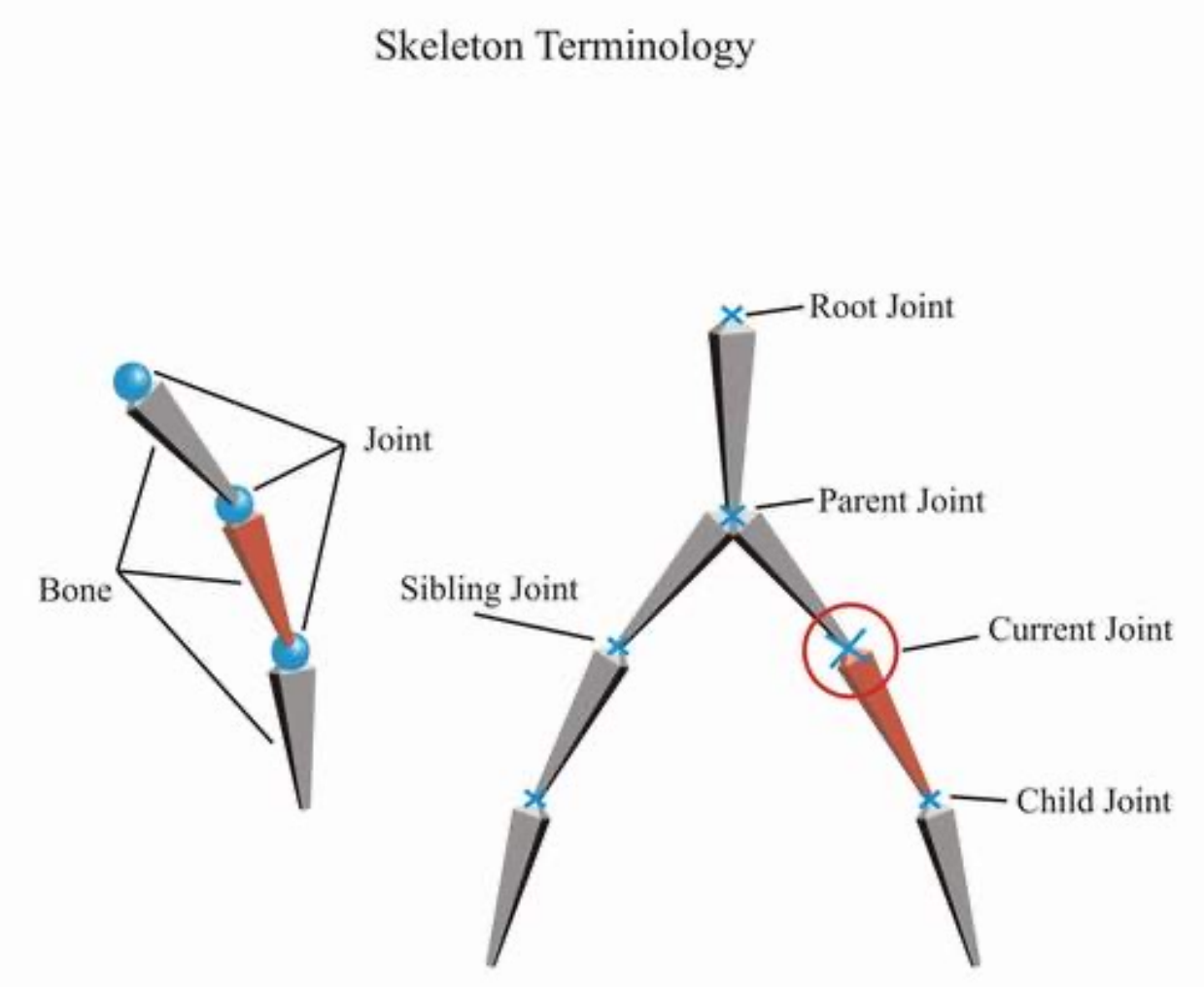
Karakter-animáció

Szécsi László

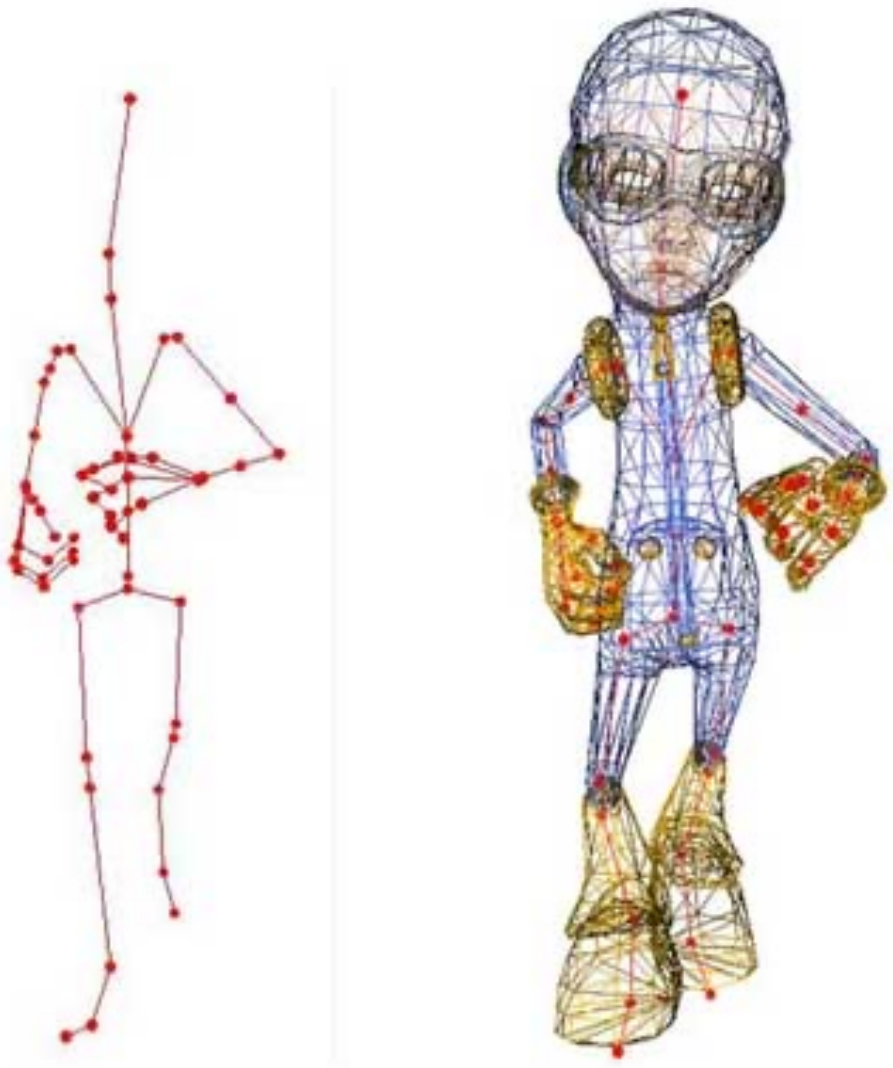
3D Grafikus Rendszerek

18. előadás

Csontanimáció



Skinning: csont és bőr



AssImp

- Open Asset Import Library
- formátumok
  - Wavefront .obj
  - Autodesk .fbx
  - Collada .dae

AssImp interface

- assimp::Importer
  - ReadFile
- aiScene
  - **aiMesh[]**
  - aiMaterial[]
  - **aiAnimation[]**
  - **aiNode hierarchy**
  - aiCamera[]
  - aiLight[]
  - aiTexture[]

aiMesh

- vertices (pozíciók)
- UVs[]
- normals
- tangents, bitangents (binormals)
- colors
- faces
- materialIndex[]
- ~~animMesh[]~~
- **bones**

Bone vs node

- Bone
  - mesh része
  - van neve
  - hozzá kötött vertexek és súlyok
- Node
  - animáció része
  - van neve
  - hierarchiába vannak szervezve
    - szülő, gyerekek
  - transzformáció a szülőhöz képest
  - hivatkoznak rá animációk, amik a transzformáció időfüggését adhatják meg

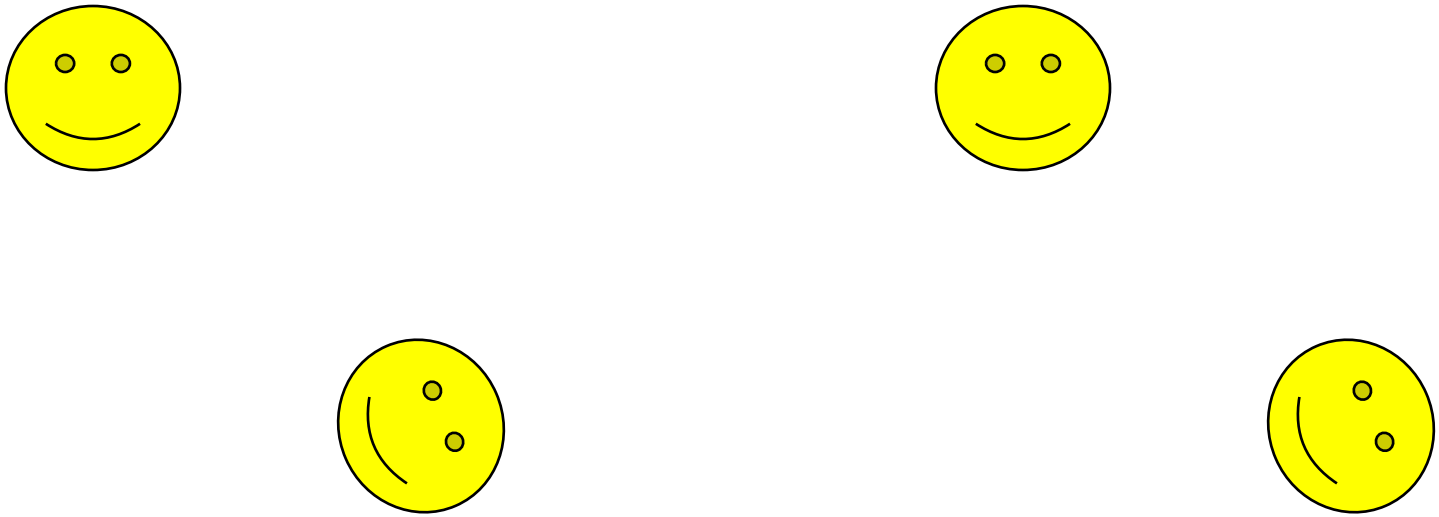
Bone vs node

- lehet bone, amihez nincs node?
- lehet node, amihez nincs bone?
- lehet node, amihez nincs animáció?

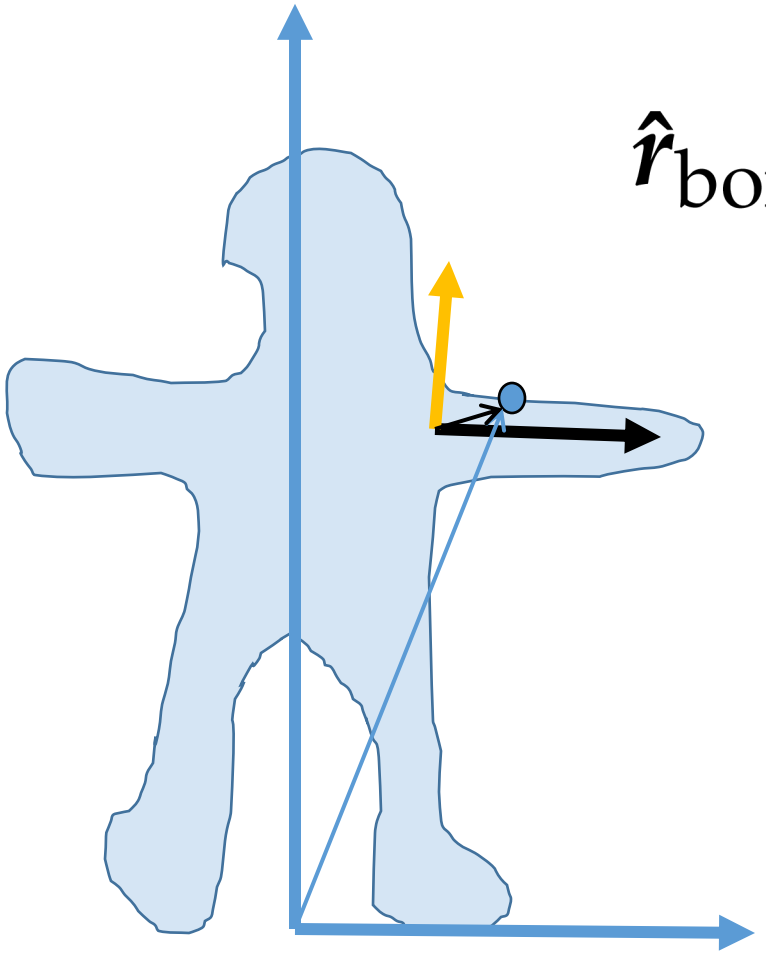
Duális kvaterniók

- elforgatás- és eltolásreprezentáció nyolc számmal
  - q: elforgatáskvaternió
  - t: pozíció a forgó koordinátarendszerben, kvaternióba kódolva
- szorzás: transzformációk egymásutánja
- pontokat, irányokat lehet vele transzformálni
- előnyök a forgatásmátrixhoz képest
  - feleannyi szám
  - szebb interpoláció

Pózinterpoláció duális kvaterniókkal

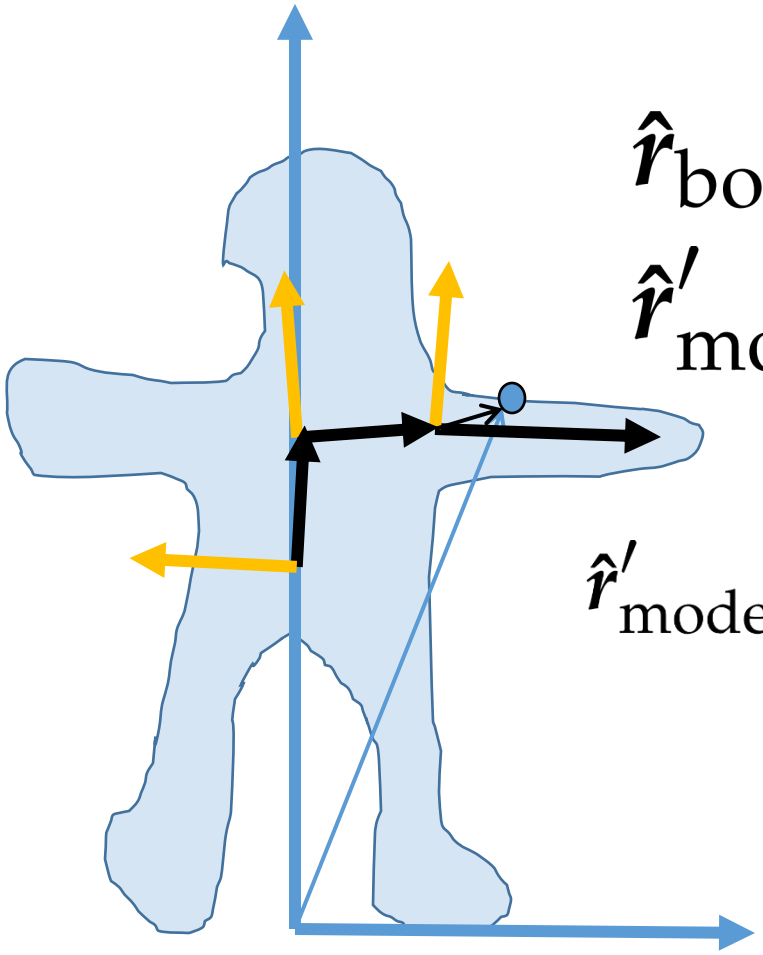


Rigging



$$\hat{\boldsymbol{r}}_{\text{bone}} = \hat{\boldsymbol{r}}_{\text{model}} \cdot \boldsymbol{T}_{\text{rigging}}$$

Csonttranszformációk



$$\hat{\boldsymbol{r}}_{\text{bone}} = \hat{\boldsymbol{r}}_{\text{model}} \cdot \boldsymbol{T}_{\text{rigging}}$$

$$\hat{\boldsymbol{r}}'_{\text{model}} = \hat{\boldsymbol{r}}_{\text{bone}} \cdot \boldsymbol{T}_{\text{bone}}$$

$$\hat{\boldsymbol{r}}'_{\text{model}} = \hat{\boldsymbol{r}}_{\text{model}} \cdot \boldsymbol{T}_{\text{rigging}} \cdot \boldsymbol{T}_{\text{bone}}$$

$$\boldsymbol{T}_{\text{bone}} = \boldsymbol{T}_{\text{hip}} \boldsymbol{T}_{\text{spine}} \boldsymbol{T}_{\text{arm}}$$

Mesh betöltés – rigged geom

- vertex buffer: per vertex attribútumok
  - BLENDINDICES
  - BLENDWEIGHTS

- töltsük fel:
  - float4x4 rigging[nBones]
  - boneName -> iBone lookup

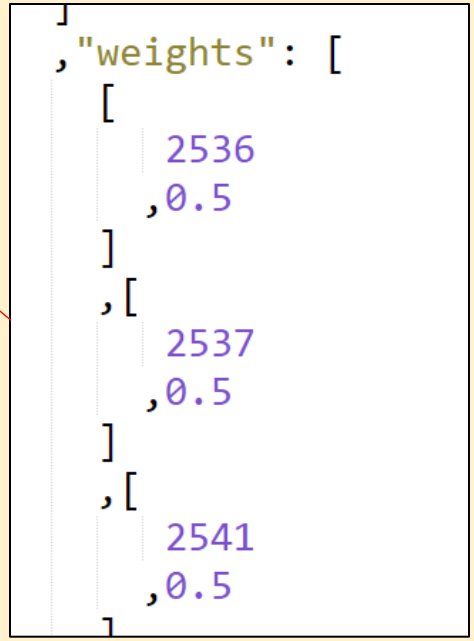
MultiMesh.kt – JSON formátum

```
@Serializable
data class JsonBone(
  val name : String,
  val offsetmatrix : Array<Float>,
  val weights : Array<Array<Float>>)

@Serializable
data class JsonRiggedMesh(
  val vertices : Array<Float>,
  val normals : Array<Float>,
  val texturecoords : Array<Array<Float>>,
  val faces : Array<Array<Short>>,
  val bones : Array<JsonBone>)

@Serializable
data class JsonRiggedModel(
  val meshes : Array<JsonRiggedMesh>)
```

rigging pózban hol van a csont



JsonLoader.kt – geometriabetöltés

```
fun loadRigged(gl : WebGL2RenderingContext,
        jsonModelFileUrl : String) : Array<RiggedGeometry> {
    val request = XMLHttpRequest()
    request.overrideMimeType("application/json")
    request.open("GET", jsonModelFileUrl, false)
    var geometries : Array<RiggedGeometry>? = null
    request.onloadend = {
      val json = Json { ignoreUnknownKeys=true }
      val jsonModel = json.decodeFromString(
                  JsonRiggedModel.serializer(), request.responseText)
      geometries = Array<RiggedGeometry>(jsonModel.meshes.size) {
        i -> RiggedGeometry(gl, jsonModel.meshes[i])
      }
      Unit
    }
    request.send()
    return geometries!!
```

RiggedGeometry.kt – csontnevek és rigging trafók

```
  val nBones = jsonMesh.bones.size
  val rigging = DualQuaternionArray(nBones)
  val boneNames = Array<String>(nBones) {
    val offsetMatrix =
Mat4(*jsonMesh.bones[it].offsetmatrix.toFloatArray())
    offsetMatrix.transpose()
    rigging[it].fromMatrix(offsetMatrix)
    jsonMesh.bones[it].name
  }
}
```

RiggedGeometry.kt – új bufferek

```
val blendIndicesBuffer = gl.createBuffer()
val blendWeightsBuffer = gl.createBuffer()
init {
  val blendIndices = Uint8Array(jsonMesh.vertices.size/3*4)
  val blendWeights = Float32Array(jsonMesh.vertices.size/3*4)
  for(i in 0 until jsonMesh.vertices.size/3*4) {
    blendWeights[i] = 0.0f
    blendIndices[i] = 0
  }

  // következő dián: adatok összevadászása

  gl.bindBuffer(GL.ARRAY_BUFFER, blendIndicesBuffer)
  gl.bufferData(GL.ARRAY_BUFFER, blendIndices, GL.STATIC_DRAW)
  gl.bindBuffer(GL.ARRAY_BUFFER, blendWeightsBuffer)
  gl.bufferData(GL.ARRAY_BUFFER, blendWeights, GL.STATIC_DRAW)
}
```

RiggedGeometry.kt – csontonkénti vertexlistából vertexenkénti csontlisták

```
val nWeightsPerVertex = Uint8Array(jsonMesh.vertices.size/3)
for(i in 0 until jsonMesh.vertices.size) {
  nWeightsPerVertex[i] = 0
}

jsonMesh.bones.forEachIndexed { iBone, bone ->
  for(weight in bone.weights) {
    val iVertex = weight[0].toInt()
    val weights = weight[1]
    blendWeights[ iVertex*4 + nWeightsPerVertex[iVertex] ] =
                                          weights
    blendIndices[ iVertex*4 + nWeightsPerVertex[iVertex] ] =
                                          iBone.toByte()
    nWeightsPerVertex[iVertex]++
  }
}
```

RiggedGeometry.kt – input layout

```
gl.bindBuffer(GL.ARRAY_BUFFER, blendIndicesBuffer)
gl.enableVertexAttribArray(3)
gl.vertexAttribPointer(3,
  4, GL.UNSIGNED_BYTE, //< four integers
  false, //< do not normalize (make unit length)
  0, //< tightly packed
  0 //< data starts at array start
)
gl.bindBuffer(GL.ARRAY_BUFFER, blendWeightsBuffer)
gl.enableVertexAttribArray(4)
gl.vertexAttribPointer(4,
  4, GL.FLOAT, //< four floats
  false, //< do not normalize (make unit length)
  0, //< tightly packed
  0 //< data starts at array start
)
```

Animációk (a.k.a clip)

- scene-ben
  - több is lehet
  - pl. walking jumping, salsa dancing

- meshAnim – mesh morphing animáció
- aiNodeAnim channels[]

aiNodeAnim

- node neve
- positionKeys[]
  - idő
  - eltolás
- rotationKeys[]
  - idő
  - kvaternió
- scalingKeys[]
  - idő
  - skálafaktorok

Betöltés: animation

- node fa mélységi bejárása
  - minden nodera: van ilyen nevű csont?
  - a csont transzformációs lánca:
    - a csont rigging transzformációja
    - és fel a nodetól a gyökérig a node-ok aktuális transzformációi
- de mik a node-transzformációk?
  - minden node-ra van egy alapértelmezett
  - az aktuális animációban ha van a node-ra hivatkozó csatorna
    - abban levő idő/érték kulcsok határozzák meg a pillanatnyi értéket

Minden frameben