<https://blog.csdn.net/gamemonkey/article/details/44058291>

<https://www.cnblogs.com/zhanglitong/p/3196752.html>

<http://blog.sina.com.cn/s/blog_13aa39c310102wolp.html> 这里有unity的骨骼动画

**3D模型动基本原理和分类**

3D模型动画的基本原理是让模型中各定点的位置随时间变化。主要种类有Morph动画、关节动画和骨骼蒙皮动画（Skinned Mesh）。从动画数据的角度来说，三者一般都采用关键帧技术，即只给出关键帧的数据，其他帧的数据使用插值得到。但由于这三种技术的不同，关键帧的数量是不一样的。

1. Morph（渐变、形变）动画

该动画是直接指定动画每一帧的顶点位置，其动画关键中存储的是Mesh所有顶点在关键帧对应时刻的位置

1. 关节动画

关节动画模型不是一个整体的Mesh，而是分成很多部分（mesh），通过一个父子层次结构将这些分散的Mesh组织在一起，父Mesh带动其下子Mesh的运动，各Mesh中的顶点坐标定义在自己的坐标系中，这样各个Mesh是作为一个整体参与运动的。动画帧中设置各子Mesh相对于其父Mesh的变换（主要是渲染，当然也可包括移动和缩放），通过子到父，一级级的变换累加（技术上，如果是矩阵操作是累乘）得到该Mesh在整个动画模型坐在的坐标空间中的变换（本文的视角就是世界坐标系），从而确定每个Mesh在世界坐标系中的位置和方向，然后以Mesh为单位渲染即可。关节动画的问题是，各部分Mesh中的顶点是固定在mesh坐标系中的，这样两个Mesh结合处就可能产生裂缝

1. 骨骼蒙皮动画（Skinned Mesh）

骨骼蒙皮动画的出现解决了关节动画的裂缝问题，而且效果非常酷。骨骼动画的基本原理可以概括为：在骨骼控制下，通过顶点混合动态计算蒙皮网格的顶点，而骨骼的运动相对于其父骨骼，并由动画关键帧数据驱动。一个骨骼动画通常包括骨骼层次结构数据、网格（Mesh）数据、网格蒙皮数据（skin info）和骨骼的动画（关键帧）数据。

**Skinned Mesh原理和结构分析**

Skinned Mesh中文一般称为骨骼蒙皮动画，这种动画包含骨骼（Bone）和蒙皮（Skinned Mesh）两个部分。Bone的层次结构和关节动画类似，Mesh则和关节动画不同：关节动画中是使用多个分散的Mesh，而Skinned Mesh中Mesh是一个整体，也就是说只有一个Mesh，实际上如果没有骨骼让Mesh运动变形，Mesh就和静态模型一样了。Skinned Mesh技术的精华在于蒙皮，所谓的皮并不是模型的贴图（也许有人会这么想过吧），而是Mesh本身，蒙皮是将Mesh中的顶点附着（绑定在骨骼上），而且每个顶点可以被多个骨骼所控制，如果贴图是皮，那么普通静态模型不也都有吗？所以我觉得应该理解为具有蒙皮信息的Mesh或可当做皮肤的Mesh，这个皮肤就是Mesh。而为了有皮肤功能，Mesh还需要有蒙皮信息，即Skin数据，没有Skin数据就是一个普通的静态Mesh了。skin数据决定顶点如何绑定到骨骼上。顶点的Skin数据包括顶点受哪些骨骼影响以及这些骨骼影响该顶点时的权重（weight），另外，对于每块骨骼还需要骨骼偏移矩阵（BoneOffsetMatrix）用来将顶点从Mesh空间变换到骨骼空间，在本文中，提到骨骼动画中的Mesh特指这个皮肤Mesh，提到模型是只骨骼动画模型整体。骨骼控制蒙皮运动，而骨骼本身的运动呢？当然是动画数据了。每个关键帧中包含时间和骨骼运动信息，运动信息可以用一个矩阵直接表示骨骼新的变换，也可以四元数表示骨骼的旋转，也可以随便自己定义什么只要能让骨骼动就行。除了使用编辑设定好的动画帧数据，也可以使用武力计算对骨骼进行实时控制。

**蒙皮：**

蒙皮，三维动画术语，也用于3D游戏中。三维动画的一种制作技术。在三维软件中创建的模型基础上，为模型添加骨骼。由于骨骼与模型是相互独立的，为了让骨骼驱动模型产生合理的运动。把模型绑定到骨骼上的技术叫做蒙皮。

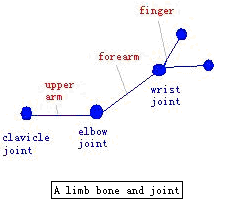
**理解骨骼和骨骼层次结构（Bone Hierarchy）**

首先要明确一个概念：骨骼决定了模型整体在世界坐标系中的位置和朝向

先看静态模型吧，静态模型没有骨骼，我们在世界坐标系中防止静态模型时，只要指定模型自身坐标系在世界坐标系的位置和朝向。在骨骼动画中，不是把Mesh直接放到坐标系中，Mesh只是作为Skin使用，是依附于骨骼的，真正决定模型在世界坐标系的位置和朝向是骨骼。在渲染静态模型时，由于模型的顶点都是定义在模型坐标系中，所有各顶点只要经过模型坐标系到世界坐标系的变换都可以渲染。而对于骨骼动画，我们设置模型的位置和朝向，实际是在设置根骨骼的位置和朝向，然后根据骨骼层次结构中父子骨骼之间的变换关系计算出各个骨骼的位置和朝向，然后根据骨骼对Mesh中顶点的绑定计算出顶点在世界坐标系中的坐标，从而对顶点进行渲染。要记住：在骨骼动画中，骨骼才是模型主体，Mesh不过是一层皮，一件衣服

如何理解骨骼？请看第二个概念：骨骼可理解为一个坐标空间

在一些文章中往往会提到关节和骨骼，那么关节是什么？骨骼是什么？下图是一个手臂的骨骼层次示例：

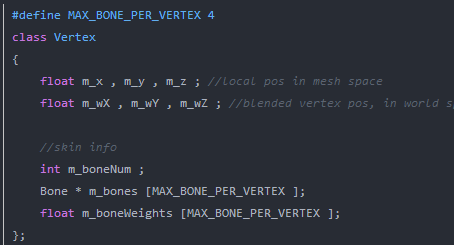


骨骼只是一个形象的说法，实际上骨骼可理解为一个坐标空间，关节可以理解为骨骼坐标空间的原点。关节的位置由它在父骨骼坐标空间中的位置描述。上图中有三块骨骼，分别是上臂、前臂和两个手指。Clavicle（锁骨）是一个关节，它是上臂的原点，同样肘关节（elbow joint）是前臂的原点，腕关节（wrist joint）是手指骨骼的原点。关节既决定了骨骼空间的位置，又是骨骼空间的旋转和缩放中心。为什么用一个4\*4的矩阵就可以表达一个骨骼，因为4\*4矩阵中含有的平移分量决定了关节的位置，旋转和缩放分量决定了骨骼空间的旋转和缩放。我们来看前臂这个骨骼，其原点位置是位于上臂上某处的，对于上臂来说，它直到自己的坐标空间某处（即肘关节所在的位置）有一个子空间，那就是前臂，至于前臂里是啥就不考虑了，当前臂绕肘关节旋转时，实际是前臂坐标空间在旋转，从而其中包含的子空间也在绕肘关节旋转，在这个例子中是finger骨骼。和实际生物骨骼不同的是，我们这个的骨骼并没有实际的骨头，所以前臂旋转时，它自己没啥可转的，改变的只是坐标空间的朝向，你可以说是上图的蓝线在转，但实际蓝线并不存在，蓝线只是画上去表示骨骼之间关系的，真正转的是骨骼空间，我们能看到在转的是wrist joint，也就是两个finger骨骼的坐标空间，因为他们是子空间，会随着父空间运动，就好比人跟着地球转一样。

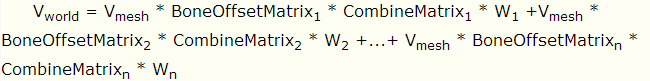
骨骼就是坐标空间，骨骼层次就是嵌套的坐标空间。关节只是描述骨骼的位置即骨骼自己的坐标空间原点在其父空间中的位置，绕关节旋转时指骨骼坐标空间（包括所有的子空间）自身的旋转。但还有两个可能的疑问，一个骨骼的长度问题，由于骨骼是坐标空间，没有所谓的长度和宽度的限制，我们看到的长度一方面是蒙皮之后的结果，另一方面子骨骼的原点（也就是关节）的位置往往决定了视觉上父骨骼的长度，比如这个upper arm线段的长度实际是有elbow joint的位置决定的。第二个问题，手指的端点是啥？实际上在我们的例子中手指没有子骨骼，所以那个断点并不存在，那么为了方便演示画上去的，实际问题中总有最下层的骨骼，他们不能决定其他骨骼了，他们的作用只剩下控制Mesh顶点。那么手指的长度如何确定？我们看到的长度应该是由手指部分的顶点和蒙皮决定的，也就是由Mesh中属于手指的那些点离腕关节的距离决定的。

**蒙皮信息和蒙皮过程**

之前曾经说过，Skinned Mesh中Mesh是作为皮肤使用的，蒙在骨骼之上。为了让普通的Mesh具有蒙皮功能，必须添加蒙皮信息，即Skin info。我们知道Mesh是由顶点构成的，建模时顶点是定义在模型自身坐标系中，即相对于Mesh原点，而骨骼动画中决定模型点点最终世界坐标的是骨骼，所以要让骨骼决定顶点的世界坐标，这就要将顶点和骨骼联系起来，Skin info正是起这个作用。



Skin info的作用是使用各个骨骼的变换矩阵对顶点进行变换并乘以权重，这样某块骨骼只能对该顶点产生部分影响，各个骨骼权重之和应该为1。Skin info主要包含当前mesh上的顶点受到哪些骨骼影响，各个骨骼的影响权重。

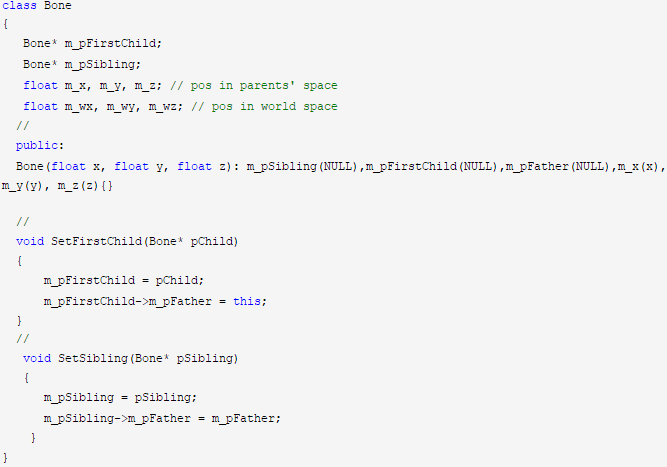


**Bone Offset Matrix的含义和计算方法**

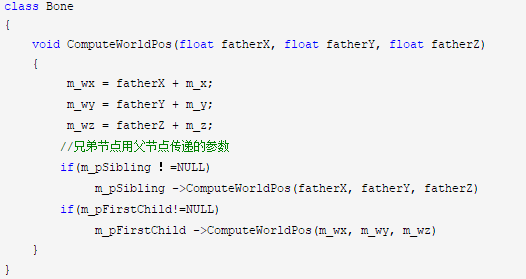
前面已经说过，骨骼动画中决定模型顶点最终会世界坐标的是骨骼，所以要让骨骼决定顶点的世界坐标，现在我们看下顶点受一块骨骼的作用时的坐标变换过程

Mesh vertex(defined in mesh space) 🡪 Bone space 🡪 World space

从这个过程可以看出，需要首先将模型顶点从模型空间变换到某块骨骼自身的骨骼空间，然后才能利用骨骼的世界变换计算顶点的世界坐标。Bone Offset Matrix的作用正是将模型从顶点空间变换到骨骼空间。那么Bone Offset Matrix如何得到，下面具体分析：



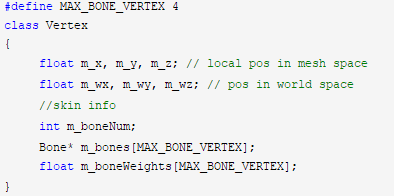
     这样，当父节点骨骼发生变换的时候，子节点的骨骼就会做相应的变换，这样的操作可以称为 UpdateBoneMatrix，这样的操作可以用一个方法ComputeWorldPos来表示，这样可以用递归的方式在Bone中实现，代码依据于文章1：



这样，当父节点骨骼发生变换的时候，子节点的骨骼都会做出相应的变换，从而得到最新的位置、朝向等信息，骨骼发生变化，从而会带动外在的mesh发生变化，所以整体的模型就表现chu出运动起来。基于此，可以理解为什么骨骼是骨骼动画的核心。

在说完骨骼后，对于整体模型在动画中骨骼的变换，可以有一个大致的理解，当时模型只是内在的，外在的表现是模型的蒙皮的变化，所以骨骼动画中的第二部分就是蒙皮的计算。这里的皮，就是前面说过的Mesh。

　　首先，需要明确的是Mesh所在的空间。在建模的时候，模型的Mesh是和骨骼一样处于同样的空间中的，Mesh中的各个顶点是基于Mesh的原点来进行定位的。但是模型在运动表现的时候，是根据骨骼的变换来做相应的动作的，对应的Mesh上的顶点就需要做出对应的转换，所以Mesh的顶点需要转换到对应的骨骼所在的坐标空间中，进行相应的位置变换，因此对应的需要添加蒙皮信息，也就是skin info，主要是当前顶点受到哪些骨骼的影响，影响的权重等，借用文章1的表述，可以用C++表示一个顶点类，代码依据于文章1：



　　当然，这儿只是一个简单的表述，具体的在引擎中会有规范的设计。那么我们的顶点在跟随骨骼做运动的时候，是如何计算自己的位置的？我们就需要引入BoneOffsetMatrix 和 Transform Matrix的概念。

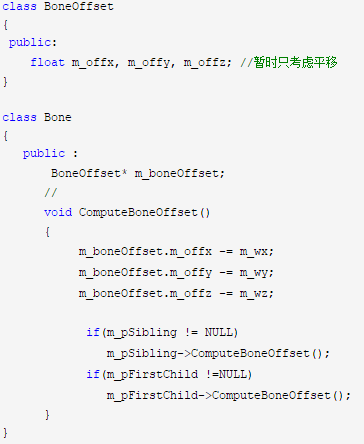
     在前面，我们已经提到，顶点需要依附于骨骼进行位置计算，但是建模的时候，顶点的位置是基于Mesh原点进行建模的，通常情况下，Mesh的原点是和模型的骨骼的根骨骼处于同一个坐标空间中，那么 BoneOffsetMatrix就是用来将Mesh中顶点从Mesh空间转换到骨骼所在空间中。

　　在建模的时候，对于每个骨骼，我们是可以得到其对应的Transform Matrix（用来层层计算到父节点所在空间中），其中根骨骼的Transform Matrix是基于世界空间的转换，所以对于每一个下面的子骨骼，要计算其Transform Matrix，需要进行一个矩阵的连乘操作。最后得到的最终矩阵连乘结果矩阵就是Combined Transform Matrix，基于这个矩阵，就可以将顶点从骨骼所在的空间转换到世界空间中。反过来，这个矩阵的逆矩阵（一般只考虑可以取逆的操作），就是从世界空间中转换到该骨骼的空间中，由于Mesh的定义基于Mesh原点，Mesh原点就在世界空间中，所以这个逆矩阵就是要求的 Offset Matrix，也被称为Inverse Matrix，这个逆矩阵一般实在初始位置中求得，通过取逆即可获得。

　　在实际的计算中，每个骨骼可能会对应多个顶点，如果每个顶点都保存其对应的骨骼的变换矩阵，那么大量的顶点就会报错非常多的变换矩阵。所以我们只需要保存当前该骨骼在初始位置，对应的从世界空间到其骨骼空间的变换矩阵，那么其对应的每个顶点在每次变换操作的时候，只需要对应的用offset Matrix来操作即可。

   对于上面的Transform Matrix和offset Matrix，是纳入了旋转、平移和缩放的。其实offset Matrix取决于骨骼的初始位置，此时一般只包含了平移（此时还没有动画，所以没有旋转和缩放），在动画中，一般也以缩放为主（所以大部分的动画的关键帧用四元数表示）。在矩阵中都包含，是处于兼容性考虑。

下面就基于平移，做一个基本的蒙皮计算：

仔细捋一捋上面的代码，就可以理解整体的蒙皮变换的过程，当然，这儿只用了矩阵变换中的平移变换，如果考虑加上旋转和缩放，则回到最初的计算公式中了。至此，对于基本的骨骼动画中的骨骼变换和蒙皮变换，有了一个详细的解释。下面说说Unity中是如何处理骨骼变换的。

**Unity3D骨骼动画处理**

前面讲解的对于骨骼动画中的骨骼变换，蒙皮的计算，都是在CPU中进行的。在实际的游戏引擎中，这些都是分开处理的，较为通用的处理是将骨骼的动画数据驱动放在CPU中，计算出骨骼的变换矩阵，然后传递给GPU中进行蒙皮计算。在DX10的时候，一般的shader给出的寄存器的大小在128的大小，一个变换矩阵为4x4，如果去除最后一行(0,0,0,1)就可以用3个float表示，那么最多可以表示，嗯，42个左右，如果考虑进行性能优化，不完全占用寄存器的大小，那么一般会限制在30根骨骼的大小上。将这些骨骼的变换矩阵在CPU进行计算后，就可以封装成skin info传递到GPU中。

   在GPU的计算中，就会取出这些mesh上的顶点进行对应的位置计算，基于骨骼的转换矩阵和骨骼的权重，得到最新的位置，从而进行一次顶点计算和描绘。之所以将骨骼动画的两个部分分开处理，一个原因就是CPU的处理能力相对而言没有GPU快捷，一般一个模型的骨骼数量是较小的，但是mesh上的顶点数量较大，利用GPU的并行处理能力优势，可以分担CPU的计算压力。

   在DX11还是DX12之后（记不太清楚），骨骼变换矩阵的计算结果不再存储在寄存器中，而是存储在一个buffer中，这样的buffer大小基于骨骼数量的大小在第一次计算的时候设定，之后每次骨骼动画数据驱动得到新的变换矩阵，就依次更改对应的buffer中存储的变换矩阵，这样就不再受到寄存器的大小而限制骨骼的根数的大小。但是实际的优化中，都会尽量优化模型的骨骼的数量，毕竟数量越多，特别是影响顶点的骨骼数量越多，那么计算量就会越大，正常的思维是优化骨骼数量而不是去扩展buffer的大小：D

   在文章2中，对于GPU的蒙皮计算做了较大的性能优化，主要的思维也是这样，在CPU中进行骨骼变换，将变换的结果传递到GPU中，从而进行蒙皮计算。基本的思维和前面说的变换思维一致，其基本的优化重点也是想利用一个buffer来缓存变换矩阵，从而优化性能。这儿我就重点分析一下shader部分的代码，其在cpu部分的代码处理基本和前面的代码思想一致：

Mesh space 是建模时使用的空间， mesh 中顶点的位置相对于这个空间的原点定义。比如在 3d max 中建模时（视 xy 平面为地面， +z 朝上），可将模型两脚之间的中点作为 Mesh空间的原点，并将其放置在世界原点，这样左脚上某一顶点坐标是（ 10 ， 10 ， 2 ），右脚上对称的一点坐标是（ -10 ， 10 ， 2 ），头顶上某一顶点的坐标是（ 0 ， 0 ， 170 ）。由于此时 Mesh 空间和世界空间重合，上述坐标既在 Mesh 空间也在世界空间，换句话说，此时实际是以世界空间作为 Mesh 空间了。在骨骼动画中，在世界中放置的是骨骼而不是 Mesh，所以这个区别并不重要。在 3d max 中添加骨骼的时候，也是将骨骼放入世界空间中，并调整骨骼的相对位置使得和 mesh 相吻合（即设置骨骼的 TransformMatrix ），得到骨架的初始姿势以及相应的 Transform Matrix( 按惯例模型做成两臂侧平举直立，骨骼也要适合这个姿态 ) 。由于骨骼的 Transform Matrix （作用是将顶点从骨骼空间变换到上层空间）是基于其父骨骼空间的，只有根骨骼的 Transform 是基于世界空间的，所以要通过自下而上一层层 Transform 变换（如果使用行向量右乘矩阵，这个 Transform 的累积过程就是C=Mbone*Mfather*Mgrandpar*...*Mroot ） , 得到该骨骼在世界空间上的变换矩阵 - Combined Transform Matrix ，即通过这个矩阵可将顶点从骨骼空间变换到世界空间。那么这个矩阵的逆矩阵就可以将世界空间中的顶点变换到某块骨骼的骨骼空间。由于 Mesh 实际上就是定义在世界空间了，所以这个逆矩阵就是 Offset Matrix 。即 OffsetMatrix 就是骨骼在初始位置（没有经过任何动画改变）时将 bone 变换到世界空间的矩阵（ CombinedTransformMatrix ）的逆矩阵，有一些资料称之为 InverseMatrix 。在几何流水线中，是通过变换矩阵将顶点变换到上层空间，最终得到世界坐标，逆矩阵则做相反的事，所以 Inverse 这种提法也符合惯例。那么 Offset 这种提法从字面上怎么理解呢？ Offset 即骨骼相对于世界原点的偏移，世界原点加上这个偏移就变成骨骼空间的原点，同样定义在世界空间中的点经过这个偏移矩阵的作用也被变换到骨骼空间了。从另一角度理解，在动画中模型中顶点的位置是根据骨骼位置动态计算的，也就是说顶点跟着骨骼动，但首先必须确定顶点和骨骼之间的相对位置（即顶点在该骨骼坐标系中的位置），一个骨骼可能对应很多顶点，如果要保存这个相对位置每个顶点对于每块受控制的骨骼都要保存，这样就要保存太多的矩阵了。。。所以只保存 mesh 空间到骨骼空间的变换（即 OffsetMatrix ），然后通过这个变换计算每个顶点在该骨骼空间中的坐标，所以 OffsetMatrix 也反应了 mesh 和每块骨骼的相对位置，只是这个位置是间接的通过和世界坐标空间的关系表达的，在初始位置将骨骼按照模型的形状摆好是关键之处。