

Material, Mesh, GameObject

Szécsi László

3D Grafikus Rendszerek

2. labor

Komponensrendszer

- cél: uniformok kényelmes beállítása
 - vannak anyaghoz kötöttek
 - szín, textúra
 - vannak nem kötöttek (máshoz kötöttek)
 - transzformációk, animációs fázis
- visszavetítés
 - legyenek Kotlin változók, amiket beállíthatunk
 - egyesek a **Material** objektumokhoz tartoznak
 - másokat máshova (**GameObject**, **Scene**, **Camera**)
 - a **UniformProvider::draw** mindegyiket állítsa be magától

} **UniformProvider**
leszármazottak

Material osztály: visszavetített változók létrehozása

```
import vision.gears.webglmath.UniformProvider

class Material(program : Program) : UniformProvider("material")
{
    init {
        addComponentsAndGatherUniforms(program)
    }
}

// minden, a programban használt uniformra
// megfelelő típusú változó létrehozása és berakása ide:
UniformProvider::uniforms : Array<Uniform>
```

Pillantsuk be a gatherUniforms-ba [részlet]

létező
kód

```
//ProgramReflection-ben:
```

a célobjektum felelőségébe
tartozó uniformokra

```
override fun gatherUniforms(target : UniformProvider){  
    for(structName in target.glsStructNames) {  
        val descList = uniformDescriptors[structName] ?: continue  
        for(uniformDesc in descList) {  
            val reflectionVariable =  
ProgramReflection.makeVar(uniformDesc.type, uniformDesc.size)  
            target.uniforms[uniformDesc.name] = reflectionVariable  
        }  
    }  
}
```

nyerjük ki a uniform leírását

hozzunk létre illeszkedő típusú változót

adjuk hozzá a változót a
célobjektumhoz, azonos nével

```
//UniformProvider-ben:  
override fun gatherUniforms(  
    target : UniformProvider){  
    components.forEach {  
        it.gatherUniforms(target)  
    }  
}
```



A shaderben nem használt uniformok

- kioptimalizálva
- nincs visszavetítve
- nincs hozzá változó a **Material**ban
- tehát lehet **null**
 - ilyenkor figyelmeztetés íródik ki
 - nem szeretnénk hibát kapni



```
val material = Material(solidProgram)
init {
    material["solidColor"]?.set(1.0f, 1.3f, 0.8f)
    material["noSuchProp"]?.set(0.0f, 0.3f, 0.8f)
}
```

safe access, hogy null esetén ne legyen hiba, semmi se történjen

Pillantsunk be a ProgramReflection::draw-ba [részlet]

létező kód

```
override fun draw(vararg uniformProviders : UniformProvider) {  
    gl.useProgram(glProgram)  
    for(provider in uniformProviders){  
        for(structName in provider.glslStructNames) {  
            val descList = uniformDescriptors[structName] ?: continue  
            for(uniformDesc in descList) {  
                provider[uniformDesc.name]!!.commit(gl,  
                    uniformDesc.location)  
            }  
        }  
    }  
}
```

minden, uniformokat adó komponensre

az ő felelősségi körébe tartozó uniformokra

nyerjük ki a uniformok leírását

töltsük fel az adatot a uniformba az azonos nevű változóból

Hogyan használjuk az anyagrendszert?

```
//in Scene constructor:
```

```
val raiderMaterial = Material(texturedProgram).apply{  
    this["colorTexture"]?.set(  
        Texture2D(gl, "media/raider.png"))  
    this["texOffset"]?.set(0.1, 0.4)  
}
```

sampler2D uniform a shaderből

vec2 uniform a shaderből

csak példa

```
//in Scene::update:
```

```
// egyelőre marad: use program, gl.uniform* az  
anyagon kívüli uniformokra (pl. modelMatrix)  
material.draw()  
quadGeometry.draw()
```

semmi uniformLocation,
semmi uniform2fv

Feladat: használja a **Material**-t

- legyen két különböző anyag
 - pl. **asteroidMaterial**, **raiderMaterial**
- ugyanazzal a programmal
- de eltérő uniform értékekkel
 - pl. más **colorTexture**
- rajzolja ugyanazt a geometriát kétszer, de különböző anyaggal, és különböző **modelMatrix** beállítással (amit egyelőre állítsunk a korábbi módon)

Mesh osztály

(Mesh = Geometry & Material)

```
import vision.gears.webglmath.UniformProvider
import vision.gears.webglmath.Geometry

class Mesh(material : Material, geometry : Geometry)
    : UniformProvider("mesh") {

    init{
        addComponentsAndGatherUniforms(
            material, geometry)
    }
}
```

per-mesh uniform nem tipikus, de elképzelhető mindenesetre lehetséges

gyerek-komponensek

Feladat: használja a **Mesh**-t

- gyártson két különböző mesh-t
 - pl. **raiderMesh**, **asteroidMesh**
- ugyanazzal a geometriával (ne legyen kettő példány)
- de különböző anyaggal
 - pl. **raiderMaterial**, **asteroidMaterial**
- rajzolja a mesheket eltérő **modelMatrix** beállításokkal (még mindig: **useProgram**, **getUniformLocation**, **commit**)
- így két sor helyett (**material.draw** és **geometry.draw**) lett egy (hurrá!)

Opcionális

GameObject osztály

```
import vision.gears.webglm.math.UniformProvider
import vision.gears.webglm.math.Vec3
import vision.gears.webglm.math.Mat4
import vision.gears.webglm.math.Vec3Array

class GameObject( vararg meshes : Mesh
) : UniformProvider("gameObject") {
    val position = Vec3()
    var roll = 0.0f ← forgatási szög z tengely körül
    val scale = Vec3(1.0f, 1.0f, 1.0f)
    init {
        addComponentsAndGatherUniforms(*meshes)
    }
}
```

GameObject::modelMatrix

```
class GameObject {
```

```
    val modelMatrix by Mat4()
```

A uniform-gyűjtéskor, ha a shaderben használjuk, keletkezne egy változó, amit pl. `avatar["modelMatrix"]`-ként el is érünk.

Helyette szeretnénk, hogy a már létező `modelMatrix` property játssza ezt a szerepet. A fenti *property delegation* ezt oldja meg: a `Mat4 provideProperty` operátora beteszi a property-t a uniformok közé.

```
    init {  
        addComponentsAndGatherUniforms(mesh)
```

```
    }
```

```
}
```

GameObject::update

```
class GameObject {  
    // ezt a metódust minden frameben meg fogjuk hívni  
    // mozgásra és a modelMatrix kiszámítására  
    fun update() {  
        // feladat: modelMatrix property beállítása  
        // scale, position, roll alapján  
        //  
        // Mat4::set()  
        // paraméter nélkül egységmatrixot állít be  
        //  
        // rotate() hozzászoroz egy elforgatásmatrixot  
        // translate(), scale() hasonlóan működik  
        // SORREND A KOMMENTBEN DIREKT VAN KEVERVE  
    }  
}
```

Feladat: használja a **GameObject**-et

```
gameObjects +=  
    GameObject(raiderMesh).apply{  
        position.set(0.5f, 0.5f)  
    }
```

- a **Scene** konstruktorban hozzon létre egy tömböt
`val gameObjects = ArrayList<GameObject>()`
- hozzon létre pár **GameObject**-et a meglevő meshekkel
- adjuk őket a **gameObjects** tömbhöz (vagy +=)
- a **Scene::update**-ben hívjuk meg az **update**-et minden **GameObject**-re
- a **Scene::update**-ben hívjuk meg a **draw**-t minden **GameObject**-re (ezt úgy örökölte)
- a **Scene::update** most már mást nem rajzol
 - de animálni animálhat, és a képet törölheti

Animáció

- játékobjektumok különböző **move** metódusokkal
 - leszármaztatás helyett delegálás egy **GameObject.Motion** osztálynak
 - ennek lehetnek leszármazottai
 - nem kell explicit új osztályokat gyártani feltétlenül, lehet névtelen is (object expression)
- a **Scene::update** hívja mindegyik játékobjektum **move**-jét

GameObject.Motion

```
class GameObject {  
  
    open class Motion(val gameObject : GameObject) {  
        open operator fun invoke(  
            dt : Float = 0.016666f,  
            t : Float = 0.0f,  
            keysPressed : Set<String> = emptySet<String>(),  
            interactors : List<GameObject> = emptyList<GameObject>(),  
            spawn : List<GameObject> = emptyList<GameObject>()  
        ) : Boolean {  
            return true  
        }  
    }  
    var move = Motion(this)  
    lehessen alosztály, override  
}
```

akkor hamis,
ha az
objektumot
törölni kell

Scene, motion példa

új property

```
val avatar = GameObject(raiderMesh).apply{
  move = object : GameObject.Motion(this){
    val velocity = Vec3(0.1f, 0.1f)
    override operator fun invoke( ... ) : Boolean {
      gameObject.position += velocity * dt
      return true
    }
  }
}

init {
  gameObject += avatar
}
```

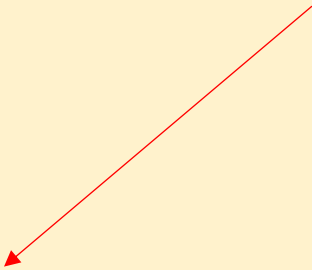
helyben definiált és
példányosított névtelen
GameObject.Motion-alsztály
(Java-style)

animációs logikát
megvalósító metódus

Scene

```
fun update(gl : WebGL2RenderingContext, keyPressed : Set<String>) {  
    val spawn = ArrayList<GameObject>()  
    val killList = ArrayList<GameObject>()  
    gameObjects.forEach {  
        if(!it.move( ... )){  
            killList.add(it)  
        }  
    }  
    killList.forEach{ gameObjects.remove(it) }  
    spawn.forEach{ gameObjects.add(it) }  
    gameObjects.forEach{ it.update() }  
    gameObjects.forEach{ it.draw() }  
}
```

megszűnő objectek
eltávolítása



újonnan létrehozott
objectek hozzáadása



Feladat

- magától forgó játékobjektum
- különböző sebességekkel, de egyenes vonalban egyenletesen mozgó játékobjektumok
- gombokkal forgatható játékobjektum
- gombokkal mozgatható játékobjektum

OrthoCamera

```
class OrthoCamera() : UniformProvider("camera") {  
    val position = Vec2(0.0f, 0.0f)  
    var roll = 0.0f  
    val windowSize = Vec2(2.0f, 2.0f)  
  
    val viewProjMatrix by Mat4()  
    init{  
        updateViewProjMatrix()  
    }  
}
```

OrthoCamera:: updateViewProjMatrix

```
fun updateViewProjMatrix() {  
    viewProjMatrix.set().  
        scale(0.5f, 0.5f, 1.0f).  
        scale(windowSize).  
        rotate(roll).  
        translate(position).  
        invert()  
}  
  
fun setAspectRatio(ar : Float) {  
    windowSize.x = windowSize.y * ar  
    updateViewProjMatrix()  
}
```

Használja a kamerát!

- vegyen fel a színtérbe egy kamerát
- **resize** metódusban **camera.setAspectRatio**
 - **toFloat()** jól jön
- vertex shader használja a **camera.viewProjMatrix** uniformot
 - transzformálja vele a világkoordinátás pozíciót
- **gameObject**tek rajzolásakor a **draw** metódusnak adjuk paraméterül a **camera**
 - mivel ő adja a uniformértéket

Feladat

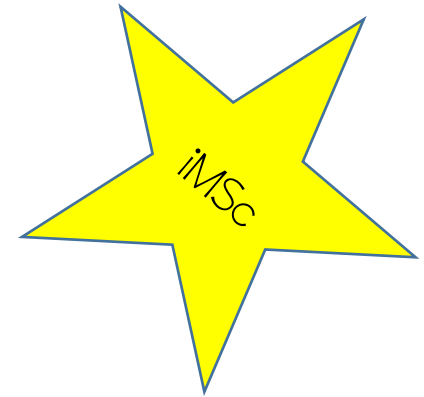
- a kamera kövesse az egyik objektumot
 - a kamera pozícióját kell minden képkockában átállítani

Bónusz feladat: eltűnő objektumok



- ha a `GameObject::move` `false`-ot ad vissza, az objektumot dobjuk ki a `gameObjects`-ből
 - ne azonnal, csak ha már minden `move` lement
- legyen egy játékobjektum, aki ha pl. a világ jobb oldalára téved ($x > 0$), megsemmisül

Bónusz feladat: scrollozó háttérkép



- 1db háttérobjektum
- geometria: teljes képernyős téglalap
- sima textúrázó FS
 - textúra: bármilyen kép (legyen 2-hatvány x 2-hatvány)
- speciális VS
 - pozíciót nem bántja
 - de számolja a világkoordinátát a képernyőkoordinátából
 - ehhez a kamera view matrixának inverzét uniformban megkapja a kamerától
 - ezzel szorozza a **vertexPosition**-t
 - a kapott világkoordináta (* freki) lesz a textúrákoordináta