단국대학교 멀티미디어공학 게임 프로그래밍 중간고사 (2009년 봄학기) 2009년 4월 21일

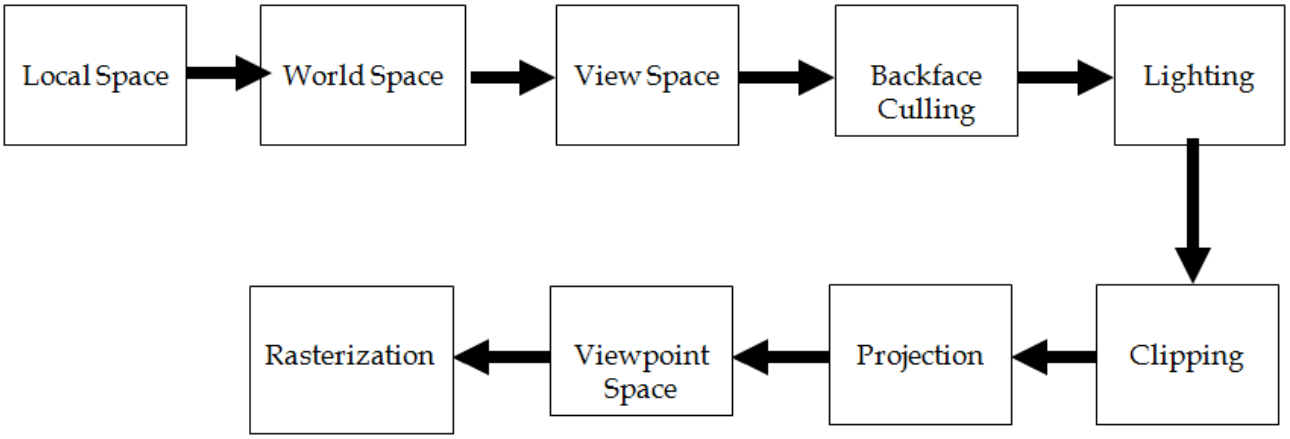
학과 _____ 학번 _____ 이름 _____

중간고사

담당교수: 단국대학교 멀티미디어공학전공 박경신

- 답은 반드시 답안지에 기술할 것. 공간이 부족할 경우 반드시 답안지 몇 쪽의 뒤에 있다고 명기한 후 기술할 것. 그 외의 경우의 답안지 뒤쪽이나 연습지에 기술한 내용은 답안으로 인정 안 함. 답에는 반드시 네모를 쳐서 확실히 표시할 것.
- 답안지에 학과, 학번, 이름 외에 본인의 암호를 기입하면 성적공고시 학번 대신 암호를 사용할 것임.

1. 다음은 기하학적 객체(모델)들로 3차원 장면을 화면에 출력하기까지의 렌더링 파이프라인 (Rendering Pipeline)을 보여주고 있다. 각각의 단계를 간단히 설명하라. (20점)



Local Space: 모델링 좌표계는 사용자가 물체를 (삼각형의 리스트로) 정의하는데 이용하는 물체 중심적 좌표계임.

World Space: 모델링 좌표계에서 다수의 모델을 구성한 후 이를 세계 좌표계를 옮겨 하나의 장면을 구성하는데 사용하는 좌표계로, 여러 모델들을 포함하고 각 물체의 관계를 정의함.

View Space: 카메라를 세계 좌표계(World Space)의 원점으로 이동하고 +Z 축으로 보도록 회전시켜 놓는 카메라 시점의 좌표계로, 세계 좌표계에 있는 모든 기하물체를 View Space로 변환되어야 함.

Backface Culling: 카메라의 후면을 향하고 있는 폴리곤은 화면에서 그리지 않도록 미리 추려내는 과정. Backface Culling을 하면 이후의 계산에 상당한 이득이 있음.

Lighting: 물체에 명암을 추가하여 장면에 사실감을 더해줌. 입체의 형태와 물체의 부피를 표현하는데 도움을 줌.

Clipping: 시야 볼륨 (View Frustrum) 외부에 있는 기하 물체를 추려내는 과정을 클리핑 (Clipping)이라 함. 기하 물체가 시야 볼륨의 내부에 완전히 속해 있으면 보존하고, 완전히 외부에 있으면 추려내고, 부분적으로 내부에 걸쳐있으면 폴리곤을 두 부분으로 분리하여 내부의 부분만 보존함.

Projection: View Space에서의 3차원 장면을 2차원으로 투영 (Projection)하는 과정. 원근 투영 (Perspective projection)은 원근법을 이용하여 기하물체를 투사함.

Viewpoint Space: 투영된 윈도우의 좌표를 화면의 뷰포트(Viewport)라는 직사각형으로 변환하는 과정.

Rasterization: 정점 (Vertex)들을 화면 좌표계로 변환하여 2차원 삼각형 (Triangle)들을 그리기 위해 화소값 (Pixel Color)들을 계산하는 과정. 전용 그래픽스 하드웨어에서 처리되며 결과물은 바로 화면에 출력될 수 있는 이미지 형태로 나타남.

2. 다음은 D3DMATERIAL9 구조체를 보여주고 있다. Diffuse, Ambient, Specular, Emissive, Power가 무엇인지 구체적으로 설명하라 (10점).

```
typedef struct _D3DMATERIAL9 {
    D3DCOLORVALUE    Diffuse;
    D3DCOLORVALUE    Ambient;
    D3DCOLORVALUE    Specular;
    D3DCOLORVALUE    Emissive;
    float            Power;
} D3DMATERIAL9;
```

Diffuse: 물체의 표면이 반사하는 난반사광(즉, 특정한 방향으로 진행하다가 표면에 닿으면 모든 방향으로 동일하게 반사되는 빛)의 양을 지정함. 일반적으로 물체 재질의 고유한 색을 표현함.

Ambient: 물체의 표면이 반사하는 환경광(즉, 전반적인 장면을 밝게 하는 빛)의 양을 지정함. 물체 표면에 광원이 없더라도 약간의 빛을 받는 것처럼 느낄 수 있도록 하는 색을 표현함.

Specular: 물체의 표면이 반사하는 정반사광(즉, 특정한 방향으로 진행하다가 표면에 닿으면 한 방향으로 강하게 반사하여 특정한 각도에서만 관찰할 수 있는 빛)의 양을 지정함. 물체의 반짝이는 표면에 빛이 반사하는 효과를 주기 위해 지정함.

Emissive: 물체의 표면 자체가 빛을 발하는 것처럼 좀 더 밝은 물체효과를 만들어냄.

Power: 정반사광의 날카로운 정도 (Sharpness)를 지정하며, 값이 높아질 수록 하이라이트가 강조됨.

3. 다음은 D3DLIGHT9 구조체를 보여주고 있다. D3DLIGHT9이 지원하는 세 가지 광원의 타입과 각 광원을 지정하는데 필요한 변수를 설명하라 (5점). 그리고 Attenuation0, Attenuation1, Attenuation2가 무엇인지 구체적으로 설명하라 (5점).

```
typedef struct _D3DLIGHT9 {
    D3DLIGHTTYPE        Type;
    D3DCOLORVALUE       Diffuse;
    D3DCOLORVALUE       Specular;
    D3DCOLORVALUE       Ambient;
    D3DVECTOR           Position;
    D3DVECTOR           Direction;
    float               Range;
    float               Falloff;
    float               Attenuation0;
    float               Attenuation1;
    float               Attenuation2;
    float               Theta;
    float               Phi;
} D3DLIGHT9;
```

D3DLIGHT9이 지원하는 3가지 광원의 타입과 각 광원을 지정하는데 필요한 변수:

점 광원 (point light) – 광원의 위치로부터 모든 방향으로 빛을 발산한다. 광원의 위치(position)와 광원의 환경광(ambient), 난반사광(diffuse), 정반사광(specular) 색을 변수로 지정해야 한다. 또한 Attenuation0,1,2를 사용하여 거리에 따라 빛의 세기가 약해지는 정도를 정의할 수 있다.

방향성광원 (directional light) – 지정된 방향으로 평행하게 빛을 발산한다. 광원의 방향(direction)과 광원의 환경광(ambient), 난반사광(diffuse), 정반사광(specular) 등의 빛의 색을 변수로 지정해야 한다.

스포트 광원 (spot light) – 손전등 빛과 같이, 광원의 위치로부터 특정한 방향으로 원뿔형태의 빛을 발산한다. 광원의 위치(position)와 방향(direction)과 광원의 환경광(ambient), 난반사광(diffuse), 정반사광(specular) 등의 빛의 색을 변수로 지정해야 한다. 또한 Attenuation0,1,2를 사용하여 거리에 따라 빛의 세기가 약해지는 정도와 Theta와 Phi를 사용하여 원뿔의 안쪽과 바깥쪽 각도(in radian)를 정의할 수 있다.

Attenuation0/1/2:

Attenuation0,1,2를 사용하여 거리에 따라 빛의 세기가 약해지는 정도를 정의할 수 있다.
Directional light에서는 무시되며 0에서 ∞까지의 값을 가진다.
Directional light이 아닌 경우 세 값을 0으로 하면 안된다.
Attenuation = $1/(a_0+a_1*D+a_2*D^2)$
Attenuation0은 상수감소를, Attenuation1은 선형감소를 Attentiation2는 이차감소를 정의한다.

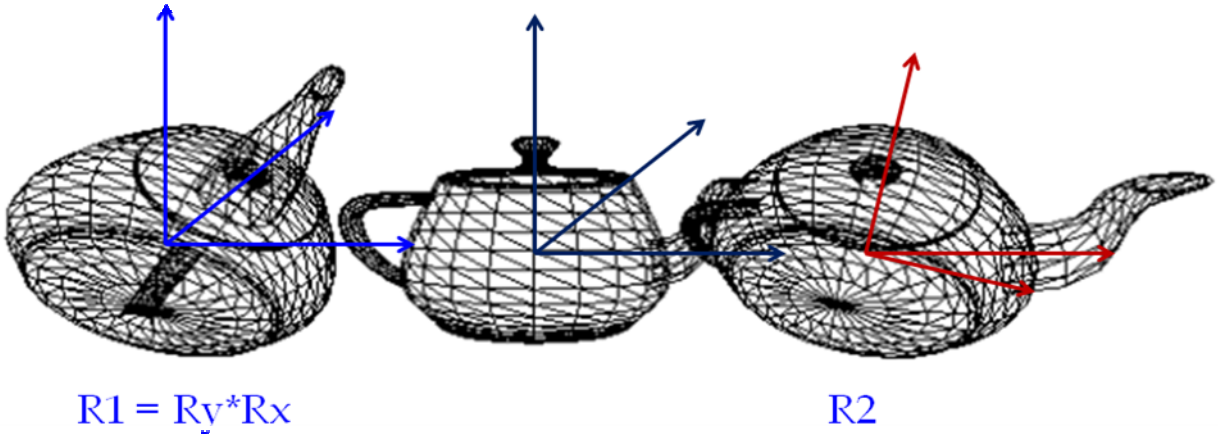
학과 _____ 학번 _____ 이름 _____

4. D3DXMatrixRotationX/Y/Z와 D3DXMatrixRotationYawPitchRoll 함수를 각각 설명하고, 아래의 간단한 코드를 참조하여 차이점을 설명하라 (10점).

```
D3DXMATRIX R1, Rx, Ry;
D3DXMatrixRotationX(&Rx, D3DXToRadian(45.0));
D3DXMatrixRotationY(&Ry, D3DXToRadian(45.0));
R1 = Ry * Rx ;

D3DXMATRIX R2;
D3DXMatrixRotationYawPitchRoll(&R2, D3DXToRadian(45.0), D3DXToRadian(45.0), 0.0);
```

D3DXMatrixRotationYawPitchRoll과 D3DXMatrixRotationX/Y/Z의 차이점은, YawPitchRoll이 단순한 x,y,z 회전의 곱이 아니라, Local Coordinate System에서 회전이 누적된다는 점이다. 따라서 간단한 코드의 결과는 아래와 같이 나타난다.



5. 다음은 간단한 Geometry 프로그램을 보여주고 있다. 출력화면의 그림을 그리시오. 그림에 Geometry의 Vertex를 표시할 것 (10점).

```
IDirect3DVertexBuffer9* g_pGeometry_VB = 0;
struct Vertex
{
        float x, y, z;
        float nx, ny, nz;
        static const DWORD FVF;
};
const DWORD Vertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL;
```

```
Vertex g_pGeometryVertices[] =
{
    {-1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f},
    { 0.0f, 1.0f,   0.0f, 0.0f, 0.707f, -0.707f},
    { 1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f},
    {-1.0f, 0.0f,   1.0f, -0.707f, 0.707f, 0.0f},
    { 0.0f, 1.0f,   0.0f, -0.707f, 0.707f, 0.0f},
    {-1.0f, 0.0f, -1.0f, -0.707f, 0.707f, 0.0f},
    { 1.0f, 0.0f, -1.0f, 0.707f, 0.707f, 0.0f},
    { 0.0f, 1.0f,   0.0f, 0.707f, 0.707f, 0.0f},
    { 1.0f, 0.0f,   1.0f, 0.707f, 0.707f, 0.0f},
    { 1.0f, 0.0f,   1.0f, 0.0f, 0.707f, 0.707f},
    { 0.0f, 1.0f,   0.0f, 0.0f, 0.707f, 0.707f},
    {-1.0f, 0.0f,   1.0f, 0.0f, 0.707f, 0.707f},
    {-1.0f, 0.0f, -1.0f, 0.0f, -1.0f, 0.0f},
    { 1.0f, 0.0f, -1.0f, 0.0f, -1.0f, 0.0f},
    {-1.0f, 0.0f,   1.0f, 0.0f, -1.0f, 0.0f},
    { 1.0f, 0.0f, -1.0f, 0.0f, -1.0f, 0.0f},
    { 1.0f, 0.0f,   1.0f, 0.0f, -1.0f, 0.0f},
    {-1.0f, 0.0f,   1.0f, 0.0f, -1.0f, 0.0f}
};

bool Setup()
{
        // Create vertex and index buffers for GEOMETRY
        Device->CreateVertexBuffer(
                18 * sizeof(Vertex),
                D3DUSAGE_WRITEONLY,
                Vertex::FVF,
                D3DPOOL_MANAGED,
                &g_pGeometry_VB,
                0);

        void * pVertices = NULL;
        g_pGeometry_VB->Lock( 0, sizeof(g_pGeometryVertices), (void**)&pVertices, 0 );
        memcpy( pVertices, g_pGeometryVertices, sizeof(g_pGeometryVertices) );
        g_pGeometry_VB->Unlock();
// 중간생략...
```

```
	return true;
}

void Cleanup()
{
	d3d::Release<IDirect3DVertexBuffer9*>(g_pGeometry_VB);
}

bool Display(float timeDelta)
{
	if( Device )
	{
		Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
		Device->BeginScene();
// 중간생략...
		// Draw GEOMETRY.
		Device->SetStreamSource(0, g_pGeometry_VB, 0, sizeof(Vertex));
		Device->SetFVF(Vertex::FVF);
		Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 6);

		Device->EndScene();
		Device->Present(0, 0, 0, 0);
	}
	return true;
}
```

6. Single-Pass Multi-texturing과 Multi-Pass Multi-texturing의 차이점을 간단히 설명하라 (10점). 5번 문제의 Geometry 프로그램에서 두 개의 텍스쳐 이미지를 읽어 들여와서 Single-Pass Multi-texturing을 하려면 추가해야 할 부분을 간단히 적어라 (10점).

멀티 텍스처(multi texture)란 말 그대로 다중 텍스처 (여러 장의 텍스처) 즉, 하나의 폴리곤이 그려질 때 여러 장의 텍스처가 중첩되어 출력되는 것을 말한다. Single-Pass Multitexturing은 하나의 폴리곤 렌더링에 여러 개의 텍스처를 중첩하여 그리는 것을 말하고, Multi-Pass Multitexturing은 다중 텍스처링을 지원하지 않는 하드웨어에서 다중 텍스처링을 흉내내는 기법으로 장면(또는 폴리곤)을 여러 번 렌더링하는 것이다.

```
LPDIRECT3DTEXTURE9      g_pTex0      = NULL; // Texture 0
LPDIRECT3DTEXTURE9      g_pTex1      = NULL; // Texture 1

// vertex 구조체에 2개의 texture coordinate 추가
struct Vertex
{
        float x, y, z;
        float nx, ny, nz;
        float u1, v1;
        float u2, v2;

        static const DWORD FVF;
};
const DWORD Vertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX2;

// 실제 Vertex에 u1,v1과 u2, v2 texture coordinate 추가
Vertex g_GeometryVertices[] =
{
    {-1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f, 0.0f, 1.0f, 0.0f, 1.0f},
   { 0.0f, 1.0f,   0.0f, 0.0f, 0.707f, -0.707f, 0.5f, 0.0f, 0.5f, 0.0f},
    { 1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f, 1.0f, 1.0f, 1.0f, 1.0f},
    {-1.0f, 0.0f,   1.0f, -0.707f, 0.707f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f},
    { 0.0f, 1.0f,   0.0f, -0.707f, 0.707f, 0.0f, 0.5f, 0.0f, 0.5f, 0.0f},
    {-1.0f, 0.0f, -1.0f, -0.707f, 0.707f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f},
    { 1.0f, 0.0f, -1.0f, 0.707f, 0.707f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f},
    { 0.0f, 1.0f,   0.0f, 0.707f, 0.707f, 0.0f, 0.5f, 0.0f, 0.5f, 0.0f},
    { 1.0f, 0.0f,   1.0f, 0.707f, 0.707f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f},
    { 1.0f, 0.0f,   1.0f, 0.0f, 0.707f, 0.707f, 0.0f, 1.0f, 0.0f, 1.0f},
```

```
    { 0.0f, 1.0f,   0.0f, 0.0f, 0.707f, 0.707f, 0.5f, 0.0f, 0.5f, 0.0f},
    {-1.0f, 0.0f,   1.0f, 0.0f, 0.707f, 0.707f, 1.0f, 1.0f, 1.0f, 1.0f},
    {-1.0f, 0.0f, -1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f},
    { 1.0f, 0.0f, -1.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f},
    {-1.0f, 0.0f,   1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f},
    { 1.0f, 0.0f, -1.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f},
    { 1.0f, 0.0f,   1.0f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f},
    {-1.0f, 0.0f,   1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f}
};

bool Setup()
{
// 중간 생략..
    // create texture and set filters
    D3DXCreateTextureFromFile(Device, "crate.jpg", &g_pTex0);
    D3DXCreateTextureFromFile(Device, "darkmap.png", &g_pTex1);
    Device->SetTexture(0, g_pTex0);
    Device->SetTexture(1, g_pTex1);
    Device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_POINT);

    // set multi-texturing
    // 스테이지0 설정 – 색상연산과 알파연산은 값을 변경없이 그대로 사용
    Device->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_SELECTARG1);
    Device->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
    Device->SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_SELECTARG1);
    Device->SetTextureStageState(0, D3DTSS_ALPHAARG1, D3DTA_TEXTURE);
    Device->SetTexture(0, g_pTex0);

    // 스테이지1 설정 – 색상연산은 현재 텍스쳐색상과 이전 스테이지 결과(즉, 벽)를
    // MODULATE으로 곱하고, 알파연산은 사용안함
    Device->SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_MODULATE);
    Device->SetTextureStageState(1, D3DTSS_COLORARG1, D3DTA_TEXTURE);
    Device->SetTextureStageState(1, D3DTSS_COLORARG2, D3DTA_CURRENT);
    Device->SetTextureStageState(1, D3DTSS_ALPHAOP, D3DTOP_DISABLE);
    Device->SetTexture(1, g_pTex1);
}
```

학과 _____ 학번 _____ 이름 _____

7. 다음은 사각형을 블렌딩(Blending)하여 화면에 출력하는 프로그램 코드의 일부를 보여주고 있다. 빈 칸에 블렌딩 방식을 간단히 설명하고, 블렌딩 최종 값을 식으로 나타내라. 이 때 srcPixel은 C_s로, destPixel은 C_d로 표현한다 (10점).

블렌딩 공식: outputPixel = srcPixel*srcBlendFactor + destPixel*dstBlendFactor
알파 블렌딩: outputPixel = C_s* A_s + C_d*(1 – A_s)

```
bool Display(float timeDelta)
{
        if( Device ) {
                Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
                Device->BeginScene();

                // _알파 블렌딩을 설정한 채로 원래 그대로 불투명한 것을 그리고 싶을 때 지정
                // _ outputPixel = Cs ____
                Device->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_ONE);
                Device->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ZERO);
                Device->SetTexture(0, Tex1);

                Device->SetStreamSource(0, Quad, 0, sizeof(Vertex));
                Device->SetFVF(Vertex::FVF);
                Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);

                // _ 후면버퍼에 텍스쳐Tex1과 현재 그리고자 하는 텍스쳐Tex2를 곱셈 블렌딩함
                // _ outputPixel = Cd *Cs_____
                Device->SetRenderState(D3DRS_SRCBLEND,D3DBLEND_ZERO);
                Device->SetRenderState(D3DRS_DESTBLEND,D3DBLEND_SRCCOLOR);
                Device->SetTexture(0, Tex2);

                Device->SetStreamSource(0, Quad, 0, sizeof(Vertex));
                Device->SetFVF(Vertex::FVF);
                Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);

                Device->EndScene();
                Device->Present(0, 0, 0, 0);
        }
        return true;
}
```

8. 스텐실 버퍼 (Stencil buffer)에 대해 설명하고, 스텐실 버퍼를 활용하는 대표적인 사례를 명시하고 구현 방법을 간단히 적어라 (10점).

스텐실 버퍼란 특수한 효과를 위한 off-screen buffer임.
Backbuffer, depth buffer와 동일한 해상도를 가지며, 깊이 버퍼의 포멧을 지정할 때 스텐실 버퍼의 포멧도 함께 지정함.
Stencil buffer는 back buffer의 일정 부분이 rendering되지 않도록 스텐실 연산을 수행함.

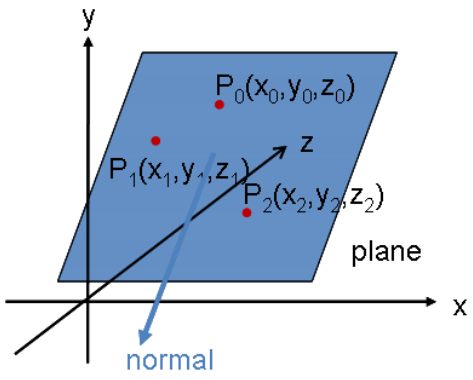
대표적인 활용으로는 벽면에 거울이 있는 경우, 벽면을 제외하고 거울이 있는 영역에 대해서만 반사되는 물체의 drawing을 수행하도록 함.

- Stencil buffer를 0값으로 clear함.
- Stencil buffer에 거울을 구성하는 기본 도형을 렌더링함 (스텐실 테스트를 항상 성공하도록 하고 테스트가 성공하면 스텐실 버퍼는 1로 지정함).
- 반사된 주전자를 후면 버퍼와 스텐실 버퍼로 렌더링함 (스텐실 테스트가 통과한 부분만 후면 버퍼에 렌더링. 즉, Stencil buffer항목이 1인 경우에만 stencil test를 pass하도록 지정함).

또 다른 활용예로는 물체의 그림자를 구현하는 경우, 물체의 기하 정보에 Shadow Matrix를 정용하여 flatten하게 만들면 여러 개의 삼각형들이 겹치게 되는데 이들을 반투명 알파 블렌딩을 사용하여 그림자를 렌더링하면 겹쳐진 영역이 더욱 어둡게 나타나는 '더블 블렌딩'이라는 부작용이 나타나게 되므로, 이를 방지하기 위하여 스텐실 버퍼를 사용함.

- Stencil buffer를 사용하여 처음으로 렌더링되는 픽셀들만 받아들이도록 스텐실 테스트를 구성함.
- 그림자 pixel이 후면버퍼에 렌더링되는 동안 스텐실 버퍼에 표시를 남겨 놓음.
- 이미 렌더링된 (스텐실 버퍼에 표시된) 그림자 영역에 다시 픽셀을 쓰려고 하면, 스텐실 테스트가 실패하므로 blending되지 않음.

9. 다음은 평면을 만들어내는 Direct3D 코드의 일부이다. 빈 칸을 완성하시오 (extra 10점).
주의: 평면의 법선 벡터를 중심으로 winding order를 잡아야 함.



```
void GetPlaneFromPoints(D3DXVECTOR3 p0, D3DXVECTOR3 p1, D3DXVECTOR3 p2, D3DXPLANE& out)
{
    D3DXVECTOR3 u, v, n;
    u = _____p2 - p0;_____
    v = _____p1 - p0;_____
    D3DXVec3Cross(_____&n, &u, &v_____);
    D3DXVec3Normalize(_____&n, &n_____);
    FLOAT d = -D3DXVec3Dot(_____&n, &p0_____);
    out.a = n.x;
    out.b = n.y;
    out.c = n.z;
    out.d = d;
}
```