECTOR v, XMVECTOR n);
```

다음 함수 XMVector3Refract()는 법선 벡터가 $\boldsymbol{n}$인 평면에 대하여 3차원 입사 벡터($\boldsymbol{v}$)의 굴절 벡터 $\boldsymbol{r}$을 반환한다.

```
XMVECTOR XMVector3Refract(XMVECTOR v, XMVECTOR n, float lambda);
XMVECTOR XMVector3RefractV(XMVECTOR v, XMVECTOR n, XMVECTOR lambda);
```

굴절률이 λ일 때, 굴절 벡터 $\boldsymbol{r}$은 다음과 같이 계산한다.

$$r = 1 - \lambda^2 (1 - (\boldsymbol{v} \cdot \boldsymbol{n})^2)$$

$(r < 0)$이면, 완전한 내부 반사(Internal reflection)이므로 굴절 벡터는 $(0, 0, 0, 0)$이다.

$(r \geq 0)$이면, 굴절 벡터 $\boldsymbol{r}$은 다음과 같다.

$$s = \lambda(\boldsymbol{v} \cdot \boldsymbol{n}) + \sqrt{r}$$

$$\boldsymbol{r} = (\lambda v_x - sn_x, \lambda v_y - sn_y, \lambda v_z - sn_z)$$

다음 함수 XMVector3InBounds()는 벡터 $\boldsymbol{v} = (v_x, v_y, v_z, v_w)$와 벡터 $\boldsymbol{b} = (b_x, b_y, b_z, b_w)$에 대한 다음 계산의 결과를 반환한다. 즉, 벡터 $\boldsymbol{v}$가 중심이 원점인 정육면체 내부의 점인가를 계산한다.

```
bool XMVector3InBounds(XMVECTOR v, XMVECTOR b);
```

$$(-b_x \leq v_x \leq b_x) \&\& (-b_y \leq v_y \leq b_y) \&\& (-b_z \leq v_z \leq b_z)$$

⑭ 3D 벡터 변환(Transformation) 함수

다음은 3차원 벡터 $\boldsymbol{v} = (v_x, v_y, v_z)$를 (4×4) 행렬 $\boldsymbol{m}$ 또는 쿼터니언을 사용하여 변환하는 함수들이다. 함수의 입력 벡터가 4차원 벡터 $\boldsymbol{v} = (v_x, v_y, v_z, v_w)$로 주어지면 점 벡터의 경우 $(v_x, v_y, v_z, 1)$로 바꾸어 변환하고, 방향(법선) 벡터의 경우 $(v_x, v_y, v_z, 0)$으로 바꾸어 변환한다.

다음 함수 XMVector3Transform()은 벡터 $\boldsymbol{v} = (v_x, v_y, v_z, v_w)$를 $(v_x, v_y, v_z, 1)$로 바꾸어

행렬 $\boldsymbol{m}$과 곱셈(변환)을 한 결과 벡터 $\boldsymbol{r}=(r_x, r_y, r_z, r_w)$를 반환한다. r_w가 1이 되도록 바꾸지 않는다(3차원 동차(Homogeneous) 좌표계로 변환(Homonizing)하지 않음).

```
XMVECTOR XMVector3Transform(XMVECTOR v, XMMATRIX m);
```

다음 함수 XMVector3TransformCoord()은 벡터 $\boldsymbol{v}=(v_x, v_y, v_z, v_w)$를 $(v_x, v_y, v_z, 1)$로 바꾸어 행렬 $\boldsymbol{m}$과 곱셈(변환)을 하고 3차원 동차 좌표계로 바꾸어 $\boldsymbol{r}=(r_x, r_y, r_z, 1)$를 반환한다. 입력 벡터 $\boldsymbol{v}$는 3차원 위치(Position) 벡터이다.

```
XMVECTOR XMVector3TransformCoord(XMVECTOR v, XMMATRIX m);
```

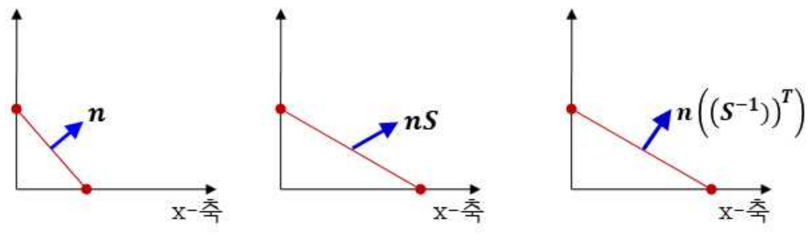
다음 함수 XMVector3TransformNormal()은 벡터 $\boldsymbol{v}=(v_x, v_y, v_z, v_w)$를 $(v_x, v_y, v_z, 0)$로 바꾸어 행렬 $\boldsymbol{m}$과 곱셈(변환)을 한 결과 벡터 $\boldsymbol{r}=(r_x, r_y, r_z, 0)$를 반환한다. 입력 벡터 $\boldsymbol{v}$는 3차원 방향(법선: Normal) 벡터이다. 법선 벡터를 변환할 때는 변환 행렬이 역행렬의 전치 행렬이 되어야 한다.

```
XMVECTOR XMVector3TransformNormal(XMVECTOR v, XMMATRIX m);
```

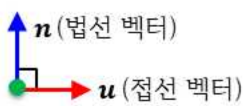
- 법선 벡터의 변환

  정점이 변환되면 위치 벡터 뿐 아니라 법선 벡터 $\boldsymbol{n}$도 변환된다. 변환 행렬 $\boldsymbol{m}$에서 x-축, y-축, -z축의 크기 변환(스케일링: Scaling) 양이 다르면, 법선 벡터 $\boldsymbol{n}$을 변환 행렬 $\boldsymbol{m}$으로 곱셈을 한 결과가 더 이상 법선 벡터가 되지 않는다.

  다음 그림에서 크기 변환 행렬 $\boldsymbol{S}$가 x-축으로 2배, y-축과 z-축으로 1배 만큼 크기 변환을 한다고 가정하자. 법선 벡터 $\boldsymbol{n}=(n_x, n_y, n_z, 0)$의 변환 행렬 $\boldsymbol{S}$로 변환한 결과는 $\boldsymbol{nS}=(2n_x, n_y, n_z, 0)$가 된다. 법선 벡터 $\boldsymbol{n}$과 법선벡터 $\boldsymbol{nS}$는 방향이 다르다.



  변환 행렬 $\boldsymbol{S}$에서 x-축, y-축, z-축의 크기 변환 양이 같으면, 변환된 법선 벡터 $\boldsymbol{nS}$의 방향은 법선 벡터 $\boldsymbol{n}$의 방향과 같지만 단위벡터가 되지 않을 수 있다(정규화 해야 함).



  다각형의 면 위에 있는 임의의 벡터 $\boldsymbol{u}$(접선 벡터: Tangent vector)와 다각형의 법선

벡터 $\boldsymbol{n}$은 서로 수직이다. 즉, $\boldsymbol{u} \bullet \boldsymbol{n} = 0$이다. 그리고 벡터의 내적 연산은 다음과 같이 행렬의 곱으로 표현할 수 있다.

$$\boldsymbol{u} \bullet \boldsymbol{n} = (u_x, u_y, u_z) \bullet (n_x, n_y, n_z) = (u_x n_x + u_y n_y + u_z n_z) = (u_x, u_y, u_z)(n_x, n_y, n_z)^T$$

$$\boldsymbol{u} \bullet \boldsymbol{n} = (u_x n_x + u_y n_y + u_z n_z) = (u_x, u_y, u_z)\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$$

이제 $\boldsymbol{u} \bullet \boldsymbol{n} = 0$을 행렬의 곱으로 표현하고, 변환 행렬을 $\boldsymbol{M}$이라고 하면 행렬의 성질을 사용하면 다음을 유도할 수 있다.

$$\boldsymbol{u} \bullet \boldsymbol{n} = \boldsymbol{u}\boldsymbol{n}^T = 0$$

$$\boldsymbol{u}\boldsymbol{n}^T = \boldsymbol{u}(\boldsymbol{M}\boldsymbol{M}^{-1})\boldsymbol{n}^T = (\boldsymbol{u}\boldsymbol{M})(\boldsymbol{M}^{-1}\boldsymbol{n}^T) = (\boldsymbol{u}\boldsymbol{M})(\boldsymbol{n}((\boldsymbol{M}^{-1})^T))^T = (\boldsymbol{u}\boldsymbol{M}) \bullet (\boldsymbol{n}((\boldsymbol{M}^{-1})^T)) = 0$$

$$\boldsymbol{u} \bullet \boldsymbol{n} = (\boldsymbol{u}\boldsymbol{M}) \bullet (\boldsymbol{n}((\boldsymbol{M}^{-1})^T)) = 0$$

위의 식은 다각형의 정점의 위치 벡터가 행렬 $\boldsymbol{M}$으로 변환되면 접선 벡터 $\boldsymbol{u}$는 $\boldsymbol{u}\boldsymbol{M}$으로 변환될 것이고, 변환된 접선 벡터 $\boldsymbol{u}\boldsymbol{M}$에 수직인 벡터(변환된 법선 벡터)는 $\boldsymbol{n}((\boldsymbol{M}^{-1})^T)$임을 보여준다. 즉, 법선 벡터 $\boldsymbol{n}$은 행렬 $\boldsymbol{M}$의 역행렬의 전치행렬 $(\boldsymbol{M}^{-1})^T$로 변환되어야 한다.

다음 함수 XMVector3TransformCoordStream()은 3차원 위치 벡터들의 배열(스트림)의 각 원소들을 행렬 $\boldsymbol{m}$과 곱셈(변환)을 하고 3차원 동차 좌표계로 바꾸어 스트림으로 반환한다.

```
XMFLOAT3* XMVector3TransformCoordStream(XMFLOAT3 *pOutputStream,
size_t outputStride, XMFLOAT3 *pInputStream, size_t inputStride,
size_t vectorCount, XMMATRIX m);
```

다음 함수 XMVector3TransformNormalStream()은 3차원 법선 벡터들의 배열(스트림)의 각 원소들을 행렬 $\boldsymbol{m}$과 곱셈(변환)을 하여 스트림(배열)으로 반환한다.

```
XMFLOAT3* XMVector3TransformNormalStream(XMFLOAT3 *pOutputStream,
size_t outputStride, XMFLOAT3 *pInputStream, size_t inputStride,
size_t count, XMMATRIX m);
```

다음 함수 XMVector3Rotate()는 3차원 벡터(v)를 쿼터니언(quaternion)을 사용하여 변환(회전)한 결과를 반환한다. 함수 XMVector3InverseRotate()는 3차원 벡터를 쿼터니언의 역(Inverse)을 사용하여 변환(회전)한 결과를 반환한다.

```
XMVECTOR XMVector3Rotate(XMVECTOR v, XMVECTOR quaternion);
XMVECTOR XMVector3InverseRotate(XMVECTOR v, XMVECTOR quaternion);
```

다음 함수 XMVector3Project()는 모델 좌표계의 3차원 벡터(v)를 월드 변환 행렬(world), 카메라 변환 행렬(view), 투영 변환 행렬(projection), 뷰포트(viewportX ~

viewportMaxZ)를 사용하여 화면 좌표계로 변환한 결과를 반환한다.

```
XMVECTOR XMVector3Project(XMVECTOR v, float viewportX, float
viewportY, float viewportWidth, float viewportHeight, float
viewportMinZ, float viewportMaxZ, XMMATRIX projection, XMMATRIX
view, XMMATRIX world);
```

다음 함수 XMVector3Unproject()는 화면 좌표계의 2차원 벡터(v)를 월드 변환 행렬(world), 카메라 변환 행렬(view), 투영 변환 행렬(projection), 뷰포트(viewportX ~ viewportMaxZ)를 사용하여 모델 좌표계로 변환한 결과를 반환한다.

```
XMVECTOR XMVector3Unproject(XMVECTOR v, float viewportX, float
viewportY, float viewportWidth, float viewportHeight, float
viewportMinZ, float viewportMaxZ, XMMATRIX projection, XMMATRIX
view, XMMATRIX world);
```

다음 함수 XMVector3ProjectStream()은 3차원 모델 좌표계의 벡터들의 배열(스트림)의 각 원소들을 화면 좌표계로 변환한 배열(스트림)을 반환한다.

```
XMFLOAT3* XMVector3ProjectStream(XMFLOAT3 *pOutputStream, UINT
outputStride, XMFLOAT3 *pInputStream, UINT inputStride, UINT
vectorCount, float viewportX, float viewportY, float viewportWidth,
float viewportHeight, float viewportMinZ, float viewportMaxZ,
XMMATRIX projection, XMMATRIX view, XMMATRIX world);
```

다음 함수 XMVector3UnprojectStream()은 2차원 화면 좌표계의 벡터들의 배열(스트림)의 각 원소들을 모델 좌표계로 변환한 배열(스트림)을 반환한다.

```
XMFLOAT3* XMVector3UnprojectStream(XMFLOAT3 *pOutputStream, size_t
outputStride, XMFLOAT3 *pInputStream, size_t inputStride, size_t
vectorCount, float viewportX, float viewportY, float viewportWidth,
float viewportHeight, float viewportMinZ, float viewportMaxZ,
XMMATRIX projection, XMMATRIX view, XMMATRIX world);
```

⑮ 행렬(Matrix) 함수

다음은 (4×4) 행렬의 초기화와 연산을 수행하는 함수들이다.

다음 함수 XMMatrixSet()은 (4×4) 행렬을 실수 16개로 초기화하여 반환한다. 실수 16개는 순서대로 4개씩 행렬의 행을 나타낸다. DirectXMath의 행렬은 행우선 행렬이다.

```
XMMATRIX XMMatrixSet(float m00, m01, m02, m03, m10, ..., m32, m33);
```

다음 함수 XMMatrixIdentity()는 (4×4) 단위행렬을 반환한다.

```
XMMATRIX XMMatrixIdentity();
```

다음 함수 XMMatrixIsIdentity()는 (4×4) 행렬(m)이 단위행렬인 가를 반환한다. 함수 XMMatrixIsInfinite()는 (4×4) 행렬(m)의 어떤 요소라도 ∞이면 참을 반환한다. 함수 XMMatrixIsNaN()는 (4×4) 행렬(m)의 어떤 요소라도 NaN(Not a Number)이면 참을 반환한다.

```
bool XMMatrixIsIdentity(XMMATRIX m);
bool XMMatrixIsInfinite(XMMATRIX m);
bool XMMatrixIsNaN(XMMATRIX m);
```

다음 함수 XMMatrixInverse()는 (4×4) 행렬(m)의 역행렬과 행렬식(determinant)을 반환한다. 함수 XMMatrixDeterminant()는 행렬(m)의 행렬식을 반환한다.

```
XMMATRIX XMMatrixInverse(XMVECTOR *determinant, XMMATRIX m);
XMVECTOR XMMatrixDeterminant(XMMATRIX m);
```

다음 함수 XMMatrixTranspose()는 (4×4) 행렬(m)의 전치행렬을 반환한다.

```
XMMATRIX XMMatrixTranspose(XMMATRIX m);
```

다음 함수 XMMatrixMultiply()는 두 개의 (4×4) 행렬(m1, m2)의 곱셈의 결과를 반환한다. 함수 XMMatrixMultiplyTranspose()는 두 개의 (4×4) 행렬(m1, m2)의 곱셈의 결과의 전치행렬을 반환한다.

```
XMMATRIX XMMatrixMultiply(XMMATRIX m1, m2);
XMMATRIX XMMatrixMultiplyTranspose(XMMATRIX m1, m2);
```

다음 함수 XMMatrixRotationX()는 x-축을 중심으로 라디안 각도(angle)만큼 회전 변환하는 행렬을 반환한다. 왼손좌표계에서 회전각은 좌표축의 화살표 위치(+∞)에서 원점을 바라볼 때 시계 방향을 양(+)의 방향으로 측정한다. XMMatrixRotationY는 y-축을 중심으로 회전하는 행렬을 반환하고, XMMatrixRotationZ는 z-축을 중심으로 회전하는 행렬을 반환한다. 회전 행렬의 4번째 행과 열은 $(0, 0, 0, 1)$이다.

```
XMMATRIX XMMatrixRotationX(float angle);
XMMATRIX XMMatrixRotationY(float angle);
XMMATRIX XMMatrixRotationZ(float angle);
```

x-축을 중심으로 θ만큼 회전을 하는 변환 행렬 $\boldsymbol{R}_x(\theta)$, y-축을 중심으로 θ만큼 회전을 하는 변환 행렬 $\boldsymbol{R}_y(\theta)$, z-축을 중심으로 θ만큼 회전을 하는 변환 행렬 $\boldsymbol{R}_z(\theta)$는 다음과 같다.

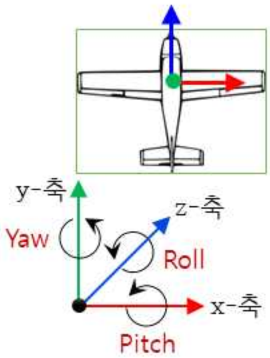
$$\boldsymbol{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

다음 함수 XMMatrixRotationRollPitchYaw()는 x-축 라디안 회전각(pitch), y-축 라디안 회전각(yaw), z-축 라디안 회전각(roll)이 주어질 때(오일러 각도: Euler angles), 먼저 z-축을 중심으로 roll 만큼 회전하고, 그 다음에 x-축을 중심으로 pitch 만큼 회전하고, 마지막으로 y-축을 중심으로 yaw 만큼 회전하는 회전 행렬을 반환한다. 회전 행렬의 4번째 행과 열은 $(0, 0, 0, 1)$이다. 함수 XMMatrixRotationRollPitchYawFromVector()는 오일러 회전각이 벡터로 주어진다.

```
XMMATRIX XMMatrixRotationRollPitchYaw(float pitch, float yaw, float
roll);
XMMATRIX XMMatrixRotationRollPitchYawFromVector(XMVECTOR Angles);
```



순차적으로 z-축을 중심으로 α만큼 회전하고, x-축을 중심으로 β만큼 회전하고, y-축을 중심으로 γ만큼 회전하는 회전 행렬 R_{zxy}는 z-축을 중심으로 α만큼 회전하는 행렬 $R_z(\alpha)$, x-축을 중심으로 β만큼 회전을 하는 행렬 $R_x(\beta)$, y-축을 중심으로 γ만큼 회전을 하는 변환 행렬 $R_y(\gamma)$의 곱과 같다.

$$R_{zxy} = R_z(\alpha) R_x(\beta) R_y(\gamma)$$

다음 함수 XMMatrixRotationAxis()는 임의의 회전축 벡터(axis)를 중심으로 라디안 각도(angle)만큼 회전하는 회전 행렬을 반환한다. 함수 XMMatrixRotationNormal()은 회전축 벡터(axis)가 단위 벡터일 때 사용한다. 회전 행렬의 4번째 행과 열은 $(0, 0, 0, 1)$이다.

```
XMMATRIX XMMatrixRotationAxis(XMVECTOR axis, float angle);
XMMATRIX XMMatrixRotationNormal(XMVECTOR axis, float angle);
```

회전축 (x, y, z)를 중심으로 θ만큼 회전하는 회전 행렬 R은 다음과 같다.

$$R = \begin{bmatrix} (1-\cos(\theta))x^2+\cos(\theta) & (1-\cos(\theta))xy+\sin(\theta)z & (1-\cos(\theta))xz-\sin(\theta)y \\ (1-\cos(\theta))xy-\sin(\theta)z & (1-\cos(\theta))y^2+\cos(\theta) & (1-\cos(\theta))yz+\sin(\theta)x \\ (1-\cos(\theta))xz+\sin(\theta)y & (1-\cos(\theta))yz-\sin(\theta)x & (1-\cos(\theta))z^2+\cos(\theta) \end{bmatrix}$$

다음 함수 XMMatrixRotationQuaternion()는 쿼터니언(Quaternion, q)가 표현하는 회전 행렬을 반환한다. 회전 행렬의 4번째 행과 열은 $(0,0,0,1)$이다.

```
XMMATRIX XMMatrixRotationQuaternion(XMVECTOR q);
```

쿼터니언 $\boldsymbol{q}=(x,y,z,w)$가 표현하는 회전을 행렬 $\boldsymbol{R}$로 표현하면 다음과 같다.

$$R = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2zw & 2xz-2yw \\ 2xy-2zw & 1-2x^2-2z^2 & 2yz+2xw \\ 2xz+2yw & 2yz-2xw & 1-2x^2-2y^2 \end{bmatrix}$$

다음 함수 XMMatrixTranslation()는 x-축으로 a만큼, y-축으로 b만큼 z-축으로 c만큼 이 평행 이동 변환을 하는 행렬을 반환한다. XMMatrixTranslationFromVector() 함수는 평행 이동의 양이 벡터이다.

```
XMMATRIX XMMatrixTranslation(float a, float b, float c);
XMMATRIX XMMatrixTranslationFromVector(XMVECTOR offset);
```

x-축으로 a만큼, y-축으로 b만큼 z-축으로 c만큼 평행 이동 변환을 하는 행렬 $\boldsymbol{T}$는 다음과 같다.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}$$

다음 함수 XMMatrixScaling()는 x-축으로 a만큼, y-축으로 b만큼 z-축으로 c만큼 크기 변환을 하는 행렬을 반환한다. XMMatrixScalingFromVector() 함수는 크기 변환의 양이 벡터이다.

```
XMMATRIX XMMatrixScaling(float a, float b, float c);
XMMATRIX XMMatrixScalingFromVector(XMVECTOR scale);
```

x-축으로 a만큼, y-축으로 b만큼 z-축으로 c만큼 크기 변환을 하는 행렬 $\boldsymbol{S}$는 다음과 같다.

$$S = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

다음 함수 XMMatrixAffineTransformation()는 크기 변환을 나타내는 3차원 벡터(scaling), 회전의 중심을 나타내는 3차원 벡터(origin), 회전 쿼터니언(rotation), 평행이동

을 나타내는 3차원 벡터(translation)에 해당하는 아핀 변환 행렬을 반환한다.

```
XMMATRIX XMMatrixAffineTransformation(XMVECTOR scaling, XMVECTOR origin, XMVECTOR rotation, XMVECTOR translation);
```

다음 함수 XMMatrixTransformation()는 크기 변환의 중심을 나타내는 3차원 벡터(sOrigin), 크기 변환의 방향을 나타내는 3차원 벡터(qScale), 크기 변환의 양을 나타내는 3차원 벡터(scaling), 회전의 중심을 나타내는 3차원 벡터(rOrigin), 회전의 방향을 표현하는 쿼터니언(qRotation), 평행이동의 양을 나타내는 3차원 벡터(translation)에 해당하는 아핀 변환 행렬을 반환한다.

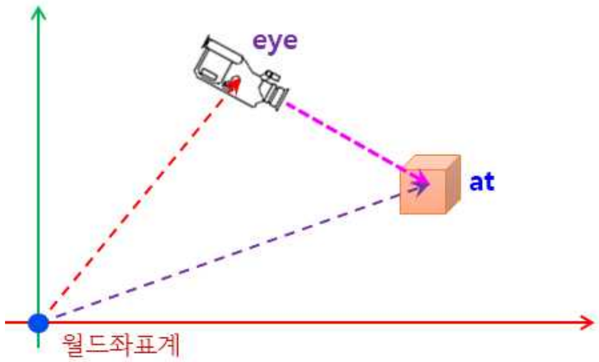
```
XMMATRIX XMMatrixTransformation(XMVECTOR sOrigin, XMVECTOR qScale, XMVECTOR scaling, XMVECTOR rOrigin, XMVECTOR qRotation, XMVECTOR translation);
```

다음 함수 XMMatrixDecompose()는 (4×4) 행렬(m)을 분해하여 크기 벡터(scaling), 회전 쿼터니언(rotation), 평행이동 벡터(translation)를 반환한다.

```
bool XMMatrixDecompose(XMVECTOR *scaling, XMVECTOR *rotation, XMVECTOR *translation, XMMATRIX m);
```

다음 함수 XMMatrixLookAtLH()는 월드 좌표계에서 카메라가 위치(eye)에서 한 점(at)을 바라보기 위한 왼손 좌표계의 카메라 변환 행렬을 반환한다. XMMatrixLookAtRH() 함수는 오른손 좌표계의 카메라 변환 행렬을 반환한다.

```
XMMATRIX XMMatrixLookAtLH(XMVECTOR eye, XMVECTOR at, XMVECTOR up);
```



$$look = normalize(at - eye)$$
$$right = normalize(up \times look)$$
$$up = normalize(look \times right)$$

right.x	up.x	look.x	0
right.y	up.y	look.y	0
right.z	up.z	look.z	0
-(eye • right)	-(eye • up)	-(eye • look)	1

다음 함수 XMMatrixLookToLH()는 월드 좌표계에서 카메라가 위치(eye)에서 주어진 방향(look)으로 바라보기 위한 카메라 변환 행렬을 반환한다.

```
XMMATRIX XMMatrixLookToLH(XMVECTOR eye, XMVECTOR look, XMVECTOR up);
```

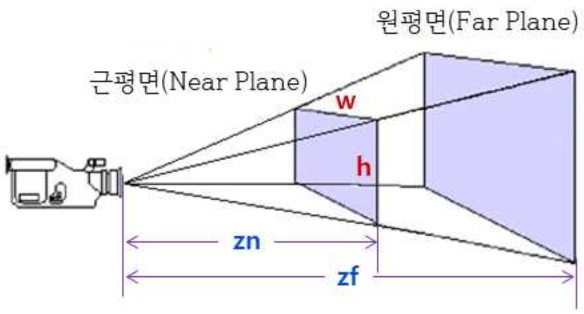
다음 함수 XMMatrixPerspectiveFovLH()는 카메라의 시야각(fov), 카메라에서 근평면까지 거리(zn), 카메라에서 원평면까지 거리(zf), 뷰포트의 종횡비(aspect)를 사용하여 원근 투영 변환 행렬 P를 반환한다.

```
XMMATRIX XMMatrixPerspectiveFovLH(float fov, float aspect, float zn,
float zf);
```

$$P = \begin{bmatrix} \frac{1}{tan(\theta) * aspect} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & \frac{z_f}{(z_f - z_n)} & 1 \\ 0 & 0 & \frac{-z_n z_f}{(z_f - z_n)} & 0 \end{bmatrix} \quad \theta = \frac{fov}{2}$$

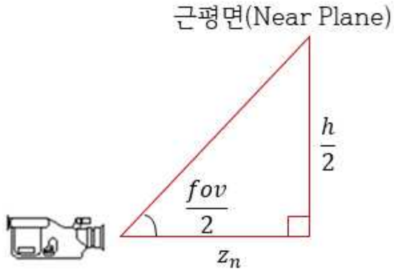
다음 함수 XMMatrixPerspectiveLH()는 다음 그림과 같이 근평면의 가로(w)와 세로(h), 카메라에서 근평면까지 거리(zn), 카메라에서 원평면까지 거리(zf)가 주어질 때 원근 투영 변환 행렬 P를 반환한다.

```
XMMATRIX XMMatrixPerspectiveLH(float w, float h, float zn, float zf);
```



$$P = \begin{bmatrix} \frac{2z_n}{w} & 0 & 0 & 0 \\ 0 & \frac{2z_n}{h} & 0 & 0 \\ 0 & 0 & \frac{z_f}{(z_f - z_n)} & 0 \\ 0 & 0 & \frac{-z_n z_f}{(z_f - z_n)} & 1 \end{bmatrix}$$

근평면의 가로(w)와 세로(h), 그리고 카메라에서 근평면까지 거리(zn)이 주어지면 삼각비로부터 다음이 성립한다. 이것은 카메라의 시야각(fov)이 주어질 때의 원근 투영 변환 행렬과 같음을 알 수 있다.

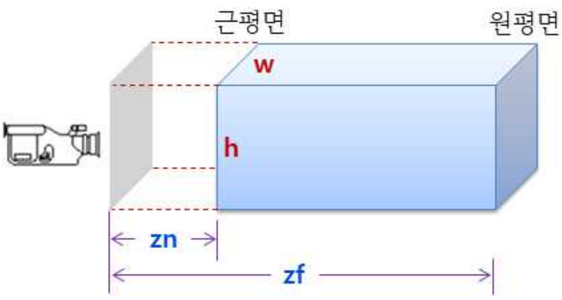


$$\frac{1}{tan\left(\frac{fov}{2}\right)} = \frac{1}{\left(\frac{h}{2z_n}\right)} = \frac{2\,z_n}{h}$$

$$\frac{1}{aspect * tan\left(\frac{fov}{2}\right)} = \frac{1}{\left(\frac{w}{h}\right)\left(\frac{h}{2z_n}\right)} = \frac{2\,z_n}{w}$$

다음 함수 XMMatrixOrthographicLH()는 다음 그림과 같이 투영 사각형의 가로(w)와 세로(h), 카메라에서 근평면까지 거리(zn), 카메라에서 원평면까지 거리(zf)가 주어질 때 직교 투영 변환 행렬 P를 반환한다. 직교 투영에서 같은 게임 객체는 카메라에서 멀고 가까운 것에 상관없이(거리에 상관없이) 같은 크기로 투영된다.

```
XMMATRIX XMMatrixOrthographicLH(float w, float h, float zn, float
zf);
```



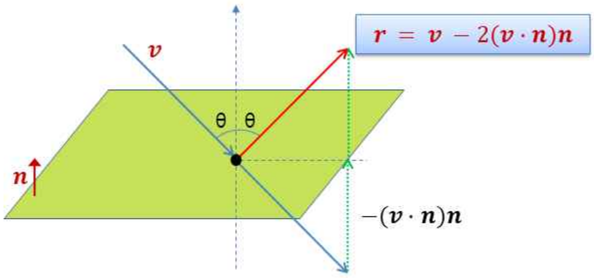
$$P = \begin{bmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{1}{(z_f - z_n)} & 0 \\ 0 & 0 & \frac{-z_n}{(z_f - z_n)} & 1 \end{bmatrix}$$

다음 함수 XMMatrixReflect()는 다음 그림과 같이 법선 벡터가 $\boldsymbol{n} = (a, b, c)$, 원점에서 평면까지의 거리가 d인 평면(plane)에 대하여 방향 벡터와 점 벡터를 반사시키는 반사 행렬(Reflection matrix) $\boldsymbol{R}$을 반환한다.

```
XMMATRIX XMMatrixReflect(XMVECTOR plane);
```

$$\boldsymbol{R} = \begin{bmatrix} 1-2a^2 & -2ab & -2ac & 0 \\ -2ab & 1-2b^2 & -2bc & 0 \\ -2ac & -2bc & 1-2c^2 & 0 \\ -2ad & -2bd & -2cd & 1 \end{bmatrix}$$

다음 그림은 방향 벡터 $\boldsymbol{v}$를 평면$(\boldsymbol{n}, d)$에 반사하는 경우이다.



방향 벡터 $\boldsymbol{v}$를 법선 벡터가 $\boldsymbol{n} = (n_x, n_y, n_z)$인 평면에 반사시킨 반사 벡터 $\boldsymbol{r}$은 다음과 같다.

$$\boldsymbol{r} = \boldsymbol{v} - 2(\boldsymbol{v} \bullet \boldsymbol{n})\boldsymbol{n}$$

이것은 방향 벡터 $\boldsymbol{v}$를 반사 행렬 $\boldsymbol{R}$으로 변환한 다음의 결과와 같음을 알 수 있다. 즉, 행렬 $\boldsymbol{R}$은 방향 벡터 $\boldsymbol{v}$를 법선 벡터가 $\boldsymbol{n} = (n_x, n_y, n_z)$인 평면에 반사시키는 행렬이다. 방향 벡터는 위치와 상관없으므로 반사 행렬 $\boldsymbol{R}$으로 변환할 때 방향 벡터 $\boldsymbol{v}$의 w-요소를 0으로 설정하여 변환한다.

$$\boldsymbol{v}\boldsymbol{R} = (v_x, v_y, v_z, 0)\begin{bmatrix} 1-2a^2 & -2ab & -2ac & 0 \\ -2ab & 1-2b^2 & -2bc & 0 \\ -2ac & -2bc & 1-2c^2 & 0 \\ -2ad & -2bd & -2cd & 1 \end{bmatrix} = \boldsymbol{r} = (r_x, r_y, r_z, 0)$$

$$r_x = v_x - 2v_x a^2 - 2v_y ab - 2v_z ac = v_x - 2a(v_x a + v_y b + v_z c) = v_x - 2a(\boldsymbol{n} \bullet \boldsymbol{v})$$

$$r_y = -2v_x ab + v_y - 2v_y b^2 - 2v_z bc = v_y - 2b(v_x a + v_y b + v_z c) = v_y - 2b(\boldsymbol{n} \bullet \boldsymbol{v})$$

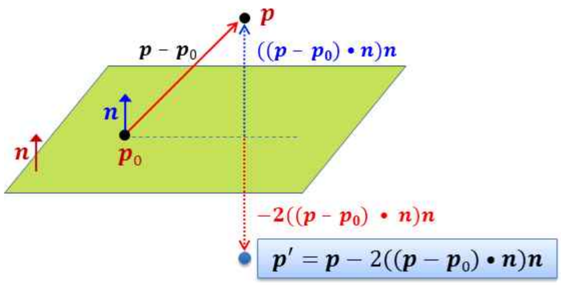
$$r_z = -2v_x ac - 2v_y bc + v_z - 2v_z c^2 = v_z - 2c(v_x a + v_y b + v_z c) = v_z - 2c(\boldsymbol{n} \bullet \boldsymbol{v})$$

$$\boldsymbol{r} = (r_x, r_y, r_z, 0) = (v_x - 2a(\boldsymbol{n} \bullet \boldsymbol{v}), v_y - 2b(\boldsymbol{n} \bullet \boldsymbol{v}), v_z - 2c(\boldsymbol{n} \bullet \boldsymbol{v}), 0)$$

$$(v_x - 2a(\boldsymbol{n} \bullet \boldsymbol{v}), v_y - 2b(\boldsymbol{n} \bullet \boldsymbol{v}), v_z - 2c(\boldsymbol{n} \bullet \boldsymbol{v}), 0) = (v_x, v_y, v_z, 0) - 2(\boldsymbol{n} \bullet \boldsymbol{v})(a, b, c, 0)$$

$$\boldsymbol{r} = (v_x, v_y, v_z, 0) - 2(\boldsymbol{n} \bullet \boldsymbol{v})(a, b, c, 0) = \boldsymbol{v} - 2(\boldsymbol{n} \bullet \boldsymbol{v})\boldsymbol{n}$$

다음 그림은 점 벡터 $\boldsymbol{p}$를 평면$(\boldsymbol{n}, d)$에 반사(평면에 대하여 대칭 이동)하는 경우이다. 점 $\boldsymbol{p}_0$는 평면 위의 한 점일 때 $d = -\boldsymbol{n} \bullet \boldsymbol{p}_0$이다.



위의 그림에서 점 벡터 $\boldsymbol{p}$를 평면$(\boldsymbol{n}, d)$에 반사시킨 점 $\boldsymbol{r}$은 다음과 같다. 벡터 $(\boldsymbol{p} - \boldsymbol{p}_0)$를 벡터 $\boldsymbol{n}$에 직교투영하면 $((\boldsymbol{p} - \boldsymbol{p}_0) \bullet \boldsymbol{n})\boldsymbol{n}$이다. 투영의 결과 벡터의 스칼라(-2) 곱을 하고 점 벡터 $\boldsymbol{p}$에 덧셈을 하면 구할 수 있다.

$$\boldsymbol{r} = \boldsymbol{p} - 2((\boldsymbol{p} - \boldsymbol{p}_0) \bullet \boldsymbol{n})\boldsymbol{n}$$

이것은 점 벡터 $\boldsymbol{p}$를 반사 행렬 $\boldsymbol{R}$으로 변환한 결과와 같음을 알 수 있다. 즉, 행렬 $\boldsymbol{R}$은 점 벡터 $\boldsymbol{p}$를 평면$(\boldsymbol{n}, d)$에 반사시키는 행렬이다. 점 벡터 $\boldsymbol{p}$를 반사 행렬 $\boldsymbol{R}$으로 변환할 때 점 벡터 $\boldsymbol{p}$의 w-요소를 1으로 설정하여 변환한다.

$$\boldsymbol{pR} = (p_x, p_y, p_z, 1)\begin{bmatrix} 1-2a^2 & -2ab & -2ac & 0 \\ -2ab & 1-2b^2 & -2bc & 0 \\ -2ac & -2bc & 1-2c^2 & 0 \\ -2ad & -2bd & -2cd & 1 \end{bmatrix} = \boldsymbol{r} = (r_x, r_y, r_z, 1)$$

$$r_x = p_x - 2p_x a^2 - 2p_y ab - 2p_z ac - 2ad = p_x - 2a(ap_x + bp_y + cp_z + d) = p_x - 2a(\boldsymbol{n} \bullet \boldsymbol{p} + d)$$

$$r_y = -2p_x ab + p_y - 2p_y b^2 - 2p_z bc - 2bd = p_y - 2b(ap_x + bp_y + cp_z + d) = p_y - 2b(\boldsymbol{n} \bullet \boldsymbol{p} + d)$$

$$r_z = -2p_x ac - 2p_y bc + p_z - 2p_z c^2 - 2cd = p_z - 2c(ap_x + bp_y + cp_z + d) = p_z - 2c(\boldsymbol{n} \bullet \boldsymbol{p} + d)$$

$$\boldsymbol{r} = (r_x, r_y, r_z, 1) = (p_x - 2a(\boldsymbol{n} \bullet \boldsymbol{p} + d), p_y - 2b(\boldsymbol{n} \bullet \boldsymbol{p} + d), p_z - 2c(\boldsymbol{n} \bullet \boldsymbol{p} + d), 1)$$

$$\boldsymbol{r} = (p_x, p_y, p_z, 1) - 2(\boldsymbol{n} \bullet \boldsymbol{p} + d)(a, b, c, 0)$$

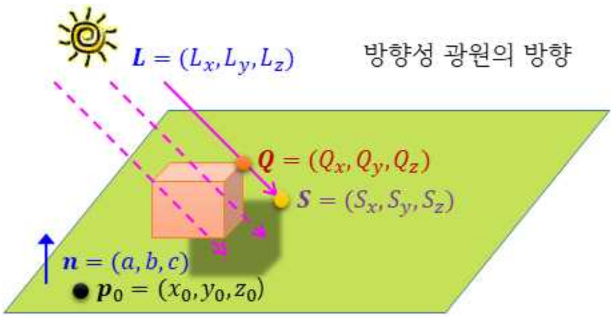
$$\boldsymbol{r} = \boldsymbol{p} - 2(\boldsymbol{n} \bullet \boldsymbol{p} + d)\boldsymbol{n}$$

$$\boldsymbol{p} = \boldsymbol{p} - 2((\boldsymbol{n} \bullet \boldsymbol{p}) - (\boldsymbol{n} \bullet \boldsymbol{p}_0))\boldsymbol{n} = \boldsymbol{p} - 2(\boldsymbol{n} \bullet (\boldsymbol{p} - \boldsymbol{p}_0))\boldsymbol{n}$$

다음 함수 XMMatrixShadow()는 법선 벡터가 $n=(a,b,c)$, 원점에서 평면까지의 거리가 d인 평면(plane)에 대하여 방향 벡터와 점 벡터(light)를 평면에 투영시키는 평면 투영 행렬(Planar projection matrix) S_{LP}를 반환한다. 평면 투영 행렬은 평면에 그림자를 그릴 때 사용할 수 있으므로 그림자 행렬(Shadow matrix)이라고도 한다.

```
XMMATRIX XMMatrixShadow(XMVECTOR plane, XMVECTOR light);
```

다음 그림에서 방향 벡터 L(광선)이 점 Q를 지나갈 때 평면과의 교점 S를 구하자. 만약 방향 벡터 L(광선)이 빛이라면 교점 S는 그림자 부분(어두운 부분)이 될 것이다. 교점 S는 점 Q를 평면에 투영한 점이다. 메쉬의 모든 정점(Q)들을 평면에 투영한 점들을 렌더링하면(어두운 색으로) 평면 위에 그림자가 그려질 것이다(3차원 메쉬를 2차원 다각형으로 투영한 것임).



방향 벡터 L(광선)이 점 Q를 지나는 광선의 매개변수 방정식은 다음과 같다.

$$r(t)=Q+tL$$

법선 벡터가 $n=(a,b,c)$이고 평면 위의 한 점 $p_0=(p_x,p_y,p_z)$이 주어질 때 평면의 방정식은 다음과 같다. 벡터 $v=(x,y,z)$는 평면 위의 임의의 점이다.

$$ax+by+cz+d=0$$

$$d=-(ax_0+by_0+cz_0)=-(n\cdot p_0)$$

$$(n\cdot v)-(n\cdot p_0)=0$$

점 Q를 지나는 방향 벡터 L(광선)과 평면의 교점 S는 평면 위의 점이므로 평면의 방정식과 광선의 방정식을 모두 만족한다.

$$(n\cdot S)-(n\cdot p_0)=0$$

$$r(t)=S=Q+tL$$

$S=Q+tL$를 $(n\cdot S)-(n\cdot p_0)=0$에 대입하면 다음을 얻는다.

$$(n\cdot S)-(n\cdot p_0)=(n\cdot (Q+tL))-(n\cdot p_0)=0$$

$$(n\cdot (Q+tL))-(n\cdot p_0)=(n\cdot Q)+t(n\cdot L)-(n\cdot p_0)=0$$

$$t(n\cdot L)=(n\cdot p_0)-(n\cdot Q)$$

$$t=\frac{(n\cdot p_0)-(n\cdot Q)}{(n\cdot L)}$$

t를 $r(t)=S=Q+tL$에 대입하면 교점 S를 구할 수 있다.

$$S = Q + tL = Q + \left\{ \frac{(n \bullet p_0) - (n \bullet Q)}{(n \bullet L)} \right\} L$$

$$S = Q + tL = \frac{1}{(n \bullet L)} ((n \bullet L)Q + \{(n \bullet p_0) - (n \bullet Q)\} L)$$

$$S = \frac{1}{(n \bullet L)} ((n \bullet L)Q - (n \bullet (Q - p_0))L)$$

교점 3차원 벡터 $S = (S_x, S_y, S_z)$는 다음과 같은 벡터이다.

$$S = \begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \frac{1}{(n \bullet L)} \begin{bmatrix} (n \bullet L)Q_x - (n \bullet (Q - p_0))L_x \\ (n \bullet L)Q_y - (n \bullet (Q - p_0))L_y \\ (n \bullet L)Q_z - (n \bullet (Q - p_0))L_z \end{bmatrix}$$

$$S = \begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \begin{bmatrix} \frac{1}{(n \bullet L)} ((n \bullet L)Q_x - (n \bullet (Q - p_0))L_x) \\ \frac{1}{(n \bullet L)} ((n \bullet L)Q_y - (n \bullet (Q - p_0))L_y) \\ \frac{1}{(n \bullet L)} ((n \bullet L)Q_z - (n \bullet (Q - p_0))L_z) \end{bmatrix}$$

3차원 벡터 $S = (S_x, S_y, S_z)$는 다음과 같은 4차원 벡터 $H = (H_x, H_y, H_z, (n \bullet L))$와 동차이고 $d = -(n \bullet p_0)$이다.

$$H = \begin{bmatrix} H_x \\ H_y \\ H_z \\ (n \bullet L) \end{bmatrix} = \begin{bmatrix} (n \bullet L)Q_x - (n \bullet (Q - p_0))L_x \\ (n \bullet L)Q_y - (n \bullet (Q - p_0))L_y \\ (n \bullet L)Q_z - (n \bullet (Q - p_0))L_z \\ (n \bullet L) \end{bmatrix}$$

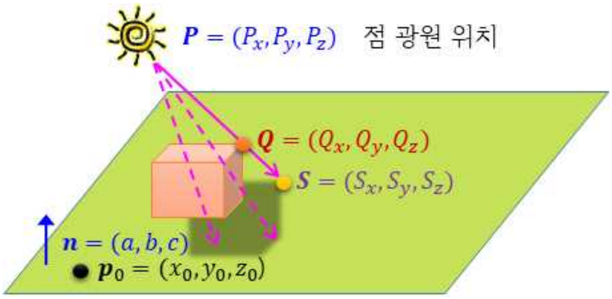
$$H = \begin{bmatrix} H_x \\ H_y \\ H_z \\ (n \bullet L) \end{bmatrix} = \begin{bmatrix} (n \bullet L)Q_x - ((n \bullet Q) + d))L_x \\ (n \bullet L)Q_y - ((n \bullet Q) + d))L_y \\ (n \bullet L)Q_z - ((n \bullet Q) + d))L_z \\ (n \bullet L) \end{bmatrix}$$

$$H = (Q_x, Q_y, Q_z, 1) \begin{bmatrix} (n \bullet L) - aL_x & -aL_y & -aL_z & 0 \\ -bL_x & (n \bullet L) - bL_y & -bL_z & 0 \\ -cL_x & -cL_y & (n \bullet L) - cL_z & 0 \\ -dL_x & -dL_y & -dL_z & (n \bullet L) \end{bmatrix}$$

점 Q를 지나는 방향 벡터 L(광선)과 평면의 교점 S는 점 Q를 다음 행렬 S_L로 변환하고 3차원 동차좌표계로 변환하여 구할 수 있다. 행렬 S_L를 방향 벡터 L(광선)에 대한 평면 투영 행렬(Planar projection matrix)이라고 한다.

$$S_L = \begin{bmatrix} (n \bullet L) - aL_x & -aL_y & -aL_z & 0 \\ -bL_x & (n \bullet L) - bL_y & -bL_z & 0 \\ -cL_x & -cL_y & (n \bullet L) - cL_z & 0 \\ -dL_x & -dL_y & -dL_z & (n \bullet L) \end{bmatrix}$$

다음 그림에서 점 P를 지나는 광선이 점 Q를 지나갈 때 평면과의 교점 S를 구하자. 만약 광선이 빛이라면 교점 S는 그림자 부분(어두운 부분)이 될 것이다.



점 P와 Q를 지나는 광선의 매개변수 방정식은 다음과 같다.

$$r(t)=tQ+(1-t)P$$

법선 벡터가 $n=(a,b,c)$이고 평면 위의 한 점 $p_0=(p_x,p_y,p_z)$이 주어질 때 평면의 방정식은 다음과 같다. 벡터 $v=(x,y,z)$는 평면 위의 임의의 점이다.

$$(n\bullet v)-(n\bullet p_0)=0$$

점 P와 Q를 지나는 광선과 평면의 교점 S는 평면 위의 점이므로 평면의 방정식과 광선의 방정식을 모두 만족한다.

$$(n\bullet S)-(n\bullet p_0)=0$$

$$r(t)=S=tQ+(1-t)P$$

$S=tQ+(1-t)P$를 $(n\bullet S)-(n\bullet p_0)=0$에 대입하면 다음을 얻는다.

$$(n\bullet S)-(n\bullet p_0)=(n\bullet (tQ+(1-t)P))-(n\bullet p_0)=0$$

$$t(n\bullet Q)+(1-t)(n\bullet P)-(n\bullet p_0)=0$$

$$t(n\bullet Q)+(n\bullet P)-t(n\bullet P)-(n\bullet p_0)=0$$

$$((n\bullet Q)-(n\bullet P))t=(n\bullet p_0)-(n\bullet P)$$

$$t=\frac{(n\bullet p_0)-(n\bullet P)}{(n\bullet Q)-(n\bullet P)}=\frac{(n\bullet p_0)-(n\bullet P)}{n\bullet (Q-P)}$$

t를 $r(t)=S=tQ+(1-t)P$에 대입하면 교점 S를 구할 수 있다.

$$S=tQ+(1-t)P=\frac{(n\bullet p_0)-(n\bullet P)}{n\bullet (Q-P)}Q+\left\{1-\frac{(n\bullet p_0)-(n\bullet P)}{n\bullet (Q-P)}\right\}P$$

$$S=\frac{1}{n\bullet (Q-P)}(((n\bullet p_0)-(n\bullet P))Q+(n\bullet (Q-P)-(n\bullet p_0)+(n\bullet P))P)$$

$$S=\frac{1}{n\bullet (Q-P)}((n\bullet (p_0-P))Q+(n\bullet Q-n\bullet P-n\bullet p_0+n\bullet P)P)$$

$$S=\frac{1}{n\bullet (Q-P)}((n\bullet (p_0-P))Q+(n\bullet Q-n\bullet p_0)P)$$

$$S=\frac{1}{n\bullet (Q-P)}((n\bullet (p_0-P))Q+(n\bullet (Q-p_0))P)$$

교점 3차원 벡터 $S=(S_x,S_y,S_z)$는 다음과 같은 벡터이다.

$$S=\begin{bmatrix}S_x\\S_y\\S_z\end{bmatrix}=\frac{1}{n\cdot(Q-P)}\begin{bmatrix}(n\cdot(p_0-P))Q_x+(n\cdot(Q-p_0))P_x\\(n\cdot(p_0-P))Q_y+(n\cdot(Q-p_0))P_y\\(n\cdot(p_0-P))Q_z+(n\cdot(Q-p_0))P_z\end{bmatrix}$$

$$S=\begin{bmatrix}S_x\\S_y\\S_z\end{bmatrix}=\begin{bmatrix}\frac{1}{n\cdot(Q-P)}((n\cdot(p_0-P))Q_x+(n\cdot(Q-p_0))P_x)\\\frac{1}{n\cdot(Q-P)}((n\cdot(p_0-P))Q_y+(n\cdot(Q-p_0))P_y)\\\frac{1}{n\cdot(Q-P)}((n\cdot(p_0-P))Q_z+(n\cdot(Q-p_0))P_z)\end{bmatrix}$$

3차원 벡터 $S=(S_x,S_y,S_z)$는 다음과 같은 4차원 벡터 $H=(H_x,H_y,H_z,n\cdot(Q-P))$와 동차이다.

$$H=\begin{bmatrix}H_x\\H_y\\H_z\\n\cdot(Q-P)\end{bmatrix}=\begin{bmatrix}(n\cdot(p_0-P))Q_x+(n\cdot(Q-p_0))P_x\\(n\cdot(p_0-P))Q_y+(n\cdot(Q-p_0))P_y\\(n\cdot(p_0-P))Q_z+(n\cdot(Q-p_0))P_z\\n\cdot(Q-P)\end{bmatrix}$$

$$H=\begin{bmatrix}H_x\\H_y\\H_z\\n\cdot(Q-P)\end{bmatrix}=\begin{bmatrix}(-d-(n\cdot P))Q_x+(n\cdot Q+d))P_x\\(-d-(n\cdot P))Q_y+(n\cdot Q+d))P_y\\(-d-(n\cdot P))Q_z+(n\cdot Q+d))P_z\\n\cdot(Q-P)\end{bmatrix}$$

$$H=(Q_x,Q_y,Q_z,1)\begin{bmatrix}-(d+n\cdot P)+aP_x & aP_y & aP_z & a\\ bP_x & -(d+n\cdot P)+bP_y & bP_z & b\\ cP_x & cP_y & -(d+n\cdot P)+cP_z & c\\ dP_x & dP_y & dP_z & -(n\cdot P)\end{bmatrix}$$

점 P와 Q를 지나는 광선과 평면의 교점 S는 점 Q를 다음 행렬 S_P로 변환하고 3차원 동차좌표계로 변환하여 구할 수 있다. 행렬 S_P를 점 P에 대한 평면 투영 행렬이라고 한다.

$$S_P=\begin{bmatrix}-(d+n\cdot P)+aP_x & aP_y & aP_z & a\\ bP_x & -(d+n\cdot P)+bP_y & bP_z & b\\ cP_x & cP_y & -(d+n\cdot P)+cP_z & c\\ dP_x & dP_y & dP_z & -(n\cdot P)\end{bmatrix}$$

행렬 S_L과 행렬 S_P를 하나의 행렬 S_{LP}로 표현하자. 먼저 방향 벡터 $L=(L_x,L_y,L_z)$과 점 $P=(P_x,P_y,P_z)$를 하나의 4차원 벡터 v로 다음과 같이 표현한다. v_w가 1이면 벡터 v는 점을 표현하는 것이다.

$$v=(P_x,P_y,P_z,1)$$

v_w가 0이면 벡터 v는 광선을 표현하는 것이며, 벡터 v는 방향 벡터 $L=(L_x,L_y,L_z)$의 반대 방향($v=-L$)이 되도록 설정한다.

$$v=-(L_x,L_y,L_z,0)$$

k를 다음과 같이 정의한다.

$$k = -(\boldsymbol{n} \cdot \boldsymbol{v}) + dv_w$$

k는 방향 벡터의 경우 $(\boldsymbol{n} \cdot \boldsymbol{L})$이 되고, 점 벡터의 경우 $(-(\boldsymbol{n} \cdot \boldsymbol{P}) + d)$가 된다.

행렬 $\boldsymbol{S}_L$과 행렬 $\boldsymbol{S}_P$를 하나의 행렬 $\boldsymbol{S}_{LP}$로 다음과 같이 표현할 수 있다.

$$\boldsymbol{S}_{LP} = \begin{bmatrix} av_x + k & av_y & av_z & av_w \\ bv_x & bv_y + k & bv_z & bv_w \\ cv_x & cv_y & cv_z + k & cv_w \\ dv_x & dv_y & vv_z & dv_w + k \end{bmatrix}$$

⑯ 평면(Plane) 함수

평면은 법선 벡터 $\boldsymbol{n} = (a, b, c)$과 원점에서 평면까지의 거리 d로 표현할 수 있다. 수학적으로 평면의 방정식은 다음과 같다. 이것은 평면 위의 임의의 점 (x, y, z)는 $ax + by + cz + d = 0$을 만족한다는 것을 의미한다. a, b, c는 평면의 법선 벡터 (a, b, c)이고 d는 원점에서 평면까지의 거리이다. 평면 $\boldsymbol{p}$는 4차원 벡터 $\boldsymbol{p} = (a, b, c, d)$로 표현할 수 있다.

다음 함수 XMPlaneFromPoints()는 세 점(p0, p1, p2)으로부터 평면을 생성하여 반환한다.

```
XMVECTOR XMPlaneFromPoints(XMVECTOR p0, XMVECTOR p1, XMVECTOR p2);
```

벡터 $(\boldsymbol{P}_1 - \boldsymbol{P}_0) \times (\boldsymbol{P}_2 - \boldsymbol{P}_0)$를 정규화하면 평면의 법선 벡터 $\boldsymbol{n} = (a, b, c)$를 구할 수 있다. 원점에서 평면까지의 거리 d는 법선 벡터와 평면 위의 점 $\boldsymbol{P}_1$의 내적 $-(\boldsymbol{n} \cdot \boldsymbol{P}_1)$으로 구할 수 있다.

다음 함수 XMPlaneFromPointNormal()은 평면 위의 한 점(p)과 평면의 법선 벡터(normal)로부터 평면을 생성하여 반환한다.

```
XMVECTOR XMPlaneFromPointNormal(XMVECTOR p, XMVECTOR normal);
```

다음 함수 XMPlaneNormalize()와 XMPlaneNormalizeEst()는 평면(p)을 정규화하여 반환한다.

```
XMVECTOR XMPlaneNormalize(XMVECTOR p);
XMVECTOR XMPlaneNormalizeEst(XMVECTOR p);
```

평면 $\boldsymbol{p} = (a, b, c, d)$가 나타내는 평면의 방정식 $ax + by + cz + d = 0$을 정규화하는 것은 평면의 법선 벡터의 크기가 1이 되도록 정규화하는 것이다.

$$ax + by + cz + d = 0$$

$$\frac{ax+by+cz+d}{\sqrt{a^2+b^2+c^2}}=0$$

$$\frac{ax}{\sqrt{a^2+b^2+c^2}}+\frac{by}{\sqrt{a^2+b^2+c^2}}+\frac{cz}{\sqrt{a^2+b^2+c^2}}+\frac{d}{\sqrt{a^2+b^2+c^2}}=0$$

다음 함수 XMPlaneDot()는 평면(p)과 4차원 점 벡터(v)의 내적 연산의 결과를 반환한다.

```
XMVECTOR XMPlaneDot(XMVECTOR p, XMVECTOR v);
```

동차 좌표계의 한 점 $\boldsymbol{v}=(v_x, v_y, v_z, v_w)$와 평면 $\boldsymbol{p}=(a,b,c,d)$의 내적 연산에서 v_w가 1이면, 점과 평면 사이의 위치 관계를 결정할 수 있다. 점 벡터 $\boldsymbol{v}=(v_x, v_y, v_z, 1)$와 평면 $\boldsymbol{p}=(a,b,c,d)$의 위치 관계는 내적 연산의 결과 $(av_x+bv_y+cv_z+d)$의 부호로 판단할 수 있다.

점 벡터 $\boldsymbol{v}=(v_x, v_y, v_z, 1)$와 평면 $\boldsymbol{p}=(a,b,c,d)$의 위치 관계는 내적 연산의 결과 $(av_x+bv_y+cv_z+d)$의 부호로 판단할 수 있다.

- □ $(av_x+bv_y+cv_z+d>0)$이면 점 $\boldsymbol{v}=(v_x, v_y, v_z, 1)$은 평면 $\boldsymbol{p}=(a,b,c,d)$ 앞에 있다.
- □ $(av_x+bv_y+cv_z+d<0)$이면 점 $\boldsymbol{v}=(v_x, v_y, v_z, 1)$은 평면 $\boldsymbol{p}=(a,b,c,d)$ 뒤에 있다.
- □ $(av_x+bv_y+cv_z+d=0)$이면 점 $\boldsymbol{v}=(v_x, v_y, v_z, 1)$은 평면 $\boldsymbol{p}=(a,b,c,d)$ 위에 있다.

다음 함수 XMPlaneDotCoord()는 평면(p)과 3차원 점 벡터(v)의 내적 연산의 결과를 반환한다. 한 점 $\boldsymbol{v}=(v_x, v_y, v_z, 1)$와 평면 $\boldsymbol{p}=(a,b,c,d)$ 사이의 위치 관계를 결정할 수 있다.

```
XMVECTOR XMPlaneDotCoord(XMVECTOR p, XMVECTOR v);
```

평면 $\boldsymbol{p}=(a,b,c,d)$이 정규화되었다면 $(av_x+bv_y+cv_z+d)$는 점 $\boldsymbol{v}=(v_x, v_y, v_z, 1)$에서 평면 $\boldsymbol{p}=(a,b,c,d)$까지의 수직 거리가 된다.

다음 함수 XMPlaneDotNormal()는 평면(p)과 3차원 방향 벡터(v, 단위벡터)의 내적 연산의 결과를 반환한다. 방향 벡터 $\boldsymbol{v}=(v_x, v_y, v_z, 0)$와 평면 $\boldsymbol{p}=(a,b,c,d)$의 내적 연산 $av_x+bv_y+cv_z$는 방향 벡터 $\boldsymbol{v}$와 평면의 법선 벡터 $\boldsymbol{n}$이 이루는 각도의 코사인 값이다.

```
XMVECTOR XMPlaneDotNormal(XMVECTOR p, XMVECTOR v);
```

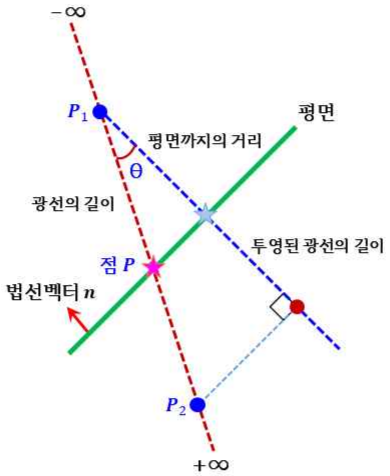
다음 함수 XMPlaneTransform()는 평면(p)을 (4×4) 행렬을 사용하여 변환하여 반환한다. 평면을 행렬을 사용하여 변환하는 것은 평면(벡터) $\boldsymbol{p}=(a,b,c,d)$를 (4×4) 행렬 $\boldsymbol{m}$으

로 변환하는 것이다.

```
XMVECTOR XMPlaneTransform(XMVECTOR p, XMMATRIX m);
```

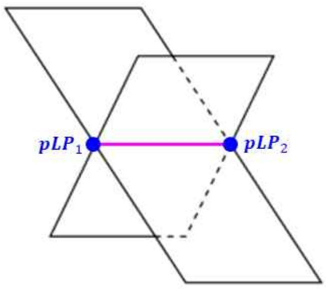
다음 함수 XMPlaneIntersectLine()는 평면(p)와 두 점(p1, p2)을 지나는 직선과의 교점을 반환한다. 두 점(p1, p2)을 지나는 직선이 평면(p)과 평행하면(평면의 법선 벡터와 수직이면) 반환되는 벡터의 모든 요소는 QNaN 값을 갖는다.

```
XMVECTOR XMPlaneIntersectLine(XMVECTOR p, XMVECTOR p1, XMVECTOR p2);
```



다음 함수 XMPlaneIntersectPlane()는 두 평면(p1, p2)이 만날 때의 직선(pLP1, pLP2)을 반환한다. 서로 평행하지 않은 두 평면은 만나게 되며 교점들은 직선이 된다. 두 평면(p1, p2)이 평행하면(두 평면의 법선 벡터가 같으면) 반환되는 벡터의 모든 요소는 QNaN 값을 갖는다.

```
void XMPlaneIntersectPlane(XMVECTOR *pLP1, XMVECTOR *pLP2, XMVECTOR
p1, XMVECTOR p2);
```



다음은 평면을 비교하는 함수들이다. 함수 XMPlaneEqual()은 두 평면(p1, p2)이 같은 가를 비교하여 같으면 참을 반환한다. 두 평면의 모든 요소가 서로 같으면 두 평면은 서로 같다. 함수 XMPlaneNearEqual()은 두 평면(p1, p2)이 정밀도(epsilon)의 범위 내에서 같은 가를 비교하여 같으면 참을 반환한다. 함수 XMPlaneNotEqual()은 두 평면(p1, p2)을

비교하여 같지 않으면 참을 반환한다. 함수 XMPlaneIsInfinite()는 평면(p)의 어떤 요소라도 ±∞이면 참을 반환한다. 함수 XMPlaneIsNaN()은 평면(p)의 어떤 요소라도 NaN이면 참을 반환한다.

```
bool XMPlaneEqual(XMVECTOR p1, XMVECTOR p2);
bool XMPlaneNearEqual(XMVECTOR p1, XMVECTOR p2, XMVECTOR epsilon);
bool XMPlaneNotEqual(XMVECTOR p1, XMVECTOR p2);
bool XMPlaneIsInfinite(XMVECTOR p);
bool XMPlaneIsNaN(XMVECTOR p);
```