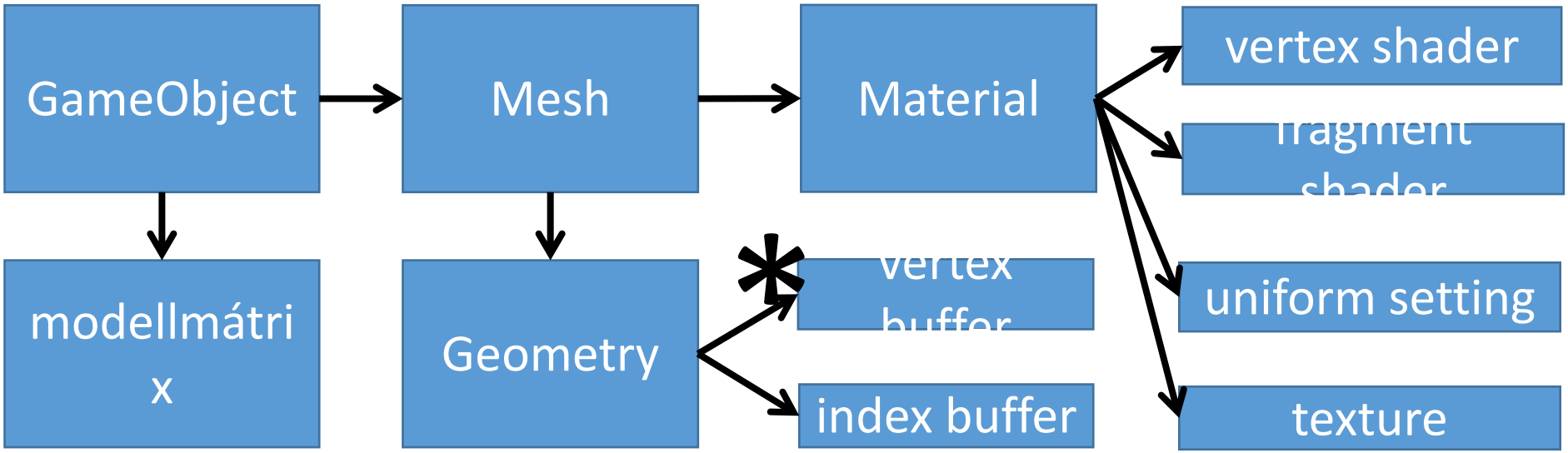
Háromszögháló-modellek

Szécsi László

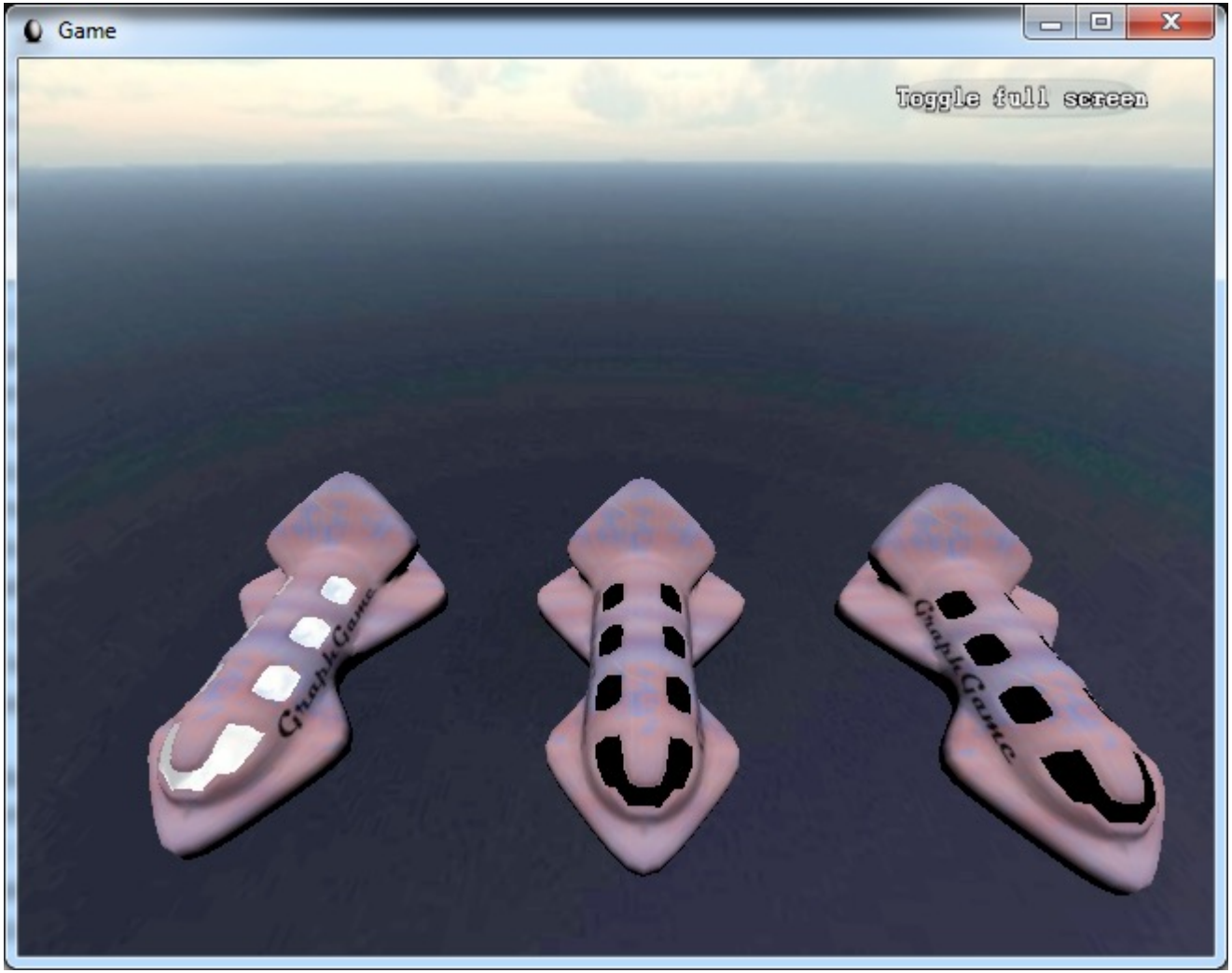
3D Grafikus Rendszerek

8. előadás

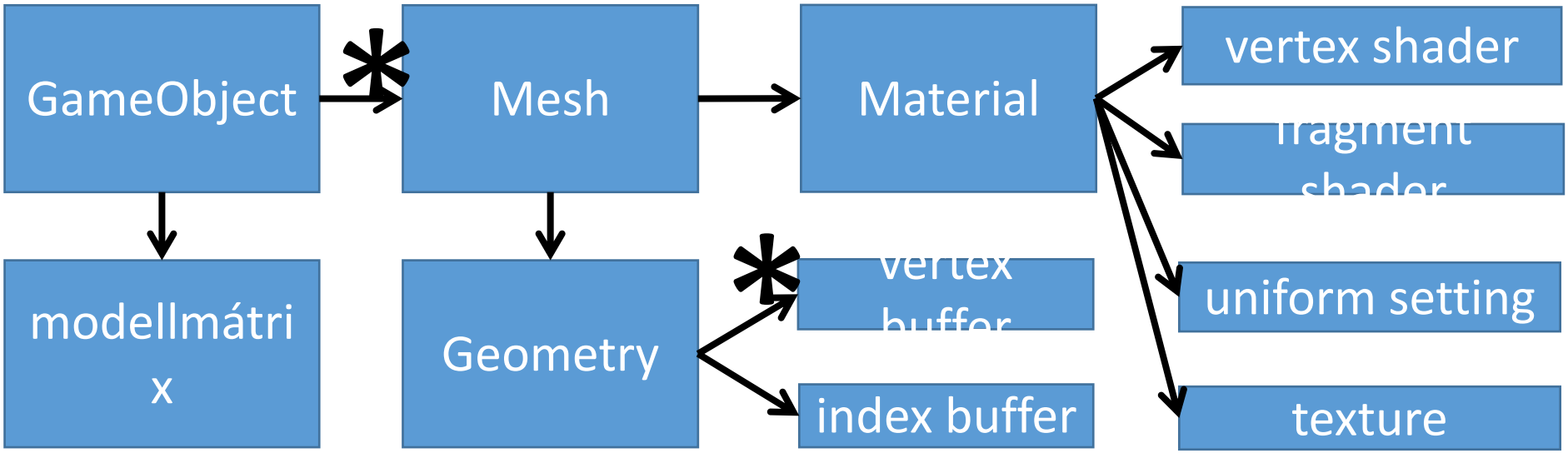
Játékobjektumok és geometriai modellek



Submesh



Játékobjektumok és geometriai modellek



Háromszögháló betöltése JSON-ból

- WebGL-barát formátum
- Kotlinból könnyen betölthető
- az adatok eleve abban a formában vannak, ahogy a csúcspont-bufferekbe be kell írni őket

- Blenderből lehet exportálni Python scripttel

Geometria JSON-ból

- `SubmeshGeometry.kt` a `TexturedQuadGeometry.kt` mintájára
  - a konstruktor kapjon egy `jsonMesh` nevű `JsonMesh` típusú mesh-leíró objektumot (ahogy az a JSON fileból jön). Ennek a következő tulajdonságai vannak:
    - `vertices` tulajdonság: 3n db koordináta folytonos tömbben
    - `normals` tulajdonság: 3n db érték folytonos tömbben
    - `texturecoords` tulajdonság: 2n db értéket tartalmazó folytonos tömbök tömbje (de tipikusan csak egy textúrakoordináta-készlet van)
    - `faces` tulajdonság: 3 indexet tartalmazó tömböcskék tömbje
      - egy folytonos tömböt lehet belőle gyártani (lásd későbbi dián)

Geometria JSON-ból

```
  gl.bindBuffer(GL.ARRAY_BUFFER,
vertexBuffer)
    gl.bufferData(GL.ARRAY_BUFFER,
      Float32Array(arrayOf<Float>(
    -1.0f, -1.0f, 0.5f,
    -1.0f, -1.0f, 0.5f,
    -1.0f, -1.0f, 0.5f,
    -1.0f, -1.0f, 0.5f
  )),
      GL.STATIC_DRAW)
```

jsonMesh.**vertices**

jsonMesh.normals

jsonMesh.texturecoords[0]

ezek mennek a tömbliterálok helyére
de a típusos
tömb gyártása marad

Index buffer

összefűzött tömb

a *draw* metódusban kelleni fog

```
val indexBuffer = gl.createBuffer()
val indexCount = jsonMesh.faces.flatten().size
init{
  val indexIterator = jsonMesh.faces.flatten().iterator()
  val indexArray =
       Array<Short>(indexCount) {indexIterator.next()}

gl.bindBuffer(GL.ELEMENT_ARRAY_BUFFER, indexBuffer)
  gl.bufferData(GL.ELEMENT_ARRAY_BUFFER,
    Uint16Array( indexArray ),
    GL.STATIC_DRAW)
}
```

Short array kell az Uint16Arraynek

JsonLoader.kt

```
import org.w3c.xhr.XMLHttpRequest
import org.w3c.dom.events.*
import kotlinx.serialization.*
import kotlinx.serialization.json.*
import vision.gears.webglmath.Geometry
@Serializable
data class JsonMesh(
    val vertices : Array<Float>,
    val normals : Array<Float>,
    val texturecoords : Array<Array<Float>>,
    val faces : Array<Array<Short>>)

@Serializable
data class JsonModel(
    val meshes : Array<JsonMesh>)
```

JsonLoader.kt

ebből az URL-ből

gyártsuk le ezeket

```
fun loadGeometries(gl : WebGL2RenderingContext,
    jsonModelFileUrl : String) : Array<Geometry> {
  val request = XMLHttpRequest()
  request.overrideMimeType("application/json")
  request.open("GET", jsonModelFileUrl, false)
  var geometries : Array<Geometry>? = null
  request.onloadend = {
    val json = Json { ignoreUnknownKeys=true }
    val jsonModel = json.decodeFromString(
                    JsonModel.serializer(), request.responseText)
    geometries = jsonModel.meshes.map{ SubmeshGeometry(gl, it) }
    Unit
  }
  request.send()
  return geometries!!
}
```

lista gyártása

mesh geometria ahogy a JSON-ben van

JsonLoader.kt

```
fun loadMeshes(
     gl : WebGL2RenderingContext,
     jsonModelFileUrl : String,
     vararg materials : Material) : Array<Mesh>{
   val geometries = loadGeometries(gl, jsonModelFileUrl)
   return (materials zip geometries).map{(m, g) -> Mesh(m, g)}.toTypedArray()
 }
```

ebből az URL-ből

gyártsuk le ezeket

Listából tömb

állítsuk párba az anyagokat és geometriákat

gyártsunk egy Mesht minden párból

Gyártsunk 3D játékobjektumot

- hozzuk létre az anyagokat mindkét submeshre
  - ugyanaz a program, más textúra
- game object létrehozása több meshsel

```
val jsonLoader = JsonLoader()
val slowpokeMeshes = jsonLoader.loadMeshes(gl,
  "media/slowpoke/slowpoke.json",
  Material(texturedProgram).apply{
    this["colorTexture"]?.set(
        Texture2D(gl, "media/slowpoke/YadonDh.png"))
  },
  Material(texturedProgram).apply{
    this["colorTexture"]?.set(
        Texture2D(gl, "media/slowpoke/YadonEyeDh.png"))
  }
)

init{
  gameObjects += GameObject(*slowpokeMeshes)
}
```