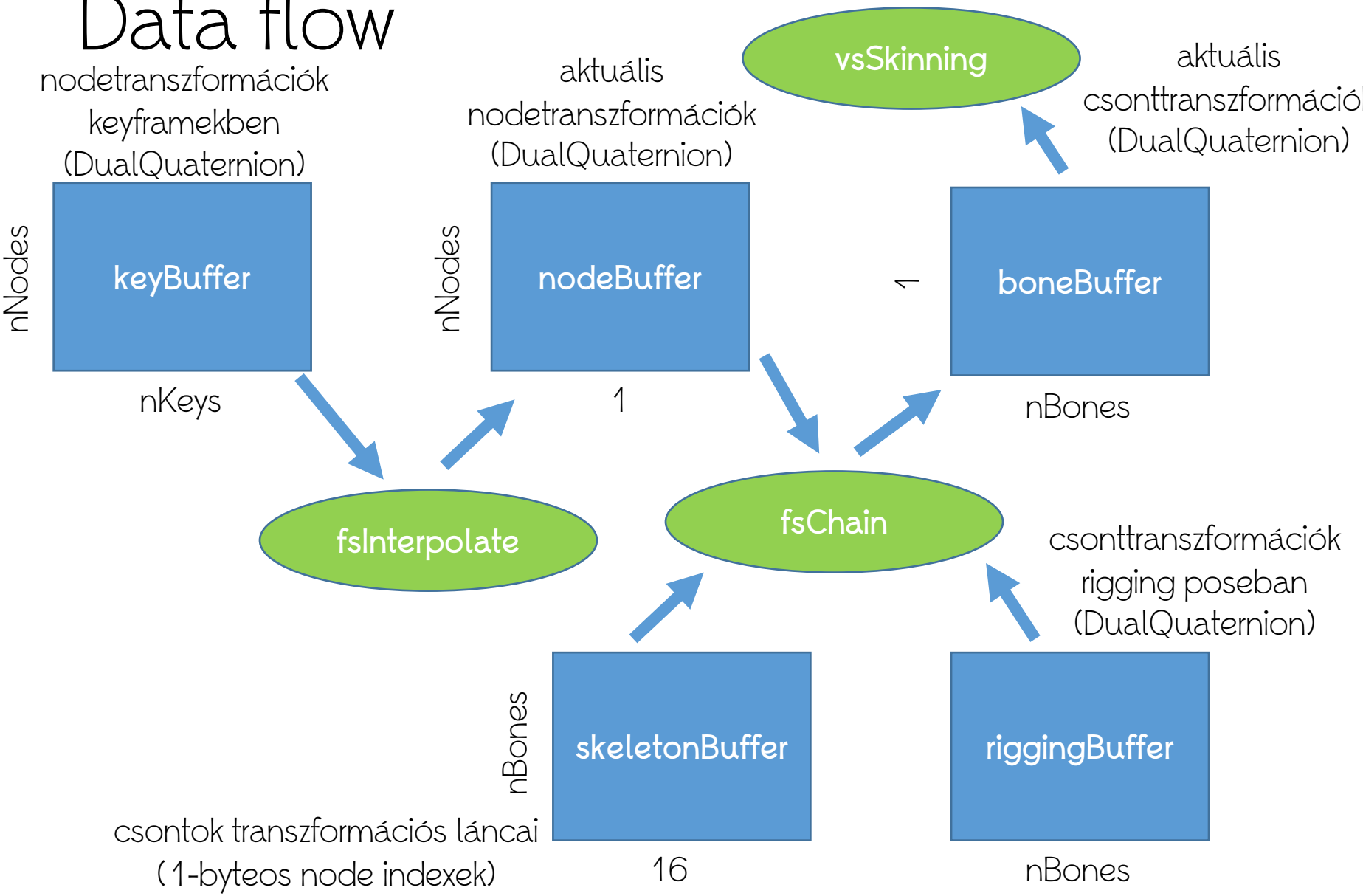
Karakteranimáció

Szécsi László

3D Grafikus Rendszerek

5. labor

Data flow


nodetranszformációk
keyframekben
(DualQuaternion)
nNodes
keyBuffer
nKeys
aktuális
nodetranszformációk
(DualQuaternion)
nNodes
nodeBuffer
1
vsSkinning
aktuális
csonttranszformációk
(DualQuaternion)
1
boneBuffer
nBones
fsInterpolate
fsChain
csonttranszformációk
rigging poseban
(DualQuaternion)
nBones
skeletonBuffer
csontok transzformációs láncai
(1-byteos node indexek)
16
riggingBuffer
nBones

Munkaterv

- modellbetöltés
- animáció betöltése, GPU-erőforrások létrehozása
- shaderek
  - interpolation-fs: kulcsokból nodeTrafók kiválasztása
    - valódi interpolációt most nem csinálunk, elég sűrűn vannak a keyframe-ek
  - chain-fs: rigging és nodetrafók összefűzése, csonttrafók kiszámítása
  - skinning-vs:
    - csonttrafók összesúlyozása, vertexek transzformálása
    - iMSc: példányok elhelyezése

Scene: rigging adatok betöltése

- **SubmeshGeometry** helyett **RiggedGeometry** betöltése

```
val riggedGeometry =
  JsonLoader.loadRigged(gl, "media/mrem.json")
```

- **MultiMesh**-hez a **RiggedGeometry** tömb helyett **Geometry** tömb (upcast, de tömbelemenként)

```
val skinnedMesh = MultiMesh(
  arrayOf(skinningMaterial),
  Array<Geometry>(riggedGeometry.size) {riggedGeometry[it]}
)
```

blendIndices és blendWeights attribútumok

- a **`RiggedGeometry`** ezeket betölti és beköti a VAO-ba mint #3 és #4 attribútum
- **`skinning-vs`**-be új bementek

```
in vec4 blendIndices; // uvec4 ??
in vec4 blendWeights;
```

- a **`skinningProgram`**-ot létrehozó **`Program`** konstruktor utolsó paraméterét változtassuk meg, hogy a fenti bemenetekre kössük a #3 és #4 attribútumokat

Scene: animáció betöltése

```
val zombieGeom = riggedGeometry[0]
val animation = Animation(gl,
     "./media/thriller_part_3.json",
     zombieGeom)
val nNodes = animation.nNodes.toInt()
```

Erőforrások létrehozása

- a **`RiggedGeometry`** felépítette a **`rigging`** tömböt
  - minden csontra egy duálkvaternió

```
val riggingTexture =
DataTexture(gl,
zombieGeom.rigging.storage,
zombieGeom.nBones*2)
```

- az **`Animation`** összerakta a **`skeleton`** tömböt
  - minden csontra egy nodelista (1 byteos indexek, 16 byte per csont)

```
val skeletonTexture = DataTexture(gl,
Uint32Array(animation.skeleton.buffer), zombieGeom.nBones,
1, GL.RGBA32UI, GL.RGBA_INTEGER, GL.UNSIGNED_INT)
```

- az **`Animation`** behúzta a **`keys`** tömböt
  - minden nodera, minden kulcsra egy duálkvaternió

```
val keyTexture =
DataTexture(gl,
animation.keys.storage,
animation.nKeys*2, nNodes)
```

Erőforrások létrehozása

- kell két framebuffer
  - nodetranszformációk: minden nodera egy duálkvaternió
    - két render target, egy a Q, egy a T kvaterniónak
  - bonetranszformációk: minden csontra egy duálkvaternió
    - két render target, egy a Q, egy a T kvaterniónak

```
 val nodeFramebuffer = Framebuffer(gl, 2, 1, nNodes, GL.RGBA32F,
GL.RGBA, GL.FLOAT)
  val boneFramebuffer = Framebuffer(gl, 2,
animation.geometry.nBones, 1, GL.RGBA32F, GL.RGBA, GL.FLOAT)
```

  - az iMSc feladatban példányonként (lásd majd a diasor végén)

Erőforrások összekötése a uniformokon keresztül (legelső dia alapján)

```
init {
   interpolationQuad["keyTexture"]?.set( ??? )
   interpolationQuad["nNodes"]?.set( ???.toFloat())
   interpolationQuad["nKeys"]?.set( ???.toFloat())

   chainQuad["skeletonTexture"]?.set(??? )
   chainQuad["riggingTexture"]?.set( ??? )
   chainQuad["nodeQTexture"]?.set( ??? )
   chainQuad["nodeTTexture"]?.set( ??? )
   chainQuad["nNodes"]?.set( ???.toFloat() )
   chainQuad["nBones"]?.set( ???.toFloat() )

   skinnedMesh["boneQTexture"]?.set( ??? )
   skinnedMesh["boneTTexture"]?.set( ??? )
   skinnedMesh["nBones"]?.set( ???.toFloat() )
}
```

`Scene::update`-ben

- mélységteszt kikapcsolása
- **`nodeFramebuffer`**be számítás az interpolációval (quadot rajzolva)
- **`boneFrambuffer`**be számítás a láncszorzással (quadot rajzolva)
- mélységteszt visszakapcsolása
- alapértelmezett framebuffer (és viewport) visszaállítása
- képernyőtörlés és karakter rajzolása [már megvan]

chain-fs

- helybenhagyás-duálkvaternió helyett
- legyen a trafó a **riggingTexture**-ben a csonthoz tartozó

```
dq.q = texture(mesh.riggingTexture,
  vec2((iBone * 2.0 - 0.25) / mesh.nBones / 2.0, 0.5));
dq.t = texture(mesh.riggingTexture,
  vec2((iBone * 2.0 + 0.25) / mesh.nBones / 2.0, 0.5));
```

skinning-vs

- helybenhagyás-duálkvaternió helyett
- vegyük a **blendIndices.x**-edik csont trafóját a **boneTexture**ből

```
dq.q = texture(
  multiMesh.boneQTexture, vec2(
  (float(blendIndices.x)+0.5)/multiMesh.nBones,
  0.0 /*iMSc: instance*/
  )
);
dq.t = ???
```

Várt eredmény

- összepakolt karakter
- minden csont az origóban

interpolation-fs

- mesh.keyTextureből vesszük a trafót
  - nyilván az iNode-hoz tartozók közül
  - egyelőre olvassuk ki csak a nullás kulcshoz tartozót

```
float iNode = (texCoord.y * mesh.nNodes);
float iKey = 0.0;
fragQ = texture(mesh.keyTexture, vec2(
  (iKey + 0.5) / mesh.nKeys / 2.0,
  iNode / mesh.nNodes));
iKey += 1.0;
fragT = ???
```

chain fs

- balról szorozgassuk hozzá a duálkvaternióhoz iBone-hoz tartozókat sorban skeletonBufferből
  - lásd köv. dia

valahogy így

```
      uvec4 ns = texture(mesh.skeletonTexture,
            vec2(iBone / mesh.nBones, 0.5));

       uint iNode = (ns.x >> 0 ) & 0xffu;
       if(iNode != 0xffu){
             DualQuat s;
             s.q = texture(mesh.nodeQTexture, vec2(0.0 /*iMSc:
instance*/, (mesh.nNodes-0.5-float(iNode)) / mesh.nNodes));
             s.t = texture(mesh.nodeTTexture, vec2(0.0 /*iMSc:
instance*/, (mesh.nNodes-0.5-float(iNode)) / mesh.nNodes));
             dq = dqmul(s, dq);
       }
       // ezt még 15-ször, minden bytera
```

Várt eredmény

- álló karakter
- a testrészei a helyünkön vannak, de nem mozog

interpolate fs

- nullás kulcs helyett
- függjön az időtől: mesh.animationTime

```
interpolationQuad["animationTime"]?.set(t)
```

- figyeljünk rá, hogy a kulcs mindig 2n legyen
- és mesh.nKeys * 2 után kezdődjön újra

Várt eredmény

- mozgó karakter
- merev részekből áll

Smooth skinning

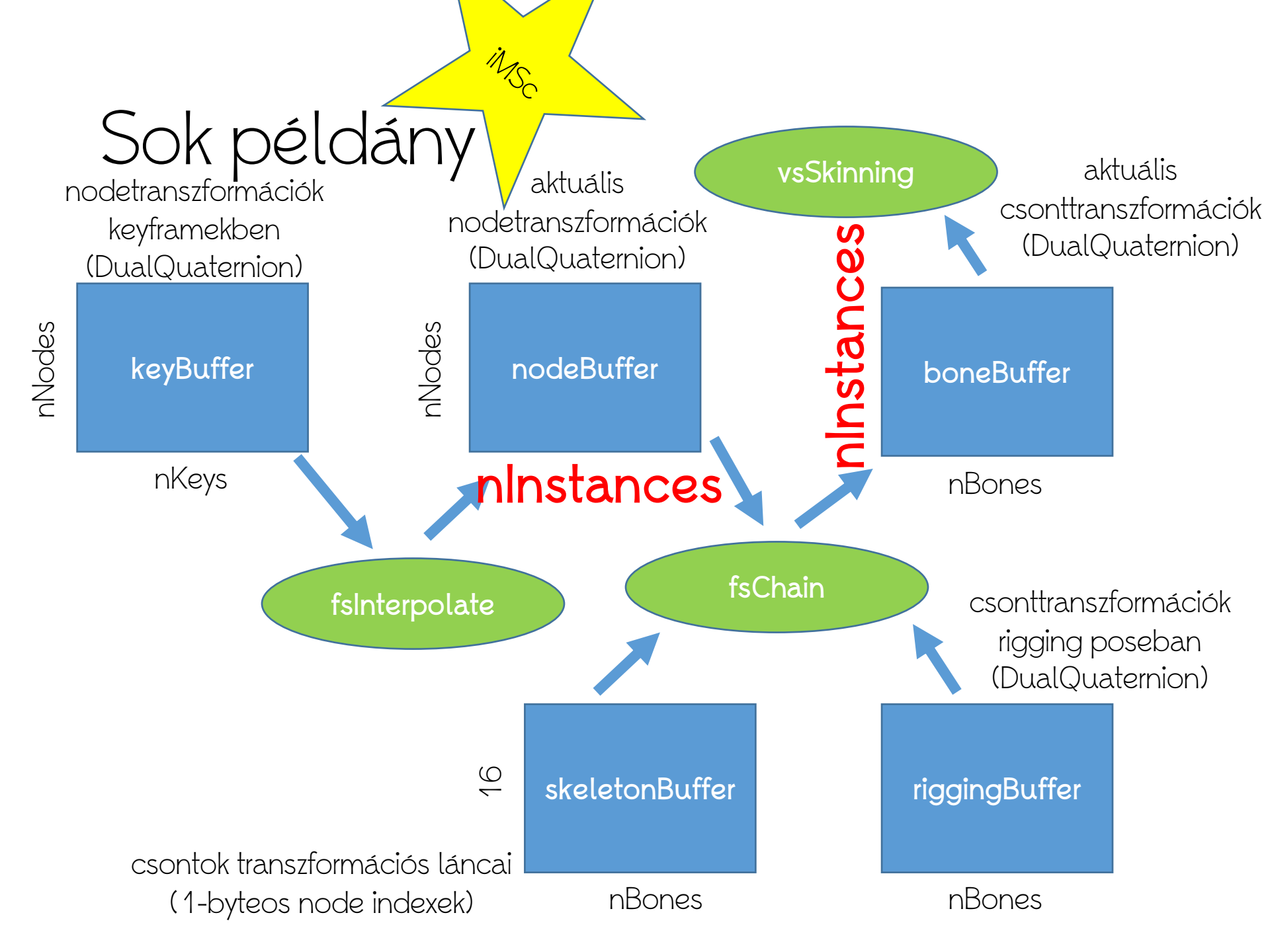
- mind a 4 blendindices elemhez olvassuk ki a trafót a textúrából
- adjuk őket össze a blendweights elemekkel súlyozva

- az eredményt normalizáljuk
  - a q-t és a t-t is osszuk a forgatáskvaternió hosszával

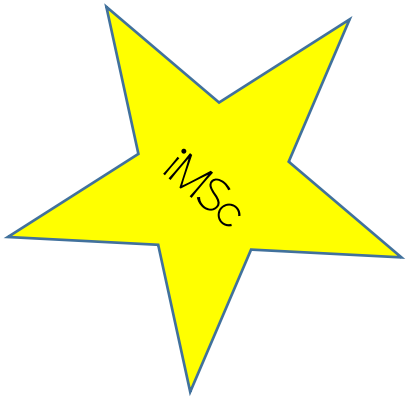
Podality fix

```
vec3 podality = vec3(
          (dot(dq0.q, dq1.q)>=0.0)?1.0:-1.0,
          (dot(dq0.q, dq2.q)>=0.0)?1.0:-1.0,
          (dot(dq0.q, dq3.q)>=0.0)?1.0:-1.0);

vec4 w = blendWeights;
w.w = 1.0-dot(w.xyz, vec3(1, 1, 1));
w.yzw *= podality;
```


iMSC
Sok példány
nodetranszformációk
keyframekben
(DualQuaternion)
aktuális
nodetranszformációk
(DualQuaternion)
vsSkinning
aktuális
csonttranszformációk
(DualQuaternion)
nNodes
keyBuffer
nKeys
nNodes
nodeBuffer
nInstances
nInstances
boneBuffer
nBones
fsInterpolate
fsChain
csonttranszformációk
rigging poseban
(DualQuaternion)
16
skeletonBuffer
csontok transzformációs láncai
(1-byteos node indexek)
nBones
riggingBuffer
nBones

Sok példány



- RiggedGeometry::draw
  - drawElementsInstanced(..., 32)
- fsInterplolate
  - texCoord.x kijelöli a példányt
  - példányazonosítótól függő időeltolás (eltérő animációs fázis)
- fsChain
  - texCoord.y kijelöli a példányt
  - nodeTexture címzésekor a példányazonosító adja az x-et
- vsSkinningben
  - float(gl_InstanceID)-től függ, hogy a boneTexture melyik sorából olvasunk
  - float(gl_InstanceID)-től függő pozícióeltolás
- várt eredmény:
  - 32, különböző fázisban táncoló zombi