Skeletal Animation

Advanced Graphics Programming

Objective

- To become familiar with the terminology of animated skinned meshes.
- To learn the mathematics of mesh hierarchy transformations and how to traverse treebased mesh hierarchies.
- To understand the idea and mathematics of vertex blending.
- To discover how to load an animated model and implement character animation

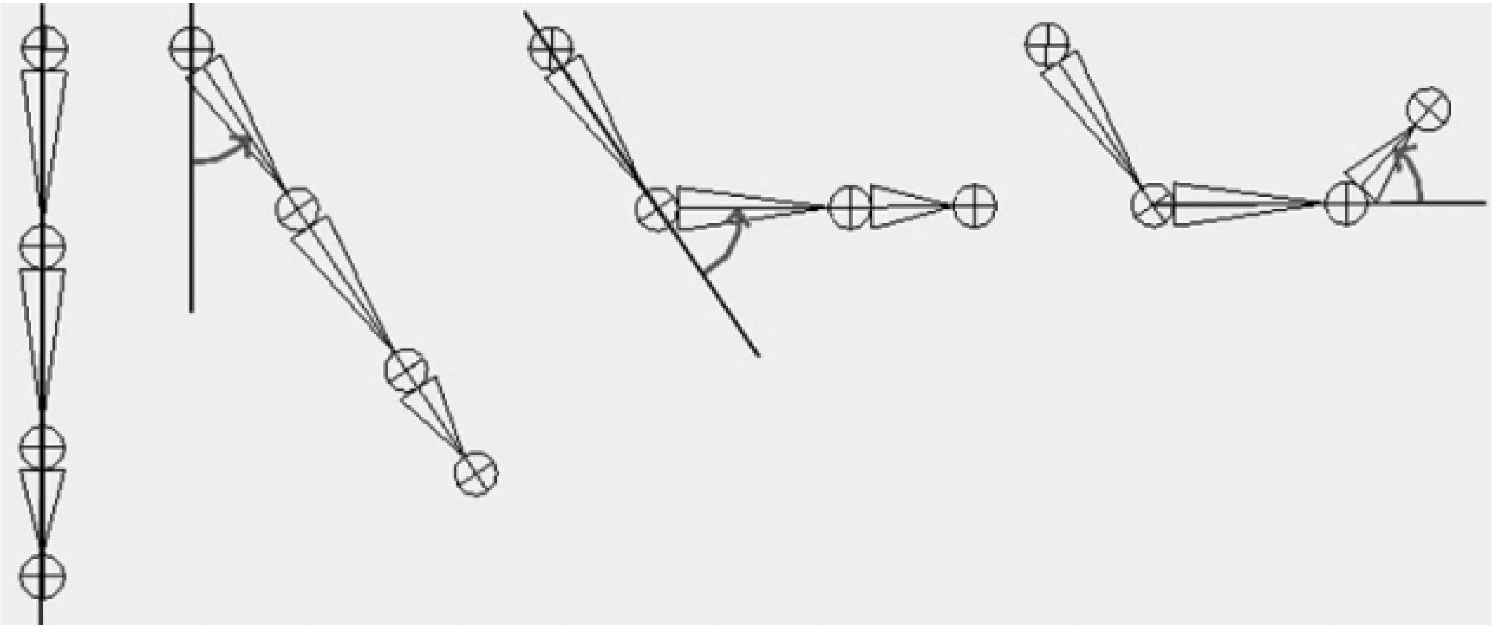
Overview

- We learn how to animate complex characters like a human or animal.
- Characters are complex because they have many moving parts that all move at the same time.
- Consider a human running—every bone is moving in some way.
- Creating such complicated animations is not practical by hand, and there are special modeling and animation tools for this task.

Frame Hierarchies

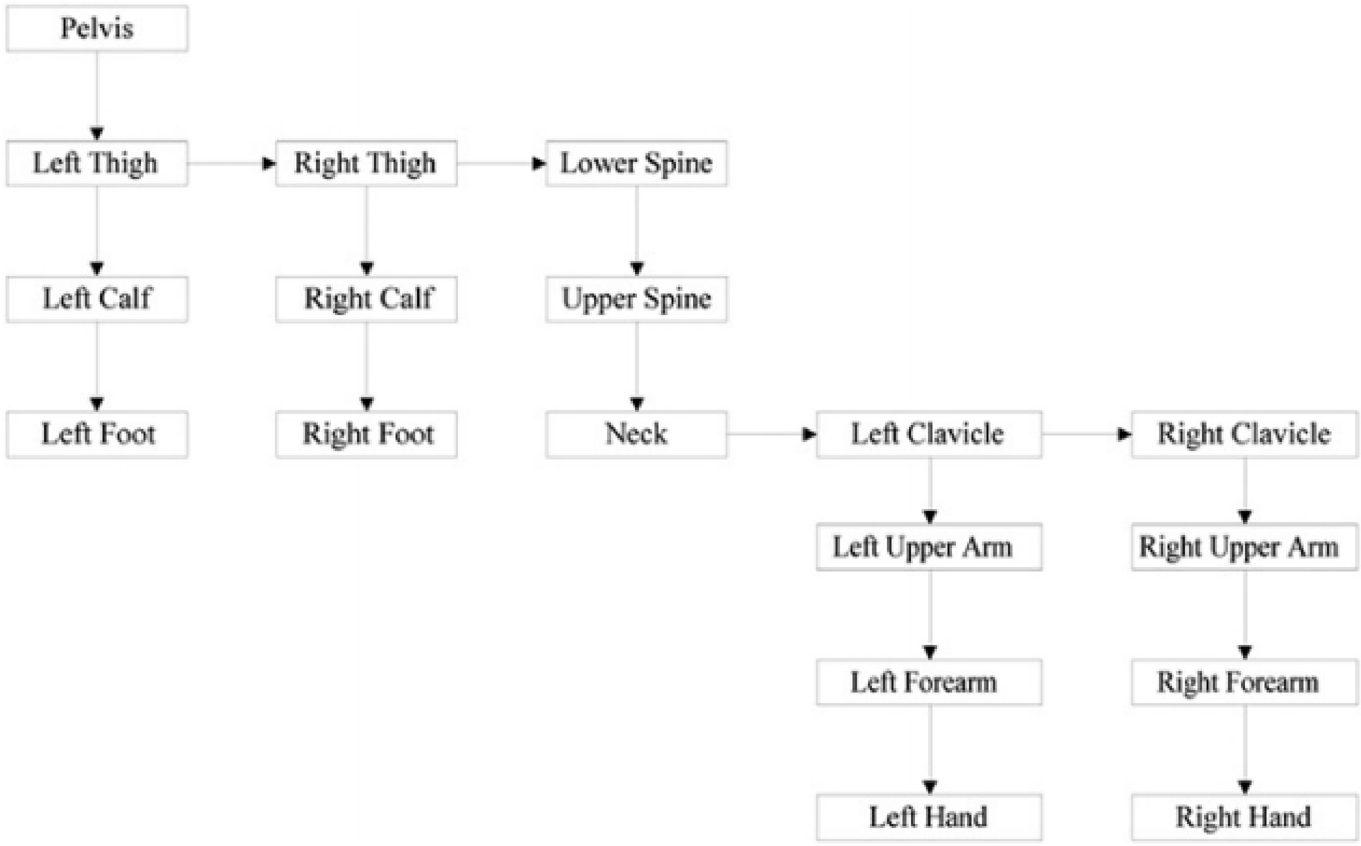
- Many objects are composed of parts, with a parent-child relationship.
- One or more child objects can move independently on their own.
- But are also forced to move when their parent moves.
- The hand can rotate in isolation about its wrist joint.
- But, if the forearm rotates about its elbow joint, then the hand must rotate with it.

Frame Hierarchies



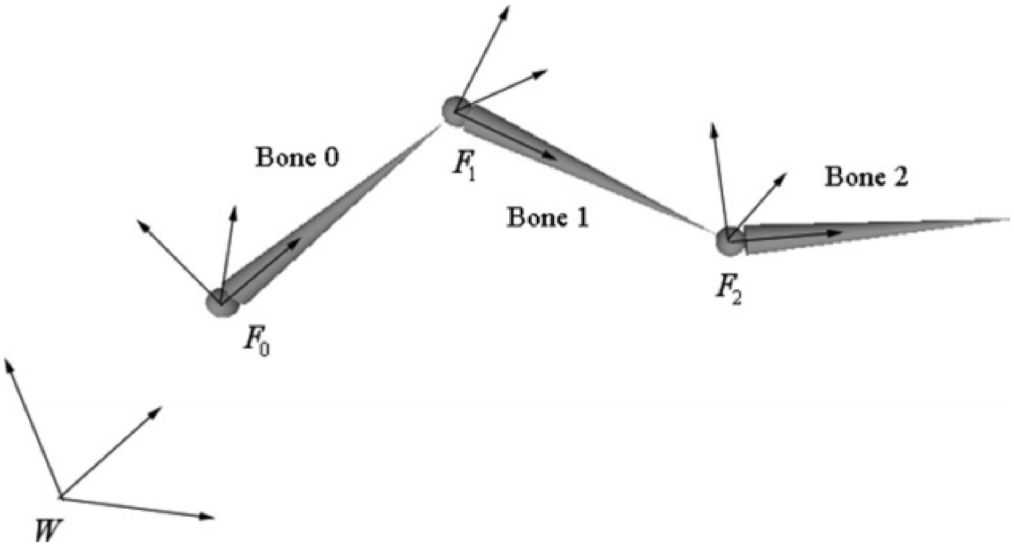
- Thus we see a definite object hierarchy: The hand is a child of the forearm; the forearm is a child of the upper arm, etc.

Frame Hierarchies



Bone Space to World Transformation

- Each object in the hierarchy is modeled about its own local coordinate system with its pivot joint at the origin to facilitate rotation.

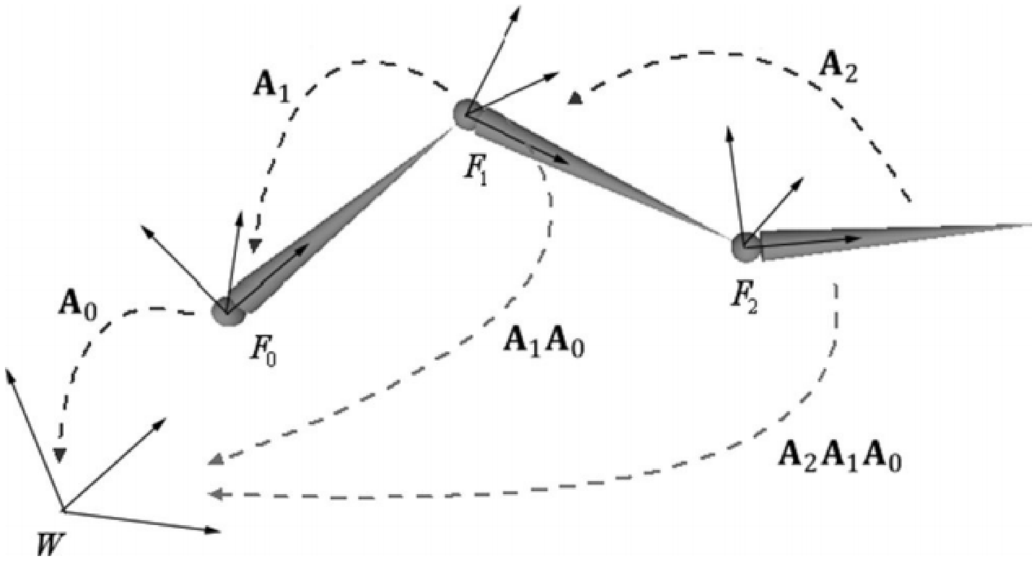


Bone Space to World Transformation

- We describe each coordinate system relative to its parent coordinate system.
- We can transform from a child's space to its parent's space with a transformation matrix.
- This is the same idea as the local-to-world transformation.
- we can transform the ith object in the arm hierarchy into world space by the matrix Mi

$$\mathbf{M}_i = \mathbf{A}_i \mathbf{A}_{i-1} \cdots \mathbf{A}_1 \mathbf{A}_0$$

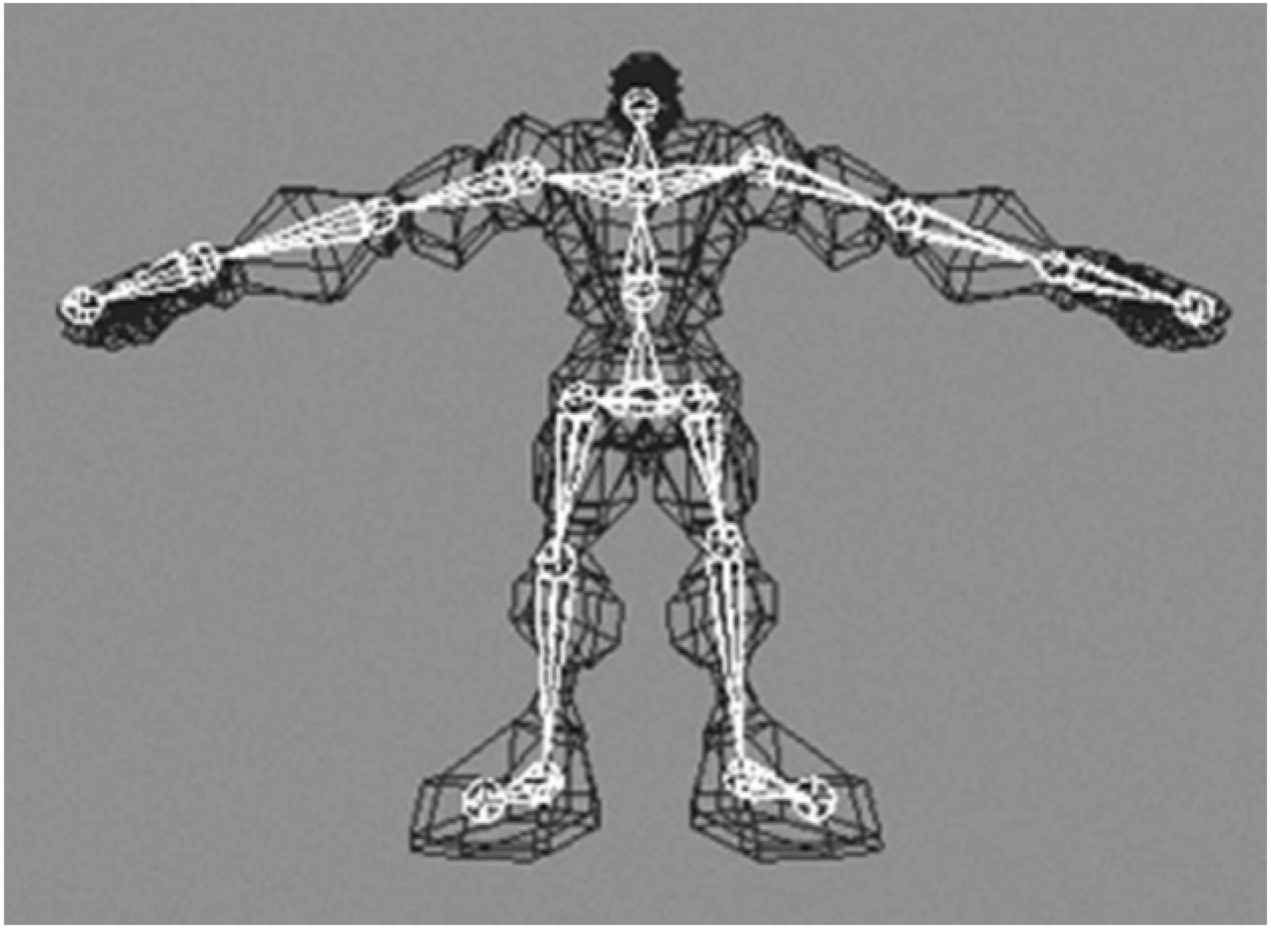
Bone Space to World Transformation



- M0 = A0 transforms the hand into world space
- M1 = A1A0, transforms the forearm into world space,
- M2 = A2A1A0, transforms the upper arm into world space,

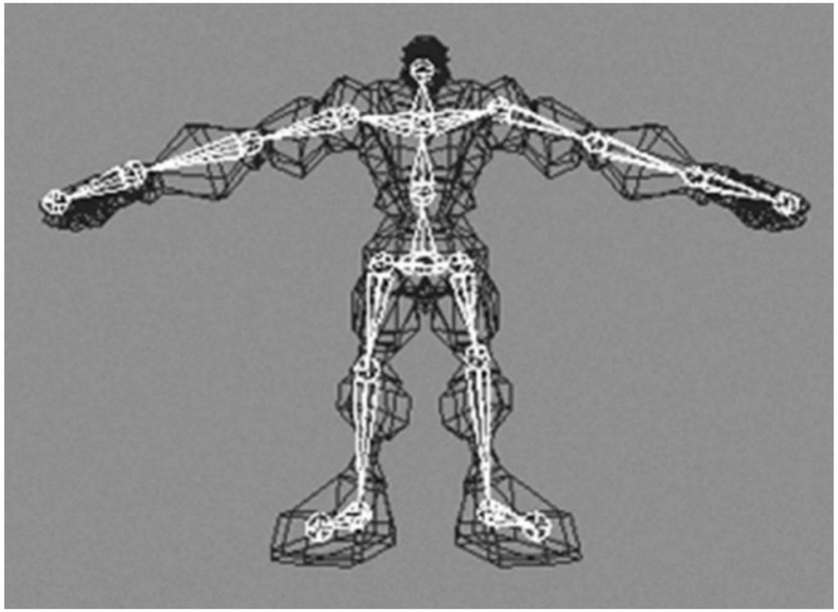
MEDIA DESIGN SCHOOL GAME DEV

Skinned Mesh

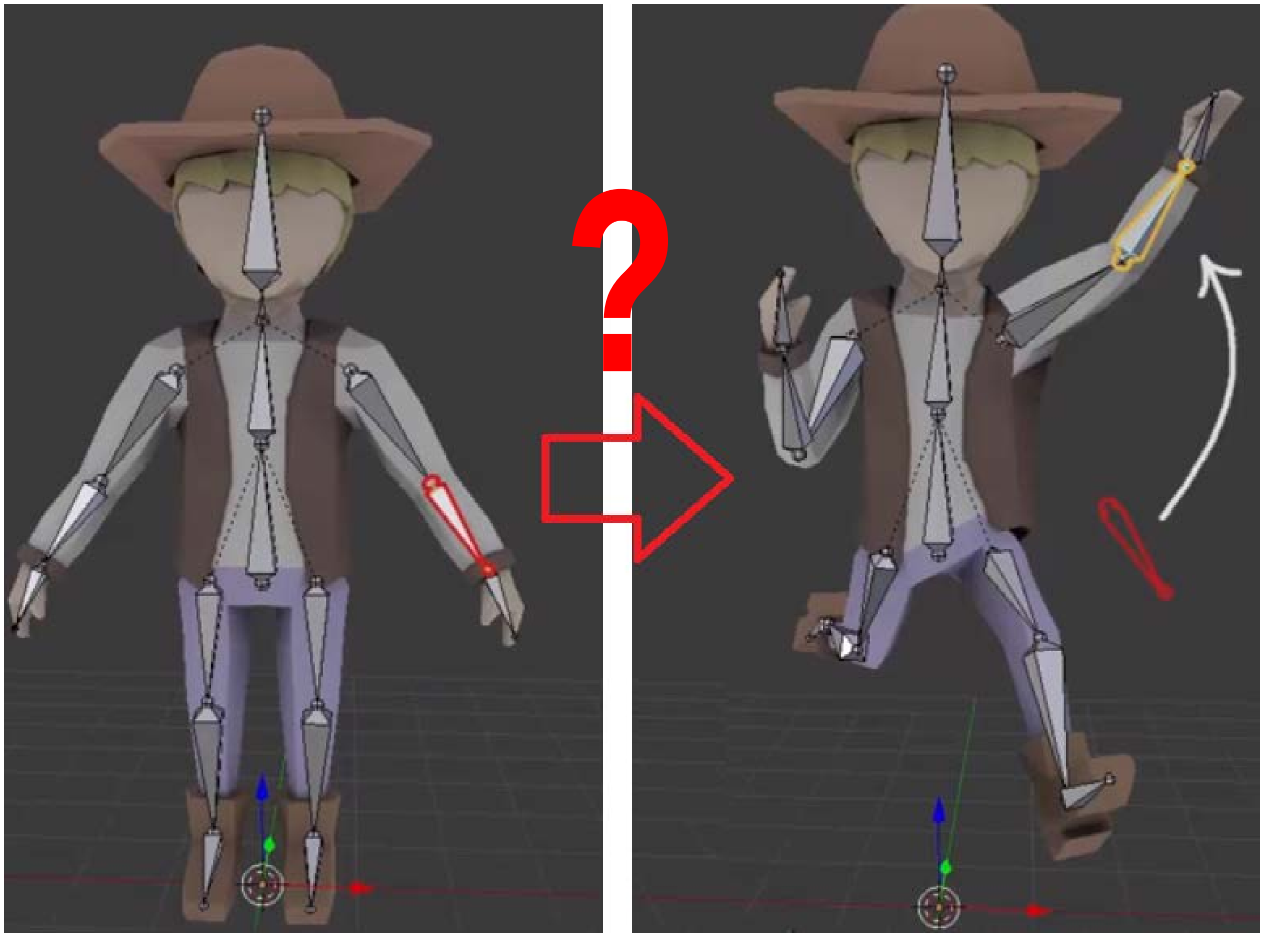


Skinned Meshes

- The highlighted chain of bones in the figure is called a ***skeleton***.
- Skeleton provides a natural hierarchal structure for driving a character animation system.
- The skeleton is surrounded by an exterior ***skin***,
- Initially, the skin vertices are relative to the ***bind space (T- pose)***, the root coordinate system



Skinned Mesh Transform



Skinned Mesh Transform

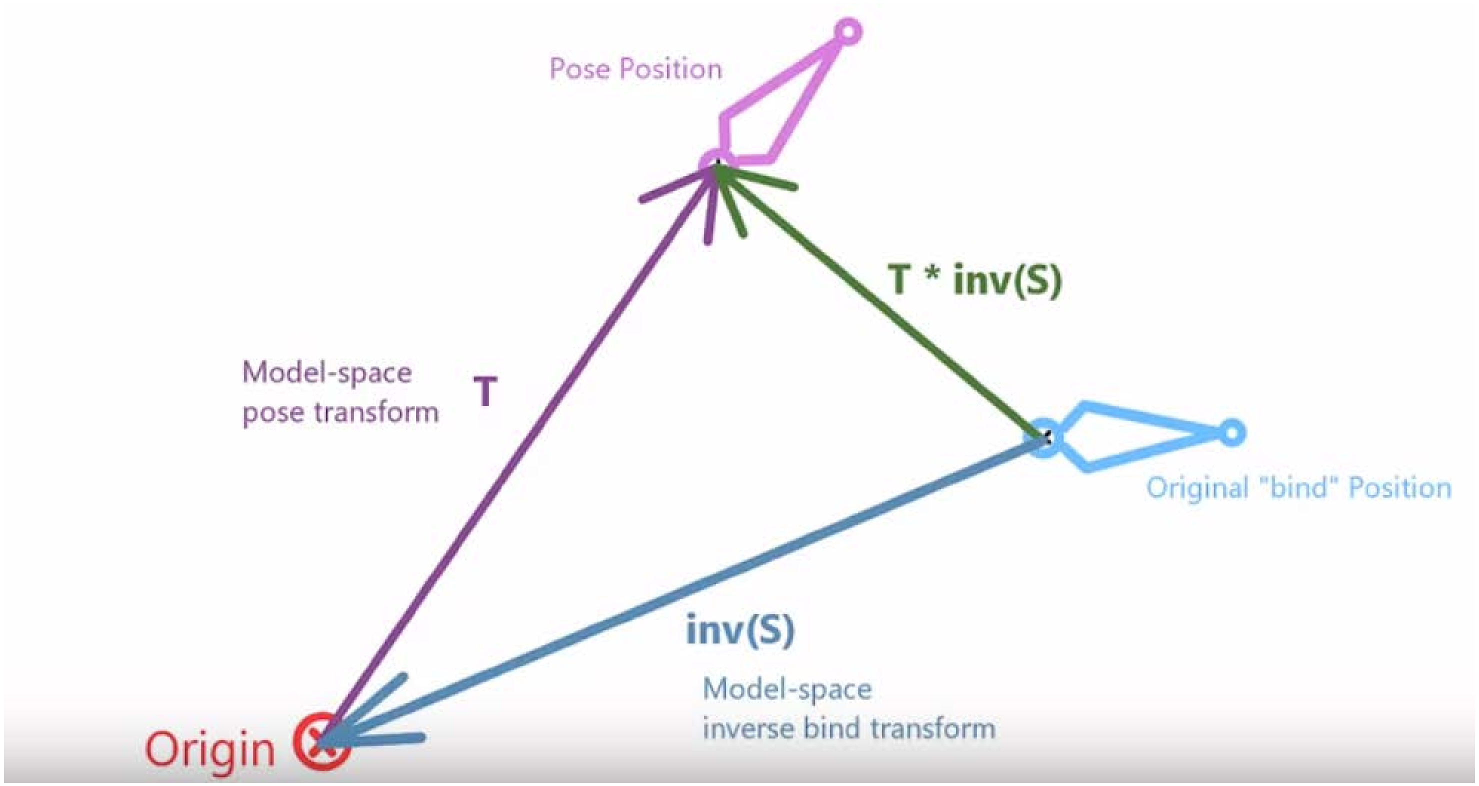
- We will transform from the root coordinate system to the world coordinate system in a separate step.
- So rather than finding the to-world matrix for each bone, we find the to-root matrix for each bone.
- It is actually more efficient to take a top-down approach. where we start at the root and move down the tree.
- So, we first need to transform the vertices from bind space to the space of the bone that influences the vertices.
- Offset transformation matrix does this.

Skinned Mesh Transform

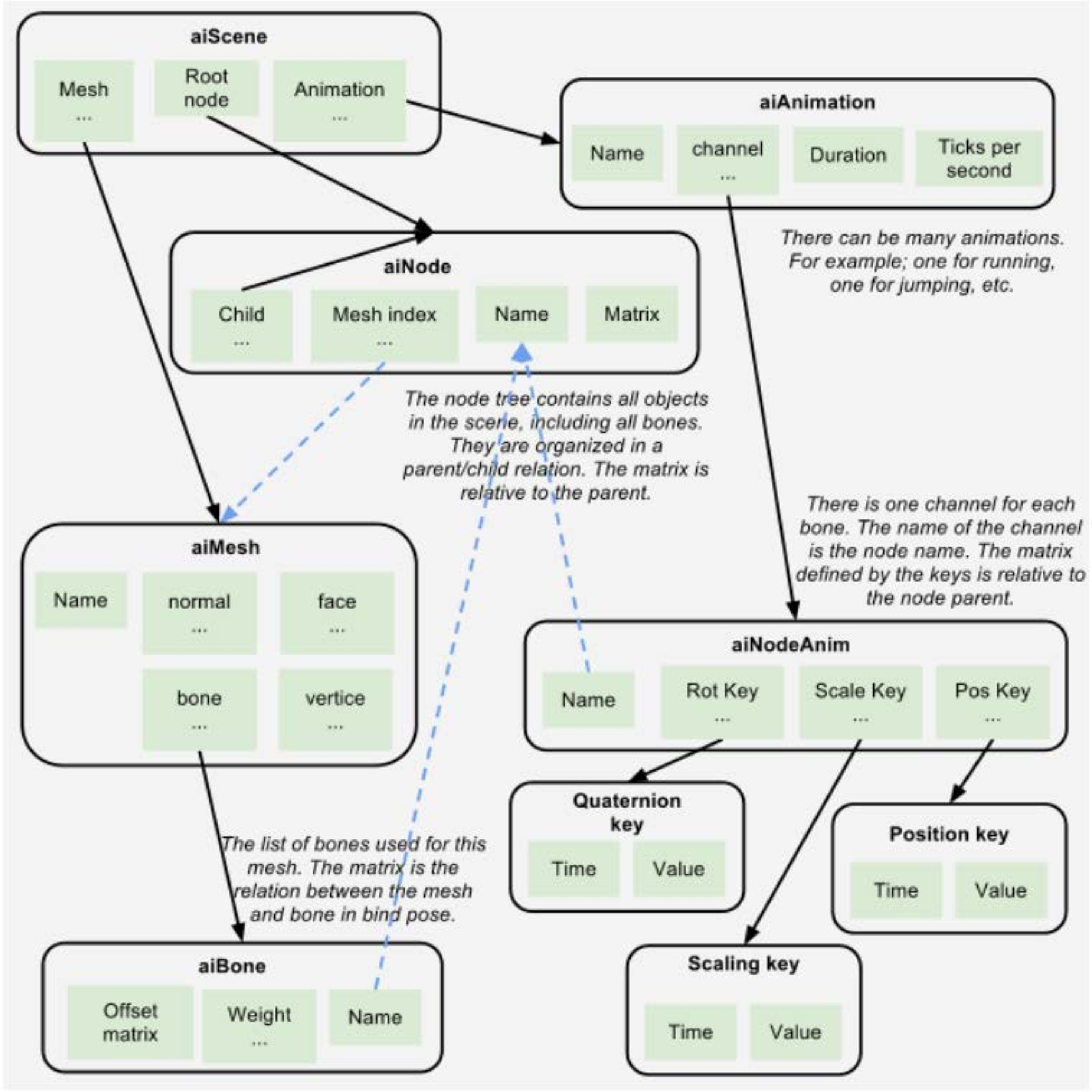
- We now introduce a new transform, call it the final transform.
- This combines a bone's *offset (inverse bind transform)* transform with its *to-root* (model space transform) transform.

$$\mathbf{F}_i = offset_i \cdot toRoot_i$$

[redacted]



Pose Position
T * inv(S)
Model-space
pose transform
T
Original "bind" Position
inv(S)
Model-space
inverse bind transform
Origin



MEDIA DESIGN SCHOOL GAME DEV

Process

- Load mesh with filename
  - Get scene
  - Get global Inverse transform
  - Get bone hierarchy
- Get mesh information
  - Position, normal, texcoords and indices.
  - Bone data (weights and ids) per vertex
  - Set buffers and attributes
  - initMaterials
  - initMesh

[redacted]

- initMesh
  - Set vertex attribute vector
  - Set index vector
  - Load bones ()
- Load bones ()
  - Gets bones index
  - And bones offset matrix
  - Stores in boneMapping map – bone name to id
  - Per vertex stores all the bones affecting it and by how much

Render

- Render
  - setShaderUniforms/ Variables
  - bindVao
  - Interate through entities
    - Set material/ texture
    - Draw each entity

setShadereffectVariables function

- setShadereffectVariables
  - Set model matrix
  - Set ViewProjection matrix
  - Set lighting information

  - Pass in bone Tranformations for all the bones in the hierarchy for each frame as an array of matrices.

setShadereffectVariables function

```
for (unsigned int i = 0; i < ARRAY_SIZE(m_boneLocation); i++) {
char name[128];
memset(name, 0, sizeof(name));
sprintf_s(name, "jointTransforms[%d]", i);
m_boneLocation[i] = glGetUniformLocation(program, name);
} // get locations

std::vector<Matrix4f> transforms;

boneTransforms(dt, transforms); // get transform matrix

for (int i = 0; i < transforms.size(); i++) {
Matrix4f Transform = transforms[i];
glUniformMatrix4fv(m_boneLocation[i], 1, GL_TRUE, (const GLfloat*)(Transform))
} // set transform matrix
```

ReadNodeHeirarchy function

- For each bone
  - Calculate interpolated scale, rotation and position information
  - Multiply by parent transform
  - Multiply by bones offset matrix which is the inverse bone transform.
  - And global inverse matrix
- Interpolation is used to calculate rotation, scale and position information between frames.

Bonetransforms function

- bonetransforms function calculates the final transformation for each bone at the current animation time.
- By going through all the children in the bone hierarchy by using the function ReadNodeHierarchy().

```
ReadNodeHeirarchy(animationTime, m_pScene->mRootNode, Identity);
transforms.resize(m_NumBones);

for (GLuint i = 0; i < m_NumBones; i++) {
transforms[i] = m_BoneInfo[i].FinalTransformation;
}
```

```
#version 420 core

const int MAX_JOINTS = 100;//max joints allowed in a skeleton
const int MAX_WEIGHTS = 4;//max number of joints that can affect a vertex

layout (location = 0) in vec3 position;
layout (location = 1) in vec2 texCoord;
layout (location = 2) in vec3 normal;
layout (location = 3) in ivec4 boneIds;
layout (location = 4) in vec4 weights;

out vec2 TexCoord;
out vec3 Normal;
out vec3 FragPos;

uniform mat4 vp;
uniform mat4 model;
uniform mat4 jointTransforms[MAX_JOINTS];
```

Vertex Shader

Vertex Shader (contd)

```
void main(void){

    vec4 totalLocalPosition = vec4(0.0);
    vec4 totalNormal = vec4(0.0);

    for(int i = 0; i < MAX_WEIGHTS; i++){

        vec4 posePosition = jointTransforms[boneIds[i]] * vec4(position, 1.0);
        totalLocalPosition += posePosition * weights[i];

        vec4 worldNormal = jointTransforms[boneIds[i]] * vec4(normal, 0.0);
        totalNormal += worldNormal * weights[i];
    }
    gl_Position =  vp * model * totalLocalPosition;

    TexCoord = texCoord;
    FragPos = vec3(model* vec4(position, 1.0f));
    Normal = mat3(transpose(inverse(model))) * normal;
}
```

Fragment Shader

- Fragment Shader is unchanged.
- Lighting and shadow calculations are done as usual.
- Pass in the object texture to render object as usual

Usage

```
//init
animatedModel = new ssAnimatedModel("theDude_idle_run.dae",
"", // texture name
camera,
animatedModelShaderProgram,
light);

animatedModel->setCurrentAnimation(0, 30);// set idle animation
animatedModel->setPosition(glm::vec3(-40.0f, 0.0f, 0.0f));
animatedModel->setScale(glm::vec3(0.0675f));
animatedModel->setSpeed(50.0f);
```

keyboard update

```
if (keyState[(unsigned char)'t'] == BUTTON_DOWN) {
    animatedModel->move(25.0f);
        if (animatedModel->bMoving == false) {
               animatedModel->bMoving = true;
               animatedModel->setCurrentAnimation(31, 50); // run animation
         }
}else if (keyState[(unsigned char)'g'] == BUTTON_DOWN) {
     animatedModel->move(-25.0f);
          if (animatedModel->bMoving == false) {
                 animatedModel->bMoving = true;
                 animatedModel->setCurrentAnimation(31, 50); // run animation
         }
}else {
       animatedModel->move(0.0f);
          if (animatedModel->bMoving == true) {
                animatedModel->bMoving = false;
                animatedModel->setCurrentAnimation(0, 30); //idle animation
          }
}
```

Render

- animatedModel->render(currentTime, terrain);

Notes

- You can use collada files or fbx files
- Note fbx files created with blender work but with Maya or or 3DSMax **doesn’t.**
- This is because there are different versions of fbx.
- Better to use collada (.dae) files if you are using 3DSMax, Maya or Blender.

Notes

- Also make sure the texture for the object is present at the correct location.
- If not then the texture wont render.
- Open the .dae file in notepad and edit the location.
  - <library_images> <image id="Map #0-image" name="Map #0"><init_from>file://theDude.png</init_from></image>

References

- OpenGL Skeletal Animation Tutorial 1-4
  - https://www.youtube.com/watch?v=f3Cr8Yx3GGA&t=6s
- Skeletal Animation With Assimp
  - http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html