Mesh Skinning

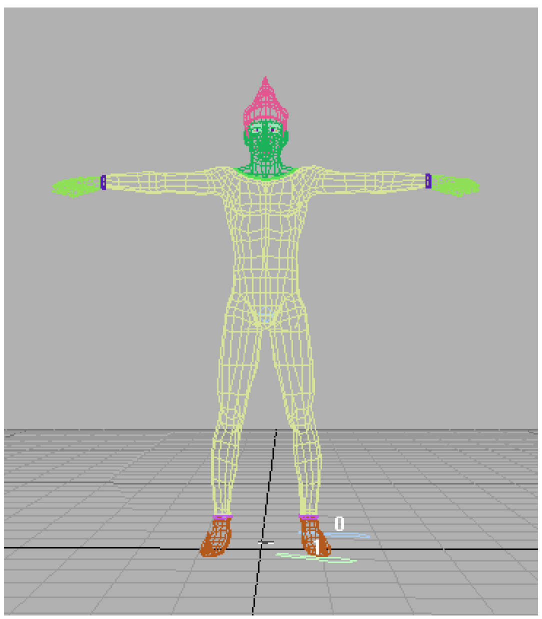
Sébastien Dominé

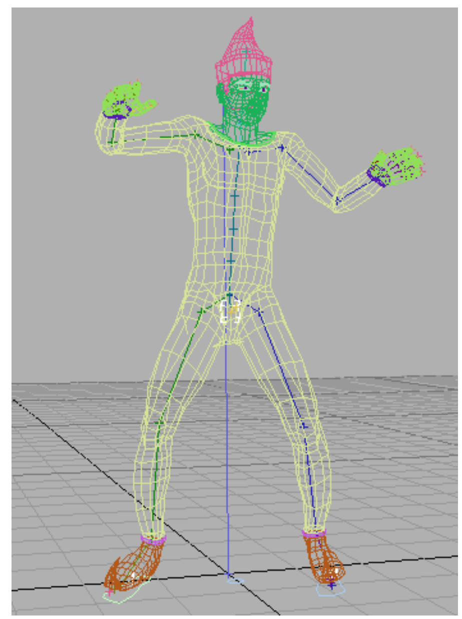
Agenda

- Introduction to Mesh Skinning
- 2 matrix skinning
- 4 matrix skinning with lighting
- Complex skinning for character modeling

Introduction to Mesh Skinning

- Allows a mesh to be deformed based on an underlying transformation matrix set.
- Usually thought of as a skin being deformed by a skeleton





Mathematics of mesh skinning 1/3

$$v' = \sum_{i}^{n} w_i M_i v \quad \textbf{with} \quad \sum_{i} w_i = 1$$

Where:

- n **is the number of matrices.**
- v **is the vertex position.**
- w_i **is the weight associated.**
- M_i **is the transformation matrix.**

Mathematics of mesh skinning 2/3

- **For the normals:**

$$n' = \sum_{i}^{N} w_i M_i^{-1^T} n \quad \textbf{with} \quad \sum_i w_i = 1$$

Where:

N **is the number of matrices.**

n **is the vertex normal.**

w_i **is the weight associated.**

$M_i^{-1^T}$ **is the inverse transpose of transformation matrix** M_i **.**

Mathematics of mesh skinning 3/3

- **Tangent basis computation:**
  - **Use the algorithm for the normal, and instead compute the skinned bi-normal and tangent.**
  - **Do a cross product of the skinned bi-normal and tangent to obtain the normal (Cheap only 2 op-codes to compute the cross product):**

```
MUL    R0,    R2.yzxw,    R1.zxyw;
MAD    R0,   -R2.zxyw,    R1.yzxw,    R0;
```

NVIDIA

Complex Skinning for Character modeling

- **2 methods:**
  1. **100% CPU free skinning method (Vertex Offset):**
     - **Consists of pre-computing vertices in bone's local space:**
       - **PROS:**
         - **CPU is not involved**
       - **CONS:**
         - **Consumes more bus bandwidth since we have to pass X times the vertices (where X is the number of bone reference per vertex)**
  2. **Light CPU usage method (Bone Offset):**
     - **Consists of pre-computing the bone's transform that moves a vertex (in model space) into bone's local space and to post-multiply the bone's matrices by it at runtime:**
       - **PROS:**
         - **About 4x less bandwidth usage (good for multiple characters)**
       - **CONS:**
         - **Uses a small amount of CPU to post-multiply the bone's matrices (again this could be pre-computed)**

NVIDIA

Complex Skinning for Character modeling – Vertex Offset Method

- **12 matrices per primitive (triangle)**
- **4 matrices per vertex**
- **28 matrices accessible at once**
  - **Use 4x3 affine transforms**
- **Needs 4 derivative of the same vertex program to process efficiently the vertices that are transformed by either 1,2,3 or 4 bones**
- **Needs to send vertices in bone's space, I.e. multiple versions of the same vertex, but each in the local bone space that the vertex is referencing**

Vertex Offset Method
Math. of complex skinning 1/2

$$v' = \sum_{i}^{n} w_i M_i v_i \quad \textbf{with} \quad \sum_{i} w_i = 1$$

Where:

- n **is the number of matrices**
- v_i **is the vertex position in** M_i **coordinate system.**
- w_i **is the weight associated.**
- M_i **is the transformation matrix (affine transform).**

Vertex Offset Method
Math. of complex skinning 2/2

- **For the normals:**

$$n' = \sum_{i}^{N} w_i R_i n_i \quad \textbf{with} \quad \sum_i w_i = 1$$

Where:

N **is the number of matrices.**

n_i **is the vertex normal in** M_i **coordinate system.**

w_i **is the weight associated.**

R_i **is the upper 3x3 matrix block of transformation matrix** M_i **. (I.e. just the rotation component of the affine transform)**

Vertex Offset Method
Data organization 1/3

- Bones are stored in the constant table:
  - 96 four dimensional vectors
  - 28x3 = 84 vectors used to store 28 affine transforms (i.e. translation + rotation)
- Vertex attributes (16 four dimensional attributes per vertex):
  - Vertex offsets (up to 4)
  - Vertex weights (up to 4)
  - Indices to constant table to get transforms (up to 4)

  - Normal offsets (up to 4)

  Or

  - Bi-normal offsets (up to 4)
  - Tangent offsets (up to 4)

Depending on the kind of lighting

Vertex Offset Method
Data organization 2/3

Vertex attributes / bone	Standard lighting	Per pixel lighting
Position	3 floats * 4	3 floats * 4
Normal	3 shorts * 4	
Weights	1 short * 4	1 short * 4
Binormal		3 shorts * 4
Tangent		3 shorts * 4
Indices	1 byte	1 byte
Total per skinned vertex	81 bytes	105 bytes

Plus texture coordinates – depends how many units are used.

Vertex Offset Method
Data organization 3/3

- Preprocess the model to batch up groups of faces that are using vertices that are using either 1,2,3 or 4 bones. This way you know when to pick the optimal vertex program to render the group of faces.
- If you have more than 28 bones, preprocess the model to break it up in groups of faces that share the same bones.

Vertex Offset Method
Vertex Program 1/5

- **Source code:**

```
char four_bone_normal_offsets_textured[] =
    "!!VP1.0 # Four bone transform \n"
    // c[0]...c[3]    contains modelview projection composite matrix
    // c[4]           contains constants: c[4].x = 2.0; c[4].y = 1.0; c[4].z = 0.0;
    // c[5]           contains (diffuse color) * Kd
    // c[6]           contains light position


    // c[8]...c[11]   contains bone one transform
    // c[12]...n      contains bone n transform

    // v[OPOS]        contains the transform indices

    // v[NRML]        contains normal offset and weight related to bone one
    // v[6]           contains normal offset and weight related to bone two
    // v[7]           contains normal offset and weight related to bone three
    // v[TEX3]        contains normal offset and weight related to bone four

    // v[TEX4]        contains vector offset
    // v[TEX5]        contains vector offset
    // v[TEX6]        contains vector offset
    // v[TEX7]        contains vector offset
```

NVIDIA

Vertex Offset Method
Vertex Program 2/5

```
// Load the matrix index for mat0
"ARL    A0.x,       v[OPOS].x;"
// We transform the offset by bone one's tranform
"DP4    R1.x,       c[A0.x + 8],    v[TEX4];"
"DP4    R1.y,       c[A0.x + 9],    v[TEX4];"
"DP4    R1.z,       c[A0.x + 10],   v[TEX4];"
// We multiply the transformed offset by the weight
"MUL    R1.xyz,     R1,             v[NRML].w;"

// We transform the normal offset by bone one's tranform
"DP3    R5.x,       c[A0.x + 8],    v[NRML];"
"DP3    R5.y,       c[A0.x + 9],    v[NRML];"
"DP3    R5.z,       c[A0.x + 10],   v[NRML];"
// We multiply the transformed normal offset by the weight
"MUL    R5.xyz,     R5,             v[NRML].w;"

// Load the matrix index for mat1
"ARL    A0.x,       v[OPOS].y;"
// We transform the offset by bone two's tranform
"DP4    R2.x,       c[A0.x + 8],    v[TEX5];"
"DP4    R2.y,       c[A0.x + 9],    v[TEX5];"
"DP4    R2.z,       c[A0.x + 10],   v[TEX5];"

// We multiply the transformed offset by the weight
"MAD    R1.xyz,     R2,             v[6].w,     R1;"
```

NVIDIA

Vertex Offset Method
Vertex Program 3/5

```
// We transform the normal offset by bone two's tranform
"DP3    R6.x,       c[A0.x + 8],    v[6];"
"DP3    R6.y,       c[A0.x + 9],    v[6];"
"DP3    R6.z,       c[A0.x + 10],   v[6];"
// We multiply the transformed normal offset by the weight
"MAD    R5.xyz,     R6,             v[6].w,     R5;"

// Load the matrix index for mat2
"ARL    A0.x,       v[OPOS].z;"
// We transform the offset by bone three's tranform
"DP4    R3.x,       c[A0.x + 8],    v[TEX6];"
"DP4    R3.y,       c[A0.x + 9],    v[TEX6];"
"DP4    R3.z,       c[A0.x + 10],   v[TEX6];"

// We multiply the transformed offset by the weight
"MAD    R1.xyz,     R3,             v[7].w,     R1;"

// We transform the normal offset by bone two's tranform
"DP3    R7.x,       c[A0.x + 8],    v[7];"
"DP3    R7.y,       c[A0.x + 9],    v[7];"
"DP3    R7.z,       c[A0.x + 10],   v[7];"
// We multiply the transformed normal offset by the weight
"MAD    R5.xyz,     R7,             v[7].w,     R5;"

// Load the matrix index for mat3
"ARL    A0.x,       v[OPOS].w;"
```

Vertex Offset Method
Vertex Program 4/5

```
// We transform the offset by bone four's tranform
"DP4    R4.x,      c[A0.x + 8],    v[TEX7];"
"DP4    R4.y,      c[A0.x + 9],    v[TEX7];"
"DP4    R4.z,      c[A0.x + 10],   v[TEX7];"

// We multiply the transformed offset by the weight
"MAD    R1.xyz,     R4,            v[TEX3].w,     R1;"

// We transform the normal offset by bone two's tranform
"DP3    R8.x,      c[A0.x + 8],    v[TEX3];"
"DP3    R8.y,      c[A0.x + 9],    v[TEX3];"
"DP3    R8.z,      c[A0.x + 10],   v[TEX3];"

// We multiply the transformed normal offset by the weight
"MAD    R5.xyz,     R8,            v[TEX3].w,     R5;"

// set the vertex w to 1.0
"SGE    R1.w,      R5,            R5;"

// normalize(R5) -> R2
"DP3    R3.w,      R5,            R5;"
"RSQ    R3.w,      R3.w;"
"MUL    R2.xyz,    R5,            R3.w;"
```

Vertex Offset Method
Vertex Program 5/5

```
// Still needs to be projected...
"DP4    o[HPOS].x,  c[0],          R1;"
"DP4    o[HPOS].y,  c[1],          R1;"
"DP4    o[HPOS].z,  c[2],          R1;"
"DP4    o[HPOS].w,  c[3],          R1;"

// light position DOT normal
"DP3    R3,         c[6],          R2;“

// Diffuse term * diffuse color
"MUL    o[COL0].xyz, R3,           c[5];"

// set the texcoord s and t
"MOV    o[TEX0].xy, v[TEX0];"
"END";
```

Complex Skinning for Character modeling – Bone Offset Method

- 12 matrices per primitive (triangle)
- 4 matrices per vertex
- 28 matrices accessible at once
  - Use 4x3 affine transforms
- Needs 4 derivative of the same vertex program to process efficiently the vertices that are transformed by either 1,2,3 or 4 bones
- Only pass vertices in model space (1 set of vertices is sent)

Bone Offset Method
Math. of complex skinning 1/2

$$v' = \sum_{i}^{n} w_i M_i M_{ref_i}^{-1} v \text{ with } \sum_i w_i = 1$$

Where:

- n **is the number of matrices**
- v **is the vertex position in model space of the reference posture.**
- w_i **is the weight associated.**
- M_i **is the transformation matrix (affine transform).**
- $M_{ref_i}^{-1}$ **is the inverse transform of the bone's reference posture transform (it transforms the vertex from model space into bone's local space)**

Bone Offset Method
Math. of complex skinning 2/2

- **For the normals:**

$$n' = \sum_{i}^{N} w_i R_i R_{ref_i}^{-1} n \quad \textbf{with} \quad \sum_i w_i = 1$$

Where:

N **is the number of matrices.**

n **is the vertex normal in model space of the reference posture.**

w_i **is the weight associated.**

R_i **is the upper 3x3 matrix block of transformation matrix (I.e. just the rotation component of the affine transform).**

$R_{ref_i}^{-1}$ **is the inverse rotation matrix of the bone's reference posture transform.**

nVIDIA.

Bone Offset Method
Data organization 1/3

- **Bones are stored in the constant table:**
  - **96 four dimensional vectors**
  - **28x3 = 84 vectors used to store 28 affine transforms (i.e. translation + rotation)**
- **Vertex attributes (16 four dimensional attributes per vertex):**
  - **Vertex position**
  - **Vertex weights (up to 4)**
  - **Indices to constant table to get transforms (up to 4)**

  - **Normal**

  Or

  - **Bi-normal**
  - **Tangent**

Depending on the kind of lighting

nVIDIA

Bone Offset Method
Data organization 2/3

Vertex attributes / bone	Standard lighting	Per pixel lighting
Position	3 floats	3 floats
Normal	3 shorts	
Weights	1 short * 4	1 short * 4
Binormal		3 shorts
Tangent		3 shorts
Indices	1 byte	1 byte
Total per skinned vertex	27 bytes	33 bytes

Plus texture coordinates – depends how many units are used.

Bone Offset Method
Data organization 3/3

- Preprocess the model to batch up groups of faces that are using vertices that are using either 1,2,3 or 4 bones. This way you know when to pick the optimal vertex program to render the group of faces.
- If you have more than 28 bones, preprocess the model to break it up in groups of faces that share the same bones.

Optimization 1/2

- Use:
  - In OpenGL, display lists to store the geometry on the GPU – in D3D, Vertex buffers:
    - Batch up all sub meshes
    - Update constant table for the motion capture playback
    - Draw the display list (buffers)
    - This is good for multiple instances of the same model
- Use Vertex Array Range / Fence if you vary the weights or other vertex attributes over time (bulging effects, etc…)

Optimization 2/2

- Use :
  - OpenGL: the texture shaders and/or register combiners to do the lighting whenever possible – it should save a few instructions.
  - D3D: Pixel shaders.
- Be careful with multipass rendering – the GPU has to process the vertices for each pass (no persistency of the processed data)
- Make the most use of the 4 texture units and the register combiners to avoid multipass rendering.
- Use appropriate data types to minimize data transfers (AGP 4x is 1066MB/s). The data gets converted to IEEE 32-bit (s23e8) floating point precision internally anyway.

Character skinning demo

- **85 bones**
- **Up to 4 matrices per vertex**
- **Source code in the OpenGL SDK:**
  - **OpenGL\src\demos\vtxprg_skin**

Questions, comments, feedback

- **Sébastien Dominé, sdomine@nvidia.com**
- **www.nvidia.com/developer**