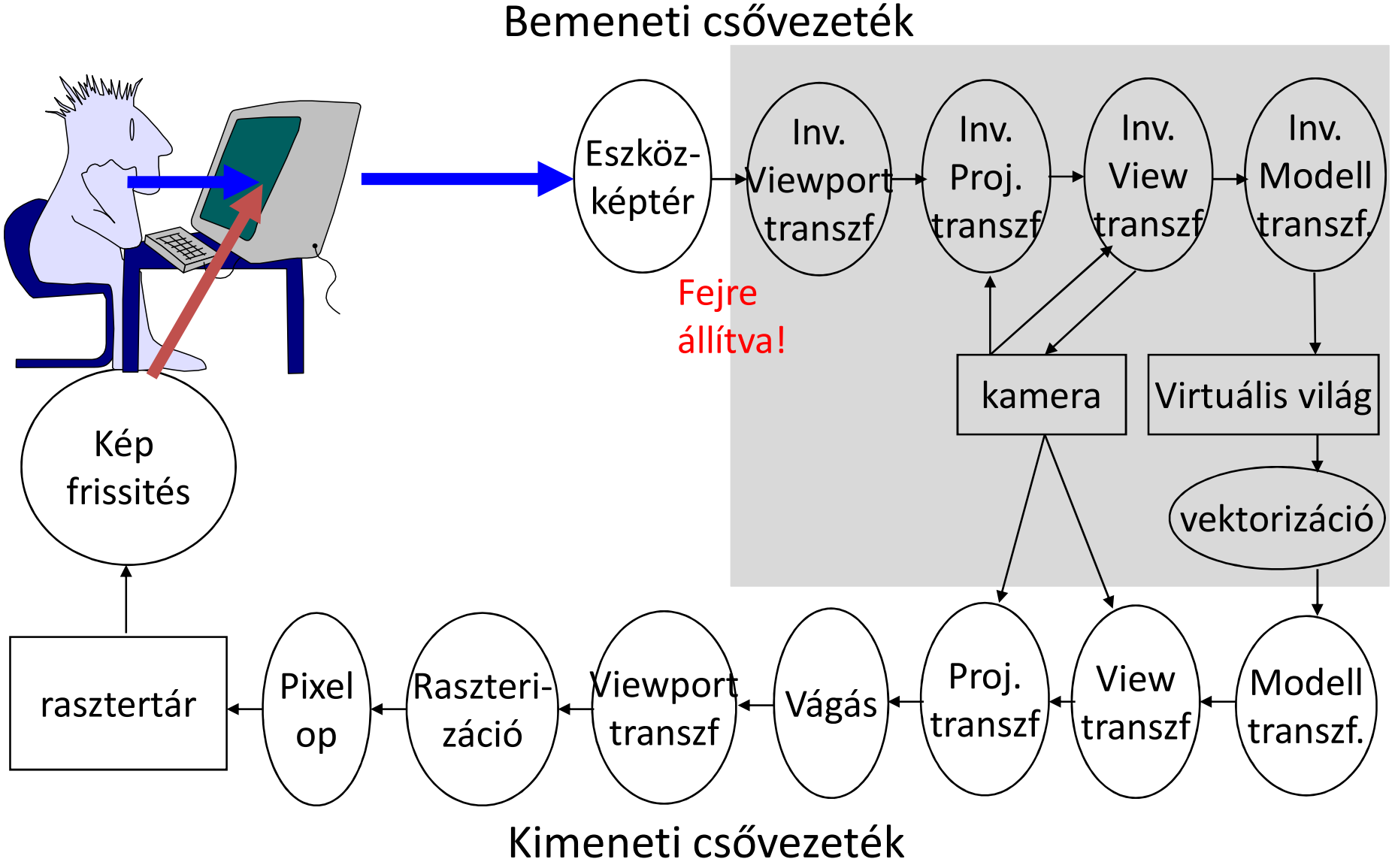
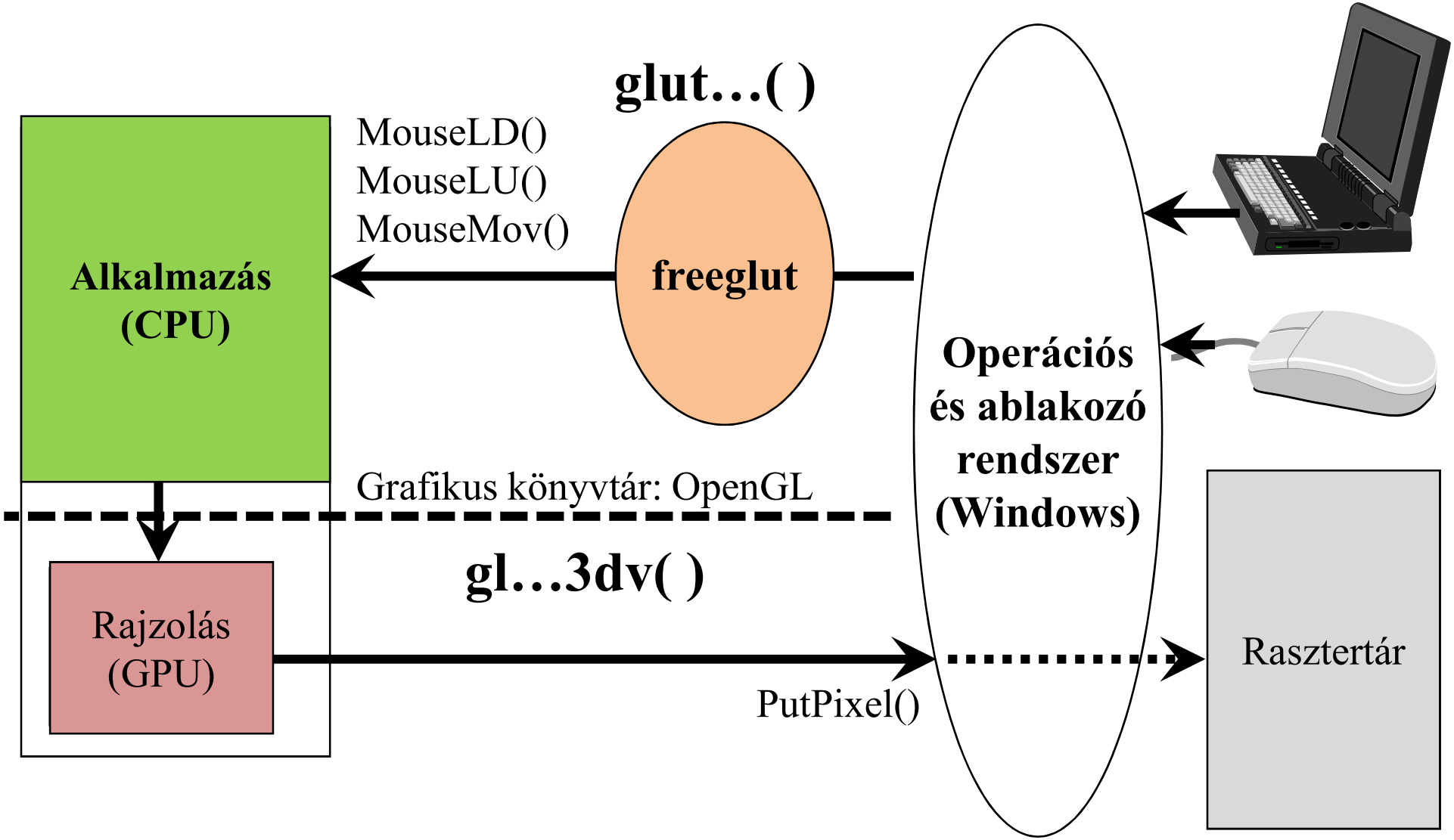
Grafikus hardver/szoftver alapok

Szirmay-Kalos László

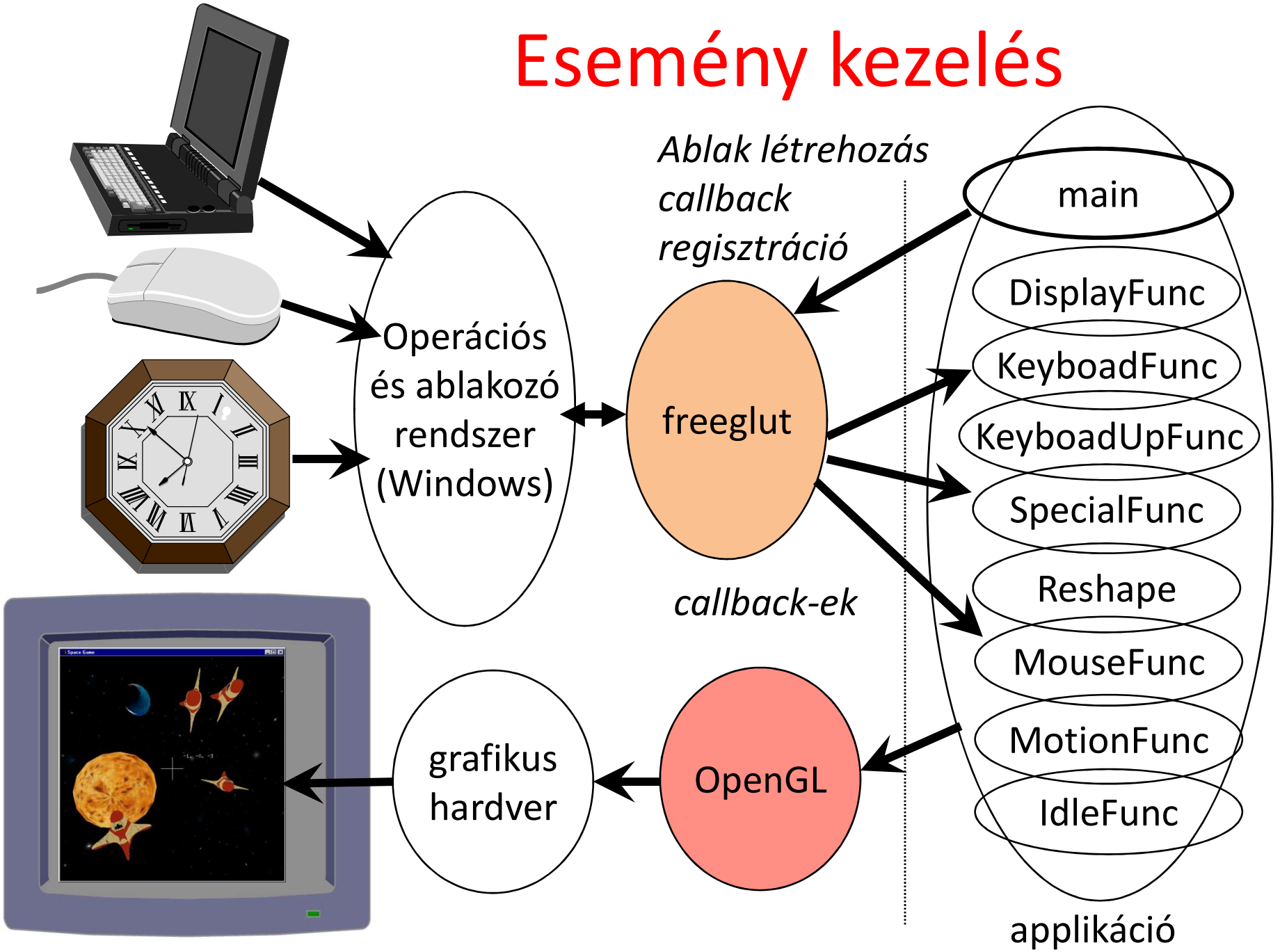
Funkcionális modell

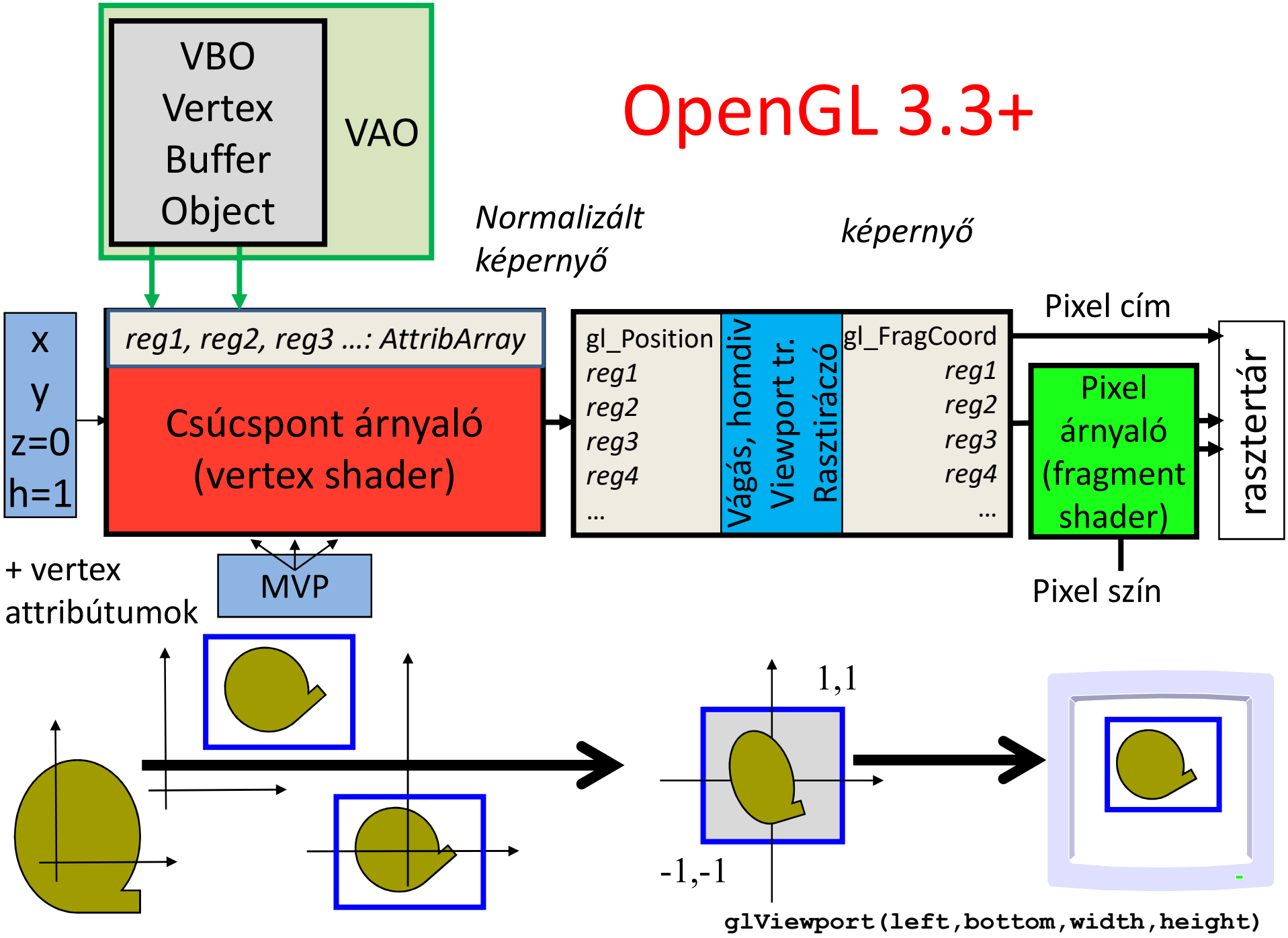


Szoftver architektúra

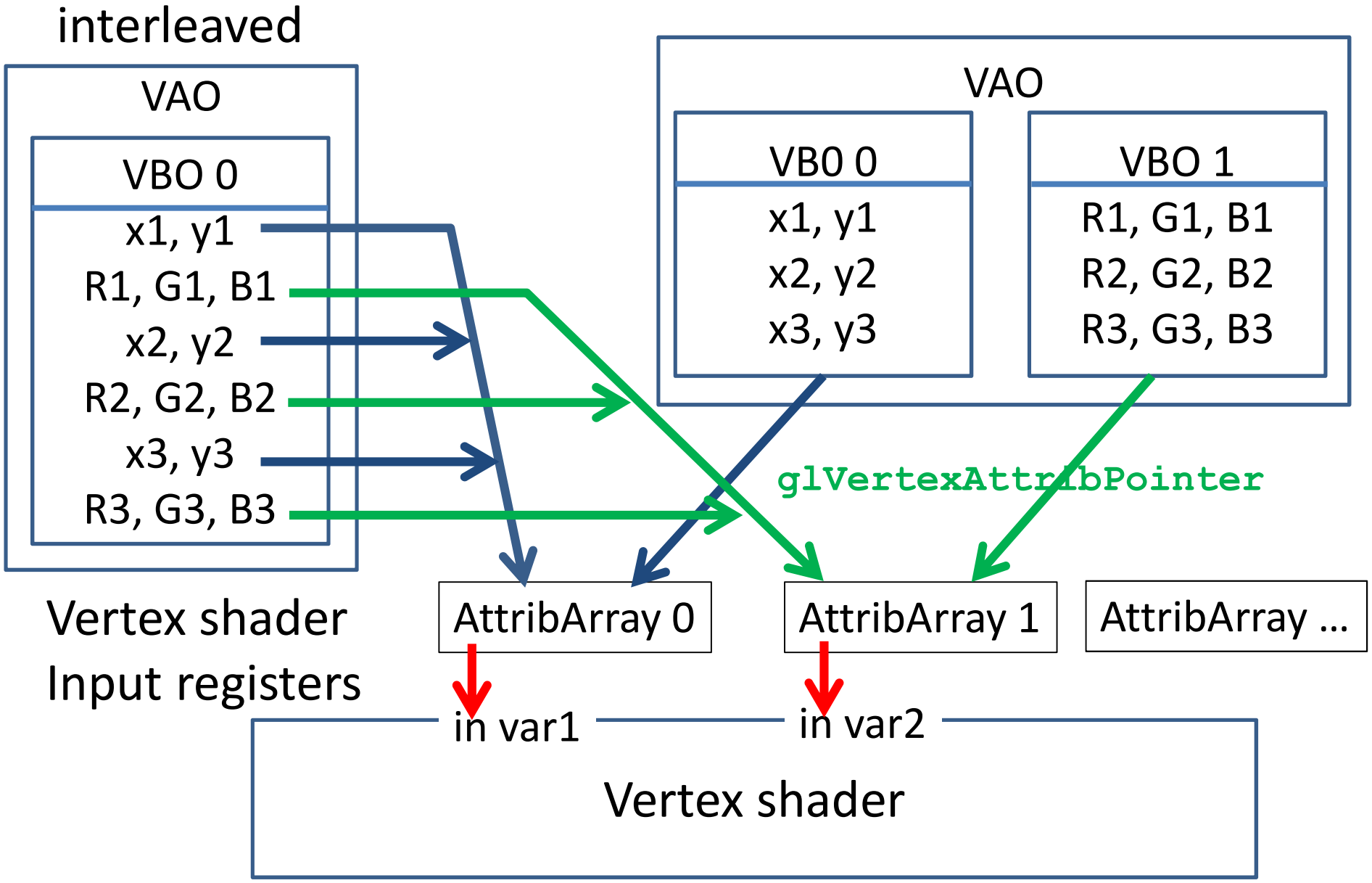


Esemény kezelés



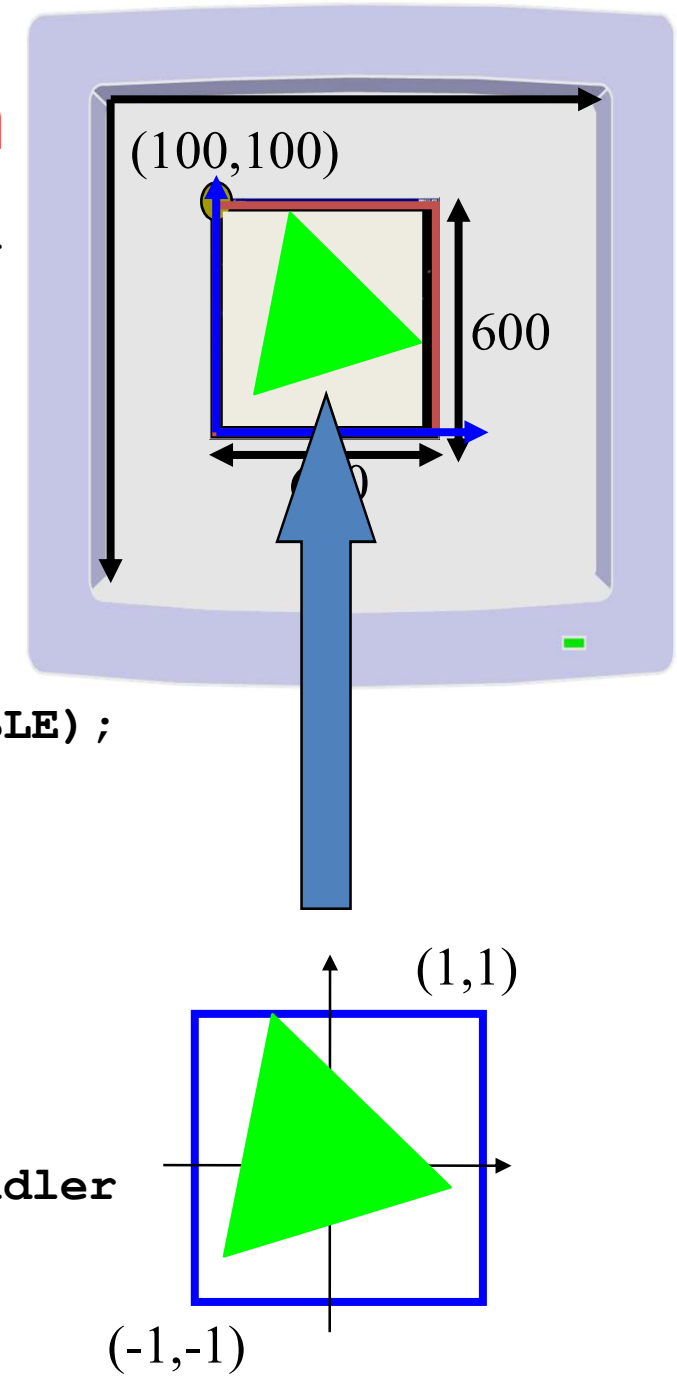

VBO
Vertex
Buffer
Object
VAO
OpenGL 3.3+
Normalizált
képernyő
képernyő
x
y
z=0
h=1
reg1, reg2, reg3 …: AttribArray
Csúcspont árnyaló
(vertex shader)
gl_Position
reg1
reg2
reg3
reg4
…
Vágás, homdiv
Viewport tr.
Rasztiráczó
gl_FragCoord
reg1
reg2
reg3
reg4
…
Pixel cím
Pixel
árnyaló
(fragment
shader)
rasztertár
Pixel szín
+ vertex
attribútumok
MVP
1,1
-1,-1
glViewport(left,bottom,width,height)

Csúcspont adatfolyamok

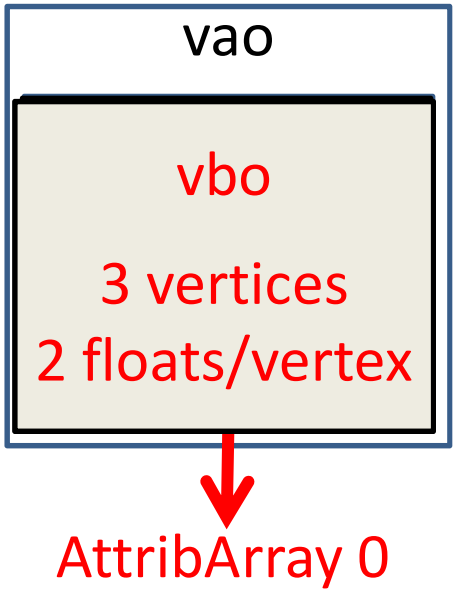


Az első OpenGL programom

```
#include <windows.h>        // Only in MsWin
#include <GL/glew.h>        // download
#include <GL/freeglut.h>    // download

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

   glutCreateWindow("Hi Graphics");
   glewExperimental = true; // magic
   glewInit(); // init glew

   glViewport(0, 0, 600, 600);
   onInitialization();

   glutDisplayFunc(onDisplay); //event handler

   glutMainLoop();
   return 1;
}
```



onInitialization()




```
unsigned int shaderProgram;
unsigned int vao; // virtual world on the GPU

void onInitialization() {
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao); // make it active

    unsigned int vbo;// vertex buffer object
    glGenBuffers(1, &vbo); // Generate 1 buffer
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    // Geometry with 24 bytes (6 floats or 3 x 2 coordinates)
    static float vertices[] = {-0.8,-0.8,  -0.6,1.0,  0.8,-0.2};
    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;
uniform mat4 MVP;
layout(location = 0) in vec2 vp;

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

```
#version 330
precision highp float;
uniform vec3 color;
out vec4 outColor;

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

```
    static const char * vertexSource = R"( … )";
    static const char * fragmentSource = R"( … )";
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexSource, NULL);
    glCompileShader(vertexShader);

    unsigned int fragmentShader=glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentSource, NULL);
    glCompileShader(fragmentShader);

    shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);

    glBindFragDataLocation(shaderProgram, 0, "outColor");

    glLinkProgram(shaderProgram);
    glUseProgram(shaderProgram);
}
```

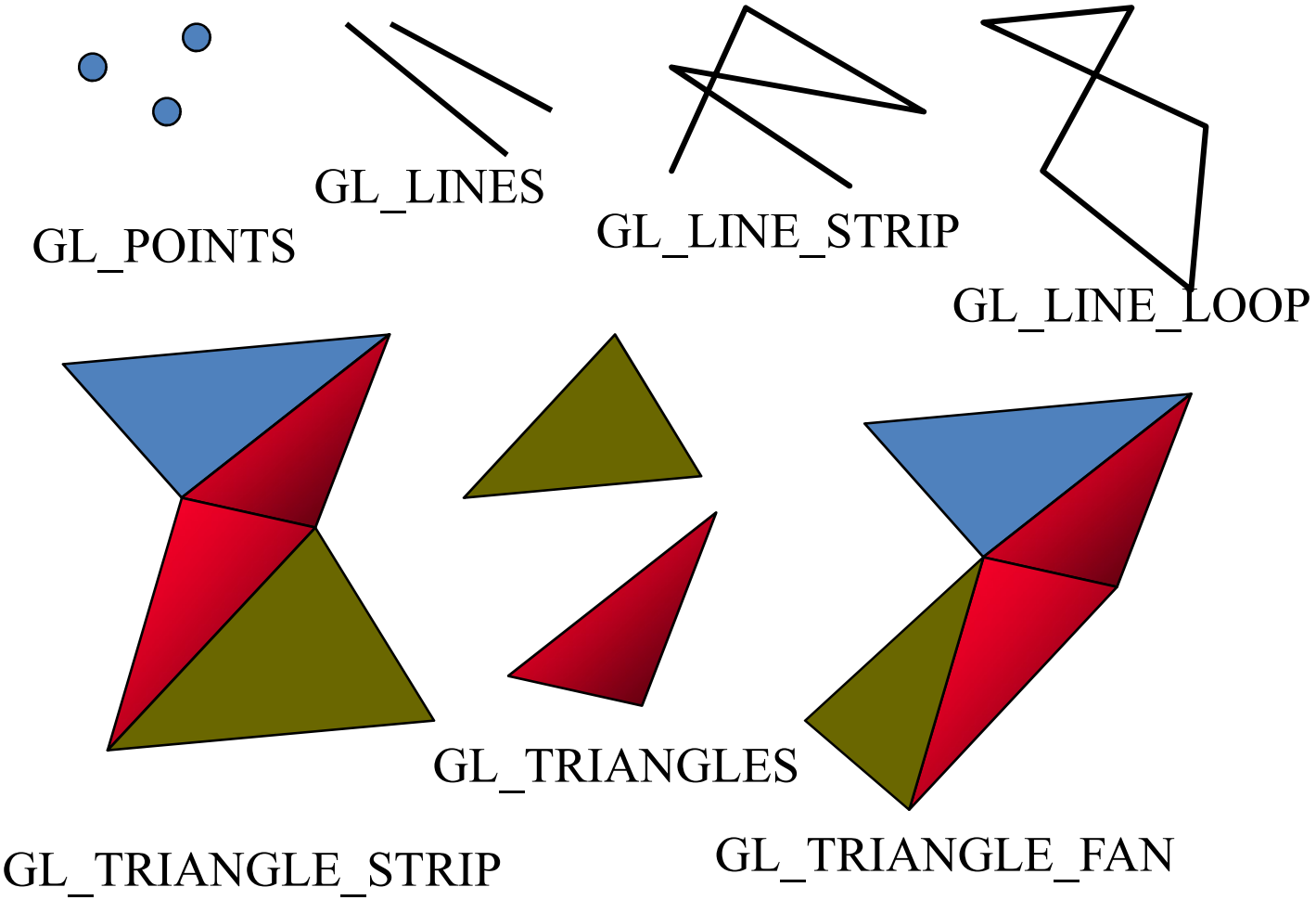
```
#version 330
precision highp float;
uniform mat4 MVP;
layout(location = 0) in vec2 vp;

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

```
#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(shaderProgram, "color");
    glUniform3f(location, 0.0f, 1.0f, 0.0f); // 3 floats

    float MVPtransf[4][4] = { 1, 0, 0, 0,     // MVP matrix,
                              0, 1, 0, 0,     // row-major!
                              0, 0, 1, 0,
                              0, 0, 0, 1 };

    location = glGetUniformLocation(shaderProgram, "MVP");
    glUniformMatrix4fv(location, 1, GL_TRUE, &MVPtransf[0][0]);

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

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

OpenGL primitívek

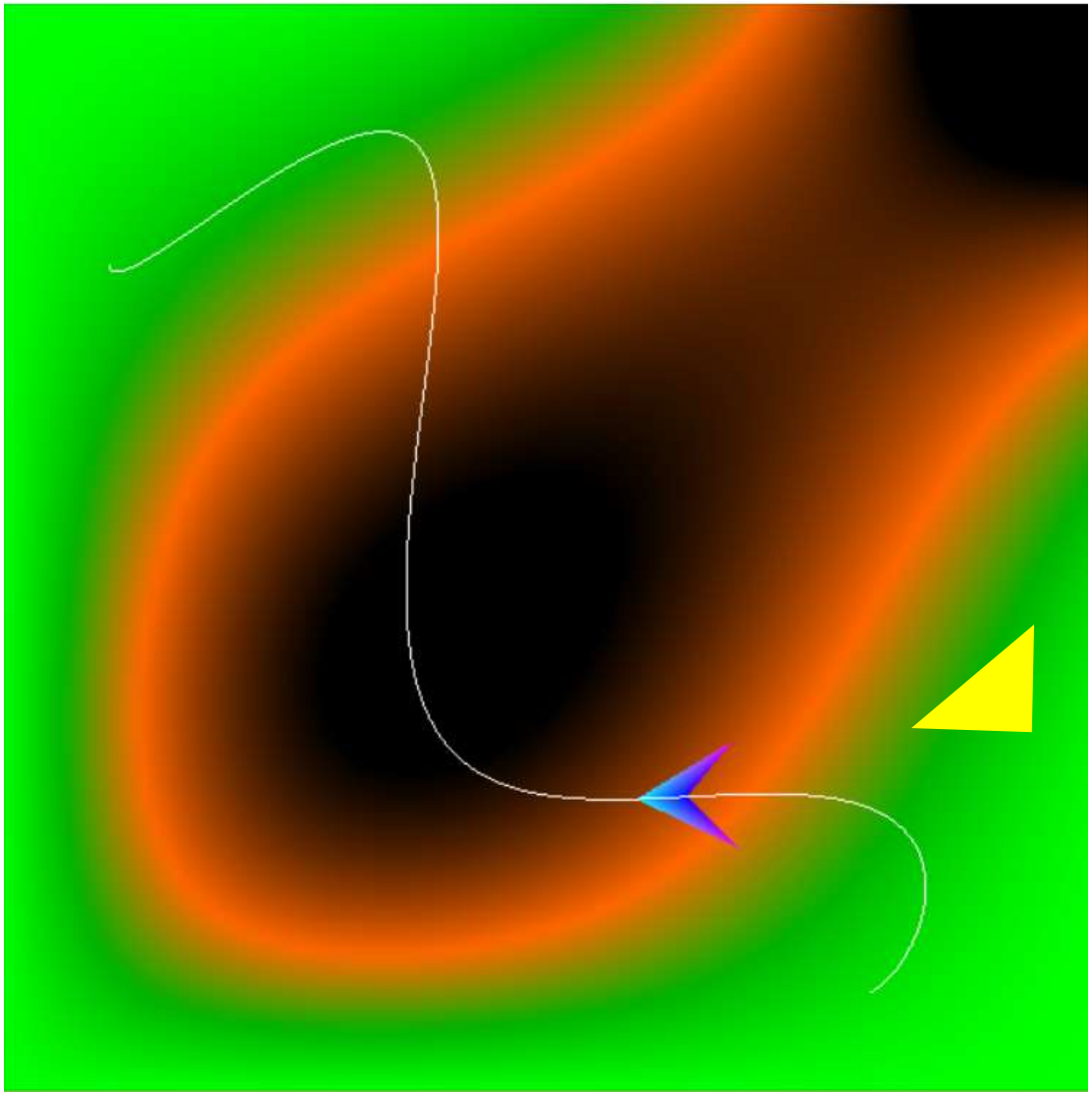
```
glDrawArrays( primitive,
              startIdx,
              numOfElements);
```



1. házi: Olimpiai hegyi kerékpárpálya

Készítsen hegyi kerékpárpálya tervezőt, amely felülnézetben, merőleges vetülettel mutatja a pályát és környékét! A terepet Bézier felülettel adjuk meg. A kontrolpontok xy vetületei szabályos N x N-es (N > 3) 2D rácsot alkotnak az 1km x 1km tartományban. A tervezőprogramban a magasságot térképszerű, de megválasztható színkódolással kell bemutatni (például de nem kötelezően: alacsony zöld, magasabb világosbarna, magas sötétbarna). A színkódoláshoz a magasságot 50 méterenként kell kiértékelni, a mintapontok között a szín lineárisan változik.
A tervező a bal egérgomb lenyomásokkal a terepre vetíti a pálya kontrolpontjait, amelyet a program fehér zárt Lagrange interpolációs görbével köt össze és printf-fel kiírja az aktuális pálya hosszát. SPACE lenyomására egy alkalmas színű nyílszerű konkáv poligonnal ábrázolt virtuális biciklista indul el a pályán a nyilat mindig a haladási irányba állítva. A haladási irány 3D-ben analitikusan (nem pedig közelítő differenciahányadossal) számítandó. A biciklista a két kontrolpont között pontosan annyi időt tölt el, amennyi a két gomblenyomás között eltelt. A pálya meredekségét a képernyőn tetszőleges helyen elhelyezett derékszögű háromszöggel szemléltesse, amelynek az emelkedési szög az egyik szöge, és állása mutatja, hogy emelkedővel küzd-e a kerékpáros vagy lejtőn gurul lefelé.

1. házi: Olimpiai hegyi kerékpárpálya



Házi keret könyvtárakkal