Direct3D Drawing

305890
Spring 2010
4/9/2010
Kyoung Shin Park

Drawing

- Vertex Buffer & Index Buffer
  - Creating vertex buffer & index buffer
  - Accessing vertex & index buffer memory
  - Getting vertex buffer & index buffer information
- Render State
- Drawing Preparations
  - Vertex buffer drawing
  - Vertex buffer& index buffer drawing
  - Example
- D3DX Geometry Object

Vertex Buffer / Index Buffer

- Vertex buffer & Index buffer
  - Vertex buffer is simply a chunk of contiguous memory that contains vertex data
  - Index buffer is a chunk of contiguous memory that contains index data
  - IDirect3DVertexBuffer9, IDirect3DIndexBuffer9
- Creating a vertex buffer

```
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
    HANDEL *pSharedHandle                    // set this to NULL
);
```

Vertex Buffer / Index Buffer

- Creating an index buffer

```
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
    HANDEL *pSharedHandle                    // set this to NULL
);
```

Vertex Buffer / Index Buffer

- D3DUSAGE constants
  - D3DUSAGE_DYNAMIC
    - Setting this flag makes the buffer dynamic (default is static)
    - 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
    - Specifies that the application will only write to the buffer
    - This allows the driver to place the buffer in the best memory location for write operations
    - Reading from a buffer created with this flag will result in an error
  - Other flags

Vertex Buffer / Index Buffer

- This example creates a static vertex buffer that has enough memory to hold 8 vertices of Vertex type:

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

- This example shows how to create a dynamic index buffer that has enough memory to hold 36 16-bit indices:

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

Accessing a Buffer’s Memory

- Accessing a buffer memory
  1. Obtain a pointer to its content by using “Lock” method
  2. Read and write
  3. “Unlock” the buffer when done accessing it

```
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 or 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

Accessing a Buffer’s Memory

- D3DLOCK constants
  - D3DLOCK_DISCARD
    - Only used for dynamic buffer. It instructs HW to discard the buffer and return a pointer to a newly allocated buffer. This prevents HW from stalling by allowing HW to continue rendering from the discarded buffer while we access the newly allocated buffer.
  - D3DLOCK_NOOVERWRITE
    - Only used for dynamic buffer. It prevents HW from stalling by allowing HW to continue rendering previously written geometry at the same time we append new geometry.
  - D3DLOCK_READONLY
    - Locking the buffer read-only. It allows for internal optimizations.

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

Getting a Vertex & Index Buffer Info

- Getting Vertex buffer/Index buffer information

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

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

- D3DVERTEXBUFFER_DESC

```
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;
```

Getting a Vertex & Index Buffer Info

- D3DINDEXBUFFER_DESC

```
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
  - "SetRenderState" is used to specify rendering states other than default value

```
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
  - D3DRS_FILLMODE rendering state => D3DFILLMODE enum value

```
// to draw wireframe mode rendering
_device->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);
// to draw solid fill mode rendering
_device->SetRenderState(D3DRS_FILLMODE, D3DFILL_SOLID);
```

Drawing Preparations

- Drawing Preparations
  1. Hook the vertex buffer to a vertex stream using SetStreamSource

```
HRESULT IDirect3DDevice9::SetStreamSource(
    UINT StreamNumber, // identifies the stream source
                       // use 0, since we do not use multiple streams
    IDirect3DVertexBuffer9 *pStreamData, // a pointer to the vertex buffer
                                         // to hook up to the stream
    UINT OffsetInBytes,   // offset from the start of the stream (in bytes)
    UNIT Stride           // size (in bytes) of each element in vertex buffer
);

// vb is a pointer to a vertex buffer that has been filled with vertices of Vertex type
_device->SetStreamSource(0, vb, 0, sizeof(Vertex));
```

Drawing Preparations

2. Hook the index buffer to an index stream

```
// ib is a pointer to an IDirect3DIndexBuffer9 type
_device->SetIndices(ib);
```

3. Setting the Vertex Declarations
   - We need to create a vertex declaration to describe the format of the vertex we are using.

```
// decl is a pointer to an IDirect3DVertexDeclaration9 type
_device->SetVertexDeclaration(decl);
```

Vertex Buffer Drawing

- DrawPrimitive
  - This method is used to draw primitives that do not use index

```
HRESULT IDirect3DDevice9::DrawPrimitive(
    D3DPRIMITIVETYPE PrimitiveType,   // primitive type
    UINT StartVertex,                 // index to an element in the vertex
                                      // buffer for starting point
    UINT PrimitiveCount               // number of primitives to draw
    // max number for PrimitiveCount is D3DCAPS9.MaxPrimitiveCount
);

// draw 4 triangles
_device->DrawPrimitives(D3DPT_TRIANGLELIST, 0, 4);
```

Vertex/Index Buffer Drawing

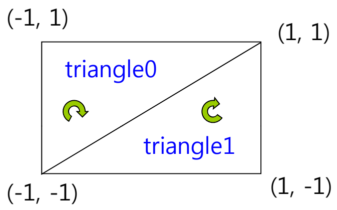
- DrawIndexedPrimitive

```
HRESULT IDirect3DDevice9::DrawIndexedPrimitive(
    D3DPRIMITIVETYPE PrimitiveType,      // primitive type
    INT BaseVertexIndex, // a base number to be added to the indices used
    UINT MinIndex,       // minimum index value that will be referenced
    UINT NumVertices,    // number of vertices that will be referenced
    UINT StartIndex,     // index to an element in the index buffer for starting point
    UINT PrimitiveCount // number of primitives to draw
);

// draw a geometry consisting of 12 triangles and 8 vertices
_device->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 8, 0, 12);
```

Drawing Example

- Example: draw 2 triangles using DrawPrimitive



Vertex buffer

0	(-1, -1)	StartVertex=0
1	(-1, 1)	
2	(1, 1)	
3	(1, -1)	
4	(-1, -1)	
5	(1, 1)	

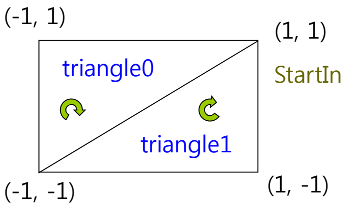
PrimitiveCount=2

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

Direct3D LHS CW

Drawing Example

- Example: draw 2 triangles using DrawIndexedPrimitive



StartIndex=0

	Index buffer	Vertex buffer
0	0	(-1, -1)
1	1	(-1, 1)
2	2	(1, 1)
3	3	(1, -1)
4	0	
5	2	

BaseVertexIndex=0
MinIndex=0

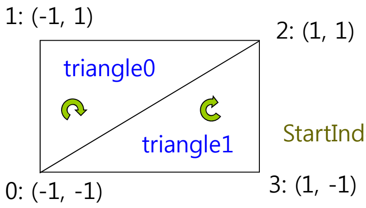
NumVertices=4

PrimitiveCount=2

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

Drawing Example

- Example: draw 1 triangle (i.e., 2nd one) specifying StartIndex in DrawIndexedPrimitives



StartIndex=3

	Index buffer	Vertex buffer
0	0	(-1, -1)
1	1	(-1, 1)
2	2	(1, 1)
3	3	(1, -1)
4	0	
5	2	

BaseVertexIndex=0
MinIndex=0

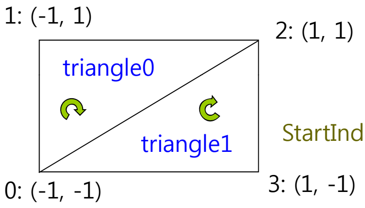
NumVertices=4

PrimitiveCount=1

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

Drawing Example

- Example: draw 1 triangle specifying BaseVertexIndex in DrawIndexedPrimitives



StartIndex=3

	Index buffer
0	0
1	1
2	2
3	3
4	0
5	2

	Vertex buffer
50	(-1, -1)
51	(-1, 1)
52	(1, 1)
53	(1, -1)

BaseVertexIndex=50
MinIndex=0

NumVertices=4

PrimitiveCount=1

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

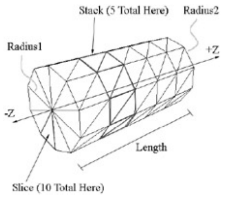
BeginScene / EndScene

- Drawing methods must always be called inside IDirect3DDevice9::BeginScene and IDirect3DDevice::EndScene pair.

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

D3DX Geometry Object

- D3DX library provies 6 mesh data creation functions:
  - D3DXCreateBox
  - D3DXCreateSphere
  - D3DXCreateCylinder // make a cone by setting one of radii to 0
  - D3DXCreateTorus
  - D3DXCreateTeapot
  - D3DXCreatePolygon



```
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 Object

- To draw mesh data, we call ID3DXMesh::DrawSubset
  - One subset is used for a mesh created by D3DXCreate* functions.

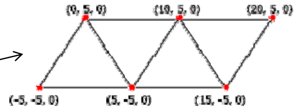
```
_device->BeginScene();
mesh->DrawSubset(0);
_device->EndScene();
```

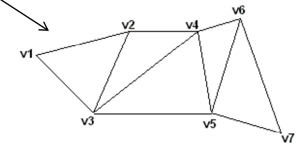
- We must release them when done using mesh data

```
mesh->Release();
mesh = 0;
```

Primitive Types

- Some primitive types are
  - D3DPT_POINTLIST
  - D3DPT_LINELIST
  - D3DPT_LINESTRIP
  - **D3DPT_TRIANGLELIST**
  - **D3DPT_TRIANGLESTRIP**
  - D3DPT_TRIANGLEFAN





v1 v2 v3 v4 v5 v6 v7
DrawPrimitive(D3DPT_TRANGLESTRIP, 0, 5)

Primitive Types

```
typedef enum D3DPRIMITIVETYPE {
  D3DPT_POINTLIST = 1,
  D3DPT_LINELIST = 2,
  D3DPT_LINESTRIP = 3,
  D3DPT_TRIANGLELIST = 4,
  D3DPT_TRIANGLESTRIP = 5,
  D3DPT_TRIANGLEFAN = 6,
  D3DPT_FORCE_DWORD = 0x7fffffff
} D3DPRIMITIVETYPE, *LPD3DPRIMITIVETYPE;
```