

Skeletal Animation

Instruction

Model Loading

- Collada file format (.dae)
 - Mesh
 - Vertices
 - Position
 - Normal
 - Texture coordinate
 - Indices
 - **Skeleton**
 - **Bone names**
 - **Hierarchy**
 - **Transformation matrix**
 - **The number of bones that influences the vertex**
 - **Weights**

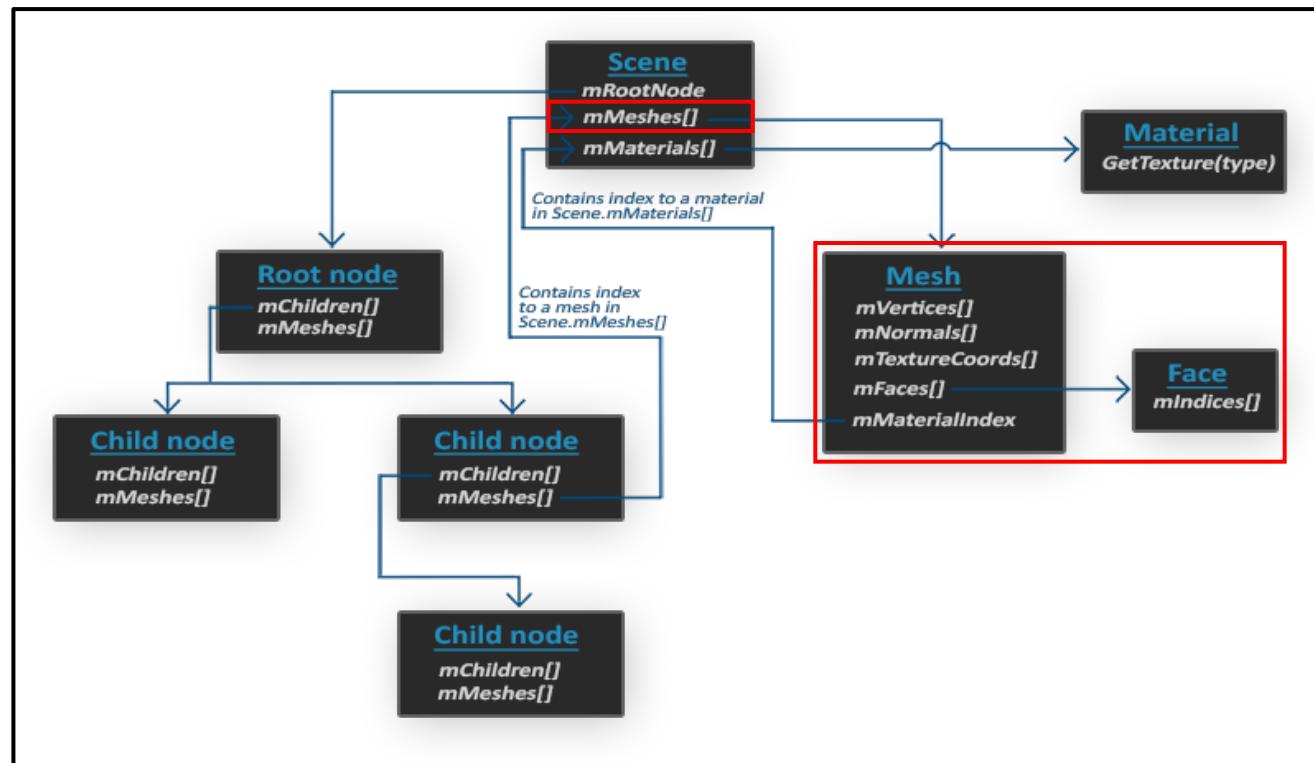
Model Loading

- Collada file format (.dae)
 - **Animation**
 - **keyframes**

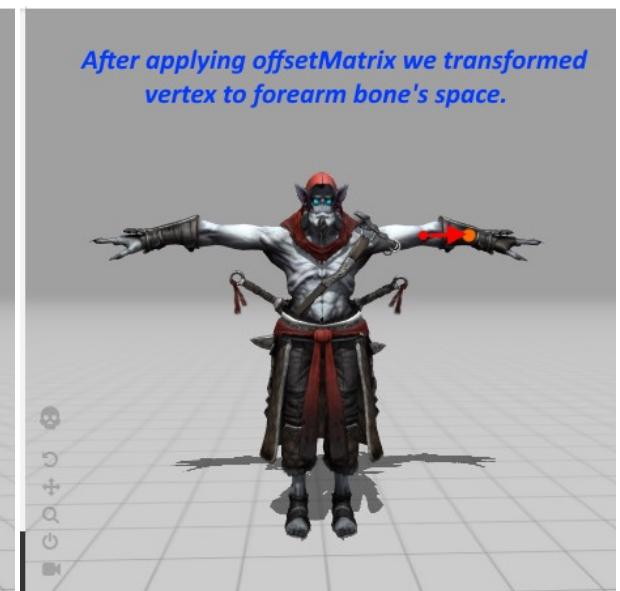
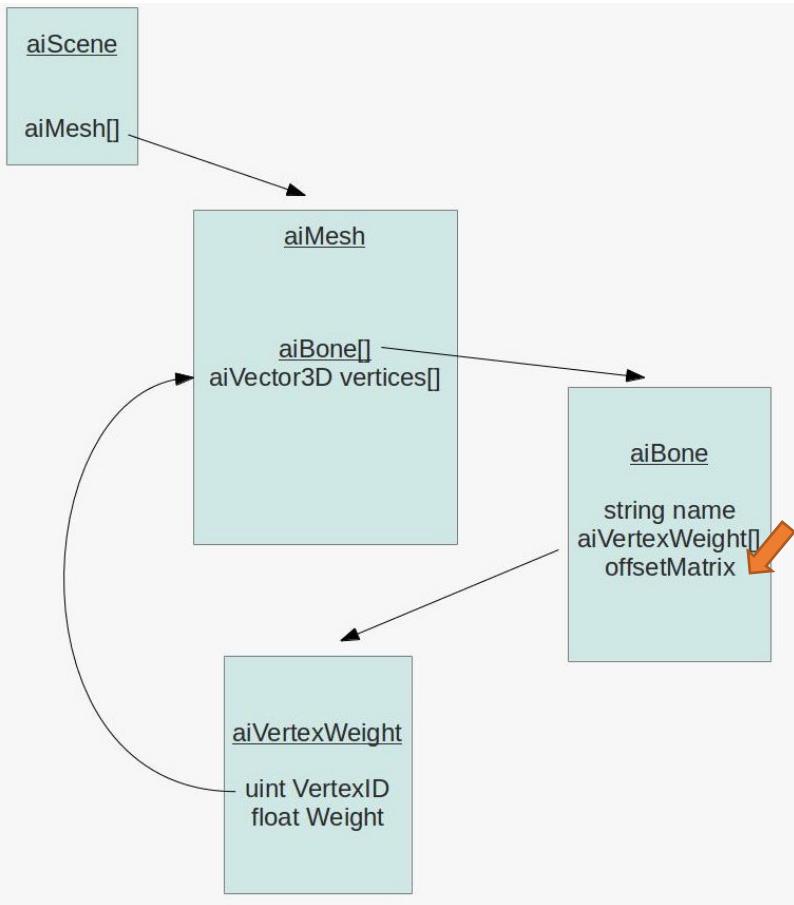
Model Loading

- Assimp 5.0.1 (3D model importer)
 - <https://github.com/assimp/assimp>

Mesh Data

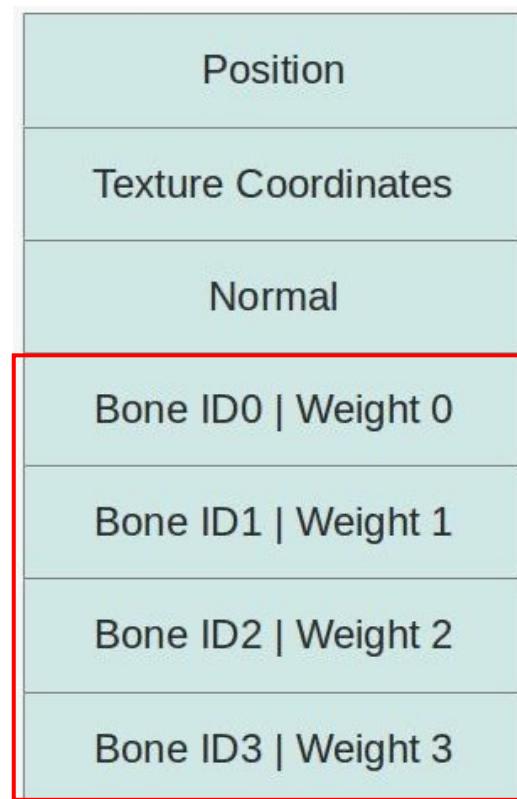


Bone Data

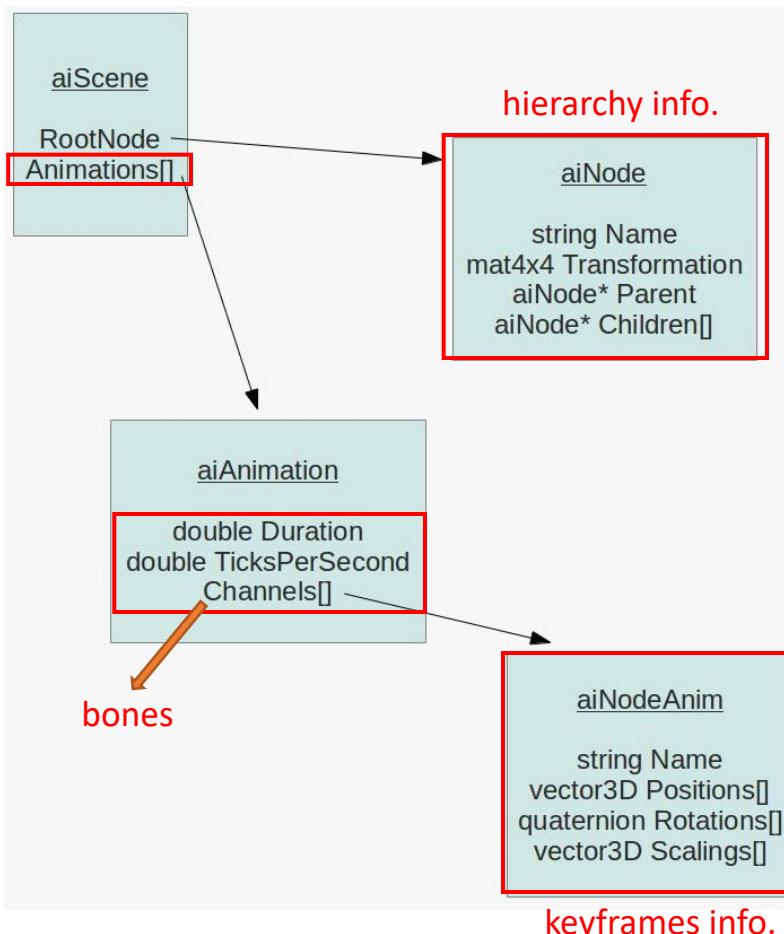


Bone Data

- Vertex Structure



Animation Data



Animation Data

Node hierarchy:

- **Scene**
 - Armature
 - bone_root
 - bone_body
 - ...
 - Camera
 - Light

Bone Transformation Calculation

- Calculate the bone transformations that go into the shader every frame

```
void boneTransform(double secs, vector<glm::mat4x4>& transforms)
{
    glm::mat4 identity_matrix(1.0f);

    double ticks = secs * ticks_per_second;
    float animation_time = 0;
    if (scene->HasAnimations()) {
        animation_time = fmod(ticks, scene->mAnimations[0]->mDuration);
    }

    if (run_animation) {
        readNodeHierarchy(animation_time, scene->mRootNode, identity_matrix); → Process the node hierarchy
    } else {
        readNodeHierarchy(0, scene->mRootNode, identity_matrix);
    }

    transforms.resize(num_bones);

    for (int i = 0; i < num_bones; i++) {
        transforms[i] = bone_matrices[i].final_transform;
    }
}
```

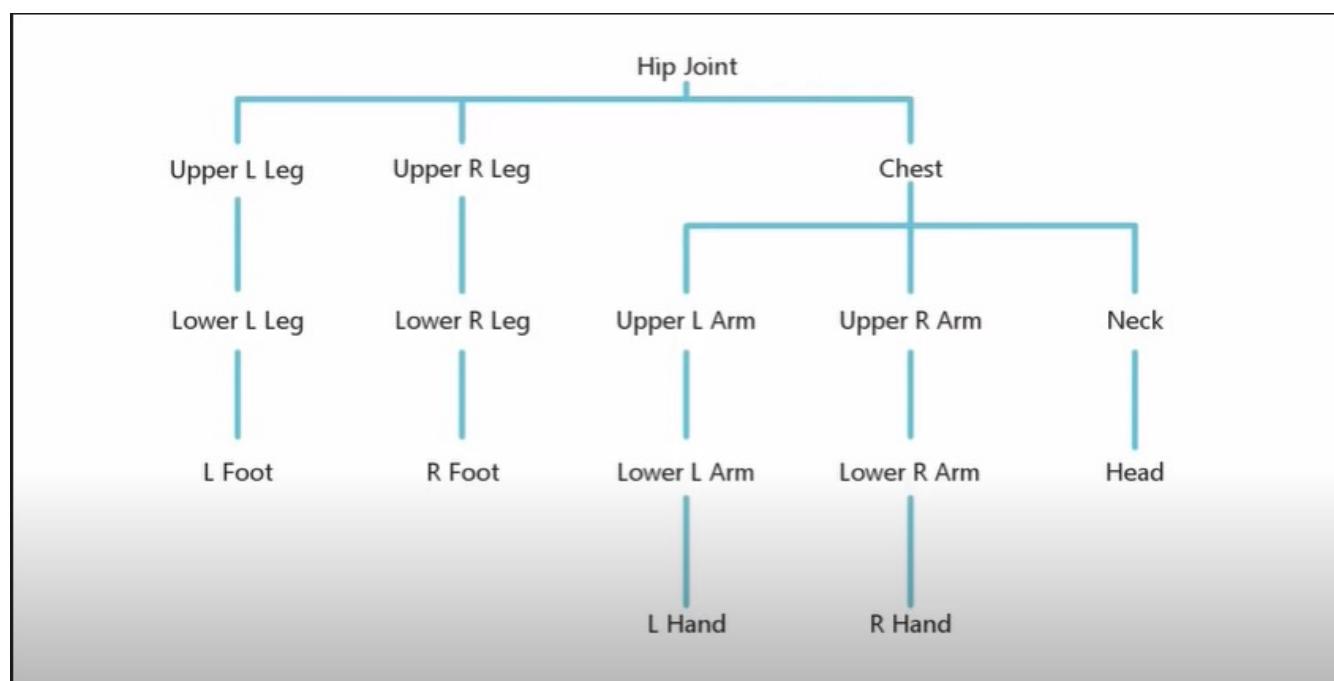
Current time Output: Final Bone transformations

The code snippet shows the implementation of the `boneTransform` function. It takes a `double secs` parameter and a `vector<glm::mat4x4>& transforms` reference. The function first initializes an `identity_matrix`. It then calculates `ticks` from `secs` and `ticks_per_second`. If the scene has animations, it calculates `animation_time` using `fmod`. It then checks if `run_animation` is true. If true, it calls `readNodeHierarchy` with `animation_time`, `scene->mRootNode`, and `identity_matrix`. Otherwise, it calls `readNodeHierarchy` with `0`, `scene->mRootNode`, and `identity_matrix`. After processing the hierarchy, it resizes the `transforms` vector and loops through all bones, setting their final transformation to the value in `bone_matrices[i].final_transform`. Red annotations highlight the `readNodeHierarchy` call with `animation_time` and the output variable `transforms`.

Bone Transformation Calculation

- Process the node hierarchy and get the **final transformation for each bone**.
 1. Search for the node name in the **channels** of the animation.
 2. Interpolate the **scaling** vector, **rotation** quaternion, and **translation** vector
 3. **globalTransformation = parentMatrix * interpolatedTransformation**
 4. **finalTransformation = globalTransformation * boneInfo[nodeName].offset**

Bone Transformation Calculation



Skinning

- Linear blending skinning (in vertex shader)

```
mat4 bone_transform = bones[bone_ids[0]] * weights[0];
bone_transform += bones[bone_ids[1]] * weights[1];
bone_transform += bones[bone_ids[2]] * weights[2];
bone_transform += bones[bone_ids[3]] * weights[3];
```



References

References

- 3D models
 - https://www.cgtrader.com/free-3d-models?file_types%5B%5D=25
- Skeletal Animation
 - <https://learnopengl.com/Guest-Articles/2020/Skeletal-Animation>
 - <https://ogldev.org/www/tutorial38/tutorial38.html>
 - https://www.youtube.com/watch?v=f3Cr8Yx3GGA&list=PLRIWtICgwaX2tKWCxdeB7Wv_rTET9JtWW
- Assimp (3D model importer)
 - <https://assimp.org/index.php>
- Blender
 - <https://www.blender.org/>