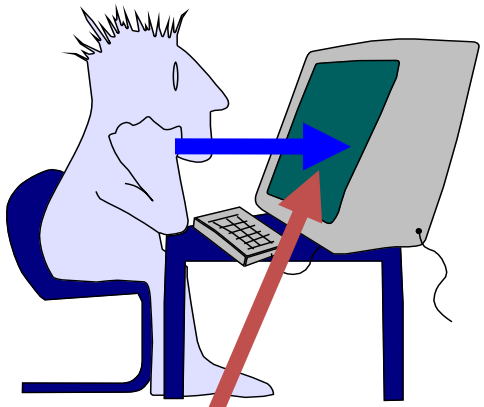
”In theory, there is no difference between theory and practice. In practice, there is.”

Benjamin Brewster

Grafikus hardver/szoftver alapok
1. Építőelemek

Szirmay-Kalos László

Interaktív rendszer: Funkcionális modell



Éppen belemerül
a virtuális világba

Bemeneti csővezeték

Eszköz-
képtér

Inv.
Viewport
transzf

Inv.
Proj.
transzf

Inv.
View
transzf

Inv.
Modell
transzf.

CPU: op

Fejre
állítva!

kamera

Virtuális világ

vektorizáció

CPU: app

Kép
frissités

fix hw

rasztertár

Pixel
proc

Raszteri-
záció

Viewport
transzf

Vágás

Proj.
transzf

View
transzf

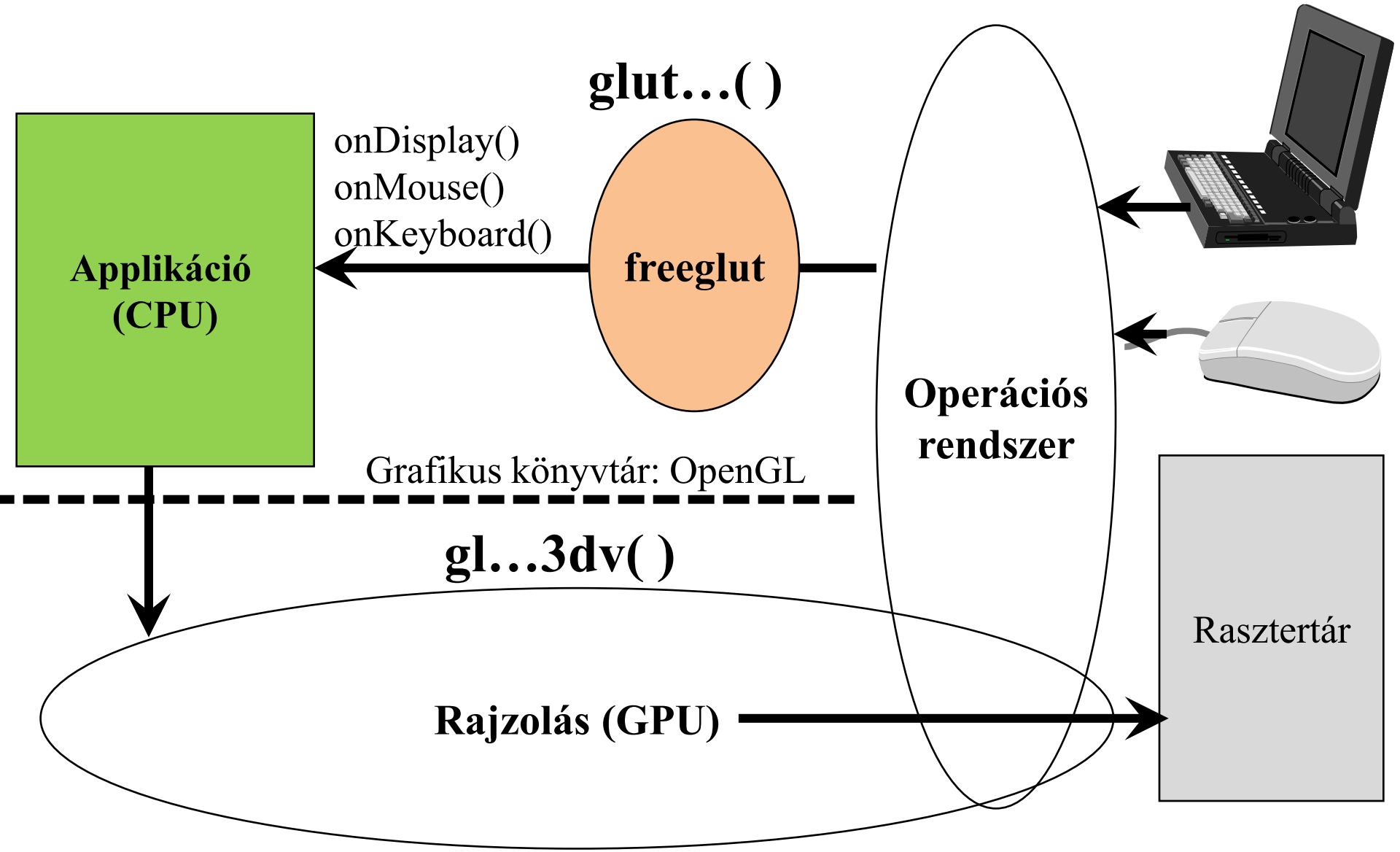
Modell
transzf.

pixel shader+blending

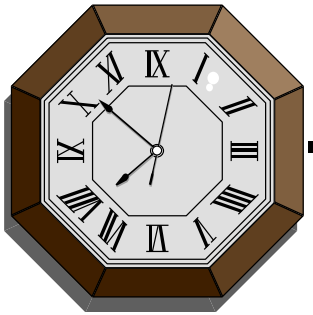
vertex shader

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

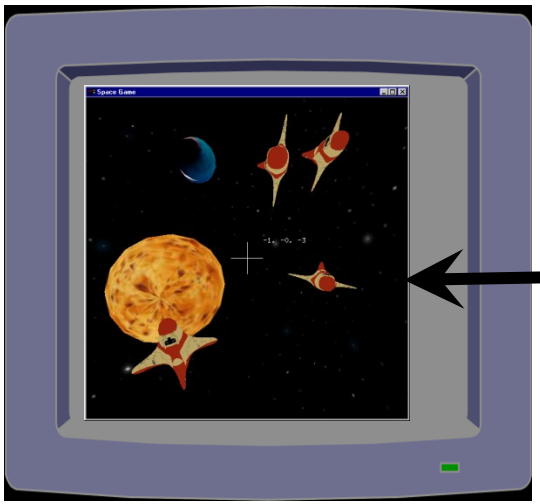
MotionFunc

IdleFunc

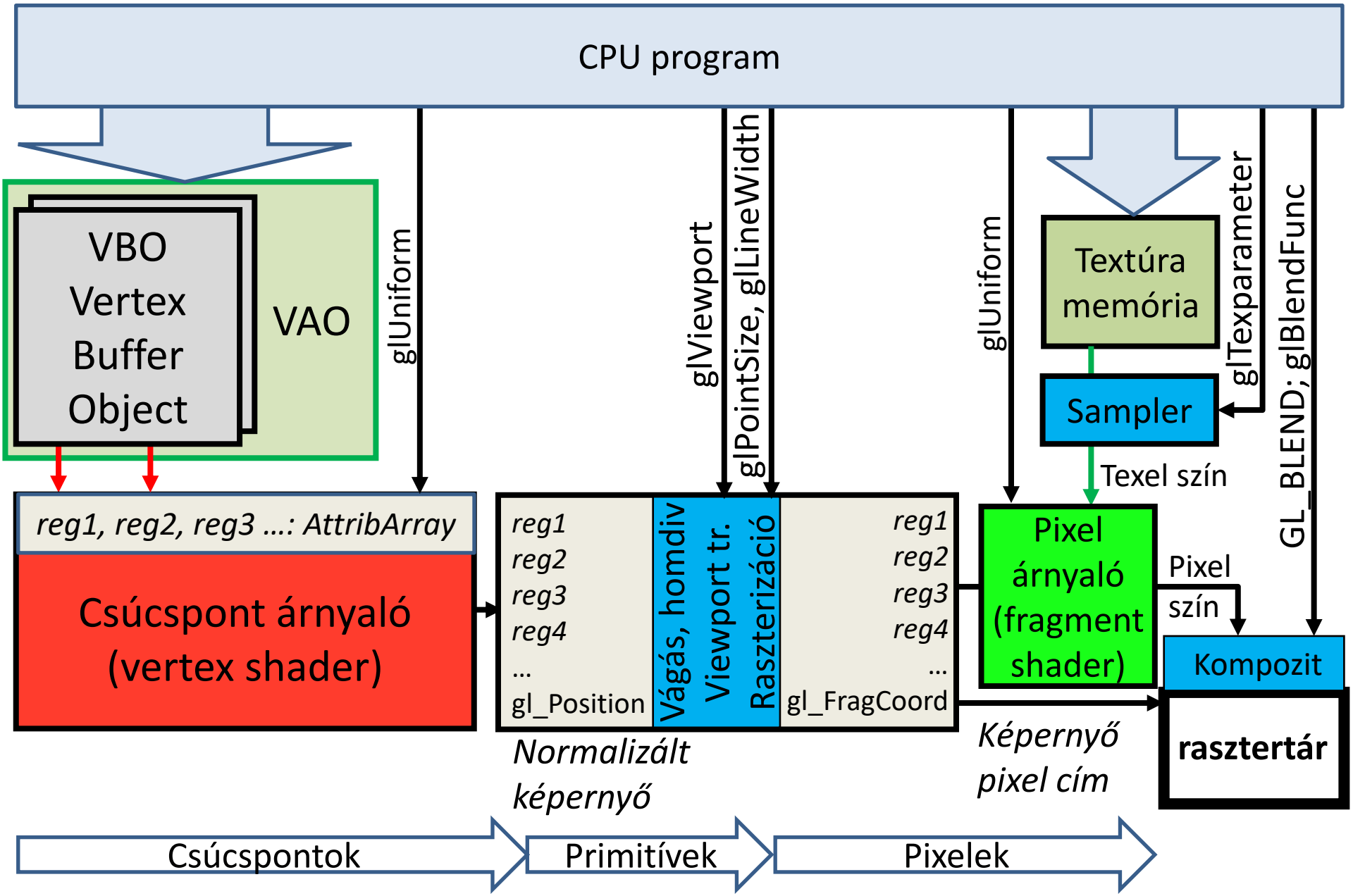
applikáció

OpenGL

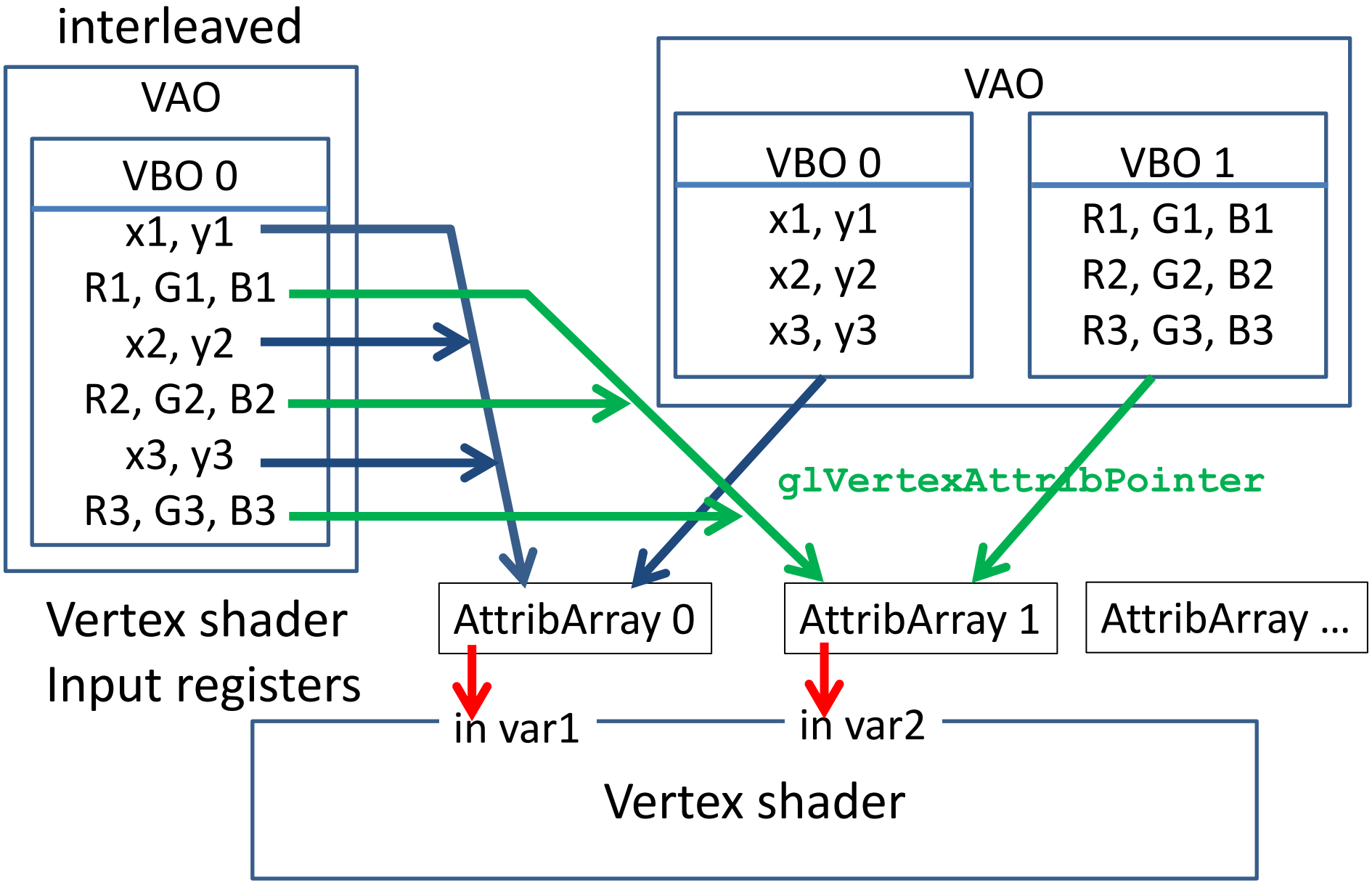
grafikus hardver



OpenGL 3.3 … 4.6 (Modern OpenGL)



Csúcspont adatfolyamok



Rajzolás: glDrawArrays
OpenGL primitívek

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

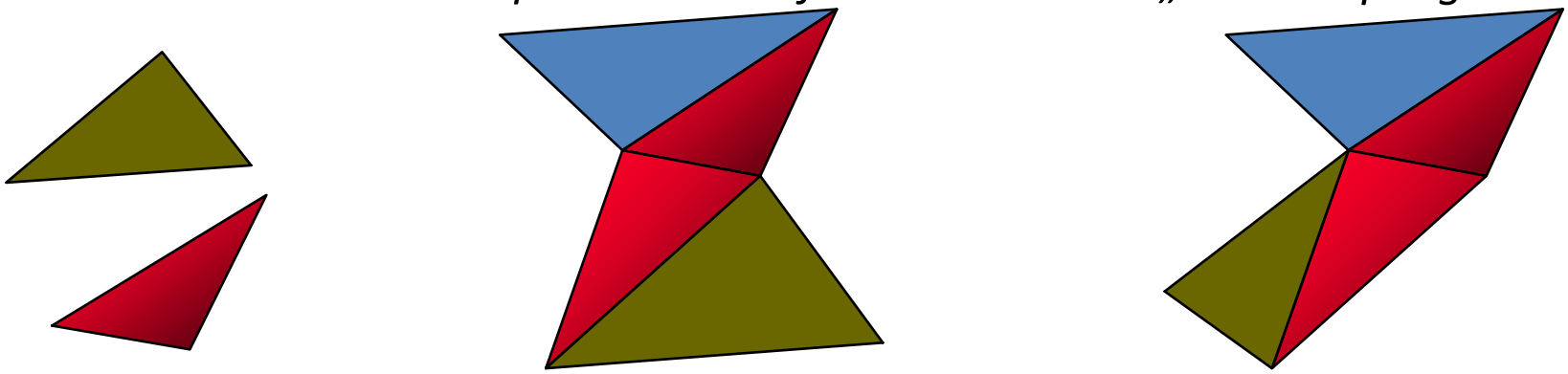
Vektorizált parametrikus görbe

GL_POINTS GL_LINES GL_LINE_STRIP GL_LINE_LOOP

Fülvágó kimenete

Tesszellált 3D parametrikus felület

„Konvex” poligon



GL_TRIANGLES GL_TRIANGLE_STRIP GL_TRIANGLE_FAN

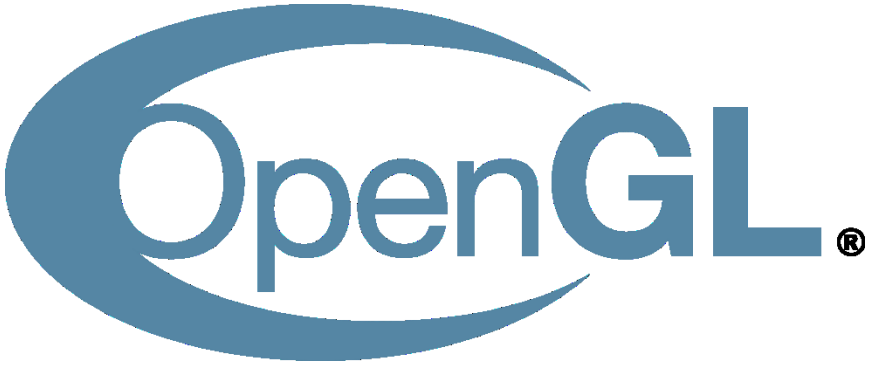
OpenGL állapotgép

Gagyi grafikus könyvtár

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

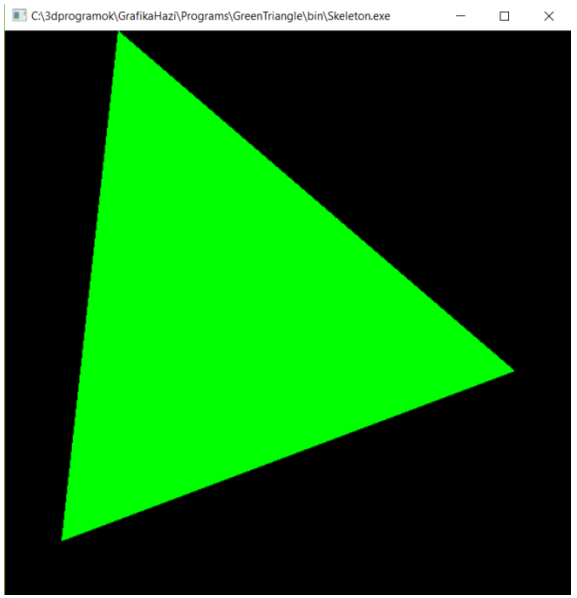
OpenGL

```
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!!!
```



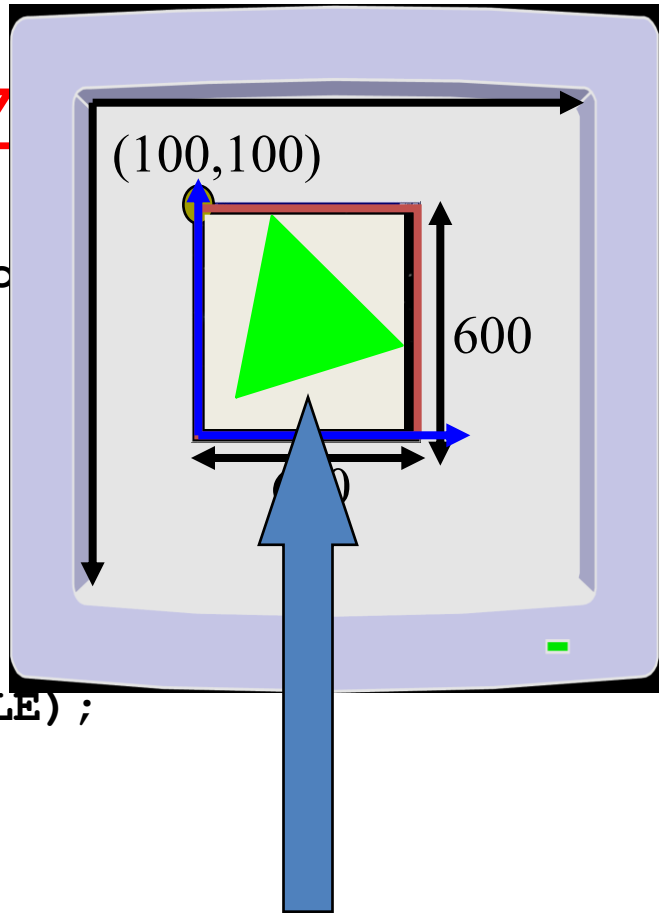
Grafikus hardver/szoftver alapok
2. Helló OpenGL/GLSL/GLUT

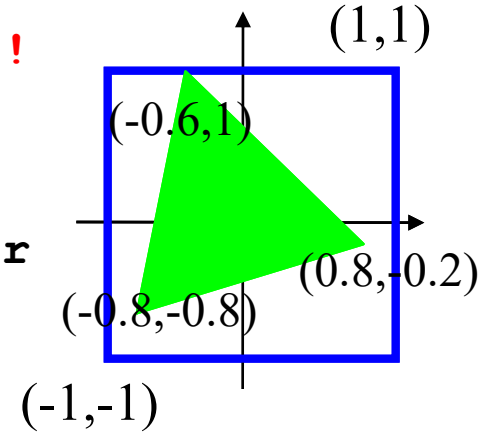
Szirmay-Kalos László



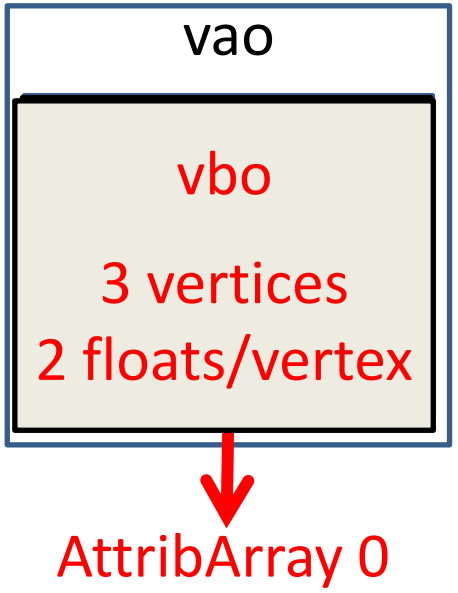
Az első OpenGL programom: Z

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

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

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

    glutDisplayFunc(onDisplay); //event handler

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

C++11

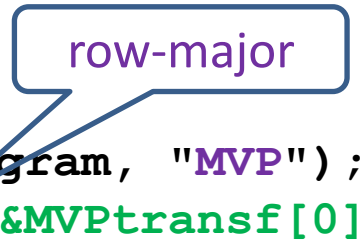
```
#version 330
precision highp float;

uniform vec3 color;
out vec4 outColor;

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

```
static const char * vertSource = R"( … )";
// vagy:                       = FileToString("vertex.glsl");

static const char * fragSource = R"( … )";
unsigned int vertShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertShader, 1, (const GLchar**)&vertSource, NULL);
glCompileShader(vertShader);

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

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

glBindFragDataLocation(shaderProgram, 0, "outColor");

glLinkProgram(shaderProgram);
glUseProgram(shaderProgram); // make it active
}
```

```
#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);
}
```

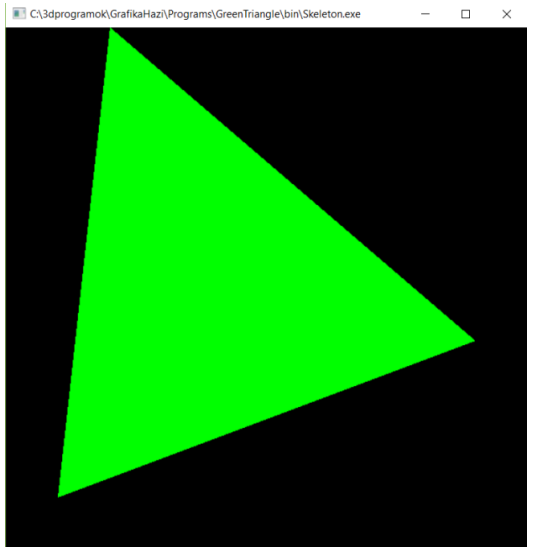
```
void onDisplay( ) {
    glClearColor(0, 0, 0, 0);      // background color
    glClear(GL_COLOR_BUFFER_BIT); // clear frame buffer

    // 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
}
```



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 {
    …
    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]();`
- `glBind[VertexArray|Buffer|Texture]();`
- `glBufferData();`

Bufferek vertex shader bemeneti regiszterekhez kötése

- `glEnableVertexAttribArray()` regiszter engedély
- `glVertexAttribPointer()` bufferből mely regiszterbe

Rajzolás és csővezeték management

- `glDrawArrays()` vao bufferek felrajzolása
- `glClearColor()` háttér törlési szín
- `glClear()` háttér törlés
- `glViewport()` fénykép méret
- `glPointSize()` pont méret
- `glLineWidth()` 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;
struct vec3;
struct vec4;
struct 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"

const char * const vertexSource;
const char * const fragmentSource;

GPUProgram gpuProgram; // vertex and fragment shaders

void onInitialization() {
    …
    gpuProgram.create(vertexSource, fragmentSource, "outColor");
}

void onDisplay() {
    glClearColor(0, 0, 0, 0);     // background color
    glClear(GL_COLOR_BUFFER_BIT); // clear frame buffer
    …
    glutSwapBuffers(); // exchange buffers for double buffering
}

void onKeyboard(unsigned char key, int pX, int pY) { … }
void onKeyboardUp(unsigned char key, int pX, int pY) { … }
void onMouseMotion(int pX, int pY) { … }
void onMouse(int button, int state, int pX, int pY) { … }
void onIdle() { … }
```

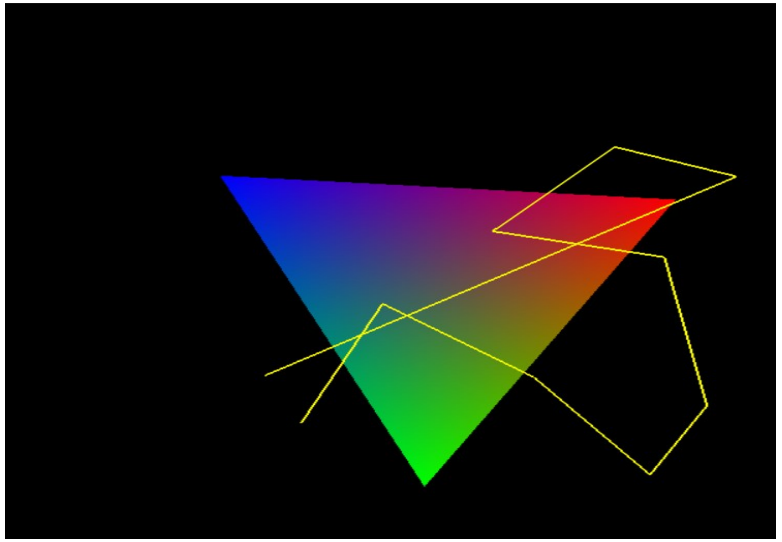
Grafikus hardver/szoftver alapok

Program: Vasarely festmény

Szirmay-Kalos László

Grafikus hardver/szoftver alapok
Program: Animáció és interakció

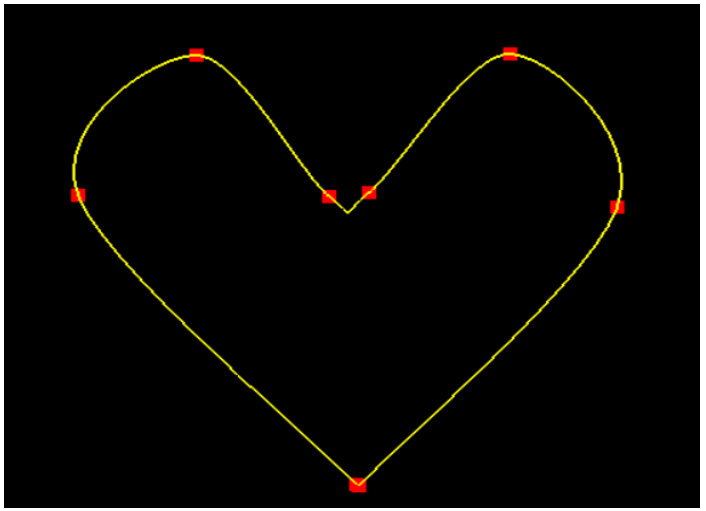
Szirmay-Kalos László



Grafikus hardver/szoftver alapok

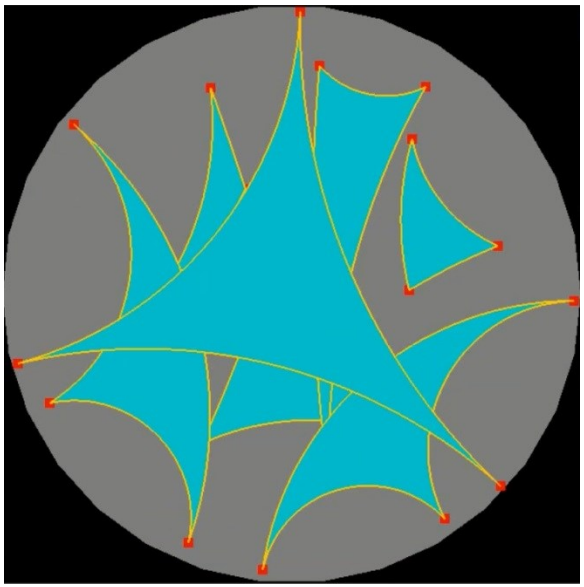
Program: Görbeszerkesztő

Szirmay-Kalos László



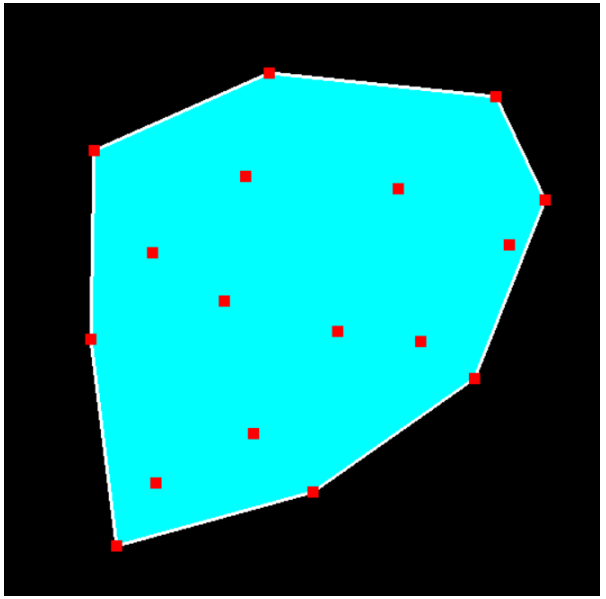
Grafikus hardver/szoftver alapok
Program: Hiperbolikus háromszögek

Szirmay-Kalos László



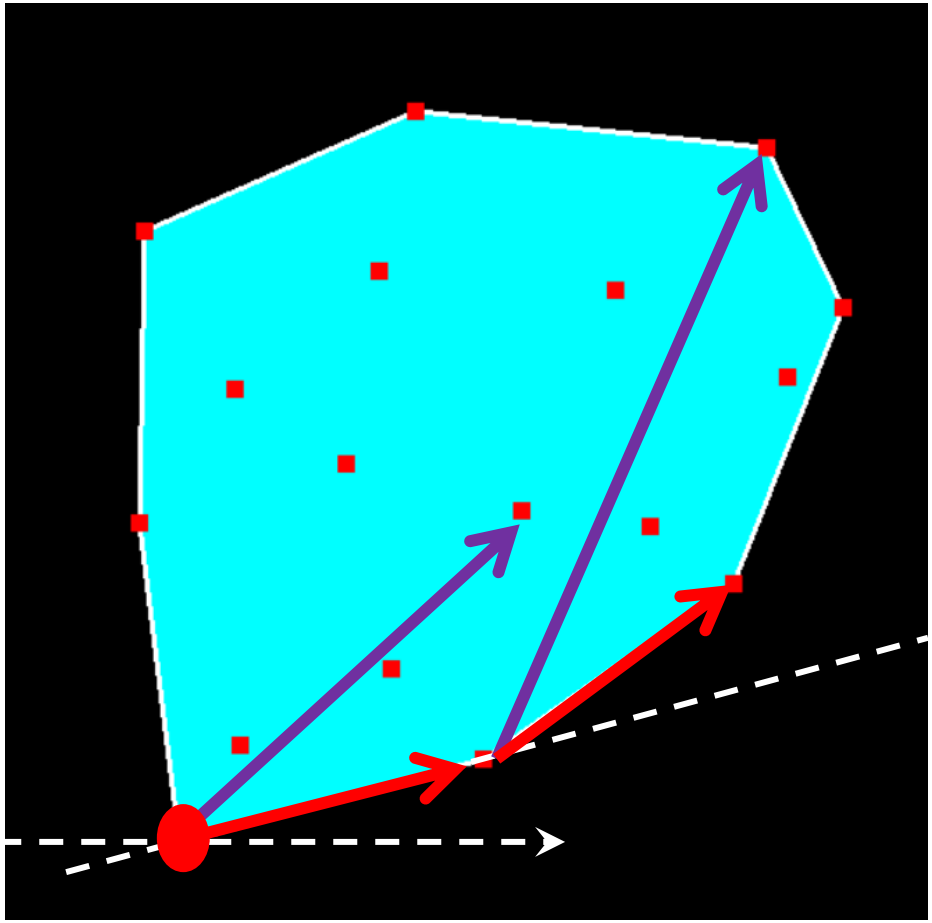
Grafikus hardver/szoftver alapok
Program: Konvex burok, interakció

Szirmay-Kalos László



Konvex burok

- Minimális konvex ponthalmaz, ami az adott pontokat tartalmazza



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);
}
```

Fragment shader:

```
uniform vec3 color;
out vec4 fragmentColor;

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

Object

```
struct Object {
   unsigned int vao, vbo;
   std::vector<vec2> vtx;

   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);
   }
   void updateGPU() {
      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 (vertices.size() > 0) {
         glBindVertexArray(vao);
         gpuProgram.setUniform(color, "color");
         glDrawArrays(type, 0, vertices.size());
      }
   }
};
```

Convex hull

```
class ConvexHull {
   Object p, h; // points and hull
public:
   void addPoint(vec2 pp) { p.vtx.push_back(pp); }
   void update() {
      if (p.vtx.size() >= 3) findHull();
      p.updateGPU();
      h.updateGPU();
   }
   vec2 * pickPoint(vec2 pp) {
      for (auto& v : p.vtx) if (length(pp-v) < 0.05f) return &v;
      return nullptr;
   }
   void findHull();
   void Draw() {
      h.Draw(GL_TRIANGLE_FAN, vec3(0, 1, 1));
      h.Draw(GL_LINE_LOOP, vec3(1, 1, 1));
      p.Draw(GL_POINTS, vec3(1, 0, 0));
   }
};
```

Convex hull előállítás

```
void ConvexHull::findHull() {
    h.vtx.clear();
    vec2 * vStart = &p.vtx[0]; // Find lowest point
    for (auto& v : p.vtx) if (v.y < vStart->y) vStart = &v;

    vec2 vCur = *vStart, dir(1, 0), *vNext;
    do { // find convex hull points one by one
        float maxCos = -1;
        for (auto& v : p.vtx) { // find minimal left turn
            float len = length(v - vCur);
            if (len > 0) {
                float cosPhi = dot(dir, v - vCur) / len;
                if (cosPhi > maxCos) { maxCos = cosPhi; vNext = &v;}
            }
        }
        h.vtx.push_back(*vNext); // save as convex hull
        dir = normalize(*vNext - vCur); // prepare for next
        vCur = *vNext;
    } while (vStart != vNext);
}
```

Virtuális világ és megjelenítése

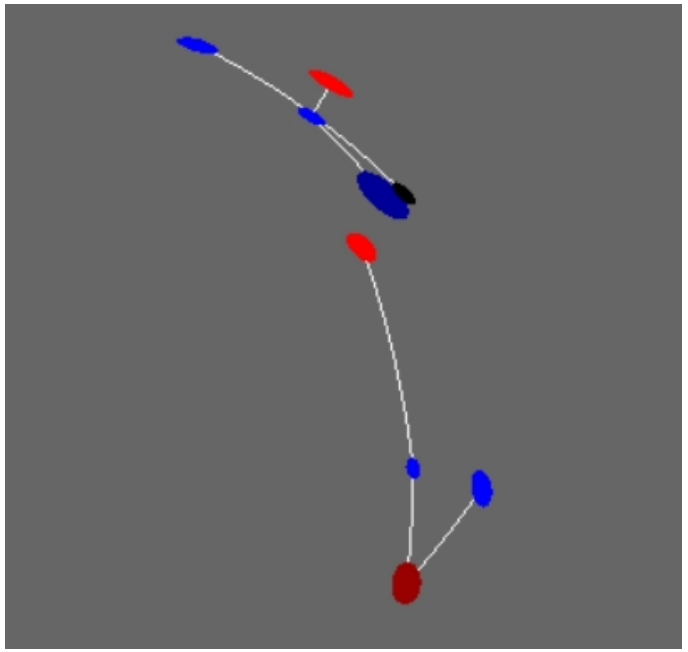
```
ConvexHull * hull;
vec2 * pickedPoint = nullptr;

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

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

Controller

```
vec2 PixelToNDC(int pX, int pY) { // if full viewport
   return vec2(2.0f * pX / windowWidth - 1; // flip y axis
               1.0f - 2.0f * pY / windowHeight);
}

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

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

Grafikus hardver/szoftver alapok
Program: Molekula dokkolás

Szirmay-Kalos László



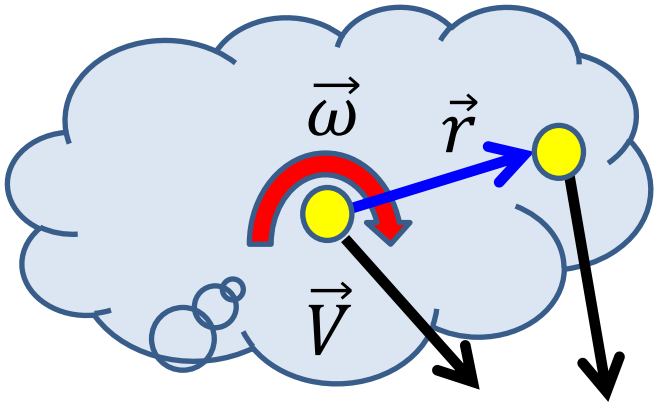
Feladatleírás

Készítsen molekula dokkoló alkalmazást. SPACE hatására mindig két molekula születik, amelyek atomjai között Coulomb erő keletkezik, amely a molekulákat mozgatja, illetve forgatja. A molekulák atomjaira a sebességgel arányos közegellenállás érvényesül. Egy molekula atomok merev, véletlen fagráf topológiájú szerkezete. Az atomok száma 2 és 8 közötti véletlen szám. Az alkotó atomok tömege a Hidrogén atom tömegének, töltése pedig az elektron töltésének véletlen pozitív egészszámszorosa. Az össztöltés minden molekulára zérus. A molekulák a 2D euklideszi térben mozognak, az atomok itt kör alakúak, az atomon belüli fagráf élei fehérek és az euklideszi geometriában szakaszok. A pozitív töltésű atomok piros, a negatívak kék árnyalatúak, az intenzitás a töltéssel arányos. A mikroszkópunk az euklideszi síkot a hiperbolikus síkra képezi le az x, y koordináták megőrzésével, majd a Beltrami-Poincaré leképzéssel jeleníti meg a 600x600 felbontású képernyőre rajzolható maximális sugarú körben. Az s,d,x,e billentyűkkel az euklideszi virtuális világot balra, jobbra, lefelé és felfelé lehet eltolni 0.1 egységgel. Az időlépés nagysága 0.01 sec lehet a rajzolás sebességétől függetlenül.

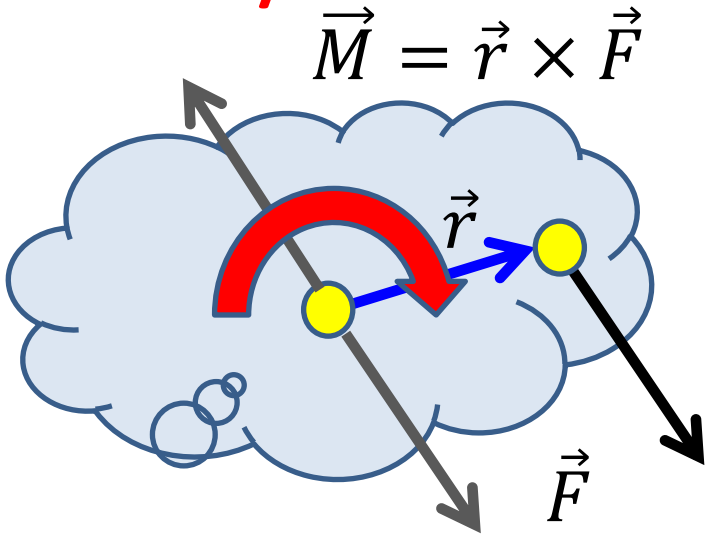
Molekula

- Konstruálás:
  - Atomok és kötések referenciahelyzetben
  - Fagráf: atom és kötés együttes felvétele
  - Súlypont számolás
  - Atomok eltolása, hogy a súlypont az origó legyen
  - Tehetetlenségi nyomaték számítása
- Rajzolás
  - Modellezési transzf: pozíció $\boldsymbol{r}_i$, forgatási szög α_i
  - Atom: kitöltött kör
  - Kötés: finoman vektorizált szakasz

2D merevtest kinematika/dinamika



$$\vec{v} = \vec{V} + \vec{\omega} \times \vec{r}$$



$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{\sum \vec{F}}{\sum m}$$

$$\frac{\mathrm{d}\omega}{\mathrm{d}t} = \frac{\sum M}{\theta}$$

$$\theta = \sum m(\vec{r})^2$$

Erők:

- 2D Coulomb: $\vec{F} = \frac{q_1 q_2}{2\pi\varepsilon d}\overrightarrow{e_{21}}$
- Közegellenállás: $\vec{F} = -\rho\vec{v}$

Dinamikai szimuláció

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{\sum \vec{F}}{\sum m}$$

$$\frac{\mathrm{d}\omega}{\mathrm{d}t} = \frac{\sum M}{\theta}$$

State: $\boldsymbol{r}_i, \boldsymbol{v}_i, \alpha_i, \omega_i$

for(t = 0; t < T; t += dt) { // onIdle

for each node i {

$$\sum \vec{F} = \ldots$$

$$\sum M = \cdots$$

$$\boldsymbol{v}_i \mathrel{+}= \sum \vec{F} / m \cdot \mathrm{d}t$$

$$\boldsymbol{r}_i \mathrel{+}= \boldsymbol{v}_i \cdot \mathrm{d}t$$

$$\omega_i \mathrel{+}= \sum M / \theta \cdot \mathrm{d}t$$

$$\alpha_i \mathrel{+}= \omega_i \cdot \mathrm{d}t$$

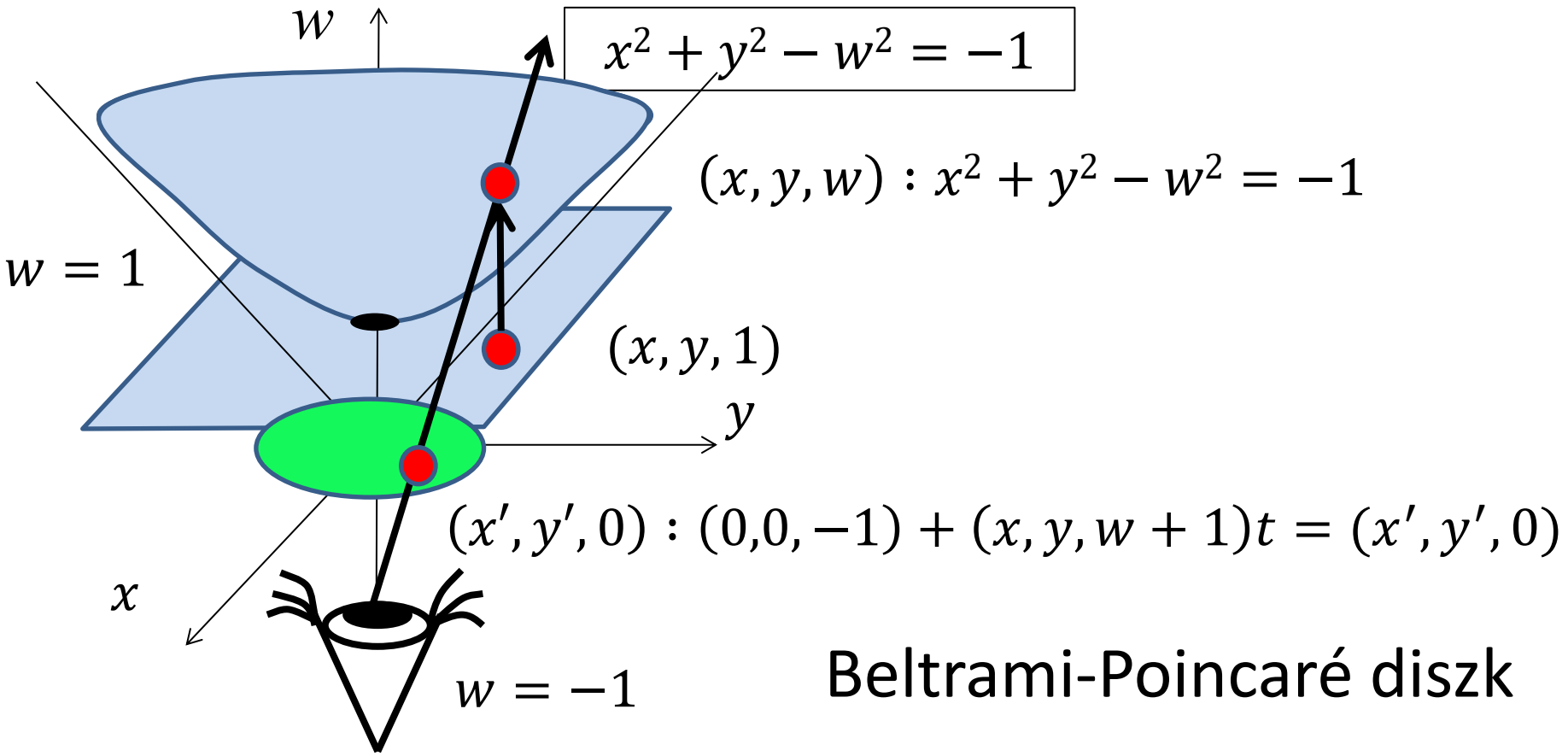
}

}

Nézeti V és P transzformáció (vertex shader)

V: eltolás, amely a kamerát a (0,0,1)-be viszi

P: a teljes síkot az origó középpontú egységsugarú körbe viszi

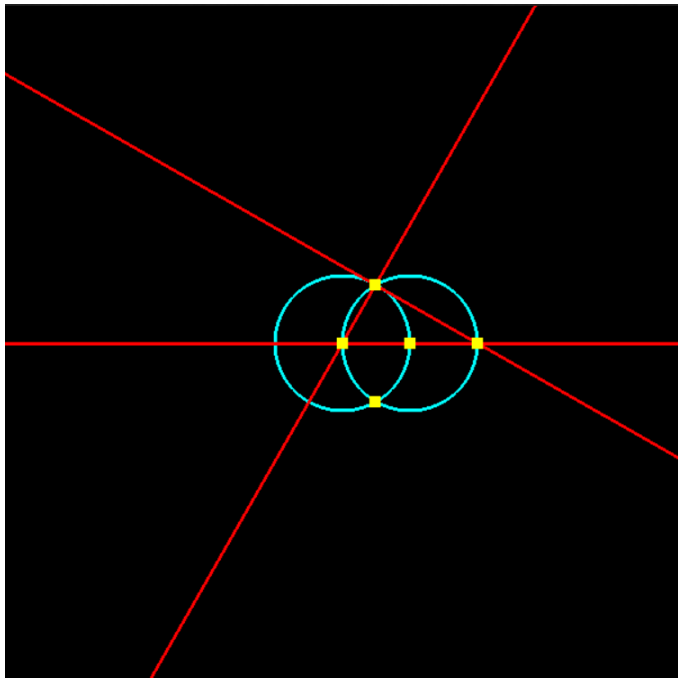


Beltrami-Poincaré diszk

Grafikus hardver/szoftver alapok

Program: Körző és vonalzó

Szirmay-Kalos László



Világ koordináták

- Origó, tengelyek, egység
- Modellezési transzformáció?
- Kamera ablak
- Kamera transzformáció: csúcspont árnyaló

Modell

- Pontok, egyenesek, körök
  - Heterogén vagy három homogén kollekció?
  - Prioritás: kör < egyenes < pont
- Felvétel: pont, egyenes, kör: kollekció építés
- Kiválasztás:
  - Csökkenő prioritásos kollekció bejárás és pont, egyenes, kör delegálás
- Metszés: egyenes-egyenes, egyenes-kör, kör-kör
  - Implicit – Implicit, Implicit – Parametrikus, Parametrikus – Parametrikus
  - Kinek a felelőssége (Modell, egyenes, kör)?
- Felrajzolás:
  - Növekvő prioritásos kollekció bejárás és pont, egyenes, kör delegálás

Pont, egyenes, kör

- Mivel reprezentálunk
  - implicit vagy parametrikus egyenlet paraméterei
- Attribútumok: szín és állapot
- Kiválasztás:
  - Pont – primitív távolság (melyik egyenlettípus?)
- CPU – GPU szinkronizálás: Hány VAO/VBO?
  - Mikor frissítjük?: ha változik, vagy felrajzolás előtt
  - Szín uniform paraméter vagy csúcspont attribútum?
  - Vektorizáció: GL_LINES, GL_LINE_LOOP, GL_POINTS
  - Hány vektorizált kör? Modellezési transzformáció?

Kontroller: forgatókönyvek

Körző befogása:

's'	Művelet = körző befogás, 1. pont jön
klikk egy létező pont1-re	pont1 fehér Nem 1. pont jön
klikk egy létező pont2-re	Sugár = |pont2 – pont1|, point1 eredeti szín

Kör rajzolás a befogott sugárral:

'c',	Művelet = kör rajzolás, 1. pont jön
klikk egy létező pont1-re	kör felvétel: középpont = pont1, kör sugár = Sugár

Egyenes rajzolás:

'l'	Művelet = egyenes rajzolás, 1. pont jön
klikk egy létező pont1-re	pont1 fehér Nem 1. pont jön
klikk egy létező pont2-re	egyenes felvétel: pont1, pont2; point1 eredeti szín

Metszéspont:

'i'	Művelet = metszéspont, 1. primitív jön
klikk egy egyenesre vagy körre	primitív1 fehér Nem 1. primitív jön
klikk egy egyenesre vagy körre	primitív2, primitív1 eredeti szín
	metszéspontok számítása és felvétele

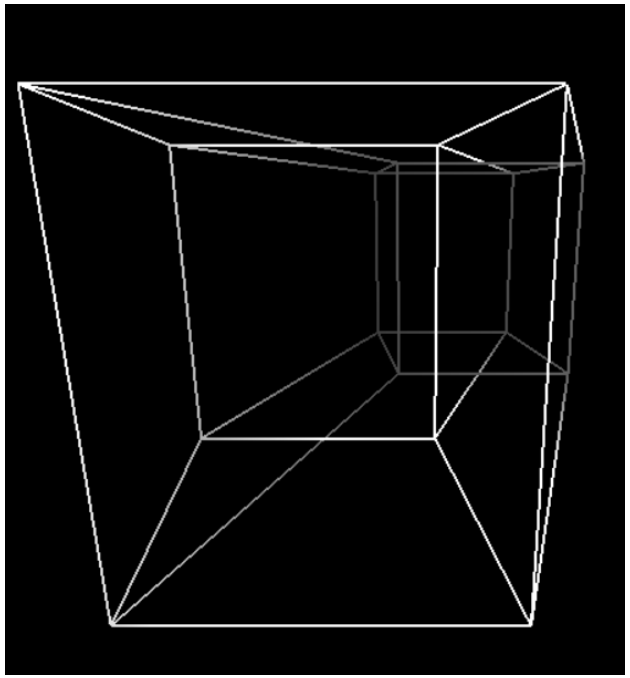
Események szerinti átrendezés

```
vec2 pont1;
enum {…} művelet;
bool első_pont_jön;
…
void onKeyboard(unsigned char key, int pX, int pY) {
   switch (key) {
   case 's': …
   }
}

void onMouse(int button, int button_state, int pX, int pY) {
   if (button==GLUT_LEFT_BUTTON && button_state==GLUT_DOWN) {
      vec2 pickPoint = InputPipelineTranszformáció (pX, pY);
      switch (művelet) {
      case …
      case …
      }
      glutPostRedisplay();      // redraw
   }
}
```

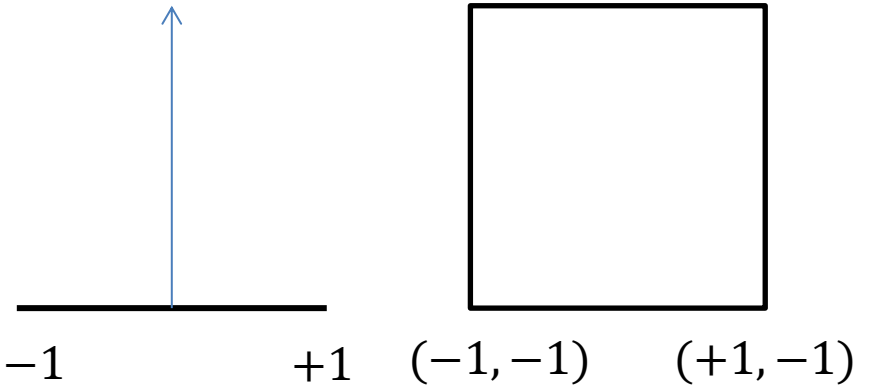
Grafikus hardver/szoftver alapok

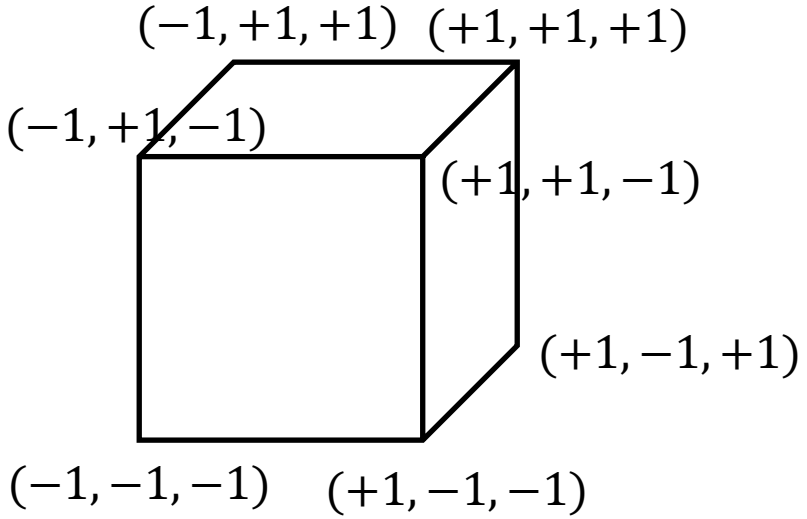
5. Program: Tesseract (4D kocka), animáció

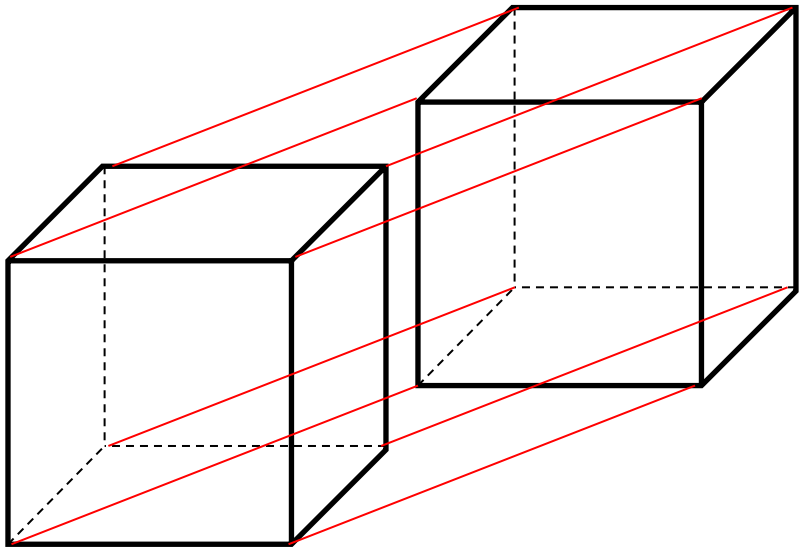
Szirmay-Kalos László



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, "fragmentColor");
}

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;  // vertex array object id
    vector<float> vtx;      // vertices of the object
    mat4 Rotate;            // Transformation matrix

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

Élek a GPU-ra

```
Tesseract::Tesseract() {
   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) != 0 ? 1 : -1);
            for (int b = 1; b < maxcode; b <<= 1) {
               vtx.push_back(((code+bit) & b) != 0 ? 1 : -1);
         }
      }
   }
   glGenVertexArrays(1, &vao); glBindVertexArray(vao);
   glGenBuffers(1, &vbo);
   glBindBuffer(GL_ARRAY_BUFFER, vbo);
   glBufferData(GL_ARRAY_BUFFER, vtx.size()*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) {
    Rotate = mat4(1, 0, 0, 0,
                  0, 1, 0, 0,
                  0, 0, cos(t), sin(t),
                  0, 0,-sin(t), cos(t));
    gpuProgram.setUniform(Rotate, "R");
}

void Tesseract:: Draw() {
    glBindVertexArray(vao);
    glDrawArrays(GL_LINES, 0, vtx.size() / 4);
}
```

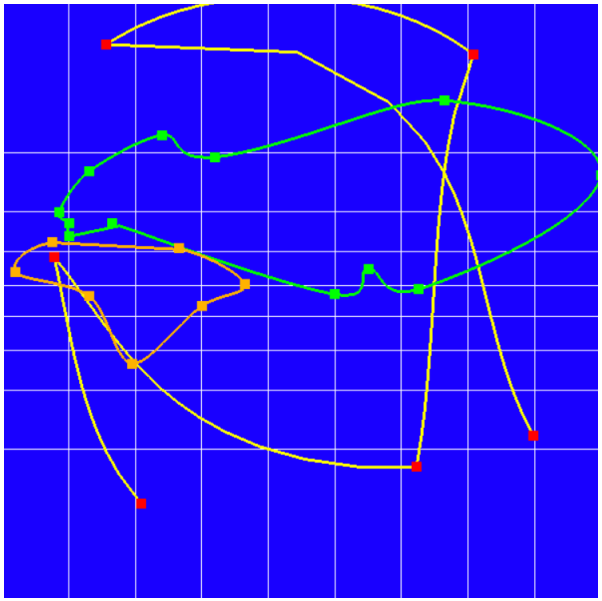
Vertex és fragment árnyalók

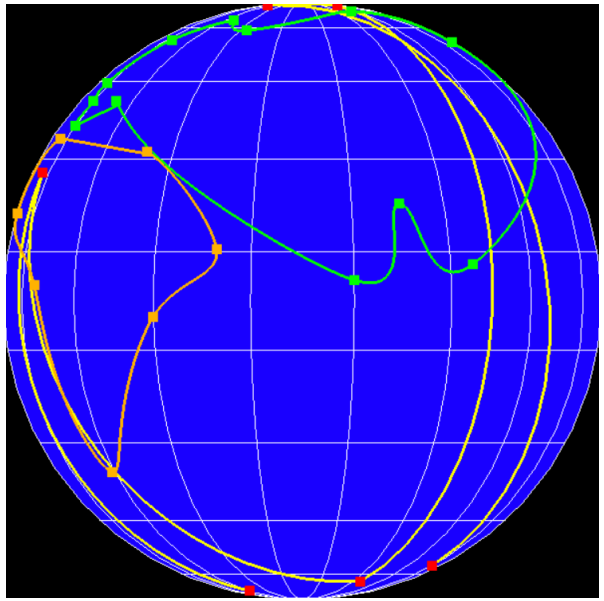
```
const float size = 0.3f, distBias = 0.4f;
const vec4 location = vec4(0, 0, 1, 1);
uniform mat4 R;

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

void main() {
    vec4 p4d = vertex * R * size + location;
    depth = 1 / (dot(p4d, p4d) - distBias);
    vec3 p3d = p4d.xyz / p4d.w;
    vec2 p2d = p3d.xy / p3d.z;
    gl_Position = vec4(p2d, 0, 1);
}
```

```
in float depthCue;
out vec4 fragColor;

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

Grafikus hardver/szoftver alapok
Program: Mercator Útvonaltervező

Szirmay-Kalos László





Specifikáció

Készítsen útvonaltervezőt. A program `m'-mel választhatóan két képet mutat a föld $(-85^o, +85^o)$ szélesség közötti és $(-20^o, 160^o)$ hosszúság közötti részéről. Az egyik a Mercator térkép, a másik merőleges vetület a lehető legnagyobb méretben. A képen a tengert, Eurázsia és Afrika határát rajzoljuk fel, a többi földrésztől eltekintünk. **A tenger színe a 430 nm hullámhosszú, Eurázsia színe 530 nm, Afrikáé pedig 560 nm-es monokromatikus fénytől megkülönböztethetetlen, feltételezve, hogy a monitor pixelek 444, 526 és 645 nm-en sugároznak.** A hosszúsági és szélességi köröket 20 fokonként vékony fehér vonallal jelöljük. Eurázsiát és Afrikát szélesség-hosszúság koordinátájú kontrolpontokkal adjuk meg, amelyekre **egy-egy O-spline-t kell illeszteni**.

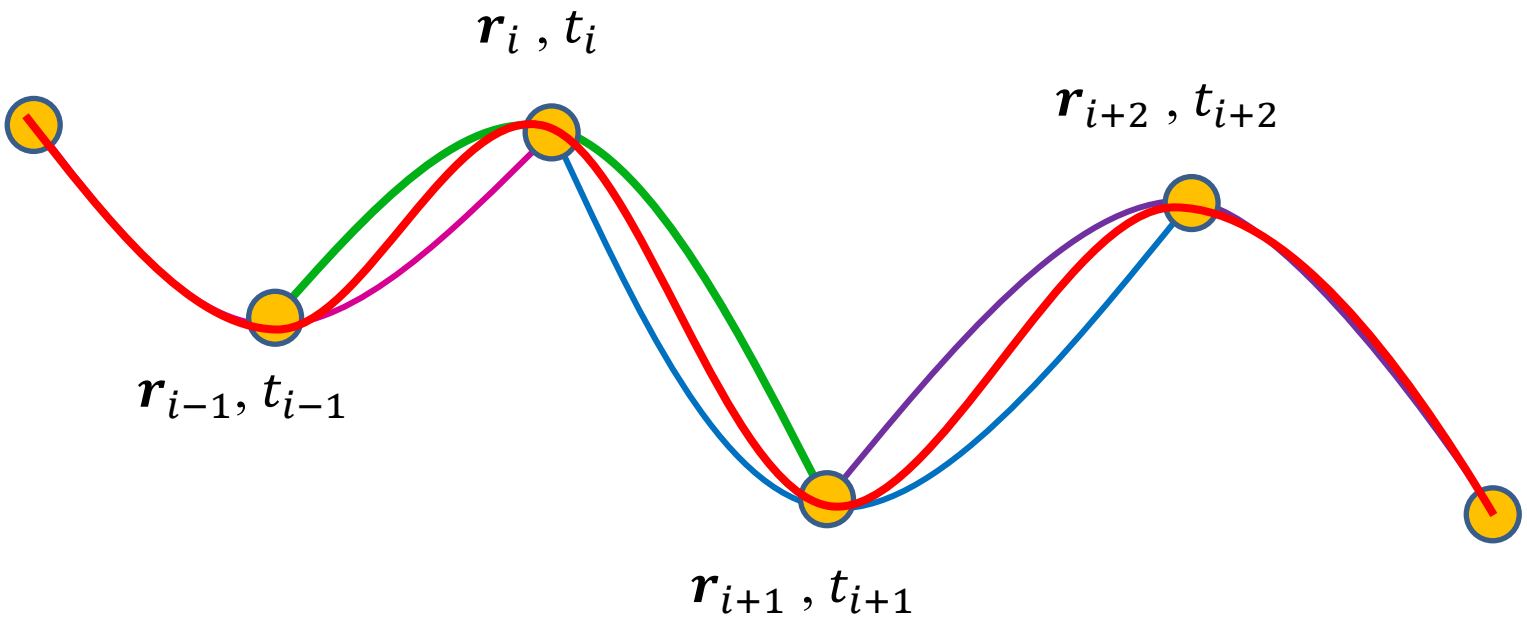
Az útvonal állomásait az egér bal gomb lenyomásával bármely(!) nézetben kijelölhetjük. Az állomásokat a rendszer a gömbön sárga legrövidebb utakkal köti össze, és kiírja printf-fel az állomás szélesség-hosszúság koordinátáit és az utolsó állomástól mért hosszát (a föld sugara 6371 km).

A feladatot 2D-s grafika szabályai szerint kell megoldani, ezért csak olyan OpenGL funkciók használhatók, amelyek az „OpenGL és GPU programozás" végéig megjelentek.

Eurázsia kontrollpontjai: (36, 0), (42, 0), (47, -3), (61, 6), (70, 28), (65, 44), (76, 113), (60, 160), (7, 105), (19, 90), (4, 80), (42, 13)

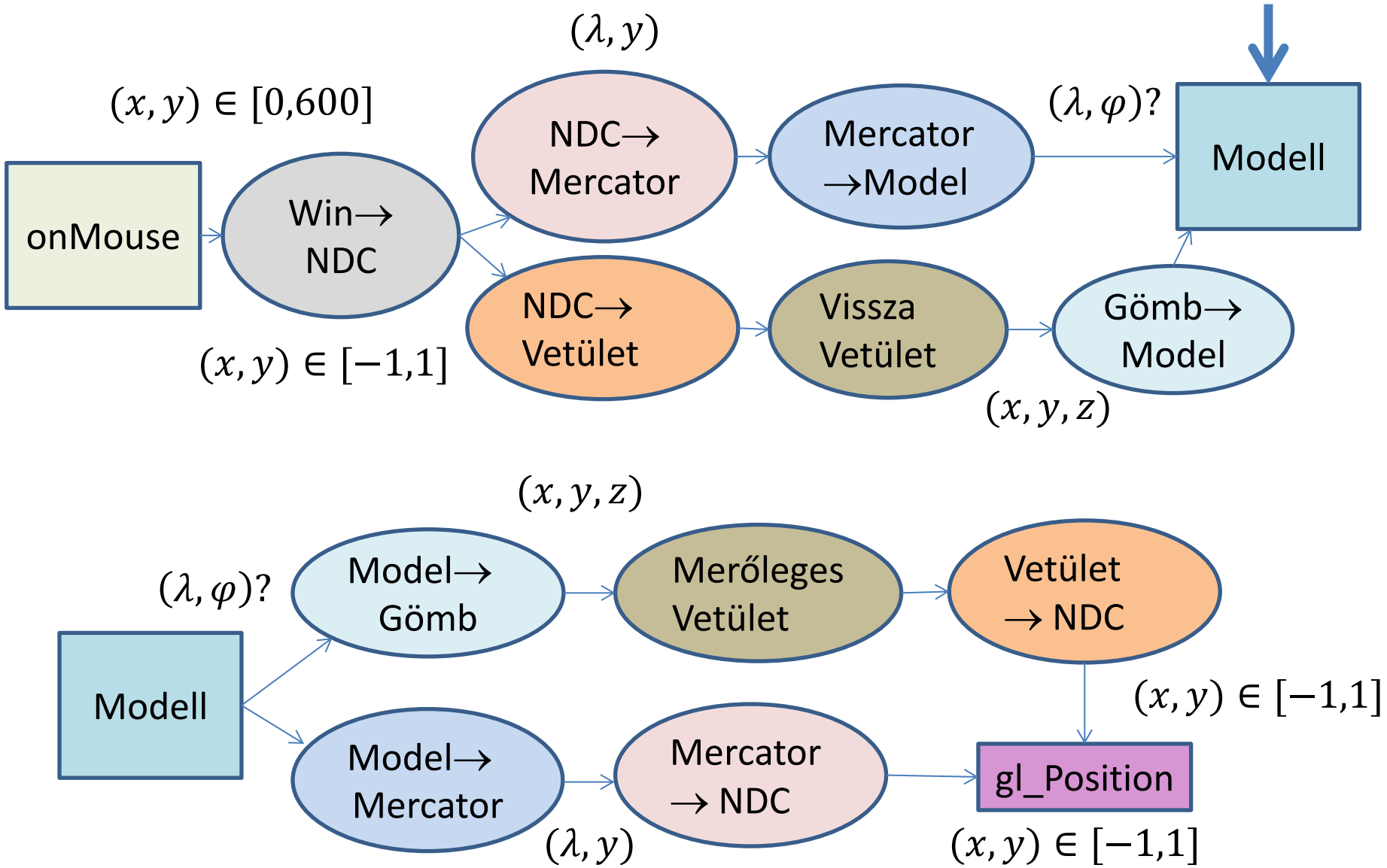
Afrika kontrollpontjai: (33, -5), (17, -16), (3, 6), (-35, 19), (-3, 40), (10, 53), (30, 33)

O-spline



- Másodfokú szegmens: $\boldsymbol{s}_i(t) = \boldsymbol{a}_i(t - t_i)^2 + \boldsymbol{b}_i(t - t_i) + \boldsymbol{c}_i$
- Összemosás: $\boldsymbol{r}(t) = (\boldsymbol{s}_i(t)(t_{i+1} - t) + \boldsymbol{s}_{i+1}(t)(t - t_i))/(t_{i+1} - t_i)$
- Ciklikus: A 0 előtt a legutolsó áll, a legutolsó után a 0.

Modell koordinátarendszer?



Objektumok

- Shader(ek)
  - Vertex shader: kimeneti csővezeték transzformációk
  - Pixel shader: konstans színű rajzolás
  - Szín, transzformációs paraméterek beállítása
- Geometry
  - Load
  - Draw
- Earth (vektorizált téglalap)
- Circle (vektorizált szakasz)
- Continent (vektorizált spline + kontrollpontok)
- Path (vektorizált gömbi geodézikus + kontrollpontok)
  - Bemeneti csővezeték transzformációk

Feladatok

- Kontrollpontok alapján vektorizált O-spline-okból VAO/VBO gyártása
- Csúcspontárnyalók a Mercator és a merőleges gömbi vetítéshez
- Inverz transzformációk a bemeneti csővezetéken.
- Színek (lásd „Színérzékelés: monokromatikus fény” diát).