Mesh from file, PerspectiveCamera, CubeTexture

Szécsi László

3D Grafikus Rendszerek

3. labor

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 **Uint16Array**nek

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

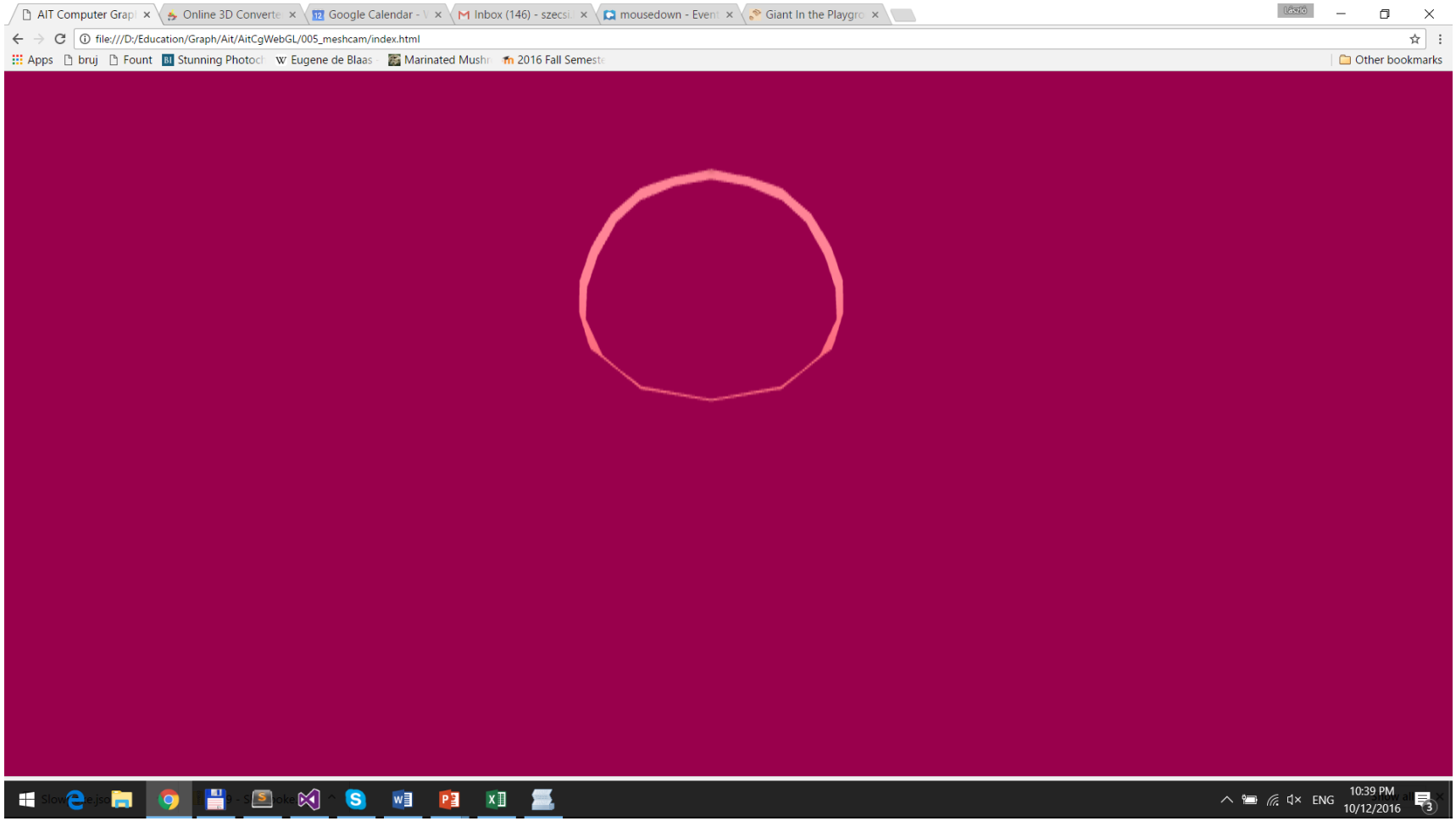
Gyártsunk 3D játékobjektumot

- 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)
}
```

Várt eredmény (2D kamerától is függ)



3D kamera

- `PerspectiveCamera` class
- view, proj mátrixok beállítása
- egérrel és WASD billentyűkkel mozgatható

PerspectiveCamera.kt – import

```
import vision.gears.webglmath.UniformProvider
import vision.gears.webglmath.Vec2
import vision.gears.webglmath.Vec3
import vision.gears.webglmath.Mat4
import kotlin.math.tan
import org.w3c.dom.events.*
```

PerspectiveCamera.kt – kameraparaméterek

```
class PerspectiveCamera(vararg programs : Program) :
UniformProvider("camera") {

  val position by Vec3(0.0f, 0.0f, 0.0f)
  var roll = 0.0f
  var pitch = 0.0f
  var yaw = 0.0f

  var fov = 1.0f
  var aspect = 1.0f
  var nearPlane = 0.1f
  var farPlane = 1000.0f
```

`PerspectiveCamera.kt` – számítható értékek

```
var ahead = Vec3(0.0f, 0.0f,-1.0f)
var right = Vec3(1.0f, 0.0f, 0.0f)
var up    = Vec3(0.0f, 1.0f, 0.0f)

val rotationMatrix = Mat4()
val viewProjMatrix by Mat4()
val rayDirMatrix by Mat4()

companion object {
  val worldUp = Vec3(0.0f, 1.0f, 0.0f)
}
```

forgatási szögekből számítható

`PerspectiveCamera.kt` – mozgás

```
var speed = 0.005f
var isDragging = false
val mouseDelta = Vec2(0.0f, 0.0f)
```

PerspectiveCamera.kt – init

```
init {
  update()
  addComponentsAndGatherUniforms(*programs)
}
```

`PerspectiveCamera.kt` – update, view

```
fun update() {
  rotationMatrix.set().
    rotate(roll).
    rotate(pitch, 1.0f, 0.0f, 0.0f).
    rotate(yaw, 0.0f, 1.0f, 0.0f)
  viewProjMatrix.set(rotationMatrix).
    translate(position).
    invert()
```

`PerspectiveCamera.kt` – update, proj

```
val yScale = 1.0f / tan(fov * 0.5f)
val xScale = yScale / aspect
val f = farPlane
val n = nearPlane
viewProjMatrix *= Mat4(
    xScale ,    0.0f ,          0.0f ,   0.0f,
      0.0f ,  yScale ,          0.0f ,   0.0f,
      0.0f ,    0.0f ,  (n+f)/(n-f) ,  -1.0f,
      0.0f ,    0.0f ,  2*n*f/(n-f) ,   0.0f)
```

`PerspectiveCamera.kt` – update, rayDirMatrix

```
    // önállóan megoldandó feladat
    // de ráér
    // az env mapping háttérhez kell csak
}
```

`PerspectiveCamera.kt` – aspect ratio
(már a 2D kamerának is volt, meg is van hívva)

```
fun setAspectRatio(ar : Float) {
  aspect = ar
  update()
}
```

PerspectiveCamera.kt – move

```
fun move(dt : Float, keysPressed : Set<String>) {
    if(isDragging) {
      yaw -= mouseDelta.x * 0.002f
      pitch -= mouseDelta.y * 0.002f
      if(pitch > 3.14f/2.0f) { pitch = 3.14f/2.0f }
      if(pitch < -3.14f/2.0f) {  pitch = -3.14f/2.0f }
      mouseDelta.set()
    }
    if("W" in keysPressed) { position += ahead * (speed * dt) }
    if("S" in keysPressed) { position -= ahead * (speed * dt)  }
    if("D" in keysPressed) { position += right * (speed * dt) }
    if("A" in keysPressed) { position -= right * (speed * dt) }
    if("E" in keysPressed) { position += up * (speed * dt) }
    if("Q" in keysPressed) { position -= up * (speed * dt) }

    update()
    ahead = (Vec3(0.0f, 0.0f, -1.0f).xyz0 * rotationMatrix).xyz
    right = (Vec3(1.0f, 0.0f,  0.0f).xyz0 * rotationMatrix).xyz
    up    = (Vec3(0.0f, 1.0f,  0.0f).xyz0 * rotationMatrix).xyz
  }
```

`PerspectiveCamera.kt` – egéreseményekFELADAT: meg is kell hívni őket

```
  fun mouseDown() {
    isDragging = true
    mouseDelta.set()
  }

  fun mouseMove(event : MouseEvent) {
    mouseDelta.x += event.asDynamic().movementX as Float
    mouseDelta.y += event.asDynamic().movementY as Float
    event.preventDefault()
  }
  fun mouseUp() {
    isDragging = false
  }
}
```

Rakjuk össze!

- orthokamera lecserélése
- mélységteszt bekapcsolása
  - `gl.enable(GL.DEPTH_TEST)`
- eseményfigyelők bekötése
- minden frameben `camera.move()`

- `GameObject` 3D-ben is ugyanúgy jó
  - orientáció maradhat csak 2D, egyelőre

rayDirMatrix számítása

```
fun update() {
  // önállóan megoldandó feladat
  // már nem ér rá annyira
  // az env mapping háttérhez kell most

  // képlet a következő dián
}
```

Sugárirány kiszámítása NDC-ből

pont a világban

pont a képernyőn

$$\boldsymbol{x}_{\mathrm{w}}\boldsymbol{VP} = \boldsymbol{x}_{\mathrm{ndc}}$$

sugárirány (normalizálatlan)

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{w}} - \boldsymbol{e}$$

sugárirány (normalizált)

$$\hat{\boldsymbol{d}} = \frac{\boldsymbol{d}}{|\boldsymbol{d}|}$$

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{ndc}}(\boldsymbol{VP})^{-1} - \boldsymbol{e}$$

szempozíció

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{ndc}}(\boldsymbol{VP})^{-1}\boldsymbol{E}^{-1}$$

szempozícióval eltolás mátrixa

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{ndc}}(\boldsymbol{EVP})^{-1}$$

(**EVP**)$^{-1}$ -et nevezzük rayDirMatrix-nak

Környezet megjelenítése háttérként

- teljes képernyős téglalapot kell rajzolni (hurrá!)
- új VS: kiszámolja a sugárirányt
  - használni kell képernyőkoordinátából-világkoordináta-mínusz-szempozíció-számító mátrixot (a.k.a. `rayDirMatrix`)
    - a kamera ezt kiszámolja és a `camera.rayDirMatrix` uniformba tölti (ha van ilyen deklaráva és használva)
  - nem transzformál (mert full viewport quad)
  - z=0.99999, minden mögé
- FS megkapja a VS-től a sugárirányt
  - ezzel címzi a textúrát
  - visszaadja a kapott színt

Doboztextúra

- kell a FS-ben egy uniform

```
// kell egy sampler uniform
uniform struct { samplerCube envTexture; } material;
```

```
// kiolvasni a sugárirányban
fragmentColor = texture ( material.envTexture, rayDir.xyz);
```

- **Scene**-ben gyártsuk le a **TextureCube**-ot, kössük be a fenti FS-t használó **Material**-ba, a **GameObject** használja ezt az anyagot

```
val envTexture = TextureCube(gl,
    "media/posx512.jpg",
    "media/negx512.jpg",
    "media/posy512.jpg",
    "media/negy512.jpg",
    "media/posz512.jpg",
    "media/negz512.jpg"
    )
```

```
backgroundMaterial["envTexture"]?.set( this.envTexture )
```

TextureCube.kt – mint a Texture2D

```
import org.khronos.webgl.WebGLRenderingContext as GL
import org.khronos.webgl.WebGLTexture
import org.w3c.dom.Image
import org.w3c.dom.events.Event
import vision.gears.webglmath.Texture

class TextureCube(
  gl : WebGL2RenderingContext,
  vararg mediaFileUrls : String
  ) : Texture {
  override val glTexture : WebGLTexture? = gl.createTexture()
```

`TextureCube.kt` – mint a `Texture2D`, de hat képre

```
  init {
    val images = Array<Image>(6) { Image() }
    var loadedCount = 0
    for(i in 0 until 6) {
      images[i].onload = {
        gl.bindTexture(GL.TEXTURE_CUBE_MAP, glTexture)
        gl.texImage2D(GL.TEXTURE_CUBE_MAP_POSITIVE_X+i, 0, GL.RGBA, GL.RGBA,
                                                    GL.UNSIGNED_BYTE, images[i])
        if(++loadedCount == 6) {
          gl.texParameteri(GL.TEXTURE_CUBE_MAP, GL.TEXTURE_MAG_FILTER, GL.LINEAR)
          gl.texParameteri(GL.TEXTURE_CUBE_MAP, GL.TEXTURE_MIN_FILTER,
                                                        GL.LINEAR_MIPMAP_LINEAR)
          gl.generateMipmap(GL.TEXTURE_CUBE_MAP);
        }
        gl.bindTexture(GL.TEXTURE_CUBE_MAP, null);
      }
      images[i].src = mediaFileUrls[i]
    }
  }
}
```

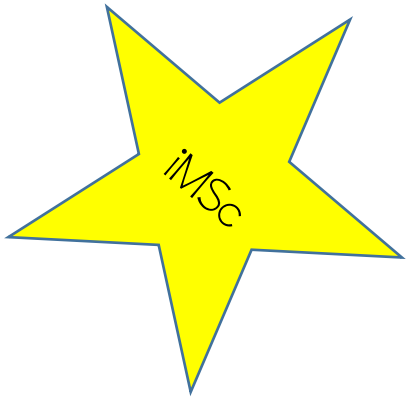
Ne felejtsük el

- gyártsuk le szükséges `shader`, `program`, `material` objektumokat
- kössük be a háttér anyagába a doboztextúrát
- legyen `quadGeometry`
- legyen egy `mesh` a `quadGeometry` és a fenti `material` használatával
- legyen egy `gameObject` a `mesh`-sel

Várt eredmény

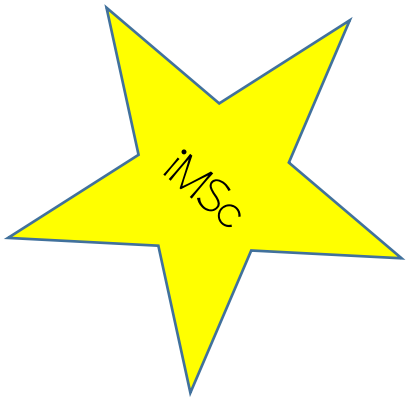
- textúrázott objektum és háttér

Végtelen sík



- új geometria-típus
- 4 koordináta per vertex
  - attribute bindingban is írjuk át
- egy vertex az origóban
- három vertex ideális pontokban körben
- három háromszög legyezőszerűen

Végtelen sík textúrázása



- homogén textúra-koordináták
- 4 textúra-koordináta per vertex
  - értékek **azonosak a pozíciókkal**, de y-t és z-t cseréljük, ha vízszintes a sík
    - a harmadikat a négyből nem fogjuk használni, de egyszerűbb vec4-et használni, mint vec3-at, mert így a meglevő geometriák is működnek majd az átírt shaderekkel
  - attribute bindingban is írjuk át
- shaderekben a textúra-koordináta `vec4`
- homogén osztás a FS-ben, mielőtt címeznénk vele
  - `tex.xy/tex.w`