Számítógépes grafika
Bevezetés

Szirmay-Kalos László

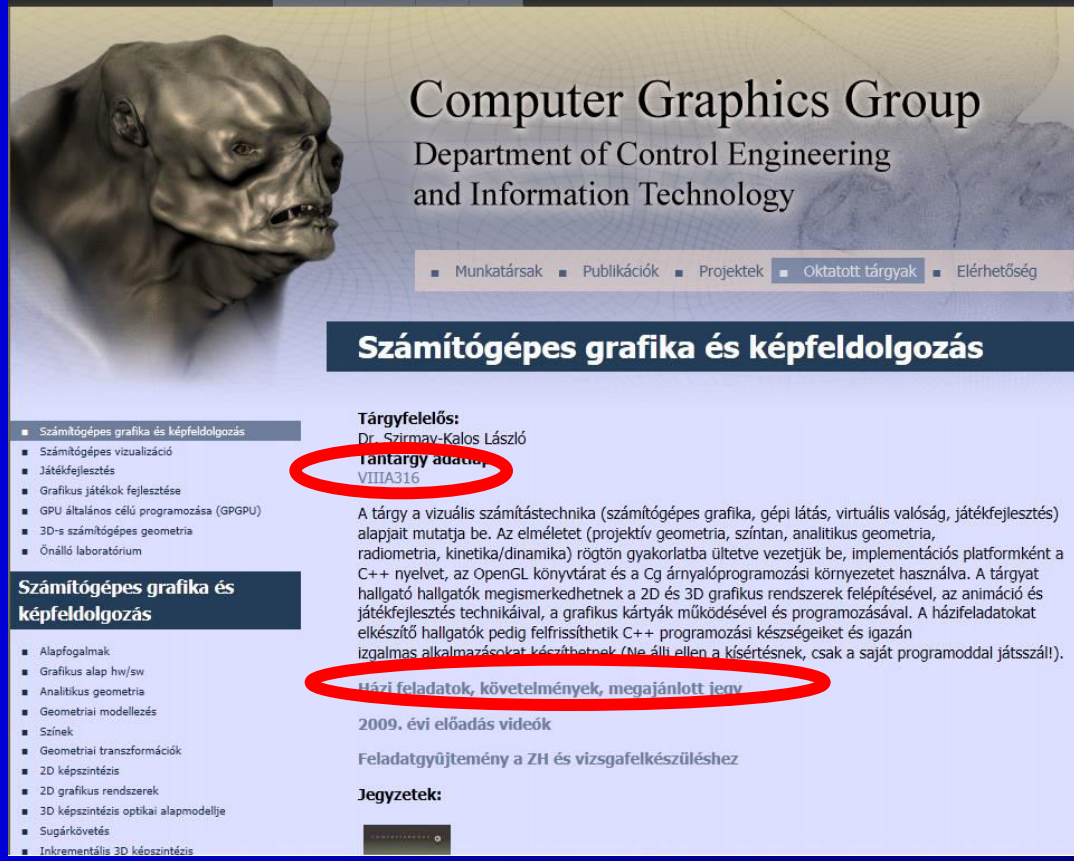
email: szirmay@iit.bme.hu

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

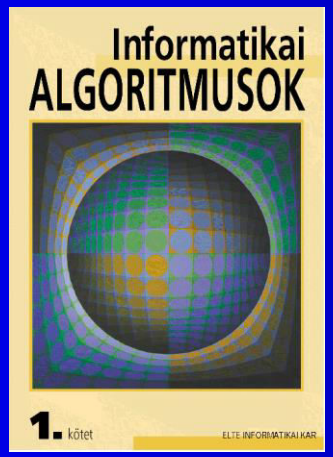
http://cg.iit.bme.hu

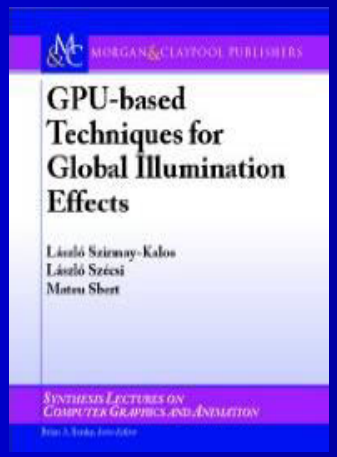


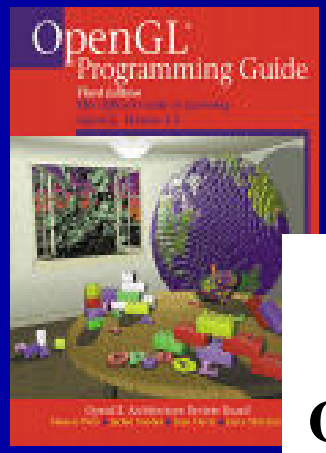




3 db kötelező kisházi
1 db önkéntes nagyházi (shader prog)
vizsga



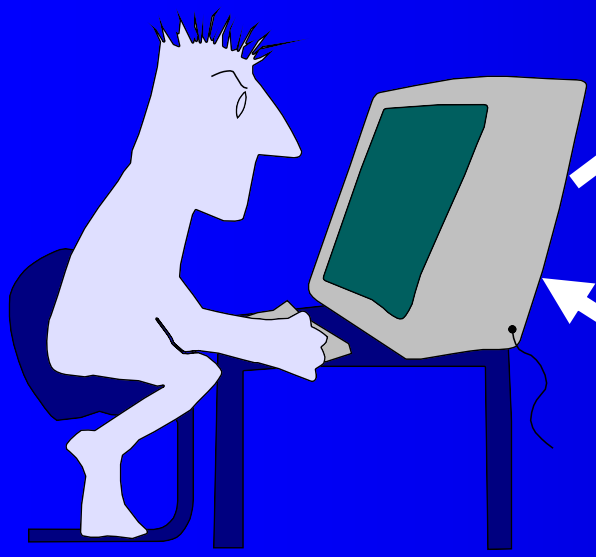




Juhász Imre OpenGL

Számítógépes grafika

illuzió



modellezés

képszintézis

számok

Virtuális világ

Számítás

mérés

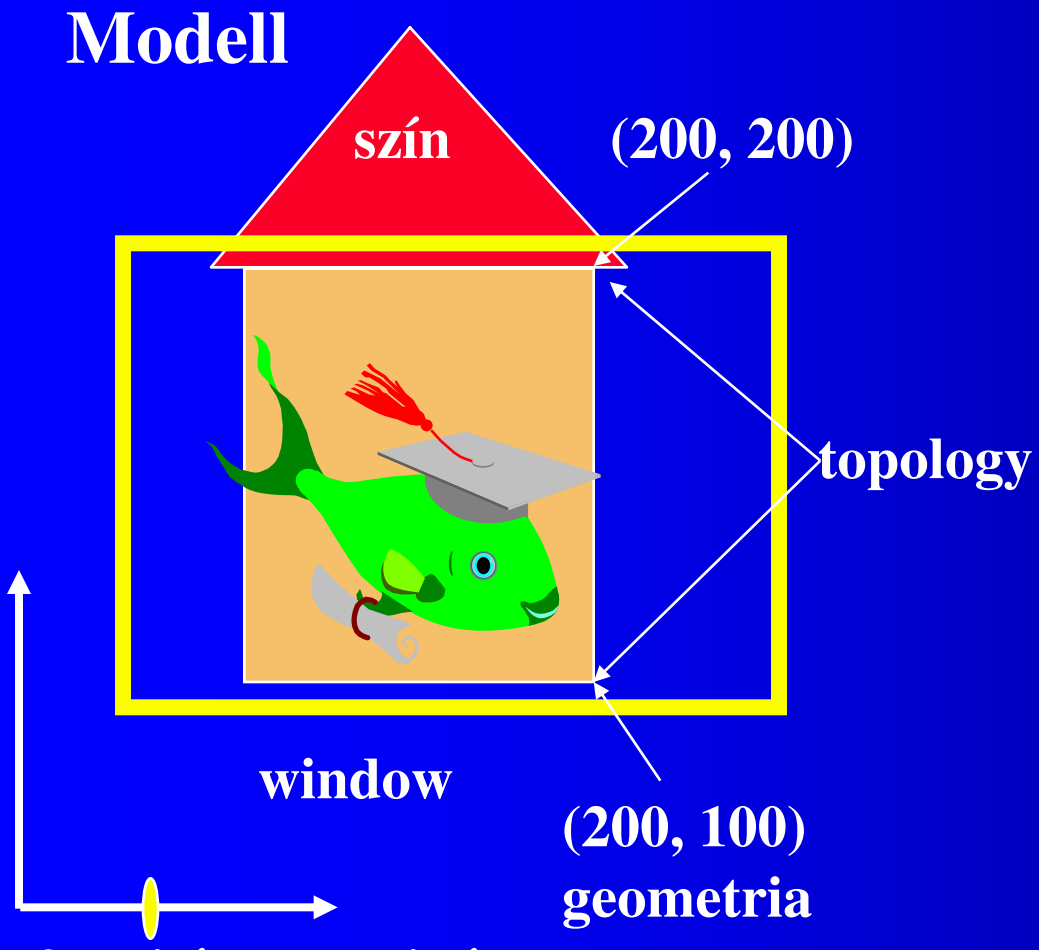
pixelek:

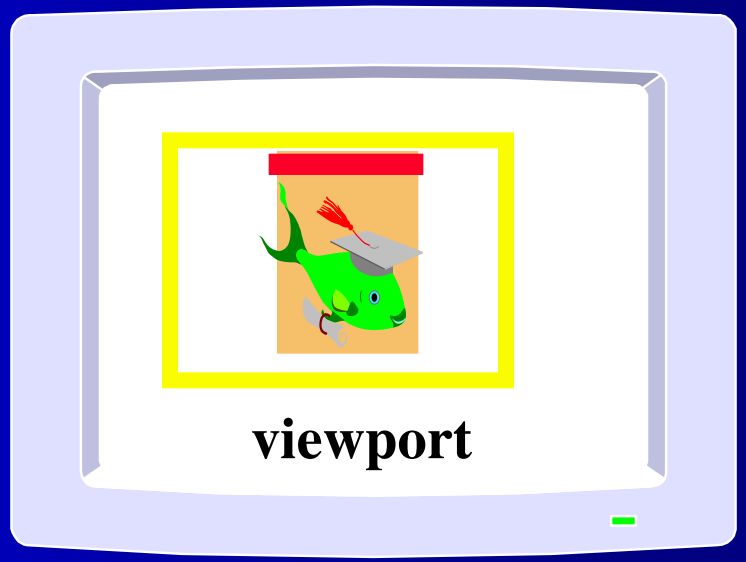
- **piros**
- **zöld**
- **kék**

Analógia:

- **optika**
- **rajzolás**
- **stb.**

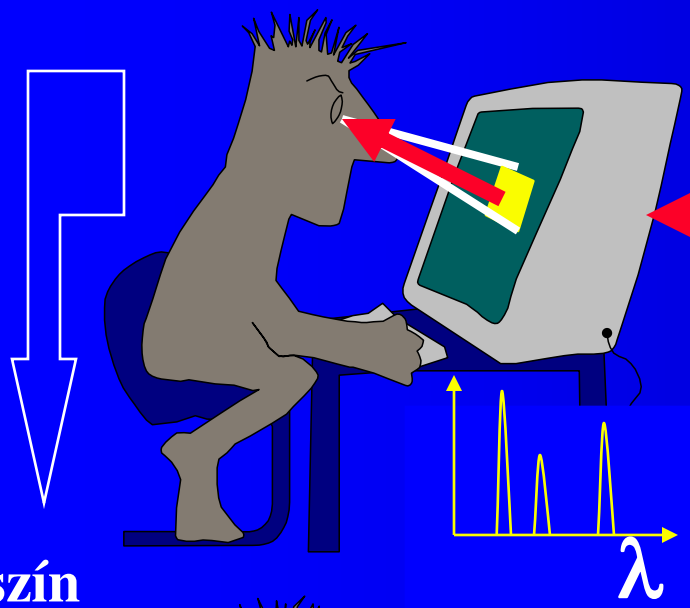
2D képszintézis analógia: rajzolás





Saját színnel rajzolás

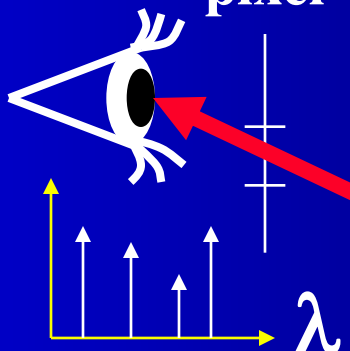
3D képszintézis analógia: optika

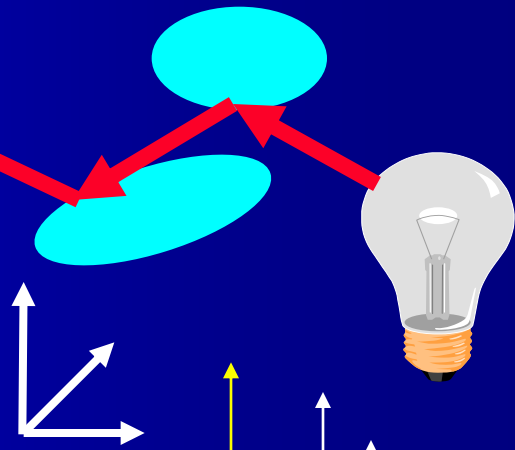


Tone mapping

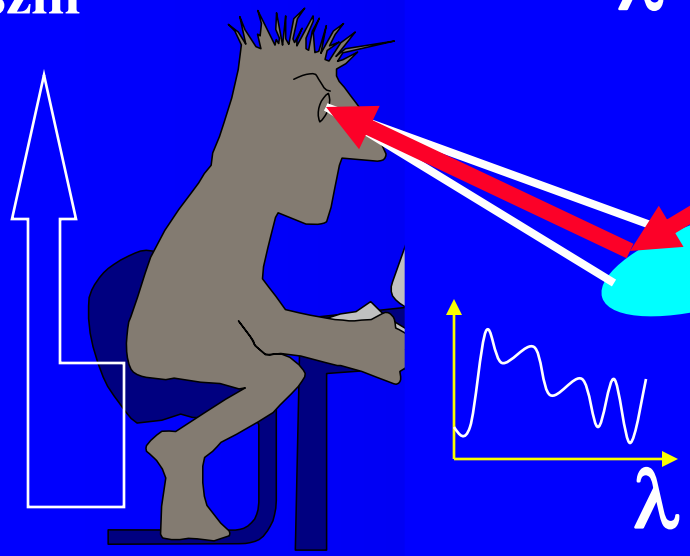
pixel

Virtuális világ

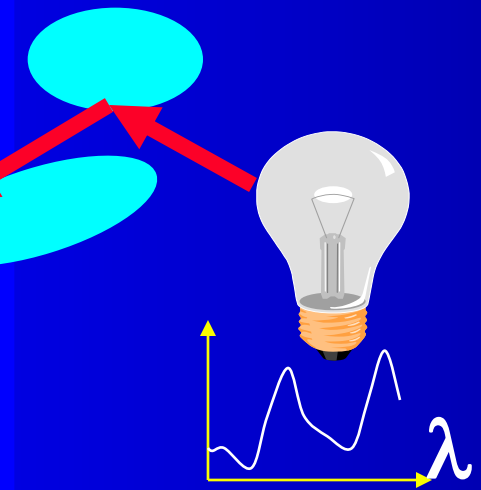




szín



Valós világ



$$\vec{\nabla} \cdot \vec{D} = \rho$$

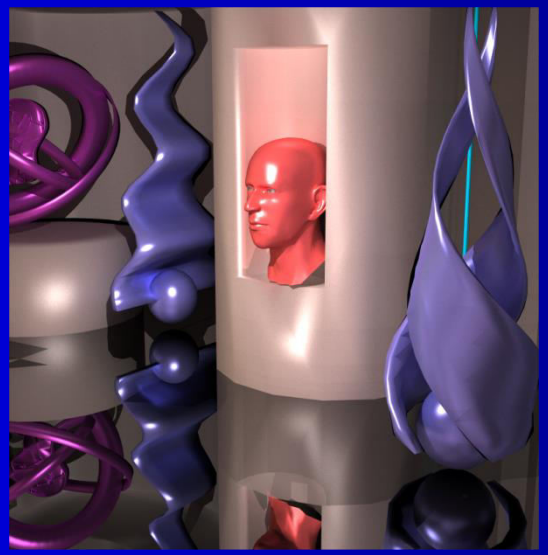
$$\vec{\nabla} \cdot \vec{B} = 0$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$$

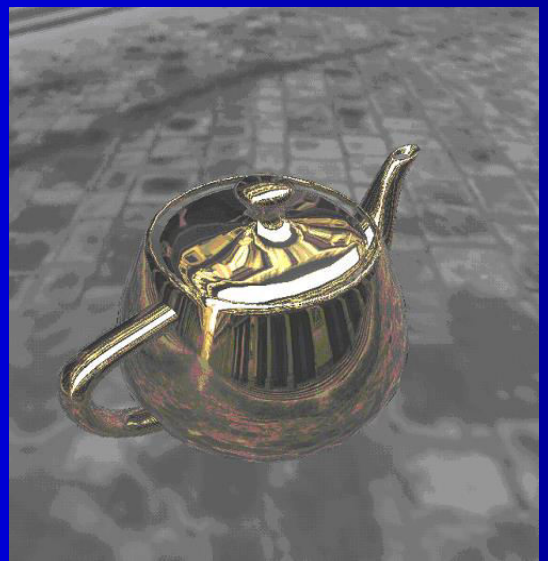
$$\vec{\nabla} \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t}$$

Optika:Fotorealisztikus képszintézis



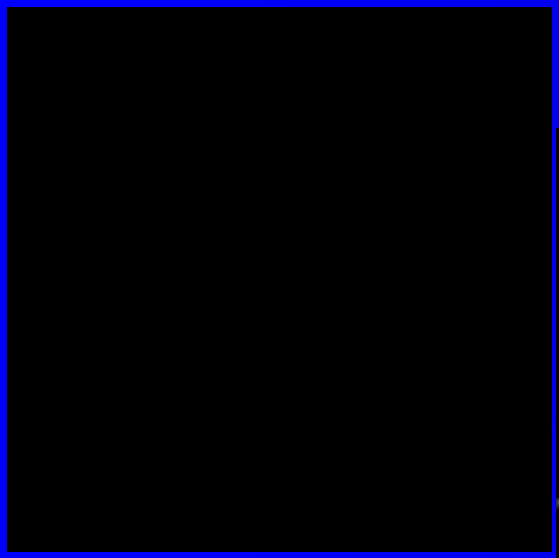


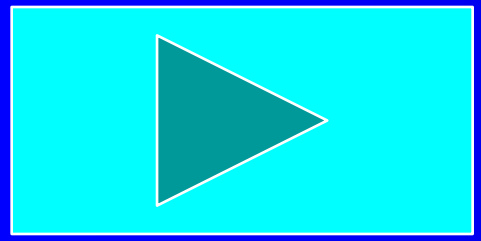




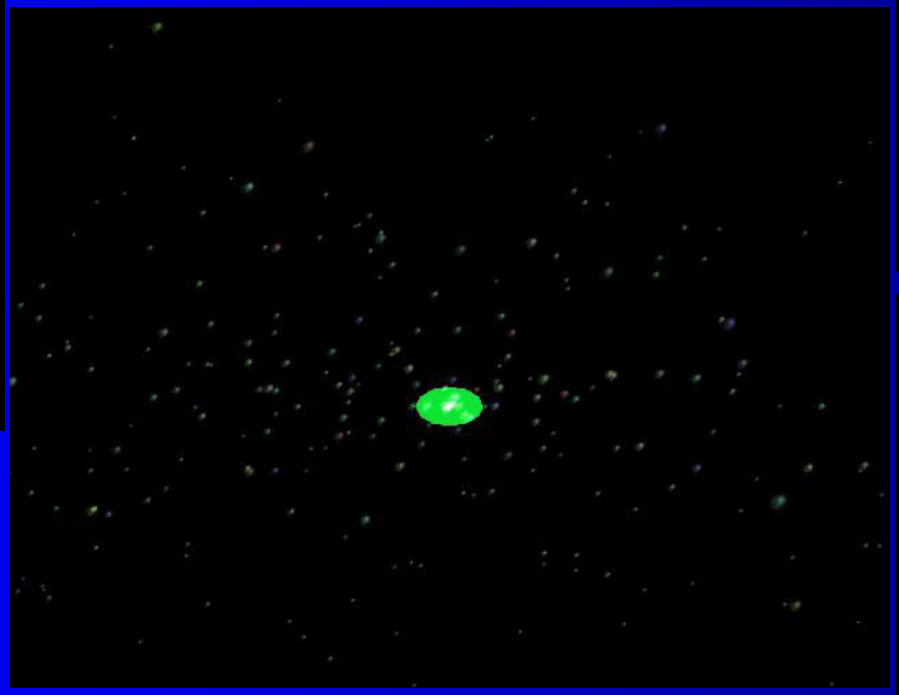


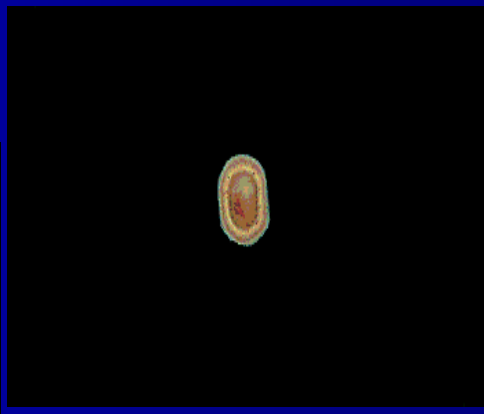
Kihívások





SSDS: Sloan Digital Sky Survey







- Nagy modell(giga/terabyte)
- Valós idő: pár nsec/pixel

Analitikus geometria gyorstalpaló

Szirmay-Kalos László

Geometriák

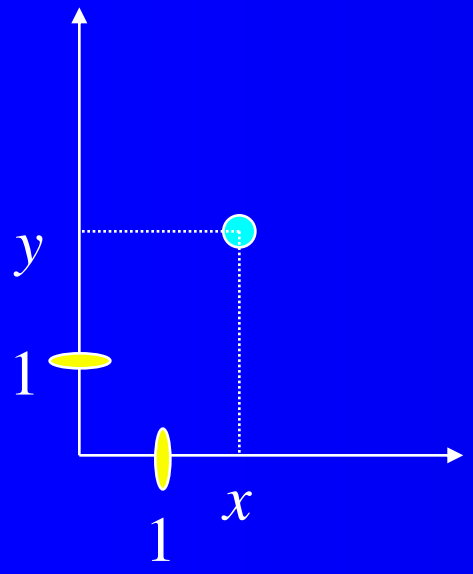
- Két pont meghatároz egy egyenest.
- Egy egyenesnek van legalább két pontja.
- Ha *a* egy egyenes, A pedig egy, nem az egyenesen lévő pont, akkor egyetlen olyan egyenes létezik, amely átmegy A-n és nem metszi a-t.

- Geometria
  - Axiómák
    - Alapigazság (tapasztalat)
    - Alapfogalmak implicit definíciója
  - Definíciók és tételek
  - Célok és eszközök
- Fontos geometriák a számítógépes grafikában
  - Euklideszi (sík/tér, metrikus)
  - Projektív
  - Fraktális
  - …

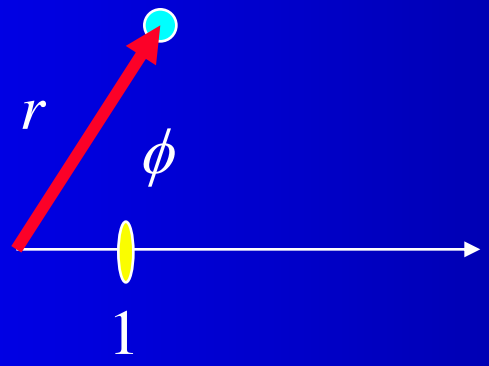
Mindent számmal: analitikus geometria



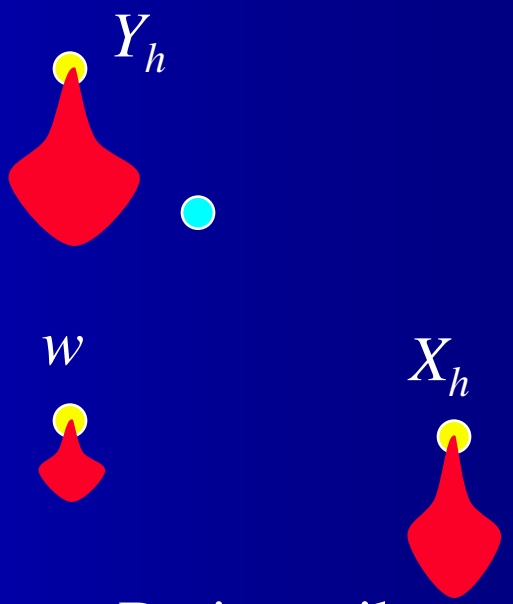
Pontok definíciója



Decartes



Polár

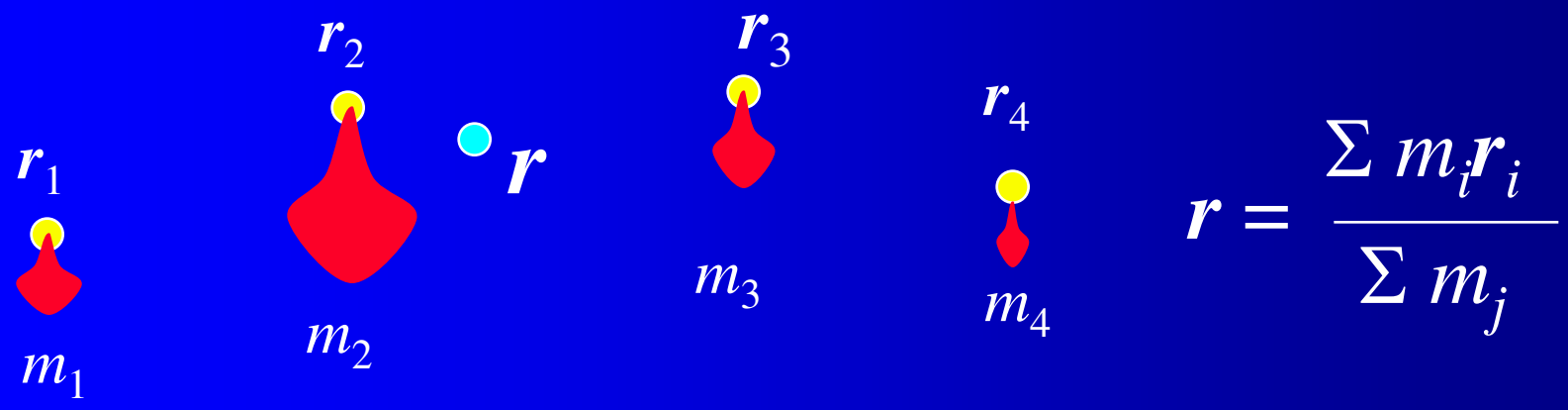


Baricentrikus
Homogén

<u>Számokkal!</u>

1. Koordinátarendszer (=referencia geometria)
2. Koordináták(=mérés)

Pontok kombinálása



- r az $r_1, r_2, \ldots r_n$ pontok kombinációja
- Súlyok a baricentrikus koordináták
- Ha a súlyok nem negatívak: konvex kombináció
- Konvex kombináció a konvex burkon belül van
- Egyenes (szakasz) = két pont (konvex) kombinációja
- Sík (háromszög) = három pont (konvex) kombinációja

Vektor

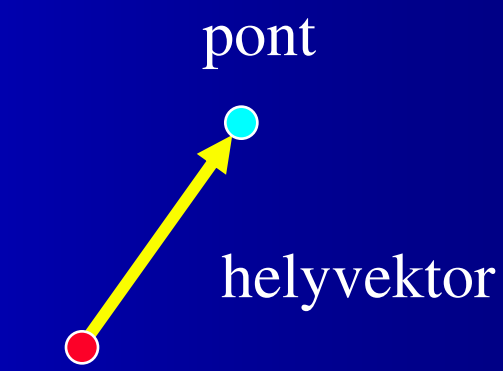
- Vektor = eltolás: $\boldsymbol{v}$
- Irány és hossz ($|\boldsymbol{v}|$)
- Helyvektor

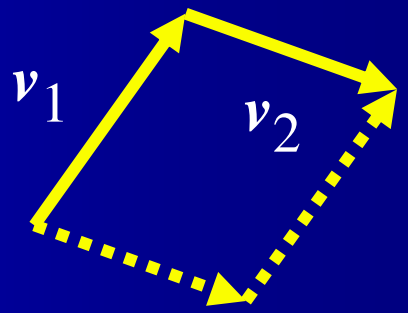
  DE vektor ≠ pont !!!
- Vektor összeadás

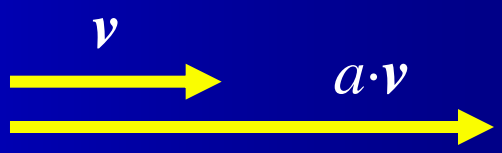
  $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$ (kommutativ, asszoc.)

  $\boldsymbol{v}_1 = \boldsymbol{v} - \boldsymbol{v}_2$ (van inverz)
- Skálázás (skalárral szorzás)

  $\boldsymbol{v}_1 = a \cdot \boldsymbol{v}$ (disztributív)







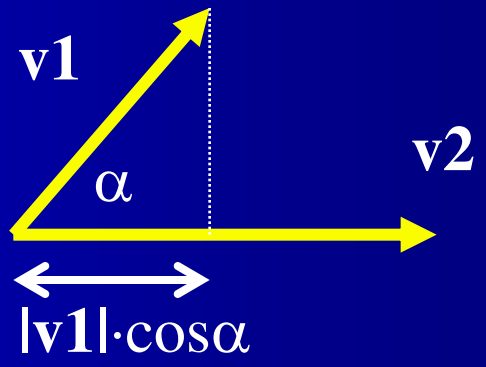
Skalár (dot, belső) szorzat

- Definíció

  v1·v2 = |v1|·|v2|·cosα

- Jelentés

  Egyik vektor vetülete a másikra * másik hossza



- Tulajdonság

  Nem asszociatív!!!

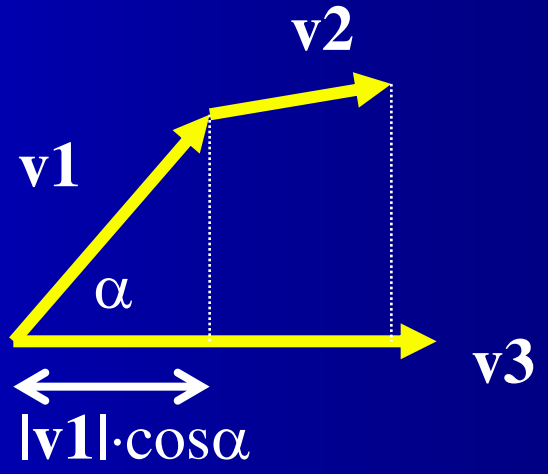
  Kommutatív

  v1·v2 = v2·v1

  Disztributív az összeadással

  v3·(v2+v1) = v3·v2 + v3·v1

  v·v = |v|²



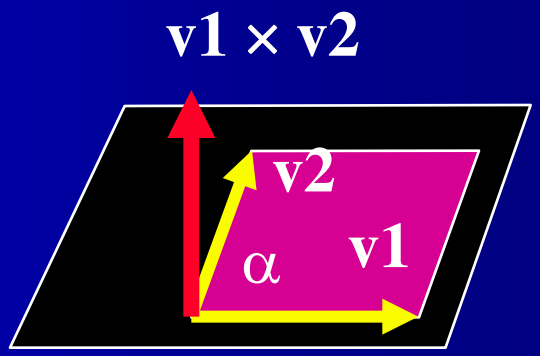
Két vektor merőleges ha a skalárszorzatuk zérus

Vektor (kereszt) szorzat

- Definíció

  $|\mathbf{v1} \times \mathbf{v2}| = |\mathbf{v1}| \cdot |\mathbf{v2}| \cdot \sin\alpha$

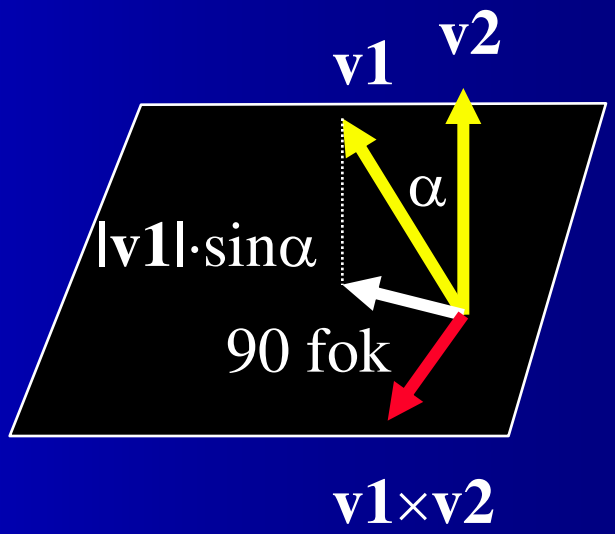
  Merőleges, jobb kéz szabály



- Jelentés

  <u>Terület és merőleges vektor</u>,

  (Egyik vektor vetülete a másikra merőleges síkra + 90 fokos elforgatás) * másik hossza



- Tulajdonságok

  Nem asszociatív!!!

  Antiszimmetrikus

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

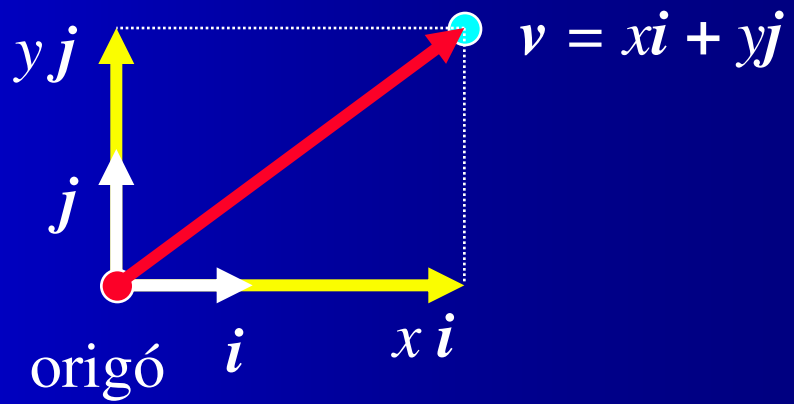
  Disztributív az összeadással

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

Két vektor párhuzamos ha vektorszorzatuk zérus.

Descartes koordináta rendszer

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



Összeadás:

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

Skalár szorzat:

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

Vektor szorzat:

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

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

Hossz:

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

Vektor és pont különböző!

- 2D: Vector (x, y)
- 3D: Vector (x, y, z)

Point (x, y)

Point (x, y, z)

- Műveletek:

Vector + Vector = Vector

Vector – Vector = Vector

Vector × Vector = Vector

Vector · Vector = Scalar

Vector * Scalar = Vector

|Vector| = Scalar

Point + Vector = Point (translation)

Point – Point = Vector

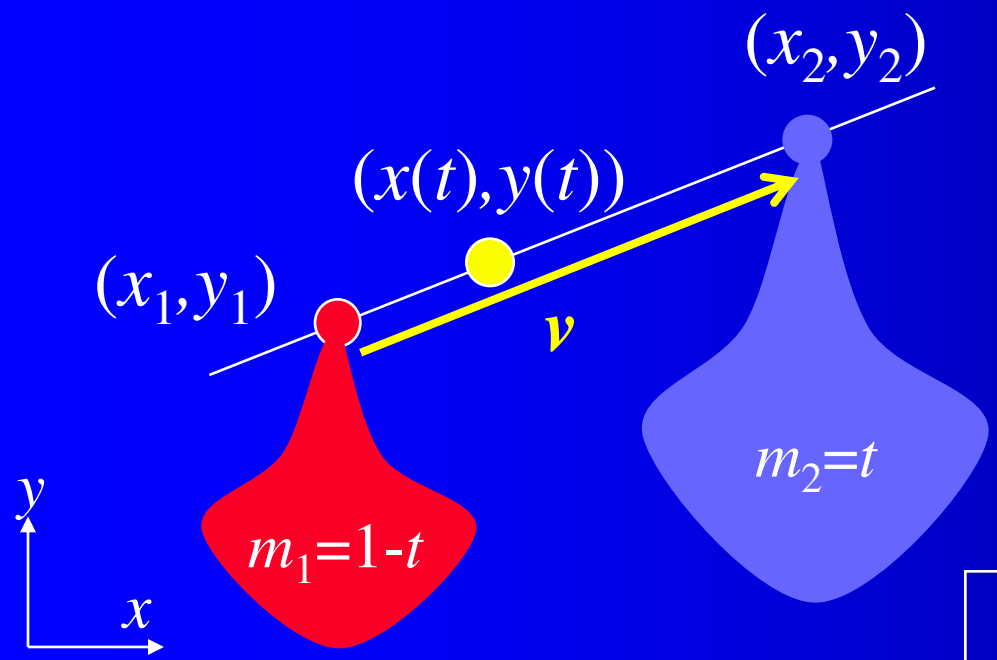
Point = Combination of Points

Distance(Point, Point) = |Point– Point|
= Scalar

Vector/Point class

```
struct float3 {
    float x, y, z;

    float3(float x0, float y0, float z0) { x=x0; y=y0; z=z0; }

    float3 operator*(float a) {
        return float3(x * a, y * a, z * a);
    }
    float3 operator+(const float3& v) {
        return float3(x + v.x, y + v.y, z + v.z);
    }
    float3 operator-(const float3& v) {
        return float3(x - v.x, y - v.y, z - v.z);
    }
    float Length() { return sqrtf(x * x + y * y + z * z); }
};

float dot(const float3& v1, const float3& v2) {
    return (v1.x * v2.x + v1.y * v2.y + v1.z * v2.z);
}
float3 cross(const float3& v1, const float3& v2) {
    return float3(v1.y * v2.z - v1.z * v2.y,
                  v1.z * v2.x - v1.x * v2.z,
                  v1.x * v2.y - v1.y * v2.x);
}
```

Egyenes (szakasz) mint két pont kombinációja



$$\boldsymbol{r} = \frac{\Sigma\, m_i \boldsymbol{r}_i}{\Sigma\, m_j}$$

fizikából

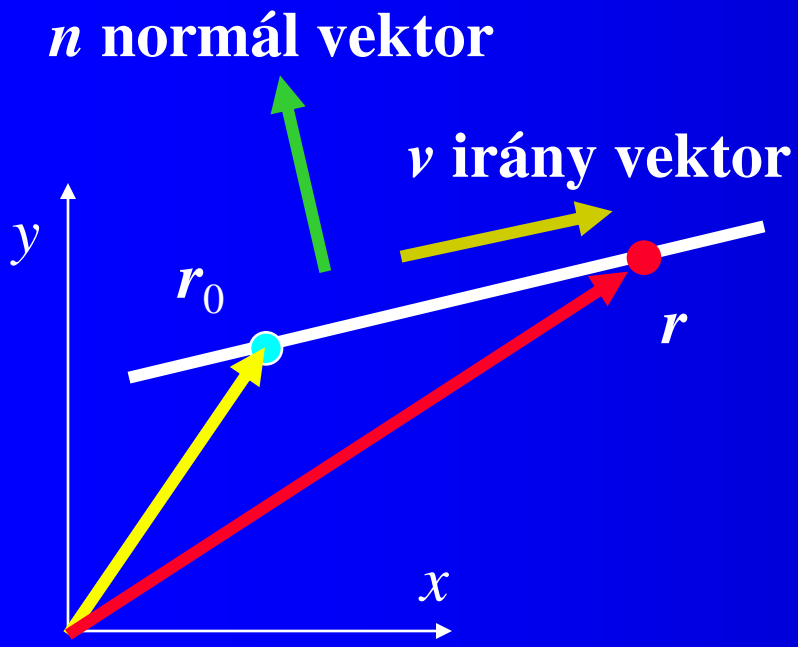
$$x(t)=x_1(1-t)+x_2t$$
$$y(t)=y_1(1-t)+y_2t$$

Paraméteres egyenlet

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

$$x(t) = x_1 + v_x\, t$$
$$y(t) = y_1 + v_y\, t$$

2D egyenes



Implicit egyenlet

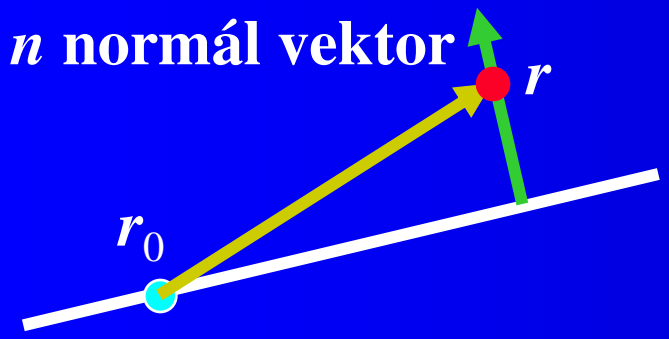
$$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)=0$$

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

$$ax+by+c=0$$

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

2D egyenestől mért távolság:

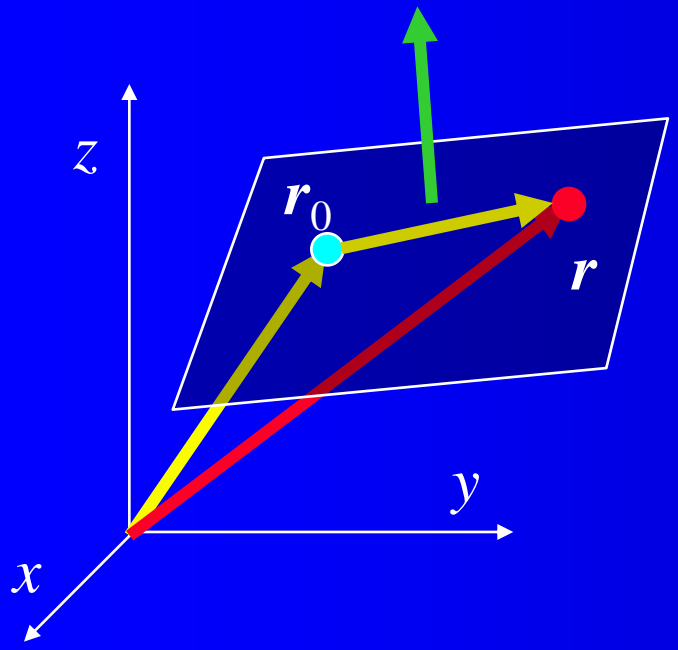


$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)$ = Vetület $\boldsymbol{n}$-re * az $\boldsymbol{n}$ hossza

Ha $\boldsymbol{n}$ egységvektor:
$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)$ az előjeles távolság!

Sík

***n* normál vektor**



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

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

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

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

Ha $\boldsymbol{n}$ egységvektor:

$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)$ a síktól mért távolság!

Kör a síkon

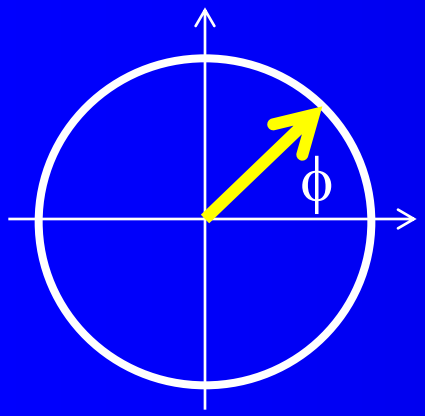
Implicit egyenlet:

Azon $\boldsymbol{r}(x, y)$ pontok mértani helye, amelyek a $\boldsymbol{c}(c_x, c_y)$ középponttól R távolságra vannak:

$$|\boldsymbol{r} - \boldsymbol{c}| = R \quad \leftrightarrow \quad (\boldsymbol{r} - \boldsymbol{c})^2 = R^2 \quad \leftrightarrow \quad (x - c_x)^2 + (y - c_y)^2 = R^2$$

Paraméteres egyenlet:

Asin(ϕ) és cos(ϕ) definíciója:



$$x(\phi) = c_x + R\cos(\phi)$$
$$y(\phi) = c_y + R\sin(\phi)$$

Gyakorlatok

- 2D:
  - A parabola implicit egyenete (azon pontok mértani helye, amelyek egyenlő távolságban vannak a $\boldsymbol{p}$ ponttól és az ($\boldsymbol{r}0$, $\boldsymbol{n}$) egyenestől),
  - Az ellipszis (a $\boldsymbol{p}1$, $\boldsymbol{p}2$ pontoktól mért távolságösszeg = C)
  - Koordinátatengelyekkel párhuzamos tengelyű ellipszis paraméteres egyenlete
- 3D
  - Gömb, henger és paraboloid implicit egyenlete
  - Két kitérő egyenes távolsága: ($\boldsymbol{r}1$, $\boldsymbol{v}1$) és ($\boldsymbol{r}2$, $\boldsymbol{v}2$)
  - Azon pontok halmaza, amelyek $\boldsymbol{p}1$, $\boldsymbol{p}2$ pontoktól mért távolságnégyzet összege = C.

Geometriai modellezés

Szirmay-Kalos László

Virtuális világ létrehozása

- Geometria
  - 2D: pont, síkgörbe, terület, fraktál ($D \leq 2$)
  - 3D: pont, térgörbe, felület, test, fraktál ($D \leq 3$)
- Transzformáció (mozgatás)
- Optikai tulajdonságok

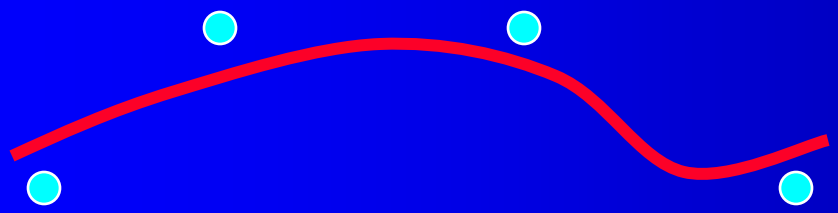
Görbék: 1D ponthalmazok

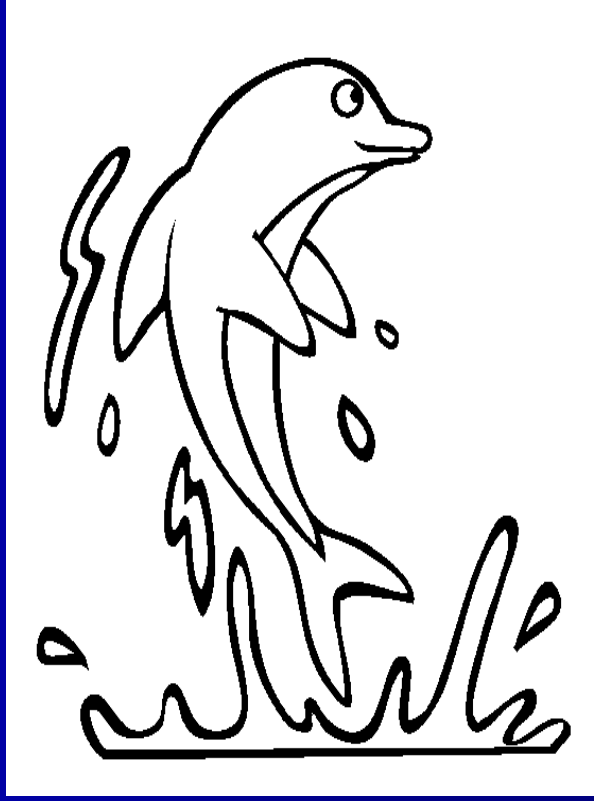
A pontok koordinátái kielégítenek egy egyenletet:

– **implicit**: $f(x, y) = 0$	$f(\boldsymbol{r}) = 0$
• 2D egyenes: $ax + by + c = 0$	$\boldsymbol{n}\cdot(\boldsymbol{r} - \boldsymbol{r}_0) = 0$
• Kör: $(x-x_0)^2+(y-y_0)^2-R^2 = 0$	$\lvert\boldsymbol{r} - \boldsymbol{r}_0\rvert^2 - R^2 = 0$
– **parametrikus**: $x = x(t),\ y = y(t)$	$\boldsymbol{r} = \boldsymbol{r}(t)$
• 3D egyenes: $x = x_0 + v_x t$ $y = y_0 + v_y t$ $z = z_0 + v_z t$	$\boldsymbol{r} = \boldsymbol{r}_0 + \boldsymbol{v}\, t, \quad t \in [-\infty,\infty]$
• Kör: $t \in [0,1]$ $x(t) = x_0 + R\cos 2\pi t$ $y(t) = y_0 + R\sin 2\pi t$	$\boldsymbol{r} = \boldsymbol{r}_0 + R(\cos 2\pi t,\ \sin 2\pi t)$
– **explicit:** $y = F(x)$	
• 2D egyenes: $y = mx + b$	

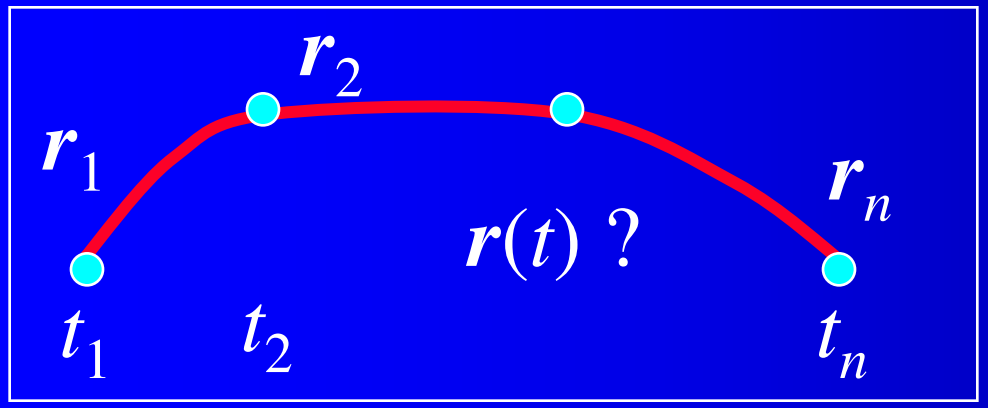
Szabadformájú görbék

- Definíció vezérlőpontokkal





- Polinom: $x(t) = \Sigma\, a_i\, t^i, \quad y(t) = \Sigma\, b_i\, t^i$
- Polinom együtthatók:
  - Interpoláció
  - Approximáció



Lagrange interpoláció

- Vezérlő pontok: $r_1, r_2, r_3, \ldots, r_n$
- Keresd: $r(t) = \Sigma\, [a_i, b_i]\, t^i$ amelyre

  $r(t_1) = r_1, r(t_2) = r_2, \ldots, r(t_n) = r_n,$
- Megoldás:

$$r(t) = \Sigma\, L_i(t)\, r_i \longrightarrow r(t_k) = \Sigma\, L_i(t_k)\, r_i = r_k$$

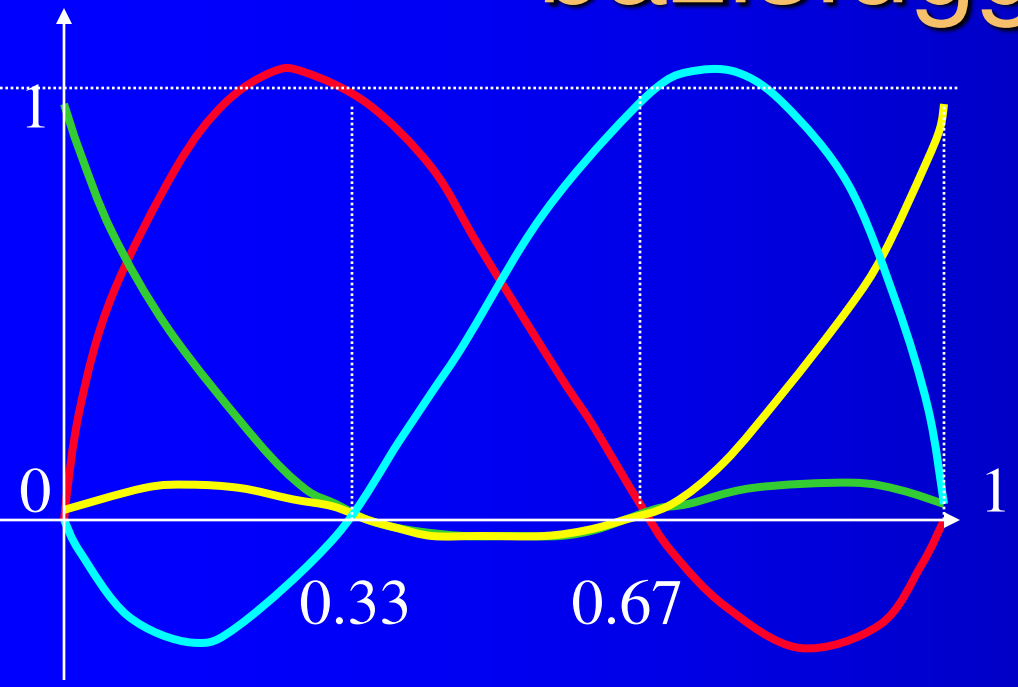
$$L_i(t) = \frac{\Pi_{j \neq i}\,(t - t_j)}{\Pi_{j \neq i}\,(t_i - t_j)}$$

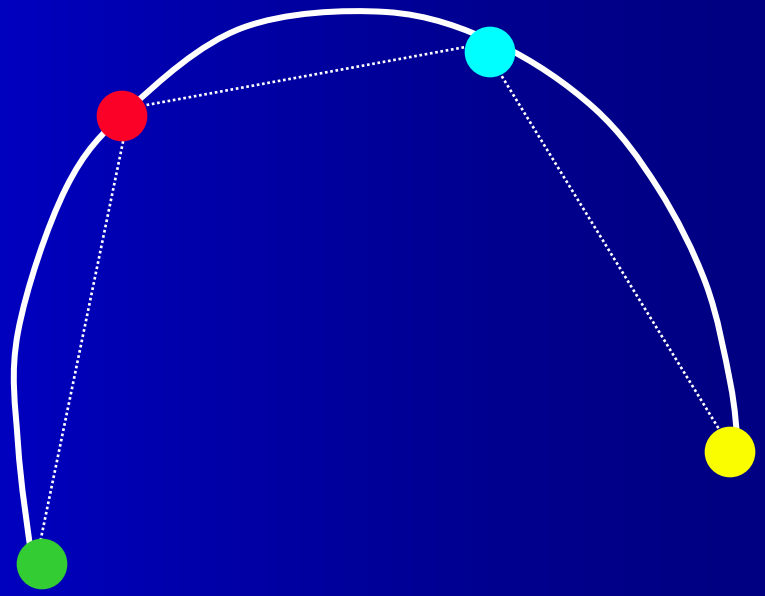
$$L_i(t_k) = \frac{\Pi_{j \neq i}(t_k - t_j)}{\Pi_{j \neq i}(t_i - t_j)} = \begin{cases} 1 \text{ if } i=k \\ 0 \text{ if } i \neq k \end{cases}$$

LagrangeCurve

```
class LagrangeCurve {
    vector<float3> cps;      // control points
    vector<float>  ts;       // parameter (knot) values

    float L(int i, float t) {
        float Li = 1.0f;
        for(int j = 0; j < cps.size(); j++)
                if (j != i) Li *= (t - ts[j])/(ts[i] - ts[j]);
        return Li;
    }
public:
    void AddControlPoint(float3 cp) {
        float ti = cps.size();     // or something better
        cps.push_back(cp); ts.push_back(ti);
    }

    float3 r(float t) {
        float3 rr(0, 0, 0);
        for(int i = 0; i < cps.size(); i++) rr += cps[i] * L(i,t);
        return rr;
    }
};
```

Lagrange interpoláció bázisfüggvényei

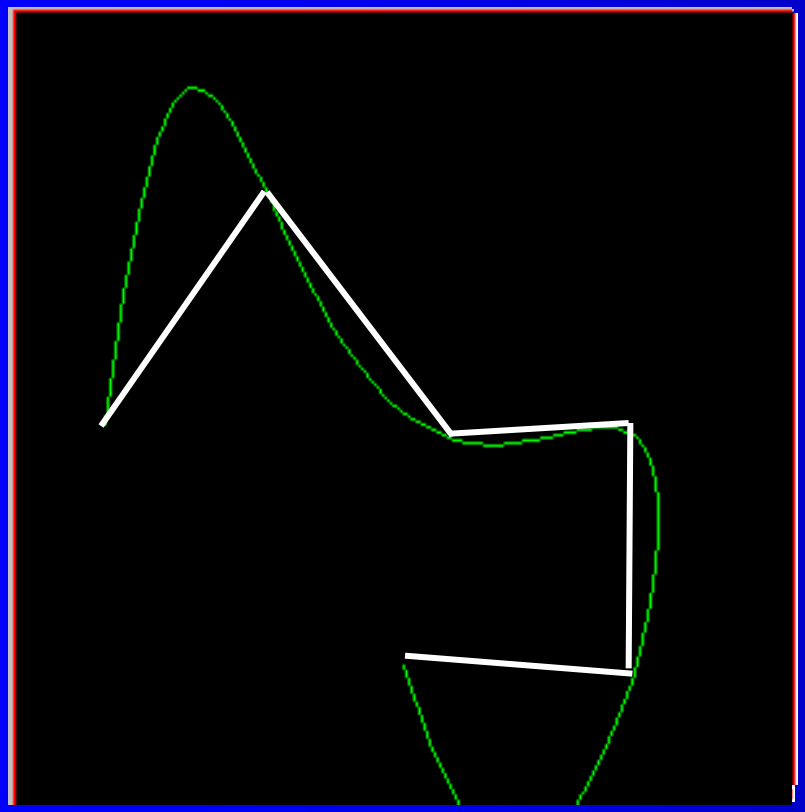


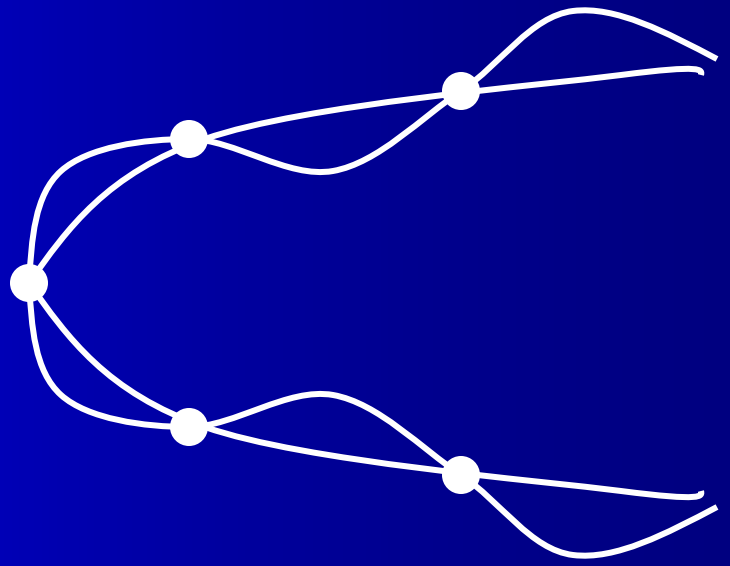


$$L_i(t) = \frac{\prod_{j \neq i} (t - t_j)}{\prod_{j \neq i} (t_i - t_j)}$$

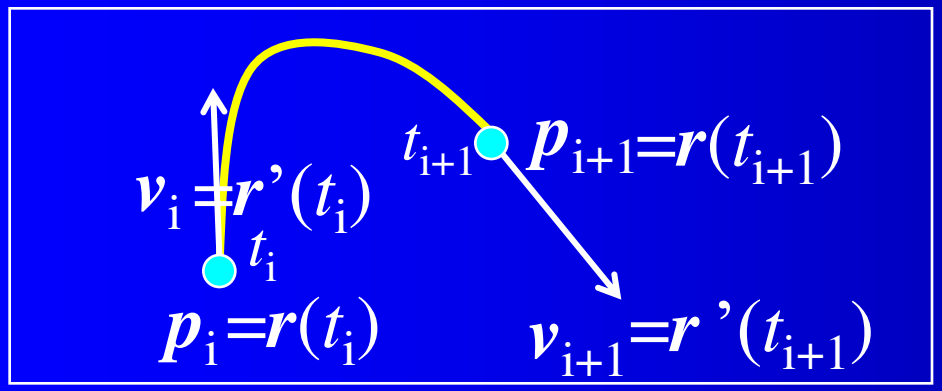
$$\boldsymbol{r}(t) = \Sigma\, L_i(t)\, \boldsymbol{r}_i$$

Lagrange interpoláció problémái





Hermite interpoláció



- $r(t) = a_3(t-t_i)^3 + a_2(t-t_i)^2 + a_1(t-t_i) + a_0$
- $r_i'(t) = 3a_3(t-t_i)^2 + 2a_2(t-t_i) + a_1$

$$r(t_i) = a_0 = p_i$$

$$r(t_{i+1}) = a_3(t_{i+1}-t_i)^3 + a_2(t_{i+1}-t_i)^2 + a_1(t_{i+1}-t_i) + a_0 = p_{i+1}$$

$$r'(t_i) = a_1 = v_i$$

$$r'(t_{i+1}) = 3a_3(t_{i+1}-t_i)^2 + 2a_2(t_{i+1}-t_i) + a_1 = v_{i+1}$$

$$a_0 = p_i$$

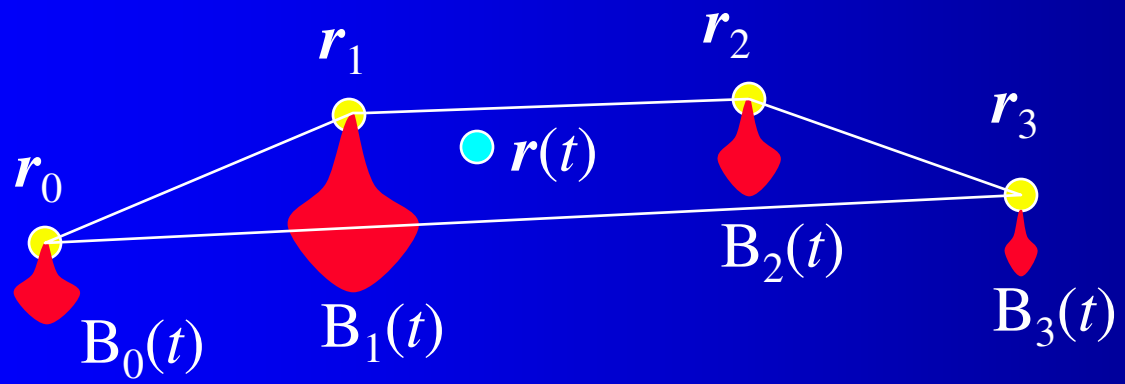
$$a_1 = v_i$$

$$a_2 = \frac{3(p_{i+1} - p_i)}{(t_{i+1} - t_i)^2} - \frac{v_{i+1} + 2v_i}{(t_{i+1} - t_i)}$$

$$a_3 = \frac{2(p_i - p_{i+1})}{(t_{i+1} - t_i)^3} + \frac{v_{i+1} + v_i}{(t_{i+1} - t_i)^2}$$

Bezier approximáció

- Keresd: $\boldsymbol{r}(t) = \Sigma\, \mathrm{B}_i(t)\, \boldsymbol{r}_i$
  - $\mathrm{B}_i(t)$: ne oszcilláljon
  - Konvex burok tulajdonság
  - $\mathrm{B}_i(t) \geq 0, \qquad \Sigma\, \mathrm{B}_i(t) = 1$



Bernstein polinomok

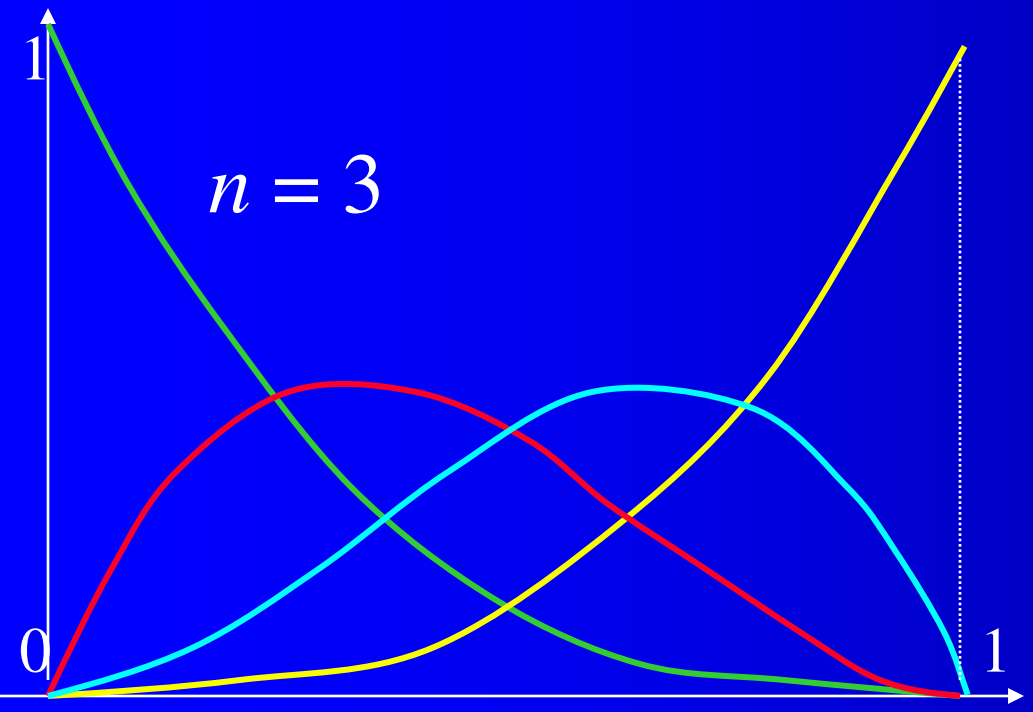
Newton féle binomiális tétel

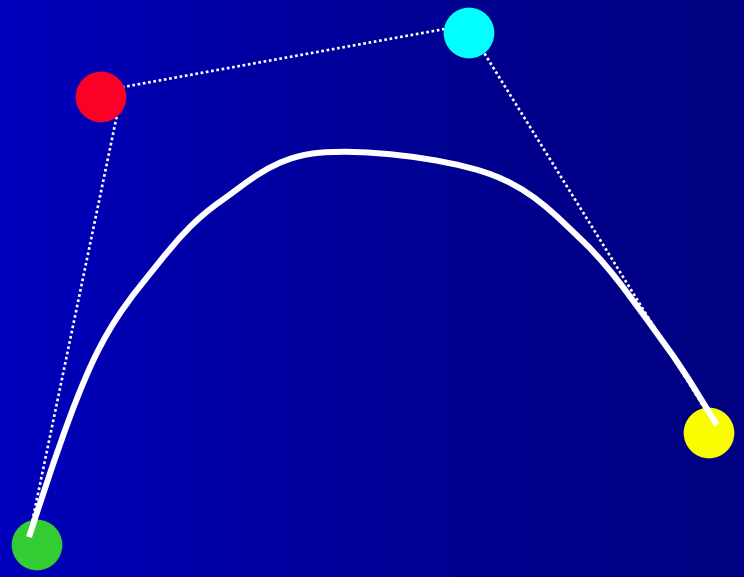
$$1^n = (t+(1-t))^n = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i}$$

$$B_i(t)$$

$B_i(t) \geq 0$, $\Sigma\, B_i(t) = 1$: OK

Bezier approximáció

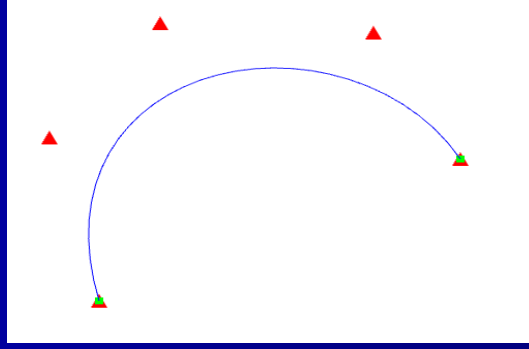




$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

$$\boldsymbol{r}(t) = \Sigma B_i(t) \boldsymbol{r}_i$$

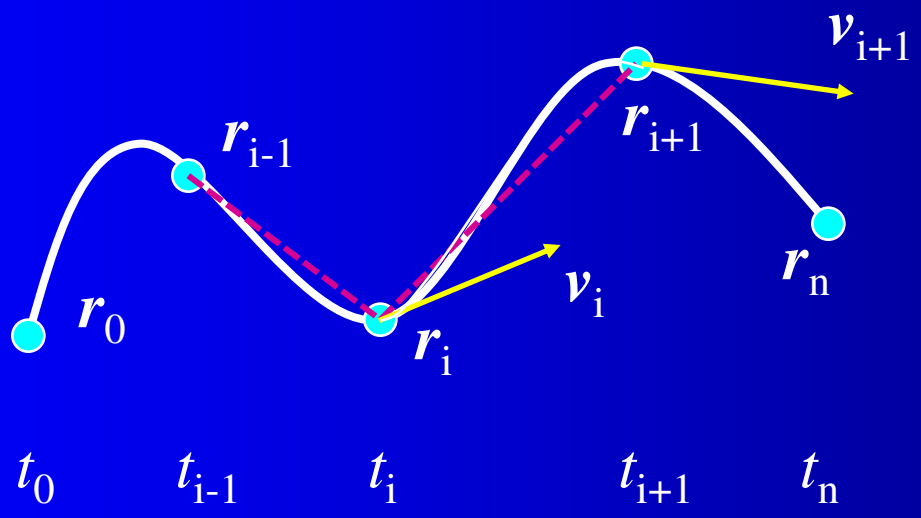
BezierCurve



```
class BezierCurve {
    vector<float3> cps;     // control points

    float B(int i, float t) {
        int n = p.size()-1; // n deg polynomial = n+1 pts!
        float choose = 1;
        for(int j = 1; j <= i; j++) choose *= (float)(n-j+1)/j;
        return choose * pow(t, i) * pow(1-t, np-i);
    }
public:
    void AddControlPoint(float3 cp) { cps.push_back(cp); }

    float3 r(float t) {
        float3 rr(0, 0);
        for(int i = 0; i < cps.size(); i++) rr += cps[i] * B(i,t);
        return rr;
    }
};
```

Catmull-Rom spline

Minden két vezérlőpont közé egy görbe szegmens
Simaság: a sebesség is legyen közös két egymás utánira



$$v_i = \frac{1}{2}\left(\frac{r_{i+1} - r_i}{t_{i+1} - t_i} + \frac{r_i - r_{i-1}}{t_i - t_{i-1}}\right)$$

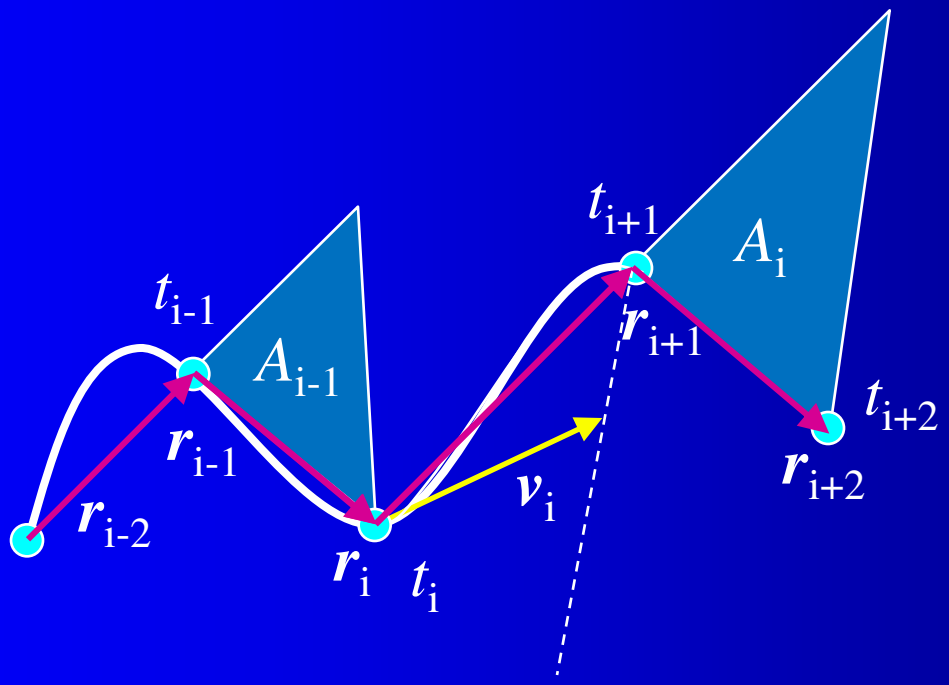
Egy görbeszegmens: Hermite interpoláció
Legeslegelső és legutolsó sebesség explicite

CatmullRom

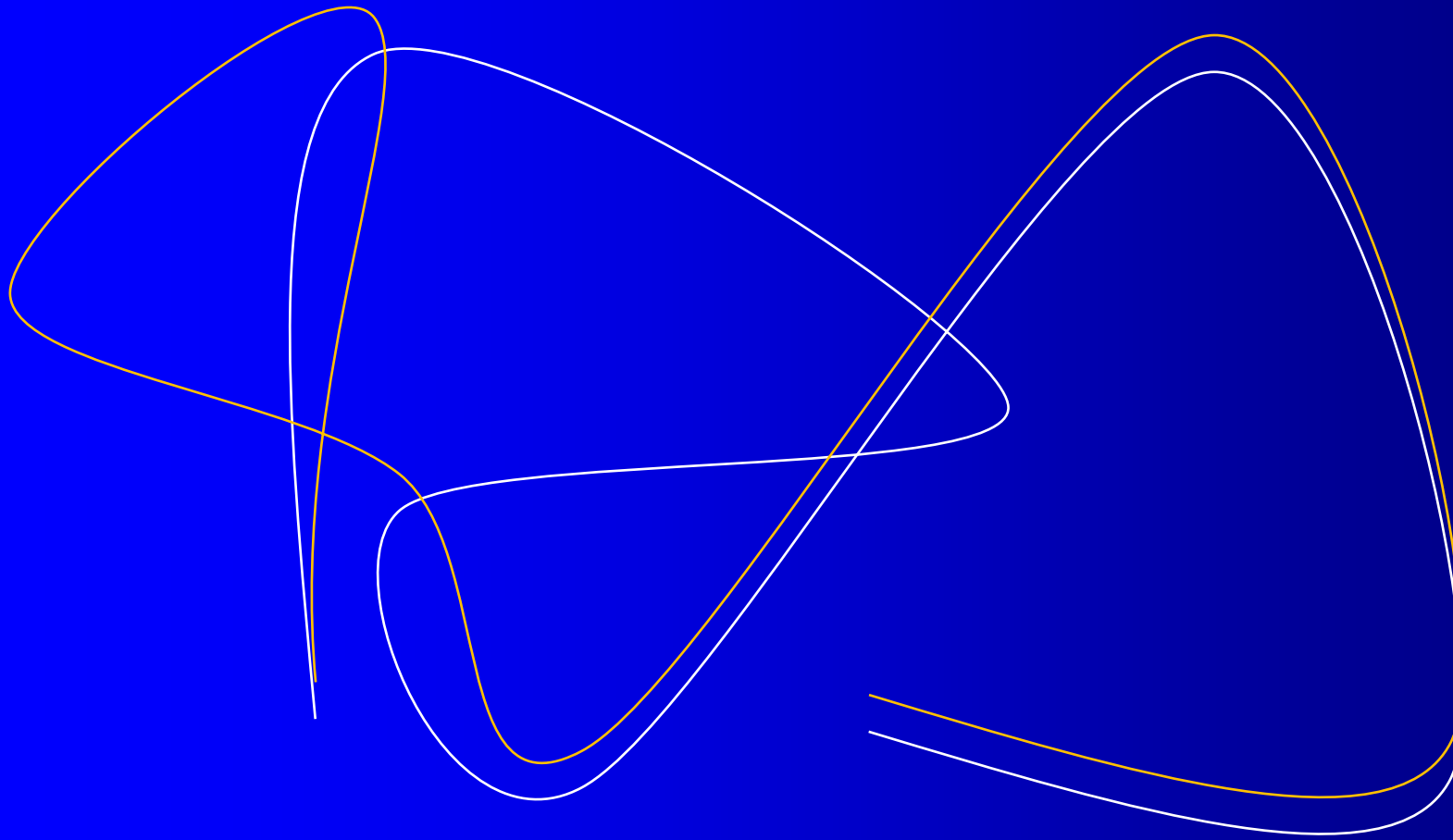
```
class CatmullRom {
    vector<float3> cps;      // control points
    vector<float>  ts;       // parameter (knot) values

    float3 Hermite(  float3 p0, float3 v0, float t0,
                     float3 p1, float3 v1, float t1,
                     float t ) {

        ???

    }

public:
    void AddControlPoint(float3 cp, float t) { ??? }

    float3 r(float t) {
        for(int i = 0; i < cps.size() - 1; i++) {
            if (ts[i] <= t && t <= ts[i+1]) return Hermite(…);
        }
    }
};
```

Renner Gábor spline

A sebességeket és az időket találjuk ki abból, hogy görbén a mozgás legyen egyenletes:



1. $t_{i+1} - t_i \approx$ ívhossz
2. $\boldsymbol{v}_i = [A_i(\boldsymbol{r}_i - \boldsymbol{r}_{i-1}) + A_{i-1}(\boldsymbol{r}_{i+1} - \boldsymbol{r}_i)]/(A_i + A_{i-1})$
3. $\boldsymbol{v}_i$ normalizálása



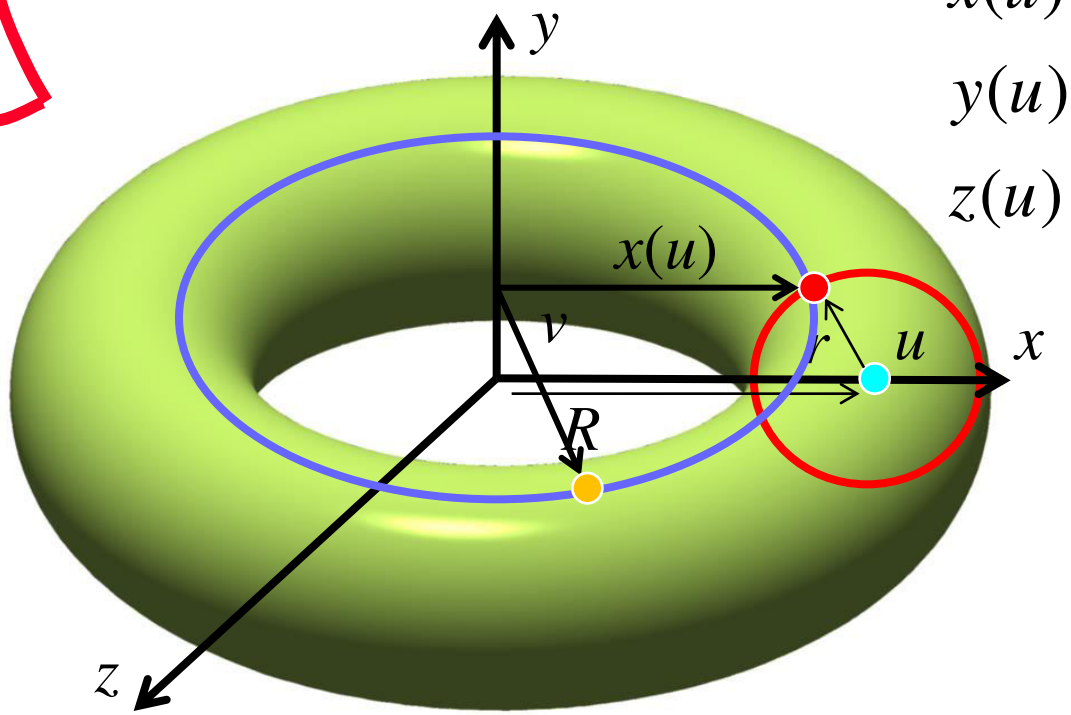
Felületek

- Felület a 3D tér 2D részhalmaza:
  - Koordináták kielégítenek egy egyenletet
  - **implicit**: $f(x, y, z) = 0$
    - gömb: $(x - x0)^2 + (y - y0)^2 + (z - z0)^2 - R^2 = 0$
  - **parametrikus**: $x = x(u,v),\ y = y(u,v),\ z = z(u,v),$ $u,v \in [0,1]$
    - gömb
      $$x = x0 + R \cos 2\pi u \ \sin \pi v$$
      $$y = y0 + R \sin 2\pi u \sin \pi v$$
      $$z = z0 + R \cos \pi v \qquad u,v \in [0,1]$$
  - **explicit** (magasságmező):
    $$z = h(x,y)$$

Tórusz (donut)

$$\boldsymbol{r}(u,v) = \boldsymbol{r}_v(u)$$





$$x(u) = R + r\cos(u)$$

$$y(u) = r\sin(u)$$

$$z(u) = 0$$

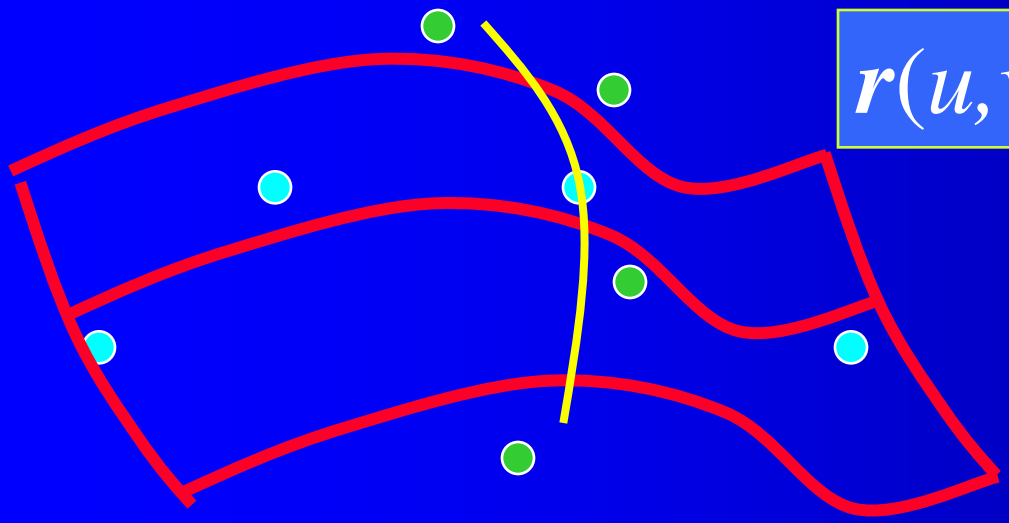
$$x(u,v) = x(u)\cos(v) = (R + r\cos(u))\cos(v)$$

$$y(u,v) = y(u) = r\sin(u)$$

$$z(u,v) = x(u)\sin(v) = (R + r\cos(u))\sin(v)$$

Szabadformájú felület: $\mathbf{r}(u,v)$

- Definíció vezérlőpontokkal



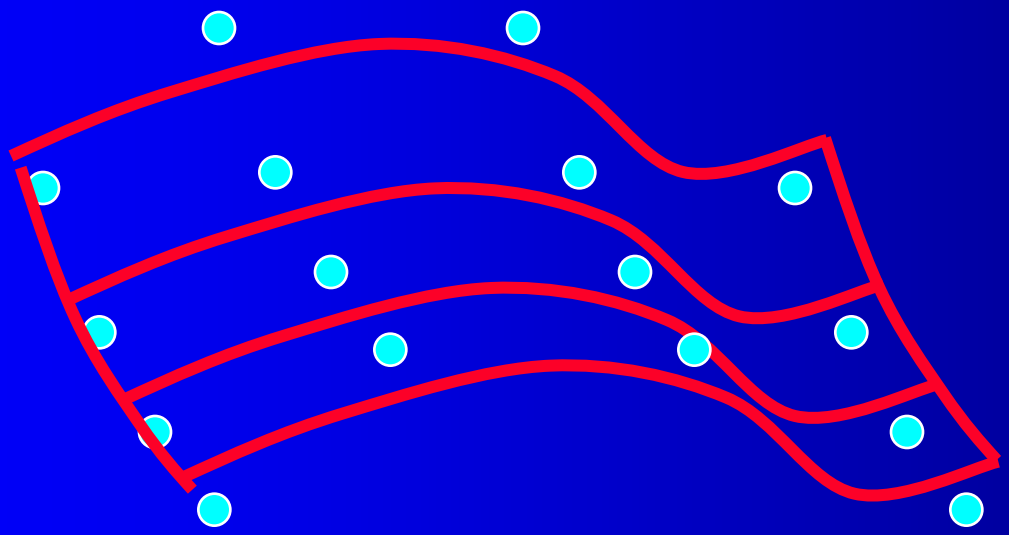
$$\mathbf{r}(u,v) = \Sigma\,\Sigma\, B_i(u)\, B_j(v)\, \mathbf{r}_{i,j}$$

$$\mathbf{r}(u,v) = \mathbf{r}_v(u) = \Sigma\, B_i(u)\, \mathbf{r}_i(v)$$

$$\mathbf{r}_i(v) = \Sigma\, B_j(v)\, \mathbf{r}_{i,j}$$

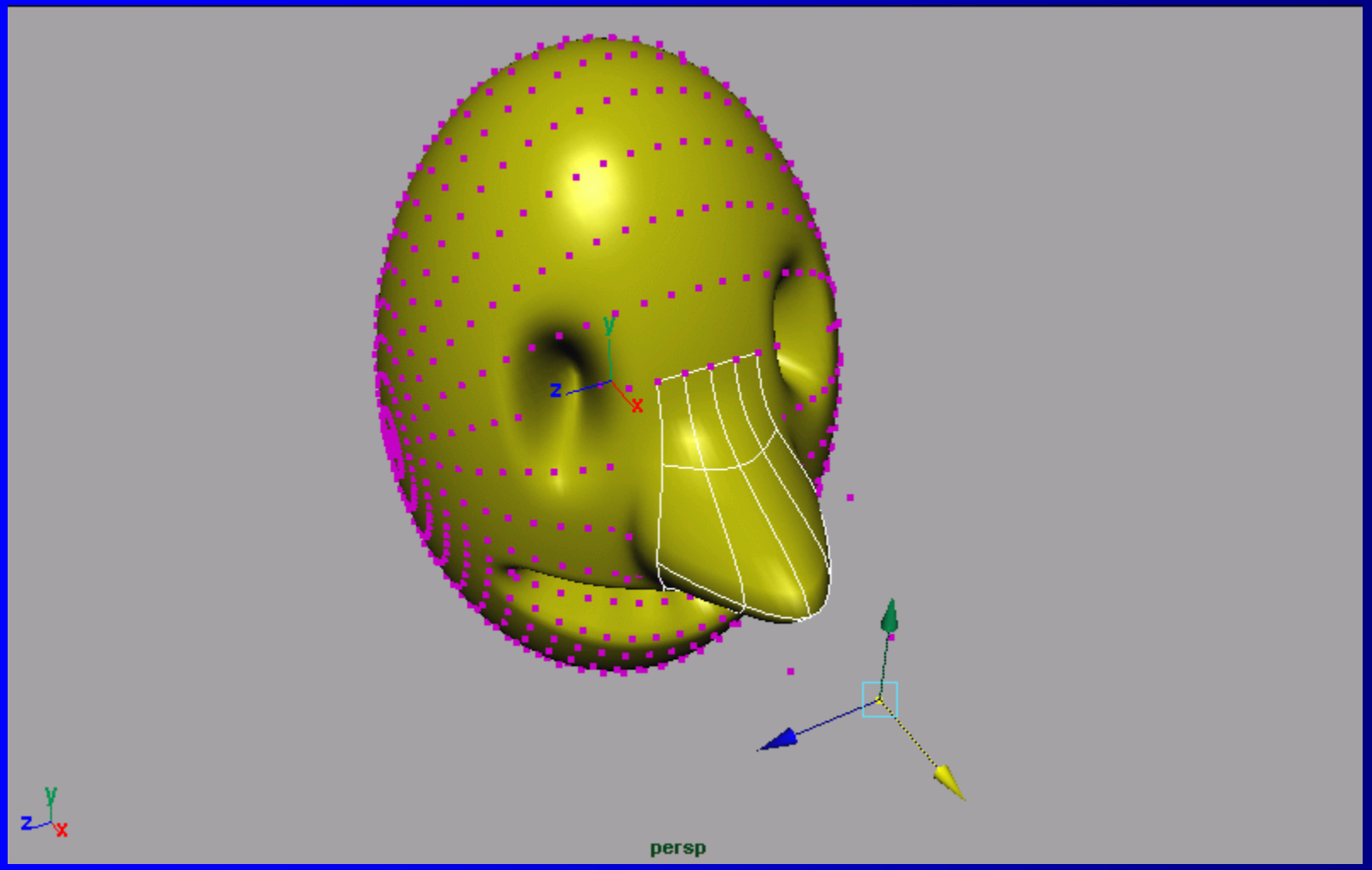
Szabadformájú felületek

Vezérlő háló:

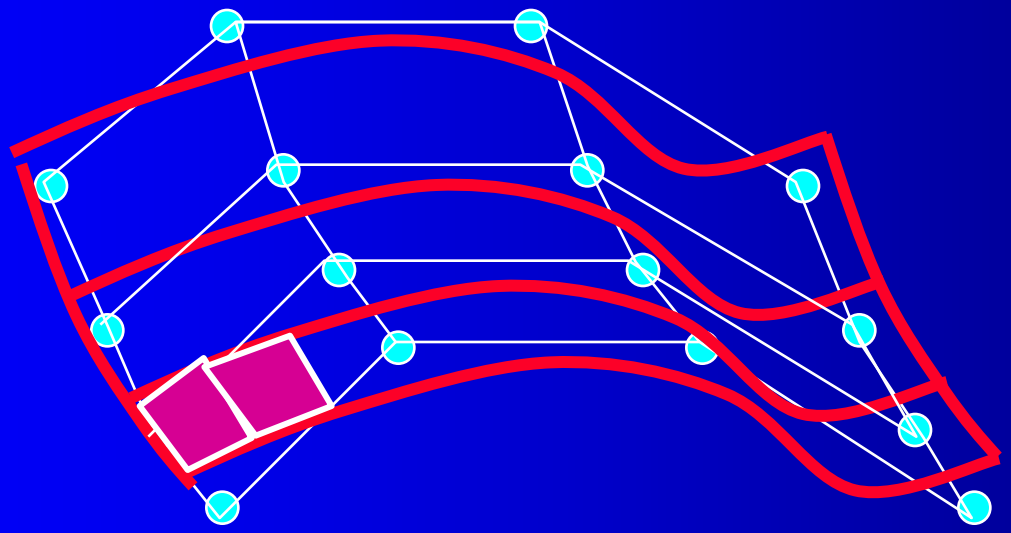


$$\boldsymbol{r}(u,v) = \sum \sum B_{i,j}(u,v)\, \boldsymbol{r}_{i,j}$$
$$= \sum \sum B_i(u)\, B_j(v)\, \boldsymbol{r}_{i,j}$$

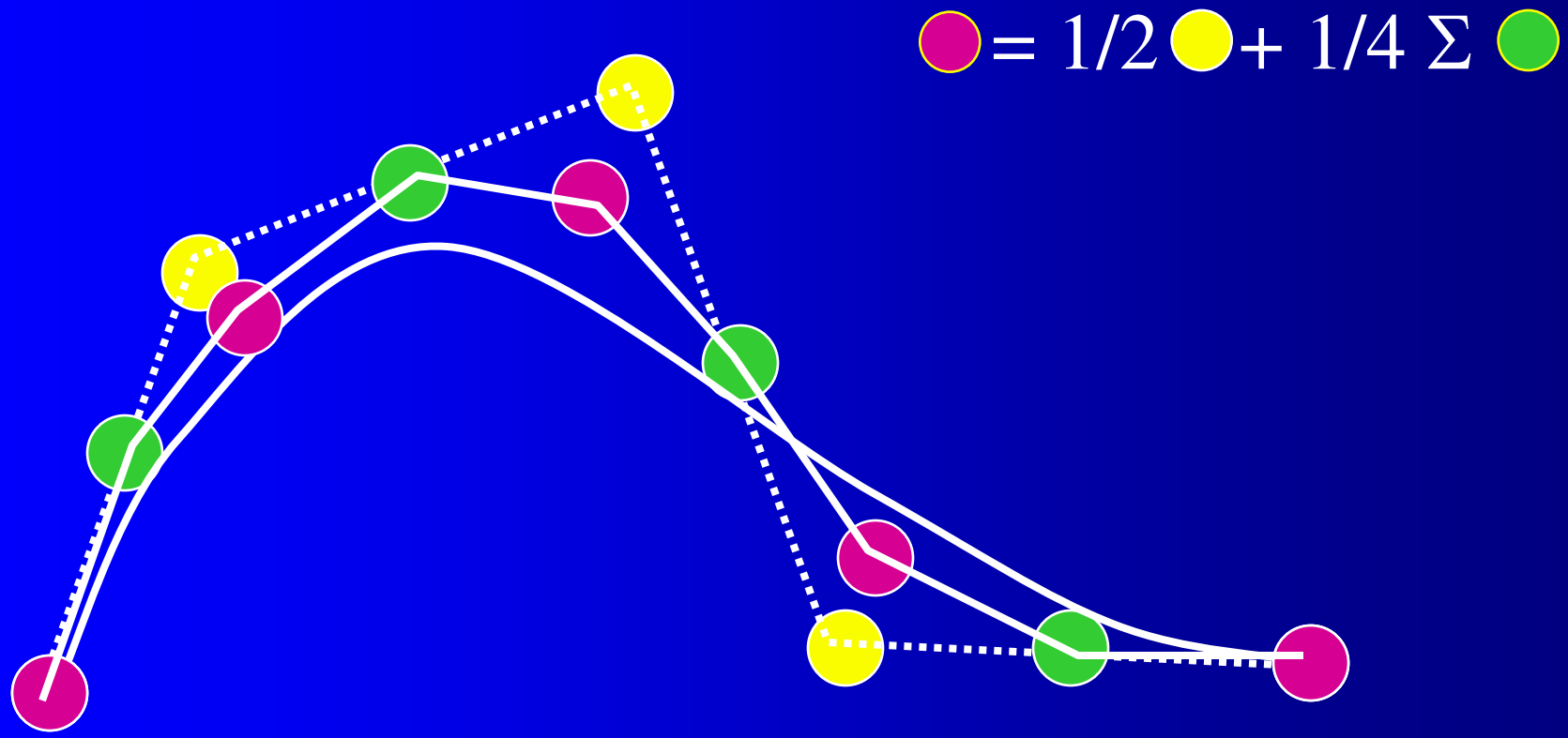
Vezérlőpontok módosítása



Poligonháló finomítása



Subdivision görbék



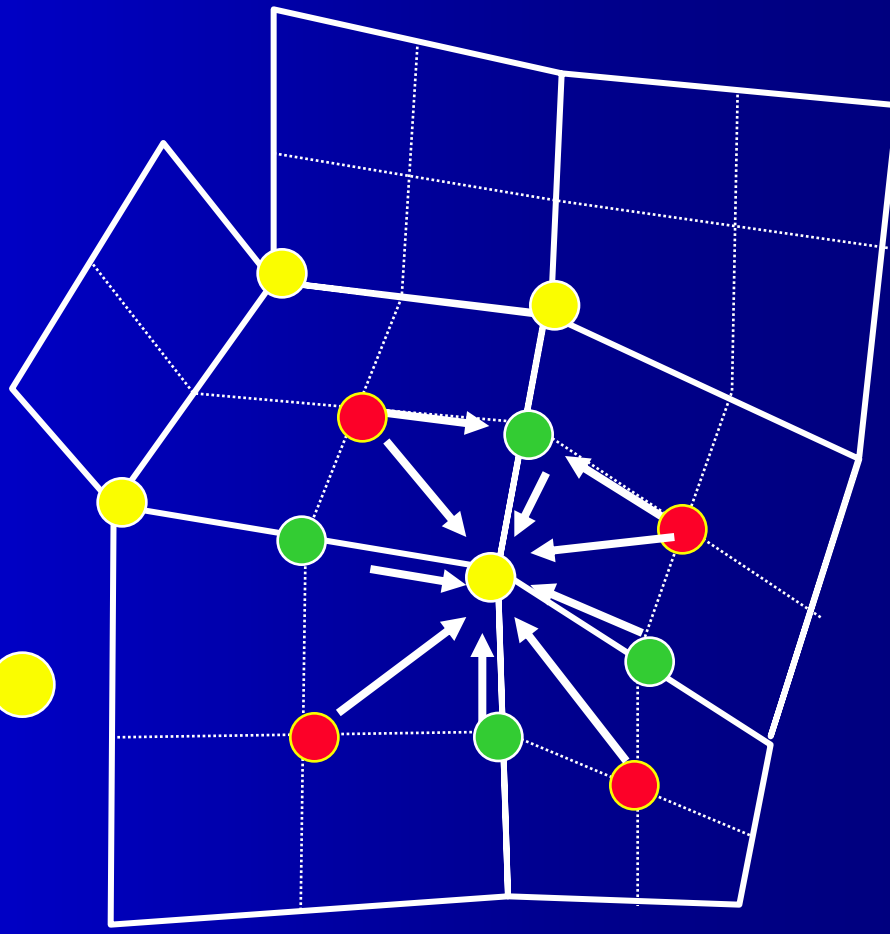
Catmull-Clark subdivision felület

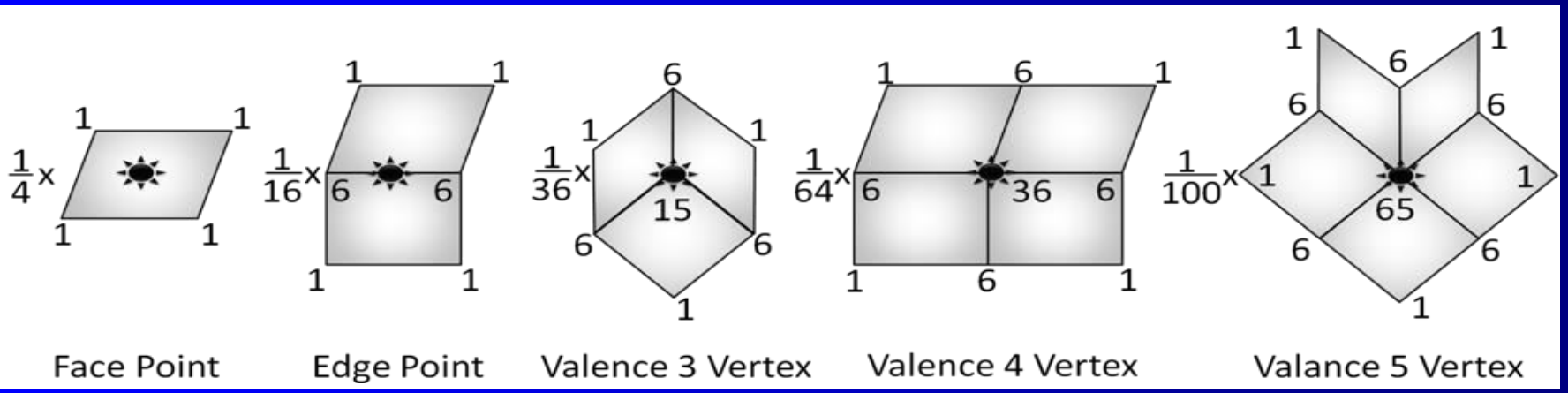
● = 1/4Σ ●

i = 1/2Σ ●

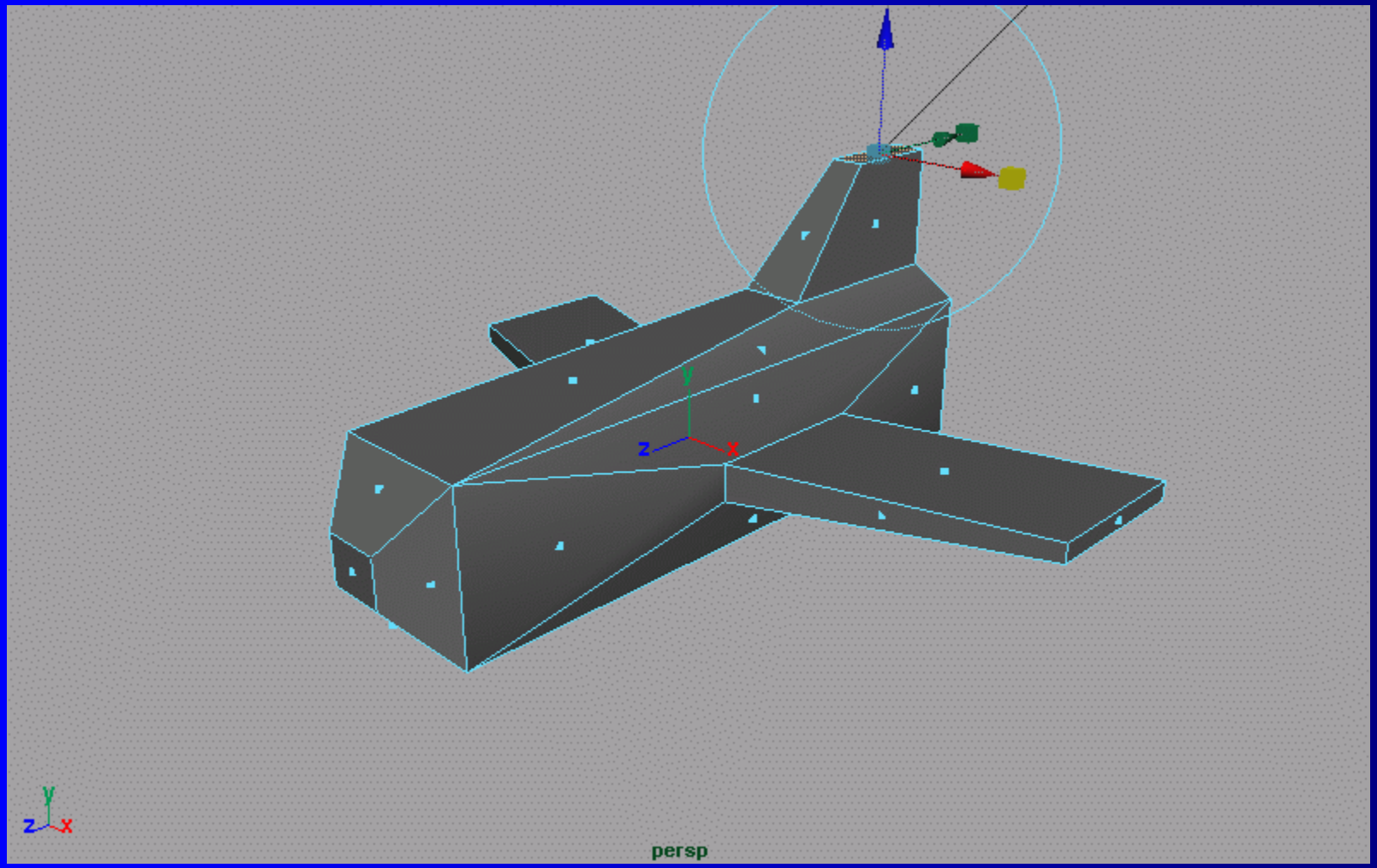
● = $1/v^2\Sigma$ ● $+2/v^2\Sigma$ *i* $+(v-3)/v$ ●

● = 1/4Σ ● + 1/2 *i*

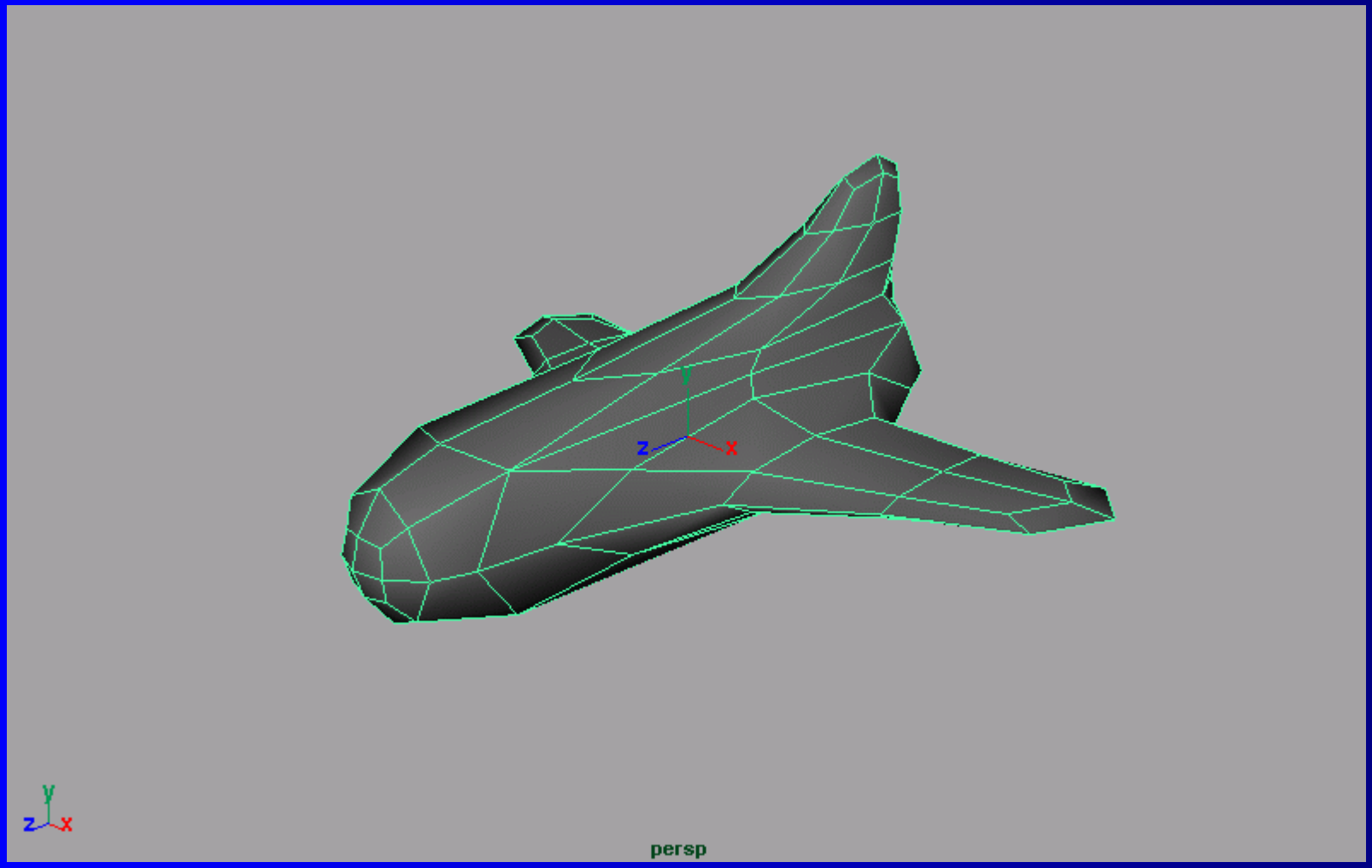




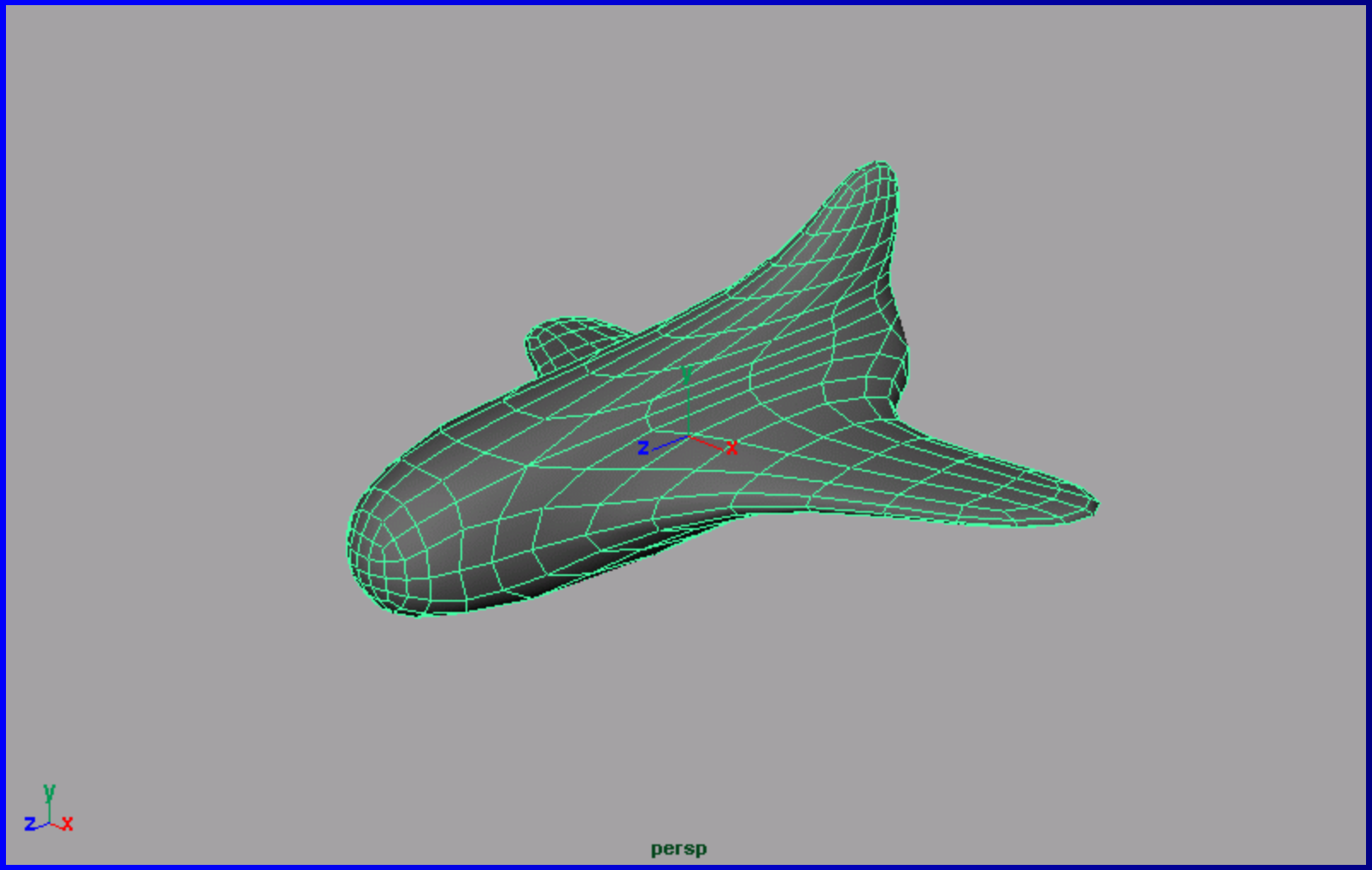
Durva poligon modell



Subdivision: 1. szint

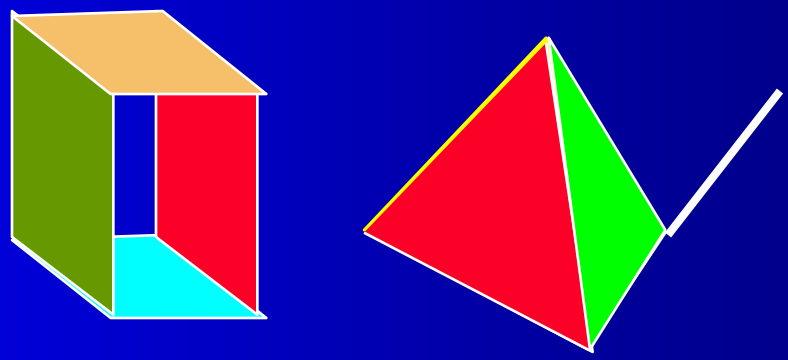


Subdivision: 2. szint



Testek

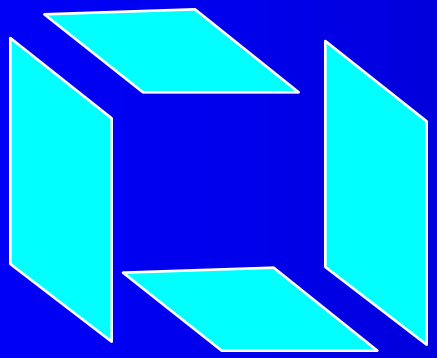
- Ellenpéldák

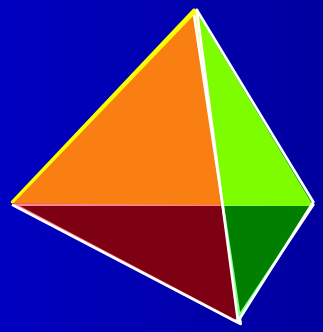


- Érvényes test:
  - Nem lehet alacsonyabb dimenziójú elfajulás
- Érvényes testet eredményező módszerek
  - 2.5 dimenziós eljárások
  - B-rep
  - Constructive Solid Geometry

B-rep

- Test = határoló felületek



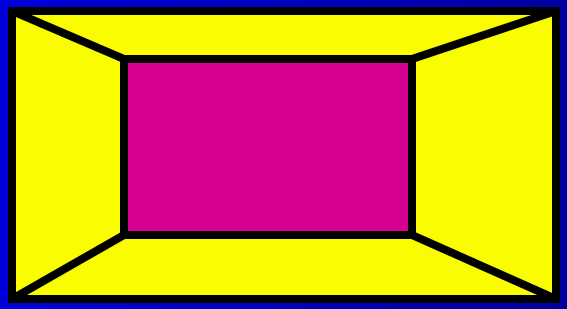


- Topológiai érvényesség (Euler tétel ha egy darabból áll és nincsenek benne lyukak) :

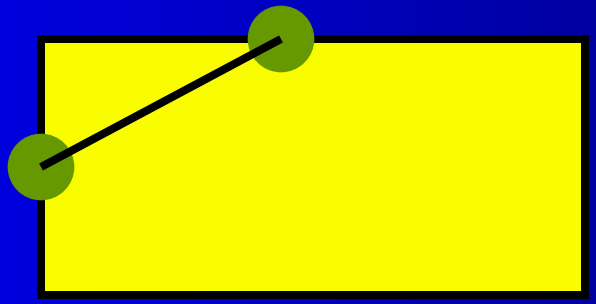
csúcsok + lapok= élek + 2

Euler operátorok

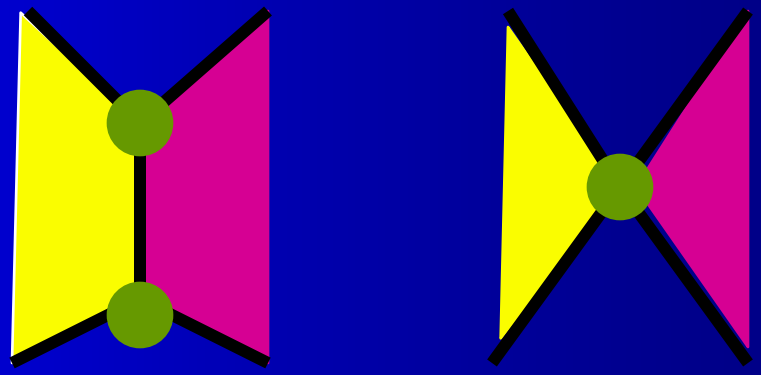
- Lap kihúzás



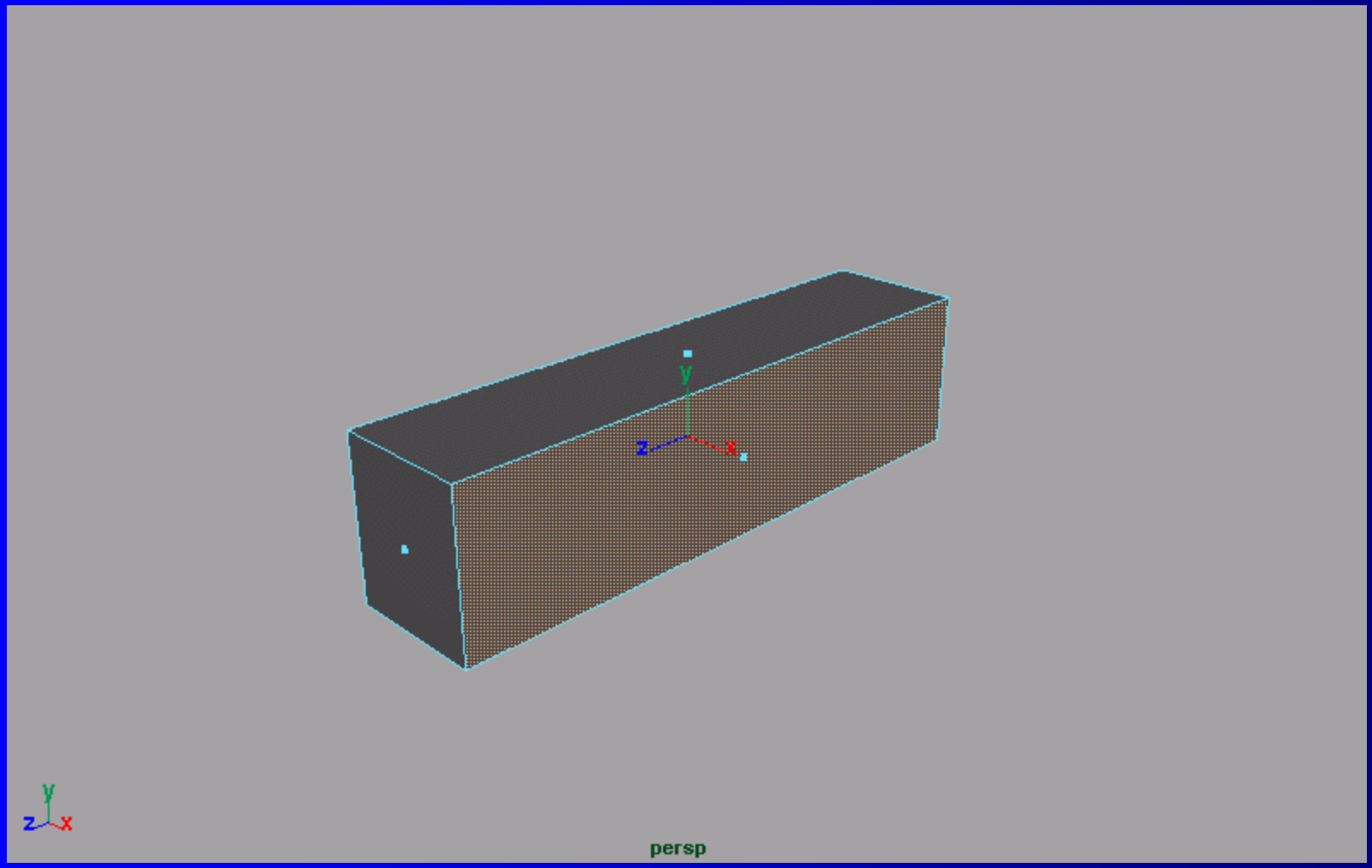
- Lap felvágás



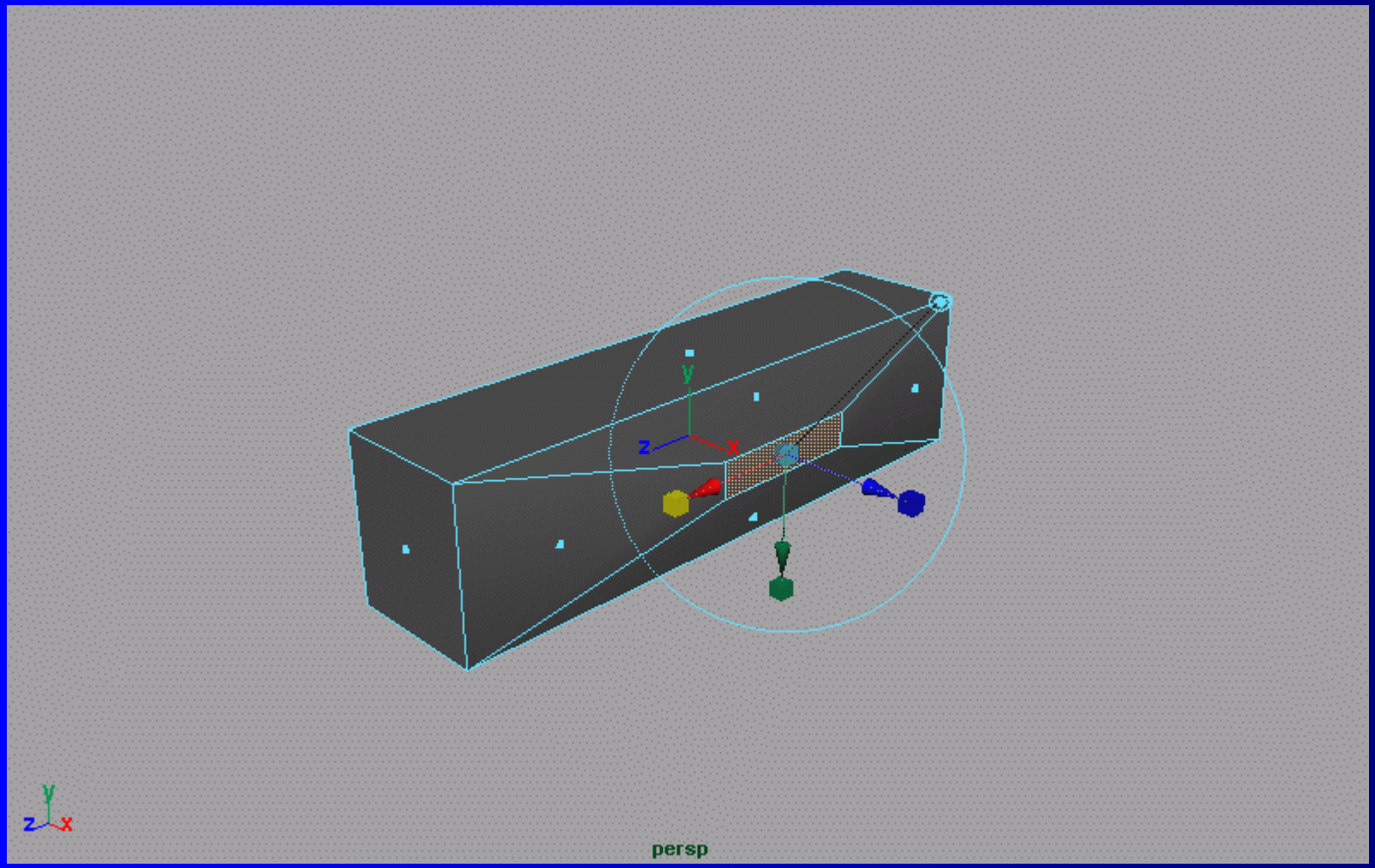
- Él törlés
- Csúcs szétvágás



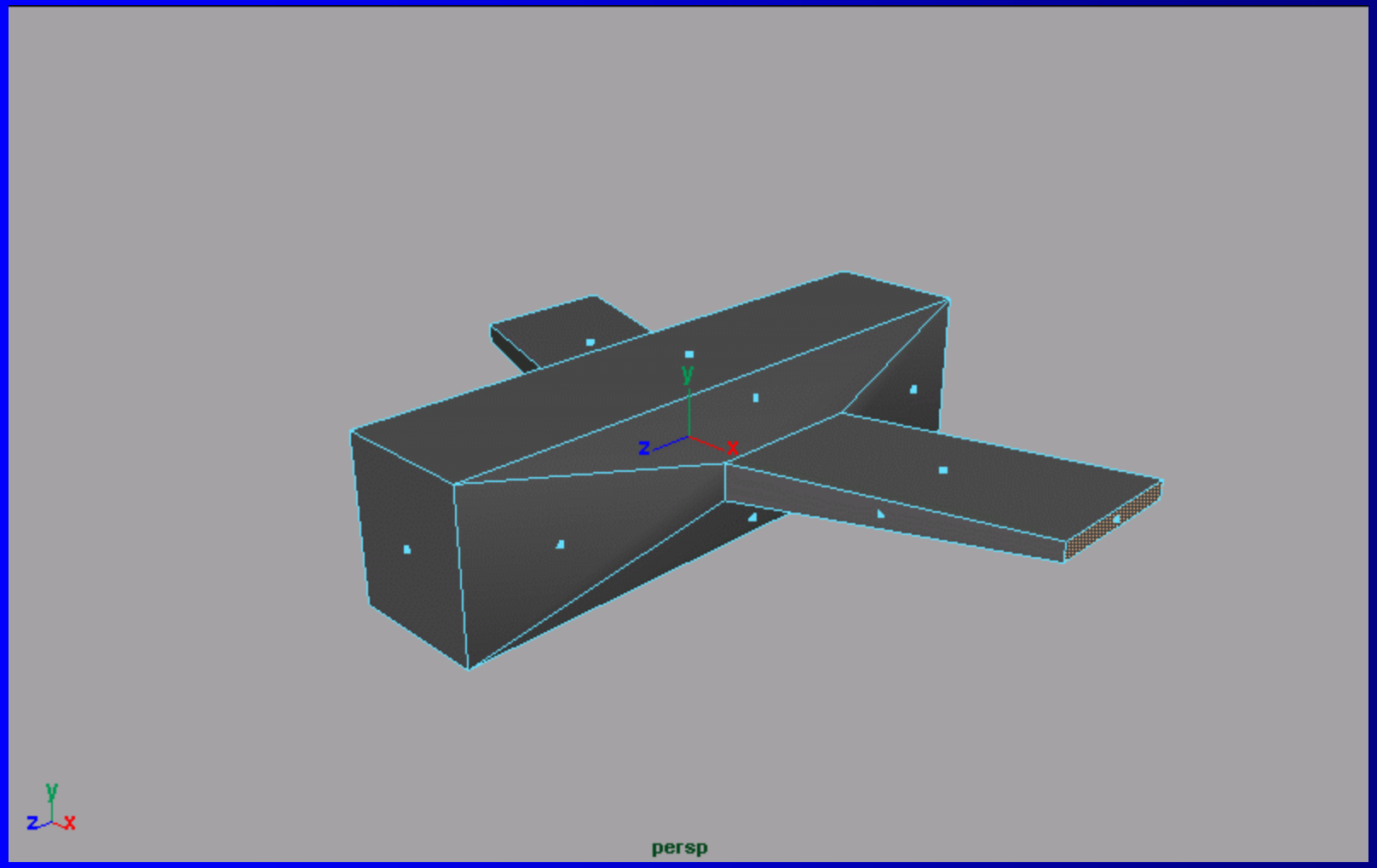
Kezdeti test



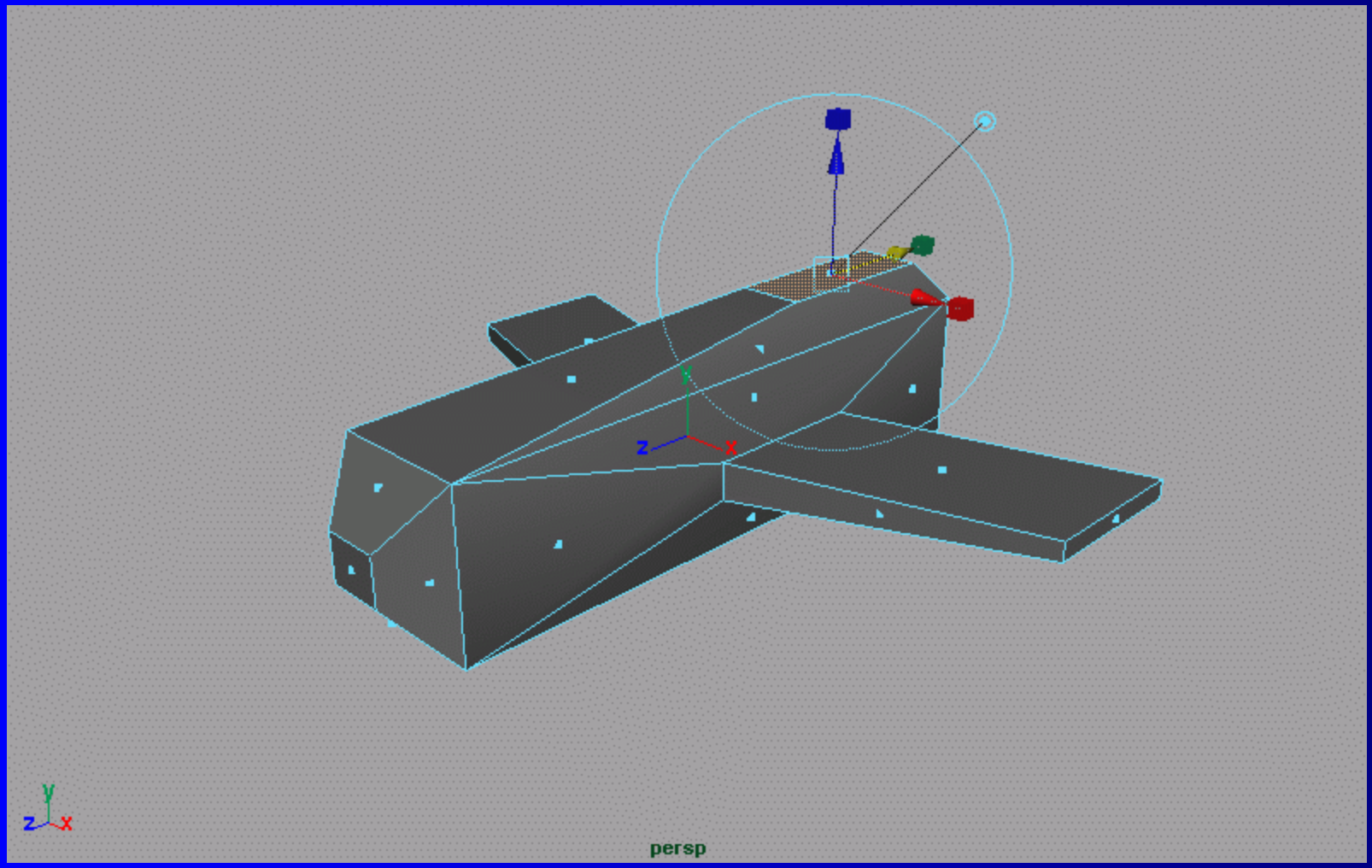
Lap kihúzás



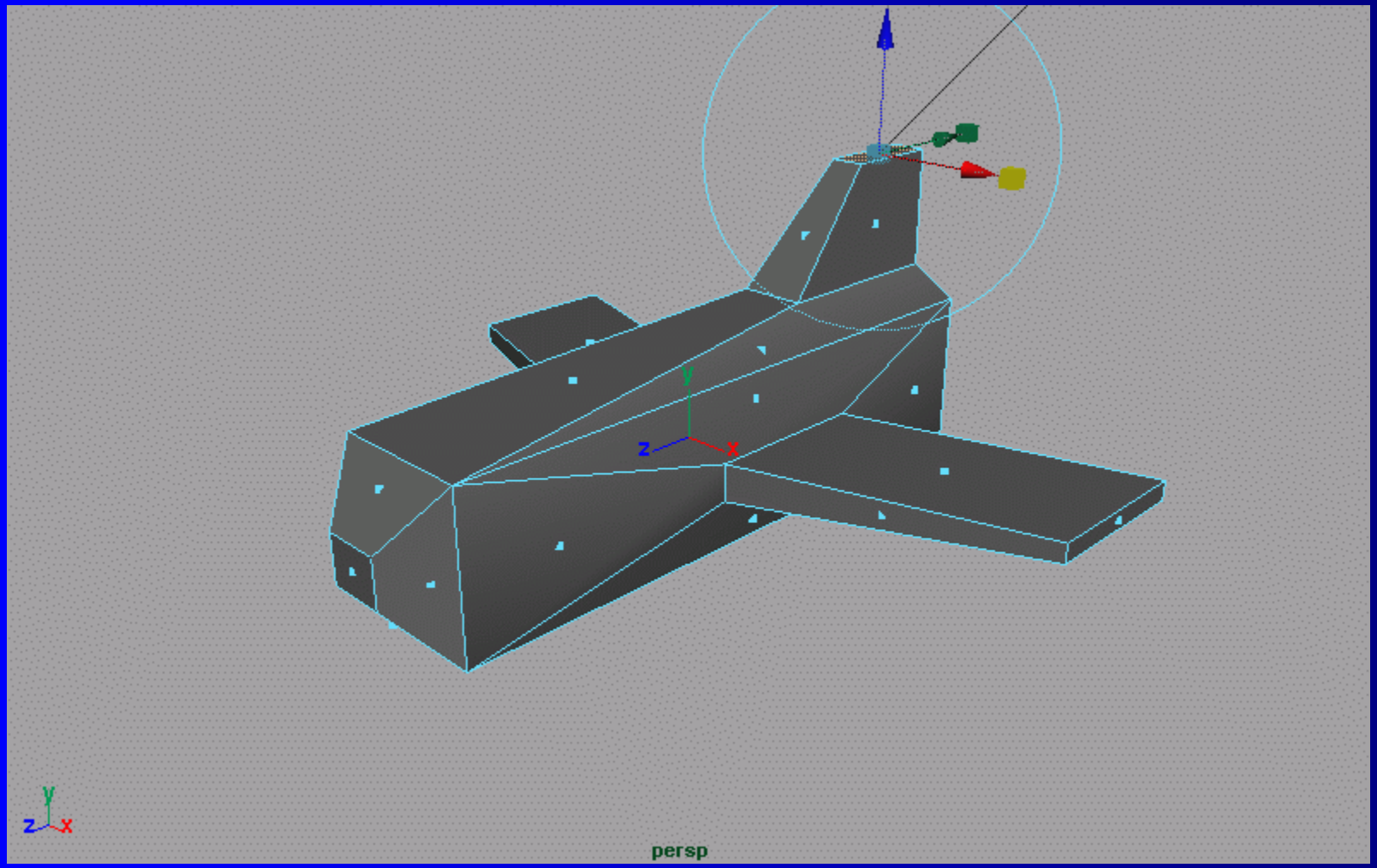
Lap kihúzás



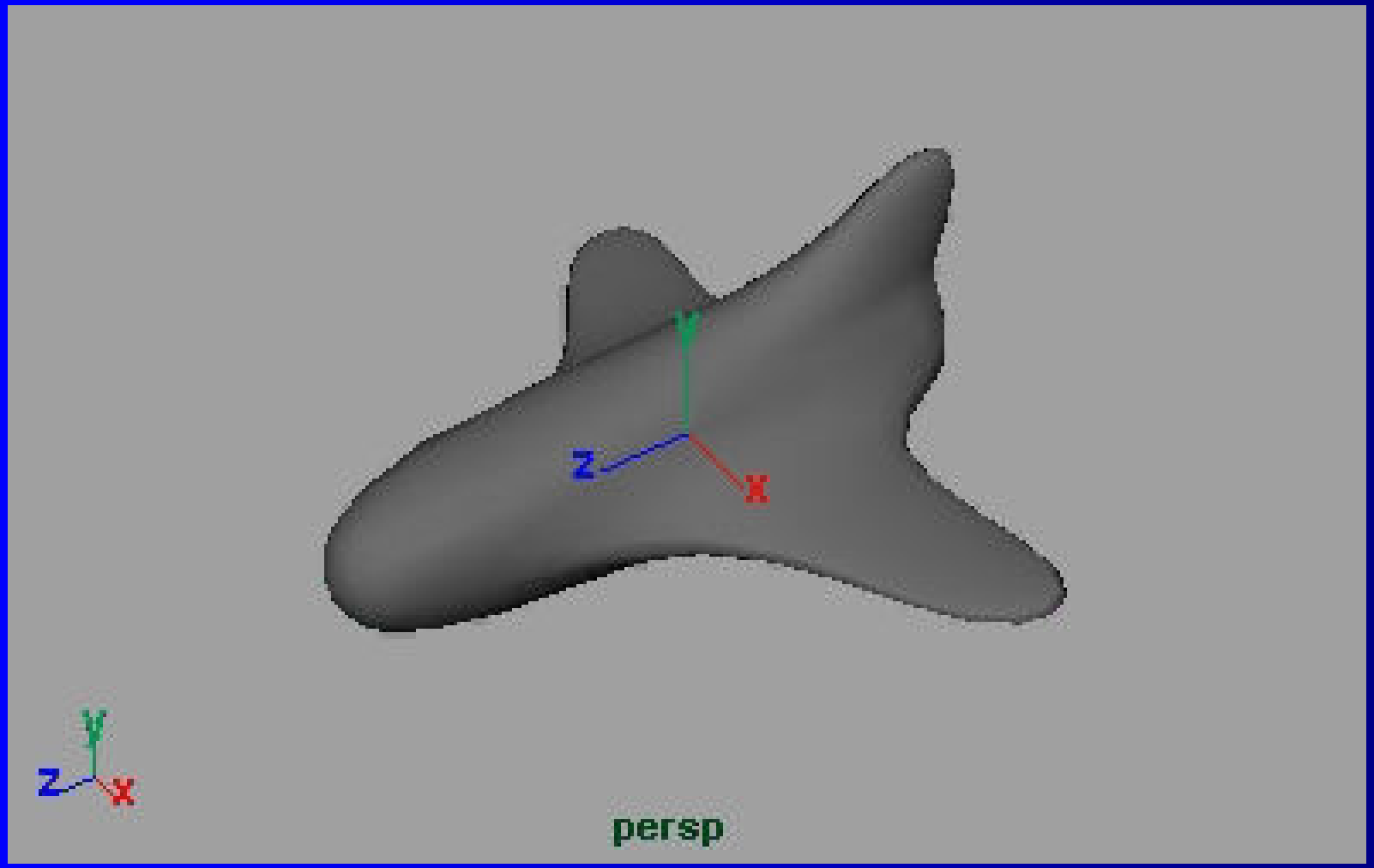
Még egy



Még mindig

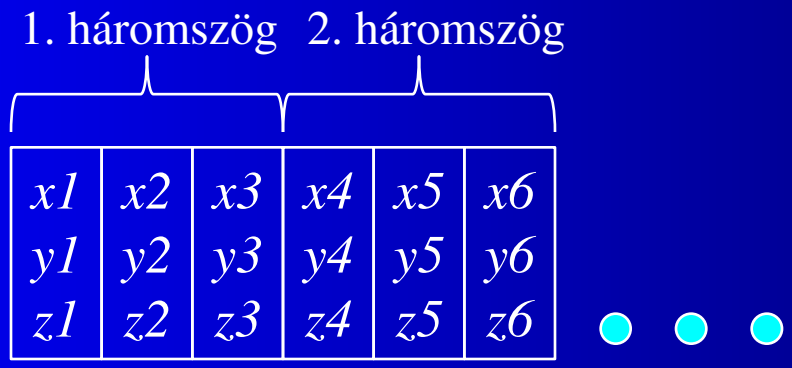


Subdivision simítás

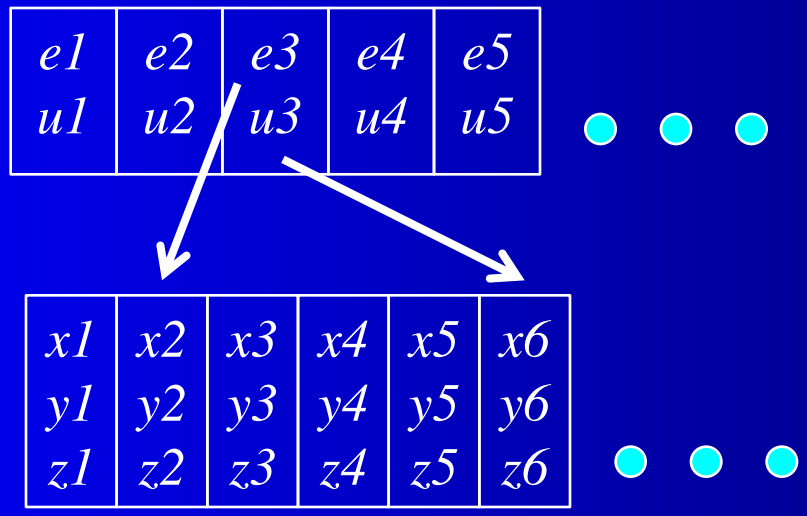


Poligon modell: leves

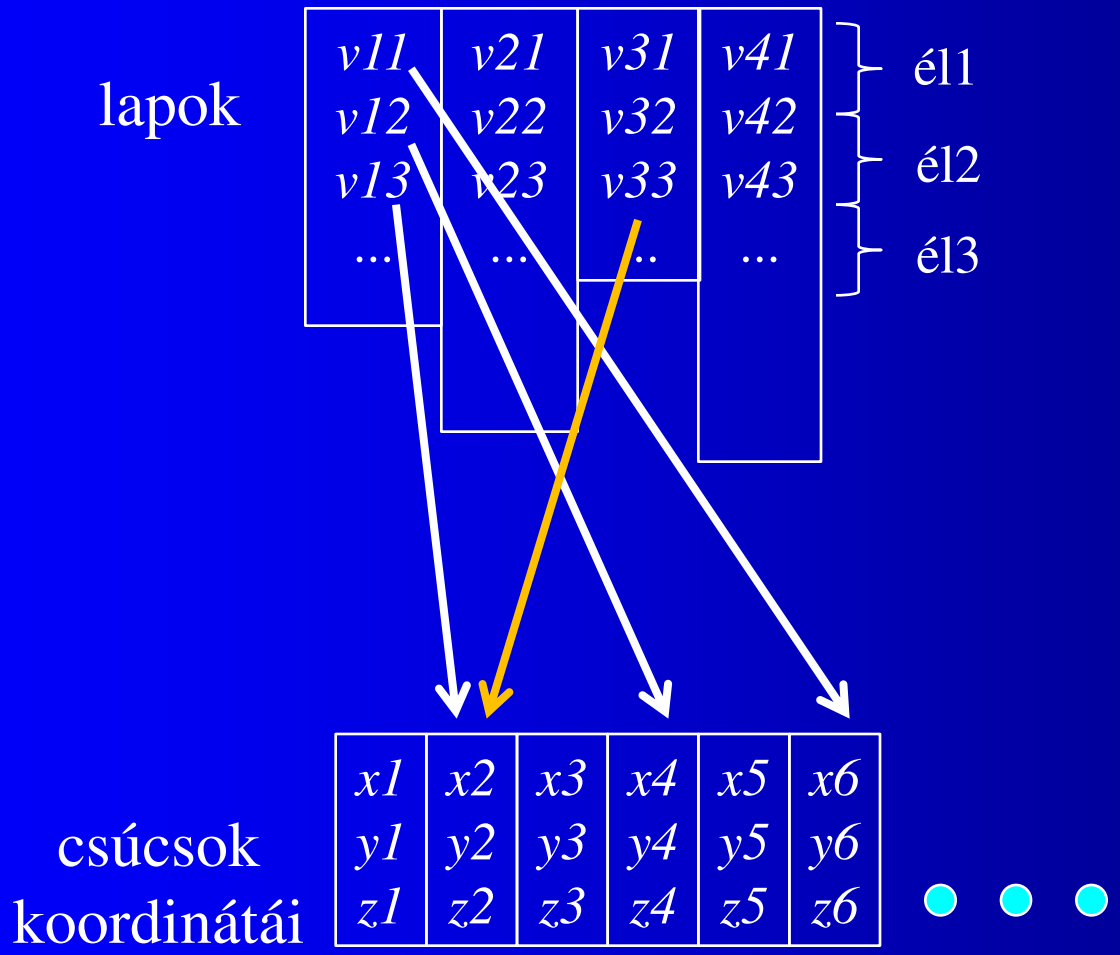
Háromszög leves



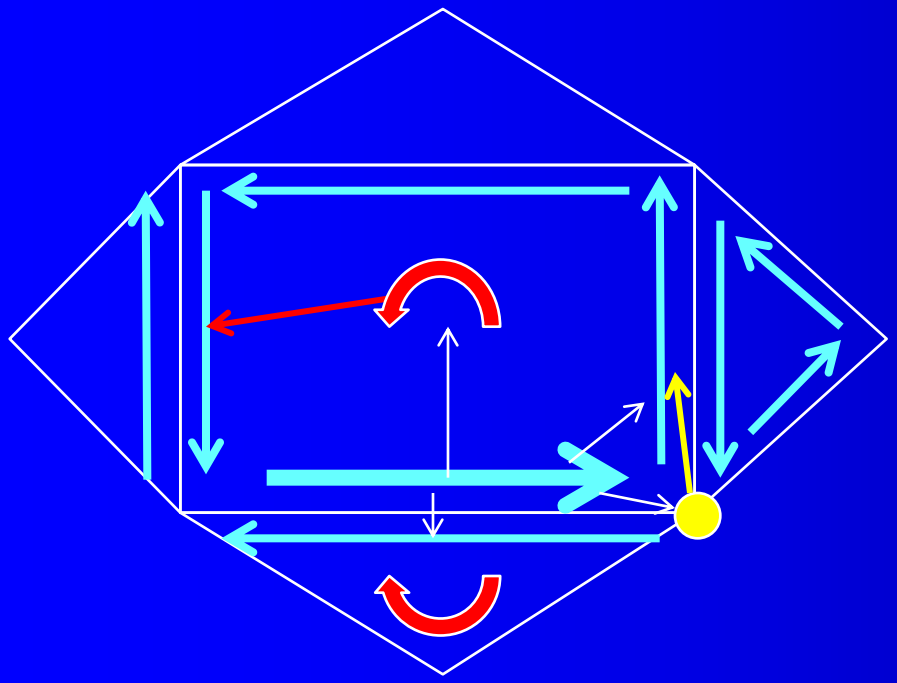
Sokszögek



Poligon modell: hierachikus



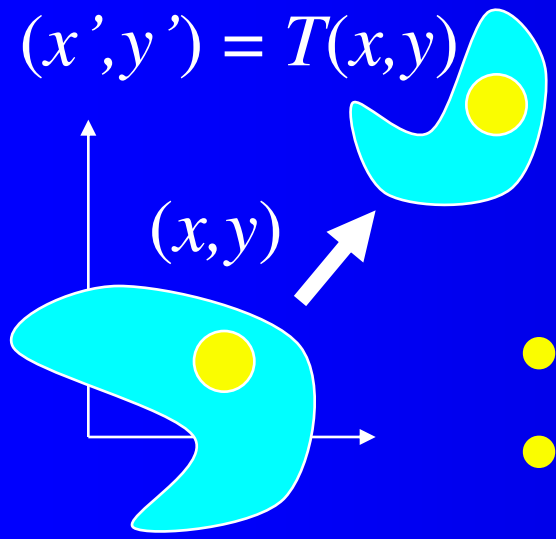
Poligon modell: half edge
(half edge, winged edge, radial edge)



```
struct hedge {
    vertex* endVertex;
    hedge* pairHedge;
    face* borderedFace;
    hedge* nextHedge;
};

struct vertex {
    float x, y, z;
    hedge* oneEmanatingHedge;
};

struct face {
    hedge* oneBorder;
};
```

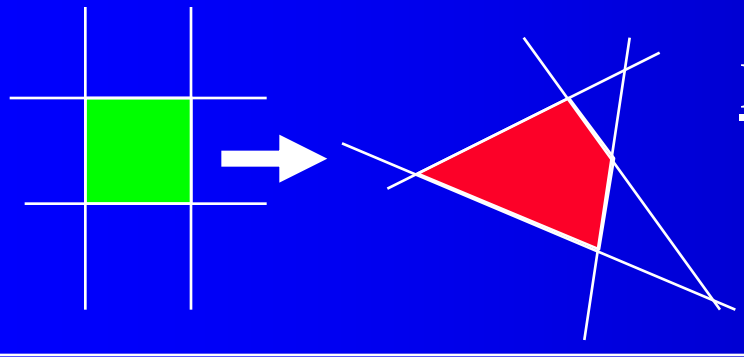
Transzformációk

Szirmay-Kalos László



Transzformációk

- Tönkre tehetik az egyenletet
- Korlátozzuk a transzformációkat és az alakzatokat úgy, hogy invariáns legyen
  - Pont, egyenes (szakasz), sík (poligon)
- **Affin transzformációk**
  - Párhuzamos egyenes tartó
  - Descartes koordinátákban lineáris



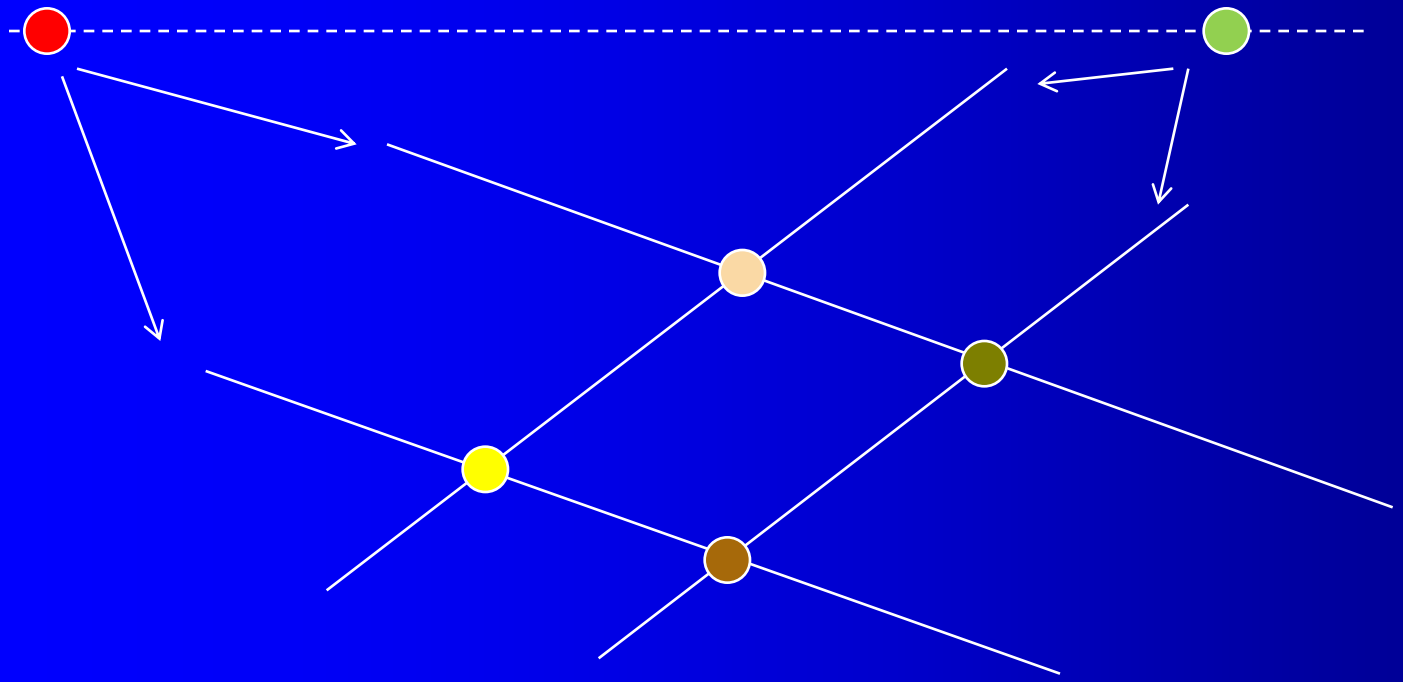
Homogén lineáris transzformációk

Egyenest egyenesbe
Homogén koordinátákban lineáris

Perspektíva



Euklideszi → Projektív sík



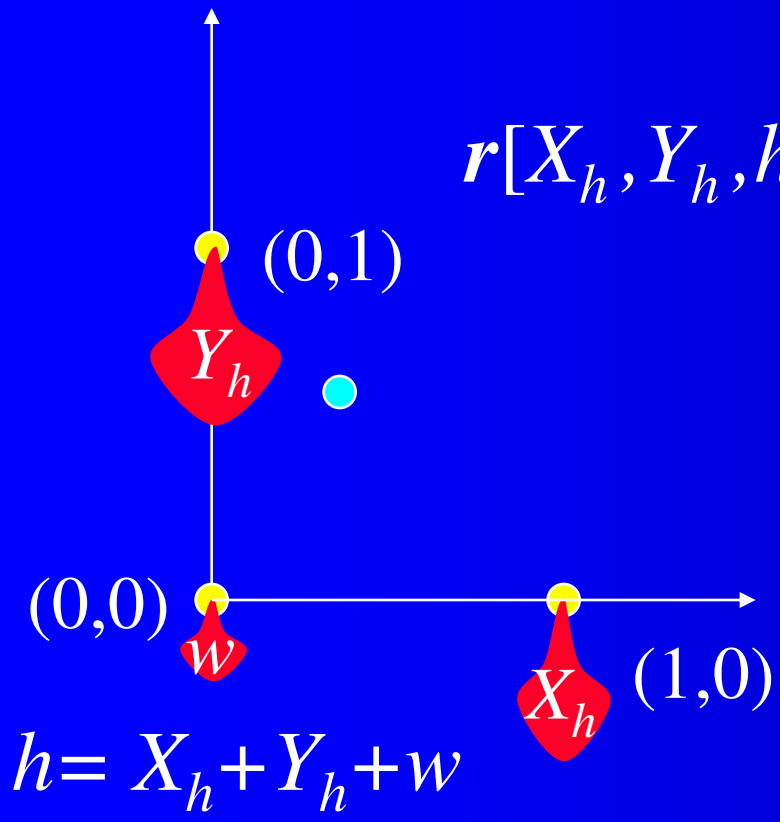
- Két pont meghatároz egy egyenest.
- Egy egyenesnek van legalább két pontja.
- ~~Ha *a* egy egyenes, A pedig egy, nem az egyenesen lévő pont, akkor egyetlen olyan egyenes létezik, amely áthegy A-n és nem metszi a-t.~~

→

- Két pont meghatároz egy egyenest.
- Egy egyenesnek van legalább két pontja.
- **Két egyenes mindig egy pontban metszi egymást**.

Homogén koordináták (2D)

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

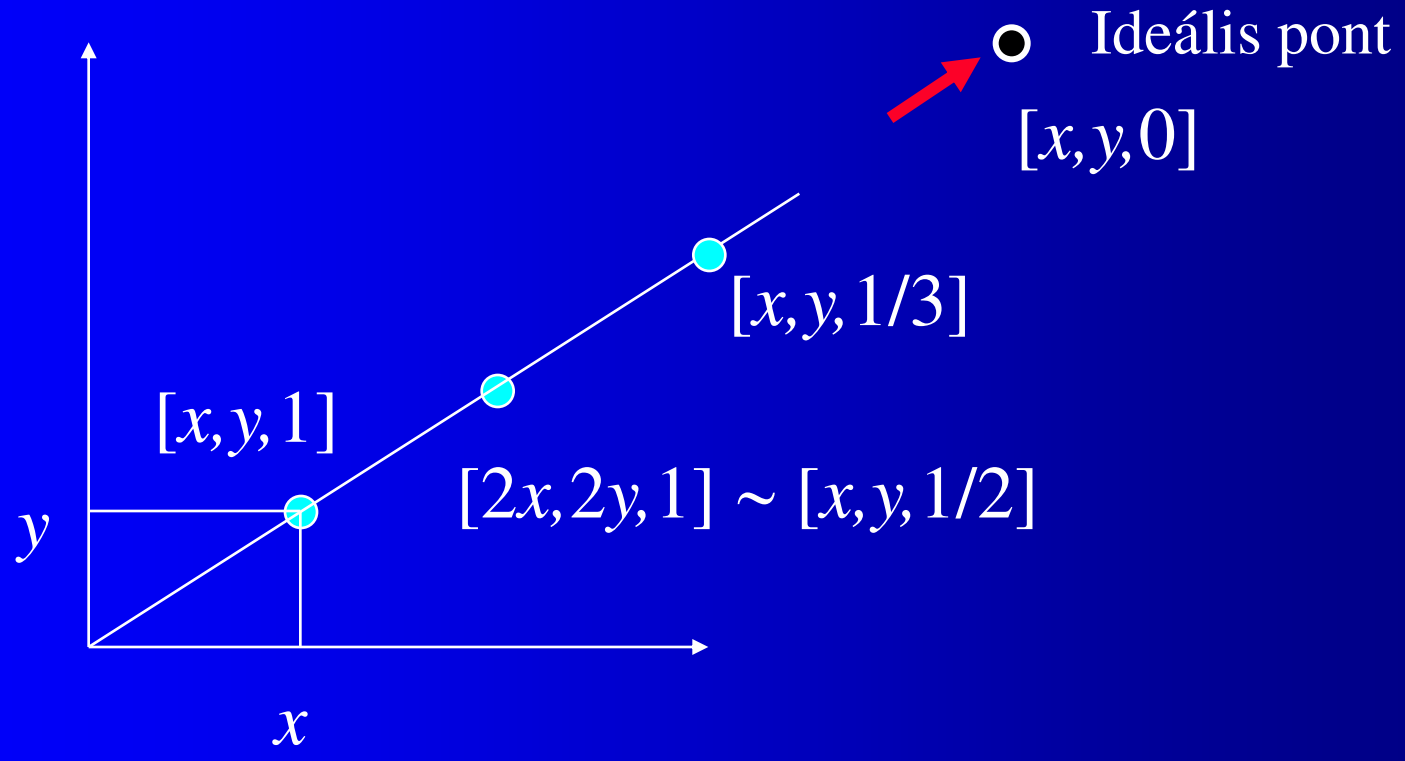


$$\boldsymbol{r}[X_h, Y_h, h] = \frac{X_h \cdot (1,0) + Y_h \cdot (0,1) + w \cdot (0,0)}{h}$$

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

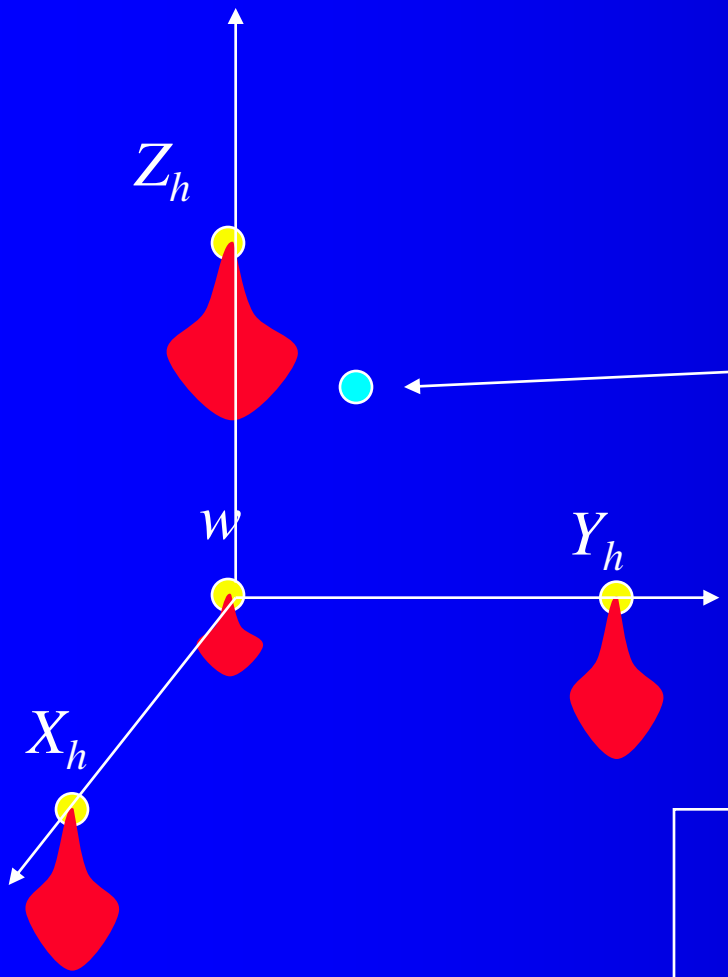
$$x = \frac{X_h}{h} \qquad y = \frac{Y_h}{h}$$

Homogén koordináták ideális pontokhoz: h=0



Euklideszi sík+ ideális pontok = projektív sík

Homogén koordináták (3D)



Teljes súly: $h = X_h + Y_h + Z_h + w$

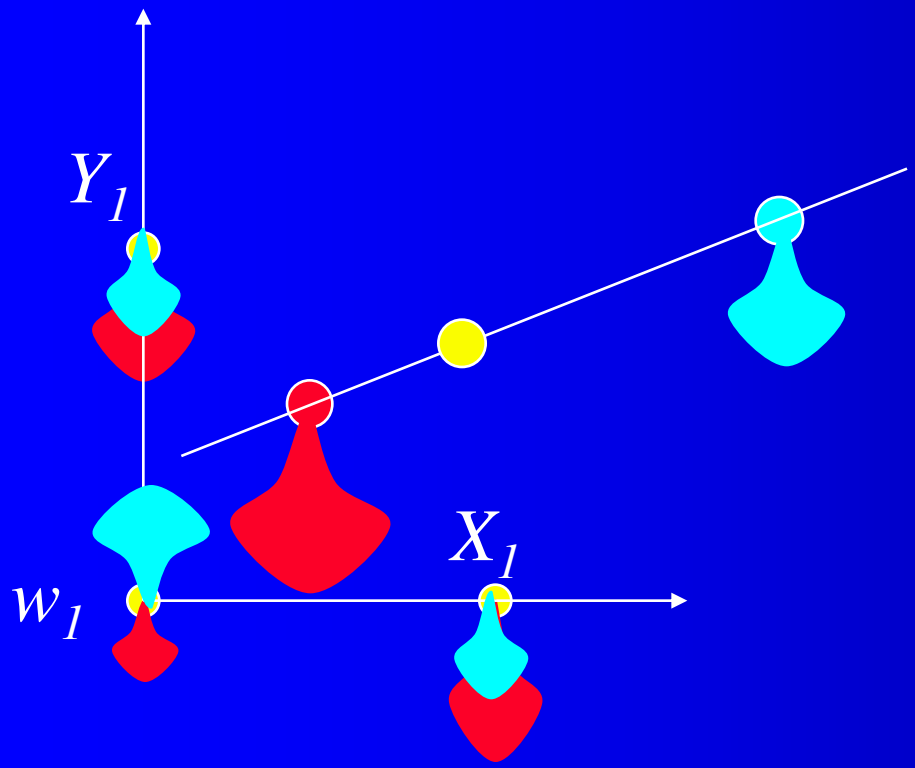
Súlypont:

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

Nem lehet mind egyszerre zérus

$$x = \frac{X_h}{h} \qquad y = \frac{Y_h}{h} \qquad z = \frac{Z_h}{h}$$

Egyenes a projektív térben

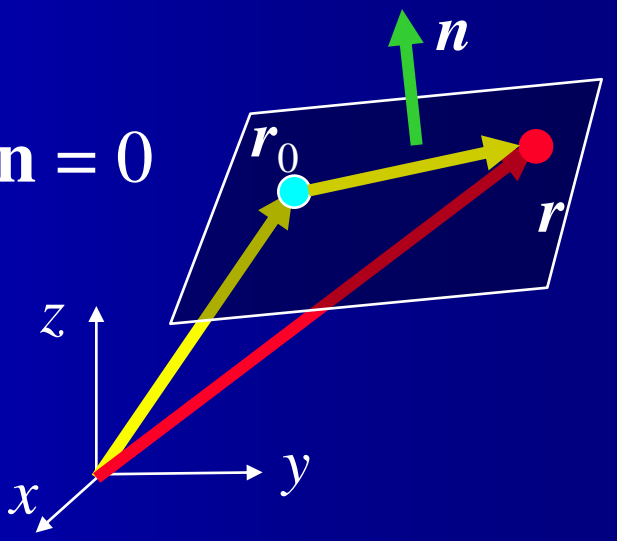


egyenes = két pont kombinációja
szakasz = két pont konvex kombinációja

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

Sík

$$(\mathbf{r} - \mathbf{r0}) \bullet \mathbf{n} = 0$$



Euklideszi tér, Descartes koordináták:

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

Euklideszi tér, homogén koordináták:

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

Projektív tér:

$$\boxed{n_x X_h + n_y Y_h + n_z Z_h + d\, h = 0} \qquad h \neq 0$$

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

Homogén lineáris transzformációk

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

$$[X_h', Y_h', h'] = [X_h, Y_h, h] \cdot \mathbf{T}_{3x3}$$

- 3D transzformáció 4x4 mátrix

$$[X_h', Y_h', Z_h', h'] = [X_h, Y_h, Z_h, h] \cdot \mathbf{T}_{4x4}$$

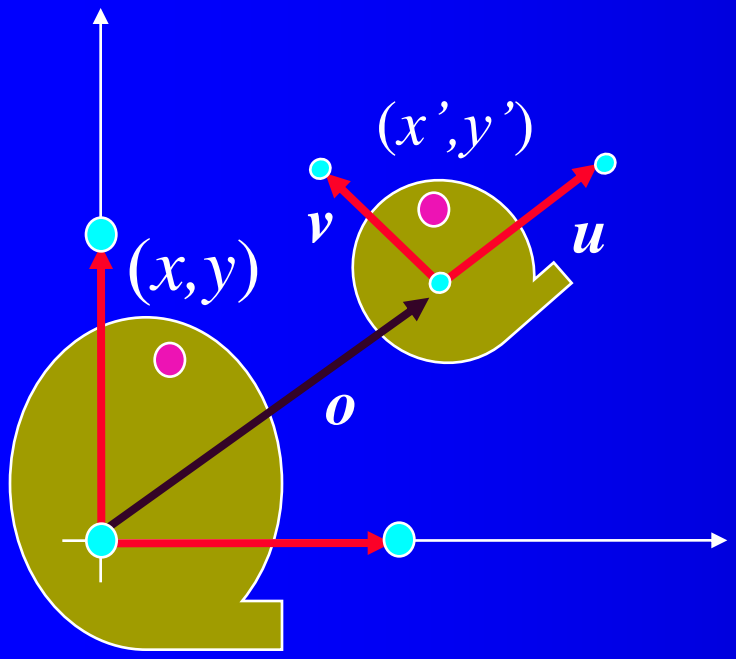
Affin transzformációk

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

$$x' = a_{11}\,x + a_{21}\,y + a_{31}$$
$$y' = a_{12}\,x + a_{22}\,y + a_{32}$$

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

Affin transzformációs mátrix sorai



$$(x', y') = \boldsymbol{o} + x\boldsymbol{u} + y\boldsymbol{v}$$

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ o_x & o_y & 1 \end{bmatrix}$$

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

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

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

Transzformációk összefűzése

$$[X_h', Y_h', Z_h', h'] = (\ldots([X_h, Y_h, Z_h, h] \cdot \boldsymbol{T}_1) \cdot \boldsymbol{T}_2) \ldots \boldsymbol{T}_n) =$$
$$= [X_h, Y_h, Z_h, h] \cdot (\boldsymbol{T}_1 \cdot \boldsymbol{T}_2 \cdot \ldots \cdot \boldsymbol{T}_n) =$$
$$= [X_h, Y_h, Z_h, h] \cdot \boldsymbol{T}$$

Mátrisszorzás asszociatív!

Homogén lineáris transzformációk tulajdonságai

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

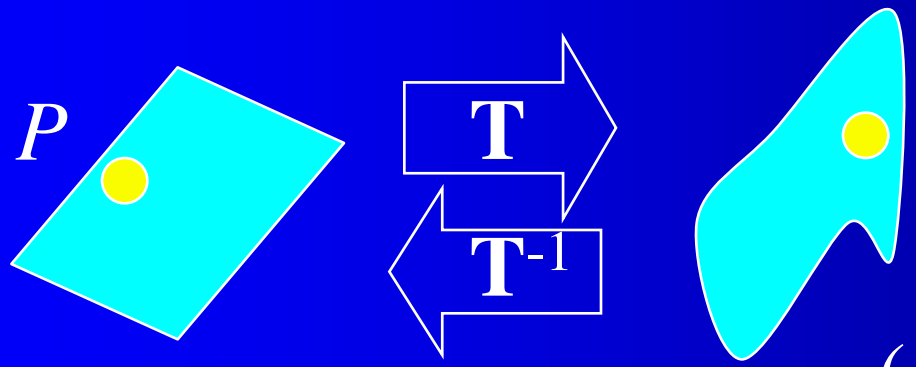
Példa: egyenest egyenesbe

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

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

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

Invertálható homogén lineáris transzformációk: síkot síkba



$P* = P \cdot \mathbf{T}$

$P \cdot N^T = 0$

$(P* \cdot \mathbf{T}^{-1}) \cdot N^T = 0$

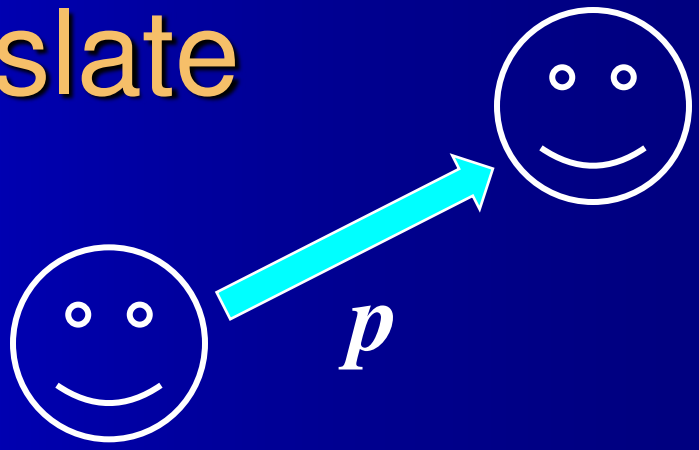
$P* \cdot (\mathbf{T}^{-1} \cdot N^T) = 0$

$P* \cdot N*^T = 0$

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

Eltolás: glTranslate

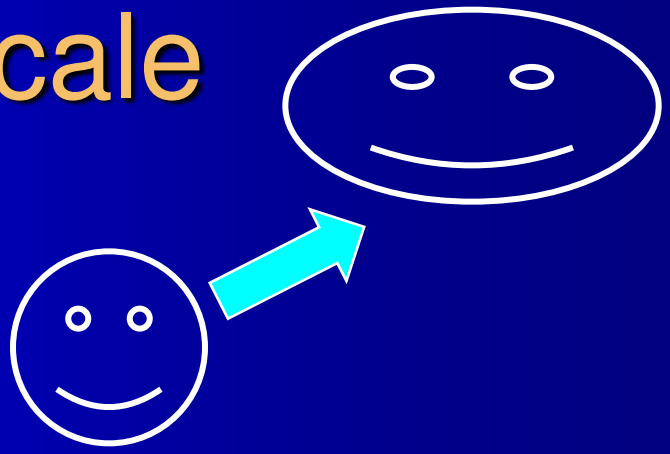
$(x', y', z') = (x+p_x, y+p_y, z+p_z)$



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

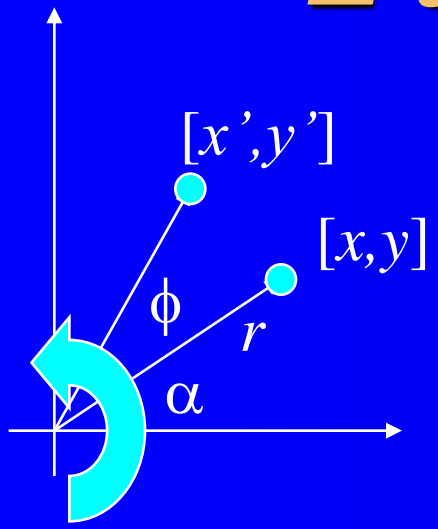
Skálázás: glScale

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



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

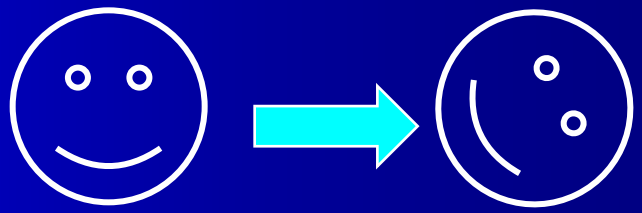
Z tengely körüli forgatás



$z' = z$

$x = r\cos\alpha$

$y = r\sin\alpha$



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

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

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

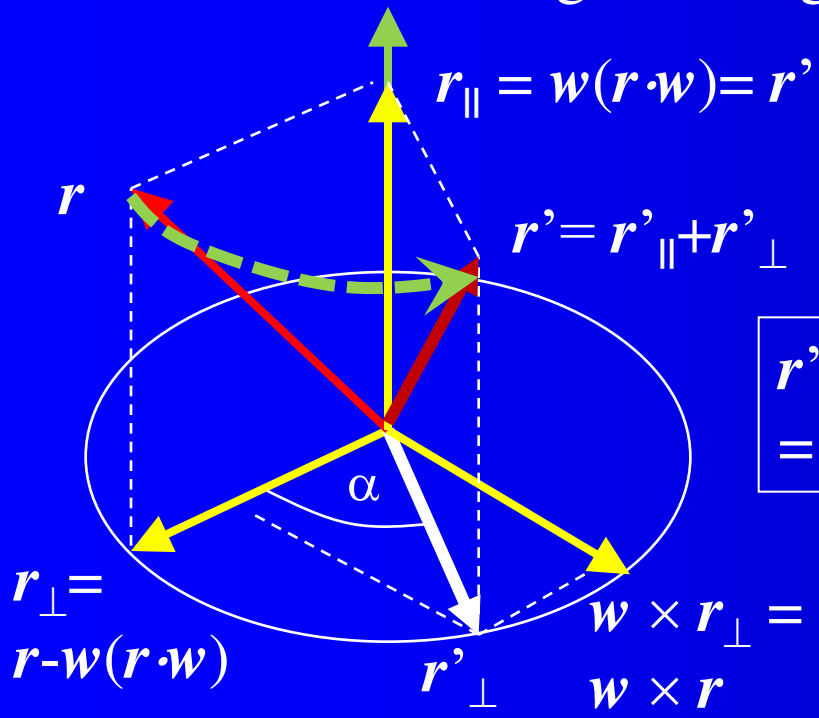
$y' = r\sin(\alpha+\phi) = x\sin\phi + y\cos\phi$

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


Z-tengely körüli forgatás
homogén lineáris
transzformációs mátrixa

Általános $\boldsymbol{w}$ tengely körüli forgatás (Rodrigues): glRotate

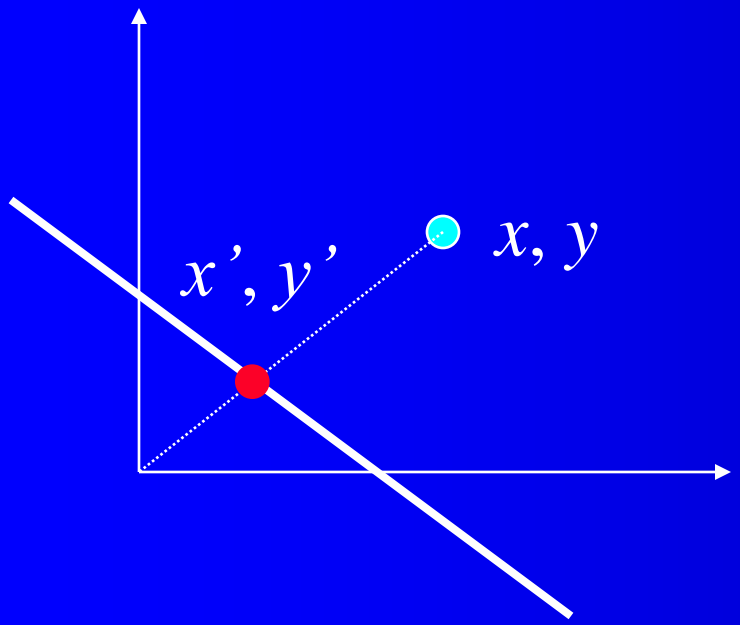
$\boldsymbol{w}$: forgatási tengely, legyen egységvektor



$$r' = w(r \cdot w) + (r - w(r \cdot w))\cos(\alpha) + w \times r \sin(\alpha)$$
$$= r\cos(\alpha) + w(r \cdot w)(1 - \cos(\alpha)) + w \times r \sin(\alpha)$$

Mátrix sorai: $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$-ra alkalmazva
Házi feladat!

Középpontos vetítés (2D)

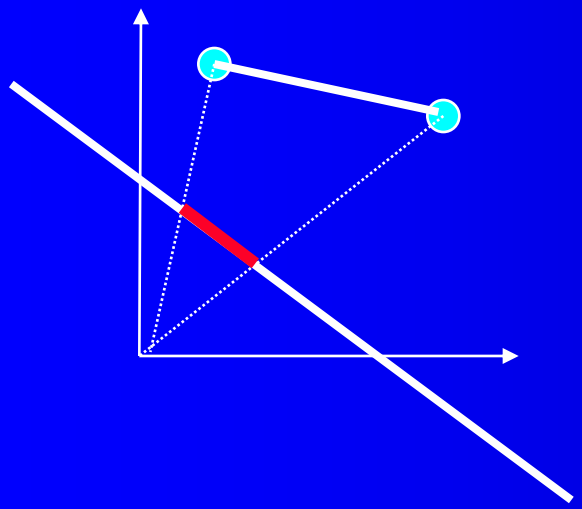


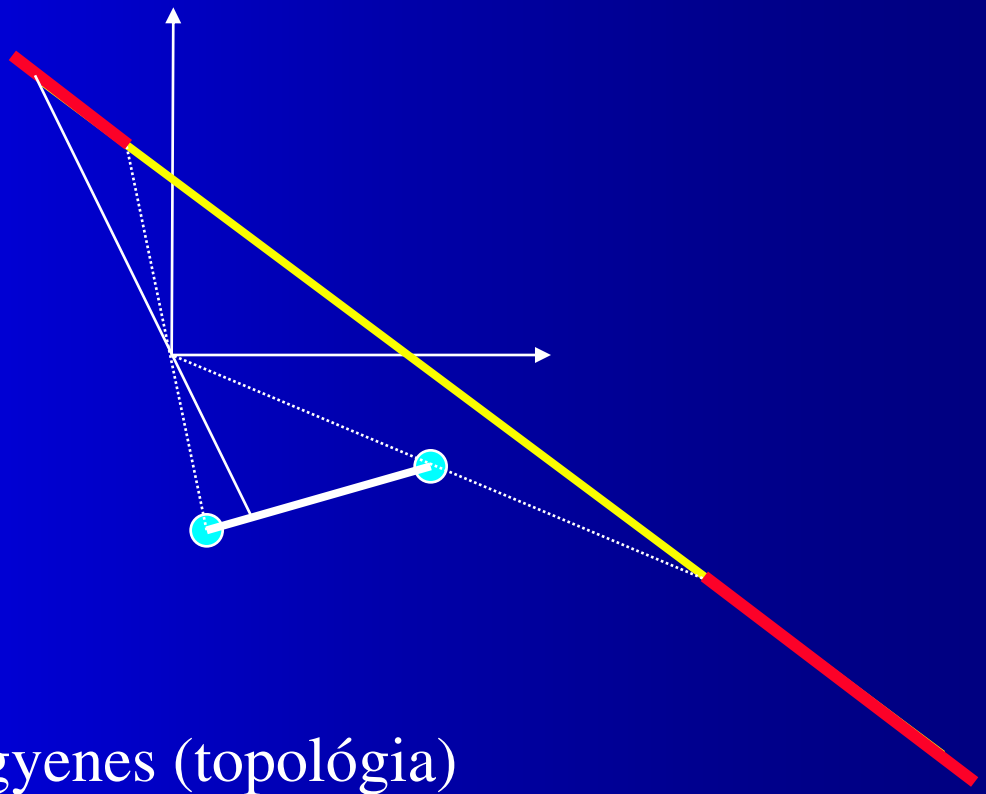
$$\begin{bmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 0 \end{bmatrix}$$

$[x, y, 1]$ $[x, y, px+qy]$

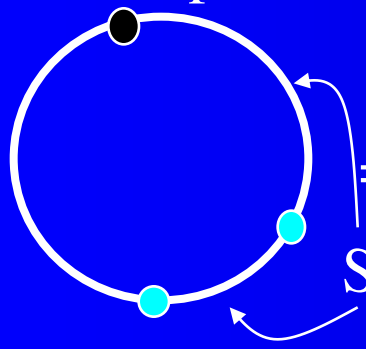
$x/y = x'/y'$ és $px+qy=1$ ⇨ $\left[\frac{x}{px+qy} \quad \frac{y}{px+qy} \right]$

Átfordulási probléma





Ideális pont



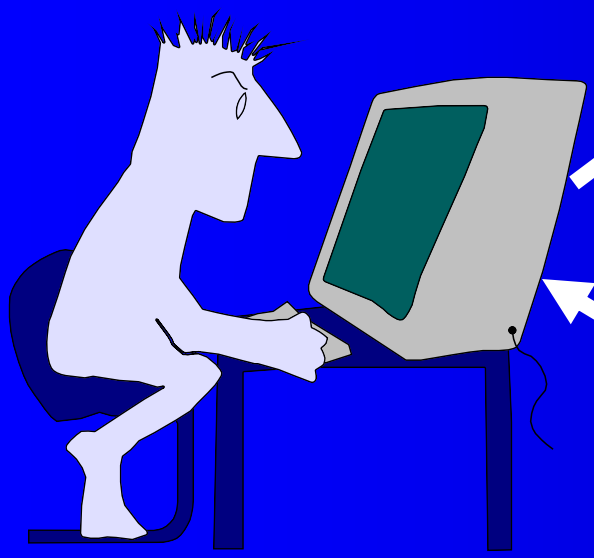
=Projektív egyenes (topológia)

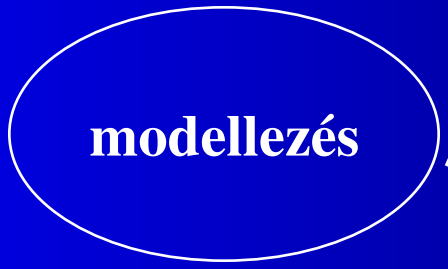
Szakasz ?????

2D képszintézis

Szirmay-Kalos László

Számítógépes grafika







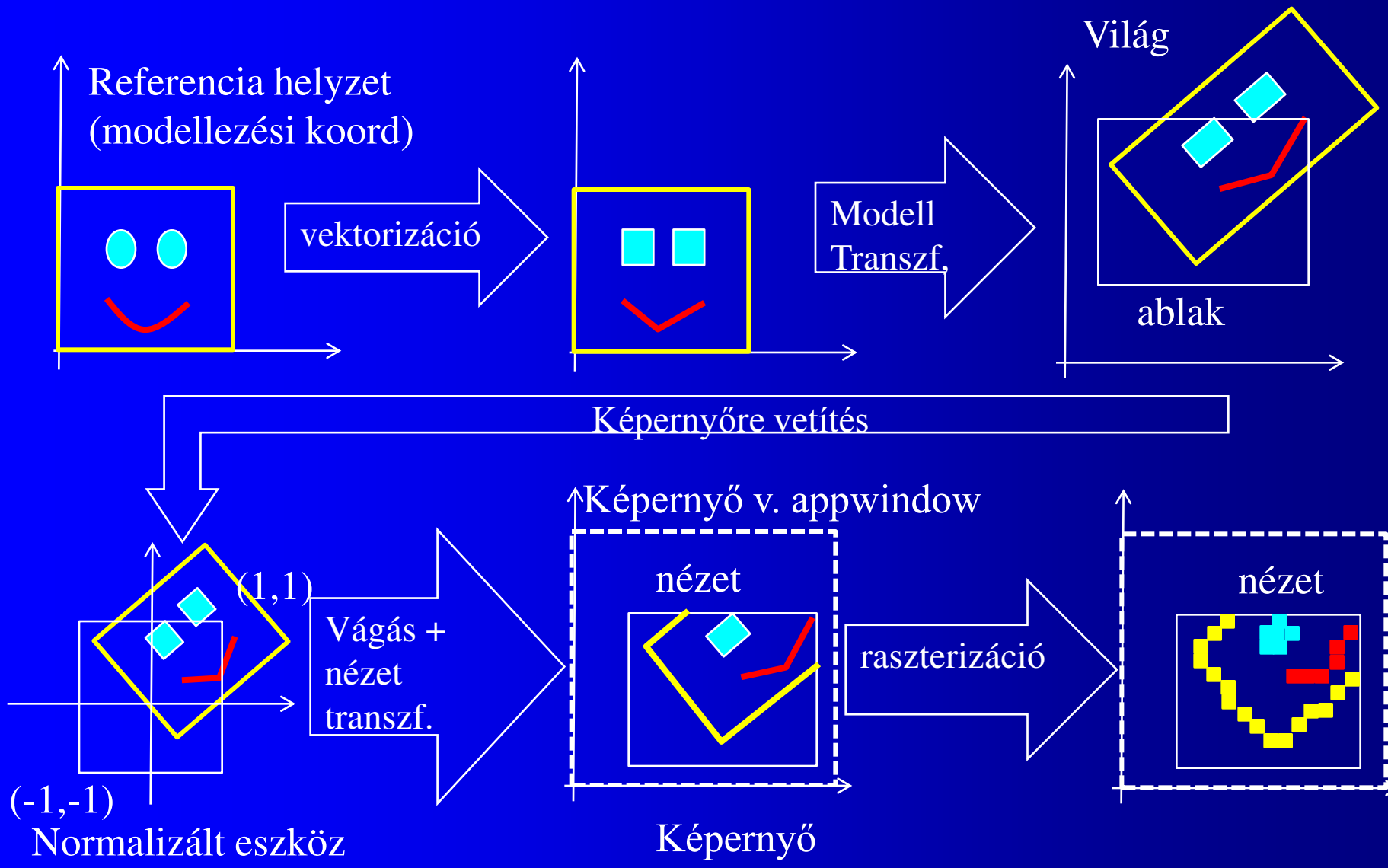
Virtuális világ modell

világ = sík

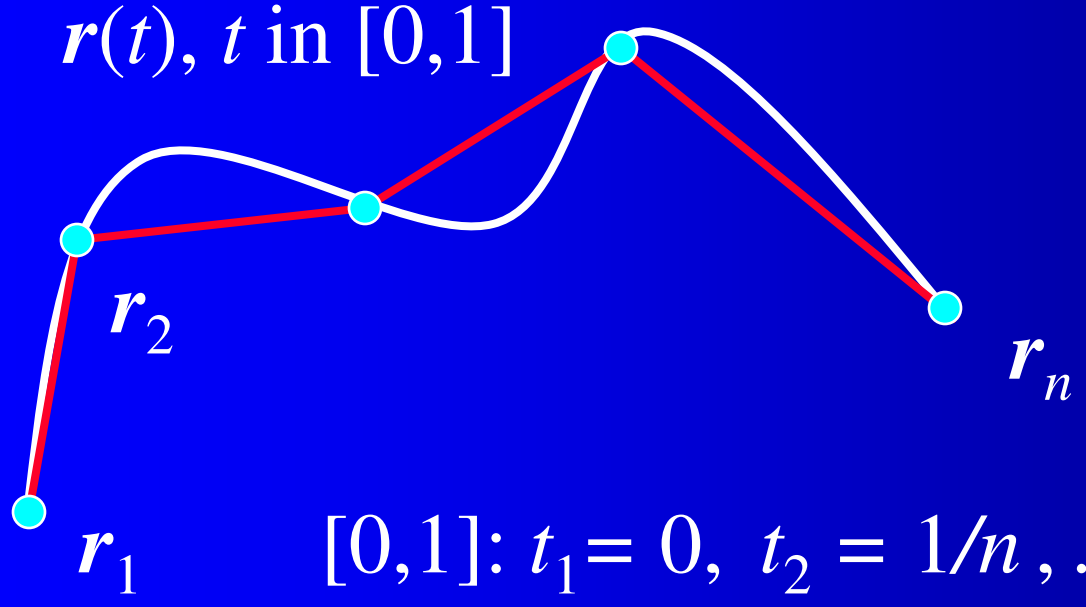
Metafórák:

- **2D rajzolás**

2D megjelenítési csővezeték



Vektorizáció

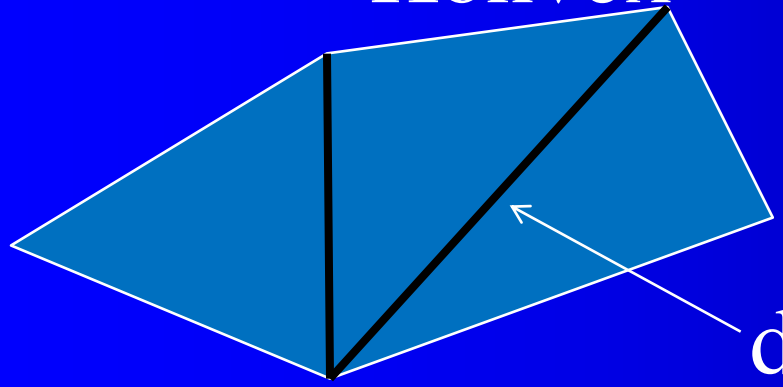


$[0,1]$: $t_1 = 0,\ t_2 = 1/n, \ldots, t_n = 1$

$\boldsymbol{r}_1 = \boldsymbol{r}(0), \boldsymbol{r}_2 = \boldsymbol{r}(t_2), \ldots, \boldsymbol{r}_n = \boldsymbol{r}(1)$

Poligon háromszögekre bontása

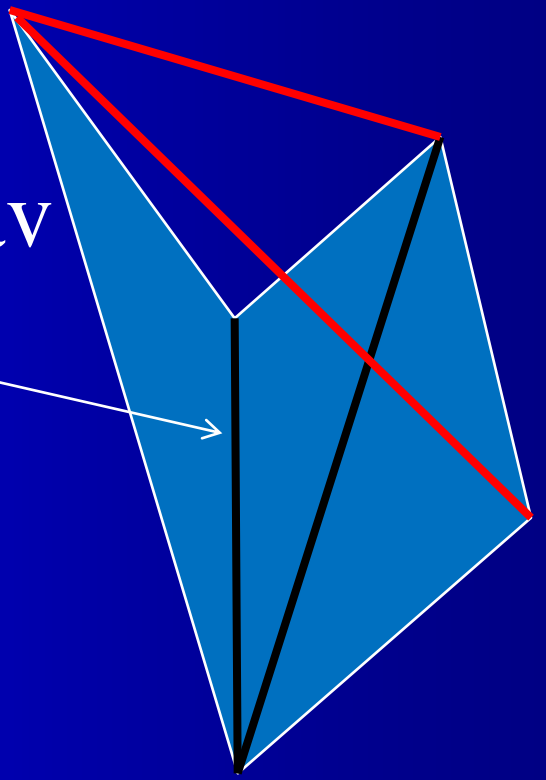
Konvex



Nem-diagonál

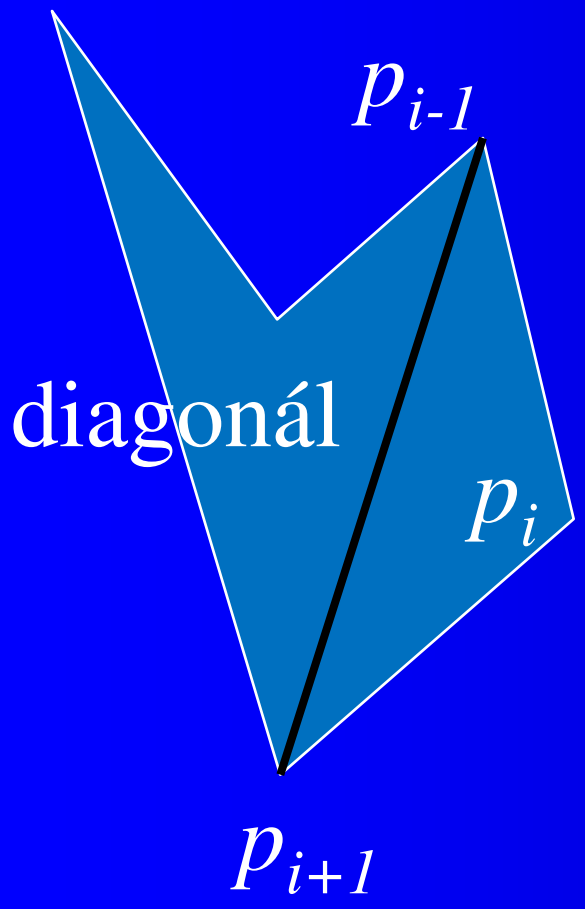
Konkáv

diagonál



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

Fül

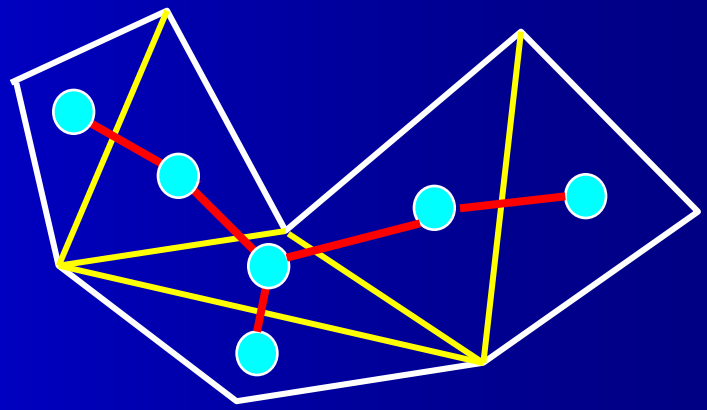




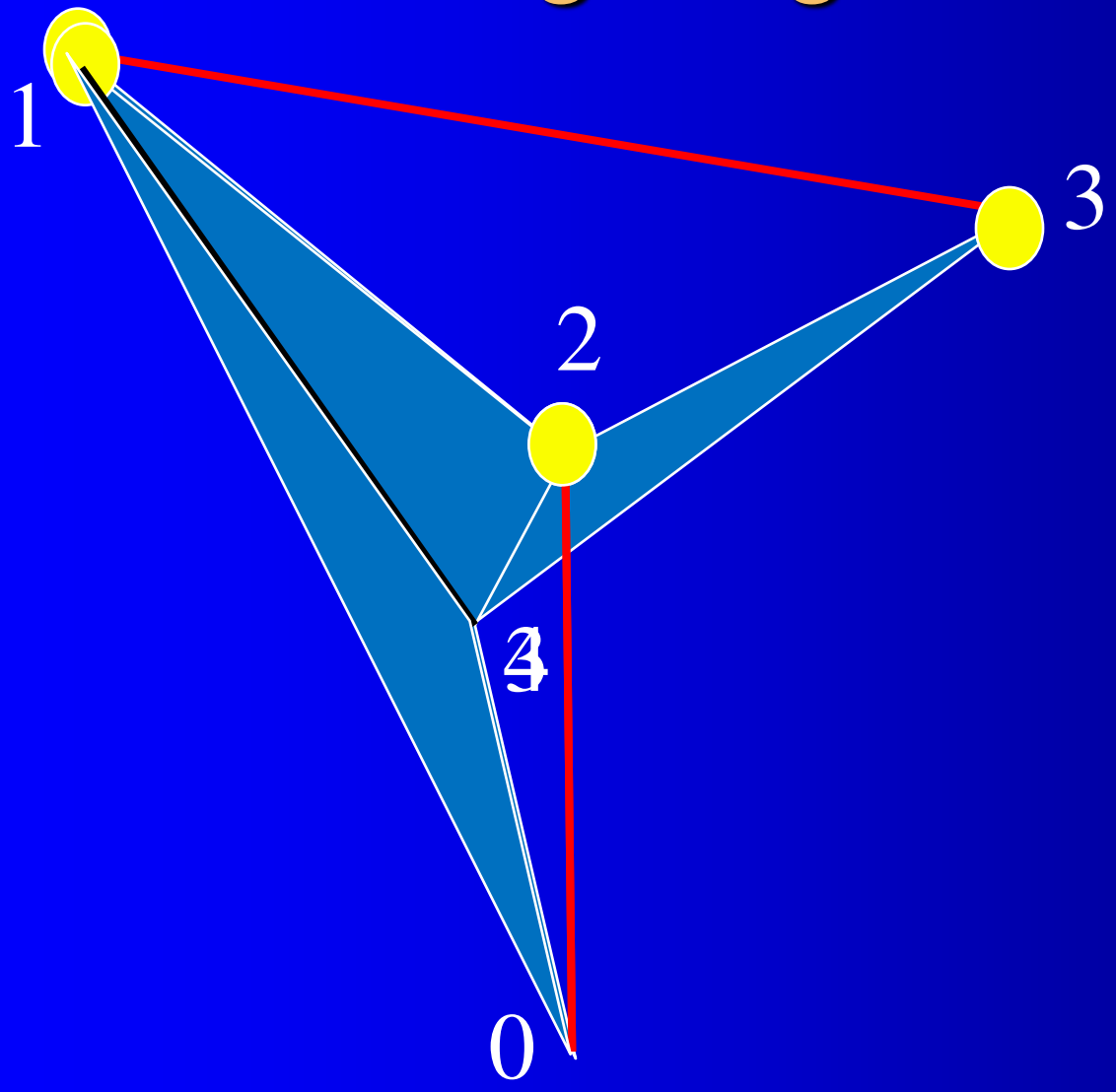
- p_i fül, ha $p_{i-1} - p_{i+1}$ diagonál
- Fül levágható!
- **Fülvágás**:

  keress fület és nyissz!

Két fül tétel: Minden legalább 4 csúcsú egyszerű sokszögnek van legalább 2 füle.

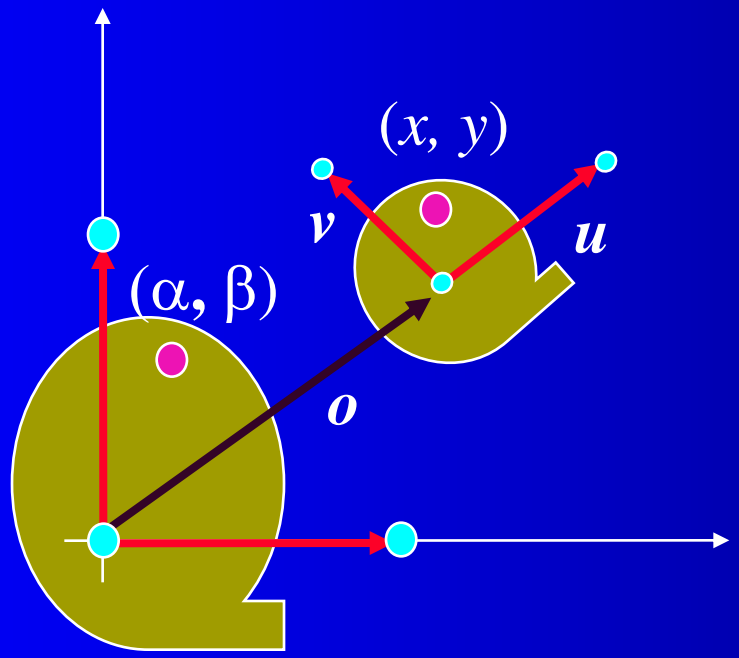


Fülvágó algoritmus



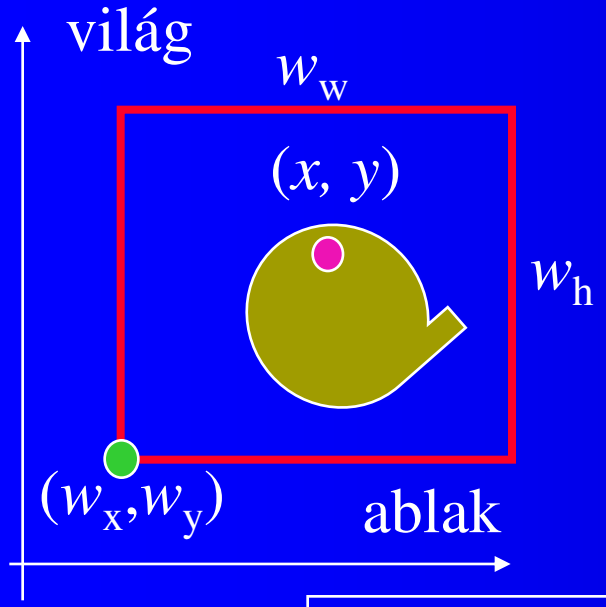
Modellezési transzformáció

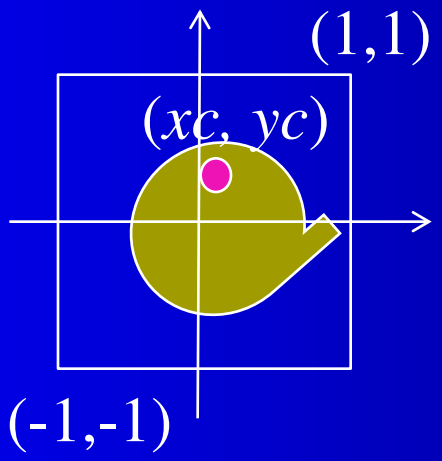
$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p_x & p_y & 1 \end{bmatrix}$$

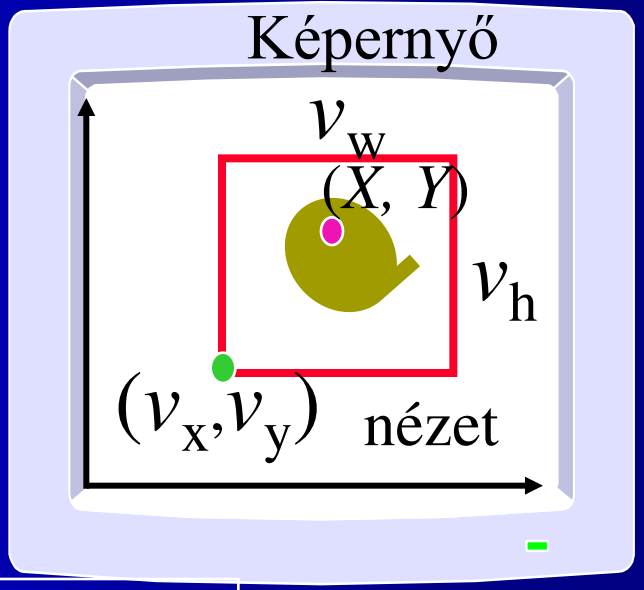


$$(x, y) = \boldsymbol{o} + \alpha\boldsymbol{u} + \beta\boldsymbol{v}$$

Képernyő projekció







$$xc = 2(x-w_x)/w_w-1$$
$$yc = 2(y-w_y)/w_h-1$$

$$X = v_w\,(xc+1)/2+v_x$$
$$Y = v_h\,(yc+1)/2+v_y$$

```
glViewport(vx, vy, vw, vh);
```

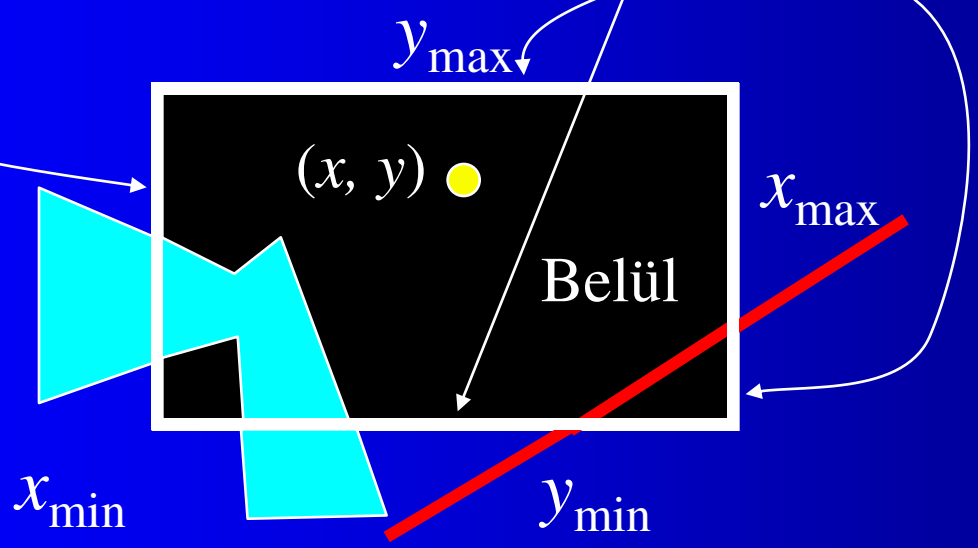
$$[xc,\ yc,\ 1] = [x,\ y,\ 1] \begin{bmatrix} 2/w_w & 0 & 0 \\ 0 & 2/w_h & 0 \\ -1-2w_x/w_w & -1-2w_y/w_h & 1 \end{bmatrix}$$

```
gluOrtho2D(wx,wx+ww,wy,wy+wh);
```

Vágás

Pont vágás:

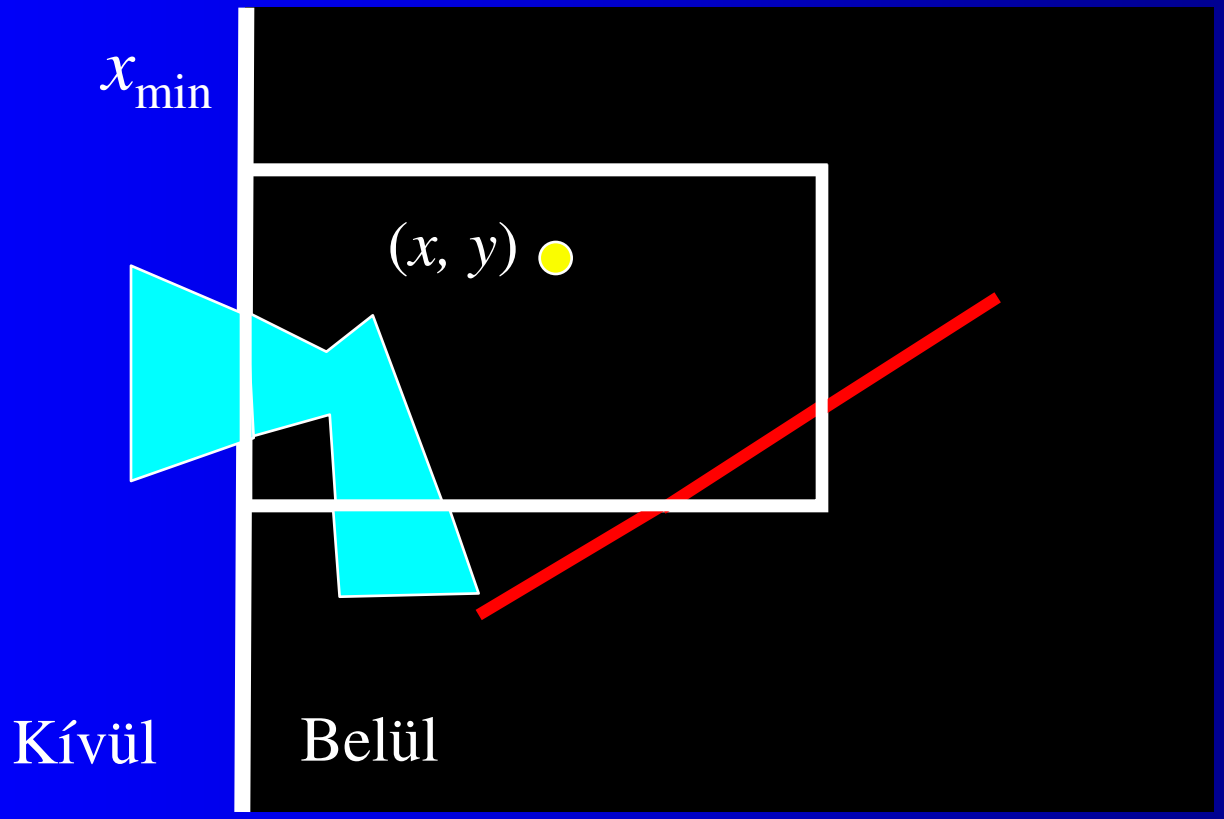
$x > x_{min} = -1 \quad x < x_{max} = +1 \quad y > y_{min} = -1 \quad y < y_{max} = +1$



Kívül

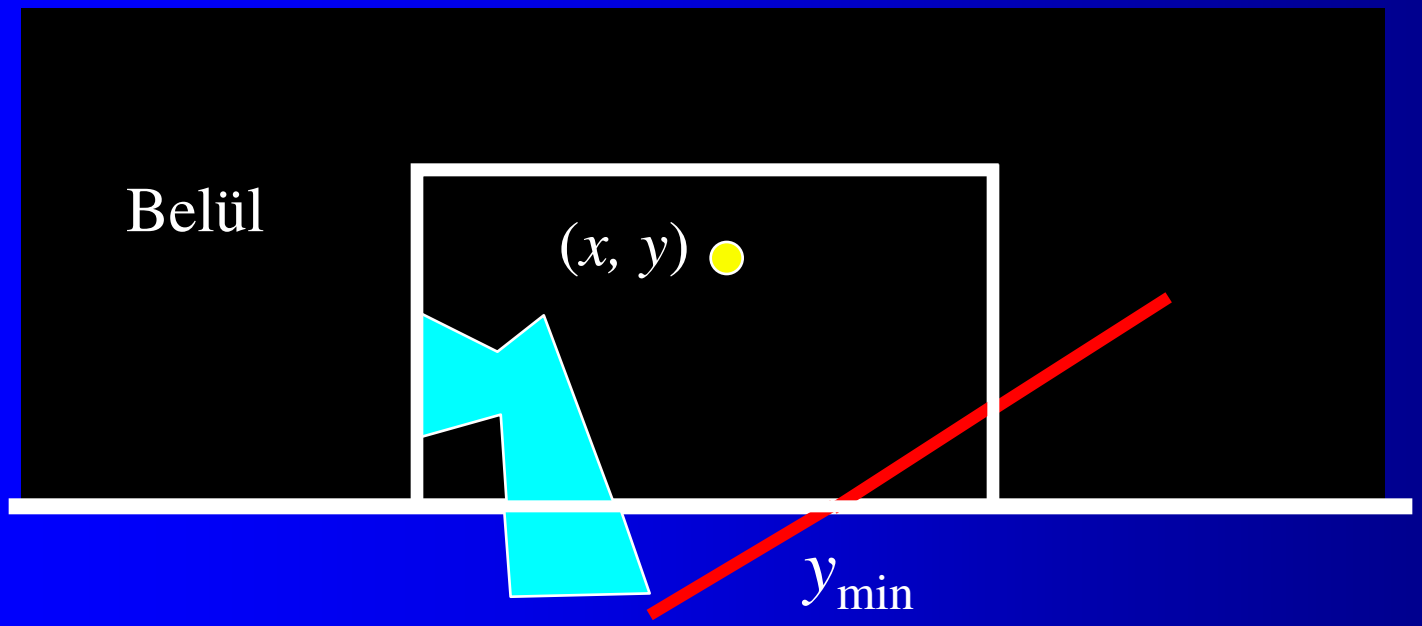
Vágás

$x > x_{\min}$



Vágás

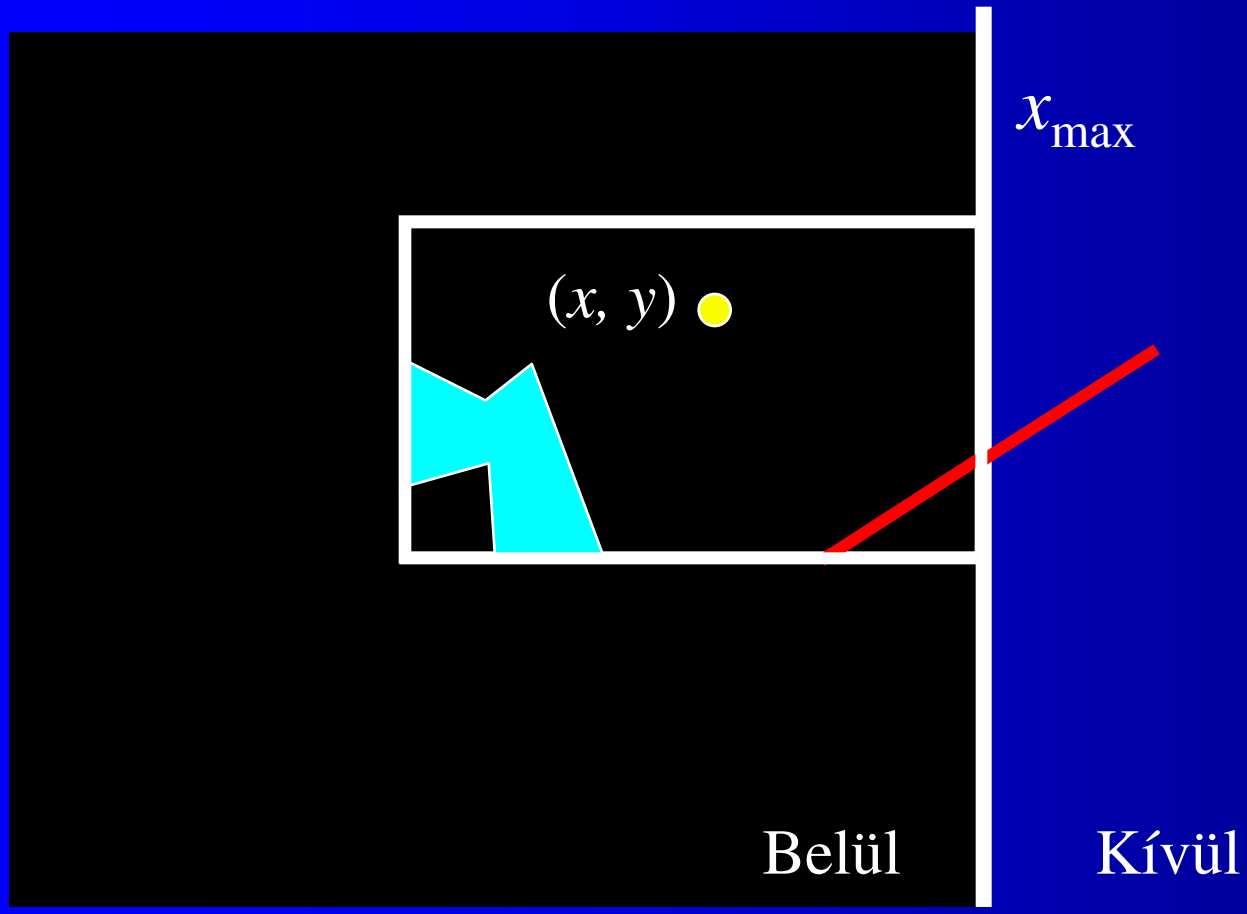
$y > y_{min}$



Kívül

Vágás

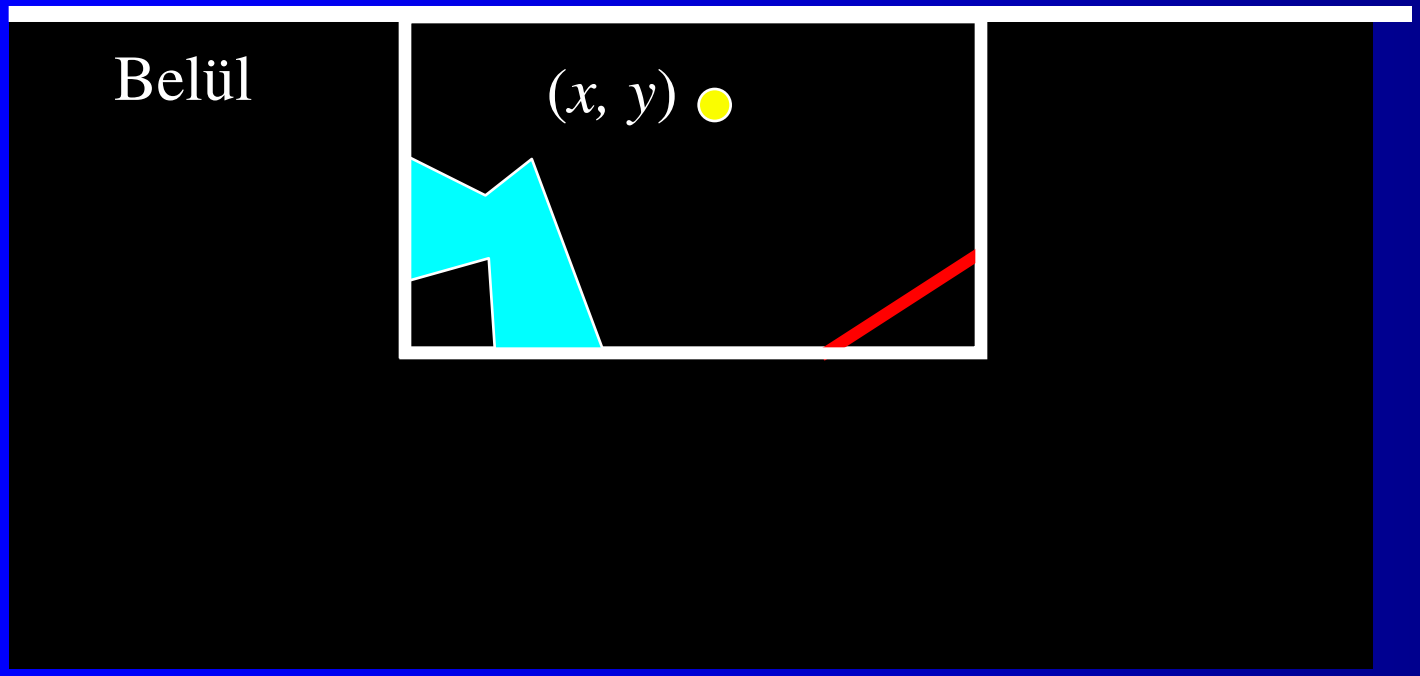
$x < x_{\max}$



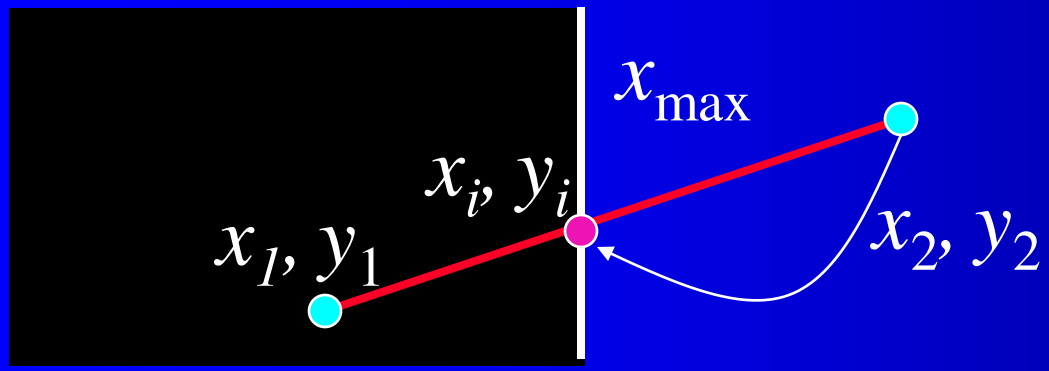
Vágás

$y < y_{max}$

Kívül



Szakasz vágás



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

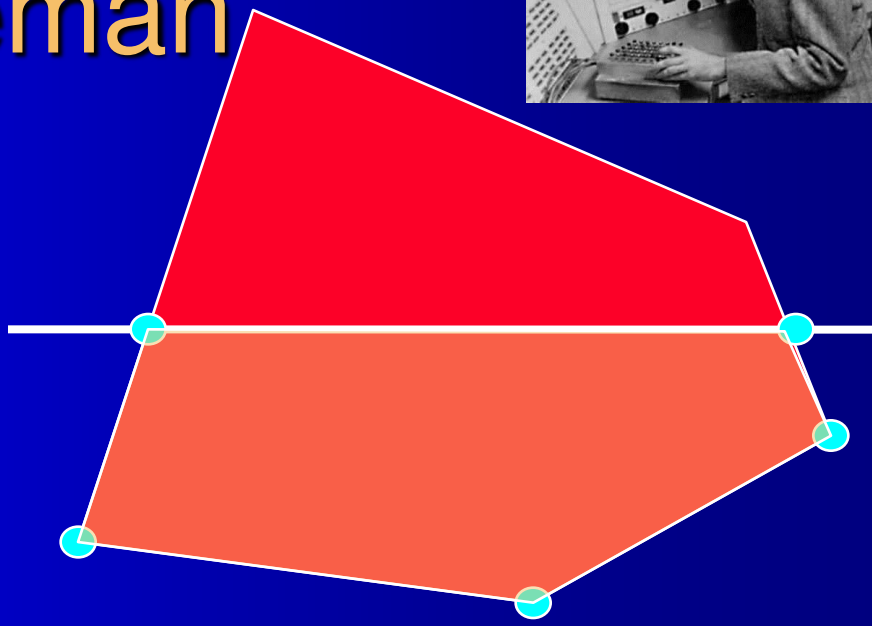
$x = x_{max}$

- Metszés:
- $x_{max} = x_1 + (x_2 - x_1)t \Rightarrow t = (x_{max} - x_1)/(x_2 - x_1)$

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

Sutherland-Hodgeman poligonvágás

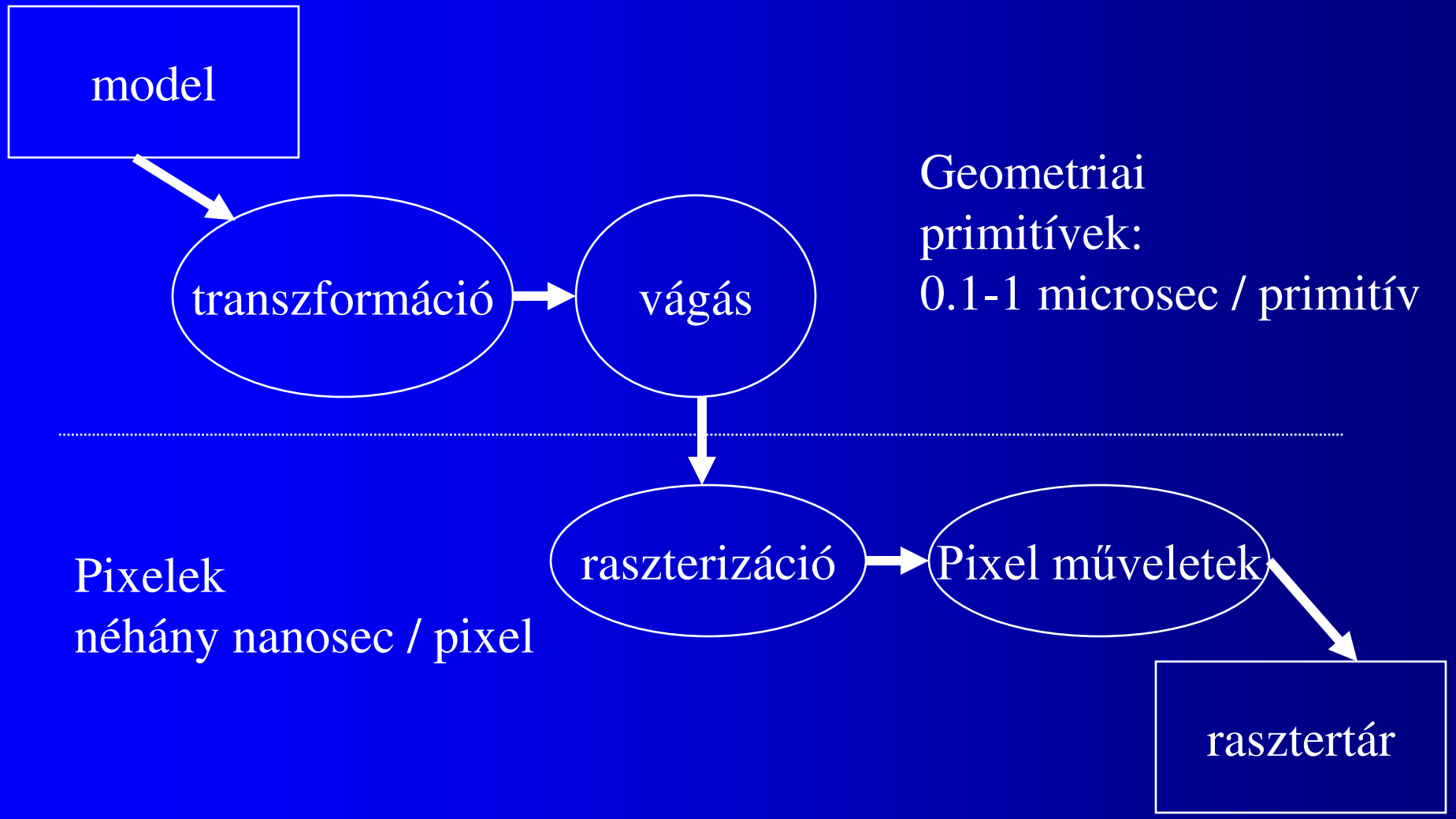




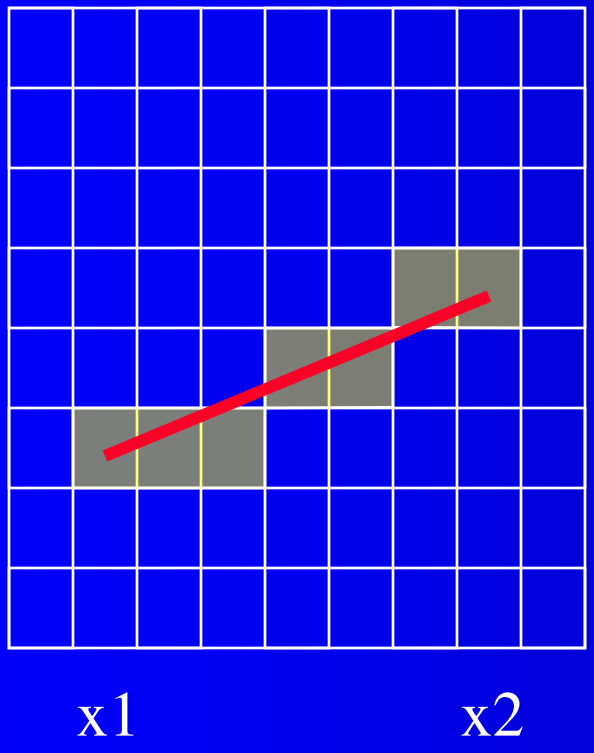
```
PolygonClip(p[n] ⇨ q[m])
    m = 0;
    for( i=0; i < n; i++) {
        if (p[i] belső) {
            q[m++] = p[i];
            if (p[i+1] külső)
                q[m++] = Intersect(p[i], p[i+1], vágóegyenes);
        } else {
            if  (p[i+1] belső)
                q[m++] = Intersect(p[i],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

Első pontot még egyszer
a tömb végére

Raszterizáció



Szakasz rajzolás



Egyenes egyenlete:

$$y = mx + b$$

Egyeneshúzás

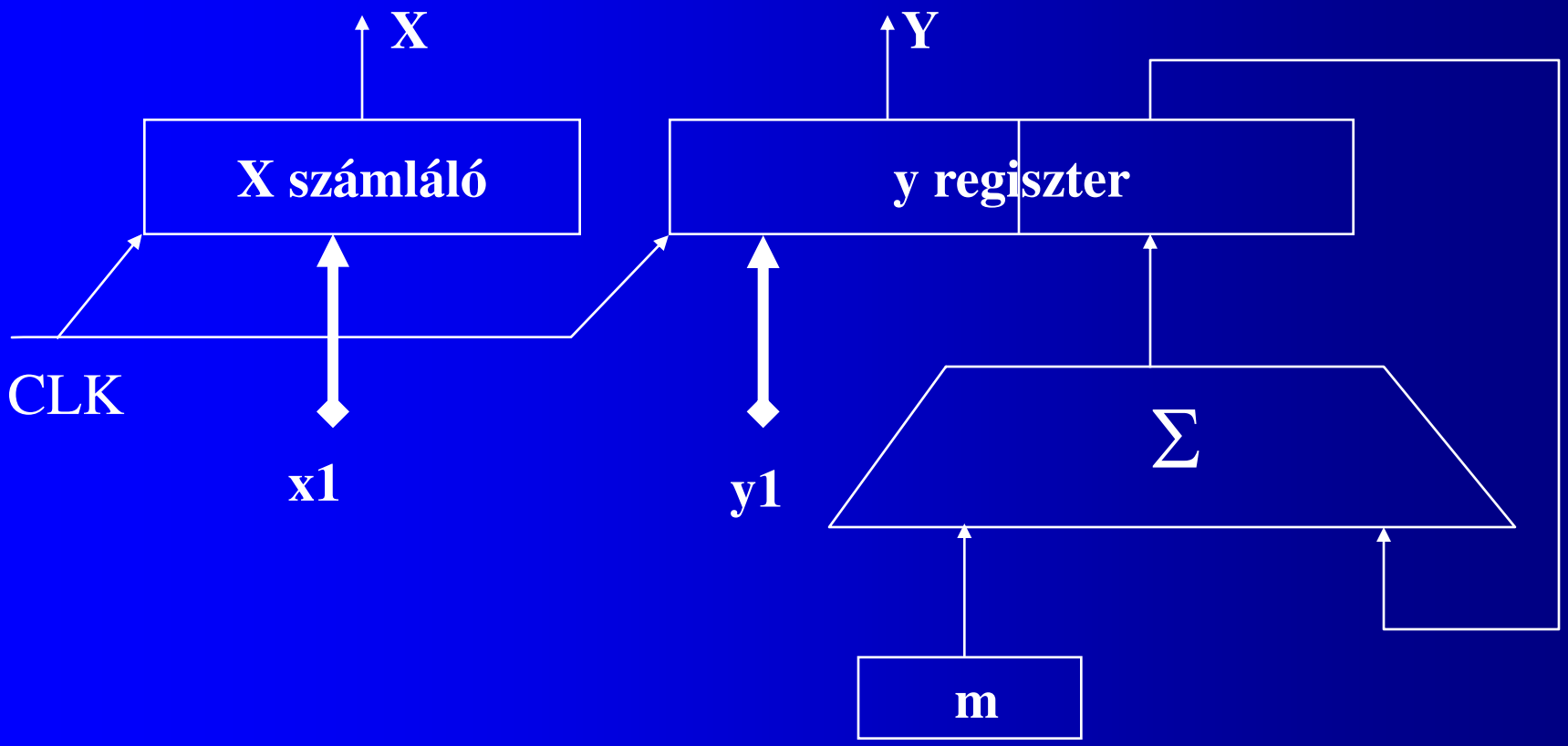
```
for( x = x1; x <= x2; x++) {
        Y = m*x + b;
        y = Round( Y );
        write( x, y );
}
```

Inkrementális elv

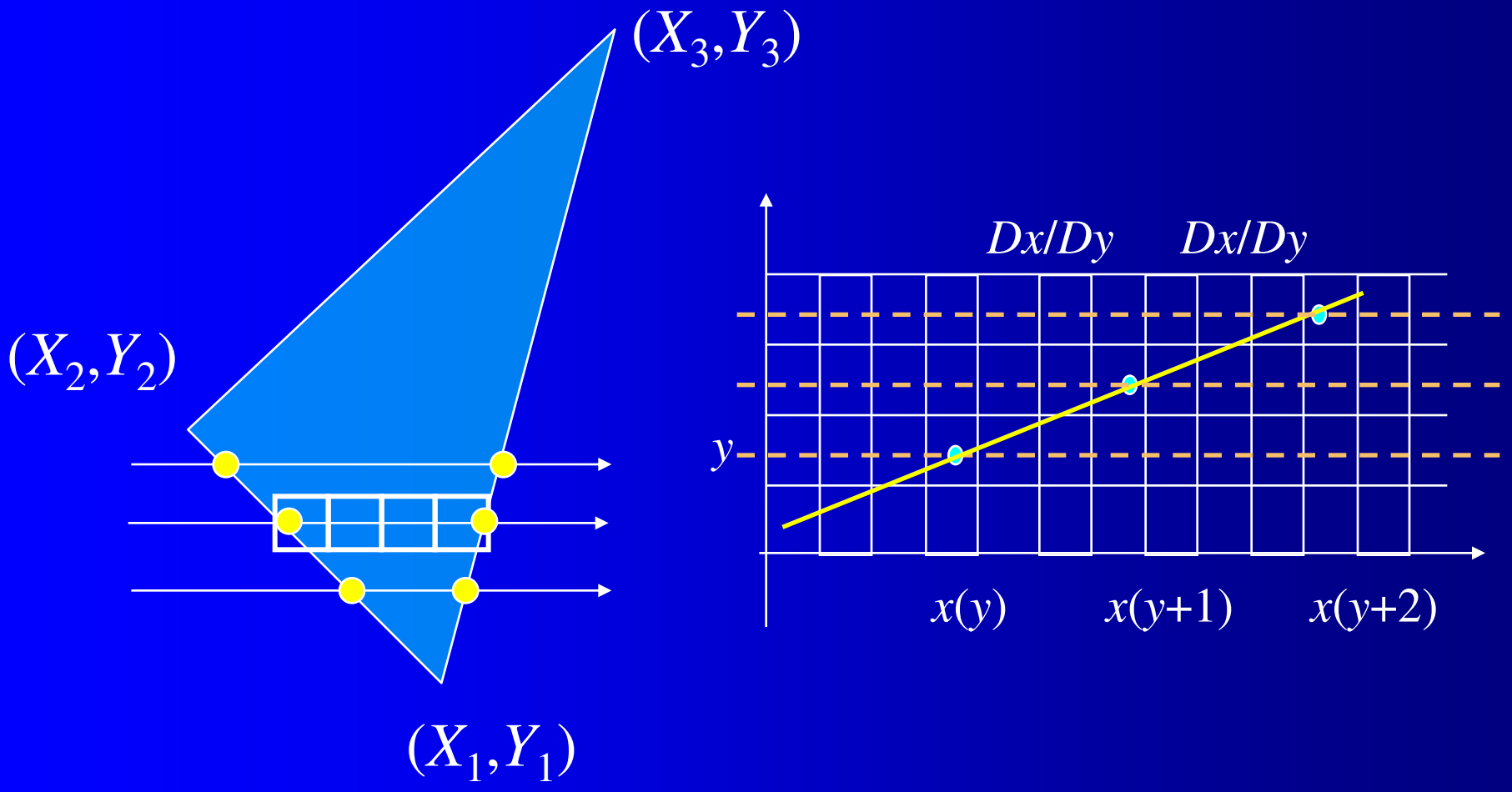
- Egyenlet: $Y(X) = mX + b = Y(X\text{-}1) + m$

```
DDADrawLine(int x1, int y1, int x2, int y2) {
        float m = (y2 - y1)/(x2 - x1);
        float y = y1;
        for(int x = x1; x <= x2; x++) {
                int Y = round(y);
                WRITE(x, Y, color);
                y = y+m;
        }
}
```

DDA szakaszrajzoló hardver

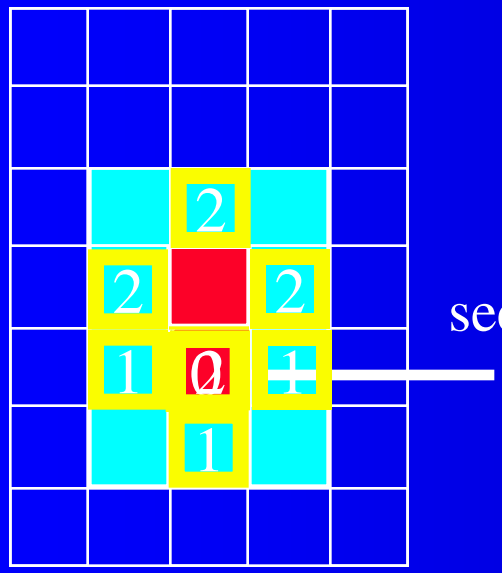


Háromszög kitöltés



Terület elárasztás

Belső pont = kék




```
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

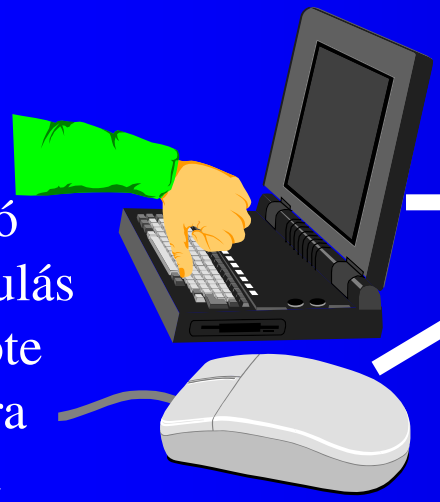
1 Write Flood(x,y-1); Flood(x,y+1); Flood(x-1,y); Flood(x+1,y);

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

2D grafikus rendszerek

Szirmay-Kalos László

Interaktív programok felépítése



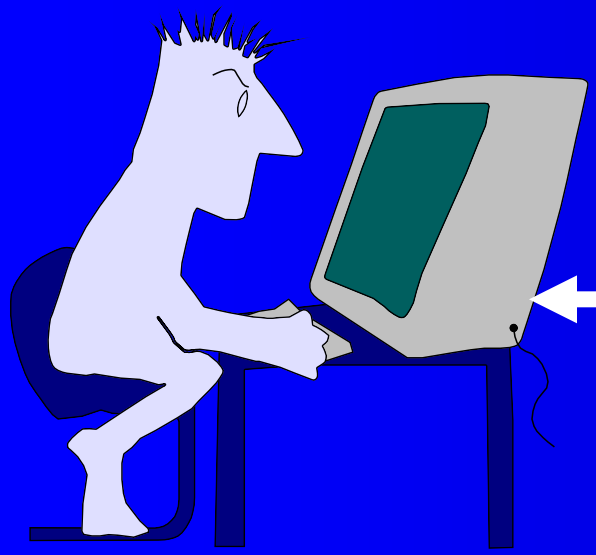
Pozíció
Gyorsulás
Wiimote
Kamera
Kinect

input csővezeték

Esemény kezelés

modellezés

Virtuális világ modell

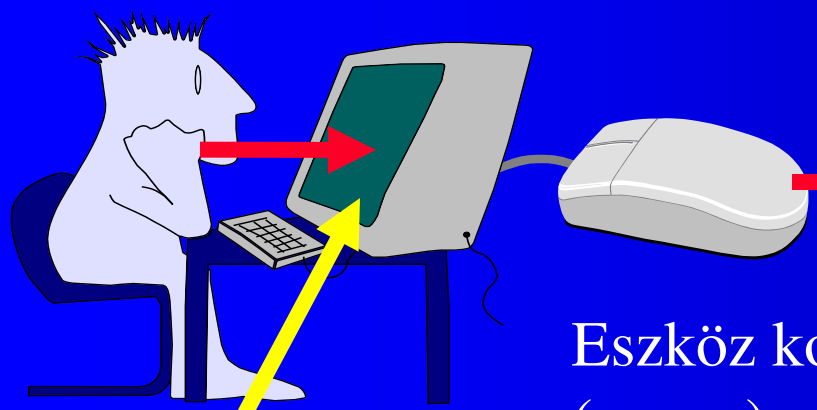


output csővezeték

Grafikus hardver

képszintézis

Funkcionális modell



Bemeneti csővezeték

T_V^{-1}

T_M^{-1}

Eszköz-
képtér

Képtér-
világ

Világ-
lokális mod.

Eszköz koord.
(xe, ye)

kamera

Virtuális világ

Kép
frissités

Kurzor
(xp, yp)

vektorizáció

rasztertár

Pixel
op

Raszteri-
záció

Nézet
transzf

Vágás

Kép
proj

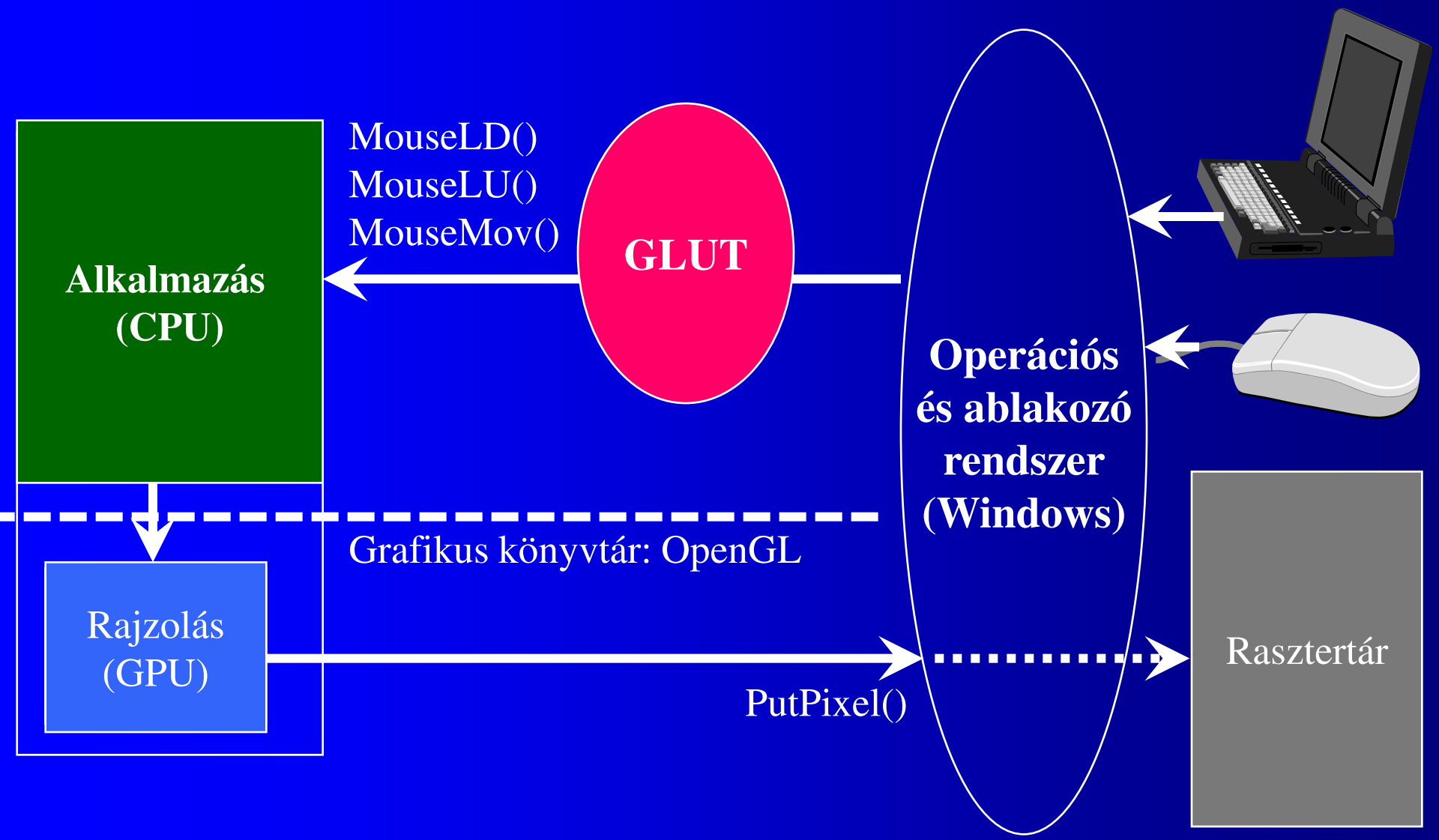
Modellezési
transzf.

Kimeneti csővezeték

T_V

T_M

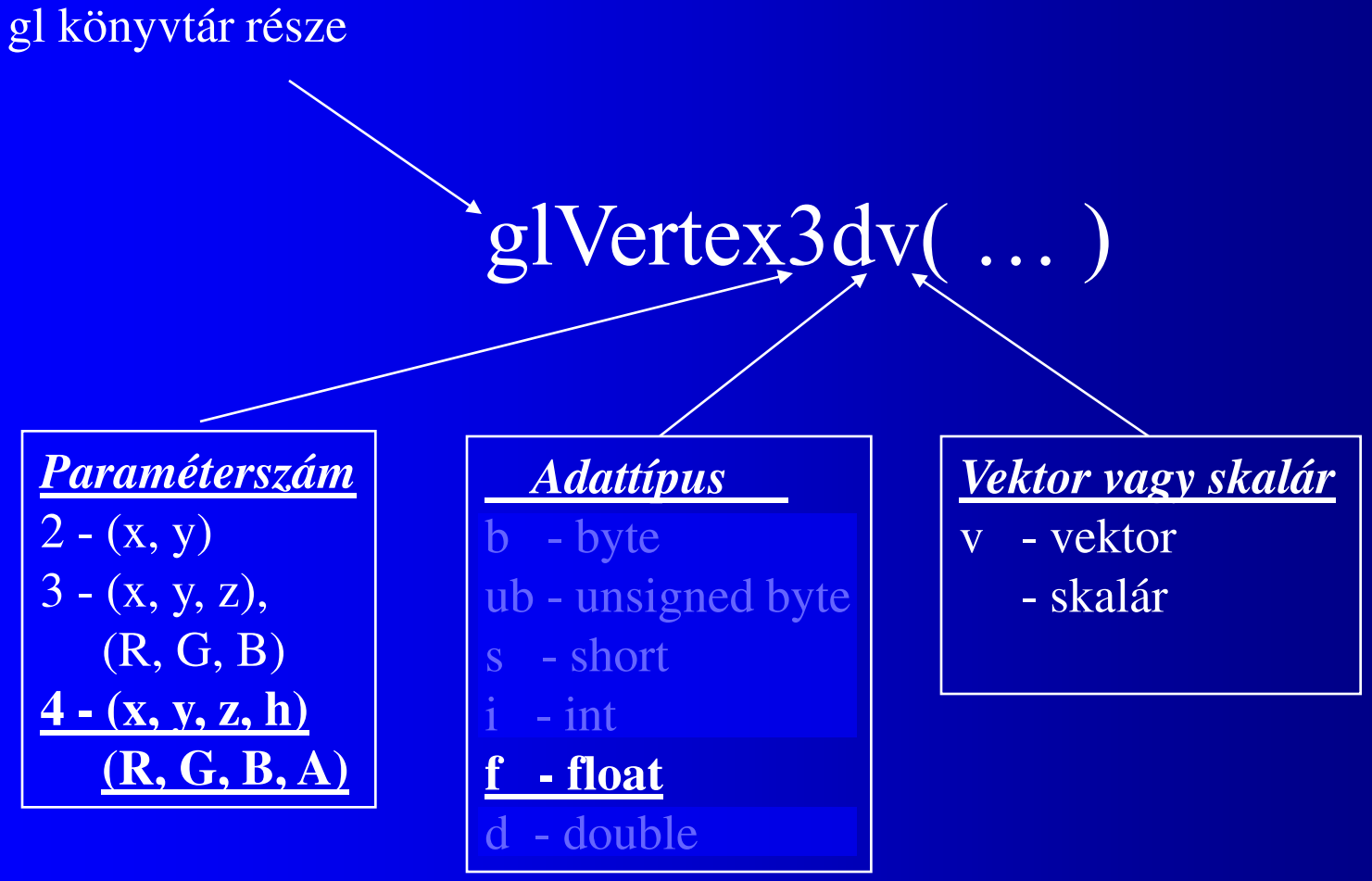
Szoftver


MouseLD()
MouseLU()
MouseMov()
GLUT
Alkalmazás
(CPU)
Operációs
és ablakozó
rendszer
(Windows)
Grafikus könyvtár: OpenGL
Rajzolás
(GPU)
PutPixel()
Rasztertár

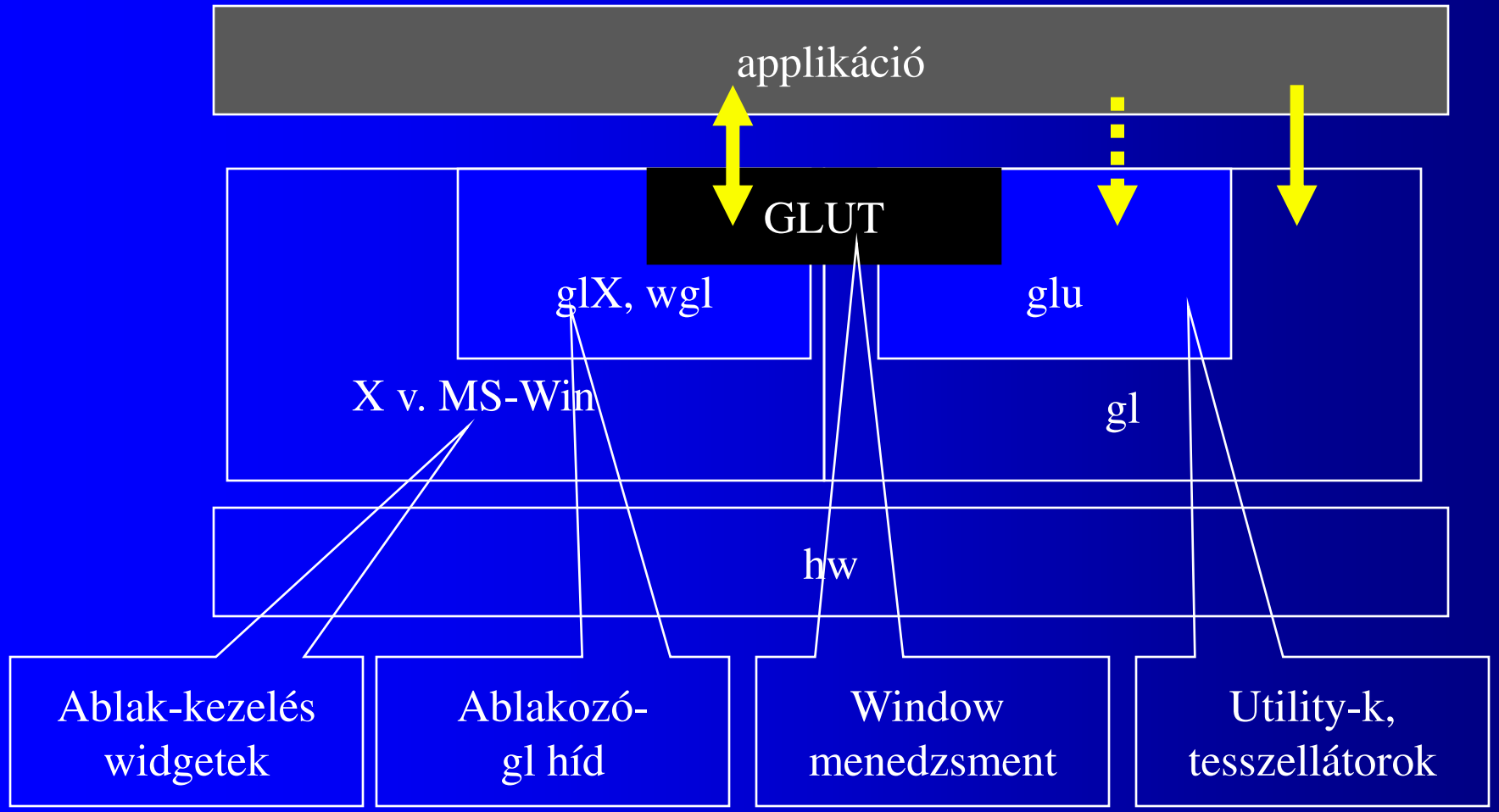
Kimeneti csővezeték: OpenGL

- Képszintézis könyvtár, API
  - 3D geometria (2D speciális eset)
  - rajzolási 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)

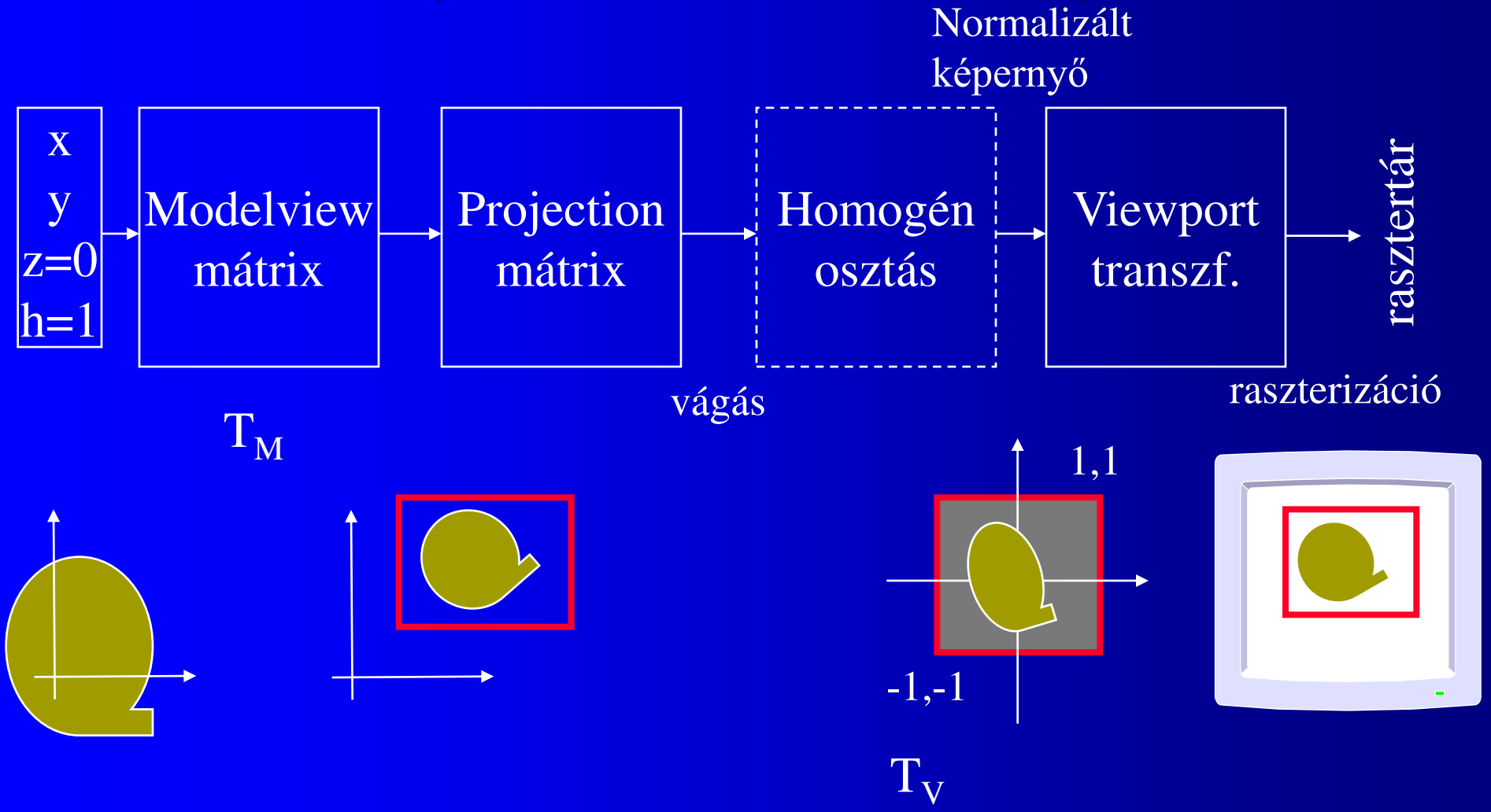
OpenGL szintaxis (C könyvtár)



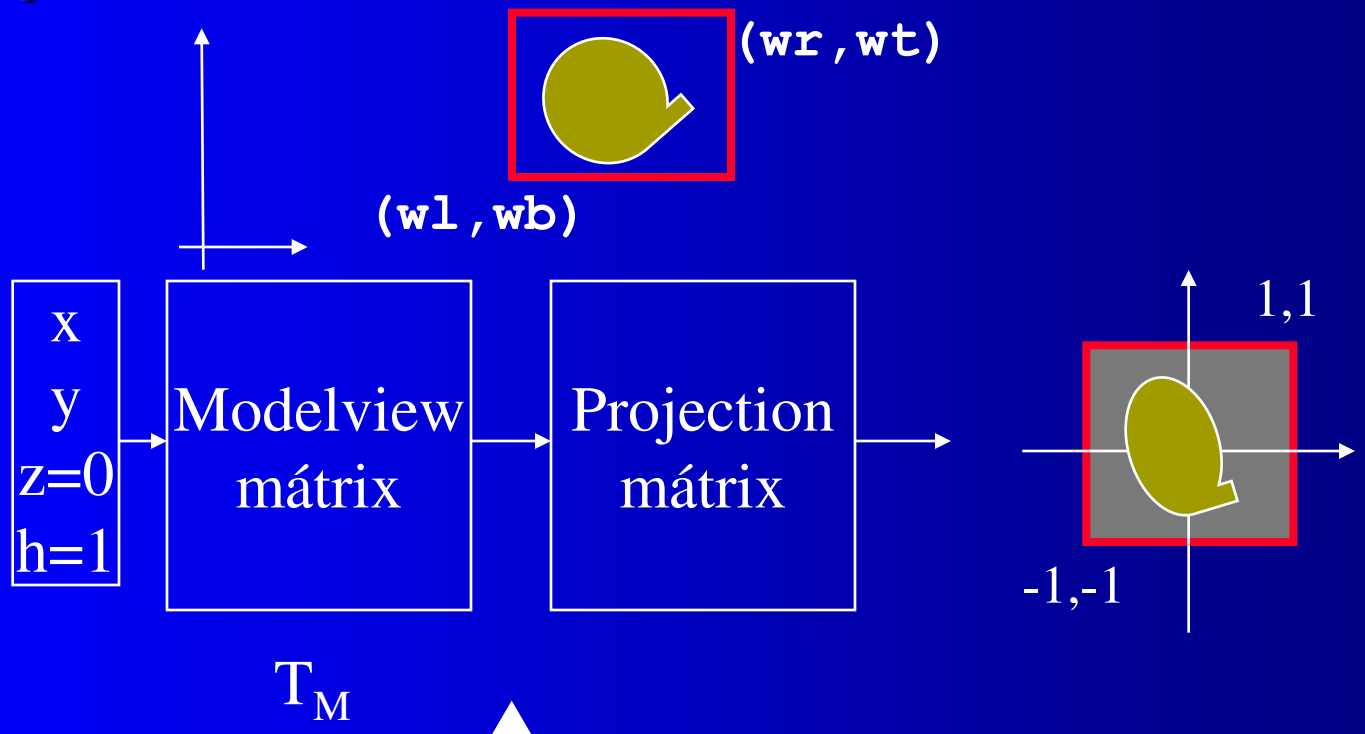
Ablakozó – OpenGL – alkalmazás elhelyezkedése



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

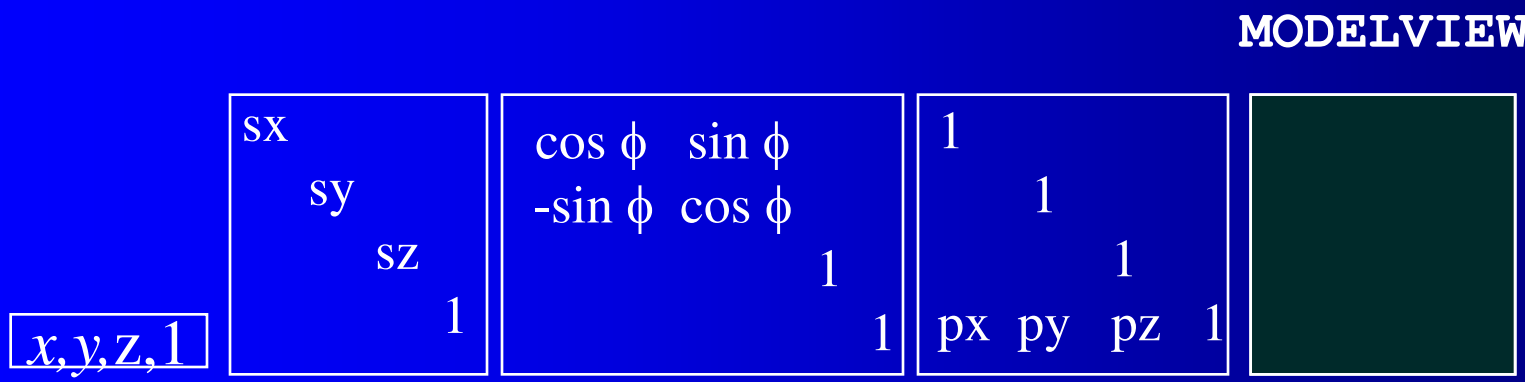
- OpenGL specifikáció:

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

- Mi:

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

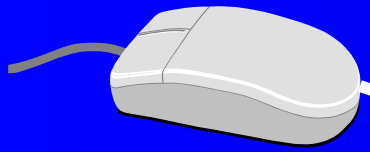
Mátrix konkatenáció

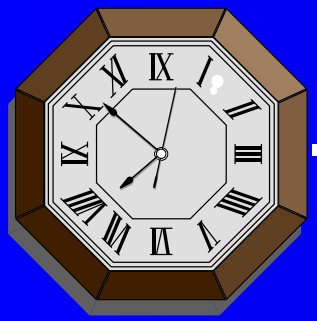



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
glTranslatef(px, py, pz);
glRotatef(phi, 0, 0, 1);
glScalef(sx, sy, sz);
. . .
glVertex2f(x,y);
```

Input/Output kezelés







Operációs és ablakozó rendszer (Windows)

Ablak létrehozás
callback
regisztráció

GLUT

callback-ek

main

DisplayFunc

KeyboadFunc

KeyboadUpFunc

SpecialFunc

Reshape

MouseFunc

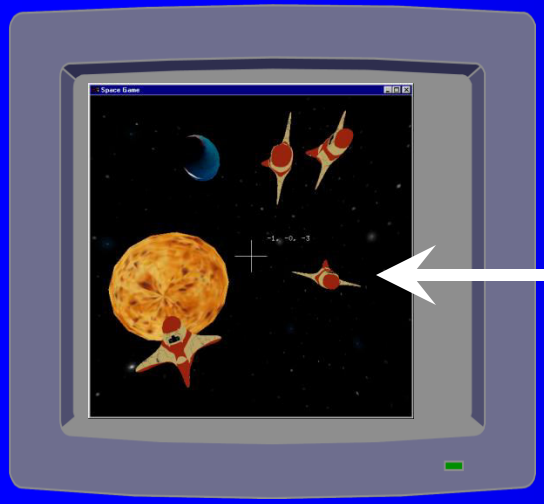
MotionFunc

IdleFunc

applikáció

OpenGL

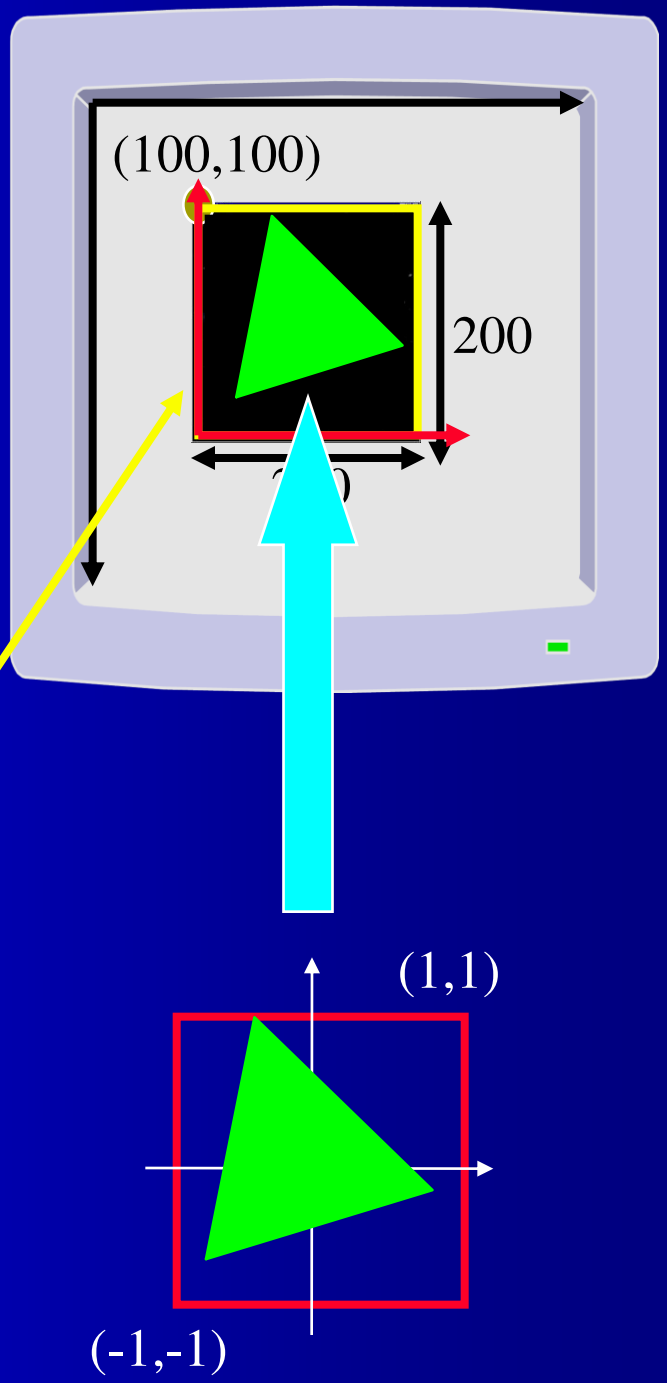
grafikus hardver



OpenGL alkalmazás (GLUT) main függvény

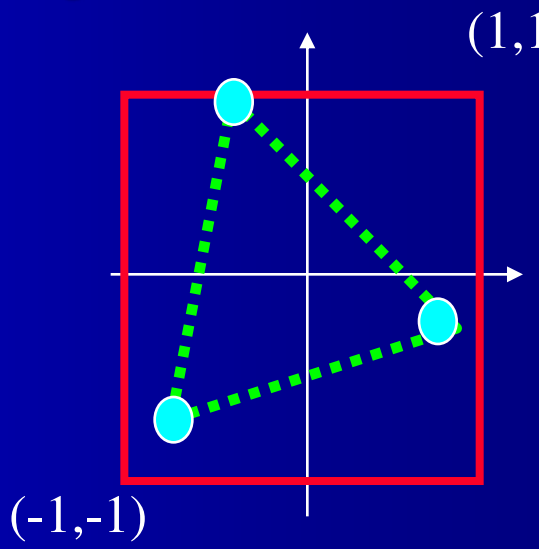
- Ablak megnyitás
- Rajzolási állapot inicializálás
- Display callback:
  - képernyő törlés, állapotváltás, rajzolás, buffercsere
- Reshape callback:
  - Nézet (viewport), kamera transzformácó átállítása
- Input callback (mouse, keyboard)
- Idle callback (animáció)
- Üzenethurok

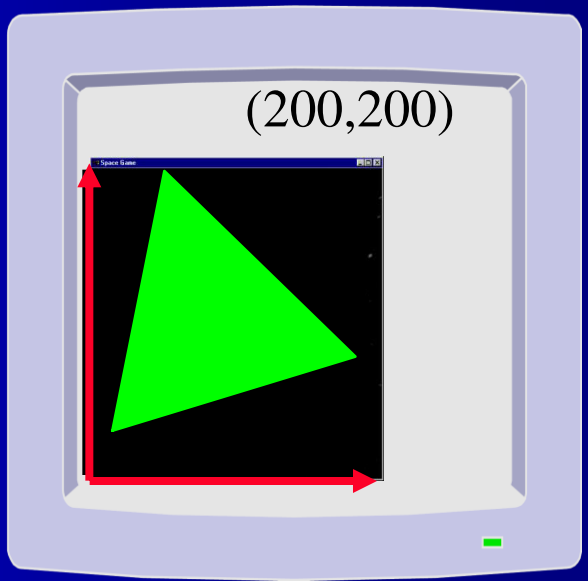
```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h> // download!!!

int awinw = 200, awinh = 200;
int viewpw = 200, viewph = 200;

main( int argc, char *argv[] ) {
    glutInitWindowSize(awinw, awinh);
    glutInitWindowPosition(100, 100);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB |
                        GLUT_DOUBLE );
    glutCreateWindow("Sample Window");
                // callback függvények
    glutMouseFunc( onMouse );
    glutDisplayFunc( onDisplay );
                // transzformáció
    glViewport(0, 0, viewpw, viewph);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity( );
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
                // fő hurok
    glutMainLoop();
}
```



Eseménykezelés és rajzolás

```
void onDisplay( ) {
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glutSwapBuffers( );
}

void onMouse(int button, int state,
             int x, int y) {
    if (button == GLUT_LEFT_BUTTON &&
        state == GLUT_DOWN) {
          float xc = 2.0*x/awinw - 1;
          float yc = 2.0*(awinh-y)/awinh - 1;
          glColor3f(0.0f, 1.0f, 0.0f);
          glBegin(GL_TRIANGLES);
               glVertex2f(xc, yc);
               glVertex2f(xc+0.2f, yc+1.8f);
               glVertex2f(xc+1.6f, yc+0.6f);
          glEnd( );
          glutSwapBuffers( );
     }
}
```





A háromszög újrarajzoláskor megmarad

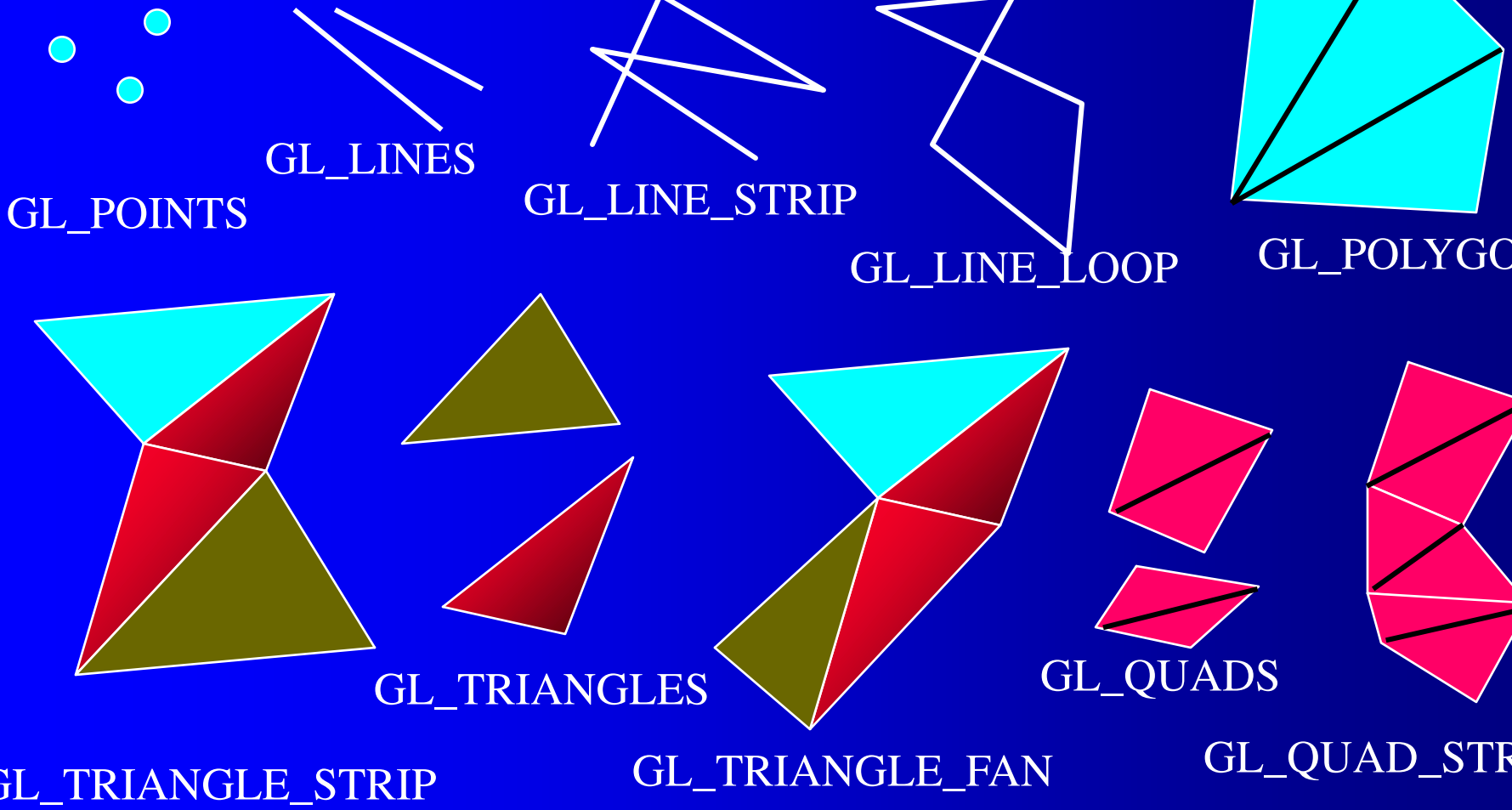
```
bool haromszog = false;
float xc, yc;

void onDisplay ( ) {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    if ( haromszog ) {
          glColor3f(0.0f, 1.0f, 0.0f);
          glBegin(GL_TRIANGLES);
             glVertex2f(xc, yc);
             glVertex2f(xc+0.2f, yc+1.8f);
             glVertex2f(xc+1.6f, yc+0.6f);
          glEnd( );
    }
    glutSwapBuffers();
}

void onMouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
          xc = 2.0*x/awinw - 1;
          yc = 2.0*(awinh-y)/awinh - 1;
          haromszog = true;
          onDisplay();
    }
}
```

Nem rajzol feleslegesen sokszor

```
bool haromszog = false;
float xc, yc;

void onDisplay ( ) {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    if ( haromszog ) {
          glColor3f(0.0f, 1.0f, 0.0f);
          glBegin(GL_TRIANGLES);
             glVertex2f(xc,yc);
             glVertex2f(xc+0.2, yc+1.8);
             glVertex2f(xc+1.6, yc+0.6);
          glEnd( );
    }
    glutSwapBuffers();
}

void onMouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
          xc = 2.0*x/awinw - 1;
          yc = 2.0*(awinh-y)/awinh - 1;
          haromszog = true;
          glutPostRedisplay();
    }
}
```

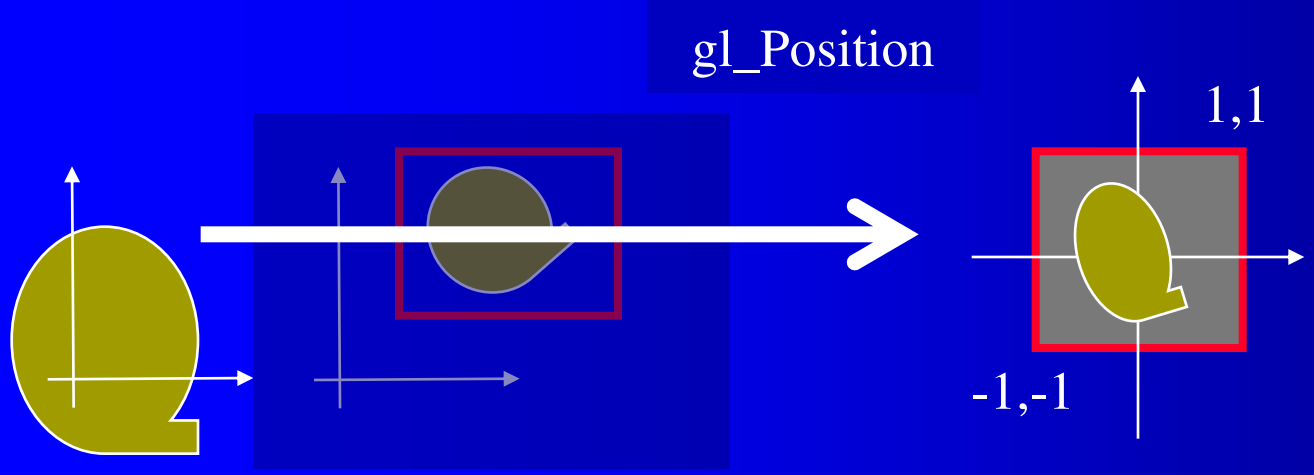
OpenGL: primitívek

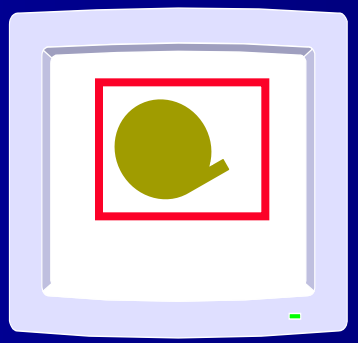
```
glBegin(primitív típus);
    Csúcspont tulajdonságok ...
    glVertex3f(x, y, z);
    ...
glEnd( );
```



Nagyfiúknak







OpenGL 3+

```
#include <GL/glew.h>
#include <GL/freeglut.h>

void main(int argc, char * argv[]) {
    glutInitWindowSize(200, 200);
    glutInitWindowPosition(100, 100);
    glutInit(&argc, argv);
    glutInitContextVersion(3, 0);
    glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
    glutCreateWindow("Sample Window");
    glutKeyboardFunc( onKeyboard );
    glutDisplayFunc( onDisplay ); // message loop
    glViewport(0, 0, 200, 200);

    glewInit();

    static float vertices[] = {-0.8,-0.8,  -0.6,1.0,  0.8,-0.2};
    GLuint vao;// vertex array object
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);
    GLuint vbo;// vertex buffer object
    glGenBuffers(1, &vbo); // Generate 1 buffer
    glBindBuffer(GL_ARRAY_BUFFER, vbo); // make it active, it is an array
    glBufferData(GL_ARRAY_BUFFER,sizeof(vertices),vertices,GL_STATIC_DRAW);
```

```
#version 130
in vec2 pos;

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

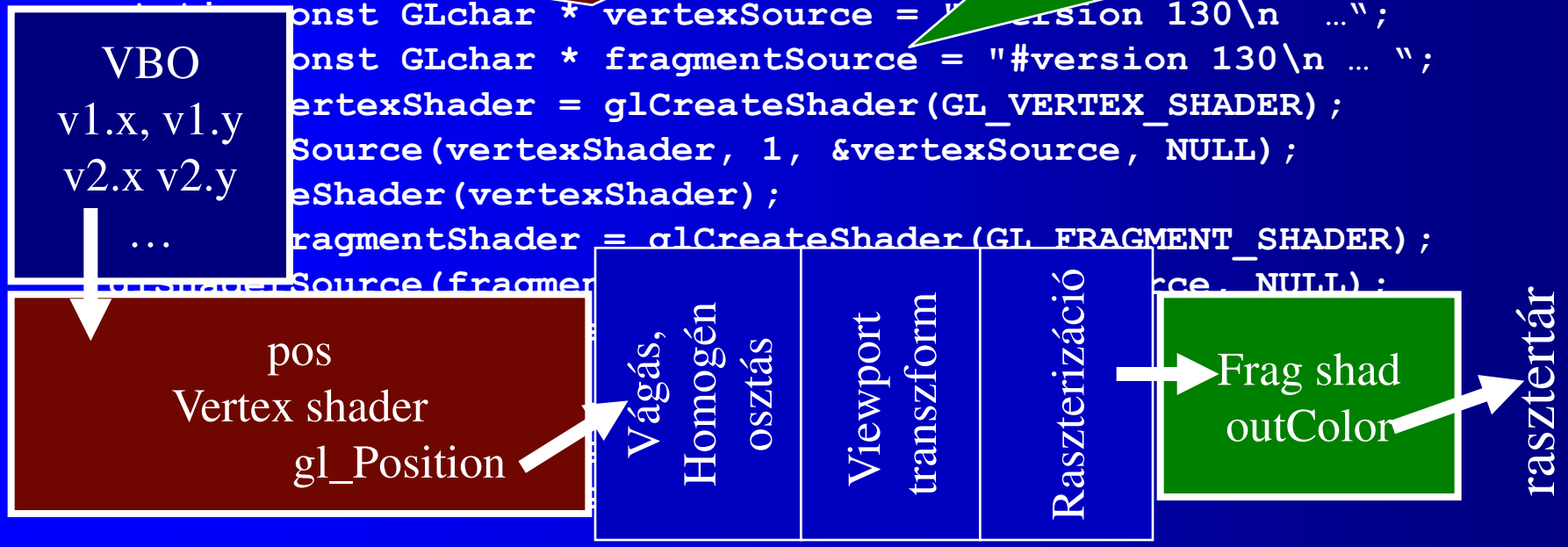
```
#version 130
precision mediump float;
out vec4 outColor;
void main() {
   outColor = vec4(0, 1, 0, 1);
}
```

```
static const GLchar * vertexSource = "#version 130\n  …";
static const GLchar * fragmentSource = "#version 130\n … ";
GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexSource, NULL);
glCompileShader(vertexShader);
GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentSource, NULL);
glCompileShader(fragmentShader);

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

glBindFragDataLocation(shaderProgram, 0, "outColor");

glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
glBindAttribLocation(shaderProgram, 0, "pos");

glLinkProgram(shaderProgram); glUseProgram(shaderProgram);

   glutMainLoop();
}
```

```
#version 130
in vec2 pos;

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

```
#version 130
precision mediump float;
out vec4 outColor;
void main() {
   outColor = vec4(0, 1, 0, 1);
}
```

```
onst GLchar * vertexSource = "ersion 130\n  …";
onst GLchar * fragmentSource = "#version 130\n … ";
ertexShader = glCreateShader(GL_VERTEX_SHADER);
Source(vertexShader, 1, &vertexSource, NULL);
eShader(vertexShader);
ragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
Source(fragmen                                  rce, NULL);
```




```
    glBindFragDataLocation(shaderProgram, 0, "outColor");

    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(0);
    glBindAttribLocation(shaderProgram, 0, "pos");

    glLinkProgram(shaderProgram); glUseProgram(shaderProgram);

    glutMainLoop();
}
```

Draw call

```
void onDisplay( ) {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    int startIdx = 0;
    int numOfElements = 3;

    glDrawArrays(GL_TRIANGLES,
                 startIdx,  numOfElements);

    glutSwapBuffers( );
}
```

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
// - faljmuveleteket vegezni (printf is fajlmuvelet!)
// - new operatort hivni az onInitialization-t kivéve, a lefoglalt adat felszabadítása nélkül
// - felesleges programsorokat a beadott programban hagyni
// - tovabbi kommenteket a beadott programba irni a forrasmegjelolest kommentjeit kiveve
// ---------------------------------------------------------------------------------------------
// A feladatot ANSI C++ … ,
// ---------------------------------------------------------------------------------------------
// A feladatmegoldasokban csak olyan gl/glu/glut fuggvenyek hasznalhatok, amelyek
// 1. Az oran a feladatkiadasig elhangzottak ES (logikai AND muvelet)
// 2. Az alabbi listaban szerepelnek: …
// ...
// NYILATKOZAT
// ---------------------------------------------------------------------------------------------
// Nev    : <VEZETEKNEV(EK)> <KERESZTNEV(EK)>
// Neptun : <NEPTUN KOD>
// ---------------------------------------------------------------------------------------------
// ezennel kijelentem, hogy a feladatot magam keszitettem, es ha barmilyen segitseget
// igenybe vettem …, akkor a forrast es az atvett reszt kommentekben egyertelmuen jeloltem ...
//=============================================================================================
#include <math.h>
#include <stdlib.h>
#if defined(__APPLE__)
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#include <GLUT/glut.h>
#else
#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__)
#include <windows.h>
#endif
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#endif
```

Házi

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~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_BUTTON && 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~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

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();// Esemenykezelo hurok
    return 0;
}
```

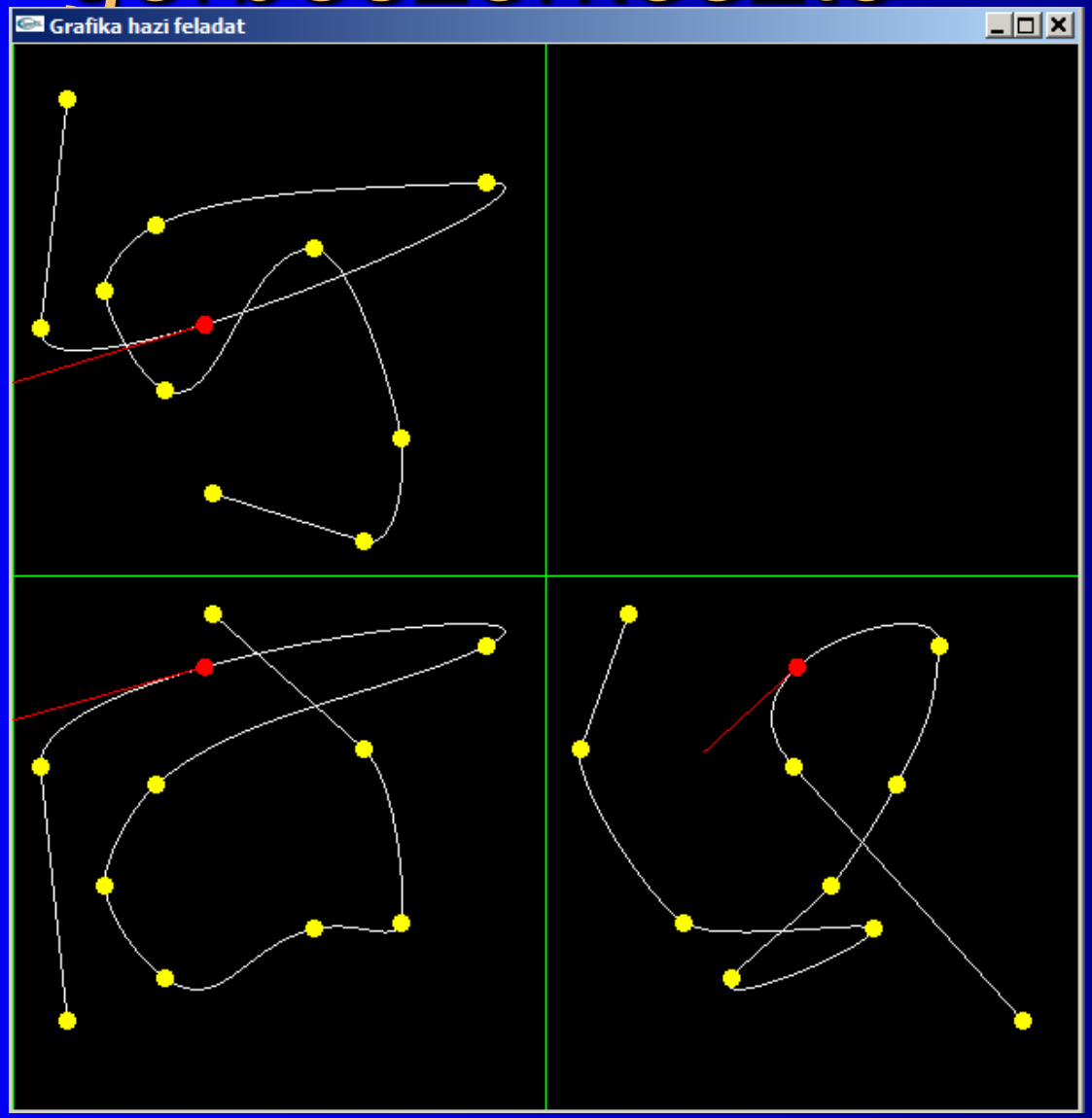
1. házi: Ortografikus görbeszerkesztő

Készítsen elől-felül-oldalnézetes ortografikus 3D Renner spline görbeszerkesztőt, amelyben a pálya menti sebesség közelítőleg 1 m/sec. A 600x600 pixel felbontású alkalmazói ablak négy egyenlő részre van osztva, amelyekben az elől, felül és oldalnézetek látható, egy negyed pedig üres. A világ 3m x 3m x 3m-res kocka, amelynek középpontjában vettük fel az origót. A nézetekben a teljes világ vetületei látszanak, az origó értelemszerűen középen van. A transzformációkat „kézzel” kell számítani, az OpenGL mátrixok most még nem alkalmazhatók.

Amíg még nincs görbénk, az egyik nézetbeli egérkattintások kontrolpontokat vesznek fel, úgy, hogy a nézetre merőleges koordináta zérus. A további nézetekbe kattintva az ott legközelebbinek talált kontrollpont áthelyezhető, a nézetre merőleges koordináta változatlanul hagyásával. A pillanatnyi állapotot minden nézetben jelezni kell, a kontrolpontokat sárga körökkel, a 3D Renner spline-t pedig a fehéren rajzolt 2D merőleges vetületeivel.

A SPACE lenyomására egy kör indul el a görbe kezdőpontjából és a sebességvektorát farokként maga után húzva, kb. 1 m/sec pályamenti sebességgel végigcsúszik a görbén.

1. házi: Ortografikus görbeszerkesztő



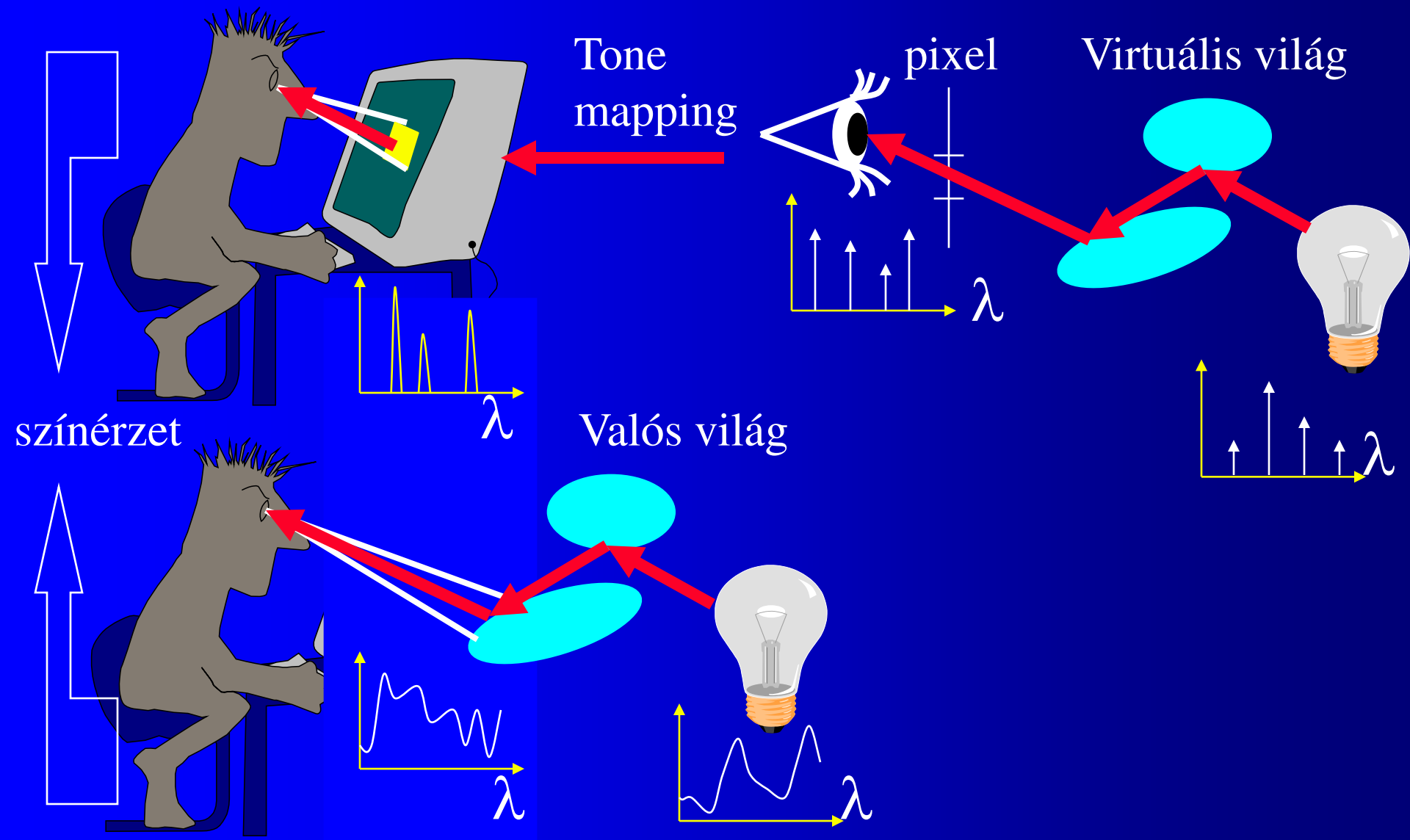
Science is either physics or stamp collecting.

Rutherford

3D képszintézis fizikai alapmodellje

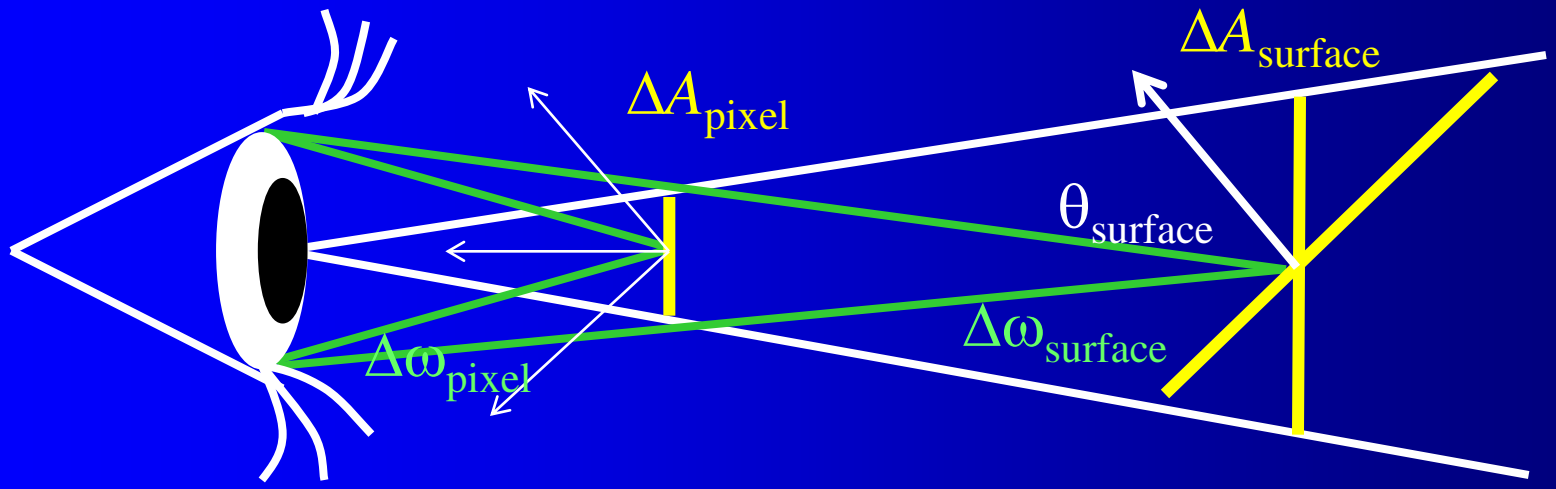
Szirmay-Kalos László

Képszintézis = valós világ illuziója



Sugársűrűség (Radiancia): $L(x,V)$

Wanted: $\Delta\Phi_{pixel} = \Delta\Phi_{surface}$



Geometria: $\Delta A_{pixel} \cos\theta_{pixel} \Delta\omega_{pixel} = \Delta A_{surface} \cos\theta_{surface} \Delta\omega_{surface}$

Radiancia = Egységnyi vetített terület által egységnyi térszögbe sugárzott teljesítmény

[W/sr/m^2]

$$L(x,V) = \frac{\Delta\Phi}{\Delta A \cos\theta \Delta\omega}$$

Fény-felület kölcsönhatás

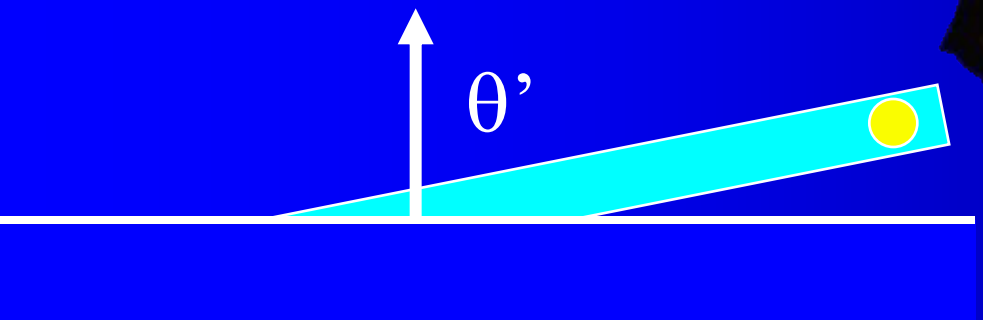
Sugársűrűség = Bejövő BRDF Geometria

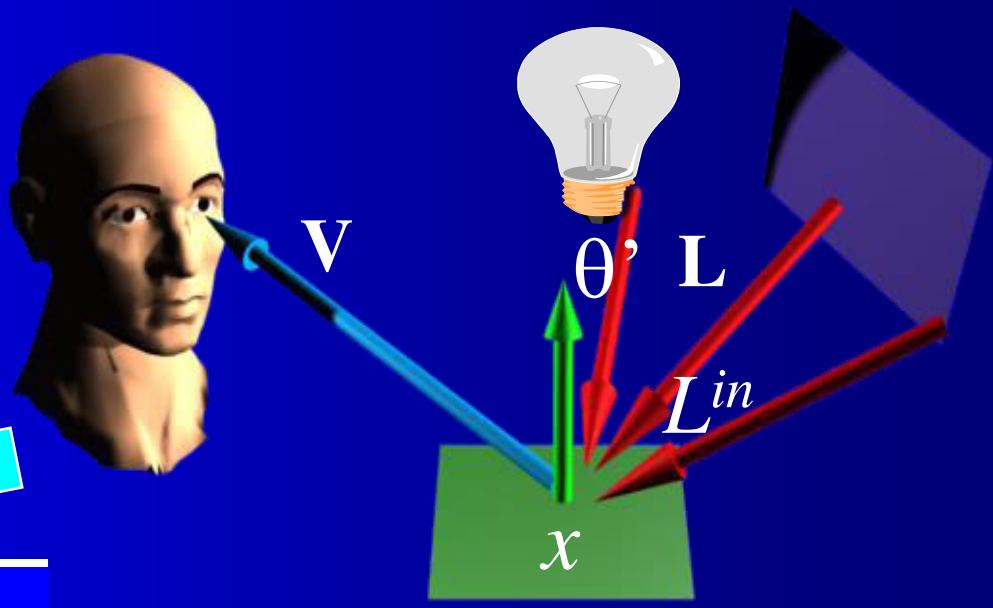
$$L(x,\mathbf{V}) = L^{in}(x,\mathbf{L}) \cdot f_r(\mathbf{L},x,\mathbf{V}) \cdot \cos\theta'$$

$$f_r(\mathbf{L},\mathrm{x},\mathbf{V}) \overset{\text{def}}{=} \frac{L(x,\mathbf{V})}{L^{in}(x,\mathbf{L})\cos\theta'}$$

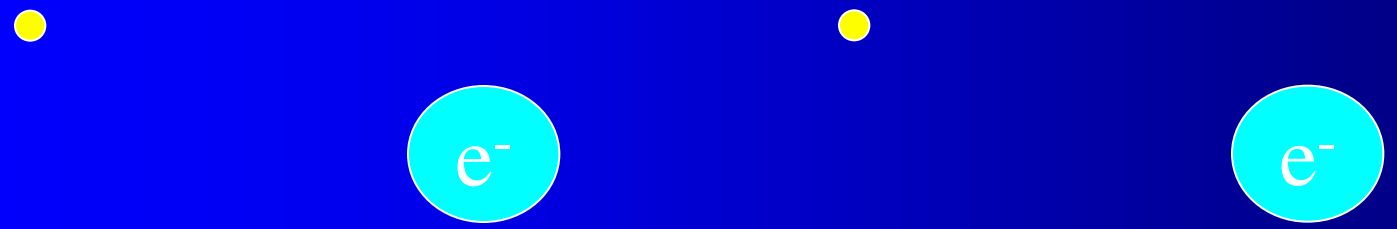
Helmholtz törvény:

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



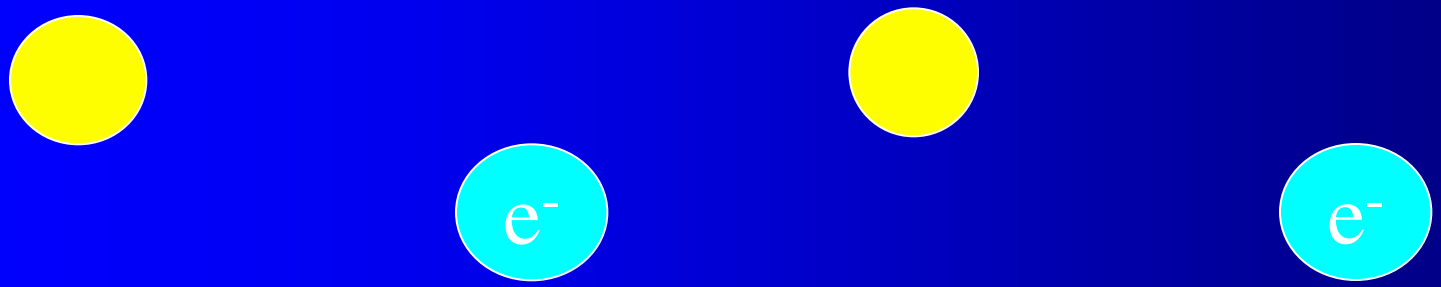


Fénynél a hullámhosszok külön kezelhetők



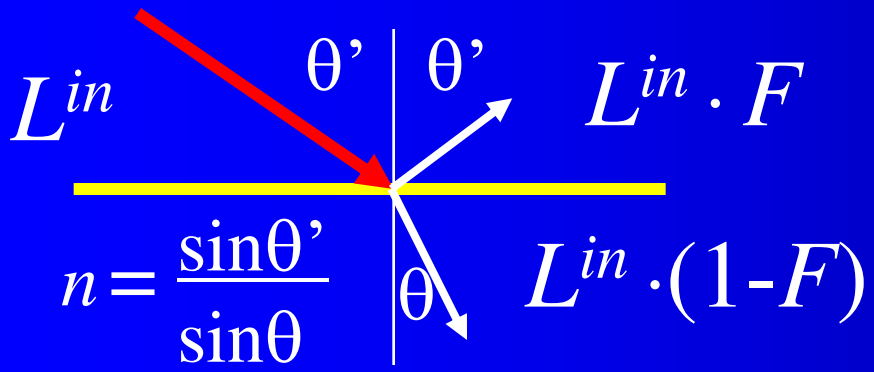
- Relativisztikus tömeg kicsi: $E = mc^2 = hf$
- A foton energia (hullámhossz) nem változik rugalmas ütközésnél
- Elnyelődési valószínűség energiafüggő

Nem mindig van így!



- Gamma fotonnál a relativisztikus tömeg az elektron tömegével összevethető
- Foton energia (hullámhossz) változik az ütközésnél (Compton szórás)
- Foszforeszkáló, fluoreszkáló anyagoknál sem kezelhetők a hullámhosszak függetlenül

Sima felületek
Fresnel egyenletek





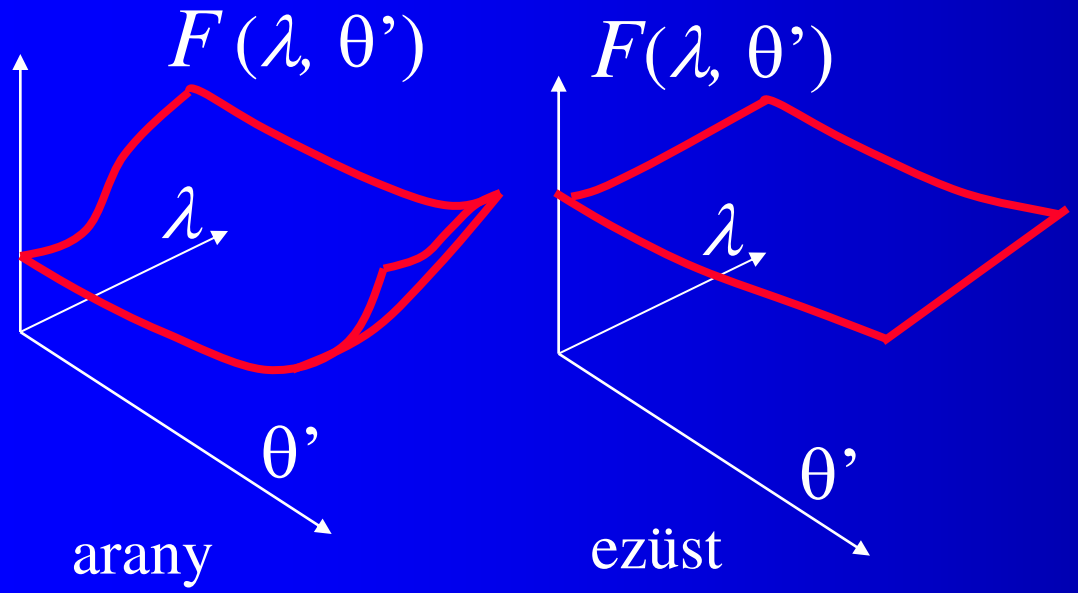


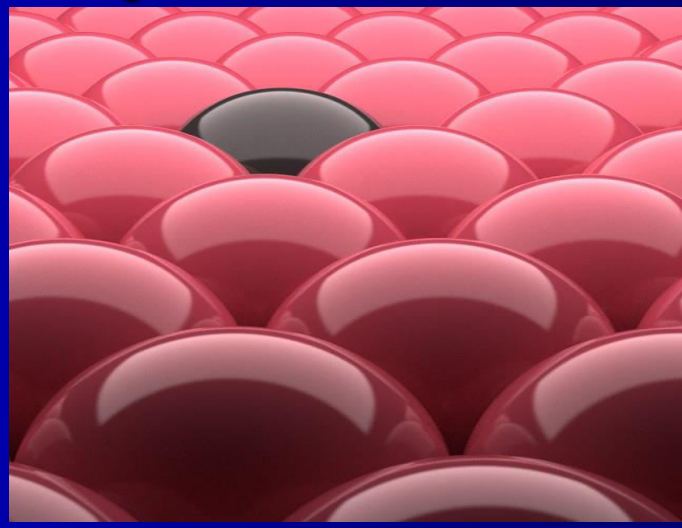
„Sima” = 1 pixelben látható felület síknak tekinthető

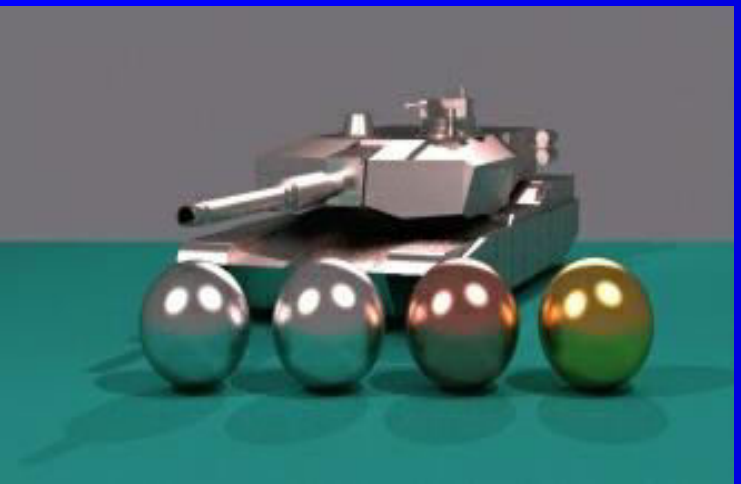
$$F = \frac{1}{2}\left|\frac{\cos\theta - (n+k\,j)\cos\theta'}{\cos\theta + (n+k\,j)\cos\theta'}\right|^2 + \frac{1}{2}\left|\frac{\cos\theta' - (n+k\,j)\cos\theta}{\cos\theta' + (n+k\,j)\cos\theta}\right|^2$$

$$F \approx F0 + (1-F0)\cdot(1-\cos\theta')^5, \qquad F0 = \frac{(n-1)^2 + k^2}{(n+1)^2 + k^2}$$

Fresnel függvény



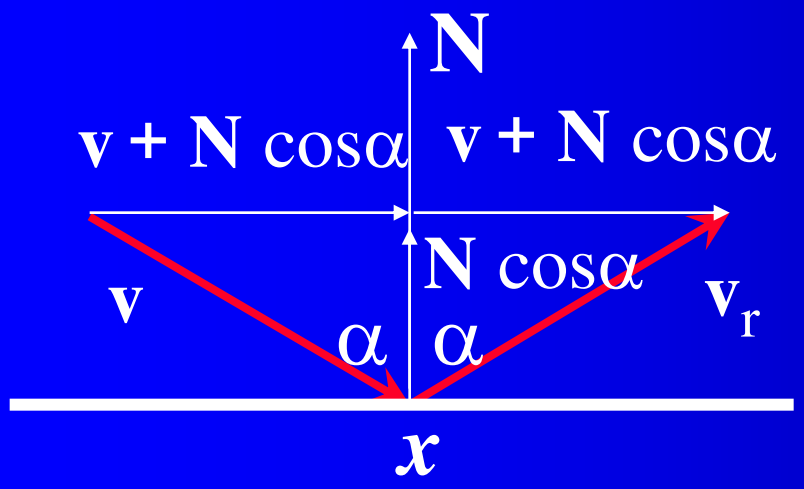








Tükörirány

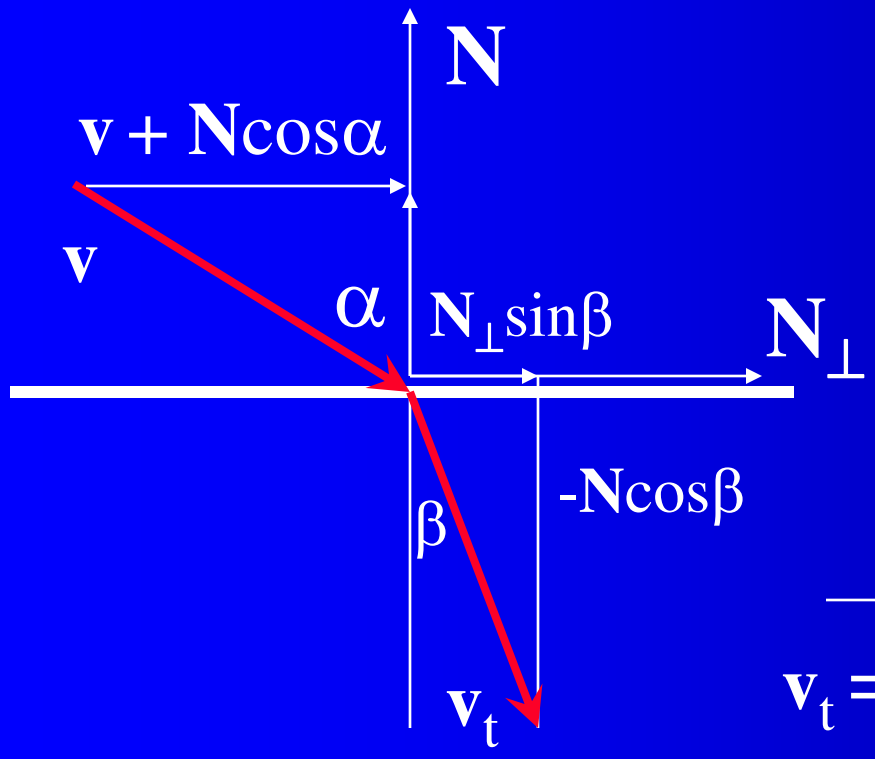


$$\cos\alpha = -(\mathbf{v}\cdot\mathbf{N})$$

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

```
float3 reflect(float3 inDir, float3 normal)
{
  return inDir - normal * dot(normal, inDir) * 2.0f;
};
```

Törési irány



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

Snellius-Descartes

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

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

$$v_t = v/n + N(\cos\alpha/n - \cos\beta)$$

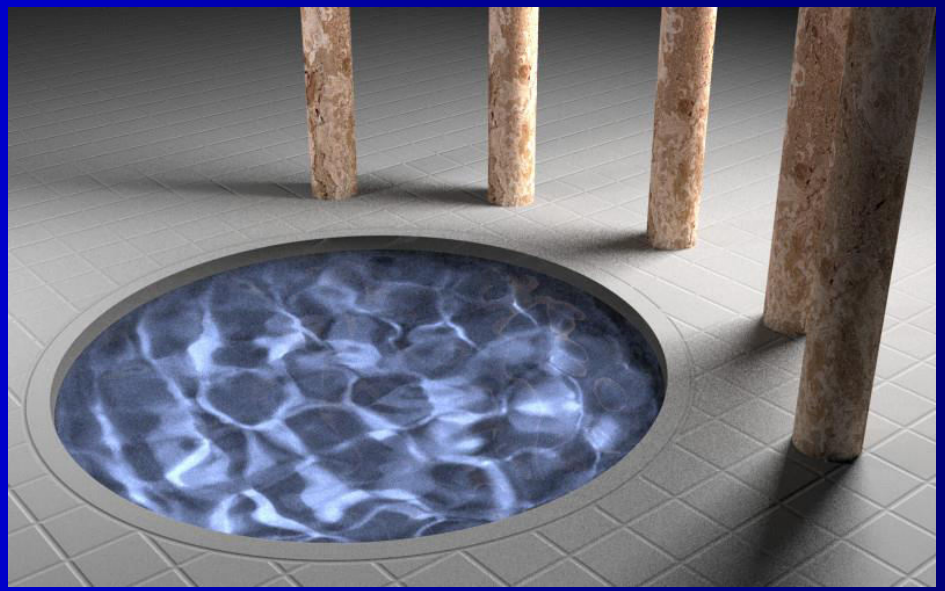
$$\cos\beta = \sqrt{1-\sin^2\beta} = \sqrt{1-\sin^2\alpha/n^2}$$

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

Fénytörő anyagok





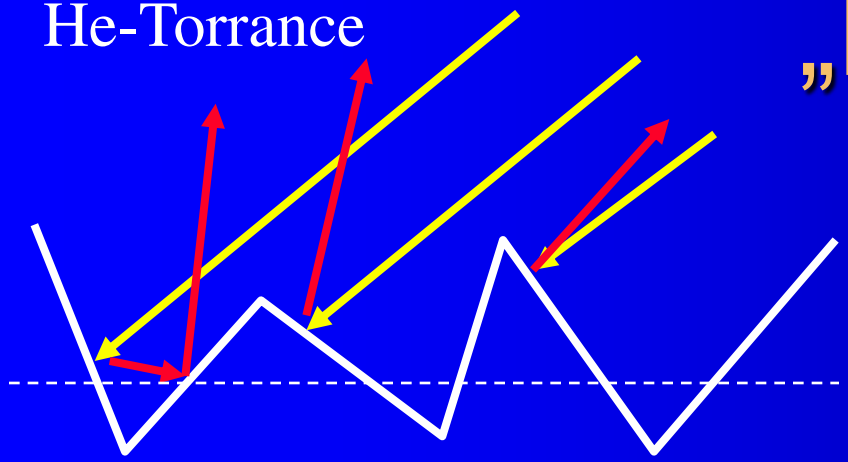


SmoothMaterial class

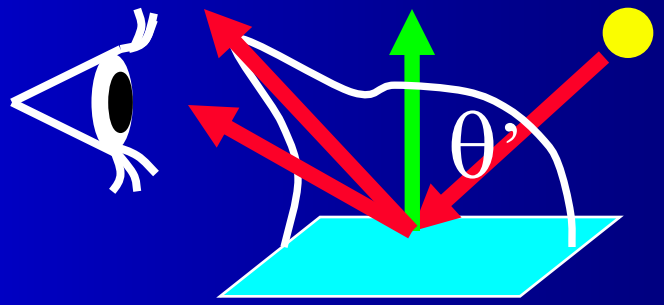
```
class SmoothMaterial {
  float3 F0; // F0
  float  n;  // n
public:
  float3 reflect(float3 inDir, float3 normal) {
     return inDir - normal * dot(normal, inDir) * 2.0f;
  }
  float3 refract(float3 inDir, float3 normal) {
    float ior = n;
    float cosa = -dot(normal, inDir);
    if (cosa < 0) { cosa = -cosa; normal = -normal; ior = 1/n; }
    float disc = 1 - (1 - cosa * cosa)/ior/ior;
    if (disc < 0) return reflect(inDir, normal);
    return inDir/ior + normal * (cosa/ior - sqrt(disc));
  }
  float3 Fresnel(float3 inDir, float3 normal) {
    float cosa = fabs(dot(normal, inDir));
    return F0 + (float3(1, 1, 1) - F0) * pow(1-cosa, 5);
  }
};
```

„Rücskös” felületek

Cook-Torrance
He-Torrance

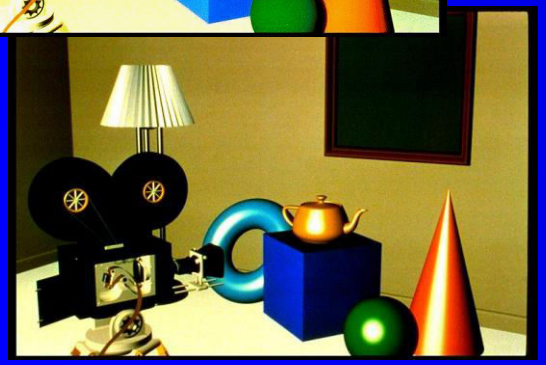


1 pixelben látható felület



Mi: viselkedésileg érvényes modell

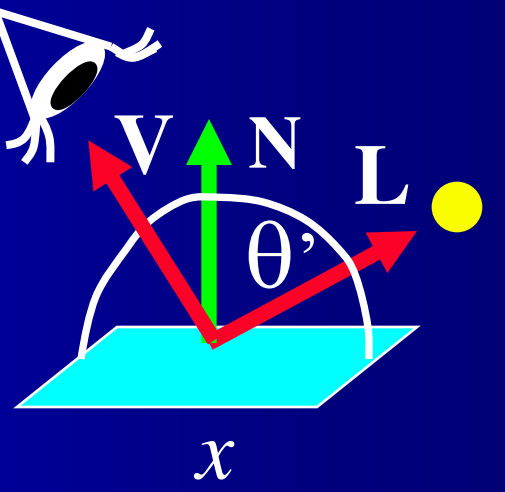








Diffúz visszaverődés



- Radiancia = Bejövő · BRDF · cosθ'
  a nézeti iránytól független
- A BRDF a nézeti iránytól független
- Helmholtz: a BRDF megvilágítási iránytól is független
- A BRDF irányfüggetlen:

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

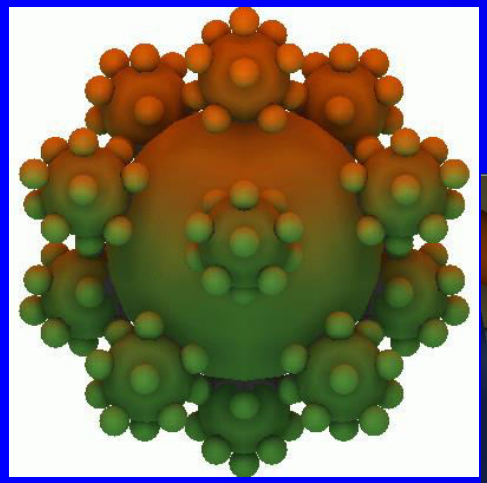
- Diffúz visszaverődés = nagyon rücskös
  - sokszoros fény-anyag kölcsönhatás
  - színes!

Lambert törvény

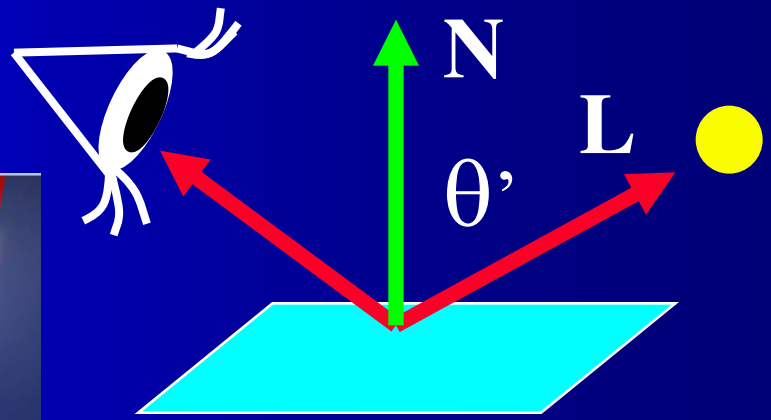
- Pont/irány fényforrásra válasz
  - BRDF irányfüggetlen,
    DE a sugársűrűség függ a megvilágítási iránytól

$$L^{ref} = L^{in}\ k_d \cos^{+}\theta'$$

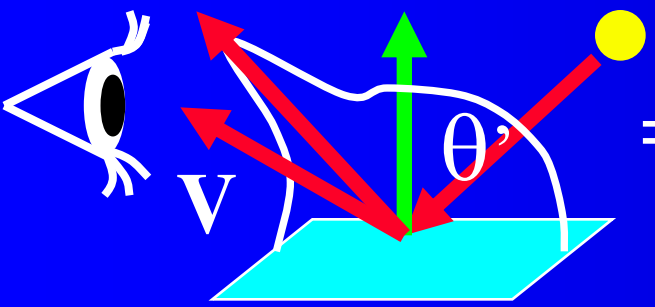
$$\cos\theta' = \mathbf{N}\cdot\mathbf{L}$$



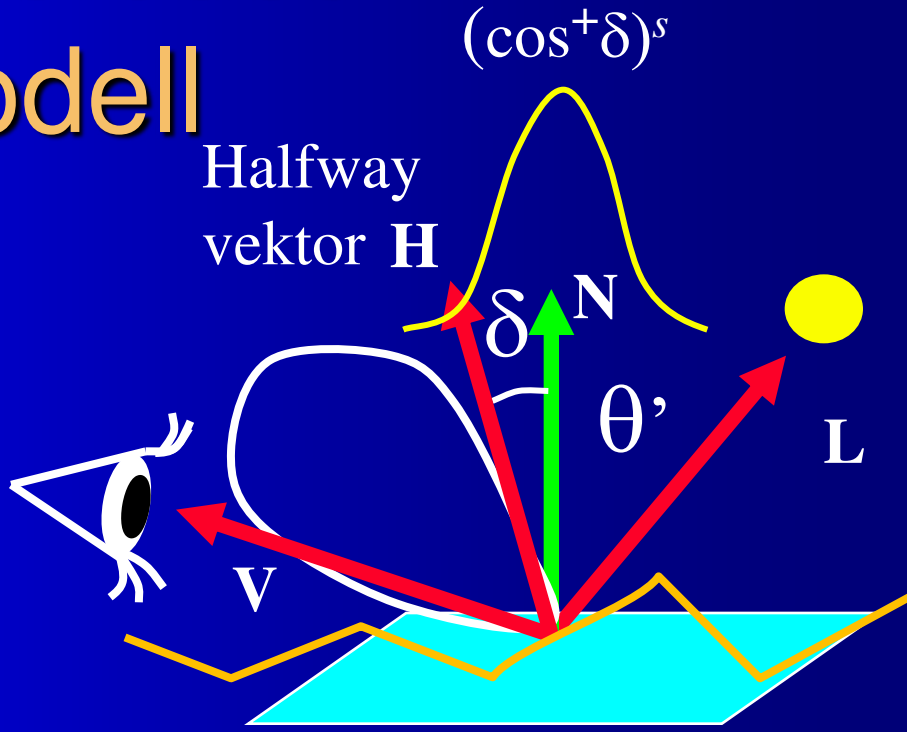




Spekuláris visszaverődés
Phong-Blinn modell



= diffúz +

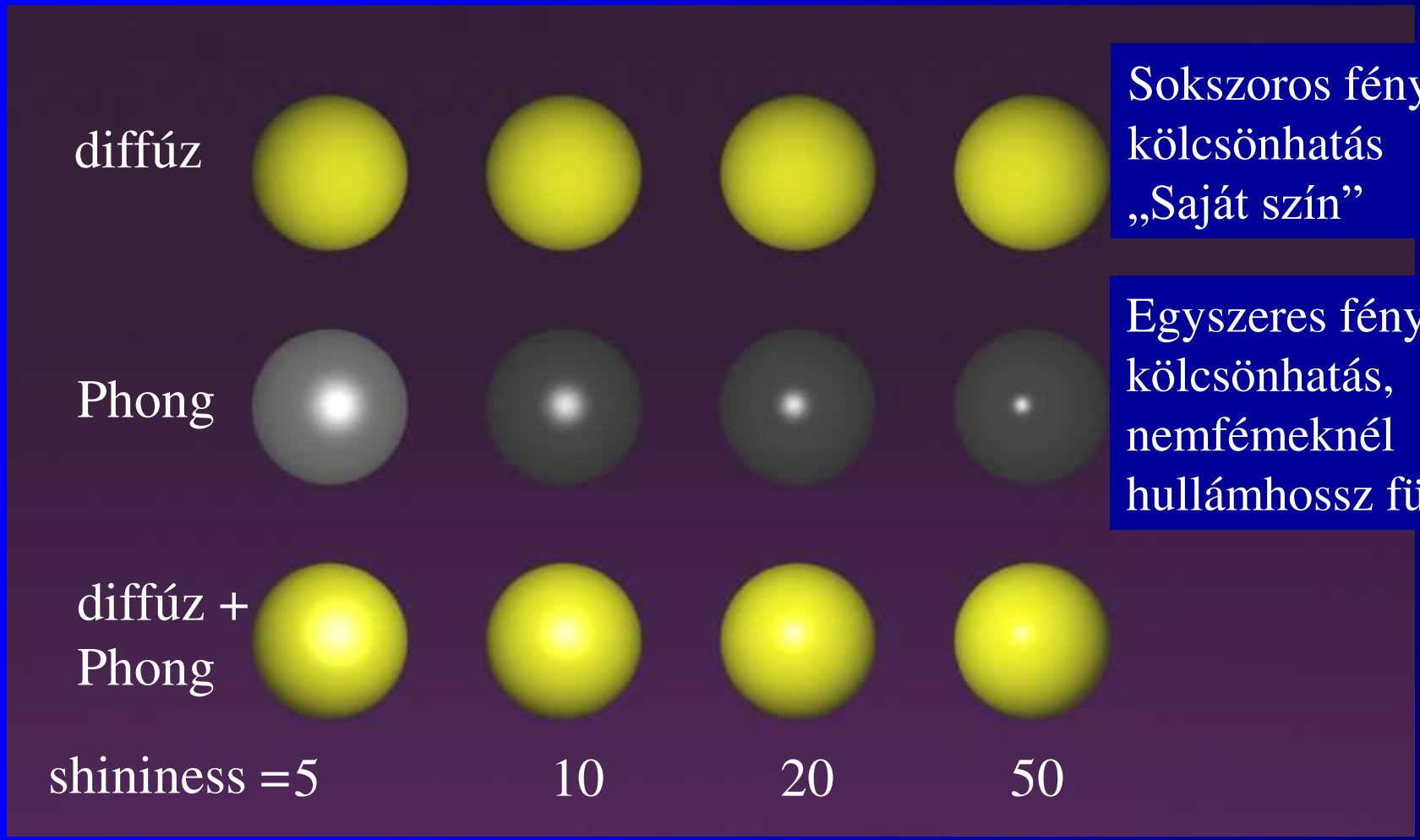


$$L^{ref} = L^{in}\ k_d \cos^{+}\theta' + L^{in}\ k_s\ (\cos^{+}\delta)^s$$

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

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

Diffúz+Phong anyagok



Sokszoros fény-anyag kölcsönhatás
„Saját szín”

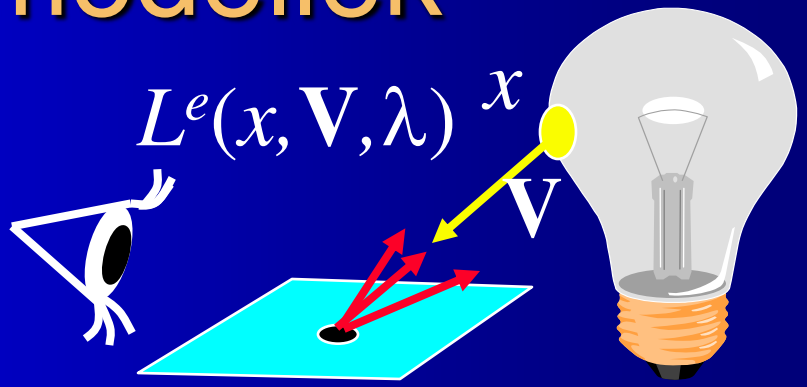
Egyszeres fény-anyag kölcsönhatás, nemfémeknél hullámhossz független

RoughMaterial class

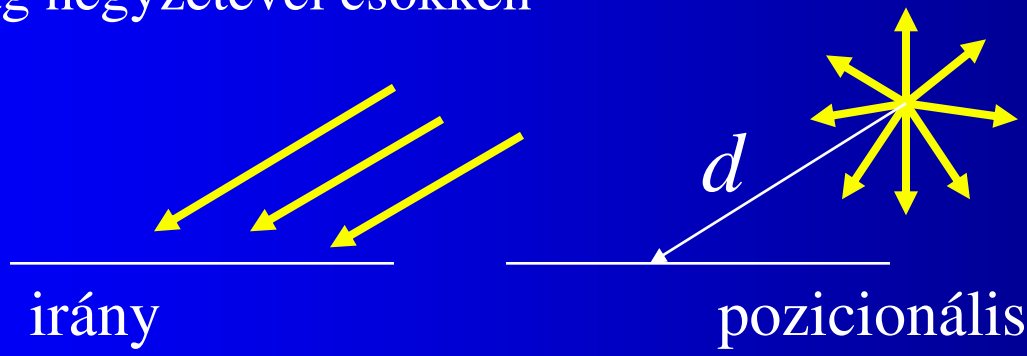
```
class RoughMaterial {
  float3 kd, ks;
  float  shininess;
public:
  float3 shade( float3 normal, float3 viewDir,
                float3 lightDir, float3 inRad)
  {
    float3 reflRad(0, 0, 0);
    float cosTheta = dot(normal, lightDir);
    if(cosTheta < 0) return reflRad;
    reflRad = inRad * kd * cosTheta;
    float3 halfway = (viewDir + lightDir).normalize();
    float cosDelta = dot(normal, halfway);
    if(cosDelta < 0) return reflRad;
    return reflRad + inRad * ks * pow(cosDelta,shininess);
  }
};
```

Fényforrás modellek

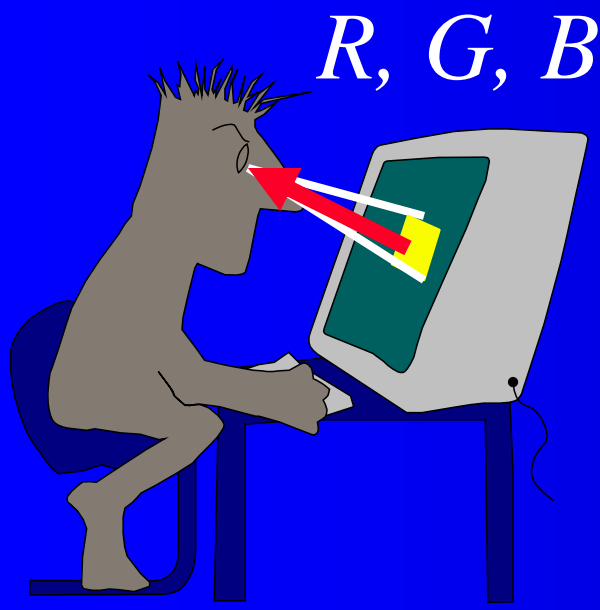
- **Geometria+sugárűsűség:**

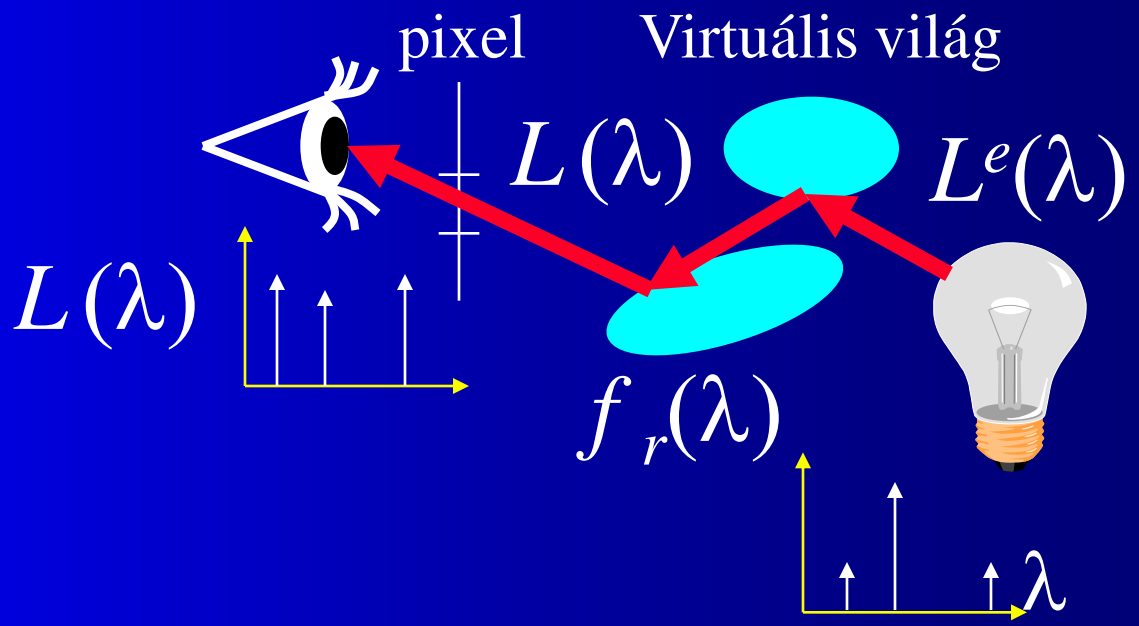


- **Absztrakt fényforrások:**
  - **Irány fényforrások**: egyetlen irányba sugároz, a fénysugarak párhuzamosak, az intenzitás független a pozíciótól
  - **Pozicionális fényforrás**: egyetlen pontból sugároz, az intenzitás a távolság négyzetével csökken



Képszintézis

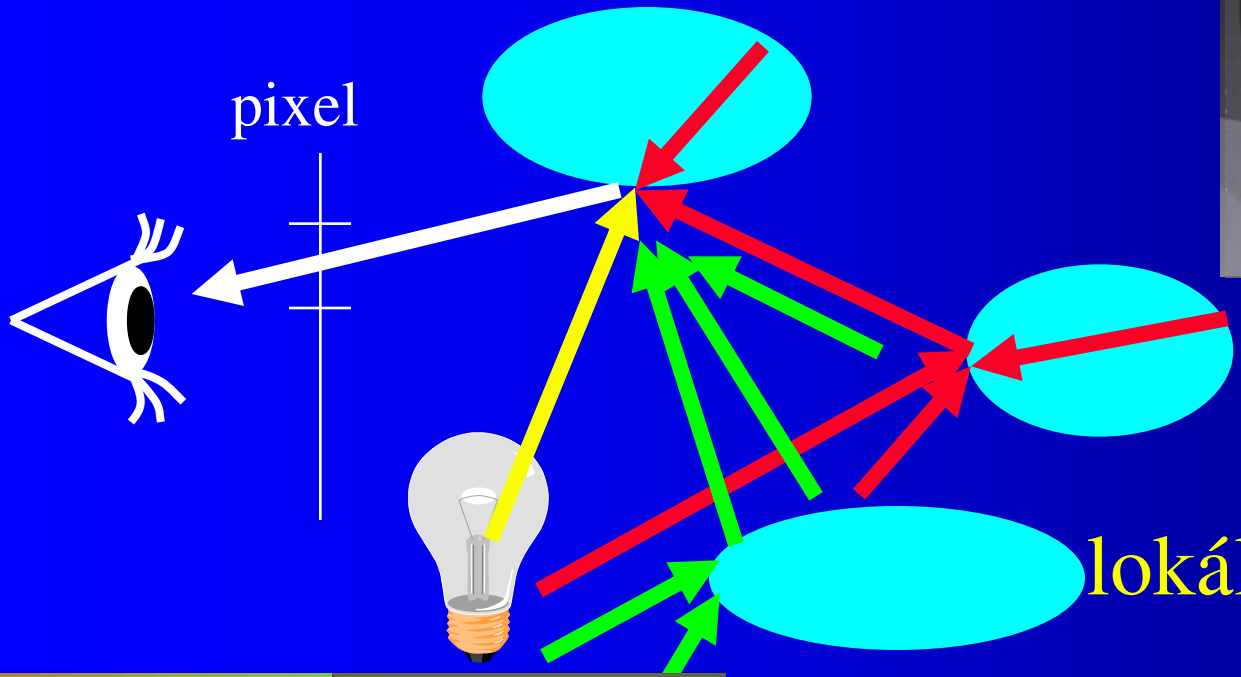




- Pixelben látható felület meghatározása
- A látható pont szem irányú sugársűrűsége
- R, G, B konverzió

Megoldási kompromisszumok

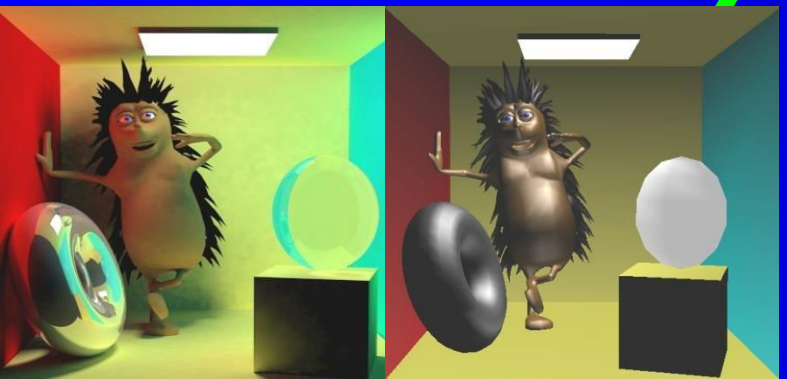




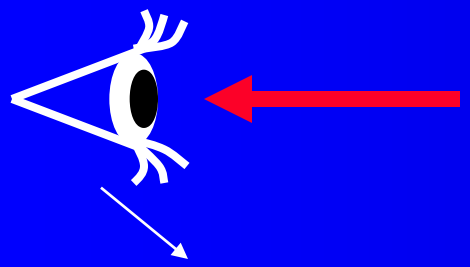
lokális illumináció

rekurzív sugárkövetés

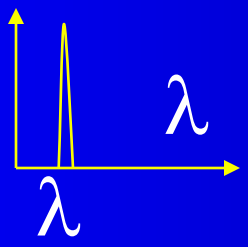
globális illumináció

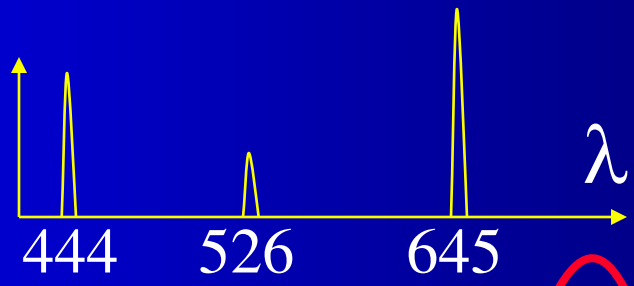


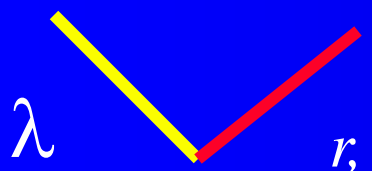
Színérzékelés: monokromatikus fény



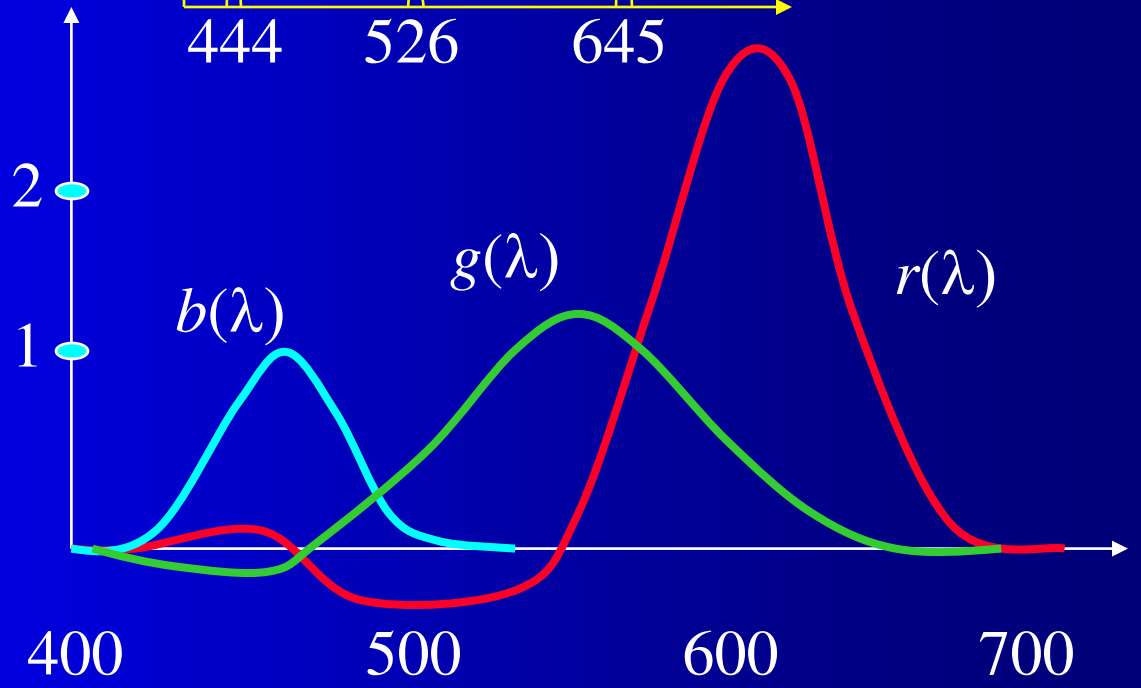
színérzet: *r, g, b*





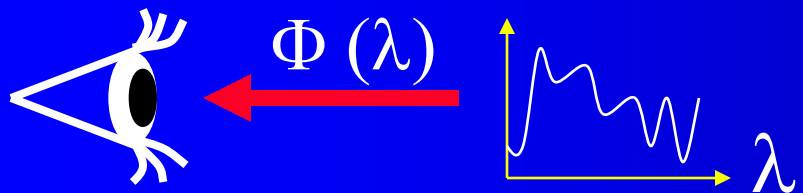






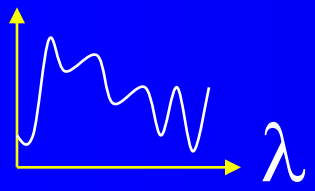


Színérzékelés: nem monokromatikus fény



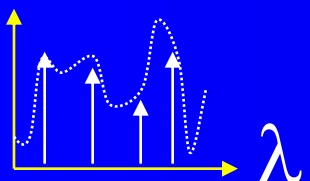
$$r = \int \Phi(\lambda)\, r(\lambda)\, d\lambda$$
$$g = \int \Phi(\lambda)\, g(\lambda)\, d\lambda$$
$$b = \int \Phi(\lambda)\, b(\lambda)\, d\lambda$$

Spektrális versus RGB



model

Képszintézis
sok hullámhosszon



Látható
spektrum

R, G, B
színillesztés

R, G, B kép

R, G, B

model

Képszintézis
3 hullámhosszon

R,G,B kép

Tone mapping

HRDI(R,G,B)⇨LDRI(r,g,b) konverzió

1. Luminancia:

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

2. Adaptált luminancia: I’
3. Relatív luminancia: $I_r = \frac{I}{I'}$
4. Megjelenített luminancia:

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

5. Szín:

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

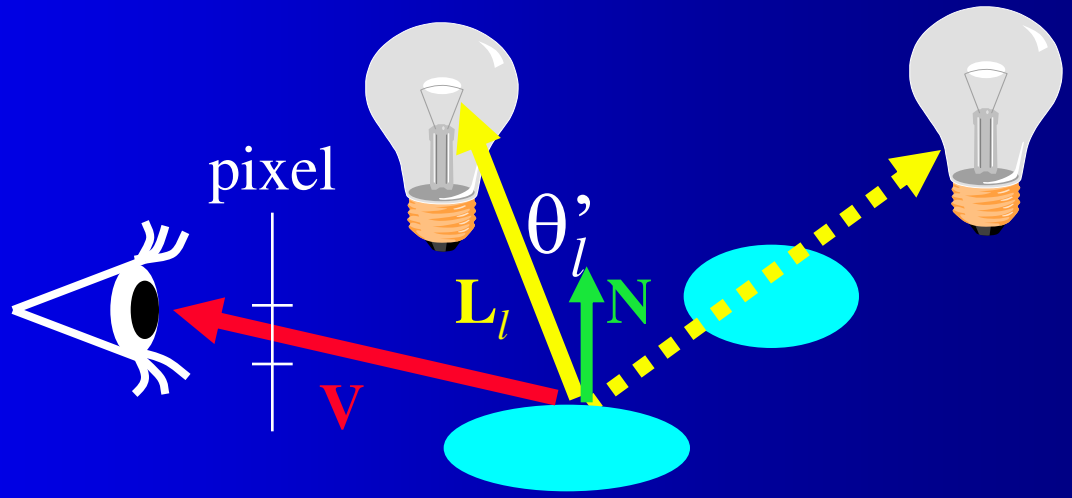


Sugárkövetés: ray-casting, ray-tracing

Szirmay-Kalos László

Lokális illuminációs (rücskös felület) absztrakt fényforrásokkal

Csak absztrakt fényforrások direkt megvilágítása

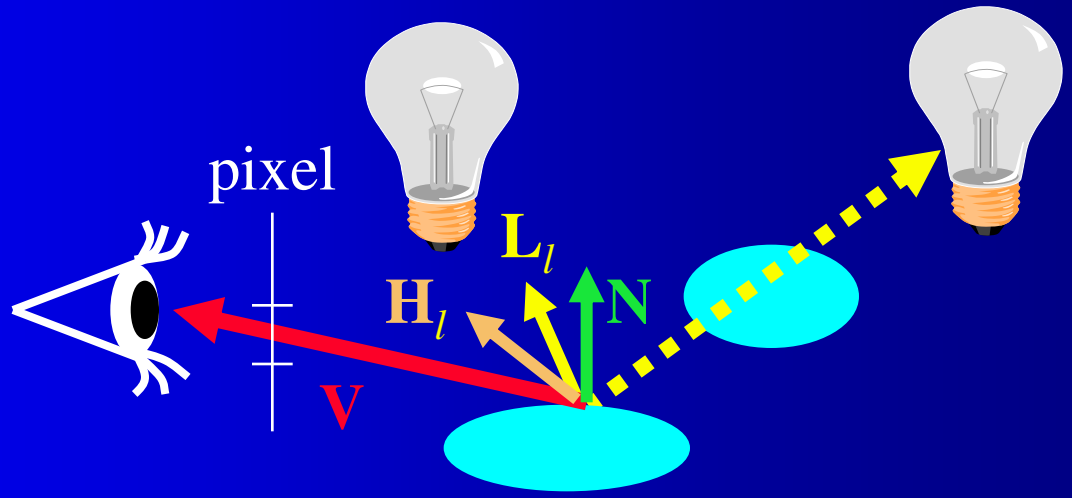


$$L(\mathbf{V}) \approx \sum_l L_l(\mathbf{L}_l) * f_r(\mathbf{L}_l, \mathbf{N}, \mathbf{V}) \cdot \cos\theta'_l$$

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

Lokális illumináció (rücskös felület) absztrakt fényforrásokkal

Csak absztrakt fényforrások direkt megvilágítása

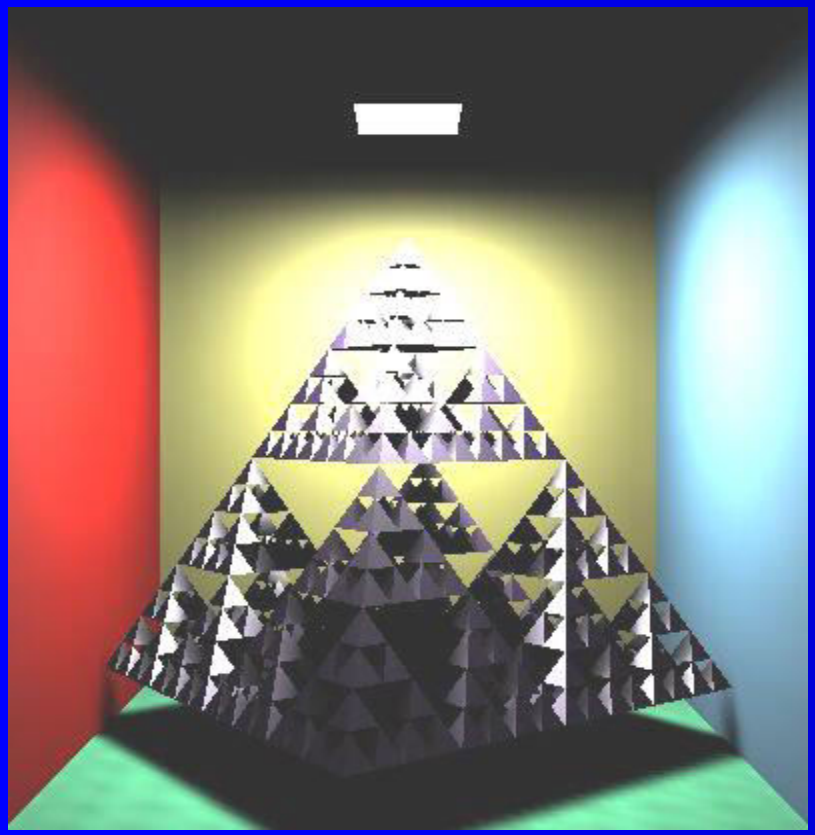


$$L(\mathbf{V}) \approx \sum_l L_l(\mathbf{L}_l) * \{k_d \cdot (\mathbf{L}_l \bullet \mathbf{N})^+ + k_s \cdot ((\mathbf{H}_l \bullet \mathbf{N})^+)^{shine}\}$$

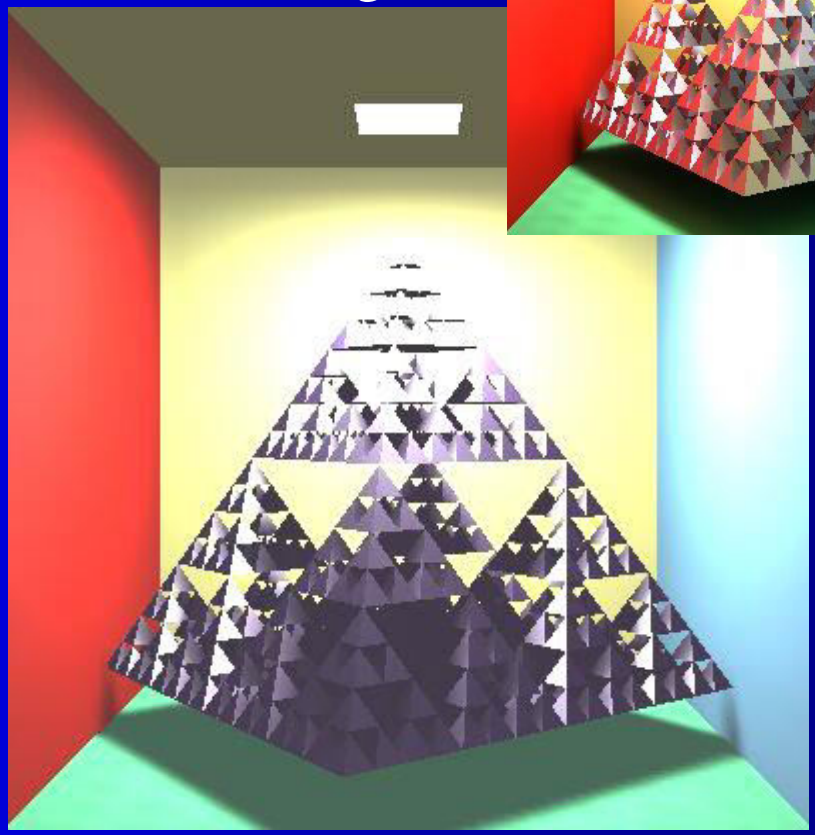
Absztrakt fényforrásokból származó megvilágítás.
(*Irányforrás = konstans; Pontforrás = távolság négyzetével csökken*
Ha takart, akkor zérus)

Ambiens tag

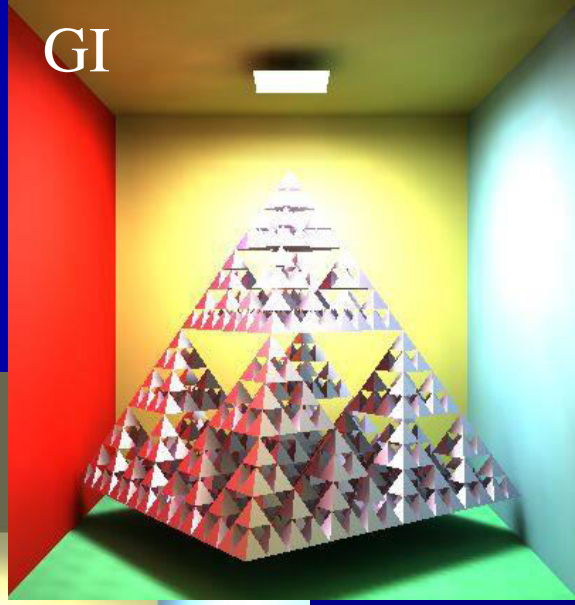
Lokális illumináció



+ ambiens tag

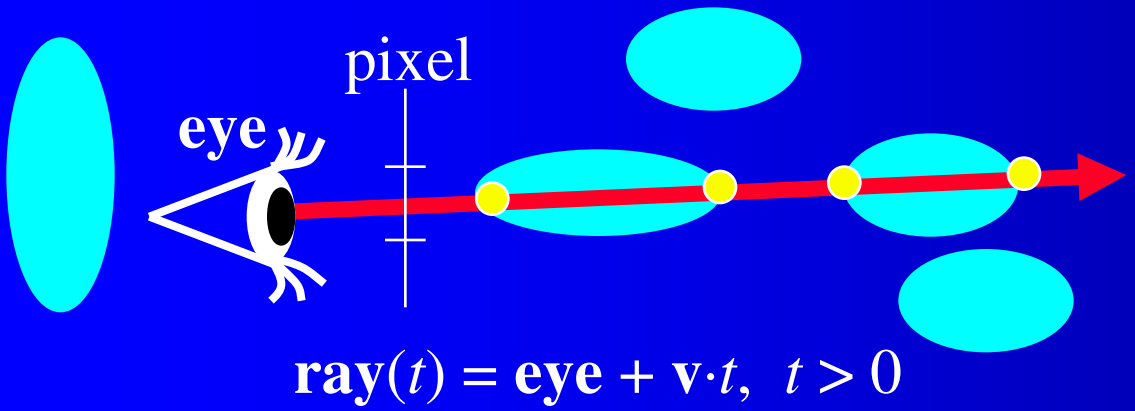


GI



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

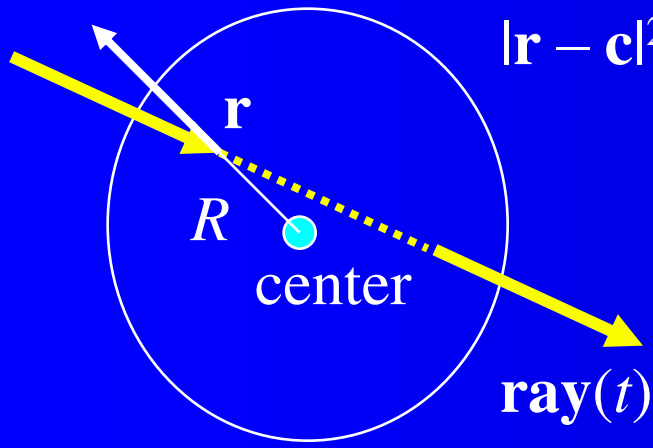
Láthatóság: trace a ray




```
struct Hit {
   float t;
   float3 position;
   float3 normal;
   Material* material;
   Hit() { t = -1; }
};
```

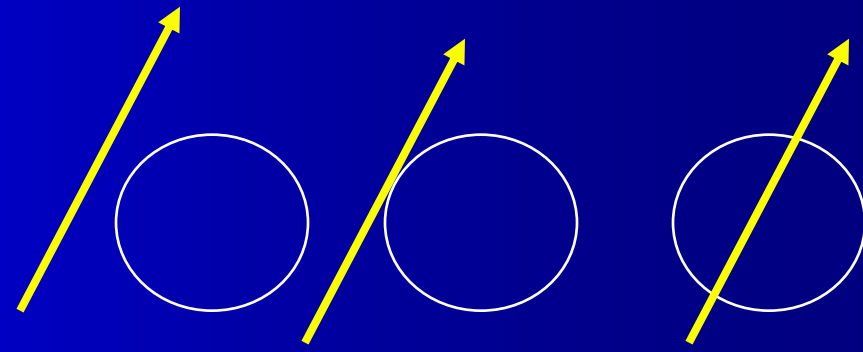
```
Hit firstIntersect(Ray ray) {
  Hit bestHit;
  for(Intersectable* obj : objects) {
    Hit hit = obj->intersect(ray); // hit.t < 0 if no intersection
    if(hit.t > 0 && (bestHit.t < 0 || hit.t < bestHit.t))
      bestHit = hit;
  }
  return bestHit;
}
```

Metszéspont számítás gömbbel



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

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



Nincs gyök 1 gyök 2 gyök

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

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

Wanted: a pozitív megoldások közül a kisebb

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

Sphere as Intersectable

```
class Intersectable
{
protected:
  Material* material;
public:
  virtual Hit intersect(const Ray& ray)=0;
};

class Sphere : public Intersectable {
  float3 center;
  float radius;
public:
  Hit intersect(const Ray& ray) {…}
};
```

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 = 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:

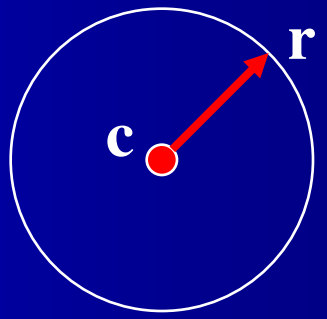
$$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)\bullet(x-x^*,\ y-y^*,\ z-z^*) = 0$$

Konkrét példa: gömb normálvektora

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

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

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



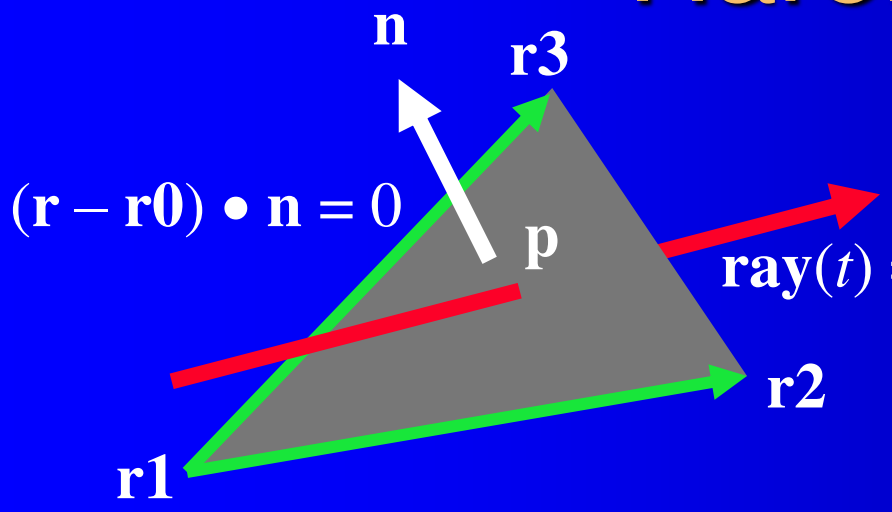
$$f(x,y,z) = (x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2 - R^2 = 0$$

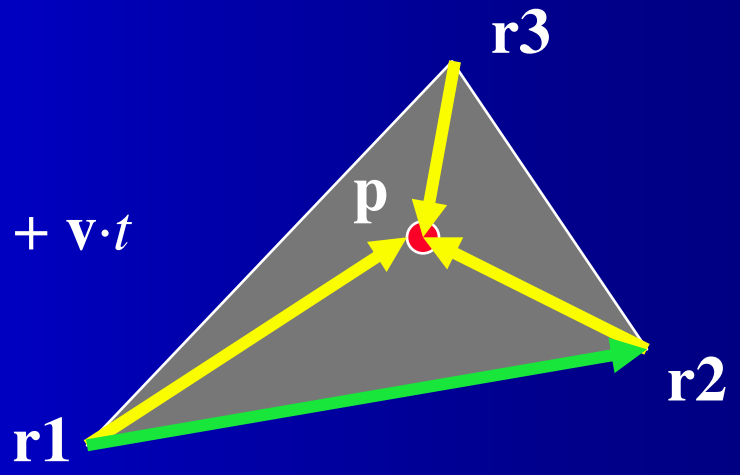
$$\frac{\partial f}{\partial x} = 2(x - c_x) + 0 + 0 - 0$$

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

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

Háromszög





1. Síkmetszés: $(\mathbf{ray}(t) - \mathbf{r1}) \bullet \mathbf{n} = 0, \quad t > 0$

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

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

2. A metszéspont a háromszögön belül van-e?

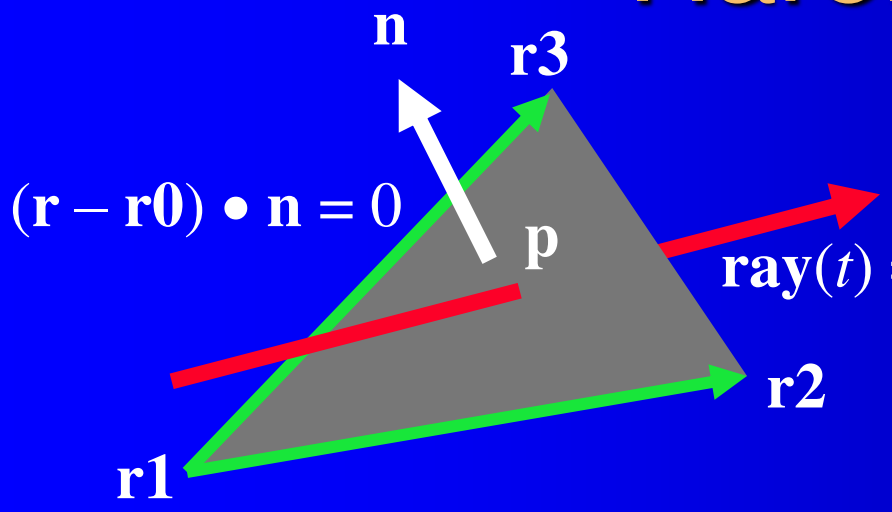
$((\mathbf{r2} - \mathbf{r1}) \times (\mathbf{p} - \mathbf{r1})) \bullet \mathbf{n} > 0$

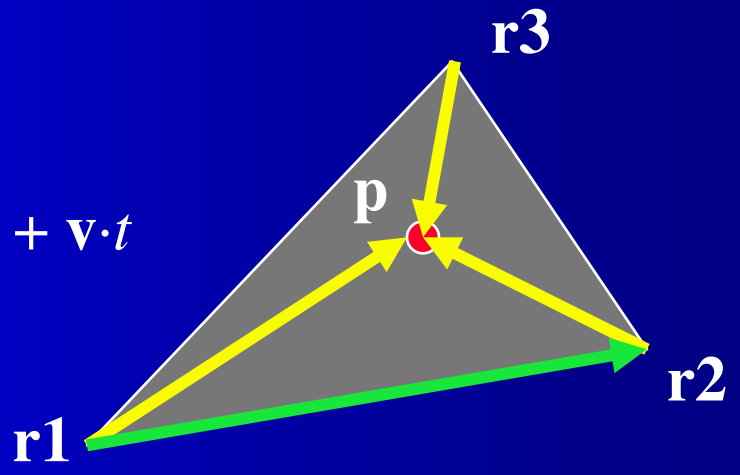
$((\mathbf{r3} - \mathbf{r2}) \times (\mathbf{p} - \mathbf{r2})) \bullet \mathbf{n} > 0$

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

Felületi normális: **n**
vagy árnyaló normálok
(shading normals)

Háromszög





1. Síkmetszés: $(\mathbf{ray}(t) - \mathbf{r1}) \bullet \mathbf{n} = 0, \quad t > 0$

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

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

2. A metszéspont a háromszögön belül van-e?

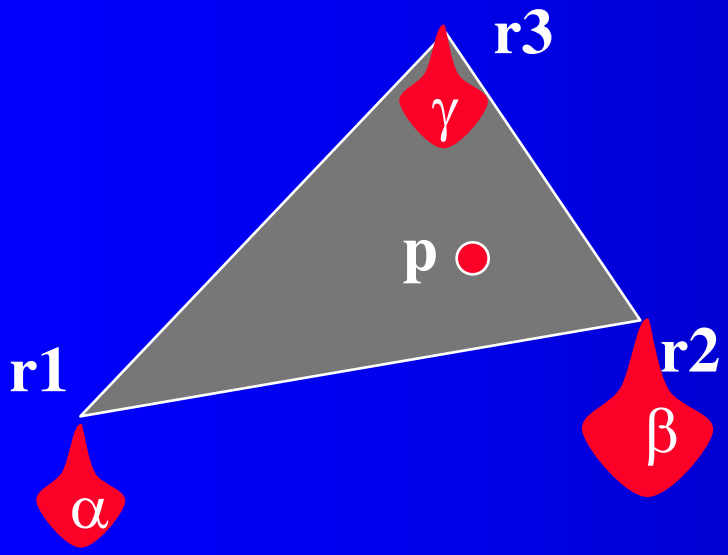
$((\mathbf{r2} - \mathbf{r1}) \times (\mathbf{p} - \mathbf{r1})) \bullet \mathbf{n} > 0$

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

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

Felületi normális: **n**
vagy árnyaló normálok
(shading normals)

Háromszög metszés baricentrikus koordinátákban



$\mathbf{p}(\alpha, \beta, \gamma) = \alpha \cdot \mathbf{r1} + \beta \cdot \mathbf{r2} + \gamma \cdot \mathbf{r3}$

$\alpha+\beta+\gamma = 1$ 1 skalár egyenlet

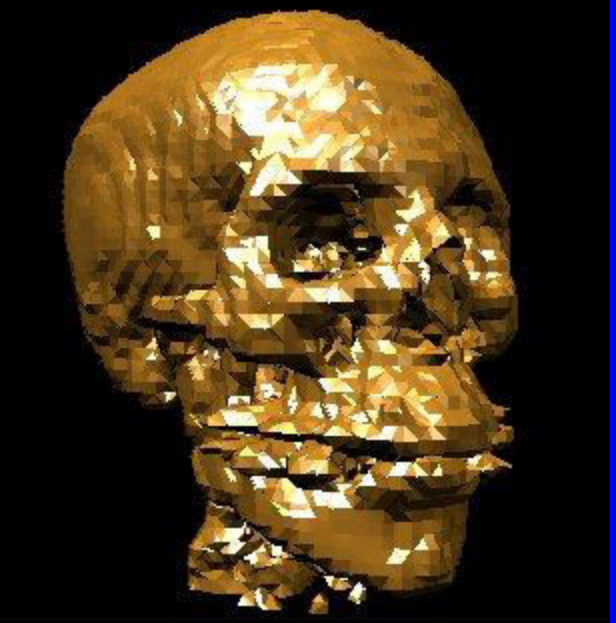
$\alpha, \beta, \gamma \geq 0$

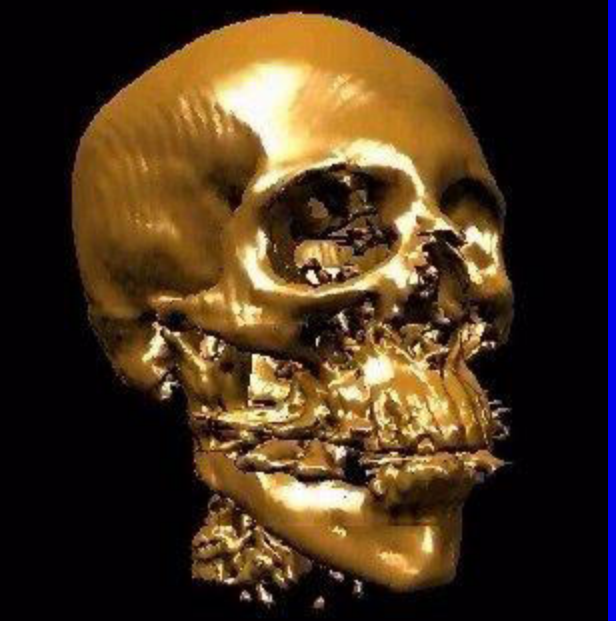
$\mathbf{ray}(t) = \mathbf{p}(\alpha, \beta, \gamma)$

$\mathbf{eye} + \mathbf{v} \cdot t = \alpha \cdot \mathbf{r1} + \beta \cdot \mathbf{r2} + \gamma \cdot \mathbf{r3}$ 3 skalár egyenlet

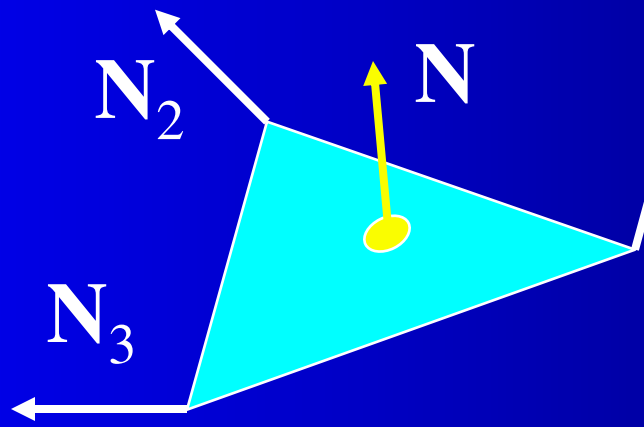
4 ismeretlen: α, β, γ, t

A megoldás után ellenőrzés, hogy mind nem negatív-e





Árnyaló normálok

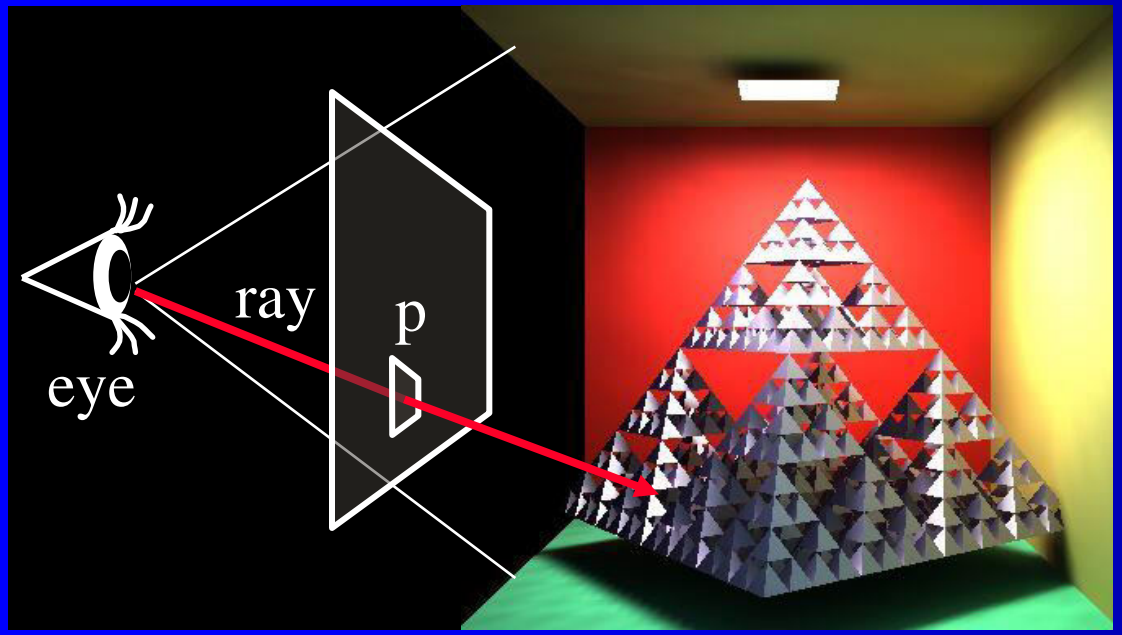


közelített felület normálvektora

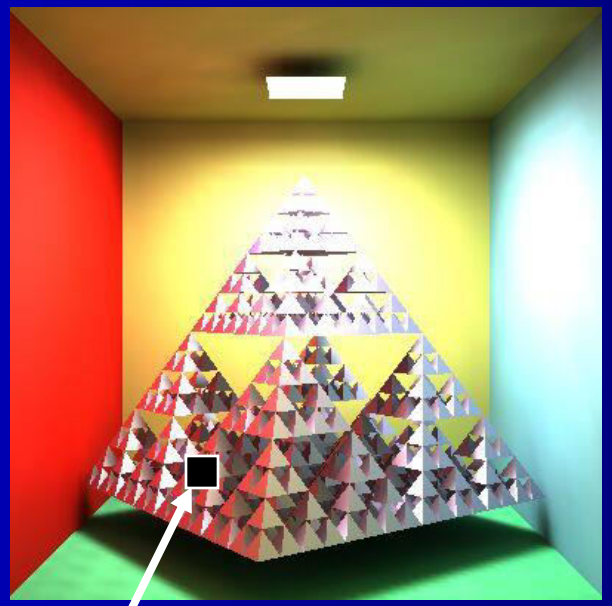
$$\mathbf{N}(\alpha, \beta, \gamma) = \alpha\mathbf{N}_1 + \beta\mathbf{N}_2 + \gamma\mathbf{N}_3$$

Ray tracing: Render

Virtual world: eye+window



Real world: user+screen

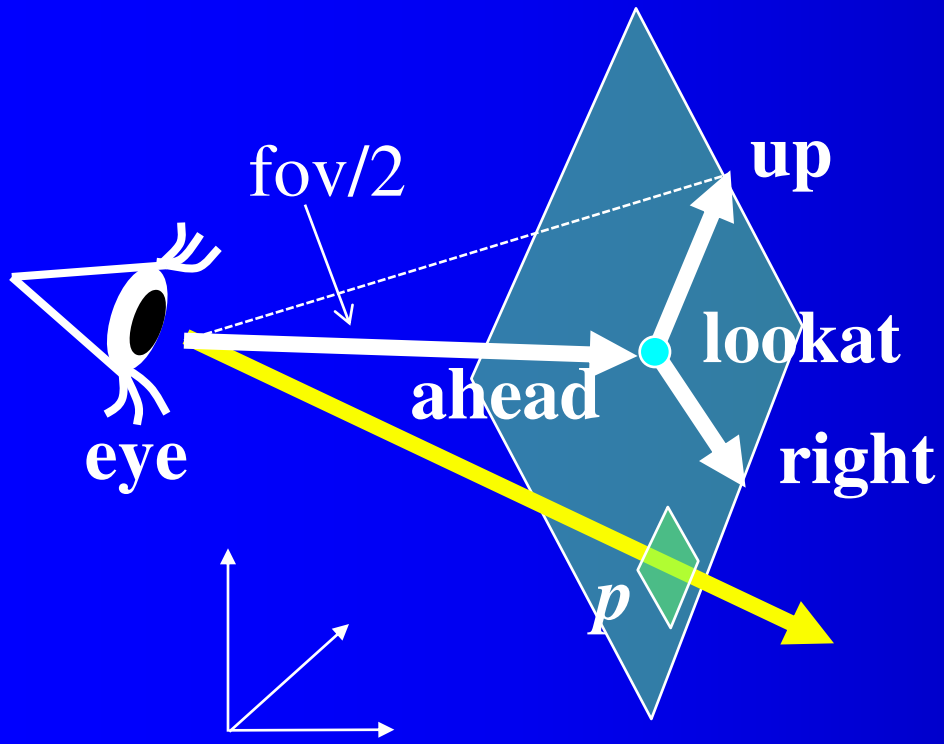


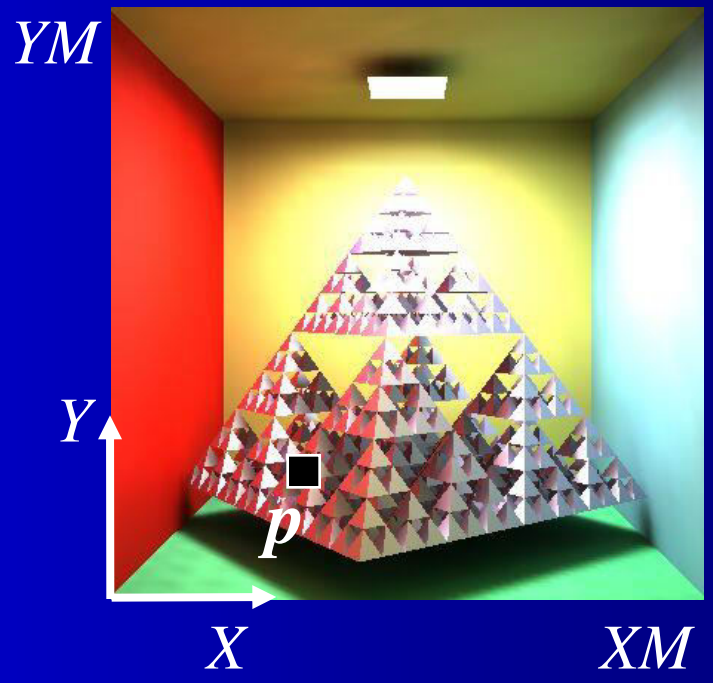
p color

Render( )

```
    for each pixel p
        Ray r = GetRay( eye ⇨ pixel p )
        color = trace(ray)
        WritePixel(p, color)
    endfor
end
```

Kamera: GetRay





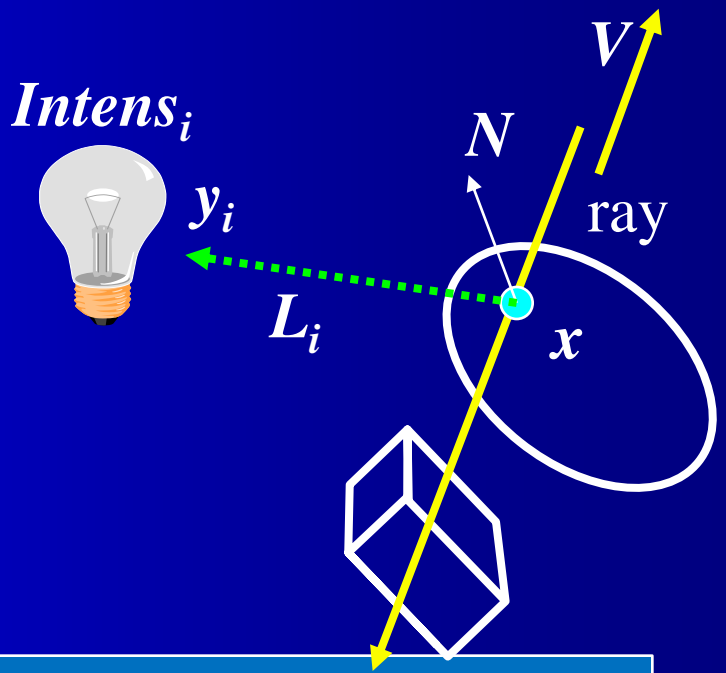
Normalizált eszköz koordináták

$\boldsymbol{p}$ = **lookat** + $\alpha \cdot$**right** + $\beta \cdot$**up**, α, β in [-1,1]

= **lookat** + $(2X/XM-1)\cdot$**right** + $(2Y/YM-1)\cdot$**up**

Ray dir =p – eye=ahead+$(2X/XM-1)\cdot$**right+**$(2Y/YM-1)\cdot$**up**

trace

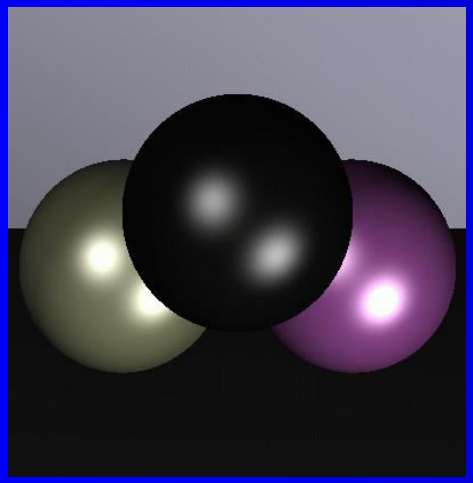



```
float3 trace(Ray ray) {

  Hit hit = firstIntersect(ray);
  if(hit.t < 0) return La; // nothing
  float3 outRadiance (0, 0, 0);
  for(each light source i){
    Ray shadowRay(x + Nε, Li);
    Hit shadowHit = firstIntersect(shadowRay);
    if(shadowHit.t < 0 || shadowHit.t > |x - yi| )
      outRadiance += hit.material->shade(N,V,Li, Intensi);
  }
  return outRadiance;
}
```

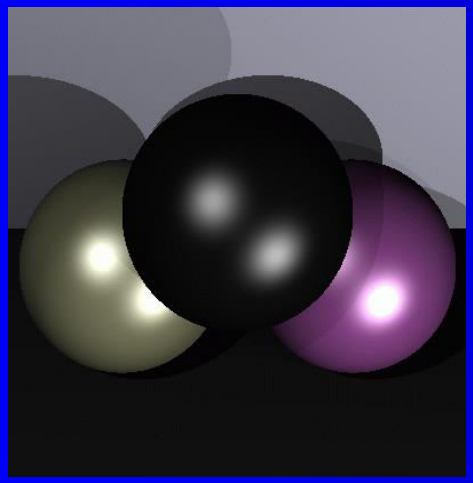
Shadow

Ray-casting képek

Lokális illumináció

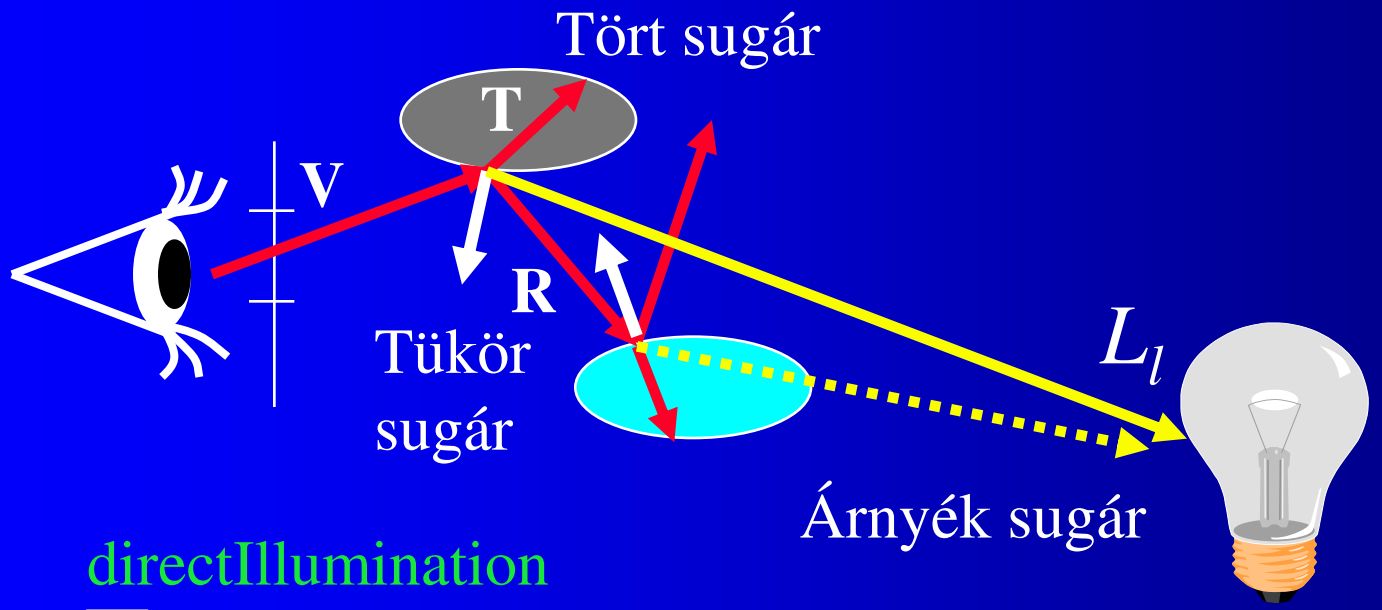


Lokális illumináció + árnyékok





Rekurzív sugárkövetés



directIllumination

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

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

Fresnel

Tükör irányból érkező fény

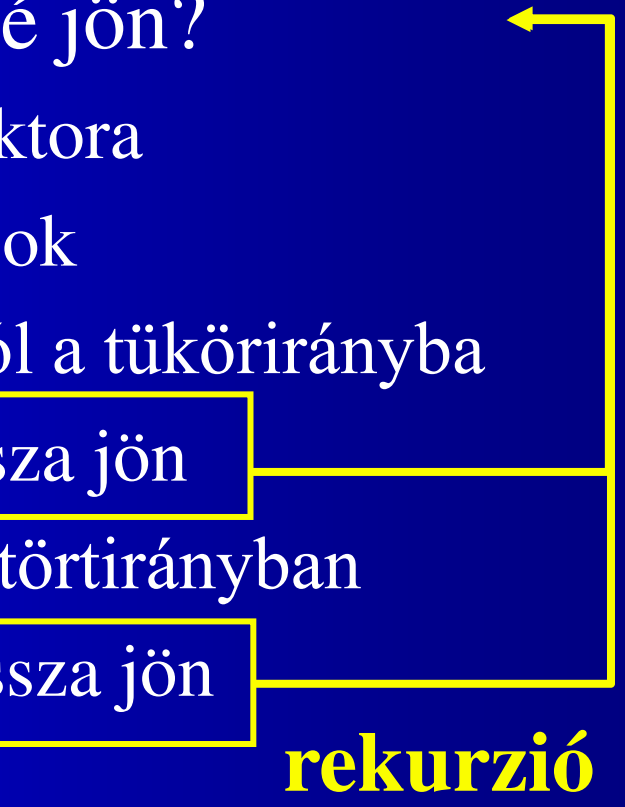
1-Fresnel

Törési irányból érkező fény

trace

Radiancia ami sugáron visszafelé jön?

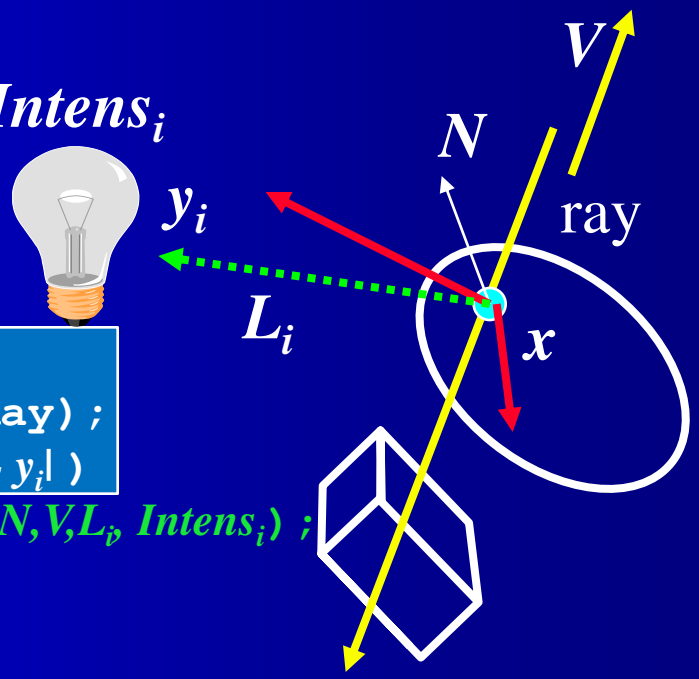
- Látható felületi pont és normál vektora
- Látható pontból látható fényforrások
- Ha tükröző, sugár a látható pontból a tükörirányba
- Radiancia ami a tükörsugáron vissza jön
- Ha törő, sugár a látható pontból a törtirányban
- Radiancia ami a törési sugáron vissza jön
- A hozzájárulások összegzése



trace function

```
float3 trace(Ray ray) {

  Hit hit = firstIntersect(ray);
  if(hit.t < 0) return La; // nothing
  float3 outRadiance (0, 0, 0);
  for(each light source i){
    Ray shadowRay(x + Nε, Li);
    Hit shadowHit = firstIntersect(shadowRay);
    if(shadowHit.t < 0 || shadowHit.t > |x - yi| )
        outRadiance += hit.material->shade(N,V,Li, Intensi) ;
  }
  if(hit.material->reflective){
    float3 reflectionDir = reflect(V,N);
    Ray reflectedRay(x + Nε, reflectionDir);
    outRadiance +=trace(reflectedRay)*F(V,N);
  }
  if(hit.material->refractive) {
    float3 refractionDir = refract(V,N);
    Ray refractedRay(x - Nε, refractionDir);
    outRadiance +=trace(refractedRay)*(float3(1,1,1)-F(V,N));
  }
  return outRadiance;
}
```

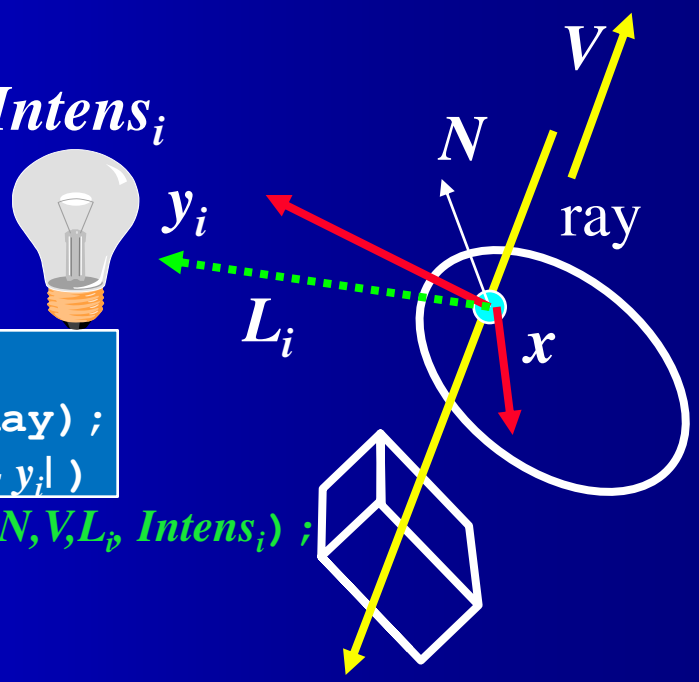
Shadow



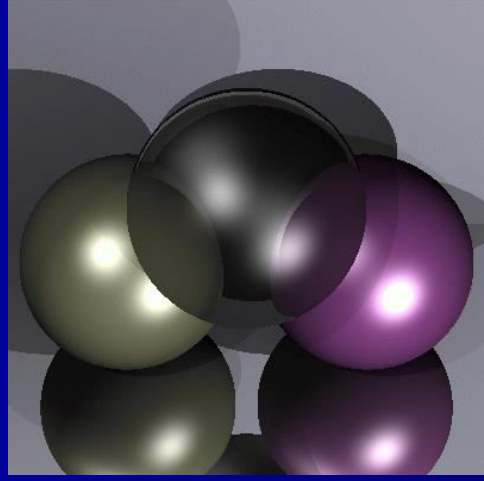
trace function

```
float3 trace(Ray ray, int depth) {
  if(depth > maxDepth) return La;
  Hit hit = firstIntersect(ray);
  if(hit.t < 0) return La; // nothing
  float3 outRadiance (0, 0, 0);
  for(each light source i){
    Ray shadowRay(x + Nε, Li);
    Hit shadowHit = firstIntersect(shadowRay);
    if(shadowHit.t < 0 || shadowHit.t > |x - yi| )
      outRadiance += hit.material->shade(N,V,Li, Intensi);
  }
  if(hit.material->reflective){
    float3 reflectionDir = reflect(V,N);
    Ray reflectedRay(x + Nε, reflectionDir);
    outRadiance +=trace(reflectedRay,depth+1)*F(V,N);
  }
  if(hit.material->refractive) {
    float3 refractionDir = refract(V,N);
    Ray refractedRay(x - Nε, refractionDir);
    outRadiance +=trace(refractedRay,depth+1)*(float3(1,1,1)-F(V,N));
  }
  return outRadiance;
}
```

Shadow

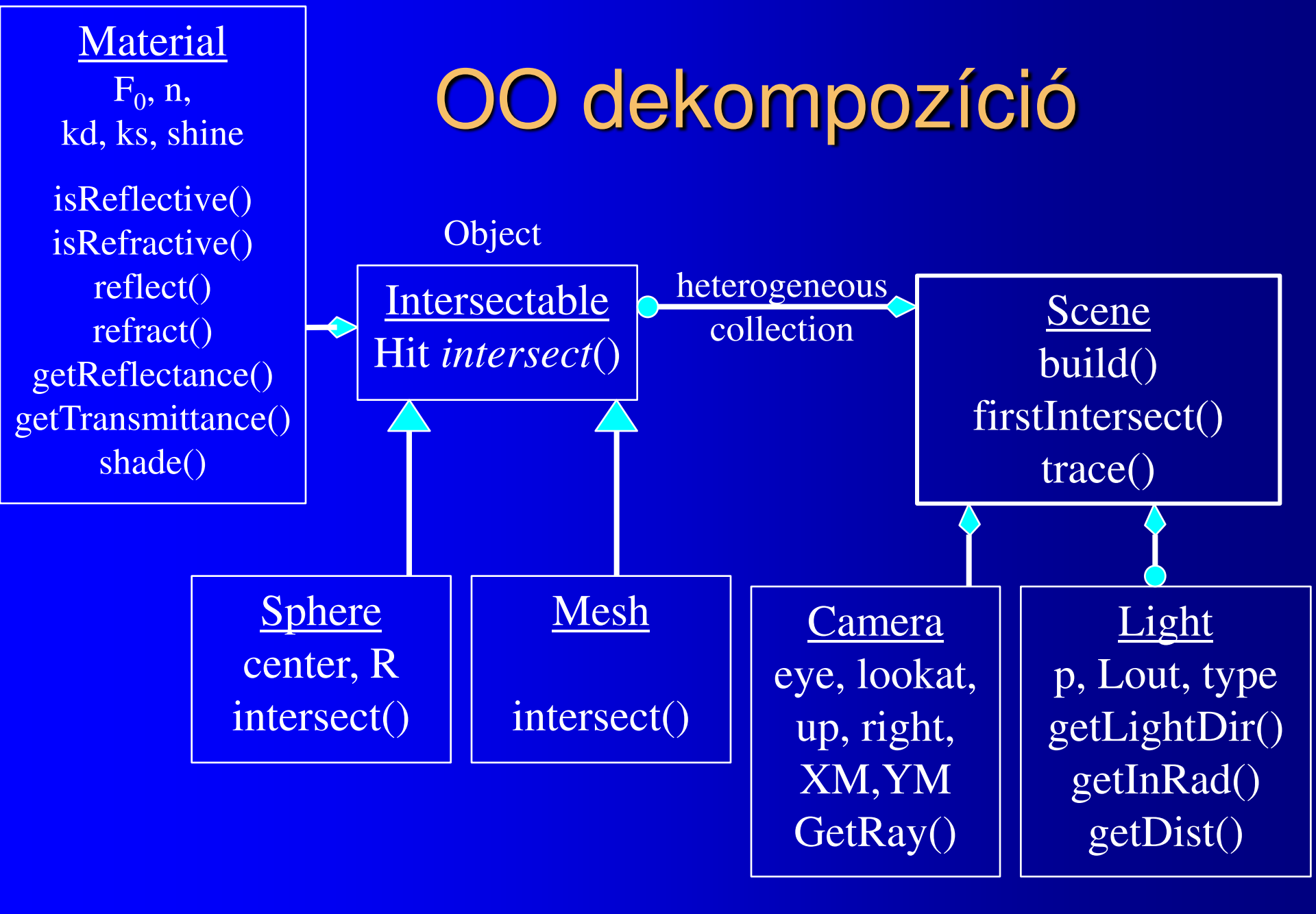


Heckbert Palika házija a névjegyén



```
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!*/
```

OO dekompozíció



Kétirányú sugárkövetés





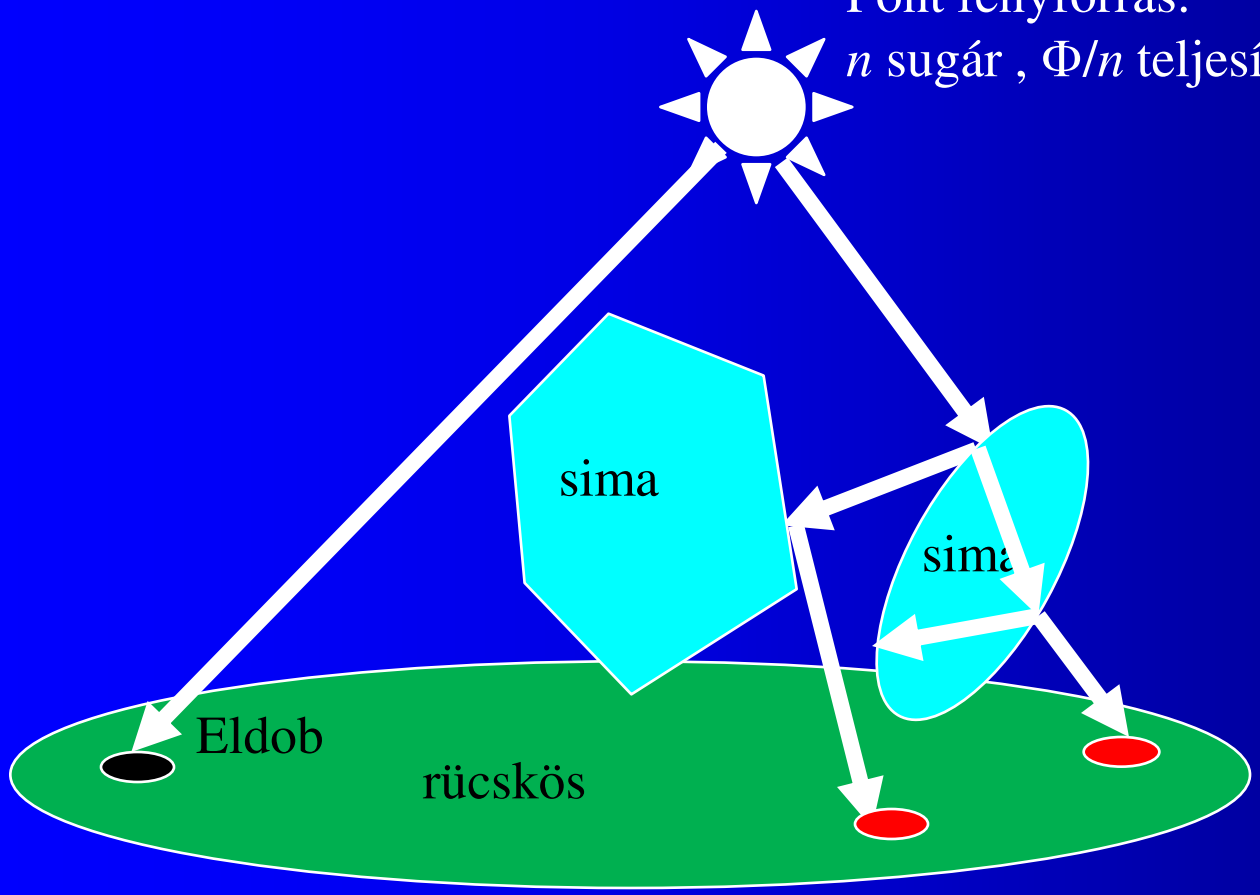
Nincsenek kausztikus foltok

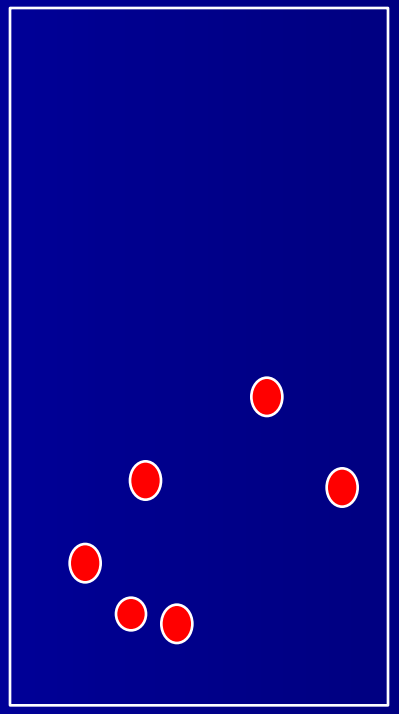




I. Foton lövés

Pont fényforrás:
n sugár , Φ/*n* teljesítmény

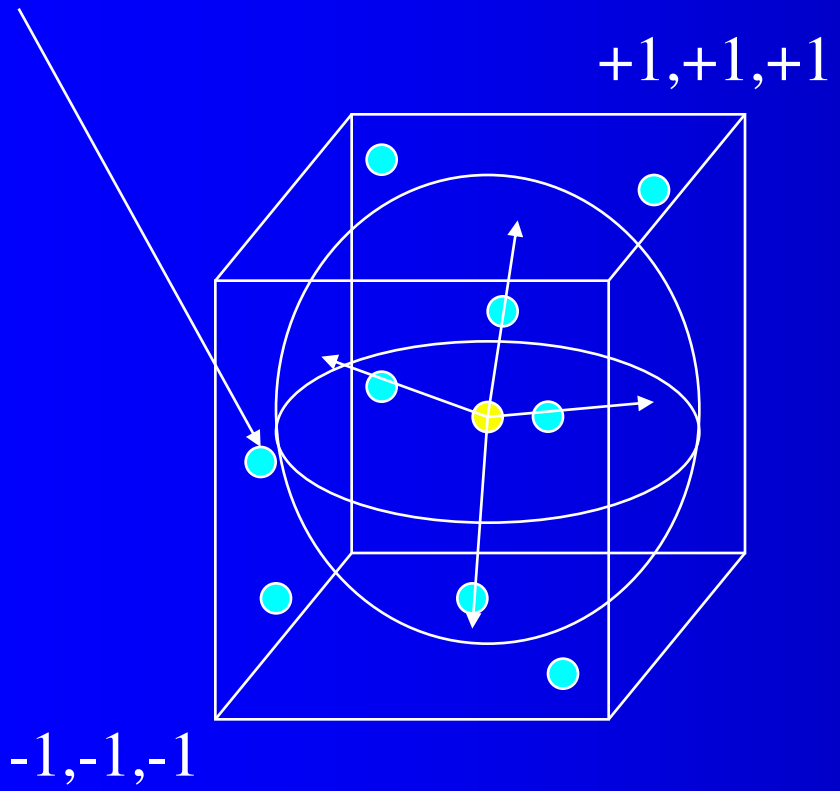




Foton térkép

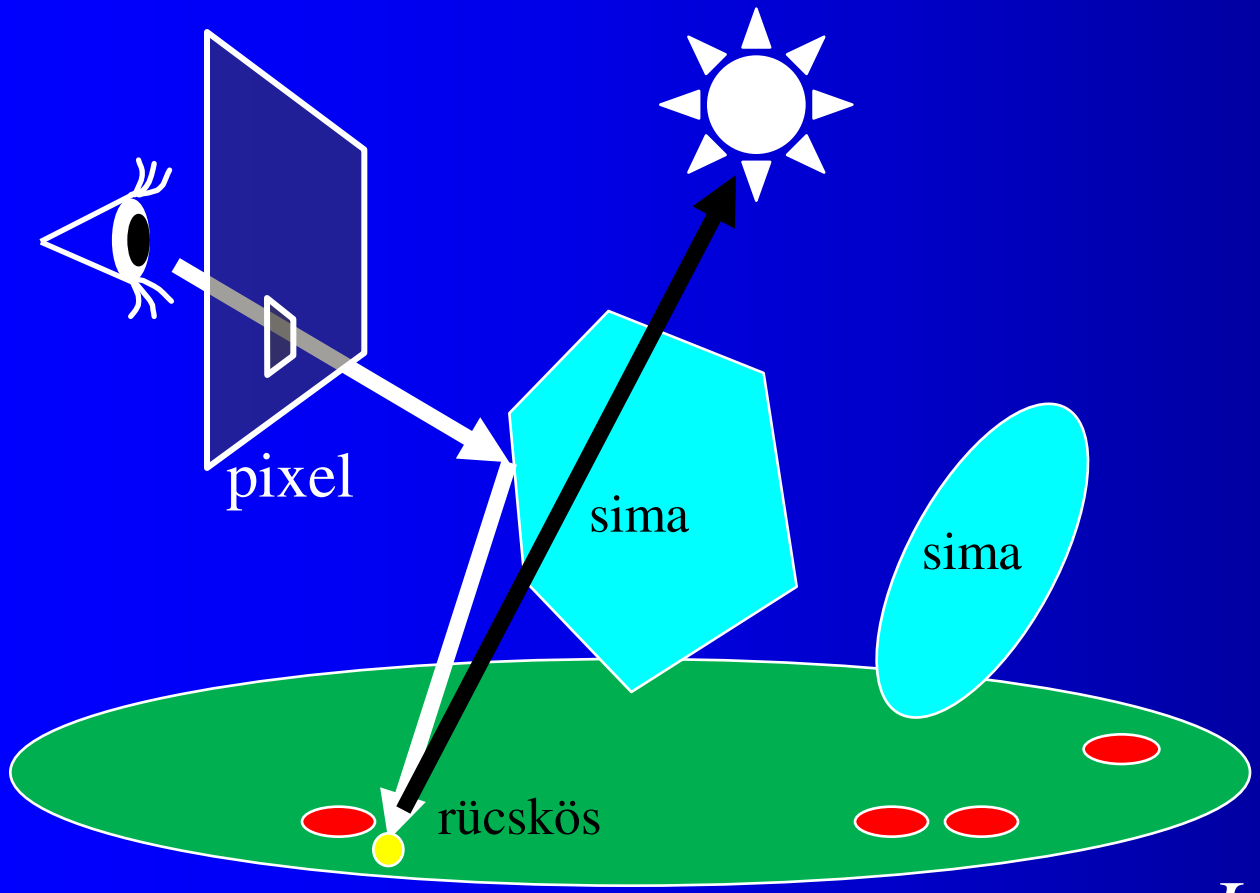
Egyenletes eloszlású irányok

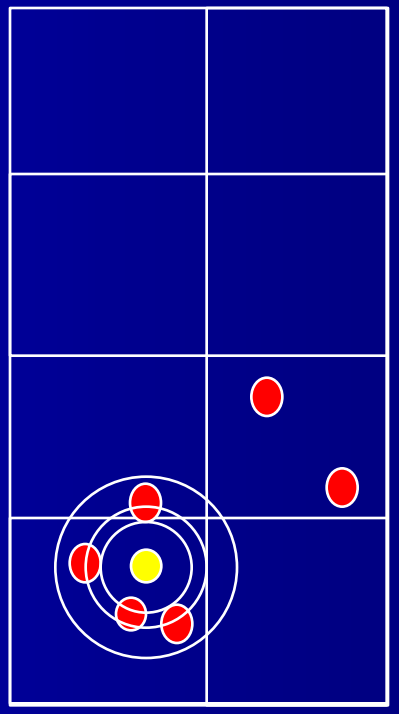
```
#define RND (2.0*(double)rand()/RAND_MAX-1.0)

(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





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

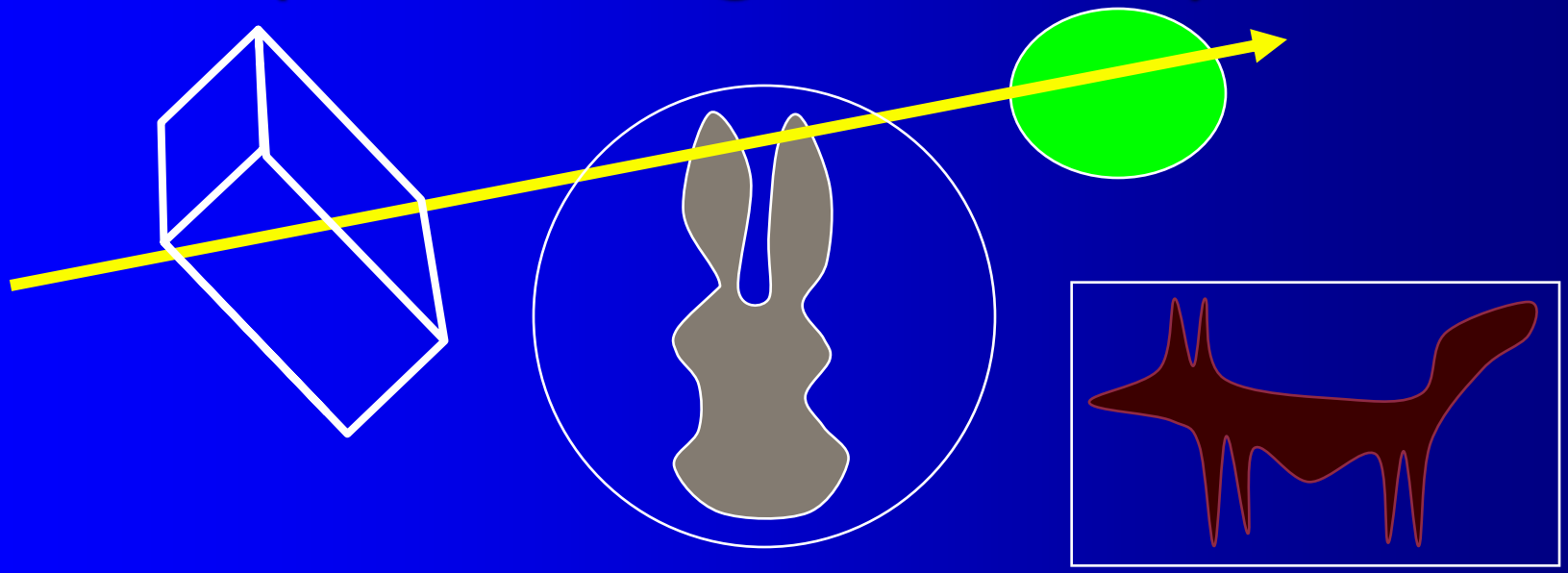
Sugárkövetés: eredmény





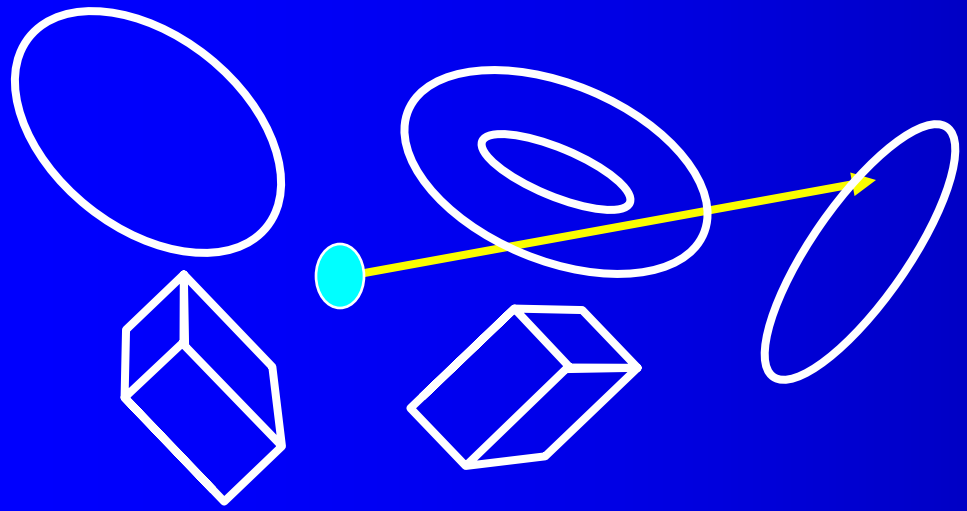
Számítási idő $\propto$ Pixelszám × Objektumszám × (Fényforrás szám+1)

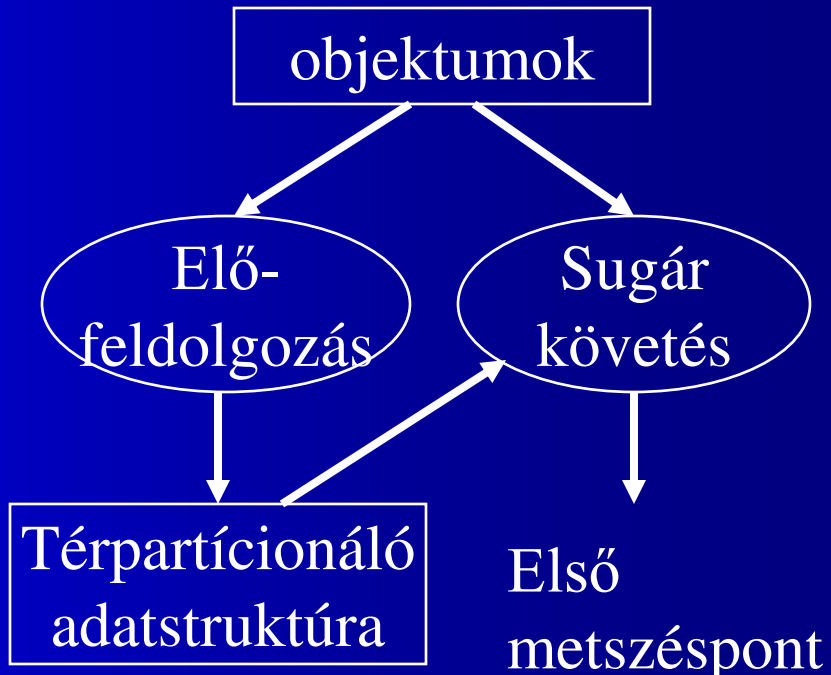
Befoglaló térfogat (Bounding Volume)



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

Térpartícionáló módszerek

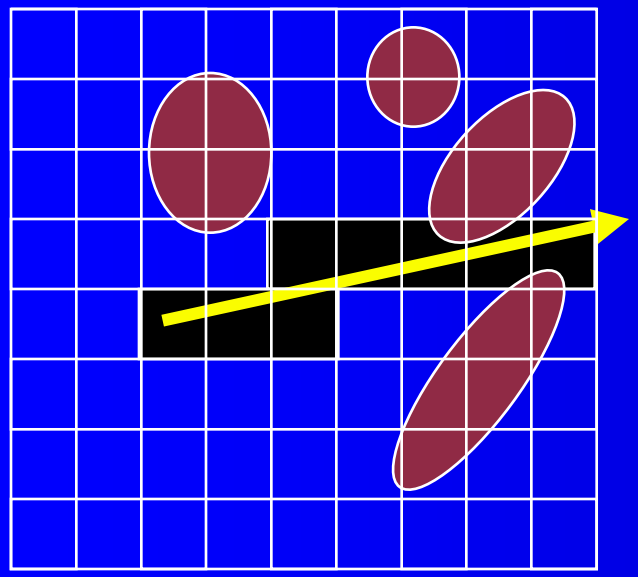


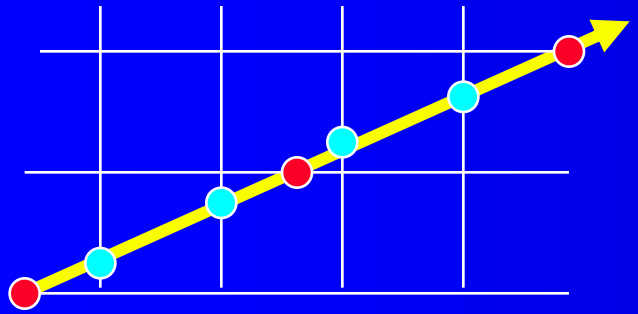


Adatstruktúra:

- 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

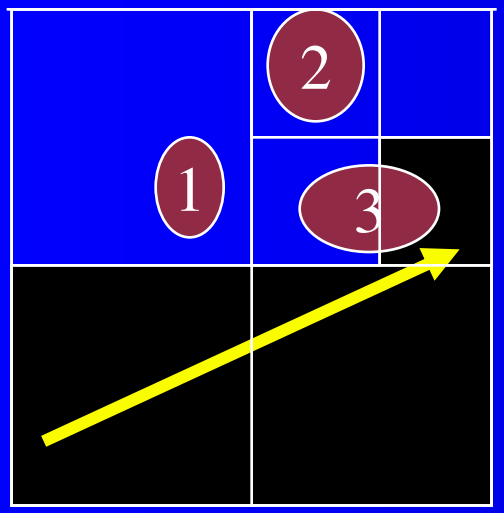
Metszés a cellában lévőkkel

IF van metszés RETURN

ENDFOR

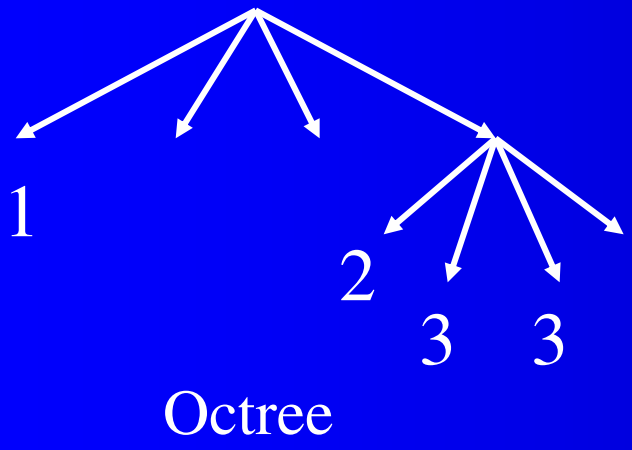
átlagos eset komplexitás: $O(1)$

Nyolcas (oktális) fa



Faépítés(cella):

IF a cellában kevés objektum van
cellában regisztráld az objektumokat
ELSE
cellafelezés: c1, c2, …, c8
Faépítés(c1); … Faépítés(c8);
ENDIF

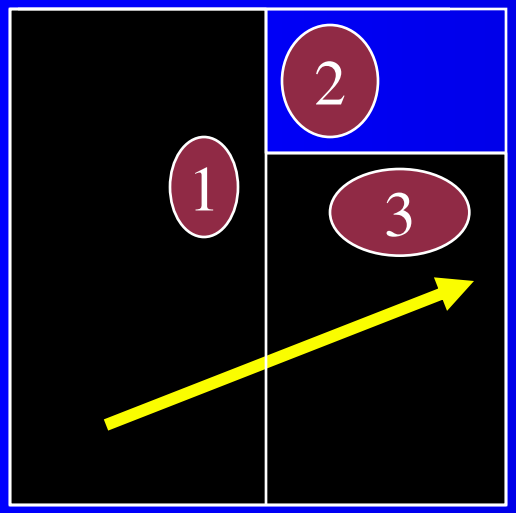


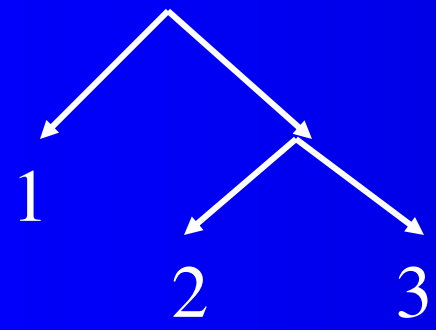
Octree

Sugárkövetés:

FOR összes sugár által metszett cellára
Metszés a cellában lévőkkel
IF van metszés RETURN
ENDFOR

Binary Space Partitioning fa (kd-fa)





kd-tree

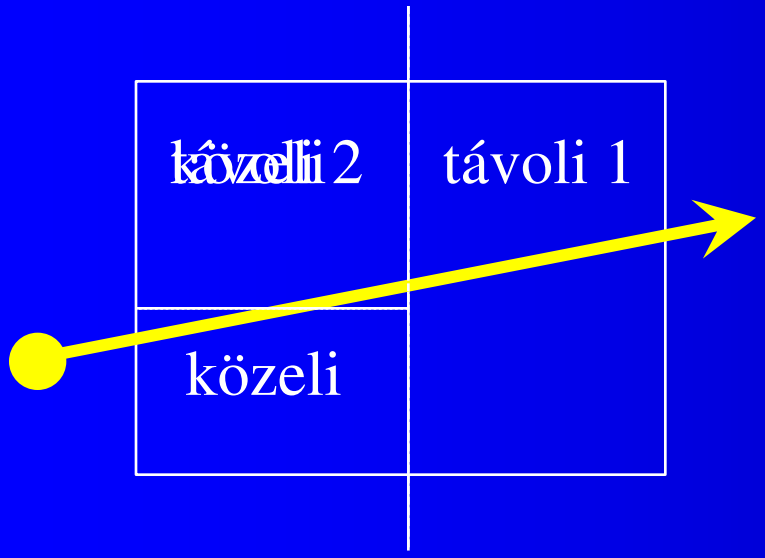
Faépítés(cella):

IF a cellában kevés objektum van
cellában regisztráld az objektumokat
ELSE
síkkeresés
cella felezés a síkkal: c1, c2
Faépítés(c1); Faépítés(c2);
ENDIF

Sugárkövetés:

FOR each cell intersecting the line
Metszés a cellában lévőkkel
IF van metszés RETURN
ENDFOR

kd-fa bejárása

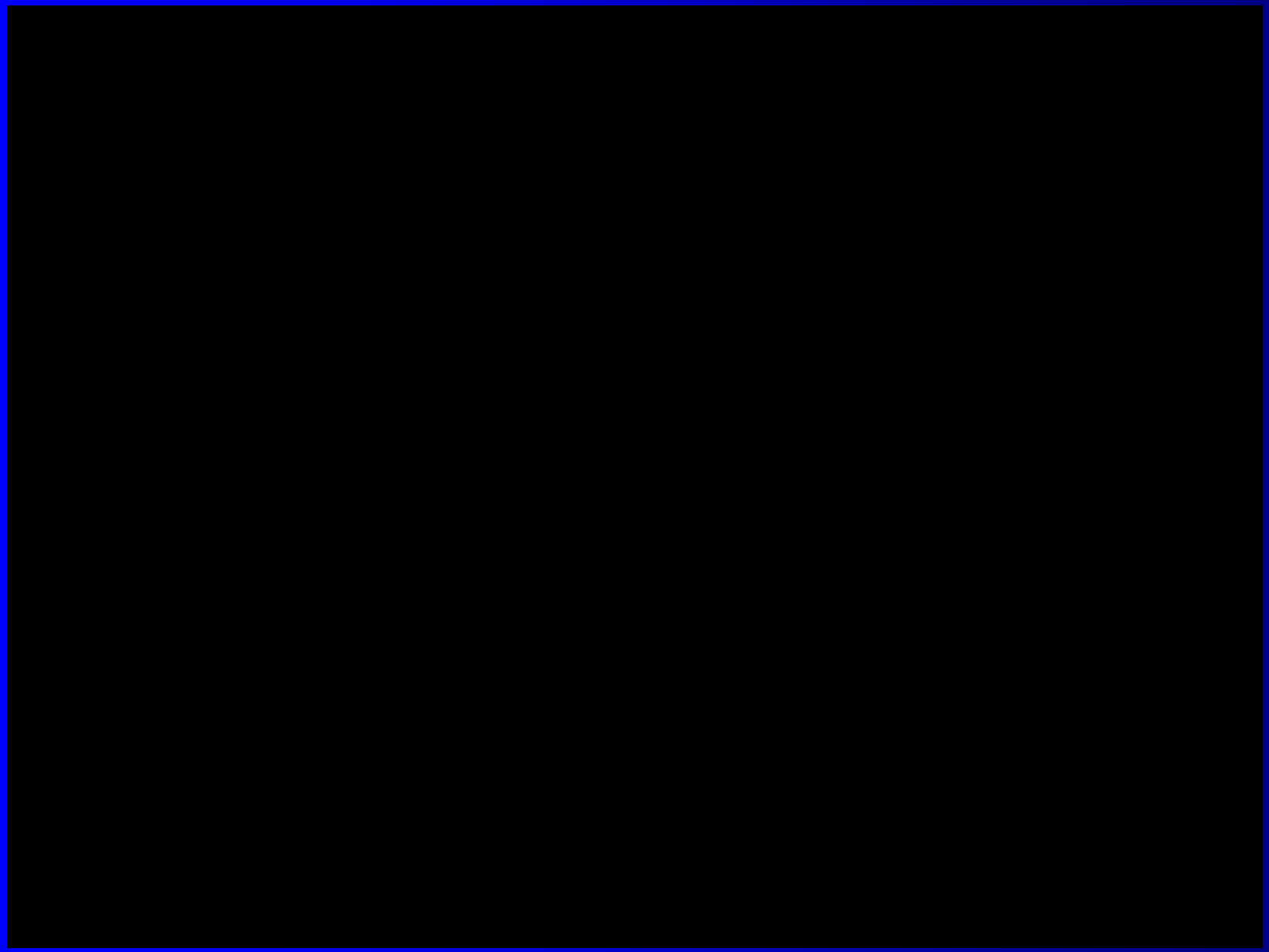



```
while (stack nem üres) {
    Pop(AABB)
    while (nem levél) {
        Ha van AABB-nek távoli
            Push(távoli)
        AABB = közeli
    }
    Regisztrált objektumok metszése
    Ha van metszéspont return
}
```

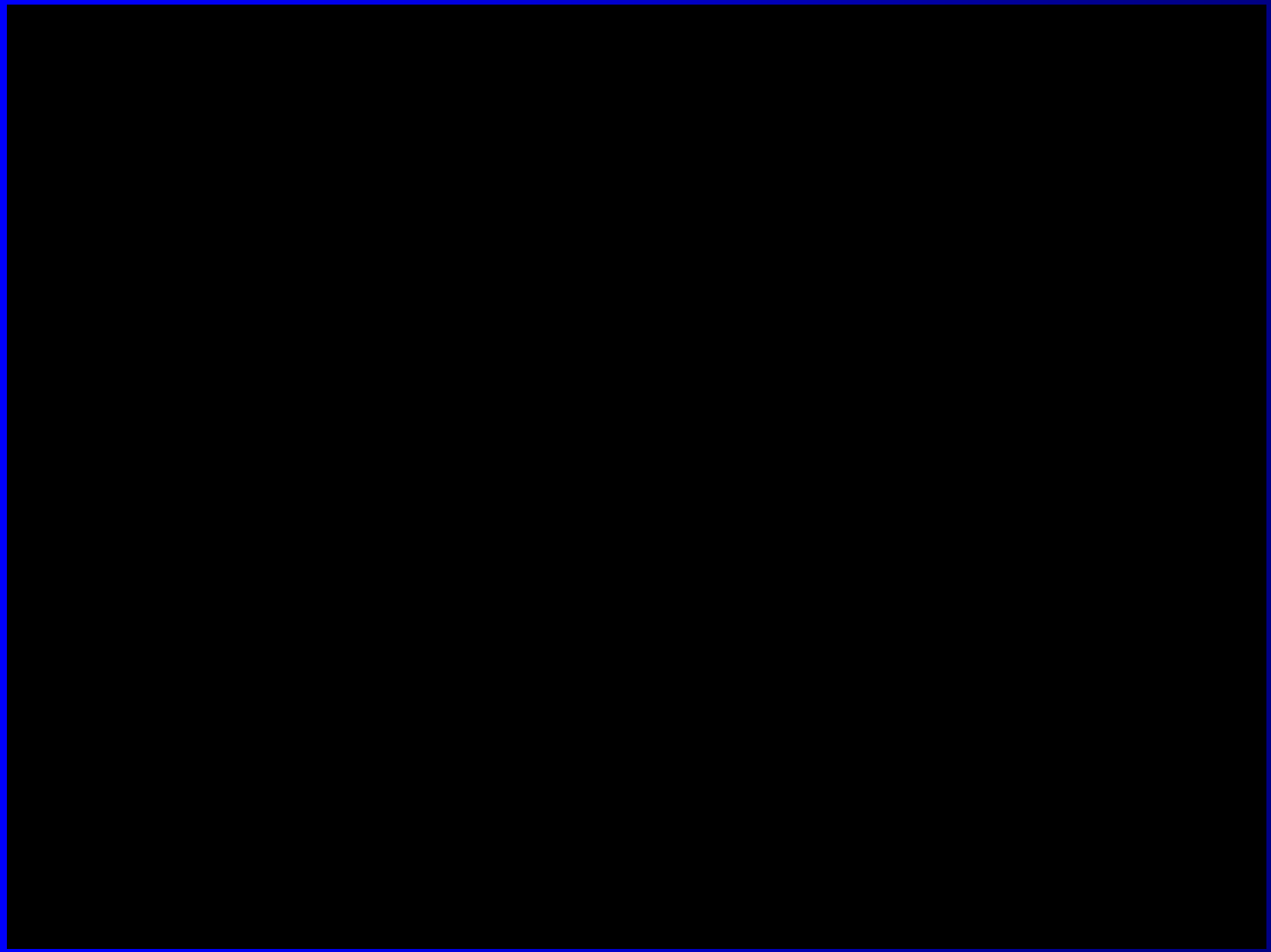
Verem tartalma:

távoli 1

Boeing



Mandelbulb



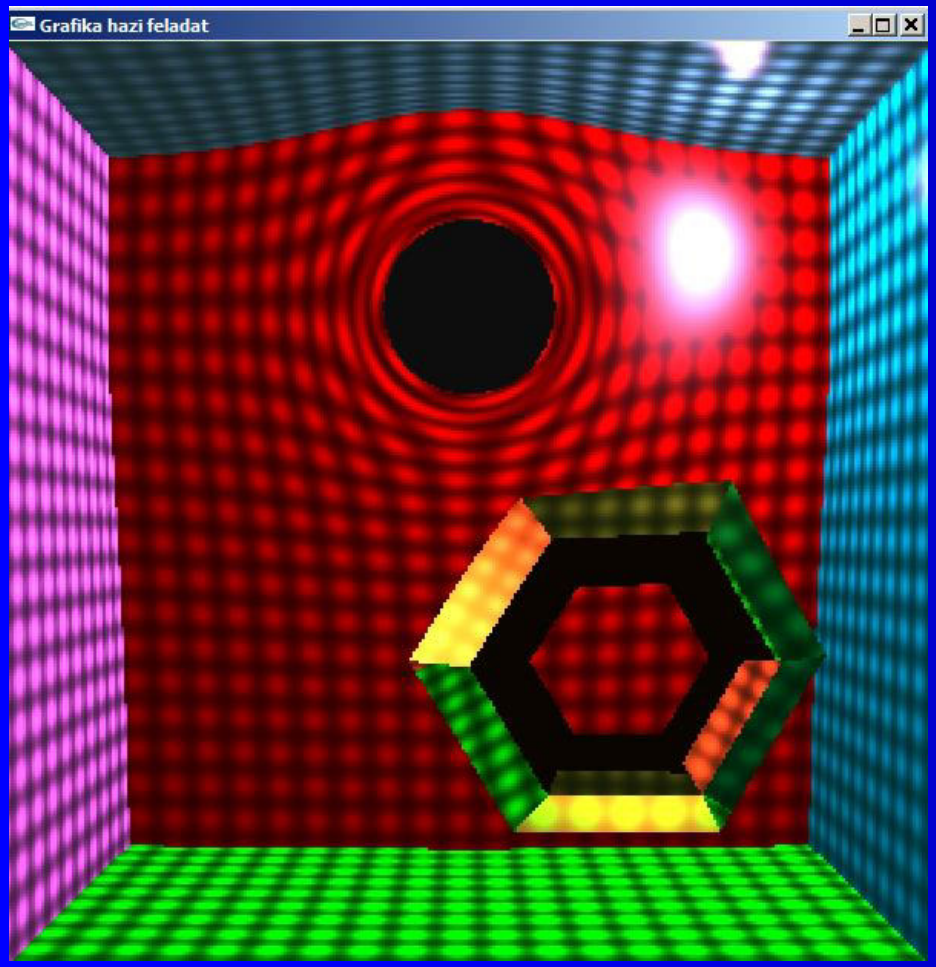
Második házi

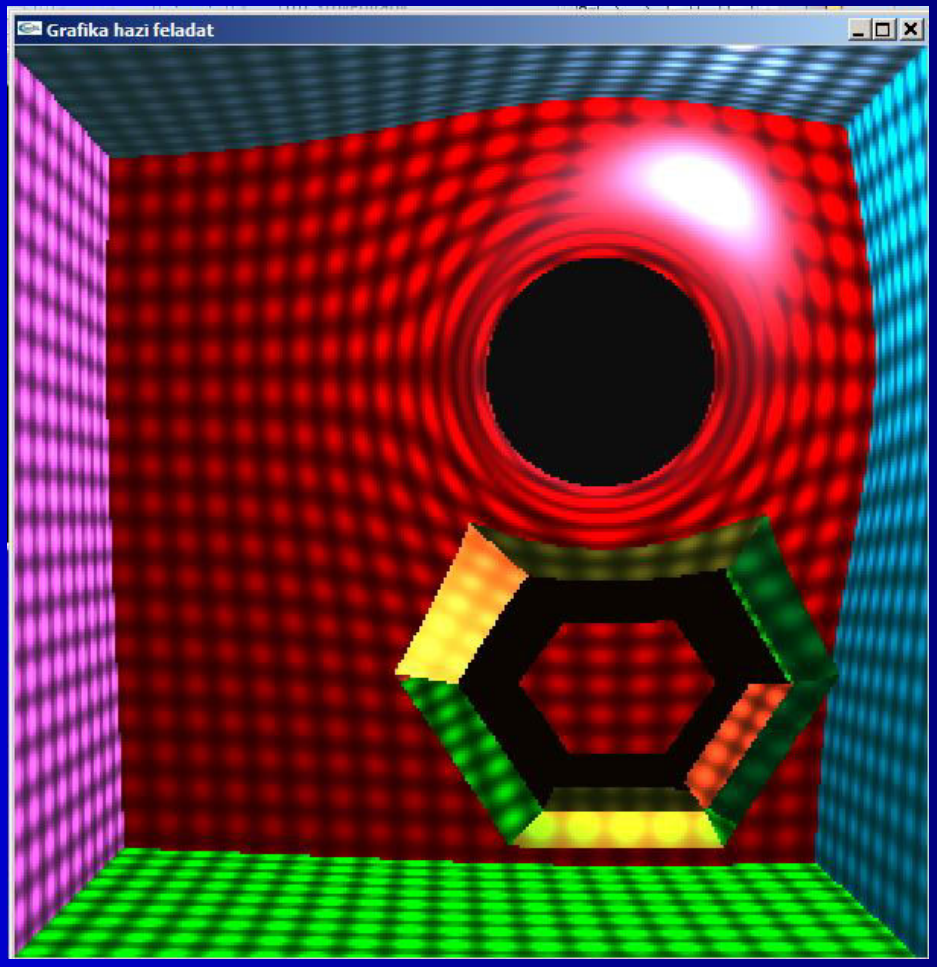
Készítsen sugárkövető programot, amely 100 mm oldalhosszúságú, egyik oldalán nyitott dobozban lévő durván tesszellált, optikailag sima, tükröző arany tóruszt, és egy Schwarzschild típusú (azaz töltés nélküli és nem forgó) Föld tömegű fekete lyukat tartalmaz. A doboz oldalai diffúz+spekuláris visszaverődésűek, a diffúz tag procedúrálisan textúrázott. A színteret ambiensfény és legalább egy pontfényforrás világítja meg. Élhet azzal az egyszerűsítéssel, hogy a fekete lyuk a fényt csak az első visszaverődés után görbíti, azaz a pont fényforrás és a megvilágított felület között nem, de a rücskös és sima felületek között, valamint a felületek és a szem között igen (a másik alternatíva a fényforrásból kilépő fény görbítésére a fotontérkép módszer). A fény elgörbülését kellő tudásszomj esetén Einstein téregyenletének Schwarzschildtól származó megoldásával, egyébként pedig az ekvivalenciaelv és a klasszikus newtoni gravitációs formula felhasználásával, sugármasírozással kell szimulálni. A rendelkezésre álló CPU idő 120 sec, szükség esetén kisebb bontású kép átmintavételezésével, távolságtól függő lépés-nagysággal és befoglaló térfogatokkal lehet a programot gyorsítani.

A szükséges fizikai állandók: A fény sebessége: 300 000 km/sec;
A föld kerülete: 40 000 km; A nehézségi gyorsulás a föld felszínén: 10 m/sec^2

Az arany törésmutatója és kioltási tényezője: r: 0.17/3.1, g: 0.35/2.7, b: 1.5/1.9
A falak diffúz és spekuláris visszaverődési tényezője és shininess paramétere, valamint a textúra szabadon megválasztható.

Aranytórusz és Schwarzschild fekete lyuk textúrázott dobozban





Inkrementális 3D képszintézis

Szirmay-Kalos László

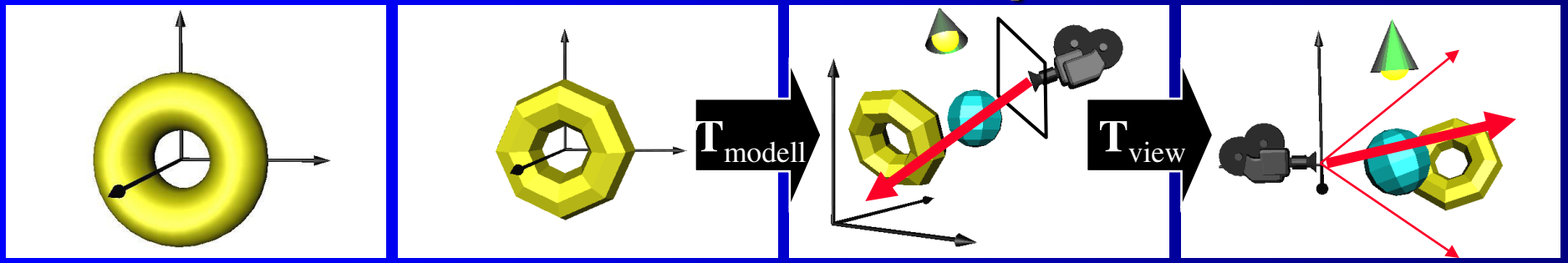
Inkrementális képszintézis

- 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 akármit: tesszelláció

3D inkrementális képszintézis

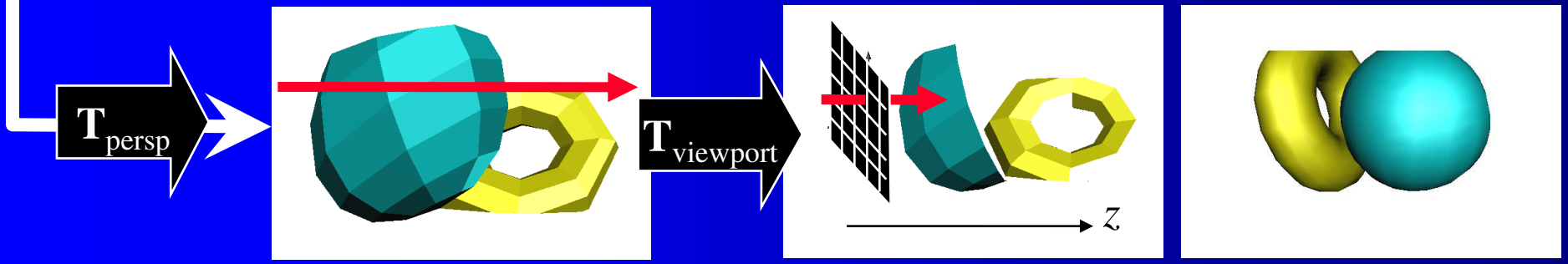


Modell
referencia helyzetben

Tesszelláció

Világkoordináta r.

Kamerakoordináta r.



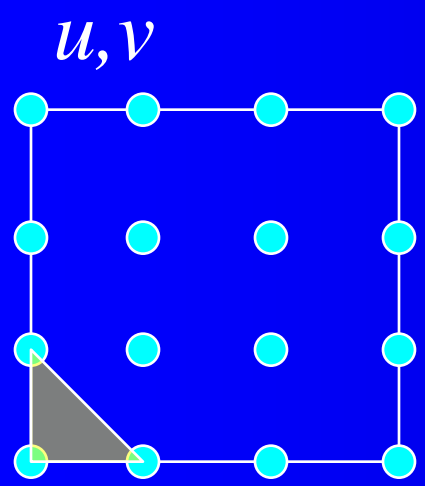
Normalizált
eszközkoordináta r.
Vágás

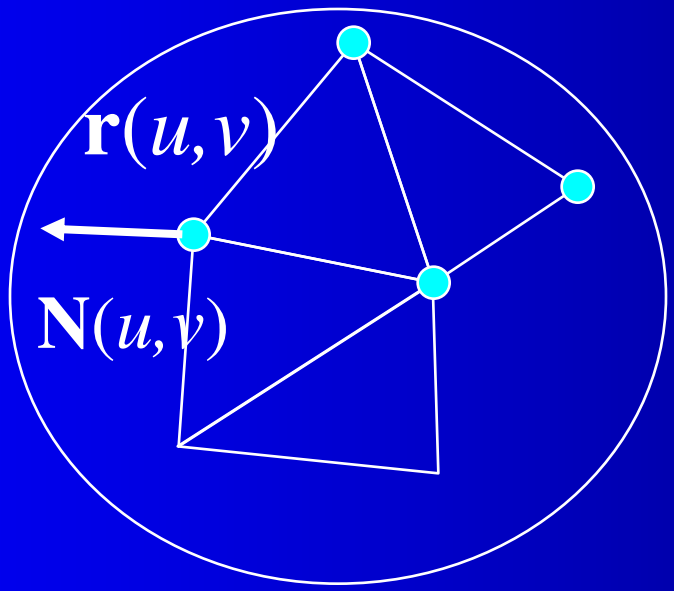
Képernyőkoordináta r.
Láthatóság + vetítés
Takarás és vetítés triviális!

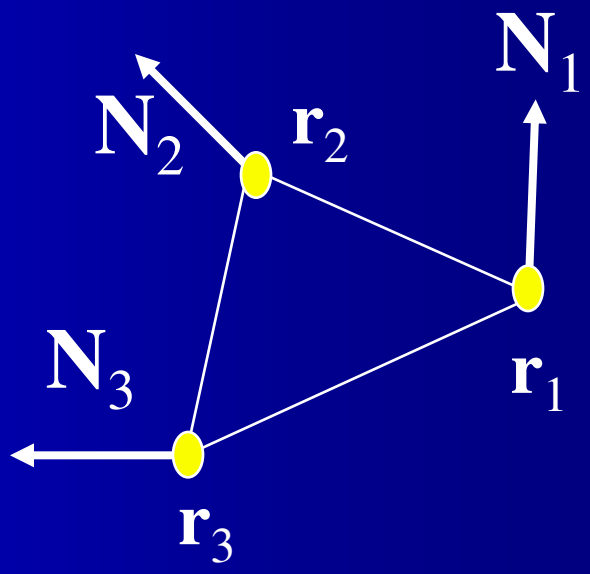
Árnyalt kép

Tesszelláció

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

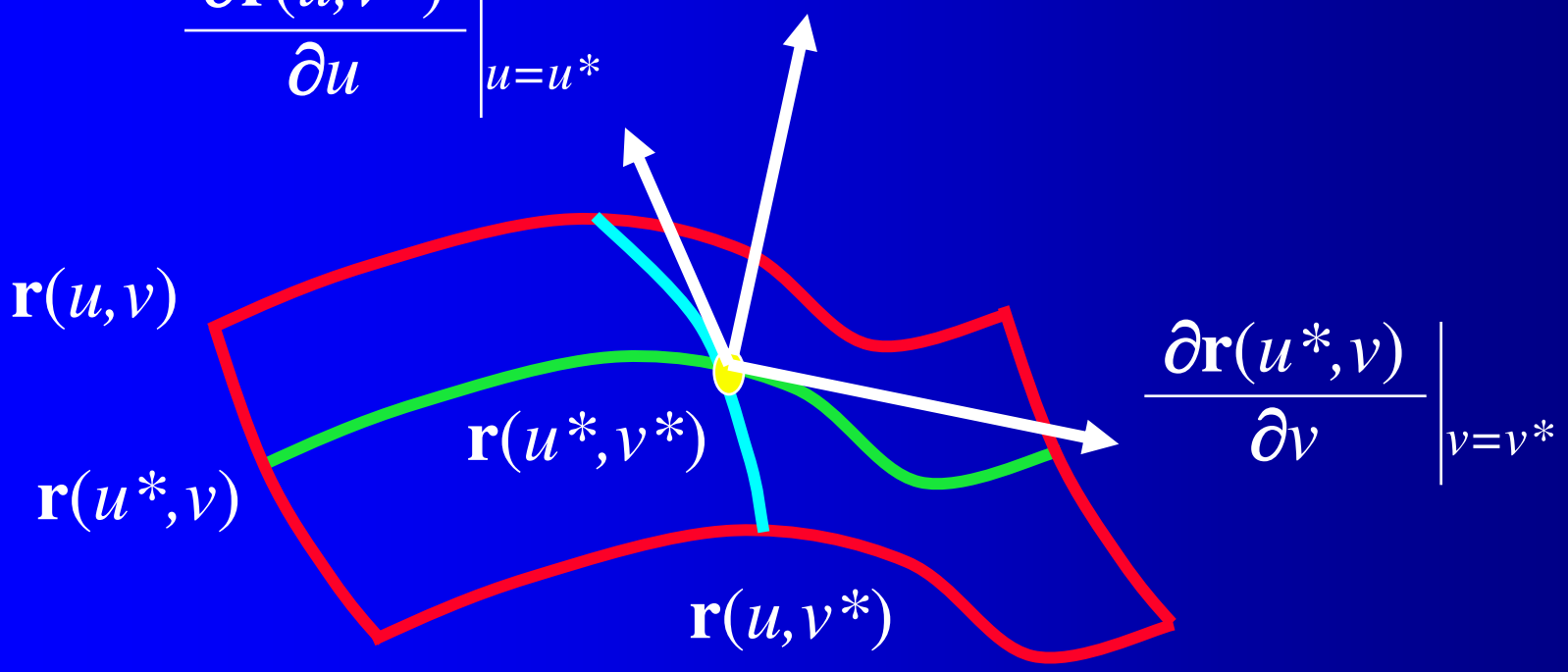




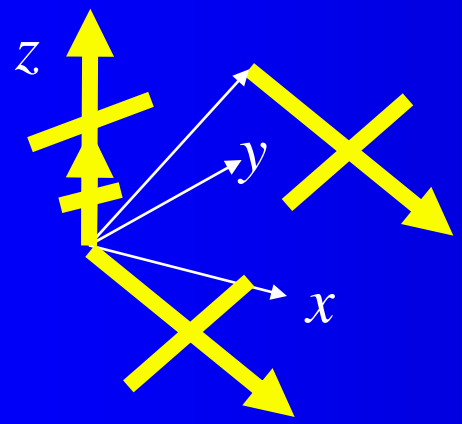


Parametrikus felületek normálvektora

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



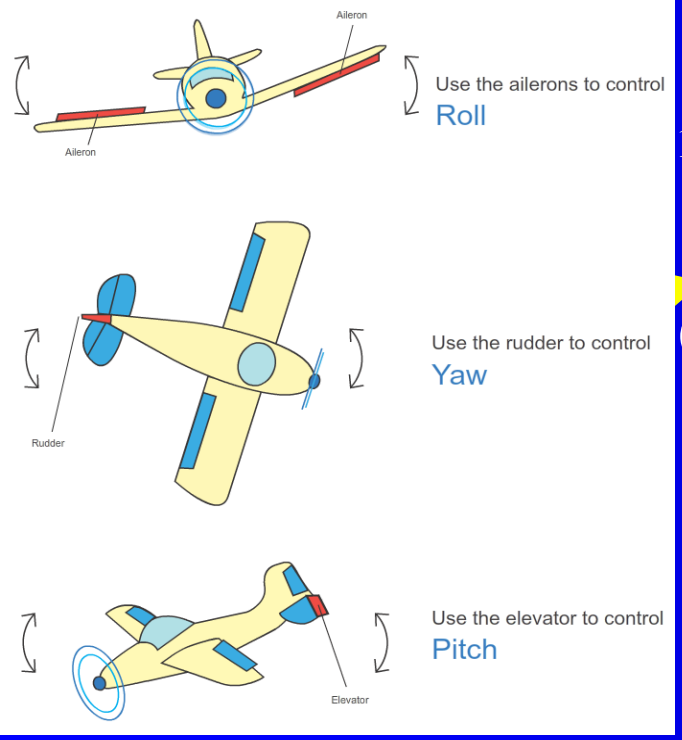
Modellezési transzformáció



1. skálázás: *sx, sy, sz*
2. orientáció: *wx, wy, wz,* α
3. pozíció: *px, py, pz*

$$\mathbf{T}_M = \begin{bmatrix} sx & & & \\ & sy & & \\ & & sz & \\ & & & 1 \end{bmatrix} \begin{bmatrix} & & & \\ & \mathbf{R} & & \\ & & & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ px & py & pz & 1 \end{bmatrix}$$

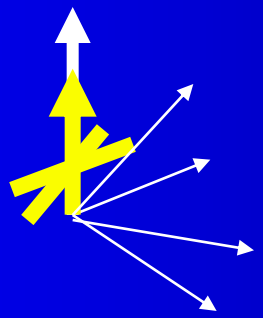
$$\boldsymbol{r}' = \boldsymbol{r}\cos(\alpha) + \boldsymbol{w}^0(\boldsymbol{r}\cdot\boldsymbol{w}^0)(1-\cos(\alpha)) + \boldsymbol{w}^0\times\boldsymbol{r}\sin(\alpha)$$

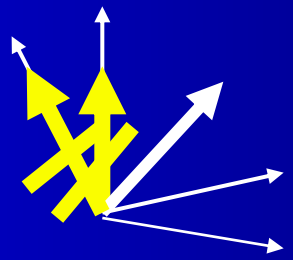


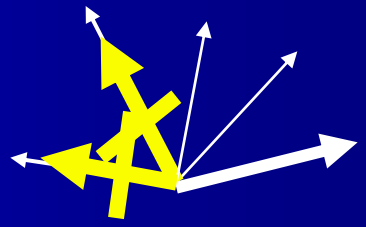
$_1,R_{22},R_{23})$

(R_{11},R_{12},R_{13})

Orientáció







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

Csavaró: Roll

Billentő: Pitch

Forduló: Yaw

Transzformációk

- Modellezési transzformáció: ✓

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

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

- Kamera transzformáció:

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

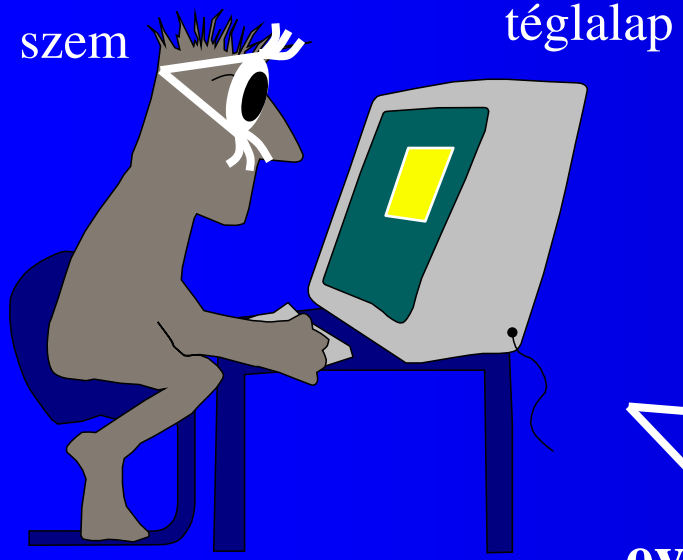
$$[\mathbf{N}_{\text{világ}},0]\,(\mathbf{T}^{-1}_{\text{view}})^{T} = [\mathbf{N}_{\text{kamera}},0]$$

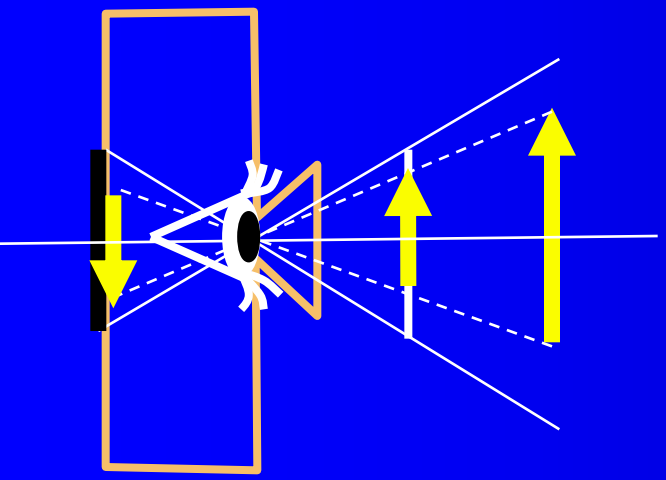
- Perspektív transzformáció:

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

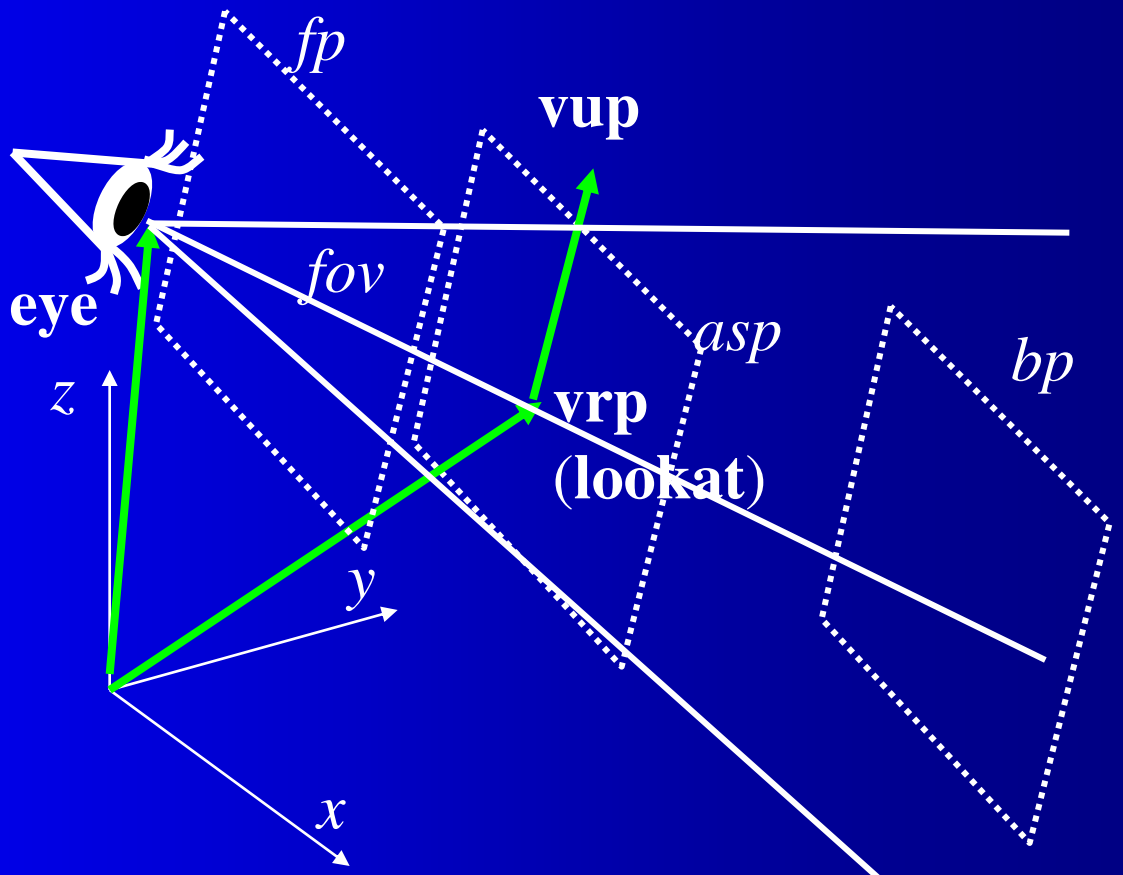
- Összetett transzformáció: $\mathbf{T}_{\text{modell}}\mathbf{T}_{\text{view}}\mathbf{T}_{\text{persp}} = \mathbf{T}_{\text{C}}$

Kamera modell

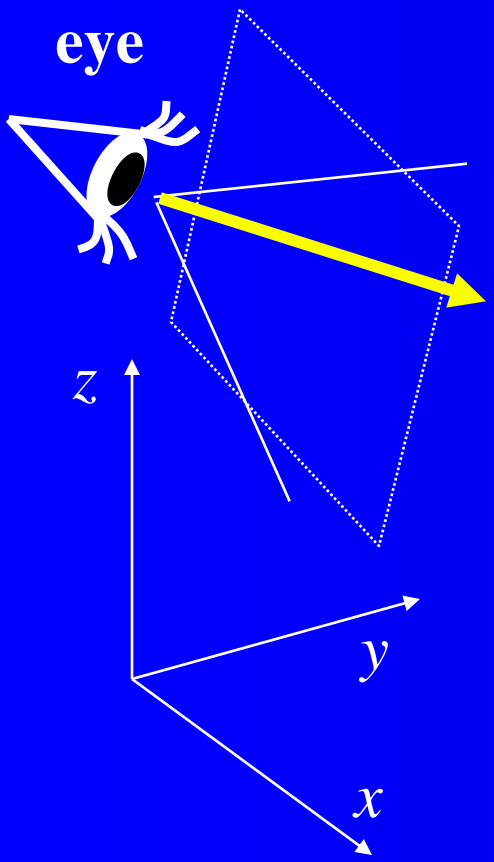




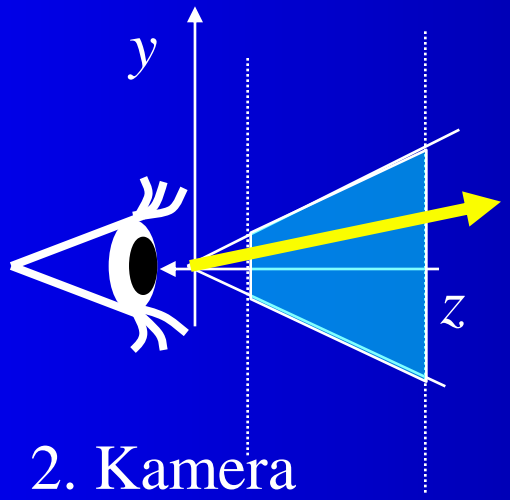
Mi: Camera obscura



Világból a képernyőre

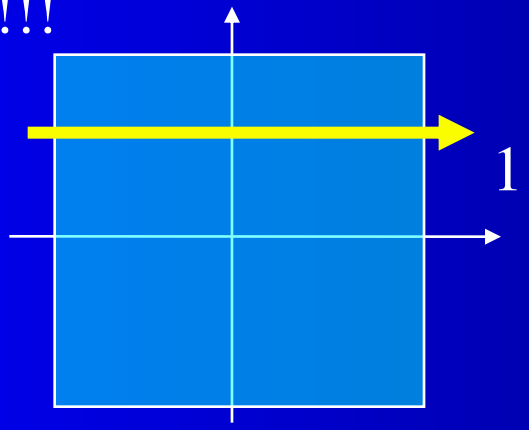


1. Világ

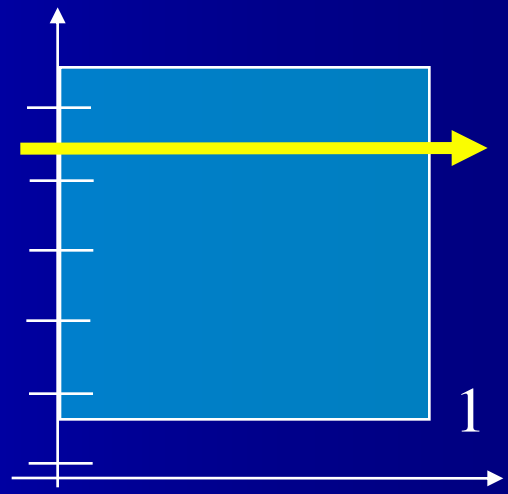


2. Kamera

Bal!!!

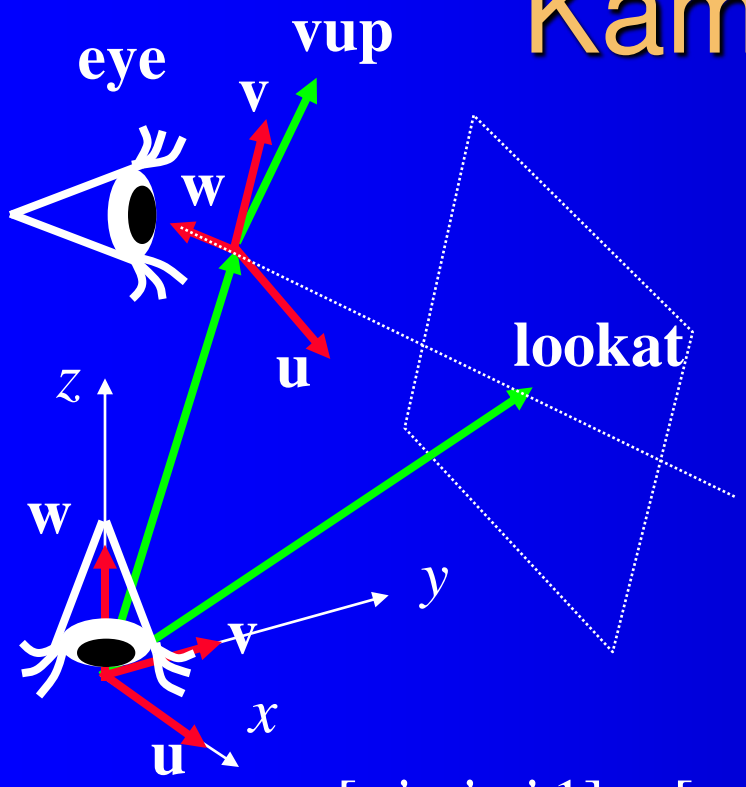


3. Normalizált képernyő



4. Képernyő

Kamera transzformáció



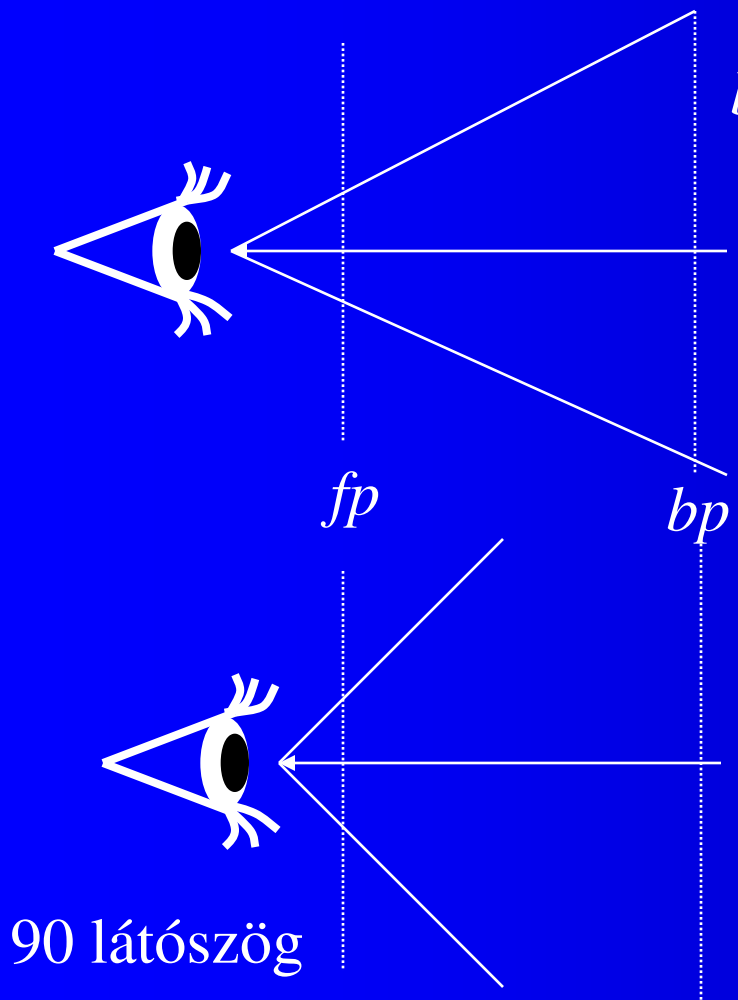
$$\mathbf{w} = (\mathbf{eye}-\mathbf{lookat})/|\mathbf{eye}-\mathbf{lookat}|$$
$$\mathbf{u} = (\mathbf{vup} \times \mathbf{w})/|\mathbf{w} \times \mathbf{vup}|$$
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

$$[x',y',z',1] = [x,y,z,1] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\mathrm{eye}_x & -\mathrm{eye}_y & -\mathrm{eye}_z & 1 \end{pmatrix} \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}$$

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

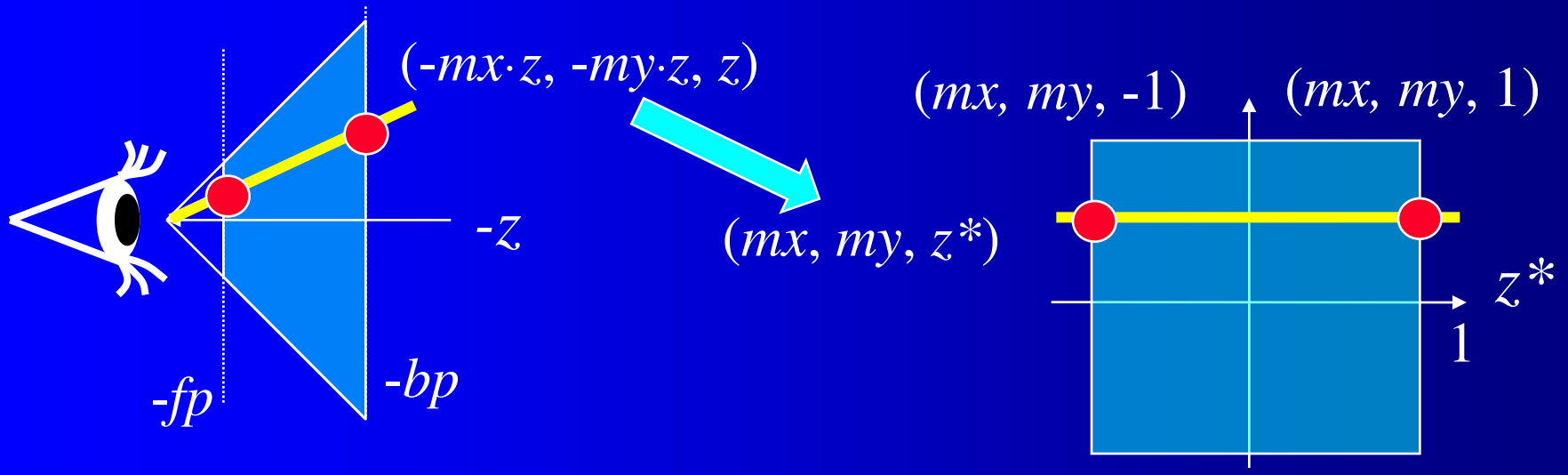
gluLookAt(eye,lookat,vup)

Normalizálás



$$\mathbf{T}_{\text{norm}} = \begin{pmatrix} 1/(\text{tg}(fov/2)\cdot asp) & 0 & 0 & 0 \\ 0 & 1/\text{tg}(fov/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

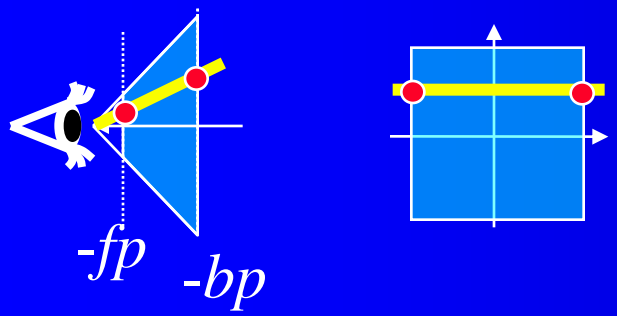
Normalizálás utáni perspektív transzformáció



$(-mx \cdot z, -my \cdot z, z) \Rightarrow (mx, my, z^*)$

$[-mx \cdot z, -my \cdot z, z, 1] \Rightarrow [mx, my, z^*, 1] \sim [-mx \cdot z, -my \cdot z, -z \cdot z^*, -z]$

Perspektív transzformáció kitalálás



$\mathbf{T}_{persp}$

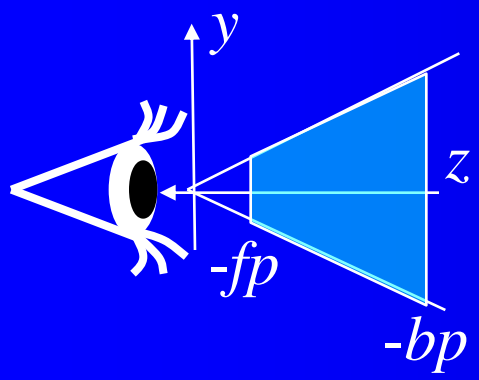
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & -1 \\ 0 & 0 & \beta & 0 \end{pmatrix}$$

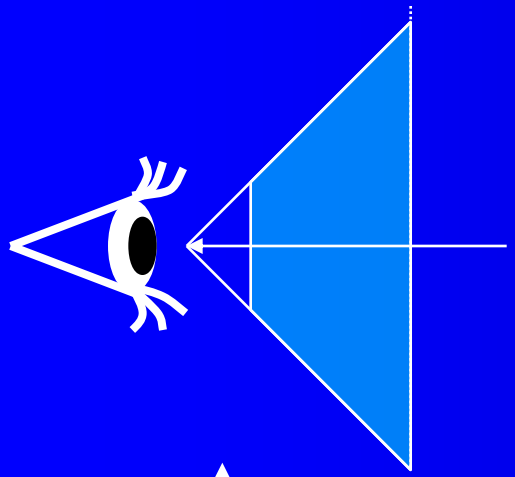
$$[-mx\cdot z,\ -my\cdot z,\ z,\ 1] \Rightarrow [-mx\cdot z,\ -my\cdot z,\ -z\cdot z^*,\ -z]$$

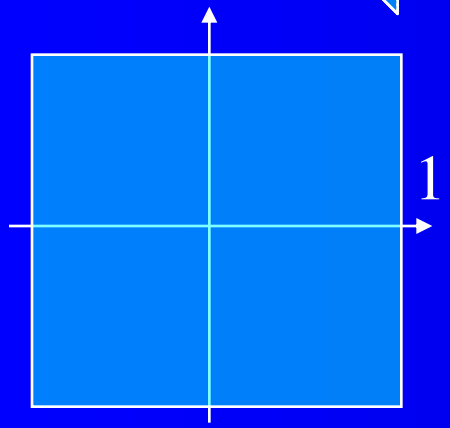
$$-z\cdot z^* = \alpha\cdot z + \beta \quad \Rightarrow \quad \boxed{z^* = -\alpha - \beta/z}$$

$$fp\cdot(-1) = \alpha(-fp) + \beta$$
$$bp\cdot(1) = \alpha(-bp) + \beta$$

$$\boxed{\begin{aligned} \alpha &= -(fp+bp)/(bp-fp) \\ \beta &= -2fp\cdot bp/(bp-fp) \end{aligned}}$$







Teljes perspektív transzformáció

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

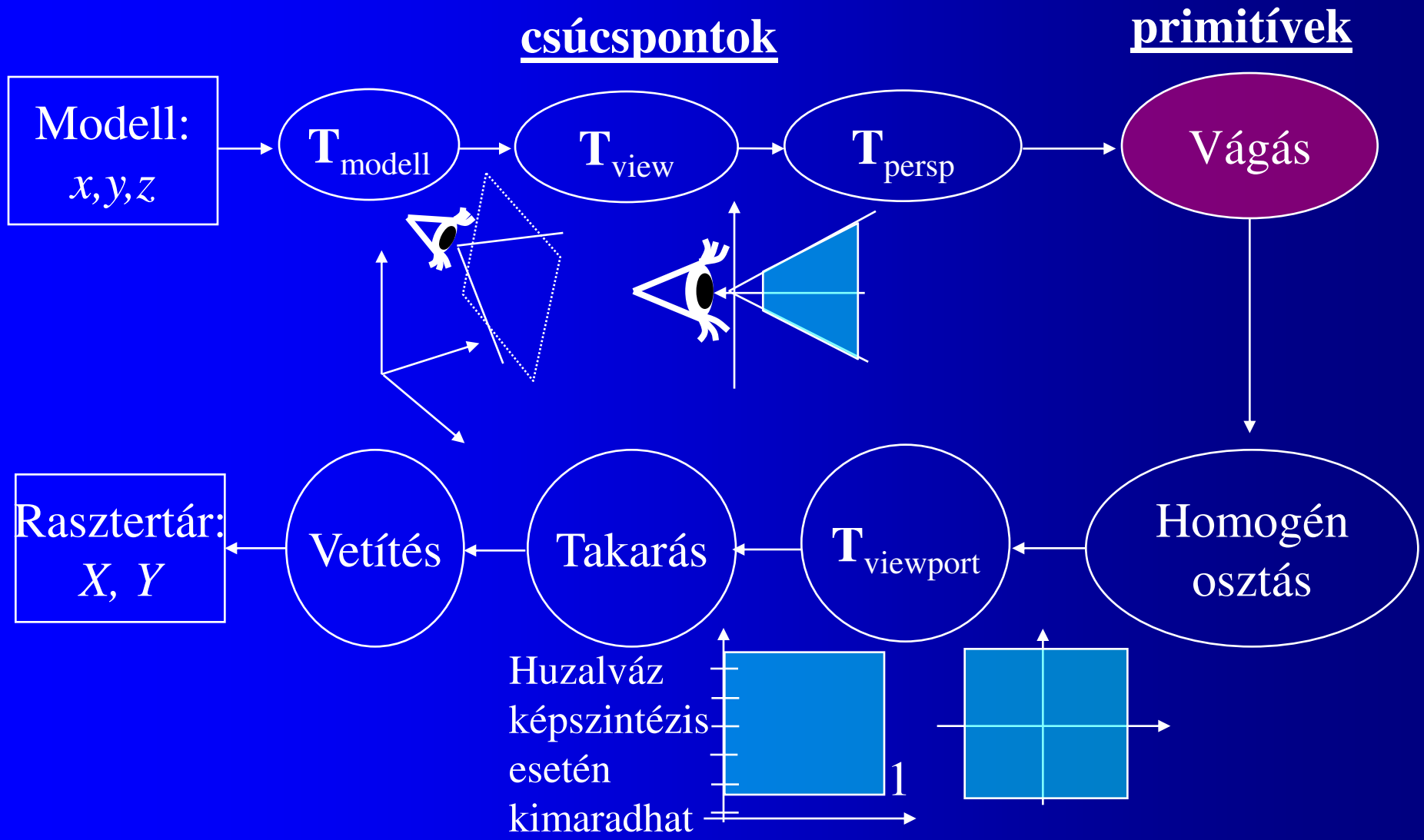
gluPerspective(fov,asp,fp,bp)

$$[X_h, Y_h, Z_h, h] = [xc, yc, zc, 1]\ \mathbf{T}_{\text{persp}}$$

$$h = -zc$$

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

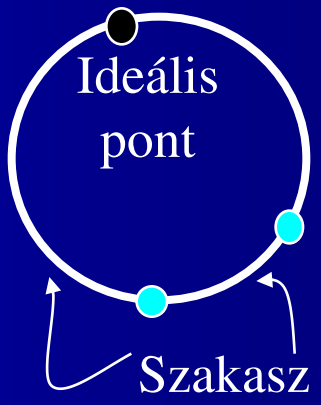
Képszintézis csővezeték

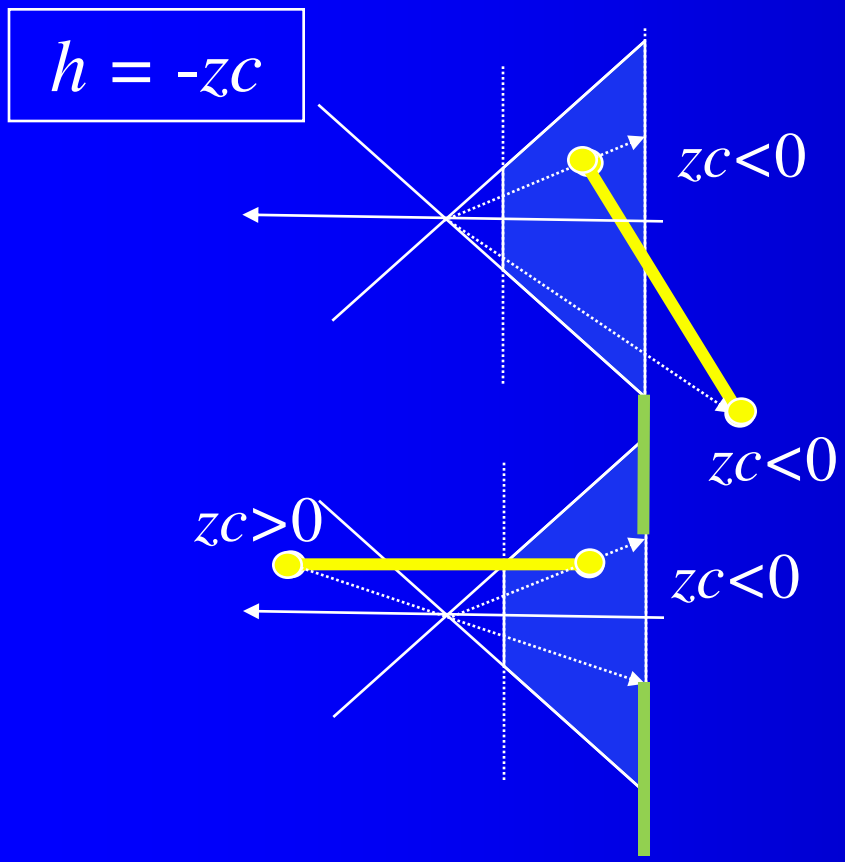


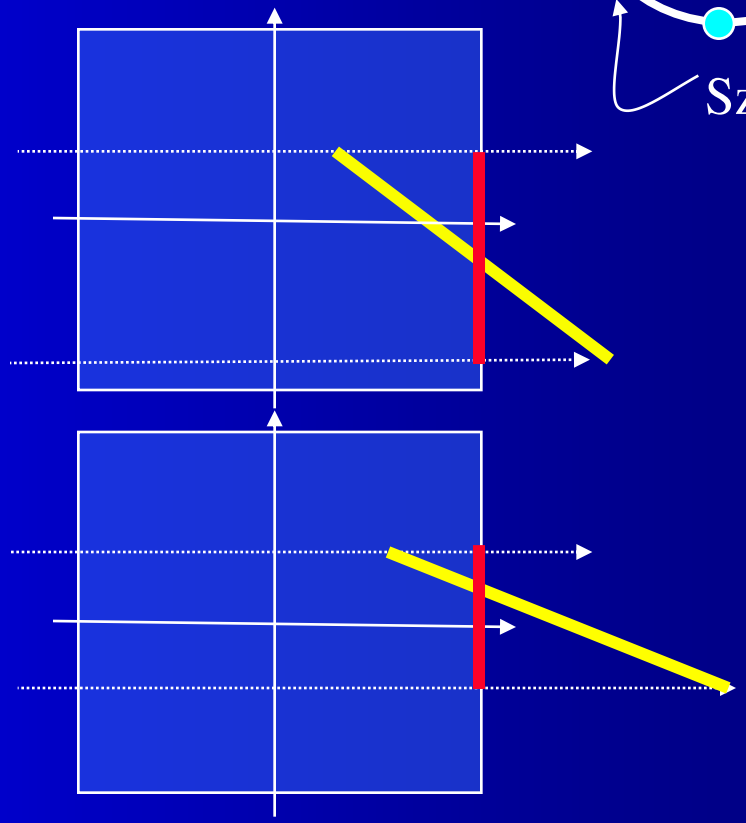
Vágni a homogén osztás előtt kell

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

Homogén lineáris transzformáció konvex kombinációkat konvex kombinációkba visz át.







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 $h = -zc$)

$-h < X_h < h$

$-h < Y_h < h$

$-h < Z_h < h$

$h = X_h$

Kívül

[3, 0, 0, 2]

$h = 2 < X_h = 3$

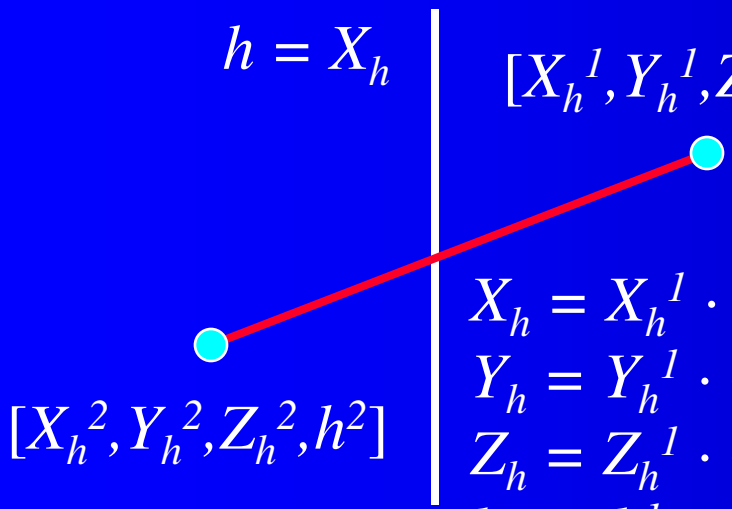
Belül

[2, 0, 0, 3]

$h = 3 > X_h = 2$

Szakasz/poligon vágás

$-h < X_h < h$
$-h < Y_h < h$
$-h < Z_h < h$

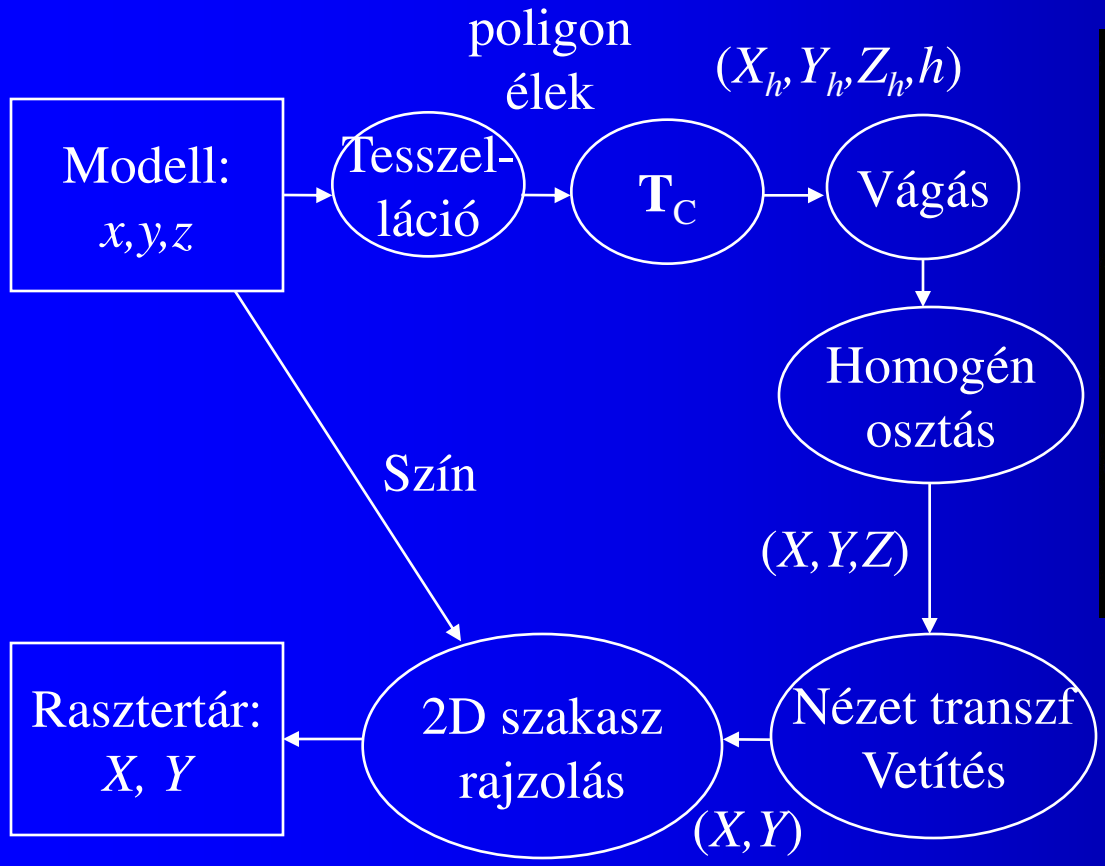


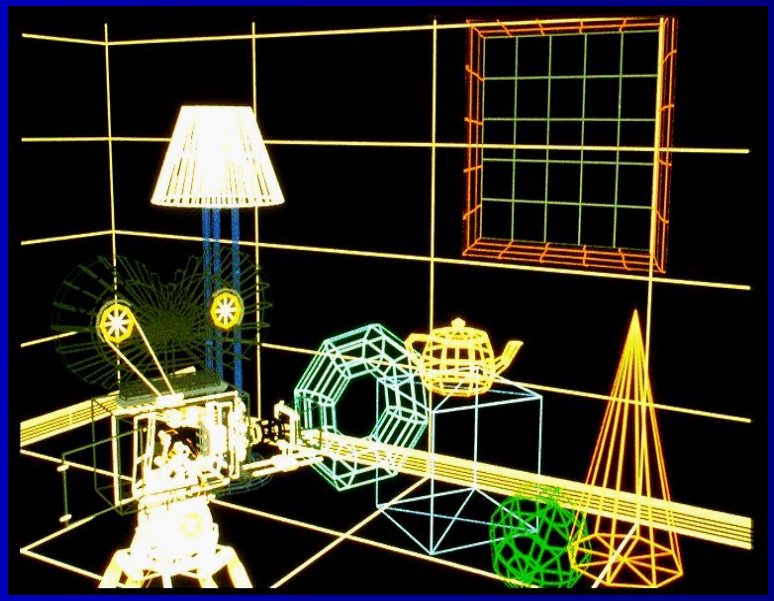
$$X_h = X_h^1 \cdot (1\text{-}t) + X_h^2 \cdot t$$
$$Y_h = Y_h^1 \cdot (1\text{-}t) + Y_h^2 \cdot t$$
$$Z_h = Z_h^1 \cdot (1\text{-}t) + Z_h^2 \cdot t$$
$$h = h^1 \cdot (1\text{-}t) + h^2 \cdot t$$

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

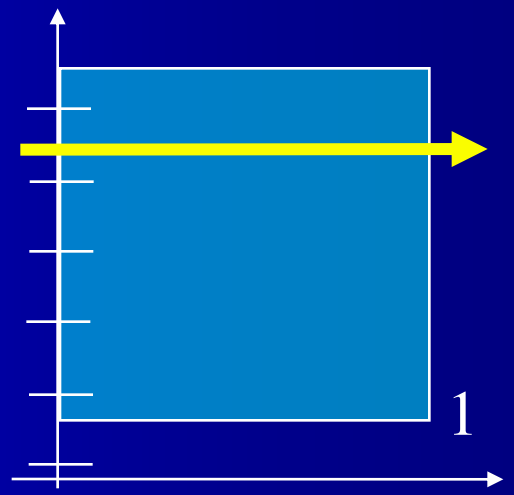
$t = \ldots$

Huzalváz képszintézis



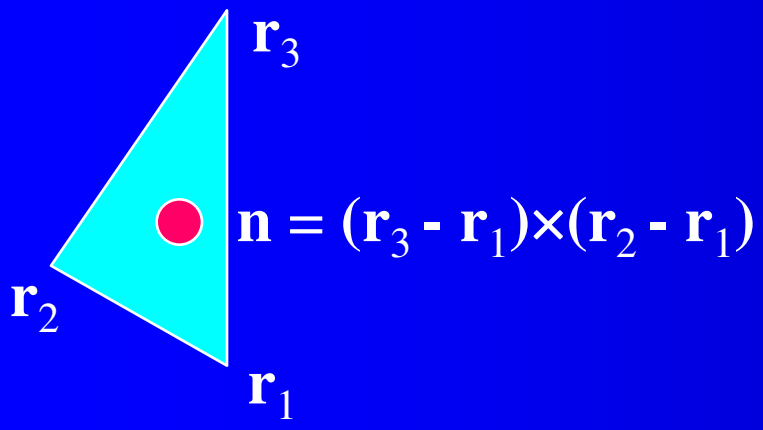


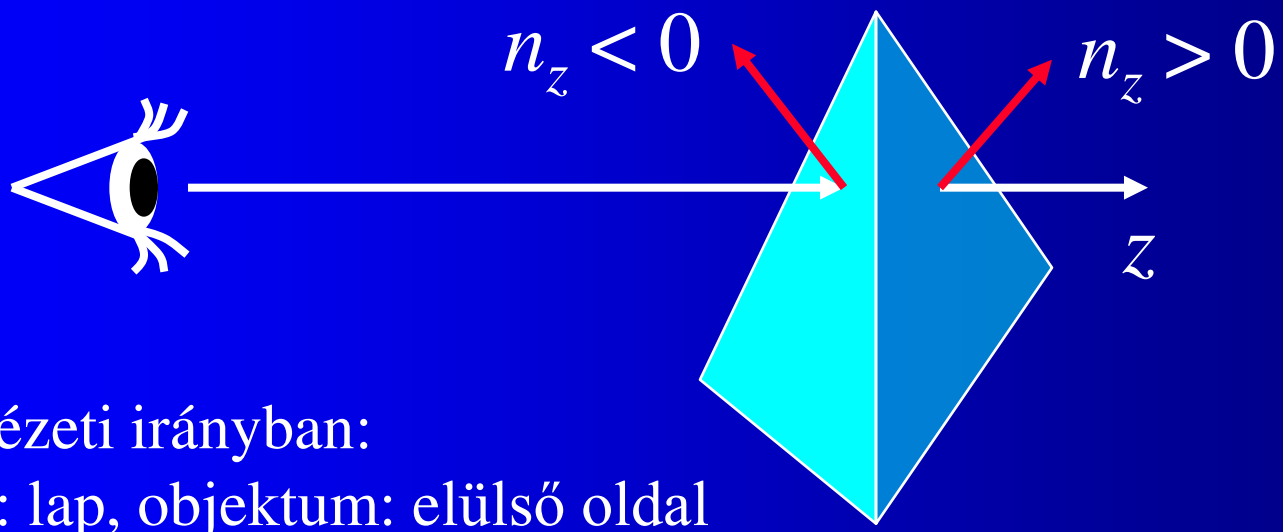
Tömör képszintézis



- 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





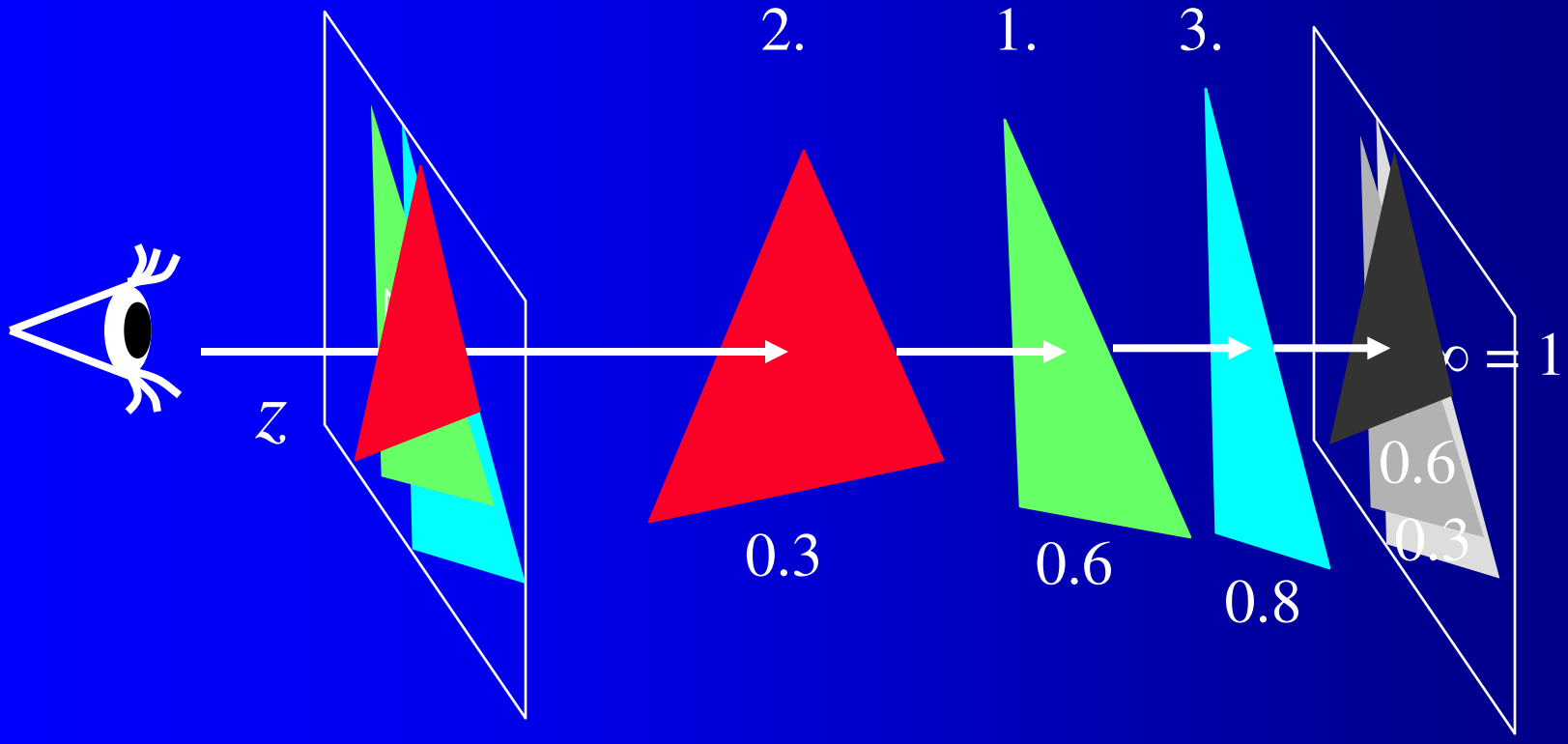
Lapok a nézeti irányban:

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

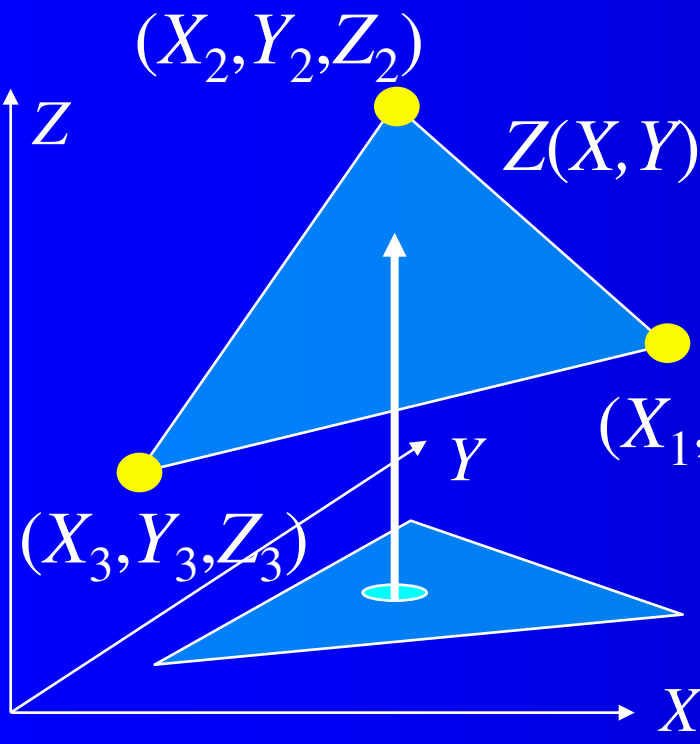
Feltételezés:

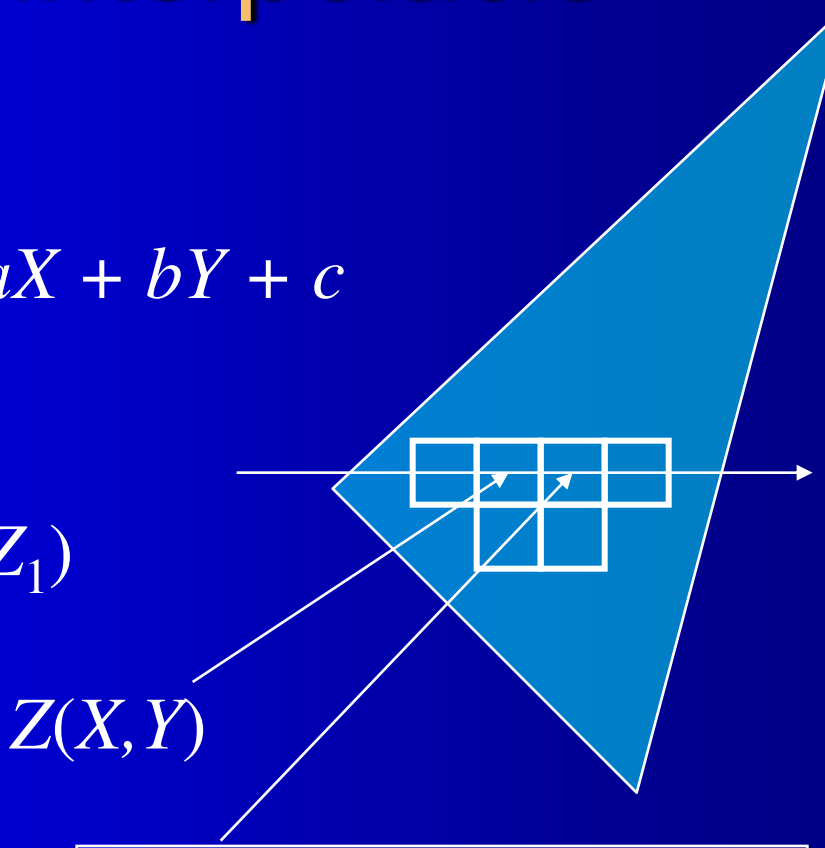
Ha kívülről, akkor csúcsok óramutatóval megegyező körüljárásúak

Z-buffer algoritmus



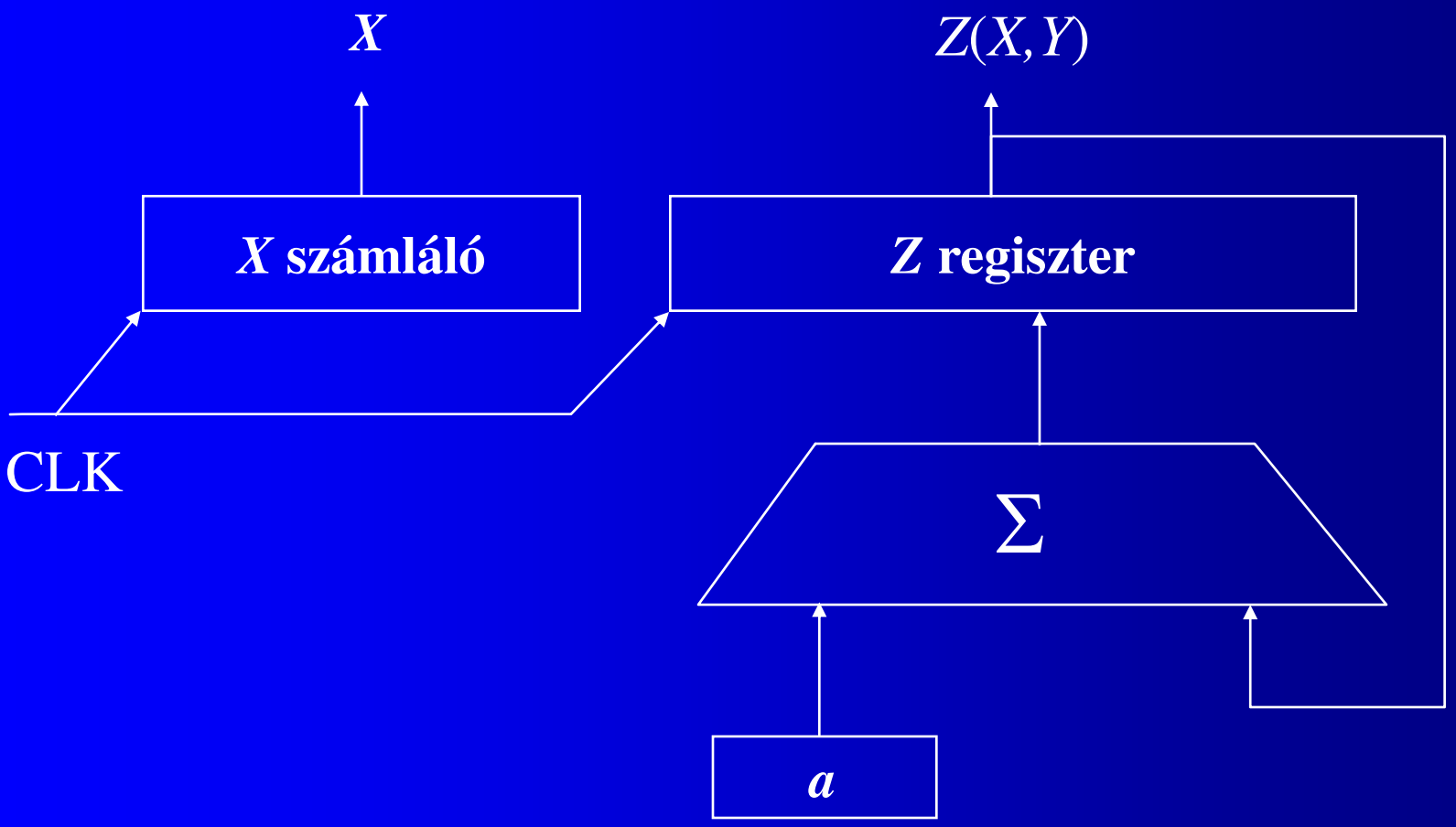
Z: lineáris interpoláció



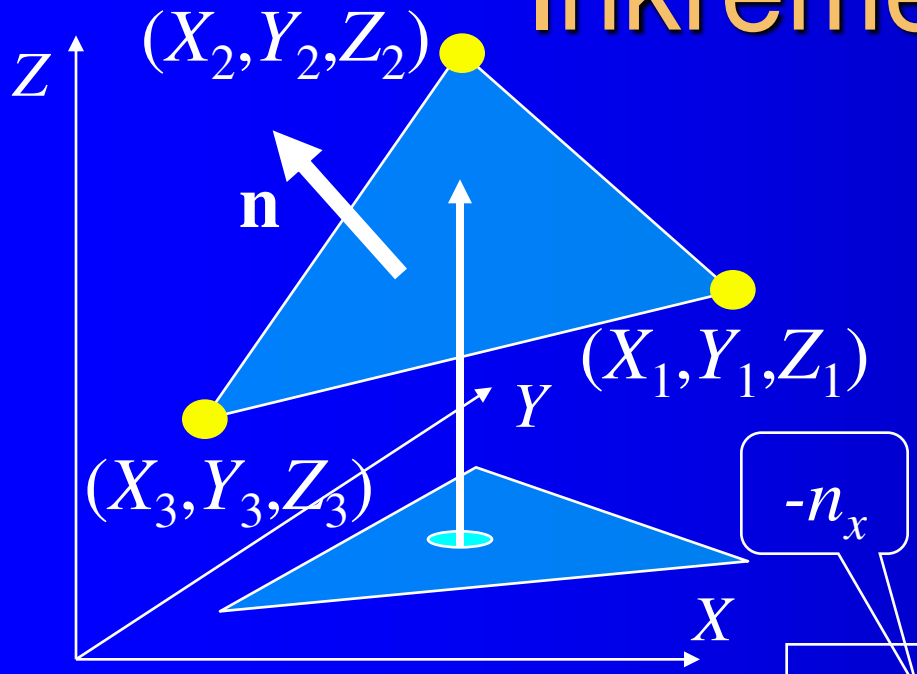


$$Z(X+1,Y) = Z(X,Y) + a$$

Z-interpolációs hardver



Inkremens (triangle setup)



$$Z_1 = aX_1 + bY_1 + c$$
$$Z_2 = aX_2 + bY_2 + c$$
$$Z_3 = aX_3 + bY_3 + c$$

$$Z_3 - Z_1 = a(X_3 - X_1) + b(Y_3 - Y_1)$$
$$Z_2 - Z_1 = a(X_2 - X_1) + b(Y_2 - Y_1)$$

$-n_x$

$$Z(X,Y) = aX + bY + c$$
$$n_xX + n_yY + n_zZ + d = 0$$

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

n_z

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

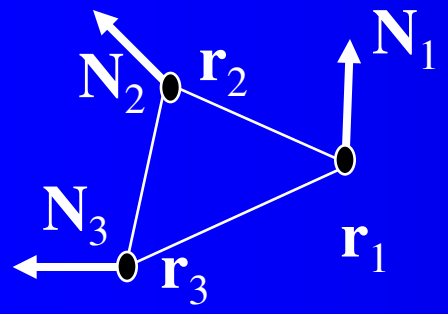
Árnyalás

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

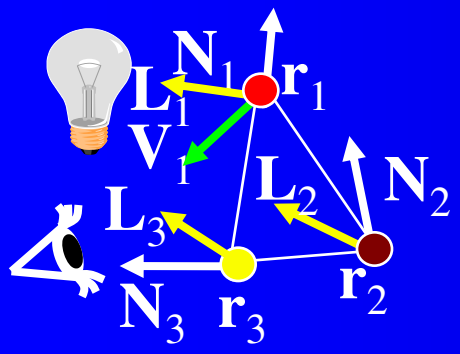


- Koherencia: ne mindent pixelenként
- Háromszögenként:
  - csúcspontonként 1-szer,
    belül az L „szín” lineáris interpolációja:
    Gouraud árnyalás, (per-vertex shading)
  - pixelenként,
    a **N**ormál (**V**iew, **L**ight) vektort interpoláljuk:
    Phong árnyalás, (per-pixel shading)

Gouraud (per-vertex) árnyalás



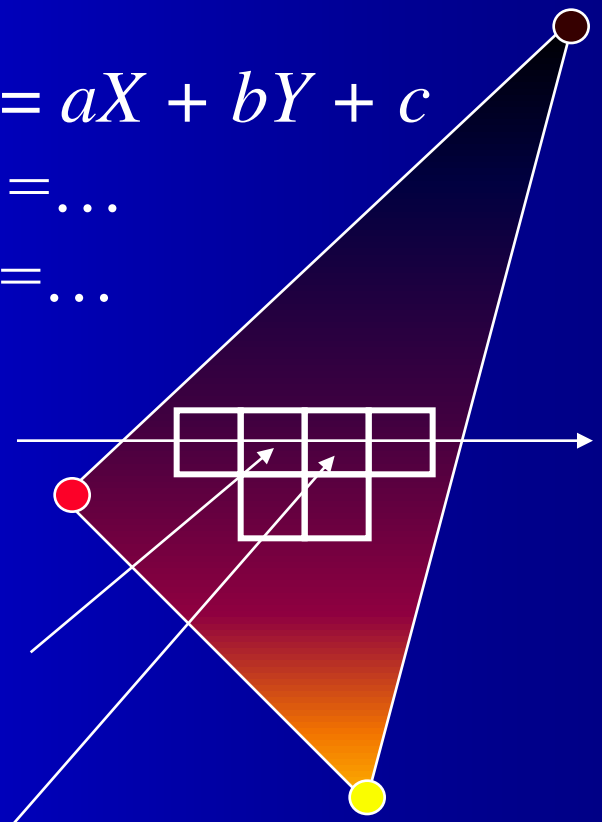
Modell
[+kamera]



(per-vertex shading)

Perspektív
+ nézeti

$$R(X,Y) = aX + bY + c$$
$$G(X,Y) = \ldots$$
$$B(X,Y) = \ldots$$

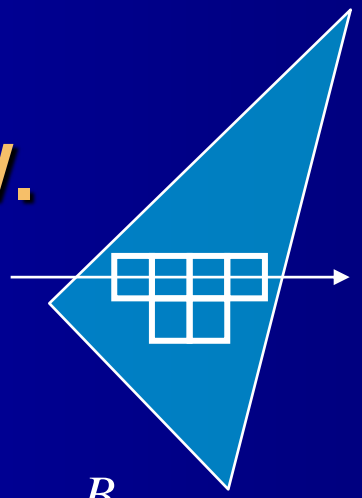


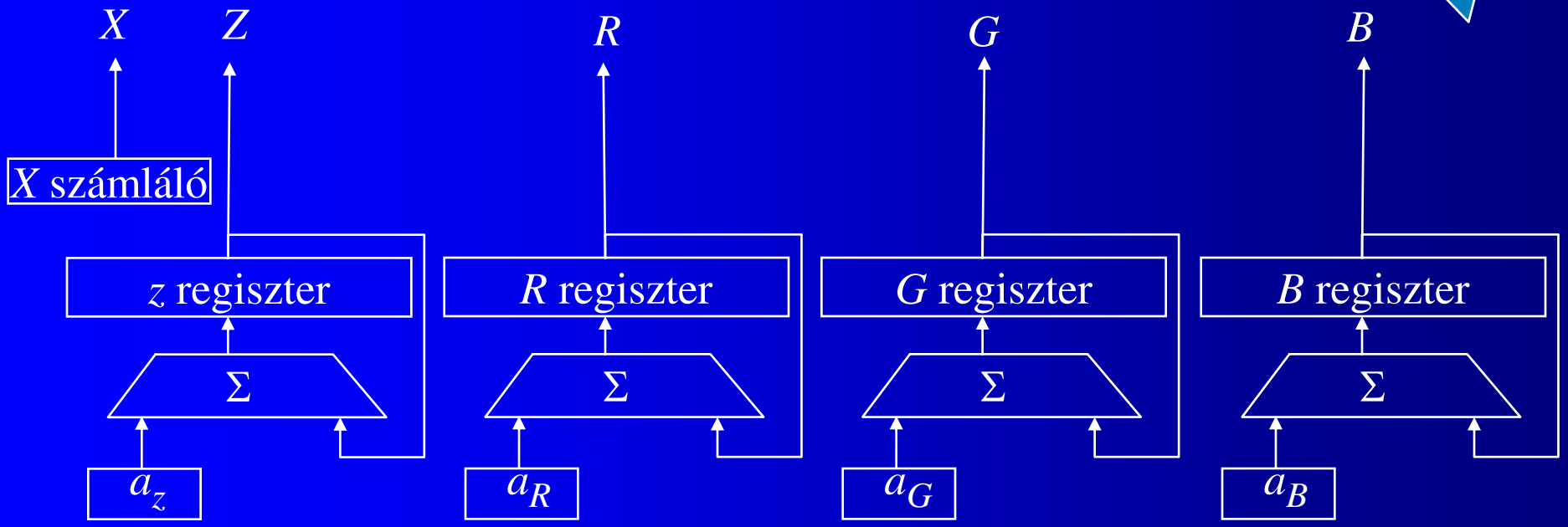
$R(X,Y)$

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

Képernyő

Z-buffer + Gouraud árnyaló hw.

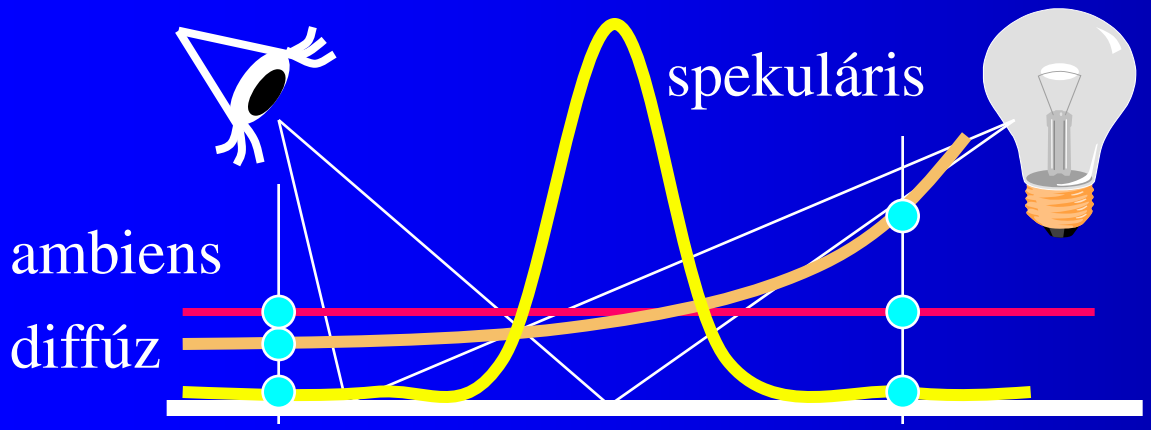




Gouraud árnyalás hasfájásai





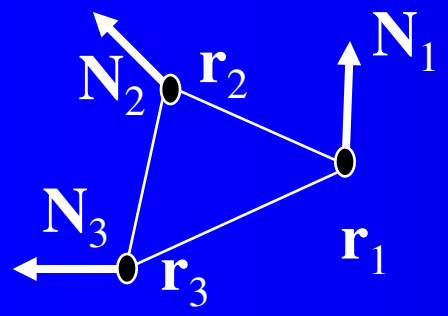


<u>További bajok:</u>

- anyagtulajdonság konstans
- árnyék nincs

különben a színt nem lehet interpolálni

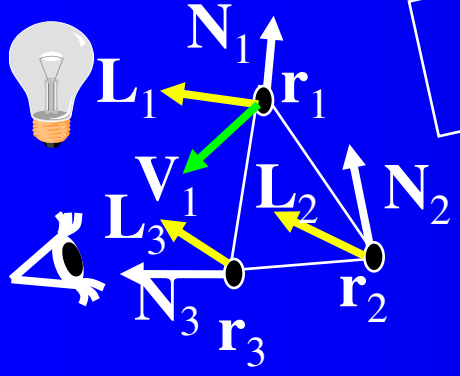
Phong (per-pixel) árnyalás



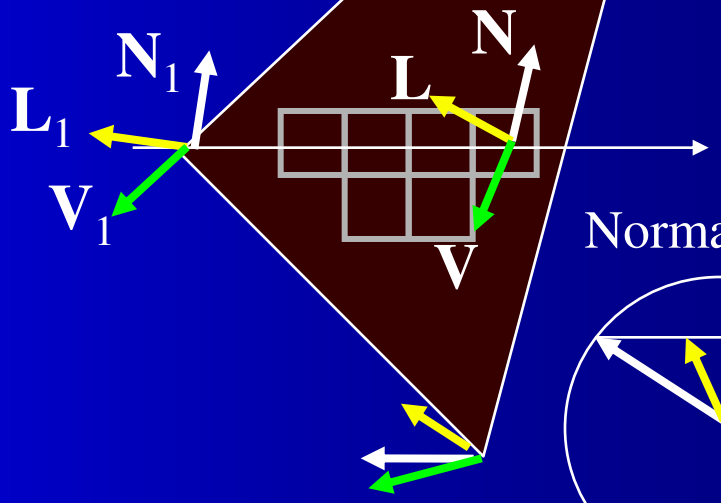
$$\mathbf{N}(X,Y) = \mathbf{A}X + \mathbf{B}Y + \mathbf{C}$$
$$\mathbf{L}(X,Y) = \ldots$$
$$\mathbf{V}(X,Y) = \ldots$$

Modell
[+kamera]

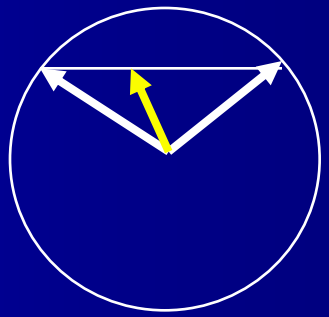
Vektor másolás



Pontokra:
perspektív
+ nézeti



Normalizálás!

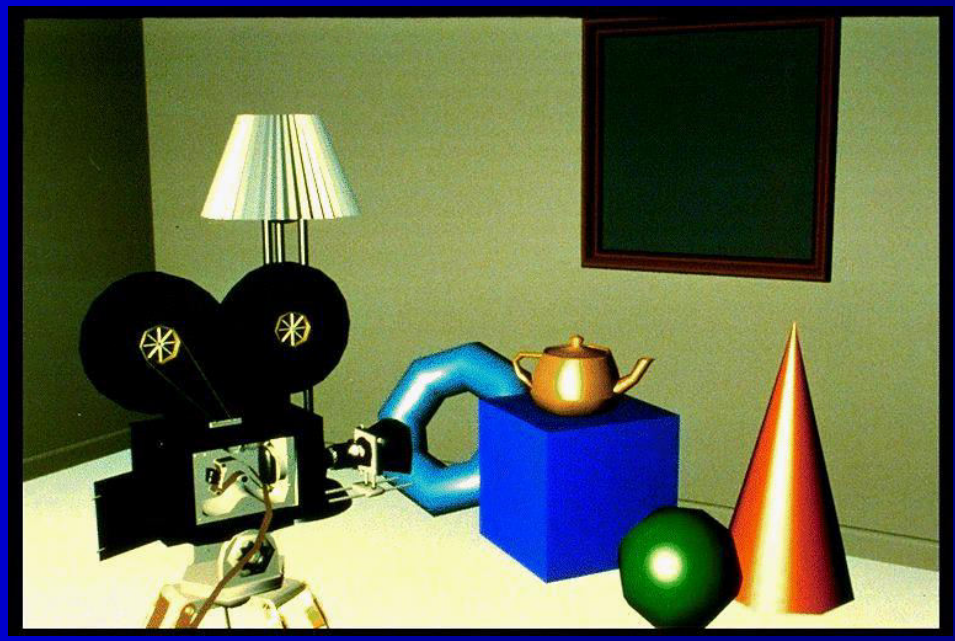


(per-pixel shading)

Képernyő

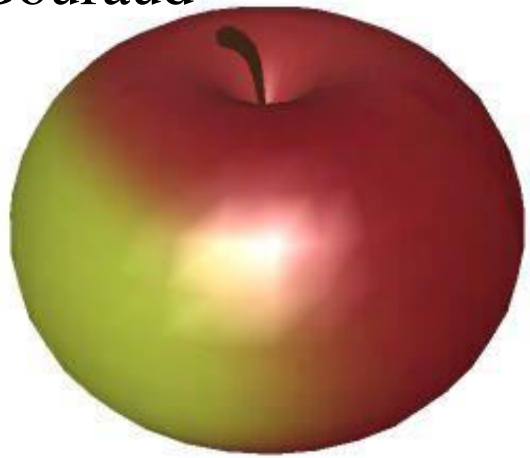
Phong árnyalás





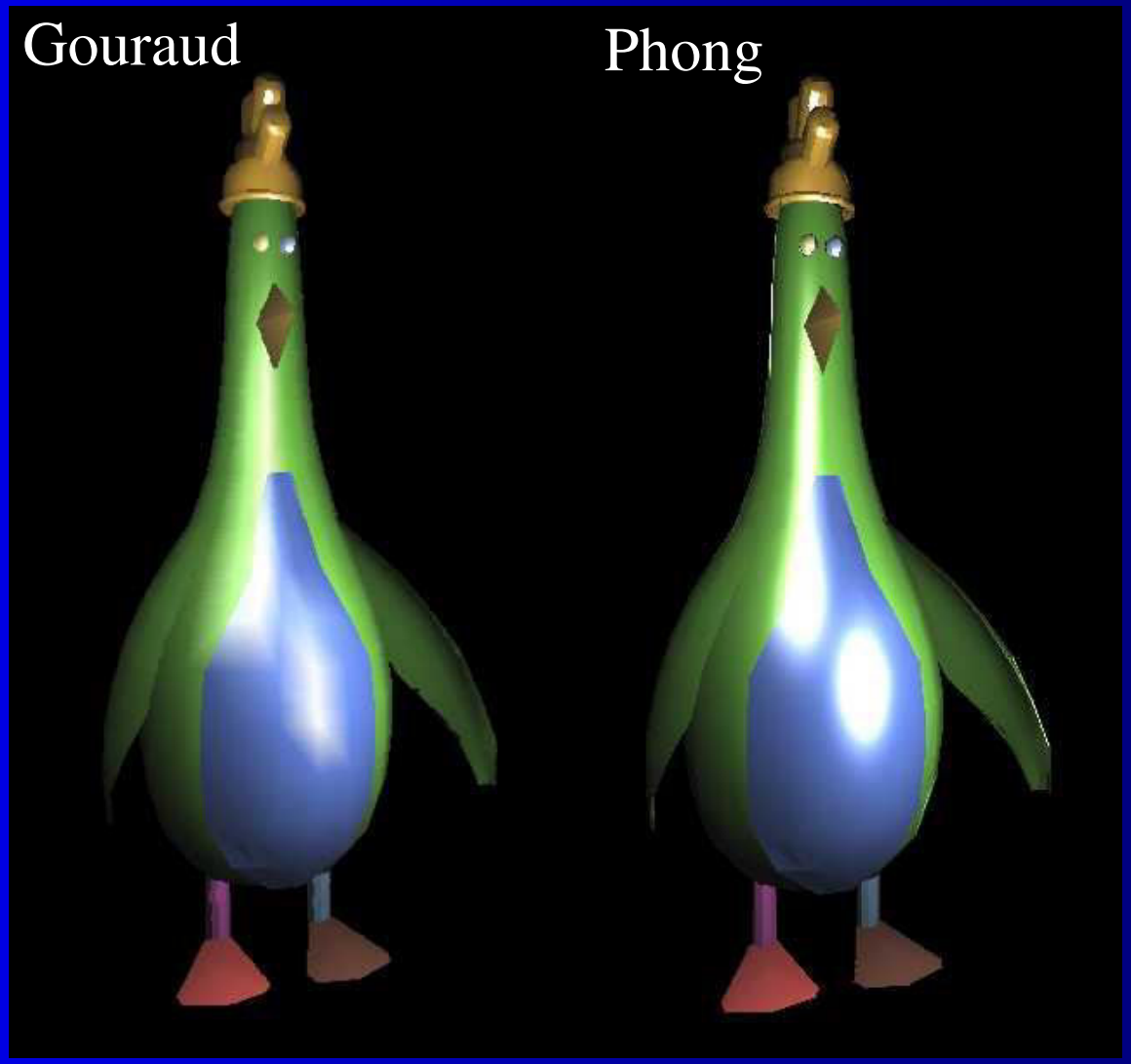
Gouraud versus Phong

Gouraud



Phong





Textúra leképzés:
anyagjellemzők változnak a felületen





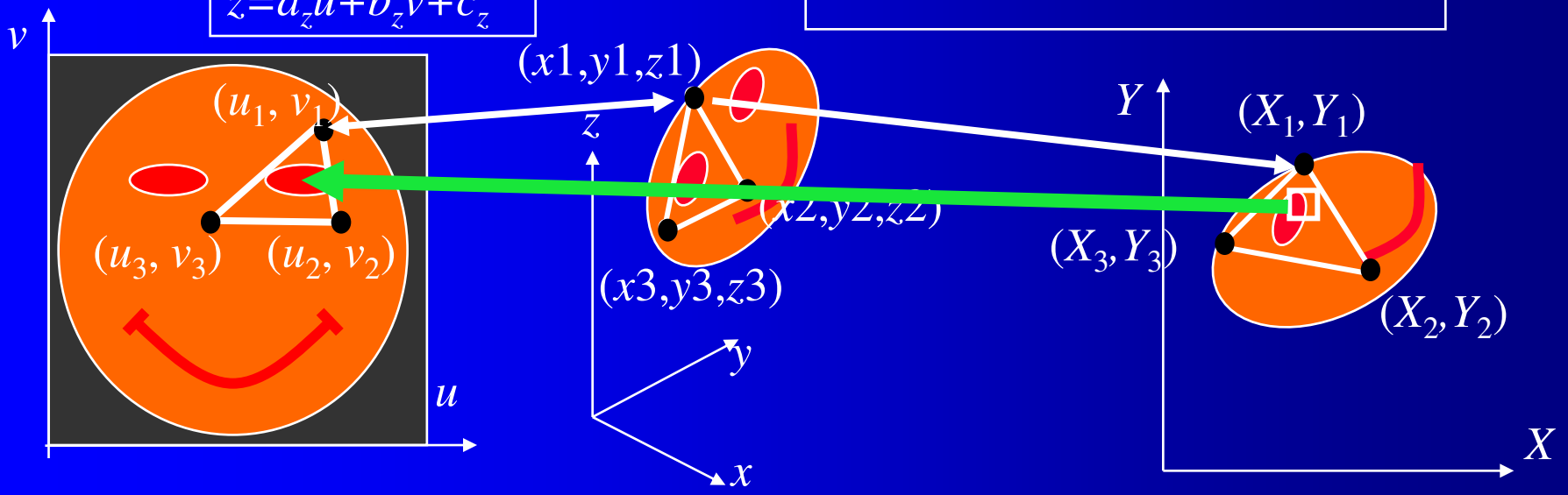
Inkrementális képszintézis

Paraméterezés

$x=a_x u+b_x v+c_x$
$y=a_y u+b_y v+c_y$
$z=a_z u+b_z v+c_z$

Model-camera-perspective-viewport transzformáció

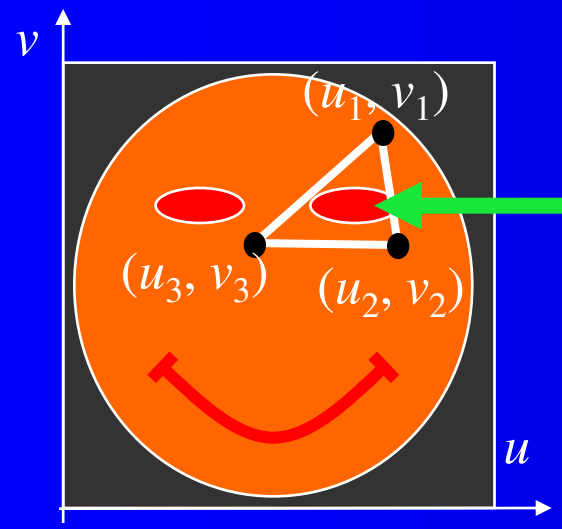
$[X_h,\ Y_h,\ Z_h,\ h] = [x,y,z,1]\mathbf{T}_C$
$(X,Y)= [X_h/h,\ Y_h/h]$



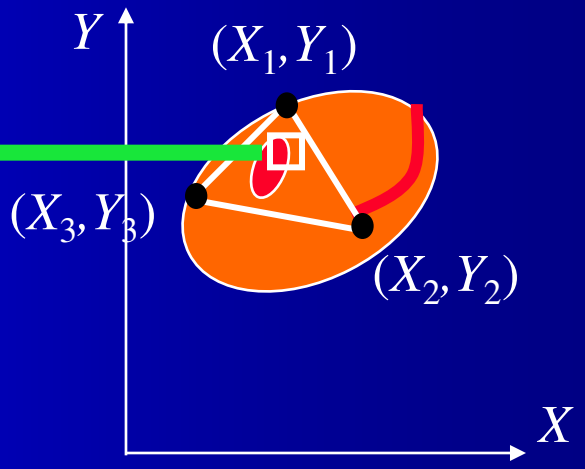
$[X \cdot h,\ Y \cdot h,\ h] = [u,v,1] \cdot \mathbf{T}$, ahol $h = -zc$

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

Lineáris interpoláció



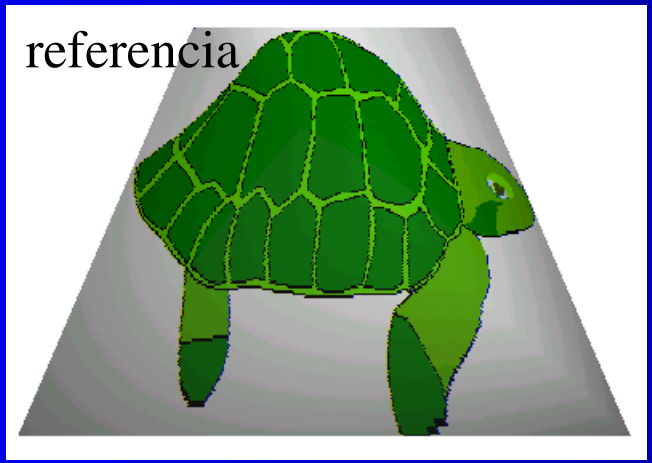
$$u = a_u X + b_u Y + c_u$$
$$v = a_v X + b_v Y + c_v$$

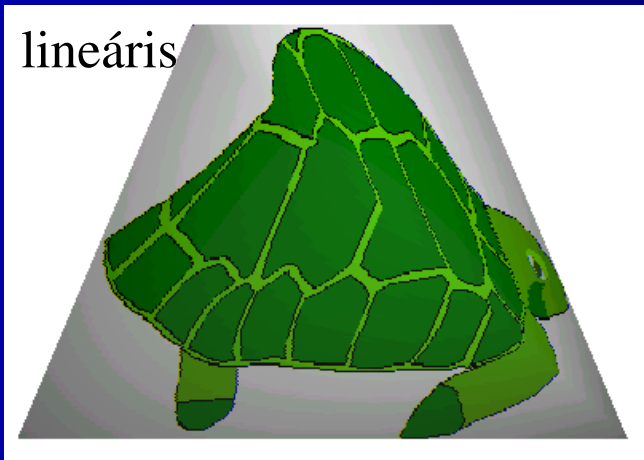


Interpolációs kényszerek:

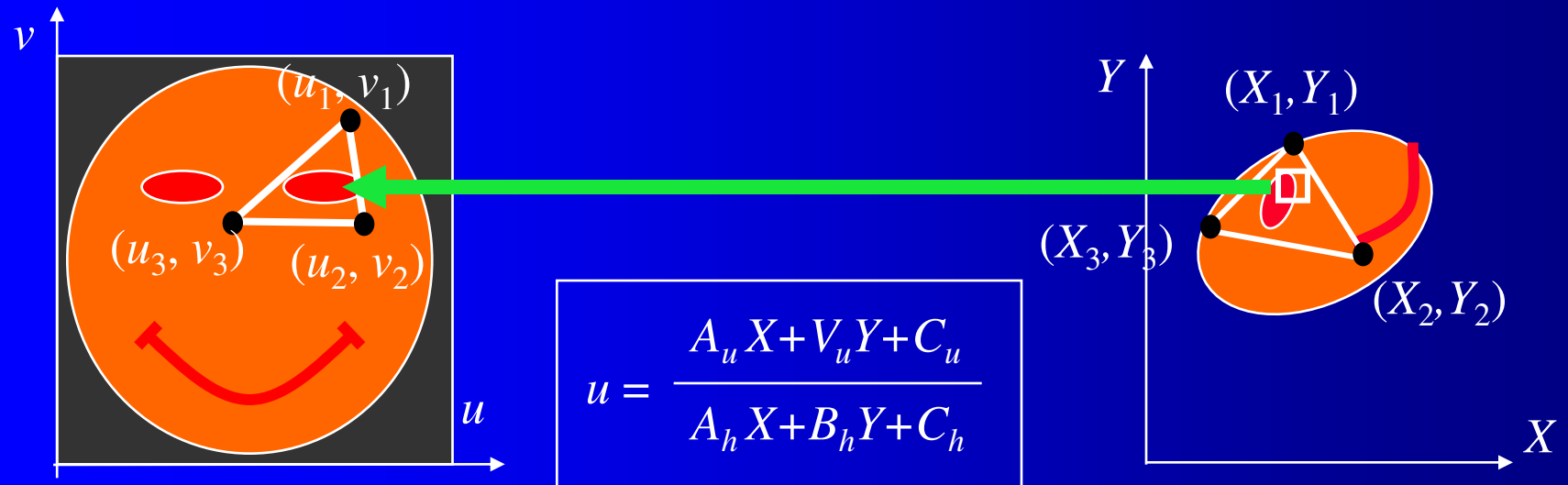
$$u_1 = a_u X_1 + b_u Y_1 + c_u$$
$$u_2 = a_u X_2 + b_u Y_2 + c_u$$
$$u_3 = a_u X_3 + b_u Y_3 + c_u$$

$$v_1 = a_v X_1 + b_v Y_1 + c_v$$
$$v_2 = a_v X_2 + b_v Y_2 + c_v$$
$$v_3 = a_v X_3 + b_v Y_3 + c_v$$





Perspektíva helyes textúrázás

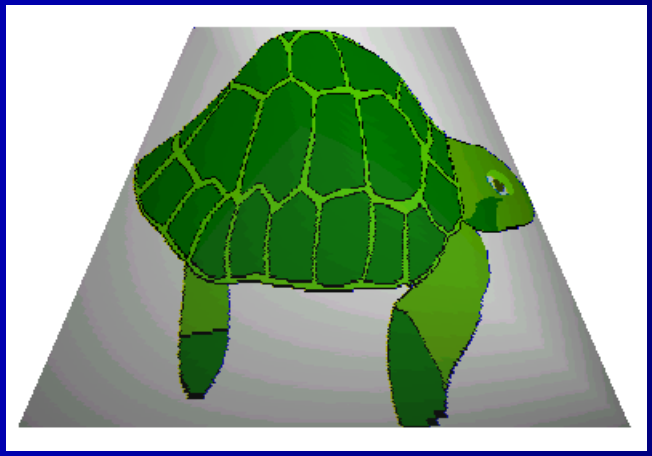


$$u = \frac{A_u X + V_u Y + C_u}{A_h X + B_h Y + C_h}$$

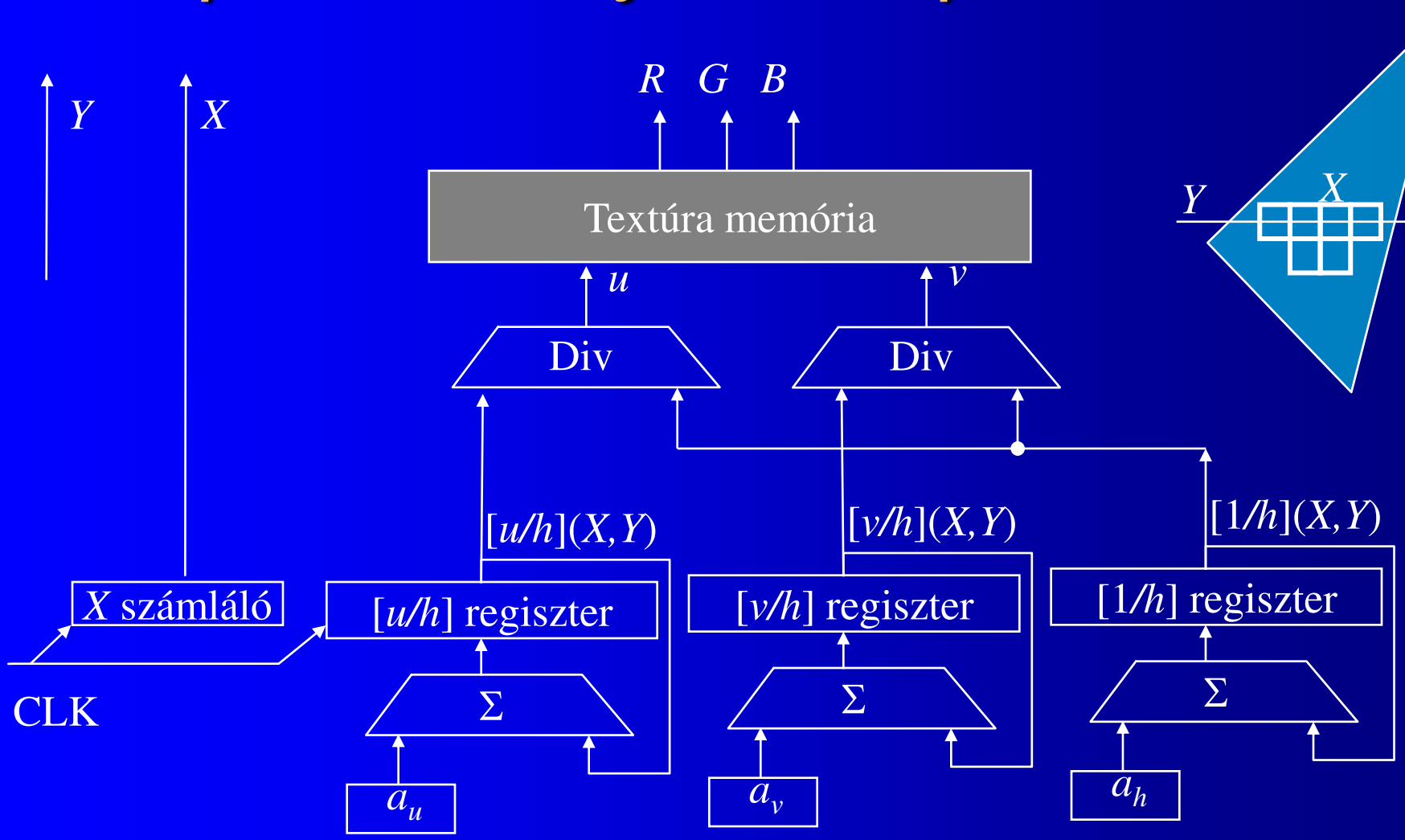
$$v = \frac{A_v X + B_v Y + C_v}{A_h X + B_h Y + C_h}$$

$$u/h = A_u X + B_u Y + C_u$$
$$v/h = A_v X + B_v Y + C_v$$
$$1/h = A_h X + B_h Y + C_h$$

$h = -zc$



Perspektíva helyes interpolációs hw.



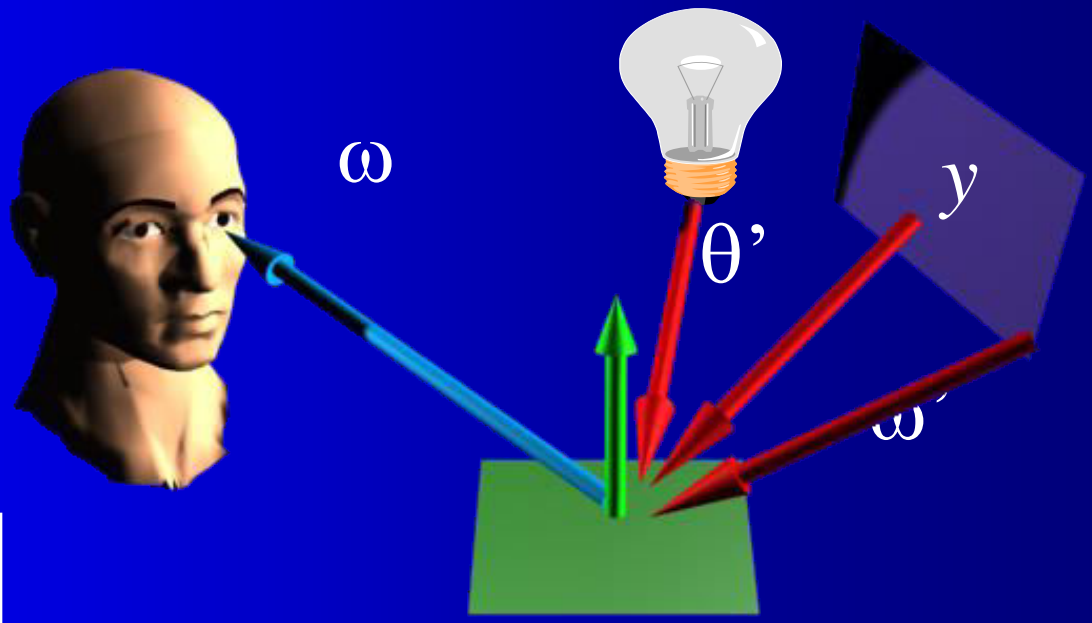
Globális illumináció (GI)

Szirmay-Kalos László

Árnyalási egyenlet

Radiancia = Emisszió + Megvilágítás * Visszaverődés

$$L(x,\ \omega) = L^{e}(x,\ \omega) + \int_{\Omega} L(y,\ \omega') \cdot f_r(\omega', \omega) \cos\theta'\, d\omega'$$



$$L = L^{e} + \tau\ L$$

GI megoldás

- Lokális illumináció

$$L = L^e + \tau L \approx L^e + \tau L^e$$

- Expanzió

$$L = L^e + \tau L = L^e + \tau L^e + \tau^2 L =$$

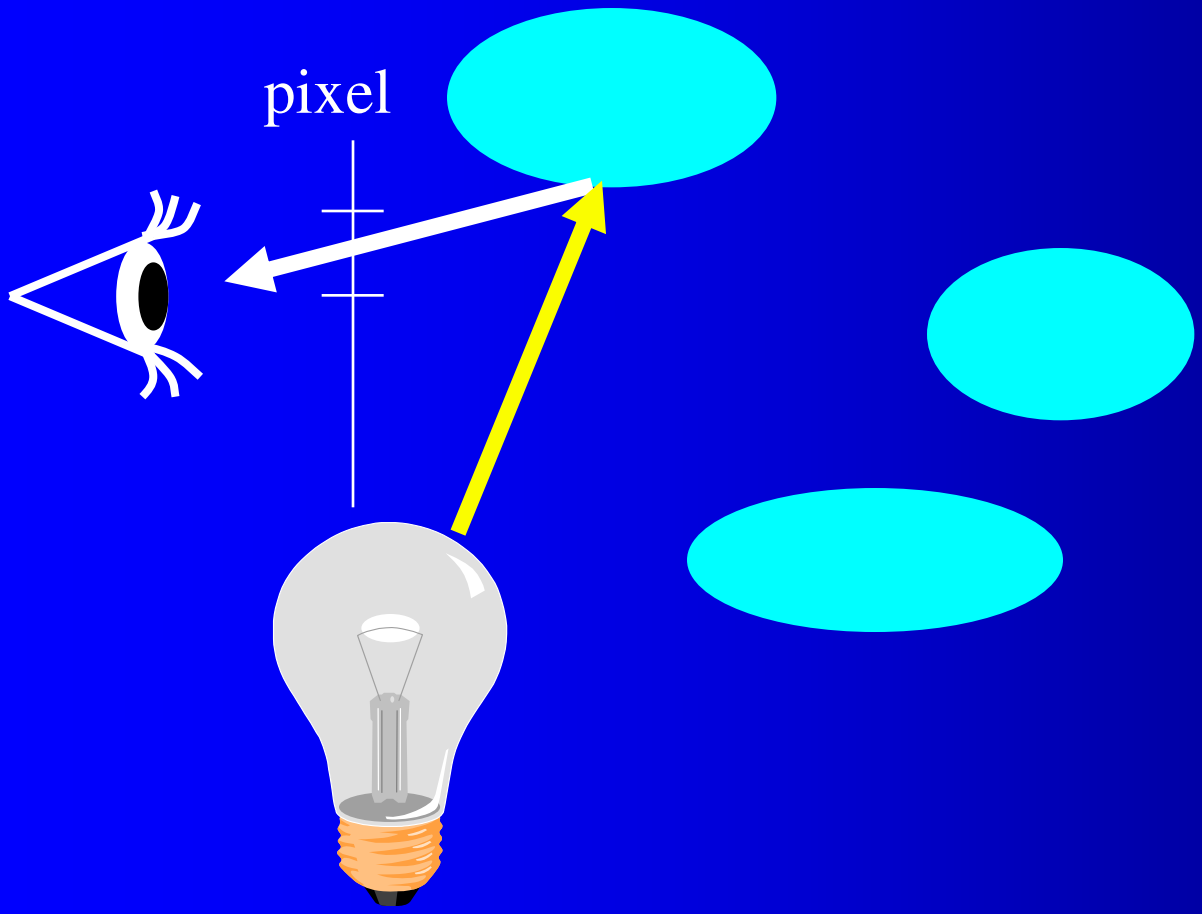
$$L^e + \tau L^e + \tau^2 L^e + \tau^3 L = \Sigma\ \tau^n L^e =$$

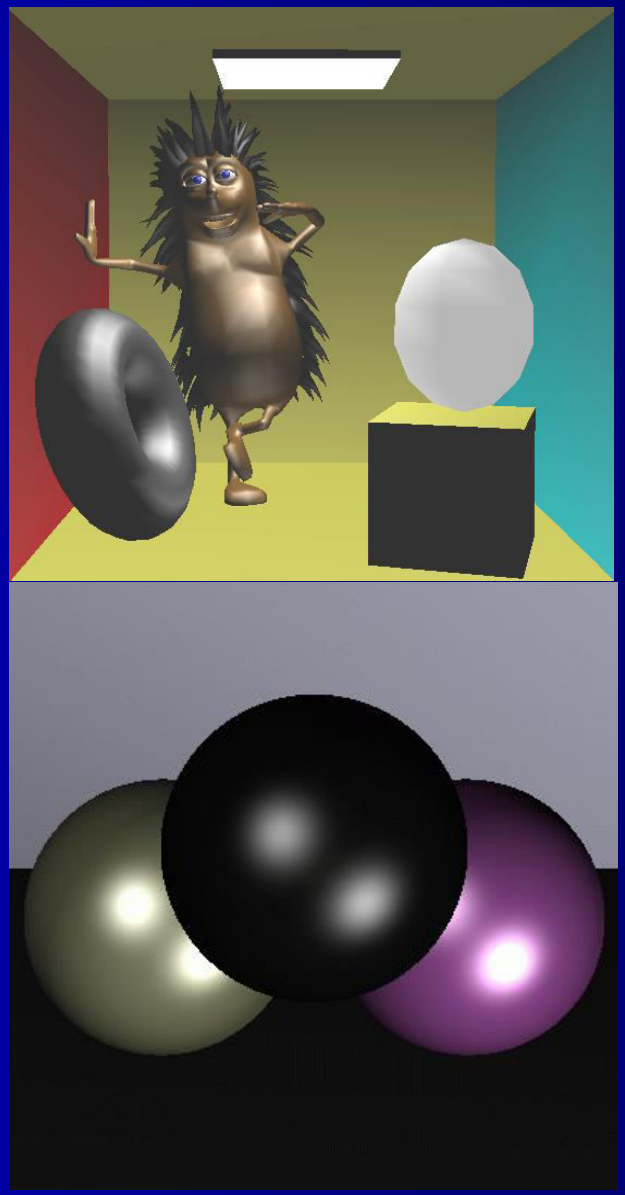
$$L^e + \tau(L^e + \tau(L^e + \ldots))$$

- Iteráció

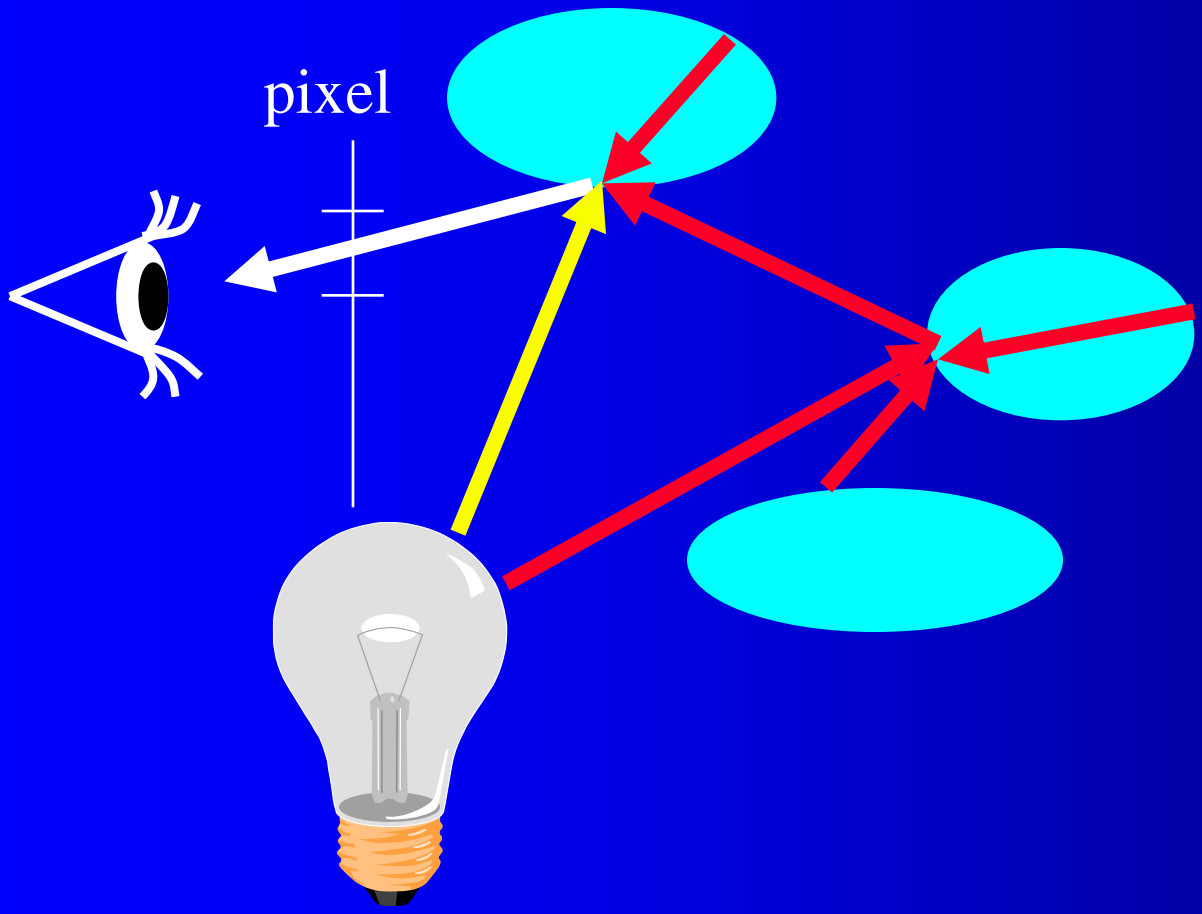
$$L_n = L^e + \tau L_{n-1}$$

Követett fényutak: Lokális illumináció

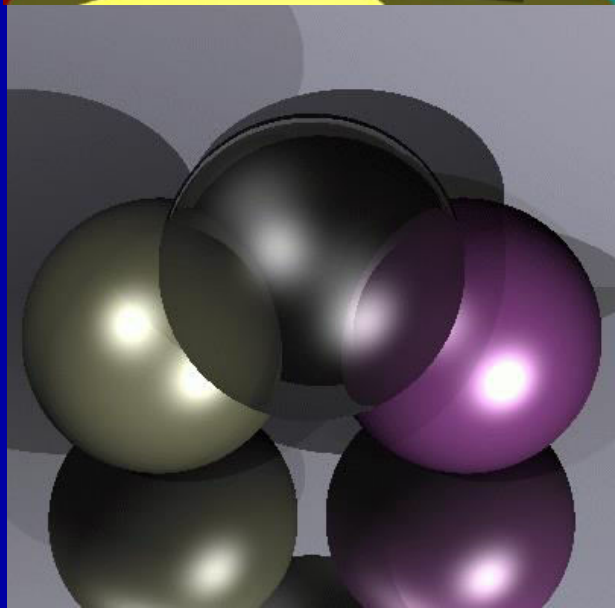




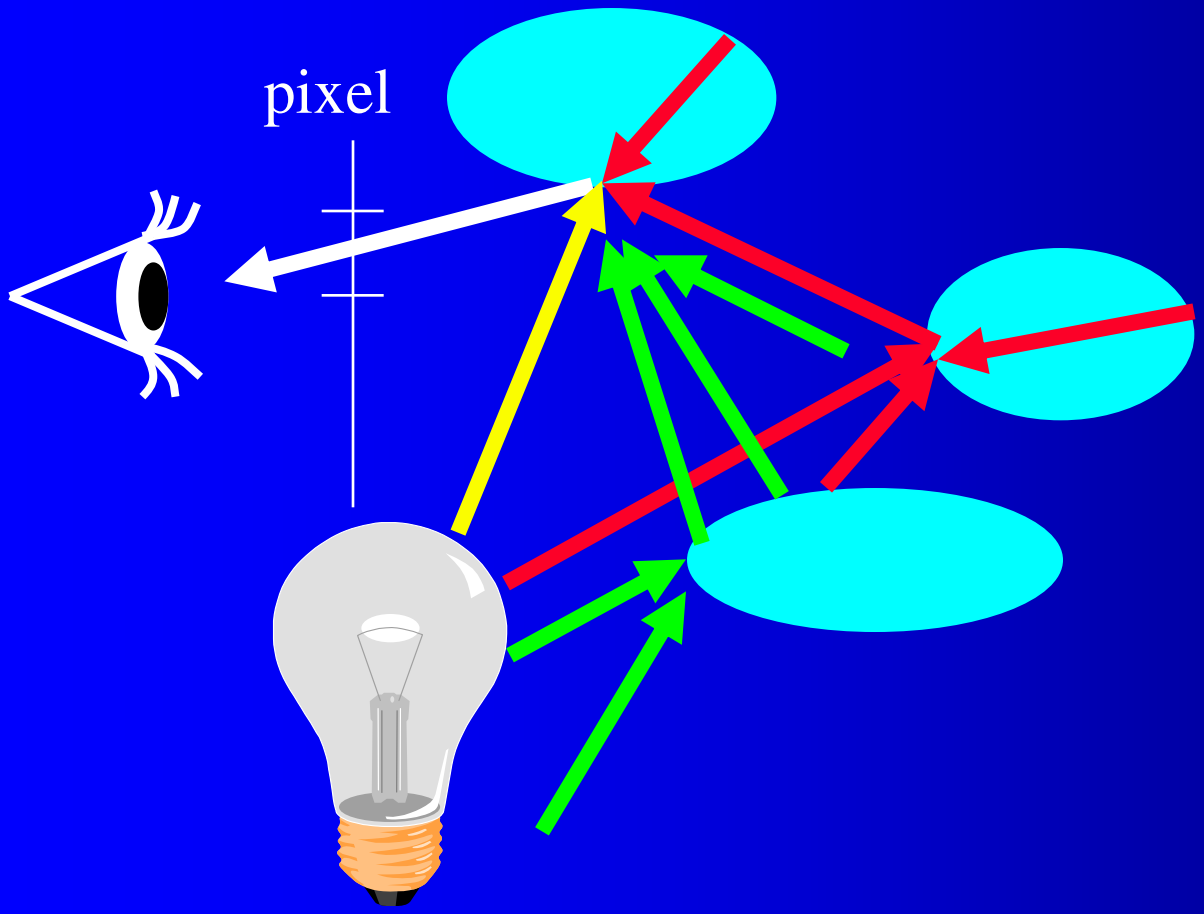
Követett fényutak: Sugárkövetés







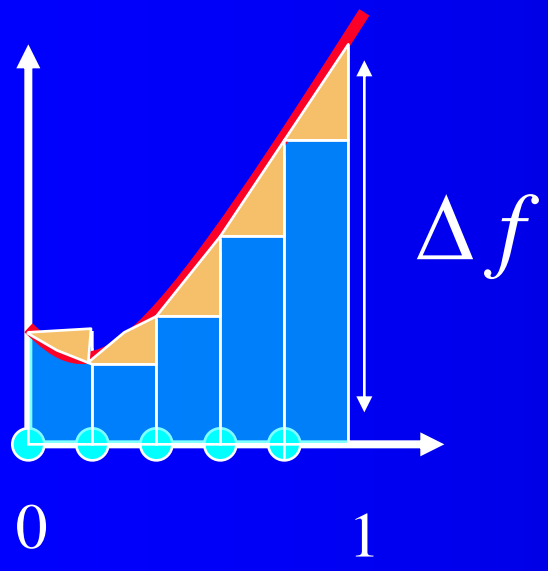
Követett fényutak: Globális illumináció







Numerikus integrálás



$$\int f(z)\,dz \approx 1/M \sum f(z_i)$$

M minta

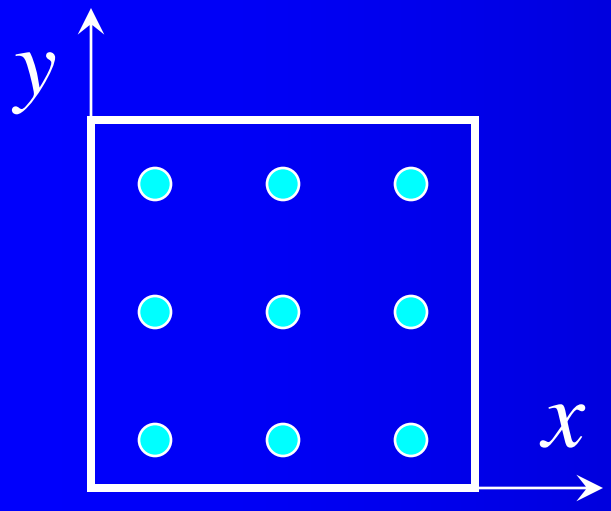
Átlagos magasság

alap

Háromszögek Száma

$$\text{Hiba} = \Delta f/2/M \cdot 1/M \cdot M = \Delta f/2/M = O(M^{-1})$$

Magasabb dimenziókban



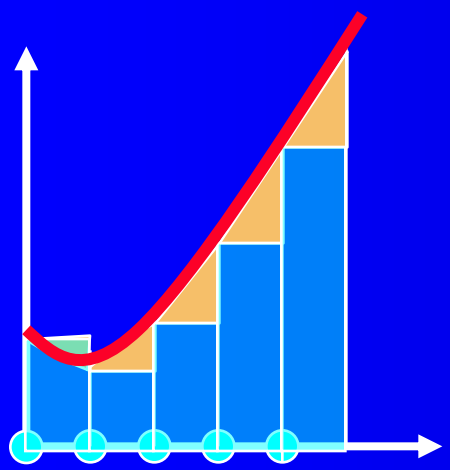
$$\int \int f(x,y)\,\mathrm{d}y\,\mathrm{d}x = \int F(x)\,\mathrm{d}x$$

$F(x)$

$n = \sqrt{M}$ mintaszám

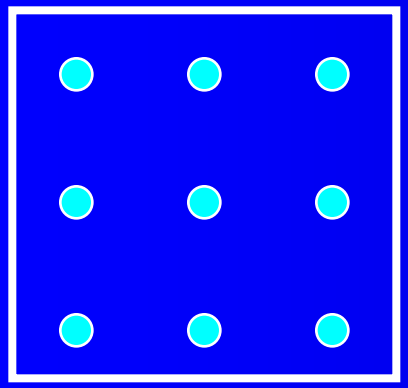
$$\text{Error} = O(n^{-1}) = O(M^{-0.5})$$

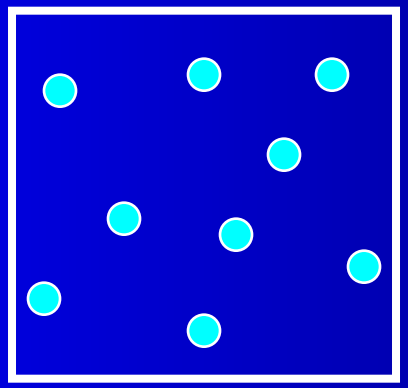
Sűrű minták magasabb dimenziókban

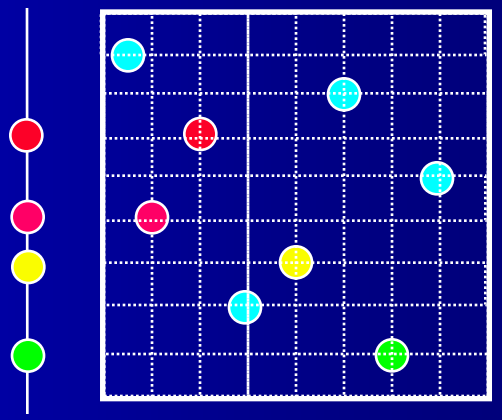


Véletlen:
Monte Carlo method

Determinisztikus,
Alacsony diszkrepancia
quasi Monte Carlo



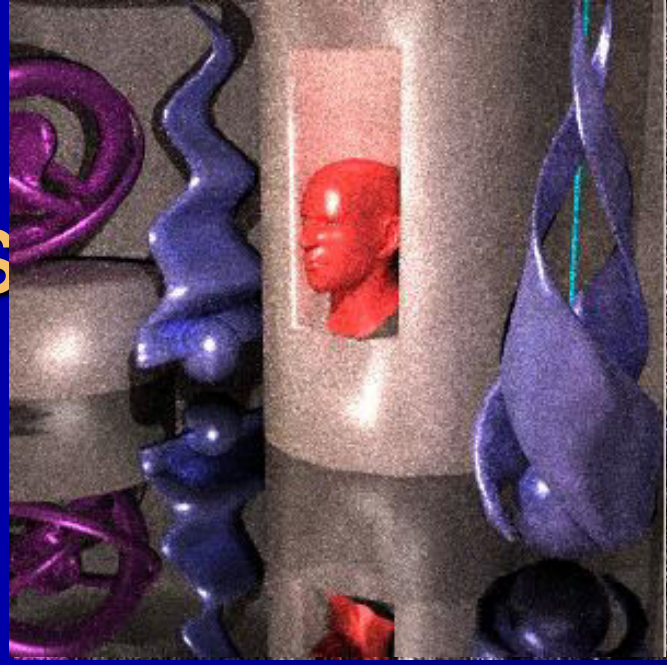






Monte Carlo Integrálás

Integrál = várható érték



100 samples/pixel

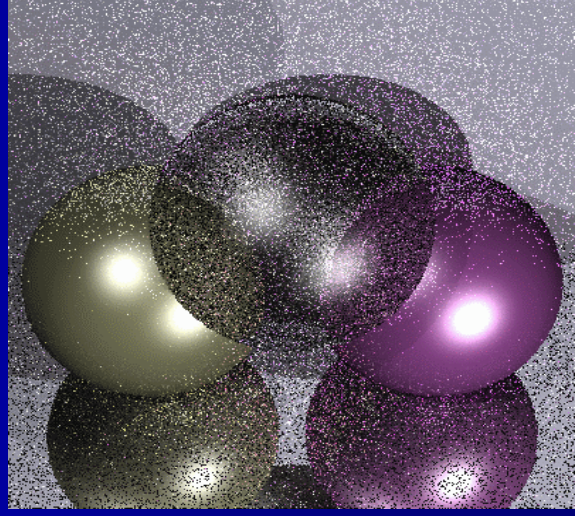
$$\int f(z)\,dz = \int f(z)/p(z)\cdot p(z)\,dz =$$

$$= E[f(z)/p(z)] \approx 1/M\ \Sigma\ f(z_m)/p(z_m)$$

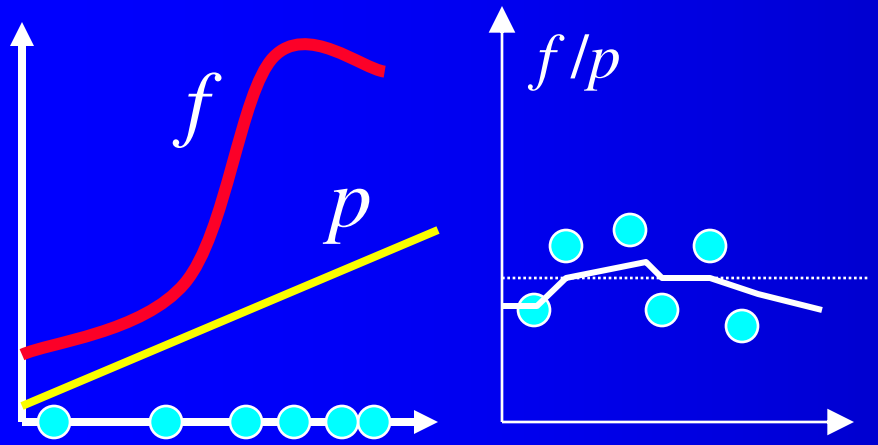
Hiba < 3·(f/p szórása) ·$M^{-1/2}$

99.7% konfidencia szinttel

Fontosság szerinti mintavételezés

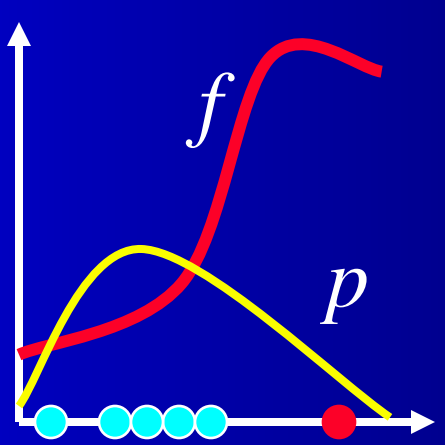


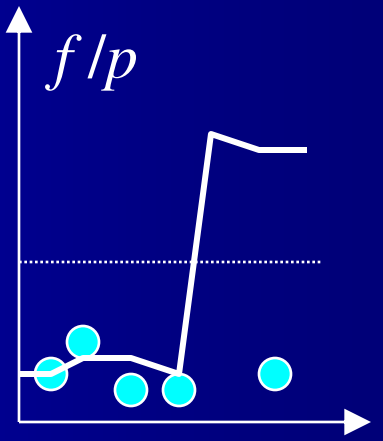
- f/p variációja legyen kicsi:
  ahol f nagy p is nagy



jó

hasonló f/p minták

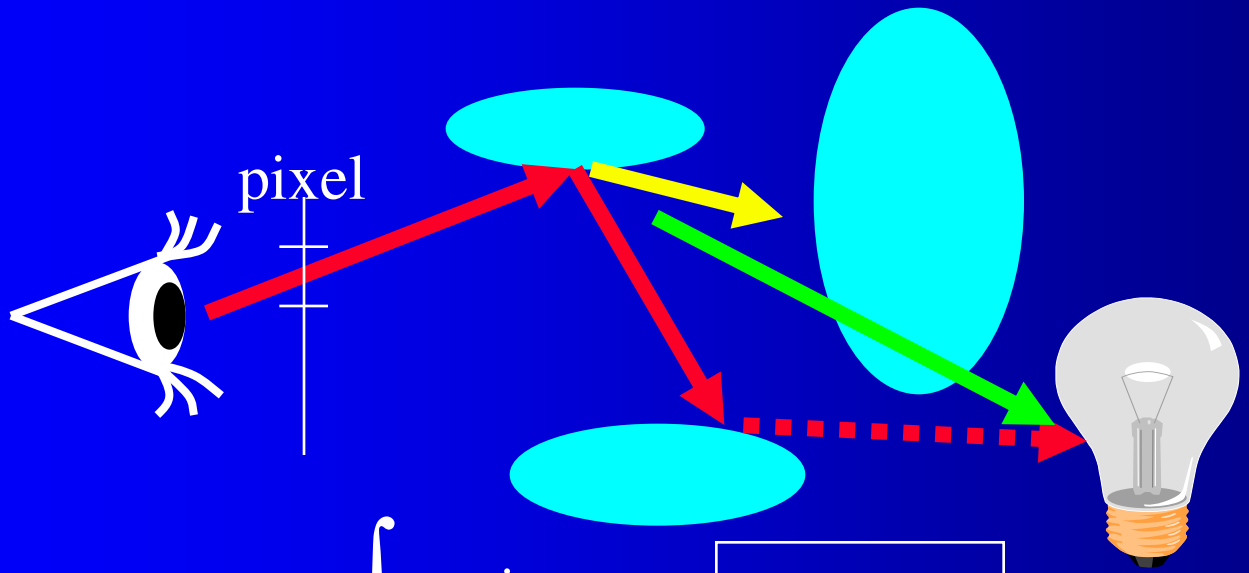




rossz

Ritka, nagy f/p minták

Véletlen bolyongás

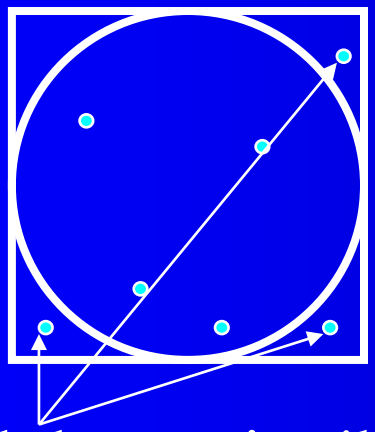


$$L = L^{e} + \int_{\Omega} L^{in}(\omega') \boxed{f_r \cos\theta'} d\omega'$$

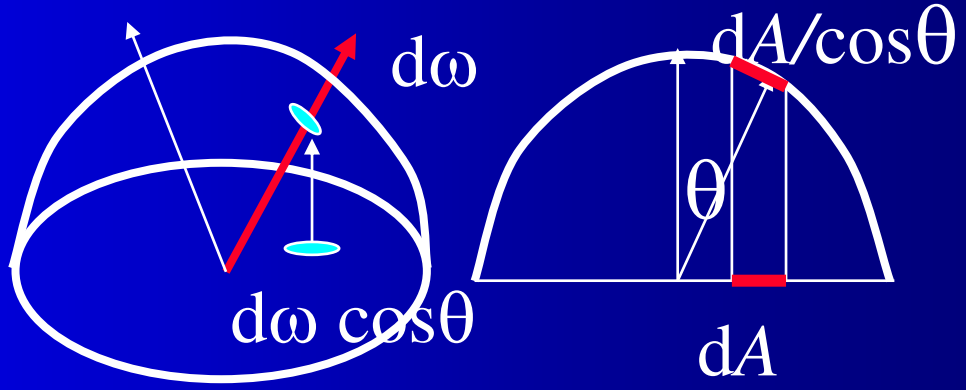
w: visszaverődés sűrűség

- BRDF mintavétel: arányos $f_r \cos\theta'$
- Fényforrás mintavétel

Diffúz BRDF mintavételezés



Eldobott minták



1. Egyenletes minták az egységnégyzetben
2. Körön kívüli minták eldobása

   do { $x = r_1, y = r_2$ } while $(x^2 + y^2 > 1)$
3. Gömbre vetítés

   $z = \sqrt{1 - x^2 - y^2}$

Végtelen dimenziós integrálok: Orosz rulett

1. MC integrál:

$$\int w_i(L^e + \ldots)\, d\omega_i = E[w_i(L^e + \ldots)\, / p(\omega_i)] = E[\, L^{refl}\,]$$

2. Számítsd ki s valószínűséggel, különben 0
3. Kompenzálj s-sel osztással

Várható érték:

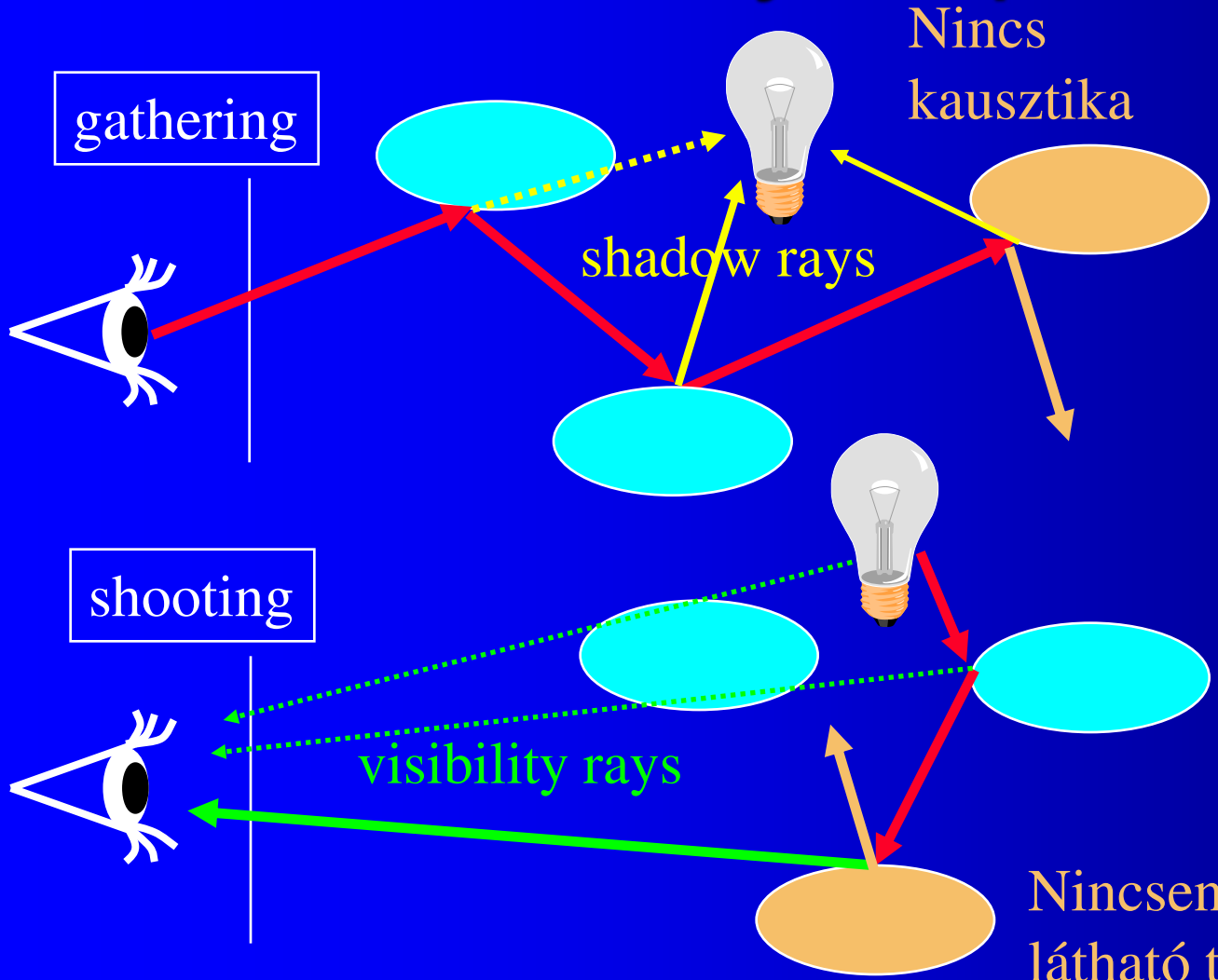
$$E[\, L^{refl*}\,] = s\, E[\, L^{refl}/s\,] + (1-s)\, 0 = E[\, L^{refl}\,]$$

Szórás nő:

$$D^2[\, L^{refl*}\,] = s\, E[\, (L^{refl}/s)^2\,] + (1-s)\, 0 - E^2[\, L^{refl}\,] =$$

$$(1/s - 1)\, E[(L^{refl})^2\,] + D^2[\, L^{refl}\,]$$

Fényút épités

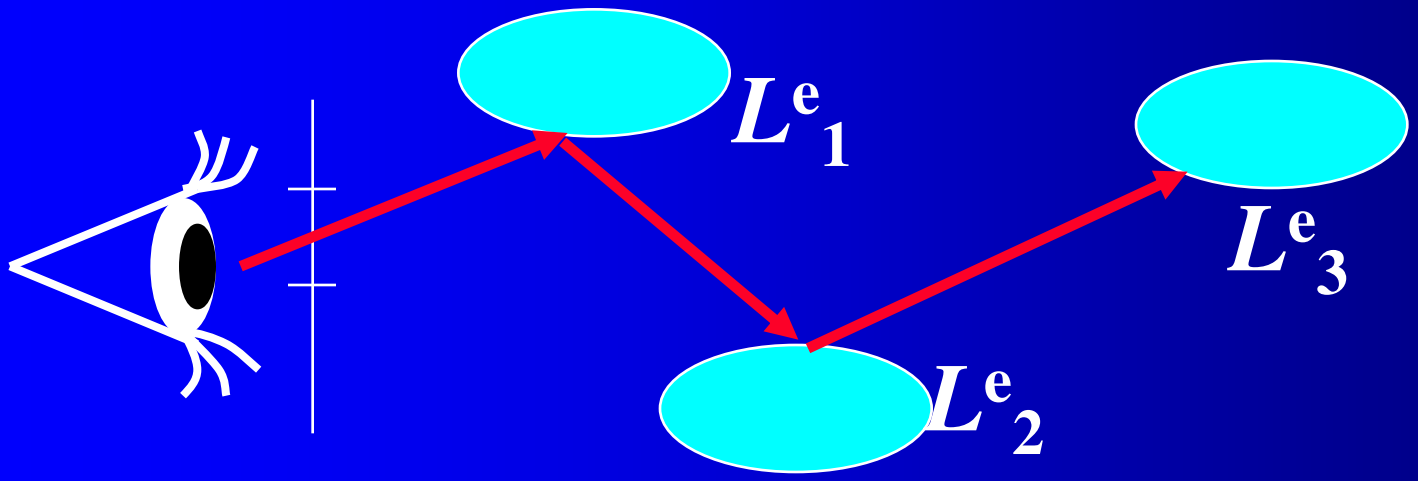


Nincs kausztika



Nincsenek látható tükrök és törés

Path tracing



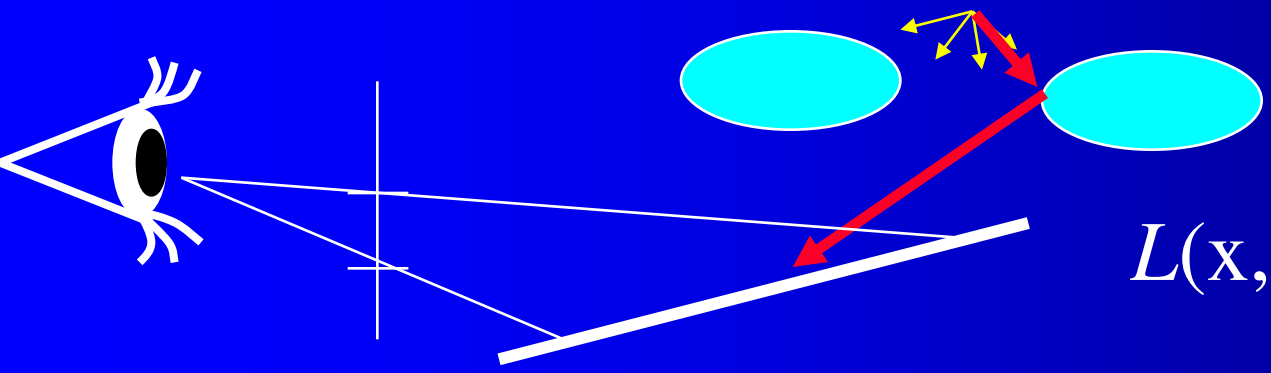
- BRDF mintavétel: Pr{köv irány} ~ Brdf cos θ‘
- Orosz rulett: Befejezés 1- a_i valószínűséggel
- $P = L^e_1 + L^e_2 \boxed{w_1/p_1/a_1} + L^e_3 \boxed{w_1/p_1/a_1} \boxed{w_2/p_2/a_2}$

≈ 1 ≈ 1 ≈ 1

Path Tracer

```
Color Trace(ray, depth)
    (object, x) = Intersect(ray)
    IF no intersection THEN RETURN L_sky
    color = Direct Lightsource(x, -ray.dir)
    if (depth == 0) color += Le (x, -ray.dir)
    prob = RussianRoulette (normal, -ray.dir)
    IF (prob == 0) RETURN color
    prob *= BRDFSampling (newdir, normal, -ray.dir)
    IF (prob == 0) RETURN color
    color += Trace(Ray(x, newdir), depth+1) *
                Brdf(newdir, normal, -ray.dir) cosθ' / prob
RETURN color
```

Light tracing

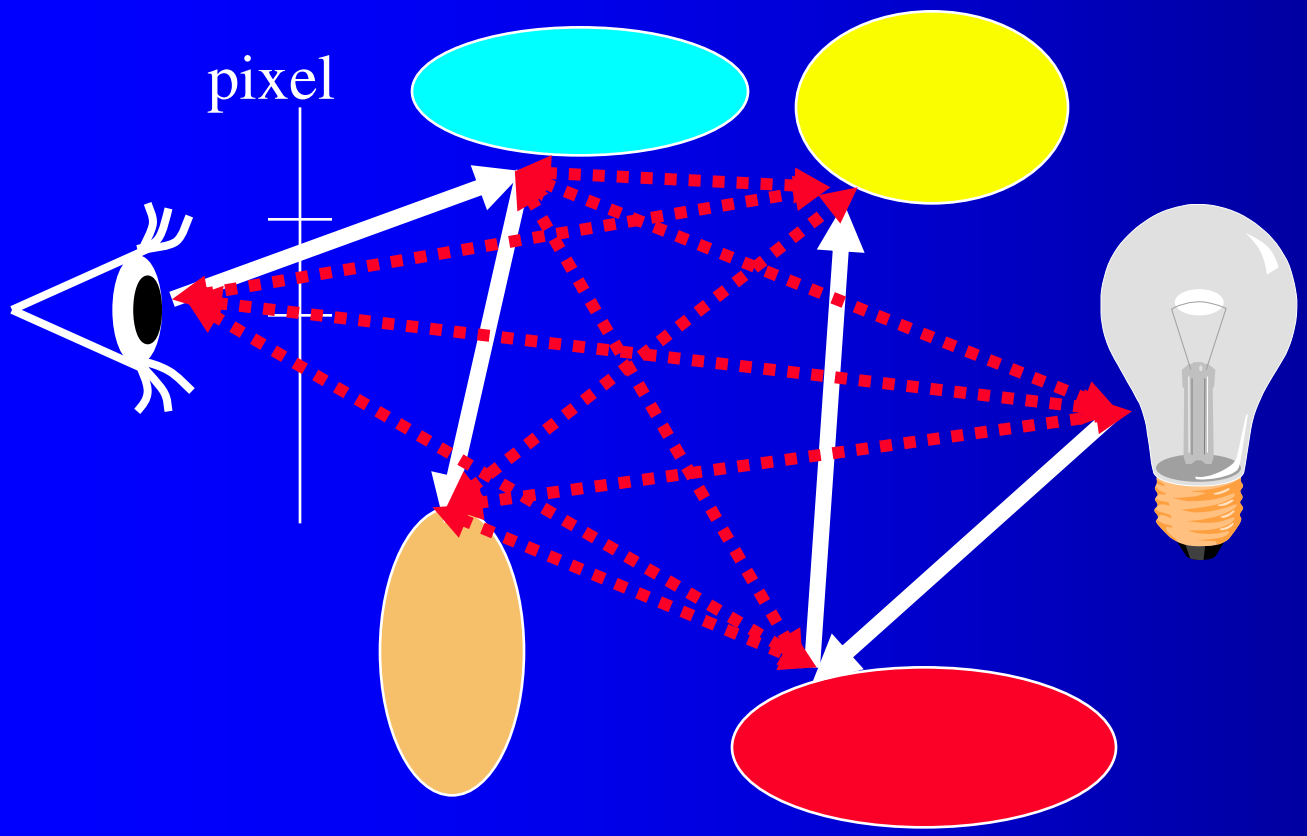


$$L(x,\omega) = \frac{d\Phi}{dA\ \cos\theta' d\omega}$$

- Source is selected with probability $p^e \propto L^e(x,\omega_1)\cos\theta$
- Pr{next direction} ~ BRDF $\cos\theta$
- Termination with 1- a_i
- $P = \boxed{L^e\cos\theta/p^e}\ \boxed{w_1/p_1/a_1}\ \boxed{w_2/p_2/a_2}\ \ldots\ w_{eye}\ g$

$\approx \Phi_{total}$ $\quad \approx 1$ $\quad \approx 1$

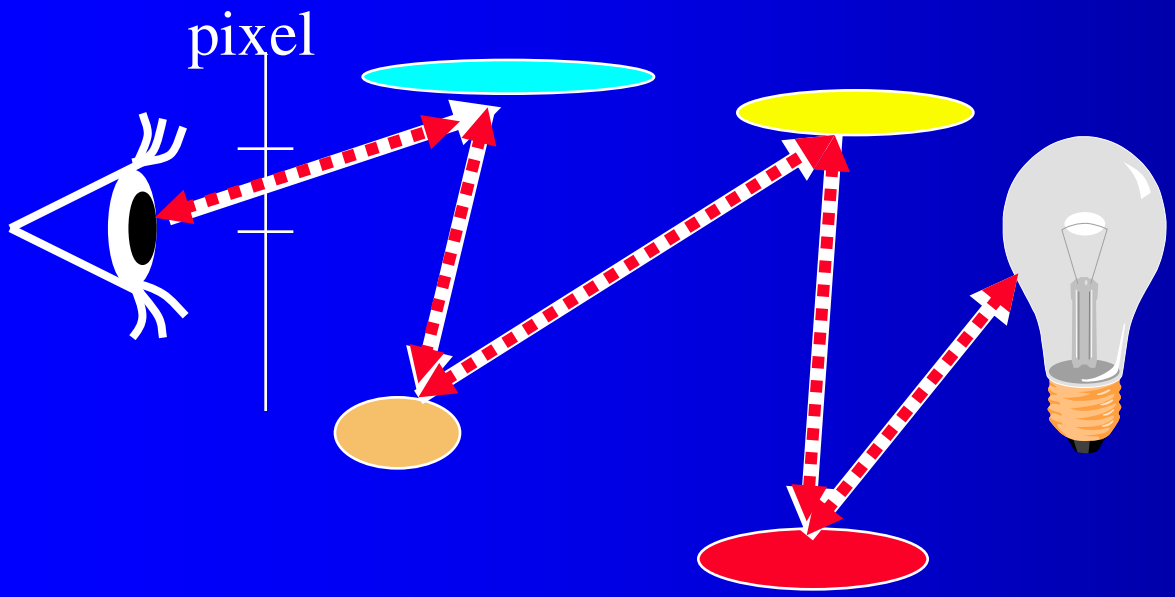
Bi-directional path tracing

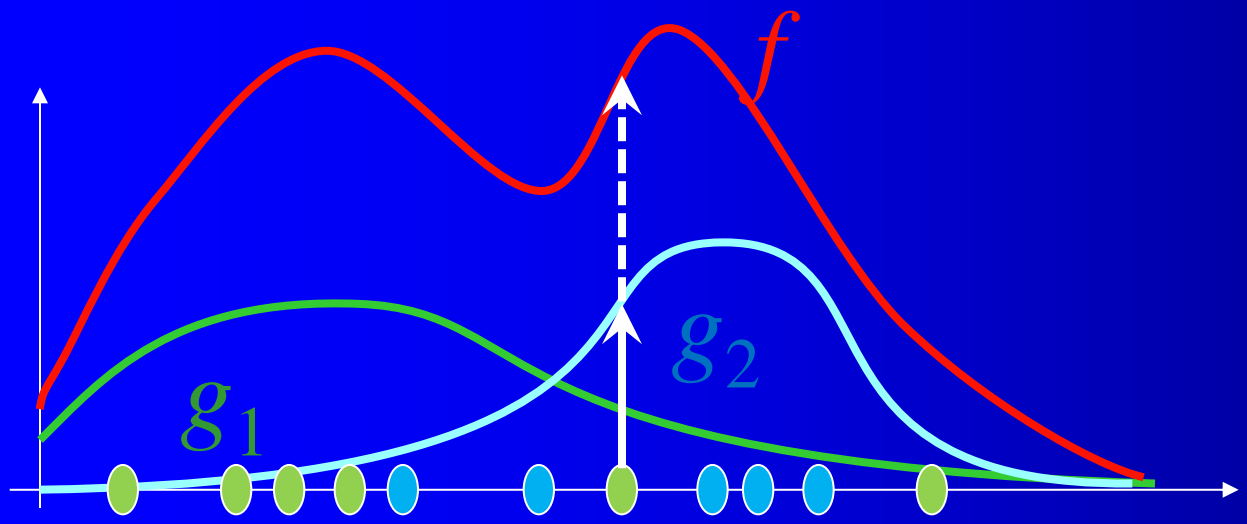




$$L = \Phi / (\mathrm{d}x \cos\theta \, \mathrm{d}\omega) = \Phi_{in} \frac{f_r(y) \cos\theta_y \, f_r(x) \cos\theta_x}{|x - y|^2}$$

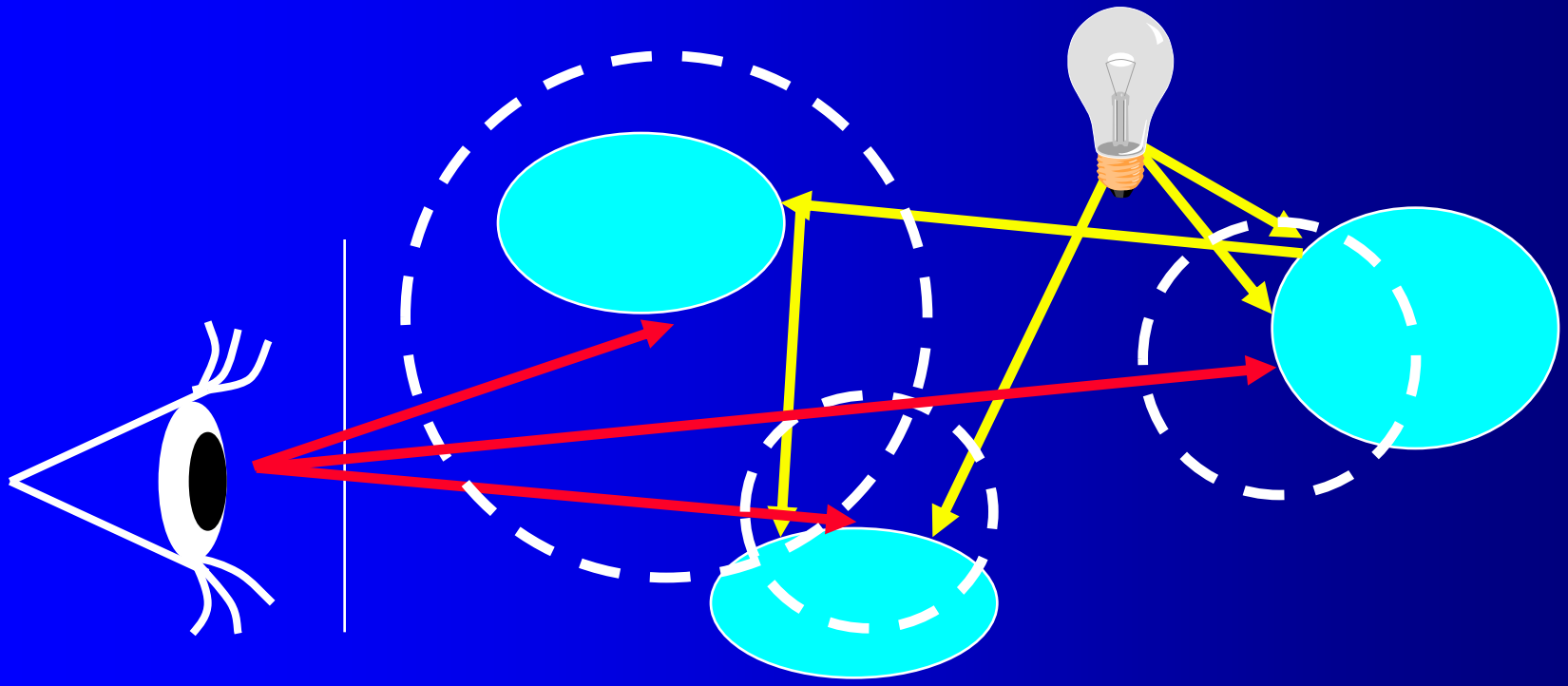
Multiple importance sampling





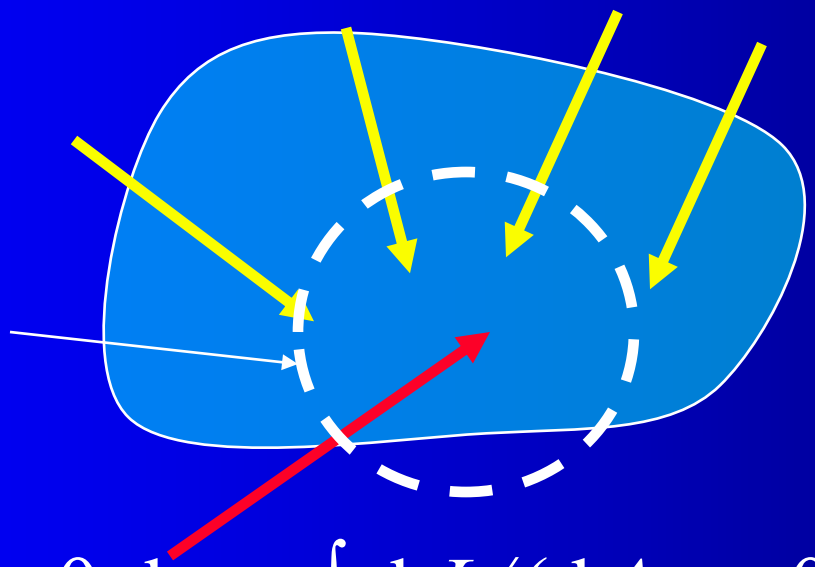


Foton térkép



Foton térkép gyűjtés

Gömb
n foton találat



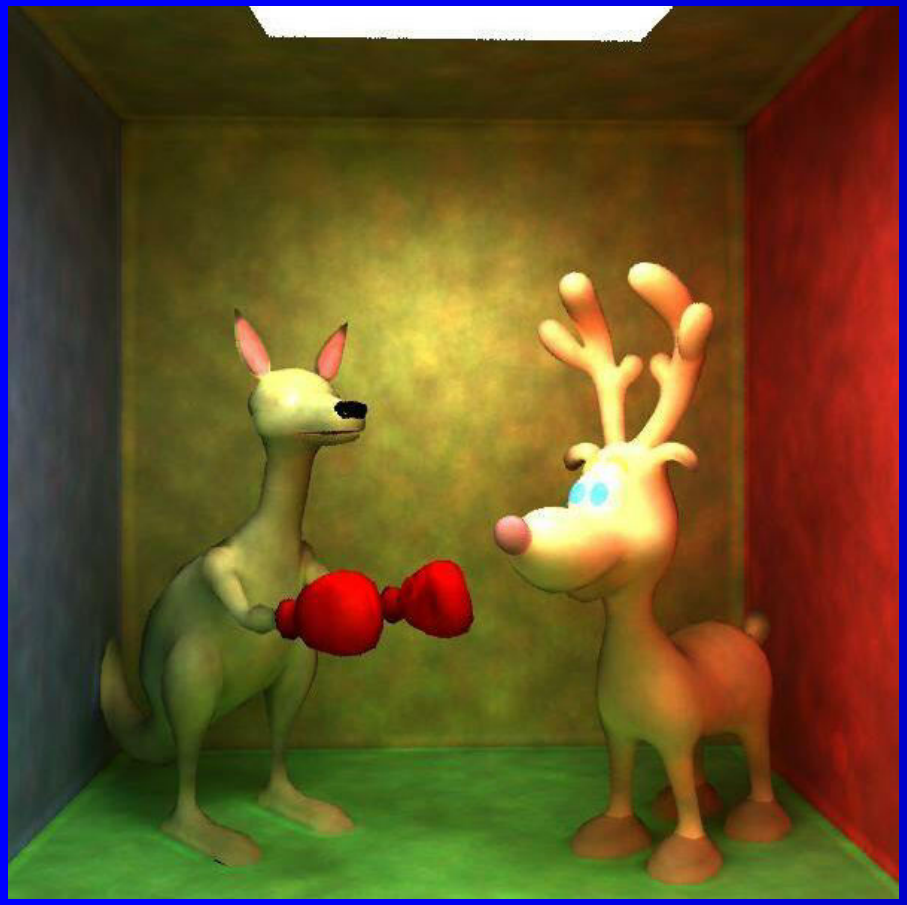
Hit:
- Position
- Direction
- Power
- Normal vector

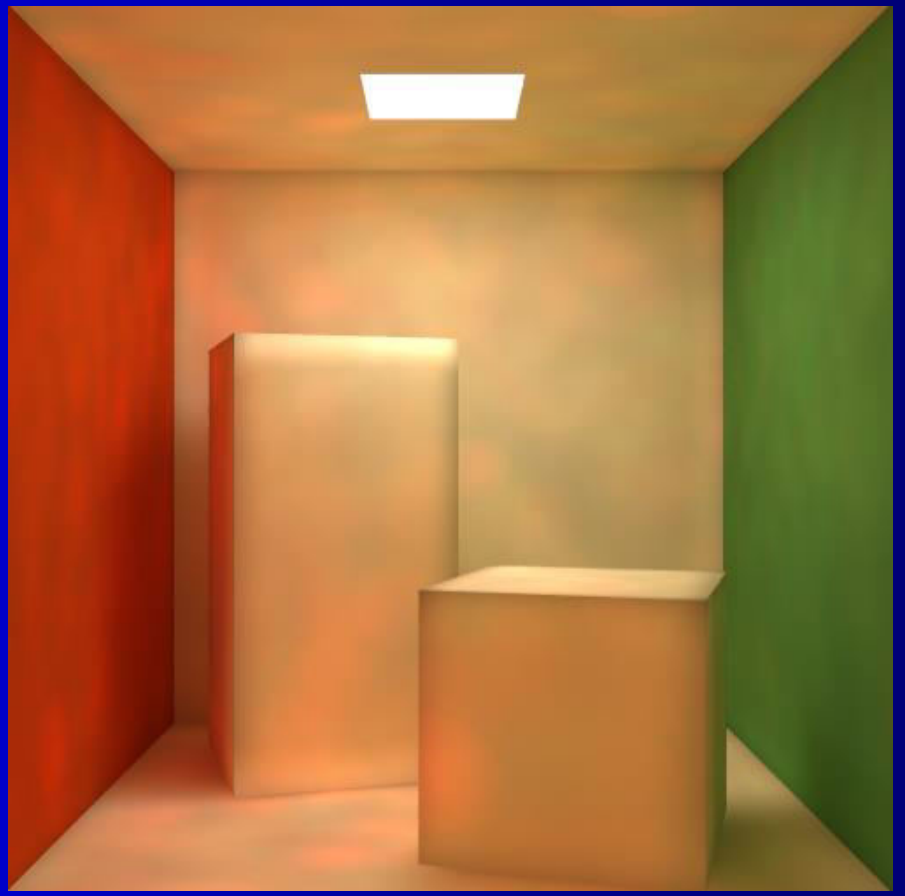
$$L = \int L^{in} f_r \cos\theta \, d\omega = \int d\Phi / (dA \cos\theta \, d\omega) \; f_r \cos\theta \, d\omega$$

$$L = \Sigma \, \Delta\Phi_i / \Delta A \; f_r$$

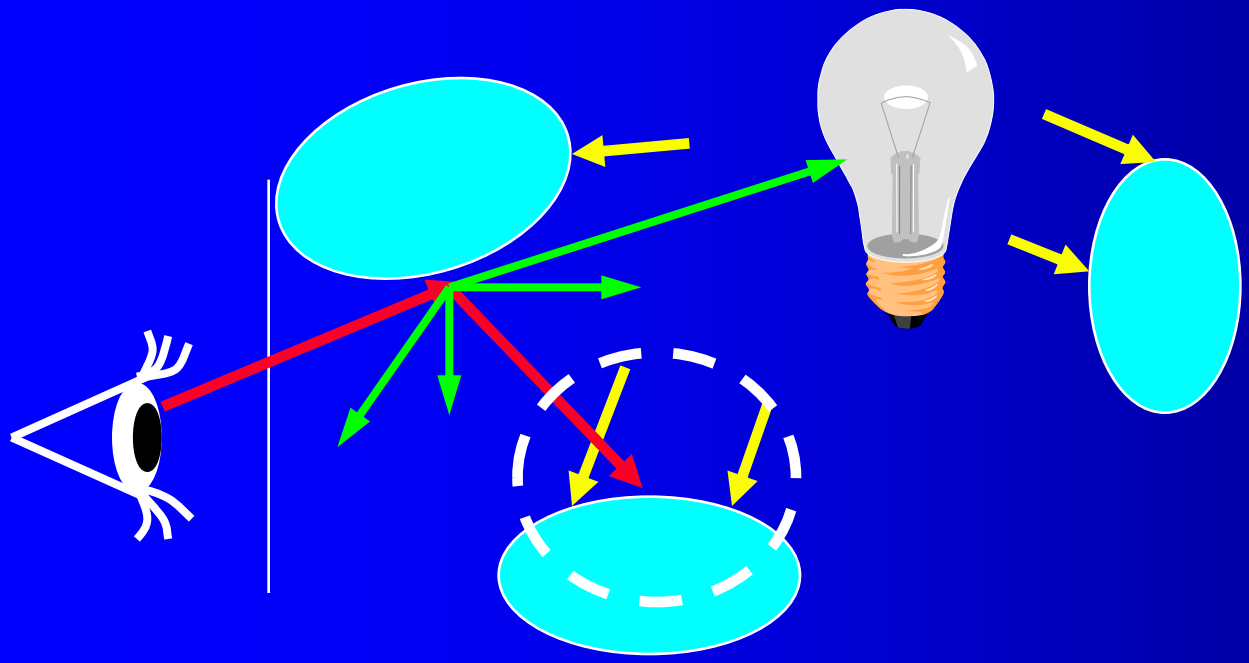
$$\Delta A = r^2 \pi$$

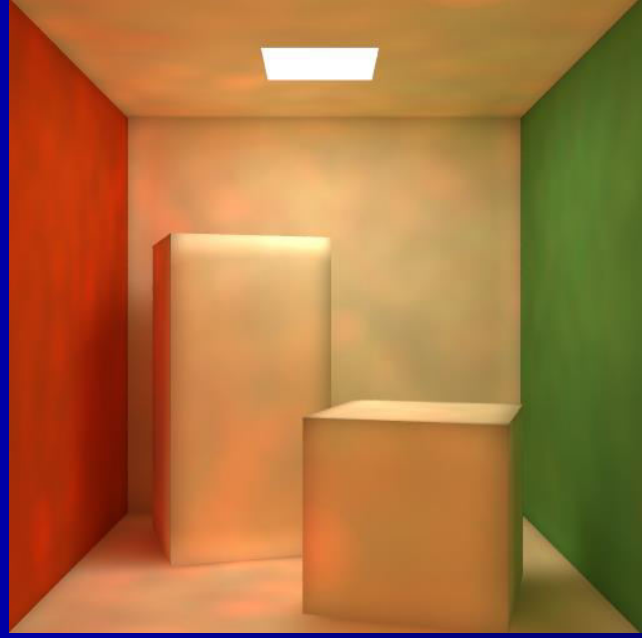
Direkt megjelenítés

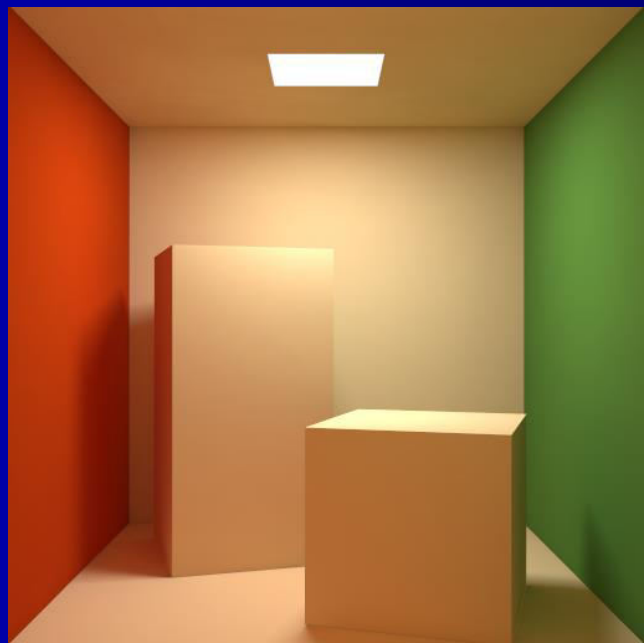




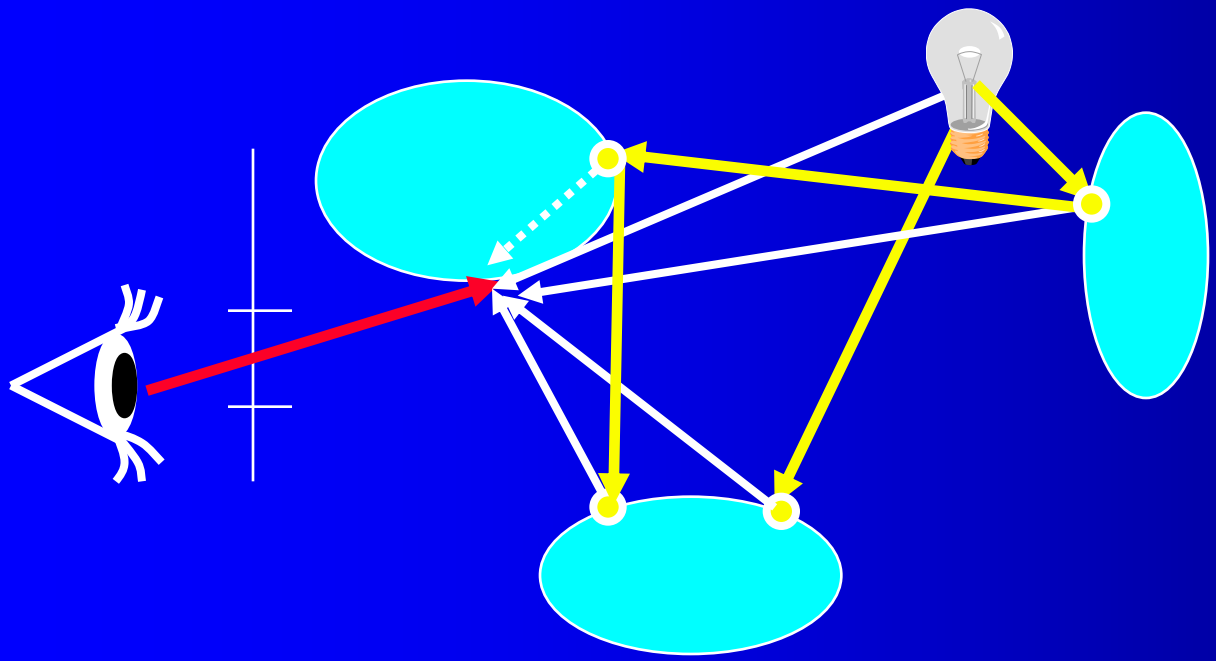
Final gathering

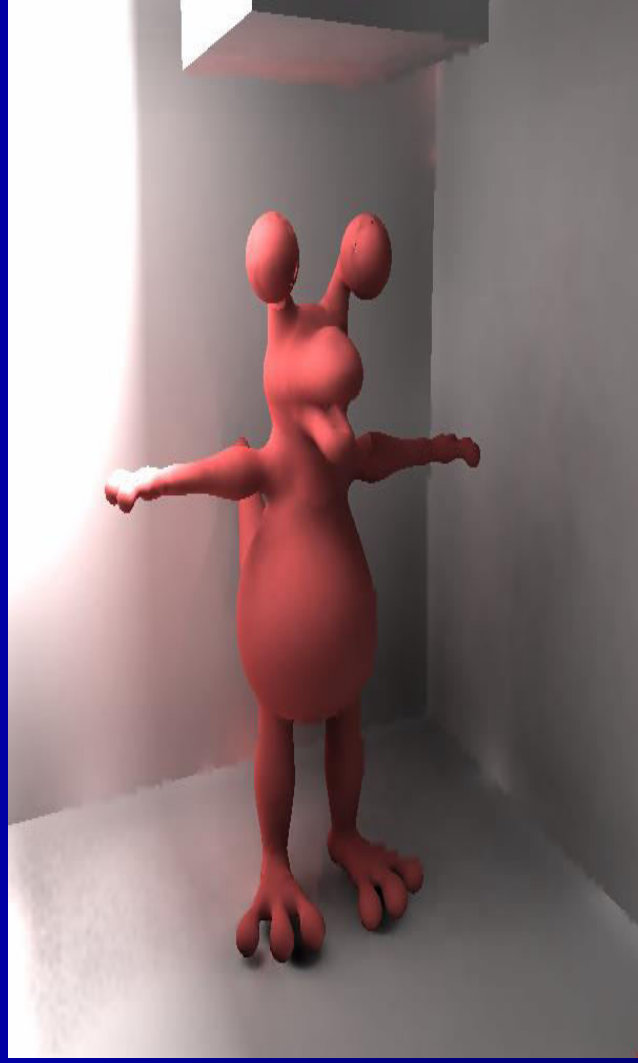






Virtuális fényforrások



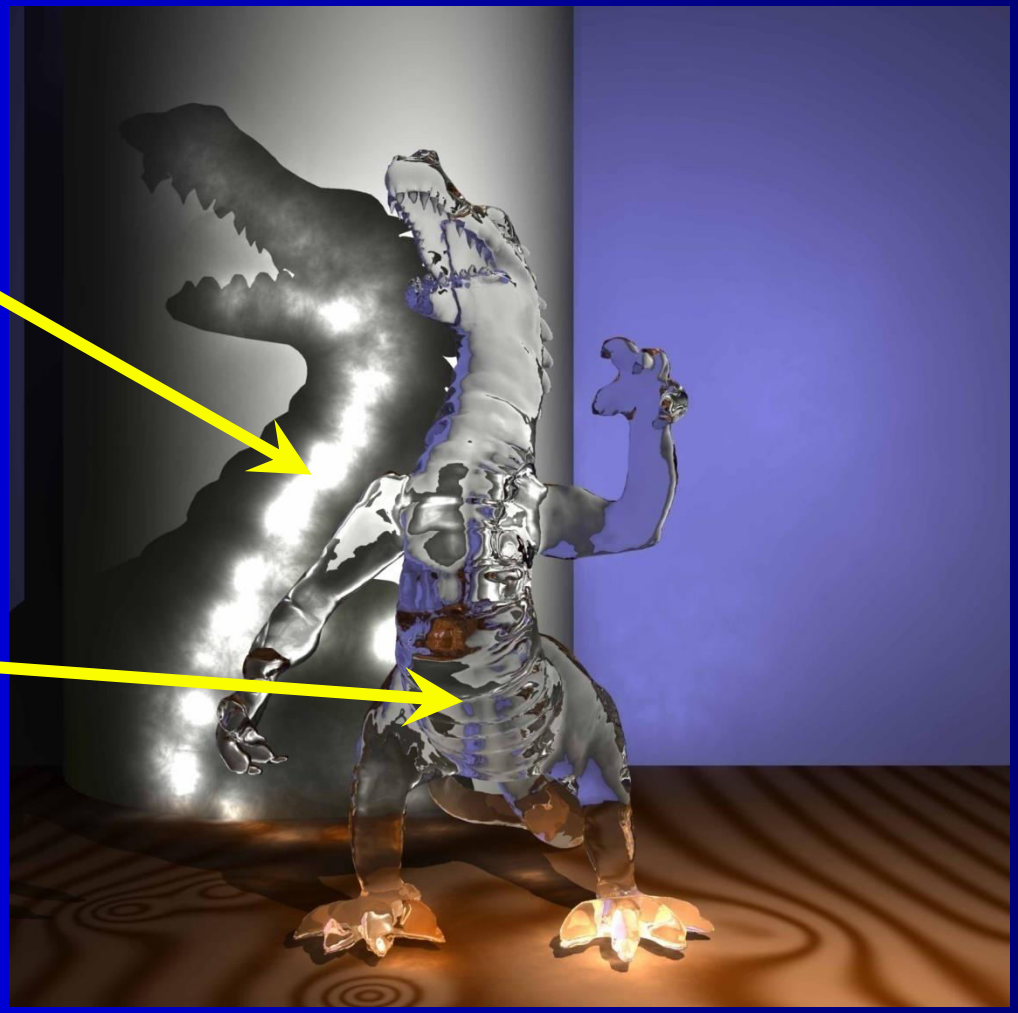


$$L = \Phi_{in} \frac{f_r(y) \cos \theta_y \, f_r(x) \cos \theta_x}{|x - y|^2} v(x,y)$$

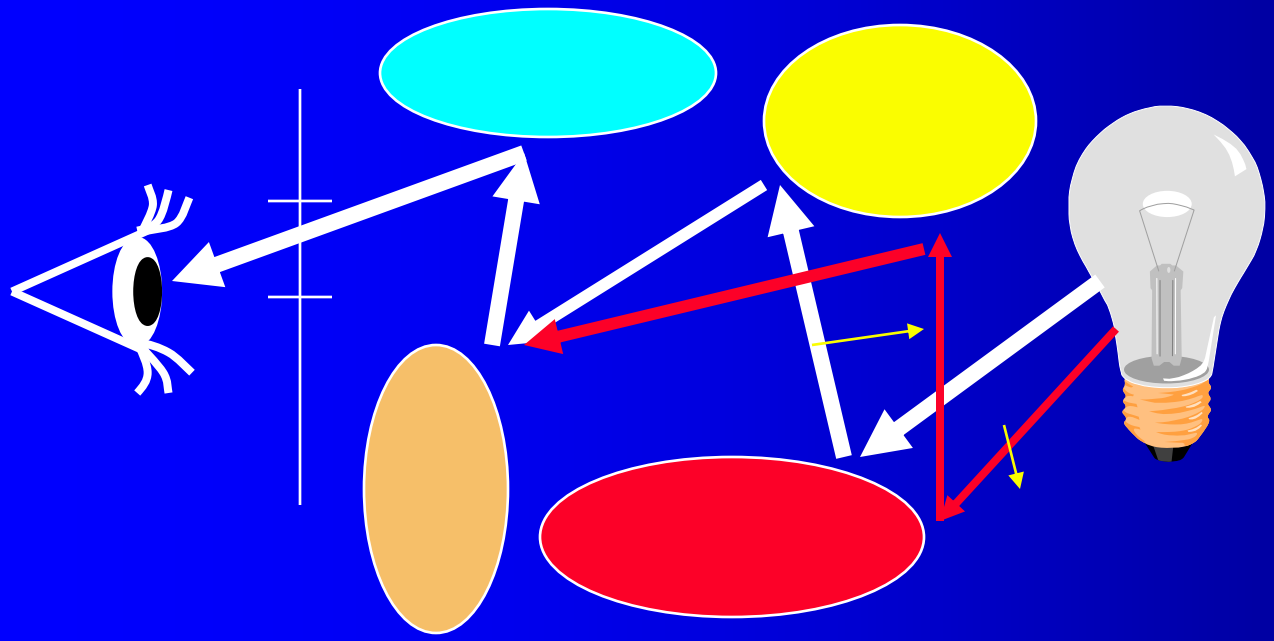
Virtuális fényforrások kiterjesztés

Caustics lövésből

Ideális tükröző, törő felületek: path tracing



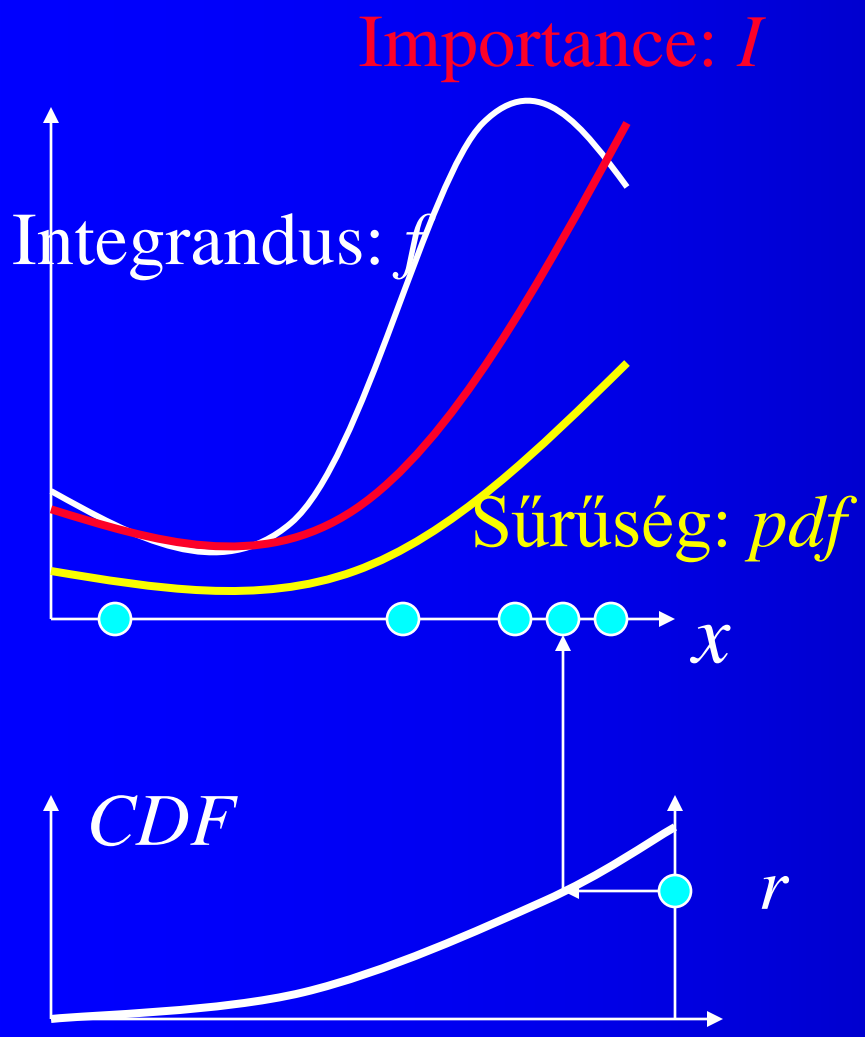
Metropolis light transport





Mutációk: irány, lépésszám váltás

Klasszikus fontosság szerinti mintavétel



1. *I* ami közelíti *f*-*t*
2. *I* normalizálása

$$pdf = I / \int I \, dx$$

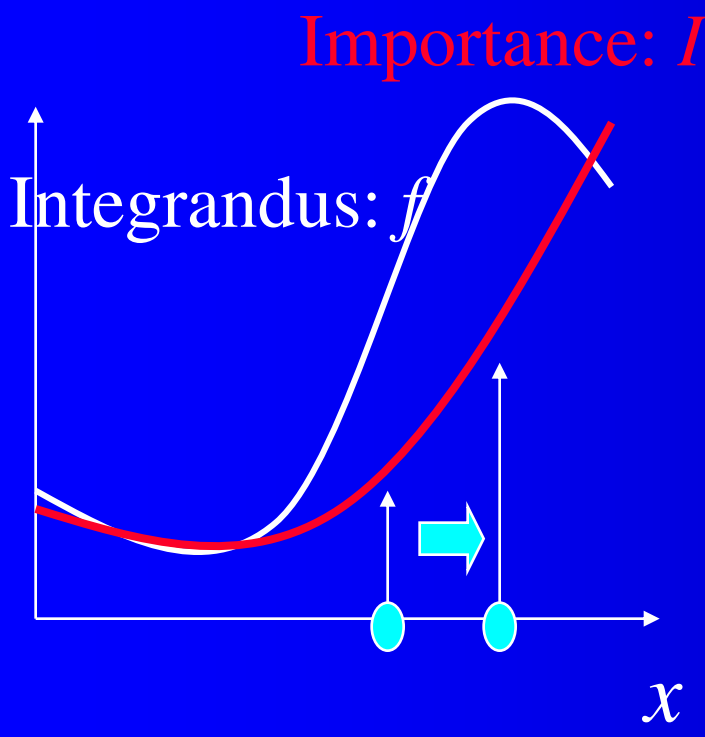
2. Valószínűségeloszlás

$$CDF(y) = \int^{y} pdf \, dx$$

3. Mintavétel:
   Egyenletes eloszlás transzformálása
   r in [0,1]:

$$x = CDF^{-1}(r)$$

Metropolis mintavételezés

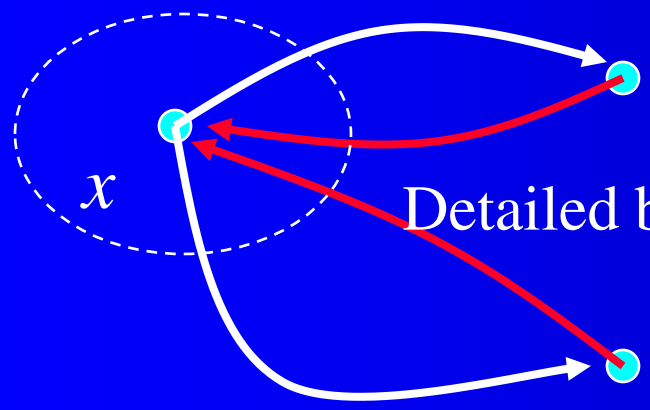


1. I ami hasonló f
2. Normalizáló konstans:
$$b = \int I \, \mathrm{d}x$$
3. Mintavétel:
Mutáció/Elfogadás
Folyamat, amely x-et $I(x)/b$ valószínűségsűrűséggel mintavételezi

Elfogadási valószínűség

- "Bármilyen" mutáció $T(x \to y)$
- Elfogadási valószínűség $a(x \to y)$ úgy, hogy a határeloszlás a fontossággal arányos legyen: $p(x) \propto I(x)$

$$p(x)\, T(x \to y)\, a(x \to y)$$



$$\frac{a(x \to y)}{a(y \to x)} = \frac{I(y)\, T(y \to x)}{I(x)\, T(x \to y)}$$

Detailed balance

Maximális konvergencia:

$$a(x \to y) = \min\left\{ \frac{I(y) \cdot T(y \to x)}{I(x) \cdot T(x \to y)}, 1 \right\}$$

Metropolis algoritmus



FOR $i=1$ TO M DO

Using z_i choose another random, tentative point z_t

$a(z_i \Rightarrow z_t) = (I(z_t)\, T(z_t \Rightarrow z_i)) / (I(z_i)\, T(z_i \Rightarrow z_t))$

// accept with probability $a(z_i \Rightarrow z_t)$

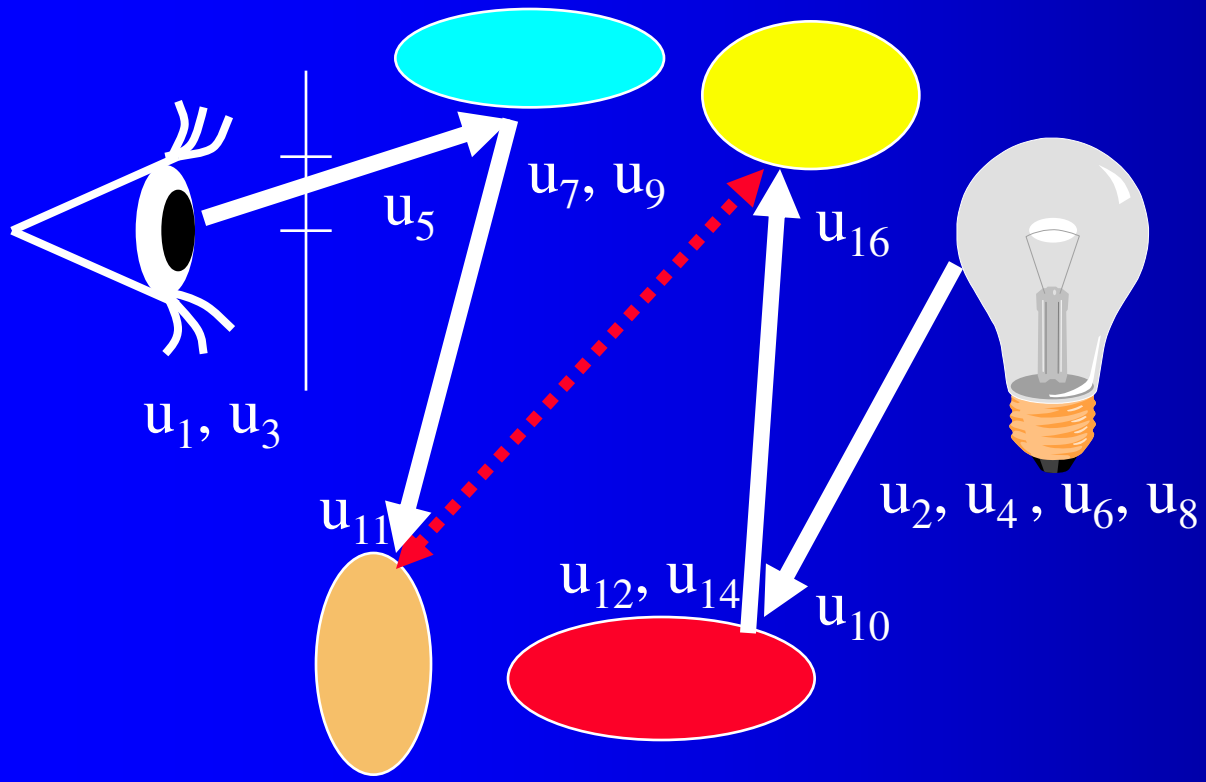
Generate random number r in [0,1]

IF $r < a(z_i \Rightarrow z_t)$ THEN $z_{i+1} = z_t$ ELSE $z_{i+1} = z_i$

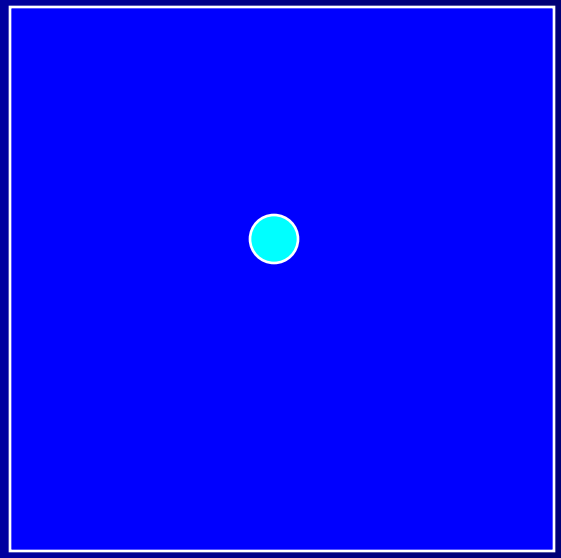
Use z_{i+1} in the integral quadrature

ENDFOR

Mutációk az elsődleges mintavételi térben



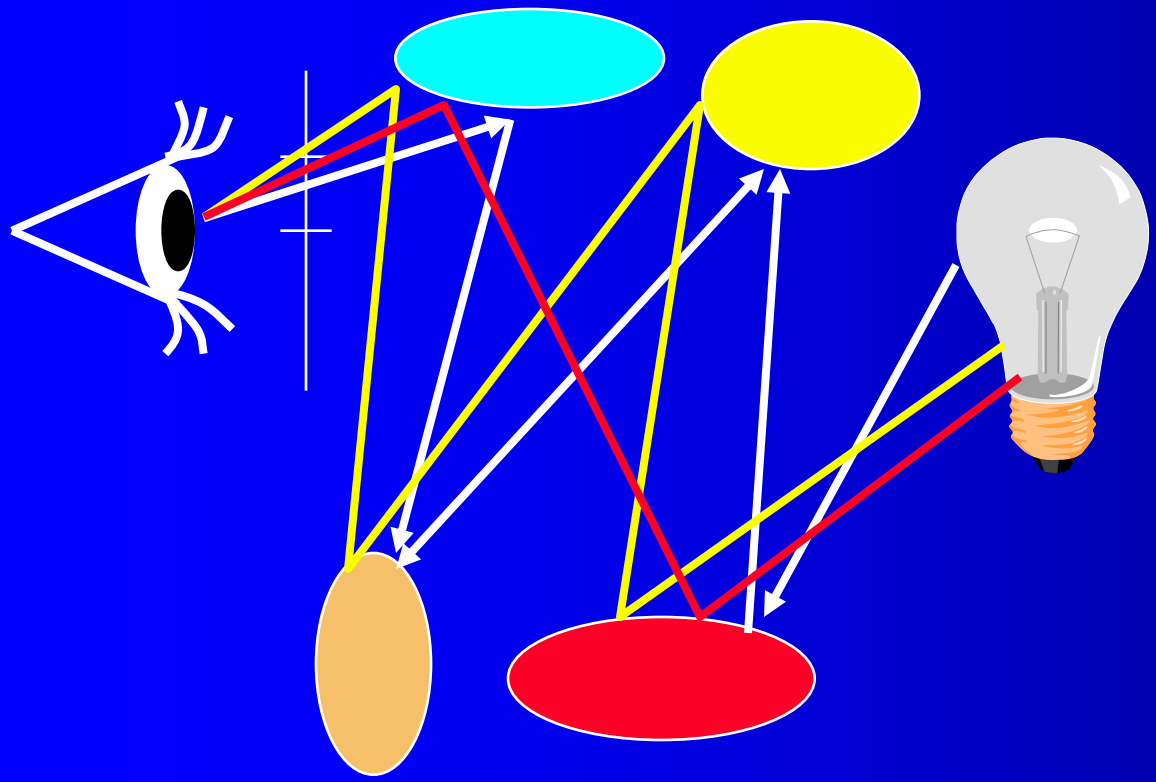
Fényutak



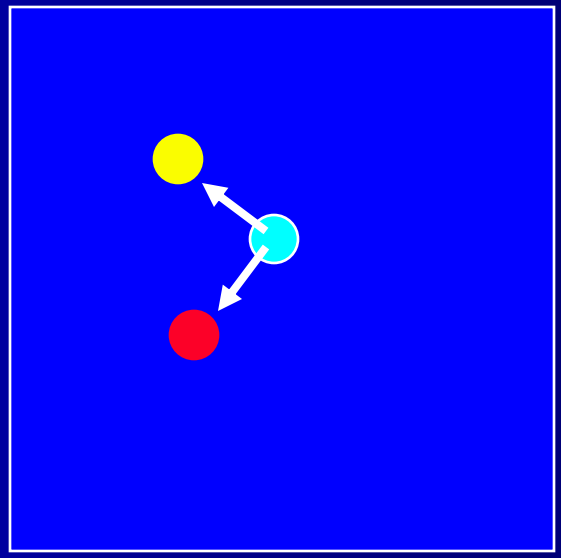
$U=(u_1, \ldots)$

Véletlen számok

Mutációk az elsődleges mintavételi térben



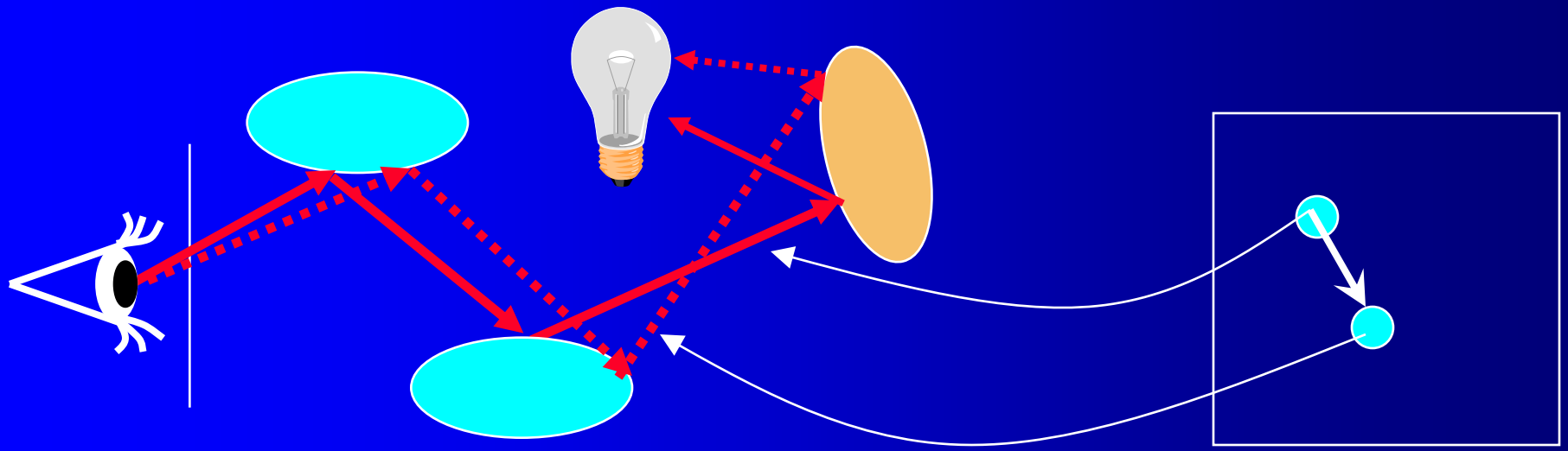
Path space



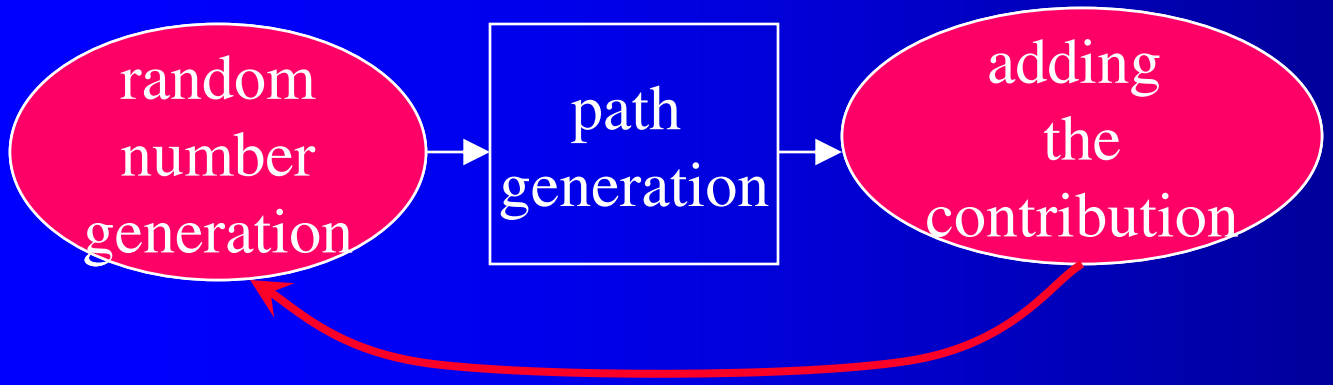
$U=(u_1,\ldots)$

Primary sample space

Mutációk az elsődleges mintavételi térben



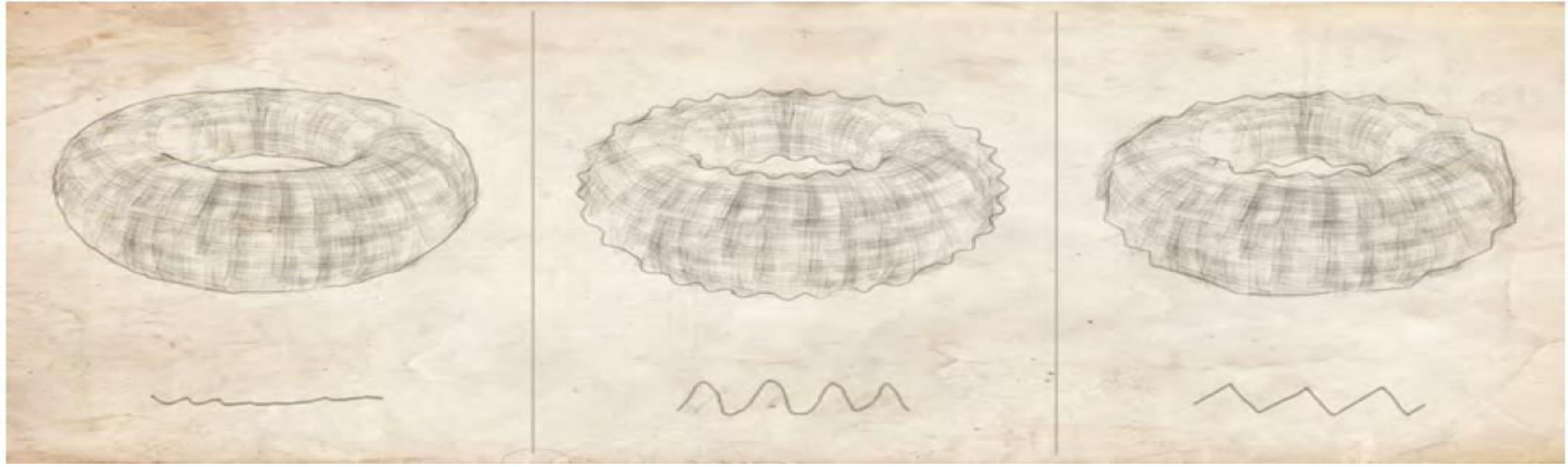
primary sample space of random numbers

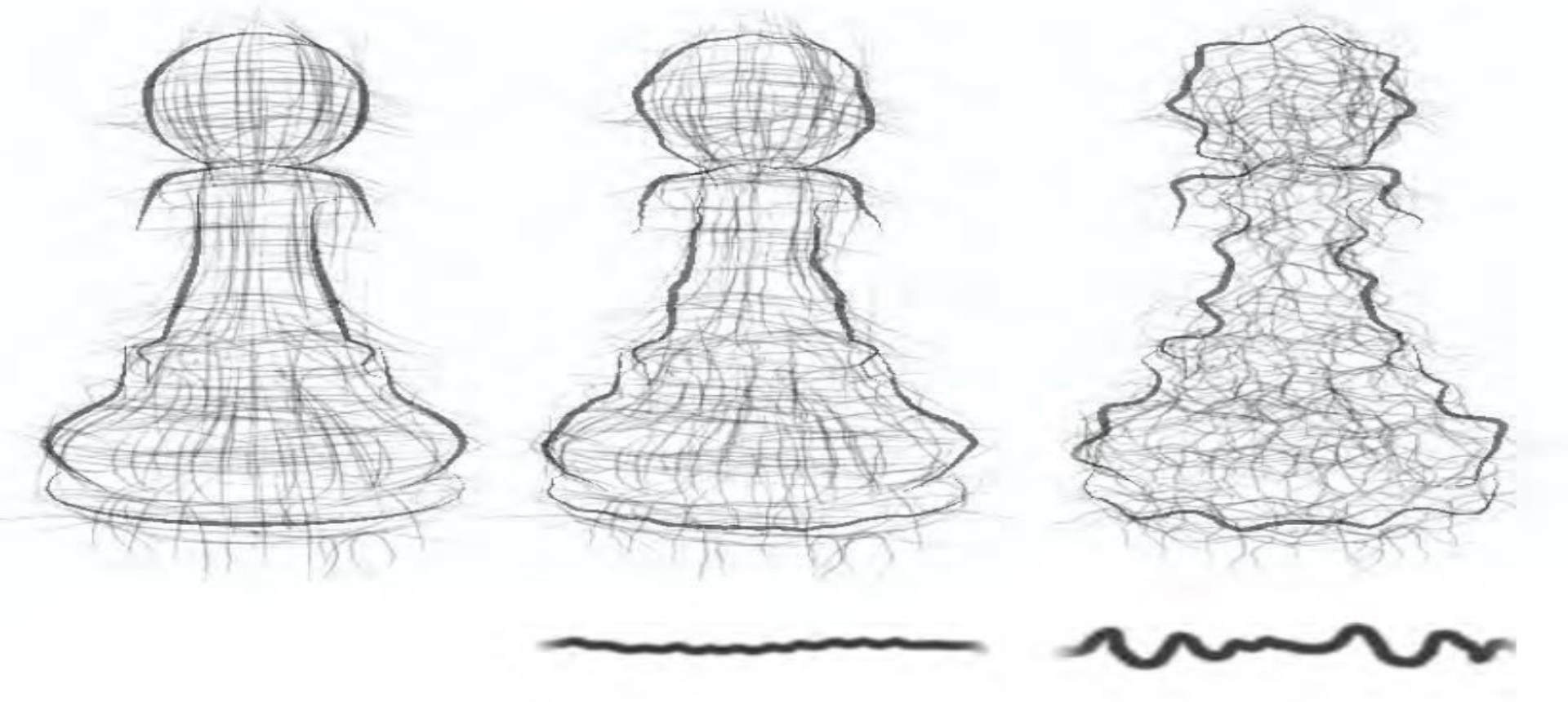





BME-IIT
20


LuxRender
BME-IIT
20





Textúra leképzés

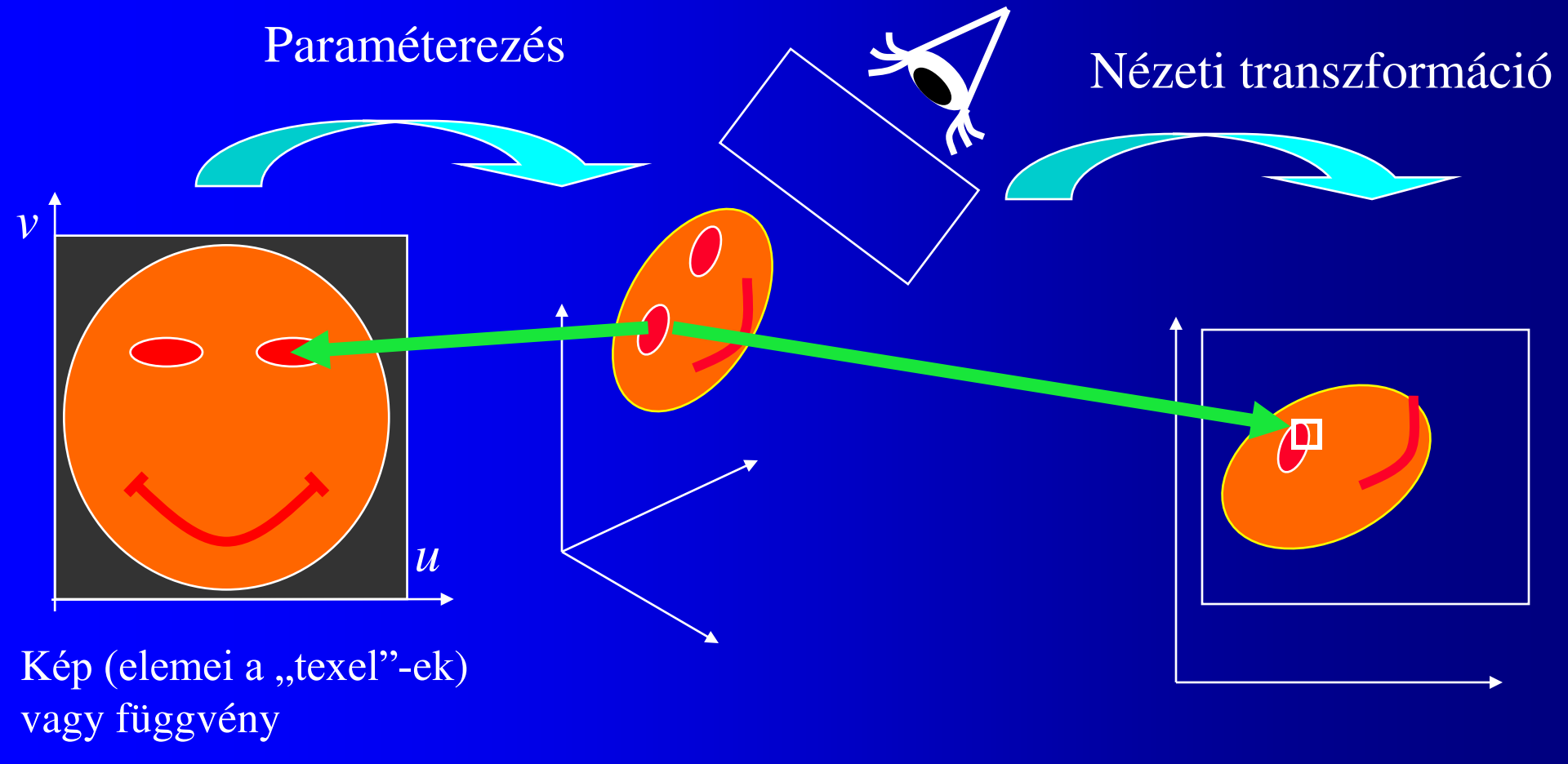
Szirmay-Kalos László

Textúra leképzés:
anyagjellemzők változnak a felületen



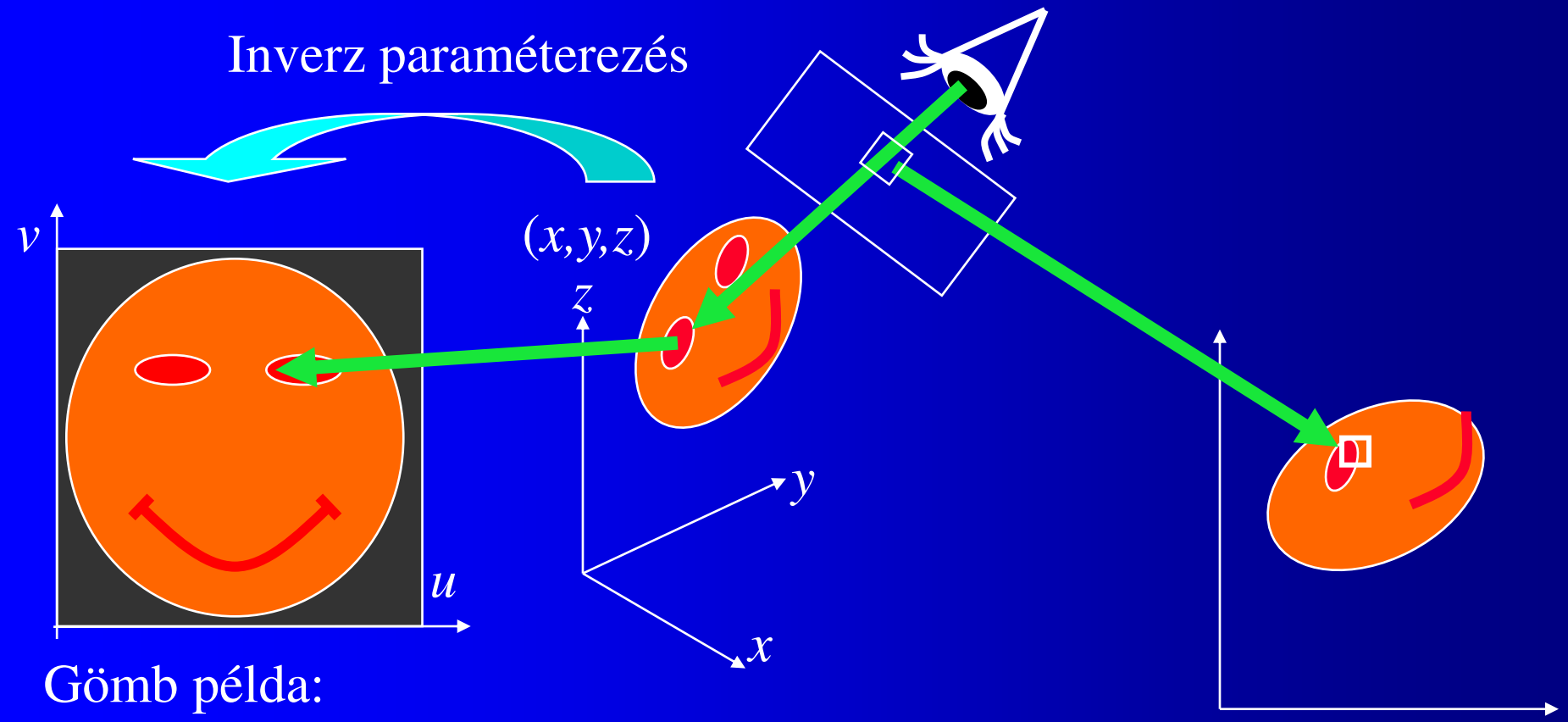


(2D) Textúra leképzés



Kép (elemei a „texel”-ek)
vagy függvény

Textúra leképzés sugárkövetésnél



Gömb példa:

$x = x0 + r \cos 2\pi u \sin \pi v$

$y = y0 + r \sin 2\pi u \sin \pi v$

$z = z0 + r \cos \pi v \qquad u,v \in [0,1]$

$u = 1/2\pi \, (\text{atan2}((y-y0), (x-x0)) + \pi)$

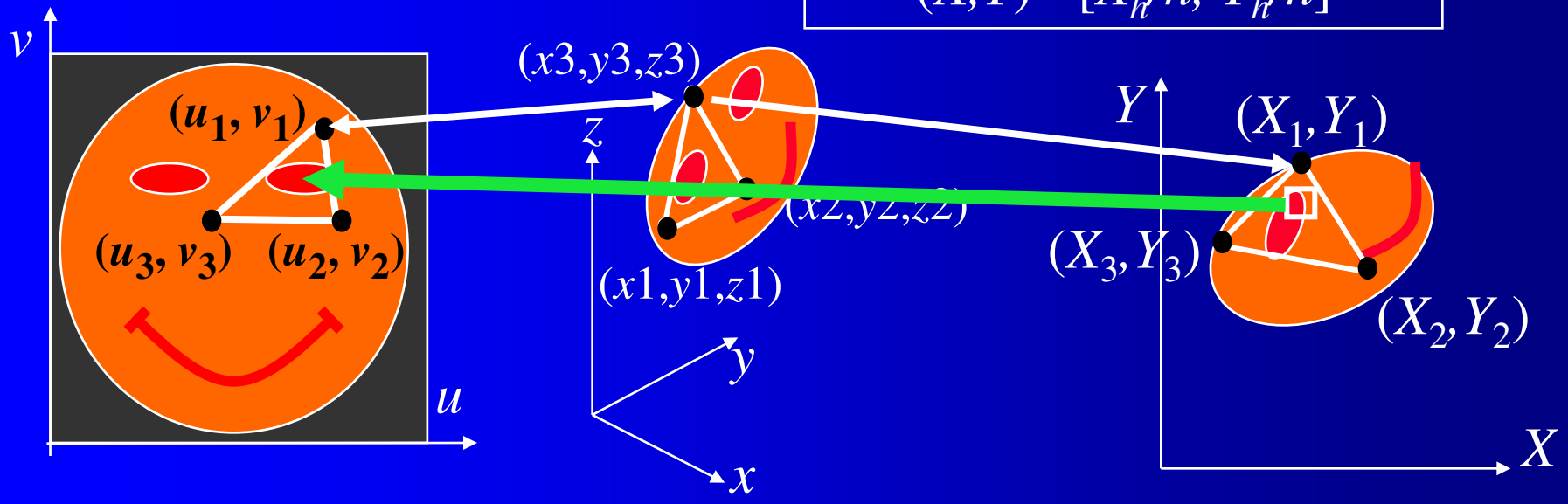
$v = 1/\pi \, \text{acos}((z-z0)/r)$

Inkrementális képszintézis

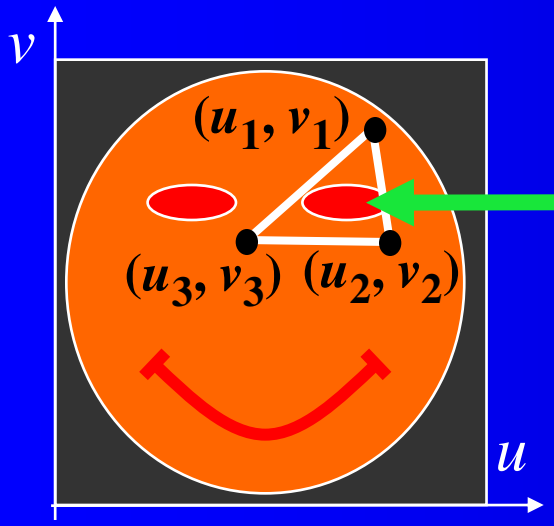
Paraméterezés

Modell-világ-képernyő transzf.

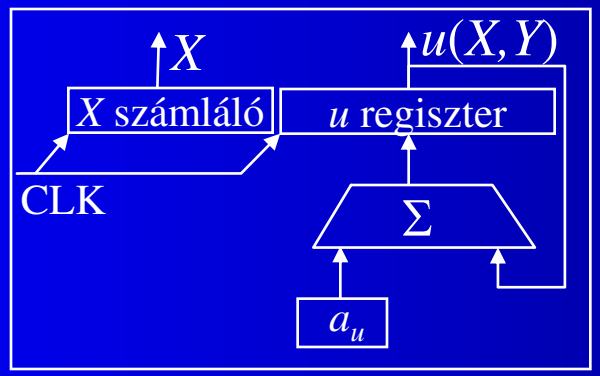
$$[X_h,\ Y_h,\ Z_h,\ h] = [x,y,z,1]\mathbf{T}_C$$
$$(X,Y) = [X_h/h,\ Y_h/h]$$

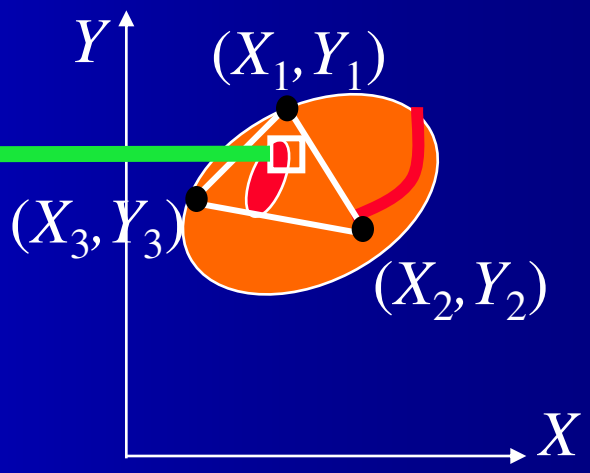


Lineáris interpoláció



$$u = a_u X + b_u Y + c_u$$
$$v = a_v X + b_v Y + c_v$$

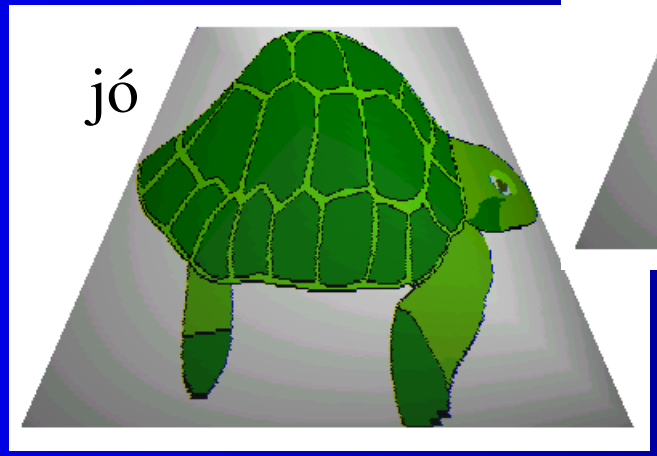


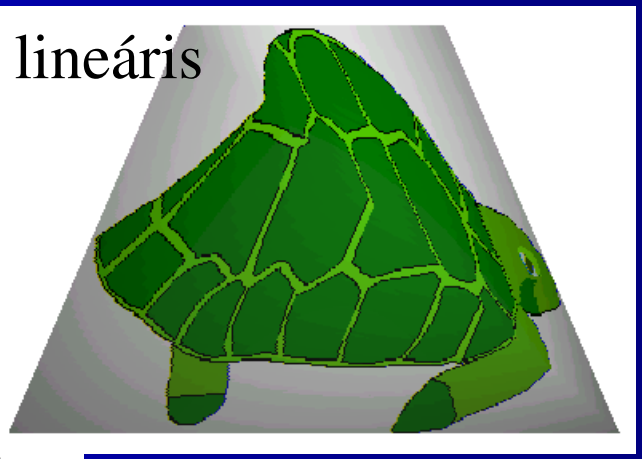


Interpolációs feltétel:

$$u_1 = a_u X_1 + b_u Y_1 + c_u$$
$$u_2 = a_u X_2 + b_u Y_2 + c_u$$
$$u_3 = a_u X_3 + b_u Y_3 + c_u$$

$$v_1 = a_v X_1 + b_v Y_1 + c_v$$
$$v_2 = a_v X_2 + b_v Y_2 + c_v$$
$$v_3 = a_v X_3 + b_v Y_3 + c_v$$





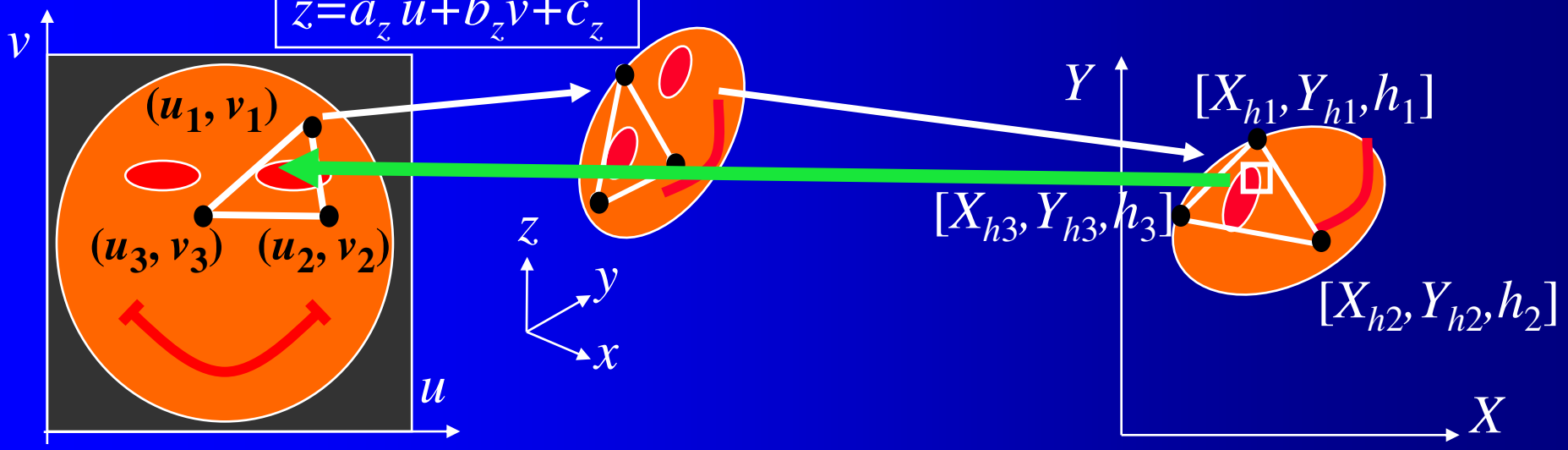
Perspektíva helyes textúrázás

Paraméterezés

$$x = a_x u + b_x v + c_x$$
$$y = a_y u + b_y v + c_y$$
$$z = a_z u + b_z v + c_z$$

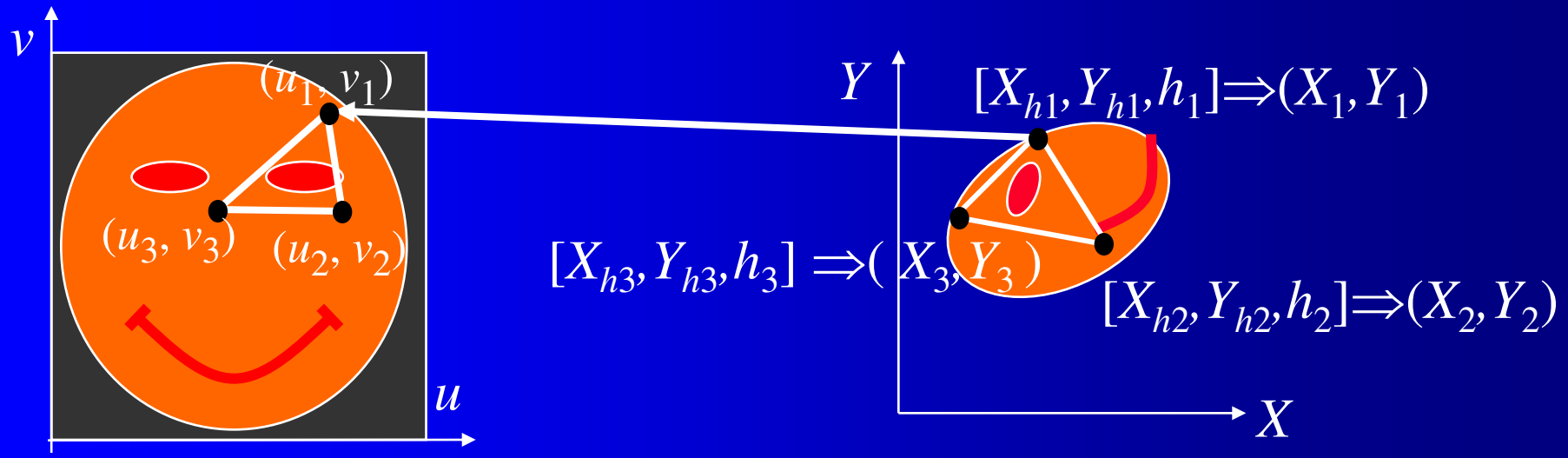
Modell-világ-képernyő transzf.

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



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

T^{-1} mátrixelemek



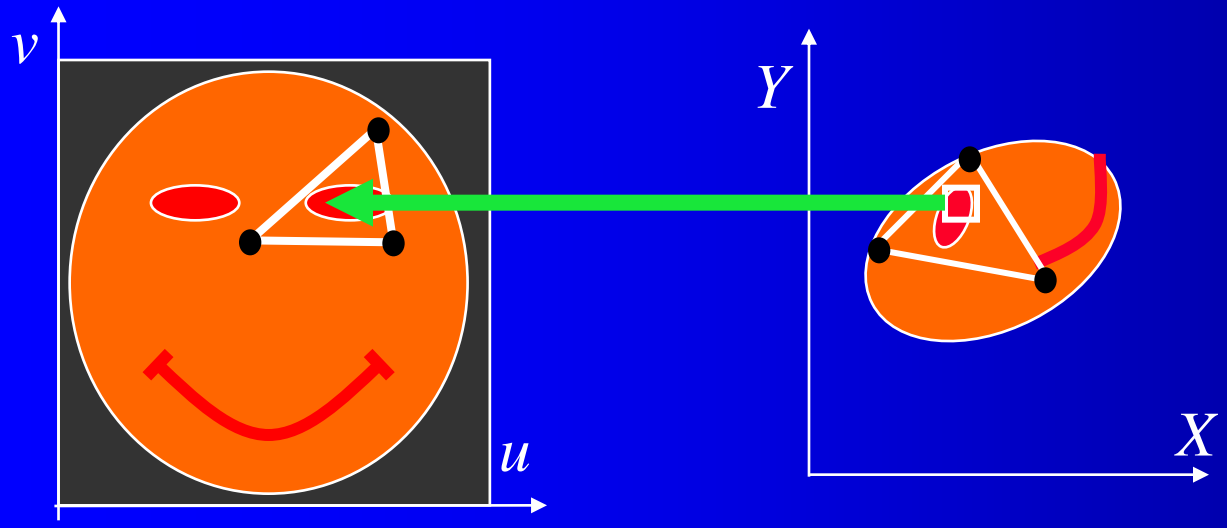
$$u_1/h_1 = a_u X_1 + b_u Y_1 + c_u$$
$$u_2/h_2 = a_u X_2 + b_u Y_2 + c_u$$
$$u_3/h_3 = a_u X_3 + b_u Y_3 + c_u$$

$$v_1/h_1 = a_v X_1 + b_v Y_1 + c_v$$
$$v_1/h_1 = a_v X_2 + b_v Y_2 + c_v$$
$$v_1/h_1 = a_v X_3 + b_v Y_3 + c_v$$

$$1/h_1 = a_h X_1 + b_h Y_1 + c_w$$
$$1/h_2 = a_h X_2 + b_h Y_2 + c_w$$
$$1/h_3 = a_h X_3 + b_h Y_3 + c_w$$

a_u, b_u, c_u $\qquad$ a_v, b_v, c_v $\qquad$ a_h, b_h, c_h

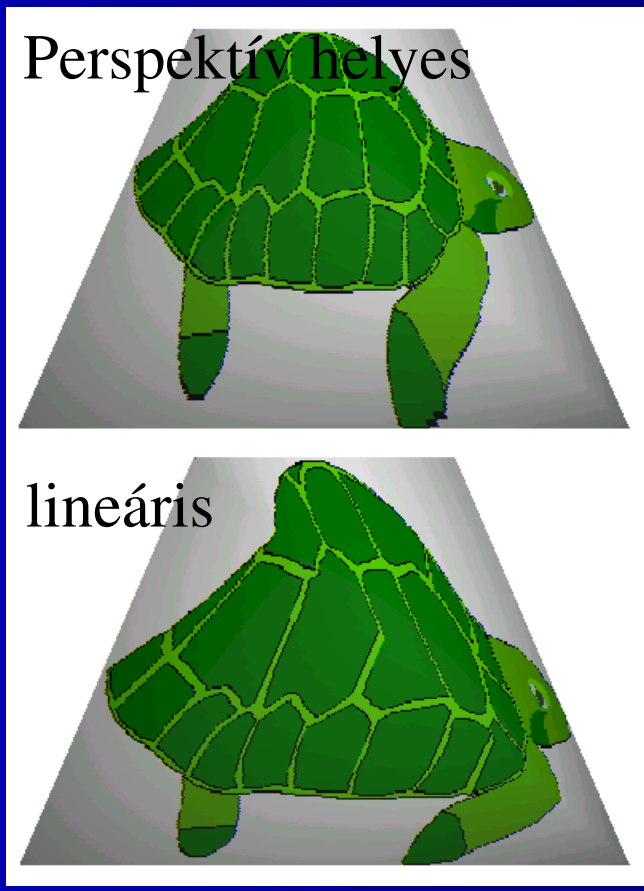
Perspektíva helyes textúrázás



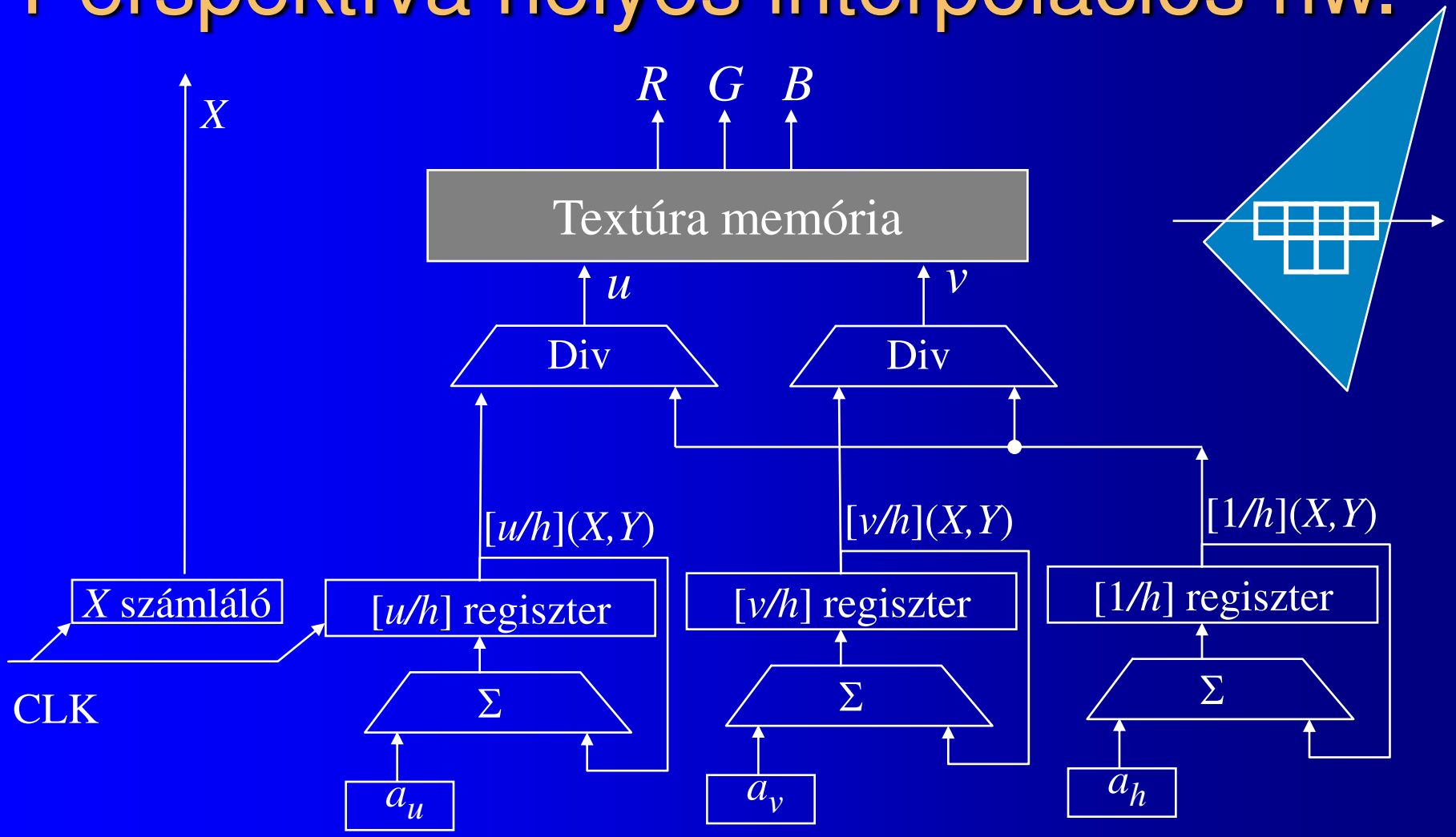
$$u/h = a_u X + b_u Y + c_u$$
$$v/h = a_v X + b_v Y + c_v$$
$$1/h = a_h X + b_h Y + c_h$$

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

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

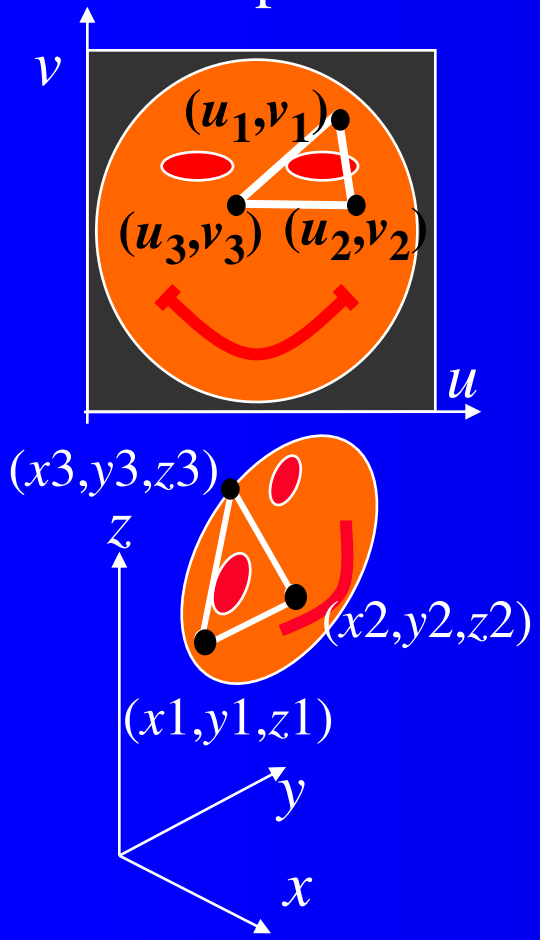


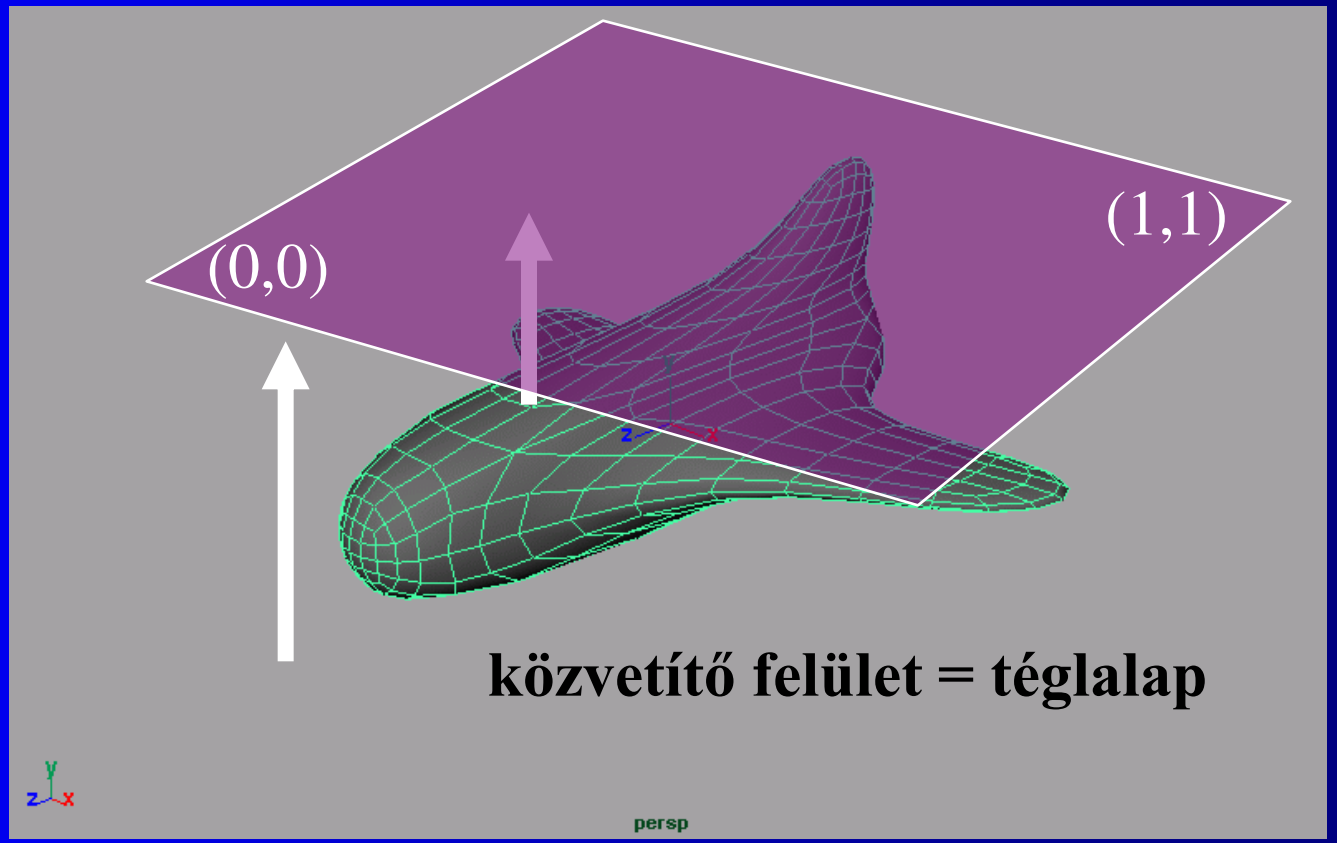
Perspektíva helyes interpolációs hw.



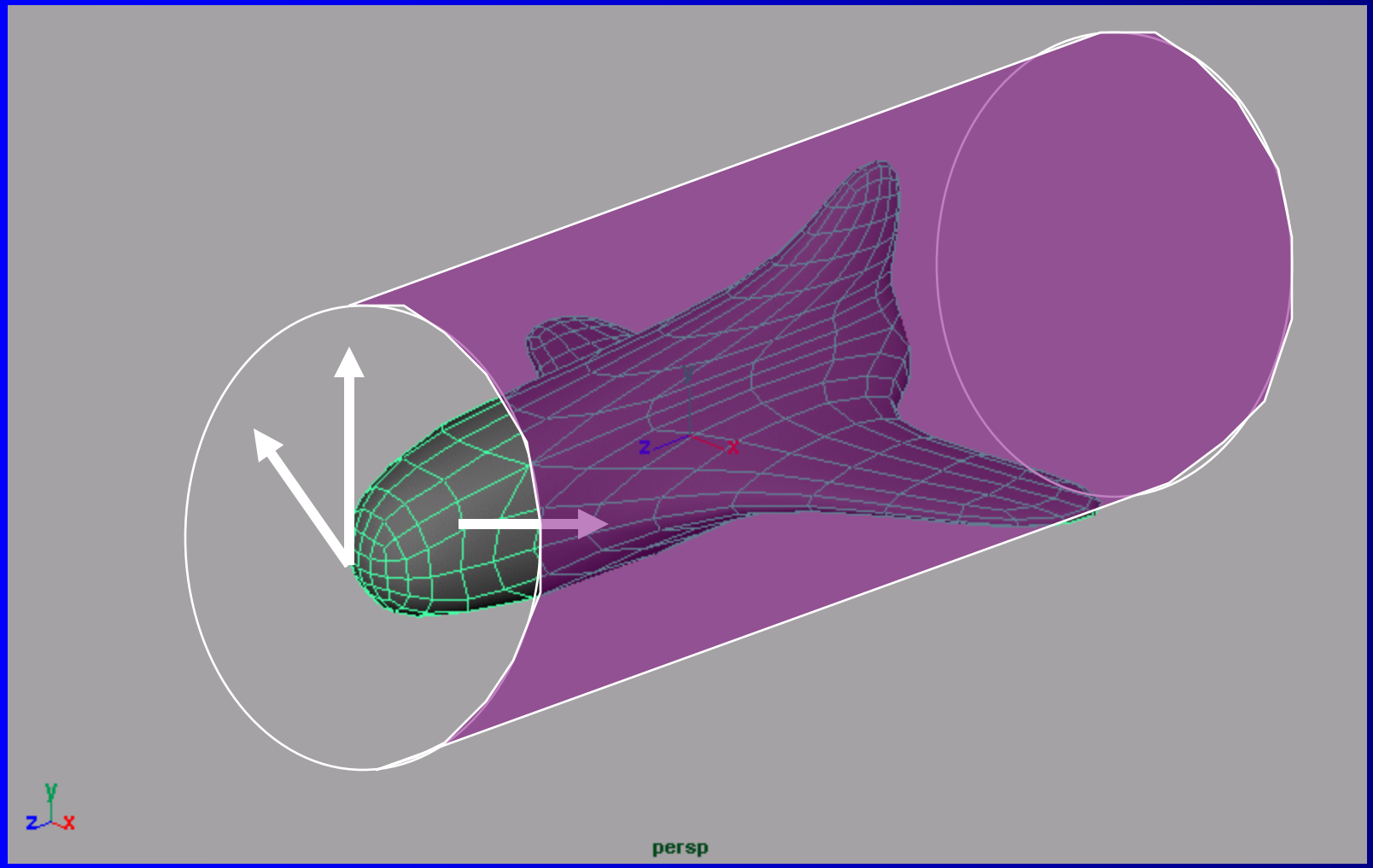
Textúra függvény definíciója

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

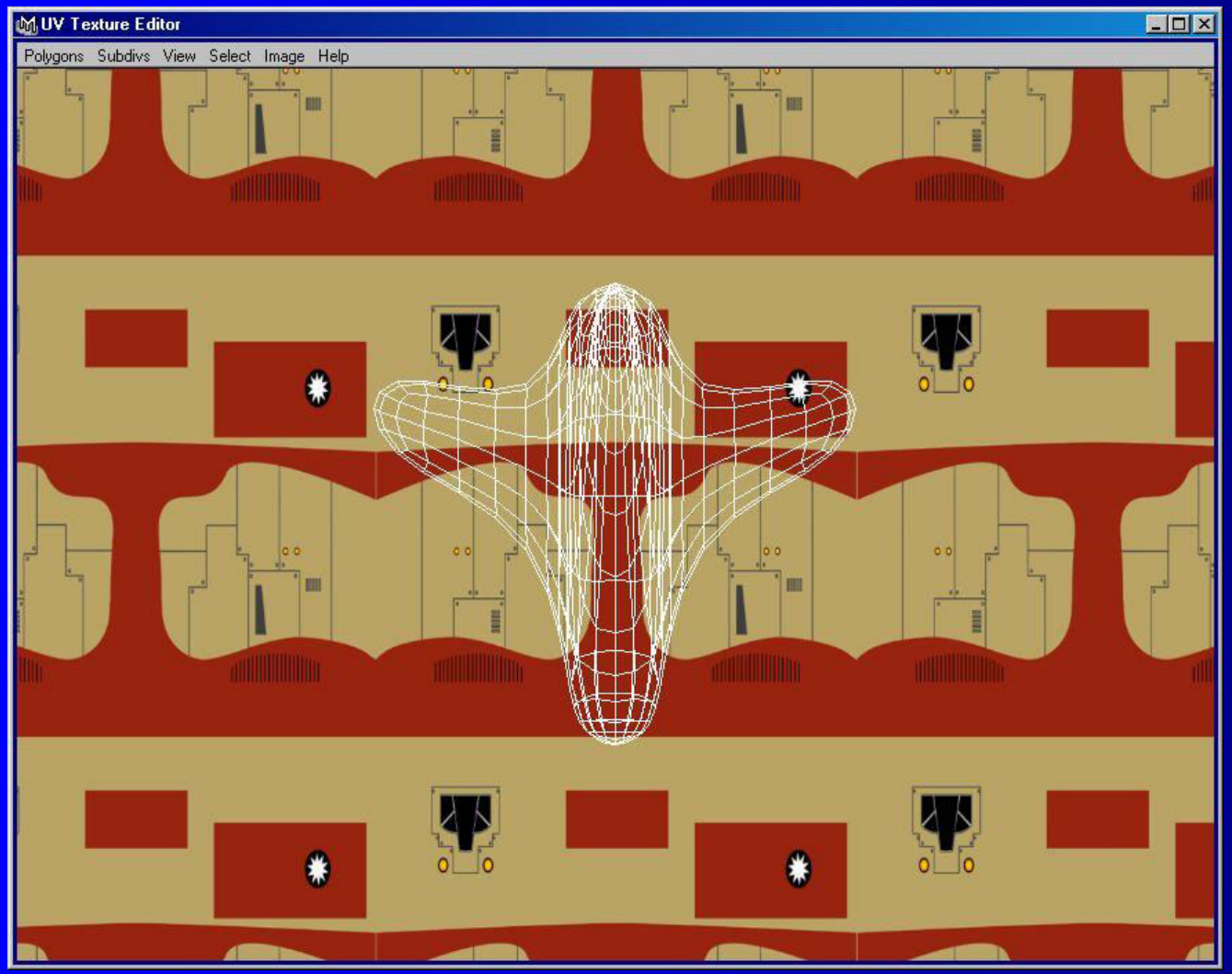




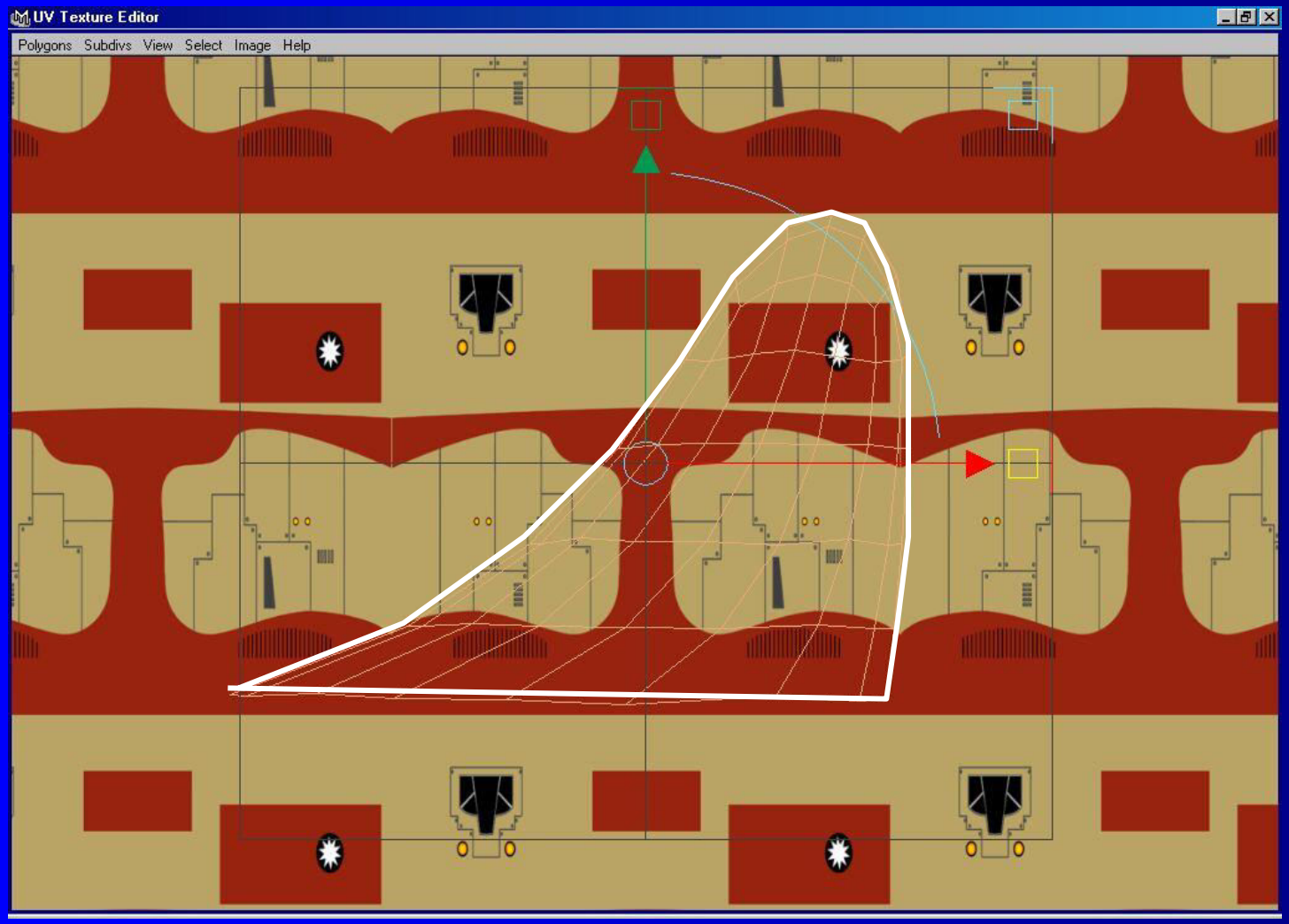
Henger (gömb) közvetítő felület



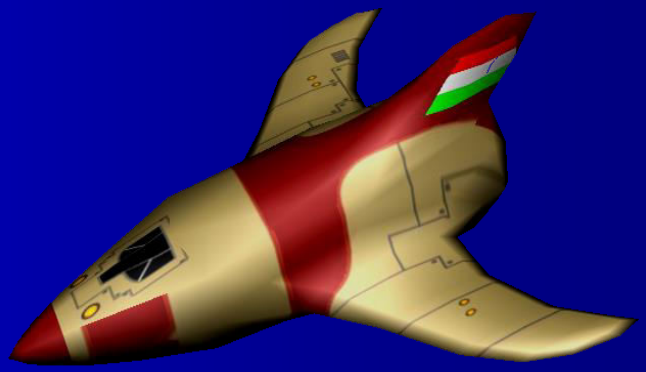
Téglalap közvetítő felület: textúratér nézet

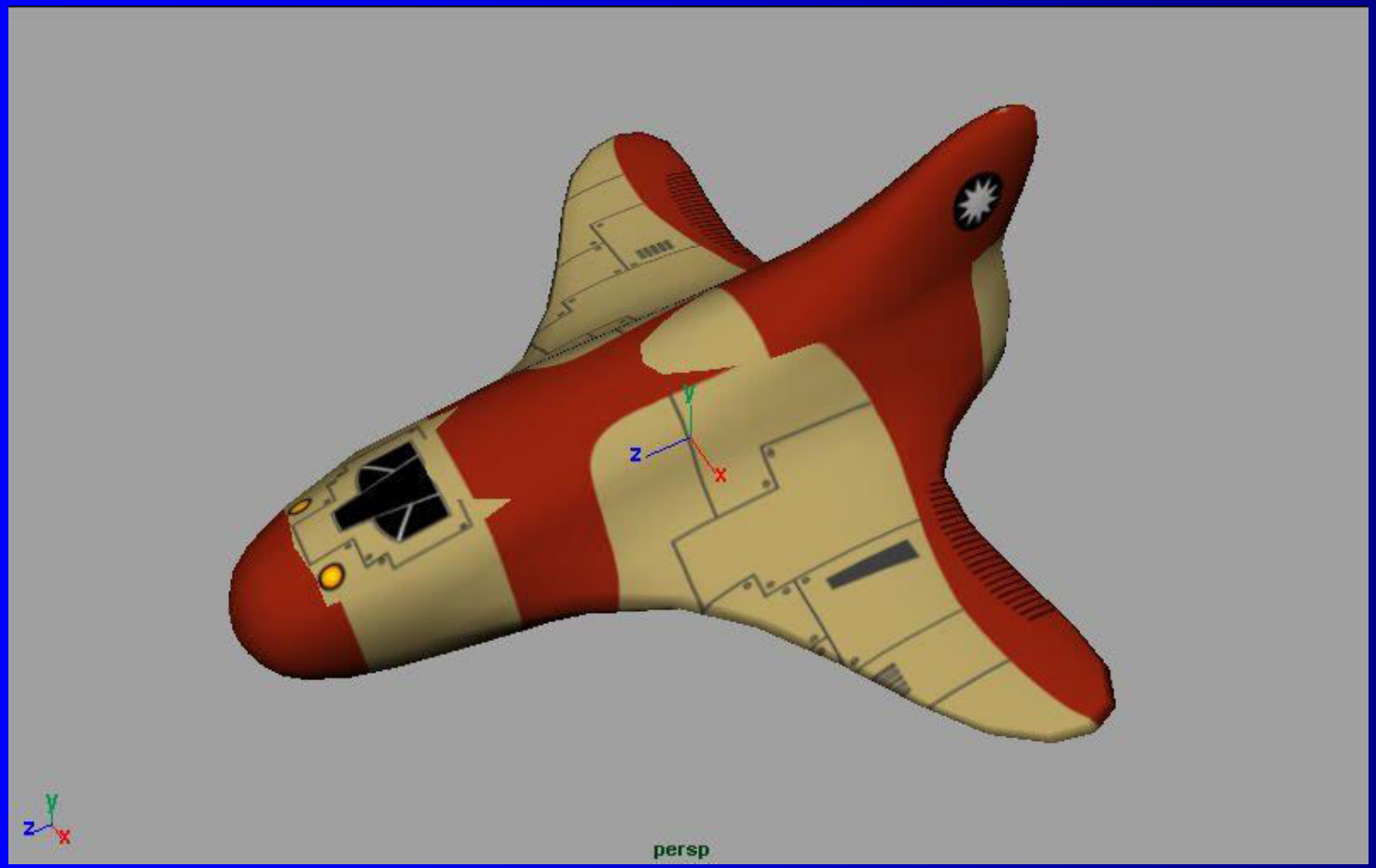


Téglalap közvetítő felület: textúratér nézet

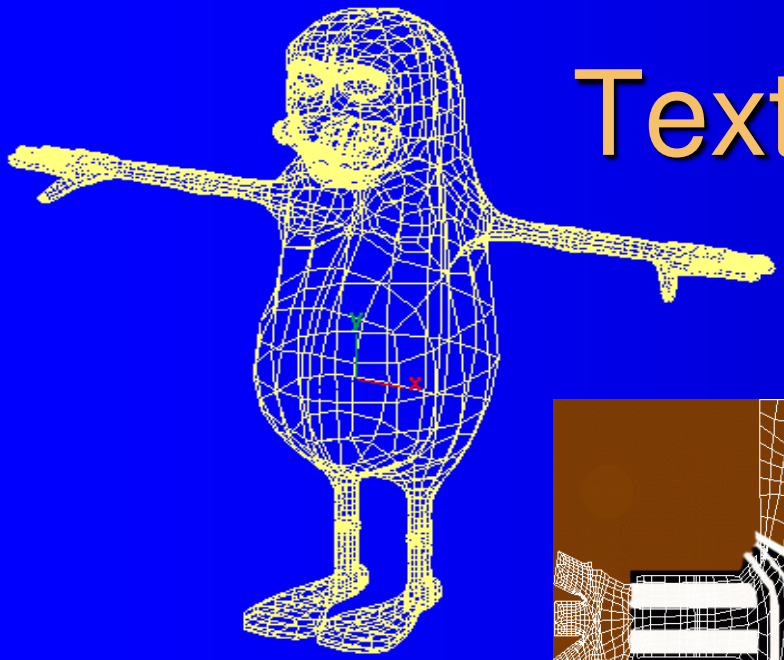


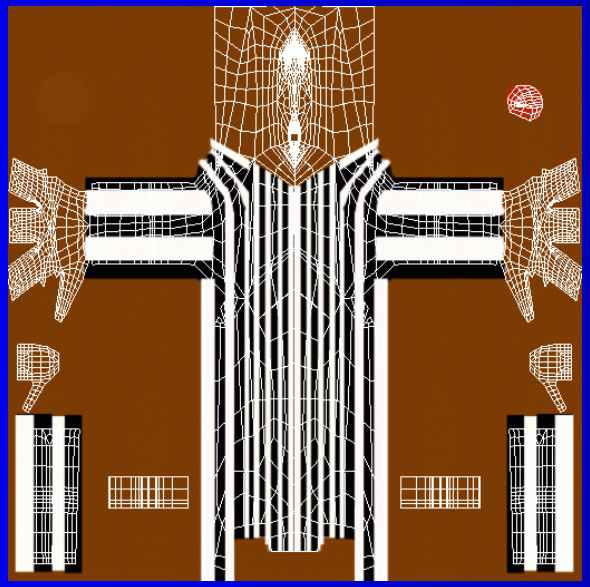
Textúrázott űrhajó

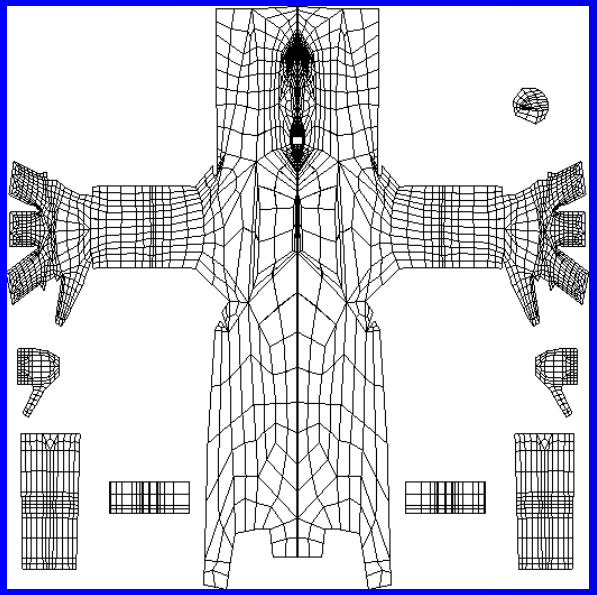




Textúrázás = kiterítés



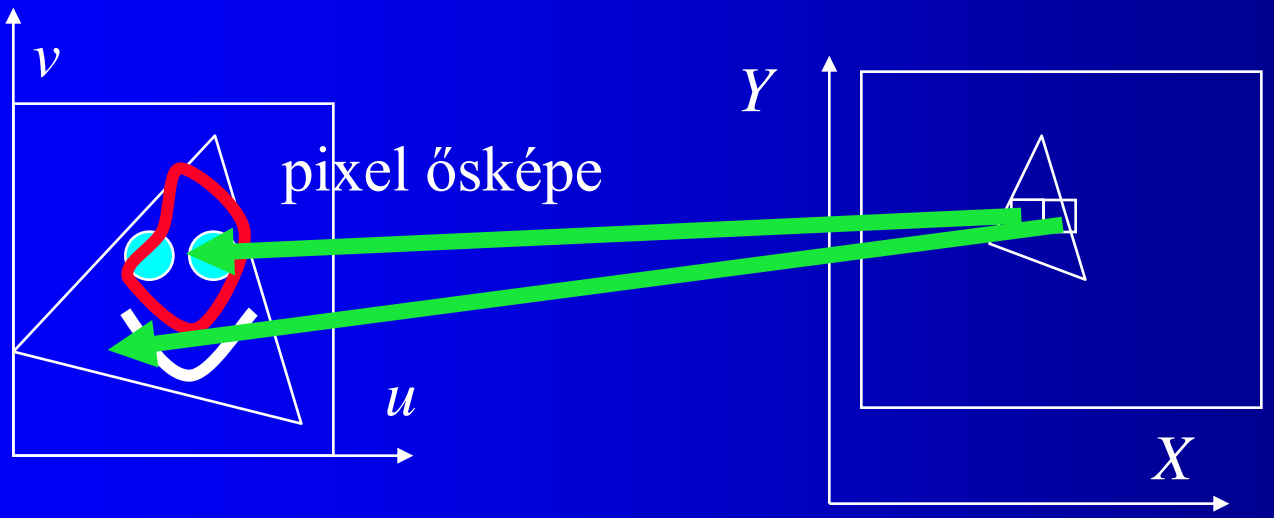




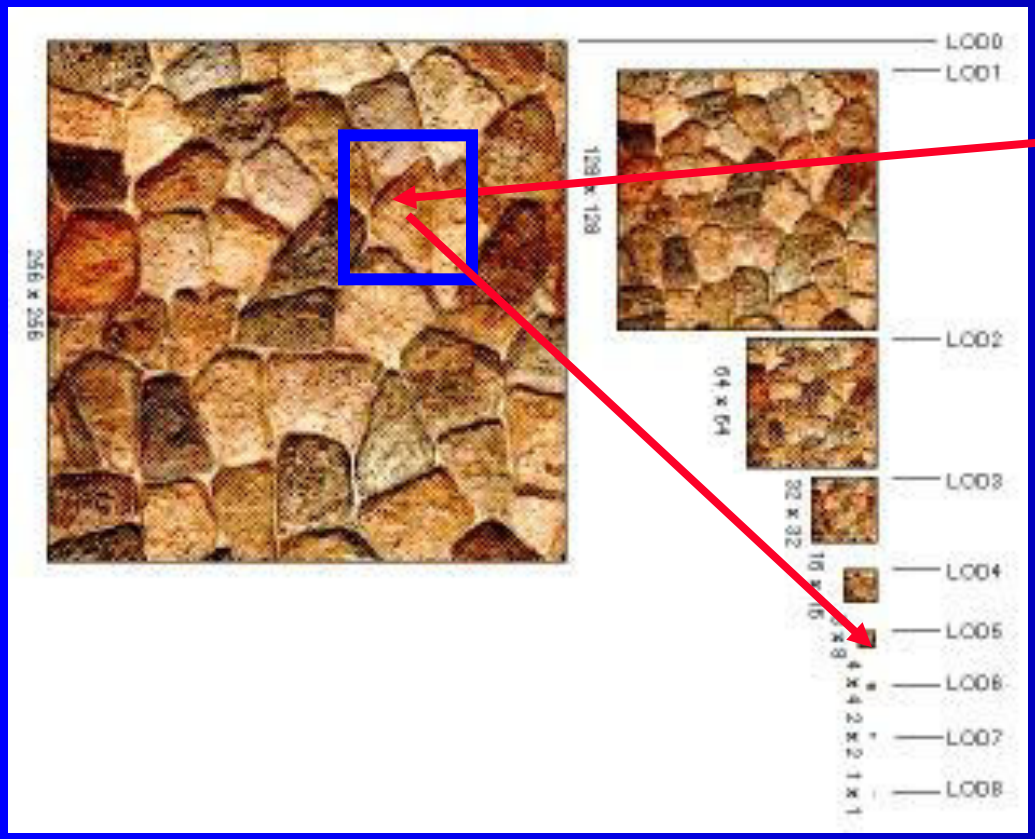


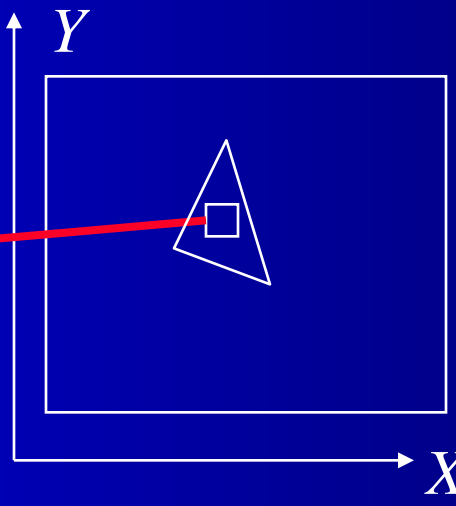
Torzításcsökkentés relaxációval

Textúrák szűrése



Mip-map adatstruktúra





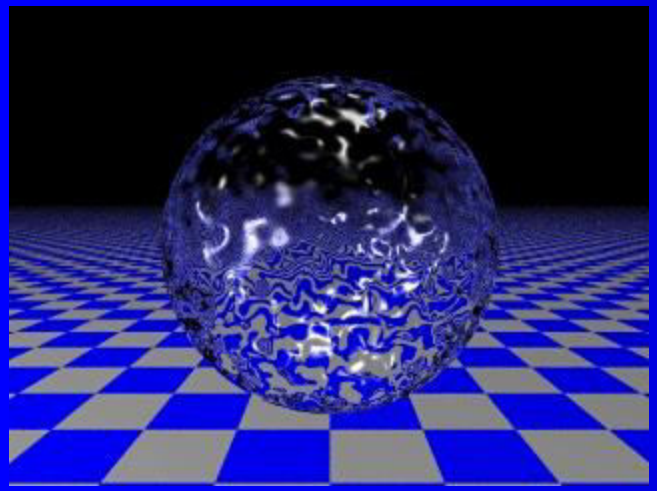
Ennek a deriváltját becsüljük:

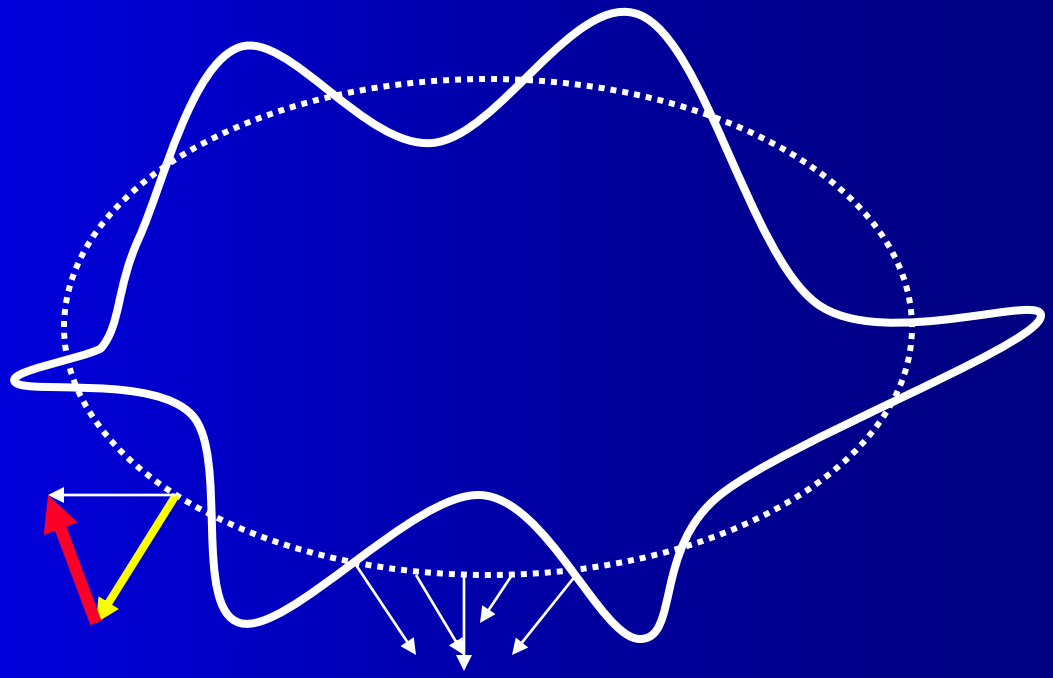
$$u = \frac{a_u X + b_u Y + c_u}{a_h X + b_h Y + c_h}$$

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

Bucka leképzés (Bump mapping)

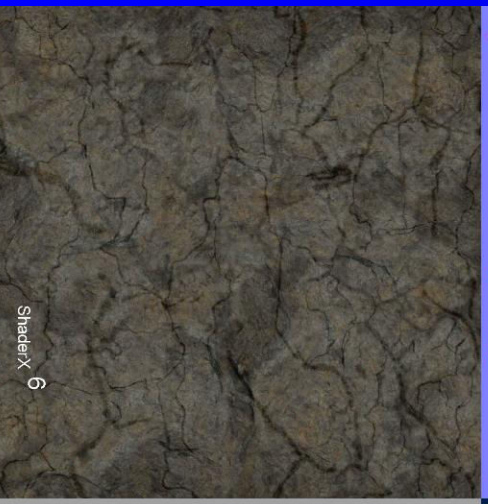




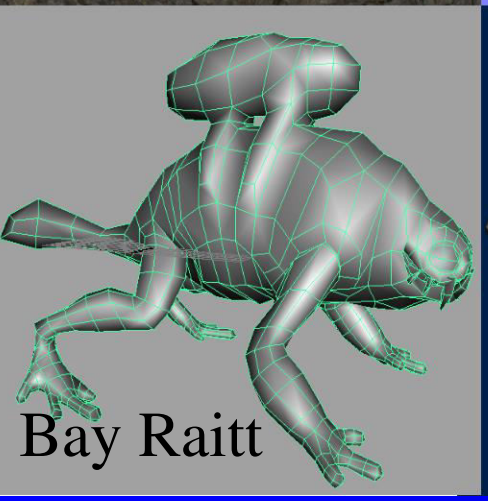


Tároljuk a normálvektort táblázatban
vagy a felületi elmozdulást

Displacement mapping











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

OpenGL

Szirmay-Kalos László

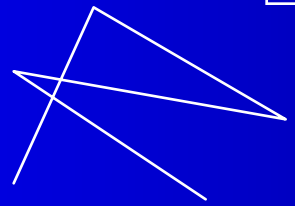
OpenGL: primitívek

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

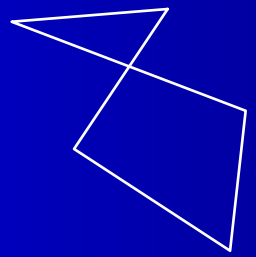


GL_POINTS

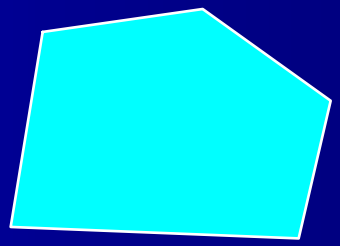
GL_LINES



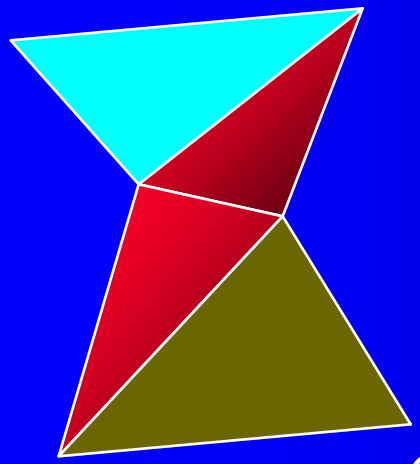
GL_LINE_STRIP



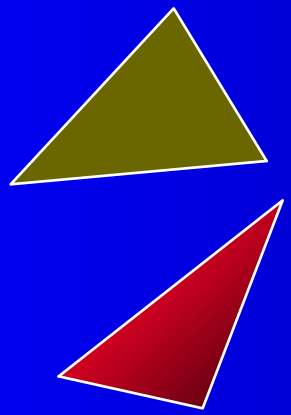
GL_LINE_LOOP



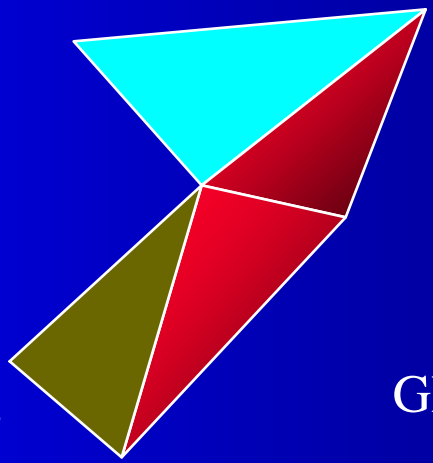
GL_POLYGON



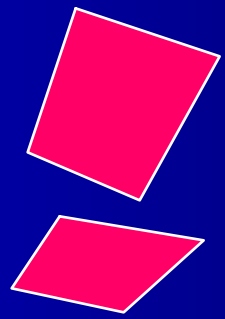
GL_TRIANGLE_STRIP



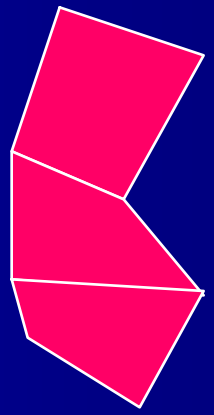
GL_TRIANGLES



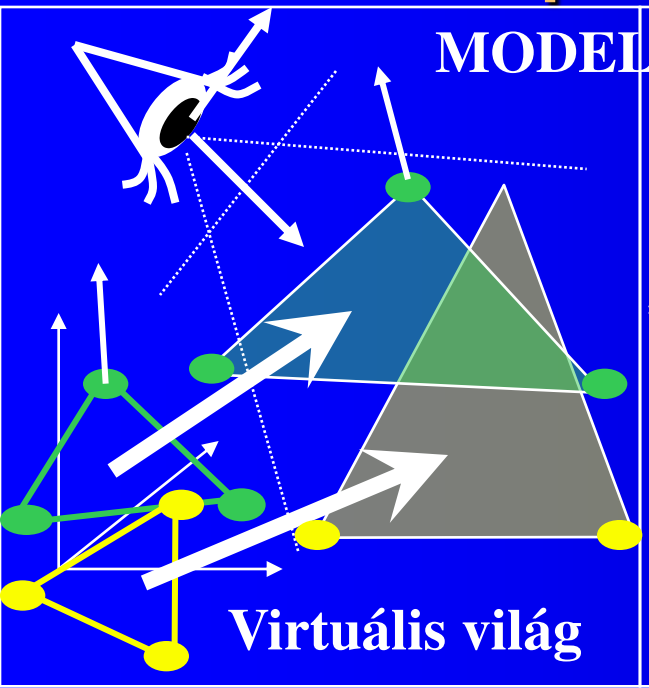
GL_TRIANGLE_FAN



GL_QUADS

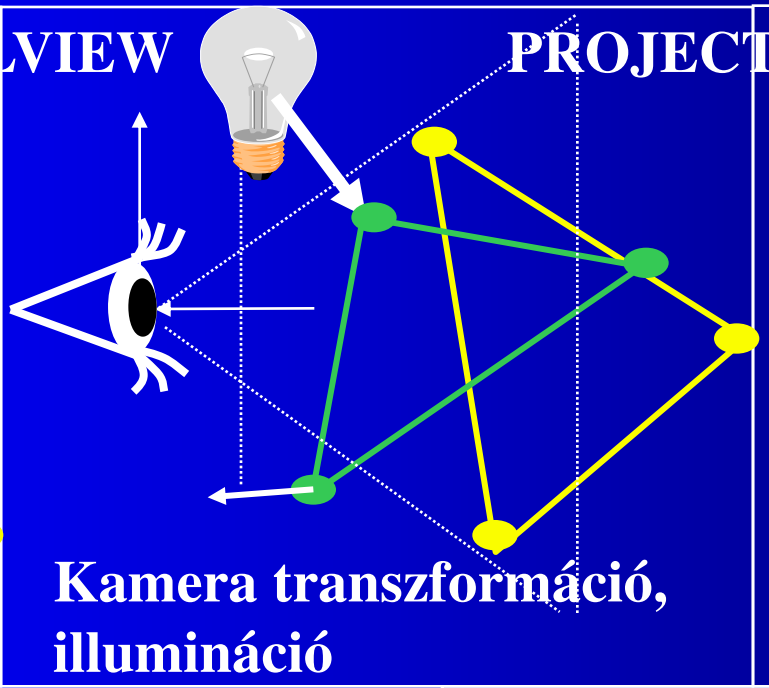


OpenGL csővezeték



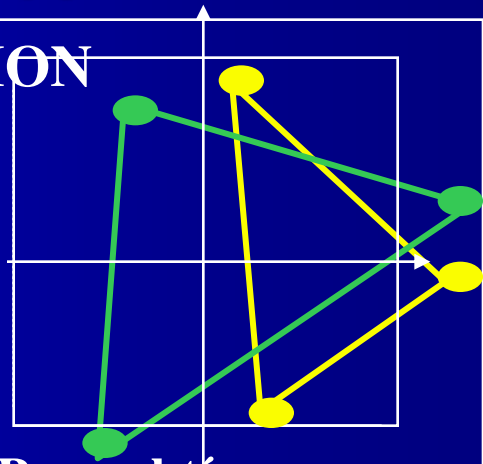
Virtuális világ

MODELVIEW

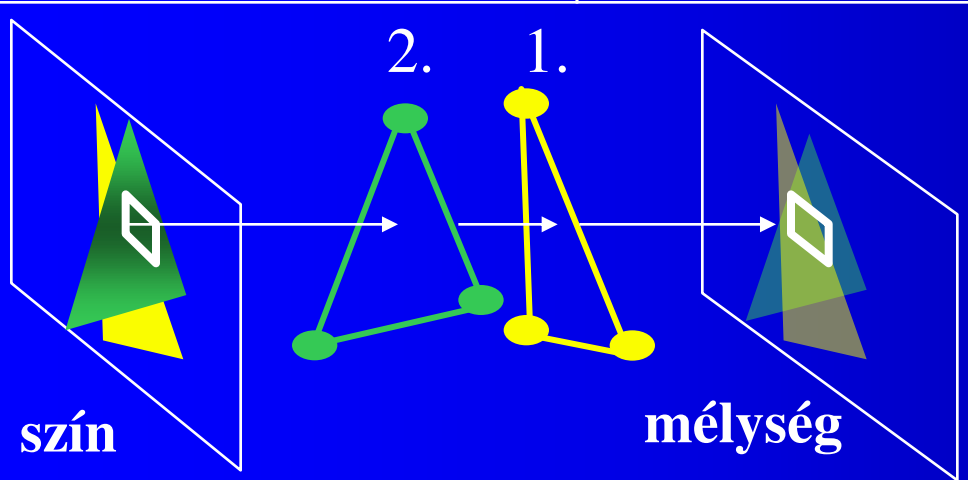


Kamera transzformáció, illumináció

PROJECTION



Perspektív transzformáció + Vágás homogén koordinátákban



szín

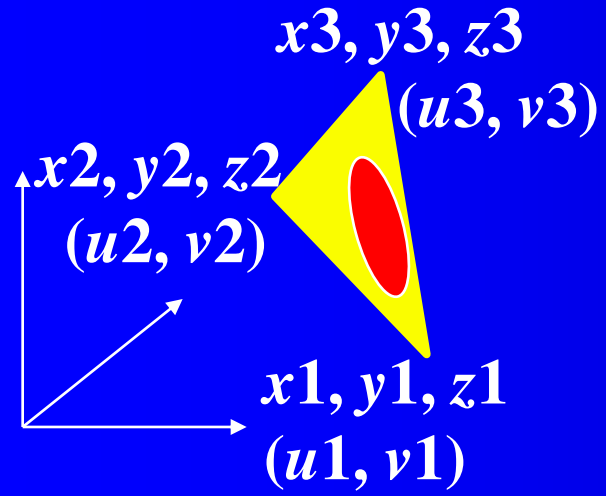
mélység

Képernyő transzf+Raszterizáció+interpoláció

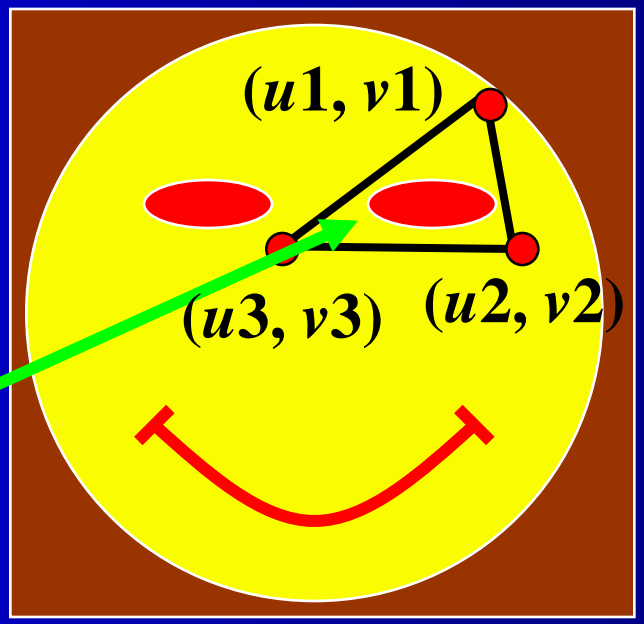


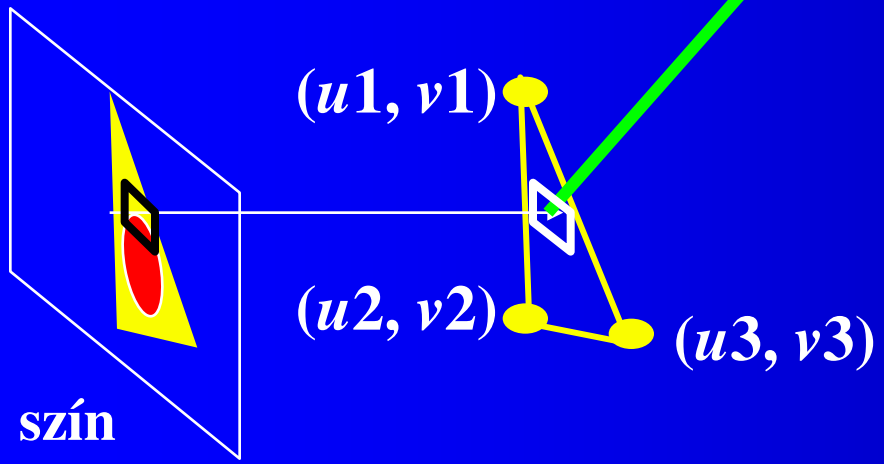
megjelenítés

Textúra leképzés



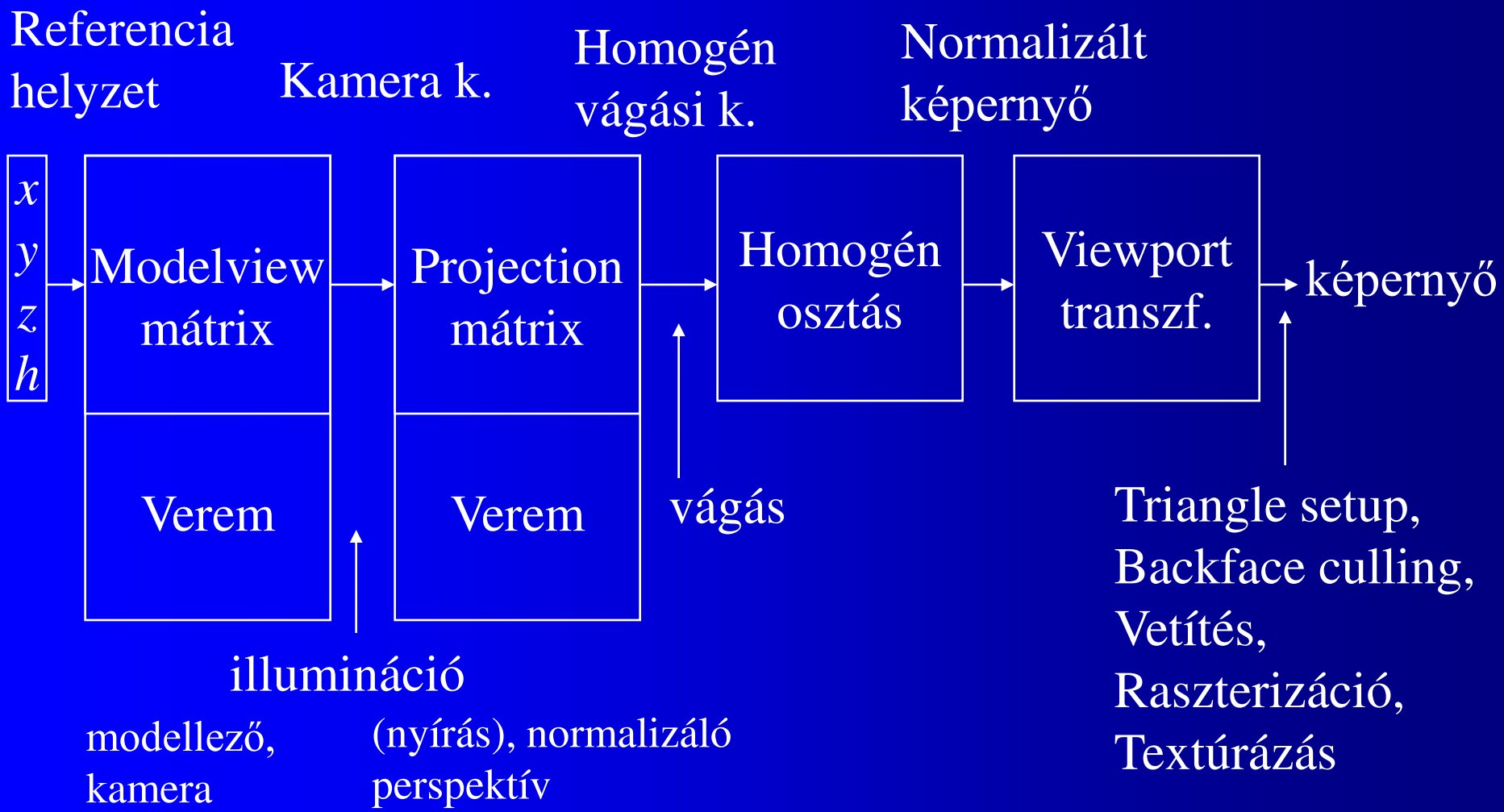
Interpolációval:
(u, v)



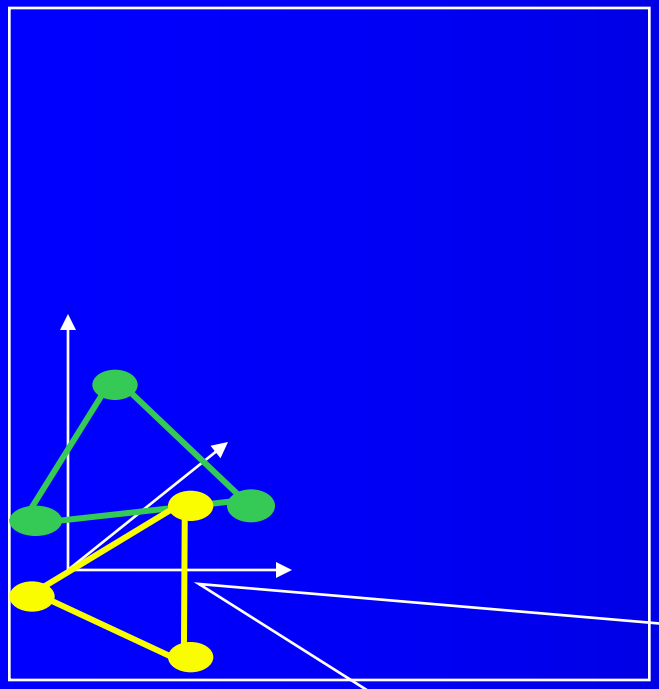


Textúra objektum a grafikus kártya memóriájában:
Kép + szűrés + kilógás kezelés

Transzformációk



A csővezeték etetése: A modell

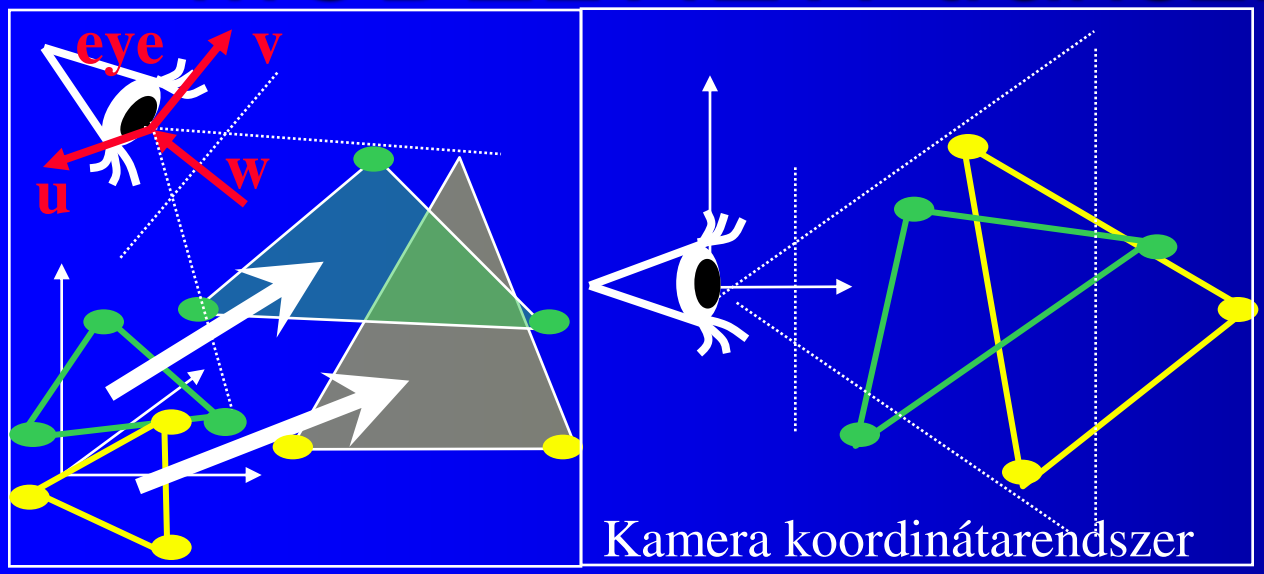


```
glBegin(GL_TRIANGLES);
        glNormal3f(nx1,ny1,nz1);
        glColor3f(r1,g1,b1);
        glTexCoord2f(u1,v1)
        glVertex3f(x1,y1,z1);

        glNormal3f(nx2,ny2,nz2);
        glColor3f(r2,g2,b2);
        glTexCoord2f(u2,v2)
        glVertex3f(x2,y2,z2);
        …
glEnd( );
```

állapot

állapot

MODELVIEW transzformáció



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

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
gluLookAt(eyex, eyey, eyez, vrpx, vrpy, vrpz, upx, upy, upz);  //VIEW
glTranslatef(px, py, pz);   //MODEL
glRotatef(ang, axisx,axisy,axisz);
glScalef(sx, sy, sz);
```

sorrend

$x,y,z,1$ | sx sy sz 1 | R | 1 1 1 px py pz 1 | T_{view} | 1 1 1 1

Illuminációk

A szín a megvilágításból keletkezzen:

glEnable(GL_LIGHTING);

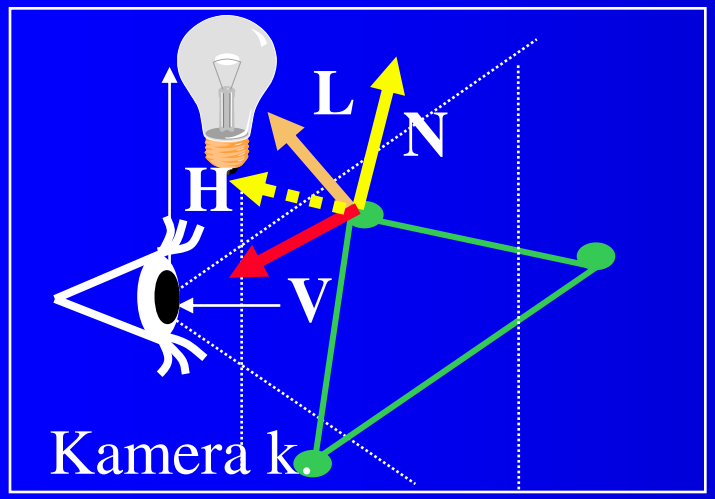
Phong-Blinn model

Color = Emission +

ka * Ia +

kd * Id · $(\mathbf{N} \bullet \mathbf{L})^{+}$ +

ks * Is · $((\mathbf{N} \bullet \mathbf{H})^{+})^{\text{shininess}}$



glLightModeli(
GL_LIGHT_MODEL_LOCAL_VIEWER,
GL_TRUE);

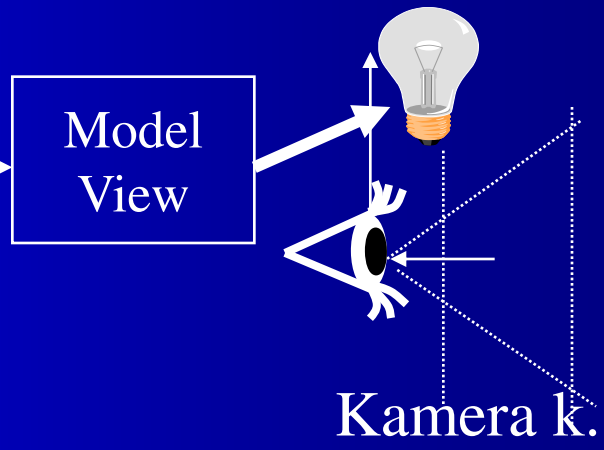
```
Glfloat kd[] = {0.0, 1.0, 1.0, 1.0};
Glfloat ks[] = {1.0, 0.5, 1.0, 1.0};
Glfloat ka[] = {0.2, 0.0, 0.2, 1.0};

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);
….
```

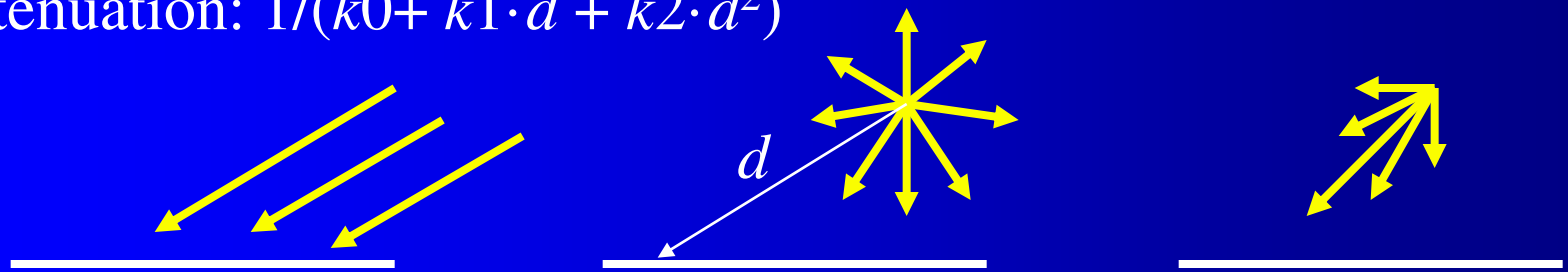
Fényforrások

```
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
```

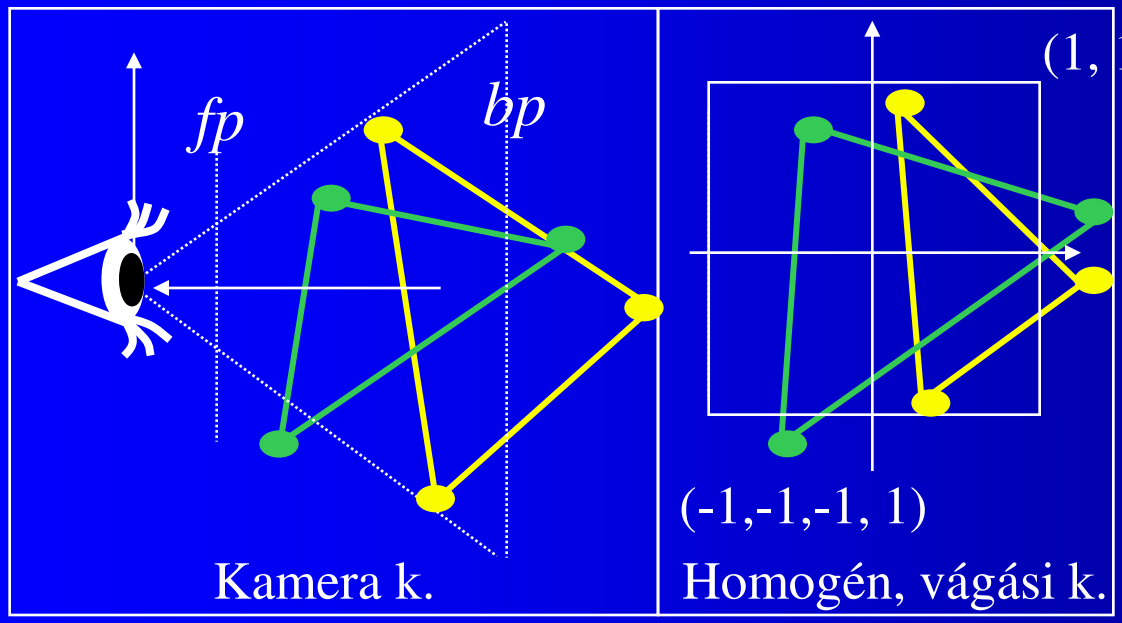
Ha $h = 0$, akkor direkcionális
egyébként pozícionális



- Külön fényintenzitás az ambiens, diffúz, spekuláris visszaverődéshez
- Direkcionális, Pozicionális (izotróp ill. spot)
- Attenuation: $1/(k0 + k1 \cdot d + k2 \cdot d^2)$



PROJECTION transzformáció



Vágás

$-h < X_h < h$

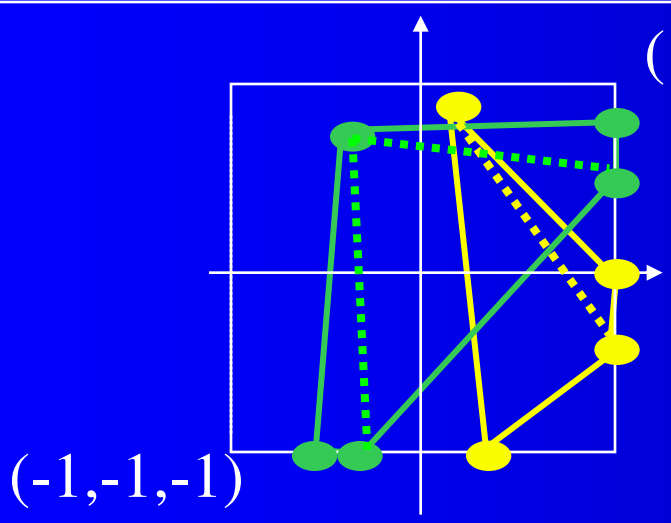
$-h < Y_h < h$

$-h < Z_h < h$

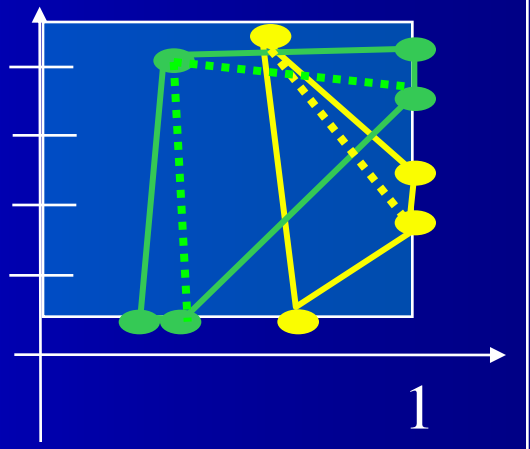
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );

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

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

Képernyő transzformáció



Normalizált képernyő k.:
Vágás és homogén osztás után



Képernyő k.

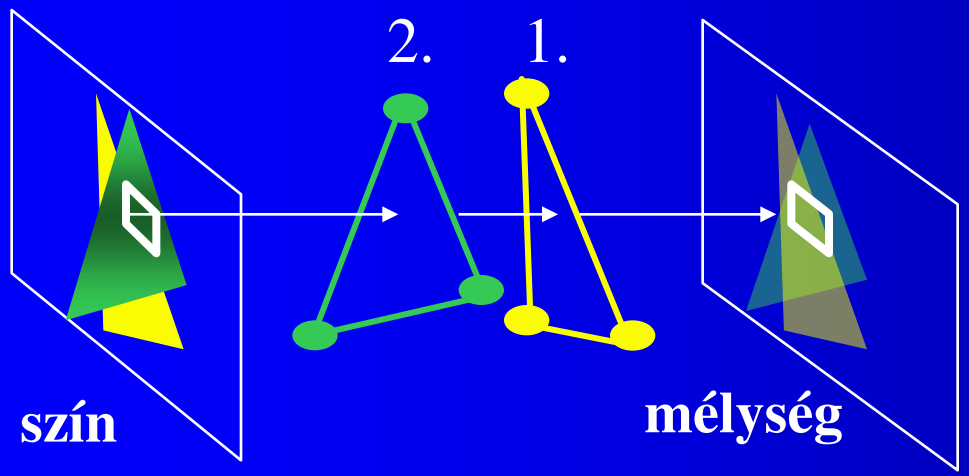
glViewport(left, bottom, width, height);

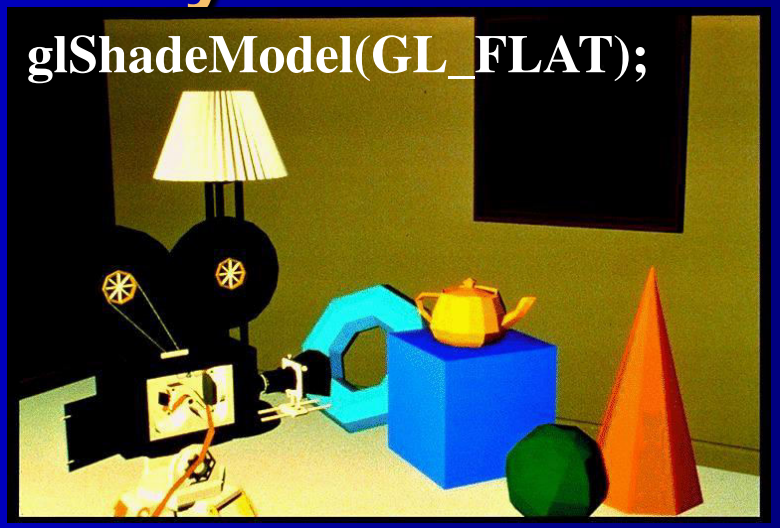
Z-bufferes takarás és konstans vagy Gouraud árnyalás

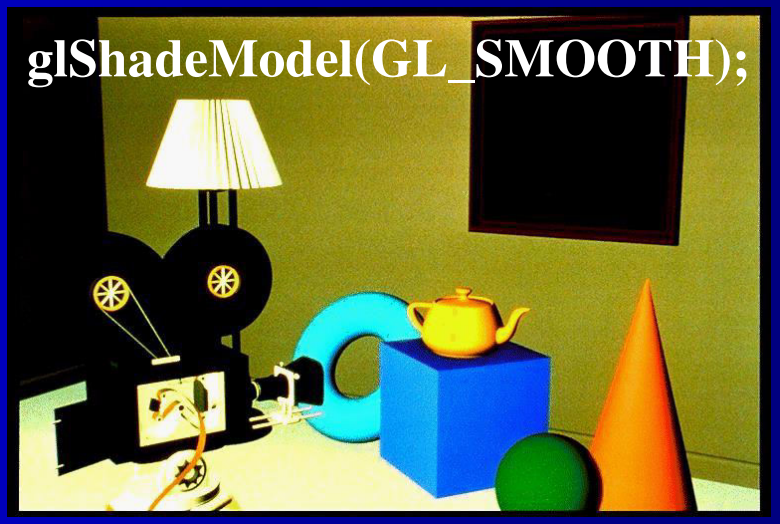
Inicializálás:

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

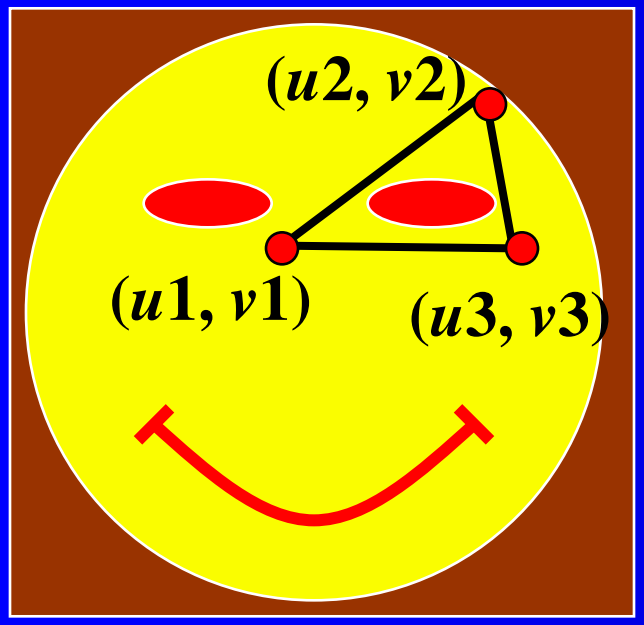
```
glEnable(GL_DEPTH_TEST);
glDisable(GL_CULL_FACE);
glShadeModel(GL_SMOOTH);
glDisable(GL_TEXTURE_2D);
```

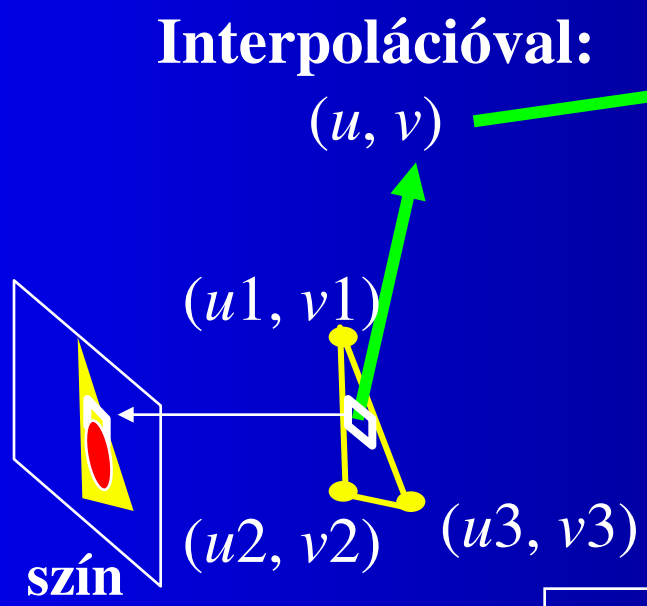


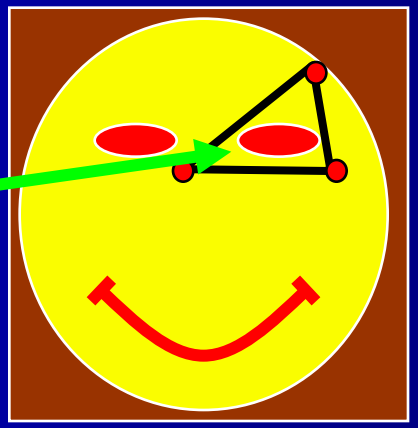




Textúra leképzés







Textúra objektum a grafikus kártya memóriájában

Ne az interpolált szín legyen a pixel szín:

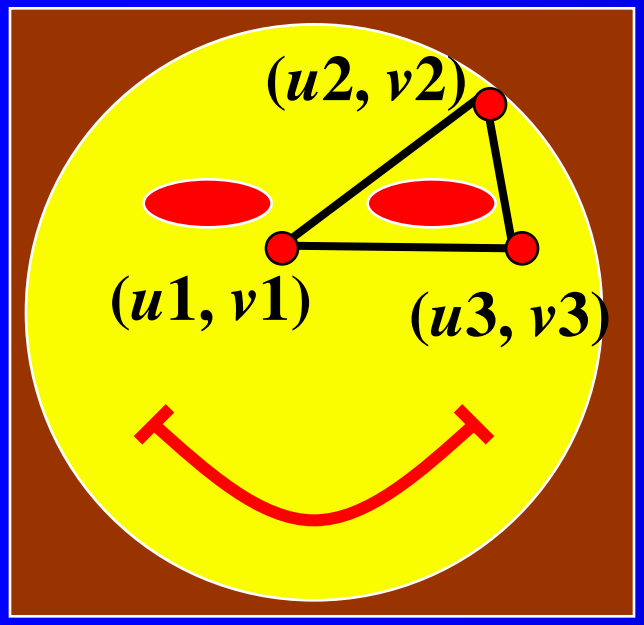
```
glEnable(GL_TEXTURE_2D);
glTexEnvi(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_REPLACE); //GL_MODULATE
```

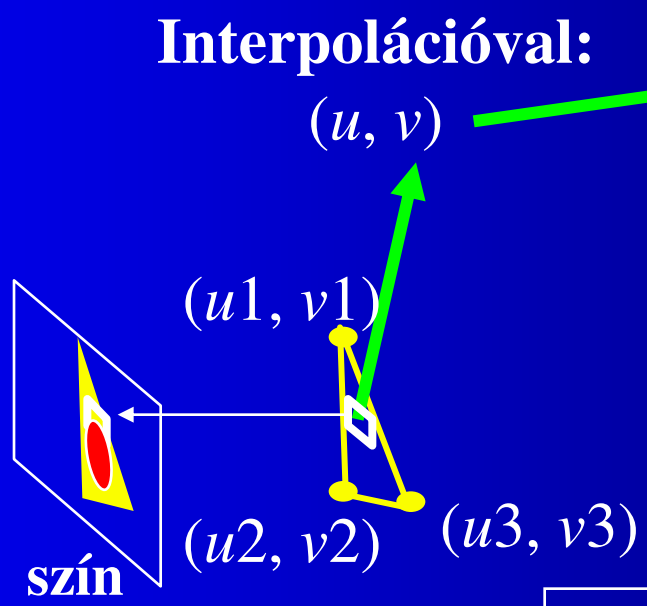
Textúra kiválasztása:

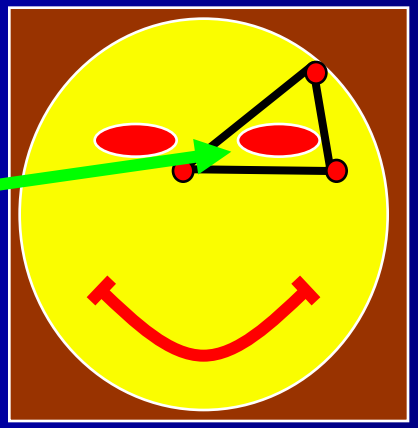
```
glBindTexture(GL_TEXTURE_2D, id);
```

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

Textúra leképzés







Textúra objektum a grafikus kártya memóriájában

Ne az interpolált szín legyen a pixel szín:

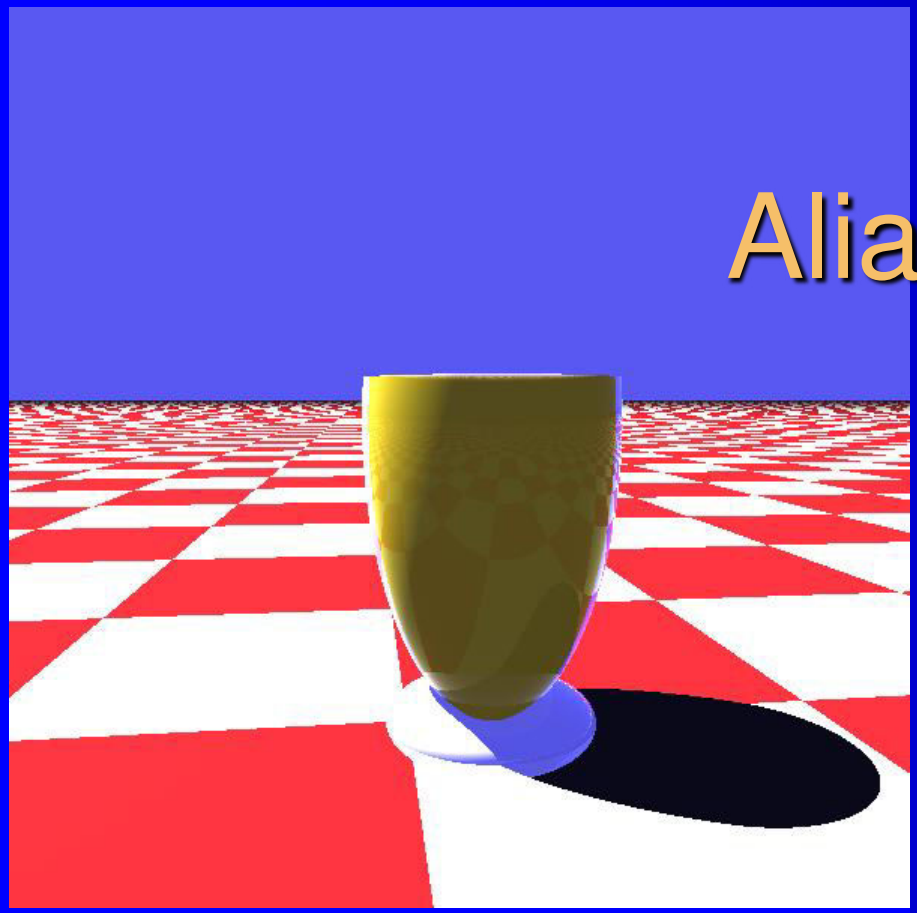
```
glEnable(GL_TEXTURE_2D);
glTexEnvi(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_REPLACE); //GL_MODULATE
```

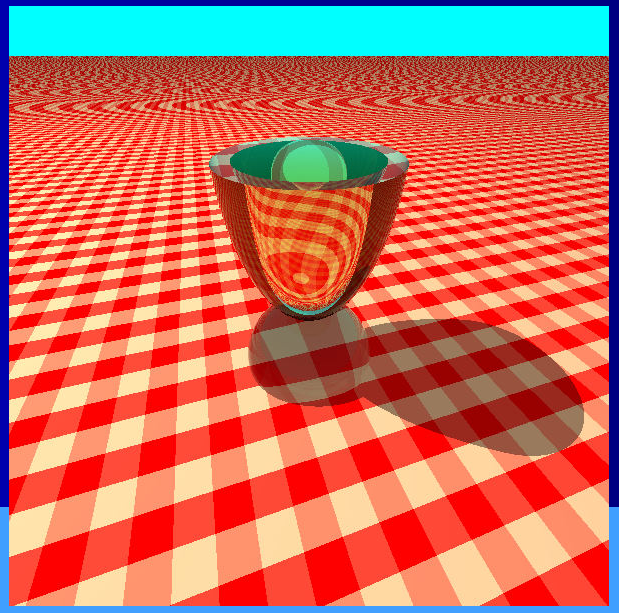
Textúra kiválasztása:

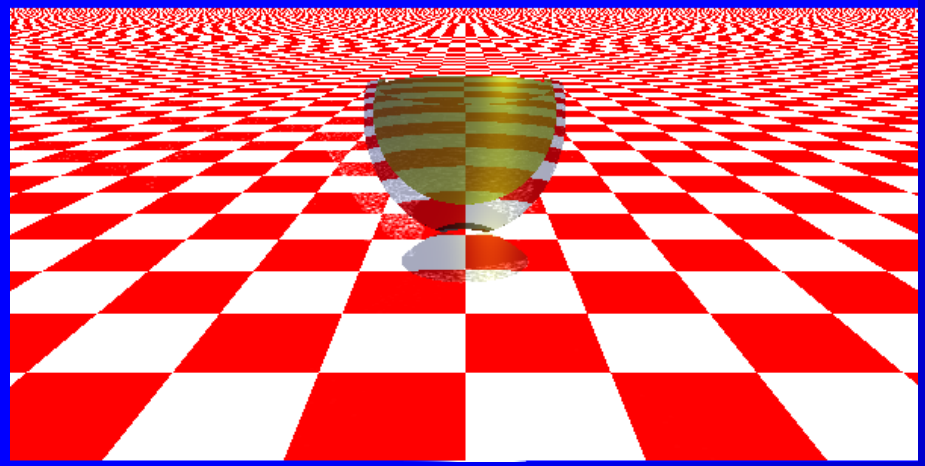
```
glBindTexture(GL_TEXTURE_2D, id);
```

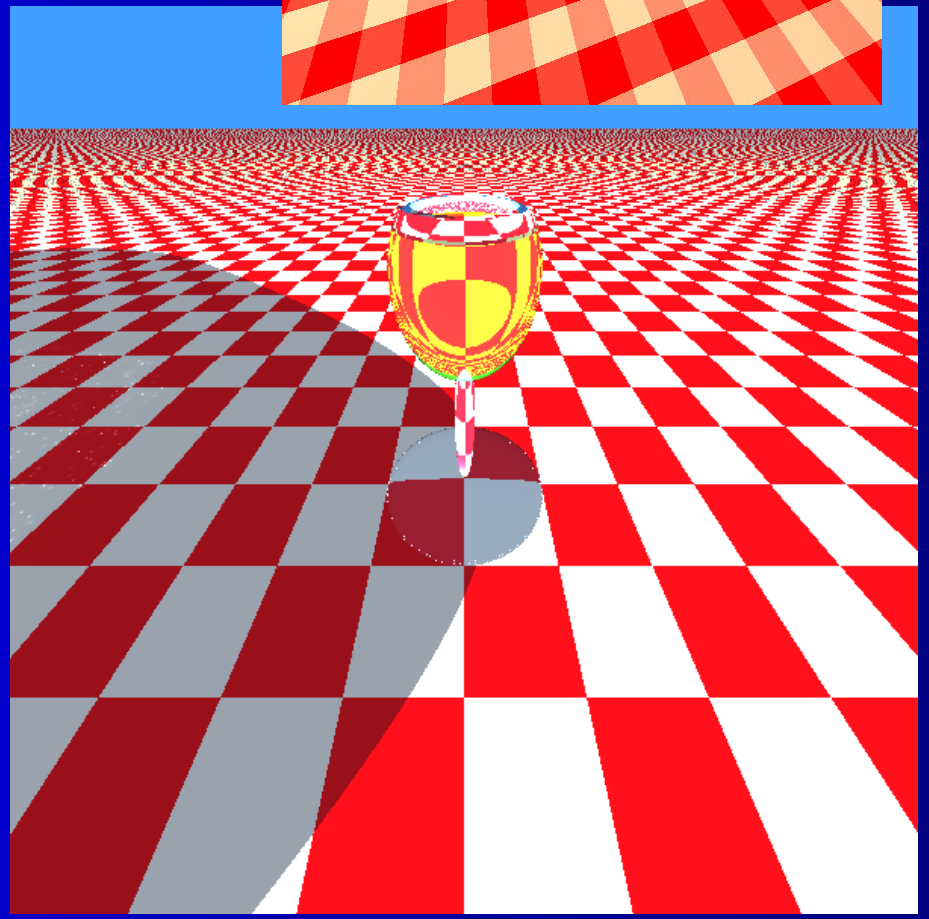
```
glBegin(GL_TRIANGLES);
glTexCoord2d(u1, v1);
glVertex3d(x1, y1, z1);
glTexCoord2d(u2, v2);
glVertex3d(x2, y2, z2);
…
```

Aliasing

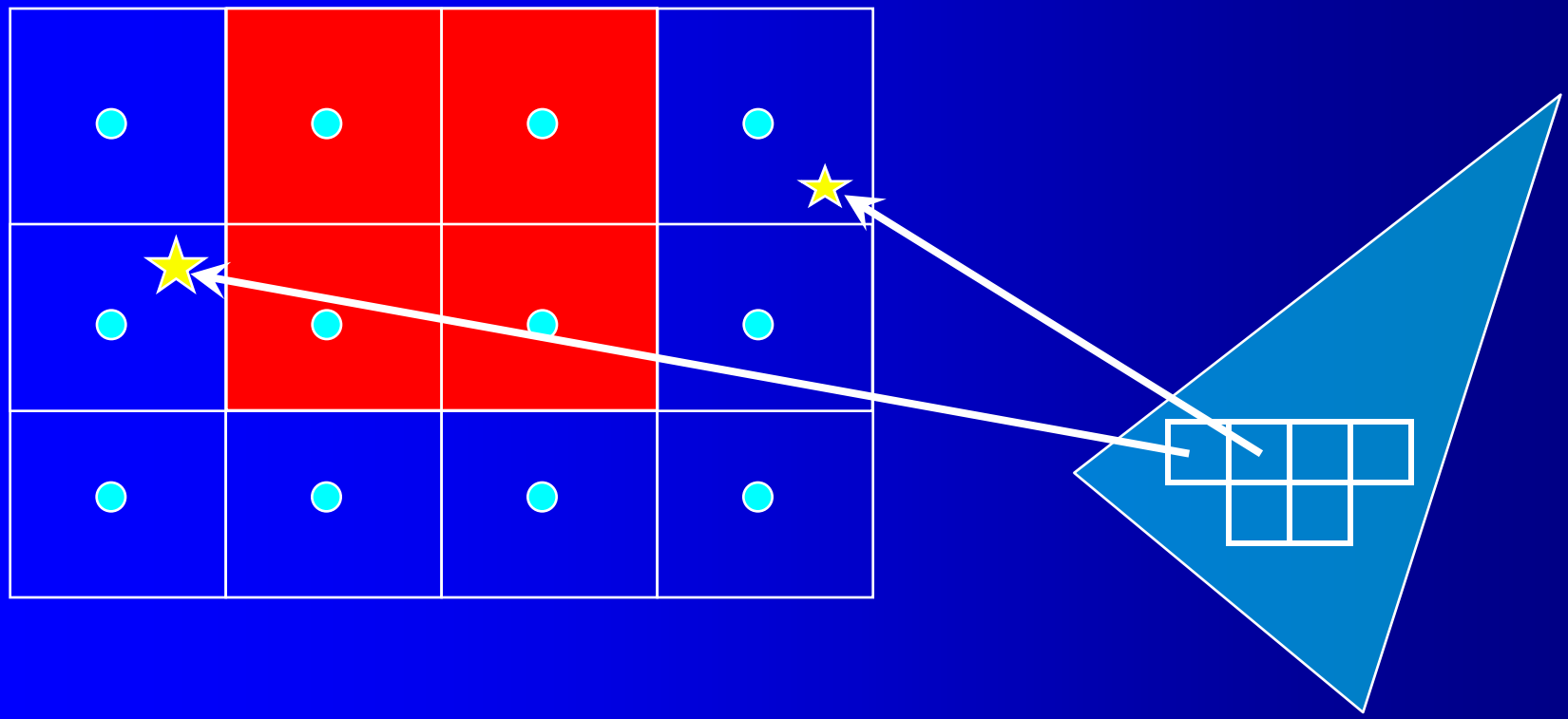








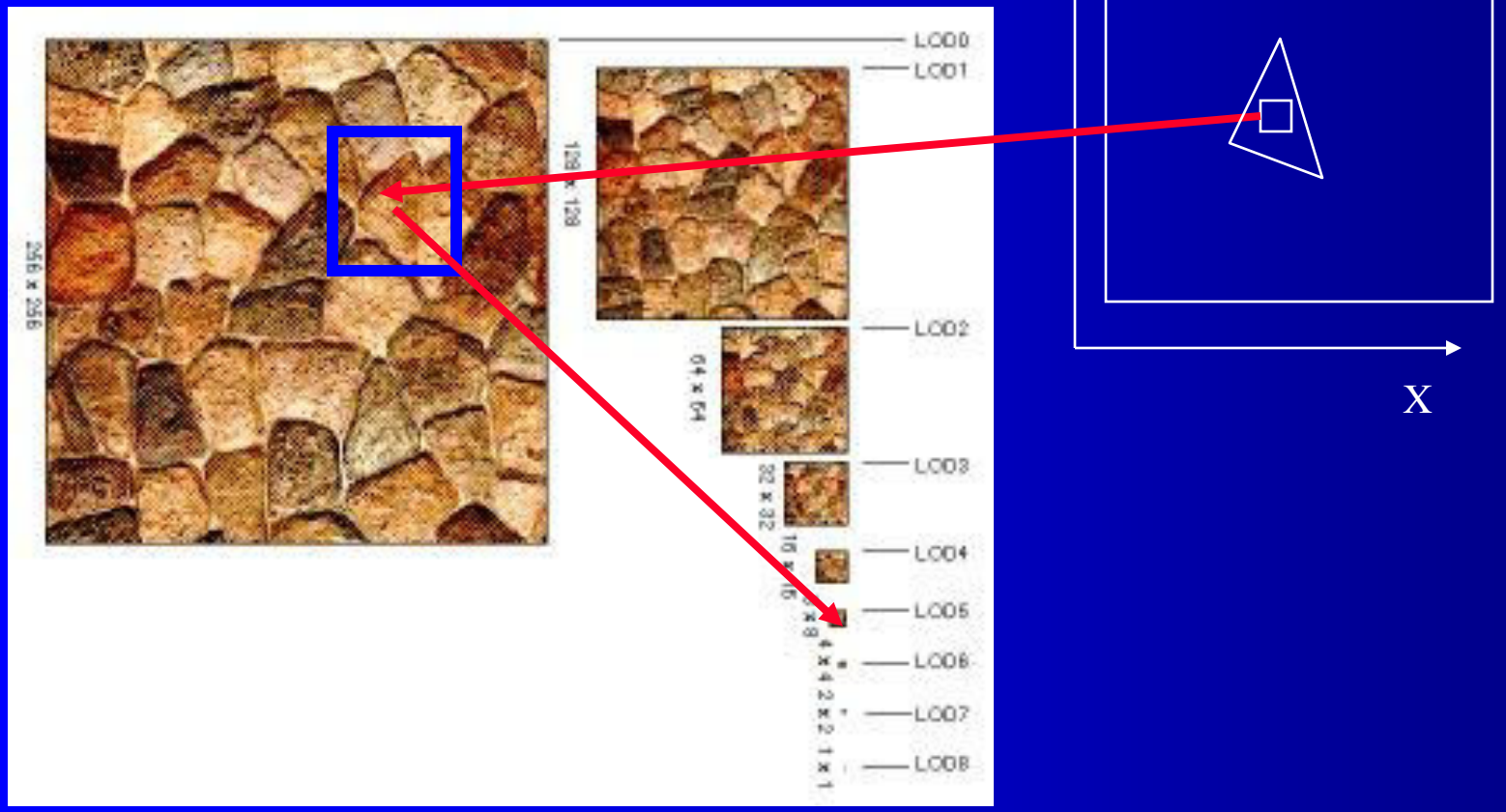
Textúra szűrés



Textúra tér

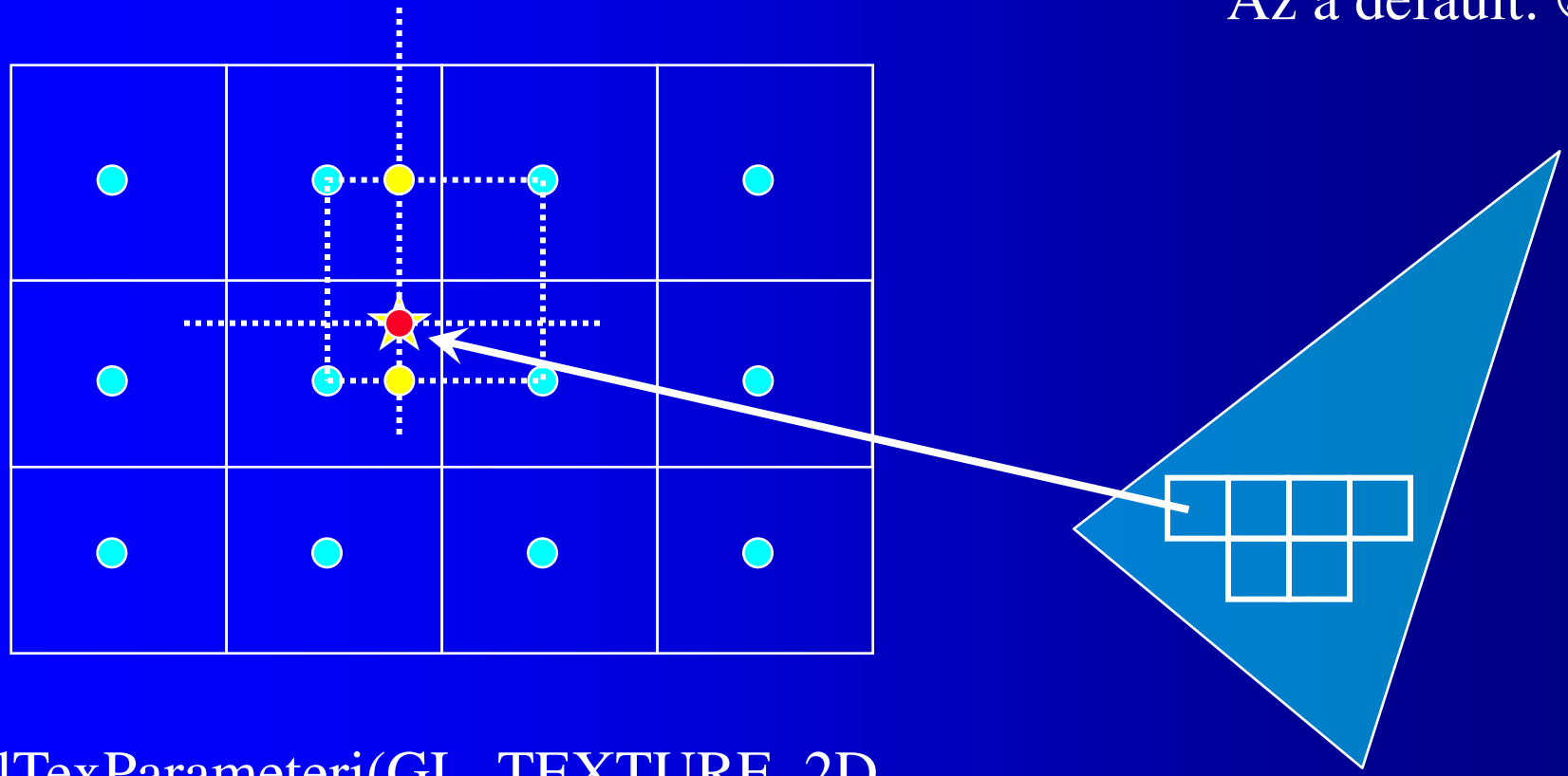
Képtér

Mip-map (multum in parvo)



Bi-linear textúra szűrés

Mip-map is van: ☺
Az a default: ☹

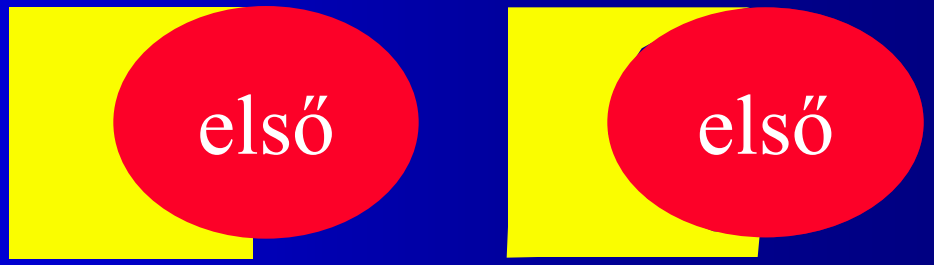


```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

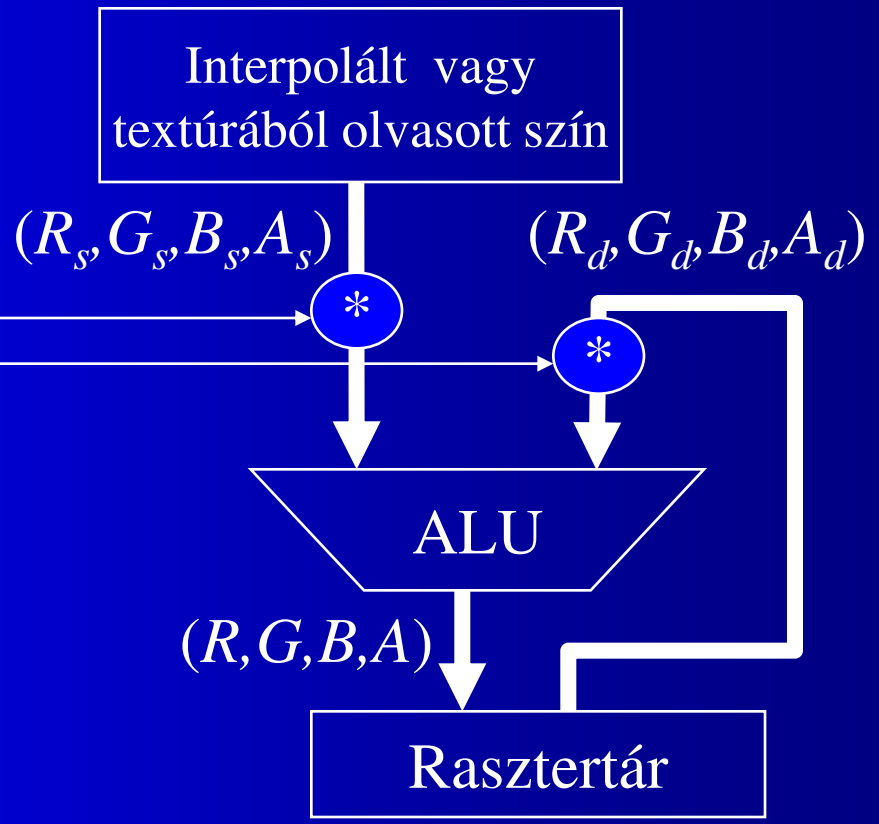
Textúra objektum létrehozása

```
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 fáljból.

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

Átlátszóság: Sorrend számít!



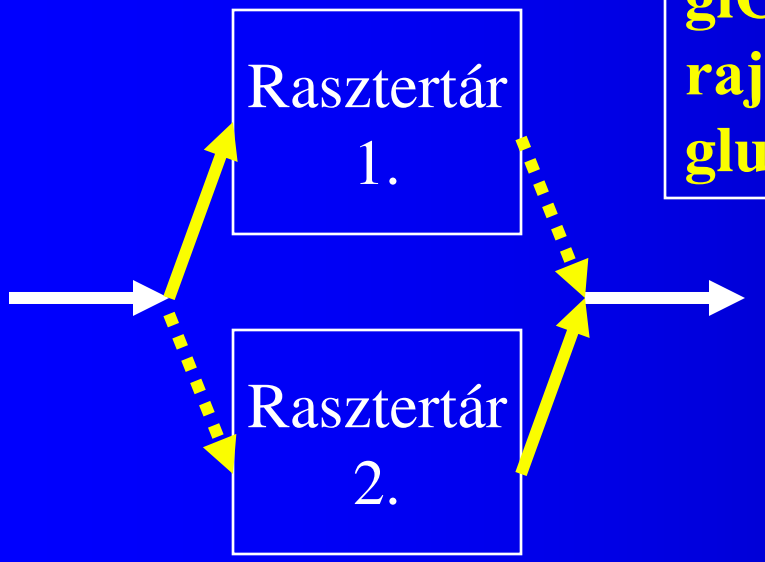

```
glEnable(GL_BLEND);

glBlendFunc(
    GL_SRC_ALPHA,
    GL_ONE_MINUS_SCR_ALPHA
);

glBegin( … );
    Geometria …
glEnd();

glDisable(GL_BLEND);
```



$$(R,G,B,A) = (R_sA_s+R_d(1\text{-}A_s),\ G_sA_s+G_d(1\text{-}A_s),\ B_sA_s+B_d(1\text{-}A_s),\ A_sA_s+A_d(1\text{-}A_s))$$

Dupla buffer animációhoz

Inicializálás:
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);

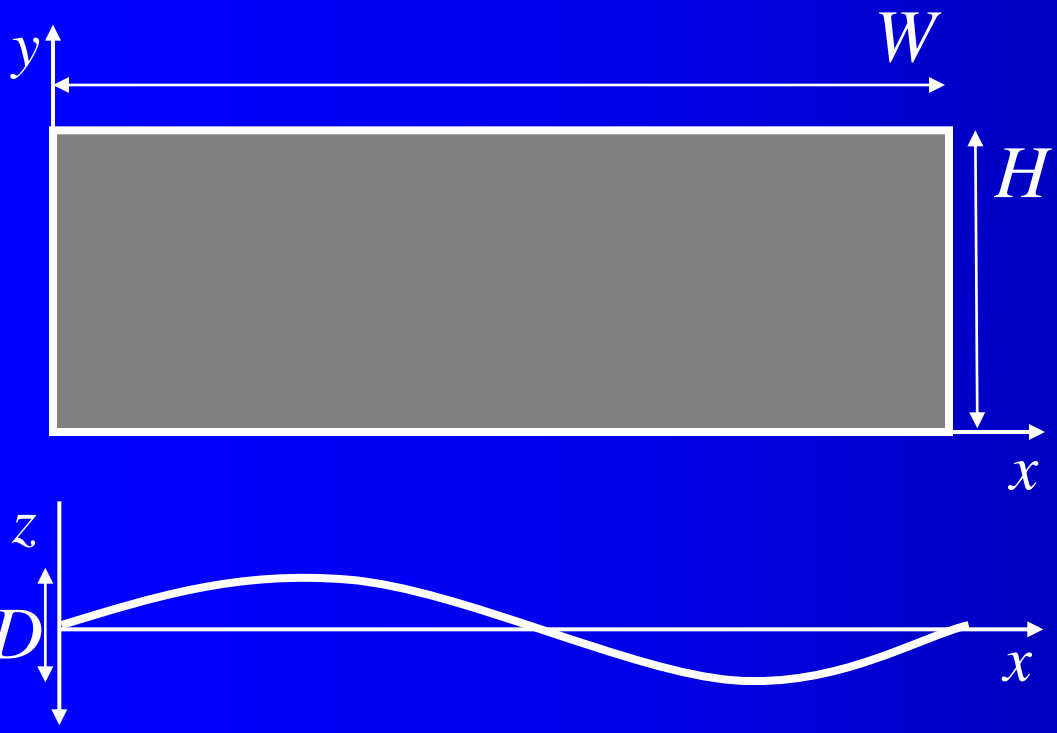



```
glClear(GL_COLOR_BUFFER_BIT);
rajzol…
glutSwapBuffers( );
```

Példa: Zászló

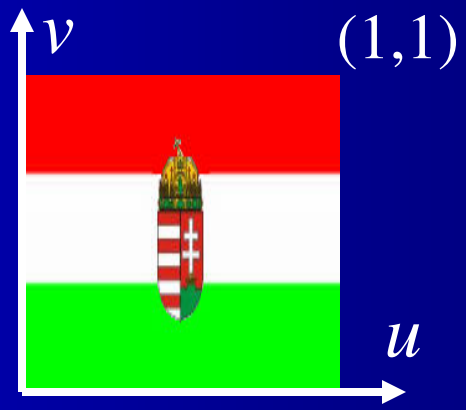
Geometria

$$x(u,v) = u \cdot W$$
$$y(u,v) = v \cdot H$$
$$z(u,v) = \sin(K \cdot u \cdot \mathrm{PI} + \mathrm{phase}(t)) \cdot D$$



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

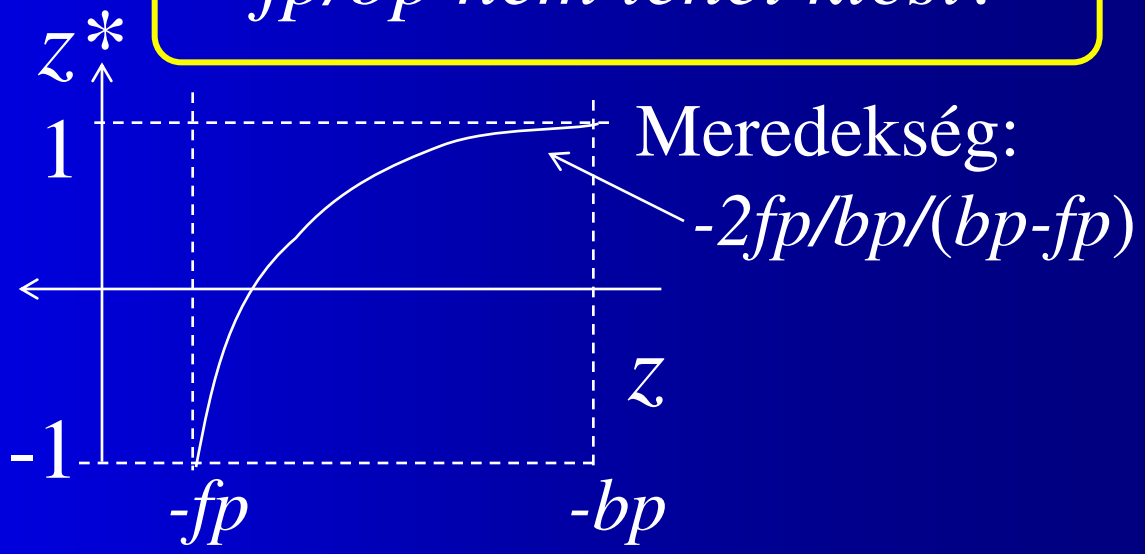
```
void onReshape(int winWidth, int winHeight) {
    glViewport(0, 0, winWidth, winHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluPerspective(54, (float)winWidth/(float)winHeight, 1, 100);
}
```

fp/bp nem lehet kicsi!

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

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

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



onInit

```
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);
```

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

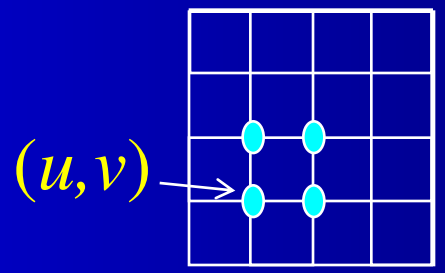
Texture

DrawFlag = tessellation+passing to OpenGL

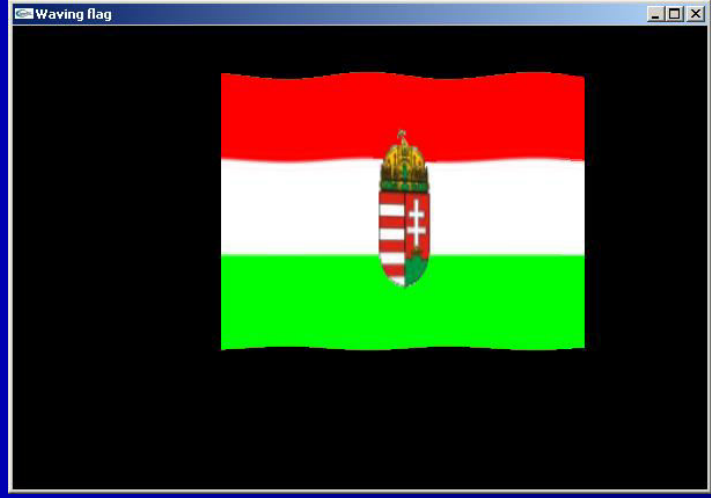
```
void FlagPoint(float u, float v) {
    glTexCoord2f(u, v);
    glVertex3f( u*W,  v*H,  sin(u*K*M_PI + phase)*D );
}
```

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

```
void DrawFlag( ) {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glTexEnvi(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 …

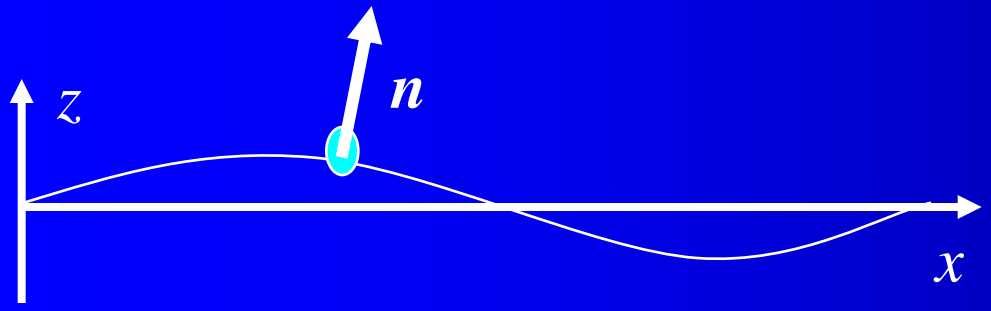
```
float phase = 0;

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





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

$$\partial \boldsymbol{r}/\partial u = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ W, & 0, & K \cdot \mathrm{PI} \cdot \cos(K \cdot u \cdot \mathrm{PI} + \mathrm{phase}) \cdot D \\ 0, & H, & 0 \end{bmatrix}$$

$$\partial \boldsymbol{r}/\partial v =$$

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

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);
```

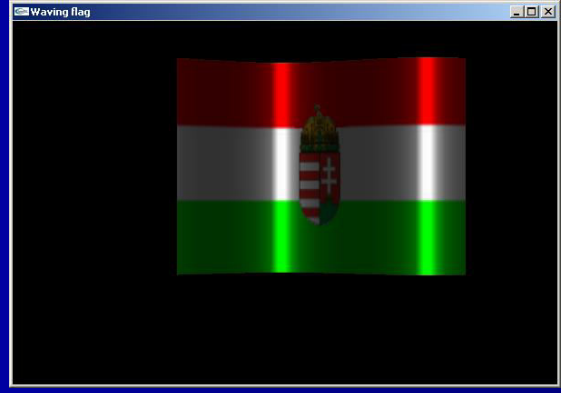
After gluLookAt

```
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 · $(\mathbf{N} \bullet \mathbf{L})^+$ +
ks * Is · $((\mathbf{N} \bullet \mathbf{H})^+)^{\text{shininess}}$

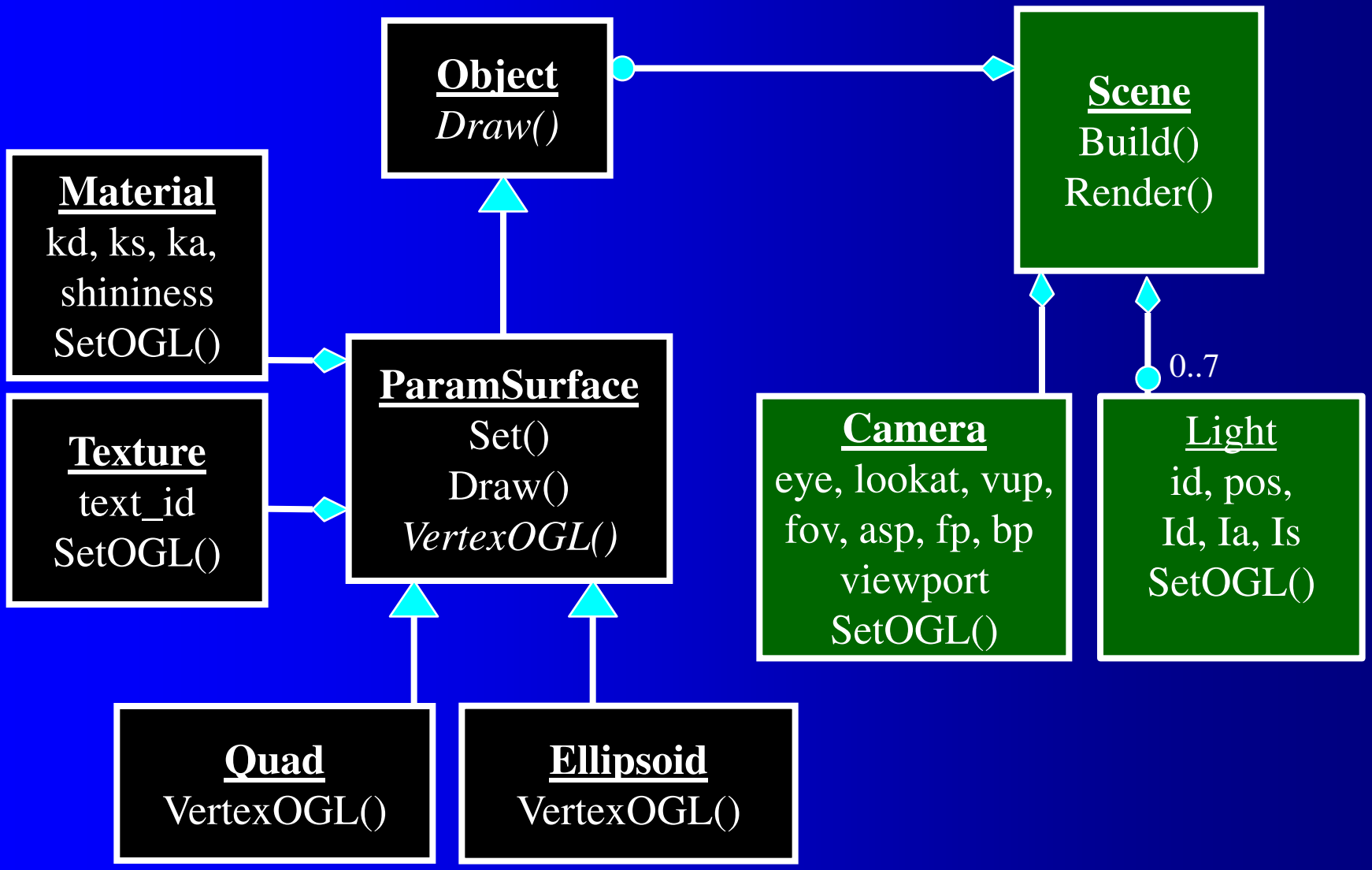
ColorVertex =
Ia +
Id · $(\mathbf{N} \bullet \mathbf{L})^+$ +
Is · $((\mathbf{N} \bullet \mathbf{H})^+)^{\text{shininess}}$

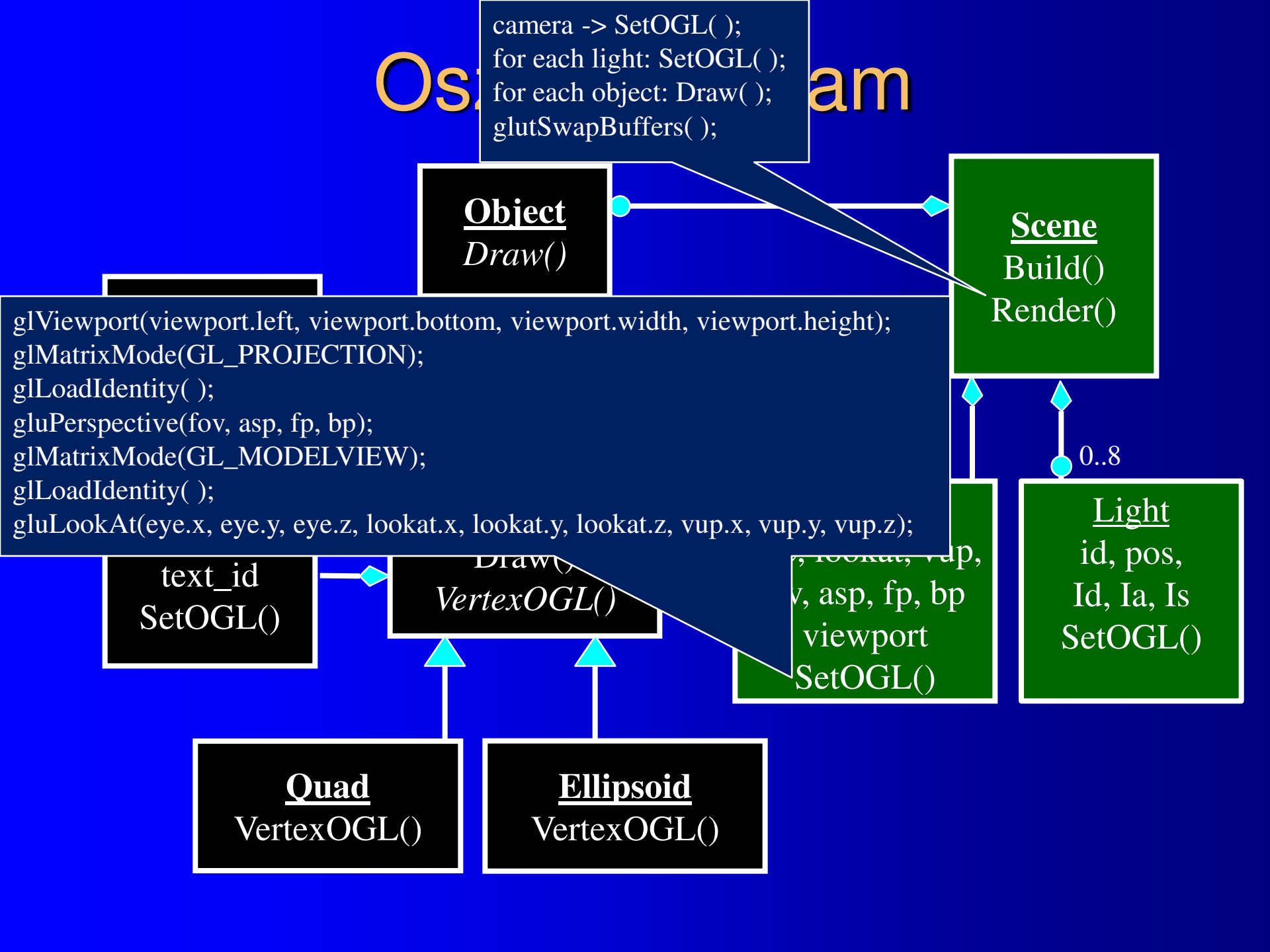
ColorPixel =
k * InterpolColorVertex



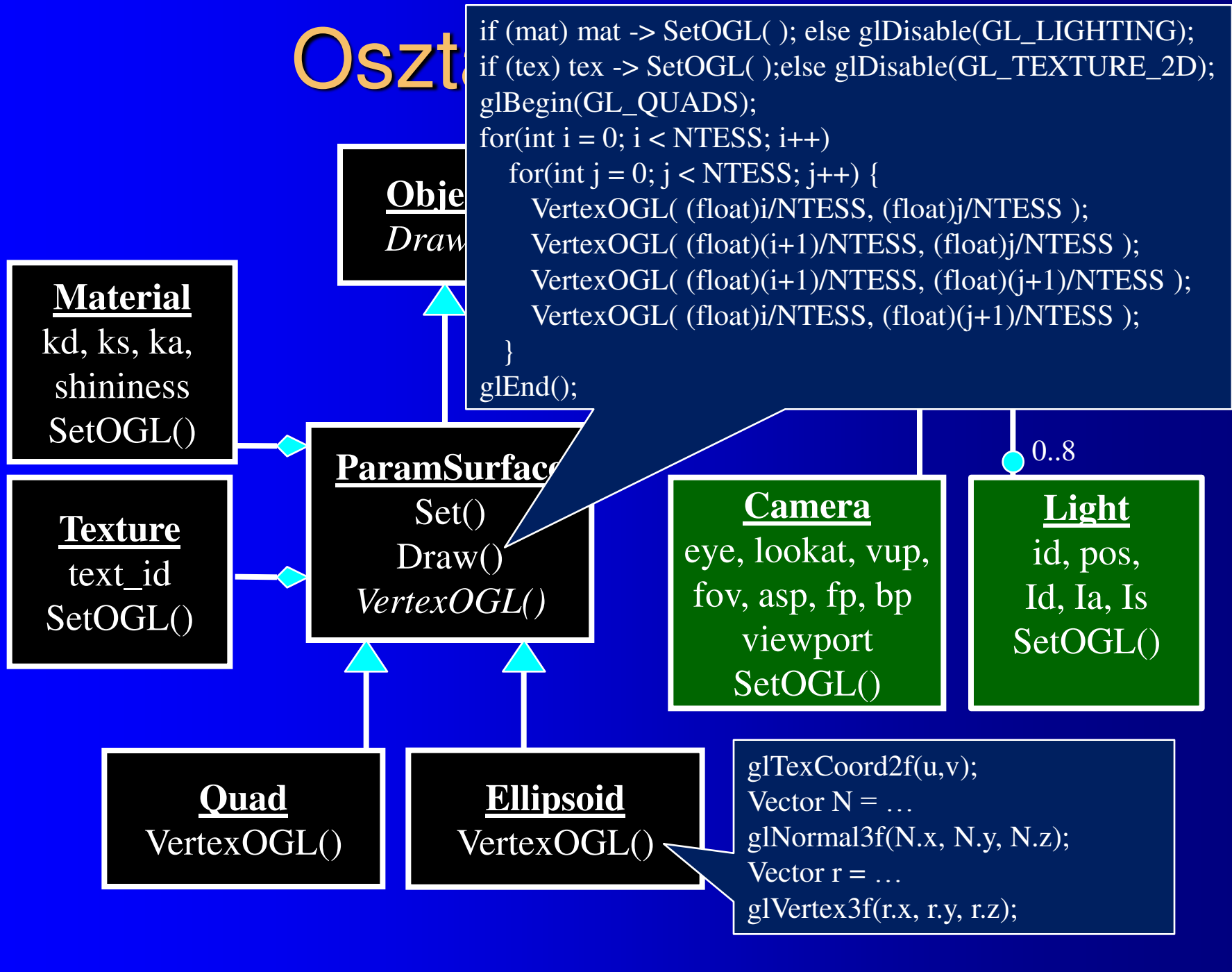

```
void DrawFlag( ) {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
    glTexEnvi(GL_TEXTURE_ENV,
              GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glEnable( GL_LIGHTING );    // illumination computation is on
    glEnable( GL_NORMALIZE ); // make 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);                          n(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);
        …                                            r(u,v)
    }
    glEnd();
    glutSwapBuffers( );
}
```

Osztálydiagram

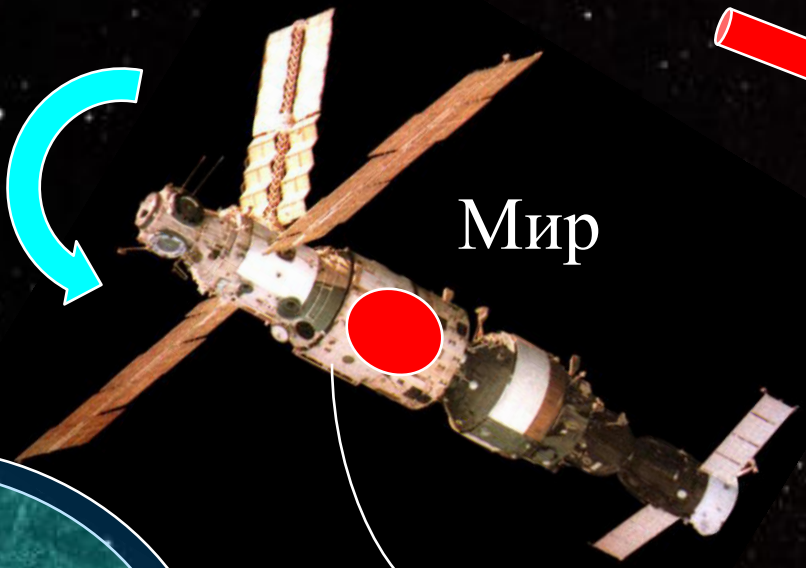



camera -> SetOGL( );
for each light: SetOGL( );
for each object: Draw( );
glutSwapBuffers( );
Object
Draw()
Scene
Build()
Render()
glViewport(viewport.left, viewport.bottom, viewport.width, viewport.height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
gluPerspective(fov, asp, fp, bp);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
gluLookAt(eye.x, eye.y, eye.z, lookat.x, lookat.y, lookat.z, vup.x, vup.y, vup.z);
0..8
text_id
SetOGL()
VertexOGL()
, asp, fp, bp
viewport
SetOGL()
Light
id, pos,
Id, Ia, Is
SetOGL()
Quad
VertexOGL()
Ellipsoid
VertexOGL()

Oszt


if (mat) mat -> SetOGL( ); else glDisable(GL_LIGHTING);
if (tex) tex -> SetOGL( );else glDisable(GL_TEXTURE_2D);
glBegin(GL_QUADS);
for(int i = 0; i < NTESS; i++)
for(int j = 0; j < NTESS; j++) {
VertexOGL((float)i/NTESS, (float)j/NTESS);
VertexOGL((float)(i+1)/NTESS, (float)j/NTESS);
VertexOGL((float)(i+1)/NTESS, (float)(j+1)/NTESS);
VertexOGL((float)i/NTESS, (float)(j+1)/NTESS);
}
glEnd();
Obje
Draw
Material
kd, ks, ka,
shininess
SetOGL()
Texture
text_id
SetOGL()
ParamSurfac
Set()
Draw()
VertexOGL()
0..8
Camera
eye, lookat, vup,
fov, asp, fp, bp
viewport
SetOGL()
Light
id, pos,
Id, Ia, Is
SetOGL()
Quad
VertexOGL()
Ellipsoid
VertexOGL()
glTexCoord2f(u,v);
Vector N = …
glNormal3f(N.x, N.y, N.z);
Vector r = …
glVertex3f(r.x, r.y, r.z);

3. házi: műhold befogás





Mesterséges hold

Forgó bolygó procedurális textúrával, nap által megvilágítva

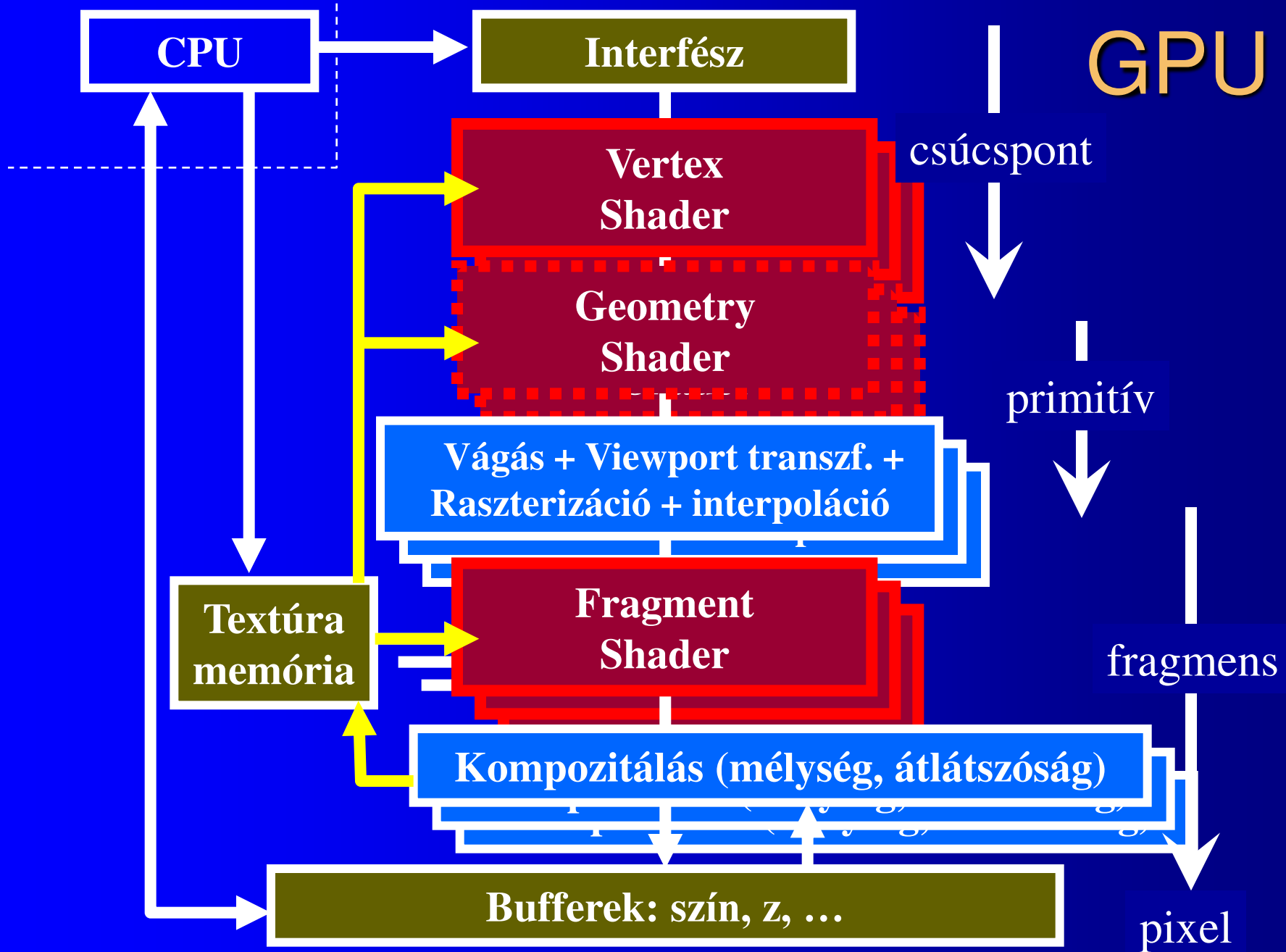
Légkör

Gumikötél

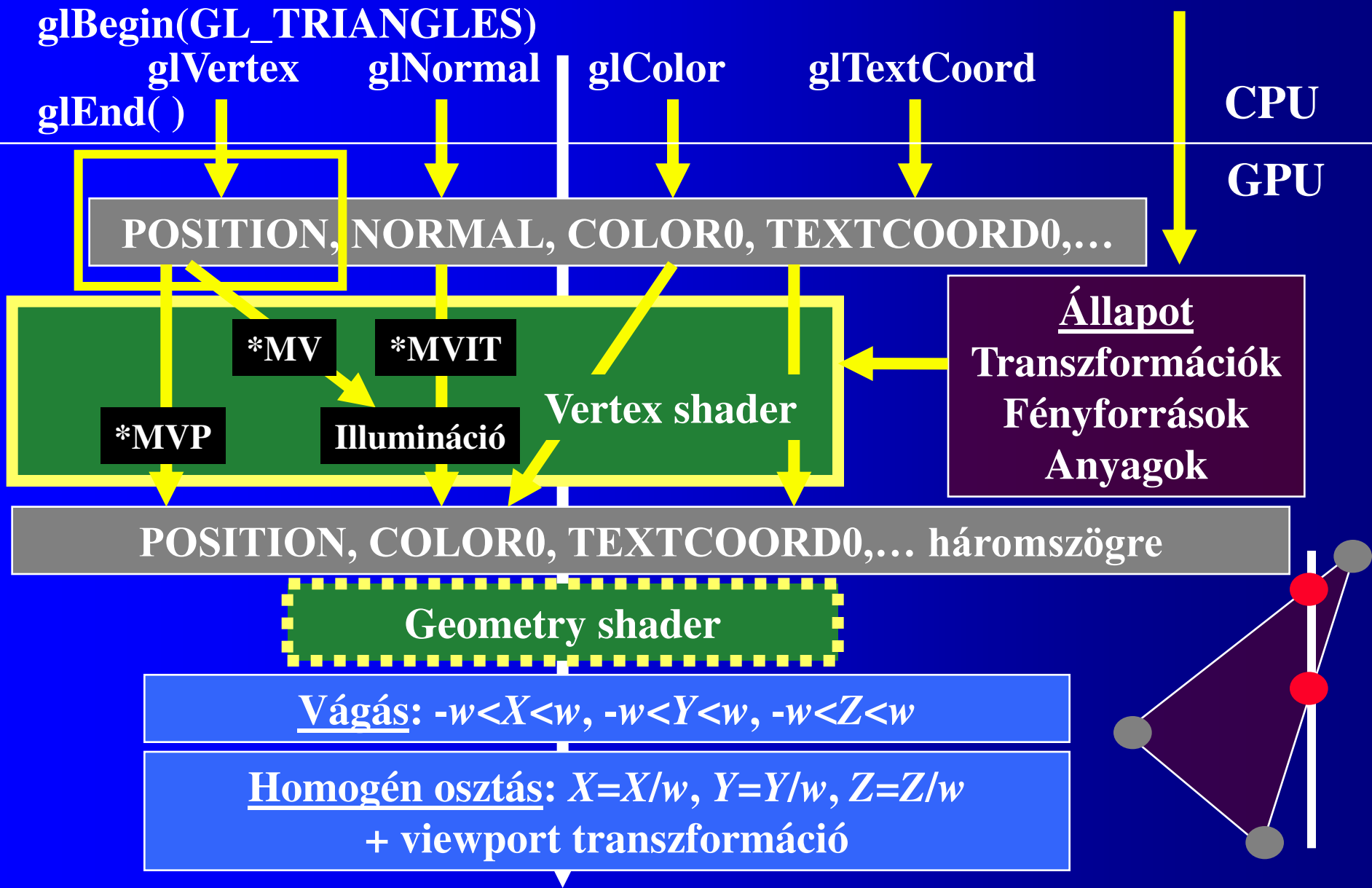
Mi a kötél végén rángatózunk

GPU

Szirmay-Kalos László


CPU
Interfész
GPU
Vertex Shader
csúcspont
Geometry Shader
primitív
Vágás + Viewport transzf. + Raszterizáció + interpoláció
Textúra memória
Fragment Shader
fragmens
Kompozitálás (mélység, átlátszóság)
Bufferek: szín, z, …
pixel

Vertex shader és környezete



“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;
}
```

Változó bemenet

Uniform

Változó kimenet

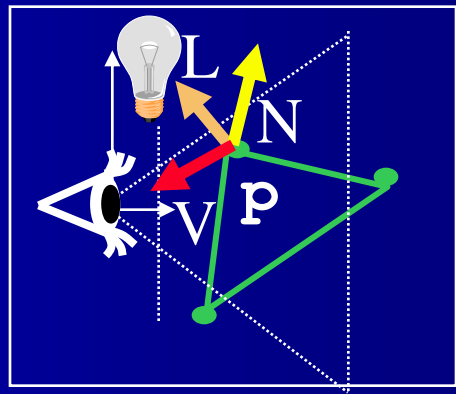
glDisable(GL_LIGHTING);

Mezők elérése:

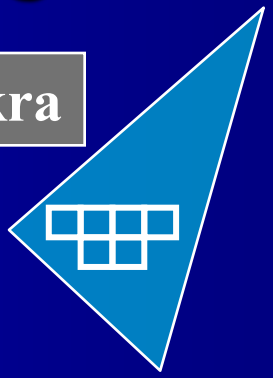
```
v.x, v.y, v.z, v.w, v.xy, v.wxx
c.r, c.rgb, c.ar, …
```

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 környezete



POSITION, COLOR0, TEXTCOORD0,…háromszög csúcsokra

Háromszög setup, kitöltés és interpoláció

POSITION, COLOR0, TEXTCOORD0,… pixelekre

Fragment shader
Textúrázás

Állapot
Textúra azonosító és paraméterek

POSITION, COLOR

Textúra memória

Kompozitálás: blending, z-bufferelés

Rasztertár (szín, mélység, …)

“Standard” pixel shader

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

glDisable(GL_TEXTURE_2D);

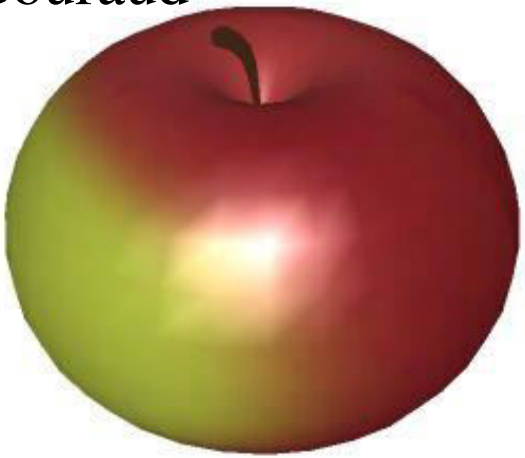
```
void main( in float2 texcoord    : TEXCOORD0,
           in float3 color       : COLOR0,
           uniform sampler2D texture_map,
           out float4 ocolor     : COLOR )
{
     ocolor = tex2D(texture_map, texcoord)*color;
}
```

glEnable(GL_TEXTURE_2D);

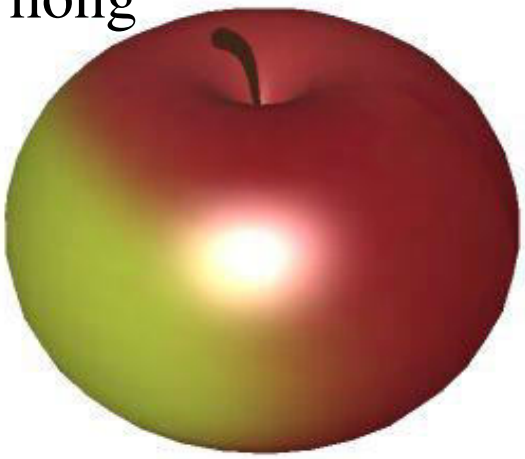
glTexEnvi(GL_TEXTURE_ENV,
GL_TEXTURE_ENV_MODE, **GL_MODULATE**);

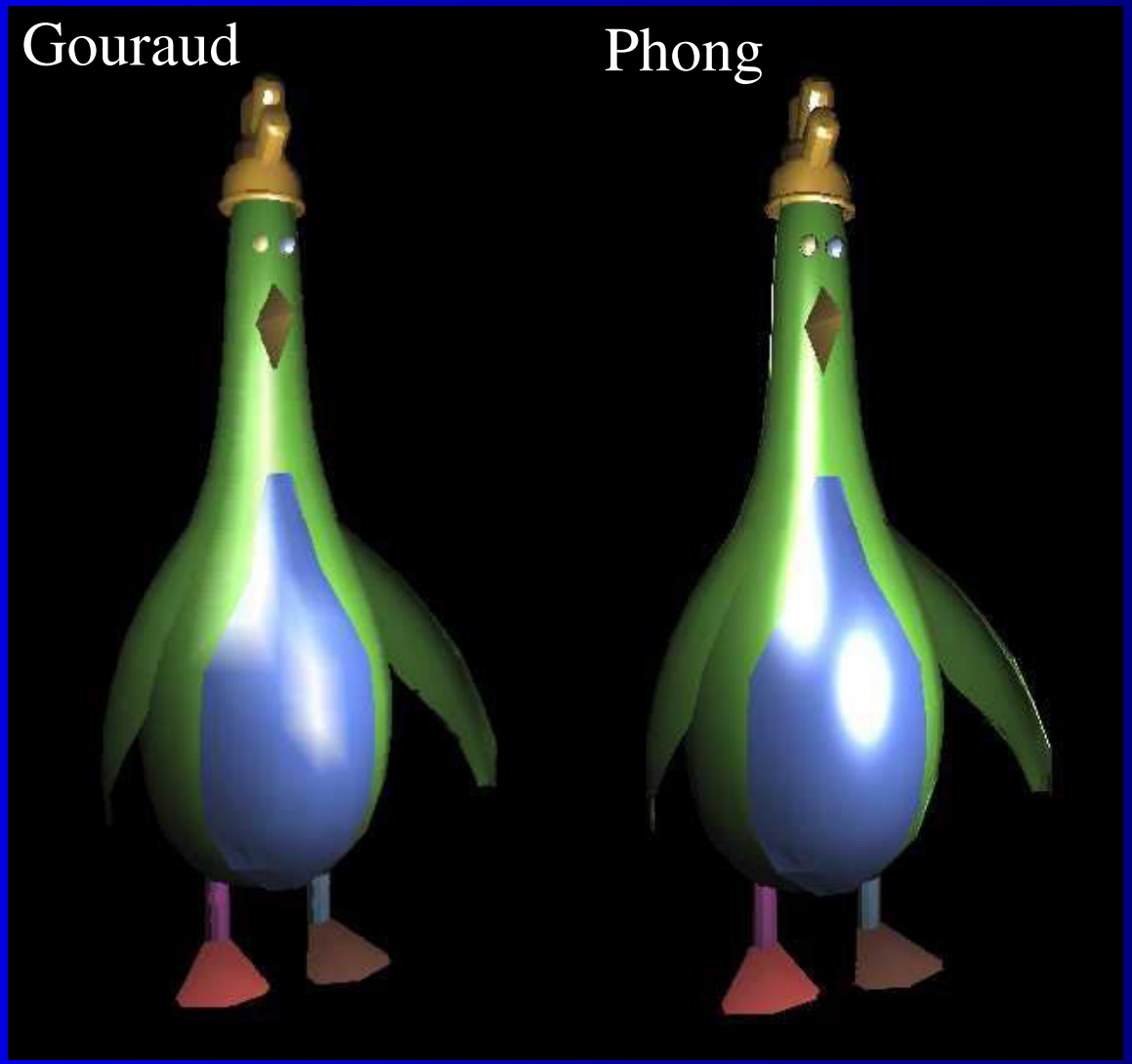
Gouraud helyett Phong árnyalás
Per-vertex helyett per-pixel árnyalás

Gouraud

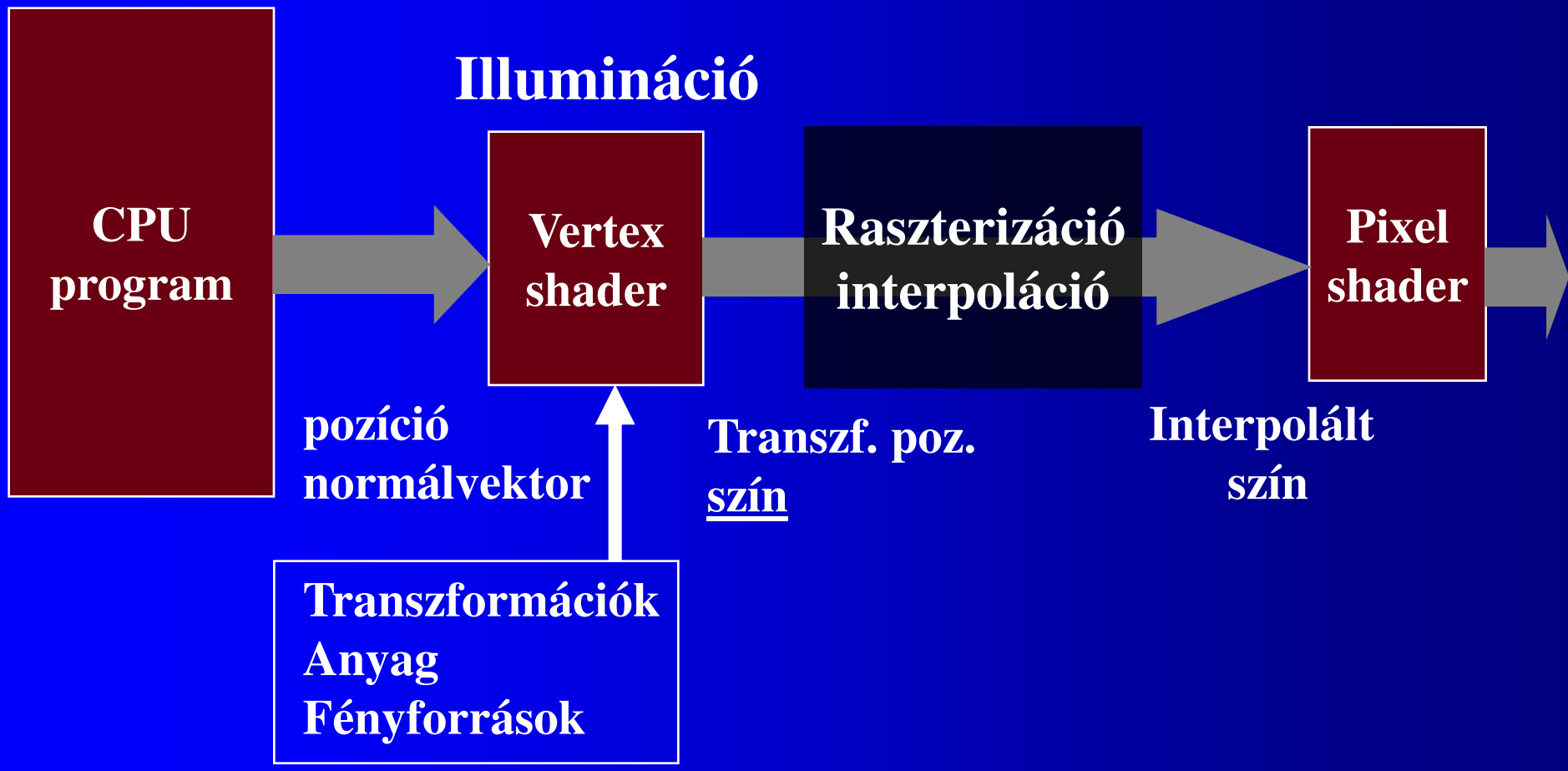


Phong

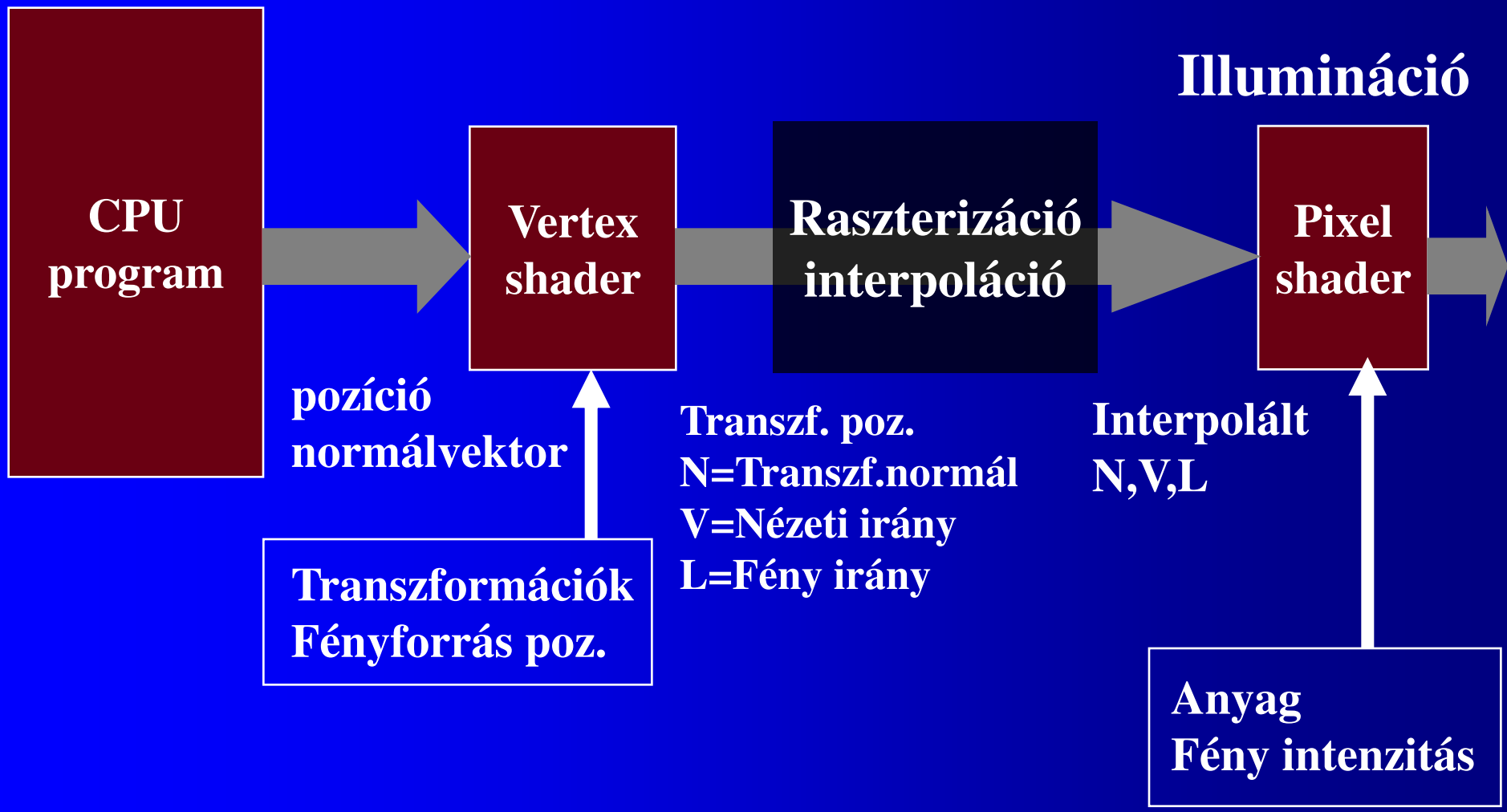




Gouraud (per-vertex) árnyalás



Phong (per-pixel) árnyalás



Programok (lábbal hajtós megoldás)

- **.cpp CPU program:**
  - Shader environment létrehozás
  - GPU képesség beállítás (profile)
  - Vertex/fragment program betöltés és fordítás: CREATE
  - Vertex/fragment program átadás a GPU-nak: LOAD
  - Vertex/fragment program kiválasztás: BIND
  - Uniform vertex/fragment input változó létrehozás

  Inicializálás

  Display

  - Uniform vertex/fragment változó értékadás
  - Változó input változó értékadás (glVertex, glColor, glTexCoord)

- **.cg vertex program**
  - Fragment program változó input + homogén pozíció
- **.cg fragment program**
  - Szín kimenet

CPU program - Inicializálás

```
...
#include <Cg/cgGL.h>                                  // Cg függvények

CGparameter LightPos, LightInt, Shine, Ks, Kd;  // uniform par a CPU-n

int main( ) {
   CGcontext shaderContext = cgCreateContext( );  // árnyaló kontextus
```

Vertex shader betöltés

```
   CGprofile vertexProf = cgGLGetLatestProfile(CG_GL_VERTEX);
   cgGLEnableProfile(vertexProf);

   CGprogram vertexProgram = cgCreateProgramFromFile(
                                        shaderContext,
                                        CG_SOURCE, "vertex.cg",
                                        vertexProf, NULL, NULL);

   cgGLLoadProgram(vertexProgram);          // GPU-ra töltés
   cgGLBindProgram(vertexProgram);          // ez legyen a futó program

      // vertex program uniform paraméterek. CPU-n Lightpos; GPU-n clightpos
   LightPos = cgGetNamedParameter(vertexProgram, "clightpos");
```

Fragmens árnyaló betöltés

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

CGprogram fragmentProgram = cgCreateProgramFromFile(
                                    shaderContext,
                                    CG_SOURCE, "fragment.cg",
                                    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");
```

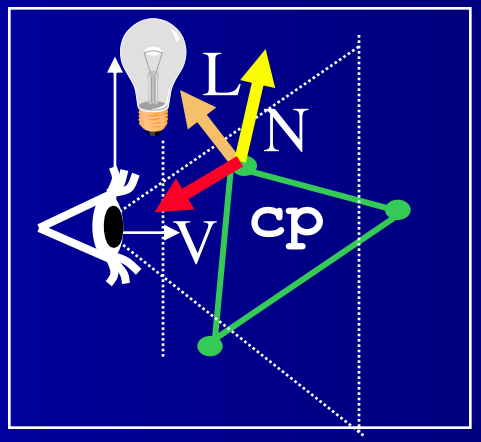
… OpenGL inicializálás

CPU program - OpenGL display

```
void Display( ) {
    glLoadIdentity();  // állapot (implicit uniform) paraméterek beállítása
    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

                        // változó paraméterek, a PASS
    glBegin( GL_TRIANGLES );
    for( … ) {
        glNormal3f(nx, ny, nz); // NORMAL regiszter
        glVertex3f(x, y, z);    // POSITION regiszter + trigger
    }
    glEnd();
}
```

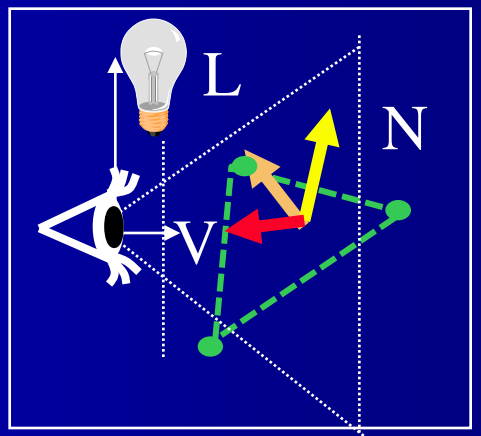
Phong árnyalás: vertex shader

```
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   clightpos, // fényforrás poz. kamerak-ban
        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 = clightpos.xyz * cp.w - cp.xyz * clightpos.w;
        cview  = -cp.xyz;
}
```



Phong árnyalás: fragment shader

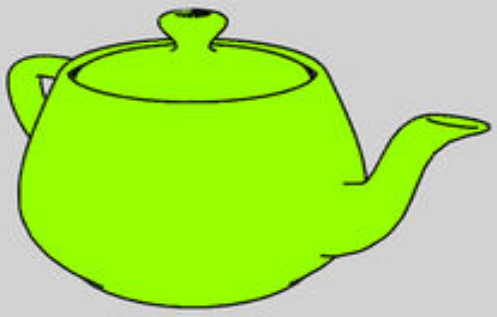
```
void main(    in float3 N          : TEXCOORD0,
              in float3 V          : TEXCOORD1,
              in float3 L          : TEXCOORD2,
              uniform float3 lightint,
              uniform float shininess,
              uniform float3 kd,
              uniform float3 ks,
              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;
}
```







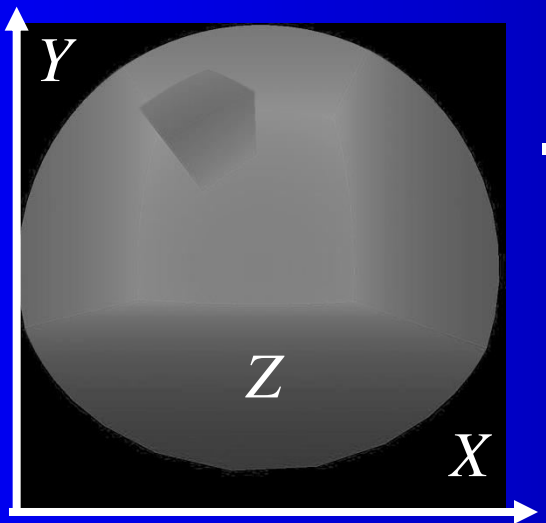
NPR

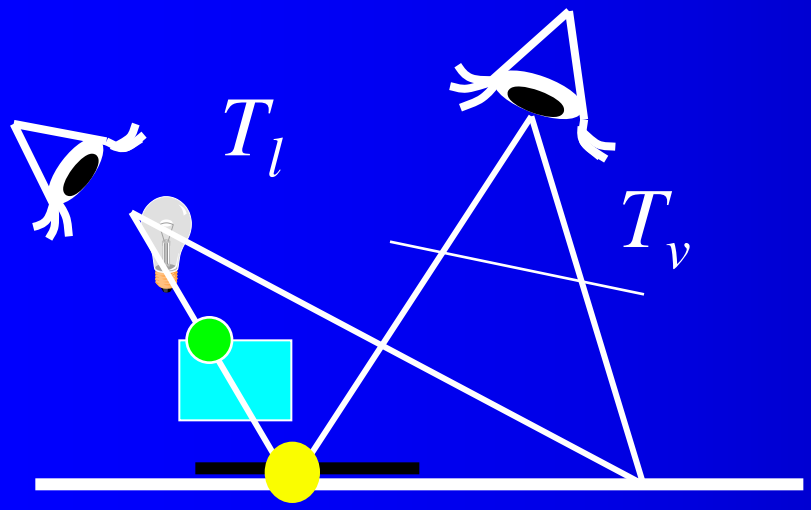


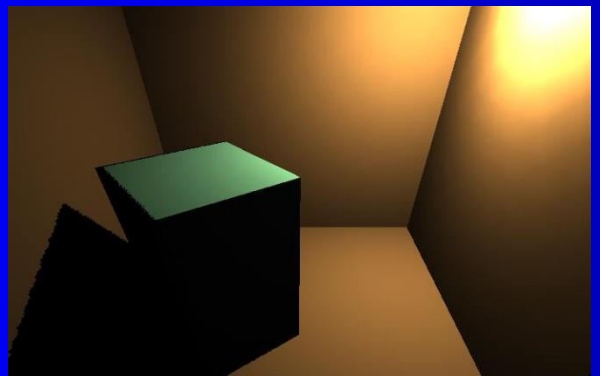


Lichthof Productions

Árnyék térképek







1. pass:

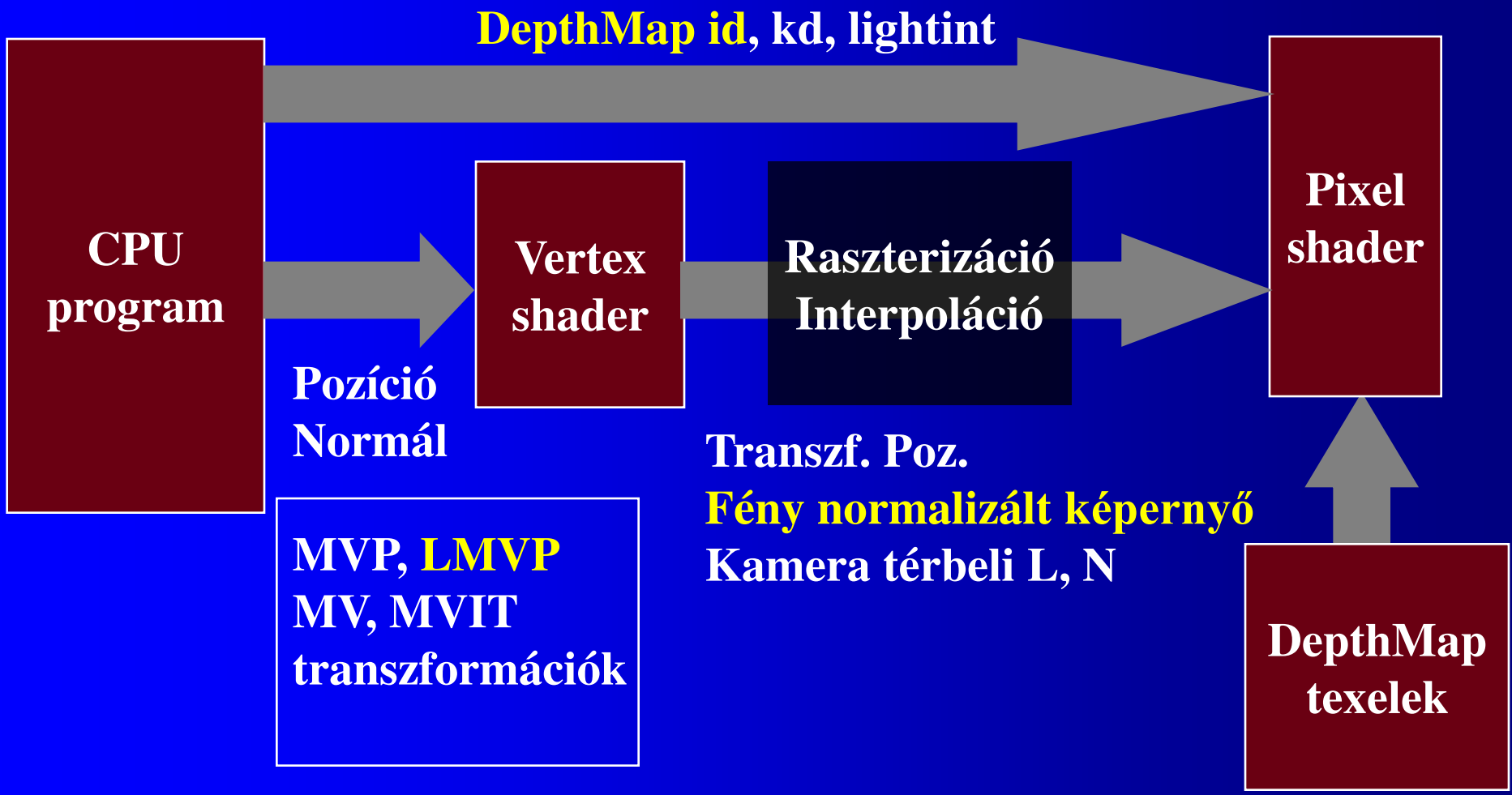
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(lightx, lighty, lightz,
lookatx, lookaty, lookatz,
lupx, lupy, lupz);
Modellezési transzformáció
Képszintézis

Z-buffer -> textúra másolás (v. közv)

2. pass:

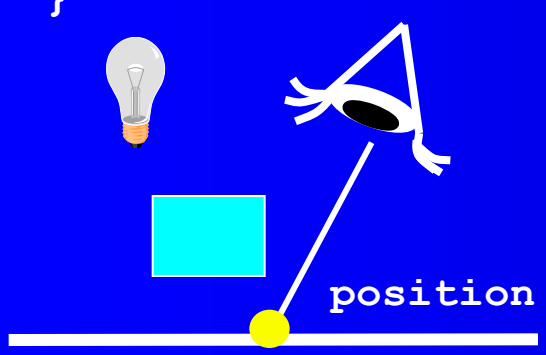
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez,
lookatx, lookaty, lookatz,
upx, upy, upz);
Modellezési transzformáció
Képszintézis **+ árnyék teszt**

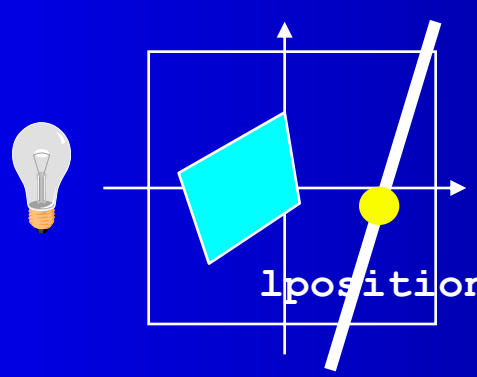
Képszintézis + árnyékteszt

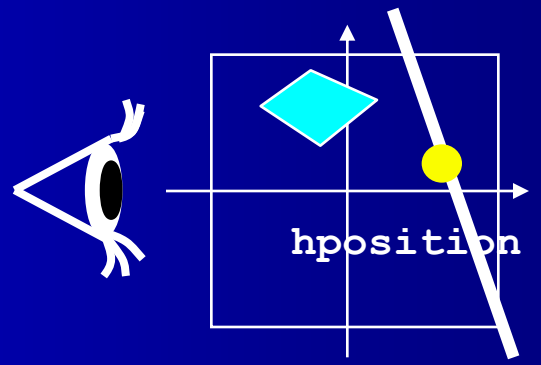


Vertex shader

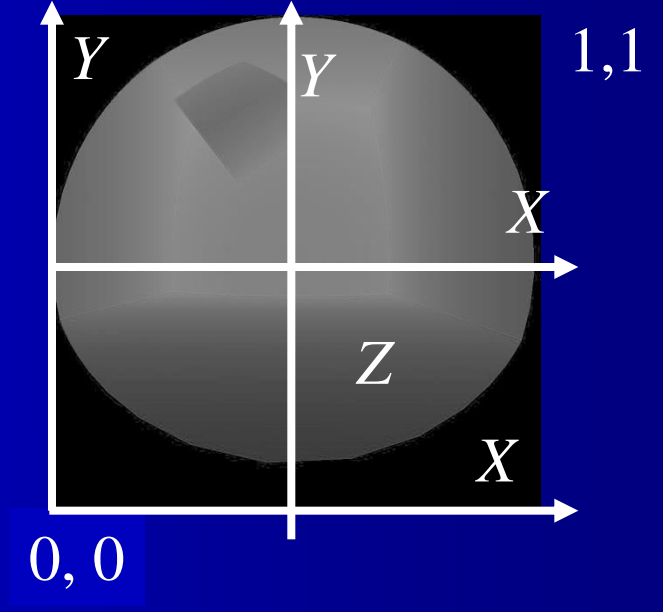
```
void main(    in float4 position     : POSITION,
              in float4 normal       : NORMAL,
              uniform float4x4 MVP, LMVP, MV, MVIT, ☹
              uniform float4 clightpos,
              out float4 hPosition   : POSITION,
              out float4 lPosition   : TEXCOORD0,
              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

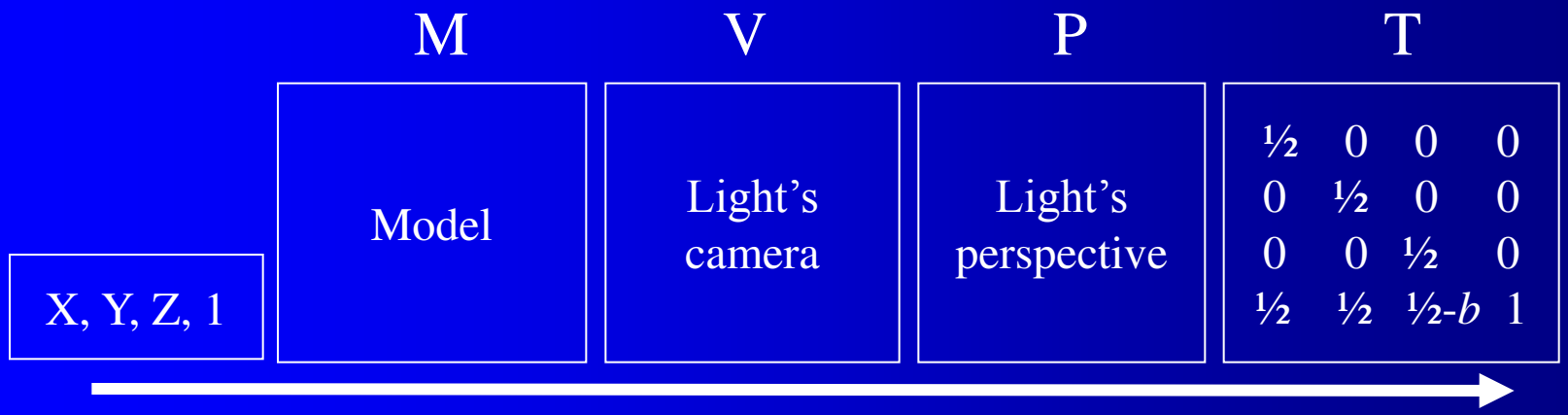



```
void main(
  in float4 lPosition : TEXCOORD0,
  in float3 N         : TEXCOORD1,
  in float3 L         : TEXCOORD2,
  uniform float4 lightint, kd, ☹
  uniform sampler2D depthMap,
  uniform float bias,
  out float4 ocolor   : COLOR )
{
  float3 lPosCartesian = lPosition.xyz/lPosition.w;
  float2 texcoord;
  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) { // == kéne
       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

Projektív textúrázás mátrixa

```
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;
```

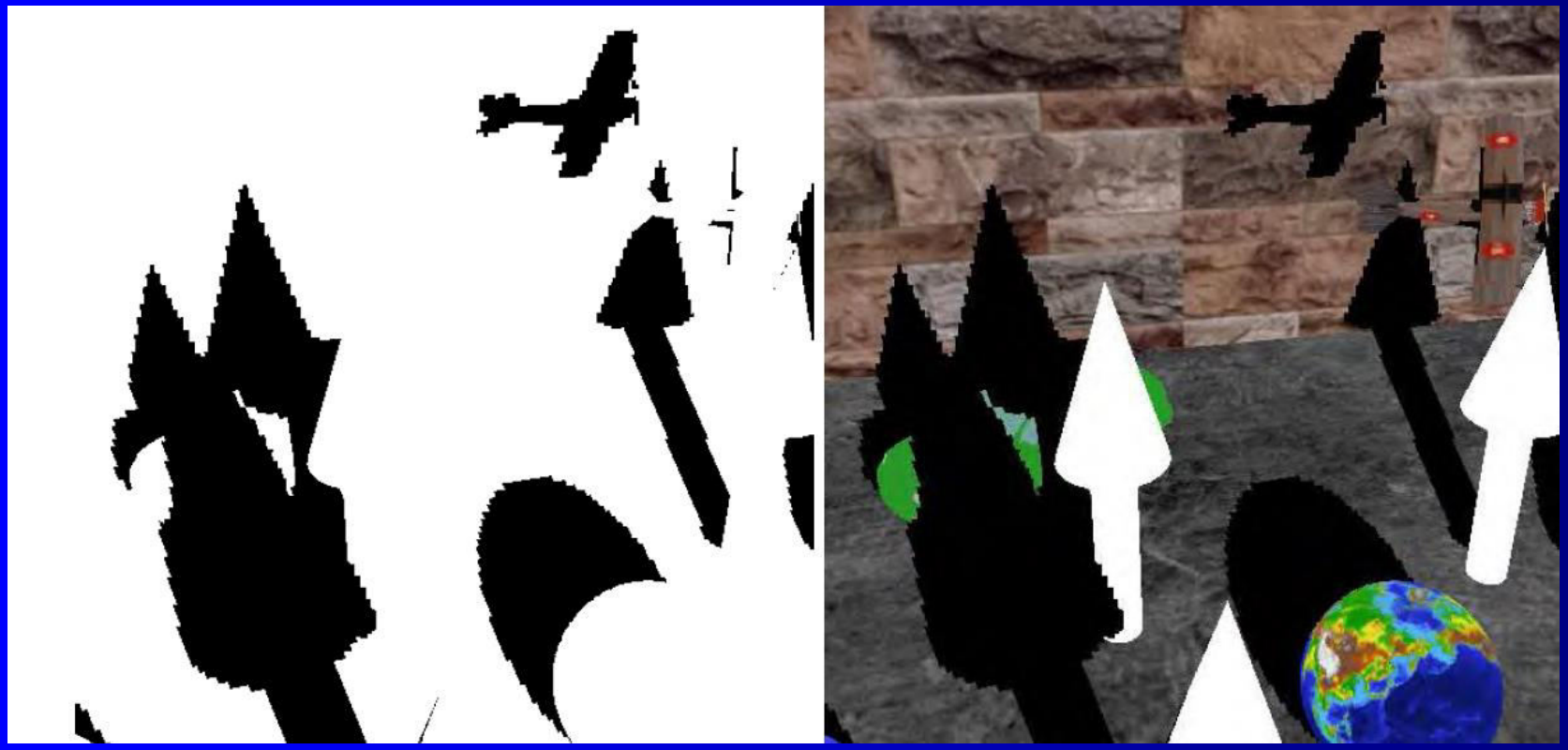


Egyszerűbb árnyékteszt

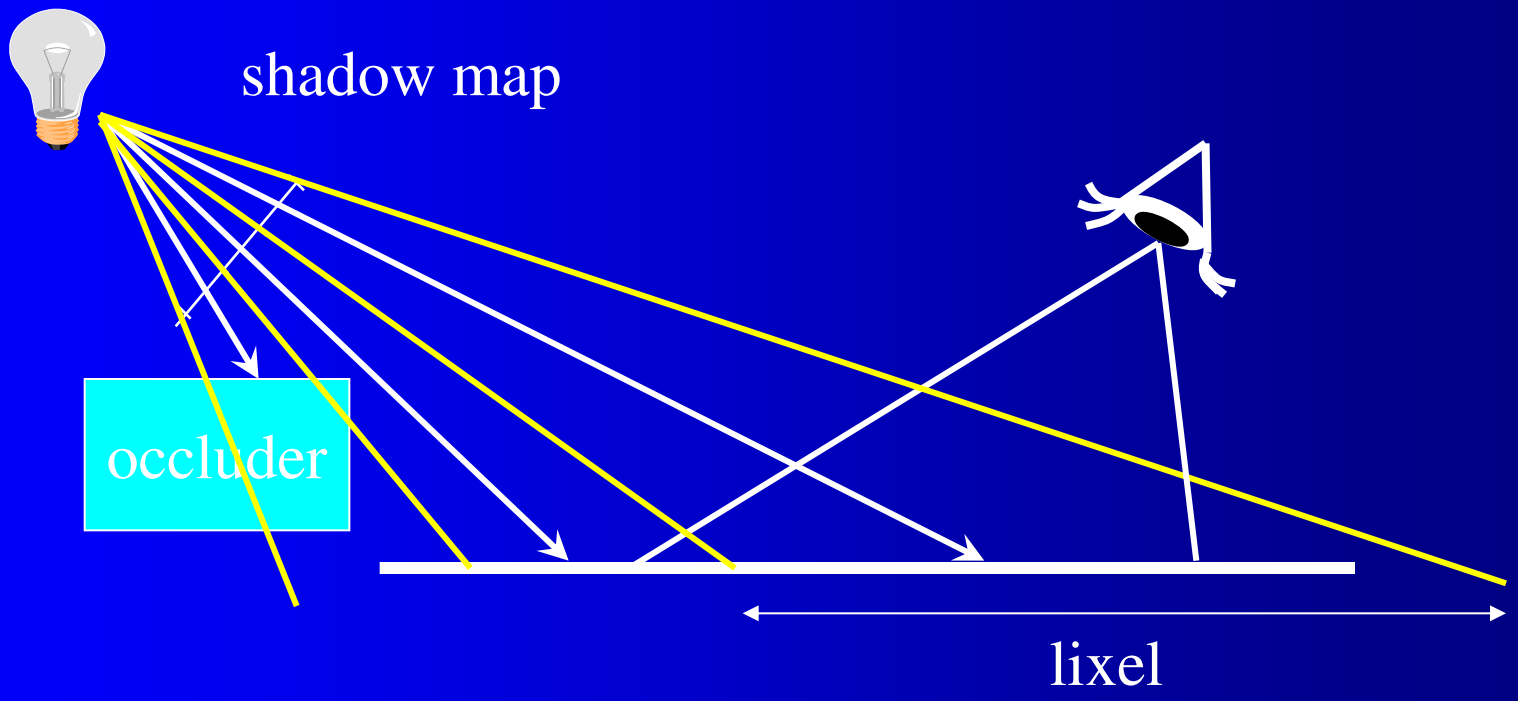
```
void SM_VS(     in float4 position     : POSITION,
                in float4 normal       : NORMAL,
                uniform float4x4 MVP, LMVPT, MV, MVIT, ☹
                uniform float4 clightpos,
                out float4 hPosition   : POSITION,
                out float4 tPosition   : TEXCOORD0,
                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

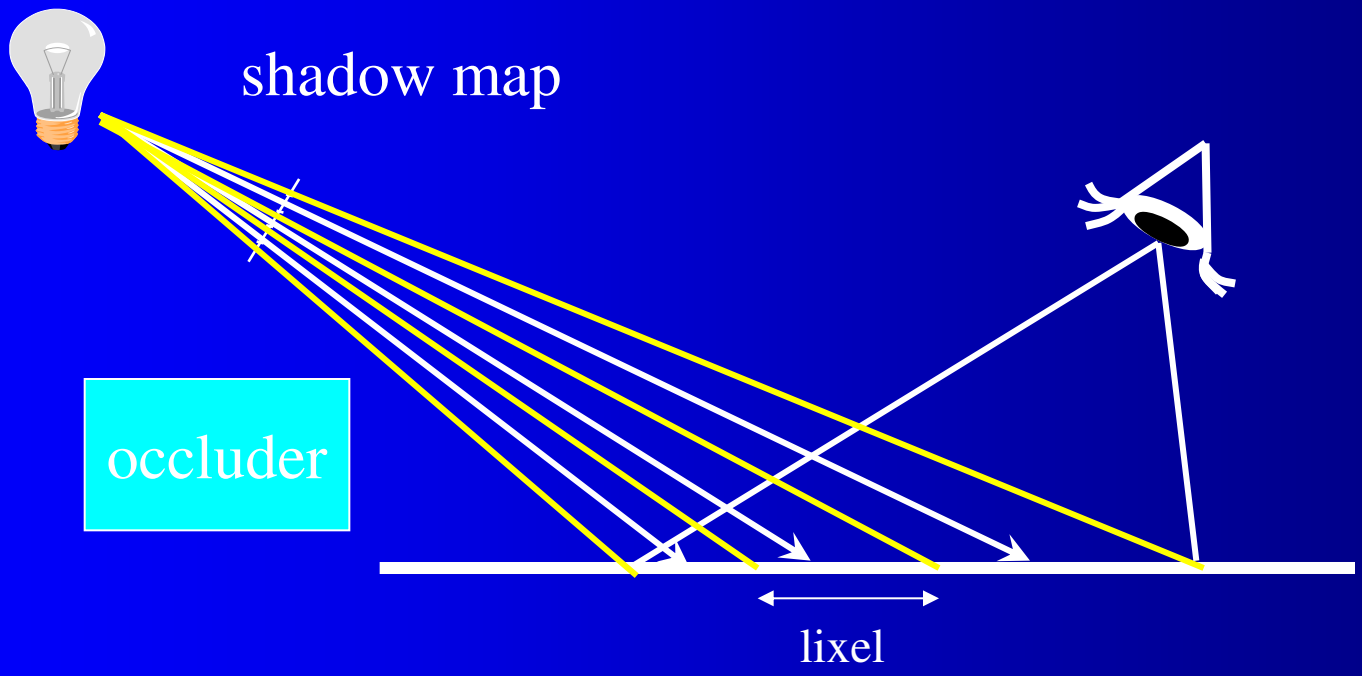
Árnyéktérkép aliasing



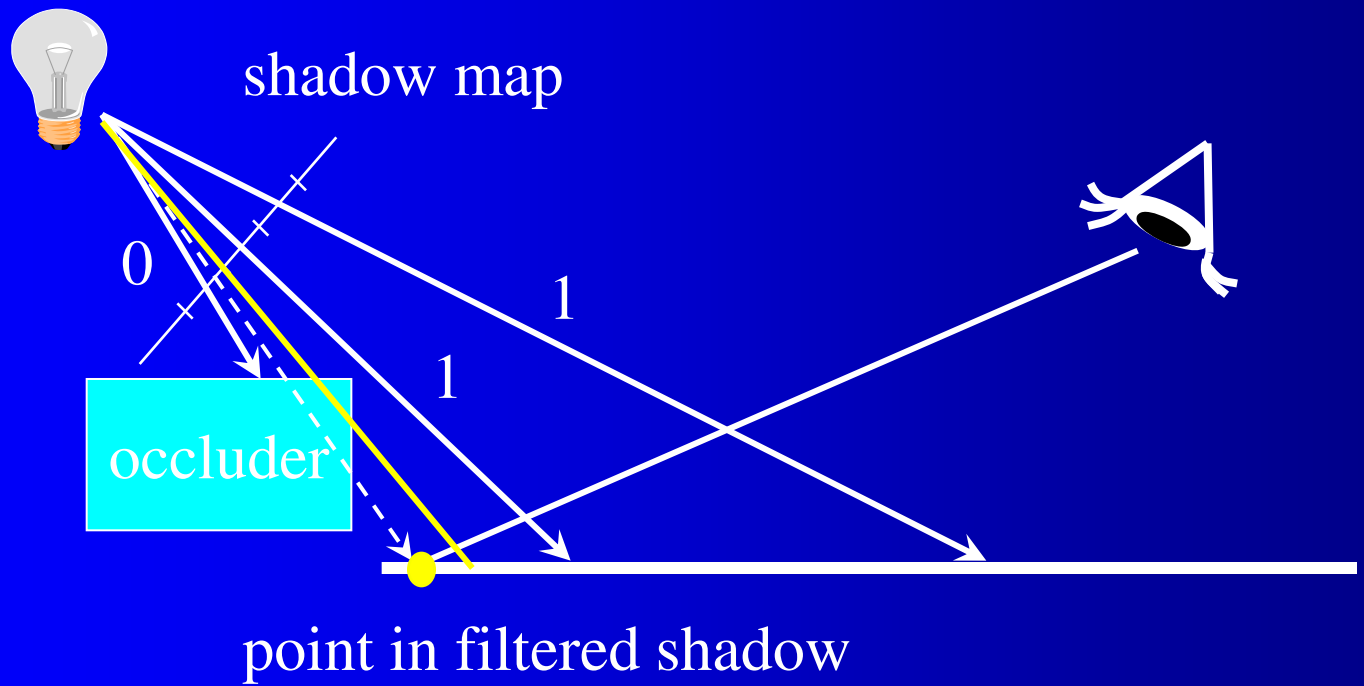
Árnyéktérkép aliasing



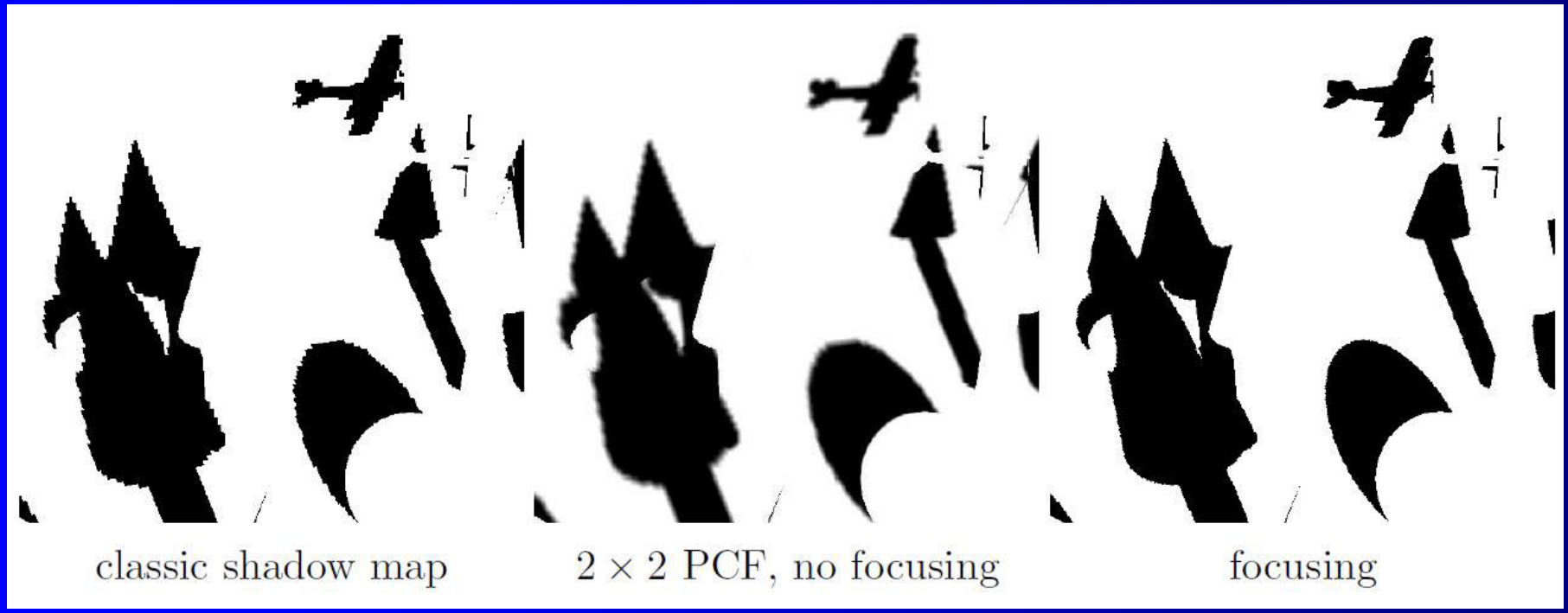
Árnyéktérkép fókuszálás



Percentage Closer Filtering



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



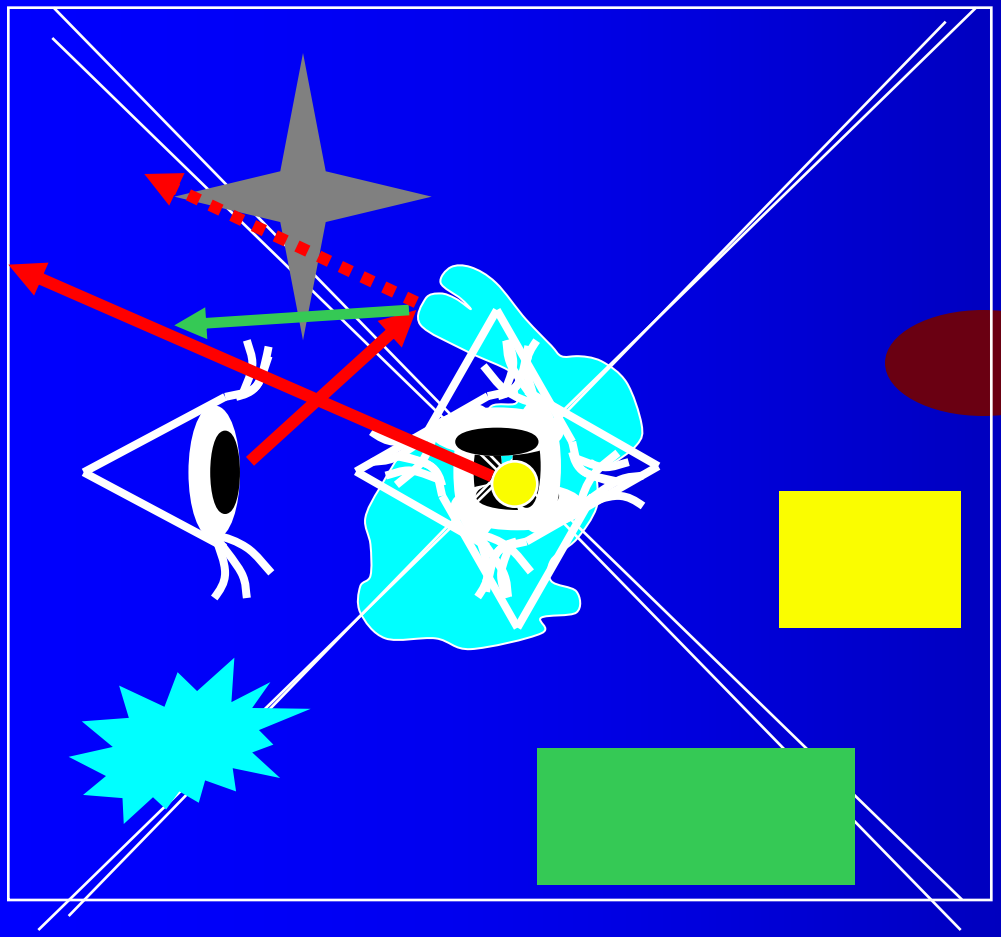


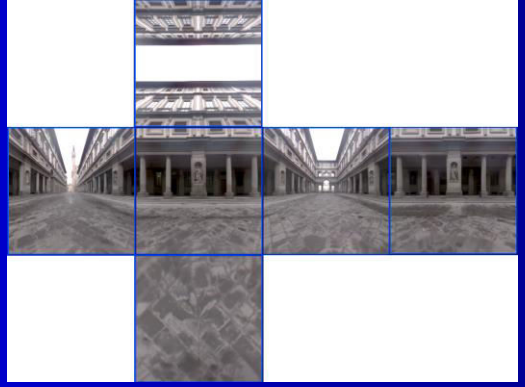
Classic shadow map



Variance shadow map

Visszaverődés és törés



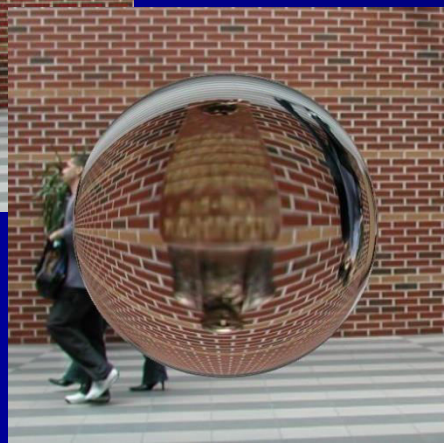




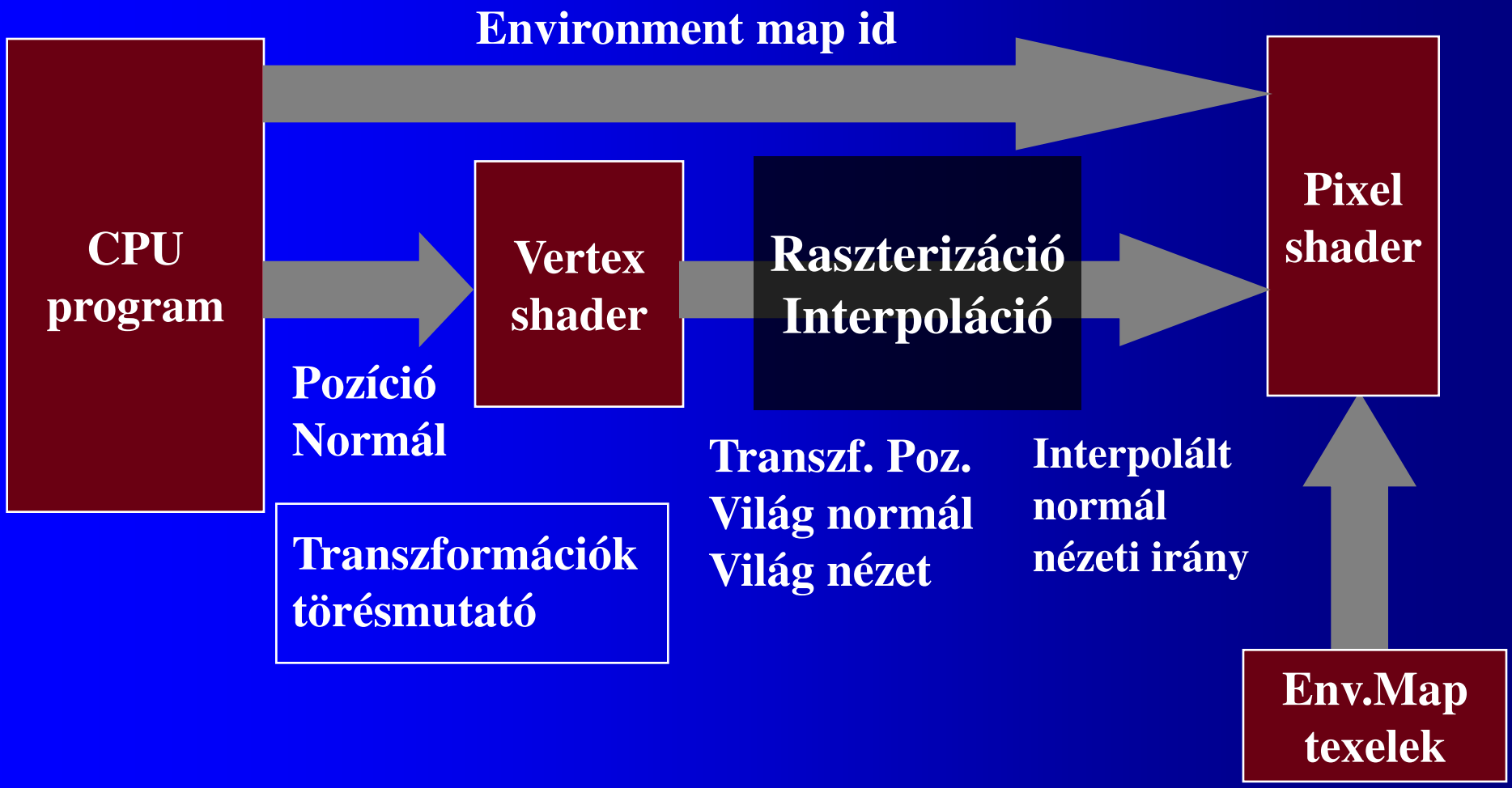








Visszaverődés/törés számítás: a sima objektum feldolgozása

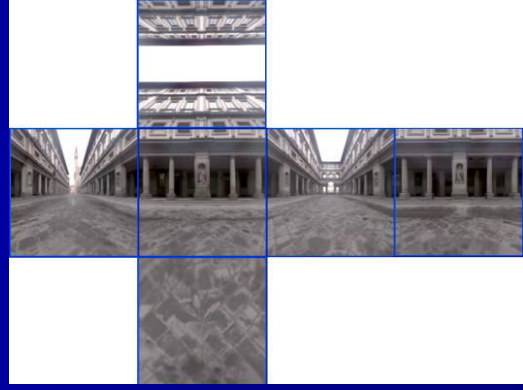


Sima objektum vertex shader

```
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;
}
```

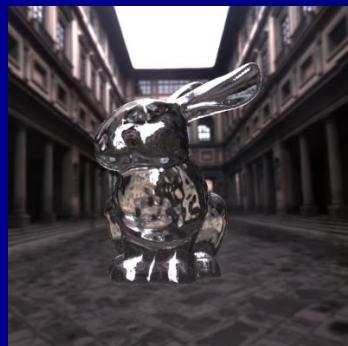
Sima objektum fragment shader

```
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;
}
```



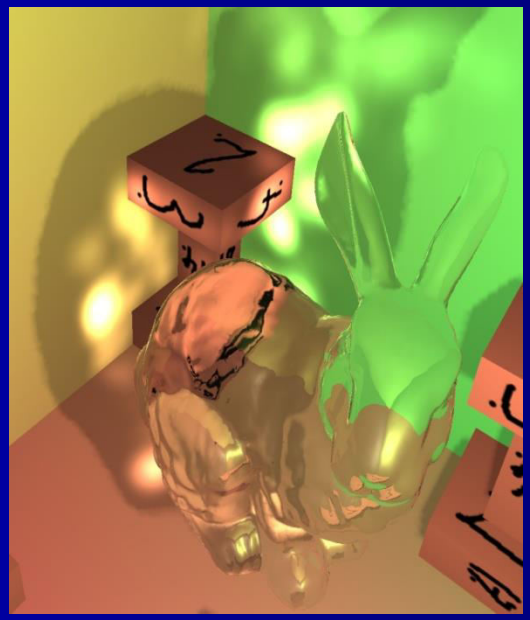




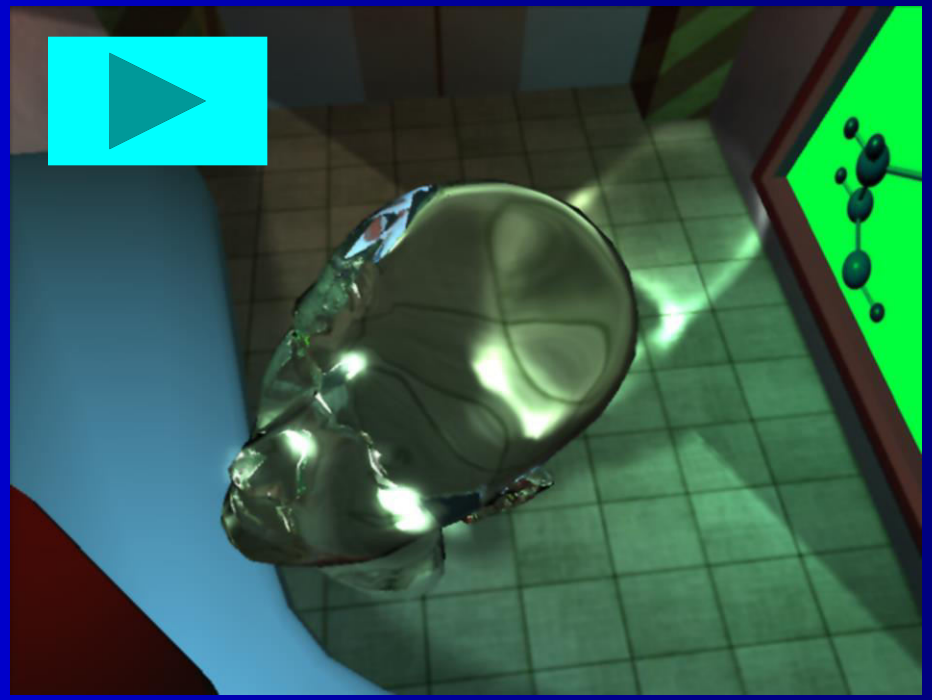


Lokalizált környezet leképzés

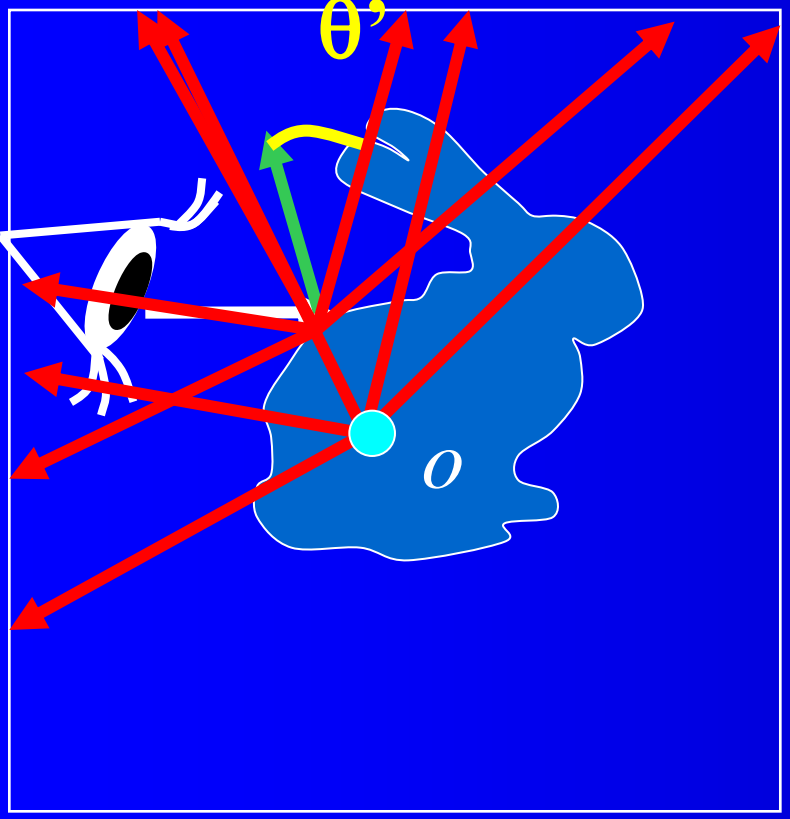




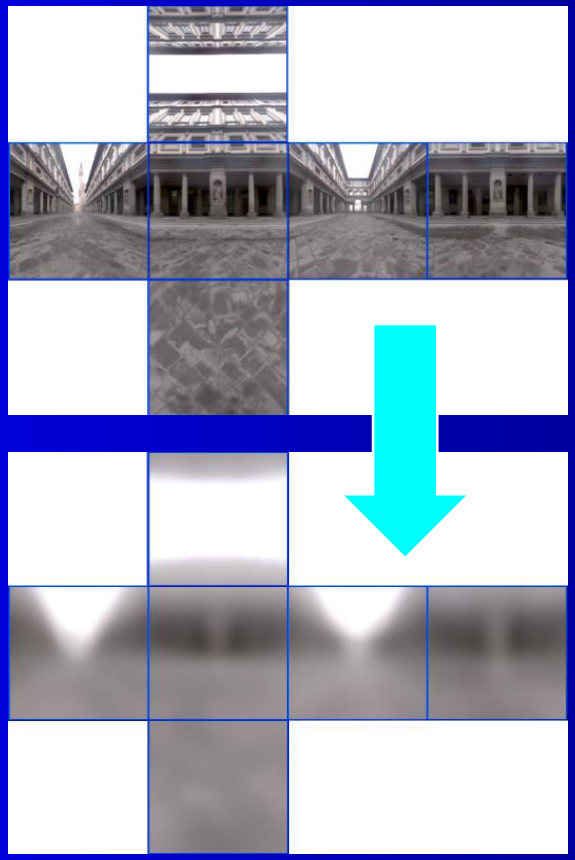




Diffúz indirekt illumináció

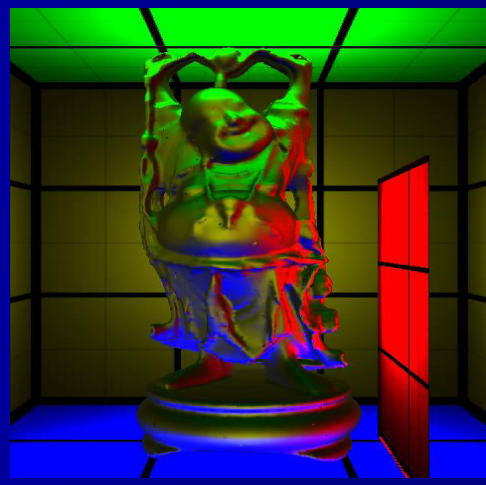


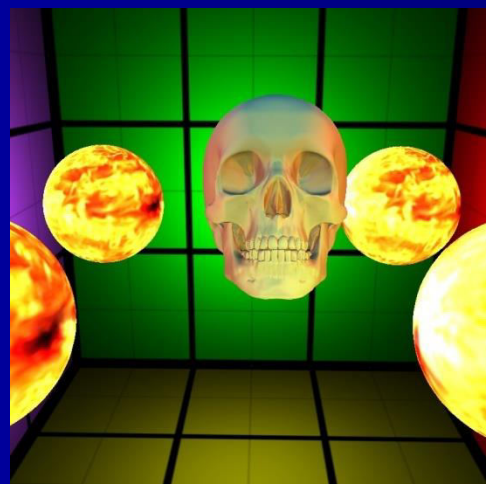
$$k_d \int_{\Omega} L^{in}(o,\omega_i)\cos\theta' d\omega_i$$



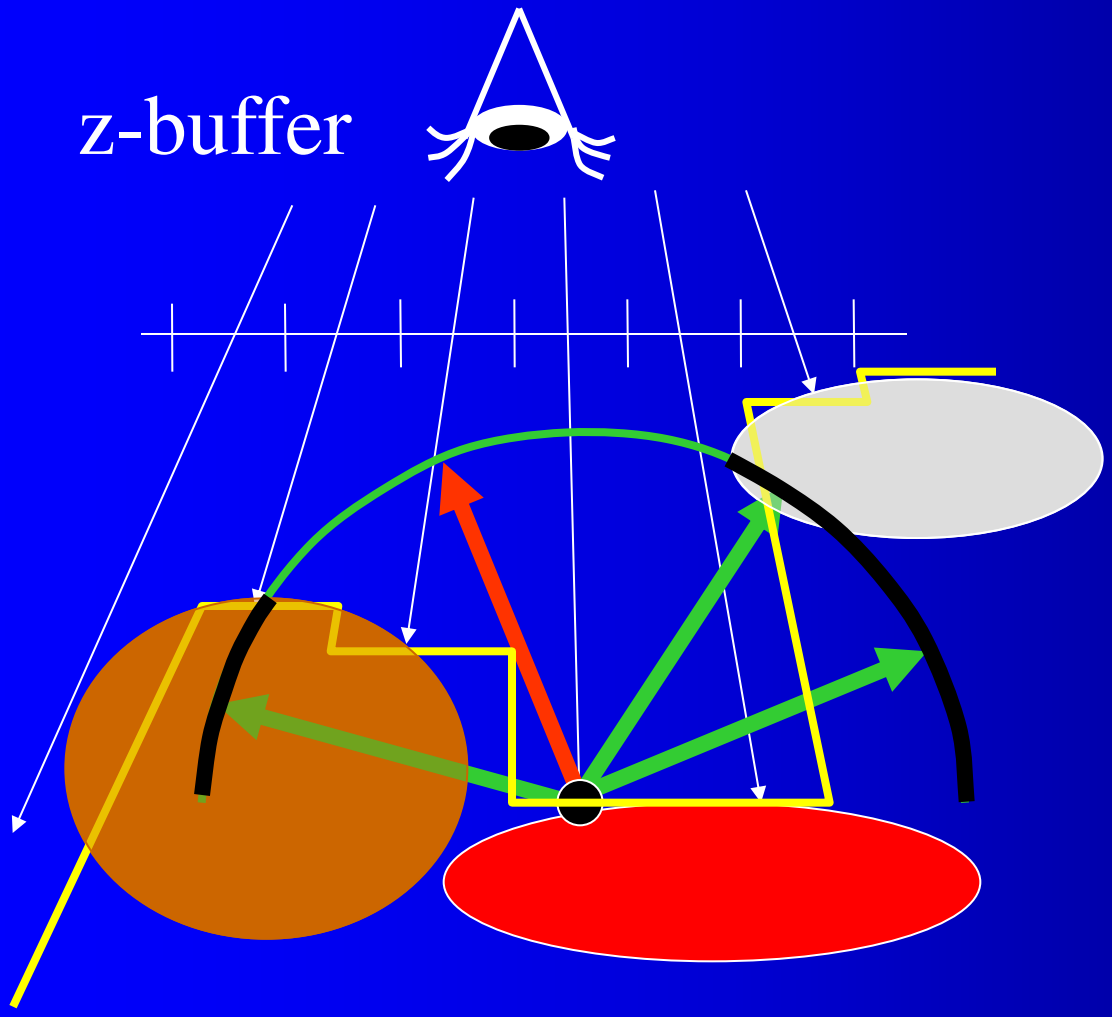




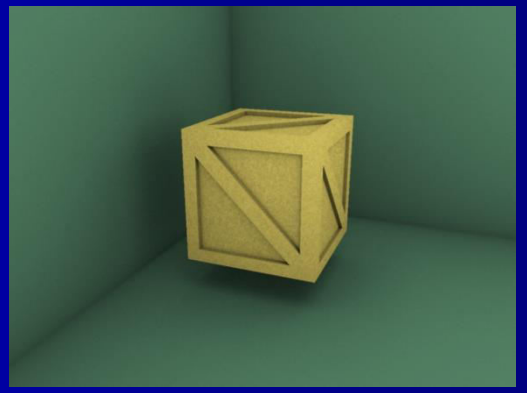




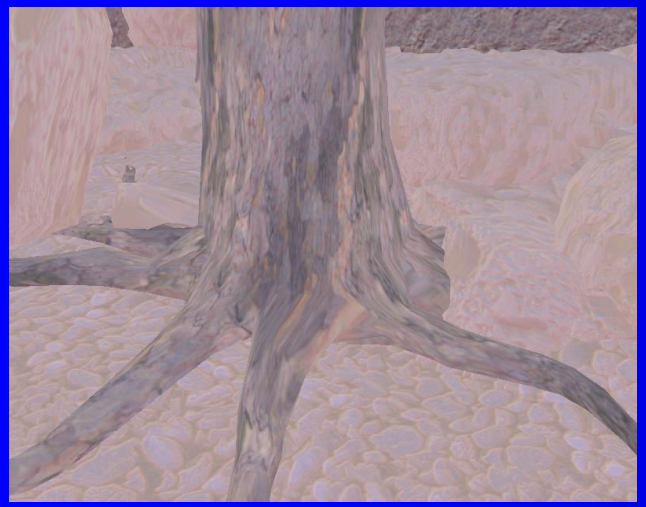
Screen-Space Ambient Occlusion

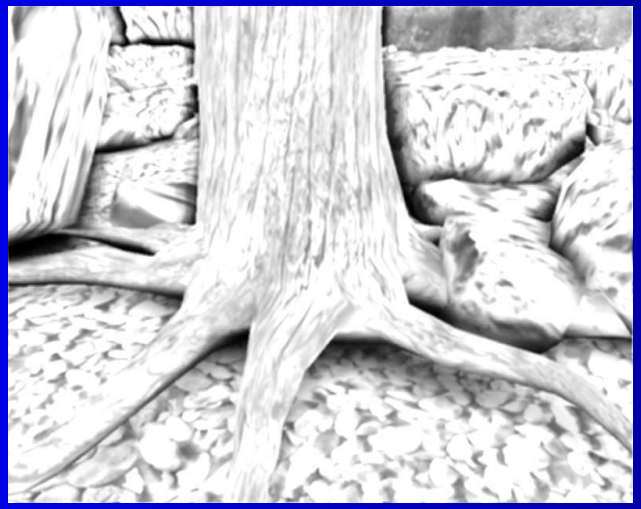


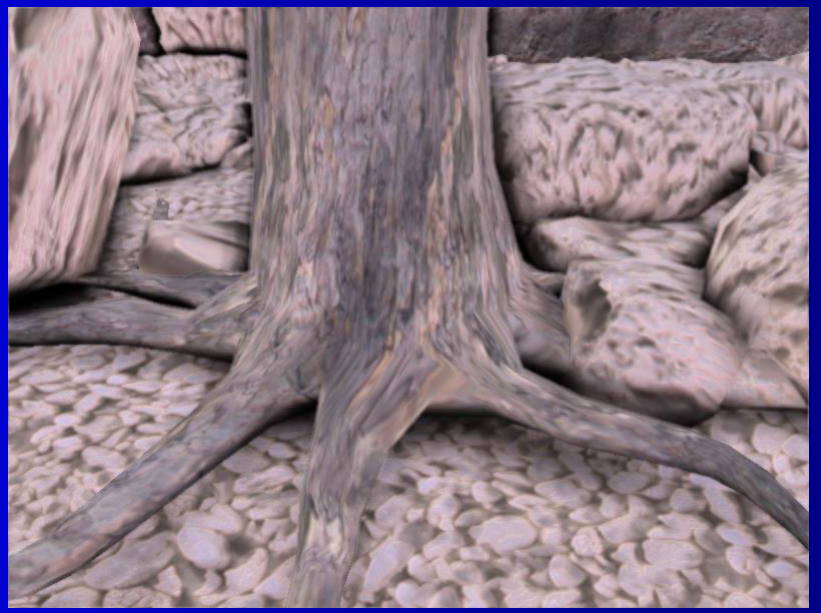




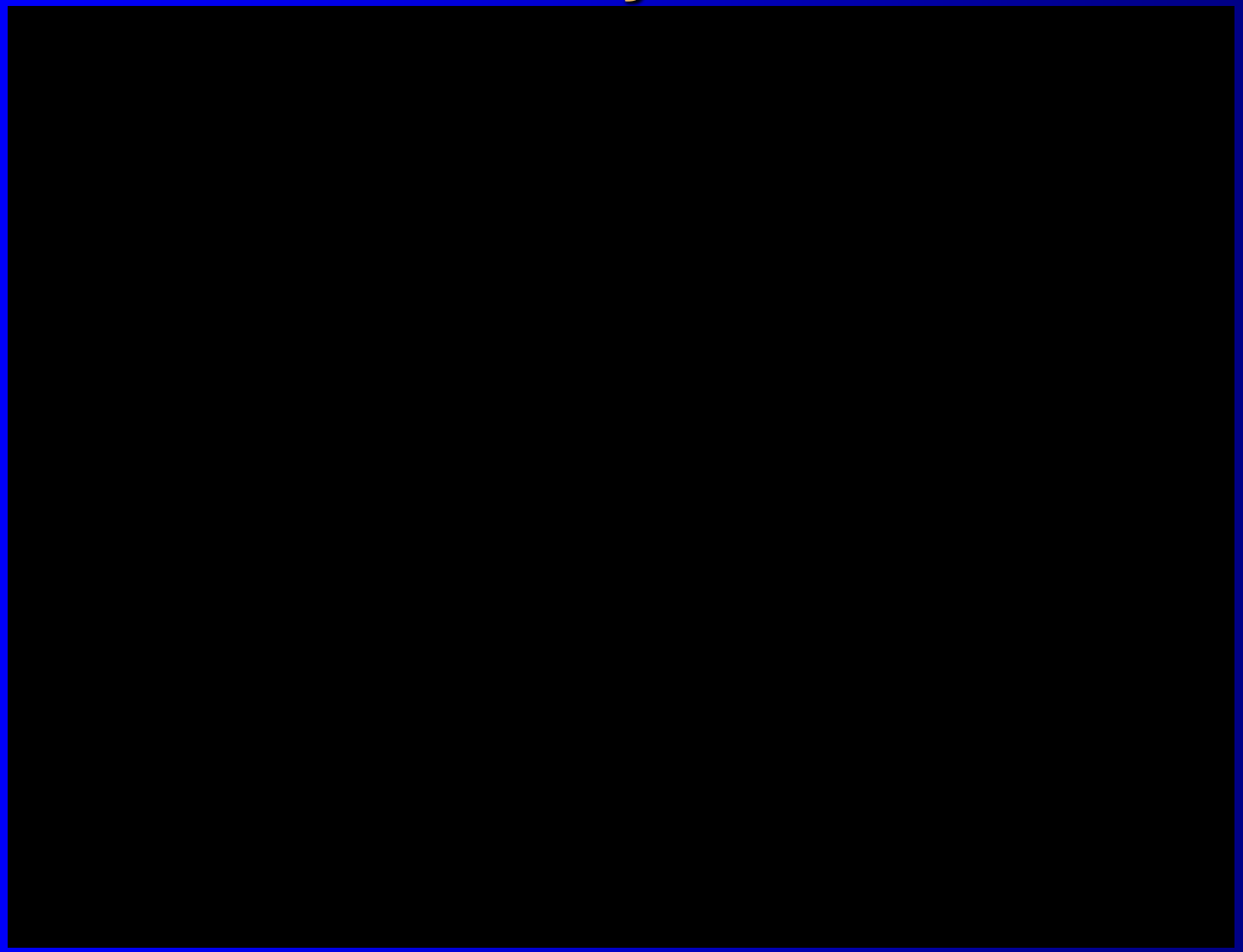
Ambient Occlusion



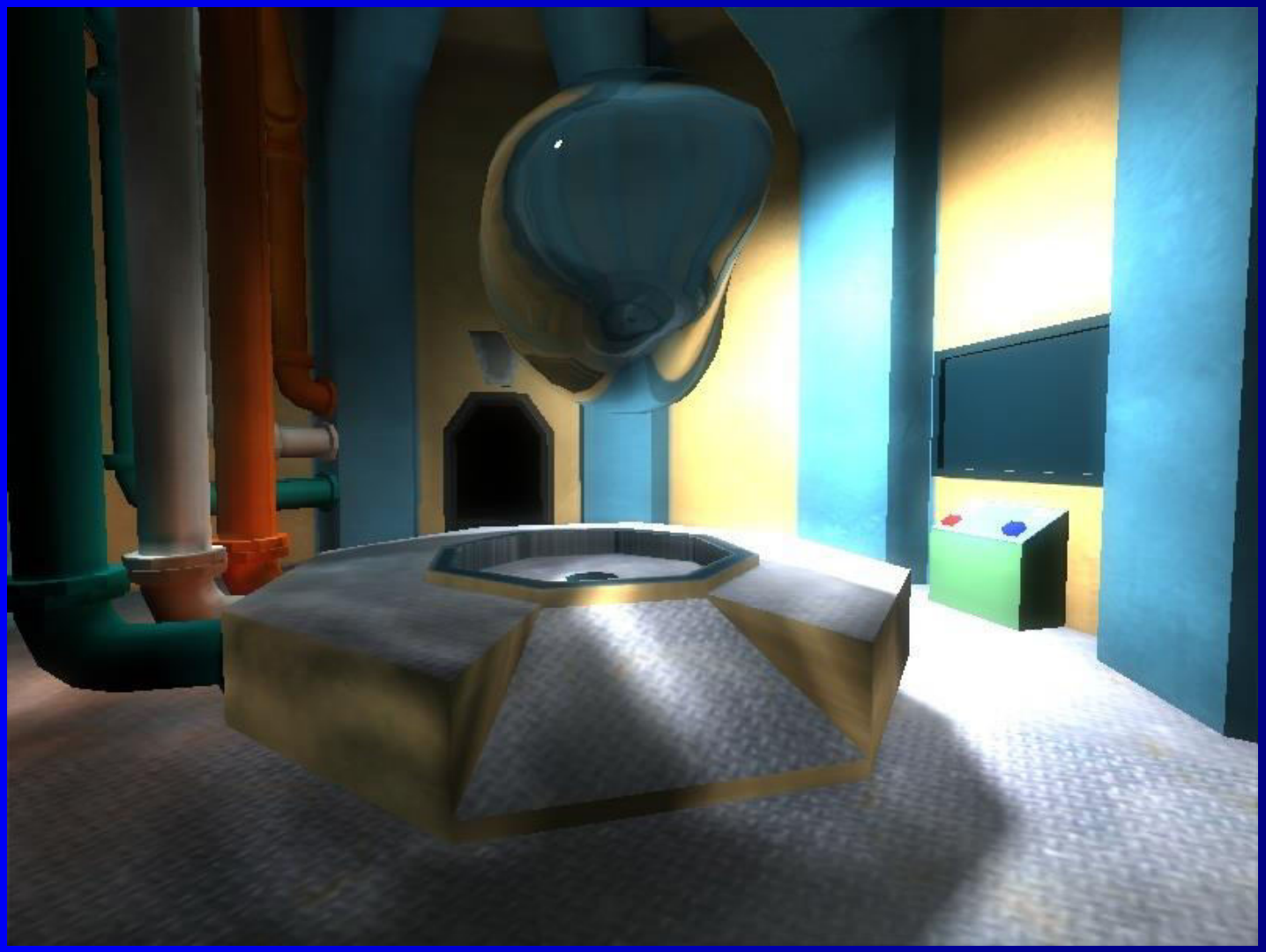




Autós játék



Űrállomás



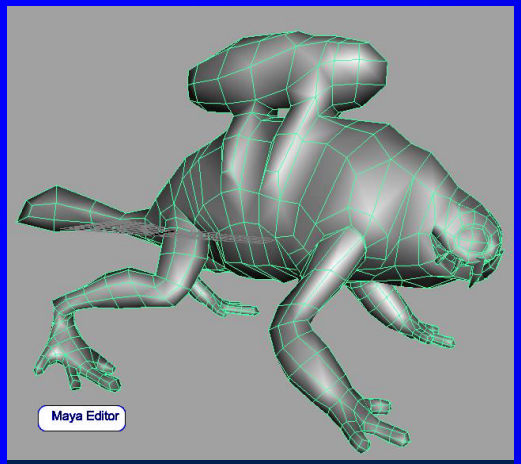
Moria



Óceán



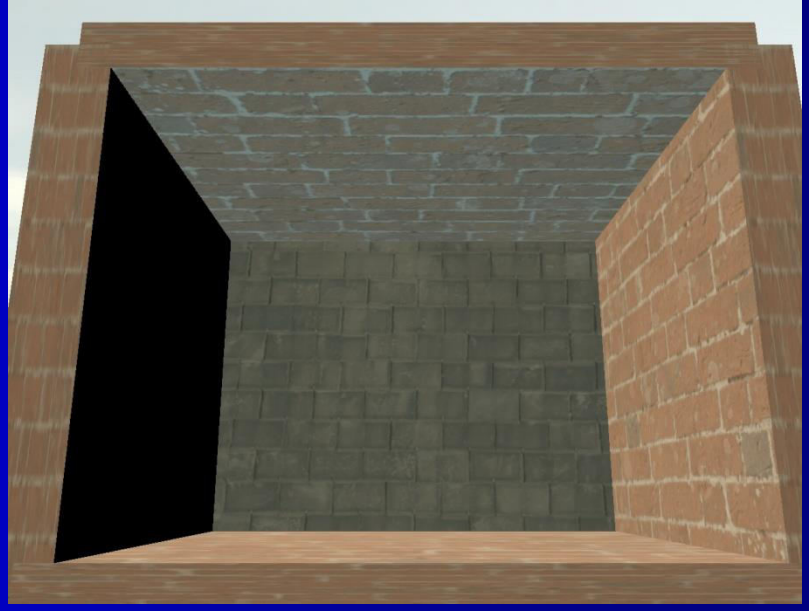
Geometria árnyaló





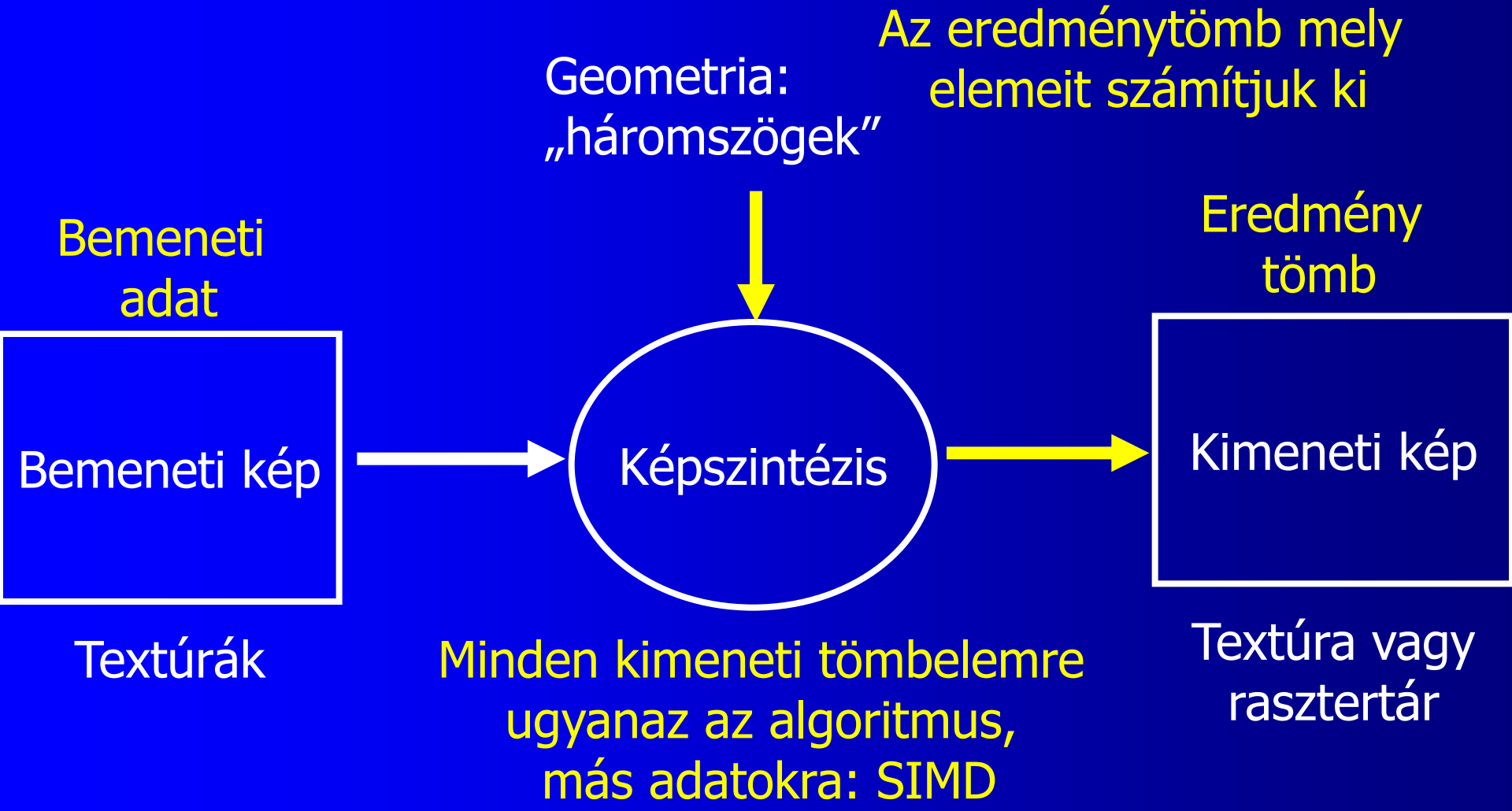
Catmull-Clark subdivision

Procedurális geometria





GPGPU


Az eredménytömb mely elemeit számítjuk ki
Geometria: „háromszögek”
Bemeneti adat
Eredmény tömb
Bemeneti kép
Képszintézis
Kimeneti kép
Textúrák
Minden kimeneti tömbelemre ugyanaz az algoritmus, más adatokra: SIMD
Textúra vagy rasztertár

„Teljes képernyős" téglalap (CPU):

```
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;
}
```

Fragment shader (Cg):

```
void main ( in float2    Tex : TEXCOORD0 ,
            uniform sampler2D bemAdat,
            out float4 result : COLOR ) {
   result = Bemeneti képből számított adat a
            tex2D(bemAdat, f(Tex)) alapján;
}
```

Melyik kimeneti tömbelemet számítjuk

Bemeneti adat

Eredmény tömb

SIMD, i.e. vektorprocesszálás

É a
kép ban





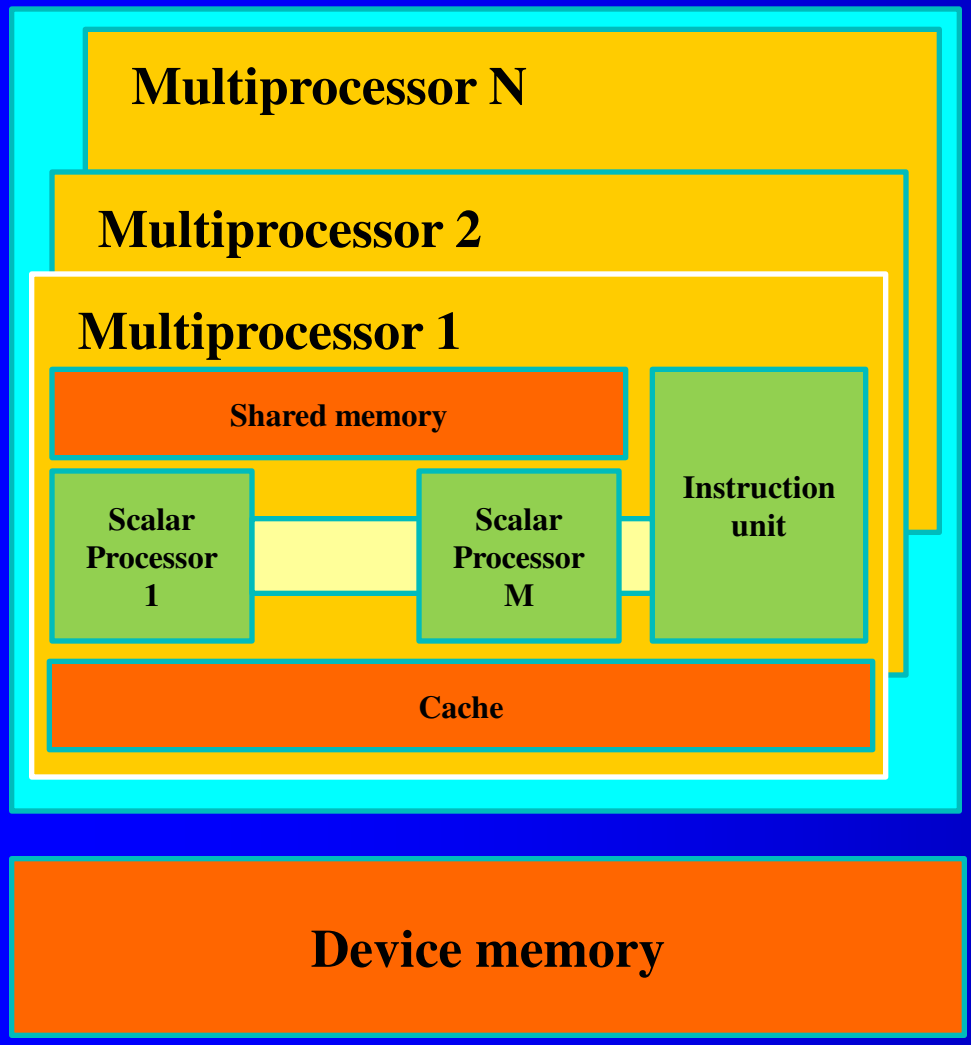






void r
in ,
u
o
{
fl);
fl);
fl Tex Tex–du))/2/du;
fl Tex Tex–dv))/2/dv;
fl gra
r abs
}

CUDA (OpenCL)



GPU

Kernel program:

Threads

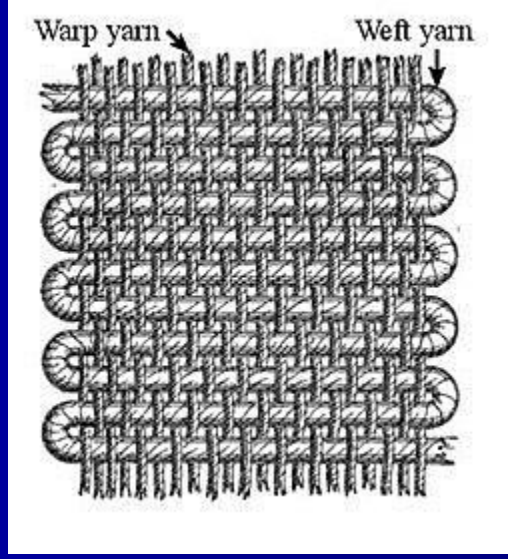
block, block,

Warp, Warp, …

SIMD

Shared memory

SIMD execution



Két N elemű vektor összeadása

A GPU-n fut, de a CPU-ról is hívható

```
#include <cuda.h>

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

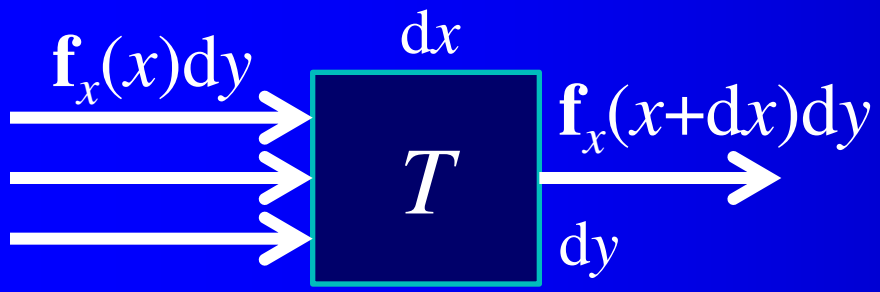
0 ,…, gridDim.x-1

0 ,…, blockDim.x-1

```
const int N = 100000;
const int Nb = N * sizeof(float);
float Ccpu[N], Acpu[N], Bcpu[N];

int main ( ) {
    … // Acpu és Bcpu tömbök feltöltése
    float *Agpu, *Bgpu, *Cgpu;
    cudaMalloc(&Agpu, Nb); cudaMalloc(&Bgpu, Nb); cudaMalloc(&Cgpu, Nb);
    cudaMemcpy(Agpu, Acpu, Nb, cudaMemcpyHostToDevice);
    cudaMemcpy(Bgpu, Bcpu, Nb, cudaMemcpyHostToDevice);

    int blockDim = 256;          // #threads egy blokkban: 128, 256, 512
    int gridDim = (N + blockDim – 1) / blockDim;   // #blocks

    AddVectorGPU<<<gridDim, blockDim>>>(Cgpu, Agpu, Bgpu, N);

    cudaMemcpy(Ccpu, Cgpu, Nb, cudaMemcpyDeviceToHost);
    cudaFree(Agpu); cudaFree(Bgpu); cudaFree(Cgpu);
    … // A Ccpu használata
}
```

2D hőáramlás, diffúzió

- Anyag-energia megőrzés



$$\frac{\partial T}{\partial t} = -k\left(\frac{\partial \mathbf{f}_x}{\partial x} + \frac{\partial \mathbf{f}_y}{\partial y}\right) = -k\nabla\cdot\mathbf{f}$$

- Hőáramot a hőmérséklet különbség hozza létre

$$\mathbf{f} = (\mathbf{f}_x, \mathbf{f}_y) = -\rho\left(\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y}\right) = -\rho\nabla T$$

$g = k\rho$

$$\frac{\partial T}{\partial t} = g\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = g\nabla^2 T$$

$$\frac{\partial^2 T}{\partial x^2}$$

$$\Delta T = g\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)\cdot \Delta t$$

```
__device__ float T(float* array2d,
    return (x<0||x>=N||y<0||y>=N) ?
}
__global
void Diffuse(float* o, float* n, float* b, float g, float dx, fl      oat dt, int N) {
    int x = threadIdx.x, y = blockIdx.x;
    float dTx2 = (T(o, N, x+1, y) – 2* T(o, N, x, y) + T(o, N, x-1,      )/(dx*dx) ;
    float dTy2 = (T(o, N, x, y+1) – 2* T(o, N, x, y) + T(o, N, x, y-1))/(dy*dy) ;
    n[y*N+x] = (T(b, N, x, y)==0) ? T(o, N, x, y) + g * (dTx2 + dTy2) * dt : T(o, N, x, y);
}
```

```
const int N = 128, Nb = N*N*sizeof(float); // number of bytes
float Tcpu[N*N] = /*initial+boundary condition*/ , Bcpu[N*N] = /*1-0  mask*/

int main ( ) {
    float *Tgpu1, *Tgpu2, *Bgpu;
    const float g = 3, dx=1, dy=1, dt = 0.1, tend=10;

    cudaMalloc(&Tgpu1, Nb); cudaMemcpy(Tgpu1, Tcpu, Nb, cudaMemcpyHostToDevice);
    cudaMalloc(&Tgpu2, Nb);
    cudaMalloc(&Bgpu, Nb);  cudaMemcpy(Bgpu, Bcpu, Nb, cudaMemcpyHostToDevice);

    for(float t = 0; t < tend; t += dt) {
        Diffuse<<<N, N>>>(Tgpu1, Tgpu2, Bgpu, g, dx, dy, dt, N);
        Diffuse<<<N, N>>>(Tgpu2, Tgpu1, Bgpu, g, dx, dy, dt, N);
    }

    cudaMemcpy(Tcpu, Tgpu1, nb, cudaMemcpyDeviceToHost);
    cudaFree(Tgpu1); cudaFree(Tgpu2);
}
```



Mikor jó?

- Párhuzamos algoritmus (>10k szál):
  A probléma elemzésével kezdődik!
- Gyűjtő típusú algoritmus (nincsenek írási memóriaütközések)
- Kevés feltételes utasítás (thread divergencia)
- Adatlokalitás
- Számításintenzív

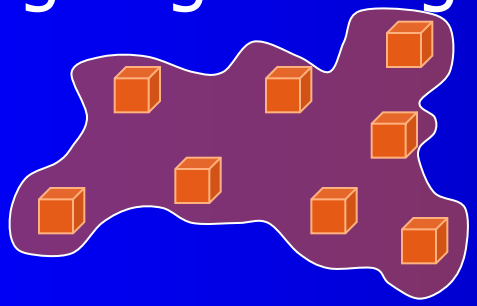
Tudományos (mérnöki) számítások

Idő és térváltozók szerinti differenciálok/integrálok: Parciális diff egyenletek



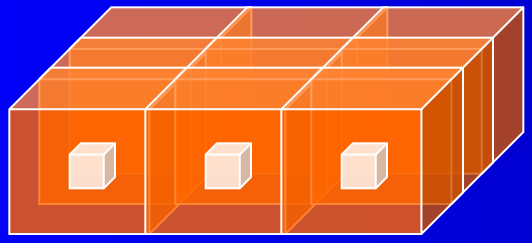
Numerikus megoldás

- Tér-idő differenciál egyenletek
- Idő és térkoordináták diszkretizálása (véges elem)
- Idő: diszkrét idő vagy diszkrét esemény
- Tér: Lagrange-i megközelítés



Mozgó részecskék

Euler-i megközelítés

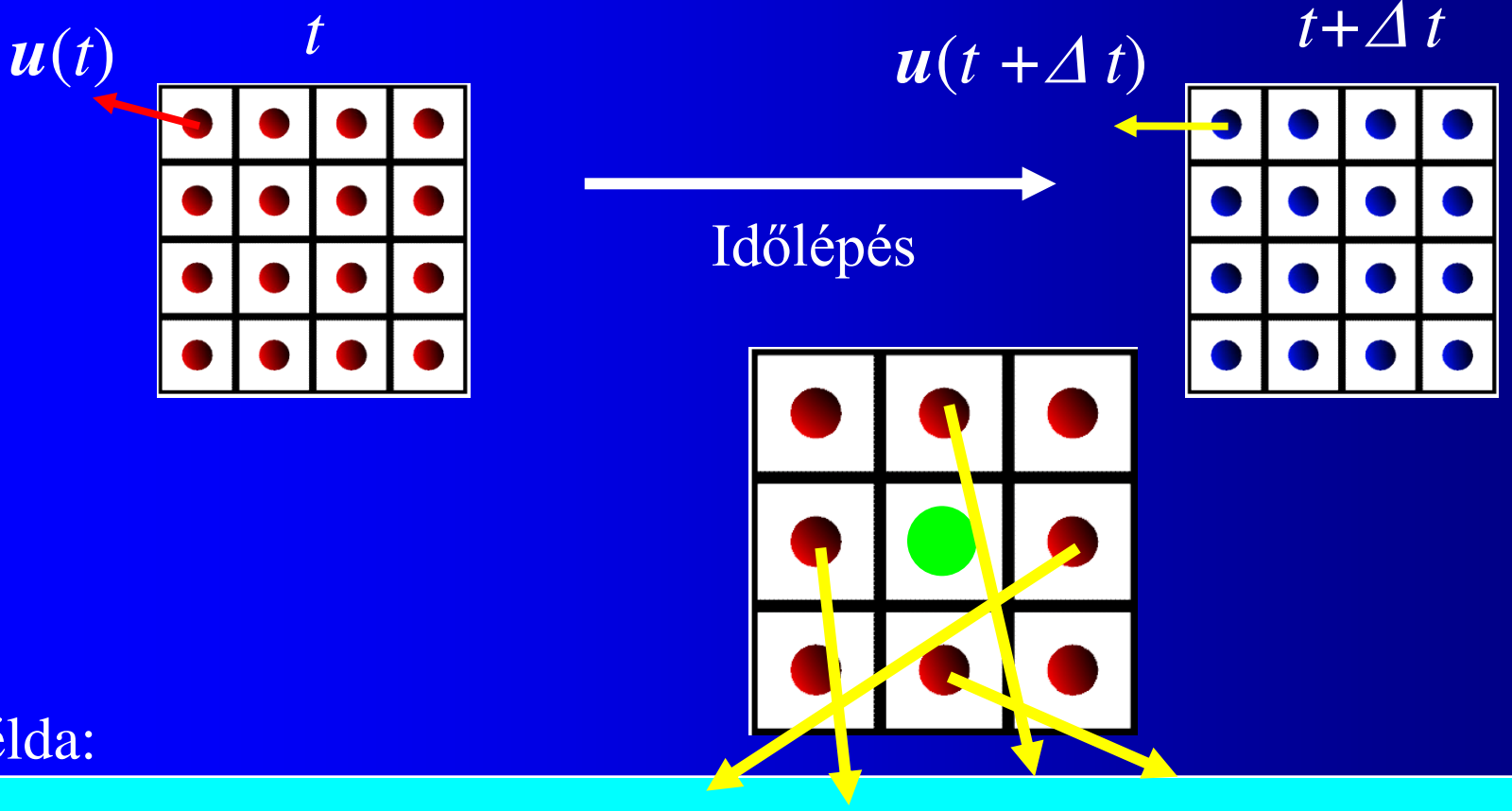


Tér mintavételezése rögzített rácson

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

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

Euler-i folyadék



Példa:

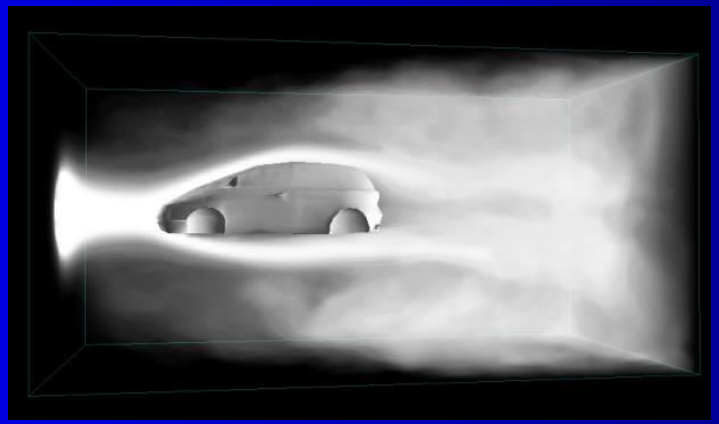
$$\nabla\cdot\vec{u} = \operatorname{div}\vec{u} = \frac{\partial\vec{u}_x}{\partial x} + \frac{\partial\vec{u}_y}{\partial y} + \frac{\partial\vec{u}_z}{\partial z} = \frac{\vec{u}_x^{\,i+1,j,k} - \vec{u}_x^{\,i-1,j,k}}{2\delta x} + \frac{\vec{u}_y^{\,i,j+1,k} - \vec{u}_y^{\,i,j-1,k}}{2\delta y} + \frac{\vec{u}_z^{\,i,j,k+1}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

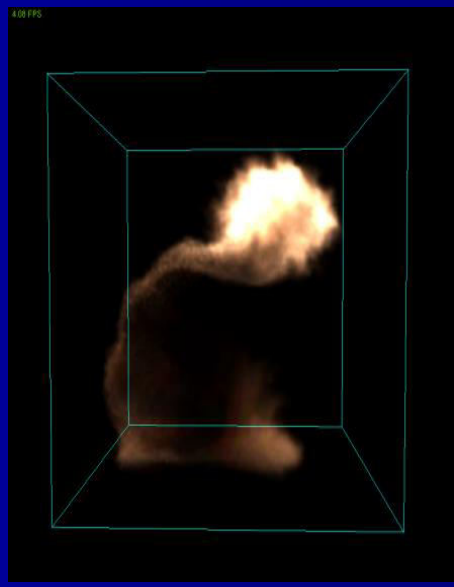
$$\frac{d\vec{u}}{dt} = -(\vec{u}\cdot\nabla)\vec{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\vec{u} + \vec{F}$$

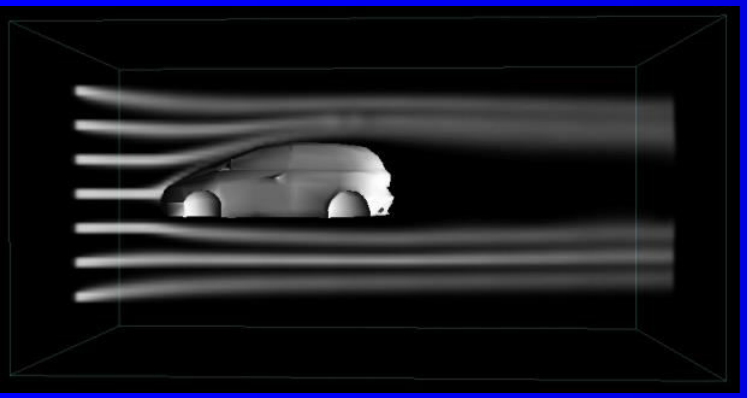
$$\nabla\cdot\vec{u} = 0$$

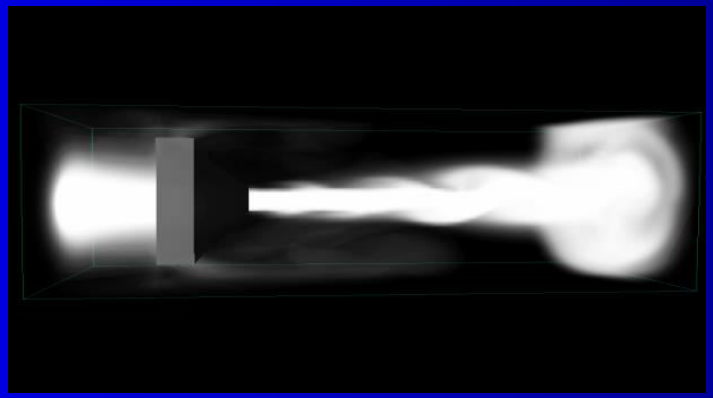
Euler-i folyadék

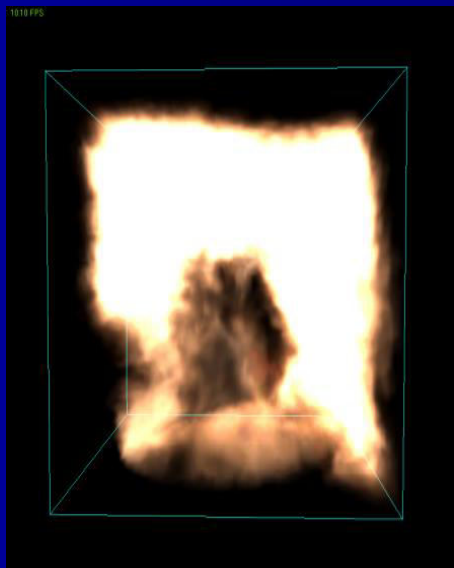




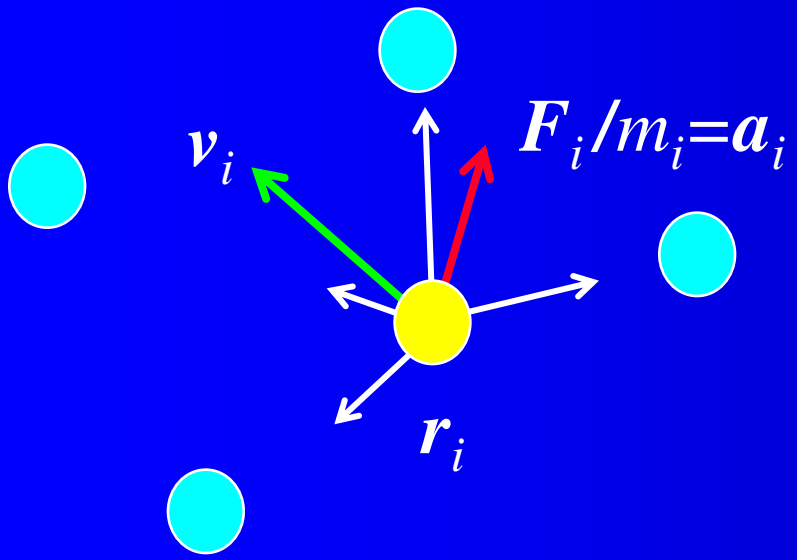




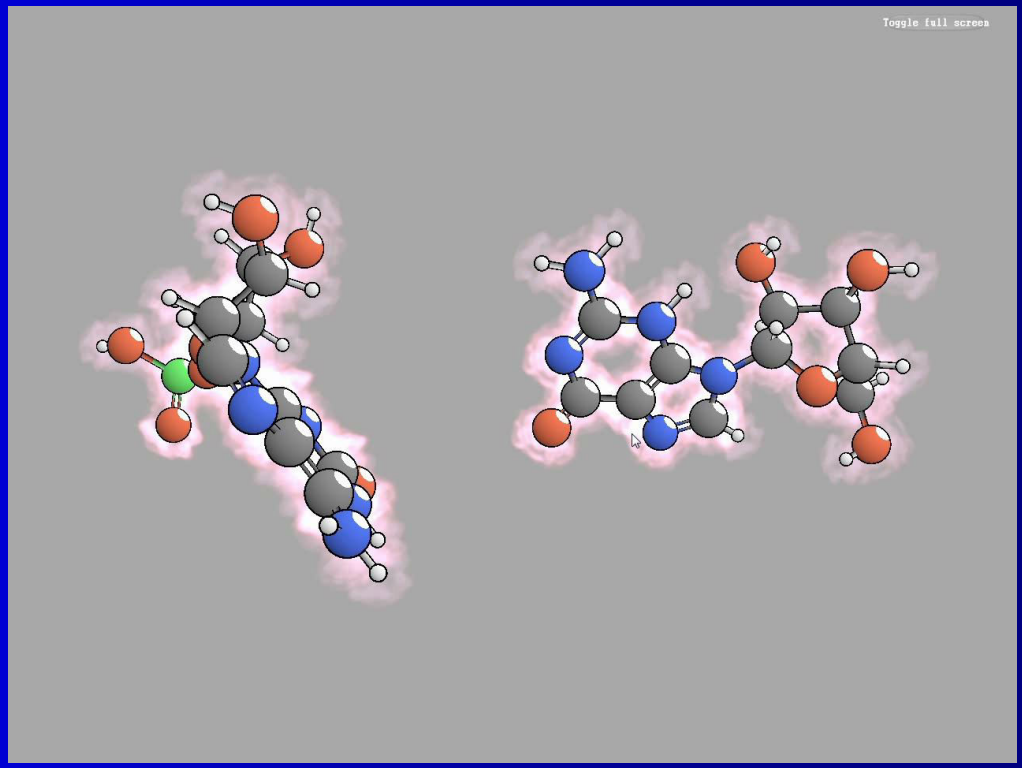




Lagrange-i módszer: Részecske rendszer, N-body

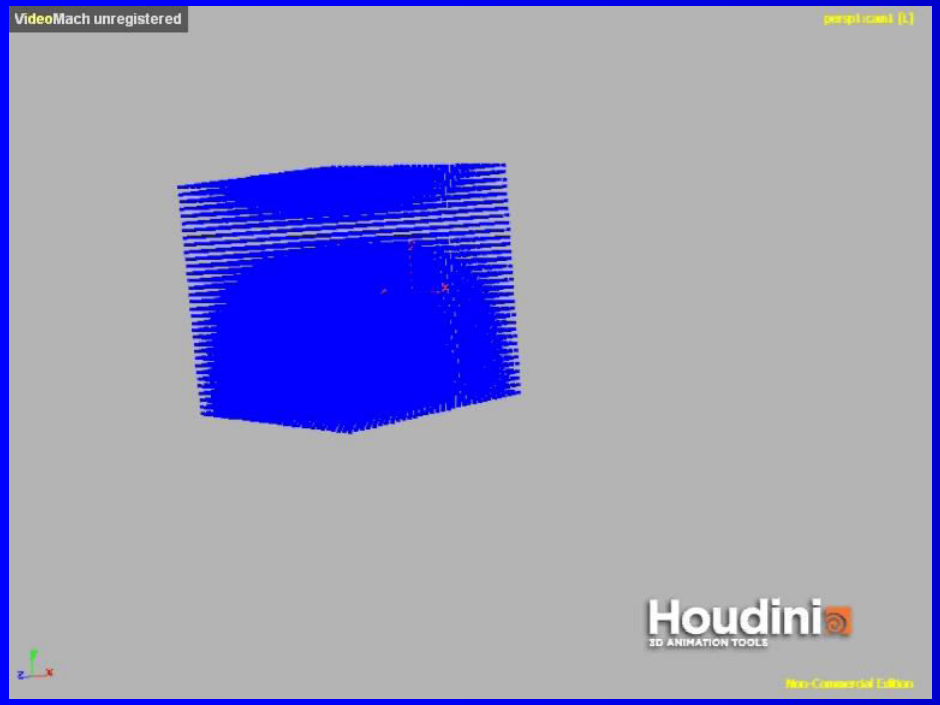


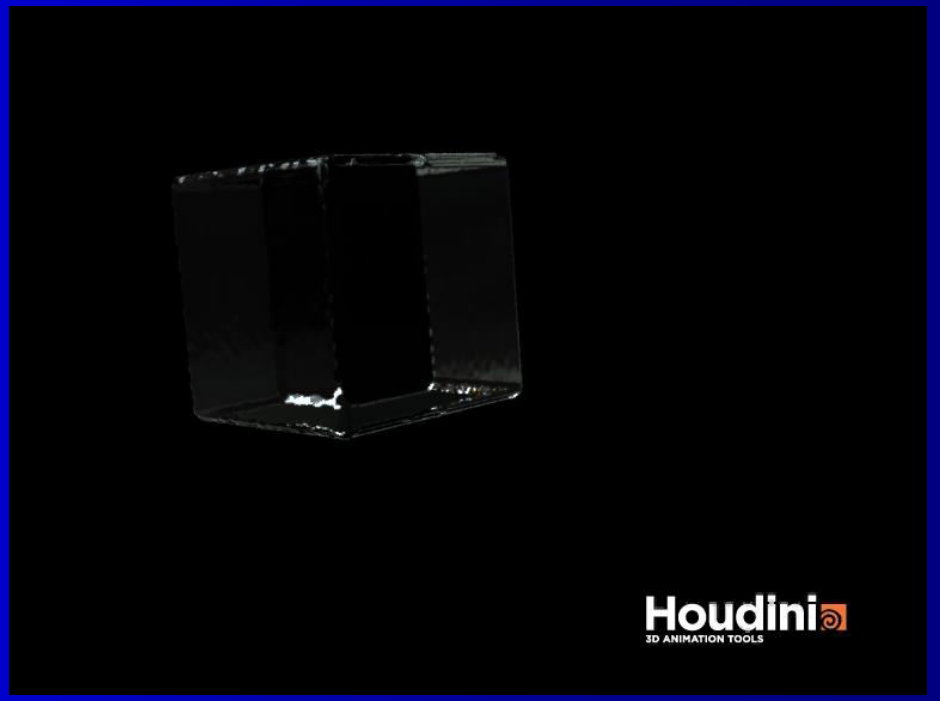
$$v_i \mathrel{+}= a_i \Delta t$$
$$r_i \mathrel{+}= v_i \Delta t$$



Előrelépő Euler séma.
Helyette: Visszalépő Euler, Runte-Kutta, Midpoint, Verlet, …

Lagrange-i folyadékáramlás
SPH=Smoothed Particle Hydrodynamics

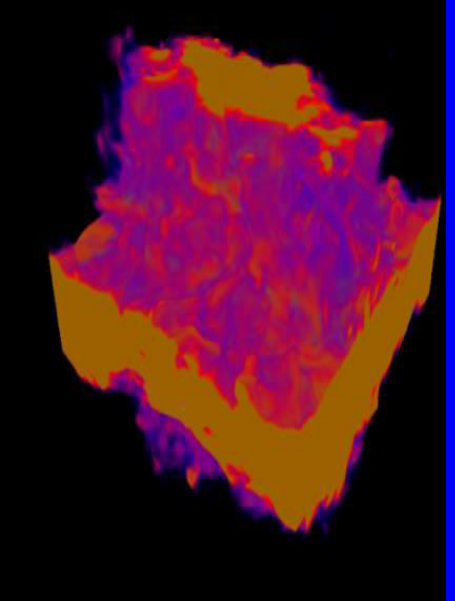




Térfogatvizualizáció

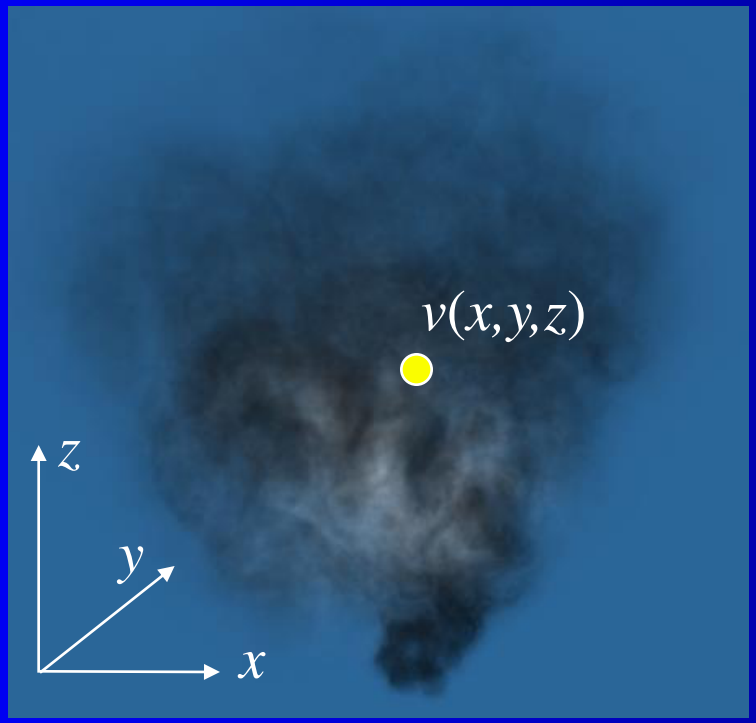
Szirmay-Kalos László

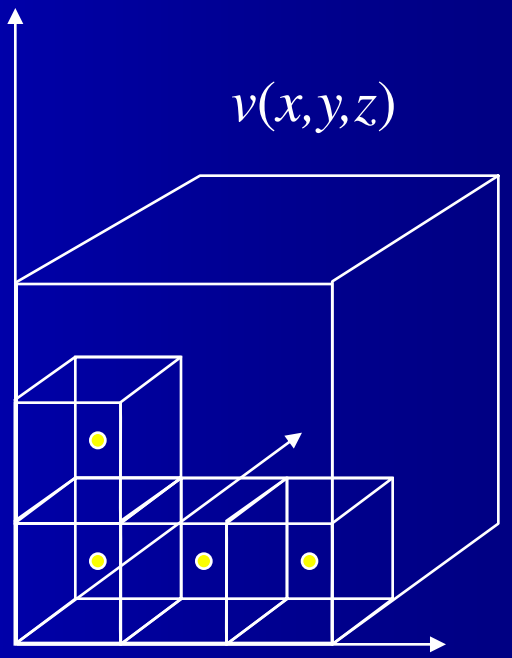




Térfogati modellek (skalár mezők)

3D tér pontjaiban egy skalár érték



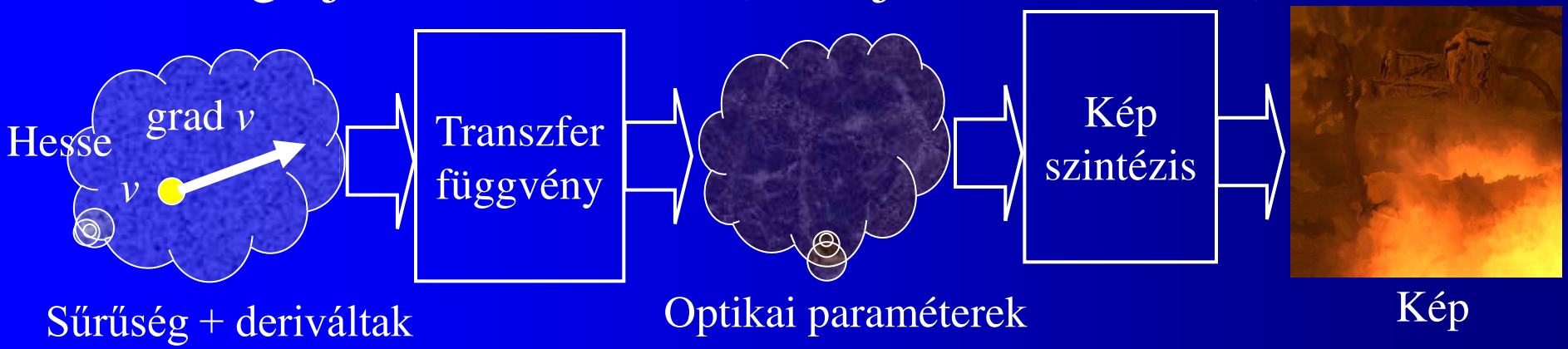


- Skalár: hőmérséklet, sűrűség nyomás, potenciál, …
- Származás: Euler-i szimuláció, Rekonstrukció (tomográfia)

tárolás: 3D textúra vagy „voxel tömb”

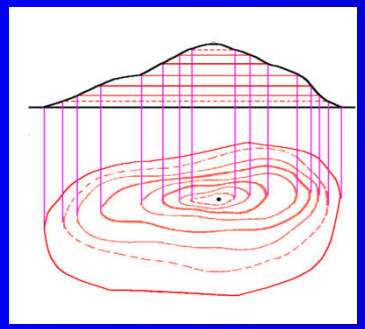
Térfogati modell megjelenítése

- Megjelenítés fényszóró anyag (participating media) analógiáját felhasználva (belsejébe belelátunk)

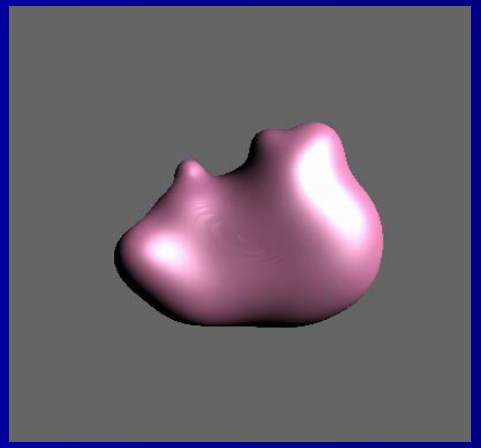


- Adott szintfelület kiemelése (külsőt lehámozzuk)

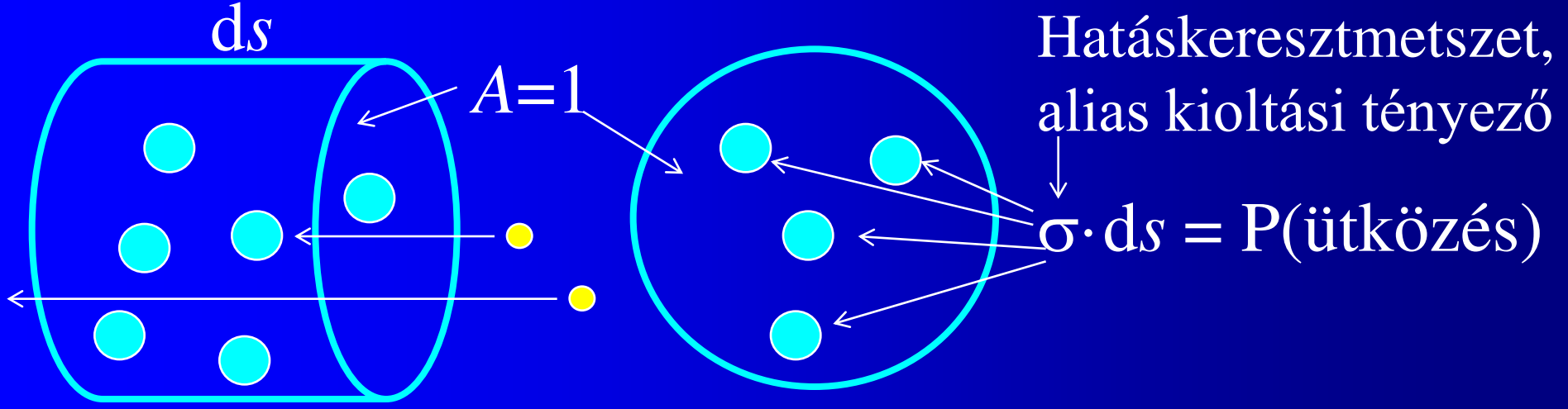






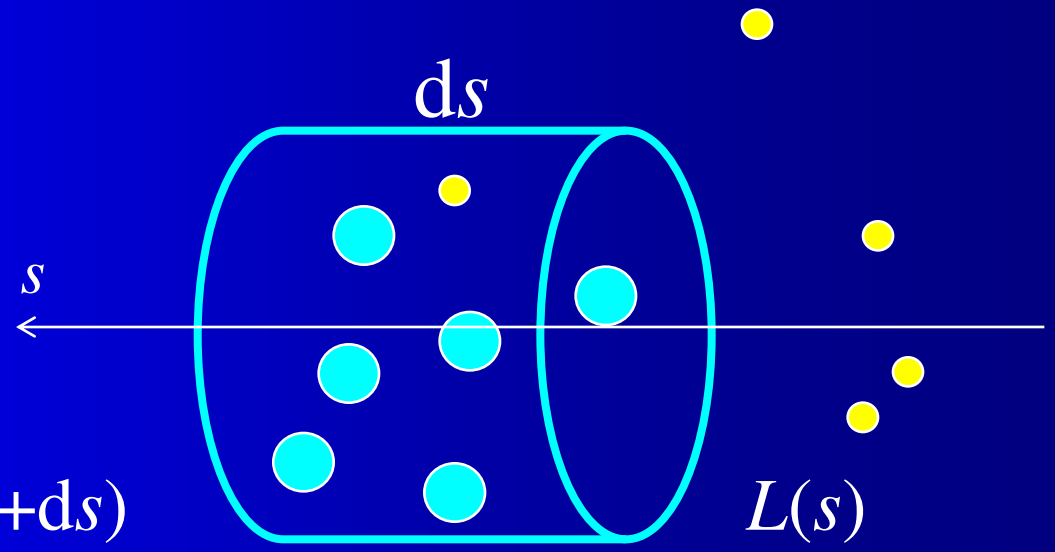


Fényszóró közeg



- Albedo *a*: a nem-elnyelődés valószínűsége feltéve, hogy az ütközés bekövetkezett
- Fekete test: albedo = 0

Sugársűrűség változása



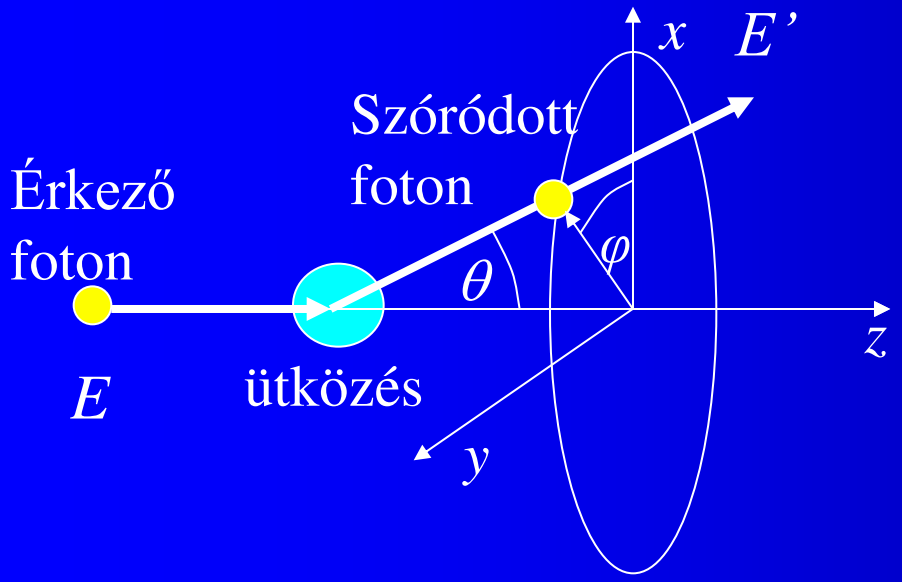
$$L(s+\mathrm{d}s) = L(s) - L(s)\cdot\sigma(s)\cdot\mathrm{d}s + \quad \text{// Kiszóródás+abszorbció}$$
$$L^e(s)\cdot\mathrm{d}s + \quad \text{// Emisszió}$$
$$\sigma(s)\cdot a(s)\cdot\mathrm{d}s\cdot\int f(\omega',\omega)L^i(\omega')\mathrm{d}\omega' \quad \text{// Beszóródás}$$

$$\boxed{\mathrm{d}L(s)/\mathrm{d}s = -L(s)\cdot\sigma(s)} + L^e(s) + L^{inscatter}(s)$$

Megoldás fényelnyelő közegre (emisszió és beszóródás nincs):

$$L(s) = L(0)\cdot\exp\left(-\int^s \sigma(s)\mathrm{d}s\right)$$

Szóródás



Klein-Nishina:

$$P(\theta) = C\left(\frac{E'}{E} + \left(\frac{E'}{E}\right)^3 - \left(\frac{E'}{E}\right)^2 \sin^2\theta\right)$$

Compton formula:

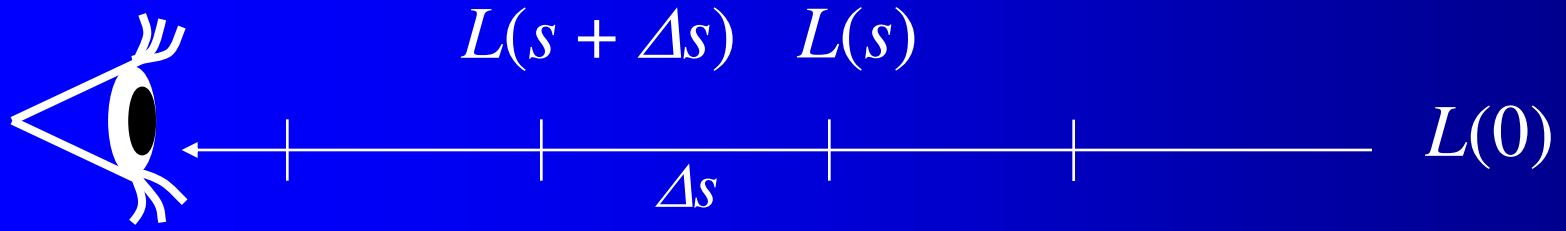
$$\frac{E'}{E} = \frac{1}{1 + \frac{E}{m_e c^2}(1 - \cos\theta)}$$

Rayleigh:

$$P(\theta) = C\left(1 + \cos^2\theta\right)$$

Sugár masírozás (ray marching)

Megoldás: $\mathrm{d}L(s)/\mathrm{d}s = -L(s)\cdot\sigma(s) + L^{e}(s) + L^{inscatter}(s)$



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

$\alpha(s)$ opacitás

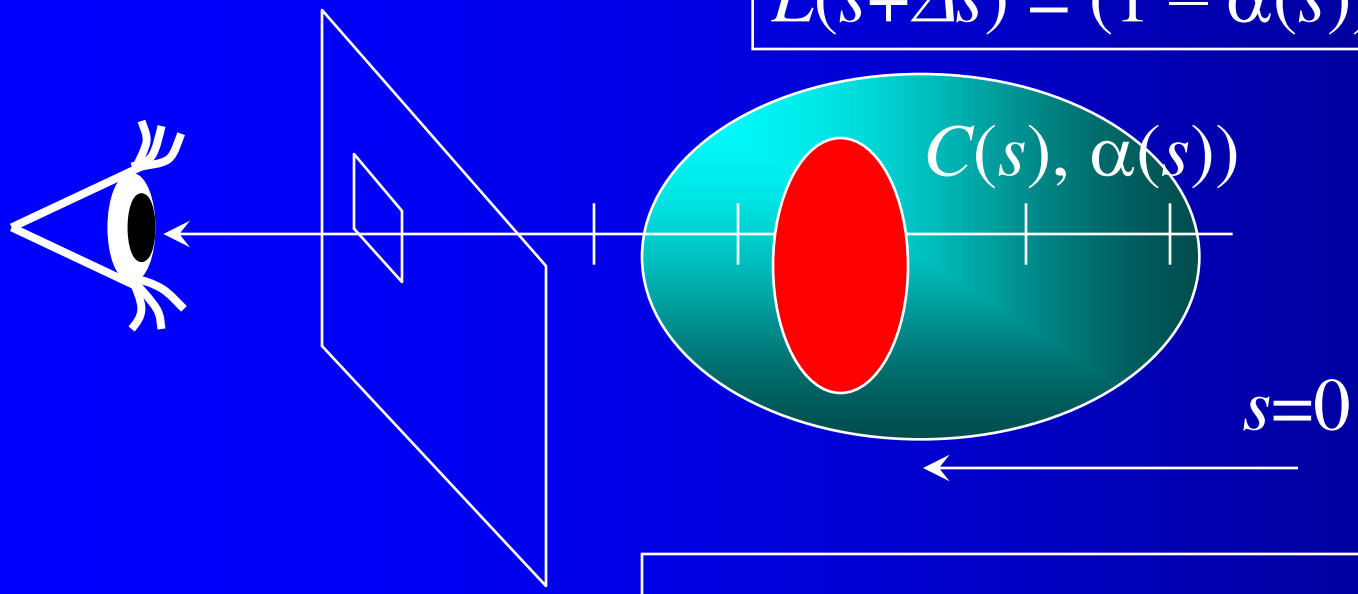
$C(s)$ hozzájárulás

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

átlátszóság

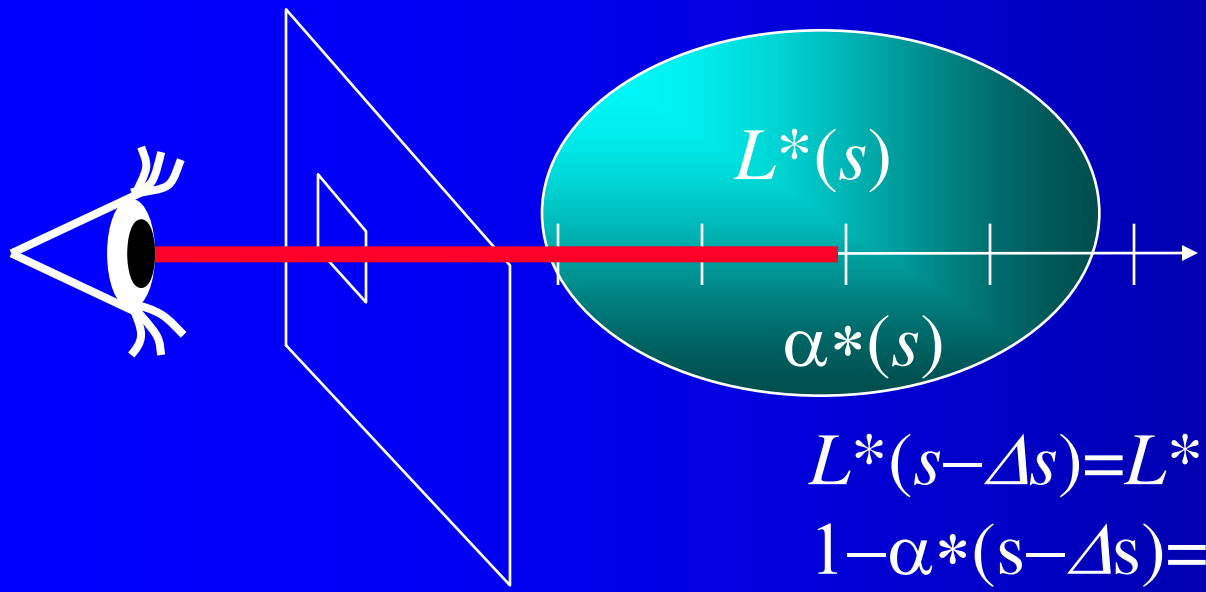
Back-to-front ray marching

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




```
L = L0;
for(s = 0; s < T; s += Δs) {
        L = (1 – α(s)) · L + C(s);
}
```

Front-to-back ray marching



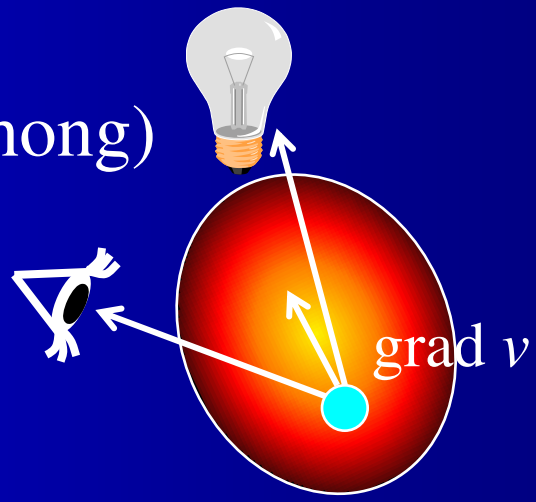
$$L^*(s-\Delta s)=L^*(s)+(1-\alpha^*(s))\cdot C(s)$$
$$1-\alpha^*(s-\Delta s)=(1-\alpha^*(s))\cdot(1-\alpha(s))$$

```
L* = α* = 0;
for( s = T; s >0 ; s −= Δs ) {
        L* += (1− α*) · C(s);
        α* = 1−(1− α*) · (1−α(s));
        if (α* > 1−ε) break;
}
```

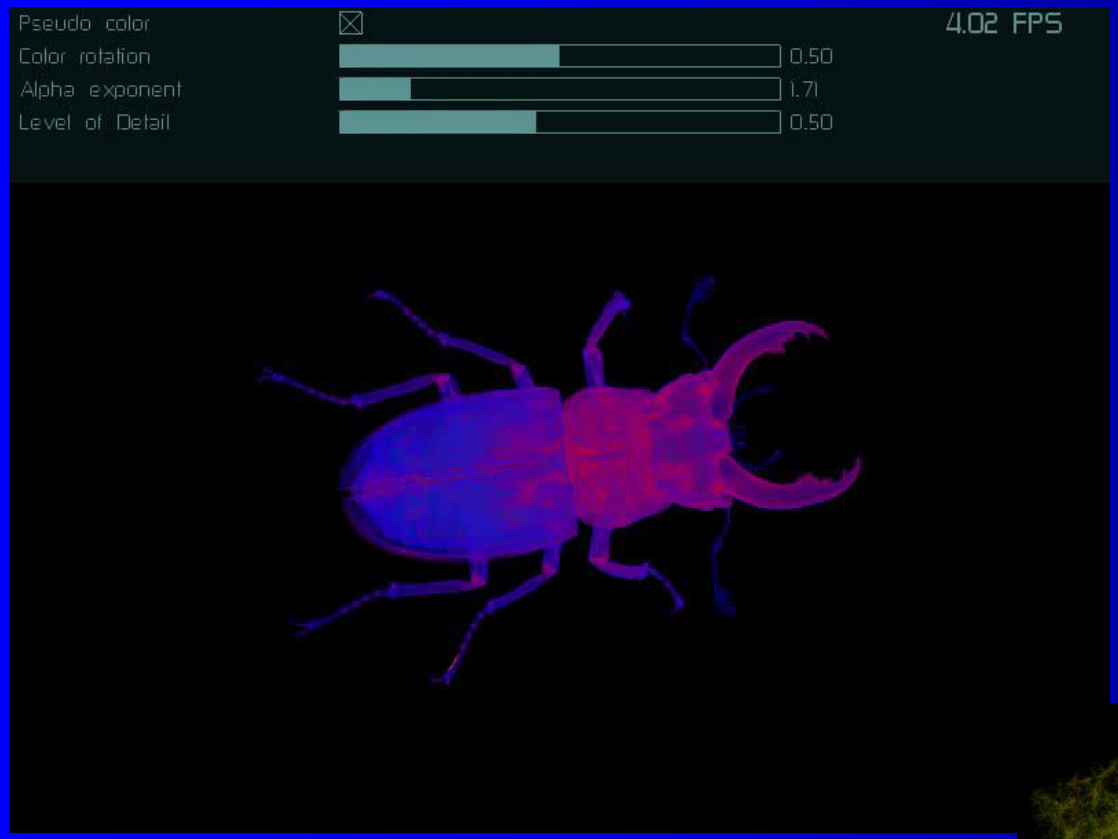
Voxel szín és opacitás: Transfer func: $(C,\alpha)=T(v\ függv)\cdot\Delta s$

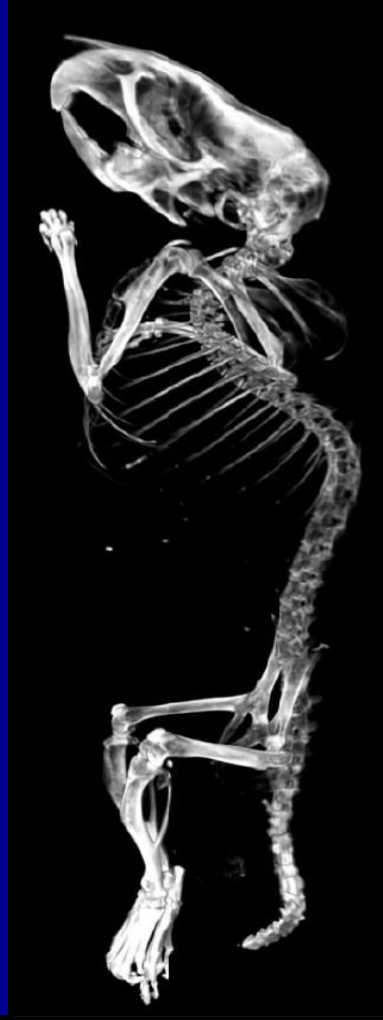


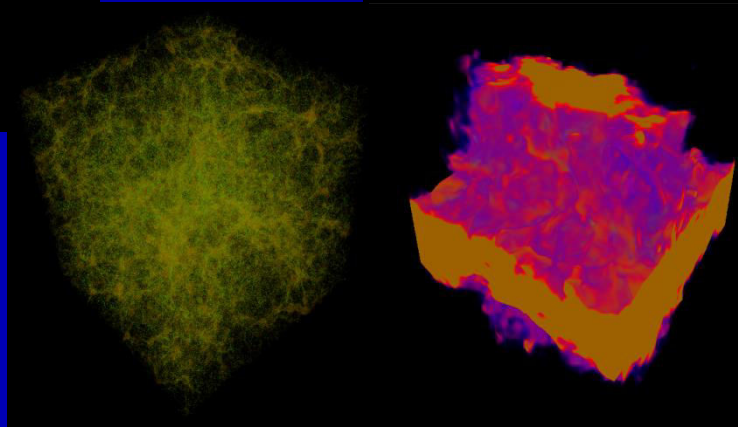
- Röntgen: $(C,\alpha)=T(v(x,y,z))\cdot\Delta s$
  - opacitás = $v(x,y,z)\cdot\Delta s$
  - $L(0) = I$, egyébként $C(s)=0$
- Klasszikus árnyalási modellek
  - opacitás: v osztályozása
  - C = árnyalási modell (diffúz + Phong)
    - normál = grad v
    - opacitás *= | grad v |
- Magasabb rendű derivált (görbület)
- Transzlucens anyagok (subsurface scattering)



$$(C,\alpha) \equiv T(v)\cdot\Delta s$$







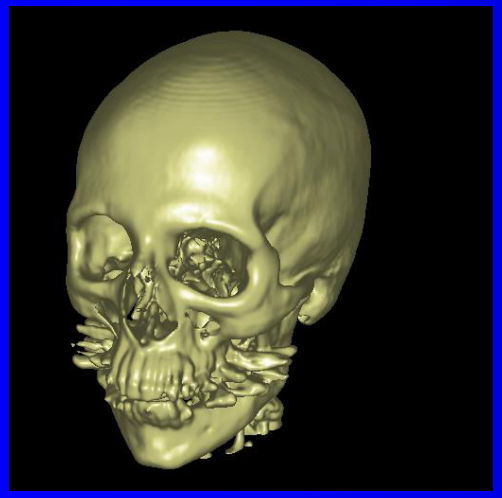
$\alpha = \mathrm{pow}(v/vmax, \mathbf{aexp})\cdot\Delta s$

$C = \mathrm{HLS}((v/vmax+\mathbf{rot})\cdot 360, 0.5, 1)\cdot\Delta s$

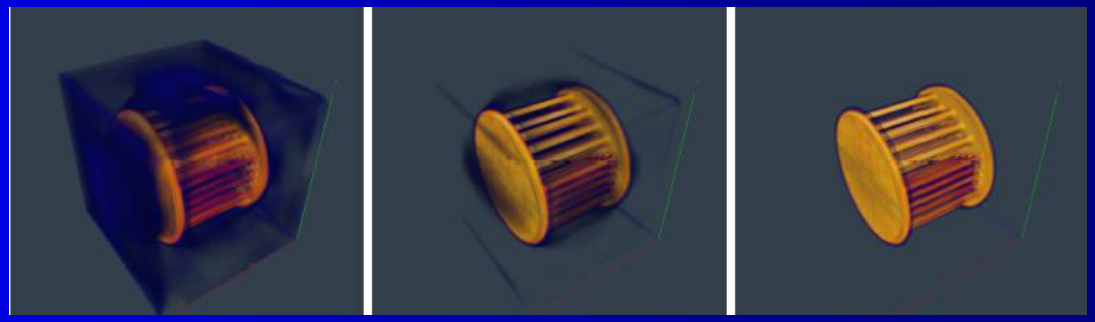
Klasszikus BRDF modellek

First hit ray casting:

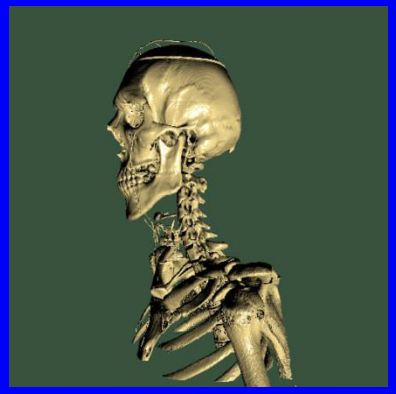
Diffúz+Phong árnyalás



Csont α: 1, másé 0



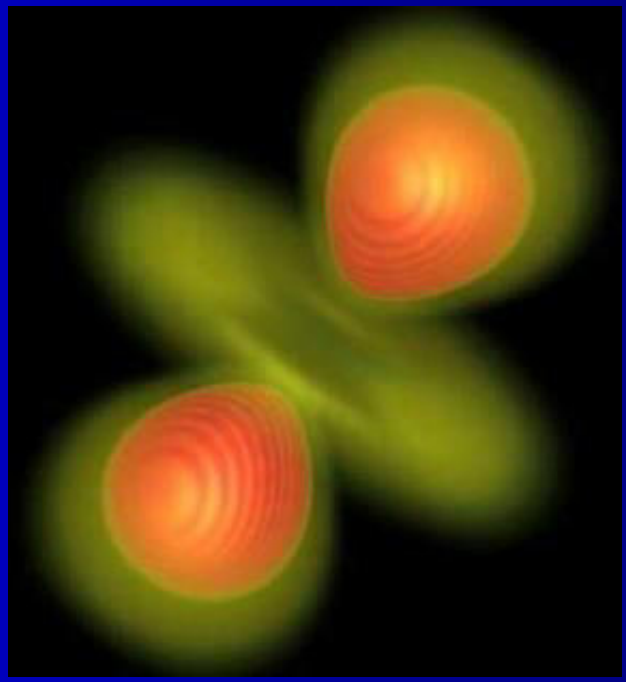
v két osztályba: kék-átlátszó, sárga-átlátszatlan



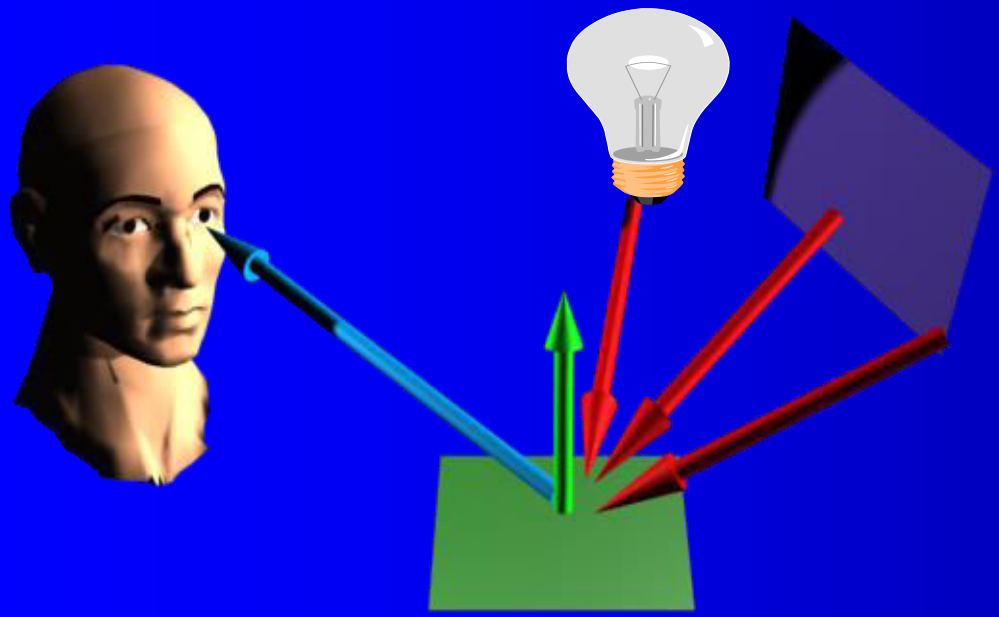
Csont α=1, másé=0

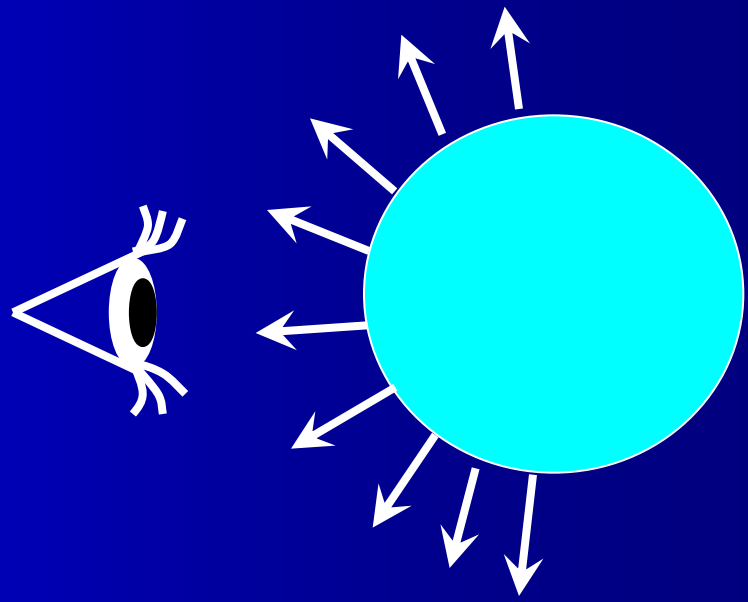


Hús α=1, másé=0



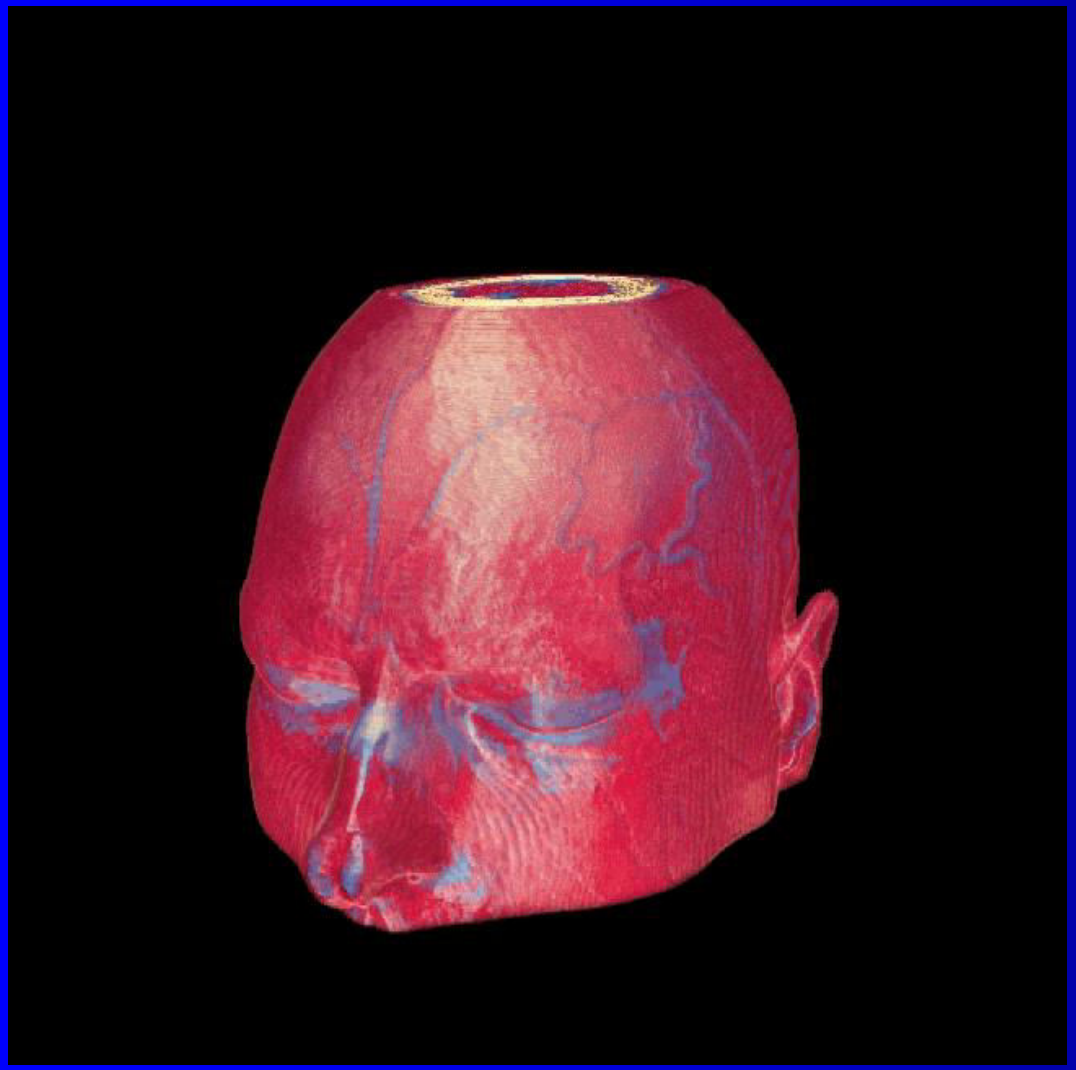
Illusztratív vizualizáció

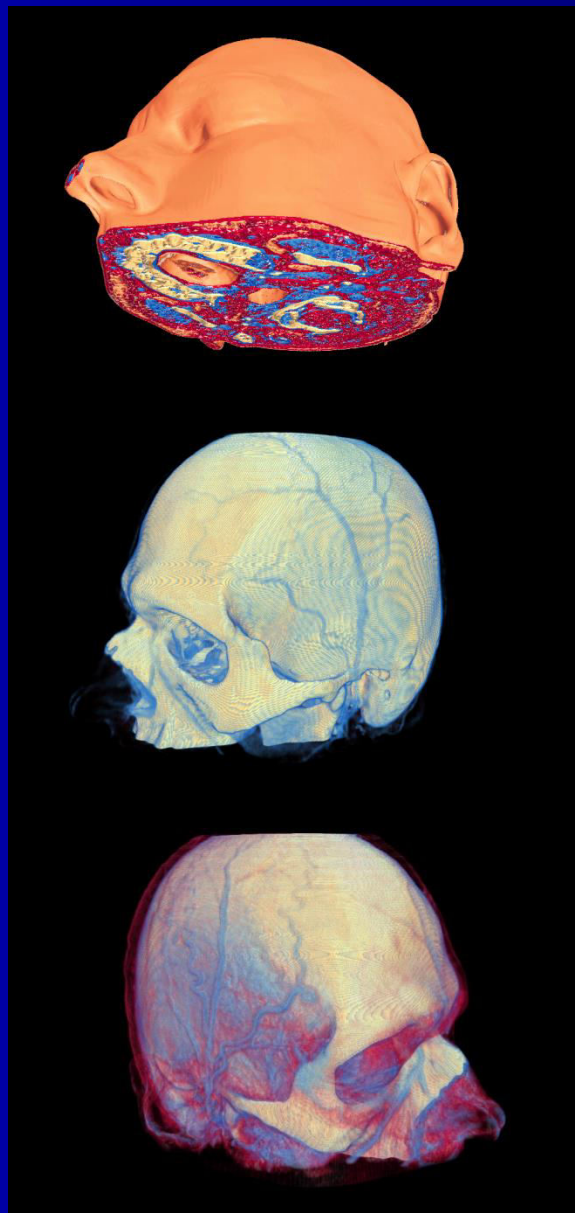




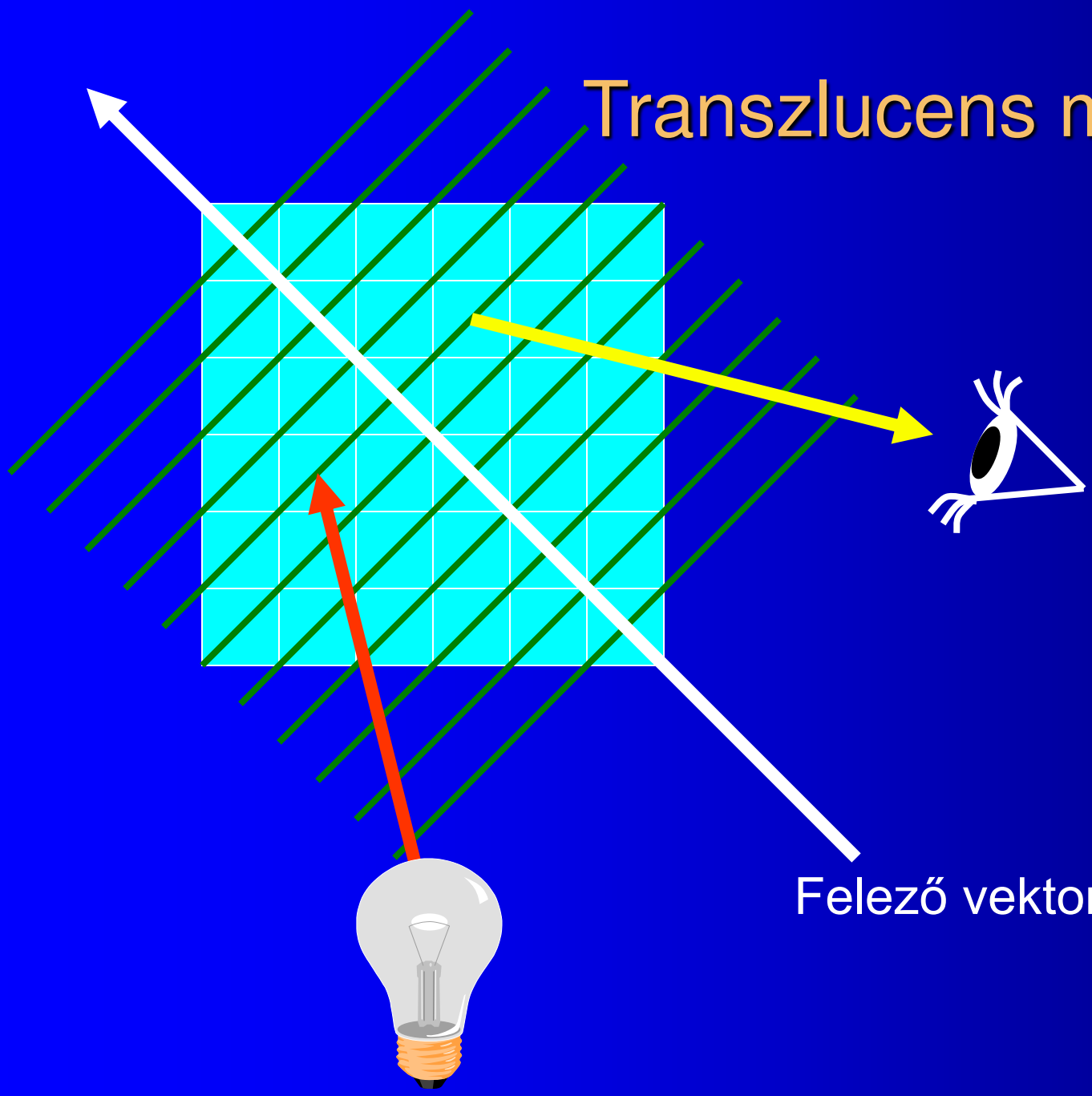


Illusztratív vizualizáció

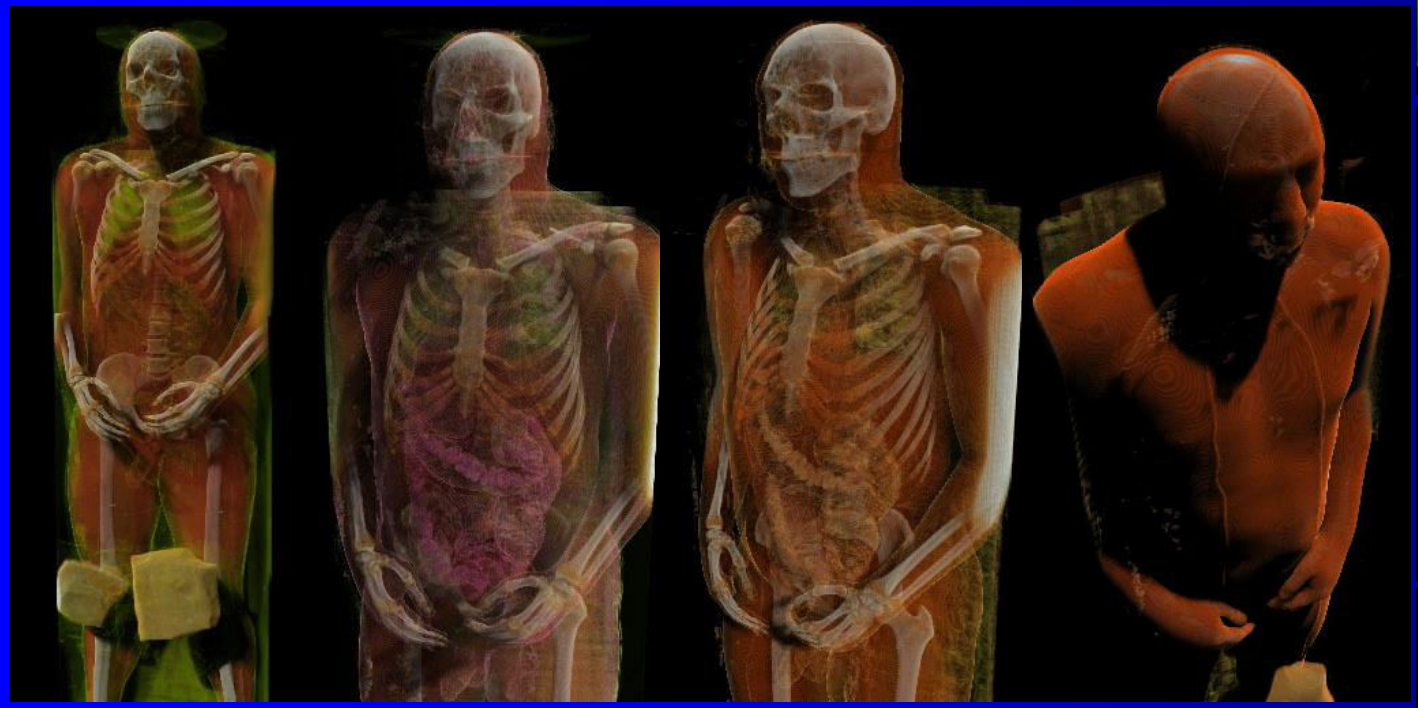


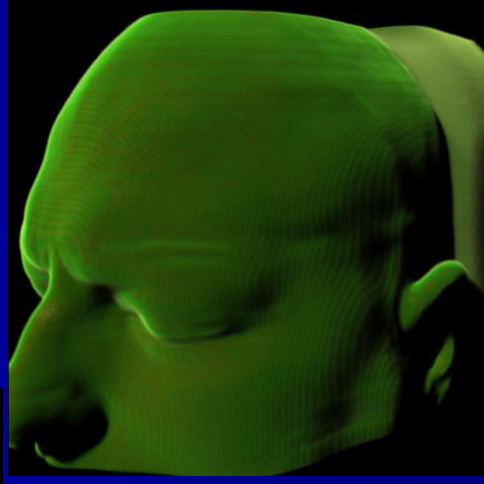


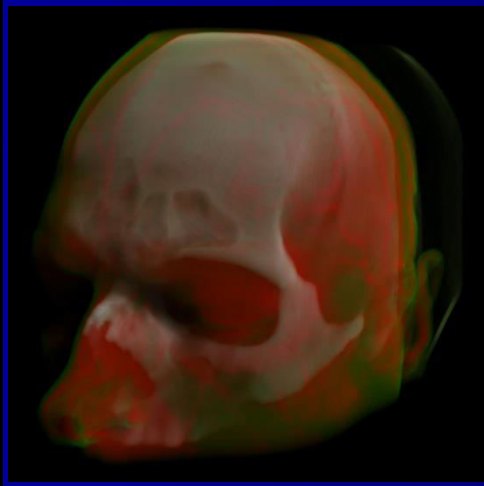
Transzlucens megjelenítés



Transzlucens megjelenítés

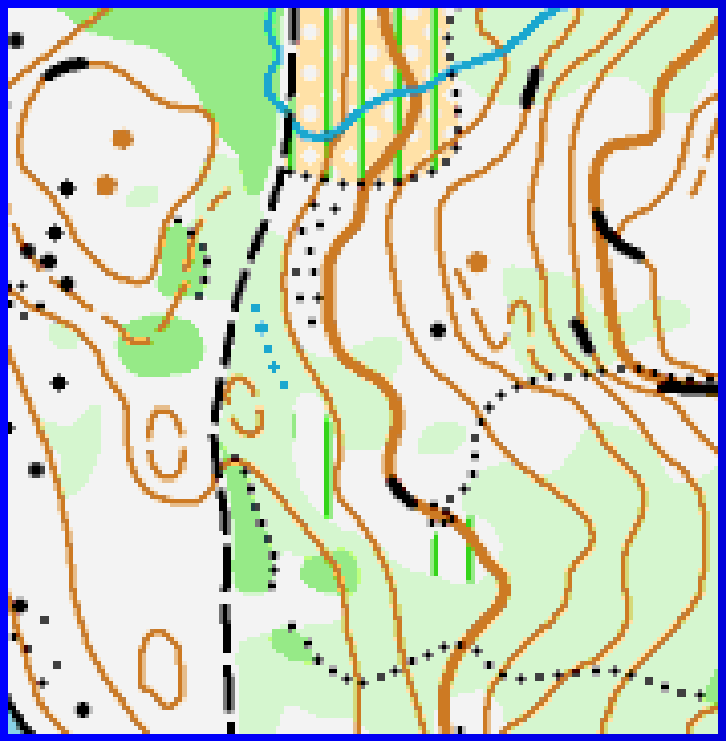


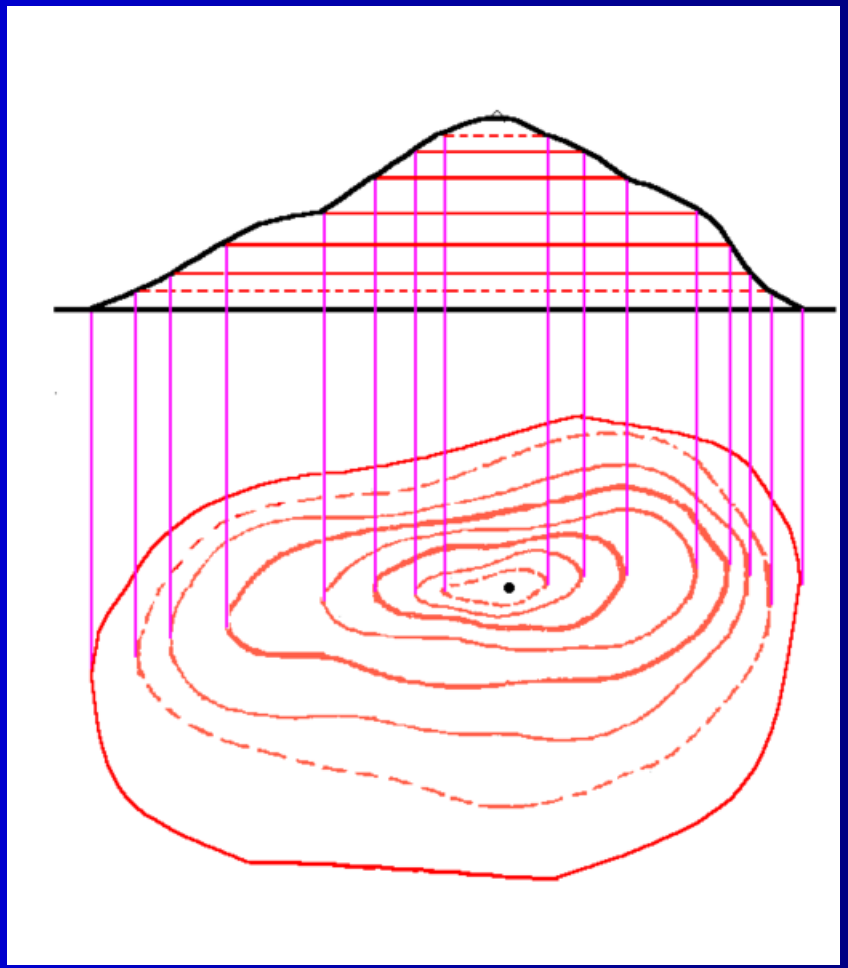




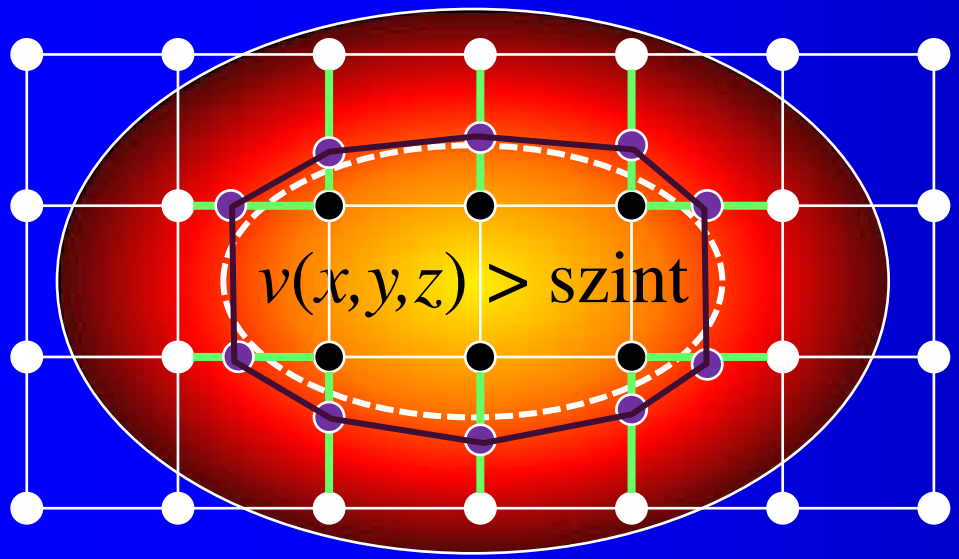


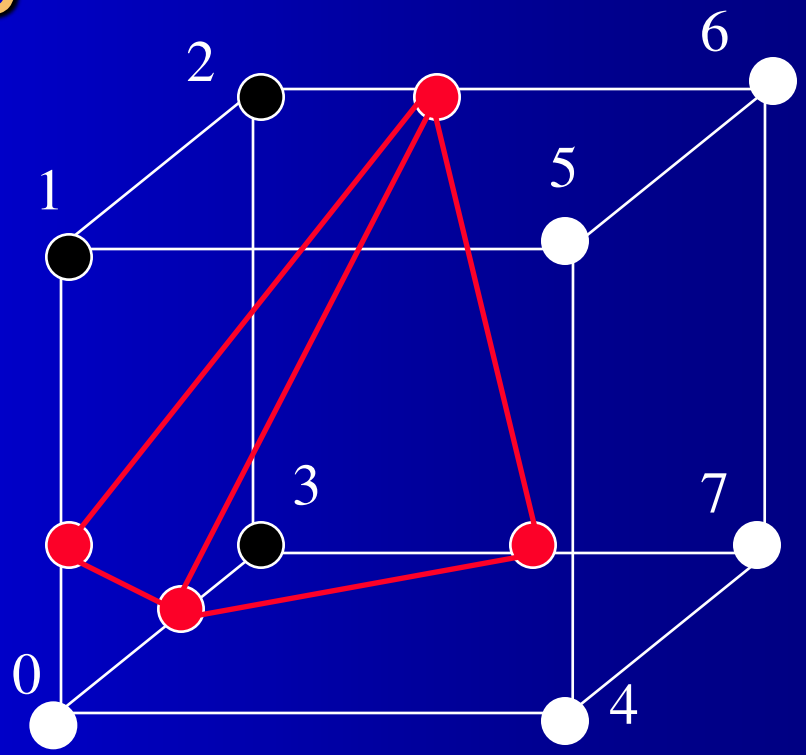
Szintvonal, szintfelület





Marching cubes



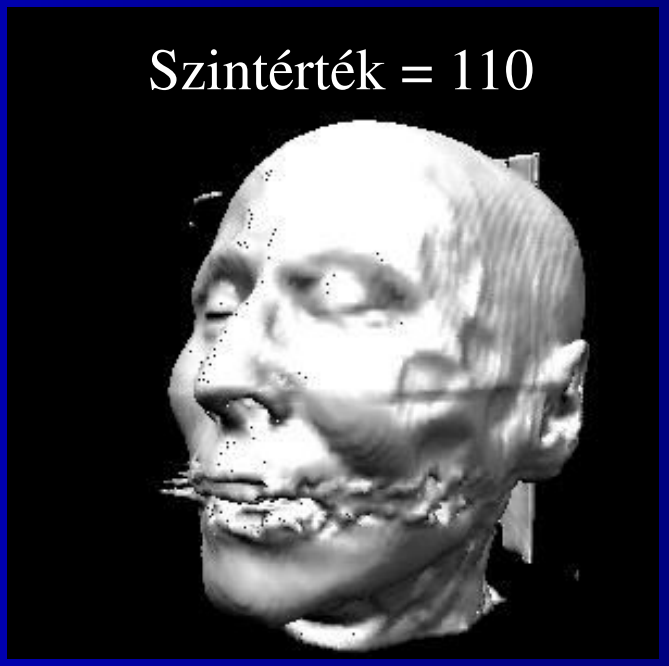


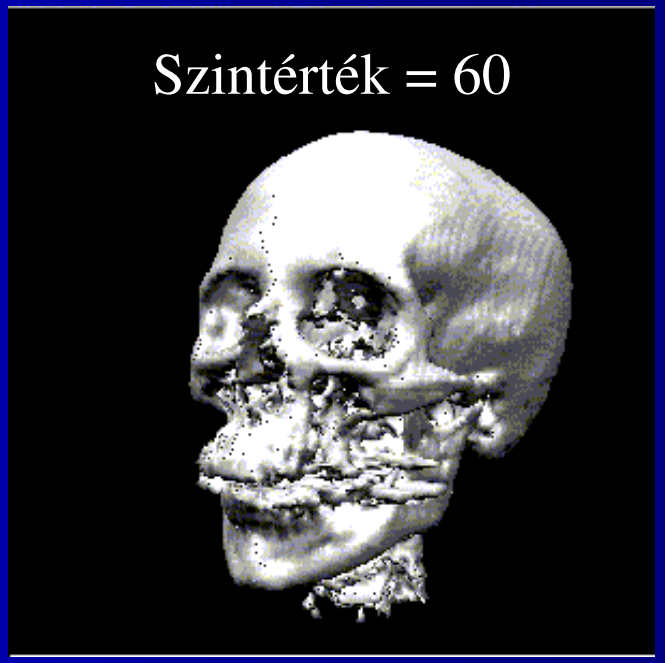
0	0;
14	2; 0-1; 0-3; 2-6; 0-3; 3-7; 2-6
255	0;

Eset: $00001110_2 = 14$ → 14

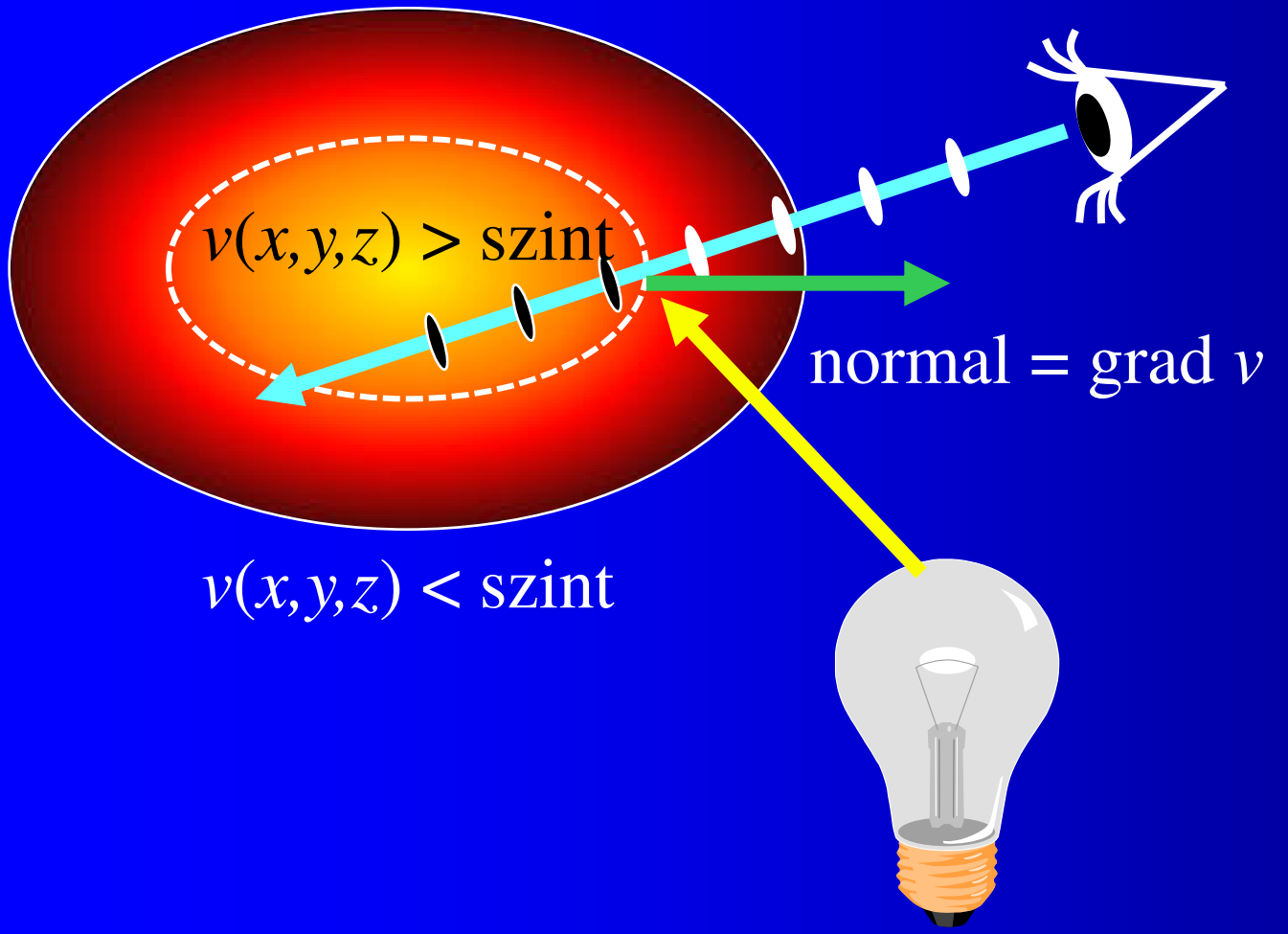
Masírozó kockák







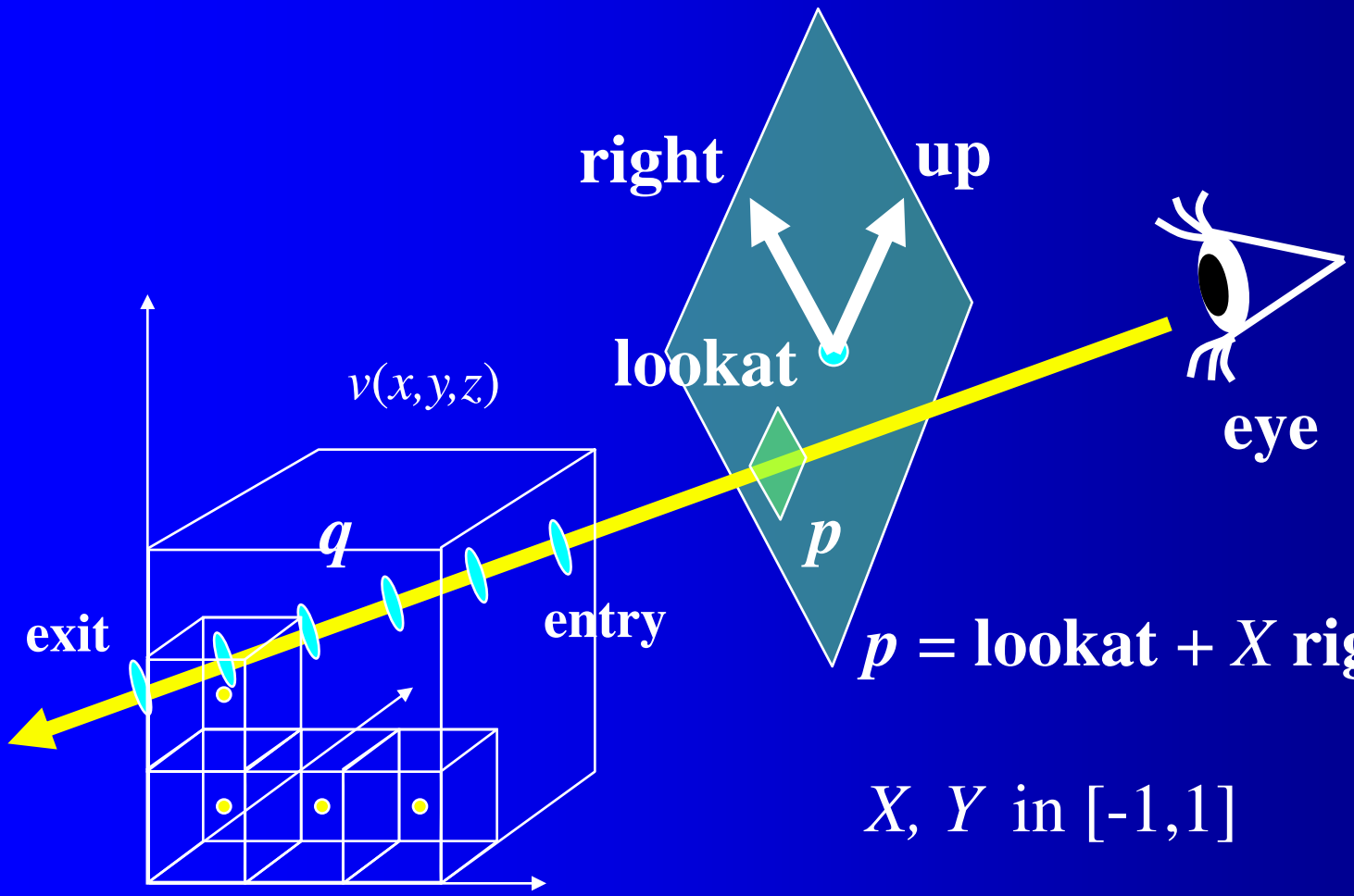
First hit (isosurface) ray casting







GPU first hit ray-casting



p = **lookat** + X **right** + Y **up**

X, Y in [-1,1]

Egység kocka 3D textúrával

CPU first hit ray-casting

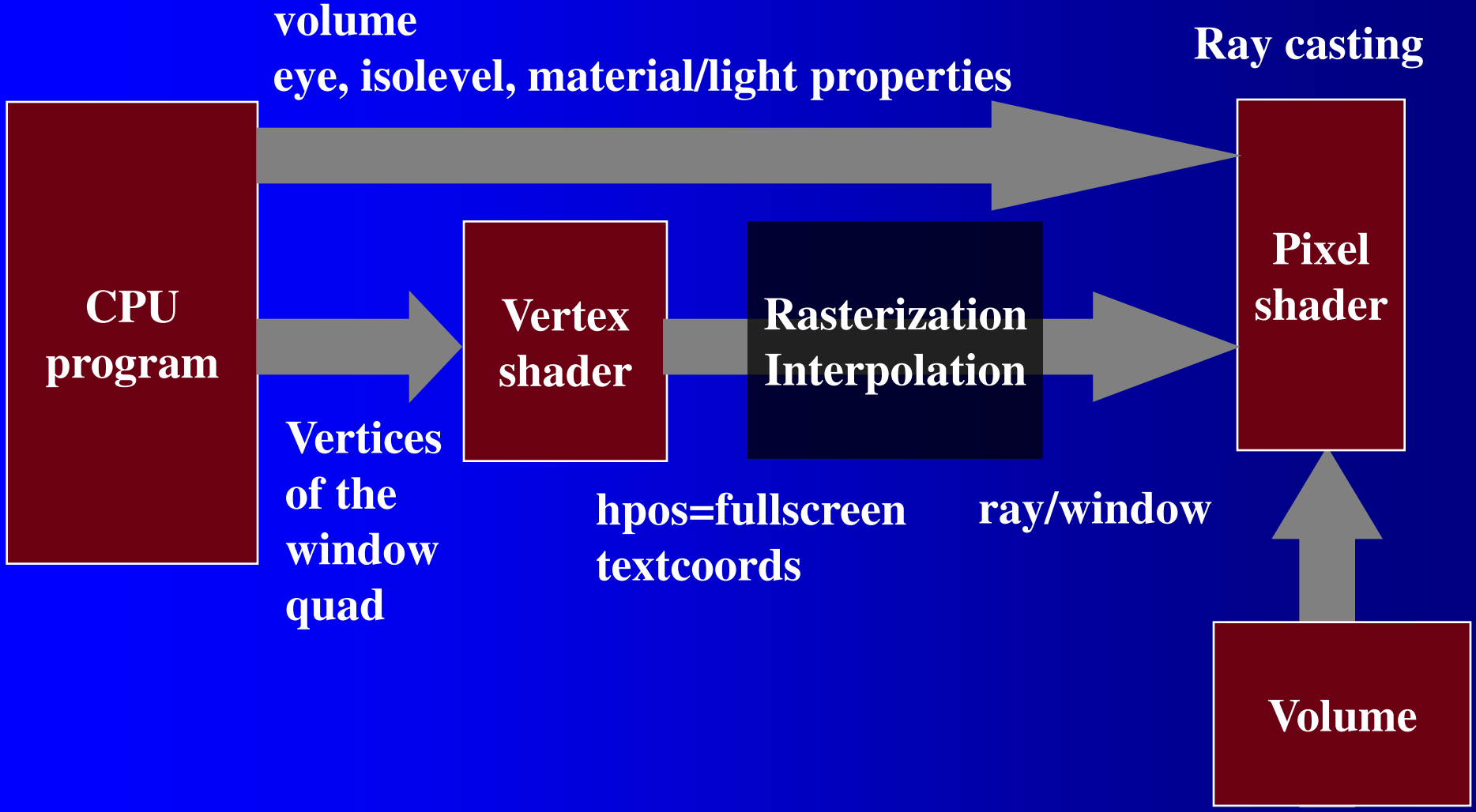
```
For each pixel
    Find pixel center p
    raydir = normalize(p – eye);
    Find exit and entry
    for(t = entry; t < exit; t+=dt) {
        q = eye + raydir * t;
        if (volume[q] > isovalue) break;
    }
    normal vector estimation;
    illumination
}
```

Full screen quad

Interpolation from the corners

central differences

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();
}
```

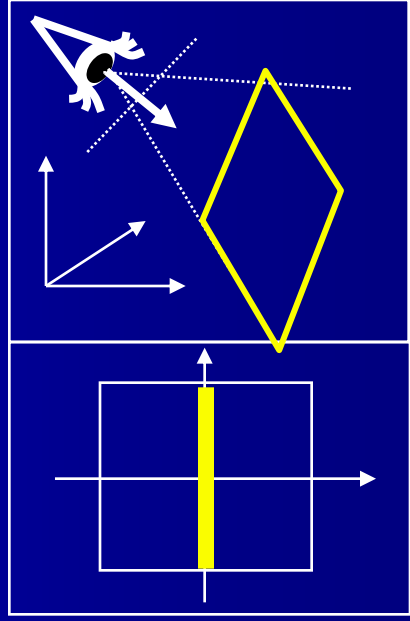


Camera window as texture coordinates



Full screen quad

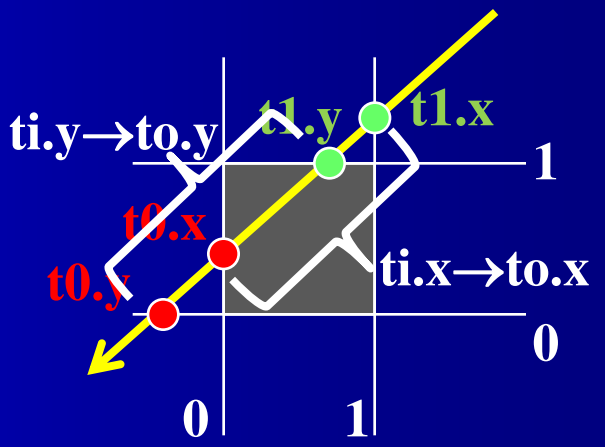
Ray casting: vertex shader



```
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

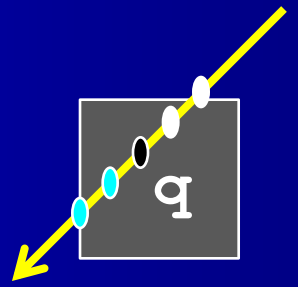
```
void FragmentShader( in float3 p : TEXCOORD0, // point on window
                     uniform float3 eye,
                     uniform sampler3D volume, // voxels
                     uniform float isolevel,
                     uniform float3 lightdir, lightint, kd ☹
                     out float3 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);
```



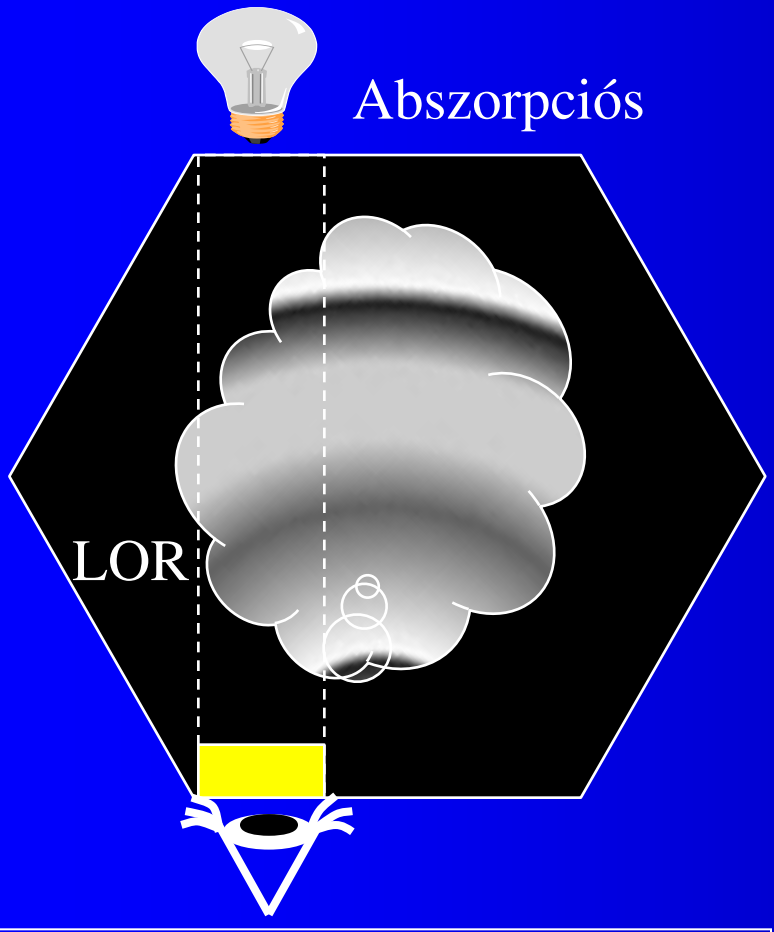
cont'd…

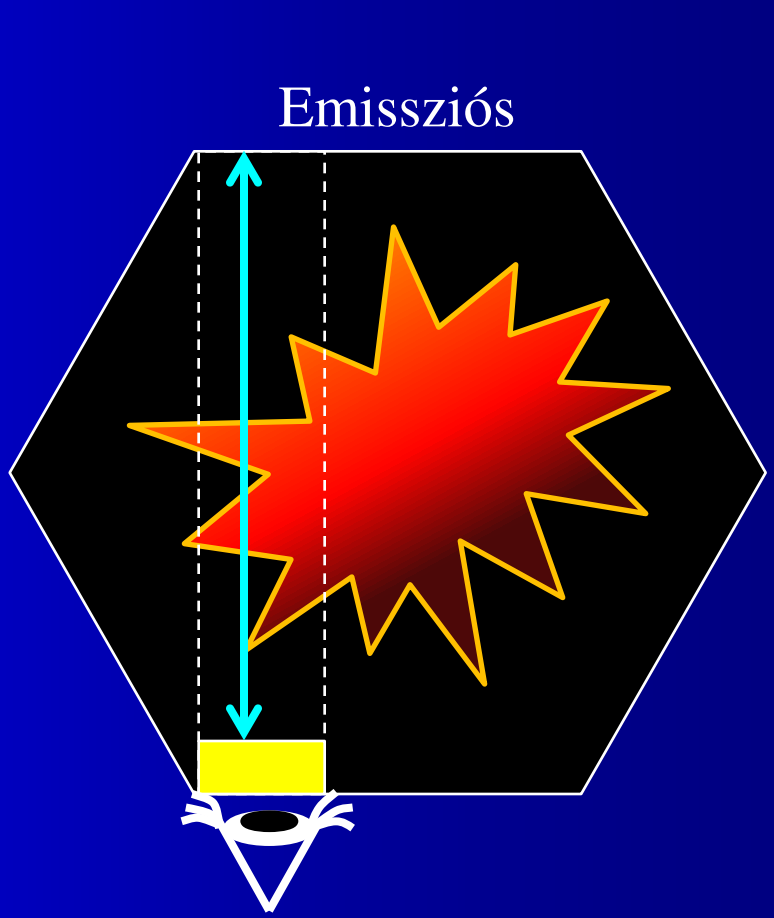
Ray casting fragment shader cont'd

```
    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);
        }
    }
}
```



Tomográfia

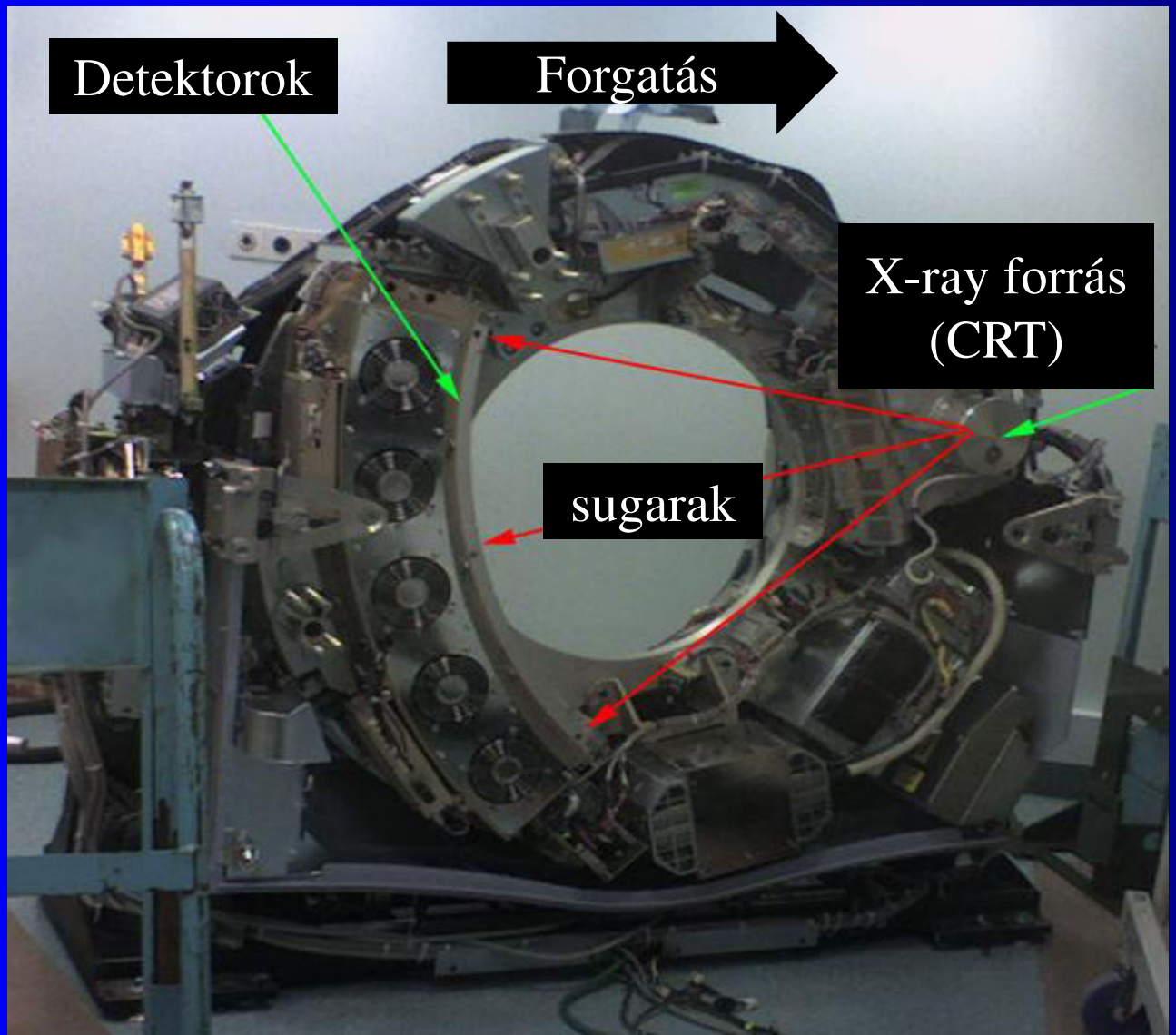




$$L(s) = L(0) \cdot \exp(-\int \sigma(s) \mathrm{d}s)$$
$$\int \sigma(s) \mathrm{d}s = -\log(L(s)/L(0))$$

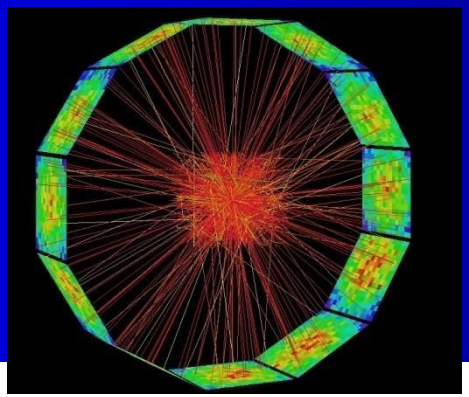
$$L(s) = \int L^e(s) \mathrm{d}s$$

X-ray Computed Tomography



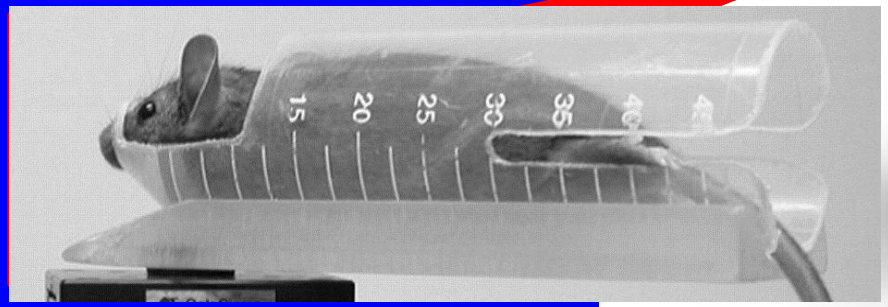
Mediso NanoPET/CT







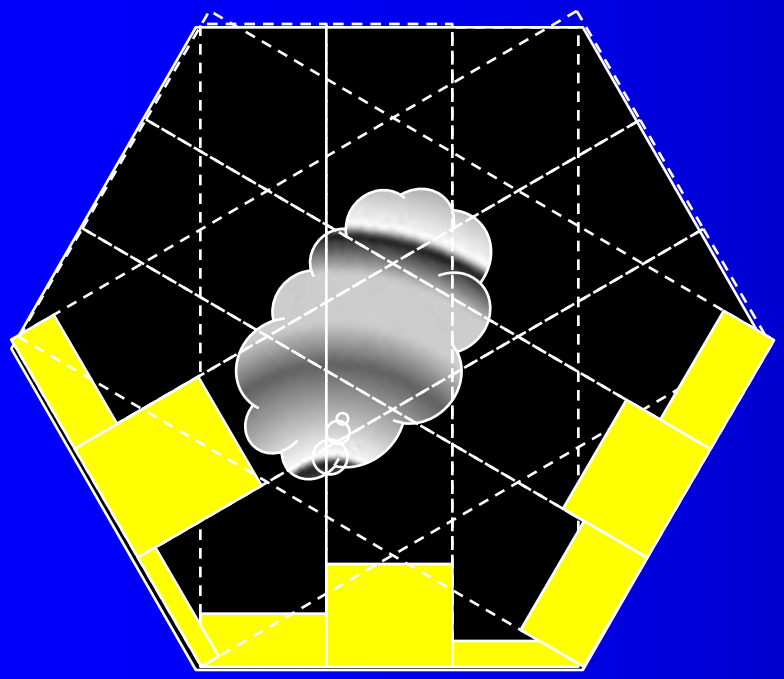


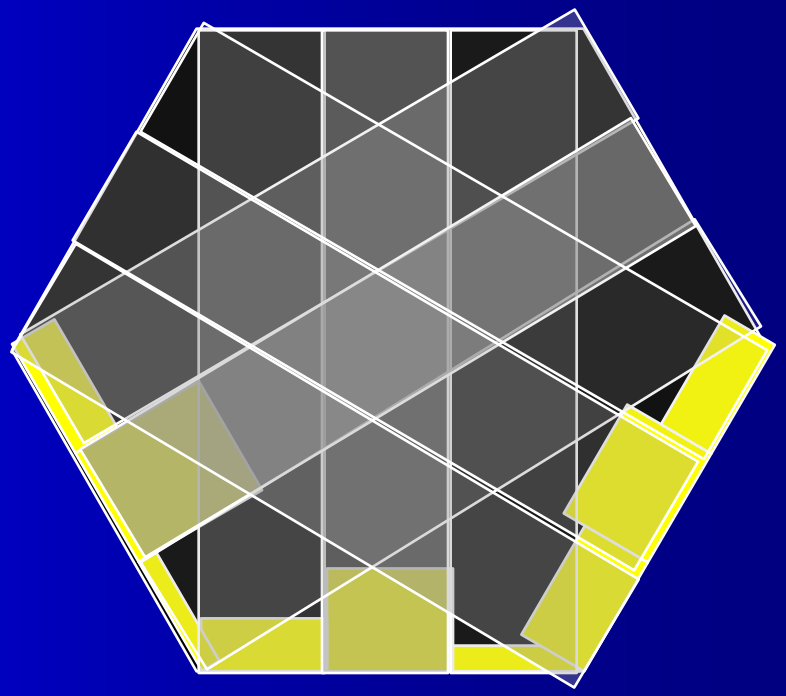


Bigger boys need big toys: AnyScan PET/CT



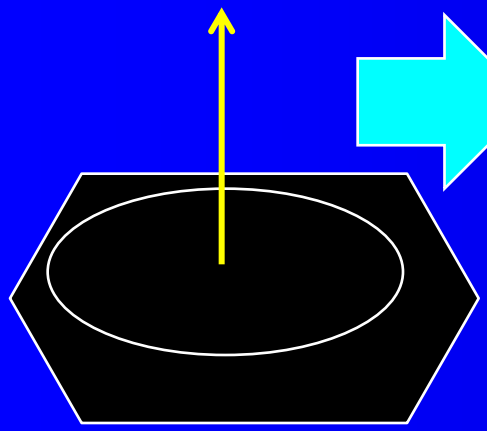
Tomográfiás rekonstrukció





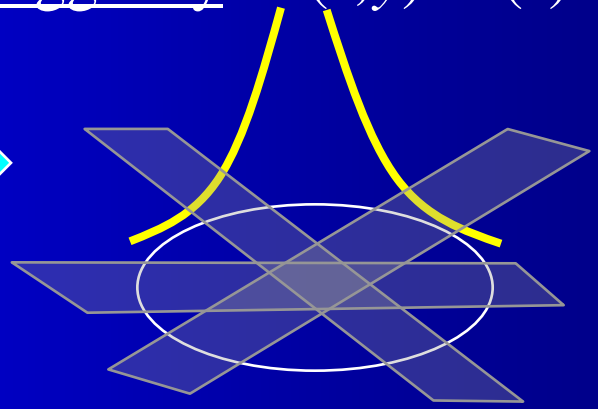
Szűrt visszavetítés (FBP=Filtered backprojection)

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



Mérés + Visszavetítés

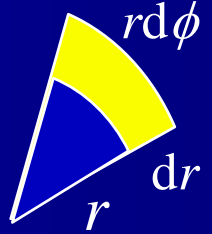
Súlyfüggvény: $w(x,y)=w(r) \sim 1/r$



R kör:

$$\iint w(x,y)\,\mathrm{d}x\mathrm{d}y=\int^{2\pi}\int^{R} w(r)\; r\mathrm{d}r\mathrm{d}\phi=$$

$$=2\pi\int^{R} w(r) r\,\mathrm{d}r \sim R$$



Korrekció Fourier térben:

$$o(x,y)=i(x,y)\otimes w(x,y) \Rightarrow \mathcal{F}_x\mathcal{F}_y\, o[\omega_x,\omega_y] = \mathcal{F}_x\mathcal{F}_y\, i[\omega_x,\omega_y]\cdot\mathcal{F}_x\mathcal{F}_y w[\omega_x,\omega_y]$$

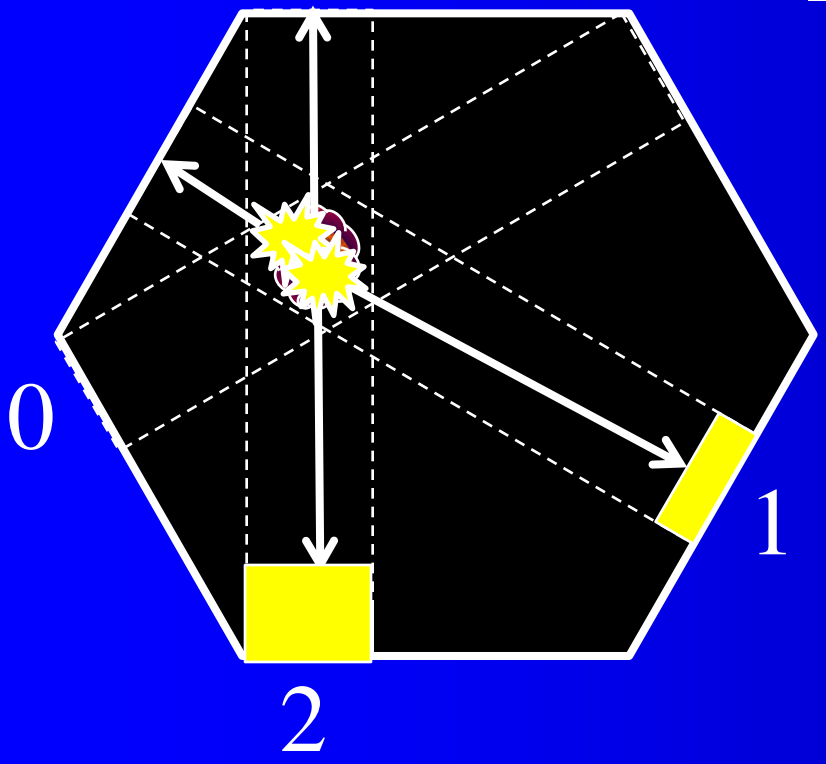
Rámpa szűrő

$1/|\boldsymbol{\omega}|$

$$\mathcal{F}_x\mathcal{F}_y\, i[\omega_x,\omega_y] = \mathcal{F}_x\mathcal{F}_y\, o[\omega_x,\omega_y]\cdot|\boldsymbol{\omega}|$$

Zaj!

Becsapódások véletlen események!

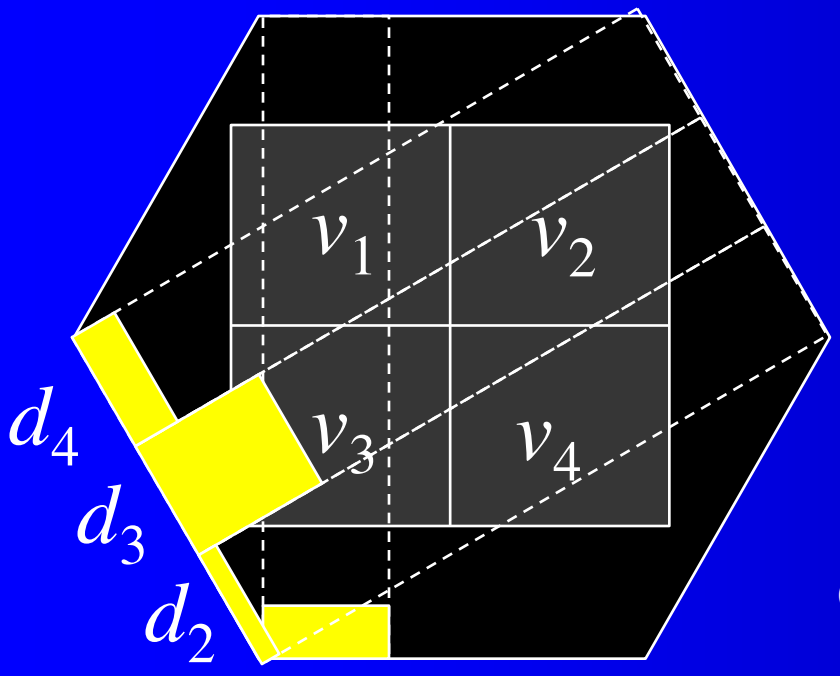


Emissziós tomográfia:
Poisson eloszlás

$$P\{d\} = \frac{\lambda^d}{d!} \cdot e^{-\lambda}$$

Nagy számok [illegible]vénye?

Algebrai visszavetítés



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

Lin egyenlet (V<D):

$$\underset{(D)}{\boldsymbol{d}} = \underset{(D\times V)}{\boldsymbol{A}} \cdot \underset{(V)}{\boldsymbol{v}},$$

$\boldsymbol{d} = [d_1, \ldots, d_D]$

$\boldsymbol{v} = [v_1, \ldots, v_V]$

Moore féle pseudo-inverz:

$$\underset{(V\times D)}{\boldsymbol{A}^T} \cdot \underset{(D)}{\boldsymbol{d}} = \underbrace{\boxed{\boldsymbol{A}^T \cdot \boldsymbol{A}}}_{(V\times V)} \cdot \boldsymbol{v}$$

$$\boxed{\boldsymbol{v} = (\boldsymbol{A}^T \cdot \boldsymbol{A})^{-1} \cdot \boldsymbol{A}^T \cdot \boldsymbol{d} = \boldsymbol{A}^+ \cdot \boldsymbol{d}}$$

Expectation Maximization



Amelyik éppen a mérési eredményt maximálja $P(\boldsymbol{d}|\boldsymbol{v})$

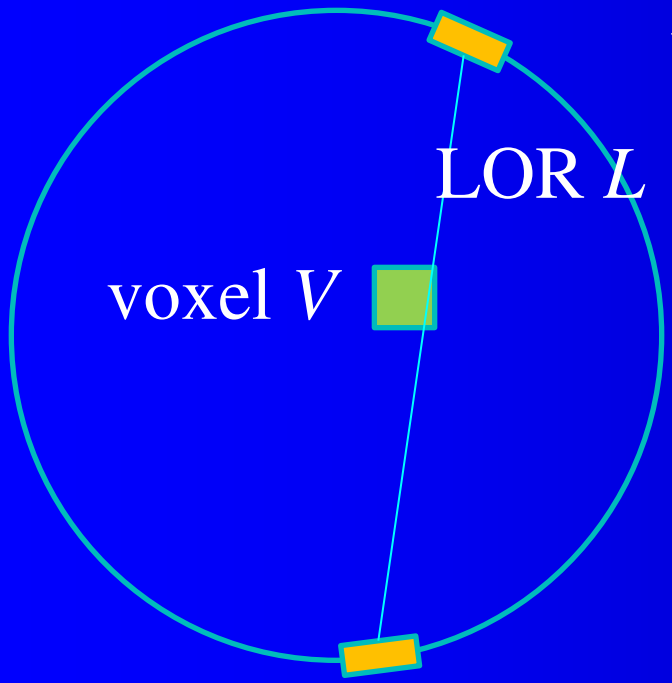
- Likelihood maximalizálás: log$P$
- Előny: a mérés statisztikai modelljét is figyelembe veszi

PET: iteratív séma

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$$

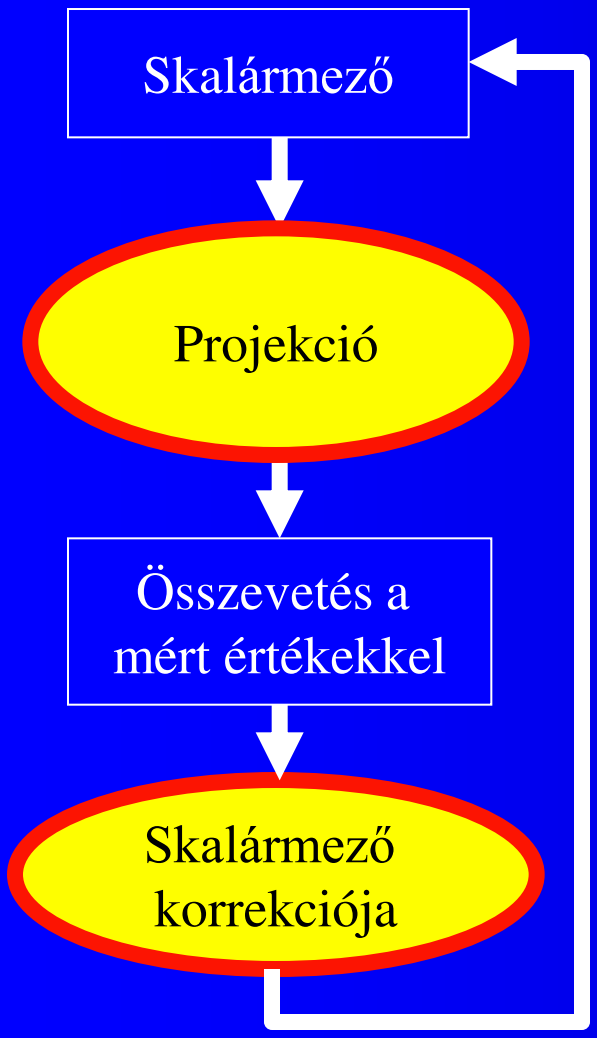
$$\mathbf{x}_V^{(n+1)} = \mathbf{x}_V^{(n)} \frac{\sum_L \mathbf{A}_{LV} \cdot \frac{\mathbf{y}_L}{\sum_{V'} \mathbf{A}_{LV'} \mathbf{x}_V^{(n)}}}{\sum_L \mathbf{A}_{LV}}$$

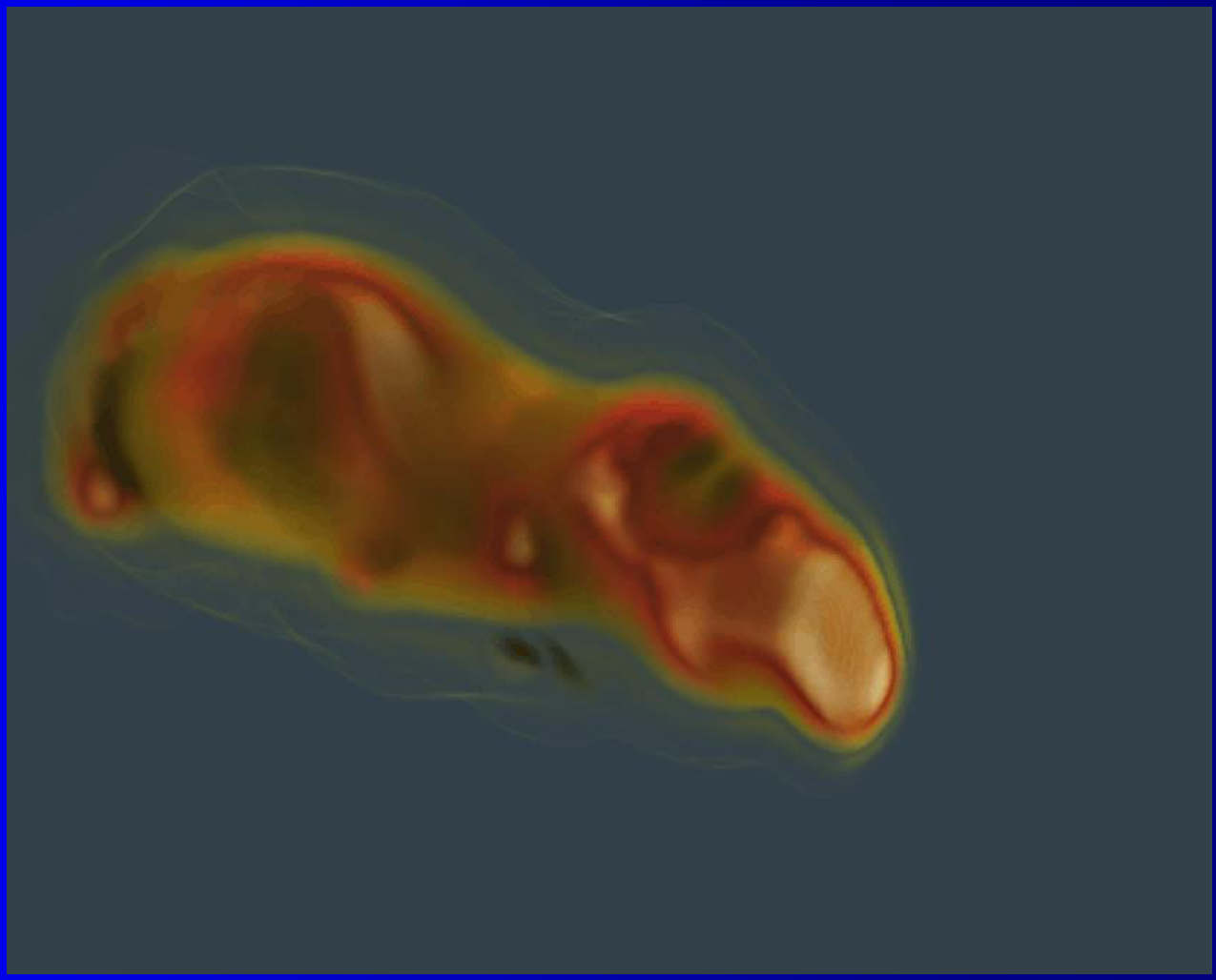
Poisson és pozitivitási kényszer



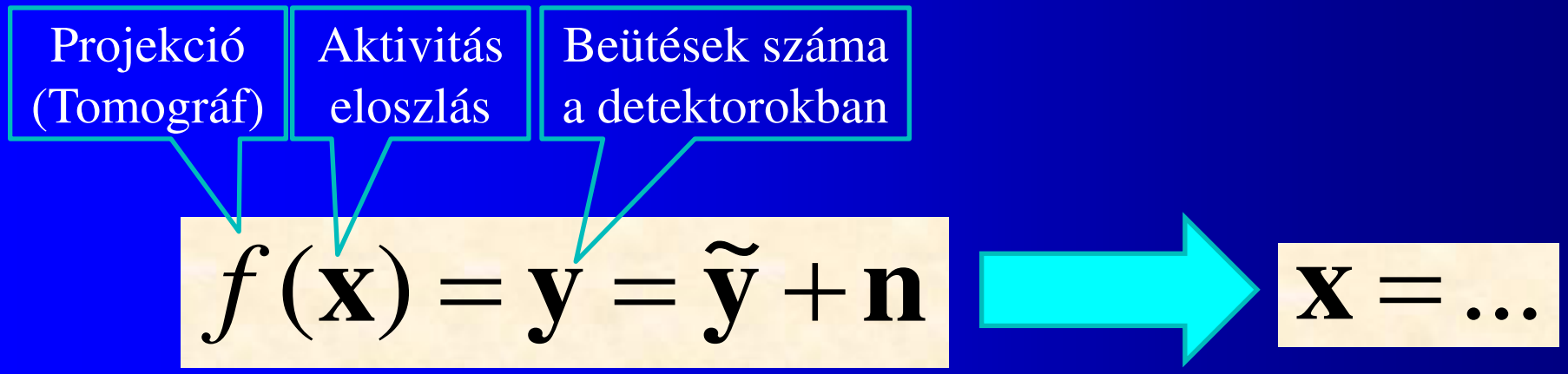
$\mathbf{A}_{LV}$: annak valószínűsége, hogy a V voxelben bekövetkezett bomlást az L LOR detektálja

Iteratív séma





Inverz problémák



- Ill-posed: nincs megoldás vagy nem egyértelmű.
- Nem ismerjük **n**-et (zaj).
- Közelítő megoldásokból melyik (zaj hatása)?
- Plusz információ bevitele: **regularizáció**

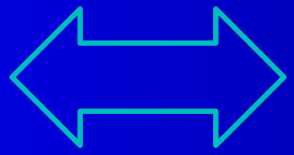
Megoldási sémák

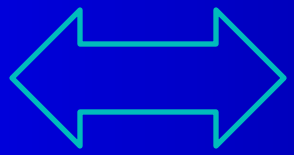
Algebrai

$$\min \left\| f(\mathbf{x}) - \mathbf{y} \right\|$$

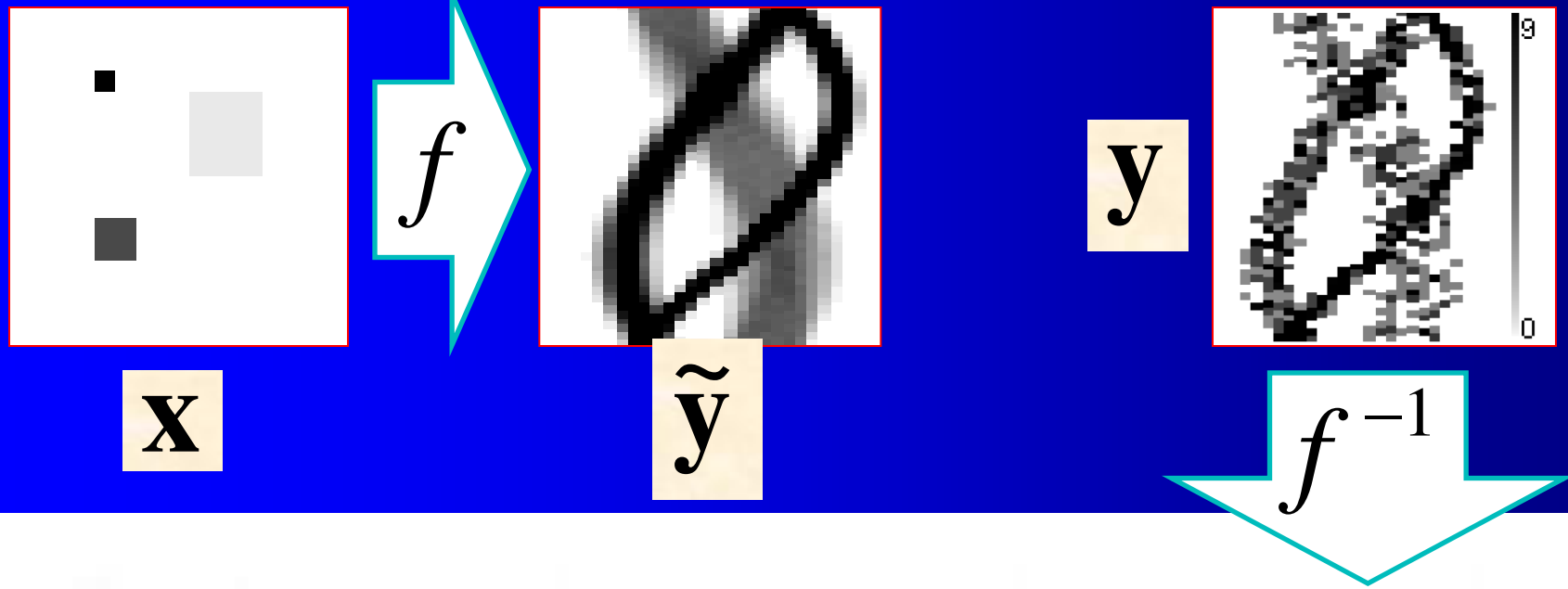
Valószínűségi

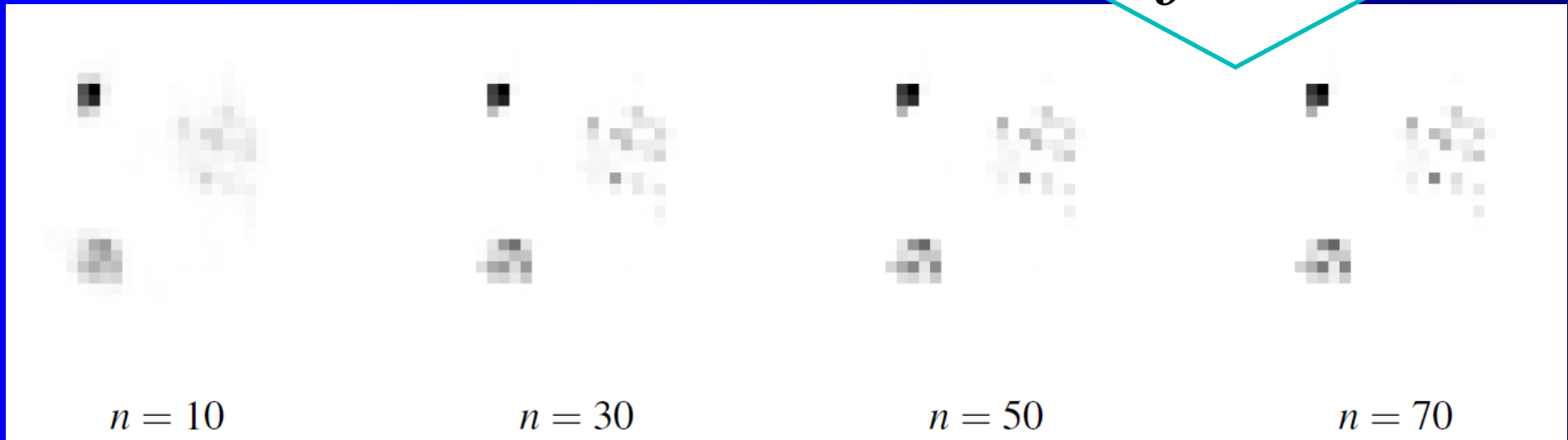
$$\max \left\{ \log P(\mathbf{y} \mid \mathbf{x}) \right\}$$

2-es norma  **y** Gauss eloszlású

Kullback-Leibler divergencia  **y** Poisson eloszlású

Overfitting





Regularizáció

Likelihood

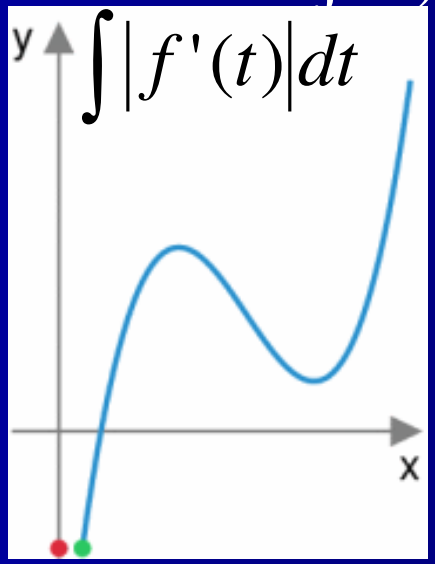
Regularizációs paraméter

Zajos megoldások büntetése

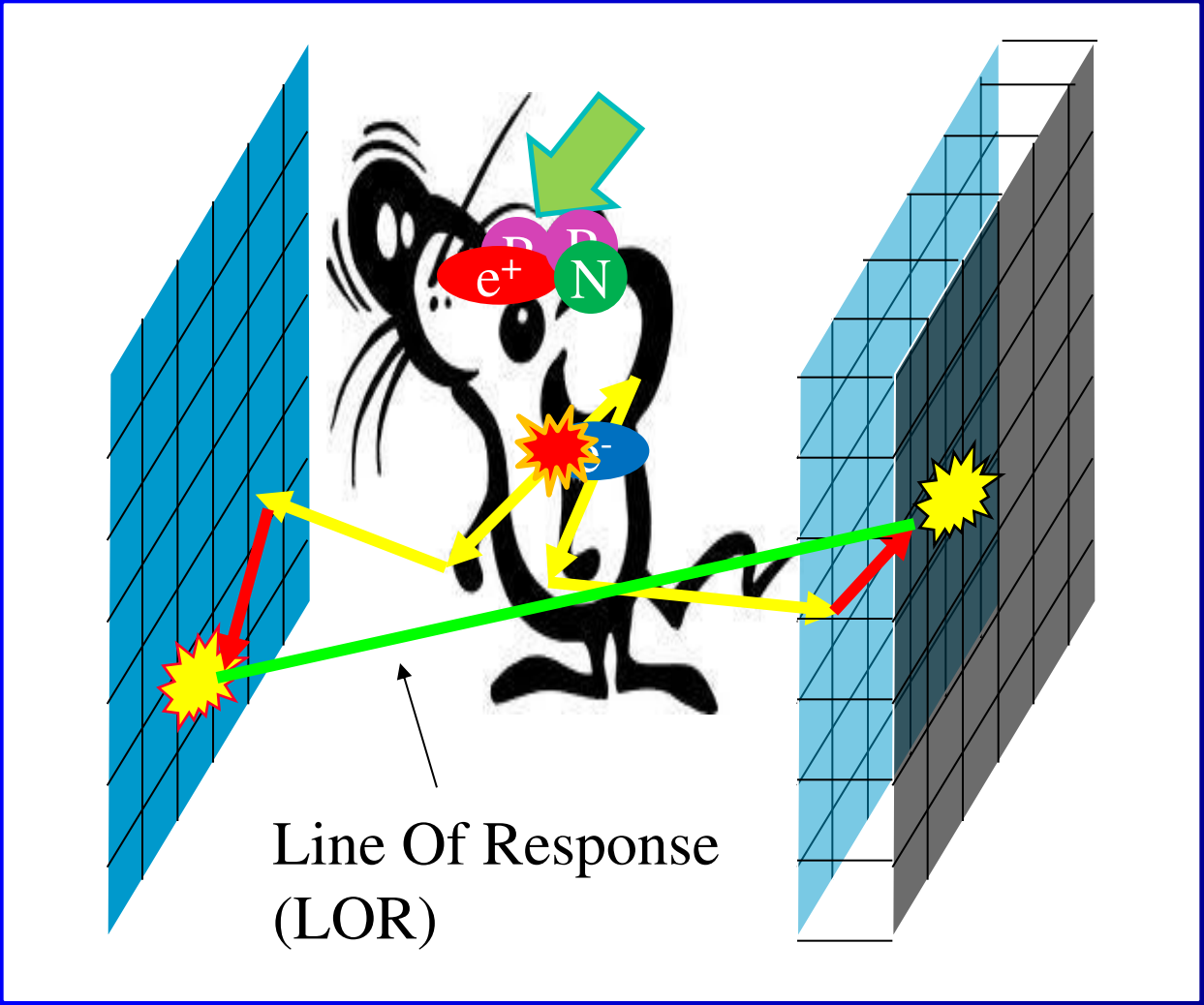
$$\min\left\{-\log P(\mathbf{y}\mid\mathbf{x})+\lambda R(\mathbf{x})\right\}$$

- $R(\mathbf{x})$ „rossz” megoldásoknál nagy, „jó” megoldásoknál kicsi (tökéletes megoldást ne büntesd, de nem ismerjük), konvex függvény.
  - **Teljes variáció (TV)**:

$$R(x)=\int_V |\nabla x(v)|\mathrm{d}v$$

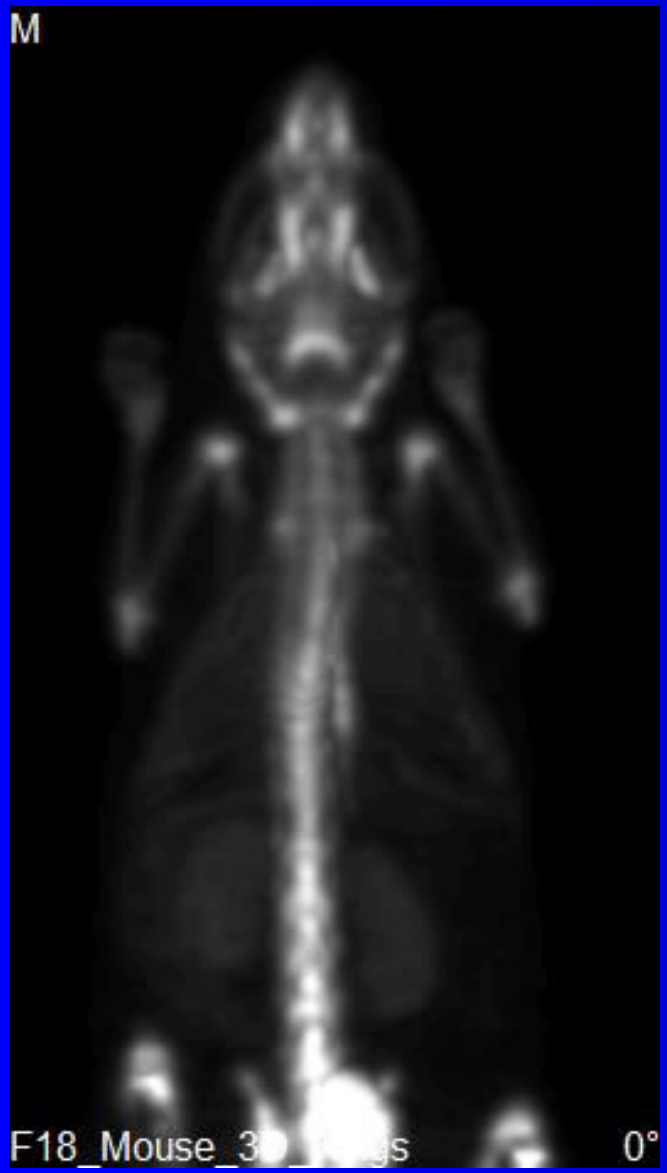


PET

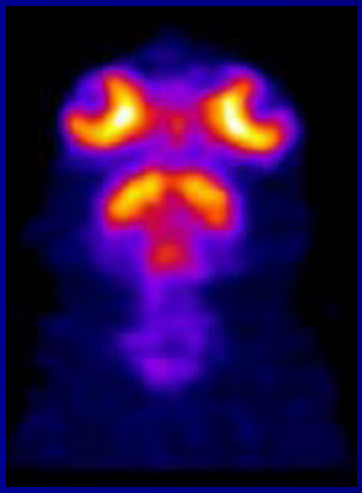


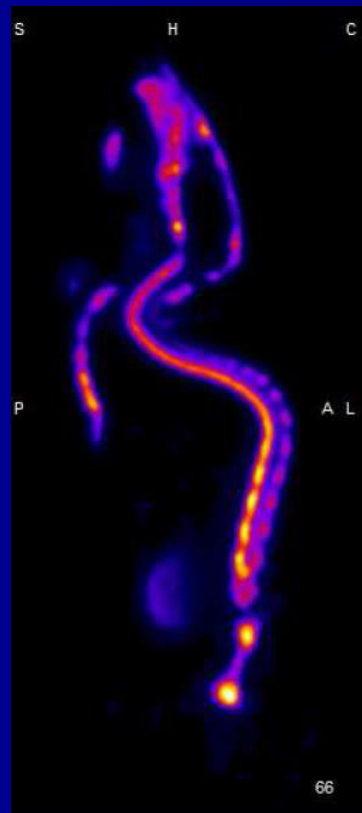
Nincs szóródás

324x315x315 voxels

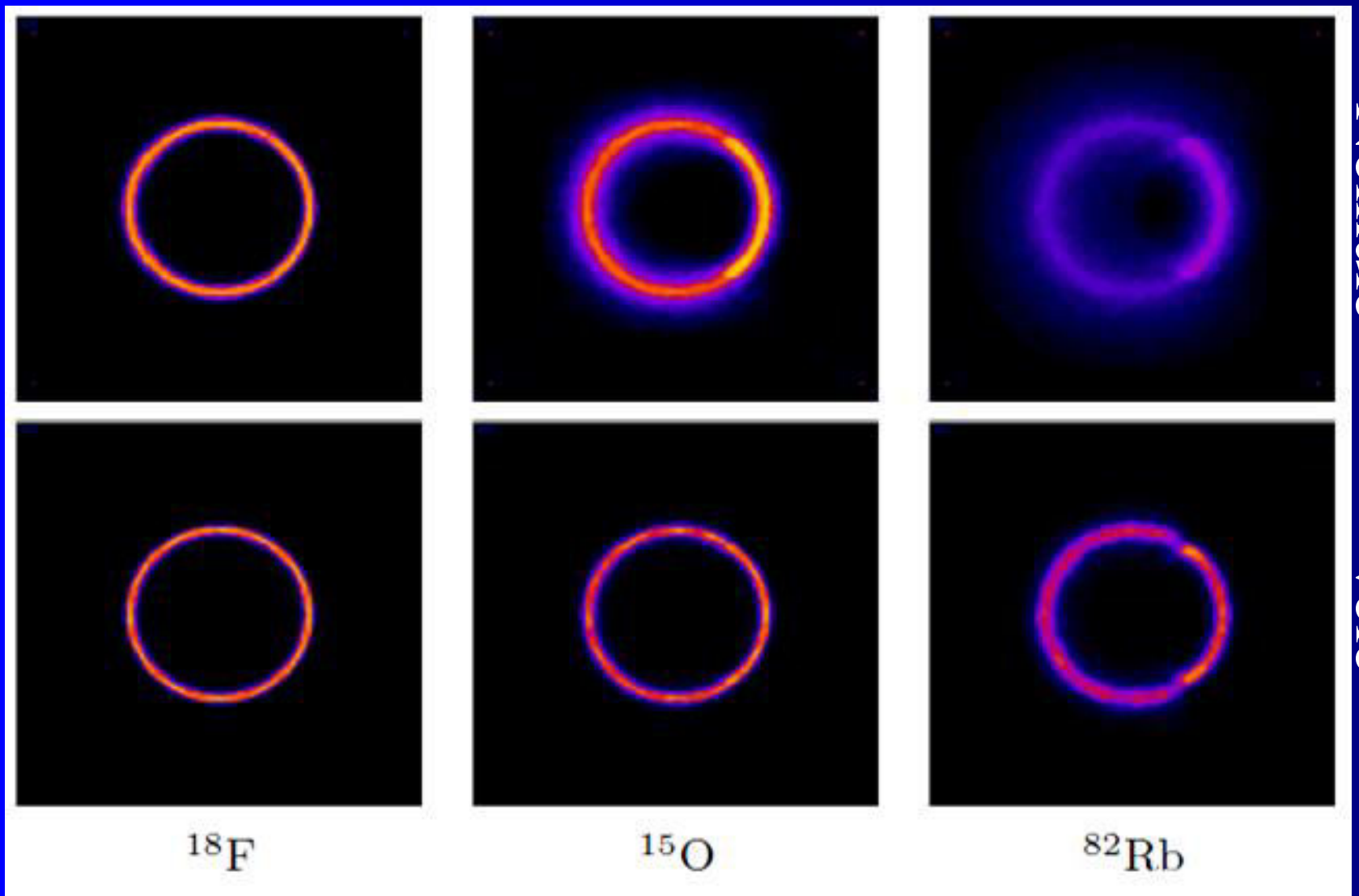




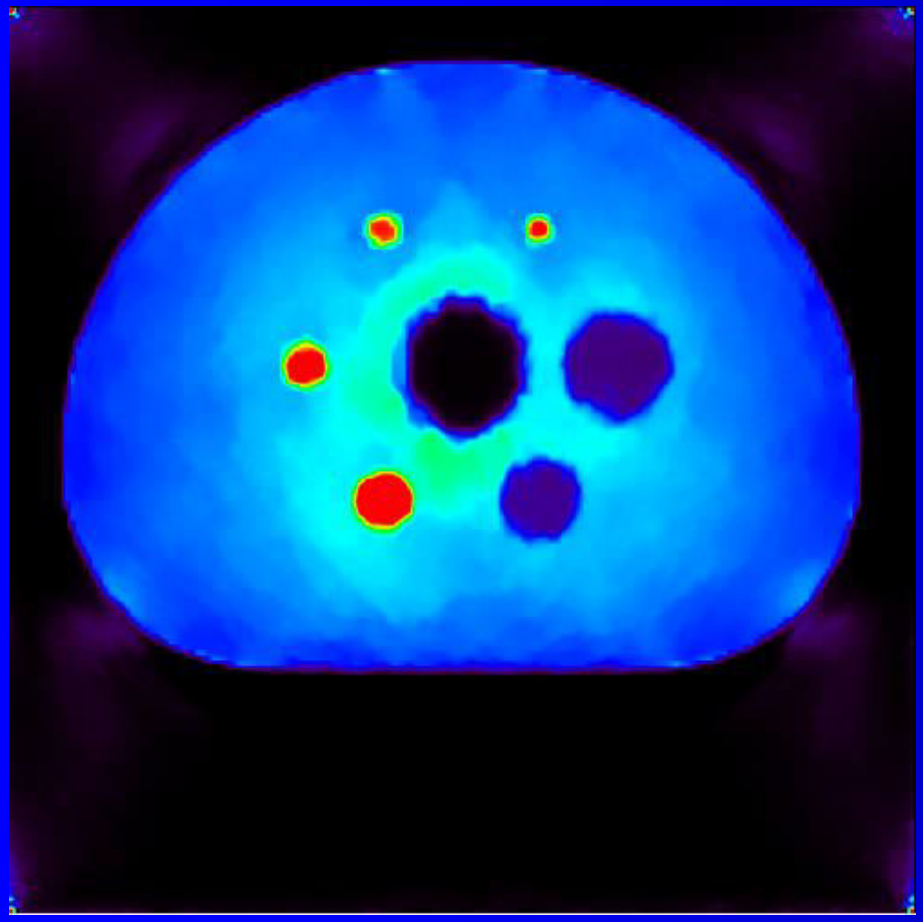




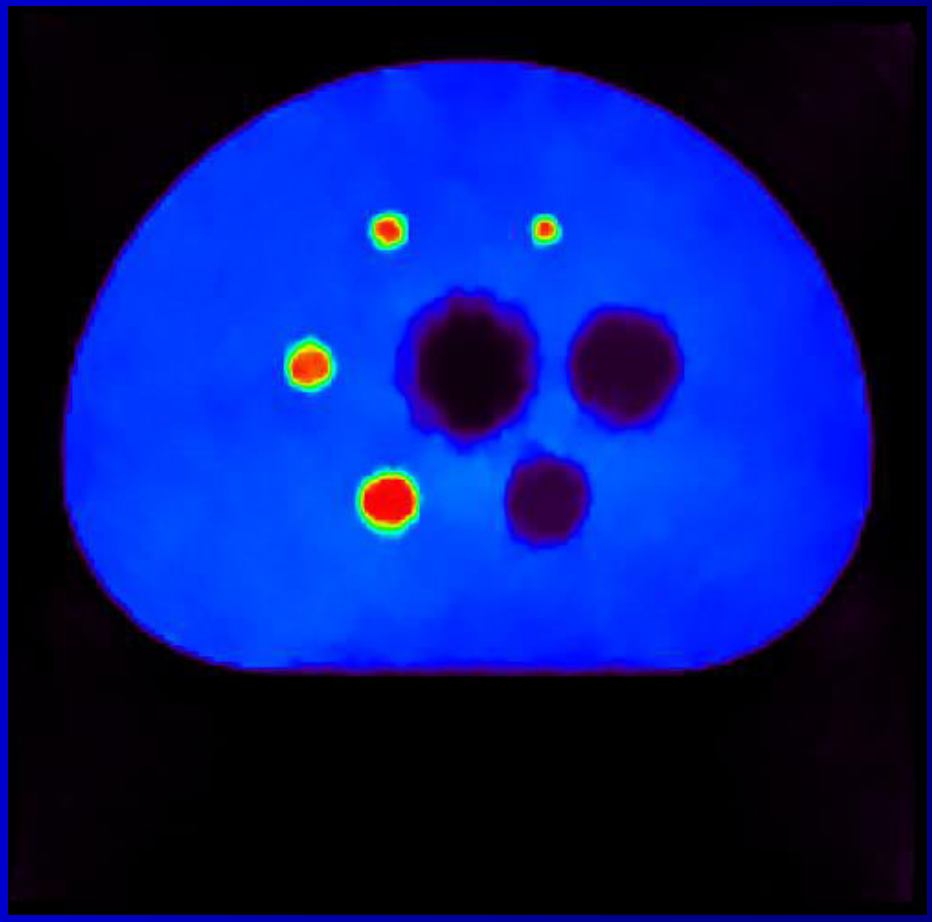
Positron range



Szóródás a testben



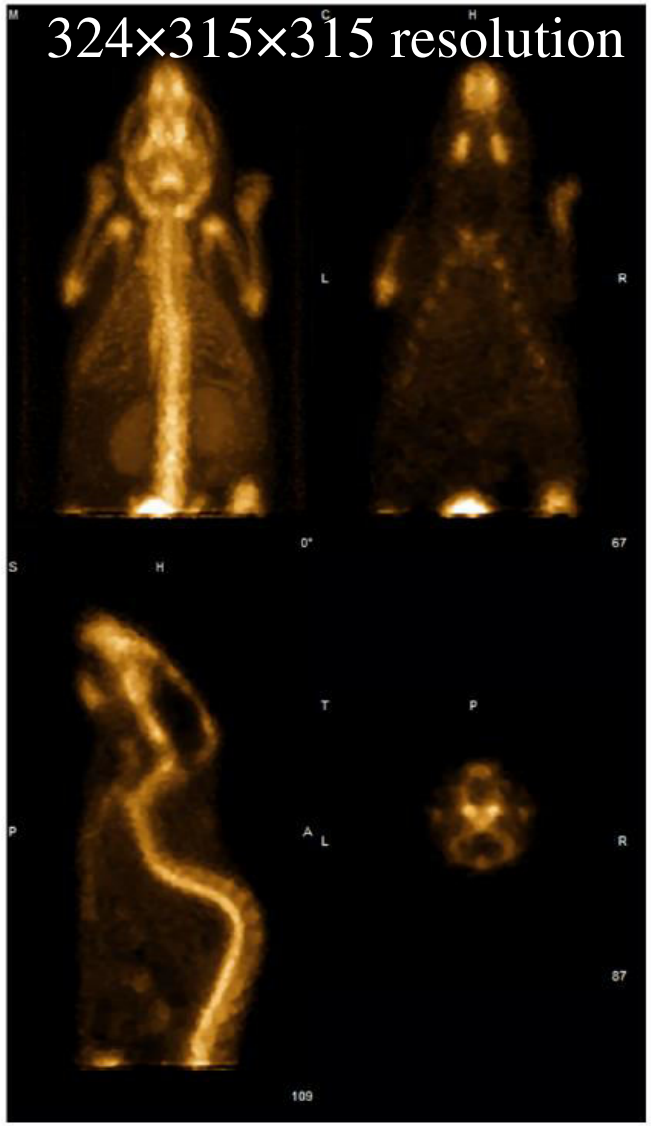
Nélküle

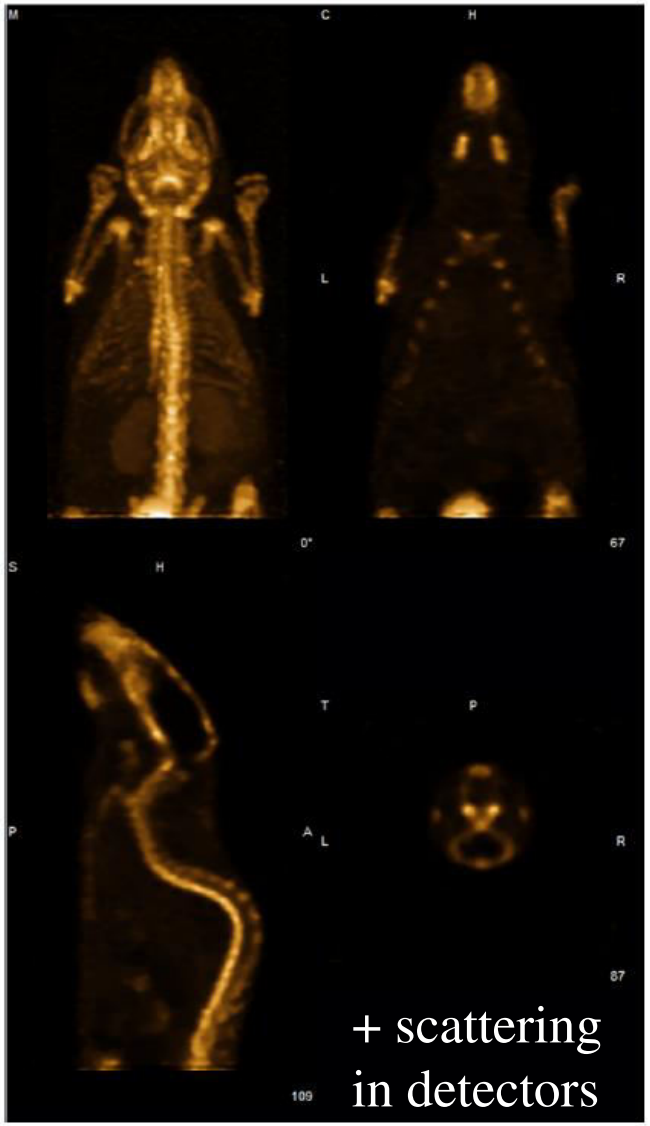


Vele

Szóródás a detektorban

. . .







Fraktálok

Szirmay-Kalos László

Fraktálok

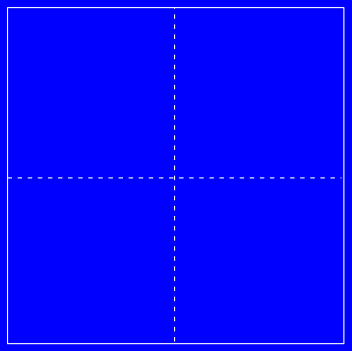
Hausdorff dimenzió

N = 2



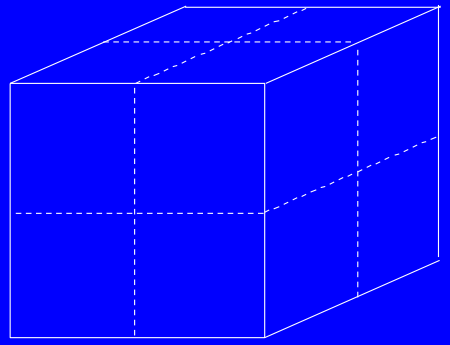
$r = \frac{1}{2}$

N = 4



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

N = 8



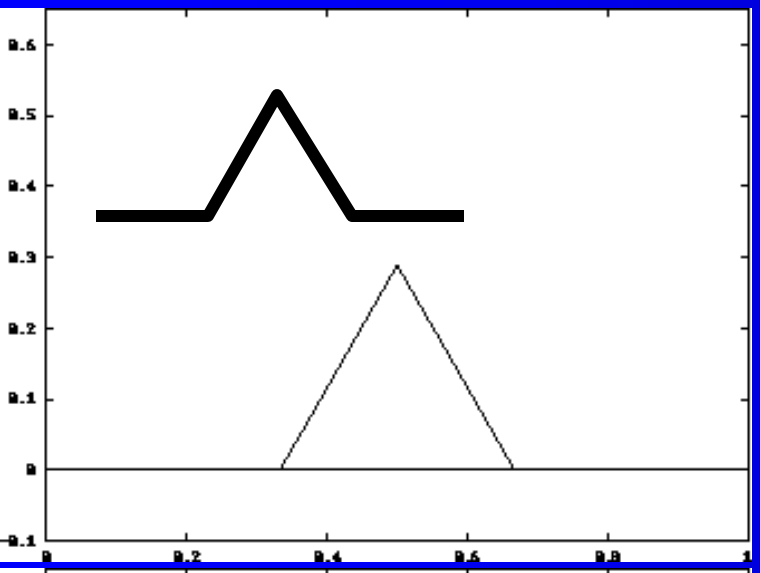
$r = \frac{1}{2}$

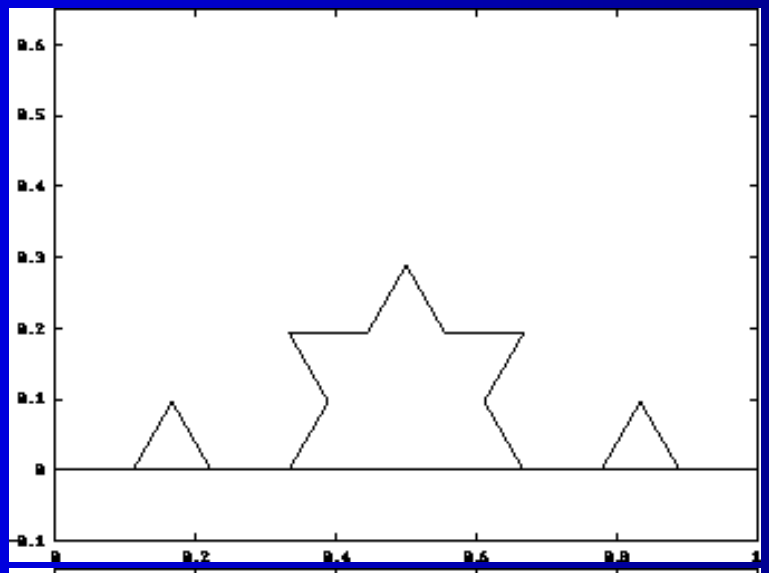
$N = 1/r^D$

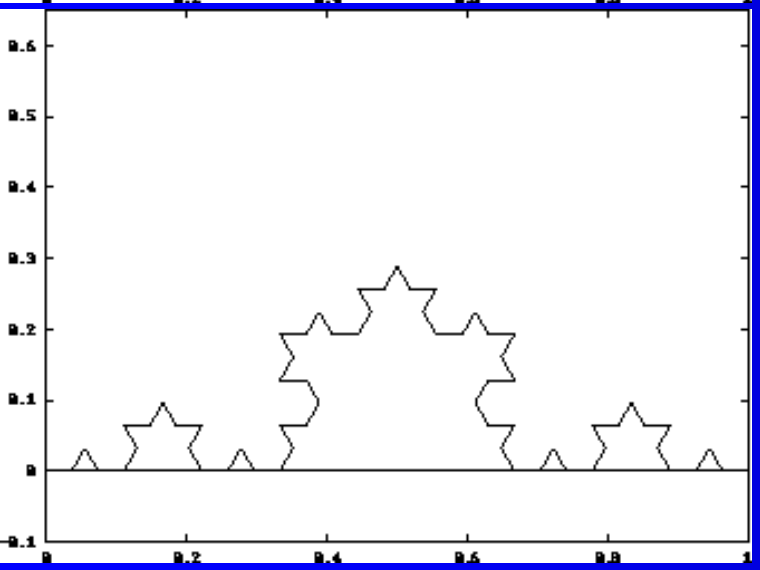
$D = (\log N) / (\log 1/r)$

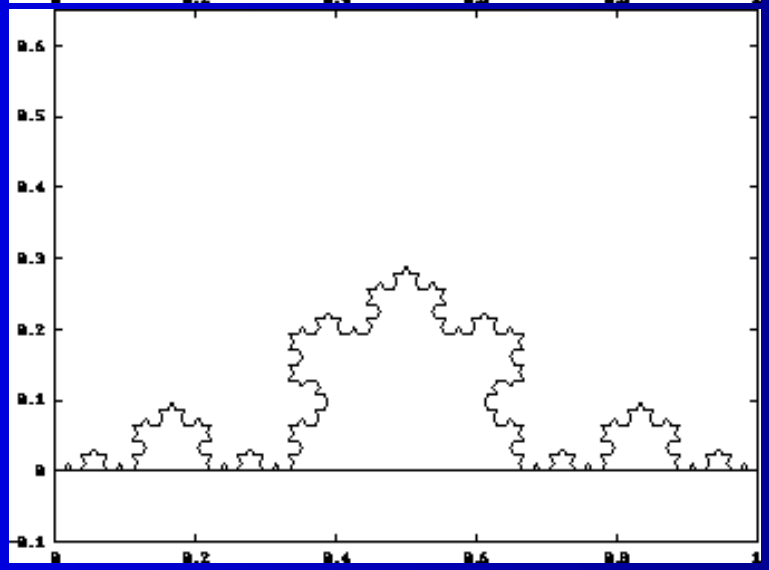
Koch görbe

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



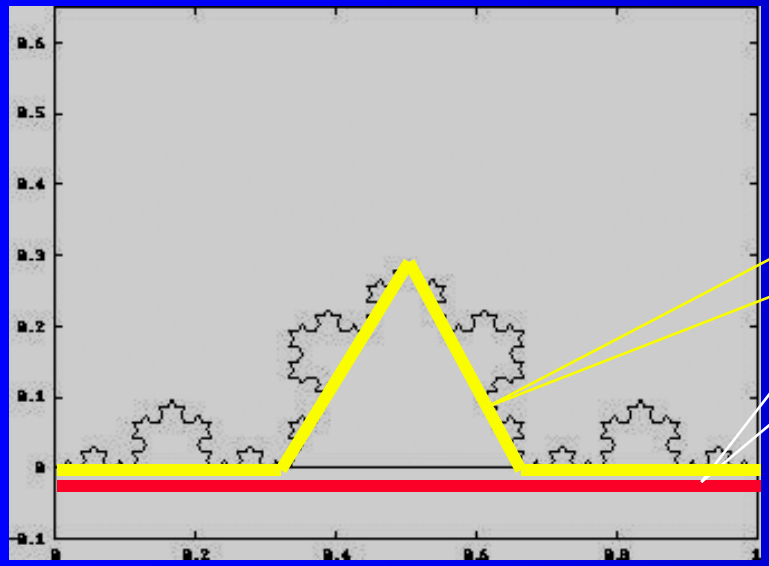






N = 4,
r = 1/3

Nem önhasonló objektumok dimenziója

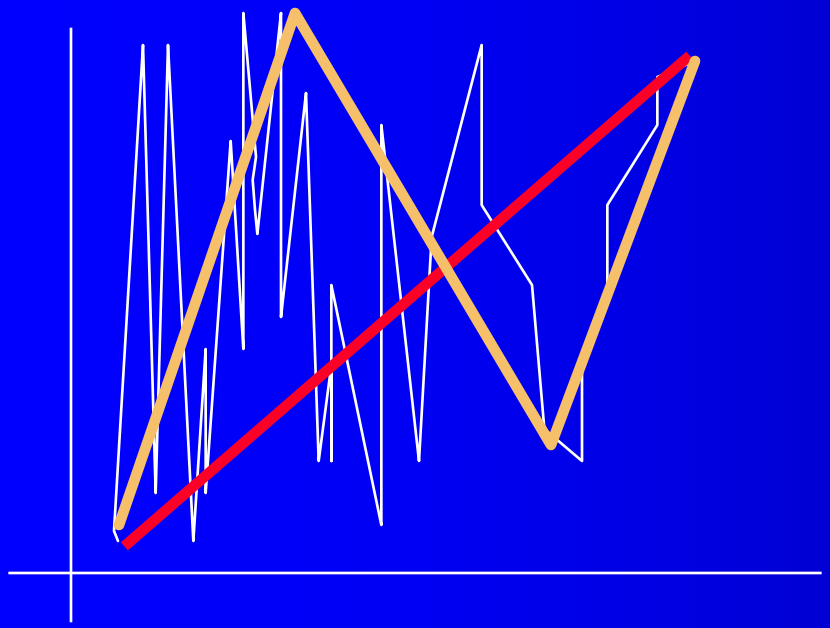


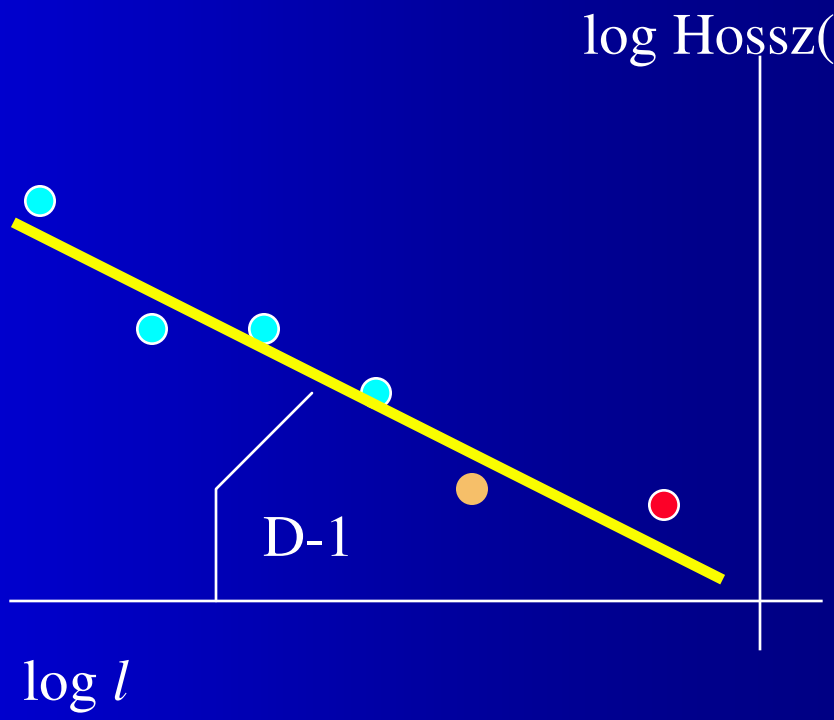
Vonalzó (l)	db
1	1
r =1/3	N = 4
r^2	N^2
r^m	N^m

$$\text{Hossz}(l) = l\,\text{db} = l\,N^m = l\,(1/r^D)^m = l\,(1/r^m)^D = 1/l^{D-1}$$

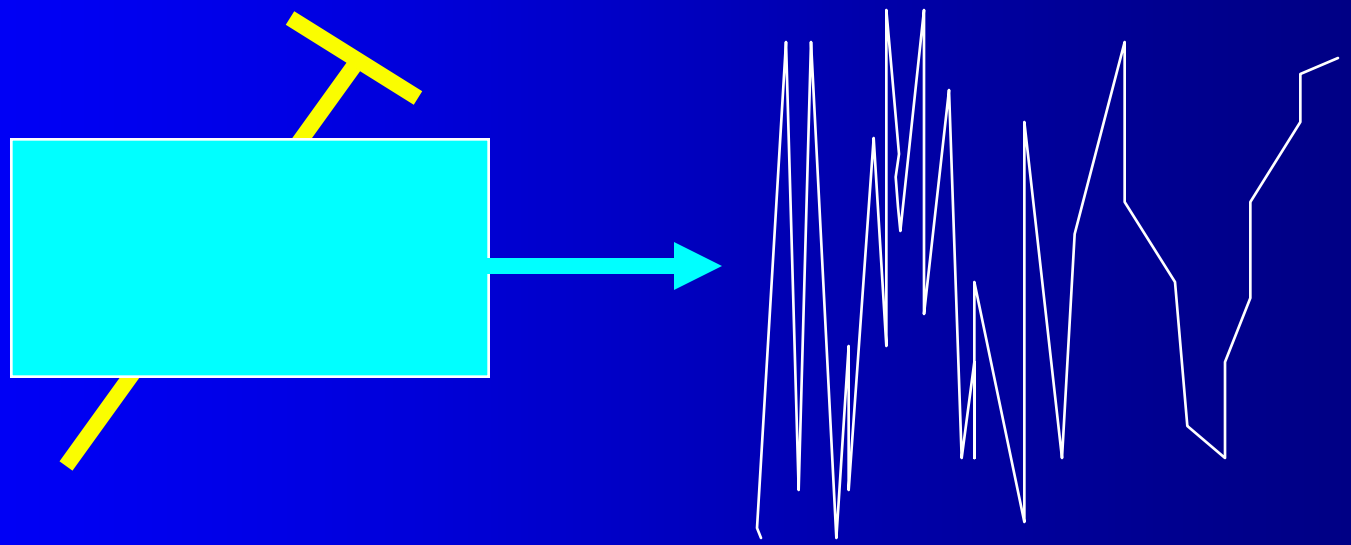
$$D = -\log \text{Hossz}(l) / \log l + 1$$

Dimenziómérés = hosszmérés





Fraktálok előállítása



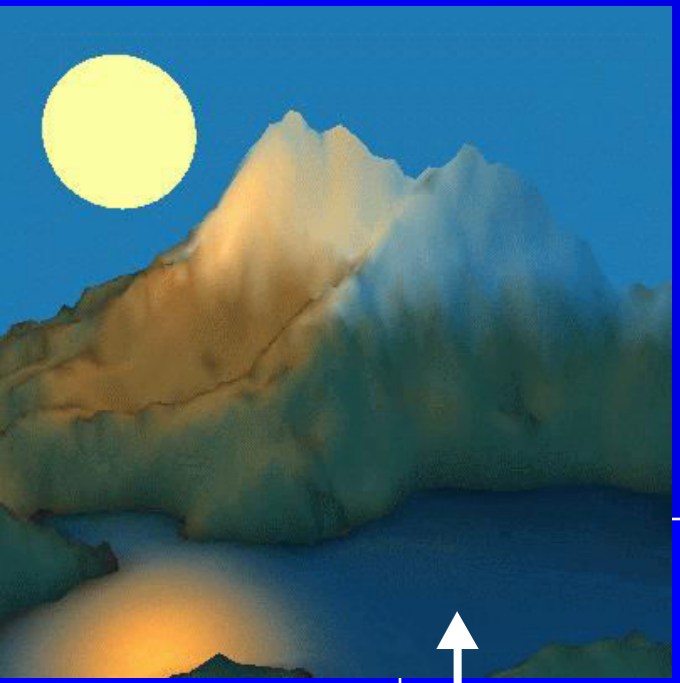
Matematikai gépek:

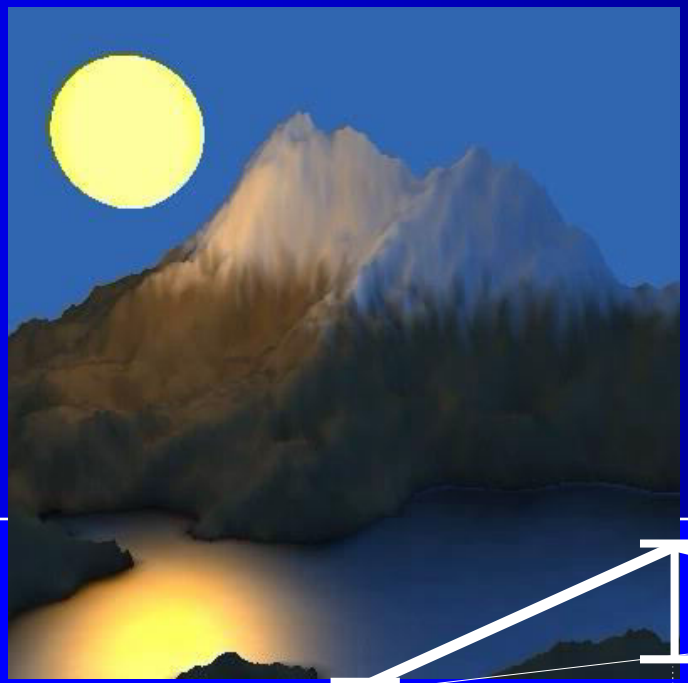
- Brown mozgás
- Kaotikus dinamikus rendszerek

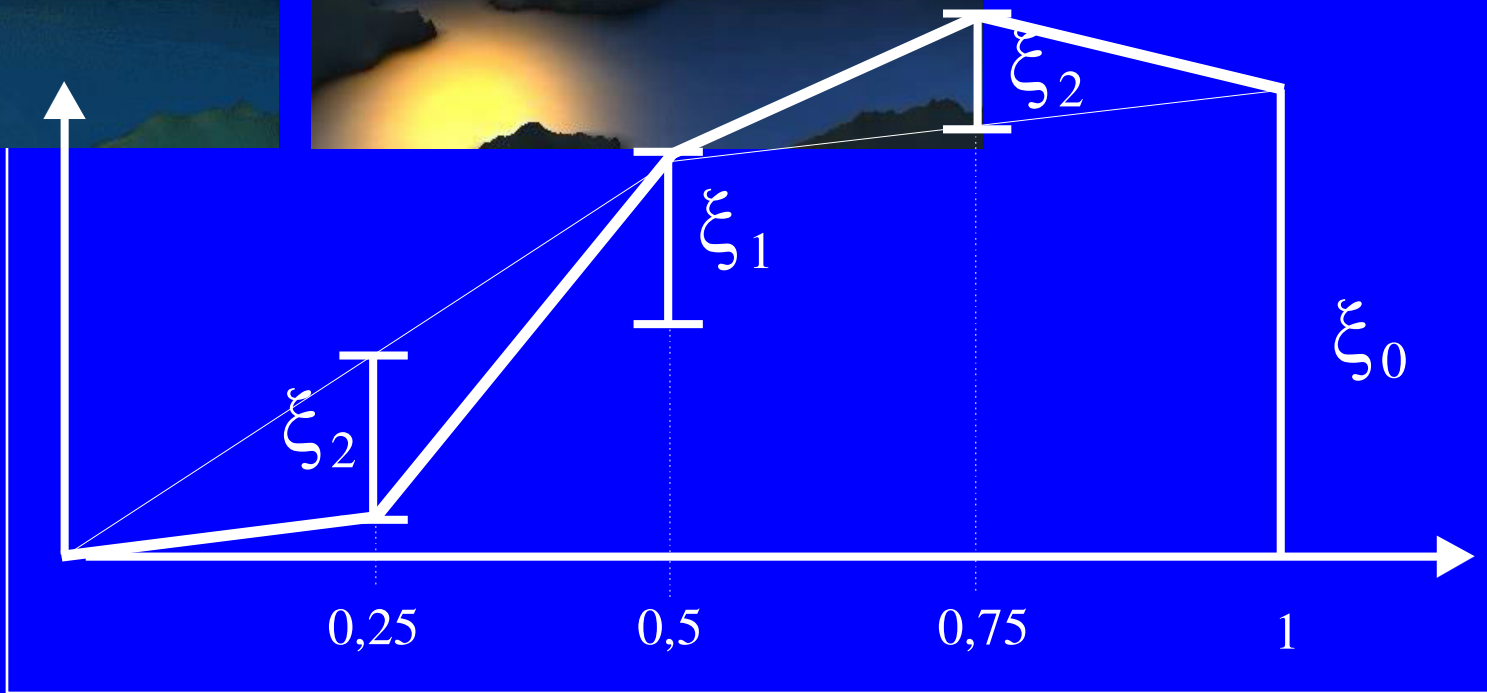
Brown mozgás - Wiener féle sztochasztikus folyamat

- Sztochasztikus folyamat (véletlen függvény)
- Trajektóriák folytonosak
- Független növekményű folyamat
- Növekmények 0 várható értékű normális eloszlás:
  - a független növekményűségből, a szórás az intervallum hosszával arányos

Brown mozgás alkalmazása

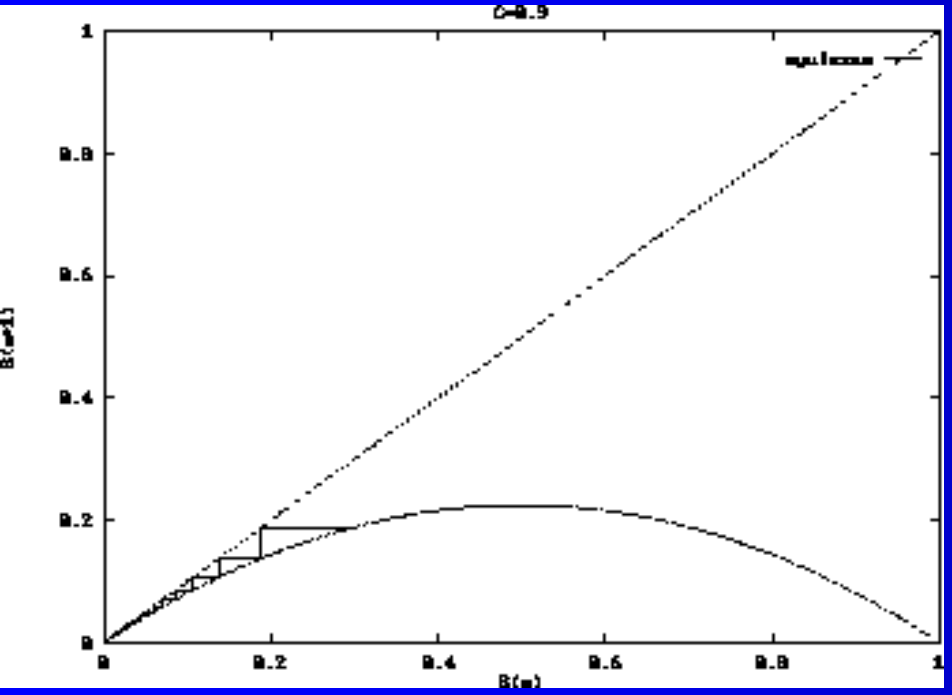


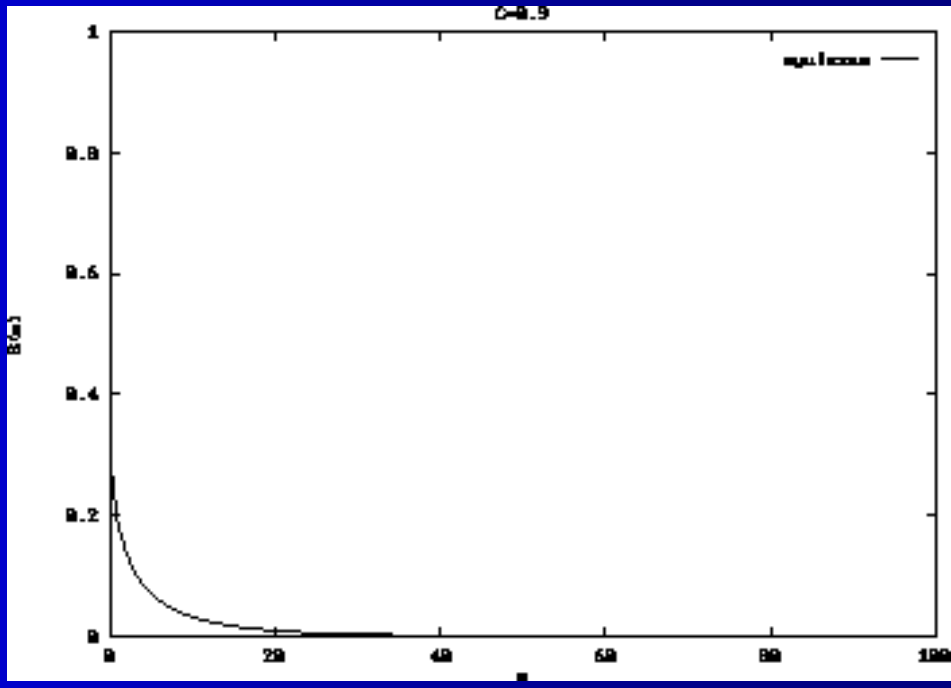




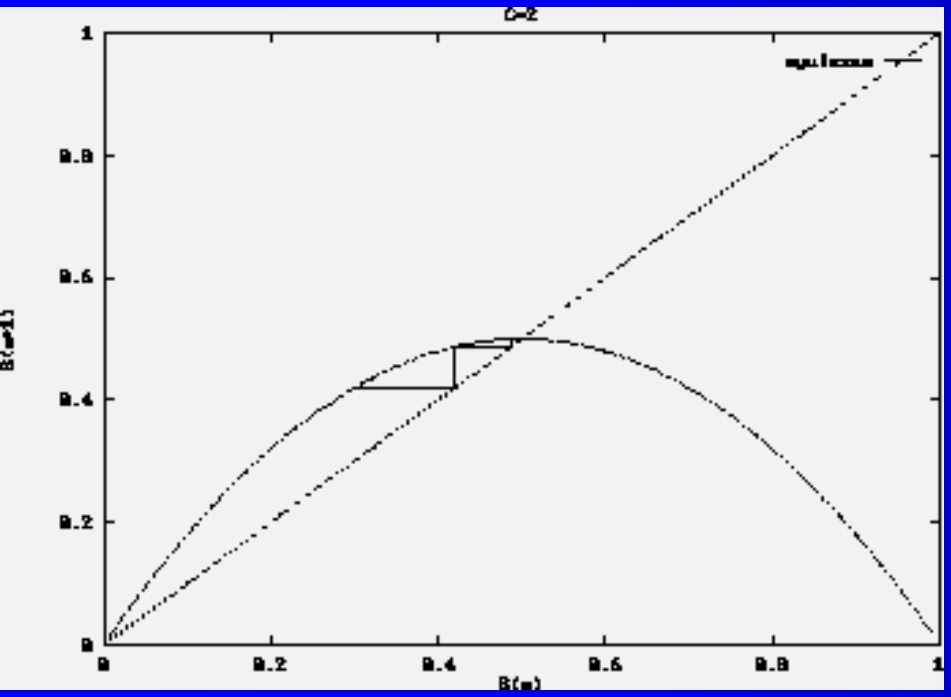
Kaotikus dinamikus rendszer: nyulak kis *C* értékre

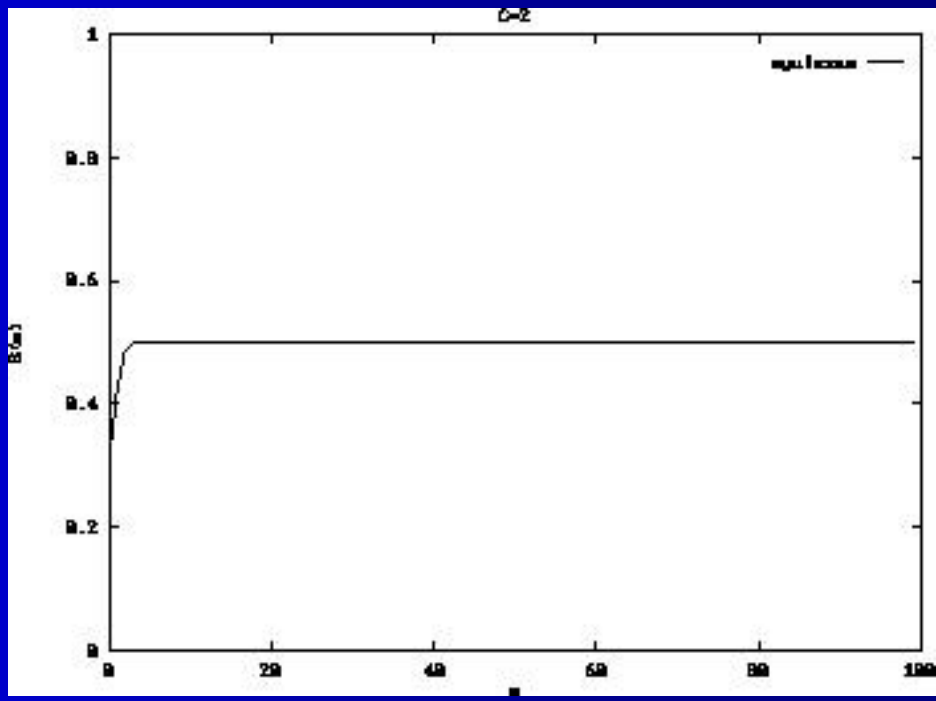
$$S_{n+1} = C\, S_n\, (1-S_n)$$



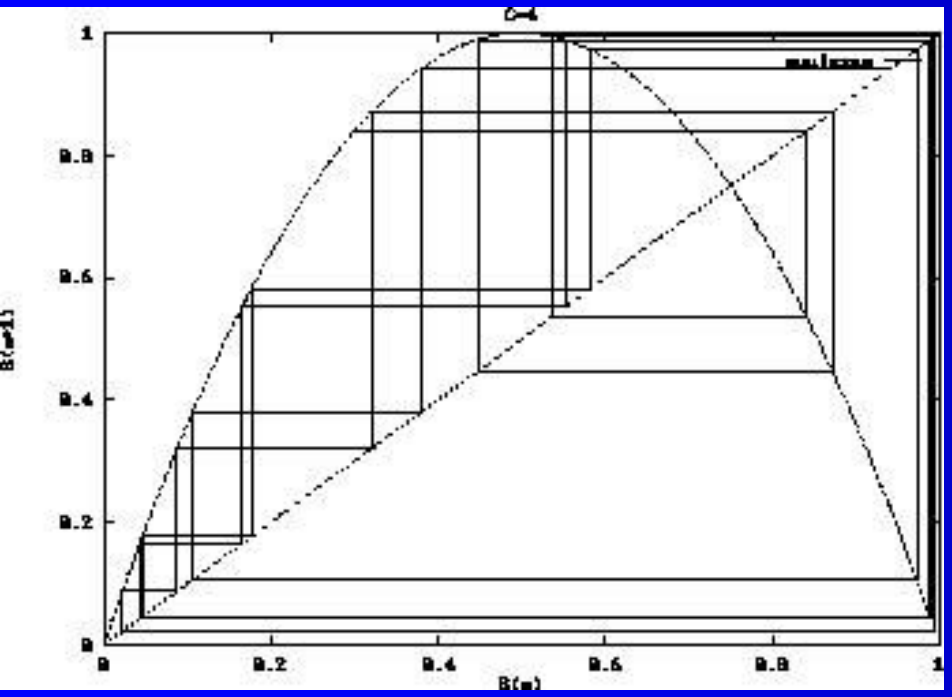


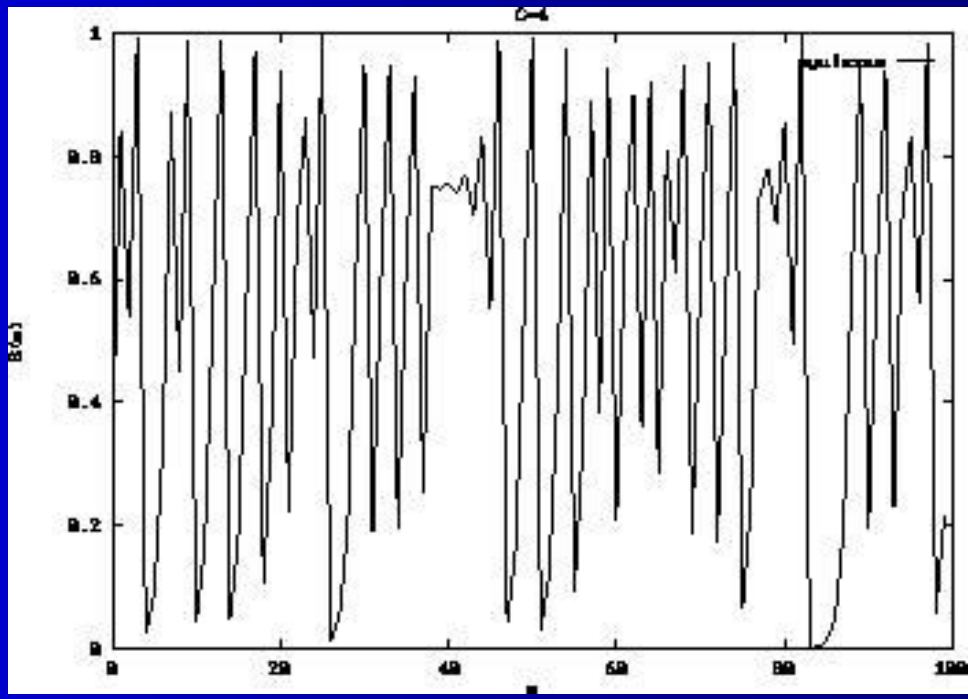
Kaotikus dinamikus rendszer: nyulak közepes *C* értékre





Kaotikus dinamikus rendszer: nyulak nagy *C* értékre



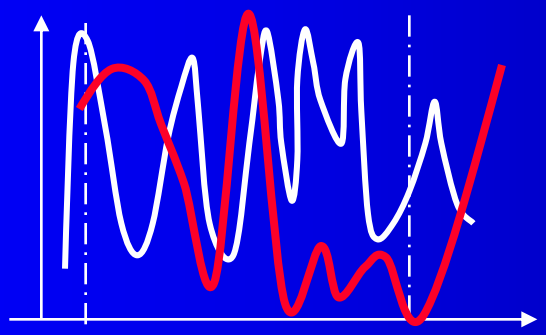


Pseudo véletlenszám generátor

- Iterált függvény:

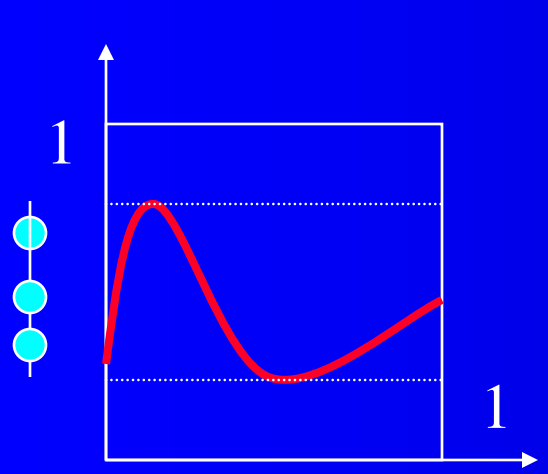
$$r_{n+1} = F(r_n)$$

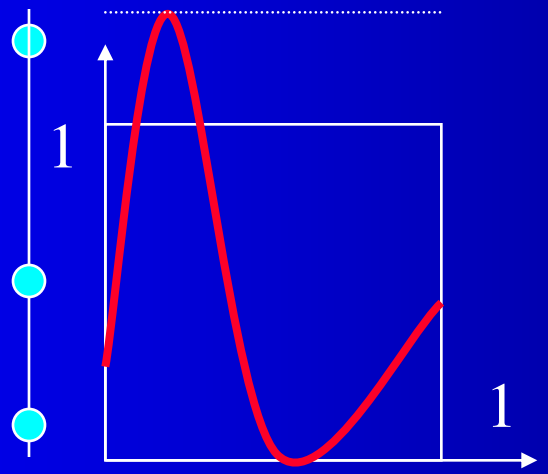
- véletlenként hat

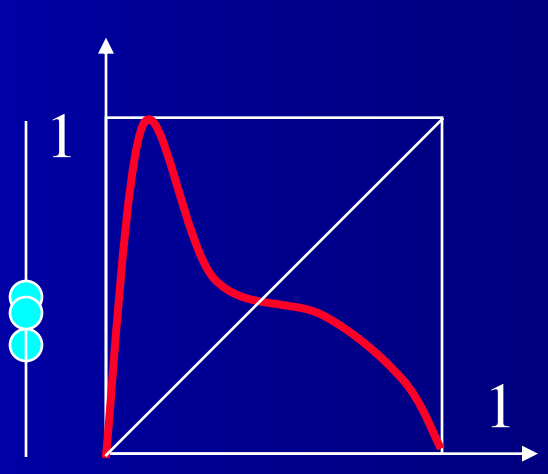


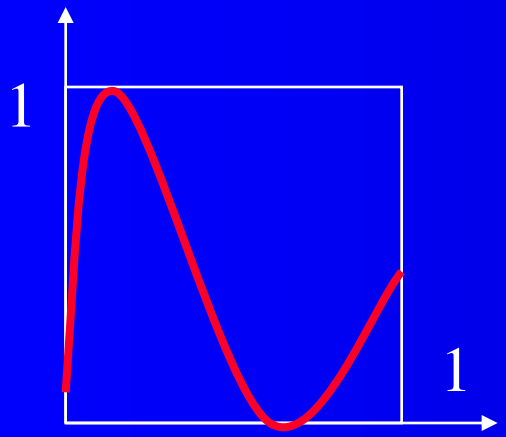
F nagy derivált!

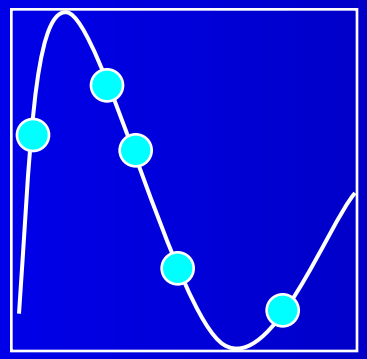
Rossz: $r_{n+1} = F(r_n)$







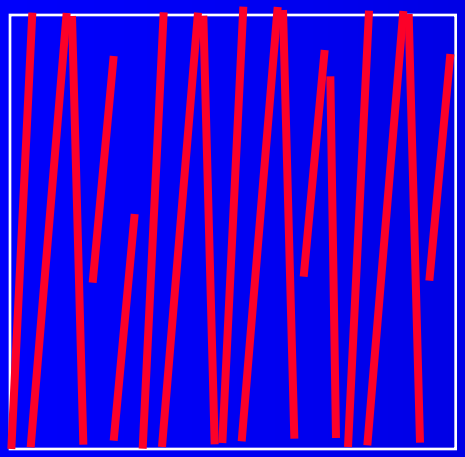


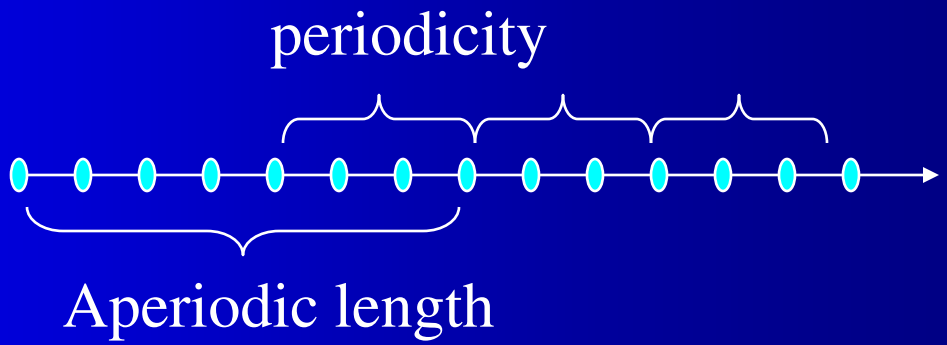


(r_n, r_{n+1}) pairs

Jó *F*

- Sűrűn kitölti a négyzetet
- Mindenütt nagy derivált
- a [0, 1]-ben van



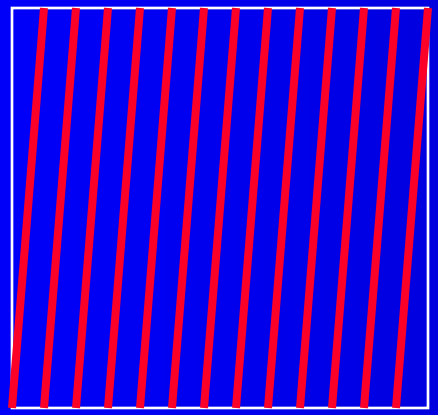


Kongruens generátor

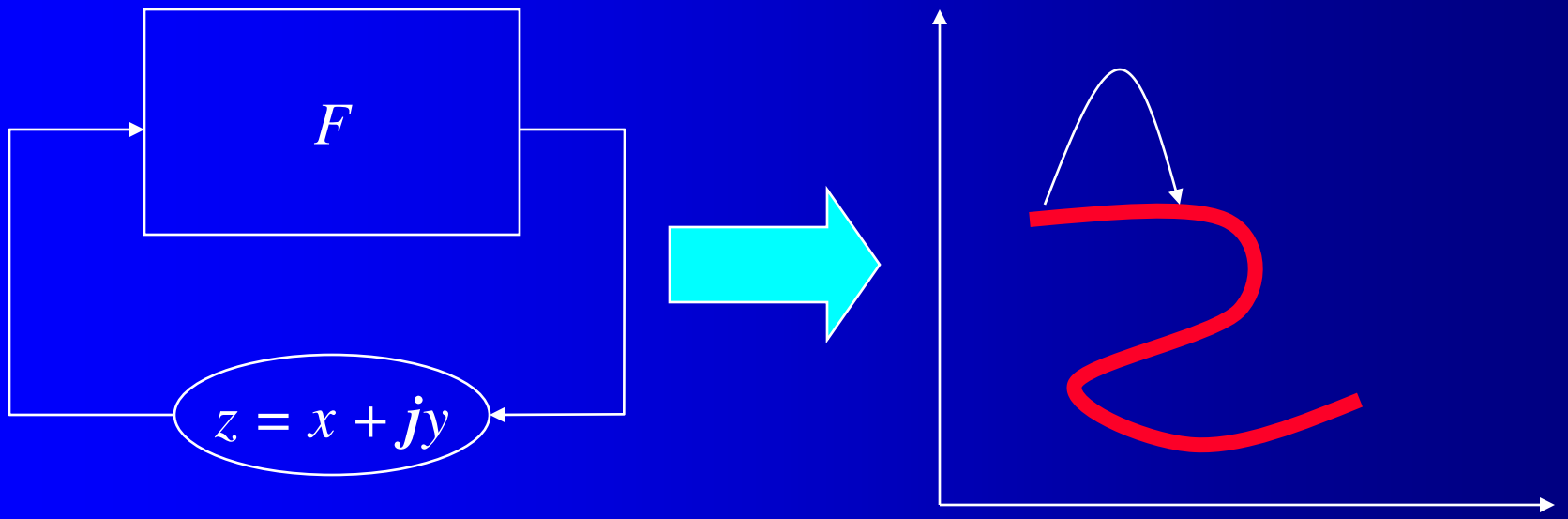
$$F(r) = \{\ g \cdot r + c\ \}$$

$g \cdot r+c$ tört része

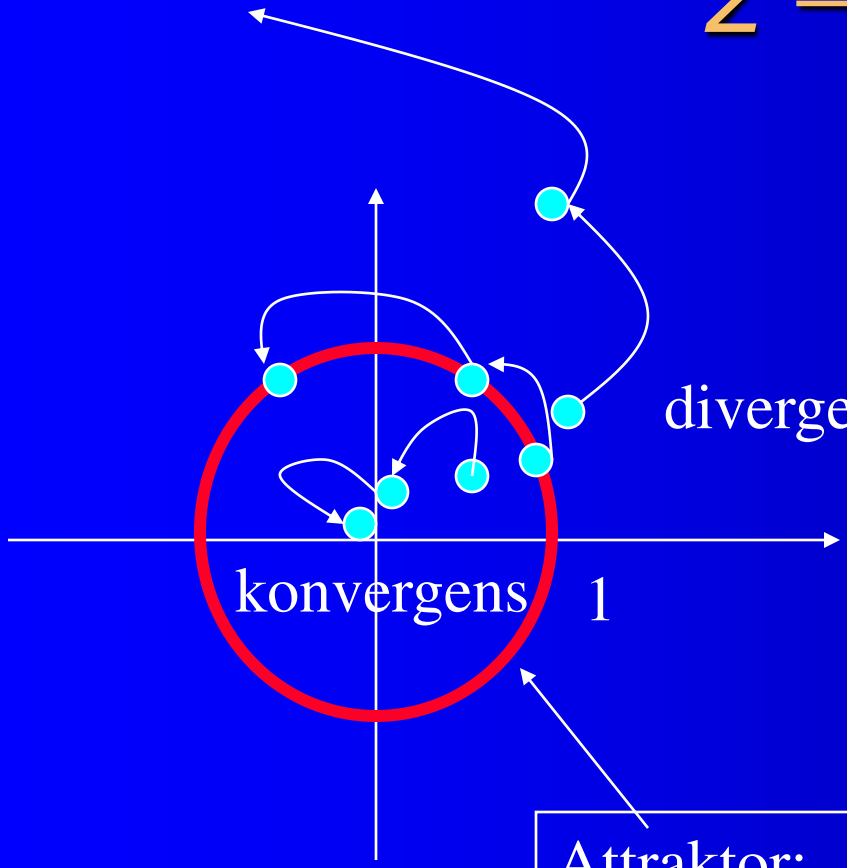
g nagy



Kaotikus rendszerek a síkon



$$z \rightarrow z^2$$



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

$$r \rightarrow r^2$$

$$\phi \rightarrow 2\phi$$

Attraktor: $H = F(H)$

Attraktor előállítása

- Attraktor a labilis és a stabilis tartomány határa: kitöltött attraktor = amely nem divergens
  - $z_{n+1} = z_n^2$: ha $z_\infty < \infty$ akkor fekete
- Attraktorhoz konvergálunk, ha az stabil
  - $z_{n+1} = z_n^2$ attraktora labilis

Inverz iterációs módszer

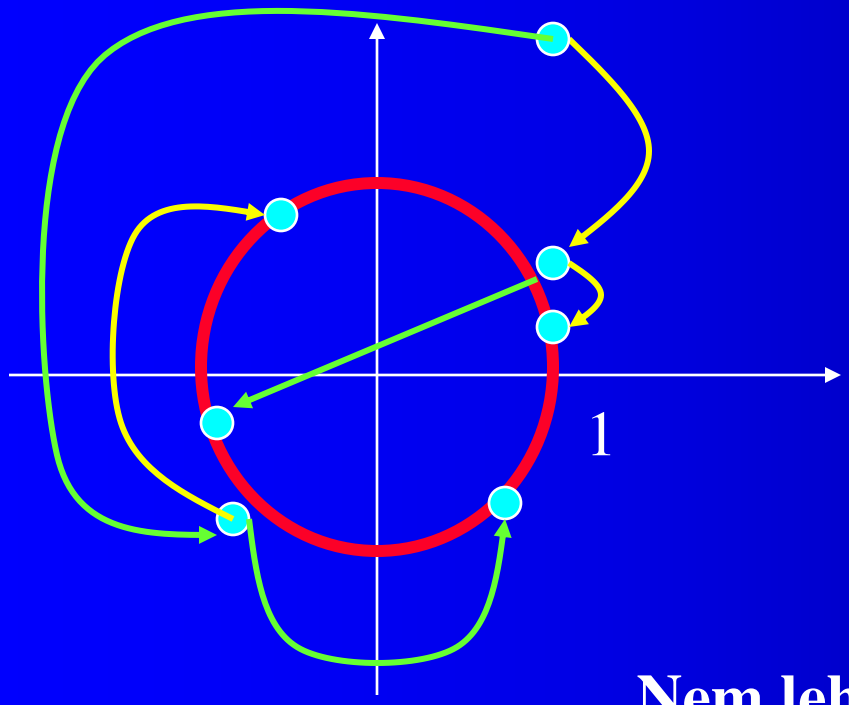
$H = F(H) \quad \rightarrow \quad H = F^{-1}(H)$

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

$z_{n+1} = \pm \sqrt{z_n}$

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

$\phi_{n+1} = \phi_n/2 + \{0,1\} \cdot \pi$



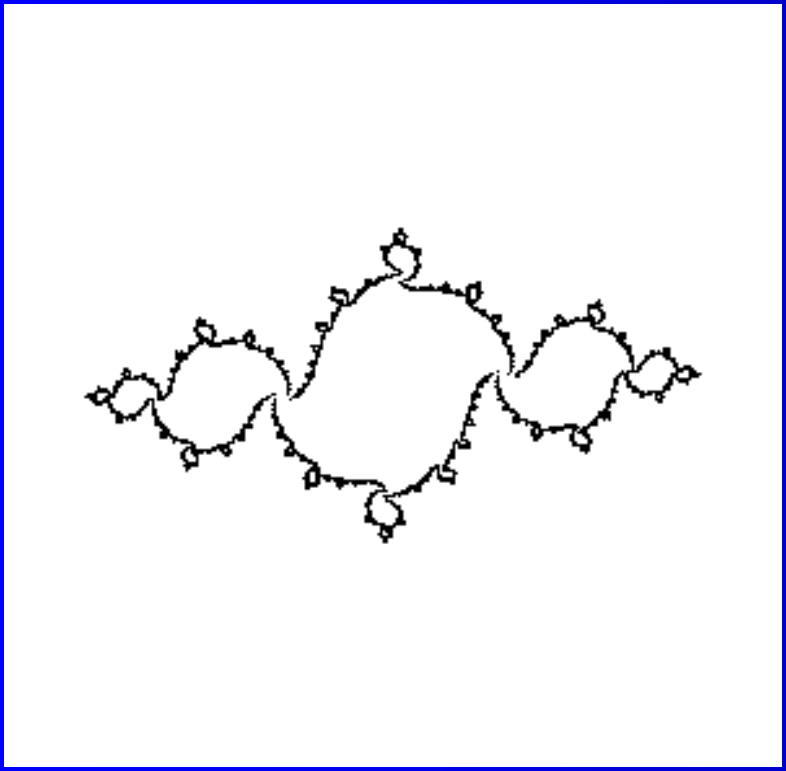
$r_n \approx 1$

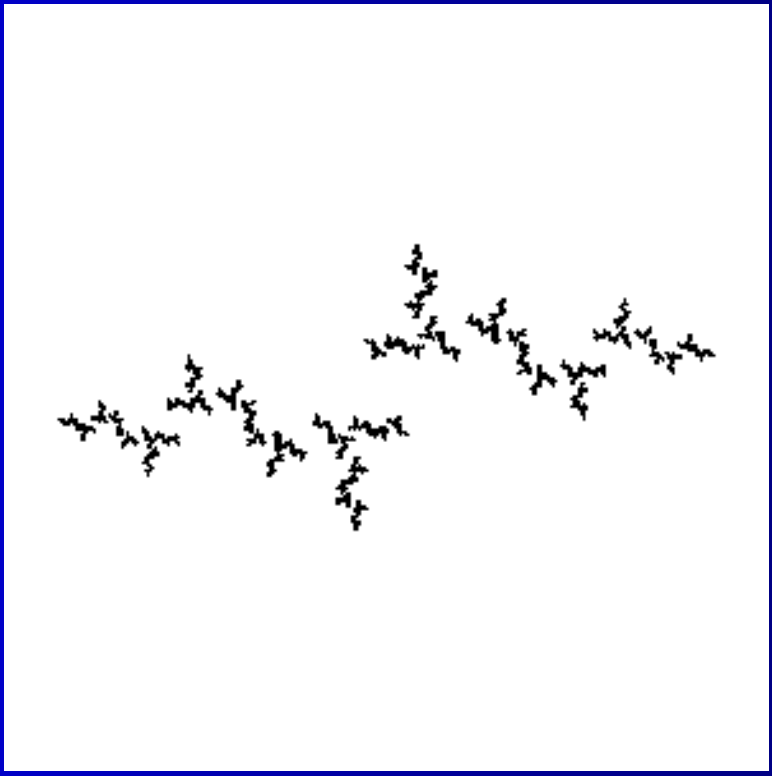
$\phi_n \approx \{0,1\}.\{0,1\}\{0,1\}\ldots \cdot \pi$

$n \quad n\text{-}1 \quad n\text{-}2$

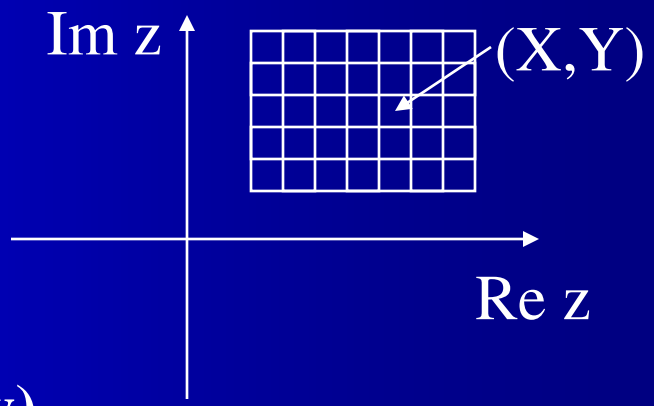
Nem lehet csak egy értékkel dolgozni ???

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





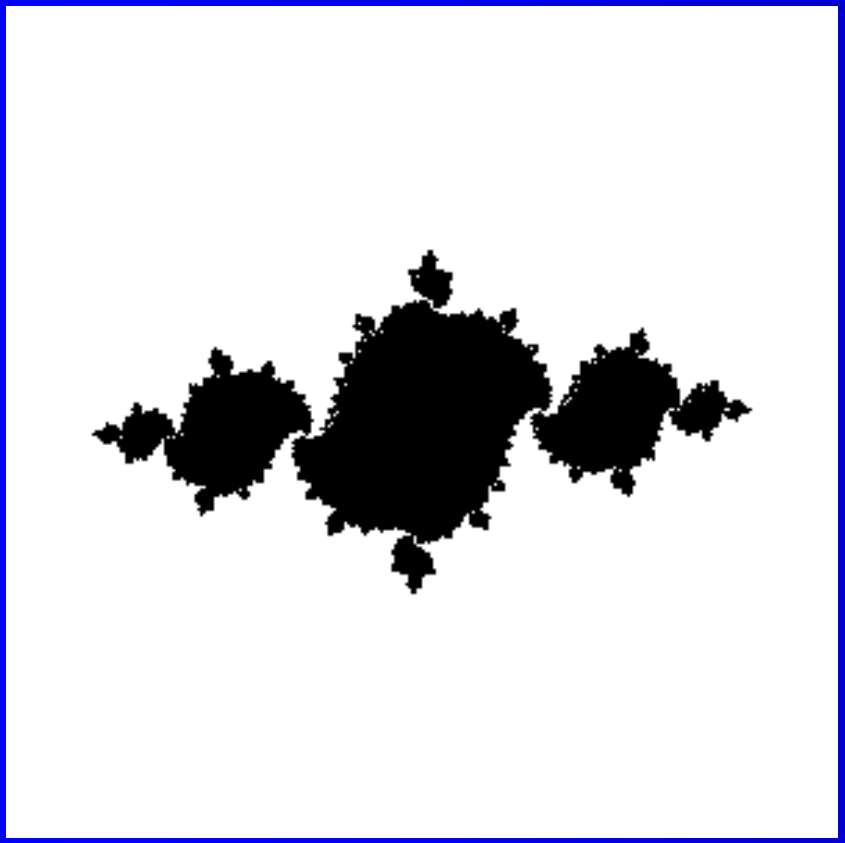
Kitöltött Julia halmaz: algoritmus

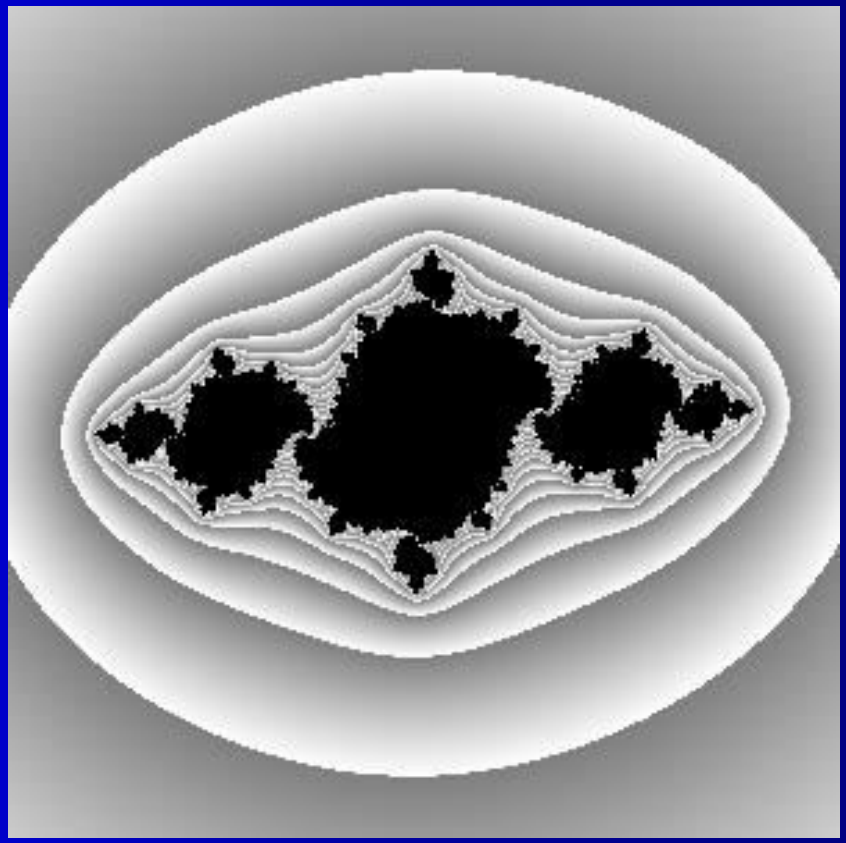


FilledJuliaDraw ( )

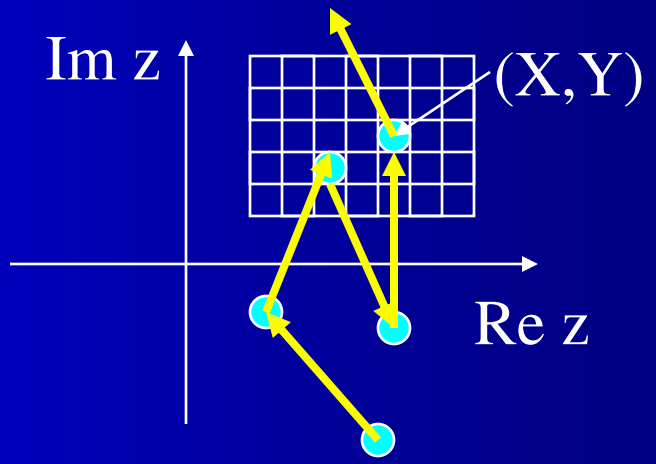
FOR Y = 0 TO Ymax DO
FOR X = 0 TO Xmax DO
ViewportWindow(X,Y → x, y)
z = x + j y
FOR i = 0 TO n DO $z = z^2 + c$
IF |z| > "infinity" THEN WRITE(X,Y, white)
ELSE WRITE(X,Y, black)
ENDFOR
ENDFOR
END

Kitöltött Julia halmaz: kép





Julia halmaz inverz iterációval

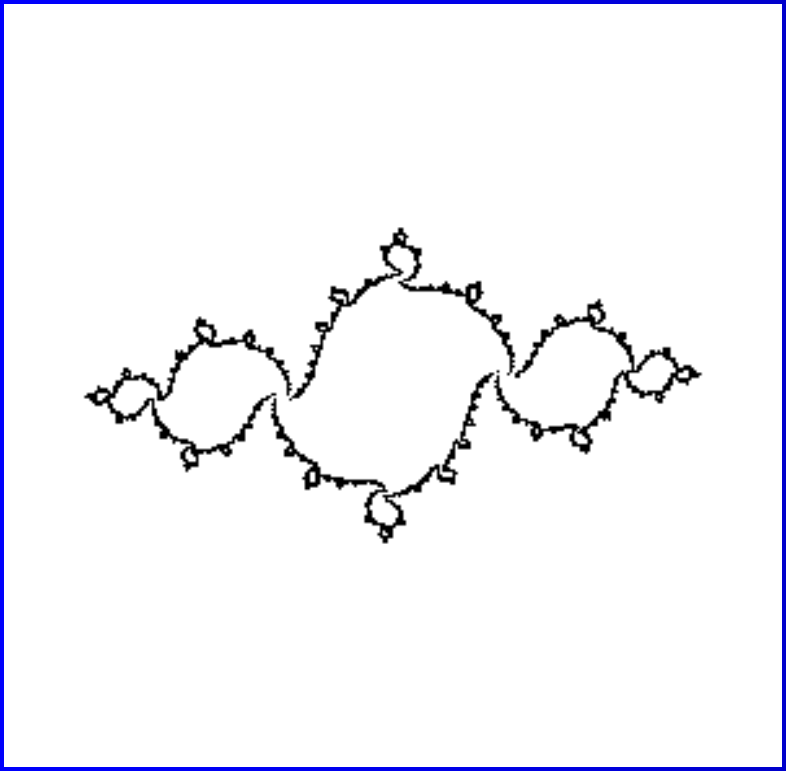


JuliaDrawInverseIterate ( )

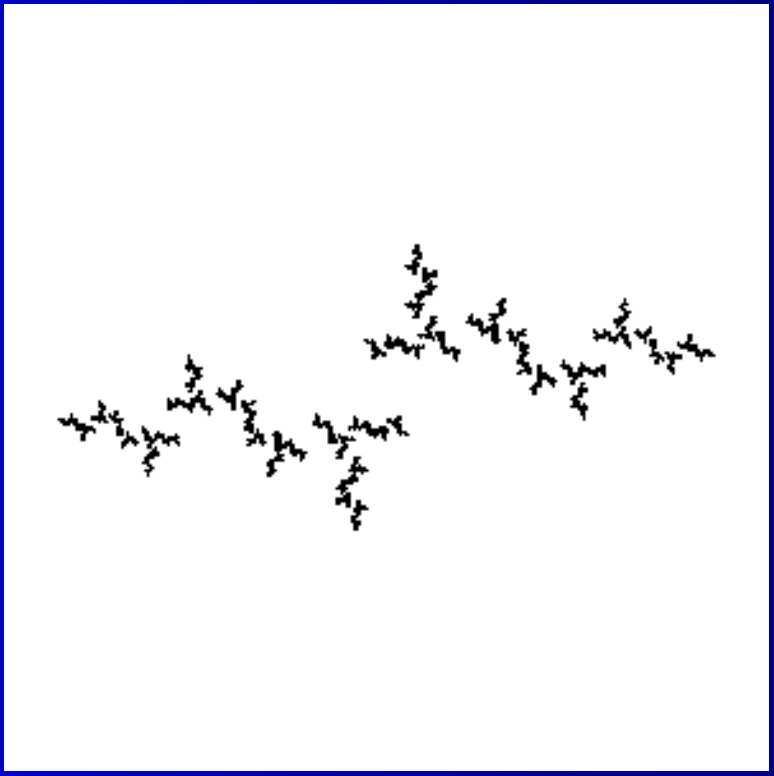
Kezdeti z érték választás
FOR i = 0 TO n DO
x = Re z, y = Im z
IF ClipWindow(x, y)
WindowViewport(x, y ⇨ X, Y)
Pixel(X, Y) = fekete
ENDIF
$z = \sqrt{z - c}$
if (rand( ) > 0.5) z = -z
ENDFOR
END

Kezdeti *z* érték:
$z^2 = z - c$ gyöke

Julia halmaz

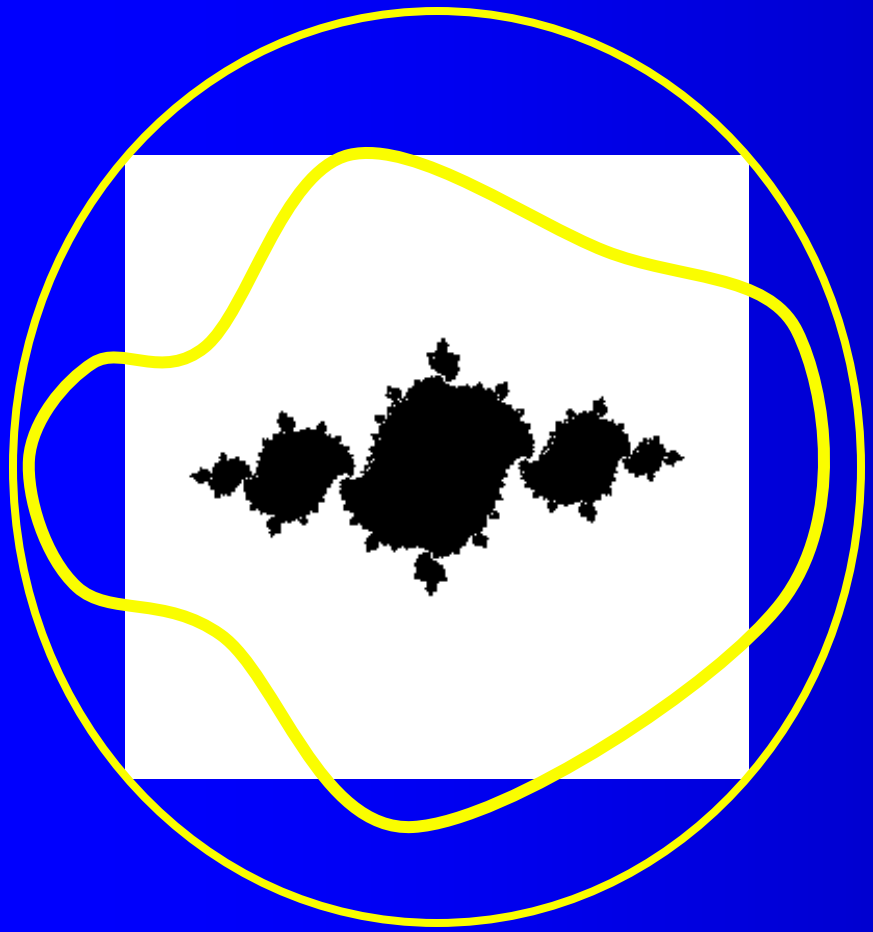


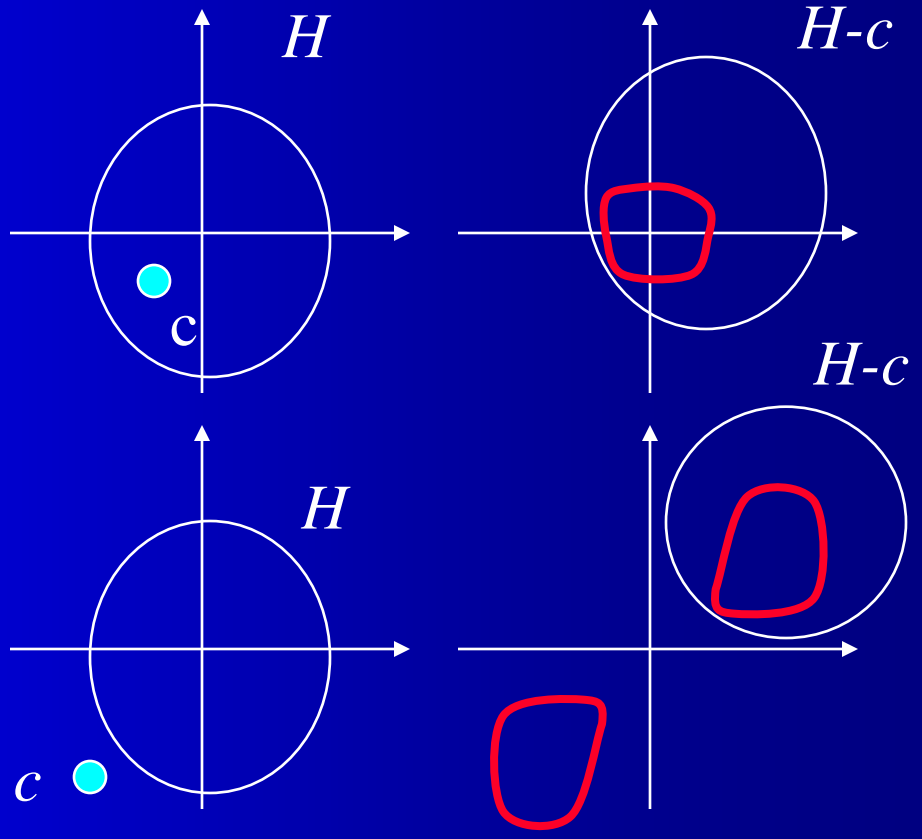
összefüggő



nem összefüggő,
Cantor féle halmaz

Julia halmaz összefüggősége

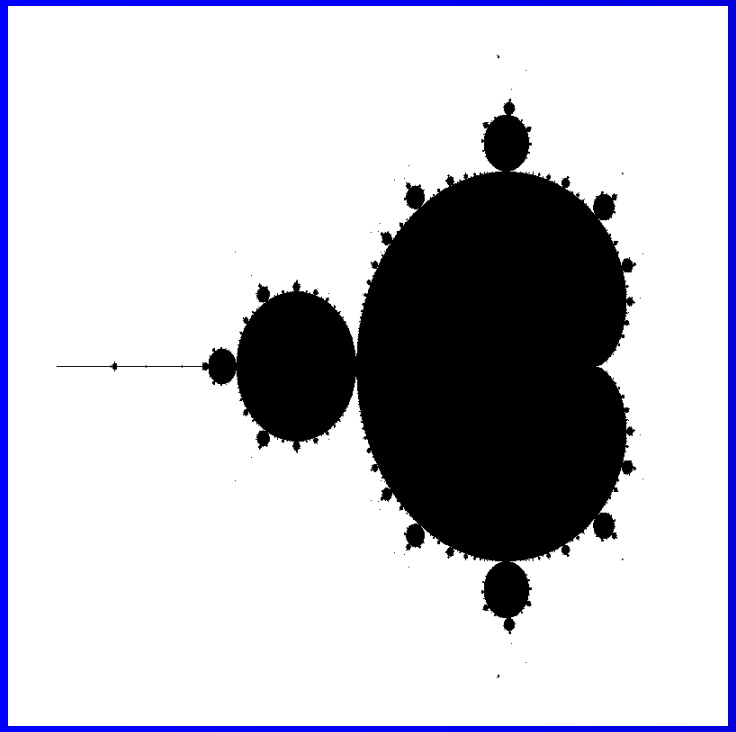


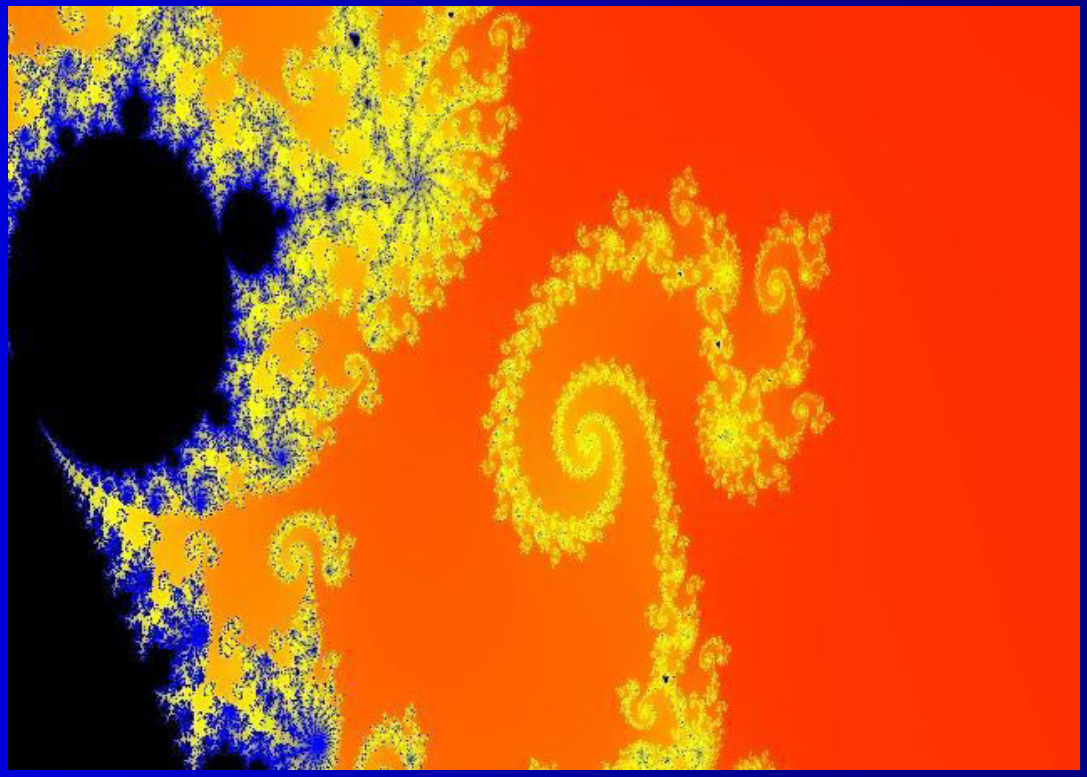


$$z_{n+1} = \pm \sqrt{z_n - c}$$

Mandelbrot halmaz

Azon c komplex számok, amelyekre a $z \to z^2 + c$ Julia halmaza összefüggő



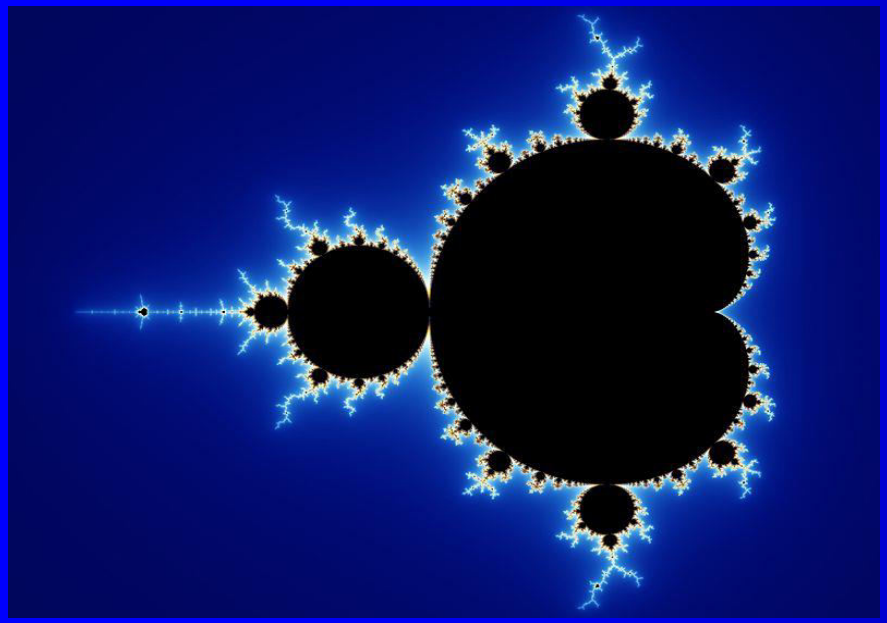


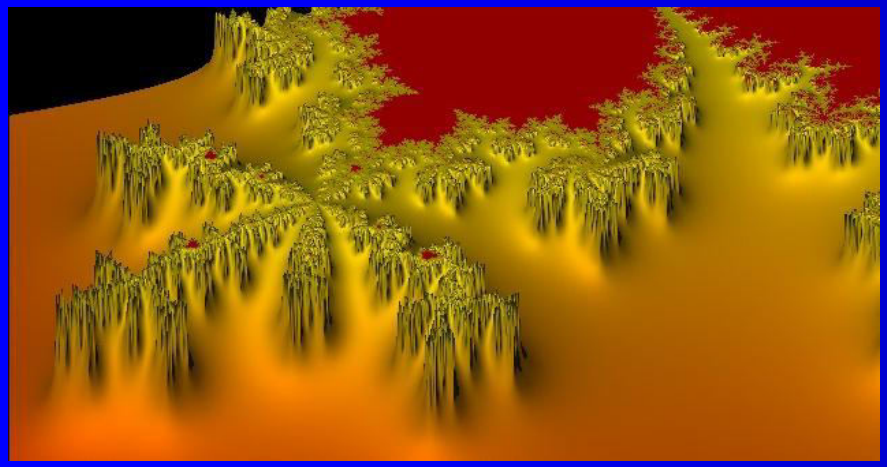
Mandelbrot halmaz, algoritmus

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 = z^2 + c
        IF |z| > "infinity" THEN  WRITE(X,Y, white)
        ELSE                      WRITE(X,Y, black)
    ENDFOR
ENDFOR
END
```

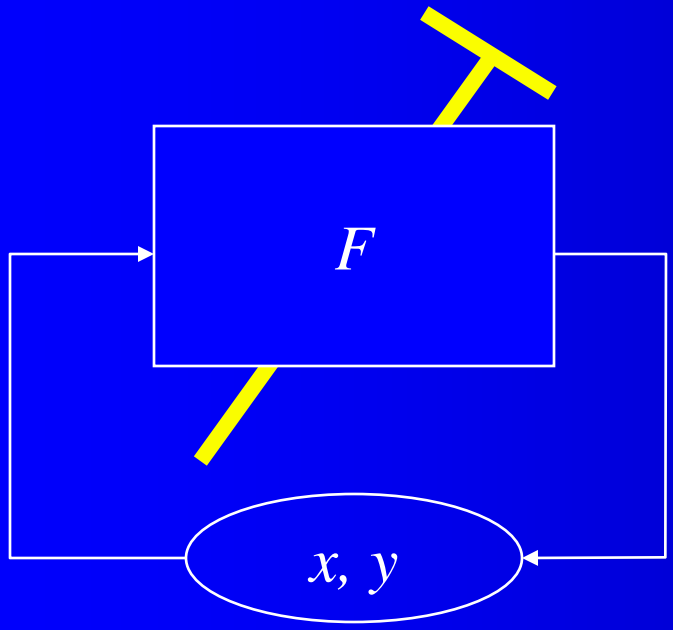
„Matematikát kijátszó” Mandelbrot halmazok

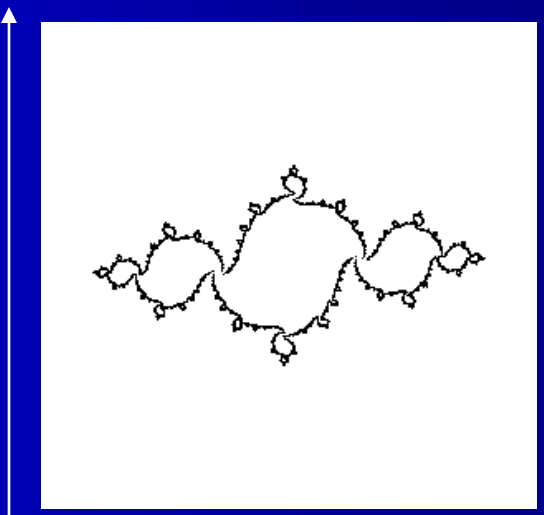






Inverz feladat: IFS modellezés





$H \Rightarrow F$

Attraktor:

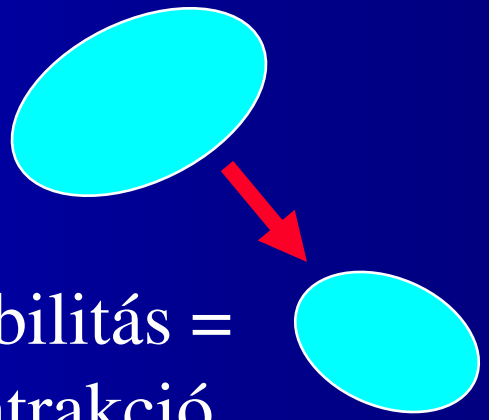
$H = F(H)$

F: szabadon vezérelhető, legyen stabil attraktora

F: többértékű lineáris leképzés

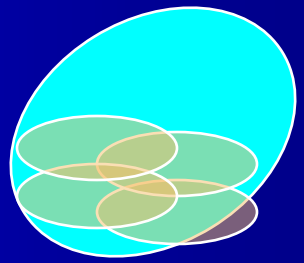
$$F = W_1 \vee W_2 \vee \ldots \vee W_n$$

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



Stabilitás = kontrakció

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

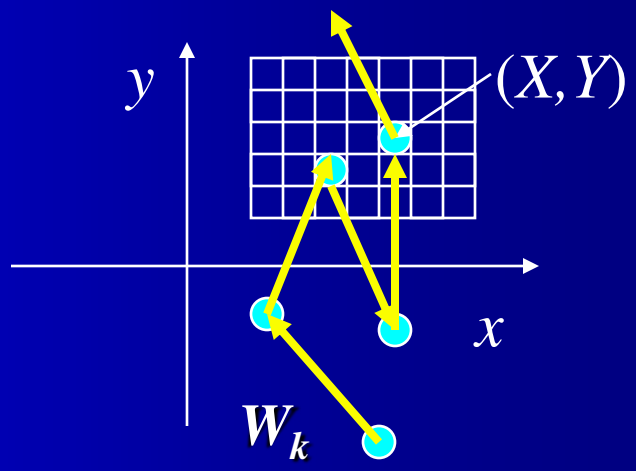


$$H = F(H)$$

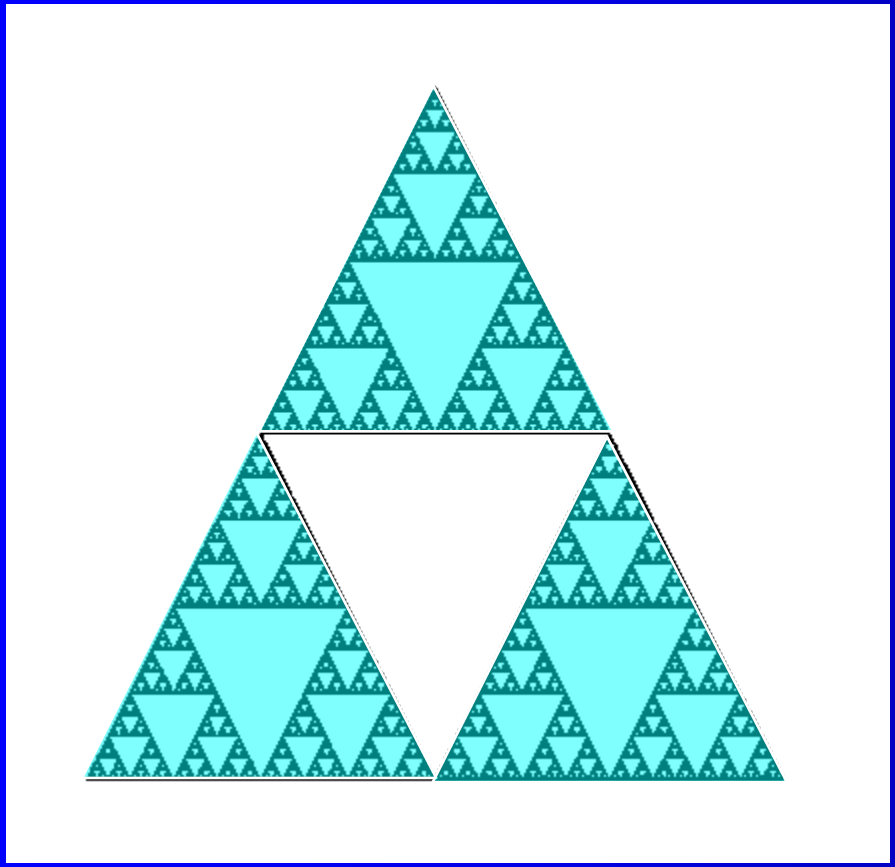
IFS rajzolás: iterációs algoritmus

IFSDraw ( )

Legyen [x,y] = [x,y] A_1 + q_1 megoldása a kezdő [x,y]
FOR i = 0 TO n DO
IF ClipWindow(x, y)
WindowViewport(x, y ⇨ X, Y)
Write(X, Y, color);
ENDIF
Válassz k-t p_k valószínűséggel
[x,y] = [x,y] A_k + q_k
ENDFOR
END

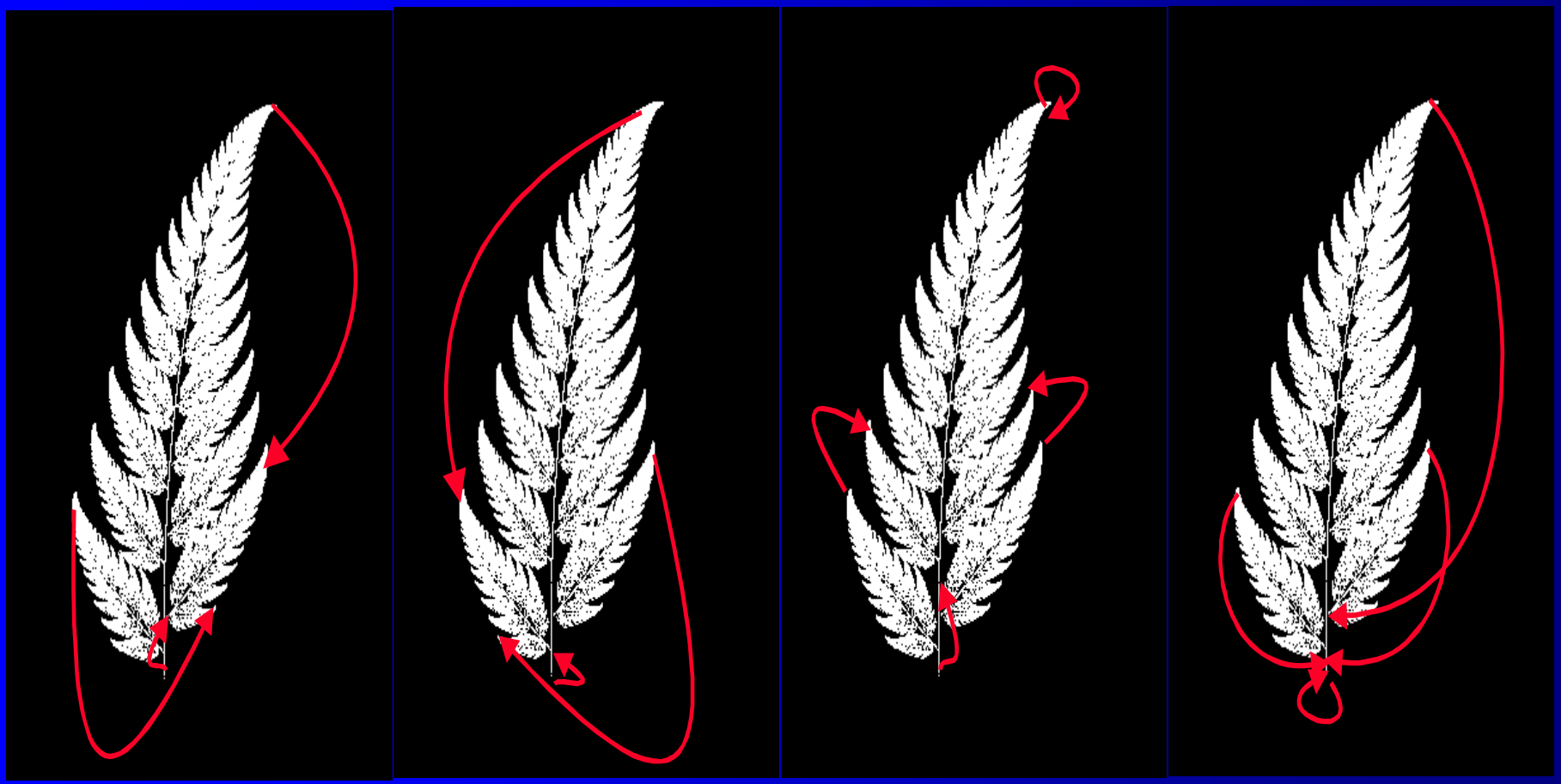


Egyszerű IFS-ek

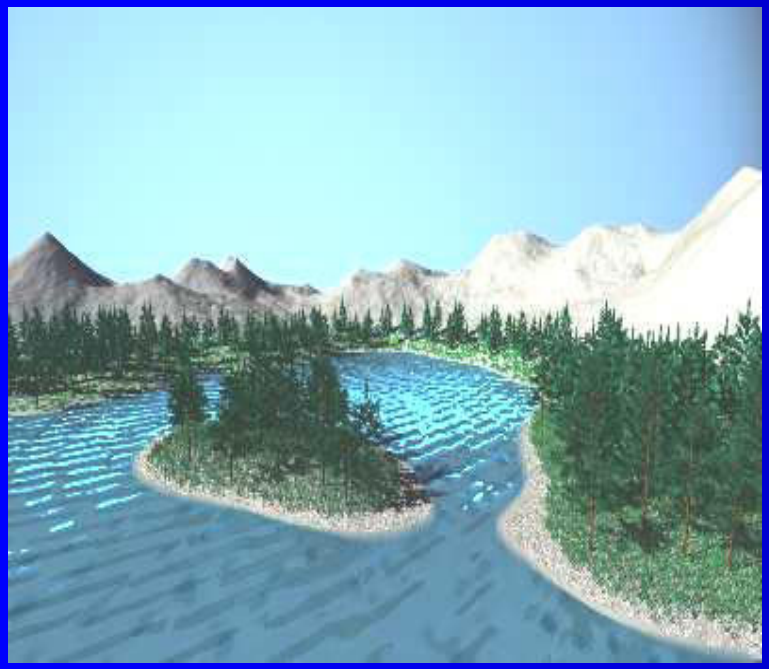




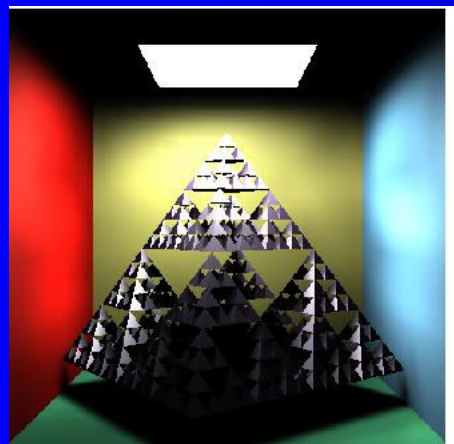
IFS modellezés

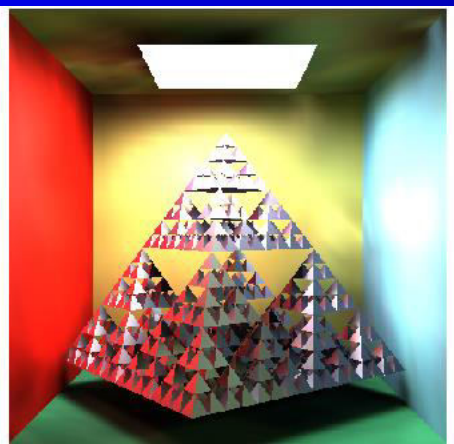


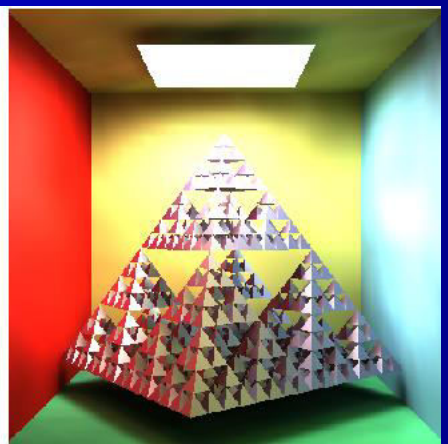
IFS képek







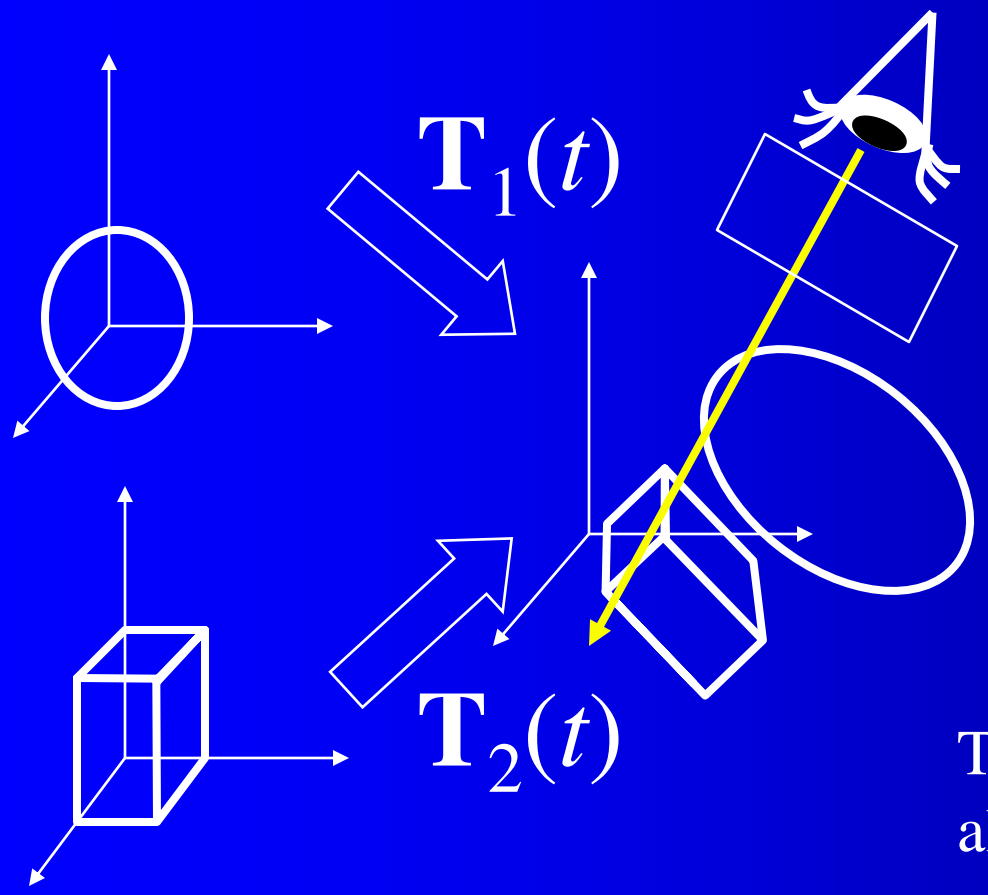




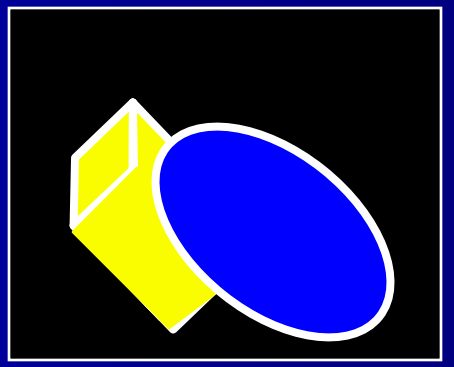
Animáció

Szirmay-Kalos László

Animáció = időfüggés



$\mathbf{T}_V(t)$



Transzformációk
alak
szín
megjelenítési attribútumok, stb

Valós idejű animáció

Óra inicializálás (t_{start})
$t_{prev} = t_{start}$;
do
    t = Óra leolvasás;
    Szimuláció(t_{prev}, t);
    for each object o: modellezési transzf $T_{M,o} = T_{M,o}(t)$;
    Nézeti transzformáció: $T_V = T_V(t)$;
    Képszintézis;
    $t_{prev} = t$;
while ($t < t_{end}$);

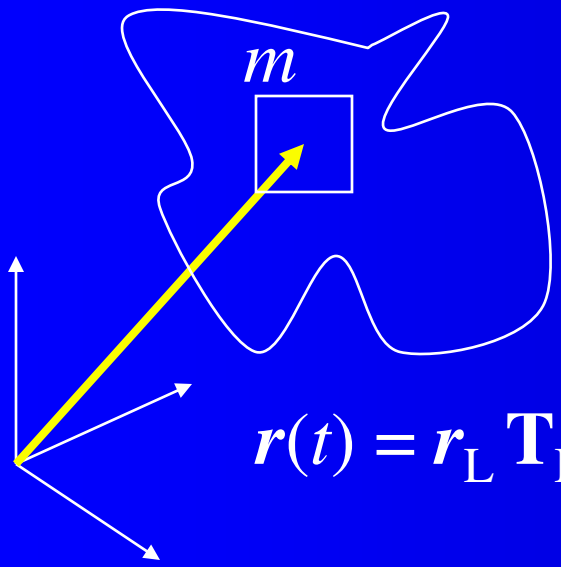
Legalább 15 ciklus másodpercenként

Valószerű mozgás

- Fizikai törvények:
  - Newton törvény
  - ütközés detektálás és válasz: impulzus megmaradás
- Fiziológiai törvények
  - csontváz nem szakad szét
  - meghatározott szabadságfokú ízületek
  - bőr rugalmasan követi a csontokat
- Energiafelhasználás minimuma



Newton törvény



$$\boldsymbol{r}(t) = \boldsymbol{r}_{\mathrm{L}}\,\mathbf{T}_{\mathrm{M}}(t)$$

$$\mathbf{F}/m = \frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{r}$$

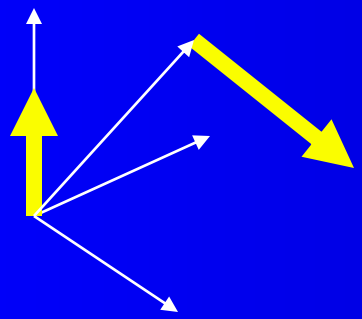
$$= \boldsymbol{r}_{\mathrm{L}}\frac{\mathrm{d}^2}{\mathrm{d}t^2}\mathbf{T}_{\mathrm{M}}(t)$$

Az erő rugalmas mechanizmuson keresztül hat, azaz folytonosan változik

$\mathbf{T}_{\mathrm{M}}(t)$ C^2 folytonos

$T_M(t)$: Mozgástervezés

- Követelmény: ált. C^2 , néha (C^1,C^0) folytonosság
- Mátrixelemek nem függetlenek
  - Tervezés független paraméterek terében



pozíció: px, py, pz
orientáció: α, β, γ
skálázás: sx, sy, sz

Tengely+szög

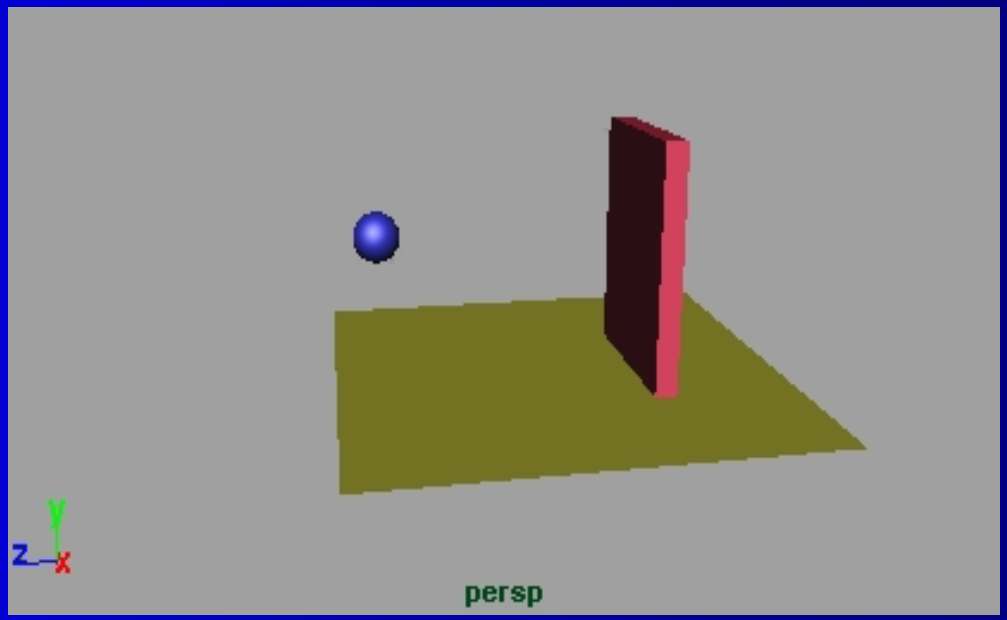
$\mathbf{p}(t)$

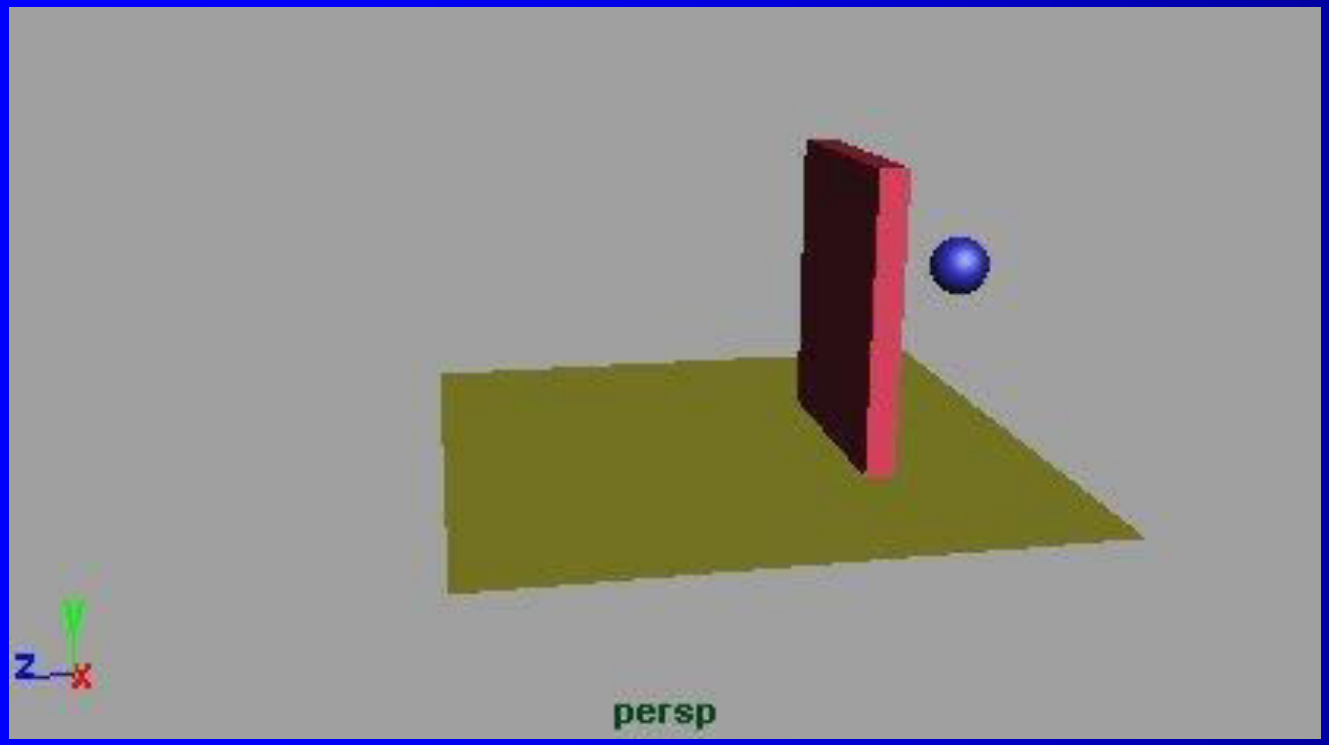
$$\mathbf{T}_M(t) = \begin{bmatrix} sx & & & \\ & sy & & \\ & & sz & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & & \\ -\sin\alpha & \cos\alpha & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & & -\sin\beta & \\ & 1 & & \\ \sin\beta & & \cos\beta & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & \cos\gamma & \sin\gamma & \\ & -\sin\gamma & \cos\gamma & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ px, & py, & pz, & 1 \end{bmatrix}$$

Mozgástervezés a paramétertérben

- **p**(t) elemei ált. C^2 , néha (C^1,C^0) folytonosak
- **p**(t) elemeinek a definíciója:
  - görbével direkt módon (**spline**)
  - képlettel: **script animation**
  - kulcsokból interpolációval**: keyframe animation**
  - görbével indirekt módon: **path animation**
  - mechanikai modellből az erők alapján: **physical animation**
  - mérésekből: **motion capture animation**

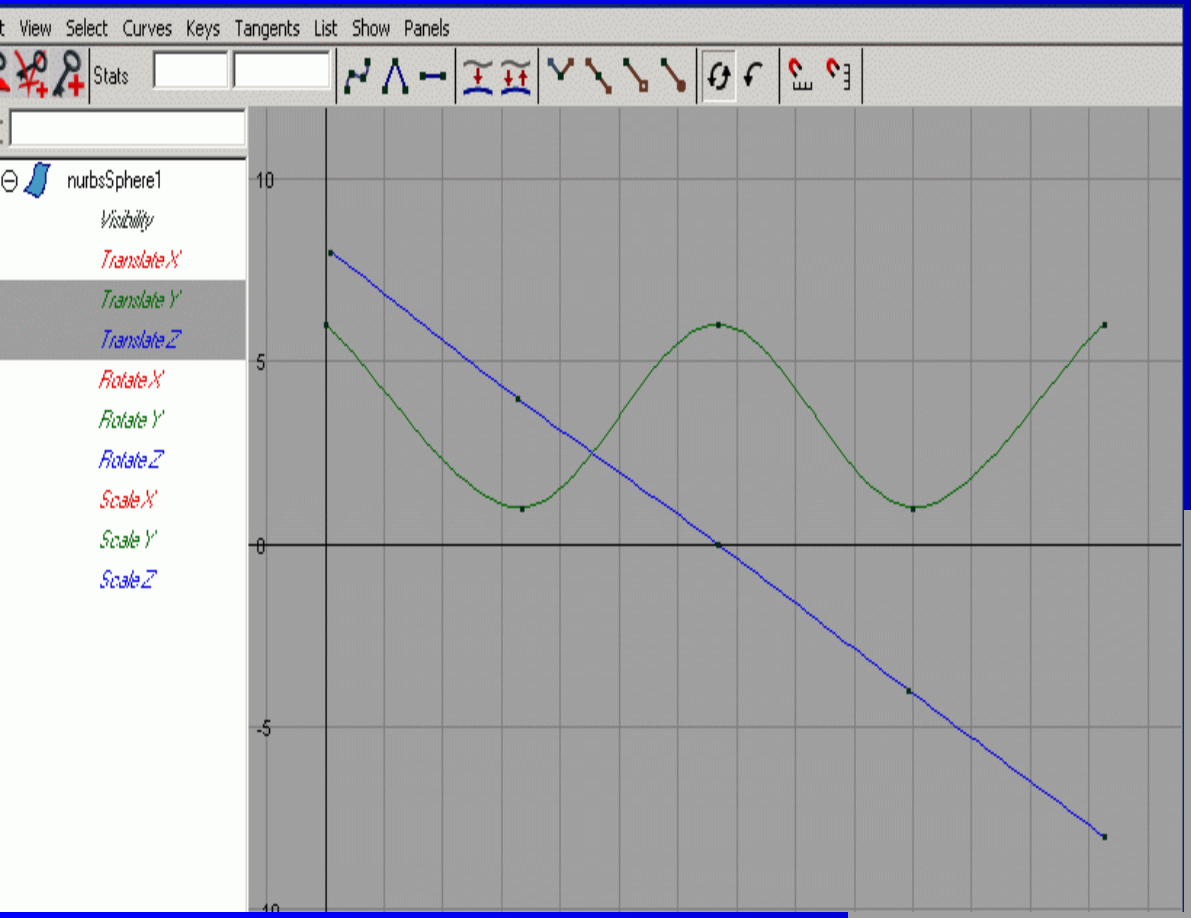
Keyframe animáció

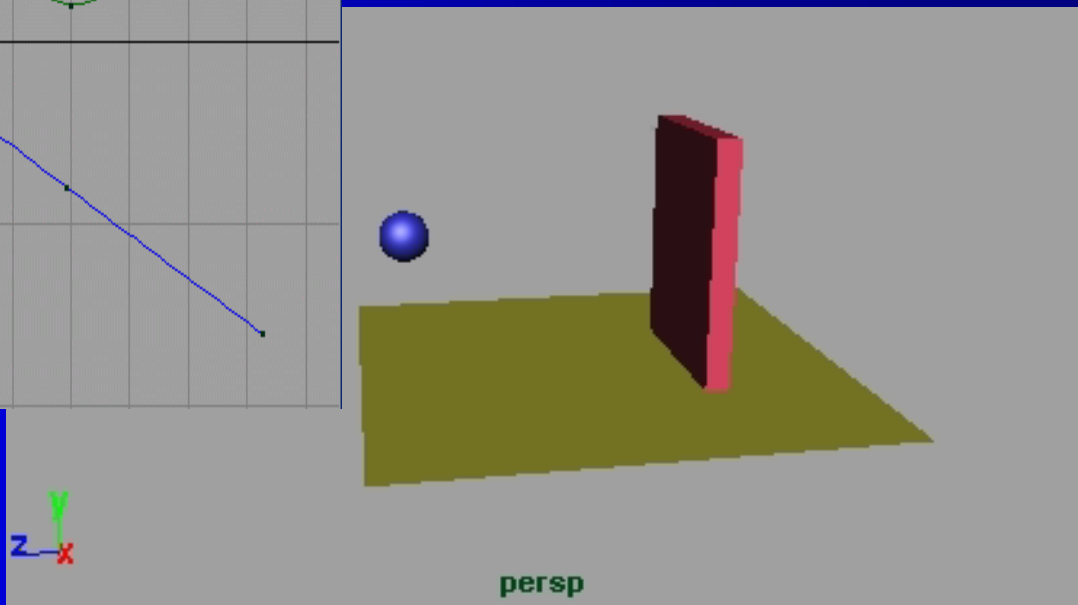




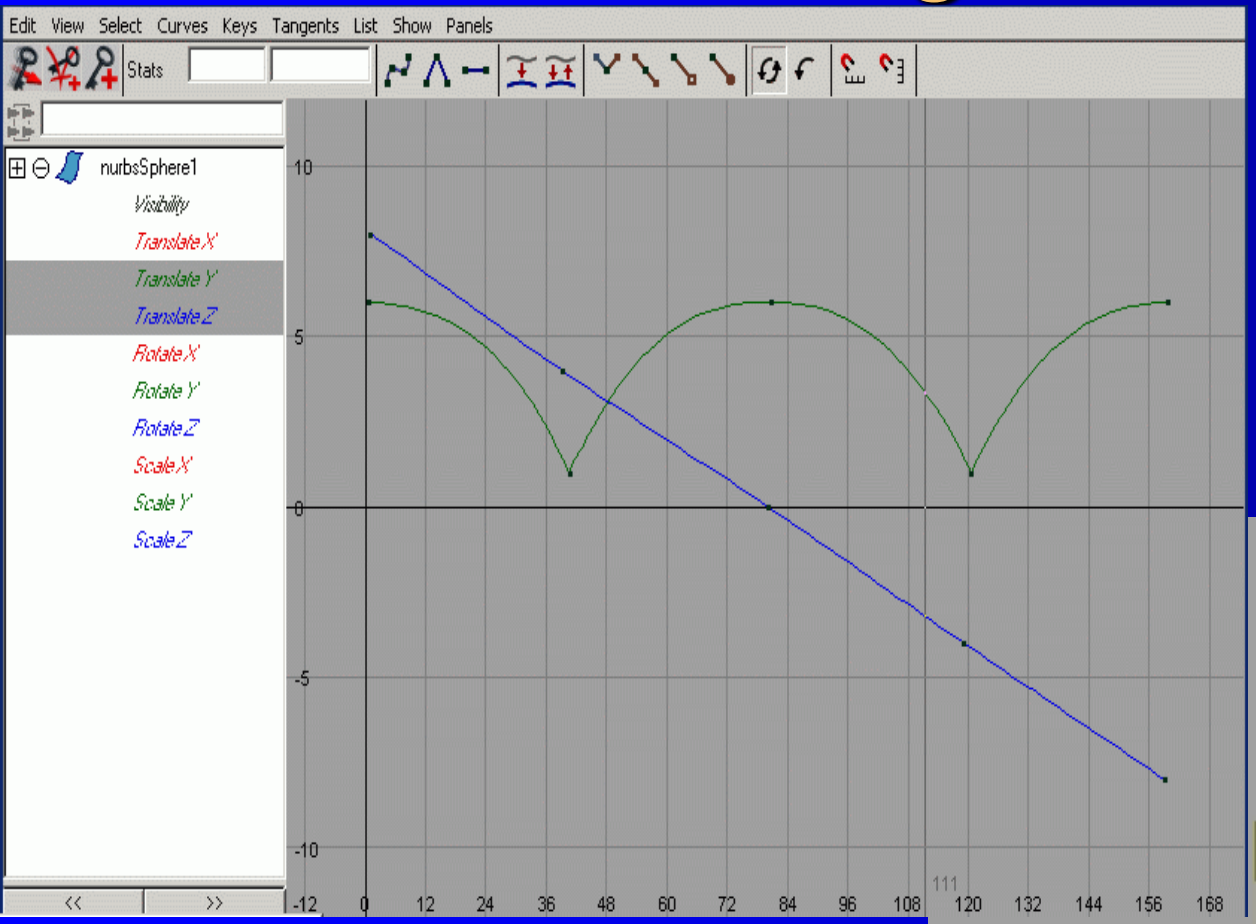
5.key

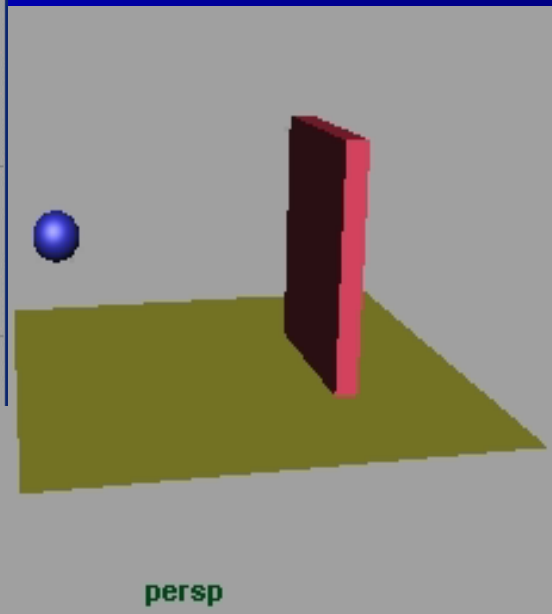
Keyframe animáció görbéi



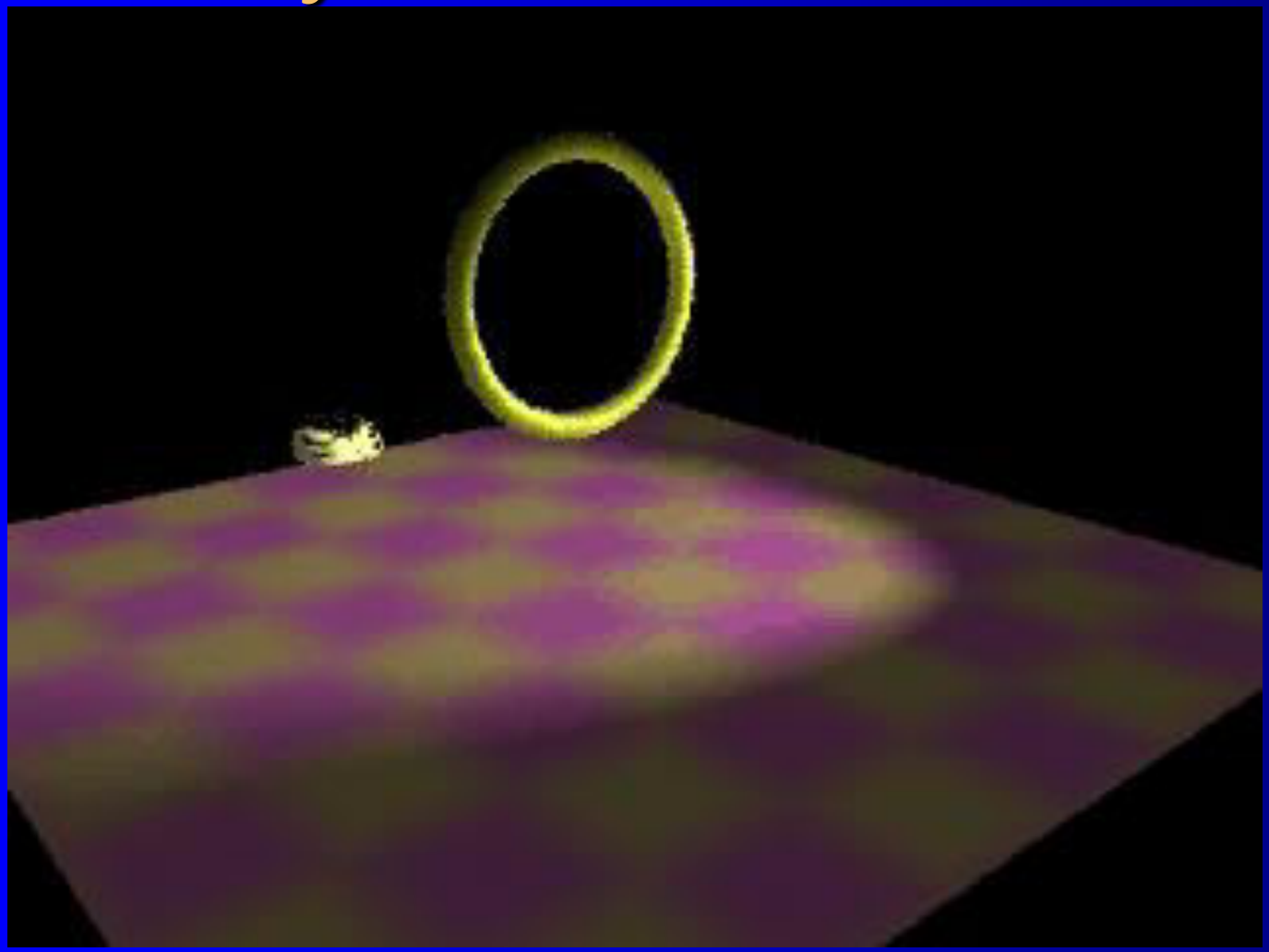


Görbék megváltoztatása

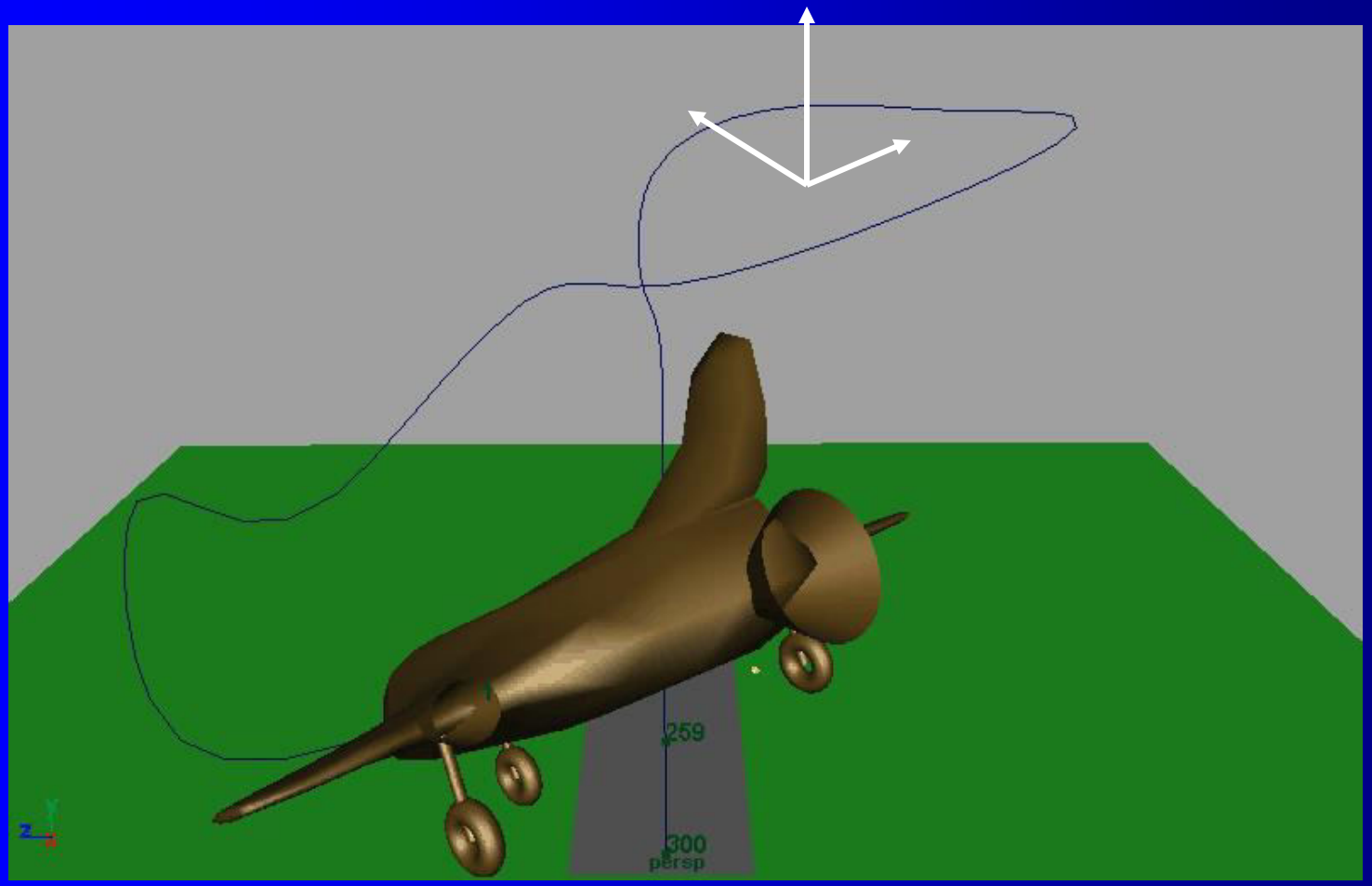




Keyframe animáció

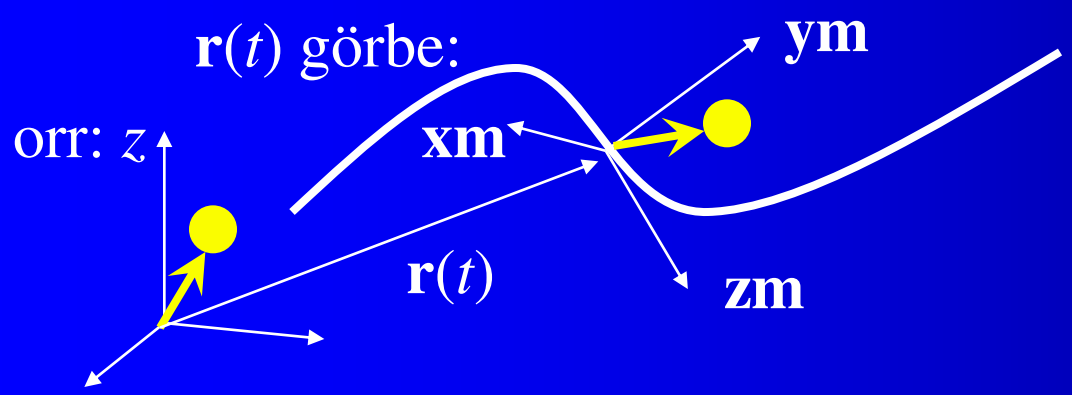


Pálya (path) animáció



t = spline paraméter vagy az ívhossz

Pálya animáció: Transzformáció



$$T_M = \begin{pmatrix} \mathbf{xm}^0(t) & 0 \\ \mathbf{ym}^0(t) & 0 \\ \mathbf{zm}^0(t) & 0 \\ \mathbf{r}(t) & 1 \end{pmatrix}$$

Explicit up vektor

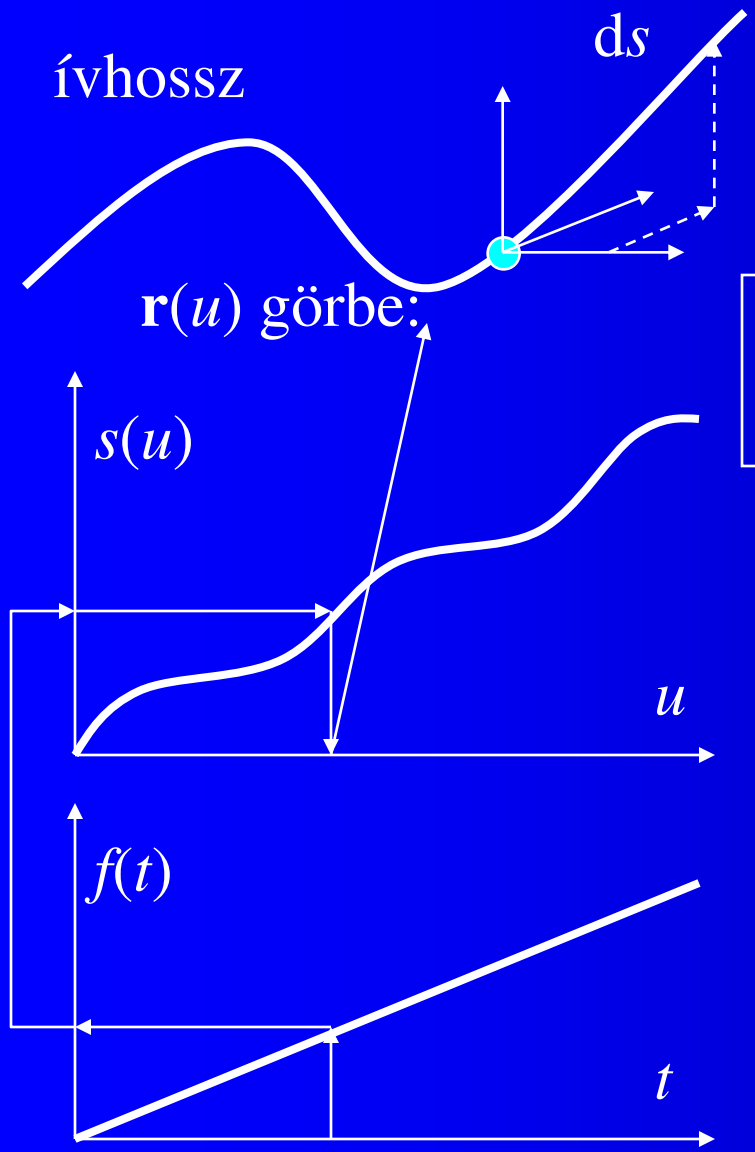
zm = r'(*t*)
xm = zm ⊗ up
ym = zm ⊗ xm

Frenet keretek:

zm = r'(*t*)
xm = zm ⊗ r''(*t*)
ym = zm ⊗ xm

A függőleges, amerre az erő hat

Ívhossz szerinti mozgás



$$dz = dz/du \cdot du$$

$$s(u) = \int^{u} \sqrt{(dx/du)^2 + (dy/du)^2 + (dz/du)^2}\, du$$

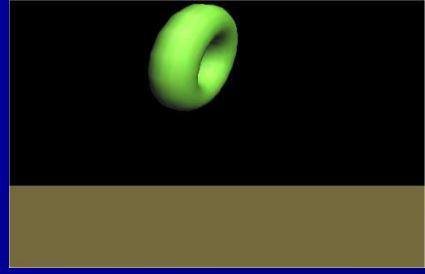
Ívhossz szerinti mozgás

Megtett út-idő függvény:

$s(u) = f(t)$, például $v \cdot t$

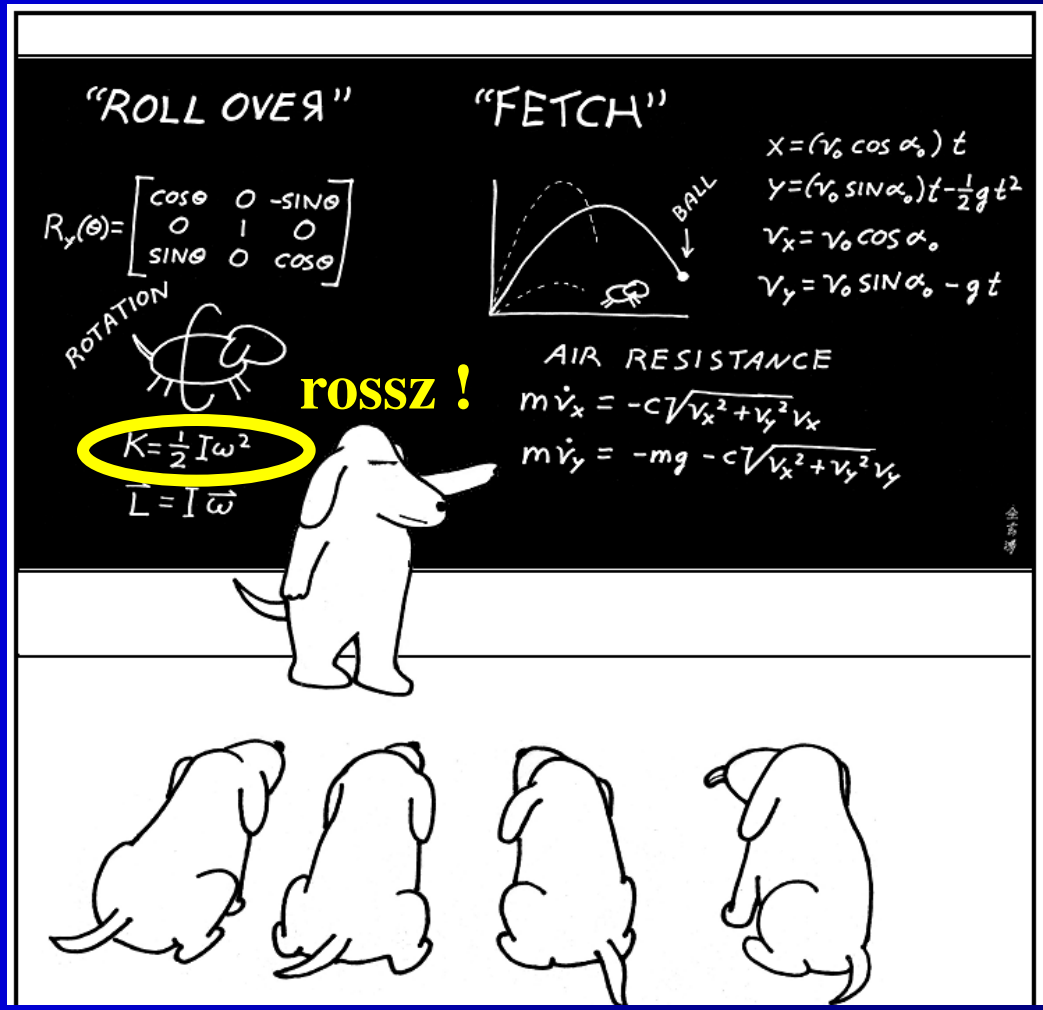
$$u = s^{-1}(f(t))$$

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

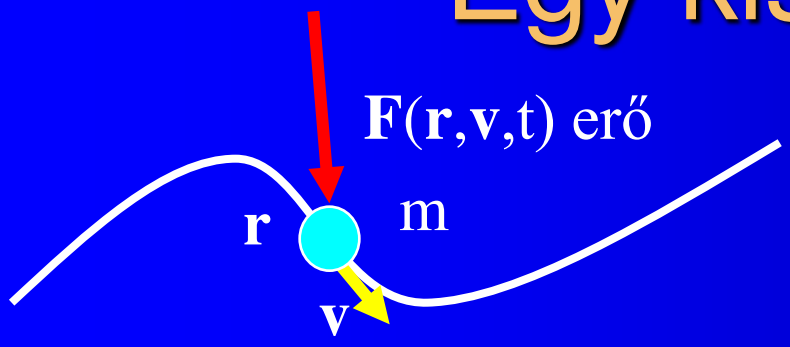


Fizikai animáció

- Erők (pl. gravitáció, turbulencia stb.)
- Tömeg, tehetetlenségi nyomaték (F = ma)
- Ütközés detektálás (metszéspontszámítás)
- Ütközés válasz
  - rugók, ha közel vannak
  - impulzus megmaradás



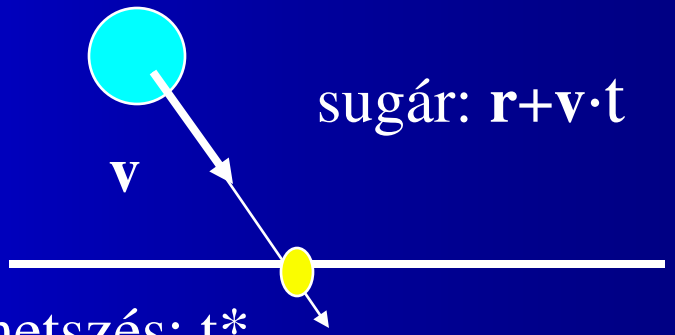
Egy kis mechanika



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

```
for( t = 0; t < T; t += dt) {
    F = Eredő erő(r, v)
    a = F/m
    v += a ·dt
    if ( ÜtközésDetektál )
        ÜtközésVálasz
    r += v ·dt
}
```

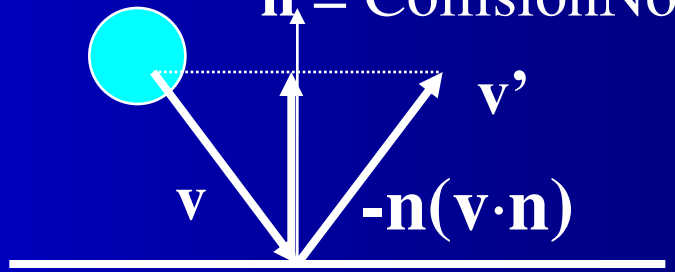
ÜtközésDetektál



metszés: t*

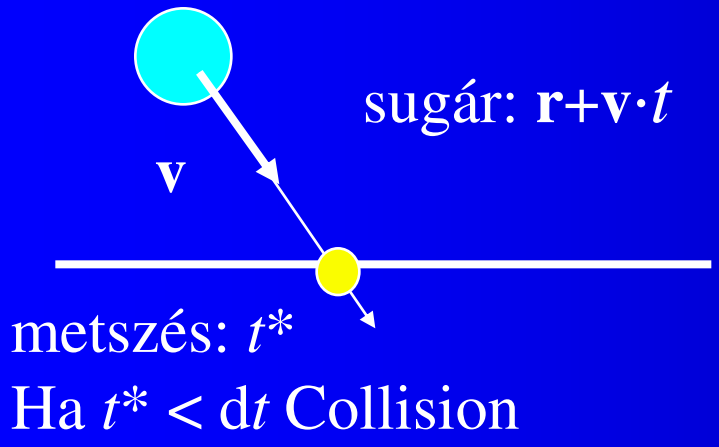
Ha t* < dt Collision

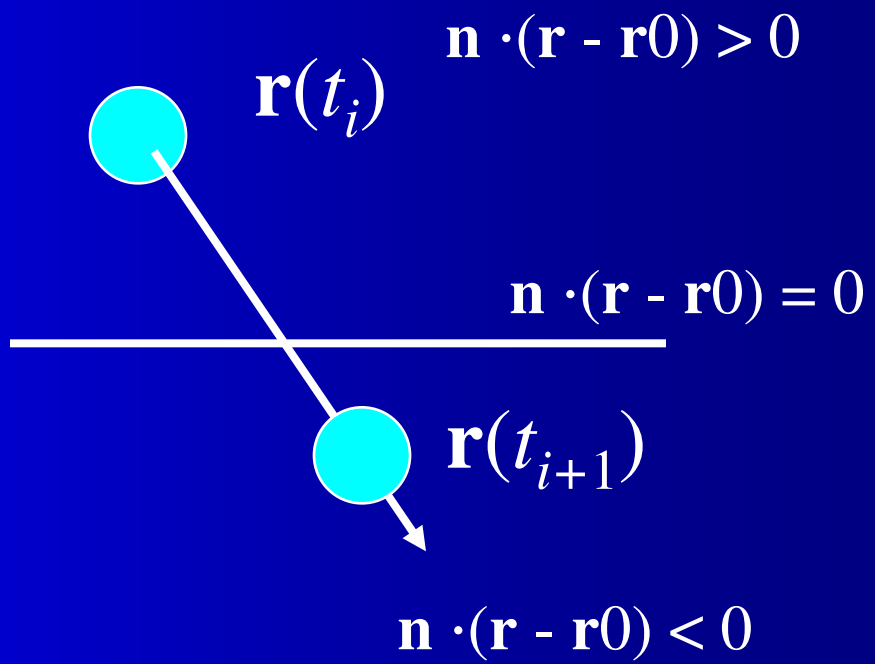
ÜtközésVálasz



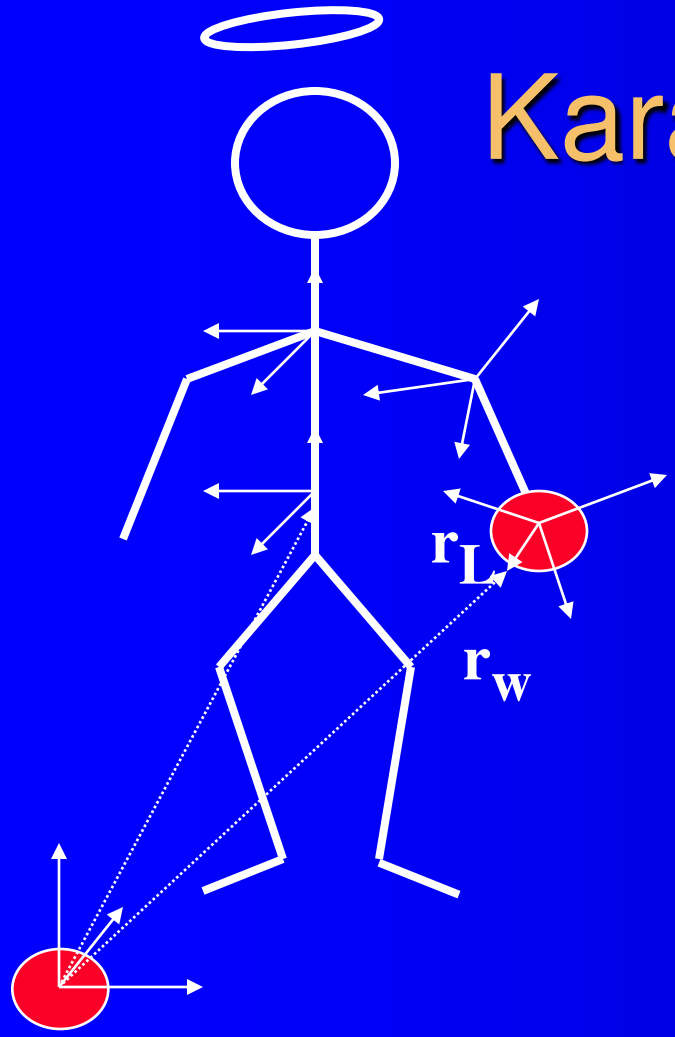
$$\mathbf{v}' = [\mathbf{v} - \mathbf{n}(\mathbf{v}\cdot\mathbf{n})] - [\mathbf{n}(\mathbf{v}\cdot\mathbf{n})\cdot bounce]$$

Folytonos-Diszkrét ütközés detektálás pontra és féltérre





Karakter animáció

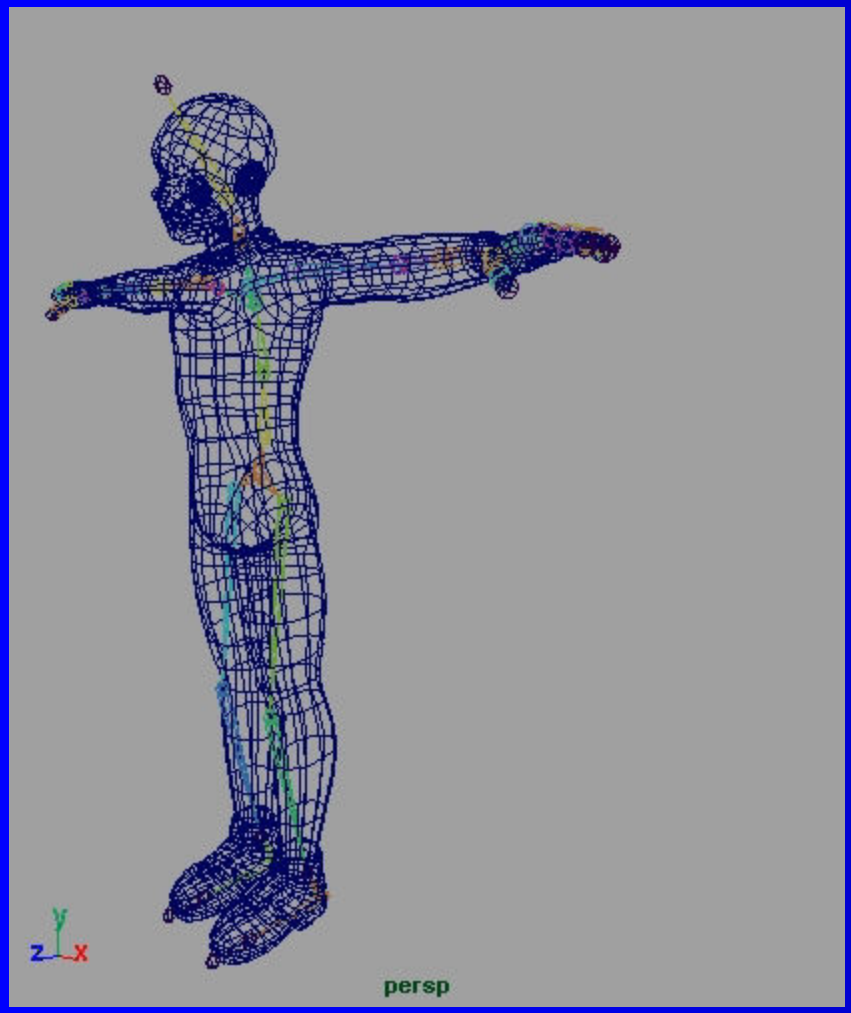


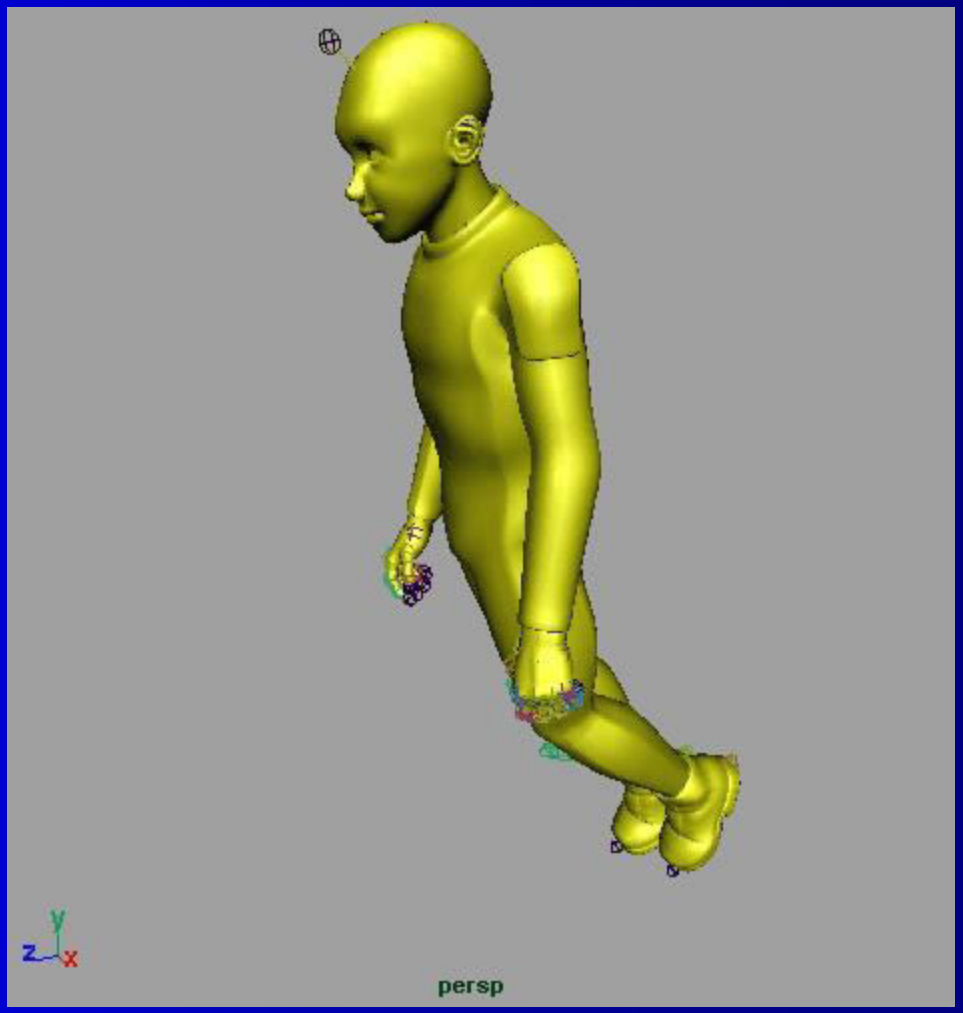


$$r_w = r_L \cdot R_{kéz} \cdot T_{alkar} \cdot R_{könyök} \cdot T_{felkar} \cdot R_{váll} \cdot T_{gerinc} \cdot T_{ember}$$

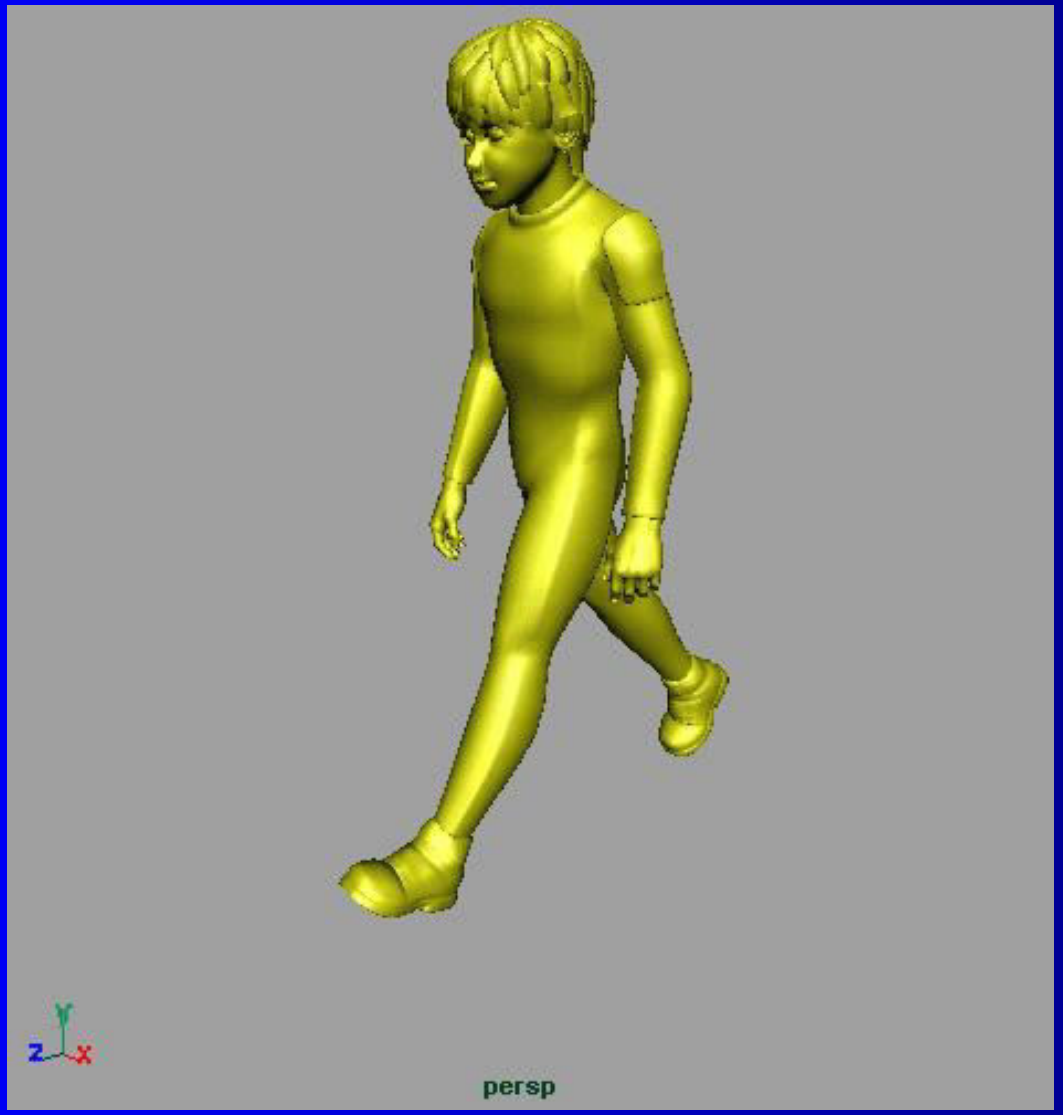
homogén koordináta 4-es

Csontváz

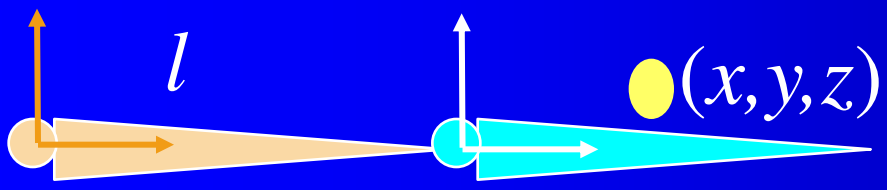




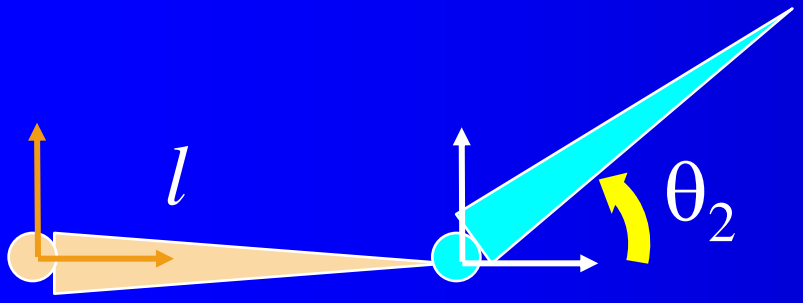
Séta



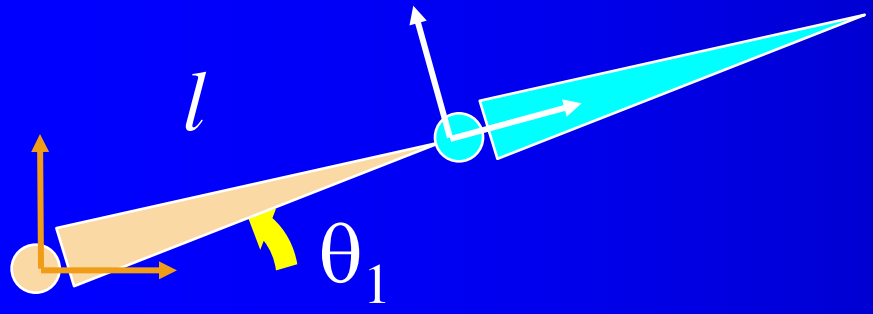
Transzformáció hierarchia



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

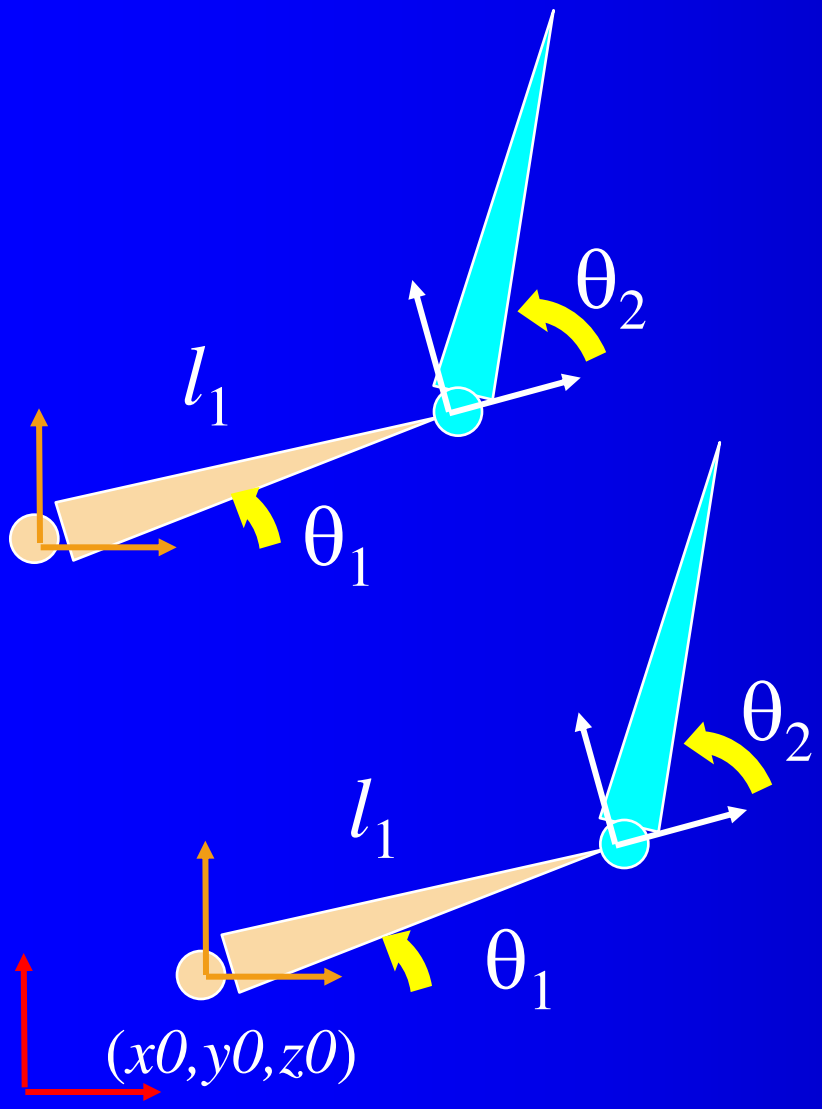


glTranslatef(*l*, 0, 0);
glRotatef(theta2, 0, 0, 1)
glVertex3f(*x*, *y*, *z*);



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

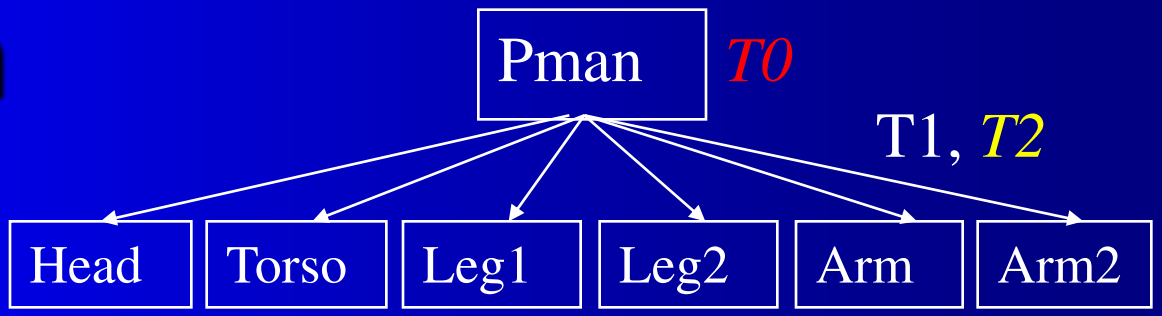
Transzformáció hierarchia




```
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



T0 = Pman előremozog

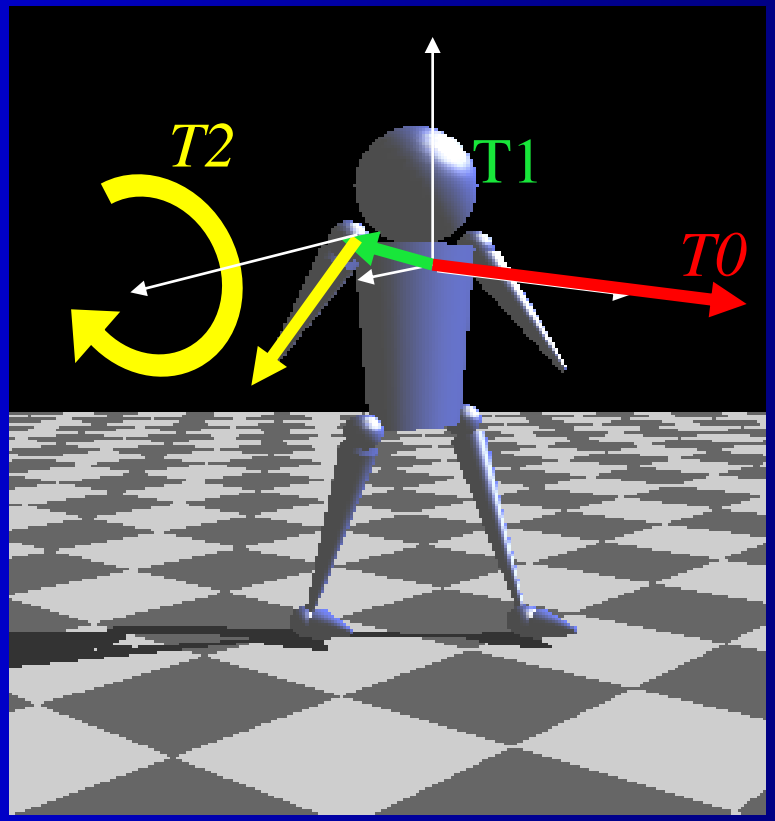
glTranslatef(xpm, ypm, zpm);

T1= váll pozíció

glTranslatef(xsh, ysh, zsh);

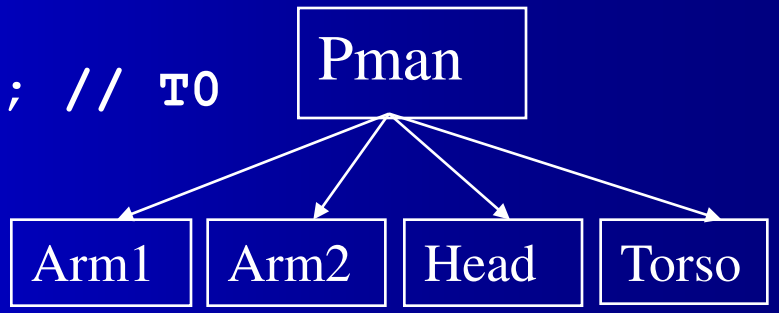
T2= váll forgatás

glRotatef(angle, xshax, yshax, zshax);

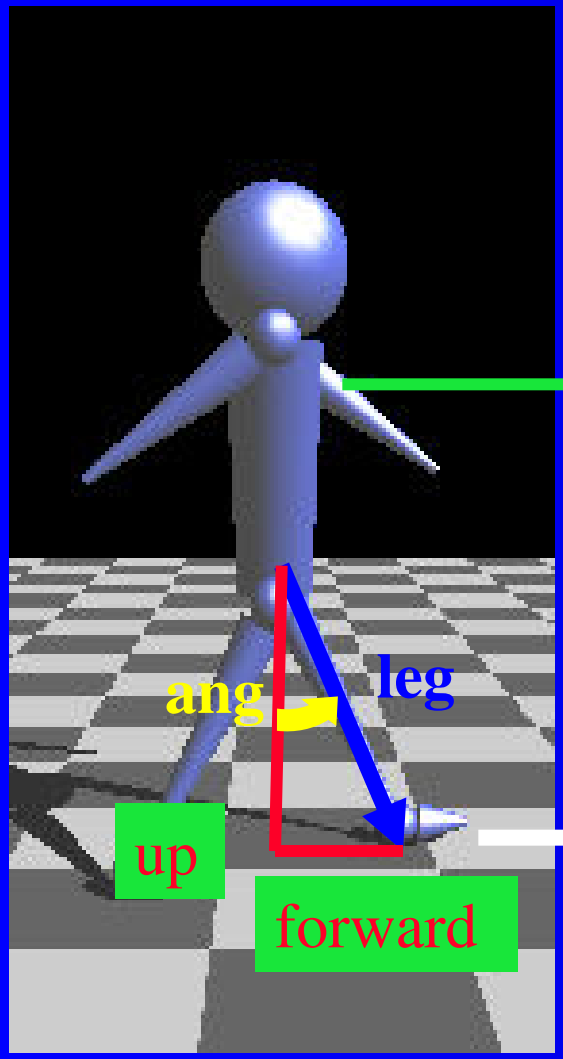


Pman rajzolás és animáció

```
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



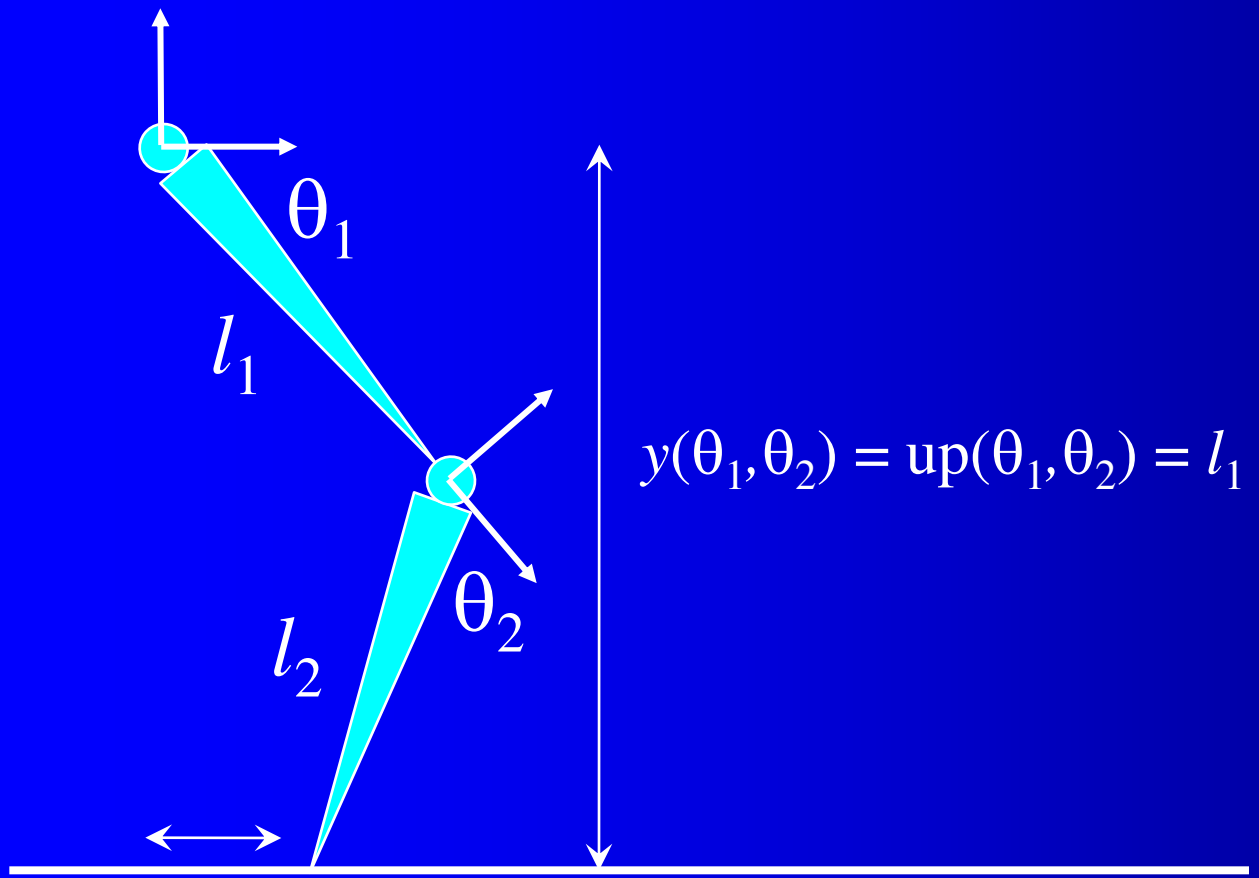
T0 = előremozgás (forward, up) ???

T2 = láb forgatás(ang)

Támaszkodó nem csúszkálhat

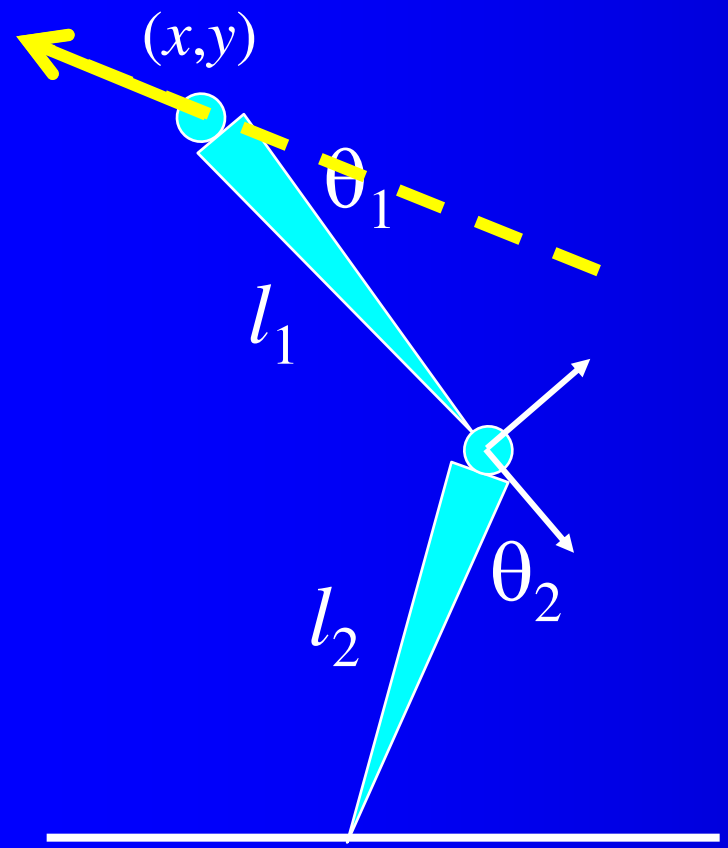
```
forward += leg * fabs(sin(angNew) - sin(angOld));
up      = leg * cos(angNew);
```

Inverz kinematika



$$y(\theta_1,\theta_2) = \mathrm{up}(\theta_1,\theta_2) = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

$$x(\theta_1,\theta_2) = \mathrm{forward}(\theta_1,\theta_2) = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$



Inverz kinematika

$$x(\theta_1,\theta_2) = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y(\theta_1,\theta_2) = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

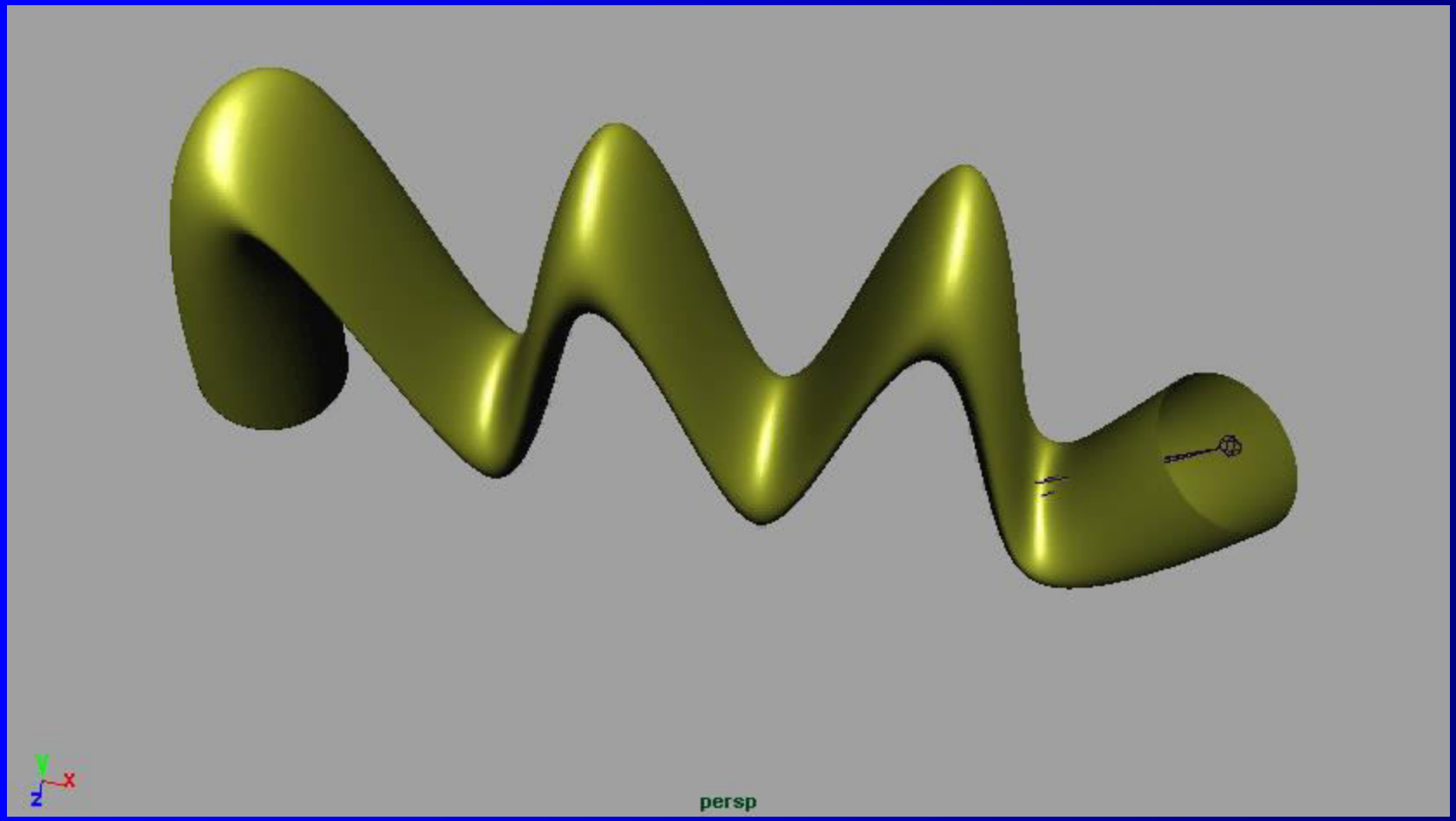
$$\theta_1(x,y) = ???$$
$$\theta_2(x,y) = ???$$

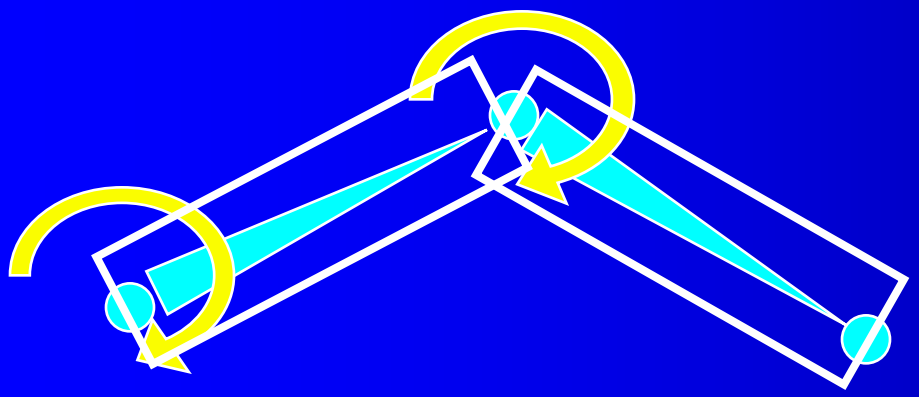
Iteratív megoldás:

$\mathbf{J}(\theta_1, \theta_2)$ Jacobi mátrix

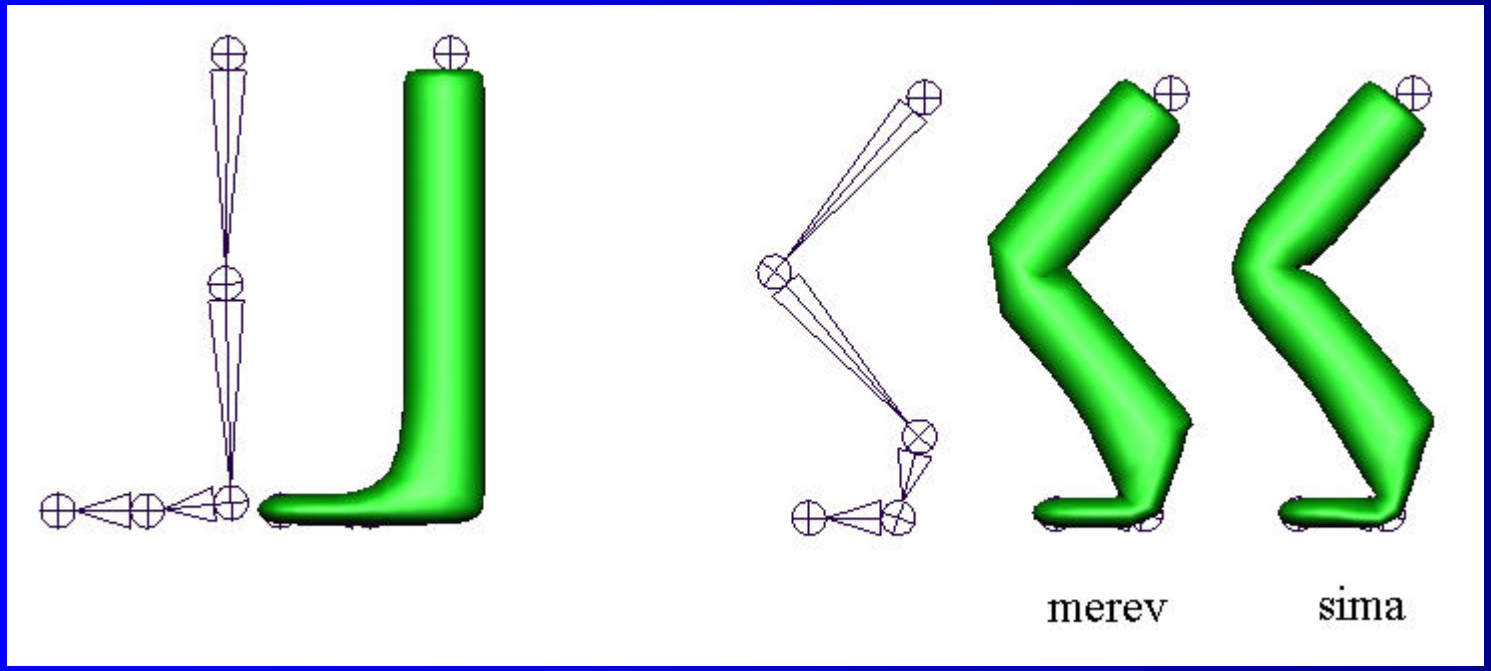
$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} -l_1 \sin\theta_1 - l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) \\ l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} d\theta_1 \\ d\theta_2 \end{bmatrix}$$

Bőrözés

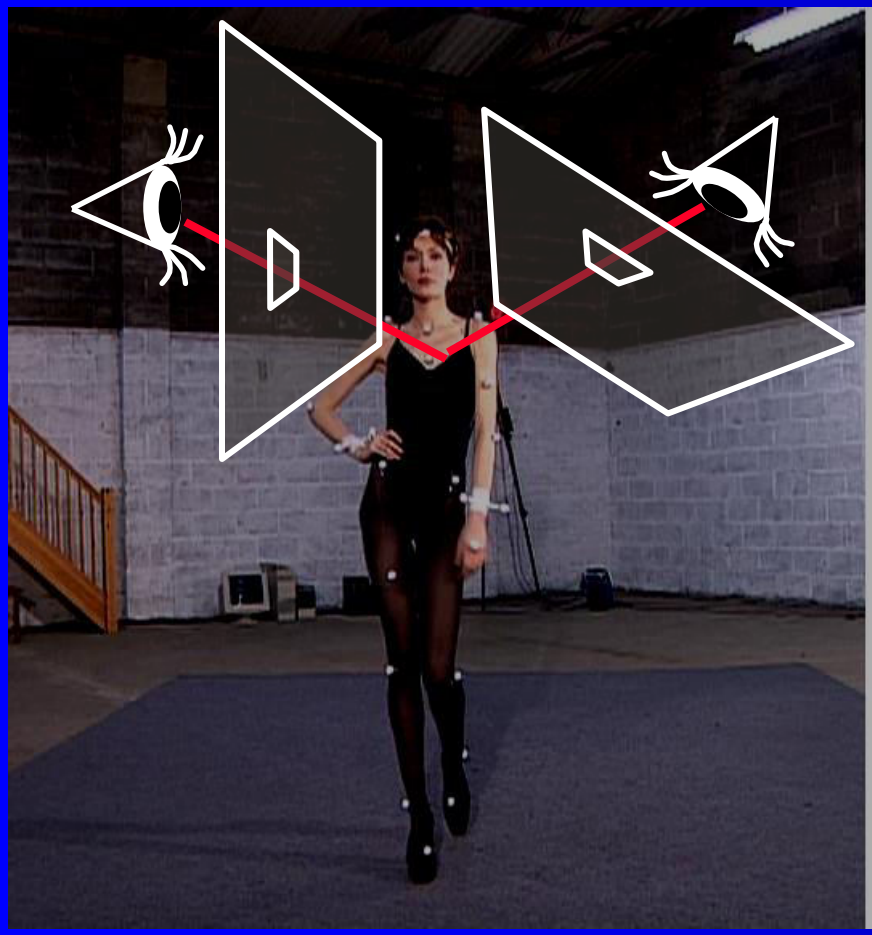




Merev és sima bőrőzés

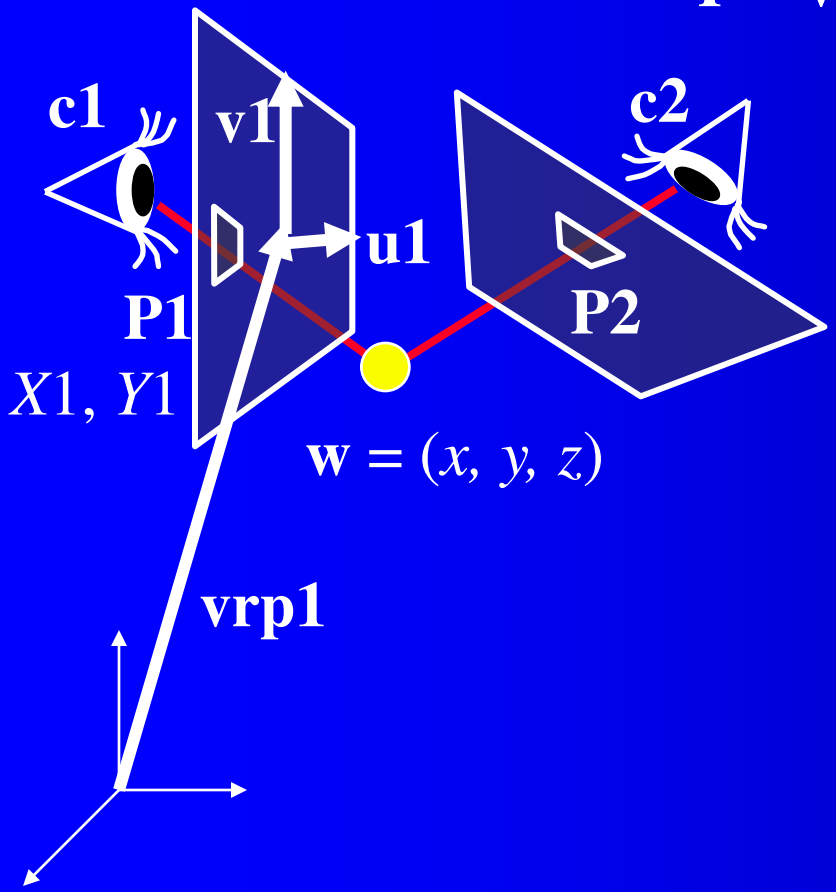


Motion Capture animáció





Sztereo látás

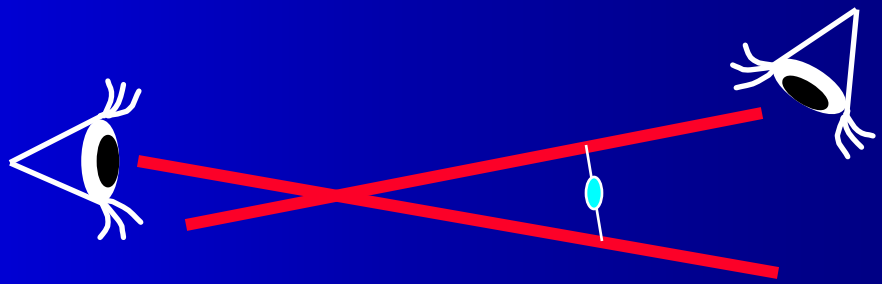


$$\mathbf{P} = \mathbf{vrp} + (2X/\mathrm{XMAX}-1)\mathbf{u} + (2Y/\mathrm{YMAX}-1)\mathbf{v}$$

$$\mathbf{c1} + (\mathbf{P1}-\mathbf{c1}) \cdot t1 = (x, y, z)$$
$$\mathbf{c2} + (\mathbf{P2}-\mathbf{c2}) \cdot t2 = (x, y, z)$$

6 egyenlet, 5 ismeretlen $(x, y, z, t1, t2)$

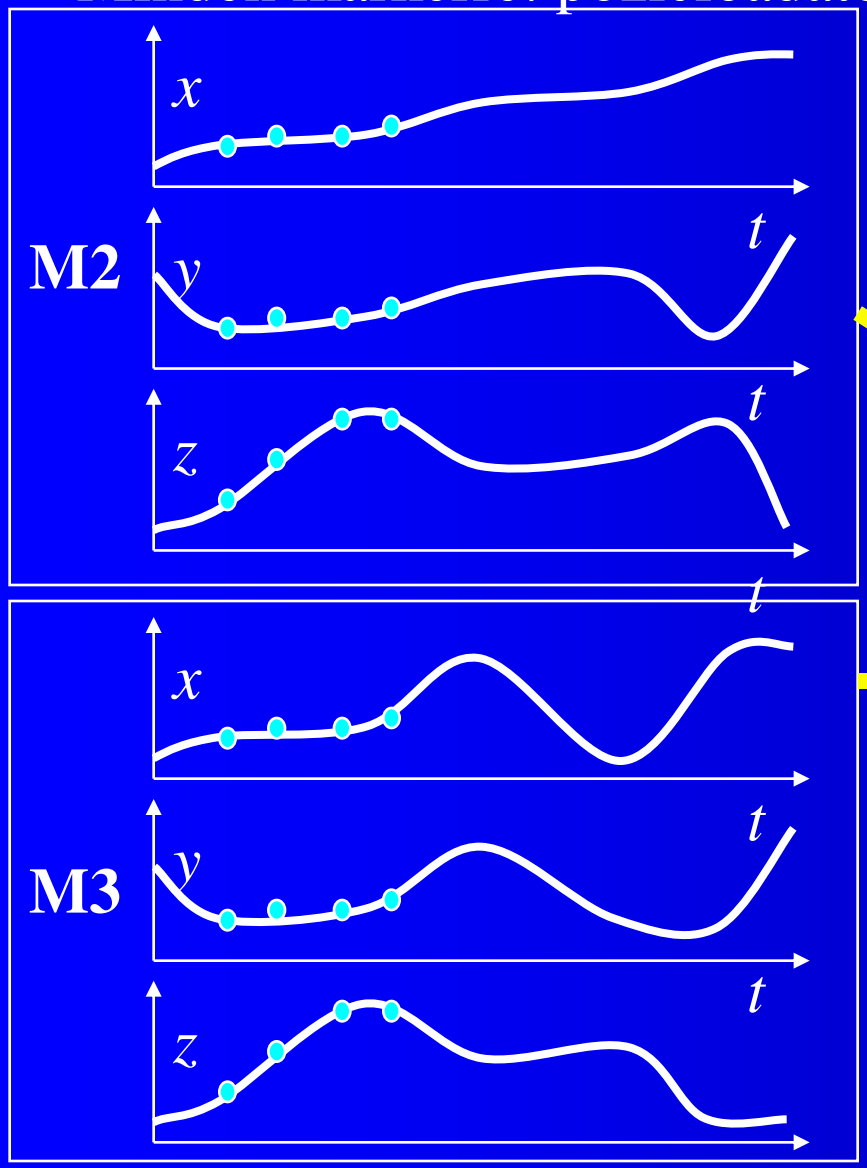
Hiba: kitérő egyenesek



c, u, v, vrp meghatározása: kamera kalibráció

Motion capture eredmény

Minden markerre: pozicióadatok



M1

M2

M3

M4



$\cos\phi=(M3-M2)^0 \cdot (M2-M1)^0$

tengely:

d=(M3-M2) x (M2-M1)

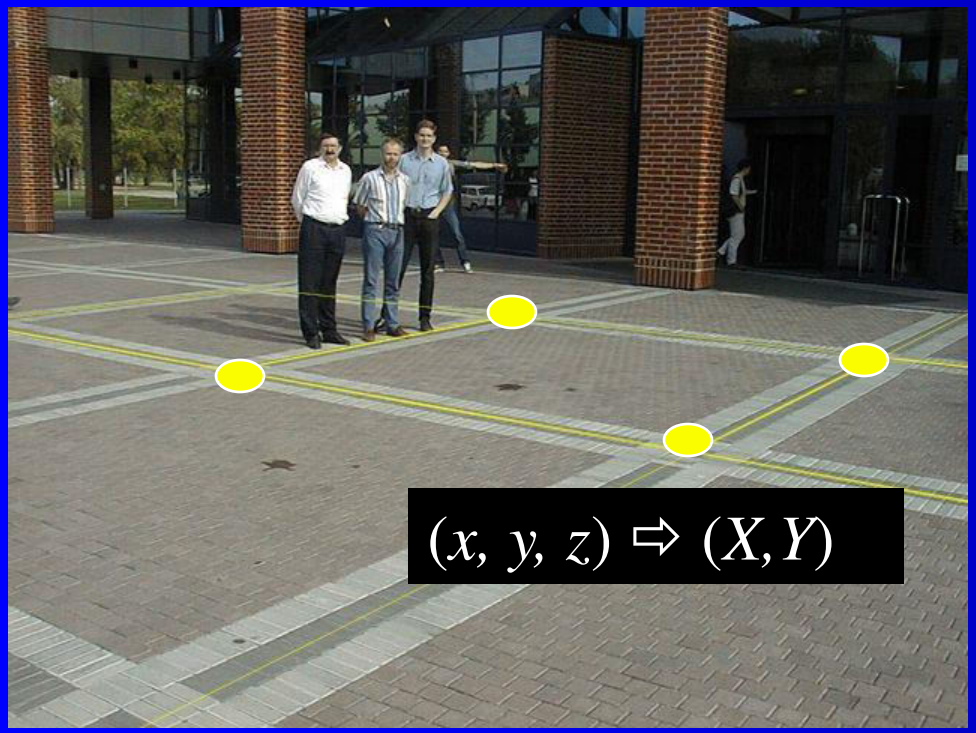
Valóságos és virtuális világok képi illesztése

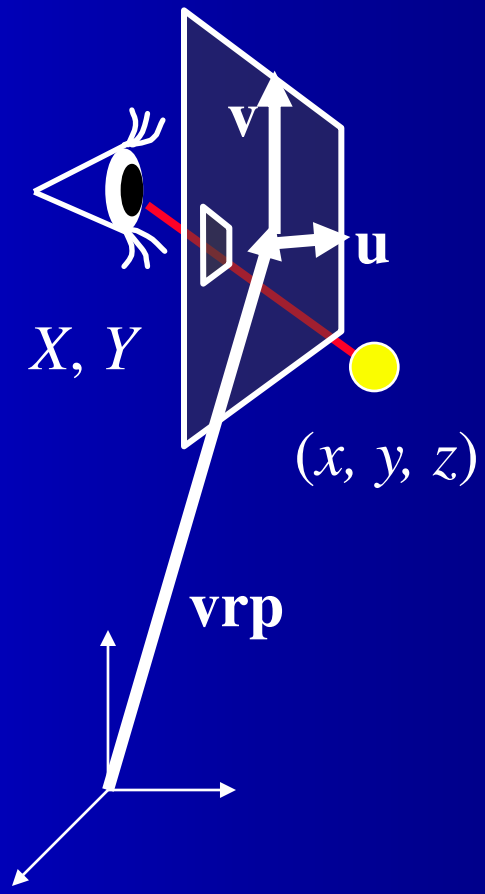




Blaskó Gábor

Kamera kalibráció





Ismeretlenek: **vrp**, f, **u**, **v**

3 + 1+2+2 = 8

Kompozitálás







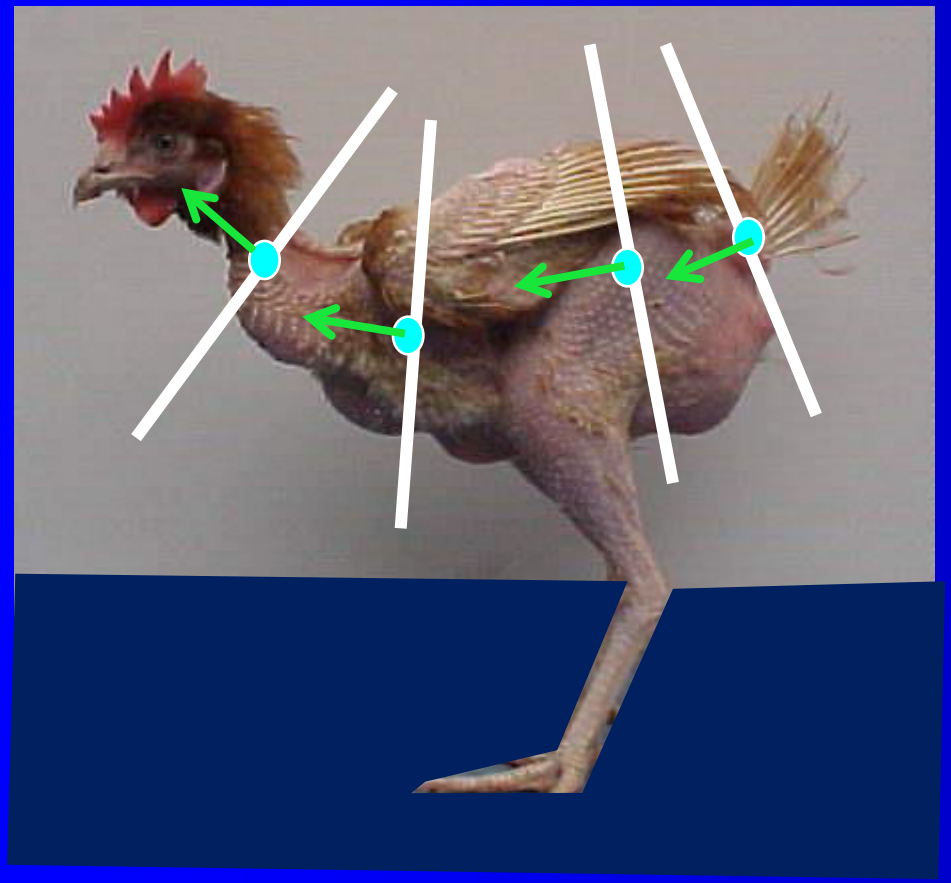

3. házi: csirguru

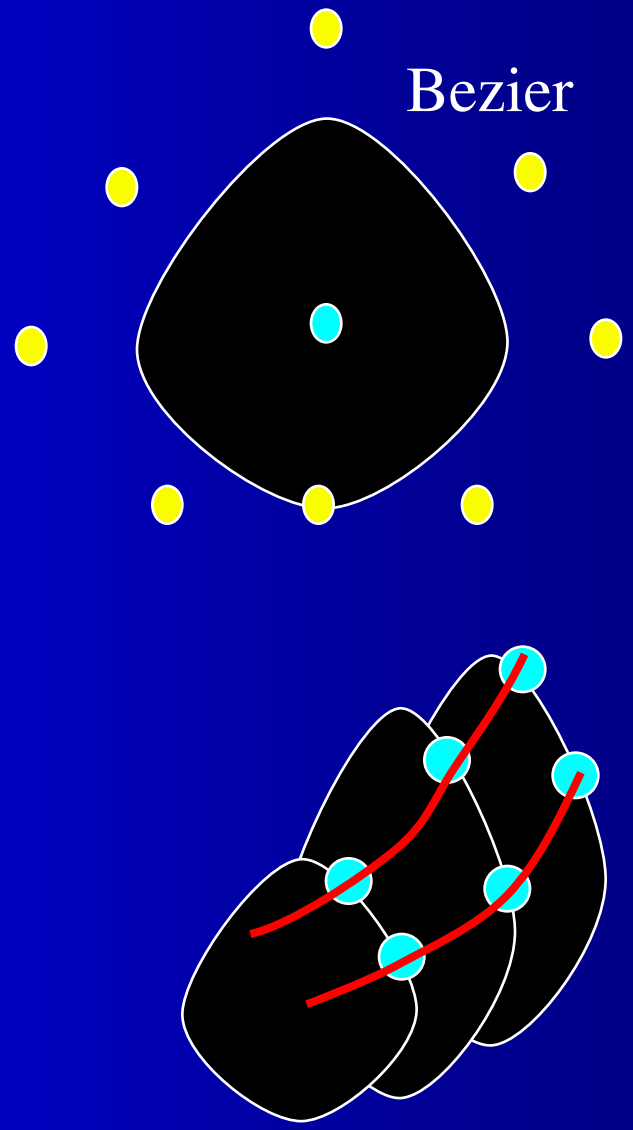
Készítsen CSIRGURU BOMBÁZÓ játékot. A csirguru a csirke és a kenguru nászából született jószág, csirke teste, csőre, szemei, taraja van, valamint egy lába, amivel kenguruszerűen ugrál. A lába a csípőízületben nem fordul el, a térízületben hátra fordul, a bokában előre (amit a normál csirke esetében hajlamosak vagyunk előre hajló térdnek hinni). Az egy szem lábujj ugyancsak felfelé hajlítható, az ízületek tengelyei egymással párhuzamosak.

A csirguruk testét néhány zárt Bézier görbével definiált keresztmetszetek Catmull-Rom spline-t alkalmazó transzfinit interpolációjával kell létrehozni. A szem, csör, taraj, lábrészek ismert egyenletű paraméteres felületekből (pl. gömb, kúp, henger, paraboloid stb.) összerakhatók.

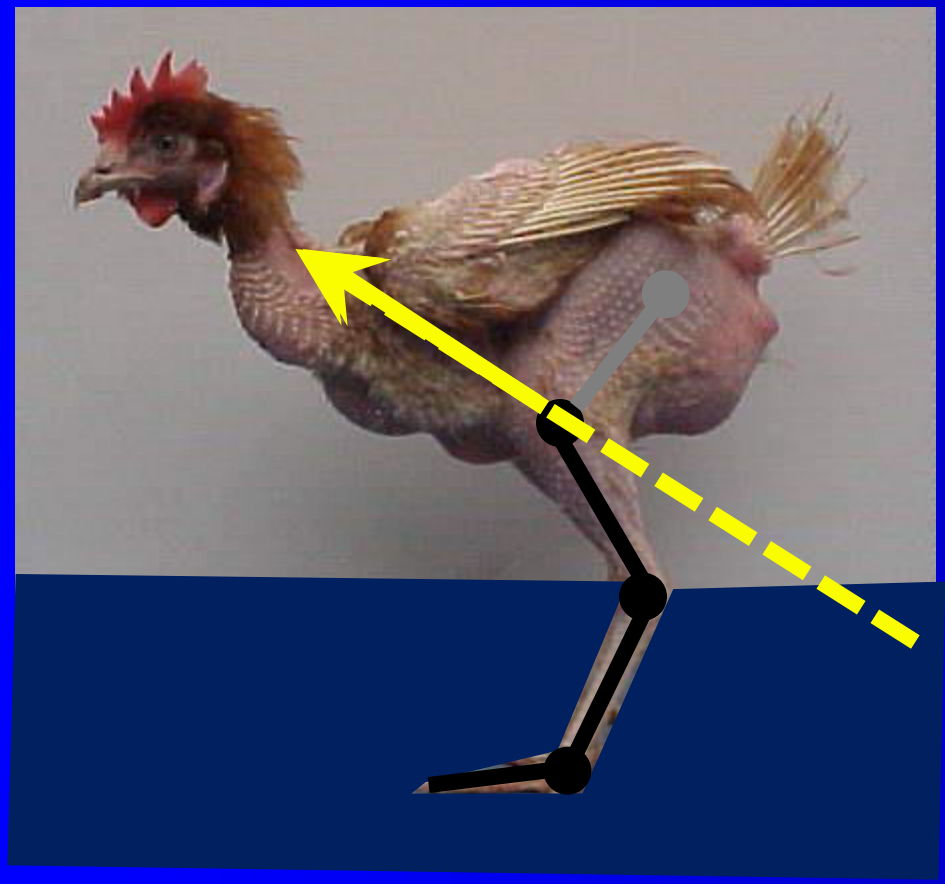
A csirguruk egy sík, textúrázott és megvilágított terepen a bomba aktuális helye alatt, a program indításától számított másodpercenként születnek meg, majd elkezdenek ugrándozni. A lendületvételi fázisban a csirguru feneke a földhöz közelít, majd a térdízület EGYENES VONALÚ GYORSULÓ MOZGÁST (inverz kinematika) végez, mialatt a lábujj a talajon marad, a térd pedig vízszintesen tartja a csirgurut. A bokaízület kiegyenesedése pillantában a csirguru elszakad a földtől és lendületvétel végén elért kezdősebességű ferde hajítással leírható pályán ugrik, a levegőben előreteszi a lábát, a földetérkezésnél pedig a lendületvételt visszafelé játsza le. Az ugrálást ismételgetheti, az egyes ugrálások között a haladási irányát ízlésesen változtatva.

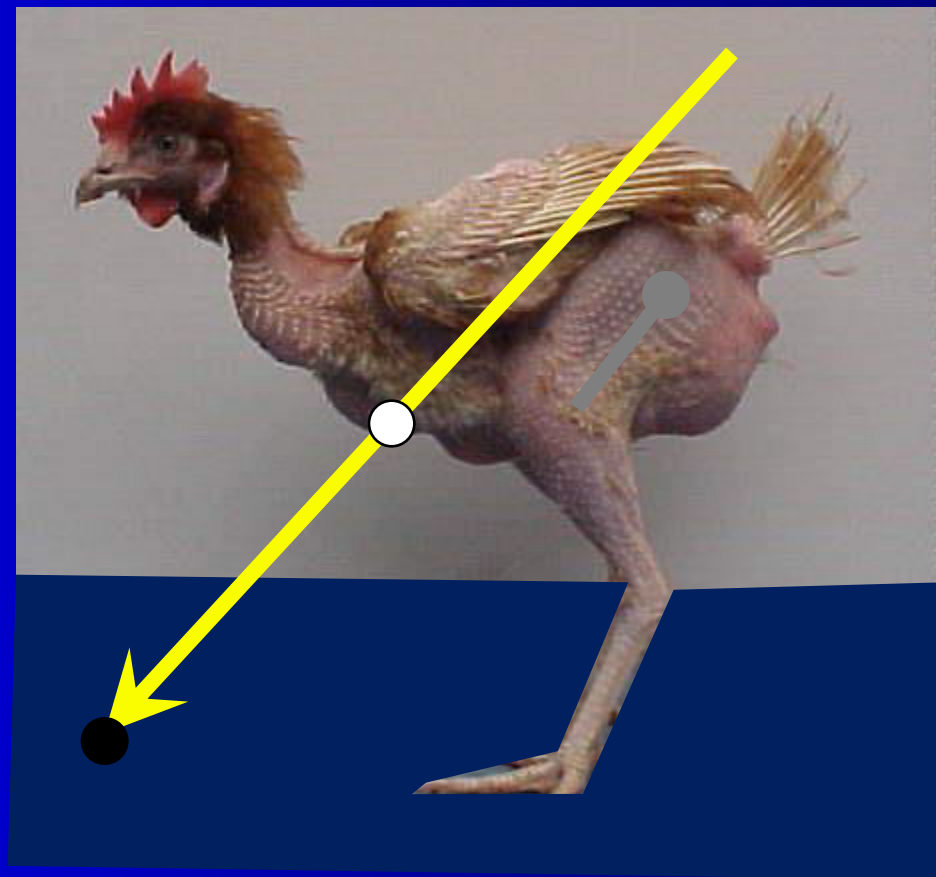
3. házi: csirguru





Csirguru animáció és árnyék

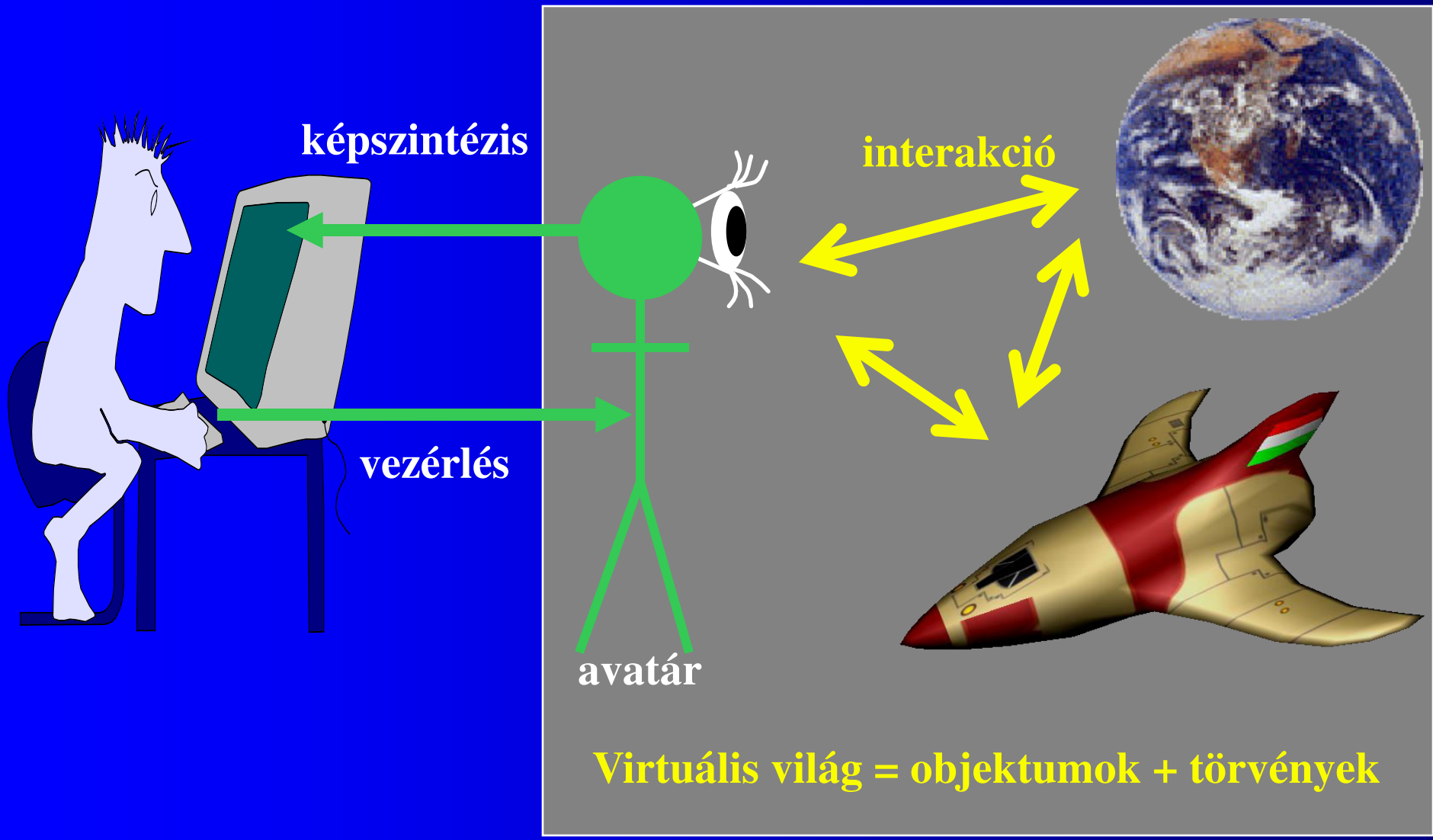




Játékfejlesztés

Szirmay-Kalos László

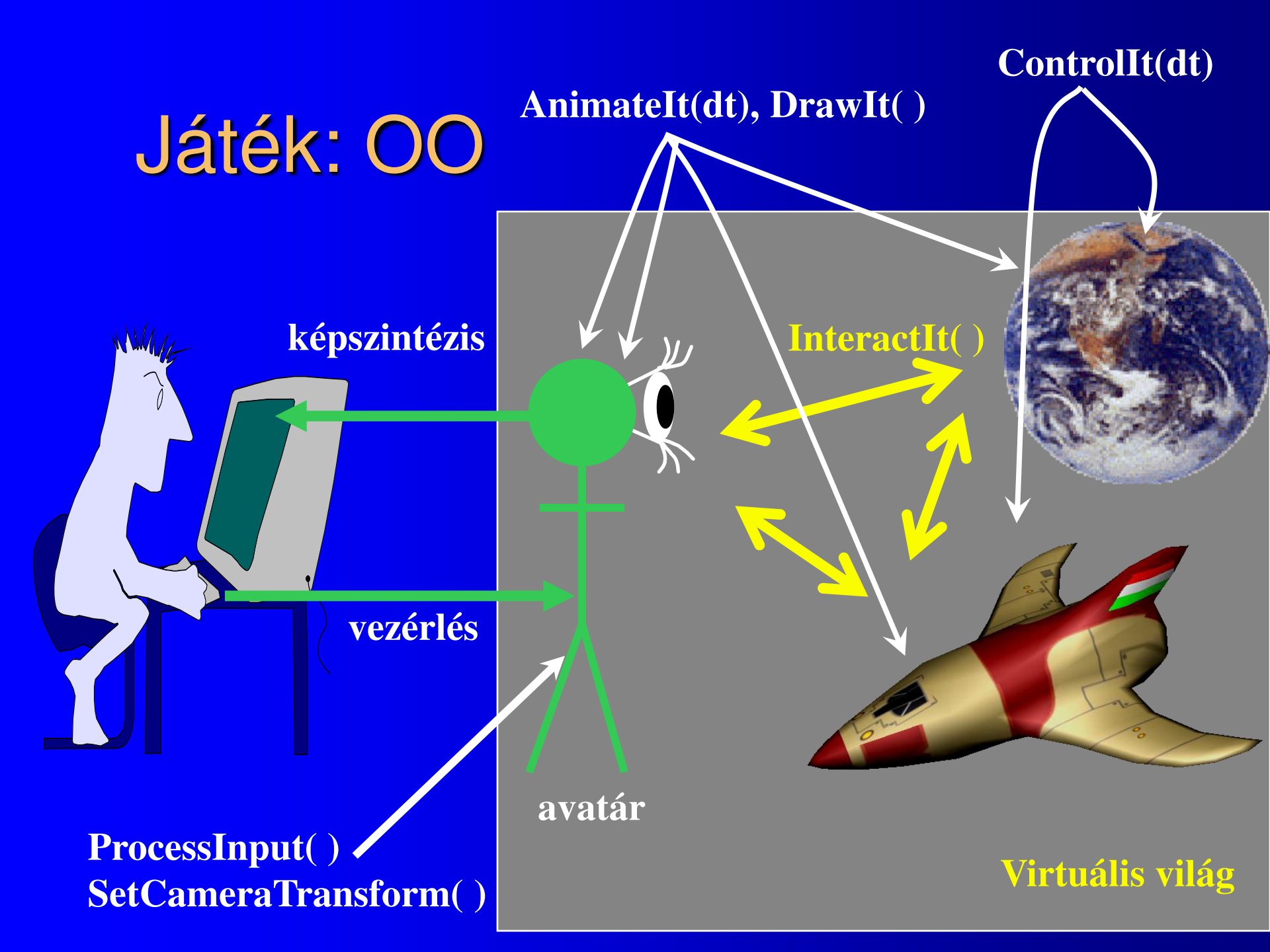
Virtuális valóság



Játékok feladatai

- Képszintézis az avatár nézőpontjából
- Az avatár vezérlése a beviteli eszközökkel (keyboard, mouse, Wii, gépi látás, stb.)
- Az „intelligens” virtuális objektumok vezérlése (AI)
- A fizikai világ szimulációja

Játék: OO

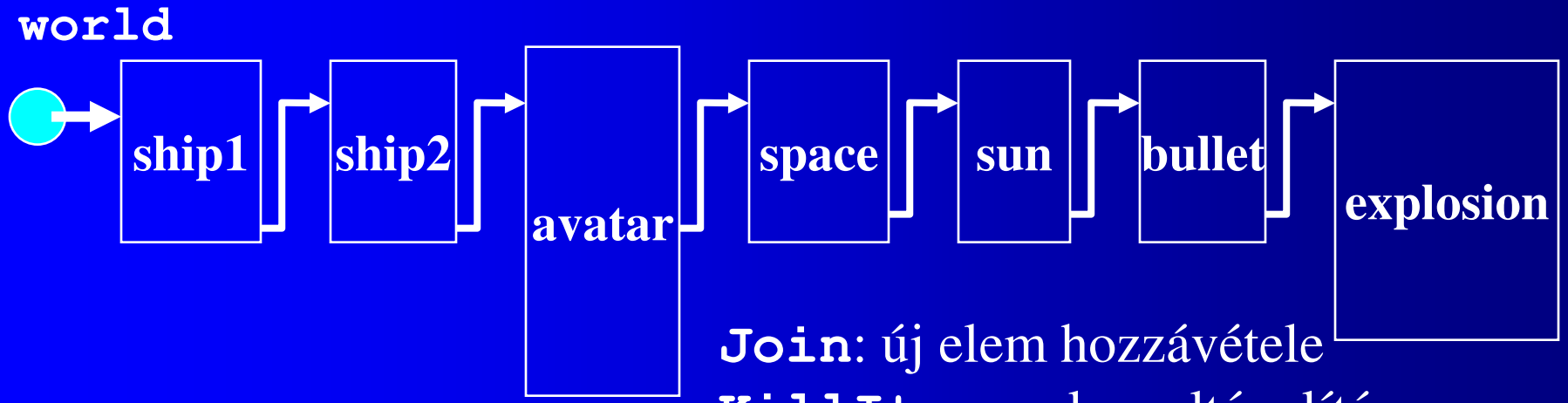

ControlIt(dt)
AnimateIt(dt), DrawIt( )
képszintézis
InteractIt( )
vezérlés
avatár
ProcessInput( )
SetCameraTransform( )
Virtuális világ

Játékobjektum: GameObject

- **`ControlIt()`**:
  - Tiszteletben tartja a virtuális világ törvényeit + „gondolkodik” és a lehetséges vezérléseket alkalmazza (pl. rakéták)
- **`InteractIt()`**:
  - Tájékozódik mások állapotáról
- **`AnimateIt()`**:
  - Mozog
- **`DrawIt()`**:
  - Lefényképeződik = átadja magát az OpenGL-nek

A virtuális világ (Scene graph)

- Objektumok dinamikusak (öldöklés)
- Heterogén kollekció (GameObject)
- Láncolt lista (fák)

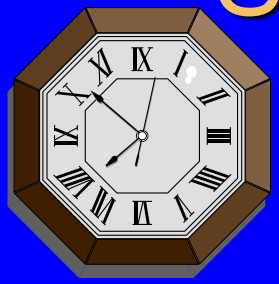


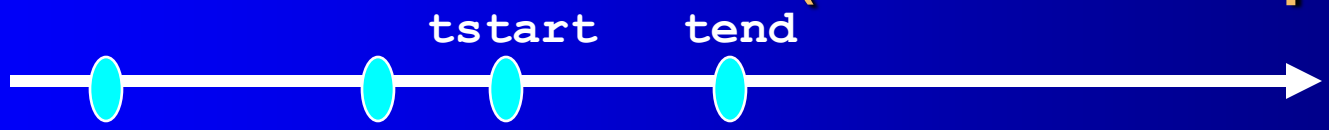
`Join`: új elem hozzávétele
`KillIt`: egy elem eltávolítása

```
Control()
Animate()
Draw()
```

Iteráció a kollekcióban

Szimulációs hurok (Game loop)

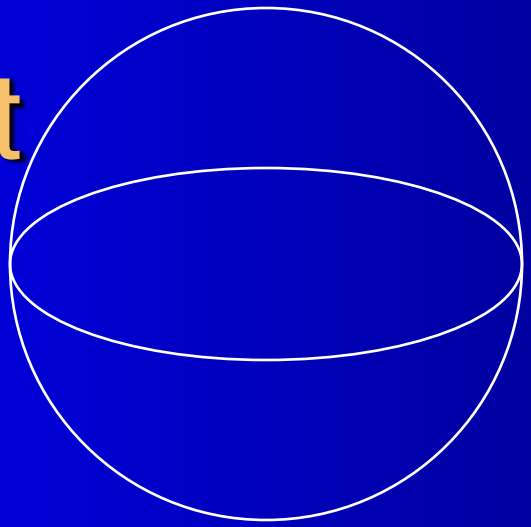





```
float tend = 0;

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( );
}
```

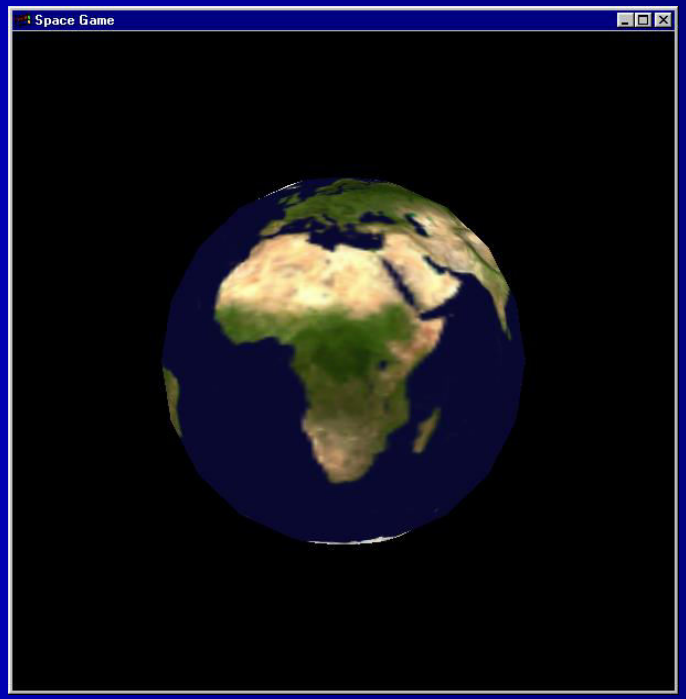


Bolygó: Planet





- Geometria: gömb
- Textúra
- Fizikai vagy
  - Tájékozódik majd követi a gravitációs törvényt
- Képletanimáció:
  - „beégetett pálya”
  - Többiek érdektelenek
  - Nincs respektált törvény



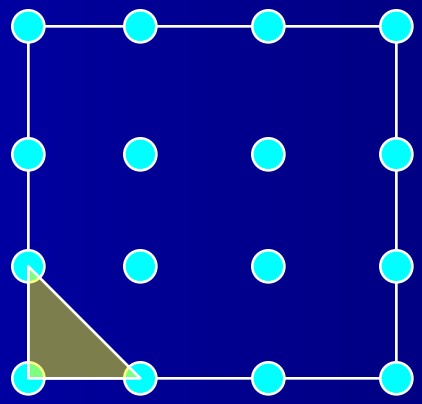
Gömbfelület tesszelláció

1. Paraméteres egyenlet:

$x = x0 + r \cos 2\pi u \sin \pi v$

$y = y0 + r \sin 2\pi u \sin \pi v$

$z = z0 + r \cos \pi v$ $\qquad u,v \in [0,1]$

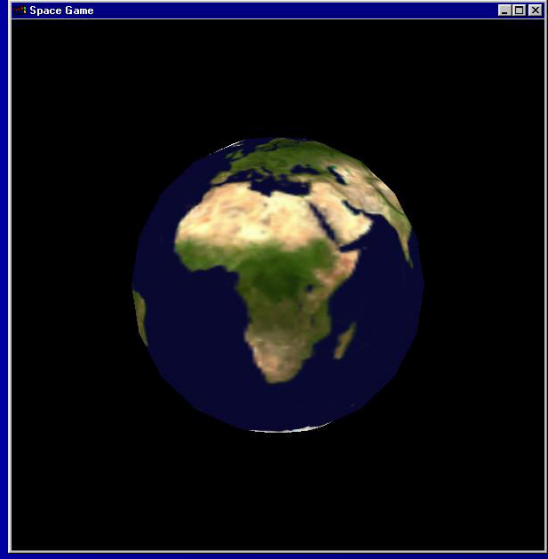


Paramétertér háromszögesítése + behelyettesítés

2. GLU kvadratikus felület:

```
// definíció
GLUquadricObj * quadric = gluNewQuadric( );
gluQuadricTexture(quadric, GL_TRUE);

...
// Rajzolás
gluSphere(quadric, R, 16, 10);
```

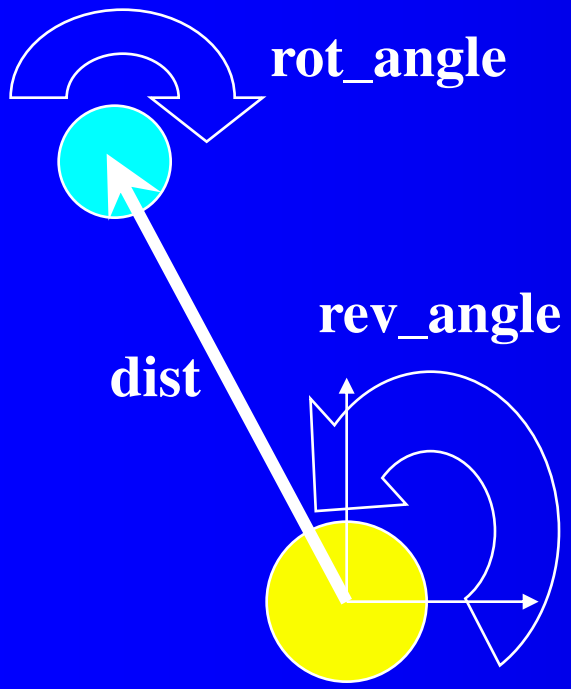
Planet: Animate, Draw



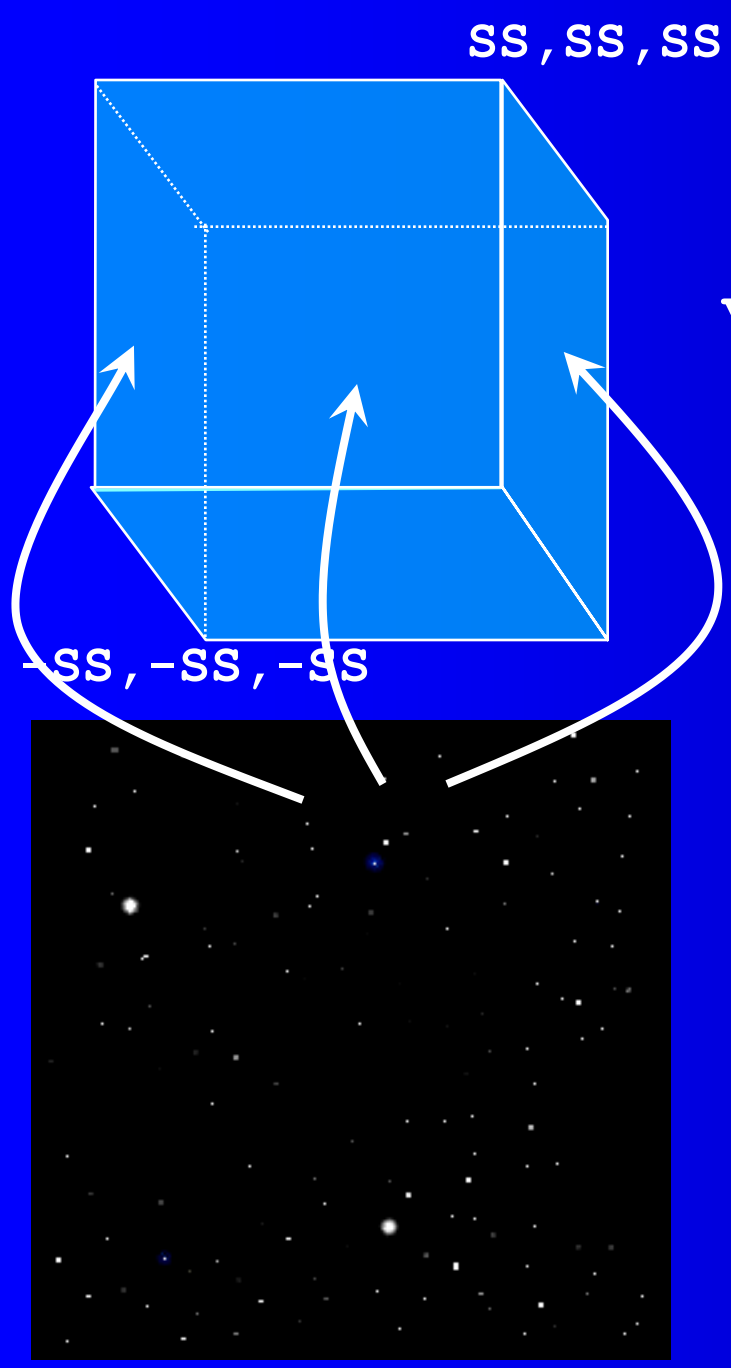

```
class Planet : public GameObject {
   float rot_angle, rot_speed, radius;
   unsigned int texture_id;
   GLUquadricObj * quadric;
public:
   Planet( … ) { … }
   void ControlIt( float dt ) { }
   void InteractIt( GameObject * o ) { }
   void AnimateIt( float dt ) { rot_angle += rot_speed * dt; }
   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 kering a Nap körül



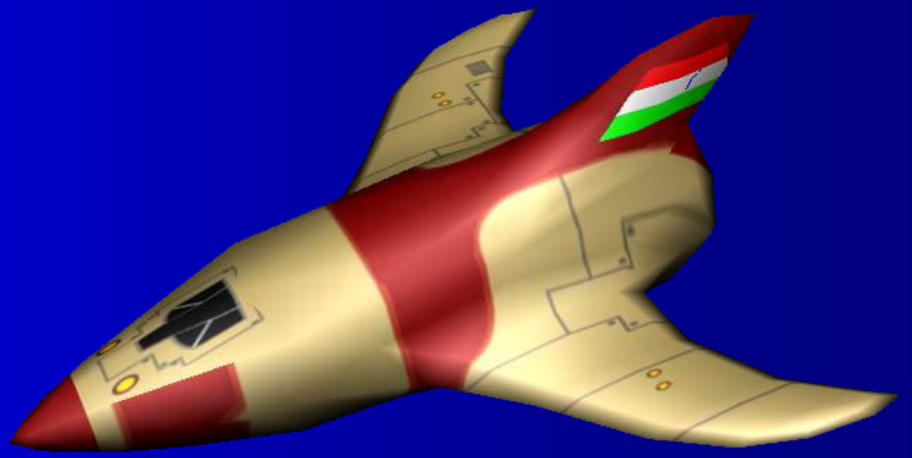

```
void Planet :: AnimateIt(float dt){
    rot_angle += rot_speed * dt;
    rev_angle += rev_speed * dt;
}

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( );
}
```



Az űr

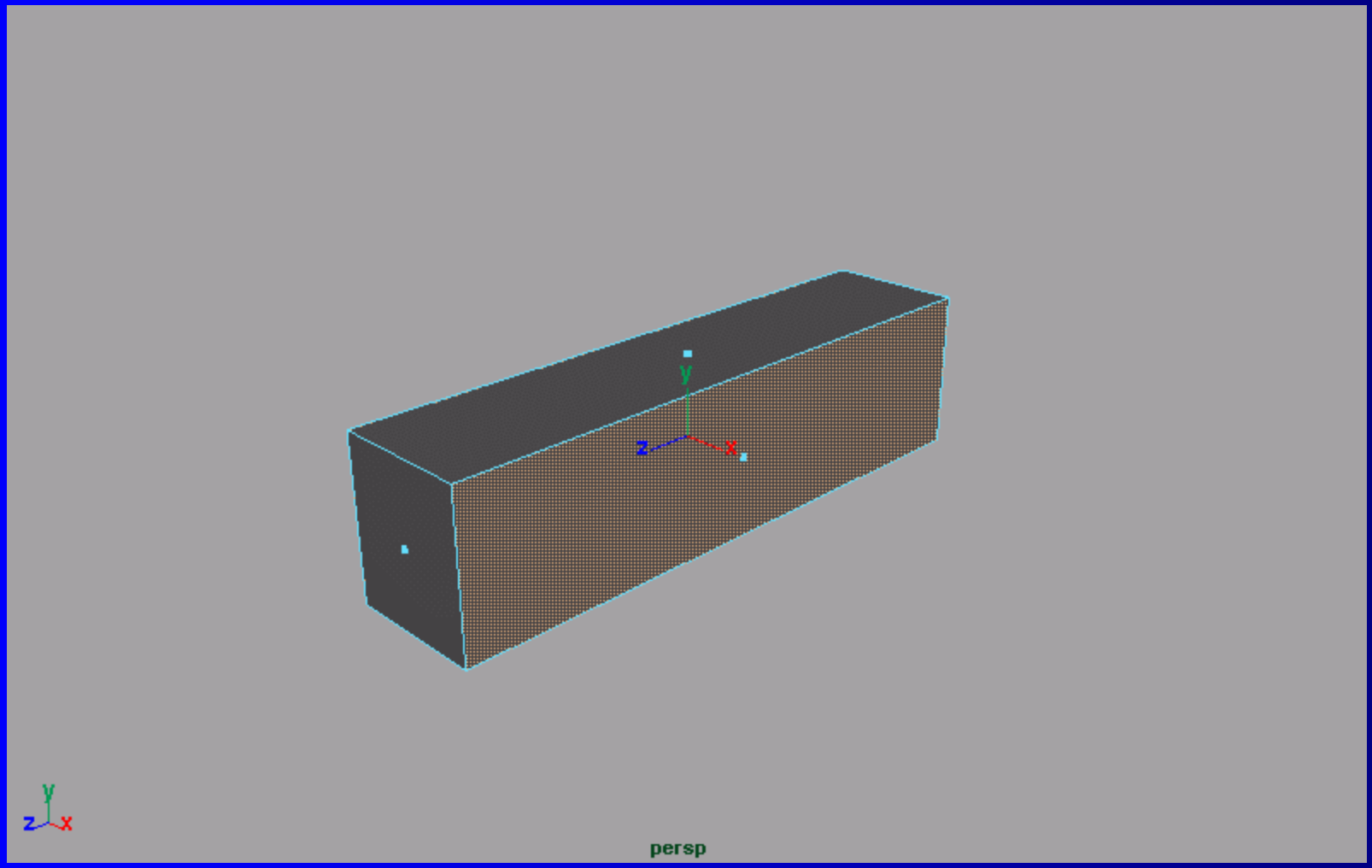
```
void Space :: DrawIt( ) {
    glBindTexture(GL_TEXTURE_2D,
                  space_texture);

    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

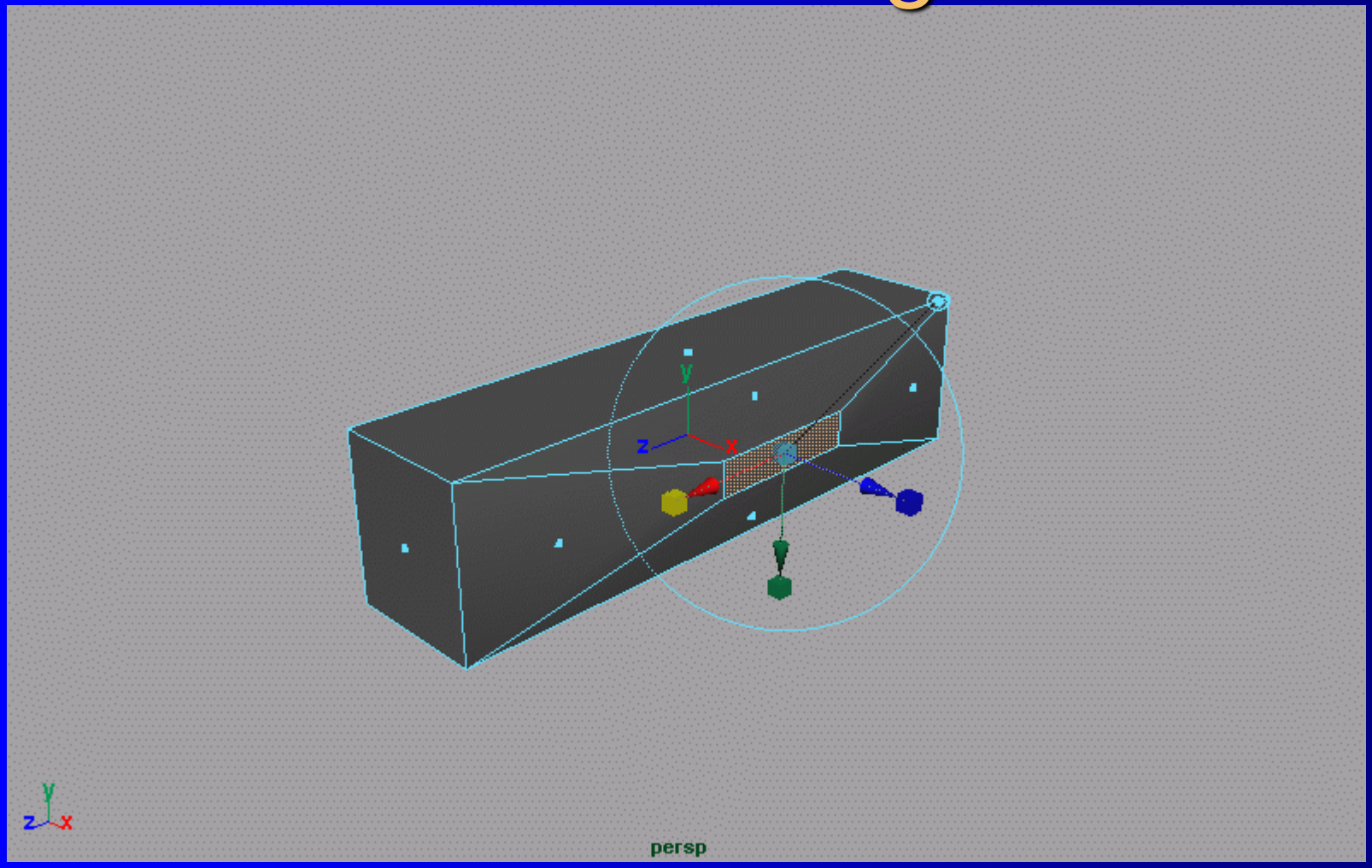


- Komplex geometria
  - négyszögháló
- Komplex textúra
- Fizikai animáció
  - erők (gravitáció, rakéták)
  - ütközések
- Viselkedés (AI)
  - A rakéták vezérlése
    - Ütközés elkerülés, avatártól menekülés, avatár üldözése

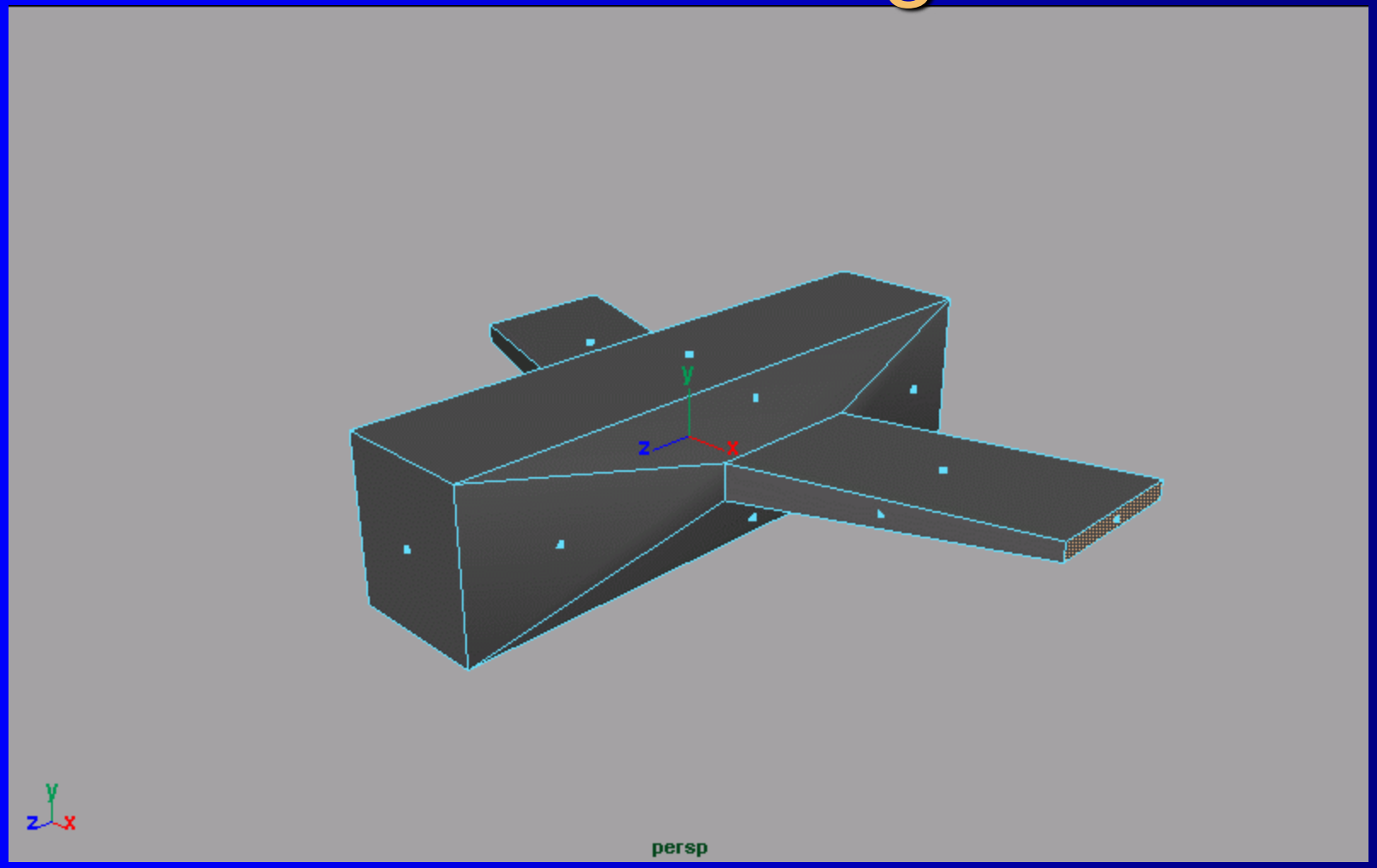
Űrhajó geometria



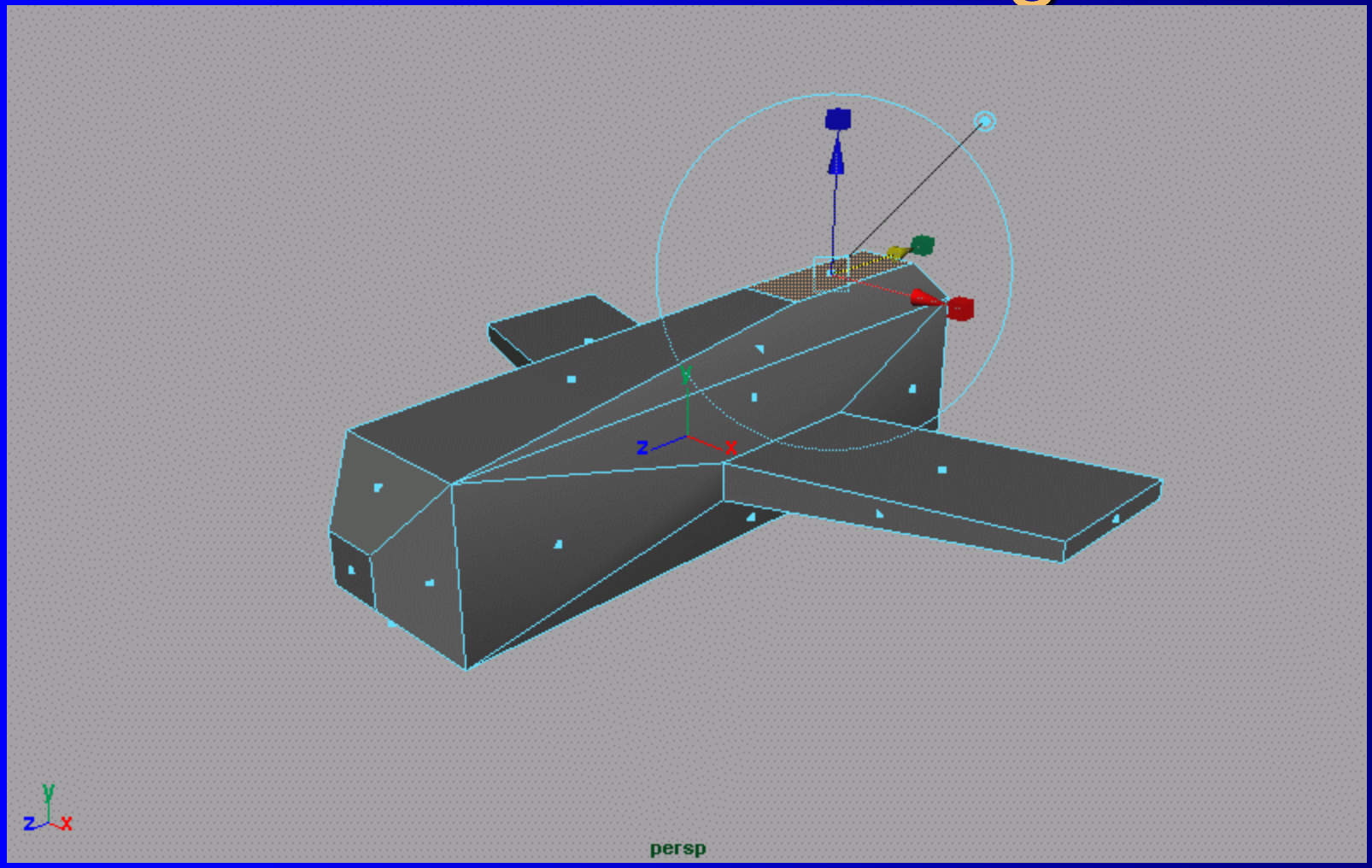
Poligon modellezés:
1. extruding



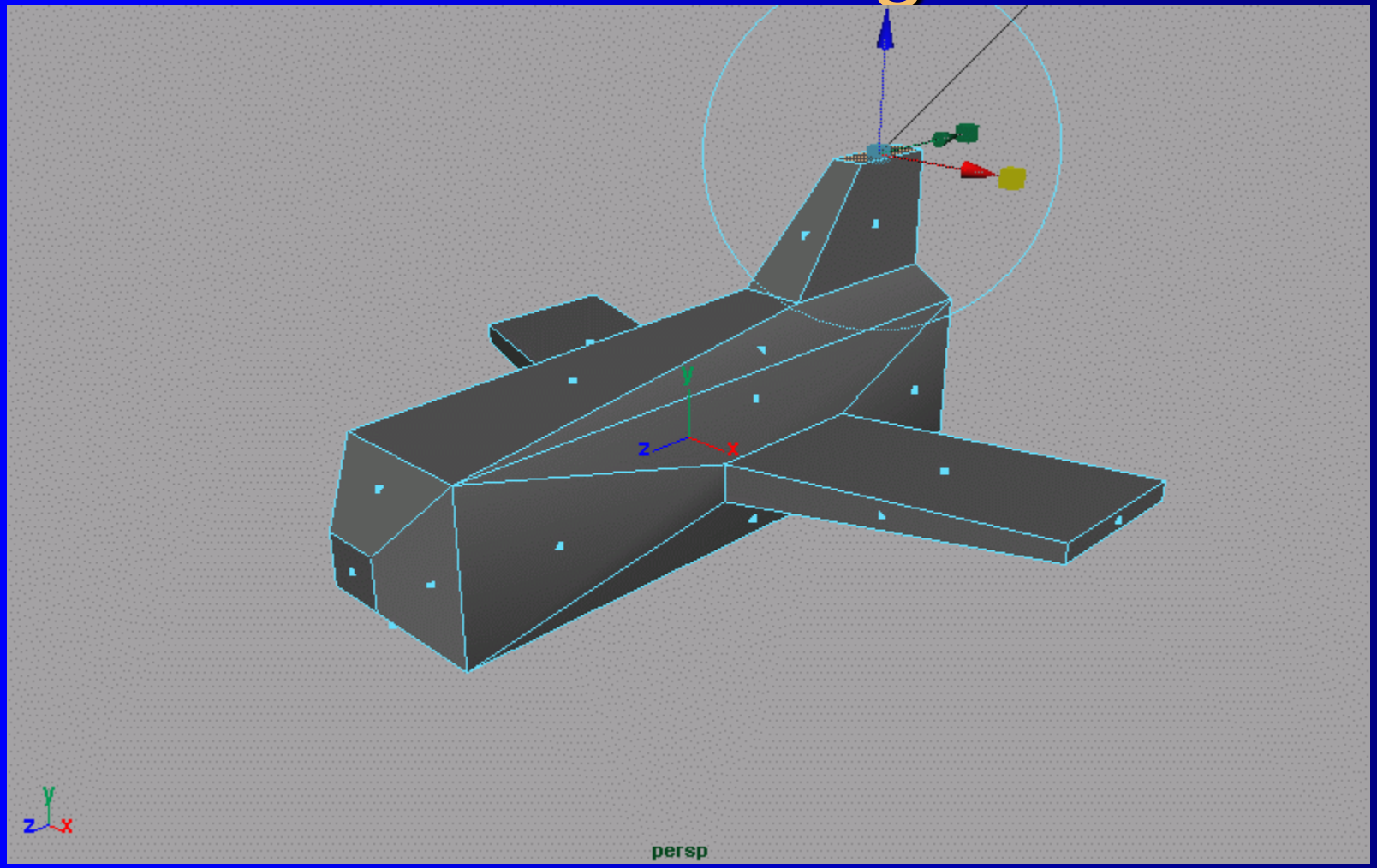
Poligon modellezés: 2. extruding



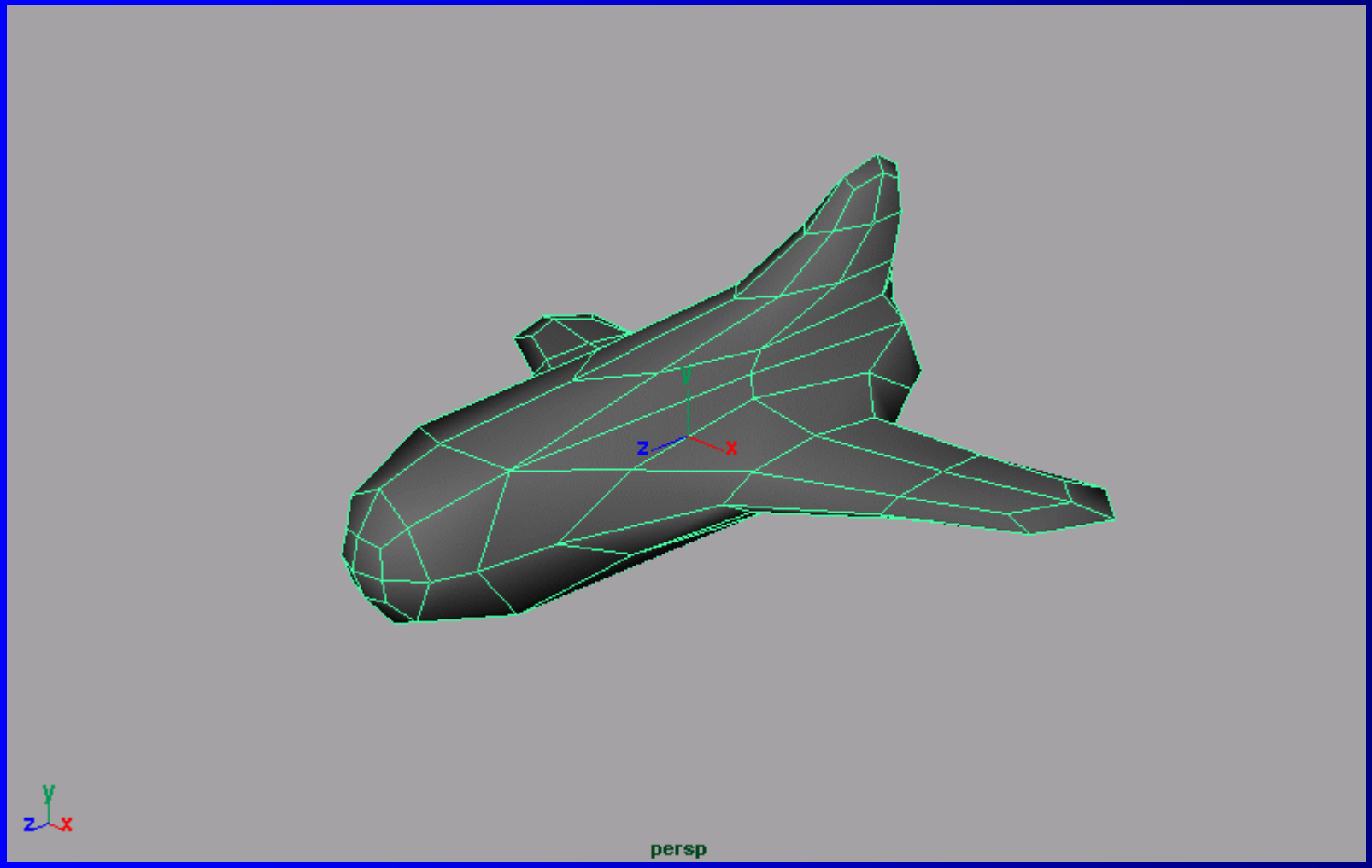
Poligon modellezés: 4. és 5. extruding



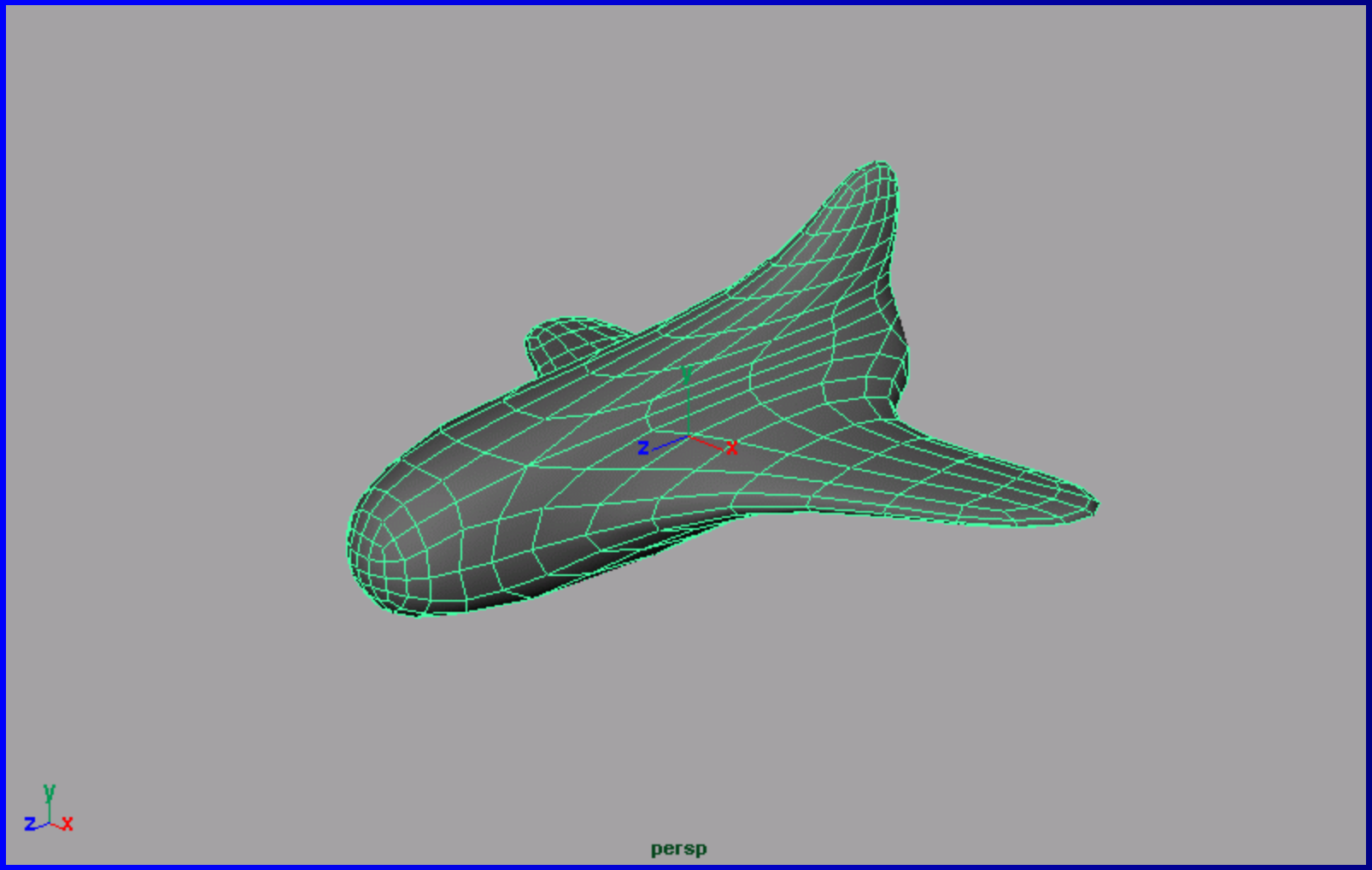
Poligon modellezés: 6. extruding



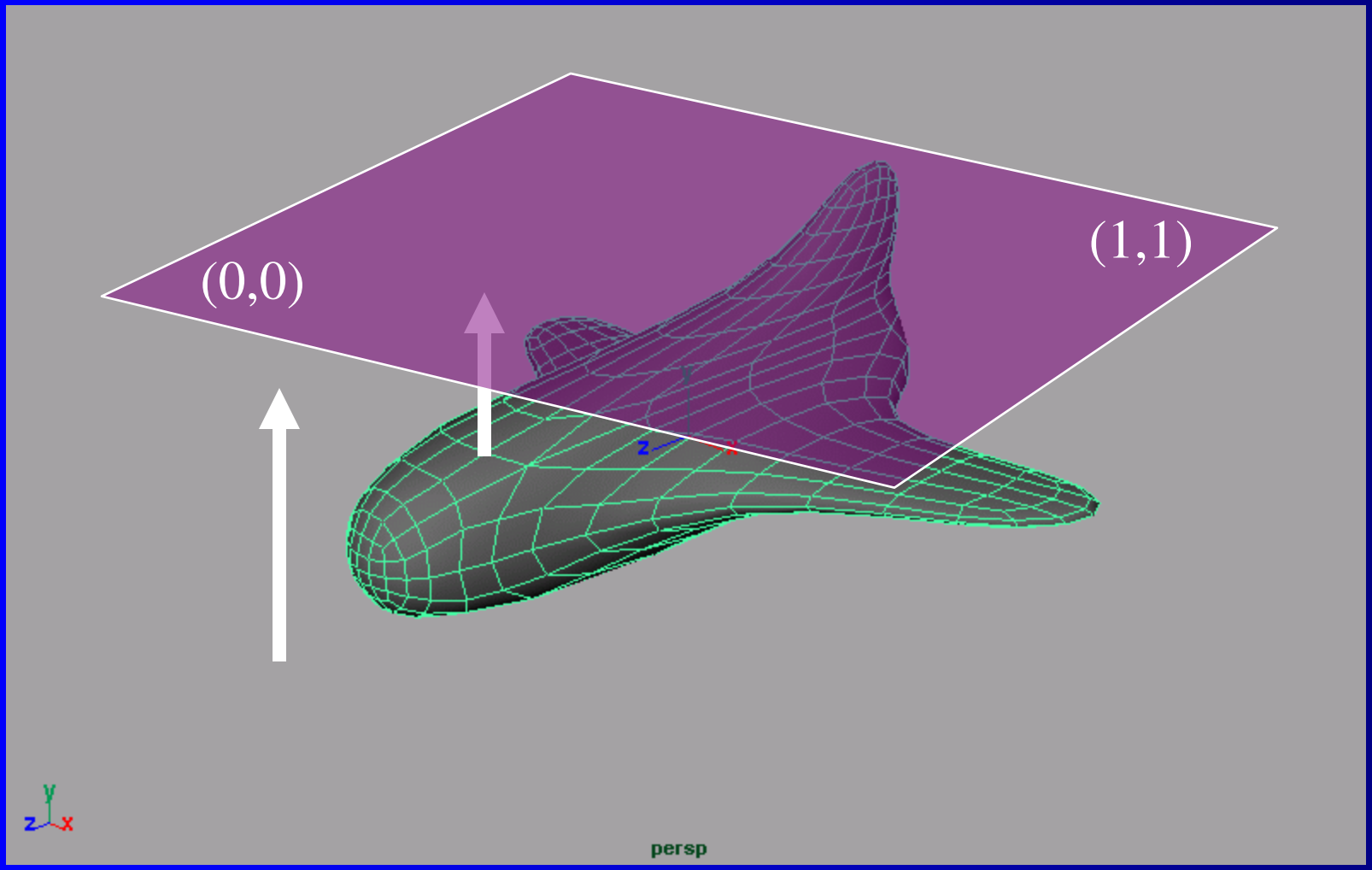
Subdivision simítás: 1 szint



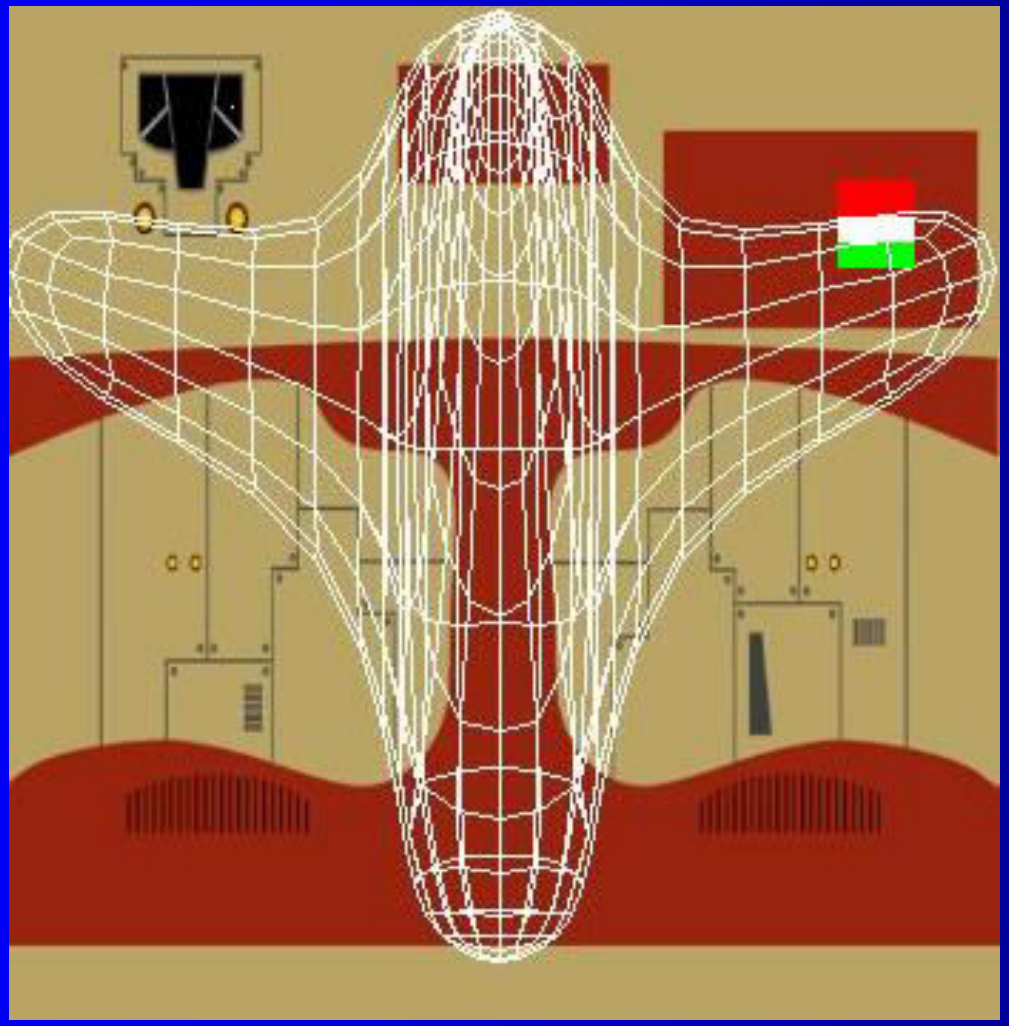
Subdivision simítás: 2. szint



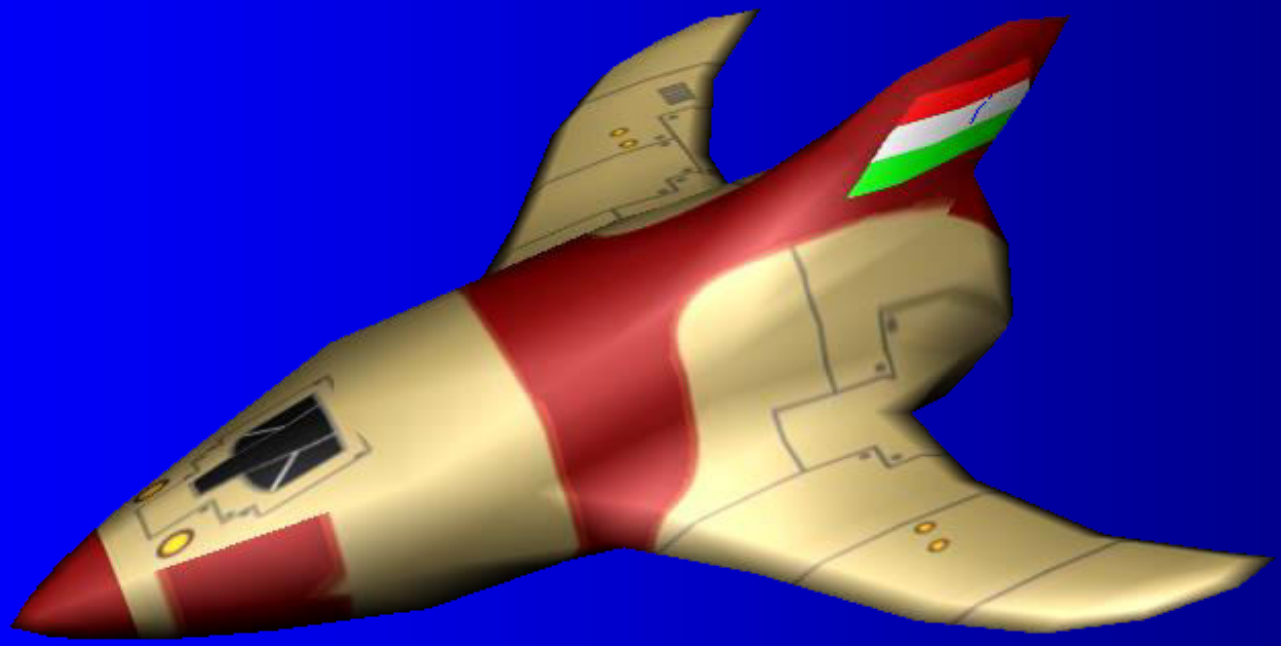
Textúra függvény definíciója



Textúra függvény definíciója



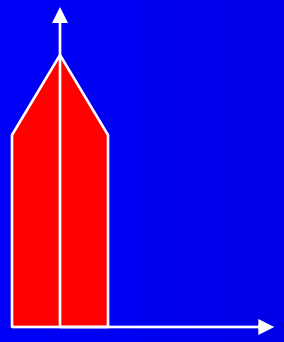
Textúrázott űrhajó

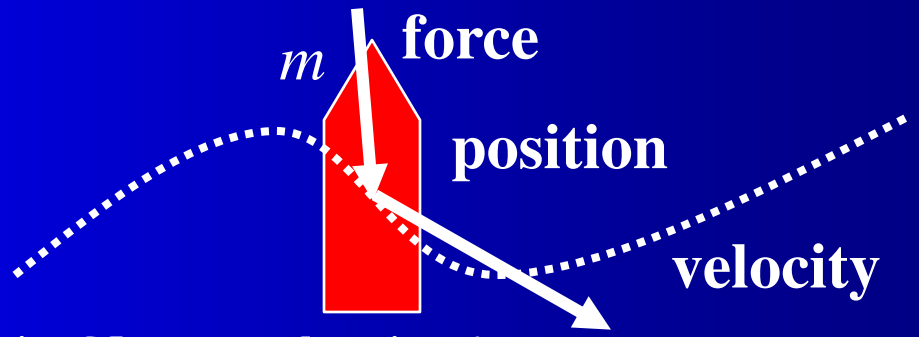


Spaceship OBJ formátumban

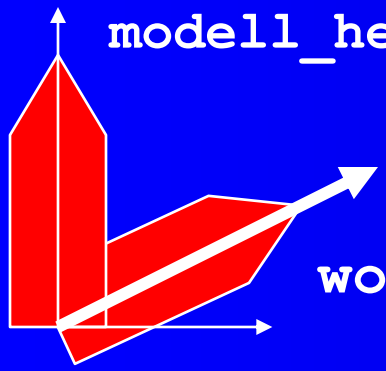
```
v -0.708698 -0.679666 2.277417
v  0.708698 -0.679666 2.277417
v -0.735419  0.754681 2.256846
...
vt 0.510655 0.078673
vt 0.509594 0.070000
vt 0.496429 0.079059
...
vn -0.843091  0.000000 0.537771
vn -0.670151 -0.543088 0.505918
vn -0.000000 -0.783747 0.621081
…
f 65/1/1 37/2/2   62/3/3   61/4/4
f 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: Newton mozgástörvényei






```
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();
}
```



Orientáció beállítása

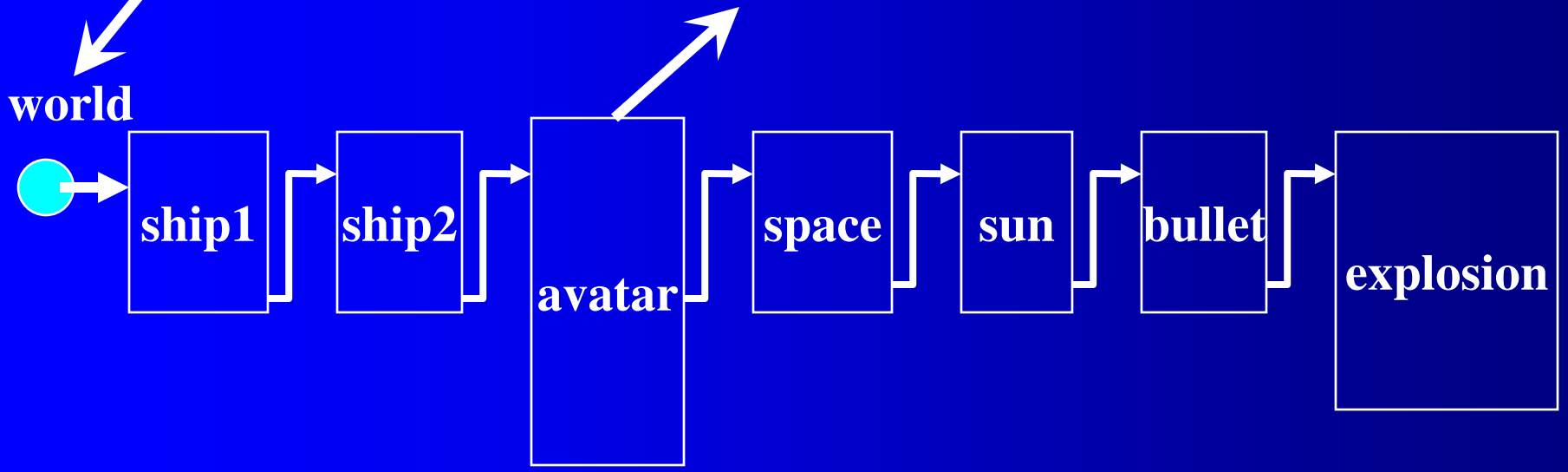
world_head = velocity.UnitVector();

```
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 :: ControlIt

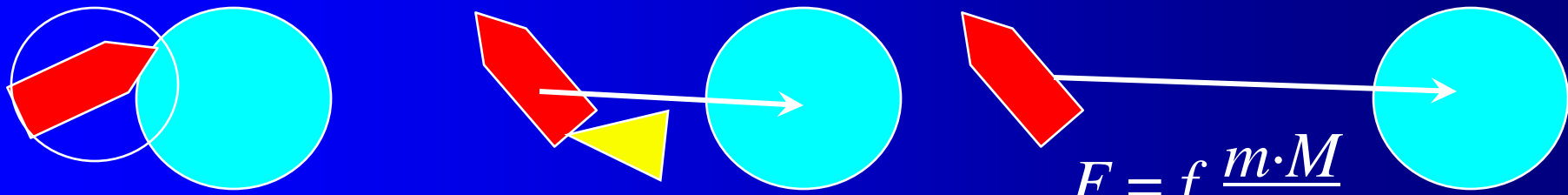
```
void Ship :: ControlIt( float dt ) {
    force = Vector(0, 0, 0);
    Interact( world );
}
```

```
void Ship::InteractIt( GameObject * object )
```



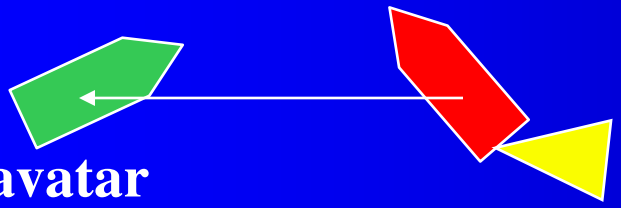
Ship: InteractIt

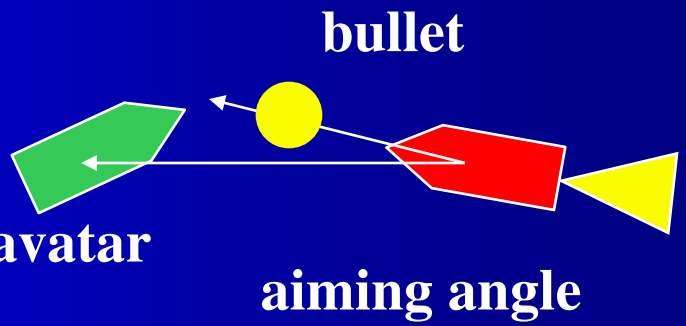
```
void Ship :: InteractIt( GameObject * object ) {
    if ( object->GetType( ) == PLANET ) {
```



$$F = f\,\frac{m \cdot M}{r^2}$$

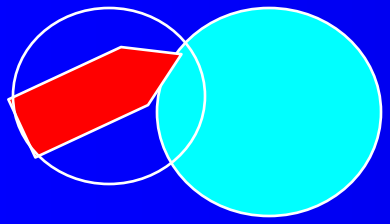
```
    }

    if ( object->GetType( ) == AVATAR ) {
```





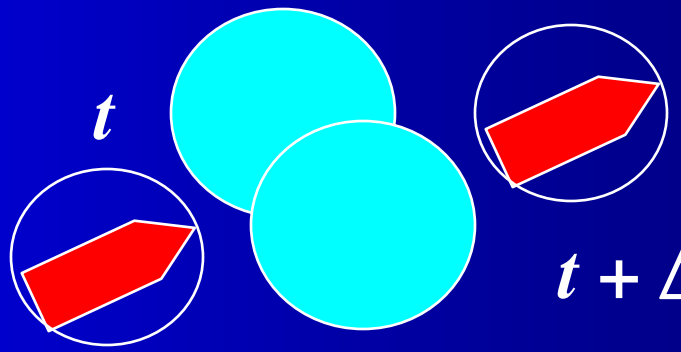

```
    }
}
```

Ütközésdetektálás lassú objektumok között



adott *t*

Probléma, ha az objektum gyors




```
dist = obj1.position - obj2.position
min = obj1.BoundingRadius() + obj2.BoundingRadius()
if (dist.Length() < min) Collision!
```

Lövedék

- Nagyon komplex geometria
- Hasonló kinézet minden irányból
- Könnyebb a képét használni



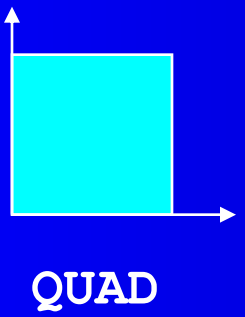


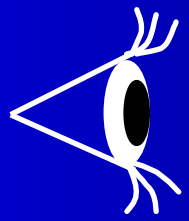
- Ütközésdetektálás = gyors mozgás

Plakátok: Billboard

Egyetlen félig átlátszó textúra egy téglalapon

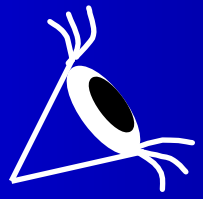




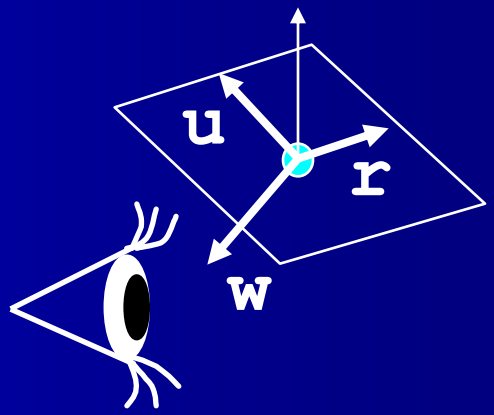




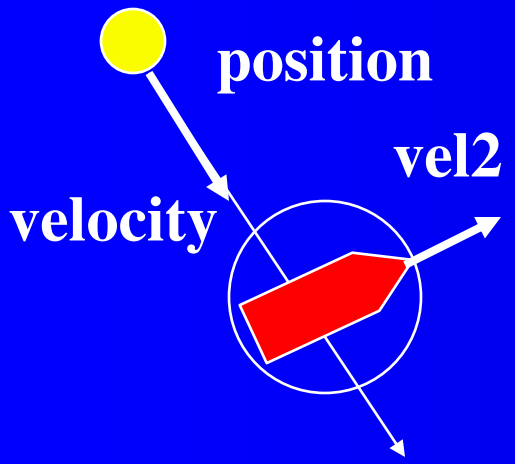




```
void Bullet :: DrawIt() {
    glPushMatrix( );
    Vector w = eye - position;
    Vector r = w % Vector(0, 1, 0);
    Vector u = r % w;
    r = r.UnitVector() * size;
    u = u.UnitVector() * size;
    glTranslatef(position.x, position.y, position.z);

    glEnable(GL_BLEND);          // átlátszóság
    glBlendFunc(GL_SRC_ALPHA, GL_ONE); // hozzáadás
    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 ütközésdetektálás: ray-tracing

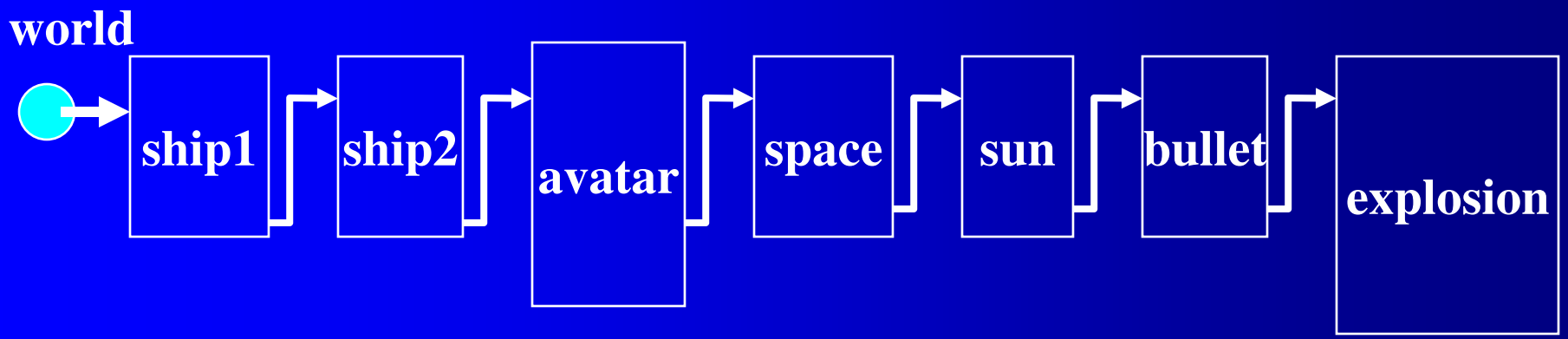


rel_velocity = velocity - vel2

ray: **rel_pos + rel_velocity**·*t*

If (ray intersects bounding sphere first
AND tintersect < dt) ***Collision!***

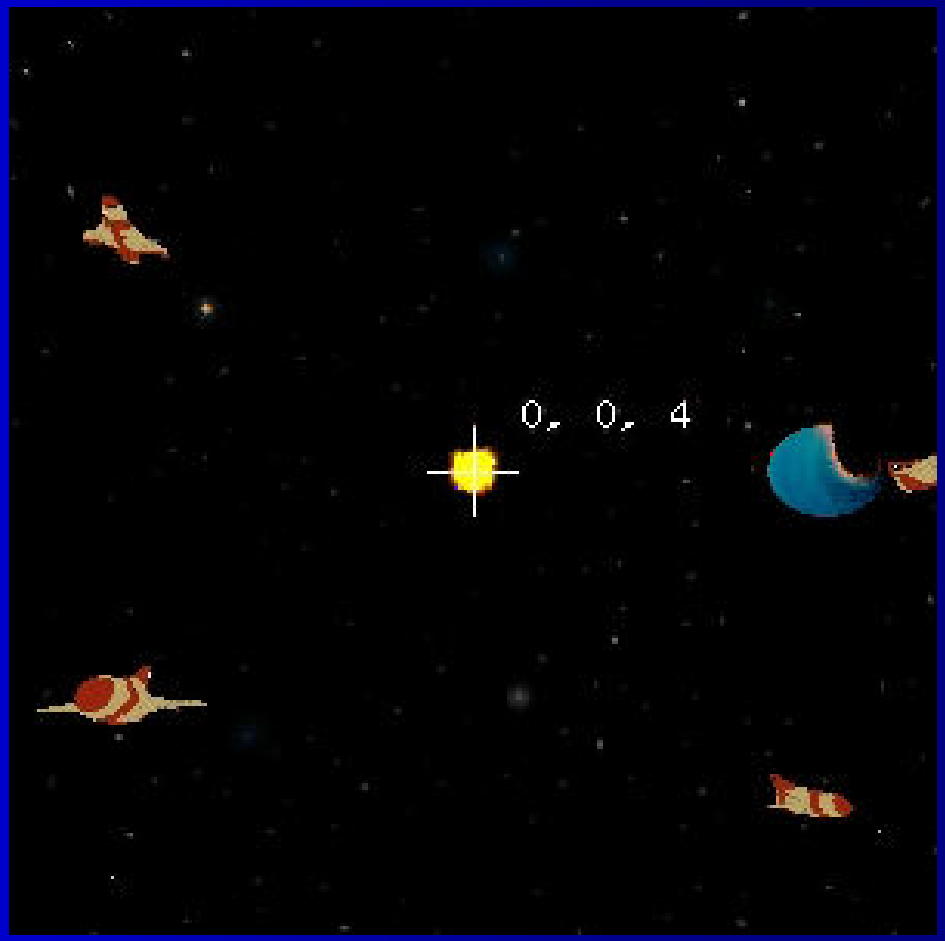
```
hit_object = world->Intersect(position,velocity,t);
```



Robbanás

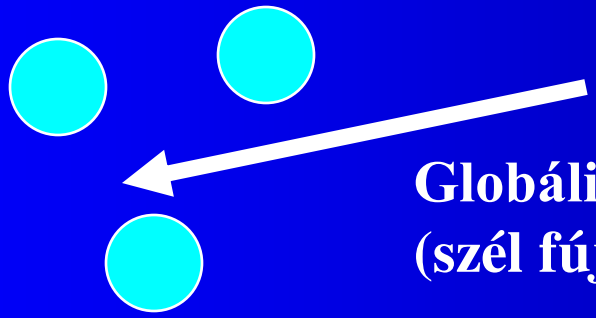
- Nagyon komplex geometria
- Hasonló kinézet minden irányból
- Plakátgyűjtemény

- Részecske rendszer



Részecske rendszerek

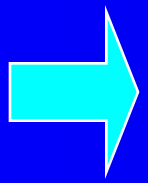




Globális erőtér
(szél fújja a füstöt)

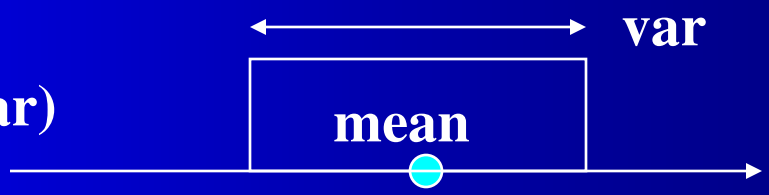


Véletlen
Kezdeti
értékek



position:	position += velocity * dt
velocity:	velocity += acceleration * dt
acceleration:	acceleration = force / weight
lifetime	
age:	age += dt; if (age > lifetime) Kill();
size, dsize:	size += dsize * dt;
weight, dweight:	weight += dweight * dt
color, dcolor:	color += dcolor * dt

Robbanás paraméterei




```
position = center;                          // kezdetben fókuszált
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));

                    // Planck törvény: sárga átlátszatlanból vörös átlátszóba
color = Color(1, Rand(0.5, 0.25) 0, 1 );
dcolor = Color(0, -0.25, 0, -1) / lifetime;
```

Avatár

- A viselkedését a klaviatúra vezérli:
  - ProcessInput
- A helye és iránya viszi a kamerát
  - SetCameraTransform
- Olyan mint egy űrhajó, de nem rajzoljuk
  - Control: gravitáció, lövedék ütközés

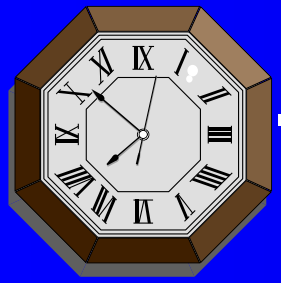
Klaviatúra kezelés



KeyboardFunc
KeyboardUpFunc
SpecialKeysFunc
SpecialKeysUpFunc

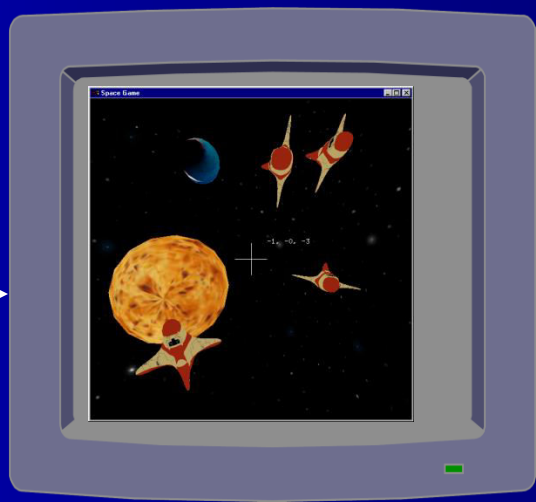
input

IsSpace, IsLeft, IsRight, IsUp, IsDown

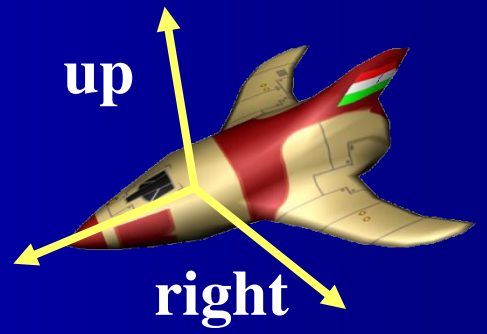


IdleFunc:
GameLoop

virtual
world



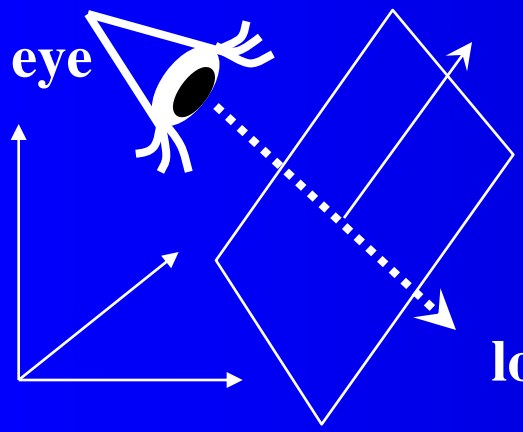
Avatar :: ProcessInput



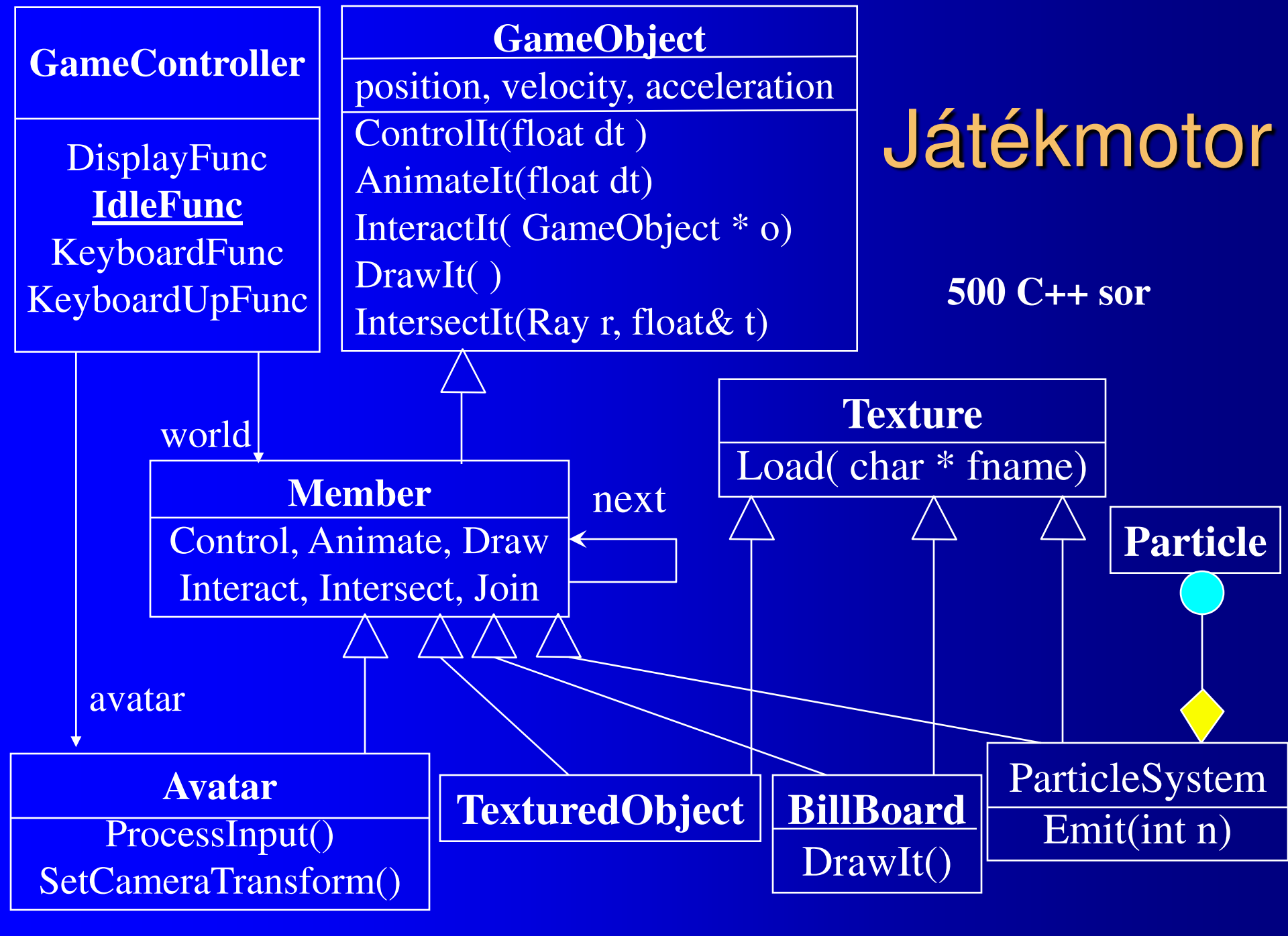

```
Avatar :: ProcessInput( KeyStates * input ) {

   if ( input->IsSpace( ) ) // tűz!
      world -> Join(new Bullet(position, velocity));

   // Kormányzás: az avatár koordinátarendszerében!
   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

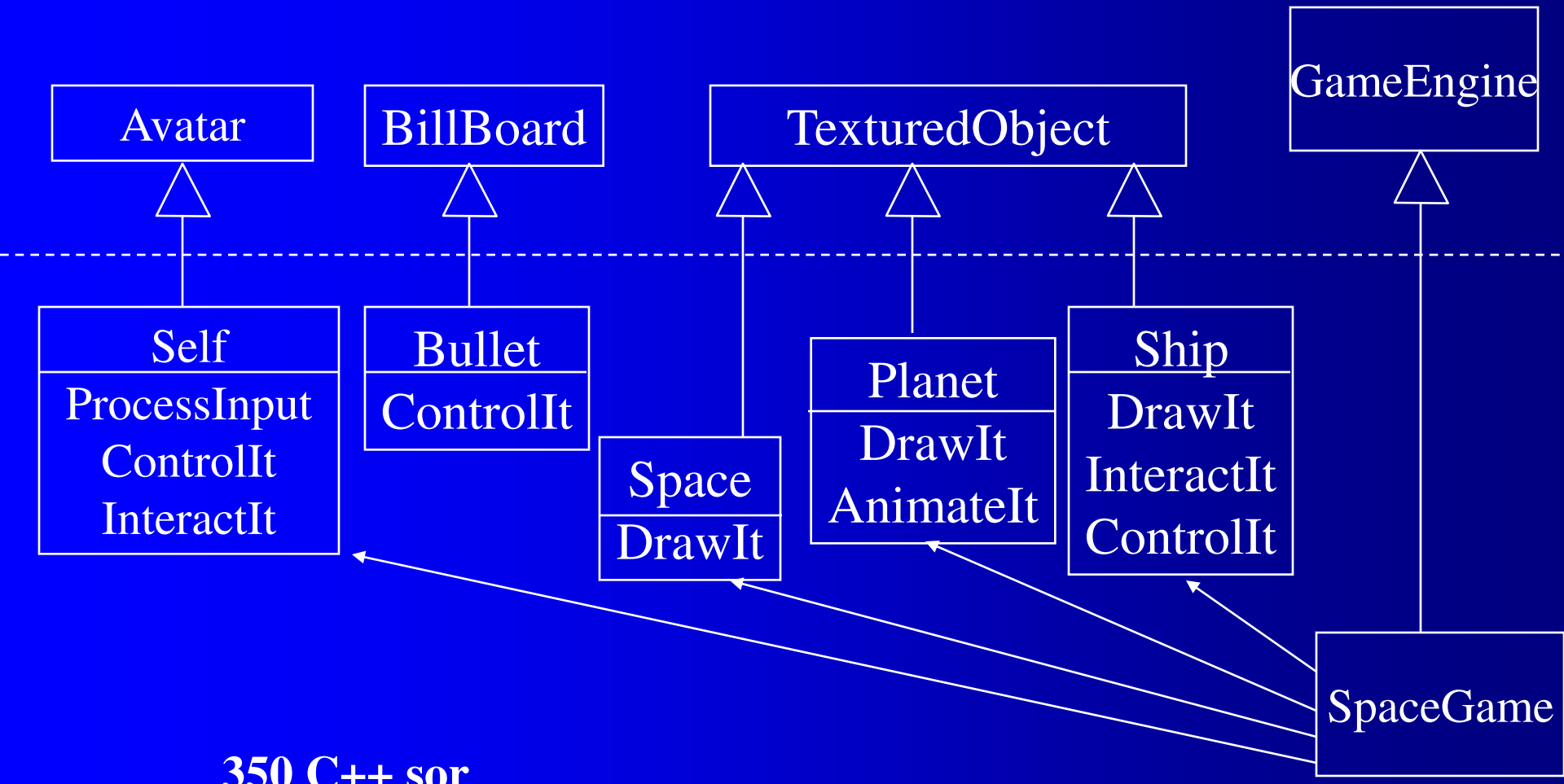
```
Avatar :: SetCameraTransform( ) {
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(position.x, position.y, position.z,
              position.x + head.x,
              position.y + head.y,
              position.z + head.z,
              up.x, up.y, up.z);
}
```



up = [0, 1, 0] vagy a
gyorsulásból és a korábbi up átlagából


GameController
DisplayFunc
IdleFunc
KeyboardFunc
KeyboardUpFunc
GameObject
position, velocity, acceleration
ControlIt(float dt)
AnimateIt(float dt)
InteractIt(GameObject * o)
DrawIt( )
IntersectIt(Ray r, float& t)
Játékmotor
500 C++ sor
world
Member
Control, Animate, Draw
Interact, Intersect, Join
next
Texture
Load(char * fname)
Particle
avatar
Avatar
ProcessInput()
SetCameraTransform()
TexturedObject
BillBoard
DrawIt()
ParticleSystem
Emit(int n)

Űrjáték



350 C++ sor

Egy földi lövöldözős játék

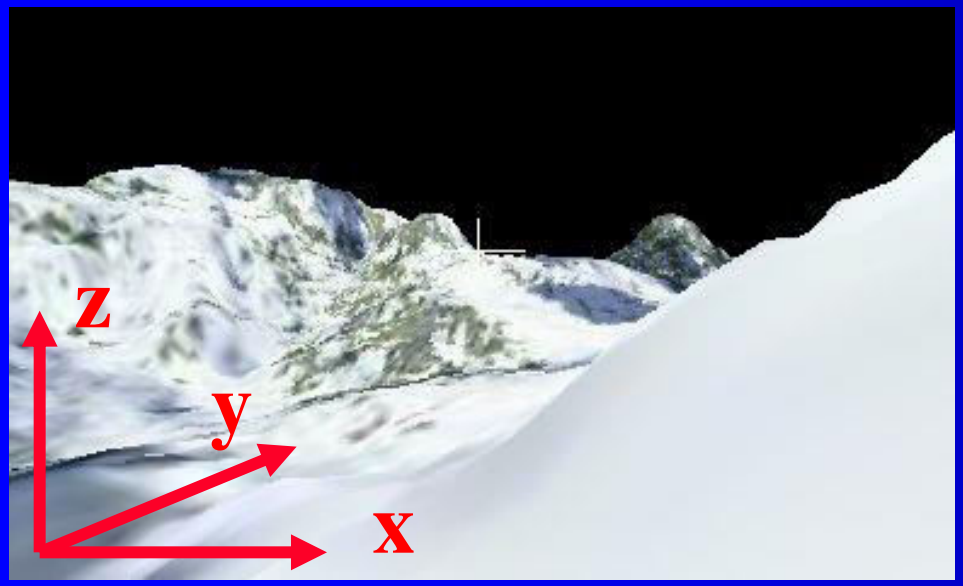


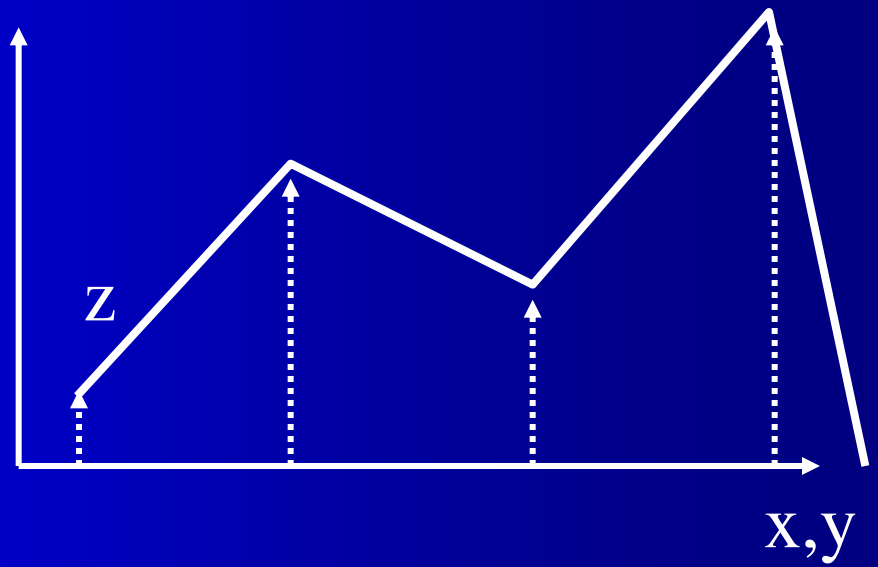
Terepek



- Komplex geometria
  - magasságmező
- Bonyolult textúra
- Nem gondolkodik
- Nem mozog
- Ütközés detektálás kell
- Megemeli az objektumokat

Terep geometria

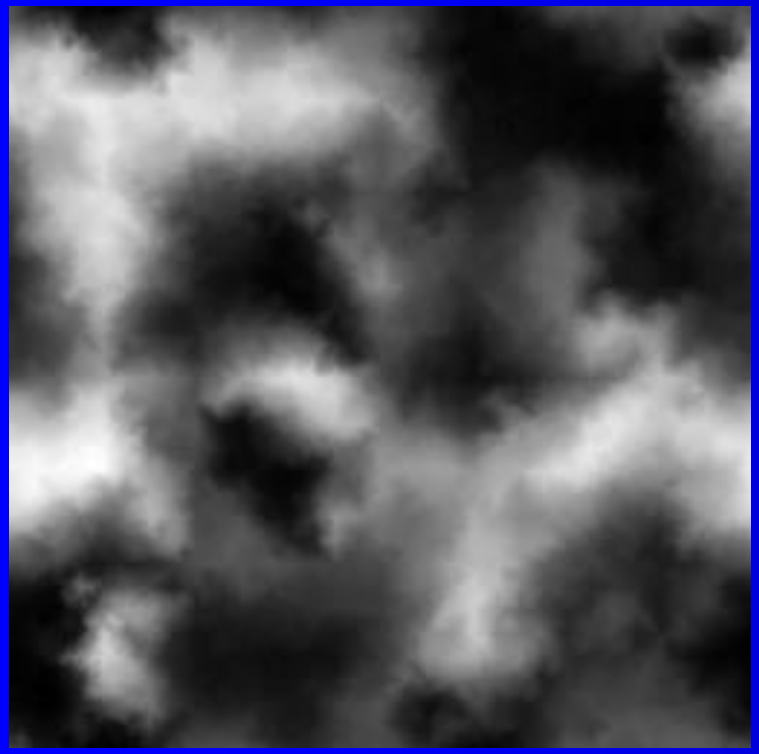


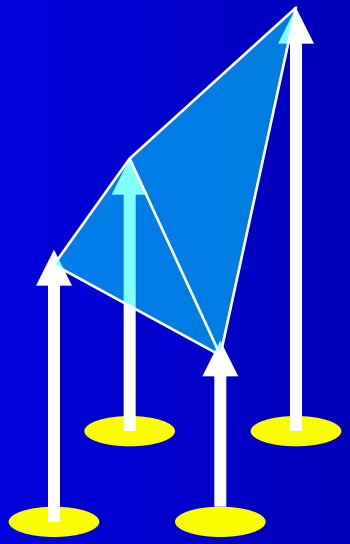


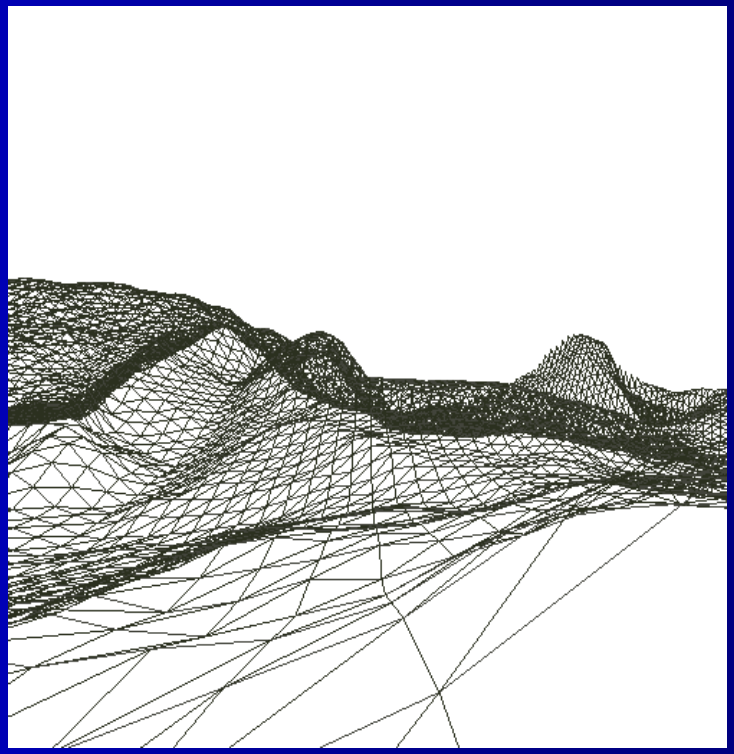
z = height(x,y)

Magasságmező:
Diszkrét minták +
Lineáris interpoláció

Diszkrét minták = FF kép



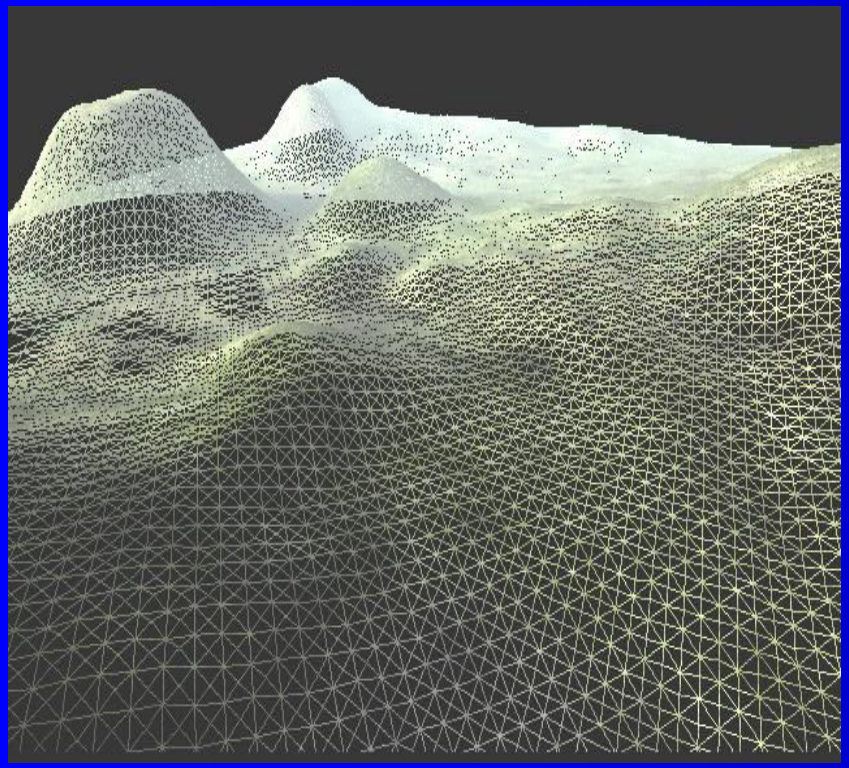


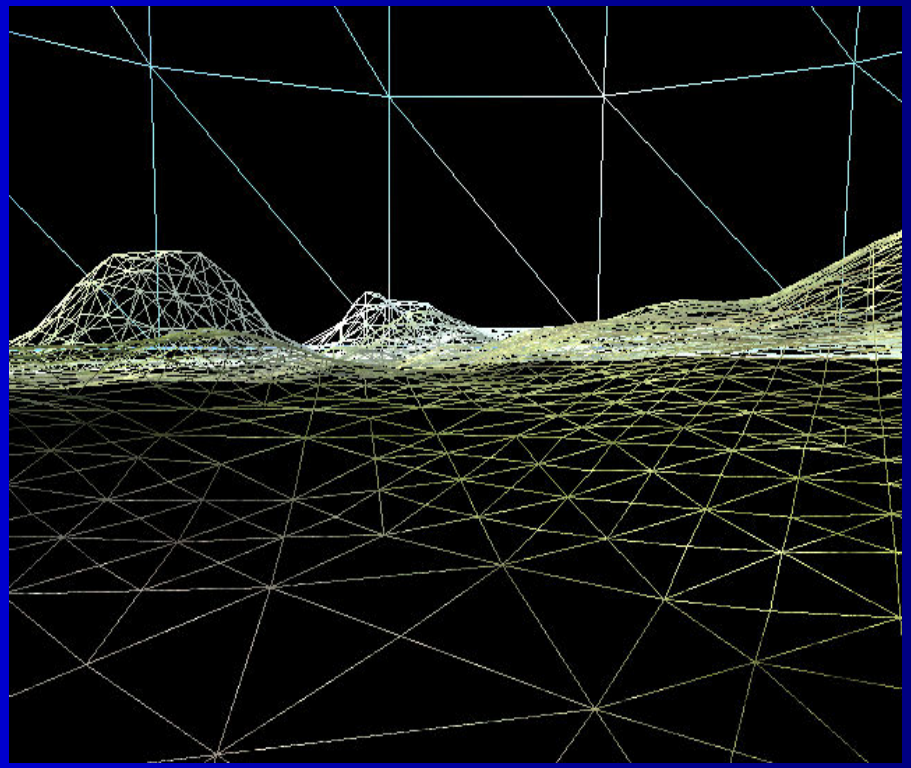


Magasság mező

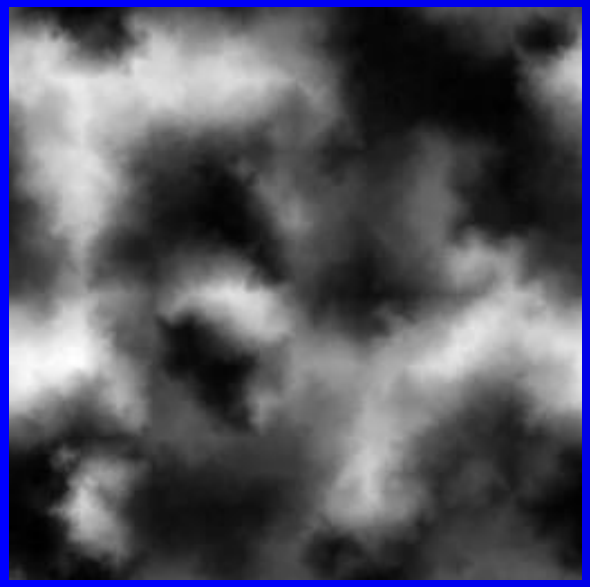
Háromszög háló

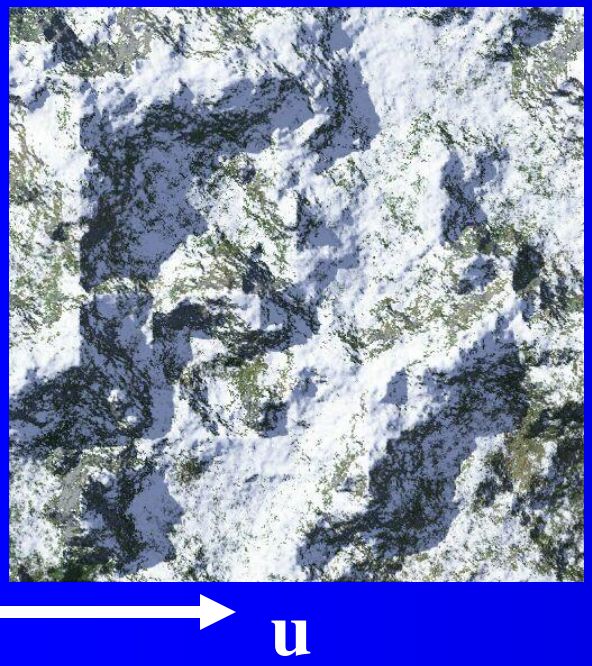
Háromszögek számának csökkentése: Level of detail

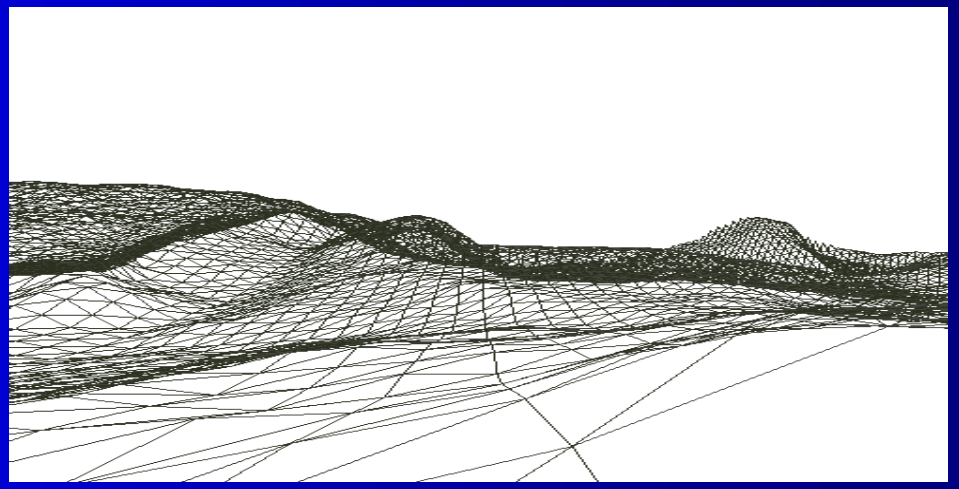




Magasságmező textúrázás: vetítés felülről



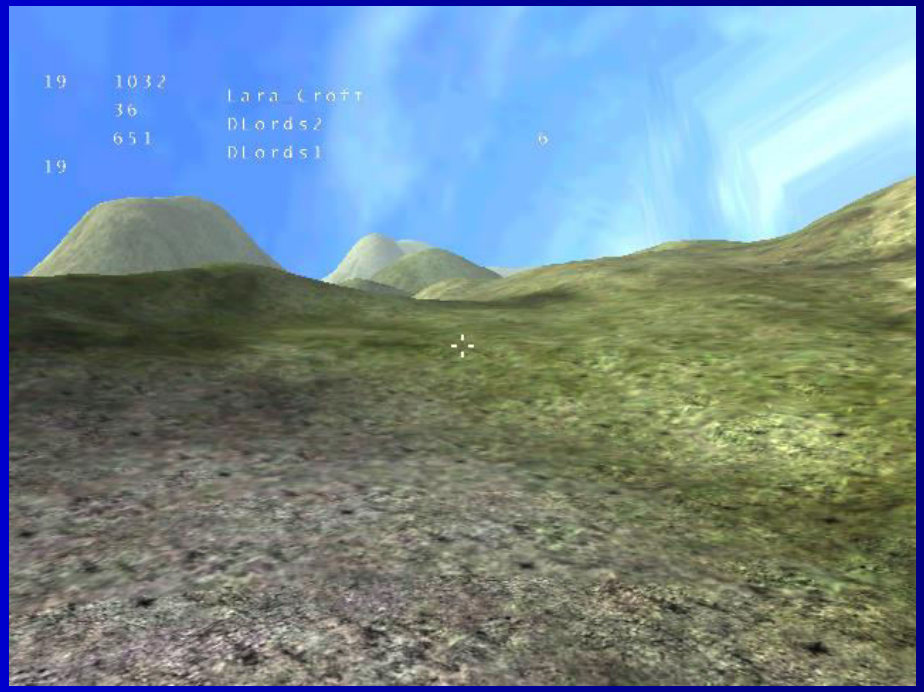






Tereptextúra javítás: Multitextúrázás, Detail map

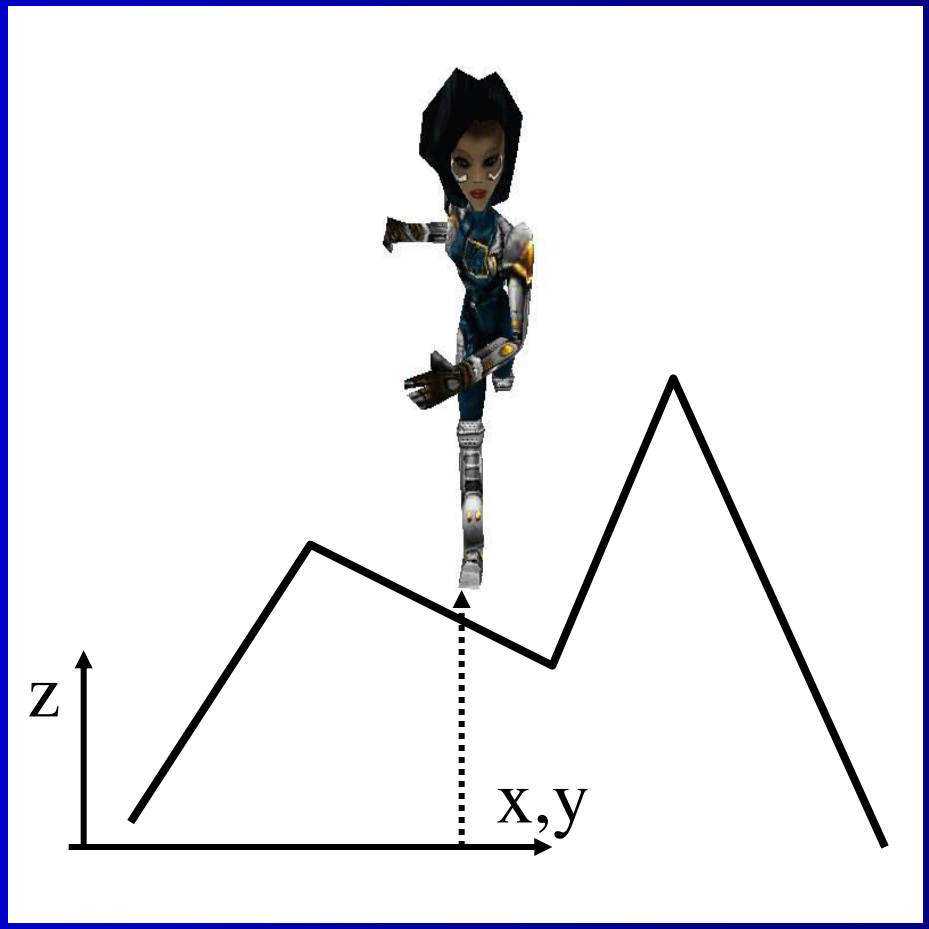




Terep ütközés detektálás

if (height(x,y) > z)

Collision!



Séta a terepen:

Position(x, y) = (x, y, height(x,y) + legsize)

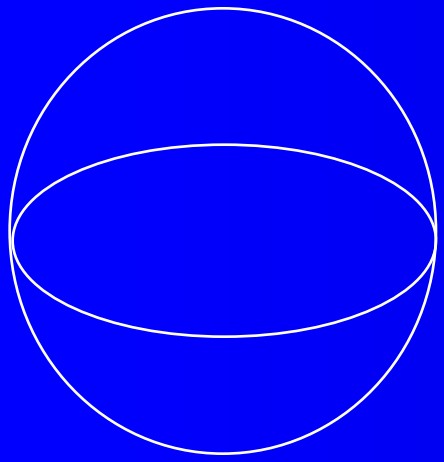
Ég



- Valamire textúrázott kép:
  - gömb
- Geometria: dóm, gömb, téglatest
- Nincs
  - vezérlés,
  - animáció,
  - ütközésdetektálás

GLU kvadratikus felület

```
GLUquadricObj * quadric;                    // definition
quadric = gluNewQuadric( );
gluQuadricTexture(quadric, GL_TRUE);

// draw
glBindTexture(GL_TEXTURE_2D, sky_texture_id);
gluSphere(quadric, sky_radius, 32, 20);
```







Ellenség

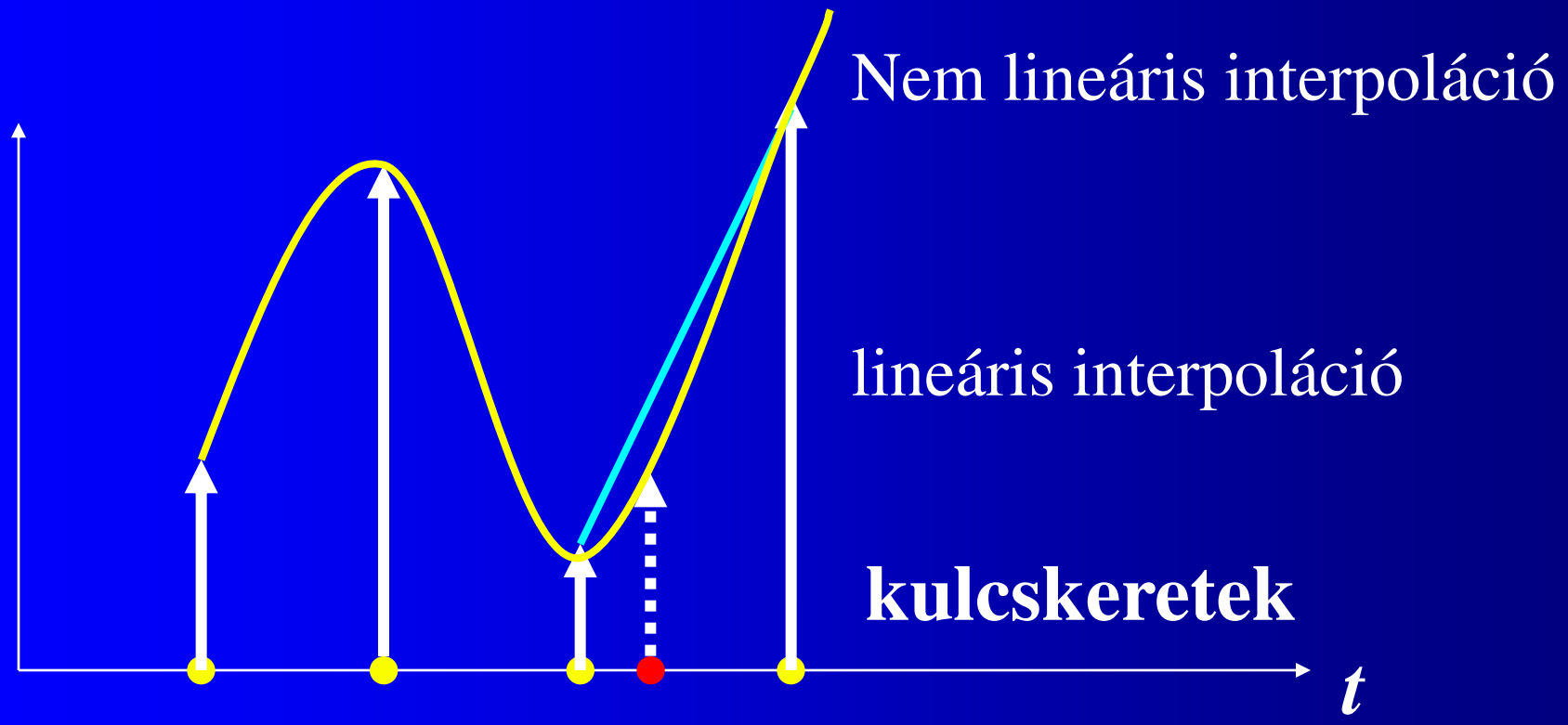
- Animált geometria
  - Kulcskeretekkel mozgásonként (clip)
    - Áll, fut, támad, meghal
  - poligonháló deformáció
- Textúrák (animált)
- AI
- Ütközés detektálás



Kulcskeret animáció: futás

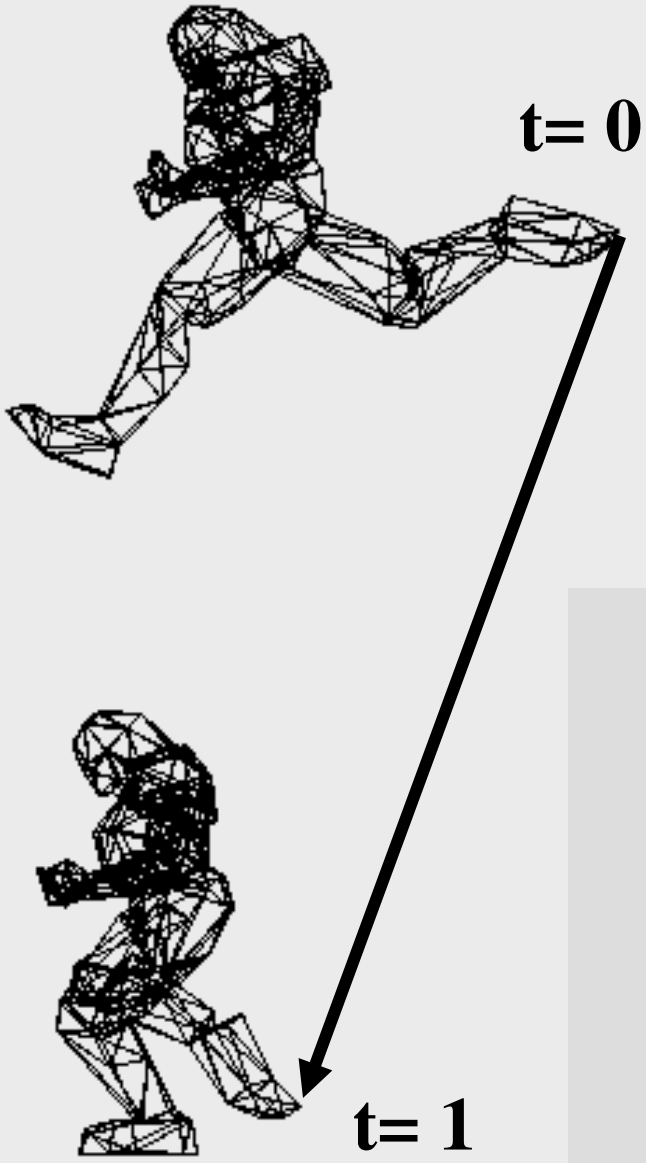


Interpoláció: Keretek a kulcskeretekből

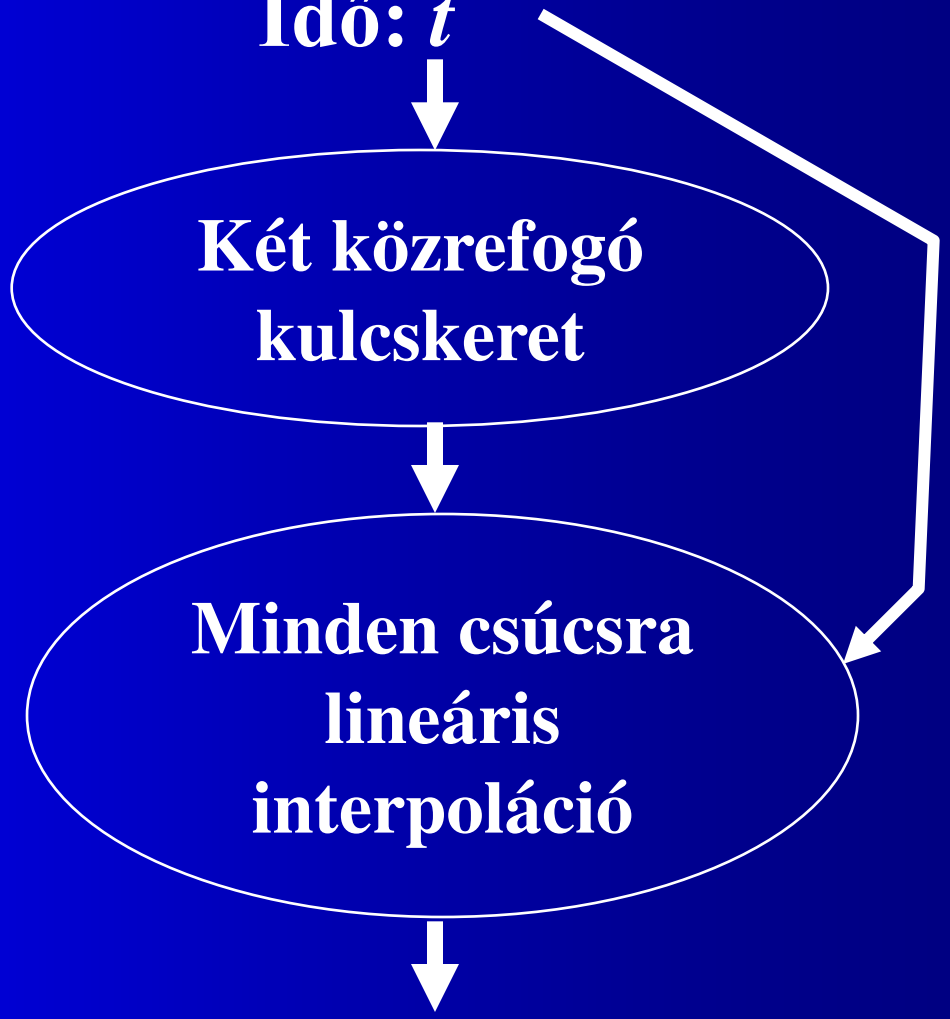


Mit interpoláljunk?

- Minőségi animáció:
  - Newton törvények: C^2 C^1 spline interpoláció
  - Interpolált keretekre is a fiziológiai tv. betartása:
    - **Csont animáció**
- Régi játékok:
  - Lineáris interpoláció
  - A poligonháló csúcspontjait interpoláljuk
    - **Poligonháló deformáció (Mesh morphing)**



Mesh morphing:



Futás poligonháló deformációval



+ pozíció animáció:

position += velocity * dt



Mozgás definíció

- Clip-ek definíciója kulcskeretekkel
- Összes clip összes kulcskeretek fájlban: MD2, MD3
- Tipikus clip-ek:
  - Run, stand, attack, die, pain, salute, crouch, wave, point, taunt, etc.

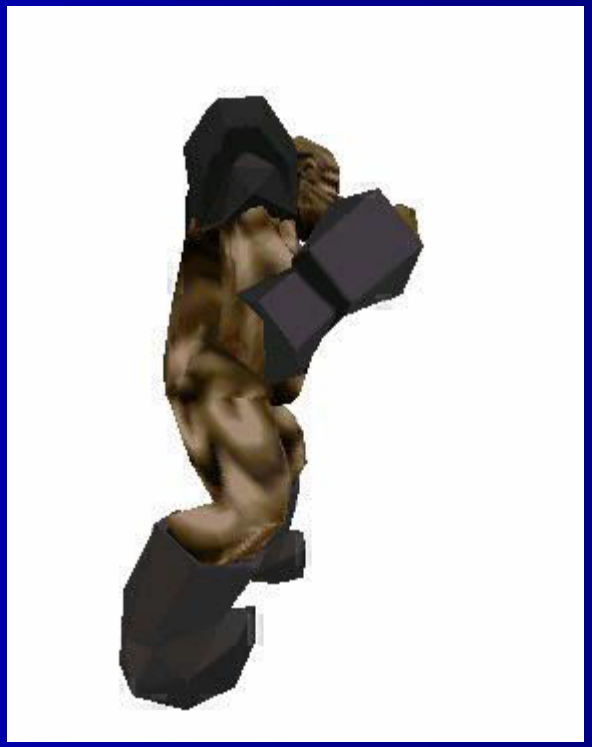
Clip-ek



All
40 kulcskeret

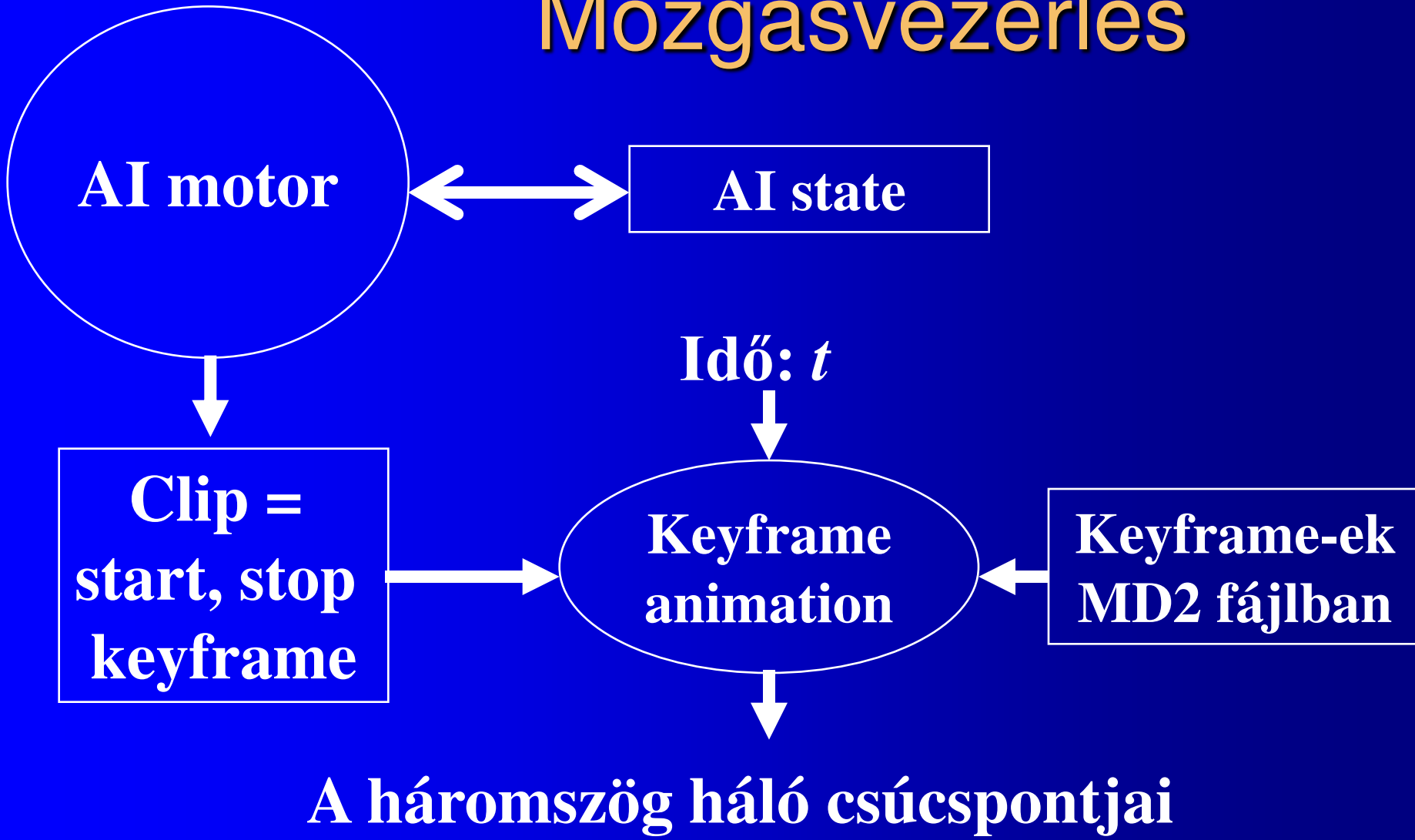


Fut
5 kulcskeret

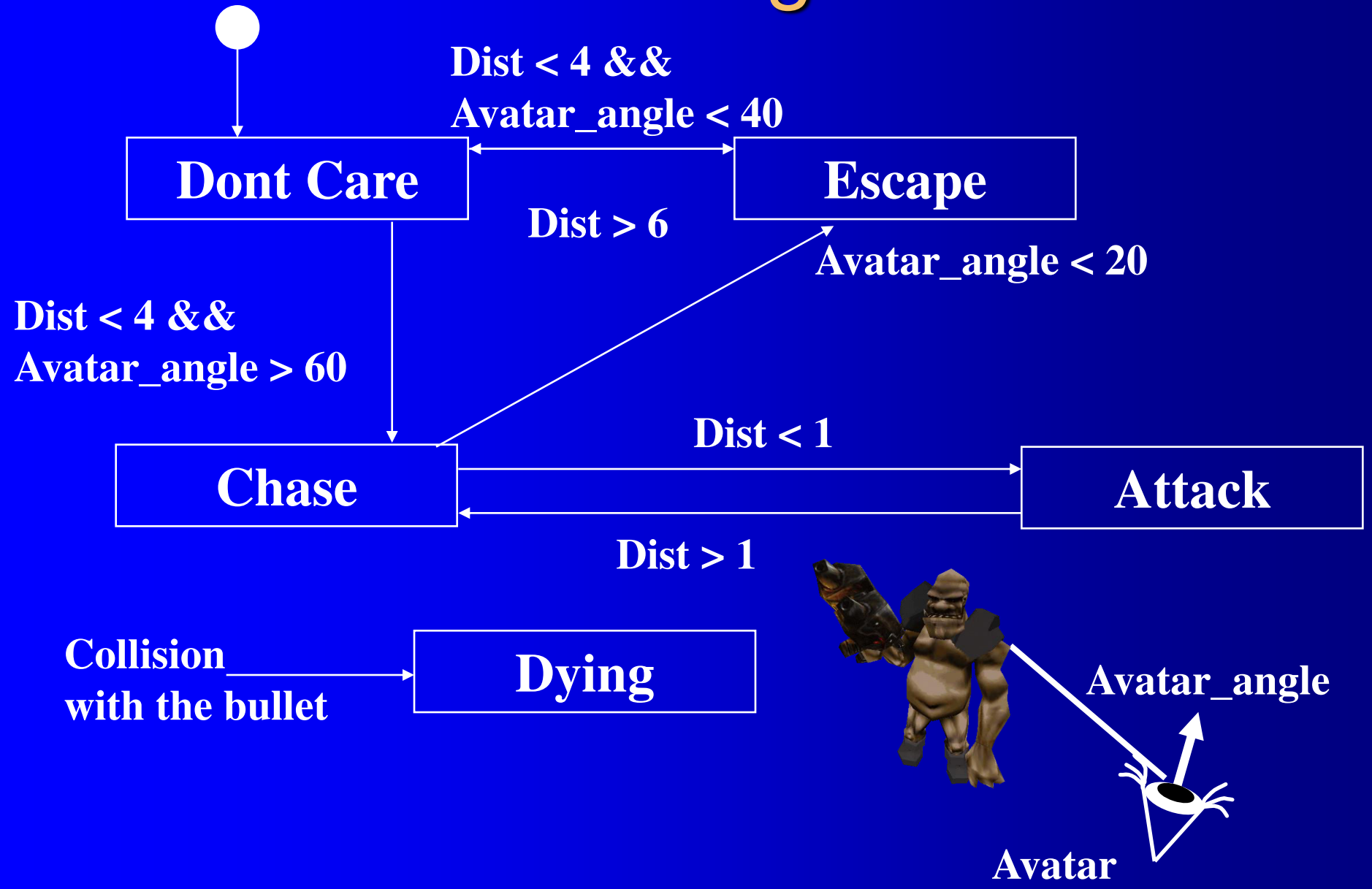


Szalutál
11 kulcskeret

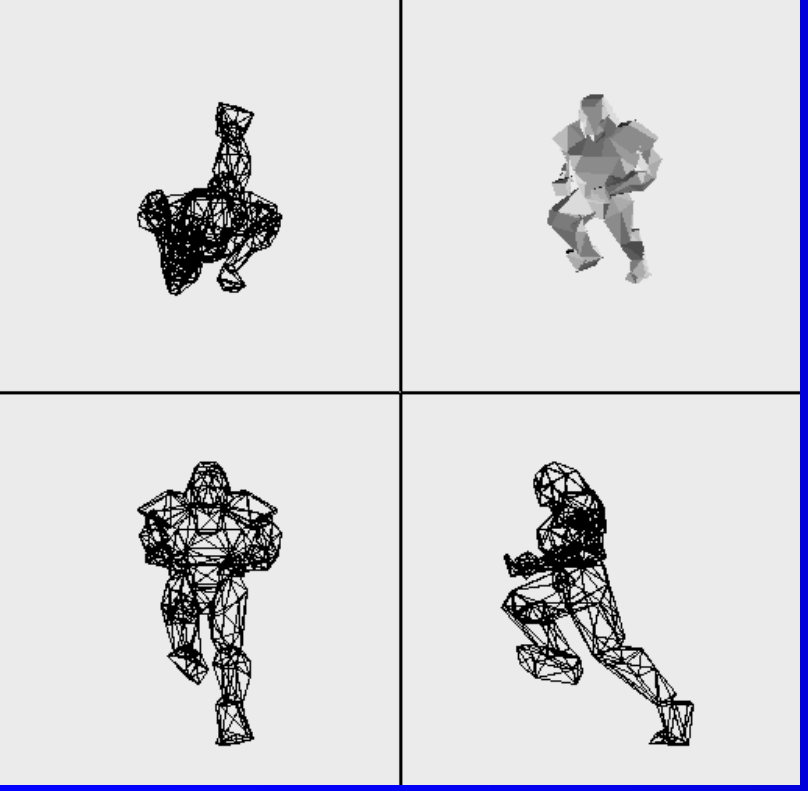
Mozgásvezérlés

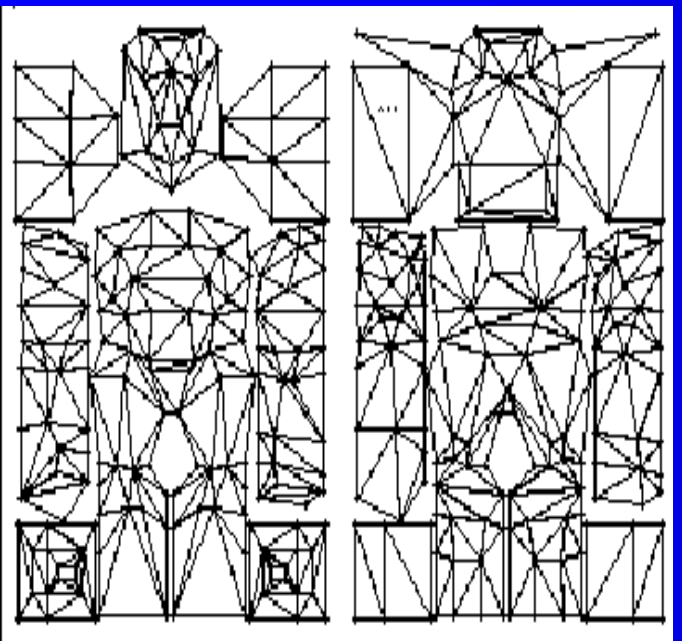


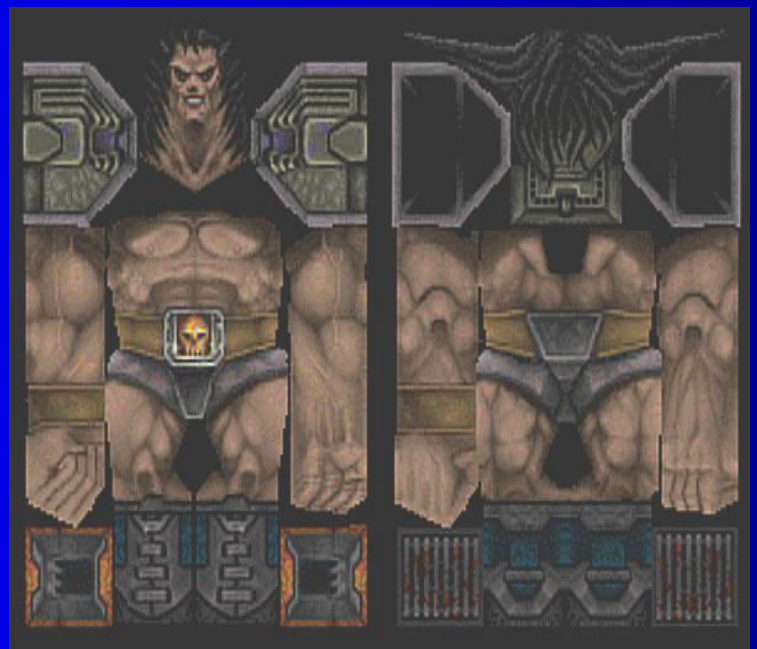
Ellenség AI


Dist < 4 &&
Avatar_angle < 40
Dont Care
Escape
Dist > 6
Avatar_angle < 20
Dist < 4 &&
Avatar_angle > 60
Dist < 1
Chase
Attack
Dist > 1
Collision
with the bullet
Dying
Avatar_angle
Avatar

Textúrázás





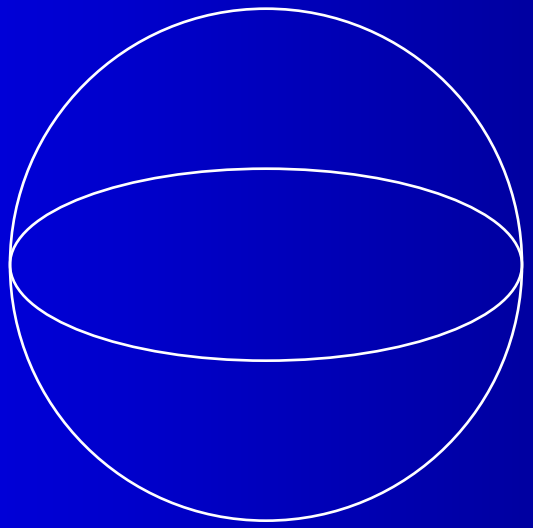




Golyó

- Geometria: gömb
- Textúrázott
- Nem intelligens
- Fizikai animáció

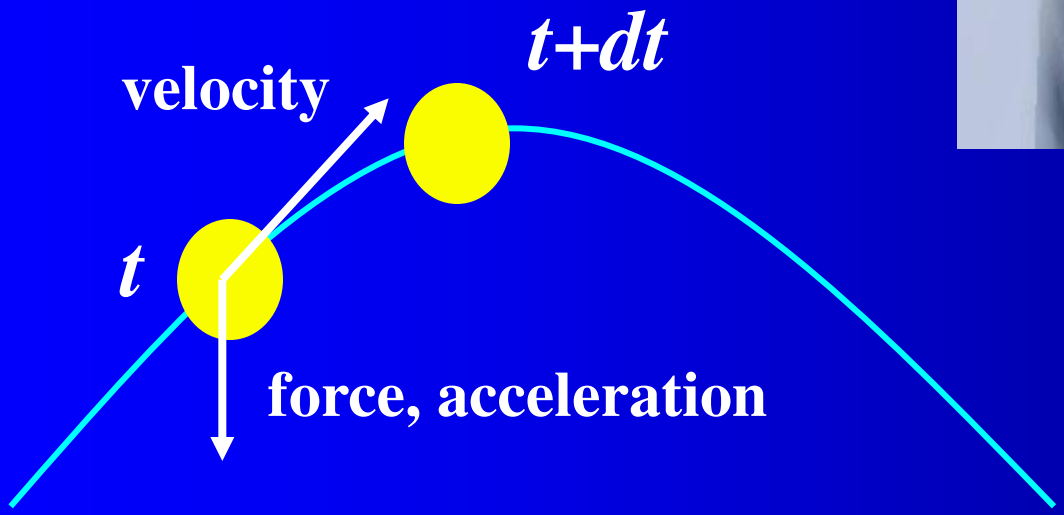






Lövedék fizikai animációja





acceleration = (0, 0, -g)
velocity += acceleration * dt
position += velocity * dt

Röpül a lövedék: 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( );
}
```