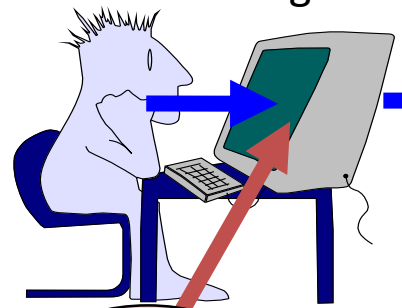
”In theory, there is no difference between theory and practice. In practice, there is.”
Benjamin Brewster

Grafikus hardver/szoftver alapok
Építőelemek

Szirmay-Kalos László

Interaktív rendszer: Funkcionális modell



Éppen belemerül
a virtuális világba

Bemeneti csővezeték

Eszköz-
képtér

Inv.
Viewport
transzf

Inv.
kamera
transzf

Inv.
Modell
transzf.

CPU:
operációs
rendszer

Fejre
állítva!

kamera

Virtuális világ

vektorizáció

CPU: app

Kép
frissítés

fix hw

rasztertár

Pixel
proc

Raszteri-
záció

Viewport
transzf

Vágás

Kamera
transzf

Modell
transzf.

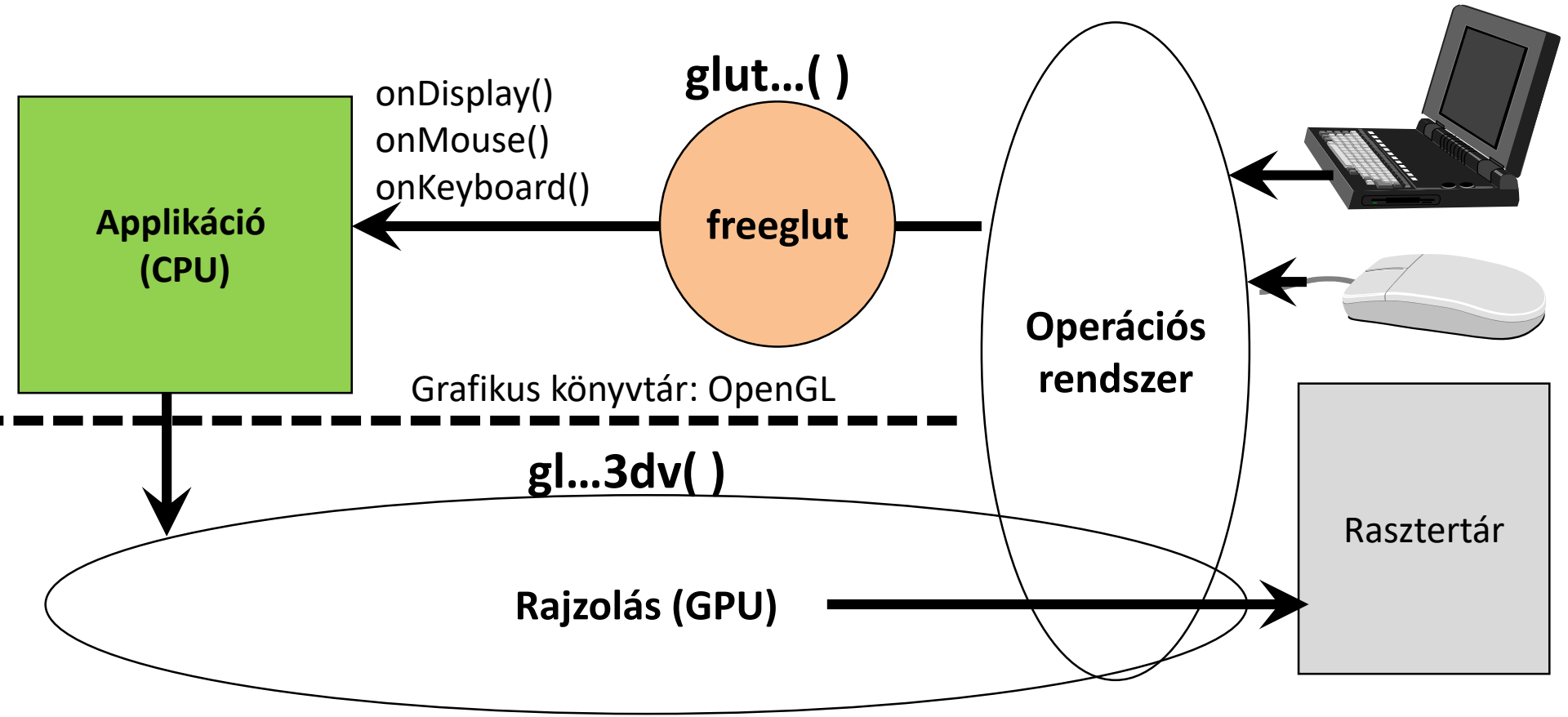
VAO
VBO

pixel shader+blending

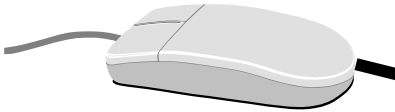
vertex shader

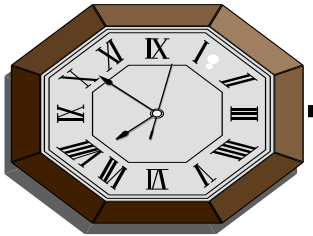
Kimeneti csővezeték: GPU

Szoftver architektúra



Esemény kezelés (GLUT)





Operációs és ablakozó rendszer (Windows)

Ablak létrehozás
callback regisztráció

freeglut

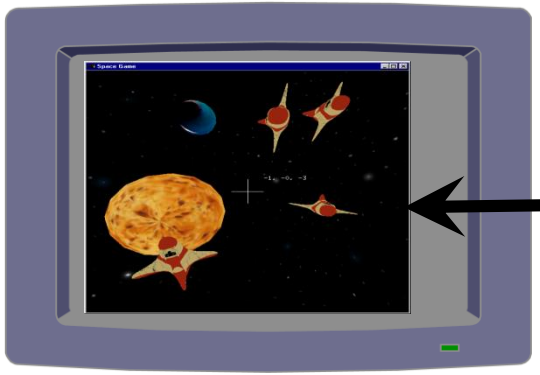
callback-ek

main
DisplayFunc
KeyboardFunc
KeyboardUpFunc
SpecialFunc
Reshape
MouseFunc
MotionFunc
IdleFunc

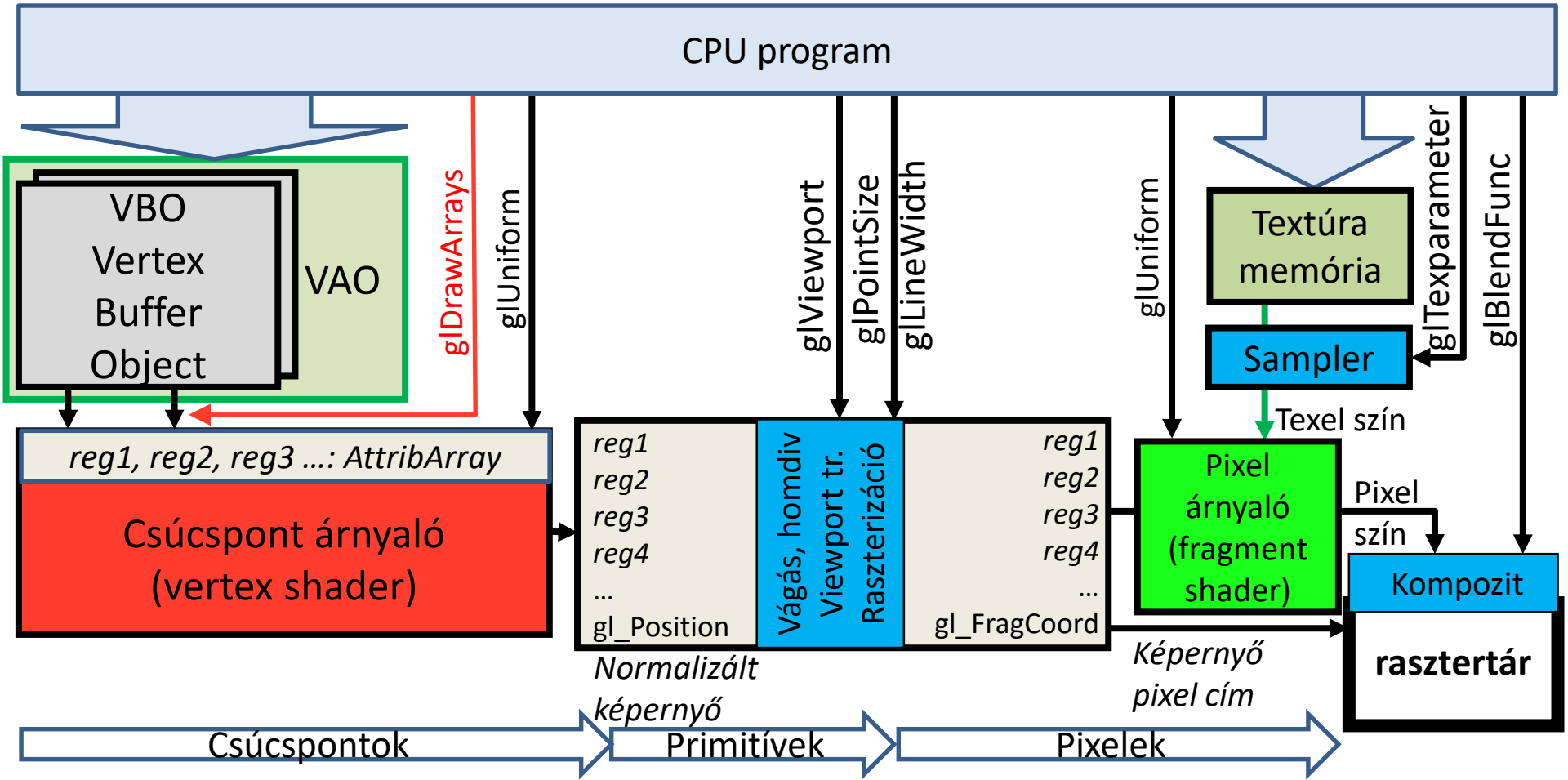
applikáció

OpenGL

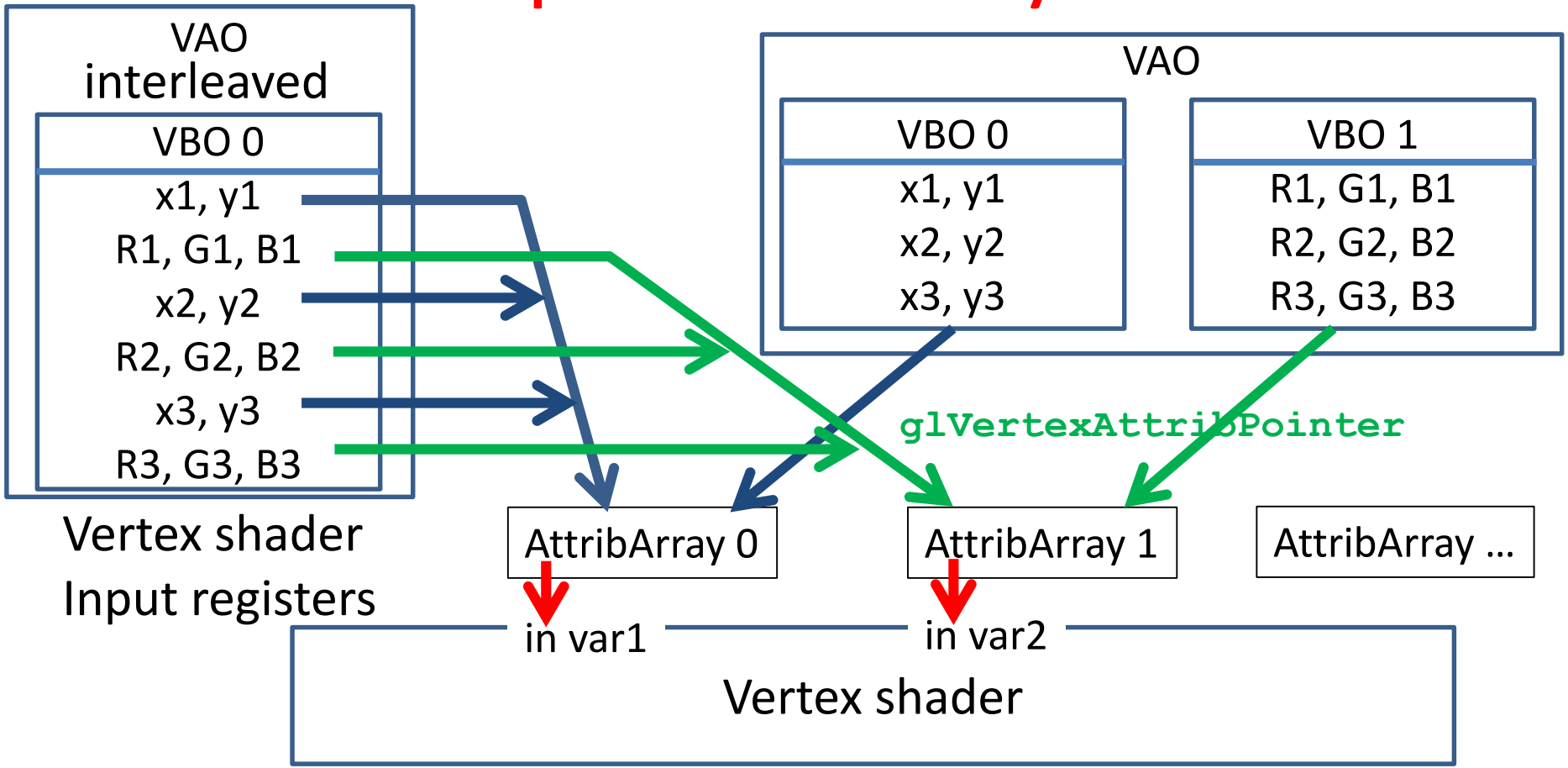
grafikus hardver



OpenGL 3.3 … 4.6 (Modern OpenGL)

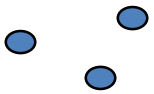


Csúcspont adatfolyamok



Rajzolás: glDrawArrays, OpenGL primitívek

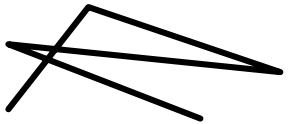
```
glBindVertexArray(vao);
glDrawArrays(primitiveType, startIdx, numOfElements);
```



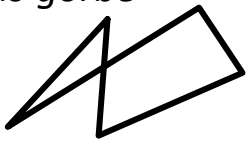
GL_POINTS

GL_LINES

Vektorizált parametrikus görbe

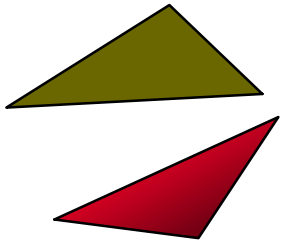


GL_LINE_STRIP



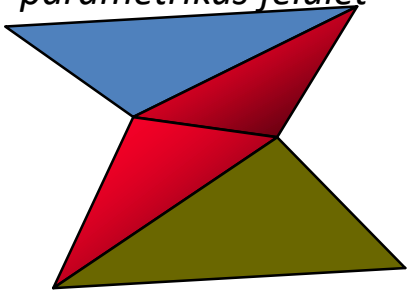
GL_LINE_LOOP

Fülvágó kimenete



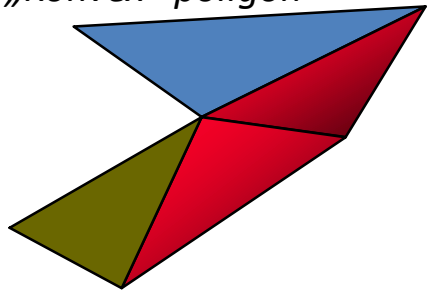
GL_TRIANGLES

Tesszellált 3D parametrikus felület



GL_TRIANGLE_STRIP

„Konvex” poligon



GL_TRIANGLE_FAN

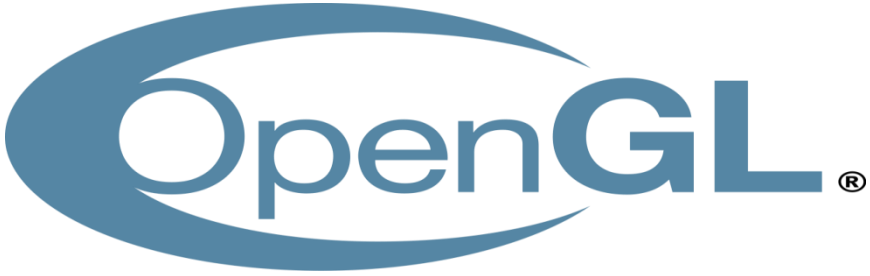
OpenGL állapotgép + erőforrások

Gagyi grafikus könyvtár

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

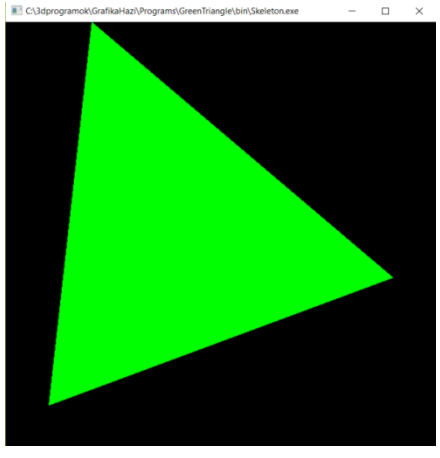
OpenGL

```
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!!!
```



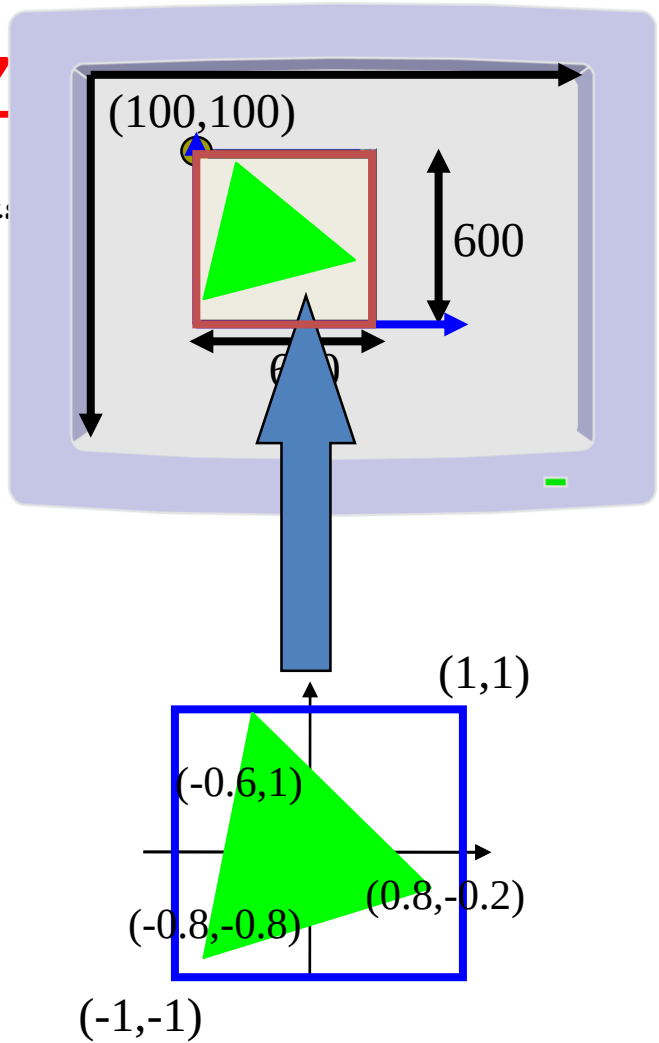
Grafikus hardver/szoftver alapok
2. Helló OpenGL/GLSL/GLUT

Szirmay-Kalos László

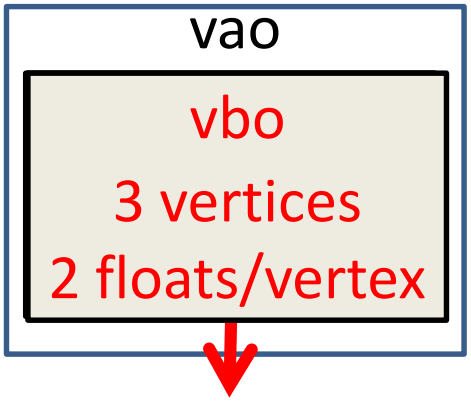


Az első OpenGL programom: Z

```
#include <windows.h>       // Only in MsWin
#include <GL/glew.h>       // MsWin/XWin, download (Ikits
#include <GL/freeglut.h>   // download (Mark Kilgard)

int main(int argc, char * argv[]) {
   glutInit(&argc, argv); // init glut
   glutInitContextVersion(3, 3); // OpenGL 3.3
   glutInitWindowSize(600, 600);
   glutInitWindowPosition(100, 100);
   glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
   glutCreateWindow("Hi Graphics");

   glewExperimental = true; // magic
   glewInit(); // NO OPENGL CALLS BEFORE THIS ☠💣!!!
   glViewport(0, 0, 600, 600);
   onInitialization();

   glutDisplayFunc(onDisplay); //event handler

   glutMainLoop(); // message loop
   return 1;
}
```



onInitialization()




```
unsigned int prog;       // GPU program id
unsigned int vao, vbo;   // virtual world on the GPU

void onInitialization() {
    glGenVertexArrays(1, &vao); // Generate 1 vao
    glBindVertexArray(vao);     // make it active
    glGenBuffers(1, &vbo);      // Generate 1 buffer
    glBindBuffer(GL_ARRAY_BUFFER, vbo); // make it active
    float vertices[] = {-0.8f,-0.8f, -0.6f,1.0f, 0.8f,-0.2f}; // Geometry
    glBufferData(GL_ARRAY_BUFFER,      // Copy to GPU target
                 sizeof(vertices),     // # bytes
                 vertices,             // address
                 GL_STATIC_DRAW);      // we do not change later
    glEnableVertexAttribArray(0);  // AttribArray 0
    glVertexAttribPointer(0,       // vbo -> AttribArray 0
      2, GL_FLOAT, GL_FALSE,       // two floats/attrib, not fixed-point
      0, NULL);                    // stride, offset: tightly packed
```

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

C++11

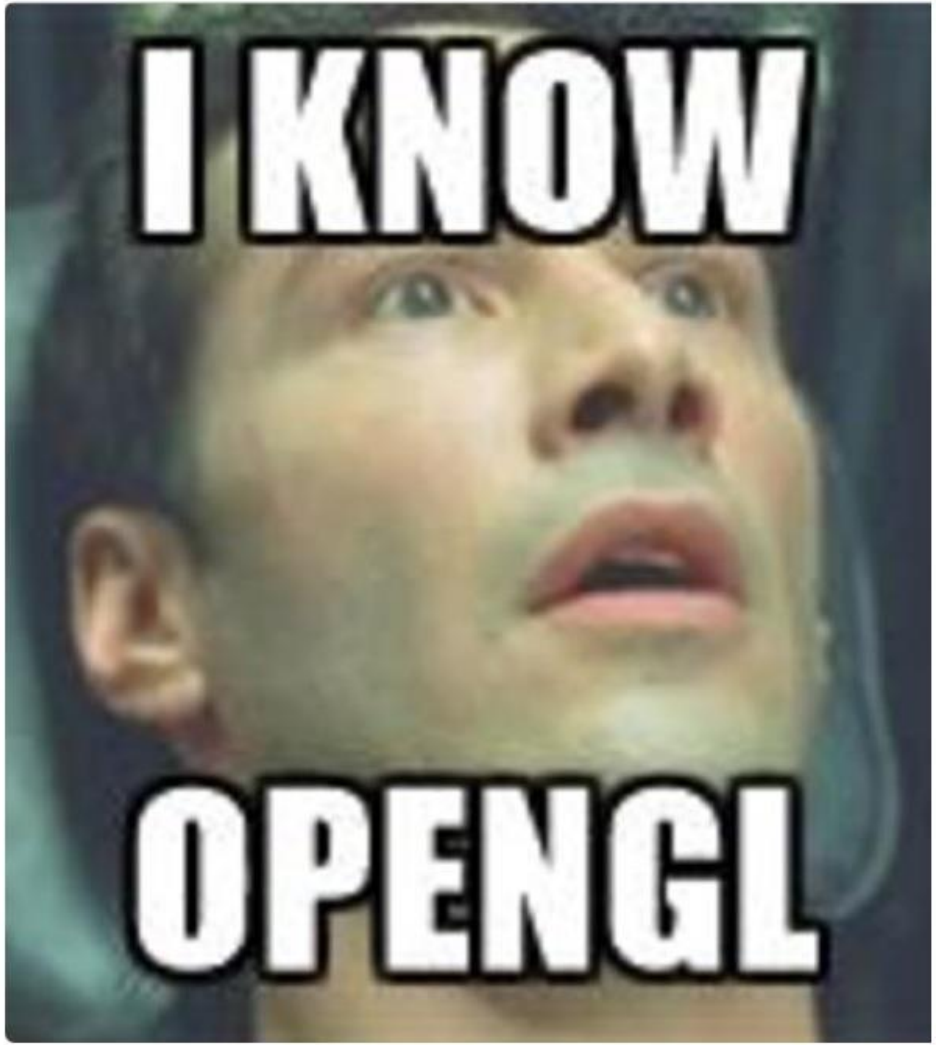
```
#version 330
precision highp float;
uniform vec3 color;
out vec4 outColor;
void main() {
    outColor = vec4(color,1);
}
```

```
const char * vertSource = R"( … )"; // OR from file
const char * fragSource = R"( … )";
unsigned int vertShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertShader, 1, &vertSource, NULL);
glCompileShader(vertShader);

unsigned int fragShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragShader, 1, &fragSource, NULL);
glCompileShader(fragShader);

prog = glCreateProgram(); // global variable
glAttachShader(prog, vertShader); glAttachShader(prog, fragShader);

glBindFragDataLocation(shaderProgram, 0, "outColor");
glLinkProgram(shaderProgram);
glUseProgram(shaderProgram); // make it active
}
```

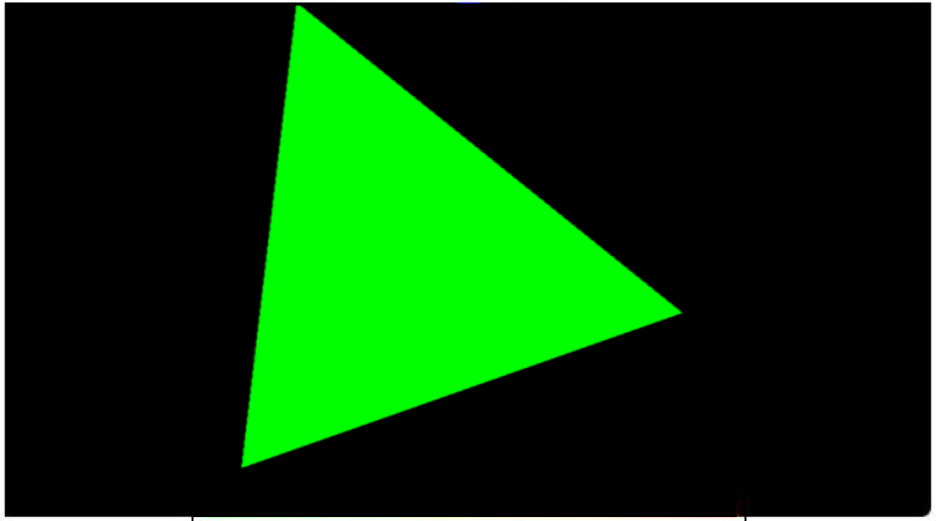
```
#version 330
precision highp float;
layout(location = 0) in vec2 vp;
void main() {
  gl_Position = vec4(vp.x,vp.y,0,1);
}
```

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

```
void onDisplay( ) {
    glClearColor(0, 0, 0, 0);      // background color
    glClear(GL_COLOR_BUFFER_BIT); // clear frame buffer

    // Set color to (0, 1, 0) = green
    int location = glGetUniformLocation(prog, "color");
    glUniform3f(location, 0.0f, 1.0f, 0.0f); // 3 floats

    glBindVertexArray(vao);
    // Draw call
    glDrawArrays(GL_TRIANGLES, 0 /*startIdx*/, 3 /*# Elements*/);

    glutSwapBuffers( ); // exchange buffers for double buffering
}
```


I KNOW
OPENGL


SHOW ME



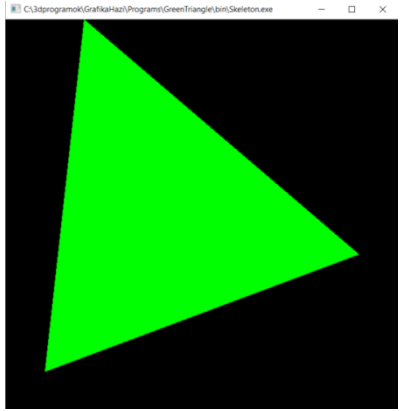
”The GL in OpenGL stands for Good Luck because you are going to need it.”

Anonymous

Grafikus hardver/szoftver alapok

Program: Keret és zöld háromszög

Szirmay-Kalos László



OpenGL starters' kit: Shader programs

- `glCreate[Shader|Program]()` létrehozás
- `glShaderSource()` forrás feltöltés
- `glCompileShader()` fordítás
- `glAttachShader()` shader hozzáadás programhoz
- `glBindFragDataLocation()` rasztertárba mi megy?
- `glLinkProgram()` szerkesztés
- `glUseProgram()` kiválasztás futásra
- `glGetUniformLocation()` uniform változó cím lekérdez
- `glUniform*()` uniform változó értékadás

```
class GPUProgram { // shader management with error handling
   …
   bool create(char* vertShader, char * fragShader,
               char * outputName, char * geomShader = 0);
   void Use();
   void setUniform(…);
};
```

OpenGL starters' kit

Erőforrás létrehozás, feltöltés és aktivizálás

- `glGen[VertexArrays|Buffers|Textures](nIds, ids);`
- `glBind[VertexArray|Buffer|Texture](type, id);`
- `glBufferData(type, nBytes, source, target);`
- `glTexImage2D(type, …);`

Bufferek vertex shader bemeneti regiszterekhez kötése

- `glEnableVertexAttribArray()` — regiszter engedély
- `glVertexAttribPointer(reg, …)` — bufferből mely regiszterbe

Rajzolás és csővezeték management

- `glDrawArrays(type, start, n)` — vao bufferek felrajzolása
- `glClearColor(r, g, b, a)` — háttér törlési szín
- `glClear(buffer)` — háttér törlés
- `glViewport(l, b, w, h)` — fénykép méret
- `glPointSize(size)` — pont méret
- `glLineWidth(width)` — vonal vastagság

framework.h

```
include: <stdio.h>, <stdlib.h>, <math.h>, <vector>, <string>
         if windows <windows.h>
         <GL/glew.h>, <GL/freeglut.h> // must be downloaded
const unsigned int windowWidth = 600, windowHeight = 600;
struct vec2, vec3, vec4, mat4;

struct Texture {
   unsigned int textureId;
   void create(…);
};

class GPUProgram {
   bool create(char * vertShader,
               char * fragShader, char * outputName,
               char * geomShader = nullptr);
   void Use();
   void setUniform(…);
};
```

framework.cpp

```
#include "framework.h"
void onInitialization(); // Init
void onDisplay(); // Redraw
void onKeyboard(unsigned char key, int pX, int pY); // Key pressed
void onKeyboardUp(unsigned char key, int pX, int pY); // Key released
void onMouseMotion(int pX, int pY); // Move mouse with key pressed
void onMouse(int button, int state, int pX, int pY); // Mouse click
void onIdle(); // Time elapsed

int main(int argc, char * argv[]) {
    glutInit(&argc, argv); glutInitContextVersion(3, 3);
    glutInitWindowSize(windowWidth, windowHeight); glutInitWindowPosition(100, 100);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutCreateWindow(argv[0]);
    glewExperimental = true; glewInit();
    onInitialization();
    glutDisplayFunc(onDisplay); // Register event handlers
    glutMouseFunc(onMouse);
    glutIdleFunc(onIdle);
    glutKeyboardFunc(onKeyboard);
    glutKeyboardUpFunc(onKeyboardUp);
    glutMotionFunc(onMouseMotion);
    glutMainLoop(); return 1;
}
```

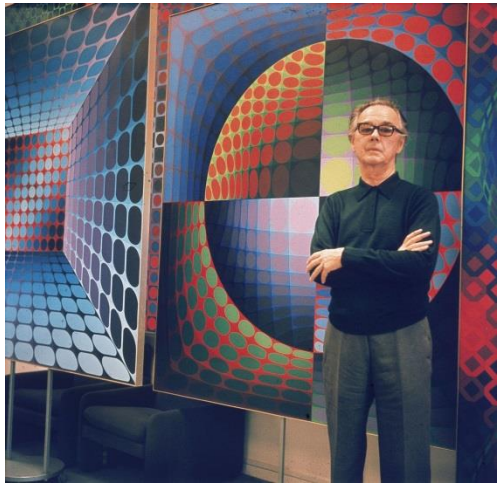
Skeleton.cpp

```
#include "framework.h"
char * vertexSource = R"(…)";
char * fragmentSource = R"(…)";

GPUProgram gpuProgram;
unsigned int vao, vbo;

void onInitialization() {
    glViewport(0, 0, windowWidth, windowHeight);
    glGenVertexArrays(1, &vao); glBindVertexArray(vao);
    glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo);
    float vtx[] = {-0.8f, -0.8f,  -0.6f, 1.0f,  0.8f, -0.2f};
    glBufferData(GL_ARRAY_BUFFER, sizeof(vtx), vtx, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, NULL);
    gpuProgram.create(vertexSource, fragmentSource, "outColor");
}

void onDisplay() {
    glClearColor(0, 0, 0, 0);     // background color
    glClear(GL_COLOR_BUFFER_BIT); // clear frame buffer
    gpuProgram.setUniform(vec3(0, 1, 0), "color");
    glBindVertexArray(vao);
    glDrawArrays(GL_TRIANGLES, 0, 3);
    glutSwapBuffers(); // exchange buffers for double buffering
}
```

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

onInitialization

```
#include "framework.h"
GPUProgram gpuProgram;
unsigned int vao, vbo;
const int nVertices = 100;
void onInitialization() {
    glViewport(0, 0, windowWidth, windowHeight);
    glGenVertexArrays(1, &vao); glBindVertexArray(vao);
    glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo);
    std::vector<vec2> vtx(nVertices);
    for (int i = 0; i < nVertices; i++) {
        float phi = i * 2.0f * M_PI / nVertices;
        vtx[i] = vec2(cosf(phi), sinf(phi)); // unit radius circle
    }
    int nBytes = vtx.size() * sizeof(vec2);
    glBufferData(GL_ARRAY_BUFFER, nBytes, vtx.data(), GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, NULL);
    gpuProgram.create(vertexSource, fragmentSource, "outColor");
}
```

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

```
float rmax = 1.0f, amplitude = 0.8f;

void onDisplay() {
   glClearColor(0, 0, 0, 0);     // background color
   glClear(GL_COLOR_BUFFER_BIT); // clear frame buffer

   const int nCircles = 20;
   const float r0 = rmax, r1 = rmax / nCircles;
   vec3 ce0(1, 0, 0), ce1(0, 0, 0), co0(0, 0, 0.5f), co1(0, 1, 1);

   for (int i = 0; i < nCircles; i++) {
      float t = (float)i / (nCircles - 1.0f);
      gpuProgram.setUniform(r0 * (1 - t) + r1 * t, "radius");
      gpuProgram.setUniform(t * (1 - t) * amplitude, "up");
      vec3 c = (i%2 == 0) ? ce0*(1-t) + ce1*t : co0*(1-t) + co1*t;
      gpuProgram.setUniform(c, "color");
      glDrawArrays(GL_TRIANGLE_FAN, 0, nVertices);
   }
   glutSwapBuffers(); // exchange buffers for double buffering
}
```

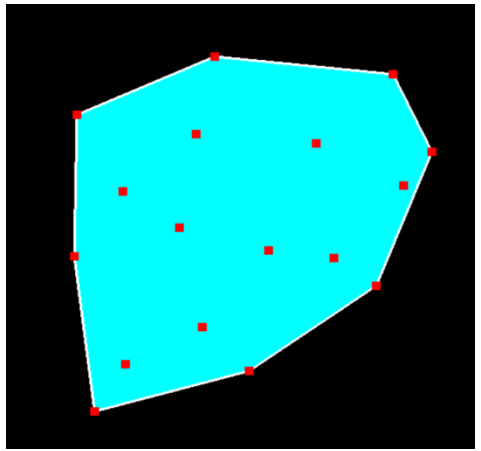
Vezérlés

```
bool pressed[256] = { false, }; // keyboard state

void onKeyboard(unsigned char key, int pX, int pY) {
   pressed[key] = true;
}

void onKeyboardUp(unsigned char key, int pX, int pY) {
   pressed[key] = false;
}

void onIdle() {
   float time = glutGet(GLUT_ELAPSED_TIME) / 1000.0f;
   if (pressed['s']) rmax = 0.7f + 0.3f * sinf(time);
   if (pressed['h']) amplitude = 0.9f * sinf(3 * time);
   glutPostRedisplay(); // redraw the scene
}
```

Grafikus hardver/szoftver alapok

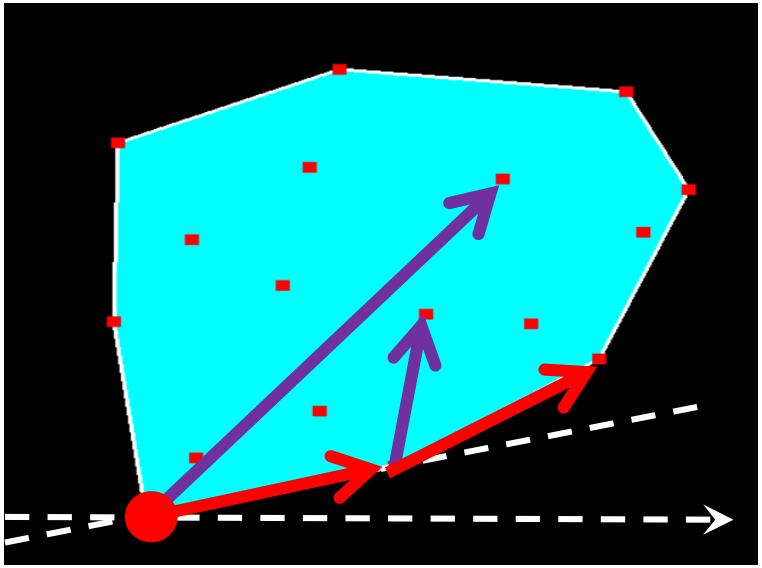
Program: Konvex burok, interakció

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 vertexPosition;

void main() {
   gl_Position = vec4(vertexPosition, 0, 1); // in NDC
}
```

Fragment shader:

```
uniform vec3 color;
out vec4 fragmentColor;

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

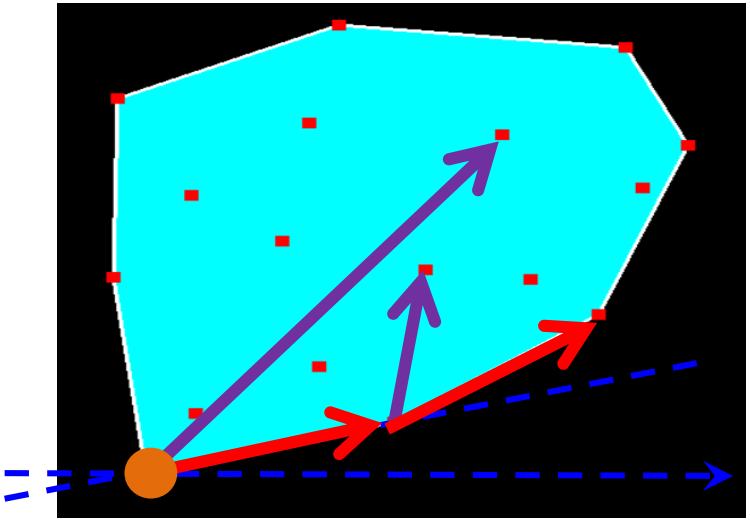
Constant color primitive object

```
class Object {
   unsigned int vao, vbo;   // GPU
   vector<vec2> vtx;        // CPU
public:
   Object() {
      glGenVertexArrays(1, &vao); glBindVertexArray(vao);
      glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo);
      glEnableVertexAttribArray(0); glVertexAttribPointer(0,2,GL_FLOAT,GL_FALSE,0,NULL);
   }
   vector<vec2>& Vtx() { return vtx; }
   void updateGPU() {       // CPU -> GPU
      glBindVertexArray(vao); glBindBuffer(GL_ARRAY_BUFFER,vbo);
      glBufferData(GL_ARRAY_BUFFER, vtx.size() * sizeof(vec2), &vtx[0], GL_DYNAMIC_DRAW);
   }
   void Draw(int type, vec3 color) {
      if (vtx.size() > 0) {
         glBindVertexArray(vao);
         gpuProgram.setUniform(color, "color");
         glDrawArrays(type, 0, vtx.size());
      }
   }
};
```

Convex hull

```
class ConvexHull {
   Object points, hull; // points and hull
public:
   void addPoint(vec2 p) { points.Vtx().push_back(p); }
   void update() {
      if (points.Vtx().size() >= 3) findHull(points.Vtx(), hull.Vtx());
      points.updateGPU();
      hull.updateGPU();
   }
   vec2 *pickPoint(vec2 pp, float threshold) {
      for (auto& p : points.Vtx()) if (length(pp - p) < threshold) return &p;
      return nullptr;
   }
   void Draw() {
      hull.Draw(GL_TRIANGLE_FAN, vec3(0, 1, 1));
      hull.Draw(GL_LINE_LOOP, vec3(1, 1, 1));
      points.Draw(GL_POINTS, vec3(1, 0, 0));
   }
};
```

Convex hull előállítás



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

   hullps.clear();
   vec2 pCur = *pLow, *pNext, dir(1, 0);
   do { // find convex hull points one by one
      float maxCos = -1;
      for(auto& p : ps) { // 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;}
         }
      }
      hullps.push_back(*pNext);        // save as convex hull
      dir = normalize(*pNext - pCur); // prepare for next
      pCur = *pNext;
   } while(pLow != pNext);
}
```

Virtuális világ és megjelenítése

```
ConvexHull * world;
vec2 * pickedPoint = nullptr;
GPUProgram gpuProgram;

void onInitialization() {
    glViewport(0, 0, windowWidth, windowHeight);
    glLineWidth(2);
    glPointSize(10);
    world = new ConvexHull;
    gpuProgram.create(vertexSrc, fragmentSrc, "fragmentColor");
}

void onDisplay() {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    world->Draw();
    glutSwapBuffers();
}
```

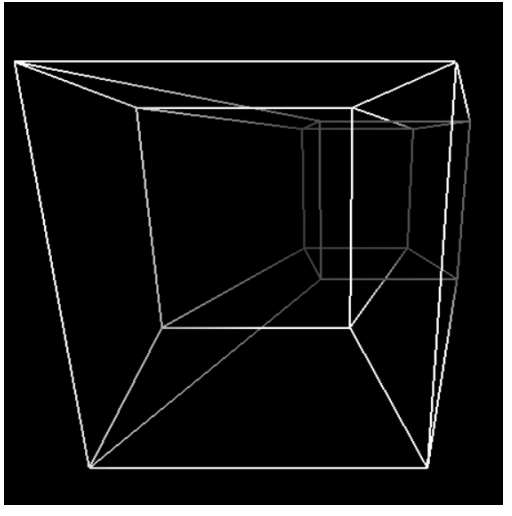
Controller

```
vec2 PixelToNDC(int pX, int pY) { // if full viewport!
   return vec2(2.0f * pX / windowWidth - 1, 1 - 2.0f * pY / windowHeight);
}

void onMouse(int button, int state, int pX, int pY) {
   if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
      world->addPoint(PixelToNDC(pX, pY));
      world->update(); glutPostRedisplay(); // redraw
   }
   if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
      pickedPoint = world->pickPoint(PixelToNDC(pX, pY), 0.05f);
   if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
      pickedPoint = nullptr;
}

void onMouseMotion(int pX, int pY) {
   if (pickedPoint) {
      *pickedPoint = vec2(PixelToNDC(pX, pY));
       world->update(); glutPostRedisplay(); // redraw
   }
}
```

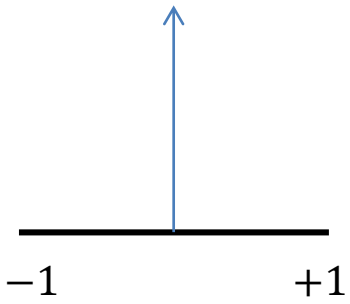
Grafikus hardver/szoftver alapok

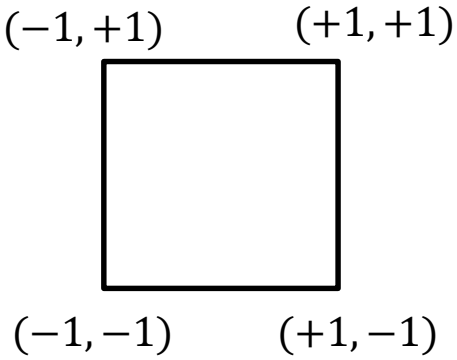
Program: Tesseract (4D kocka), animáció

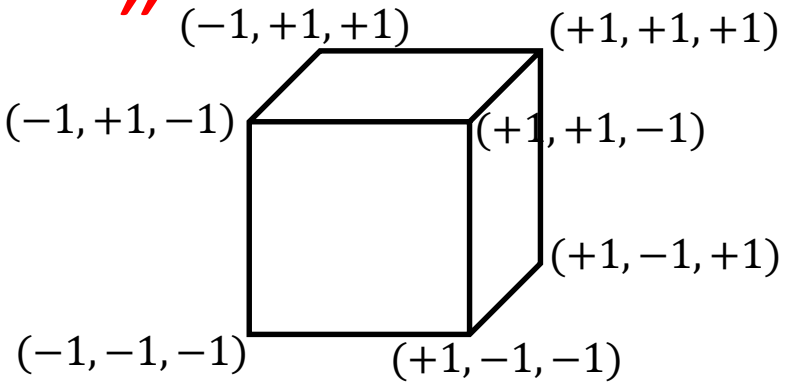
Szirmay-Kalos László

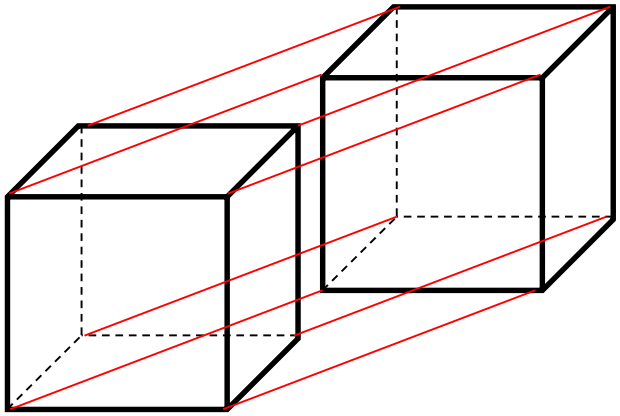


N-dimenziós „kocka”









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: $\binom{4}{3}2 = 8$

```
#include "framework.h"

const char *vertexSrc = …, *fragmentSrc = …;
GPUProgram gpuProgram; // vertex and fragment shaders

class Tesseract {
   void Animate(float t);
   void Draw();
} *cube;

void onInitialization() {
   glViewport(0, 0, windowWidth, windowHeight); glLineWidth(2);
   cube = new Tesseract;
   gpuProgram.create(vertexSrc, fragmentSrc, "fragColor");
}

void onDisplay() {
   glClearColor(0, 0, 0, 0); glClear(GL_COLOR_BUFFER_BIT);
   cube->Draw();
   glutSwapBuffers();
}

void onIdle() {
   cube->Animate(glutGet(GLUT_ELAPSED_TIME) / 1000.0f);
   glutPostRedisplay();
}
```

Tesseract objektum

```
class Tesseract {
    const int D = 4;
    const int maxcode = (1 << D) - 1;
    unsigned int vao, vbo;  // GPU: vertex array object id

public:
    Tesseract();              // copy edges to GPU
    void Animate(float t);    // set transformation
    void Draw();              // trigger GPU
};
```

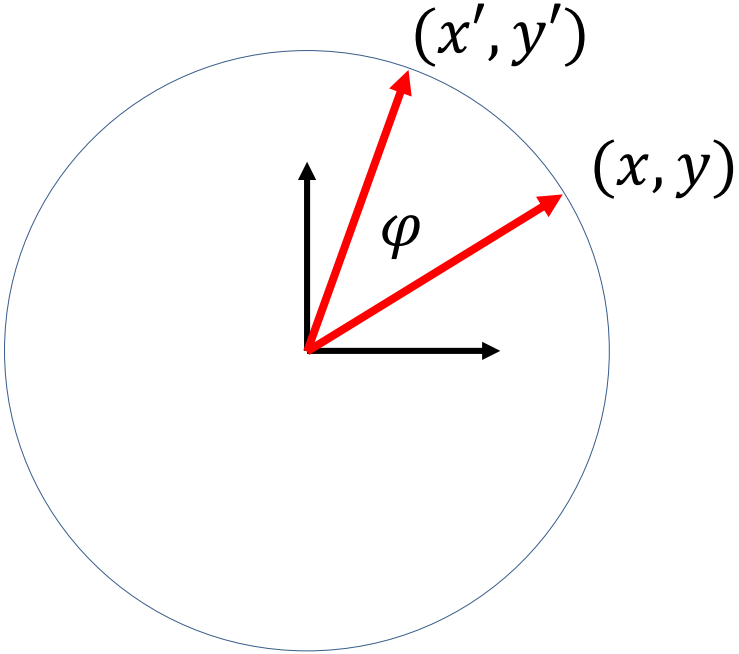
Élek a GPU-ra

```
Tesseract::Tesseract() {
   vector<float> vtx;   // CPU: vertices of the object
   for (int code = 0; code <= maxcode; code++) {
      for (int bit = 1; bit < maxcode; bit <<= 1) {
         if ((code & bit) == 0) {
           for (int b=1; b<maxcode; b<<=1) vtx.push_back((code&b)?1:-1);
           for (int b=1; b<maxcode; b<<=1) vtx.push_back(!((code+bit)&b)?1:-1);
         }
      }
   }
   glGenVertexArrays(1, &vao); glBindVertexArray(vao);
   glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo);
   glBufferData(GL_ARRAY_BUFFER, 32*8*sizeof(float), &vtx[0], GL_STATIC_DRAW);
   glEnableVertexAttribArray(0);
   glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, NULL);
}
```

Animáció és rajzolás

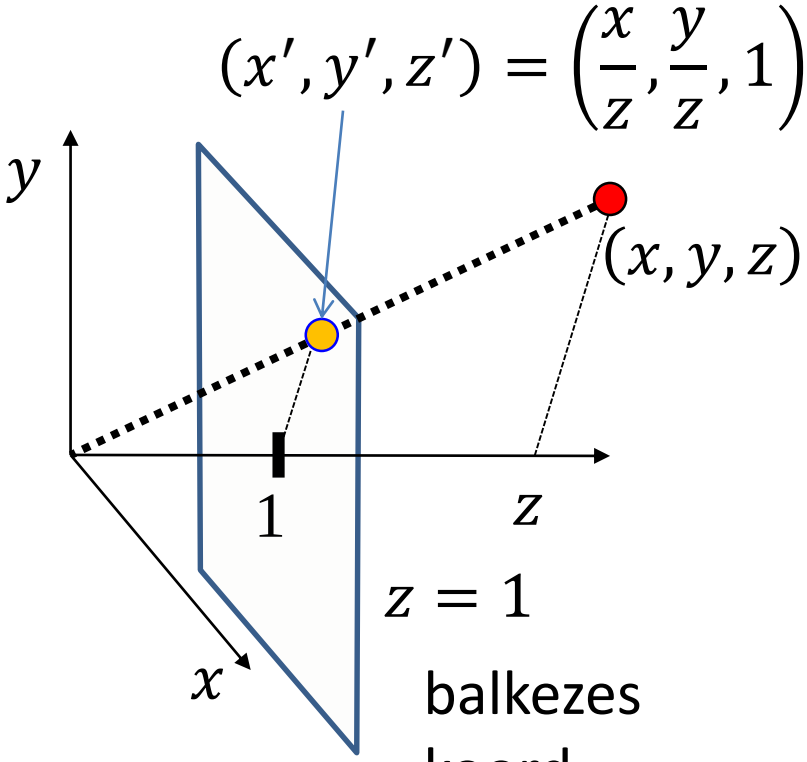
```
void Tesseract::Animate(float t) {
    gpuProgram.setUniform(cosf(t/2), "cosfi");
    gpuProgram.setUniform(sinf(t/2), "sinfi");
    gpuProgram.setUniform(cosf(t/3), "costh");
    gpuProgram.setUniform(sinf(t/3), "sinth");
}

void Tesseract:: Draw() {
    glBindVertexArray(vao);
    glDrawArrays(GL_LINES, 0, 32 * 2);
}
```

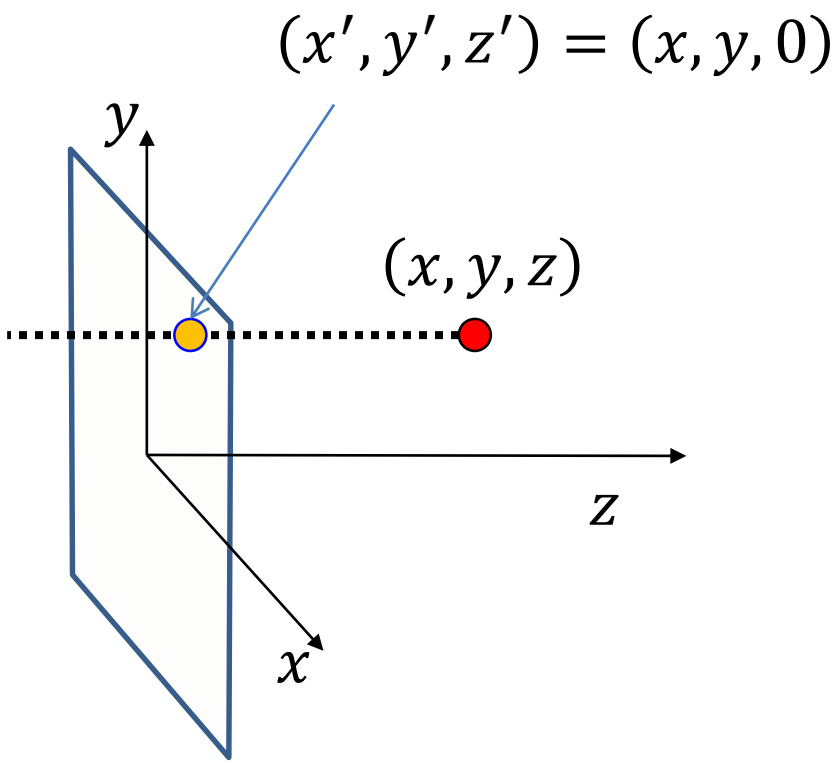
2D forgatás



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

3D→2D vetítés

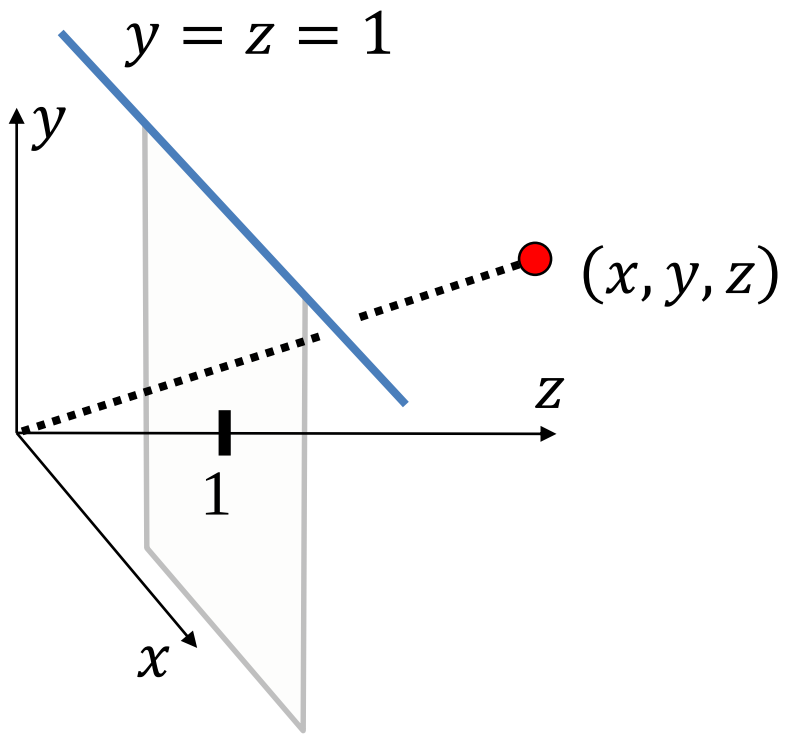




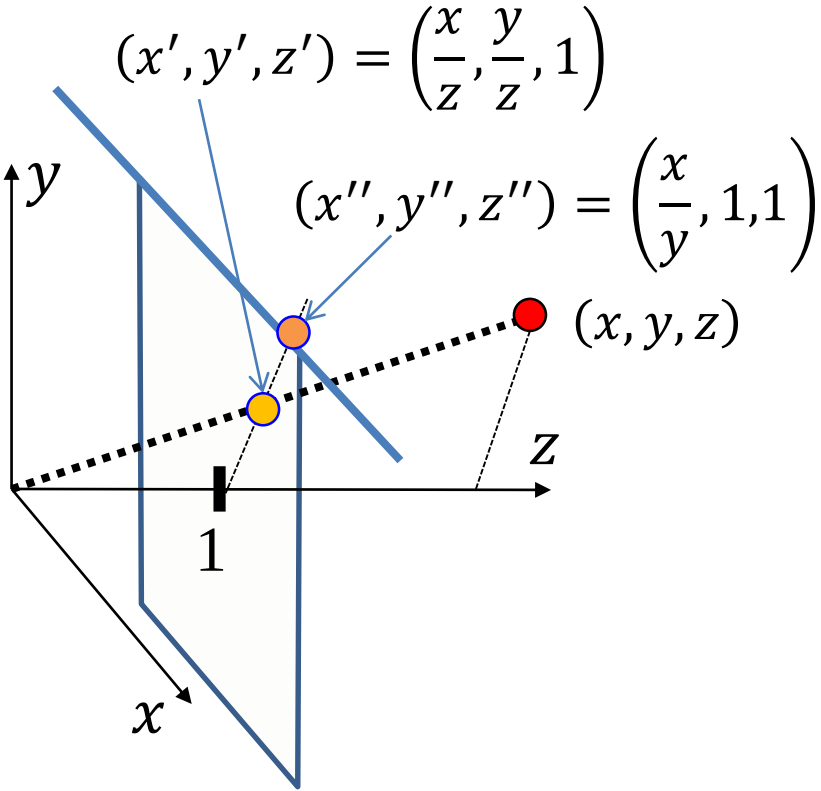
4D→2D vetítés

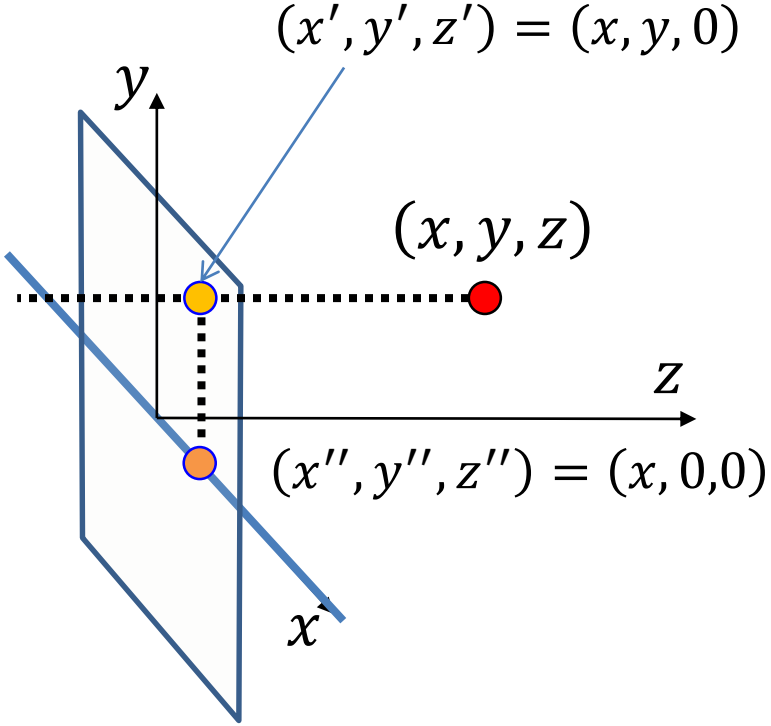
?

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 cosfi, sinfi, costh, sinth;

layout(location = 0) in vec4 vtx;
out float depthCue;

void main() {
   vec4 vr1 = vec4(vtx.xy, vtx.z*cosfi-vtx.w*sinfi, vtx.z*sinfi+vtx.w*cosfi);
   vec4 vr2 = vec4(vr1.x*costh-vr1.w*sinth, vr1.yz, vr1.x*sinth+vr1.w*costh);
   vec4 p4d = vr2 * 0.4f + vec4(0, 0, 1, 1); // scale and translate
   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);
}
```