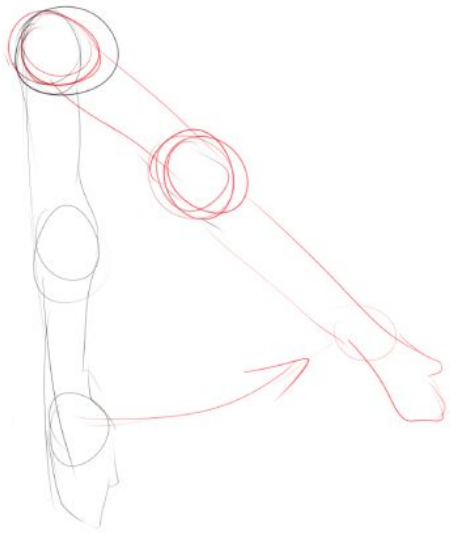
MS3D 模型骨骼动画

MS3D Model Skeleton Animation

骨骼动画介绍

当前有两种模型动画的方式：顶点动画和骨骼动画。顶点动画中，每帧动画其实就是模型特定姿态的一个“快照”。通过在帧之间插值的方法，引擎可以得到平滑的动画效果。在骨骼动画中，模型具有互相连接的“骨骼”组成的骨架结构，通过改变骨骼的朝向和位置来为模型生成动画。

骨骼动画比顶点动画要求更高的处理器性能，但同时它也具有更多的优点，骨骼动画可以更容易、更 快捷地创建。不同的骨骼动画可以被结合到一起——比如，模型可以转动头部、射击并且同时也在走路。一些引擎可以实时操纵单个骨骼，这样就可以和环境更加准确地进行交互——模型可以俯身并向某个方向观察或射击，或者从地上的某个地方捡起一个东西。多数引擎支持顶点动画，但不是所有的引擎都支持骨骼动画。



一些引擎包含面部动画系统，这种系统使用通过音位（phoneme）和情绪修改面部骨骼集合来表达面部表情和嘴部动作。

以上是百度知道对骨骼动画的解释，所谓骨骼动画就是一种关节的动画，关节直接有某种动作的变化，比如手臂和手的就是一种骨骼动画，手臂的转动会带动手的移动和转动，这个关系就是父节点运动对子节点的影响。至于数学上的关系后面阐述。

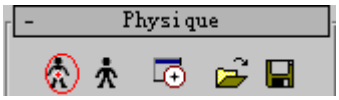
MAX 里的蒙皮和骨骼动画关键帧

接下来**简单**介绍如何在 max 里蒙皮步骤。

1、建立模型后，在对应的位置建立骨骼，建立骨骼有很多种方法，这里用的 MAX9.0 有一个叫做 Biped 的工具，快速建立人体骨骼。(注意一般来如你建立的模型的姿势对后面蒙皮很重要所以一般建模的人体模型都是四肢张开的这样方便后面的处理)。

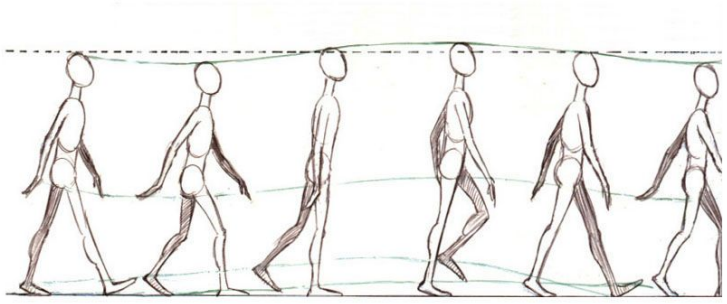
2、接下来你需要做的是将骨骼的关节移动到你模型的关节上。点击任意一骨骼，单击左图中的小人，进入体型模式，这一步调整需要精确，将骨骼摆放到模型的最佳位置。

3、调整完后你的模型的姿势就是这个模型的标准姿势，可以关掉体型模式调整模式。



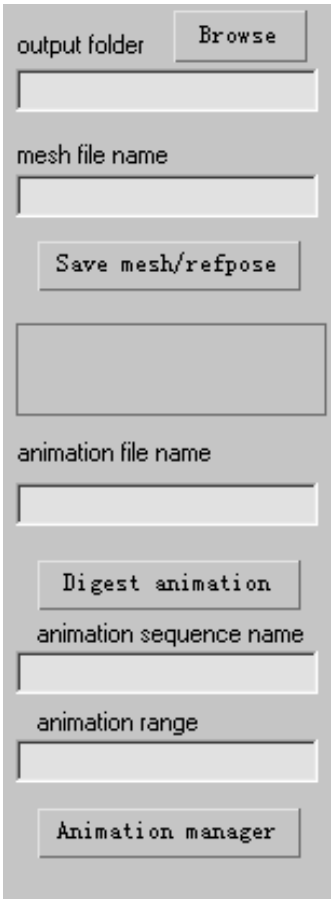
4、接下来要蒙皮，单击模型，进入网格编辑，添加修改器列表的 Physigue，然后点击附加到节点此时 MAX 会让你选择一个骨骼，你需要选择这个骨骼的跟节点，然后单击初始化，这个时候 max 为你的模型自动绑定了节点。

5、单击网格修改器列表里的 Physigue，点中封套，此时你可以修改封套，以正确的将顶点绑定到节点。这是个需要耐心的步骤，如果在这之前你已经完成了模型分 UV 的步骤，绘制贴图纹理的步骤，那修改封套对你来说不算什么。完成这步模型的处理已基本完成。



6、设置动画关键帧，这一步根据你模型的需要而定，你可以为模型添加一系列动作这样你就可以在程序里播放对应的动画。添加动画的步骤比较复杂，你可以寻找 max 的动作库导入动作，也可以自己设置模型几个关键的 pose，并且调整动画曲线就可以做出骨骼动画，至于器原理可以参考动画原理，其中有介绍人走路的动画，如果你能自己独立完成一个人走路的动画序列，那么相信你也能完胜其他动画了。

如何从 MAX 导入带骨骼动画的模型到 MilkShape 3D



1、剩下最后一步导出到 ms3d 格式，可惜 max 并没有提供这个插件，如果你不会写 max 的输出插件，那么你可以像我一样，通过 Unreal Tech ActorX 插件间接的导入网格和骨骼到 ms3d，这个插件可以在官方网 http://udn.epicgames.com/Two/ActorX.html 下载来。安装好插件后，你可以通过左图的设置输出.Psk 和.Psa。其中 psk 是网格，psa 是骨骼动画。在输入网格的时候要注意，为了减少 ms3d 里的麻烦，你必须将体型模式下的骨骼导出，前面说过那个是标准姿势甚至在后面程序里也很关键。位骨骼动画设置名字和关键帧范围后，可以安吉 Animation manager 后，右移你的动作 save。这个时候 MAX 的任务也就完成了。

2、打开 MilkShape 3D,先导入 psa，在导入 psk，这个时候，如果你前面没啥差错骨骼和模型是严格的按照你的标准姿势，单击后你可以拖动滚动条块观察你刚才导入的动画，然后你需要设置一下贴图，这里 MilkShape 3D 对贴图名称有限定，不过貌似新版本没，名字尽量用英文。之后如果模型动作贴图都没有错误，保存被 ms3d 格式就可以了。此时导出的就是一个能够在游戏程序里使用的模型了。

MilkShape 3D 模型格式(ms3d)

任何一个模型文件想要使用都要了解他的文件结构，建立对应的数据类型通过文件指针读取保存到内从中。一下的格式是从 ms3d 官方开发包里提取出来的。
首先从这段代码可以了解到 ms3d 支持的顶点数和面数，六万面对一个 PC 游戏来说足够了，当然人家次时代和这个不是一个数量级的。

```
#define MAX_VERTICES    65534
#define MAX_TRIANGLES   65534
#define MAX_GROUPS      255
#define MAX_MATERIALS   128
#define MAX_JOINTS      128
```

接下来是 ms3d 文件里得东西。由于内容很多所以精简出来，就是一个 ms3d 模型由几个网格组和骨骼构成。

1. 每个网格保存了里面所以三角形（TRIANGLES）的索引，而每个 TRIANGLES 保存了三个顶点的索引，每个顶点保存了三个 float 型，也就是点的 XYZ 坐标。其中每个顶点指定了一个绑定的节点，当然也可以指定多个节点并附上权值来规定节点对顶点的影响程度。
2. 每个骨骼都存储了该骨骼对应于父节点的相对变换包括 Position 和 Rotation 变换。根节点的的坐标是也是绝对坐标。

由此可以设计出读取 ms3d 格式的函数。你也可以用别人现成的不过一定要选择一个适合的，应为以后的编程都是基于这个读取展开的，如果其类写的不好，会增加不必要的麻烦。

参考：

```
*************** 顶点 ****************
typedef struct
{
    byte    flags;
    float   vertex[3];
    char    boneId;
    byte    referenceCount;
} ms3d_vertex_t;
```

```
***************三角形****************
typedef struct
{
    word    flags;
    word    vertexIndices[3];
    float   vertexNormals[3][3];
    float   s[3];
    float   t[3];
    byte    smoothingGroup;
```

```
************多边形组*****************
typedef struct
{
    byte            flags;
    char            name[32];
    word            numtriangles;
    word triangleIndices[numtriangles];
    char            materialIndex;
} ms3d_group_t;
***********材质和贴图***************
typedef struct
{
    char            name[32];
    float           ambient[4];
    float           diffuse[4];
    float           specular[4];
    float           emissive[4];
    float           shininess;
```

```
    byte    groupIndex;
} ms3d_triangle_t;
************关键帧******************
typedef struct
{
    float           time;
    float           rotation[3];
} ms3d_keyframe_rot_t;
typedef struct
{
    float           time;
    float           position[3];
} ms3d_keyframe_pos_t;
```

```
    float           transparency;
    char            mode;
    char            texture[128];
    char            alphamap[128];
} ms3d_material_t;
**************关节***************
typedef struct
{
    byte            flags;
    char            name[32];
    char            parentName[32];
    float           rotation[3];
    float           position[3];
    word            numKeyFramesRot;
    word            numKeyFramesTrans;
    ms3d_keyframe_rot_t
keyFramesRot[numKeyFramesRot];
ms3d_keyframe_pos_t
keyFramesTrans[numKeyFramesTrans];
} ms3d_joint_t;
```

骨骼动画的数学准备

这才是关键假设以上工作都完整无误的完成，那么接下来就开始最烦的地方了。我基于Opengl 的 GLFW 写的代码，其中窗口的实现的不做介绍，只对关键代码解释。

对于理解骨骼动画我们需要一些线性代数的知识，并且理清骨骼动画的实现方式。

概念一：变换矩阵

所谓变换矩阵就是对坐标里面某个点进行平移、旋转缩放之类的变换。这里我们只涉及平移和旋转，只要有线性代数的这很容易理解。这里给出比较傻瓜的理解。

$$\mathbf{R}=\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{T}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & \Delta z & 1 \end{bmatrix}$$

R 矩阵是旋转矩阵，T 矩阵是移动矩阵.向量 v 先旋转再平移，新的向量 v’计算如 v’ = vRT;其中 V 是一个 vec4型的变量（X,Y,Z,1）;其实经过观察我们可以吧 RT 矩阵合起来表示成一个矩阵 M 他就是这个点的最终的坐标变换，如图：

$$\mathbf{M}=\mathbf{RT}=\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & \Delta z & 1 \end{bmatrix}=\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ \Delta x & \Delta y & \Delta z & 1 \end{bmatrix}$$

这个矩阵对后面的计算格外重要。

概念二：坐标变换

首先假设有一点 V 在绝对坐标系的（0，0，0，1），那么将 V 乘一个变换矩阵 M 则可以得到一个新的 V1 点，以它做原点设置一坐标系称为相对坐标系，那么我们可以将一个点的坐标通过矩阵 M 相互转换，比如已知某点 V2 对于 V1 点的相对坐标，那么只需要右乘一个 M 就可以将它变为绝对坐标，反之如果知道 V2 的绝对坐标只需要右乘一个变换矩阵 M 的逆矩阵即可求的 V2 的绝对坐标。

拥有以上的两个概念后我们来分析骨骼动画，当然这两个概念我介绍比较笼统，如果能参阅相关的线性代数书更好。

骨骼动画的实现

回到骨骼中，前面我们建立的模型的标准姿势，这个姿势上每个点都是在绝对坐标系上的坐标（不考虑之前 Opengl 的变换），也就是我们根据每个顶点的坐标可以直接渲染出模型的标准姿势。

但是骨骼不一样，骨骼在 ms3d 格式离得存储方式是按照绝对变化存储的，这样可以表达出骨骼建的关系。

举个例子：有 2 个人以用样的速度走楼梯。

一开始你问：“我知道第一个人站在楼梯的第二个台阶上，你能告诉我第 2 个人站在那里？”

“可以，那个人站在第四个台阶上”

但是我这样说的后果很烦，如果你每次都问我我每次回答的台阶都不一样，有什么方法告诉你一次就能省略以后的回答呢？

没错我可以说：“他站在比第一个人高 2 层的台阶上”

这样一来你只需将上一个人的台阶数加 2 就可以了，这个思路就是骨骼在 ms3d 里的存储方式，他没有直接告诉你第一个关节和第二个关节的绝对坐标，而是告诉你子关节对于父关节的变换，因此只要知道第一个根节点的绝对坐标我就可以求出所有节点的位置。

因此第一个步骤是将骨骼搭建起来。

步骤一：初始化骨骼和绑定顶点

```
void SetupJointMatrices(CMS3DFile *mesh)
{
    先为每个节点设置三个矩阵，分别是
    绝对矩阵_absoluteJointMatrices，这个矩阵里面包含了改节点的变换
                                    和他的绝对位置
    相对变换矩阵_relativeJointMatrices ，这个就是前面例子里的“高两
                                        个台阶”的概念
    最终变换矩_finalJointMatrices ，这个的作用是将前面所有变化累积
                                    起来。

     mesh->_absoluteJointMatrices = new mat4[mesh->GetNumJoints()];
     mesh->_relativeJointMatrices = new mat4[mesh->GetNumJoints()];
     mesh->_finalJointMatrices = new mat4[mesh->GetNumJoints()];

   接着遍历每一个节点，这里 ms3d 里的第一个节点就是根节点
     for(int i = 0; i <mesh->GetNumJoints(); i++)
     {
       mat4 relativeMatrix; 设置一个暂时的变化矩阵
       mat4_identity( relativeMatrix);    单位化!! 这步要严谨
       ms3d_joint_t *jt;
       mesh->GetJointAt(i, &jt);
     下面的函数如前面概念一是将旋转矩阵 R 和平移矩阵 T 合为矩阵 M
       mat4_rotateX(relativeMatrix, jt->rotation[0]);
       mat4_rotateY(relativeMatrix, jt->rotation[1]);
       mat4_rotateZ(relativeMatrix, jt->rotation[2]);
```

```
            mat4_translate(relativeMatrix,jt->position[0], jt->position[1],
jt->position[2]);
            if(strcmp(jt->parentName,"") == 0 ) 判断是不是根节点
            {
    如果是根关节那么它的绝对矩阵就是它的相对变换矩阵
    mat4_copy(mesh->_absoluteJointMatrices[i],relativeMatrix);
          }
          else
          {
              否者的如果是子关节，那么将上一个节点的绝对矩阵乘上这个节点的变幻矩
阵就可以得到这个子节点的绝对矩阵了.这就好比上面例子告诉我的得知前一个人的台阶根
据“高 2 个台阶”的规律得到这个认得台阶。

              int index =  mesh->FindJointByName(jt->parentName);
              mat4 temp;

mat4_multiply( mesh->_absoluteJointMatrices[index],relativeMatrix,temp);
              mat4_copy(mesh->_absoluteJointMatrices[i],temp);
          }
    将相对相对矩阵保存起来方便运算以后的总变换
          mat4_copy(mesh->_relativeJointMatrices[i],relativeMatrix);
        >_absoluteJointMatrices[i]);

        }

    }
```

通过这步骨骼被搭建起来了，所有节点绝对矩阵_absoluteJointMatrices 里都可以看到节点的准确位置。这个骨骼被准确的摆放成了你模型里的标准姿势，而以后的变换都是根据这个标准变幻的，注意你不需要每次都改变节点的绝对，这样做的后果是模型骨骼变化古怪，顶点也是一样的，每次顶点都是通过标准姿势变化的带的。

下面代码是通过计算标准位置的时候顶点对与其绑定节点的相对坐标。

```
    void SetupVertices(CMS3DFile *mesh)
    {
        遍历每一个顶点
          for(size_t j=0;j<mesh->GetNumVertices();j++)
          {
            ms3d_triangle_t *mTr;
            ms3d_vertex_t   *mVt;
            mesh->GetVertexAt(j,&mVt);
            vec4 v,v2,v1;
            mat4 matrix;
            mat4 matrix2;
            if(mVt[j].boneId >= 0)
```

```
            {

                mat4_copy(matrix,mesh->_absoluteJointMatrices[mVt->boneId]);
                 mat4_inverse(matrix,matrix2);求对以节点的逆矩阵
                vec4_fromXYZ(v1,mVt->vertex[0], mVt->vertex[1], mVt->vertex[2]);
                根据上面的概念二，我通过步得到了顶点对于节点的位置坐标。
                mat4_transform(matrix2,v1,v2);
                mVt->vertex[0]=v2[0];
                mVt->vertex[1]=v2[1];
                mVt->vertex[2]=v2[2];

                }
            }

    }
```

通过以上两部节点建立，顶点绑定，这个时候准备工作已经完成，接下来需要进行动画，

步骤二、更新节点的绝对变换

前面说过节点的绝对变换矩阵里面包含节点的位置信息，转动信息，和其父节点的变化，也就是说只要我们根据动画的关节帧，以正确的顺序遍历所有节点，并且根据前一个节点变换后的矩阵乘以次节点对以前一个节点的相对变换，那么我们就刷新了所有骨骼的位置。如下代码：

```
void AdvanceAnimation(CMS3DFile *mesh)

{      遍历所有节点

    for(int i=0;i< mesh->GetNumJoints();i++)

   {

        mesh->GetJointAt(i,&jt);

        mat4_identity(keyFrameMatrix); 建立一个缓存用的 keyFrameMatrix（相对变换矩阵）

        if(jt->numKeyFramesRot == 0 && jt->numKeyFramesTrans == 0)

        {   如果这个模型的转动帧和移动帧为0，那么直接吧骨骼的绝对矩阵复制到
            _finalJointMatrices 最终变换矩阵里。continue 跳过后面的骨骼刷性。

            mat4_copy(mesh->_finalJointMatrices[i] ,mesh->_absoluteJointMatrices[i]);

            continue;
```

```
}
```

一下很长的一步识读取关键帧，计算中间差值，这一部是通过计算两个关键帧直接的时间差和变化差之的关系，求出该时刻的转动和移动程度是多少。如果你对下面有了解可以略过。

```
frame=mesh->m_currentTranslationKeyframe[i]; 首先或者移动的关键帧数
while ( frame < jt->numKeyFramesTrans && jt->keyFramesTrans[frame].time < time )
  {
      frame++;不断循环找到现在所在关键帧的位置。
  }
mesh->m_currentTranslationKeyframe[i] = frame;
if(frame==0)
{ 、、如果是第一帧那么保持标准姿势不变
p[0]=jt->keyFramesTrans[frame].position[0];
p[1]=jt->keyFramesTrans[frame].position[1];
p[2]=jt->keyFramesTrans[frame].position[2];
}
else if( frame ==jt->numKeyFramesTrans)
{
  如果是最后一帧，减一，此步骤防止程序指针越界
p[0]=jt->keyFramesTrans[frame-1].position[0];
p[1]=jt->keyFramesTrans[frame-1].position[1];
p[2]=jt->keyFramesTrans[frame-1].position[2];
}else
{
      如果现在帧在两个关键帧之间开始刷新骨骼。
```

```
ms3d_keyframe_pos_t &curFrame=jt->keyFramesTrans[frame];

ms3d_keyframe_pos_t &prevFrame=jt->keyFramesTrans[frame-1];

float timeDelta = curFrame.time-prevFrame.time;
```

计算差值，原理很简单看下一句代码你就知道了

```
float interpValue = ( float )(( time-prevFrame.time )/timeDelta );

p[0]=prevFrame.position[0]+( curFrame.position[0]-prevFrame.position[0] )*interpValue;;

p[1]=prevFrame.position[1]+( curFrame.position[1]-prevFrame.position[1] )*interpValue;;

p[2]=prevFrame.position[2]+( curFrame.position[2]-prevFrame.position[2] )*interpValue;;
```

保存三个方向的变化值

```
}
```

以下步骤同上是求旋转的变换

```
frame =mesh-> m_currentRotationKeyframe[i];
while ( frame < jt->numKeyFramesRot&& jt->keyFramesRot[frame].time < time )
  {
      frame++;
  }
mesh->m_currentRotationKeyframe[i] = frame;
if ( frame == 0 )
{   r[0]=jt->keyFramesRot[frame].rotation[0];
 r[1]=jt->keyFramesRot[frame].rotation[1];
 r[2]=jt->keyFramesRot[frame].rotation[2];
```

```
}else if(frame == jt->numKeyFramesRot)

{     r[0]=jt->keyFramesRot[frame-1].rotation[0];

 r[1]=jt->keyFramesRot[frame-1].rotation[1];

 r[2]=jt->keyFramesRot[frame-1].rotation[2];

}else

{       ms3d_keyframe_rot_t &curFrame = jt->keyFramesRot[frame];

        ms3d_keyframe_rot_t &prevFrame = jt->keyFramesRot[frame-1];

            float timeDelta = curFrame.time-prevFrame.time;

            float interpValue = ( float )(( time-prevFrame.time )/timeDelta );

            r[0]=
prevFrame.rotation[0]+( curFrame.rotation[0]-prevFrame.rotation[0] )*interpValue;

            r[1]                                                                =
prevFrame.rotation[1]+( curFrame.rotation[1]-prevFrame.rotation[1] )*interpValue;

            r[2]                                                                =
prevFrame.rotation[2]+( curFrame.rotation[2]-prevFrame.rotation[2] )*interpValue;
```

保存了三个旋转变换

```
}

 setRotationRadians(keyFrameMatrix,r);

 setTranslation(keyFrameMatrix,p);
```

上面两个步骤是将旋转变换和平移变换放置到 keyFrameMatrix（用之前别忘了单位化）中

```
mat4 finalRelativeMatrix,mt;
```

设置一个缓存用的最终变换矩阵

```
mat4_multiply2(mesh->_relativeJointMatrices[i],keyFrameMatrix,mt);
```

通过矩阵乘法将前面（初始化时候）求的 relativeJointMatrices（相对变换矩阵）和此时动画的变换矩阵累积起来，成为最终的相对变换矩阵。

```
mat4_copy(finalRelativeMatrix,mt);

        if(strcmp(jt->parentName,"") == 0)

        {这个时候要判断如果是第一个节点那么最终变换矩阵就是他的最终的相对变换矩阵

          mat4_copy(mesh->_finalJointMatrices[i],finalRelativeMatrix);

         }

          else

          {

            如果不是跟节点那么将他的最终的相对变换矩阵乘以父节点的最终变换矩阵就可以得到次节点的最终变换矩阵，原理和前面设置节点的步骤一样

            int index =    mesh->FindJointByName(jt->parentName);

            mat4 tm;

            mat4_multiply2(mesh->_finalJointMatrices[index],finalRelativeMatrix,tm);

            mat4_copy(mesh->_finalJointMatrices[i],tm); } }}
```

刷新骨骼后那么节点会根据每次累积的变换从标准姿势摆放到对应位置，记住！不要改变相对变换矩阵_relativeJointMatrices[i]和最终变换矩阵_absoluteJointMatrices[i]。应为这两个数据保存了节点的标准状态和节点间的相对变换。

步骤三、渲染顶点。

这里定点的信息其实在之前的初始化顶点的时候全部改变了，他里面保存的数据并不是顶点的绝对坐标，而是相对于骨骼的相对坐标，这是为了方便后面的新的顶点位置的计算，我们不需要考虑节点变换了多少，只需要将节点的最终变换矩阵乘以顶点的相对位置就可以求出新的顶点绝对坐标（世界坐标）了。

代码如下：

```
glBegin(GL_TRIANGLES);

        for(size_t i=0;i <Gn;i++)

        {     遍历所有组

              ms3d_group_t *mGp;
```

```
mesh->GetGroupAt(i,&mGp);
glBindTexture(GL_TEXTURE_2D, mesh->texture[Gn]);
for(size_t j=0;j<mGp->numtriangles;j++)
{
```

遍历所有三角形

```
   ms3d_triangle_t *mTr;
   ms3d_vertex_t      *mVt[3];
   mesh->GetTriangleAt(mGp->triangleIndices[j],&mTr);
   mesh->GetVertexAt(mTr->vertexIndices[0],&mVt[0]);
   mesh->GetVertexAt(mTr->vertexIndices[1],&mVt[1]);
   mesh->GetVertexAt(mTr->vertexIndices[2],&mVt[2]);

   float nv1[3],nv2[3],nv3[3];
   vec4 v1,v2;
   mat4 matrix;
```

或者顶点绑定的骨骼最终变化矩阵

```
   mat4_copy(matrix,mesh->_finalJointMatrices[mVt[0]->boneId]);
   vec4_fromXYZ(v1,mVt[0]->vertex[0], mVt[0]->vertex[1], mVt[0]->vertex[2]);
   mat4_transform2(matrix,v1,v2);
```

这部是关键，计算顶点最新的位子

```
   nv1[0]=v2[0];nv1[1]=v2[1];nv1[2]=v2[2];
```

注意！不要覆盖顶点的原始信息，应为我们的相对位置没有改变，改变的只是变化矩阵，所以为了不改变顶点，我们需要用一个新的变量保存

一下是 OpenGL 渲染步骤

```
   mat4_copy(matrix,mesh->_finalJointMatrices[mVt[1]->boneId]);
   vec4_fromXYZ(v1,mVt[1]->vertex[0], mVt[1]->vertex[1], mVt[1]->vertex[2]);
```

```
            mat4_transform2(matrix,v1,v2);
            nv2[0]=v2[0];nv2[1]=v2[1];nv2[2]=v2[2];

            mat4_copy(matrix,mesh->_finalJointMatrices[mVt[2]->boneId]);
            vec4_fromXYZ(v1,mVt[2]->vertex[0], mVt[2]->vertex[1], mVt[2]->vertex[2]);
            mat4_transform2(matrix,v1,v2);

            nv3[0]=v2[0];nv3[1]=v2[1];nv3[2]=v2[2];
            glTexCoord2f(mTr->s[0],1.0f - mTr->t[0]);
            glVertex3fv(nv1);

            glTexCoord2f(mTr->s[1],1.0f - mTr->t[1]);
            glVertex3fv(nv2);

            glTexCoord2f(mTr->s[2],1.0f - mTr->t[2]);
            glVertex3fv(nv3);
        }
    }
    glEnd();
```