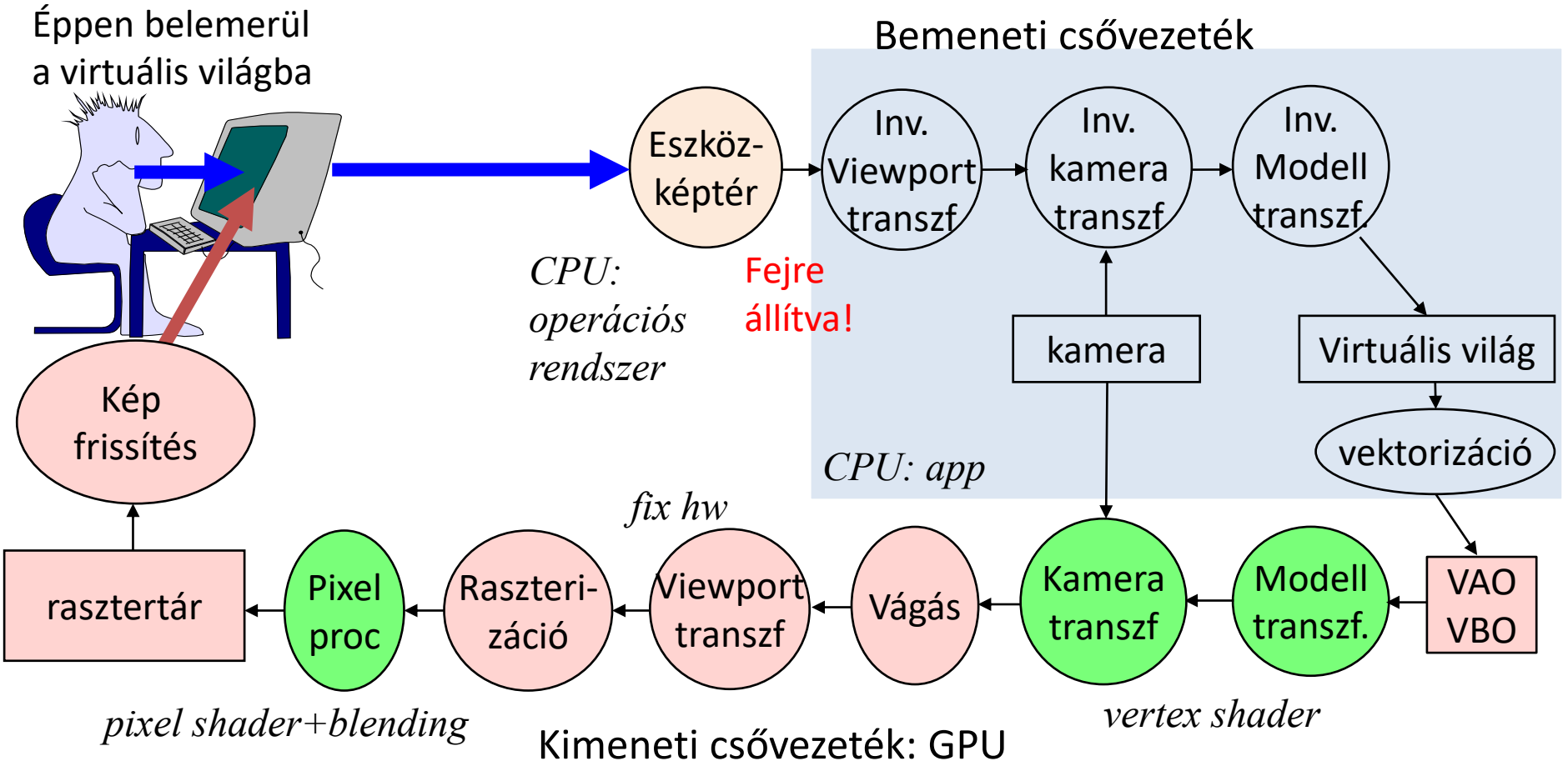
”In theory, there is no difference between theory and practice. In practice, there is.”
Benjamin Brewster

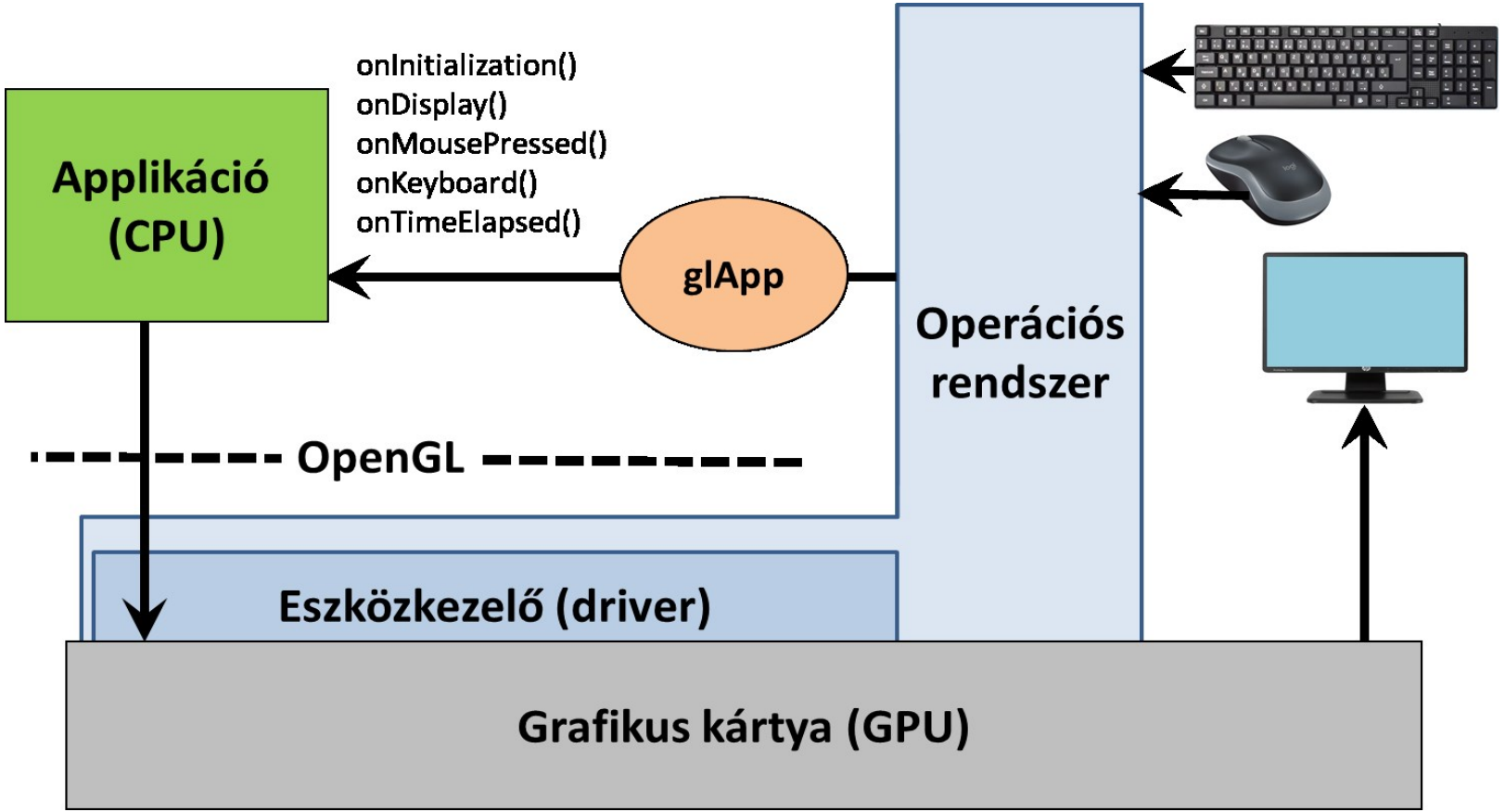
Grafikus hardver/szoftver alapok
Építőelemek

Szirmay-Kalos László

Interaktív rendszer: Funkcionális modell



Szoftver architektúra

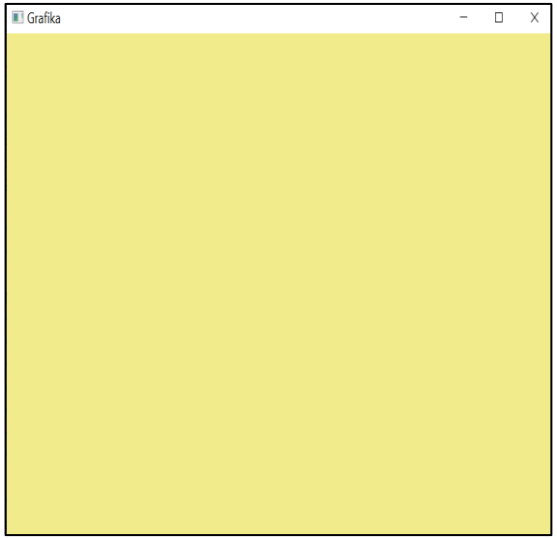


Applikáció és esemény kezelés

```
class glApp {
public:
   glApp(const char* caption);
   // glApp(int major, int minor, int winWidth, int winHeight, const char* caption);
   void refreshScreen(); // Ablak érvénytelenítése
   // Eseménykezelők
   virtual void onInitialization() {}    // Inicializáció
   virtual void onDisplay() {}           // Ablak érvénytelen
   virtual void onKeyboard(int key) {}   // Klaviatúra gomb lenyomás
   virtual void onKeyboardUp(int key) {} // Klaviatúra gomb elenged
   // Egér gomb lenyomás/elengedés
   virtual void onMousePressed(MouseButton but, int pX, int pY) {}
   virtual void onMouseReleased(MouseButton but, int pX, int pY) {}
   // Egér mozgatás lenyomott gombbal
   virtual void onMouseMotion(int pX, int pY) {}
   virtual void onTimeElapsed(float startTime, float endTime) {} // Telik az idő
};
```

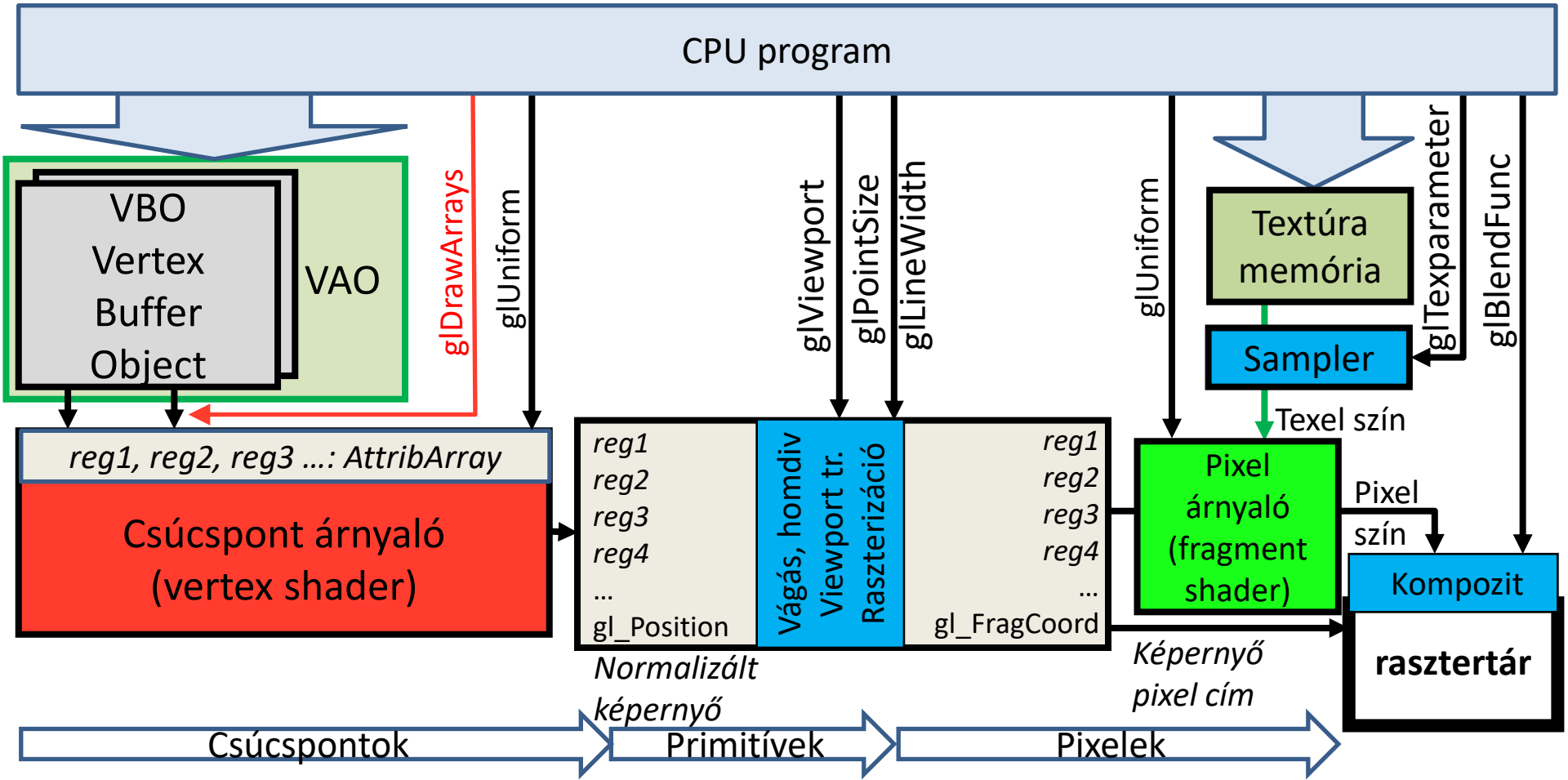
Színes háttér

```
#include "framework.h"
class MyApp : public glApp {
   float red, green, blue, alpha; // színcsatornák
   void setRandomColor() {        // véletlen szín
      red   = (float)rand() / RAND_MAX;
      green = (float)rand() / RAND_MAX;
      blue = (float)rand() / RAND_MAX; alpha = 1.0f;
   }
public:
   MyApp() : glApp("Grafika") { } // Megfogócsík szöveg
   void onInitialization() { setRandomColor(); } // véletlen szín
   void onDisplay() {
      glClearColor(red, green, blue, alpha);     // háttér szín
      glClear(GL_COLOR_BUFFER_BIT);              // rasztertár törlés
   }
   void onKeyboard(int key) {
      setRandomColor();      // véletlen szín
      refreshScreen();       // onDisplay esemény
   }
} app;
```

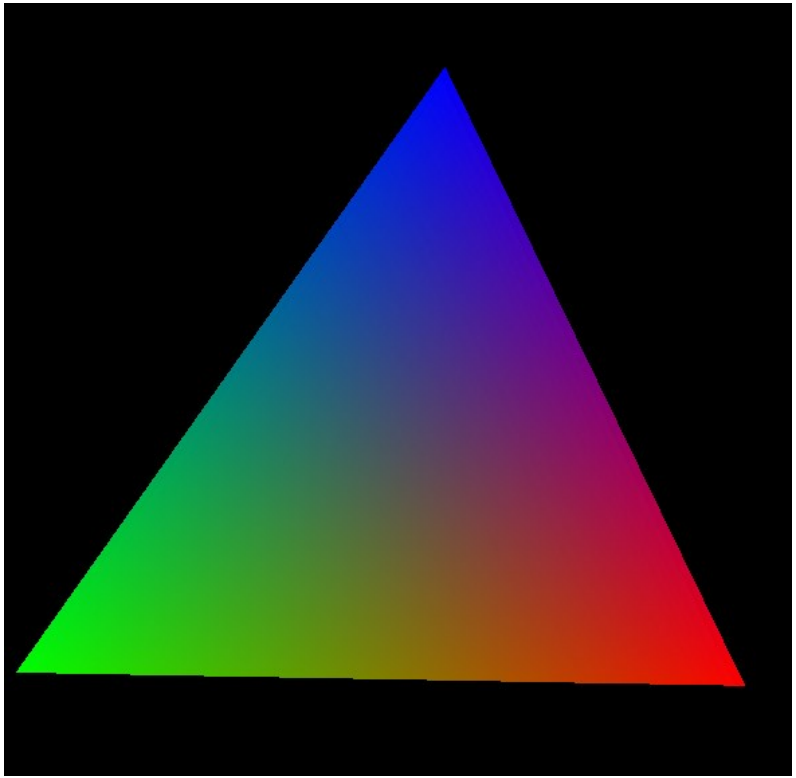


framework.h
framework.cpp
glad.c
lodepng.cpp
Lib: GLFW, GLM

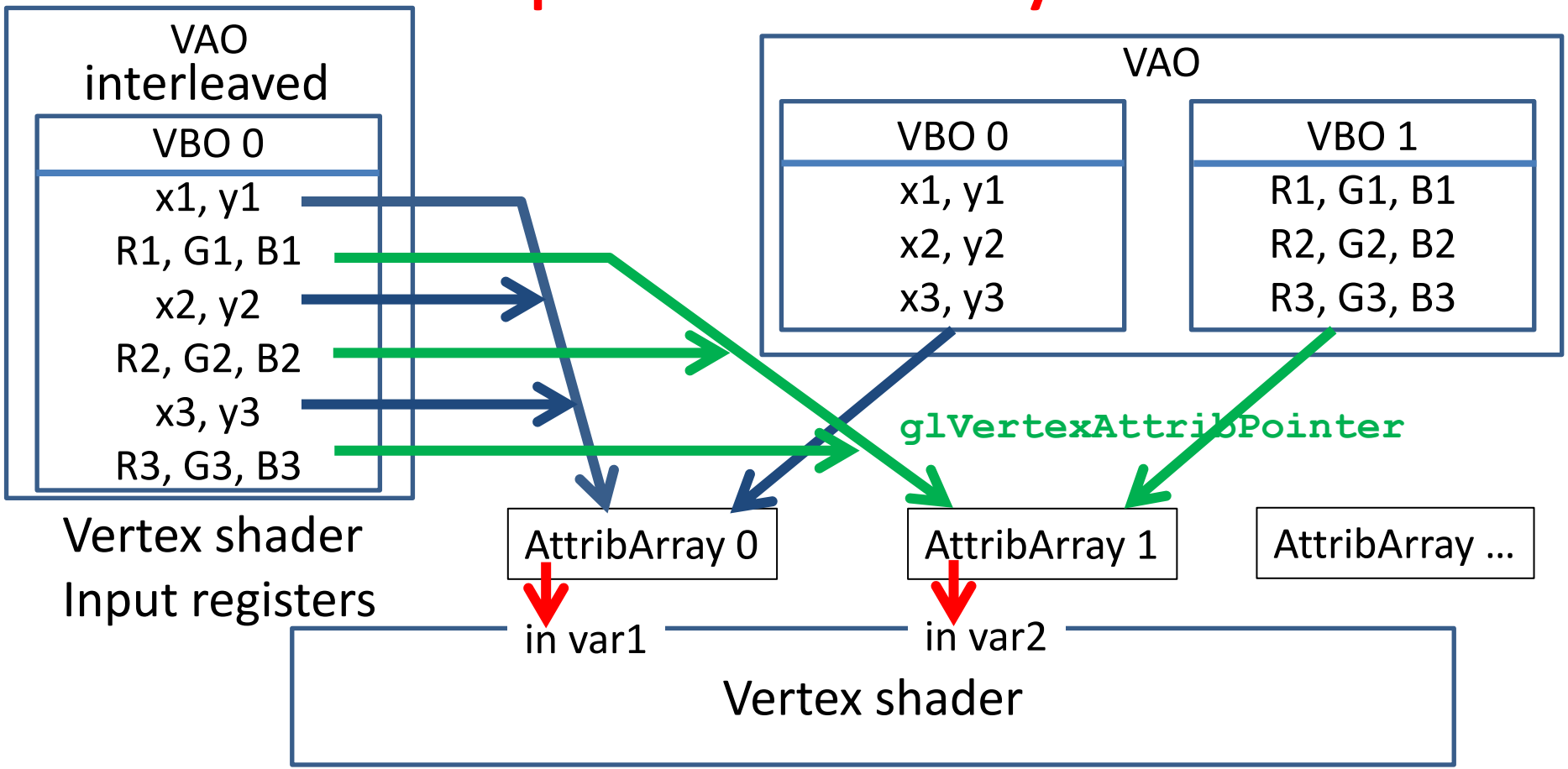
OpenGL 3.3 … 4.6 (Modern OpenGL)



Csúcspont tulajdonságok interpolációja

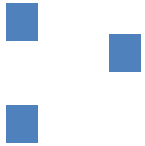


Csúcspont adatfolyamok



Rajzolás: glDrawArrays, OpenGL primitívek

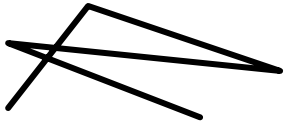
```
glBindVertexArray(vao);
glDrawArrays(primitiveType, startIdx, numOfElements);
```



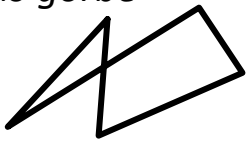
GL_POINTS

GL_LINES

Vektorizált parametrikus görbe

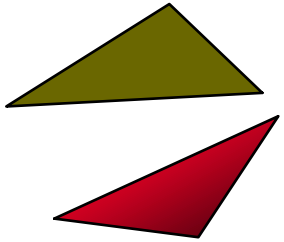


GL_LINE_STRIP



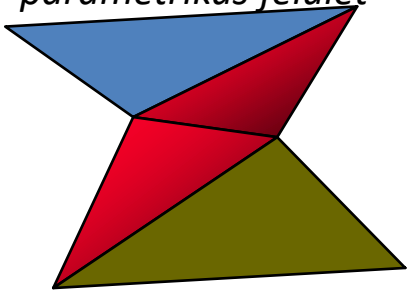
GL_LINE_LOOP

Fülvágó kimenete



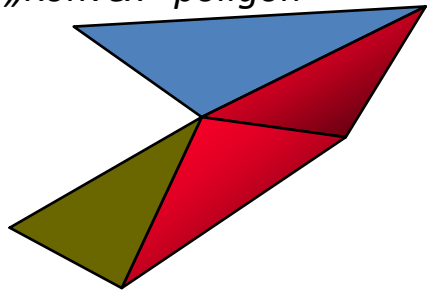
GL_TRIANGLES

Tesszellált 3D parametrikus felület



GL_TRIANGLE_STRIP

„Konvex” poligon



GL_TRIANGLE_FAN

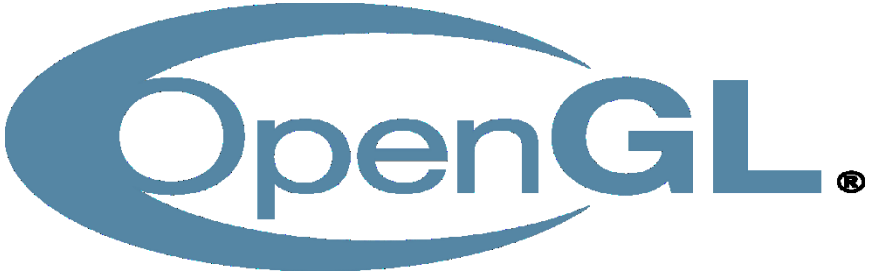
OpenGL állapotgép + erőforrások

Gagyi grafikus könyvtár

```
fillOval(x1,y1,x2,y2, texture, color, width,…);
```

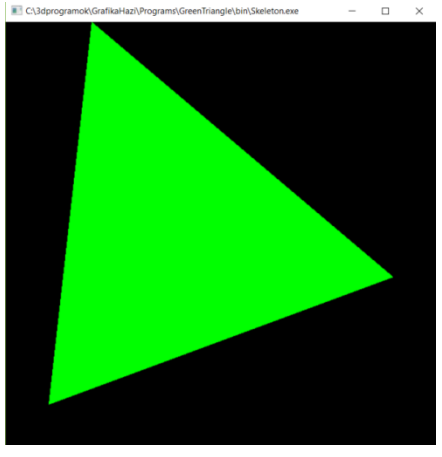
OpenGL

```
glPointSize(3);
glLineWidth(5);
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBindTexture(GL_TEXTURE_2D, textureId);

glBufferData(GL_ARRAY_BUFFER, 10, v, GL_STATIC_DRAW);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, …);

glDrawArrays(GL_TRIANGLES, 0, 3); // Mind!!!
```

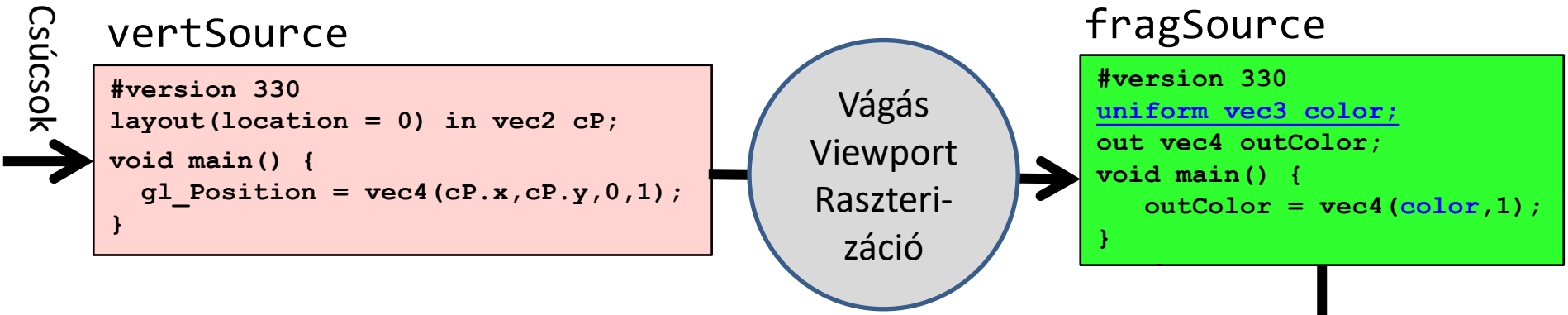


Grafikus hardver/szoftver alapok
2. Helló OpenGL/GLSL

Szirmay-Kalos László



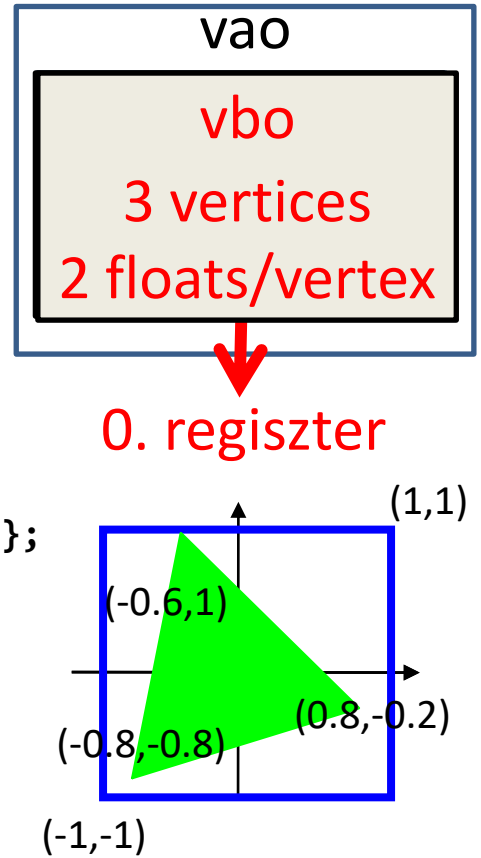
Az első OpenGL programom: Zöld háromszög




```
class GreenTriangApp : public glApp {
   unsigned int vao;  // geometria
   unsigned int prog; // árnyalóprogramok
public:
   MyApp() : glApp("Green triangle") { }
   void onInitialization();
   void onDisplay();    // Ablak újrarajzolás
} app;
```

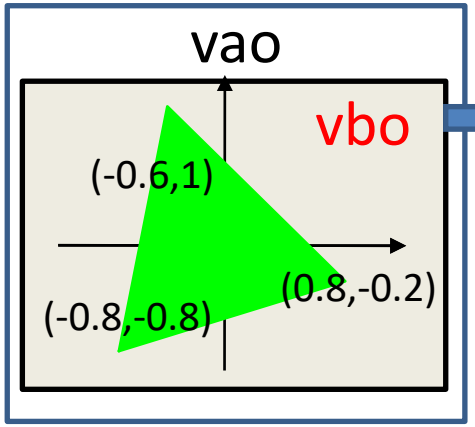
onInitialization()

```
void GreenTriangApp::onInitialization() {
    glGenVertexArrays(1, &vao); // 1 vao id-t kérek
    glBindVertexArray(vao);     // aktiválom
    unsigned int vbo;           // Vertex Buffer Object
    glGenBuffers(1, &vbo);      // 1 vbo azonosítót kérek
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    // Geometria 24 bájtban (6 float vagy 3 x 2 koordináta)
    float vertices[] = { -0.8f, -0.8f, -0.6f, 1.0f, 0.8f, -0.2f };
    glBufferData(GL_ARRAY_BUFFER, // GPU-n a másolás célja
                 sizeof(vertices), // bájtok száma
                 vertices, // cím
                 GL_STATIC_DRAW); // nem változtatjuk
    glEnableVertexAttribArray(0); // 0. bemeneti regiszter
    glVertexAttribPointer(0, // vbo -> 0. bemeneti regiszter
                          2, GL_FLOAT, GL_FALSE, // két float/attrib
                          0, NULL); // szorosan pakolt

    …
```



onInitialization() folytatás

```
    …
    // Csúcspont árnyaló: azonosító generálás, forrás feltöltés, fordítás
    unsigned int vertShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertShader, 1, &vertSource, NULL);
    glCompileShader(vertShader);

    // Pixel árnyaló: azonosító generálás, forrás feltöltés, fordítás
    unsigned int fragShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragShader, 1, &fragSource, NULL);
    glCompileShader(fragShader);

    // Program = csúcspont-pixelárnyaló pár létrehozása
    prog = glCreateProgram();
    glAttachShader(prog, vertShader);
    glAttachShader(prog, fragShader);
    glLinkProgram(prog); // a csúcspont-pixelárnyaló pár szerkesztése
    glUseProgram(prog);  // ez fusson
}
```



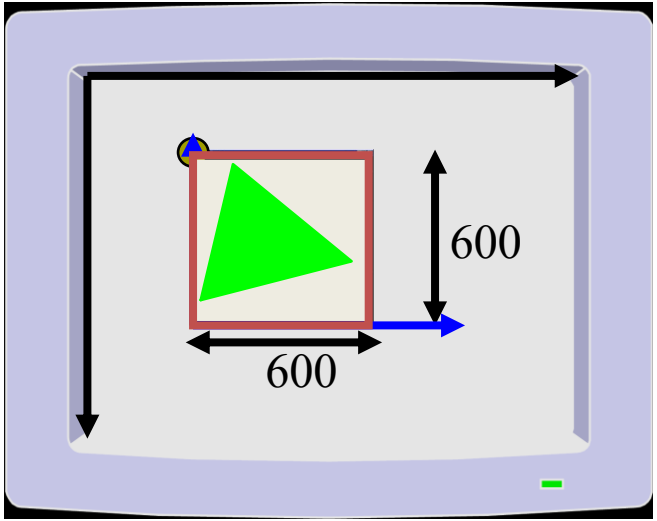
0. regiszter

onDisplay()

```
#version 330
layout(location = 0) in vec2 cP;
void main() {
  gl_Position = vec4(cP.x,cP.y,0,1);
}
```

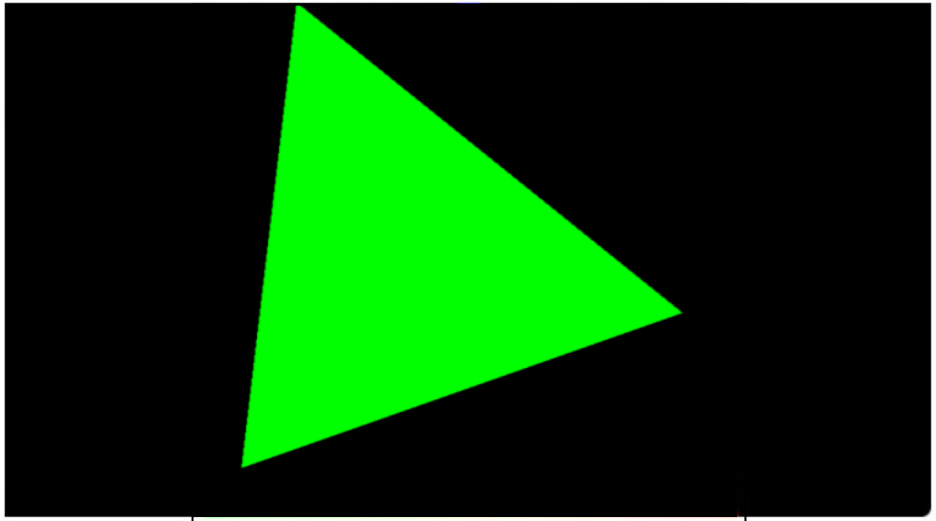
```
#version 330
uniform vec3 color;
out vec4 outColor;
void main() {
   outColor = vec4(color,1);
}
```

```
void GreenTriangApp::onDisplay() {   // Ablak újrarajzolás
   glClearColor(0, 0, 0, 0); // háttér szín
   glClear(GL_COLOR_BUFFER_BIT); // törlés
   glViewport(0, 0, windowWidth, windowHeight);
   int location = glGetUniformLocation(prog, "color");
   glUniform3f(location, 0.0f, 1.0f, 0.0f); // 3 float, zöld
   glBindVertexArray(vao); // Rajzolás hívás
   glDrawArrays(GL_TRIANGLES, 0 /*start*/, 3 /*Elemeszám*/);
}
```




I KNOW
OPENGL


SHOW ME



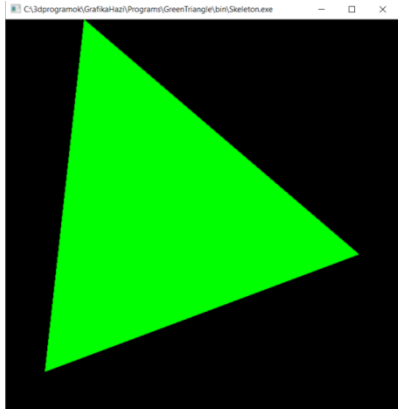
”The GL in OpenGL stands for Good Luck because you are going to need it.”

Anonymous

Grafikus hardver/szoftver alapok

Program: Keret és zöld háromszög

Szirmay-Kalos László



framework.h: GPUProgram

```
class GPUProgram {
    unsigned int prog;
public:
    GPUProgram();
    GPUProgram(char * vertSrc, char * fragSrc);
    void Use(); // Ez fusson
    void setUniform(int i, const string& name);
    void setUniform(float f, const string& name);
    void setUniform(const vec2& v, const string& name);
    void setUniform(const vec3& v, const string& name);
    void setUniform(const vec4& v, const string& name);
    void setUniform(const mat4& mat, const string& name);
};
```

framework.h: Geometry

```
template<class T> class Geometry {
protected:
   unsigned int vao, vbo; // GPU
   vector<T> vtx;         // CPU
public:
   Geometry() {
      glGenVertexArrays(1, &vao); glBindVertexArray(vao);
      glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo);
   }
   void updateGPU();
   void Bind() { glBindVertexArray(vao); }
   vector<T>& Vtx() { return vtx; }
   void Draw(GPUProgram* gpuProgram, int type, vec3 color);
   virtual ~Geometry() {
      glDeleteBuffers(1, &vbo); glDeleteVertexArrays(1, &vao);
   }
};
```

Geometry::updateGPU

```
void Geometry::updateGPU() { // CPU -> GPU
   glBindVertexArray(vao);
   glBindBuffer(GL_ARRAY_BUFFER, vbo);
   glBufferData(GL_ARRAY_BUFFER, vtx.size()*sizeof(T), &vtx[0], GL_DYNAMIC_DRAW);
   glEnableVertexAttribArray(0);
   glVertexAttribPointer(0, sizeof(T)/sizeof(float), GL_FLOAT, GL_FALSE, 0, NULL);
}
```

Geometry::Draw

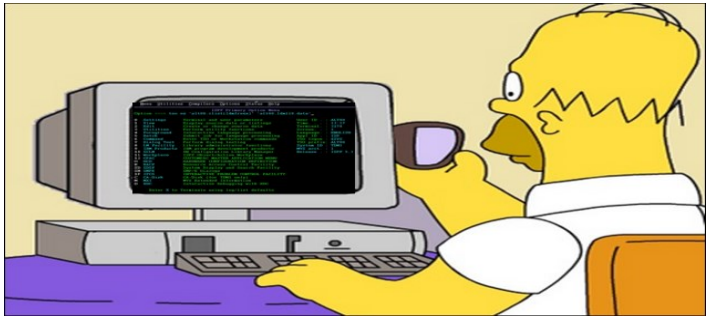
```
#version 330
layout(location = 0) in vec2 cP;
void main() {
  gl_Position = vec4(cP.x,cP.y,0,1);
}
```

```
#version 330
uniform vec3 color;
out vec4 outColor;
void main() {
   outColor = vec4(color,1);
}
```

```
void Geometry::Draw(GPUProgram* gpuProgram, int type, vec3 color) {
   if (vtx.size() > 0) {
      gpuProgram->setUniform(color, "color");
      Bind();
      glDrawArrays(type, 0, (int)vtx.size());
   }
}
```

GreenTriangApp

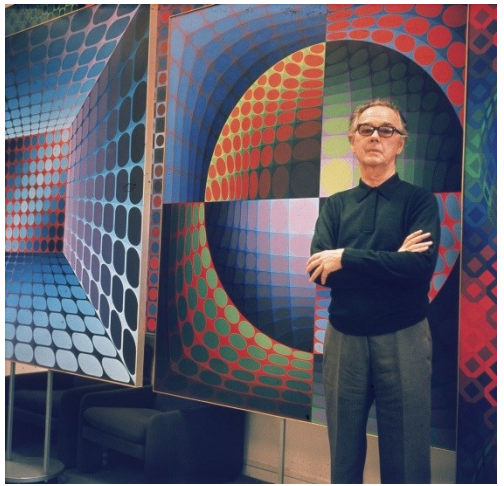
```
class GreenTriangleApp : public glApp {
   Geometry<vec2>* triangle;  // geometria
   GPUProgram* gpuProgram;   // csúcspont és pixel árnyalók
public:
   GreenTriangleApp() : glApp("Green triangle") { }
   void onInitialization() {  // Inicializáció,
      triangle = new Geometry<vec2>;
      triangle->Vtx() = { vec2(-0.8f, -0.8f), vec2(-0.6f, 1.0f), vec2(0.8f, -0.2f) };
      triangle->updateGPU();
      gpuProgram = new GPUProgram(vertSource, fragSource);
   }
   void onDisplay() {  // Ablak újrarajzolás
      glClearColor(0, 0, 0, 0);     // háttér szín
      glClear(GL_COLOR_BUFFER_BIT); // rasztertár törlés
      glViewport(0, 0, winWidth, winHeight);
      triangle->Draw(gpuProgram, GL_TRIANGLES, vec3(0.0f, 1.0f, 0.0f));
   }
} app;
```



Grafikus hardver/szoftver alapok

Program: Vasarely festmény

Szirmay-Kalos László



Mi ez?

- Körök, kicsik fedik a nagyokat
- 20 kör vagy 1 kör transzformálva?
- Középpont elindul felfelé, majd visszatér

Applikáció

```
class VasarelyApp : public glApp {
   GPUProgram* gpuProgram;  // árnyaló programok
   Circle* circle;          // egység sugarú kör
   float amplitude = 0.8f;  // betüremkedés illuzió nagysága
   bool animate = false;    // mozgassuk?
public:
   VasarelyApp() : glApp("Vasarely") { }
   void onInitialization() {
      circle = new Circle();
      gpuProgram = new GPUProgram(vertSource, fragSource);
   }
   void onDisplay();
   void onKeyboard(int key) { if (key == 'a') animate = !animate; }
   void onTimeElapsed(float startTime, float endTime) {
      if (animate) {
         amplitude = sinf(3 * endTime); // hullámzás
         refreshScreen(); // újrarajzolás
      }
   }
} app;
```

Csúcspont és pixel árnyalók

Vertex shader:

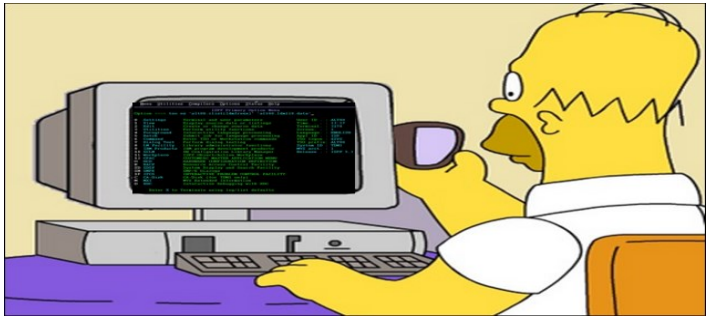
```
uniform float radius, up;
layout(location = 0) in vec2 vp;

void main() {
    gl_Position = vec4(vp.x * radius, vp.y * radius + up, 0, 1);
}
```

Fragment shader:

```
uniform vec3 color;
out vec4 outColor;

void main() {
    outColor = vec4(color, 1);
}
```

onDisplay

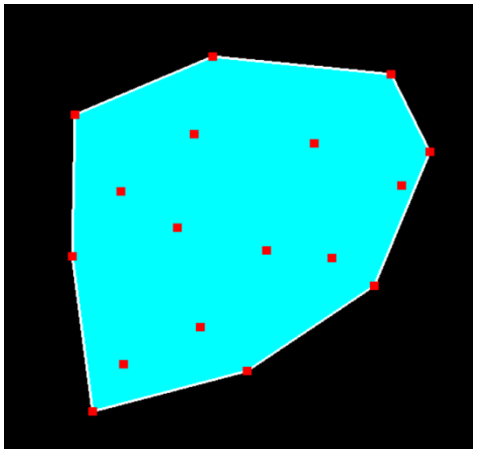
```
void VasarelyApp::onDisplay() {
   glClearColor(0, 0, 0, 0);     // háttér szín
   glClear(GL_COLOR_BUFFER_BIT); // alkalmazói ablak törlése
   glViewport(0, 0, winWidth, winHeight); // keletkező fénykép mérete

   const int nCircles = 20;
   const float r0 = 1.0f, r1 = 1.0f / nCircles; // sugár tartománya
   vec3 ce0(1, 0, 0), ce1(0, 0, 0), co0(0, 0, 0.5f), co1(0, 1, 1); // kezdő-vég szín

   for (int i = 0; i < nCircles; i++) {       // körök egyenként
      float t = (float)i / (nCircles - 1.0f); // interpolációs változó
      gpuProgram->setUniform(r0 * (1 - t) + r1 * t, "scaling"); // transzformáció
      gpuProgram->setUniform(t * (1 - t) * amplitude, "up");
      vec3 color = (i % 2 == 0) ? ce0 * (1 - t) + ce1 * t : co0 * (1 - t) + co1 * t;
      circle->Draw(gpuProgram, GL_TRIANGLE_FAN, color);
   }
}
```



Grafikus hardver/szoftver alapok

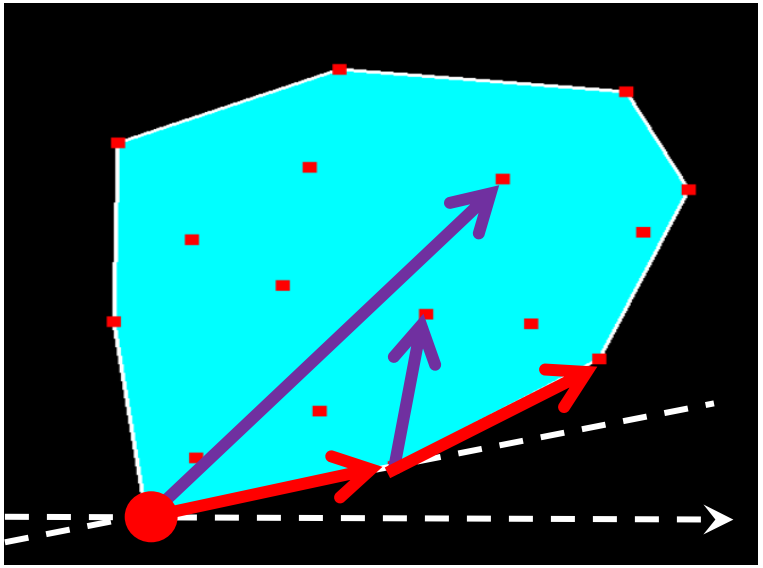
Program: Konvex burok, interakció

Szirmay-Kalos László



Konvex burok

Minimális, az adott pontokat tartalmazó konvex halmaz



Legalsó pontból indulunk,
a kezdő irány balról jobbra.

While (vissza nem érünk) {
    Következő pont, amelyhez
    minimálisat kell fordulni.
}

Csúcspont és pixel árnyalók

Vertex shader:

```
layout(location = 0) in vec2 cP;

void main() {
    gl_Position = vec4(cP, 0, 1);
}
```

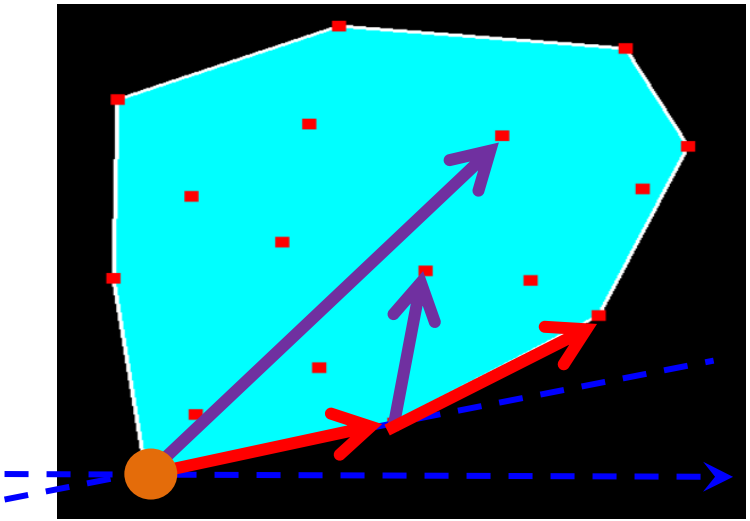
Fragment shader:

```
uniform vec3 color;
out vec4 outColor;

void main() {
    outColor = vec4(color, 1); // RGB -> RGBA
}
```

Convex hull

```
struct ConvexHull {
   Geometry<vec2> setPoints, hullPoints;
   void addPoint(vec2 p) { setPoints.Vtx().push_back(p); }
   void update() {
      if (setPoints.Vtx().size() >= 3) {
         findHull();
         hullPoints.updateGPU();
      }
      setPoints.updateGPU();
   }
   vec2 * pickPoint(vec2 p) {
      for (auto& v : setPoints.Vtx()) if (length(p - v) < 0.05f) return &v;
      return nullptr;
   }
   void findHull();
   void Draw(GPUProgram * gpuProgram) {
      if (hullPoints.Vtx().size() >= 3) {
         hullPoints.Draw(gpuProgram, GL_TRIANGLE_FAN, vec3(0, 1, 1));
         hullPoints.Draw(gpuProgram, GL_LINE_LOOP, vec3(1, 1, 1));
      }
      setPoints.Draw(gpuProgram, GL_POINTS, vec3(1, 0, 0));
   }
};
```

Konvex burok előállítás



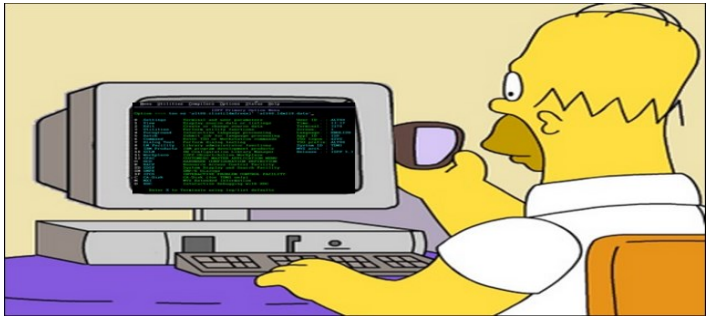
```
void ConvexHull::findHull()
{
   vec2 *pLow = &setPoints.Vtx()[0]; // Find lowest point
   for(auto& p : setPoints.Vtx())
      if (p.y < pLow->y) pLow = &p;

   hullPoints.Vtx().clear();
   vec2 pCur = *pLow, *pNext, dir(1, 0);
   do { // find convex hull points one by one
      float maxCos = -1;
      for(auto& p : setPoints.Vtx()) { // find minimal left turn
         float len = length(p - pCur);
         if (len > 0) {
            float cosPhi = dot(dir, p - pCur) / len;
            if (cosPhi > maxCos) { maxCos = cosPhi; pNext = &p;}
         }
      }
      hullPoints.Vtx().push_back(*pNext);       // save as convex hull
      dir = normalize(*pNext - pCur); // prepare for next
      pCur = *pNext;
   } while(pLow != pNext);
}
```

Applikáció

```
class ConvexHullApp : public glApp {
   ConvexHull* hull = nullptr;
   vec2* pickedPoint = nullptr;
   GPUProgram gpuProgram;  // árnyaló programok
   vec2 PixelToNDC(int pX, int pY);
public:
   ConvexHullApp() : glApp("ConvexHull") { }
   void onInitialization() {
      glViewport(0, 0, winWidth, winHeight);
      glLineWidth(3);
      glPointSize(10);
      hull = new ConvexHull;
      gpuProgram.create(vertexSource, fragmentSource);
   }
   void onDisplay() {
      glClearColor(0, 0, 0, 0);
      glClear(GL_COLOR_BUFFER_BIT);
      hull->Draw(&gpuProgram);
   }
   void onMousePressed(MouseButton button, int pX, int pY);
   void onMouseReleased(MouseButton button, int pX, int pY);
   void onMouseMotion(int pX, int pY);
} app;
```

Controller

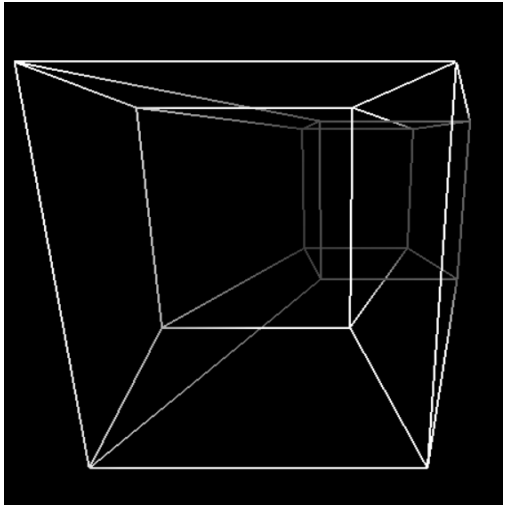
```
vec2 ConvexHullApp:: PixelToNDC(int pX, int pY) {
   return vec2(2.0f * pX / winWidth - 1, 1.0f - 2.0f * pY / winHeight);
}
void ConvexHullApp::onMousePressed(MouseButton button, int pX, int pY) {
   if (button == MOUSE_LEFT) {
      hull->addPoint(PixelToNDC(pX, pY));
      hull->update();
      refreshScreen();
   }
   if (button == MOUSE_RIGHT) {
      pickedPoint = hull->pickPoint(PixelToNDC(pX, pY));
   }
}
void ConvexHullApp::onMouseReleased(MouseButton button, int pX, int pY) {
   if (button == MOUSE_RIGHT) pickedPoint = nullptr;
}
void ConvexHullApp::onMouseMotion(int pX, int pY) {
   if (pickedPoint) {
      *pickedPoint = PixelToNDC(pX, pY);
      hull->update();
      refreshScreen();
   }
}
```



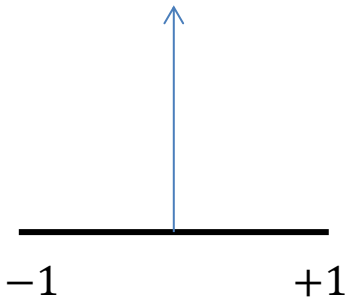
Grafikus hardver/szoftver alapok

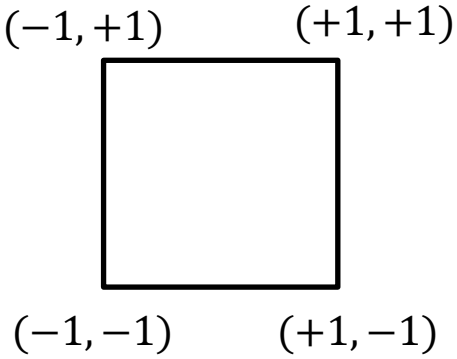
Program: Tesseract (4D kocka), animáció

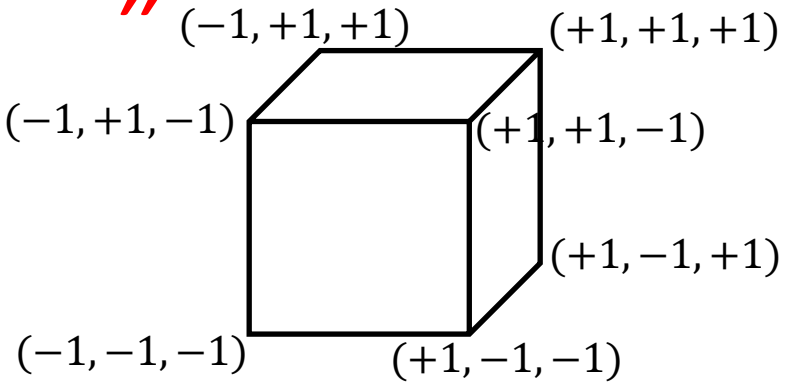
Szirmay-Kalos László

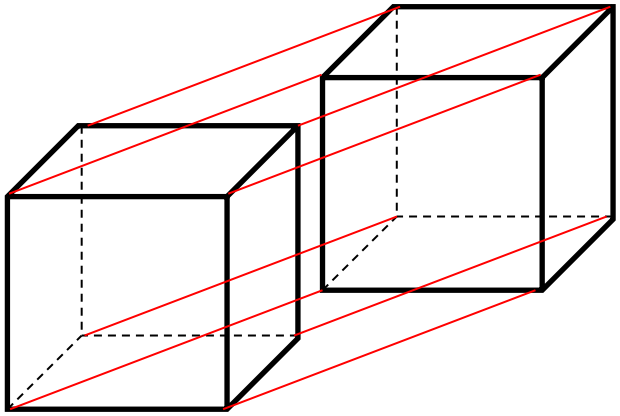


N-dimenziós „kocka”









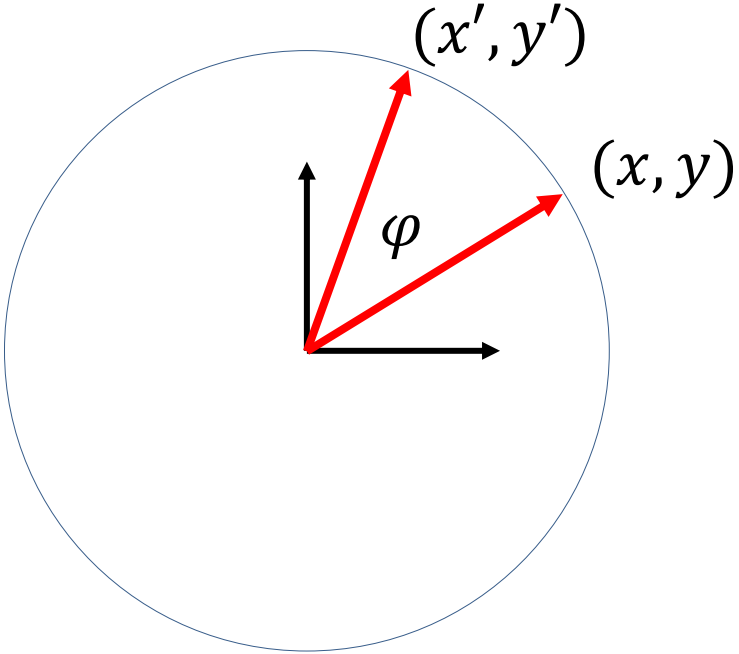
Tesseract:

- Csúcsok: $2^4 = 16$ darab
  $(\pm 1, \pm 1, \pm 1, \pm 1)$
- Élek: $\binom{4}{1}2^3 = 32$ darab
  1 csúcsból 4, 1 Hamming-ra
- Lap: $\binom{4}{2}2^2 = 24$
- Határoló 3D test: $\binom{4}{3}2 = 8$

Tesseract

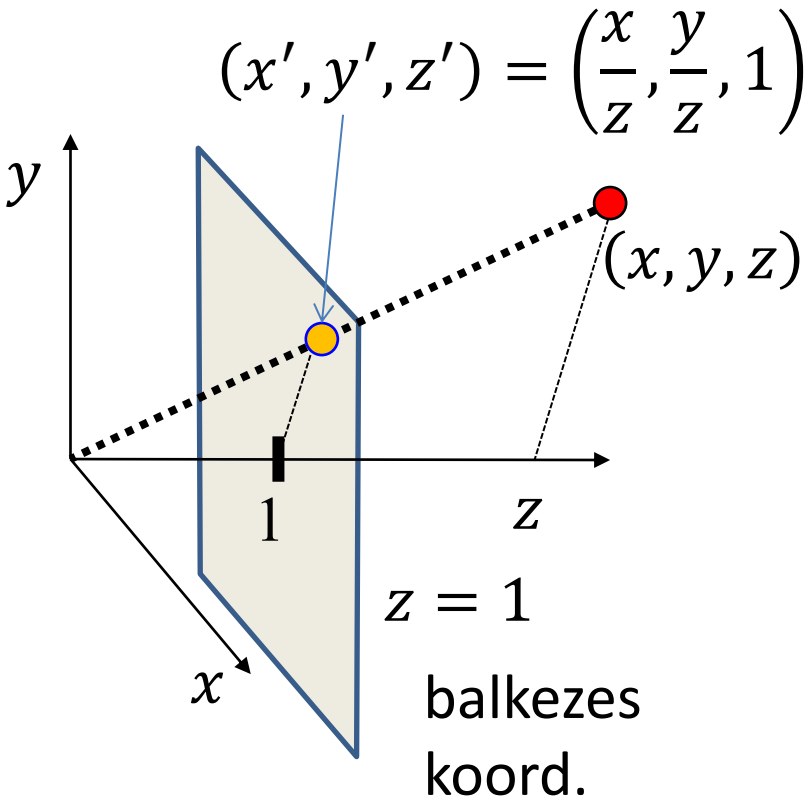
```
class Tesseract : public Geometry<vec4> {
   float alpha, beta;
   void addPoint(int c) {
      vec4 p(c&1 ? 1 : -1, c&2 ? 1 : -1, c&4 ? 1 : -1, c&8 ? 1 : -1);
      vtx.push_back(p);
   }
public:
   Tesseract() {
      const int maxcode = 15;
      for (int code = 0; code <= maxcode; code++) {
         for (int bit = 1; bit <= maxcode; bit <<= 1) {
            if ((code & bit) == 0) { addPoint(code); addPoint(code + bit); }
         }
      }
      updateGPU();
   }
   void Animate(float t) { alpha = t / 2; beta = t / 3; }
   void Draw(GPUProgram* gpuProgram) {
      gpuProgram->setUniform(cosf(alpha), "cosa");
      gpuProgram->setUniform(sinf(alpha), "sina");
      gpuProgram->setUniform(cosf(beta), "cosb");
      gpuProgram->setUniform(sinf(beta), "sinb");
      Bind();
      glDrawArrays(GL_LINES, 0, (int)vtx.size());
   }
};
```

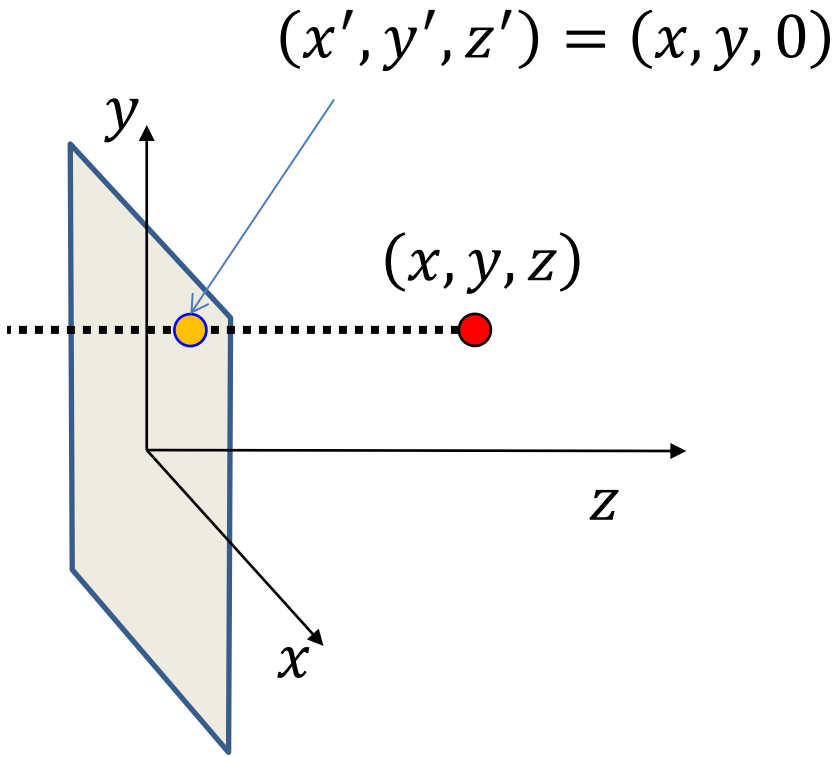
2D forgatás



$$x' = x\cos(\varphi) - y\sin(\varphi)$$
$$y' = x\sin(\varphi) + y\cos(\varphi)$$

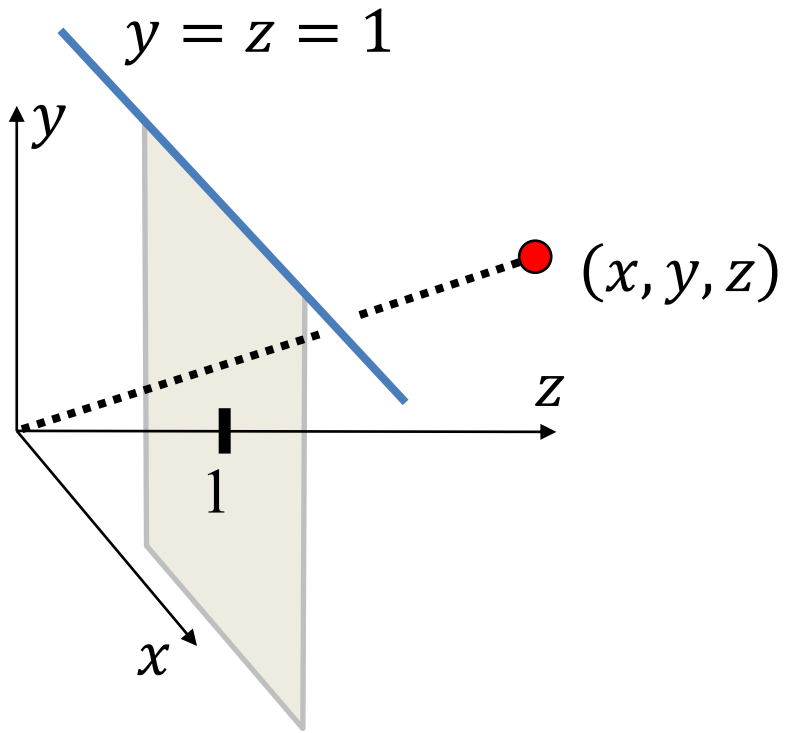
3D→2D vetítés



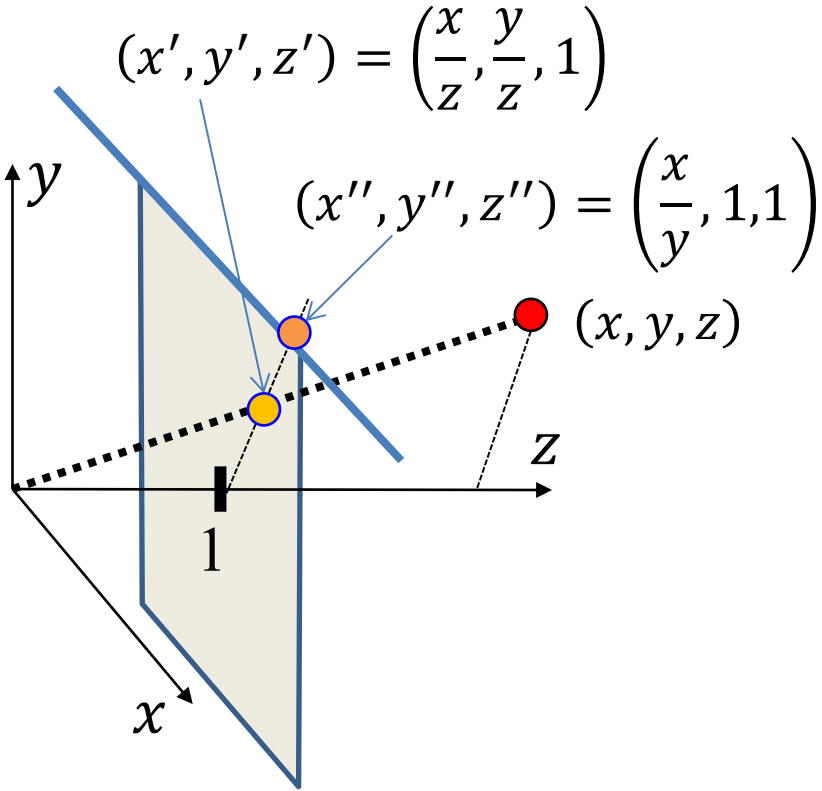


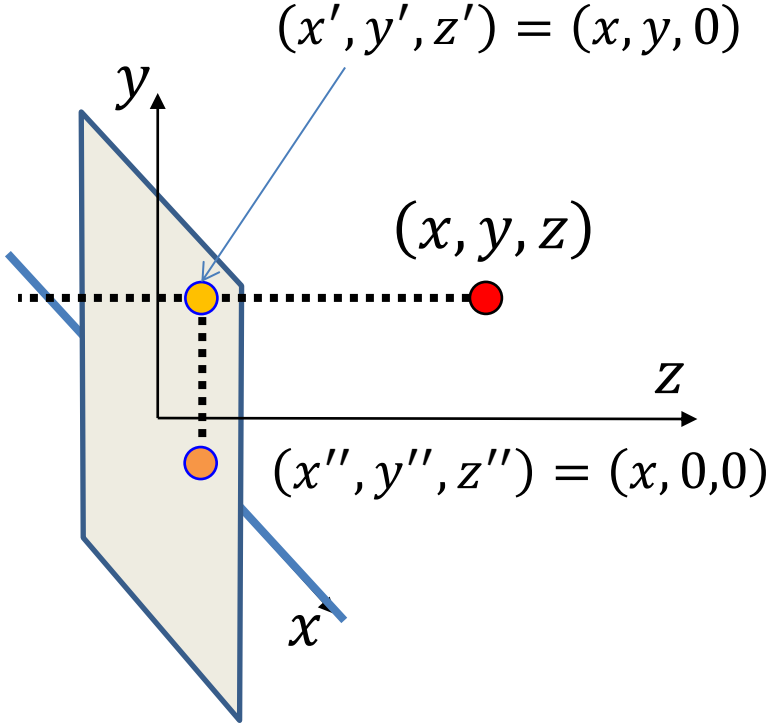
4D→2D vetítés

3D→1D vetítés



3D→2D→1D vetítés





4D→3D→2D vetítés

- 4D→3D $(w = 1)$: $(x', y', z', w') = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)$
- 3D→2D $(z = 1)$: $(x'', y'', z'') = \left(\frac{x'}{z'}, \frac{y'}{z'}, 1\right)$
- 4D→2D $(w = z = 1)$: $(x'', y'', z'', w'') = \left(\frac{x}{z}, \frac{y}{z}, 1,1\right)$
- Homogén koordináták: $\left(\frac{x}{z}, \frac{y}{z}, 1,1\right) \sim (x, y, z, z)$

Vertex és fragment árnyalók

```
uniform float cosa, sina, cosb, sinb;
layout(location = 0) in vec4 vtx;
out float depthCue;

void main() {
   vec4 vr1, vr2;
   vr1 = vec4(vtx.xy, vtx.z*cosa-vtx.w*sina, vtx.z*sina+vtx.w*cosa);
   vr2 = vec4(vr1.x*cosb-vr1.w*sinb, vr1.yz, vr1.x*sinb+vr1.w*cosb);
   vec4 p4d = vr2 * 0.4f + vec4(0, 0, 1, 1);
   depthCue = 1 / (dot(p4d, p4d) - 0.4f);
   gl_Position = vec4(p4d.xyz, p4d.z);
}
```

```
in float depthCue;
out vec4 fragColor;

void main() {
   fragColor = vec4(depthCue, depthCue, depthCue, 1);
}
```