

Számítógépes grafika és képfeldolgozás

Szirmay-Kalos László

Irányítástechnika és Informatika Tanszék

email: szirmay@iit.bme.hu

Web: <http://cg.iit.bme.hu>

<http://cg.iit.bme.hu>





Computer Graphics Group

Department of Control Engineering
and Information Technology

■ Munkatársak ■ Publikációk ■ Projektek ■ Oktatott tárgyak ■ Elérhetőség

Számítógépes grafika és képfeldolgozás

- Számítógépes grafika és képfeldolgozás
 - Számítógépes vizualizáció
 - Játékfejlesztés
 - Grafikus játékok fejlesztése
 - GPU általános célú programozása (GPGPU)
 - 3D-s számítógépes geometria
 - Önálló laboratórium

Számítógépes grafika és képfeldolgozás

- Alapfogalmak
- Grafikus alap hw/sw
- Analitikus geometria
- Geometria modellezés
- Színek
- Geometria transzformációk
- 2D képsztízis
- 2D grafikus rendszerek
- 3D képsztízis optikai alapmodellje
- Sugárkövetés
- Inkrementális 3D képsztízis

Tárgyfelelős:
Dr. Szirmay-Kalos László
Tantárgy azonosítója:
VIII/A316

A tárgy a vizuális számítástechnika (számítógépes grafika, gépi látás, virtuális valóság, játékfejlesztés) alapjait mutatja be. Az elméletet (projektív geometria, szintan, analitikus geometria, radiometria, kinetika/dinamika) rögtön gyakorlatba ültetve vezetjük be, implementációs platformként a C++ nyelvet, az OpenGL könyvtárat és a Cg árnyalóprogramozási környezetet használva. A tárgyat hallgató hallgatók megismerkedhetnek a 2D és 3D grafikus rendszerek felépítésével, az animáció és játékfejlesztés technikáival, a grafikus kártyák működésével és programozásával. A házfeladatokat elkészítő hallgatók pedig felfrissíthetik C++ programozási készségeiket és igazán izgalmas alkalmazásokat készíthetnek (Ne állj ellen a kísértésnek, csak a saját programoddal játsszál!).

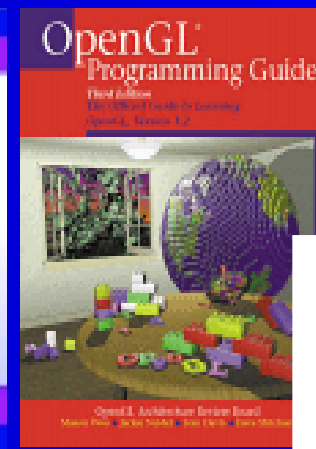
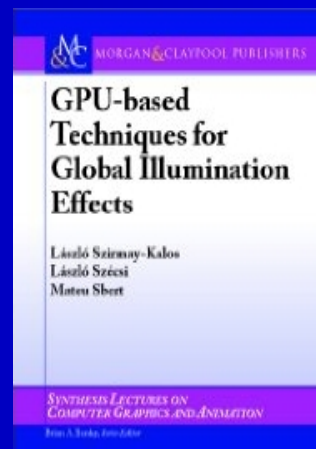
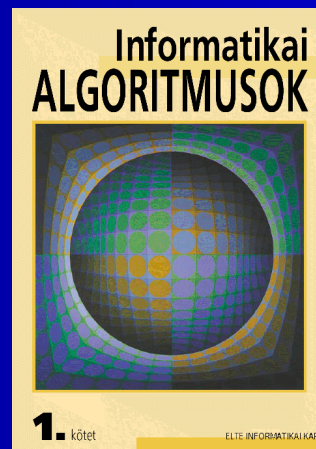
Házi feladatok, követelmények, megajánlott irodalom

2009. évi előadás videók

Feladatgyűjtemény a ZH és vizsgafelkészüléshez

Jegyzetek:

5 db kötelező kisházi
1 db önkéntes nagyházi
vizsga (Cg shader)



Juhász
Imre
OpenGL

Számítógépes grafika

illúzió



Képpontok:

- vörös
- kék
- zöld

modellezés

képszintézis

számok

Virtuális világ
modell

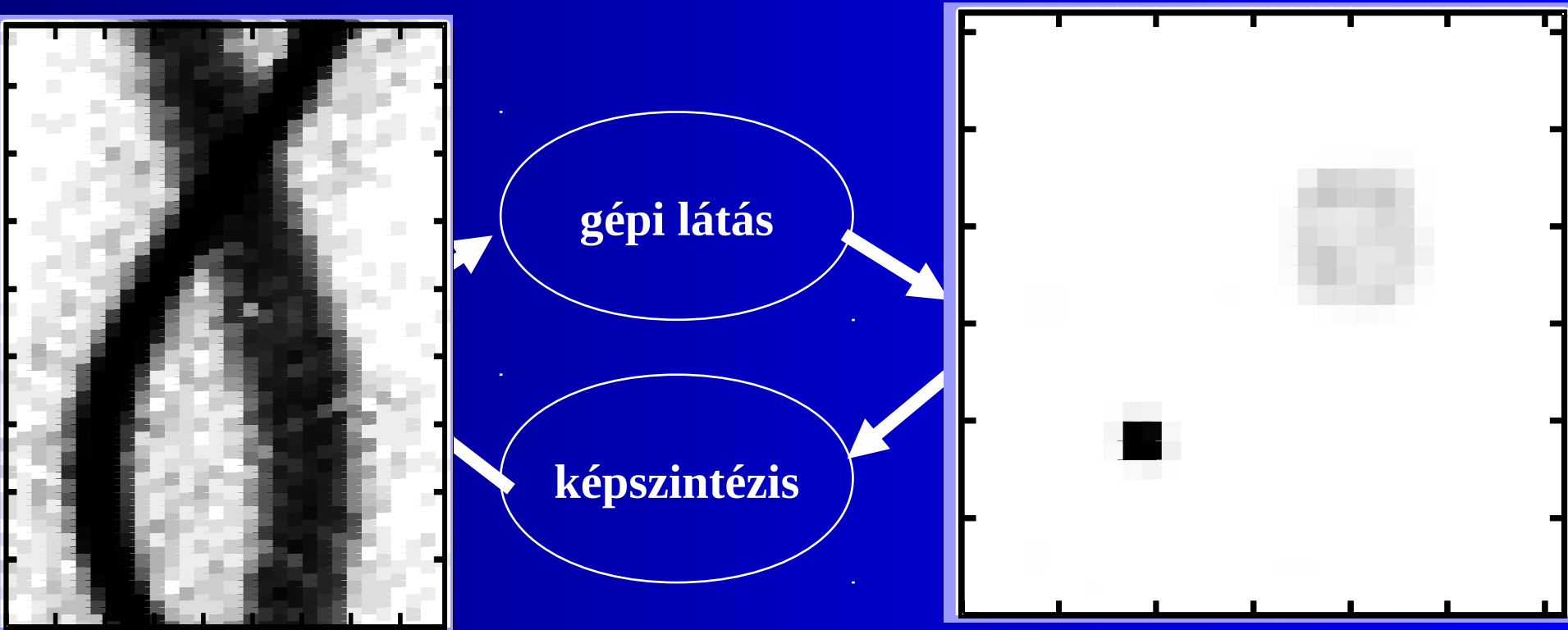
Metafórák:

- optika
- 2D rajzolás
- festés
- analógiák

számítás

mérés

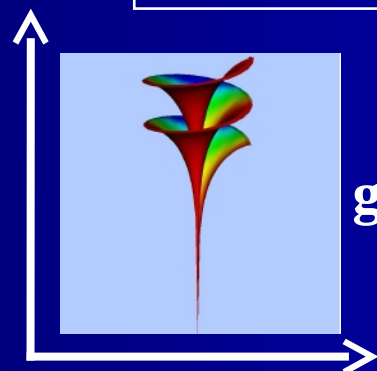
Gépi látás / képfeldolgozás / orvosi képalkotás (rekonstrukció)





optikai param
textúrák

modell

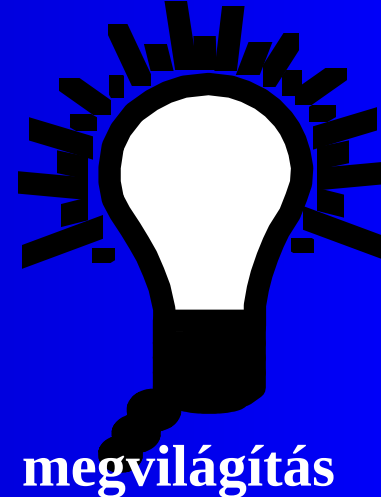


geometria

$$\begin{aligned}x(u,v) &= a \cdot \cos(u) \cdot \sin(v) \\y(u,v) &= a \cdot \sin(u) \cdot \sin(v) \\z(u,v) &= a \cdot \cos(v) + \\&\quad a \cdot \log(\operatorname{tg}(v/2)) + b \cdot u\end{aligned}$$

Modellezés

mozgás



megvilágítás

képszintézis

stúdió
objektumok

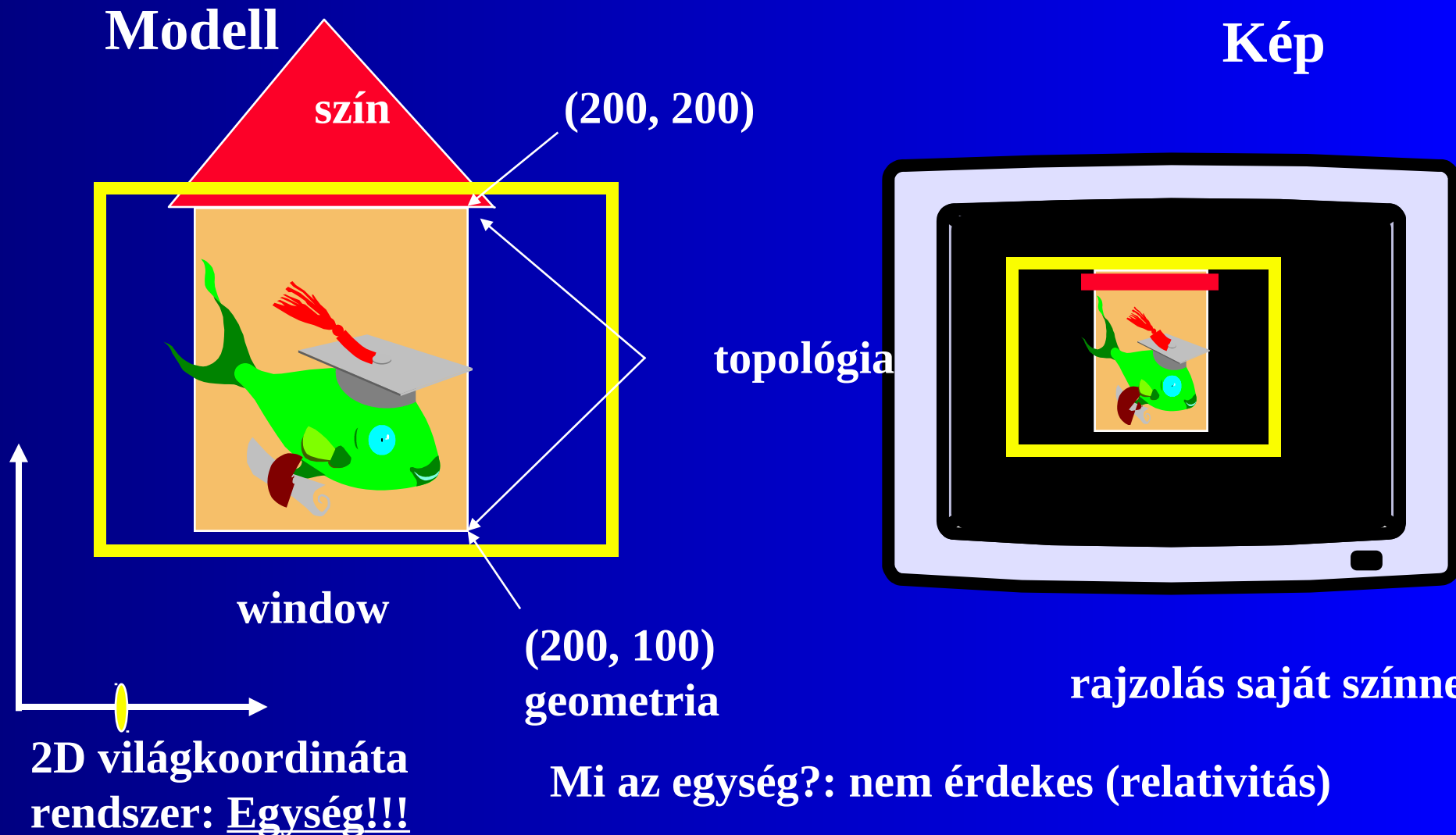


kamera

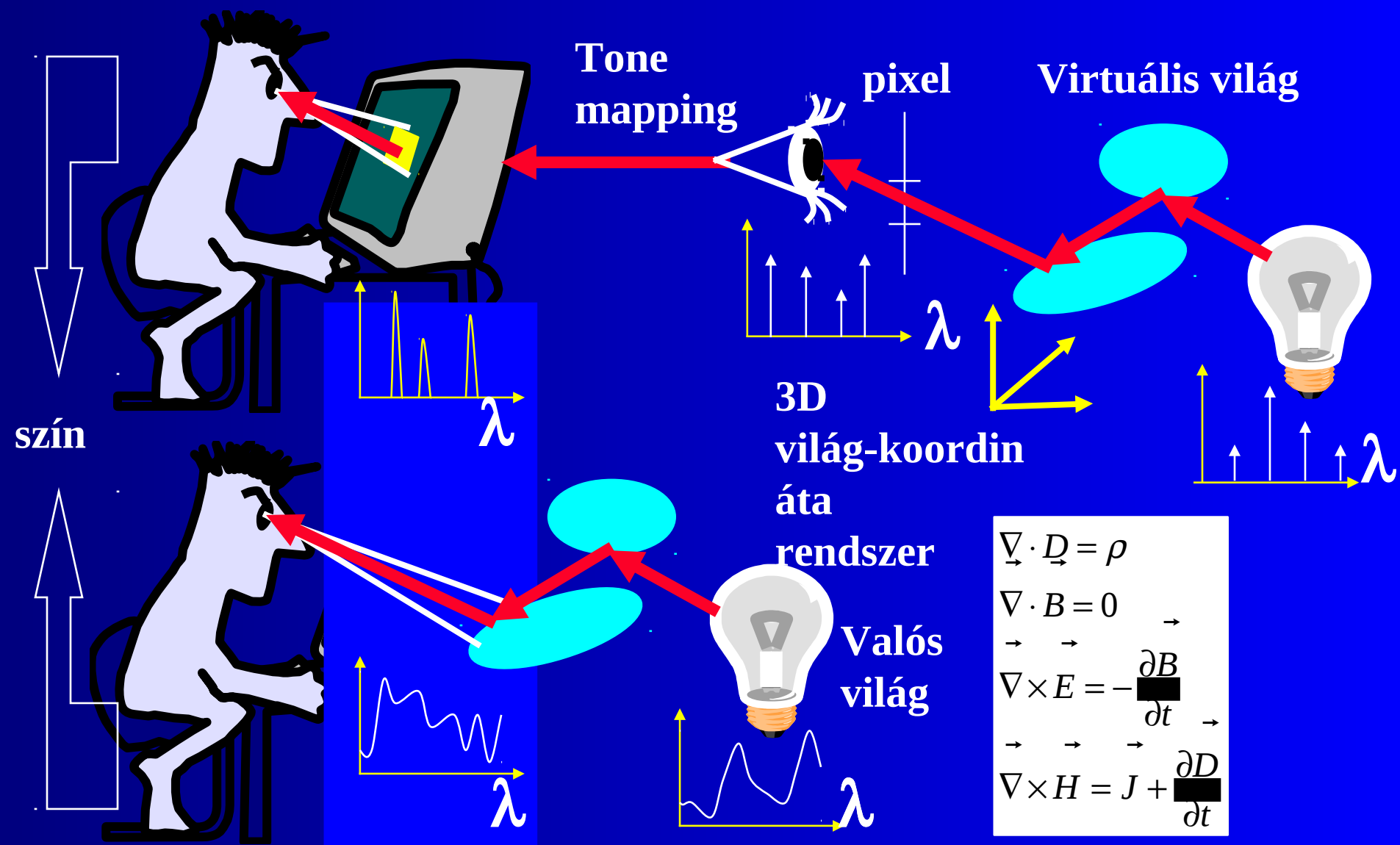


kép (sorozat)

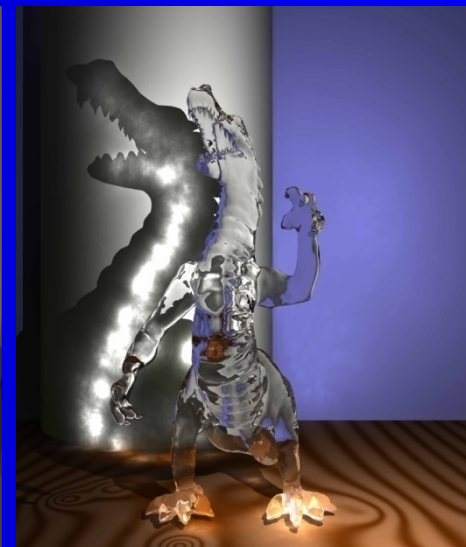
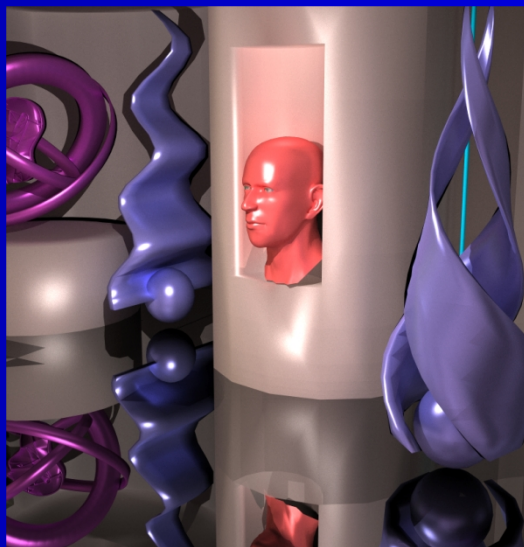
2D világ lefényképezése: rajzolás, festés



3D világ lefényképezése: Optika



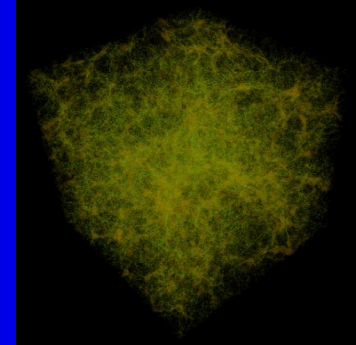
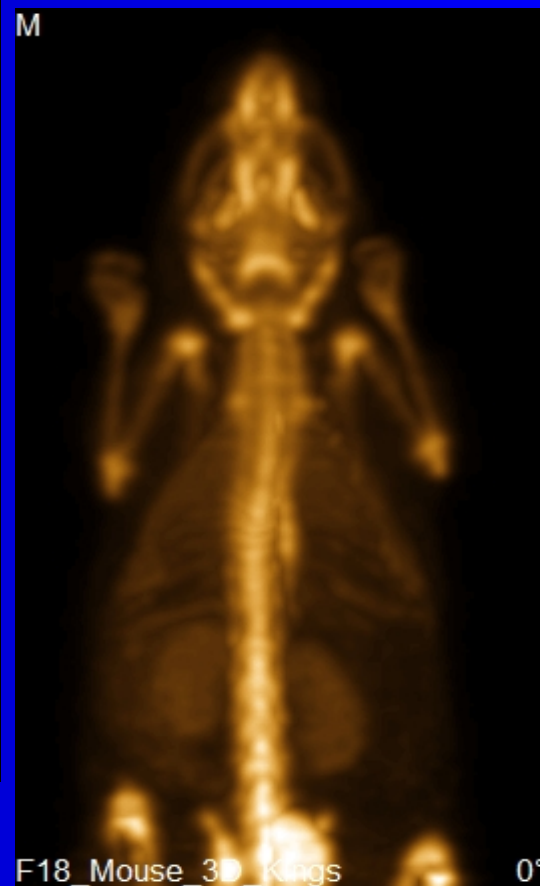
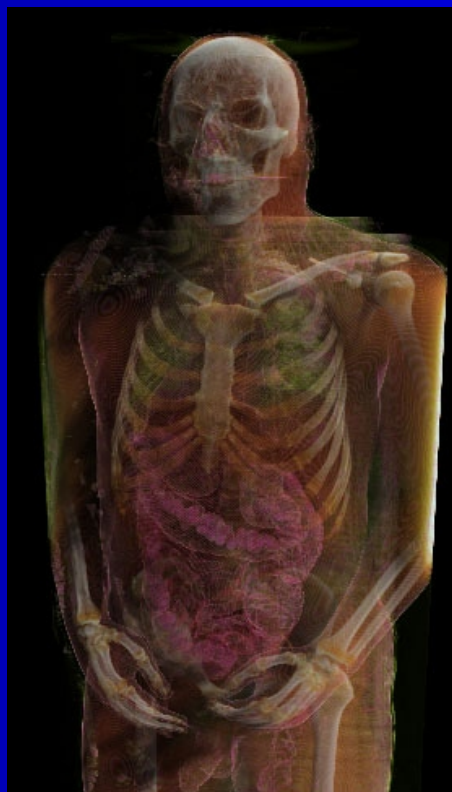
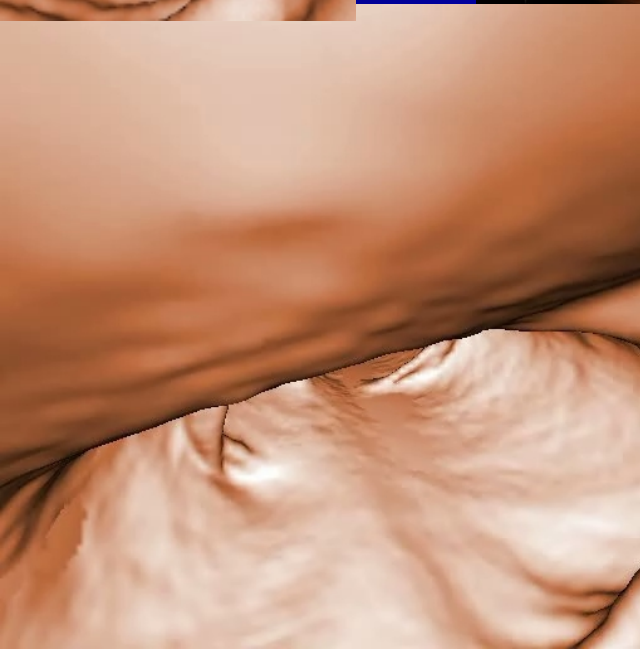
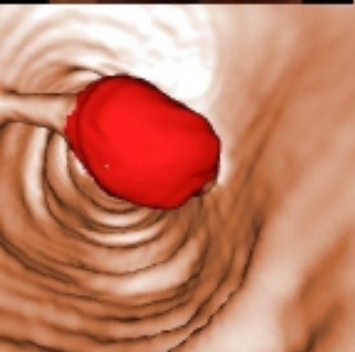
Fotorealisztikus képszintézis



Optika szimulációja =
Realisztikus képek

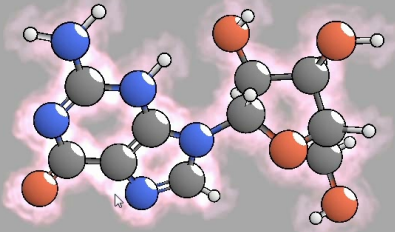
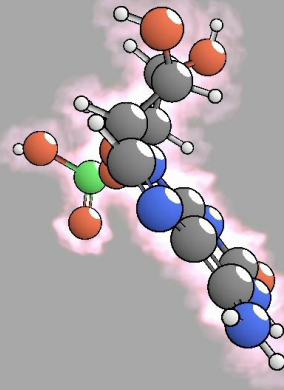
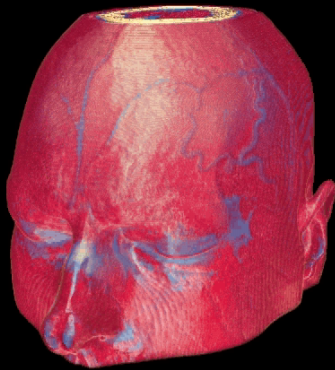
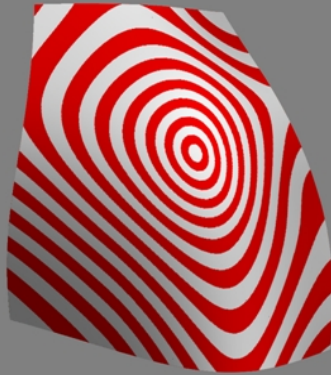
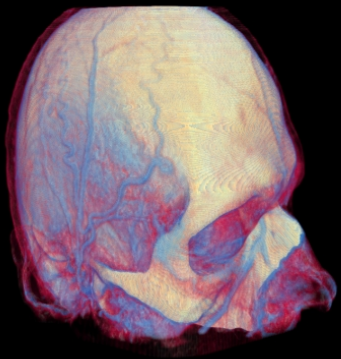


Optikai analógia vizualizációban

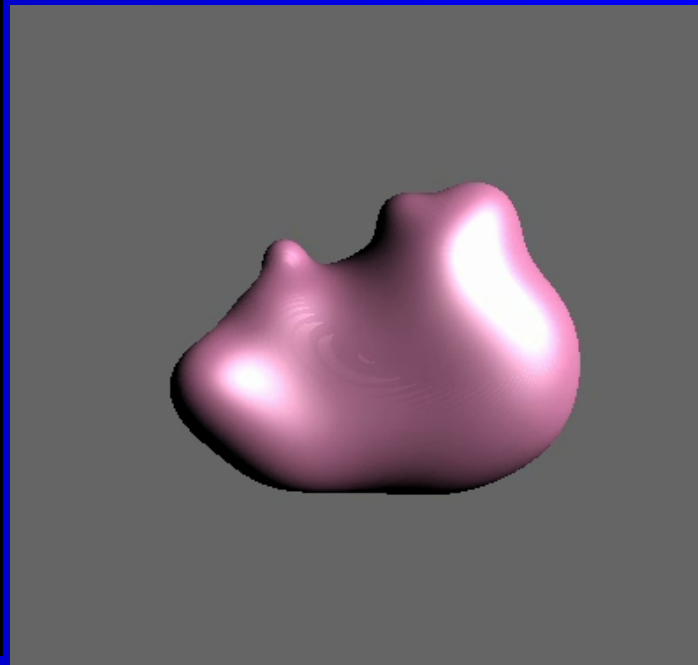
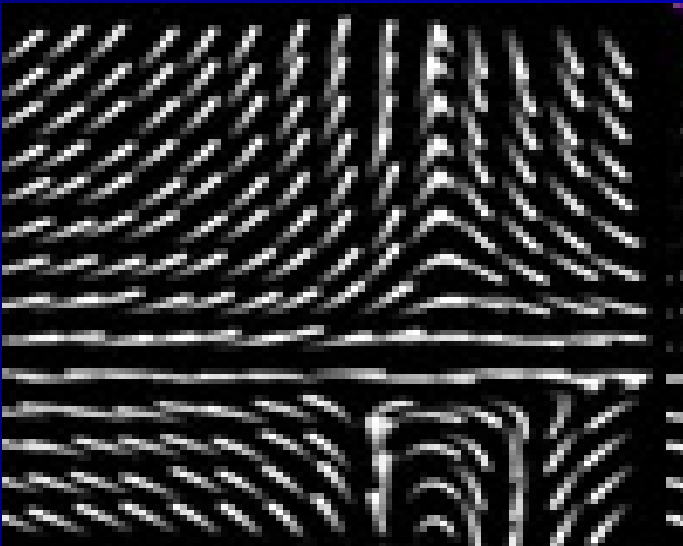
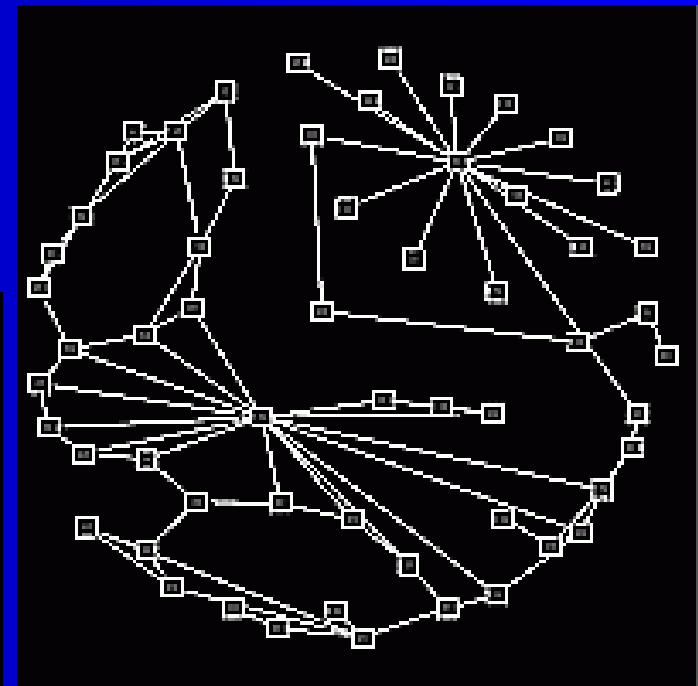
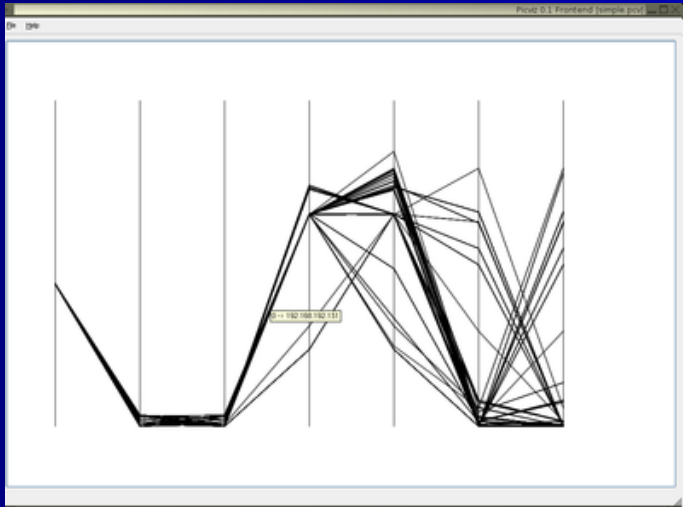


Optikai paraméterek nem optikai jellemzőket
kódolnak: láthatatlant láthatóvá

Illusztratív képszintézis

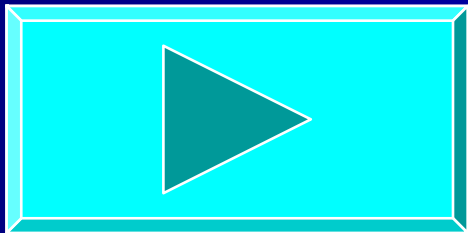
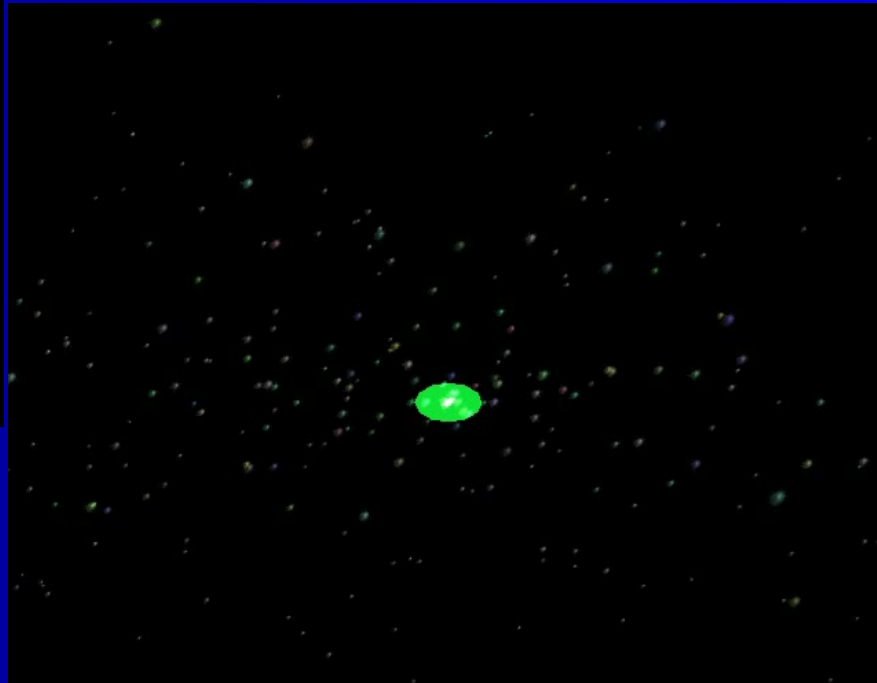


Adatvizualizáció



Kihívások

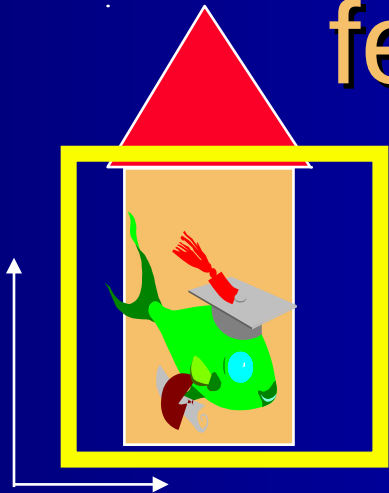
SSDS: Sloan Digital Sky Survey



- Bonyolult modell (giga/terabyte)
- Valós időben: néhány nsec/pixel

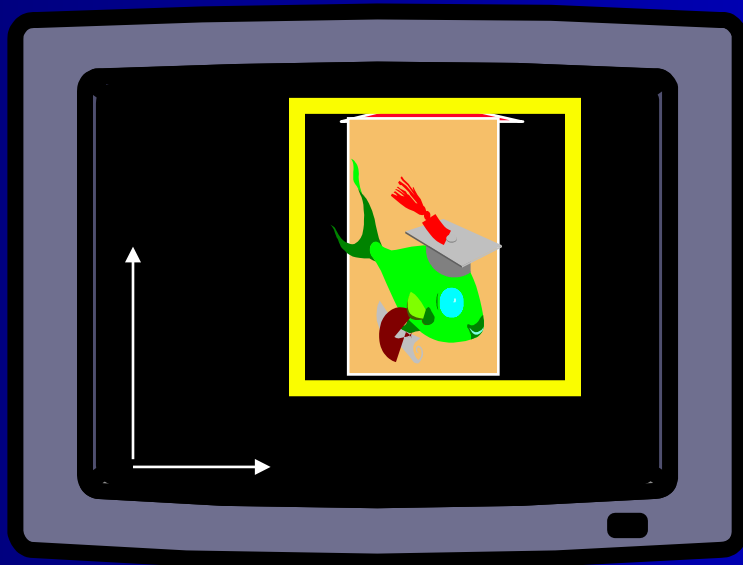


Képszintézis feladatai



Vektorizáció

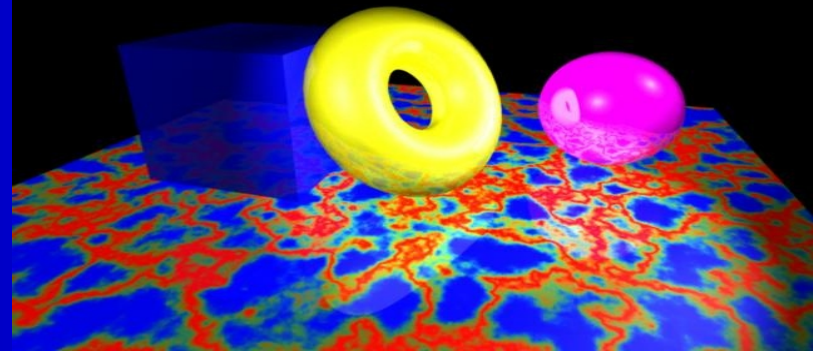
Transzformációk



Vágás

Takarás

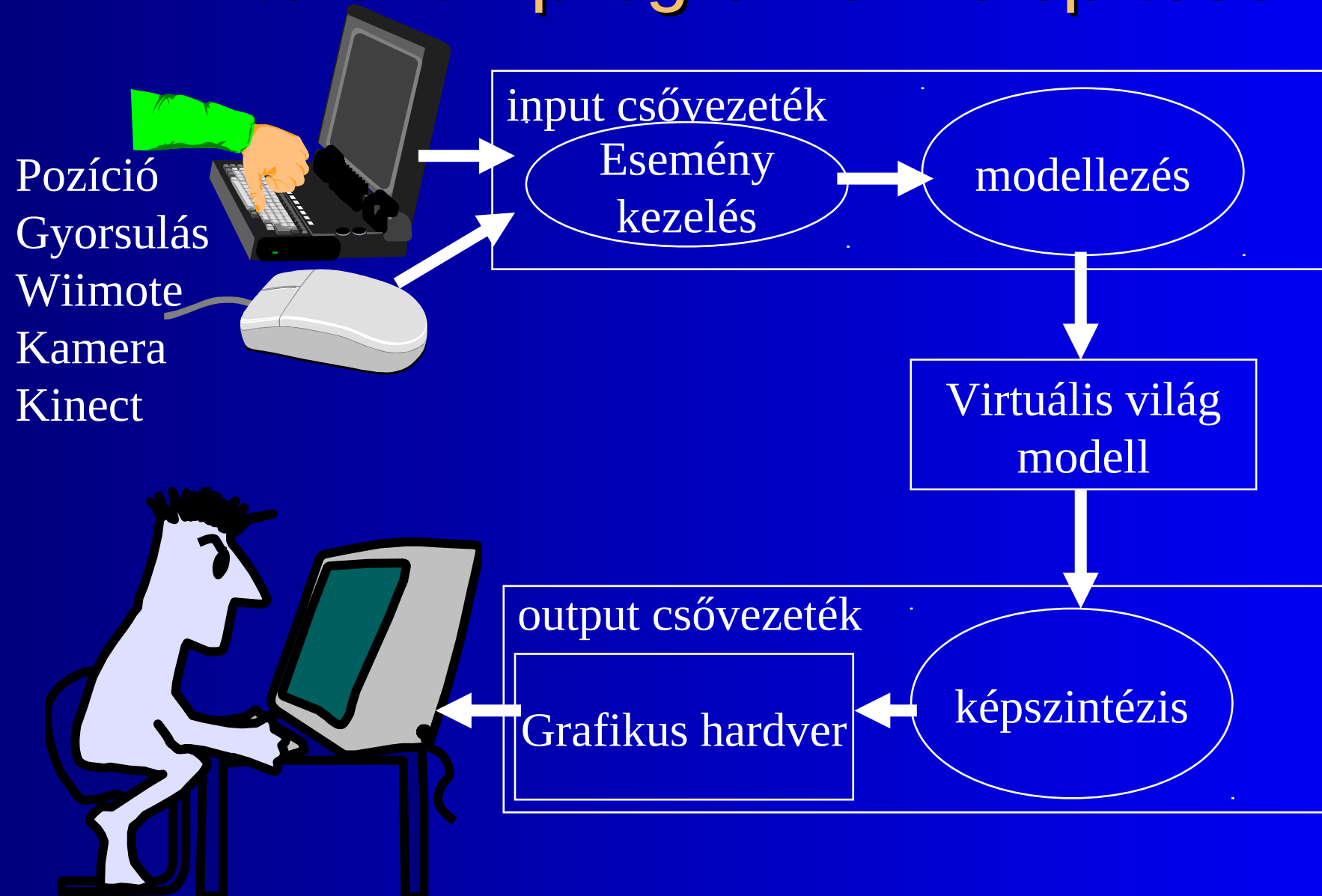
Árnyalás



Minimál grafikus hardver és szoftver

Szirmay-Kalos László

Interaktív programok felépítése



2D Display

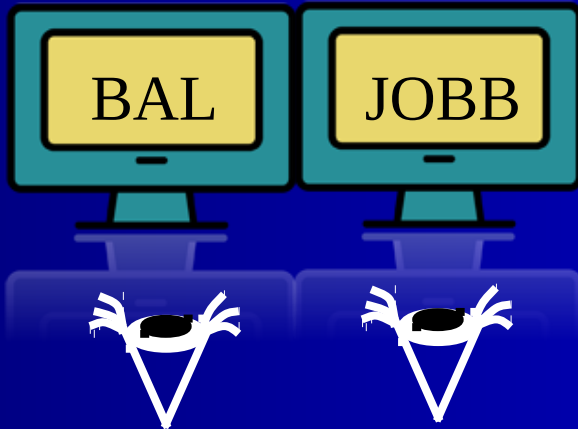


- Pixel:
 - Diffúz
 - Irányfüggetlen intenzitás
- Valós élet: mélység
- Nincs
 - Sztereo parallaxis
 - Mozgás parallaxis
- Olyan mint egy téglalap

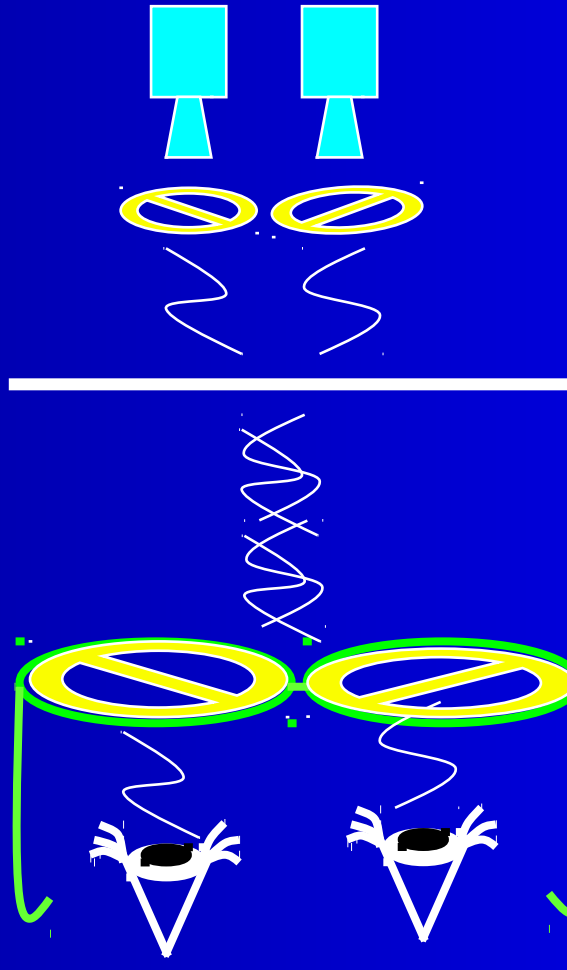
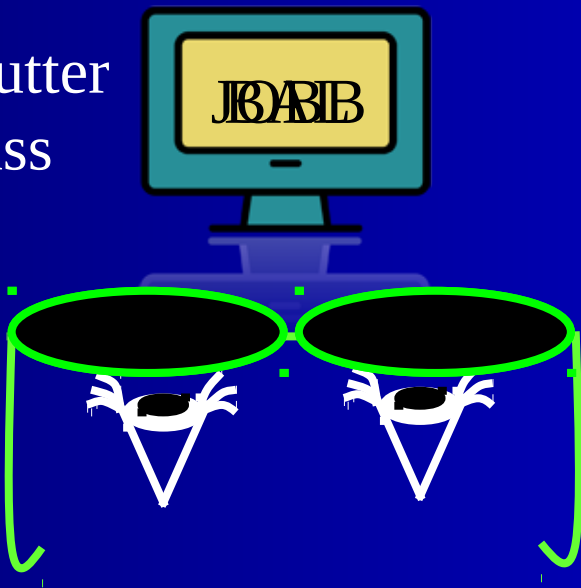


Sztereó 3D

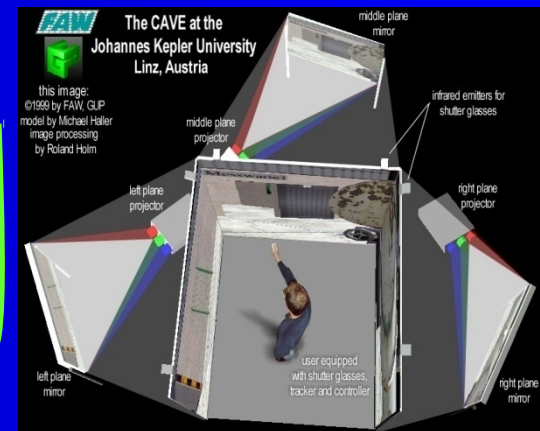
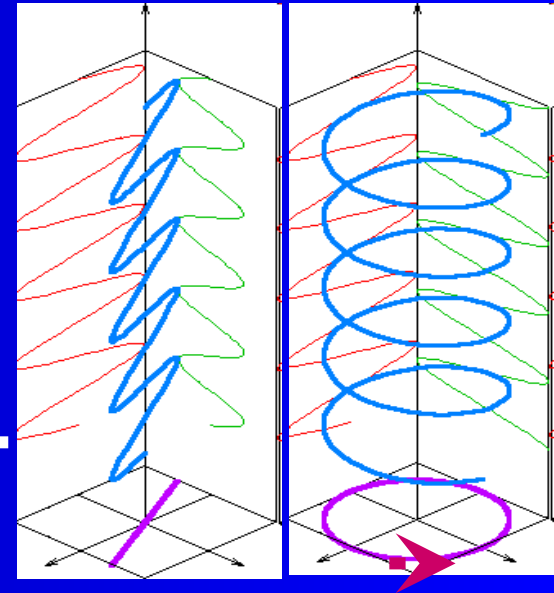
HMD



Shutter
glass



Nincs
mozgás parallaxis
mélységélesség



Mozgásparallaxis követéssel



- Távolság 3 ponttól
- 3 gömb metszete a néző helye

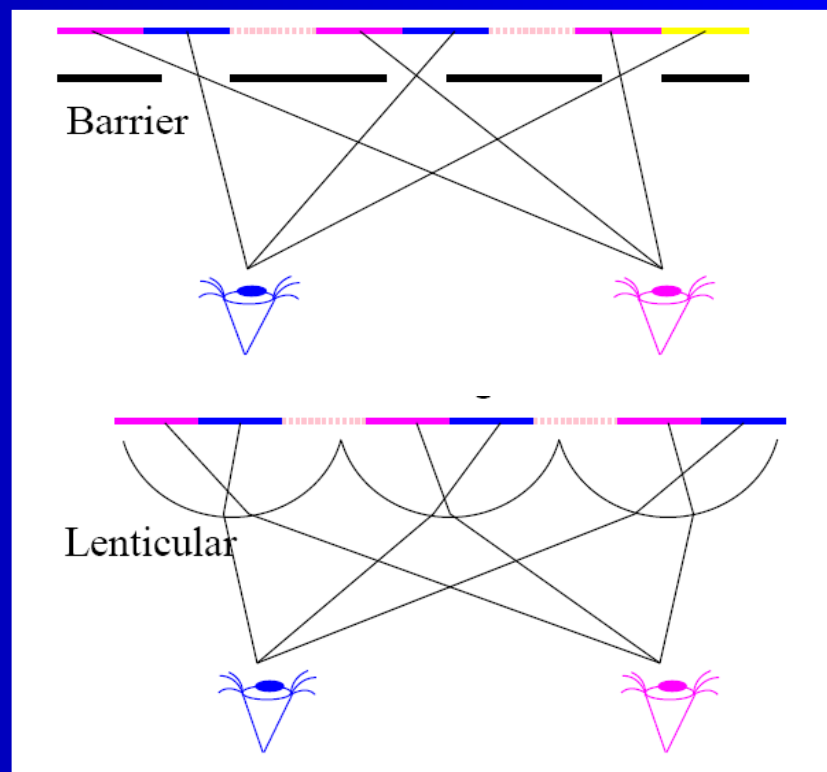


Autosztereoszkópikus display

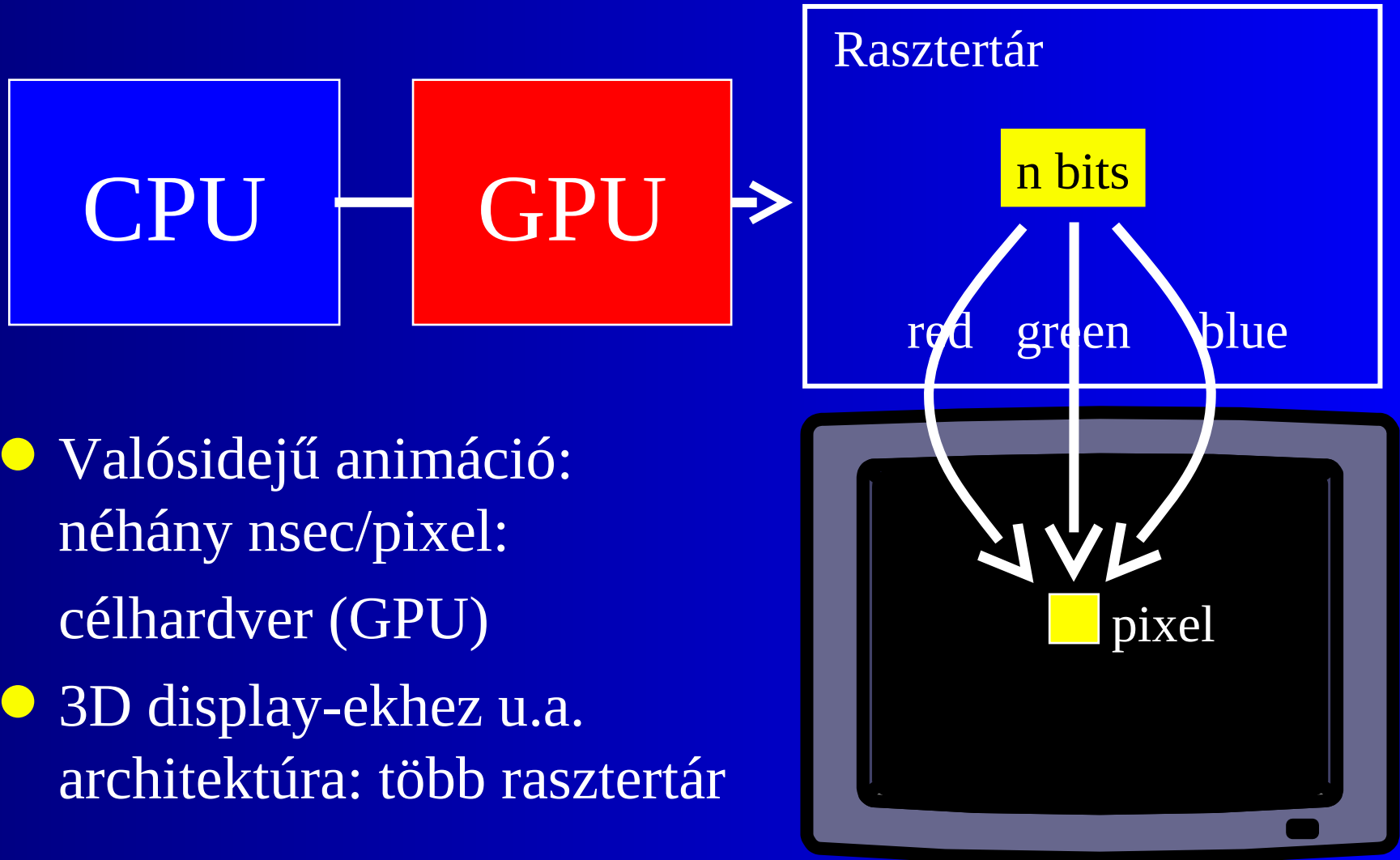
Feltételezett távolság



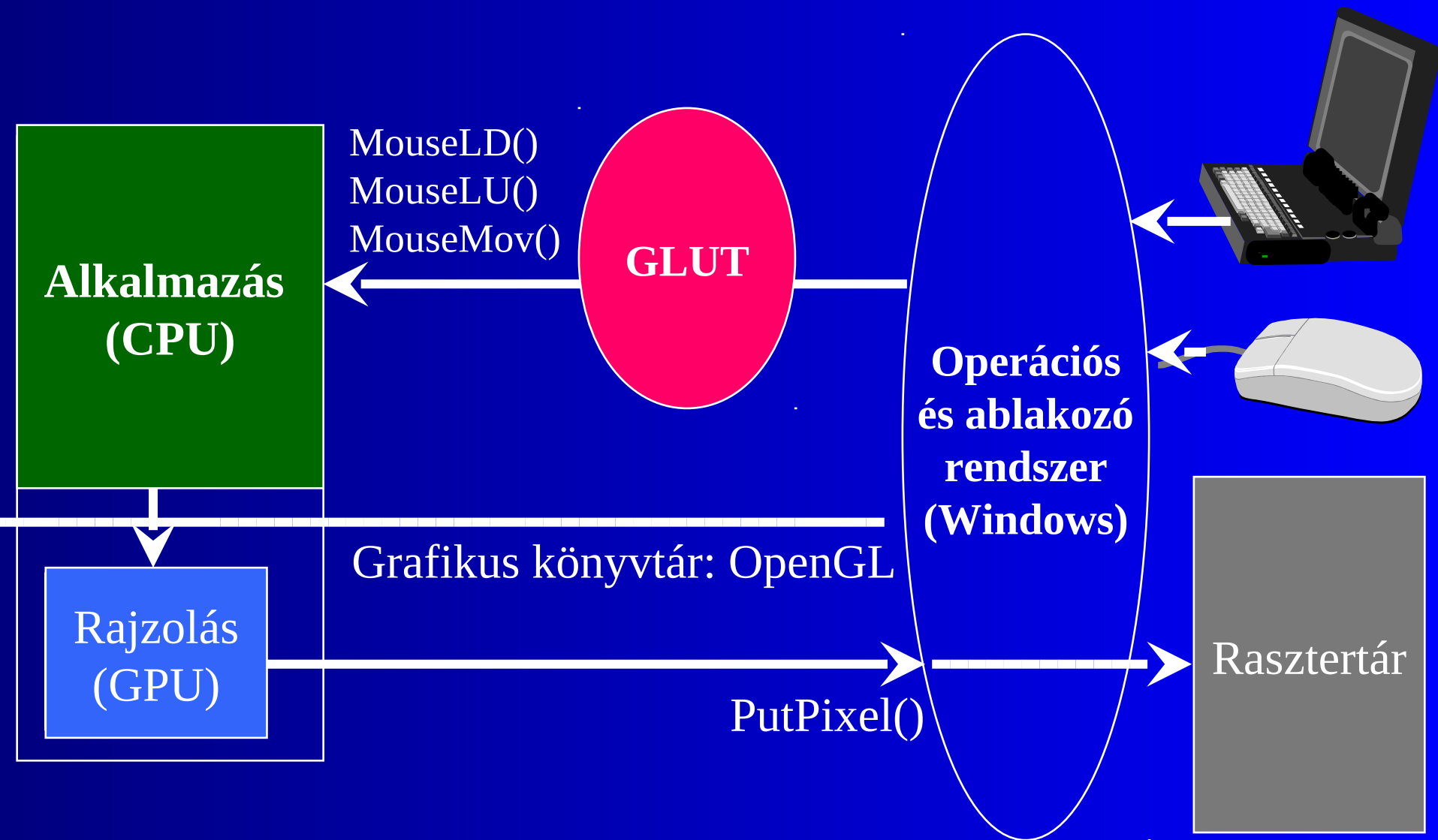
Pixel más fényt emittál
különböző irányokban



Hardver: rasztergrafika



Szoftver



Kimeneti csővezeték: OpenGL

- Képszintézis könyvtár, API
 - 3D geometria (2D speciális eset)
 - rajzolósi állapot (state)
 - grafikus kártya szolgáltatásai
- Op. rendszer, ablakozó rendszer független
 - MVC: csak view, nem controller, nem model
 - nem ablakoz
 - nem kezel bemeneti eszközöket
 - nem kezel fájlokat (pl. képformátumok)

OpenGL szintaxis (C könyvtár)

gl könyvtár része

glVertex3dv(...)

Paraméterszám

2 - (x, y)

3 - (x, y, z),
(R, G, B)

4 - (x, y, z, h)
(R, G, B, A)

Adattípus

b - byte

ub - unsigned byte

s - short

i - int

f - float

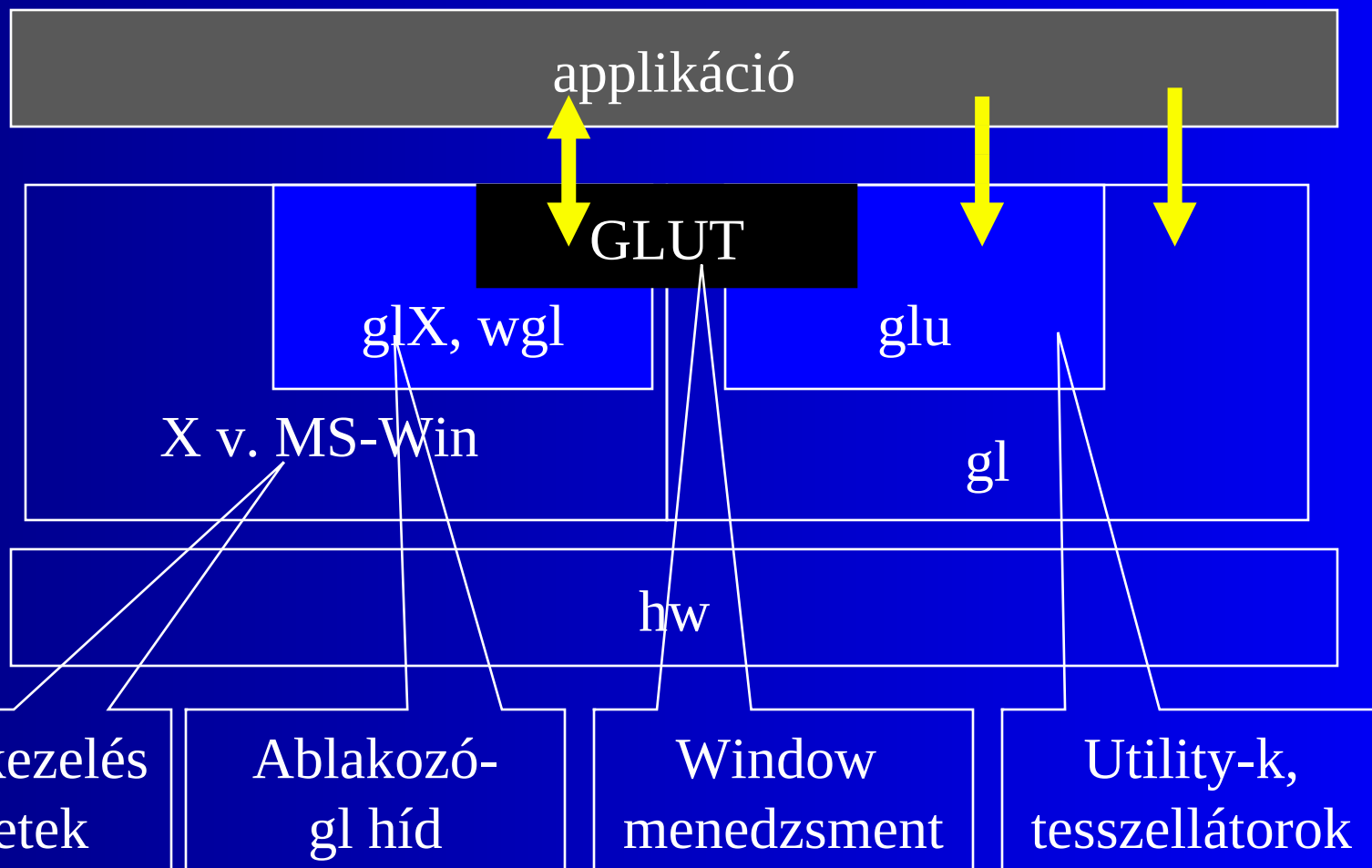
d - double

Vektor vagy skalár

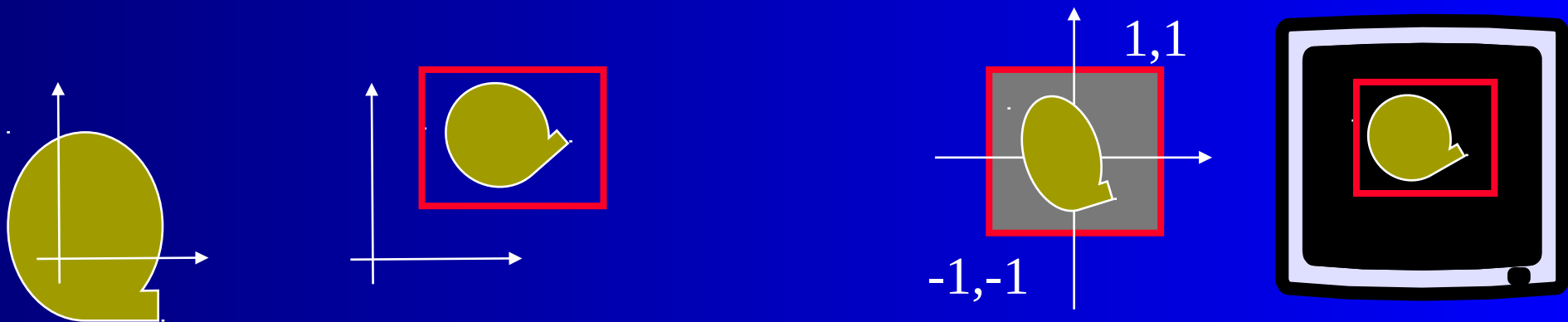
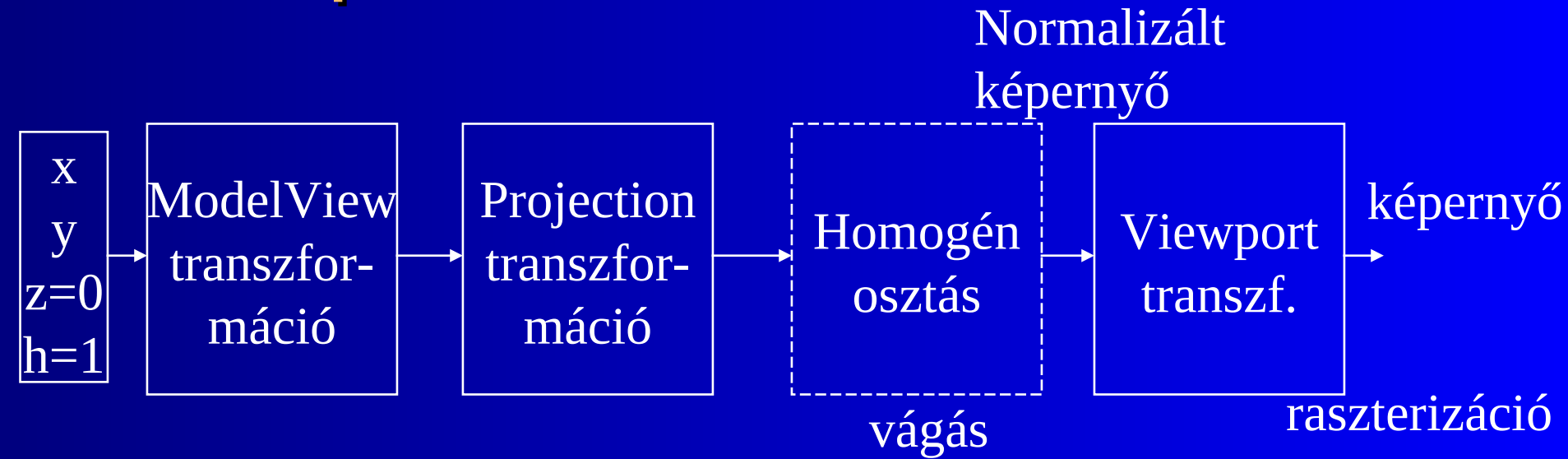
v - vektor

- skalár

Ablakozó – OpenGL – alkalmazás elhelyezkedése

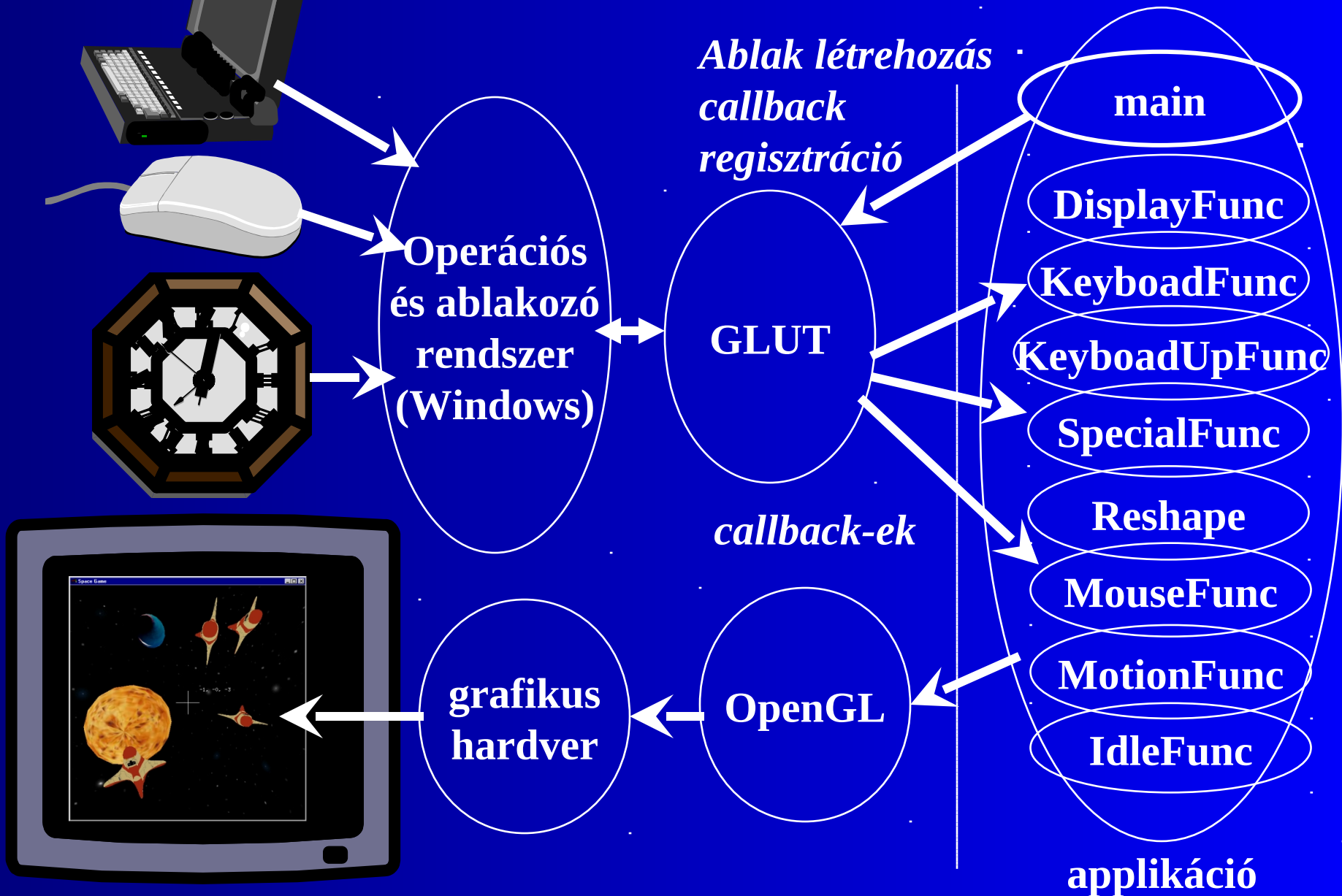


OpenGL transzformációk



Ha a ModelView és Projection egységtranszformáció,
akkor a rajzolási célterület: $(-1,1) \times (-1,1)$

Input/Output kezelés



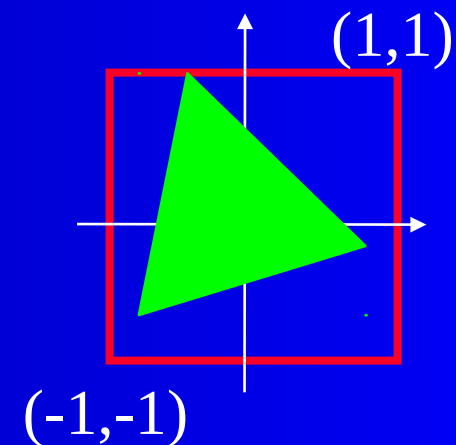
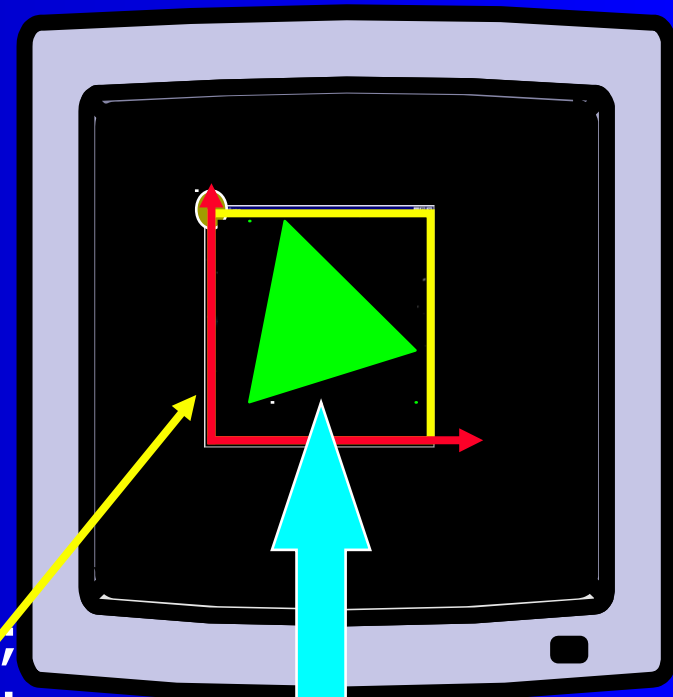
OpenGL alkalmazás (GLUT) main függvény

- Ablak megnyitás
- Rajzolási állapot inicializálás
- Display callback:
 - képernyő törlés, állapotváltás, rajzolás, buffercsere
- Reshape callback:
 - Nézet (viewport), kamera transzformáció átállítása
- Input callback (mouse, keyboard)
- Idle callback (animáció)
- Üzenethurok

```

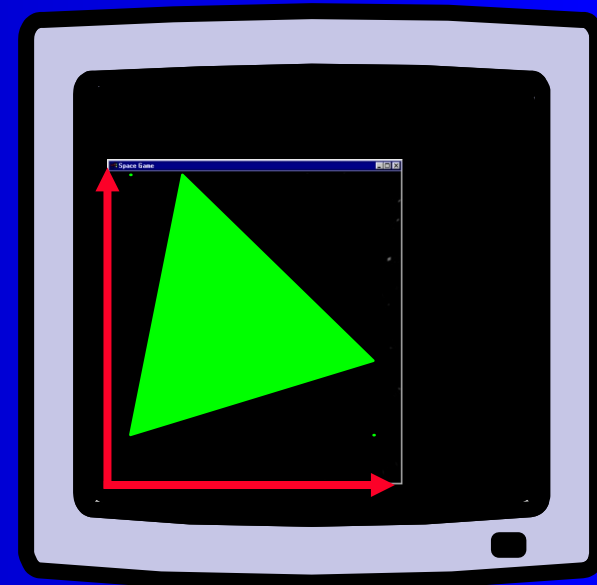
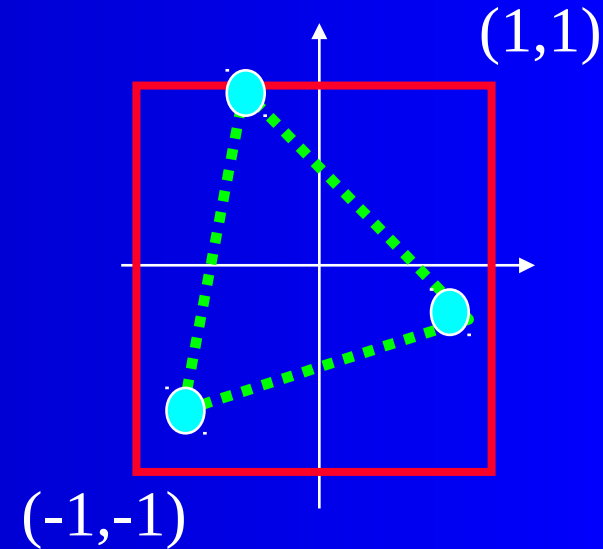
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h> // download!!!
main( int argc, char *argv[] ) {
    glutInitWindowSize(200, 200);
    glutInitWindowPosition(100, 100);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB |
                        GLUT_DOUBLE );
    glutCreateWindow("Sample Window");
    // callback függvények
    glutKeyboardFunc( Keyboard );
    glutDisplayFunc( ReDraw );
    // transzformáció
    glViewport(0, 0, 200, 200);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity( );
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    // fő hurok
    glutMainLoop();
}

```



Eseménykezelés és rajzolás

```
void ReDraw( ) {  
    glClearColor(0, 0, 0, 0);  
    glClear(GL_COLOR_BUFFER_BIT);  
    glutSwapBuffers( );  
}  
  
void Keyboard(unsigned char key,  
              int x, int y) {  
    if (key == 'd') {  
        glColor3d( 0.0, 1.0, 0.0 );  
        glBegin(GL_TRIANGLES);  
            glVertex2d(-0.8, -0.8);  
            glVertex2d(-0.6, 1.0);  
            glVertex2d( 0.8, -0.2);  
        glEnd( );  
        glutSwapBuffers( );  
    }  
}
```



A háromszög újrarajzoláskor megmarad

```
bool haromszog = false;

void ReDraw( ) {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    if ( haromszog ) {
        glColor3d( 0.0, 1.0, 0.0 );
        glBegin(GL_TRIANGLES);
            glVertex2d(-0.8, -0.8);
            glVertex2d(-0.6, 1.0);
            glVertex2d(0.8, -0.2);
        glEnd( );
    }
    glutSwapBuffers();
}

void Keyboard(unsigned char key, int x, int y) {
    if (key == 'd') {
        haromszog = true;
        ReDraw( );
    }
}
```



```
bool haromszog = false;
```

```
void ReDraw( ) {
```

```
    glClearColor(0, 0, 0, 0);
```

```
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    if ( haromszog ) {
```

```
        glColor3d( 0.0, 1.0, 0.0 );
```

```
        glBegin(GL_TRIANGLES);
```

```
            glVertex2d(-0.8, -0.8);
```

```
            glVertex2d(-0.6, 1.0);
```

```
            glVertex2d(0.8, -0.2);
```

```
        glEnd( );
```

```
    }
```

```
    glutSwapBuffers( );
```

```
}
```

```
void Keyboard(unsigned char key, int x, int y) {
```

```
    if (key == 'd') {
```

```
        haromszog = true;
```

```
        glutPostRedisplay();
```

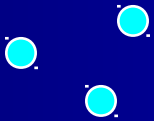
```
    }
```

```
}
```

Nem rajzol
feleslegesen
sokszor

OpenGL: primitívek

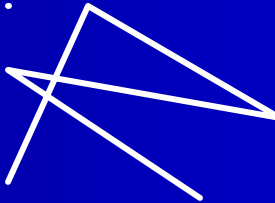
```
glBegin(primitív típus);  
    Csúcspont tulajdonságok ...  
    glVertex3f(x, y, z);  
    ...  
glEnd( );
```



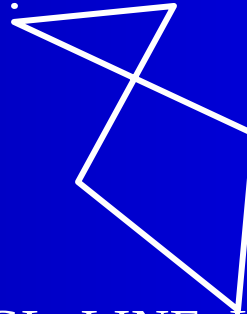
GL_POINTS



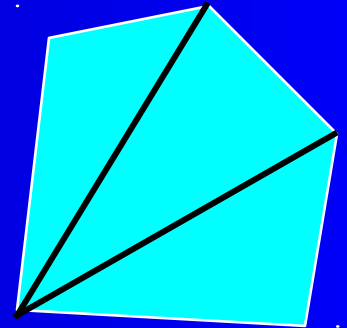
GL_LINES



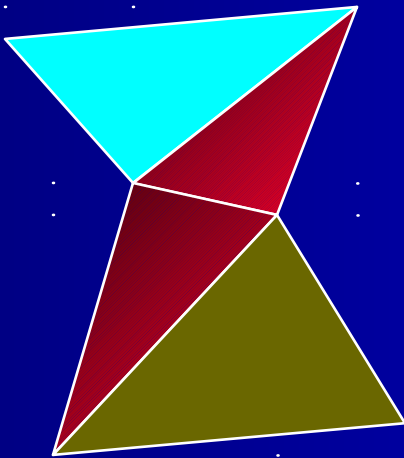
GL_LINE_STRIP



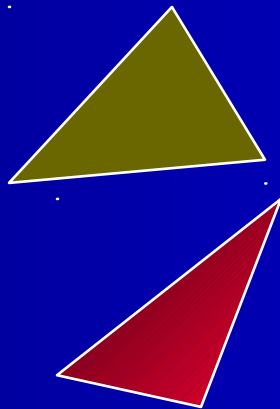
GL_LINE_LOOP



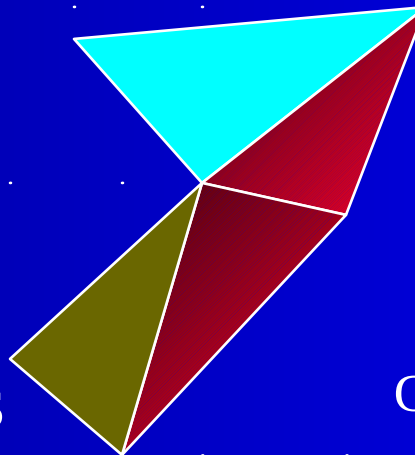
GL_POLYGON



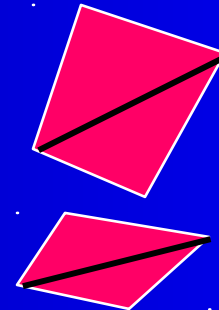
GL_TRIANGLE_STRIP



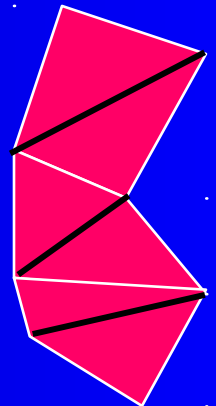
GL_TRIANGLES



GL_TRIANGLE_FAN

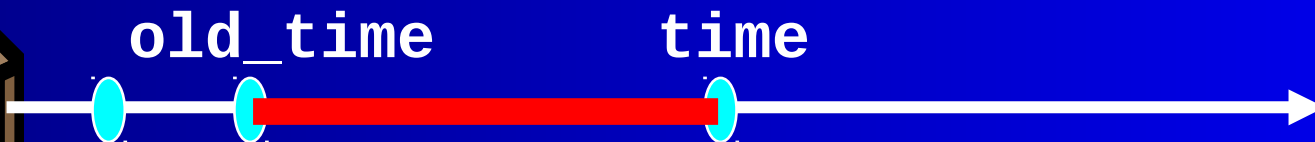
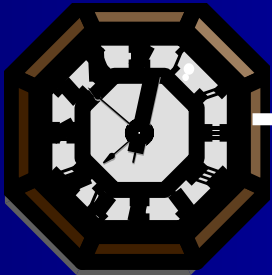


GL_QUADS



GL_QUAD_STRIP

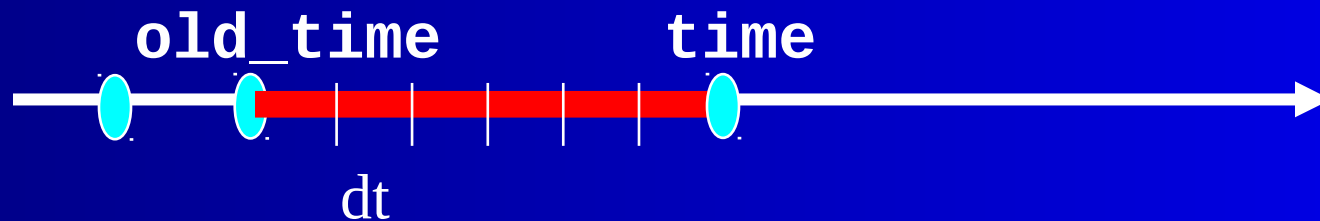
Időkezelés (szimuláció)



```
float time : glutGet(GLUT_ELAPSED_TIME); ☠
```

```
void IdleFunc( ) { // idle call back  
    float old_time = time;  
    time = glutGet( GLUT_ELAPSED_TIME );  
  
    SimulateWorld(old_time, time);  
  
    glutPostRedisplay(); // felrajzolás  
}
```

Diszkrét idő szimuláció



```
SimulateWorld(float tstart, float tend) {  
    for(float ts = tstart; ts < tend; ts += dt) {  
        float te = min(tend, ts + dt);  
        for each object obj: obj.Control(ts, te);  
        for each object obj: obj.Animate(ts, te);  
    }  
}
```

Házi

```
//=====
// Szamitogepes grafika hazi feladat keret. Ervenyes 2013-tol.
// A //~::~::~::~::~::~::~::~::~::~::~
// sorokon beluli reszben celszeru garazdalkodni.
// A program csak ebben a fajlban lehet, a fajl 1 byte-os ASCII karaktereket tartalmazhat.
// Tilos:
// - mast "beincludolni", illetve mas konyvtarat hasznalni
// - fajlmuveleteket vegezni (printf is fajlmuvelet!)
// - new operatort hivni az onInitialization-t kivéve, a lefoglalt adat felszabadítása nélkül
// - felesleges programsorokat a beadott programban hagyni
// - további kommenteket a beadott programba írni a forrasmegjelolest kommentjeit kiveve
// -----
// A feladatot ANSI C++ ... ,
// -----
// A feladatmegoldasokban csak olyan gl/glu/glut függvények használhatók, amelyek
// 1. Az oran a feladatkiadásig elhangzottak ES (logikai AND muvelet)
// 2. Az alábbi listaban szerepelnek: ...
// ...
// NYILATKOZAT
// -----
// Nev      : <VEZETEKNEV(EK)> <KERESZTNEV(EK)>
// Neptun   : <NEPTUN KOD>
// -----
// ezennel kijelentem, hogy a feladatot magam keszitettem, es ha barmilyen segitseget
// igénybe vettem ..., akkor a forrast es az atvett reszt kommentekben egyertelmuen jeloltem ...
//=====
#include <math.h>
#include <stdlib.h>
#ifdef __APPLE__

#include <OpenGL/gl.h>

#include <OpenGL/glu.h>

#include <GLUT/glut.h>

#else
```

```
//~~~~~Innentol modosithatod...
class Vector { ... };
class Color { ... };

Color image[screenWidth*screenHeight]; // kép

void onInitialization( ) { // Inicializacio,
    glViewport(0, 0, screenWidth, screenHeight);
    ...
}

void onDisplay( ) {
    glClearColor(0.1f, 0.2f, 0.3f, 1.0f); // torlesi szin beallitasa
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // kepernyo torles
    glDrawPixels(screenWidth, screenHeight, GL_RGB, GL_FLOAT, image);
    glColor3f(0,0,1); glBegin(GL_TRIANGLES); glVertex2f(-0.2f, -0.2f); ... glEnd();
    glutSwapBuffers();
}

void onKeyboard(unsigned char key, int x, int y) { // Billentyuzet esemeny
    if (key == 'd') glutPostRedisplay( ); // d beture rajzold ujra a kepet
}

void onKeyboardUp(unsigned char key, int x, int y) { }

void onMouse(int button, int state, int x, int y) { // Eger esemeny
    if (button == GLUT_LEFT && state == GLUT_DOWN);
}

void onMouseMotion(int x, int y) { }

void onIdle( ) { // `Idle' esemenykezelo, jelzi, hogy az ido telik
    long time = glutGet(GLUT_ELAPSED_TIME); // program inditasa ota eltelt ido
}

// ...Idaig modosithatod~~~~~
```

Házi

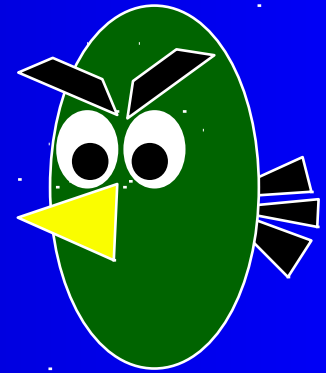
Még mindig házi

// A C++ program belepesi pontja, nem bántani!

```
int main(int argc, char **argv) {  
    glutInit(&argc, argv);           // GLUT inicializalasa  
    glutInitWindowSize(600, 600);  
    glutInitWindowPosition(100, 100);  
        // 8 bites R,G,B,A + dupla buffer + melyseg buffer  
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);  
    glutCreateWindow("Grafika hazi feladat");  
  
    glMatrixMode(GL_MODELVIEW);      // A MODELVIEW egysegmatrix  
    glLoadIdentity();  
    glMatrixMode(GL_PROJECTION);     // A PROJECTION egysegmatrix  
    glLoadIdentity();  
  
    onInitialization();              // Az altalad irt inicializalas  
  
    glutDisplayFunc(onDisplay);       // Esemenykezelok regisztralasa  
    glutMouseFunc(onMouse);  
    glutIdleFunc(onIdle);  
    glutKeyboardFunc(onKeyboard);  
    glutKeyboardUpFunc(onKeyboardUp);  
    glutMotionFunc(onMouseMotion);  
    glutMainLoop(); // Esemenykezelok hurok  
    return 0;  
}
```


0. házi: Kaksiulotteisia vihaisia lintuja

punainen lintu



vihreä lintu

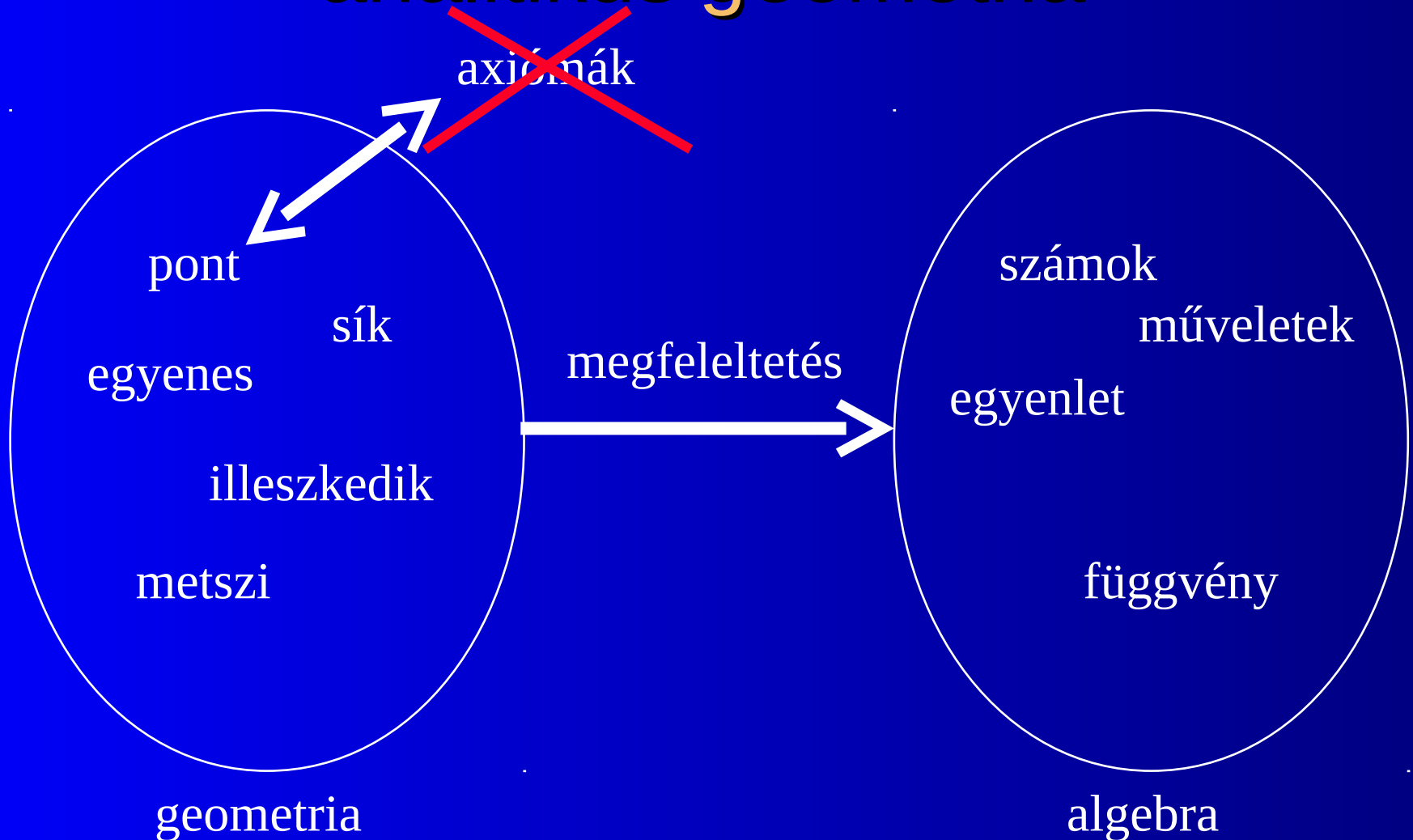
Analitikus (koordináta) geometriai gyorsfalpaló

Szirmay-Kalos László

Geometriák

- Geometriák
 - Axiómák
 - Igaz állítások
 - alapfogalmak implicit definíciója
 - Definíciók + tételek
 - Célok és eszközrendszer
- Számítógépes grafikában fontos geometriák
 - Euklideszi (sík/tér, metrikus, analitikus)
 - Projektív (tér, analitikus)
 - Differenciál geometria
 - Fraktál geometria
 - Diszkrét geometria

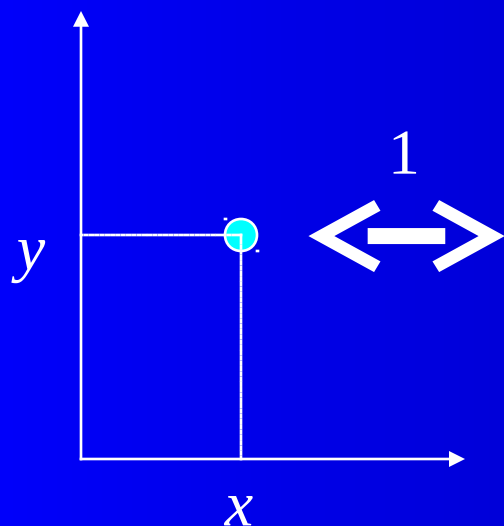
Mindent számmal: analitikus geometria



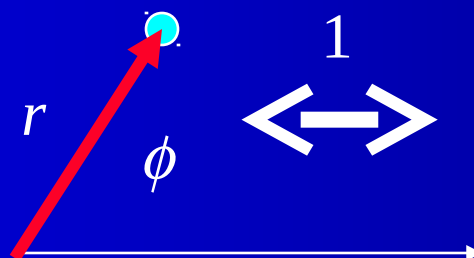
Pontok, alakzatok megadása

- Mindent számmal!
 - Koordináta rendszer
 - Koordináták megadása
- Koordináta rendszerek
 - Descartes
 - Polár
 - Homogén

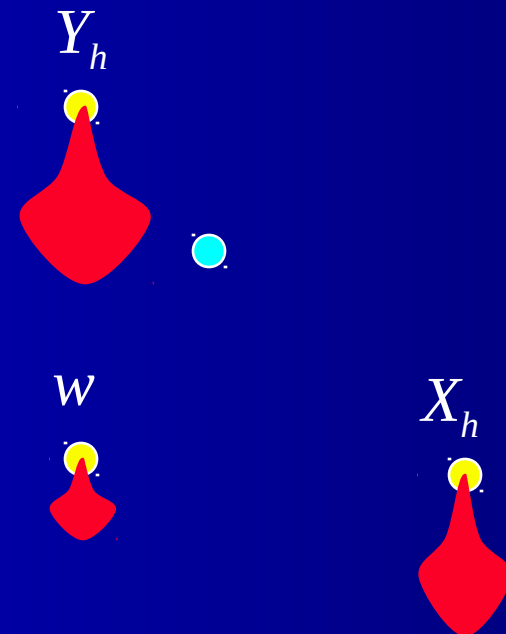
Koordináta rendszerek



Descartes



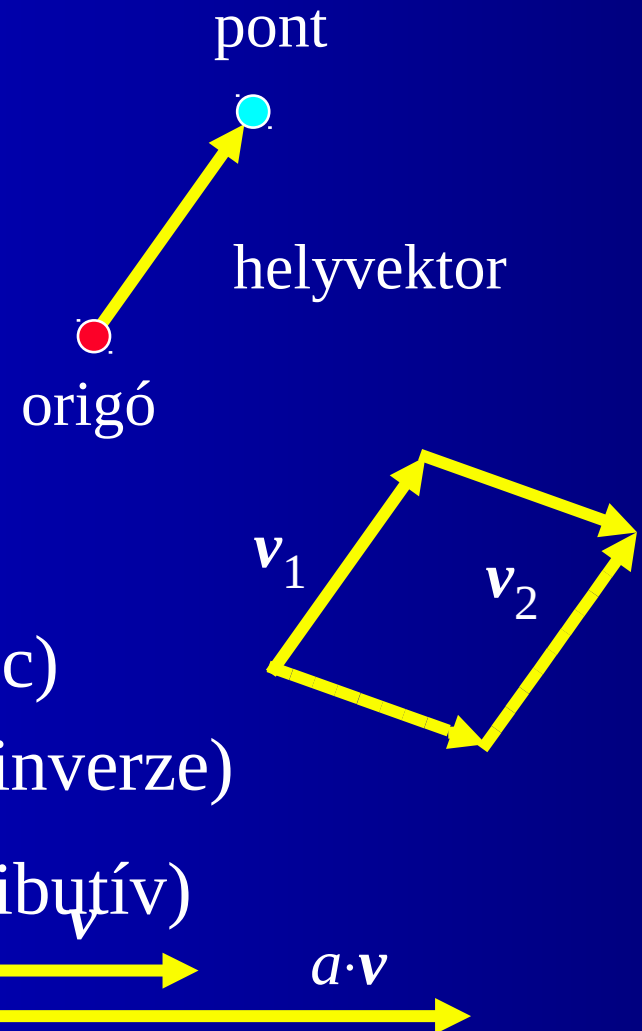
Polár



Baricentrikus
Homogén

Pontfüggvények: Mozgatás

- Vektor = eltolás: $\mathbf{v}$
- Iránya és hossza ($|\mathbf{v}|$) van
- Helyvektor
 - De vektor $\neq$ pont !!!
- Vektorműveletek
 - $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$ (kommutatív, asszoc)
 - $\mathbf{v}_1 = \mathbf{v} - \mathbf{v}_2$ (összeadásnak van inverze)
 - $\mathbf{v}_1 = a \cdot \mathbf{v}$ (összeadásra disztributív)



Skaláris szorzás

- Definíció

- $\mathbf{v1} \cdot \mathbf{v2} = |\mathbf{v1}| \cdot |\mathbf{v2}| \cdot \cos \alpha$

- Jelentés

- Egyik vektor vetülete a másikra * másik hossza

- Tulajdonságok

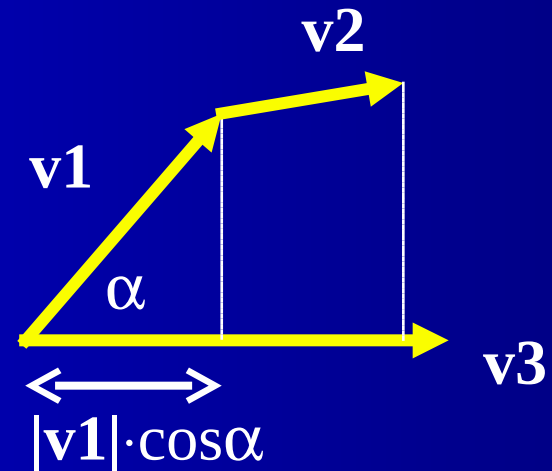
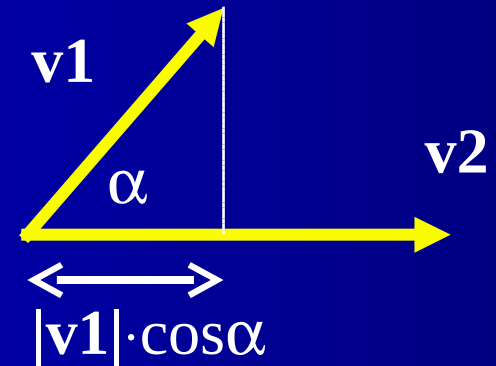
- Kommutatív

$$\mathbf{v1} \cdot \mathbf{v2} = \mathbf{v2} \cdot \mathbf{v1}$$

- Összeadással disztributív

$$\mathbf{v3} \cdot (\mathbf{v2} + \mathbf{v1}) = \mathbf{v3} \cdot \mathbf{v2} + \mathbf{v3} \cdot \mathbf{v1}$$

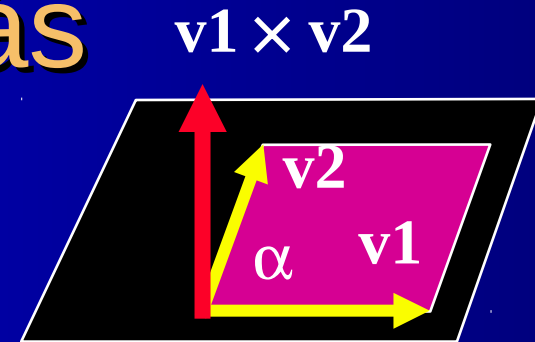
- $\mathbf{v} \cdot \mathbf{v} = |\mathbf{v}|^2$



Vektoriális szorzás

- Definíció

- $|\mathbf{v1} \times \mathbf{v2}| = |\mathbf{v1}| \cdot |\mathbf{v2}| \cdot \sin \alpha$
- Merőleges, jobbkéz szabály



- Jelentés

- Paralelogramma területe, síkjára merőleges
- (Egyik vektor vetülete a másikra merőleges síkra + 90 fokos forgatás) * másik hossza

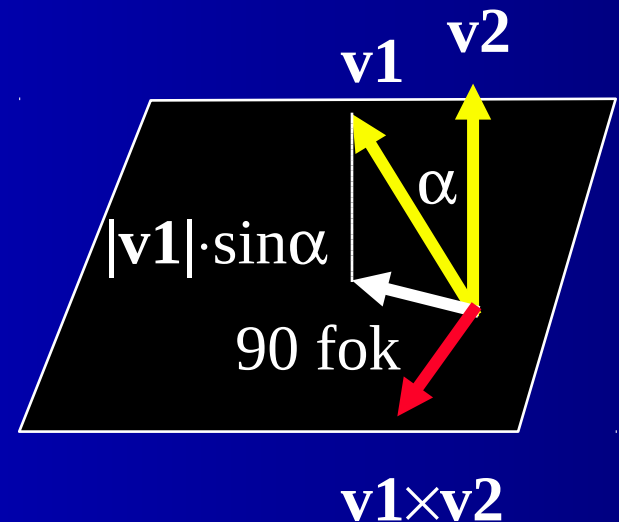
- Tulajdonságok

- Alternáló

$$\mathbf{v1} \times \mathbf{v2} = - \mathbf{v2} \times \mathbf{v1}$$

- Összeadással disztributív

$$\mathbf{v3} \times (\mathbf{v2} + \mathbf{v1}) = \mathbf{v3} \times \mathbf{v2} + \mathbf{v3} \times \mathbf{v1}$$



Descartes koordinátarendszer

- Egyértelmű ($x = \mathbf{v} \cdot \mathbf{i}$, $y = \mathbf{v} \cdot \mathbf{j}$)

- Műveletek koordinátákban

Összeadás:

$$\mathbf{v}_1 + \mathbf{v}_2 = (x_1 + x_2)\mathbf{i} + (y_1 + y_2)\mathbf{j}$$

Skaláris szorzás:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = (x_1\mathbf{i} + y_1\mathbf{j}) \cdot (x_2\mathbf{i} + y_2\mathbf{j}) = (x_1x_2 + y_1y_2)$$

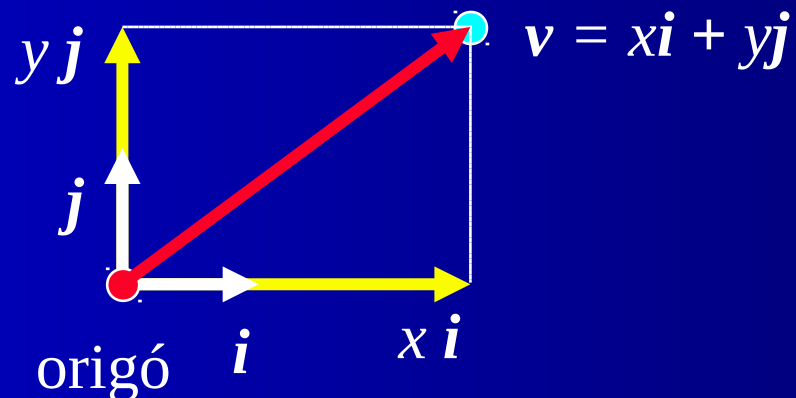
Vektoriális szorzás:

$$\mathbf{v}_1 \times \mathbf{v}_2 = (x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}) \times (x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}) = \\ (y_1z_2 - y_2z_1)\mathbf{i} + (x_2z_1 - x_1z_2)\mathbf{j} + (x_1y_2 - y_1x_2)\mathbf{k}$$

$$\begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

Abszolút érték: _____

$$|\mathbf{v}| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{x^2 + y^2 + z^2}$$



Vektor és Pont nem ugyanaz!

- 2D: vektor (x, y)
- 3D: vektor (x, y, z)
- Műveletek:
 - Vektor + Vektor = Vektor
 - Vektor – Vektor = Vektor
 - Vektor $\times$ Vektor = Vektor
 - Vektor $\cdot$ Vektor = Skalár
 - Vektor * Skalár = Vektor

Pont (x, y)

Pont (x, y, z)

Pont + Vektor = Pont (eltolás)

Pont – Pont = Vektor

Vektor és Pont tényleg nem ugyanaz!

- 2D: vektor $(x, y, 0)$
- 3D: vektor $(x, y, z, 0)$
- Műveletek:

Vektor + Vektor = Vektor

Vektor – Vektor = Vektor

Vektor · Vektor = Skalár

Vektor * Skalár = Vektor

Pont $(x, y, 1)$

Pont $(x, y, z, 1)$

Pont + Vektor = Pont (eltolás)

Pont – Pont = Vektor

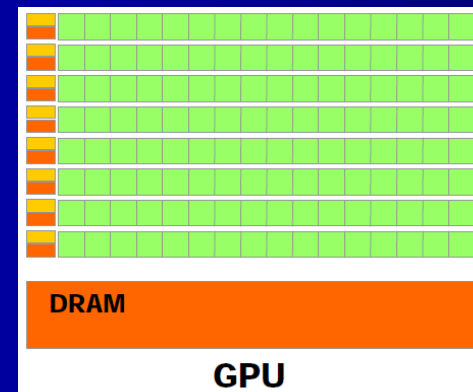
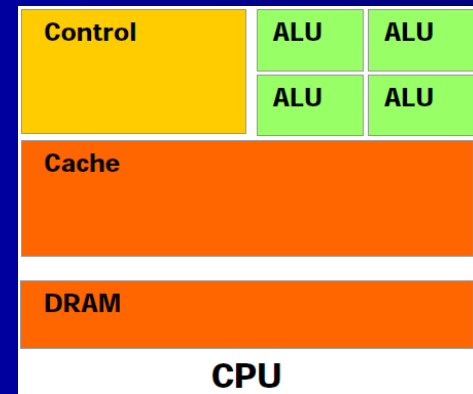
Vektor osztály

```
struct Vector {  
    float x, y, z, w;  
  
    Vector(float x0, float y0, float z0, float w0 = 0) {  
        x = x0; y = y0; z = z0; w = w0;  
    }  
  
    Vector operator*(float a) {  
        return Vector(x * a, y * a, z * a, w * a);  
    }  
  
    Vector operator+(Vector& v) {  
        return Vector(x + v.x, y + v.y, z + v.z, w + v.w);  
    }  
  
    Vector operator-(Vector& v) {  
        return Vector(x - v.x, y - v.y, z - v.z, w - v.w);  
    }  
  
    float operator*(Vector& v) {  
        return (x * v.x + y * v.y + z * v.z);  
    }  
  
    Vector operator%(Vector& v) {  
        return Vector(y*v.z-z*v.y, z*v.x - x*v.z, x*v.y-y*v.x);  
    }  
  
    float Length() { return sqrt(x * x + y * y + z * z); }  
};
```

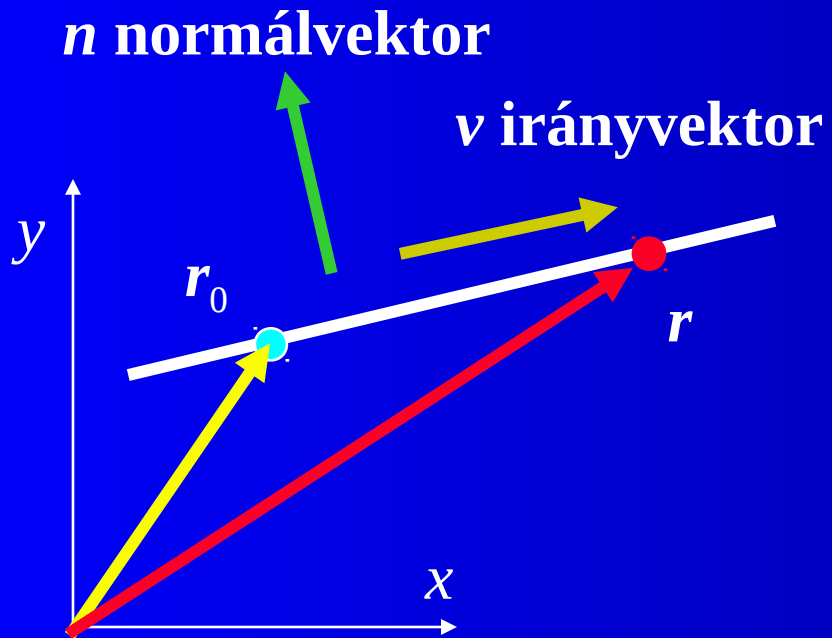
SSE (3DNow!)

```

struct Vector {
    float x, y, z, w;
public:
    Vector operator+( Vector& v ) {
        __declspec(align(16)) Vector res;
        __asm {
            mov     esi, this
            mov     edi, v
            movups   xmm0, [esi]    ; unaligned
            movups   xmm1, [edi]
            addps    xmm0, xmm1
            movaps   res, xmm0      ; aligned
        }
        return res;
    }
};
    
```



2D egyenes



$$n \cdot (r - r_0) = 0$$

$$n_x(x - x_0) + n_y(y - y_0) = 0$$

$$ax + by + c = 0$$

$$(x, y, 1) \cdot (a, b, c) = 0$$

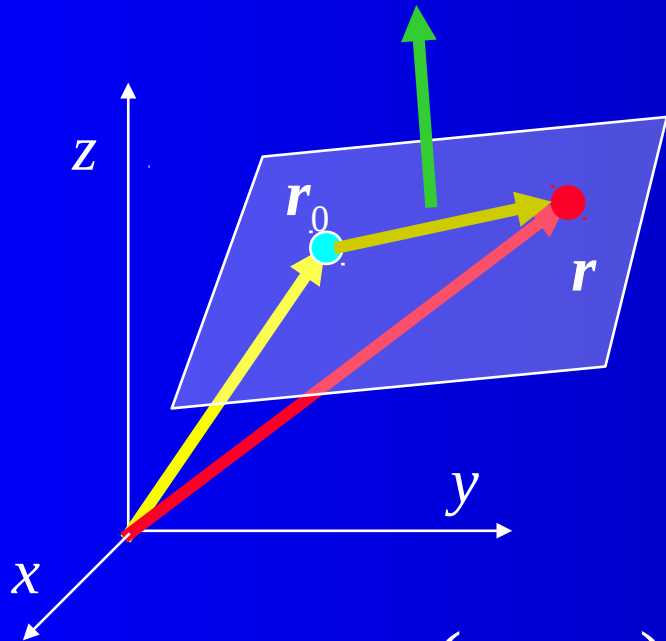
$$r = r_0 + v t, \quad t \in [-\infty, \infty]$$

$$x = x_0 + v_x t$$

$$y = y_0 + v_y t$$

Sík

n normálvektor



$$n \cdot (r - r_0) = 0$$

$$n_x(x - x_0) + n_y(y - y_0) + n_z(z - z_0) = 0$$

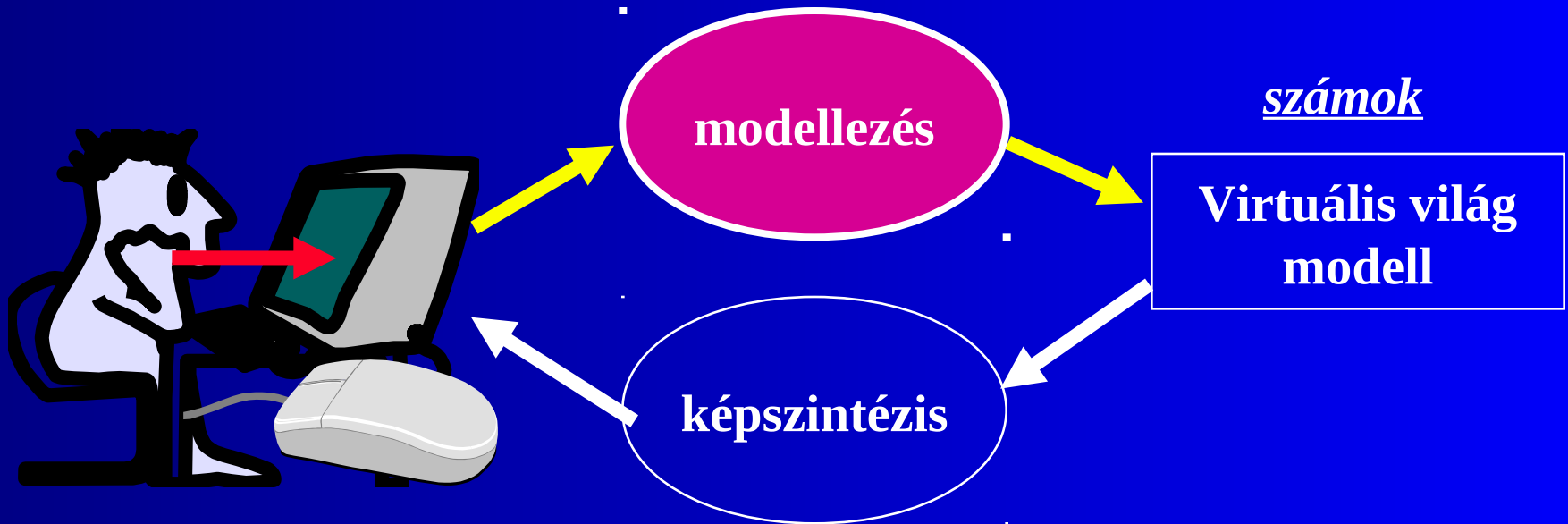
$$ax + by + cz + d = 0$$

$$(x, y, z, 1) \cdot (a, b, c, d) = 0$$

Geometriai modellezés

Szirmay-Kalos László

Számítógépes grafika elemei



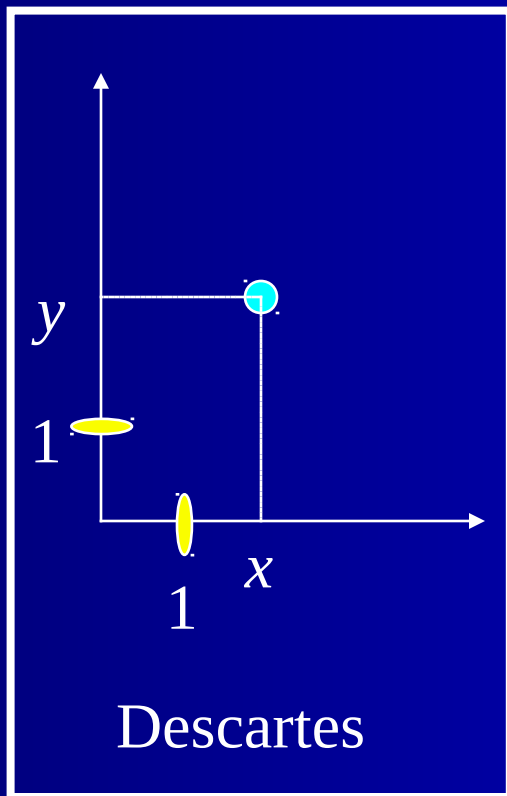
Modellezés feladatai

- Geometria megadása
 - 2D: pont, görbe, terület, fraktálok ($D \leq 2$)
 - 3D: pont, görbe, felület, test, fraktálok ($D \leq 3$)
- Transzformációk (mozgatás)
 - Referencia helyzetből a világkoordináta rendszerben az aktuális helyzetbe (animáció, kényelmes definíció)
- Színek, felületi optikai tulajdonságok

Pontok megadása

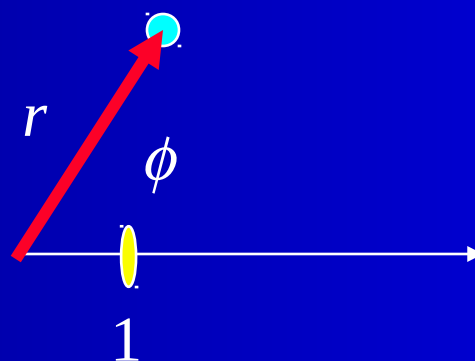
Mindent számmal!

1. Koordináta rendszer
2. Koordináták megadása



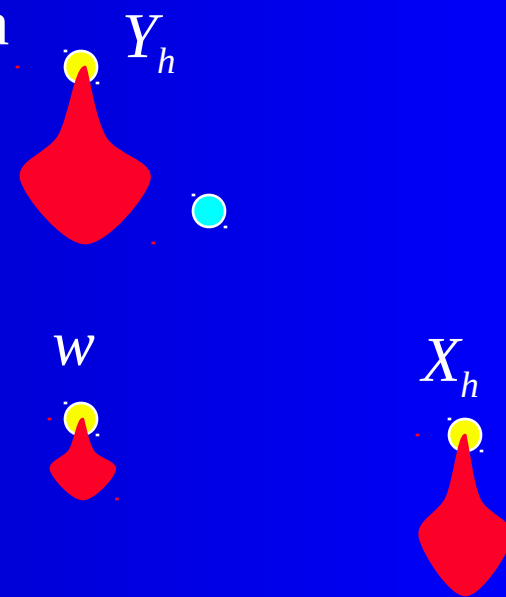
Descartes

eltolás
szemléletes!



Polár

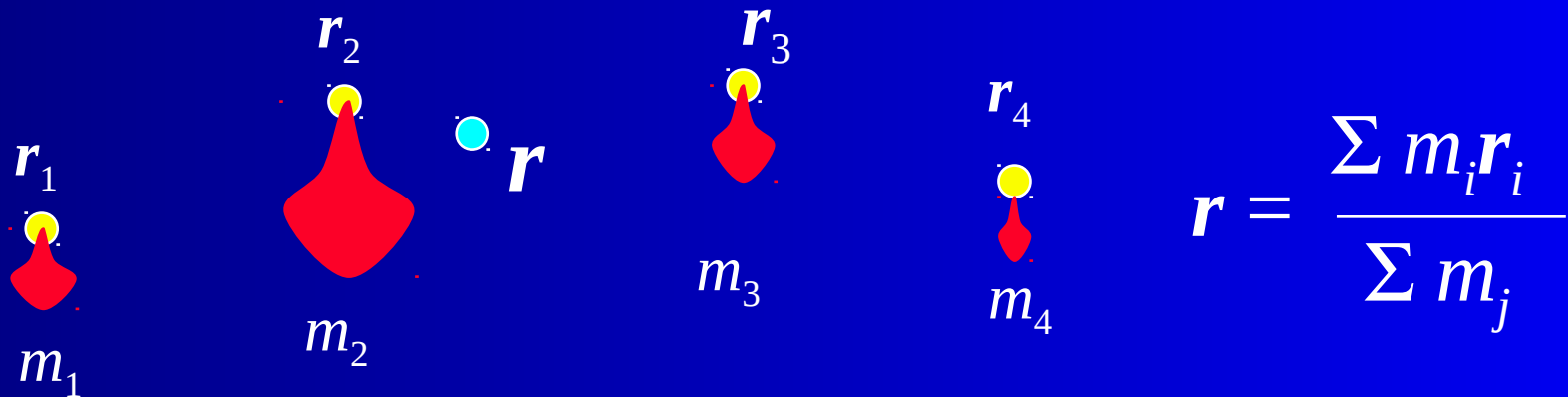
elforgatás



Baricentrikus
Homogén

vetítés

Baricentrikus koordináták



- r az $r_1, r_2, \dots, r_n$ pontok kombinációja
- A tömegek a baricentrikus koordináták
- Ha a tömegek nem negatívak: konvex kombináció
- Konvex kombináció a konvex burok belsejében
- Egyenes (szakasz) = két pont (konvex) kombinációja
- Sík (háromszög) = három pont (konvex) kombinációja

Görbék = pontok halmaza

Koordinátáik kielégítenek egy egyenletet (nem egyértelmű):

– **implicit**: $f(x, y) = 0$

- 2D egyenes: $ax + by + c = 0$

- Kör: $(x-x_0)^2 + (y-y_0)^2 - R^2 = 0$

$$f(\mathbf{r}) = 0$$

$$\mathbf{n} \cdot (\mathbf{r} - \mathbf{r}_0) = 0$$

$$|\mathbf{r} - \mathbf{r}_0|^2 - R^2 = 0$$

– **paraméteres**: $x = x(t), y = y(t)$

- 3D egyenes: $x = x_0 + v_x t$

$$y = y_0 + v_y t$$

- Kör: $t \in [0, 1]$ $z = z_0 + v_z t$

$$x(t) = x_0 + R \cos 2\pi t$$

$$y(t) = y_0 + R \sin 2\pi t$$

$$\mathbf{r} = \mathbf{r}(t)$$

$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v} t, \quad t \in [-\infty, \infty]$$

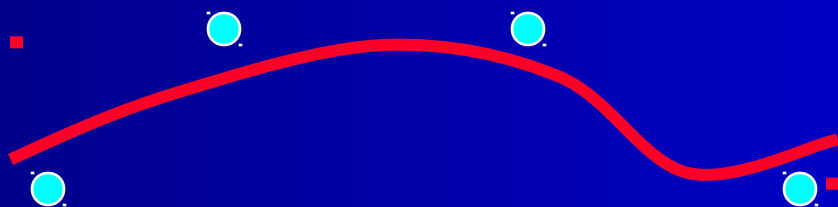
$$\mathbf{r} = \mathbf{r}_0 + R(\cos 2\pi t, \sin 2\pi t)$$

– **explicit**: $y = F(x)$

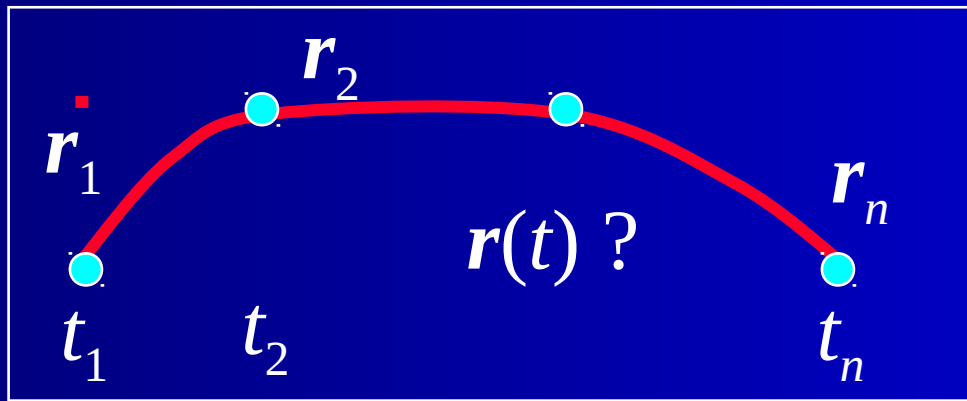
- 2D egyenes: $y = mx + b$

Szabadformájú görbék

- Definíció vezérlőpontokkal



- Polinom: $x(t) = \sum a_i t^i$, $y(t) = \sum b_i t^i$
- A polinomegyütthatók származtatása:
 - Interpoláció
 - Approximáció



Lagrange interpoláció

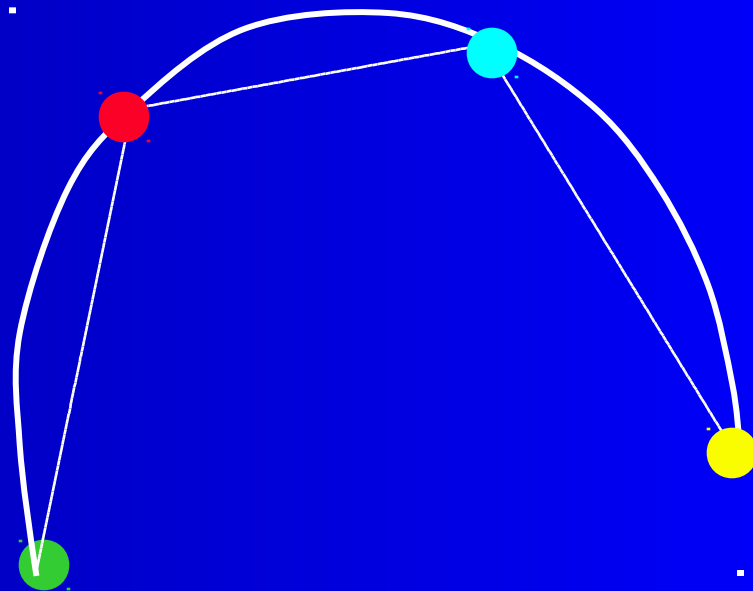
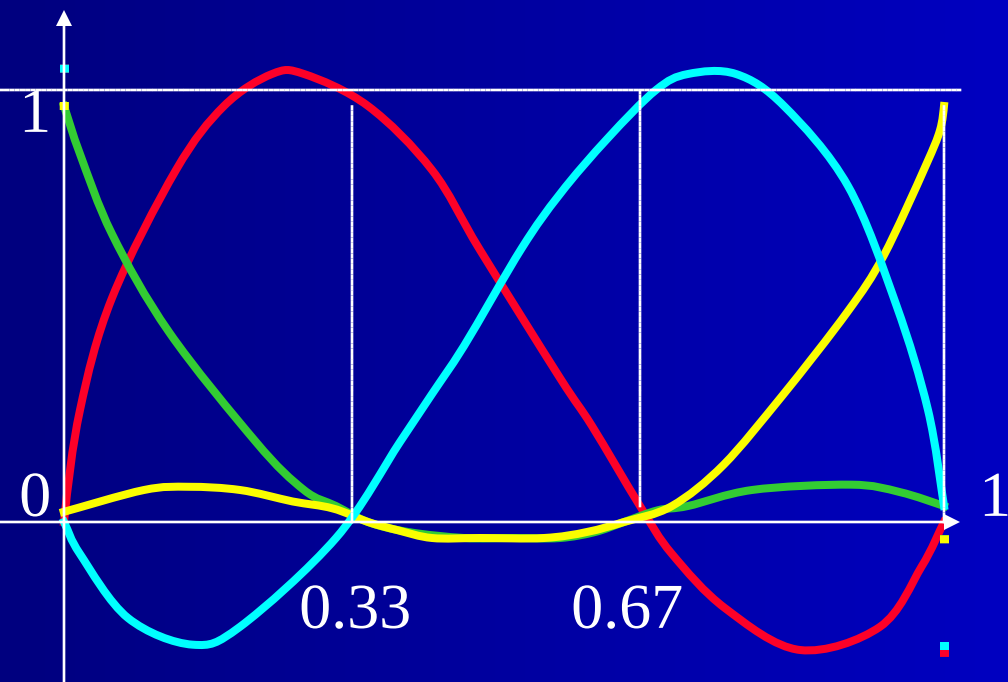
- Vezérlőpontok: $r_1, r_2, r_3, \dots, r_n$
- Keressük azt az $r(t) = \sum [a_i, b_i] t_i$ -t amelyre
 $r(t_1) = r_1, r(t_2) = r_2, \dots, r(t_n) = r_n,$
- Megoldás:

$$- r(t) = \sum L_i(t) r_i$$

$$L_i(t) = \frac{\prod_{j \neq i} (t - t_j)}{\prod_{j \neq i} (t_i - t_j)}$$

$\longrightarrow r(t_k) = r_k$
 $\uparrow$
 $L_i(t_k) = \begin{cases} 1 & \text{ha } i=k \\ 0 & \text{egyébként} \end{cases}$

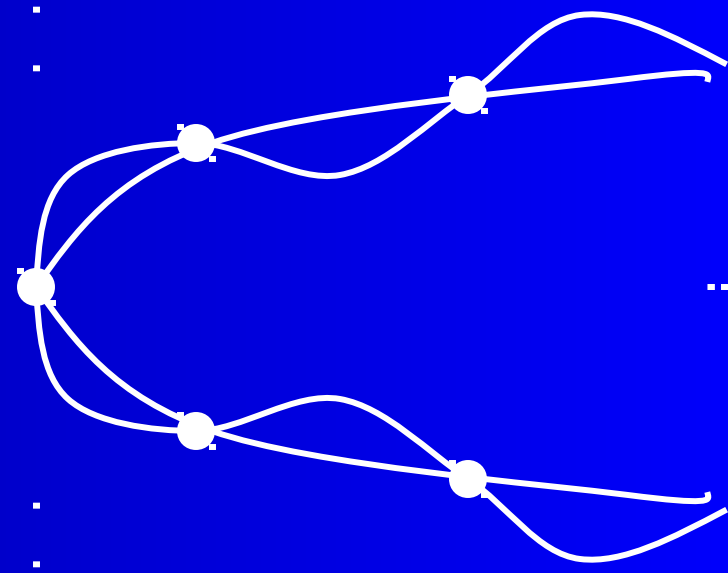
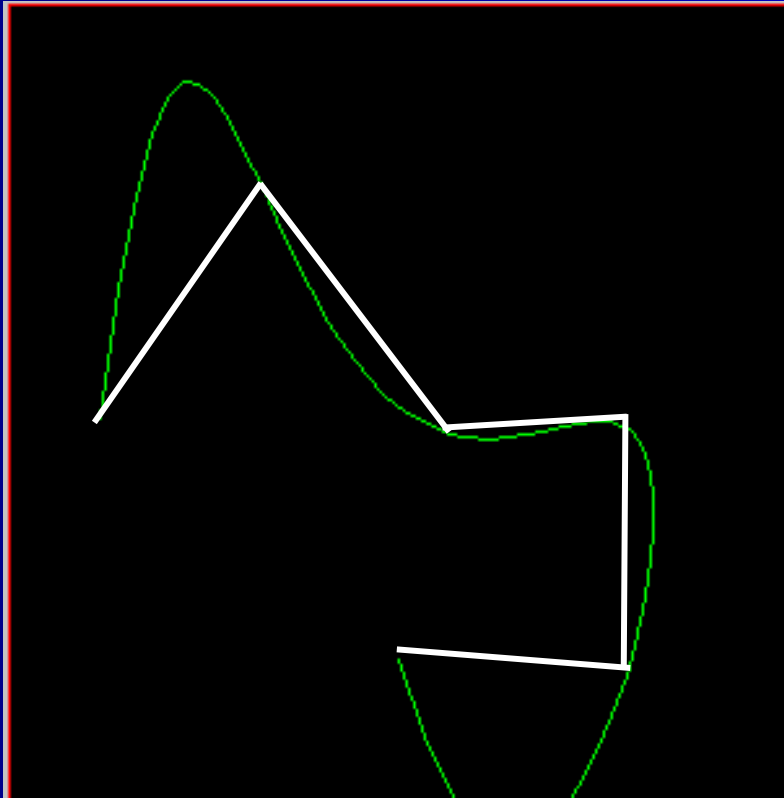
Lagrange interpoláció bázisfüggvényei



$$L_i(t) = \frac{\prod_{j \neq i} (t - t_j)}{\prod_{j \neq i} (t_i - t_j)}$$

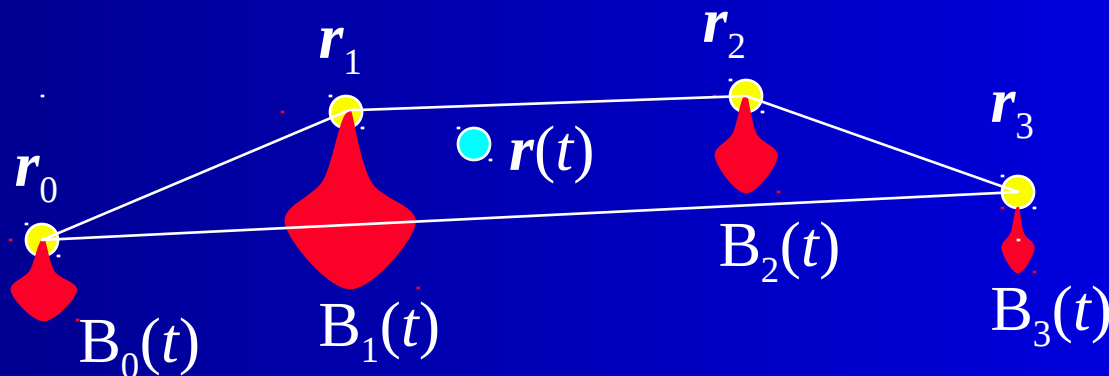
$$r(t) = \sum L_i(t) r_i$$

Görbeszerkesztés Lagrange interpolációval



Bezier approximáció

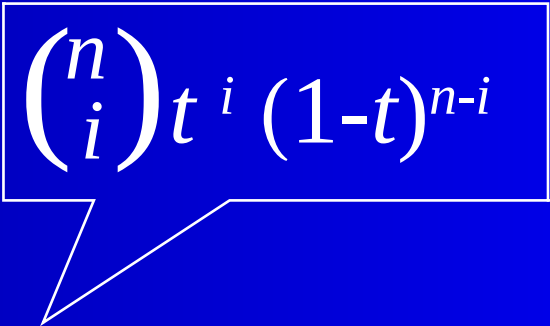
- Keresett görbe: $r(t) = \sum B_i(t) r_i$
 - $B_i(t)$: ne okozzon indokolatlan hullámokat
 - Konvex burok tulajdonság
 - $B_i(t) \geq 0$, $\sum B_i(t) = 1$



Bernstein polinomok

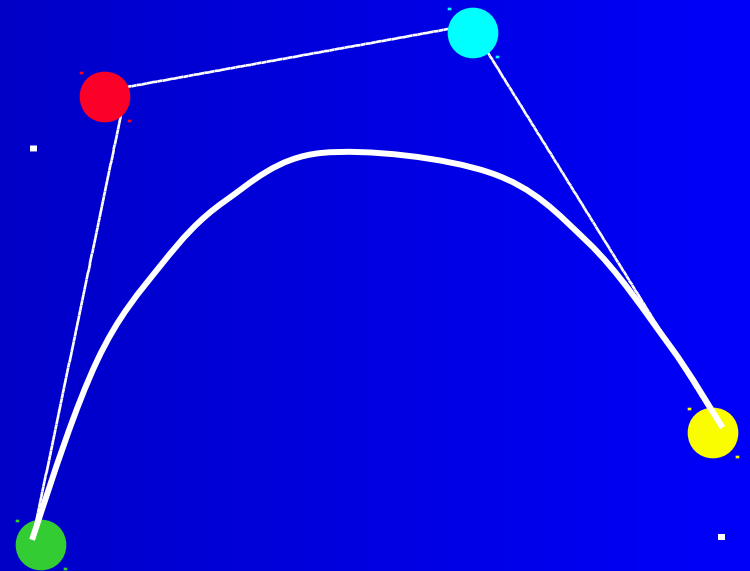
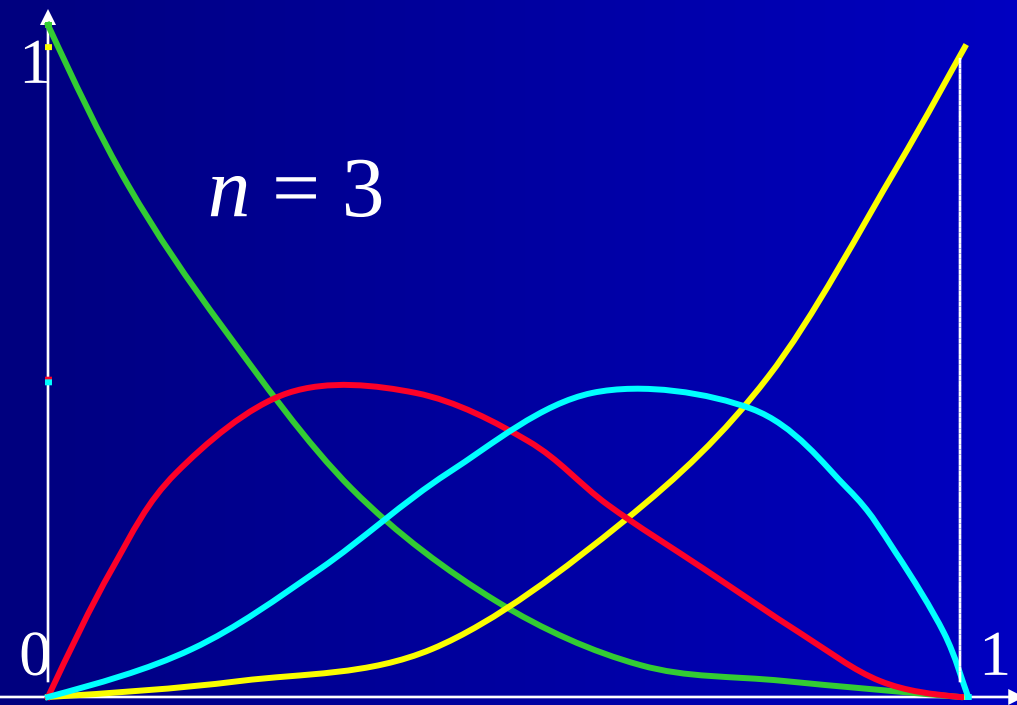
$$1^n = (t + (1-t))^n = \sum_{i=0}^n \boxed{\binom{n}{i} t^i (1-t)^{n-i}}$$

$B_i(t)$



$B_i(t) \geq 0, \sum B_i(t) = 1$ teljesül

Bezier approximáció bázisfüggvényei

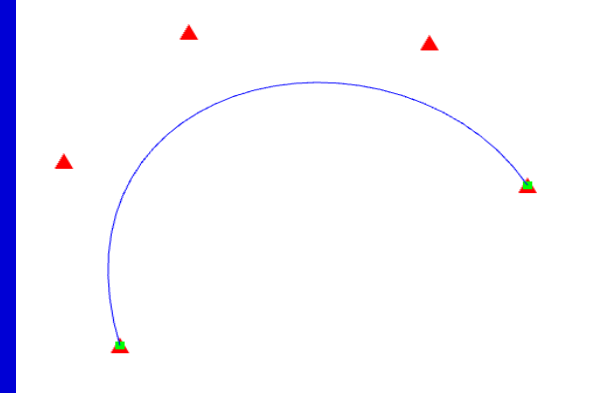


$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

$$\mathbf{r}(t) = \sum B_i(t) \mathbf{r}_i$$

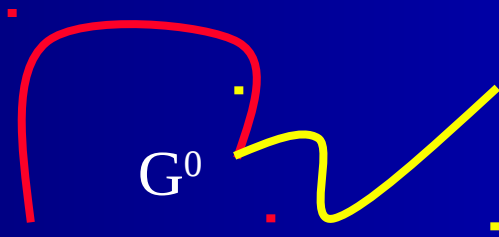
BezierCurve

```
class BezierCurve {  
    Vector * p;  
    int      np; // np-th degree polynomial = np+1 points  
  
    float B(int i, float t) {  
        float choose = 1;  
        for(int j = 1; j <= i; j++)  
            choose *= (float)(np-j+1)/j;  
        return choose * pow(t, i) * pow(1-t, np-i);  
    }  
public:  
    BezierCurve(int n) { np = n; p = new Vector[np+1];}  
  
    Vector r(float t) {  
        Vector rr(0, 0);  
        for(int i = 0; i <= np; i++) rr += p[i] * B(i,t);  
        return rr;  
    }  
};
```



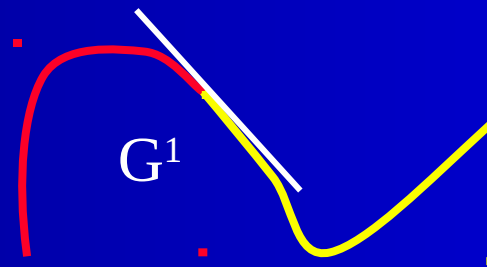
Bonyolult görbék

- Nagyon magas fokszámú polinom
- Összetett görbék:
 - Több alacsony fokszámú + folytonos illesztés
 - folytonossági kategóriák



$$C^0:$$
$$\mathbf{r}_1(t_{\text{veg}}) = \mathbf{r}_2(t_{\text{kezd}})$$

$$G^0 = C^0$$



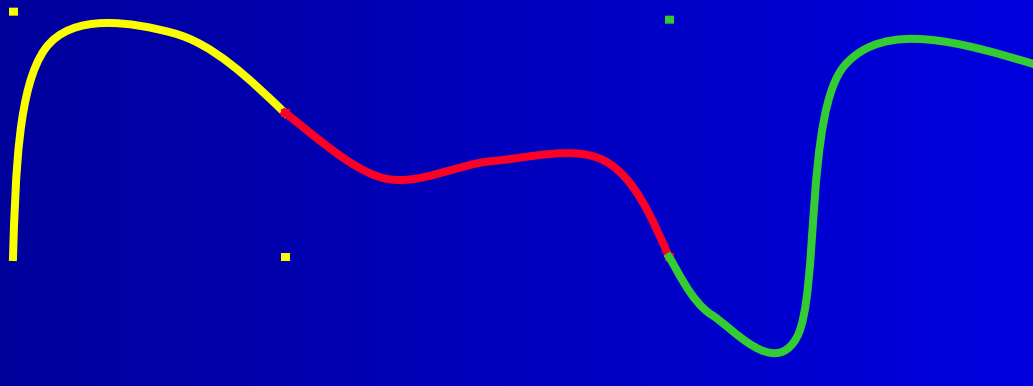
$$C^1:$$
$$\mathbf{r}_1'(t_{\text{veg}}) = \mathbf{r}_2'(t_{\text{kezd}})$$

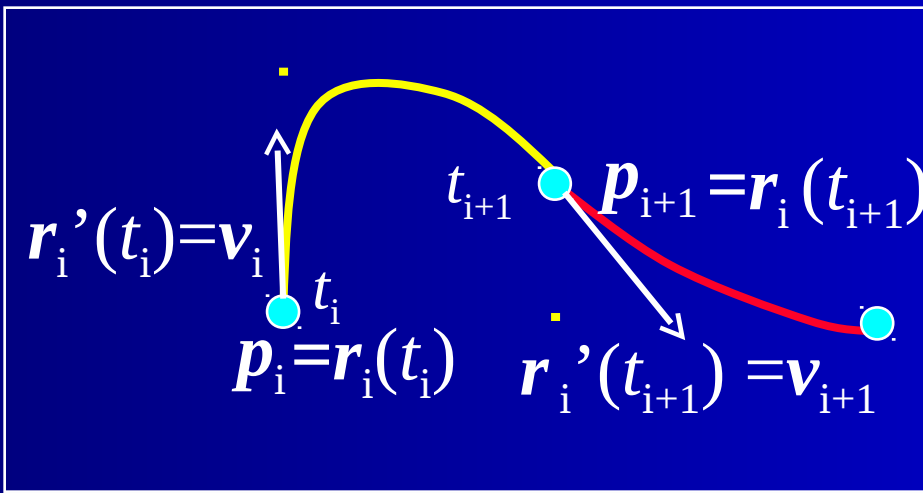
$$C^1 \subseteq G^1$$

$$C^2:$$
$$\mathbf{r}_1''(t_{\text{veg}}) = \mathbf{r}_2''(t_{\text{kezd}})$$

Spline

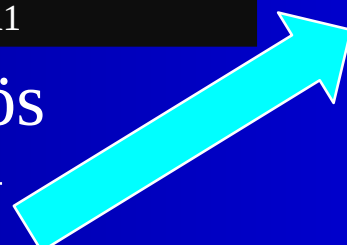
- Spline: C^2 folytonos összetett görbe
 - Harmadfokú spline
 - B-spline





Harmadfokú spline

- $r_i(t) = a_{i3}(t-t_i)^3 + a_{i2}(t-t_i)^2 + a_{i1}(t-t_i) + a_{i0}$
- $r_i'(t) = 3a_{i3}(t-t_i)^2 + 2a_{i2}(t-t_i) + a_{i1}$
- C^1 folytonosság: v_i közös



$$\begin{aligned} a_{i0} &= p_i \\ a_{i1} &= v_i \\ a_{i2} &= \frac{3(p_{i+1} - p_i)}{(t_{i+1} - t_i)^2} - \frac{v_{i+1} + 2v_i}{t_{i+1} - t_i} \\ a_{i3} &= \frac{2(p_i - p_{i+1})}{(t_{i+1} - t_i)^3} + \frac{v_{i+1} - v_i}{(t_{i+1} - t_i)^2} \end{aligned}$$

$$r_i(t_i) = a_{i0} = p_i$$

$$r_i(t_{i+1}) = a_{i3}(t_{i+1}-t_i)^3 + a_{i2}(t_{i+1}-t_i)^2 + a_{i1}(t_{i+1}-t_i) + a_{i0} = p_{i+1}$$

$$r_i'(t_i) = a_{i1} = v_i$$

$$r_i'(t_{i+1}) = 3a_{i3}(t_{i+1}-t_i)^2 + 2a_{i2}(t_{i+1}-t_i) + a_{i1} = v_{i+1}$$

Harmadfokú spline C^2 folytonosság

- Követelmény: $\mathbf{r}_i''(t_{i+1}) = \mathbf{r}_{i+1}''(t_{i+1})$

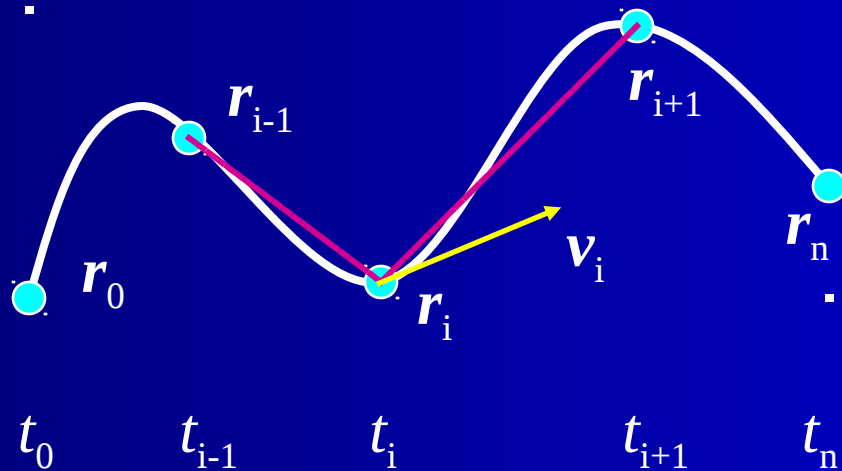
- Második derivált: $\mathbf{r}_i''(t) = 6\mathbf{a}_{i3}(t-t_i)^2 + 2\mathbf{a}_{i2}$

$$\mathbf{r}_i''(t_{i+1}) = 6\mathbf{a}_{i3}(t_{i+1} - t_i) + 2\mathbf{a}_{i2} = \dots \mathbf{v}_i \dots \mathbf{v}_{i+1} \dots$$

$$\mathbf{r}_{i+1}''(t_{i+1}) = 2\mathbf{a}_{(i+1),2} = \dots \mathbf{v}_{i+1} \dots \mathbf{v}_{i+2} \dots$$

- C^2 folytonosság az ismeretlen közös első derivált kiszámításával

Kochanek-Bartels (Catmull-Rom) „spline”

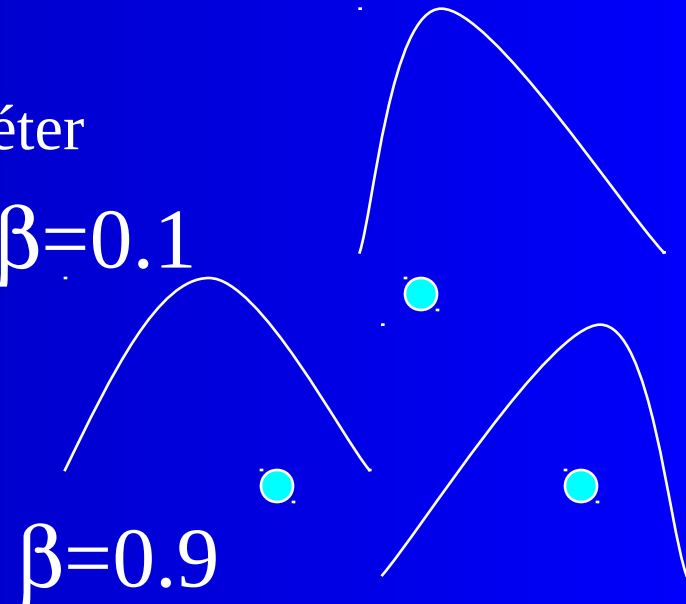
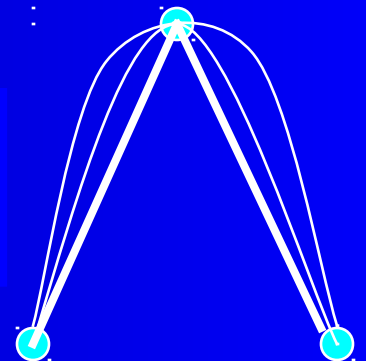


$\tau = -1$
Catmull-Rom

tenzió
paraméter

$$\mathbf{v}_i = \frac{1}{2} \left[\frac{\mathbf{r}_{i+1} - \mathbf{r}_i}{t_{i+1} - t_i} + \frac{\mathbf{r}_i - \mathbf{r}_{i-1}}{t_i - t_{i-1}} \right] (1 - \tau)$$

β $(1-\beta)$ $\beta=0.5$

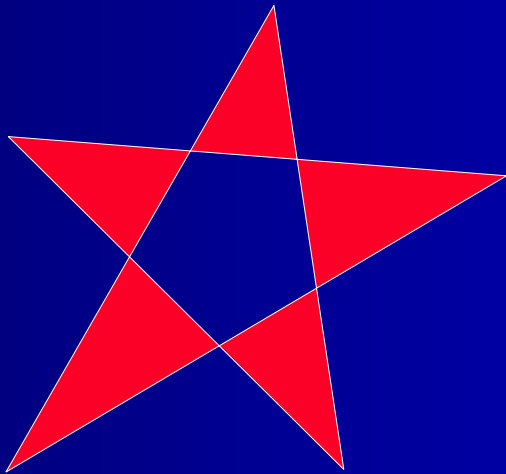




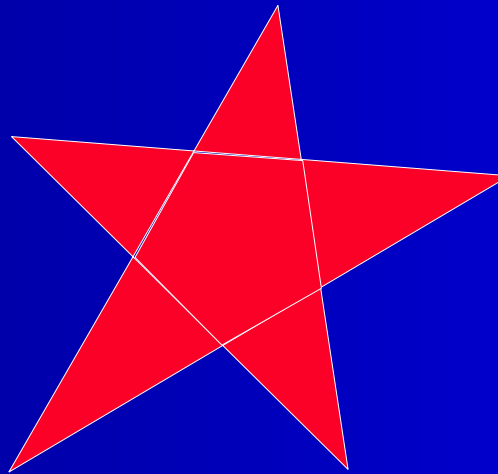
Területek

Határ + belső tartományok azonosítása

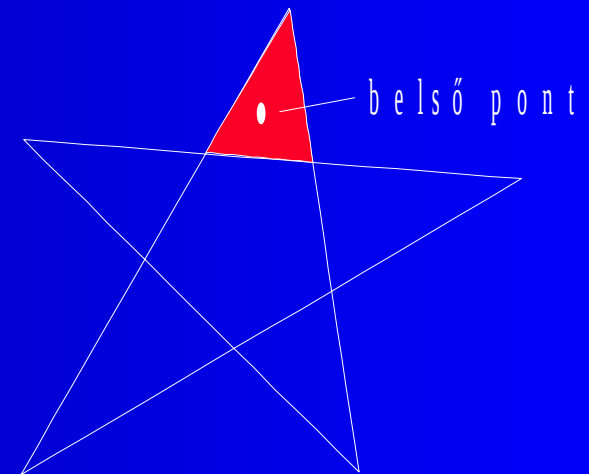
Belső tartományok:



1)



2)



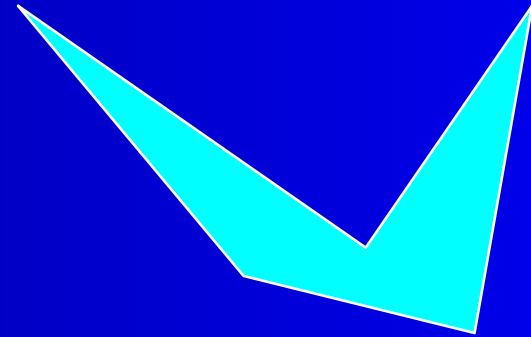
3)

GL_POLYGON

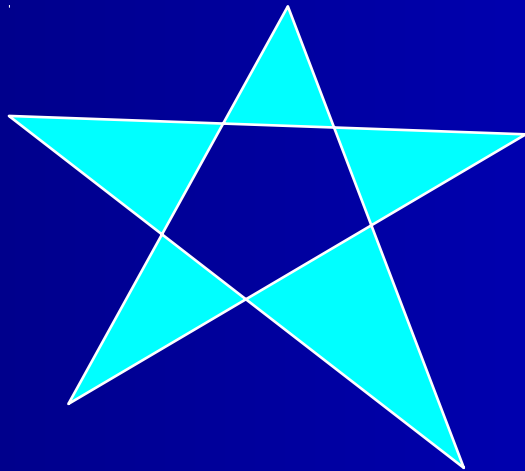


Egyszerű konvex

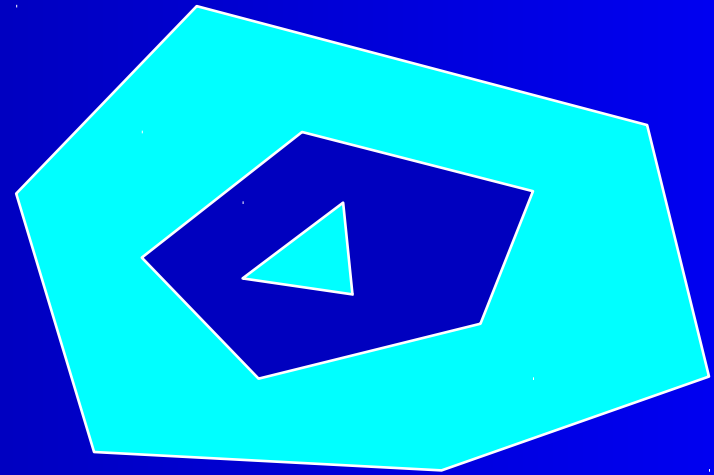
Sokszögek: poligon



Egyszerű konkáv



Nem egyszerű,
egyszeresen összefüggő



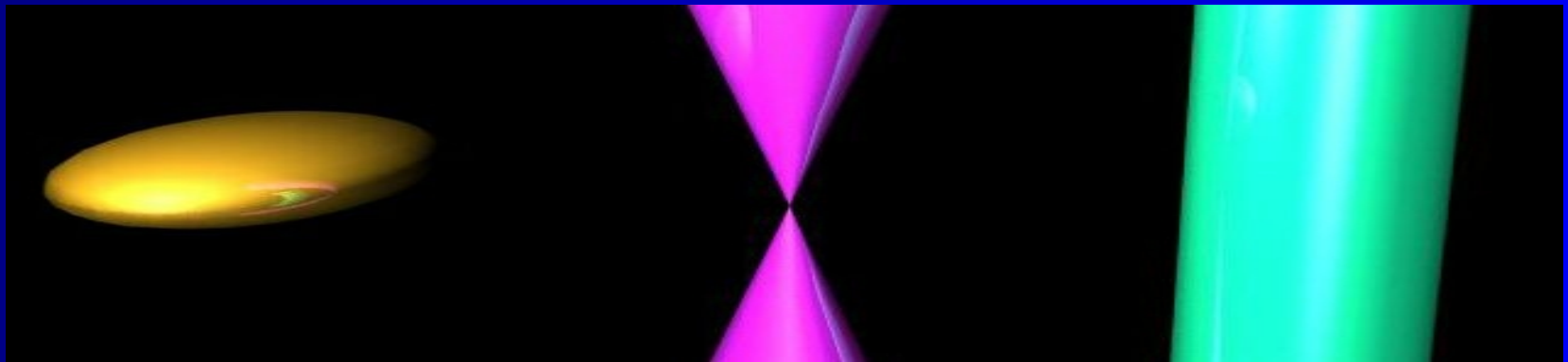
Nem egyszerű,
többszörösen összefüggő

Felületek

- Felület 3D pontok halmaza:
 - koordinátáik kielégítenek egy egyenletet
 - implicit: $f(x, y, z) = 0$
 - gömb: $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - R^2 = 0$
 - paraméteres: $x = x(u, v), y = y(u, v), z = z(u, v),$
 $u, v \in [0, 1]$
 - gömb: $x = x_0 + R \cos 2\pi u \sin \pi v$
 $y = y_0 + R \sin 2\pi u \sin \pi v$
 $z = z_0 + R \cos \pi v \quad u, v \in [0, 1]$
- Klasszikus felületek
 - definíció = paraméterek megadása

Kvadratikus felületek

- $\underline{\mathbf{x}}^T \mathbf{A} \underline{\mathbf{x}} = 0$ $\underline{\mathbf{x}}^T = [x, y, z, 1]$
- A koordináták legfeljebb másodfokon
- gömb, ellipszoid, sík, paraboloid, hiperboloid, hengerfelület,...



Ellipszoid

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1 = 0$$

Végtelen kúp

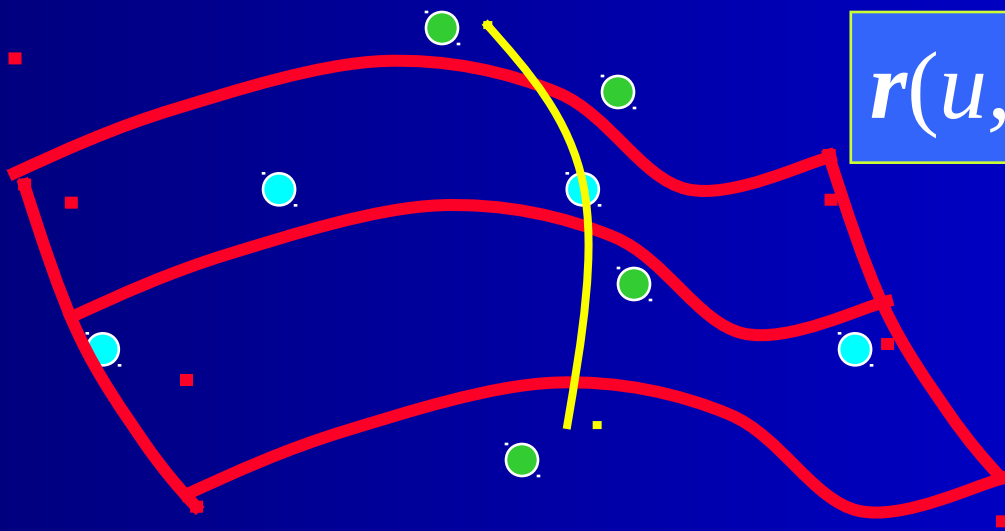
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - z^2 = 0$$

Végtelen henger

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$$

Szabadformájú felületek: $r(u,v)$

- Definíció kontrolpontokkal



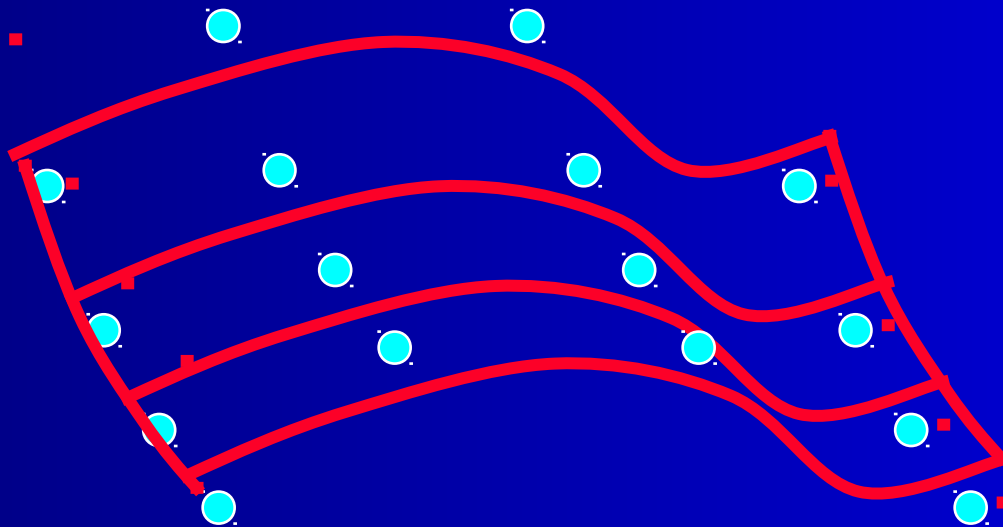
$$r(u,v) = \sum \sum B_i(u) B_j(v) r_{i,j}$$

$$r(u,v) = r_v(u) = \sum B_i(u) r_i(v)$$

$$r_i(v) = \sum B_j(v) r_{i,j}$$

Szabadformájú felületek

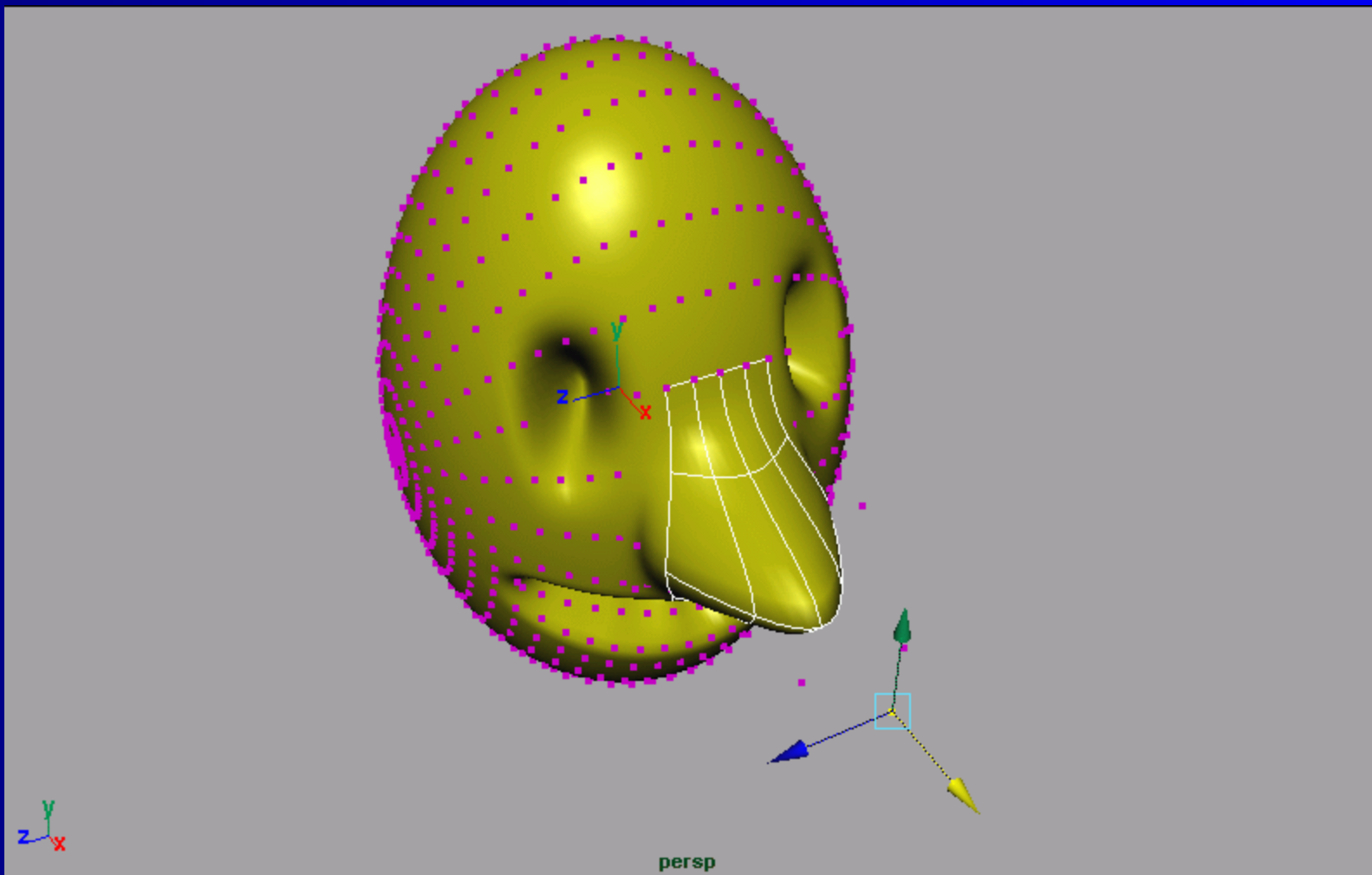
- 2D kontrolpont sereg: szorzatfelületek



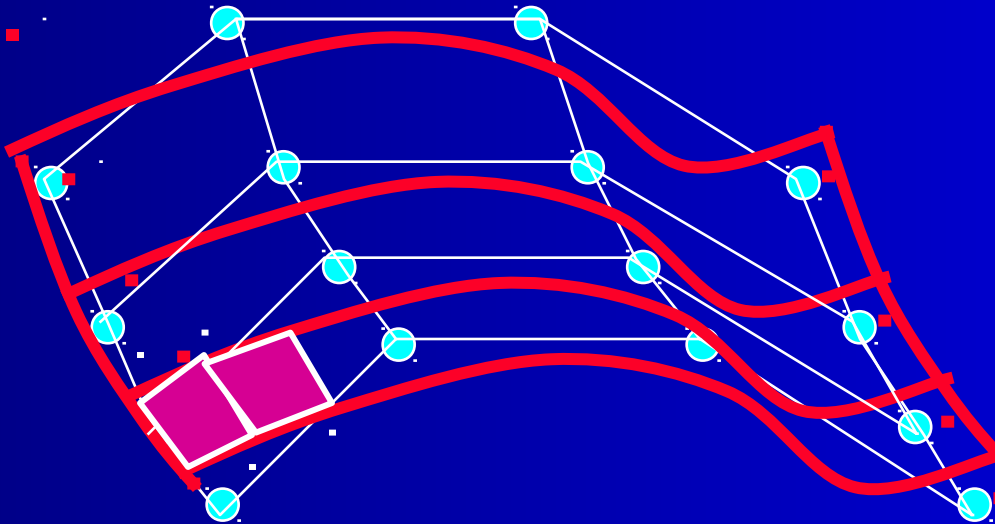
Súlyfüggvények:
Interpoláció,
Approximáció

$$\begin{aligned} \mathbf{r}(u,v) &= \sum \sum B_{i,j}(u,v) \mathbf{r}_{i,j} \\ &= \sum \sum B_i(u) B_j(v) \mathbf{r}_{i,j} \end{aligned}$$

Vezérlőpontok, súlyok módosítása

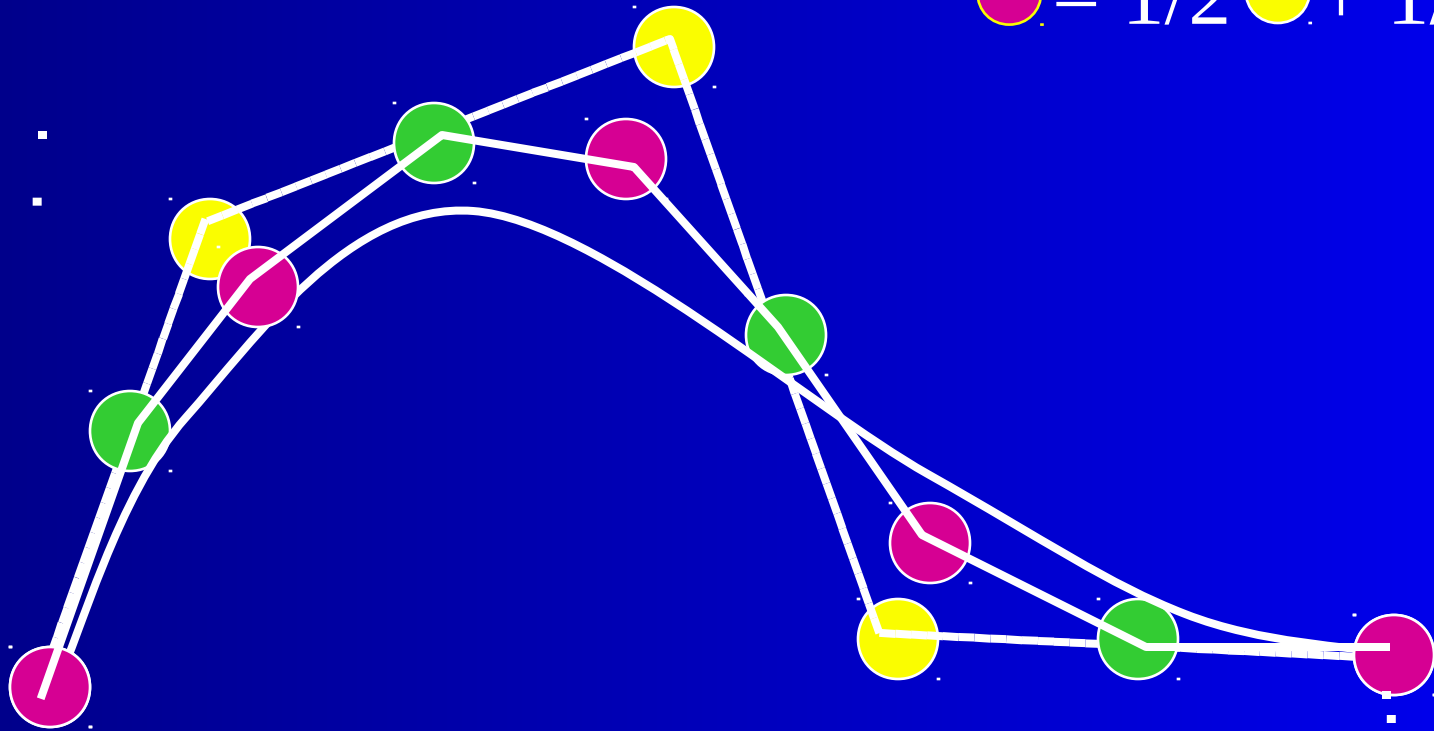


Felületmodellezés mint poligonháló módosítás



Felosztásos (subdivision) módszerek

$$\text{pink circle} = 1/2 \text{ yellow circle} + 1/4 \sum \text{green circles}$$



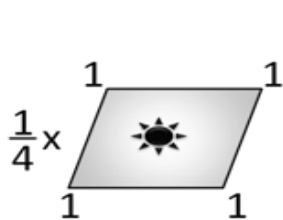
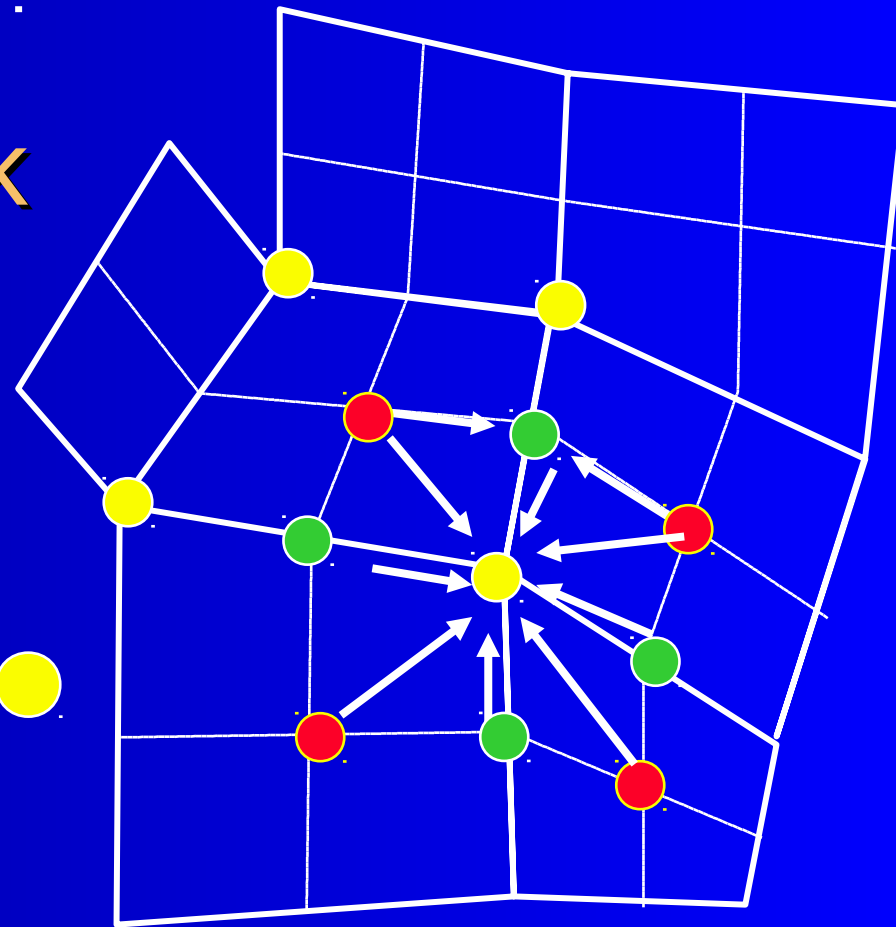
Subdivision felületek (Catmull-Clark)

$$\text{Red Circle} = 1/4 \Sigma \text{ Yellow Circle}$$

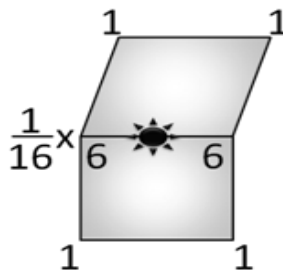
$$\text{Green } i = 1/2 \Sigma \text{ Yellow Circle}$$

$$\text{Yellow Circle} = 1/v^2 \Sigma \text{ Red Circle} + 2/v^2 \Sigma \text{ Green } i + (v-3)/v \text{ Yellow Circle}$$

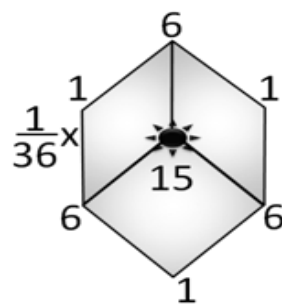
$$\text{Green Circle} = 1/4 \Sigma \text{ Red Circle} + 1/2 \text{ Green } i$$



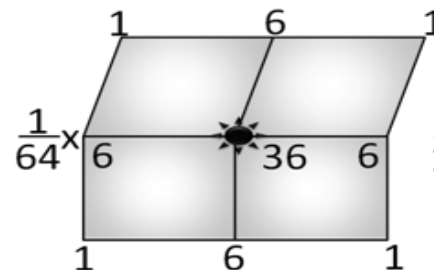
Face Point



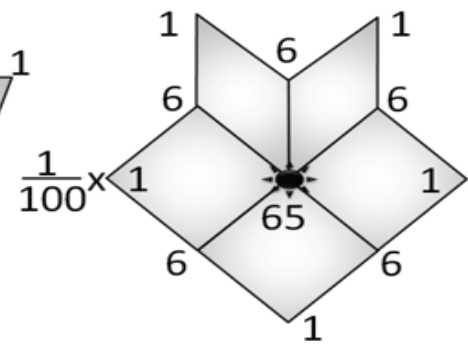
Edge Point



Valence 3 Vertex

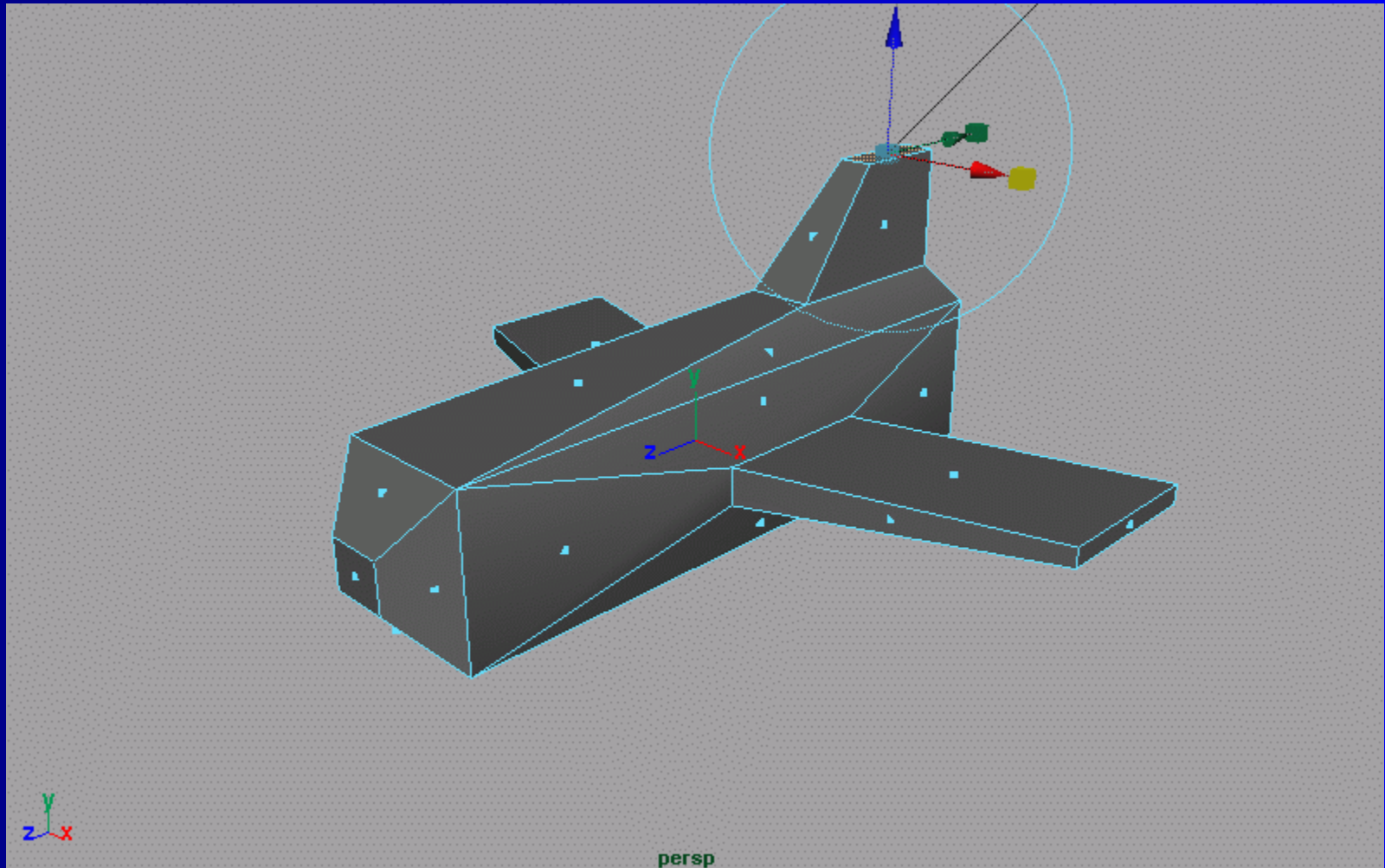


Valence 4 Vertex

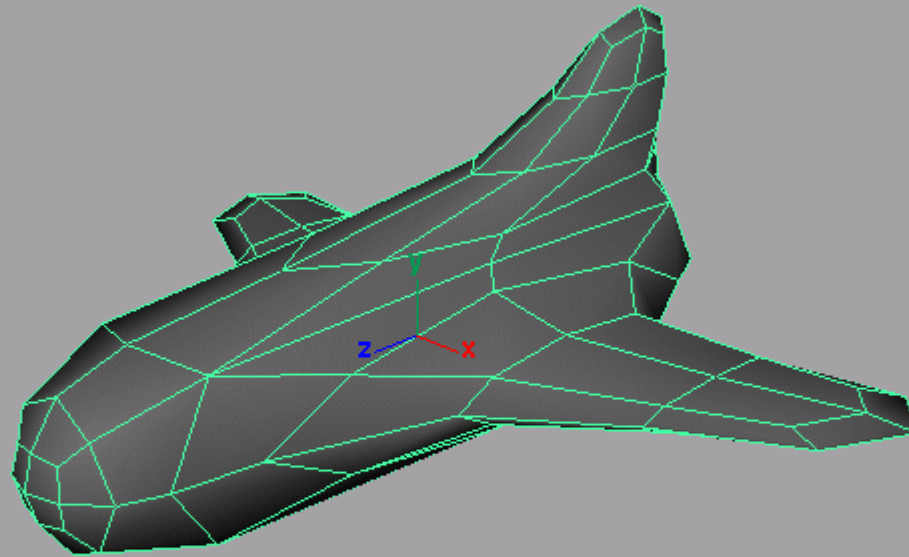


Valence 5 Vertex

Durva poligon modell

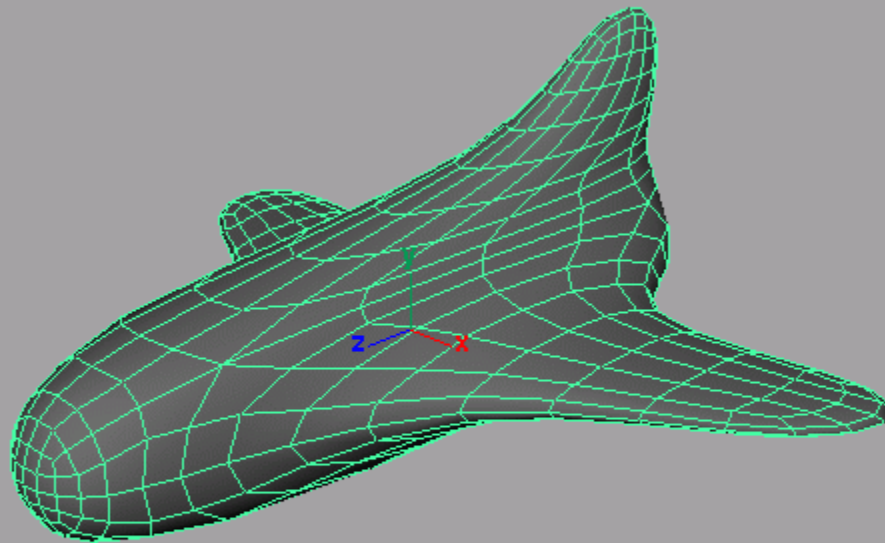


Subdivision simítás: 1 szint



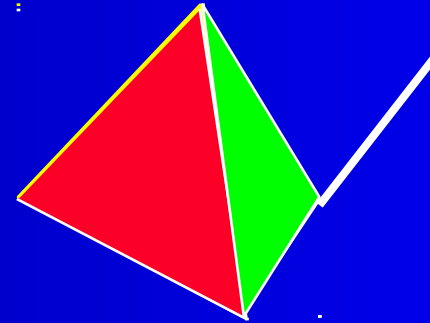
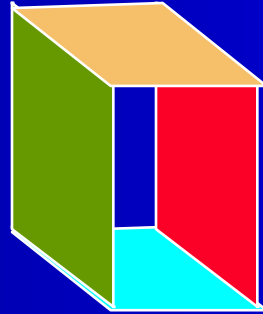
persp

Subdivision simítás: 2. szint



persp

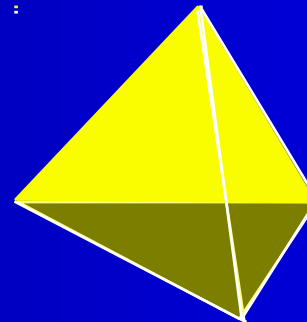
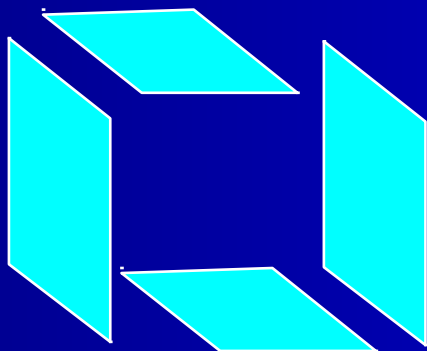
Testek



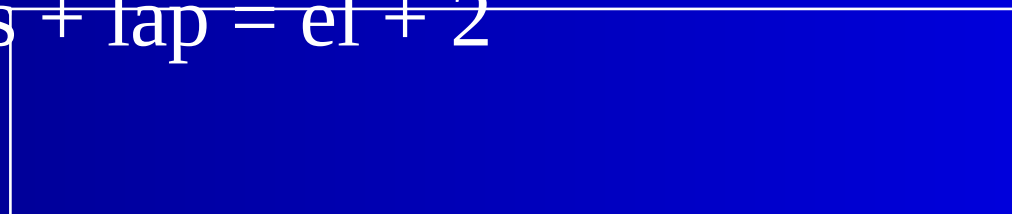
- Ellenpéldák
- Érvényes testek: reguláris halmaz
 - nem lehetnek alacsony dimenziós elfajulásai
 - minden határpont környezetében van belső pont
- Garantáltan érvényes testet építő módszerek
 - 2.5 dimenziós eljárások
 - speciális felületi modellezés: B-rep
 - Konstruktív tömörtest geometria

Felületmodellezők

- Test = határfelületek gyűjteménye

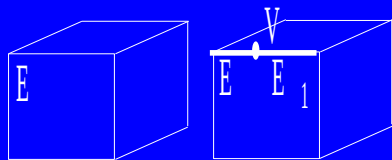


- Topológiai ellenőrzés (Euler tétel):
 $\text{csúcs} + \text{lap} = \text{él} + 2$



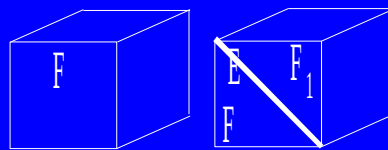
B-rep: Euler operátorok

M V E



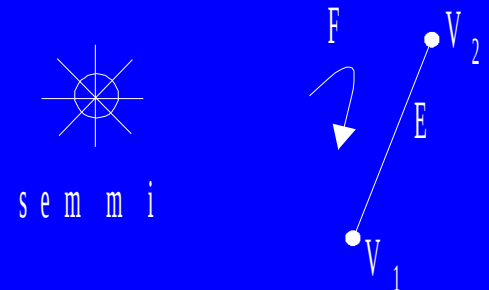
Csinálj csúcsot
és élt

M E F



Csinálj élt
és lapot

M E V V F

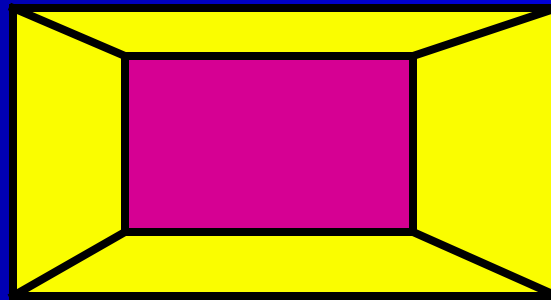


semmi

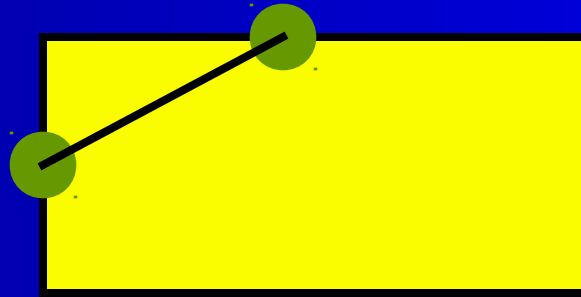
Csinálj élt,
két csúcsot
és lapot

Gyakorlati Euler operátorok

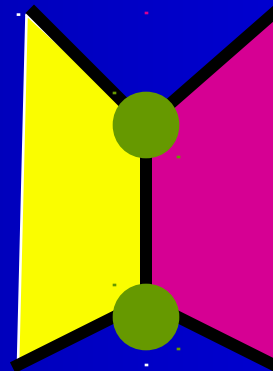
- Face extrude



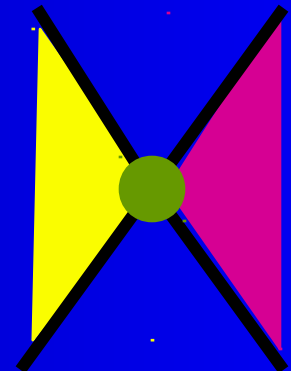
- Face split



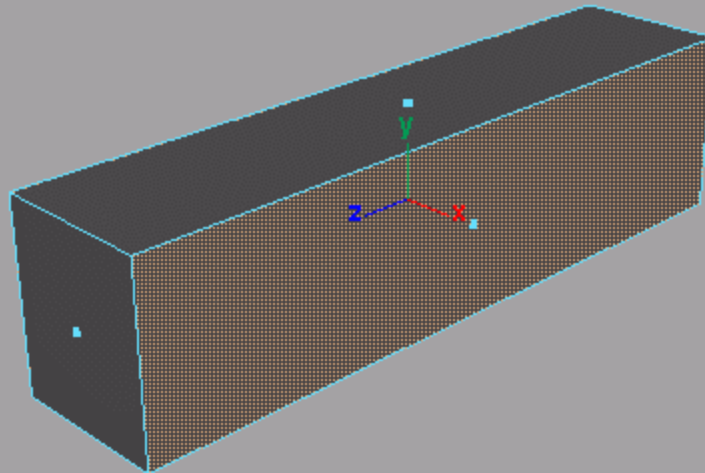
- Edge Collapse



- Vertex split



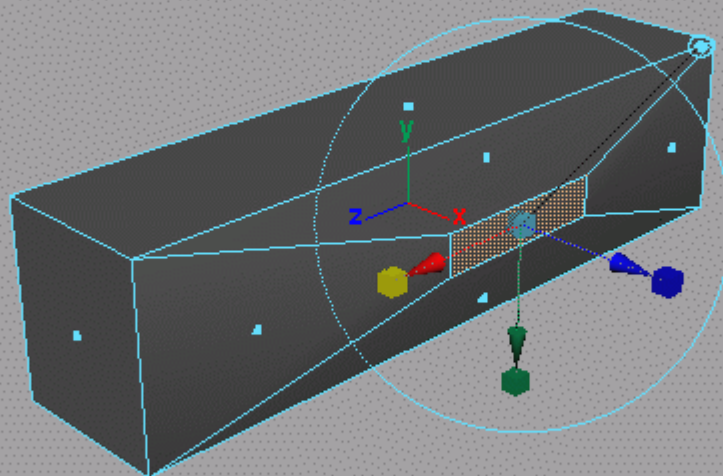
Poligon modellezés: téglatest



persp

Poligon modellezés:

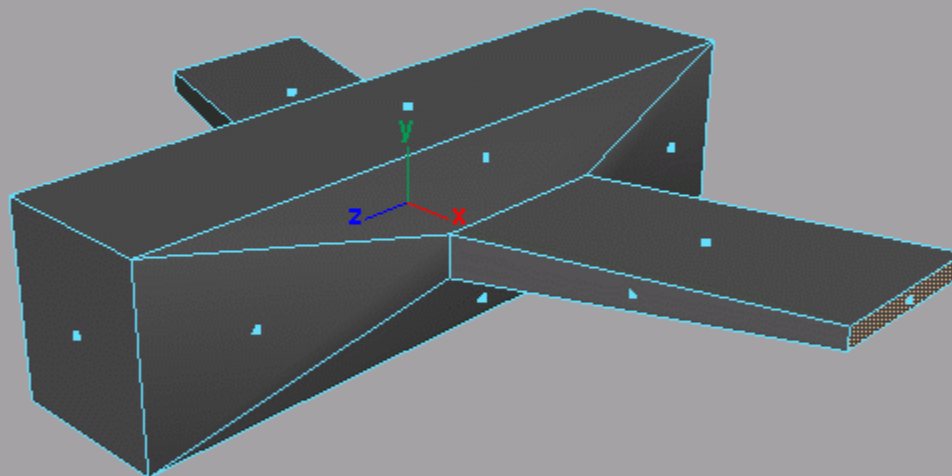
1. extruding



persp

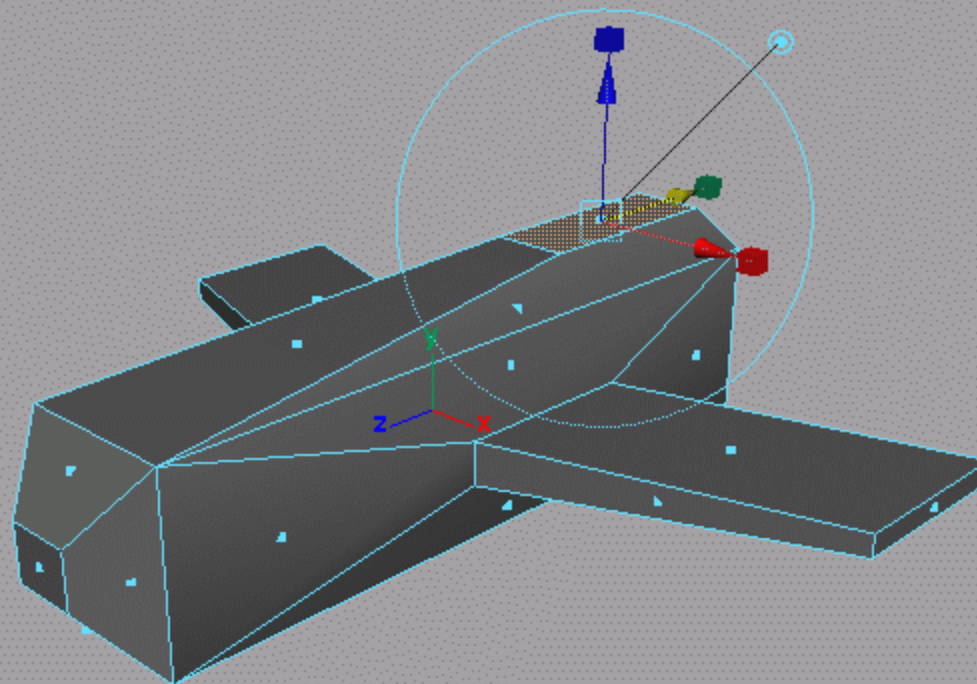
Poligon modellezés:

2. extruding



persp

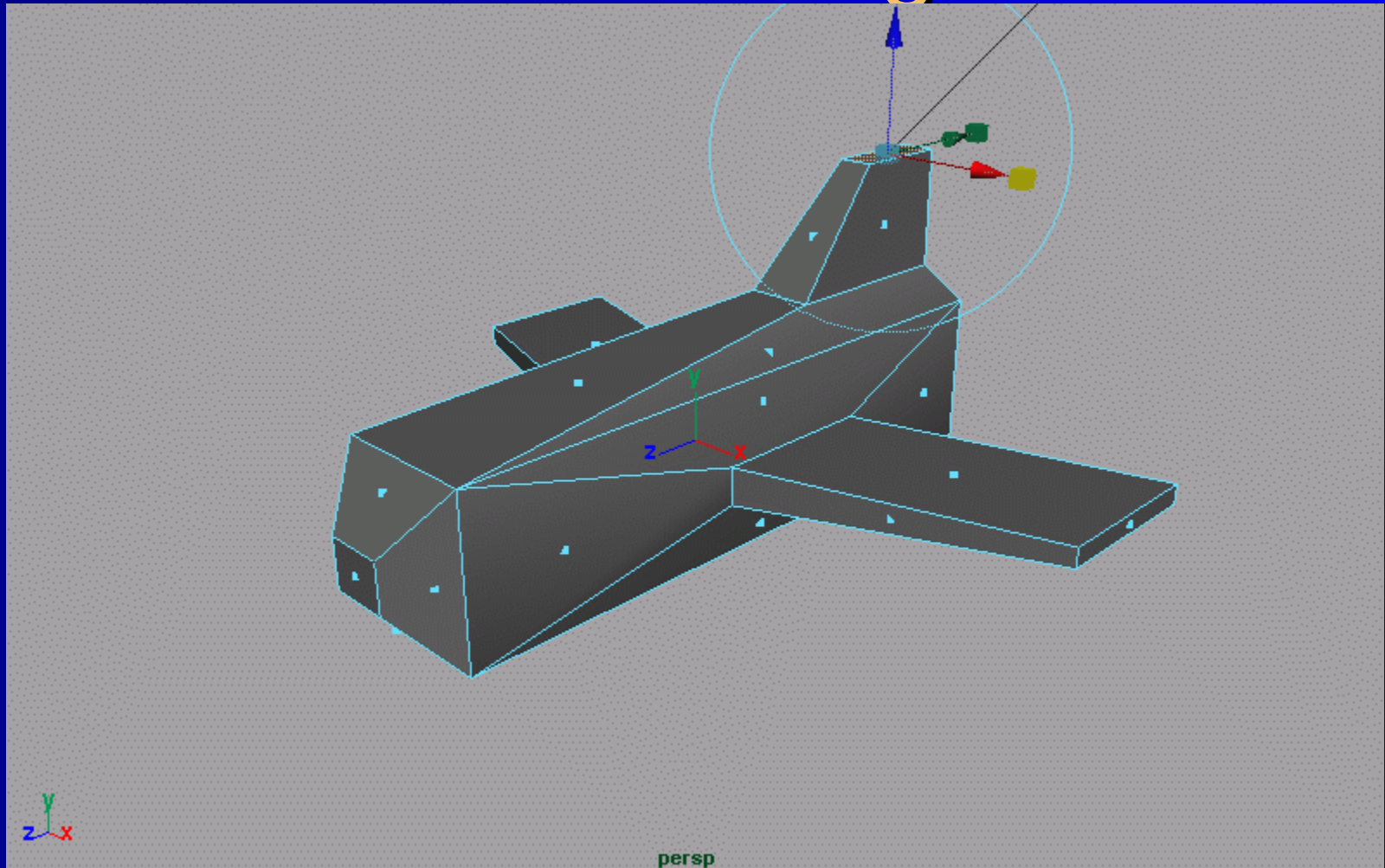
Poligon modellezés: 4. és 5. extruding



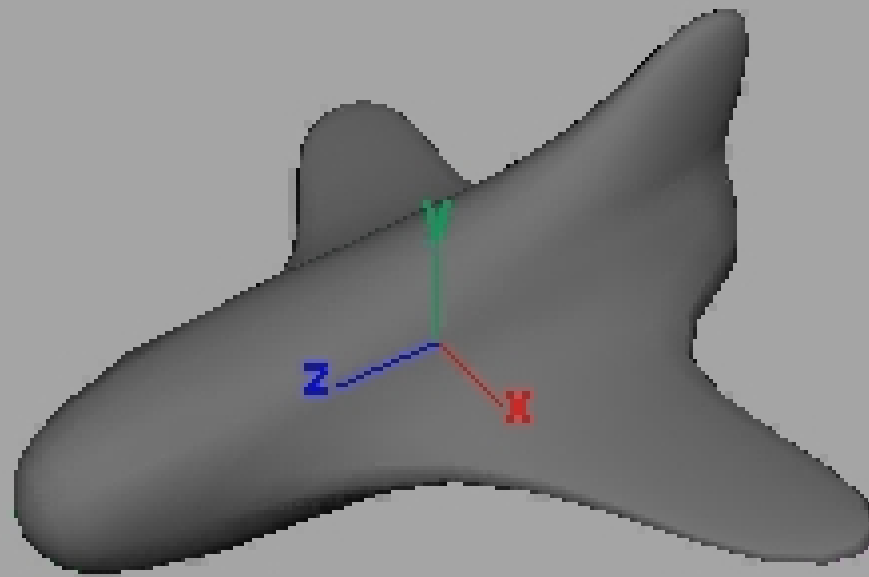
persp

Poligon modellezés:

6. extruding

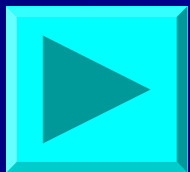


Subdivision simítás



persp

Poligon modellezés



Színek

Szirmay-Kalos László

Színelméleti alapok

- Fény - Színérzet

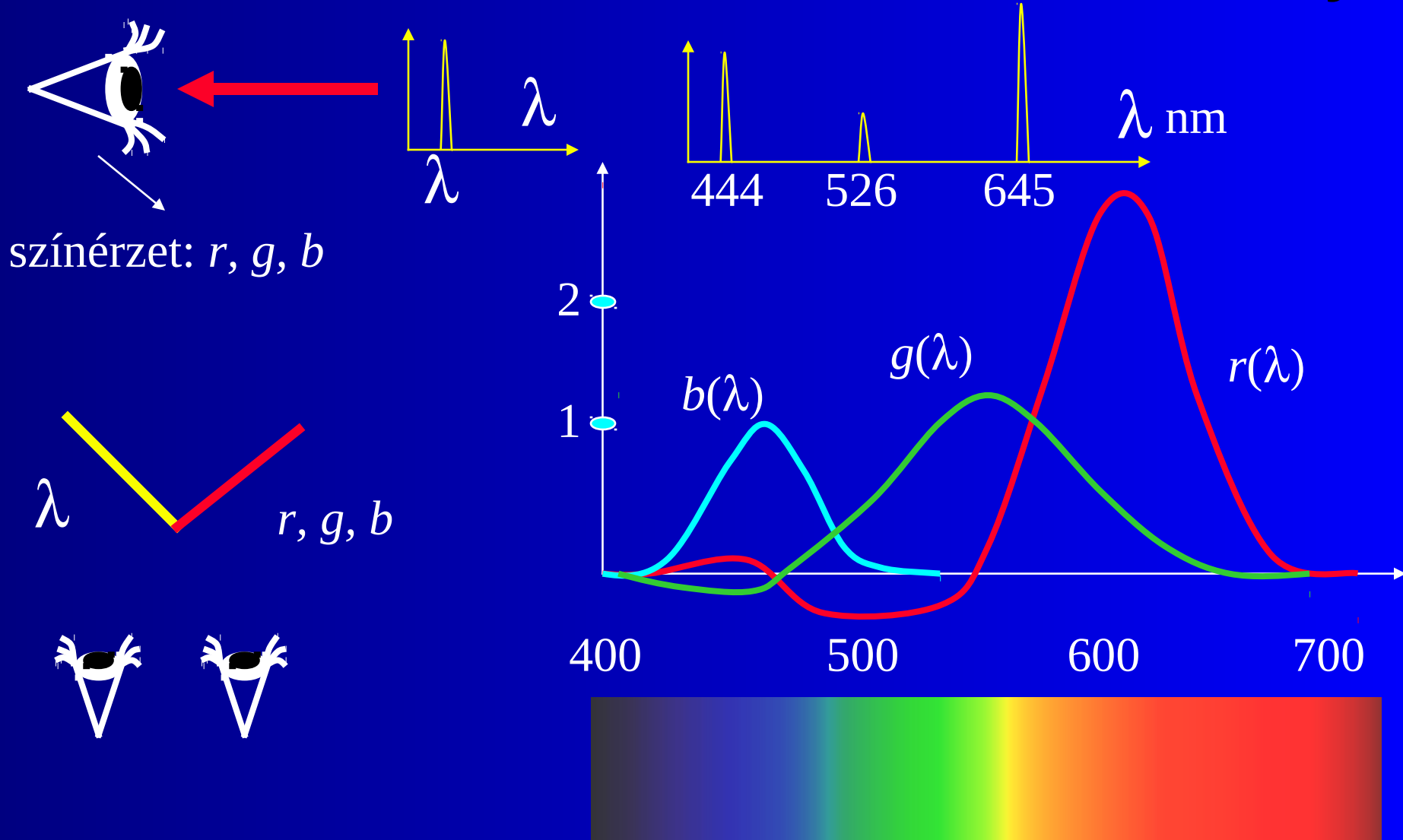
- elektromágneses hullám
- érzéklés

- Amplitúdó + hullámhossz:

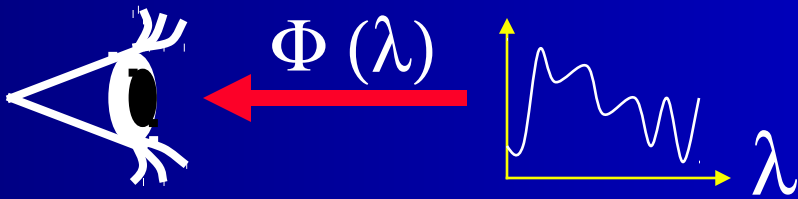
- Energia spektrum 3 érzékelővel: tristimulus értékek

- Fázist, polarizáltságot nem

Színérzékelés: monokromatikus fény



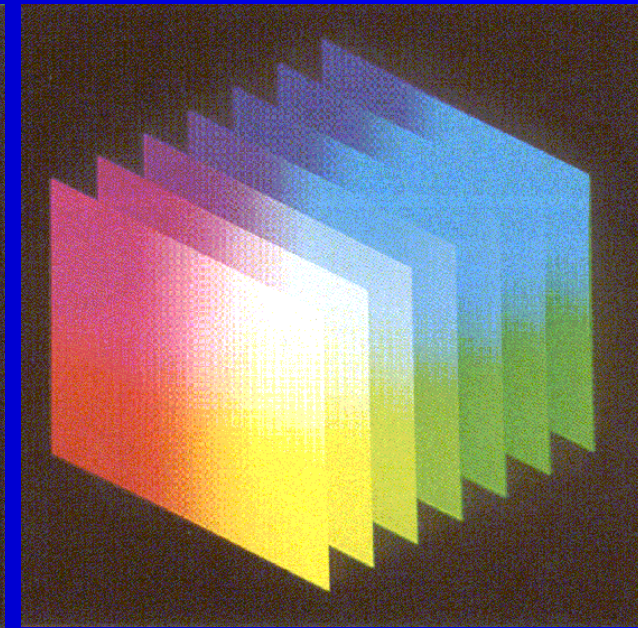
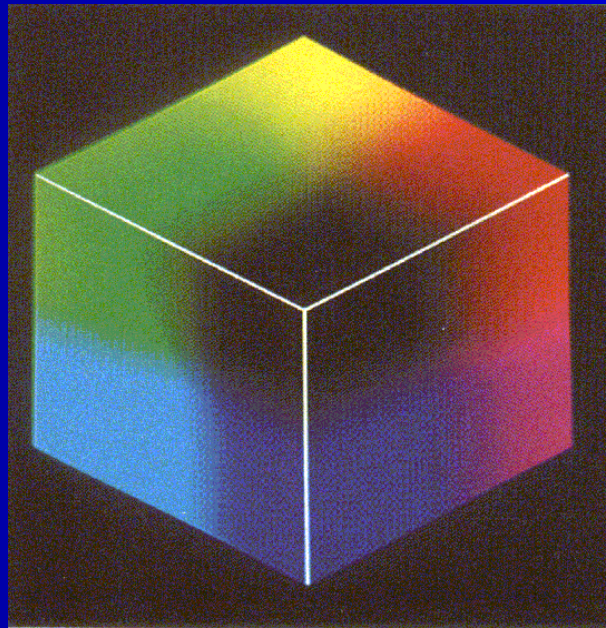
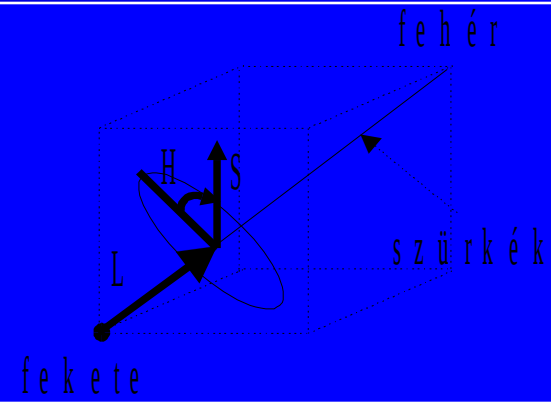
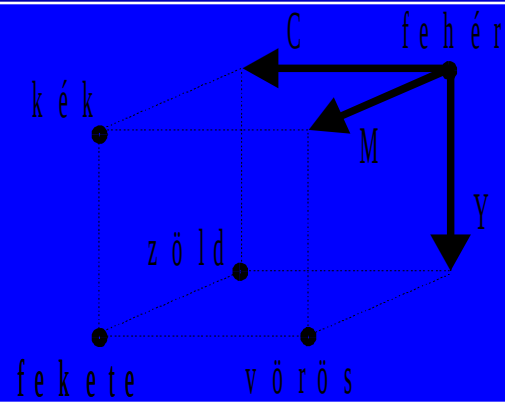
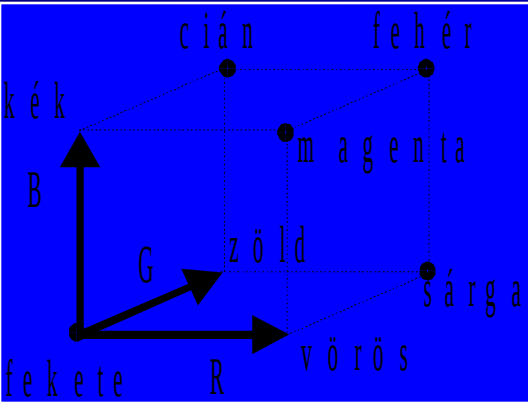
Színérzékelés: nem monokromatikus fény



$$\begin{aligned} r &= \int \Phi(\lambda) r(\lambda) d\lambda \\ g &= \int \Phi(\lambda) g(\lambda) d\lambda \\ b &= \int \Phi(\lambda) b(\lambda) d\lambda \end{aligned}$$

színérzet: linearitás + additivitás

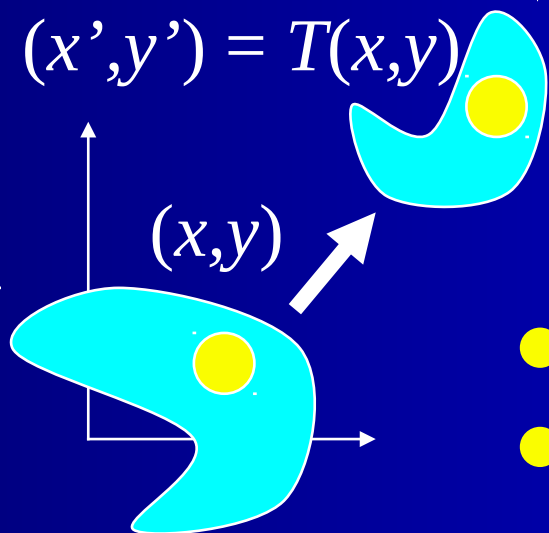
Színek definiálása



Transzformációk

Szirmay-Kalos László

Transzformációk



- Tönkre tehetik az egyenletet
- Korlátozzuk a transzformációkat és az alakzatokat úgy, hogy invariáns legyen
 - Pont, egyenes (szakasz), sík (poligon)

Egybevágósági:
Eltolás, forgatás, tükrözés

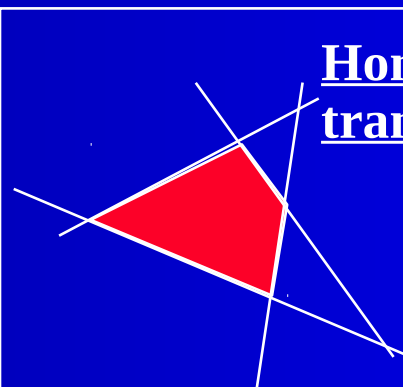
Hasonlósági

Affin transzformációk =

Párhuzamos egyenes tartó
Descartes koordinátákban lineáris

Homogén lineáris
transzformációk

Egyenest egyenesbe
Homogén
koordinátákban lineáris



Affin transzformációk

Ha az új Descartes koordináták az eredetiek lineáris függvényei, akkor a transzformáció párhuzamos egyenes tartó és affinnak nevezzük.

Affin transzformáció

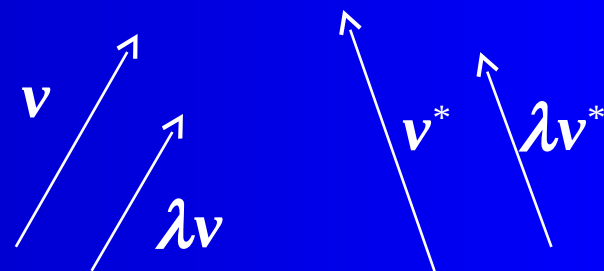
$$x' = ax + by + c$$

$$y' = dx + ey + f$$

Egyenes

$$x(t) = x_0 + v_x t$$

$$y(t) = y_0 + v_y t$$



Transzformált egyenes

$$x'(t) = a(x_0 + v_x t) + b(y_0 + v_y t) + c = (ax_0 + by_0 + c) + (av_x + bv_y)t = x_0' + v_x^* t$$

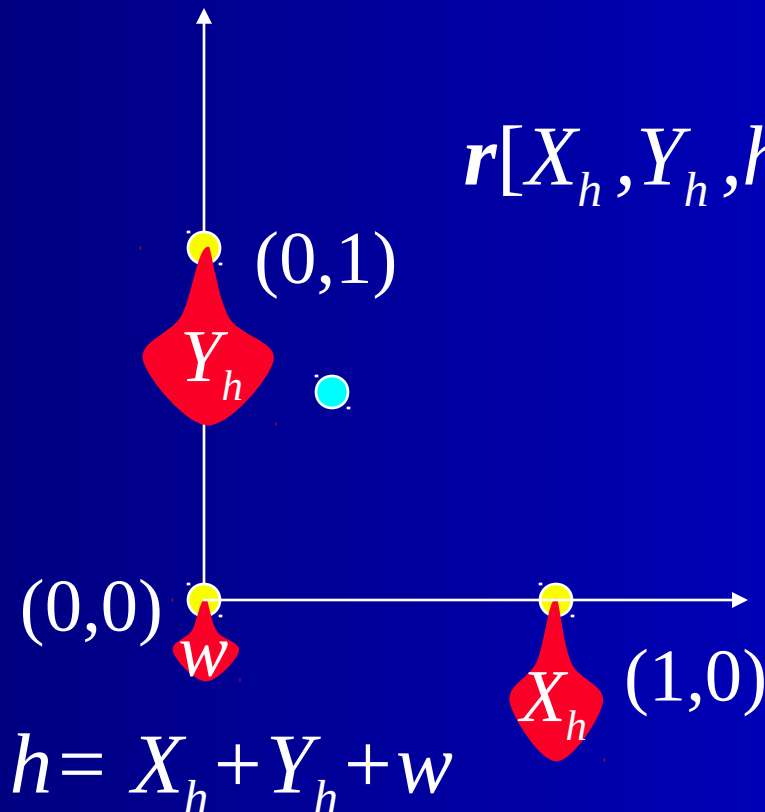
$$y'(t) = d(x_0 + v_x t) + e(y_0 + v_y t) + f = (dx_0 + ey_0 + f) + (dv_x + ev_y)t = y_0' + v_y^* t$$

Homogén koordináták (2D)

$$(x, y) \rightarrow [x, y, 1] \sim [x \cdot h, y \cdot h, h] = [X_h, Y_h, h]$$

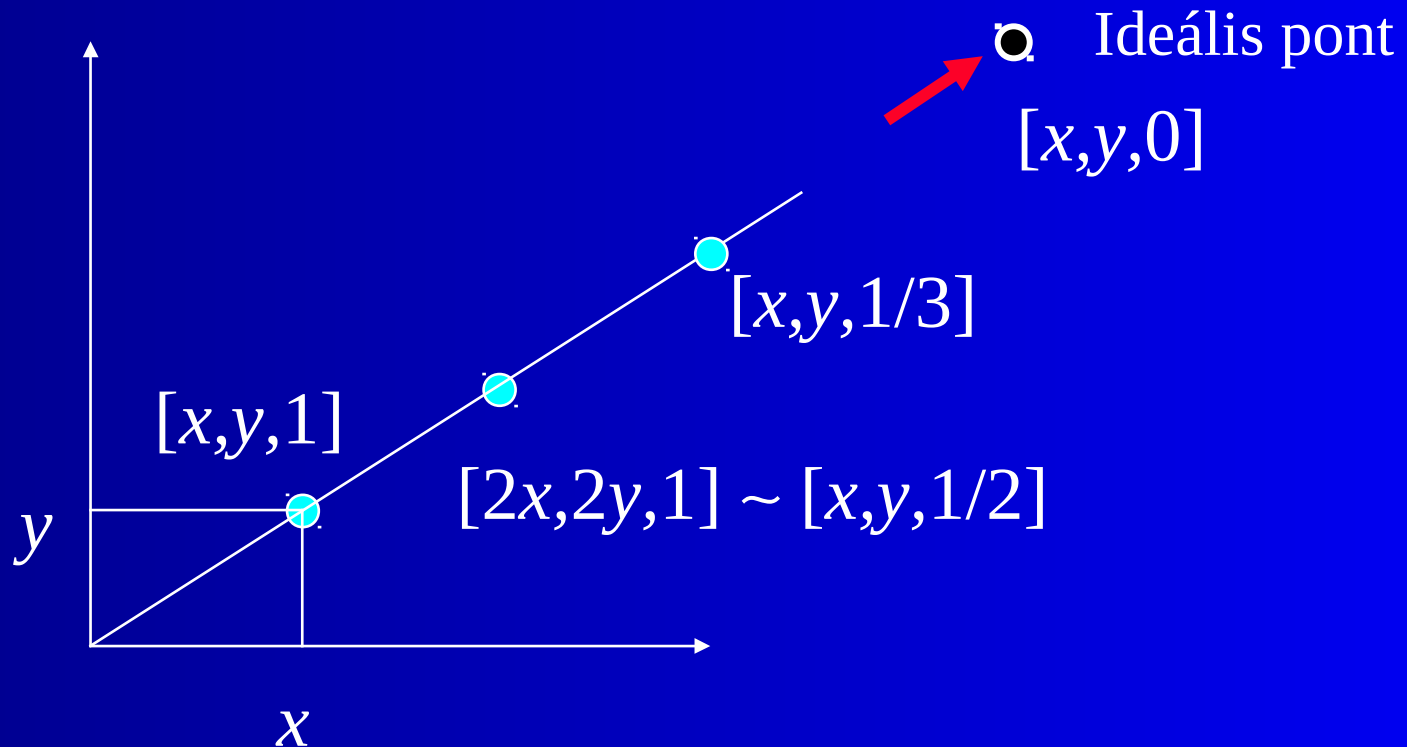
$$r[X_h, Y_h, h] = \frac{X_h(1,0) + Y_h(0,1) + w(0,0)}{h}$$

$$r[X_h, Y_h, h] = \left(\frac{X_h}{h}, \frac{Y_h}{h} \right)$$



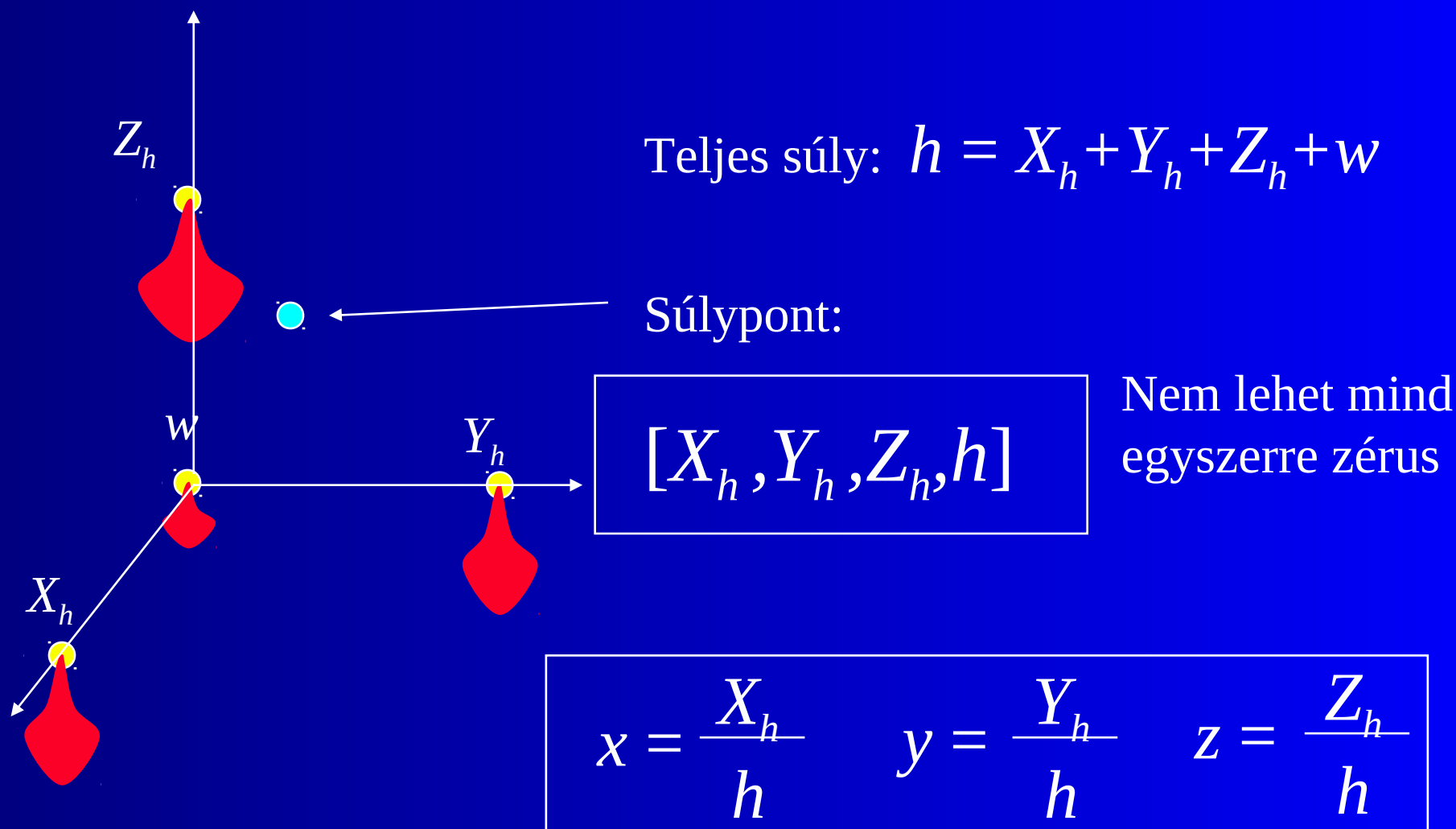
$$x = \frac{X_h}{h} \quad y = \frac{Y_h}{h}$$

Homogén koordináták ideális pontokhoz: $h=0$

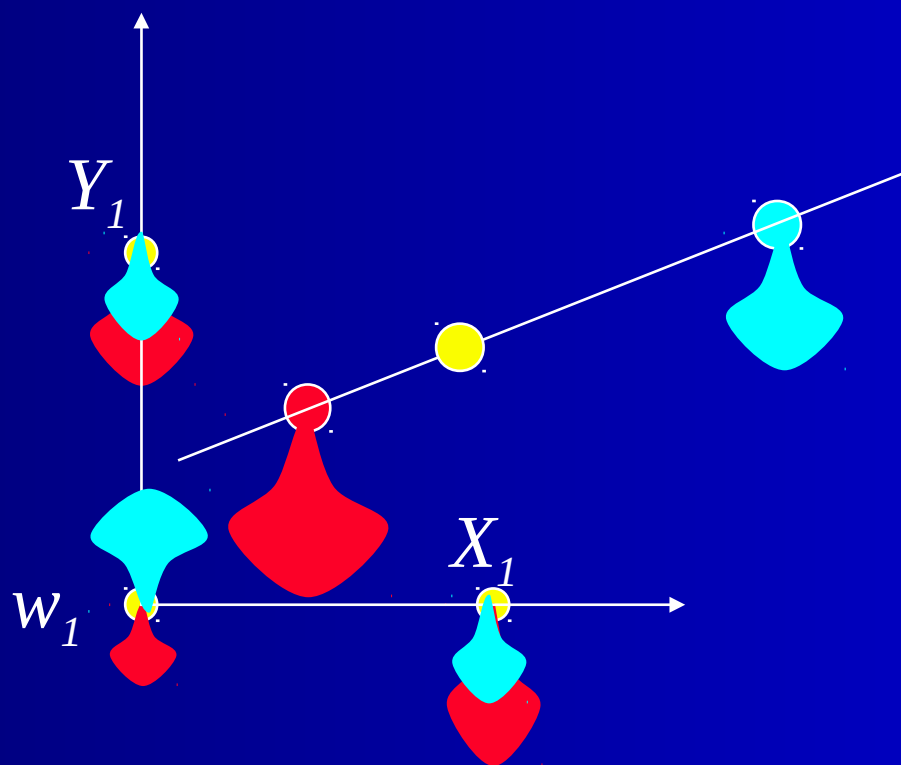


Euklideszi sík + ideális pontok = projektív sík

Homogén koordináták (3D)



Egyenes a projektív térben

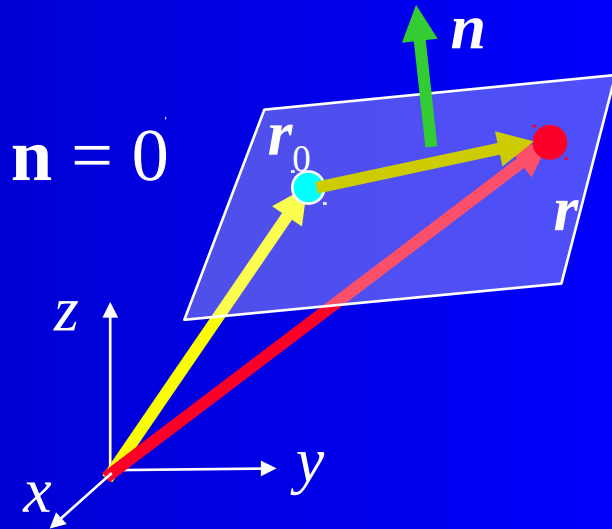


egyenes = két pont kombinációja
szakasz = két pont konvex
kombinációja

$$[X(t), Y(t), Z(t), h(t)] = [X_1, Y_1, Z_1, h_1] \cdot t + [X_2, Y_2, Z_2, h_2] \cdot (1-t)$$

Sík

$$(\mathbf{r} - \mathbf{r}_0) \bullet \mathbf{n} = 0$$



Euklideszi tér, Descartes koordináták:

$$n_x x + n_y y + n_z z + d = 0$$

Euklideszi tér, homogén koordináták:

$$n_x X_h/h + n_y Y_h/h + n_z Z_h/h + d = 0$$

$$h \neq 0$$

Projektív tér:

$$n_x X_h + n_y Y_h + n_z Z_h + d h = 0$$

$$\cancel{h \neq 0}$$

$$[X_h, Y_h, Z_h, h] \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \\ d \end{pmatrix} = 0$$

Homogén lineáris transzformációk

- Homogén koordinátavektor szorzása mátrixszal
- 2D transzformáció 3x3 mátrix

$$[X_h', Y_h', h'] = [X_h, Y_h, h] \cdot \mathbf{T}_{3 \times 3}$$

- 3D transzformáció 4x4 mátrix

$$[X_h', Y_h', Z_h', h'] = [X_h, Y_h, Z_h, h] \cdot \mathbf{T}_{4 \times 4}$$

Affin transzformációk

- Ha az utolsó oszlop $[0,0,1]^T$ vagy $[0,0,0,1]^T$
- Descartes koordinátákra lineáris
- Párhuzamos egyenestartó

$$x' = a_{11}x + a_{21}y + a_{31}$$

$$y' = a_{12}x + a_{22}y + a_{32}$$

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}$$

Transzformációk összefűzése

$$\begin{aligned}[X_h', Y_h', Z_h', h'] &= (\dots ([X_h, Y_h, Z_h, h] \cdot T_1) \cdot T_2) \dots T_n = \\ &= [X_h, Y_h, Z_h, h] \cdot (T_1 \cdot T_2 \cdot \dots \cdot T_n) = \\ &= [X_h, Y_h, Z_h, h] \cdot T\end{aligned}$$

Mátrisszorzás asszociatív!

Homogén lineáris transzformációk tulajdonságai

- Pontot pontba, egyenest egyenesbe, konvex kombinációkat konvex kombinációkba képeznek le

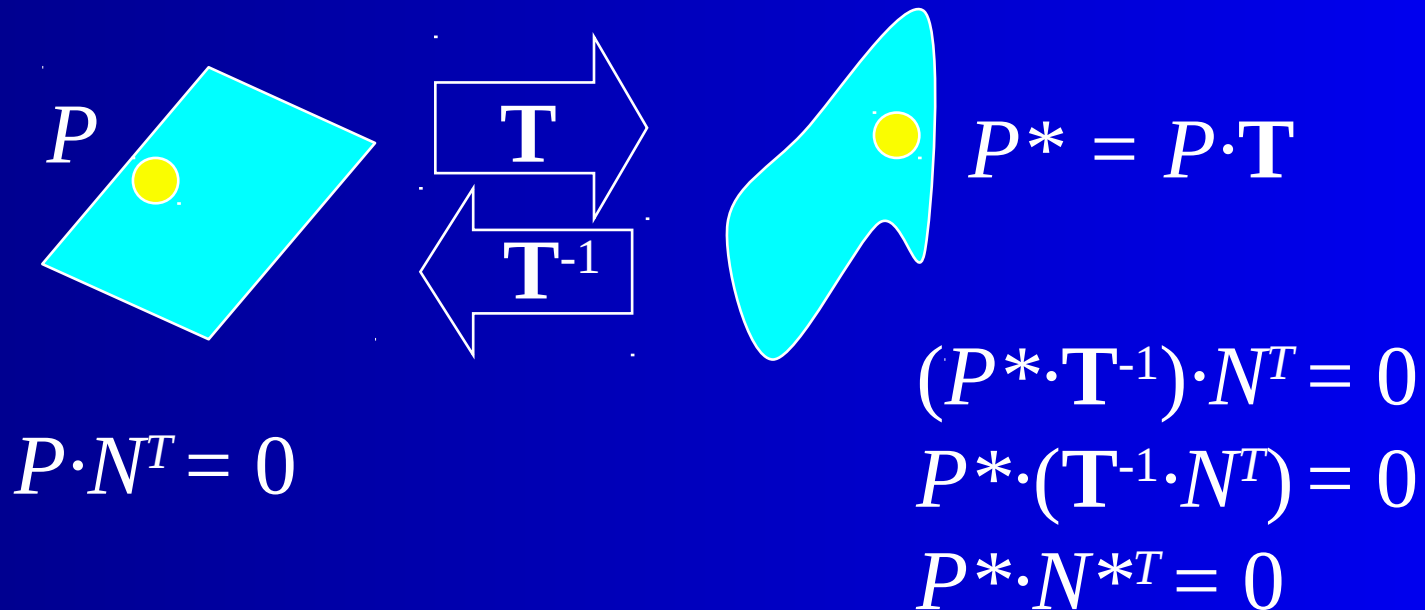
Példa: egyenest egyenesbe

$$[X(t), Y(t), Z(t), h(t)] = [X_1, Y_1, Z_1, h_1] \cdot t + [X_2, Y_2, Z_2, h_2] \cdot (1-t)$$

$$P(t) = P_1 \cdot t + P_2 \cdot (1-t) \quad // \cdot \mathbf{T}$$

$$P^*(t) = P(t) \cdot \mathbf{T} = (P_1 \cdot \mathbf{T}) \cdot t + (P_2 \cdot \mathbf{T}) \cdot (1-t)$$

Invertálható homogén lineáris transzformációk: síkot síkba

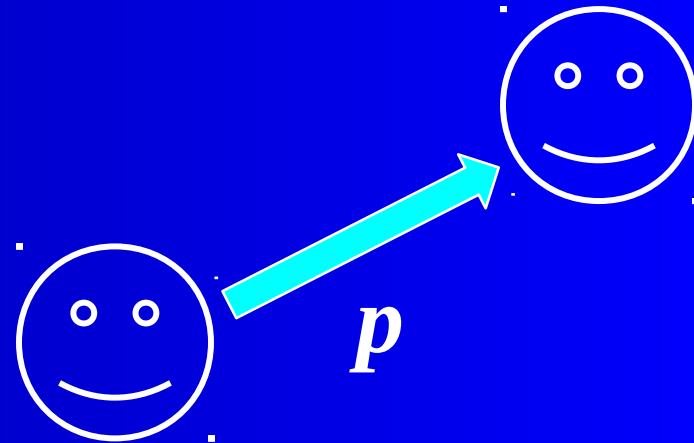


$$N^* = N \cdot (\mathbf{T}^{-1})^T$$

Inverse transpose

Eltolás

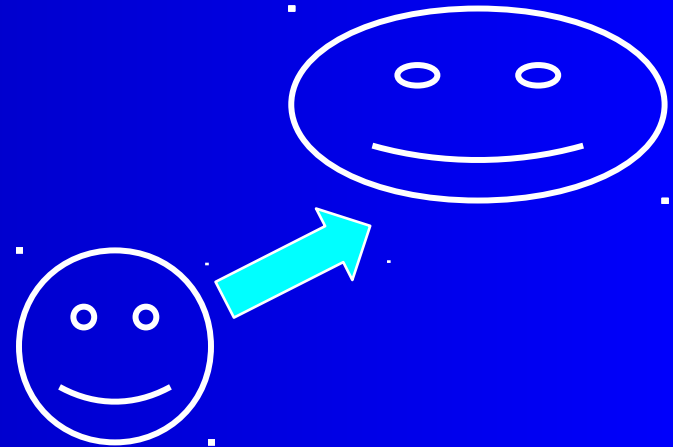
$$(x', y', z') = (x + p_x, y + p_y, z + p_z)$$



$$[x', y', z', 1] = [x, y, z, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix}$$

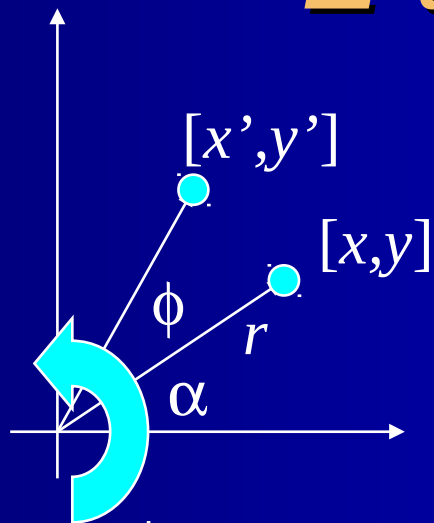
Skálázás

$$x' = S_x \cdot x, \quad y' = S_y \cdot y, \quad z' = S_z \cdot z$$



$$[x', y', z', 1] = [x, y, z, 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

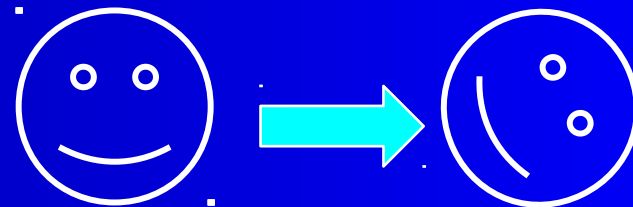
Z tengely körüli forgatás



$$z' = z$$

$$x = r \cos \alpha$$

$$y = r \sin \alpha$$



$$x' = r \cos(\alpha + \phi) = \boxed{r \cos \alpha} \cos \phi - \boxed{r \sin \alpha} \sin \phi$$

$$y' = r \sin(\alpha + \phi) = r \cos \alpha \sin \phi + r \sin \alpha \cos \phi$$

$$x' = r \cos(\alpha + \phi) = x \cos \phi - y \sin \phi$$

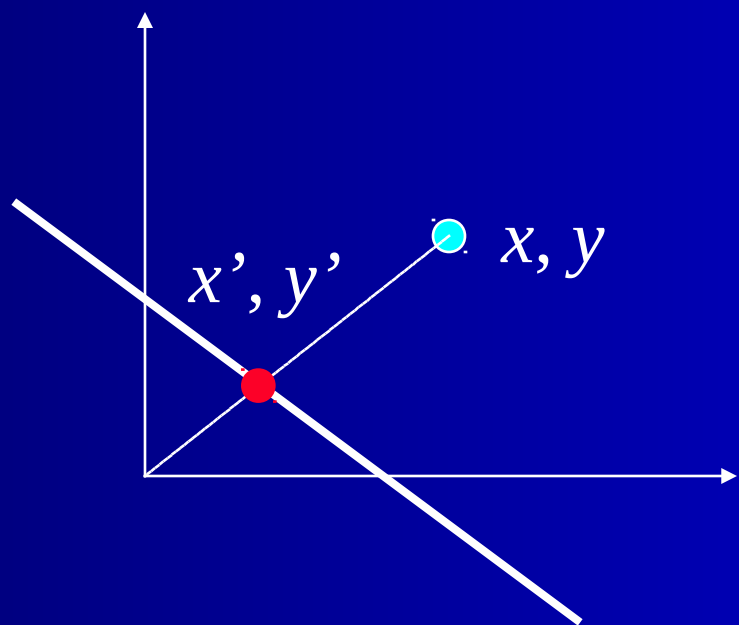
$$y' = r \sin(\alpha + \phi) = x \sin \phi + y \cos \phi$$

$$[x', y', z', 1] = [x, y, z, 1] \begin{bmatrix} \cos \phi & \sin \phi & 0 & 0 \\ -\sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-tengely körüli forgatás homogén lineáris transzformációs mátrixa



Középpontos vetítés (2D)



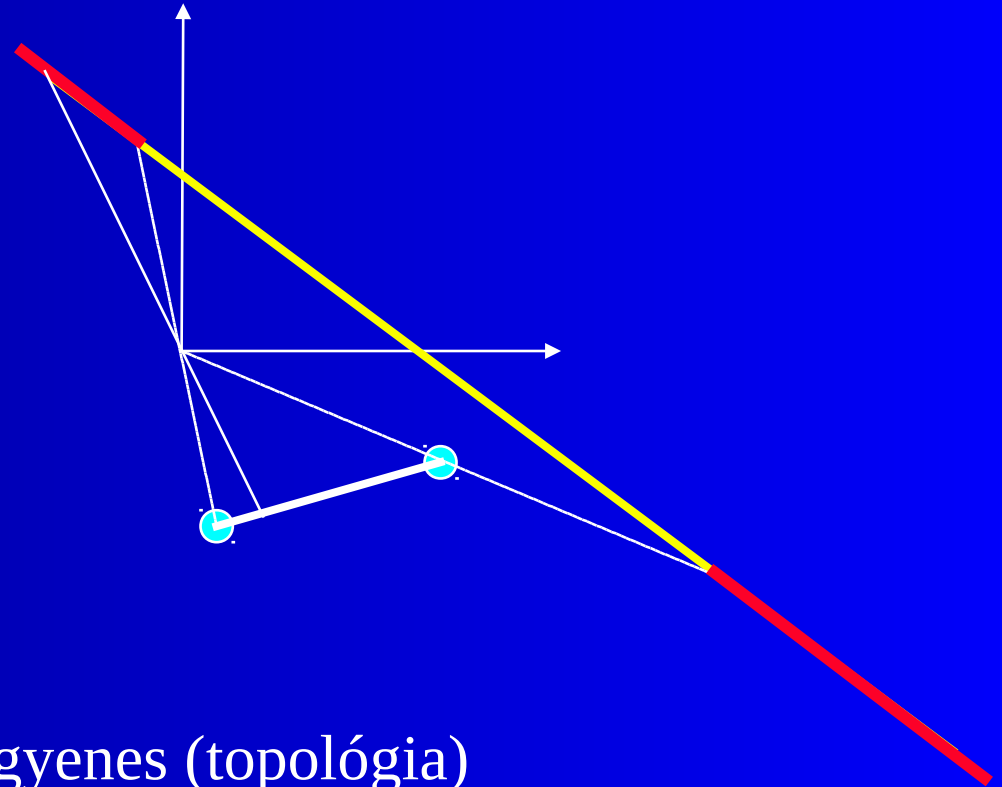
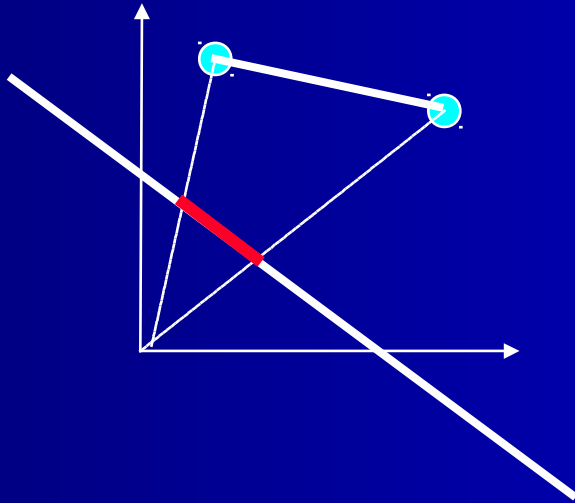
$$x/y = x'/y' \text{ és } px + qy = 1$$

$$\begin{matrix} \nearrow \\ [x, y, 1] \end{matrix} \begin{bmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} \\ [x, y, px + qy] \end{matrix}$$

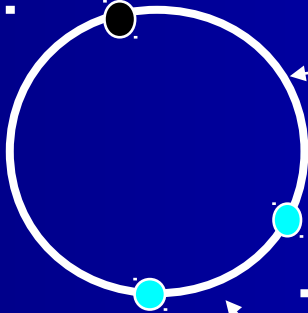


$$\begin{bmatrix} \frac{x}{px + qy} & \frac{y}{px + qy} \end{bmatrix}$$

Átfordulási probléma



Ideális pont



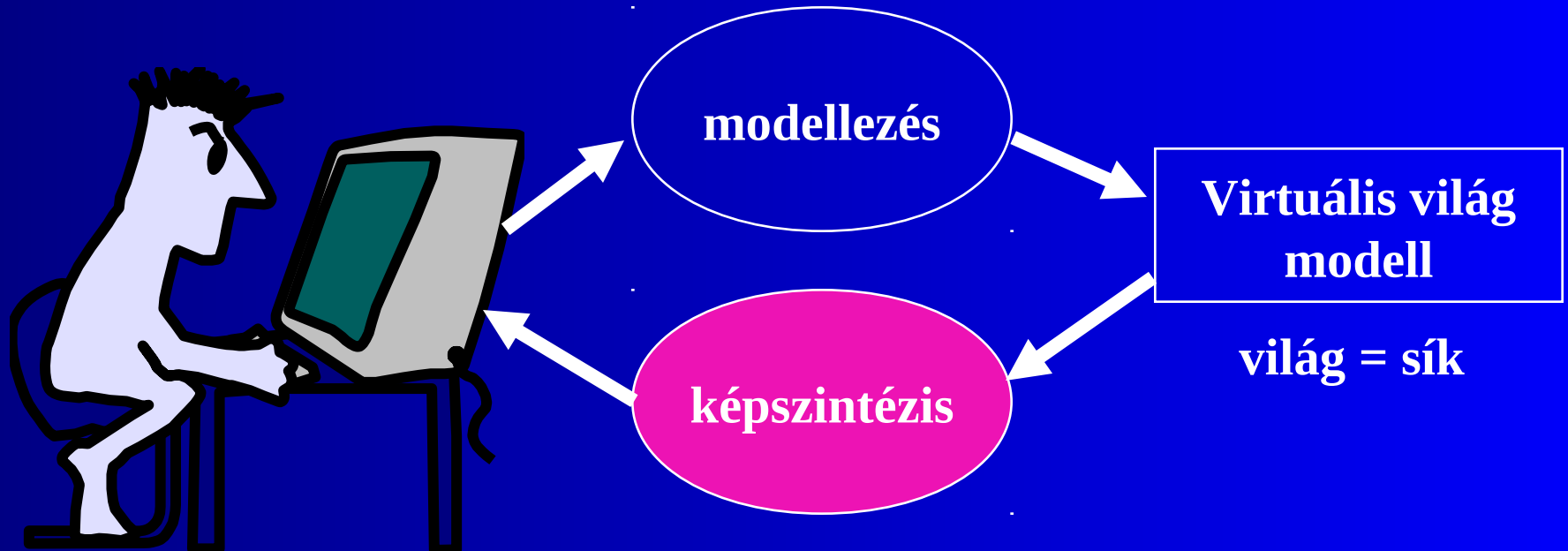
=Projektív egyenes (topológia)

Szakasz ?????

2D képszintézis

Szirmay-Kalos László

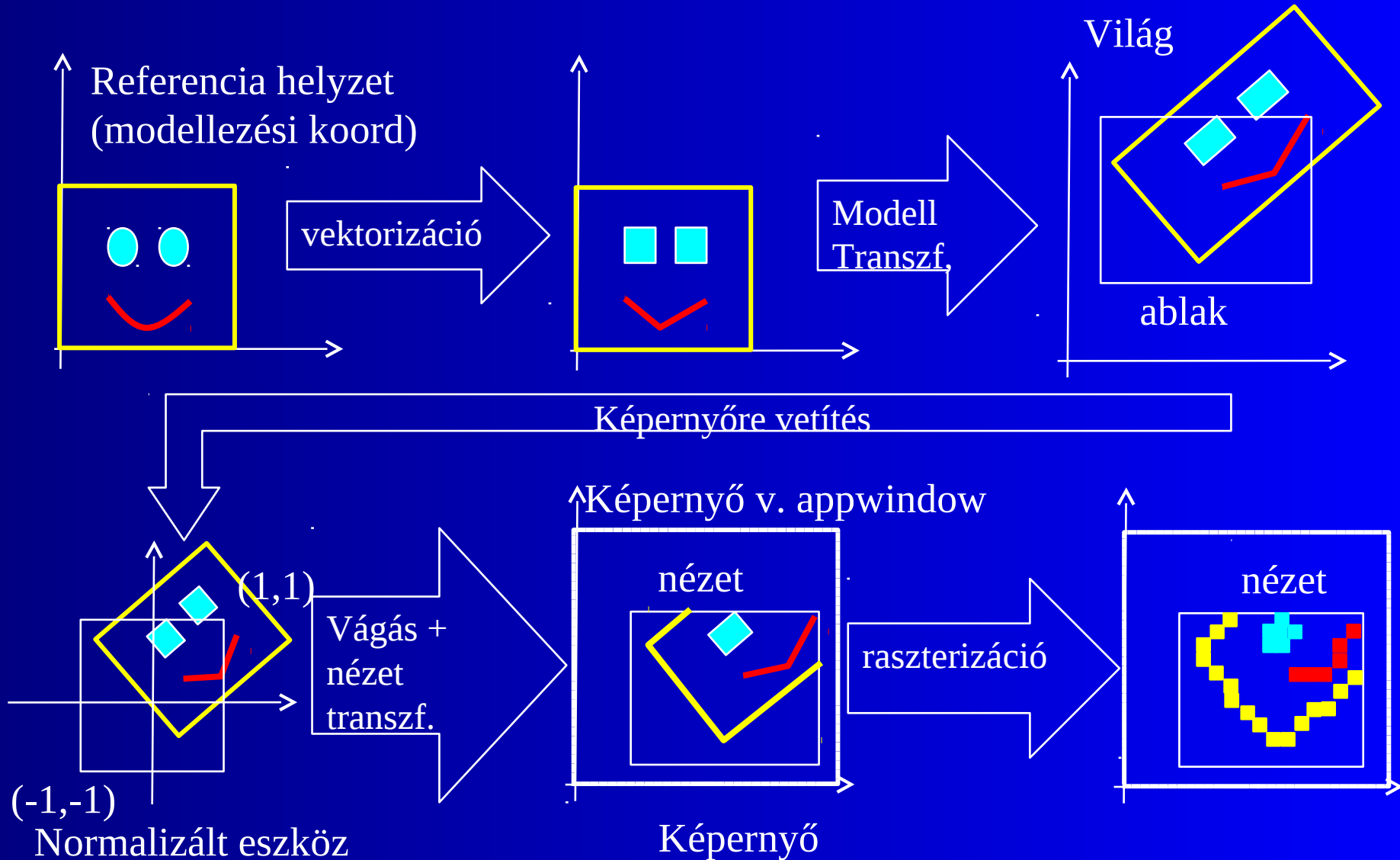
Számítógépes grafika



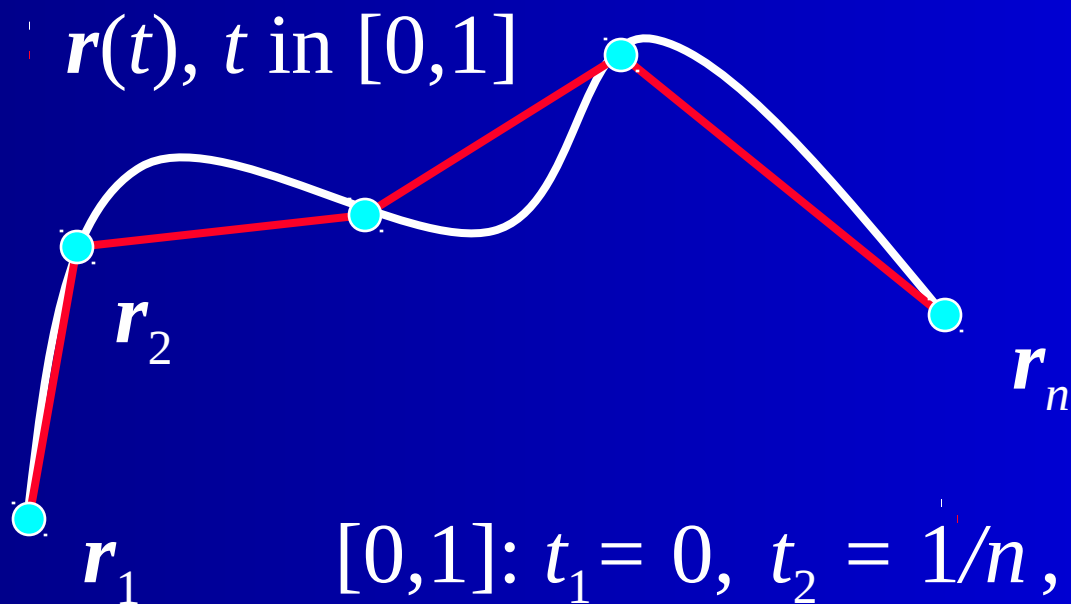
Metafórák:

- 2D rajzolás

2D megjelenítési csővezeték



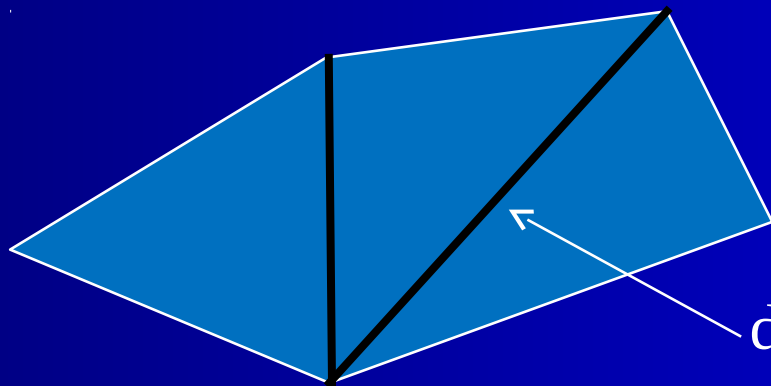
Vektorizáció



$$r_1 = r(0), r_2 = r(t_2), \dots, r_n = r(1)$$

Poligon háromszögekre bontása

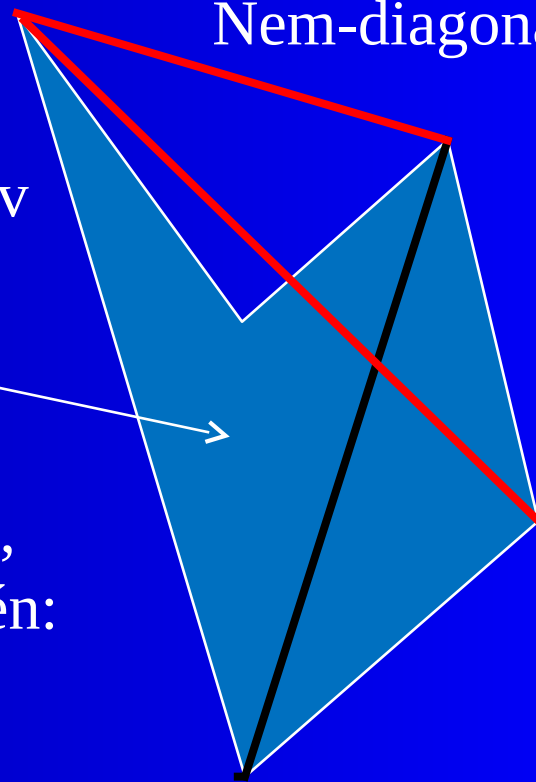
Konvex



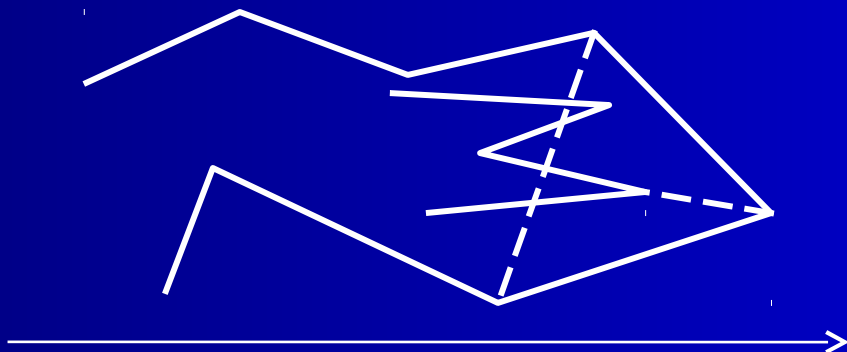
diagonál

Konkáv

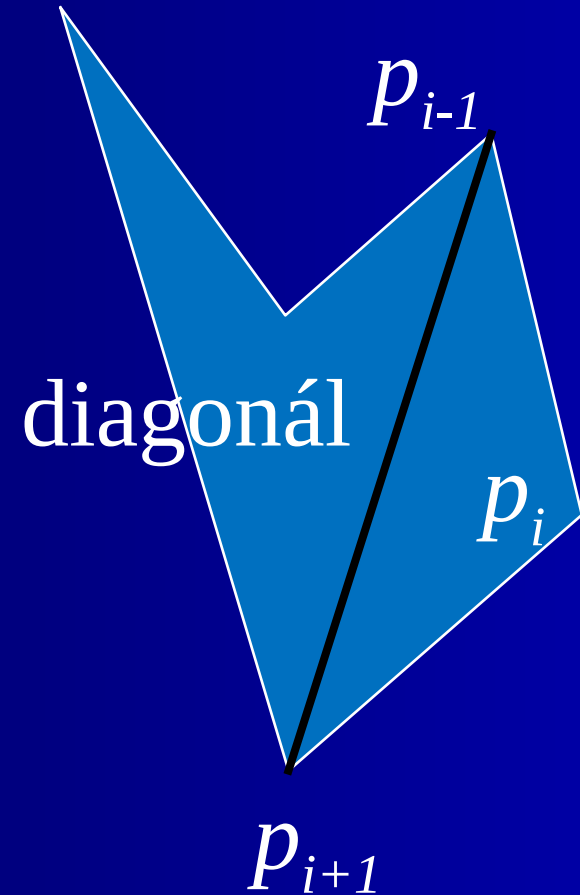
Nem-diagonál



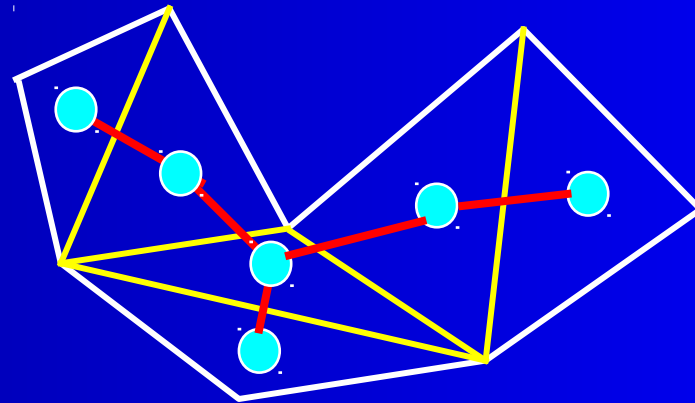
Minden 4+ csúcsú sokszögnek van diagonálja,
azaz mindegyik felbontható diagonálok mentén:



Fül

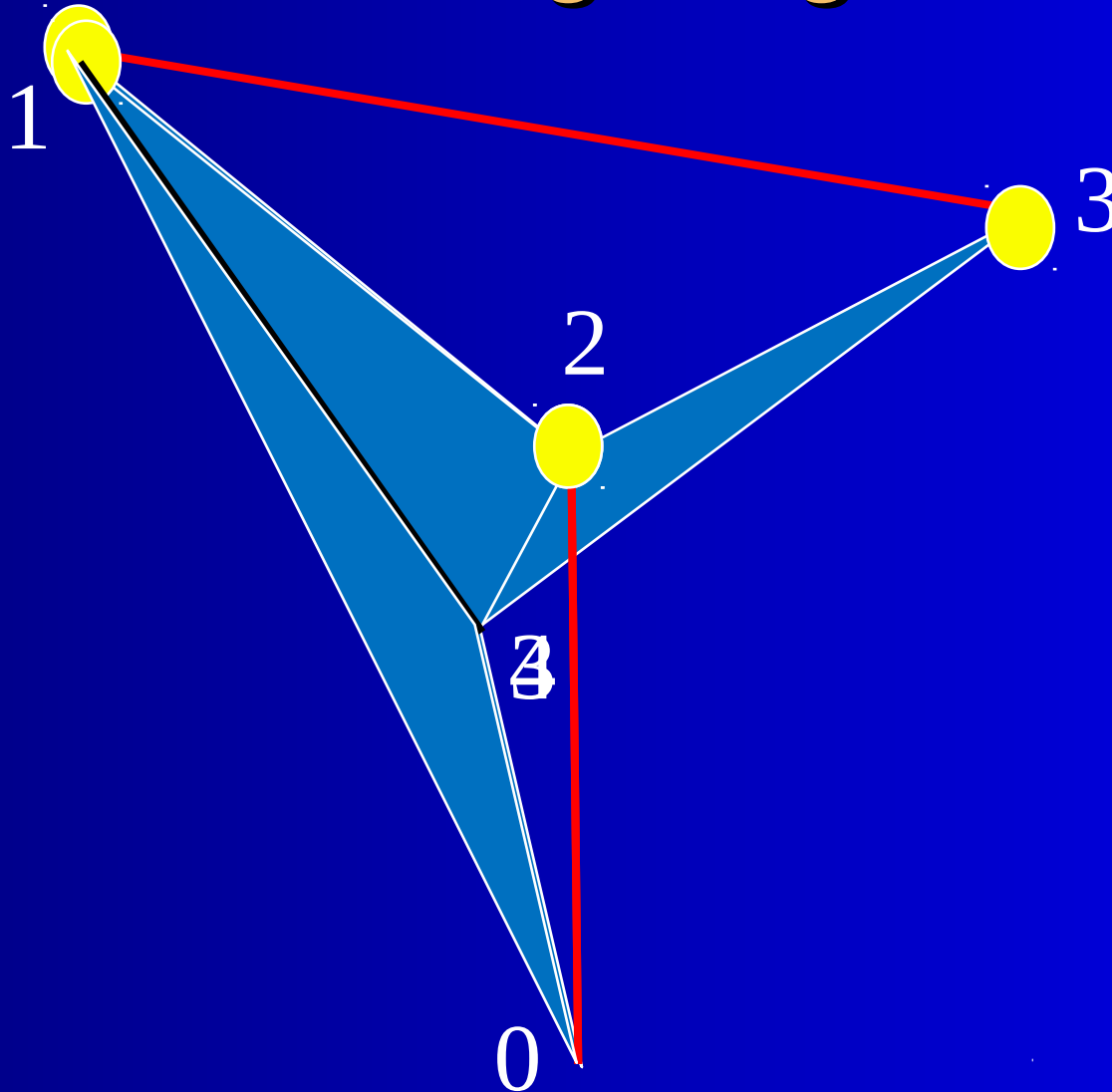


- p_i fül, ha $p_{i-1} - p_{i+1}$ diagonál
- Fül levágható!
- **Két fül tétel:** Minden legalább 4 csúcsú sokszögnek van legalább 2 füle.



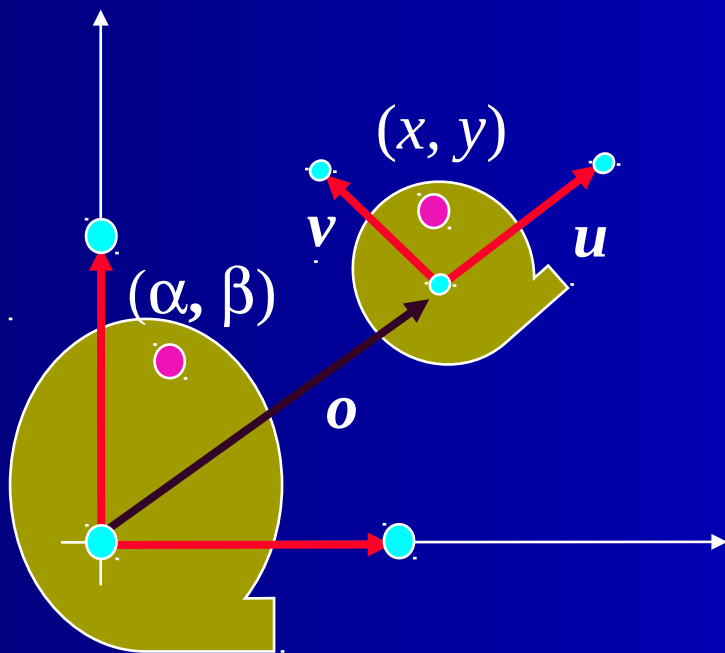
- **Fügvágás:** keress fület és nyissz!.

Fülvágó algoritmus



Modellezési transzformáció

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \phi & \sin \phi & 0 \\ -\sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p_x & p_y & 1 \end{bmatrix}$$



$$(x, y) = \mathbf{o} + \alpha\mathbf{u} + \beta\mathbf{v}$$

$\mathbf{T}_M$

$$\begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ o_x & o_y & 1 \end{bmatrix}$$

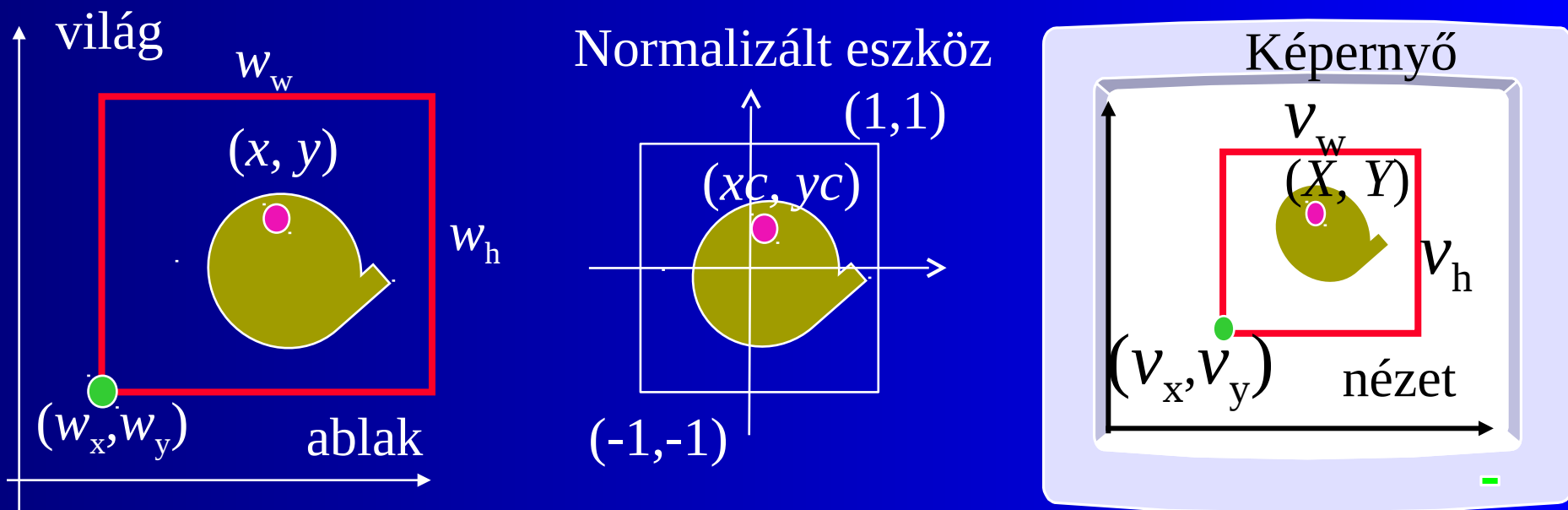
$$[x, y, 1] = [\alpha, \beta, 1] \cdot$$

$$[0, 0, 1] \quad [o_x \quad o_y \quad 1]$$

$$[1, 0, 1] \quad [o_x + u_x \quad o_y + u_y \quad 1]$$

$$[0, 1, 1] \quad [o_x + v_x \quad o_y + v_y \quad 1]$$

Képernyő projekció



$$\begin{aligned} xc &= 2(x - w_x) / w_w - 1 \\ yc &= 2(y - w_y) / w_h - 1 \end{aligned}$$

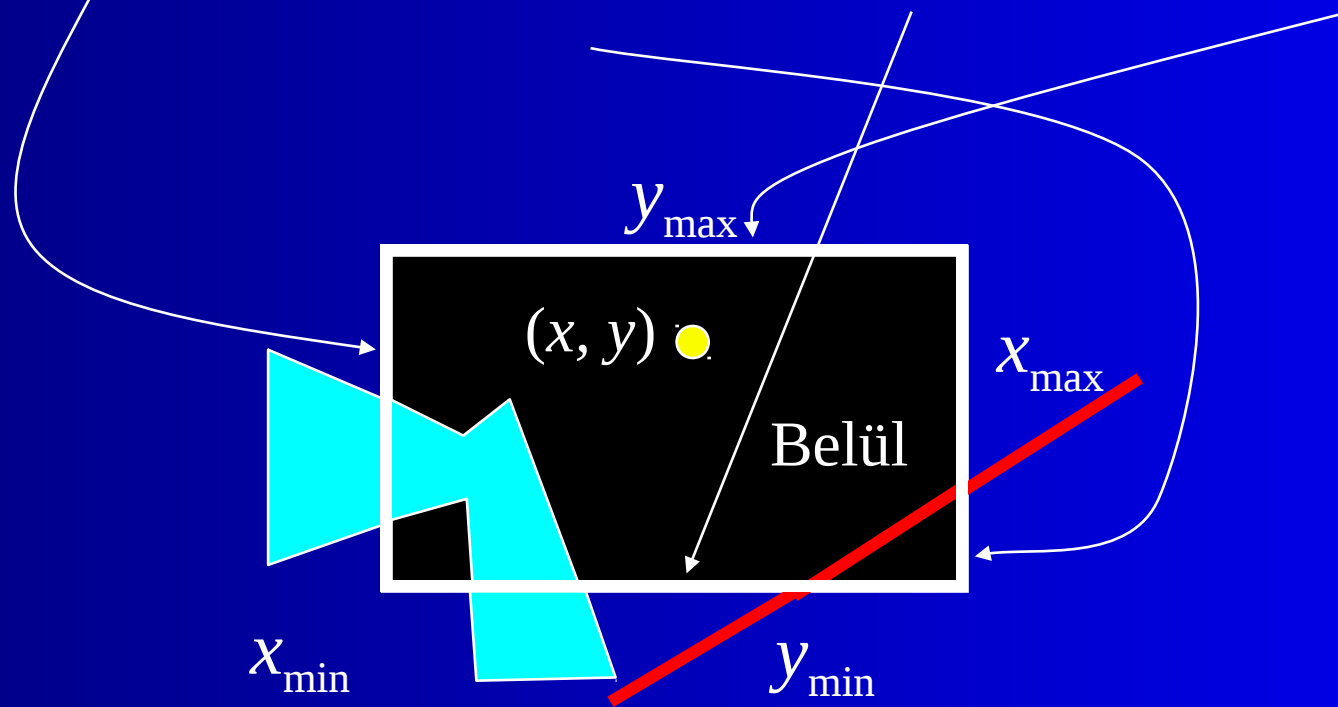
$$\begin{aligned} X &= v_w (xc + 1) / 2 + v_x \\ Y &= v_h (yc + 1) / 2 + v_y \end{aligned}$$

$$[xc, yc, 1] = [x, y, 1] \begin{bmatrix} 2/w_w & 0 & 0 \\ 0 & 2/w_h & 0 \\ -1-2w_x/w_w & -1-2w_y/w_h & 1 \end{bmatrix}$$

Vágás

Pont vágás:

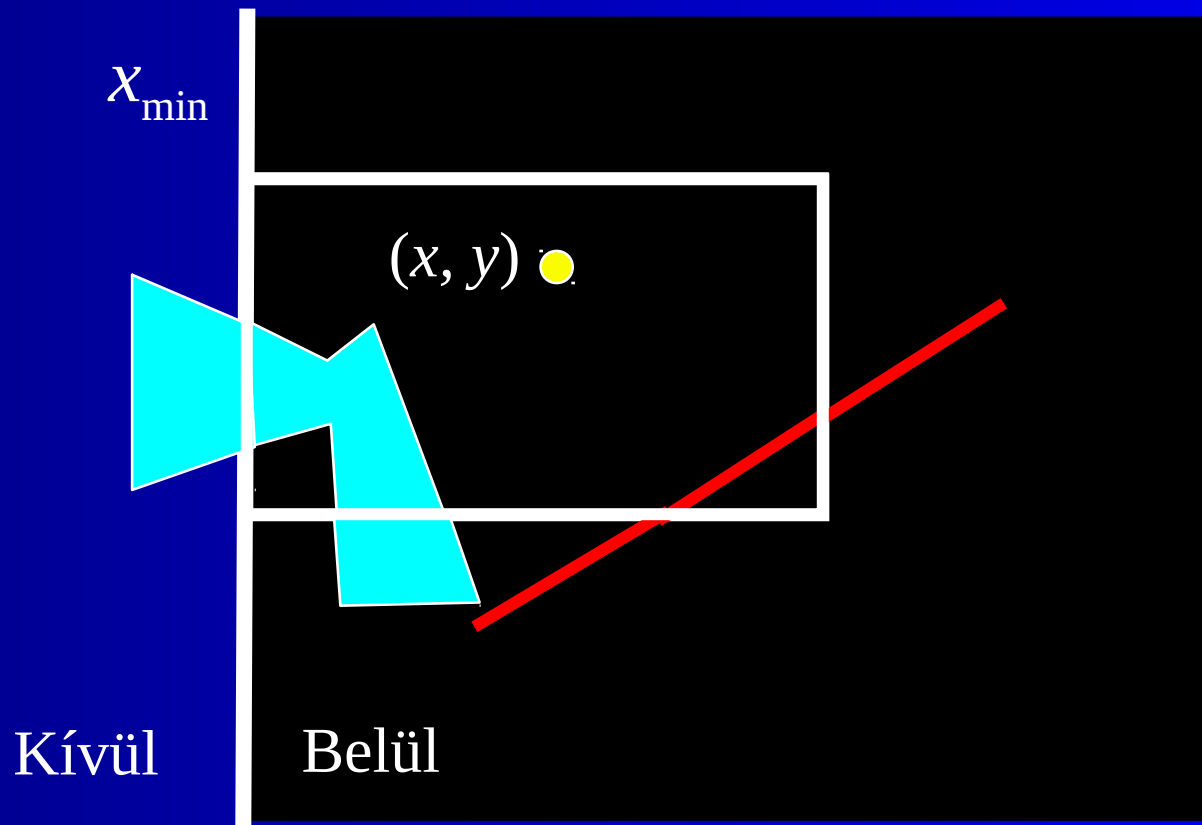
$$x > x_{\min} = -1 \quad x < x_{\max} = +1 \quad y > y_{\min} = -1 \quad y < y_{\max} = +1$$



Kívül

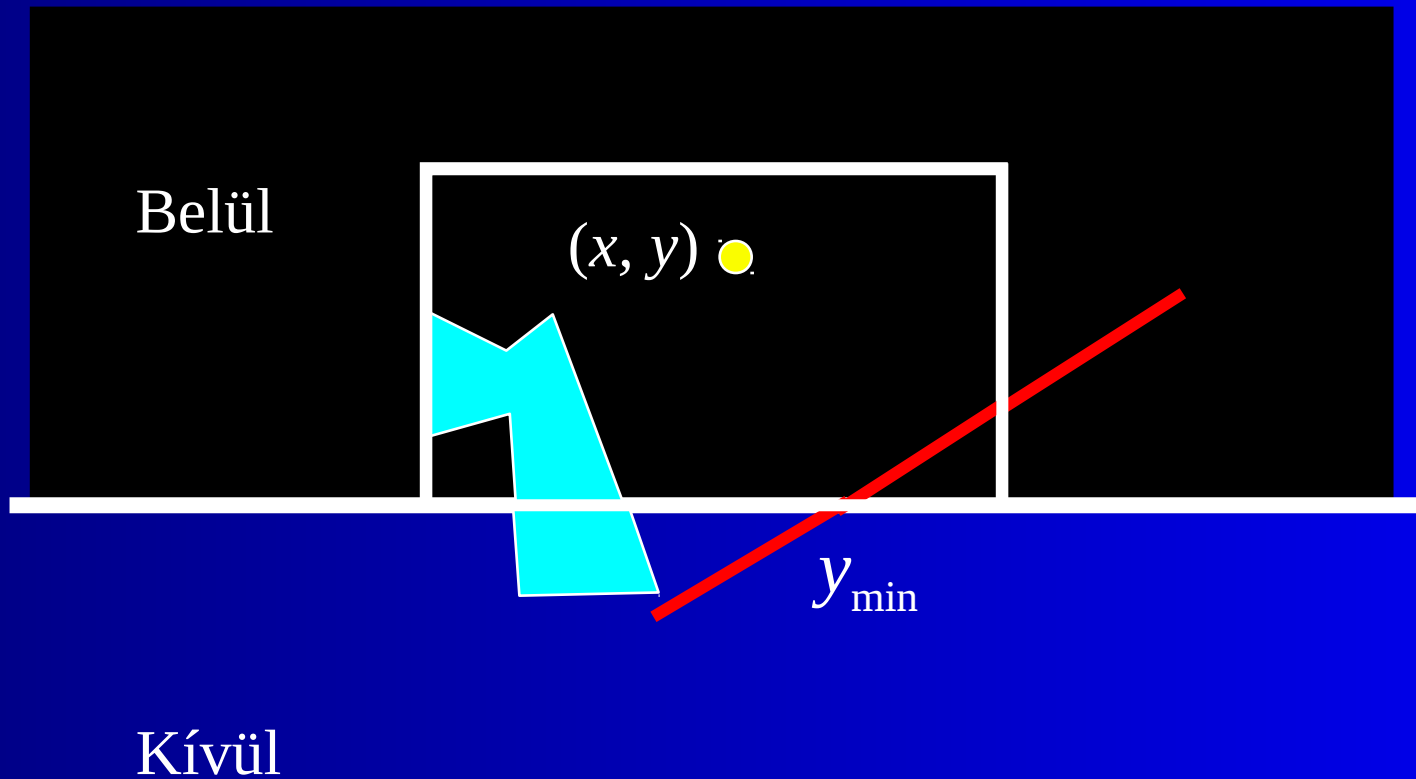
Vágás

$$x > x_{\min}$$



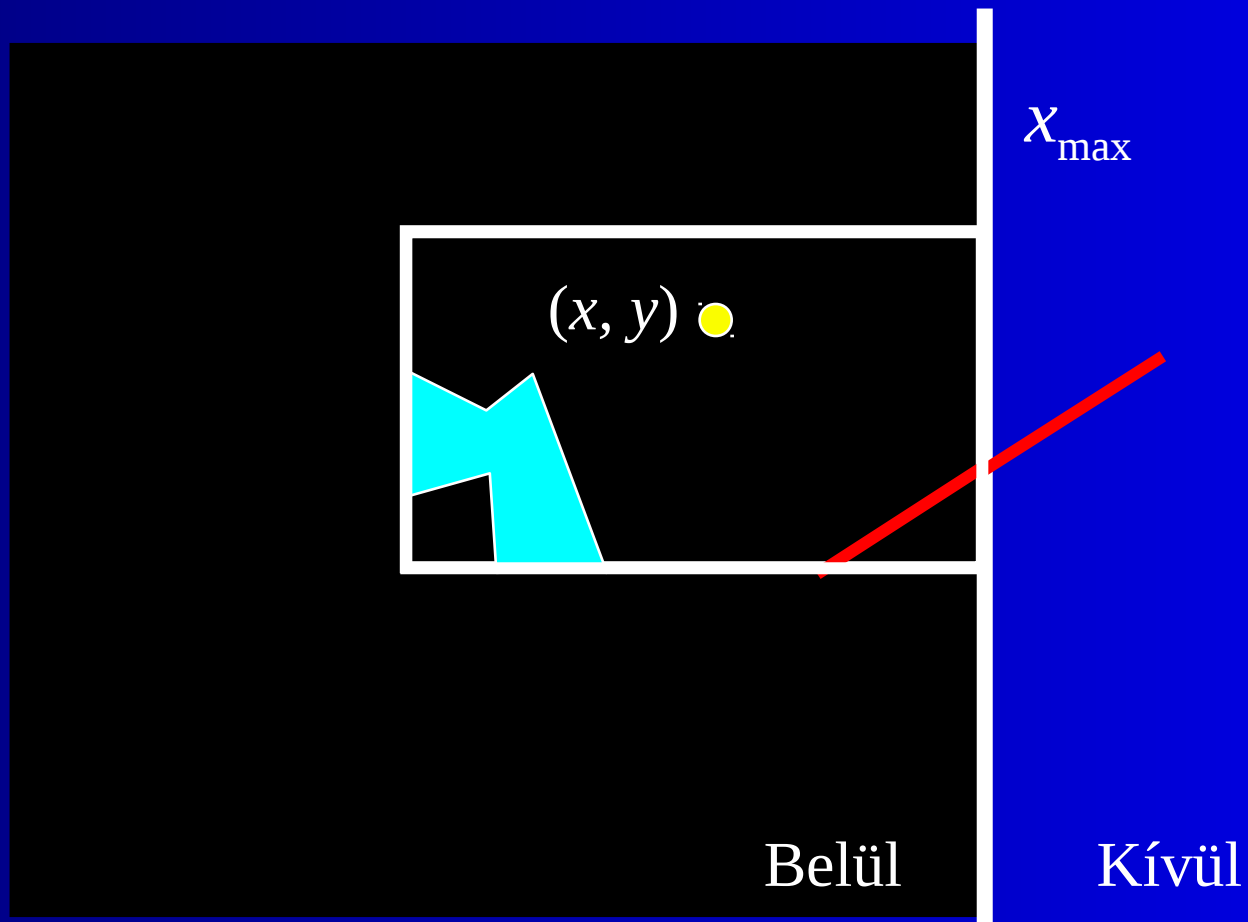
Vágás

$$y > y_{\min}$$



Vágás

$$x < x_{\max}$$



Vágás

$$y < y_{\max}$$

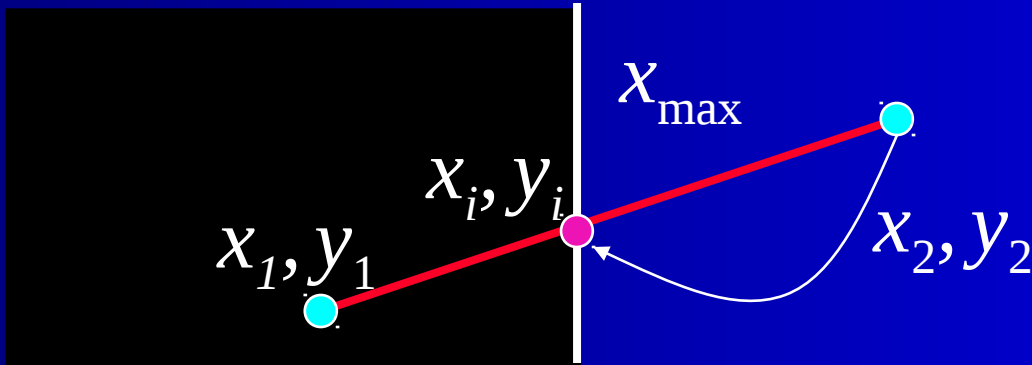
Kívül

Belül

(x, y) ●



Szakasz vágás



$$x(t) = x_1 + (x_2 - x_1)t, \quad y(t) = y_1 + (y_2 - y_1)t$$

$$x = x_{\max}$$

– Metszés:

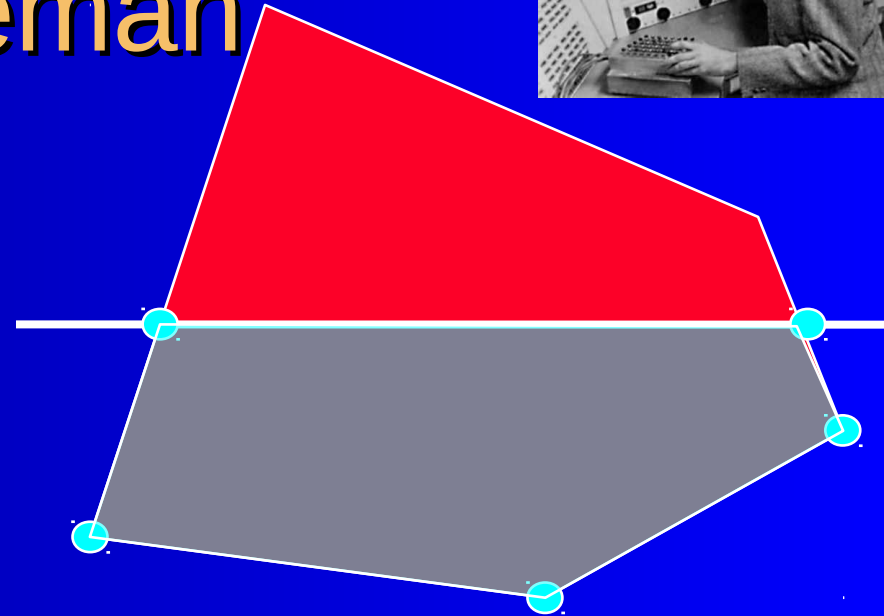
$$- x_{\max} = x_1 + (x_2 - x_1)t \quad \Leftrightarrow \quad t = (x_{\max} - x_1) / (x_2 - x_1)$$

$$x_i = x_{\max} \quad y_i = y_1 + (y_2 - y_1) (x_{\max} - x_1) / (x_2 - x_1)$$

Sutherland-Hodgeman poligonvágás

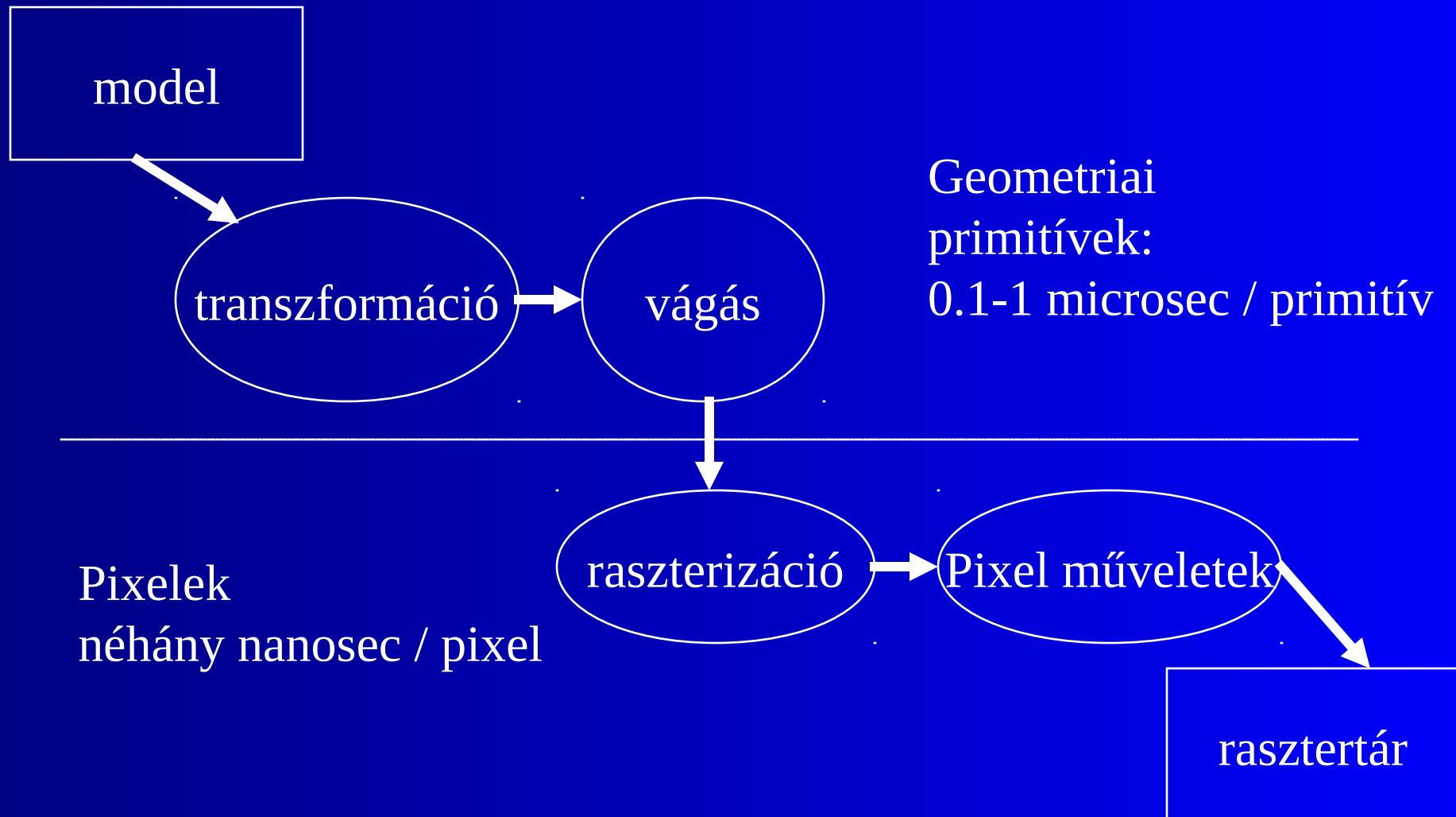


```
PolygonClip(p[n] ⇒ q[m])  
  m = 0;  
  for( i=0; i < n; i++) {  
    if (p[i] belső) {  
      q[m++] = p[i];  
      if (p[i+1] külső)  
        q[m++] = Intersect(p[i], p[i+1], vágóegyenes);  
    } else {  
      if (p[i+1] belső)  
        q[m++] = Intersect(p[i], p[i+1], vágóegyenes);  
    }  
  }  
}
```

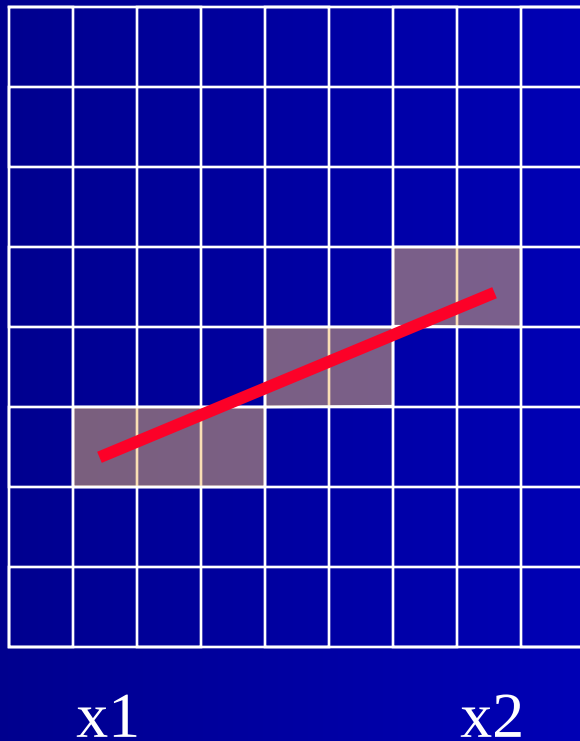


Első pontot még egyszer
a tömb végére

Raszterizáció



Szakasz rajzolás



Egyenes egyenlete:

$$y = mx + b$$

Egyeneshúzás

```
for( x = x1; x <= x2; x++) {  
    Y = m*x + b;  
    y = Round( Y );  
    write( x, y );  
}
```

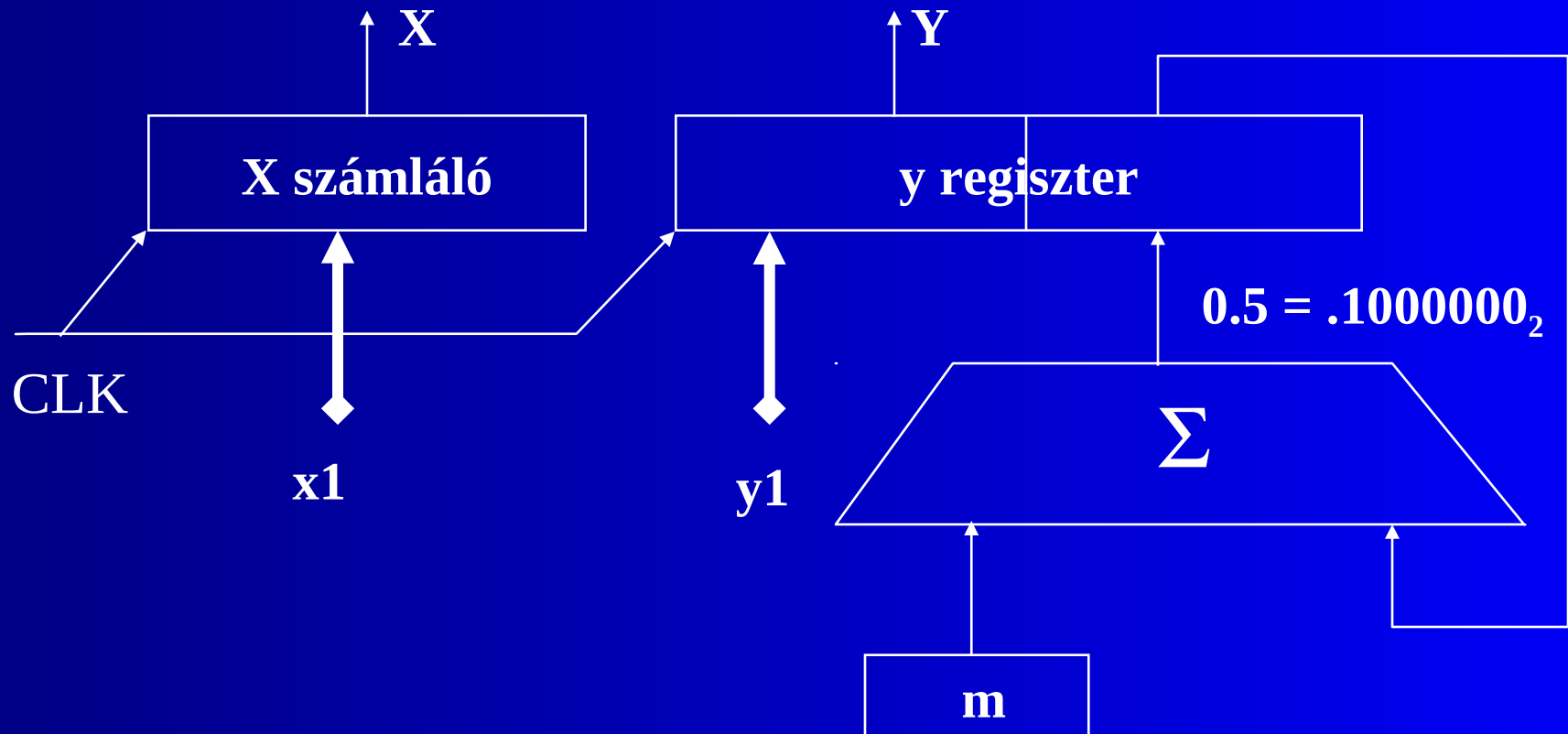
Inkrementális elv

- Szakasz rajzolás: $Y(X) = mX + b$
 - $Y(X) = F(Y(X-1)) = Y(X-1) + m$
- Összeadás: fixpontos ábrázolás: $y = Y 2^T$
 - T : hiba < 1 a leghosszabb műveletsornál
 - $N 2^{-T} < 1$: $T > \log_2 N$
 - $\text{round}(y)$: $y+0.5$ -t csonkítjuk

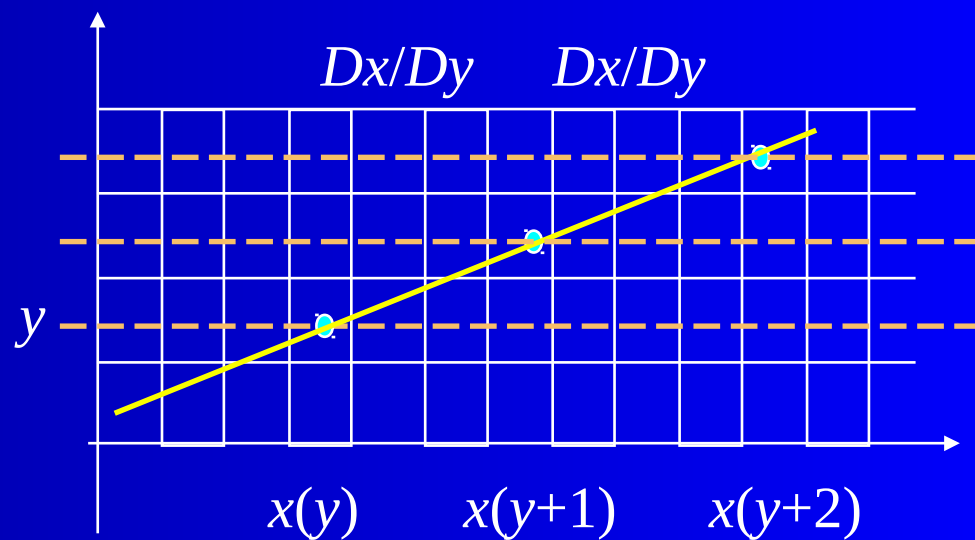
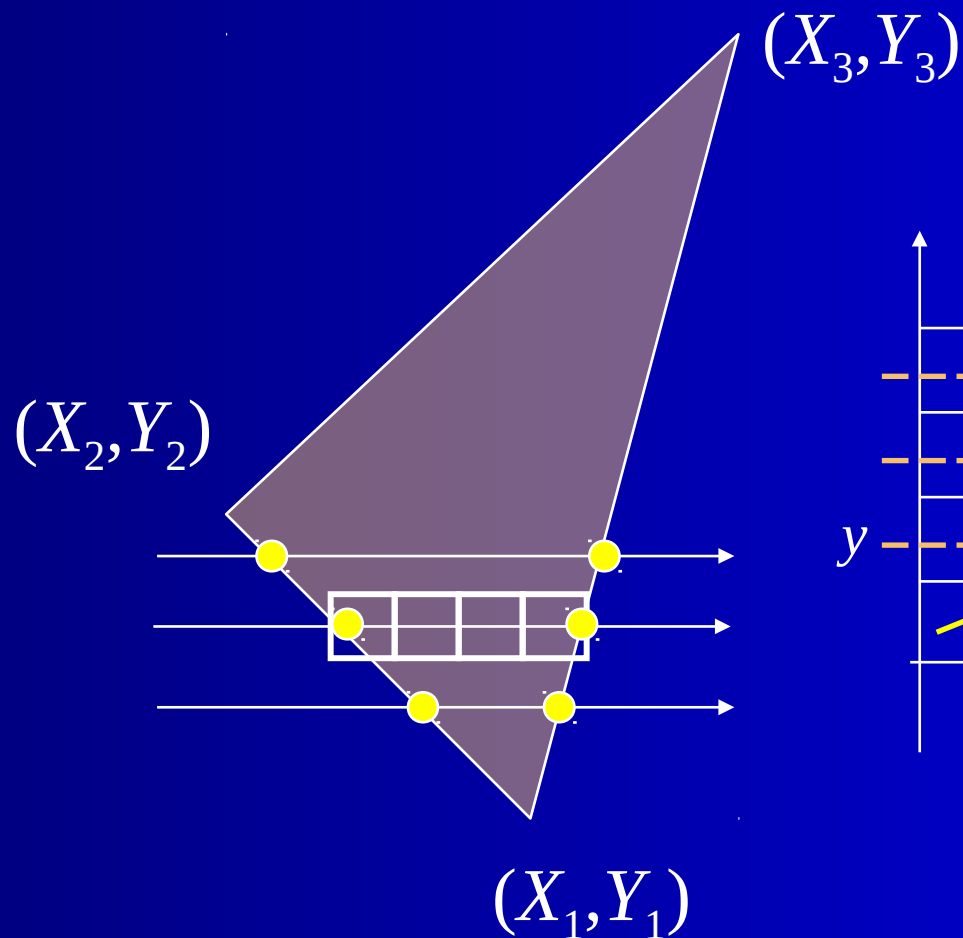
DDA szakaszrajzolás

```
DDADrawLine(x1, y1, x2, y2) {  
    m = (y2 - y1)/(x2 - x1);  
    y = y1;                                + 0.5  
    FOR X = x1 TO x2 {  
        Y = round(y);                      trunc(y)  
        WRITE(X, Y, color);  
        y = y+m;  
    }  
}
```


DDA szakaszrajzoló hardver

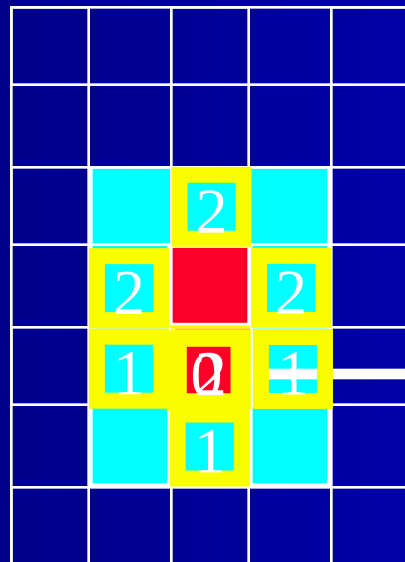


Háromszög kitöltés



Terület elárasztás

Belső pont = kék



seed

```
Flood(x, y) {
    if (pixel[x][y] inner point) {
        Write(x, y, color);
        Flood(x, y-1);
        Flood(x, y+1);
        Flood(x-1, y);
        Flood(x+1, y);
    }
}
```

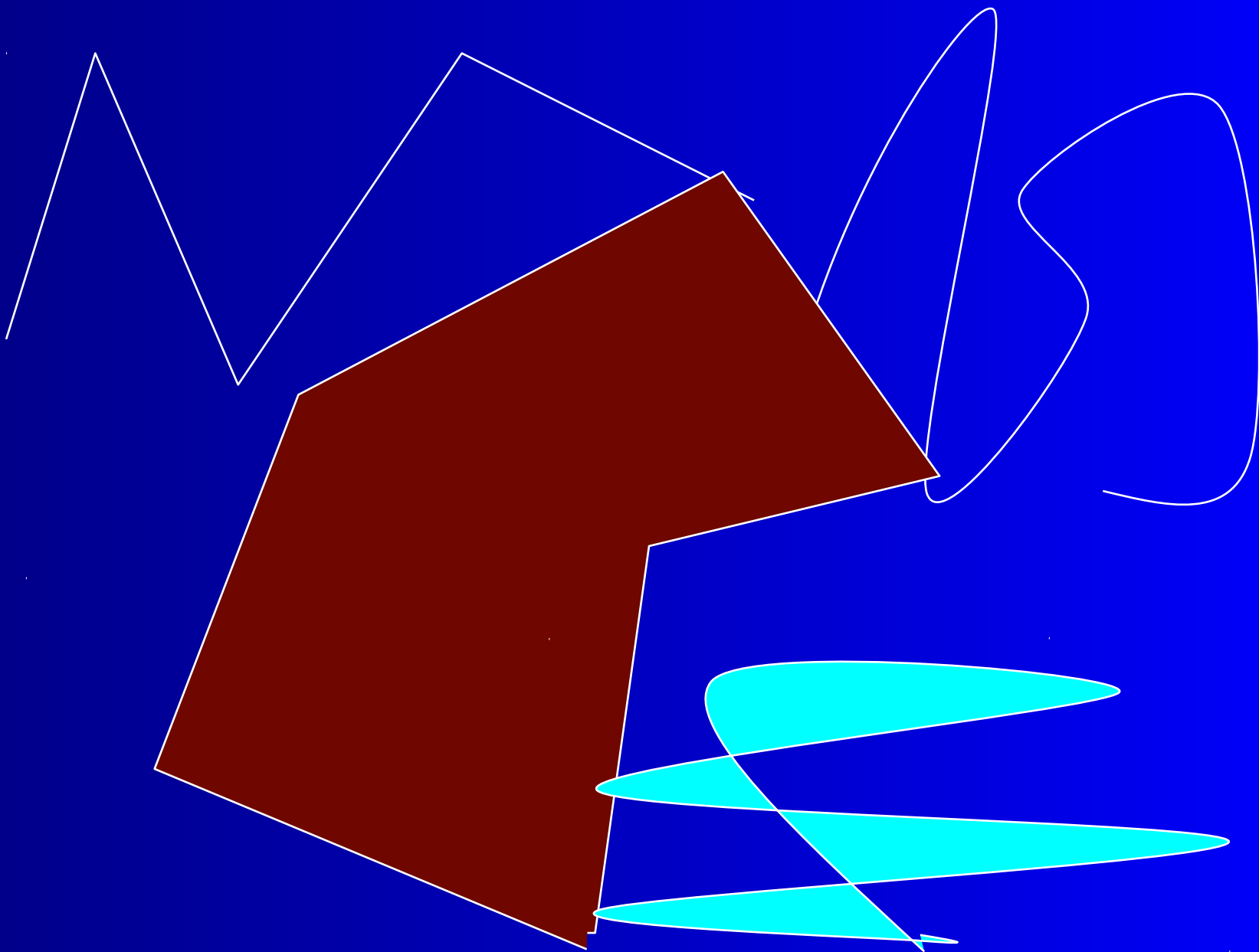
0 Flood

1 Write Flood(x,y-1); Flood(x,y+1); Flood(x-1,y); Flood(x+1,y);

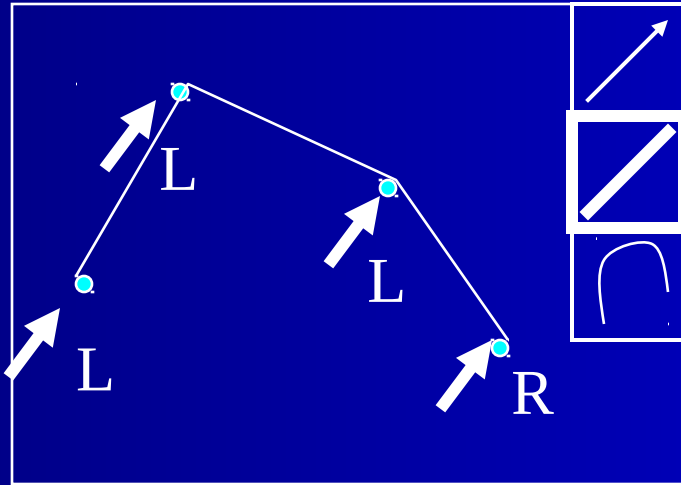
2 Write Flood(x,y-1); Flood(x,y+1);Flood(x-1,y);Flood(x+1,y);

2D grafikus rendszerek

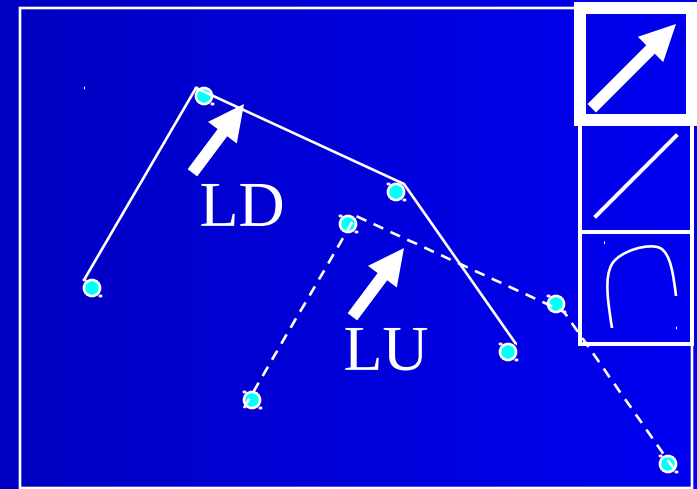
Szirmay-Kalos László



2D grafikus editor: GUI, use-case, dinamikus modell

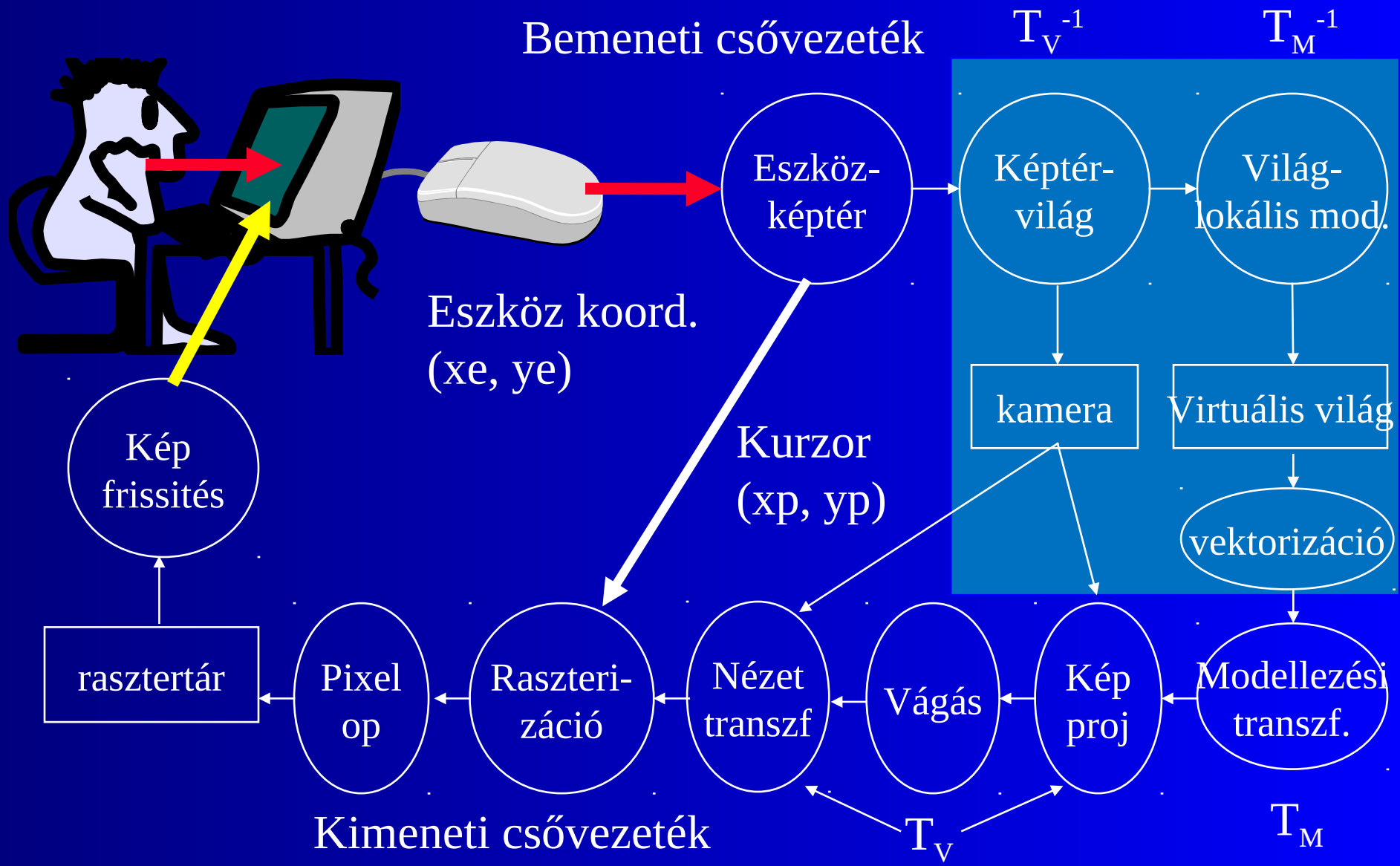


MouseDown	első pont
MouseDown	második
...	
MouseDown	n.
MouseDown	utolsó

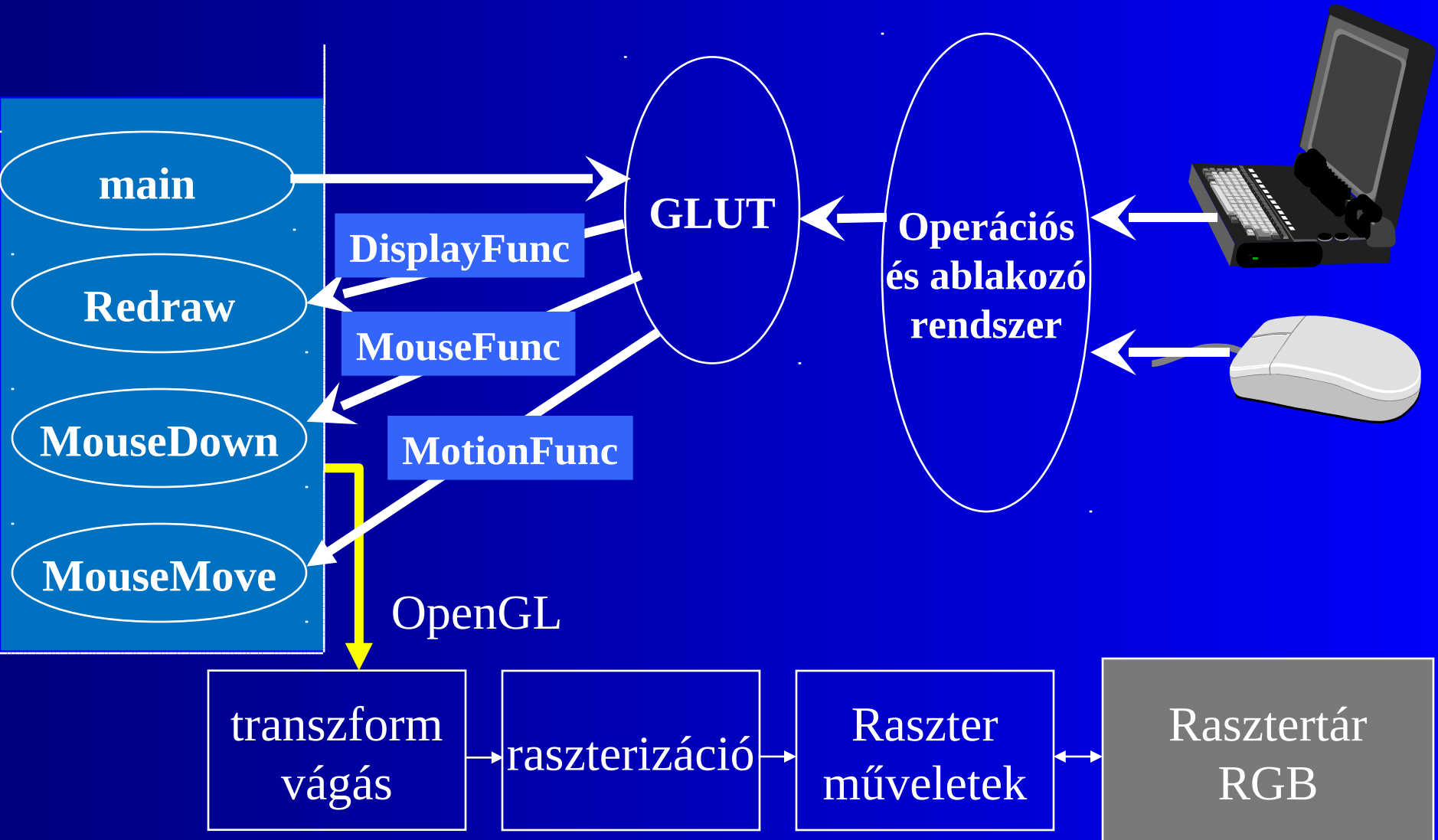


MouseDown	pick?
MouseMove	mozgat
MouseUp	letesz

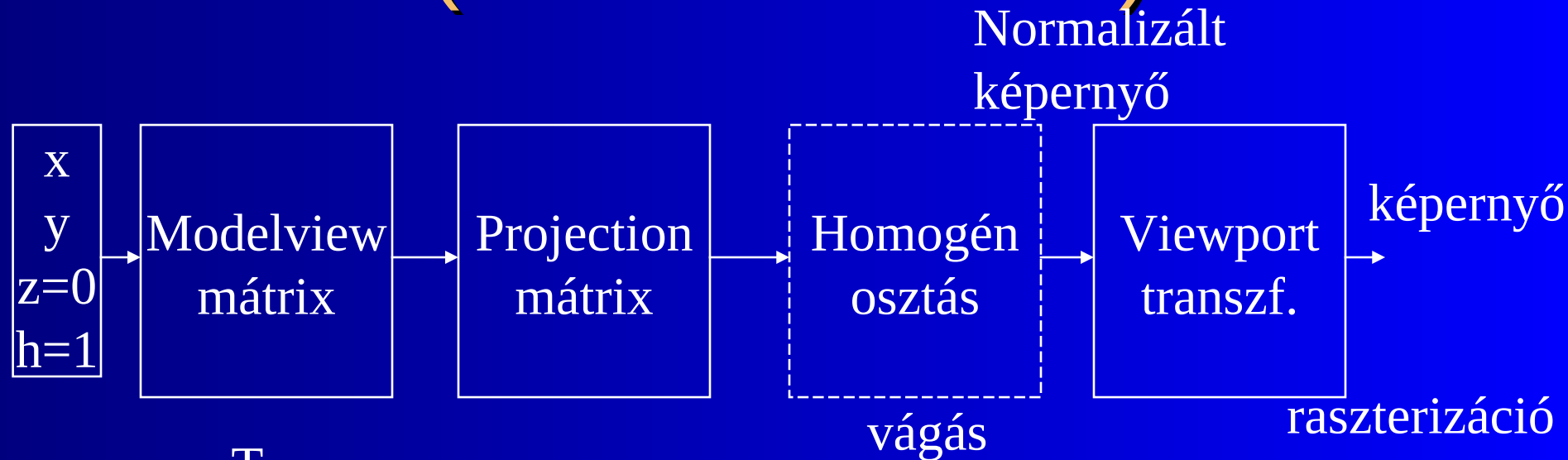
Funkcionális modell



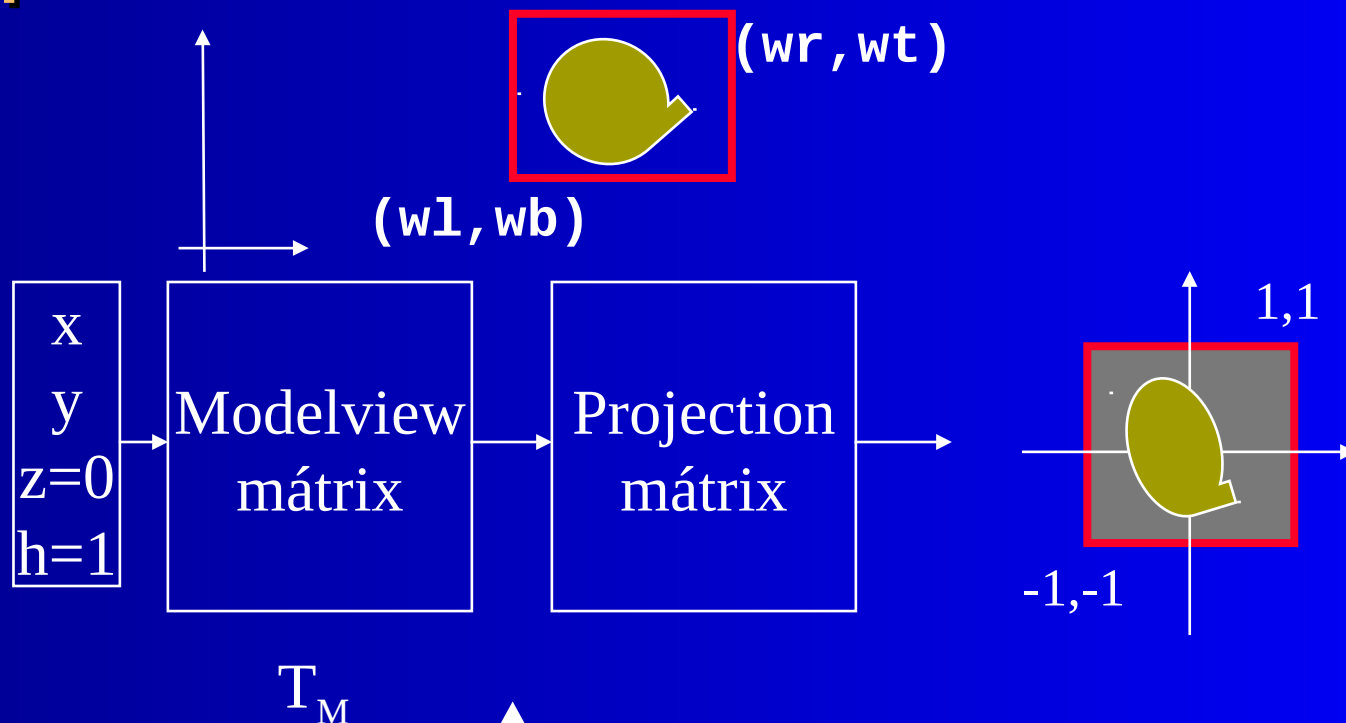
2D grafikus editor GLUT/OpenGL-lel



OpenGL transzformációk (2D-s utánérzés)



OpenGL transzformációk



```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity( );  
glTranslatef(px, py, 0);  
glRotatef(phi, 0, 0, 1);  
glScalef(sx, sy, 1);
```

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(wl, wr, wb, wt);
```

OpenGL mátrix

- OpenGL specifikáció:

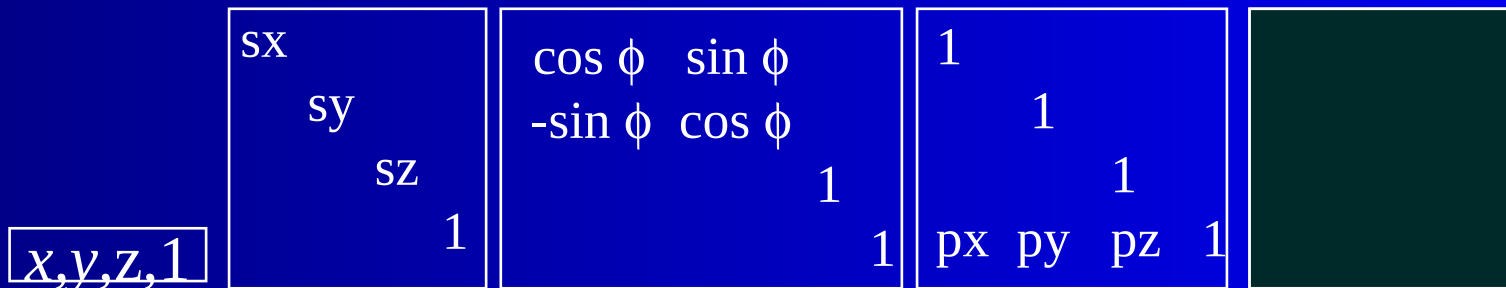
$$\begin{bmatrix} x' \\ y' \\ z' \\ h' \end{bmatrix} = \begin{bmatrix} T_0 & T_4 & T_8 & T_{12} \\ T_1 & T_5 & T_9 & T_{13} \\ T_2 & T_6 & T_{10} & T_{14} \\ T_3 & T_7 & T_{11} & T_{15} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ h \end{bmatrix}$$

- Mi:

$$\begin{bmatrix} x' & y' & z' & h' \end{bmatrix} = \begin{bmatrix} x & y & z & h \end{bmatrix} \cdot \begin{bmatrix} T_0 & T_1 & T_2 & T_3 \\ T_4 & T_5 & T_6 & T_7 \\ T_8 & T_9 & T_{10} & T_{11} \\ T_{12} & T_{13} & T_{14} & T_{15} \end{bmatrix}$$

Mátrix konkatenáció

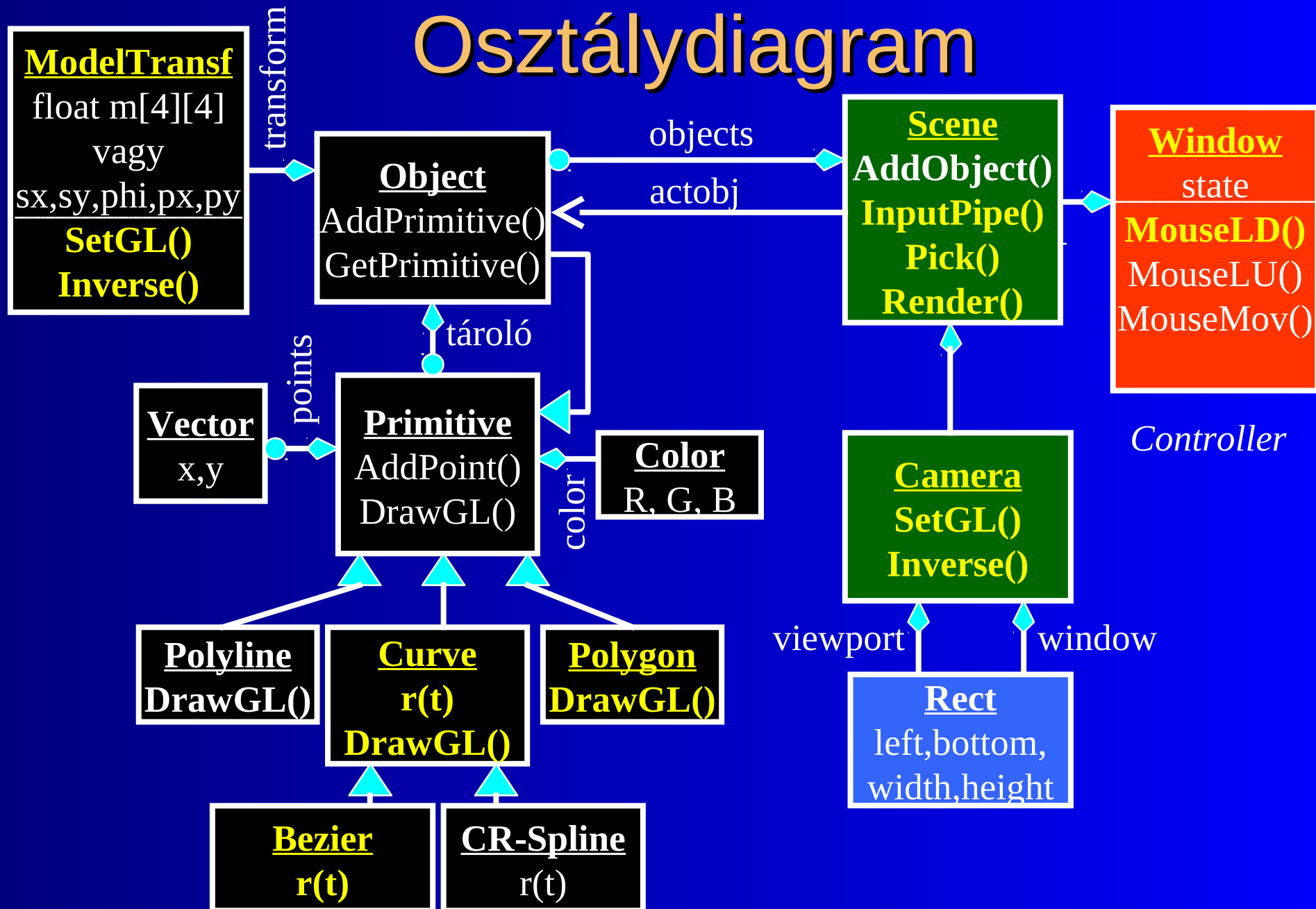
MODELVIEW



```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity( );  
glTranslatef(px, py, pz);  
glRotatef(phi, 0, 0, 1);  
glScalef(sx, sy, sz);  
.  
.  
.  
glVertex2f(x, y);
```



Osztálydiagram



GLUT: inicializálás

```
int main(int argc, char *argv[]) {  
    glutInit(&argc, argv);  
    glutInitWindowSize(appWinWidth, appWinHeight);  
    glutInitWindowPosition(100, 100);  
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);  
    glutCreateWindow("2D graphics editor");  
  
    glutMouseFunc(MouseDown); // callback  
    glutMotionFunc(MouseMove);  
    glutDisplayFunc(ReDraw);  
    glutReshapeFunc(Reshape);  
  
    glutMainLoop();           // fő hurok  
}
```

GLUT: eseménykezelés

Window window;

int appWinHeight, appWinWidth;

void ReDraw() {

glClearColor(0, 0, 0, 0); glClear(GL_COLOR_BUFFER_BIT);

window.scene.Render();

glutSwapBuffers();

}

void Reshape(int width, int height) {

appWinWidth = width; appWinHeight = height;

window.scene.camera.viewport = Rect(0, 0, width, height);

glutPostRedisplay();

}

void MouseDown(int button, int state, int x, int y) {

if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)

 window.MouseLD(x, appWinHeight - y); // Windows kr.

glutPostRedisplay();

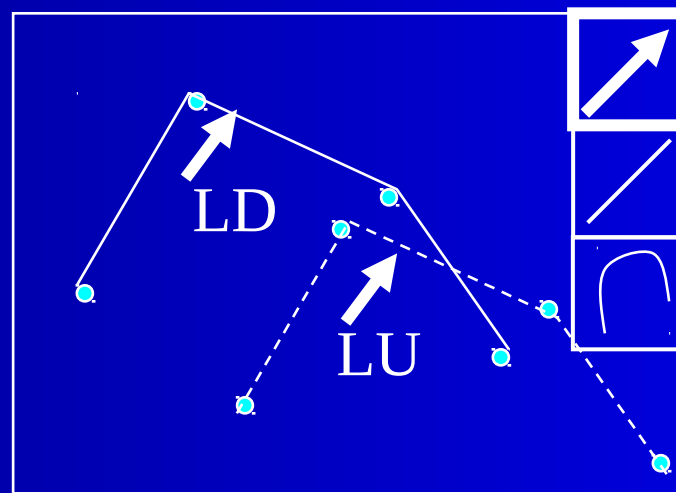
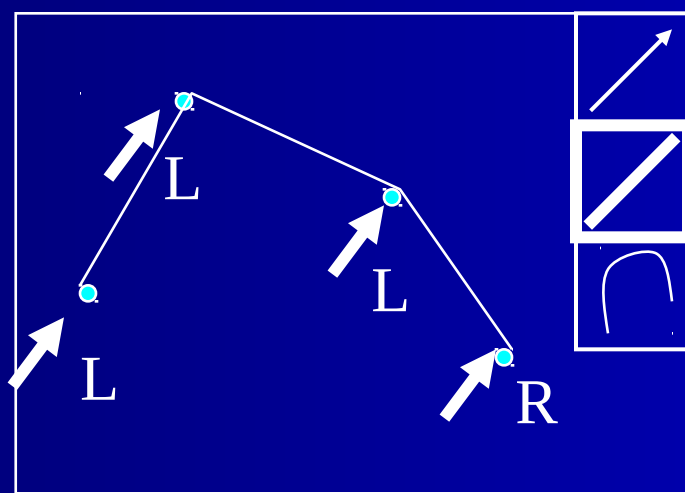
}

void MouseMove(int x, int y) {

 window.MouseMov(x, appWinHeight - y); // Windows kr.

glutPostRedisplay();

}



Állapotgép

state	MouseLD	MouseRD	MouseMove	MouseLU	Lmenu	Pmenu
LINE	actobj = AddObj() InputPipe(x,y) state = DRAW					state=PICK
DRAW	InputPipe(x,y) Render() state = DRAW;	InputPipe(x,y) Render() state = LINE;			_____	_____
PICK	if (Pick(x,y)) actobj=picked state = MOVE				state=LINE	
MOVE			actobj->transf Render() state=MOVE	state=PICK		

MouseLD

Scene
AddObject()
InputPipe()
Pick()
Render()

Window
state
MouseLD()
MouseLU()
MouseMov()

```
struct Window {
```

```
    Scene scene;
```

```
    enum {LINE, DRAW, PICK, MOVE} state;
```

```
void MouseLD( int X, int Y ) {
```

```
    switch (state) {
```

```
        case LINE:    scene.AddObject();  
                      scene.InputPipe(X, Y);  
                      state = DRAW;  
                      break;
```

```
        case DRAW:    scene.InputPipe(X, Y);  
                      scene.Render( );  
                      state = DRAW;  
                      break;
```

```
        case PICK:    if (scene.Pick(X,Y) != NULL) state = MOVE;  
                      break;
```

```
    }
```

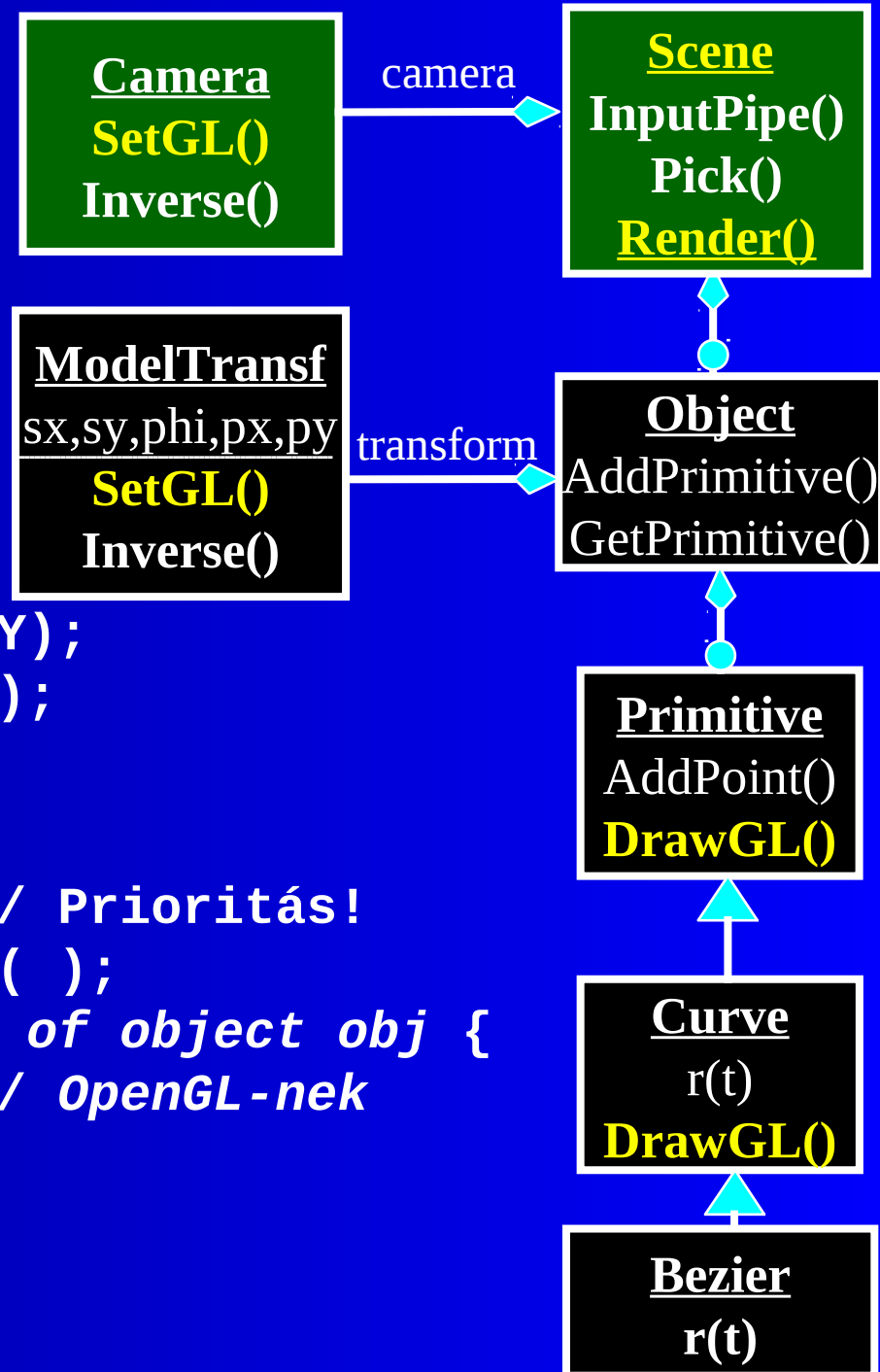
```
}
```

```
};
```

Scene :: Render

```

class Scene {
    Object * objects;
    Object * actobj;
    Camera camera;
public:
    void AddObject( );
    void InputPipe(int X, int Y);
    Object * Pick(int X, int Y);
    void Render ( ) {
        camera.SetGL();
        for each object obj { // Prioritás!
            obj->transform.SetGL( );
            for each primitive p of object obj {
                p -> DrawGL( ); // OpenGL-nek
            }
        }
    }
};
    
```



2D Camera

```
class Camera {
```

```
    Rect viewport, window;
```

```
void SetGL( ) {
```

```
    glViewport(viewport.left, viewport.bottom,  
               viewport.width, viewport.height);
```

```
    glMatrixMode( GL_PROJECTION );
```

```
    glLoadIdentity();
```

```
    gluOrtho2D(window.left, window.left + window.width,  
               window.bottom, window.bottom + window.height);
```

```
}
```

```
Vector Inverse( Vector scrPos ) {
```

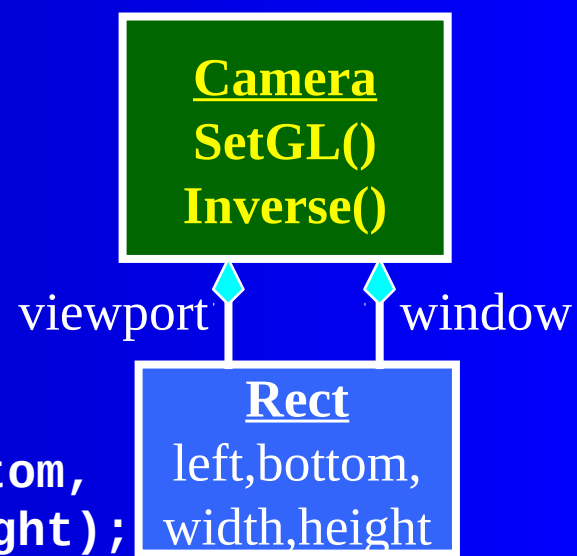
```
    float sx = window.width / viewport.width;
```

```
    float sy = window.height / viewport.height;
```

```
    return Vector(  
        (scrPos.x - viewport.left) * sx + window.left,  
        (scrPos.y - viewport.bottom) * sy + window.bottom  
    );
```

```
}
```

```
};
```



ModelTransf

ModelTransf
sx,sy,phi,px,py
SetGL()
Inverse()

```
class ModelTransf {  
    float sx, sy, phi;  
    Vector p;  
public:  
    void SetGL( ) {  
        glMatrixMode(GL_MODELVIEW);  
        glLoadIdentity( );  
        glTranslatef(p.x, p.y, 0);  
        glRotatef(phi, 0, 0, 1);  
        glScalef(sx, sy, 1);  
    }  
  
    Vector Inverse( Vector wPos ) {  
        wPos -= p;    // inverse translation;  
        mx = wPos.x * cos(-phi) - wPos.y * sin(-phi);  
        my = wPos.x * sin(-phi) + wPos.y * cos(-phi);  
        return Vector( mx/sx, my/sy );  
    }  
};
```

Curve rajzolás

```
class Curve : public Primitive {
```

```
    float tstart, tend;
```

```
public:
```

```
    virtual Vector r( float t ) = 0;
```

```
void DrawGL( ) {
```

```
    glColor3f( color.R, color.G, color.B );
```

```
    glBegin( GL_LINE_STRIP );
```

```
    for(int i = 0; i < NVECTOR; i++) {
```

```
        float t = tstart + (tend - tstart) * i / NVECTOR;
```

```
        Vector p = r( t );
```

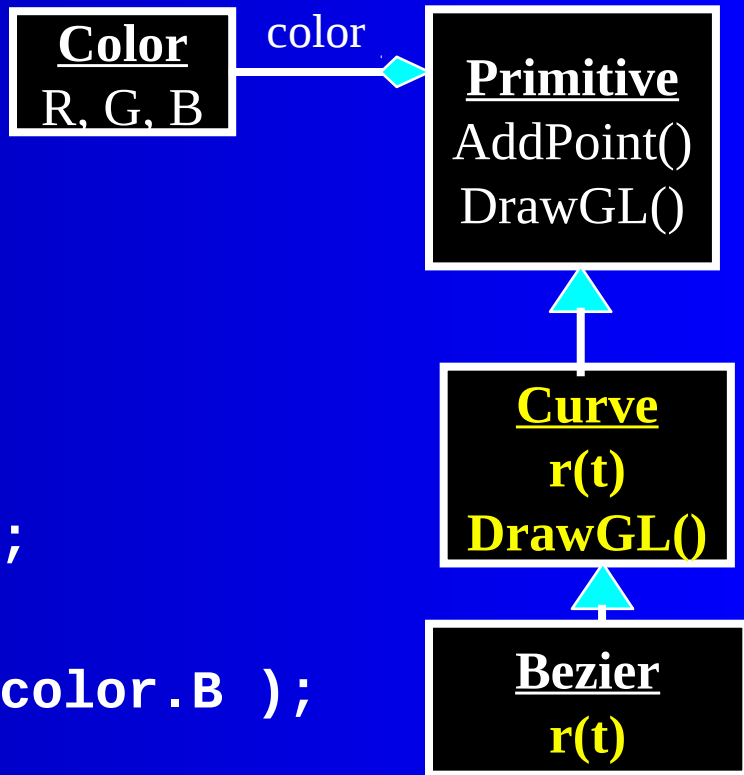
```
        glVertex2f(p.x, p.y);
```

```
    }
```

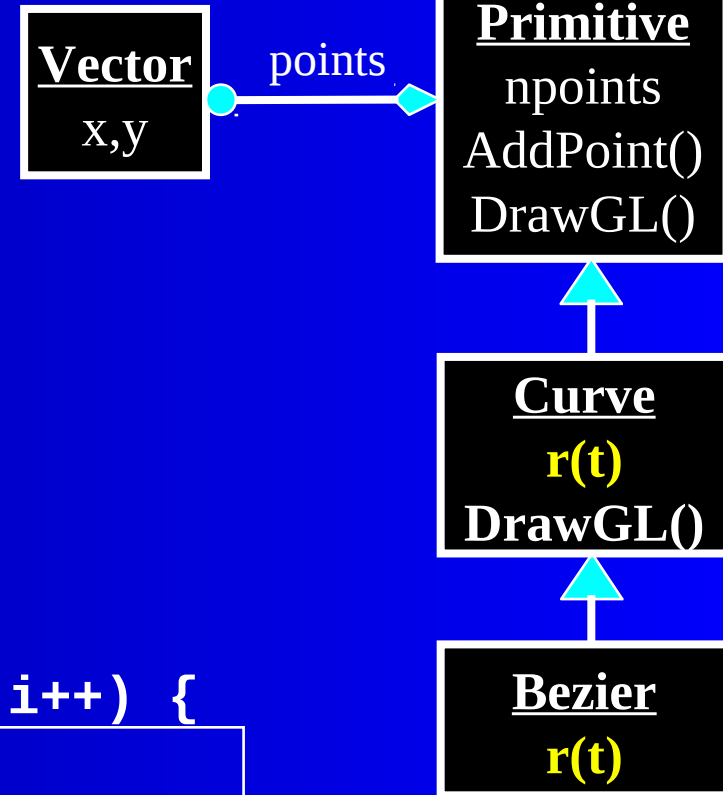
```
    glEnd( );
```

```
}
```

```
};
```



Bezier görbe: $r(t)$



```
class Bezier : public Curve {  
public:
```

```
    Vector r( float t ) {
```

```
        Vector rr(0, 0);
```

```
        int n = npoints - 1;
```

```
        for(int i = 0; i < npoints; i++) {
```

```
            float Bi = 1;
```

```
            for(int j = 1; j <= i; j++)
```

```
                Bi *= (float)(n-j+1)/j;
```

```
            Bi *= pow(t, i) * pow(1-t, n-i);
```

```
            rr += points[i] * Bi;
```

```
        }
```

```
        return rr;
```

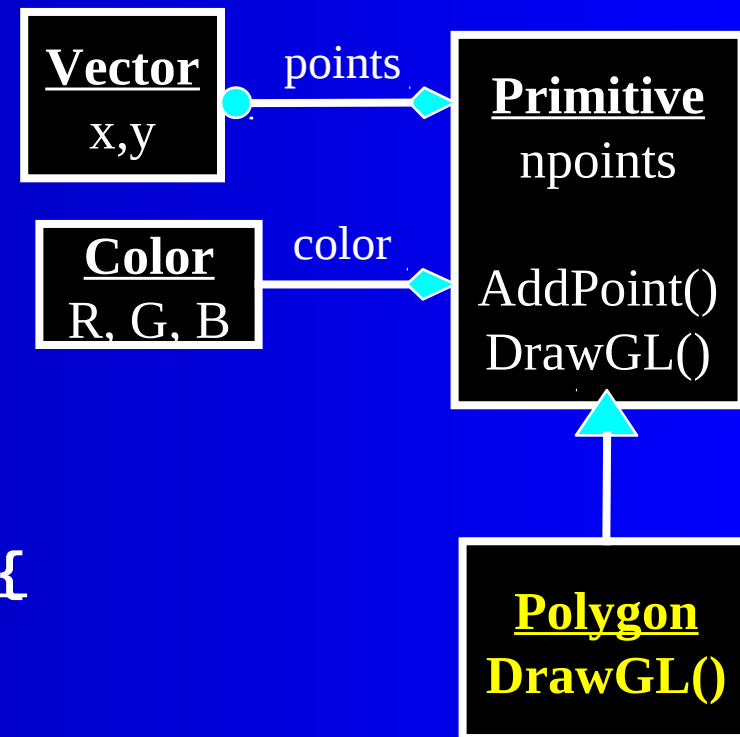
```
    }
```

```
};
```

$$r(t) = \sum B_i(t) r_i$$

$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

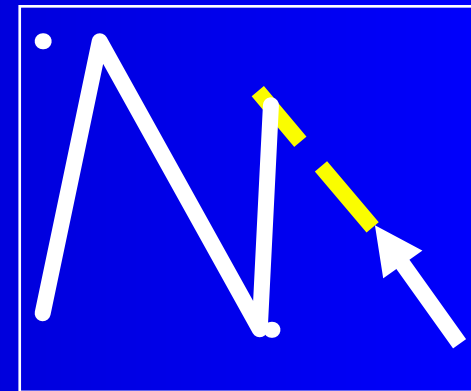
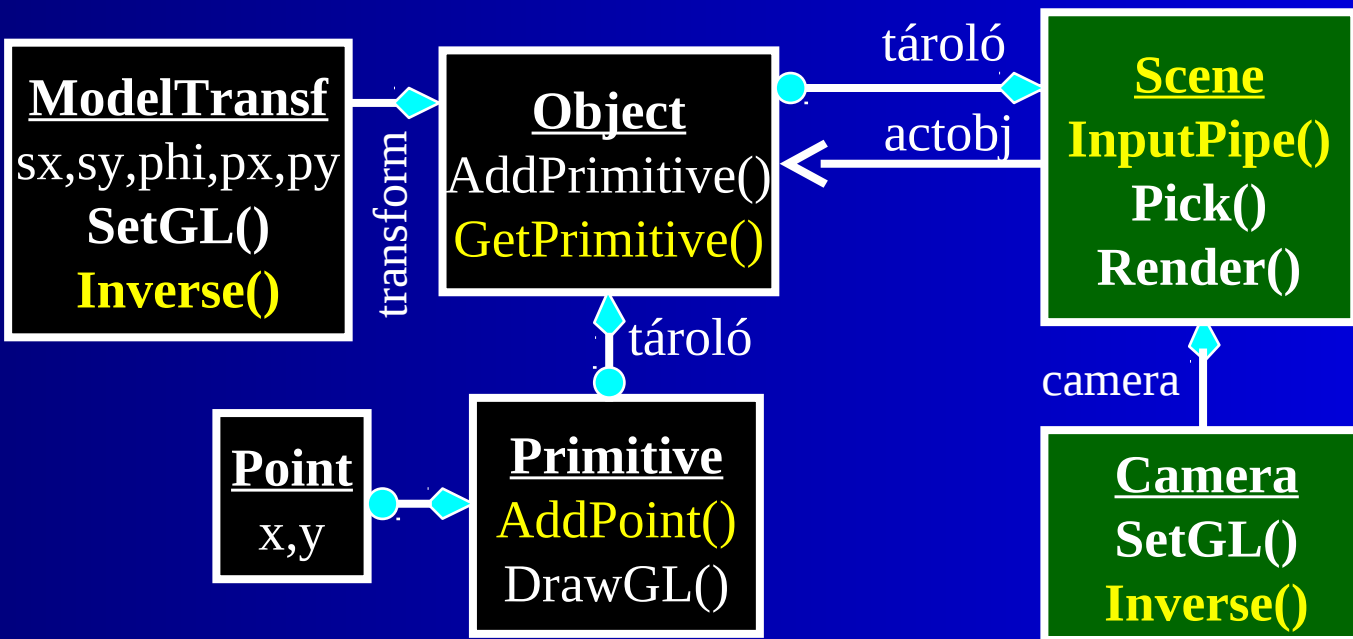
(Konvex) poligon rajzolás



```
class Polygon : public Primitive {
public:
    void DrawGL( ) {
        glColor3d( color.R, color.G, color.B );
        glBegin( GL_POLYGON ); //GL_TRIANGLE_FAN
        for(int i = 0; i < npoints; i++) {
            glVertex2f(points[i].x, points[i].y);
        }
        glEnd( );
    }
};
```

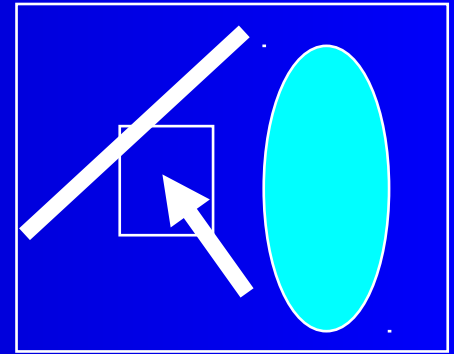
Bemeneti csővezeték: pontok beépítése a virtuális világba

```
void Scene :: InputPipe( int X, int Y ) {  
    Vector wPos = camera.Inverse( Vector(X, Y) );  
    Vector mPos = actobject -> transform.Inverse(wPos);  
    actobject -> GetPrimitive( ) -> AddPoint(mPos);  
}
```



Kiválasztás (Pick)

```
Scene :: Pick( int X, int Y ) {  
    camera.SetGL();  
    actobj = NULL;  
    for each object obj {  
        obj->transform.SetGL( );  
        for each primitive p of object obj {  
            Color pushCol = p -> color;  
            p -> color = ToColor( obj );  
            p -> DrawGL( ); // OpenGL-nek  
            p -> color = pushCol;  
        }  
    }  
    unsigned char ima[10 * 10 * 3];  
    glReadPixels(X-5,Y-5,10,10,GL_RGB,GL_UNSIGNED_BYTE,&ima[0]);  
    for(int x = 0; x < 10; x++)  
        for(int y = 0; y < 10; y++) {  
            int idx = 3*(10 * y + x);  
            obj = FromColor(ima[idx], ima[idx+1], ima[idx+2]);  
            if (obj != NULL) actobj = obj;  
        }  
}
```



2. feladat

Készítsen 2D editort, amivel egy 100m x 100m-es területet lehet berendezni, amelybe egy 50m x 50m-es ablakú kamerával nézünk bele. Az ablakot minden 's' lenyomásra a (10, 20) vektor tolja el. Ha az ablak elérte a terület szélét, akkor visszaugrik a kiindulási állapotba. A program az egér balgomb lenyomásokat kontrollpontnak, a gomblenyomás idejét pedig csomóértéknek (kontrollponthoz rendelt paraméterértéknek) tekinti. A kontrollpontokra egy fehér Tenziós Catmull-Rom (TCR) spline-t illeszt, ahol a tenzió mindenhol -0.5. A kontrollpontok nyugalmi állapotukban 1m sugarú piros körök.

Space lenyomására a program aktivizálódik, a spline-t megduplázza és az új változatot 60 fokkal a kontrollpontok súlypontja körül elforgatja, majd kék Bézier görbét csinál belőle (mialatt az eredeti TCR marad a helyén). Ezután egy-egy 1 m sugarú sárga kör indul el minden görbén, mégpedig a TCR -n a paraméterezésnek megfelelő időzítéssel. A Bézier-t a saját köre úgy futja be, hogy ugyanakkor érkezzen a végére, mint a TCR -n szaladó társa. A futamok újra kezdődnek. A futás alatt a kontrollpontokat ábrázoló körök sugara folyamatosan változik amivel azt mutatják, hogy milyennek kell lennie az ide elhelyezett súly abszolút értékének, hogy a görbe aktuális pontját a kontrollpontok kombinációjaként kapjuk meg. Ha a súly pozitív, a kör piros. Ha negatív, akkor türkisz kék.

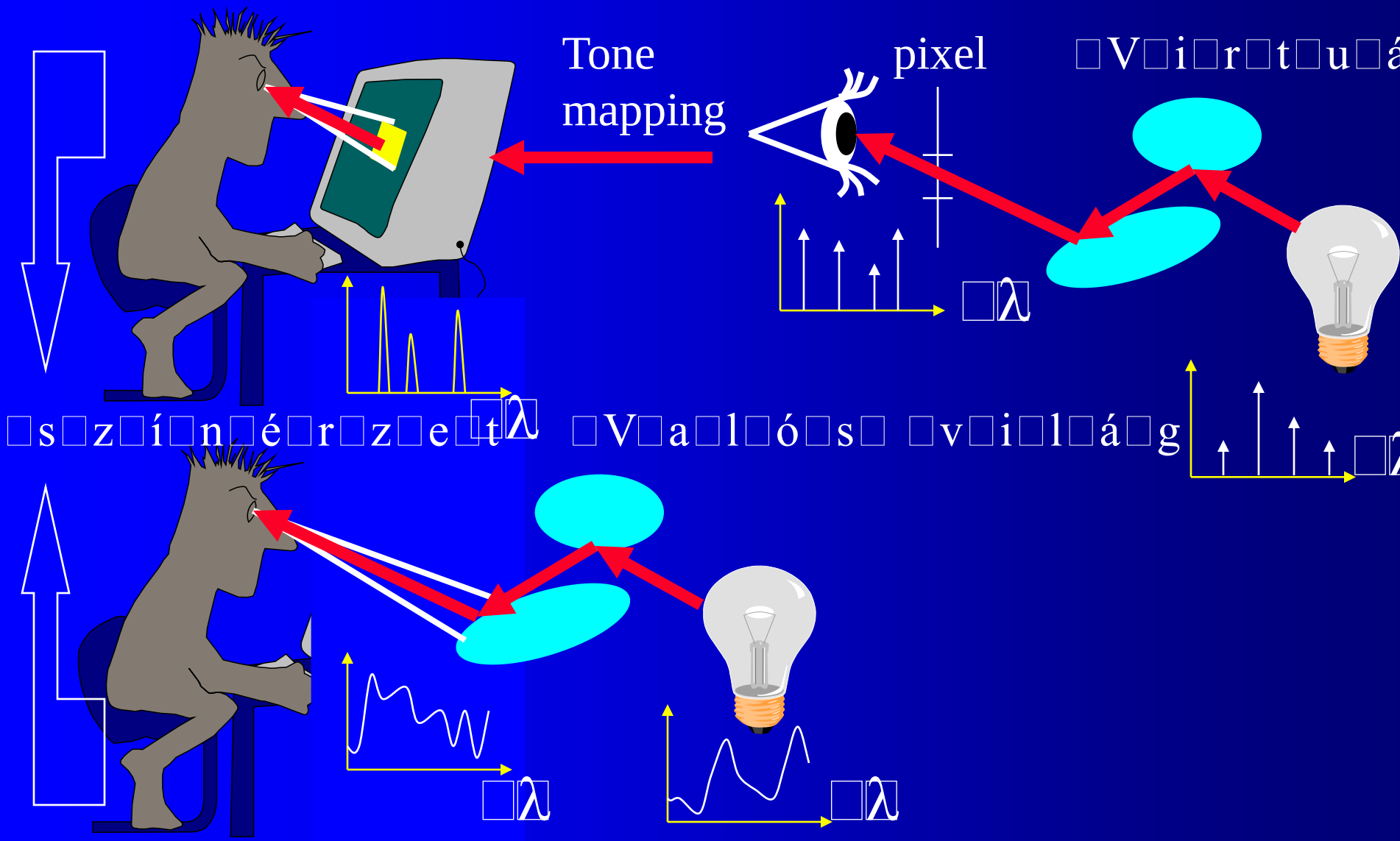
*Science is either physics or
stamp collecting.*

Rutherford

3D képszintézis fizikai alapmodellje

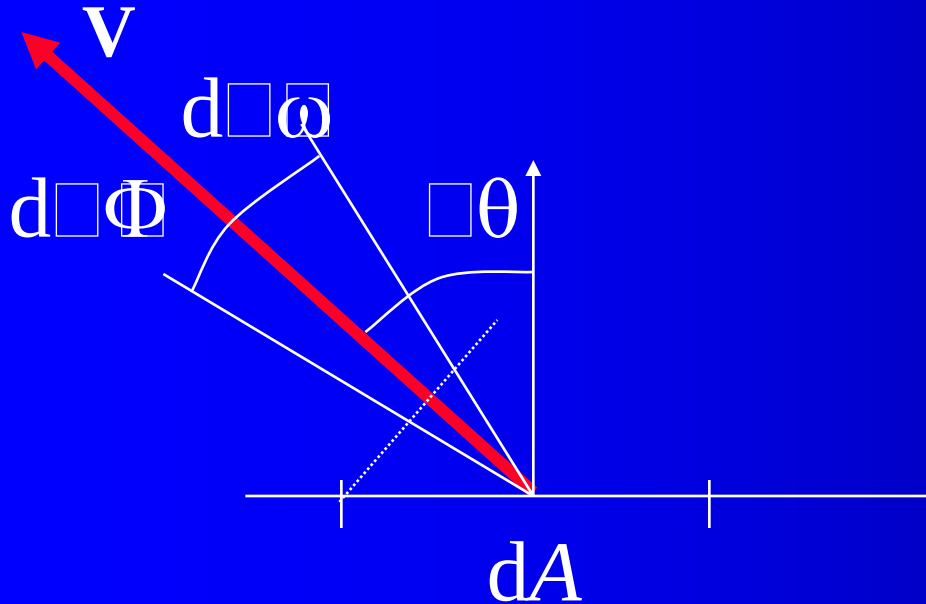
Szirmay-Kalos László

Képszínezés a valós világ izidja

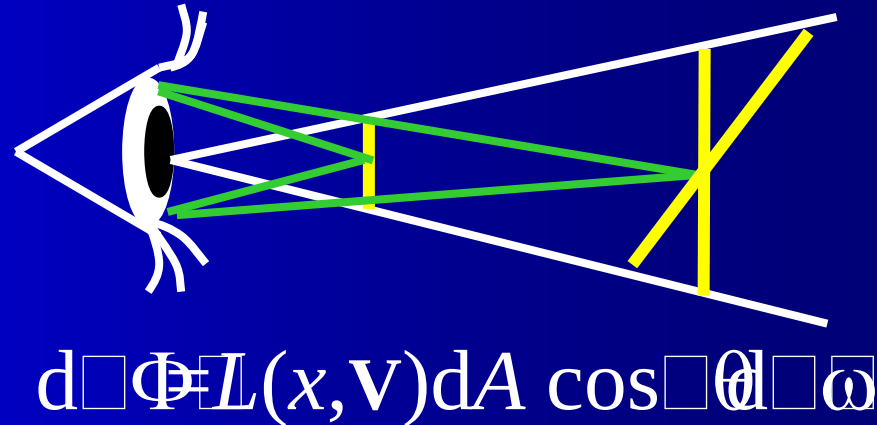


Sugárssűrűség (Radiance): $L(x, y, \theta, \phi)$

● Egy egyenesen elterjedő sugár energiájának a területegységre és a szögterületre vonatkozó arányát nevezzük sugárssűrűségnek.
[Watt/ sr/ m²]



$$L(x, y, \theta, \phi) = \frac{d\Phi}{dA \cos\theta d\Omega}$$



$$d\Phi = L(x, y, \theta, \phi) dA \cos\theta d\Omega$$

Fény-nyület kölcsönhatás kó

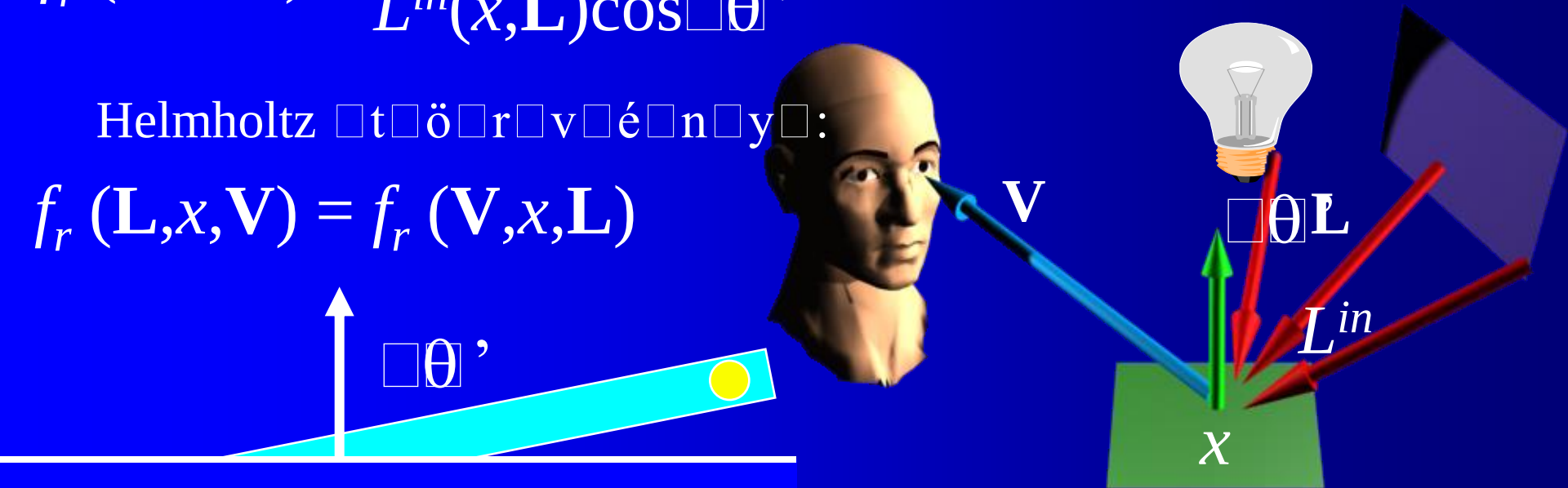
Sugárzás Csr Csé gGeometriaBejő

$$L(x, V) = L^{in}(x, L) f_r(L, x, V) \cos \theta,$$

$$f_r(L, x, V) \stackrel{\text{def}}{=} \frac{L(x, V)}{L^{in}(x, L) \cos \theta},$$

Helmholtz térvétele:

$$f_r(L, x, V) = f_r(V, x, L)$$

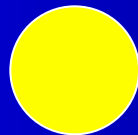
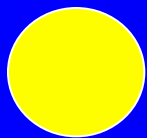


k ü l l ö n k e z e



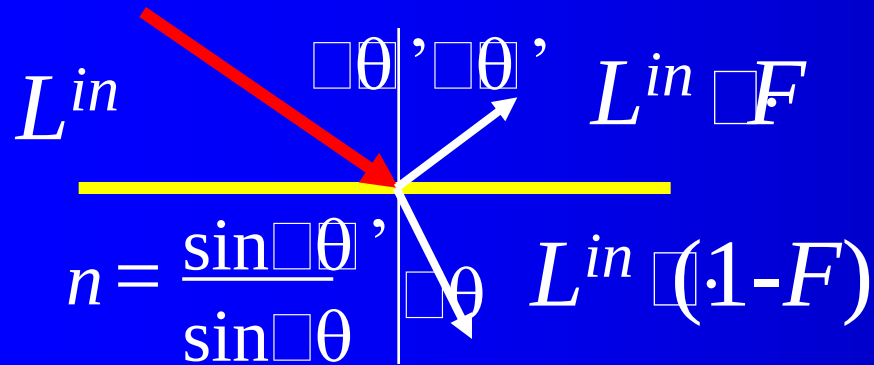
- Elnyel 5 dési való

Nem mindig van így!



- Gamma fotonnál a relatív elektron tömegével $E = m_e c^2$
- Foton energia ($h \cdot \nu$) általában sokkal kisebb, mint a Compton hullámhossz
- Foszforészeknél, fémeknél kezeletlen 5 keV a $h \cdot \nu$

Sima felület Fresnel egyenletek

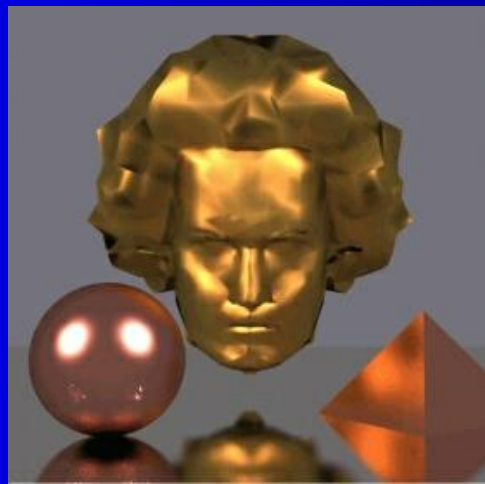
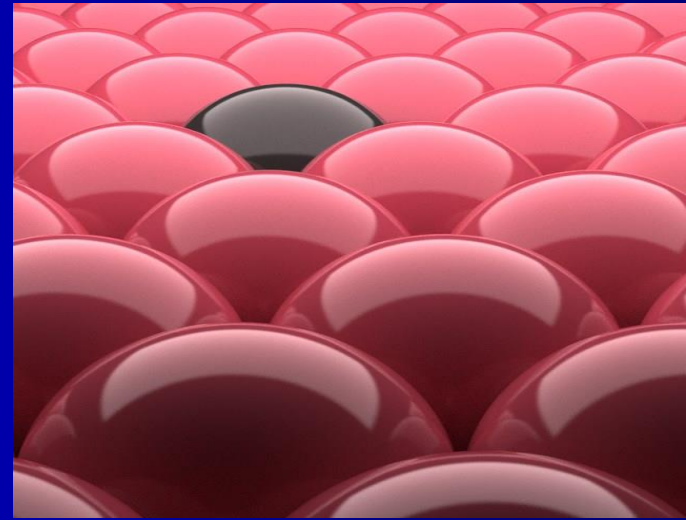
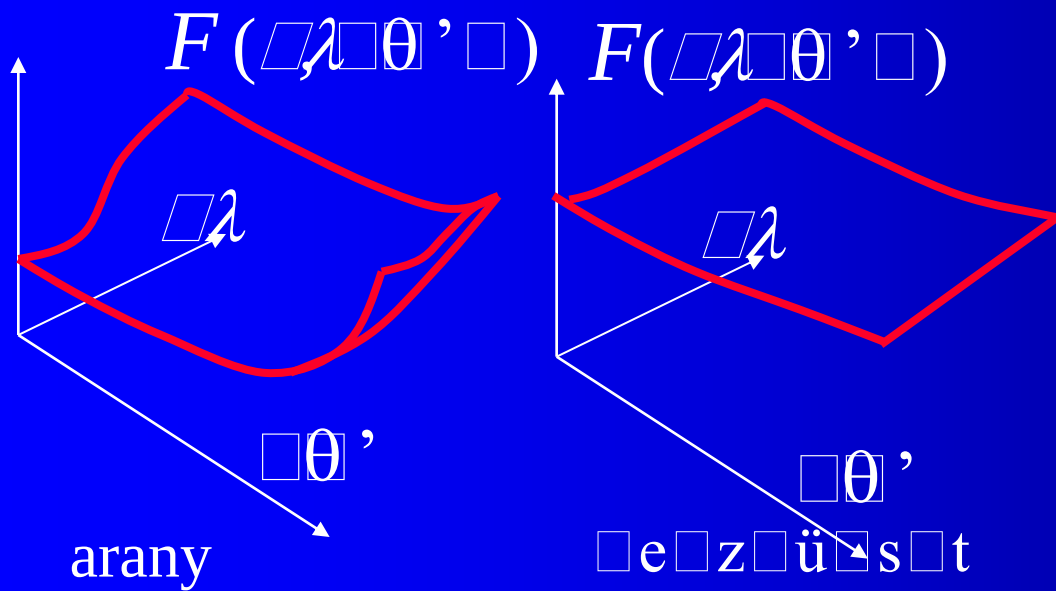


Sima $\approx$ 1 pixelben látható a felület s

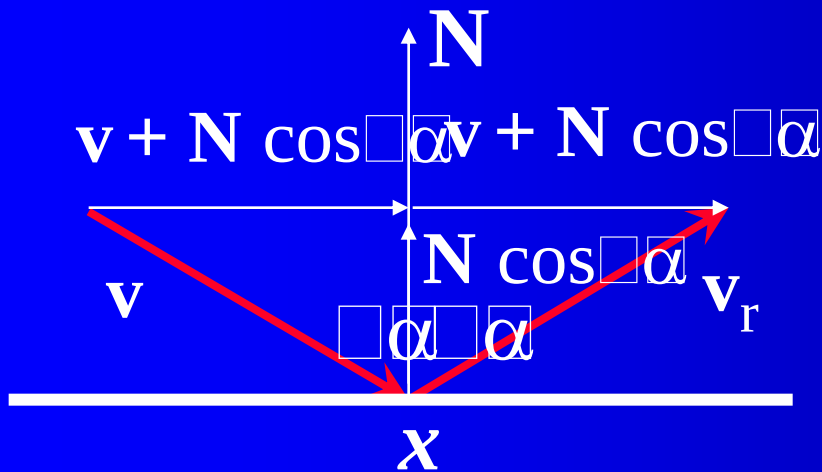
$$F = \frac{1}{2} \left| \frac{\cos \theta (n+kj) \cos \theta'}{\cos \theta (n+kj) \cos \theta'} \right|^2 + \frac{1}{2} \left| \frac{\cos \theta' (n+kj) \cos \theta}{\cos \theta' (n+kj) \cos \theta} \right|^2$$

$$F \approx F_0 + (1-F_0) (1-\cos \theta)^5, \quad F_0 = \frac{(n-1)^2 + k^2}{(n+1)^2 + k^2}$$

Fresnel függvény



Tükörirány számítás



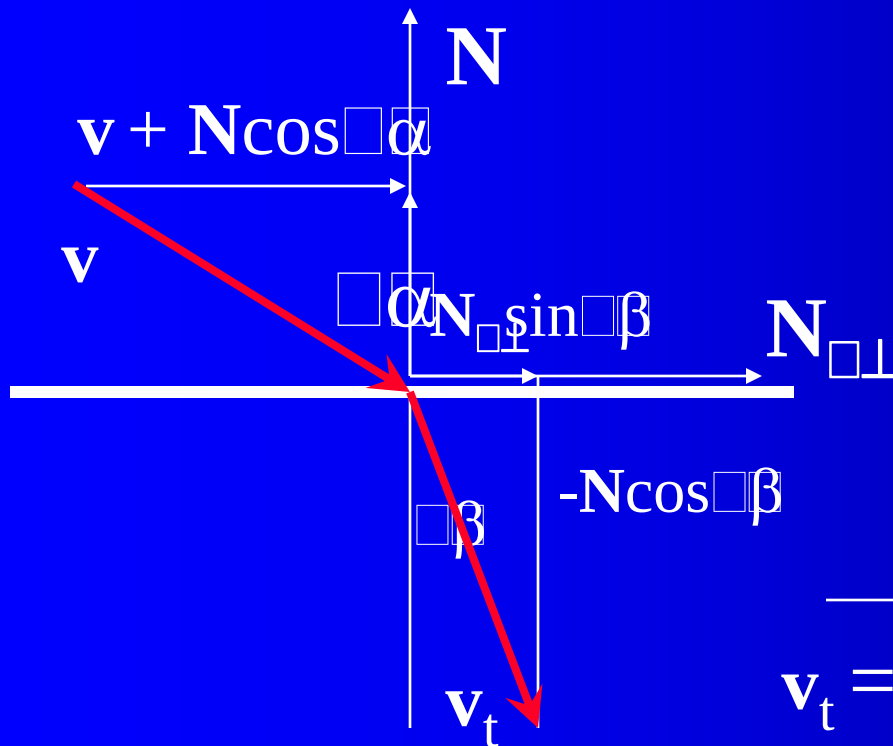
$$\cos \alpha = -(\mathbf{v} \cdot \mathbf{N})$$

$$\mathbf{v}_r = \mathbf{v} + 2 \mathbf{N} \cos \alpha$$

$$\mathbf{L} = \mathbf{v}_r, \mathbf{V} = \mathbf{v}$$

```
ReflectDir( L, N, V) {  
    L = V - N * (N * V) * 2;  
}
```

Törési törvény síri



$$n = \frac{\sin \alpha}{\sin \beta}$$

Snellius-
Descartes

$$N_{\perp} = \frac{v \cdot N \cos \alpha}{\sin \alpha}$$

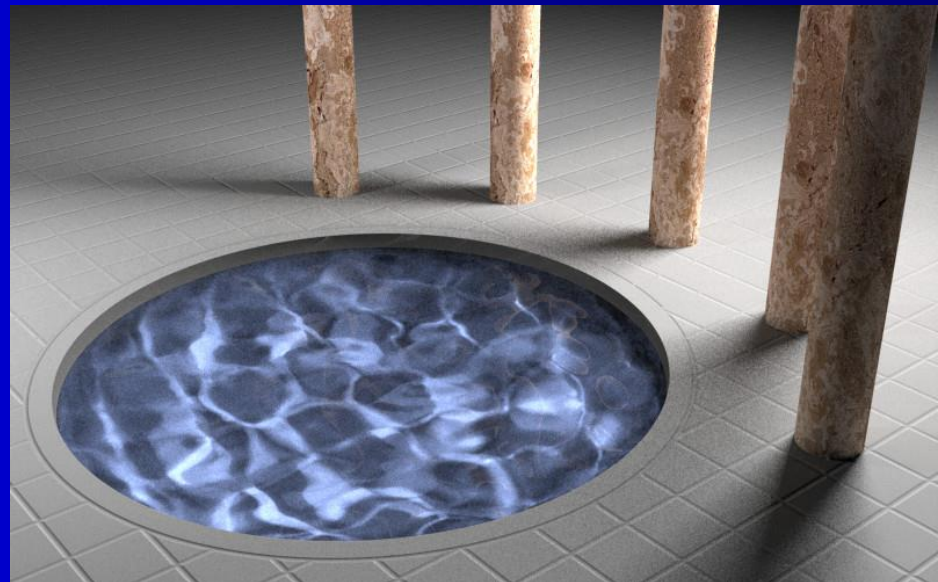
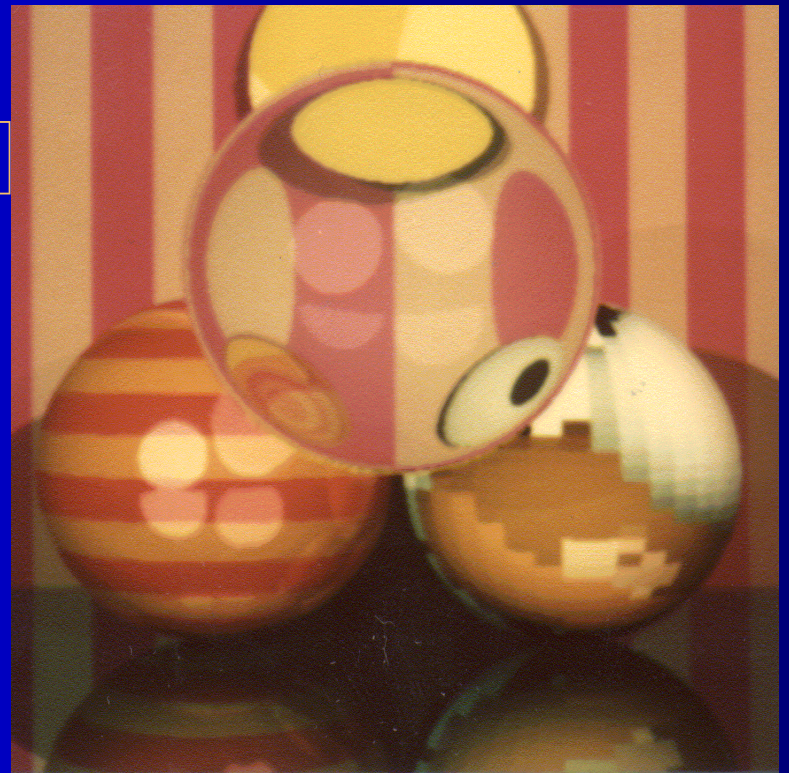
$$v_t = N_{\perp} \sin \beta - N \cos \beta$$

$$v_t = v \cdot n + N (\cos \alpha - \sqrt{1 - \sin^2 \beta})$$

$$\cos \beta = \sqrt{1 - \sin^2 \beta} = \sqrt{1 - \sin^2 \alpha / n^2}$$

$$v_t = v \cdot n + N (\cos \alpha - \sqrt{1 - (1 - \cos^2 \alpha) n^2})$$

Fényörny anyagok



```
class SmoothSurface {  $\frac{(n-1)^2 + k^2}{(n+1)^2 + k^2}$ 
```

```
    Color F0;    // r,g,b
    float n;     // we ignore here the wavelength dependency
```

```
public:
```

```
    void ReflectionDir(Vector& R, Vector& N, Vector& V) {
```

```
        float cosa = -N * V;
```

```
        R = V + N * cosa * 2;
```

$$\mathbf{v}_t = \mathbf{v}n' + \mathbf{N} (\cos\alpha - \sqrt{1 - (1 - \cos^2\alpha)n^2})$$

```
    }
```

```
    BOOL RefractionDir(Vector& T, Vector& N, Vector& V) {
```

```
        float cosa = -N * V, cn = n;
```

```
        if (cosa < 0) { cosa = -cosa; N = -N; cn = 1/n; }
```

```
        float disc = 1 - (1 - cosa * cosa) / cn / cn;
```

```
        if (disc < 0) return FALSE;
```

```
        T = V / cn + N * (cosa / cn - sqrt(disc));
```

```
        return TRUE;
```

```
    }
```

```
    Color Fresnel(Vector& N, Vector& V) {
```

```
        float cosa = fabs(N * V);
```

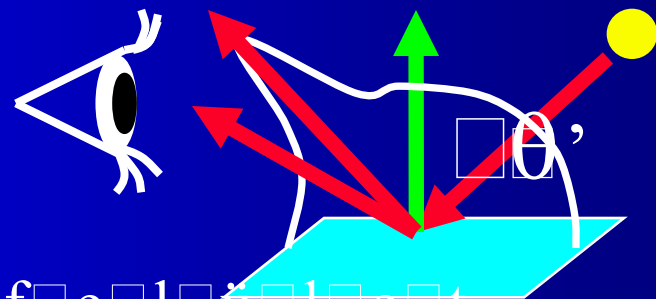
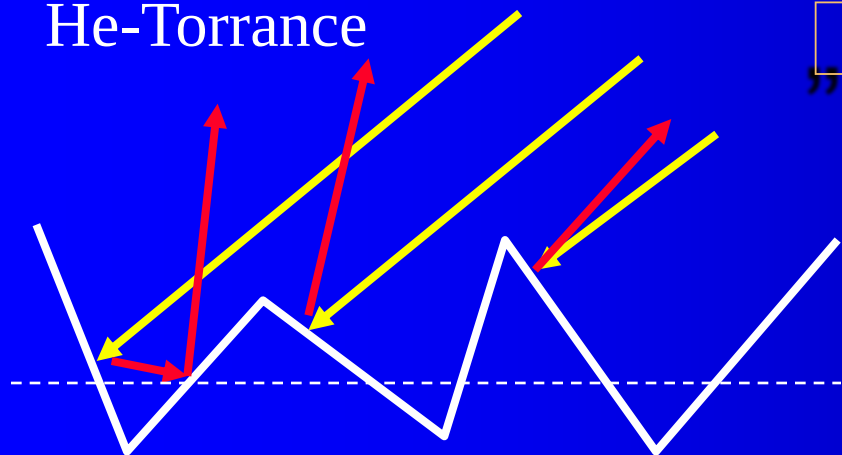
```
        return F0 + (1-F0) * pow( 1-cosa, 5 ); //  $F \approx F0 + (1-F0) (1-\cos\theta)^5$ 
```

```
    }
```

```
};
```

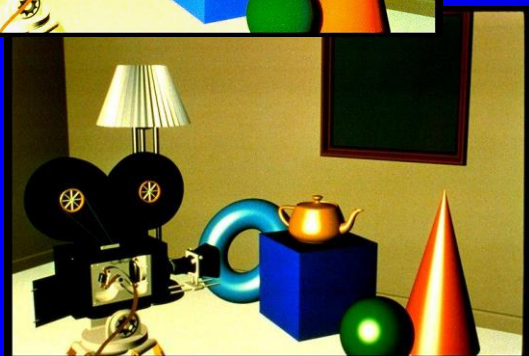
Cook-Torrance
He-Torrance

„Rückwärts“

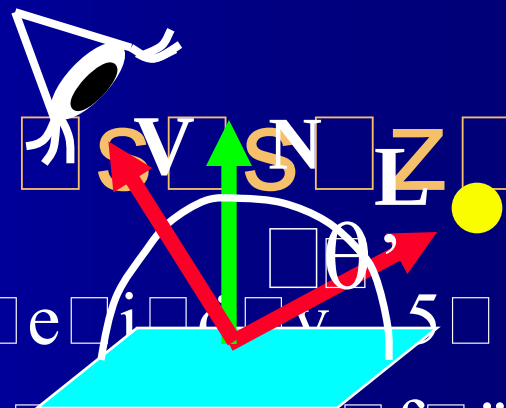


1 pixelben látható felület

Mi: viselked



Díffúziós visszaverődés



● Radián BRDF $\cos \theta$ Bejövő

anézeti irányított fű

● A BRDF anézeti irányított fű

● Helmholtz: a BRDF r

● A BRDF irányított fű g g e t l

$$f_r(\mathbf{L}, x, \mathbf{V}) = k_d(x, \lambda)$$

● Díffúziós visszaverő 5d

– sokszor a moya gféknöyl csönha t

– színes

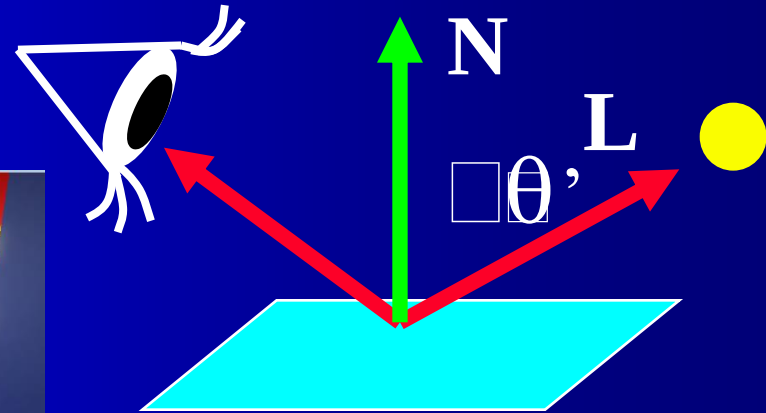
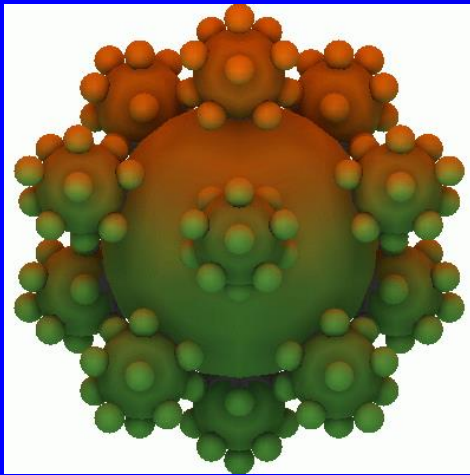
Lambert's cosine law

● Point / Irradiance $L_{ref} = n L_{in} \cos \theta$

BRDF Irradiance f_{λ}

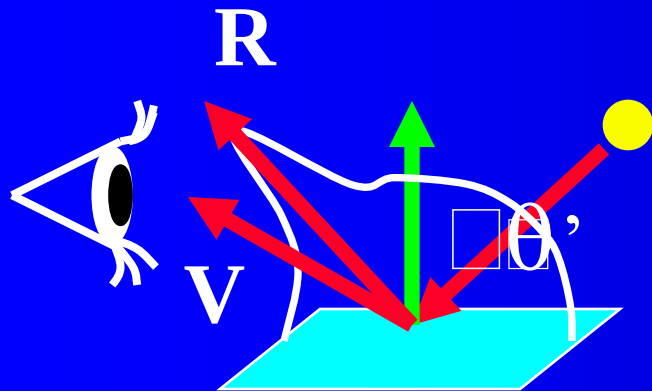
DE as $\cos \theta$ N L

me $\cos \theta$ N L

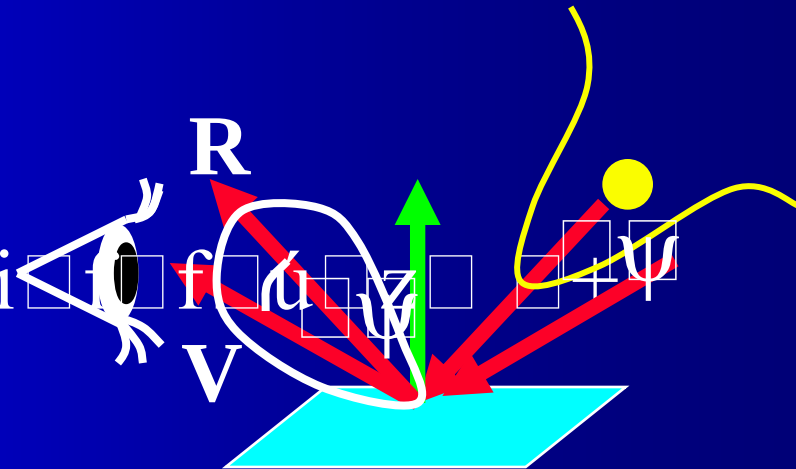


Speculáris visszavertés: a Phong modell

$$\cos \psi = \mathbf{R} \cdot \mathbf{V}$$



$$\cos \theta = \mathbf{R} \cdot \mathbf{V}$$

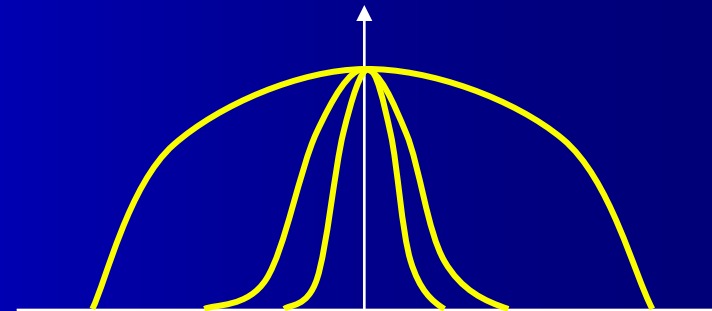


Kétféle egyenletes eloszlás, egy az irány, egy a szín

$$L^{ref} = L^{in} k_s (\cos^+ \psi)^n$$

$$f_r(\mathbf{L}, \mathbf{x}, \mathbf{V}) = k_s \cos^n \psi \cos \theta / \theta$$

Nem szimmetrikus!



Diffrakció + Phong anyagok

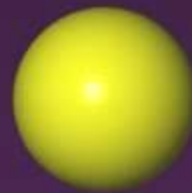
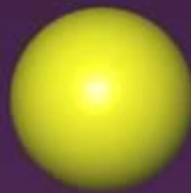
diffrakció



Phong



diffrakció + Phong



$n =$

5

10

20

50

Sokszor anyag

kölcsön

„Saját” s

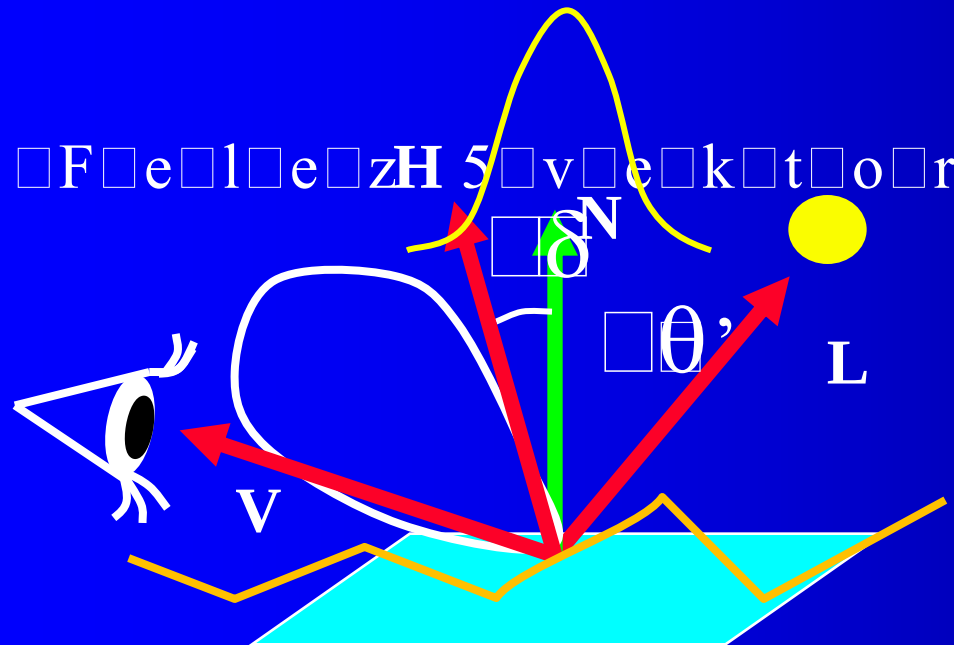
Egyszeres anyag

kölcsön

nefém

hullámh

Phong-Blinn modell (OpenGL)



$$\mathbf{H} = (\mathbf{L} + \mathbf{V}) / |\mathbf{L} + \mathbf{V}|$$

$$\cos \delta = \mathbf{N} \cdot \mathbf{H}$$

$$L^{ref} = L^{in} k_s (\cos^+ \delta)^n$$

$$f_r(\mathbf{L}, \mathbf{x}, \mathbf{V}) = k_s \cos^n \delta \cos \theta / \theta$$

Röcsökös f

class RoughSurface {

Color kd, ks; // r,g,b

float shininess;

public:

Color ReflectedRadiance(Vector& L, Vector& N, Vector& V, Color Lin) {

float costheta = N * L;

if (costheta < 0) return Color(0, 0, 0);

Color Lref= Lin * kd * costheta; // diffuse reflection

Vector H = L + V;

H.Normalize ();

float cosdelta = N * H;

if (cosdelta < 0) return Lref;

Lref = Lref +

Lin * ks * pow(cosdelta, shininess); // glossy reflection

return Lref;

}

};

Fényterjedési modellek

● Geometriai + sugár

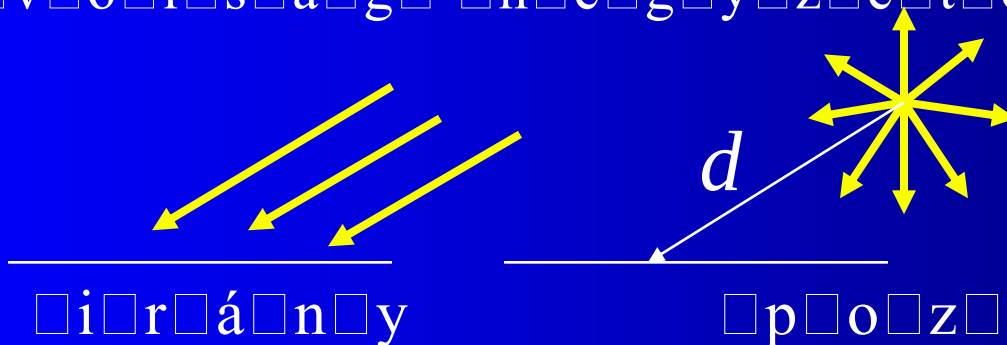
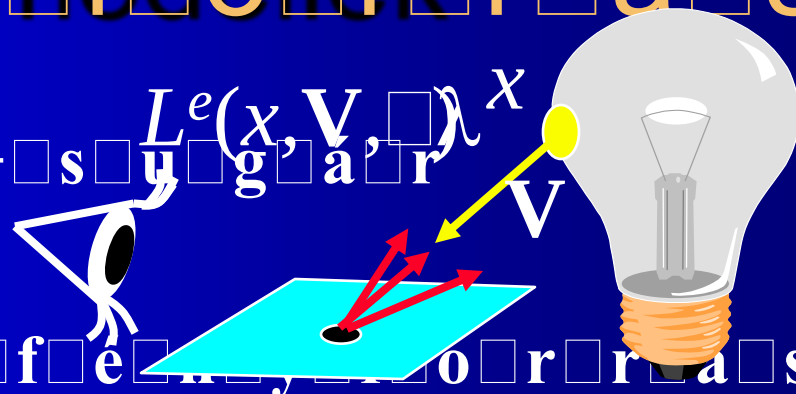
● Abstrakt fényterjedési

– Irányi fényterjedési

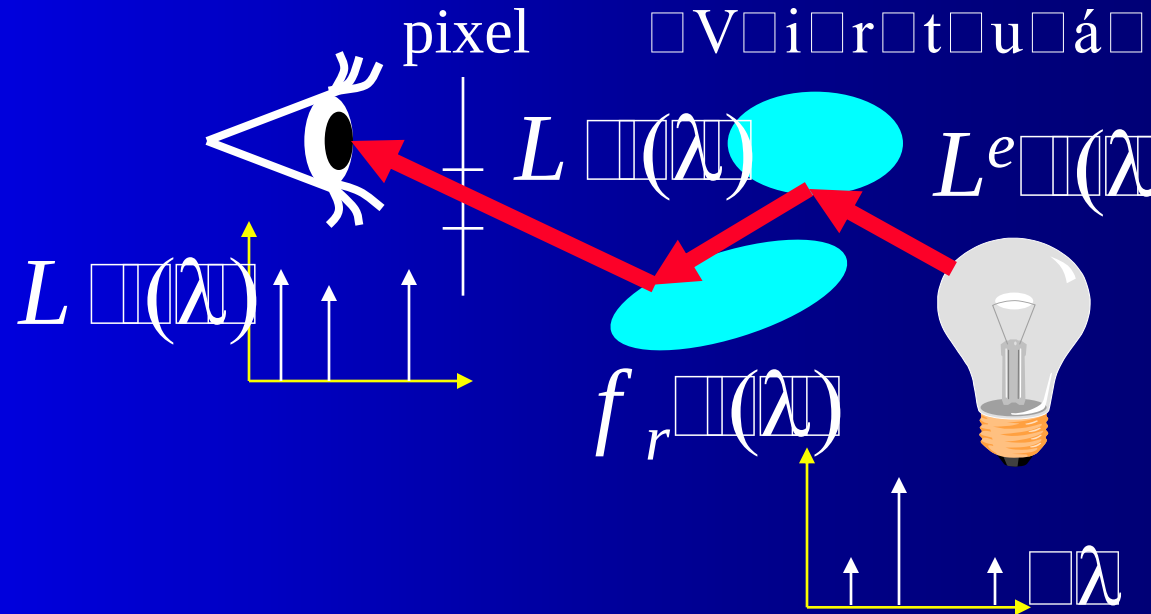
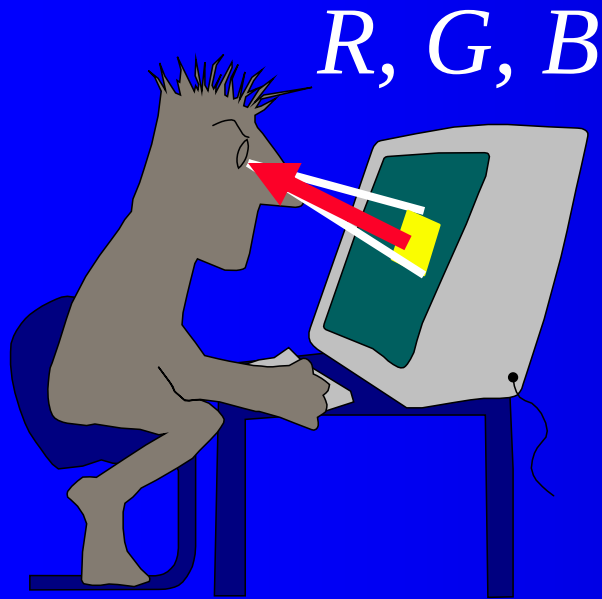
párhuzamosak, azaz irányi

– Positionális fényterjedési

távolság négyzetével csö

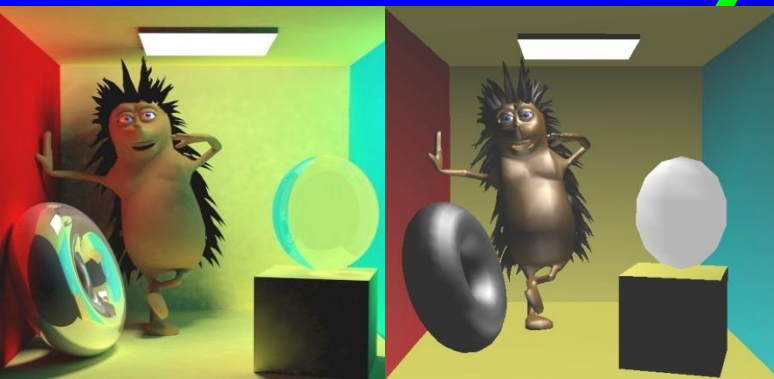
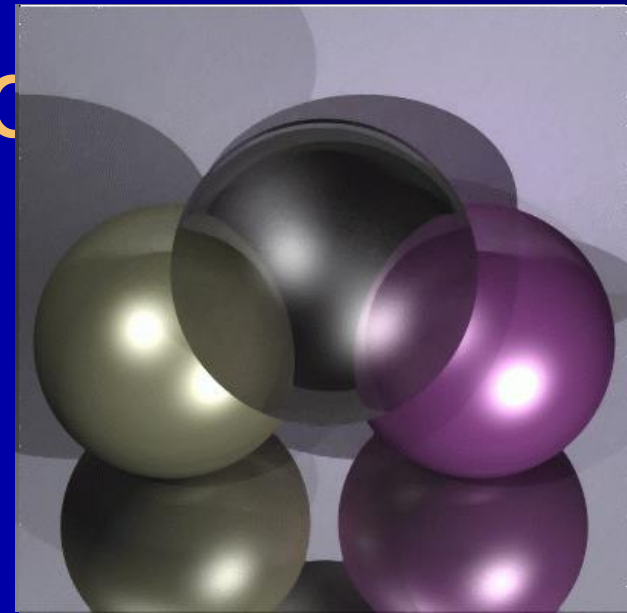
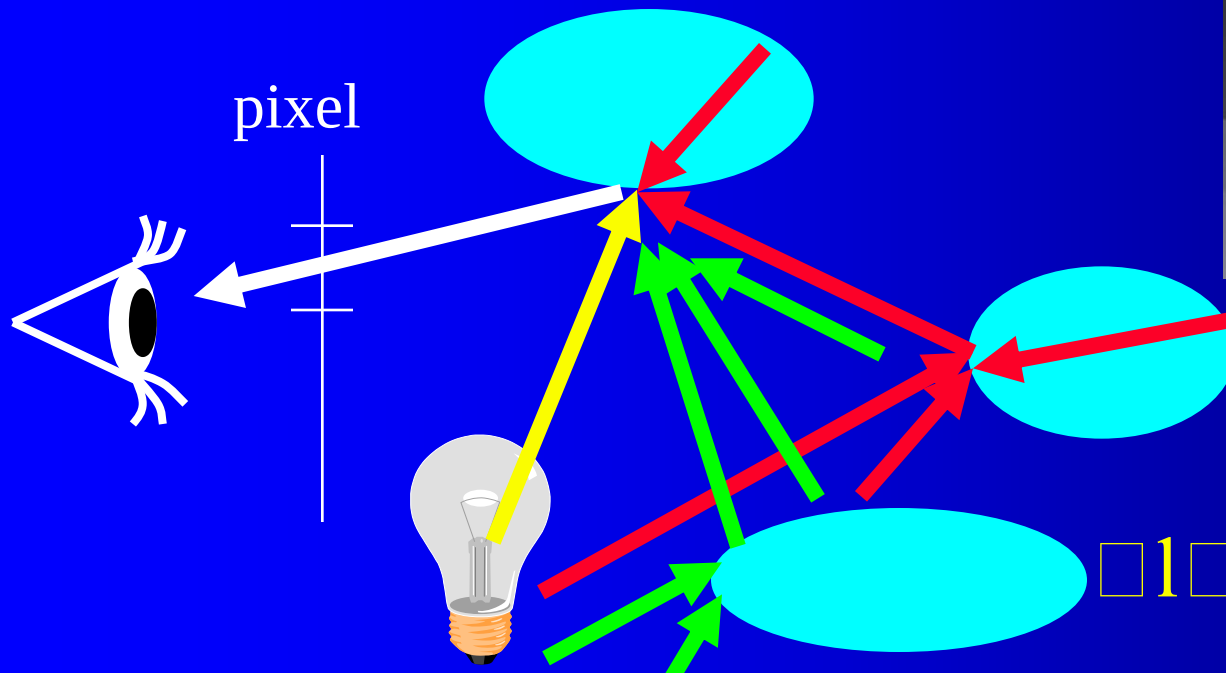


Képszíntézis



- Pixelben látó f
- A látó pont sze
- R, G, B konverzión

Megoldási kompromisszumok

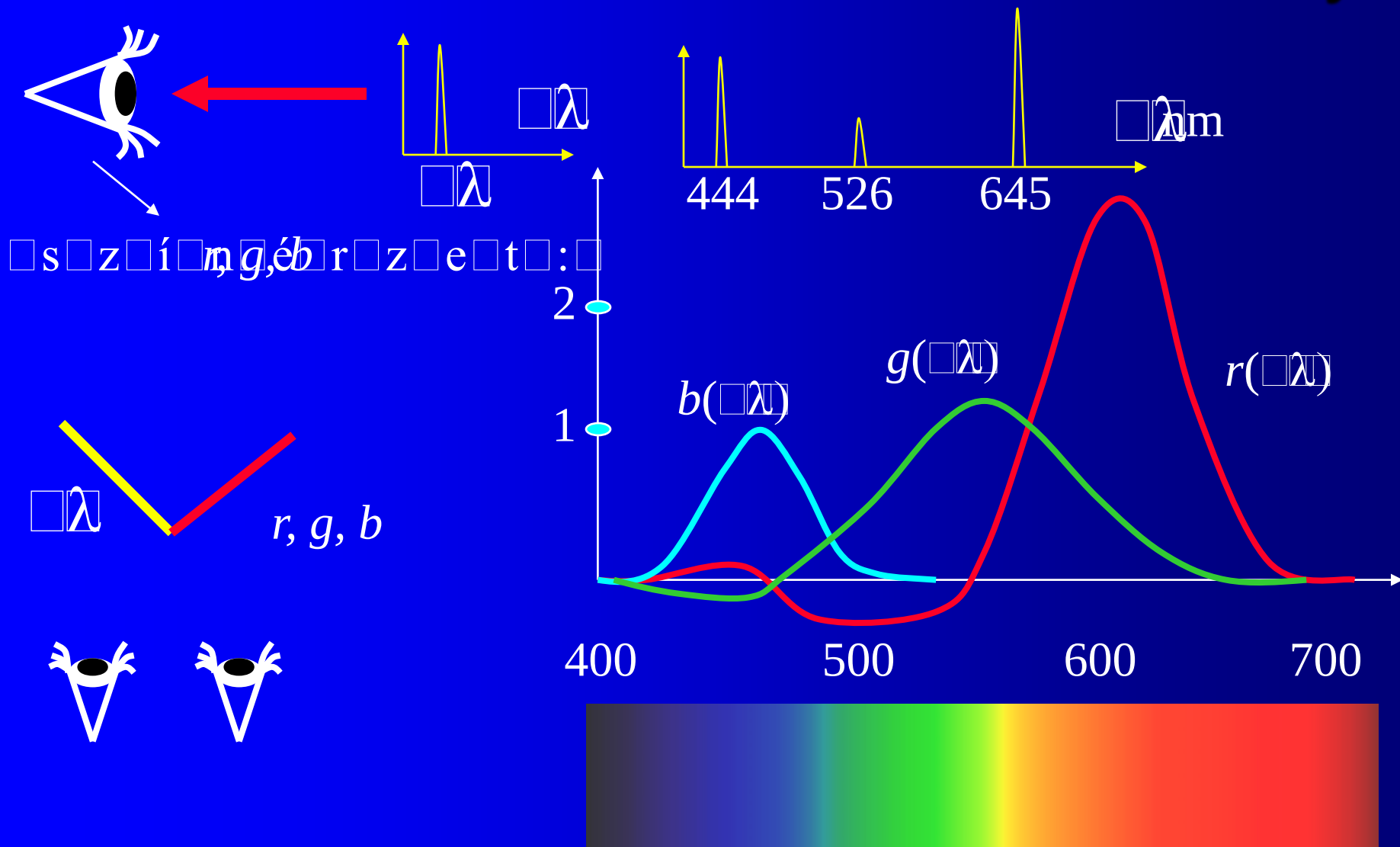


l o k á l i s

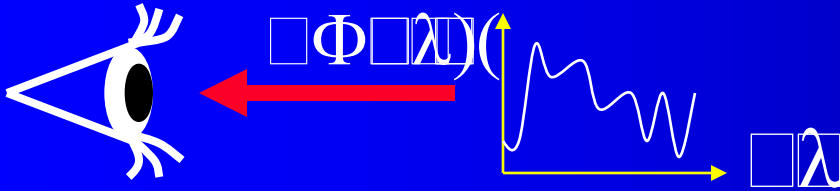
r e k u r z í

g l o b á l i

Sínérzékelés: monokromatikus fény:



Színérzékelés nem monokromatikus fény

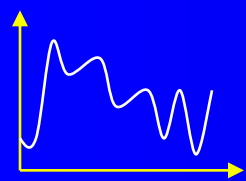


$$r = \int \Phi(\lambda) \bar{r}(\lambda) d\lambda$$

$$g = \int \Phi(\lambda) \bar{g}(\lambda) d\lambda$$

$$b = \int \Phi(\lambda) \bar{b}(\lambda) d\lambda$$

Spektrális vs RGB



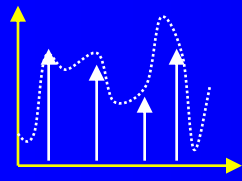
λ

model

R, G, B →

model

Képszíntézis
sok hullámhossz



λ

Látható
spektrum

R, G, B

színelésztés

R, G, B kép

R, G, B

Tone mapping

HRDI(R,G,B) $\rightarrow$ LDRI(r,g,b) $\rightarrow$ k $\rightarrow$ o $\rightarrow$ n $\rightarrow$ v $\rightarrow$ c

1. Luminancia:

$$I = 0.21R + 0.72G + 0.07B$$

2. A $\rightarrow$ d $\rightarrow$ a $\rightarrow$ l $\rightarrow$ t $\rightarrow$ I $\rightarrow$ $'$

3. Relat $\rightarrow$ l $\rightarrow$ u $\rightarrow$ m $\rightarrow$ i $\rightarrow$ n $\rightarrow$ a $\rightarrow$ n $\rightarrow$ c $\rightarrow$ i $\rightarrow$ a $\rightarrow$ l $\rightarrow$ t $\rightarrow$ I $\rightarrow$ $'$

4. M $\rightarrow$ e $\rightarrow$ g $\rightarrow$ j $\rightarrow$ l $\rightarrow$ u $\rightarrow$ m $\rightarrow$ i $\rightarrow$ n $\rightarrow$ a $\rightarrow$ n $\rightarrow$ c $\rightarrow$ i $\rightarrow$ a $\rightarrow$ l $\rightarrow$ t $\rightarrow$ I $\rightarrow$ $'$

$$D = \frac{I_r}{1 + I_r}$$

5. S $\rightarrow$ z $\rightarrow$ i $\rightarrow$ n $\rightarrow$ $:$

$$(r,g,b) = (R,G,B) \frac{D}{I}$$

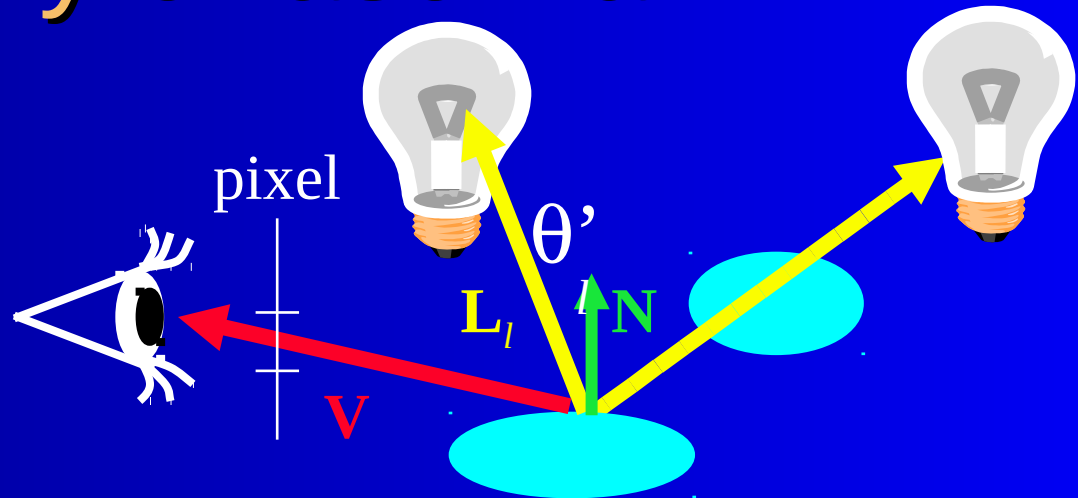


Sugárkövetés: ray-casting, ray-tracing

Szirmay-Kalos László

Lokális illumináció absztrakt fényforrásokkal

Csak absztrakt
fényforrások
direkt megvilágítása

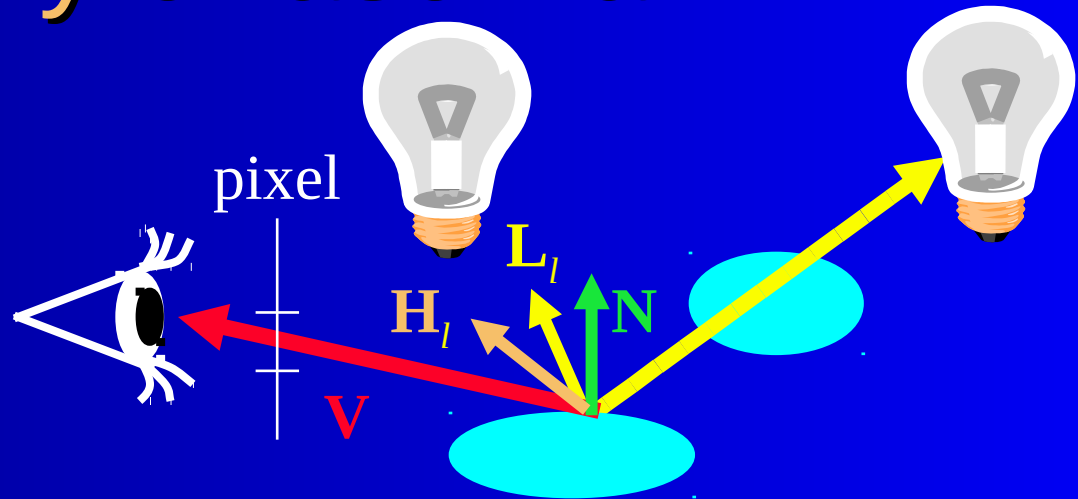


$$L(\mathbf{V}) \approx \sum_l L_l(\mathbf{L}_l) * f_r(\mathbf{L}_l, \mathbf{N}, \mathbf{V}) \cdot \cos \theta'_l$$

Absztrakt fényforrásokból származó megvilágítás.
(Irányforrás = konstans; Pontforrás = távolság négyzetével csökken
Ha takart, akkor zérus)

Lokális illumináció absztrakt fényforrásokkal

Csak absztrakt
fényforrások
direkt megvilágítása

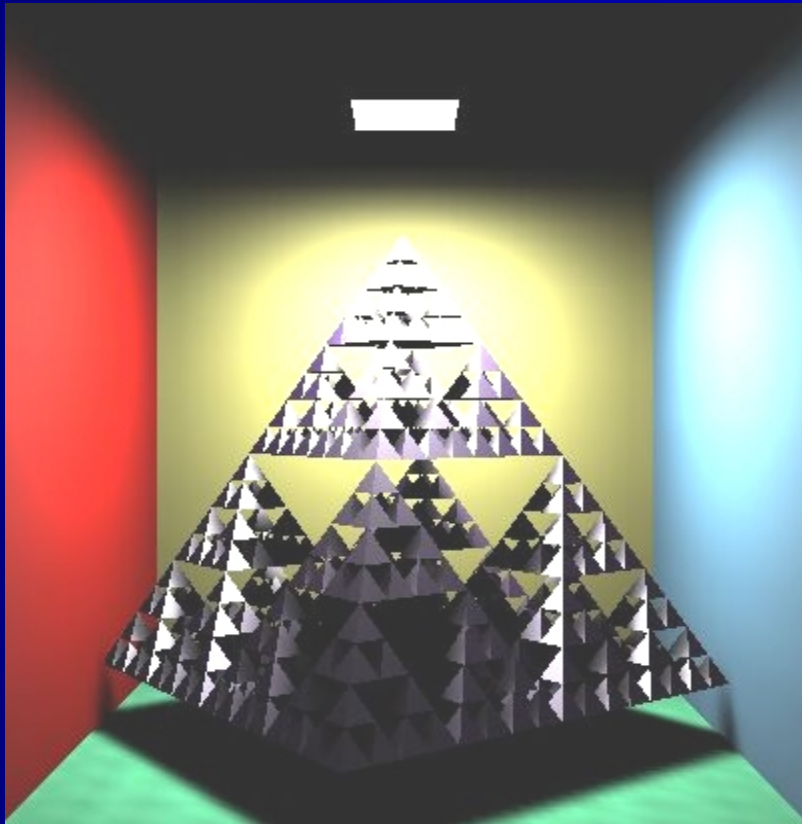


$$L(\mathbf{V}) \approx \sum_l L_l(\mathbf{L}_l) * \{k_d \cdot (\mathbf{L}_l \cdot \mathbf{N})^+ + k_s \cdot ((\mathbf{H}_l \cdot \mathbf{N})^+)^{shine}\}$$

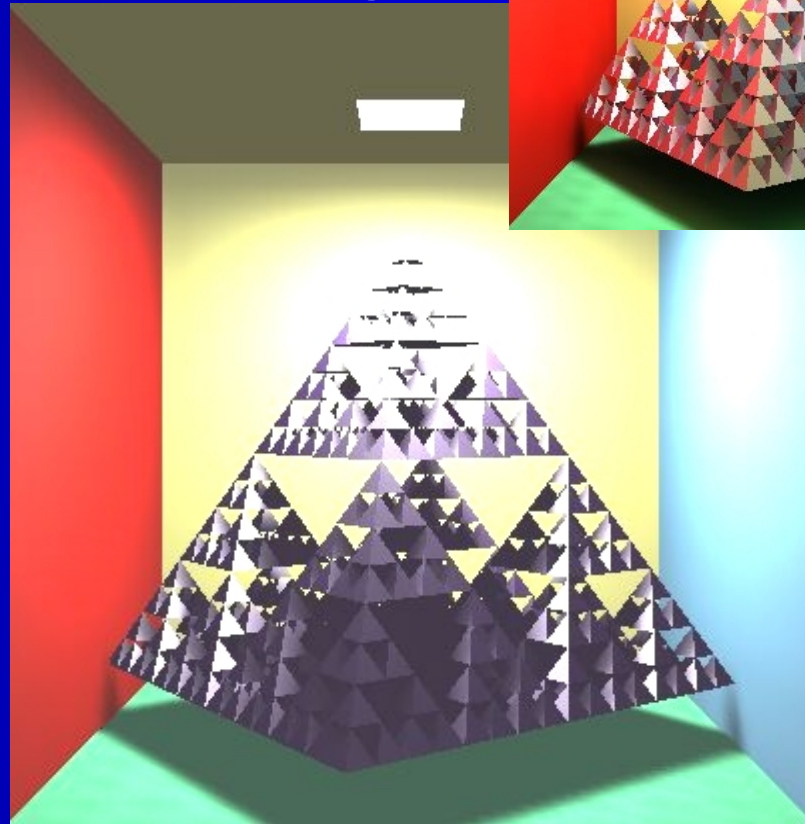
Absztrakt fényforrásokból származó megvilágítás.
(Irányforrás = konstans; Pontforrás = távolság négyzetével csökken
Ha takart, akkor zérus)

Ambiens tag

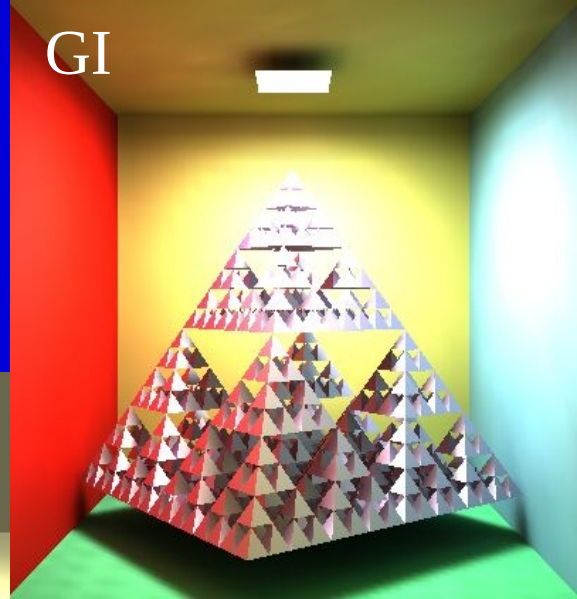
Lokális illumináció



+ ambiens tag

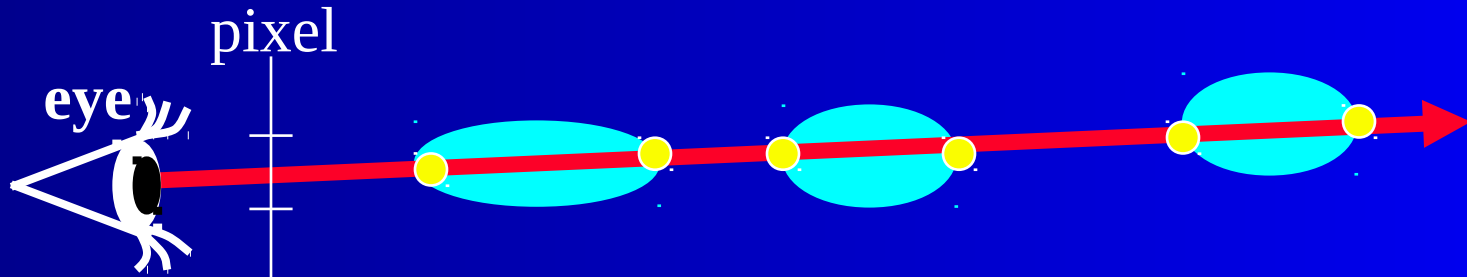


GI



$$L(\mathbf{V}) \approx \sum_l L_l(\mathbf{L}_l) * f_r(\mathbf{L}_l, \mathbf{N}, \mathbf{V}) \cdot \cos \theta'_l \left[+ k_a * L_a \right]$$

Láthatóság a szemből



$$\text{ray}(t) = \text{eye} + \mathbf{v} \cdot t, \quad t > 0$$

intersectAll(ray $\Rightarrow$ t, iobject, x, inormal)

t = FLT_MAX;

FOR each object

tnew = **Intersect(ray, object $\Rightarrow$ normal)**; // < 0 ha nincs metszés

IF (tnew > 0 && tnew < t)

{ t = tnew; iobject = object; inormal = normal; }

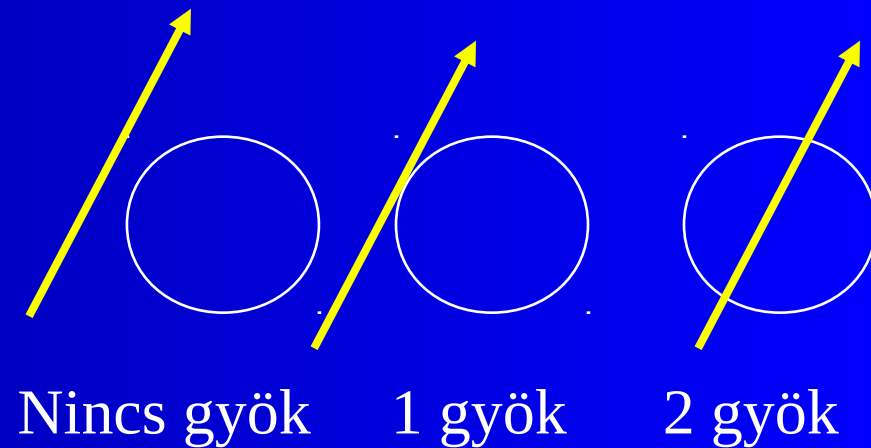
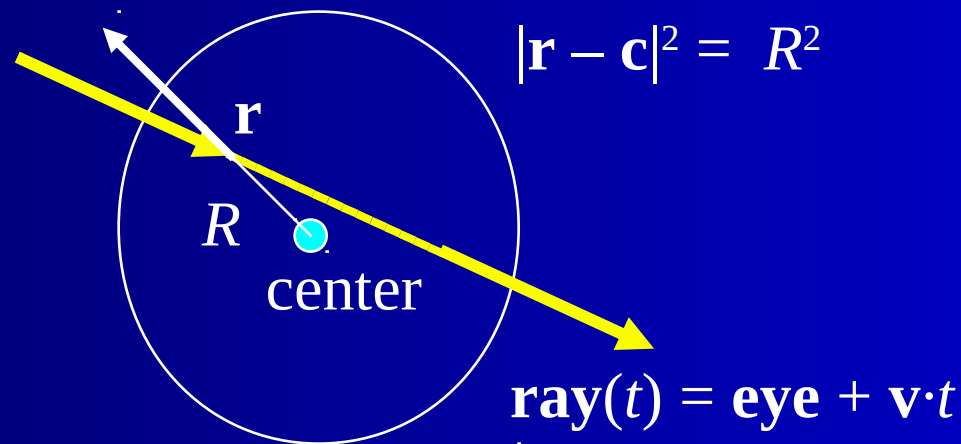
ENDFOR

IF (t < FLT_MAX) { x = eye + $\mathbf{v} \cdot t$; RETURN (t, iobject, x, inormal); }

RETURN „no intersection”

END

Metszéspont számítás gömbbel



$$|\text{ray}(t) - \mathbf{c}|^2 = (\text{ray}(t) - \mathbf{c}) \bullet (\text{ray}(t) - \mathbf{c}) = R^2$$

$$(\mathbf{v} \bullet \mathbf{v})t^2 + 2((\text{eye} - \mathbf{c}) \bullet \mathbf{v})t + ((\text{eye} - \mathbf{c}) \bullet (\text{eye} - \mathbf{c})) - R^2 = 0$$

Wanted: a pozitív megoldások közül a kisebb

Felületi normális: $(\text{ray}(t) - \mathbf{c})/R$

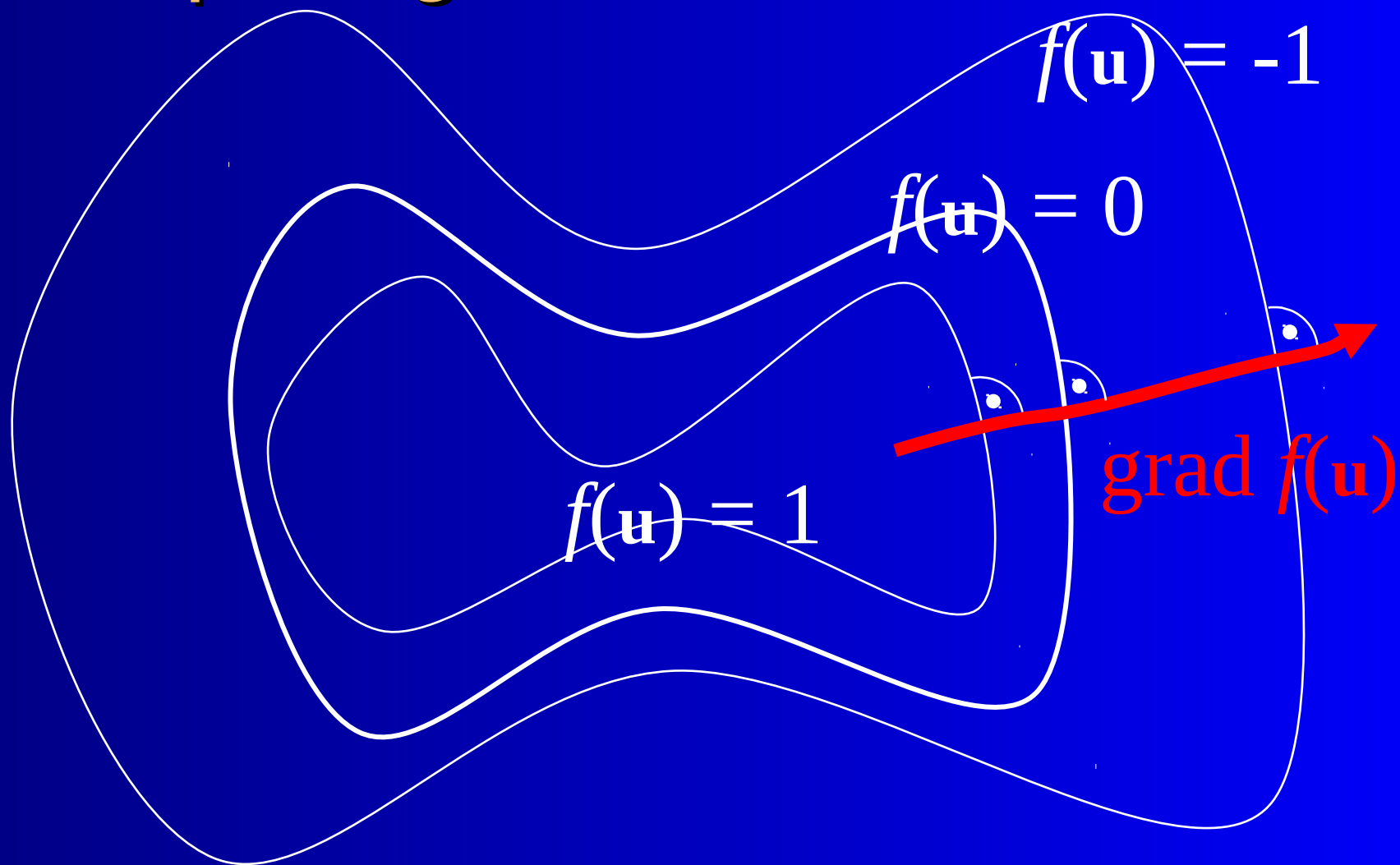
Implicit felületek

- A felület pontjai: $f(x,y,z) = 0$ vagy $f(\mathbf{r}) = 0$
- A sugár pontjai: $\mathbf{ray}(t) = \mathbf{eye} + \mathbf{v} \cdot t$
- A metszéspont: $f(\mathbf{ray}(t)) = 0$,
 - 1 ismeretlenes, ált. nemlineáris egyenlet: t^*
 - $(x^*, y^*, z^*) = \mathbf{eye} + \mathbf{v} \cdot t^*$
- Normálvektor = $\text{grad } f \Big|_{x^*, y^*, z^*}$
 - $0 = f(x,y,z) = f(x^* + (x-x^*), y^* + (y-y^*), z^* + (z-z^*)) \approx$
 $f(x^*, y^*, z^*) + \frac{\partial f}{\partial x}(x-x^*) + \frac{\partial f}{\partial y}(y-y^*) + \frac{\partial f}{\partial z}(z-z^*)$

Az érintősík
egyenlete:

$$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \bullet (x-x^*, y-y^*, z-z^*) = 0$$

2D analógia: Implicit görbe normálvektora

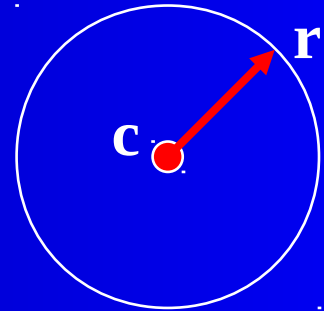


Konkrét példa: gömb normálvektora

$$|\mathbf{r} - \mathbf{c}|^2 = R^2$$

$$|\mathbf{r} - \mathbf{c}|^2 - R^2 = 0$$

$$f(\mathbf{r}) = 0$$



$$f(x,y,z) = (x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2 - R^2 = 0$$

$$\frac{\partial f}{\partial x} = 2(x - c_x) + 0 + 0 - 0$$

$$\frac{\partial f}{\partial x} = 2(x - c_x) \quad \frac{\partial f}{\partial y} = 2(y - c_y) \quad \frac{\partial f}{\partial z} = 2(z - c_z)$$

$$\text{grad } f(x,y,z) = 2 (x-c_x, y-c_y, z-c_z)$$

Kvadratikus felületek

Kvadratikus felület: $[x,y,z,1] A \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0 \quad \Rightarrow \quad \text{Másodfokú egyenlet}$



Ellipszoid

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1 = 0$$

Végtelen kúp

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - z^2 = 0$$

Végtelen henger

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$$

Kvadratikus felület

$$[\mathbf{eye} + \mathbf{v}t, 1] = \mathbf{o} + t\mathbf{d} \longrightarrow \mathbf{o} = [\mathbf{eye}, 1], \mathbf{d} = [\mathbf{v}, 0]$$

$$(\mathbf{o} + t\mathbf{d})\mathbf{A}(\mathbf{o} + t\mathbf{d})^T = 0$$

$$(\mathbf{o}\mathbf{A} + t\mathbf{d}\mathbf{A})(\mathbf{o} + t\mathbf{d})^T = 0$$

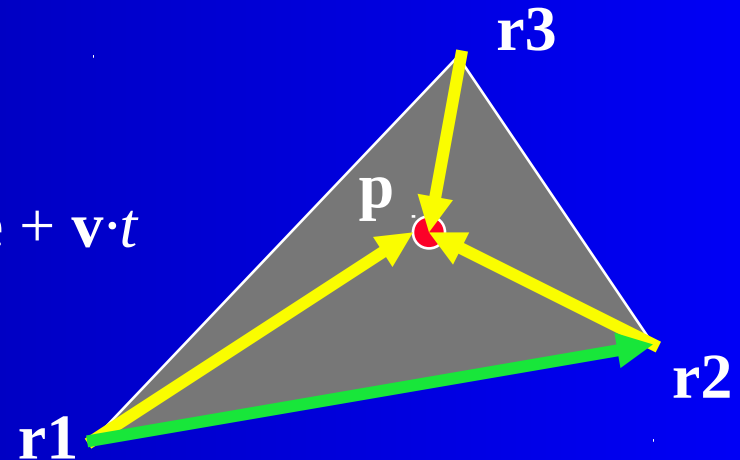
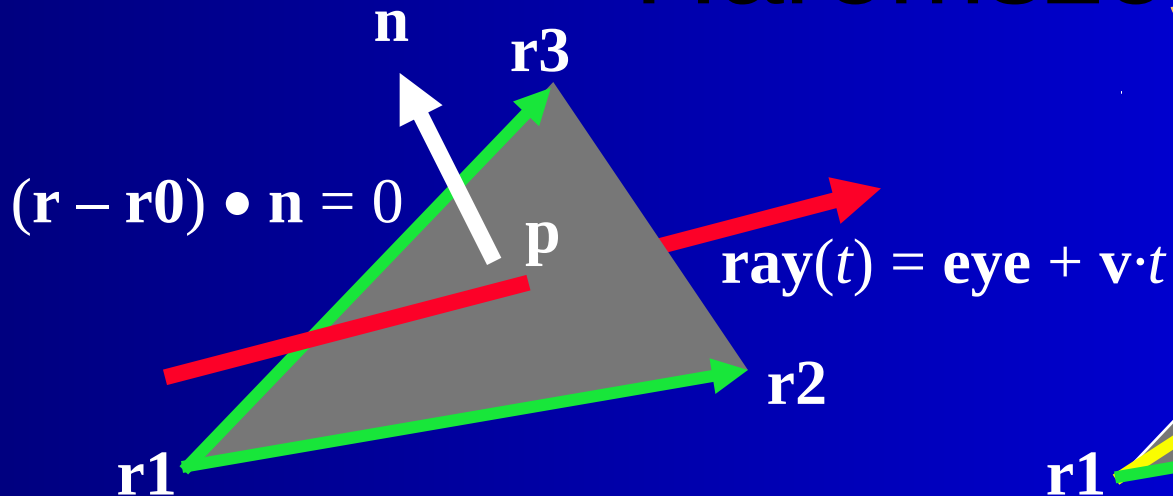
$$\mathbf{o}\mathbf{A}\mathbf{o}^T + \mathbf{o}\mathbf{A}t\mathbf{d}^T + t\mathbf{d}\mathbf{A}\mathbf{o}^T + t\mathbf{d}\mathbf{A}t\mathbf{d}^T = 0$$

$$\mathbf{d}\mathbf{A}\mathbf{d}^T \cdot t^2 + (\mathbf{o}\mathbf{A}\mathbf{d}^T + \mathbf{d}\mathbf{A}\mathbf{o}^T) \cdot t + \mathbf{o}\mathbf{A}\mathbf{o}^T = 0$$

másodfokú egyenlet t -re

$$\frac{\partial}{\partial \mathbf{x}} [x, y, z, 1] \mathbf{A} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = [1, 0, 0, 0] \mathbf{A} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \neq [x, y, z, 1] \mathbf{A} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Háromszög



1. Síkmetszés:
normál:

$$(ray(t) - r1) \cdot n = 0, \quad t > 0$$

$$n = (r2 - r1) \times (r3 - r1)$$

$$t = \frac{(r1 - eye) \cdot n}{v \cdot n}$$

2. A metszéspont a háromszögön belül van-e?

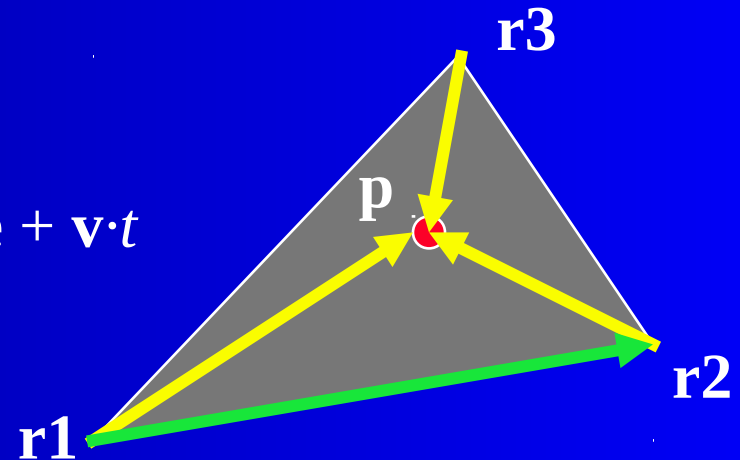
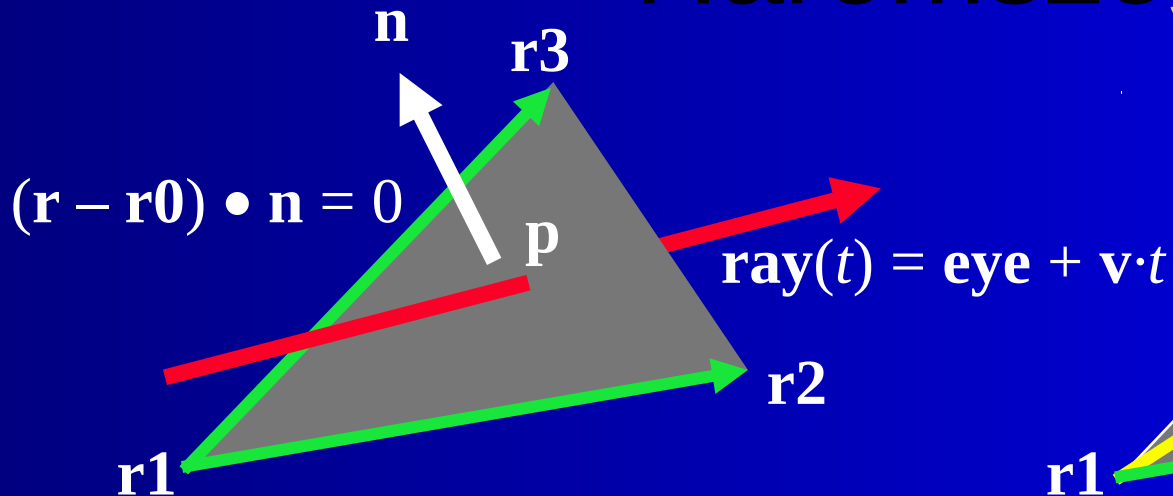
$$((r2 - r1) \times (p - r1)) \cdot n > 0$$

$$((r3 - r2) \times (p - r2)) \cdot n > 0$$

$$((r1 - r3) \times (p - r3)) \cdot n > 0$$

Felületi normális: n
vagy árnyaló normálok
(shading normals)

Háromszög



1. Síkmetszés:
normál:

$$(\text{ray}(t) - \mathbf{r1}) \cdot \mathbf{n} = 0, \quad t > 0$$

$$\mathbf{n} = (\mathbf{r2} - \mathbf{r1}) \times (\mathbf{r3} - \mathbf{r1})$$

$$t = \frac{(\mathbf{r1} - \text{eye}) \cdot \mathbf{n}}{\mathbf{v} \cdot \mathbf{n}}$$

2. A metszéspont a háromszögön belül van-e?

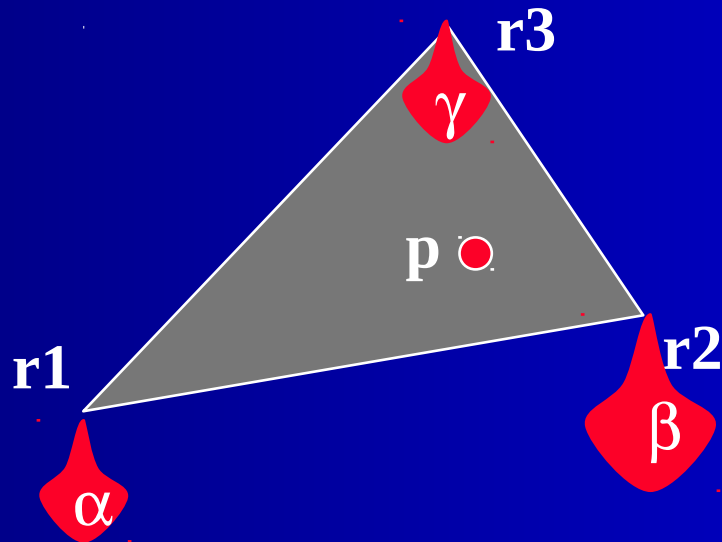
$$((\mathbf{r2} - \mathbf{r1}) \times (\mathbf{p} - \mathbf{r1})) \cdot \mathbf{n} > 0$$

$$((\mathbf{r3} - \mathbf{r2}) \times (\mathbf{p} - \mathbf{r2})) \cdot \mathbf{n} > 0$$

$$((\mathbf{r1} - \mathbf{r3}) \times (\mathbf{p} - \mathbf{r3})) \cdot \mathbf{n} > 0$$

Felületi normális: $\mathbf{n}$
vagy árnyaló normálok
(shading normals)

Háromszög metszés baricentrikus koordinátákban



$$p(\alpha, \beta, \gamma) = \alpha \cdot r_1 + \beta \cdot r_2 + \gamma \cdot r_3$$

$$\alpha + \beta + \gamma = 1 \quad 1 \text{ skalár egyenlet}$$

$$\alpha, \beta, \gamma \geq 0$$

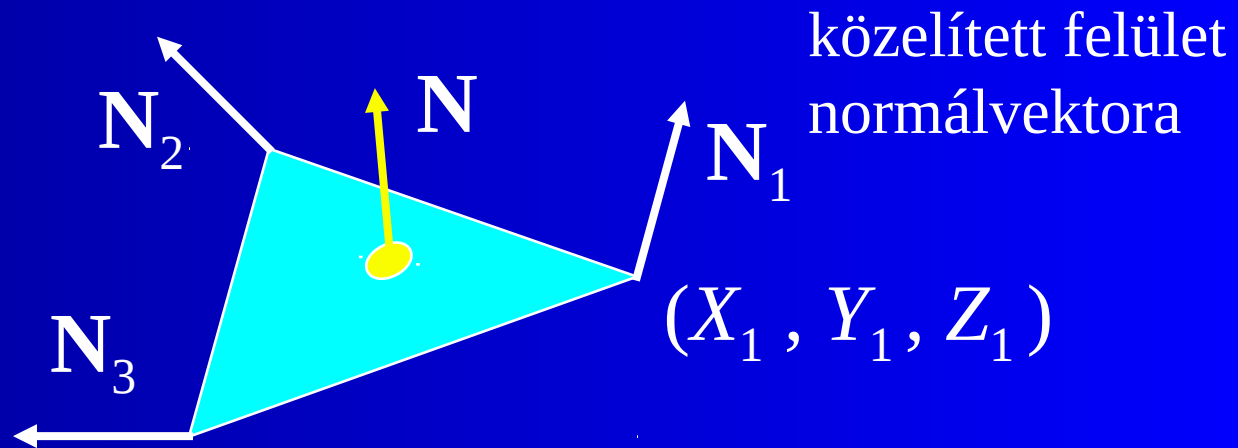
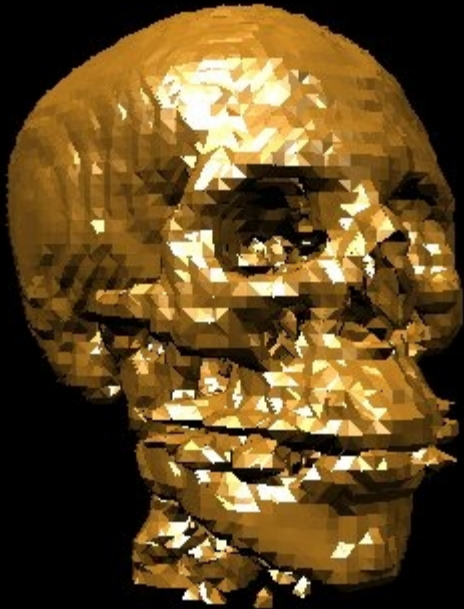
$$\text{ray}(t) = p(\alpha, \beta, \gamma)$$

$$\text{eye} + v \cdot t = \alpha \cdot r_1 + \beta \cdot r_2 + \gamma \cdot r_3 \quad 3 \text{ skalár egyenlet}$$

4 ismeretlen: α, β, γ, t

A megoldás után ellenőrzés, hogy mind nem negatív-e

Árnyaló normálok



$$\mathbf{N}(\alpha, \beta, \gamma) = \alpha \mathbf{N}_1 + \beta \mathbf{N}_2 + \gamma \mathbf{N}_3$$



Parametrikus felületek

$\mathbf{r}(u,v)$, $u,v \in [0,1]$

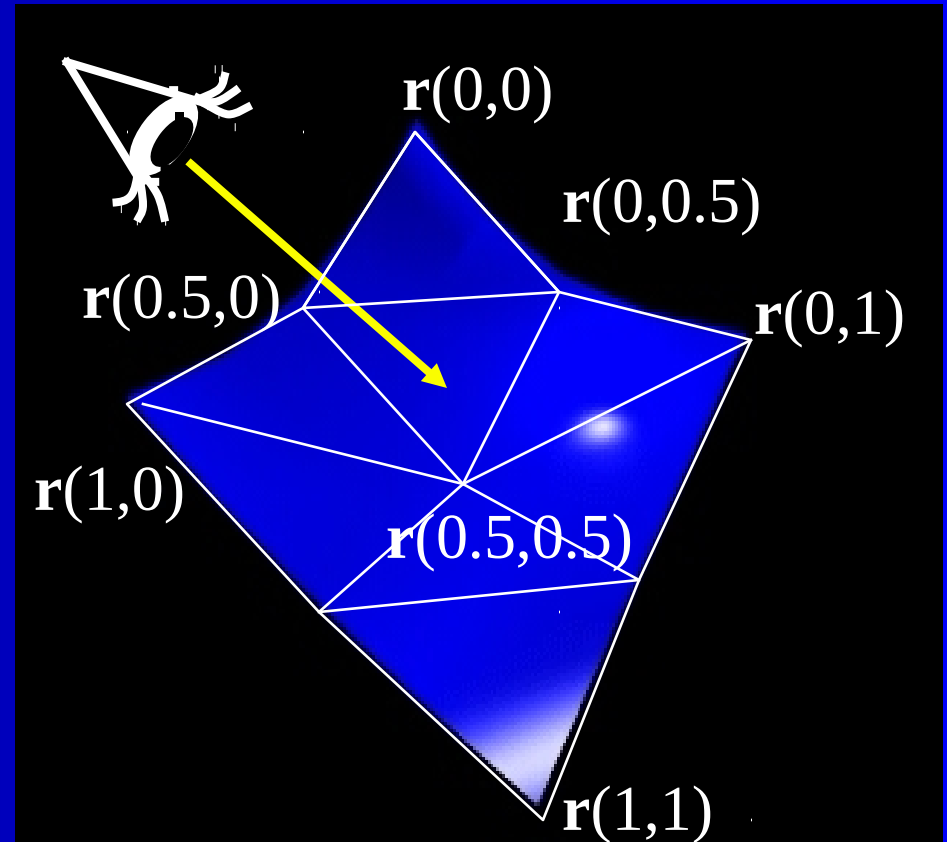
$\mathbf{ray}(t) = \mathbf{eye} + \mathbf{v} \cdot t$, $t > 0$

$\mathbf{r}(u,v) = \mathbf{ray}(t)$

Háromismeretlenes ált.
nem lineáris egyenletrendszer
megoldás: u^*, v^*, t^*

Teszt:

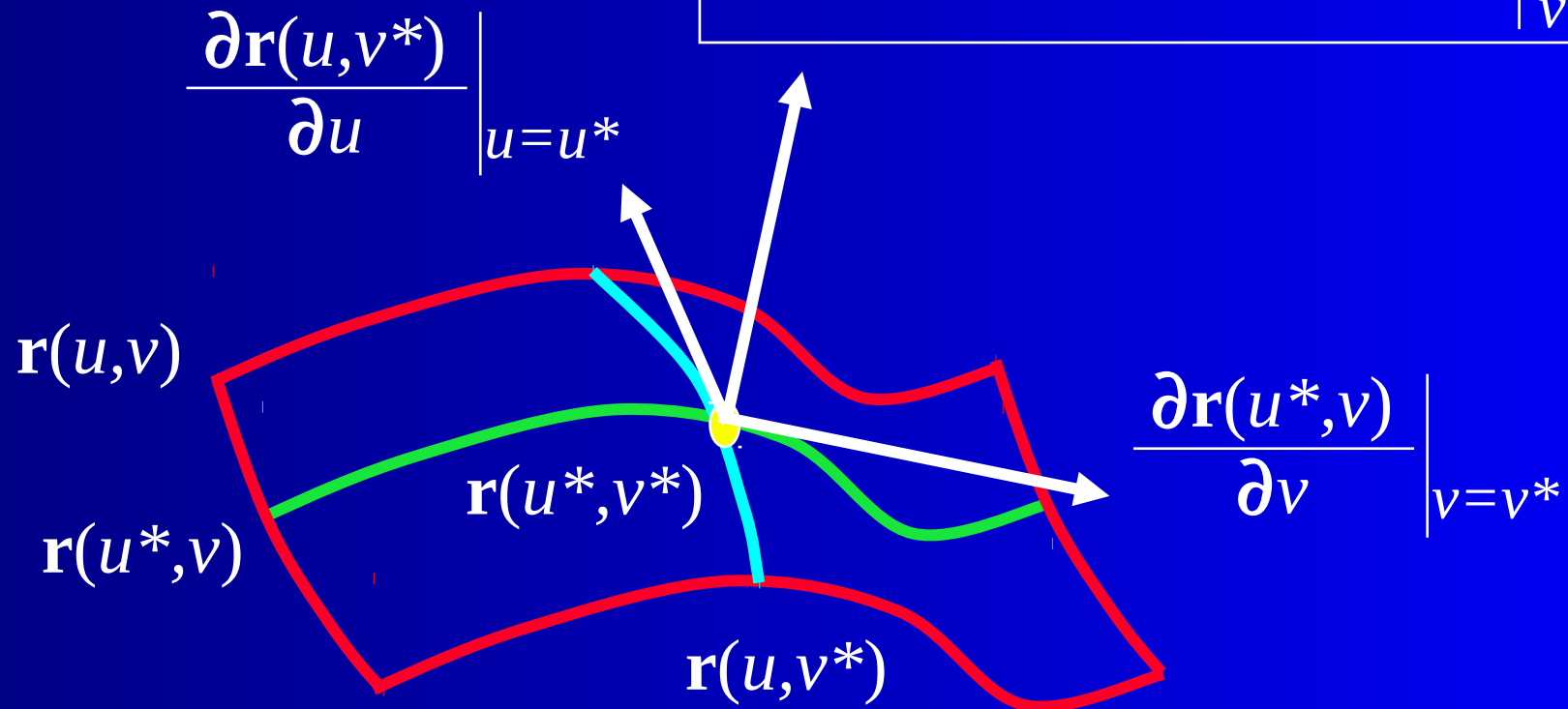
$0 < u^*, v^* < 1$,
 $t^* > 0$



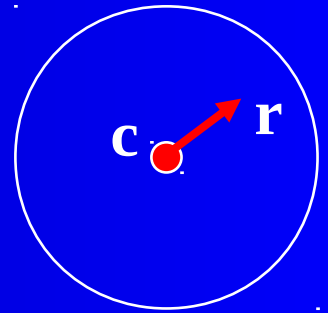
Rekurzív tesszelláció:
nem robusztus

Parametrikus felületek normálvektora

$$\mathbf{n}(u,v) = \left. \frac{\partial \mathbf{r}(u,v)}{\partial u} \times \frac{\partial \mathbf{r}(u,v)}{\partial v} \right|_{\substack{u=u^* \\ v=v^*}}$$



Gömb, az öreg haver



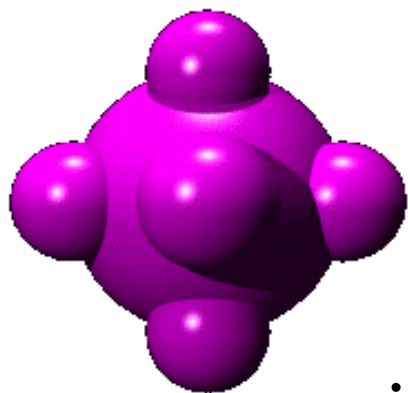
$$\mathbf{r}(u,v) = [c_x + R\cos(u)\sin(v), c_y + R\sin(u)\sin(v), c_z + R\cos(v)]$$

$$\frac{\partial \mathbf{r}}{\partial u} = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -R\sin(u)\sin(v) & R\cos(u)\sin(v) & 0 \\ R\cos(u)\cos(v) & R\sin(u)\cos(v) & -R\sin(v) \end{bmatrix}$$

$$\mathbf{n}(u,v) = \frac{\partial \mathbf{r}}{\partial u} \times \frac{\partial \mathbf{r}}{\partial v} =$$

$$\begin{aligned} & [-R^2\cos(u)\sin^2(v), -R^2\sin(u)\sin^2(v), -R^2(\sin^2(u)+\cos^2(u))\sin(v)\cos(v)] = \\ & = -R\sin(v) [R\cos(u)\sin(v), R\sin(u)\sin(v), R\cos(v)] \\ & = -R\sin(v) [x-c_x, y-c_y, z-c_z]. \end{aligned}$$

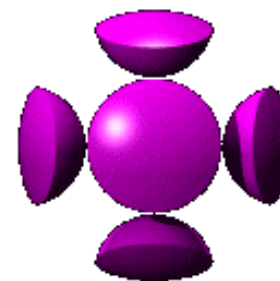
Constructive Solid Geometry



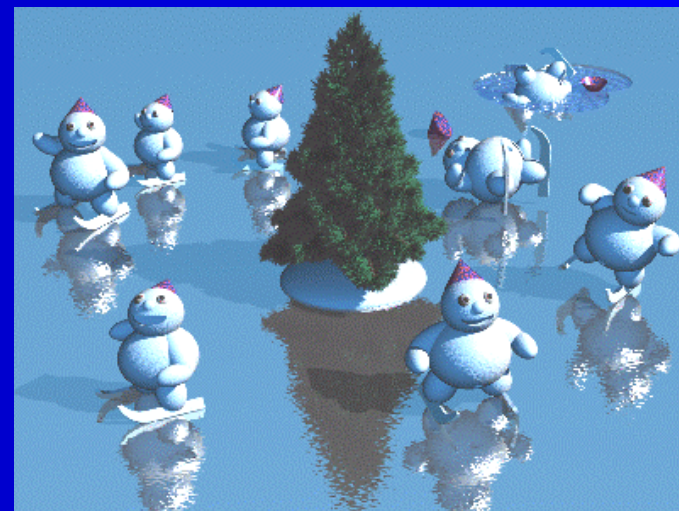
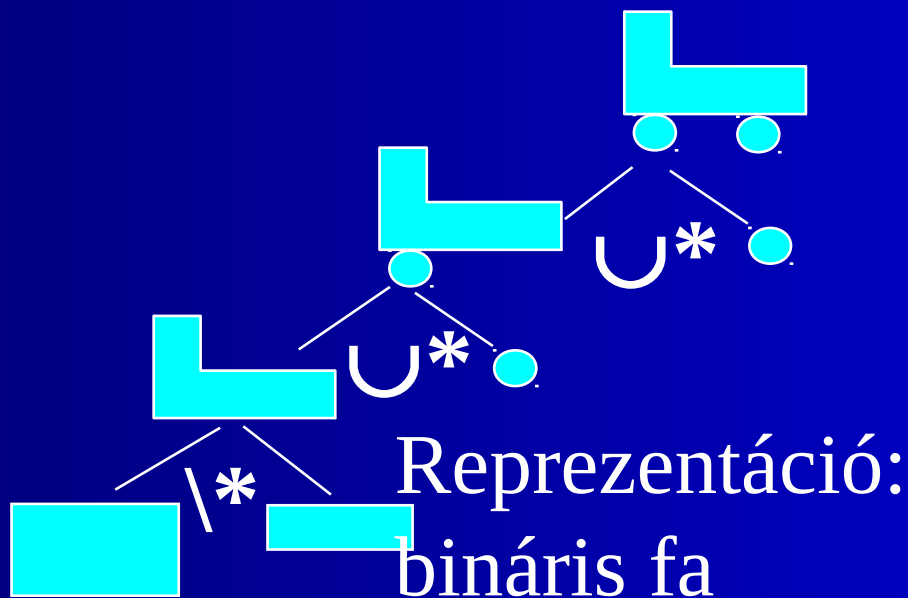
unió



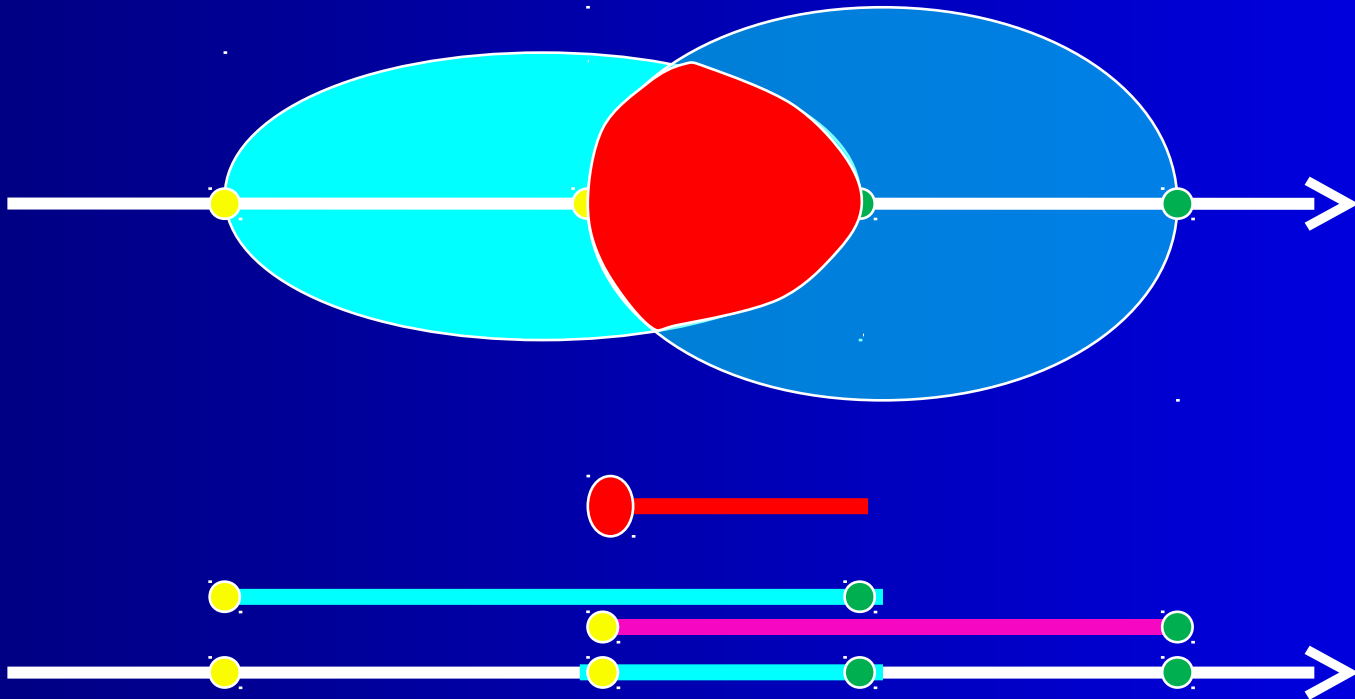
különbség



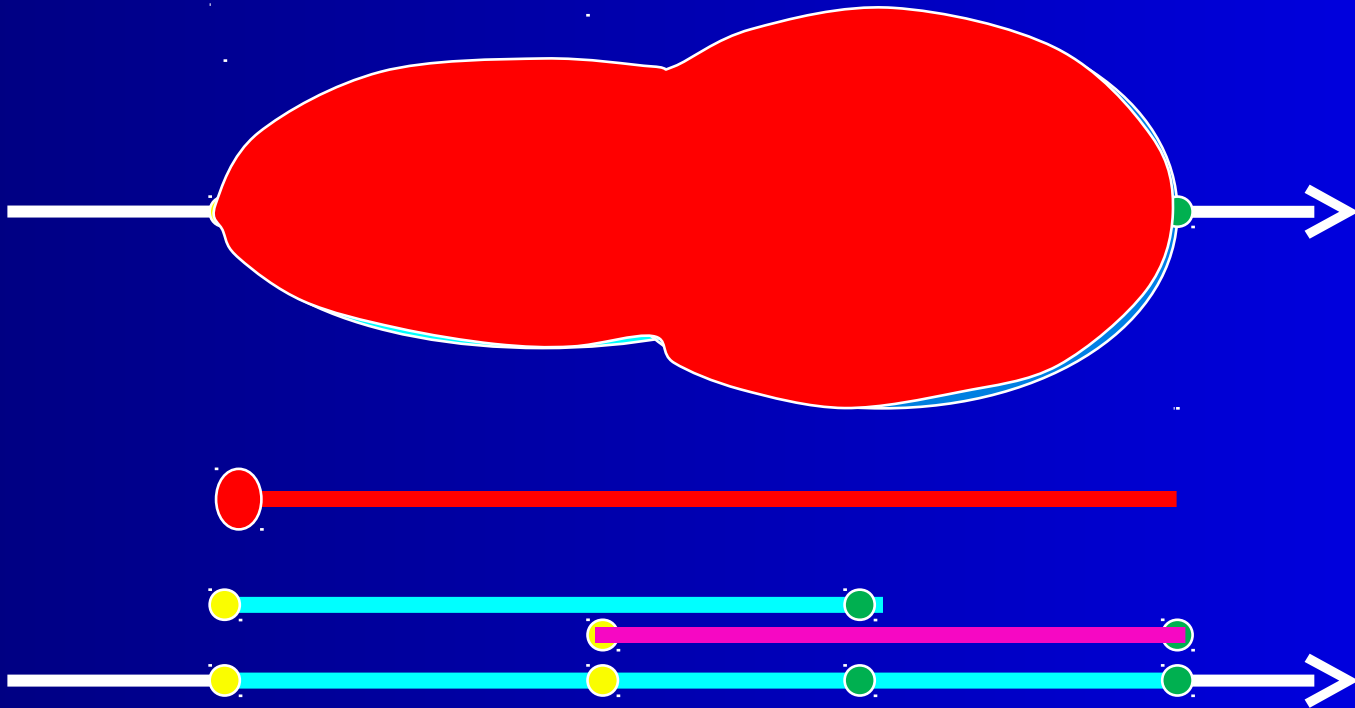
interszekció



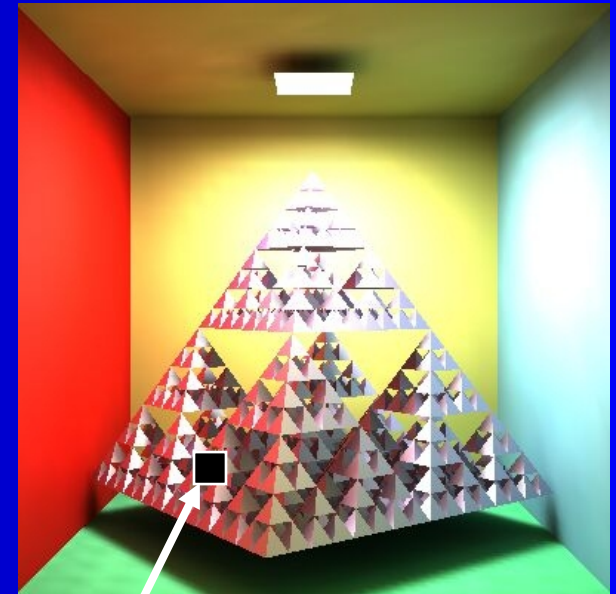
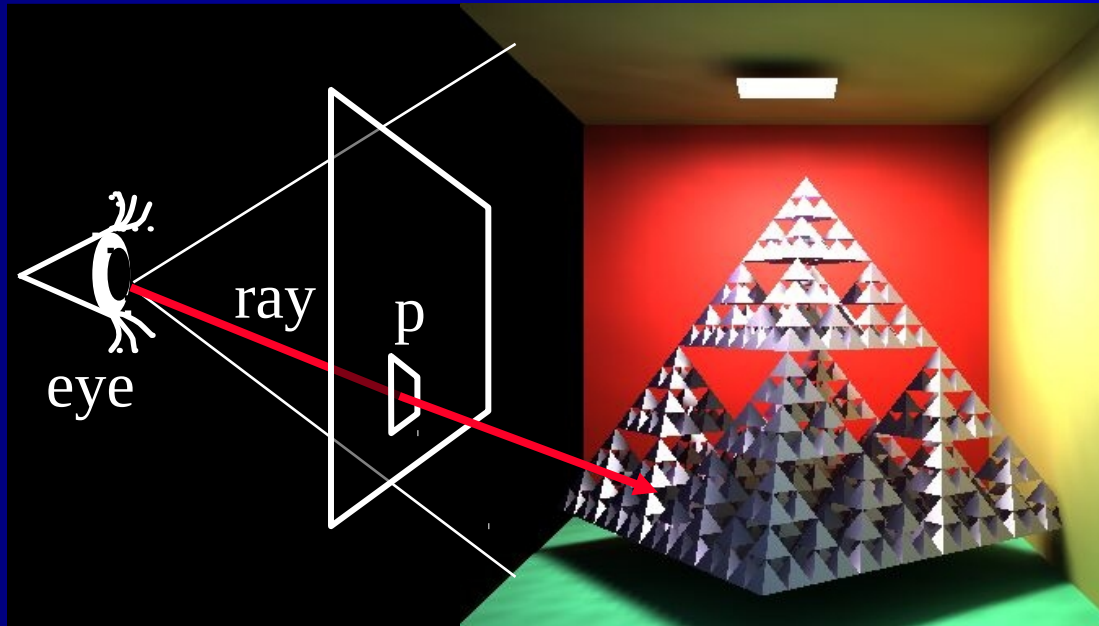
CSG sugárkövetés: Metszet



CSG sugárkövetés: Unió



Sugárkövetés



render()

for each pixel p

Ray r = camera.GetRay (eye $\Rightarrow$ pixel p)

float3 color = **trace**(ray)

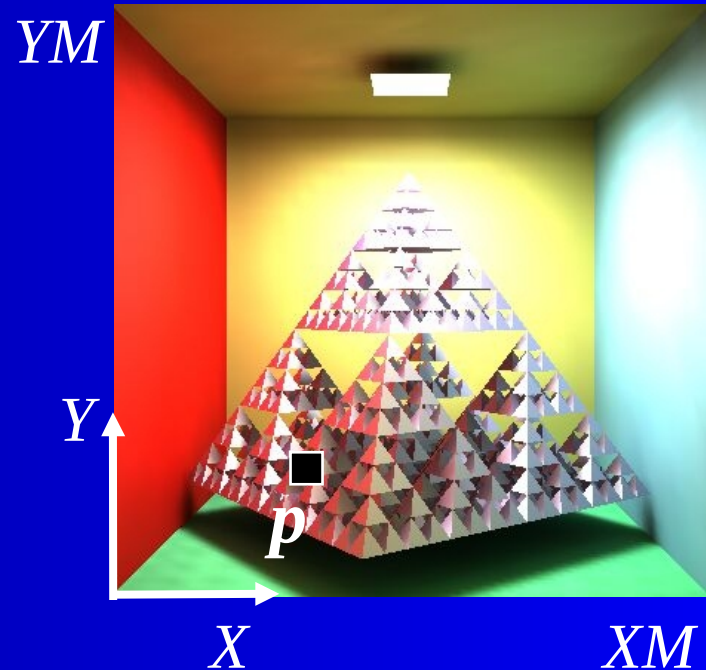
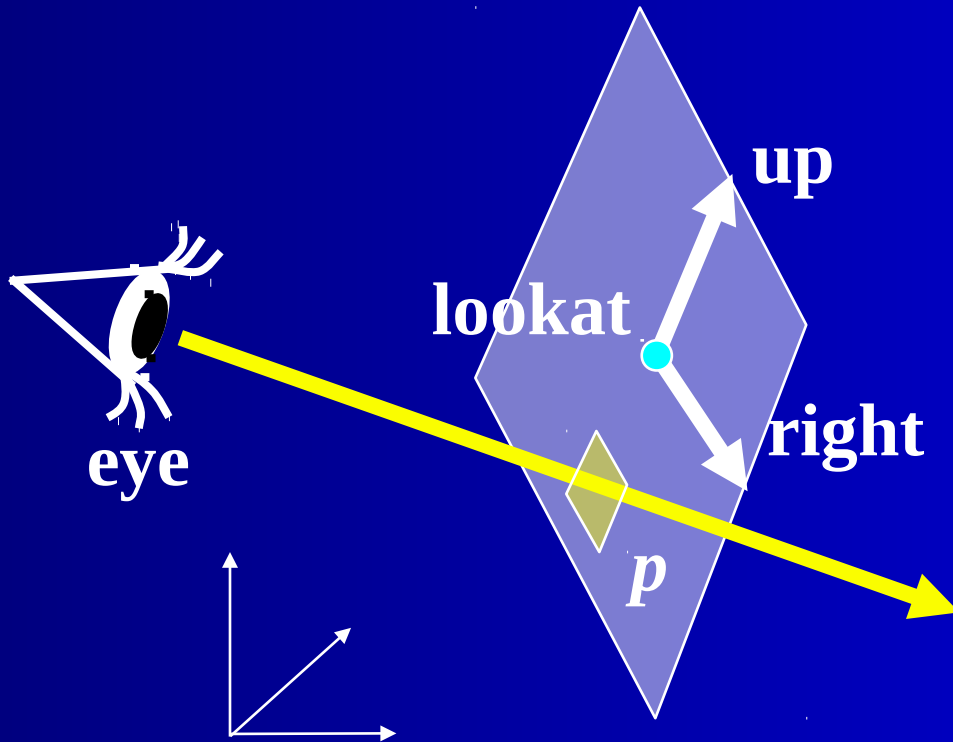
image[pixel] = color;

endfor

end

p color

Kamera



$$p = \text{lookat} + (2X/XM-1) \cdot \text{right} + (2Y/YM-1) \cdot \text{up}$$

+ fél pixel, hogy a középen legyen

trace

Color trace(ray)

hit = intersectAll(ray)

IF (hit.t < 0) RETURN L_a

color = $k_a * L_a$

x = hit.position; $\mathbf{N}$ = hit.normal;

FOR each lightsource l DO

shadowRay.origin = x; shadowRay.dir = light[l].pos - x;
shadowHit = intersectAll(shadowRay); y = shadowHit.pos;
IF (shadowHit.t < 0 || |x-y| > |x-light[l].pos|)

$\mathbf{V} = -\text{Normalize}(\text{ray.dir})$; $\mathbf{L}_l = \text{Normalize}(\text{shadowRay.dir})$;

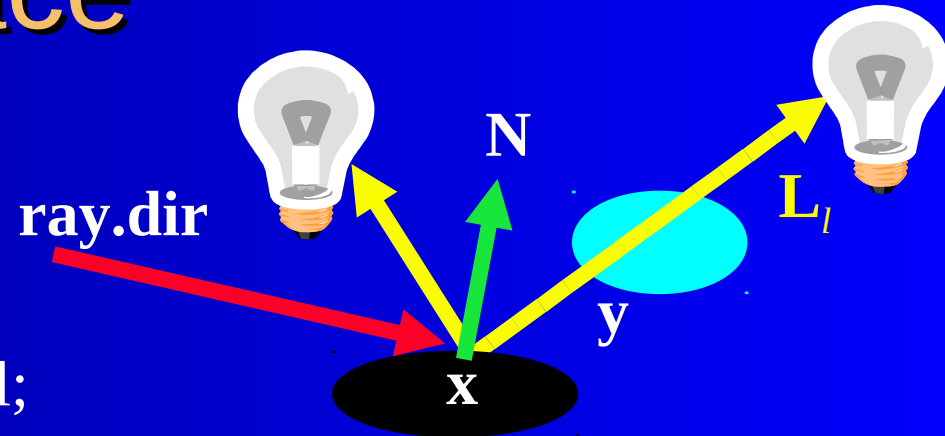
$\mathbf{H}_l = \text{Normalize}(\mathbf{V} + \mathbf{L}_l)$;

color += light[l].intensity * ($k_d \cdot (\mathbf{L}_l \cdot \mathbf{N})^+ + k_s \cdot ((\mathbf{H}_l \cdot \mathbf{N})^+)^{\text{shine}}$)

ENDIF

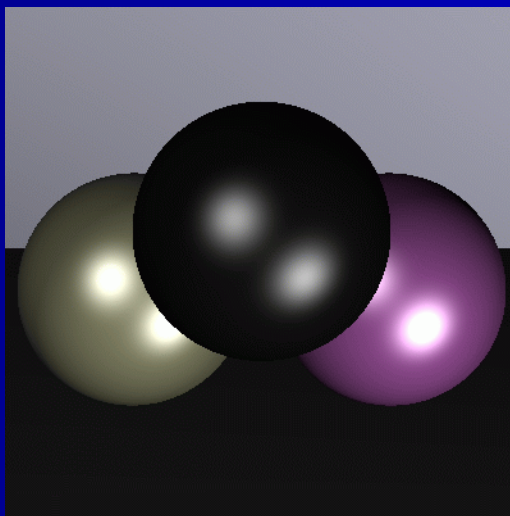
ENDFOR

RETURN color

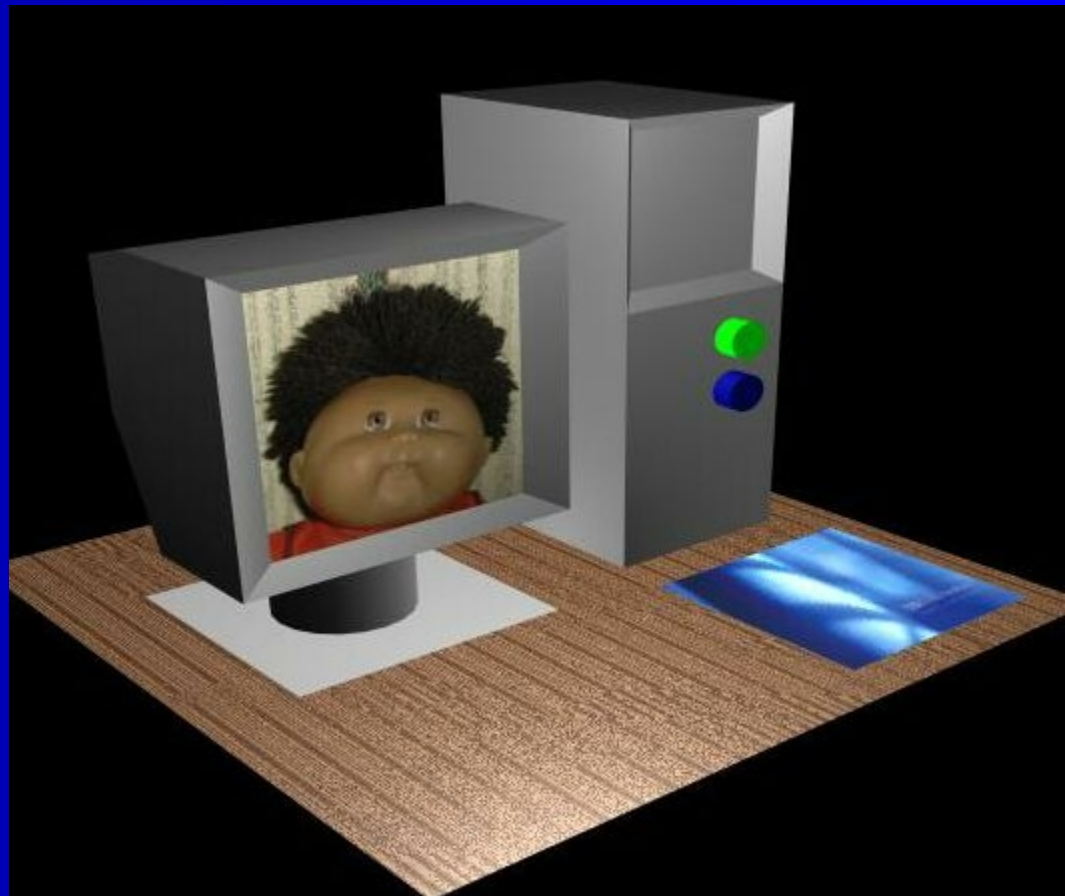
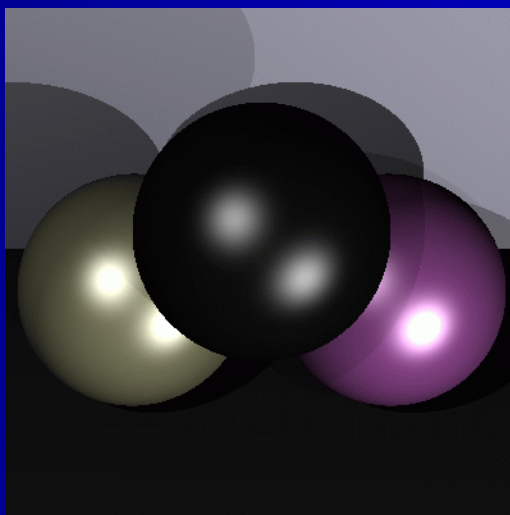


Ray-casting képek

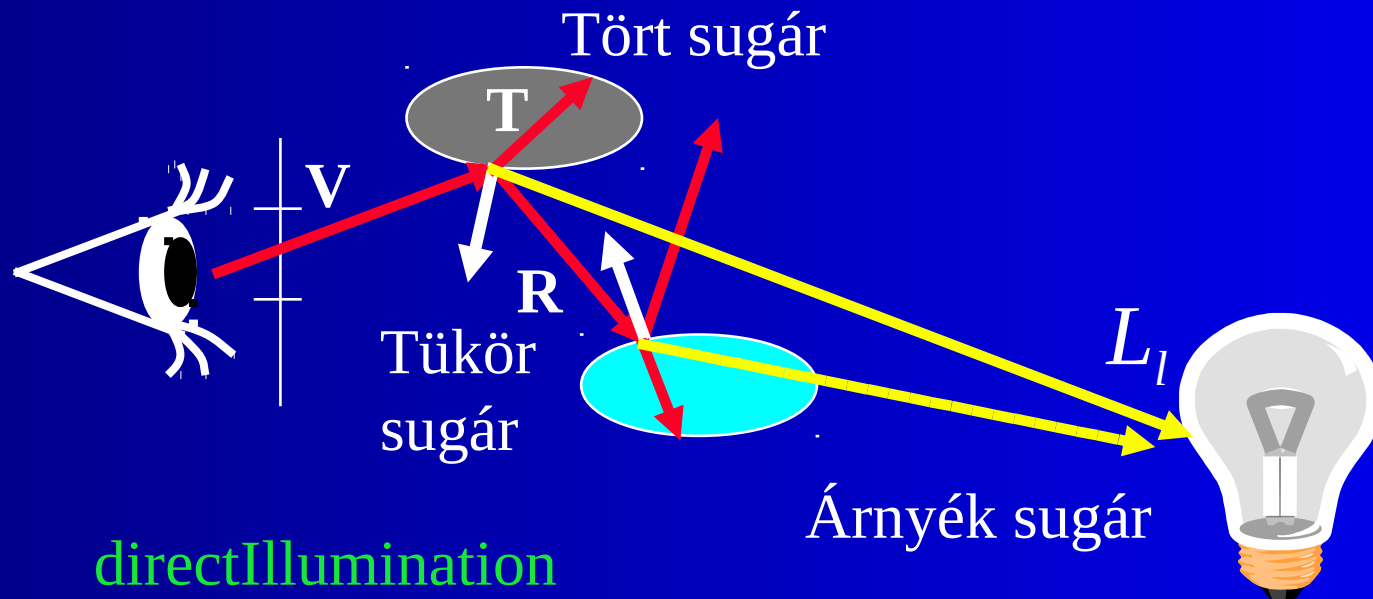
**Lokális
illumináció**



**Lokális
illumináció
+
árnyékok**



Rekurzív sugárkövetés



$$L(V) \approx \sum_l L_l(L_l) * (k_d \cdot (L_l \cdot N)^+ + k_s \cdot ((H_l \cdot N)^+)^{shine}) + k_a * L_a$$

$$+ F(V \cdot N) * L^{in}(R) + (1 - F(V \cdot N)) * L^{in}(T)$$

Fresnel

Tükör irányból
érkező fény

1-Fresnel

Törési irányból
érkező fény

trace

Radiancia ami sugáron visszafelé jön?

- Látható felületi pont és normál vektora
- Látható pontból látható fényforrások
- Ha tükröző, sugár a látható pontból a tükörirányba

– Radiancia ami a tükörsugáron vissza jön

– Ha törő, sugár a látható pontból a törtirányban

– Radiancia ami a törési sugáron vissza jön

– A hozzájárulások összegzése

rekurzió



trace function

Color trace(ray)

hit = intersectAll(ray)

IF (hit.t < 0) RETURN L_a

color = directIllumination

IF (isReflective)

reflected ray = reflect(ray)

 color += Fresnel * trace(reflected ray)

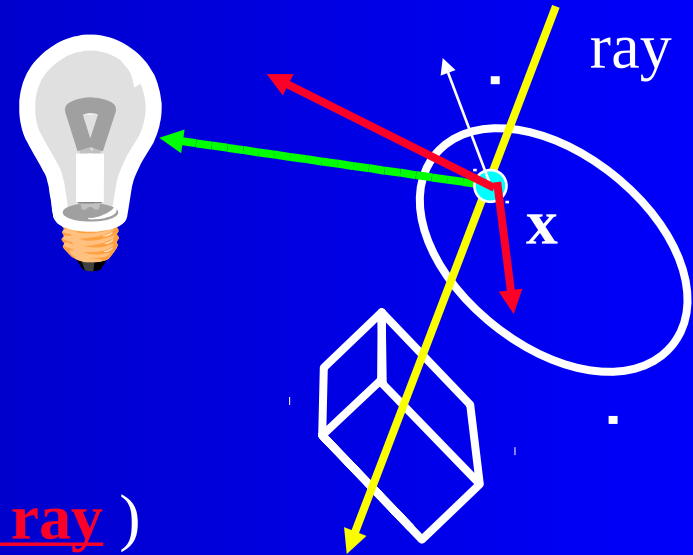
ENDIF

IF (isRefractive && refracted ray = refract(ray))

 color += (Color(1,1,1)-Fresnel) * trace(refracted ray)

ENDIF

RETURN color



trace function

Color trace(ray, d)

IF (d > dmax) RETURN L_a

hit = intersectAll(ray)

IF (hit.t < 0) RETURN L_a

color = directIllumination

IF (isReflective)

reflected ray = reflect(ray)

color += Fresnel * trace(reflected ray, d+1)

ENDIF

IF (isRefractive && refracted ray = refract(ray))

color += (Color(1,1,1)-Fresnel) * trace(refracted ray, d+1)

ENDIF

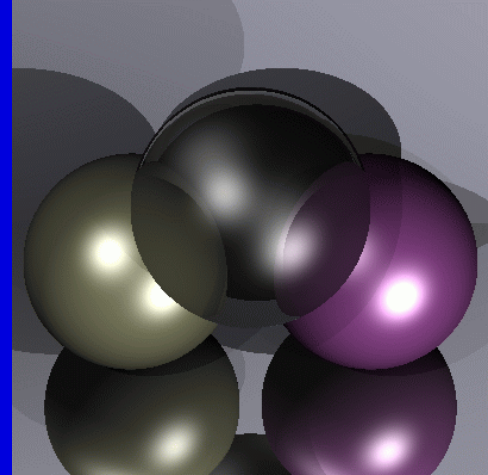
RETURN color

$\text{ray.dir} - 2 \cdot \mathbf{N} \cdot (\mathbf{N} \bullet \text{ray.dir})$

$\mathbf{N} = \text{hit.normal}$

$\text{ray.dir}/n + \mathbf{N} \cdot ((\mathbf{N} \bullet \text{ray.dir})/n - \sqrt{1 - (1 - (\mathbf{N} \bullet \text{ray.dir})^2)/n^2})$

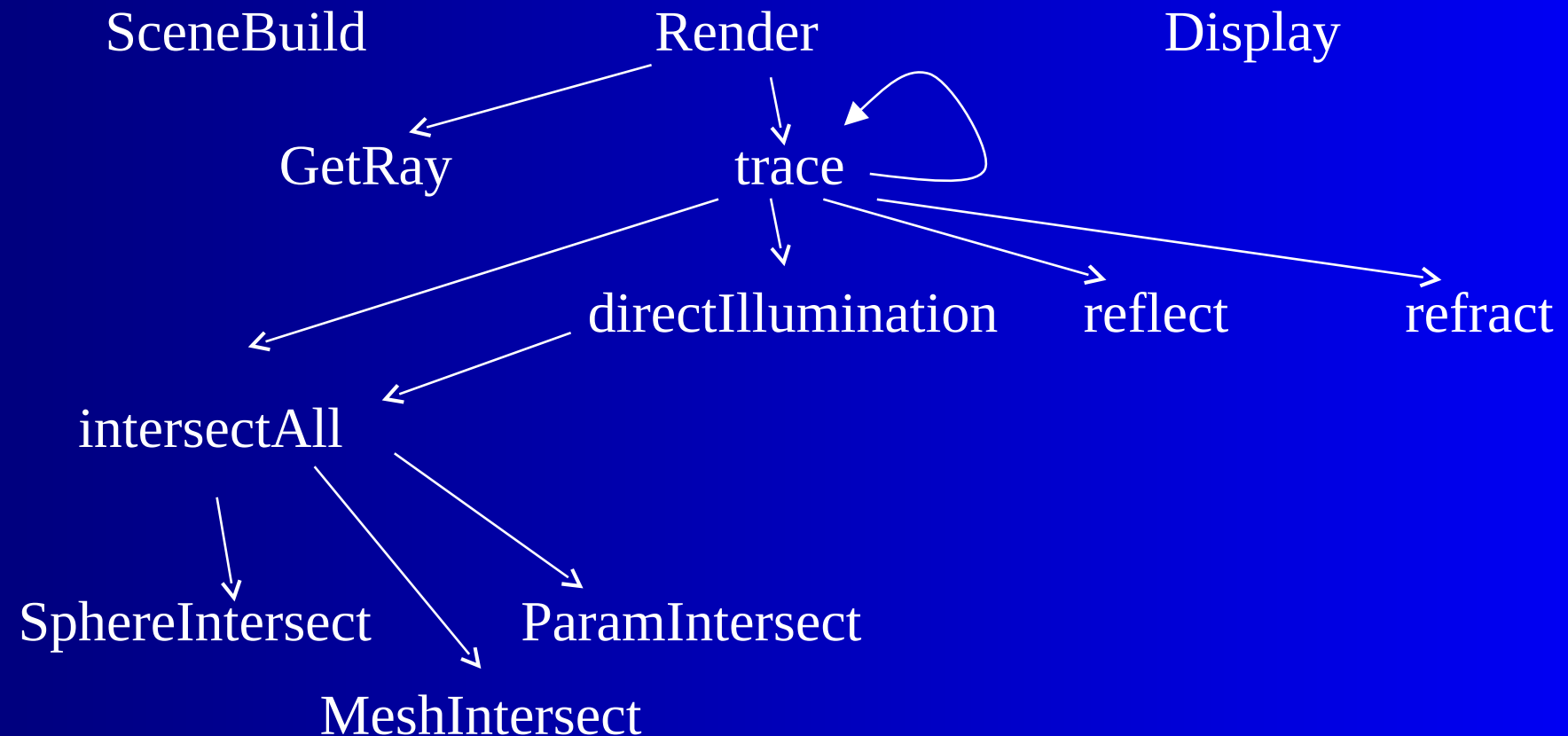
Paul Heckbert névjegye



```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02}; struct sphere{ vec cen,color;double rad,kd,ks,kt,kl,ir}*s,
*best,sph[]={0.,6.,.5,1.,1.,1.,.9,.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,1.,8.,8,1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,
.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,1.,5.,0.,0.,0.,.5,1.5,};yx; double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A ,B;
{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B; {B.x+=a* A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}
vec vunit(A)vec A;{return vcomb(1./sqrt( vdot(A,A)),A,black);}struct sphere *intersect(P,D)vec P,D;{best=0;tmin=1e30;
s= sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s ->rad,u=u>0?sqrt(u):1e31,u=b-u>
1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u: tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb
(-1.,P=vcomb(tmin,D,P),s->cen )));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l ->kl*vdot(N,
U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e ,l->color,color);U=s->color;color.x*=U.x;color.y*=
U.y;color.z*=U.z;e=1-eta* eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt( e),N,black))):
black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd, color,vcomb(s->kl,U,black))));}

main(){printf("%d %d\n",32,32);while(yx<32*32) U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),
U=vcomb(255., trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```

Implementáció funkcionális dekompozícióval



OO dekompozíció

Material

F0, n, type
kd, ks, shine

ReflRadiance

IsReflective

IsRefractive

reflect

refract

Fresnel

Object

Intersect

heterogén
kollekció

Scene

Build

Render

trace

intersectAll

Quadratic

A4x4

Intersect

Mesh

Intersect

Sphere

cent, radius

Intersect

Triangle

r1,r2,r3,N

Intersect

Camera

eye, lookat, up,
right, XM, YM

GetRay

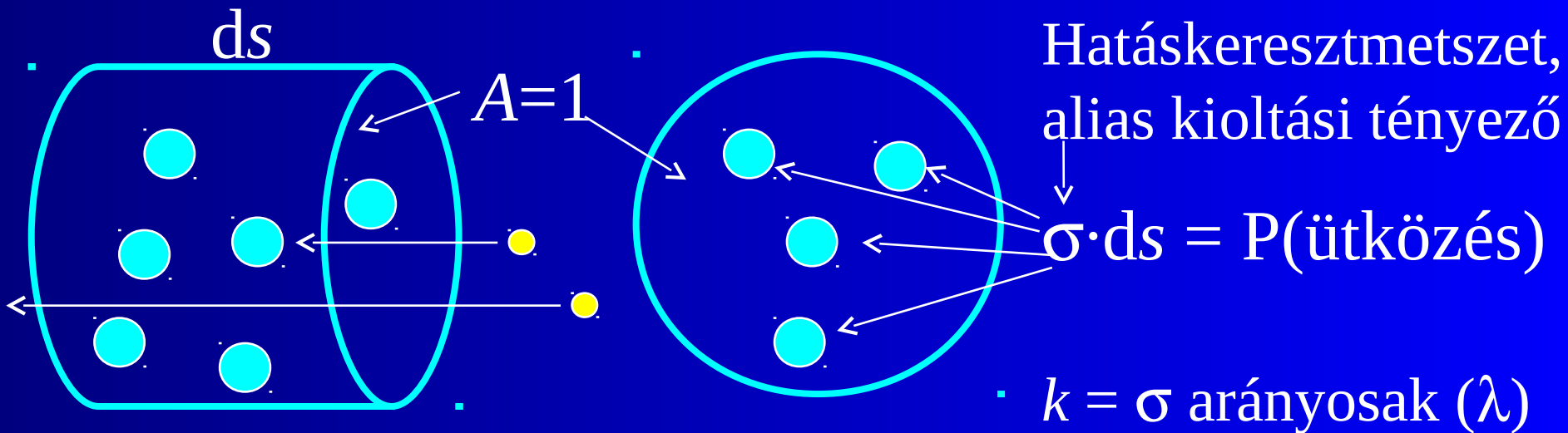
Light

p, Lout, type

GetRad

GetDir

Fényelnyelő közeg



$$L(s+ds) = L(s) - L(s) \cdot \sigma(s) \cdot ds$$

$$\boxed{dL/ds = -L(s) \cdot \sigma(s)}$$

$$dL/L = -\sigma(s)ds$$

$$\ln L(S) - \ln L(0) = -\int^S \sigma(s)ds$$

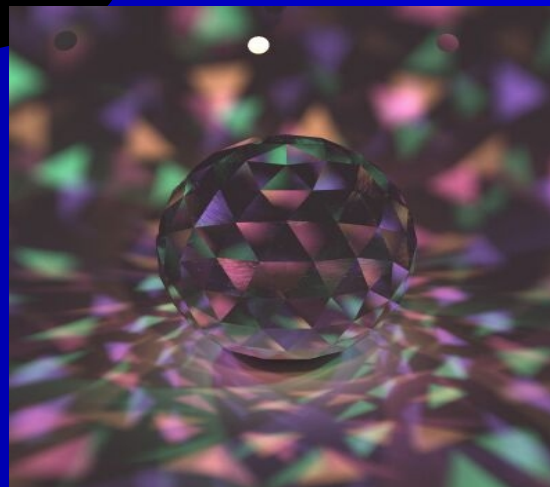
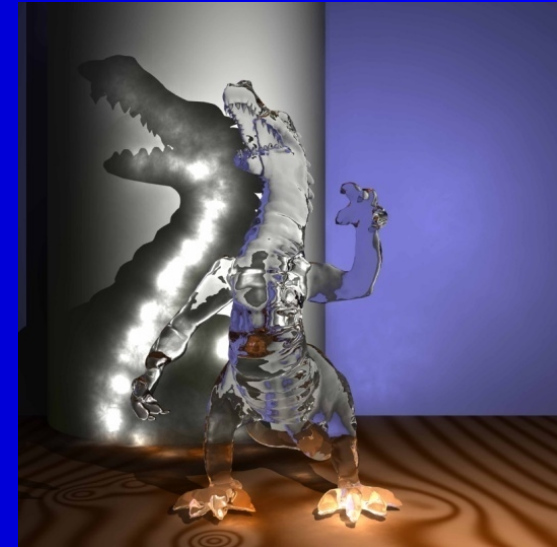
Megoldás fényelnyelő közegre:

$$\boxed{L(S) = L(0) \cdot \exp(-\int^S \sigma(s)ds)}$$

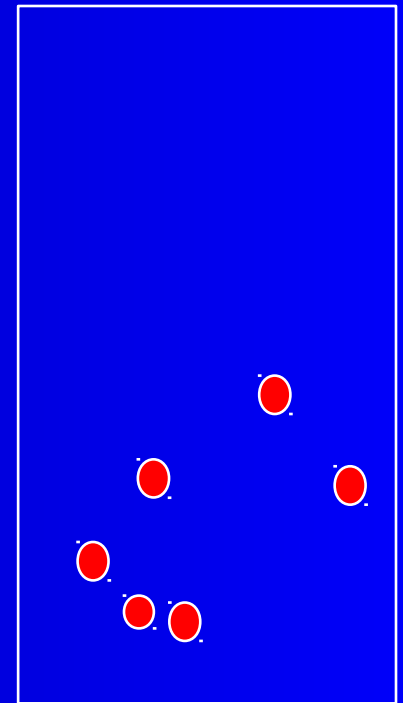
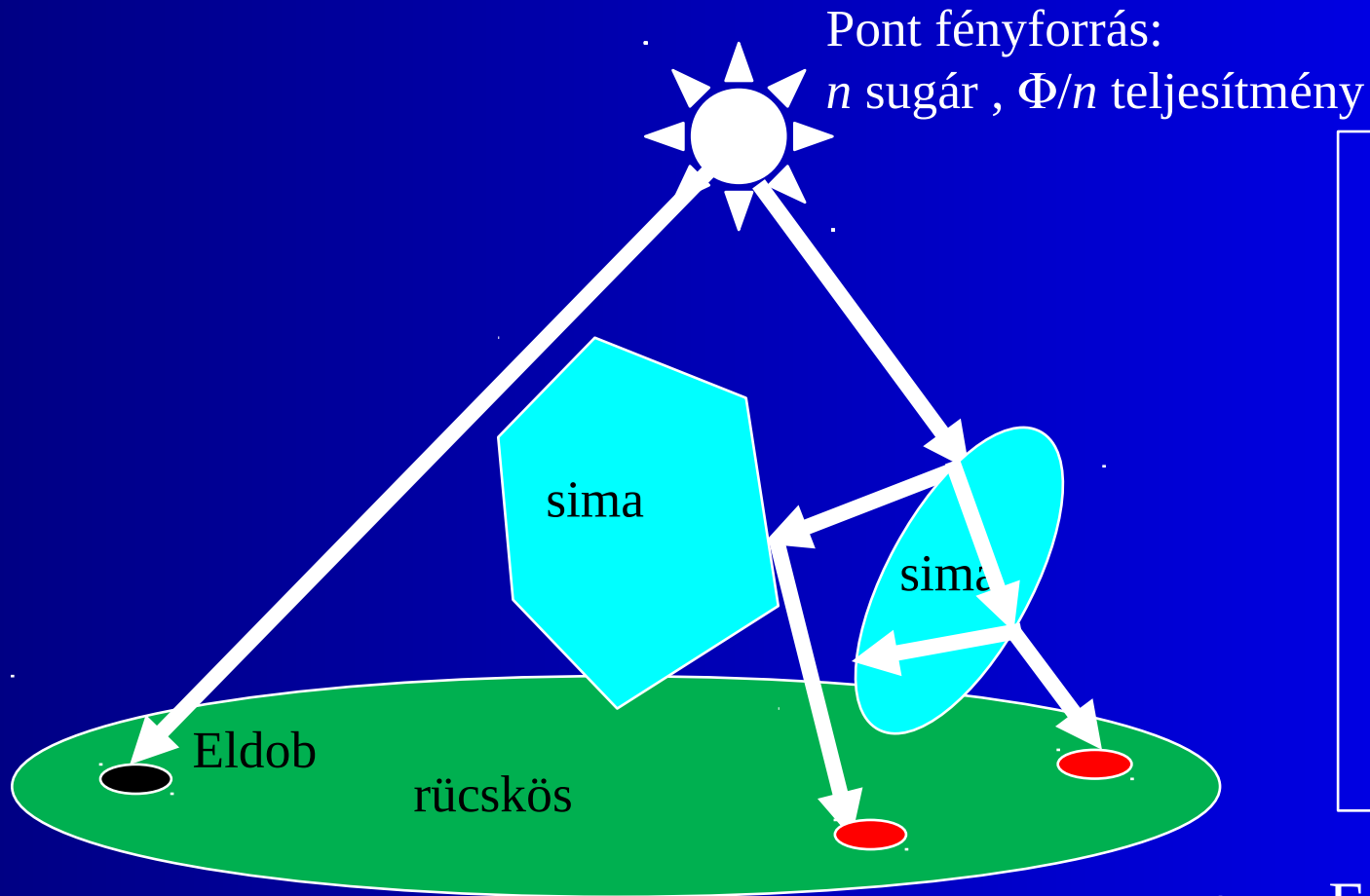
Kétirányú sugárkövetés



Nincsenek
kausztikus
fontok



I. Foton lövés



Foton térkép

Sugárkövetés: Shoot függvény

Shoot(power, ray, d)

IF (d > dmax) RETURN

hit = intersectAll(ray)

IF (hit.t < 0) RETURN

IF (isReflective)

 reflected_ray = reflect(ray)

 Shoot (Fresnel * power, reflected_ray, d+1)

ENDIF

IF (isRefractive && refracted_ray = refract(ray))

 Shoot ((1-Fresnel) * power, refracted_ray, d+1)

ENDIF

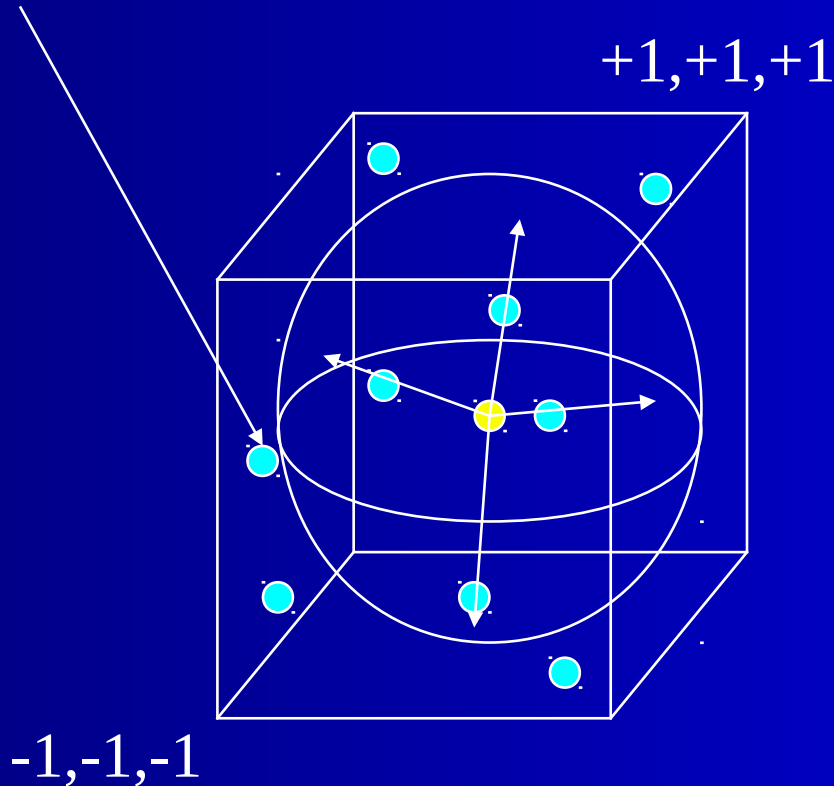
IF (rough material && d > 0) Add (power, x) to photon map

RETURN

Egyenletes eloszlású irányok

```
#define RND (2.0*(double)rand()/RAND_MAX-1.0)
```

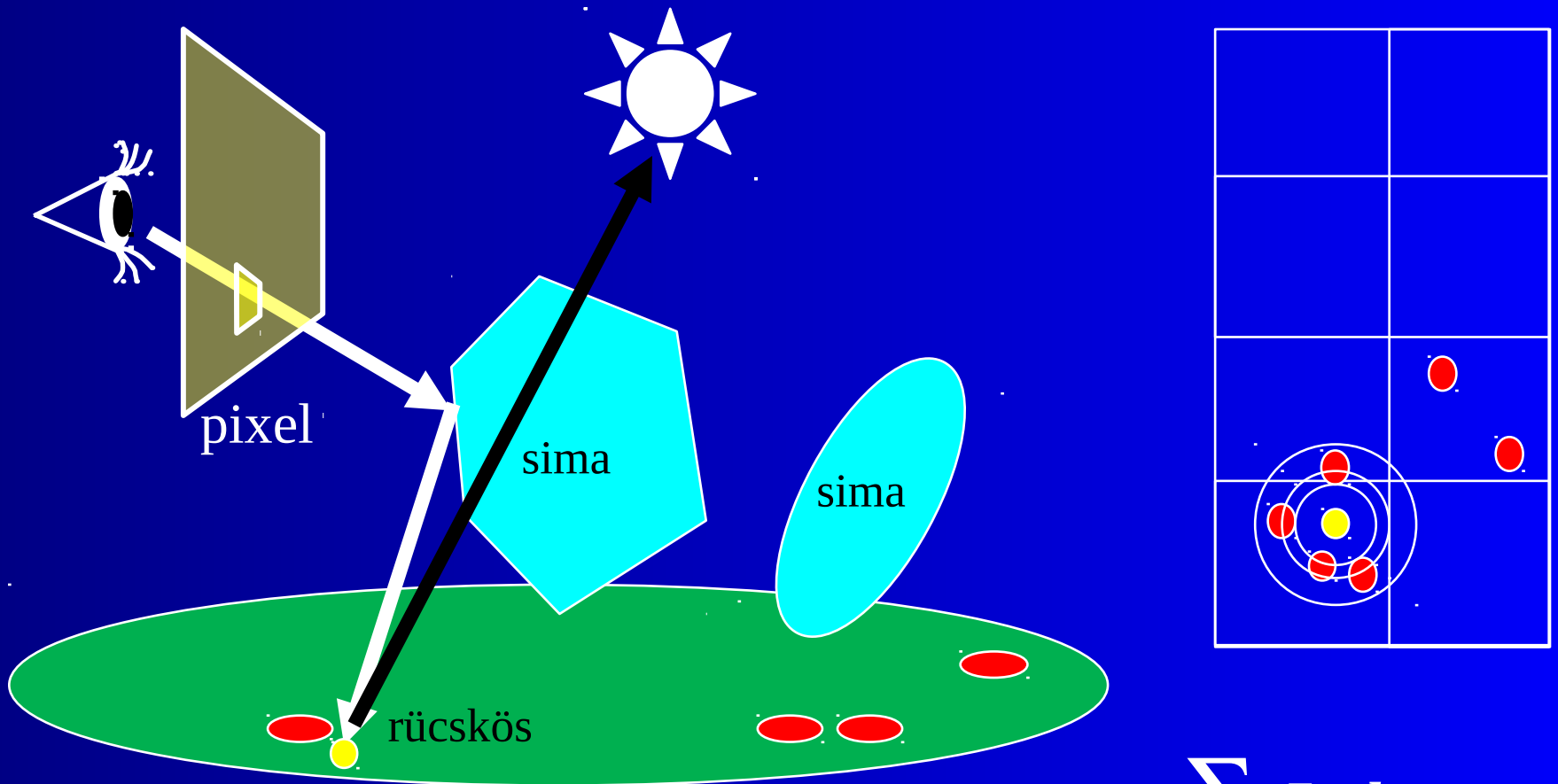
```
(x,y,z)=(RND, RND, RND);
```



```
do {  
    x = RND; y = RND; z=RND;  
} while (x*x + y*y + z*z > 1);
```

```
len = sqrt(x*x + y*y + z*z);  
x /= len; y /= len; z /= len;
```

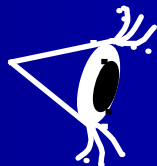
II. Radiancia gyűjtés



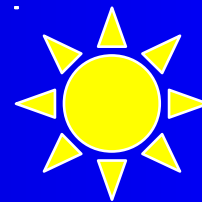
$$L \approx \sum \Phi_i k_d / A$$

3. házi: CSG sugárkövetés

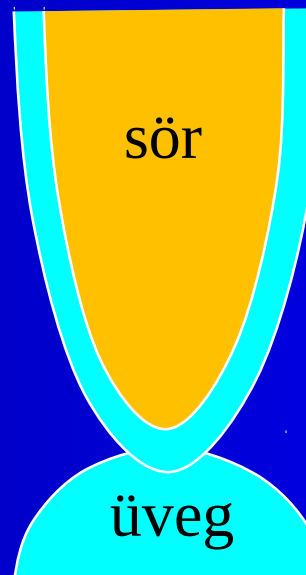
Világoskék égbolt



Erős, sárgás
pontfényforrás



Piros-fehér diffúz abrosz

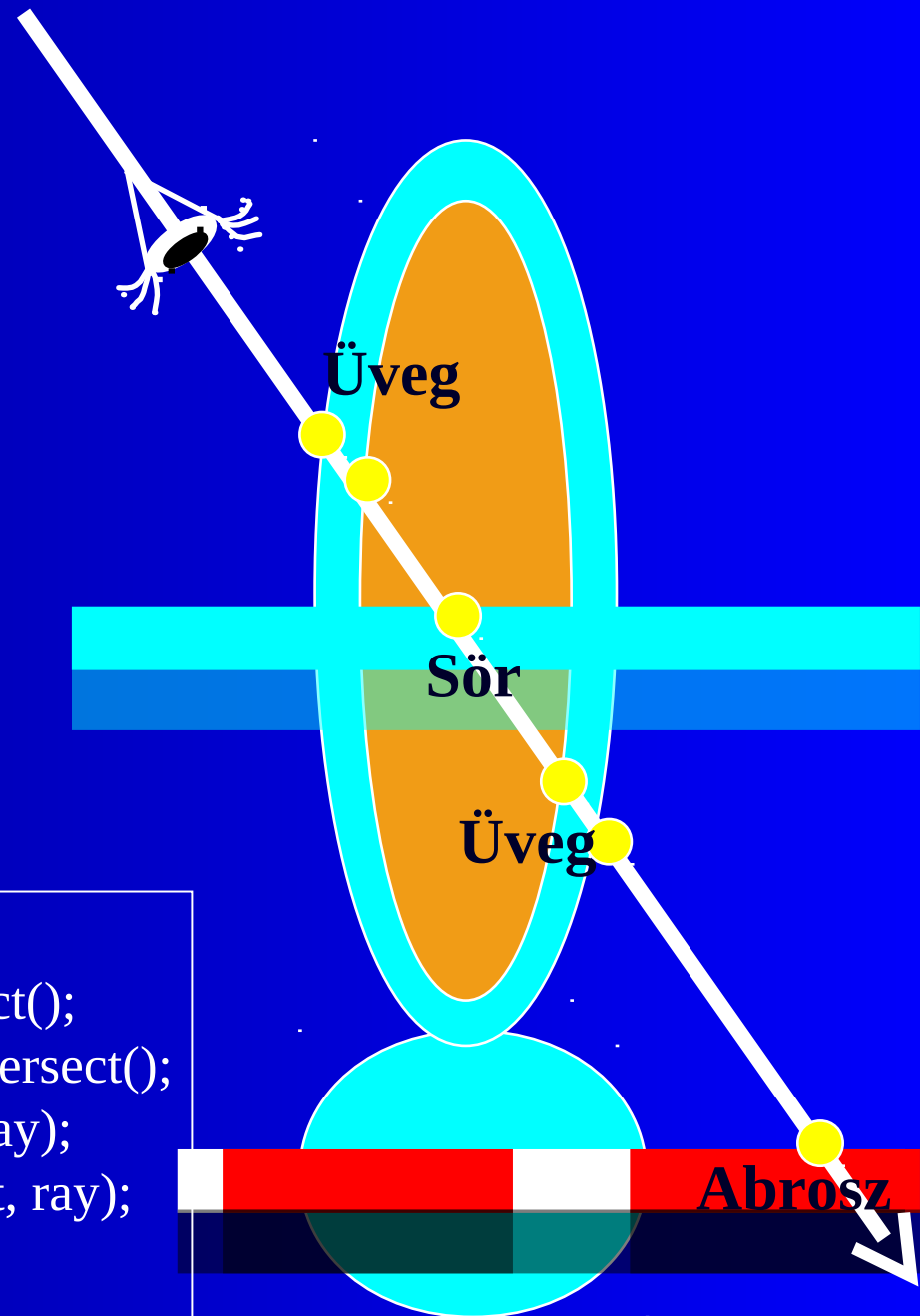


- Primitívek: **csak** általános kvadratikus felület, féltér
- Üveg: $n=1.5$, $\sigma = (?, ?, ?)[\underline{1/m}]$, Sör: $n=1.3$, $\sigma = (?, ?, ?)[\underline{1/m}]$

Halmaz műveletek



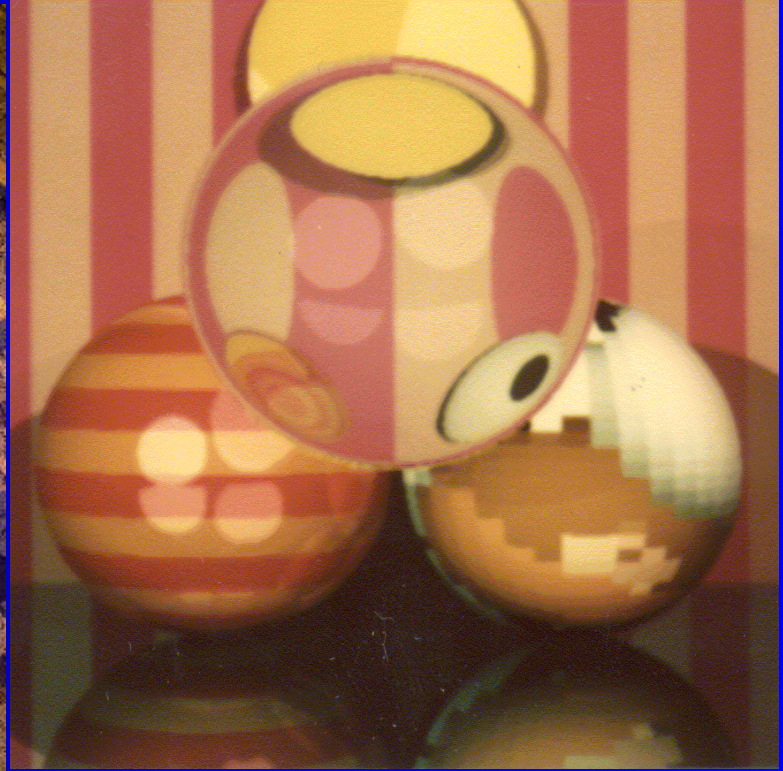
```
hitList intersectAll(node, ray) {  
    if (node->type==Elli) return ElliIntersect();  
    if (node->type==Felter) return FelterIntersect();  
    hitList hleft = intersectAll(node->left, ray);  
    hitList hright = intersectAll(node->right, ray);  
    return Merge(node->type, hleft, hright);  
}
```



Tanácsok

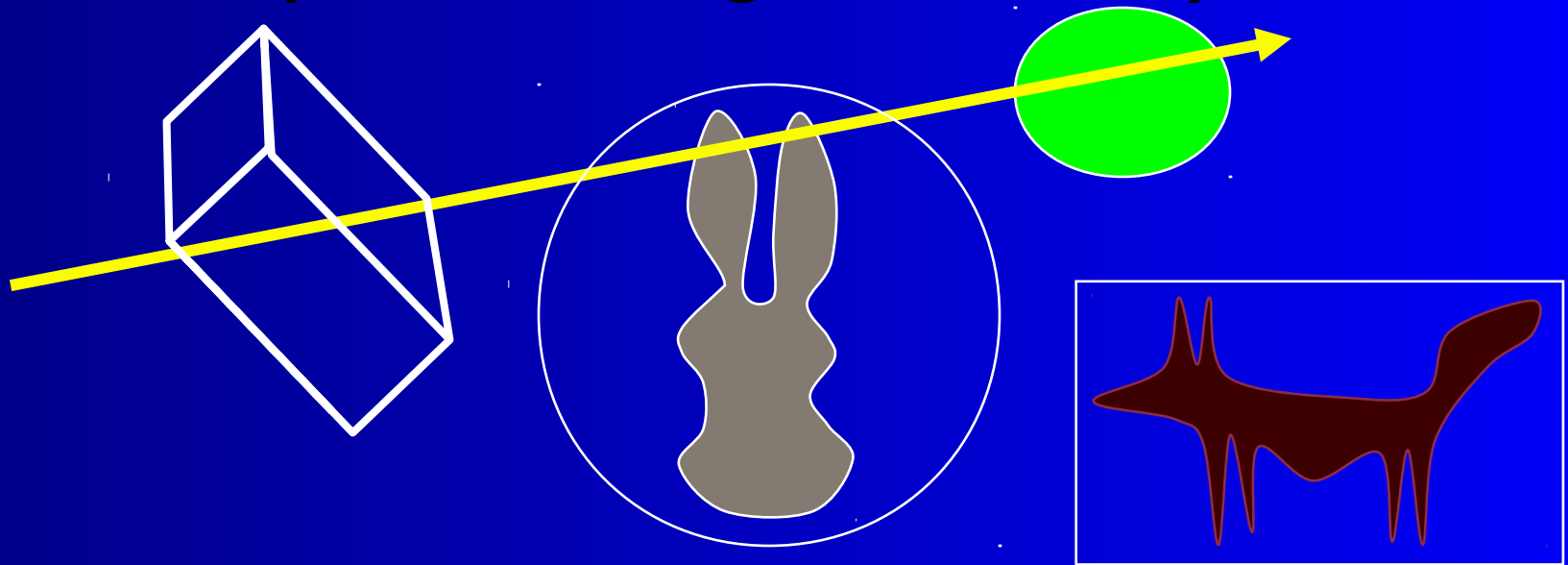
- Tervezd meg a színteret (koordináta rendszer, egység, felül/oldal nézet számokkal).
- Tervezd meg a programot (SD, OO)
- Árnyék pattanás:
 - A $t > 0$ helyett $t > \text{EPSILON}$
- A hátsó lap találatnál a normálvektort fordítani kell (N és V skaláris szorzatának előjele).

Sugárkövetés: eredmény



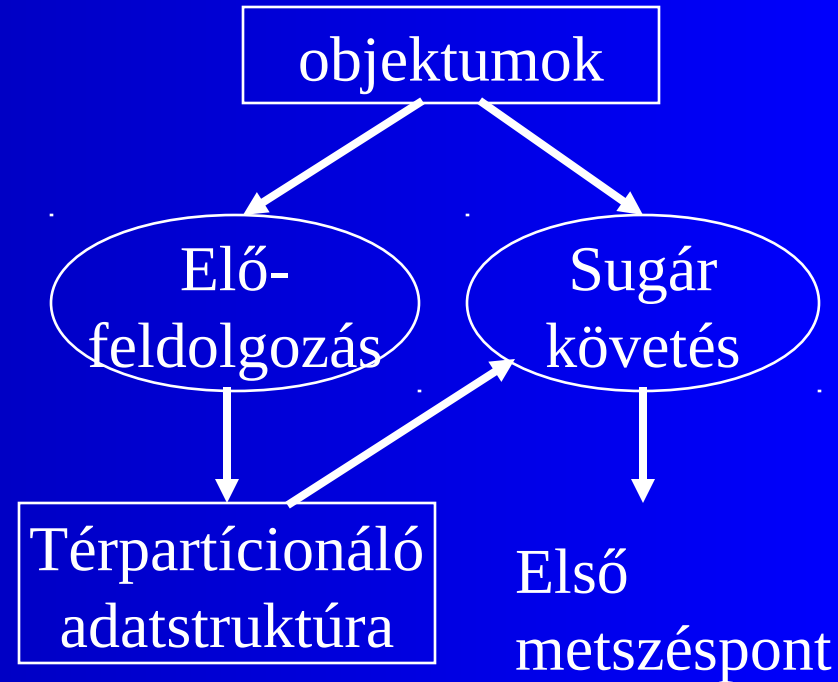
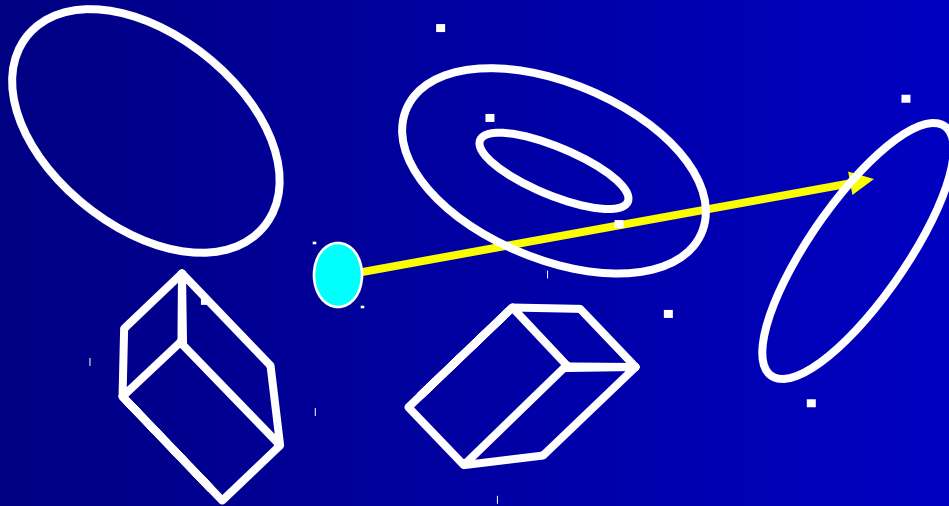
Számítási idő $\propto$ Pixelszám $\times$ Objektumszám $\times$ (Fényforrás szám+1)

Befoglaló térfogat (Bounding Volume)



```
double IntersectBV( ray, object )      // < 0 ha nincs  
    IF ( Intersect( ray, bounding volume of object ) < 0 ) RETURN -1;  
    RETURN Intersect( ray, object );  
END
```

Térpartícionáló módszerek



Adatstruktúra:

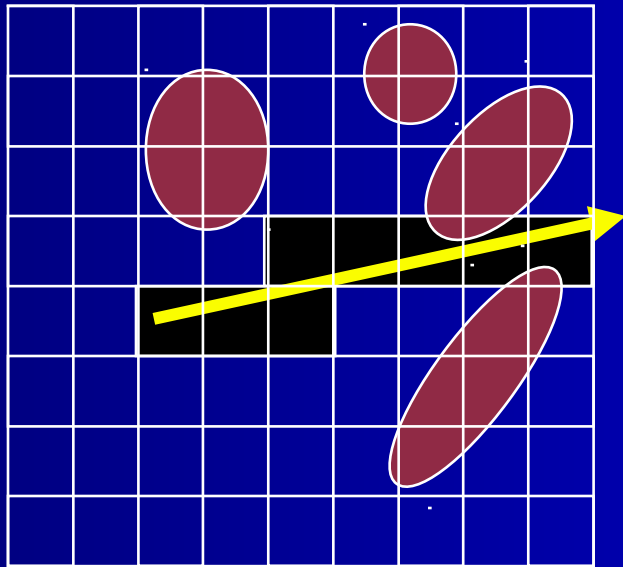
- Ha ismert a sugár, akkor a potenciális metszett objektumok számát csökkenti
- Ha a potenciálisak közül találunk egyet, akkor a többi nem lehet közelebb

Reguláris térháló

Előfeldolgozás:

Minden cellára a metszett objektumok

komplexitás: $O(n \cdot c) = O(n^2)$



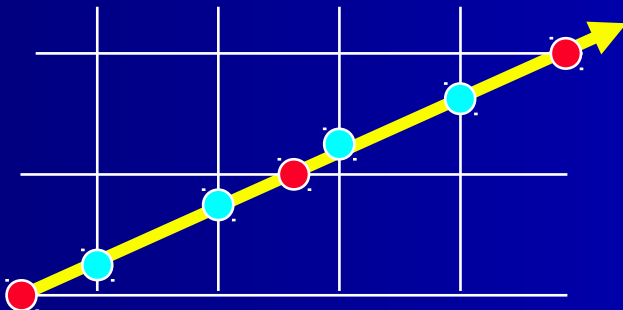
Sugárkövetés:

FOR each cell of the line // line drawing

Metszés a cellában lévőekkel

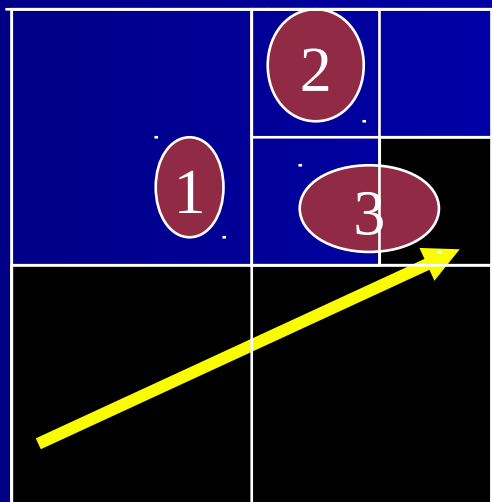
IF van metszés RETURN

ENDFOR



átlagos eset komplexitás: $O(1)$

Nyolcas (oktális) fa



Faépítés(cella):

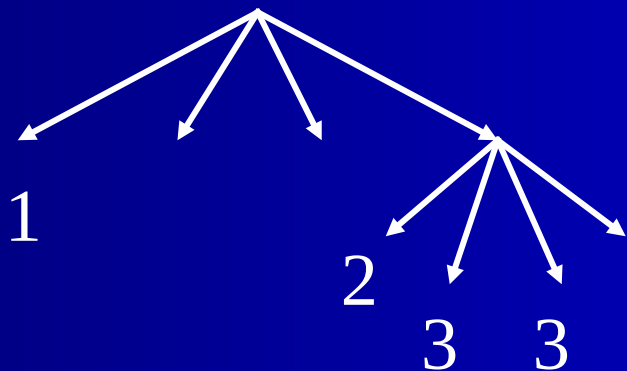
IF a cellában kevés objektum van
cellában regisztráld az objektumokat

ELSE

cellafelezés: $c1, c2, \dots, c8$

Faépítés($c1$); ... Faépítés($c8$);

ENDIF



Octree

Sugárkövetés:

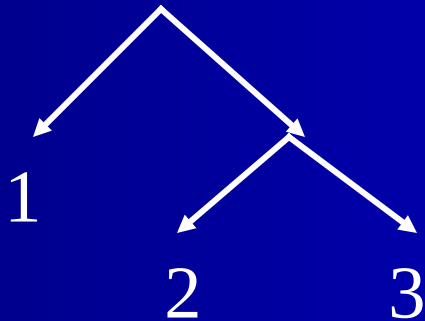
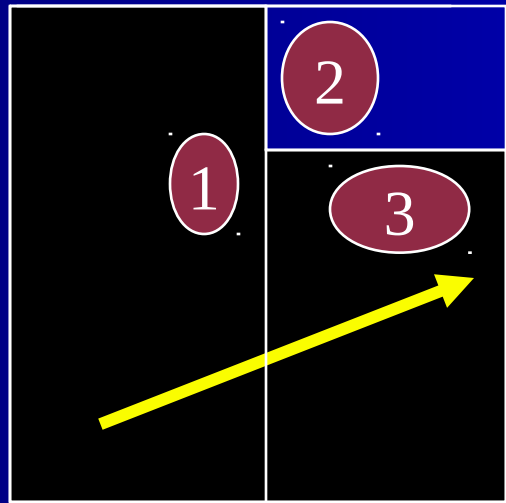
FOR összes sugár által metszett cellára

Metszés a cellában lévővel

IF van metszés RETURN

ENDFOR

Binary Space Partitioning fa (kd-fa)



kd-tree

Faépítés(cella):

IF a cellában kevés objektum van
cellában regisztráld az objektumokat

ELSE

sík keresés

cella felezés a síkkal: c1, c2

Faépítés(c1); Faépítés(c2);

ENDIF

Sugárkövetés:

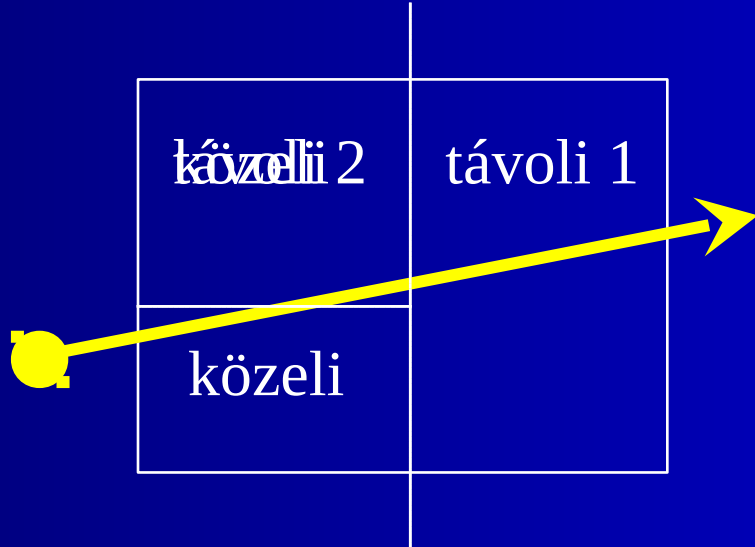
FOR each cell intersecting the line

Metszés a cellában lévőekkel

IF van metszés RETURN

ENDFOR

kd-fa bejárása



```
while (stack nem üres) {
  Pop(AABB)
```

```
  while (nem levél) {
```

```
    Ha van AABB-nek távoli
```

```
      Push(távoli)
```

```
      AABB = közeli
```

```
  }
```

```
  Regisztrált objektumok metszése
```

```
  Ha van metszéspon return
```

```
}
```

```
  Ha van távoli AABB
  távoli 1
```

Inkrementális 3D képszintézis

Szirmay-Kalos László

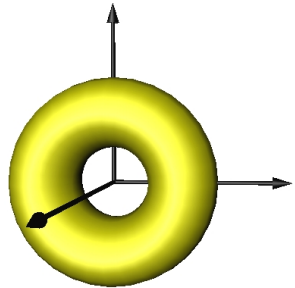
Inkrementális képszintézis

- Sugárkövetés számítási idő $\propto$

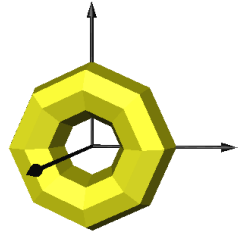
Pixelszám $\times$ Objektumszám $\times$ (Fényforrás szám+1)

- koherencia: oldjuk meg nagyobb egységekre
- feleslegesen ne számoljunk: vágás
- transzformációk: minden feladathoz megfelelő koordinátarendszert
 - vágni, transzformálni nem lehet akármilyen: tesszelláció

3D inkrementális képszintézis

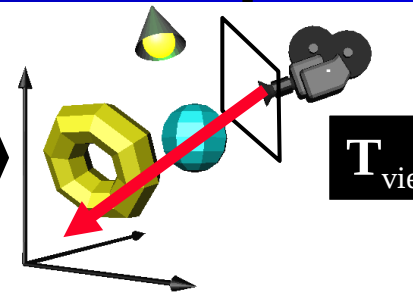


Modell
referencia helyzetben



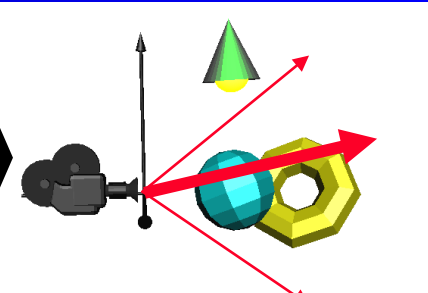
Tesszelláció

T_{modell}

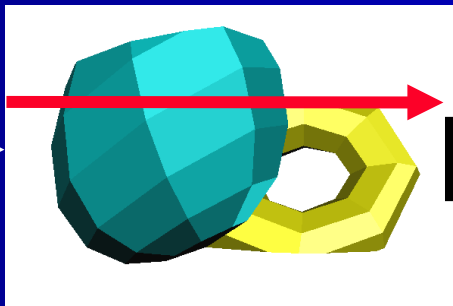


Világkoordináta r.

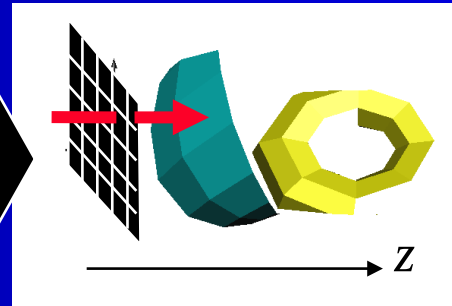
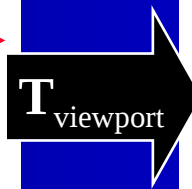
T_{view}



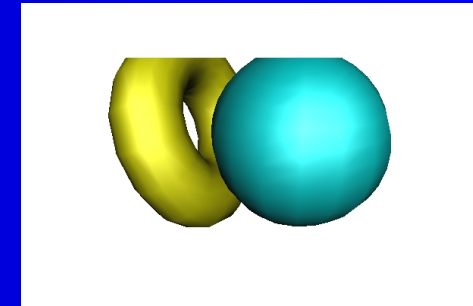
Kamerakoordináta r.



Normalizált
eszközkoordináta r.
Vágás



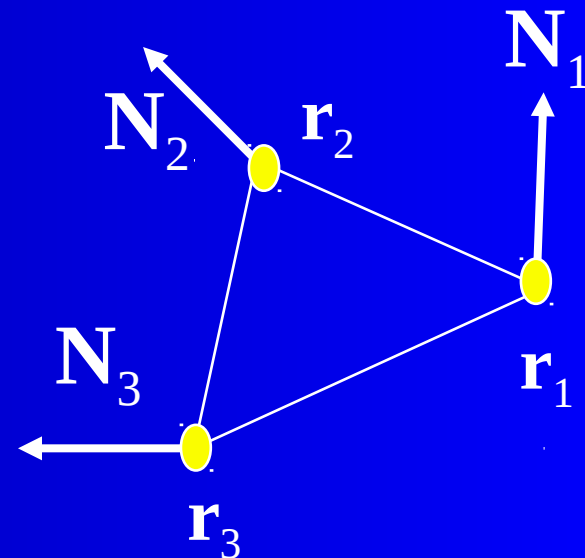
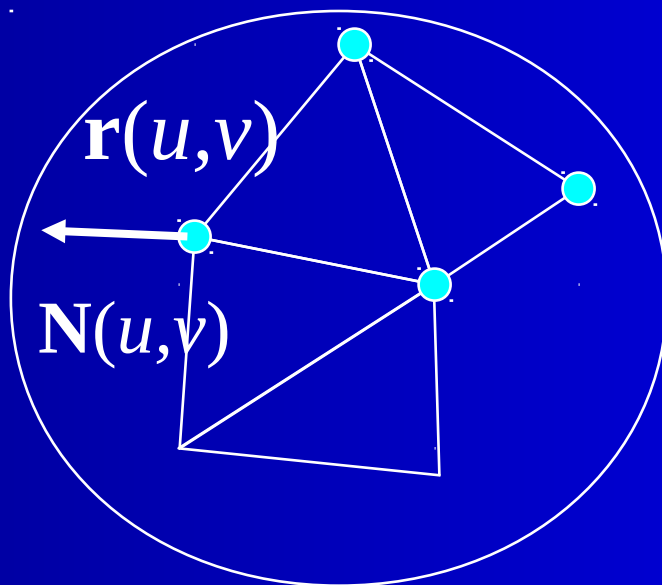
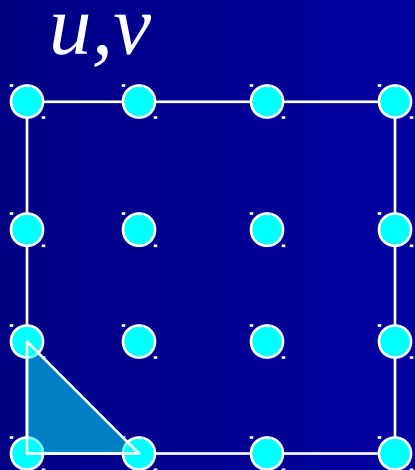
Képernyőkoordináta r.
Láthatóság + vetítés
Takarás és vetítés triviális!



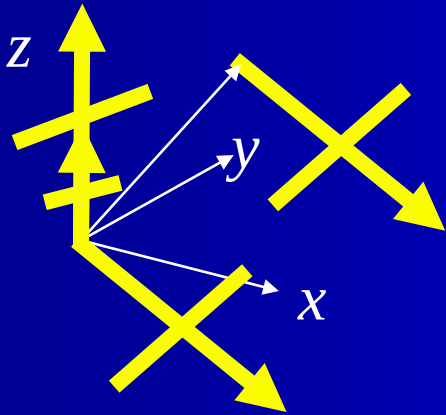
Árnyalt kép

Tesszelláció

- Felületi pontok kijelölése: $\mathbf{r}_{n,m} = \mathbf{r}(u_n, v_m)$
- Normálvektor:
$$\mathbf{N}(u_n, v_m) = \frac{\partial \mathbf{r}(u, v)}{\partial u} \times \frac{\partial \mathbf{r}(u, v)}{\partial v}$$
- „Paraméterterben szomszédos” pontokból háromszögek

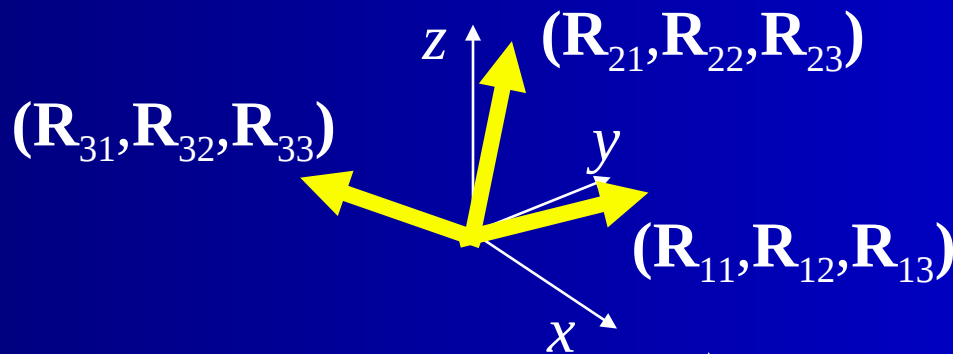


Modellezési transzformáció

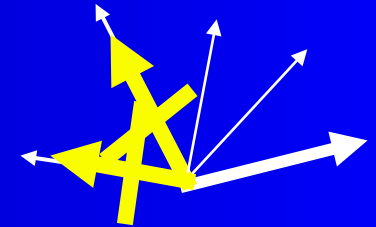
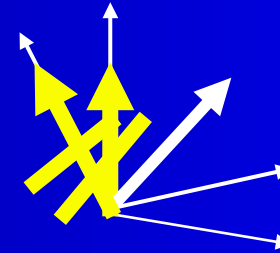
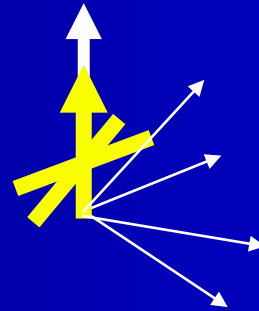


1. skálázás: s_x, s_y, s_z
2. orientáció: α, β, γ
3. pozíció: p_x, p_y, p_z

$$\mathbf{T}_M = \begin{bmatrix} s_x & & & \\ & s_y & & \\ & & s_z & \\ & & & 1 \end{bmatrix} \mathbf{R} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1 & p_x & p_y & p_z \end{bmatrix}$$



Orientáció



$$\mathbf{T}_M = \begin{bmatrix} \mathbf{R} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha & & \\ -\sin\alpha & \cos\alpha & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & -\sin\beta & & \\ & 1 & & \\ \sin\beta & \cos\beta & & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ \cos\gamma & \sin\gamma & & \\ -\sin\gamma & \cos\gamma & & \\ & & & 1 \end{bmatrix}$$

Csavaró: Billentő: Forduló:

Orientáció = ortonormált mátrix

Roll

Pitch

Yaw

Orientáció: 3 szabadságfok

Orientáció = három szög+fix tengelyek vagy tengely+szög

Emlékeztető: Más algebrai alap?

- Idáig (2D-ben)

- Pont = számpár
- Eltolás = számpár (vektor)
- Elforgatás, skálázás, nyírás, stb. = 2×2 -s mátrix

- Más lehetőség?

- Pont = komplex szám (z)
- Eltolás = komplex szám (összeadás): $z' = z + v$
- Forgatás, skálázás (forgatva nyújtás) = komplex szám (szorzás):
$$z' = z * f, \text{ ahol } f = r e^{i\varphi}$$

Forgatás =
egység abszolút
értékű komplex
számmal szorzás

Kvaterniók

- Komplex számok általánosítása
- $\mathbf{q} = [s, x, y, z] = [s, \mathbf{w}] = s + xi + yj + zk$
- $\mathbf{q}_1 + \mathbf{q}_2 = [s_1 + s_2, x_1 + x_2, y_1 + y_2, z_1 + z_2]$
- $a\mathbf{q} = [\underline{as, ax, ay, az}]$
- $|\mathbf{q}| = \sqrt{s^2 + x^2 + y^2 + z^2}$
- Szorzás:
 - $\boxed{i^2 = j^2 = k^2 = ijk = -1}, ij=k, ji=-k, jk=i, kj=-i, ki=j, ik=-j$Szorzás asszociatív, de nem kommutatív,
Összeadásra disztributív
 - Van egységelem: $[1, 0, 0, 0]$
 - Van inverz, konjugált/absnégyzet mint a komplexnél:
 $\mathbf{q}^{-1} = [s, -\mathbf{w}] / |\mathbf{q}|^2, \mathbf{q}^{-1} \cdot \mathbf{q} = \mathbf{q} \cdot \mathbf{q}^{-1} = [1, 0, 0, 0]$

Egység kvaternió = forgatás az origón átmenő w tengely körül

- $q = [\cos(\alpha/2), w \sin(\alpha/2)], \quad |w| = 1$
- $q \cdot [0, u] \cdot q^{-1} = [0, v]$
 - v az u elforgatottja a w körül α szöggel
- w egységvektor körül α szöggel forgató mátrix:
 - $(0, R_{11}, R_{12}, R_{13}) = q \cdot [0, 1, 0, 0] \cdot q^{-1}$
 - $(0, R_{21}, R_{22}, R_{23}) = q \cdot [0, 0, 1, 0] \cdot q^{-1}$
 - $(0, R_{31}, R_{32}, R_{33}) = q \cdot [0, 0, 0, 1] \cdot q^{-1}$
- Rodriguez formula
$$\text{glRotate}(\alpha, w_x, w_y, w_z);$$

Kvaternió vers

Specifikációhoz és
főleg interpolációhoz
kvaternió

- Orientáció specifikációja:
 - Kvaternió: tengely + szög
 - Mátrix: három Euler szög (elemi forgatások a koordinátatengelyek körül)
- Orientáció interpolációja:
 - Kvaternió: „közbülső” egységkvaterniók, természetes
 - Mátrix: Euler szögek, természetellenes
- Orientációváltás végrehajtása:
 - Kvaternió: 2 kvaternió szorzás, forgatásokkal konkatenálható!
 - Mátrix: vektor-mátrix szorzás, bármilyen homogén-lináris transzformációval konkatenálható!

Az orientáció váltás
végrehajtásához mátrix

Transzformációk

- Modellezési transzformáció: ✓

$$[\mathbf{r}, 1] \mathbf{T}_{\text{modell}} = [\mathbf{r}_{\text{világ}}, 1]$$

$$[\mathbf{N}, 0] (\mathbf{T}_{\text{modell}}^{-1})^T = [\mathbf{N}_{\text{világ}}, 0]$$

- Kamera transzformáció:

$$[\mathbf{r}_{\text{világ}}, 1] \mathbf{T}_{\text{view}_1} = [\mathbf{r}_{\text{kamera}}, 1]$$

$$[\mathbf{N}_{\text{világ}}, 0] (\mathbf{T}_{\text{view}})^T = [\mathbf{N}_{\text{kamera}}, 0]$$

- Perspektív transzformáció:

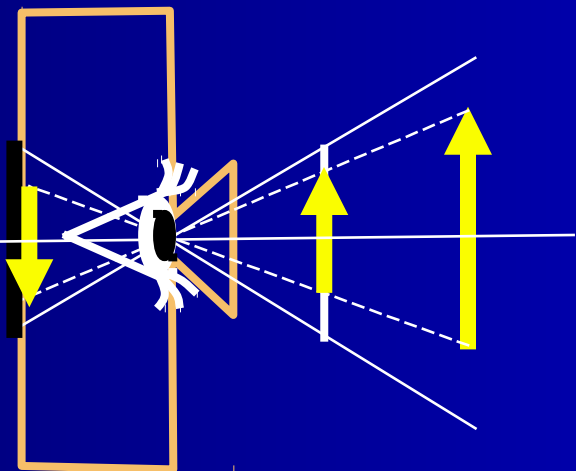
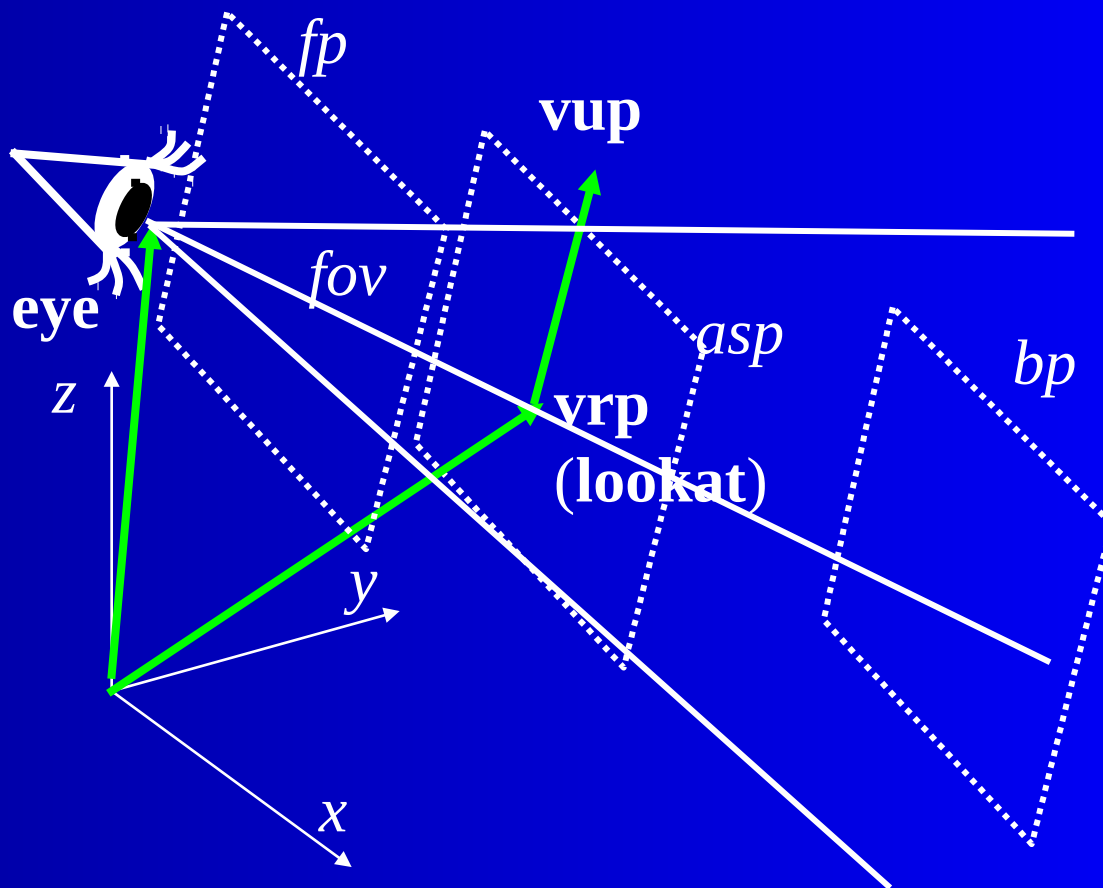
$$[\mathbf{r}_{\text{kamera}}, 1] \mathbf{T}_{\text{persp}} = [\mathbf{r}_{\text{képernyő}}, h, h]$$

- Összetett transzformáció: $\mathbf{T}_{\text{modell}} \mathbf{T}_{\text{view}} \mathbf{T}_{\text{persp}} = \mathbf{T}_{\text{C}}$

szem

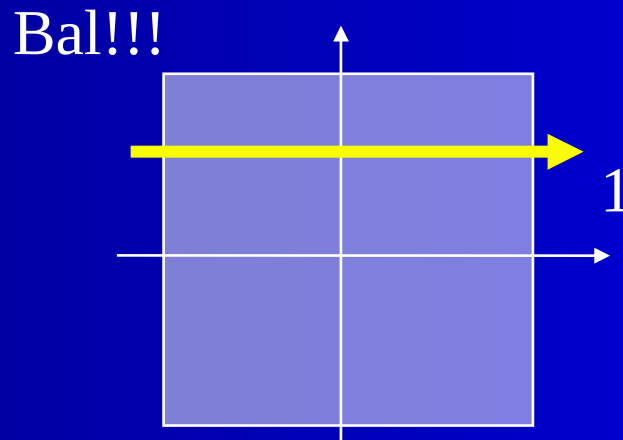
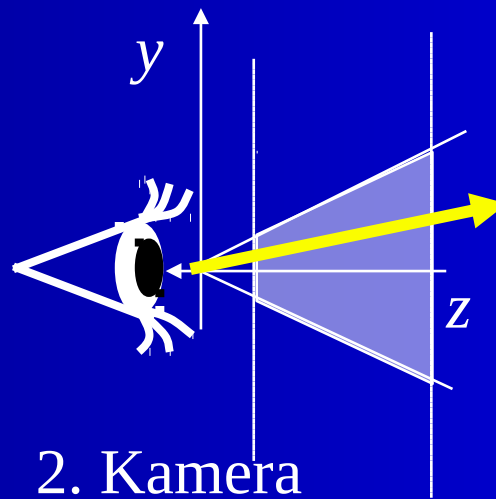
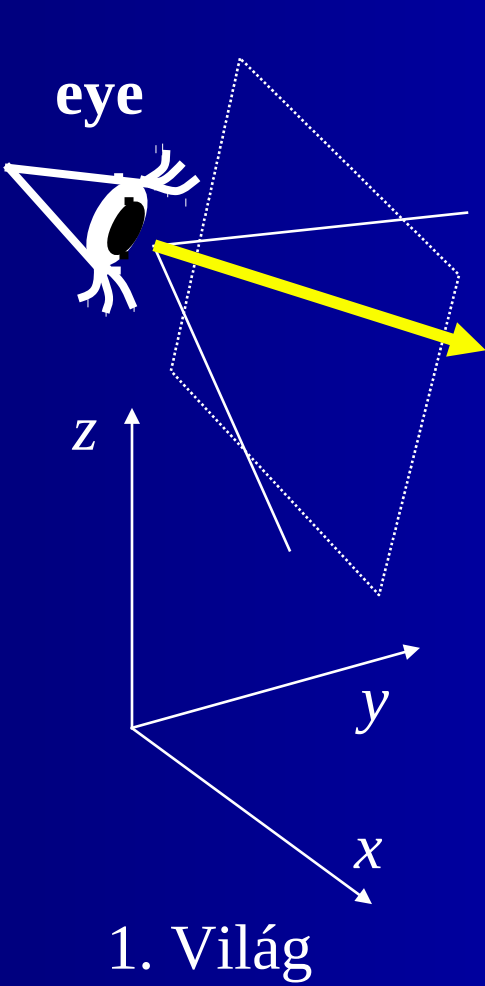
Nézeti
téglalap

Kamera modell

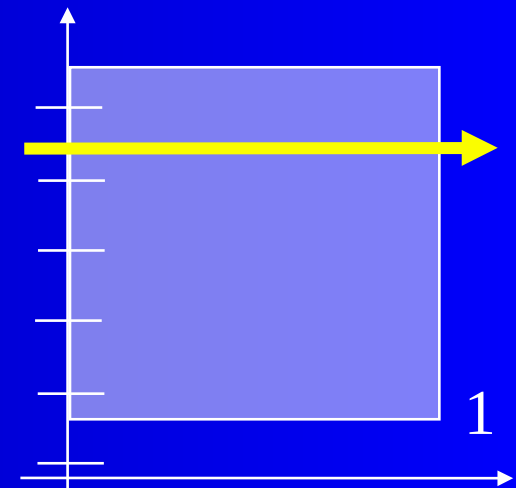


Mi: Camera obscura

Világból a képernyőre

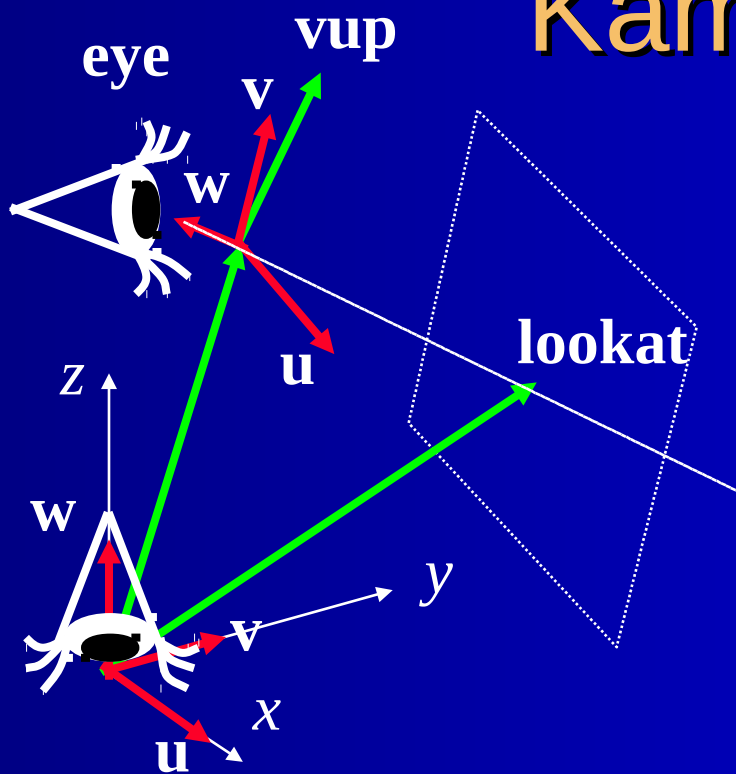


3. Normalizált képernyő



4. Képernyő

Kamera transzformáció



$$\mathbf{w} = (\text{eye} - \text{lookat}) / |\text{eye} - \text{lookat}|$$

$$\mathbf{u} = (\mathbf{vup} \times \mathbf{w}) / |\mathbf{w} \times \mathbf{vup}|$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

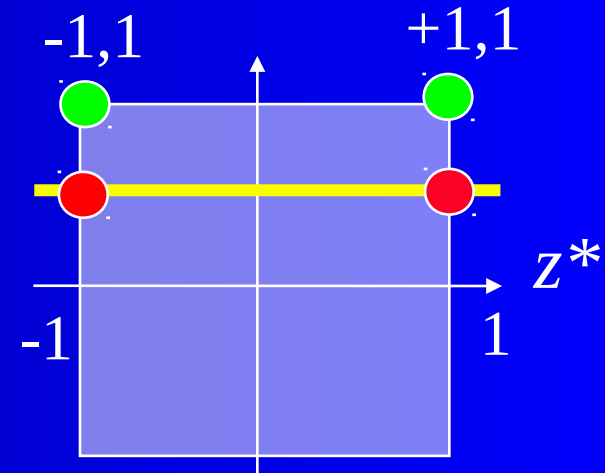
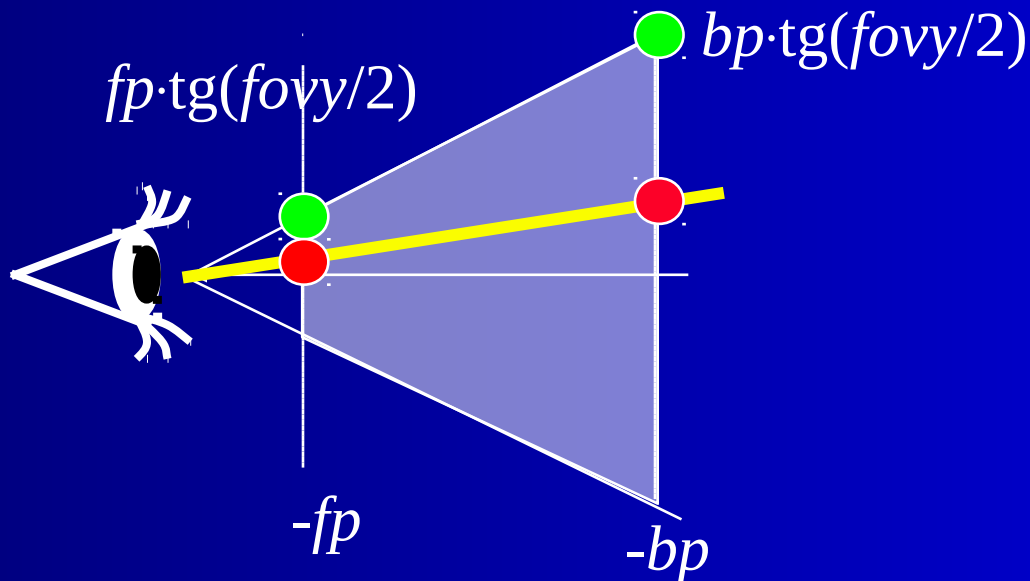
$$[x', y', z', 1] = [x, y, z, 1]$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\text{eye}_x & -\text{eye}_y & -\text{eye}_z & 1 \end{pmatrix} \cdot \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}$$

`gluLookAt(eye, lookat, vup)`

$$\begin{pmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Perspektív transzformáció



$$(x, y, -fp) \Rightarrow (x/(fp \cdot \text{tg}(fovx/2)), y/(fp \cdot \text{tg}(fovy/2)), -1)$$

$$(x, y, -bp) \Rightarrow (x/(bp \cdot \text{tg}(fovx/2)), y/(bp \cdot \text{tg}(fovy/2)), +1)$$

Perspektív transzformáció

gluPerspective(fov,asp,fp,bp)

$$\begin{pmatrix} 1/(\text{tg}(\text{fov}/2) \cdot \text{asp}) & 0 & 0 & 0 \\ 0 & 1/\text{tg}(\text{fov}/2) & 0 & 0 \\ 0 & 0 & -(\text{fp} + \text{bp})/(\text{bp} - \text{fp}) & -1 \\ 0 & 0 & -2\text{fp} \cdot \text{bp}/(\text{bp} - \text{fp}) & 0 \end{pmatrix}$$

$$[x, y, -\text{fp}, 1] \rightarrow [x/(\text{tg}(\text{fovx}/2)), y/(\text{tg}(\text{fovy}/2)), -\text{fp}, \text{fp}]$$

$$(x/(\text{fp} \cdot \text{tg}(\text{fovx}/2)), y/(\text{fp} \cdot \text{tg}(\text{fovy}/2)), -1)$$

$$[x, y, -\text{bp}, 1] \rightarrow [x/(\text{tg}(\text{fovx}/2)), y/(\text{tg}(\text{fovy}/2)), \text{bp}, \text{bp}]$$

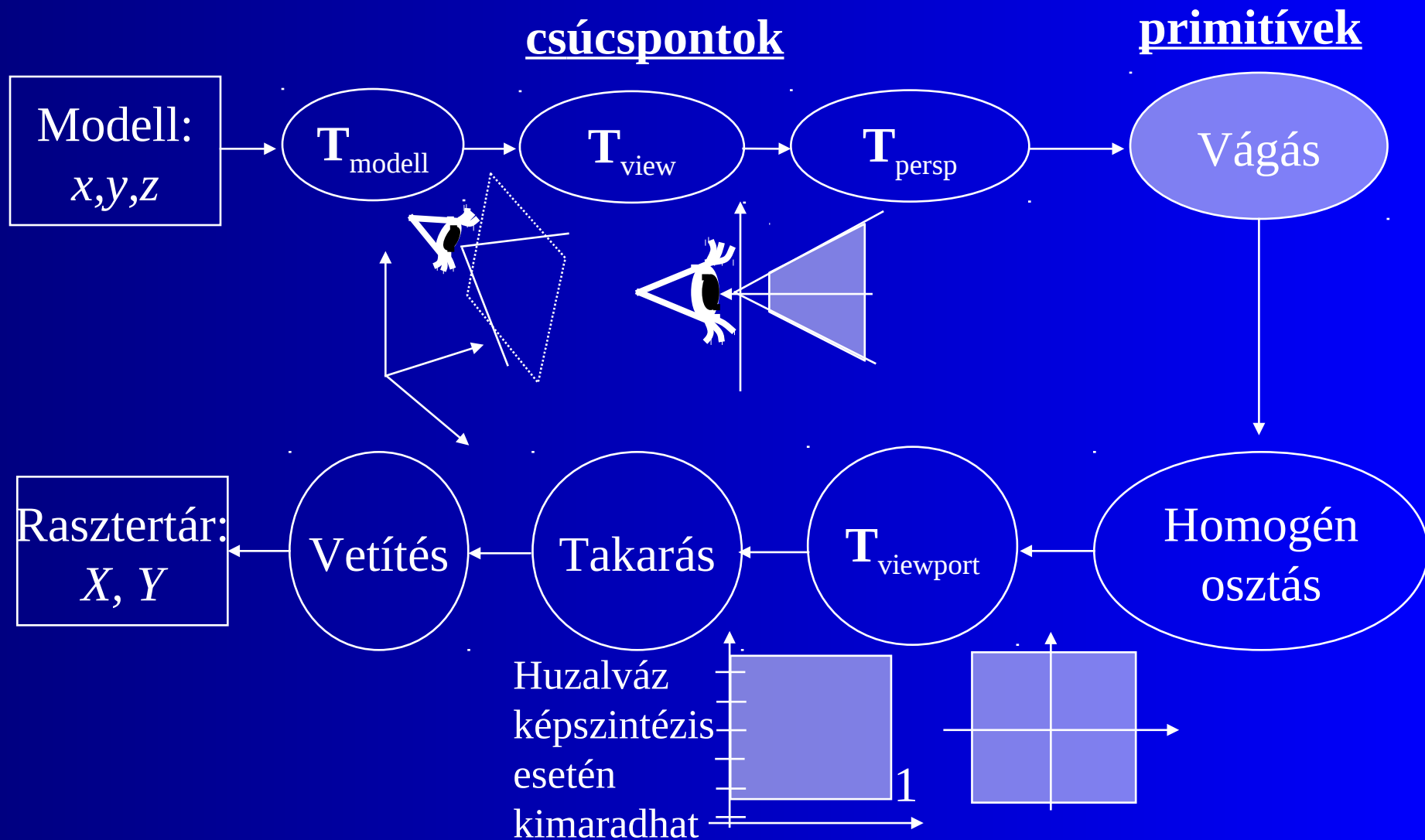
$$(x/(\text{bp} \cdot \text{tg}(\text{fovx}/2)), y/(\text{bp} \cdot \text{tg}(\text{fovy}/2)), +1)$$

$$[X_h, Y_h, Z_h, h] = [x_c, y_c, z_c, 1] \mathbf{T}_{\text{persp}}$$

$$h = -z_c$$

$$[X, Y, Z, 1] = [X_h/h, Y_h/h, Z_h/h, 1]$$

Képszintézis csővezeték

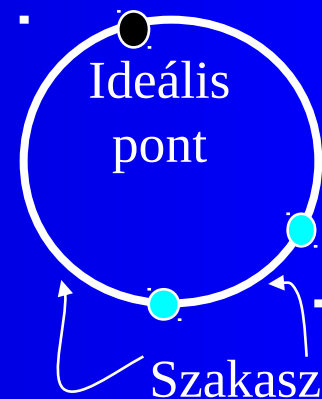
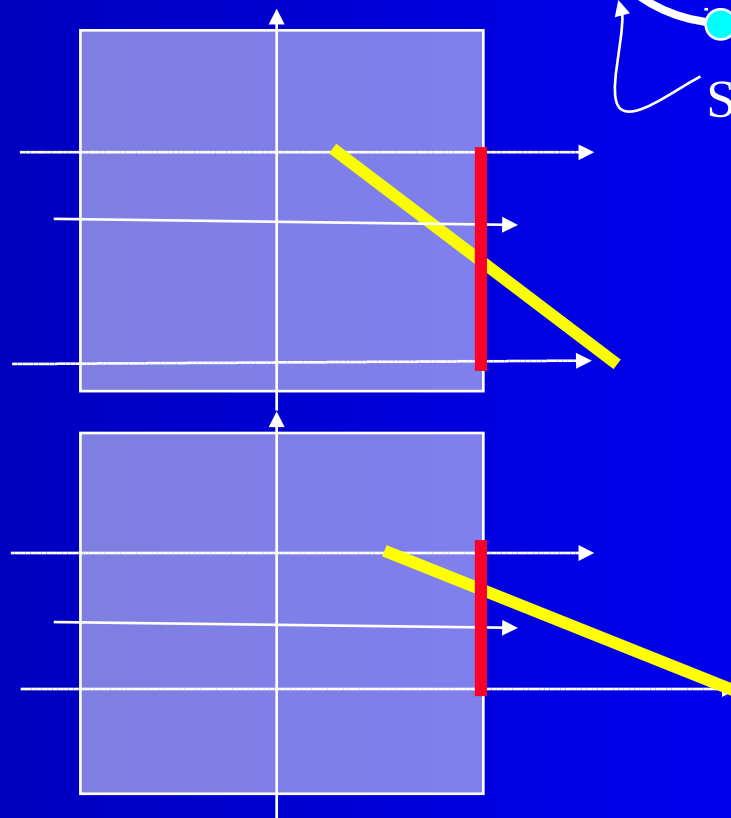
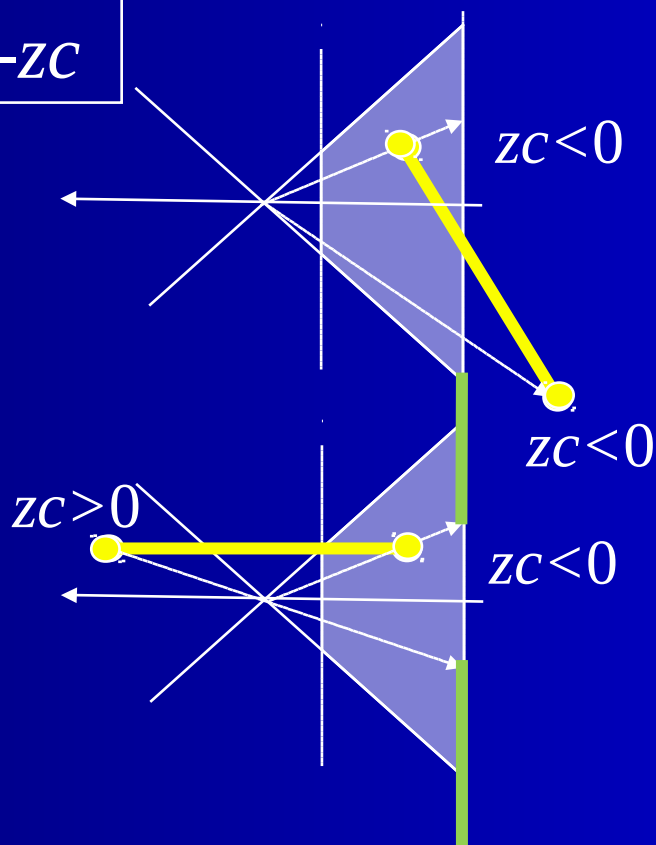


Vágni a homogén osztás előtt kell

$$[X(t), Y(t), Z(t), h(t)] = [X_1, Y_1, Z_1, h_1] \cdot t + [X_2, Y_2, Z_2, h_2] \cdot (1-t)$$

Homogén lineáris transzformáció konvex kombinációkat konvex kombinációkba visz át.

$$h = -zc$$



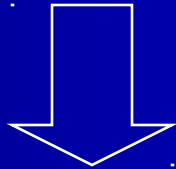
Vágás homogén koordinátákban

Cél: $-1 < X = X_h/h < 1$

$$-1 < Y = Y_h/h < 1$$

$$-1 < Z = Z_h/h < 1$$

Vegyük hozzá: $h > 0$ (mert $h = -zc$)



$$-h < X_h < h$$

$$-h < Y_h < h$$

$$-h < Z_h < h$$

Kívül ●

$$[3, 0, 0, 2]$$

$$h = 2 < X_h = 3$$

$$h = X_h$$

Belül ●

$$[2, 0, 0, 3]$$

$$h = 3 > X_h = 2$$

Szakasz/poligon vágás

$$-h < X_h < h$$

$$-h < Y_h < h$$

$$-h < Z_h < h$$

$h = X_h$

$[X_h^1, Y_h^1, Z_h^1, h^1]$

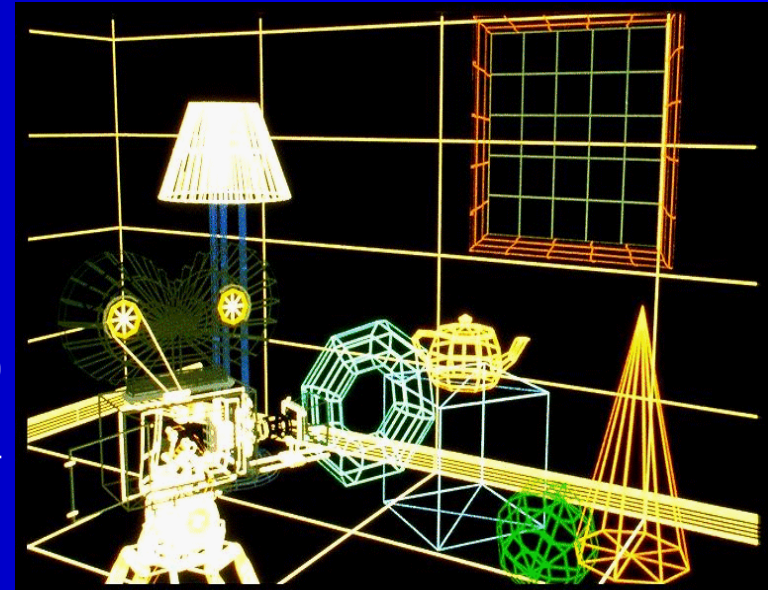
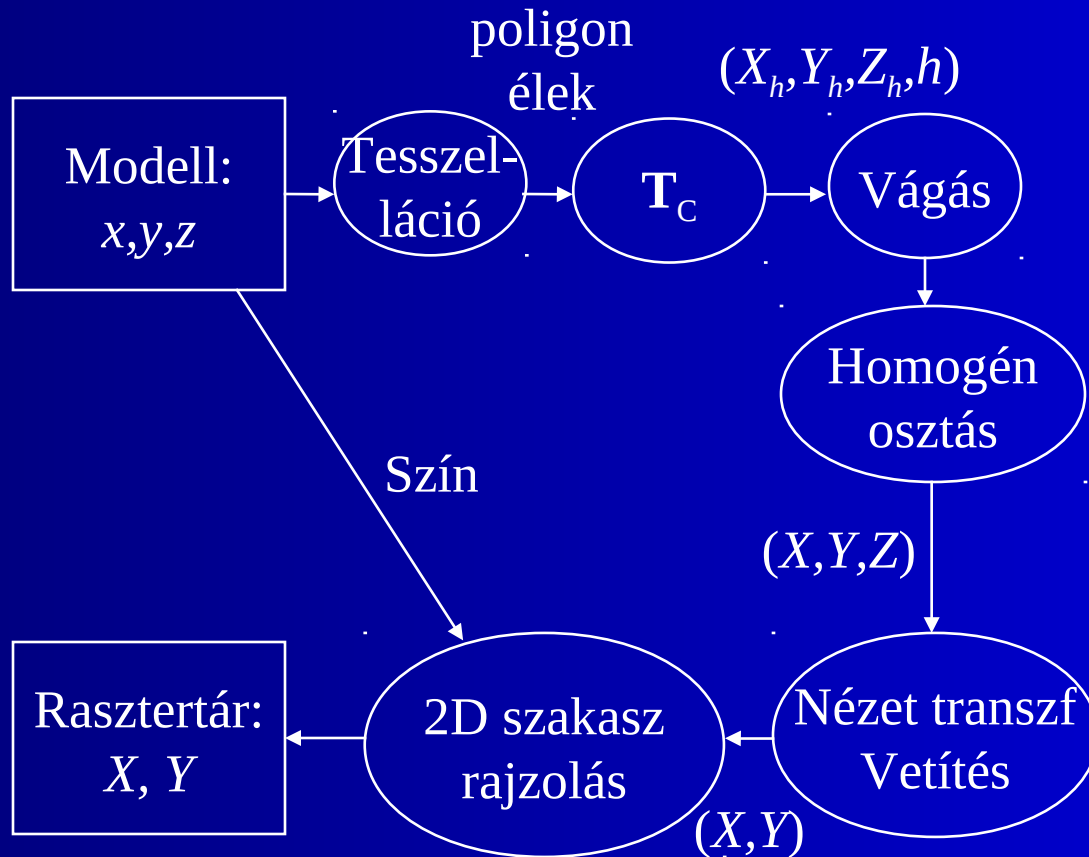
$[X_h^2, Y_h^2, Z_h^2, h^2]$

$$X_h = X_h^1 \cdot (1-t) + X_h^2 \cdot t$$
$$Y_h = Y_h^1 \cdot (1-t) + Y_h^2 \cdot t$$
$$Z_h = Z_h^1 \cdot (1-t) + Z_h^2 \cdot t$$
$$h = h^1 \cdot (1-t) + h^2 \cdot t$$

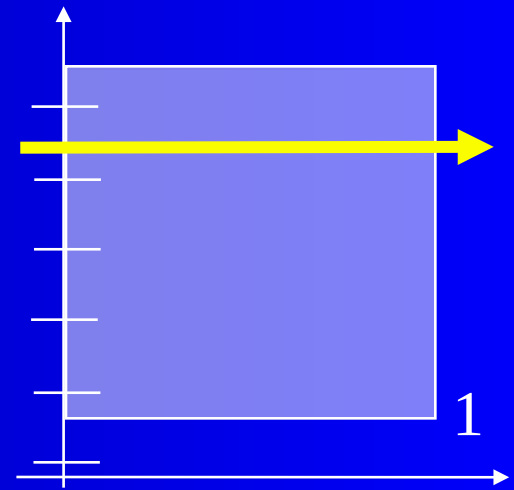
$$h = h^1 \cdot (1-t) + h^2 \cdot t =$$
$$= X_h = X_h^1 \cdot (1-t) + X_h^2 \cdot t$$

$$t = \dots$$

Huzalváz képszintézis



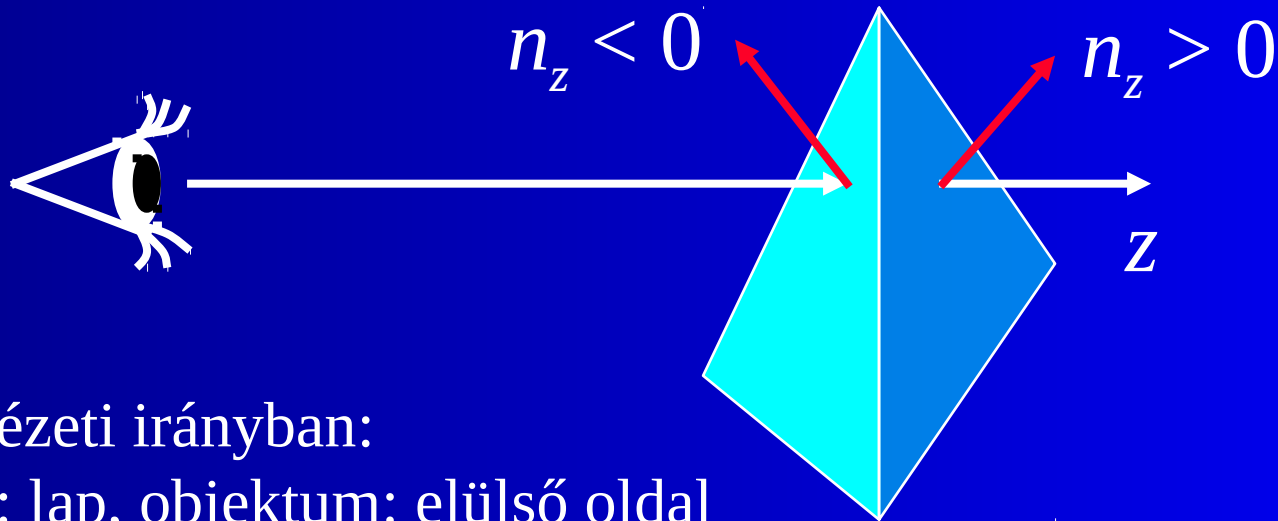
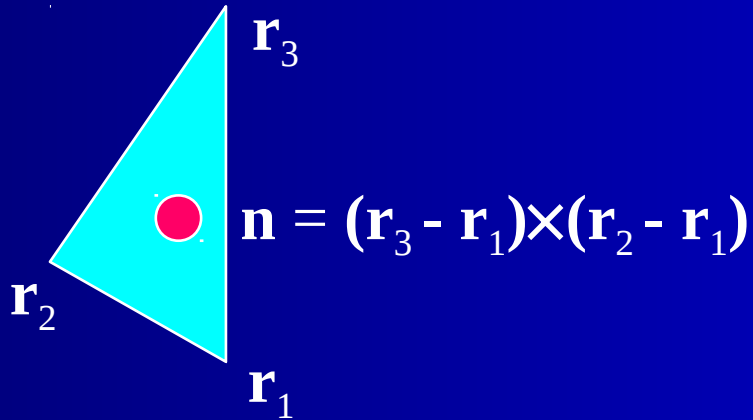
Tömör képszintézis



- Képernyő koordinátarendszerben
 - vetítősugarak a z tengellyel párhuzamosak!
- Objektumtér algoritmusok (folytonos):
 - láthatóság számítás nem függ a felbontástól
- Képtér algoritmusok (diszkrét):
 - mi látszik egy pixelben
 - Sugárkövetés ilyen volt!

Hátsólab eldobás:

back-face culling



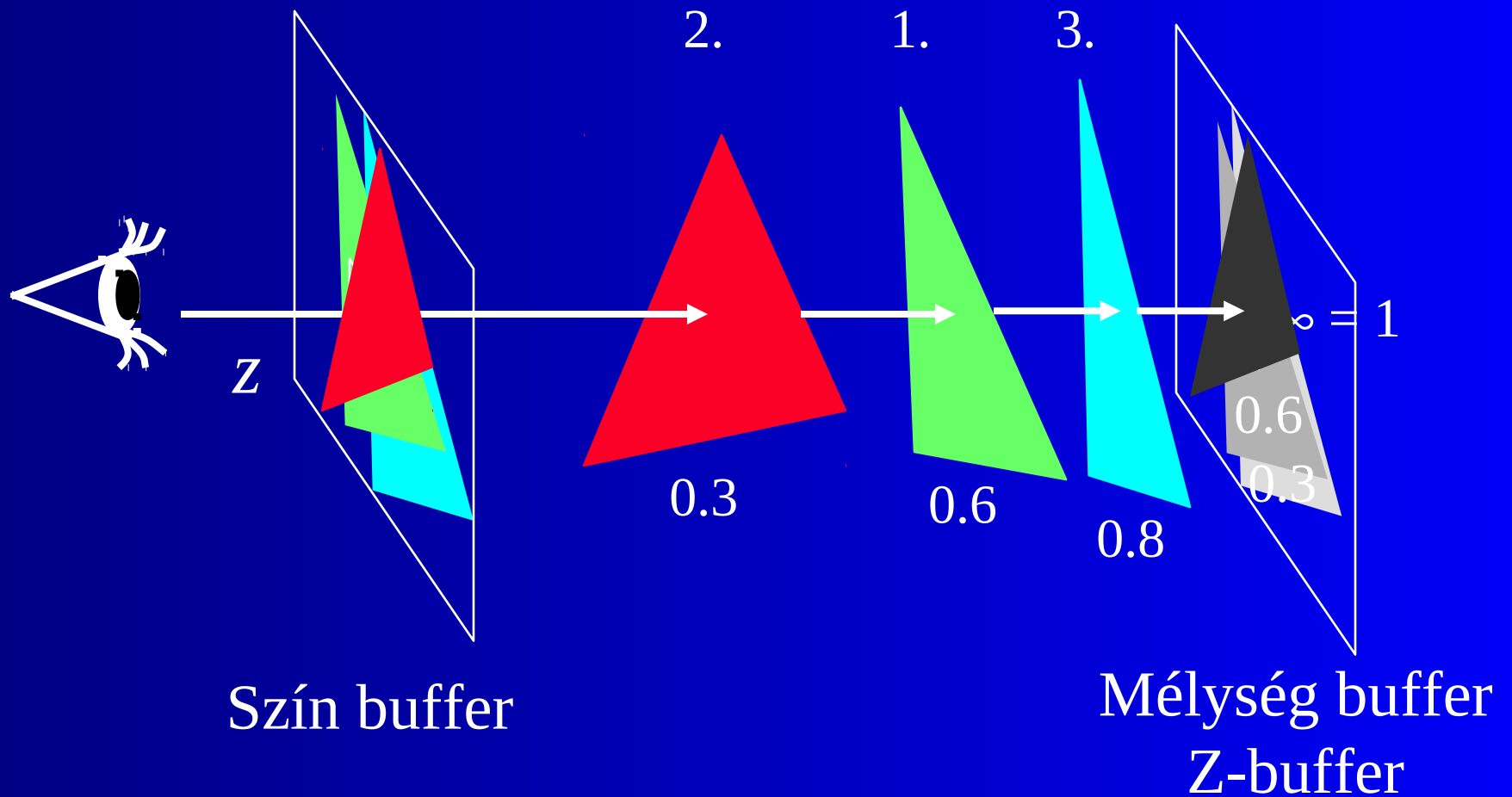
Lapok a nézeti irányban:

- Kívülről: lap, objektum: elülső oldal
- Belülről: objektum, lap: hátsó oldal

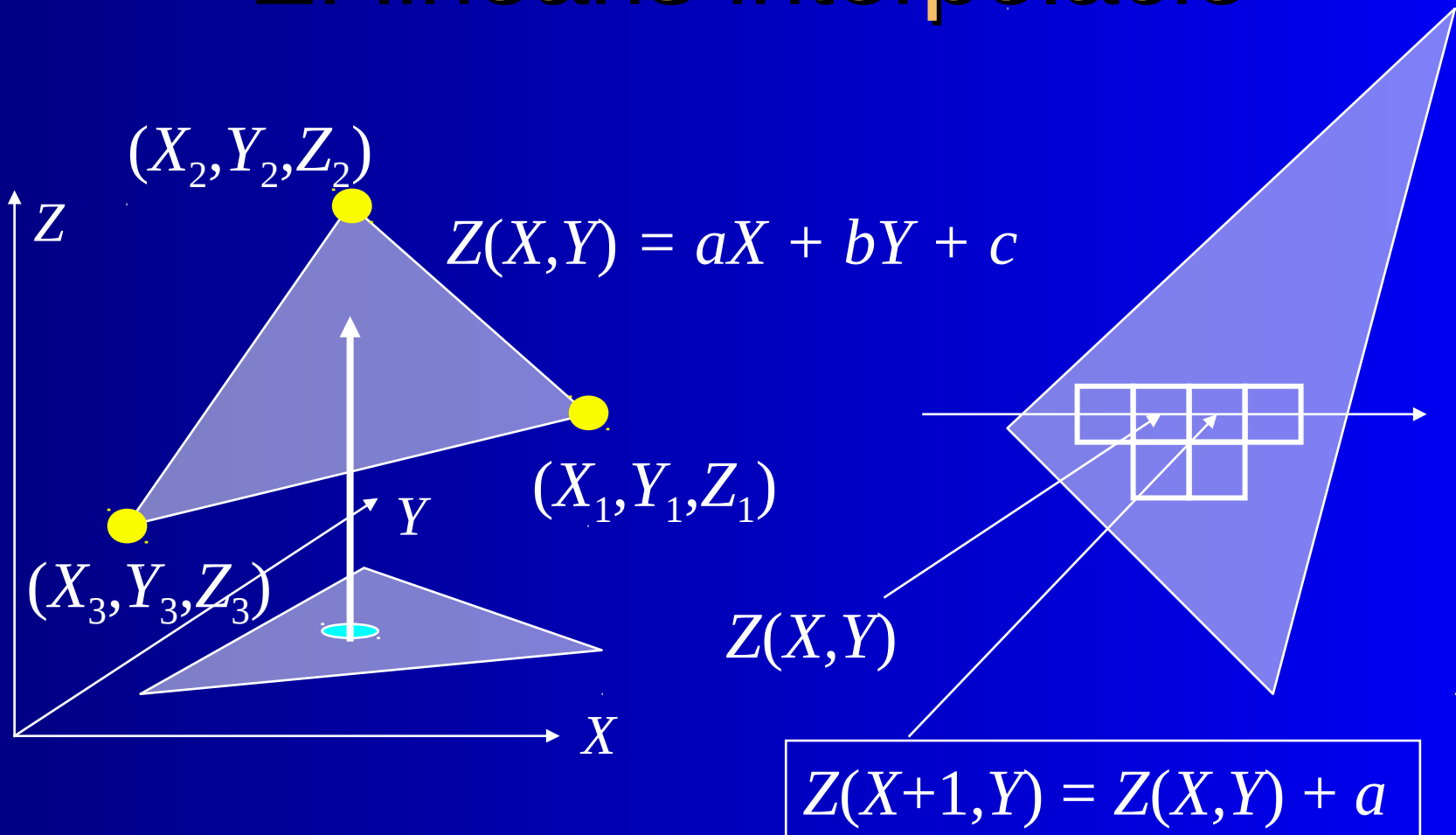
Feltételezés:

Ha kívülről, akkor csúcsok óramutatóval megegyező körüljárásúak

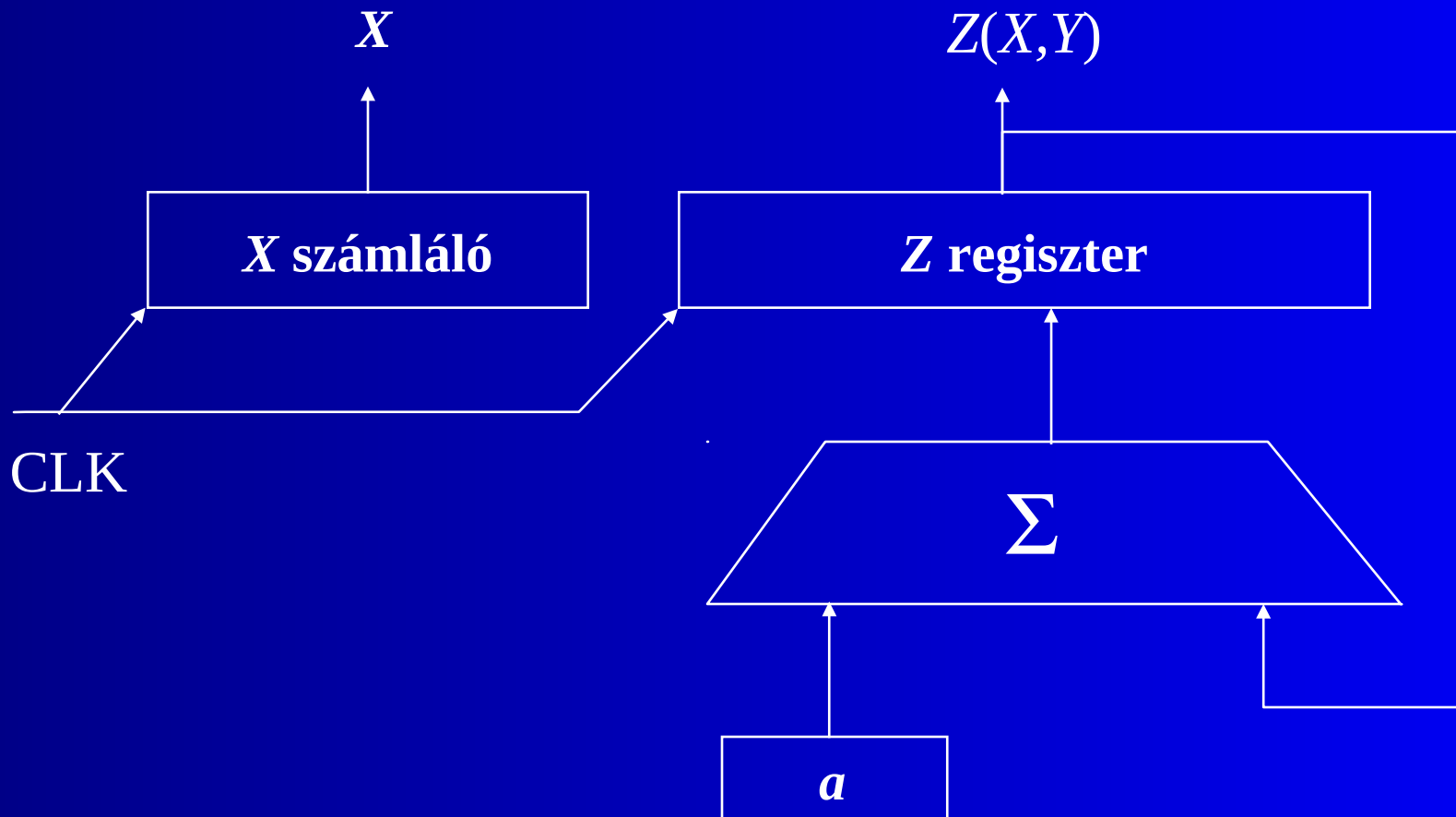
Z-buffer algoritmus



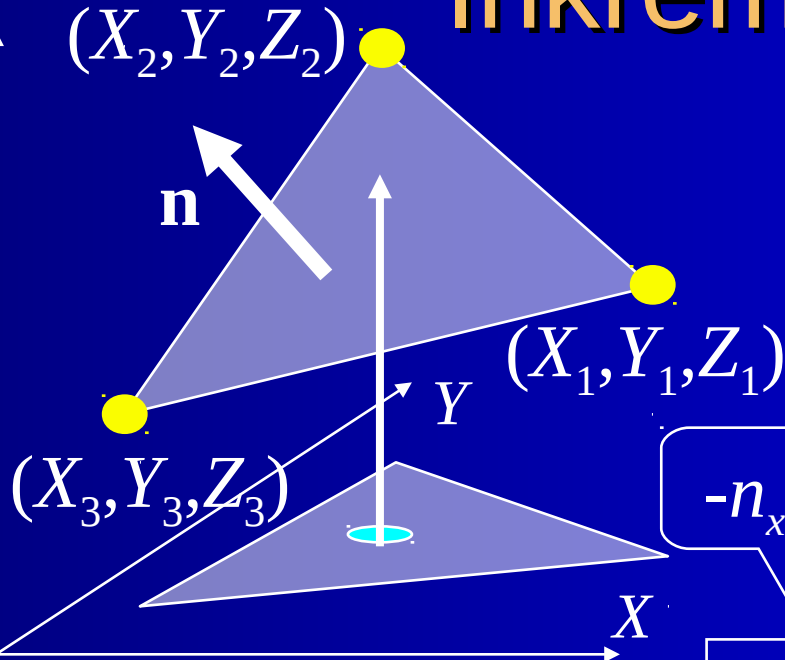
Z: lineáris interpoláció



Z-interpolációs hardver



Inkremens (triangle setup)



$$Z(X,Y) = aX + bY + c$$

$$n_x X + n_y Y + n_z Z + d = 0$$

$$Z_1 = aX_1 + bY_1 + c$$

$$Z_2 = aX_2 + bY_2 + c$$

$$Z_3 = aX_3 + bY_3 + c$$

$$Z_3 - Z_1 = a(X_3 - X_1) + b(Y_3 - Y_1)$$

$$Z_2 - Z_1 = a(X_2 - X_1) + b(Y_2 - Y_1)$$

$$a = \frac{(Z_3 - Z_1)(Y_2 - Y_1) - (Y_3 - Y_1)(Z_2 - Z_1)}{(X_3 - X_1)(Y_2 - Y_1) - (Y_3 - Y_1)(X_2 - X_1)}$$

$$\mathbf{n} = (\mathbf{r}_3 - \mathbf{r}_1) \times (\mathbf{r}_2 - \mathbf{r}_1) = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ X_3 - X_1 & Y_3 - Y_1 & Z_3 - Z_1 \\ X_2 - X_1 & Y_2 - Y_1 & Z_2 - Z_1 \end{vmatrix}$$

n_z

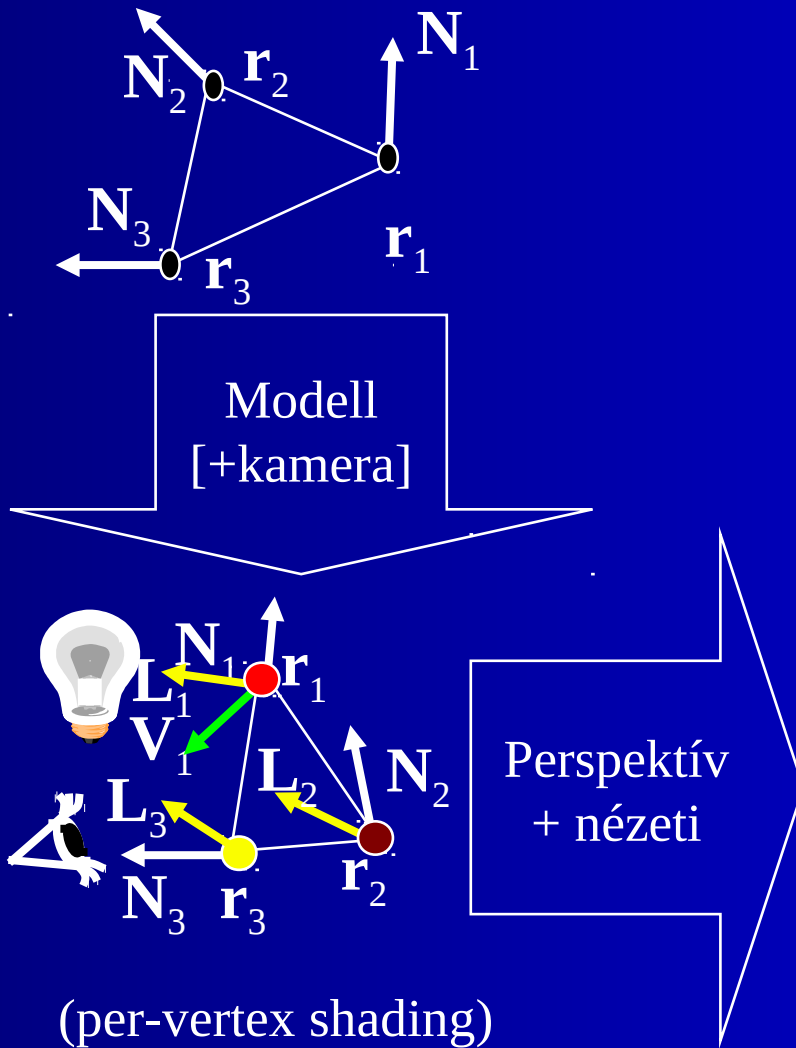
Árnyalás

$$L(\mathbf{V}) \approx \sum_l L_l(\mathbf{L}_l) * f_r(\mathbf{L}_l, \mathbf{N}, \mathbf{V}) \cdot \cos \theta'$$

- Koherencia: ne mindent pixelenként
- Háromszögenként:
 - csúcspontonként 1-szer,
belül az L „szín” lineáris interpolációja:
Gouraud árnyalás, (per-vertex shading)
 - pixelenként, a Normál ($\mathbf{V}$ View, $\mathbf{L}$ Light)
vektort interpoláljuk: Phong árnyalás,
(per-pixel shading)



Gouraud (per-vertex) árnyalás



$$R(X,Y) = aX + bY + c$$

$$G(X,Y) = \dots$$

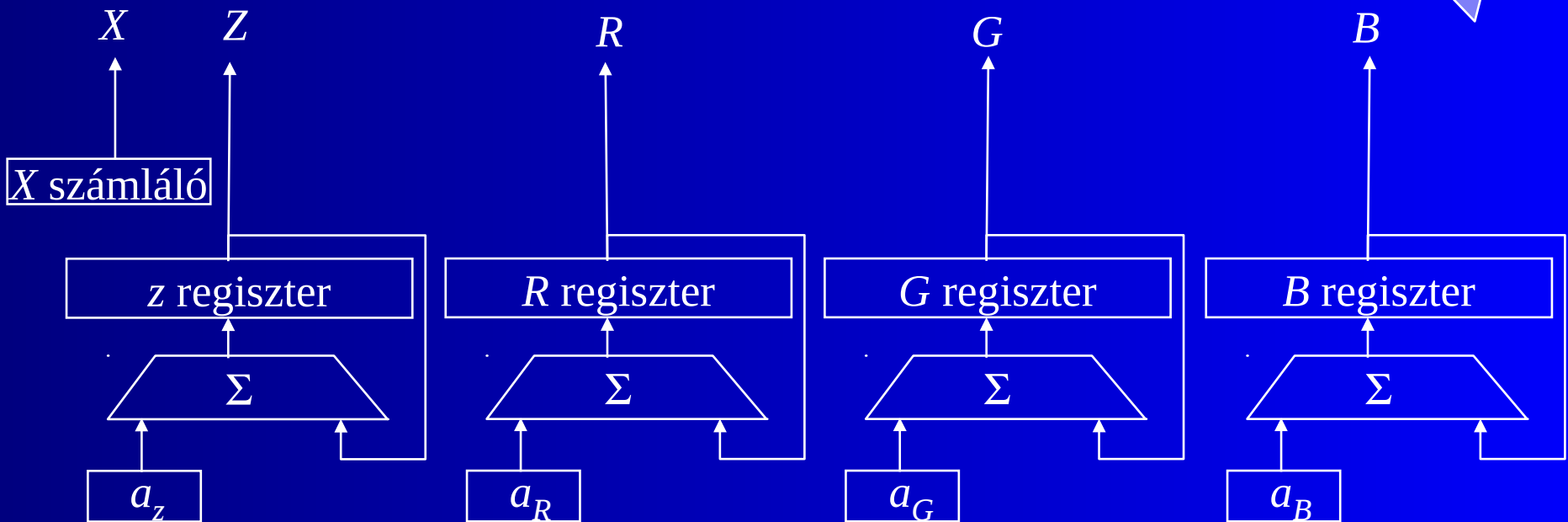
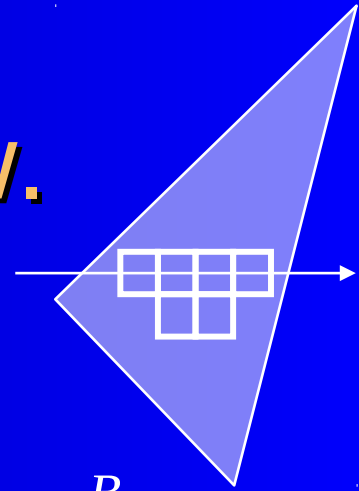
$$B(X,Y) = \dots$$

$$R(X,Y)$$

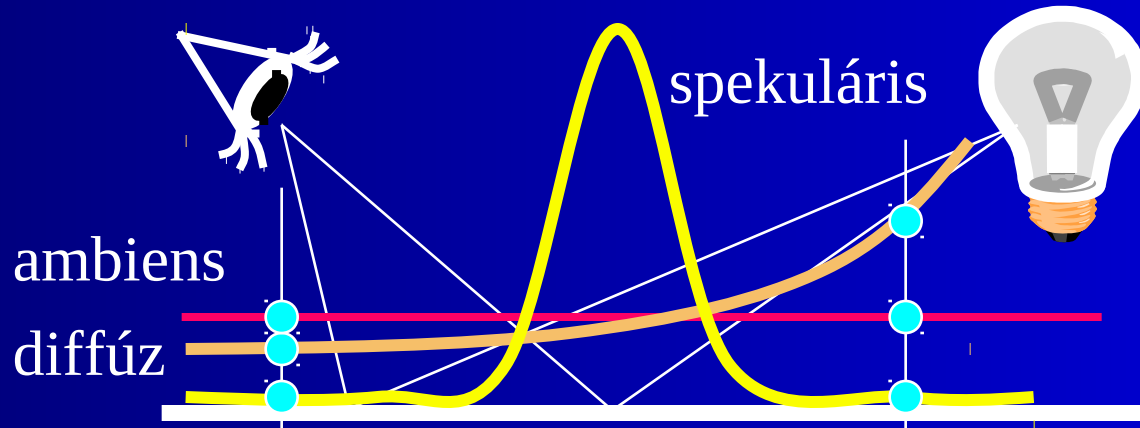
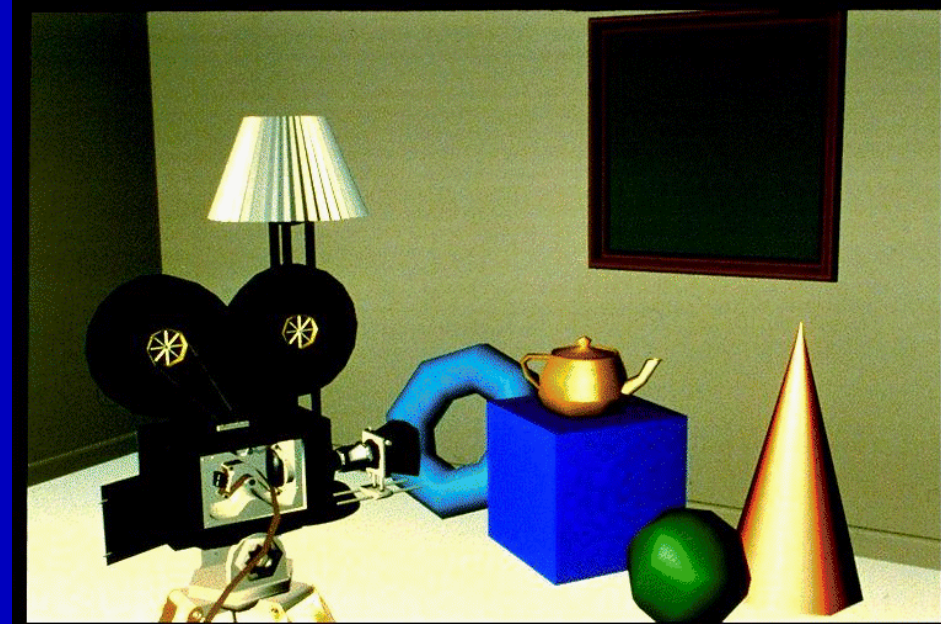
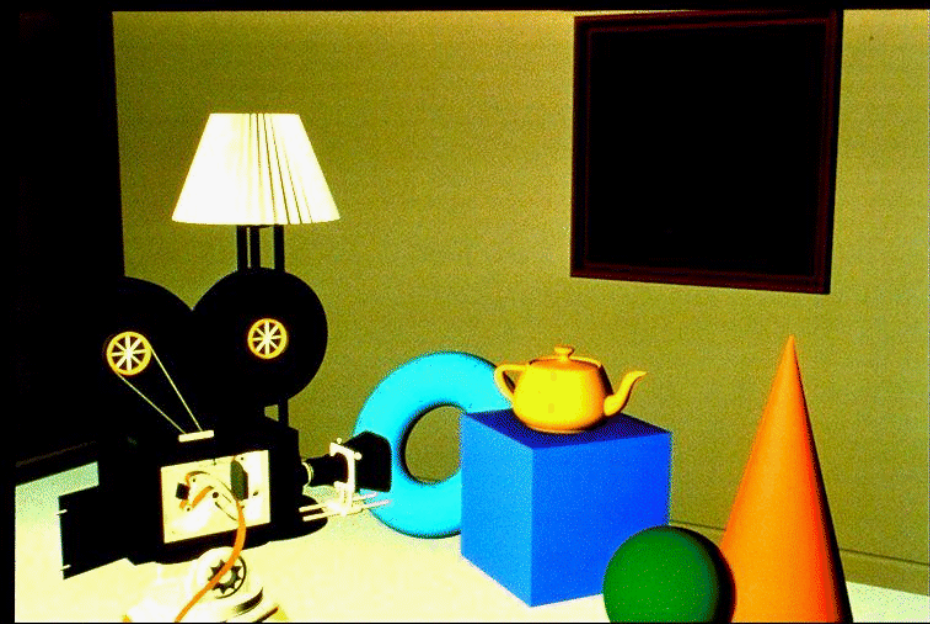
$$R(X+1,Y) = R(X,Y) + a$$

Képernyő

Z-buffer + Gouraud árnyaló hw.



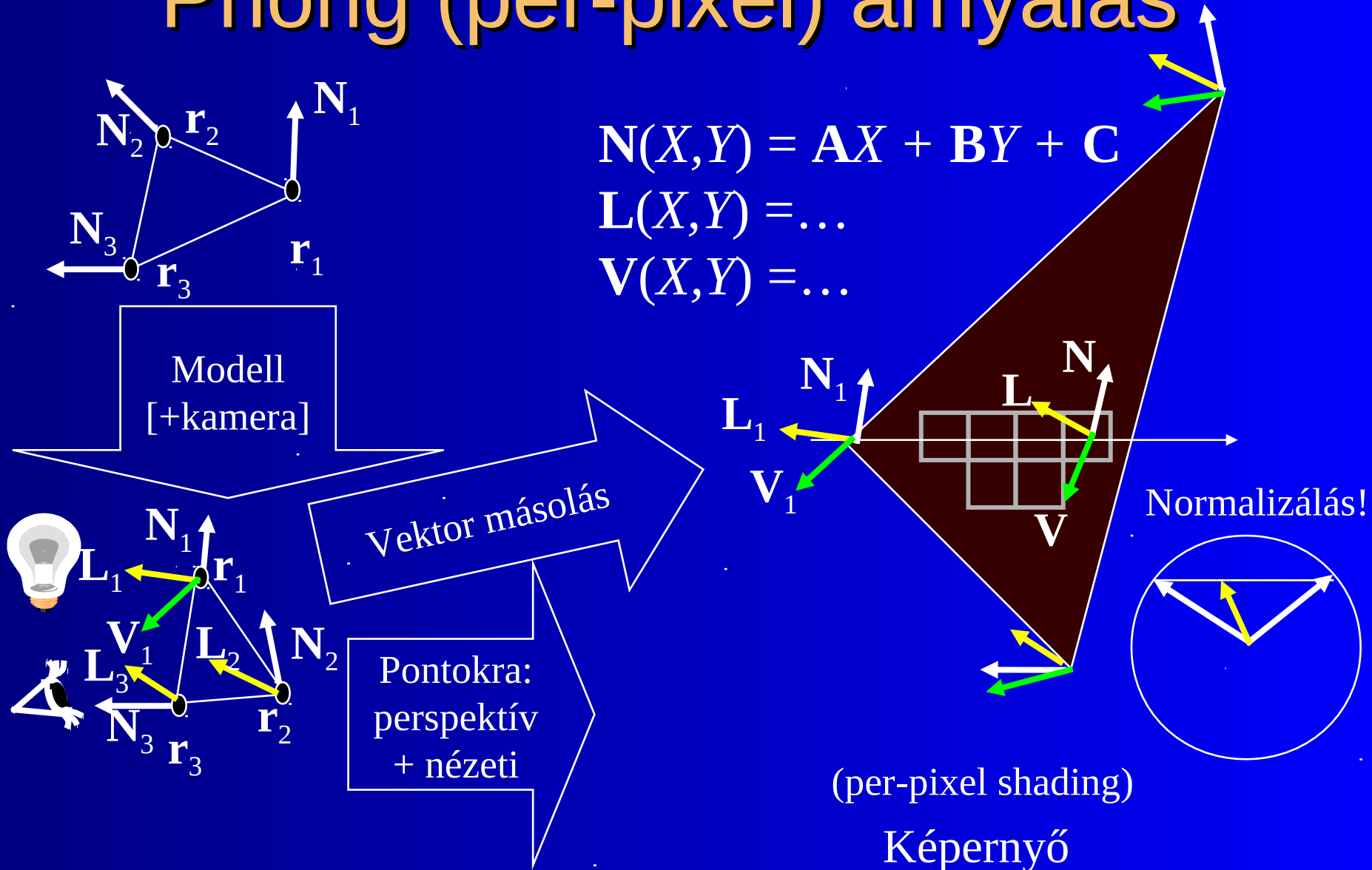
Gouraud árnyalás hasfájásai



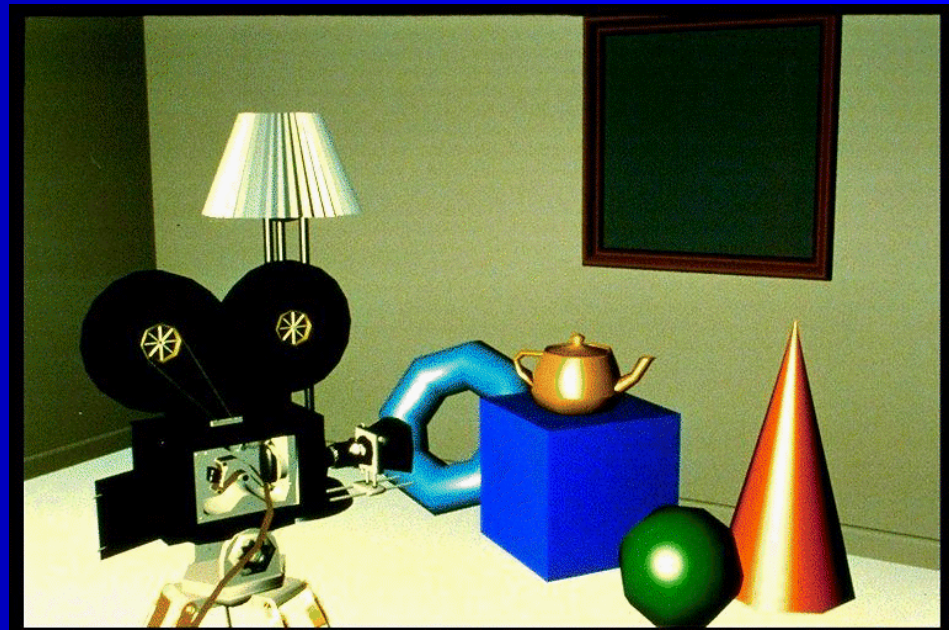
További bajok:

- anyagtulajdonság konstans
 - árnyék nincs
- különben a színt nem lehet interpolálni

Phong (per-pixel) árnyalás

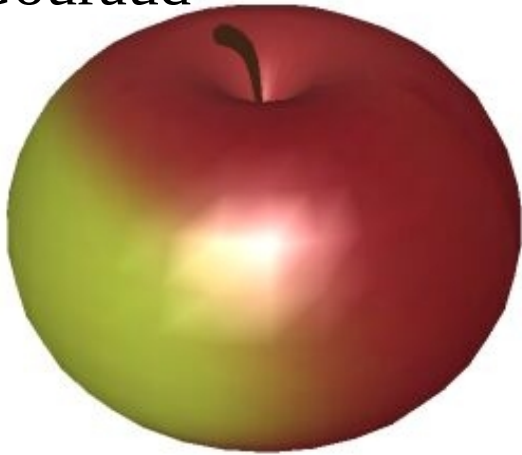


Phong árnyalás

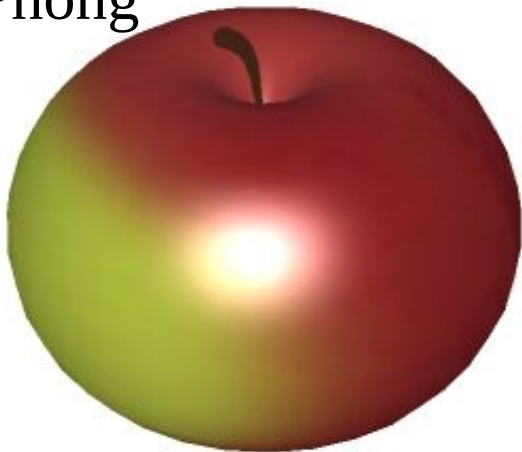


Gouraud versus Phong

Gouraud



Phong



Gouraud



Phong



Textúra leképzés: anyagjellemzők változnak a felületen



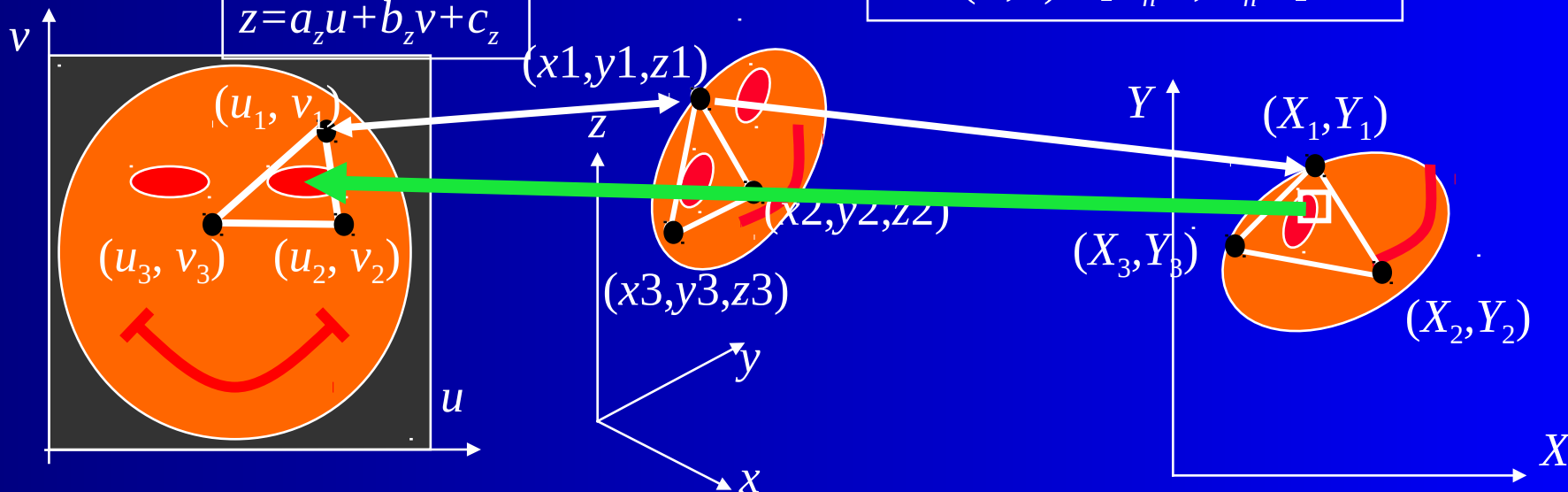
Inkrementális képszintézis

Paraméterezés

$$\begin{aligned}x &= a_x u + b_x v + c_x \\y &= a_y u + b_y v + c_y \\z &= a_z u + b_z v + c_z\end{aligned}$$

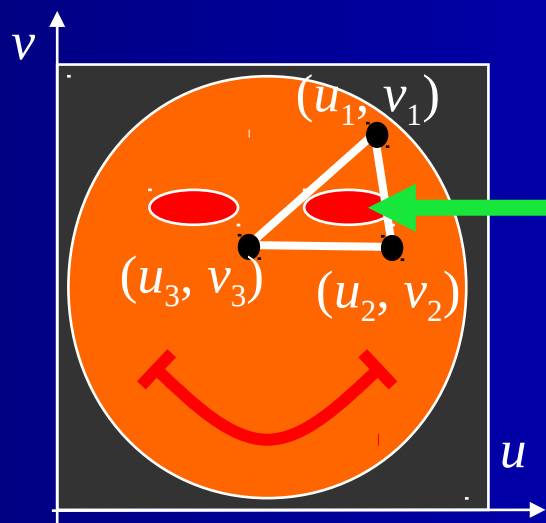
Model-camera-perspective-viewport
transzformáció

$$\begin{aligned}[X_h, Y_h, Z_h, h] &= [x, y, z, 1] \mathbf{T}_C \\(X, Y) &= [X_h/h, Y_h/h]\end{aligned}$$

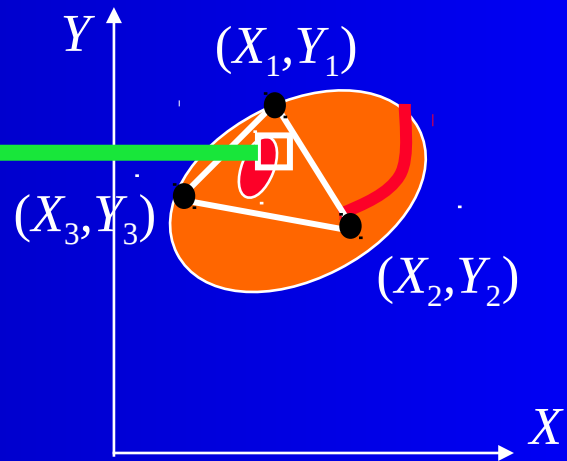


$$\begin{aligned}[X \cdot h, Y \cdot h, h] &= [u, v, 1] \cdot \mathbf{T}, \text{ ahol } h = -zc \\[u/h, v/h, 1/h] &= [X, Y, 1] \cdot \mathbf{T}^{-1}\end{aligned}$$

Lineáris interpoláció



$$\begin{aligned}u &= a_u X + b_u Y + c_u \\v &= a_v X + b_v Y + c_v\end{aligned}$$

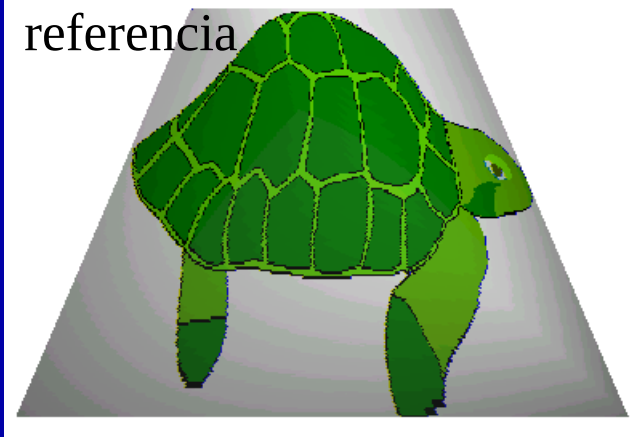


Interpolációs kényszerek:

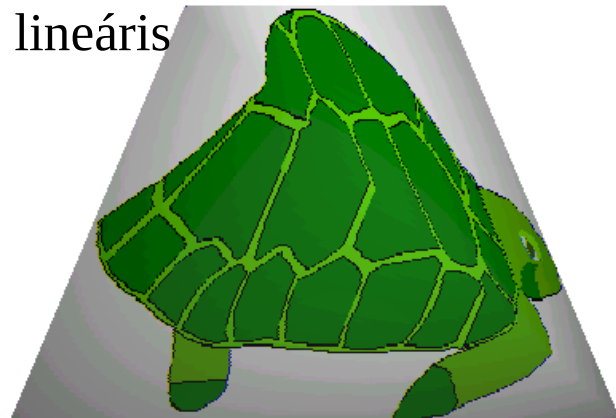
$$\begin{aligned}u_1 &= a_u X_1 + b_u Y_1 + c_u \\u_2 &= a_u X_2 + b_u Y_2 + c_u \\u_3 &= a_u X_3 + b_u Y_3 + c_u\end{aligned}$$

$$\begin{aligned}v_1 &= a_v X_1 + b_v Y_1 + c_v \\v_2 &= a_v X_2 + b_v Y_2 + c_v \\v_3 &= a_v X_3 + b_v Y_3 + c_v\end{aligned}$$

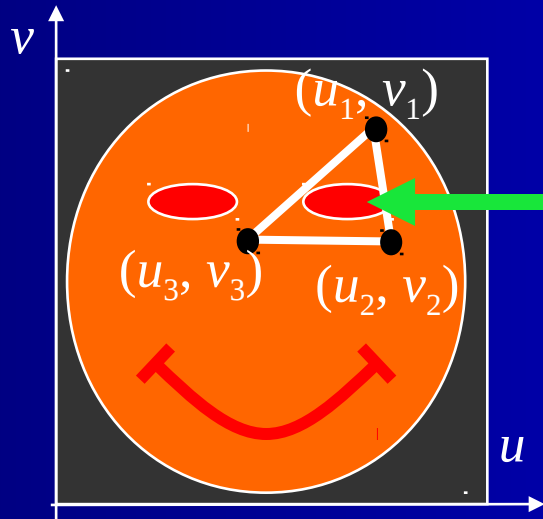
referencia



lineáris



Perspektíva helyes textúrázás

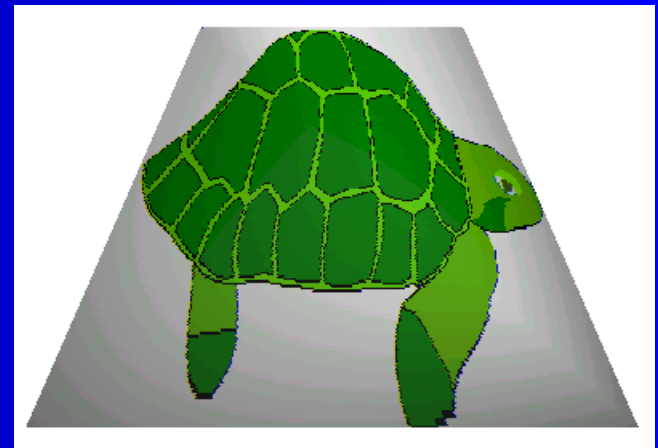
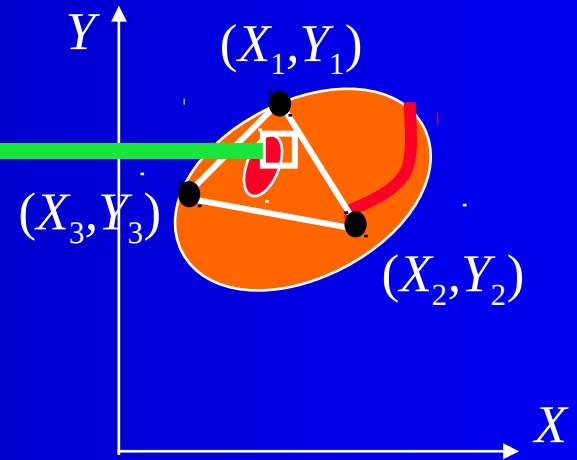


$$u = \frac{A_u X + B_u Y + C_u}{A_h X + B_h Y + C_h}$$

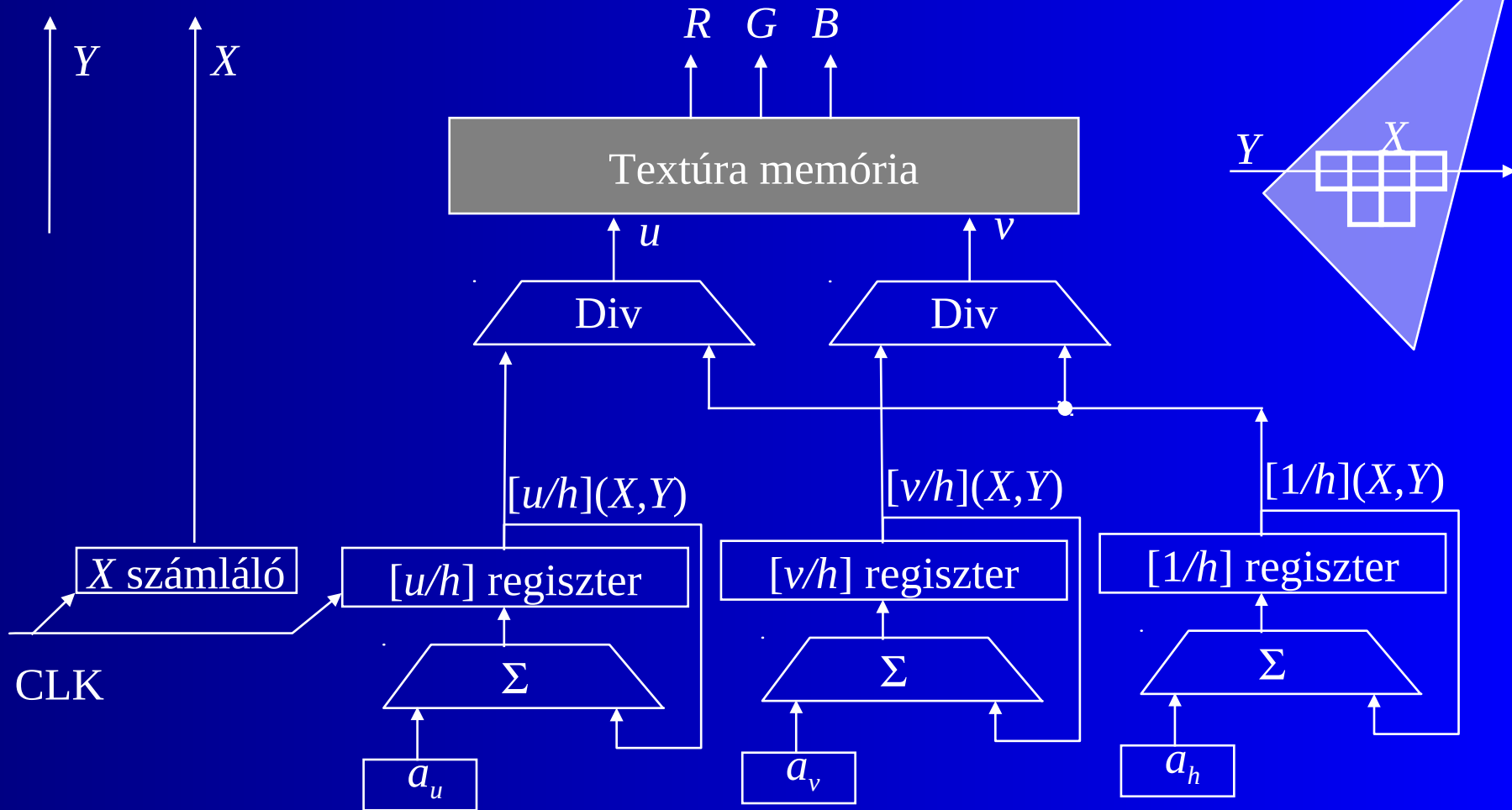
$$v = \frac{A_v X + B_v Y + C_v}{A_h X + B_h Y + C_h}$$

$$\begin{aligned} u/h &= A_u X + B_u Y + C_u \\ v/h &= A_v X + B_v Y + C_v \\ 1/h &= A_h X + B_h Y + C_h \end{aligned}$$

$$h = -zc$$



Perspektíva helyes interpolációs hw.



Textúra leképzés

Szirmay-Kalos László

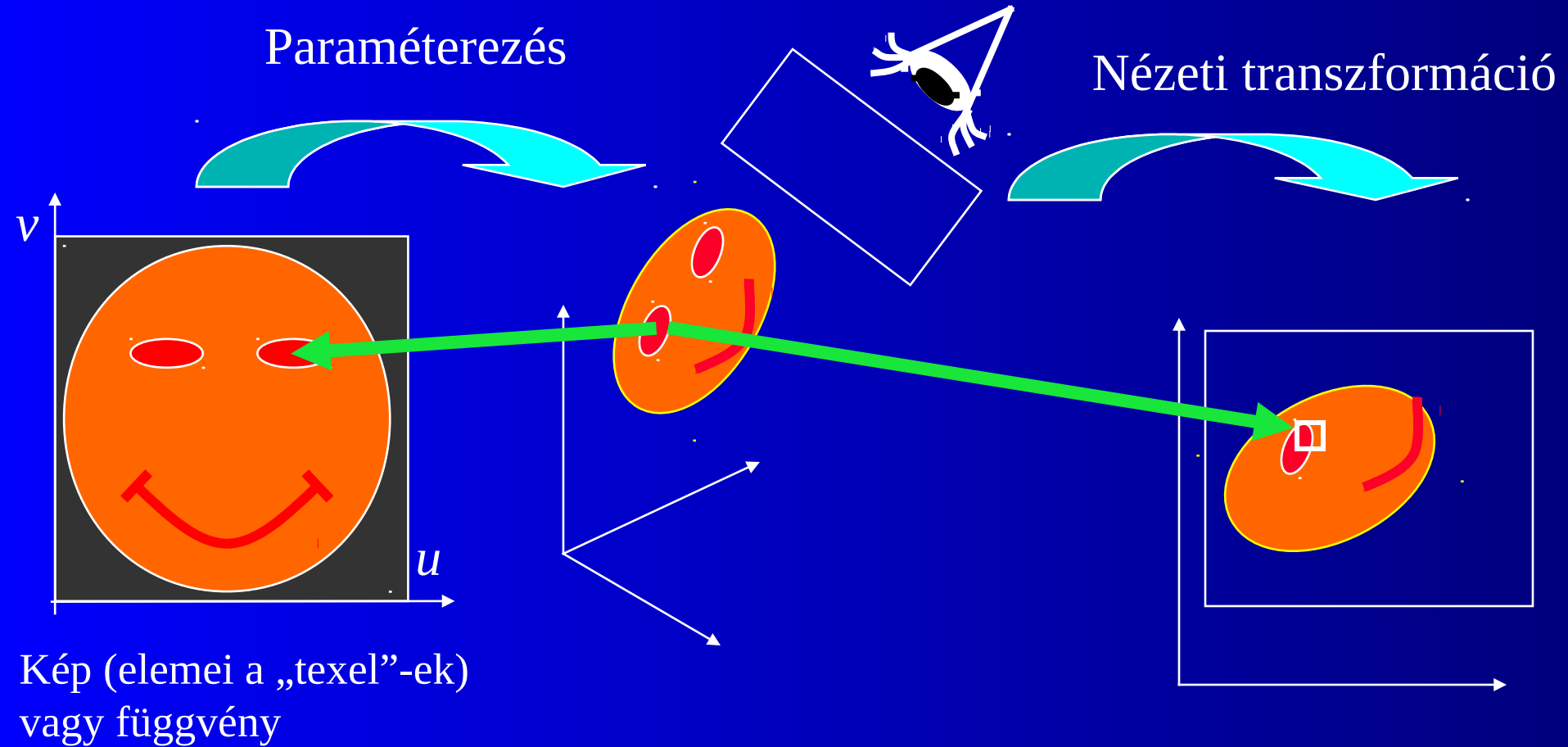
Textúra leképezés: anyagjellemzők változnak a felületen



(2D) Textúra leképzés

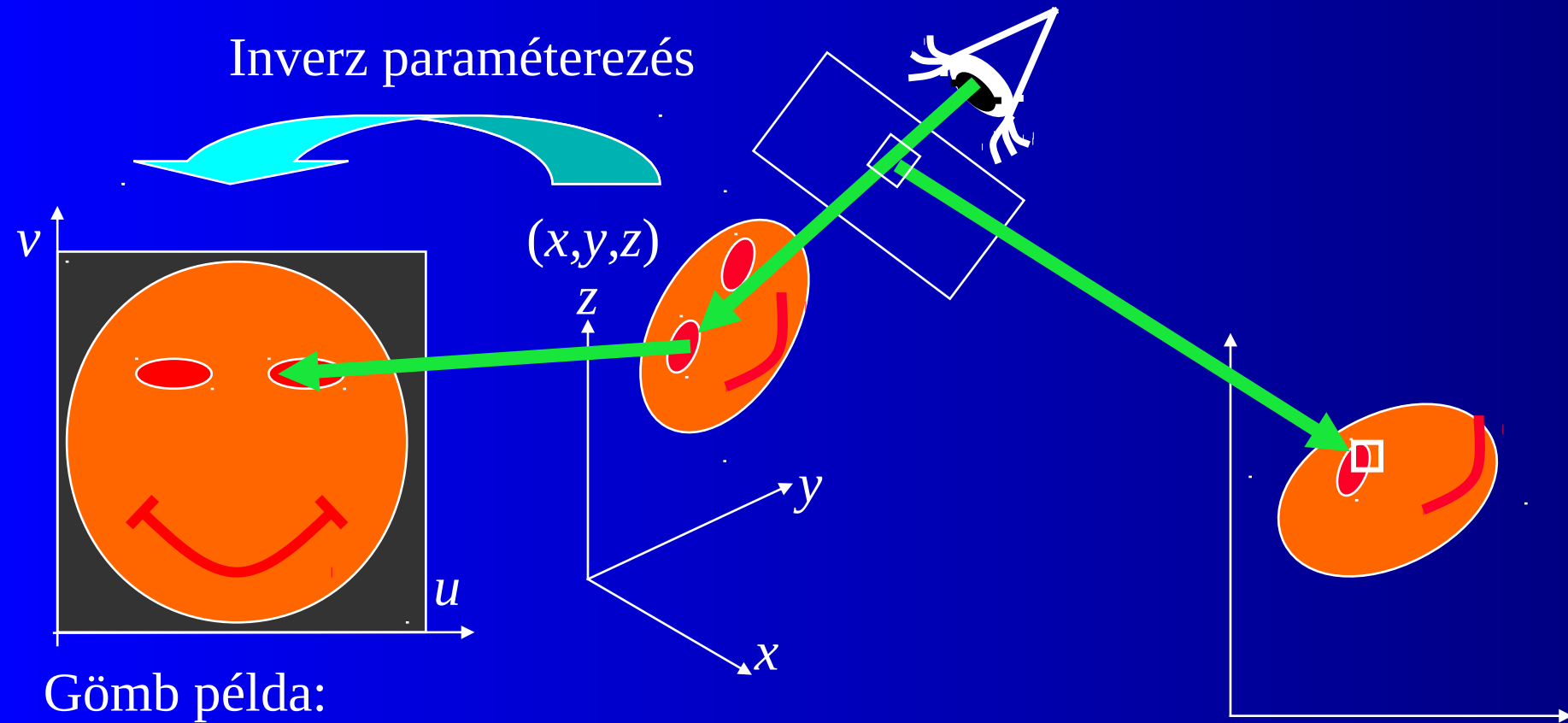
Paraméterezés

Nézeti transzformáció



Textúra leképezés sugárkövetésnél

Inverz paraméterezés



Gömb példa:

$$x = x_0 + r \cos 2\pi u \sin \pi v$$

$$y = y_0 + r \sin 2\pi u \sin \pi v$$

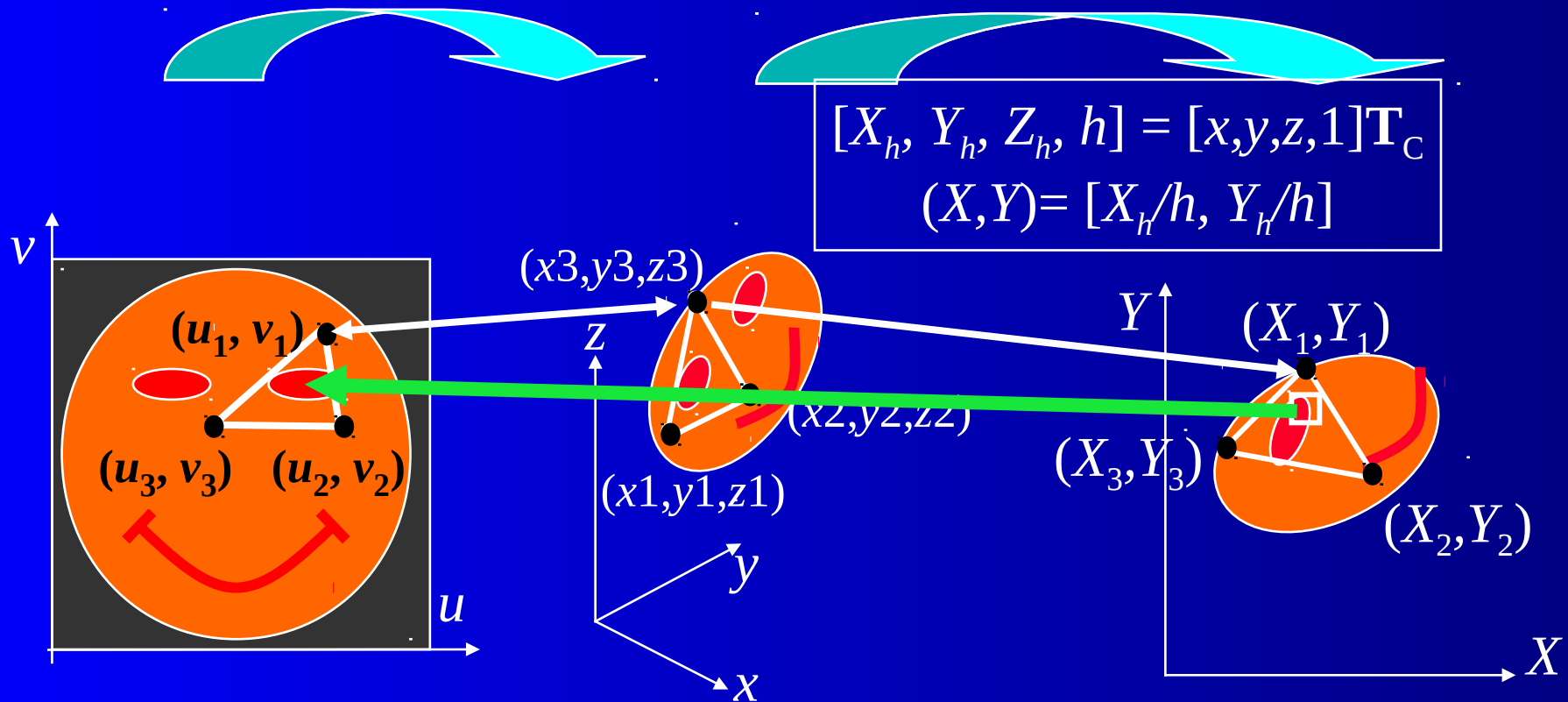
$$z = z_0 + r \cos \pi v \quad u, v \in [0, 1]$$

$$\left. \begin{array}{l} x = x_0 + r \cos 2\pi u \sin \pi v \\ y = y_0 + r \sin 2\pi u \sin \pi v \\ z = z_0 + r \cos \pi v \end{array} \right\} \begin{array}{l} u = 1/2\pi (\text{atan2}((y-y_0), (x-x_0)) + \pi) \\ v = 1/\pi \arccos((z-z_0)/r) \end{array}$$

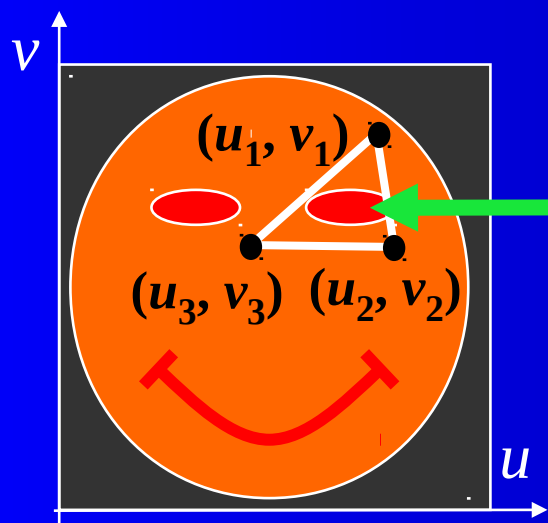
Inkrementális képszintézis

Paraméterezés

Modell-világ-képernyő transzf.

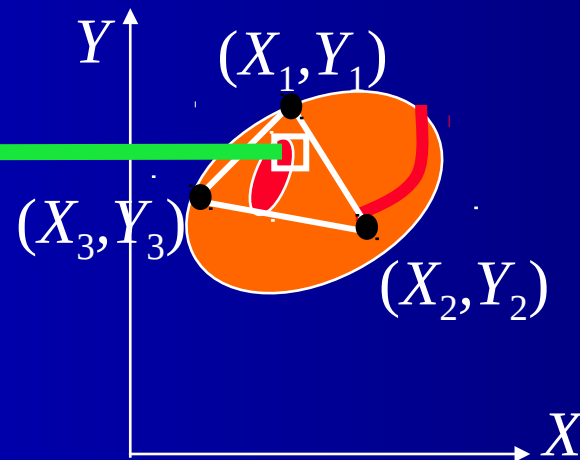
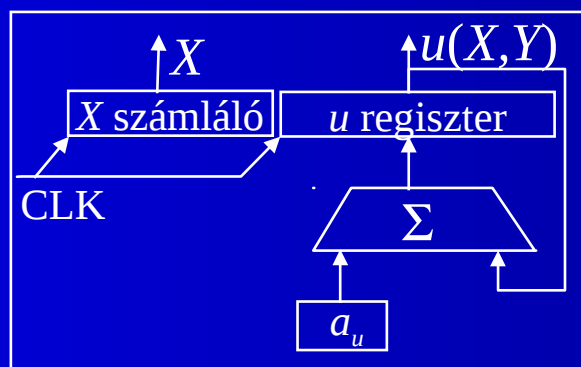


Lineáris interpoláció



$$u = a_u X + b_u Y + c_u$$

$$v = a_v X + b_v Y + c_v$$



Interpolációs feltétel:

$$u_1 = a_u X_1 + b_u Y_1 + c_u$$

$$u_2 = a_u X_2 + b_u Y_2 + c_u$$

$$u_3 = a_u X_3 + b_u Y_3 + c_u$$

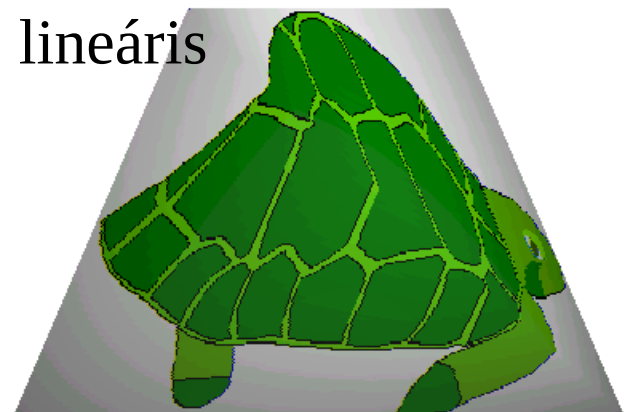
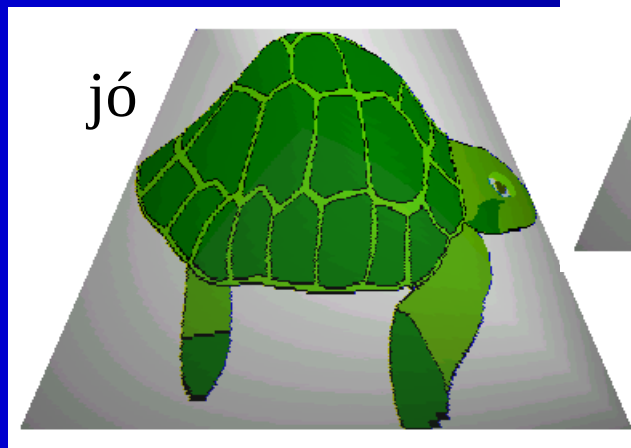
$$v_1 = a_v X_1 + b_v Y_1 + c_v$$

$$v_2 = a_v X_2 + b_v Y_2 + c_v$$

$$v_3 = a_v X_3 + b_v Y_3 + c_v$$

lineáris

jó



Perspektíva helyes textúrázás

Paraméterezés

$$x = a_x u + b_x v + c_x$$

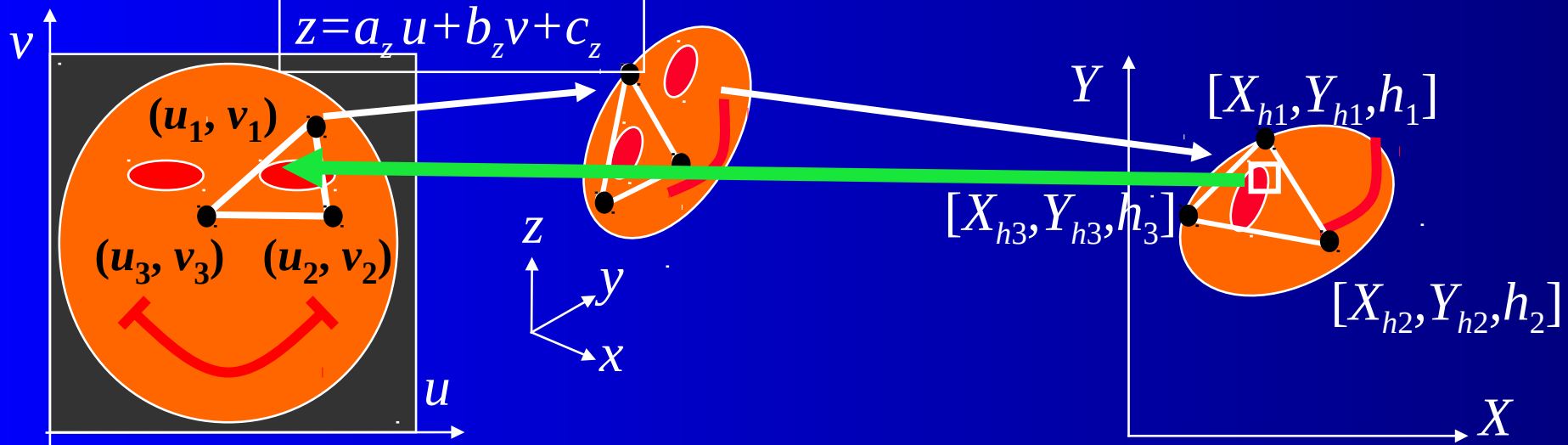
$$y = a_y u + b_y v + c_y$$

$$z = a_z u + b_z v + c_z$$

Modell-világ-képernyő transzf.

$$[X_h, Y_h, Z_h, h] = [x, y, z, 1] \mathbf{T}_C$$

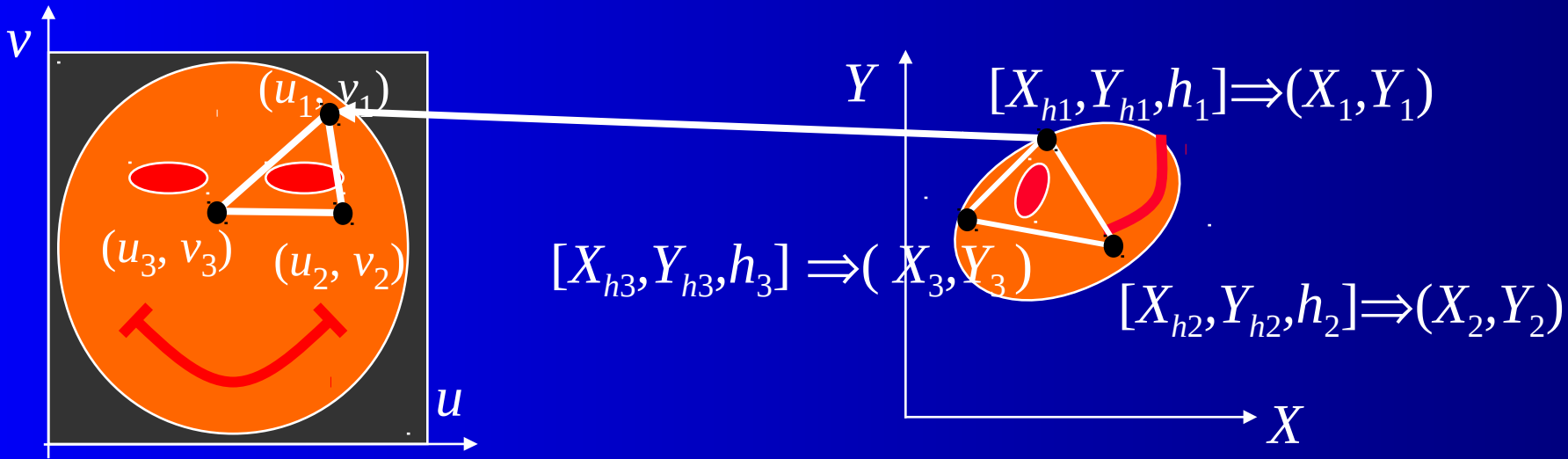
$$(X, Y) = [X_h/h, Y_h/h]$$



$$[u, v, 1] \mathbf{T} = [X_h, Y_h, h],$$

$$[u/h, v/h, 1/h] = [X, Y, 1] \mathbf{T}^{-1}$$

T⁻¹ mátrixelemek



$$\begin{aligned} u_1/h_1 &= a_u X_1 + b_u Y_1 + c_u \\ u_2/h_2 &= a_u X_2 + b_u Y_2 + c_u \\ u_3/h_3 &= a_u X_3 + b_u Y_3 + c_u \end{aligned}$$

$$a_u, b_u, c_u$$

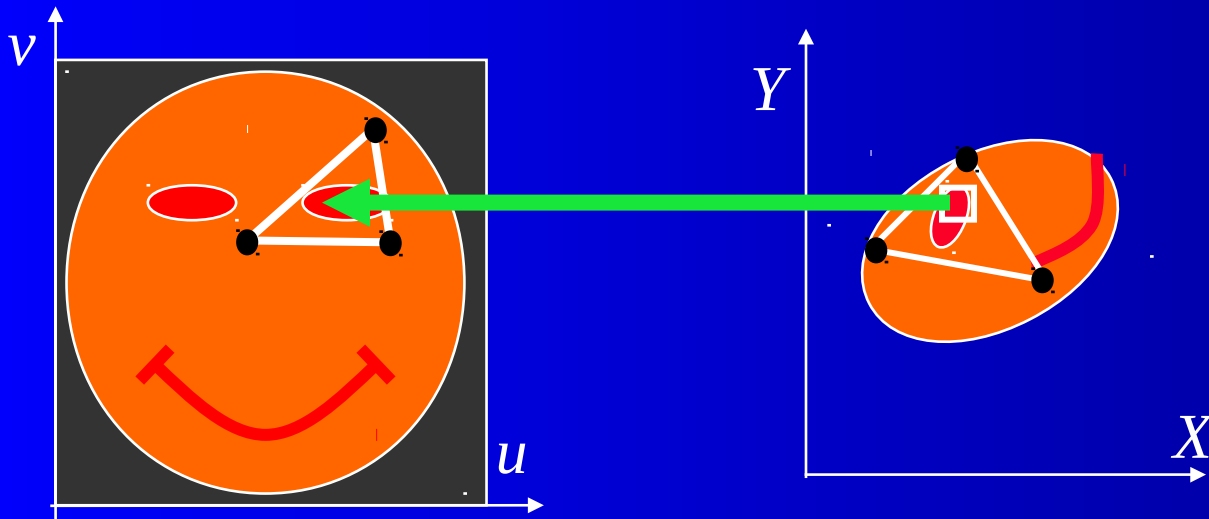
$$\begin{aligned} v_1/h_1 &= a_v X_1 + b_v Y_1 + c_v \\ v_2/h_2 &= a_v X_2 + b_v Y_2 + c_v \\ v_3/h_3 &= a_v X_3 + b_v Y_3 + c_v \end{aligned}$$

$$a_v, b_v, c_v$$

$$\begin{aligned} 1/h_1 &= a_h X_1 + b_h Y_1 + c_h \\ 1/h_2 &= a_h X_2 + b_h Y_2 + c_h \\ 1/h_3 &= a_h X_3 + b_h Y_3 + c_h \end{aligned}$$

$$a_h, b_h, c_h$$

Perspektíva helyes textúrázás



$$\begin{aligned}u/h &= a_u X + b_u Y + c_u \\v/h &= a_v X + b_v Y + c_v \\1/h &= a_h X + b_h Y + c_h\end{aligned}$$

$$\begin{aligned}u &= \frac{a_u X + b_u Y + c_u}{a_h X + b_h Y + c_h} \\v &= \frac{a_v X + b_v Y + c_v}{a_h X + b_h Y + c_h}\end{aligned}$$

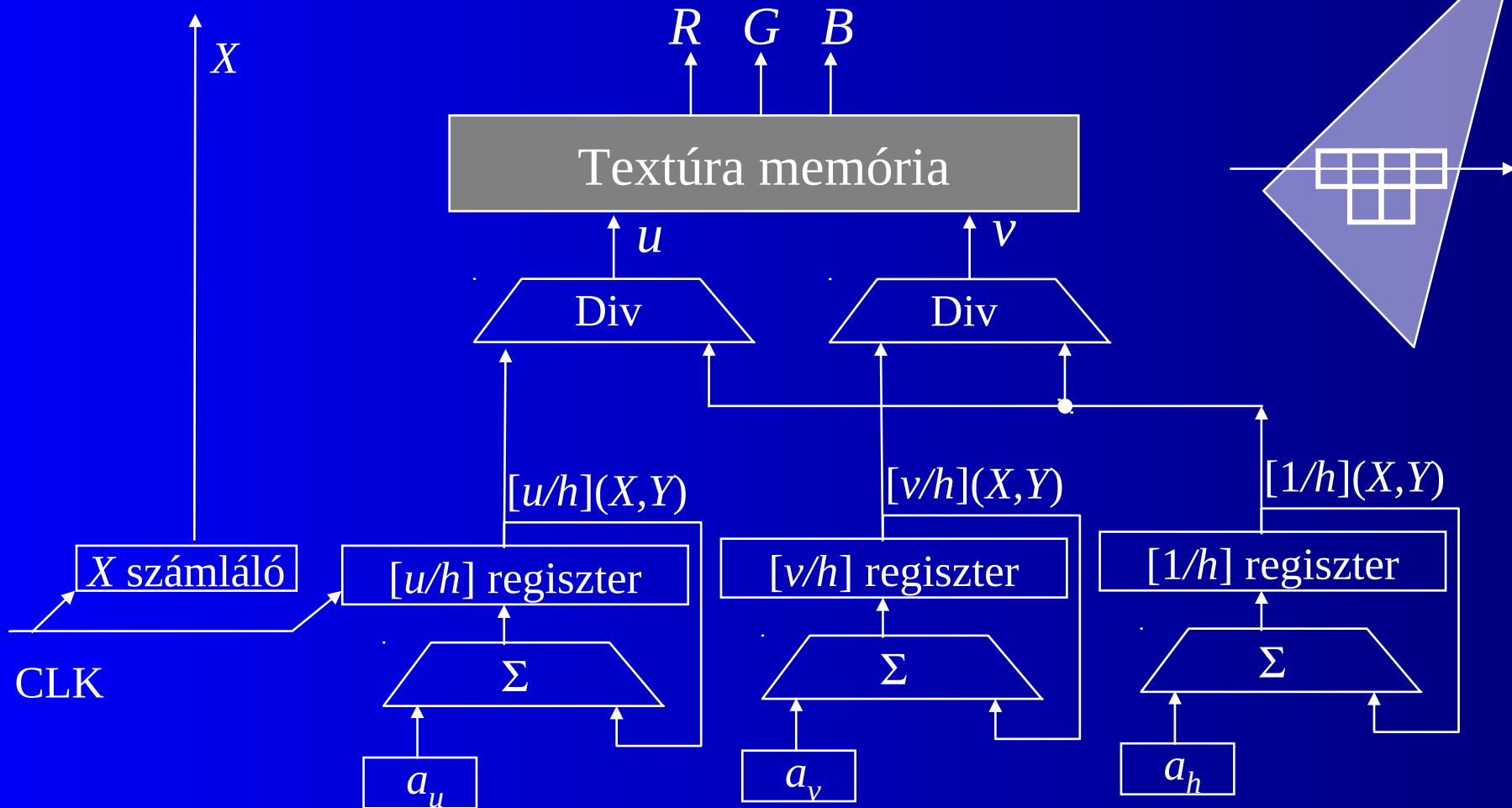
Perspektív helyes



lineáris

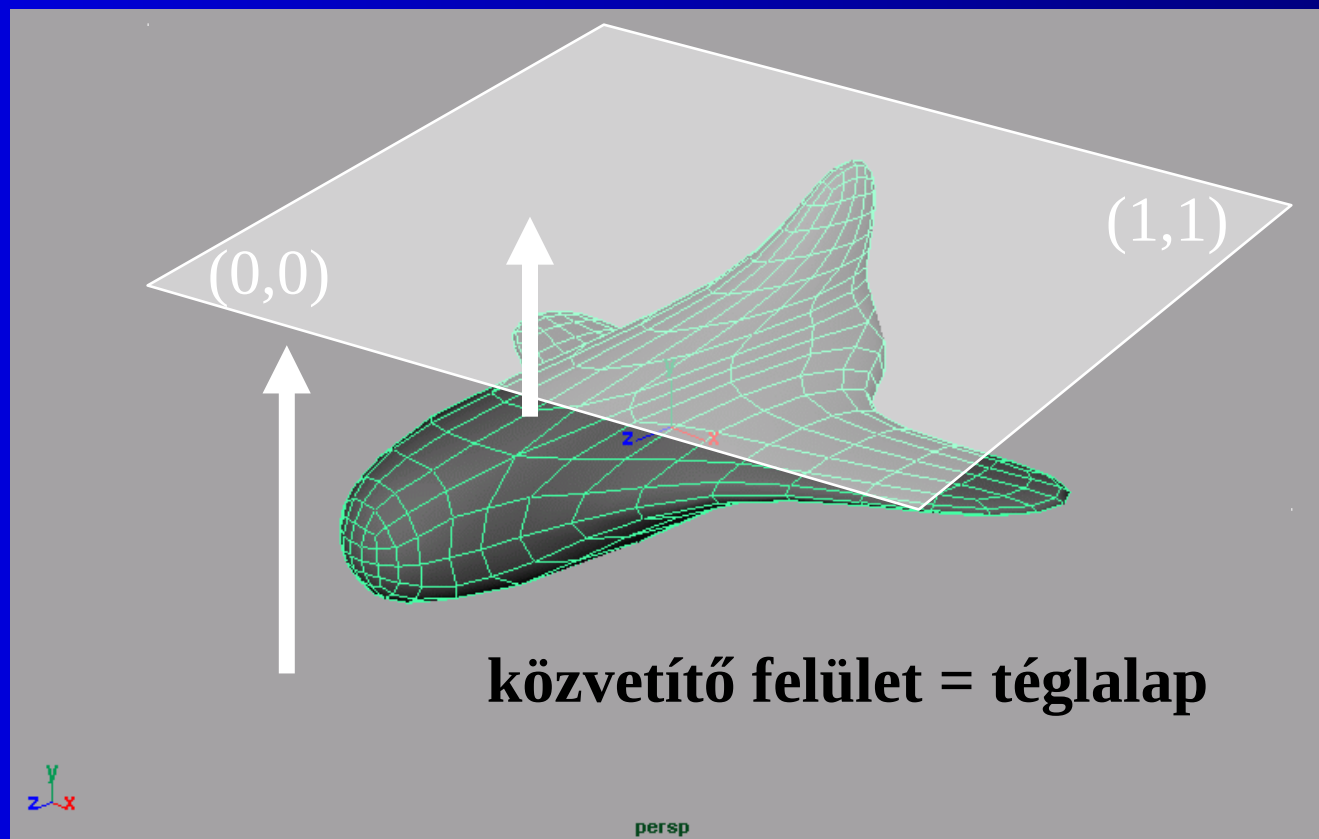
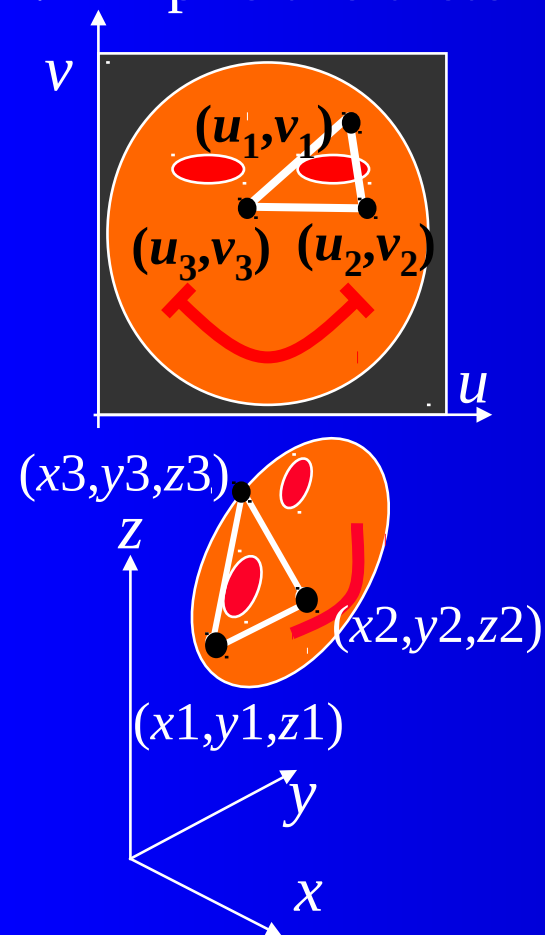


Perspektíva helyes interpolációs hw.

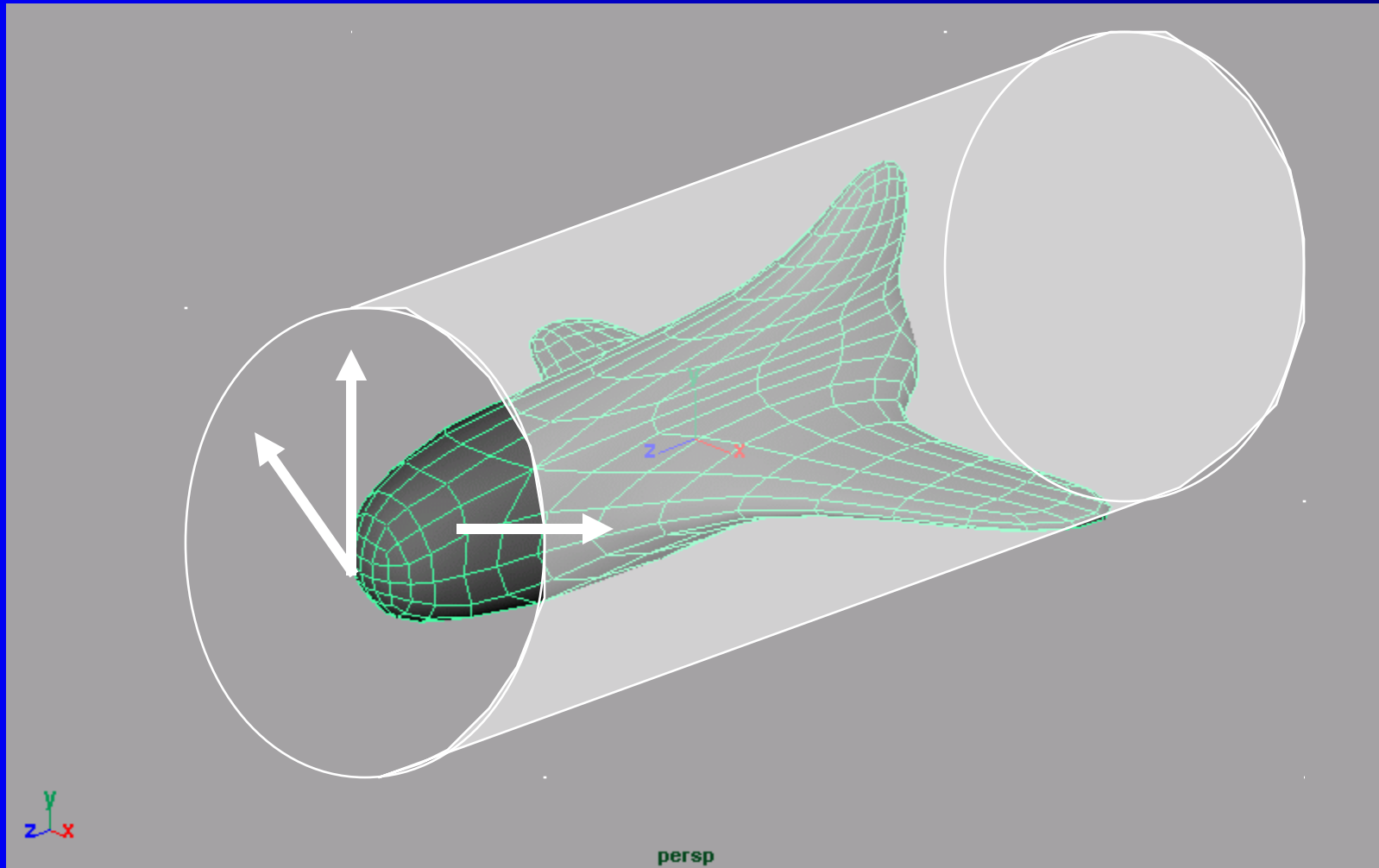


Textúra függvény definíciója

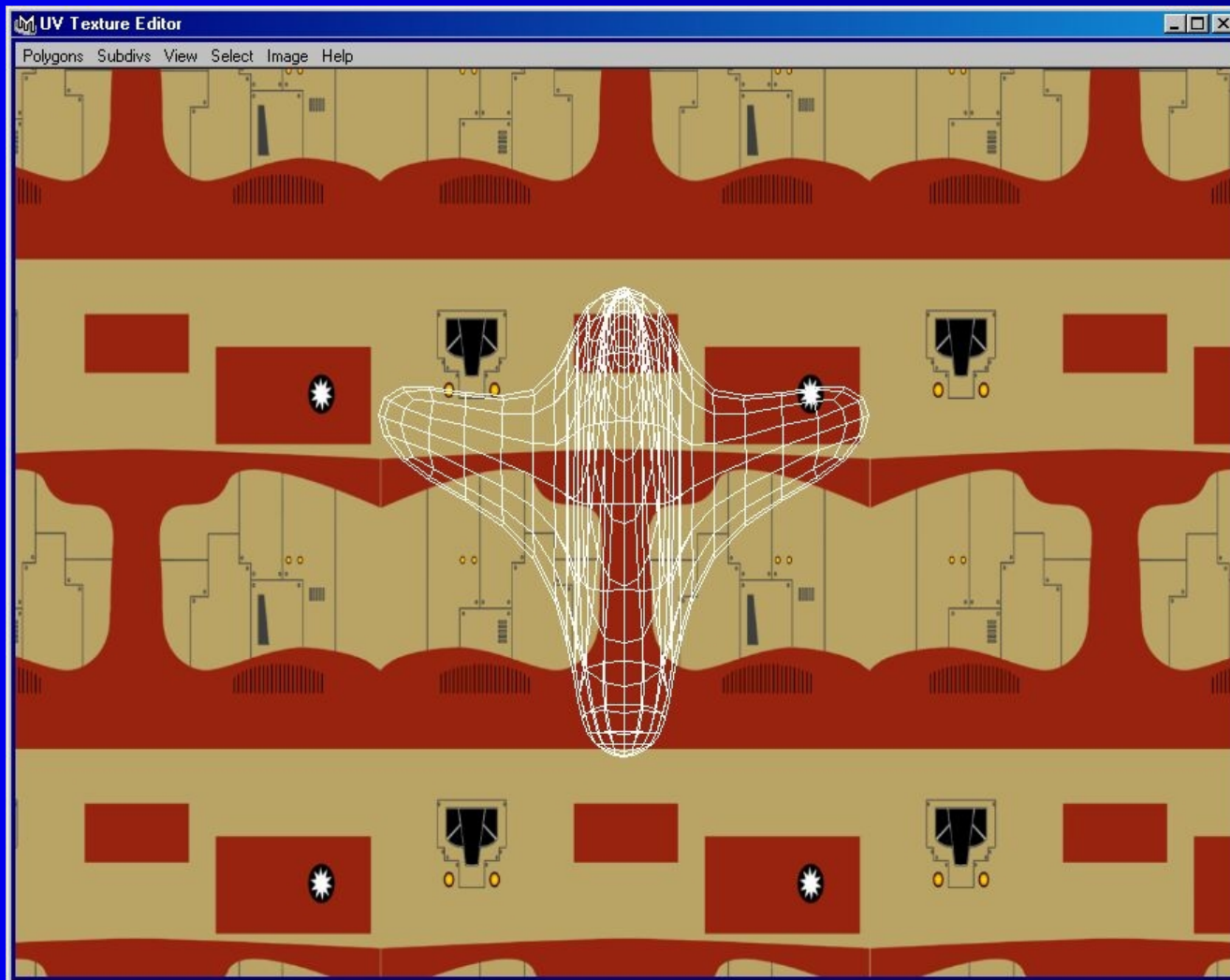
1. Parametrikus felületeknél triviális
2. Implicit felületeknél és poligonmodellnél: **közvetítő felületek**



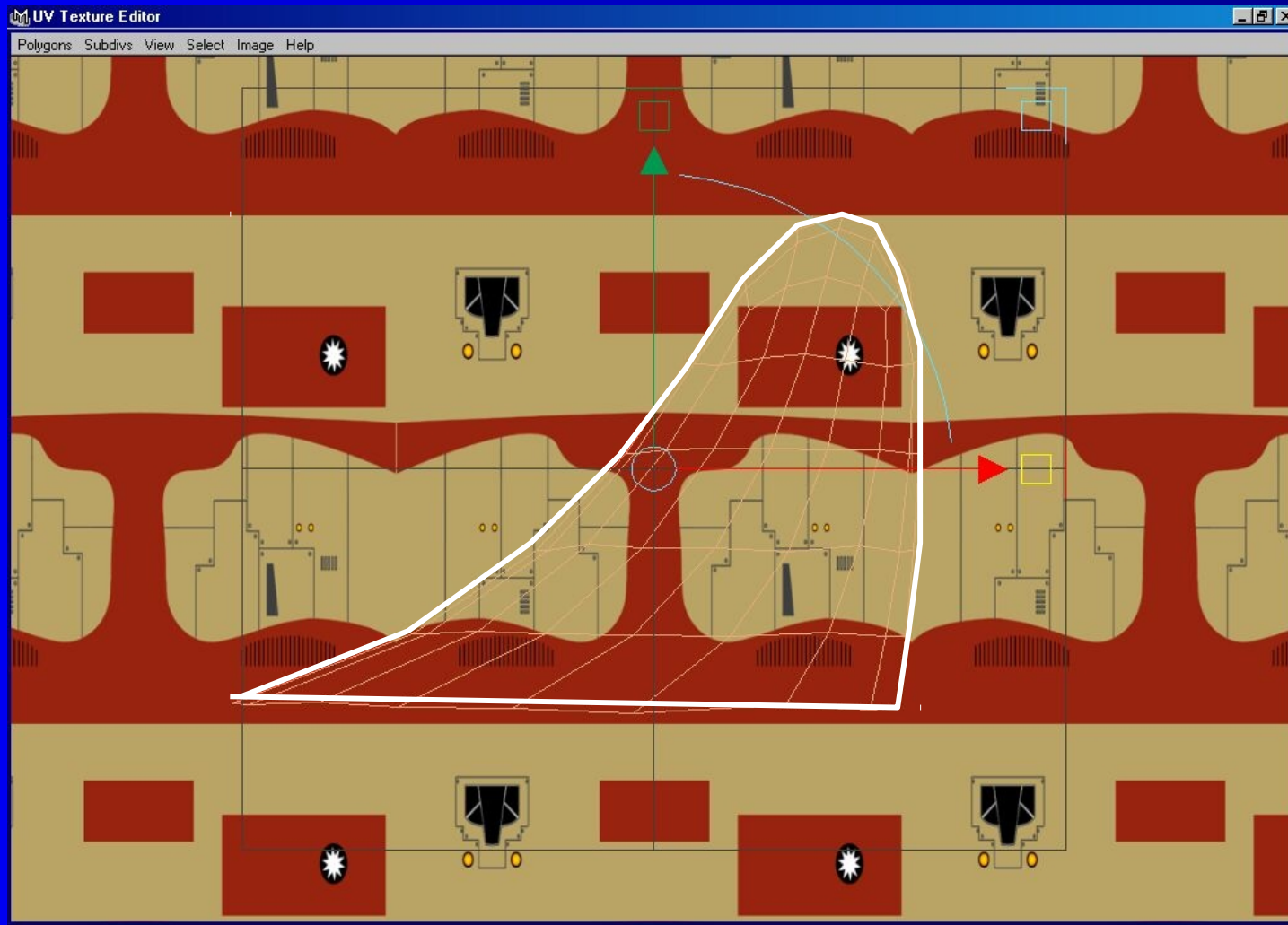
Henger (gömb) közvetítő felület



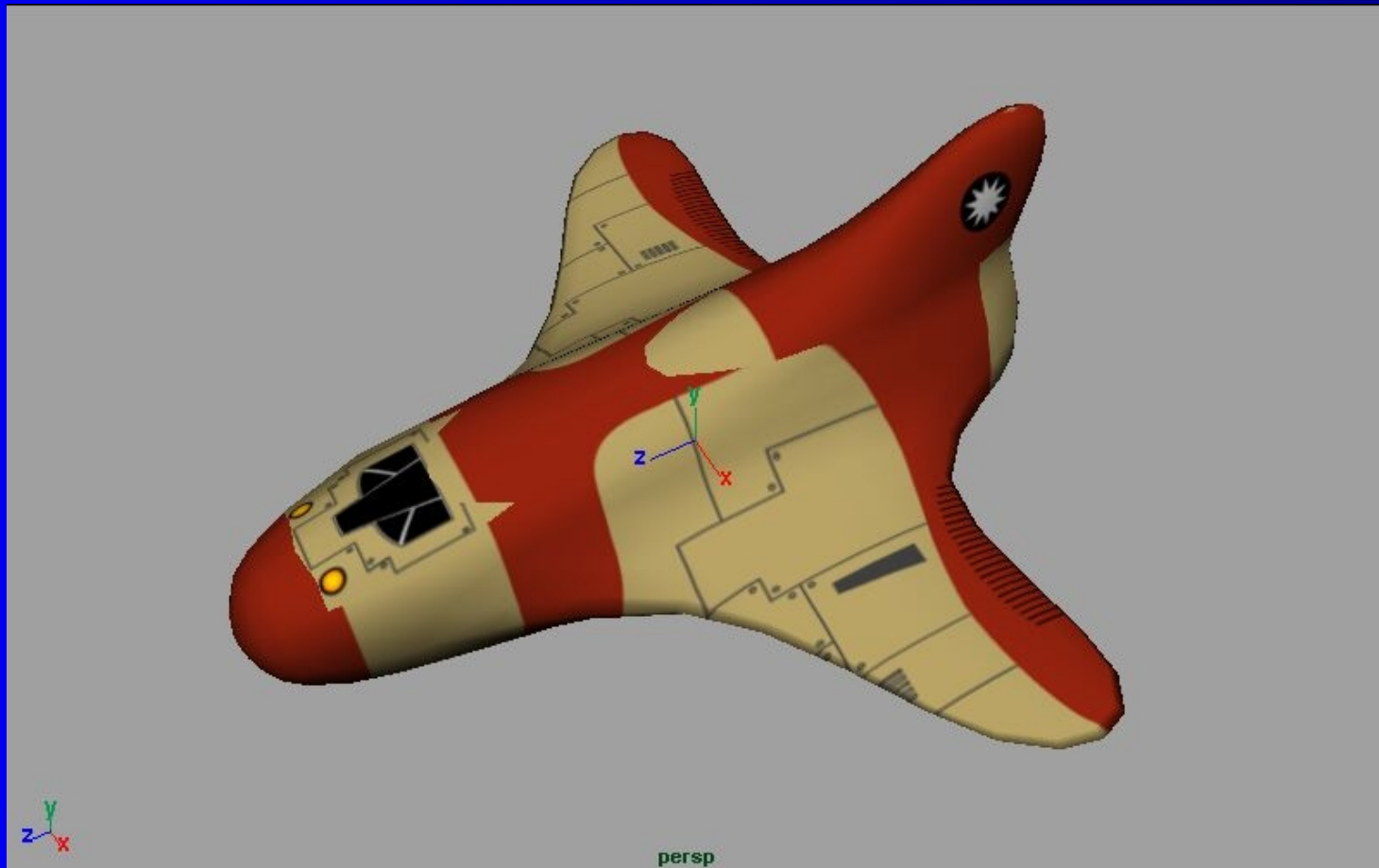
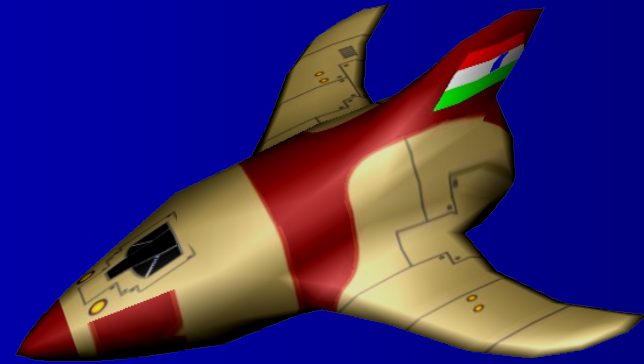
Téglalap közvetítő felület: textúratér nézet



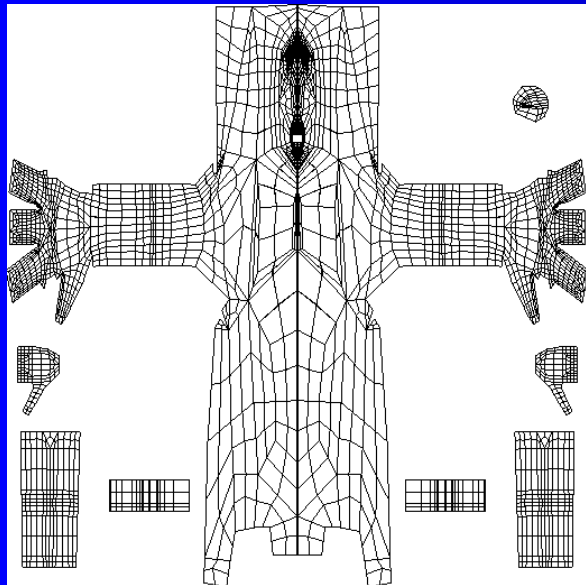
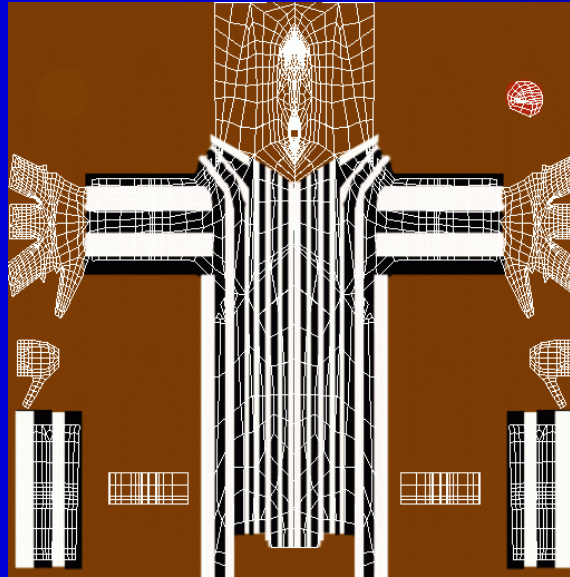
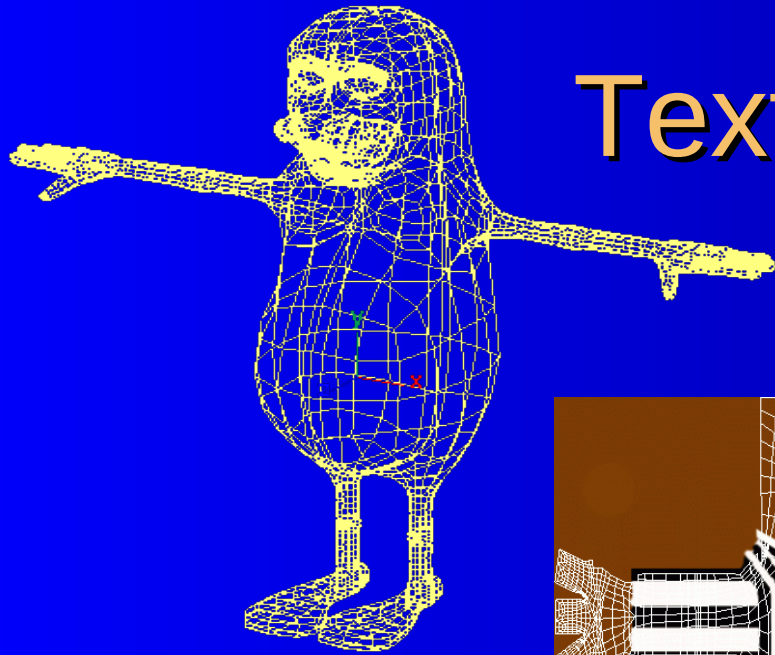
Téglalap közvetítő felület: textúratér nézet



Textúrázott űrhajó

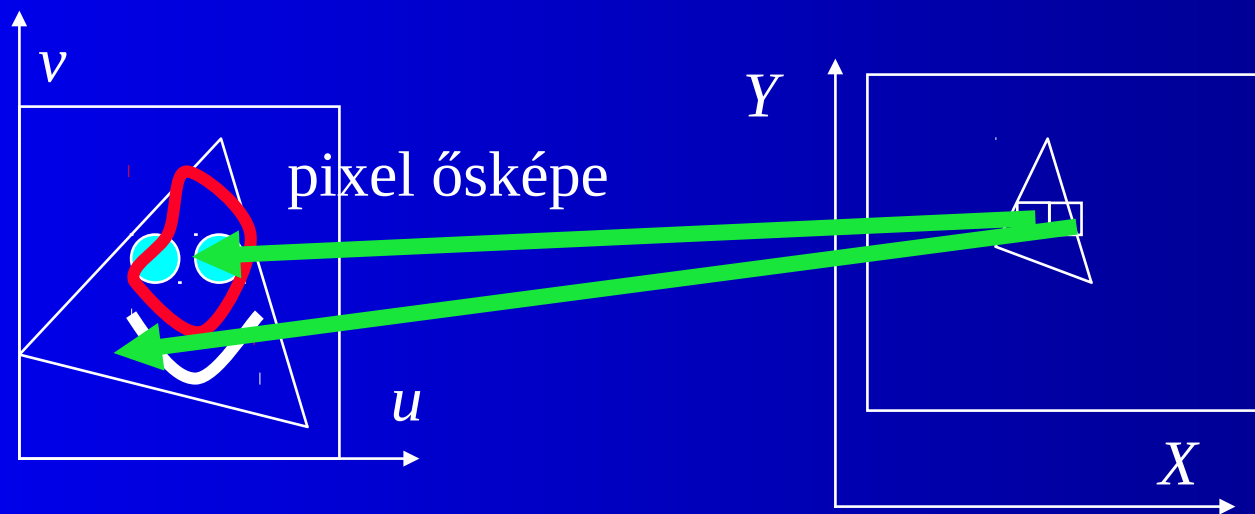


Textúrázás = kiterítés

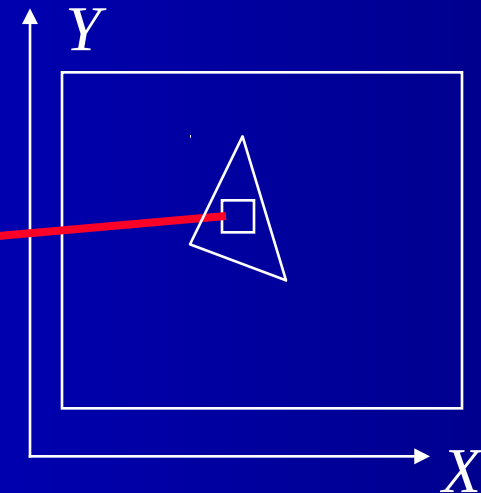
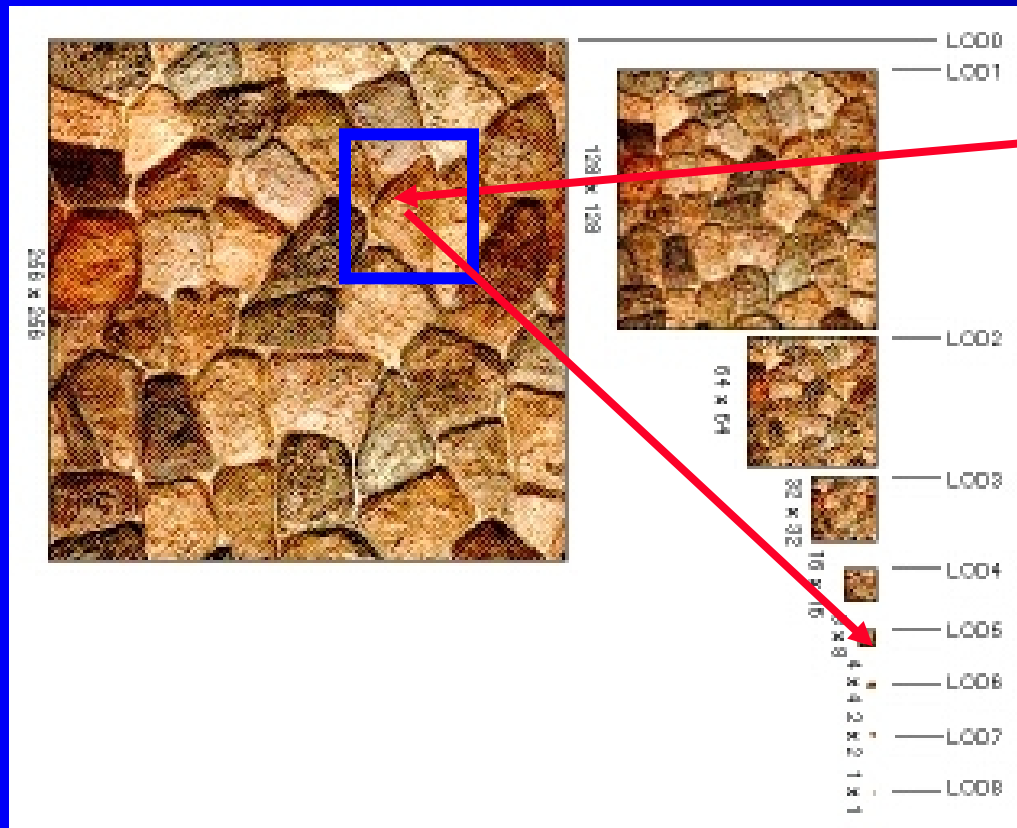


Torzításcsökkentés relaxációval

Textúrák szűrése



Mip-map adatstruktúra

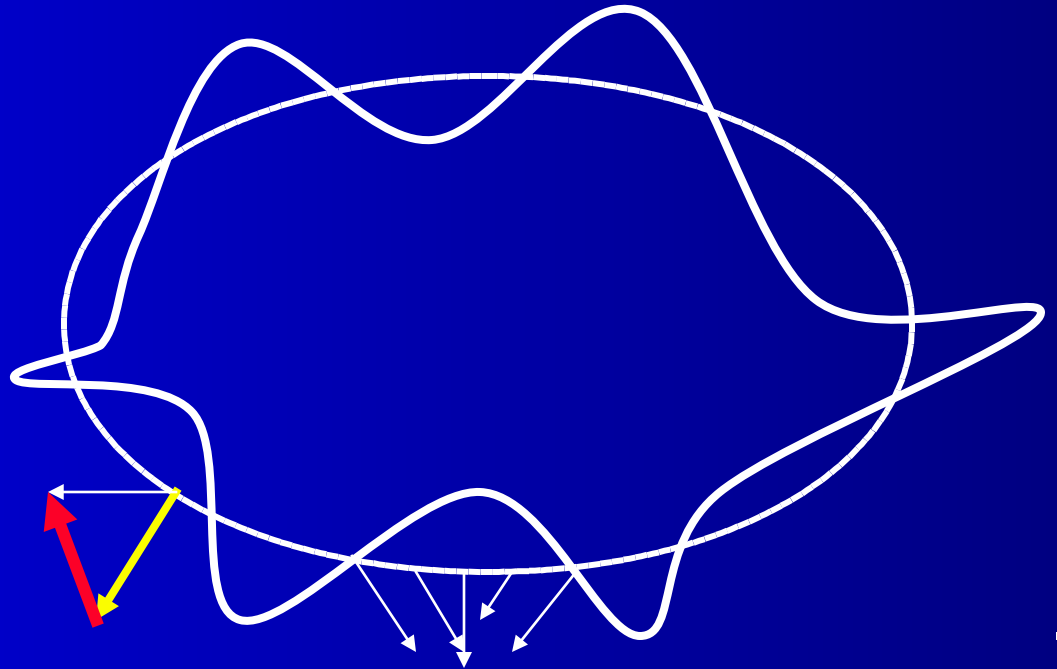
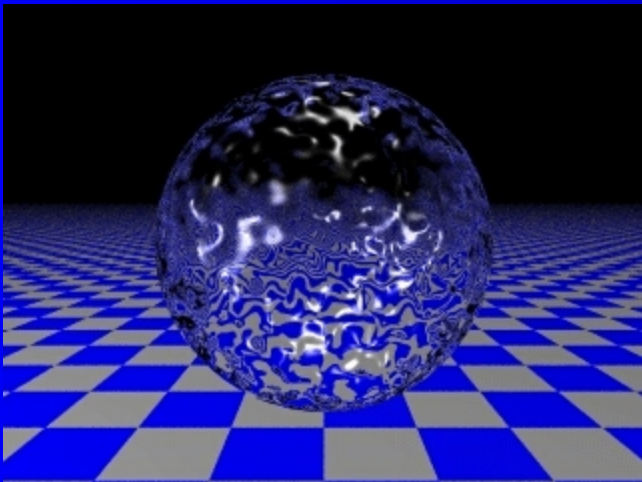
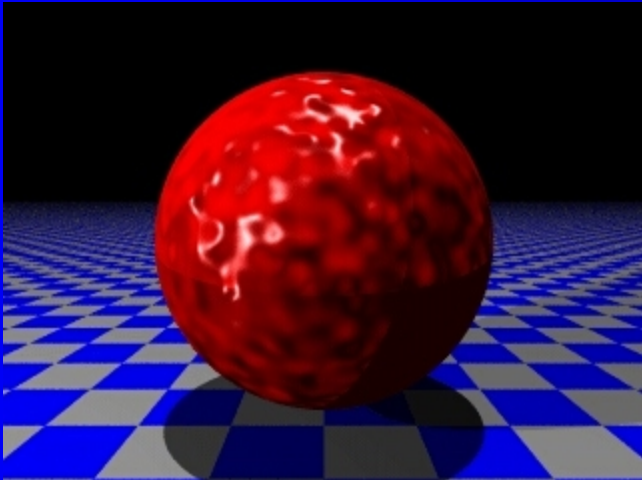


Ennek a deriváltját becsüljük:

$$u = \frac{a_u X + b_u Y + c_u}{a_h X + b_h Y + c_h}$$

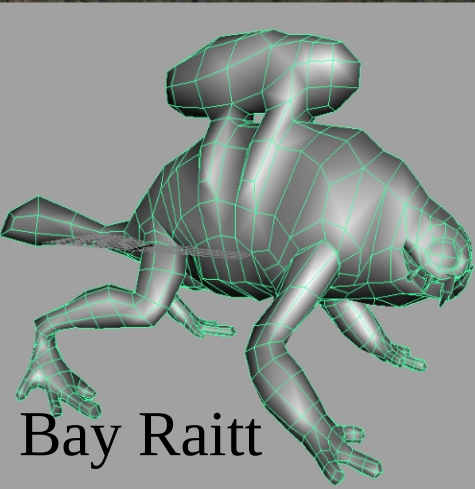
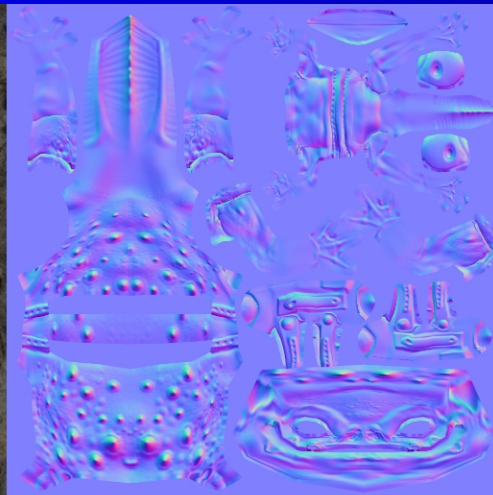
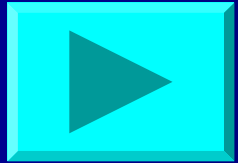
$$v = \frac{a_v X + b_v Y + c_v}{a_h X + b_h Y + c_h}$$

Bucka leképzés (Bump mapping)



Tároljuk a normálvektort táblázatban
vagy a felületi elmozdulást

Displacement mapping



Bay Raitt

Digic Pictures

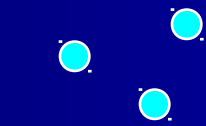
*In theory, there is no difference
between theory and practice.
In practice, there is.*

OpenGL

Szirmay-Kalos László

OpenGL: primitívek

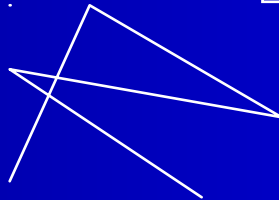
```
glBegin(GL_TRIANGLES);  
    Csúcspont tulajdonságok ...  
    glVertex3d(x, y, z);  
    ...  
glEnd( );
```



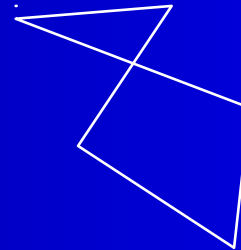
GL_POINTS



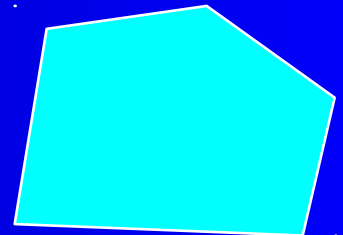
GL_LINES



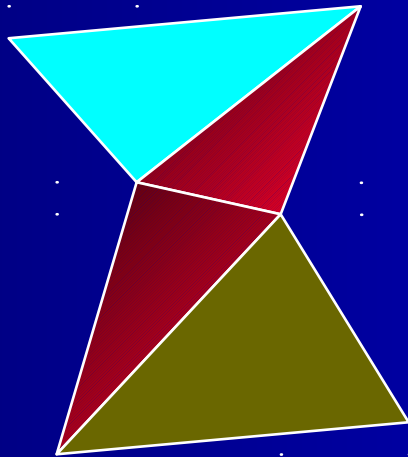
GL_LINE_STRIP



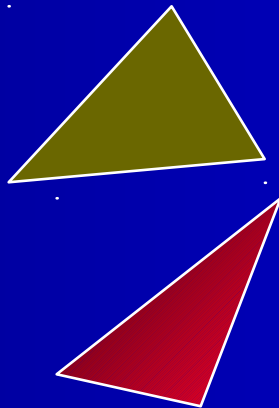
GL_LINE_LOOP



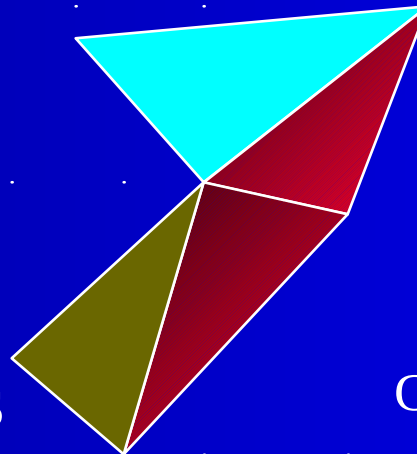
GL_POLYGON



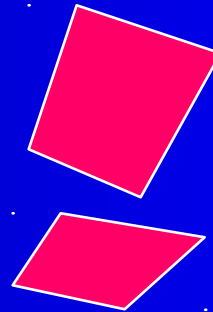
GL_TRIANGLE_STRIP



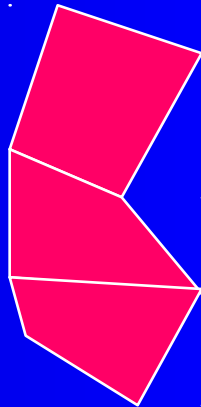
GL_TRIANGLES



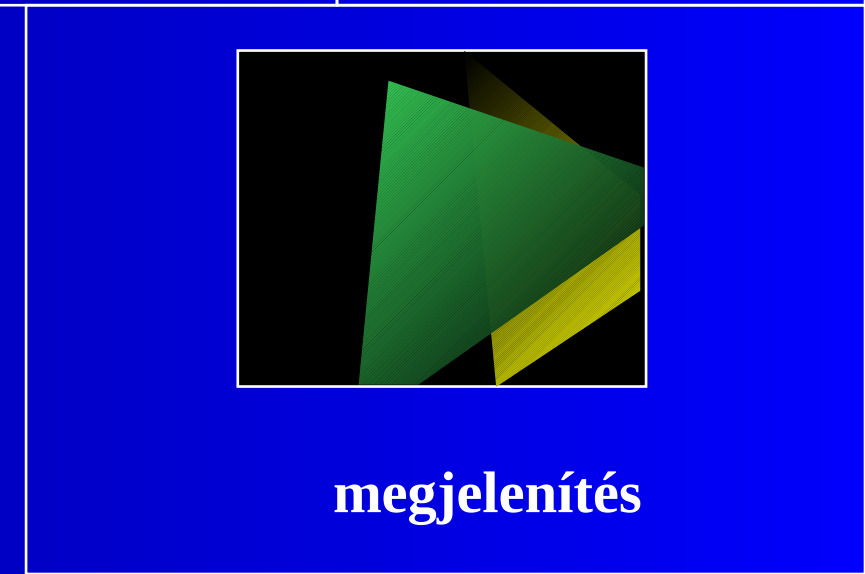
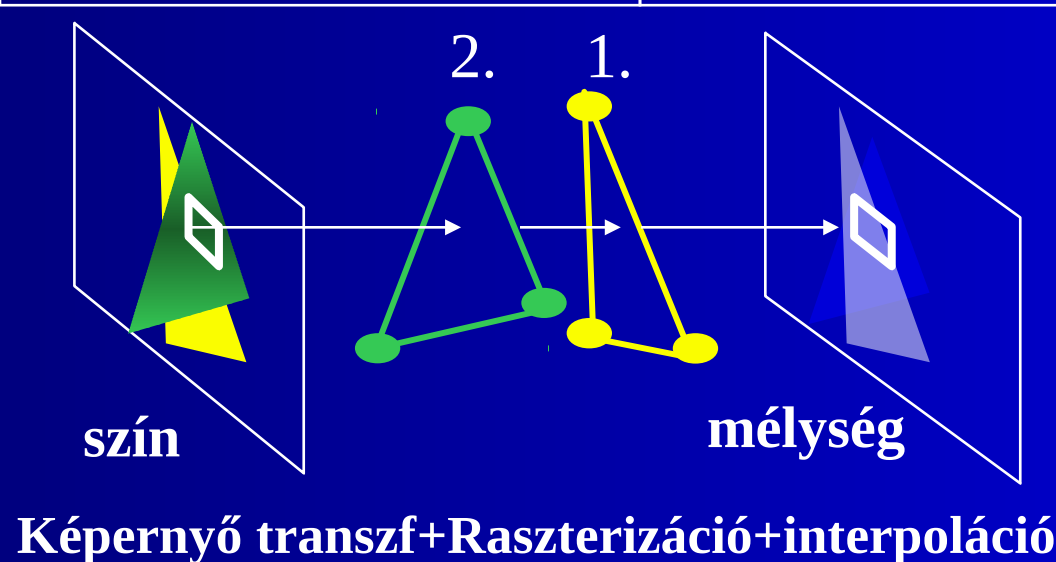
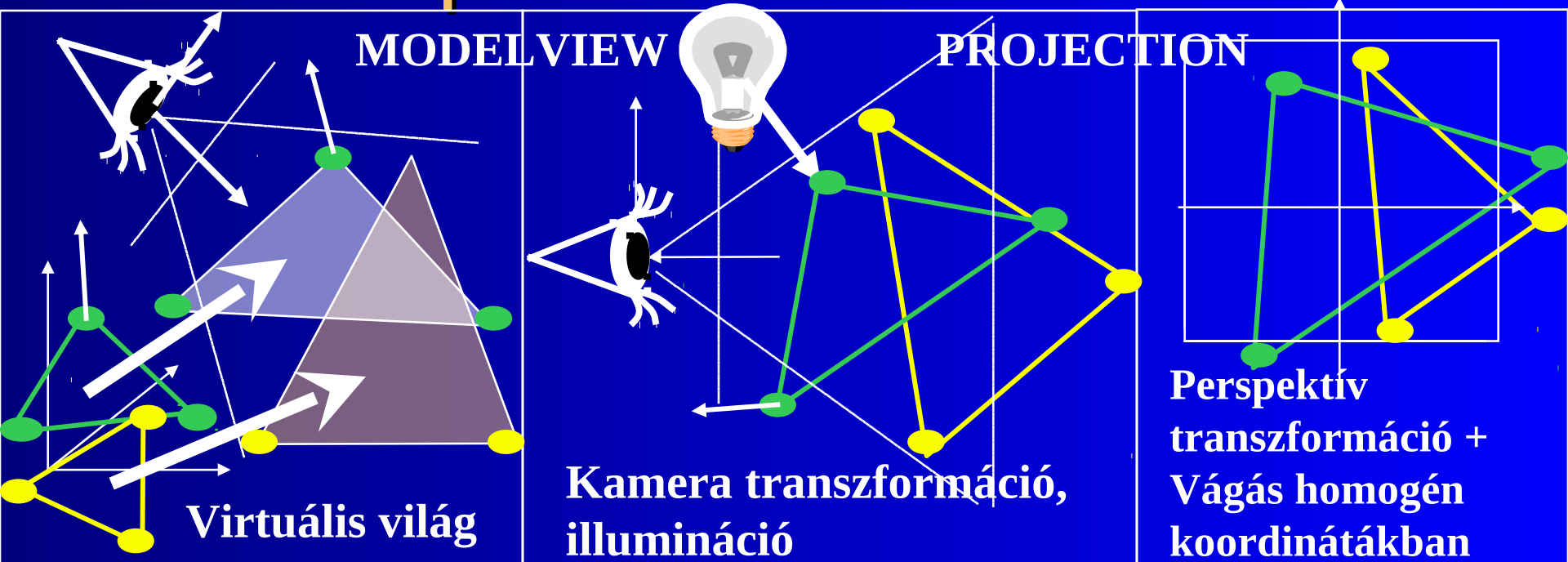
GL_TRIANGLE_FAN



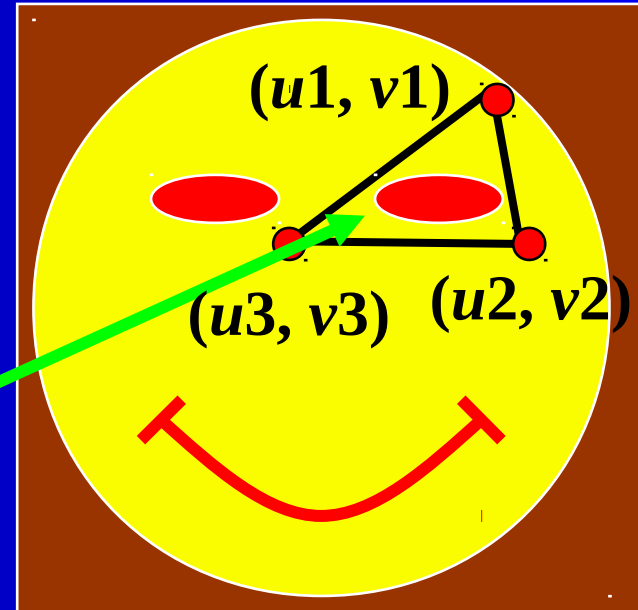
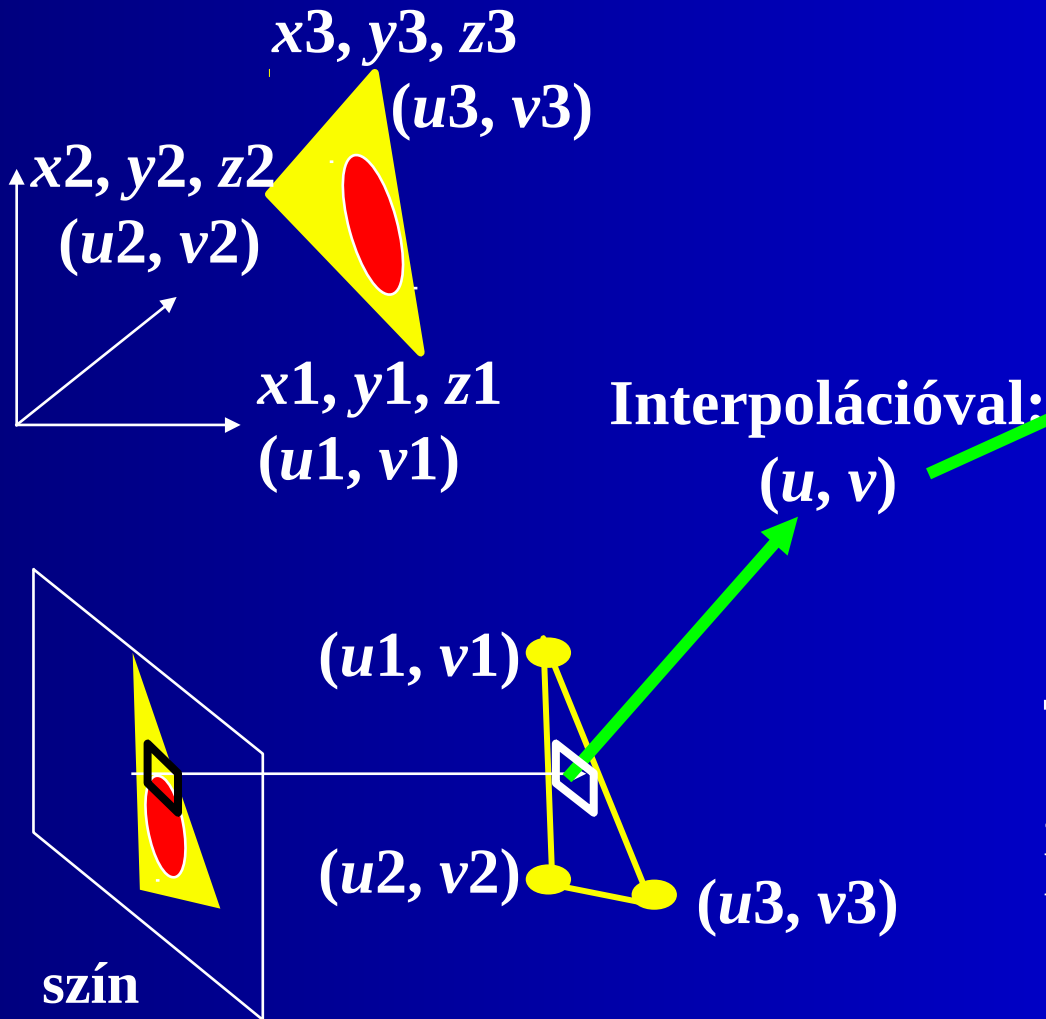
GL_QUADS



OpenGL csővezeték



Textúra leképzés



Textúra objektum a
grafikus kártya memóriájában:
Kép + szűrés + kilógás kezelés

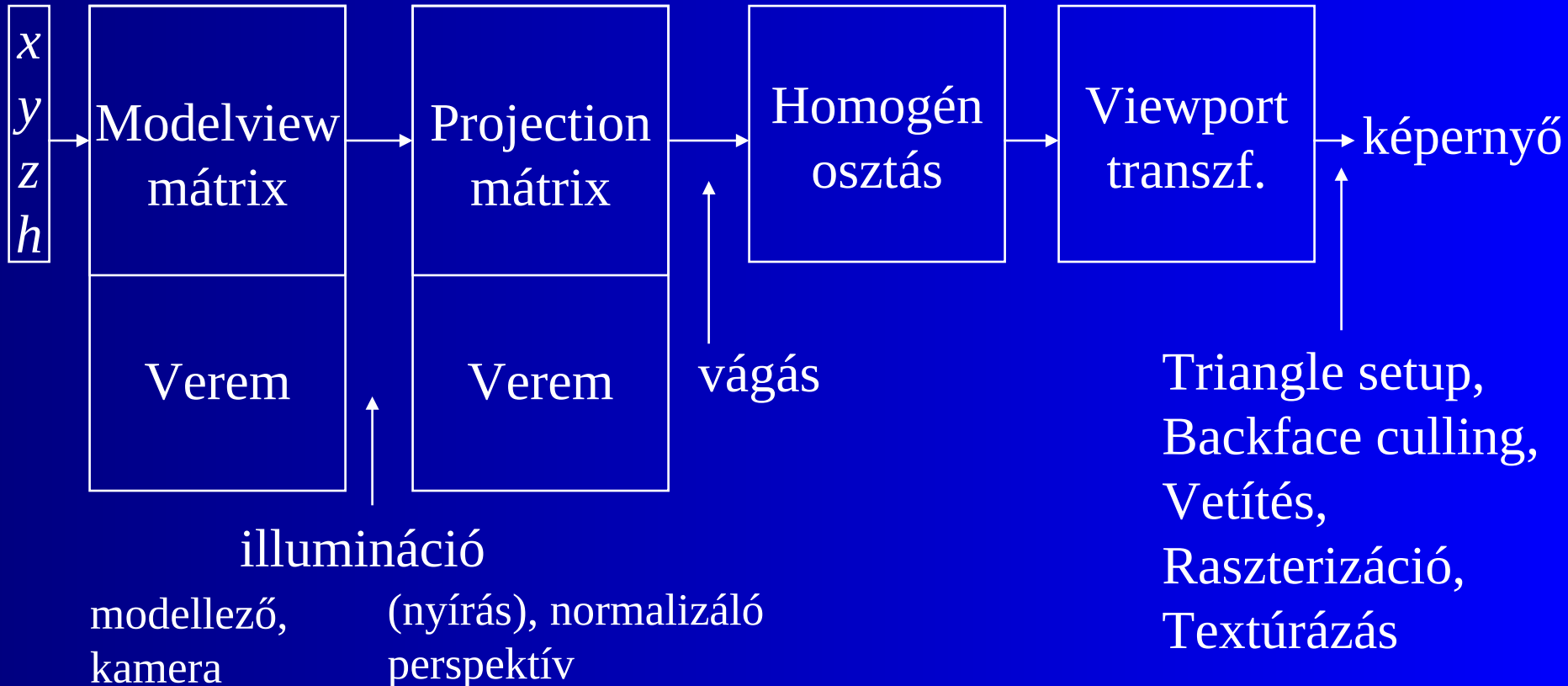
Transzformációk

Referencia
helyzet

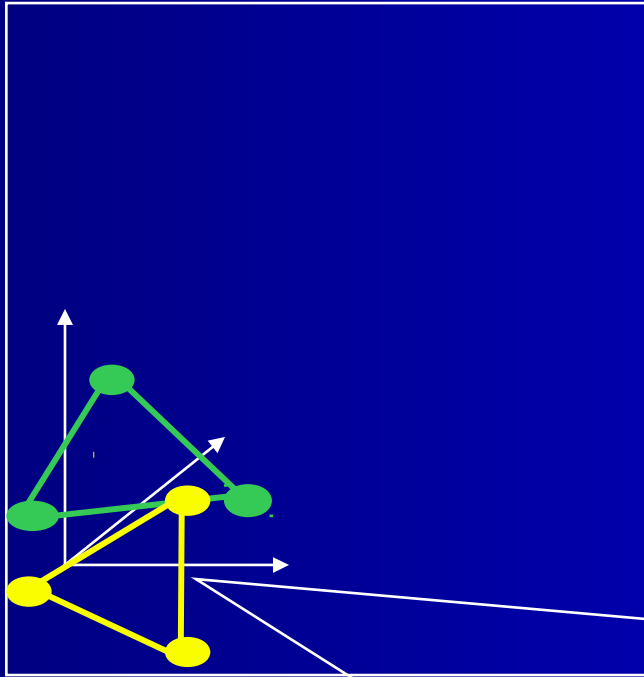
Kamera k.

Homogén
vágási k.

Normalizált
képernyő



A csővezeték etetése: A modell



```
glBegin(GL_TRIANGLES);
```

```
    glNormal3f(nx1,ny1,nz1);
```

```
    glColor3f(r1,g1,b1);
```

```
    glTexCoord2f(u1,v1)
```

```
    glVertex3f(x1,y1,z1);
```

állapot

```
    glNormal3f(nx2,ny2,nz2);
```

```
    glColor3f(r2,g2,b2);
```

```
    glTexCoord2f(u2,v2)
```

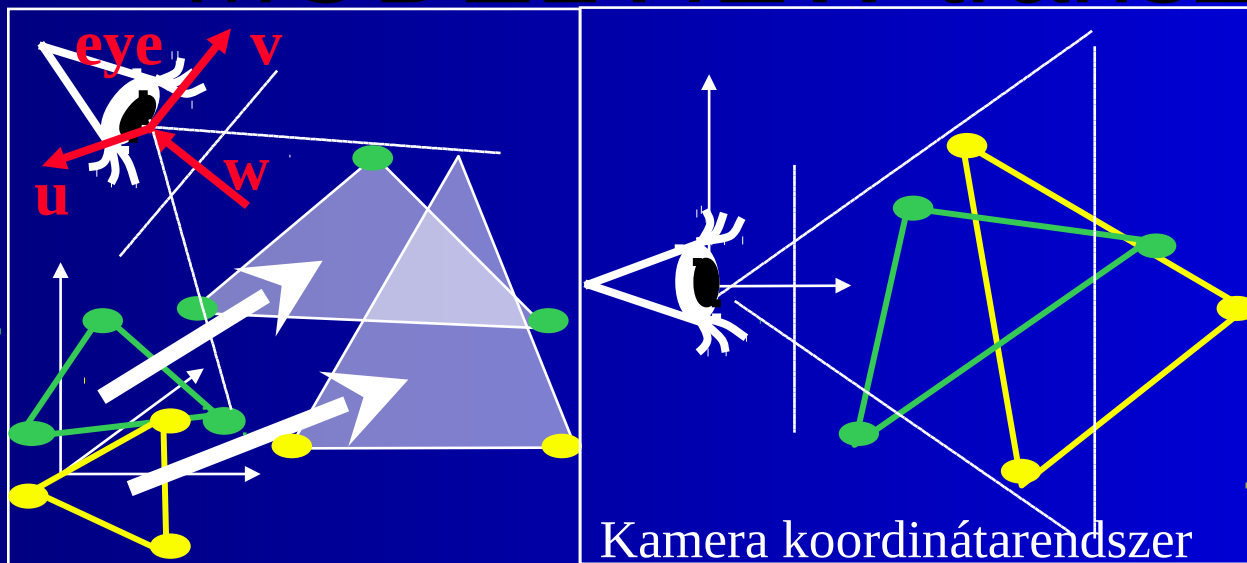
```
    glVertex3f(x2,y2,z2);
```

állapot

```
    ...
```

```
glEnd( );
```

MODELVIEW transzformáció



$$\begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ \text{eye} & & & 1 \end{pmatrix}^{-1}$$

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity( );
```

```
gluLookAt(eyex, eyey, eyez, vrp_x, vrp_y, vrp_z, up_x, up_y, up_z); //VIEW
```

```
glTranslatef(px, py, pz); //MODEL
```

```
glRotatef(ang, axis_x, axis_y, axis_z);
```

```
glScalef(sx, sy, sz);
```

↑ sorrend

$\begin{bmatrix} x & y & z & 1 \end{bmatrix}$

$$\begin{bmatrix} s_x & & & \\ & s_y & & \\ & & s_z & \\ & & & 1 \end{bmatrix}$$

$$\begin{bmatrix} R & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ p_x & p_y & p_z & 1 \end{bmatrix}$$

$$\begin{bmatrix} T_{\text{view}} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

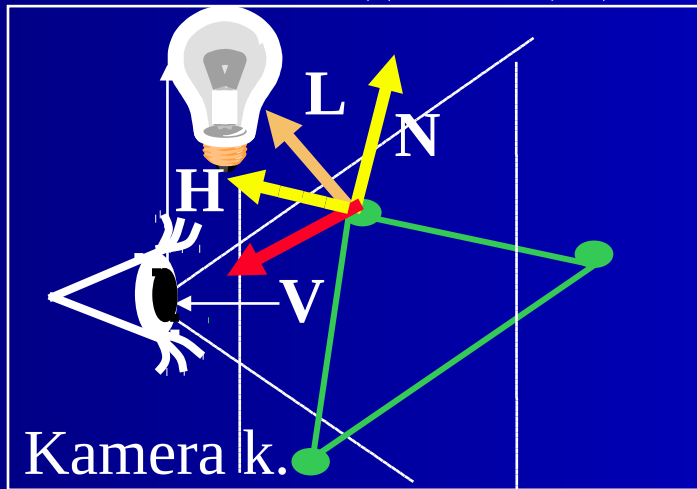
Illumináció

A szín a megvilágításból keletkezik:

```
glEnable(GL_LIGHTING);
```

Phong-Blinn model

Color = Emission +
ka * Ia +
kd * Id · (N • L)⁺ +
ks * Is · ((N • H)⁺)^{shininess}



```
glLightModeli(  
GL_LIGHT_MODEL_LOCAL_VIEWER,
```

```
Glfloat kd[] = {0.0, 1.0, 1.0, 1.0};  
Glfloat ks[] = {1.0, 0.5, 1.0, 1.0};  
Glfloat ka[] = {0.2, 0.0, 0.2, 1.0};
```

```
glMaterialfv( GL_FRONT,  
              GL_AMBIENT, ka);  
glMaterialfv( GL_FRONT,  
              GL_DIFFUSE, kd);  
glMaterialfv( GL_FRONT,  
              GL_SPECULAR, ks);  
glMaterialf( GL_FRONT,  
             GL_SHININESS, 20);
```

```
glEnable( GL_NORMALIZE );  
glBegin(GL_TRIANGLES);  
glNormal3d(0.0, 0.0, 1.0);  
glVertex3d(10.0, 20.0, 10.0);  
  
glNormal3d(1.0, 0.0, 0.0);  
glVertex3d(10.0, 10.0, 10.0);  
....
```

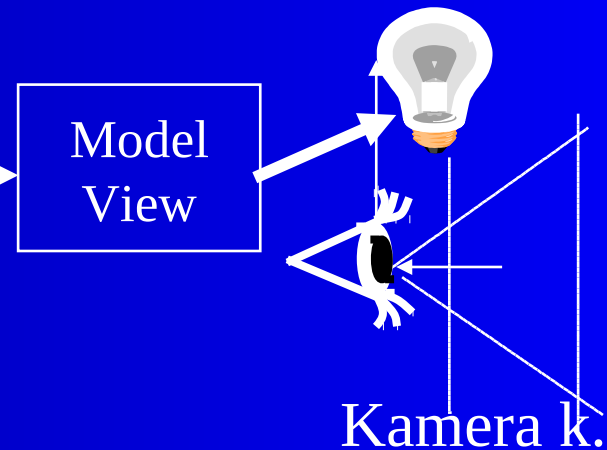
Fényforrások

```
GLfloat I[] = {0.0, 1.0, 1.0, 0.0};  
GLfloat pos[] = {10, 20, 10, 1.0 };
```

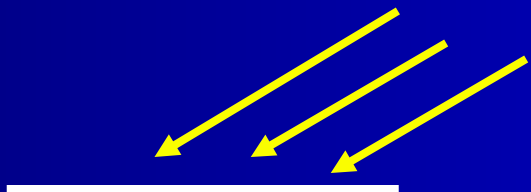
```
glLightfv(GL_LIGHT0, GL_DIFFUSE, I);  
glLightfv(GL_LIGHT0, GL_POSITION, pos);  
glLightfv(GL_LIGHT1, ....
```

```
glEnable(GL_LIGHT0); // bekapcs
```

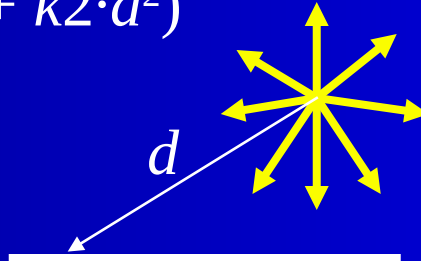
Ha $h = 0$, akkor direkcionális
egyébként pozicionális



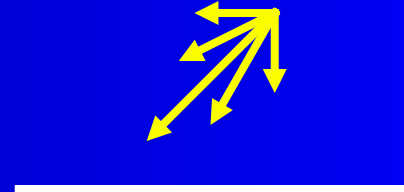
- Külön fényintenzitás az ambiens, diffúz, spekuláris visszaverődéshez
- Direkcionális, Pozicionális (izotróp ill. spot)
- Attenuation: $1/(k_0 + k_1 \cdot d + k_2 \cdot d^2)$



direkcionális

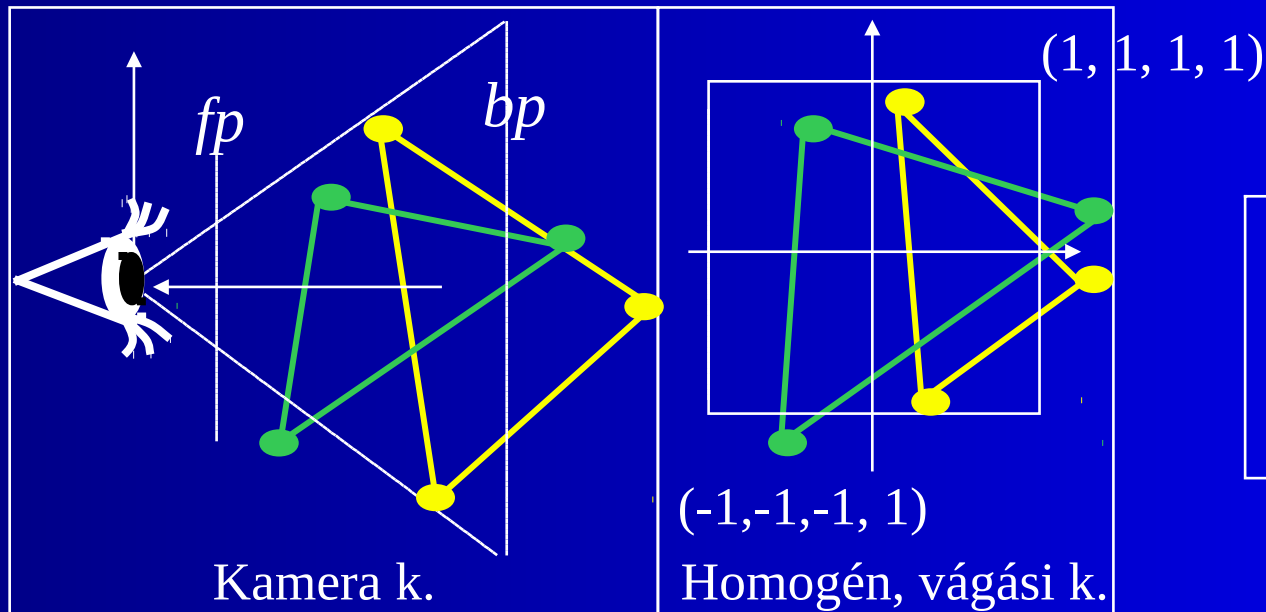


Izotróp pozicionális



Spot: $\cos^n \alpha$

PROJECTION transzformáció



Vágás

$$-h < X_h < h$$

$$-h < Y_h < h$$

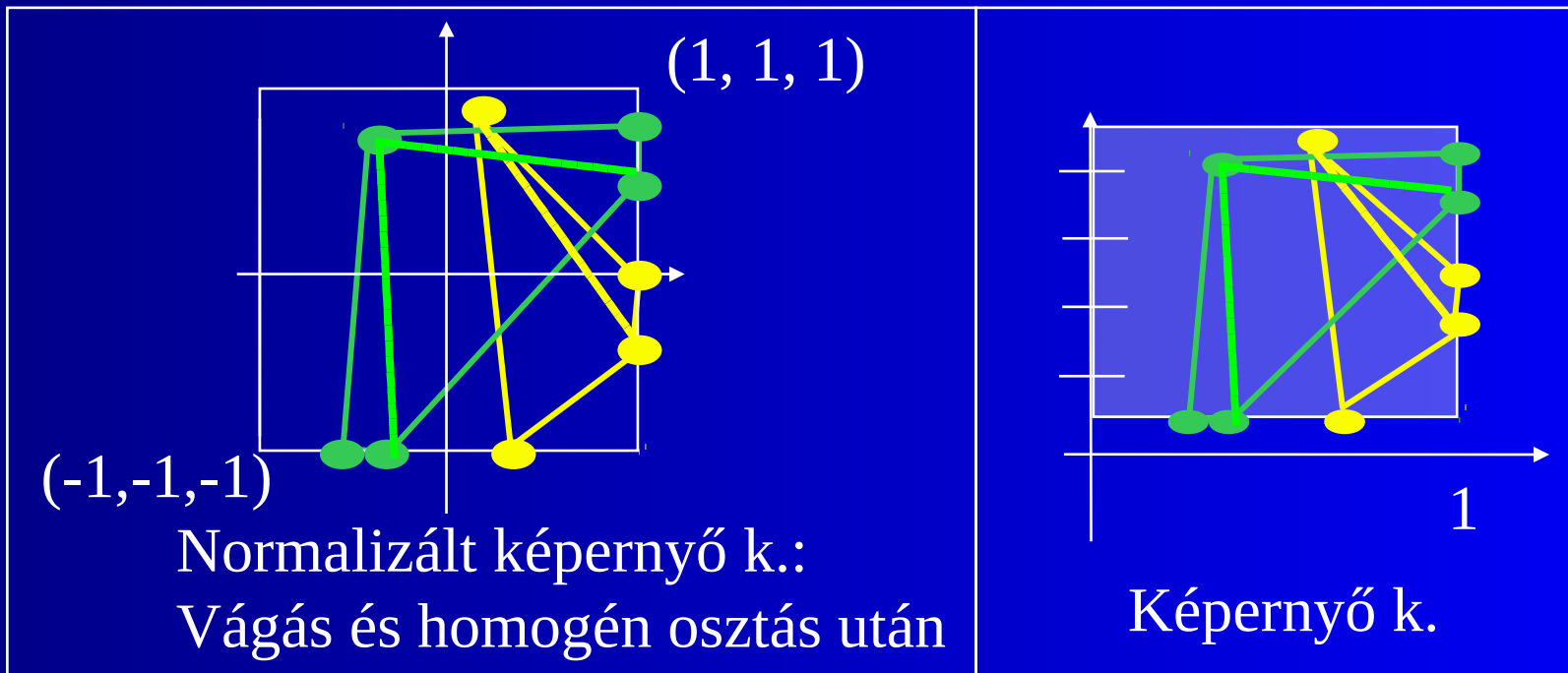
$$-h < Z_h < h$$

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
```

```
gluPerspective(fov, asp, fp, bp);
```

$$\begin{pmatrix} 1/(\text{tg}(\text{fov}/2) \cdot \text{asp}) & 0 & 0 & 0 \\ 0 & 1/\text{tg}(\text{fov}/2) & 0 & 0 \\ 0 & 0 & -(fp+bp)/(bp-fp) & -1 \\ 0 & 0 & -2fp \cdot bp/(bp-fp) & 0 \end{pmatrix}$$

Képernyő transzformáció



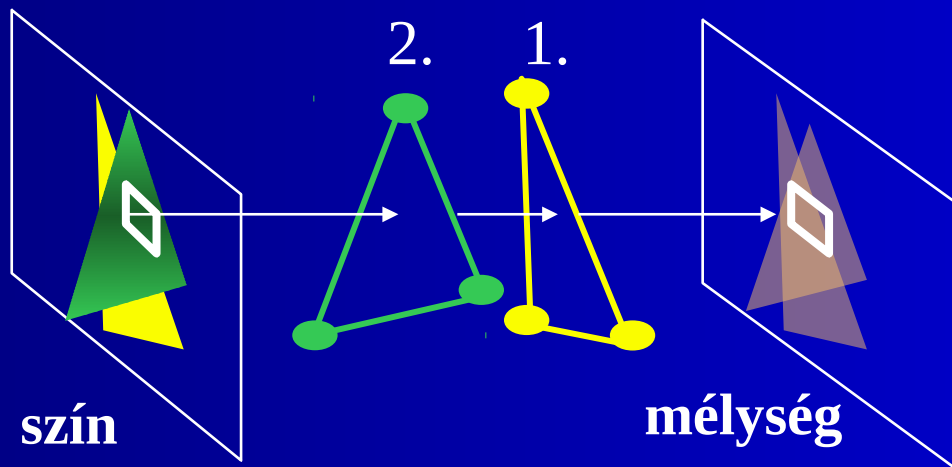
`glViewport( left, bottom, width, height );`

Z-bufferes takarás és konstans vagy Gouraud árnyalás

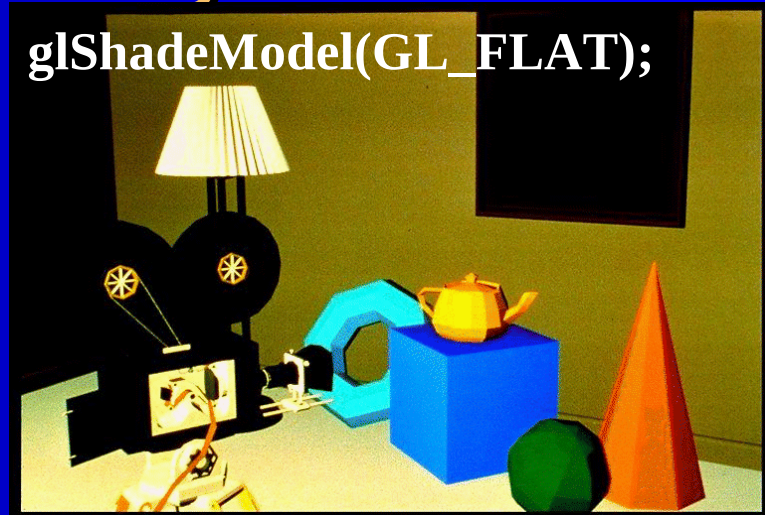
Inicializálás:

```
glutInitDisplayMode(GLUT_RGBA |  
GLUT_DEPTH);
```

```
glEnable(GL_DEPTH_TEST);  
glDisable(GL_CULL_FACE);  
glShadeModel(GL_SMOOTH);  
glDisable(GL_TEXTURE_2D);
```



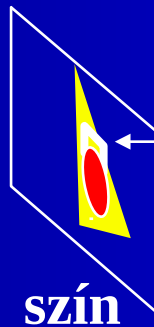
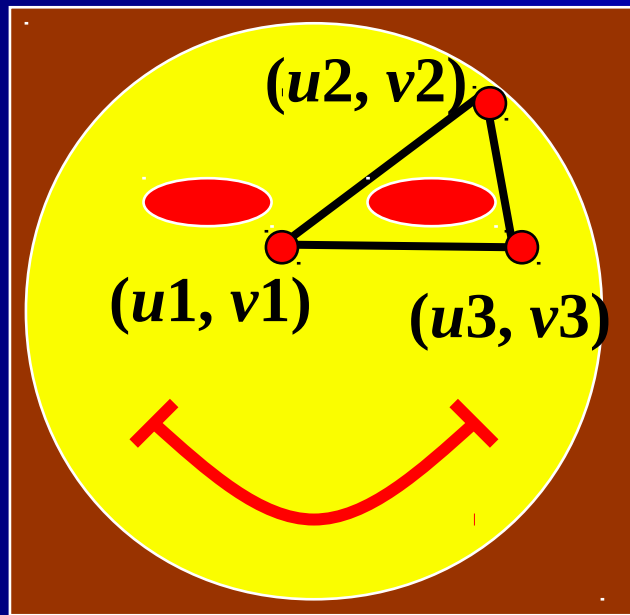
```
glShadeModel(GL_FLAT);
```



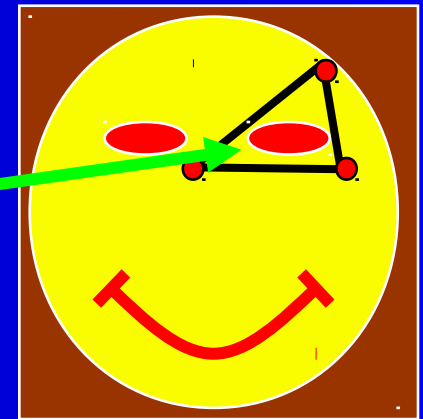
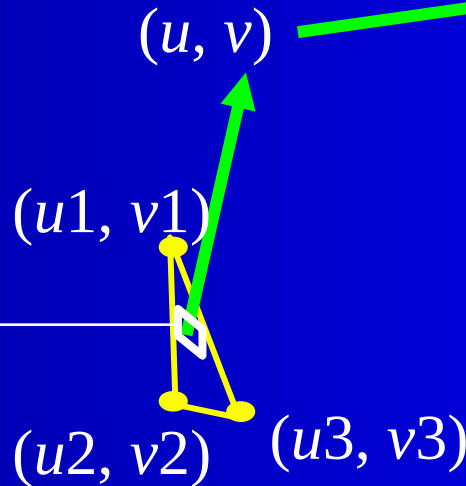
```
glShadeModel(GL_SMOOTH);
```



Textúra leképzés



Interpolációval:



Textúra objektum
a grafikus kártya
memóriájában

Ne az interpolált szín legyen a pixel szín:

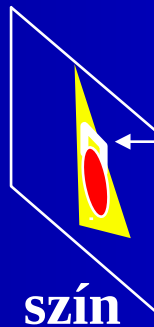
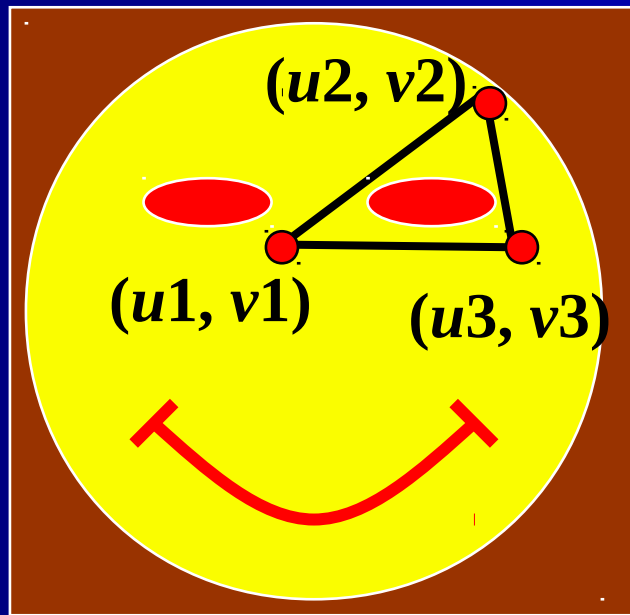
```
glEnable(GL_TEXTURE_2D);  
glTexEnvf(GL_TEXTURE_ENV,  
          GL_TEXTURE_ENV_MODE,  
          GL_REPLACE); //GL_MODULATE
```

Textúra kiválasztása:

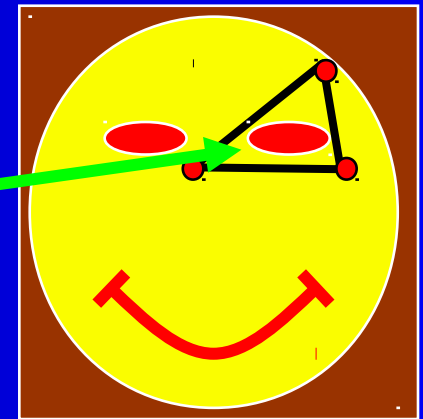
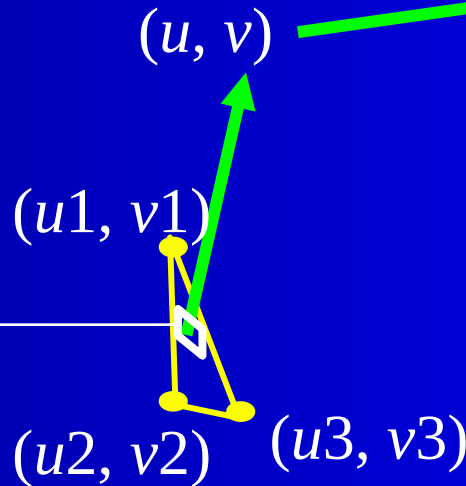
```
glBindTexture(GL_TEXTURE_2D, id);
```

```
glBegin(GL_TRIANGLES);  
glTexCoord2d(u1, v1);  
glVertex3d(x1, y1, z1);  
glTexCoord2d(u2, v2);  
glVertex3d(x2, y2, z2);  
...
```

Textúra leképzés



Interpolációval:



Textúra objektum
a grafikus kártya
memóriájában

Ne az interpolált szín legyen a pixel szín:

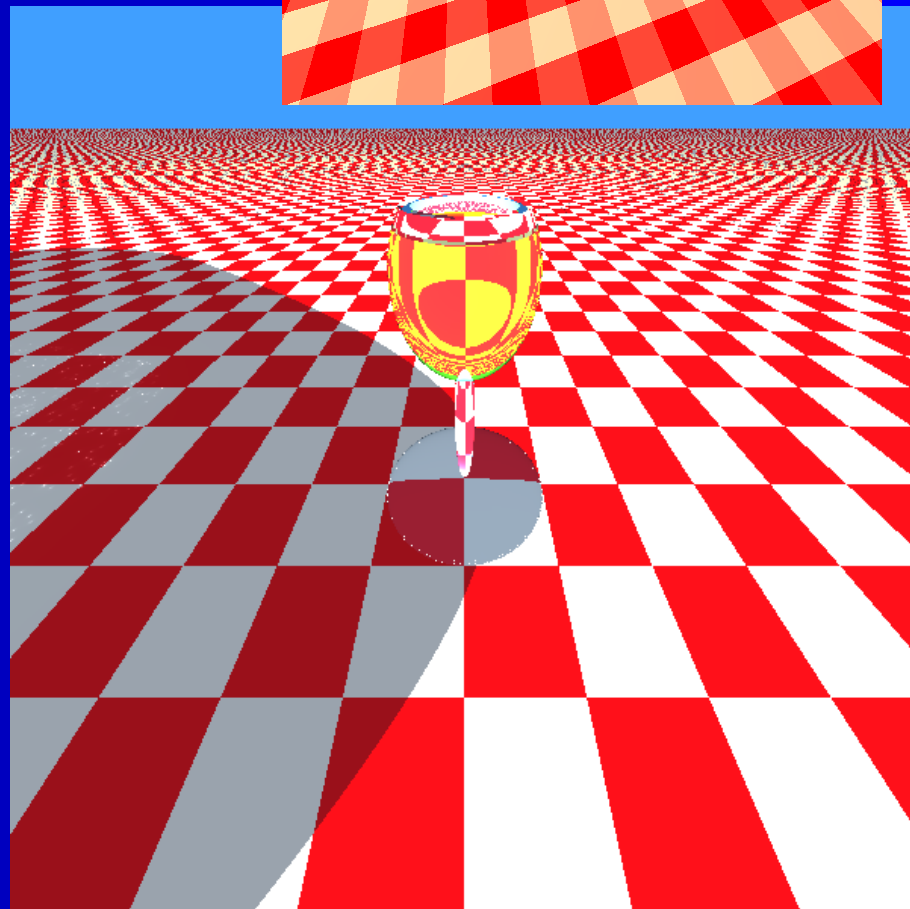
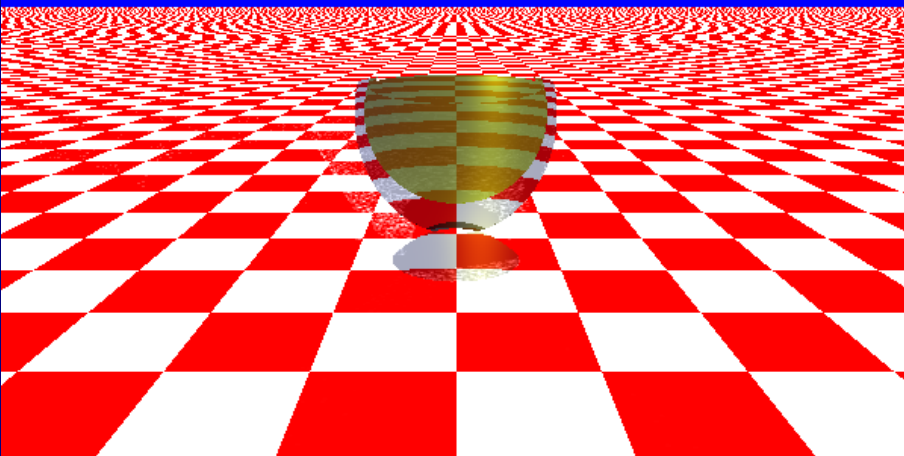
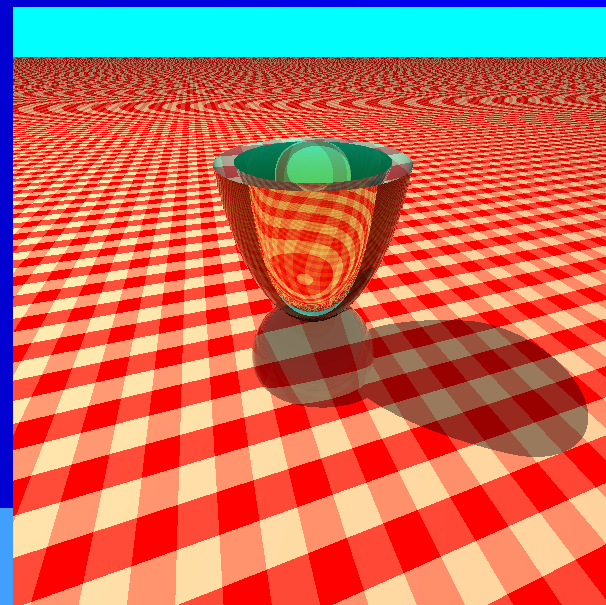
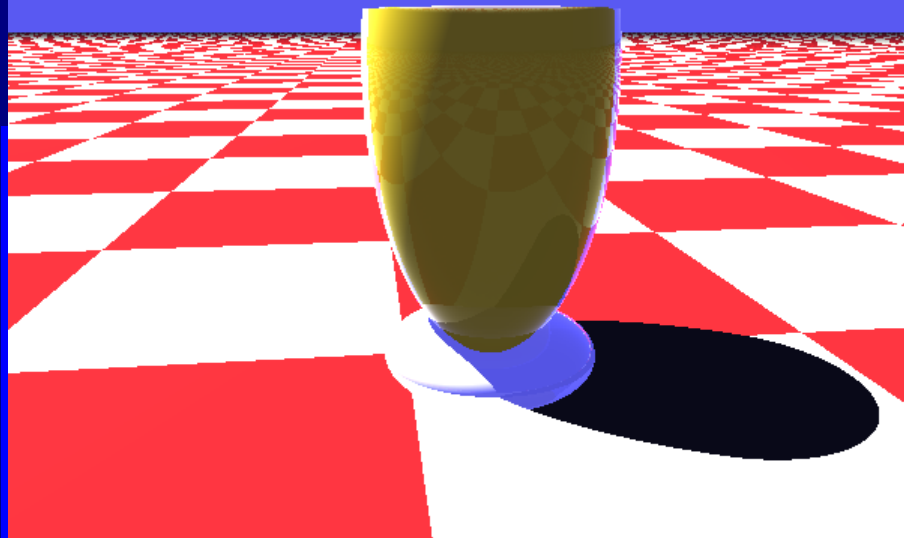
```
glEnable(GL_TEXTURE_2D);  
glTexEnvf(GL_TEXTURE_ENV,  
          GL_TEXTURE_ENV_MODE,  
          GL_REPLACE); //GL_MODULATE
```

Textúra kiválasztása:

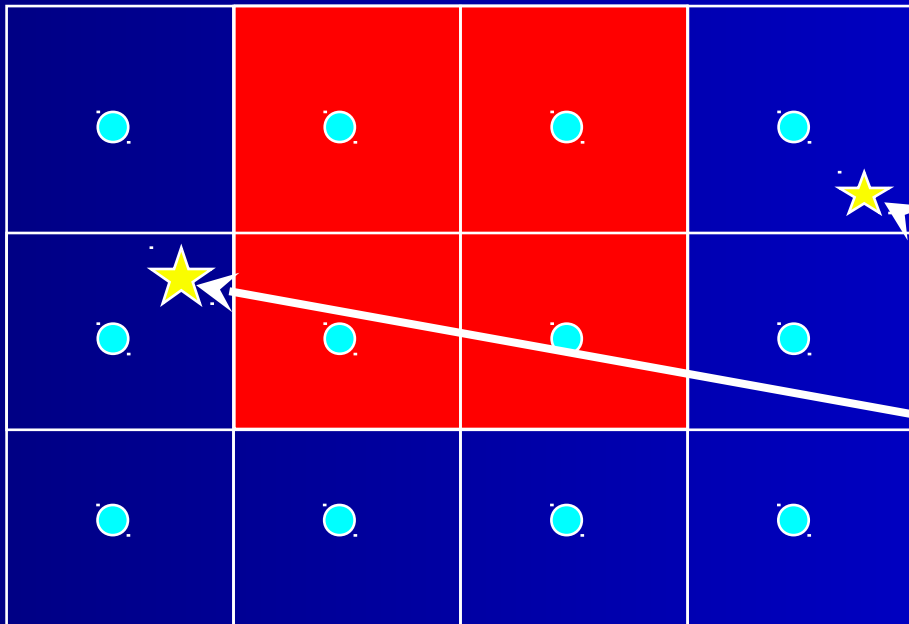
```
glBindTexture(GL_TEXTURE_2D, id);
```

```
glBegin(GL_TRIANGLES);  
glTexCoord2d(u1, v1);  
glVertex3d(x1, y1, z1);  
glTexCoord2d(u2, v2);  
glVertex3d(x2, y2, z2);  
...
```

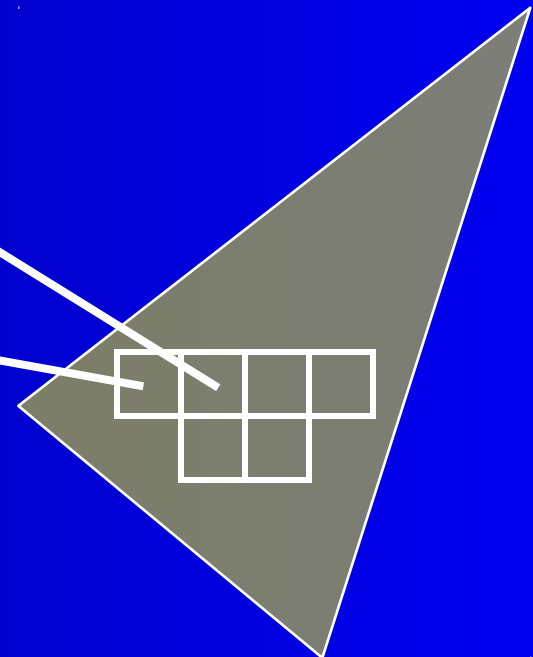
Aliasing



Textúra szűrés

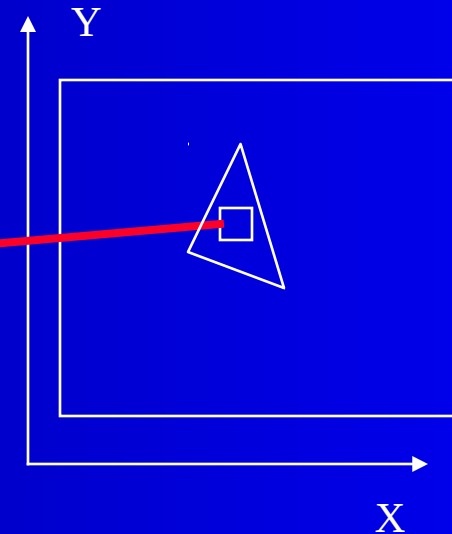
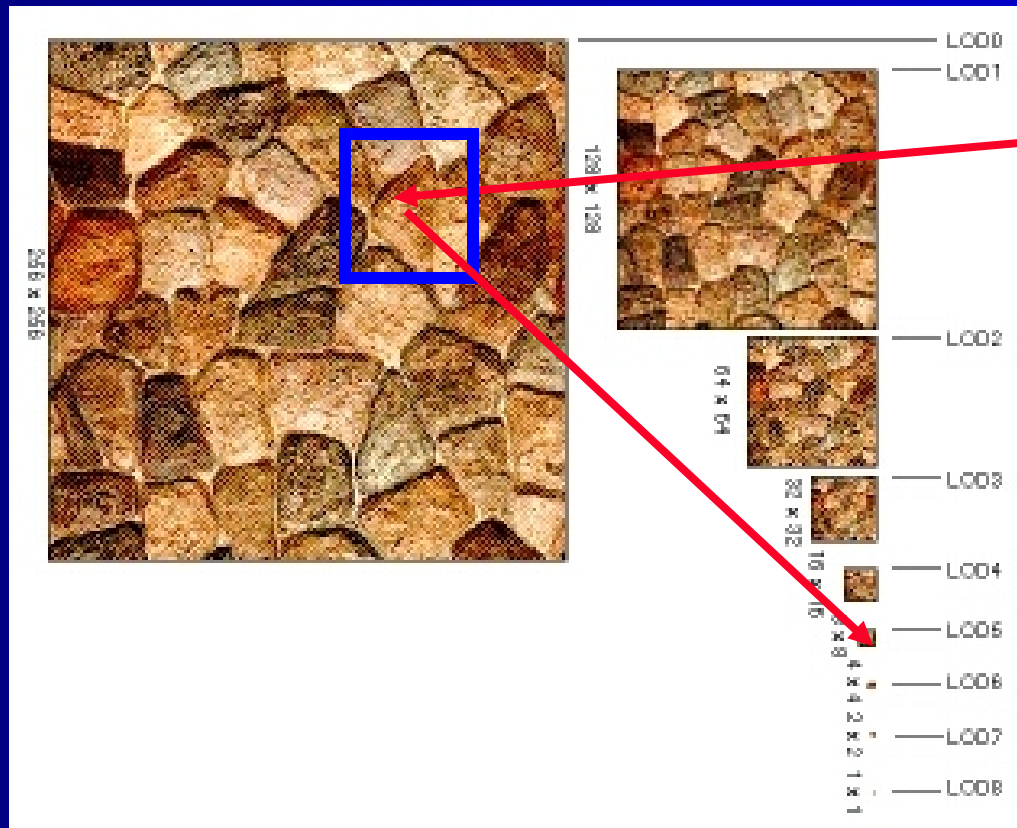


Textúra tér



Képtér

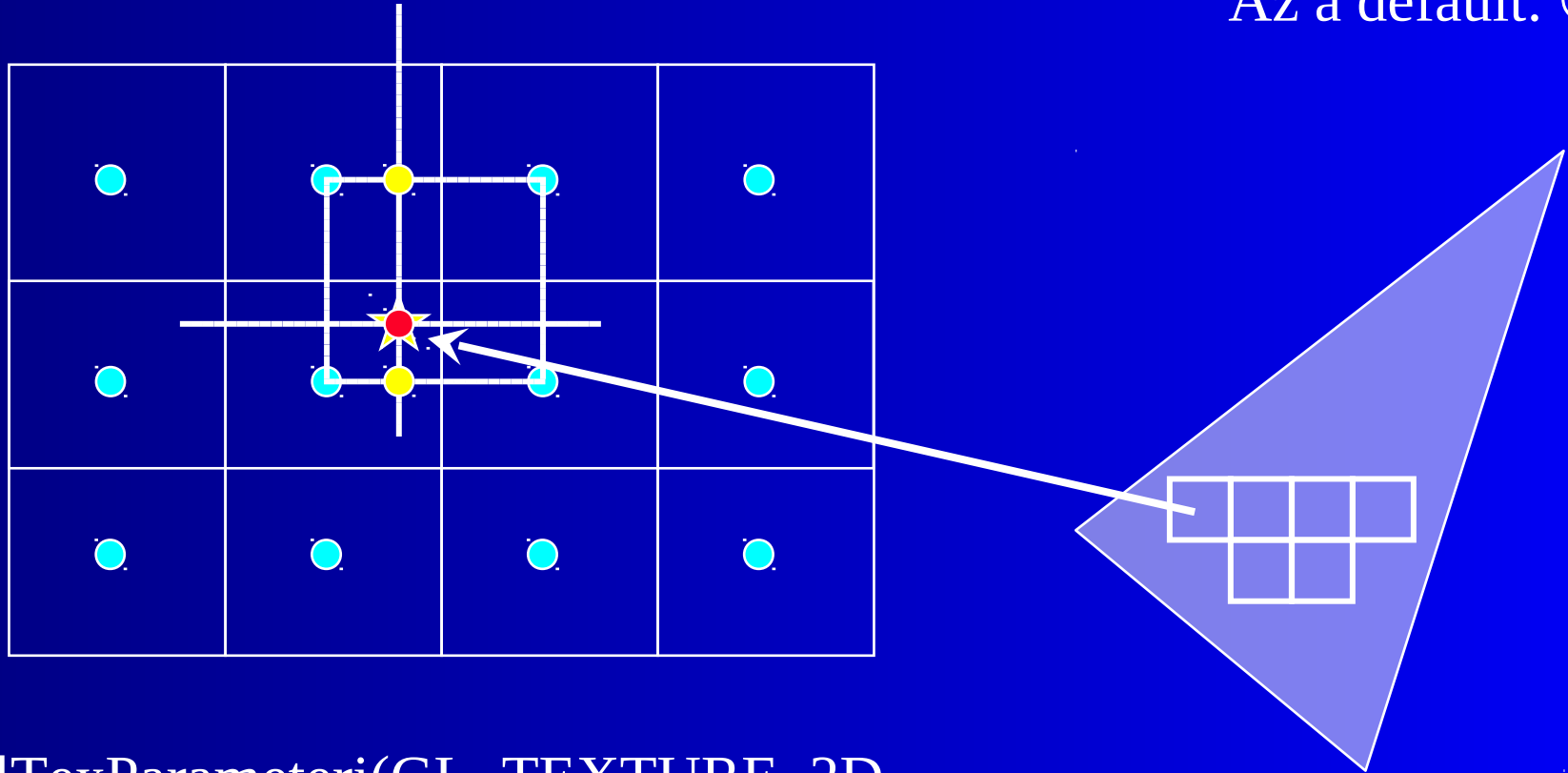
Mip-map (multum in parvo)



Bi-linear textúra szűrés

Mip-map is van: 😊

Az a default: ☹️



```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

Textúra objektum létrehozása

```
unsigned int texids;
```

```
glGenTextures(1, &texids);
```

```
glBindTexture( GL_TEXTURE_2D, texids );
```

```
int level = 0, border = 0, width = 256, height = 256; // 2 hatvány !!!
```

```
unsigned char image[256*256*3]; // Feltöltés BMP, TGA, JPG fájlból.
```

```
glTexImage2D(GL_TEXTURE_2D,  
             level, GL_RGB, width, height, border,  
             GL_RGB, GL_UNSIGNED_BYTE, &image[0]);
```

```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

```
glTexEnvf(GL_TEXTURE_ENV,  
          GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

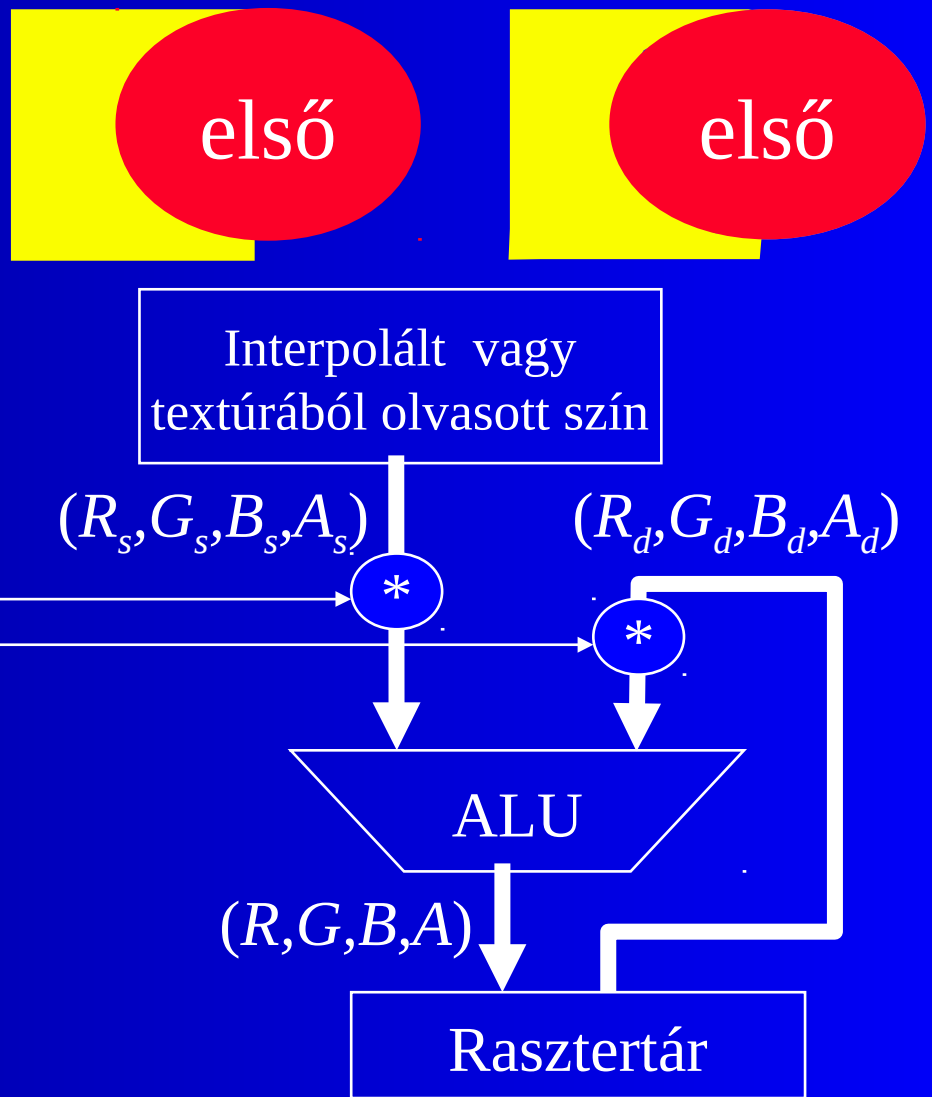
Átlátszóság: Sorrend számít!

```
glEnable(GL_BLEND);
```

```
glBlendFunc(  
    GL_SRC_ALPHA, _____  
    GL_ONE_MINUS_SRC_ALPHA  
);
```

```
glBegin( ... );  
    Geometria ...  
glEnd();
```

```
glDisable(GL_BLEND);
```

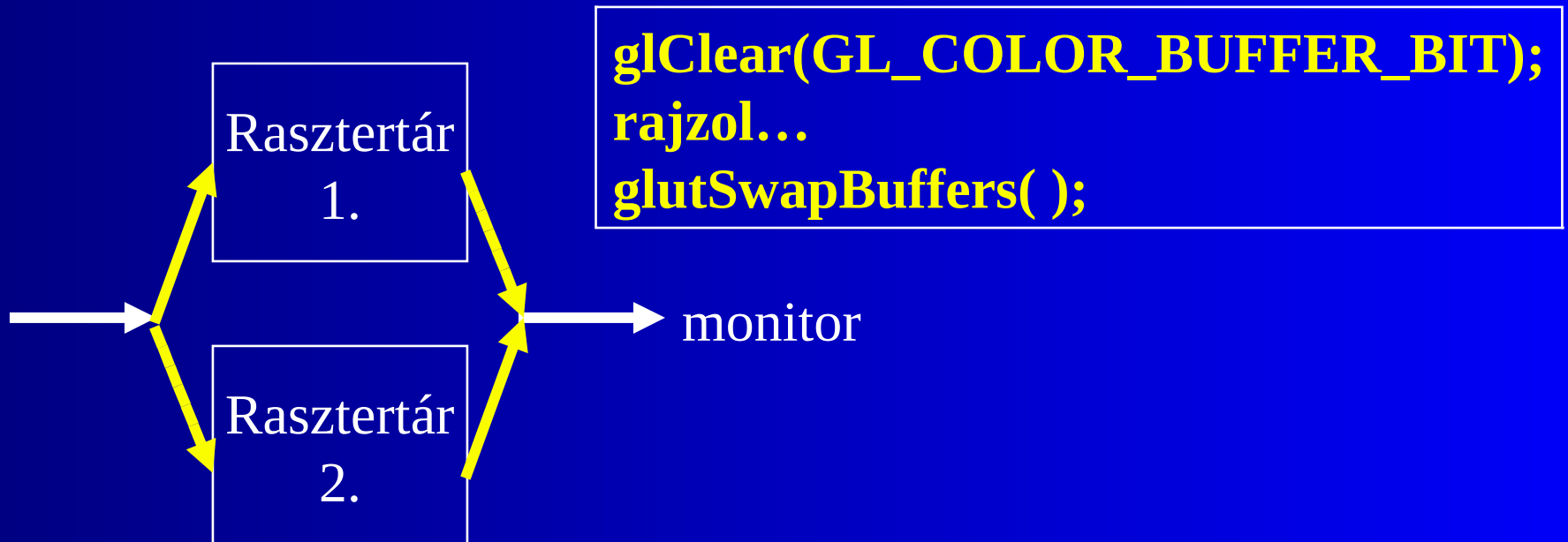


$$(R, G, B, A) = (R_s A_s + R_d (1 - A_s), G_s A_s + G_d (1 - A_s), B_s A_s + B_d (1 - A_s), A_s A_s + A_d (1 - A_s))$$

Dupla buffer animációhoz

Inicializálás:

```
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
```



Példa: Zászló

Geometria

$$x(u,v) = u \cdot W$$

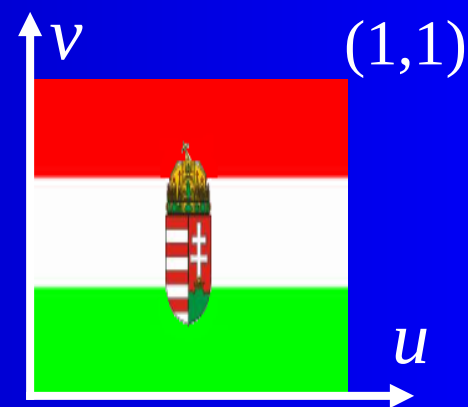
$$y(u,v) = v \cdot H$$

$$z(u,v) = \sin(K \cdot u \cdot \text{PI} + \text{phase}(t)) \cdot D$$

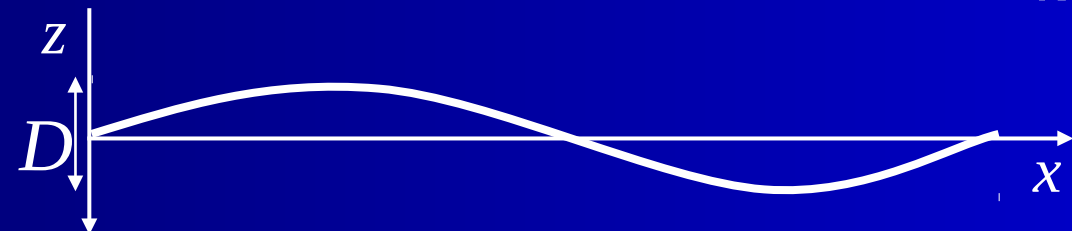


Textúra

Paraméterezés
triviális mivel
paraméteres felület.



Textúra kép



main

```
int main( int argc, char * argv[ ] ) {  
    glutInit(&argc, argv);  
    glutInitWindowSize(600, 600); // app window creation  
    glutInitDisplayMode(GLUT_RGB |  
                        GLUT_DOUBLE |  
                        GLUT_DEPTH);  
    glutCreateWindow("Waving flag...");  
  


---

  
    onInit( ); // Initialization  
  


---

  
    glutDisplayFunc( DrawFlag ); // Display callback registration  
    glutIdleFunc( AnimateFlag ); // Idle callback registration  
    glutReshapeFunc( onReshape ); // Reshape callback registration  
  


---

  
    glutMainLoop();  
}
```

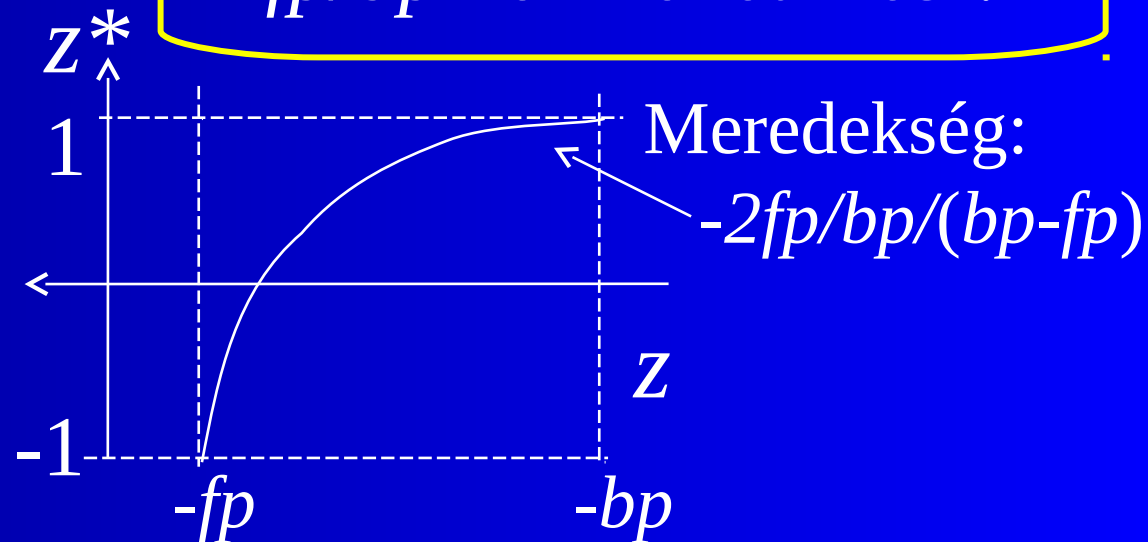
onReshape

```
void onReshape(int winWidth, int winHeight) {  
    glViewport(0, 0, winWidth, winHeight);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity( );  
    gluPerspective(54, (float)winWidth/(float)winHeight, 1, 100);  
}
```

$$z^* = -\alpha - \beta/z$$

$$\alpha = -(fp + bp)/(bp - fp)$$

$$\beta = -2fp \cdot bp/(bp - fp)$$



void onInit () {

```
glEnable(GL_DEPTH_TEST);  
glDisable(GL_CULL_FACE);  
glShadeModel(GL_SMOOTH);  
  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
gluLookAt(0, 0, 0, 0, 0, -1, 0, 1, 0);  
glTranslatef(-15, -10, -50);
```

onInit

ModelView
transzformáció

```
glEnable(GL_TEXTURE_2D); // texturing is on  
unsigned int texture, width, height;  
glGenTextures(1, &texture); // id generation  
glBindTexture(GL_TEXTURE_2D, texture); // binding  
static unsigned char bitmap[3*maxwidth*maxheight]; // texture on CPU  
LoadBmp("flag.bmp", &bitmap[0], &width, &height); // your job! Power of 2?  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,  
             GL_RGB, GL_UNSIGNED_BYTE, bitmap); //Texture->OpenGL  
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
}
```

Texture

DrawFlag = tessellation+passing to OpenGL

```
void FlagPoint(float u, float v) {
```

```
    glTexCoord2f(u, v);
```

```
    glVertex3f( u*W, v*H, sin(u*K*M_PI + phase)*D );
```

```
}           $x(u,v)$   $y(u,v)$   $z(u,v)$ 
```

```
void DrawFlag() {
```

```
    glClearColor(0, 0, 0, 0);
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

```
    glBegin(GL_QUADS);
```

```
    for (float u = 0; u < 1; u+=1.0/N) for (float v = 0; v < 1; v+=1.0/M) {
```

```
        FlagPoint(u, v);
```

```
        FlagPoint(u+1.0/N, v);
```

```
        FlagPoint(u+1.0/N, v+1.0/M);
```

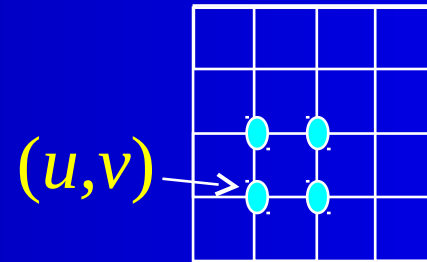
```
        FlagPoint(u, v+1.0/M);
```

```
    }
```

```
    glEnd();
```

```
    glutSwapBuffers( );
```

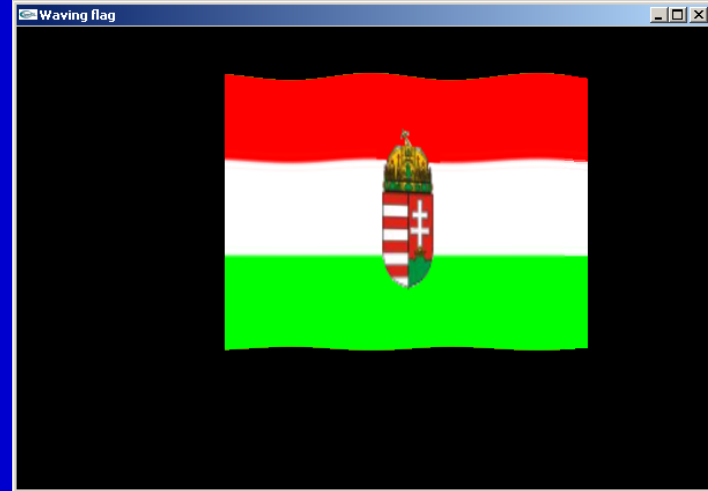
```
}
```



AnimateFlag

Egy rossz megoldás ...

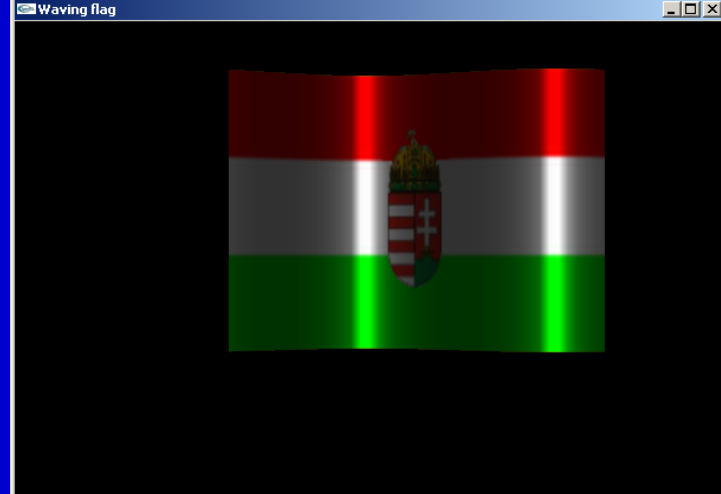
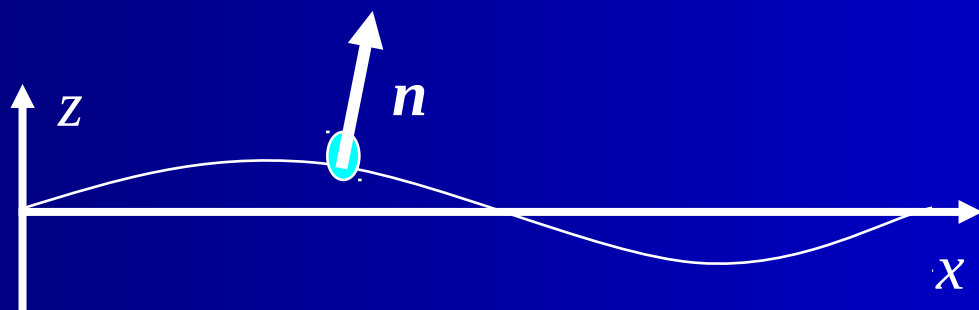
```
float phase = 0;  
void AnimateFlag() {  
    phase += 0.3;  
    DrawFlag(); // vagy glutPostRedisplay();  
}
```



A jó megoldás ...

```
float phase = 0;  
void AnimateFlag() {  
    long ctime = glutGet(GLUT_ELAPSED_TIME);  
    phase = 0.2 * ctime;  
    DrawFlag(); // vagy glutPostRedisplay();  
}
```

Zászló textúra és illumináció



$$\mathbf{r}(u,v) = [u \cdot W, \quad v \cdot H, \quad \sin(K \cdot u \cdot \text{PI} + \text{phase}) \cdot D] \quad]$$

$$\begin{matrix} \partial \mathbf{r} / \partial u = \begin{bmatrix} i & j & k \\ W, & 0, & K \cdot \text{PI} \cdot \cos(K \cdot u \cdot \text{PI} + \text{phase}) \cdot D \\ 0, & H, & 0 \end{bmatrix} \\ \partial \mathbf{r} / \partial v = \end{matrix} \quad]$$

$$\mathbf{n}(u,v) = \partial \mathbf{r} / \partial u \times \partial \mathbf{r} / \partial v = [-K \cdot \text{PI} \cdot H \cdot \cos(K \cdot u \cdot \text{PI} + \text{phase}) \cdot D, 0, W \cdot H]$$

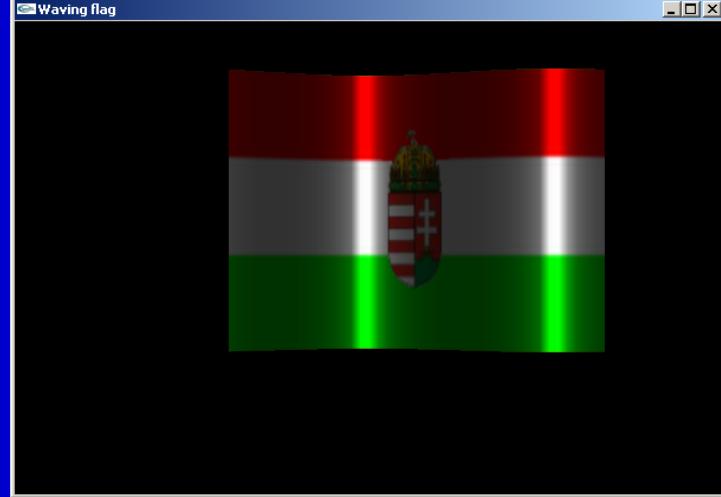
Fényforrás és anyagtulajdonságok

```
float pos[4] = {0, 0, 1, 0}; // irányfényforrás  
float Ia[4] = {0, 0, 0, 1}, Id[4] = {0.5, 0.5, 0.5, 1}, Is[4] = {2, 2, 2, 1};
```

```
glLightfv(GL_LIGHT0, GL_AMBIENT, Ia);  
glLightfv(GL_LIGHT0, GL_DIFFUSE, Id);  
glLightfv(GL_LIGHT0, GL_SPECULAR, Is);  
glLightfv(GL_LIGHT0, GL_POSITION, pos);  
glEnable(GL_LIGHT0);
```

After gluLookAt

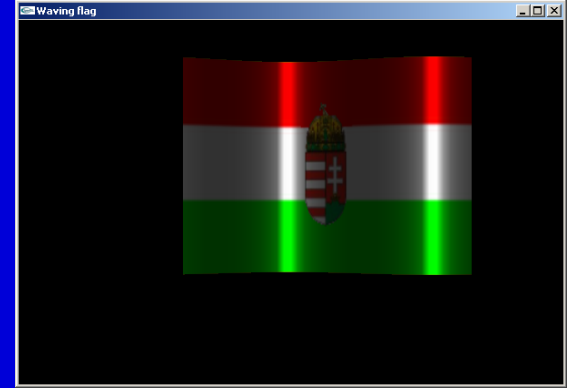
```
float col[4] = {1, 1, 1, 1};  
glMaterialfv( GL_FRONT, GL_DIFFUSE, col);  
glMaterialfv( GL_FRONT, GL_SPECULAR, col);  
glMateriali( GL_FRONT, GL_SHININESS, 20);
```



Color =
 $ka * Ia +$
 $kd * Id \cdot (\mathbf{N} \cdot \mathbf{L})^+ +$
 $ks * Is \cdot ((\mathbf{N} \cdot \mathbf{H})^+)^{\text{shininess}}$

ColorVertex =
 $Ia +$
 $Id \cdot (\mathbf{N} \cdot \mathbf{L})^+ +$
 $Is \cdot ((\mathbf{N} \cdot \mathbf{H})^+)^{\text{shininess}}$

ColorPixel =
 $k * \text{InterpolColorVertex}$



```

void DrawFlag( ) {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
    glTexEnvf(GL_TEXTURE_ENV,
              GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glEnable( GL_LIGHTING ); // illumination computation is on
    glEnable( GL_NORMALIZE ); // make the normal unit vector

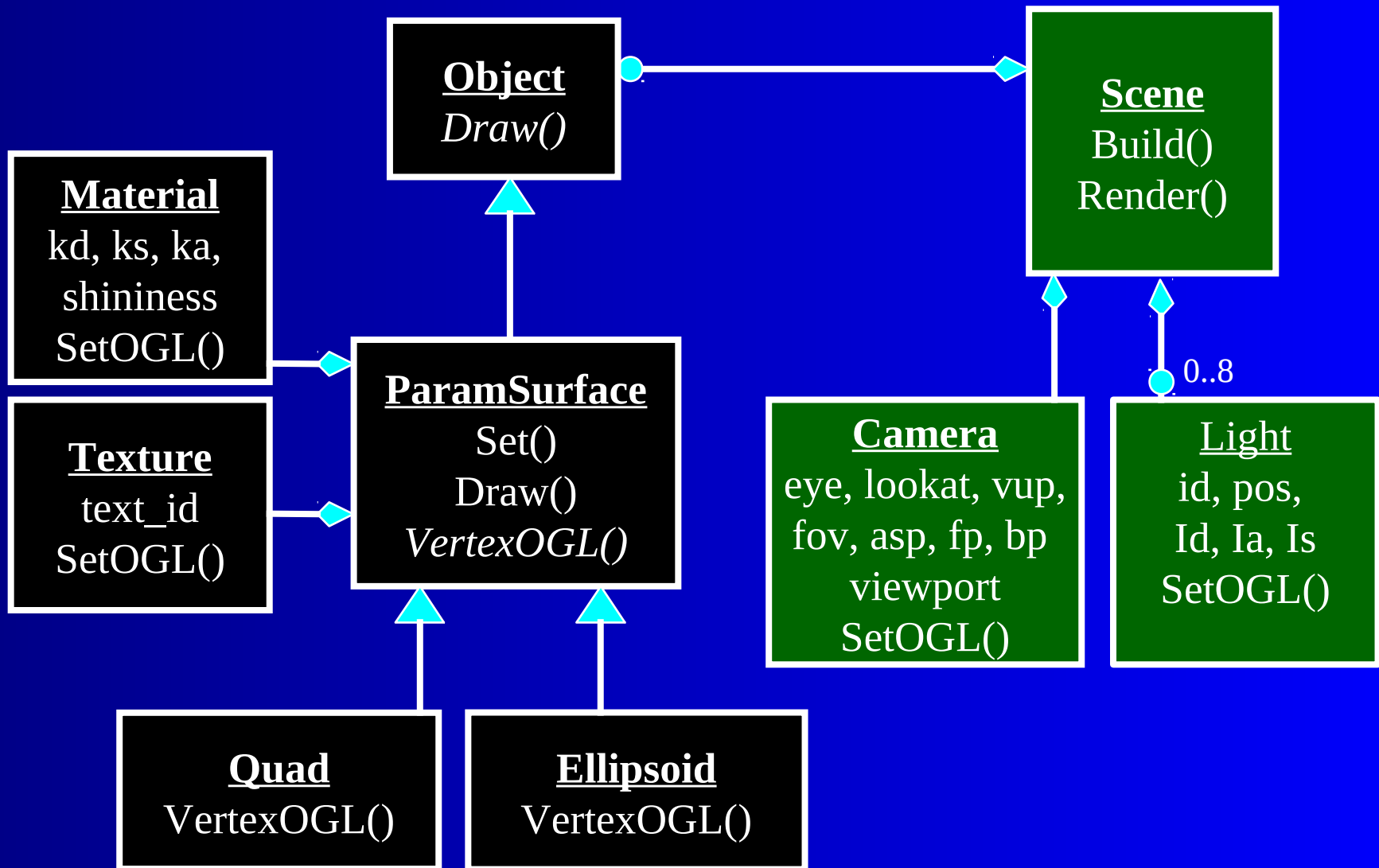
    glBegin(GL_QUADS);
    for (float u = 0; u < 1; u+=1.0/N) for (float v = 0; v < 1; v+=1.0/M) {
        glTexCoord2f(u, v);
        glNormal3f(-K*M_PI*cos(u*K*M_PI+phase)*D, 0, W); //H ignored
        glVertex3f(u*W, v*H, sin(u*K*M_PI + phase)*D);
        ...
    }
    glEnd();
    glutSwapBuffers( );
}

```

$\mathbf{n}(u,v)$

$\mathbf{r}(u,v)$

Osztálydiagram



OS: am

```
camera -> SetOGL( );  
for each light: SetOGL( );  
for each object: Draw( );  
glutSwapBuffers( );
```

Object
Draw()

Scene
Build()
Render()

```
glViewport(viewport.left, viewport.bottom, viewport.width, viewport.height);  
glMatrixMode(GL_PROJECTION);  
glLoadIdentity( );  
gluPerspective(fov, asp, fp, bp);  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity( );  
gluLookAt(eye.x, eye.y, eye.z, lookat.x, lookat.y, lookat.z, vup.x, vup.y, vup.z);
```

text_id
SetOGL()

Draw()
VertexOGL()

Draw()
lookat, vup,
fov, asp, fp, bp
viewport
SetOGL()

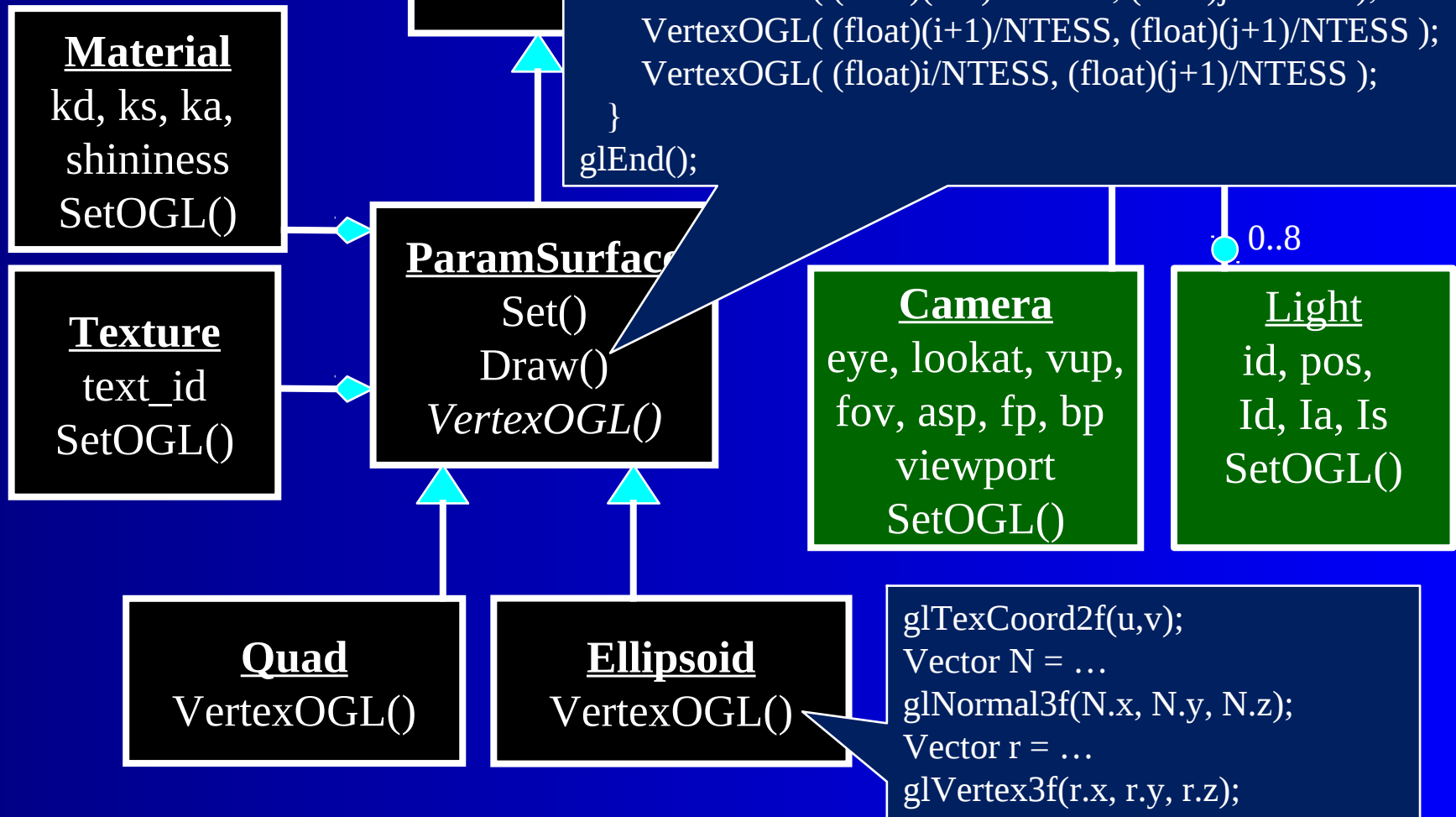
Light
id, pos,
Id, Ia, Is
SetOGL()

Quad
VertexOGL()

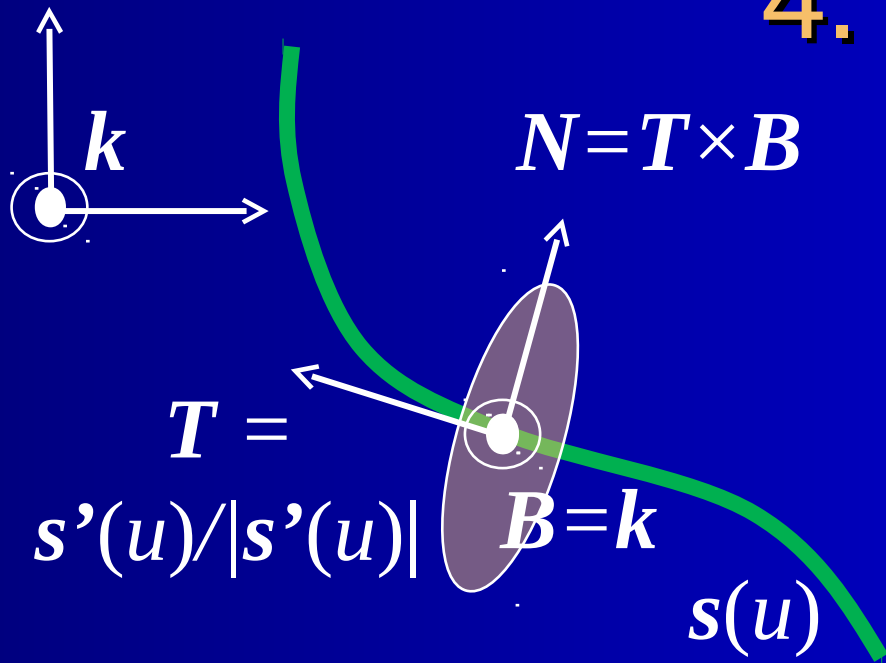
Ellipsoid
VertexOGL()

0..8

Osztály



4. házi

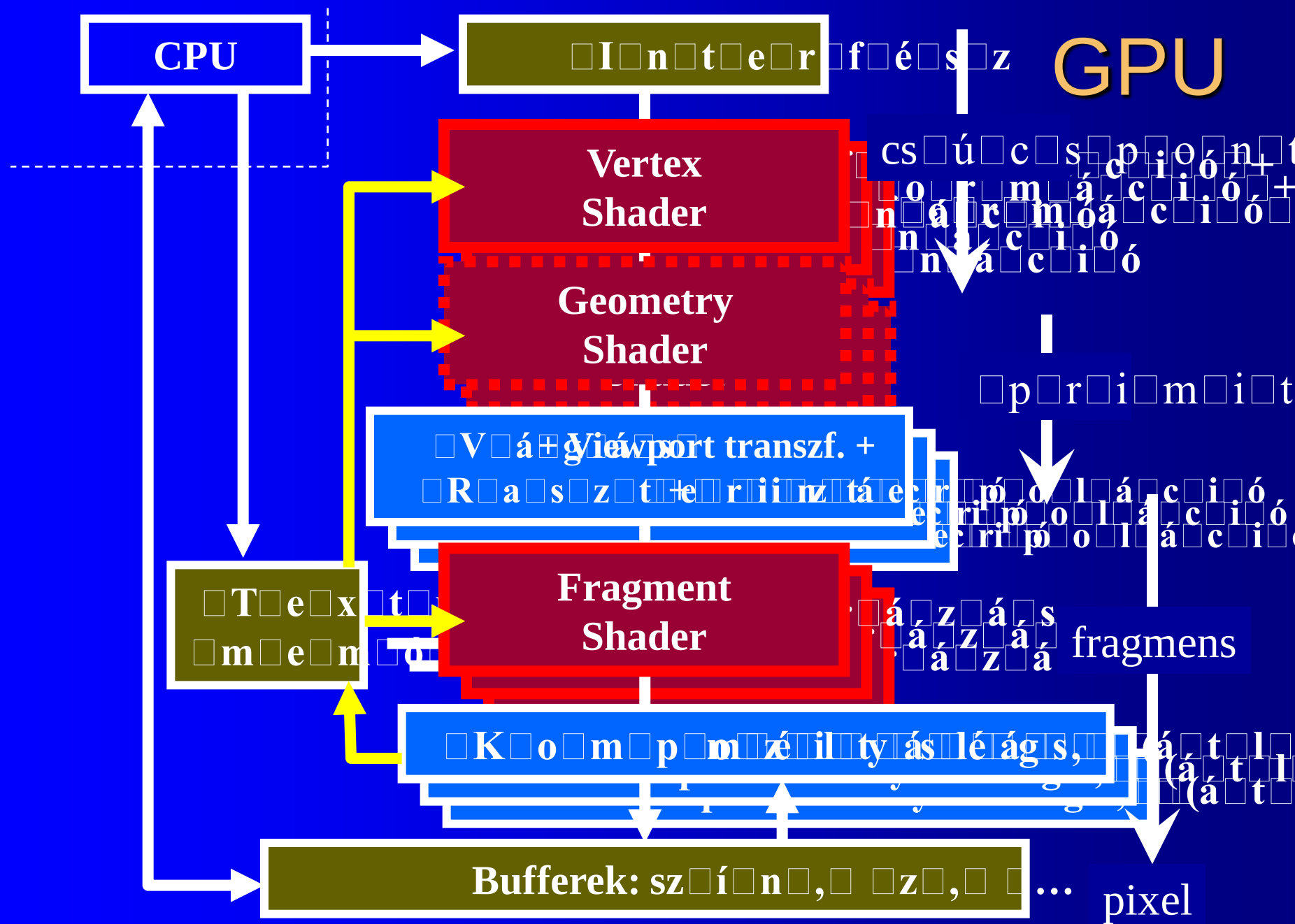


$$r(u,v) = s(u) + B(u)R(u)\cos(v) + N(u)R(u)\sin(v)$$

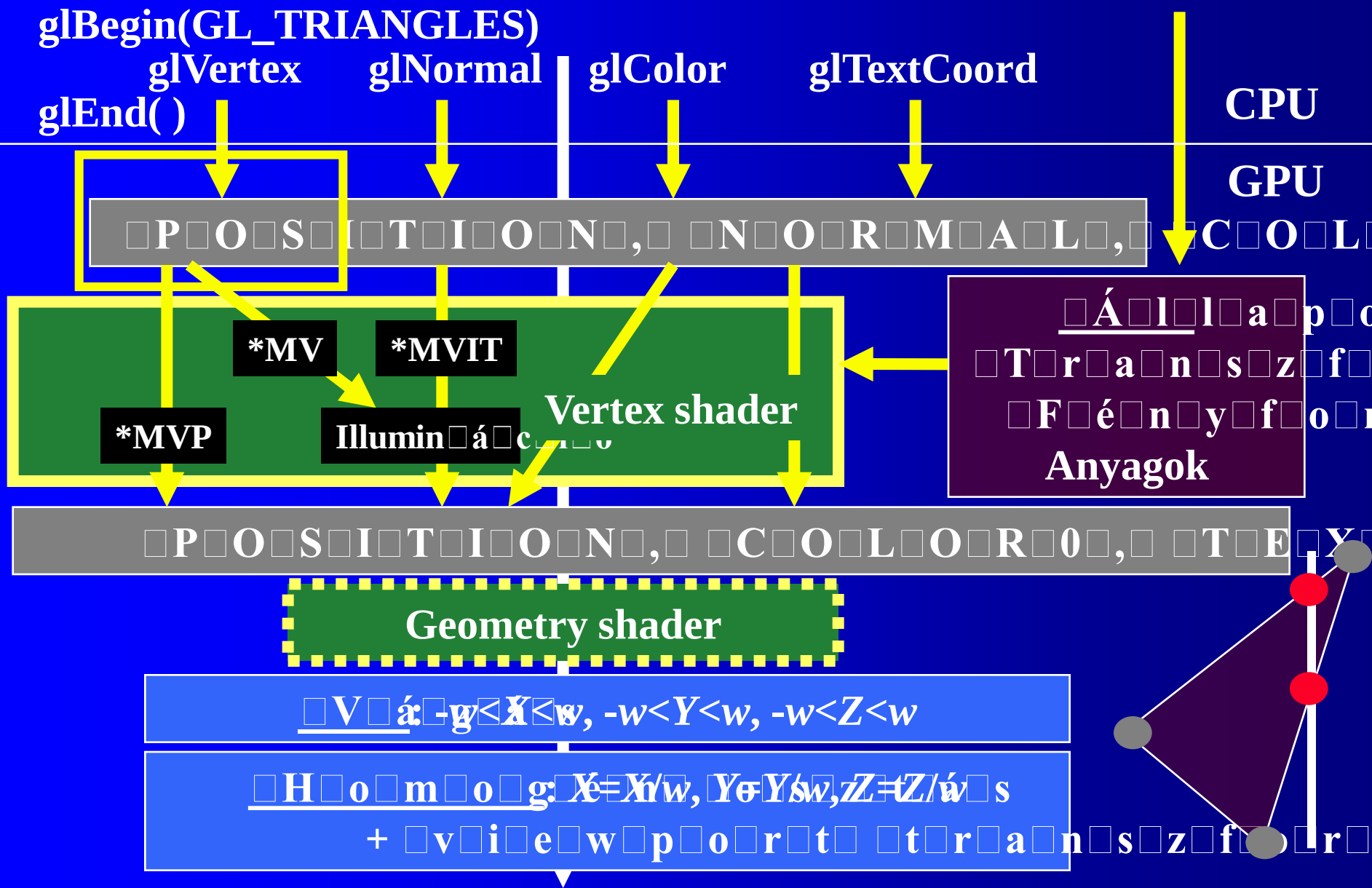


GPU

Szirmay-Kalos László



Vertex shader



“Standard” vertex shader (Cg, Shader Model 3.0)

```
void main(
    in float4 position           : POSITION,
    in float3 normal             : NORMAL,
    in float4 color              : COLOR0,
    in float2 texcoord           : TEXCOORD0,
    uniform float4x4 modelviewproj : state.matrix.mvp,
    out float4 hposition         : POSITION,
    out float4 ocolor           : COLOR0,
    out float2 otexcoord         : TEXCOORD0 )
{
    hposition = mul(modelviewproj, position);
    otexcoord = texcoord;
    ocolor = color;
}
```

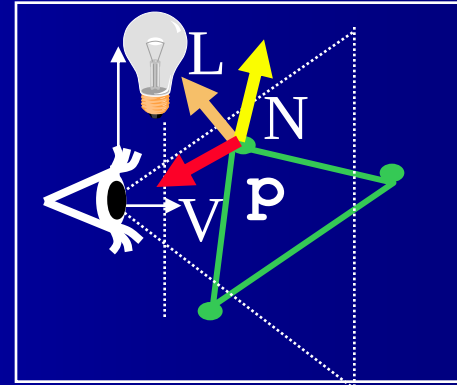
glDisable(GL_LIGHTING);

Mez k e l é r é s e:

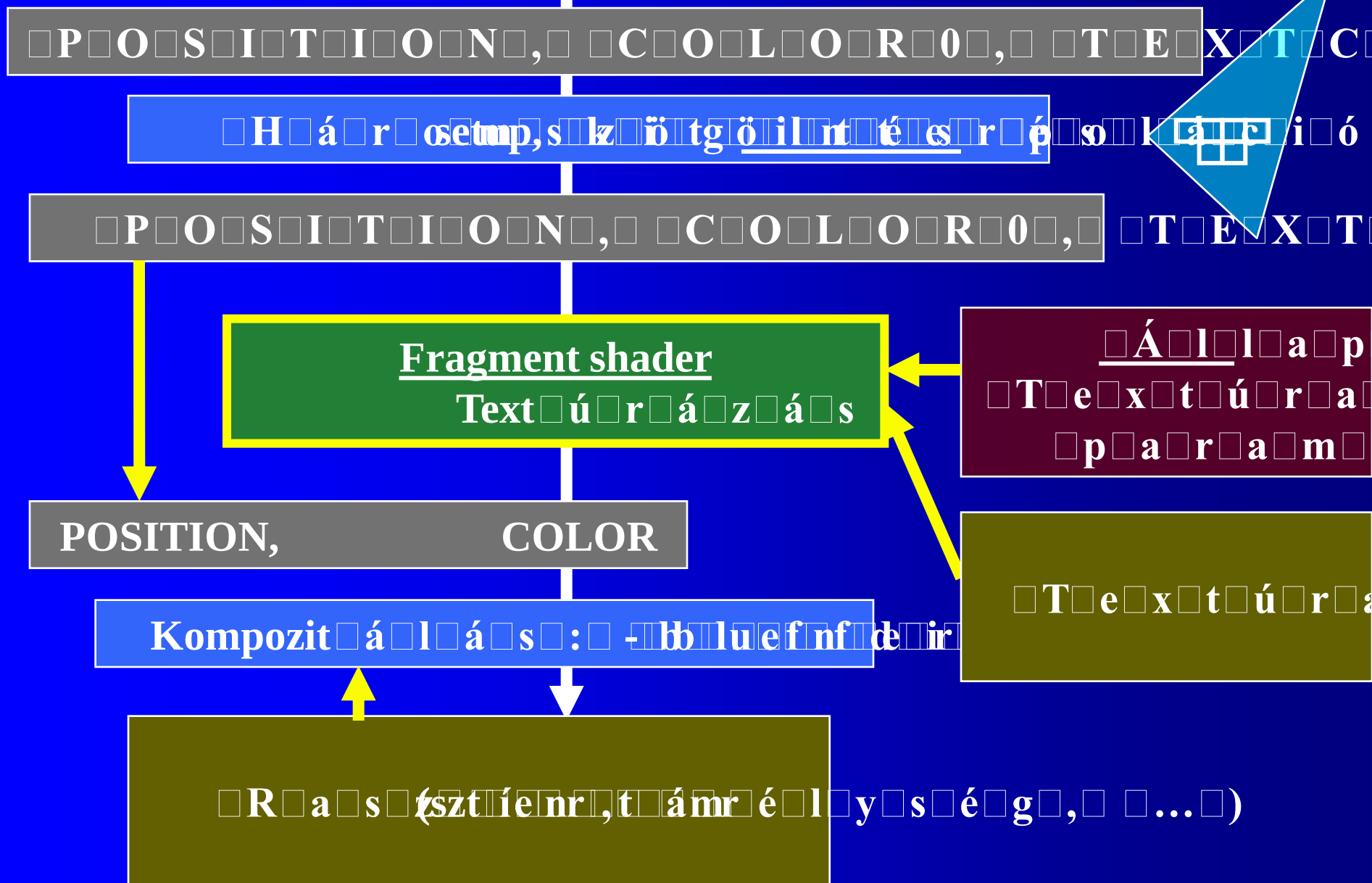
v.x, v.y, v.z, v.w, v.xy, v.wxx
c.r, c.rgb, c.ar, ...

glEnable(GL_LIGHTING);

```
main( in float4 position : POSITION,
      in float4 normal   : NORMAL,
      uniform float4x4 modelview, modelviewIT, modelviewproj, []
      uniform float4 lightpos, Idiff, Iamb, Ispec, []
      uniform float4 em, ka, kd, ks, []
      uniform float shininess,
      out float4 hposition: POSITION,
      out float4 ocolor   : COLOR0
  ) {
    hposition = mul(modelviewproj, position);
    float3 N = mul(modelviewIT, normal).xyz;
    N = normalize(N); // glEnable(GL_NORMALIZE)
    float4 p = mul(modelview, position);
    float3 L = normalize(lightpos.xyz/lightpos.w - p.xyz/p.w);
    float costheta = dot(N, L); if (costheta < 0) costheta = 0;
    float3 V = normalize(-p.xyz);
    float3 H = normalize(L + V);
    float cosdelta = dot(N, H); if (cosdelta < 0) cosdelta = 0;
    ocolor = em + Iamb * ka + Idiff * kd * costheta +
              Ispec * ks * pow(cosdelta, shininess);
  }
```



Pixel shader és környezet



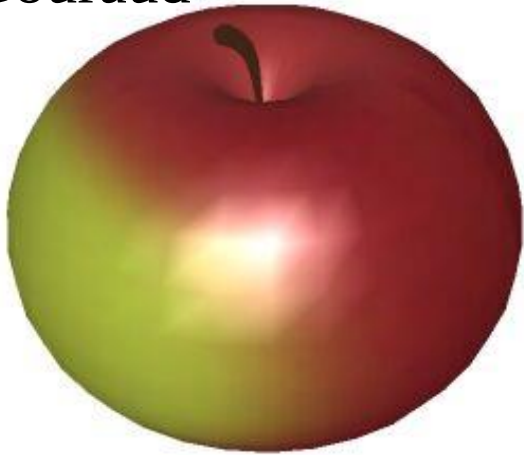
“a6d” pixel shader

```
void main( in float4 color : COLOR0,  
           out float4 ocolor : COLOR )  
{  
    ocolor = color;  
}                                     glDisable(GL_TEXTURE_2D);
```

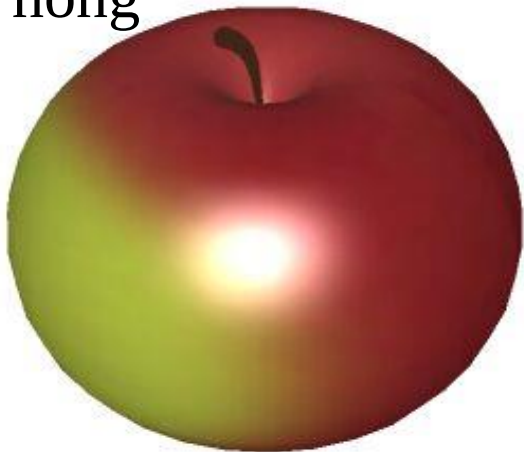
```
void main( in float2 texcoord      : TEXCOORD0,  
           in float3 color        : COLOR0,  
           uniform sampler2D texture_map,  
           out float4 ocolor      : COLOR )  
{  
    ocolor = tex2D(texture_map, texcoord) * color;  
}                                     glEnable(GL_TEXTURE_2D);  
glTexEnv(GL_TEXTURE_ENV,  
         GL_TEXTURE_ENV_MODE, GL_MODULATE);
```


Gouraud helyett Phong árnyalás Per-vertex helyett per-pixel

Gouraud



Phong



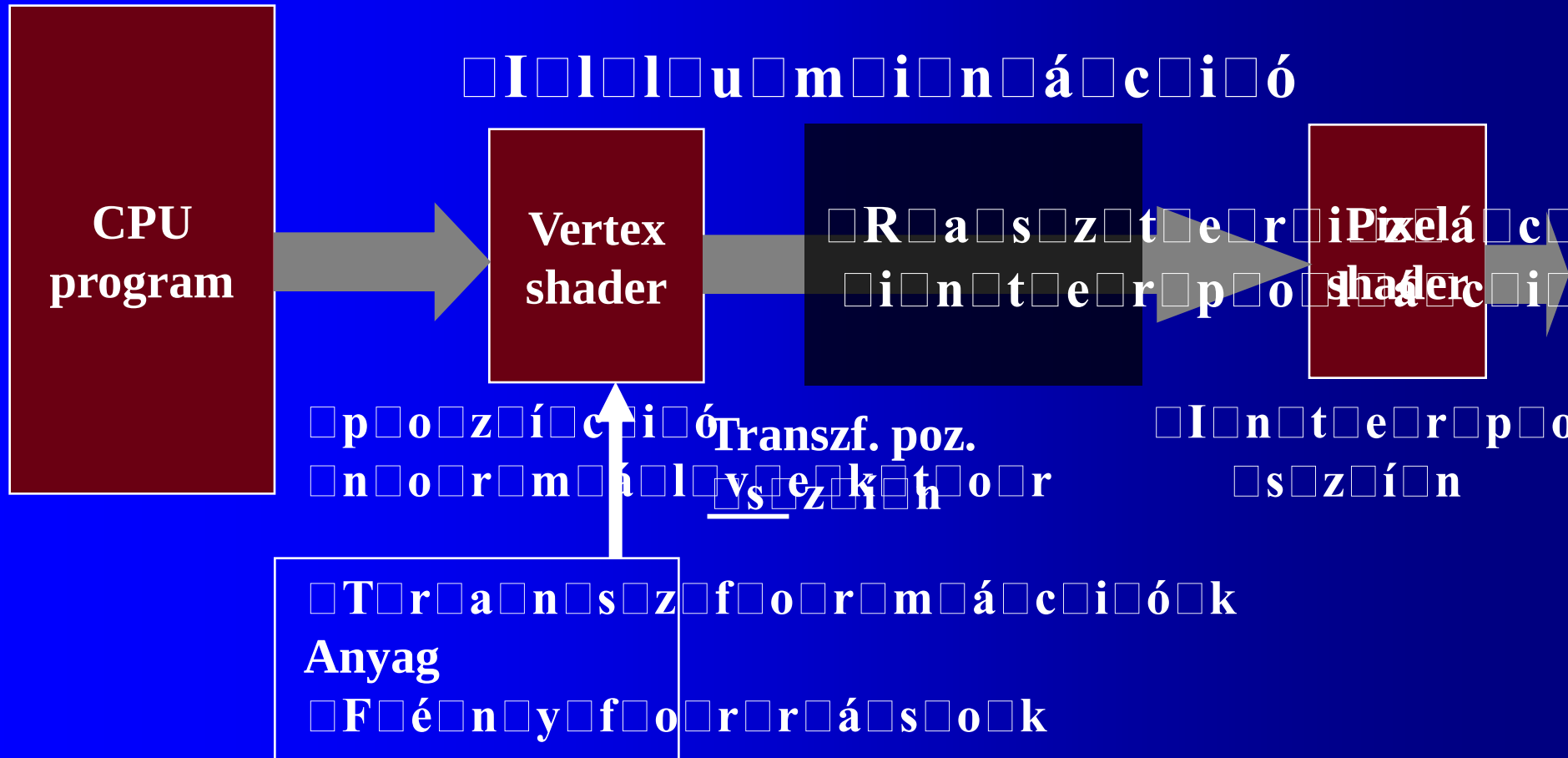
Gouraud



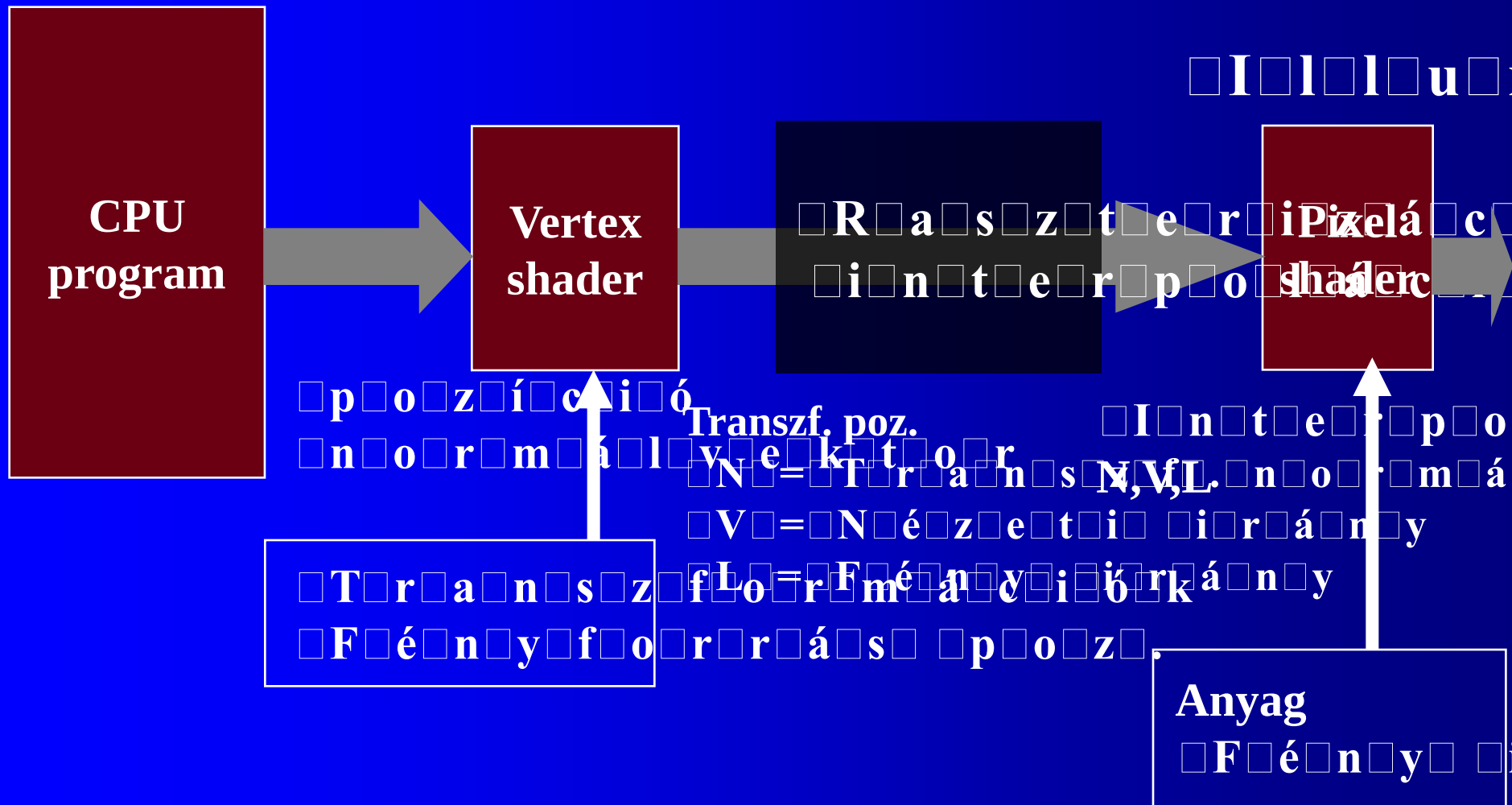
Phong



Gouraud (per-vertex) árnyéklás



Phong (per-pixel) árnyékolás



Programok (labahajtos megoldás)

● .cpp CPU program:

- Shader environment létre
- GPU kernelok beállítás
- Vertex/fragment program betöltés és for
- Vertex/fragment shader program
- Vertex/fragment shader program
- Uniform vertex/fragment
- Uniform vertex/fragment
- Változózó input változó

● .cg vertex program

- Fragment shader program válto

● .cg fragment program

- Színi kiíratás

VerTeX

de #.

Fragmentprogramok létrehozása

```
CGprofile fragmentProf = cgGLGetLatestProfile(CG_GL_FRAGMENT);  
cgGLEnableProfile(fragmentProf);
```

```
CGprogram fragmentProgram = cgCreateProgramFromFile(  
    shaderContext,  
    CG_SOURCE, "fragment.glsl",  
    fragmentProf,  
    NULL, NULL);
```

```
cgGLLoadProgram(fragmentProgram); // GPU-ra töltés  
cgGLBindProgram(fragmentProgram); // ez a program fusson
```

// fragmens program uniform paraméterek

```
Shine = cgGetNamedParameter(fragmentProgram, "shininess");  
Kd = cgGetNamedParameter(fragmentProgram, "kd");  
Ks = cgGetNamedParameter(fragmentProgram, "ks");  
LightInt = cgGetNamedParameter(fragmentProgram, "Lightint");
```

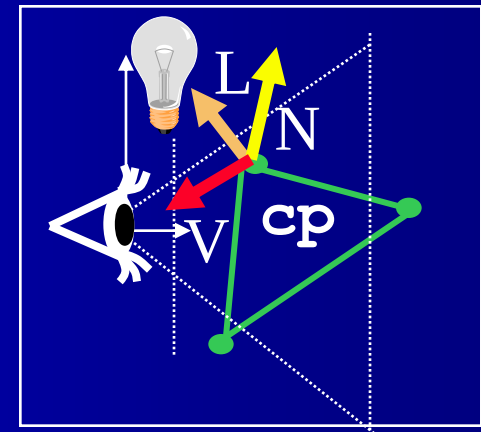
OpenGL inicializálás

CPU program - OpenGL display

```
void Display( ) {  
    glLoadIdentity(); // átlalapot (paraméterleírkéihez)  
    gluLookAt(0, 0, -10, 0, 0, 0, 0, 1, 0);  
    glRotatef(angle, 0, 1, 0);  
  
    // explicit uniform paraméterek beállítása  
    cgGLSetParameter4f(LightPos, 10, 20, 30, 1); // clightpos  
    cgGLSetParameter3f(LightInt, 1, 1, 1);      // lightint  
    cgGLSetParameter1f(Shine, 40);              // shininess  
    cgGLSetParameter3f(Kd, 1, 0.8, 0.2);        // kd  
    cgGLSetParameter3f(Ks, 2, 2, 2);            // ks  
  
    // választó terek, a PASS rama  
    glBegin( GL_TRIANGLES );  
    for( ; ; ) {  
        glNormal3f(nx, ny, nz); // NORMAL regiszter  
        glVertex3f(x, y, z);    // POSITION regiszter + trigger  
    }  
    glEnd();  
}
```

Phong árnyalás: vertex shader:

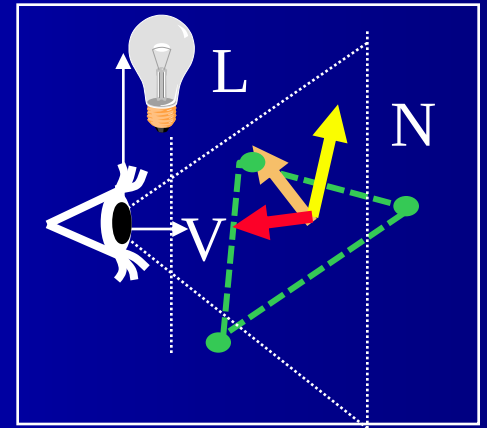
```
void main(  
    in float4 position      : POSITION,  
    in float4 normal        : NORMAL,  
    uniform float4x4 MVP    : state.matrix.mvp,  
    uniform float4x4 MV     : state.matrix.modelview,  
    uniform float4x4 MVIT   : state.matrix.modelview.invtrans,  
    uniform float4   c_l_i_g_h_t_p_o_s, // / / fényforrás  
    out float4 hposition    : POSITION,  
    out float3 cnormal      : TEXCOORD0,  
    out float3 cview        : TEXCOORD1,  
    out float3 clight       : TEXCOORD2  
)  
{  
    hposition = mul(MVP, position);  
    float4 cp = mul(MV, position);  
    cnormal = mul(MVIT, normal).xyz;  
    clight = c_l_i_g_h_t_p_o_s.xyz * cp.w - cp.xyz * c_l_i_g_h_t_p_o_s.w;  
    cview   = -cp.xyz;  
}
```



Phong árnyalás: fragment shader

```
void main(      in float3 N          : TEXCOORD0,
                in float3 V          : TEXCOORD1,
                in float3 L          : TEXCOORD2,
                uniform float3 lightint,
                uniform float shininess,
                uniform float3 kd,
                uniform float3 ks,
                out float3 ocolor     : COLOR )
{
    N = normalize( N );
    V = normalize( V );
    L = normalize( L );

    float3 H = normalize(V + L);
    float costheta = max(dot(N, L), 0);
    float cosdelta = max(dot(N, H), 0);
    ocolor = lightint *
              (kd * costheta + ks * pow(cosdelta, shininess));
}
```



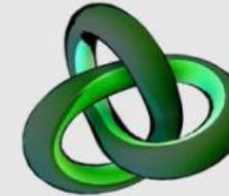
NPR

```
void main(      in float3 N          : TEXCOORD0,
                in float3 V          : TEXCOORD1,
                in float3 L          : TEXCOORD2,
                uniform float3 kd,
                out float3 ocolor     : COLOR )
{
    N = normalize( N );
    V = normalize( V );
    L = normalize( L );

    float costheta = dot(N, L);
    float y = (costheta > 0.5) ? 1 : 0.5;
    if (abs(dot(N, V)) < 0.2) ocolor = float3(0, 0, 0);
    else                      ocolor = y * kd;
}
```

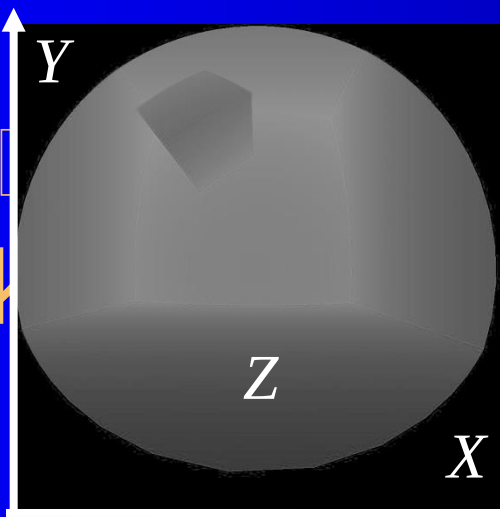


NPR



Lichthof Productions

Árnyék
térképek



1. pass:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(lightx, lighty, lightz,
          lookatx, lookaty, lookatz,
          lupx, lupy, lupz);
```

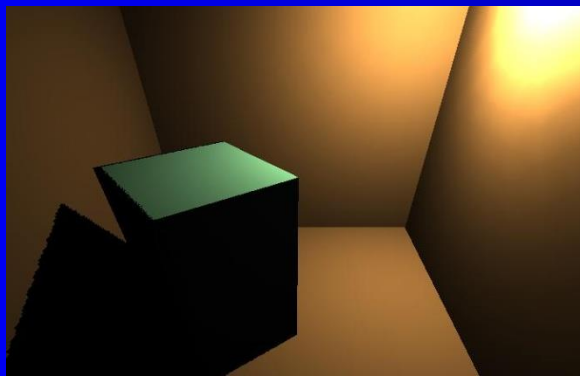
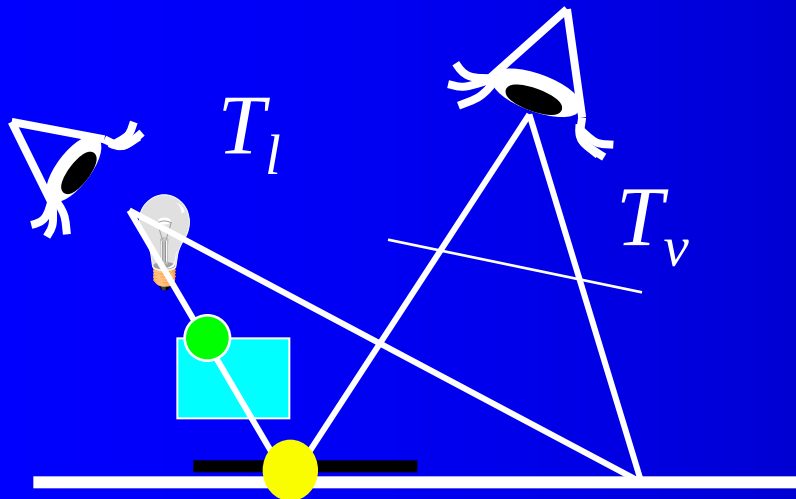
Modellezés
Képszíntézis

Z-buffer -> textúra máss

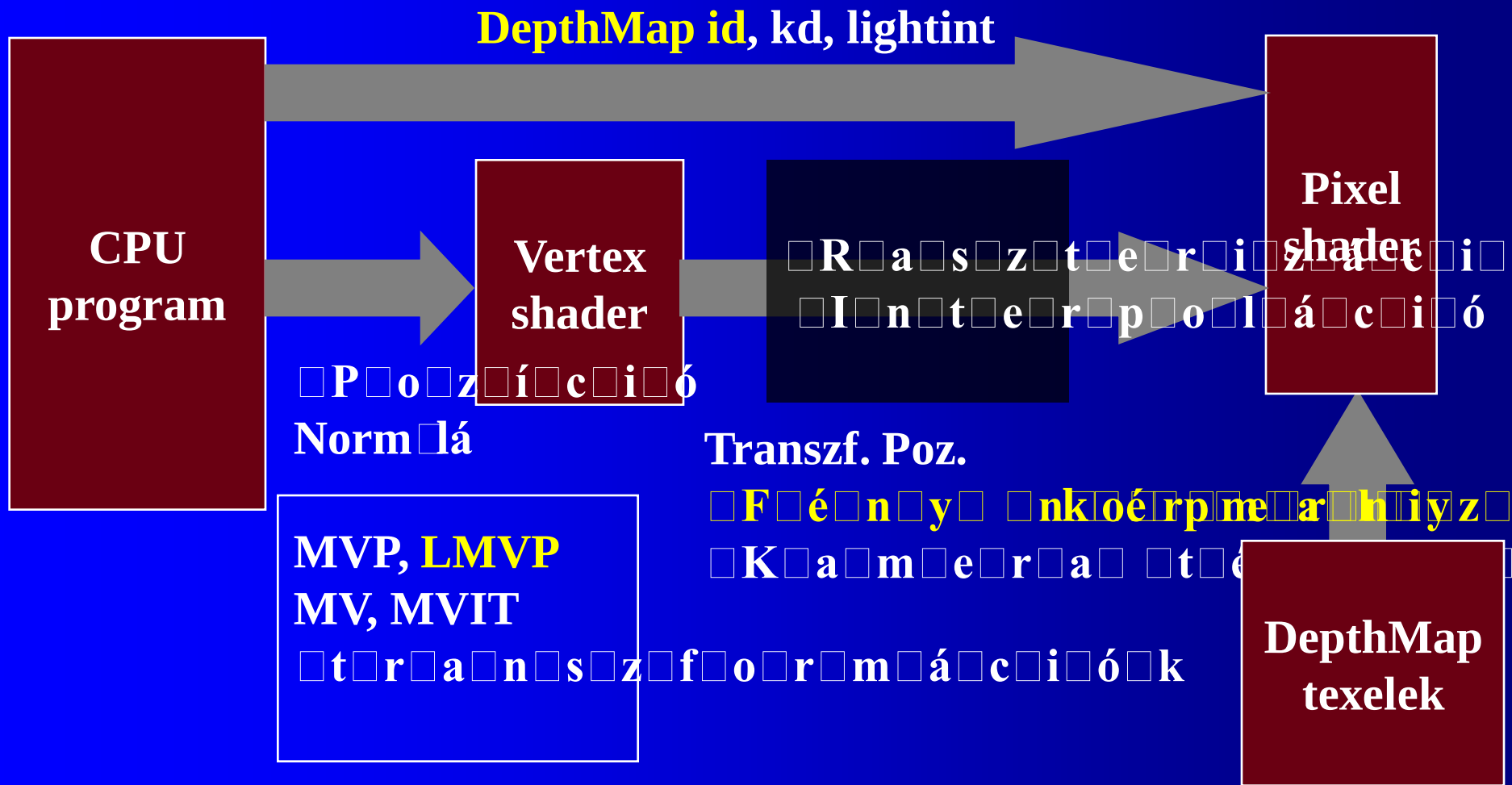
2. pass:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez,
          lookatx, lookaty, lookatz,
          upx, upy, upz);
```

Modellezés
Képszíntézis

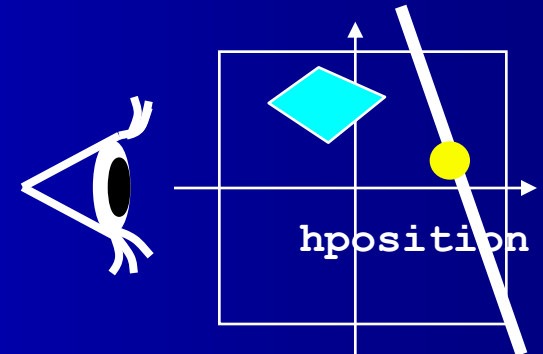
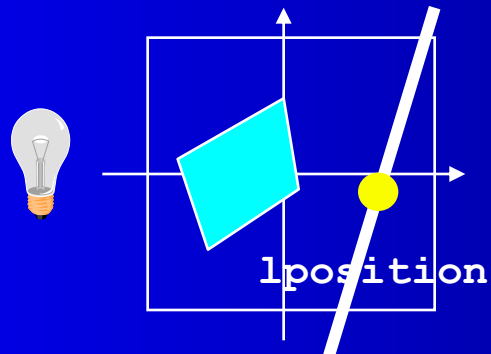
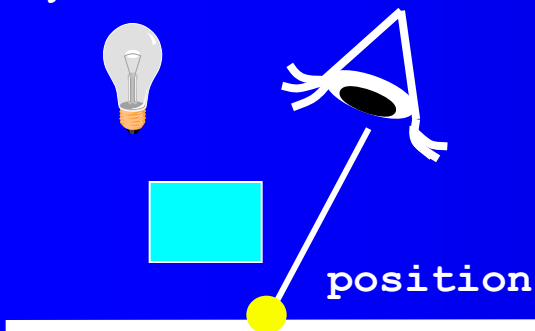


Képszerkesztő architektúrái



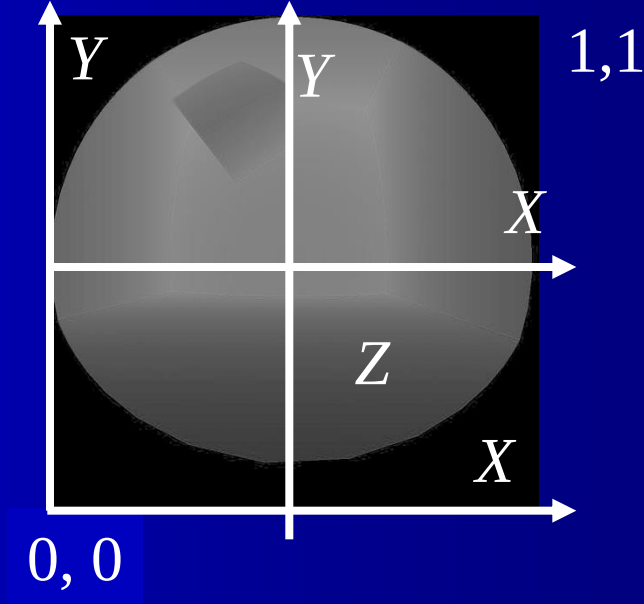
Vertex shader

```
void main(    in float4 position      : POSITION,
              in float4 normal      : NORMAL,
              uniform float4x4 MVP, LMVP, MV, MVIT, ☹️
              uniform float4 clightpos,
              out float4 hPosition  : POSITION,
              out float4 lPosition  : TEXCOORD0,
              out float3 cnormal    : TEXCOORD1,
              out float3 clight     : TEXCOORD2
) {
    hPosition = mul(MVP, position); // to eye's clip space
    lPosition = mul(LMVP, position); // to light's clip space
    float4 cp = mul(MV, position); // camera space (Phong shade)
    cnormal = mul(MVIT, normal).xyz;
    clight = clightpos.xyz * cp.w - cp.xyz * clightpos.w;
}
```



Fragment shader

```
void main(  
    in float4 lPosition : TEXCOORD0,  
    in float3 N          : TEXCOORD1,  
    in float3 L          : TEXCOORD2,  
    uniform float4 lightint, kd, ☹  
    uniform sampler2D depthMap,  
    uniform float bias,  
    out float4 ocolor    : COLOR )  
{  
    float3 lPosCartesian = lPosition.xyz/lPosition.w;  
    float2 texcoord;  
    texcoord.x = (lPosCartesian.x + 1)/2;  
    texcoord.y = (lPosCartesian.y + 1)/2;  
    float this_depth = (lPosCartesian.z + 1)/2 - bias;  
    float stored_depth = tex2D(depthMap, texcoord);  
    if (this_depth <= stored_depth) { // == k☹☹☹  
        N = normalize(N); L = normalize(L);  
        ocolor = lightint * kd * max(dot(N,L), 0);  
    } else  
        ocolor = float3(0, 0, 0);  
}
```

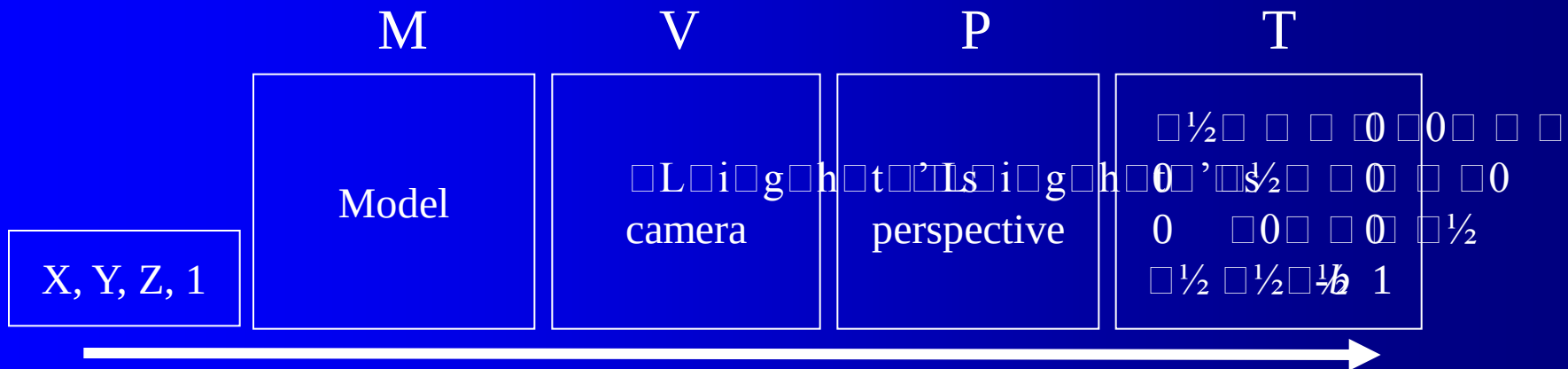


Clipping
space to
Texture
space

Projektor textúrázás ívmatrixa

```
float3 lPosCartesian = lPosition.xyz/lPosition.w;

texcoord.x = (lPosCartesian.x + 1)/2;
texcoord.y = (lPosCartesian.y + 1)/2;
float this_depth = (lPosCartesian.z + 1)/2 - bias;
```



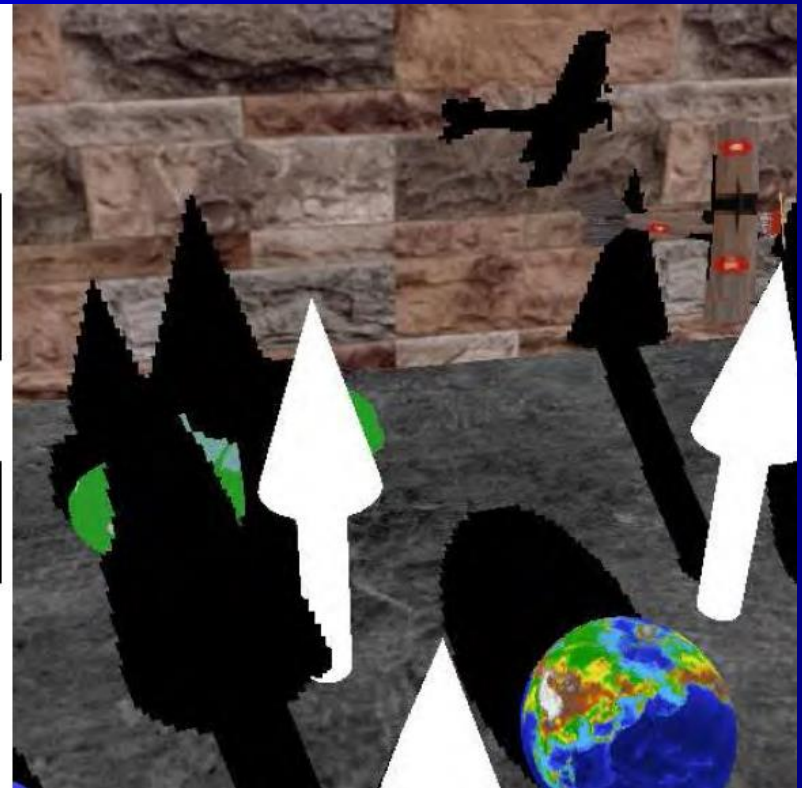
Egyes ábrák

```
void SM_VS(
    in float4 position      : POSITION,
    in float4 normal        : NORMAL,
    uniform float4x4 MVP, LMVPT, MV, MVIT, ☹
    uniform float4 clightpos,
    out float4 hPosition    : POSITION,
    out float4 tPosition    : TEXCOORD0,
    out float3 cnormal, clight ☹)
{
    hPosition = mul(MVP, position); // to eye's clip space
    tPosition = mul(LMVPT, position); // to depth texture space
    float4 cp = mul(MV, position); // camera space
    cnormal = mul(MVIT, normal).xyz;
    clight = clightpos.xyz * cp.w - cp.xyz * clightpos.w;
}
```

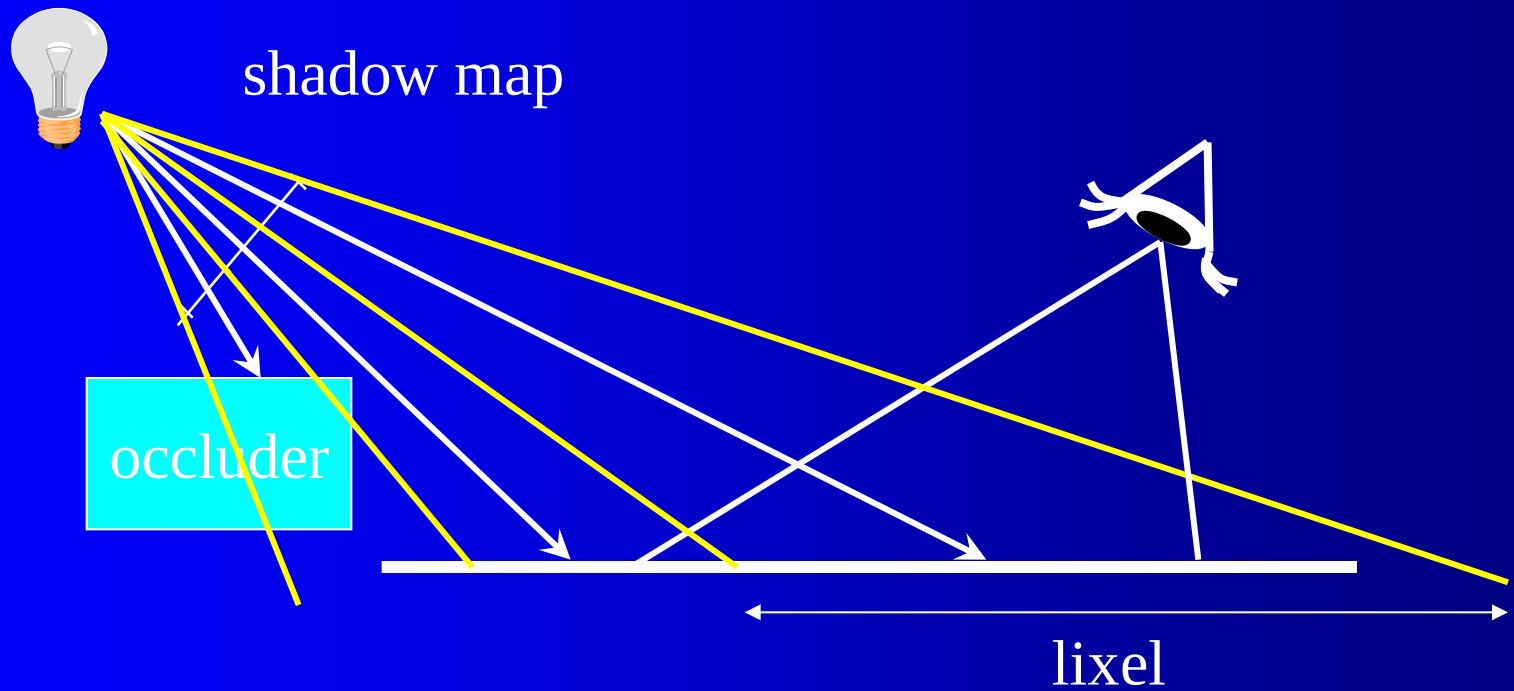
```
void SM_FS (
    in float4 tPosition : TEXCOORD0,
    in float3 N, L ☹
    uniform float4 lightint, kd, ☹
    uniform sampler2D depthMap,
    out float4 ocolor      : COLOR )
{
    float costheta = max(dot(normalize(N), normalize(L)), 0);
    ocolor = lightint * kd * costheta * tex2Dproj(depthMap, tPosition);
}
```

Returns 0/1
if it is a
depth texture

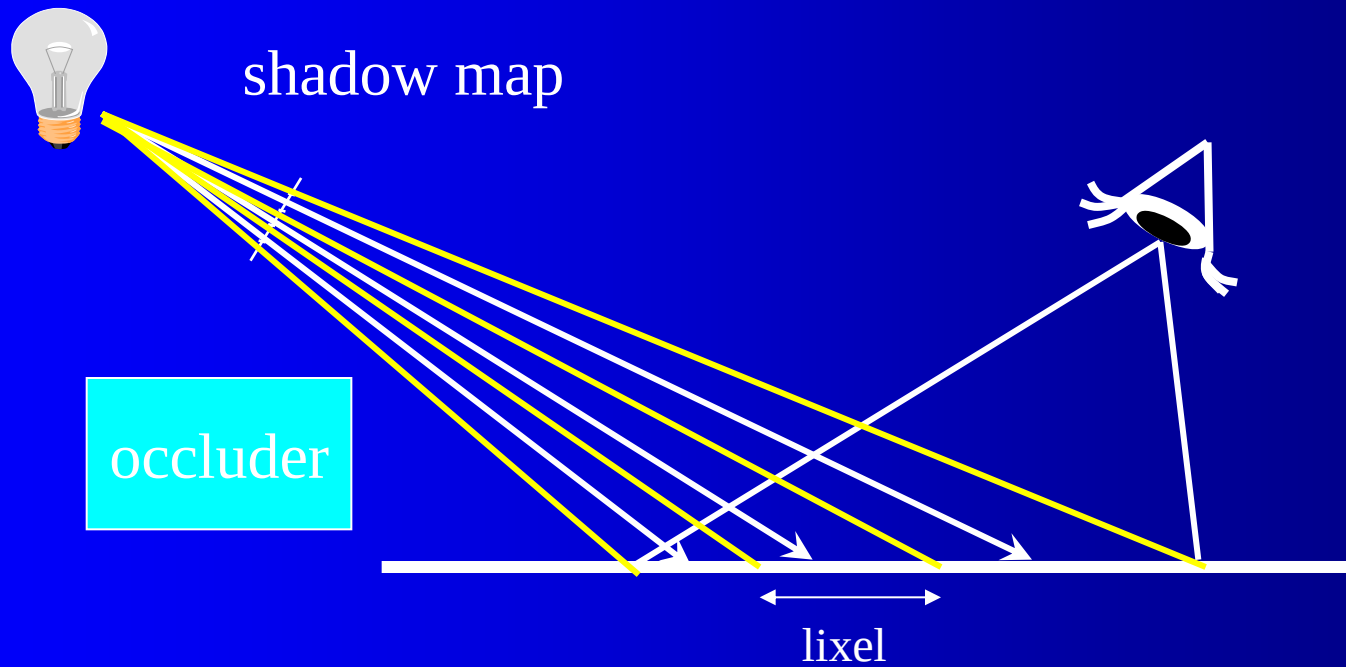
Árnyékterkép-érealizáció



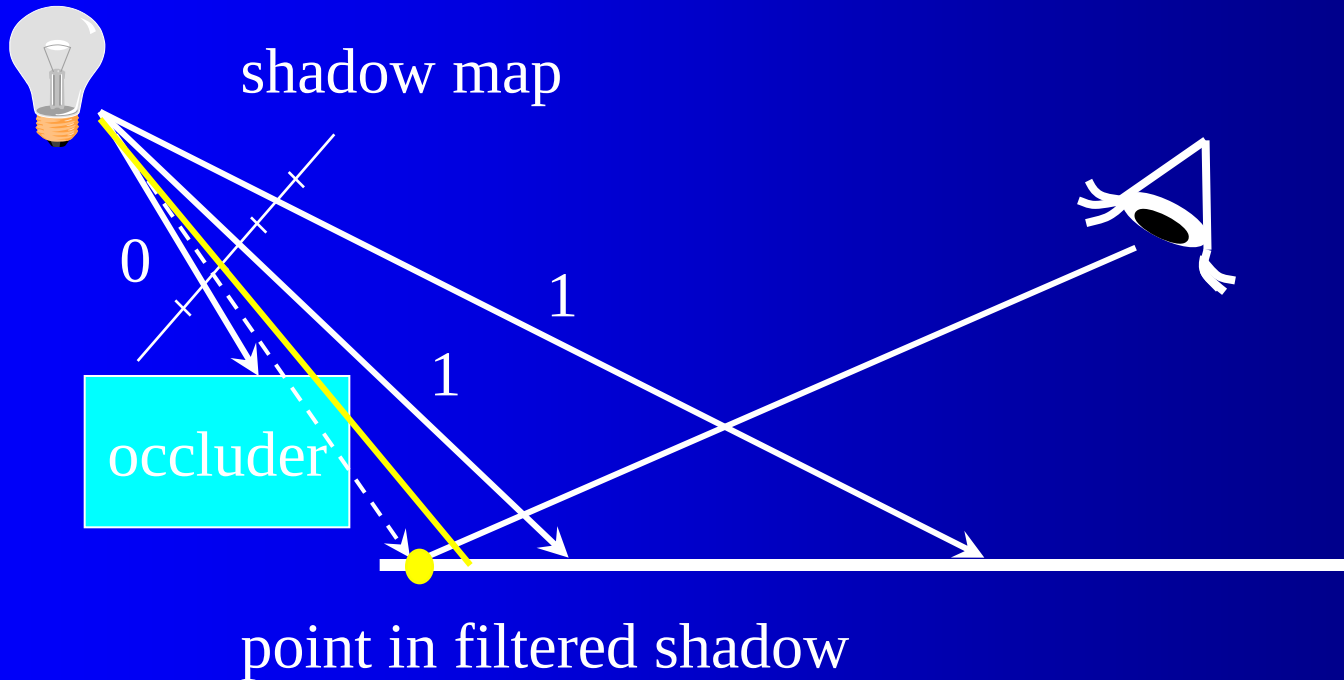
Árnyékterkép-élességi rök



Árnyékalkotóeszköz



Percentage Closer Filtering



- Shadow map: depths at discrete points
- In between: depth is a random variable Z
- $P(Z > z)$ = shadowing factor



classic shadow map

2×2 PCF, no focusing

focusing

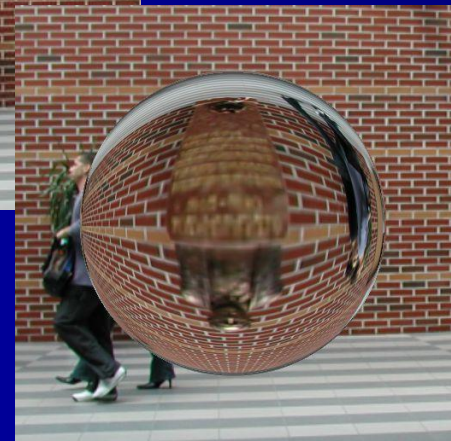
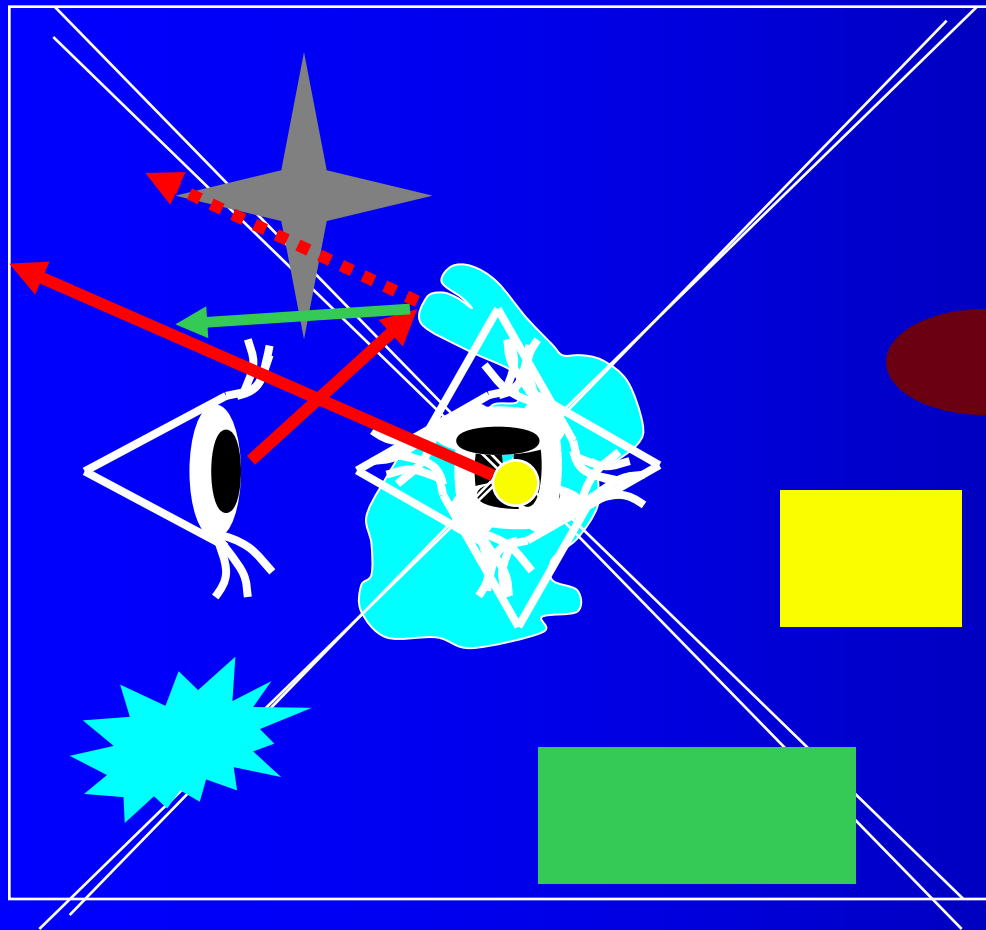
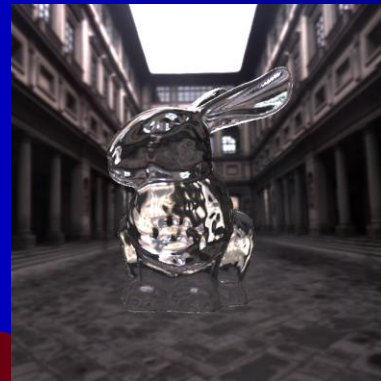
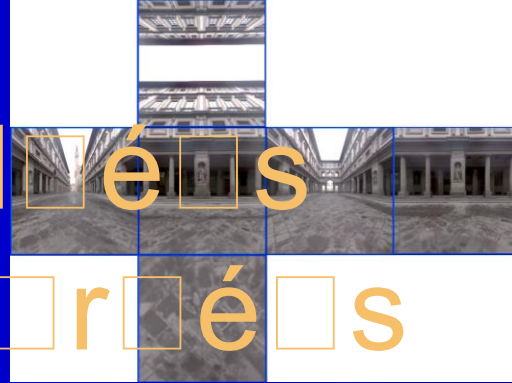


Classic shadow map

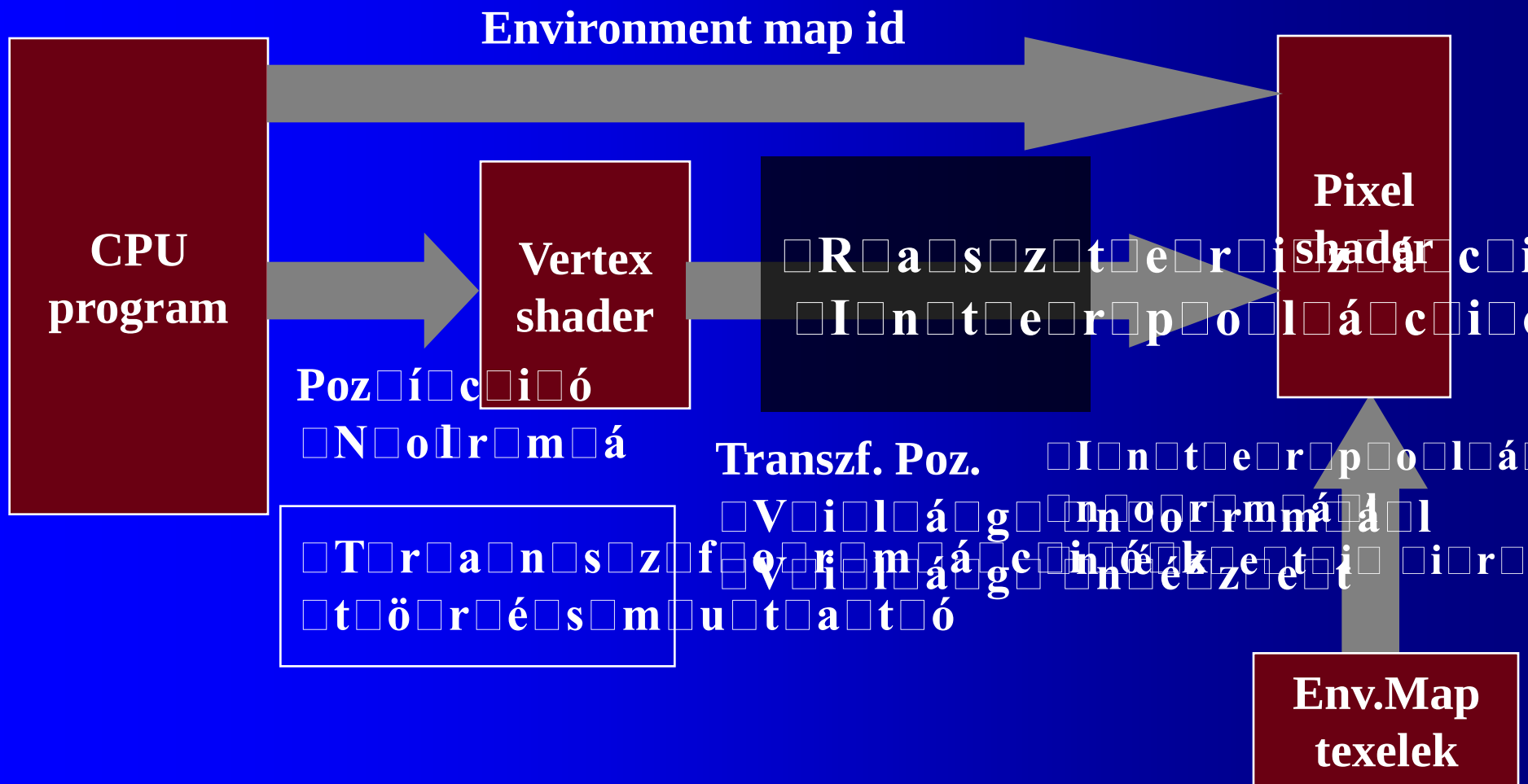


Variance shadow map

Visszaverődés □ d □ é □ s □ é □ t □ ö □ r □ é □ s



Visszavértesítés/ törés számitás



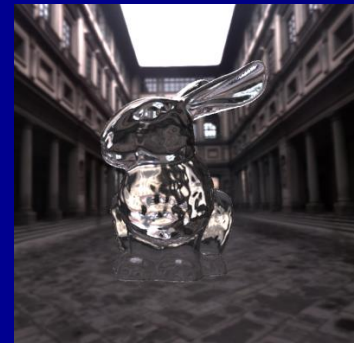
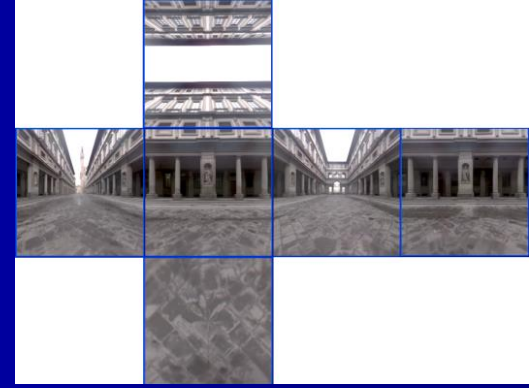
Sima objekum vertex shader

```
void main(    in float4 position : POSITION,
              in float4 normal   : NORMAL,
              uniform float4x4 MVP,          // modelviewproj
              uniform float4x4 M,           // model
              uniform float4x4 MIT,         // IT of model
              uniform float3 eye,           // eye in world
              out float4 hPos : POSITION,
              out float3 V    : TEXCOORD0, // view in world
              out float3 N    : TEXCOORD1) // normal in world
{
    hPos      = mul(MVP, position);

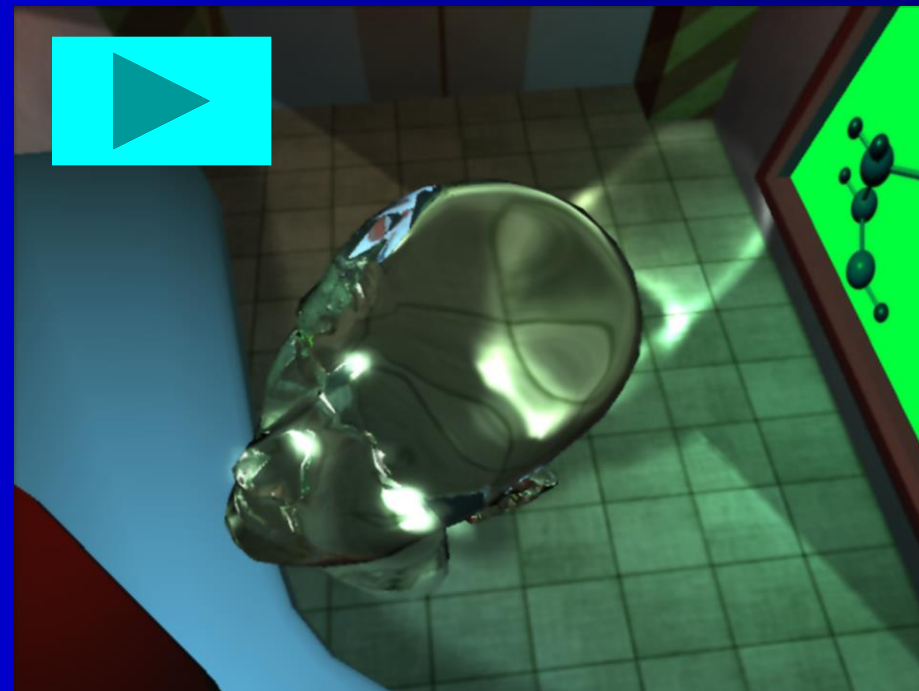
    float3 p = mul(M, position).xyz; // transform to world sp.
    V = eye - p;
    N = mul(MIT, normal).xyz;
}
```

Sima objektum fragment shader

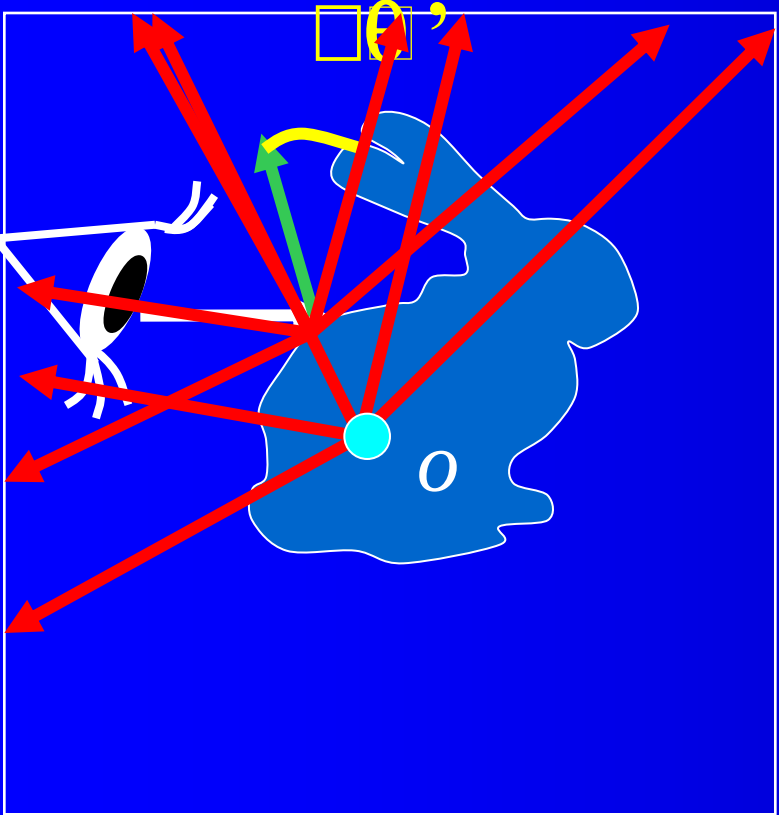
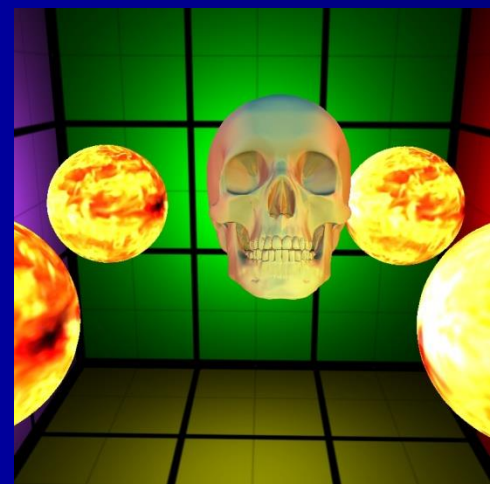
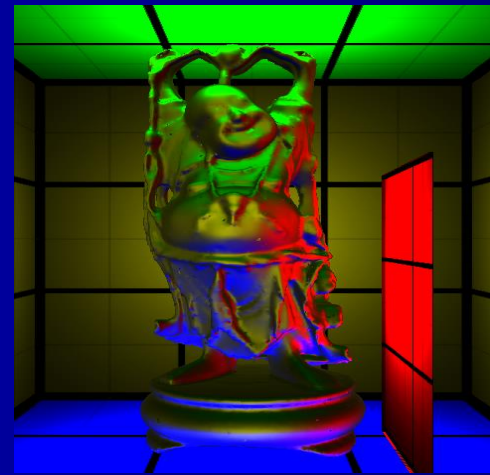
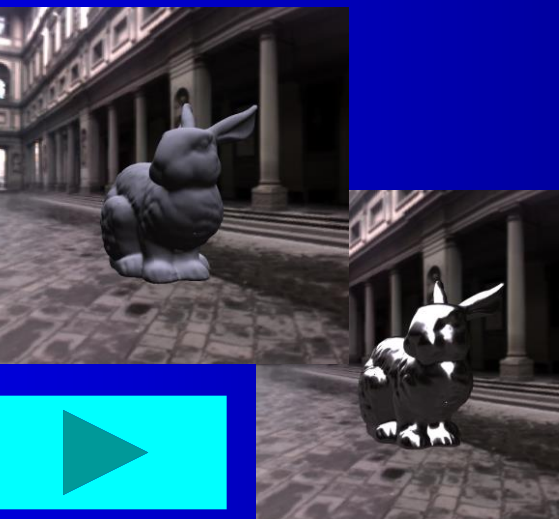
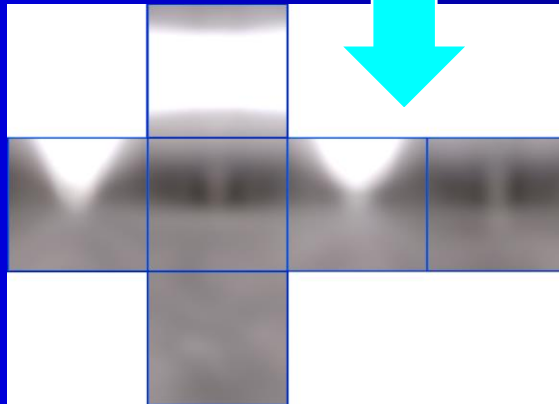
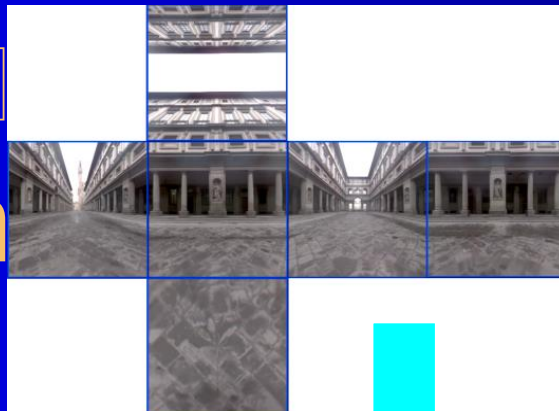
```
void main(  
    in float3 V : TEXCOORD0,  
    in float3 N : TEXCOORD1,  
    uniform float n, // törésmutató  
    uniform float F0, //  $F0 = [(n-1)/(n+1)]^2$   
    uniform samplerCUBE envMap,  
    out float3 color )  
{  
    V = normalize(V);  
    N = normalize(N);  
    float3 T = refract(V, N, 1/n);  
    float3 R = reflect(V, N);  
    float3 refractedRad = texCUBE(envMap, T).rgb;  
    float3 reflectedRad = texCUBE(envMap, R).rgb;  
    float F = F0 + (1-F0) * pow(1-dot(N,V), 5);  
    color = F * reflectedRad + (1-F) * refractedRad;  
}
```



Lokalizáció
környezeti
leképezés



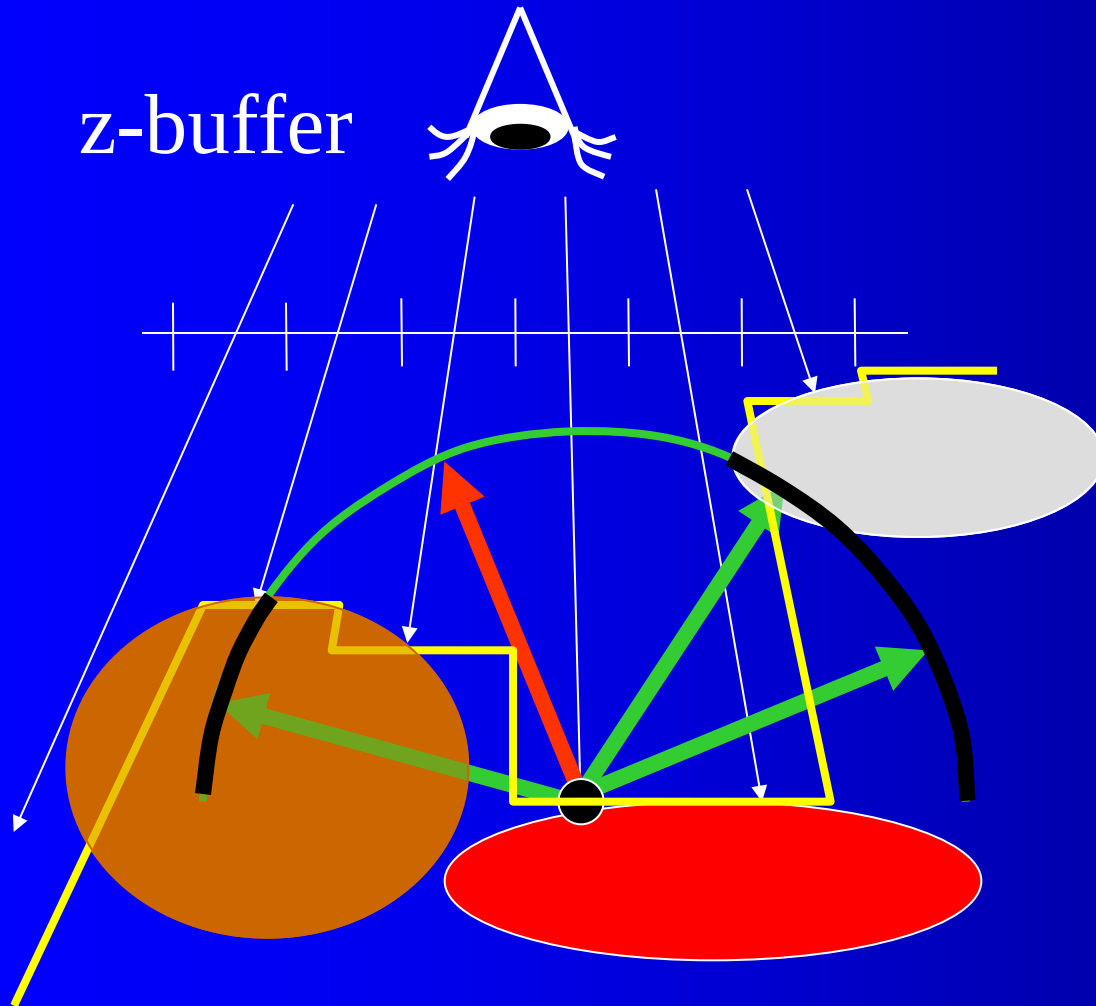
Diffuse illumination



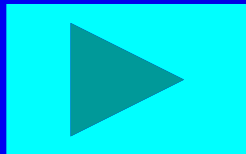
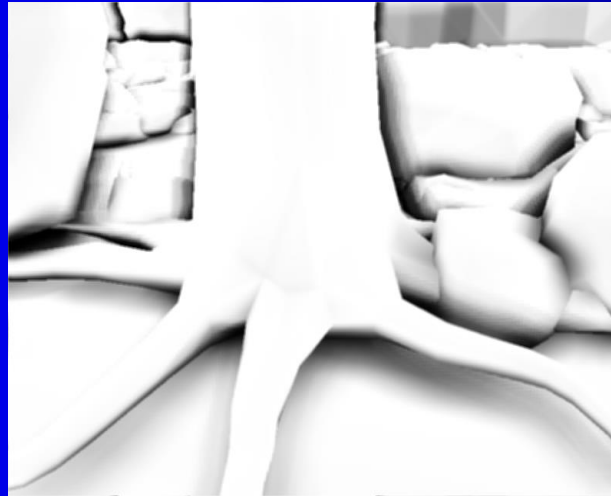
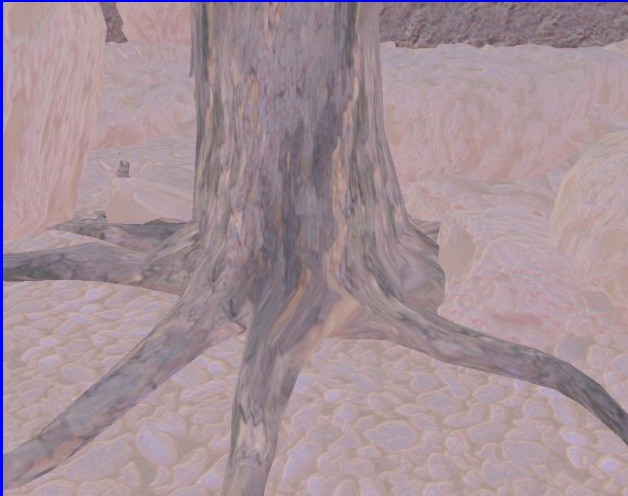
$$\int_{\Omega} L^{\text{in}}(o, \omega) \cos \theta d\omega$$



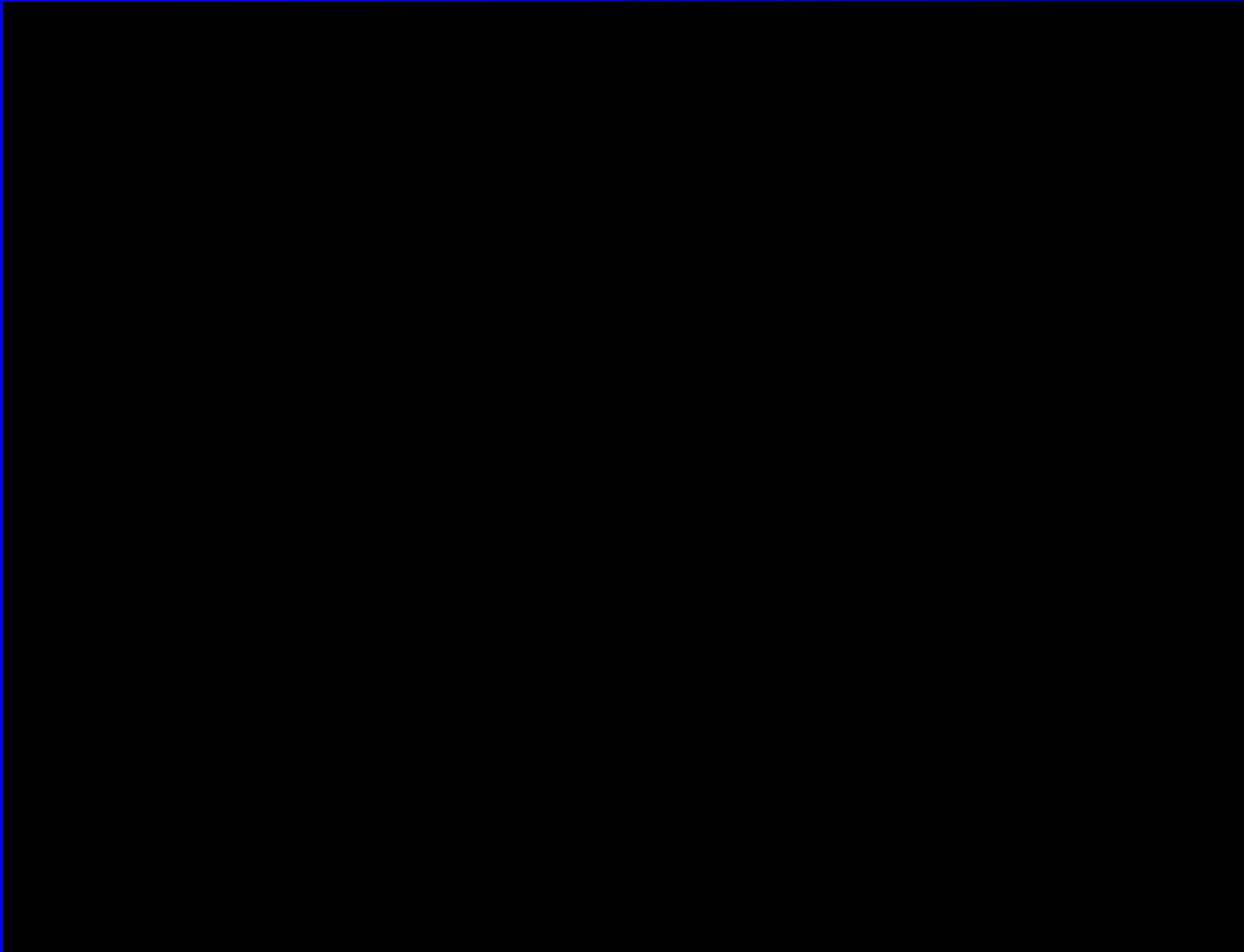
Screen-Space Ambient Occlusion



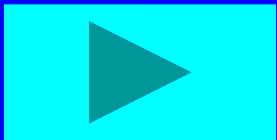
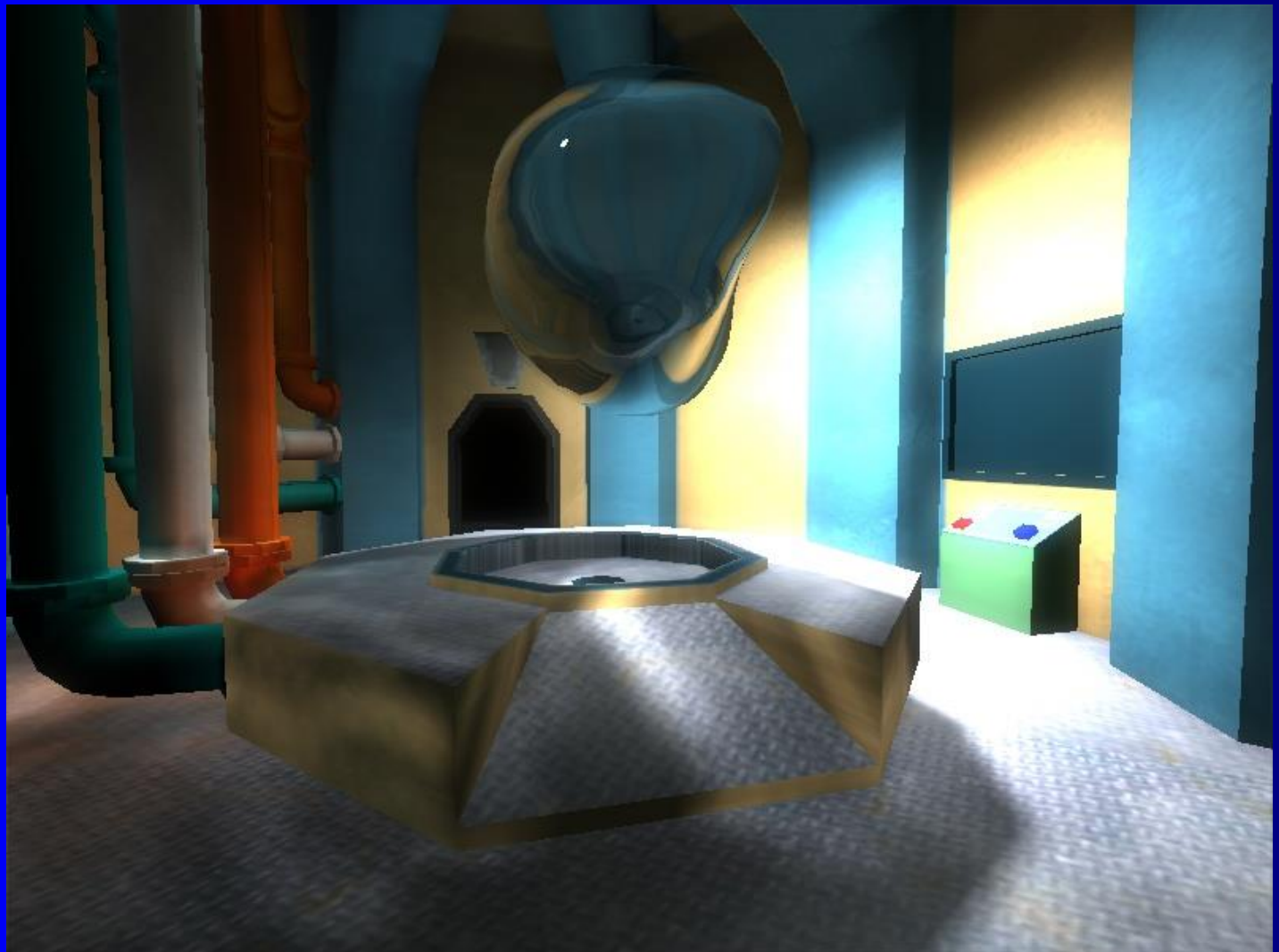
Ambient Occlusion



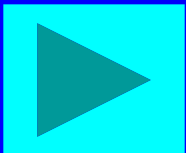
Autós játékos

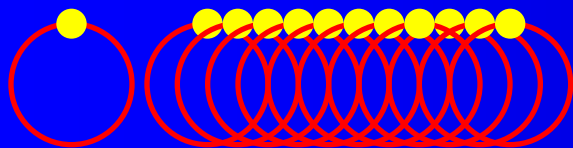


Úrallomás

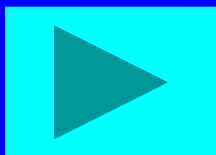


Moria

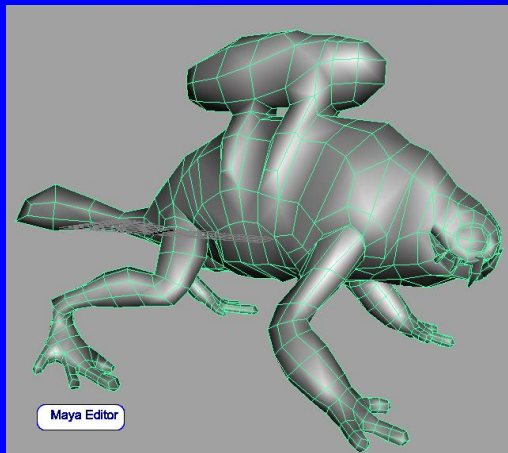




Óc e á n



Geometria



Catmull-Clark
subdivision

Proceduralis
geometria



GPGPU

Geometria:

Az eredmény

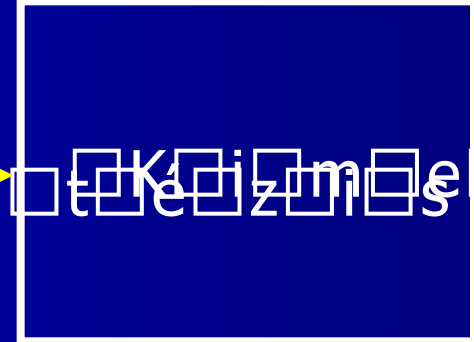
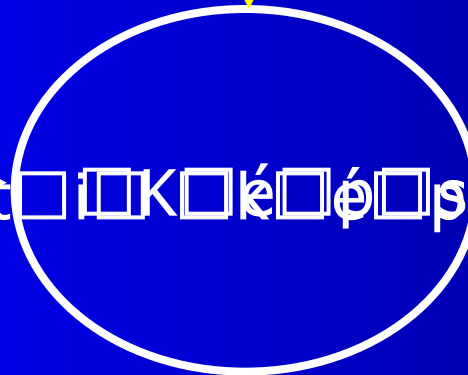
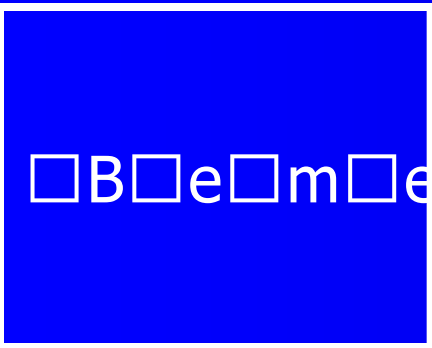
előlemez

„háromszögök

Ered

töml

Bemeneti
adat



Textúrák Mindezen kéimen

ugyanaz az algoritmus,

más adatokra:

Text

ra

„Teleskepėnny

```
glViewport(0, 0, HRES, VRES)
glBegin(GL_QUADS);
glTexCoord2f(1,1); glVertex4f(-1,-1, 0, 1);
glTexCoord2f(1,0); glVertex4f(-1, 1, 0, 1);
glTexCoord2f(0,0); glVertex4f( 1, 1, 0, 1);
glTexCoord2f(0,1); glVertex4f( 1,-1, 0, 1);
glEnd( );
```

Vertex shader (Cg):

```
void main ( in float4 Pos : POSITION,
            in float2 Tex : TEXCOORD0,
            out float4 hPos : POSITION,
            out float2 oTex : TEXCOORD0 ) {
    hPos = Pos;
    oTex = Tex;
}
```

Melyik kimeneti

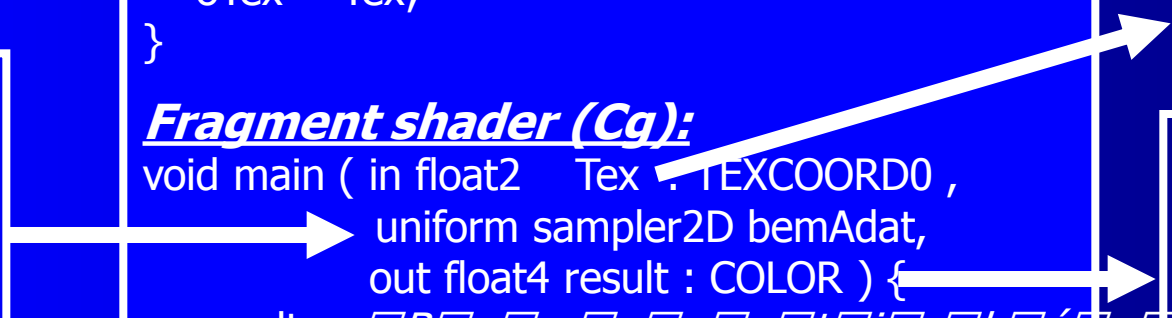
t ö m b e
s z á m í

Fragment shader (Cg):

```
void main ( in float2 Tex : TEXCOORD0 ,
           uniform sampler2D bemAdat,
           out float4 result : COLOR ) {
    result = Bemene
            tex2D(bemAdat, f(Tex)) all;
}
```

Bemeneti adat

Ereignis



É
k

e
d

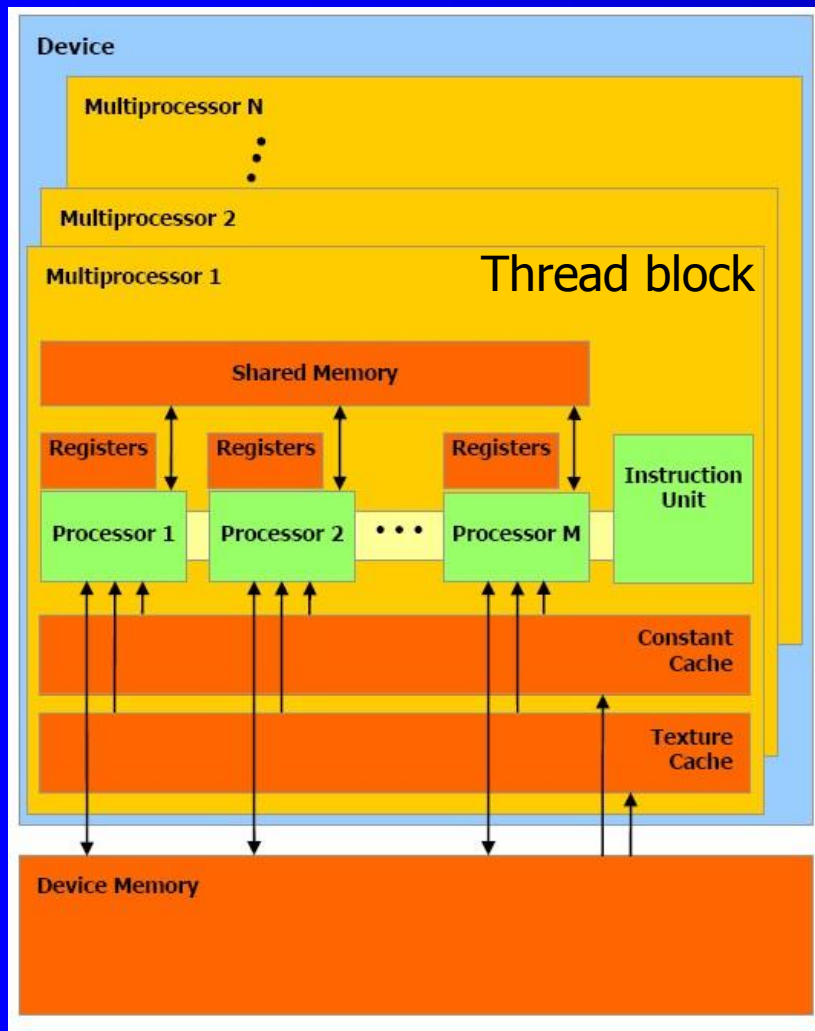
l
o
á
s
z
á

void ma
in f
uni
out
{
floa
floa
floa
floa
floa
res
}



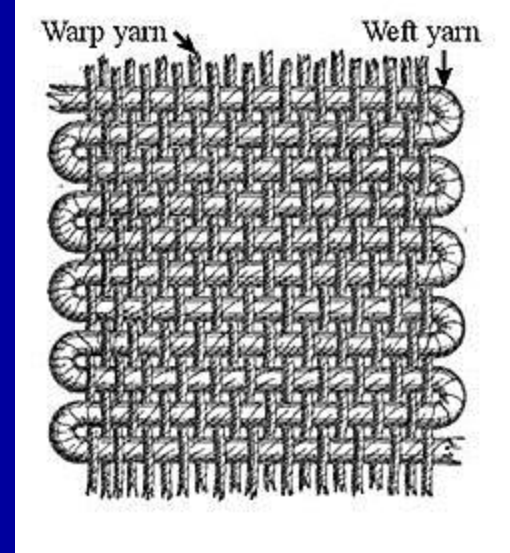
$$\text{Tex}(\frac{du}{dv}) \frac{du}{dv};$$
$$\text{Tex}(\frac{dv}{du}) \frac{dv}{du};$$

CUDA (OpenCL)



GPU

Kernel program:



Threads

block, block,

□W□a□r□p□,□

SIMD

Shared memory

SIMD execution

□p□,□□

Két N elemű vektor összeadása

A GPU-n fut, de a CPU-ról is hívható

```
__global__ void AddVectorGPU( float *C, float *A, float *B, int N ) {  
    int i = blockIdx.x * blockDim.x + threadIdx.x; // sorszám  
    if (i < N)  
        C[i] = A[i] + B[i];  
}
```

0, ..., 1 gridDim, 1 block

```
float C[100000], A[100000], B[100000];
```

```
int main ( ) {  
    ...  
    int N = 100000;  
    ...  
    int blockDim = 256; // #threads in a block: 128, 256, 512  
    int gridDim = (N + blockDim - 1) / blockDim; // #blocks  
    AddVectorGPU<<<gridDim, blockDim>>>(C, A, B, N);  
    ...  
}
```


Tudományos (m)énnyelzők (m)

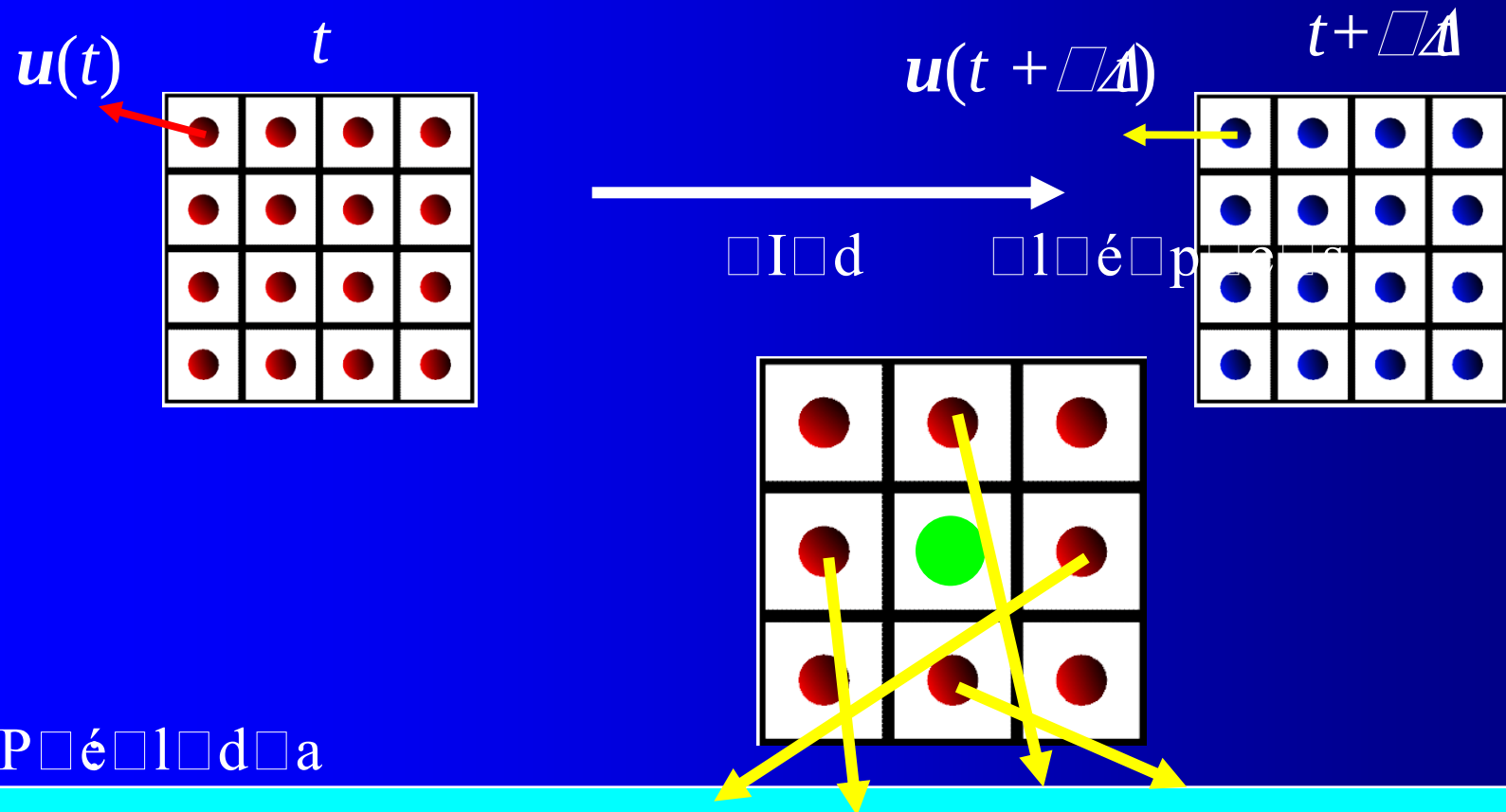
Z-tengely körüli forgatás
homogén lineáris
transzformációs mátrixa



$$\frac{d\vec{u}}{dt} = -(\vec{u} \cdot \nabla)\vec{u} - \frac{1}{\rho} \nabla p + \nu \nabla^2 \vec{u} + \vec{F}$$

$$\nabla \cdot \vec{u} = 0$$

Euler-i folyadék



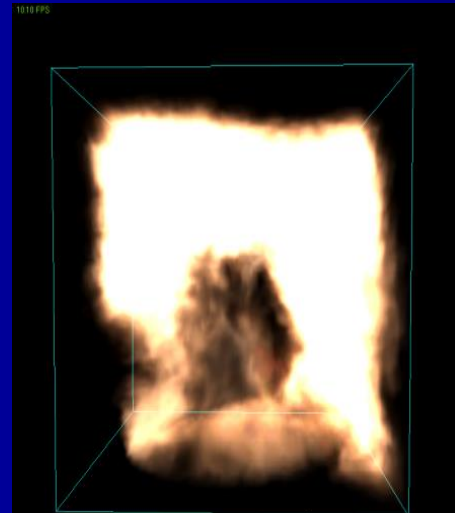
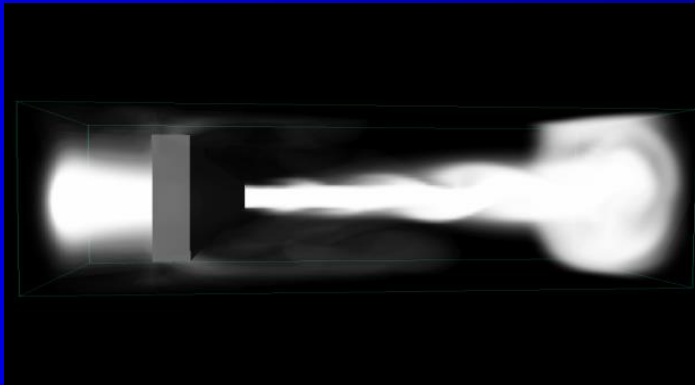
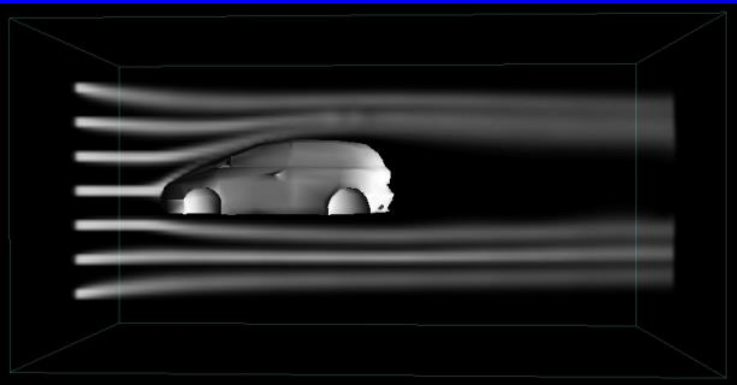
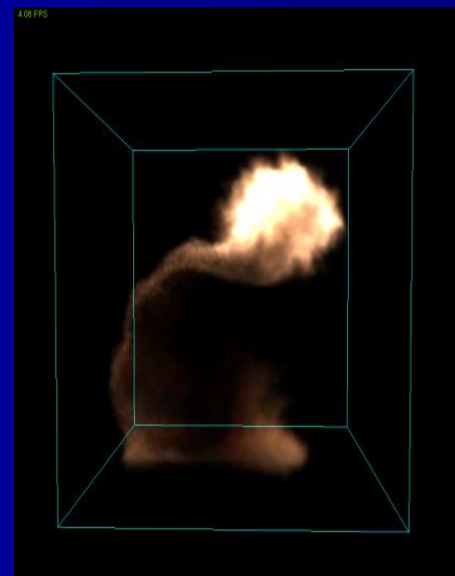
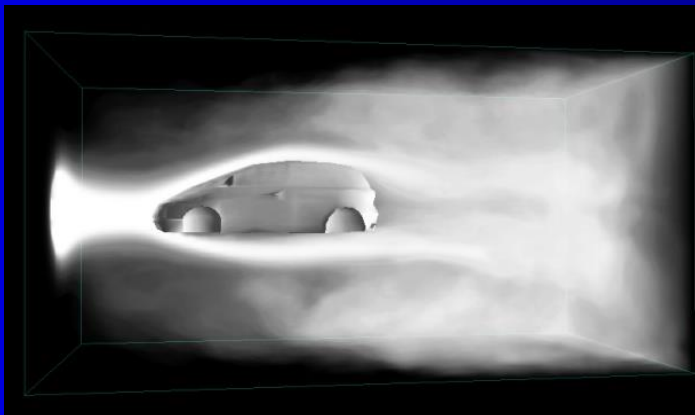
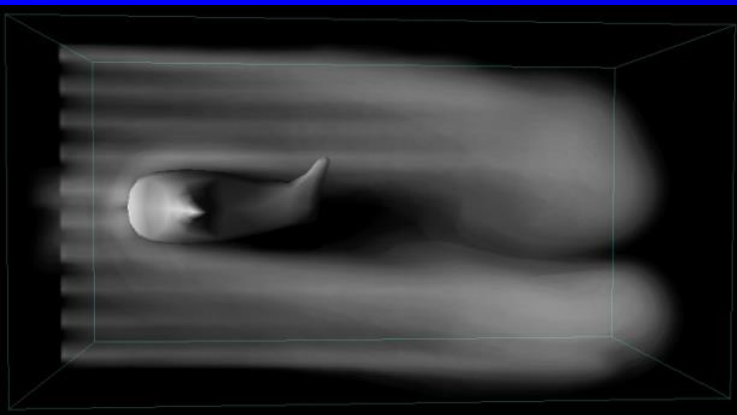
Példá

$$\nabla \cdot \vec{u} = \text{div } \vec{u} = \frac{\partial \vec{u}_x}{\partial x} + \frac{\partial \vec{u}_y}{\partial y} + \frac{\partial \vec{u}_z}{\partial z} = \frac{\vec{u}_x^{i+1,j,k} - \vec{u}_x^{i-1,j,k}}{2\delta x} + \frac{\vec{u}_y^{i,j+1,k} - \vec{u}_y^{i,j-1,k}}{2\delta y} + \frac{\vec{u}_z^{i,j,k+1} - \vec{u}_z^{i,j,k-1}}{2\delta z}$$

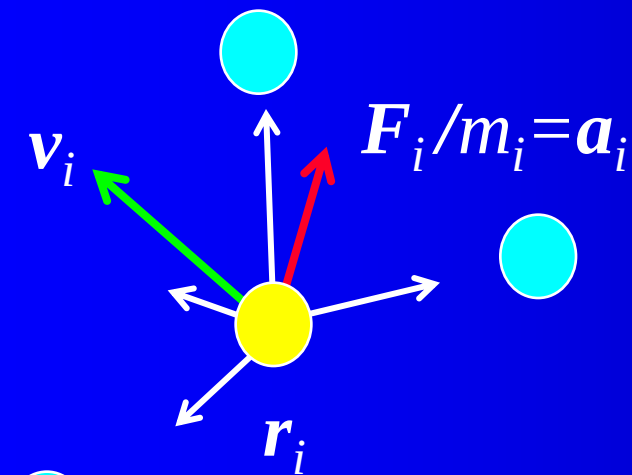
$$\frac{d\vec{u}}{dt} = -(\vec{u} \cdot \nabla)\vec{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\vec{u} + \vec{F}$$

$$\nabla \cdot \vec{u} = 0$$

Euler-i folyadék

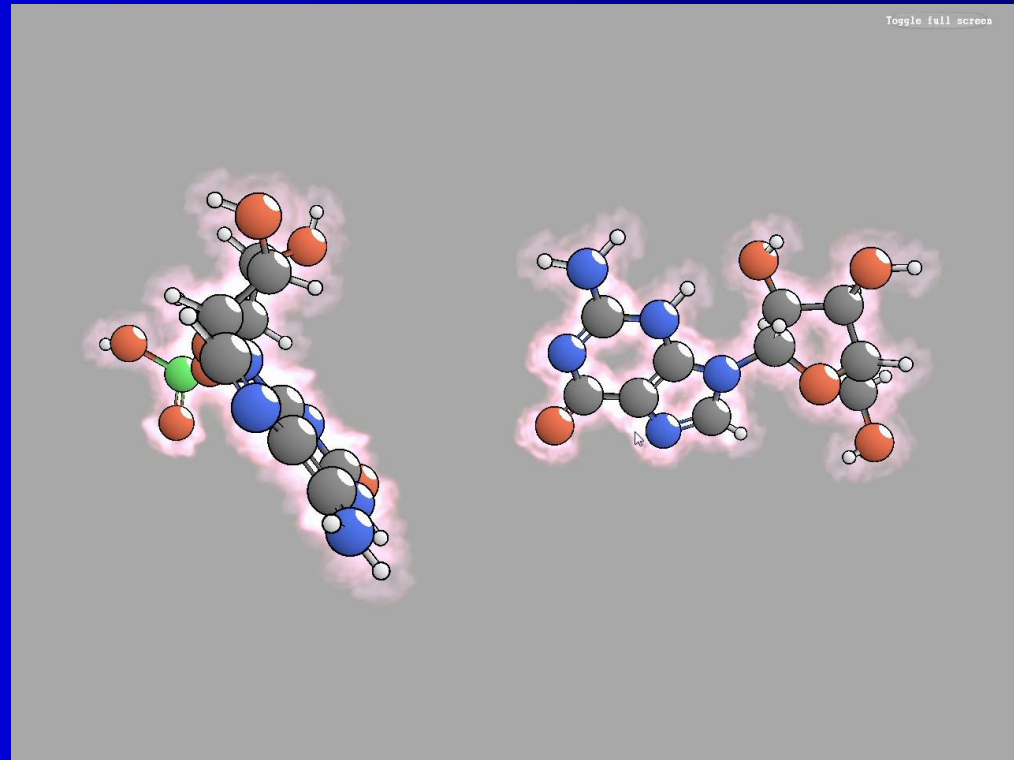


Lagrange-módszer Részecskerendszer, k-body



$$\mathbf{v}_i += \mathbf{a}_i \Delta t$$

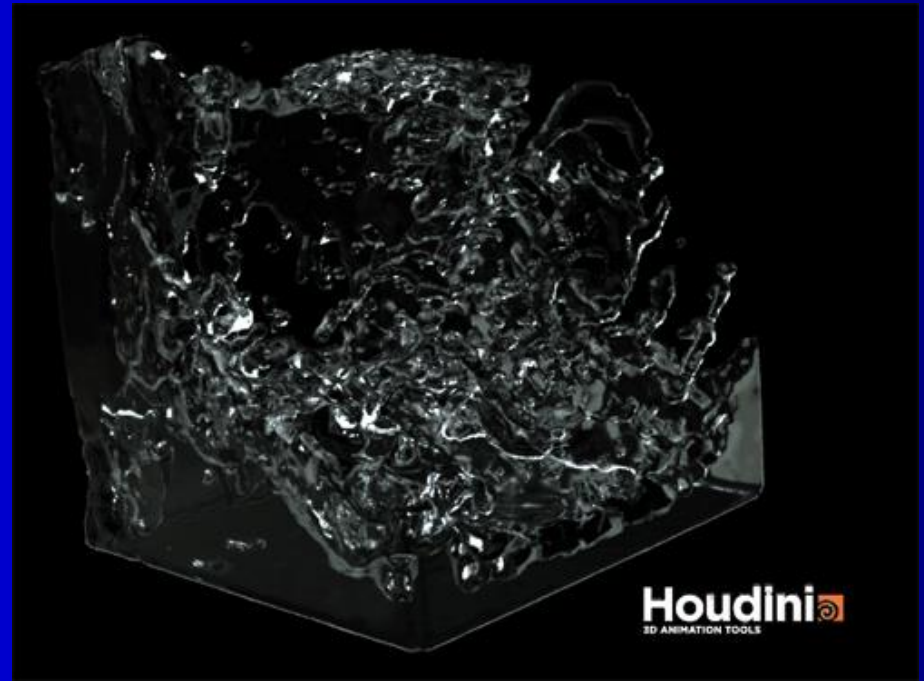
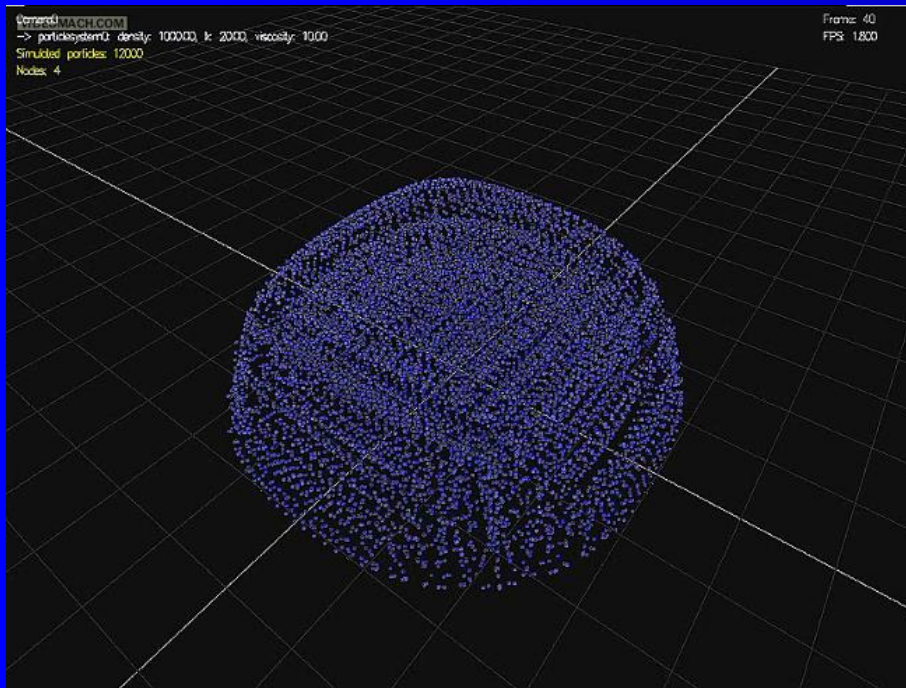
$$\mathbf{r}_i += \mathbf{v}_i \Delta t$$



Emlékeztető: Euler-séma.
Helyet: $\mathbf{V} = \mathbf{K} \cdot \mathbf{z}$ $\mathbf{z}$ a $\mathbf{p}$ $\mathbf{M}^{-1}$

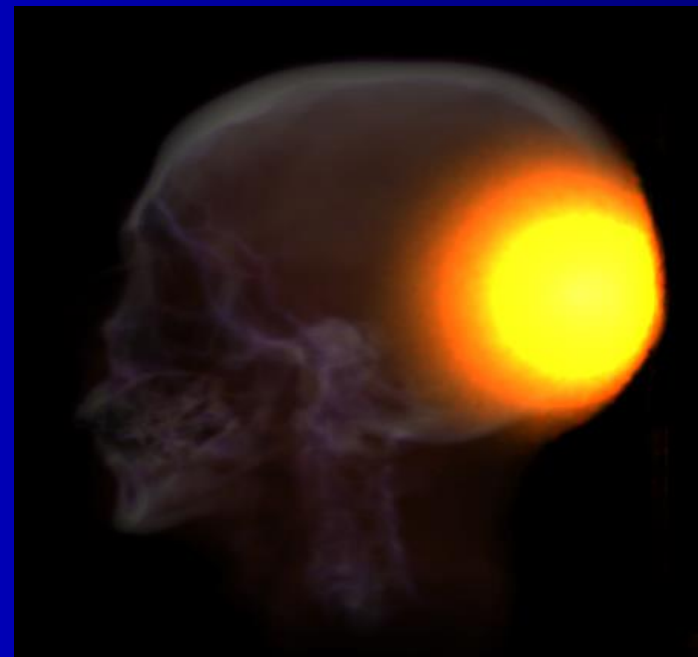
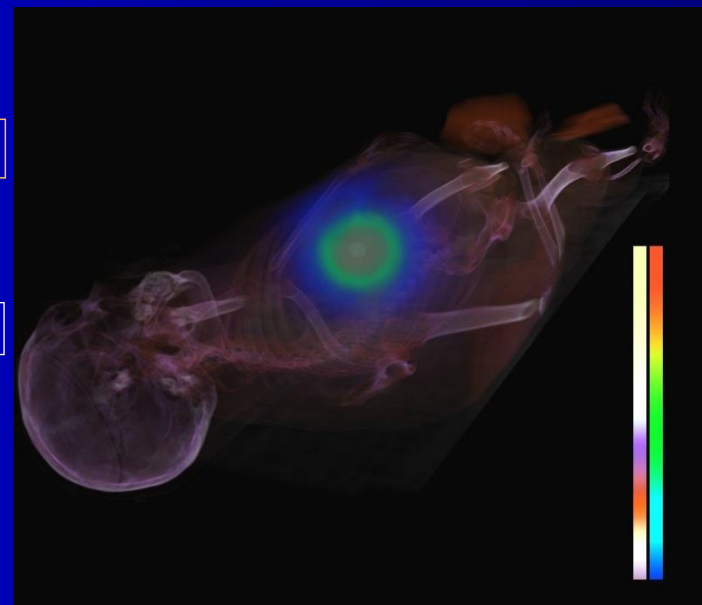
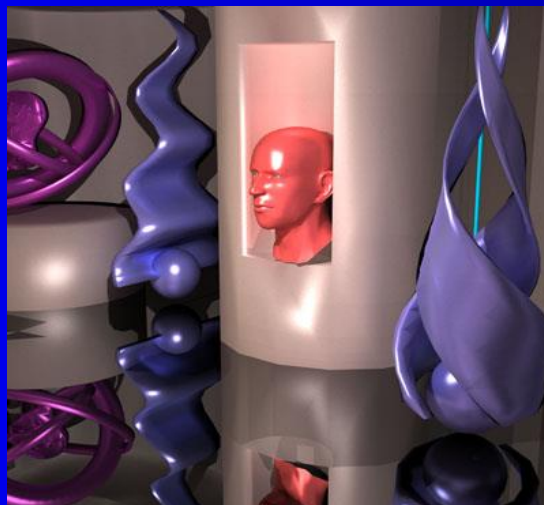
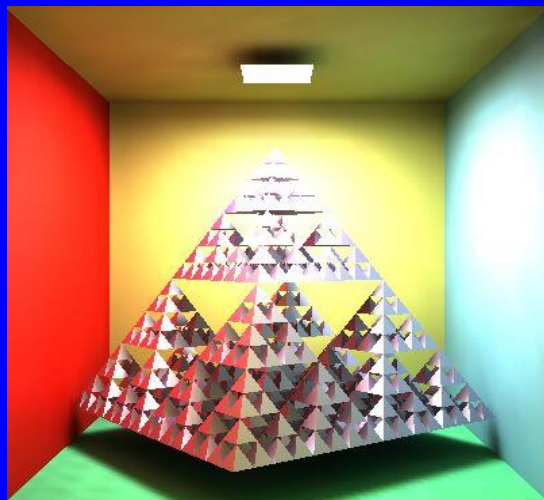
Lagrange-i folyadékáramlás

SPH=Smoothed Particle Hydrodynamics



Transport problems

$$L(x) = L^e(x) + \int_{\Omega} L(y) K(y \rightarrow x) dy$$



Részecskefizika

(Monte Carlo)



- 4096^3 effective resolution
- 5 million photons/frame
- 1 sec/frame



- 4096^3 effective resolution
- 50 million photons/frame
- 9 sec/frame

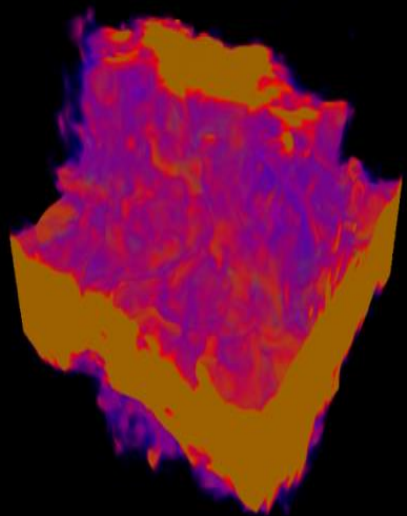
Térfigatványozásiótví

Szirmay-Kalos László



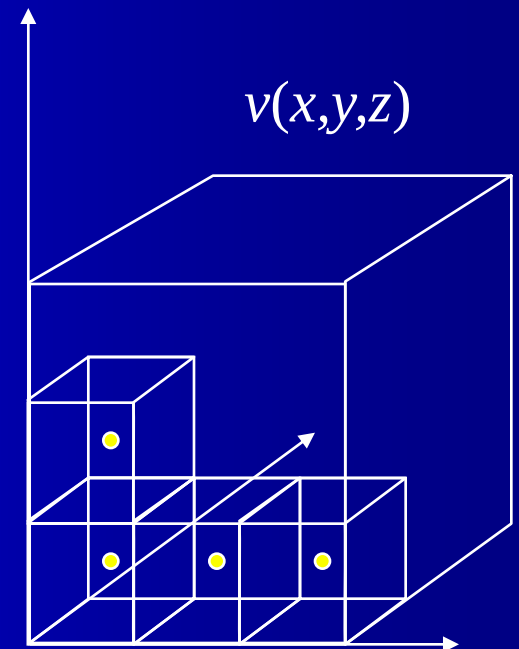
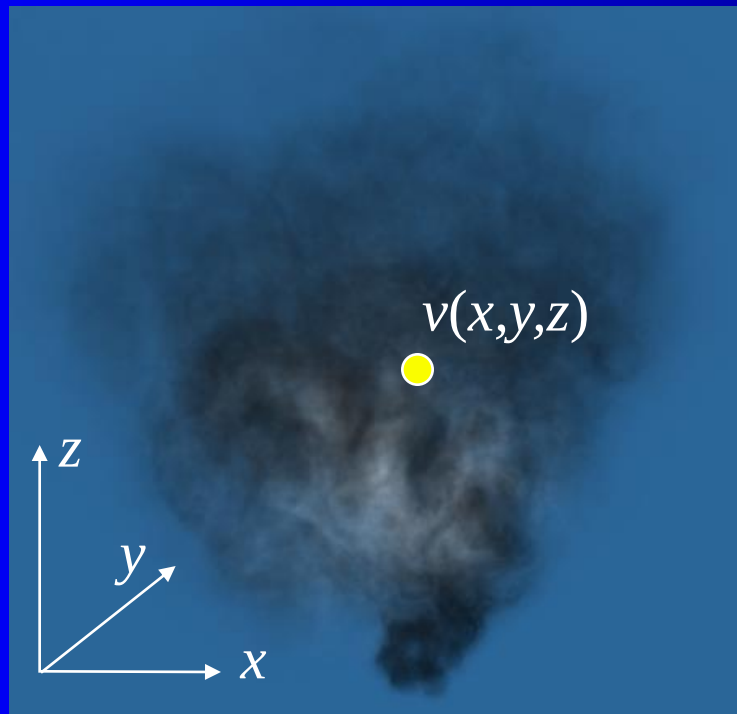
F18_Mouse_30_Kegs

0°



Térfogati modellek (skalármezők)

3D tér pontjainban egy



- Skalár: a 5 mérséklek
nyomás, a potternoclás
- Származás: a Működés, a
Rekonstrukció (tóm)

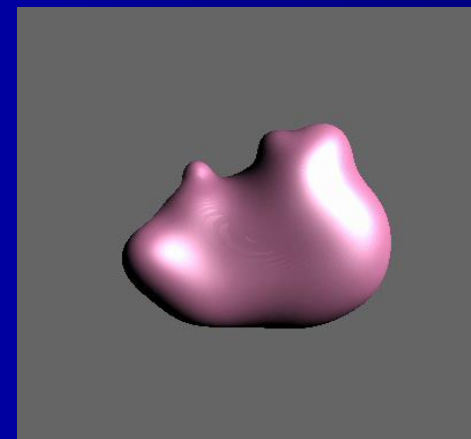
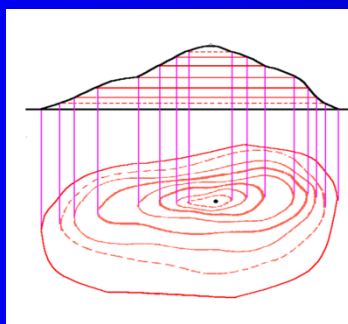
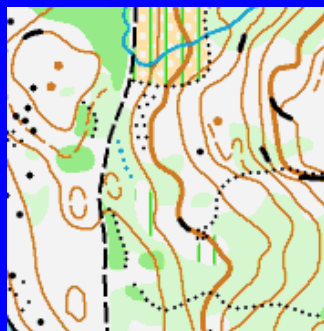
Térfigatási modell megjelenítése

- Megjelenítés fényforrás
anatólógiaját felhasznalva

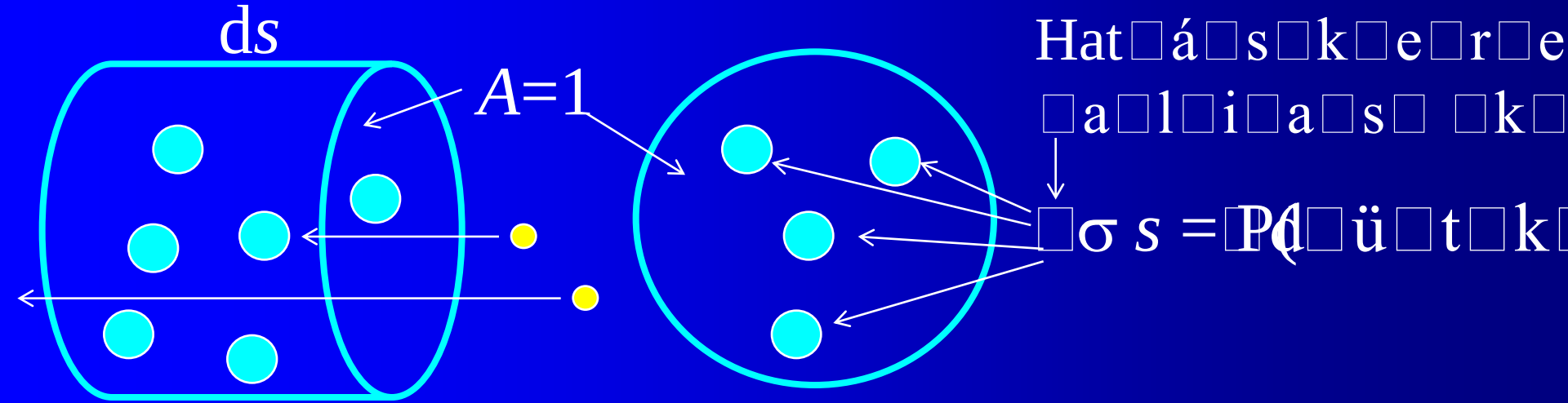


S Cr Cség + d $\frac{\partial}{\partial t}$ v $\frac{\partial}{\partial t}$ a $\frac{\partial}{\partial t}$ p a r a m é t é p

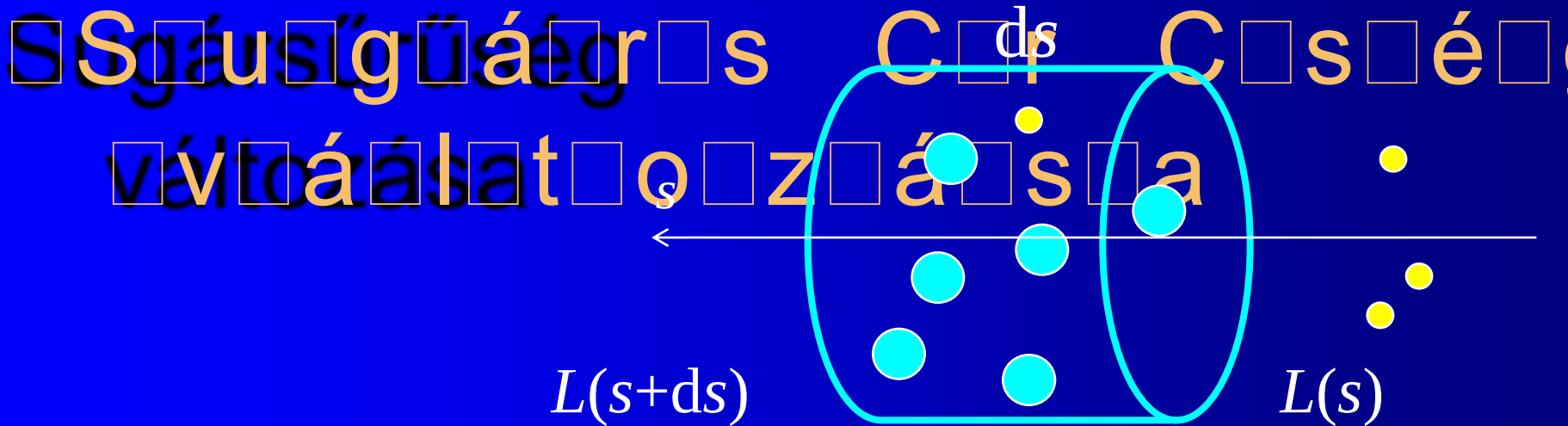
- Adottt színt felület k



Fényezőszérgó



- Albedo a : a nem- $\frac{1}{\text{felvett}}$ 5 d $\frac{1}{\text{éves}}$ felvételt, a hóg $\frac{1}{\text{y}}$ az
- Fekete test: albedo = 0



$$L(s+ds) = L(s) - L(s) \kappa(s) ds + L^e(s) ds + L^i(s) ds$$

where $\kappa(s)$ is the extinction coefficient, $L^e(s)$ is the emission, and $L^i(s)$ is the in-scattering.

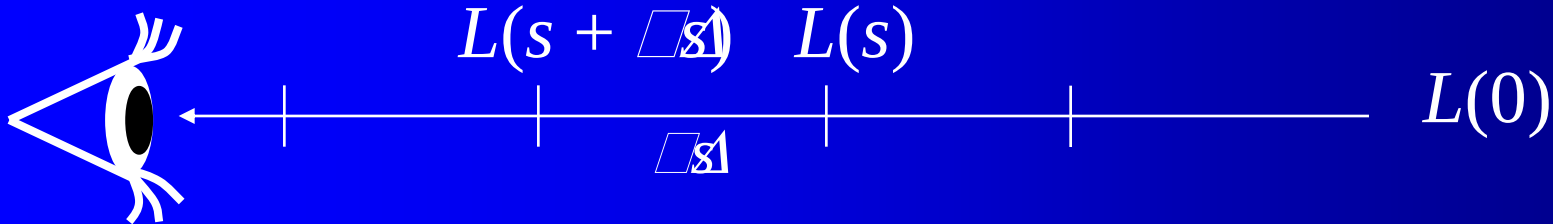
$$\frac{dL(s)}{ds} = -L(s) \kappa(s) + L^e(s) + L^{in\text{scatter}}(s)$$

Modeldás fénynyel

$$L(s) = L(0) \exp\left(-\int_0^s \kappa(s') ds'\right)$$

(emiszió nélkül)

Sugárzás (ray tracing) o



$$L(s + \Delta s) = L(s) - L(s) \Delta s + (L^e(s) + L^{inscatter}(s) \Delta s)$$

$\mu(s)$

$C(s)$

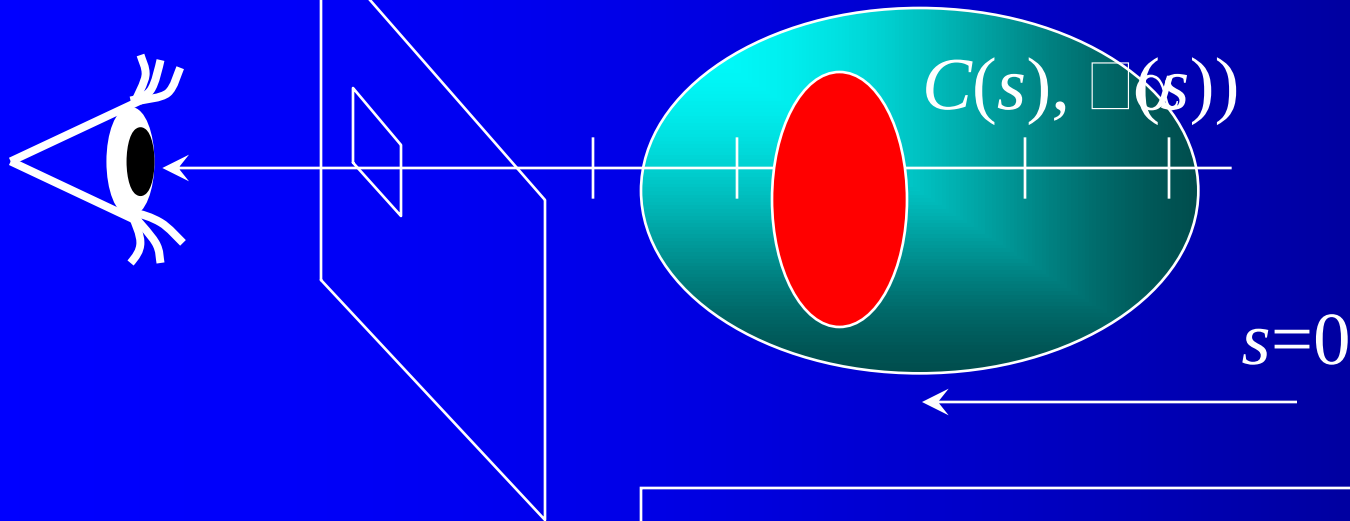
előpáncírozás

$$L(s + \Delta s) = (1 - \mu(s) \Delta s) L(s) + C(s)$$

átviteli szóság

Back-to-front ray marching

$$L(s + \Delta s) = (1 - \alpha(s))L(s) + C(s)$$



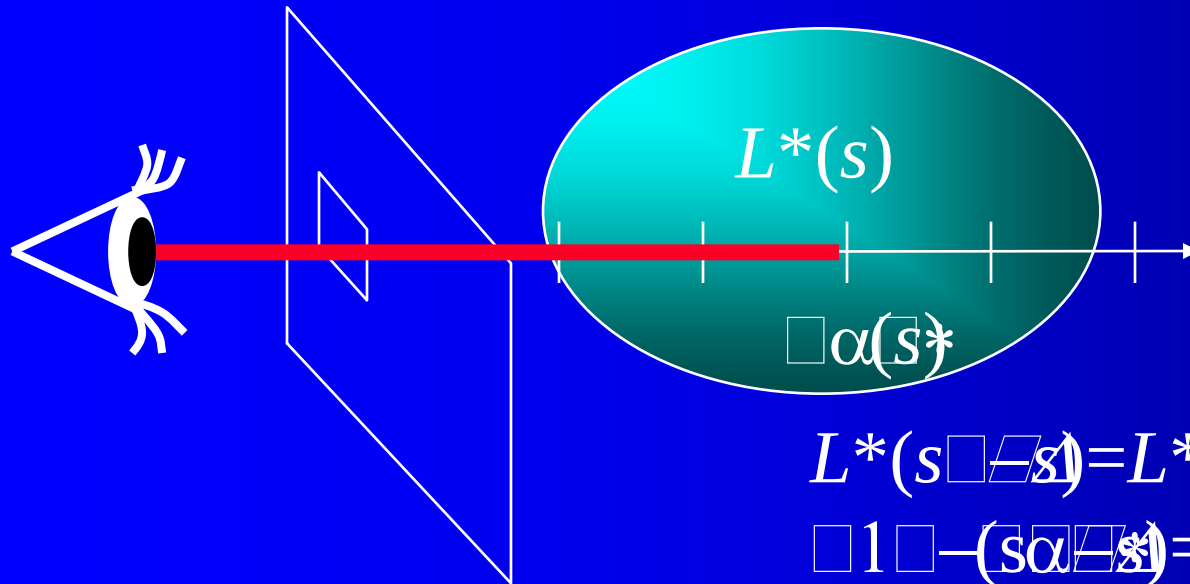
$L = L_0;$

for($s = 0; s < T; s += \Delta s$) {

$L = (1 - \alpha(s))L + C(s);$

}

Front-to-back ray marching



$$L^*(s - \Delta s) = L^*(s) + (1 - \alpha(s)) C(s)$$

$$1 - \alpha(s - \Delta s) = (1 - \alpha(s)) (1 - \alpha(s))$$

$L^* \leftarrow \alpha * 0; \quad \leftarrow$

for($s \leftarrow T; s > 0; s \leftarrow s - \Delta s$) {

$L^* \leftarrow L^* + (1 - \alpha) C(s); \quad \leftarrow$

$\alpha \leftarrow (1 - \alpha) * (1 - \alpha(s)) \alpha \leftarrow$

if ($\alpha \leftarrow \alpha > \text{break}; -\epsilon$)

}

Voxel színe átszűrési

Transfer funk: $(C, \alpha) = T(v, f, \vec{u}, g)$

● Rövid $(t, g) = T(v(x, y, z), s)$

– opacitás $\alpha(x, y, z)$ és $s =$

$L(0) = I$, $e.g. C(s) = 0$ békén

● Klasszikus ár:

– opacitás α és s átlósan

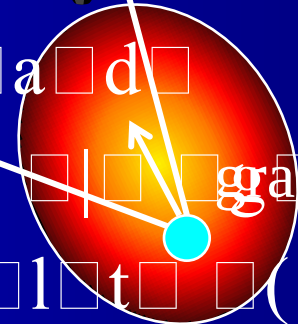
$\epsilon =$ árnyalás n oldal

● normál $=$ g rad

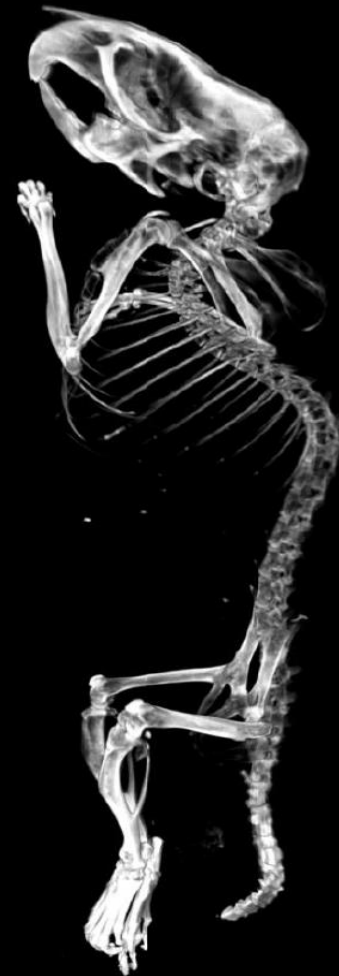
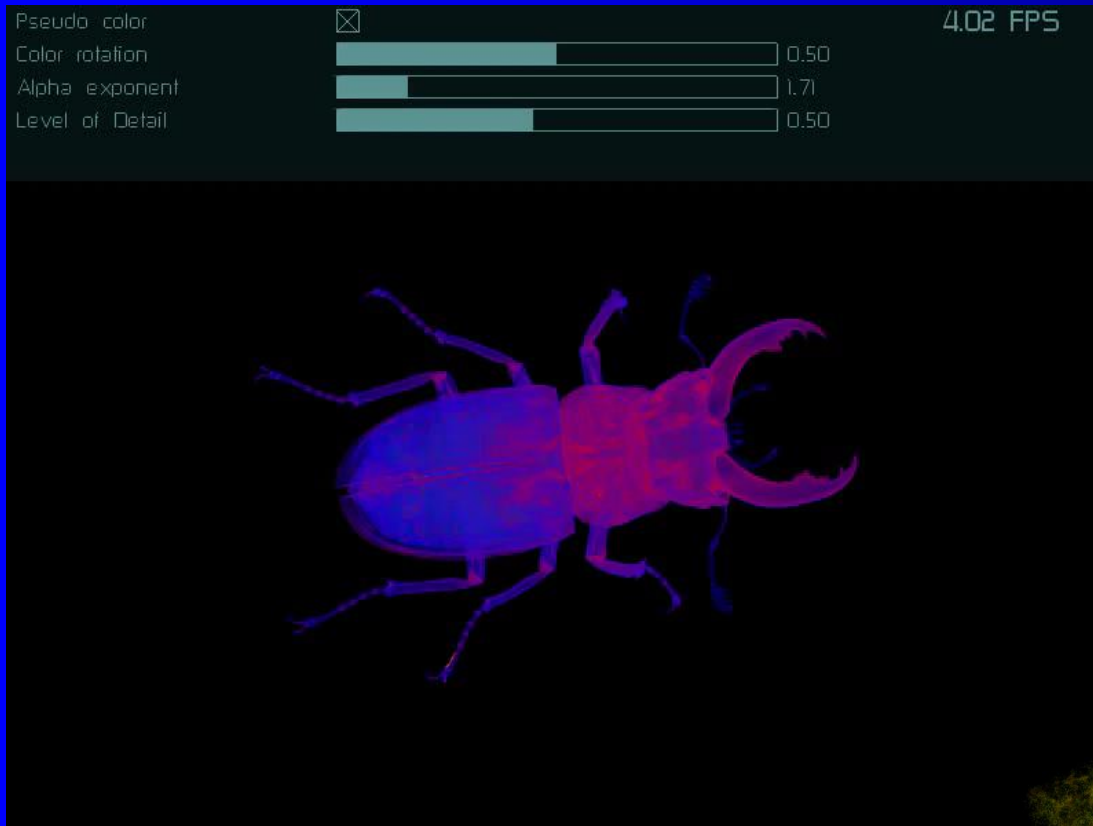
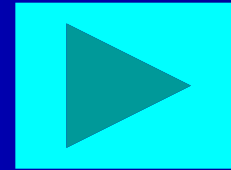
● opacitás α $\ast$ $=$ g rad

● Magasabb rend C d e r i v α l t (g)

● Transzlucens anyagok (subsurface scattering)

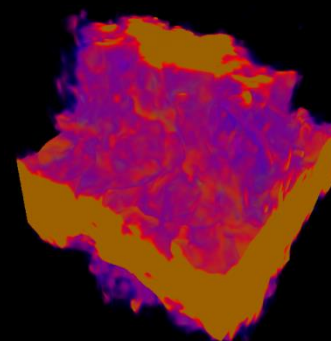
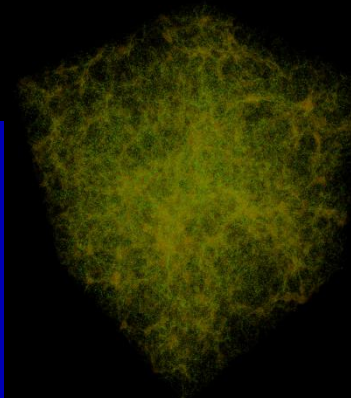


$$(C, \alpha) = T(v) \cdot s$$



$$\alpha = \text{pow}(v/v_{\text{max}}, \text{aexp})$$

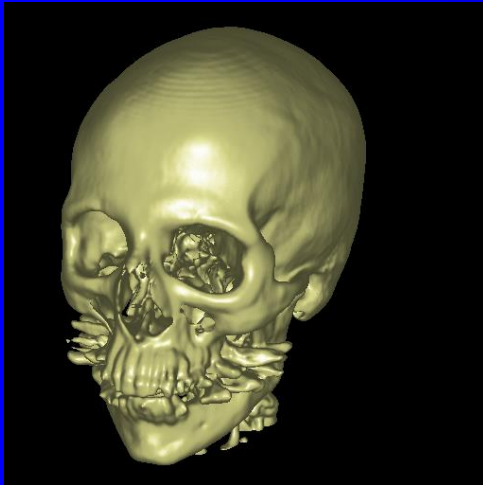
$$C = \text{HLS}((v/v_{\text{max}} + \text{rot}) \cdot 360, 0.5, 1)$$



Klasszikus BRDF modellek

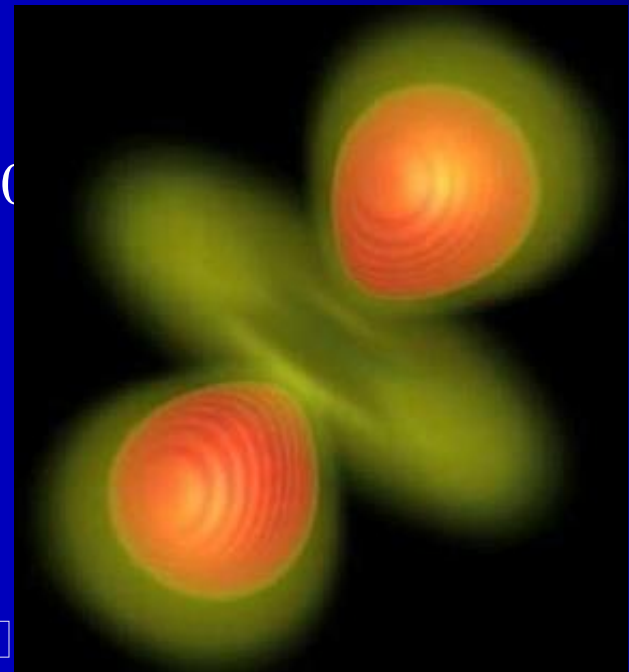
First hit ray casting:

□D□i□f□f□ú□z□+□P□h□o□n□g□ □á□r□n□y□a□l□á□s



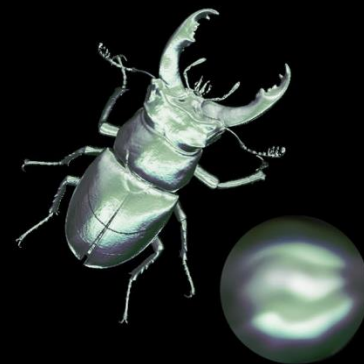
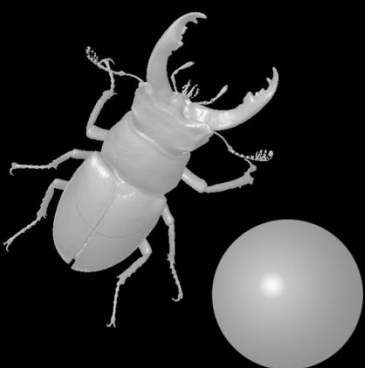
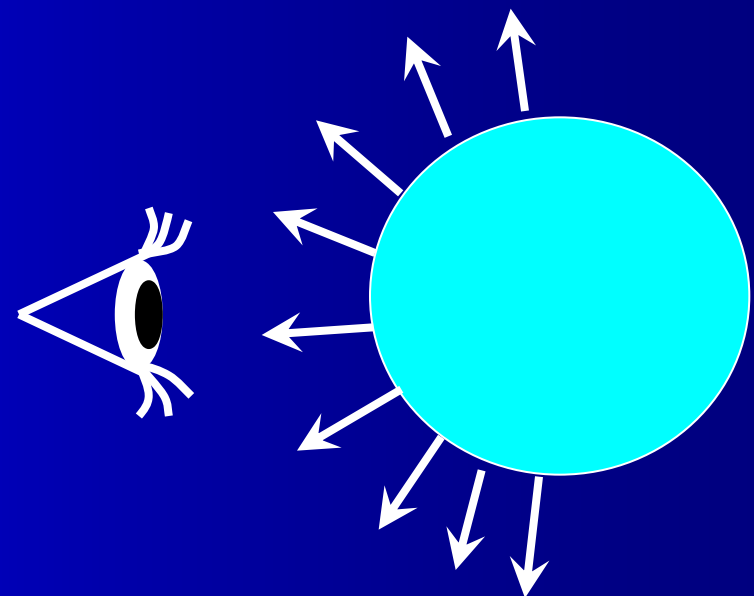
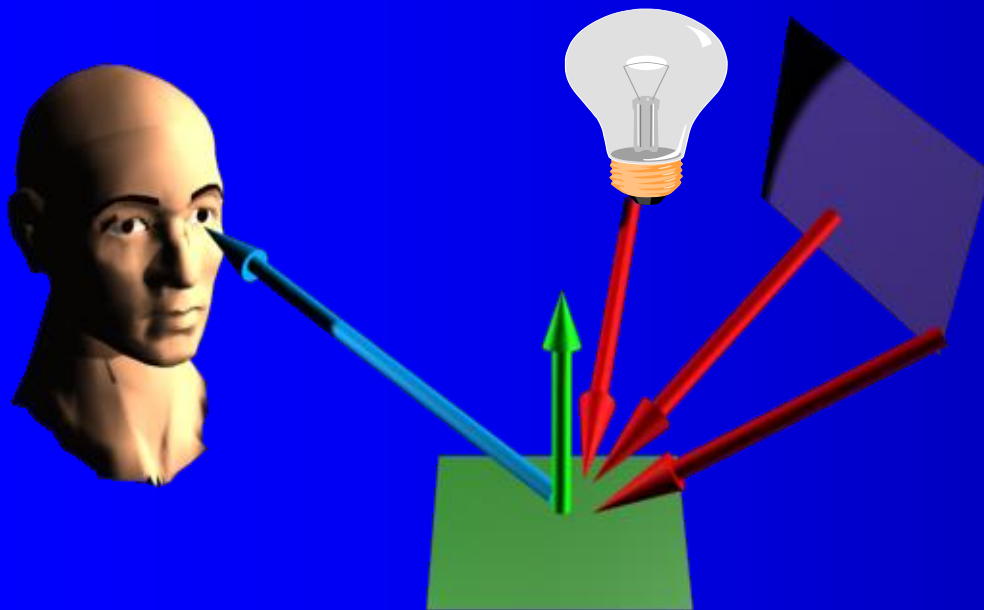
v □k□é□t□ □o□s-záttál láyt-bá az □á□k

Csont □α:□ □l□,□ □m□á□s□é□ □C

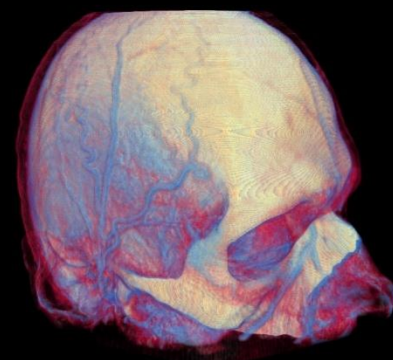
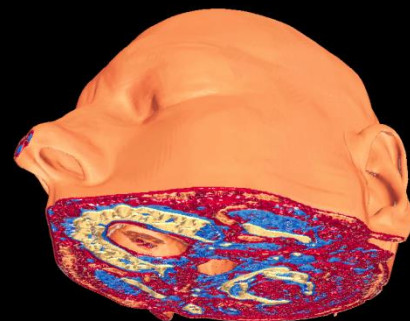
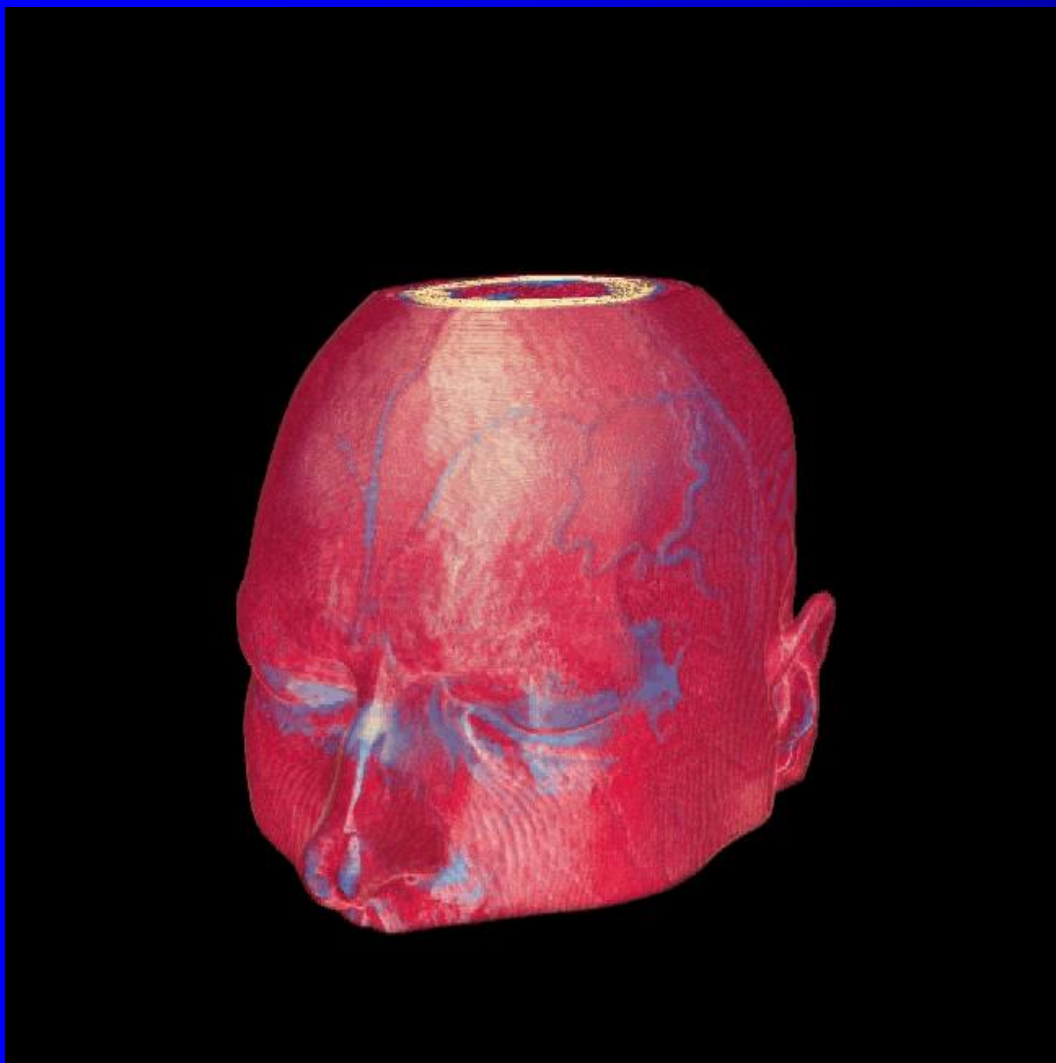


Csont □α□l□,□=□m□á□s□é□ □H□é□m□á□

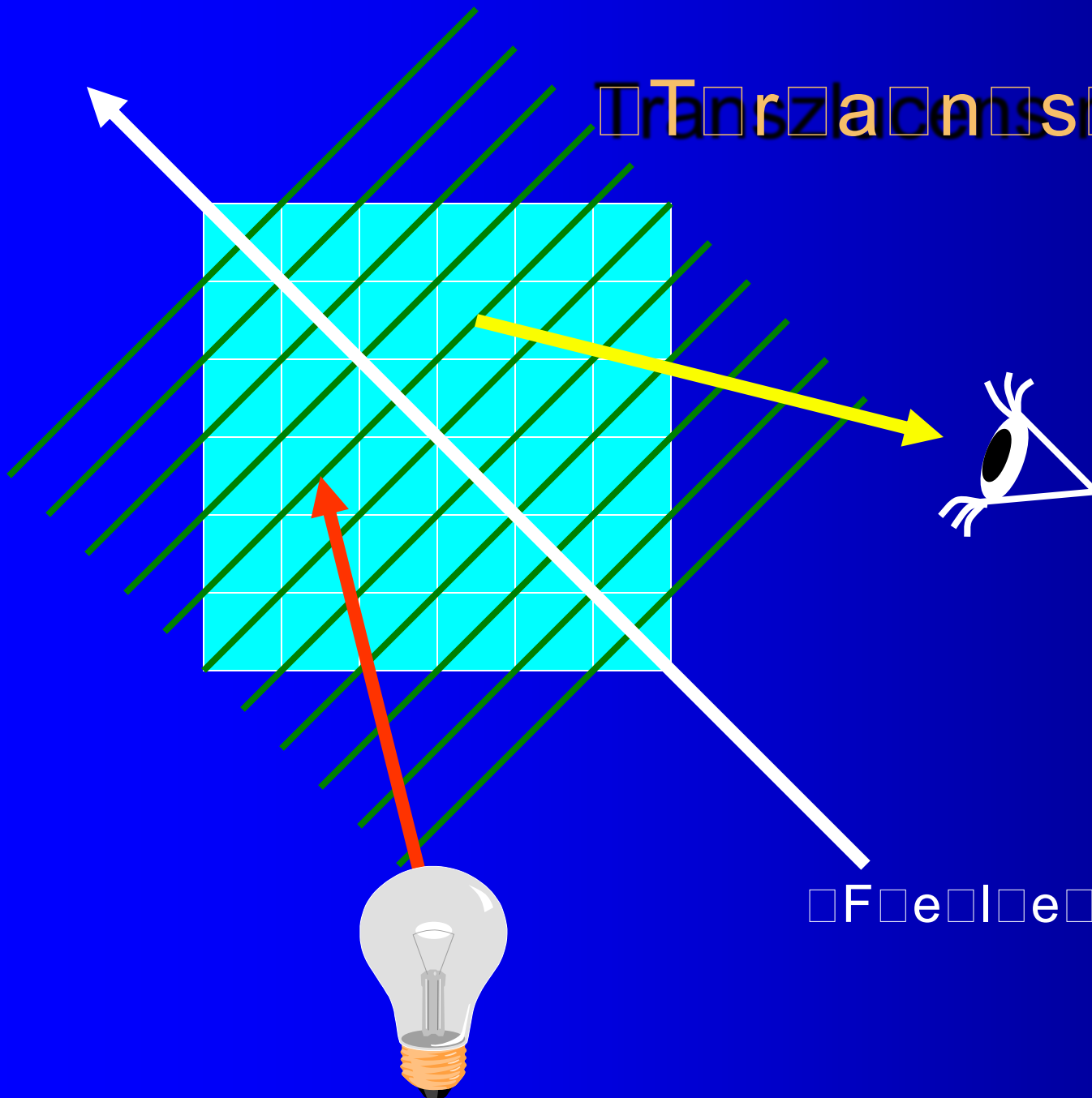
Illustratív vizualizáció



Illustratív vizualizáció



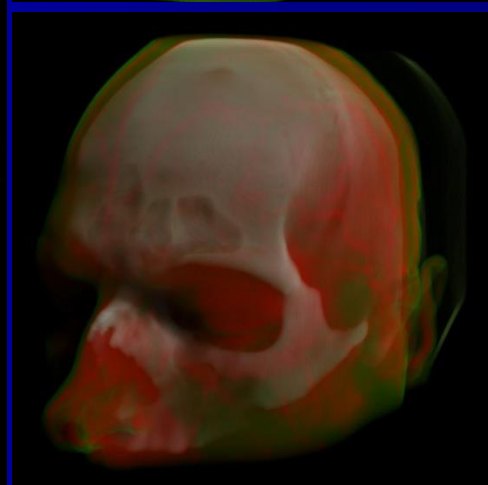
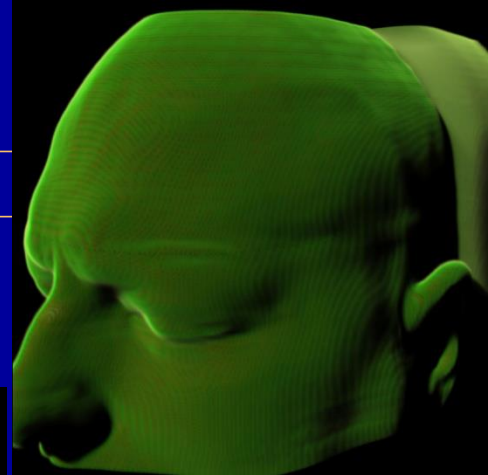
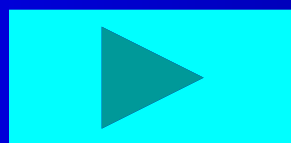
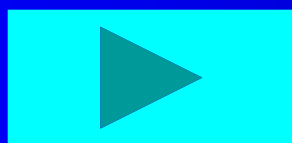
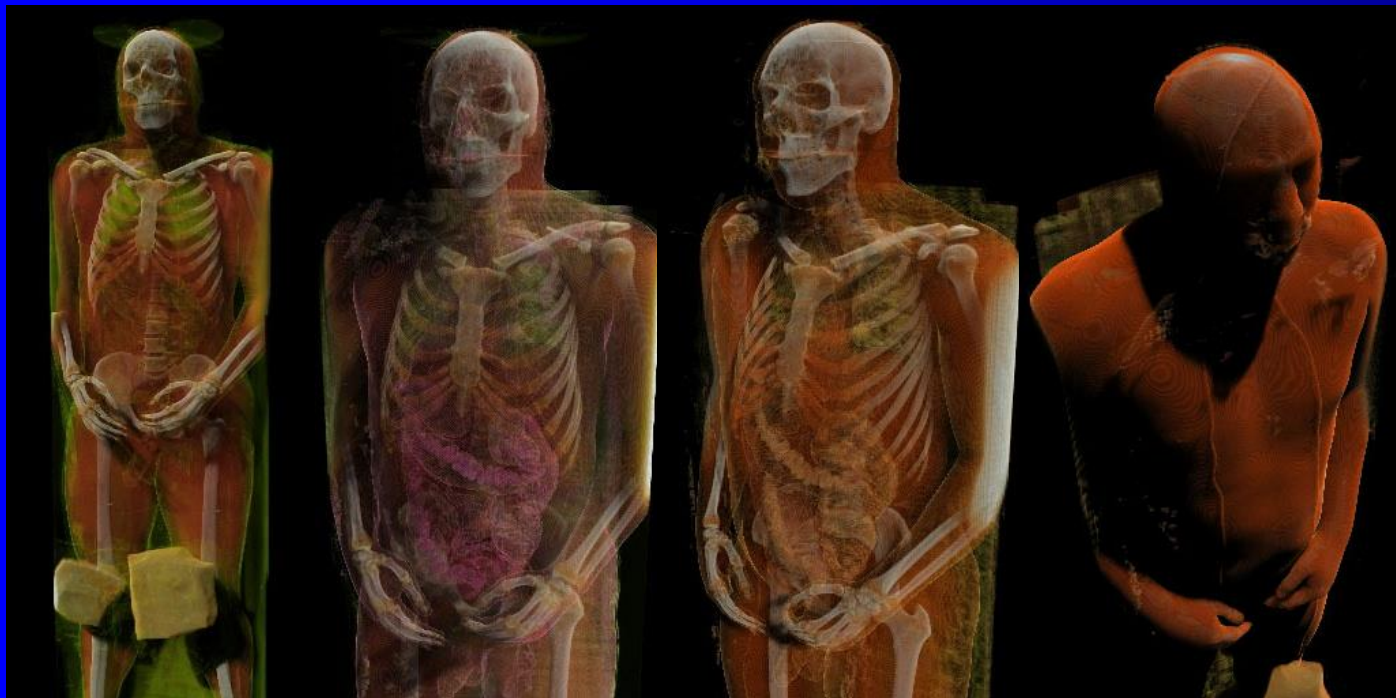
Transzparensszegélyezés



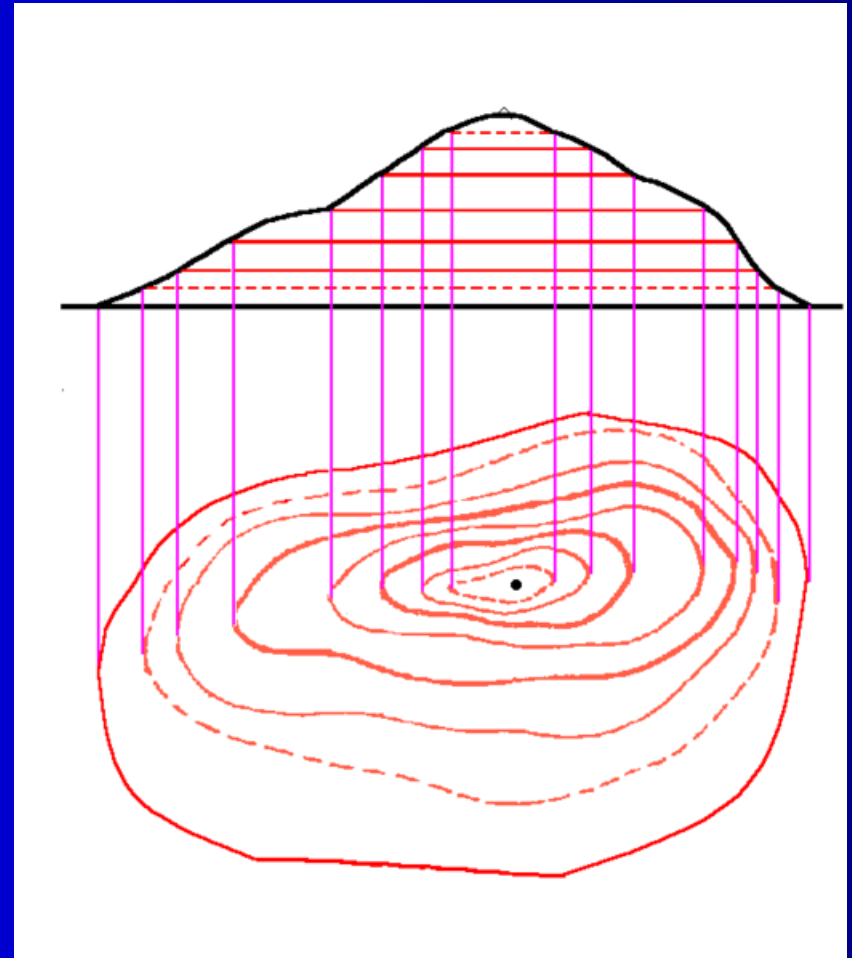
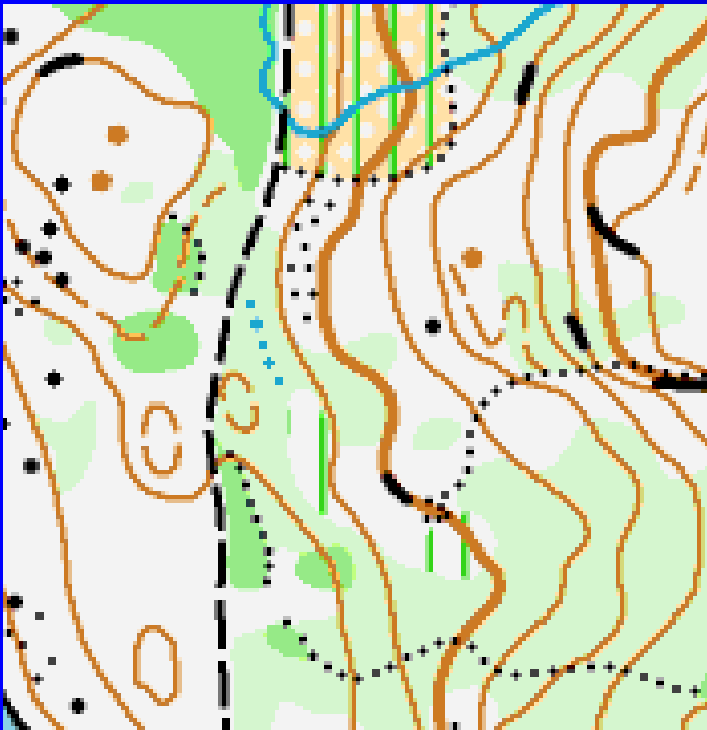
Félez

vék

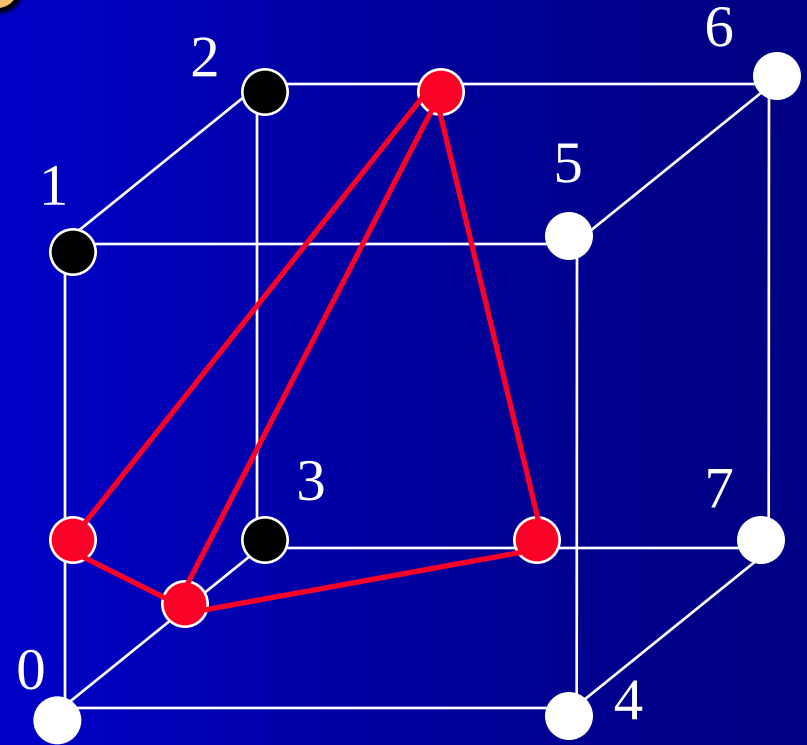
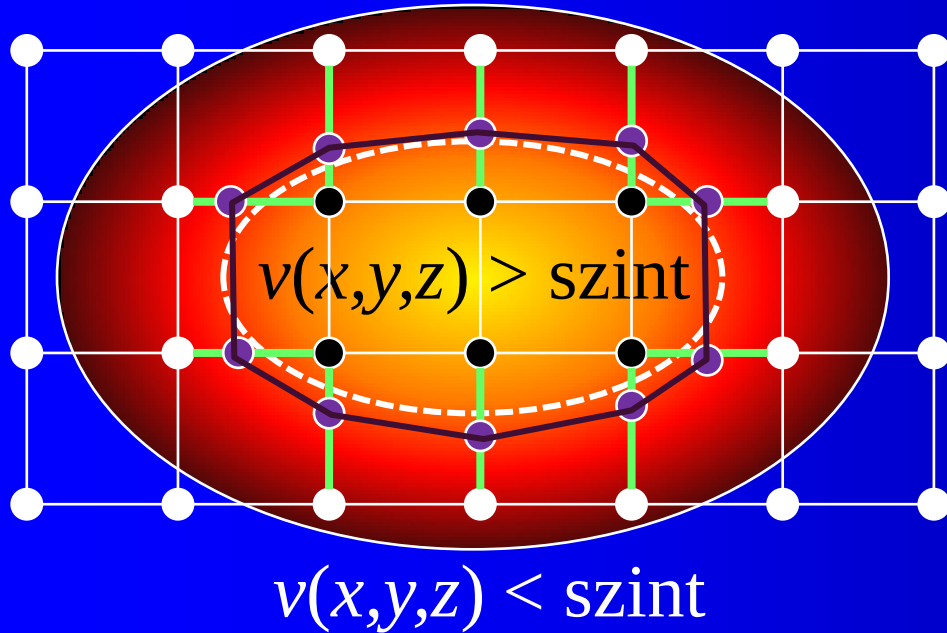
Transzlucens megjelenítés



Szintvonal, szintfelület



Marching cubes



0	0;
1	
14	2; 0-1; 0-3; 2-6; 0-3; 3-7; 2-6
15	
255	0;

Eset: $00001110_2 = 14$

14 ←

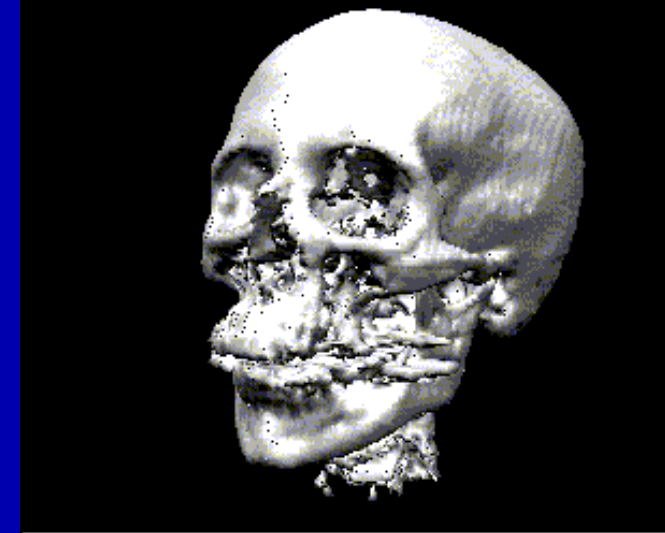
Másírók



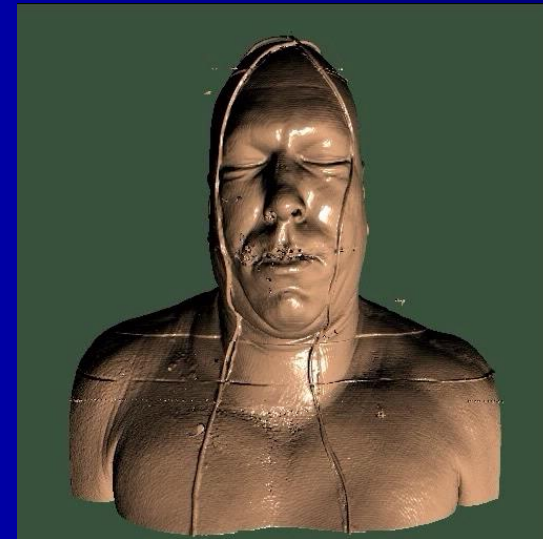
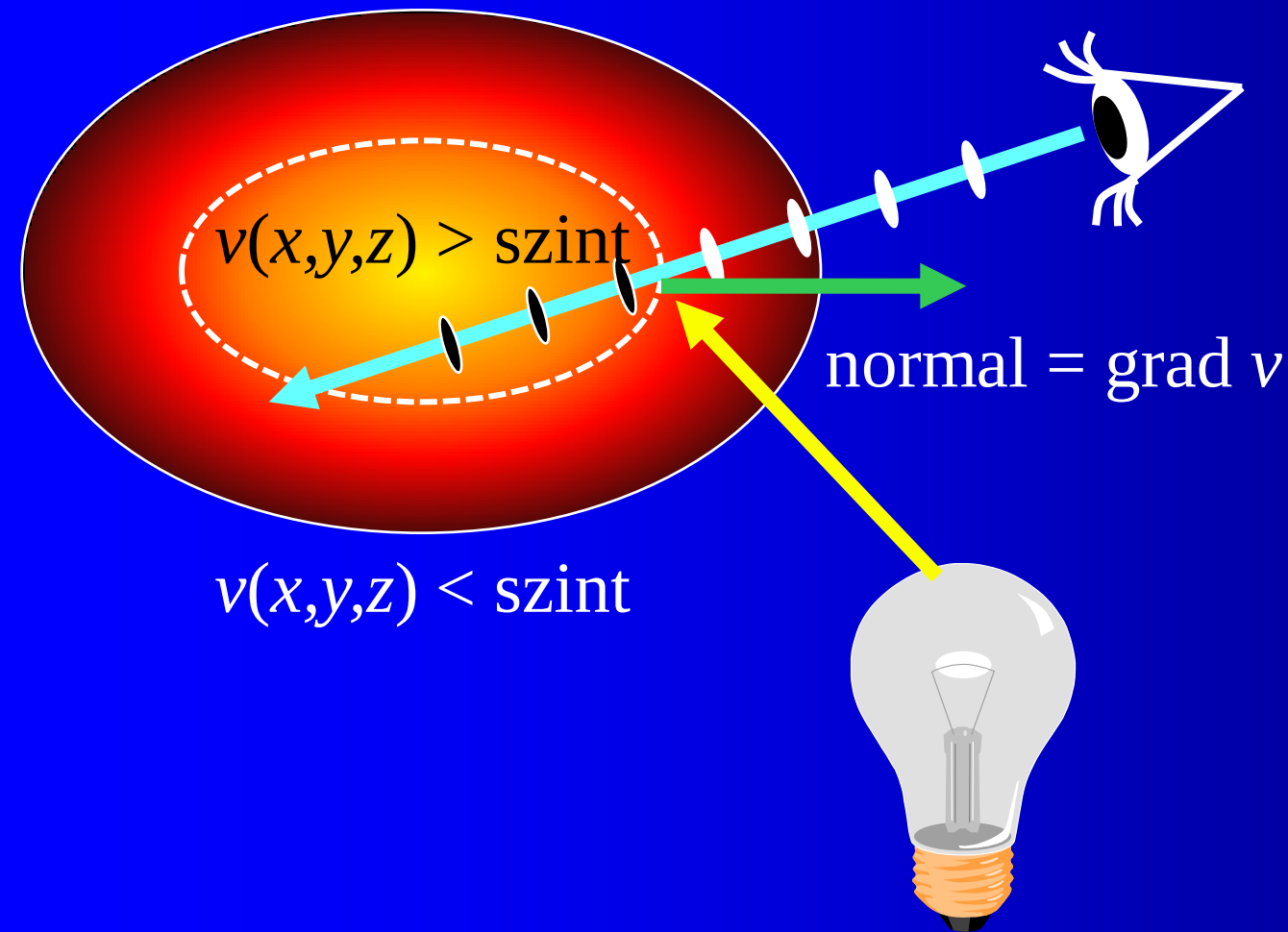
Szintér



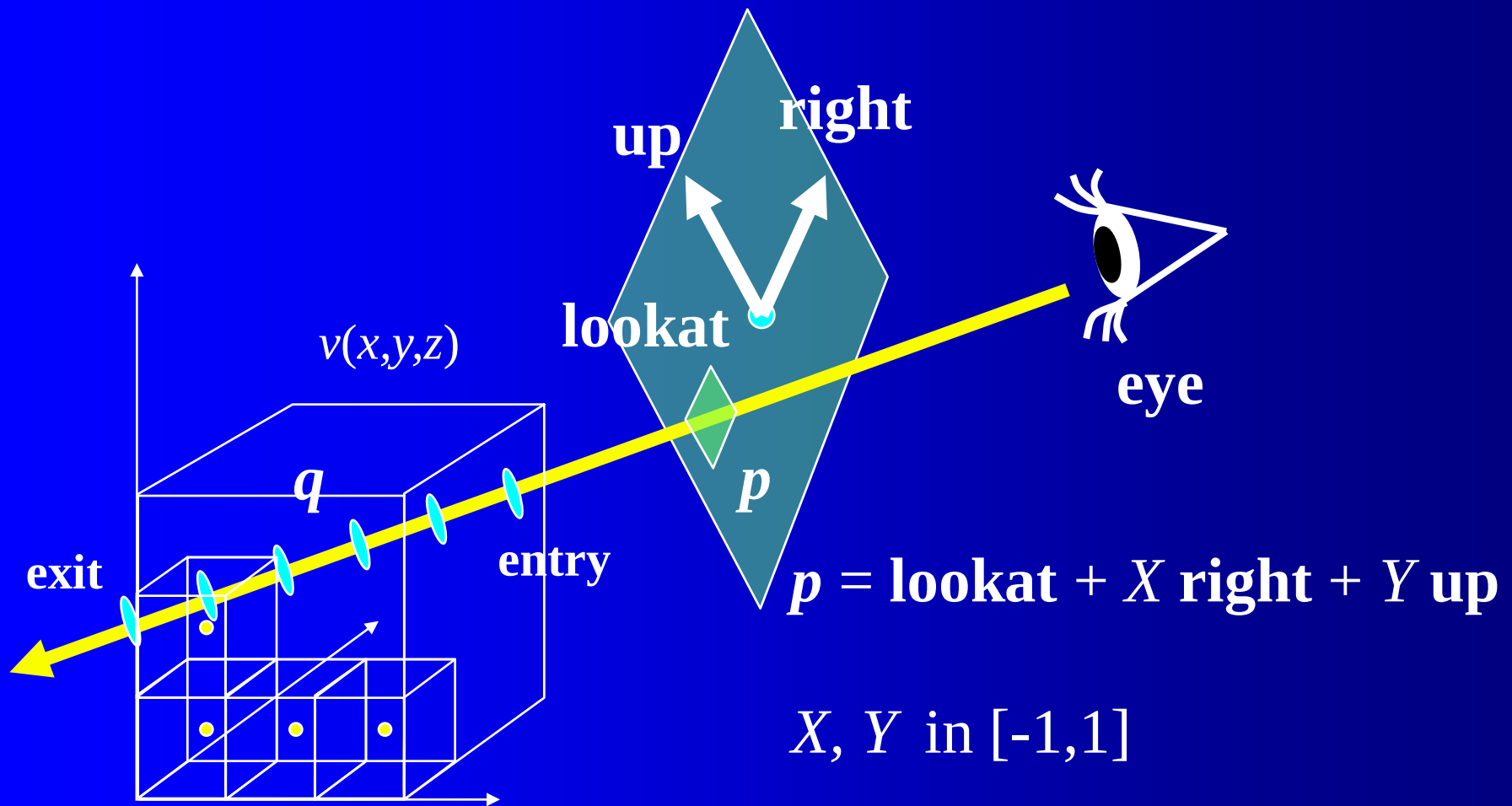
Szintér



First hit (isosurface) ray casting



GPU first hit ray-casting

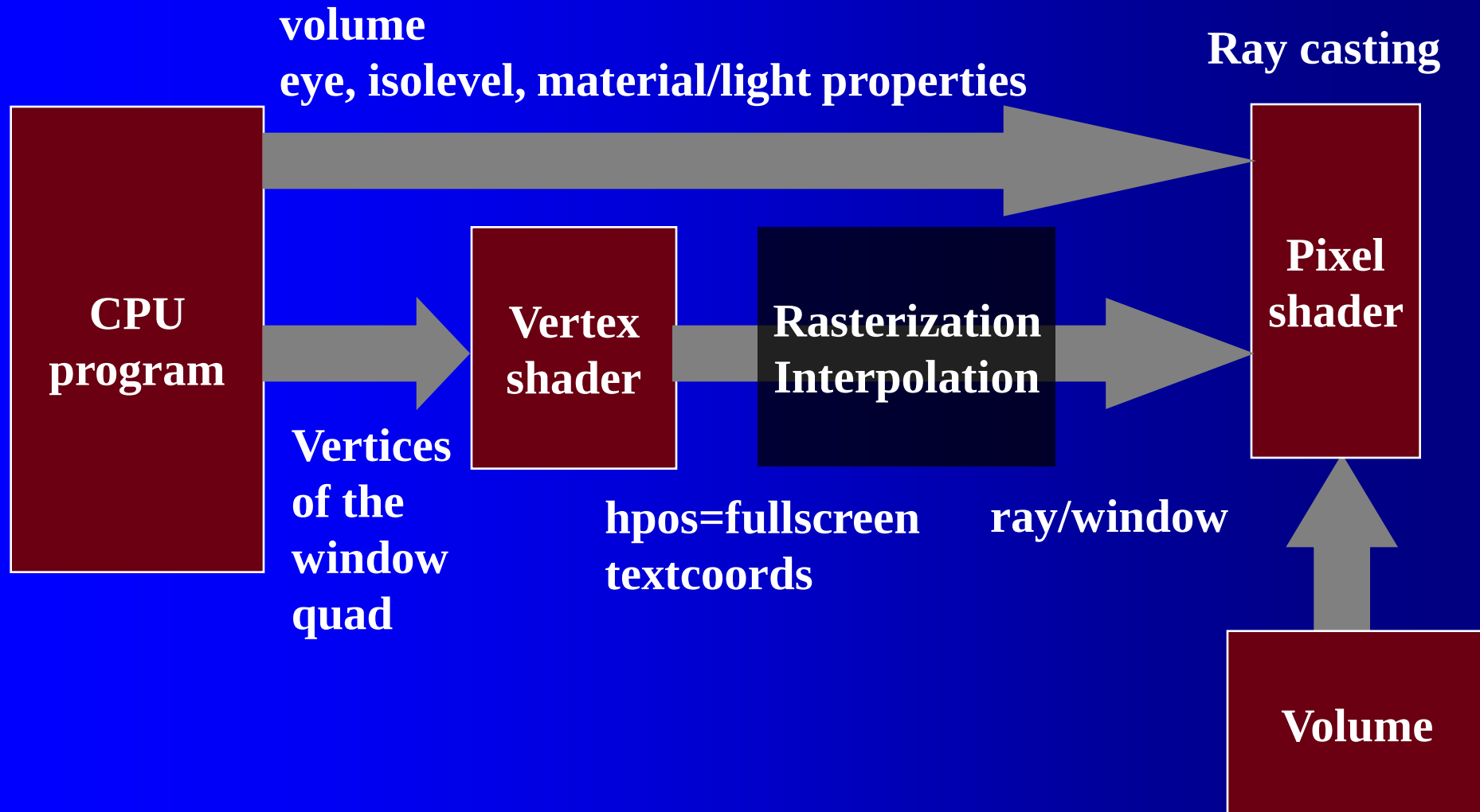


□E□g□y□s□é□g□ □k□o□c□k□a□ □3□D□ □t□e□x□

CPU first hit ray-casting

```
For each pixel ← Full screen quad
  Find pixel center p ← Interpolation
  raydir = normalize(p - eye); from the corners
  Find exit and entry ← Clipping
  for(t = entry; t < exit; t+=dt) {
    q = eye + raydir * t;
    if (volume[q] > isovalue) break;
  }
  normal vector estimation; ← central differences
  illumination
}
```

GPU Isosurface ray-casting



CPU program - OpenGL display

```
void Display( ) {  
    // PASS: non uniform parameters  
    glBegin( GL_QUADS );  
    Vector p = lookat - Right + Up;  
    glTexCoord3f(p.x, p.y, p.z); glVertex3f(-1, 1, 0);  
    p = lookat - Right - Up;  
    glTexCoord3f(p.x, p.y, p.z); glVertex3f(-1, -1, 0);  
    p = lookat + Right - Up;  
    glTexCoord3f(p.x, p.y, p.z); glVertex3f(1, -1, 0);  
    p = lookat + Right + Up;  
    glTexCoord3f(p.x, p.y, p.z); glVertex3f(1, 1, 0);  
    glEnd();  
}
```



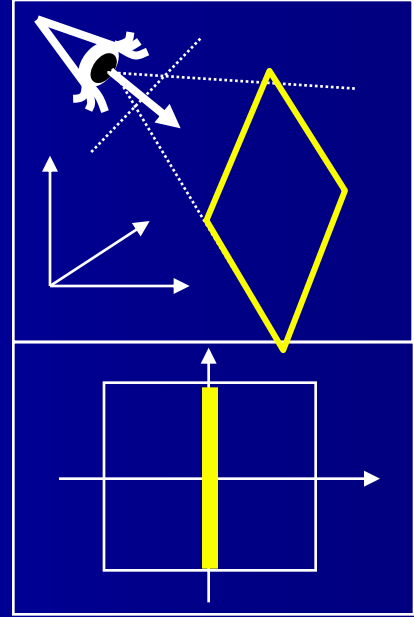
Camera window as texture coordinates



Full screen quad

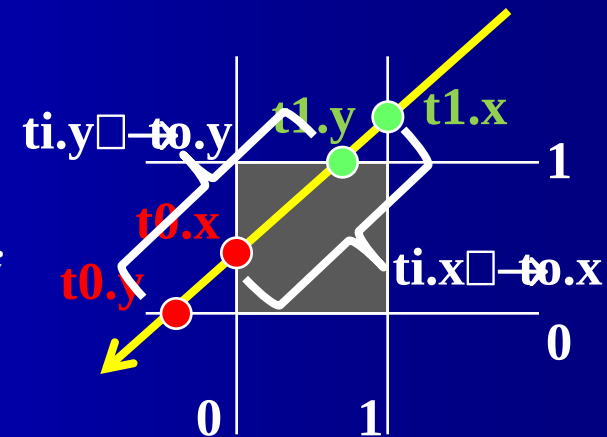
Ray casting: vertex shader

```
void VertexShader(  
    in  float4 hPosIn      : POSITION,  
    in  float3 wPosIn      : TEXCOORD0,  
  
    out float4 hPosOut     : POSITION,  
    out float3 wPosOut     : TEXCOORD0 )  
{  
    hPosOut = hPosIn;  
    wPosOut = wPosIn;  
}
```



Ray casting: fragment shader

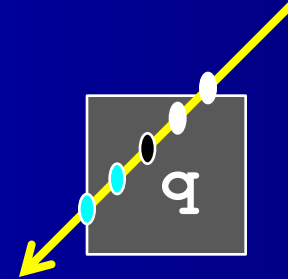
```
void FragmentShader( in float3 p : TEXCOORD0, // point on window
                      uniform float3 eye,
                      uniform sampler3D volume, // voxels
                      uniform float isolevel,
                      uniform float3 lightdir, lightint, kd
                      out float3 color : COLOR )
{
    float3 raydir = normalize(p - eye);
    float3 t0 = (float3(0,0,0)-eye)/raydir;
    float3 t1 = (float3(1,1,1)-eye)/raydir;
    float3 ti = min(t0, t1);
    float3 to = max(t0, t1);
    float entry = max(max(ti.x, ti.y), ti.z);
    float exit = min(min(to.x, to.y), to.z);
    color = float(0, 0, 0);
    bool found = (exit <= entry);
}
```



Content 'd...

Ray casting fragment shader control

```
if ( !found ) {
    float3 q, normal;
    float dt = (exit - entry) / STEPS;
    for(t = entry; t < exit; t += dt) {
        if ( !found ) {
            q = eye + raydir * t;
            if (tex3D(volume, q).r > isolevel) found = true;
        }
    }
    if ( found ) {
        normal.x = tex3d(volume, q + float3(1/RES,0,0)) -
                  tex3d(volume, q - float3(1/RES,0,0));
        normal.y = tex3d(volume, q + float3(0,1/RES,0)) -
                  tex3d(volume, q - float3(0,1/RES,0));
        normal.z = tex3d(volume, q + float3(0,0,1/RES)) -
                  tex3d(volume, q - float3(0,0,1/RES));
        normal = normalize( normal );
        color = lightint * kd * max(dot(lightdir, normal), 0);
    }
}
```

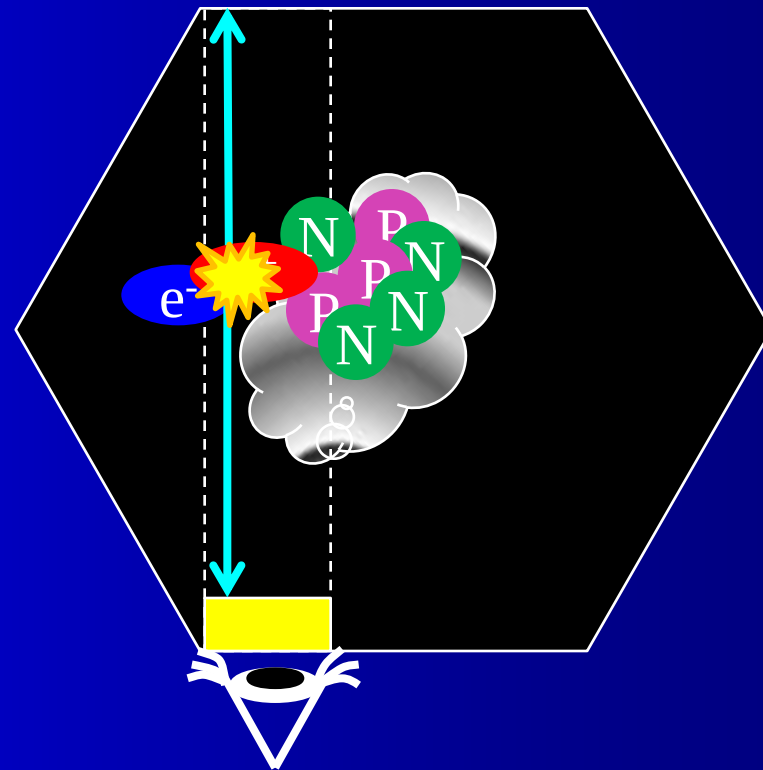
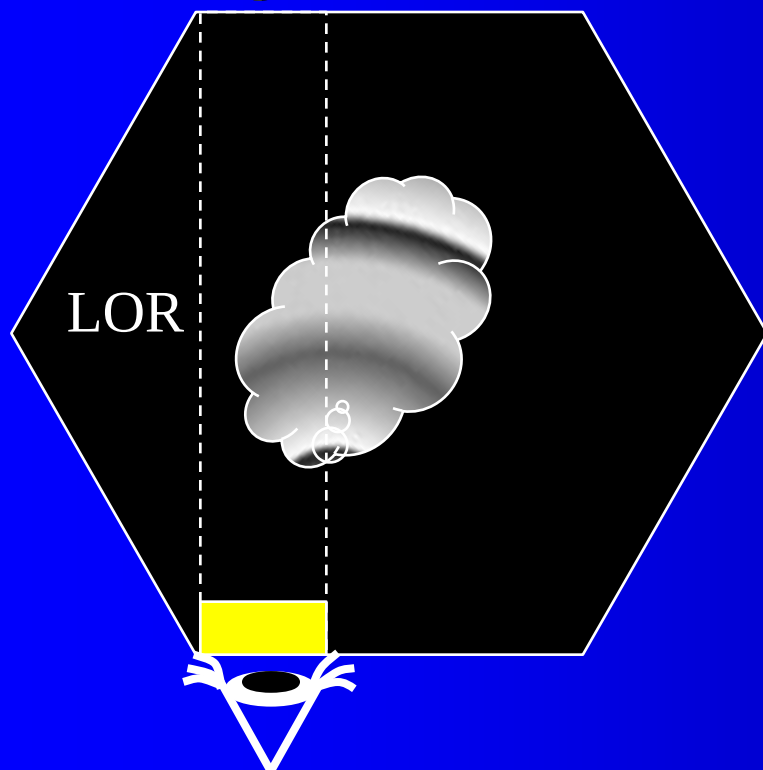


Tomografia



Abstrakció

Emisszió

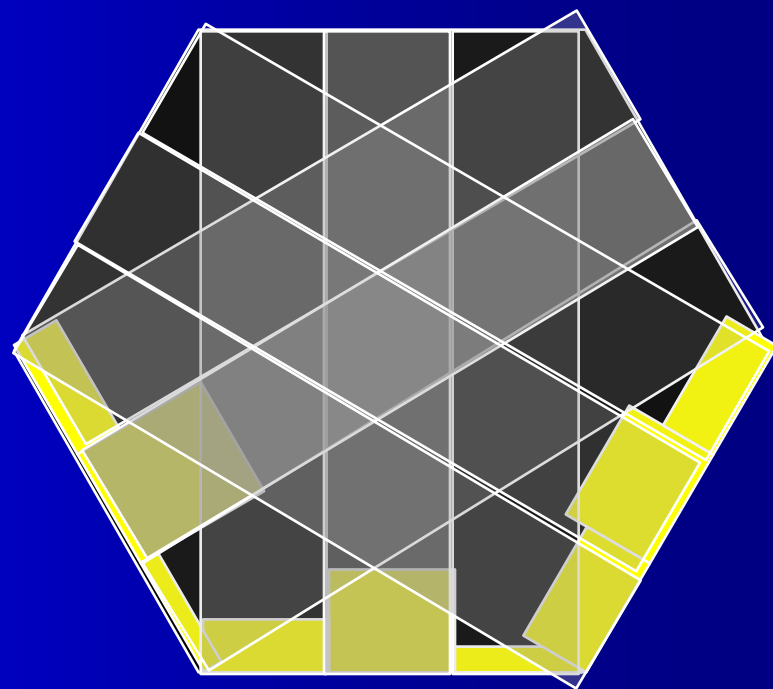
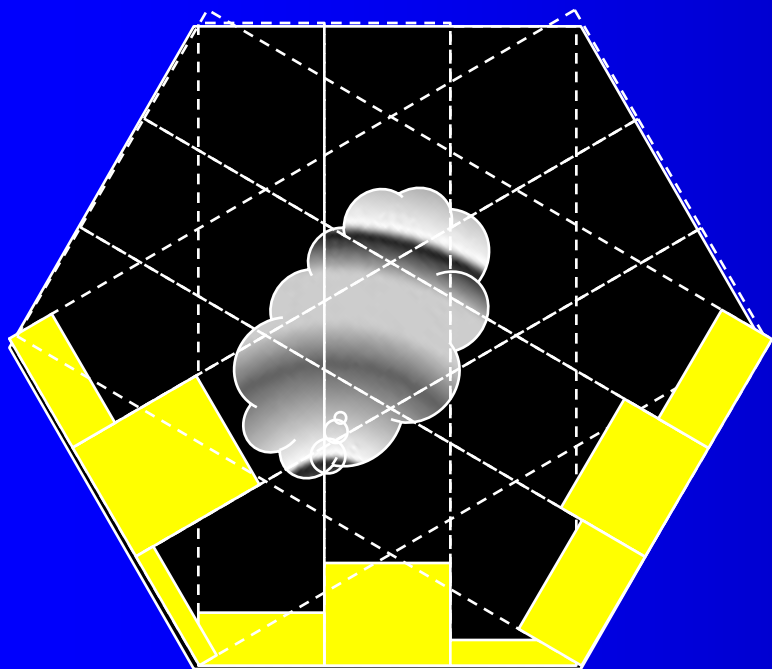


$$L(s) = L(0) \exp\left(-\int_0^s \mu(\mathbf{r}) ds\right)$$

$$\int_0^s \mu(\mathbf{r}) ds = -\log(L(s)/L(0))$$

$$L(s) = \int e(s) ds$$

Tomográfia és rekonstrukció

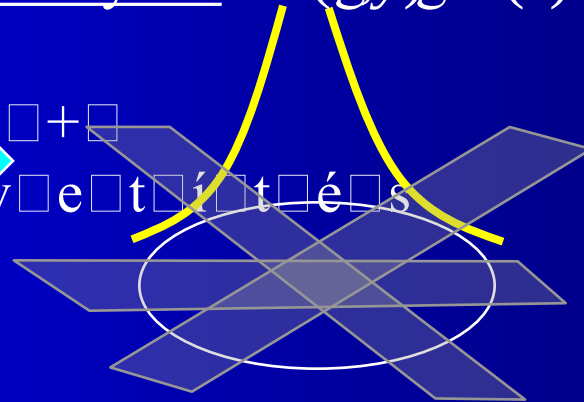
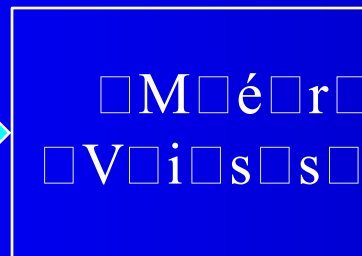


Sűrűségvizualizáció

(FBP=Filtered backprojection)

$$i(x,y) = \delta(x,y)$$

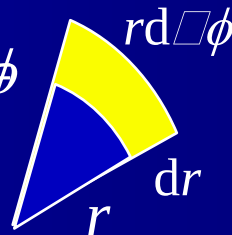
$$\text{Súlyfüggvény: } w(x,y) = w(r) = \frac{1}{r}$$



R kör

$$\iint w(x,y) dx dy = \int_0^{2\pi} \int_0^R w(r) r dr d\phi$$

$$= 2\pi \int_0^R w(r) r dr = R$$



Körrekező Fourier transzformáció:

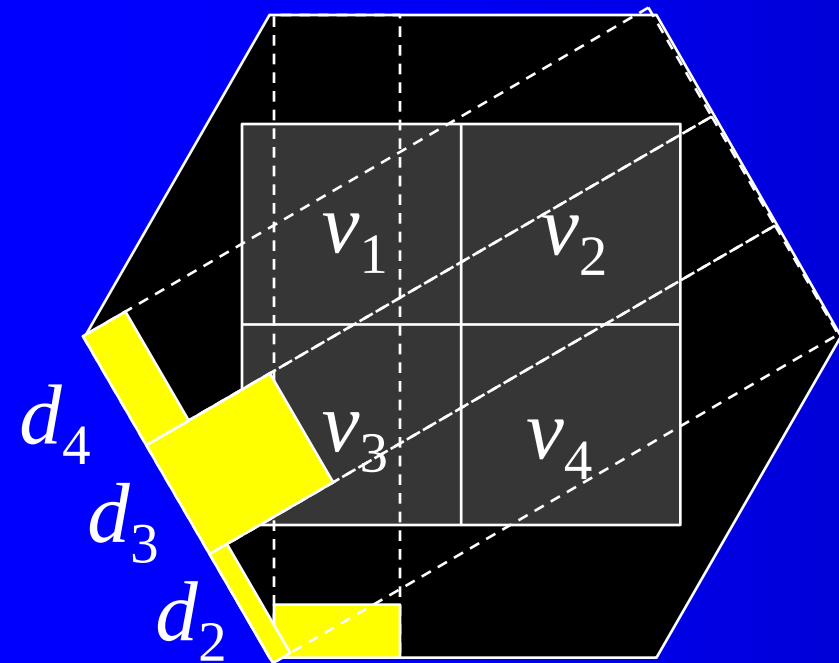
$$o(x,y) = i(x,y) \otimes w(x,y) \rightarrow F_x F_y o[\phi, \psi] = F_x F_y i[\phi, \psi] F_x F_y w[\phi, \psi]$$

Rámpa és C

$$F_x F_y i[\phi, \psi] = F_x F_y o[\phi, \psi] \cdot \frac{1}{|\phi|}$$

$$\frac{1}{|\phi|}$$

Algebrai visszavetítés



$$d_1 = A_{11}v_1 + A_{13}v_3$$

Lin egyenlet ($V < D$):

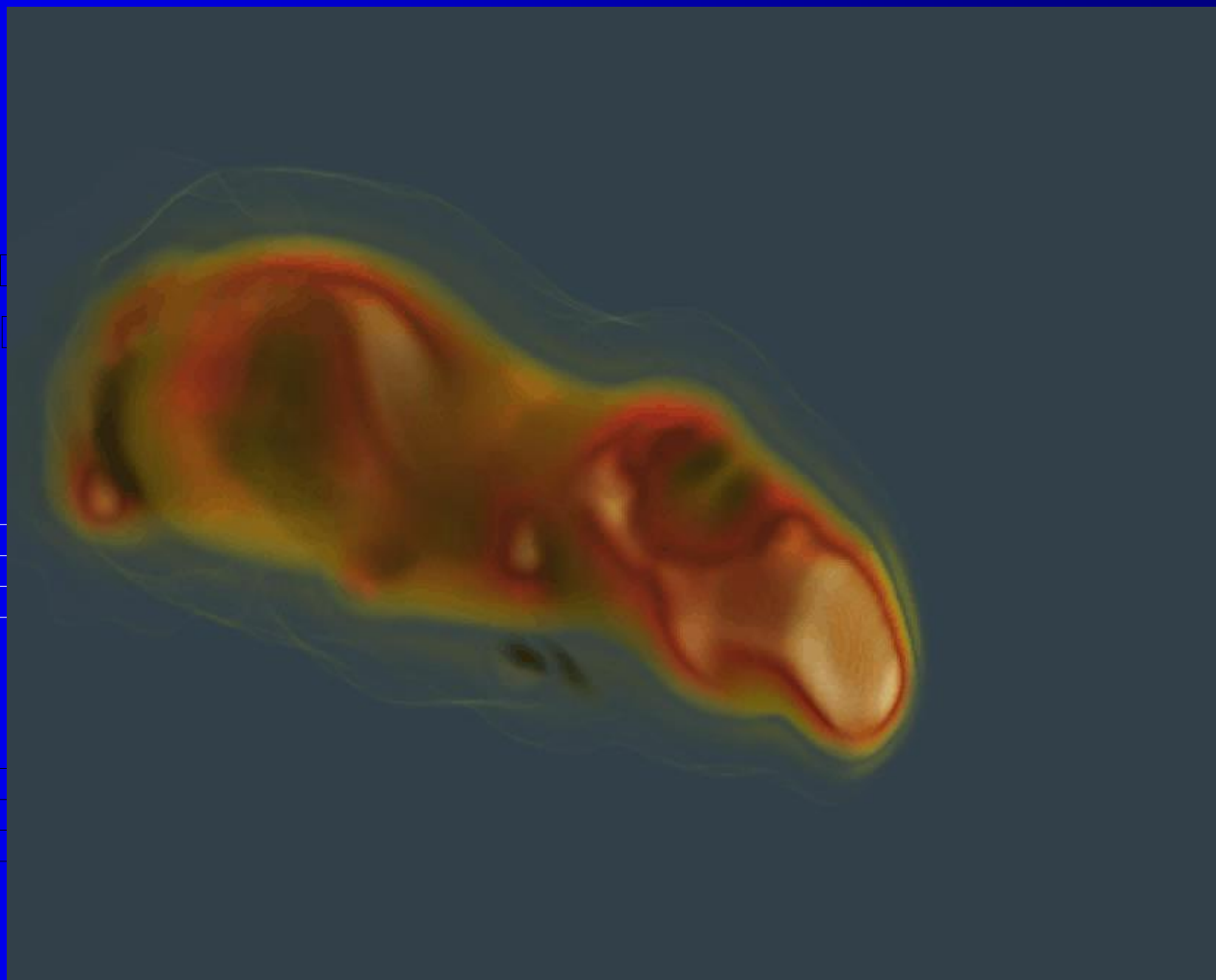
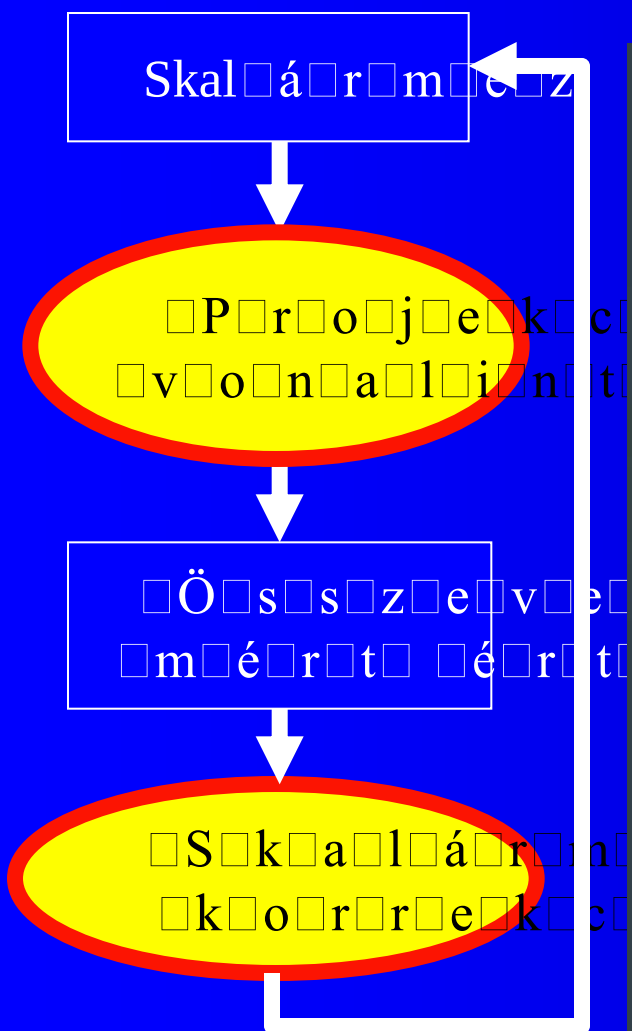
$$\begin{aligned} \underset{(D)}{\mathbf{d}} &= \underset{(D \times V)}{\mathbf{A}} \underset{(V)}{\mathbf{v}}, & \mathbf{d} &= [d_1, \dots, d_D] \\ & & \mathbf{v} &= [v_1, \dots, v_V] \end{aligned}$$

Moore pseudoinverse:

$$\underset{(V \times D)}{\mathbf{A}^T} \underset{(D)}{\mathbf{d}} = \underset{(V \times V)}{\mathbf{A}^T \mathbf{A}} \underset{(V)}{\mathbf{v}}$$

$$\mathbf{v} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{d} = \mathbf{A}^+ \mathbf{d}$$

Iteratív sémák



Fraktálok

Szirmay-Kalos László

Fraktálak

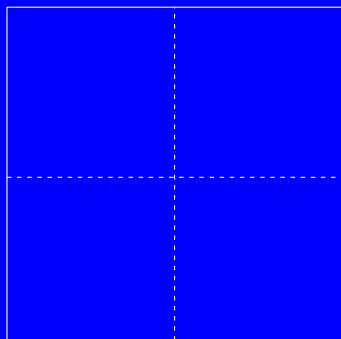
Házasdorff-dimensió

$$N = 2$$



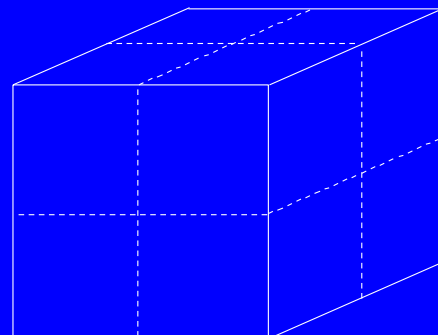
$$r = \frac{1}{2}$$

$$N = 4$$



$$r = \frac{1}{2}$$

$$N = 8$$



$$r = \frac{1}{2}$$

$$N = 1/r^D$$

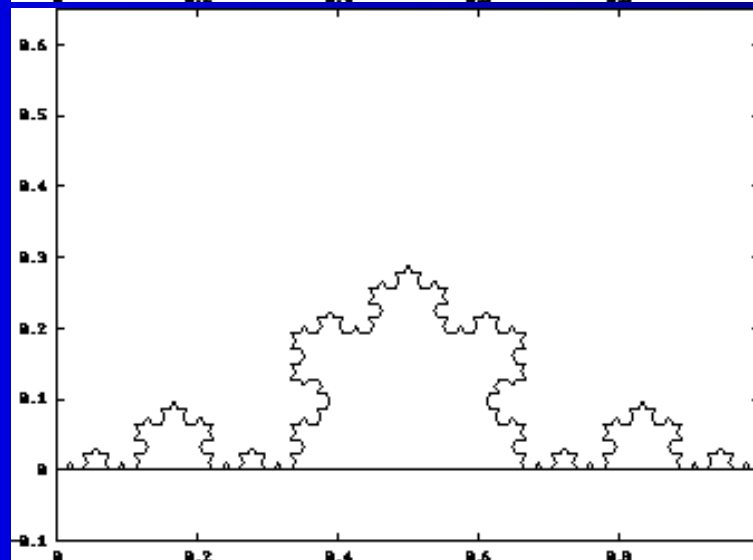
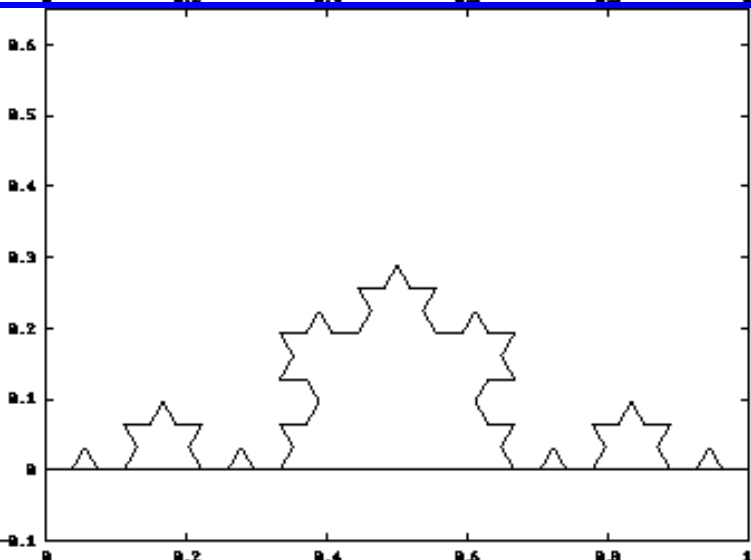
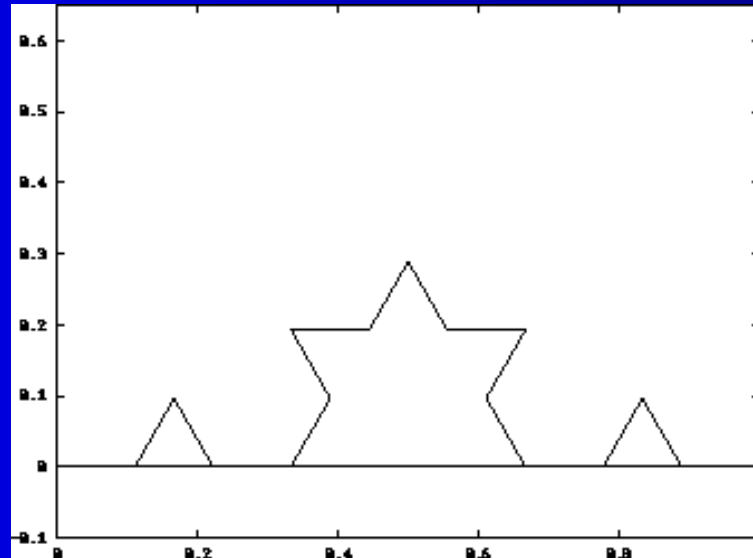
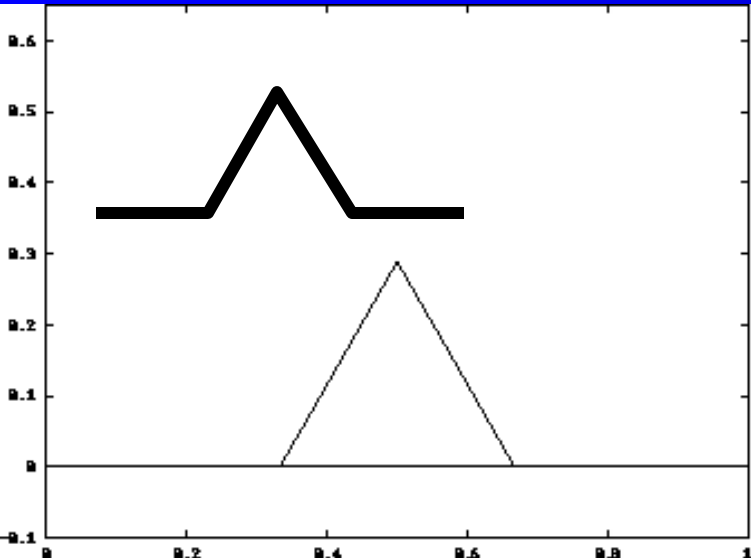
$$D = (\log N) / (\log 1/r)$$

Kochsche Schneeflocke

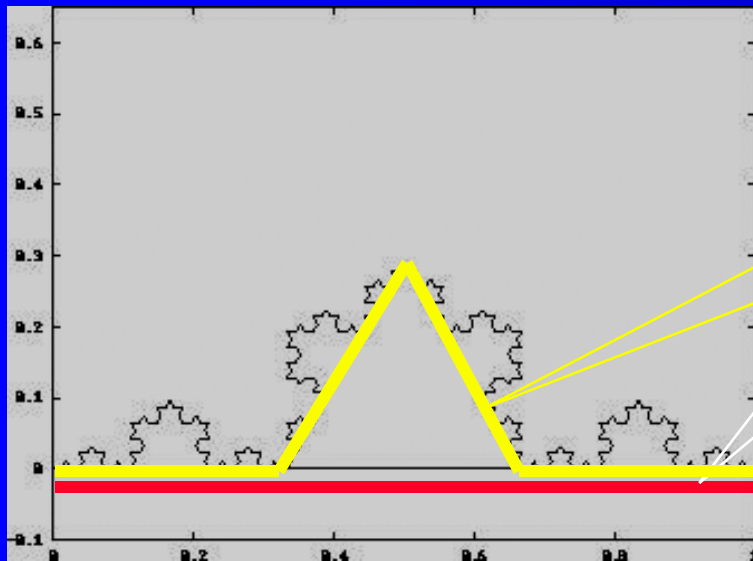
$$D = (\log 4) / (\log 3) = 1.26$$

$$N = 4,$$

$$r = 1/3$$



Négyes oldalú objektumok dimenziója



$V_o(l)$ a l zdb

1	1
$r = 1/3$	$N = 4$

r^2

N^2

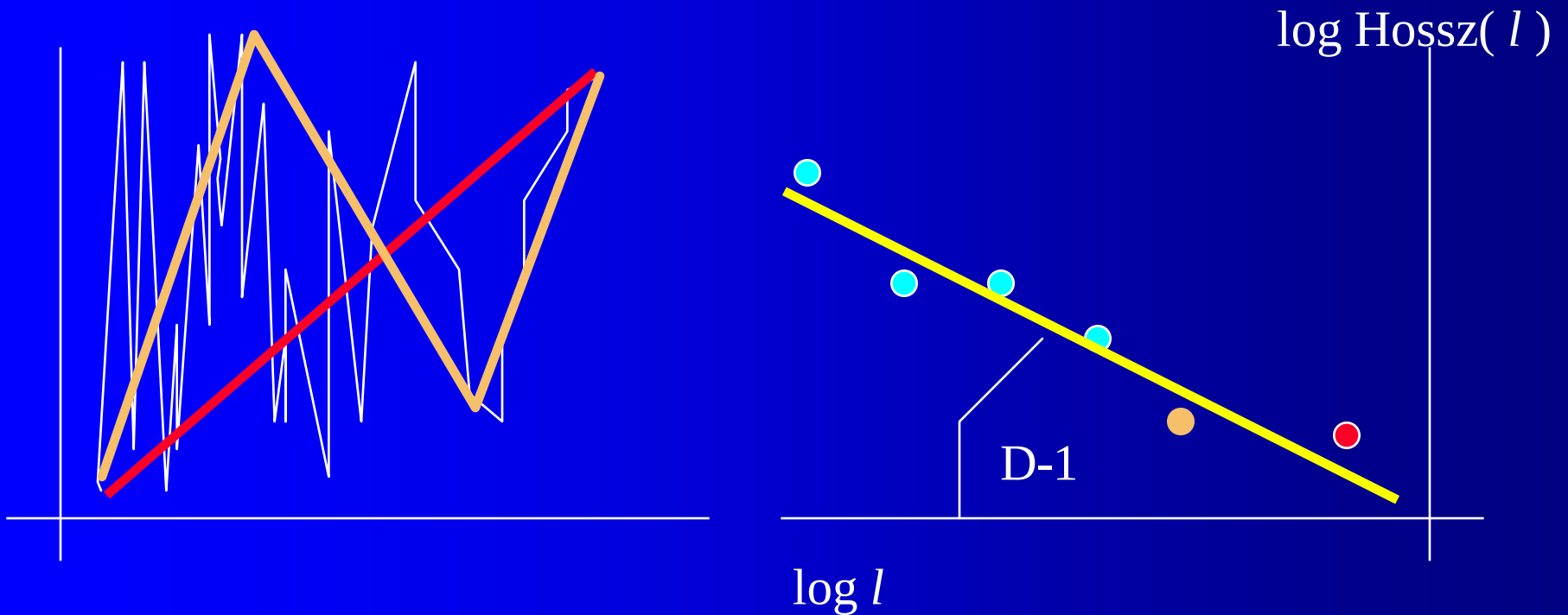
r^m

N^m

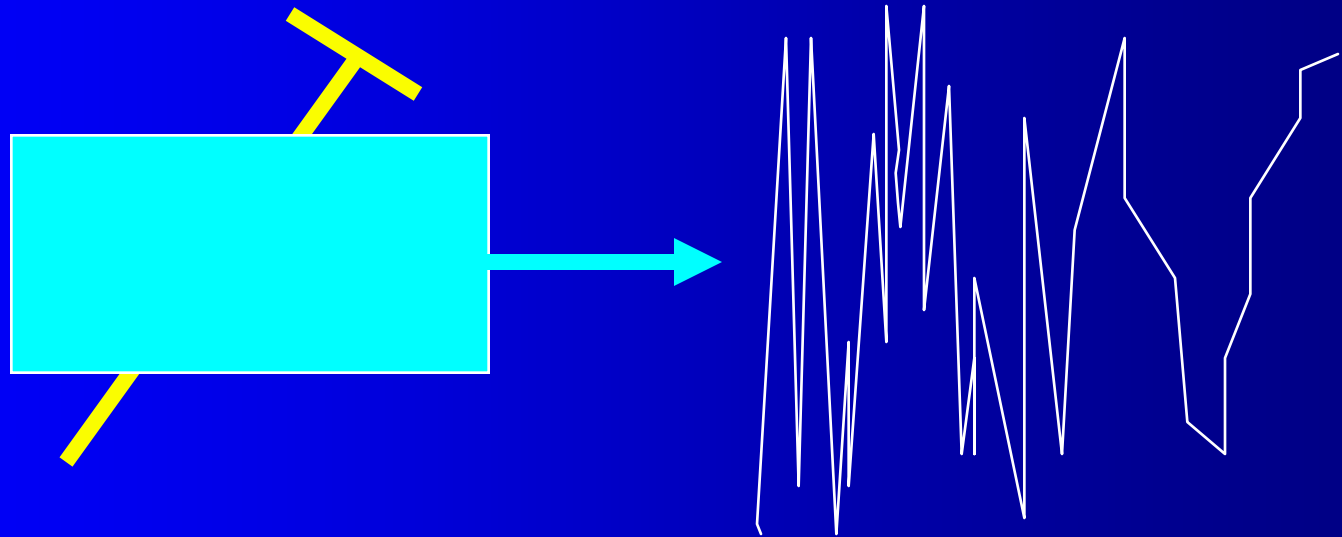
$$\begin{aligned} \text{Hossz}(l) &= l \text{ db} = l N^m = l (1/r^D)^m = \\ &= l (1/r^m)^D = 1/l^{D-1} \end{aligned}$$

$$D = -\log \text{Hossz}(l) / \log l + 1$$

Dimenziómérés = hossz mérés



Fraktálok előállítása



Matematikai gépek:

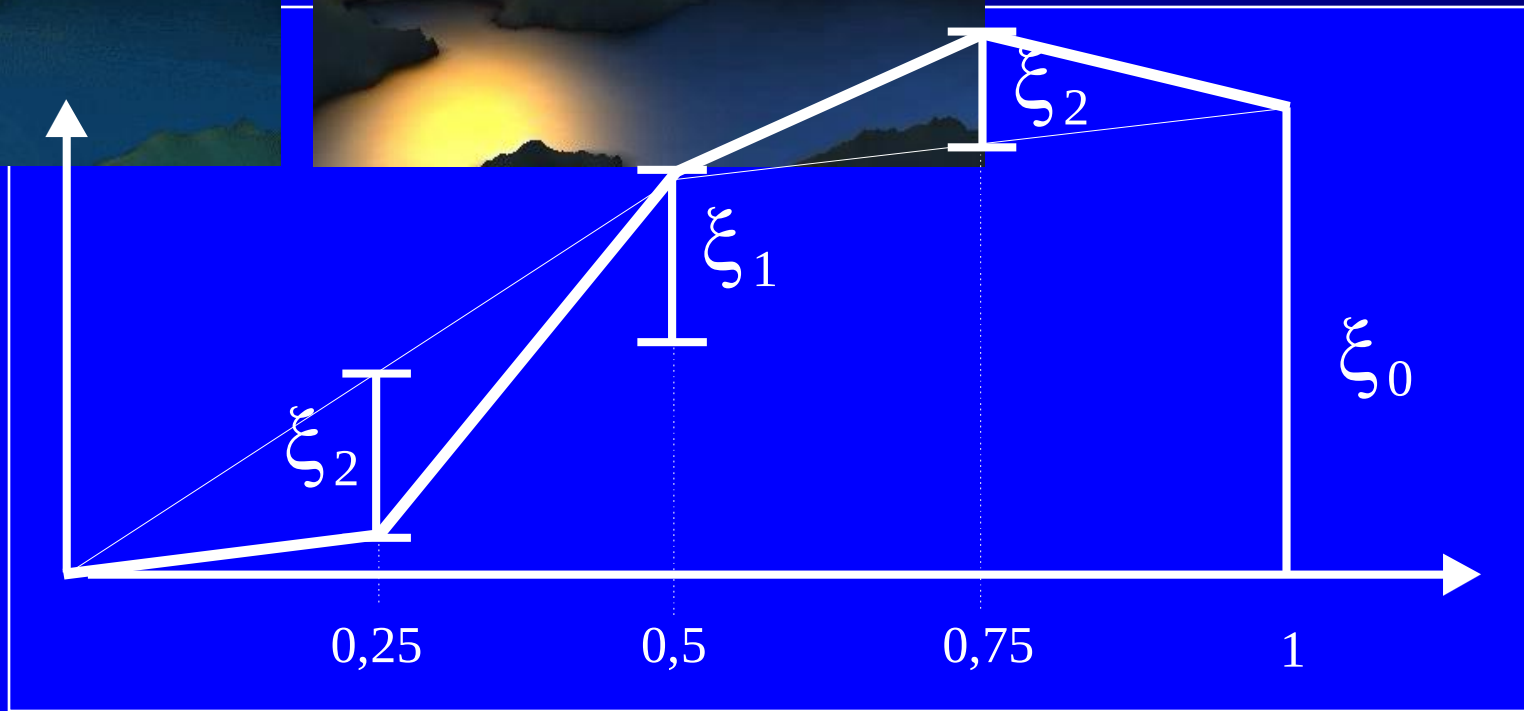
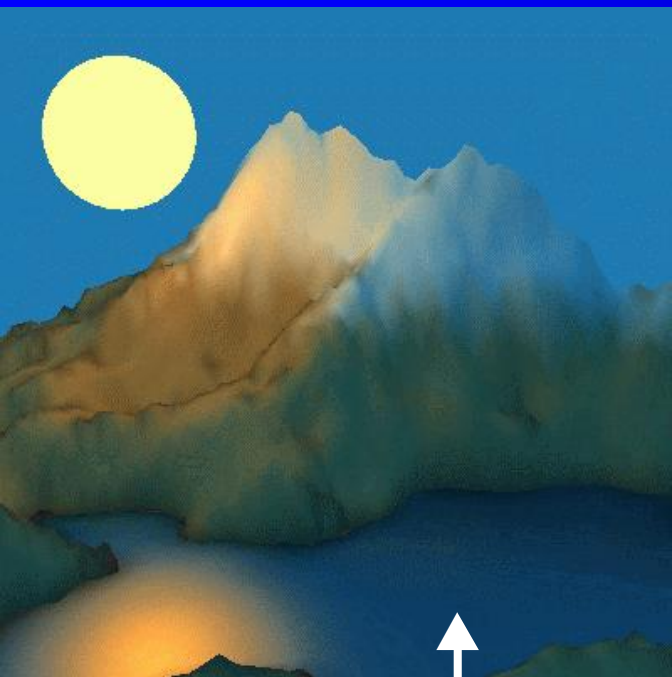
- Brown mozgás

Kaotikus dinamikus rendszerek

Brown-mozgás - Wiener-féle sztochasztikus folyamat

- Sztochasztikus folyamat
- Trajektóriák foly
- Függgetlen növekm
- Növekmények 0- v eloszlás:
 - a független növekm
 - independently

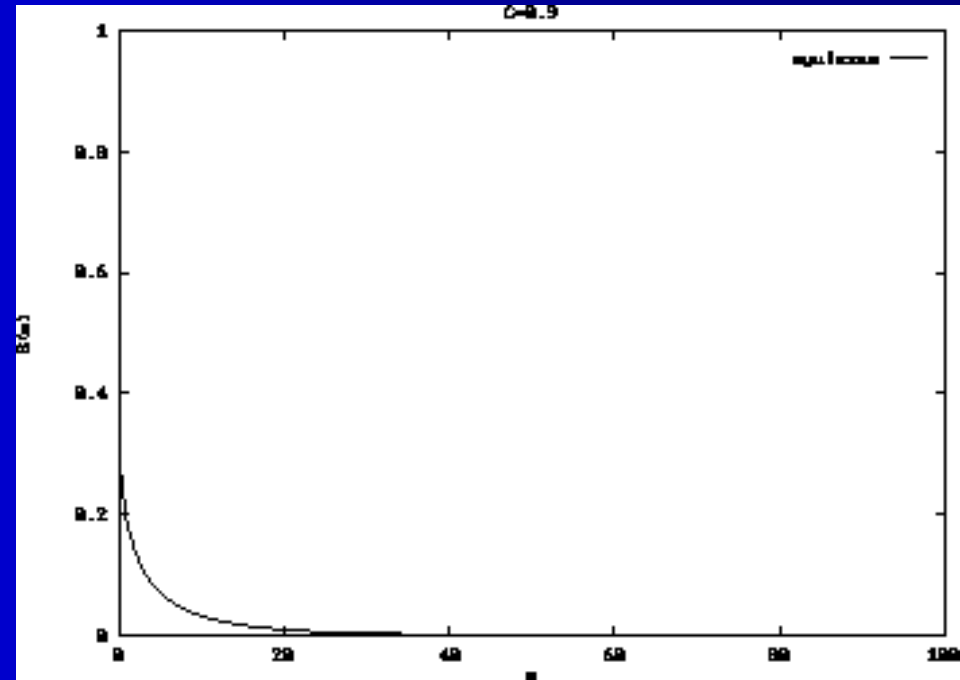
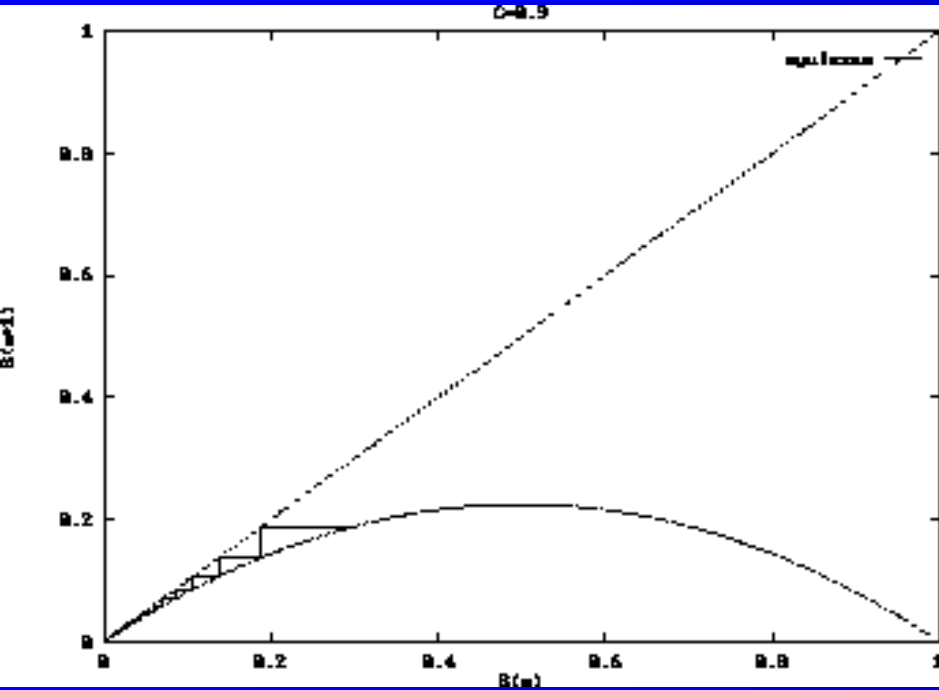
Brown mozgás alkalmazása



Kaotikus dinamikus rendszer:

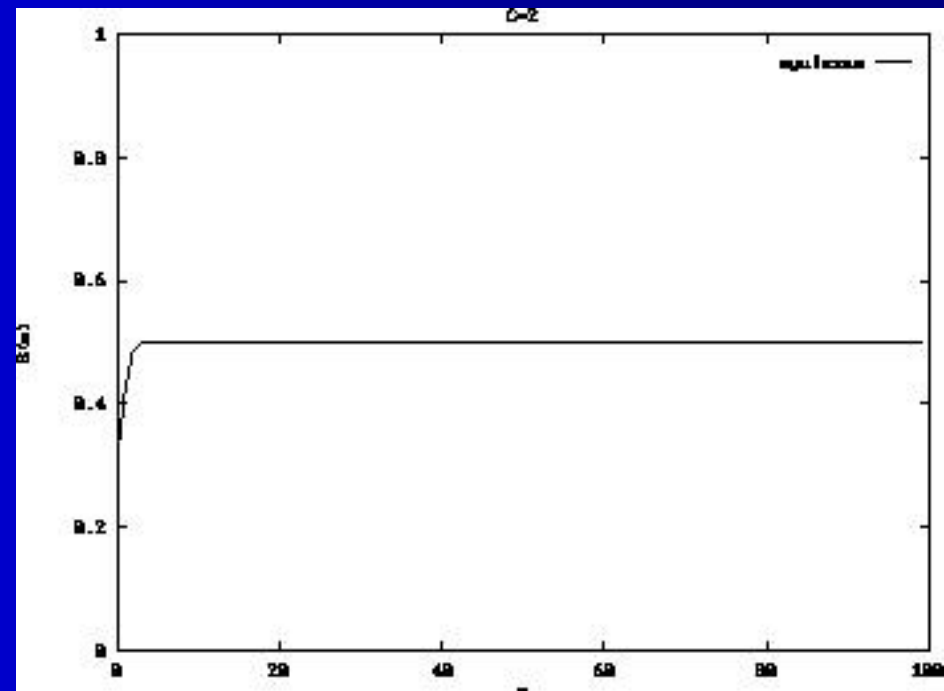
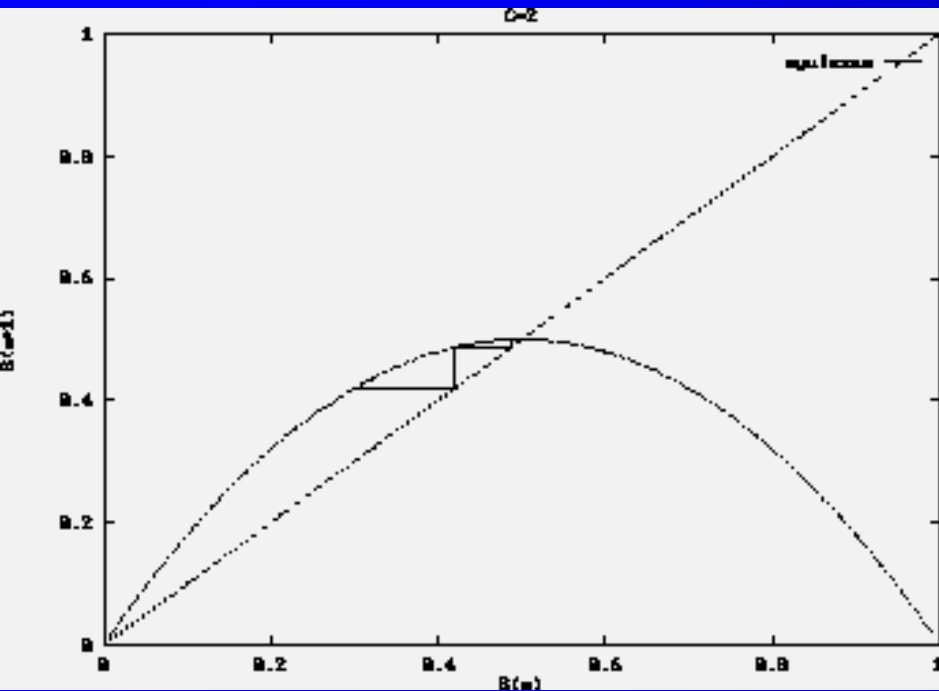
nyulak kis C értékre tényleg

$$S_{n+1} = C S_n (1 - S_n)$$

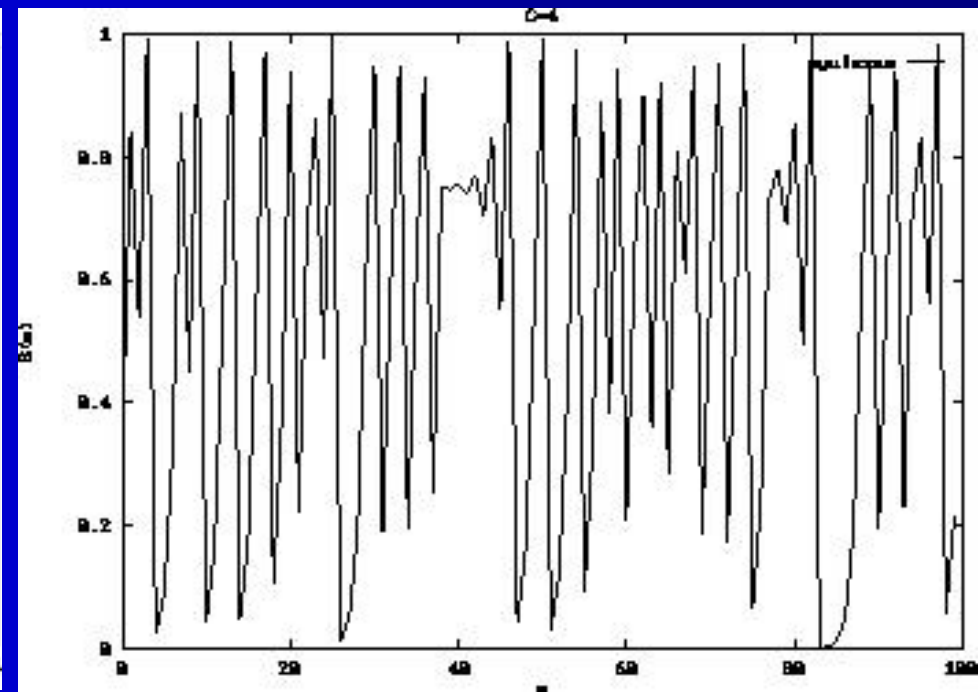
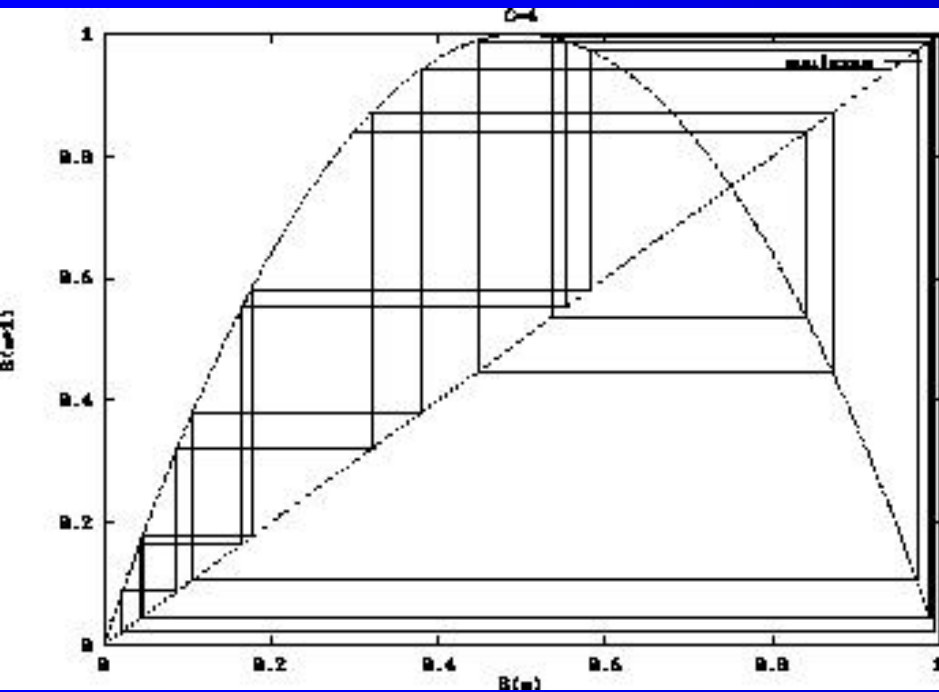


Kaotikus dinamikus rendszer:

nyúlak közepes értékű rögzített



Kaotikus dinamikus rendszer: nyulak nagy C értékre

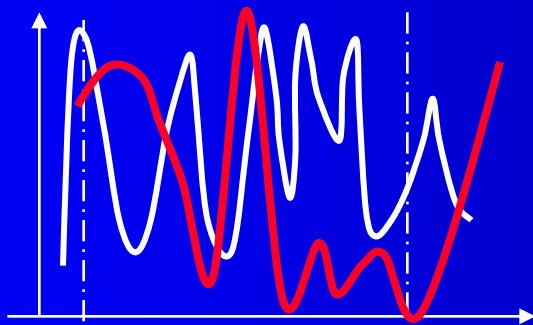


Pseudo véletlen szám generátor

- Iteratív függvény

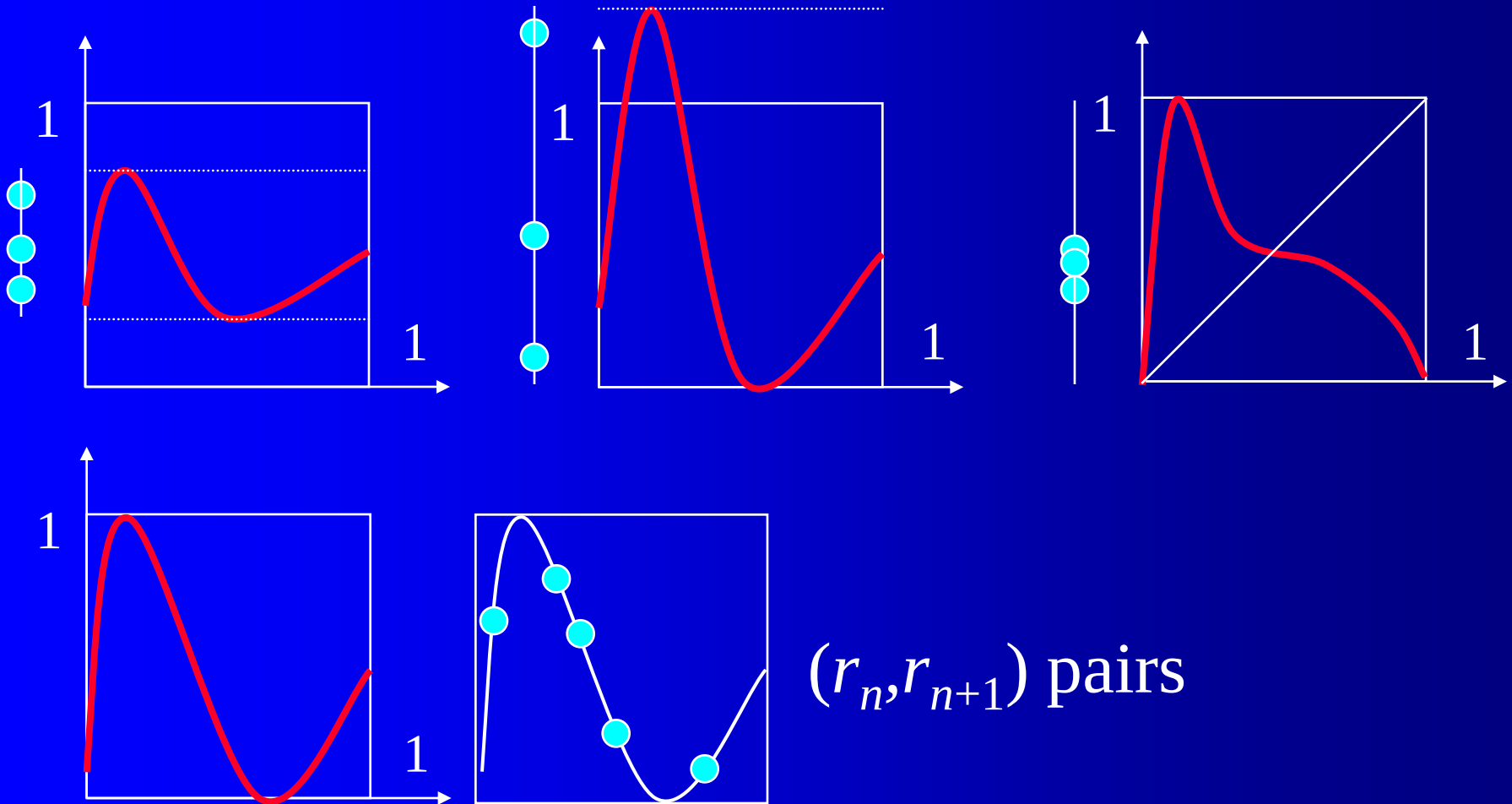
$$r_{n+1} = F(r_n)$$

- véletlenként h



F nagyidei

Rossz: $r_{n+1} = F(r_n)$



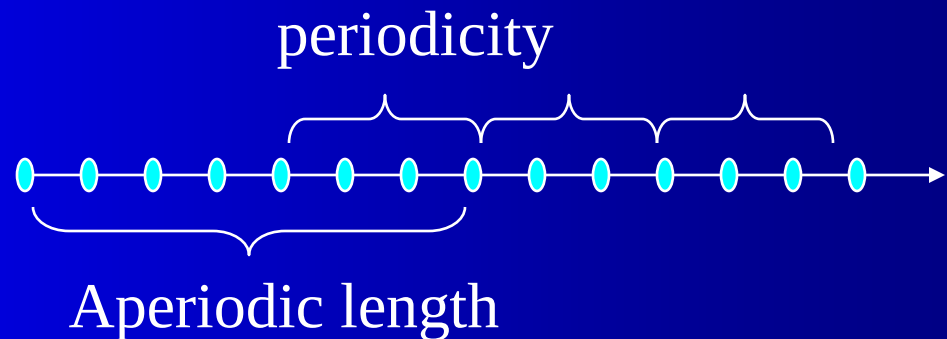
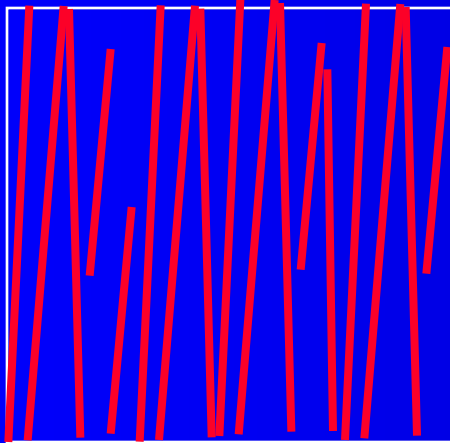
(r_n, r_{n+1}) pairs

JóFó

● S C r C n k i t ö l t i a

● M i n d e n ü t t n a g y d

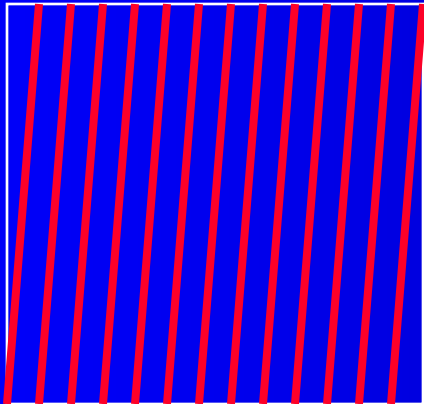
● a $[0, 1]$ -ben van



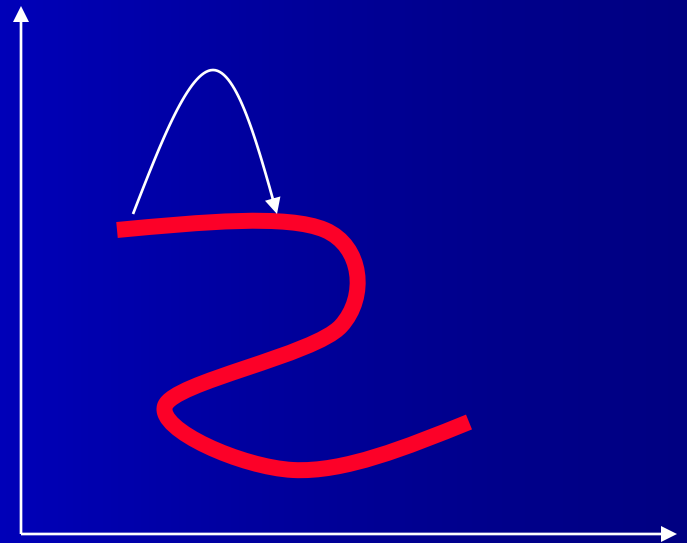
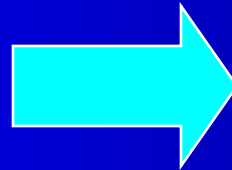
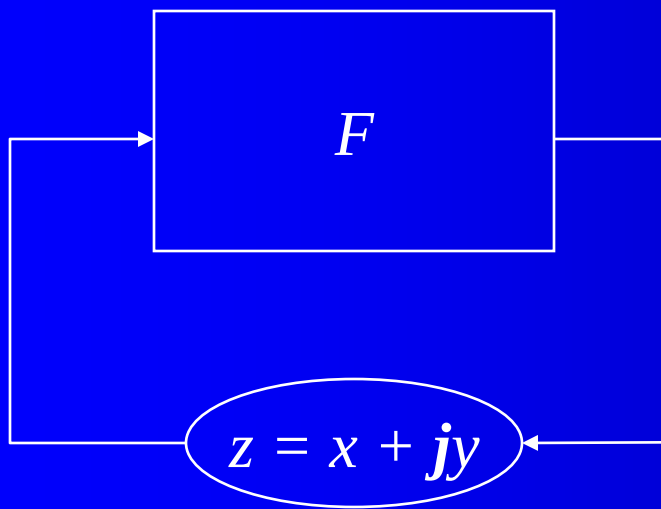
Kongruenzgenerátor

$$F(r) = \{ g \cdot r + c \}$$

$g \cdot r + c$ tőrt
 g nagy



Kaotikus rendszerek síkrajza



$$z \mapsto z^2$$



$$z = r e^{j\phi}$$

$$r \mapsto r^2$$

$$\phi \mapsto 2\phi$$

Attraktor előállítása

- A attraktor a alábbiak közül ha $t \rightarrow \infty$ z_t a -ra konvergál, akkor a attraktor, ha $t \rightarrow \infty$ z_t nem konvergál, akkor a nem attraktor.

$$z_{n+1} = z_n^2 : \text{ha } z_0 < 1 \text{ akkor fekete}$$

- A attraktor a alábbiak közül ha $t \rightarrow \infty$ z_t a -ra konvergál, akkor a attraktor, ha $t \rightarrow \infty$ z_t nem konvergál, akkor a nem attraktor.

$$z_{n+1} = z_n^2 \text{ attraktora labilis}$$

Inverz iterációs módszer

$$H = F(H)$$

$\rightarrow$

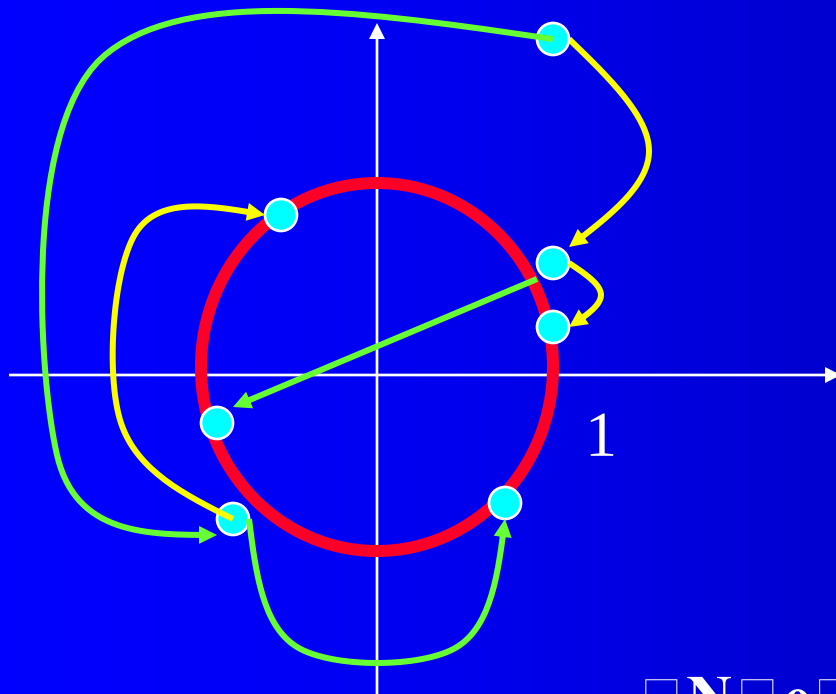
$$H = F^{-1}(H)$$

$$z_{n+1} = z_n^2$$

$$z_{n+1} = \sqrt[n]{z_n}$$

$$r_{n+1} = \sqrt[n]{r_n}$$

$$\phi_{n+1} = \phi_n/2 + \begin{cases} 0 & \text{if } \phi_n \in [0, \pi) \\ \pi & \text{if } \phi_n \in [\pi, 2\pi) \end{cases}$$



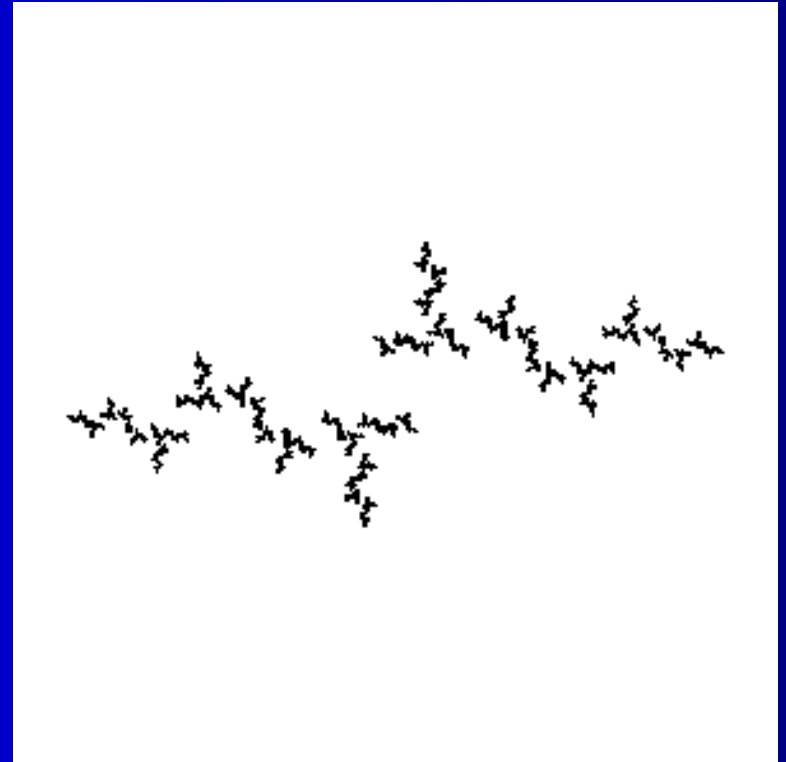
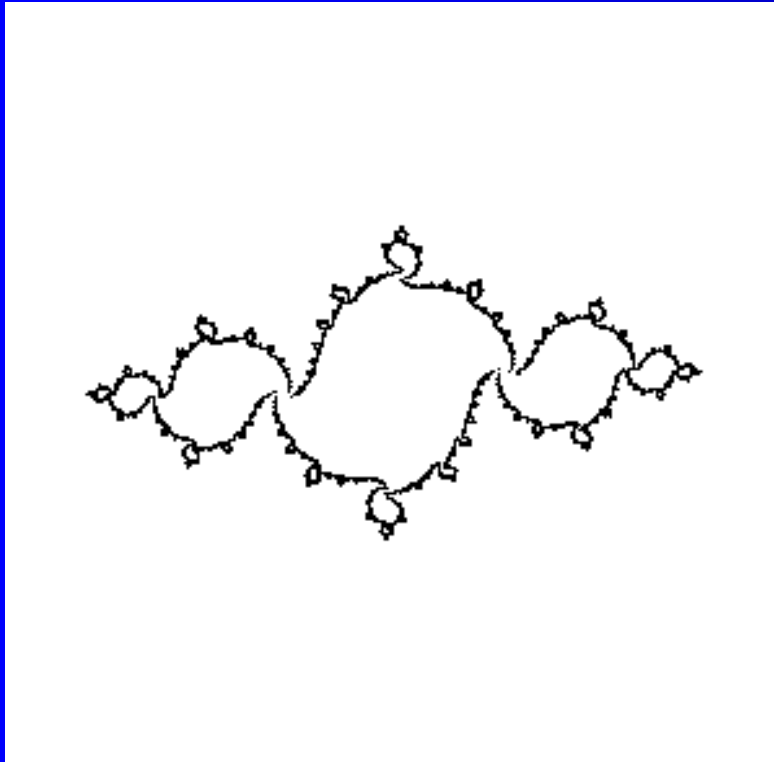
$$r_n \approx 1$$

$$\phi_n \approx \begin{cases} 0 & \text{if } \phi_{n-1} \in [0, \pi) \\ \pi & \text{if } \phi_{n-1} \in [\pi, 2\pi) \end{cases}$$

$$n \quad n-1 \quad n-2$$

Neem lehet csak

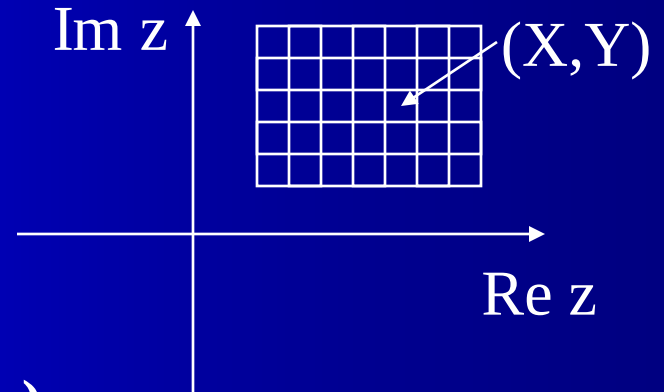
Julia halmaz: $z \mapsto z^2 + c$



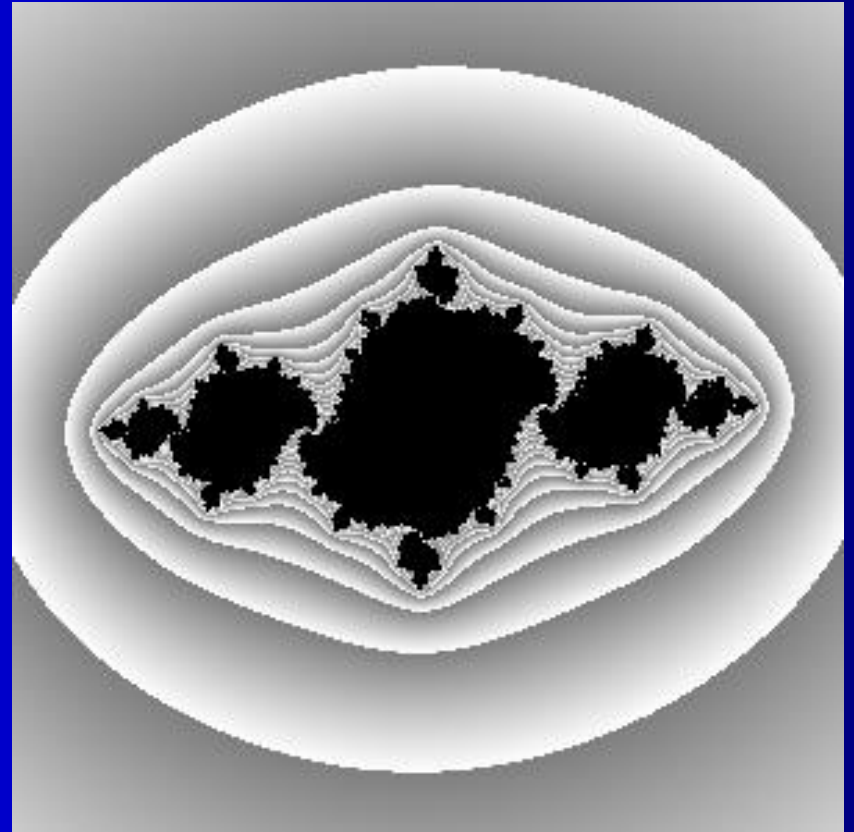
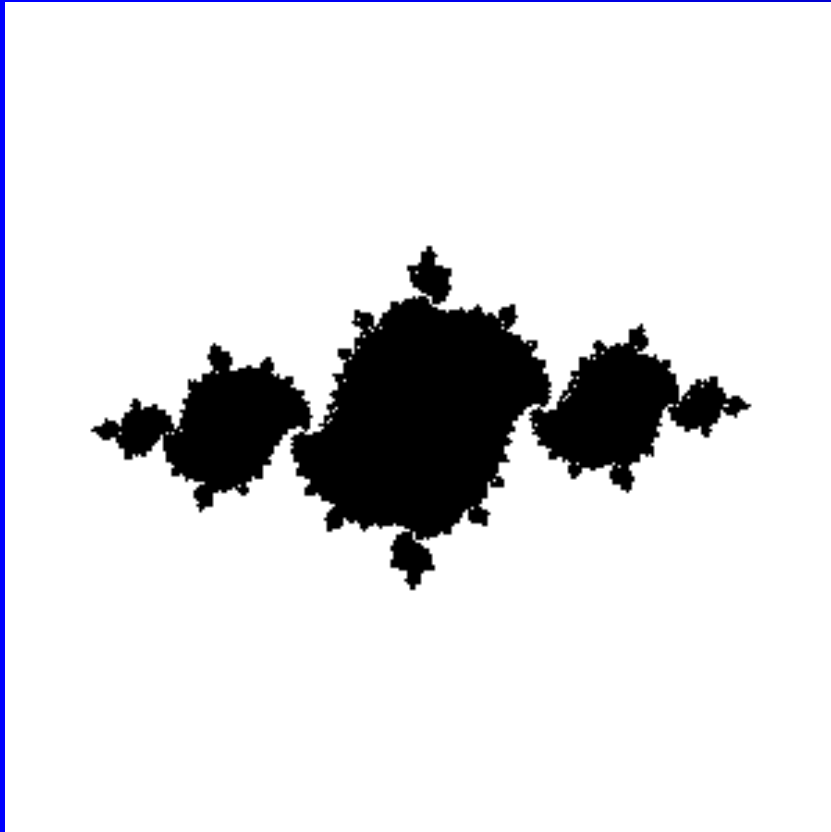
Költött Julia halmaz: algoritmus

FilledJuliaDraw ()

```
FOR Y = 0 TO Ymax DO
  FOR X = 0 TO Xmax DO
    ViewportWindow(X,Y → x, y)
    z = x + j y
    FOR i = 0 TO n DO z = z2 + c
      IF |z| > 2 THEN "infinit y"
    ELSE WRITE(X,Y, black)
    ENDFOR
  ENDFOR
ENDFOR
END
```



Költő Julia halmaz: kép



Julia halmaz inverz iterációval

JuliaDrawInverseIterate ()

Kézdeti z értéket választás

FOR $i = 0$ TO n DO

$x = \text{Re } z, \quad y = \text{Im } z$

IF ClipWindow(x, y)

WindowViewport($x, y \Rightarrow X, Y$)

Pixel(X, Y) = fekete

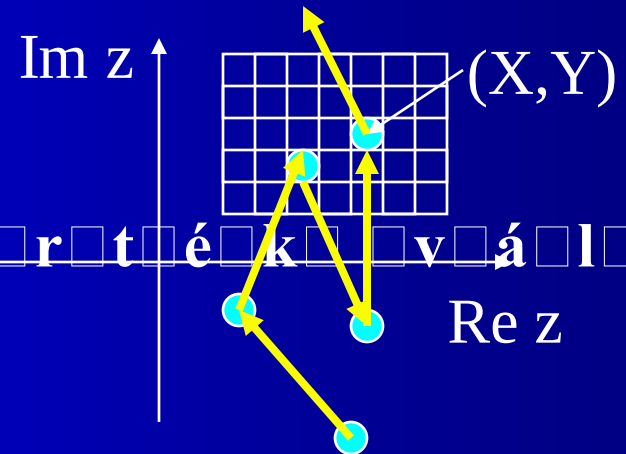
ENDIF

$z = \sqrt[n]{z} - c$

if (rand() > 0.5) $z = -z$

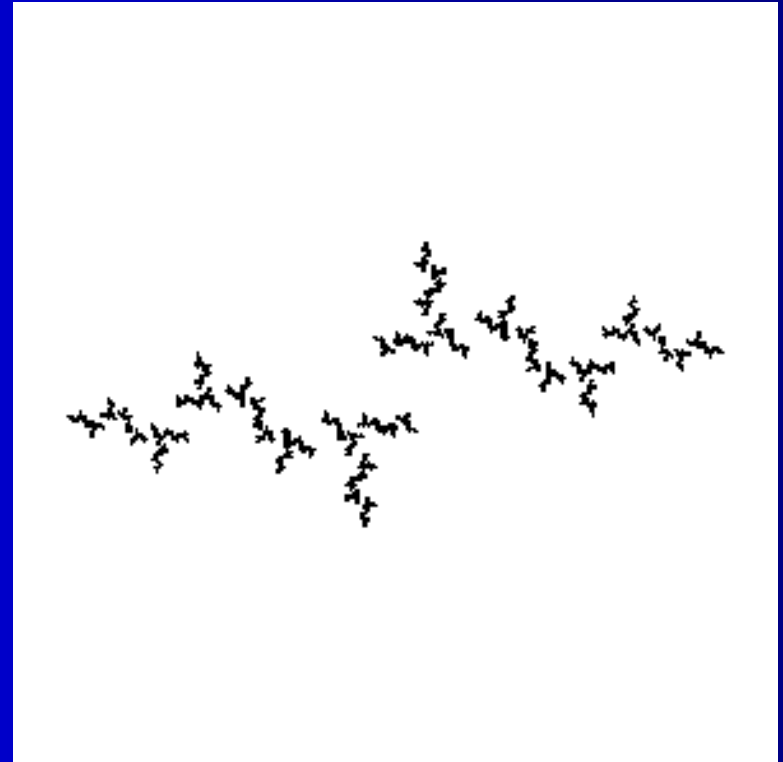
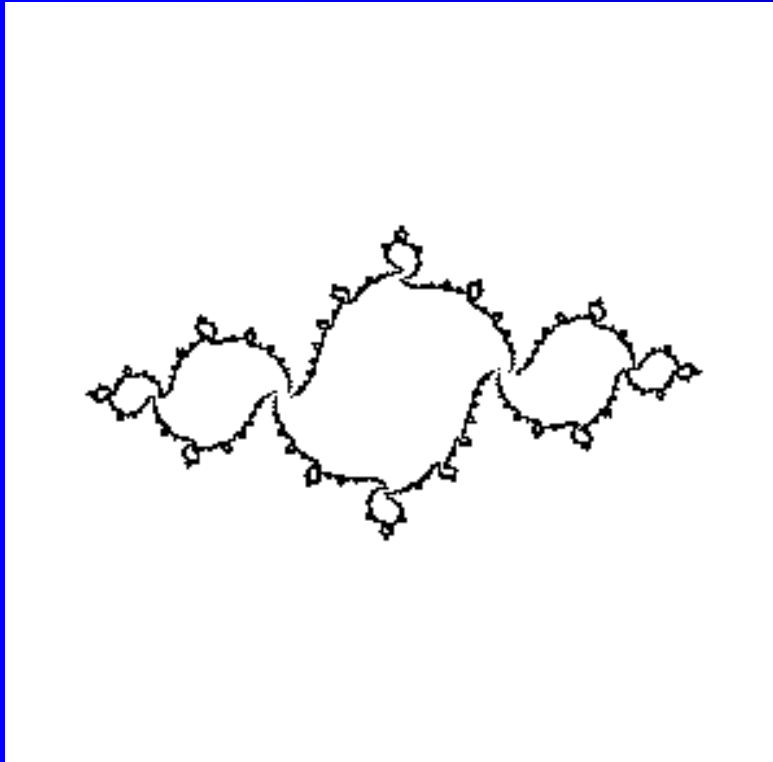
ENDFOR

END



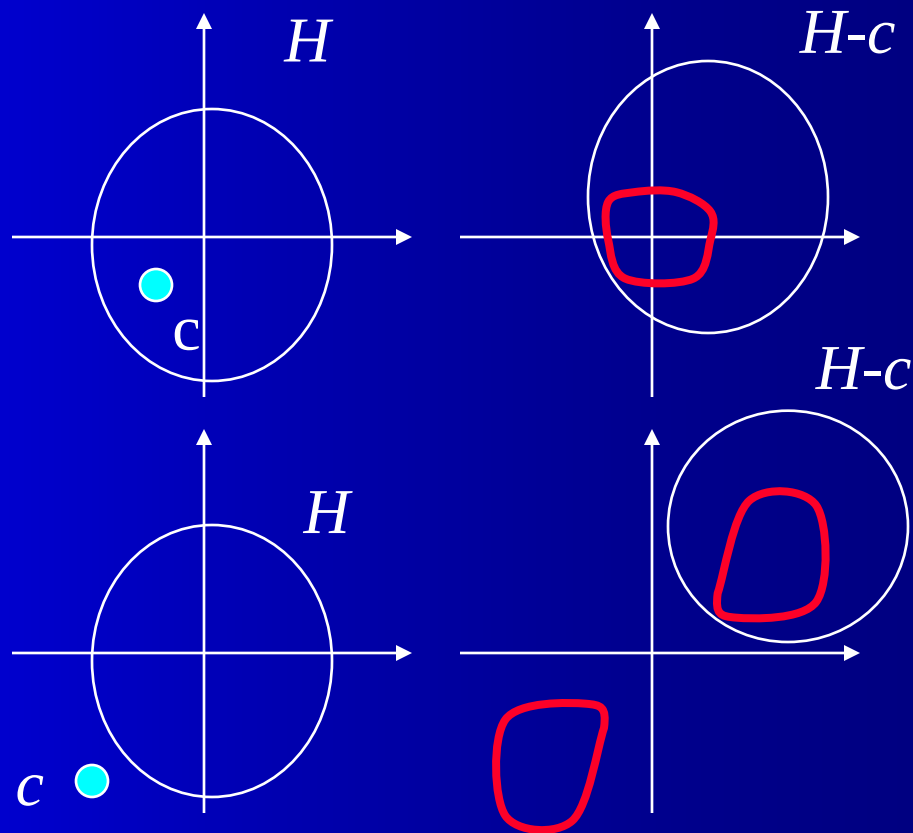
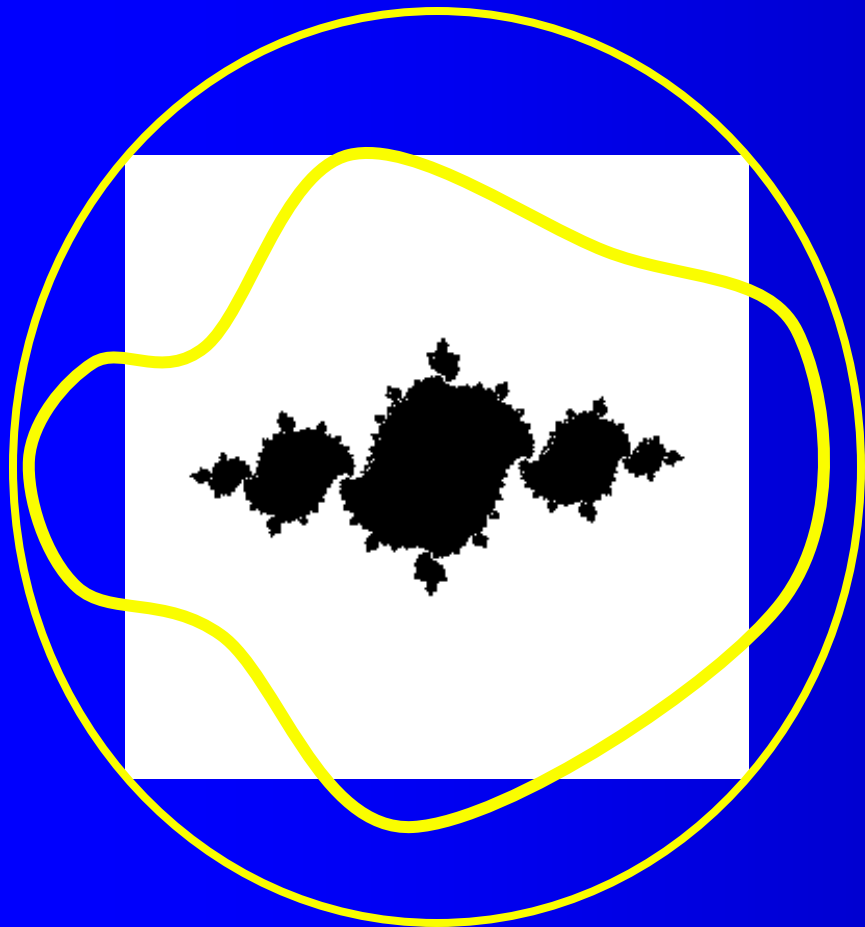
Kezdeti z értéket
 $z^2 = z - c$ egyenlet

Julia halmaz



□ö□s□s□z□e□f□ü□g□g □n□e□m□ □ö□s□s□z□e□
5
□C□a□n□t□o□r□ □f□é□l□

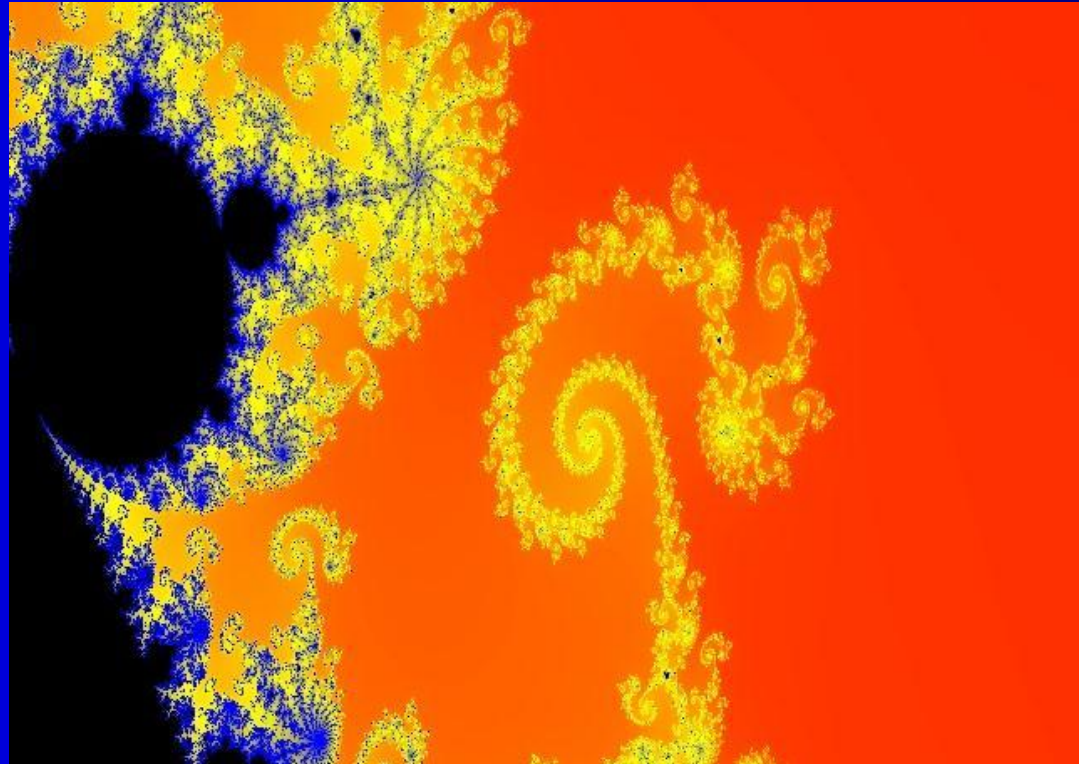
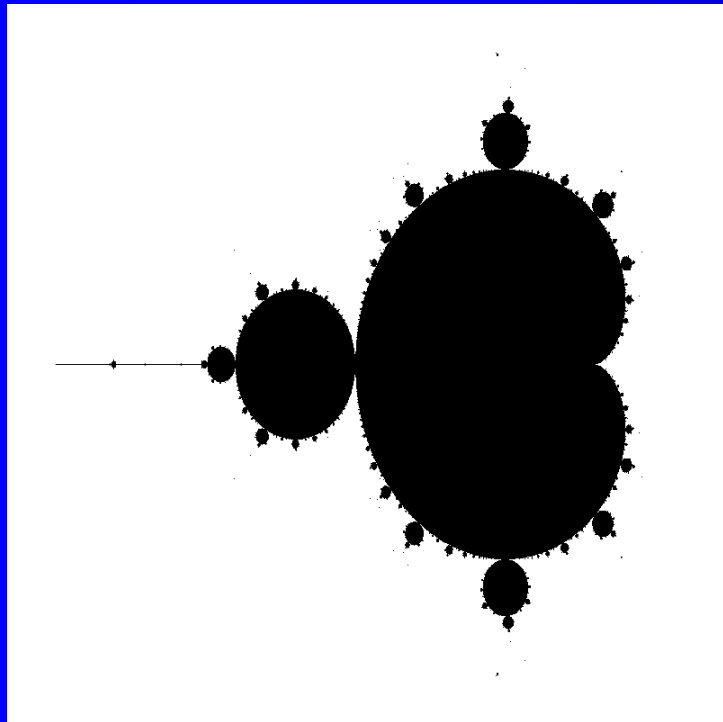
Julia-halmazok összefüggése



$$z_{n+1} = z_n^2 - c$$

Mandelbrot halmaz

Azon c komplex szám, amelyre a
 $z \rightarrow z^2 + c$ Jénuilíal halmaz:

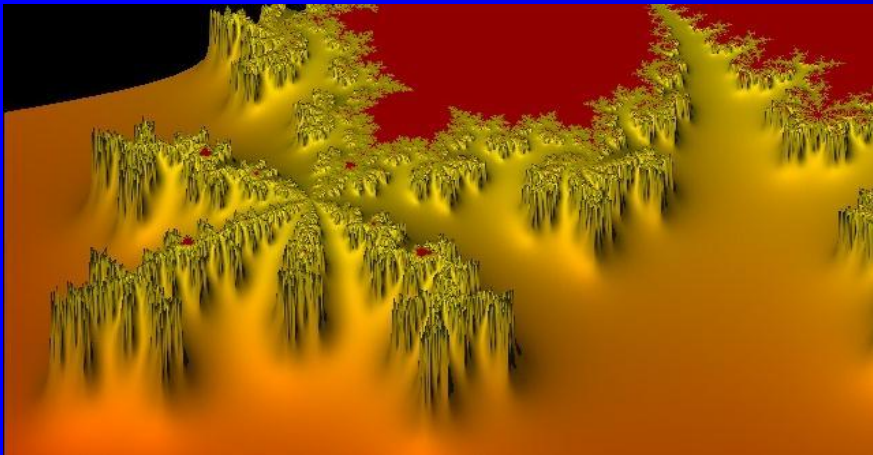
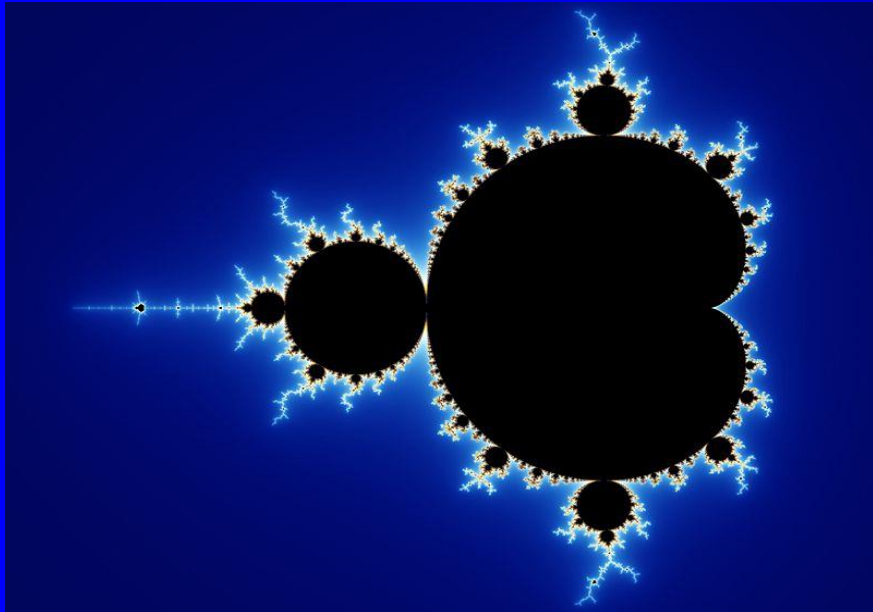


Mandelbrot halmaz, algoritmus

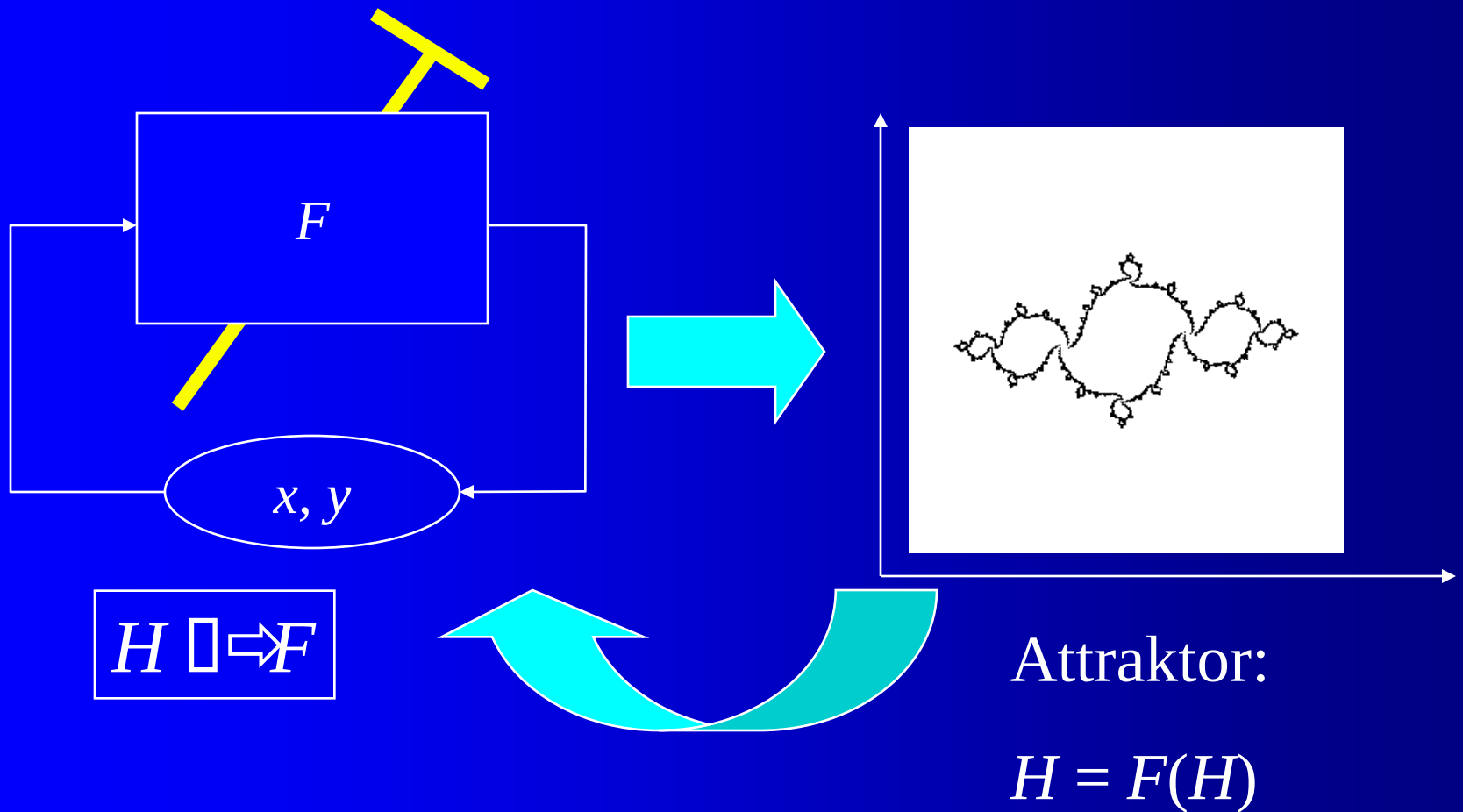
MandelbrotDraw ()

```
FOR Y = 0 TO Ymax DO
  FOR X = 0 TO Xmax DO
    ViewportWindow(X,Y → x, y)
    c = x + j y
    z = 0
    FOR i = 0 TO n DO z = z2 + c
      IF |z| > 2 THEN "i n f i n i t y"
    ELSE
      WRITE(X,Y, black)
    ENDFOR
  ENDFOR
ENDFOR
END
```

„Matematikájszótár” Mandelbrot halmazok



Inverz feladat: IFS modellezés



F : szabadozon vezérelhet 5,

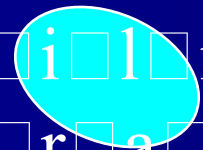
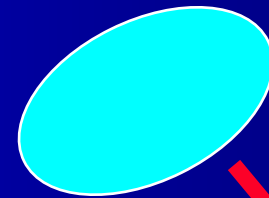
Ftöbbértékű leképezések

$$F = W_1 \cup W_2 \cup \dots \cup W_n$$

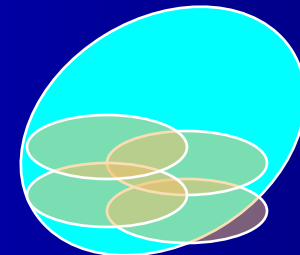
$$W(x,y) = [ax + by + c, dx + ey + f]$$

$$H = W_1(H) \cup W_2(H) \cup \dots \cup W_n(H)$$

$$H = F(H)$$



Szabványos
kontra



IFSDrajzolás iterációs algoritmus

IFSDraw ()

Legyen $[x,y] = [x,y] A_1 + q_1$ mmegkezdés

FOR i = 0 TO n DO

IF ClipWindow(x, y)

WindowViewport(x, y $\Rightarrow$ X, Y)

Write(X, Y, color);

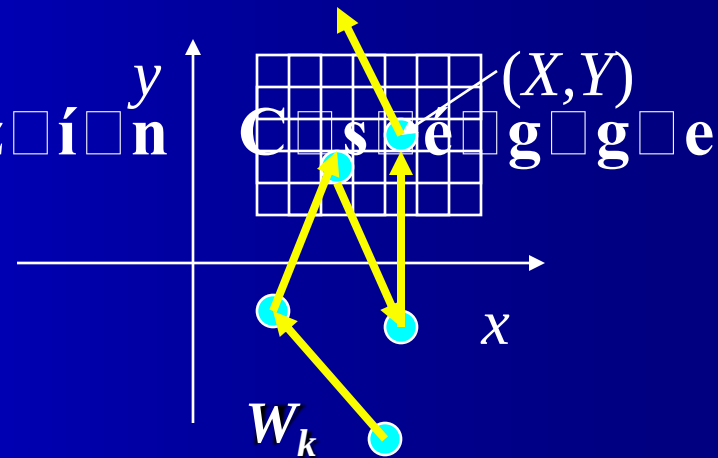
ENDIF

Máskor $[x,y] = [x,y] A_k + q_k$ színe

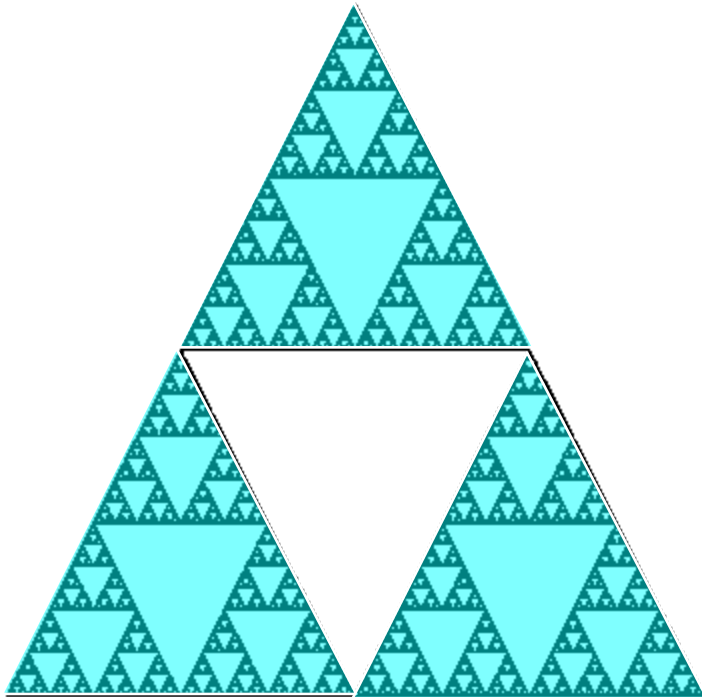
$[x,y] = [x,y] A_k + q_k$

ENDFOR

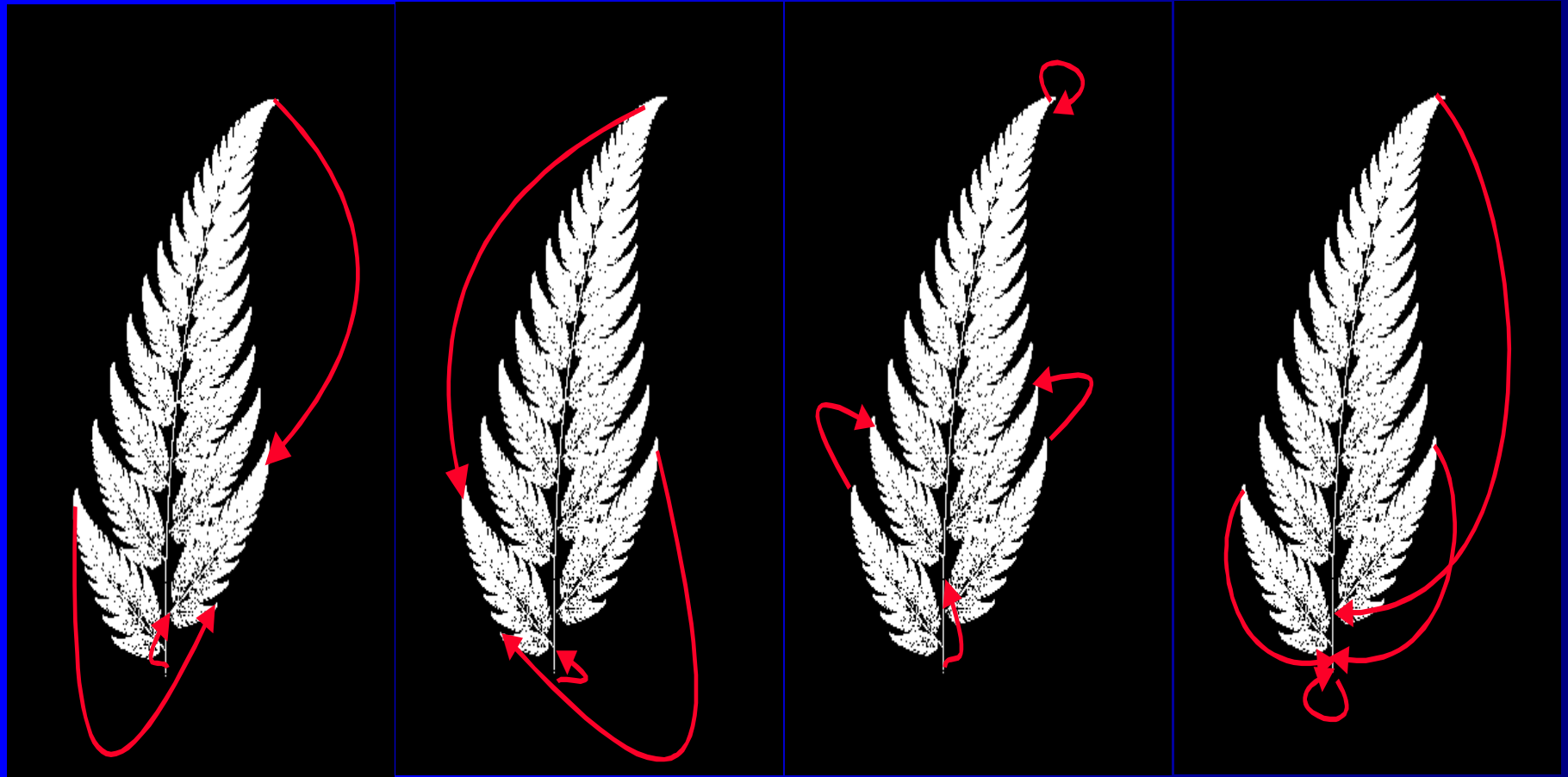
END



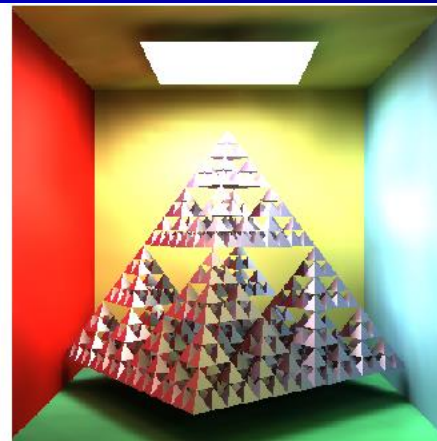
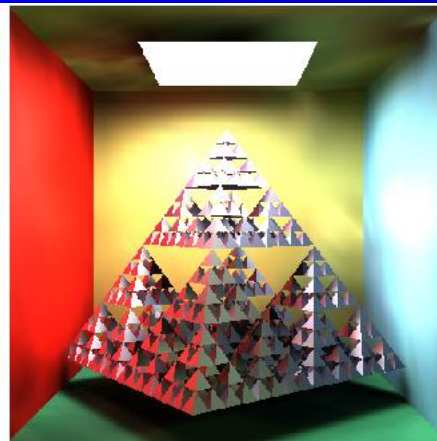
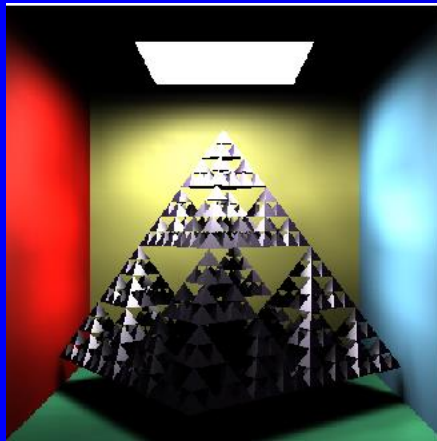
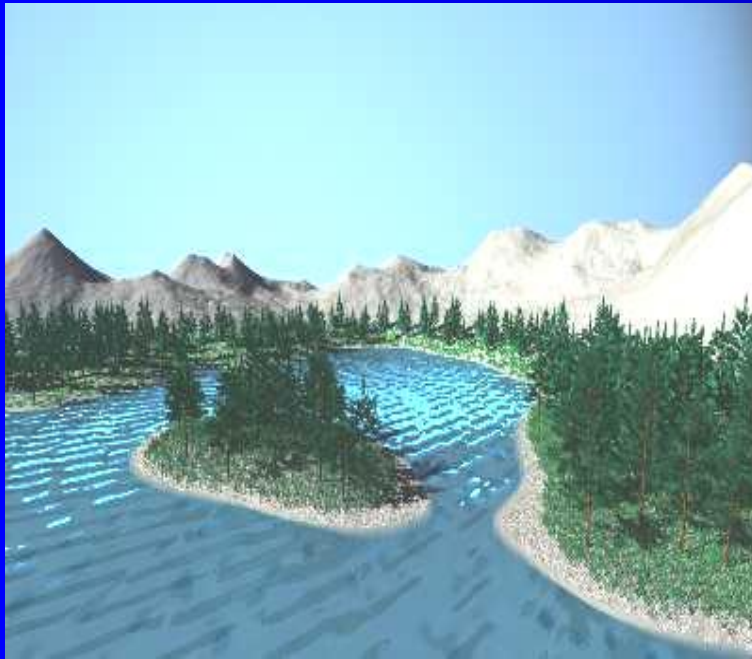
Egyszerűsze



IFS model



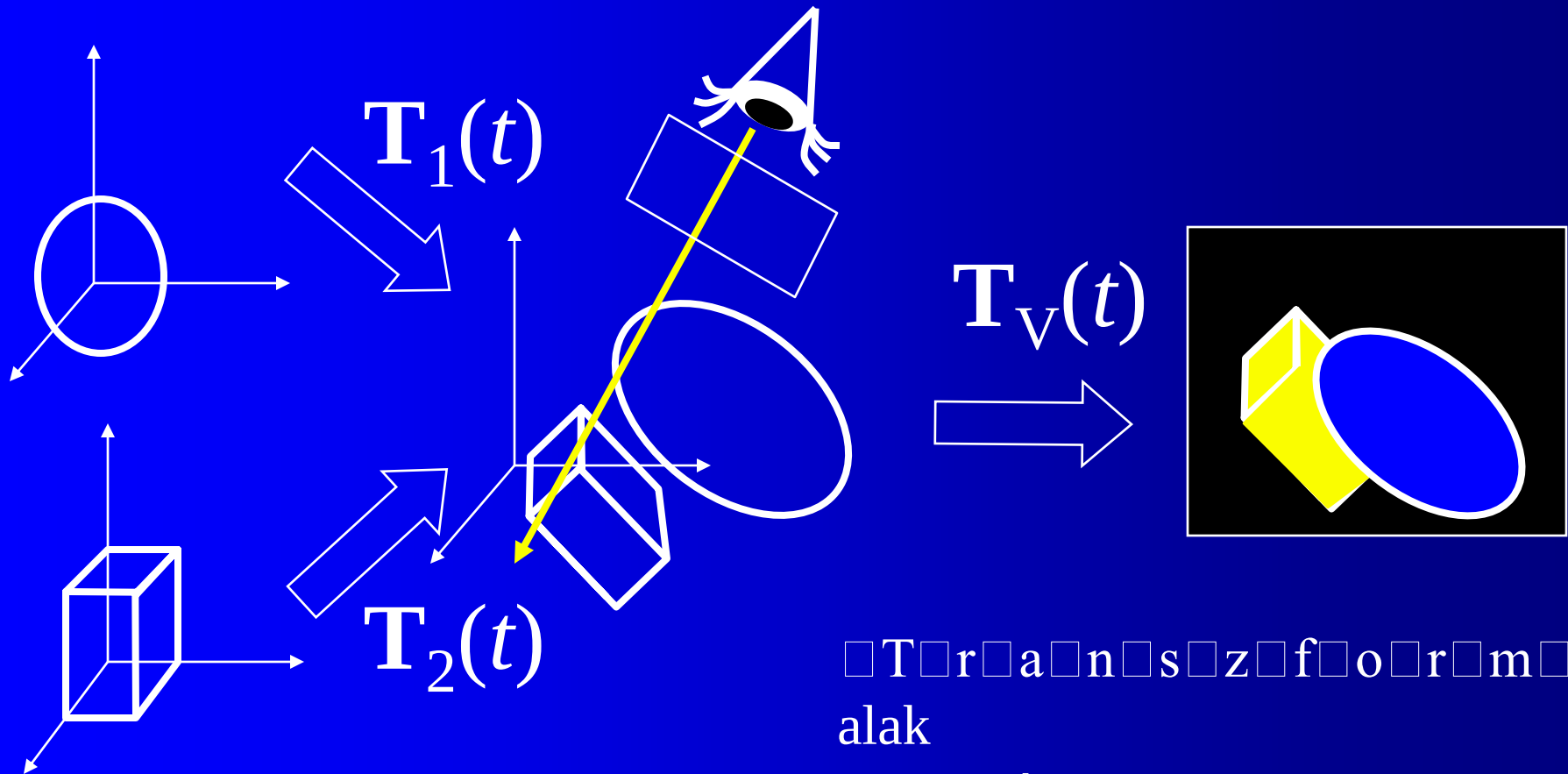
FS Képek



Animáció

Szirmay-Kalos László

Animáció márció =



Traanszformációk
alak

szín

megjelölés

Validajóasimáid e j

```
Ór a in i) c i a l i z á l á s ( t
t_prev = t_start;
do
    t = Ór a l e o l v a s á s
    Szimul á c i o ( t
    for e a c h o b j e c t T(t):
        N é z e t i t v = a v ( t ); s z f o r m á
    K é p s ; z i n t é z i s
    t_prev = t;
while (t < t_end);
```

Valószínűségi C

● Fizikai törvények

– Newton törvény

– ütközés detektálás

● Fiziológiai törvények

– csontváz nem száka

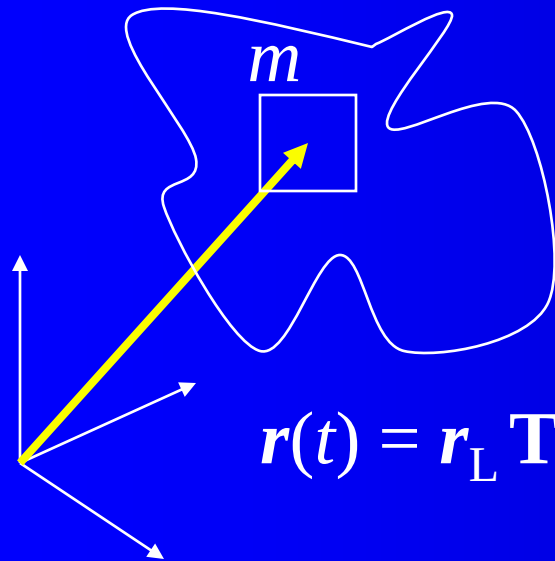
– meghatározottt szába

– b 5r rugalmasan köv

● Energiafelháson á



Newton's equations



$$\mathbf{r}(t) = \mathbf{r}_L \mathbf{T}_M(t)$$

$$\mathbf{F}/m = \frac{d^2}{dt^2} \mathbf{r}$$

$$= \mathbf{r}_L \frac{d^2}{dt^2} \mathbf{T}_M(t)$$

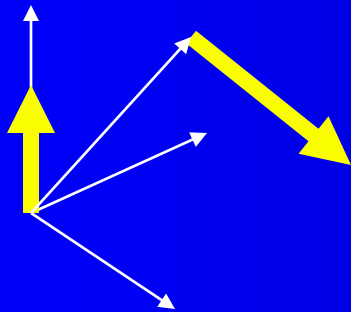
Az érő rugalmas $\mathbf{T}_M(t) \in \mathbb{C}^2$ folytonos
 keresztülhát, azaz az erő
 vektorozik

$T_M(t)$: Mozgástervezés

● Kővetelmények: a fájl

● Mátrixelemek nem

– Tervezési függvény



pozíció, p_x, p_y, p_z

orientáció, α, β, γ $\mathbf{p}(t)$

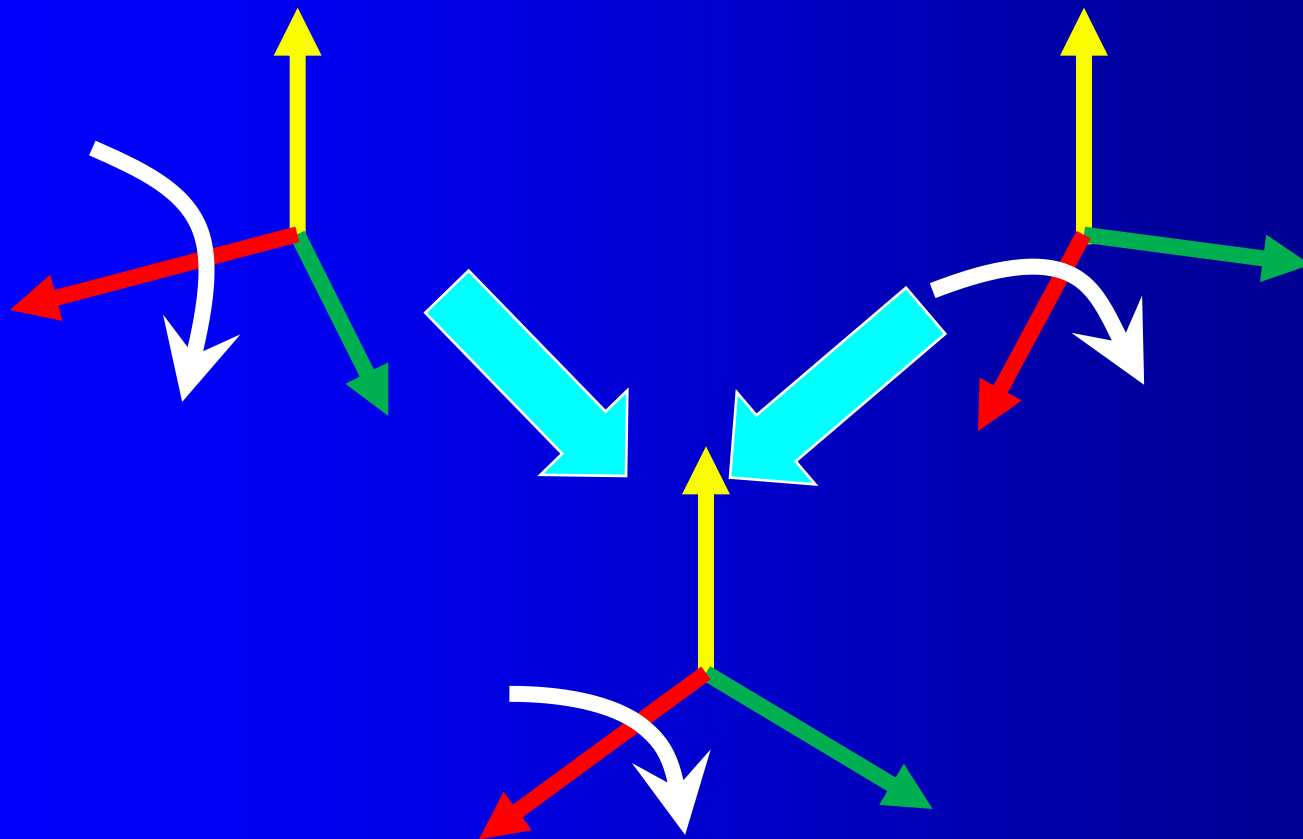
skalársz, s_x, s_y, s_z

Követelmény

$T_M(t) =$

$$\begin{bmatrix} s_x \\ s_y \\ s_z \\ 1 \end{bmatrix} \begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \\ & & 1 \\ 1 & & & \end{bmatrix} \begin{bmatrix} \cos \beta & -\sin \beta \\ & 1 \\ \sin \beta & \cos \beta \\ 1 & & & \end{bmatrix} \begin{bmatrix} 1 \\ \cos \gamma & \sin \gamma \\ -\sin \gamma & \cos \gamma \\ 1 & & & \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ p_x, p_y, p_z, 1 \end{bmatrix}$$

Kvantitatív mérési módok

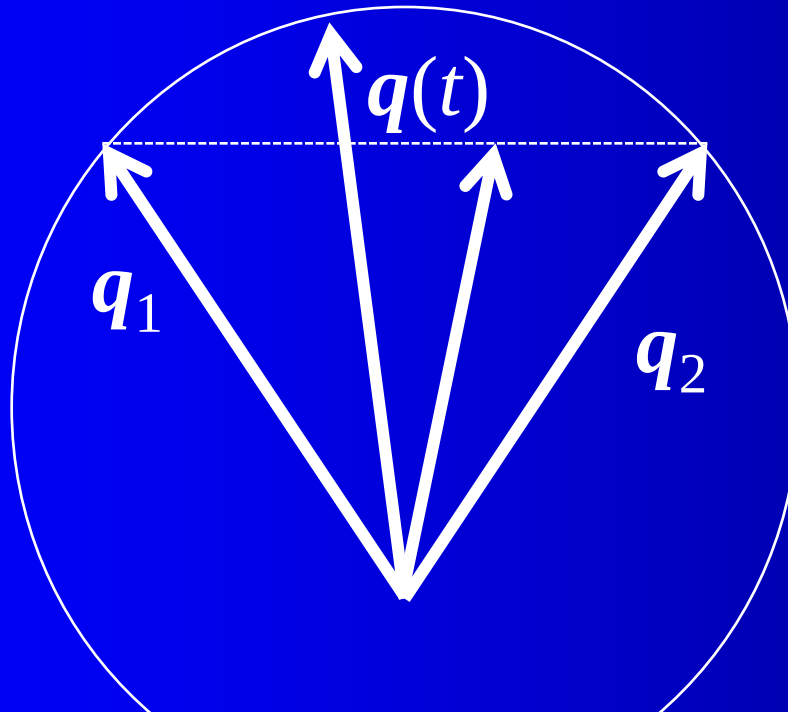


Kiváratás a Katerniók térbeli interpretációjára

$$q[0,u]q^{-1}=[0,v]$$

$$\bullet q_1 = [\cos(\alpha/2), w_1 \sin(\alpha/2)], \quad |w_1| = 1$$

$$\bullet q_2 = [\cos(\alpha/2), w_2 \sin(\alpha/2)], \quad |w_2| = 1$$

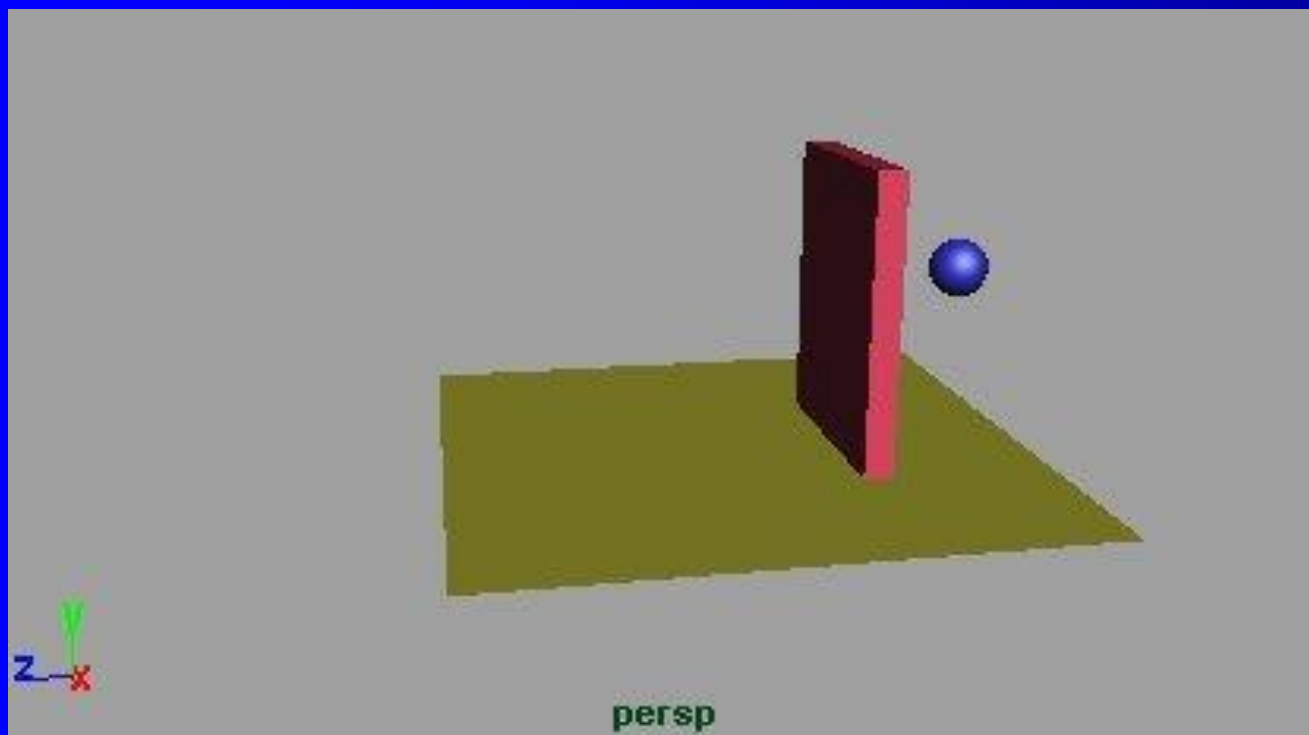
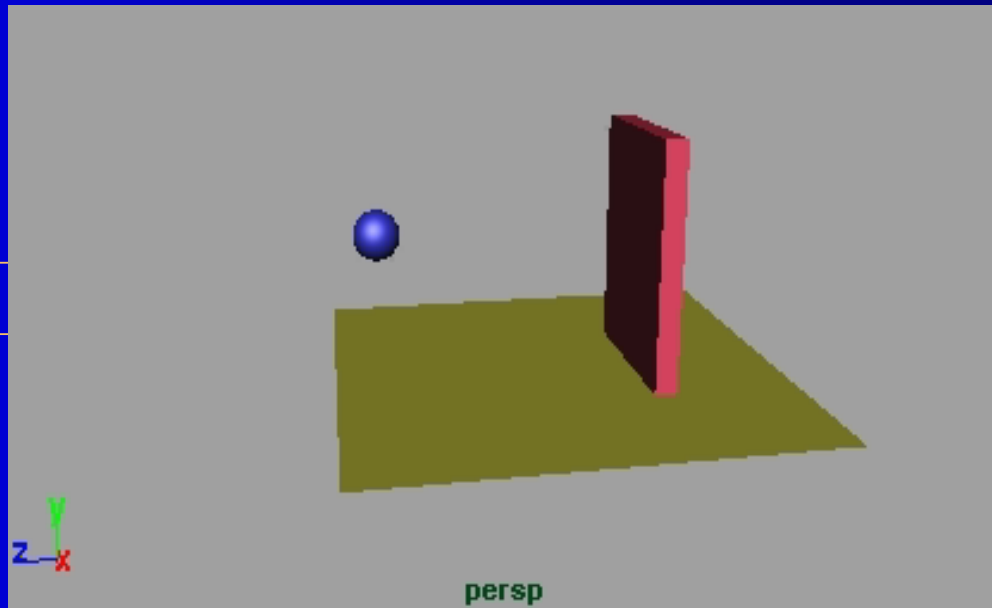
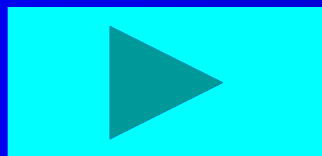


Mozgásvezetés paraméterei

- $p(t)$ a f e m n e f o y t o n s a t C
- $p(t)$ a e l e m e i n e k a $$
 - görbe v e $spline$ i r e k t m $ó$
 - képscript a n i m a t i o n $:$
 - kulcsok b $ó$ k e y f r a m e n a n i m a t i o n
 - görbe v e l p a t h a n i m a t i o n e k t $$
 - mechanikai m o d e l p h y s i c a l a n i m a t i o n
 - mérési m o d e l m o t i o n k a p t u r e a n i m a t i o n

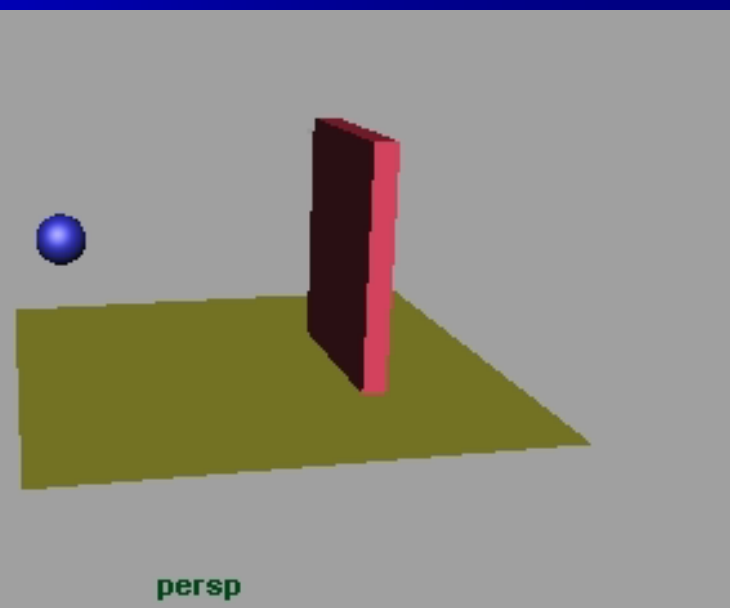
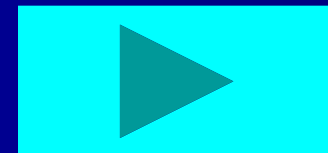
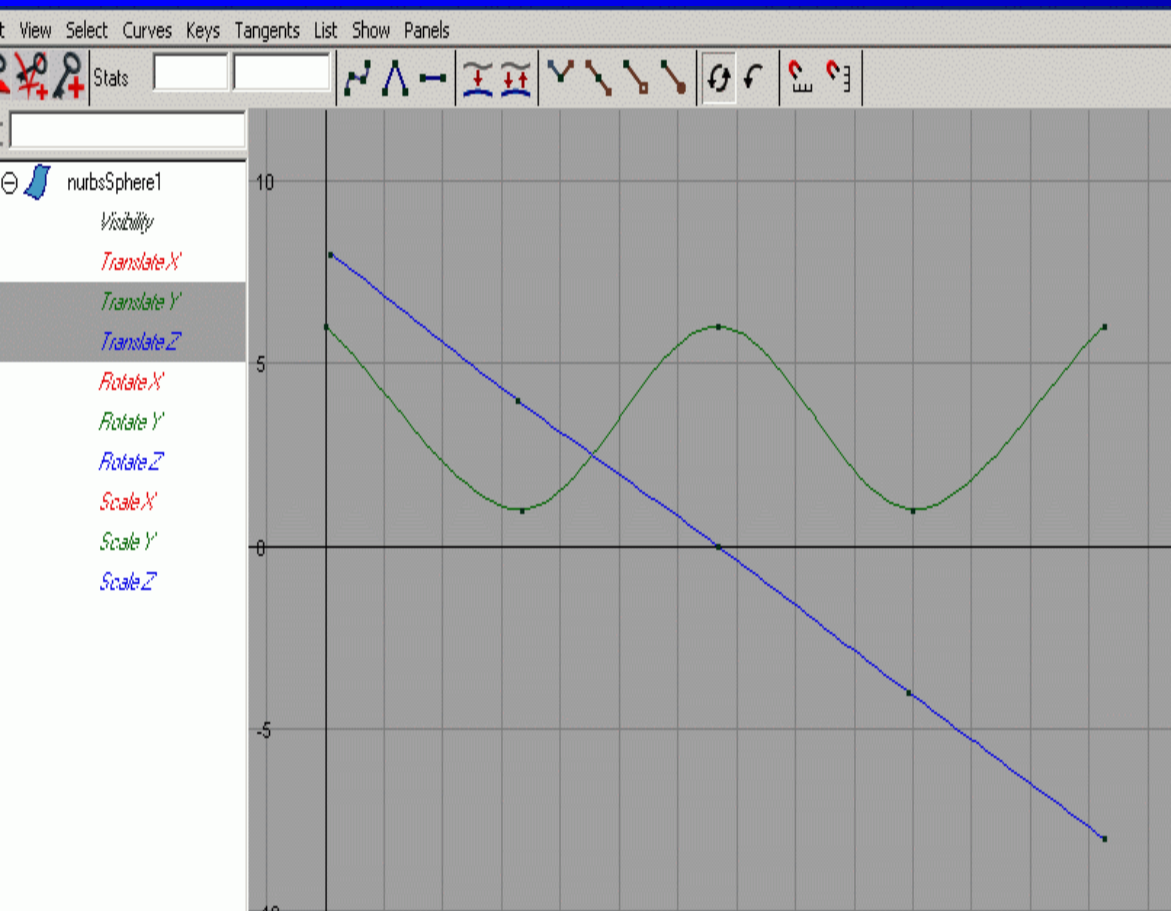
Keyframe

animáció

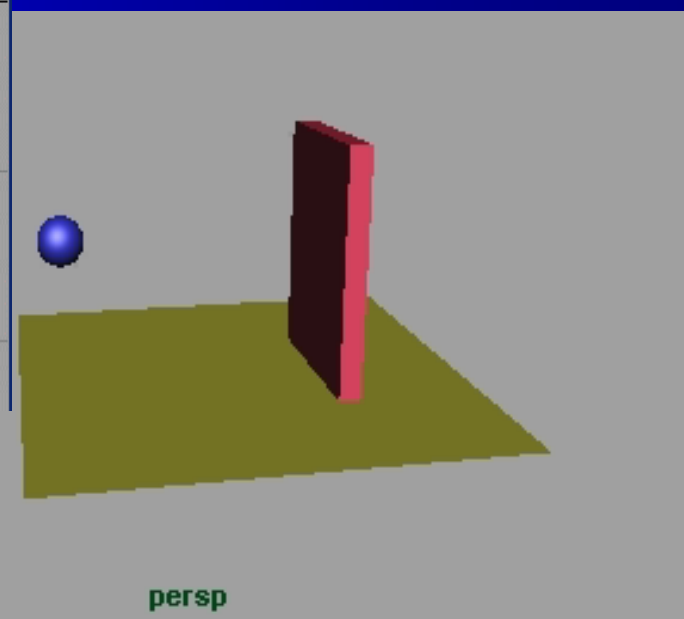
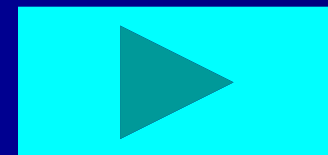
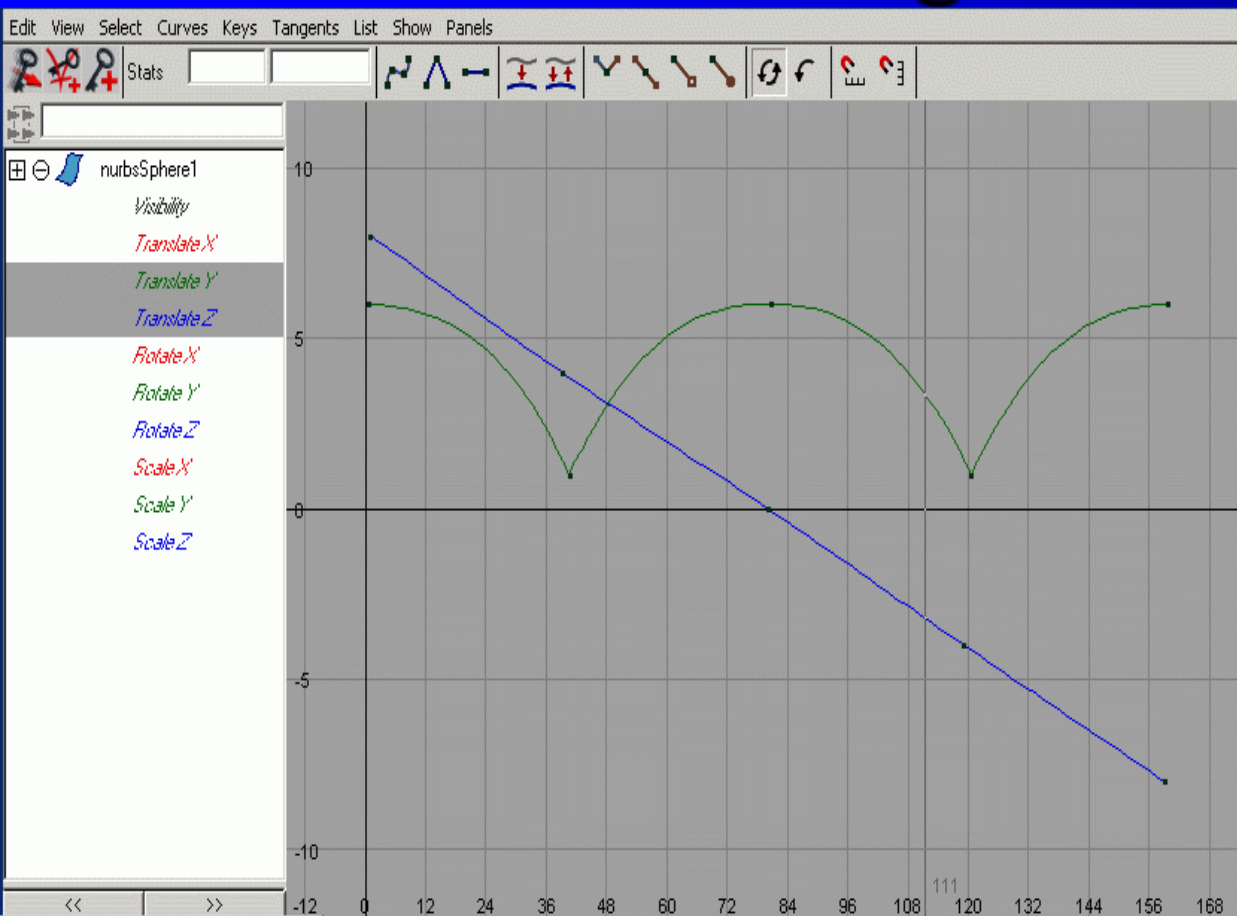


5.key

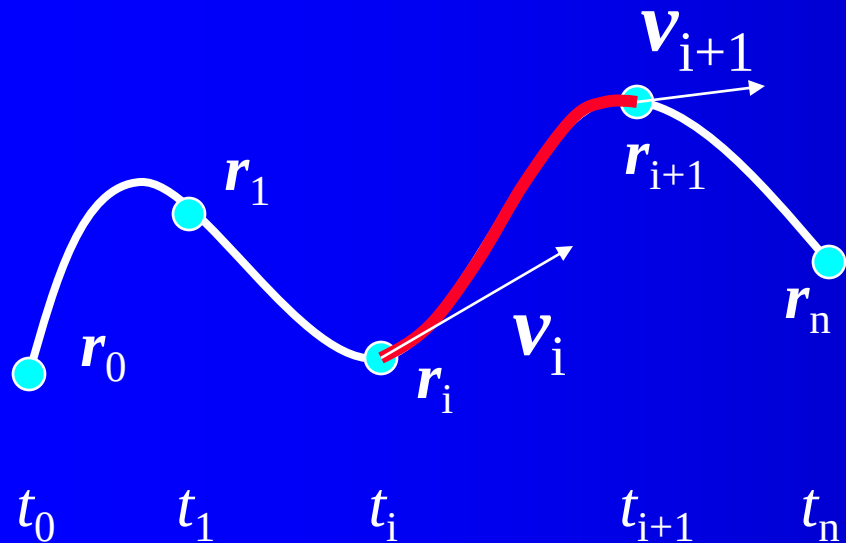
Keyframe animációban



Görbék megváltoztatása



Interpoláció: 3. lépés: a kiterjesztett Lagrange polinom



$$r_i(t) = a_i t^3 + b_i t^2 + c_i t + d_i$$

$$\begin{aligned} r_i(t_i) &= r_i, & r_i(t_{i+1}) &= r_{i+1} \\ r_i'(t_i) &= v_i, & r_i'(t_{i+1}) &= v_{i+1} \end{aligned}$$

Ismeretlen v_i -ket megkérdezzük:

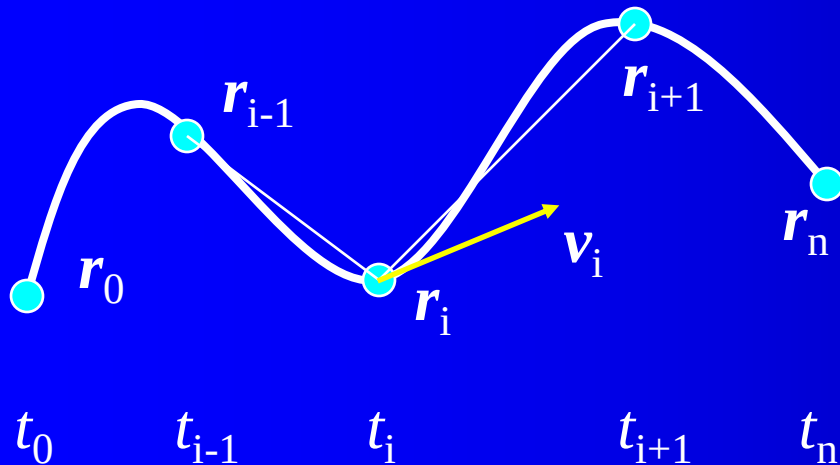
- C^2 folytonosság kikötése:

$$r_i'(t_{i+1}) = (r_{i+1}'(t_{i+1})) \quad (\text{és sebességek egyeznek})$$

- Tevékenység: párosítsd a Bartels, Catmull-Rom

Kohanek-Bartels (Catmull-Rom)

„spline polinome“



$$\mathbf{v}_i = \frac{1}{2} \left[\frac{\mathbf{r}_{i+1} - \mathbf{r}_i}{t_{i+1} - t_i} + \frac{\mathbf{r}_i - \mathbf{r}_{i-1}}{t_i - t_{i-1}} \right]$$

β $(1-\beta)$

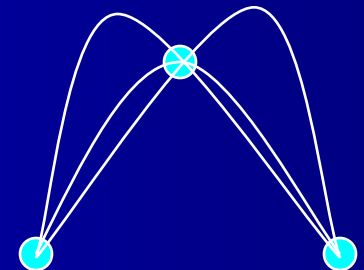
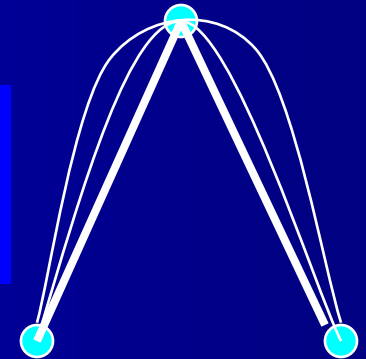
$\tau = 1$
Catmull-Rom

tension

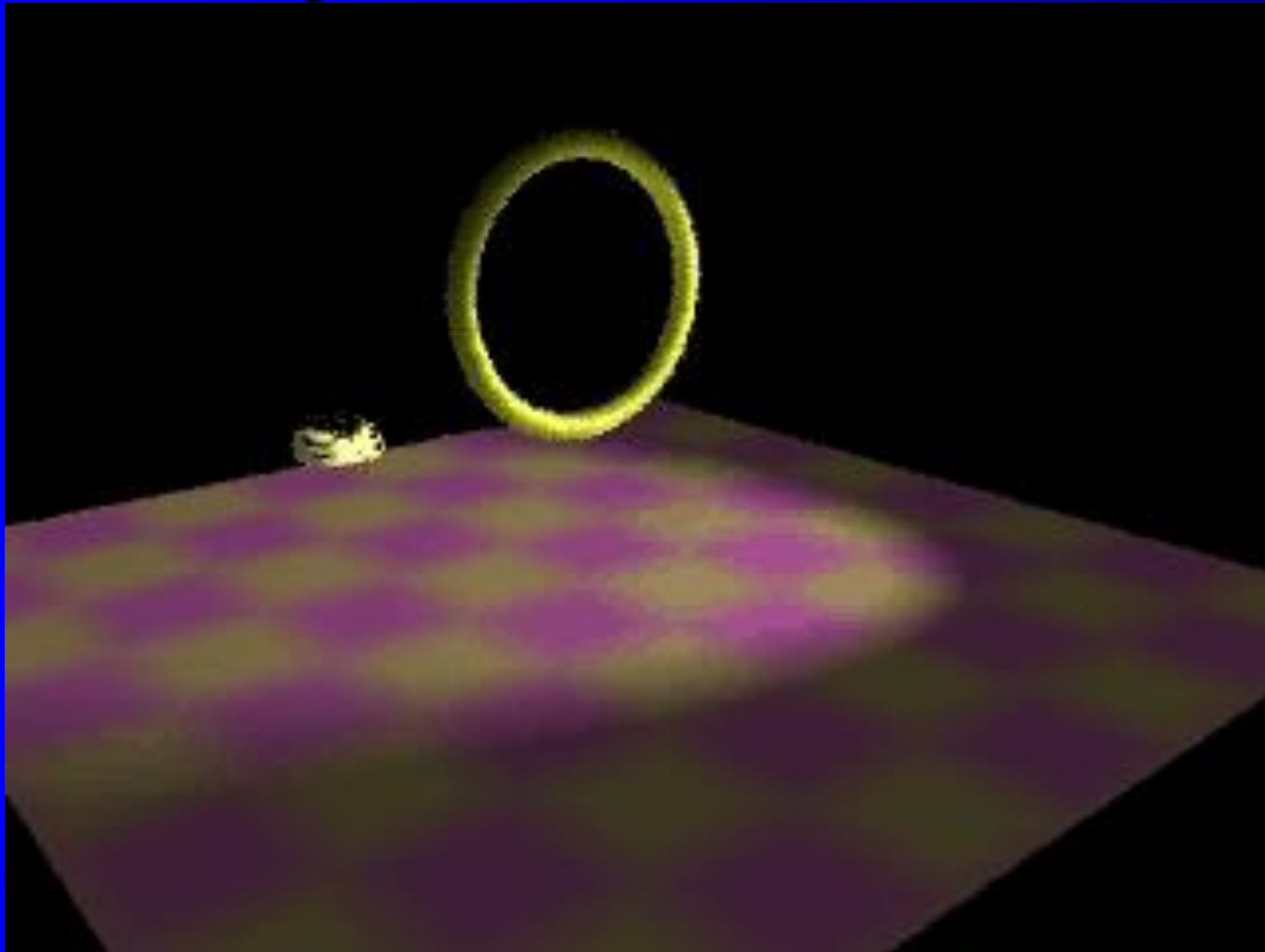
paramètre

$(1-\tau)$

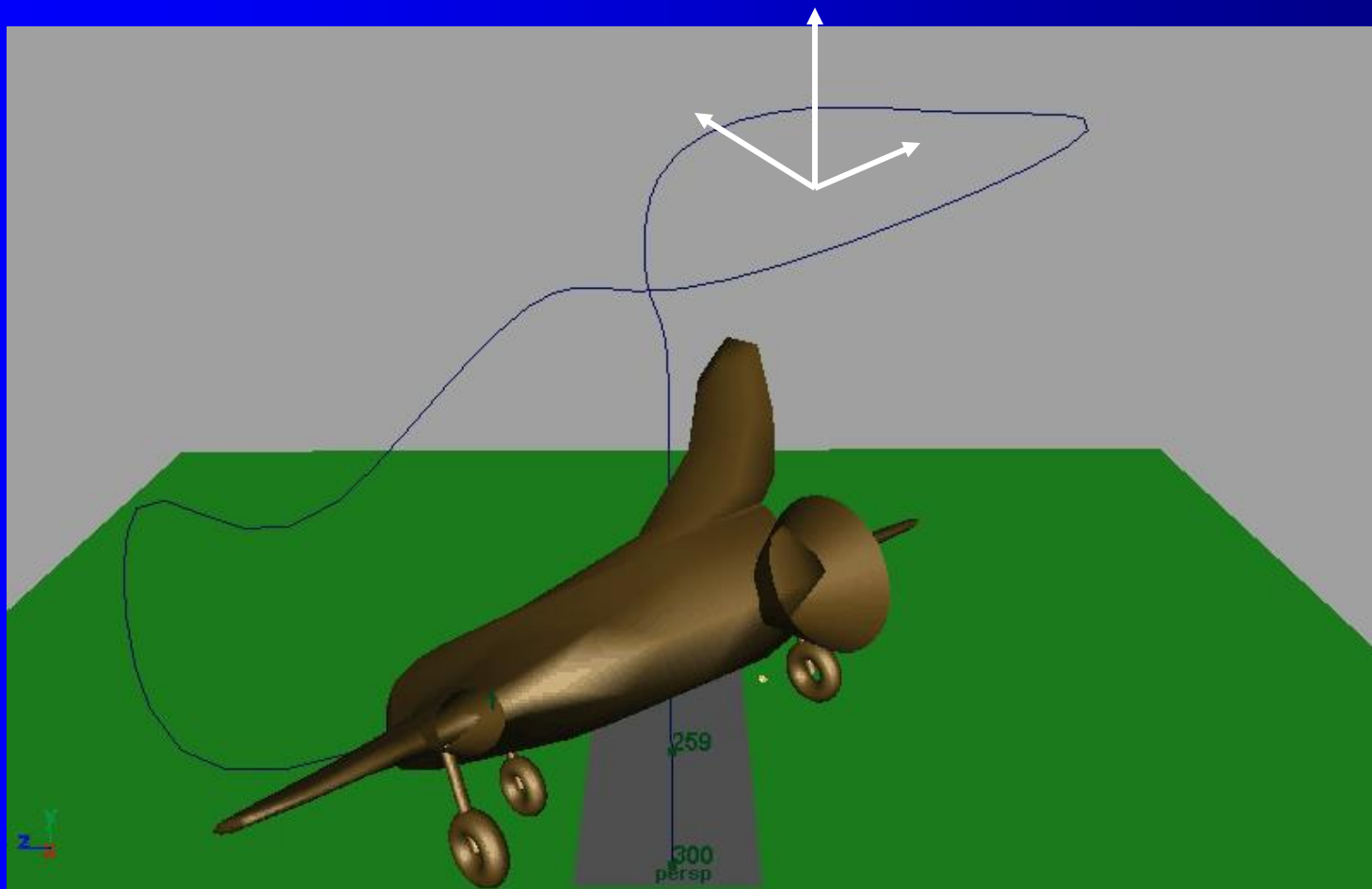
$\beta = 0.5$



Keyframe animation

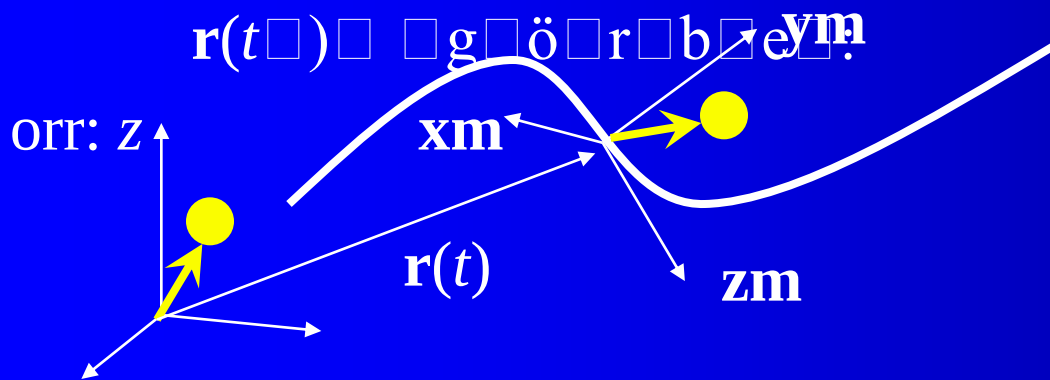


Polya (path) animation



$t =$ spline parameter

Pályaelőírás: Transzformáció



$$T_M = \begin{pmatrix} x_m^0(t) & 0 \\ y_m^0(t) & 0 \\ z_m^0(t) & 0 \\ r(t) & 1 \end{pmatrix}$$

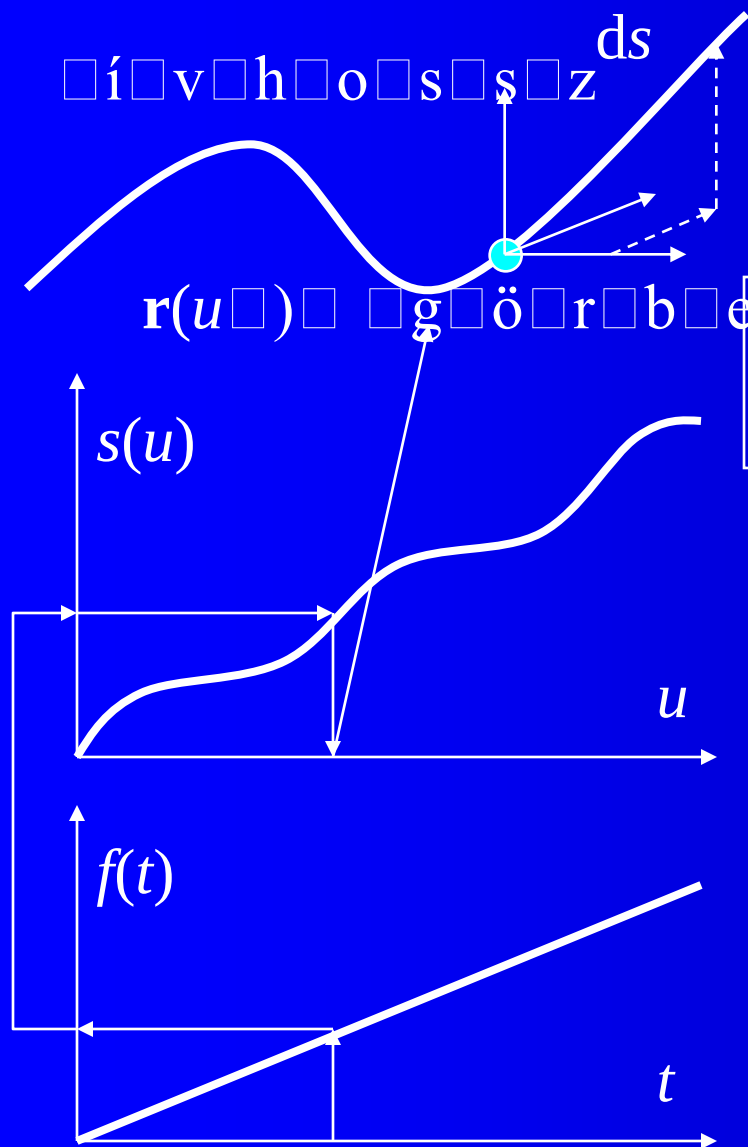
Explicit up vektor

$$\begin{aligned} z_m &= r'(t) \\ x_m &= z_m \otimes \uparrow \\ y_m &= z_m \otimes x_m \end{aligned}$$

Frenet keretek:

$$\begin{aligned} z_m &= r'(t) \\ x_m &= z_m \otimes r''(t), \quad (A = f''g - g''f) \\ y_m &= z_m \otimes x_m \end{aligned}$$

Ívhoss szerinti szövezés



$$dz = \frac{dz}{du} du$$

$$s(u) = \int_0^u \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2 + \left(\frac{dz}{du}\right)^2} du$$

Ívhossz szerinti szövezés

Megittét 5t füüt gg

$$s(u) = f(t), \quad \text{példá } u$$

$$u = s^{-1}(f(t))$$

$$\mathbf{r} = \mathbf{r}(u)$$

Fizikai animációk

- Erő (pálya, turbulencia stb.)
- Tömeg, nyomaték
- Ütközés (meztetés)
- Ütközés + rugó, + impulzus

"ROLL OVER"

$$R_x(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

ROTATION



rossz !

$K = \frac{1}{2} I \omega^2$
 $L = I \omega$

"FETCH"



BALL

$$x = (v_0 \cos \alpha_0) t$$

$$y = (v_0 \sin \alpha_0) t - \frac{1}{2} g t^2$$

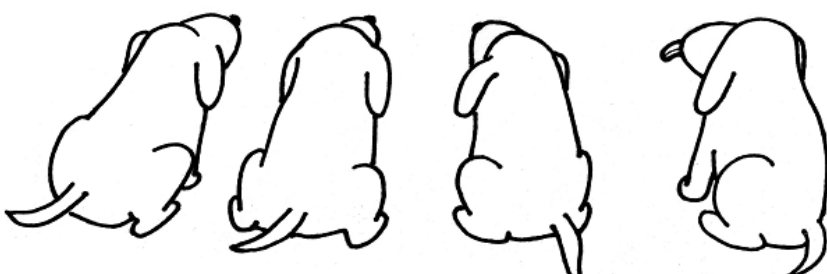
$$v_x = v_0 \cos \alpha_0$$

$$v_y = v_0 \sin \alpha_0 - g t$$

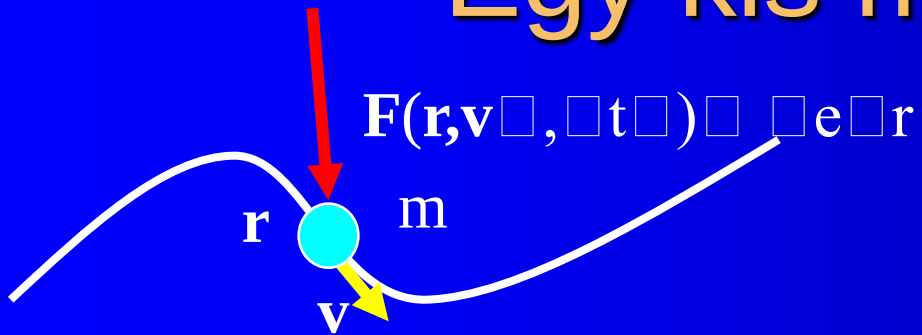
AIR RESISTANCE

$$m \dot{v}_x = -c \sqrt{v_x^2 + v_y^2} v_x$$

$$m \dot{v}_y = -mg - c \sqrt{v_x^2 + v_y^2} v_y$$

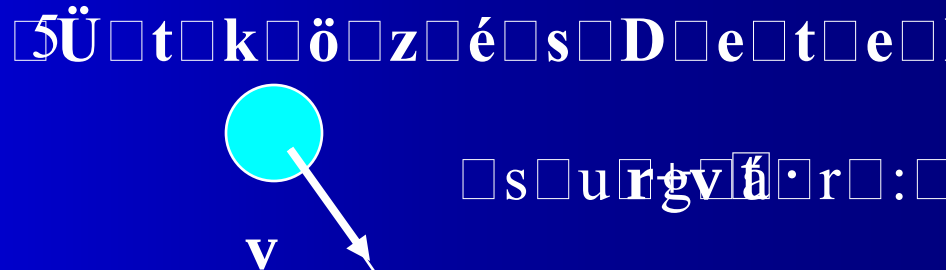
Egy kis mechanika



$$d\mathbf{r}/dt = \mathbf{v}$$

$$d\mathbf{v}/dt = \mathbf{F}(\mathbf{r}, \mathbf{v}, t)/m$$

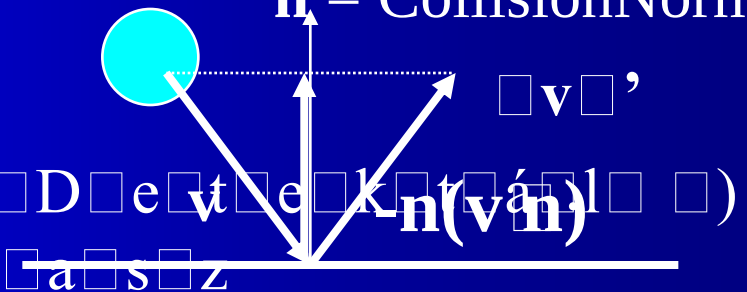
```
for( t = 0; t < T; t += dt) {
     $\mathbf{F} = \mathbf{F}_{\text{red}}(\mathbf{r}, \mathbf{v})$ 
     $\mathbf{a} = \mathbf{F}/m$ 
     $\mathbf{v} += \mathbf{a} \cdot dt$ 
    if (  $\mathbf{U}_{\text{tközésDetek}}$  )
         $\mathbf{U}_{\text{tközés}} \mathbf{V}_{\text{álalás}}$ 
     $\mathbf{r} += \mathbf{v} \cdot dt$ 
}
```



$\mathbf{m} \cdot \mathbf{v}^* \cdot \mathbf{z} \cdot \mathbf{é} \cdot \mathbf{s} : \mathbf{t}$

Ha $t^* < dt$ Collision

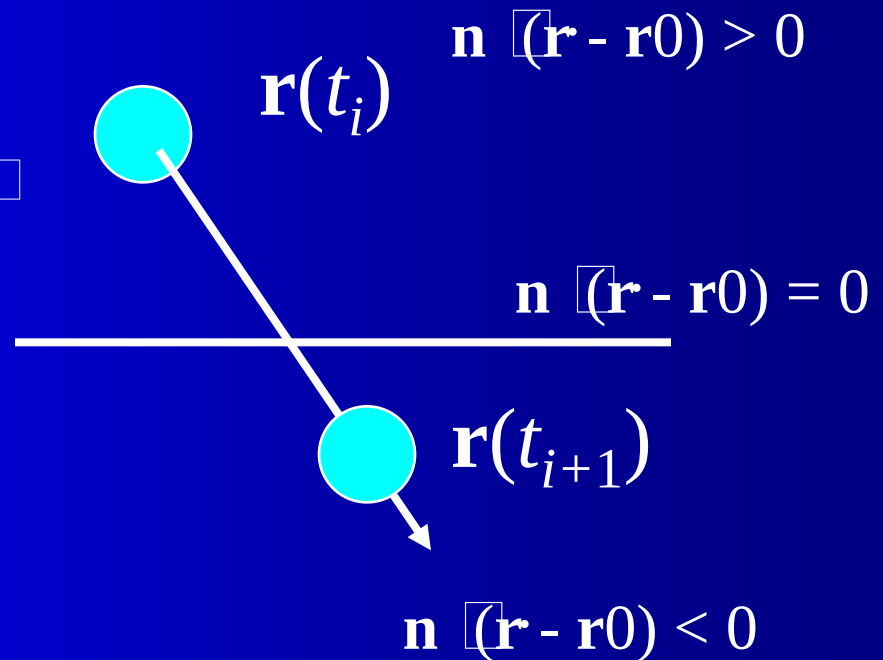
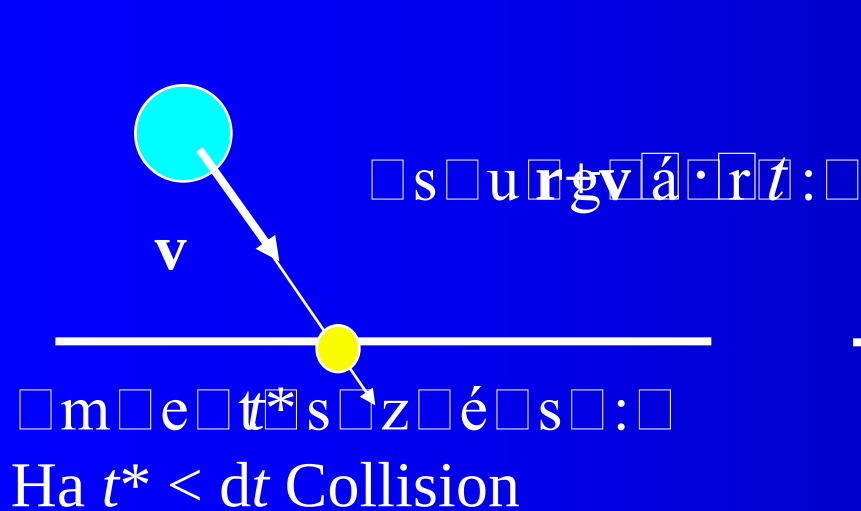
$\mathbf{U}_{\text{tközés}} \mathbf{V}_{\text{álalás}}$
 $\mathbf{n} = \text{CollisionNormal}$



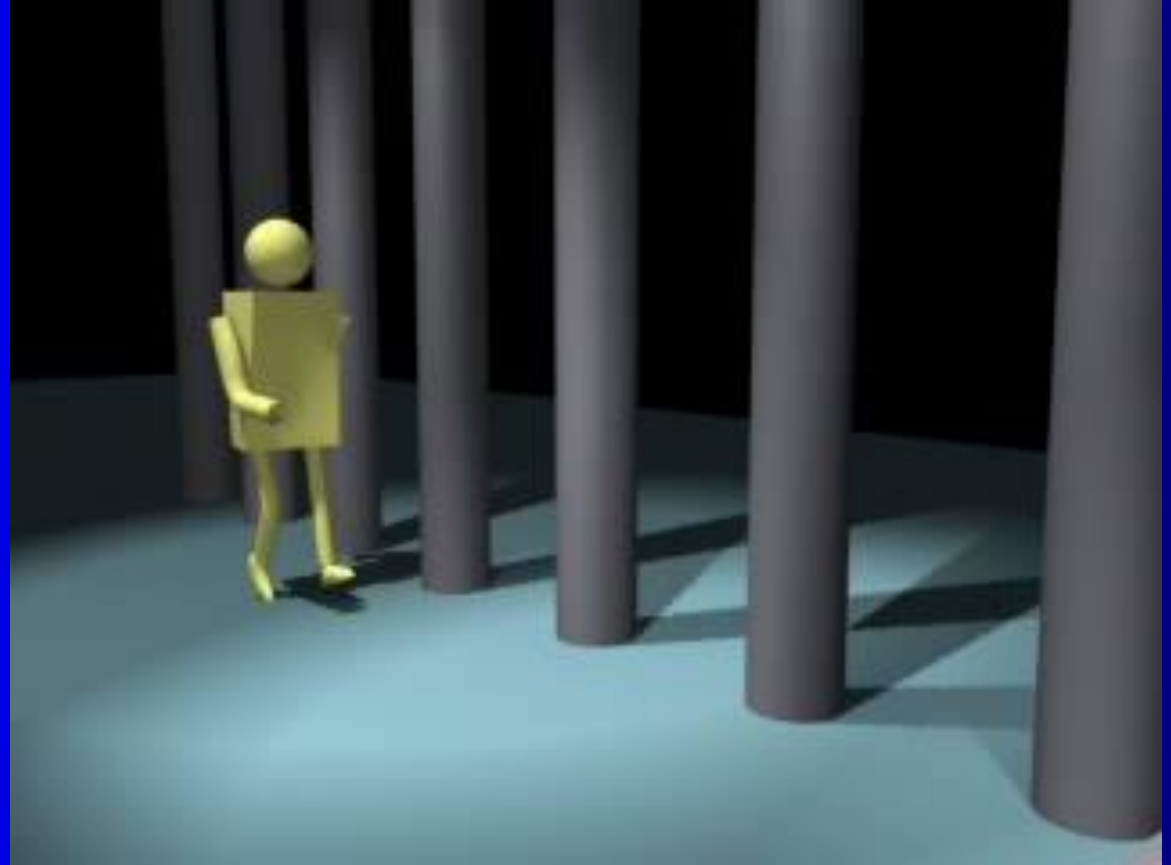
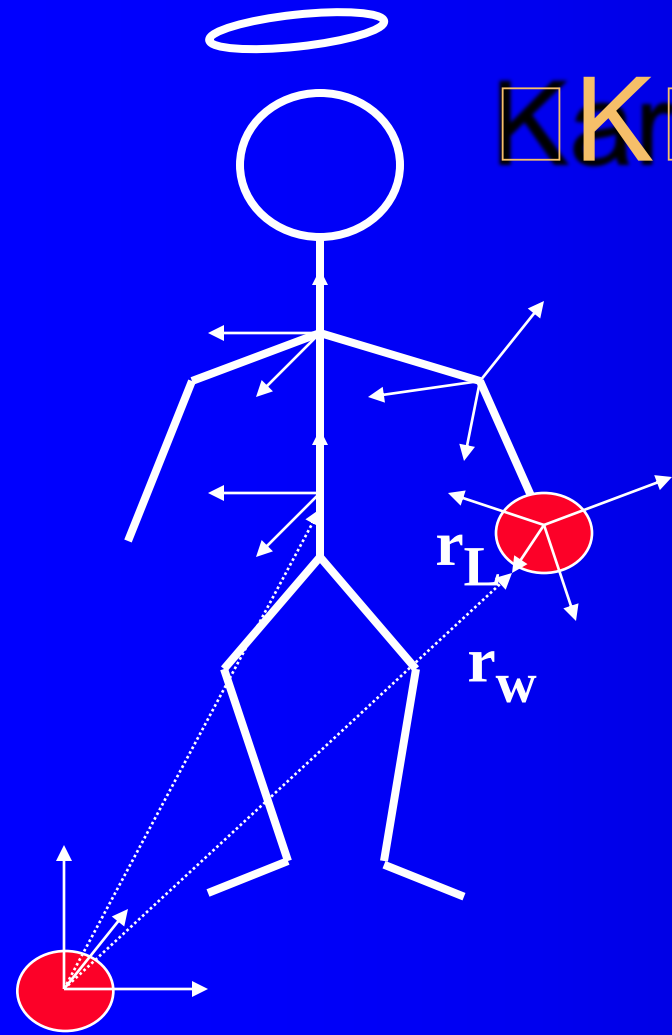
$$\mathbf{v} = [\mathbf{v} - \mathbf{n}(\mathbf{v} \cdot \mathbf{n})] - [\mathbf{n}(\mathbf{v} \cdot \mathbf{n}) \cdot \text{bounce}]$$

Folytonos-Diszkrét szimuláció

Ütközés-détektálás és megoldás



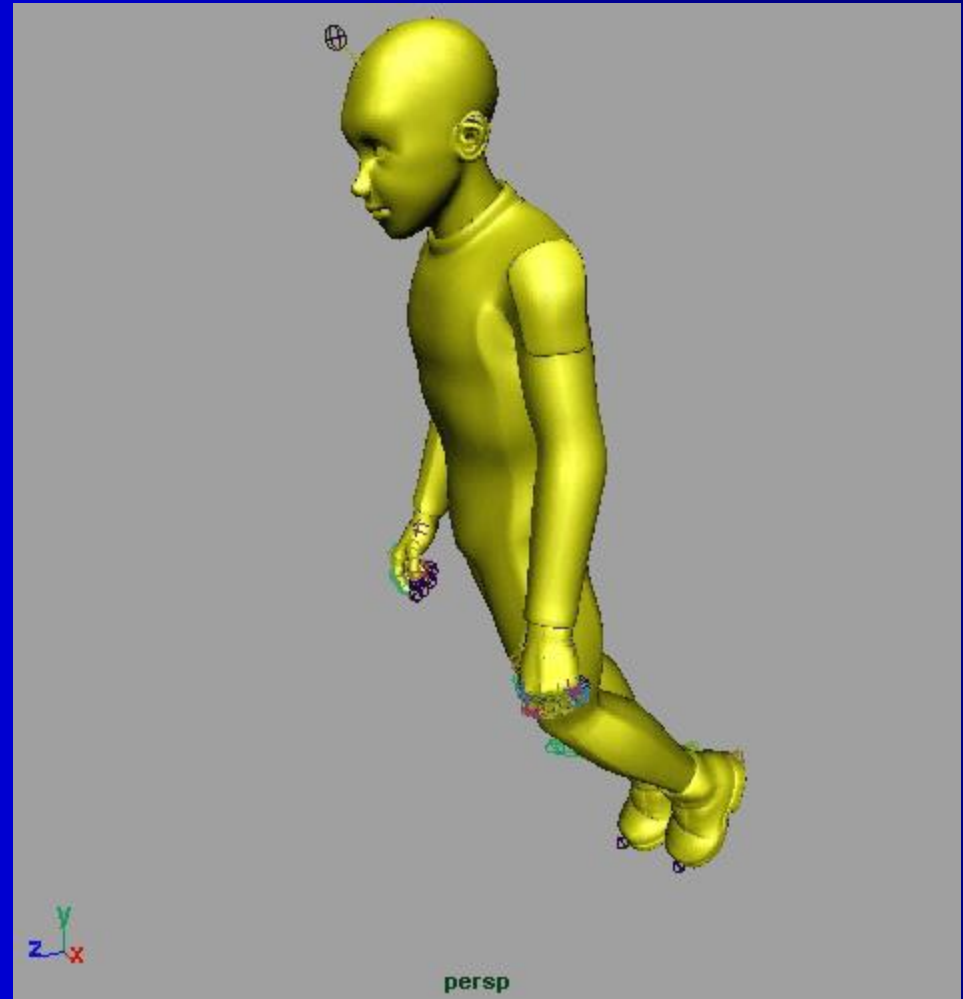
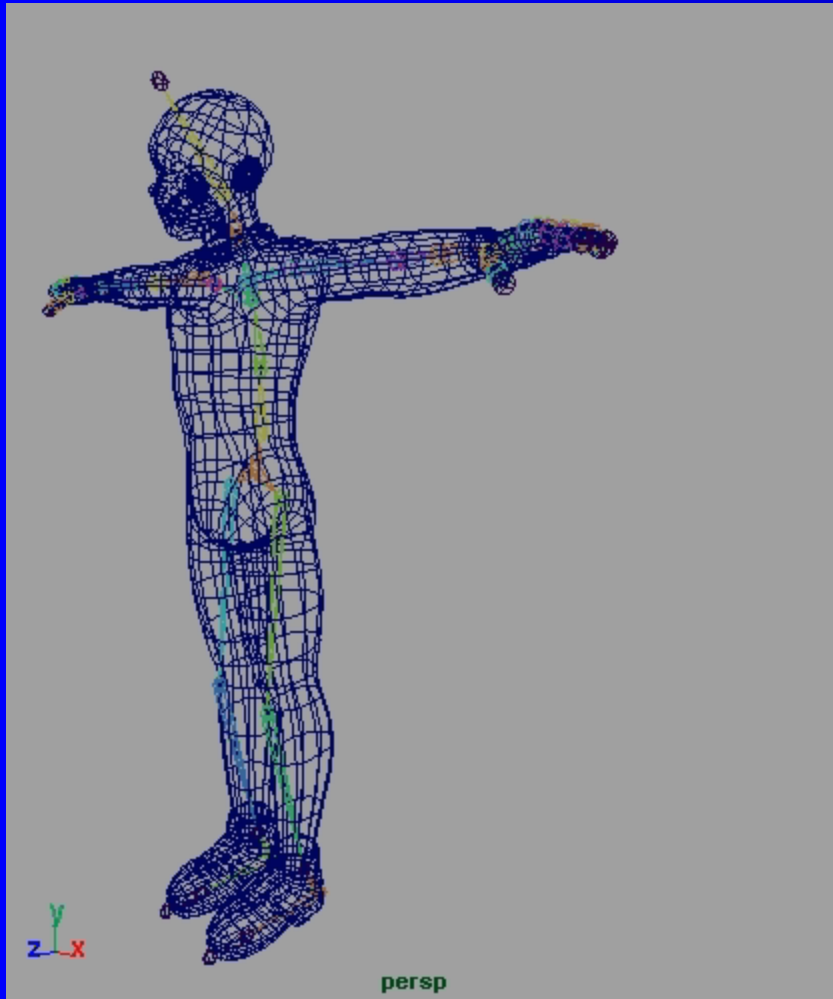
Karakteranimáció



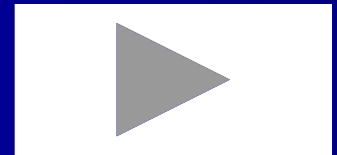
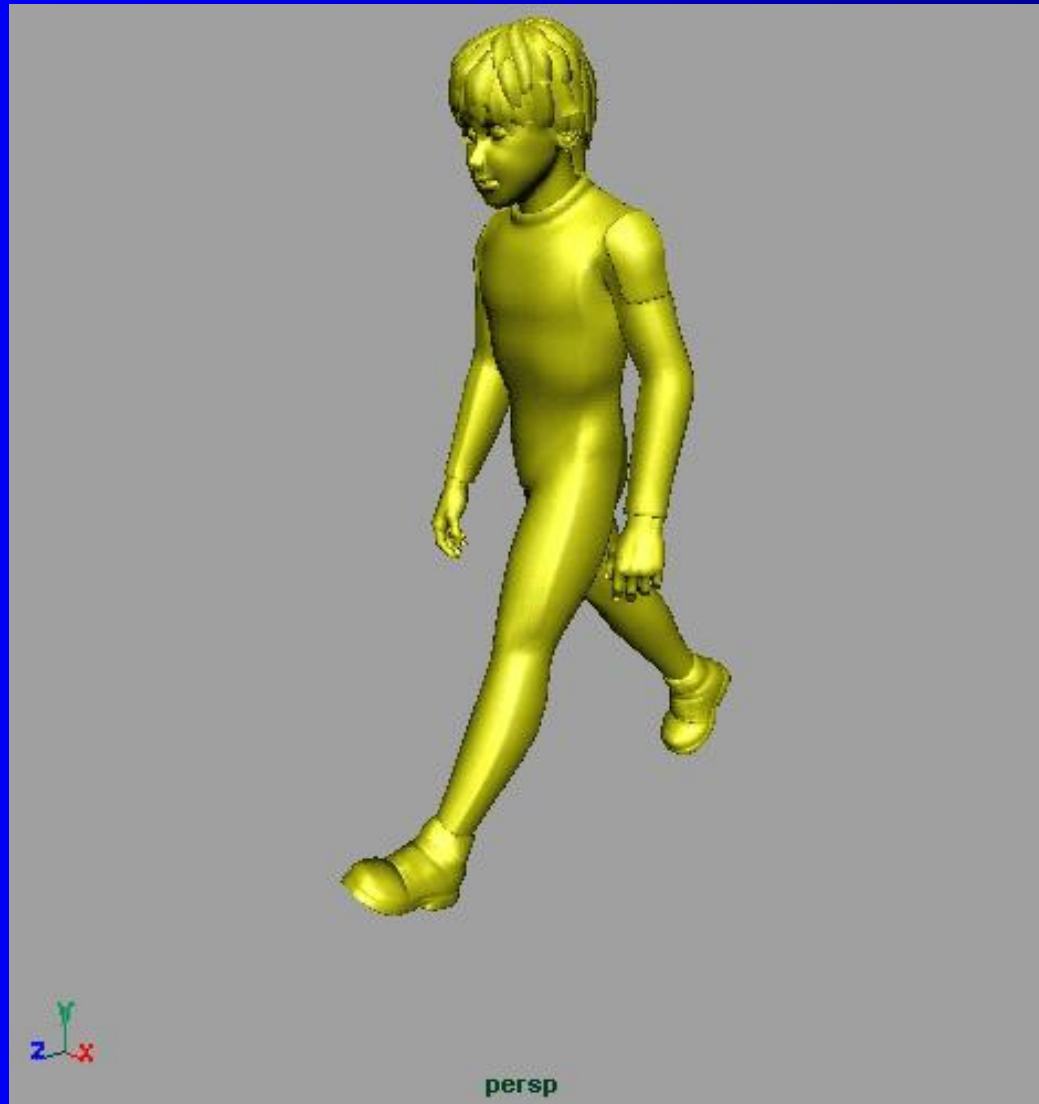
$r_w \equiv r_L$
 $R_{k \rightarrow a}$
 $T_{a \rightarrow k}$
 $R_{k \rightarrow o}$
 $T_{o \rightarrow k}$
 $R_{v \rightarrow a}$
 $T_{a \rightarrow v}$
 T_{ember}

homogén koordináta 4

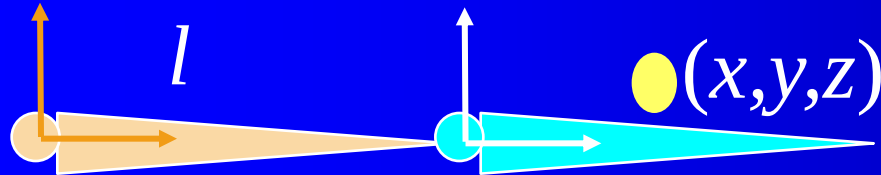
Csontváza



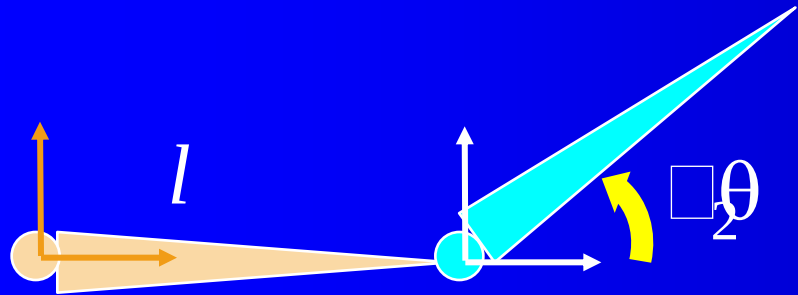
Séta



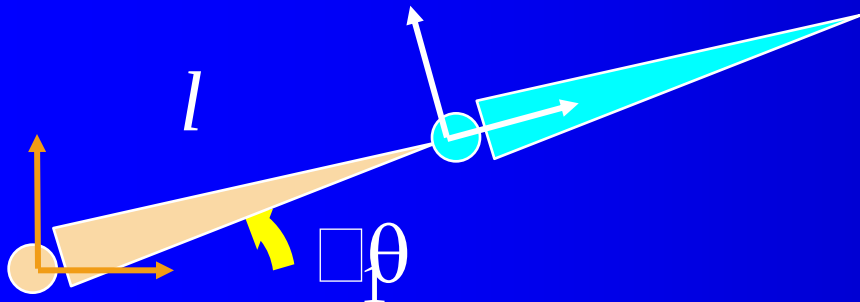
Transformáció hierarchiája



```
glTranslatef(l, 0, 0);  
glVertex3f(x, y, z);
```

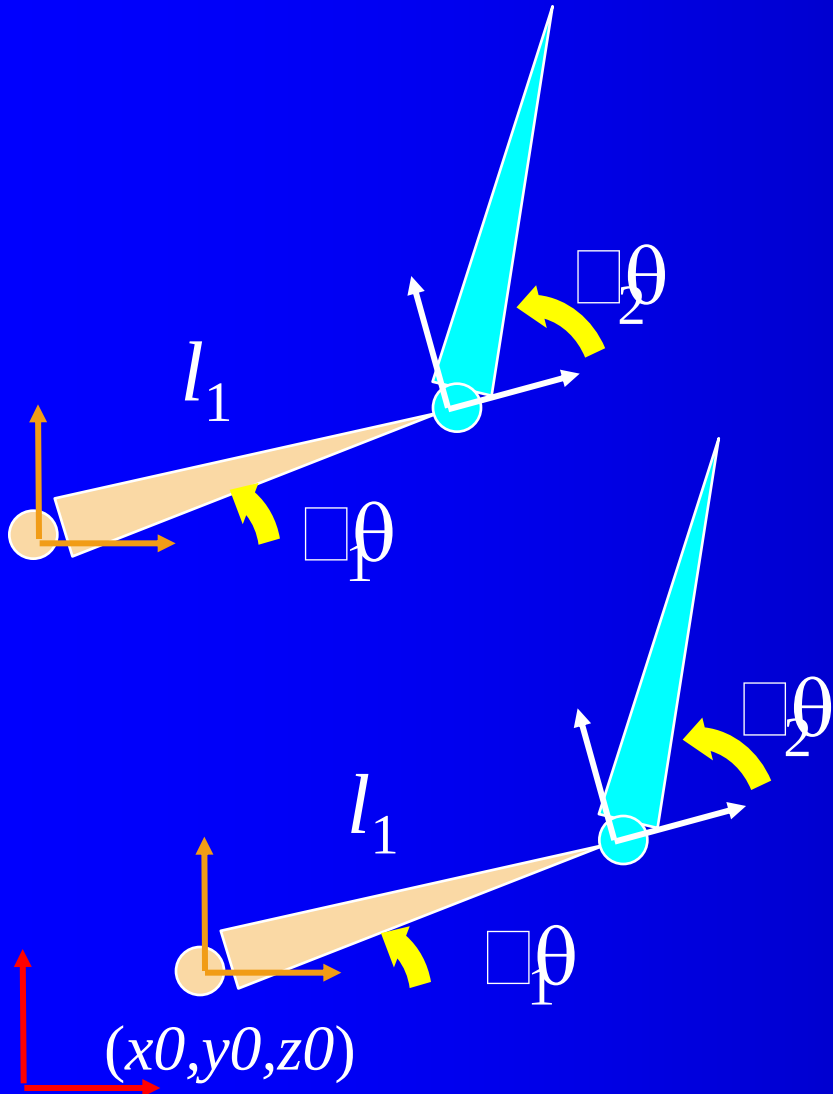


```
glTranslatef(l, 0, 0);  
glRotatef(theta2, 0, 0, 1)  
glVertex3f(x, y, z);
```



```
glRotatef(theta1, 0, 0, 1)  
glTranslatef(l, 0, 0);  
glVertex3f(x, y, z);
```

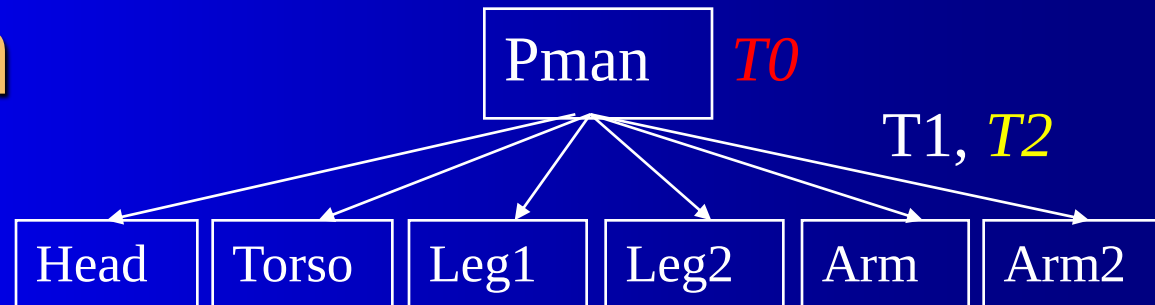
Transzformáció hierarchia



```
glRotatef(theta1, 0, 0, 1)  
glTranslatef(l, 0, 0);  
glRotatef(theta2, 0, 0, 1)  
glVertex3f(x, y, z);
```

```
glTranslatef(x0, y0, z0);  
glRotatef(theta1, 0, 0, 1)  
glTranslatef(l, 0, 0);  
glRotatef(theta2, 0, 0, 1)  
glVertex3f(x, y, z);
```

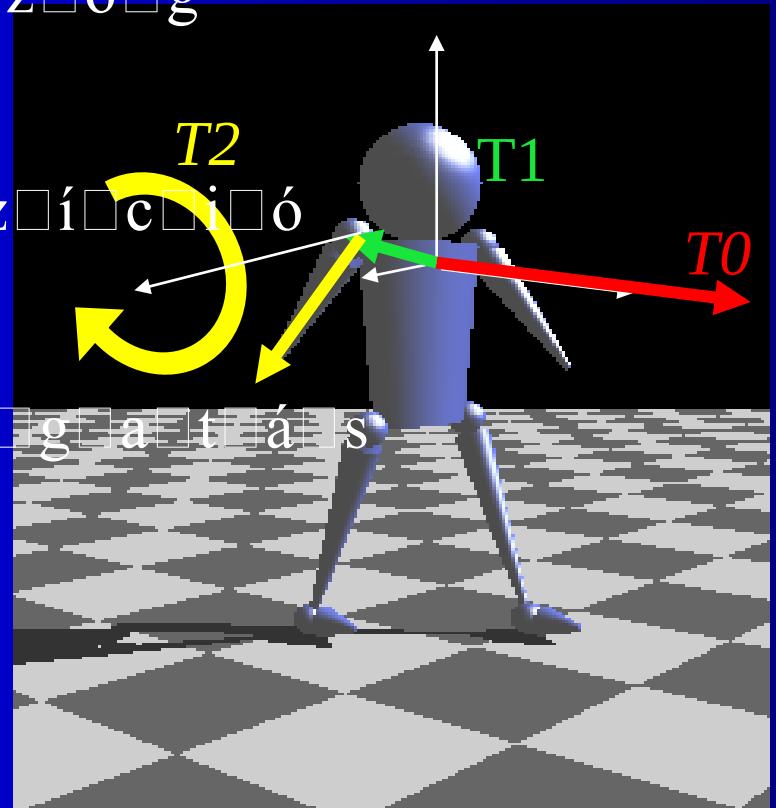
PMan



$T_0 = \text{Pman} \rightarrow \text{e} \rightarrow \text{l} \rightarrow \text{5} \rightarrow \text{r} \rightarrow \text{e} \rightarrow \text{m} \rightarrow \text{o} \rightarrow \text{z} \rightarrow \text{o} \rightarrow \text{g}$
 $\text{glTranslatef}(xpm, ypm, zpm);$

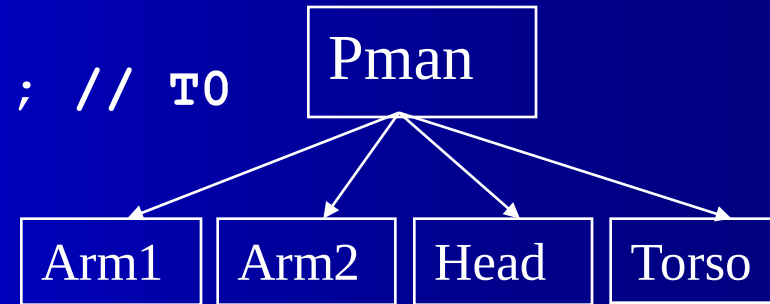
$T_1 = \text{v} \rightarrow \text{á} \rightarrow \text{l} \rightarrow \text{l} \rightarrow \text{p} \rightarrow \text{o} \rightarrow \text{z} \rightarrow \text{í} \rightarrow \text{c} \rightarrow \text{i} \rightarrow \text{o}$
 $\text{glTranslatef}(xsh, ysh, zsh);$

$T_2 = \text{v} \rightarrow \text{á} \rightarrow \text{l} \rightarrow \text{l} \rightarrow \text{f} \rightarrow \text{o} \rightarrow \text{r} \rightarrow \text{g} \rightarrow \text{l} \rightarrow \text{a} \rightarrow \text{t} \rightarrow \text{á} \rightarrow \text{s}$
 $\text{glRotatef}(\text{angle}, xshax, yshax, zshax);$



Pman rajzolás és animálás

```
void DrawPman(float dt) {  
    xpm += vx*dt; ypm += vy*dt; zpm += vz*dt;  
    glPushMatrix();  
    glTranslatef(xpm, ypm, zpm); // T0
```



```
    angle += av * dt;  
    if (angle>30 || angle<-30) av*=-1;  
    glPushMatrix();  
        glTranslatef(xsh, ysh, zsh); // T1  
        glRotatef(angle, xshax, yshax, zshax); // T2  
        DrawArm( );
```

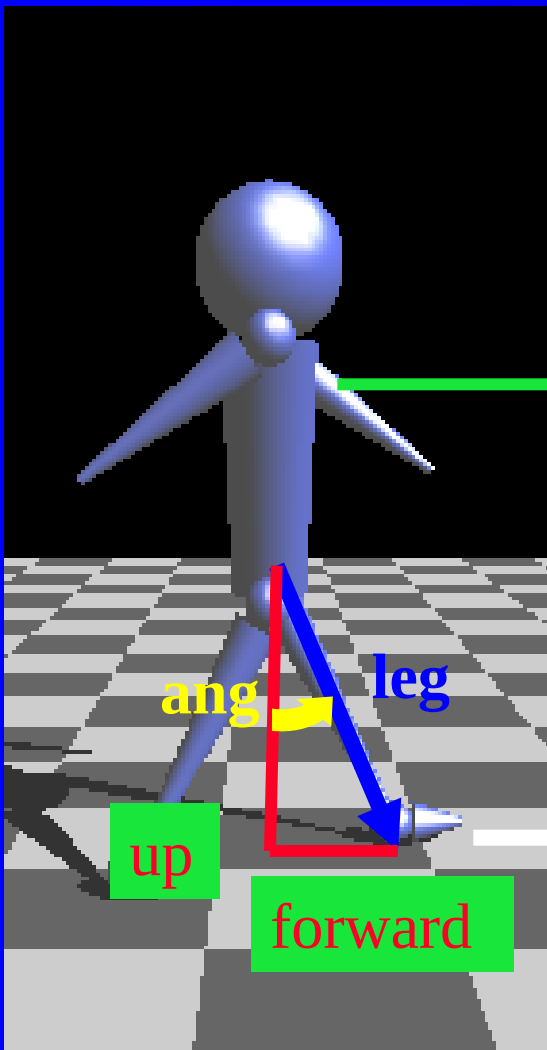
```
    glPopMatrix();
```

□ *Other arm, legs, torso, head*

```
    glPopMatrix();
```

```
}
```

Inverz kinematika



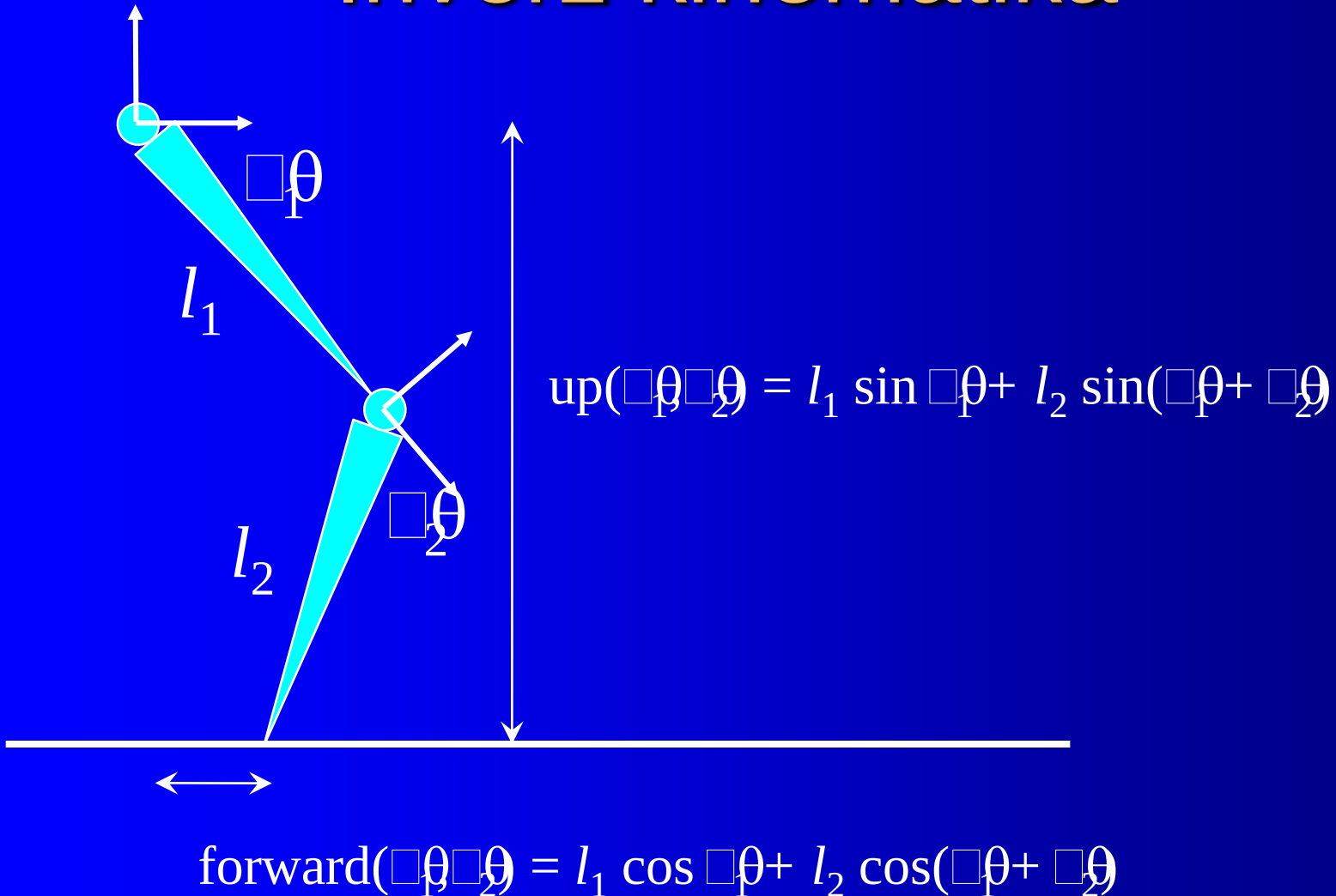
$T_0 = e_1 \quad 5 \text{ remozgá}$

$T_2 = l \text{ áb f o r g a t á}$

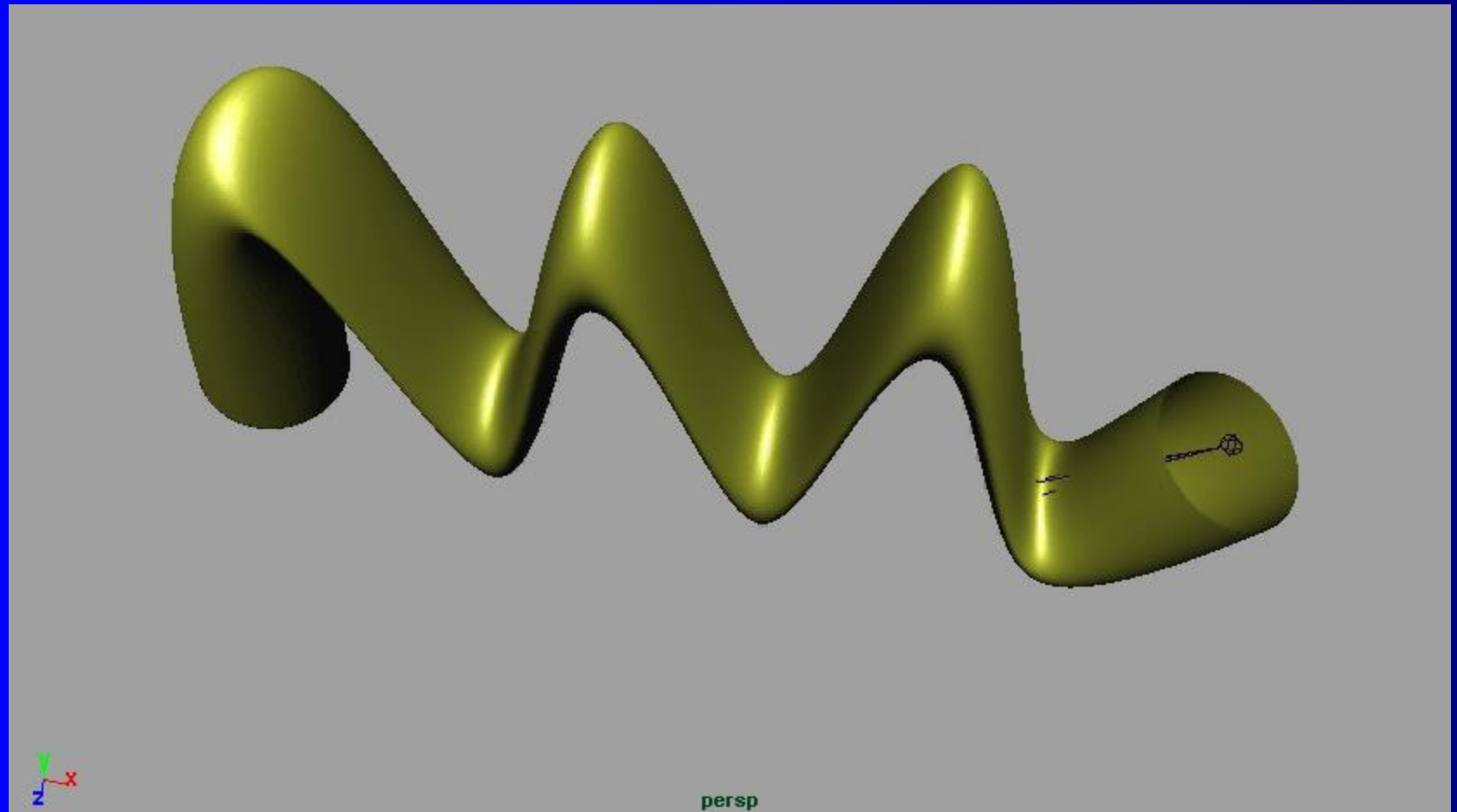
$T \text{ á m a s z k o d ó n}$

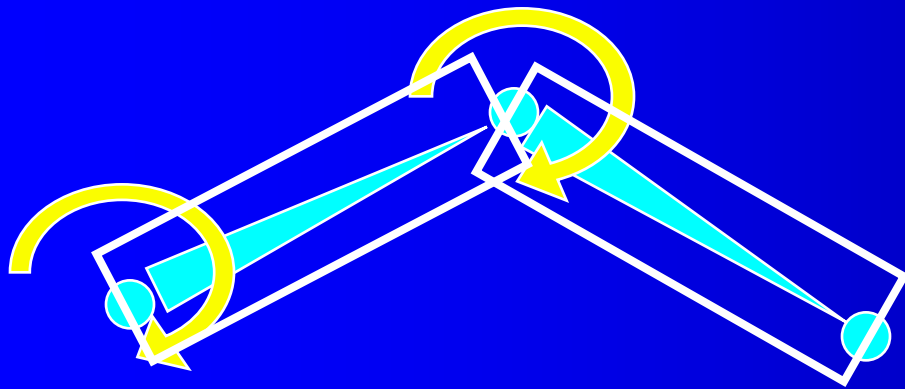
```
forward += leg * fabs(sin(angNew) - sin(angOld));
up       = leg * cos(angNew);
```

Inverz kinematika

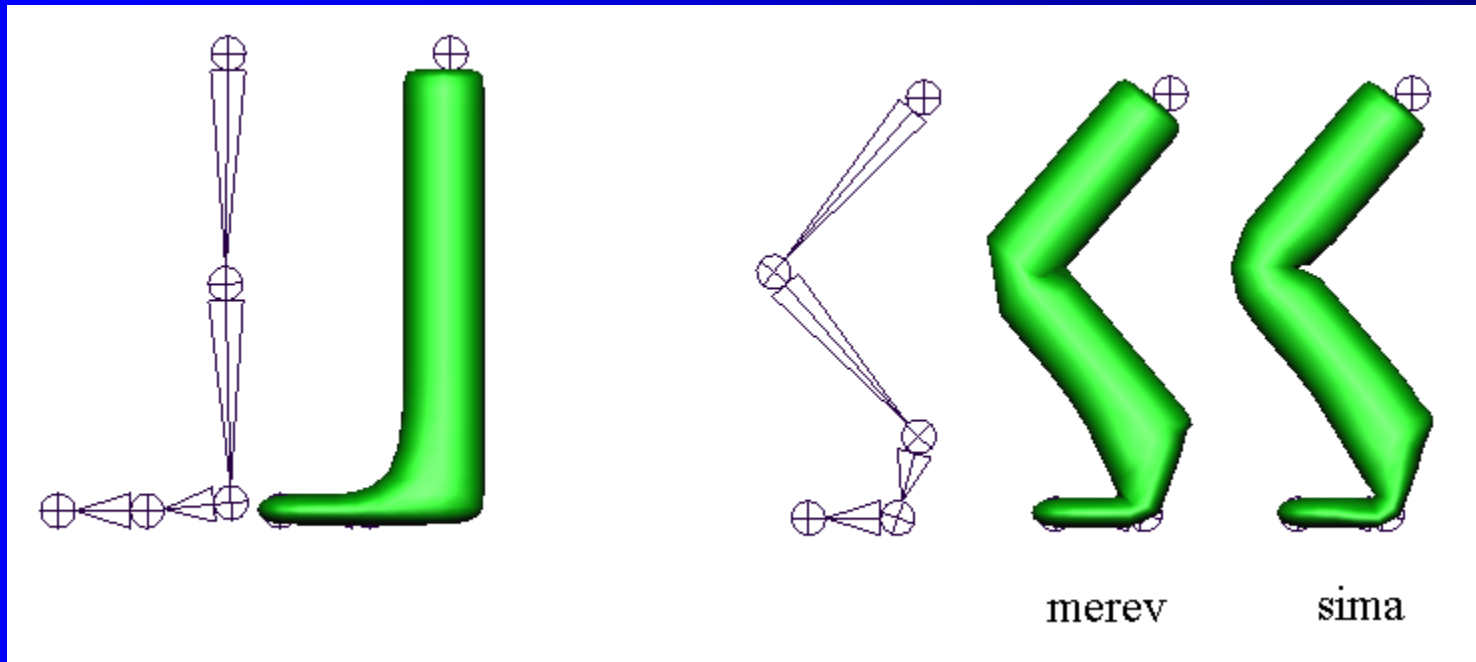


Börzses





M e r e v
 b ö r z ö s



5. házi h á z i

● G ó l y a l é

J: jobbra ford (0.5sec)

B: balra ford (0.5sec)

SP: lecsap (0.5sec)

S i m a b-n5

● B é k a :

u g r i k , m

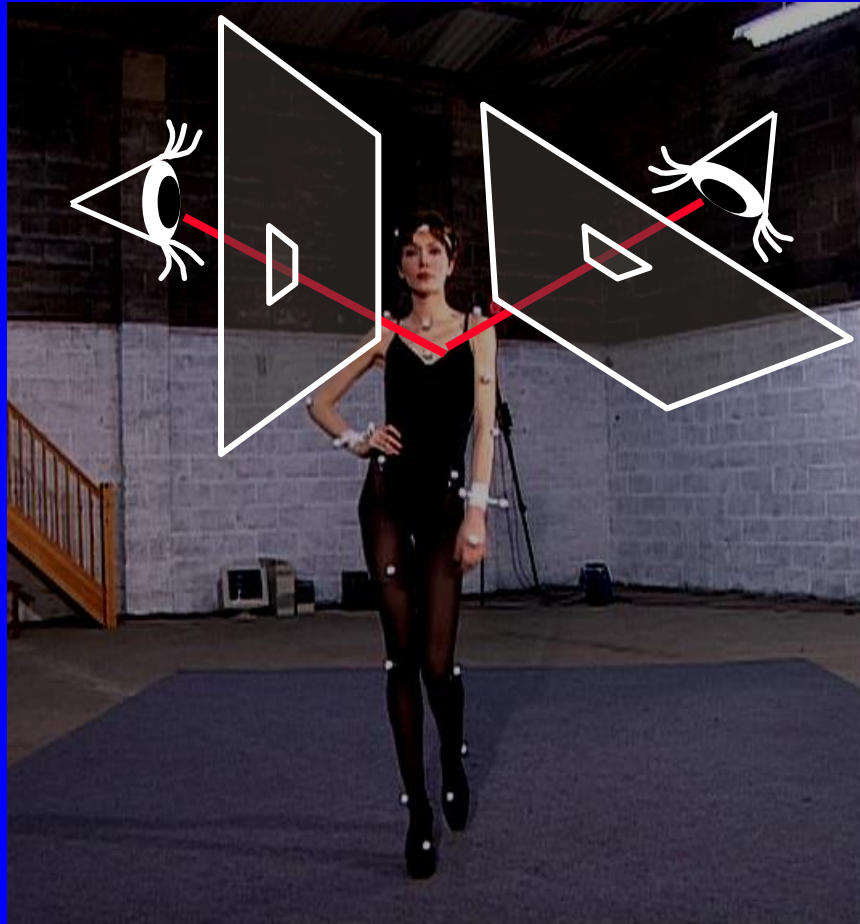
● S í k r a v e



5.5. h á z i

● Mozgás a m e g a 4. h á
1 1 é p é s / s e c s e b e s s é g
(inverz kinematika). A J' b y i C l e l n e t n y o n
a l a t t 1 0 f o k o t j o b b r a
k ö r ü l, B h' a b y i C l e l n e t n y o n m á
S P A C E h a t á s á r a a n y a
t a l á l (v é g t a g o k n e m
b é k á k m e r e v t e s t k é n t
d e k é t u g r á s k ö z ö t t 1
t e r e p e t i l l e t 5 e n k é t
e l s 5 b e n a t e r e p s í k é
a m i t a n a p v e t (s í k r a
l e h e t 5 s é g, h o g y a t e

Motion Capture animation



Stereó

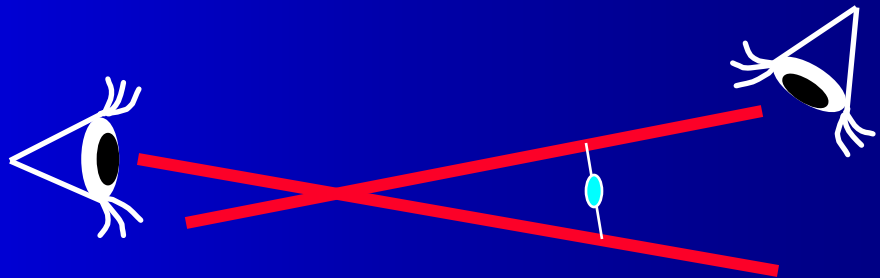
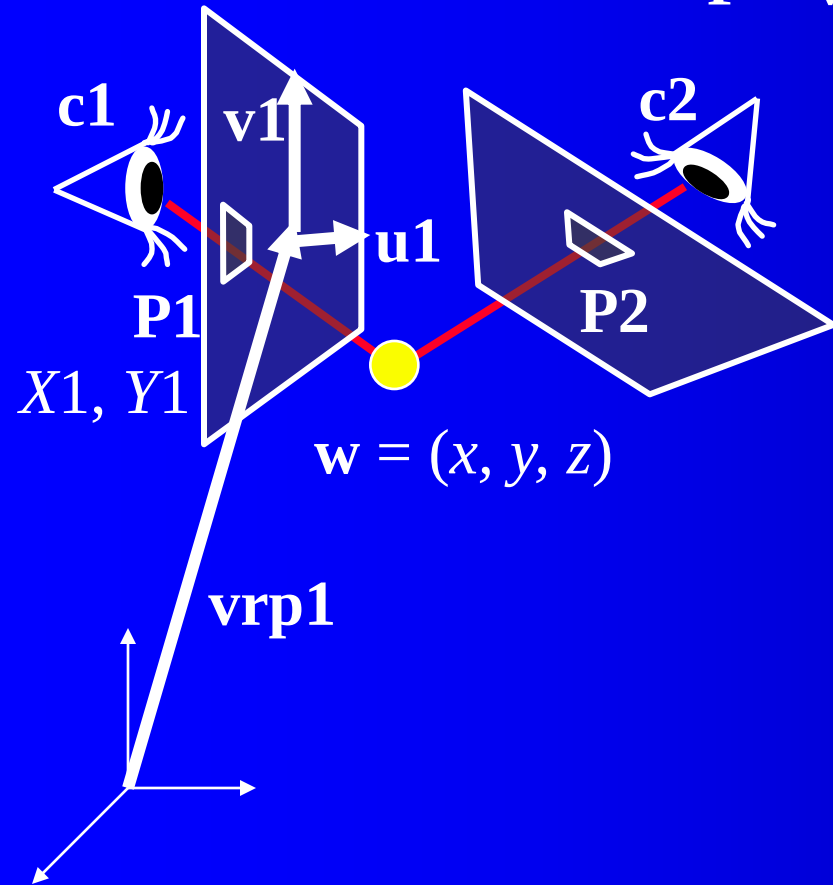
$$\mathbf{P} = \mathbf{vrp} + (2X/X_{MAX}-1)\mathbf{u} + (2Y/Y_{MAX}-1)\mathbf{v}$$

$$\mathbf{c1} + (\mathbf{P1}-\mathbf{c1})t1 = (x, y, z)$$

$$\mathbf{c2} + (\mathbf{P2}-\mathbf{c2})t2 = (x, y, z)$$

6 egyenlet, 5 ismeretlen $(x, y, z, t1, t2)$

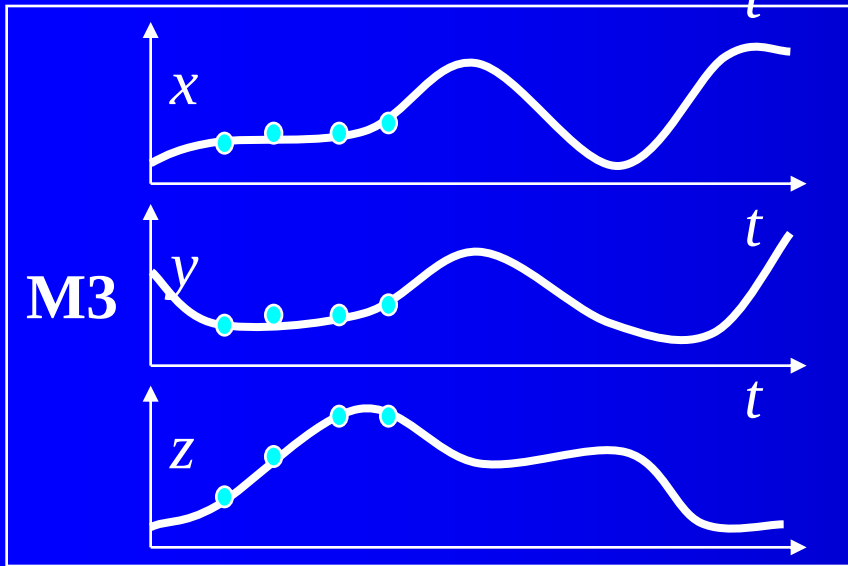
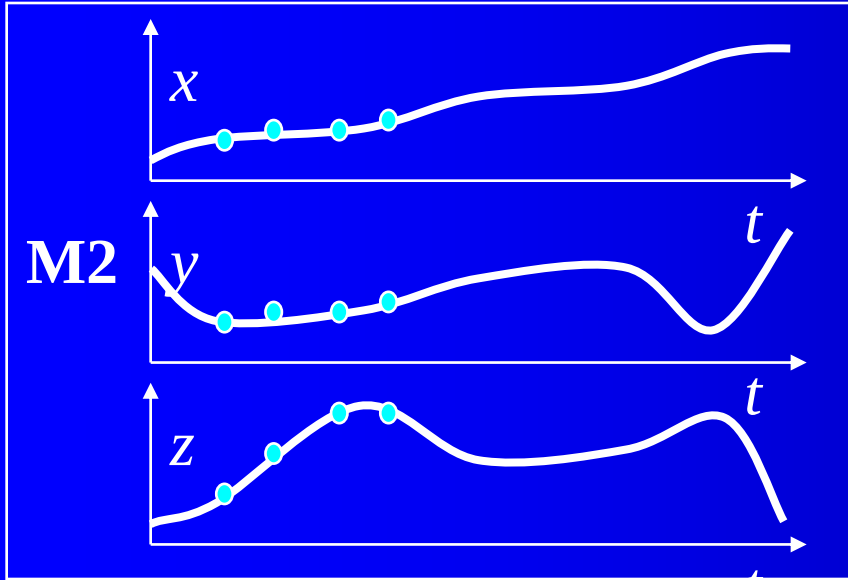
Hiba: 5



c, u, v, vrp meghatározás: kamer

Motion capture redundancy

Main design marker positions



M1

M2

M3

M4



$$\cos \phi (M3-M2)^0 (M2-M1)^0$$

tengely:

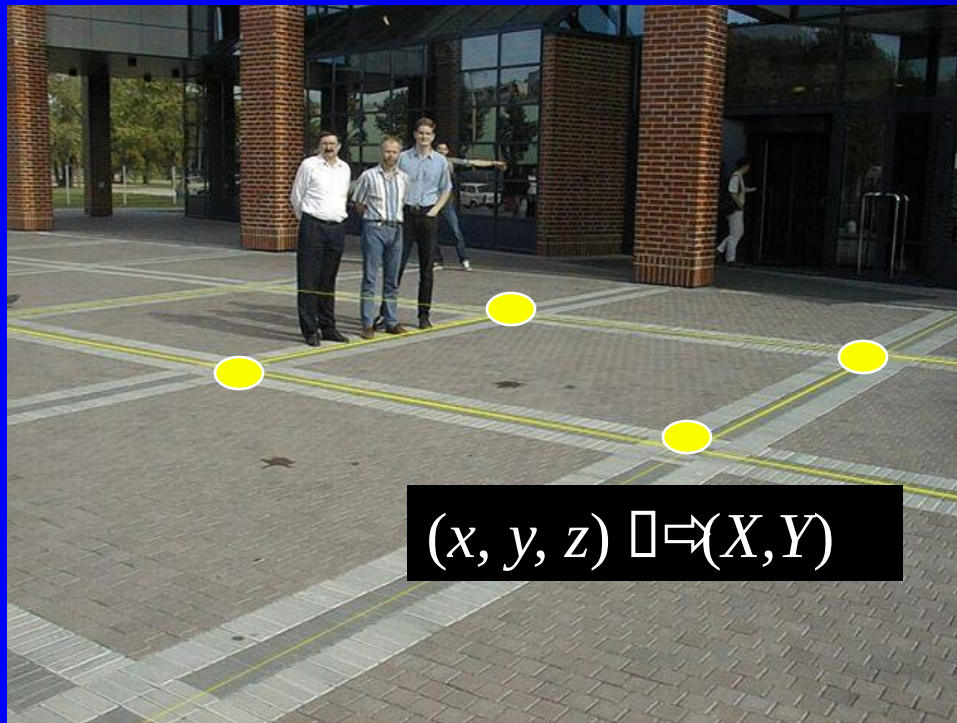
$$\mathbf{d} = (M3-M2) \times (M2-M1)$$

Valóságos és virtuális világok és képzése

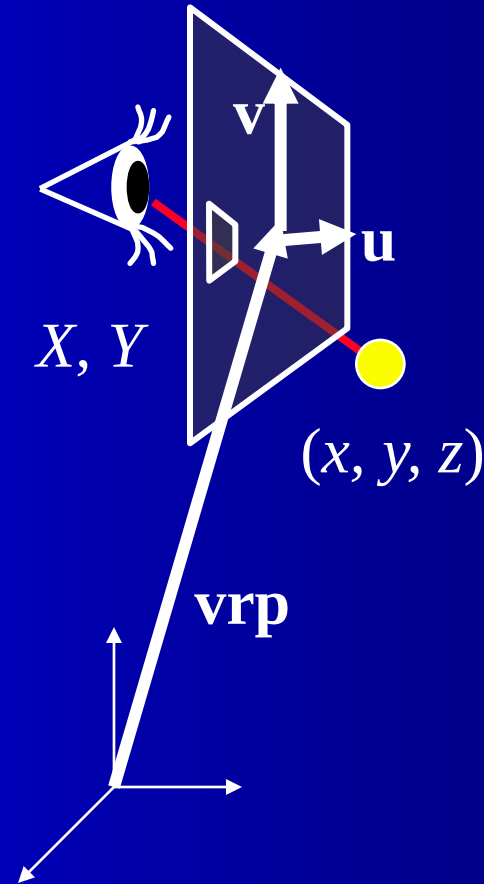


Blauskó Gabor

Kamera kalibráció



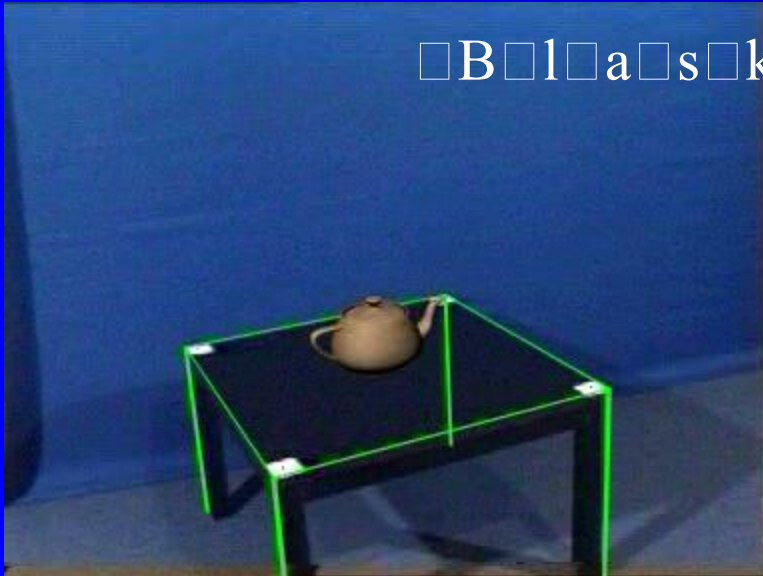
Ismeretlenek: $\mathbf{vrp}$, f , $\mathbf{u}$, $\mathbf{v}$
 $3 + 1 + 2 + 2 = 8$



Kompozitálás



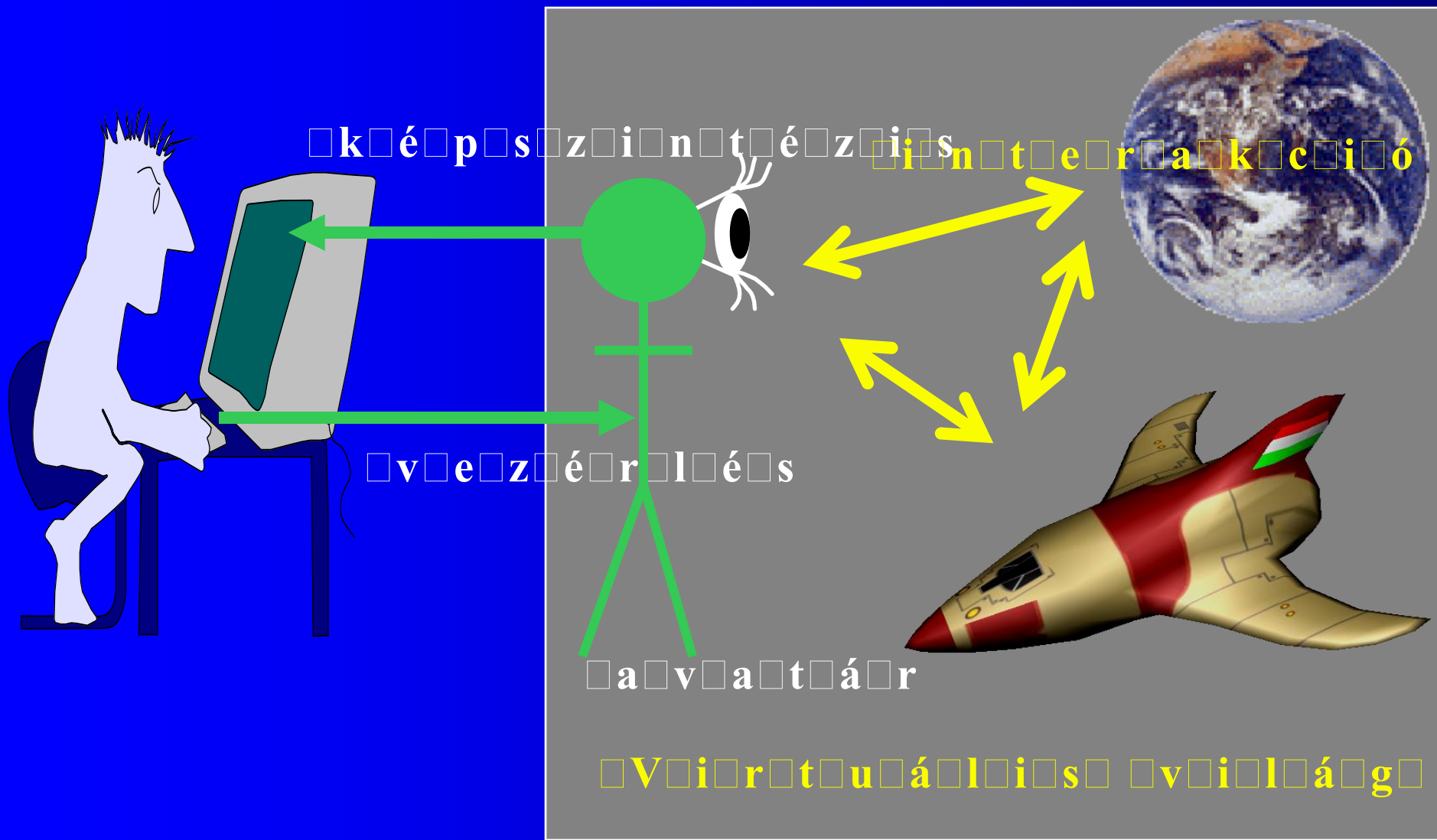
Blasó Gabor



Játékfejlesztés

Szirmay-Kalos László

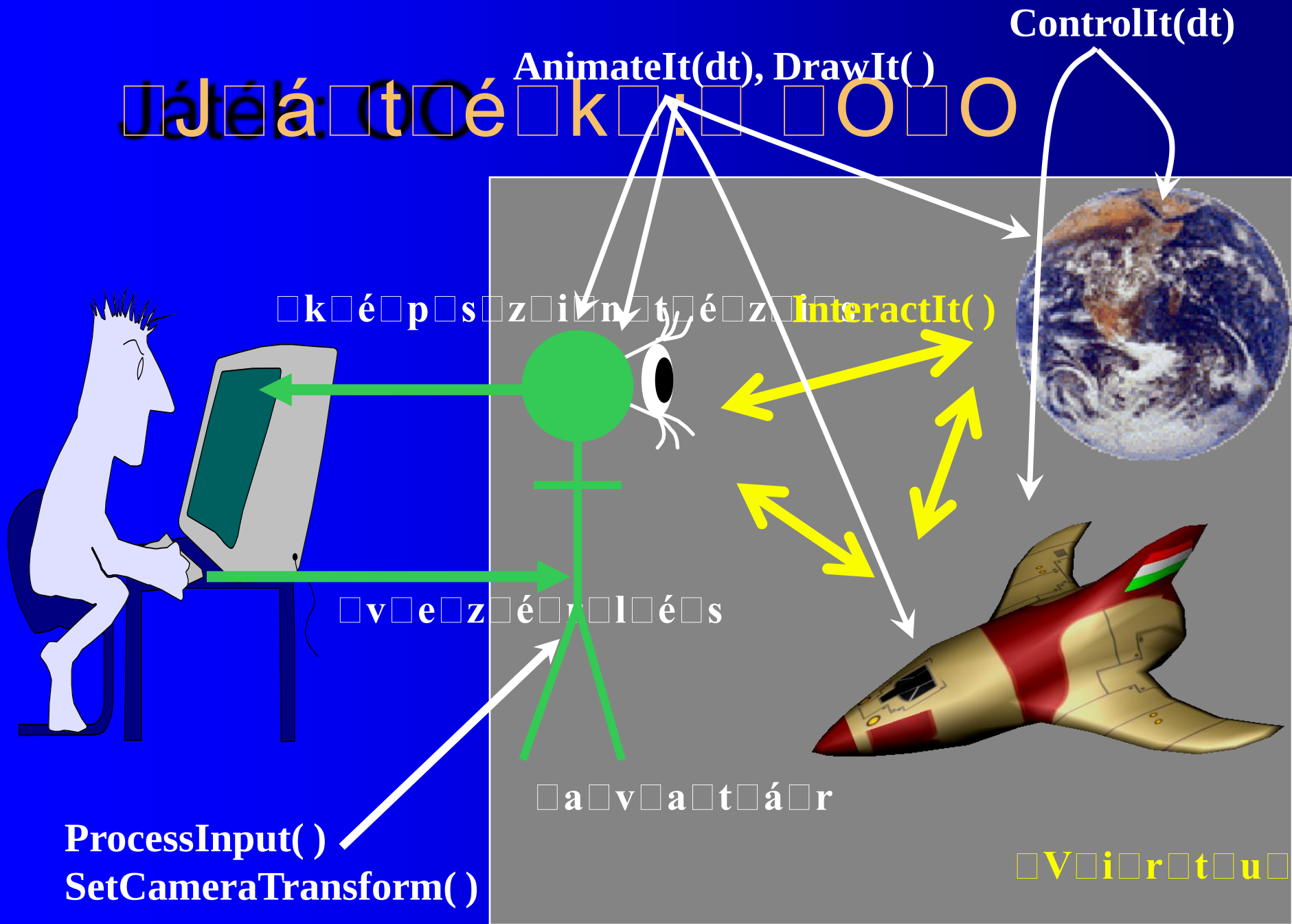
Virtuális világ



Játékok feladatai

- Képzésintézés az
- Az átvátlásról
(keyboard, mouse, Wii,
- Az „intelligens”
velérlés (AI)
- A fizikai világs

Játék: O O



Játékobjektum: GameObject t

●ControlIt():

- Fizteletben tartja a virtuális világot
- „gondolkodik” és a
- alkalmazza (pl. a

●InteractIt():

- Tájekozódik mások

●AnimateIt():

- Mozog

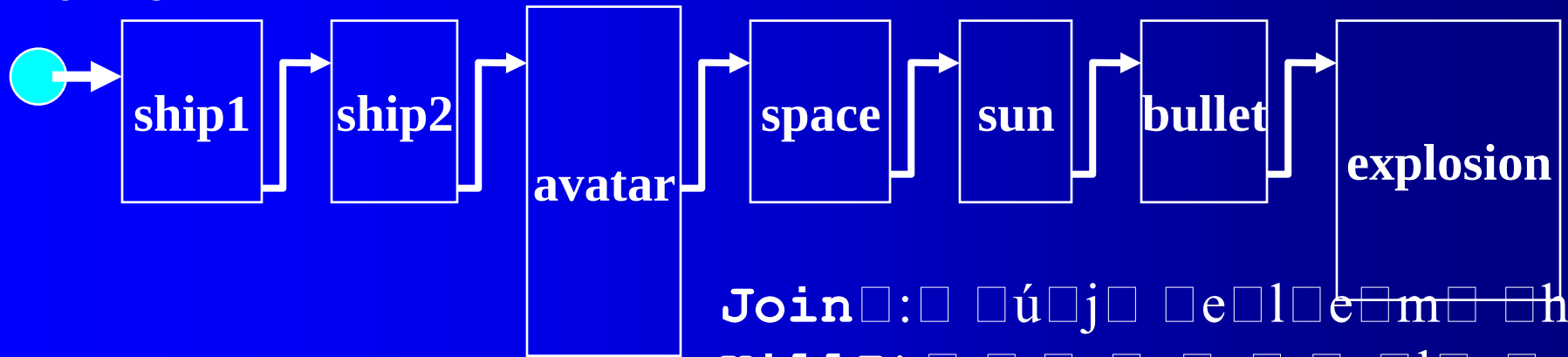
●DrawIt():

- Lefényképez a

A Virtual World (Scene Graph) v

- Objeektumok dinamiku
- Hetérogén kollekció
- Láncolat lista (fák)

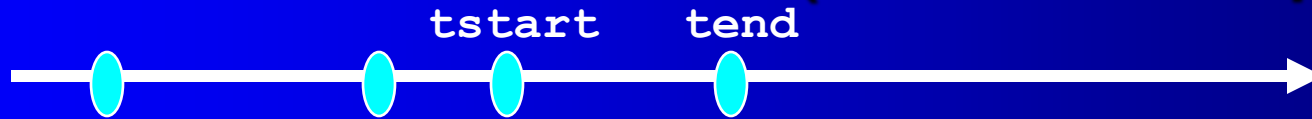
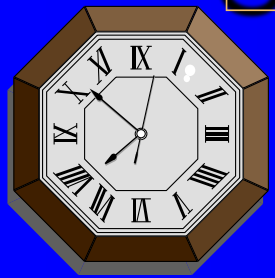
world



Control()
Animate()
Draw()

} Iteráció a kollekció

Számítás (Game Loop)



```
float tend = 0;
```

```
void IdleFunc( ) {           // idle call back
    float tstart = tend;
    tend = glutGet(GLUT_ELAPSED_TIME);

    ← avatar -> ProcessInput( );
    for(float t = tstart; t < tend; t += dt) {
        float Dt = min(dt, tend - t);
        for each obj of world: obj->ControlIt(Dt);
        for each obj of world: obj->AnimateIt(Dt);
    }
    glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
    avatar->SetCameraTransform();
    for each obj of world: obj->DrawIt();
    glutSwapBuffers( );
}
```



Bolygó: Planetó

● Geometria:

● Textúra

● Fizikai vagy

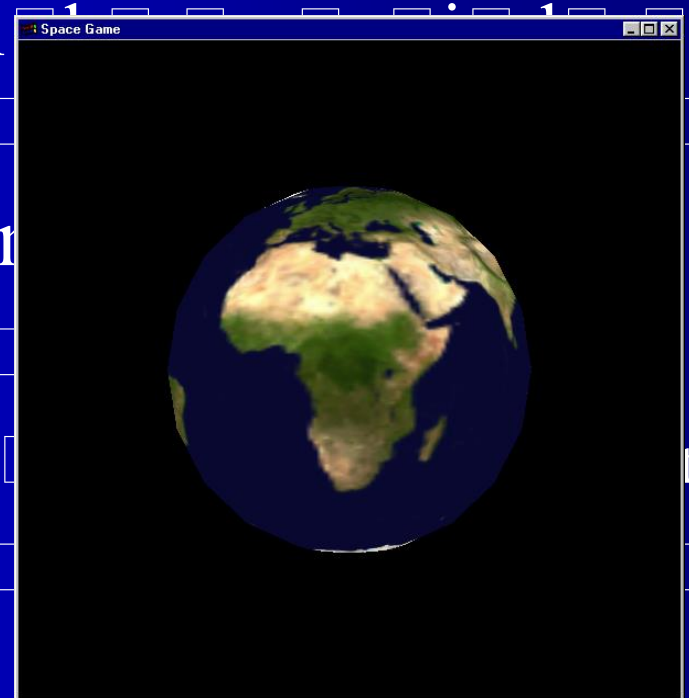
– Tájközöldi
gravitáció

● Képletai

– „beégettet

– Többiek

– Nincs



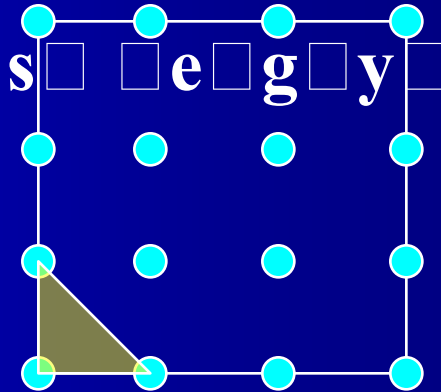
Gömbfelület felosztása

1. Paraméterezés

$$x = x_0 + r \cos 2\pi u \sin \varphi$$

$$y = y_0 + r \sin 2\pi u \sin \varphi$$

$$z = z_0 + r \cos \varphi \quad u, v \in [0, 1]$$



Paraméterezés használom

2. GLU kvadratikus felület:

// definíció

```
GLUquadricObj * quadric = gluNewQuadric( );  
gluQuadricTexture(quadric, GL_TRUE);
```

...

// Rájömlás

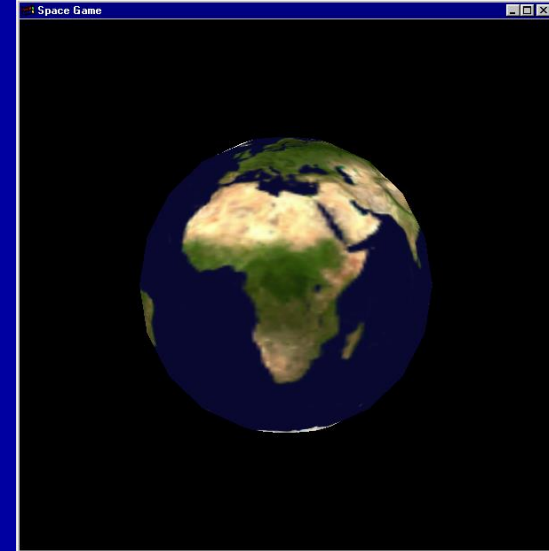
```
gluSphere(quadric, R, 16, 10);
```

Planet: Animate, Draw

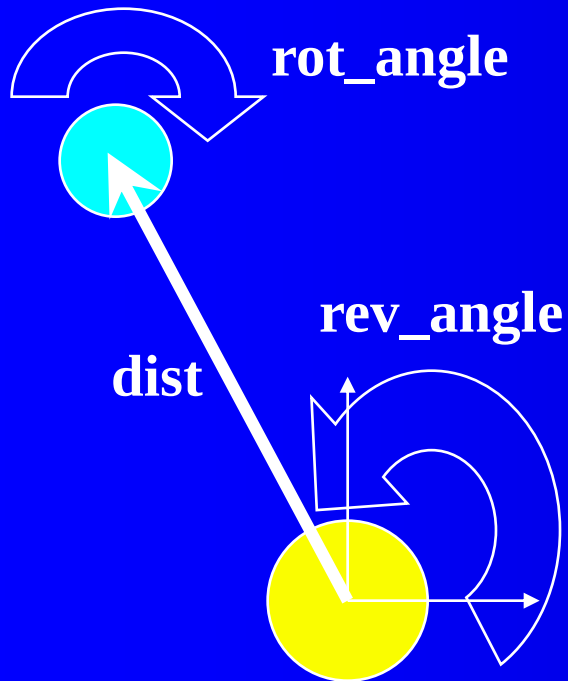
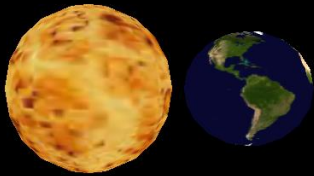
```
class Planet : public GameObject {
    float rot_angle, rot_speed, radius;
    unsigned int texture_id;
    GLUquadricObj * quadric;
public:
    Planet( int ... ) { ... }
    void ControlIt( float dt ) { }
    void InteractIt( GameObject * o ) { }

    void AnimateIt( float dt ) { rot_angle += rot_speed * dt; }

    void DrawIt( ) {
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture_id);
        glPushMatrix( );
            glRotatef( rot_angle, 0, 0, 1 );
            gluSphere( quadric, radius, 16, 10 );
        glPopMatrix( );
        glDisable(GL_TEXTURE_2D);
    }
};
```



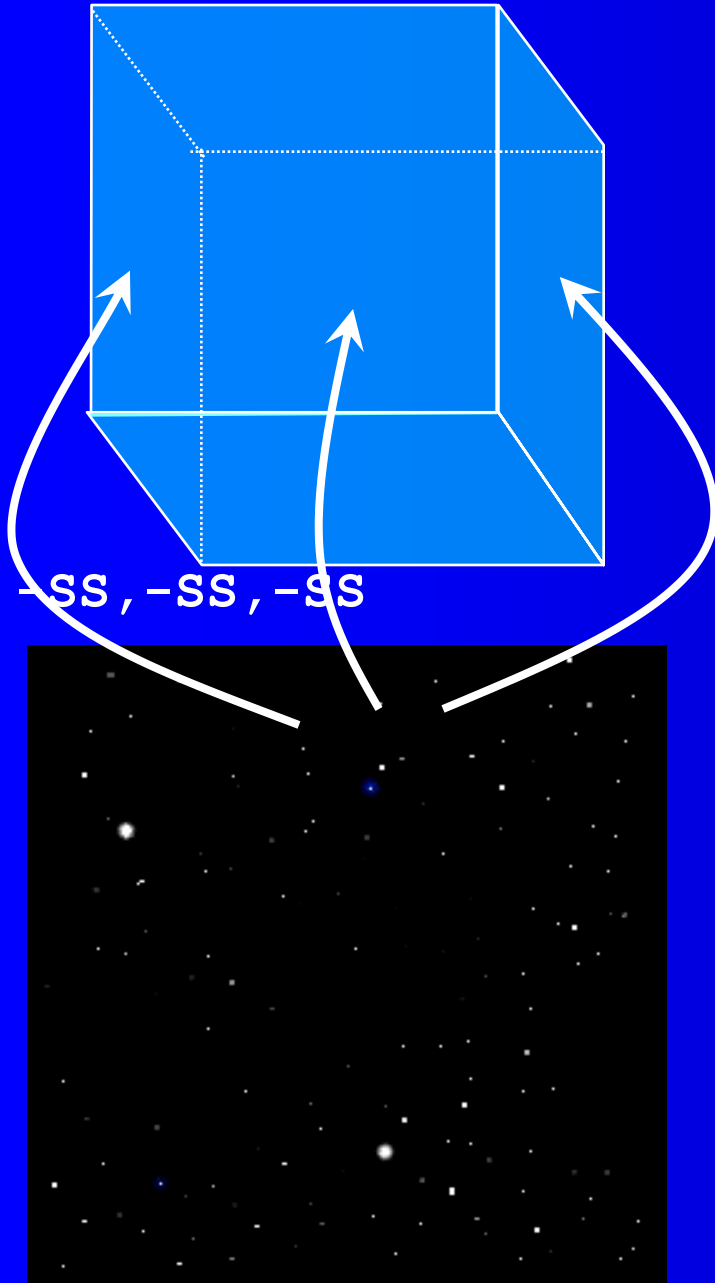
A Föld keringésének körülményei



```
void Planet :: AnimateIt(float dt) {  
    rot_angle += rot_speed * dt;  
    rev_angle += rev_speed * dt;  
}
```

```
void Planet :: DrawIt( ) {  
    glBindTexture(GL_TEXTURE_2D,  
                  texture_id);  
  
    glPushMatrix( );  
    glRotatef(rev_angle, 0, 0, 1);  
    glTranslatef(dist, 0, 0 );  
    glRotatef(rot_angle, 0, 0, 1);  
    gluSphere(quadric, 1, 16, 10);  
    glPopMatrix( );  
}
```

SS, SS, SS

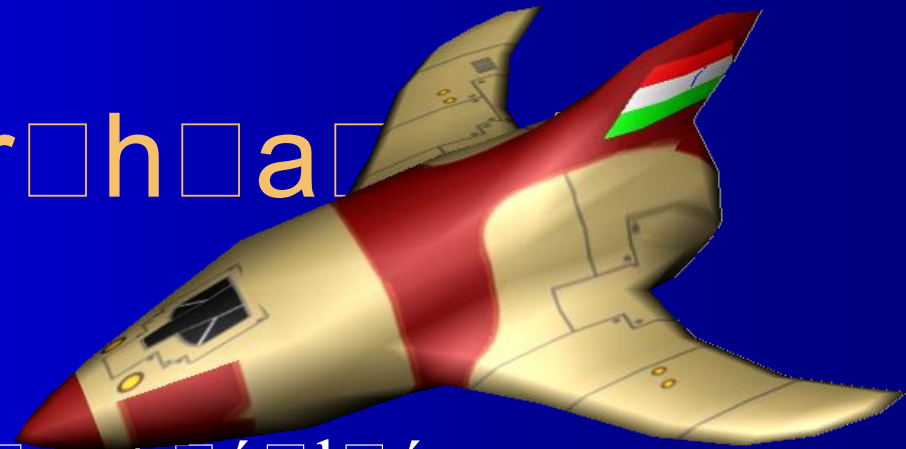


Az

Cr

```
void Space :: DrawIt( ) {  
    glBindTexture(GL_TEXTURE_2D,  
                  space_texture);  
  
    glBegin(GL_QUADS);  
    glTexCoord2f(0, 0);  
    glVertex3i(-SS, -SS, -SS);  
  
    glTexCoord2f(0, 1);  
    glVertex3i(-SS,  SS, -SS);  
  
    glTexCoord2f(1, 1);  
    glVertex3i( SS,  SS, -SS);  
  
    glTexCoord2f(1, 0);  
    glVertex3i( SS, -SS, -SS);  
    ...  
    glEnd();  
}
```

Az űrhajó Cserhája



● Komplex geometria

– négy szög három

● Komplex és tetszőleges

● Fizikai áncolás

– erők ($\vec{g}$ gravitáció, $\vec{v}$

– ütközések

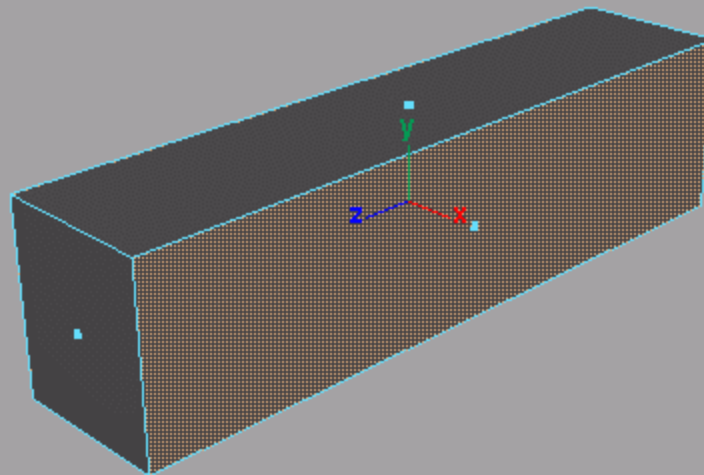
● Viseelkedés (AI)

– A rakéták vezérlés

● Ütközések elkerülése, a

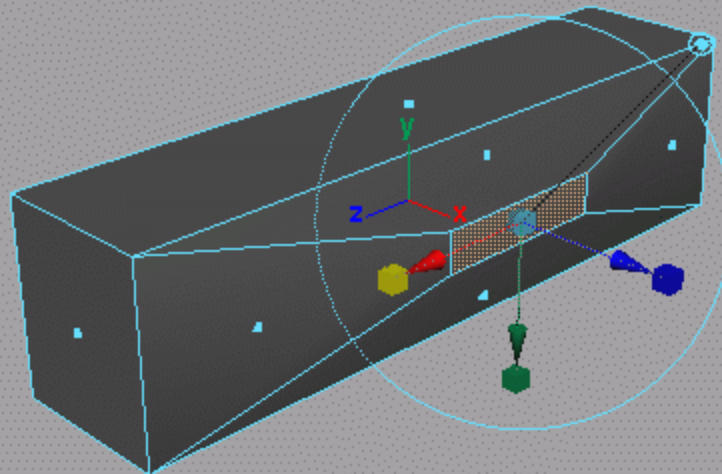
Úrajó geometria

Brachajó



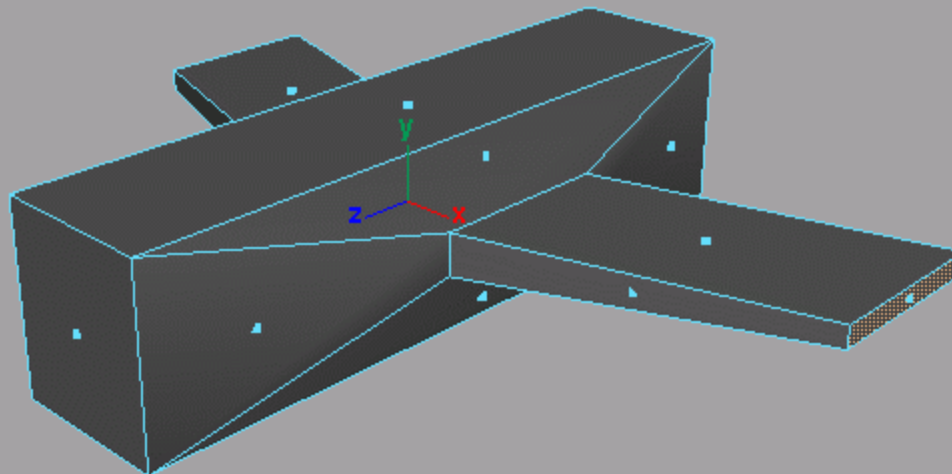
persp

Polygon Modellözés: 1. extruding



persp

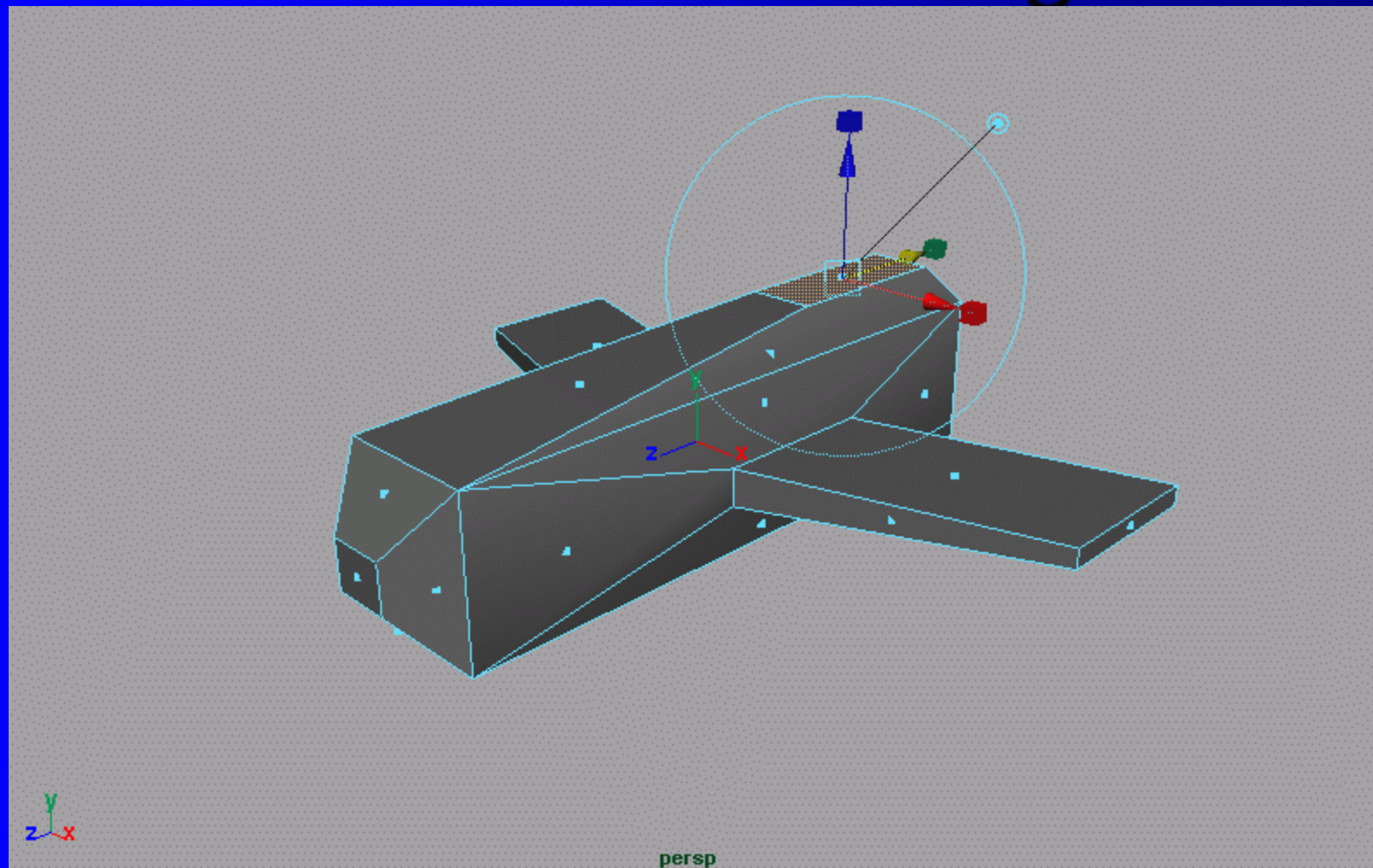
Polygon Modellözés: 2. extruding



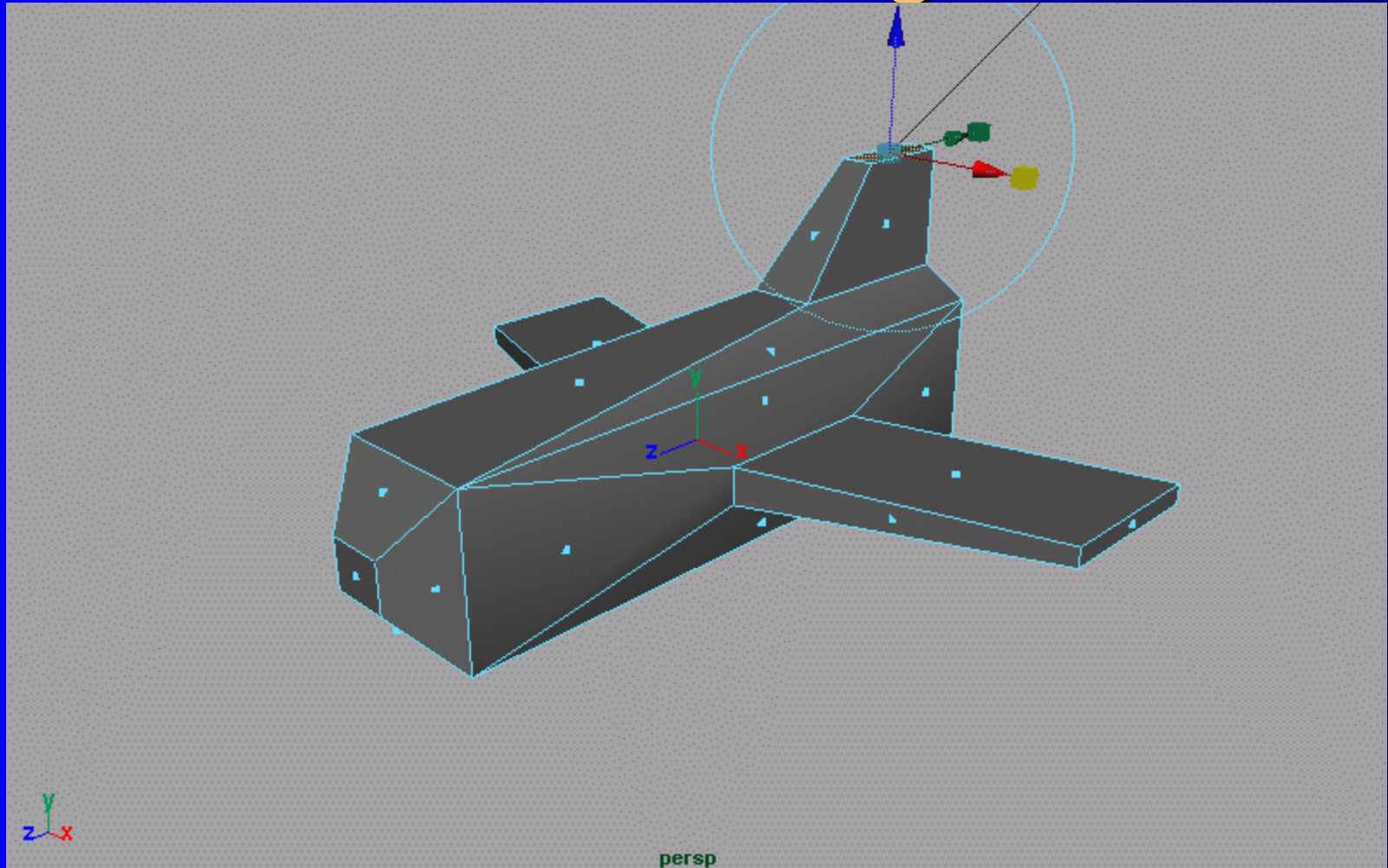
persp

Polygon Modellálás: 4. és 5. lépés

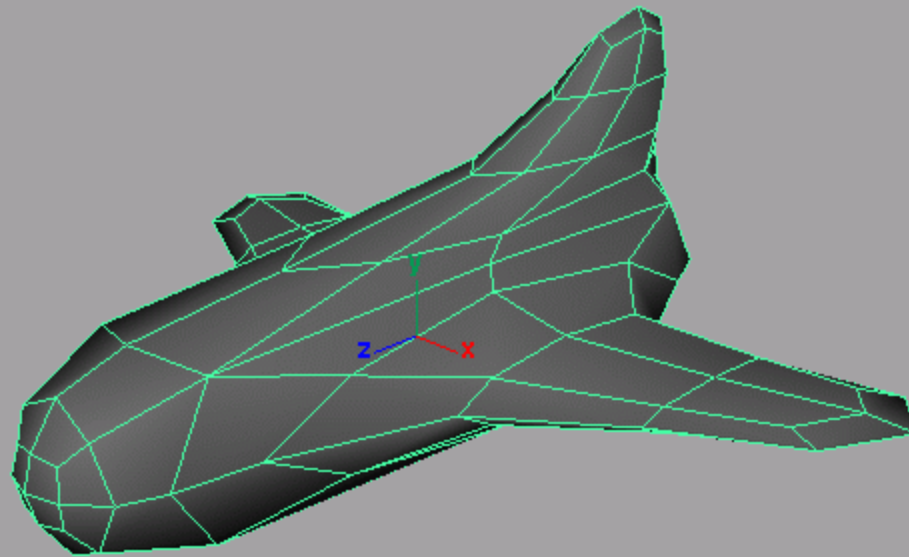
4. és 5. lépés



Polygonmodellelés: 6. extruding

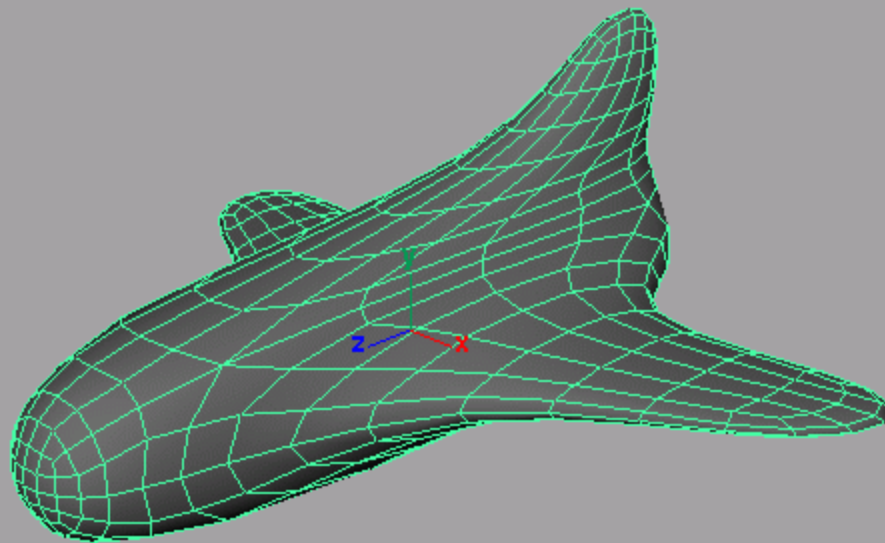


Subdivision szint



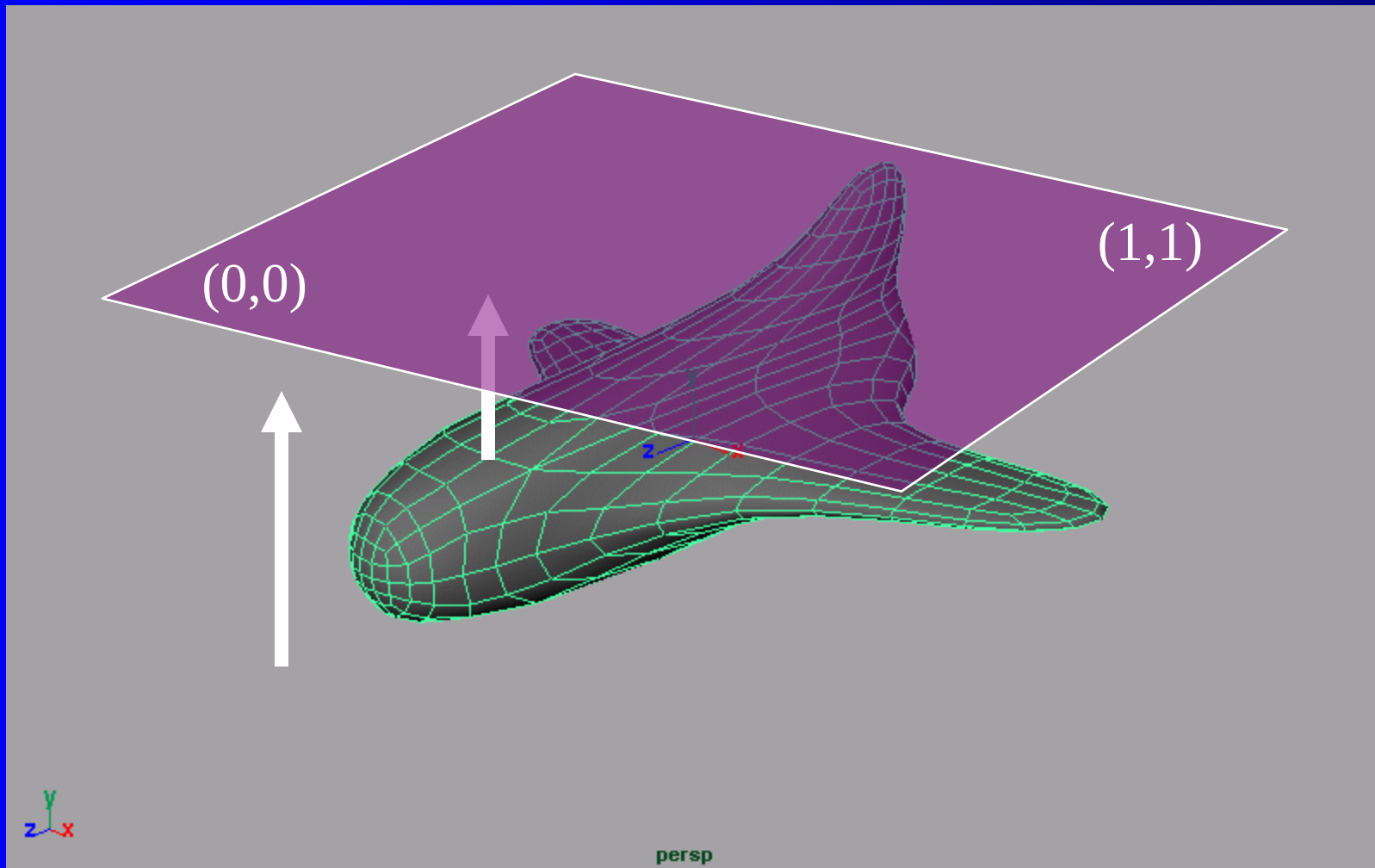
persp

Subdivision

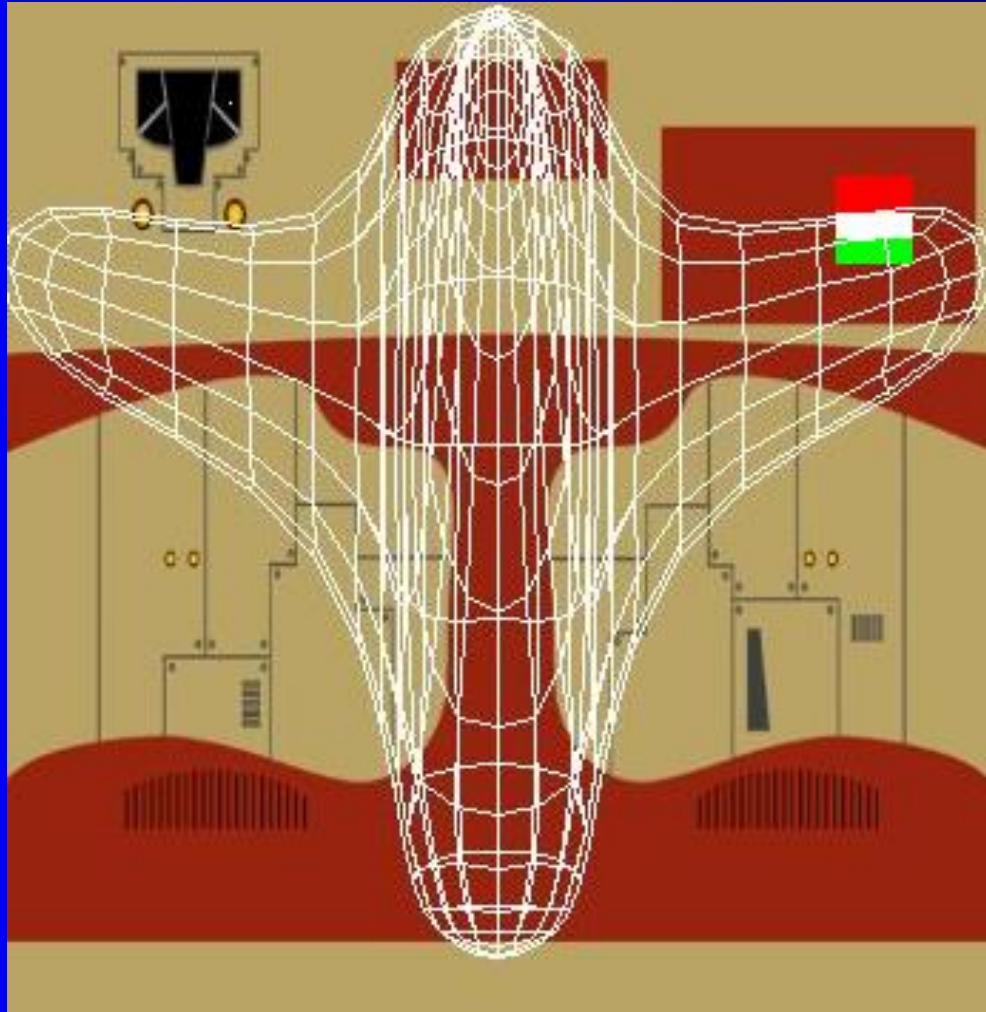


persp

Textúrafüggvény definíció



Textúrafüggvény definíció



Textúrázott űrhajó



Spaceship OBJ format

```
v -0.708698 -0.679666 2.277417
v 0.708698 -0.679666 2.277417
v -0.735419 0.754681 2.256846
```

...

```
vt 0.510655 0.078673
vt 0.509594 0.070000
vt 0.496429 0.079059
```

...

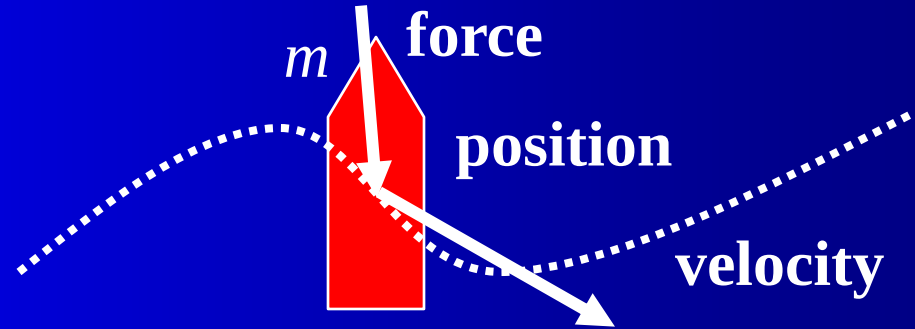
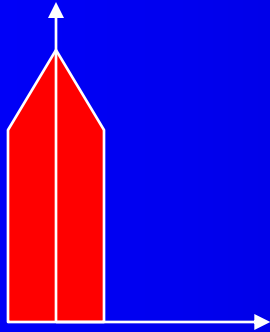
```
vn -0.843091 0.000000 0.537771
vn -0.670151 -0.543088 0.505918
vn -0.000000 -0.783747 0.621081
```

□...

```
f 65/1/1 37/2/2 62/3/3 61/4/4
f 70/8/5 45/217/6 67/218/7 66/241/8
f 75/9/9 57/10/10 72/11/11 71/12/12
```

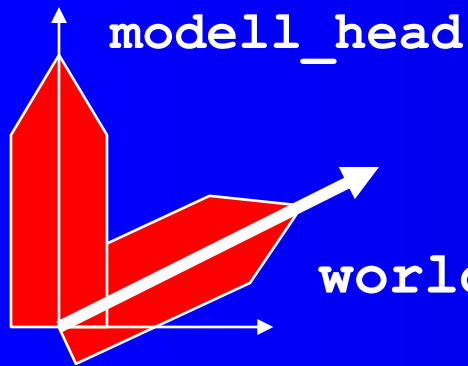
...

Animate it: Newton's



```
void Ship :: AnimateIt( float dt ) {  
    acceleration = force/m;  
    velocity += acceleration * dt;  
    position += velocity * dt;  
}
```

```
void Ship :: DrawIt( ) {  
    glPushMatrix( );  
    glTranslatef(position.x, position.y, position.z);  
    glBegin( GL_QUADS ); ... ; glEnd( );  
    glPopMatrix( );  
}
```



□Orientációbeállítás□

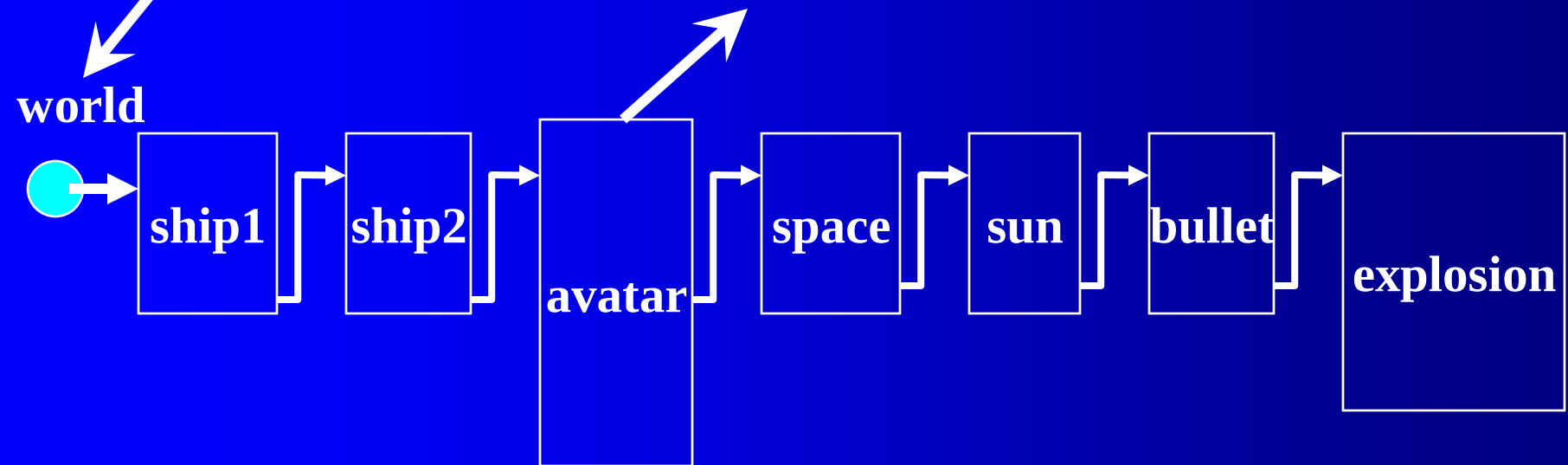
```
world_head = velocity.UnitVector();
```

```
void Ship :: DrawIt() {  
    glPushMatrix( );  
    glTranslatef(position.x, position.y, position.z);  
  
    Vector modell_head( 0, 0, 1 );  
    Vector world_head = velocity.UnitVector();  
    Vector rotate_axis = modell_head % world_head;  
  
    float cos_rotate_angle = world_head * modell_head;  
    glRotatef( acos(cos_rotate_angle)* 180 / M_PI,  
              rotate_axis.x,rotate_axis.y,rotate_axis.z);  
    glBegin( GL_QUADS ); ... ; glEnd( );  
    glPopMatrix( );  
}
```

Ship :: ControllIt

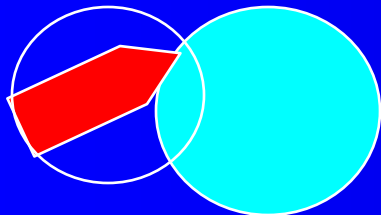
```
void Ship :: ControllIt( float dt ) {  
    force = Vector(0, 0, 0);  
    Interact( world );  
}
```

```
void Ship::InteractIt( GameObject * object )
```

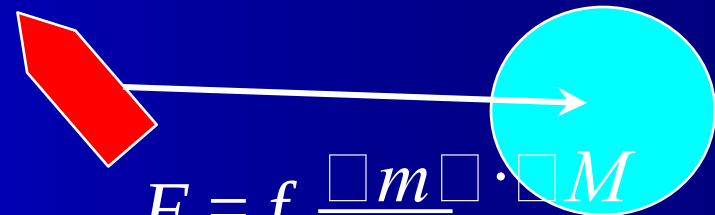
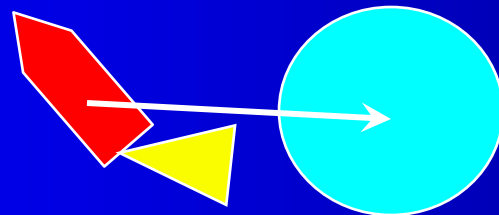


Ship: InteractIt

```
void Ship :: InteractIt( GameObject * object ) {  
    if ( object->GetType( ) == PLANET ) {
```



```
}
```

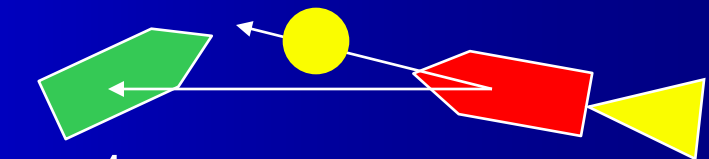


$$F = f \frac{m \cdot M}{r^2}$$

```
    if ( object->GetType( ) == AVATAR ) {  
        bullet
```



avatar



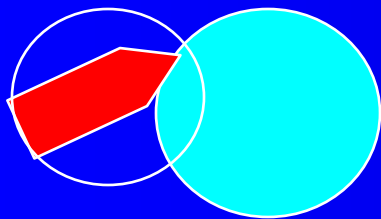
avatar

aiming angle

```
}
```

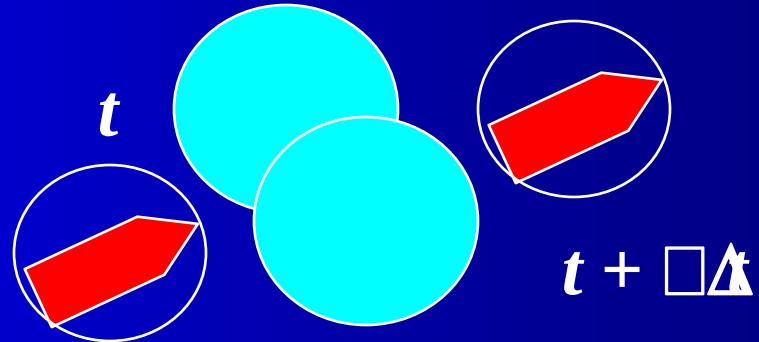
```
}
```

Ütközékdéktálás és időtől függő objektumok közötti



adott t

Probléma, ha



```
dist = obj1.position - obj2.position  
min = obj1.BoundingRadius() + obj2.BoundingRadius()  
if (dist.Length() < min) Collision!
```

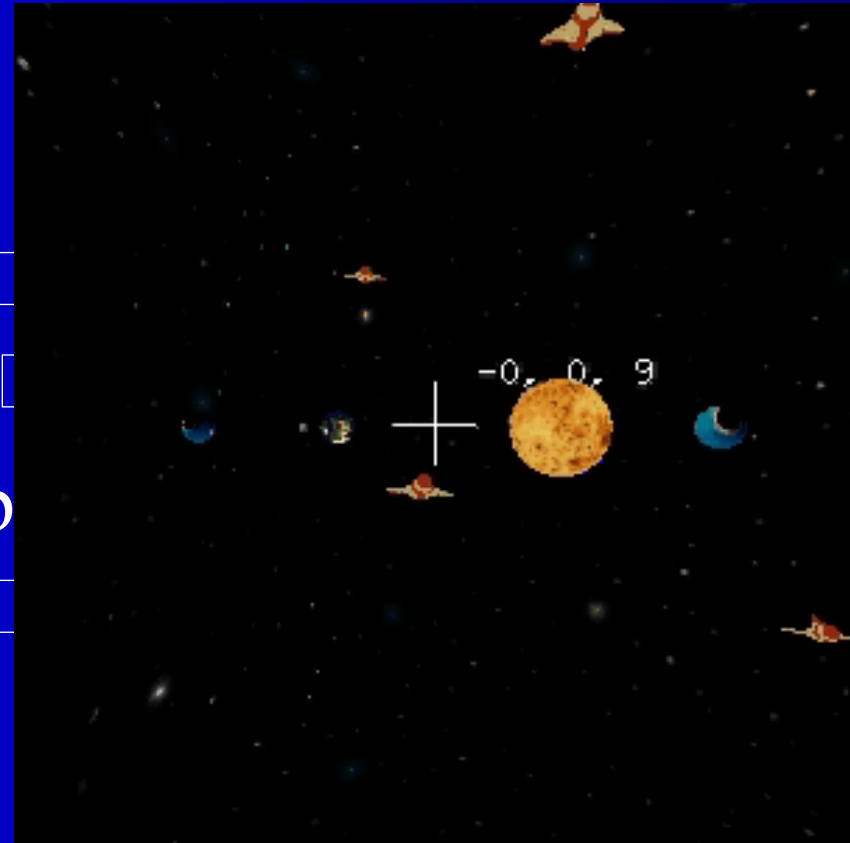
Lövedék

● Nagyon komplex geometria

● Haszonló mindezen

● Könnyebb használni

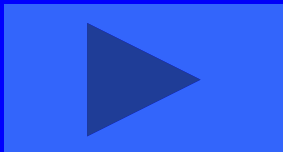
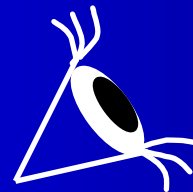
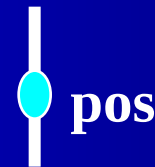
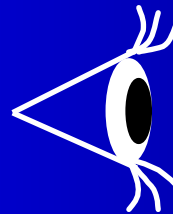
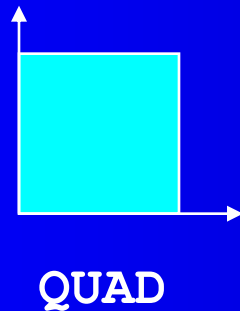
átlat szó



● Ütközésetek általás

Paklák: Billboard

Egyetlen félig átl



```
void Bullet :: DrawIt() {
```

```
    glPushMatrix( );
```

```
    Vector w = eye - position;
```

```
    Vector r = w % Vector(0, 1, 0);
```

```
    Vector u = r % w;
```

```
    r = r.UnitVector() * size;
```

```
    u = u.UnitVector() * size;
```

```
    glTranslatef(position.x, position.y, position.z);
```

```
    glEnable(GL_BLEND);           □/□/□ □á□t□l□á□t□s□z□ó□s□á□g
```

```
    glBlendFunc(GL_SRC_ALPHA, GL_ONE); // □h□o□z□z□á□a□d□á□s
```

```
    glBindTexture(GL_TEXTURE_2D, bullet_texture);
```

```
    glBegin(GL_QUADS);
```

```
    glTexCoord2f(0,0); glVertex3f(-r.x-u.x, -r.y-u.y, -r.z-u.z);
```

```
    glTexCoord2f(1,0); glVertex3f( r.x-u.x,  r.y-u.y,  r.z-u.z);
```

```
    glTexCoord2f(1,1); glVertex3f( r.x+u.x,  r.y+u.y,  r.z+u.z);
```

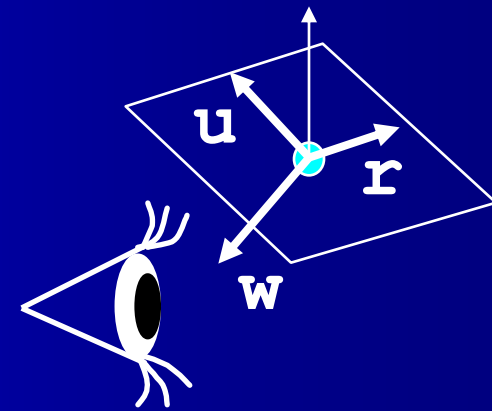
```
    glTexCoord2f(0,1); glVertex3f(-r.x+u.x, -r.y+u.y, -r.z+u.z);
```

```
    glEnd();
```

```
    glDisable(GL_BLEND);
```

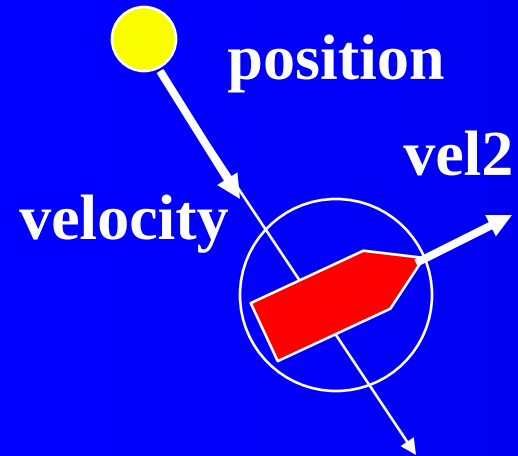
```
    glPopMatrix( );
```

```
}
```



Gyors ütközésdetektálók

ray-tracing



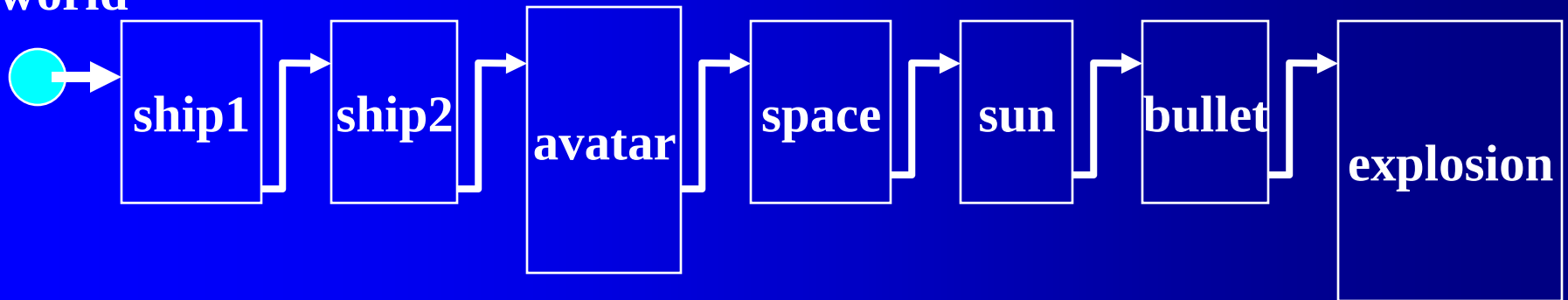
$$\text{rel_velocity} = \text{velocity} - \text{vel2}$$

$$\text{ray: rel_pos} + \text{rel_velocity} \cdot t$$

If (ray intersects bounding sphere first
AND $t_{\text{intersect}} < dt$) **Collision!**

```
hit_object = world->Intersect(position, velocity, t);
```

world



Robotbanás

- Nagyon komplex geometria

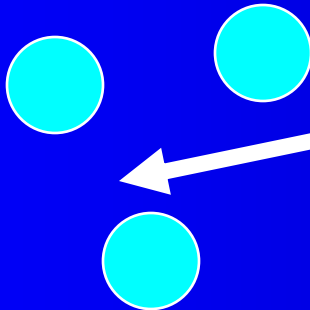
- Haszonló mindezen

- Plakátgy

- Részecsk



Részecskerendszer



Globális 5×10^6
(szél, fúj, j, a, a, f, ü)

Velocity

Kezdeti

érték

position:

velocity:

acceleration:

lifetime

age:

size, dsize:

weight, dweight:

color, dcolor:

$\text{position} += \text{velocity} * dt$

$\text{velocity} += \text{acceleration} * dt$

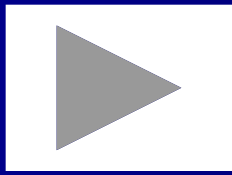
$\text{acceleration} = \text{force} / \text{weight}$

$\text{age} += dt$; if ($\text{age} > \text{lifetime}$) Kill();

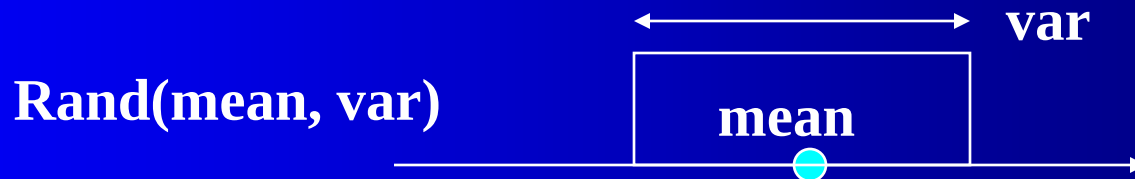
$\text{size} += \text{dsize} * dt$;

$\text{weight} += \text{dweight} * dt$

$\text{color} += \text{dcolor} * dt$



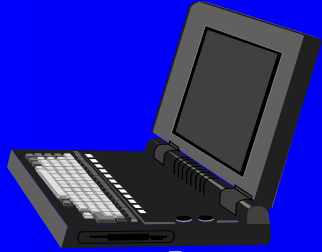
Róbbanás



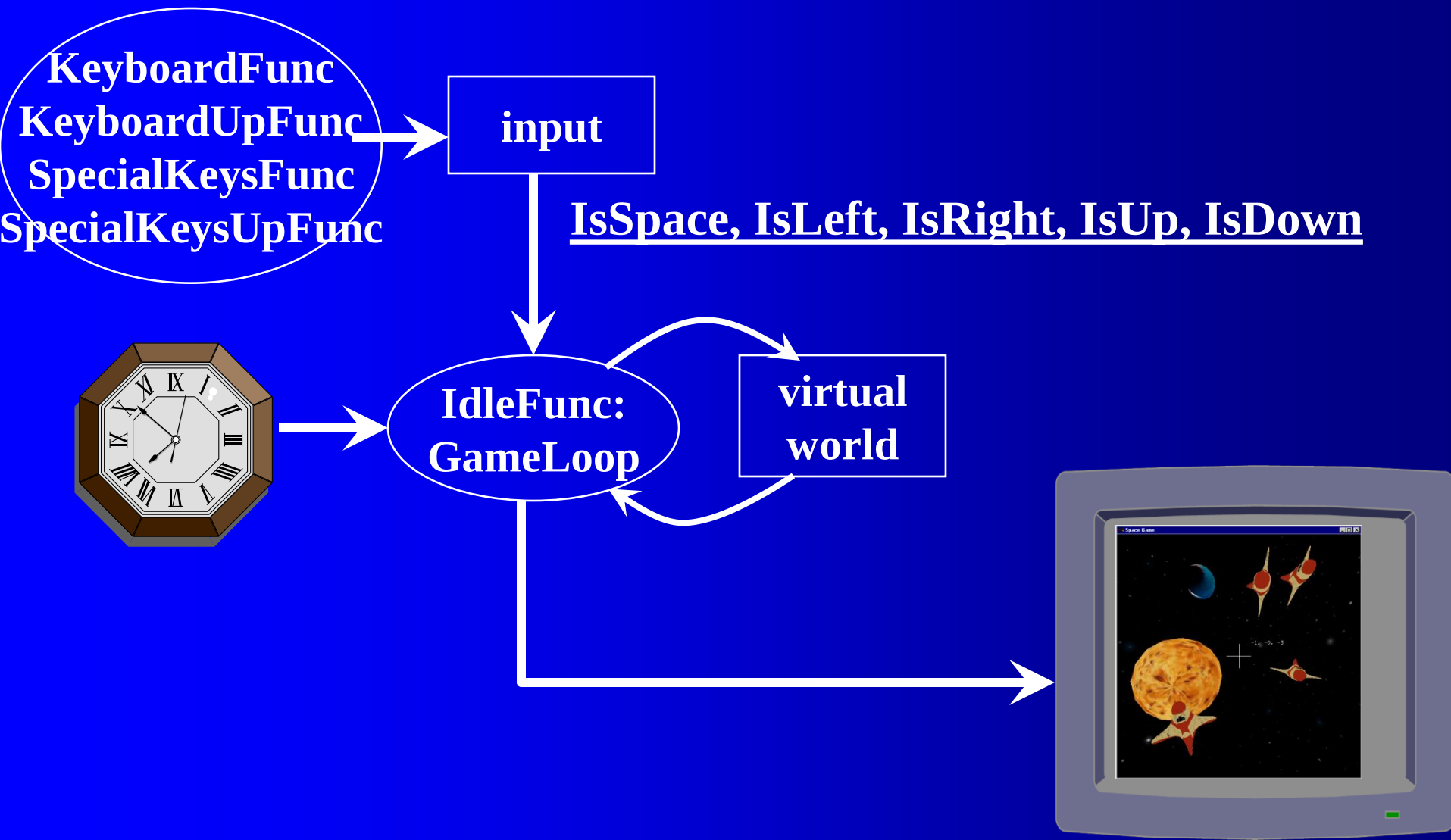
```
position = center; // kezdetben  
lifetime = Rand(2, 1);  
  
size = 0.001; // kezdetben kicsi  
dsize = Rand(0.5, 0.25) / lifetime;  
  
velocity = Vector(Rand(0,0.4), Rand(0,0.4), Rand(0,0.4));  
acceleration = Vector(Rand(0,1), Rand(0,1), Rand(0,1));  
  
// Plánck törvény: sárga  
color = Color(1, Rand(0.5, 0.25), 0, 1);  
dcolor = Color(0, -0.25, 0, -1) / lifetime;
```

Avatar

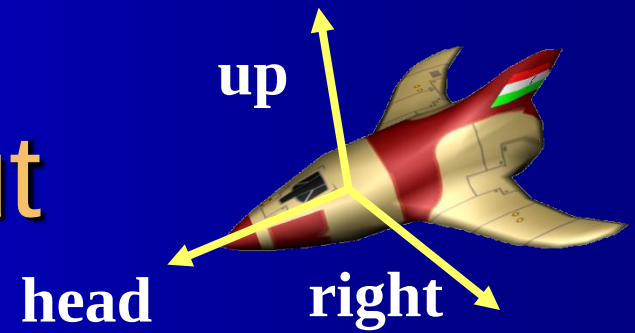
- Avatar `viselkedését` `ProcessInput`
- Avatar `helye` `és irány` `SetCameraTransform`
- `Ollyan` `mint egy C`
`≠ Controller: gravitáció`



Klaviatúra vezérlés



Avatar :: ProcessInput



```
Avatar :: ProcessInput( KeyStates * input ) {
```

```
    if ( input->IsSpace( ) ) //  $\Delta t! C \Delta z$   
        world -> Join(new Bullet(position, velocity));
```

```
    //  $Korm \Delta a \Delta n \Delta y \Delta z \Delta a \Delta s : \Delta a \Delta z \Delta a \Delta v \Delta a \Delta t \Delta a$ 
```

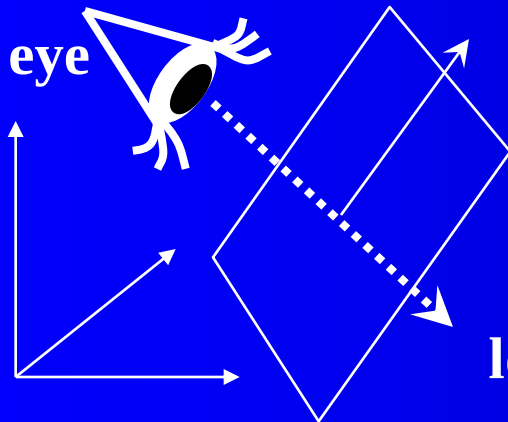
```
    Vector head = velocity.UnitVector( );
```

```
    if ( input->IsUp( ) )      force += up * (-1);  
    if ( input->IsDown( ) )    force += up;  
    if ( input->IsLeft( ) )    force += up % head;  
    if ( input->IsRight( ) )   force += head % up;
```

```
}
```

Avatar :: SetCameraTransform

```
Avatar :: SetCameraTransform( ) {  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
    gluLookAt(position.x, position.y, position.z,  
              position.x + head.x,  
              position.y + head.y,  
              position.z + head.z,  
              up.x, up.y, up.z);  
}
```

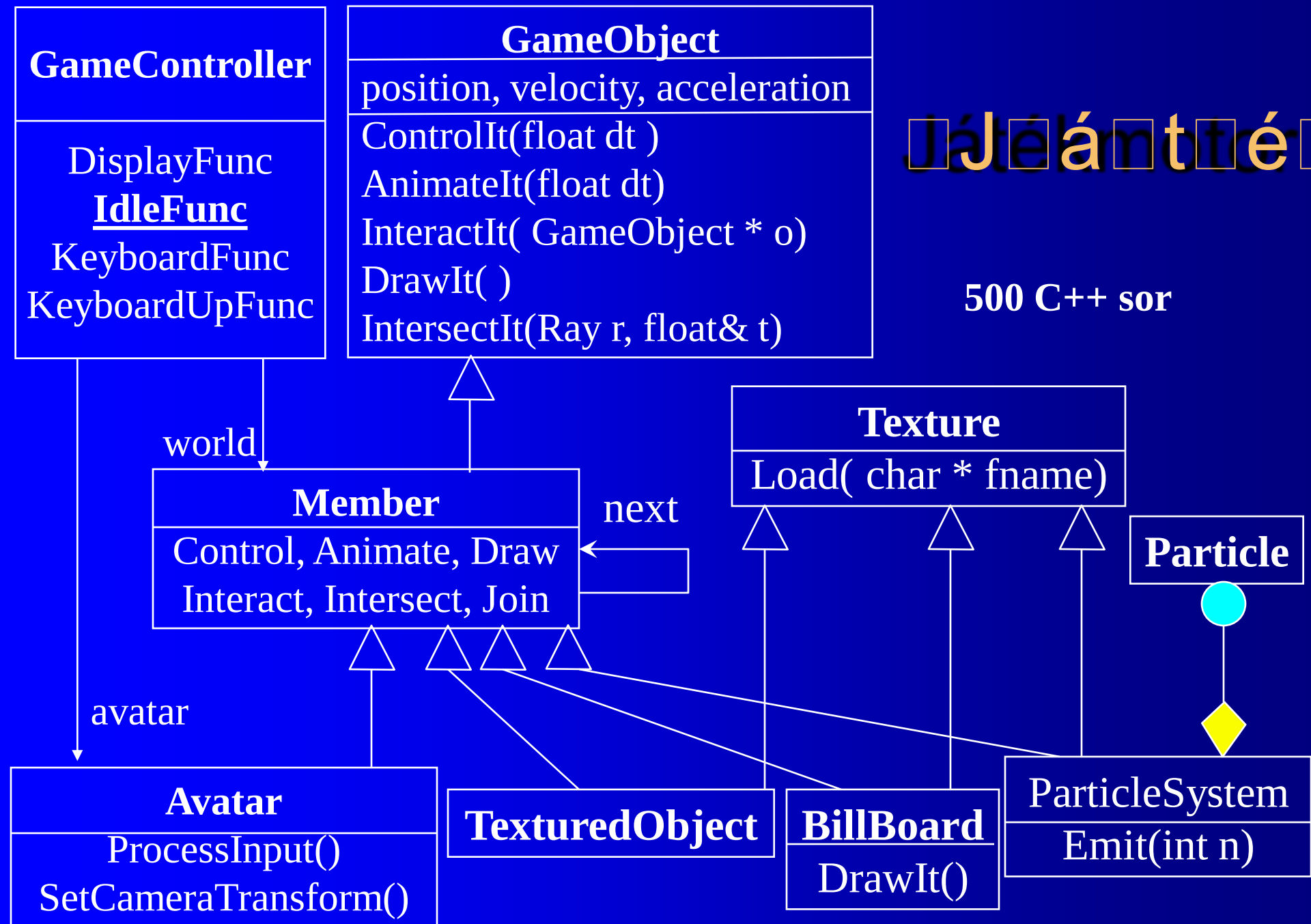


up = [0, 1, 0] vagy a

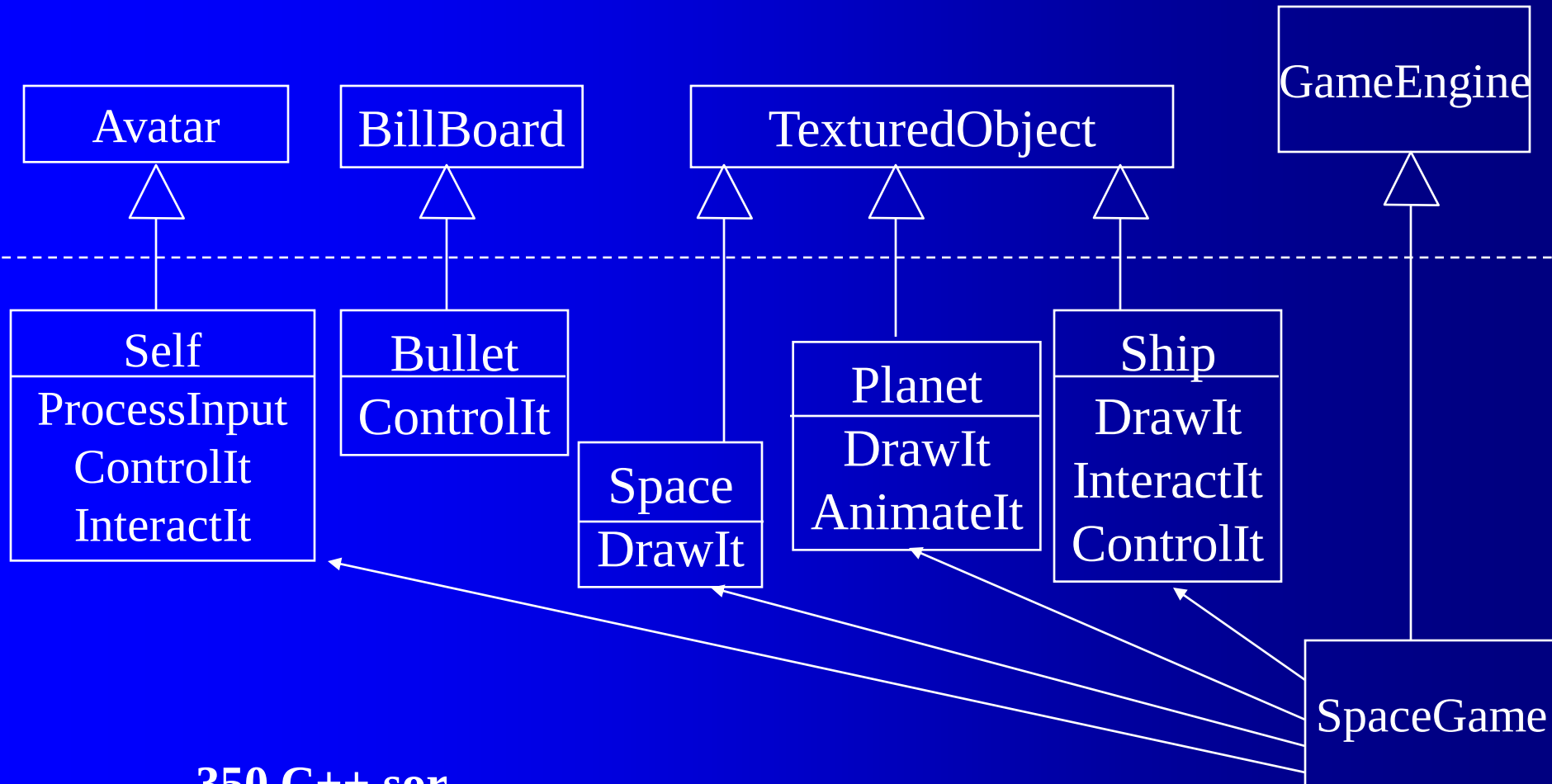
g y o r s u l á s b ó l é s

Játékmotor

500 C++ sor



Űrjáték Borjárték



350 C++ sor

Egy földlövészöld játék lő



Terepek



- Komplex geometria

 - magasság

- Boonyoölölt tectúr

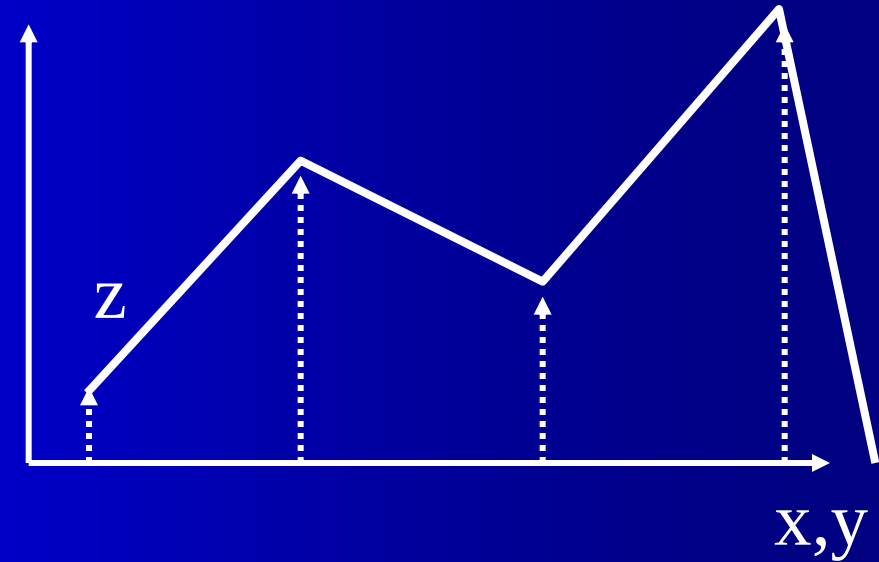
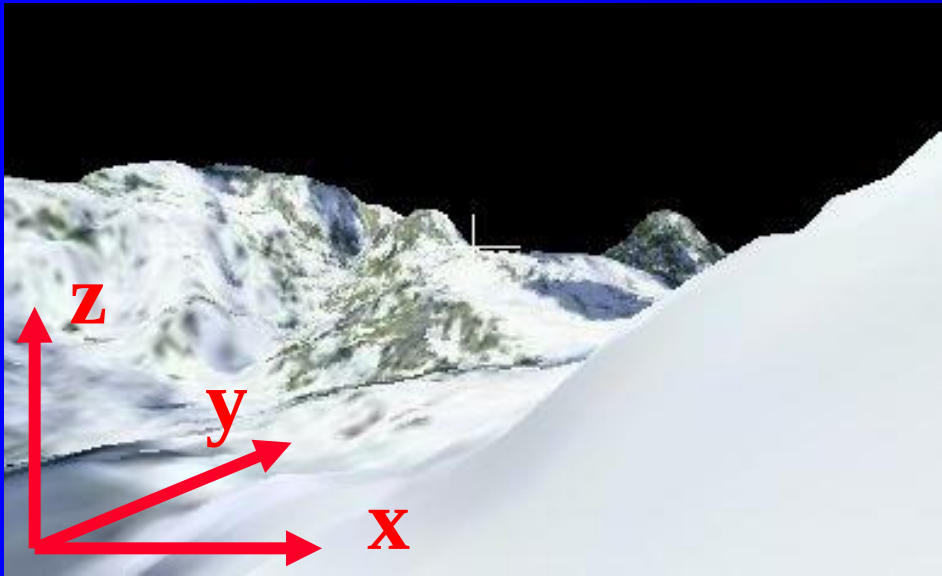
- Nem gondolkodik

- Nem mozog

- Ütközés detektál

- Megemeli az objektumokat

Terep geometria



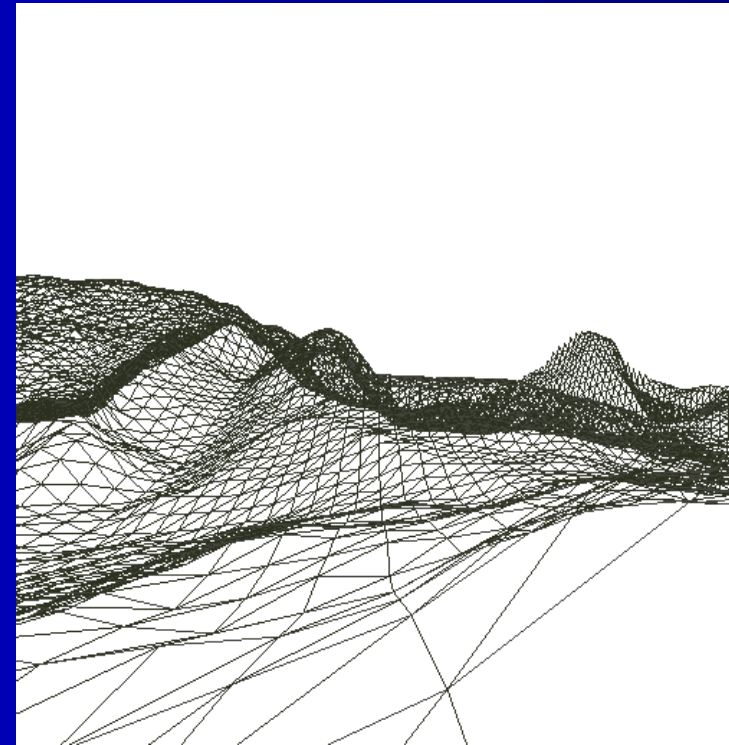
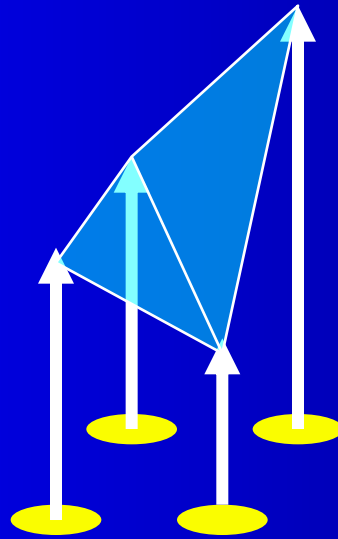
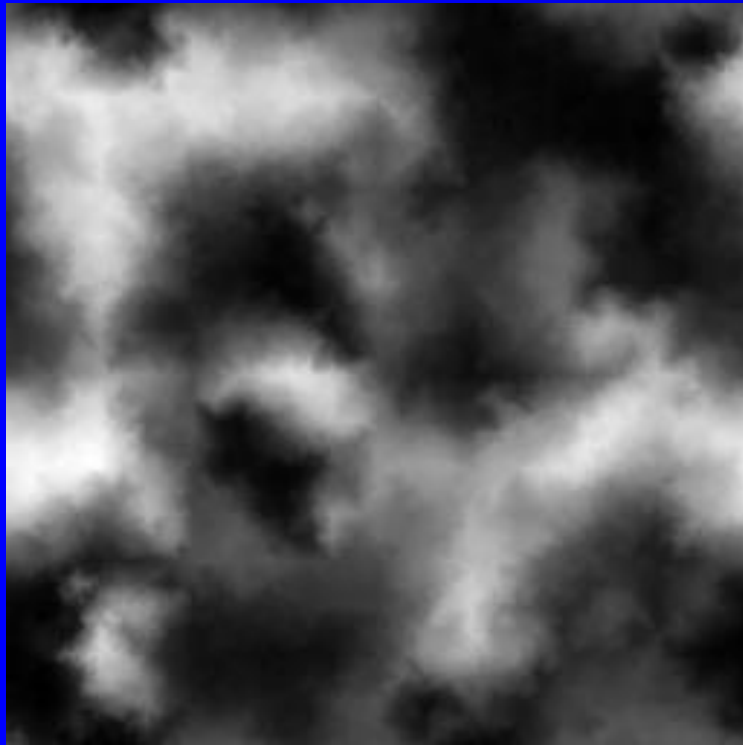
$$z = \text{height}(x,y)$$

□ M □ a □ g □ a □ s □ s □ á □ g □

Dis □ z □ k □ r □ é □ t □ □ m □ i

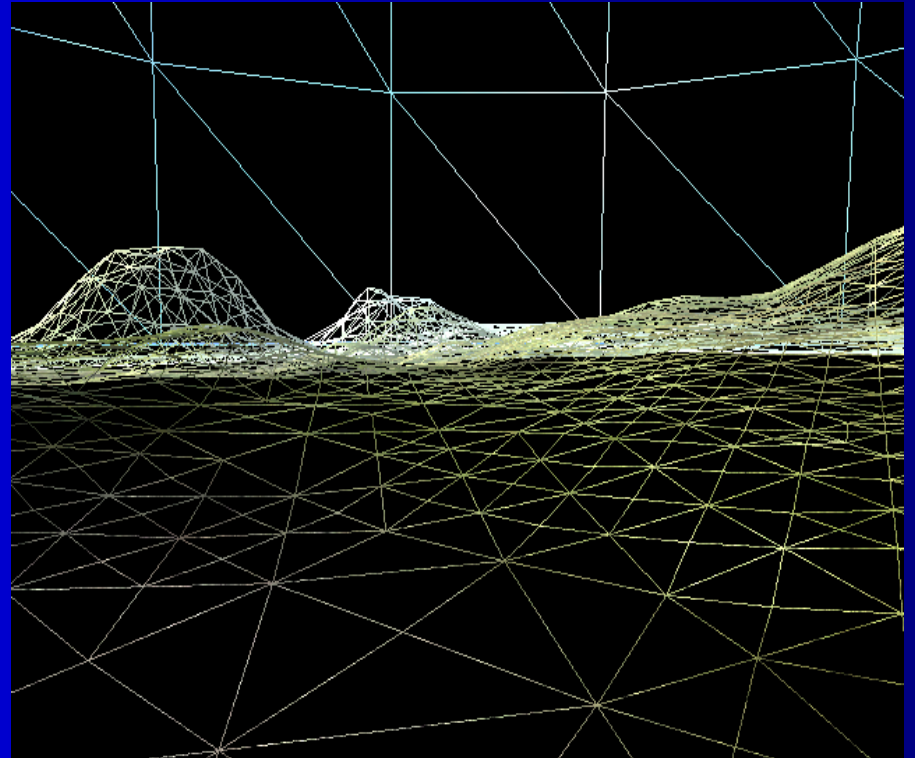
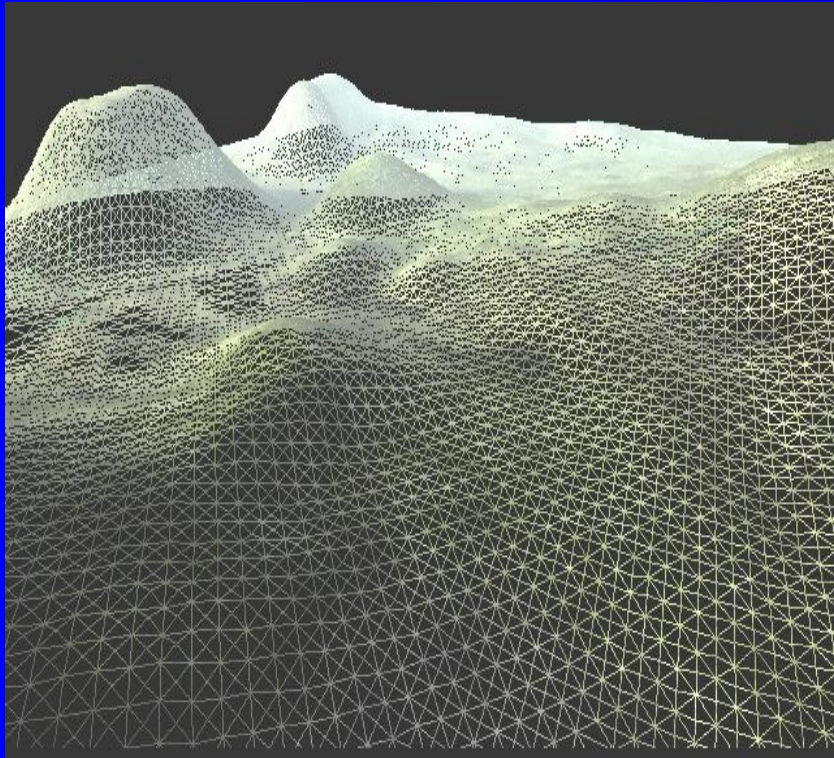
Line □ á □ r □ i □ s □ □ i □ n □ t

□ D □ i □ s □ z □ i □ n □ t □ á □ k □ r □ é □ f □ t □ e □ p □ m □ k □ i □



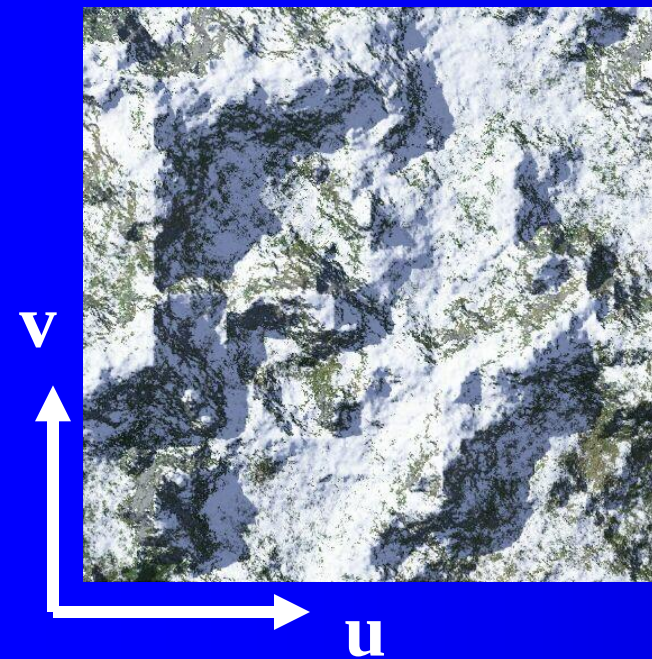
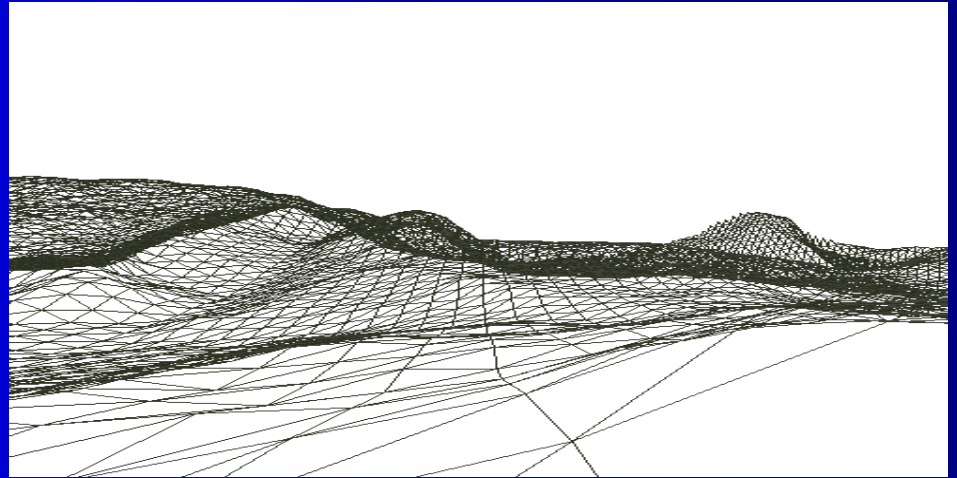
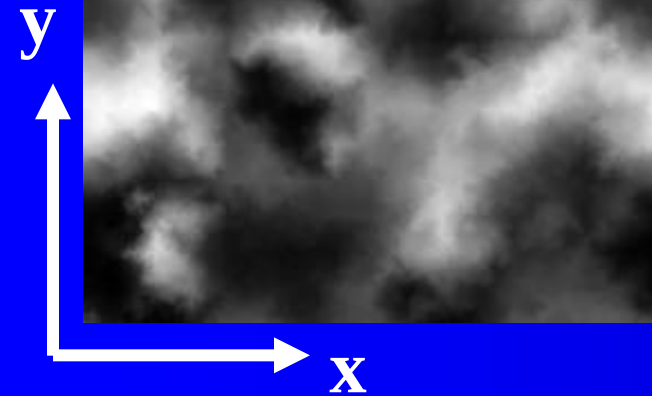
□ M □ a □ g □ a □ s □ s □ á □ g □ □ m □ e □ H □ á □ r □ o □ m □ s □

Háromszögmátrixok
csökkenése: Levegő



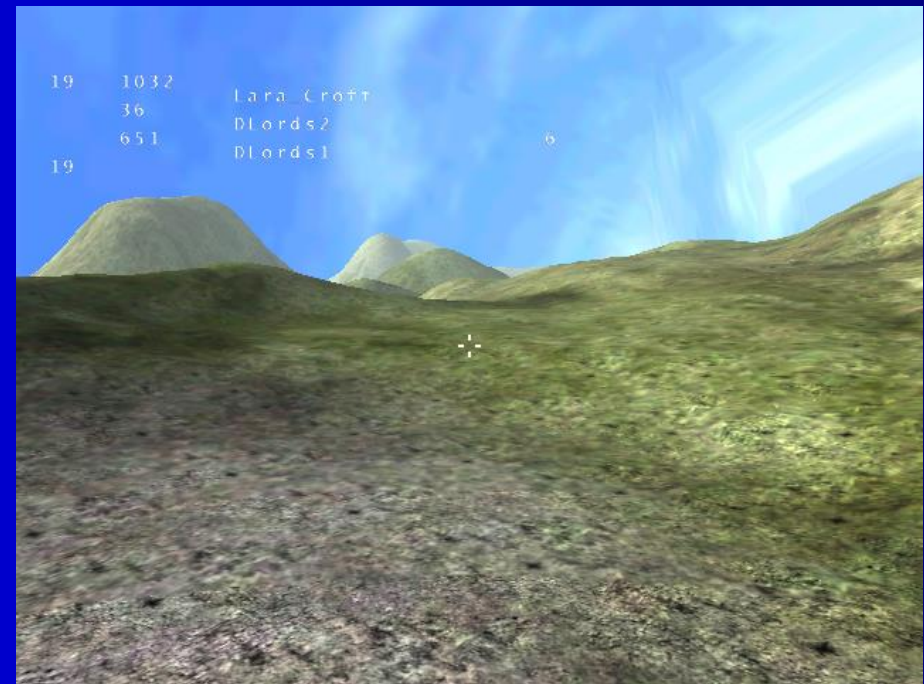
Magas-sá

textúra: úv-reá-tí-á



Tereptextúrajavítás: xú

Multitextúra

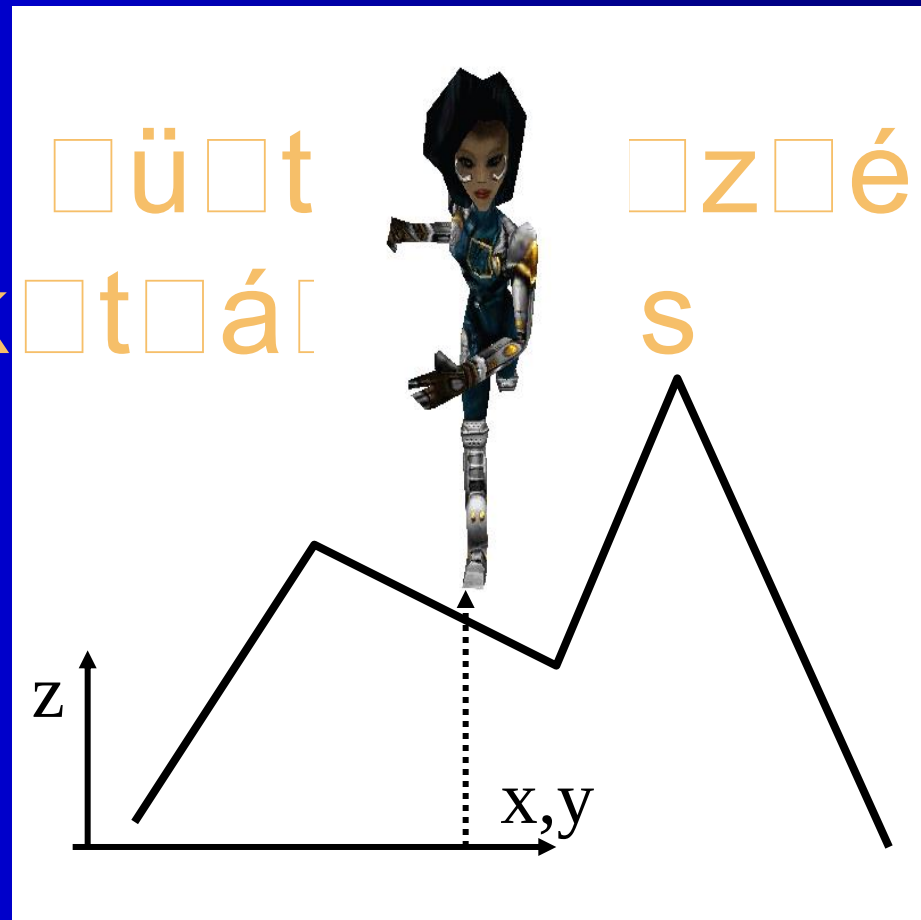


Térpütközés

detektálás

if (height(x,y) > z)

Collision!



Séta a terepén:

Position(x, y) = (x, y, height(x,y) + legsize)

ÉÉg

● Válaszaimra
 – gomb

● Geometriai: dom,

● Nincs

– vezérlés,

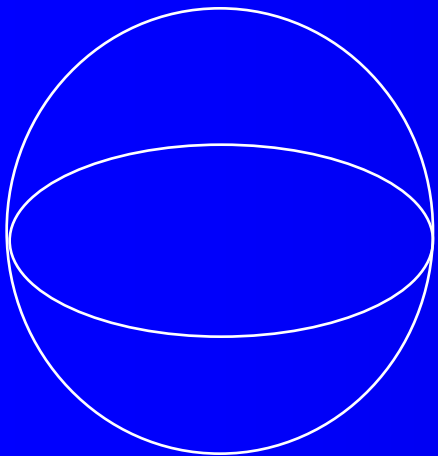
– animáció,

– ütközéssdetektálás

GLU kvadratisk vieldra

```
GLUquadricObj * quadric;           // definition
quadric = gluNewQuadric( );
gluQuadricTexture(quadric, GL_TRUE);

// draw
glBindTexture(GL_TEXTURE_2D, sky_texture_id);
gluSphere(quadric, sky_radius, 32, 20);
```



Ellenség

● Animáltság

• Kulcskeretekkel

• mozgás

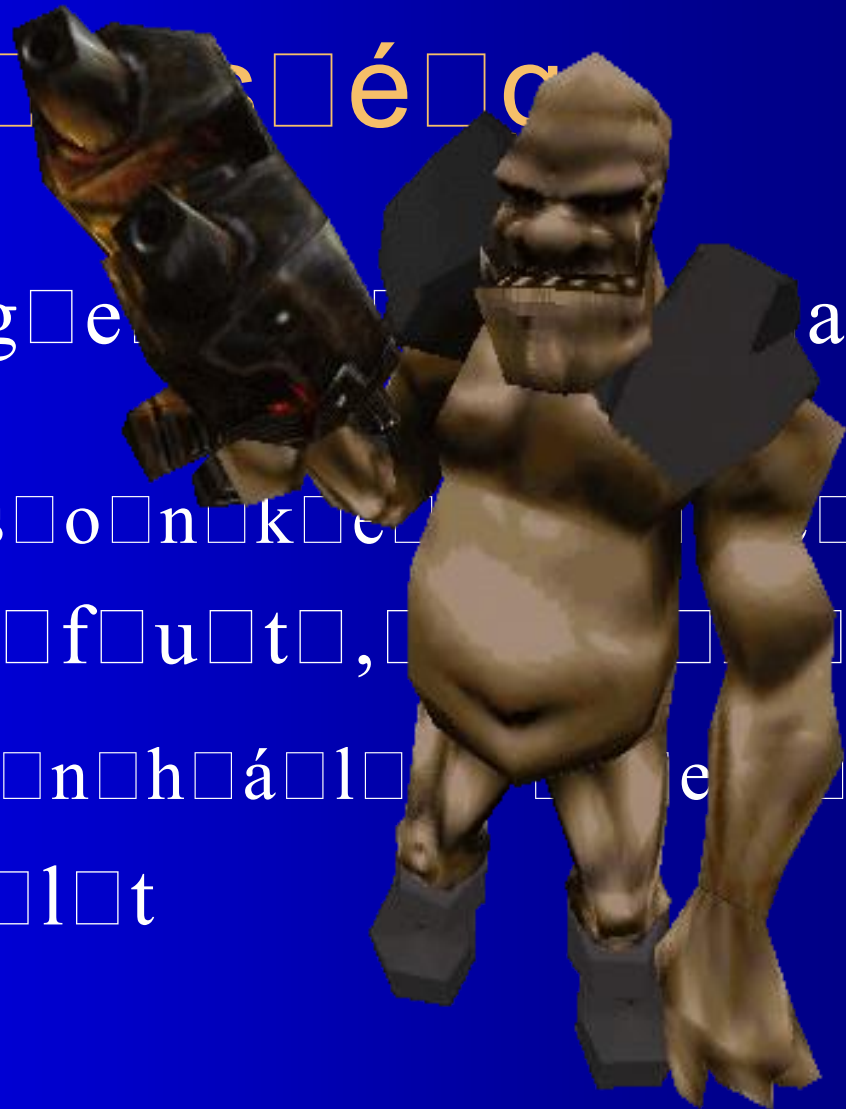
• Áll, fut, a

• poligon

● Textúra

● AI

● Űrtközés

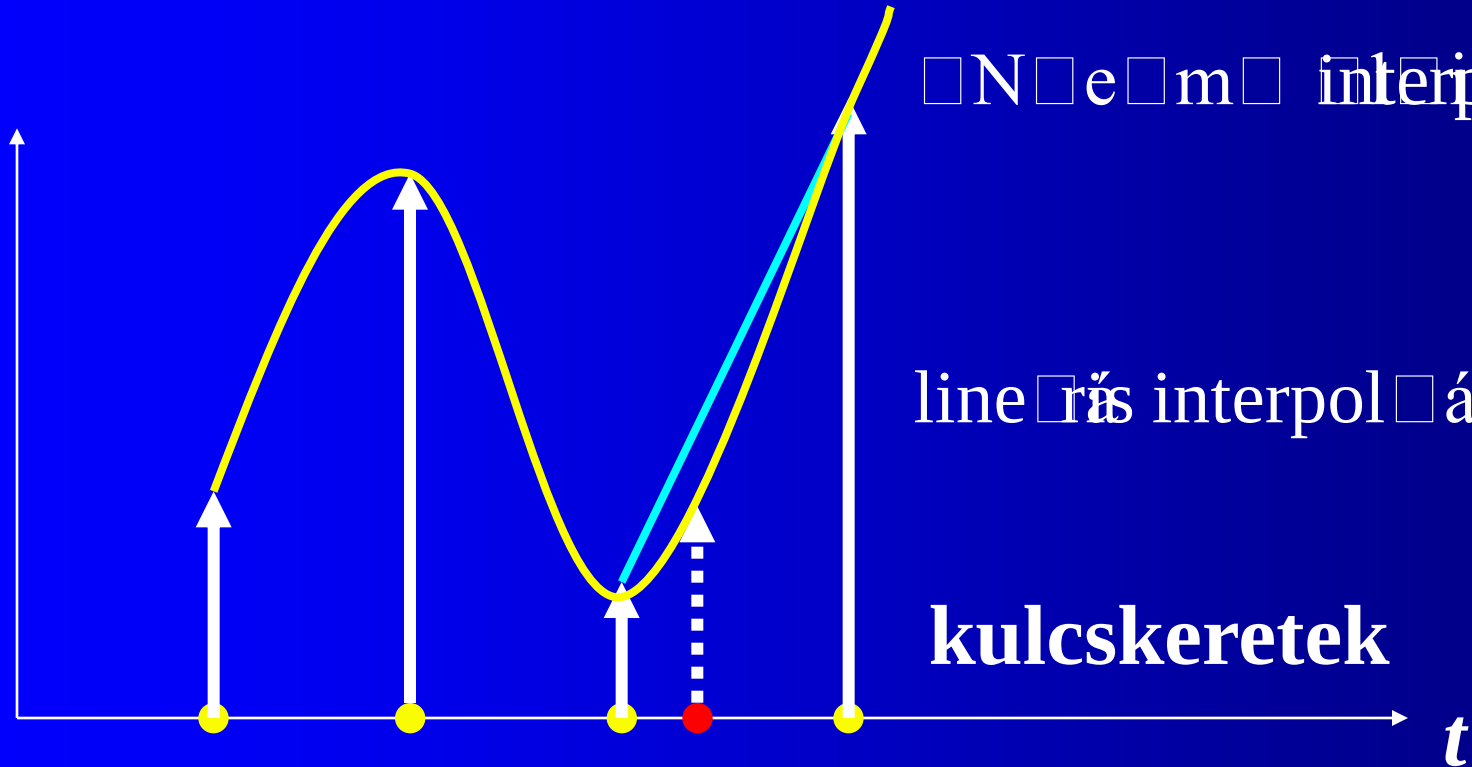


Kulcskeret animáció: futás



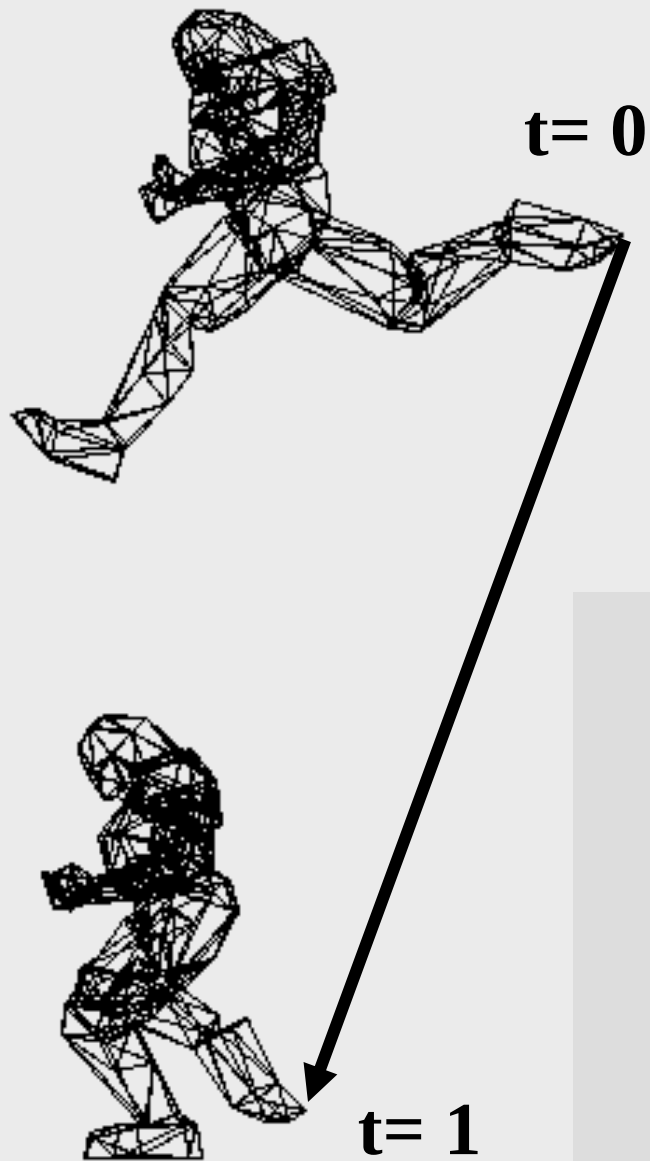
Interpoláció: Kereteklaáció

kulcskeretek

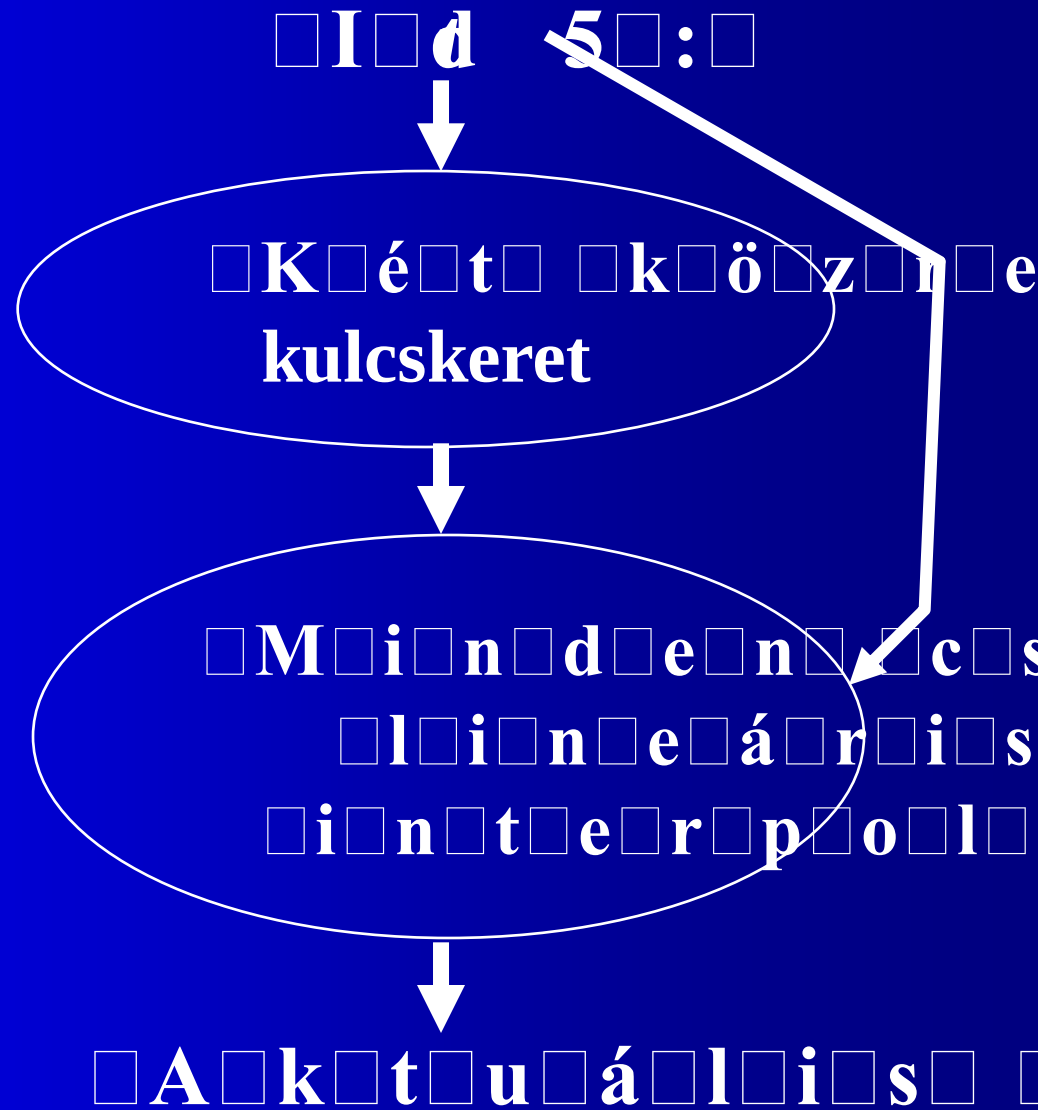


Minimalist?

- Min $\rightarrow$ ségi a nima cíó
- Newton törvények: $\rightarrow$ in
- $\rightarrow$ Interpoláló $\rightarrow$ keret
- Csonat a nima cíó
- Régi $\rightarrow$ jatek
- Lineáris $\rightarrow$ interpoláció
- $\rightarrow$ A poligonháló csúcs
- Poligon (Morphing) $\rightarrow$ formáció



Mesh morphing:



Futás poligonháló deformáció



+ pozíció +

position += velocity * dt



■ M ■ o ■ z ■ g ■ n ■ í ■ á ■ l ■ o ■ s ■ ■ d ■ e ■

■ ● Clip- ■ e ■ k ■ ■ d ■ e ■ f ■ i ■ n ■ í ■ c ■ i ■ ó ■ j ■ a ■ ■

■ ● ■ Ö ■ s ■ s ■ z ■ e ■ s ■ ■ c ■ l ■ i ■ p ■ ■ ö ■ : ■ s ■ s ■ z ■

MD2, MD3

■ ● Tipikus clip-ek:

■ Run, stand, attack, die, pain, salute, crouch,
wave, point, taunt, etc.

Clip-ek



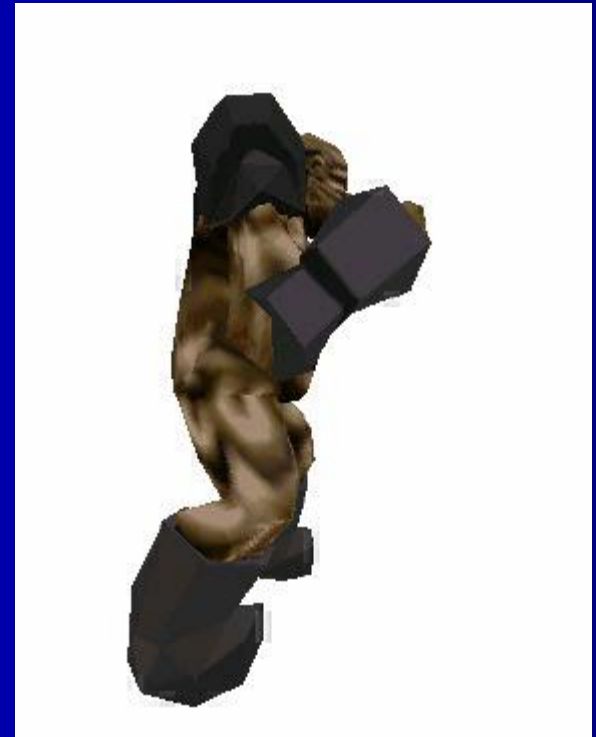
□A□l□l

40 kulcskeret



Fut

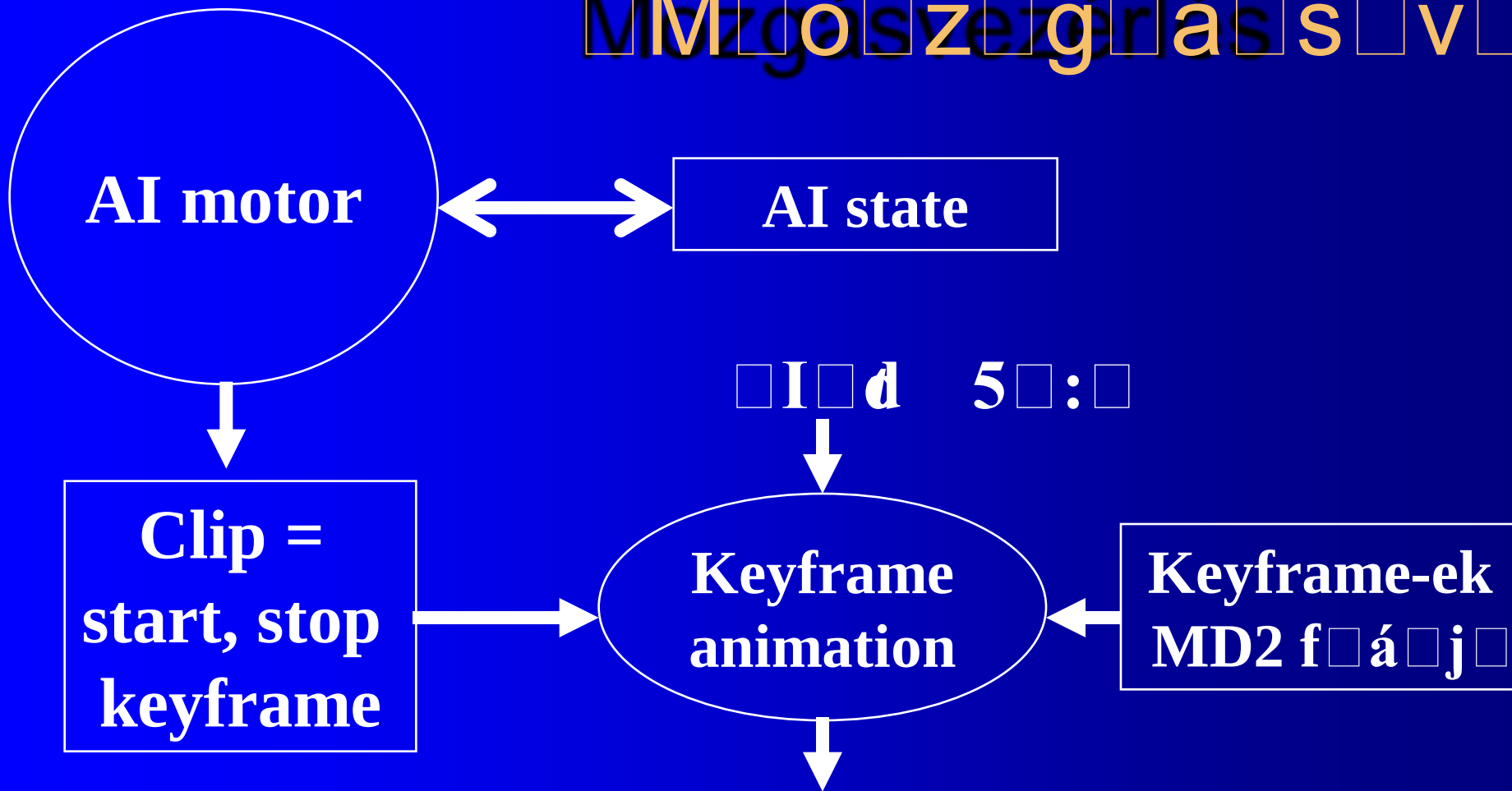
5 kulcskeret



□S□z□a□l□u□

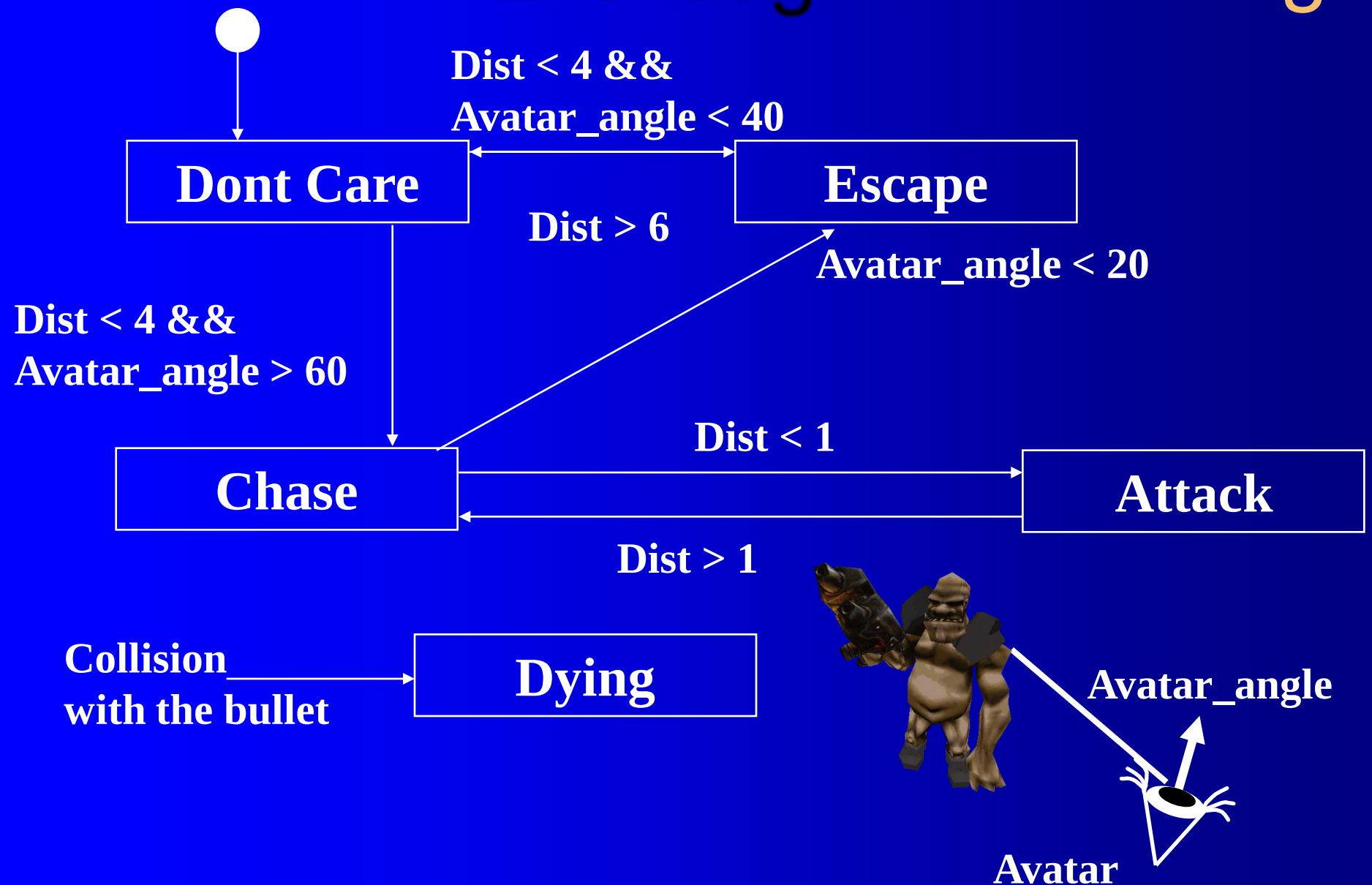
11 kulcskeret

Mozgásvezérlás

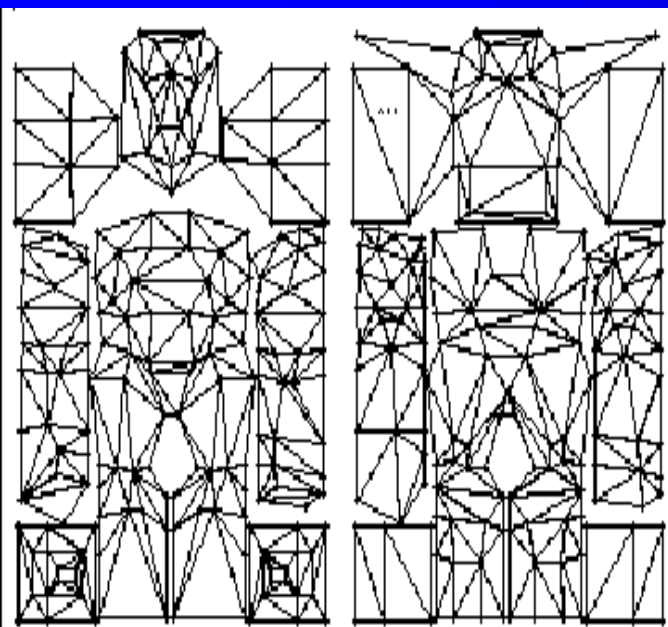


Animációs hálózati motor

Ellátás



Textúrázás



Gyólyó

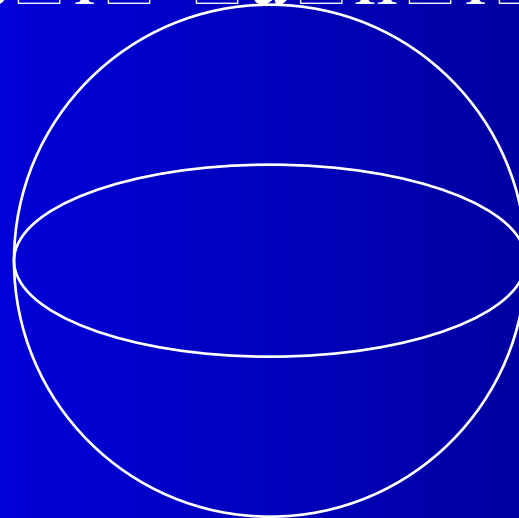


● Géométeria:

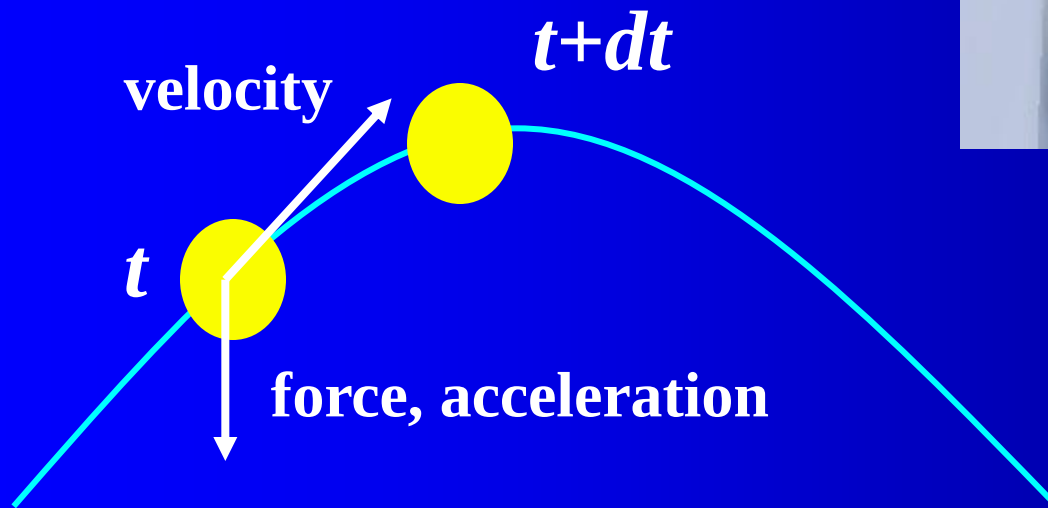
● Textúrazt

● Nem intelligens

● Fizikai a n i m á c i ó



Lövedék fizikai animációja



acceleration = (0, 0, -g)

velocity += acceleration * dt

position += velocity * dt

Röpel: Animate, Draw

```
void Bullet::AnimateIt( float dt ) {  
    acceleration = Vector(0,0,-g);  
    velocity += acceleration * dt;  
    position += velocity * dt;  
}
```

```
void Bullet::DrawIt( ) {  
    glPushMatrix( );  
        glTranslate(position.x, position.y, position.z);  
        glBindTexture(GL_TEXTURE_2D, bullet_texture);  
        gluSphere(quadric, 1.0, 16, 10);  
    glPopMatrix( );  
}
```