

Direct3D Drawing

305890
2007년 봄학기
3/23/2007
박경신

Drawing

- Vertex buffer/Index buffer
 - Vertex buffer와 Index buffer 생성
 - Vertex buffer와 Index buffer 메모리에 접근하기
 - Vertex buffer와 Index buffer에 대한 정보 얻기
- Render State
- Drawing 준비
 - Vertex buffer를 이용한 drawing
 - Vertex buffer와 index buffer를 이용한 drawing
 - Example
- D3DX 기하 물체 (geometry object)
- 예제 - 삼각형, 입방체, 주전자, D3DXCreate*

Vertex Buffer / Index Buffer 생성

- Vertex buffer, Index buffer
 - Vertex data와 index data를 보관하는 연속적인 memory 덩어리
 - Buffer를 video memory에 저장할 수 있어서 빠른 rendering이 가능
 - IDirect3DVertexBuffer9, IDirect3DIndexBuffer9
- Vertex buffer 생성

```
HRESULT IDirect3DDevice9::CreateVertexBuffer(  
    UINT Length,          // buffer size in bytes - n*sizeof(Vertex)  
    DWORD Usage,         // usage - 0 indicates no usage value D3DUSAGE_XXX  
    DWORD FVF,           // combination of D3DFVF  
    D3DPOOL Pool,        // member of D3DPOOL enum type  
    IDirect3DVertexBuffer9 **ppVertexBuffer, // created vertex buffer  
    HANDLE *pSharedHandle // set this to NULL  
);
```

Vertex Buffer / Index Buffer 생성

- Index buffer 생성

```
HRESULT IDirect3DDevice9::CreateIndexBuffer(  
    UINT Length,          // buffer size in bytes  
    DWORD Usage,         // usage value D3DUSAGE_XXX  
    DWORD Format,         // D3DFORMAT enum type - D3DFMT_INDEX16/32  
    D3DPOOL Pool,        // member of D3DPOOL enum type  
    IDirect3DIndexBuffer9 **ppIndexBuffer, // created index buffer  
    HANDLE *pSharedHandle // set this to NULL  
);
```

Vertex Buffer / Index Buffer 생성

□ D3DUSAGE constants

- D3DUSAGE_DYNAMIC: dynamic buffer를 생성함. Default는 지정하지 않을 시 static임.
 - Static buffer는 video memory(접근 속도가 느림)에 보관됨. 자주 바뀌지 않는 데이터의 경우에 유리함
 - Dynamic buffer는 AGP memory(빠른 속도로 갱신이 가능함)에 보관됨. Video memory로 전송해야 하므로 갱신이 없는 경우에는 static buffer 보다 느리나, 자주 갱신하는 경우 (즉, 매 프레임마다 기하정보를 갱신해야 하는 경우)에는 dynamic buffer가 빠름.
- D3DUSAGE_WRITEONLY
 - Buffer에 write만 수행할 것임을 지정함. 이 flag가 설정된 buffer에 read를 시도하면 오류가 발생함.
- Other flags

Vertex Buffer / Index Buffer 생성

- 예: Vertex형의 vertex 8개를 보관할 만큼의 static vertex buffer를 생성

```
IDirect3DVertexBuffer9* _vb;
_device->CreateVertexBuffer( 8*sizeof(Vertex),
                             0, // usage
                             D3DFVF_XYZ,
                             D3DPOOL_MANAGED,
                             &_vb, 0);
```

- 예: 36개의 16-bit index를 보관할 만큼의 dynamic index buffer를 생성

```
IDirect3DIndexBuffer9* _ib;
_device->CreateIndexBuffer( 36*sizeof(WORD),
                           D3DUSAGE_DYNAMIC | D3DUSAGE_WRITEONLY,
                           D3DFMT_INDEX16,
                           D3DPOOL_MANAGED,
                           &_ib, 0);
```

Vertex Buffer / Index Buffer 접근

□ Buffer 접근

1. Lock method를 사용하여 buffer pointer를 얻음
2. 읽기 및 쓰기 (D3DUSAGE_WRITEONLY 경우 읽기 불가)
3. 사용이 끝난 후에는 반드시 Unlock method를 이용해 buffer의 잠금을 해제

```
HRESULT IDirect3DVertexBuffer9::Lock(
    UINT OffsetToLock, // offset into the vertex data to lock (in bytes)
    UINT SizeToLock, // size of the vertex data to lock (in bytes)
    VOID **ppbData, // pointer to memory buffer of vertex data
    DWORD Flags // flags for the type of lock to perform, 0 또는 D3DLOCK_XXX
);
HRESULT IDirect3DIndexBuffer9::Lock(
    UINT OffsetToLock, // offset into the index data to lock (in bytes)
    UINT SizeToLock, // size of the index data to lock (in bytes)
    VOID **ppbData, // pointer to memory buffer of index data
    DWORD Flags // flags for the type of lock to perform
);
```

전체 buffer를 lock하려면 OffsetToLock = SizeToLock = 0으로 하면 index buffer를 lock함

Vertex Buffer / Index Buffer 접근

□ D3DLOCK constants

- D3DLOCK_DISCARD
 - dynamic buffer에만 사용가능. HW에게 이전 buffer를 버리라고 지시함. 새로 할당된 buffer에 접근하면서 동시에 버려진 buffer에서 rendering을 수행함. HW 지연을 막음.
- D3DLOCK_NOOVERWRITE
 - Dynamic buffer에만 사용가능. Buffer에 data를 추가하는 작업만 가능함. 현재 rendering 중인 memory에 overwrite할 수 없음. 새 data를 추가하면서 동시에 rendering할 수 있음.
- D3DLOCK_READONLY
 - Buffer를 read-only로 지정. 약간의 내부 최적화를 가능하게 함.

```
Vertex *vertices;
_vb->Lock(0, 0, (void*)&vertices, 0); // lock entire buffer
vertices[0] = Vertex(-1.0, 0.0, 2.0);
vertices[1] = Vertex(0.0, 1.0, 2.0);
vertices[2] = Vertex(1.0, 0.0, 2.0);
_vb->Unlock(); // unlock
```

Vertex Buffer / Index Buffer 정보 얻기

□ Vertex buffer/Index buffer 정보 얻기

```
D3DVERTEXBUFFER_DESC vbDescription;  
_vertexBuffer->GetDesc(&vbDescription); //retrieve description
```

```
D3DINDEXBUFFER_DESC ibDescription;  
_indexBuffer->GetDesc(&ibDescription); //retrieve description
```

□ D3DVERTEXBUFFER_DESC 구조체

```
typedef struct _D3DVERTEXBUFFER_DESC {  
    D3DFORMAT Format;           // describe the surface format of buffer  
    D3DRESOURCETYPE Type;     // identify this resource is a vertex buffer  
    DWORD Usage;              // combination of D3DUSAGE flags  
    D3DPOOL Pool;             // the class of memory allocated for the buffer  
    UNIT Size;                 // size of vertex buffer (in bytes)  
    DWORD FVF;                // describe vertex format of the vertices  
} D3DVERTEXBUFFER_DESC;
```

Vertex Buffer / Index Buffer 정보 얻기

□ D3DINDEXBUFFER_DESC 구조체

```
typedef struct _D3DINDEXBUFFER_DESC {  
    D3DFORMAT Format;           // describe the surface format of buffer  
    D3DRESOURCETYPE Type;     // identify this resource is a index buffer  
    DWORD Usage;              // usage  
    D3DPOOL Pool;             // the class of memory  
    UNIT Size;                 // size of index buffer (in bytes)  
} D3DINDEXBUFFER_DESC;
```

Render State

□ Render state

- Default 모드 값으로 동작함
- Default가 아닌 다른 값으로 바꿀 경우 SetRenderState을 호출

```
HRESULT IDirect3DDevice9::SetRenderState(  
    D3DRENDERSTATETYPE State, // device state variable to be modified  
    DWORD Value               // New value for the device render state to be set  
);
```

□ D3DRENDERSTATETYPE

- Enum of many state variables about 100
- 예: State가 D3DRS_SHADEMODE이면 Value는 D3DSHADEMODE enum형의 멤버를 주어야 함.

```
// 물체를 wireframe으로 rendering하고자 할 때  
_device->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);
```

Drawing 준비

□ Vertex buffer/ Index buffer를 만든 후 내용물을 rendering할 마지막 준비

1. **SetStreamSource**를 사용하여 Stream source 지정
 - Stream source를 vertex buffer와 연결하여 buffer의 기하정보를 rendering pipeline으로 보낼 수 있도록 함.

```
HRESULT IDirect3DDevice9::SetStreamSource(  
    UINT StreamNumber, // vertex buffer를 연결할 stream 소스 번호 지정  
    IDirect3DVertexBuffer9 *pStreamData, // 명시한 data stream과  
                                     // bind할 vertex buffer  
    UINT OffsetInBytes, // vertex buffer에서의 시작 위치 (in bytes)  
    UNIT Stride         // vertex buffer에의 각 요소의 byte 수  
);
```

```
_device->SetStreamSource(0, vb, 0, sizeof(Vertex));
```

Drawing 준비

2. **SetFVF**를 사용하여 Vertex format 지정
 - 이후의 drawing에서 이용될 vertex format을 지정함.

```
_device->SetFVF(D3DFVF_XYZ | D3DFVF_DIFFUSE | D3DFVF_TEX1);
```

3. **SetIndices**를 사용하여 Index buffer 지정
 - 이후의 drawing에서 이용될 index buffer을 지정함.
 - 한 번에 하나의 index buffer만 이용할 수 있음. 다른 index buffer를 가진 물체를 그리기 위해서는 index buffer를 전환하여야 함.

```
_device->SetIndices(_ib);
```

Vertex Buffer를 이용한 Drawing

- **DrawPrimitive**를 사용하여 index정보를 이용하지 않고 primitive를 drawing.

```
HRESULT IDirect3DDevice9::DrawPrimitive(  
    D3DPRIMITIVETYPE PrimitiveType, // rendering할 primitive type  
    UINT StartVertex,                // vertex buffer 의 첫번째 index  
    UINT PrimitiveCount              // rendering할 primitives 수  
    // 한번 호출 시마다 가능한 최대 개수는 D3DCAPS9.MaxPrimitiveCount임  
);  
  
// 4개의 삼각형을 그림  
_device->DrawPrimitives(D3DPT_TRIANGLELIST, 0, 4);
```

Vertex/Index Buffer를 이용한 Drawing

- **DrawIndexedPrimitive**를 사용하여 primitive를 drawing

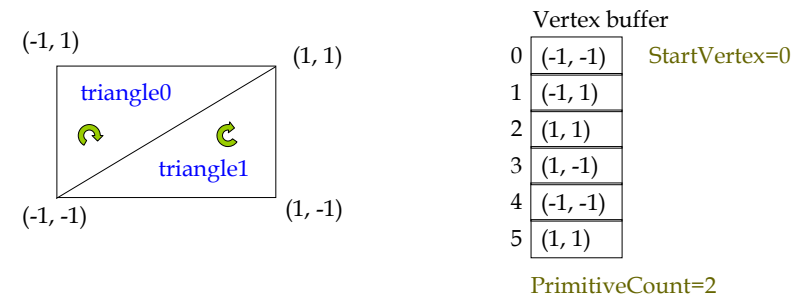
```
HRESULT IDirect3DDevice9::DrawIndexedPrimitive(  
    D3DPRIMITIVETYPE PrimitiveType, // rendering할 primitive type  
    INT BaseVertexIndex, // vertex buffer에서 첫번째 vertex의 offset  
    UINT MinIndex,       // 이번 호출에 이용될 vertex들의 최소 vertex index  
    UINT NumVertices,    // 이번 호출에 이용될 vertex들의 수  
    UINT StartIndex,     // vertex buffer를 접근하는 index buffer에서 첫번째 index  
    UINT PrimitiveCount // rendering할 primitives 수  
);
```

```
//MinIndex와 index stream에 대한 모든 index들은 BaseVertexIndex에 상대적임  
//MinIndex와 NumVertices는 사용될 vertex index들의 범위를 명시함
```

```
_device->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 8, 0, 12);
```

Drawing Example

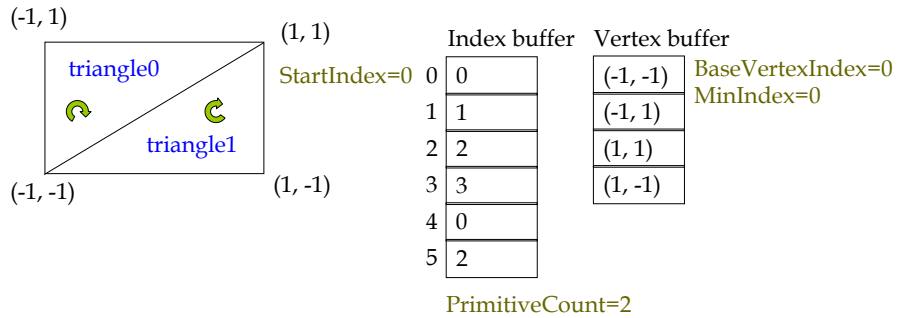
- Eg.1: 2개의 삼각형을 index를 사용하지 않고 drawing



```
DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);
```

Drawing Example

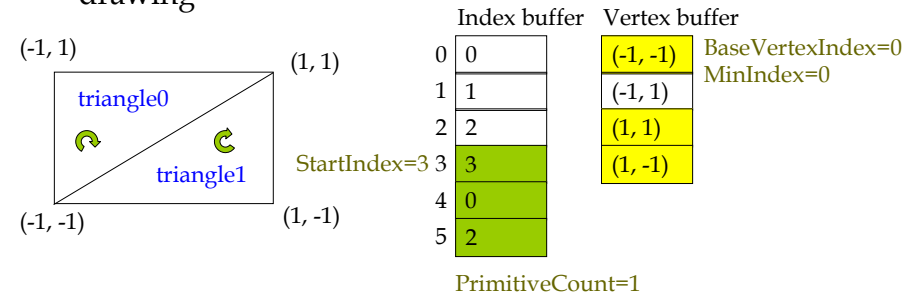
□ Eg.2: 2개의 삼각형을 index를 사용해서 drawing



`DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 4, 0, 2);`

Drawing Example

□ Eg.3: 1개의 삼각형(즉, 두 번째 것)을 index를 사용해서 drawing

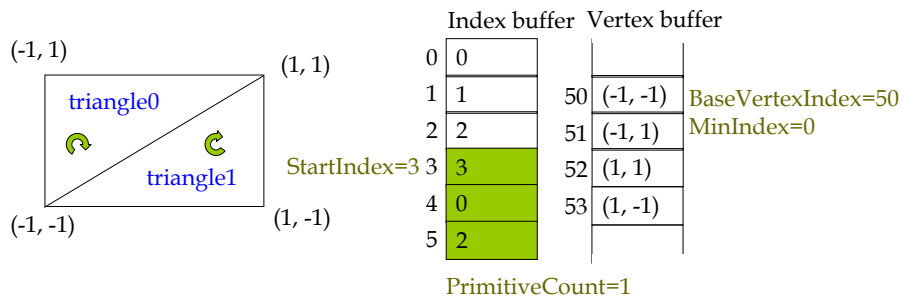


`DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 4, 3, 1);`

Drawing Example

□ Eg.4: Offset indexing을 사용해서 1개의 삼각형을 drawing

- 일반적으로 BaseVertexIndex를 0으로 지정함.
- 그러나 vertex data의 구조는 알지만 data가 놓여질 위치를 아직 모를 경우, 또는 동일 구조의 vertex data가 vertex buffer에 여러 번 놓여있는 경우 이를 하나의 index buffer로 그릴 경우 유용함.



`DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 50, 0, 4, 3, 1);`

BeginScene / EndScene

□ 모든 drawing method들은

`IDirect3DDevice9::BeginScene`과
`IDirect3DDevice9::EndScene` 내부에 포함되어야 함.

```

_device->BeginScene();
...
_device->DrawPrimitive(...);
...
_device->EndScene();
    
```

- 새로운 BeginScene은 다른 BeginScene/EndScene 안에 불릴 수 없음.
- `IDirect3DDevice9::Present` 또는 `IDirect3DSwapChain9::Present`가 불려질 때 적어도 하나의 BeginScene/EndScene이 존재해야 함.
- EndScene이 Present전에 불려야 함.

D3DX Geometry

- 기하 자료를 간편하게 구성하도록 하기 위해
 - 6가지 간단한 물체의 mesh data를 생성하는 method를 제공
 - D3DXCreateBox, D3DXCreateSphere, D3DXCreateCylinder, D3DXCreateTeapot, D3DXCreatePolygon, D3DXCreateTorus

```
HRESULT WINAPI D3DXCreateTeapot(
    LPDIRECT3DDEVICE9 pDevice,
    LPD3DXMESH **ppMesh, // output here
    LPD3DXBUFFER **ppAdjacency
        // array of three DWORDs per face
        // that specify the three neighbors for each face
        // NULL can be specified
);
ID3DXMesh* mesh = 0;
D3DXCreateTeapot(_device, &mesh, 0);
```

D3DX Geometry

- Mesh data의 drawing은 ID3DXMesh::DrawSubset을 호출
 - Mesh의 subset을 지정해야 함
 - D3DXCreate* 함수로 생성된 mesh는 1개의 subset만을 가짐.
`_device->BeginScene();`
`mesh->DrawSubset(0);`
`_device->EndScene();`
- Mesh 이용이 끝난 뒤에는 이를 해제해 주어야 함.
`mesh->Release();`
`mesh = 0;`

Examples

- Triangle - wireframe 삼각형을 그림
 - Vertex buffer, render state, drawing commands
- Cube - 돌고 있는 wireframe 입방체를 그림
 - Vertex buffer, index buffer, world transformation, view transformation, render states, drawing commands
- Teapot - wireframe 주전자를 그림
 - D3DXCreateTeapot 함수 사용, ID3DXMesh::DrawSubset 방법
- D3DXCreate - wireframe D3DX shape들을 그림.
카메라가 장면 위를 자유롭게 움직임.
 - D3DXCreate* 함수 사용, 복잡한 transformation, camera movement

Triangle

```
#include "d3dUtility.h"
//
// Globals
//

IDirect3DDevice9* Device = 0;

const int Width = 640;
const int Height = 480;

IDirect3DVertexBuffer9* Triangle = 0; // vertex buffer to store
// our triangle data.

//
// Classes and Structures
//

struct Vertex
{
    Vertex(){}

    Vertex(float x, float y, float z)
    {
        _x = x; _y = y; _z = z;
    }

    float _x, _y, _z;

    static const DWORD FVF;
};
const DWORD Vertex::FVF = D3DFVF_XYZ;
```

Triangle

```
bool Setup()
{
    //
    // Create the vertex buffer.

    Device->CreateVertexBuffer(
        3 * sizeof(Vertex), // size in bytes
        D3DUSAGE_WRITEONLY, // flags
        Vertex::FVF, // vertex format
        D3DPPOOL_MANAGED, // managed memory pool
        &Triangle, // return create vertex buffer
        0); // not used - set to 0

    //
    // Fill the buffers with the triangle data.
    //

    Vertex* vertices;
    Triangle->Lock(0, 0, (void*)&vertices, 0);

    vertices[0] = Vertex(-1.0f, 0.0f, 2.0f);
    vertices[1] = Vertex( 0.0f, 1.0f, 2.0f);
    vertices[2] = Vertex( 1.0f, 0.0f, 2.0f);

    Triangle->Unlock();

    //
    // Set the projection matrix.
    //

    D3DXMATRIX proj;
    D3DXMatrixPerspectiveFovLH(
        &proj, // result
        D3DX_PI * 0.5f, // 90 - degrees
        (float)Width / (float)Height, // aspect ratio
        1.0f, // near plane
        1000.0f); // far plane
    Device->SetTransform(D3DTS_PROJECTION, &proj);

    //
    // Set wireframe mode render state.
    //

    Device->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);

    return true;
}
```

Triangle

```
void Cleanup()
{
    d3d->Release<IDirect3DVertexBuffer9*>(Triangle);
}

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

        Device->SetStreamSource(0, Triangle, 0, sizeof(Vertex));
        Device->SetFVF(Vertex::FVF);

        // Draw one triangle.
        Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);

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

//
// WndProc
//
LRESULT CALLBACK d3d::WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch( msg )
    {
        case WM_DESTROY:
            ::PostQuitMessage(0);
            break;

        case WM_KEYDOWN:
            if( wParam == VK_ESCAPE )
                ::DestroyWindow(hwnd);
            break;
    }
    return ::DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Triangle

```
int WINAPI WinMain(HINSTANCE hinstance,
                  HINSTANCE previnstance,
                  PSTR cmdLine,
                  int showCmd)
{
    if(!d3d::InitD3D(hinstance,
                    Width, Height, true, D3DDEVTYPE_HAL, &Device))
    {
        ::MessageBox(0, "InitD3D() - FAILED", 0, 0);
        return 0;
    }

    if(!Setup())
    {
        ::MessageBox(0, "Setup() - FAILED", 0, 0);
        return 0;
    }

    d3d::EnterMsgLoop( Display );

    Cleanup();

    Device->Release();

    return 0;
}
```

Cube

In Setup() - create vertex/index buffer

```
bool Setup()
{
    //
    // Create vertex and index buffers.
    //

    Device->CreateVertexBuffer(
        8 * sizeof(Vertex),
        D3DUSAGE_WRITEONLY,
        Vertex::FVF,
        D3DPPOOL_MANAGED,
        &VB,
        0);

    Device->CreateIndexBuffer(
        36 * sizeof(WORD),
        D3DUSAGE_WRITEONLY,
        D3DFMT_INDEX16,
        D3DPPOOL_MANAGED,
        &IB,
        0);
}
```

Cube

In Setup() – define vertex

```
//
// Fill the buffers with the cube data.
//
// define unique vertices:
Vertex* vertices;
VB->Lock(0, 0, (void*)&vertices, 0);

// vertices of a unit cube
vertices[0] = Vertex(-1.0f, -1.0f, -1.0f);
vertices[1] = Vertex(-1.0f, 1.0f, -1.0f);
vertices[2] = Vertex(1.0f, 1.0f, -1.0f);
vertices[3] = Vertex(1.0f, -1.0f, -1.0f);
vertices[4] = Vertex(-1.0f, -1.0f, 1.0f);
vertices[5] = Vertex(-1.0f, 1.0f, 1.0f);
vertices[6] = Vertex(1.0f, 1.0f, 1.0f);
vertices[7] = Vertex(1.0f, -1.0f, 1.0f);

VB->Unlock();
```

Cube

In Setup() – define indices

```
// define the triangles of the cube:
WORD* indices = 0;
IB->Lock(0, 0, (void*)&indices, 0);

// front side
indices[0] = 0; indices[1] = 1; indices[2] = 2;
indices[3] = 0; indices[4] = 2; indices[5] = 3;

// back side
indices[6] = 4; indices[7] = 6; indices[8] = 5;
indices[9] = 4; indices[10] = 7; indices[11] = 6;

// left side
indices[12] = 4; indices[13] = 5; indices[14] = 1;
indices[15] = 4; indices[16] = 1; indices[17] = 0;

// right side
indices[18] = 3; indices[19] = 2; indices[20] = 6;
indices[21] = 3; indices[22] = 6; indices[23] = 7;

// top
indices[24] = 1; indices[25] = 5; indices[26] = 6;
indices[27] = 1; indices[28] = 6; indices[29] = 2;

// bottom
indices[30] = 4; indices[31] = 0; indices[32] = 3;
indices[33] = 4; indices[34] = 3; indices[35] = 7;

IB->Unlock();
```

Cube

In Setup() – define camera & projection matrix

```
//
// Position and aim the camera.
//
D3DXVECTOR3 position(0.0f, 0.0f, -5.0f);
D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);
D3DXMATRIX V;
D3DXMatrixLookAtLH(&V, &position, &target, &up);

Device->SetTransform(D3DTS_VIEW, &V);

//
// Set the projection matrix.
//
D3DXMATRIX proj;
D3DXMatrixPerspectiveFovLH(
    &proj,
    D3DX_PI * 0.5f, // 90 - degree
    (float)Width / (float)Height,
    1.0f,
    1000.0f);
Device->SetTransform(D3DTS_PROJECTION, &proj);

//
// Switch to wireframe mode.
//

Device->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);

return true;
}
```

Cube

In Display() – rotation matrix for cube spinning & draw cube

```
bool Display(float timeDelta)
{
    if( Device )
    {
        //
        // spin the cube:
        //
        D3DXMATRIX Rx, Ry;

        // rotate 45 degrees on x-axis
        D3DXMatrixRotationX(&Rx, 3.14f / 4.0f);

        // increment y-rotation angle each frame
        static float y = 0.0f;
        D3DXMatrixRotationY(&Ry, y);
        y += timeDelta;

        // reset angle to zero when angle reaches 2*PI
        if( y == 6.28f )
            y = 0.0f;

        // combine x- and y-axis rotation transformations.
        D3DXMATRIX p = Rx * Ry;

        Device->SetTransform(D3DTS_WORLD, &p);

        //
        // draw the scene:
        //
        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
        Device->BeginScene();

        Device->SetStreamSource(0, VB, 0, sizeof(Vertex));
        Device->SetIndices(IB);
        Device->SetFVF(Vertex::FVF);

        // Draw cube.
        Device->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 8, 0, 12);

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


Teapot

```
bool Setup()
{
    //
    // Create the teapot geometry.
    //

    D3DXCreateTeapot(Device, &Teapot, 0);

    //
    // Position and aim the camera.
    //

    D3DXVECTOR3 position(0.0f, 0.0f, -3.0f);
    D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
    D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);
    D3DMATRIX V;
    D3DXMatrixLookAtLH(&V, &position, &target, &up);
    Device->SetTransform(D3DTS_VIEW, &V);

    //
    // Set projection matrix.
    //

    D3DMATRIX proj;
    D3DXMatrixPerspectiveFovLH(
        &proj,
        D3DX_PI * 0.5f, // 90 - degree
        (float)Width / (float)Height,
        1.0f,
        1000.0f);
    Device->SetTransform(D3DTS_PROJECTION, &proj);

    //
    // Switch to wireframe mode.
    //

    Device->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);

    return true;
}
```

Teapot

```
void Cleanup()
{
    d3d::Release<ID3DXMesh*>(Teapot);
}

bool Display(float timeDelta)
{
    if( Device )
    {
        //
        // Spin the teapot:
        //

        D3DMATRIX Ry;
        static float y = 0.0f;
        D3DXMatrixRotationY(&Ry, y);

        y += timeDelta;
        if(y >= 6.28f)
            y = 0.0f;

        Device->SetTransform(D3DTS_WORLD, &Ry);

        //
        // Draw the Scene:
        //

        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
        Device->BeginScene();

        // Draw teapot using DrawSubset method with 0 as the argument.
        Teapot->DrawSubset(0);

        Device->EndScene();
        Device->Present(0, 0, 0, 0);

    }
    return true;
}
```

D3DXCreate

```
bool Setup()
{
    //
    // Create the objects.
    //

    D3DXCreateTeapot(
        Device,
        &objects[0],
        0);

    D3DXCreateBox(
        Device,
        2.0f, // width
        2.0f, // height
        2.0f, // depth
        &objects[1],
        0);

    // cylinder is built aligned on z-axis
    D3DXCreateCylinder(
        Device,
        1.0f, // radius at negative z end
        1.0f, // radius at positive z end
        3.0f, // length of cylinder
        10, // slices
        10, // stacks
        &objects[2],
        0);

    D3DXCreateTorus(
        Device,
        1.0f, // inner radius
        3.0f, // outer radius
        10, // sides
        10, // rings
        &objects[3],
        0);

    D3DXCreateSphere(
        Device,
        1.0f, // radius
        10, // slices
        10, // stacks
        &objects[4],
        0);
}
```

D3DXCreate

```
//
// Build world matrices - position the objects in world space.
// For example, ObjWorldMatrices[1] will position Objects[1] at
// (-5, 0, 5). Likewise, ObjWorldMatrices[2] will position
// Objects[2] at (5, 0, 5).
//

D3DXMatrixTranslation(&ObjWorldMatrices[0], 0.0f, 0.0f, 0.0f);
D3DXMatrixTranslation(&ObjWorldMatrices[1], -5.0f, 0.0f, 5.0f);
D3DXMatrixTranslation(&ObjWorldMatrices[2], 5.0f, 0.0f, 5.0f);
D3DXMatrixTranslation(&ObjWorldMatrices[3], -5.0f, 0.0f, -5.0f);
D3DXMatrixTranslation(&ObjWorldMatrices[4], 5.0f, 0.0f, -5.0f);

//
// Set the projection matrix.
//

D3DMATRIX proj;
D3DXMatrixPerspectiveFovLH(
    &proj,
    D3DX_PI * 0.5f, // 90 - degree
    (float)Width / (float)Height,
    1.0f,
    1000.0f);
Device->SetTransform(D3DTS_PROJECTION, &proj);

//
// Switch to wireframe mode.
//

Device->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);

return true;
}

void Cleanup()
{
    for(int i = 0; i < 5; i++)
        d3d::Release<ID3DXMesh*>(Objects[i]);
}
```

```

bool Display(float timeDelta)
{
    if( Device )
    {
        // Animate the camera:
        // The camera will circle around the center of the scene. We use the
        // sin and cos functions to generate points on the circle, then scale them
        // by 10 to further the radius. In addition the camera will move up and down as it circles about the scene.
        static float angle = (3.0f * D3DX_PI) / 2.0f;
        static float cameraHeight = 0.0f;
        static float cameraHeightDirection = 5.0f;

        D3DXVECTOR3 position( cosf(angle) * 10.0f, cameraHeight, sinf(angle) * 10.0f );

        // the camera is targetted at the origin of the world
        D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);

        // the worlds up vector
        D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);

        D3DXMATRIX V;
        D3DXMatrixLookAtLH(&V, &position, &target, &up);
        Device->SetTransform(D3DTS_VIEW, &V);

        // compute the position for the next frame
        angle += timeDelta;
        if( angle >= 6.28f )
            angle = 0.0f;

        // compute the height of the camera for the next frame
        cameraHeight += cameraHeightDirection * timeDelta;
        if( cameraHeight >= 10.0f )
            cameraHeightDirection = -5.0f;

        if( cameraHeight <= -10.0f )
            cameraHeightDirection = 5.0f;

        // Draw the Scene:
        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
        Device->BeginScene();

        for(int i = 0; i < 5; i++)
        {
            // Set the world matrix that positions the object.
            Device->SetTransform(D3DTS_WORLD, &ObjWorldMatrices[i]);

            // Draw the object using the previously set world matrix.
            Objects[i]->DrawSubset(0);
        }

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

```