Topic 22 – 3D Modeling: Skinning

3D Modeling:

Skinning (& Quaternions)

•CITS3003 Graphics & Animation

Thanks to both Richard McKenna and Marco Gillies for permission to use their slides as a base. •1

Topic 22 – 3D Modeling: Skinning

Jack Skellington, from *The Nightmare Before Christmas*

Motivation – non-rigid transformations

- Hierarchies of transformations allow parts of complex objects to move relative to each other
- All vertices in a basic part are transformed the same way
  - Basic parts retain their shape
  - We generally have *rigid transformations*
- This works well for objects like tanks with rigid parts
- But, what about human characters and other objects that have parts that rotate and move, but with surfaces that are flexible rather than rigid?

What is skinning?

- Provides a means to animate a model while reducing the "folding" and "creasing" of polygons as the model animates
- Sometimes called skeletal animation
- A model is represented in two parts:
  - skin
  - skeleton

Topic 22 – 3D Modeling: Skinning

How is skinning implemented?

- Define a bone hierarchy in a model
  - separate from the mesh (triangle) data
  - is linked to the model's mesh, so that animating the bone hierarchy animates the mesh's vertices
- Multiple bones affect each vertex in the skin mesh
- Each bone has a weighting value to determine how much influence it has in proportion to the others
  - This allows smooth transitions around bone joints

Topic 22 – 3D Modeling: Skinning

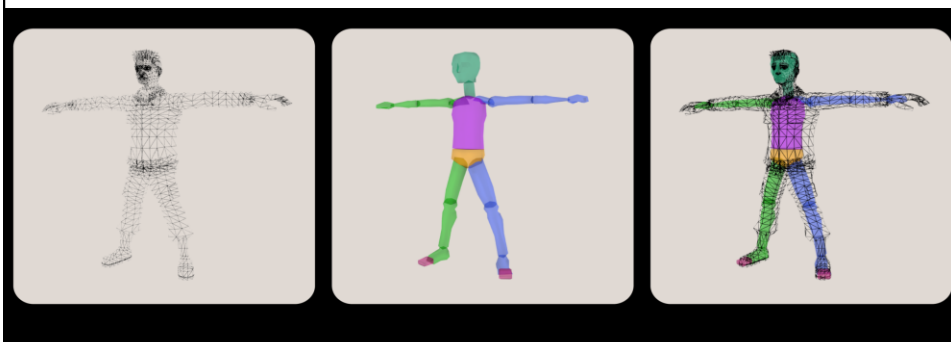
How are bones moved?

- Bone transformations
  - each bone has a 4 × 4 matrix that specifies how it transforms the points and vectors it influences for the current pose
  - the transformation is relative to a ***rest pose***
- Transforming a bone has what effect?
  - transforms all children bones (and their children, etc.)
  - transforms corresponding vertices following the bone weights
  - transforms normal vectors similarly (more complicated if there is non-uniform scaling)
- So, to calculate the position and normal for a vertex:
  - the bone transformations affecting the vertex are averaged according to the bone weights, then the transformation is applied to both the position and normal

Topic 22 – 3D Modeling: Skinning

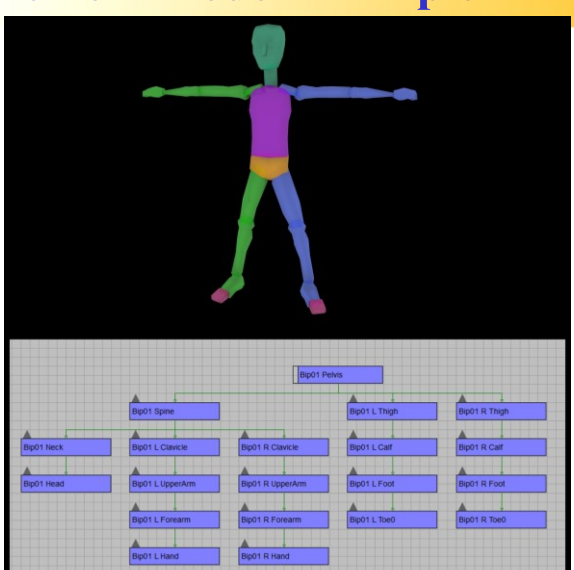
Hierarchical Model

- Associate bones with vertices



Topic 22 – 3D Modeling: Skinning

Hierarchical Model Example



Limitations of skinning

- Doesn’t provide realistic muscle movement
  - Ex: a character flexing an arm muscle
  - historically more important for movies than games
  - games are starting to use muscle controllers as well
    - attached to bones, mimic muscle movements

Skinned Data

- Necessary data to define relationship between bone hierarchy and the skin:
  - Number of vertices
  - Number of bones
  - Vertex Weights
  - Vertex Indices
  - Inverse Matrices

- Values are extracted from the modeler when loading an animation

Topic 22 – 3D Modeling: Skinning

Inverse Bone Matrices (InvBMs)

- Used to transform the skin data from the rest pose object space (also called ***skin space***) to ***bone space***, i.e., to coordinates relative to the position and orientation of a particular bone.
- An InvBM is the inverse of the transformation matrix from the ***root bone*** in the hierarchy to a particular bone in the hierarchy in the rest/default pose.
- In the project these are provided by the **Open Asset Importer** as:

```
bone->mOffsetMatrix
```

Topic 22 – 3D Modeling: Skinning

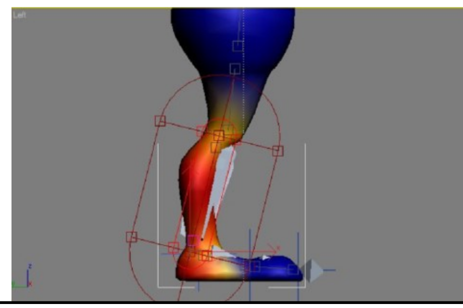
High-level Skinning Algorithm

1. Transforms vertex positions from their object space for the rest pose (a.k.a., skin space) into bone space using the InvBMs.
2. Performs the key-frame based animation for that frame, which transform the vertex positions back into skin space in its animated pose.
3. The model is then rendered.

Topic 22 – 3D Modeling: Skinning

Smooth Skinning & Bone Weights

- To make deformations appear smooth, we need to average between the effects of different bones close to joints.
- Bone weights can be set using modelling software. They are displayed using colours in Blender in *weight paint mode*: red is for close to 1, blue for close to 0.
- There are a number of standard ways of setting weights.
  - **Automatic geometric weighting** (as in the Project)
  - **Manual *weight-painting*** (Often used for tweaking)
  - ***Bone Envelopes*** which specify a distance that each bone affects.



Topic 22 – 3D Modeling: Skinning

Smooth Skinning On a GPU

- Smooth Skinning is well suited to implementing as a vertex shader
- You deform vertices one by one based on the joint transforms
  - Use uniform variables for the transformations of the bones in the current pose and change them each time the model is drawn.
- You need to get the bone weights for each vertex to the GPU
  - Use attributes, i.e., vertex shader “in” variables
  - These don’t change after the model has been loaded.
  - For the best performance, normally only a small number of bones are allowed to affect each vertex.

Topic 22 – 3D Modeling: Skinning

Skinning – GLSL vertex shader

```
// in variables set for each vertex from the static mesh data
in  ivec4 BoneIDs;      // Integer IDs of the 4 bones affecting this vertex
in   vec4 BoneWeights;  // The corresponding 4 weights between 0.0 and 1.0

const int MAX_BONES = 32; // Adjust based on expected model complexity

// the current bone transformations, set each time the model is drawn
uniform mat4 BoneTransforms[MAX_BONES];

void main()  {
    // Calculate a weighted average of the given 4 bones transformations
    mat4 BoneTransform =  BoneTransforms[BoneIDs[0]] * BoneWeights[0]
                        + BoneTransforms[BoneIDs[1]] * BoneWeights[1]
                        + … … …

    vec4 btPosition = BoneTransform * vPosition;     // Transform the position
    vec4 btNormal = BoneTransform * vNormal;         // and the normal
    // The rest of the vertex shader should use btPosition and btNormal
}
```

Topic 22 – 3D Modeling: Skinning

Skinning – C++ initialisation

```
for (unsigned int boneID = 0 ; boneID < mesh->mNumBones ; boneID++) {
    for (unsigned int j = 0 ; j < mesh->mBones[boneID]->mNumWeights ; j++) {
        int VertexID = mesh->mBones[boneID]->mWeights[j].mVertexId;
        float Weight = mesh->mBones[boneID]->mWeights[j].mWeight;

        // Insertion sort, keeping the 4 largest weights
        for(int slotID=0; slotID < 4; slotID++) {
            if(boneWeights[VertexID][slotID] < Weight) {
               for(int shuff=3; shuff>slotID; shuff--) {
                 boneWeights[VertexID][shuff] = boneWeights[VertexID][shuff-1];
                 boneIDs[VertexID][shuff] = boneIDs[VertexID][shuff-1];
               }
               boneWeights[VertexID][slotID] = Weight;
               boneIDs[VertexID][slotID] = boneID;
               break;
            }
        }
    }
}
```

Topic 22 – 3D Modeling: Skinning

Skinning – C++ bone transforms

```
// rotIndex is for the rotation key just before or at currentTime
if(rotIndex+1 == channel-> mNumRotationKeys)
    curRotation = channel->mRotationKeys[rotIndex].mValue;
else {
    float t0 = channel->mRotationKeys[rotIndex].mTime;
    float t1 = channel->mRotationKeys[rotIndex+1].mTime;
    float wgt1 = (currentTime-t0)/(t1-t0);

    // Interpolate using quaternions
    aiQuaternion::Interpolate(curRotation,
                              channel->mRotationKeys[rotIndex].mValue,
                              channel->mRotationKeys[rotIndex+1].mValue, wgt1);
    curRotation = curRotation.Normalize();
}

// now build a transformation matrix from it. First rotation, then position.
aiMatrix4x4 trafo = aiMatrix4x4(curRotation.GetMatrix());
trafo.a4 = curPosition.x; trafo.b4 = curPosition.y; trafo.c4 = curPosition.z;

targetNode->mTransformation = trafo; // assign this transformation to the node
```

Topic 22 – 3D Modeling: Skinning

C++ bone transform composition

```
mat4 boneTransforms[mesh->mNumBones];

for(unsigned int a=0; a<mesh->mNumBones; a++) {
    const aiBone* bone = mesh->mBones[a];

    // find the node, looking recursively through the hierarchy for the name
    aiNode* node = scene->mRootNode->FindNode(bone->mName);

    aiMatrix4x4 b = bone->mOffsetMatrix;  // start with the mesh-to-bone matrix
    // then add the bone’s transformation for the current pose by composing all
    // nodes pose transformations up the parent chain
    const aiNode* tempNode = node;
    while( tempNode) {
        b = tempNode->mTransformation * b;
        tempNode = tempNode->mParent;
    }
    boneTransforms[a] = mat4(vec4(b.a1, b.a2, b.a3, b.a4), vec4(…), …  );
}
glUniformMatrix4fv(uBoneTransforms, mesh->mNumBones, GL_TRUE,
                                    (const GLfloat *)boneTransforms );
```

Problems With Euler Angles

- Bones are positioned and rotated in modeling programs, often using angles
  - pitch, yaw, and roll are *Euler angles*, which is generally how they are manipulated when creating key frames (e.g., in Blender)
- We need to interpolate interim bone transforms
- Euler angles are problematic when interpolating
  - Can produce unexpected results for rotations that require combinations of the three angles
  - **Gimbal lock** – the second transform may align the first and third, leading to 2 degrees of freedom rather than 3

Quaternions

- An extension of complex numbers
- An alternative for interpolation
  - don't suffer from Gimbal Lock
- A rotation can be represented:
  - using 3 Euler Angles
  - using 4 Quaternion Components
    - 1 real (a scalar that roughly represents "non-rotation")
    - 3 imaginary (determine a 3D vector that we rotate around, and the amount that we rotate)

Topic 22 – 3D Modeling: Skinning

Quaternion Advantages

- Interpolation is easy between 2 quaternions
- Smoother animation
- Take up less space than matrices
- Can easily be converted to matrices for rendering
- Converting to quaternions is less efficient, but this can be done just once when exporting from modelling software.

Topic 22 – 3D Modeling: Skinning

But what *is* a quaternion?

- A quadruple of real numbers

$$Q = (q_x, q_y, q_z, q_w)$$
$$= q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k} + q\mathbf{w}$$

where $i^2 = j^2 = k^2 = -1$

Alternatively denoted as $[w, x, y, z]$

Or $[s, \mathbf{v}]$

where s is the scalar w

$\mathbf{v}$ is the 3D vector (x, y, z)

Topic 22 – 3D Modeling: Skinning

Quaternion Addition

- All quaternions for a 3D rotation have unit length
- Adding and subtracting quaternions is easy
- Example:
  - $\mathbf{p} = [w_1, x_1, y_1, z_1]$
  - $\mathbf{q} = [w_2, x_2, y_2, z_2]$
  - $\mathbf{p} + \mathbf{q} = [w_1 + w_2, x_1 + x_2, y_1 + y_2, z_1 + z_2]$

Topic 22 – 3D Modeling: Skinning

Quaternion Multiplication

- Same effect as multiplying two rotation matrices:
  - the rotations will be added together.
  - Example:
    - $\mathbf{p} = [s_1, \mathbf{v}_1]$
    - $\mathbf{q} = [s_2, \mathbf{v_2}]$

$$\mathbf{p} \cdot \mathbf{q} = [s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$$

Here $\times$ is cross product, and $\cdot$ is dot product.

Topic 22 – 3D Modeling: Skinning

Quaternion Conversions

- What is a quaternion anyway?
  - Artists can't relate
- Plus, OpenGL doesn't support them directly
- So what do we do?
  - do all rotation interpolation between frames using quaternions
  - convert results to matrices - to pass as the bone transformations
  - the "blending" between bone transformations in the vertex shader uses matrices instead of quaternions (quaternions can only interpolate rotations that are around the same centre)
  - [*Dual quaternions* have recently been used for better blending by representing combined rotations and translations. See: http://isg.cs.tcd.ie/projects/DualQuaternions/]

Topic 22 – 3D Modeling: Skinning

Conversion Calculations

- Euler angles to quaternions:

$$Q_x = [\cos(\text{pitch}/2), \sin(\text{pitch}/2), 0, 0]$$
$$Q_y = [\cos(\text{yaw}/2), 0, \sin(\text{yaw}/2), 0]$$
$$Q_z = [\cos(\text{roll}/2), 0, 0, \sin(\text{roll}/2)]$$

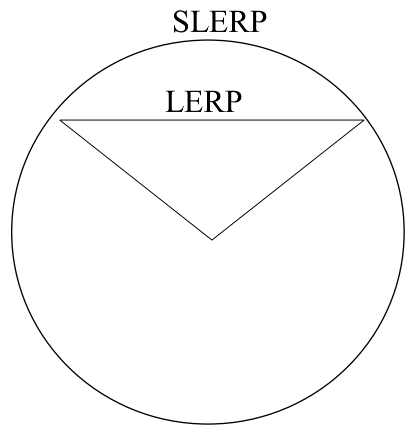
- Quaternion (x, y, z, w) to 3×3 matrix:

$$\begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

Topic 22 – 3D Modeling: Skinning

LERPs AND SLERPs

- 2 Ways of Interpolating Quaternions
  - LERP
    - Linear Interpolation
  - SLERP
    - Spherical Linear Interpolation



Topic 22 – 3D Modeling: Skinning

Interpolating

LERP: $(\mathbf{q}_0, \mathbf{q}_1, t) = (1 - t)\mathbf{q}_0 + t\mathbf{q}_1$

SLERP:

$$(\mathbf{q}_0, \mathbf{q}_1, t) = (\sin((1 - t)\theta))\mathbf{q}_0 / (\sin\theta) + (\sin(t\theta))\mathbf{q}_1 / (\sin\theta)$$

where $\theta = \arccos(\mathbf{q}_0 \cdot \mathbf{q}_1)$

References

- Wikipedia:
  - http://en.wikipedia.org/wiki/Skeletal_animation
  - http://en.wikipedia.org/wiki/Quaternion
    http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
    http://en.wikipedia.org/wiki/Gimbal_lock