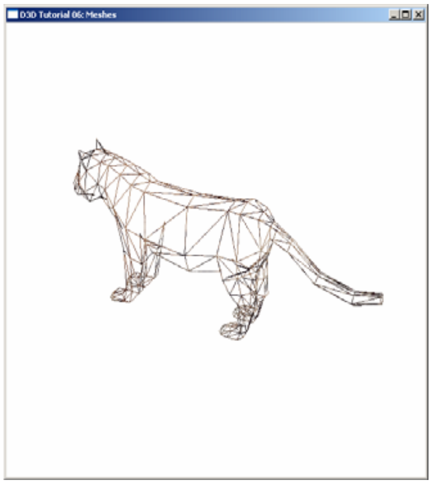
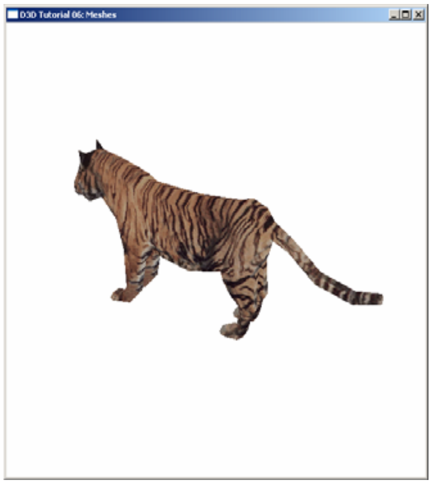
Meshes

- Meshes what are they?
- How to use them?
- Progressive Meshes?

Meshes





What is a mesh?

- A collection of vertices, edges and faces which define the geometric shape of a a 3D object.

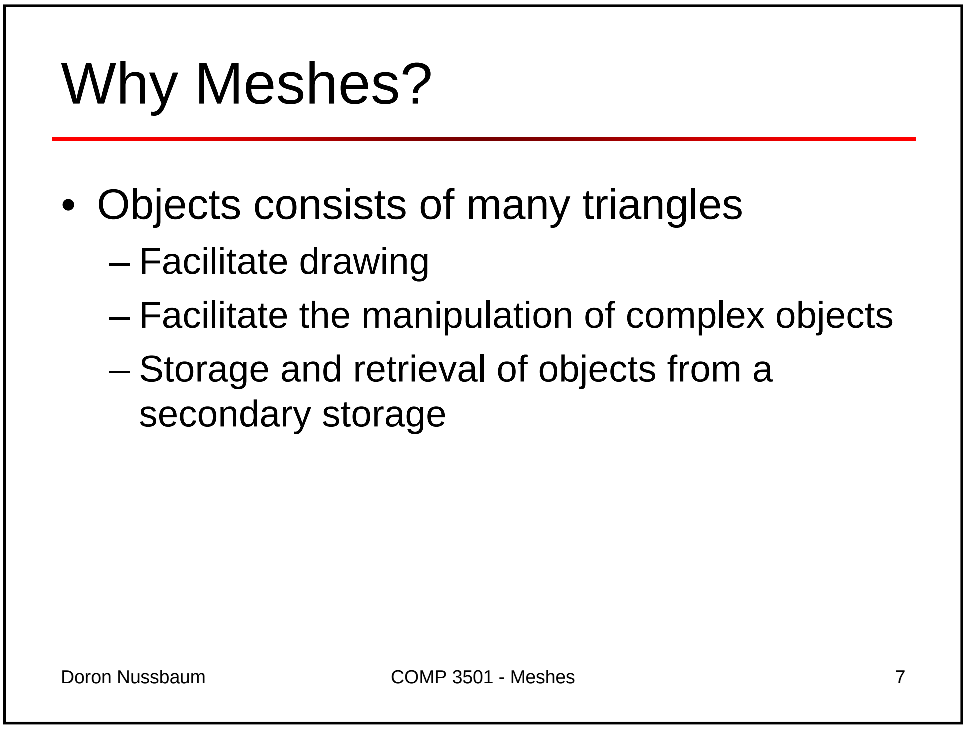
- In our case the faces are
  - Triangles
  - Quads

What information is required

- Mesh data consists of two types of data
  - Geometrical
    - Where is the object in space
    - How does its shape look
  - Topological –
    - abstract relationship of the object's geometry
- Minimum required information
  - Geometry – vertices – geometry
  - Topology – faces or edges

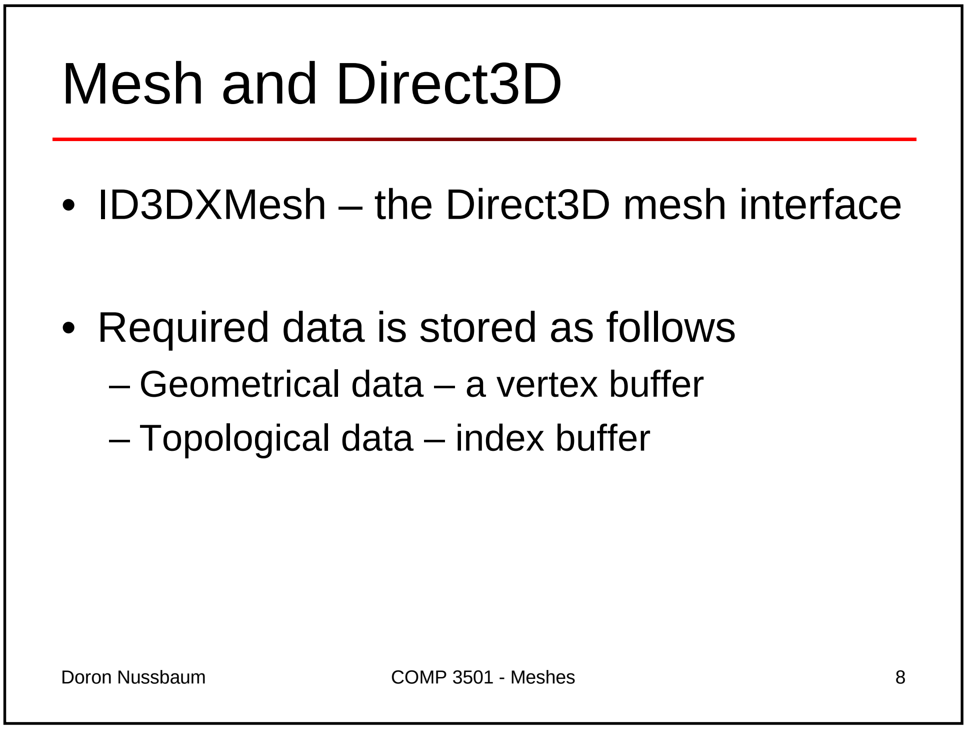
Mesh representation

- A number of ways to represent the data
  - Doubly Connected Edge List
  - Graph representations
- Complete representation
  - All data is stored and pre-comupted
- Partial representation
  - Only part of the topological data is stored
  - Missing information can be constructed as needed



Why Meshes?

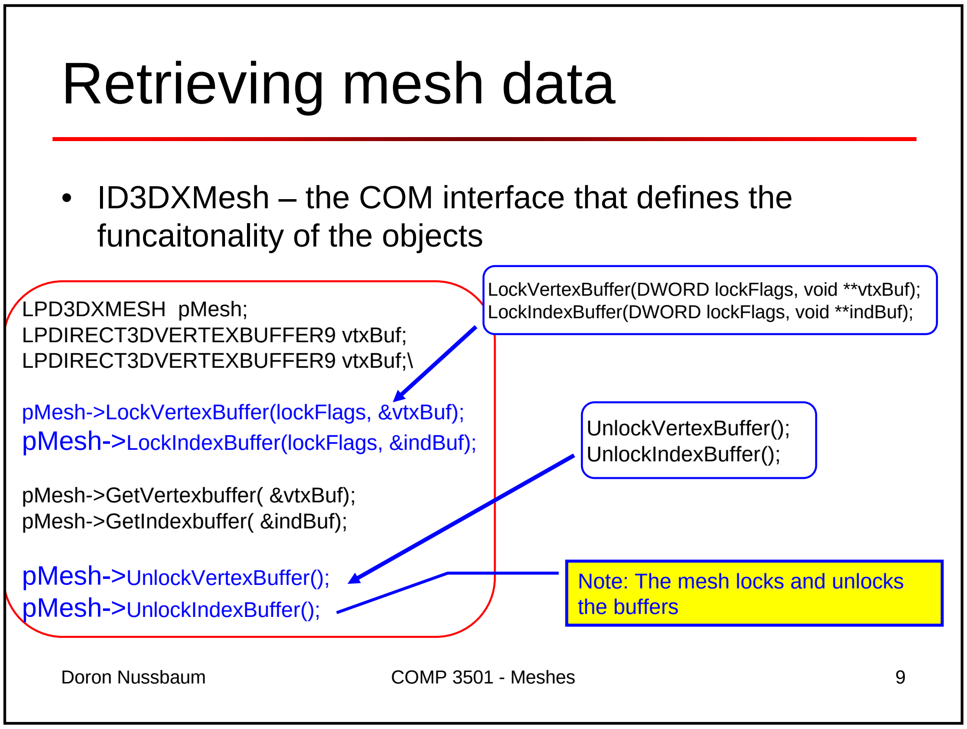
- Objects consists of many triangles
  - Facilitate drawing
  - Facilitate the manipulation of complex objects
  - Storage and retrieval of objects from a secondary storage



Mesh and Direct3D

- ID3DXMesh – the Direct3D mesh interface

- Required data is stored as follows
  - Geometrical data – a vertex buffer
  - Topological data – index buffer



Retrieving mesh data

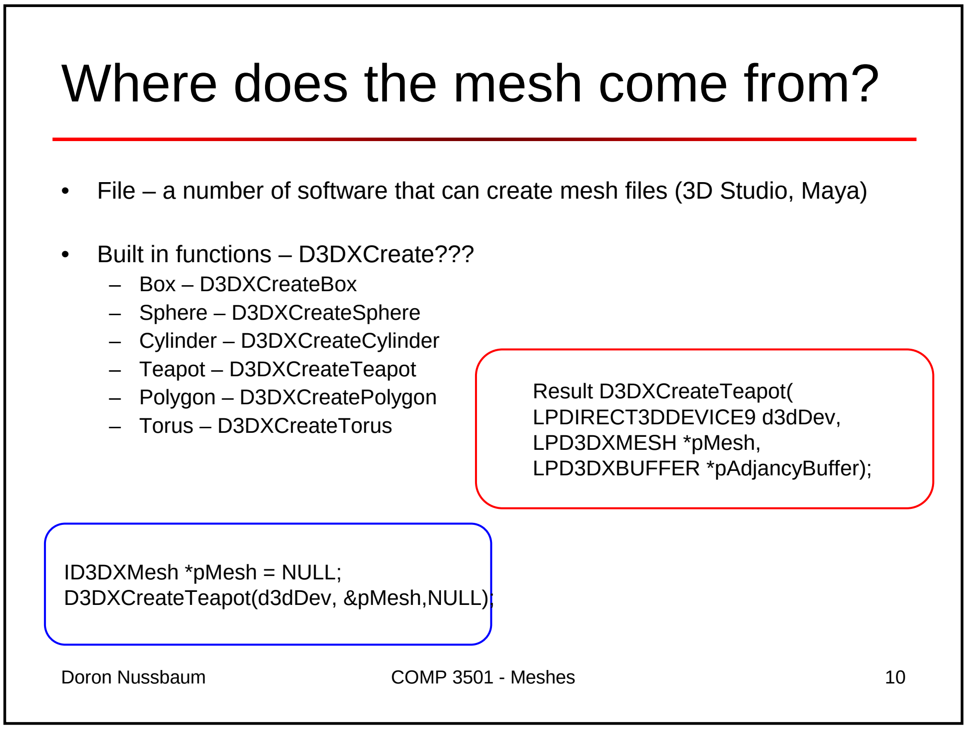
- ID3DXMesh – the COM interface that defines the funcaitonality of the objects

```
LPD3DXMESH  pMesh;
LPDIRECT3DVERTEXBUFFER9 vtxBuf;
LPDIRECT3DVERTEXBUFFER9 vtxBuf;\

pMesh->LockVertexBuffer(lockFlags, &vtxBuf);
pMesh->LockIndexBuffer(lockFlags, &indBuf);

pMesh->GetVertexbuffer( &vtxBuf);
pMesh->GetIndexbuffer( &indBuf);

pMesh->UnlockVertexBuffer();
pMesh->UnlockIndexBuffer();
```

```
LockVertexBuffer(DWORD lockFlags, void **vtxBuf);
LockIndexBuffer(DWORD lockFlags, void **indBuf);
```

```
UnlockVertexBuffer();
UnlockIndexBuffer();
```

Note: The mesh locks and unlocks the buffers



Where does the mesh come from?

- File – a number of software that can create mesh files (3D Studio, Maya)
- Built in functions – D3DXCreate???
  - Box – D3DXCreateBox
  - Sphere – D3DXCreateSphere
  - Cylinder – D3DXCreateCylinder
  - Teapot – D3DXCreateTeapot
  - Polygon – D3DXCreatePolygon
  - Torus – D3DXCreateTorus

```
Result D3DXCreateTeapot(
LPDIRECT3DDEVICE9 d3dDev,
LPD3DXMESH *pMesh,
LPD3DXBUFFER *pAdjancyBuffer);
```

```
ID3DXMesh *pMesh = NULL;
D3DXCreateTeapot(d3dDev, &pMesh,NULL);
```

Vertex Buffer Navigation Functions

- DWORD GetFVF() – returns the flags that define the format of the vertices

- DWORD GetNumVertices() – returns the number of vertices in the vertex buffer

- DWORD GetNumBytesPerVertex() – returns the number of vertices in the vertex buffer

- DWORD GetNumFaces() – returns the number of vertices in the vertex buffer

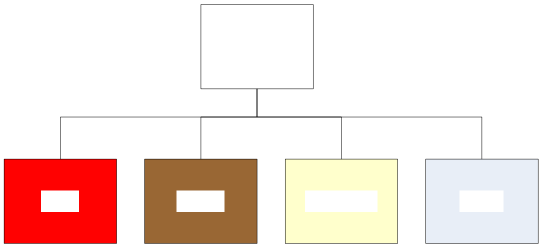
Managing the Mesh Triangles

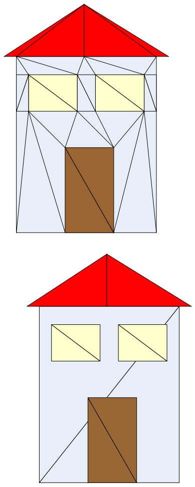
- Rendering Needs
  - Manipulate the geometry
    - Move hands of a mesh which represents a human
  - Manipulate the cladding
    - Terrain – draw lakes, mountains, gras

- Solution
  - Organize the mesh into groups/subsets
  - Enable one to manipulate each subset independently

Mesh Subsets

- Divide the mesh into subsets



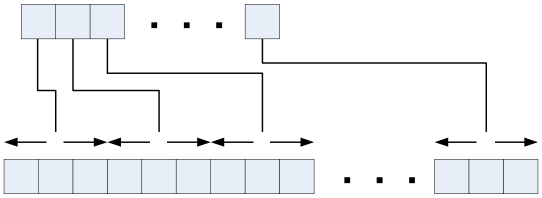


Meshes Subsets

- Organize the set of mesh triangles into subsets.
- Each subset would be rendered and/or manipulated using the same attributes
  - Rendering – material, texture, render states…
- Mechanism
  - Assign each triangle in the mesh an attribute id
  - Issues rendering command based on the attribute id

Assigning Attributes

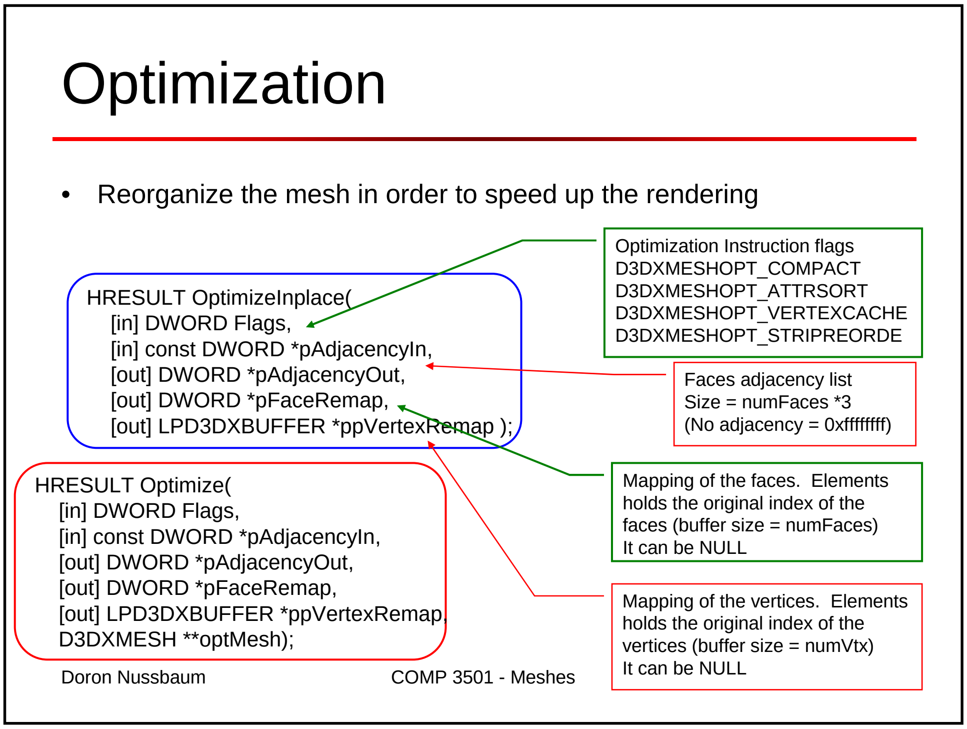
- Use an attributed buffer
- 1-1 correspondence between triangles and attribute entity



```
// managing the attribute buffer

long *buf = NULL;
pMesh->LockAttributeBuffer(lockingFlags, &buffer);

// read or write from the buffer

pMesh->UnlockAttributeBuffer();
```

```
//Rendering the mesh subsets

for (i = 0; i < numSubsets; i++) {
    d3dDev->SetMaterial(0,matBuf[i]);
    d3dDev->SetTexture(0,textureBuf[i]);
   pMesh->DrawSubset[i];
}
```

0 15



Optimization

- Reorganize the mesh in order to speed up the rendering

```
HRESULT OptimizeInplace(
    [in] DWORD Flags,
    [in] const DWORD *pAdjacencyIn,
    [out] DWORD *pAdjacencyOut,
    [out] DWORD *pFaceRemap,
    [out] LPD3DXBUFFER *ppVertexRemap );
```

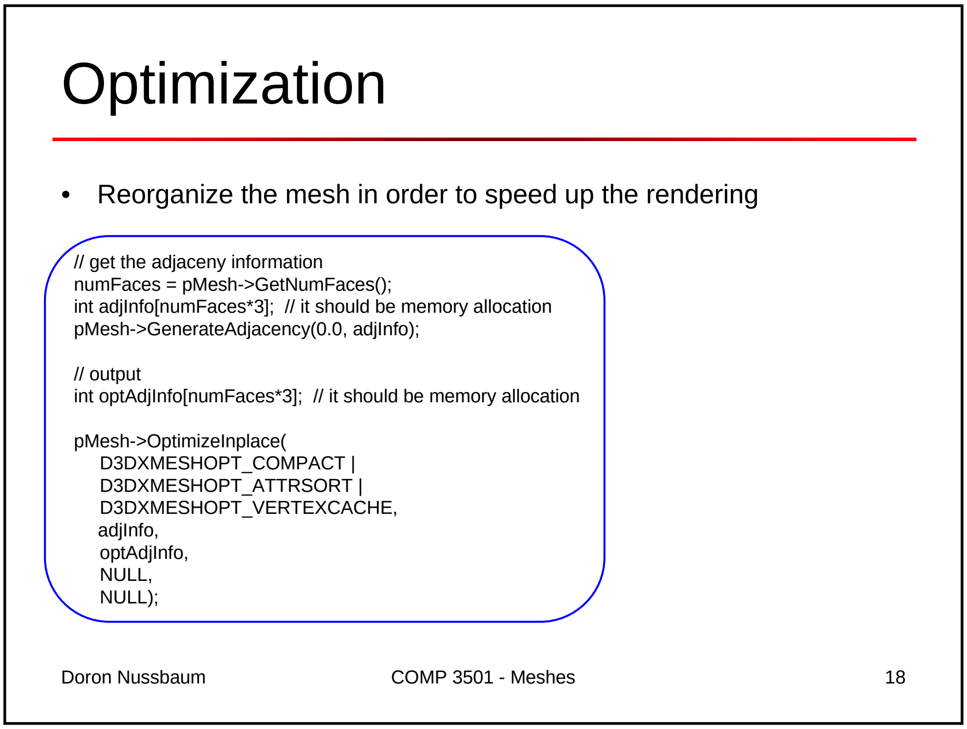
```
HRESULT Optimize(
    [in] DWORD Flags,
    [in] const DWORD *pAdjacencyIn,
    [out] DWORD *pAdjacencyOut,
    [out] DWORD *pFaceRemap,
    [out] LPD3DXBUFFER *ppVertexRemap,
    D3DXMESH **optMesh);
```

Optimization Instruction flags
D3DXMESHOPT_COMPACT
D3DXMESHOPT_ATTRSORT
D3DXMESHOPT_VERTEXCACHE
D3DXMESHOPT_STRIPREORDE

Faces adjacency list
Size = numFaces *3
(No adjacency = 0xffffffff)

Mapping of the faces. Elements holds the original index of the faces (buffer size = numFaces)
It can be NULL

Mapping of the vertices. Elements holds the original index of the vertices (buffer size = numVtx)
It can be NULL



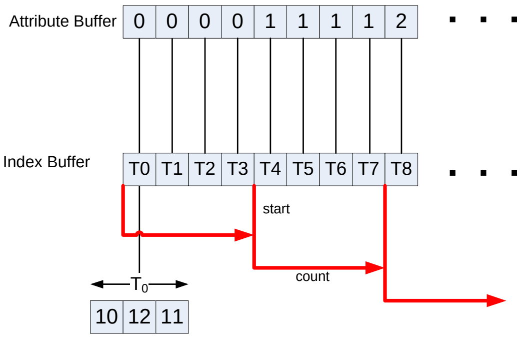
Optimization

- Reorganize the mesh in order to speed up the rendering

```
// get the adjaceny information
numFaces = pMesh->GetNumFaces();
int adjInfo[numFaces*3];  // it should be memory allocation
pMesh->GenerateAdjacency(0.0, adjInfo);

// output
int optAdjInfo[numFaces*3];  // it should be memory allocation

pMesh->OptimizeInplace(
    D3DXMESHOPT_COMPACT |
    D3DXMESHOPT_ATTRSORT |
    D3DXMESHOPT_VERTEXCACHE,
    adjInfo,
    optAdjInfo,
    NULL,
    NULL);
```

Attribute Table

- Optimization under the D3DXMESHOPT_ATTSORT groups the geometry together by attribute



Attribute Table

```
typedef struct D3DXATTRIBUTERANGE {
    DWORD AttribId;
    DWORD FaceStart;
    DWORD FaceCount;
    DWORD VertexStart;
    DWORD VertexCount;
} D3DXATTRIBUTERANGE, *LPD3DXATTRIBUTERANGE;
```

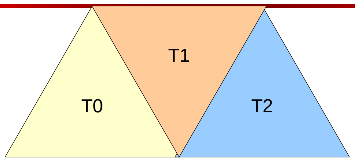
```
HRESULT GetAttributeTable(
    [out] D3DXATTRIBUTERANGE *pAttribTable,
    [out] DWORD cAttribTableSize );

HRESULT SetAttributeTable(
    [in] D3DXATTRIBUTERANGE *pAttribTable,
    [in] DWORD cAttribTableSize );
```

```
int numSubsets = 0;
pMesh->GetAttributeTable(NULL, &numSubsets);

// allocate memory
D3DXATTRIBUTERANGE attTable = new D3DXATTRIBUTERANGE[numSubsets];

pMesh->GetAttributeTable(attTable, &numSubsets);
```

Adjacency information

- Purpose
  - – Assists in optimizing meshes
  - – Assists in navigating the mesh
- Adjacency data
  - – 1-1 correspondence with triangles
  - – Each triangle has three value
    - Buffer size if numFaces*3
  - – Information for triangle *j* is in
    - 3*j
    - 3*j+1
    - 3*j+2

Adjacency



Adjacency array	-1	1	-1	0	-1	2	1	-1	-1

Index Buffer	V0	V1	V2	V3	V4	V5	V6	V7	V8

```
int adjData[pMesh->numFaces() * 3];

pMesh->GenerateAdjacency(0.01f, adjData);
```

Level of accuracy

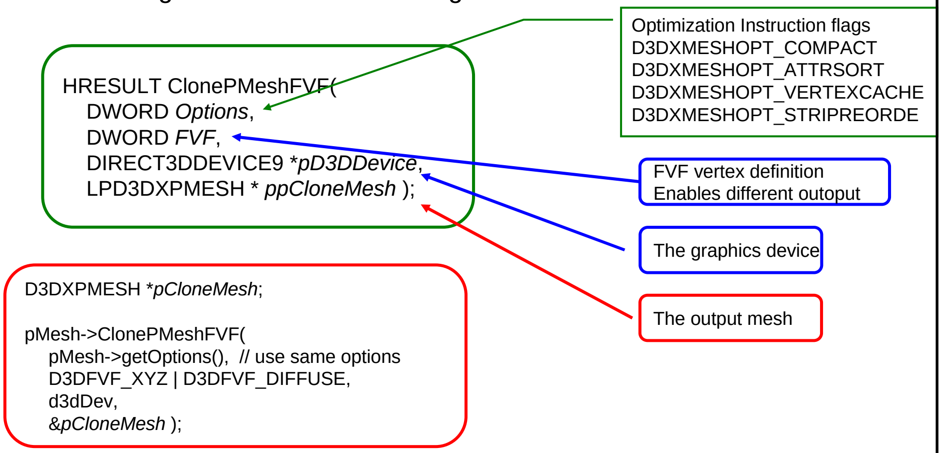
Data buffer

Creating a Mesh

- Three options
  - – Create from scratch – D3DXCreateMeshFVF
  - – Cloning – CloneMeshFVF
  - – Reading from a file

Cloning

- Creating a mesh from an existing mesh



```
HRESULT ClonePMeshFVF(
    DWORD Options,
    DWORD FVF,
    DIRECT3DDEVICE9 *pD3DDevice,
    LPD3DXPMESH * ppCloneMesh );
```

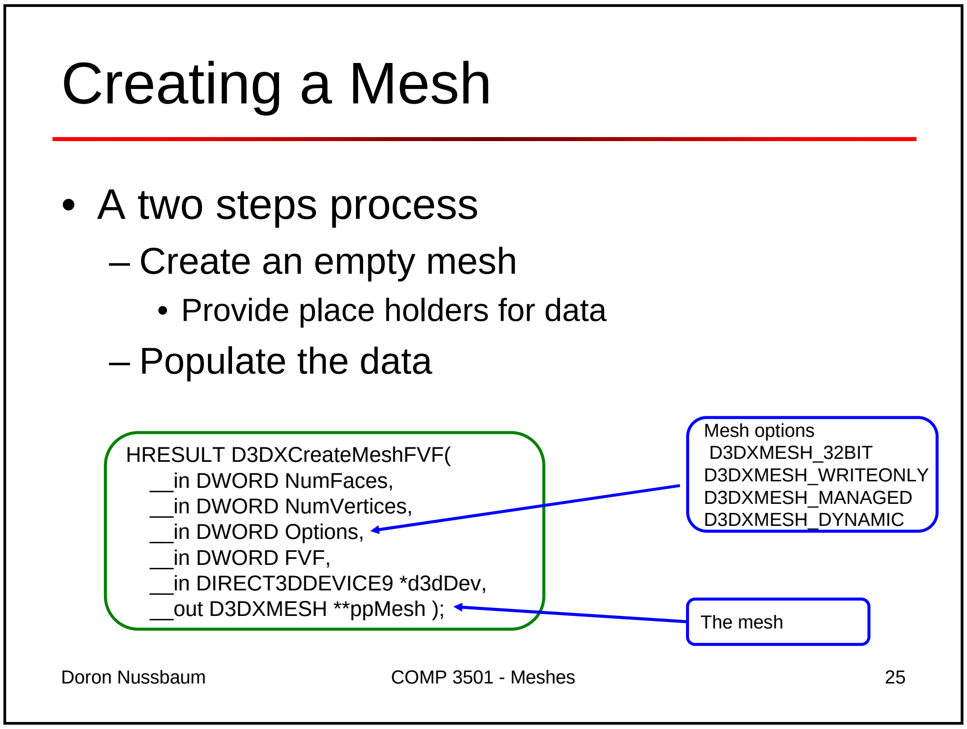
Optimization Instruction flags
D3DXMESHOPT_COMPACT
D3DXMESHOPT_ATTRSORT
D3DXMESHOPT_VERTEXCACHE
D3DXMESHOPT_STRIPREORDE

FVF vertex definition
Enables different outoput

The graphics device

The output mesh

```
D3DXPMESH *pCloneMesh;

pMesh->ClonePMeshFVF(
    pMesh->getOptions(),  // use same options
    D3DFVF_XYZ | D3DFVF_DIFFUSE,
    d3dDev,
    &pCloneMesh );
```



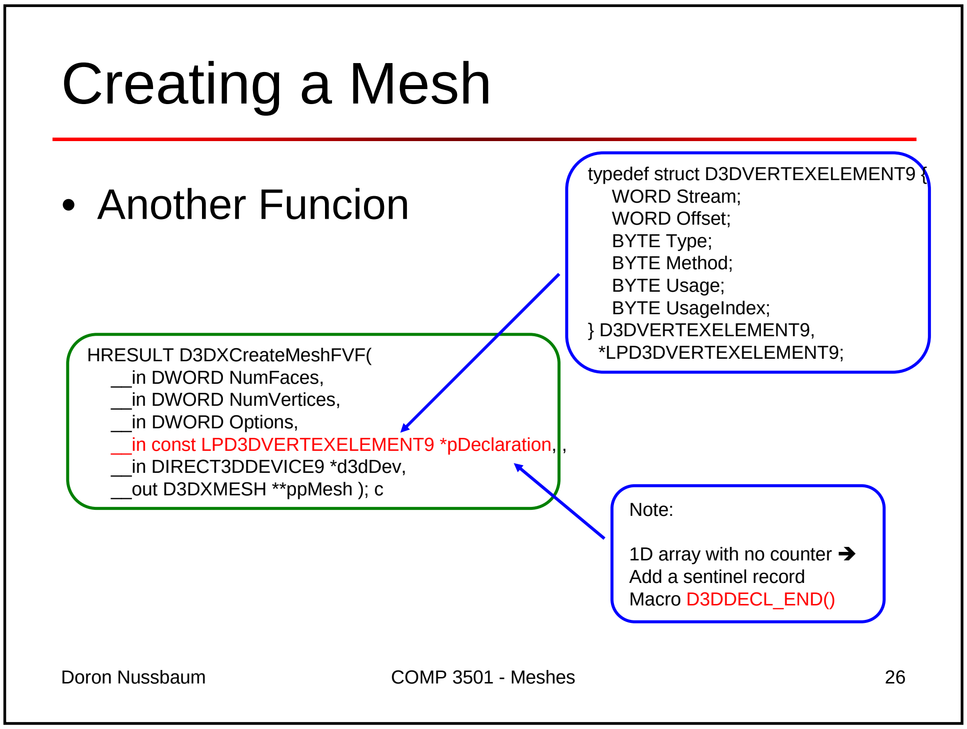
Creating a Mesh

- A two steps process
  - Create an empty mesh
    - Provide place holders for data
  - Populate the data

```
HRESULT D3DXCreateMeshFVF(
   __in DWORD NumFaces,
   __in DWORD NumVertices,
   __in DWORD Options,
   __in DWORD FVF,
   __in DIRECT3DDEVICE9 *d3dDev,
   __out D3DXMESH **ppMesh );
```

Mesh options
D3DXMESH_32BIT
D3DXMESH_WRITEONLY
D3DXMESH_MANAGED
D3DXMESH_DYNAMIC

The mesh



Creating a Mesh

- Another Funcion

```
typedef struct D3DVERTEXELEMENT9 {
   WORD Stream;
   WORD Offset;
   BYTE Type;
   BYTE Method;
   BYTE Usage;
   BYTE UsageIndex;
} D3DVERTEXELEMENT9,
 *LPD3DVERTEXELEMENT9;
```

```
HRESULT D3DXCreateMeshFVF(
   __in DWORD NumFaces,
   __in DWORD NumVertices,
   __in DWORD Options,
   __in const LPD3DVERTEXELEMENT9 *pDeclaration, ,
   __in DIRECT3DDEVICE9 *d3dDev,
   __out D3DXMESH **ppMesh ); c
```

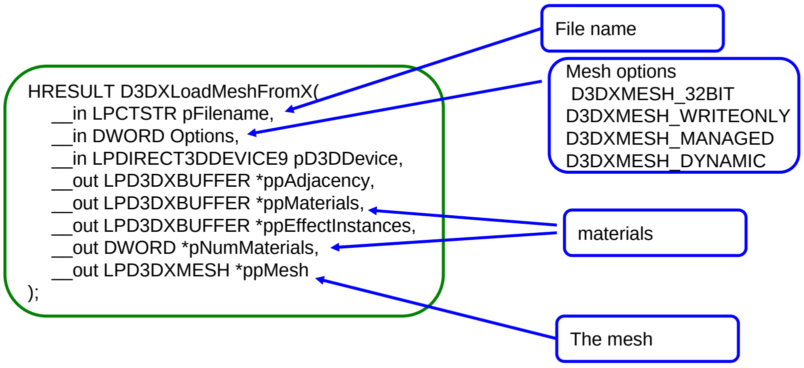
Note:

1D array with no counter ➔
Add a sentinel record
Macro D3DDECL_END()

```
D3DXMESH *pMesh = NULL;
int numSubsets = 3;
IDirect3DTexture9 *Textures[3] = {NULL, NULL, NULL);

Rc = D3DXCreateMeshFVF(8,12, D3DXMANAGED, MY_VTX_FVF, d3DDev, &pMesh);

Struct MY_VERTEX *vtx;
pMesh->LockVertexBuffer(0, (void **) &vtx);
Vtx[0] = …

pMesh->UnlockVertexBuffer();

short *indexBuf;
pMesh->LockIndexBuffer(0, (void **) &indexBuf);
indexBuf[0] = …

pMesh->UnlockIndexBuffer();

// attribute buffer

// Optimize mesh
pMesh->OptimizeInPlace(D3DXMESHOPT_ATTRSORT, & adjBuffer, NULL, NULL, NULL);

// draw the mesh
```

Loading a mesh from X file

- XFile contains objects with all the rquired information for drawing
  - Geometry
  - Topology
  - Cladding
- Generated by modeling tools
  - Maya
  - 3D Max
  - LightWave 3D



```
HRESULT D3DXLoadMeshFromX(
    __in LPCTSTR pFilename,
    __in DWORD Options,
    __in LPDIRECT3DDEVICE9 pD3DDevice,
    __out LPD3DXBUFFER *ppAdjacency,
    __out LPD3DXBUFFER *ppMaterials,
    __out LPD3DXBUFFER *ppEffectInstances,
    __out DWORD *pNumMaterials,
    __out LPD3DXMESH *ppMesh
);
```

File name

Mesh options
D3DXMESH_32BIT
D3DXMESH_WRITEONLY
D3DXMESH_MANAGED
D3DXMESH_DYNAMIC

materials

The mesh

```
typedef struct D3DXMATERIAL {
    D3DMATERIAL9 MatD3D;
    LPSTR pTextureFilename;
} D3DXMATERIAL, *LPD3DXMATERIAL;
```

```
typedef struct D3DMATERIAL9 {
    D3DCOLORVALUE Diffuse;
    D3DCOLORVALUE Ambient;
    D3DCOLORVALUE Specular;
    D3DCOLORVALUE Emissive;
    float Power;
} D3DMATERIAL9, *LPD3DMATERIAL9;
```

Diffuse - Value specifying the diffuse color of the material.

Ambient - Value specifying the ambient color of the material.

Specular - Value specifying the specular color of the material.

Emissive - Value specifying the emissive color of the material.

Power - value specifying the sharpness of specular highlights.

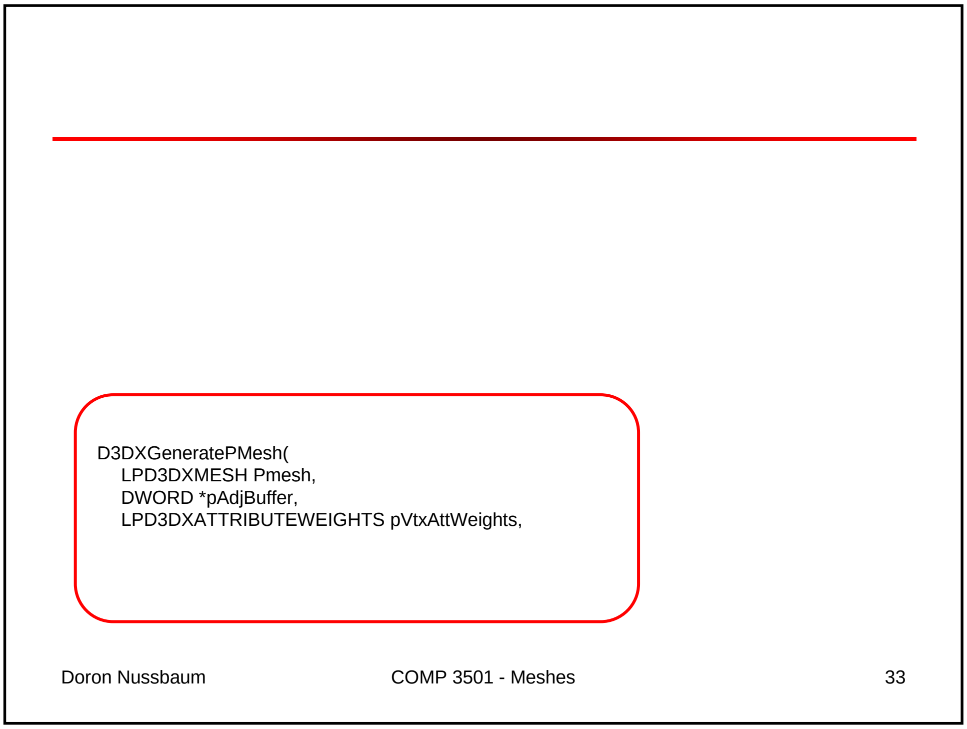
]

Progressive Meshes

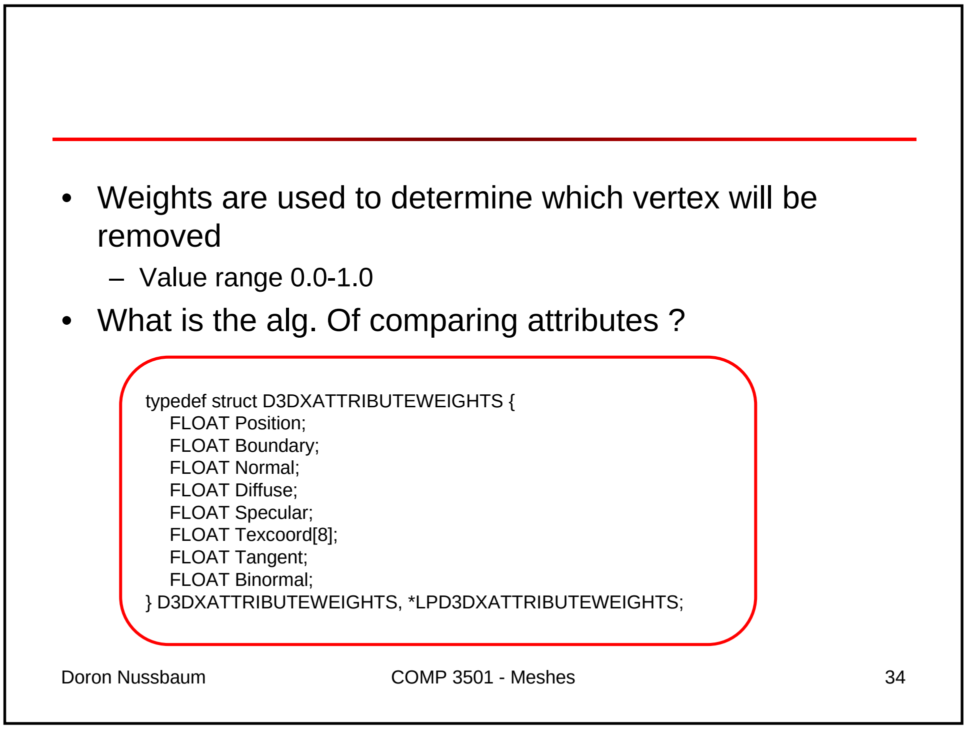
- Uses edge collapse
  - by combining two vertices into a single vertex
  - One or two faces are removed
- Direct3D enables
  - Creation of a progressive mesh
  - Manipulation of a progressive mesh by the programmer
  - Level of progressive mesh is to be defined at runtime

```
HRESULT D3DXGeneratePMesh( LPD3DXMESH pMesh,
    CONST DWORD * pAdjacency,
    CONST D3DXATTRIBUTEWEIGHTS * pVertexAttributeWeights,
    CONST FLOAT * pVertexWeights,
    DWORD MinValue,
    DWORD Options,
    LPD3DXPMESH * ppPMesh );
```

pMesh - the source mesh.
pAdjacency – adjacency of the source mesh
pVertexAttributeWeights – array containing the weight for each vertex component (can be NULL)
pVertexWeights – array containing the vertex weights (can be NULL).
MinValue – the minimum number of requested vertices or faces for simplifciation.
Options – specify how to simplify (D3DXMESHSIMP_VERTEX, D3DXMESHSIMP_FACE)
ppPMesh – output mesh



```
D3DXGeneratePMesh(
    LPD3DXMESH Pmesh,
    DWORD *pAdjBuffer,
    LPD3DXATTRIBUTEWEIGHTS pVtxAttWeights,
```



- Weights are used to determine which vertex will be removed
  - Value range 0.0-1.0
- What is the alg. Of comparing attributes ?

```
typedef struct D3DXATTRIBUTEWEIGHTS {
    FLOAT Position;
    FLOAT Boundary;
    FLOAT Normal;
    FLOAT Diffuse;
    FLOAT Specular;
    FLOAT Texcoord[8];
    FLOAT Tangent;
    FLOAT Binormal;
} D3DXATTRIBUTEWEIGHTS, *LPD3DXATTRIBUTEWEIGHTS;
```

ID3DXPMesh Functions

- GetMaxFaces()
  - Returns the maximum number of faces
- GetMaxVertices()
  - Returns the maximum number of vertices
- GetMinFaces()
  - Returns the minimum number of faces
- GetMinVertices()
  - Returns the minimum number of vertices
- SetNumFaces(int numFaces)
  - Sets the number of faces
  - pPMesh->setNumFaces(40);
- SetNumVertices(int numVtx)
  - Sets the number of vertices
  - pPMesh->setNumVertices(40);

Bounding Volumes

- Direct3D can compute simple bounding volume
  - Sphere
  - Box
- The function accept an array of vertices and compute the bounding volume

```
HRESULT D3DXComputeBoundingBox(
    D3DXVECTOR3 *pFirstPosition,
    DWORD NumVertices,
    DWORD dwStride,
    D3DXVECTOR3 *pMin,
    D3DXVECTOR3 *pMax );
```

```
HRESULT D3DXComputeBoundingBox(
    D3DXVECTOR3 *pFirstPosition,
    DWORD NumVertices,
    DWORD dwStride,
    D3DXVECTOR3 *pCentre,
    FLOAT *pRadius );
```

```
void *vtx = NULL;
float centre;
flat radius;

pMesh->LockVertexBuffer(0, (void **) &vtx);

rc = D3DXComputeBoundingSphere(
    (D3DXVECTOR3 *) vtx,
    pMesh->GetNumVertices(),
    D3DXGetFVFVertexSize(pMesh->GetFVF()),
    &centre,
    &radius);

pMesh->UnlockVertexBuffer();
```