Árnyalás

Szécsi László

3D Grafikus Rendszerek

4. labor

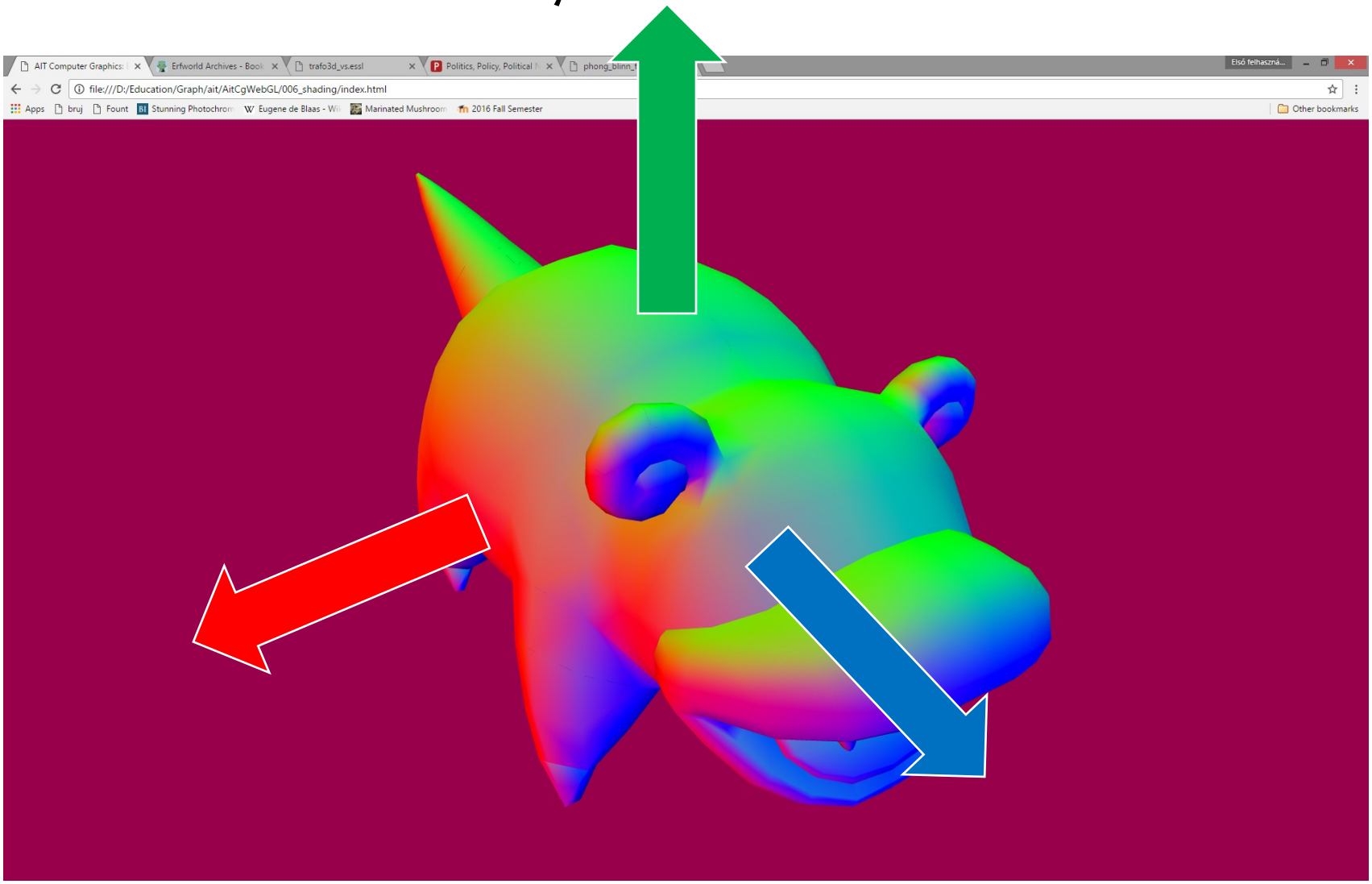
`trafo-vs.glsl`: 3D transzformáló vertex shader

- marad a pozíció szorzása `gameObject.modelMatrix`szal és `camera.viewProjMatrix`szal
- új kimenetek:
  - `vec4 modelPosition = vertexPositon;` egy-az-egyben
  - `vec4 worldPosition` a `vertexPositon` csak a `gameObject.modelMatrix`szal szorozva
- új uniform és `GameObject` property: `gameObject.modelMatrixInverse`
  - normálvektorok transzformációjához
  - balról szorozzuk a mátrixszal a `vec4(vertexNormal, 0)`-t
  - új kimenet: `vec4 worldNormal`

`envmapped-fs.glsl`: Normálmegjelenítő fragment shader

- adjuk át a világtérbeli normálvektort a VS-ből az FS-be (kimenet és bemenet kell)
- normalizáljuk a `worldNormal`-t
  - `vec3 normal = normalize(worldNormal.xyz);`
- adjunk vissza `vec4(abs(normal), 1)`–et mint színt

Várt eredmény



Environment mapping

- FS (ezt használó **`Program`** és **`Material`** is kell)

```
// kell a környezet
uniform struct{ samplerCube envmapTexture; } material;
```

```
// árnyalt felületi pont pozíciója
vec3 x = worldPosition.xyz / worldPosition.w;
vec3 viewDir = normalize(camera.position - x);
```

```
// olvassuk ki a környezetet a tükörirányban
texture( material.envmapTexture, reflect(-viewDir, normal))
```

- állítsuk be a textúrát az anyagnak, hozzunk létre Mesht (vagy MultiMesht) és játékobjektumot

```
val skyCubeTexture =
  TextureCube(gl,
    "media/posx.jpg", "media/negx.jpg",
    "media/posy.jpg", "media/negy.jpg",
    "media/posz.jpg", "media/negz.jpg"
    )
```

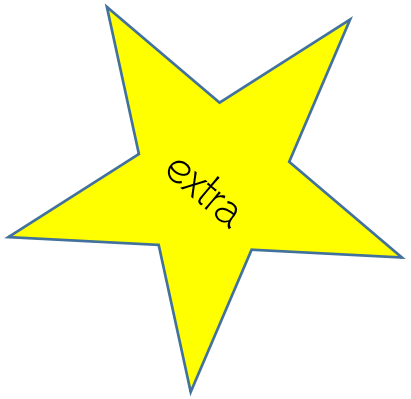
```
envmappedMaterial["envmapTexture"]?.
set(skyCubeTexture)
```

Várt eredmény

kameramozgatáskor változik a tükröződés,

nincsenek a színek a felületre rögzítve

Procedural normal mapping



- felület implicit egyenlete $f(\boldsymbol{r}) = 0$
  - a felület most nem egyenlettel adott, szóval ez ismeretlen
- felület normálvektora az implicit felület gradiense
  $\hat{\boldsymbol{n}} = [\nabla f(\boldsymbol{r})]^{\wedge}$
  - a gradiens ismeretlen, a normálvektorok viszont adottak
- keverjünk a függvényhez hozzá zajt: $A\xi(\nu\boldsymbol{r})$
- ennek normálvektora

amplitúdó

freki

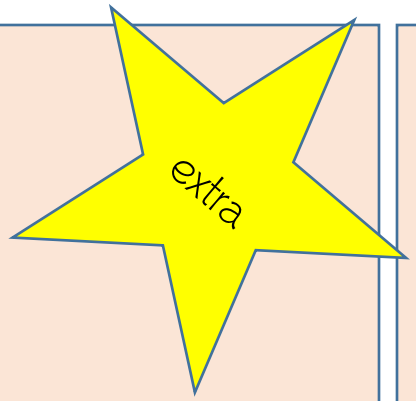
$$\hat{\boldsymbol{n}}_{\text{perturbed}} = [\nabla (f(\boldsymbol{r}) + A\xi(\nu\boldsymbol{r}))]^{\wedge} = [\nabla f(\boldsymbol{r}) + A\nabla\xi(\nu\boldsymbol{r})]^{\wedge}$$

- tehát végül csak ennyit kell implementálni a FS-ben:

$$\hat{\boldsymbol{n}}_{\text{perturbed}} = [\hat{\boldsymbol{n}} + A\nabla\xi(\nu\boldsymbol{r})]^{\wedge}$$

A és ν konstansok alkalmasan választva.
A zajfüggvény gradiense a következő dián.

Egyszerű zajfüggvény és gradiense




```
float noise(vec3 r) {
  uvec3 s = uvec3(
    0x1D4E1D4E,
    0x58F958F9,
    0x129F129F);
  float f = 0.0;
  for(int i=0; i<16; i++) {
    vec3 sf =
    vec3(s & uvec3(0xFFFF))
  / 65536.0 - vec3(0.5, 0.5, 0.5);

    f += sin(dot(sf, r));
    s = s >> 1;
  }
  return f / 32.0 + 0.5;
}
```

```
vec3 noiseGrad(vec3 r) {
  uvec3 s = uvec3(
    0x1D4E1D4E,
    0x58F958F9,
    0x129F129F);
  vec3 f = vec3(0, 0, 0);
  for(int i=0; i<16; i++) {
    vec3 sf =
    vec3(s & uvec3(0xFFFF))
  / 65536.0 - vec3(0.5, 0.5, 0.5);

    f += cos(dot(sf, r)) * sf;
    s = s >> 1;
  }
  return f;
}
```

Egyszerű árnyaló FS `maxblinn-fs.glsl`-ben lehet dolgozni

- definiáljunk egy fényirányt
  - felületről a fény felé mutat
  - egyelőre bedrótozott konstans, de **normalizálva**
  - később majd uniformra cseréljük
- számítsuk a fényirány és a normál közötti szög koszinuszát
  - a `dot` beépített függvénnyel
- mi legyen, ha a koszinusz negatív?
- adjuk vissza a textúraszínt, ezzel a koszinusszal szorozva
  - csak az rgb-t szorozzuk, az alfa maradjon átlátszatlan

Fények adatai uniformokban

```
uniform struct {
  vec4 position;
  vec3 powerDensity;
} lights[8];
```

```
import vision.gears.webglmath.UniformProvider
import vision.gears.webglmath.Vec3
import vision.gears.webglmath.Vec4

class Light(id : Int, vararg programs : Program) :
UniformProvider("lights[$id]") {

  val position by Vec4(0.0f, 1.0f, 0.0f, 0.0f)
  val powerDensity by Vec3(0.0f, 0.0f, 0.0f)

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

Fényforrások beállítása a `Scene`-ben

- konstruktor

fényforrások száma
legfeljebb uniform tömbméret

tömbindex

```
val lights = Array<Light>(1) { Light(it, *Program.all) }
init{
  lights[0].position.set(1.0f, 1.0f, 1.0f, 0.0f).normalize();
  lights[0].powerDensity.set(1.0f, 1.0f, 0.0f);
}
```

fényirány legyen egységhosszú

- `update`

```
gameObjects.forEach { it.draw( camera, *lights ) }
```

uniformokat adó komponensek átadása

Több színes fényforrás

```
vec3 shade(
  vec3 normal, vec3 lightDir,
  vec3 powerDensity, vec3 materialColor) {

  return
    árnyalt szín;
}
```

visszaverődési modell

- állítsunk be legalább két fényforrást, eltérő irányokkal
- eltérő teljesítménysűrűség-spektrumokkal
- a FS-ben írjunk egy **for** ciklust, ami összegzi a fényforrások hozzájárulásait a felület színéhez

```
vec3 lightDir = lights[i].position.xyz;
vec3 powerDensity = lights[i].powerDensity;

fragmentColor.rgb += shade(normal,
  lightDir, powerDensity,
  texture(material.colorTexture, tex.xy/tex.w).rgb);
```

fényforrásmodell

Várt eredmény

- textúrázott modell megvilágítva
- de a fény adatai Kotlinból állíthatók

Pontfényforrások de az irányfényforrások is működjenek

```
vec3 lightDiff = ???;
vec3 lightDir = ???;
vec3 powerDensity = ???;
```

a fény "színe"
a pontfény teljesítménye / 4 pi
VAGY
az irányfény
teljesítménysűrűsége

a felület "színe"
vagyis a diffúz visszaverődési tényező

negatív nem lehet,
helyette nulla

$$\ddot{L}_{\text{eye}}(x) = \frac{\ddot{M}}{|y - x \cdot w|^2} \circ \ddot{k}_d \left((y - x \cdot w)^\wedge \cdot \hat{n} \right)^+$$

a felületi pont
látható "színe"

fény pozíciója — 1 pontfényre
VAGY
fény iránya (egységhosszú) — 0 irányfényre

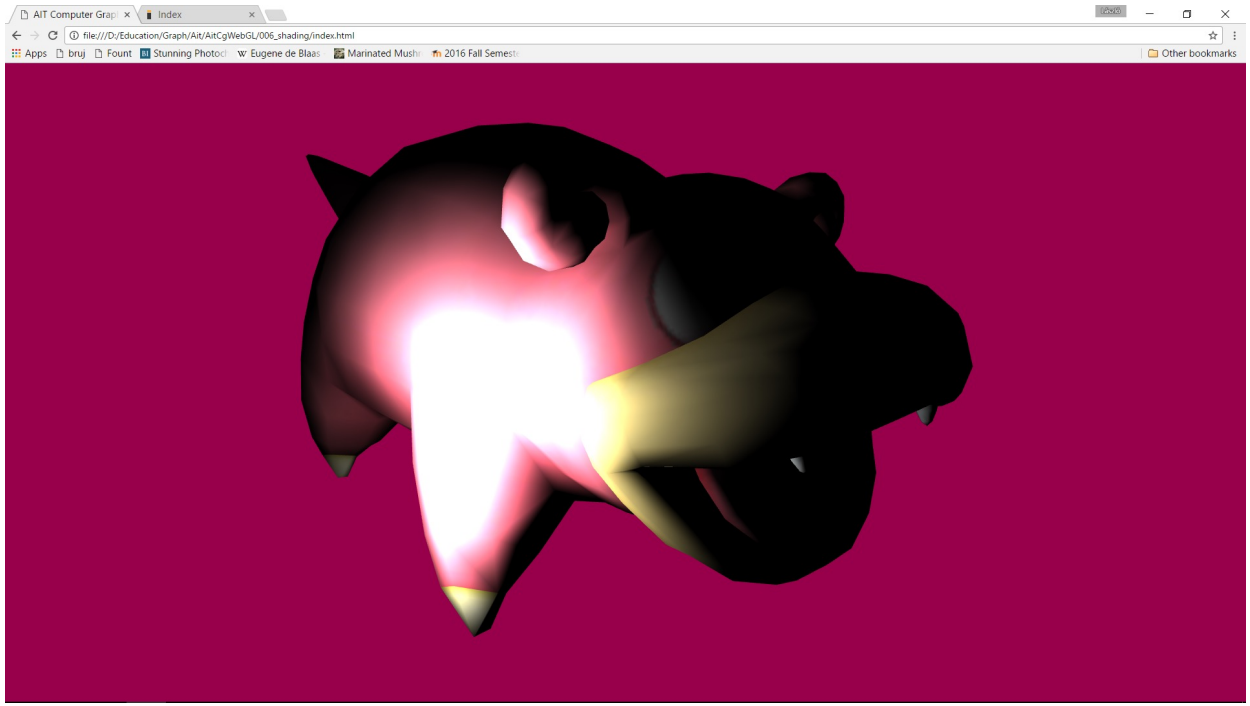
a fény beesési
szögének
koszinusza, ha
pozitív

a fényirány és
felületi normális
skalárszorzata

- a **shade** függvény nem változik
  - a visszaverődési modell ugyanaz

Feladat

- egy pontfényforrás keringjen a tárgy körül



Phong-Blinn visszaverődési modell

az árnyalt pontból a kamera felé, egységvektor

```
vec3 shade(
  vec3 normal, vec3 lightDir, vec3 viewDir,
  vec3 powerDensity, vec3 materialColor, vec3 specularColor, float shininess) {

  float cosa = ???;
  vec3 halfway = ???;
  float cosDelta = ???;

  return
    powerDensity * materialColor * cosa
  + powerDensity * ???;
}
```

anyagi jellemzők

$$\vec{L} = \vec{M} \circ \vec{k}_s \left((\hat{n} \cdot \hat{h})^+ \right)^\gamma$$

Maximum Blinn visszaverődési modell

```
vec3 shade(
  vec3 normal, vec3 lightDir, vec3 viewDir,
  vec3 powerDensity, vec3 materialColor, vec3 specularColor, float shininess) {

  float cosa = ???;
  float cosb = clamp(dot(viewDir, normal), 0, 1);

  vec3 halfway = ???;
  float cosDelta = ???;

  return
    ???
  + powerDensity * ???
       * cosa / max(cosb, cosa);
}
```

$$\vec{L} = \vec{M} \circ \vec{k}_s \cos^{\gamma} \delta \frac{\cos\theta'}{\max(\cos\theta, \cos\theta')}$$

$(\hat{n} \cdot \hat{l})^+$

$(\hat{n} \cdot \hat{v})^+$

Render target textúra

- textúrába rajzolás
- OpenGL: FBO, framebuffer object
  - ebben lesz egy vagy több render target textúra és megfelelő méretű mélységbuffer is
  - ezeket külön kell törölni
- kész textúra későbbi menetekben felhasználható

Textúra, file nélkül

```
import org.khronos.webgl.WebGLRenderingContext as GL
import org.khronos.webgl.WebGLTexture
import vision.gears.webglmath.Texture
class RenderTargetTexture(
  gl : WebGL2RenderingContext,
  width : Int = 512,
  height : Int = 512,
  internalFormat : Int = GL.RGBA,
  format : Int = internalFormat,
  type : Int = GL.UNSIGNED_BYTE
  ) : Texture {
  override val glTexture : WebGLTexture? = gl.createTexture()
  init {
    gl.bindTexture(GL.TEXTURE_2D, glTexture)
    gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_WRAP_S, GL.CLAMP_TO_EDGE)
    gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_WRAP_T, GL.CLAMP_TO_EDGE)
    gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER, GL.NEAREST)
    gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER, GL.NEAREST)
    gl.texImage2D(GL.TEXTURE_2D, 0, internalFormat, width, height, 0, format, type, null)
  }
}
```

Framebuffer (depth-stencil & render targets)

```
import org.khronos.webgl.WebGLRenderingContext as GL
import org.khronos.webgl.WebGLTexture

class Framebuffer(
  gl : WebGL2RenderingContext,
  targetCount : Int = 1,
  val width : Int = 512,
  val height : Int = 512,
  internalFormat : Int = GL.RGBA,
  format : Int = internalFormat,
  type : Int = GL.UNSIGNED_BYTE
  ) {

  val glFramebuffer = gl.createFramebuffer()
  val targets = Array<RenderTargetTexture>(targetCount) {
    RenderTargetTexture(gl, width, height, internalFormat, format, type)
  }
  val depthBuffer = gl.createRenderbuffer()
```

Framebuffer (depth-stencil & render targets)

```
  init {
    gl.bindFramebuffer(GL.FRAMEBUFFER, glFramebuffer)
    gl.drawBuffers(IntArray(targetCount){GL.COLOR_ATTACHMENT0 + it})
    targets.forEachIndexed { i, target ->
      gl.framebufferTexture2D(GL.FRAMEBUFFER, GL.COLOR_ATTACHMENT0 + i,
       GL.TEXTURE_2D, target.glTexture, 0)
    }
    gl.bindRenderbuffer(GL.RENDERBUFFER, depthBuffer)
    gl.renderbufferStorage(GL.RENDERBUFFER, GL.DEPTH_COMPONENT16, width, height)
    gl.framebufferRenderbuffer(GL.FRAMEBUFFER,
        GL.DEPTH_ATTACHMENT, GL.RENDERBUFFER, depthBuffer)
    gl.bindFramebuffer(GL.FRAMEBUFFER, null)
  }
  fun bind(gl : WebGL2RenderingContext){
    gl.bindFramebuffer(GL.FRAMEBUFFER, glFramebuffer)
    gl.viewport(0, 0, width, height)
  }
}
```

ezzel lehet beállítani

Visszaállításhoz

```
import org.khronos.webgl.WebGLRenderingContext as GL

class DefaultFramebuffer(
  val width : Int,
  val height : Int
  ){
  fun bind(gl : WebGL2RenderingContext){
    gl.bindFramebuffer(GL.FRAMEBUFFER, null)
    gl.viewport(0, 0, width, height)
  }
}
```

Sceneben, létrehozni a **resize**-ban tudjuk

```
lateinit var defaultFramebuffer : DefaultFramebuffer
lateinit var postProcFramebuffer : Framebuffer

fun resize(gl : WebGL2RenderingContext, canvas : HTMLCanvasElement) {
  gl.viewport(0, 0, canvas.width, canvas.height)
  camera.setAspectRatio(canvas.width.toFloat() / canvas.height.toFloat())

  defaultFramebuffer = DefaultFramebuffer(
                                canvas.width, canvas.height)
  postProcFramebuffer = Framebuffer(gl, 1,
                                canvas.width, canvas.height)
  postProcMaterial["rawTexture"]?.set(
     postProcFramebuffer.targets[0] )
```

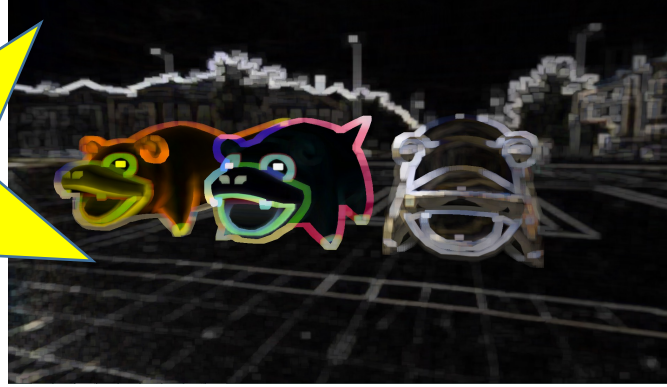
render target textúra bekötése bemenetre

Feladat: post-processing

- színtér rajzolása textúrába
- teljes képernyős téglalap rajzolása
  - program: `vsQuad` + `fsPostProc`
    - 2D textúrából olvas torzított textúrakoordinátákkal
    - eltolás mínusz fél-féllel, középen legyen a nulla, a végén visszatolás
    - origótól való távolságtól függő mértékű forgatás, fél fölött már legyen nulla
  - material: olvasson a render targetből
  - mesh: `FlipQuadGeometry` (u.a.m. a `TexturedQuadGeometry` csak a v koordináta fordítva van)
  - game object: az nem kell, a mesh rajzolása közvetlenül mehet

Éldetektálás (glowing edges)

extra



- szomszédos képpontok feldolgozása
- eredmény a csatornánkénti maximum és minimum különbsége