void useVertexTexture(LPDIRECT3DDEVICE9 device, GameObjectManager& gameObjectManager)

{

//

// 텍스터링 과정

//

LPDIRECT3DVERTEXBUFFER9 vb = nullptr;

LPDIRECT3DINDEXBUFFER9 ib = nullptr;

LPDIRECT3DTEXTURE9 texture = nullptr;

D3DXCreateTextureFromFile(device, L"media/crate.jpg", &texture);

// save a ptr to the device

device->CreateVertexBuffer(

24 \* sizeof(TVertex),

D3DUSAGE\_WRITEONLY,

TVertex::FVF,

D3DPOOL\_MANAGED,

&vb,

0);

TVertex\* v;

vb->Lock(0, 0, (void\*\*)&v, 0);

// build box

// fill in the front face vertex data

v[0] = TVertex(-1.0f, -1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f);

v[1] = TVertex(-1.0f, 1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f);

v[2] = TVertex(1.0f, 1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f);

v[3] = TVertex(1.0f, -1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f);

// fill in the back face vertex data

v[4] = TVertex(-1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f);

v[5] = TVertex(1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f);

v[6] = TVertex(1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f);

v[7] = TVertex(-1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f);

// fill in the top face vertex data

v[8] = TVertex(-1.0f, 1.0f, -1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f);

v[9] = TVertex(-1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f);

v[10] = TVertex(1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f);

v[11] = TVertex(1.0f, 1.0f, -1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f);

// fill in the bottom face vertex data

v[12] = TVertex(-1.0f, -1.0f, -1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f);

v[13] = TVertex(1.0f, -1.0f, -1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f);

v[14] = TVertex(1.0f, -1.0f, 1.0f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f);

v[15] = TVertex(-1.0f, -1.0f, 1.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f);

// fill in the left face vertex data

v[16] = TVertex(-1.0f, -1.0f, 1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f);

v[17] = TVertex(-1.0f, 1.0f, 1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f);

v[18] = TVertex(-1.0f, 1.0f, -1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f);

v[19] = TVertex(-1.0f, -1.0f, -1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f);

// fill in the right face vertex data

v[20] = TVertex(1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f);

v[21] = TVertex(1.0f, 1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f);

v[22] = TVertex(1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f);

v[23] = TVertex(1.0f, -1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f);

vb->Unlock();

device->CreateIndexBuffer(

36 \* sizeof(WORD),

D3DUSAGE\_WRITEONLY,

D3DFMT\_INDEX16,

D3DPOOL\_MANAGED,

&ib,

0);

WORD\* i = 0;

ib->Lock(0, 0, (void\*\*)&i, 0);

// fill in the front face index data

i[0] = 0; i[1] = 1; i[2] = 2;

i[3] = 0; i[4] = 2; i[5] = 3;

// fill in the back face index data

i[6] = 4; i[7] = 5; i[8] = 6;

i[9] = 4; i[10] = 6; i[11] = 7;

// fill in the top face index data

i[12] = 8; i[13] = 9; i[14] = 10;

i[15] = 8; i[16] = 10; i[17] = 11;

// fill in the bottom face index data

i[18] = 12; i[19] = 13; i[20] = 14;

i[21] = 12; i[22] = 14; i[23] = 15;

// fill in the left face index data

i[24] = 16; i[25] = 17; i[26] = 18;

i[27] = 16; i[28] = 18; i[29] = 19;

// fill in the right face index data

i[30] = 20; i[31] = 21; i[32] = 22;

i[33] = 20; i[34] = 22; i[35] = 23;

ib->Unlock();

D3DMATERIAL9 \* material = new D3DMATERIAL9();

material->Ambient = WHITE;

material->Diffuse = WHITE;

material->Specular = WHITE;

material->Emissive = BLACK;

material->Power = 5.0f;

GameObjectBuffer \* obj = new GameObjectBuffer(vb, ib, TVertex::FVF, sizeof(TVertex), material, texture);

obj->Move(3, 0, 0);

//obj->SetScale(D3DXVECTOR3(1, 1, 5));

gameObjectManager.AddObjectBuffer(obj);

//

// 필터링 과정

//

// D3DSAMP\_MAGFILTER - 확대에서 사용할 필터링

// D3DSAMP\_MINFILTER - 축소에서 사용할 필터링

// 1. Nearest point sampling - 근접점 샘플링

// D3DTEXF\_POINT

// 디폴트 필터링 방식. 가장 떨어지는 품질의 결과를 만들어냄. 속도는 가장 빠름

device->SetSamplerState(0, D3DSAMP\_MAGFILTER, D3DTEXF\_POINT);

device->SetSamplerState(0, D3DSAMP\_MINFILTER, D3DTEXF\_POINT);

// 2. Linear filtering - 선형 필터링

// D3DTEXF\_LINEAR

// 비교적 높은 품질에 속도도 빠름

// 3. Anisotropic filtering - 비등방성 필터링

// D3DTEXF\_ANISOTROPIC

// 가장 높은 품질. 속도는 제일 느림

// 이것을 이용할 때는 비등방성 필터링의 품질을 결정하는 D3DSAMP\_MAXANISOTROPIC 레벨을

// 지정해야 한다. 높을수록 품질이 좋아진다

// 값의 범위는 D3DCAPS9 구조체 참조

device->SetSamplerState(0, D3DSAMP\_MAXANISOTROPY, 4);

//

// 밉맵

//

// 텍스처의 크기를 급격하게 변경하는 것을 막는 테크닉으로 텍스처의 밉맵을 만드는 방법이 있다.

// 밉맵을 이용하는 방법을 결정하기 위해 밉맵 필터를 지정할 수 있다.

device->SetSamplerState(0, D3DSAMP\_MIPFILTER, D3DTEXF\_NONE);

// D3DTEXF\_NONE - 밉맵을 사용하지 않음

// D3DTEXF\_POINT - 스크린 삼각형과 가장 비슷한 크기의 밉맵을 선택

// D3DTEXF\_LINEAR - 가장 비슷한 두 개의 밉맵 레벨에 MIN과 MAG 필터에 따른 필터링을 적용

// 선형적으로 조합된 두 개의 레벨을 이용해 최종 픽셀 컬러를 계산

// 밉맵 - 어드레스 모드

// 텍스처의 좌표는 [0, 1]을 넘을 수 있다.

// 이 범위를 넘는 좌표를 처리하는 방법은 D3D ADDRESS MODE에 의해 정의된다.

// 모드에는 WRAP, BORDER COLOR, CLAMP, MIRROR 가 있다

device->SetSamplerState(0, D3DSAMP\_ADDRESSU, D3DTADDRESS\_WRAP);

device->SetSamplerState(0, D3DSAMP\_ADDRESSV, D3DTADDRESS\_WRAP);

// D3DTADDRESS\_WRAP - WRAP 모드

// D3DTADDRESS\_BORDER - BORDER 모드

// BORDER 색상 처리

device->SetSamplerState(0, D3DSAMP\_BORDERCOLOR, 0x000000ff);

// D3DTADDRESS\_CLAMP - CLAMP 모드

// D3DTADDRESS\_MIRROR - MIRROR 모드

}