3D模型格式和3D动画

郑立松

Albert Zheng <lisong.zheng@gmail.com>

2012年12月

常用3D模型格式比较

Name	Developer	data type.	Skeletal Animation	Vertex Animation	Other features	Plus	Minus
3DS - 3D studio	Autodesk	application/octet-stream (binary)	✓	✓	Native support with 3ds max, common interchange format.	Easy to use, lots of documentation.	Designed for Editor
Autodesk FBX	Autodesk	text/plain	✓	✓	Native support with 3ds max, mudbox etc, common interchange format, very widely used.	Easy to use, tons of official documentation. Interchange	Plaintext can create large files
DAE - COLLADA - COLLAborative Design Activity	Khronos / Sony Computer Entertainment	text/**xml**	✓	✓	Effects, Shaders (GLSL, GLES, Cg), Materials, Textures, Lights, Cameras, Physics, Instancing	Easy to use, lots of documentation (Khronos shares the copyright). Interchange	Plaintext can create large files
MD5 - Quake 4 and Doom 3 Object Format	Id Tech.	application/octet-stream (binary)	✓	✓	Camera Animation, Cutscenes	Gpl'd. Designed for game!	
OBJ - Wavefront OBJ Format	Wavefront Technologies	text/plain	✗	✗		Simple Clean model format, easy to read (both by human and computer), most software exports to obj.	Bulky files, no animation.

小结 – 3D模型格式

- 应用场景：
  - 数据交换、转换：DAE，FBX
  - 场景和静态模型：MD5
  - 动画模型：MD5
- 自研3D引擎目前采用的格式：
  - DAE：数据交换
  - MD5：引擎运行时 → 未来发展为自有的格式
  - 最终目标：生产版本会制定自己的专用格式，量身定做和优化。

3D动画实现原理和类型

- 3D动画的实现原理：
  - 一般都是基于**关键帧动画(Keyframed Animation)**，即，“提供一种机制，让各顶点位置（以及其他属性）随时间发生变化”。
  - 导出时只关注关键帧的数据，然后在运行时用**线性**或**球面插值**计算中间帧的数据。
- 常见的三种动画类型：

1) 顶点动画(Vertex Animation)：

a) 变形动画(Morph Target Animation)

b) 姿态动画(Pose Animation)

2) 骨骼动画(Skeletal Animation)

3) 骨骼蒙皮动画(Skinned Mesh Animation)

顶点动画

- **顶点动画(Vertex Animation)**：通过对物体的运动进行快照(在变形动画中被称为变形目标“Morph Targets”)，得到关键帧上各顶点的实际位置。
  - **变形动画(Morph Target Animation)**：保存**每个顶点**的**绝对**位置。
  - **姿态动画(Pose Animation)**：只保存**发生变化的顶点**的**相对**位置，更节省资源。
- 优点：
  - 执行时高效，因为它只需要简单的把每一帧进行线性插值就可以得到全部动画效果，而不用进行复杂的运算。
- 缺点：
  - **耗资源**：一种资源密集型(Resource-Intensive)方案，在执行的过程中会把所有(在模型资源中)动画需要移动的顶点数据进行拷贝，这项工作会耗费大量资源；
  - **灵活性不够**：无法对同一个模型(例如手部模型)同时混合使用不同的变形动画，除非是不同的模型，才能混合动画(比如两套动画分别控制胳膊和手)。
    - 注：姿态动画可以混合，但会引入“影响权重”， 用来决定在最终混合的时候同一个顶点偏移到这个姿态的幅度。
- 应用场景：某些比较细致的动画效果只能通过这种方法实现（比如脸部动画效果：“嘴巴动”，“眼睛眨动”）。

顶点动画图例

骨骼动画

- **骨骼动画(Skeletal Animation)**的基本思路：如果将皮肤看作是刚体，运动时都是皮肤跟随着骨骼在动，皮肤相对于它的骨骼本身并没有发生运动，所以只要描述清楚骨骼的运动就行了。
  1) 骨骼被建模为一个类似多叉树一样的父子结构，用矩阵描述各个骨骼的相对于父骨骼的运动。
  2) 人体各组件的皮肤Mesh绑定到相应骨骼上，使用骨骼变换矩阵进行变换，从而实现跟随着骨骼运动的效果。
- 优点：
  - 节省资源：只须保存各个关键帧上的各骨骼的运动矩阵。
  - 灵活：可以灵活实现动画混合，让你可以同时将不同的动画作用在一个模型上。
- 缺点：运动时在两段骨骼的关节交接处，容易产生裂缝，影响效果。

Rigid Body Animation Model and Its Issue

- The rigid body character animation model has a definite flaw by separating a character's mesh into unconnected parts.

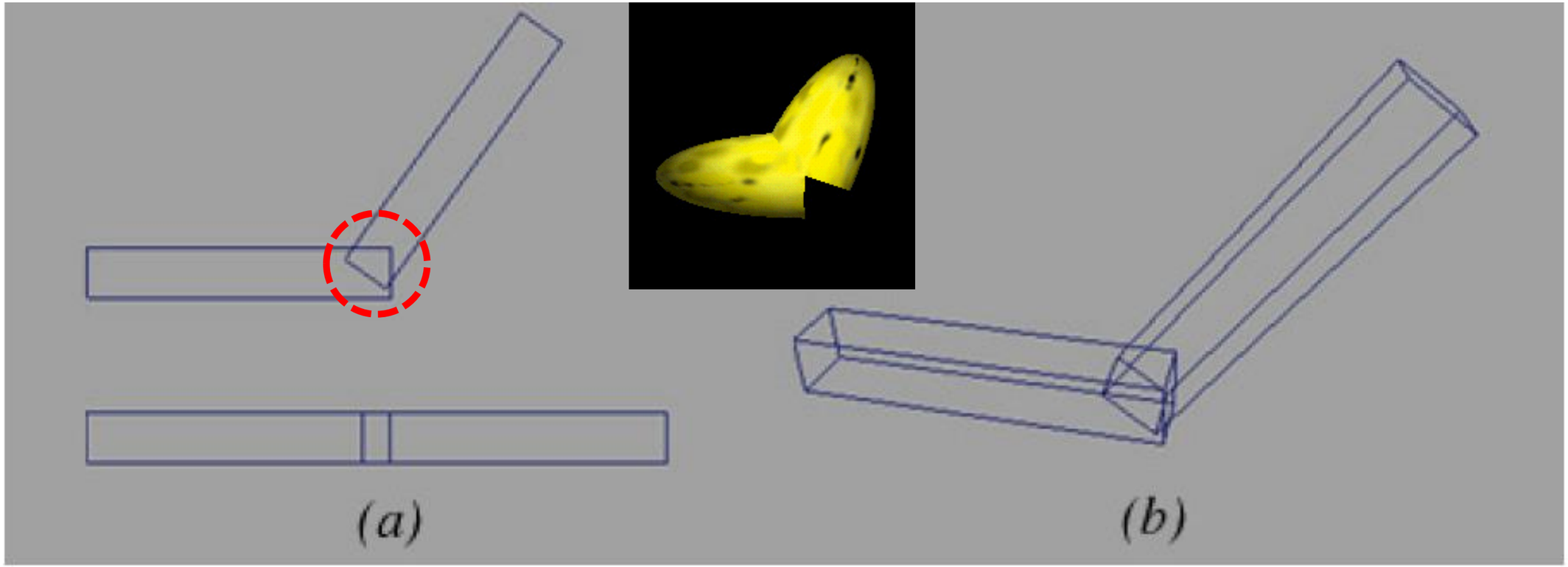


Figure 9: a) Observe the sharp unnatural bending that occurs when we rotate a bone using two separate meshes. b) A perspective view of the unnatural bending.

骨骼蒙皮动画

- **骨骼蒙皮动画(Skinned Mesh Animation)**：在骨骼动画的基础上，加以改进，解决了骨骼动画的裂缝问题 。
  - 骨骼动画产生裂缝的原因：它将人体Mesh分解为若干个组件Mesh，然后每一个人体组件Mesh只受到一个骨骼的影响，例如，上臂Mesh绑定到上臂骨骼。
  - 改进方案：将人体皮肤Mesh看作一个整体，通过**顶点混合(Vertex Blend)**技术将各顶点“蒙皮(Skinning)”到骨架上的多个骨骼，通过权重来决定运动中几个骨骼对同一顶点的影响。
- 优点：骨骼动画的优点 + 解决刚体动画的裂缝问题
- 代表游戏：现代端游

Vertex Blending in Skinning Information

- **Vertex Blending**: A vertex can be linked (influenced) by one or more bones in the bone hierarchy. The amount a bone influences a vertex is determined by a **Weight** value as shown in Figure 3.4.

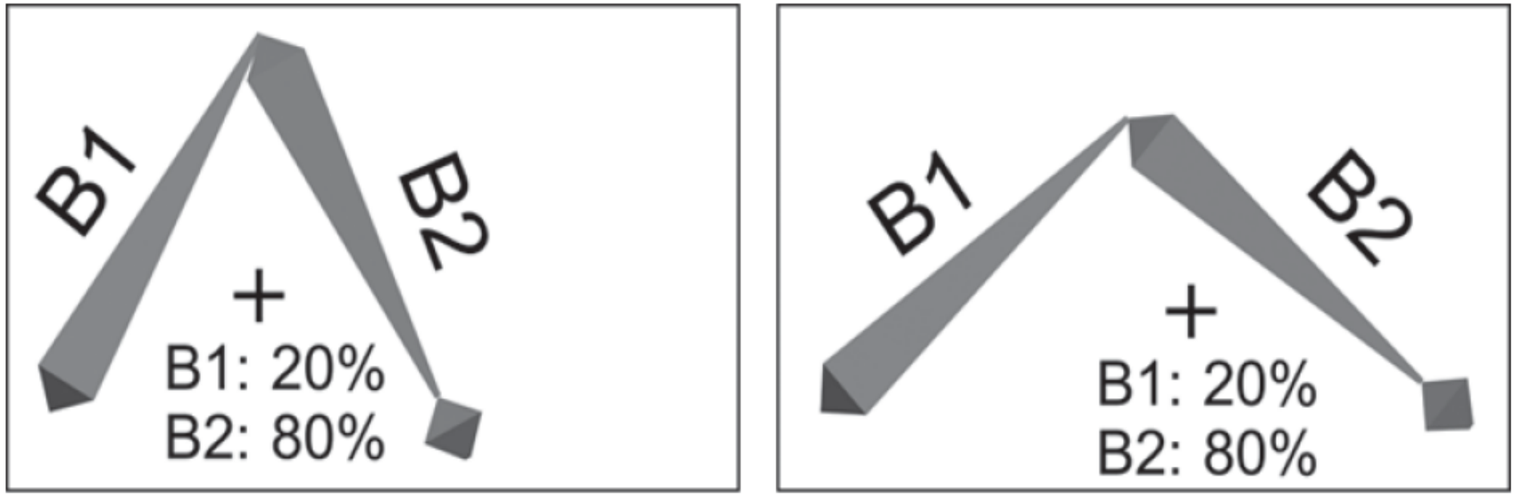


FIGURE 3.4
An example of how a vertex (the cross) is affected by two bones (B1 and B2) with the weights 20% and 80%, respectively. Notice how the vertex follows B2 more than B1 due to the weights.

A Solution: Vertex Blending

- Treating the character's skin as One Continuous Mesh.
- Key Idea: The position of the vertices near the joint are determined by a weighted average of the two influential bone transformations.

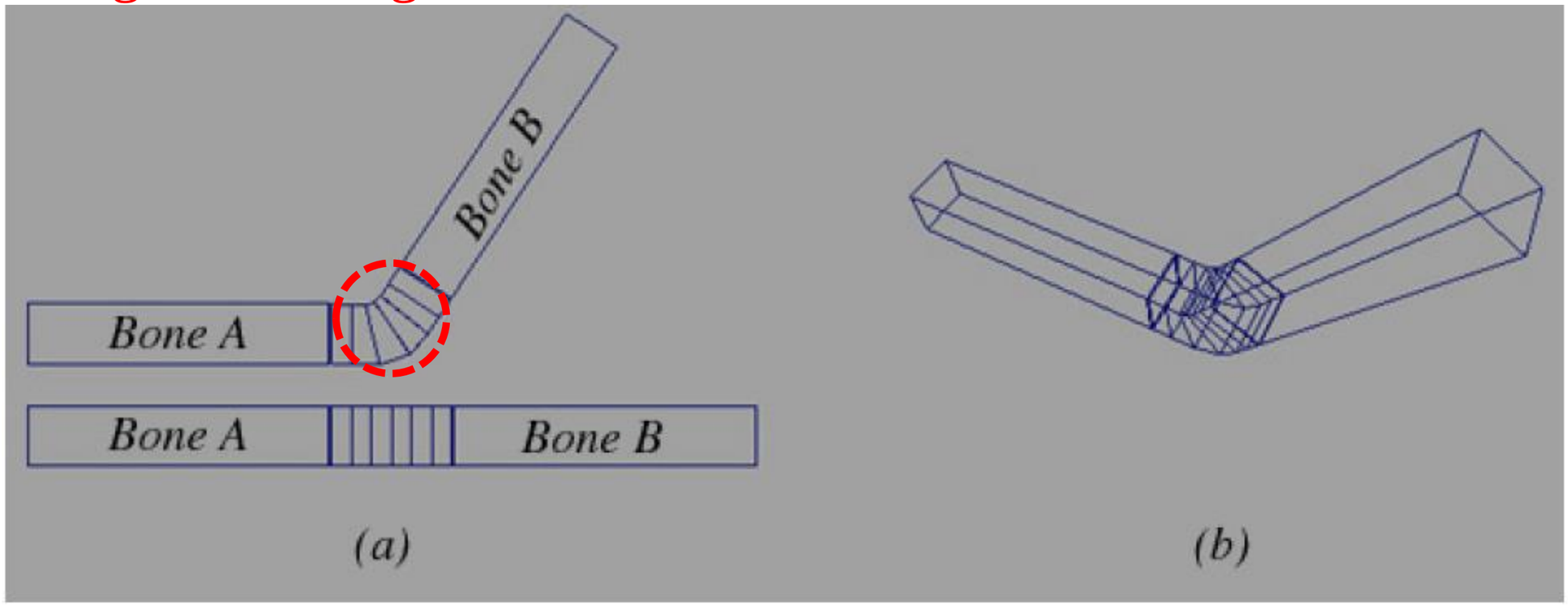
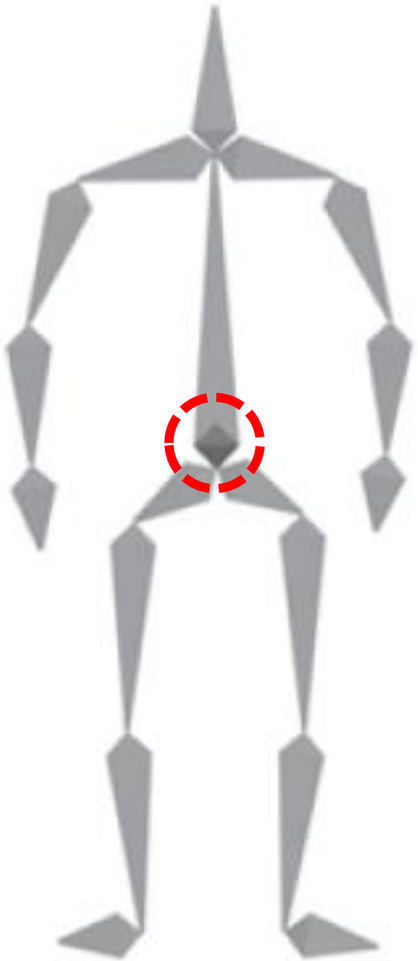


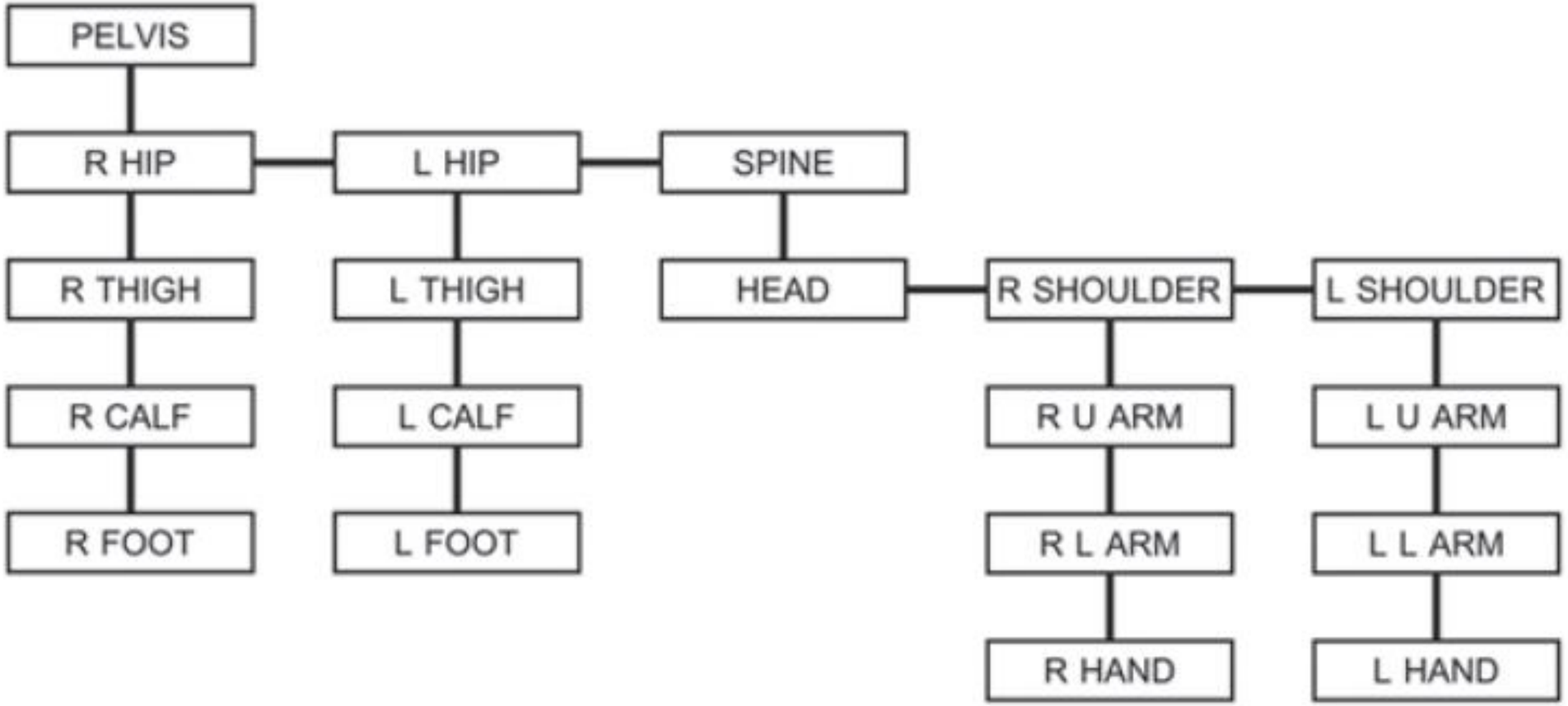
Figure 10: Note the skin is one continuous mesh that spans both bones. a) Observe that the vertices near the joint are influenced by both Bone A and Bone B to create a smooth transitional blend from bone A's position to bone B's position. b) A perspective view of vertex blending.

Illustration of Skinned Mesh Animation

1) Modeling ***Bone Hierarchy***: The first step is to create a bone hierarchy, more commonly by a 3D modeling program.
2) Modeling the **Skin *Mesh***, and defining ***Skinning Information***: This is also done in a 3D modeling program .
3) dynamically Skinning, and rendering the character in 3D engine runtime: This can be done in two different ways, using either **software skinning** or **hardware skinning**.

An example of a bone hierarchy





Software Skinning

- **Software Skinning**:
  - The positions of each vertex in a mesh are calculated using the mathematical formula in CPU.
  - The result is stored in a temporary mesh that is then consequently rendered.
- **Pro**: Simple, straightforward.
- **Cons**: But also very slow compared to hardware skinning.

Rendering Steps of Software Skinning

- Brief overview of the rendering steps of Software Skinning:
  1) Load a bone hierarchy and associated meshes, skinning information.
  2) For each frame, update the skeleton pose.
  3) Update the target mesh.
  4) Render the target mesh like a static mesh.

Hardware Skinning

- **Hardware Skinning**: The positions of each vertex in a mesh are calculated using Vertex Shader in GPU.
- Pro: However, what you lose in functionality, you make up readily in speed.
- Cons: Hardware skinning also has some limits, such as:
  1) *How many bones can influence a vertex*?
  2) *How many bones you can have per character* without having to split up the mesh into several parts?
     - If you have a character with more than the limitation, then the mesh will need to be split into ***multiple parts*** and *rendered in* ***several passes***.

Special notes for Flash Stage3D AGAL

- 《Real Time Rendering》 notes that we usually do not need more that four bone influences per vertex.
  - Therefore, in our design *we will consider a maximum of **FOUR** influential bones per vertex*.
- Special notes for Stage3D AGAL:
  1) Stage3D AGAL 1.0 only guarantees 128 vertex constant registers. Divide that by four and you have only enough memory for 32 4x4 matrices. → ***导致游戏中的人体最多只能建模32根骨骼.***
     - Solution #1 for resolving this problem: is to split the skinned mesh into multiple meshes and render it in parts, such that the matrix palette of each part can fit into the constant registers.
     - Solution #2 for resolving this problem: is to use Dual Quaternion (only needs 2 registers) rather than Matrix. Now you can model 32*2 bones per character in your game. → ***这个技巧可以让你建模最多64=32*2根骨骼。***
  2) AGAL 2.0 guarantees 250 constant registers →***人体最多只能建模62根骨骼***, which is usually enough in practice, and so we do not need to split the mesh with version 2.0. If using Dual Quaternion technique, it can be up to a maximum of 62*2 bones per character. → ***采用Dual Quaternion技巧，可建模高达124根骨骼.***

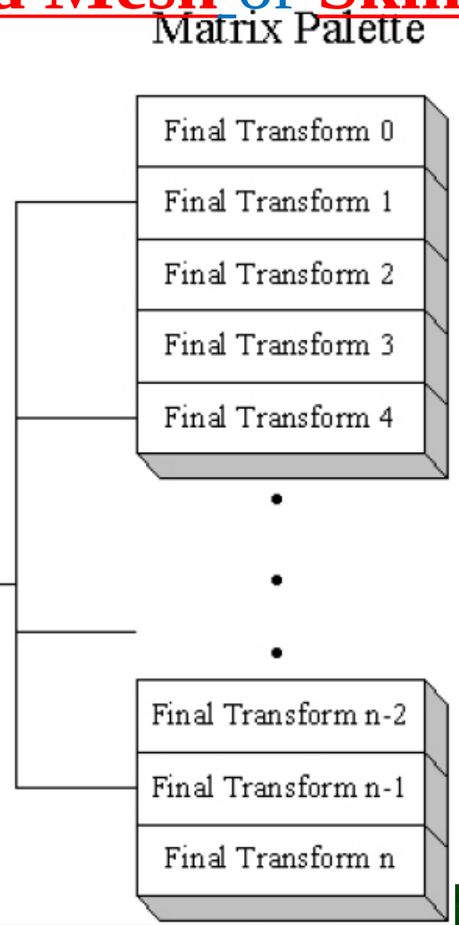
Rendering Steps of Hardware Skinning

- Brief overview of the rendering steps of Hardware Skinning:
  1) Load a bone hierarchy and associated meshes, skinning information.
  2) Convert the mesh to an **Indexed Blended Mesh**.
  3) For each frame, update the skeleton pose.
  4) Upload the **Matrix Palette** (bone matrices) to the vertex shader.
  5) Render the Indexed Blended Mesh using the vertex shader.

Indexed Blended Mesh

- The **Matrix Palette** is simply another name for the array of current bone transformations.
- A continuous mesh whose vertices have this format is configured for vertex blending and we call it a **Indexed Blended Mesh** or **Skinned Mesh**.

```
struct BlendVertex
{
    D3DXVECTOR3 pos;
    D3DXVECTOR2 texCoords;
    D3DXVECTOR4 weights;
    BYTE boneIndices[4];
};
```



Behind-the-scenes of Skinned Mesh Animation

- In computer graphics, **the bones** used to animate characters **are just a helping structure**, something that will never be rendered to the screen.
- Actually each bone determines ***4 transformation matrixes***:
  - **Local Matrix:** describing its position, orientation, and scale in relation to its parent bone.
    - Responsible for transforming a bone ***from its Bone Space to its Parent Bone Space***.
  - **Combined Matrix**: responsible for transforming a bone ***from its Bone Space to Character Space***.
  - **Offset Matrix**: responsible for transforming a Vertex ***from Bind Pose Space to the respective Bone Space***.
  - **Final Matrix**: responsible for transforming a Vertex ***from Bind Pose Space to Character Space***.

4 transformation matrixes of 3 Poses

Transformation Matrix	3 Poses in Skinned Mesh Animation		
	Bind Pose	**Key Frames Poses**	**Intermediate Poses**
Local Matrix	modeling Local Matrix in 3dsMax	modeling Animation's Transformation Data in 3dsMax, and update the Data to Local Matrix in runtime.	calculating Animation's Transformation Data by interpolating in runtime, and update the Data to Local Matrix in runtime.
Offset Matrix	modeling Offset Matrix in 3dsMax		
Combined Matrix	Calculating by matrix multiplication in runtime		
Final Matrix	Calculating by matrix multiplication in runtime		

How to calculate Combined Matrix?

- The combined transformation matrix is responsible for transforming a bone from its Bone Space to Character Space.
- Mathematically, the combined transformation matrix of the i^{th} bone C_i is given by:

$$\mathbf{C_i = L_i P_i}$$

  - Where L_i is the local transformation matrix of the i^{th} bone, and P_i is the combined transformation matrix of the i^{th} bone's parent.

Why needs Offset Matrix?

- Issue: In vertex blending, we define the character's skin as one continuous mesh in character space of bind pose. Consequently, because the vertices are not in bone space, we cannot simply use the combined transforms of the bones.
- To solve this, we introduce a new transform called the **Offset Transform**. Each bone in the bone hierarchy has a corresponding offset matrix.
- An offset matrix transforms vertices, in the bind pose, from bind space to the space of the respective bone.
  - The **Bind Pose** is the default layout of the character mesh geometry.

How to calculate Final Matrix?

- So we now introduce a new transform, call it the **Final Transform**, which combines a bone's offset transform with its combined transform.

$$\mathbf{F_i = O_i\, C_i}$$

- The final position v′ of any vertex v can be calculated with:

$$\mathbf{v}' = \left(\sum_{i=0}^{n-1} w_i \mathbf{v}\mathbf{F}_i \right) = \left(w_0 \mathbf{v}\mathbf{F}_0 + w_1 \mathbf{v}\mathbf{F}_1 + \ldots + w_{n-1} \mathbf{v}\mathbf{F}_{n-1} \right)$$

$F_i = O_i C_i$ 的示意图

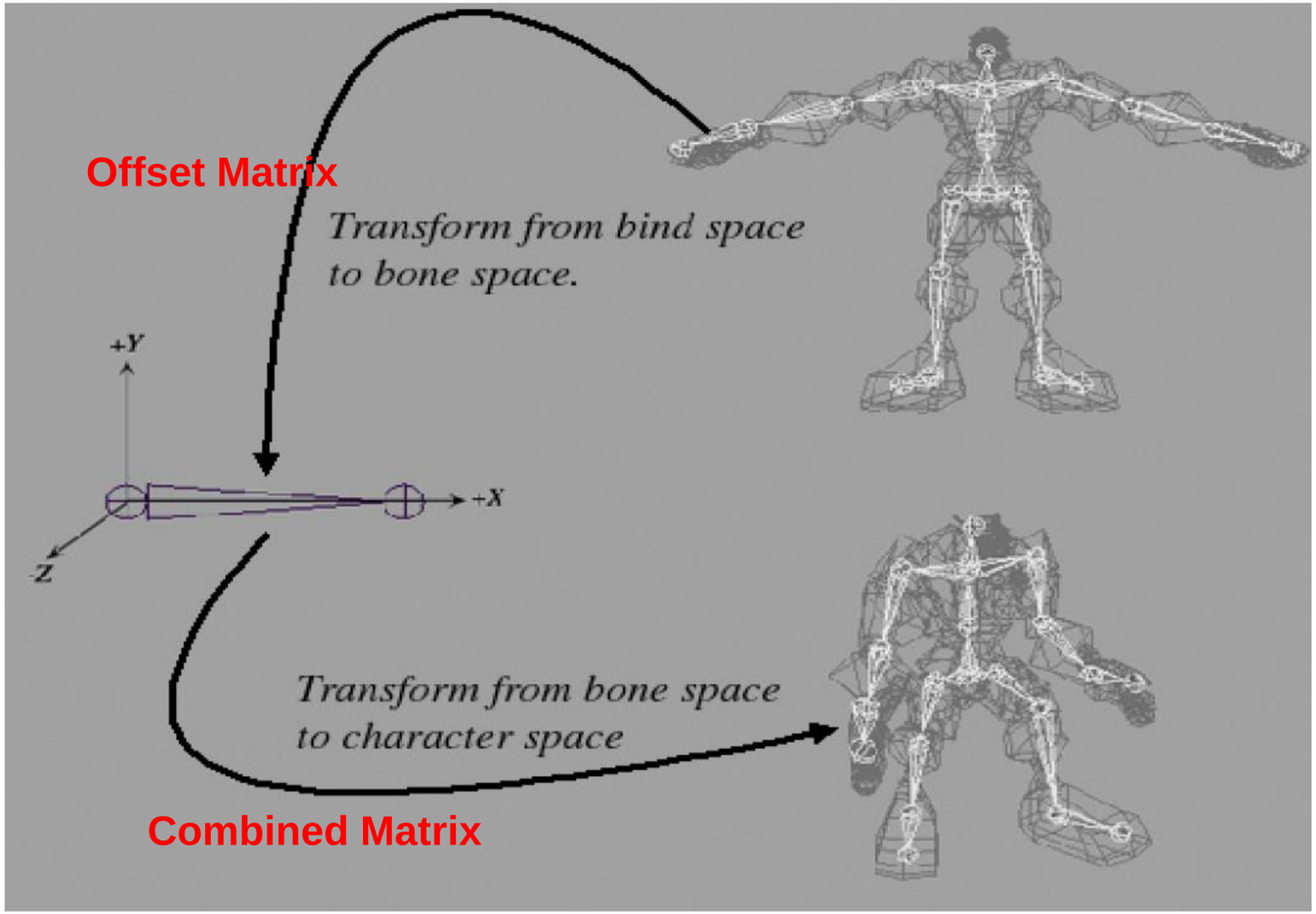
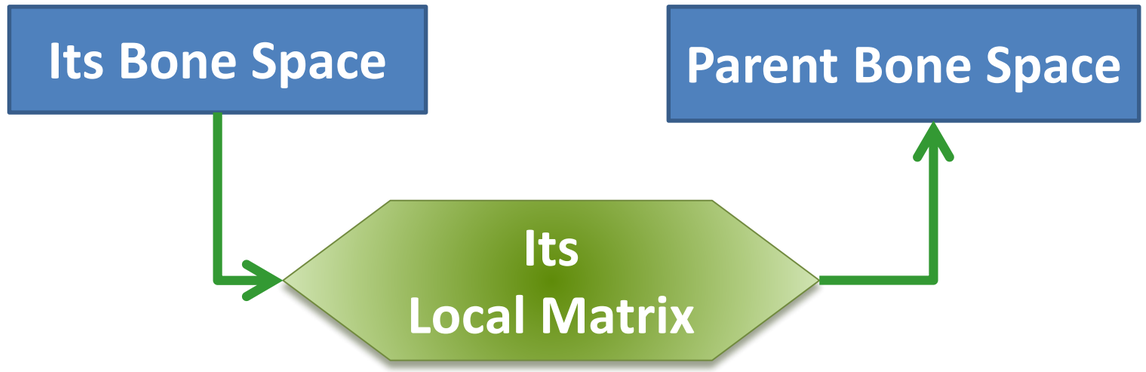
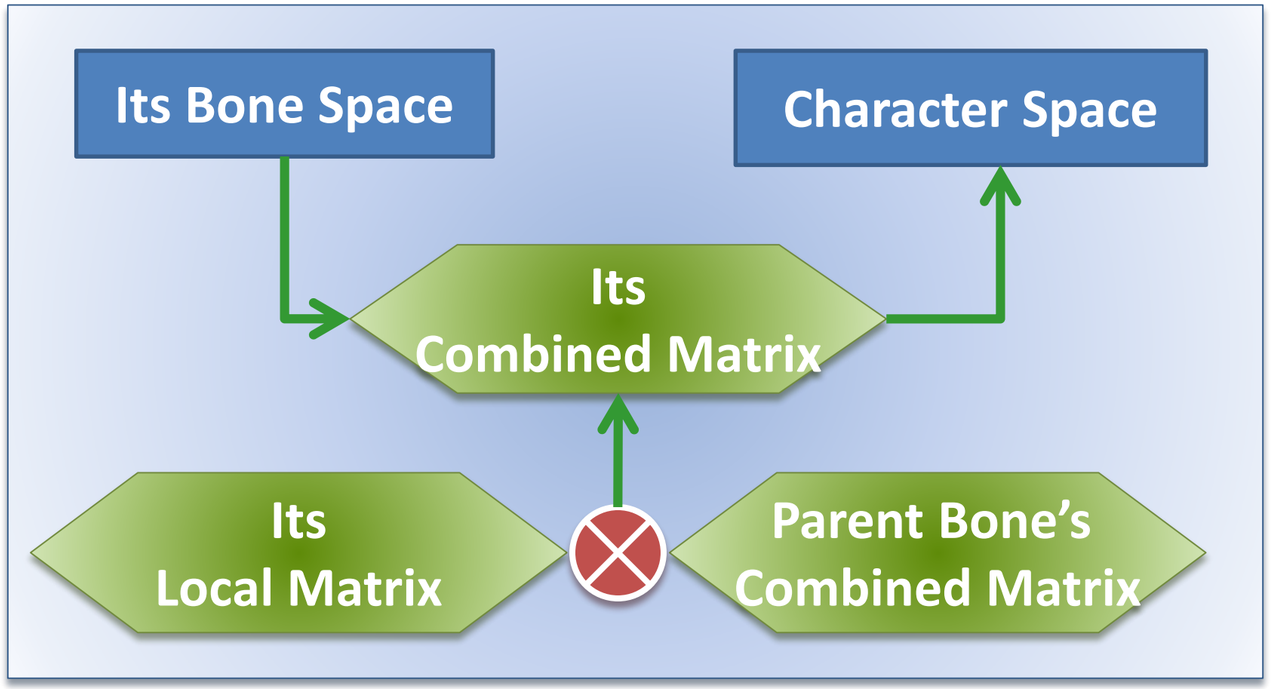


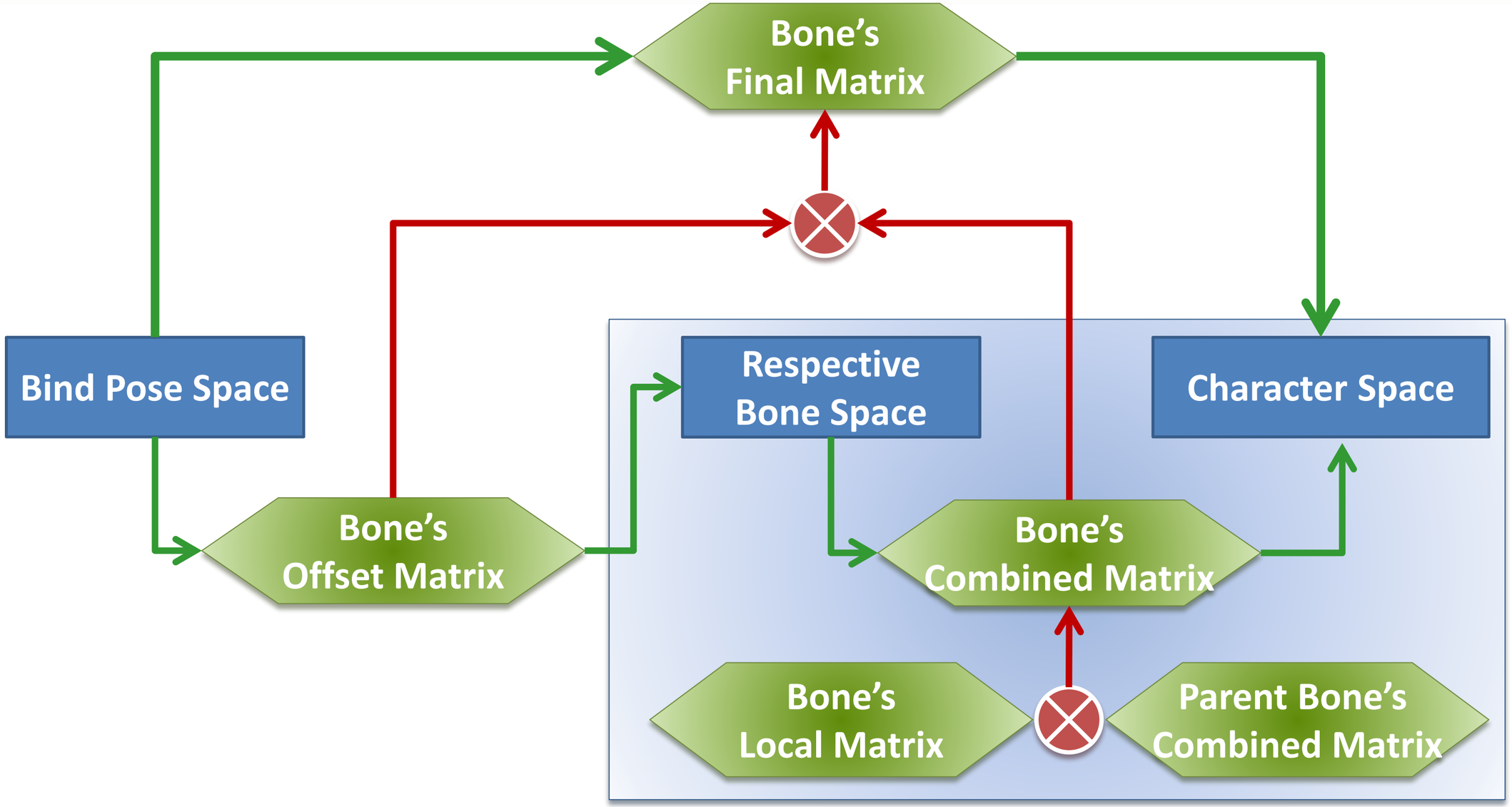
Figure 11: Transformation summary.

小结 – 骨骼的变换


Its Bone Space
Parent Bone Space
Its
Local Matrix


Its Bone Space
Character Space
Its
Combined Matrix
Its
Local Matrix
Parent Bone's
Combined Matrix

小结 – Skin Mesh顶点的变换

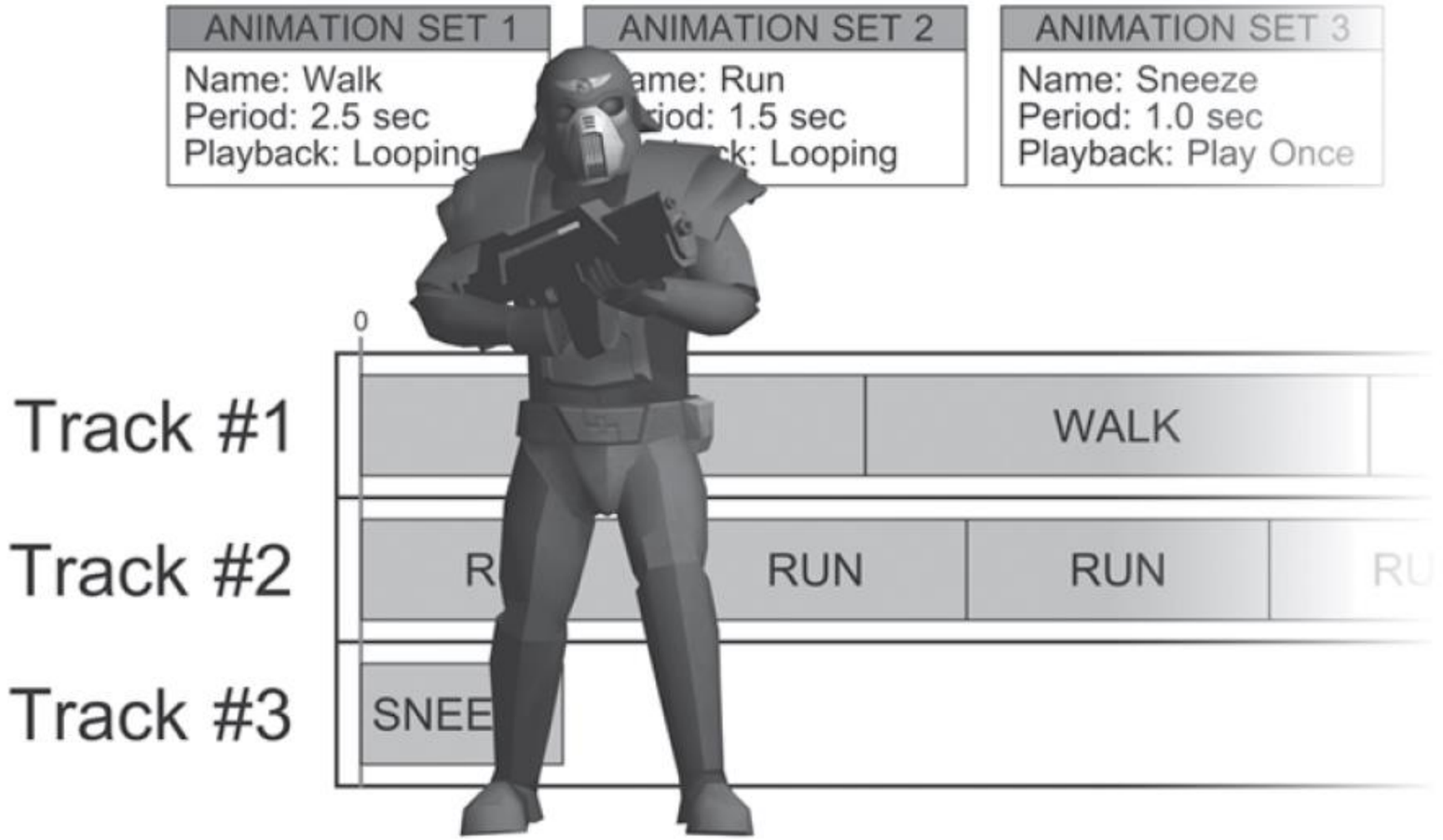


Notes of Interpolation

- We ***never interpolate matrices*** because matrix rotations do not interpolate correctly—we must use quaternions to interpolate rotations correctly.
  - Therefore, keyframe transformation data is usually always stored by a *Rotation Quaternion*, *Scaling Vector*, and *Translation Vector* (*RST-values*), and not as a matrix.
  - After interpolating the RST-values (i.e., interpolating rotation quaternions, scaling vectors, and translation vectors) for all bones, we can proceed to build the bone-matrix for each bone out of the interpolated RST-values.
  - Lastly, in the case a file format does present keyframe data as a matrix, we must decompose the matrix into RST-values in order to perform the interpolation.

Animation Set

- An **Animation** is a collection of transformation data (i.e. RTS-values) of all the bones of a certain of key frame for a distinct Animation Sequence.
- **Animation Set** are simply collections of Animations.



Animation Track

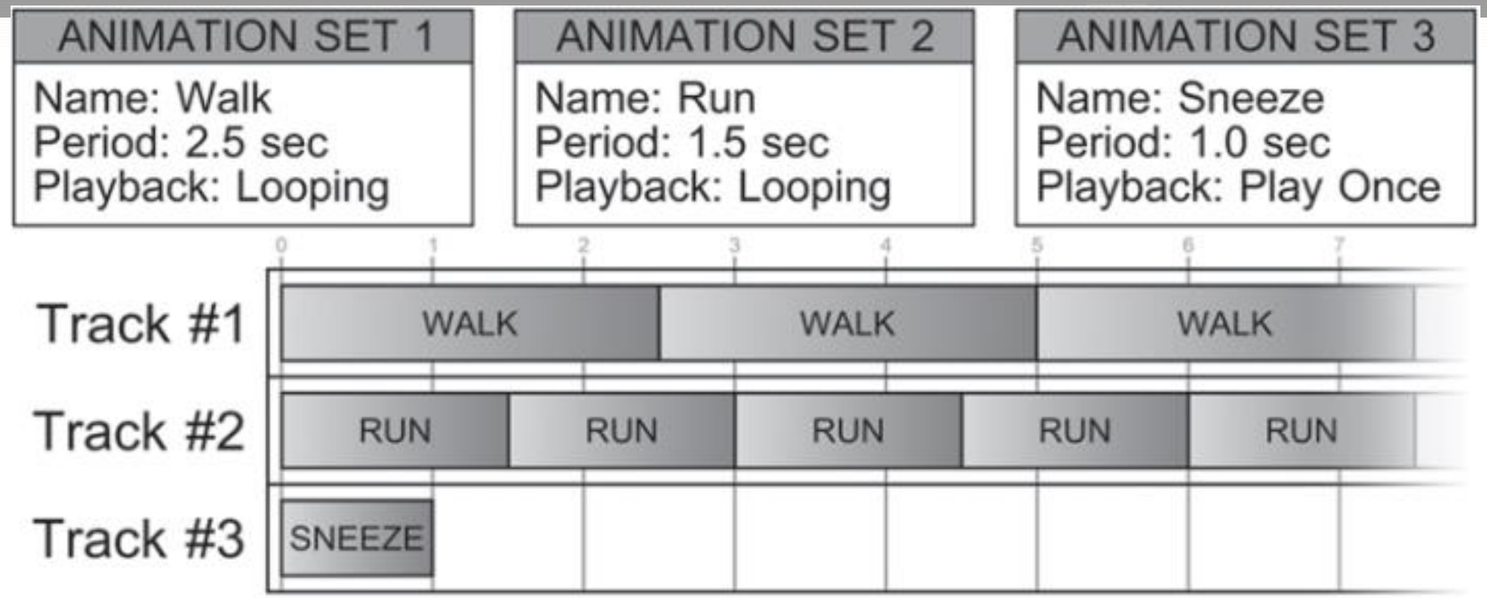


FIGURE 5.2
Three animation sets assigned to a separate animation track.

- The same animation set also can be assigned to more than one track:

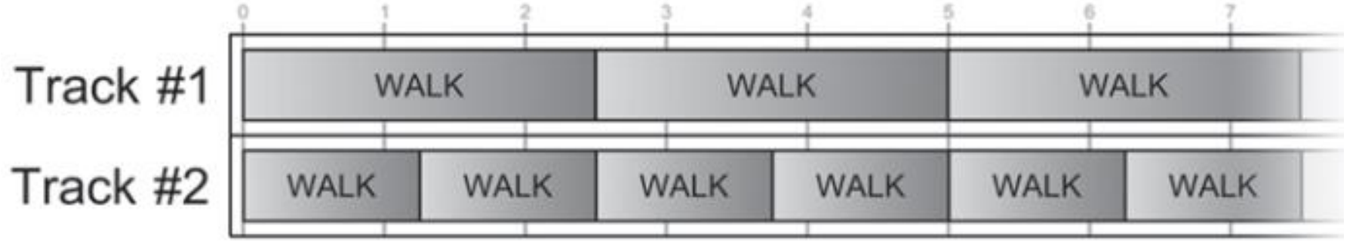


FIGURE 5.3
The track's speed property affects the animation playback.

List of properties for manipulating track

Property	Function	Description
Animation Set	SetTrackAnimationSet()	A pointer to an animation set
Enabled	SetTrackEnable()	Enables/disables the track
Position	SetTrackPosition()	Track position (time) of the animation set
Weight	SetTrackWeight()	The weight of the track (used when blending animations)
Speed	SetTrackSpeed()	The speed of the track
Priority	SetTrackPriority()	The priority of the track (can be set to low or high)

Animation Blending

- **Animation Blending**: Allows you to take two existing animation sequences and mix them together to create a new sequence.
- How to blend?
  1) Adding animation sets to different tracks
  2) Set properties of tracks: Speed, Weight, Priority
     - Weight: Specify how much weight each track contributes to the final blended animation.
     - Priority: During the blending, all high priority tracks will be blended separately and all low priority tracks will be blended separately. Then the result of these two separate blends will be blended together to produce the final blended animation.
       - This could also be used to turn off low-priority tracks when a character is far away from the player/camera.
  3) Enable tracks

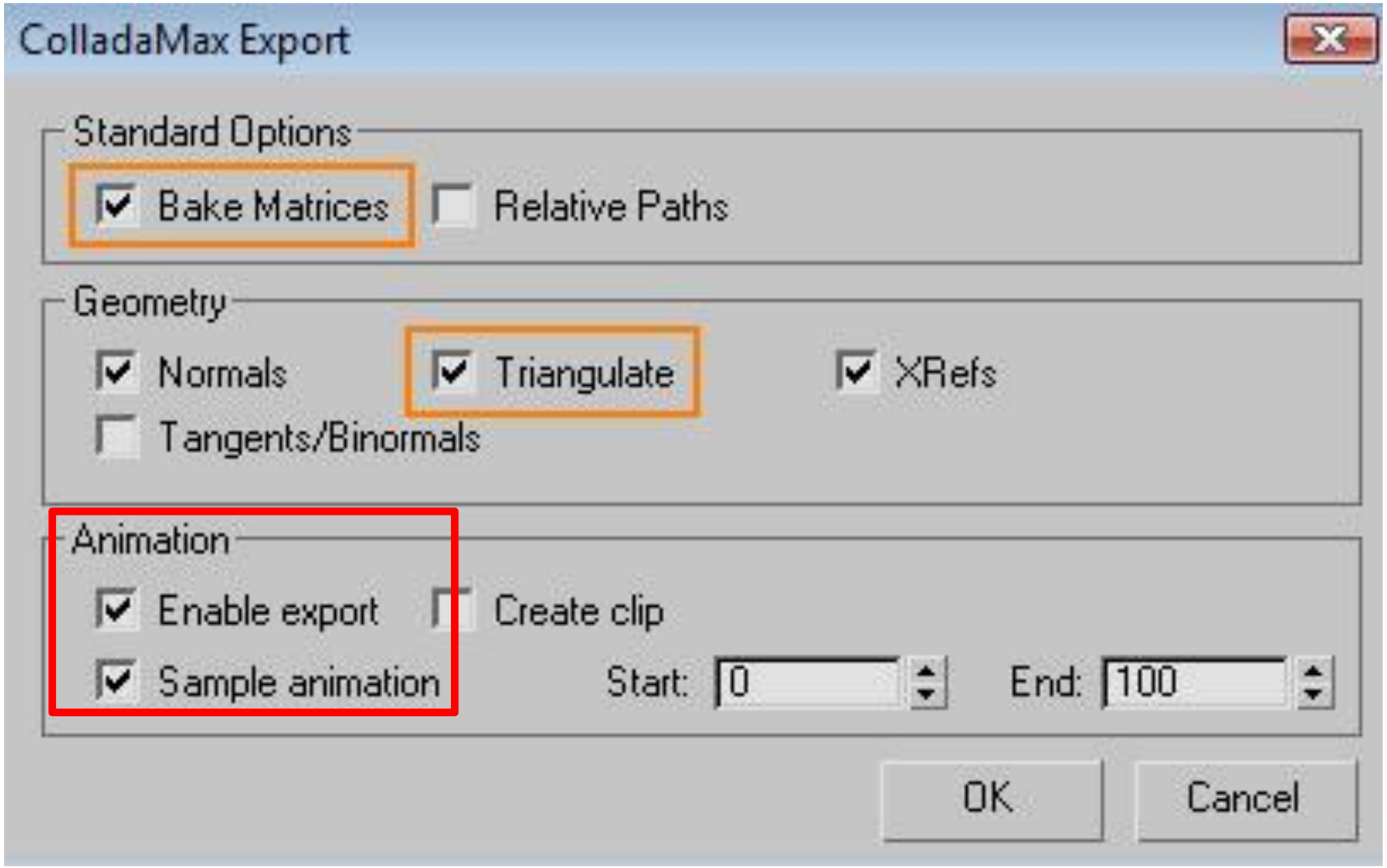
Animation Callback Events

- Animation callbacks are events that are synchronized with your animations.
  - One example might be playing the sound of a footstep.

小结 – 骨骼蒙皮动画需要的信息

1) Skin Mesh
2) Bone Hierarchy and Bones:
   - Bone Hierarchy
   - Local Matrix for Bone from bone space to parent bone space
   - Offset Matrix for Mesh’s Vertex transformation
3) Skinning Information:
   - How Vertices of Skinned Mesh are influenced by several bones.
   - and the Influence Weights
4) Multiple Animation Sets:
   - Animation’s Transformation Data for each bones of each keyframes.

DAE模型格式分析 – 3dsMax导出插件



DAE模型格式分析 – 整体框架

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema" version="1.4.1">
  <asset>
  </asset>
  <library_animations>
  </library_animations>
  <library_lights>
  </library_lights>
  <library_images>
  </library_images>
  <library_materials>
  </library_materials>
  <library_effects>
  </library_effects>
  <library_geometries>
  </library_geometries>
  <library_controllers>
  </library_controllers>
  <library_visual_scenes>
  </library_visual_scenes>
  <scene>
  </scene>
</COLLADA>
```

骨骼蒙皮动画所需的信息

1) Skin Mesh：
   - 存放在<library_geometries>节点中 → 材质信息<library_materials>, <library_effects>节点中 → Texture信息<library_images>节点中
2) Bone Hierarchy and Bones：
   - 存放在<library_visual_scenes>节点中。
3) Skinning Information：
   - 存放在<library_controllers>节点中，i.e. 哪些顶点被哪些骨骼节点作用，权重是多少等。
4) Animation Set：
   - 存放在<library_animations>节点中，如果有剪辑将被存放在<library_animation_clips>节点中。

Local Matrix in DAE

- <library_visual_scenes>节点中：

```
<node id="astroBoy_newSkeleton_root" name="root" sid="Bone1" type="JOINT">
  <matrix sid="transform">0.079633 -0.996824 0 0 0.992539 0.079291 -0.092624 0.057089 0
  <node id="astroBoy_newSkeleton_spine01" name="spine01" sid="Bone2" type="JOINT">
    <matrix sid="transform">-0.915962 -0.140877 0.375723 0 0.152015 -0.988378 0 0 0.371
    <node id="astroBoy_newSkeleton_spine02" name="spine02" sid="Bone3" type="JOINT">
      <matrix sid="transform">0.9933 0.067026 0.094141 0 -0.058502 0.994172 -0.090553 0
      <node id="astroBoy_newSkeleton_neck01" name="neck01" sid="Bone4" type="JOINT">
        <matrix sid="transform">0.94438 -0.00004 0.328855 0 -0.042957 0.991416 0.123484
        <node id="astroBoy_newSkeleton_head" name="head" sid="Bone5" type="JOINT">
          <matrix sid="transform">0.996675 -0.080362 0.013433 0 0.076472 0.979545 0.186
          <node id="astroBoy_newSkeleton_headEnd" name="headEnd" type="NODE">
            <matrix>0 -1 0 0 1 0 0 0 0 0 1 0.99098 0 0 0 1</matrix>
```

Offset Matrix in DAE

- <library_controlls>节点中：
  - For Bind Pose of the whole Bone Hierarchy:

```
<controller id="boyShape-skin-skin">
  <skin source="#boyShape-skin">
    <bind_shape_matrix>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</bind_shape_matrix>
```

  - For Bind Pose of individual Bone:

```
<joints>
  <input semantic="JOINT" source="#boyShape-skin-skin-joints"/>
  <input semantic="INV_BIND_MATRIX" source="#boyShape-skin-skin-bind_poses"/>
</joints>
<source id="boyShape-skin-skin-bind_poses">
  <float_array id="boyShape-skin-skin-bind_poses-array" count="704">0 0.995701 0.092623 -0.3
  <technique_common>
    <accessor source="#boyShape-skin-skin-bind_poses-array" count="44" stride="16">
      <param name="TRANSFORM" type="float4x4"/>
    </accessor>
  </technique_common>
</source>
```

Bone's Influence Weight in DAE

- <library_controlls>节点中：
  - For Influence Weight of Bone:

```
<vertex_weights count="2619">
  <input semantic="JOINT" source="#boyShape-skin-skin-joints" offset="0"/>
  <input semantic="WEIGHT" source="#boyShape-skin-skin-weights" offset="1"/>
  <vcount>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  <v>33 0 33 0 33 0 33 0 33 0 33 0 33 0 33 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
</vertex_weights>
```

  - Simplified example:

```
<vertex_weights count="4">
    <input semantic="JOINT" source="#joints" offset="0"/>
    <input semantic="WEIGHT" source="#weights" offset="1"/>
    <vcount>3 2 2 3</vcount>
    <v>
        1 0 0 1 1 2
        1 3 1 4
        1 3 2 4
        1 0 3 1 2 2
    </v>
</vertex_weights>
```

Animation Sets in DAE

- There can be multiple Animation Sets in <library_animations>, and one <animation> defines one Animation Set.

```
<library_animations>
  <animation id="astroBoy_newSkeleton_root-transform">
  </animation>
  <animation id="blendParent1">
  </animation>
  <animation id="astroBoy_newSkeleton_spine01-transform">
  </animation>
  <animation id="astroBoy_newSkeleton_spine02-transform">
```

- Two forms of <channel> definition:
  - Either
    - **<channel source=**"#astroSkeleton-sampler" **target=**"astroBoy_Skeleton/transform(Column)(Row)"/>
  - Or
    - **<channel source=**"#astroSkeleton-sampler" **target=**"astroBoy_Skeleton/transform"/>

Avatar换装实现方案

- <library_geometries> <geometry> <mesh> <triangles>会有多个的原因是：
  - 对triangle进行grouping，可以支持一个Mesh采用多种不同的材质（Texture，光照参数等）。
  - 不同的<triangles> group(i.e. 不同的人体组件)可采用不同的材质。
- Tip：一个Mesh对triangle进行grouping、且采用多种不同的材质，这能是实现Avatar换装的一种方案。Stage3D实现起来有些麻烦：
  - 采用一遍渲染的话，Stage 3D AGAL 1.0的纹理采样寄存器fs只有8个，意味着角色可换装的组件数最多只能8个。
    - 注：AGAL 2.0的fs寄存器已扩展至16个。
  - 或，采用多遍渲染的实现，只使用一个fs寄存器，角色的每个组件单独渲染，性能不好。