Driver-Inter Ltd.

XMesh User Guide

Владимир Беляев, Наталья Смирнова

02.09.2009

X-Mesh или, если быть точнее, **ID3DXMesh** служит для хранения и рендеринга геометрических данных. Фактически, **ID3DXMesh** содержит:

- вершинный и индексный буфер,
- набор аттрибутов, задающих материалы (D3DMATERIAL9) и имена текстур для треугольников.

Таким образом, **ID3DXMesh** хранит и может рендерить геометрию, на которой лежат несколько материалов.

Источником данных в формате x-mesh служит плагин для 3DSMAX и Maya (очень мощные тулы для создания геометрии и многого другого). Хотя, так как формат текстовый, его можно легко сгенерировать другим способом.

Замечание: данный документ не рассматривает загрузку и использование текстур. В тексте есть только пометки в тех местах кода, в которых при необходимости можно добавить использование текстур.

А класс у нас будет примерно такой:

```
class MyMesh
{
private:
  ID3DXMesh    *m_pMesh;
  D3DMATERIAL9 *m_pMaterials;
  UINT          m_numMaterials;
public:
  void load(char *pFileName, IDirect3DDevice9 *pDev);
  void render(IDirect3DDevice9 *pDev);
};
```

Загрузка

Прочитать **ID3DXMesh** из файла можно с помощью ряда функций D3DXLoadMesh*(). В настоящем руководстве мы рассмотрим только одну – D3DXLoadMeshFromX(). Она принимает следующие параметры:

- LPCTSTR *pFilename* – имя файла,
- DWORD *Options* – комбинация флагов, которые управляют созданием меши. Для лабораторных работ можно задать 0 и не мучаться,
- LPDIRECT3DDEVICE9 *pD3DDevice* – здесь все ясно,
- LPD3DXBUFFER **ppAdjacency* – задание этого параметра в указатель на указатель на ID3DXBuffer позволит получить информацию о связности треугольной сетки (информацию о соседях). Для лабораторных работ это абсолютно не нужно и можно смело задать NULL,
- LPD3DXBUFFER **ppMaterials* – а вот сюда уже нужно задать указатель на указатель на ID3DXBuffer. Это необходимо, чтобы получить набор структур D3DXMATERIAL, в которых лежит D3DMATERIAL9 и имя текстуры. Указатель на этот буфер следует куда-нибудь сохранить, чтобы потом использовать эти данные для освещения (а в последствии и для текстурирования),

- LPD3DXBUFFER **ppEffectInstances* – задаем сюда NULL,
- DWORD **pNumMaterials* – получаем количество материалов,
- LPD3DXMESH **ppMesh* – получаем сам указатель на **ID3DXMesh**.

```
HRESULT hr = D3DXLoadMeshFromX(pFileName, 0, pDevice,
         NULL, &m_pMaterials, NULL, &m_numMaterials, &m_pMesh);
```

После создания меши, следует подготовить ее к использованию: а именно отсортировать треугольники по аттрибутам. Формат меши не гарантирует, что все треугольники с одинаковыми аттрибутами лежат рядом друг с другом, и, если ничего не предпринимать, то придется рендерить мешь по треугольникам, так как даже для двух материалов возможен случай, когда для каждого нового треугольника будет необходимо вызывать **SetRenderState()** или **SetMaterial()**.

Чтобы произвести необходимую сортировку следует вызывать метод

```
m_pMesh->OptimizeInplace(D3DXMESHOPT_COMPACT | D3DXMESHOPT_ATTRSORT,
                         NULL, NULL, NULL, NULL).
```

Таким образом загрузка будет выглядеть примерно так:

```
void MyMesh::load(char *pFileName, IDirect3DDevice9 *pDev)
{
   // грузим мешь из файла
   ID3DXBuffer *pMat = NULL;
   HRESULT hr = D3DXLoadMeshFromX(pFileName, 0, pDev,
         NULL, &pMat, NULL, &m_numMaterials, &m_pMesh);

   // далее сразу разбираем полученные материалы для удобства рендеринга

   // получаем массив материалов из буфера
   D3DXMATERIAL* pD3DXMaterials=(D3DXMATERIAL*)pMat->GetBufferPointer();

   // создаем массив более простых материалов
   m_pMaterials = new D3DMATERIAL9[m_numMaterials];

   // место для создания массива текстур

   // ну а теперь заполним созданные ранее массивы
   for (DWORD i = 0; i < m_numMaterials; ++i)
   {
         // сохраняем материалы
         m_pMaterials[i] = pD3DXMaterials[i].MatD3D;
         m_pMaterials[i].Ambient = m_pMaterials[i].Diffuse;

         // Здесь место для загрузки текстуры из файла по имени из
```

```
            // материала
      }


      // не забываем удалить буфер
      if (pMat)
            pMat->Release();


      // оптимизируем меш
      m_pMesh->OptimizeInplace(D3DXMESHOPT_COMPACT |
                  D3DXMESHOPT_ATTRSORT, NULL, NULL, NULL, NULL);
   }
```

Визуализация

После этого, вместо задания вершинного и индексного буферов, задания FVF флагов и вызова **DrawIndexedPrimitive()**. Достаточно вызвать **ID3DXMesh::DrawSubset(),** передавая в качестве параметров индекс материала.

В итоге получаем следующий цикл рендеринга:

```
   void MyMesh::render(IDirect3DDevice9 *pDev)
   {
        for (UINT i = 0; i < m_numMaterials; i++)
        {
          // Устанавливаем материал
          pDev->SetMaterial(&(m_pMaterials[i]));


          // Устанавливаем текстуру
          // Место для установки текстуры, после того, как мы узнаем, что
          // это такое


          // Рендерим часть меши
          pMesh->DrawSubset(i);
        }
   }
```

Ну и, конечно же, перед выходом из приложения не забываем сделать **Release()** меши и буферу, а так же освободить выделенную память.