A grafikus hardware programozása WebGL-lel

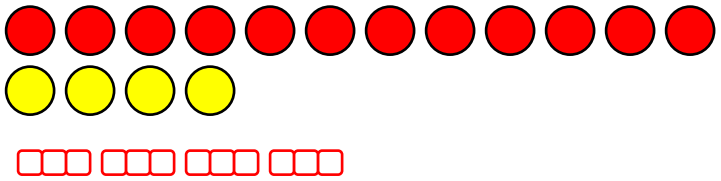
Szécsi László

3D Grafikus Rendszerek

3. előadás

GPU pipeline input

- vertex bufferek
- index buffer
- rajzolási állapot
  - egyes pipeline-elemek működési beállításai
  - programozható elemek shader programjai
- erőforrások – globális memóriában adatok
  - globális (uniform) változók
  - textúrák
  - adatbufferek

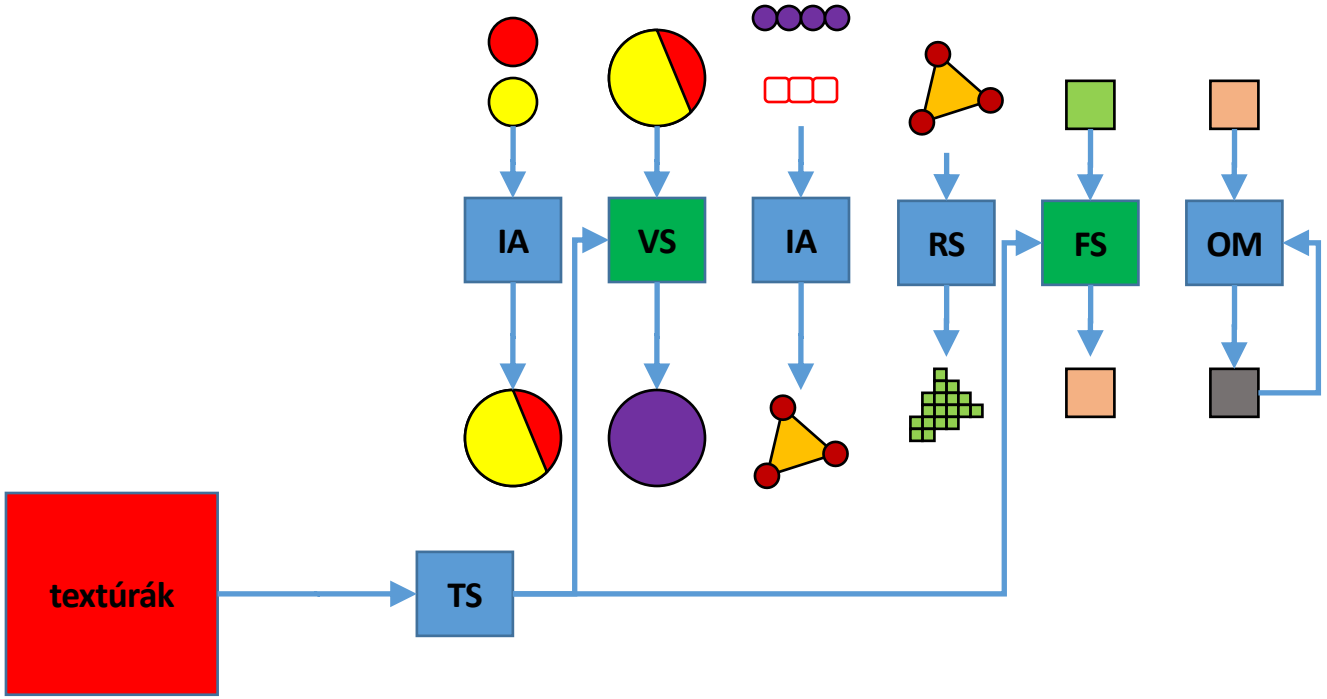


csak olvasható

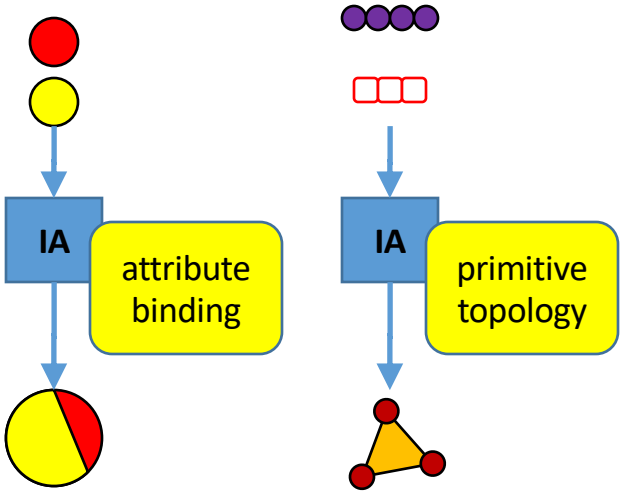
GPU pipeline output

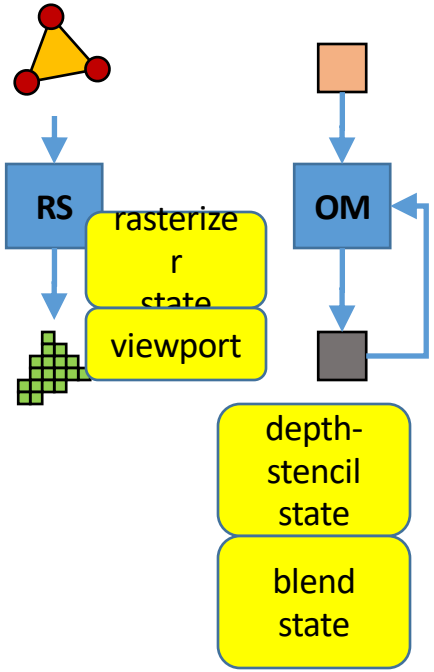
- kép
  - render target
    - frame buffer
    - textúra
  - mélységbuffer (+stencil)

GPU pipeline tesszellátor nélkül



Rajzolási állapot





Lépések I

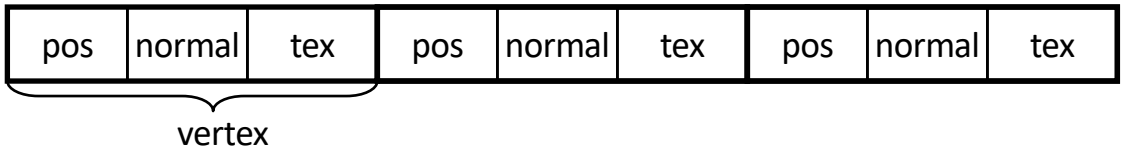
- vertexek összeállítása
- Vertex Shader
  - model trafó
  - árnyalás
  - view és proj trafó
- primitívek összeállítása
  - vertexek összeválogatása
    - vertex bufferbeli sorrend alapján
    - index bufferbeli indexek alapján

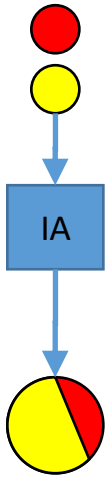
Lépések II

- raszterizálás
  - hátsólap-eldobás
  - vágás
  - homogén osztás
  - viewport trafó
- Fragment shader
  - pixelszín meghatározása
- z-teszt
- blending

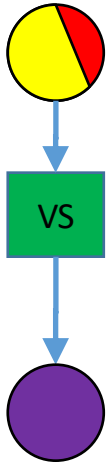
Vertexek összeállítása

- Vertex buffer
  - rekordok tömbje





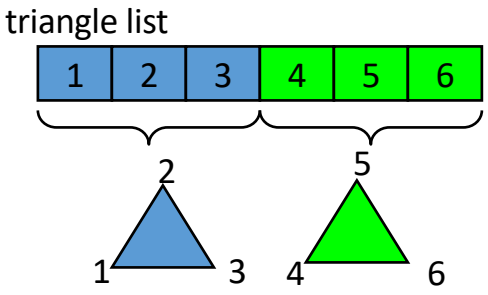
Vertex shader



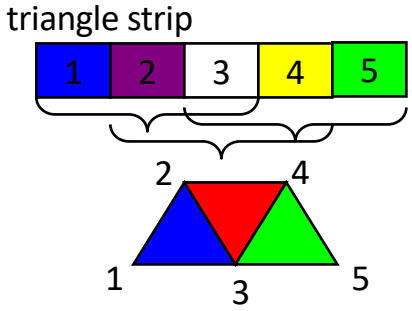
- bejövő adat
  - uniform: model, view, proj mátrixok, fények, ...
  - minden vertexre más: vertex és instance elemek [pozíció, normál, szín, texcoord]
- kimenő adat
  - pozíció homogén normalizált képernyő koordinátákban
  - árnyalás eredménye [szín]
  - bármi más [normál, texcoord]

Primitívek összeállítása

- non-indexed
  - vertex bufferben egymás után következők



`WebGL.RenderingContext.`*`TRIANGLES`*



`WebGL.RenderingContext.`*`TRIANGLE_STRIP`*

Raszterizáló egység: lapeldobás

- körüljárási irány (képernyőn) alapján eldobhatunk lapokat
- ha a modellünkben (index bufferben) konzisztens a háromszögek körüljárási iránya, ezzel eldobhatjuk a test hátsó lapjait
  - a belsejét úgysem látjuk, ha poliéder

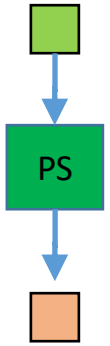
Raszterizáló egység: vágás

- ami kilógna a képernyőről, vágjuk le
- háromszögvágás szakaszvágásra visszavezethető
  - szakasz vágás = pont vágás + metszéspont számítás
- normalizált képernyőkoordinátákban
  - [-1, -1, 0] [1, 1, 1] téglatestre vágunk [Descartes]

Raszterizáció

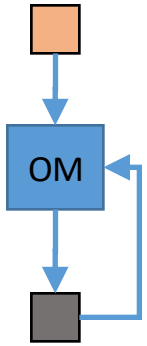
- homogén osztás
- viewport trafó: pixel koordináták
- lineáris interpoláció
  - vertex output adatok (kivéve pozíció) interpolálása minden kitöltendő pixelhez
  - pixel shader indítása minden pixelszín meghatározásához

Fragment shader



- bejövő adat
  - vertex shader kimenő adatai lineárisan interpolálva
  - pixel koordináták
- kimenő adat
  - pixel szín [RGBA]
  - mélység, ha felül akarjuk írni a háromszög csúcsainak z-jéből interpolált eredetit
    - ha nem, akkor a z-teszt előrehozható a pixel shader elé

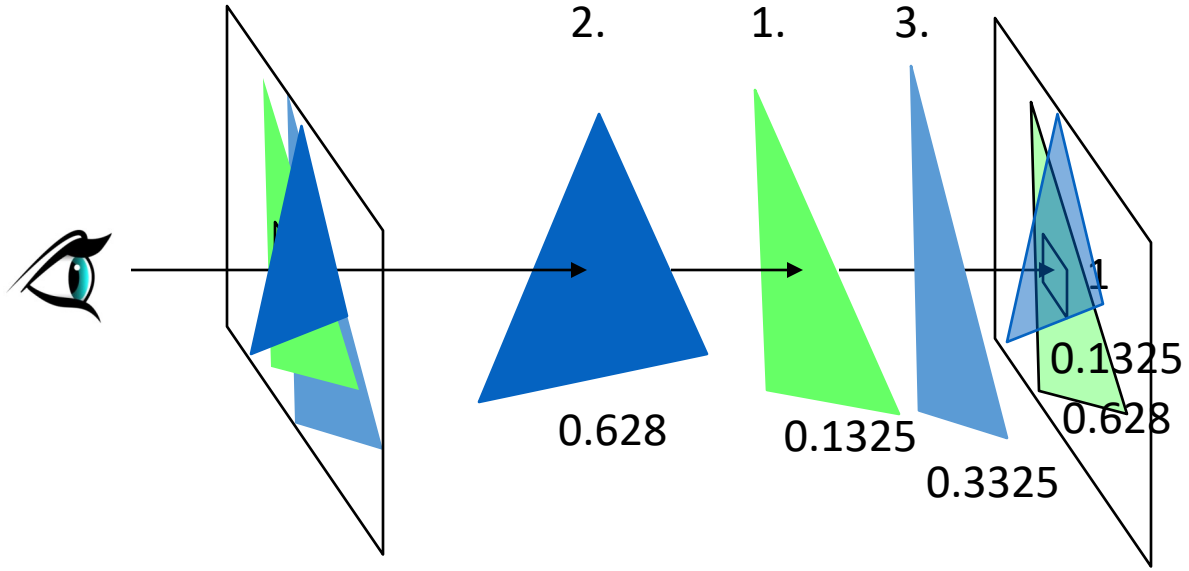
Kimeneti műveletek



- bufferek
  - célterület: frame buffer vagy textúra
  - mélység-stencil buffer
- műveletek
  - mélység teszt
  - stencil teszt
  - keverés [alfa blending]

Z-teszt

- a 3D takarási probléma megoldására



Keverés [alpha blending]

- A célterületen már meglevő érték [dest] és az újonnan számított szín [src] kombinálása
- mindkettőhöz megadható egy súly (0, 1, srcalpha, dstalpha, 1-alpha ...)
- megadható a függvény (add, subtract, min, max ...)
  - átlátszóság: hátulról előre rajzolás + blending
  - src * srcalpha + dst * (1- srcalpha)

WebGL2

- Vertex shader
- Fragment shader
- Vertex buffer
- Texture
- Framebuffer
- Render state
- **3D texture**
- **Vertex Array Object**
- int math
- **dinamikus ciklusok**
- Floating-point texture
- Compressed texture
- FS depth write
- Multiple render target

- nincs
  - Geometry shader
  - Tessellation shader

WebGL2

- hasonló, mint a desktop OpenGL:

```
// ...
gl.bindBuffer(/* ... */)
gl.vertexAttribPointer(/* ... */)
gl.useProgram(/* ... */)
gl.drawArrays(/* ... */)
```

Scene – viewport és törlés

```
class Scene (val gl : WebGL2RenderingContext){

  fun resize(gl : WebGL2RenderingContext,
             canvas : HTMLCanvasElement) {
    gl.viewport(0, 0, canvas.width, canvas.height)
  }

  fun update(gl : WebGL2RenderingContext,
             keysPressed : Set<String>) {
    gl.clearColor(0.3f, 0.0f, 0.3f, 1.0f)
    gl.clearDepth(1.0f)
    gl.clear(GL.COLOR_BUFFER_BIT or GL.DEPTH_BUFFER_BIT)
  }
}
```

Rajzolás

- kell geometria
  - vertex bufferek, index buffer
  - attribútum kötések
- kell GPU program
  - vertex shader
  - pixel shader

Scene – geometria és GPU program

```
val vsIdle = Shader(gl, GL.VERTEX_SHADER,
                        "shaders/idle-vs.glsl")
val fsSolid = Shader(gl, GL.FRAGMENT_SHADER,
                        "shaders/solid-fs.glsl")
val solidProgram = Program(gl, vsIdle, fsSolid)
val triangleGeometry = TriangleGeometry(gl)

fun update(...) {
  gl.useProgram(solidProgram.glProgram)
  triangleGeometry.draw()
}
```

Shader források külön fileban

- XMLHttpRequest kell a betöltéséhez

Vertex shader: content/shaders/idle-vs.glsl

```
#version 300 es

in vec4 vertexPosition;

void main(void) {
  gl_Position = vertexPosition;
}
```

Fragment shader: content/shaders/solid-fs.glsl

```
#version 300 es

precision highp float;

out vec4 fragmentColor;

void main(void) {
  fragmentColor = vec4(1, 1, 0.0, 1);
}
```

Shader objektum

```
class Shader (
  val gl : WebGL2RenderingContext,
  val shaderType : Int,
  val sourceUrl : String) {

  val glShader = gl.createShader(shaderType)

  init {
    gl.shaderSource(glShader, source)
    gl.compileShader(glShader)
  }
}
```

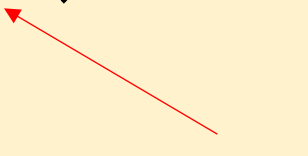
ebbe még be kell húzni a file tartalmát

kellenek még a hibaüzenetek

File betöltés

```
val request = XMLHttpRequest()
request.overrideMimeType("text/plain")
request.open("GET", sourceUrl, false)

request.onloadend = {
  val source = request.responseText
  /* fordítás, stb. */
}
request.send()
```



Program

```
companion object{
  var all = emptyArray<Program>()
  val PC =  arrayOf("vertexPosition",
                    "vertexColor")
  val PNT = arrayOf("vertexPosition",
                    "vertexNormal",
                    "vertexTexCoord")
}
```

```
class Program(  val gl : WebGL2Rend
                val vertexShader :
                val fragmentShader : Shader,
                attributeBindings : Array<String> = Program.PNT ) {
  val glProgram = gl.createProgram()

  init {
    gl.attachShader(glProgram, vertexShader.glShader)
    gl.attachShader(glProgram, fragmentShader.glShader)

    var attributeIndex = 0
    attributeBindings.forEach{
      gl.bindAttribLocation(glProgram, attributeIndex++, it)
    }

    gl.linkProgram(glProgram)
  }
}
```

TexturedQuadGeometry – vertex buffer

```
class TexturedQuadGeometry(val gl : WebGL2RenderingContext) :
Geometry() {

  val vertexBuffer = gl.createBuffer()
  init{
    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)
  }
```

TexturedQuadGeometry – normal buffer

```
val vertexNormalBuffer = gl.createBuffer()
init{
  gl.bindBuffer(GL.ARRAY_BUFFER,
                          vertexNormalBuffer)
  gl.bufferData(GL.ARRAY_BUFFER,
    Float32Array( arrayOf<Float>(
       0.0f,  0.0f, 1.0f,
       0.0f,  0.0f, 1.0f,
       0.0f,  0.0f, 1.0f,
       0.0f,  0.0f, 1.0f
    )),
    GL.STATIC_DRAW)
}
```

TexturedQuadGeometry – texcoord buffer

```
val vertexTexCoordBuffer = gl.createBuffer()
init{
  gl.bindBuffer(GL.ARRAY_BUFFER,
                        vertexTexCoordBuffer)
  gl.bufferData(GL.ARRAY_BUFFER,
    Float32Array( arrayOf<Float>(
       0.0f,  1.0f,
       0.0f,  0.0f,
       1.0f,  1.0f,
       1.0f,  0.0f
       )),
    GL.STATIC_DRAW)
}
```

TexturedQuadGeometry – index buffer

```
val indexBuffer = gl.createBuffer()
init{
  gl.bindBuffer(GL.ELEMENT_ARRAY_BUFFER,
                                  indexBuffer)
  gl.bufferData(GL.ELEMENT_ARRAY_BUFFER,
    Uint16Array( arrayOf<Short>(
      0, 1, 2,
      1, 2, 3
    )),
    GL.STATIC_DRAW)
}
```

QuadGeometry – VAO

```
val inputLayout = gl.createVertexArray()
init{
  gl.bindVertexArray(inputLayout)

  gl.bindBuffer(GL.ARRAY_BUFFER, vertexBuffer)
  gl.enableVertexAttribArray(0)
  gl.vertexAttribPointer(0,
    3, GL.FLOAT, //< three pieces of float
    false, //< do not normalize (make unit length)
    0, //< tightly packed
    0 //< data starts at array start
  )
```

Attribútumok megadásának összefoglalója

- bemenő változó deklarációja a VS-ben

```
in vec4 vertexPosition;
```

- a Program.js-ben a #0 számhoz rendeljük

```
gl.bindAttribLocation(glProgram, 0, "vertexPosition");
```

- a TexturedQuadGeometry.js-ben
  - létrehozzuk és kitöltjük a buffert
  - leírjuk a tartalmát

```
gl.bindBuffer(gl.ARRAY_BUFFER,
this.vertexBuffer);
gl.enableVertexAttribArray(0);
gl.vertexAttribPointer(0,
    3, gl.FLOAT,
    false, 0,  0
  );
```

```
val vertexBuffer = gl.createBuffer()
  init{
    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)
  }
```

Alapértelmezett bemenetek

ha az attribútumadat kevesebb elemmel rendelkezik, mint a VS bemenet, a hiányzó értékek alapértelmezése x=0, y=0, z=0, w=1

e is declared in VS

```
in vec4 vertexPosition;
```

an attribute #0 in Program.js

```
cation(this.glProgram, 0, 'vertexPosition');
```

eometry.js

- v eate and fill buffer
- a xplain layout

```
this.vertexBuffer =
gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER,
this.vertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER,
  new Float32Array( [
    -0.5, -0.5, 0.5,
    -0.5,  0.5, 0.5,
     0.5,  0.0, 0.5, ] ),
  gl.STATIC_DRAW);
```

```
gl.bind    r(gl.ARRAY_BUFFER,
this.ve    uffer);
gl.enableVertexAttribArray(0);
gl.vertexAttrib Pointer(0,
    3, gl.FLOAT,
    false, 0, 0
  );
```

QuadGeometry – draw

```
fun draw() {
    gl.bindVertexArray(inputLayout)
    gl.bindBuffer(GL.ELEMENT_ARRAY_BUFFER,
                                    indexBuffer)

    gl.drawElements(GL.TRIANGLES, 6,
                            GL.UNSIGNED_SHORT, 0)
  }
```

WebGL-ben nem a VAO része!!!