Model DEMO 设计文档

熊新科

2005．2．27

Model demo 主要演示了模型的载入和渲染，本 demo 所使用的模型是由 ID Software 所定义的 MD2 格式，MD2 格式在《Quake II》游戏中所被广泛使用。

MD2 的文件由两部分组成，一部分是文件头（fileheader）；另一部分是文件数据（filedata）。文件头包含有很多重要的数据、如：文件数据的大小，文件格式的版本号等等。因此每个文件的文件头的大小都应该是一致的。这也就是为什么我们一般都用一个结构体来描述这些文件头。与之相反的是每个文件的文件数据都不同的。因为文件数据包含了模型的顶点（vertices）数，三角形（triangles）面数，纹理映射坐标（texture coordinate）等等。下图表示了 MD2 模型文件的大略格式。

因此我们可以使用一个结构体来表示 MD2 文件头的组织。如下：

```
//MD2 的文件头数据，这个数据结构的成员顺序是要严格定义不能调乱的
struct MD2_FileHeader
{
    int m_nIdentify;          //识别号
    int m_nVersion;           //版本号
    int m_nSkinWidth;         //模型皮肤纹理的宽度
    int m_nSkinHeight;        //模型皮肤纹理的高度
    int m_nFrameSize;         //每帧的大小，以字节为单位
    int m_nSkinNum;           //与模型相关联的皮肤数量
    int m_nVertexNum;         //模型的顶点数
    int m_nUVNum;             //纹理坐标的数量
    int m_nTriangleNum;       //模型中三角形面片的数量
    int m_nOpenGLCmdNum;      //OpenGL 的命令数目
    int m_nFrameNum;          //帧数目
    int m_nSkinsOffset;       //皮肤数据在文件中的偏移地址
```

```
    int m_nTexCoordOffset;      //纹理映射坐标数据在文件中的偏移量
    int m_nTrianglesOffset;     //三角形面片数据在文件中的偏移量
    int m_nFrameOffset;         //帧数据在文件中的偏移量
    int m_nOpenGLCmdOffset;     //OpenGL 命令在文件中的偏移量
    int m_nEndOffset;           //文件结束位置在文件中的偏移量
};
```

下面是结构体中的各成员变量的具体含义：

m_nFrameSize

指定每一帧（frame）的大小。所谓的“帧”是“动画”（animation）的基本元素。这个帧和动画片中的每一张静态的图片的含义是类似。当我们在程序中以一定的时间间隔连续而且循环播放一系列静态的帧的时候，就能形成动画。所以，在 MD2 文件中，一个“帧”就存储了模型在一个特殊位置（或者说动作）时的顶点和三角形面片的信息。典型的 MD2 文件最多可由 199 帧。共 21 个动画组成。

一个帧包含了组成这帧的三角形面片中的所有顶点，注意每一帧的三角形面片数都是相同的。而且我们要利用三角形面片数来给这些顶点分配存储空间。

m_nVertexNum

指定了该模型所拥有的顶点总数。它是指各帧所拥有的顶点总数之和。

m_nTriangleNum

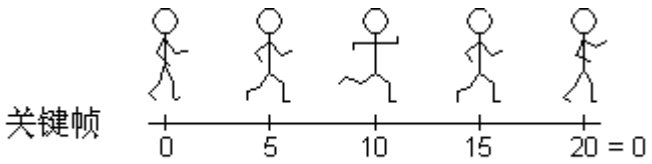
指定了该模型所有的三角形面片数

m_nOpenGLCmdNum

指定了该模型所有的 OpenGL 命令数目。

m_nFrameNum

指定了该模型所有的帧的数量。事实上，MD2 中的帧都是“关键帧”(keyframe)。所谓的关键帧就是指按照一定的时间间隔所取的离散的帧。因为如果不是用关键帧的话，每个模型甚至可能会需要 200 到 300 帧才能组成一个平滑的动画。这样一来模型文件的体积将极速膨胀。所以我们只是需要保存关键帧，中间的过渡帧我们可以利用线性插值（linear interpolation）的方法。在需要渲染的时候计算出来。下面的图示就很好地表示了关键帧的概念：



整个平滑的动画需要 20 帧，MD2 模型只是需要存储编号为 0，5，10，15，20 的这些帧，剩下的帧在需要渲染的时候可以实时计算。每一个模型都是由“帧数目”×“顶点数目”个顶点所组成的。所以我们要定义一个结构体来存储顶点数据。如下：

```
struct MD2_TriVertex
{
```

```
    unsigned char m_X , m_Y , m_Z;
    unsigned char m_LightNormalIndex;
};
```

使用 unsigned char 类型来存储顶点的 X,Y,Z 坐标的话，那么顶点的范围值就只能从 0 到 255,但是相对于使用 float 类型来存储数据，我们能够有效地压缩数据，使得文件的体积大为减小。

接着我们需要定义一个结构体来存储顶点的纹理映射坐标。如下：

```
struct MD2_UVTexCoord
{
    short int m_nU , m_nV;
};
```

如顶点数据一样，纹理坐标也是使用了压缩的方法，在 MD2 文件中，纹理坐标使用 short int 类型的数据，但是为了使用这些坐标，我们首先要把从文件中读取出的数据做一下转换才能使用。这是因为，纹理坐标是从 0 到 1 的浮点实数，而如果直接使用 shortint 类型的数的话，用坐标除以纹理的高宽是只能得到 0 或者是 1 的。所以转换的伪代码如下：

```
实际纹理 U 坐标 =  (float)m_nU / MD2_FileHeader.m_nSkinWidth;
实际纹理 V 坐标 =  (float)m_nV / MD2_FileHeader.m_nSkinHeight;
```

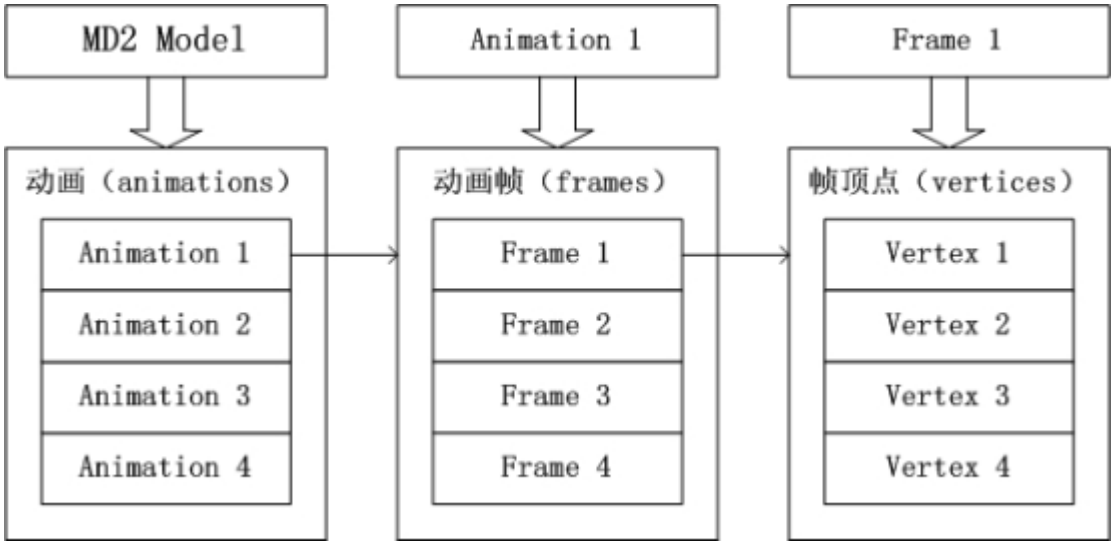
定义结构体如下，可以存储 MD2 模型中的“帧”的相关数据。

```
struct MD2_AliasFrame
{
    float m_fScaleX;
    float m_fScaleY;
    float m_fScaleZ;
    float m_fTranslateX;
    float m_fTranslateY;
    float m_fTranslateZ;
    char m_szName[16];
    MD2_TriVertex m_TriVtx[1];
};
```

这样，每一帧的数据都能以这种数据结构方式存储。所以一个典型的模型将会有 199 个帧对象。当解压了每一个顶点数据之后，将会使用结构体的放缩系数 m_fScaleX；m_fScaleY；m_fScaleZ（实际上就是放缩矩阵）和平移系数 m_fTranslateX；m_fTranslateY；m_fTranslateZ（实质上是平移矩阵）把模型放缩和平移到合适的大小和位置。

m_szName 存储了该帧的名字。最后，变量 m_TriVtx[1]是本帧的第一个顶点位置，其他的顶点将跟在这个顶点之后。所以每个动画都含有 N 帧，每帧含有个 MD2_FileHeader.m_nVertexNum 个顶点。

综上。动画，帧和顶点的关系如下图：



每个顶点还需要和它所映射的纹理坐标关联起来。但是我们是采用一个三角形面片的方式来存储的，方式如下：

```
struct MD2_Triangle
{
    short int m_nIndexA;
    short int m_nIndexB;
    short int m_nIndexC;
    short int m_nIndexA_UV;
    short int m_nIndexB_UV;
    short int m_nIndexC_UV;
};
```

但是要注意，这结构数据成员仅仅是指这些顶点或者是纹理坐标的索引，而并不是指顶点或纹理坐标本身。顶点数据和纹理数据必须存储在的数据中，或者是你在渲染模型时解压到程序员指定的数据区域中去。

有了上述的数据结构，就可以着手进行程序设计了。具体的设计方面可以看代码