

*"In jeder besonderen Naturlehre nur so viel  
eigentliche Wissenschaft angetroffen werden  
köinne, als darin Mathematik anzutreffen ist"*

*Immanuel Kant*

# Geometriák és algebrák

## 1. Klasszikus geometriák

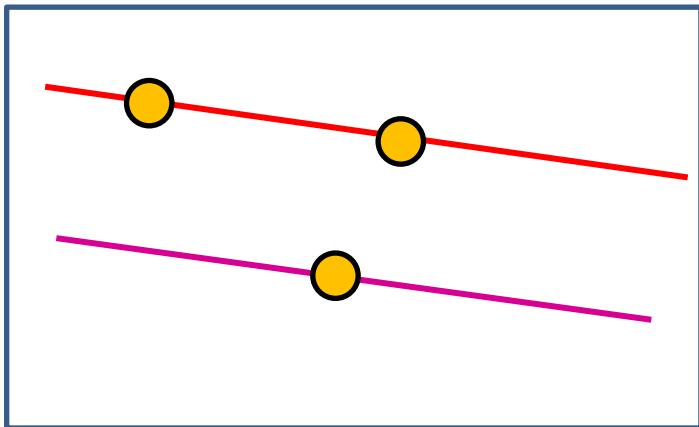
Szirmay-Kalos László



# Euklidészi síkgeometria

## Axiómák

- Alapigazság (tapasztalat + hit)
- Alapfogalmak implicit definíciója
- Tételek kiindulási pontja



### Euklideszi síkgeometria axiómái:

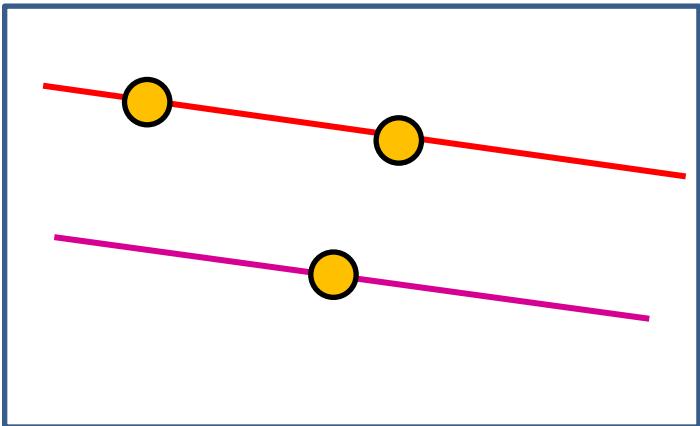
1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. Egy egyeneshez egy rajta kívül fekvő ponton át egy nem metsző egyenes húzható (párhuzamosság).
4. Átmozgatható (egybevágó) dolgok mérete megegyező (a sík homogén és izotróp).
5. A részek összege az egész mérete.
6. ...



# Euklidészi síkgeometria

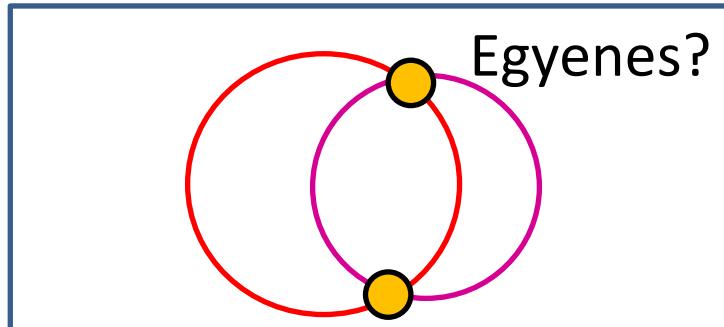
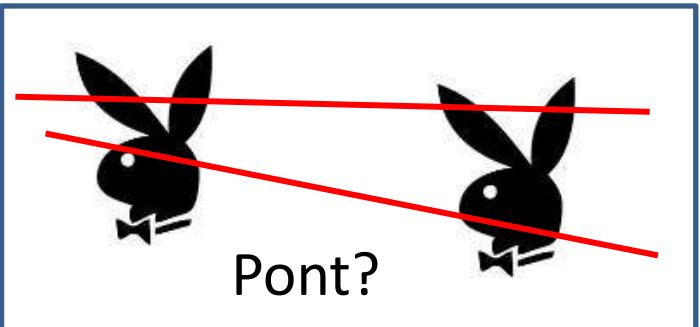
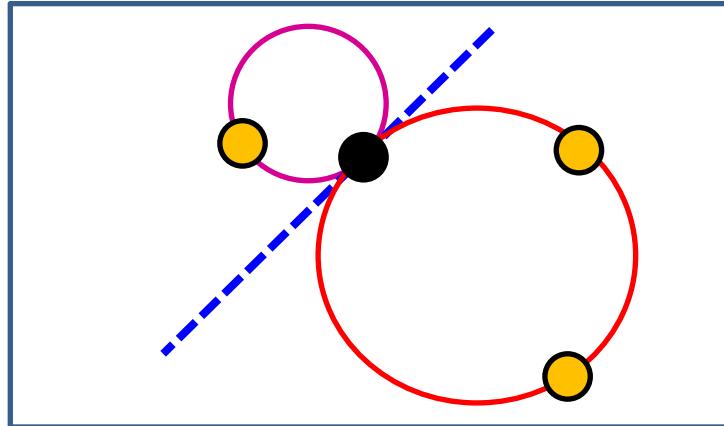
## Axiómák

- Alapigazság (tapasztalat + hit)
- Alapfogalmak implicit definíciója
- Tételek kiindulási pontja



Euklideszi síkgeometria axiómái:

1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. Egy egyeneshez egy rajta kívül fekvő ponton át egy nem metsző egyenes húzható (párhuzamosság).



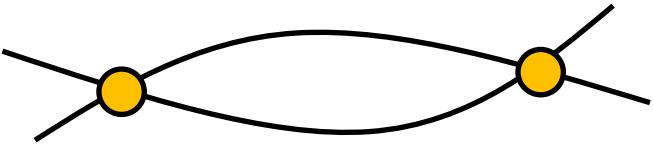


# Euklidészi síkgeometria

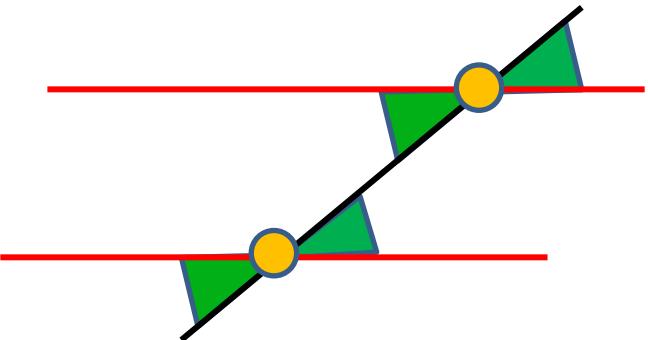
## Axiómák

- Alapigazság (tapasztalat + hit)
- Alapfogalmak implicit definíciója
- Tételek kiindulási pontja

T1. Két különböző egyenes legfeljebb egy pontban metszi egymást



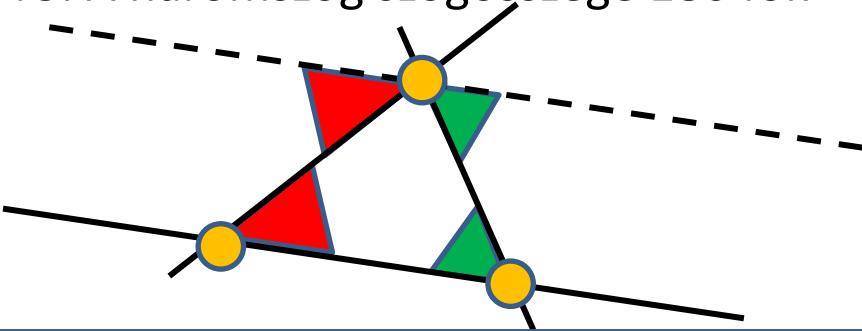
T2. Ha egy egyenes két másikat ugyanolyan szögben metsz  $\Leftrightarrow$  két másik egyenes párhuzamos.



Euklideszi síkgeometria axiómái:

1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. Egy egyeneshez egy rajta kívül fekvő ponton át egy nem metsző egyenes húzható (párhuzamosság).
4. Átmozgatható (egybevágó) dolgok mérete megegyező.
5. A részek összege az egész mérete.
6. ...

T3. A háromszög szögösszege 180 fok

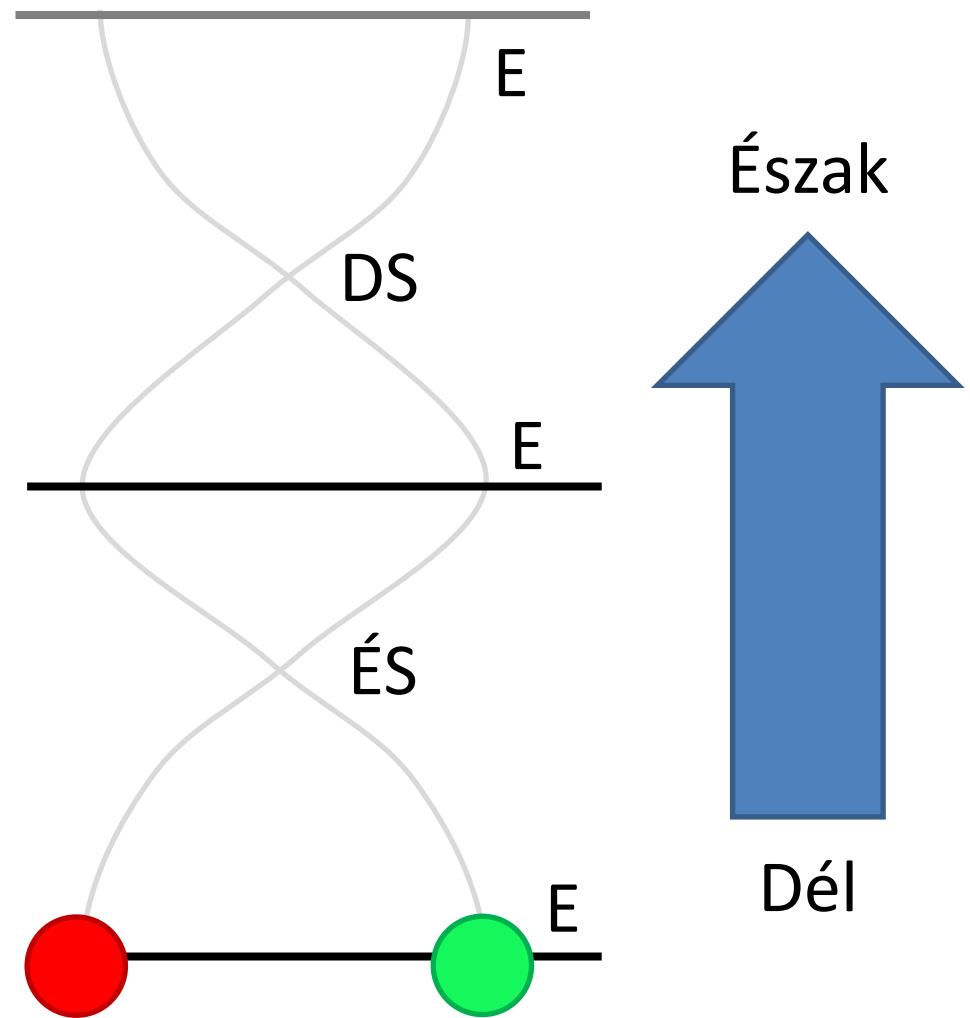
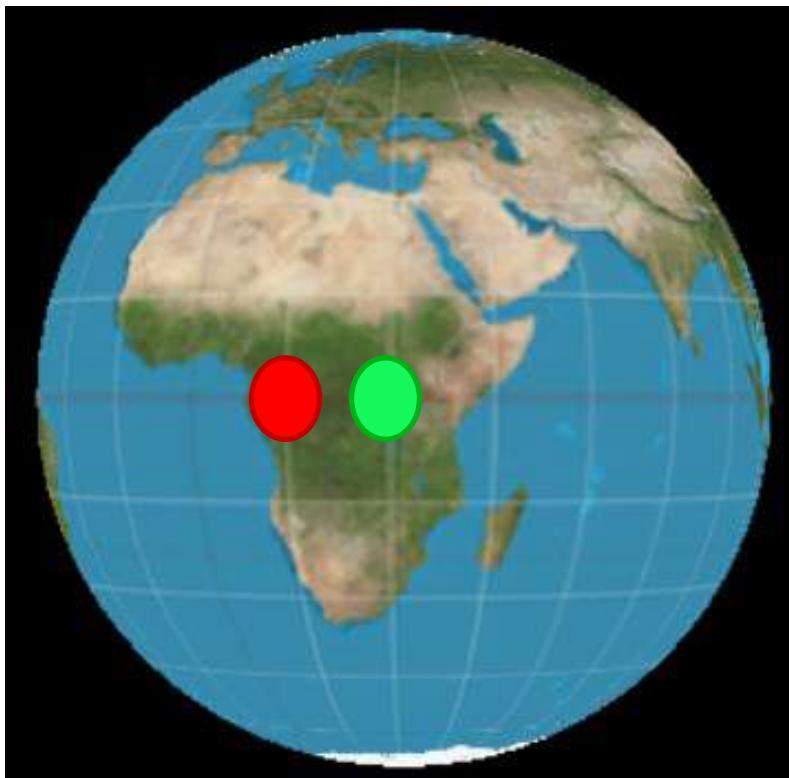


T4. Pitagorasz:  $a^2+b^2 = c^2$

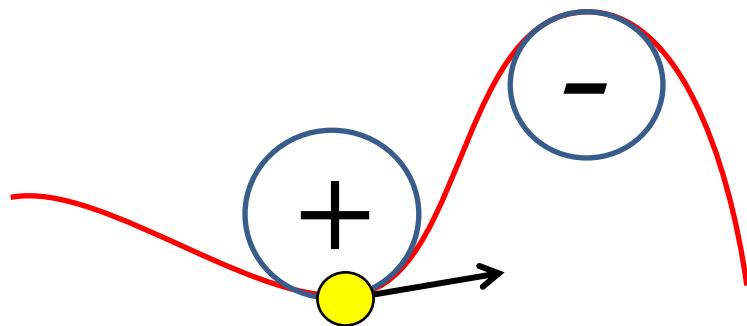
T5. Kör területe:  $\pi R^2$

• • •

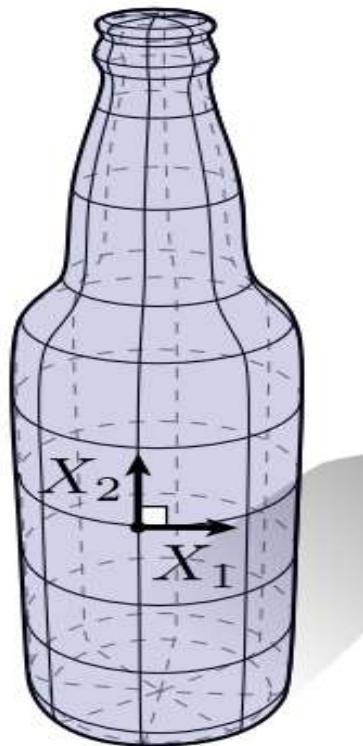
# Jó az euklideszi geometria föld (Geo) mérésre (meter)?



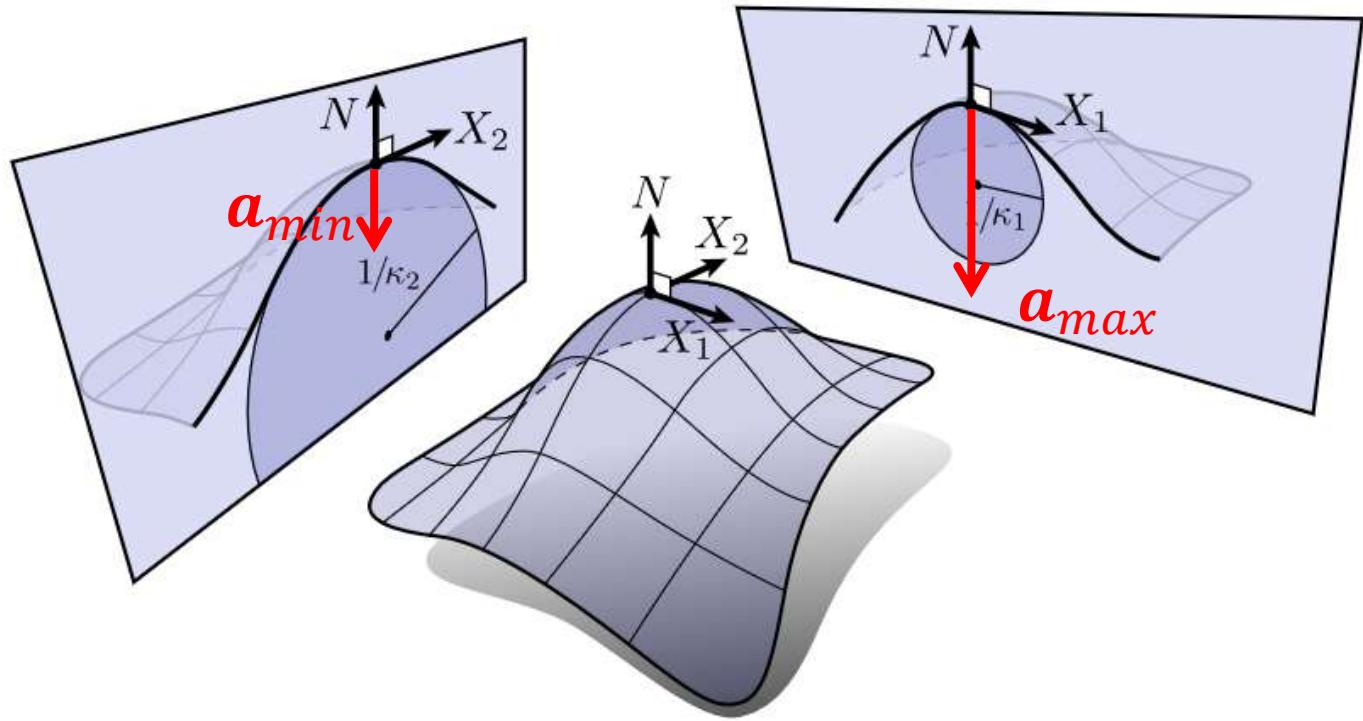
# Görbék görbülete: $\kappa$



- Egységsebességű mozgás centripetális gyorsulás:  
$$a_{cp} = \frac{v^2}{R}$$
- (Másodrendben) simulókör sugarának reciproka

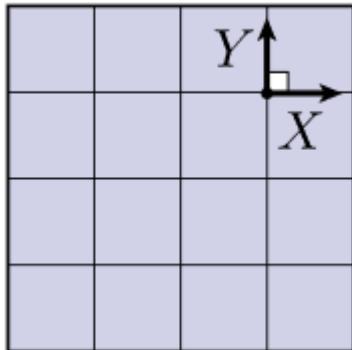


# Felületek: fő görbületi irányok és Gauss görbület (Krümmung)



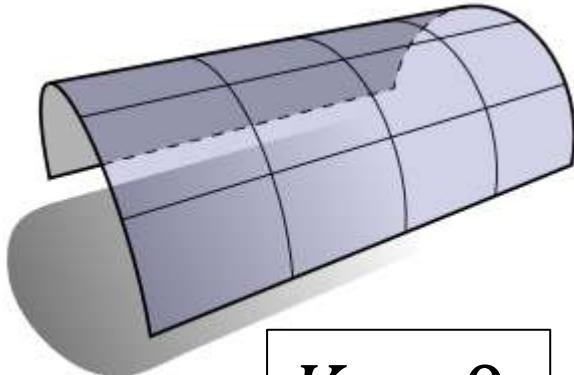
$$K = \kappa_1 \kappa_2 = a_{min} \cdot a_{max}$$

**Theorema Egregium:**  $K$  attól függ, hogy a felületen hogyan mérünk szöget és távolságot, és független a felület térbeli alakjától.

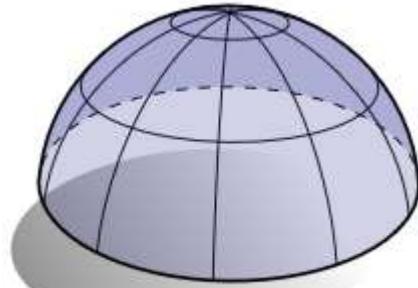


# Gauss görbület:

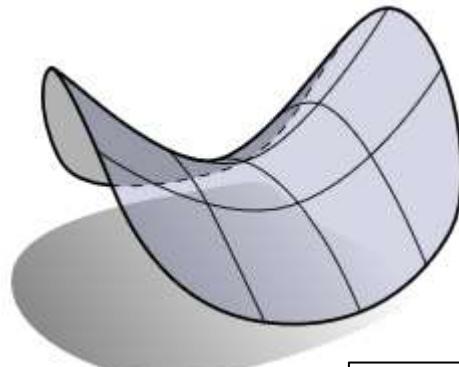
$$K = \kappa_1 \kappa_2$$



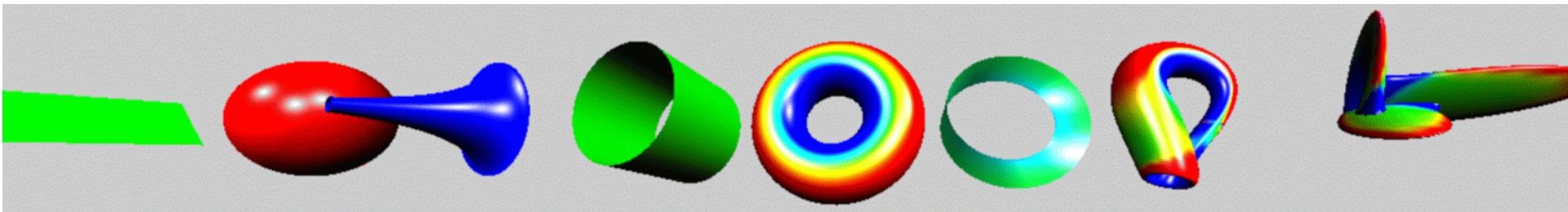
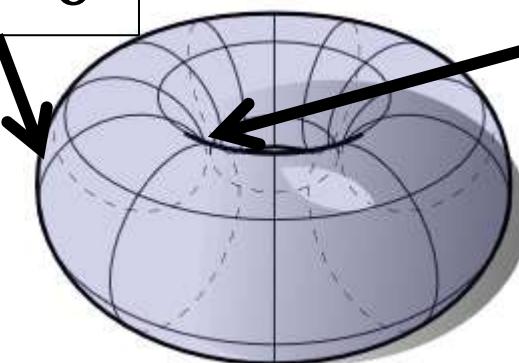
$$K = 0$$



$$K > 0$$



$$K < 0$$



Square

Sphere

Tractricoid

Cylinder

Torus

Möbius

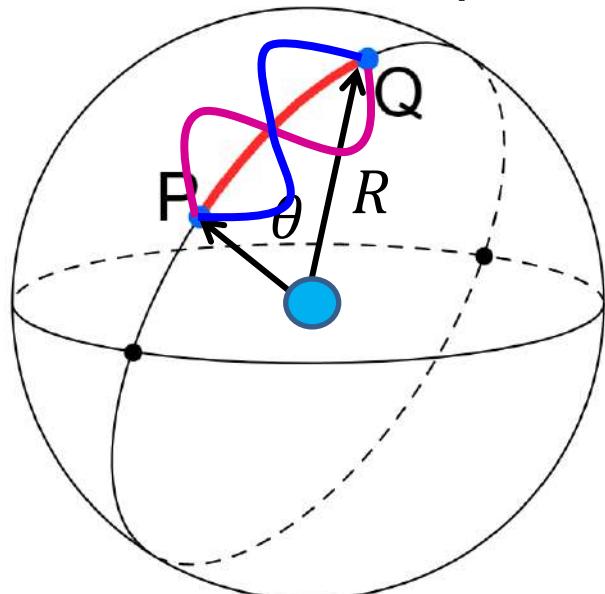
Klein

Boy

# Gömbi geometria

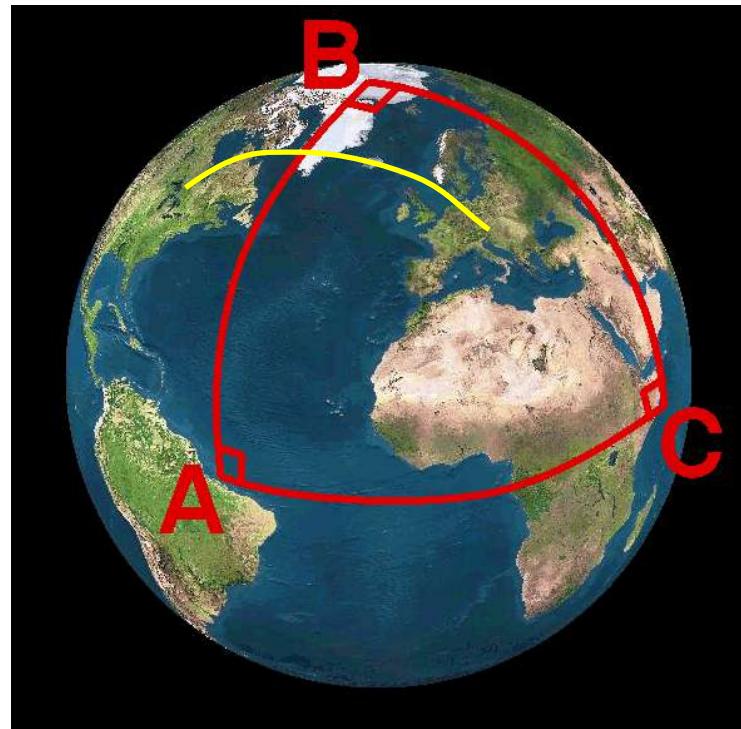
$$x^2 + y^2 + z^2 = R^2$$

- Állandó pozitív görbület
- Mi az egyenes?  
**főkör** = legrövidebb út  
(geodézikus vonal)
- Távolság =  $R\theta = \theta/\sqrt{K}$



Euklidészi geometria axiómái nem jók:

1. Két pont **nem mindig** határozza meg egyértelműen egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. **Két egyenes mindenkor metszi egymást két pontban.**
4. Átmozgatható (egybevágó) dolgok mérete megegyező
5. A részek összege az egész mérete
6. ...

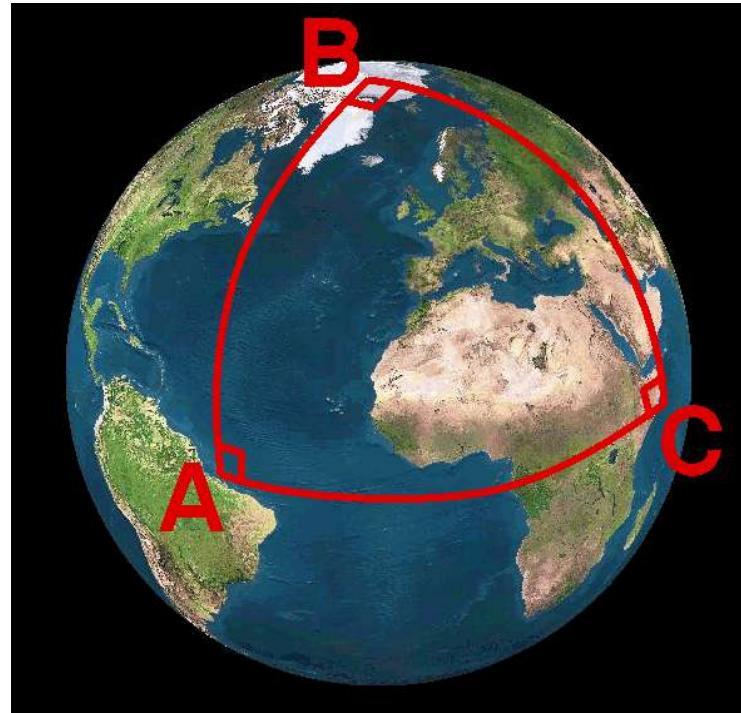


# Elliptikus geometria

- Állandó pozitív görbület
- Egyenes=főkör=legrövidebb út
- Háromszög szögösszeg  $> 180^\circ$  (terüettel arányosan)
- Háromszög területe:  
$$(\alpha + \beta + \gamma - \pi)/K$$
- Hasonlóság = Egybevágóság
- Derékszögű háromszög:  
$$a^2 + b^2 > c^2$$
- Kör kerülete:  $\frac{2\pi}{\sqrt{K}} \sin(r\sqrt{K})$
- Kör területe:  $\frac{4\pi}{K} \sin^2\left(\frac{r\sqrt{K}}{2}\right)$

Csak a párhuzamossági axióma nem jó:

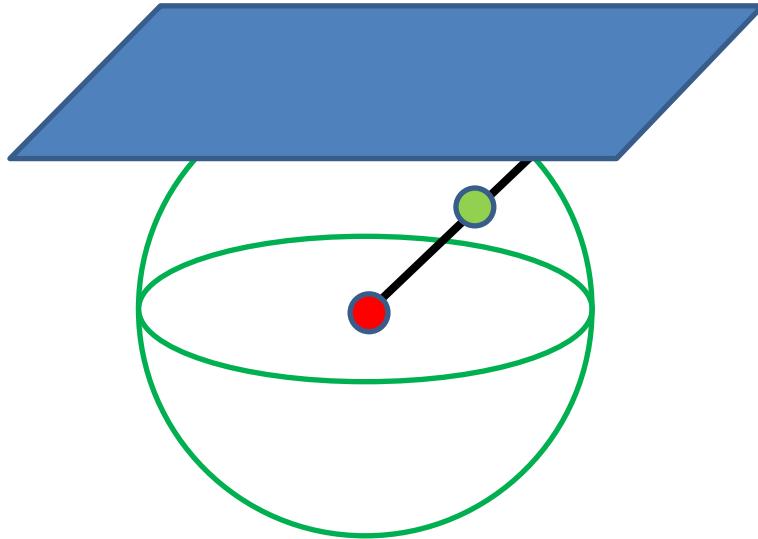
1. Két pont (=átmérő) meghatároz egy egyenest (=geodétilkus, főkör).
2. Egy egyenesnek van legalább két pontja.
3. Két egyenes mindenkorban metszi egymást.
4. Átmozgatható (egybevágó) dolgok mérete megegyező
5. A részek összege az egész mérete
6. ...



# Egy jó térkép ...

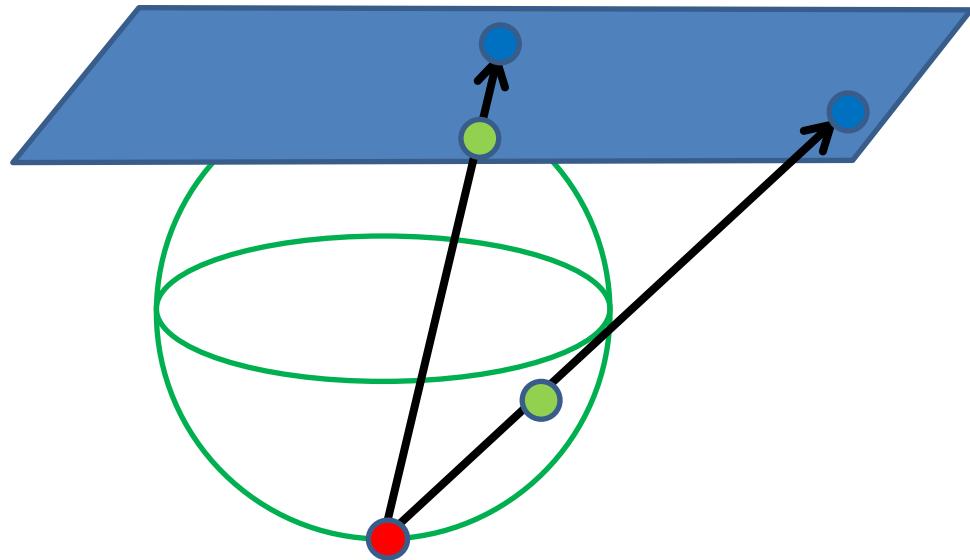
- Euklideszi
  - Valódi helyek és utak
    - Egy-egy értelmű folytonos leképzés
  - Orientáció (jobbra/balra) valódi
- Topológiai hasonlóság
- 
- Valódi egyenes/kör annak látszik
  - Szögtartó
  - Távolságárány tartó
  - Területarány tartó
- Gauss görbület megegyezik
- Geometriai hasonlóság

# Gömb vetítése a síkra



## Középpontos vetítés

- Csak a felső félgömböt
- Egyenestartás: tárgy és vetítőszugarak egy síkban, vetület ennek és a képsíknak a metszete
- Nem kör és szögtartó
- Nem távolságtartó
- Beltrami-Klein



## Sztereografikus vetítés

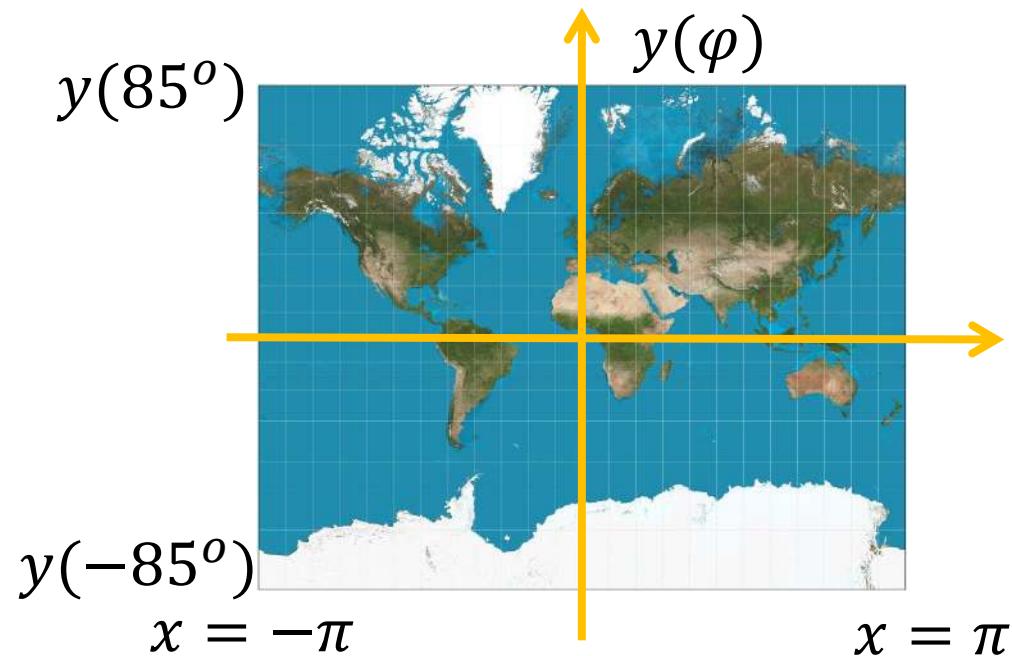
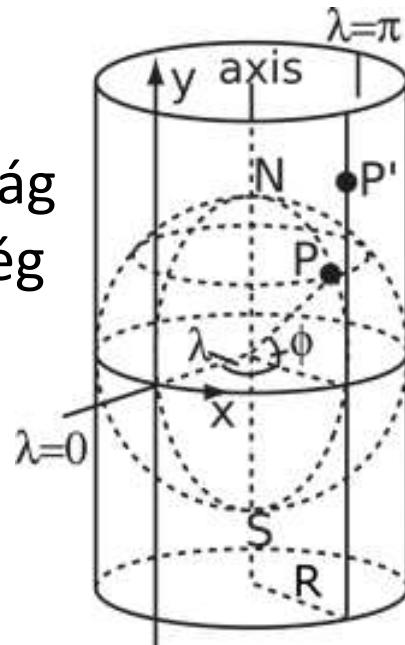
- A déli pólus kivételével egészet
- Nem egyenestartó
- Körtartó és szögtartó
- Nem távolságtartó
- Beltrami-Poincaré



# (Gerardus) Mercator térkép

- Szögtartó, de torzítja a távolságot és területet

$\lambda$  = hosszúság  
 $\phi$  = szélesség



$$x = \lambda$$

$$y = \log \left( \tan \left( \frac{\pi}{4} + \frac{\varphi}{2} \right) \right)$$



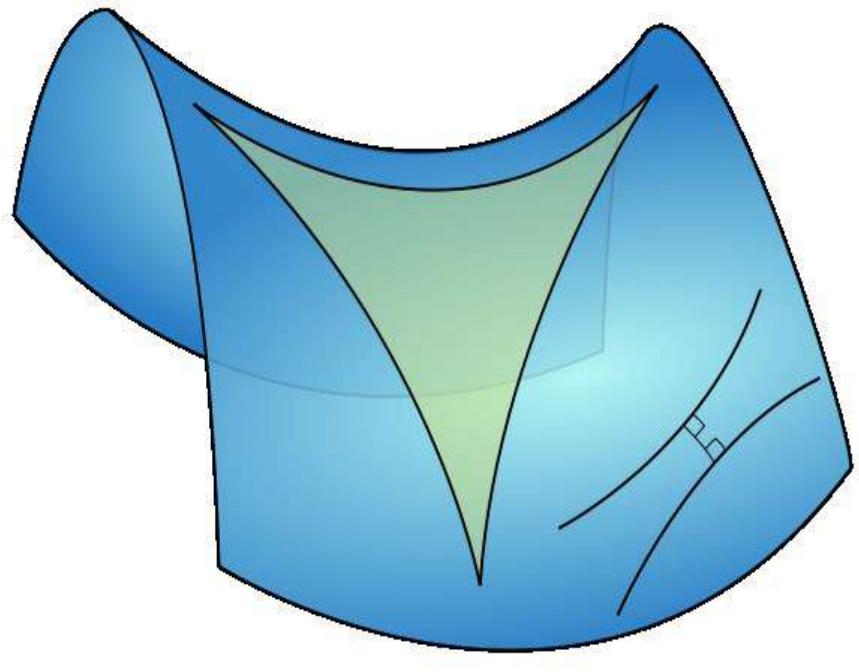


# Hiperbolikus (Bolyai) geometria

- Állandó negatív görbület
- Egyenes = legrövidebb út
- Háromszög szögösszeg  $< 180^\circ$
- Háromszög területe:  
$$(\alpha + \beta + \gamma - \pi)/K$$
- Hasonlóság = Egybevágóság
- Derékszögű háromszög:  
$$a^2 + b^2 < c^2$$
- Kör kerülete:  
$$\frac{2\pi}{\sqrt{-K}} \sinh(r\sqrt{-K}) = \frac{2\pi}{\sqrt{K}} \sin(r\sqrt{K})$$
- Kör területe:  $\frac{4\pi}{-K} \sinh^2\left(\frac{r\sqrt{-K}}{2}\right)$

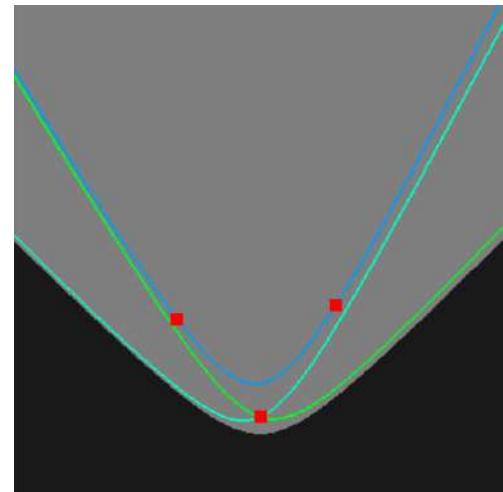
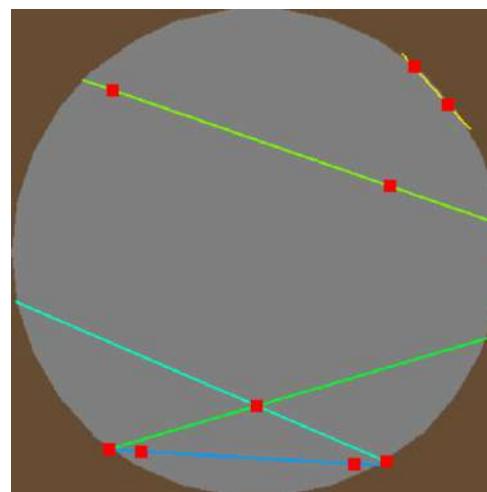
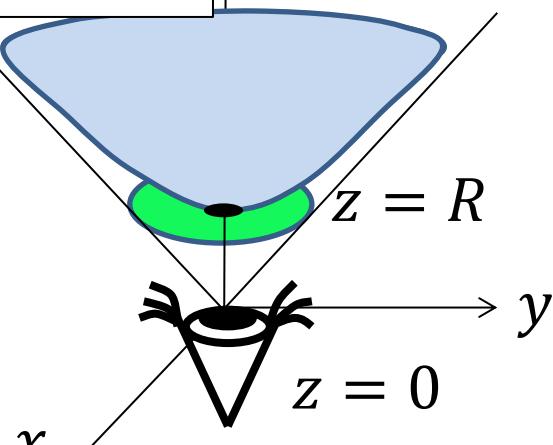
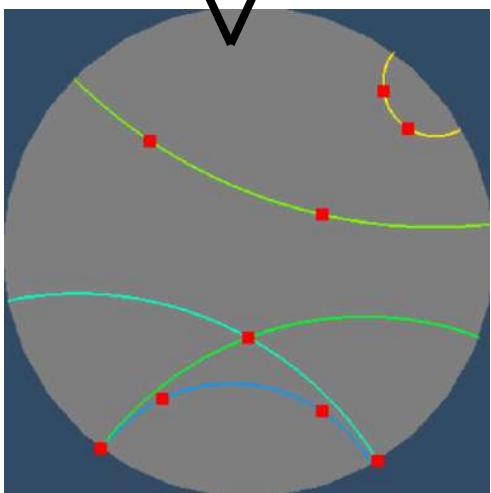
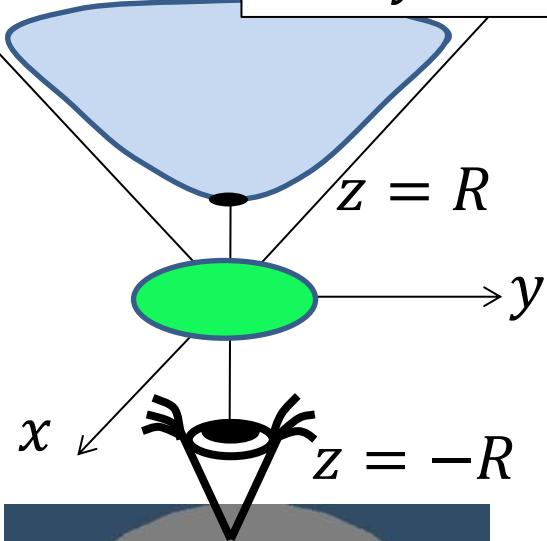
Hiperbolikus geometria axiómái:

1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. Egy egyeneshez egy rajta kívül fekvő ponton át több nem metsző egyenes húzható.
4. Átmozgatható (egybevágó) dolgok mérete megegyező
5. A részek összege az egész mérete
6. ...



# Beltrami – Poincaré/Klein diszk

$$x^2 + y^2 - z^2 = -R^2$$



Klein: Egyenestartó

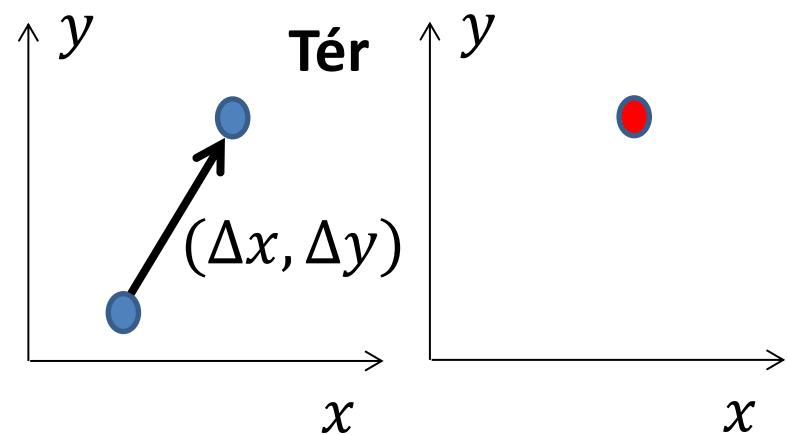
Minkowski tér  
Ez egy gömb!



Poincaré: Szög és körtartó,  
egyenesből alapkörre  
merőleges körív

# Metrika (távolság, szög): skaláris szorzás

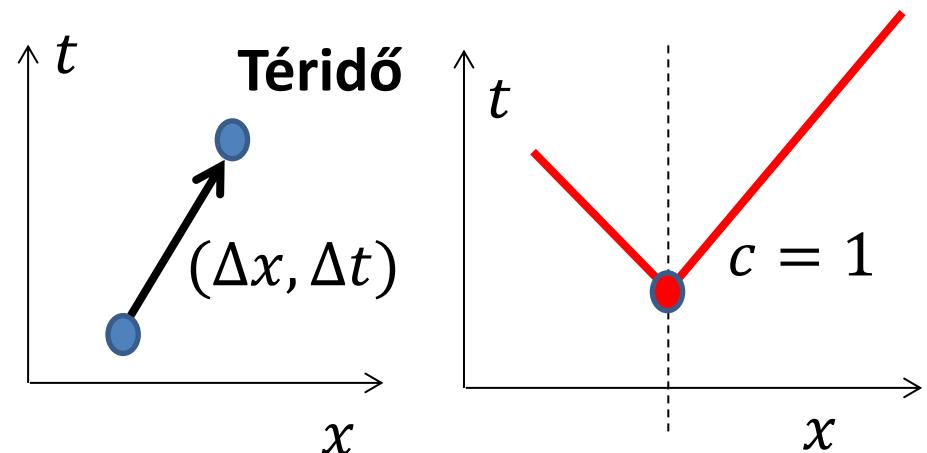
- Mozgatás (egybevágóság) nem változtatja meg
- Elmozdult megfigyelők ugyannak látják
- Két „pont” megegyezik, ha a közöttük lévő vektor önmagával vett skaláris szorzata zérus.



Két pont nem különbözik:  $|\Delta x| = |\Delta y| = 0$

Euklideszi különbözőség  $\boxed{\Delta s^2 = \Delta x^2 + \Delta y^2}$

Skalár szorzat:  $x_1 x_2 + y_1 y_2$

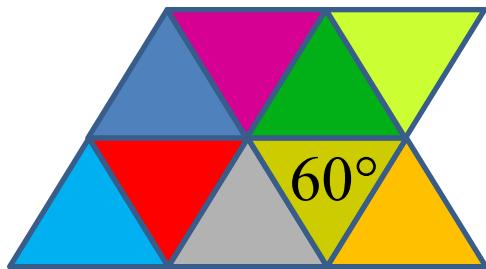


Két esemény nem különbözik:  $|\Delta x| = c|\Delta t|$

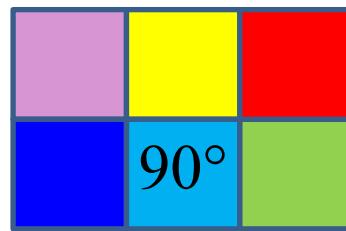
Minkowski különbözőség:  $\boxed{\Delta s^2 = \Delta x^2 - \Delta t^2}$

Skalár szorzat:  $x_1 x_2 - t_1 t_2$

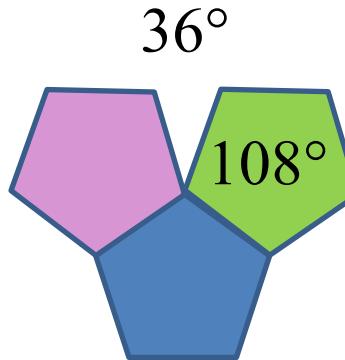
# Euklideszi sík csempézése szabályos, egybevágó sokszögekkel (tesszelláció)



{3, 6}

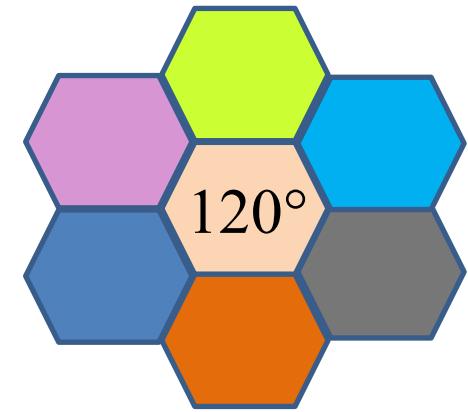


{4, 4}



36°

108°

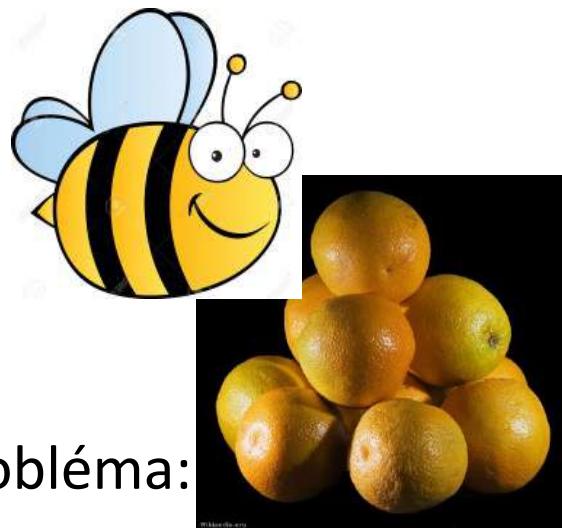


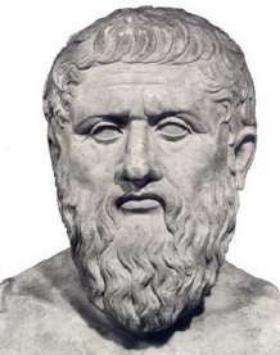
{6, 3}

Schläfli szimbólum: {n, k}

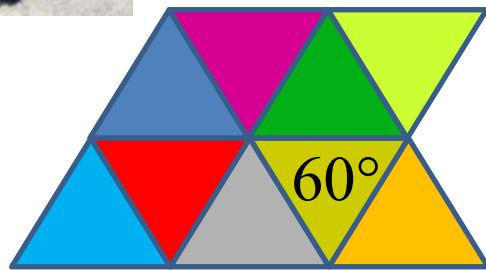
$$\frac{1}{n} + \frac{1}{k} = \frac{1}{2}$$

Kepler probléma:

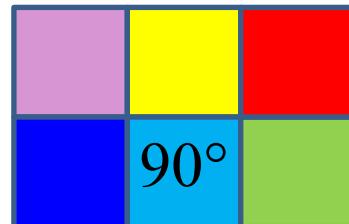




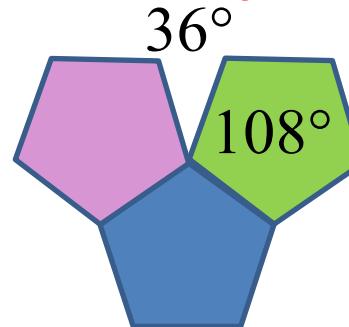
# Platóni szabályos testek (gömb csempézése)



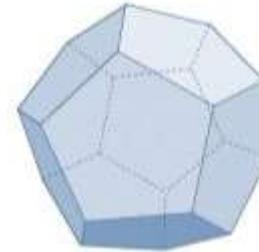
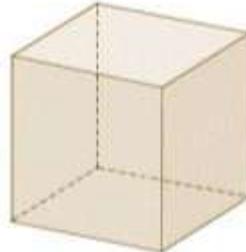
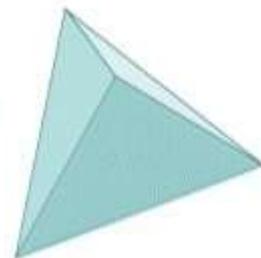
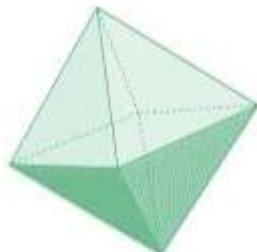
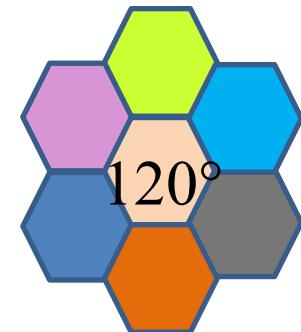
5      4      3



3



3



$$K = \pi/3$$

$$2\pi/3$$

$$\pi$$

$$\pi/2$$

$$\pi/5$$

$$\{3, 5\}$$

$$\{3, 4\}$$

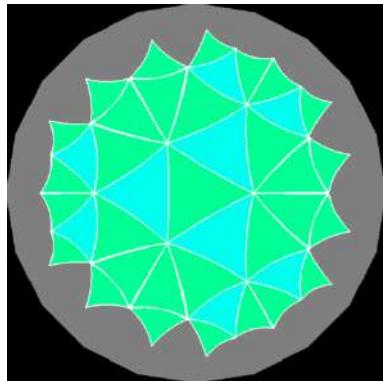
$$\{3, 3\}$$

$$\{4, 3\}$$

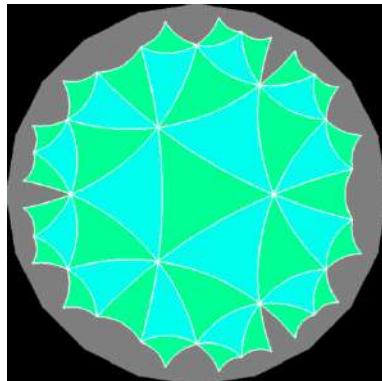
$$\{5, 3\}$$

Schläfli:  $\frac{1}{n} + \frac{1}{k} > \frac{1}{2}$

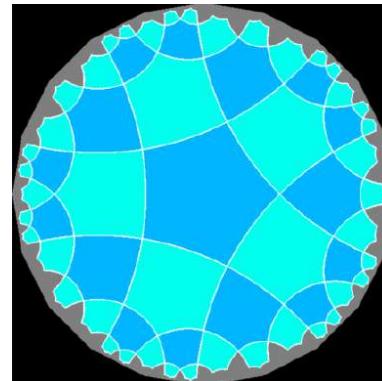
# Hiperbolikus sík csempézése szabályos, egybevágó sokszögekkel (tesszelláció)



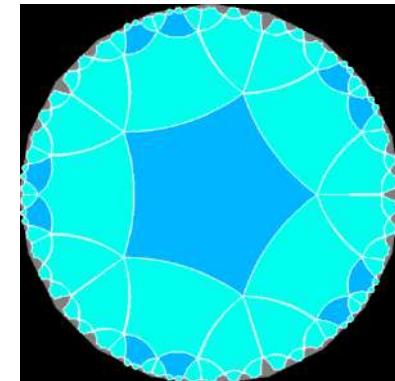
{3, 7}



{3, 8}



{5, 4}



{5, 5}

Schlafli:  $\frac{1}{n} + \frac{1}{k} < \frac{1}{2}$





# Escher és a Poincaré diszkrétum

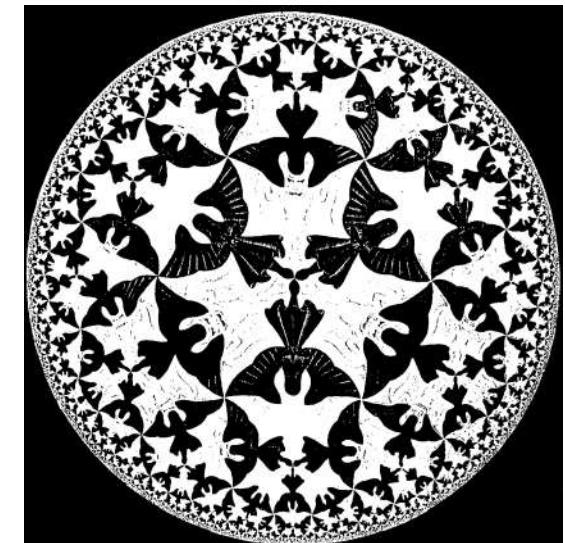
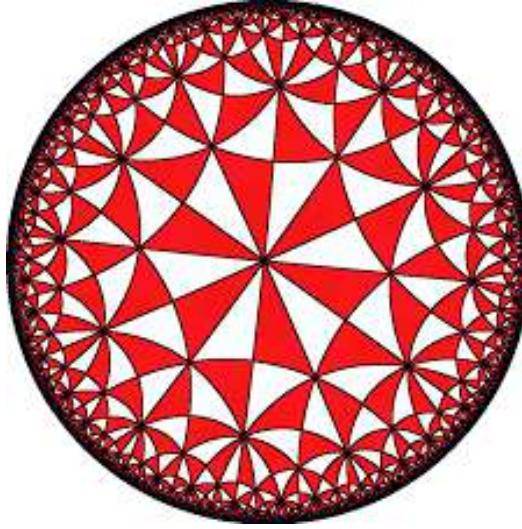
Circle limit III



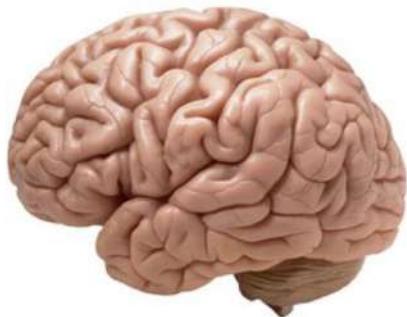
Circle limit I



Circle limit IV



# Hiperbolikus geometria

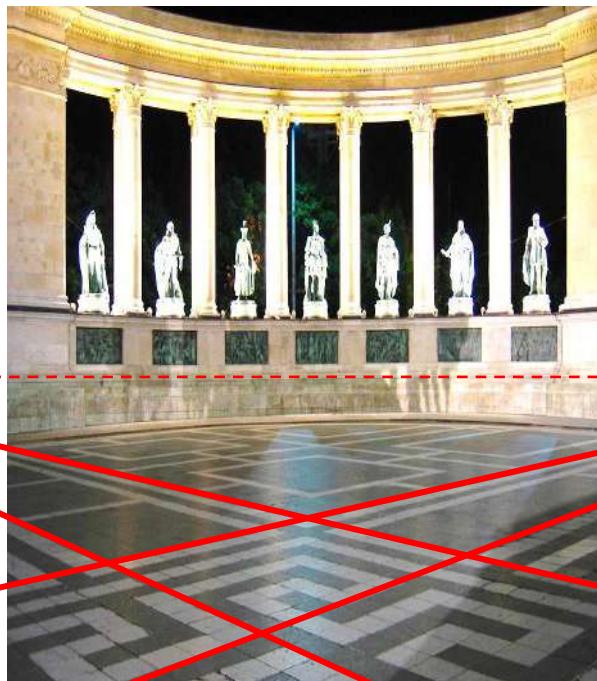


# Projektív geometria

- A „végtelen” is része a síknak
- Nincsenek párhuzamosok
- Programozási előny: nincs szingularitás és speciális eset
- Descartes és polár (és bármilyen távolsági) koordináták nem jók
- Az árnyék vagy a skála nélküli vonalzó geometriája

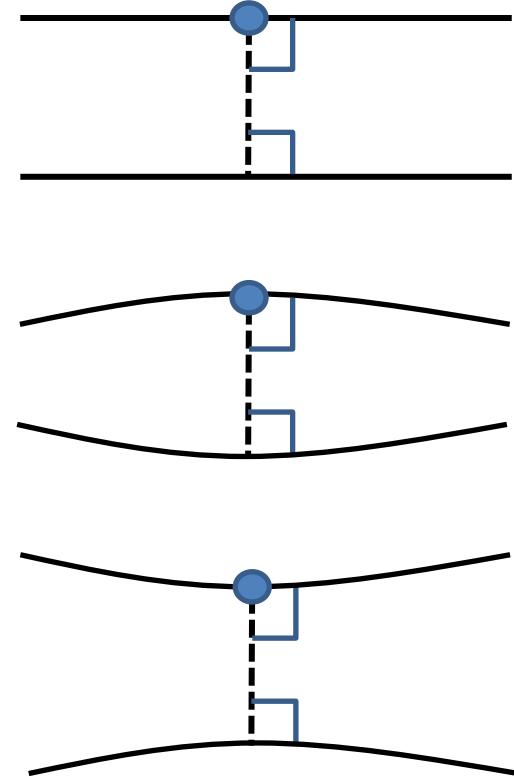
Projektív geometria axiómái:

1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. **Két egyenes pontosan egy pontban metszi egymást**



# Modell geometriák (homogén, izotróp)

- Euklideszi
  - 1 nem metsző egyenes (párhuzamos)
  - Zérus görbület
- Elliptikus
  - 0 nem metsző egyenes
  - Pozitív görbület
- Hiperbolikus
  - Egynél több nem metsző egyenes
  - Negatív görbület
- Projektív
  - 0 nem metsző egyenes
  - Nem metrikus: végtelen is része



*“Tant que l’Algèbre et la Géométrie ont été séparées, leurs progrès ont été lents et leurs usages bornés; mais lorsque ces deux sciences se sont réunies, elles se sont prêté des forces mutuelles et ont marché ensemble d’un pas rapide vers la perfection.”*

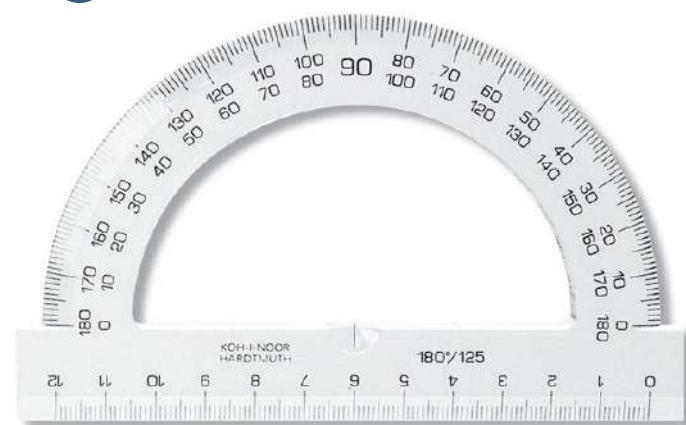
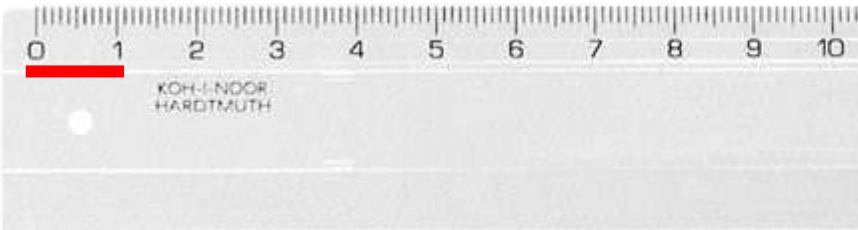
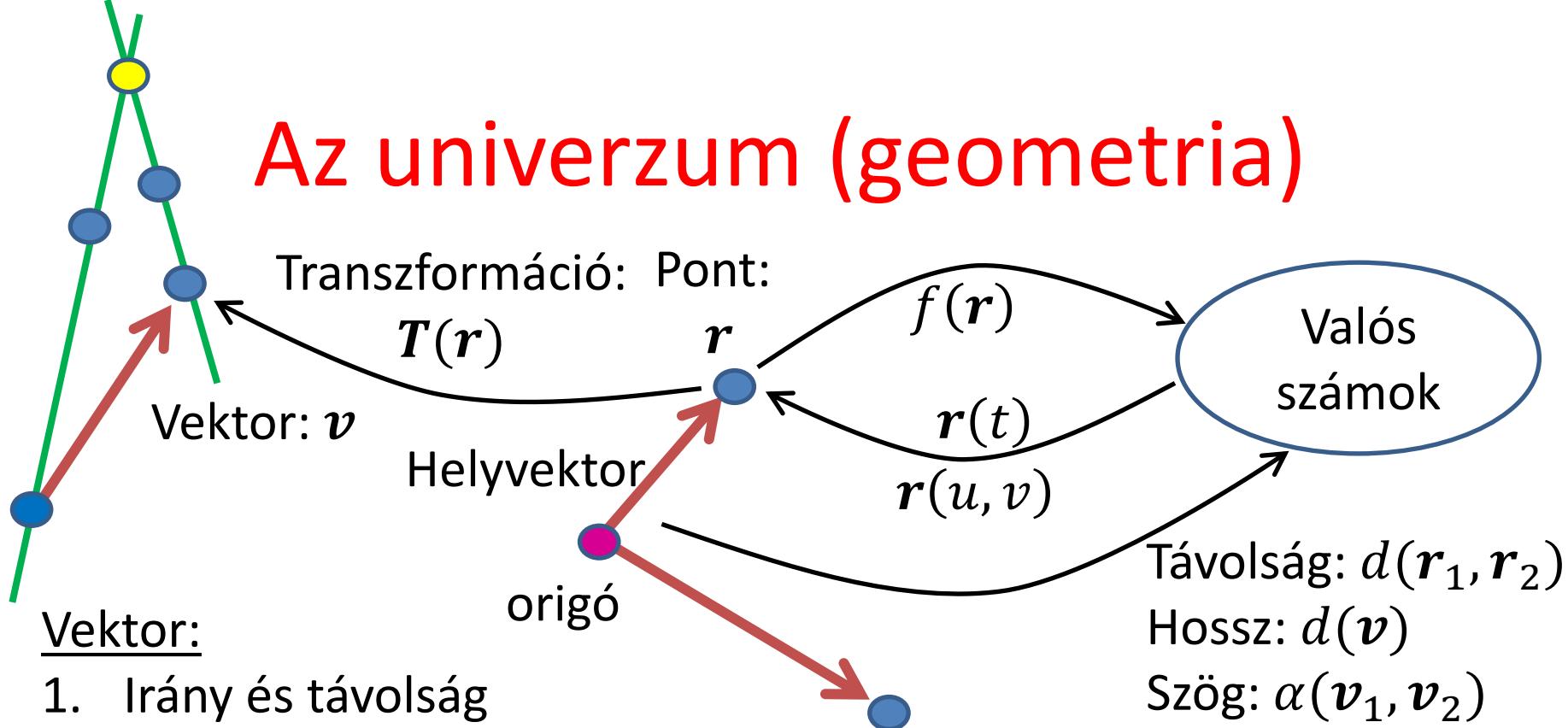
*Joseph-Louis Lagrange*

# Geometriák és algebrák

## 2. Vektoralgebra

Szirmay-Kalos László

# Az univerzum (geometria)

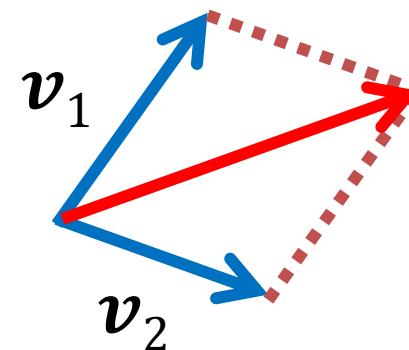


Mérés: távolság és szög

# Műveletek vektorokkal

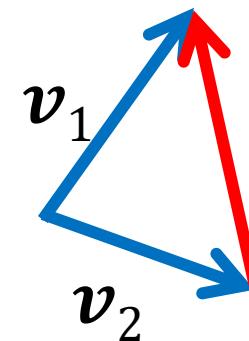
- Összeadás

$$v = v_1 + v_2 \text{ (kom, asszoc)}$$



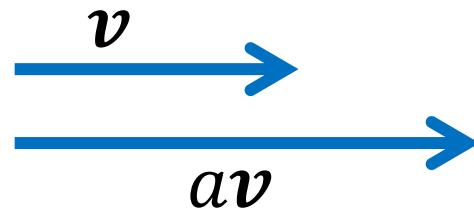
- Kivonás

$$v = v_1 - v_2$$



- Skálázás (skalárral szorzás)

$$v_1 = av \quad (\text{disztributív})$$



# Skalár (dot, belső) szorzat

- Definíció:  $\mathbf{v}_1 \cdot \mathbf{v}_2 = |\mathbf{v}_1| |\mathbf{v}_2| \cos(\alpha)$

- Jelentés

Egyik vektor vetülete a másikra \* másik hossza

- Tulajdonság: Nem asszociatív(!), Kommutatív, Disztributív az összeadással

$$\mathbf{v}_3 \cdot (\mathbf{v}_2 + \mathbf{v}_1) = \mathbf{v}_3 \cdot \mathbf{v}_2 + \mathbf{v}_3 \cdot \mathbf{v}_1$$

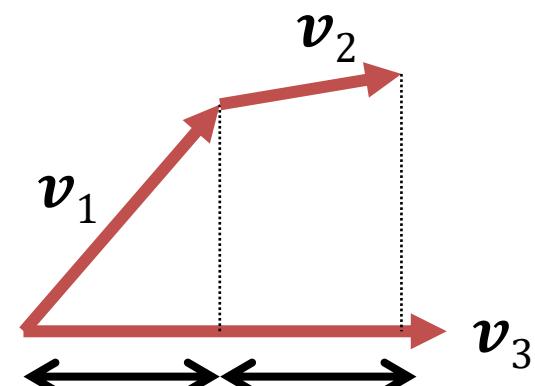
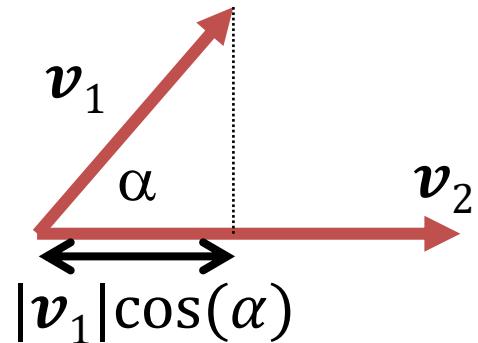
- Varázspálca (metrika):

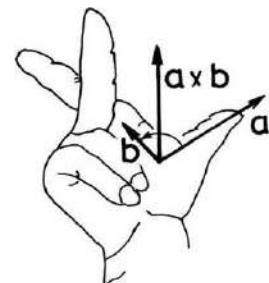
$$-\mathbf{v} \cdot \mathbf{v} = |\mathbf{v}|^2$$

$$-\mathbf{v}^0 = \mathbf{v} / \sqrt{\mathbf{v} \cdot \mathbf{v}}$$

$$-\mathbf{v}_1 \cdot \mathbf{v}_2 / |\mathbf{v}_1| |\mathbf{v}_2| = \cos(\alpha)$$

– Két vektor merőleges  $\Leftrightarrow$  skalárszorzatuk zérus





# Vektor (cross, kereszt) szorzat

- Definíció:  $|\boldsymbol{v}_1 \times \boldsymbol{v}_2| = |\boldsymbol{v}_1||\boldsymbol{v}_2|\sin(\alpha)$

- Jelentés:

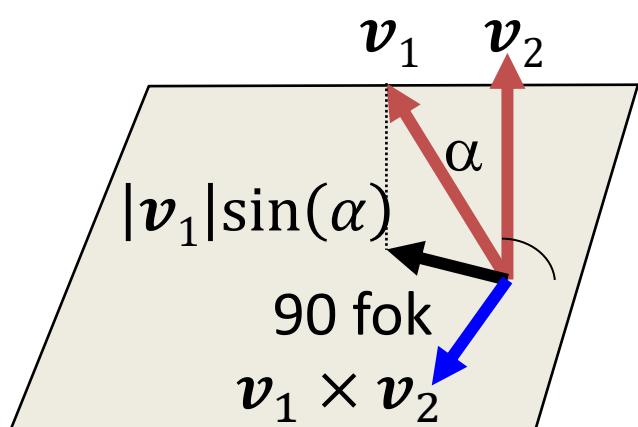
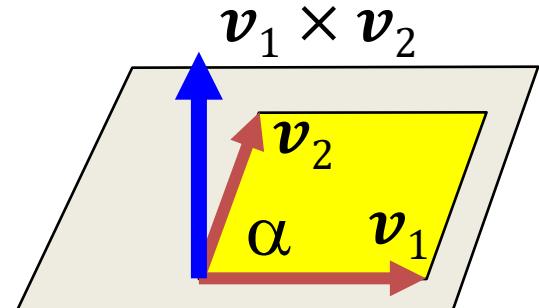
- Nagyság (terület, hossz, térfogat) és merőleges vektor
- (Egyik vektor vetülete a másikra merőleges síkra + 90 fokos elforgatás) \* másik hossza

- Tulajdonságok:

- Nem asszociatív!

- Antiszimmetrikus:  $\boldsymbol{v}_1 \times \boldsymbol{v}_2 = -\boldsymbol{v}_2 \times \boldsymbol{v}_1$

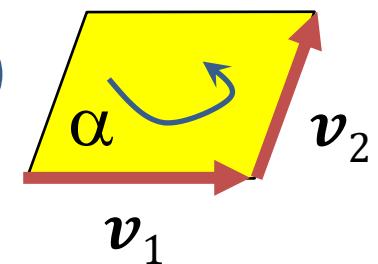
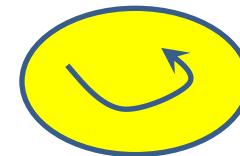
- Disztributív:  $\boldsymbol{v}_1 \times (\boldsymbol{v}_2 + \boldsymbol{v}_3) = \boldsymbol{v}_1 \times \boldsymbol{v}_2 + \boldsymbol{v}_1 \times \boldsymbol{v}_3$





# Külső (wedge) szorzat

$$B = \mathbf{v}_1 \wedge \mathbf{v}_2$$



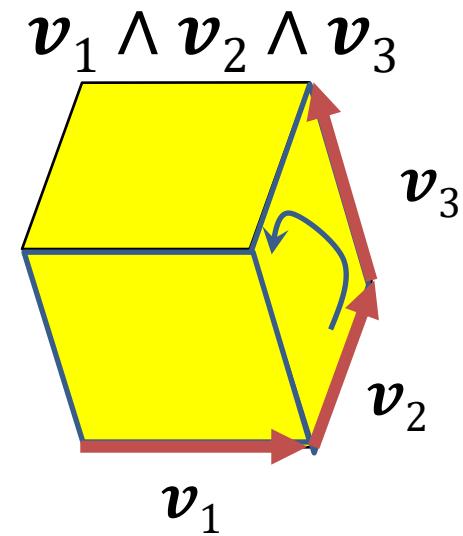
- **Definíció:**  $|\mathbf{v}_1 \wedge \mathbf{v}_2| = |\mathbf{v}_1| |\mathbf{v}_2| \sin(\alpha)$

- **Jelentés:** Irányított terület/térfogat

- **Multivektor:**  $\mathbf{V} = s + \mathbf{v} + \mathbf{B}$ 
  - Összeadás, skálázás a szokásos módon

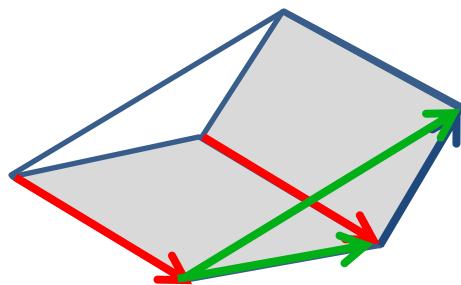
- **Tulajdonságok:**

- Asszociatív
- Antiszimmetrikus:  $\mathbf{v}_1 \wedge \mathbf{v}_2 = -\mathbf{v}_2 \wedge \mathbf{v}_1$
- Disztributív:  $\mathbf{v}_1 \wedge (\mathbf{v}_2 + \mathbf{v}_3) = \mathbf{v}_1 \wedge \mathbf{v}_2 + \mathbf{v}_1 \wedge \mathbf{v}_3$

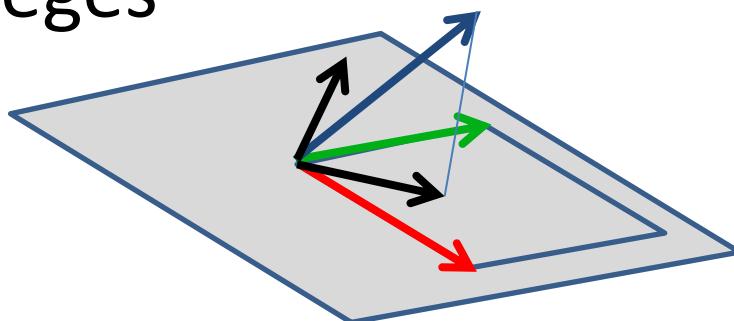


# Műveletek bivektorokkal

- Számmal szorzás: értelemszerű
- Összeadás:

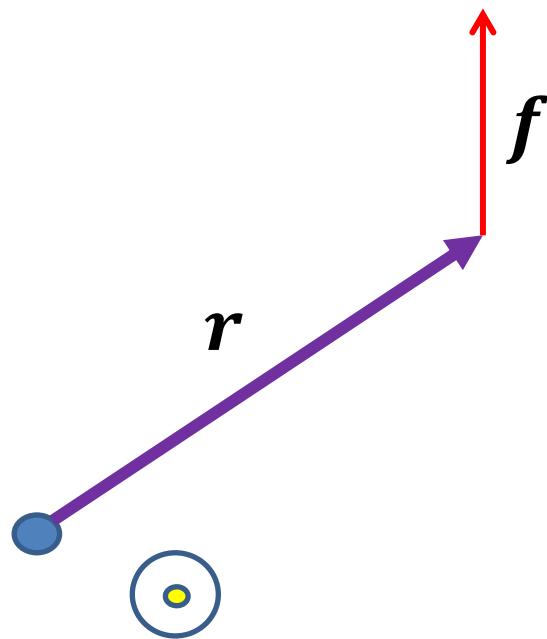


- Belső szorzat: A bivektor síkjában, a vektorra merőleges



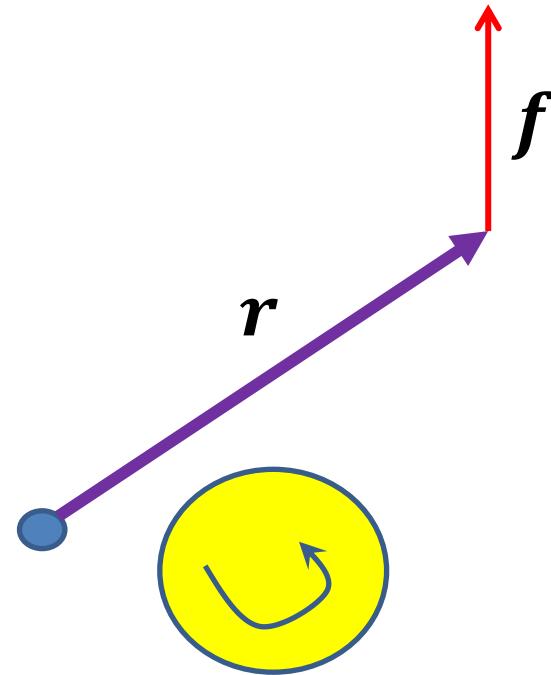
# Külső szorzat példa: Forgatónyomaték

Vektoriális szorzat



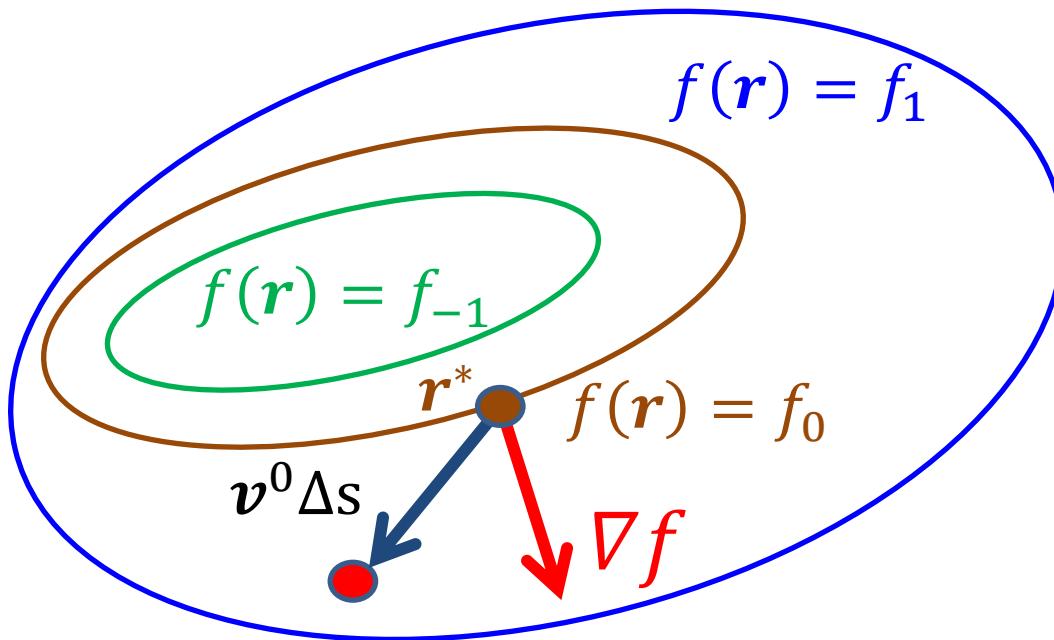
$$\mathbf{M} = \mathbf{r} \times \mathbf{f}$$

Külső szorzat



$$\mathbf{M} = \mathbf{r} \wedge \mathbf{f}$$

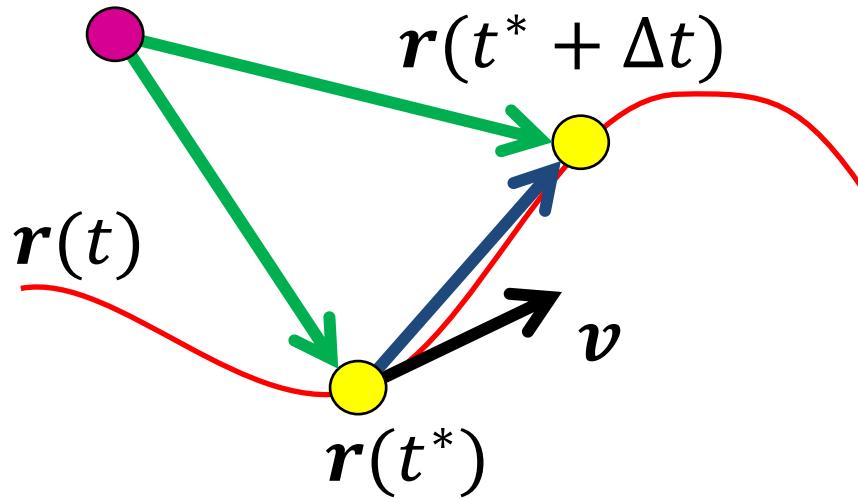
# Gradiens: $\nabla f$



$$\lim_{\Delta s} \frac{f(r^* + v^0 \Delta s) - f(r^*)}{\Delta s} = \frac{df(r^* + v^0 s)}{ds} \quad \text{iránymenti derivált}$$

$\frac{df}{dr} = \nabla f$  a maximális növekedés iránya és rátája

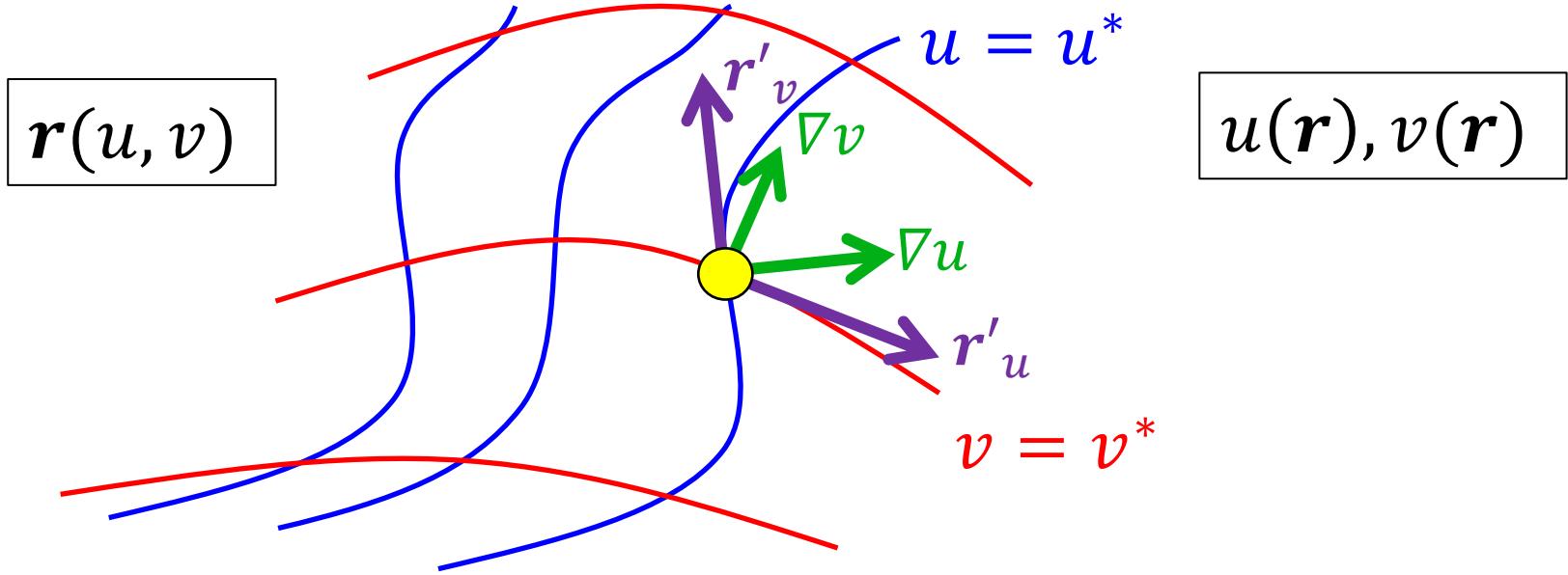
# Paraméter szerinti deriválás



Sebesség (érintő) vektor:

$$\mathbf{v} = \frac{d\mathbf{r}(t)}{dt} = \lim \frac{\mathbf{r}(t^* + \Delta t) - \mathbf{r}(t^*)}{\Delta t}$$

# Koordináták

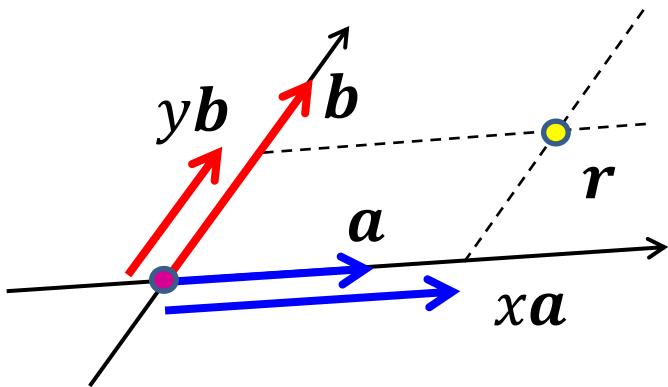


- Az  $r(t)$  vagy  $r(u, v)$  minden  $t$ -hez vagy  $(u, v)$ -hez egy pontot rendel.
- Az ilyen számok (számtöbbesek) a **koordináták**

# Megfigyelések

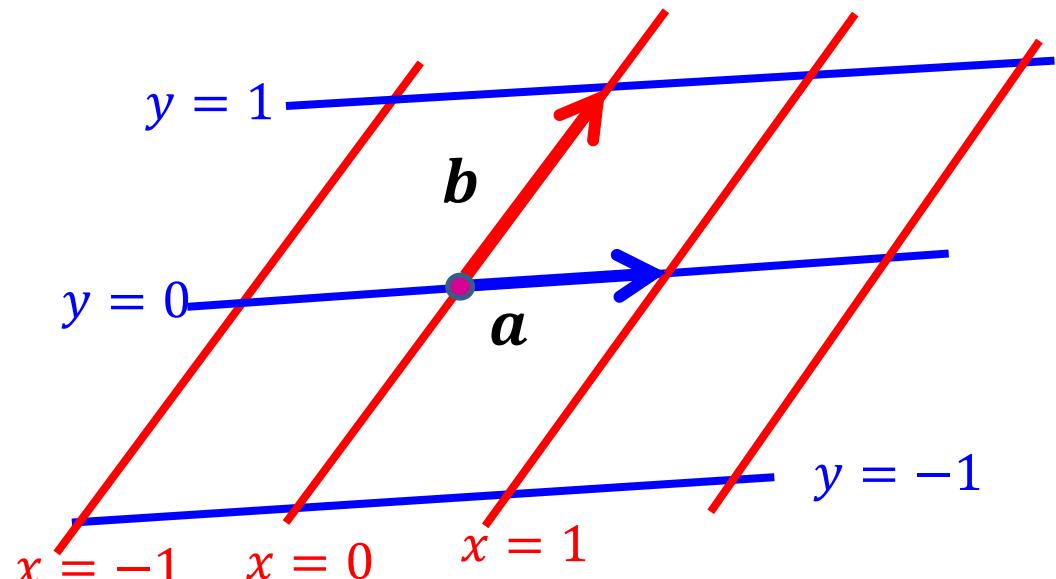
- Idáig nem használtuk a koordinátákat.
- A pontok, vektorok, távolság, szög a geometria részei, a koordinátarendszertől nem függnek.
- A vektoralgebrai műveletek és deriválás koordinátarendszer nélkül is végrehajthatók.
- A koordinátarendszer szükséges rossz annak érdekében, hogy számokkal dolgozhassunk: a komponensek nem csak a geometriai objektumtól, hanem a koordinátarendszertől is függenek.
- Vektor = elsőrendű tensor
  - Tenzor = ( $0, 1, 2, \dots$  dimenziós) tömb, amely a tensor szabályok szerint transzformálódik koordinátarendszer váltáskor
  - A vektorok és műveleteik függetlenek a koordinátarendszertől, de komponensei függenek
  - A fizikai törvényekben csak tensorok szerepelhetnek

# Kontravariáns koordinátarendszer



$$\mathbf{r}(x, y) = x\mathbf{a} + y\mathbf{b}$$

$$\mathbf{a} = \mathbf{r}'_x, \quad \mathbf{b} = \mathbf{r}'_y$$



- **Összeg:**

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

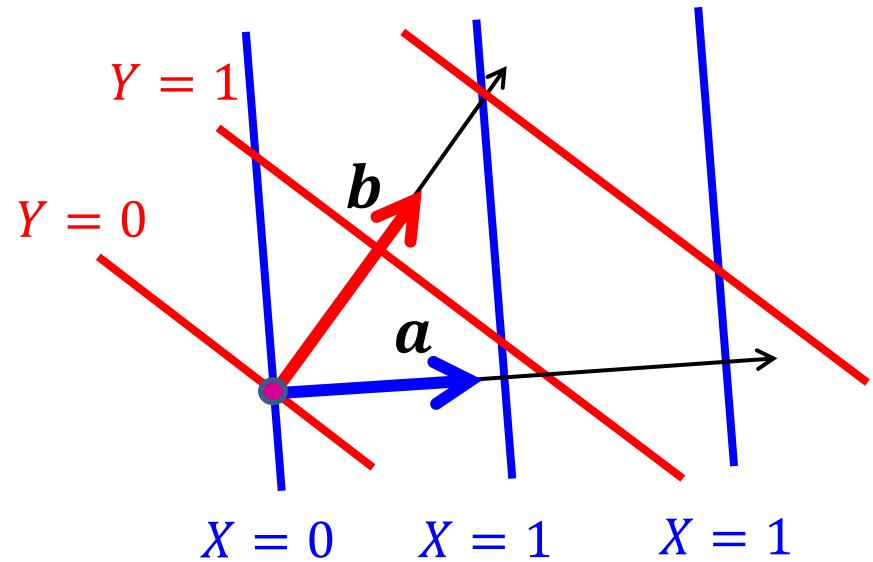
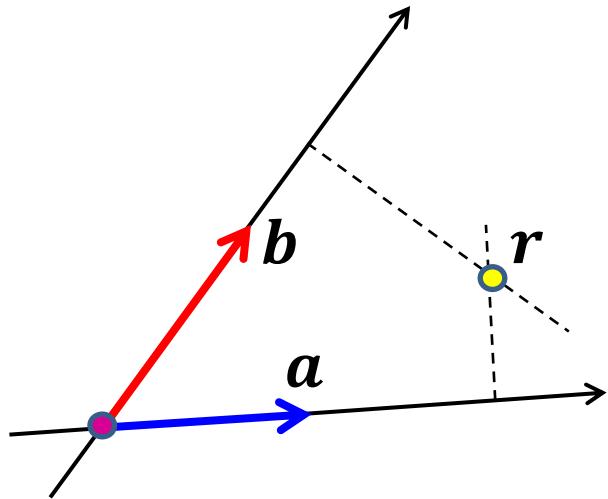
- **Skalár szorzat:**

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = (x_1\mathbf{a} + y_1\mathbf{b}) \cdot (x_2\mathbf{a} + y_2\mathbf{b}) = x_1 x_2 \mathbf{a} \cdot \mathbf{a} + (x_1 y_2 + x_2 y_1) \mathbf{a} \cdot \mathbf{b} + y_1 y_2 \mathbf{b} \cdot \mathbf{b}$$

- **Hossz:**

$$|\mathbf{v}| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{x^2 \mathbf{a} \cdot \mathbf{a} + 2xy\mathbf{a} \cdot \mathbf{b} + y^2 \mathbf{b} \cdot \mathbf{b}}$$

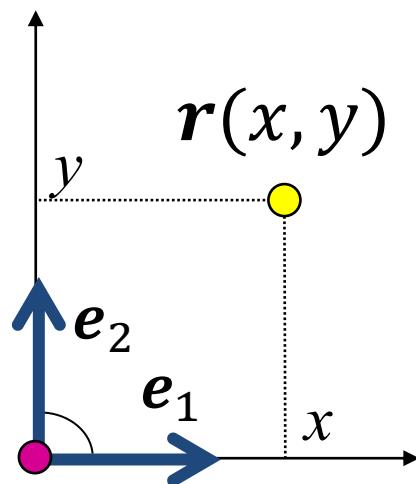
# Kovariáns koordinátarendszer



$$X = \mathbf{a} \cdot \mathbf{r}, \quad Y = \mathbf{b} \cdot \mathbf{r}$$

$$\mathbf{a} = \nabla X, \quad \mathbf{b} = \nabla Y$$

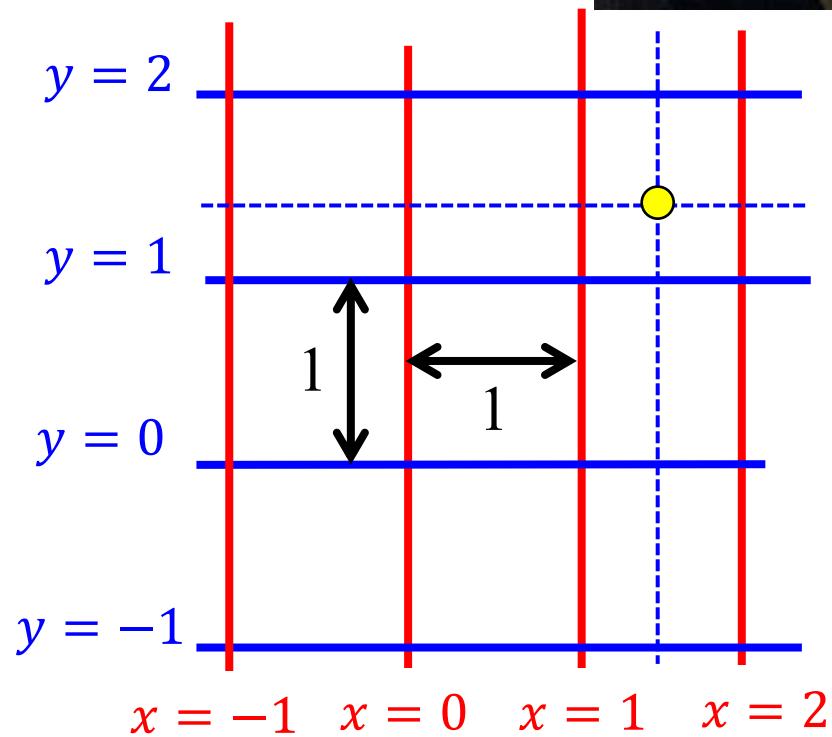
# (René) Descartes koordináta rendszer



•	1	$e_1$	$e_2$
1	1	$e_1$	$e_2$
$e_1$	$e_1$	1	0
$e_2$	$e_2$	0	1

$$\mathbf{r}(x, y) = x\mathbf{e}_1 + y\mathbf{e}_2$$

$$x = \mathbf{e}_1 \cdot \mathbf{r}, \quad y = \mathbf{e}_2 \cdot \mathbf{r}$$



$$\mathbf{r}'_x = \mathbf{e}_1, \quad \mathbf{r}'_y = \mathbf{e}_2,$$

$$\mathbf{e}_1 = \nabla x, \quad \mathbf{e}_2 = \nabla y$$

# Műveletek Descartes koordinátákkal

- Összeadás:

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

- Skalár szorzat:

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

- Hossz:

$$|\boldsymbol{v}| = \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}} = \sqrt{x^2 + y^2} \quad \underline{\text{Pitagorász tétele!}}$$

- Gradiens:

$$\frac{df(x+v_x s, y+v_y s)}{ds} = \frac{\partial f}{\partial x} v_x + \frac{\partial f}{\partial y} v_y = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \cdot \boldsymbol{v}^0$$

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

# Vektoriális szorzat (csak 3D-ben)

- 3D-ben:

$$\begin{aligned}\boldsymbol{v}_1 \times \boldsymbol{v}_2 &= (x_1 \boldsymbol{e}_1 + y_1 \boldsymbol{e}_2 + z_1 \boldsymbol{e}_3) \times (x_2 \boldsymbol{e}_1 + y_2 \boldsymbol{e}_2 + z_2 \boldsymbol{e}_3) = \\ &(y_1 z_2 - y_2 z_1) \boldsymbol{e}_1 + (x_2 z_1 - x_1 z_2) \boldsymbol{e}_2 + (x_1 y_2 - y_1 x_2) \boldsymbol{e}_3\end{aligned}$$

$$= \begin{vmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

- 2D-ben:  $\times \boldsymbol{v} = \begin{vmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 \\ x & y \end{vmatrix} = (y, -x)$

- 4D-ben:  $\times (\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3) = \begin{vmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 & \boldsymbol{e}_4 \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{vmatrix}$

# Külső szorzat (Bármilyen D-ben)

- **2D: Terület körüljárással**

$$\begin{aligned}\boldsymbol{v}_1 \wedge \boldsymbol{v}_2 &= (x_1 \mathbf{e}_1 + y_1 \mathbf{e}_2) \wedge (x_2 \mathbf{e}_1 + y_2 \mathbf{e}_2) = \\ &(x_1 y_2 - y_1 x_2) \mathbf{e}_1 \wedge \mathbf{e}_2 = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} \mathbf{e}_1 \wedge \mathbf{e}_2\end{aligned}$$

- **3D: Terület 3D orientációval**

$$\begin{aligned}\boldsymbol{v}_1 \wedge \boldsymbol{v}_2 &= (x_1 \mathbf{e}_1 + y_1 \mathbf{e}_2 + z_1 \mathbf{e}_3) \wedge (x_2 \mathbf{e}_1 + y_2 \mathbf{e}_2 + z_2 \mathbf{e}_3) = \\ &(y_1 z_2 - y_2 z_1) \mathbf{e}_2 \wedge \mathbf{e}_3 + (x_2 z_1 - x_1 z_2) \mathbf{e}_3 \wedge \mathbf{e}_1 + \\ &(x_1 y_2 - y_1 x_2) \mathbf{e}_1 \wedge \mathbf{e}_2 = \begin{vmatrix} \mathbf{e}_2 \wedge \mathbf{e}_3 & \mathbf{e}_3 \wedge \mathbf{e}_1 & \mathbf{e}_1 \wedge \mathbf{e}_2 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}\end{aligned}$$

**Három szorzata: Térfogat körüljárással**

$$\boldsymbol{v}_1 \wedge \boldsymbol{v}_2 \wedge \boldsymbol{v}_3 = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3$$

*“Algebra is the offer made by the devil:  
Give me your soul, give up geometry and  
you will have this marvelous machine.”*

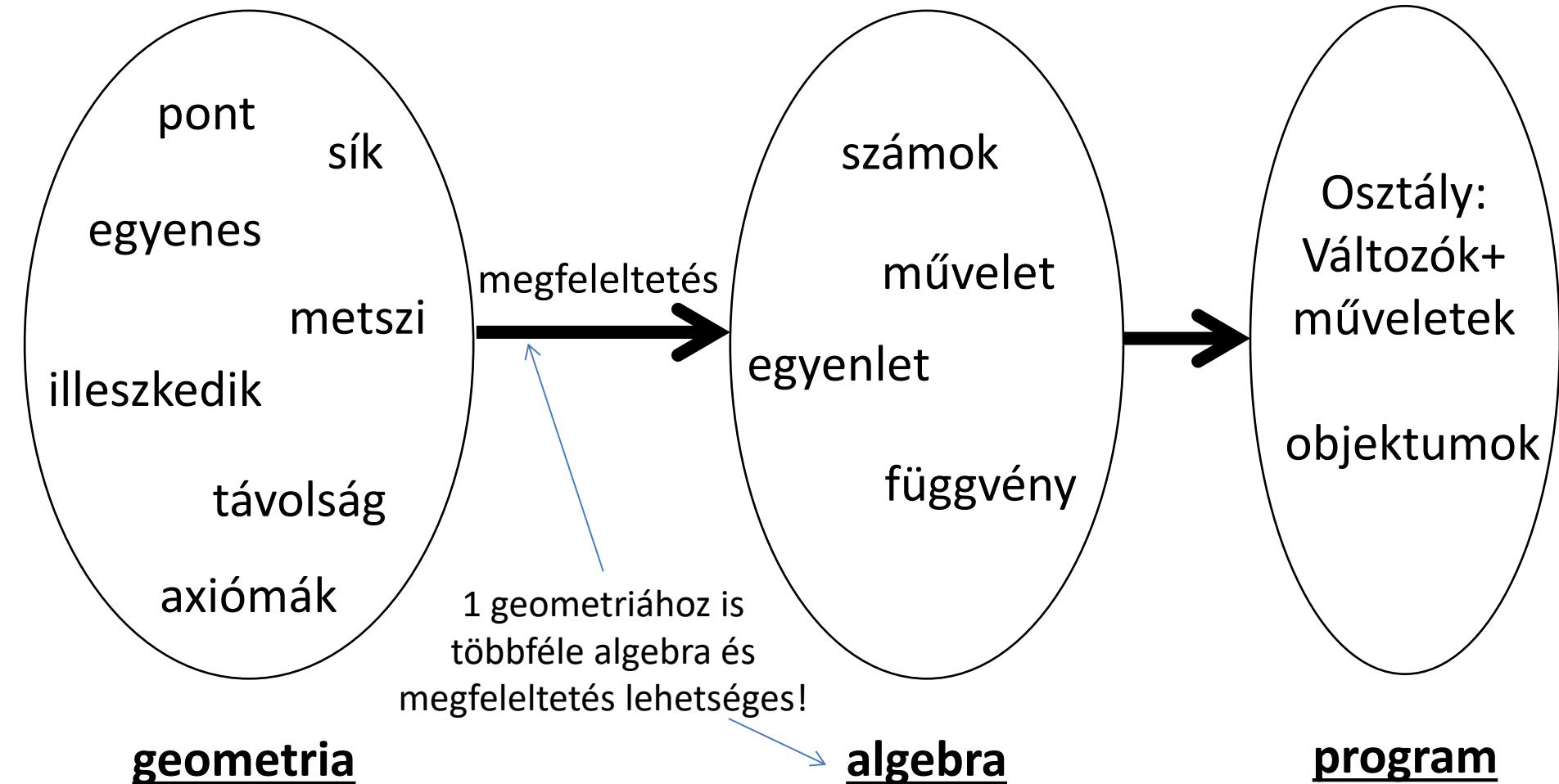
*Michael Francis Atiyah*

# Geometriák és algebrák

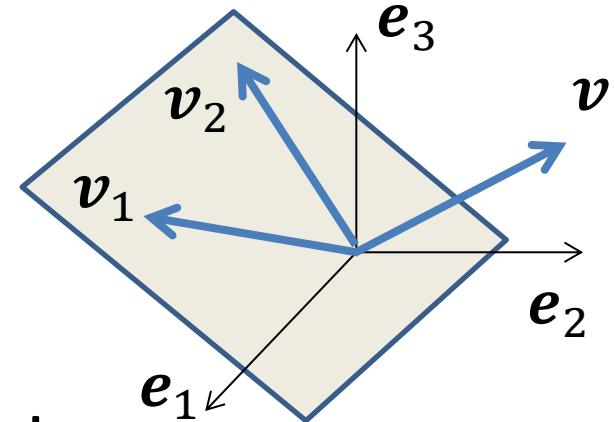
## 4. Analitikus geometriák

Szirmay-Kalos László

# Mindent számmal!

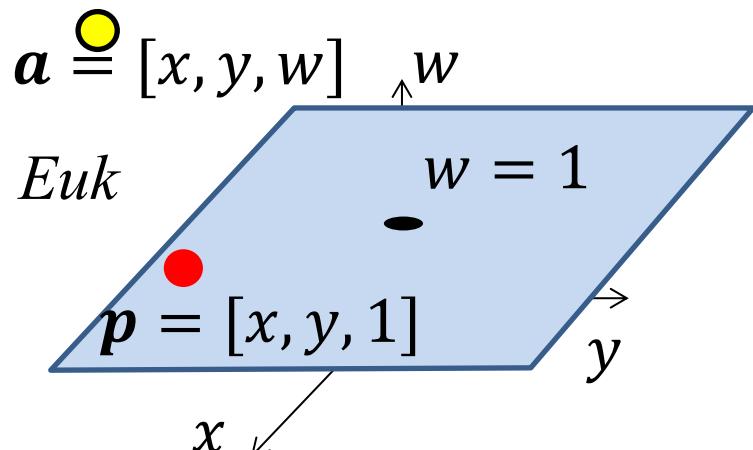


# Lineáris algebra kritikája

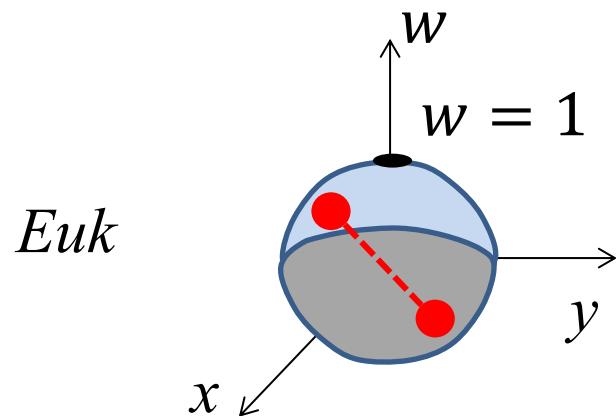


- 1 vektor kifeszít egy 1D alteret
  - origón átmenő egyenes
  - Nem origón átmenő???
- 2 vektor kifeszít egy 2D alteret
  - origót tartalmazó sík
  - Nem origón átmenő???
  - Eukleidész: „Két pont definiál egy egyenest”
- 3 vektor kifeszít egy 3D teret

# Analitikus geometriák: külső nézet

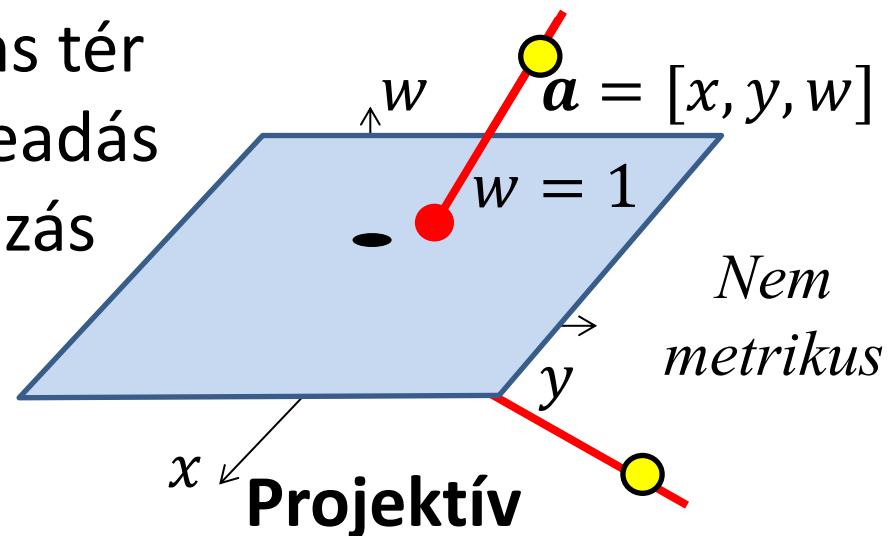


**Euklideszi:**  $w = 1$

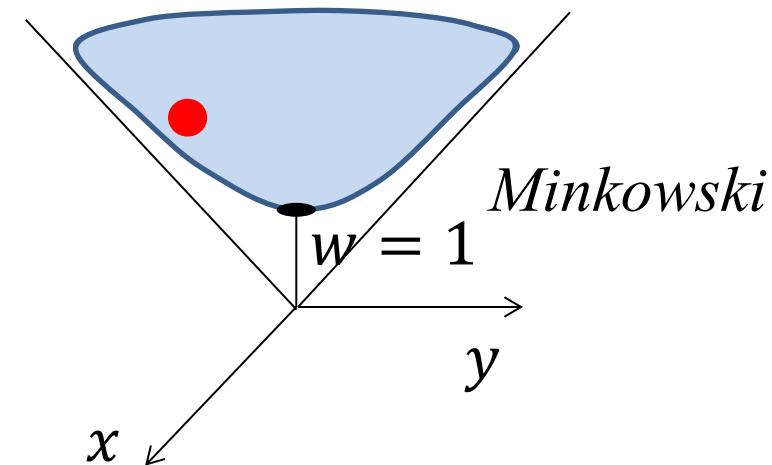


**Gömbi:**  $x^2 + y^2 + w^2 = 1$

Ambiens tér  
• Összeadás  
• Skálázás



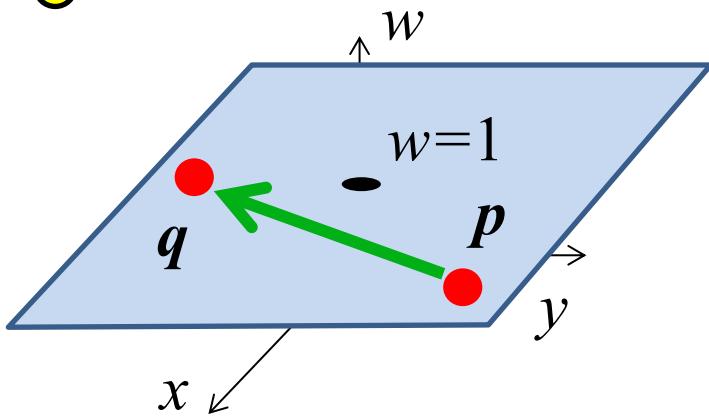
Nem metrikus



**Hiperbolikus:**  $x^2 + y^2 - w^2 = -1$

# Euklideszi síkgeometria

$a = [x, y, w]$



Skaláris szorzás az ambiens térben:

$$a_1 \cdot a_2 = x_1 x_2 + y_1 y_2 + w_1 w_2$$

- Kommutatív:  $a \cdot b = b \cdot a$
- Disztributív:  $a \cdot (b + c) = a \cdot b + a \cdot c$
- Skálázás:  $(sa) \cdot b = s(a \cdot b)$

Ambiens tér elemei:

- Pontok:

$$p = [x, y, 1]$$

Miért nem  $w = 0$ ? Az eltolás nem volna lineáris leképzés

- Vektorok:

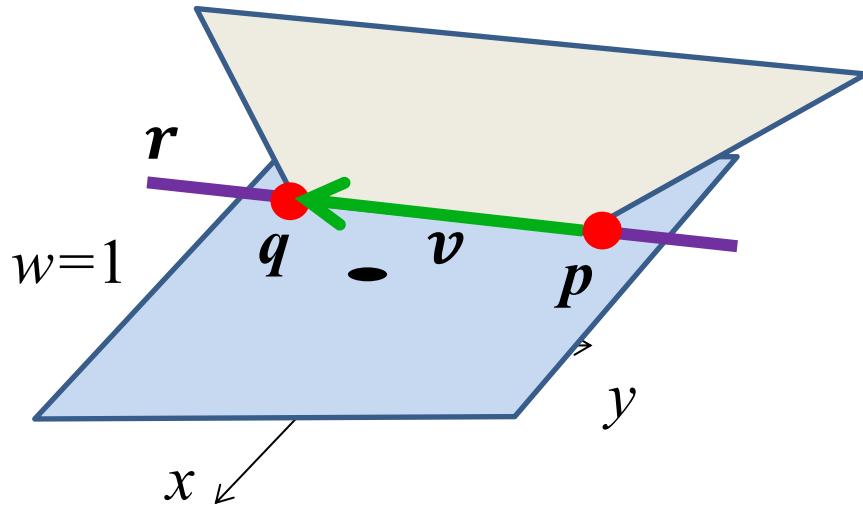
$$v = q - p, \quad v = [x, y, 0]$$

- Egyéb  $w$ -k nem pontok és nem vektorok

# Vektorok tulajdonságai

- Két pont különbsége:  $\boldsymbol{v} = \boldsymbol{q} - \boldsymbol{p}$
- Ambiens tér eleme:  $\boldsymbol{v} = [x, y, 0]$
- Vektor hossza:  $|\boldsymbol{v}| = \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}} = \sqrt{x^2 + y^2}$
- Egységvektor:  $|\boldsymbol{v}^0| = 1, \boldsymbol{v}^0 \cdot \boldsymbol{v}^0 = 1, \boldsymbol{v}^0 = \frac{\boldsymbol{v}}{\sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}}$
- Merőlegesség:  $\boldsymbol{v}_\perp \cdot \boldsymbol{v} = 0, x_\perp x + y_\perp y = 0,$   
 $\boldsymbol{v}_\perp = [-y, x, 0]\lambda$
- Párhuzamosság:  $\boldsymbol{v}_\parallel = \lambda \boldsymbol{v}, \boldsymbol{v}_\parallel = [x, y, 0]\lambda$

# Egyenes: paraméteres egyenlet



Állandó sebességű mozgás:

$$\mathbf{r}(t) = \mathbf{p} + \mathbf{v}t$$

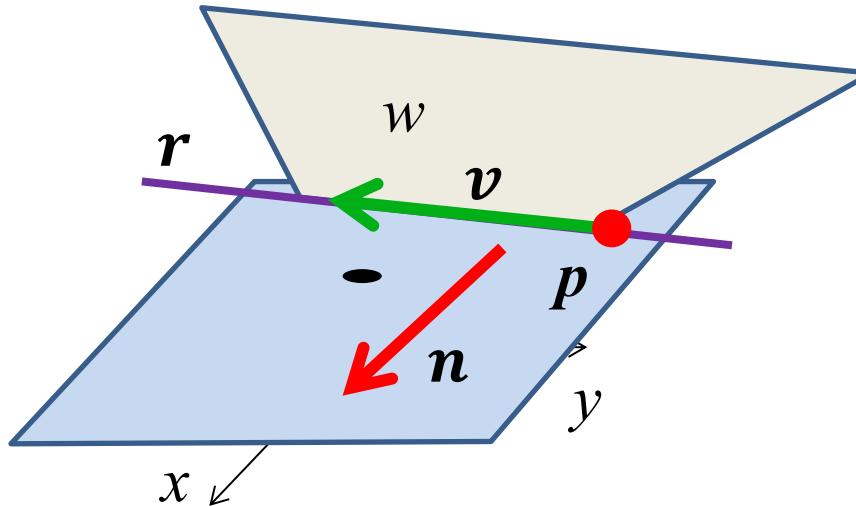
Sebesség:  $\dot{\mathbf{r}}(t) = \mathbf{v}$

Gyorsulás:  $\ddot{\mathbf{r}}(t) = 0$

Sík görbülete zérus

- Állandó sebességű mozgás koordinátákkal:  
 $[x(t), y(t), w(t)] = [p_x, p_y, 1] + [v_x, v_y, 0]t$
- Két pont kombinációja:  $\mathbf{r}(t) = \mathbf{p} + (\mathbf{q} - \mathbf{p})t = \mathbf{p}(1 - t) + \mathbf{q}t$   
 $[x(t), y(t), w(t)] = [p_x, p_y, 1](1 - t) + [q_x, q_y, 1]t$
- Ambiens tér (origó,  $\mathbf{p}, \mathbf{q}$ ) síkjának és a  $w = 1$  síknak metszete

# Egyenes implicit egyenletei



- **Implicit egyenlet:**  $\mathbf{n} = \mathbf{v}_\perp \Rightarrow \mathbf{n} \cdot \mathbf{v} = 0 \Rightarrow \mathbf{n} \cdot (\mathbf{r} - \mathbf{p}) = 0$   
 $[n_x, n_y, 0] \cdot [x - p_x, y - p_y, 0] = \boxed{n_x x + n_y y + d = 0}$   
ahol  $d = -\mathbf{n} \cdot \mathbf{p} = -n_x p_x - n_y p_y$
- Ambiens tér egy eleme:  $\mathbf{N} = [n_x, n_y, d]$   
 $\mathbf{N} \cdot \mathbf{r} = [n_x, n_y, d] \cdot [x, y, w] = n_x x + n_y y + dw = 0,$   
ahol  $\mathbf{r}$  pont, azaz  $w = 1$ . Egyenes = két sík metszete.
- Két egyenes megegyezik, ha  $\mathbf{N}_1 = \mathbf{N}_2 \lambda$

# Távolság

Távolság  $d(p, q)$ : két pont különbsége = egységsebességű egyenesvonalú mozgás ideje:

$$\mathbf{r}(d) = \mathbf{p} + \mathbf{v}^0 d = \mathbf{q}$$

$$\Rightarrow \mathbf{v}^0 d = \mathbf{q} - \mathbf{p}$$

$$\Rightarrow \mathbf{v}^0 \cdot \mathbf{v}^0 d^2 = (\mathbf{q} - \mathbf{p}) \cdot (\mathbf{q} - \mathbf{p})$$

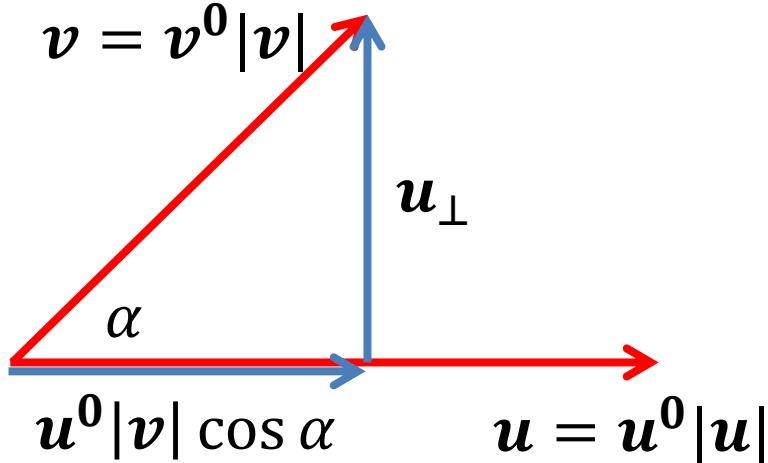
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{q} - \mathbf{p}) \cdot (\mathbf{q} - \mathbf{p})}$$

# Szög

- Szög  $\alpha(\mathbf{u}, \mathbf{v})$ : két irány (egységvektor) különbsége

$$\alpha(\mathbf{u}, \mathbf{v}) = \arccos \left( \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}| |\mathbf{v}|} \right)$$

- Miért definiáljuk így?



$$\begin{aligned}\mathbf{u} \cdot \mathbf{v} &= u^0 |\mathbf{u}| \cdot (u^0 |\mathbf{v}| \cos \alpha + u_\perp) \\ &= u^0 \cdot u^0 |\mathbf{u}| |\mathbf{v}| \cos \alpha \\ &= |\mathbf{u}| |\mathbf{v}| \cos \alpha\end{aligned}$$

# Tényleg Euklideszi síkgeometria?

1. Két különböző pont  $\mathbf{q}, \mathbf{p}$  meghatároz egy  $\mathbf{r}(t)$  egyenest.

$$\mathbf{r}(t) = \mathbf{p} + (\mathbf{q} - \mathbf{p})t = \mathbf{p}(1 - t) + \mathbf{q}t$$

2. Egy egyenesnek van legalább két pontja.

$\mathbf{r}(t_1)$  és  $\mathbf{r}(t_2)$  különböző, ha  $t_1$  és  $t_2$  különböző.

3. Egy  $n_{1x}x + n_{1y}y + d_1 = 0$  egyeneshez egy rajta kívül fekvő  $\mathbf{p}_2$  ponton át egy nem metsző  $n_{2x}x + n_{2y}y + d_2 = 0$  egyenes húzható.

Metszés = közös pont = egyenlet megoldás

$$n_{1x}x + n_{1y}y = -d_1$$

$$n_{2x}x + n_{2y}y = -d_2$$

Nincs metszés vagy egybeesés ha determináns zérus:  
 $(n_{2x}, n_{2y}) = (n_{1x}, n_{1y})\lambda$ . Mivel  $\mathbf{p}_2$  nincs rajta az első egyenesen, csak nincs metszés lehet.

Tudjuk, hogy  $d_2 = -n_{2x}p_{2x} - n_{2y}p_{2y} = -(n_{1x}p_{2x} + n_{1y}p_{2y})\lambda$

Nem metsző egyenes paraméterei:  $[n_{1x}, n_{1y}, -(n_{1x}p_{2x} + n_{1y}p_{2y})]\lambda$

# Euklideszi térgéometria

- Ambiens tér 4 dimenziós:  $a = [x, y, z, w]$
- Skaláris szorzás:  $a_1 \cdot a_2 = x_1x_2 + y_1y_2 + z_1z_2 + w_1w_2$
- Pontok:  $p = [x, y, z, 1]$ 
  - Távolság:  $d(p, q) = \sqrt{(q - p) \cdot (q - p)}$
  - Szög:  $\alpha(u, v) = \arccos\left(\frac{u \cdot v}{|u||v|}\right)$
- Vektorok:  $v = [x, y, z, 0]$ 
  - Hossz:  $|v| = \sqrt{v \cdot v} = \sqrt{x^2 + y^2 + z^2}$
- Síkok olyanok, mint az egyenes síkgeometriában

# Sík

- Egy  $\mathbf{p}$  pontot tartalmazó,  $\mathbf{a}$  és  $\mathbf{b}$  nem párhuzamos vektorok által kifeszített 2D altér ( $u, v$  valós paraméterek):

$$\mathbf{r}(u, v) = \mathbf{p} + \mathbf{a}u + \mathbf{b}v$$

- Legyen  $\mathbf{n}$  az  $\mathbf{a}$  és  $\mathbf{b}$  vektorokra merőleges vektor:

$$\mathbf{n} \cdot (\mathbf{r} - \mathbf{p}) = 0 \Rightarrow \mathbf{n} \cdot \mathbf{r} + d = 0, \text{ ahol } d = -\mathbf{n} \cdot \mathbf{p}$$

- Ambiens tér egy eleme:  $\mathbf{N} = [n_x, n_y, n_z, d]$

$$\mathbf{N} \cdot \mathbf{r} = [n_x, n_y, n_z, d] \cdot [x, y, z, w] =$$

$$n_x x + n_y y + n_z z + dw = 0 \quad \text{és} \quad w = 1$$

- A sík a 4D ambiens tér két 3D alterének a metszete
- Két sík megegyezik, ha  $\mathbf{N}_1 = \mathbf{N}_2 \lambda$

# Vektor/Pont/Sík/RGBA osztály

```
struct vec4 {
    float x, y, z, w; // vek: w=0; pont: w=1; sík: w=d

    vec4(float x0, float y0, float z0, float w0) {
        x = x0; y = y0; z = z0, w = w0;
    }
    vec4 operator*(float s) const {
        return vec4(x * s, y * s, z * s, w * s);
    }
    vec4 operator+(const vec4& v) const {
        return vec4(x + v.x, y + v.y, z + v.z, w + v.w);
    }
    vec4 operator-(const vec4& v) const {
        return vec4(x - v.x, y - v.y, z - v.z, w - v.w);
    }
    vec4 operator*(const vec4& v) const {
        return vec4(x * v.x, y * v.y, z * v.z, w * v.w);
    }
};
```

# vec4 műveletek

```
float dot(const vec4& a, const vec4& b) {
    return a.x * b.x + a.y * b.y + a.z * b.z + a.w * b.w;
}

float length(const vec4& v) {
    return sqrtf(dot(v, v));
}

vec4 normalize(const vec4& v) {
    return v * (1/length(v));
}

vec4 lerp(const vec4& p, const vec4& q, float t) {
    return p * (1 - t) + q * t;
}
```

# SSE, 3Dnow!

```
struct vec4 {
    float x, y, z, w;

    vec4 operator+( const vec4& v ) const {
        __declspec(align(16)) vec4 res;
        __asm {
            mov      esi, this
            mov      edi, v
            movups  xmm0, [esi]          ; unaligned
            movups  xmm1, [edi]
            addps   xmm0, xmm1
            movaps  res, xmm0          ; aligned
        }
        return res;
    }
};
```

# 3D vektor/Pont/RGB

```
struct vec3 {
    float x, y, z;

    vec3(float x0, float y0, float z0) { x = x0; y = y0; z = z0; }

    vec3 operator*(float a) { return vec3(x * a, y * a, z * a); }

    vec3 operator+(vec3& v) { // vektor, szín, pont + vektor
        return vec3(x + v.x, y + v.y, z + v.z);
    }

    vec3 operator-(vec3& v) { // vektor, szín, pont - pont
        return vec3(x - v.x, y - v.y, z - v.z);
    }

    vec3 operator*(vec3& v) { return vec3(x*v.x, y*v.y, z*v.z); }
};

float dot(const vec3& v1, const vec3& v2) {
    return (v1.x * v2.x + v1.y * v2.y + v1.z * v2.z);
}

vec3 cross(const vec3& v1, const vec3& v2) {
    return vec3( v1.y * v2.z - v1.z * v2.y,
                v1.z * v2.x - v1.x * v2.z,
                v1.x * v2.y - v1.y * v2.x);
}
```

# Euklideszi sík/tér analitikus geometriája

- Mindent számmal, ha számítógéppel nyomulunk
- Koordináták
  - Gyakran érdemes külső nézetet alkalmazni
  - Ambiens tér, extra koordináta ( $\text{pont}=1$ ,  $\text{vektor}=0$ )
  - Metrika (távolság, szög) = skaláris szorzás
- Egyenes:
  - Állandó sebességű mozgás
  - Legrövidebb út
- Implementáció: vec4
  - esetleg vec3, vec2, ha fejben tartjuk a kihagyott koordinátákat

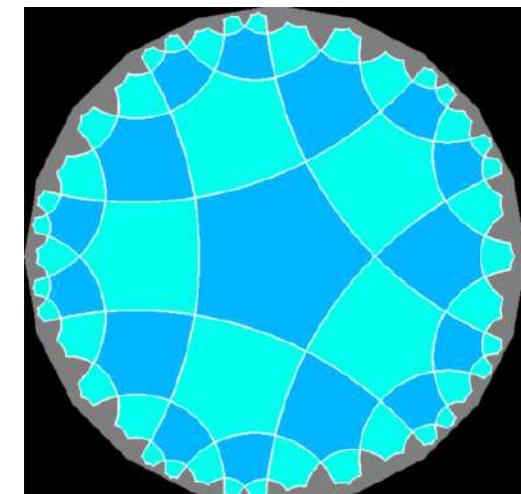
*„A paralellákat azon az úton ne próbáld; tudom  
én azt az utat is mindvégig - külön meg mértem  
azt a feneketlen éjszakát én is, az életemnek  
 minden világossága, minden öröme kialudt benne  
- az Istenre kérlek! haggy békét a paralelláknak.”*

*Bolyai Farkas*

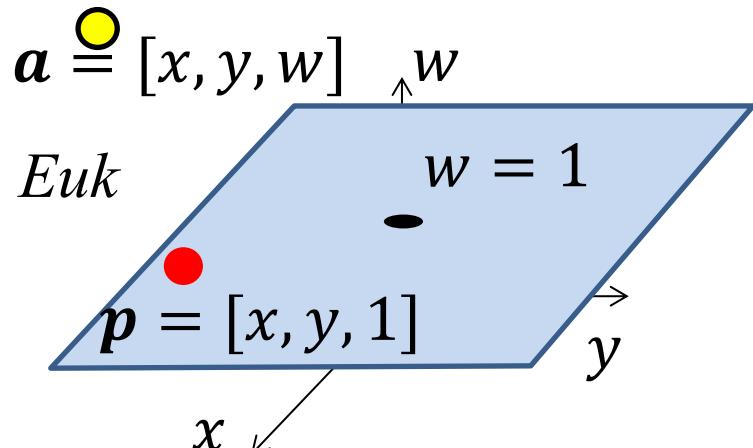
# Geometriák és algebrák

## 5. Nemeuklideszi geometriák

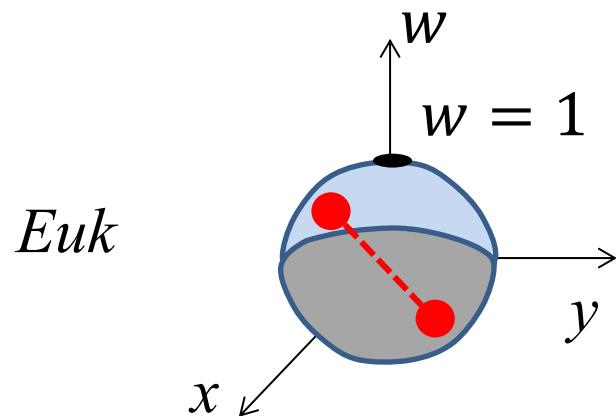
Szirmay-Kalos László



# Analitikus geometriák: külső nézet

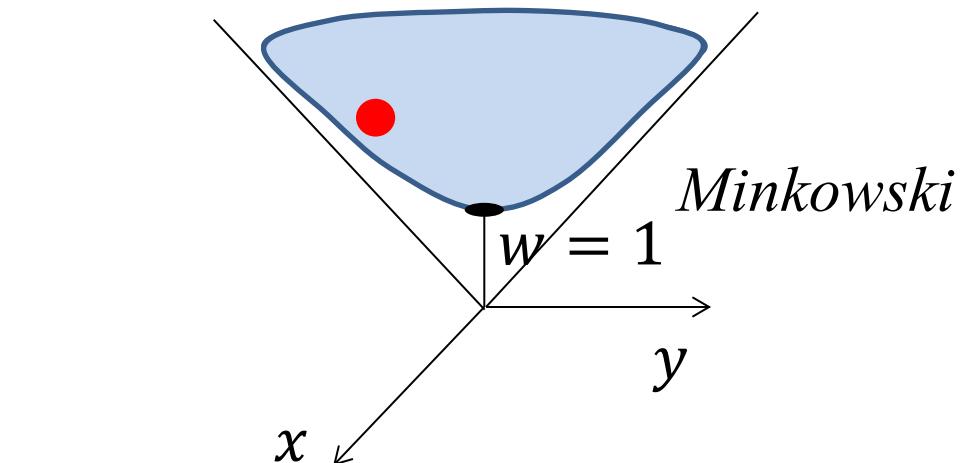
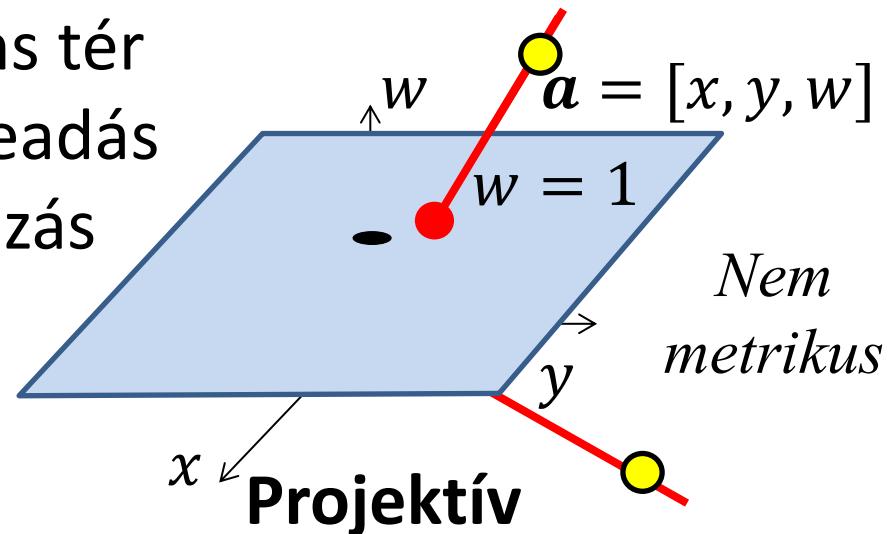


**Euklideszi:**  $w = 1$

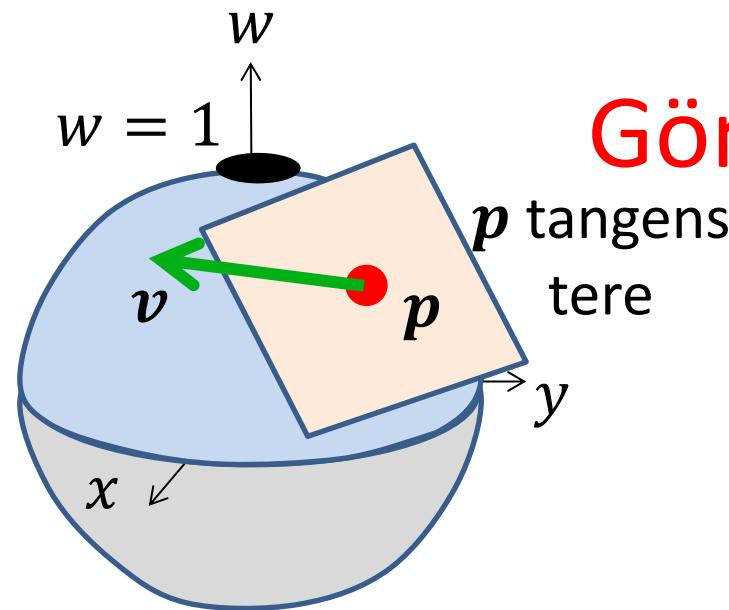


**Gömbi:**  $x^2 + y^2 + w^2 = 1$

Ambiens tér  
• Összeadás  
• Skálázás



**Hiperbolikus:**  $x^2 + y^2 - w^2 = -1$



# Gömbi (elliptikus) geometria

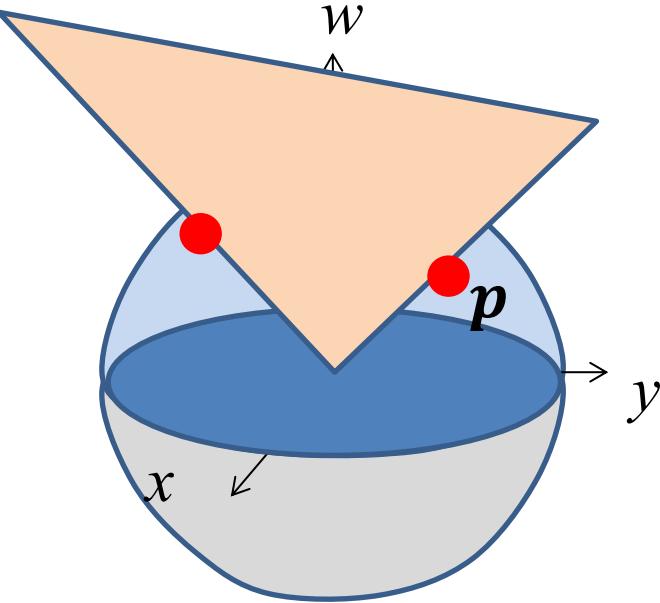
Skaláris szorzás az euklideszi ambiens térben:

$$\mathbf{a}_1 \cdot \mathbf{a}_2 = x_1 x_2 + y_1 y_2 + w_1 w_2$$

## Ambiens tér elemei:

- **Pontok:**  $p \cdot p = 1$ ,  $x^2 + y^2 + w^2 = 1$
- **Vektorok a  $p$  pontban:**  $p \cdot v = 0$  és  $v \cdot v = 1$ 
  - A vektorok mások a tér különböző pontjaiban
  - A vektor a pont tangens terének eleme
  - Csak egységvektorokkal foglalkozunk

# Egyenes



Egységebességű mozgás:

$$\mathbf{r}(t) = \mathbf{p} \cos t + \mathbf{v} \sin t$$

Gömbi kombináció (slerp):

$$\mathbf{r}(t) = \mathbf{p} \frac{\sin(1-t)d}{\sin d} + \mathbf{q} \frac{\sin td}{\sin d}$$

- Gömbön marad:

$$\mathbf{r}(t) \cdot \mathbf{r}(t) = \mathbf{p} \cdot \mathbf{p} \cos^2 t + \mathbf{v} \cdot \mathbf{v} \sin^2 t + 2\mathbf{p} \cdot \mathbf{v} \sin t \cos t = 1$$

- Egységebességű mozgás:

$$\dot{\mathbf{r}}(t) = -\mathbf{p} \sin t + \mathbf{v} \cos t$$

$$\dot{\mathbf{r}}(t) \cdot \dot{\mathbf{r}}(t) = \mathbf{p} \cdot \mathbf{p} \sin^2 t + \mathbf{v} \cdot \mathbf{v} \cos^2 t - 2\mathbf{p} \cdot \mathbf{v} \sin t \cos t = 1$$

- Ambiens tér (**origó,  $p, q$** ) síkjának és a geometria metszete
- Gauss görbület:  $\ddot{\mathbf{r}}(t) = -\mathbf{r}(t) \Rightarrow K = \ddot{\mathbf{r}}_{min} \cdot \ddot{\mathbf{r}}_{max} = 1$

# Távolság

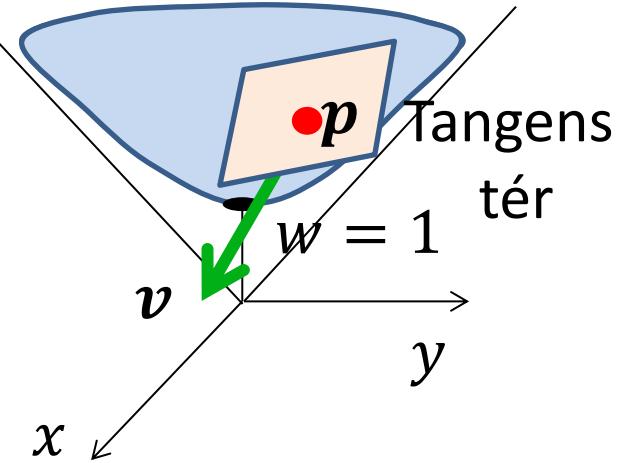
Távolság  $d(p, q)$ : két pont különbsége =  
egységsebességű egyenesvonalú mozgás ideje:

$$\mathbf{r}(d) = \mathbf{p} \cos d + \mathbf{v} \sin d = \mathbf{q}$$

$$\mathbf{r} \cdot \mathbf{p} = \mathbf{p} \cdot \mathbf{p} \cos d + \mathbf{v} \cdot \mathbf{p} \sin d = \mathbf{q} \cdot \mathbf{p}$$

$$\cos d = \mathbf{q} \cdot \mathbf{p}$$

$$d(p, q) = \arccos(\mathbf{q} \cdot \mathbf{p})$$



# Hiperbolikus geometria

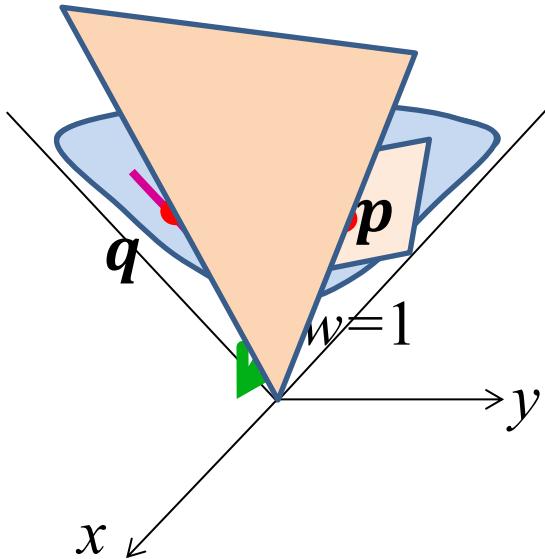
Skaláris szorzás Lorentz szorzat:

$$\mathbf{a}_1 \cdot \mathbf{a}_2 = x_1 x_2 + y_1 y_2 - w_1 w_2$$

Ambiens tér: Minkowski tér

## Ambiens tér elemei:

- Pontok:  $p \cdot p = -1$ ,  $x^2 + y^2 - w^2 = -1$  és  $w \geq 0$
- Vektorok a  $p$  pontban:  $p \cdot v = 0$  és  $v \cdot v = 1$ 
  - A vektorok mások a tér különböző pontjaiban
  - A vektor a pont tangens terének eleme
  - Csak egységvektorokkal foglalkozunk



# Egyenes

Egységebességű mozgás:

$$\mathbf{r}(t) = \mathbf{p} \cosh t + \mathbf{v} \sinh t$$

Hiperbolikus kombináció:

$$\mathbf{r}(t) = \mathbf{p} \frac{\sinh(1-t)d}{\sinh d} + \mathbf{q} \frac{\sinh td}{\sinh d}$$

- Hiperboloidon marad:

$$\mathbf{r}(t) \cdot \mathbf{r}(t) = \mathbf{p} \cdot \mathbf{p} \cosh^2 t + \mathbf{v} \cdot \mathbf{v} \sinh^2 t + 2\mathbf{p} \cdot \mathbf{v} \sinh t \cosh t = -1$$

- Egységebességű mozgás:

$$\dot{\mathbf{r}}(t) = \mathbf{p} \sinh t + \mathbf{v} \cosh t$$

$$\dot{\mathbf{r}}(t) \cdot \dot{\mathbf{r}}(t) = \mathbf{p} \cdot \mathbf{p} \sinh^2 t + \mathbf{v} \cdot \mathbf{v} \cosh^2 t + 2\mathbf{p} \cdot \mathbf{v} \sinh t \cosh t = 1$$

- Ambiens tér (**origó,  $p, q$** ) síkjának és a geometria metszete
- Gauss görbület:  $\ddot{\mathbf{r}}(t) = \mathbf{r}(t) \Rightarrow K = \ddot{\mathbf{r}}_{min} \cdot \ddot{\mathbf{r}}_{max} = -1$

# Távolság

Távolság  $d(p, q)$ : két pont különbsége =  
egységsebességű egyenesvonalú mozgás ideje:

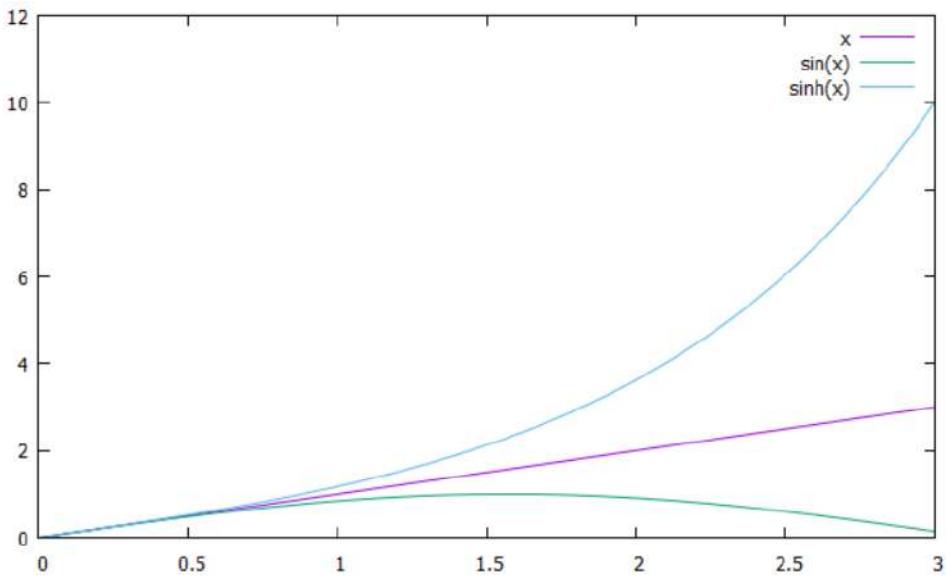
$$r(d) = p \cosh d + v \sinh d = q$$

$$r \cdot p = p \cdot p \cosh d + v \cdot p \sinh d = q \cdot p$$

$$\cosh d = -q \cdot p$$

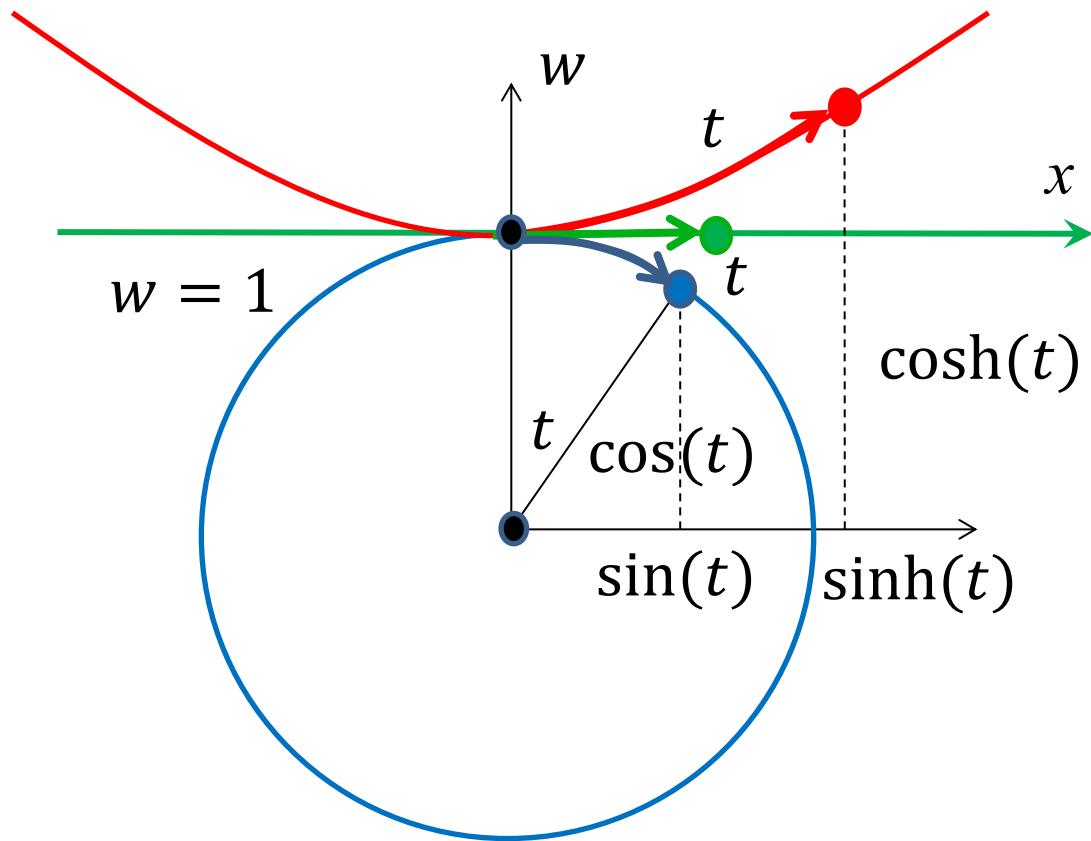
$$d(p, q) = \operatorname{arccosh}(-q \cdot p)$$

# Összehasonlítás



	Hiperbolikus	Euklideszi	Gömbi
Három szög terület	$(\alpha + \beta + \gamma - \pi)/K$		$(\alpha + \beta + \gamma - \pi)/K$
Kör	$\mathbf{p} \cdot \mathbf{c} = \cosh(r\sqrt{-K}) / K$	$(\mathbf{p} - \mathbf{c})^2 = r^2$	$\mathbf{p} \cdot \mathbf{c} = \cos(r\sqrt{K}) / K$
Kör kerület	$\frac{2\pi}{\sqrt{-K}} \sinh(r\sqrt{-K})$	$2\pi r$	$\frac{2\pi}{\sqrt{K}} \sin(r\sqrt{K})$
Kör terület	$\frac{4\pi}{-K} \sinh^2\left(\frac{r\sqrt{-K}}{2}\right)$	$\pi r^2$	$\frac{4\pi}{K} \sin^2\left(\frac{r\sqrt{K}}{2}\right)$

# Összehasonlítás



# Összehasonlítás

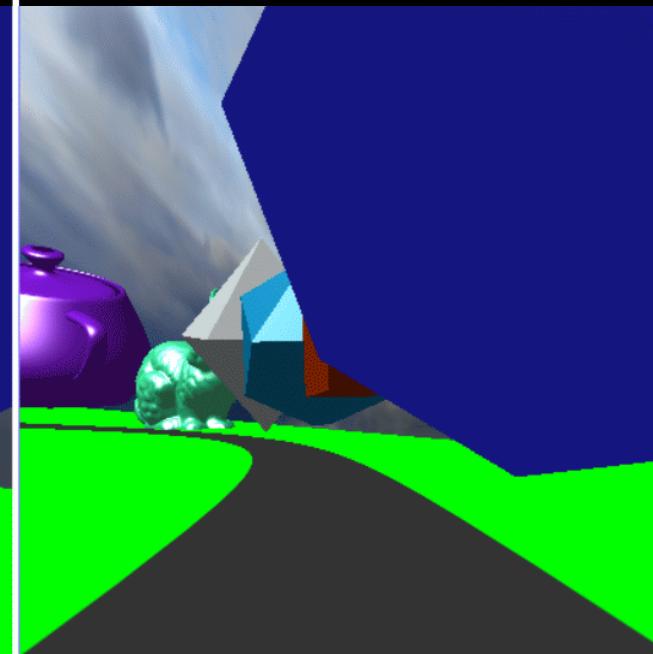
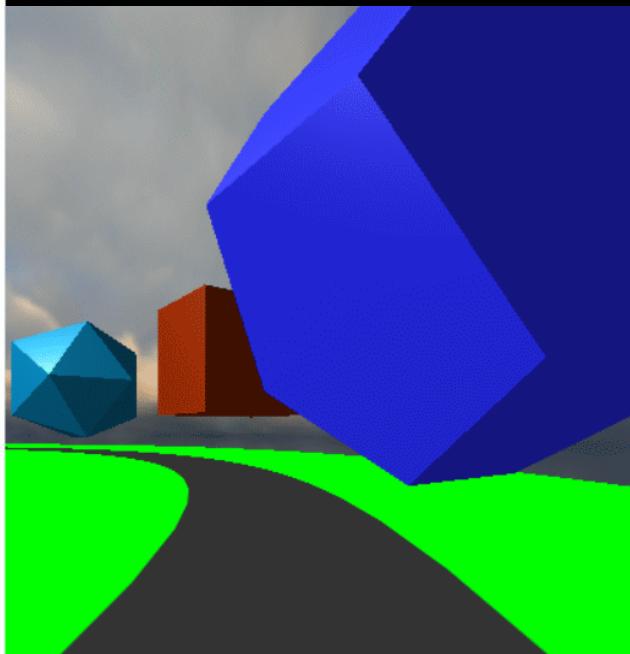
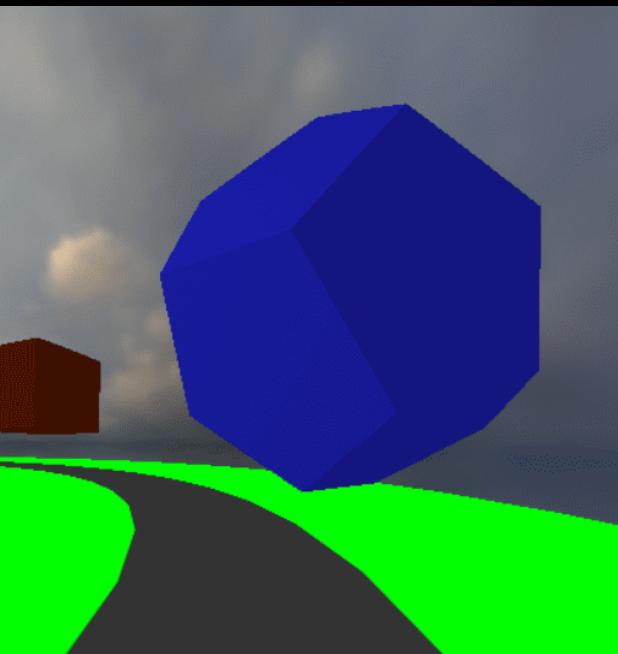
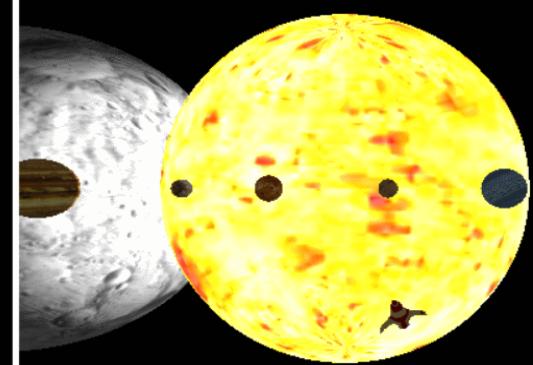
Hiperbolikus



Euklideszi



Gömbi



# Vektor/Pont/Sík/RGBA osztály

```
struct vec4 {
    float x, y, z, w; // 4D ambiens tér egy pontja

    vec4(float x0, float y0, float z0, float w0) {
        x = x0; y = y0; z = z0, w = w0;
    }

    vec4 operator*(float s) {
        return vec4(x * s, y * s, z * s, w * s);
    }

    vec4 operator+(vec4& v) {
        return vec4(x + v.x, y + v.y, z + v.z, w + v.w);
    }

    vec4 operator-(vec4& v) {
        return vec4(x - v.x, y - v.y, z - v.z, w - v.w);
    }

    vec4 operator*(vec4& v) {
        return vec4(x * v.x, y * v.y, z * v.z, w * v.w);
    }
};
```

# vec4 műveletek

```
float s = 1; // 1: Euclidean; -1: Minkowski

float dot(vec4& a, vec4& b) {
    return a.x * b.x + a.y * b.y + a.z * b.z + s * a.w * b.w;
}

float length(vec4& v) { return sqrtf(dot(v, v)); }

vec4 normalize(vec4& v) { return v * 1/length(v); }

vec4 lerp(vec4& p, vec4& q, float t) {
    return p * (1-t) + q * t;
}

vec4 slerp(vec4& p, vec4& q, float t) {
    float d = acos(dot(p, q));
    return p * (sin((1-t)*d)/sin(d)) + q * (sin(t*d)/sin(d));
}

vec4 hlerp(vec4& p, vec4& q, float t) {
    float d = acosh(-dot(p, q));
    return p*(sinh((1-t)*d)/sinh(d)) + q*(sinh(t*d)/sinh(d));
}
```

# Nem-euklideszi sík analitikus geometriája

- Ambiens koordináták
  - Metrika (távolság, szög) = skaláris szorzás, ami hiperbolikus geometriában Lorentz szorzat
  - Vektorok halmaza attól függ, hogy hol vagyunk
- Egyenes:
  - Állandó sebességű mozgás
  - Legrövidebb út
- A gömbi és hiperbolikus geometria majdnem ugyanolyan, csak néhol előjelet kell váltani és a sin-t sinh-ra kell kicserélni.

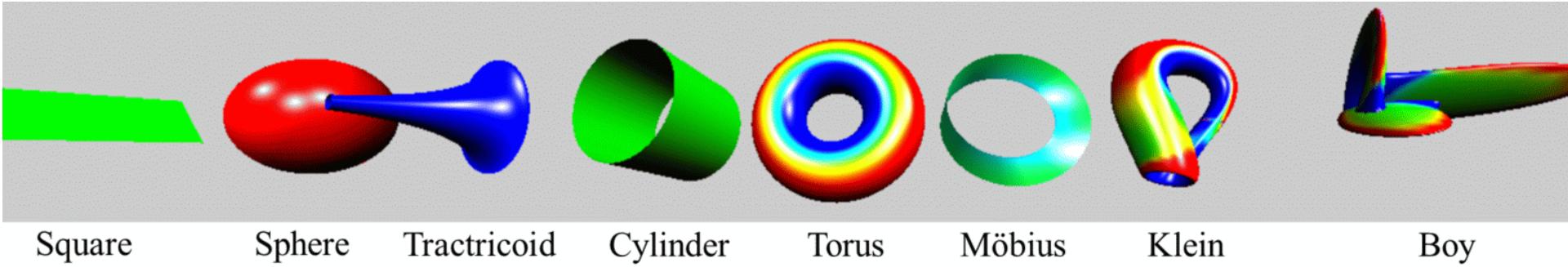
*“Point set topology is a disease from  
which the human race will soon recover.”*

*Henri Poincaré*

# Geometriák és algebrák

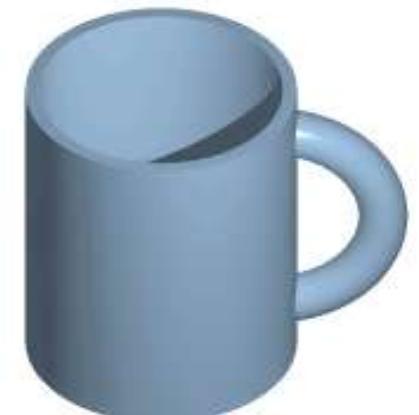
## 6. Felületek topológiája

Szirmay-Kalos László



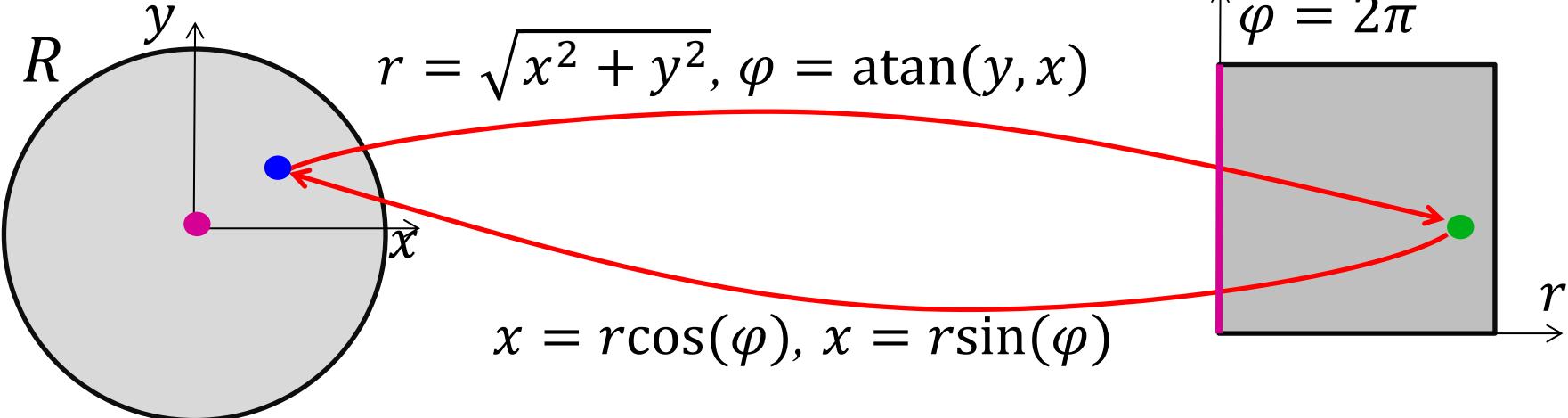
# Topológia

- Geometriai objektumok olyan tulajdonságai, amelyek megmaradnak folytonos deformáció során (invertálható folytonos függvény).
- Gyurmázás ragasztás és szakítás nélkül: **homeomorf**
- Alkalmazás:
  - Lehet a teljes objektumról (pl. gömbről) folytonos térképet csinálni?
  - **Létezhet a valóságban egy test (érvényes)?**
  - Hogyan tapétázzunk képet egy objektumra?
  - Hány ötszögből lehet focilabdát varni?
  - Hogyan nyerjünk Field medált és 1 millió USD-t és hogyan utasítsuk vissza?

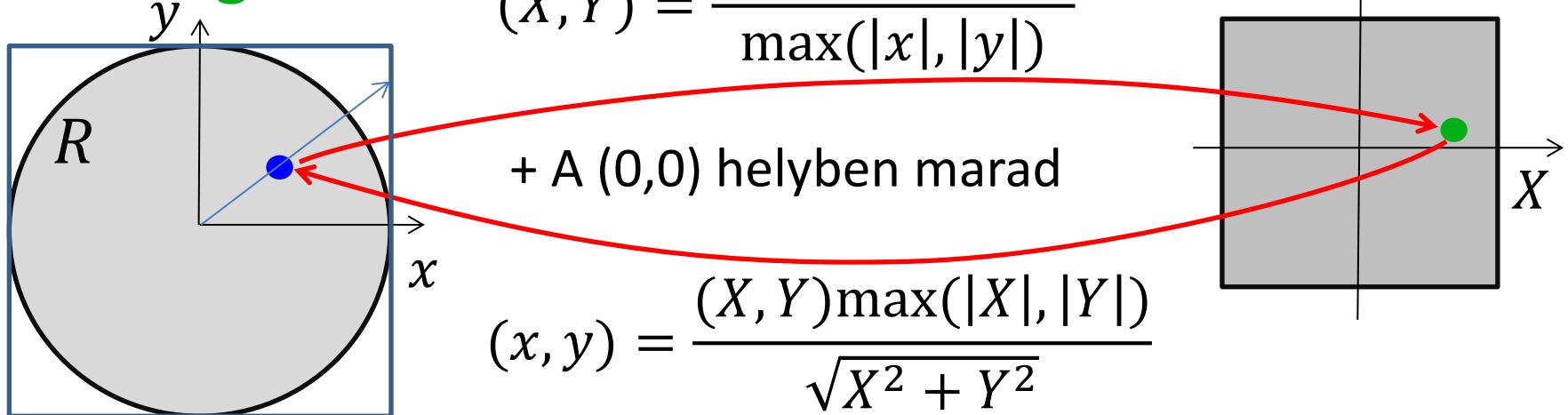


# Példa: kör és négyzet

- Rossz megoldás: polár szögek



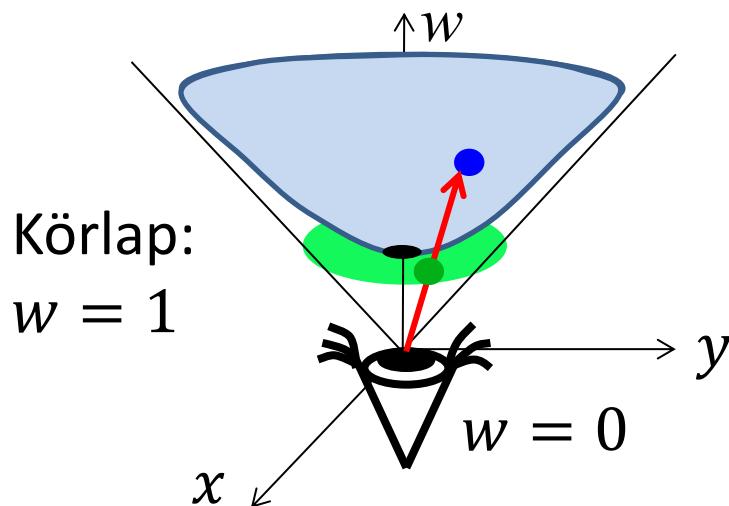
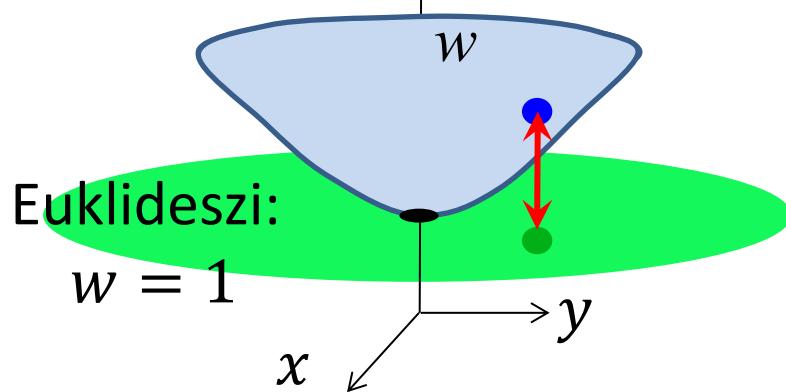
- Jó megoldás:



# Euklideszi sík, hiperbolikus sík, nyílt körlap, nyílt négyzet

Hiperbolikus:

$$x^2 + y^2 - w^2 = -1$$



Hiperbolikus  $\rightarrow$  Euklideszi:  
 $(x, y, w) \rightarrow (x, y, 1)$

Euklideszi  $\rightarrow$  Hiperbolikus:

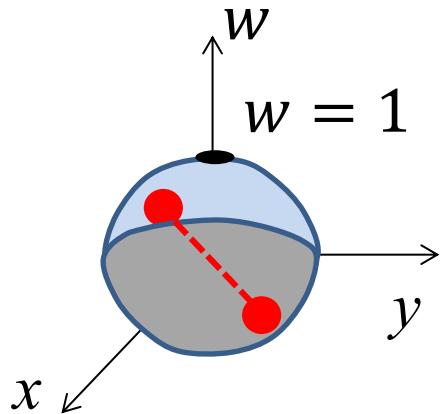
$$(x, y, 1) \rightarrow \left( x, y, \sqrt{x^2 + y^2 + 1} \right)$$

Hiperbolikus  $\rightarrow$  Körlap:  
 $(x, y, w) \rightarrow (x/w, y/w, 1)$

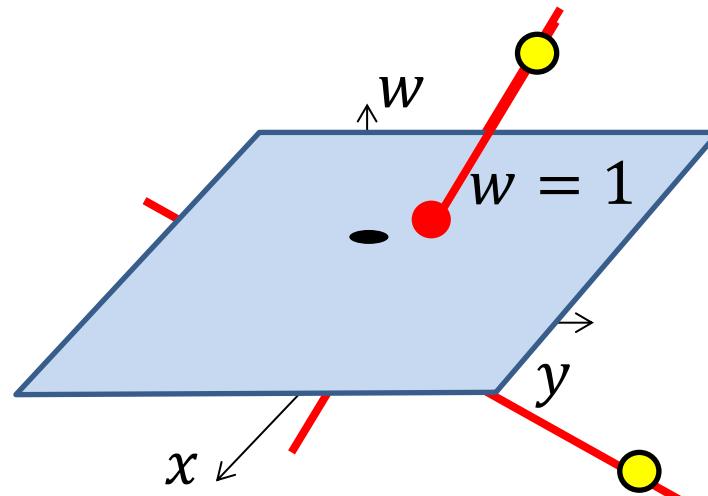
Körlap  $\rightarrow$  Hiperbolikus:

$$(x, y, 1) \rightarrow \frac{(x, y, 1)}{\sqrt{1 - x^2 - y^2}}$$

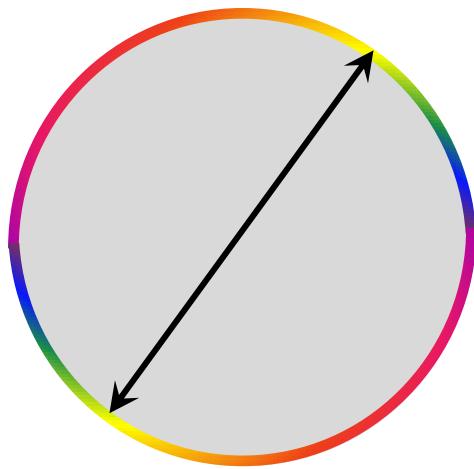
# Projektív sík, elliptikus sík, zárt körlap



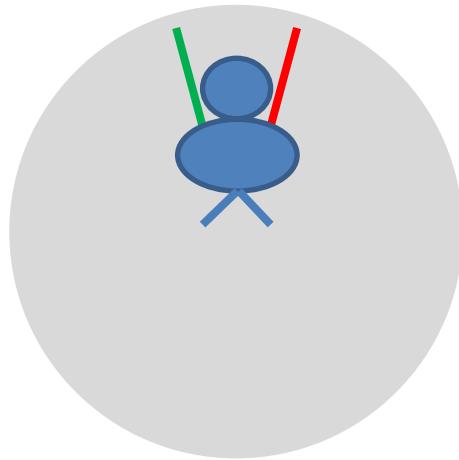
Elliptikus



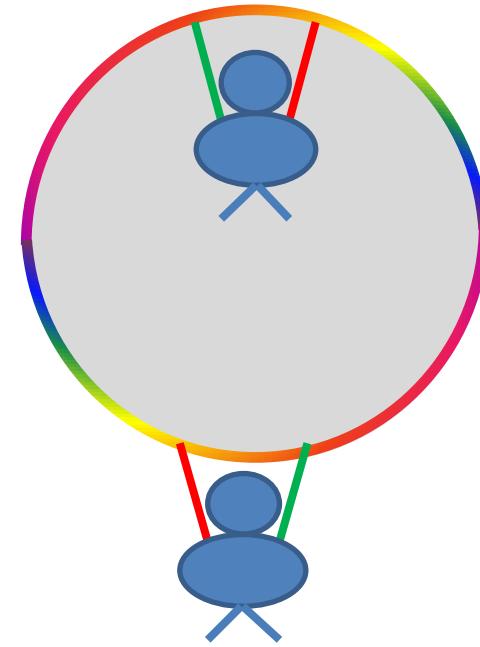
Projektív



# Irányíthatóság: jobb-bal értelmezhető

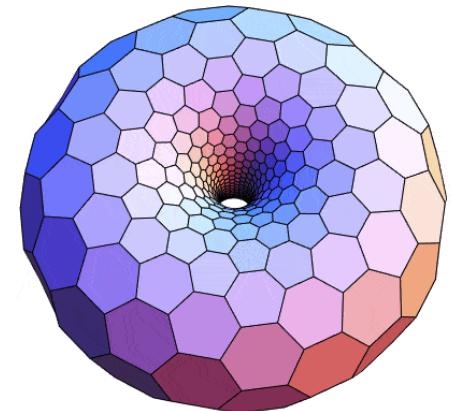
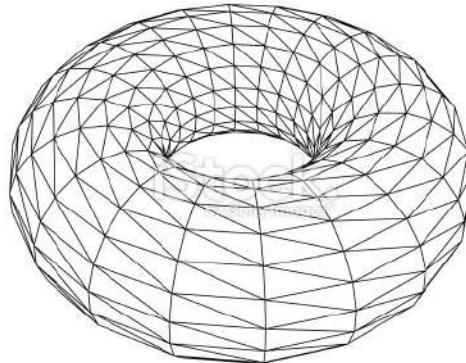
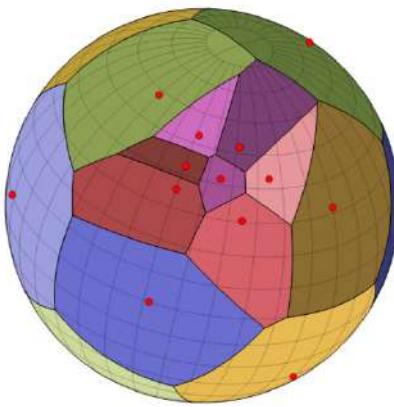


Euklideszi, Hiperbolikus  
**Irányítható**



Projektív, Elliptikus  
**Nem irányítható**

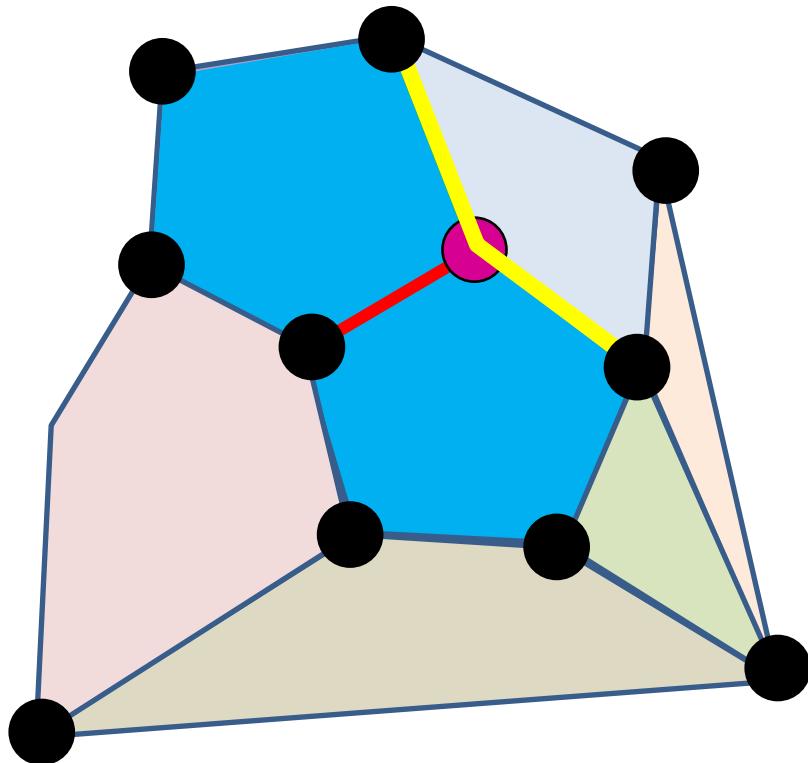
# Euler karakterisztika: $c - e + l = \chi$



- $c$ : Csúcsok száma. Csúcs = pont, amelyben élek találkoznak.
- $e$ : Élek száma. Él = folytonos vonal, amely két (nem feltétlenül különböző) csúcsban végződik. Egy él legfeljebb két lap közös határa (**határoltság**). Élek nem metszik saját magukat és egymást.
- $l$ : Lapok száma. A lapot egy él-csúcs sorozat határolja. Lapok lemezzé deformálhatók.
- Az Euler karakterisztika és a határoltság nem változik, ha a felületet deformáljuk.

# Az Euler karakterisztika invariáns

- $\chi$  csak felület topológijától függ, amelyre a csúcsokat, éleket és lapokat felrajzoltuk.



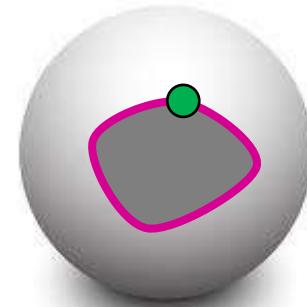
$$\begin{array}{r} c - e + l = \chi \\ \hline +1 & -1 \\ -1 & +1 \end{array}$$

# Karakterisztikák

$$\underline{c - e + l = \chi}$$

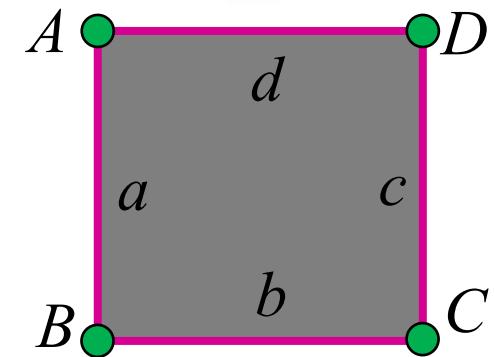
- **Gömb és a lyuk nélküli testek:**

$$\text{NH, I, } 1 - 1 + 2 = 2$$



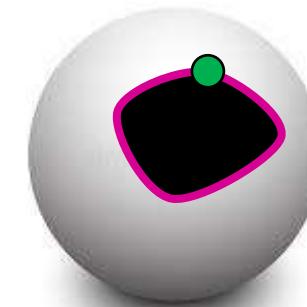
- **Lemez:** Nincs folytonos föld térkép!

$$\text{H, I, } 4 - 4 + 1 = 1$$



- **Gömb+Lyuk:**

$$\text{H, I, } 1 - 1 + 1 = 1$$

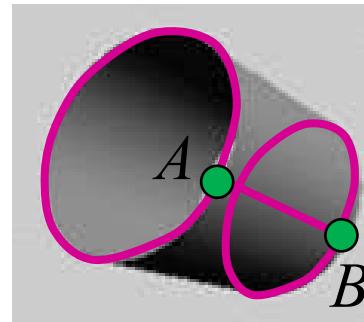
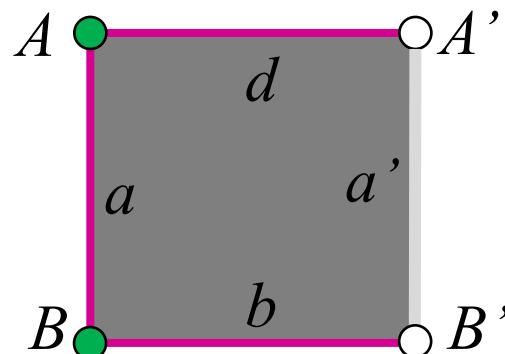


# Karakterisztikák

$$\frac{c - e + l}{\chi}$$

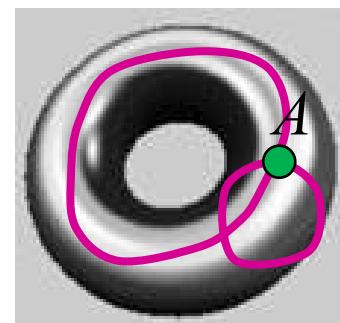
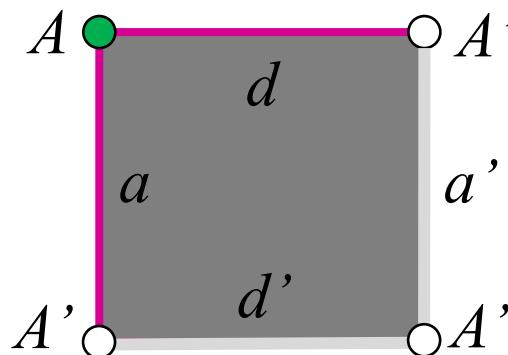
- **Henger:**

$$H, I, \quad 2 - 3 + 1 = 0$$



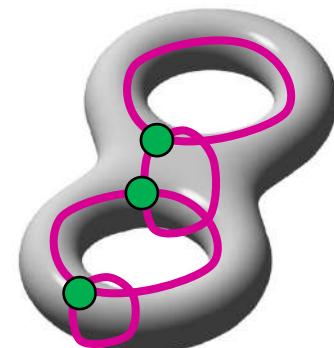
- **Tórusz + 1 testlyukas testek:**

$$NH, I, \quad 1 - 2 + 1 = 0$$



- **2 testlyukas testek:**

$$NH, I, \quad 3 - 6 + 1 = -2$$

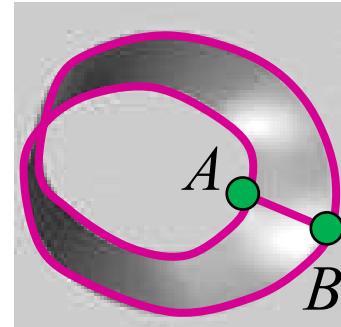
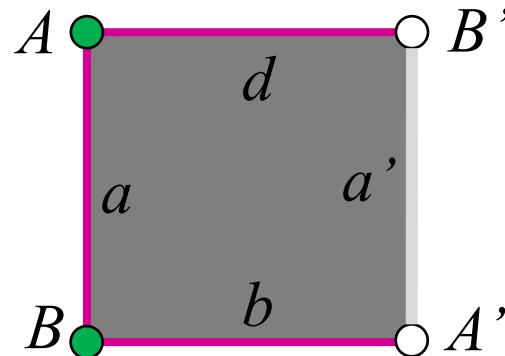


# Karakterisztikák

$$\frac{c - e + l}{\text{---}} = \chi$$

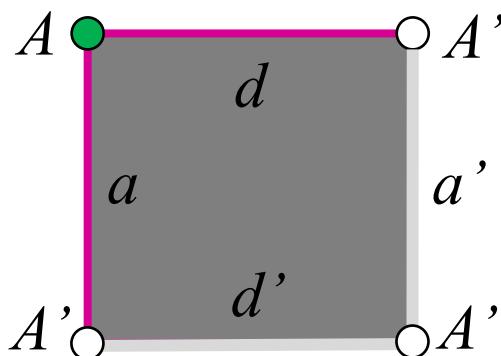
- Möbius szalag:

$$H, NI, 2 - 3 + 1 = 0$$



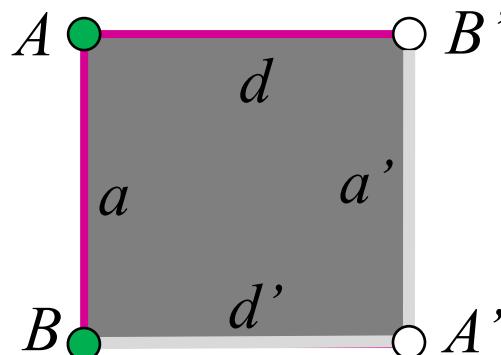
- Klein kancsó:

$$NH, NI, 1 - 2 + 1 = 0$$



- Projektív sík, Boy, Steiner:

$$NH, NI, 2 - 2 + 1 = 1$$



# Adidas labda



$$P = 12$$

$$\begin{aligned}\chi = 2 &= c - e + l = \frac{5P + 6H}{3} - \frac{5P + 6H}{2} + P + H \\ &= \frac{10P + 12H - 15P - 18H + 6P + 6H}{6} = \frac{P}{6}\end{aligned}$$

# Nem határolt, korlátos, irányítható felületek osztályozása



- 3D tartományt fog közre, a felület nem metszi magát
- Euler karakterisztika (Euler-Poincaré):

$$\chi = c - e + l = 2(n - g)$$

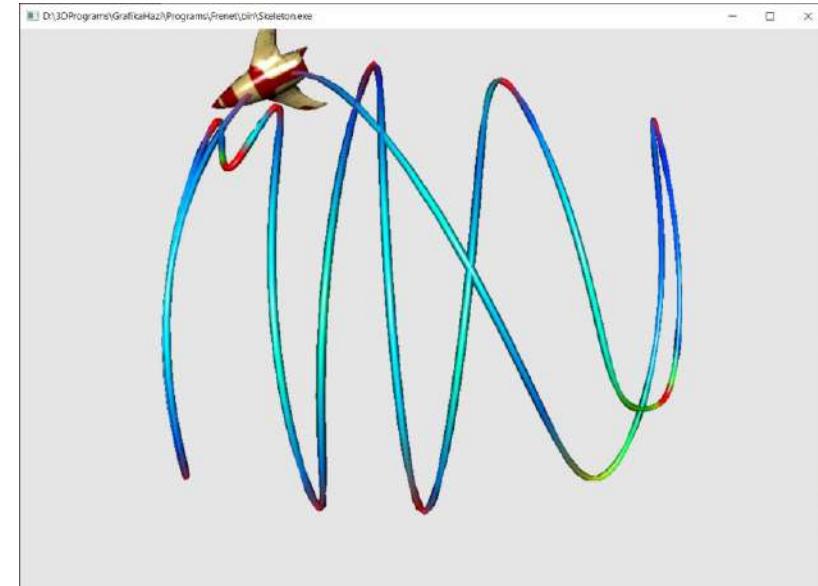
- $n$ : A különálló darabok;  $g$ : testlyukak száma
- Euler karakterisztika  $\cdot 2\pi =$  teljes görbület (Descartes)

*“Navigare necesse est,  
vivere non est necesse.”*

*Cnaeus Pompeius Magnus*

# Automatikus deriválás

Szirmay-Kalos László



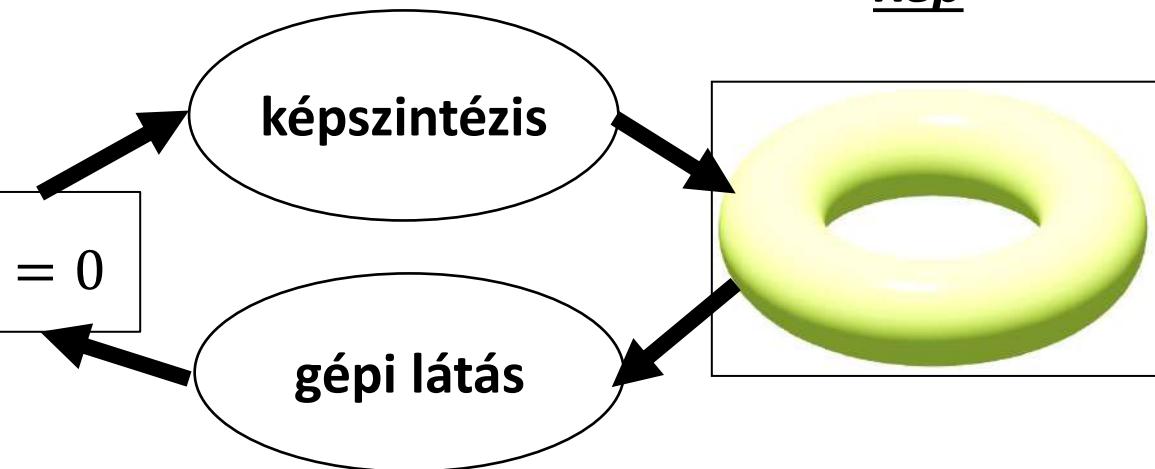
# Inverz feladatok

## Virtuális világ modell

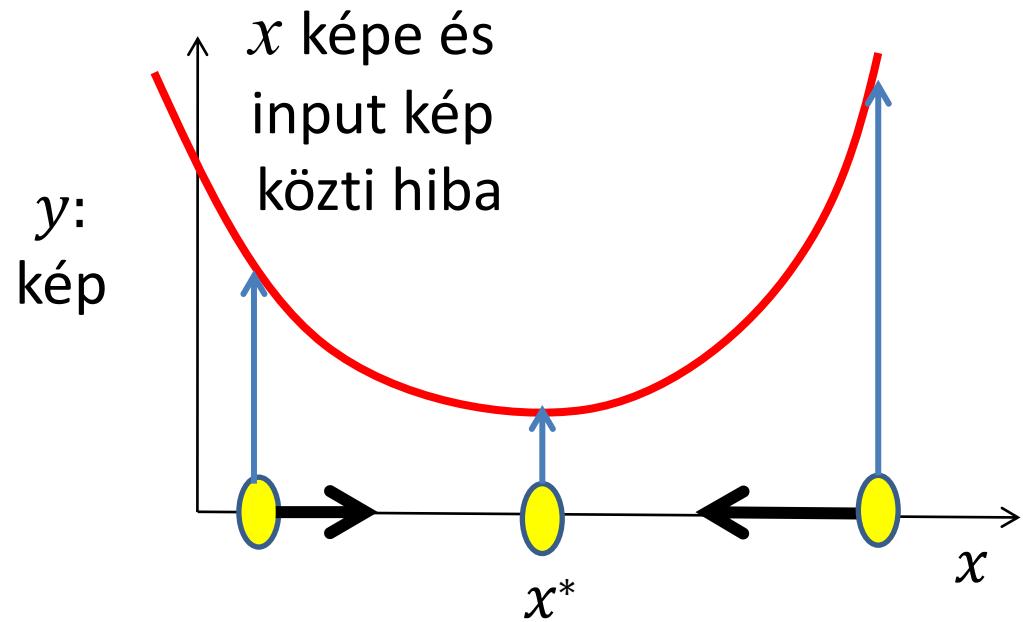
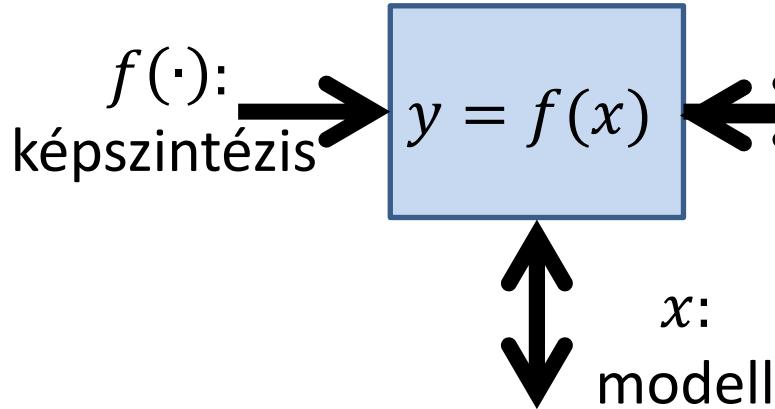
Tórusz  $R, \rho$  sugarakkal

$$(R - \sqrt{x^2 + y^2})^2 + z^2 - \rho^2 = 0$$

Kép



## Függvény inverz

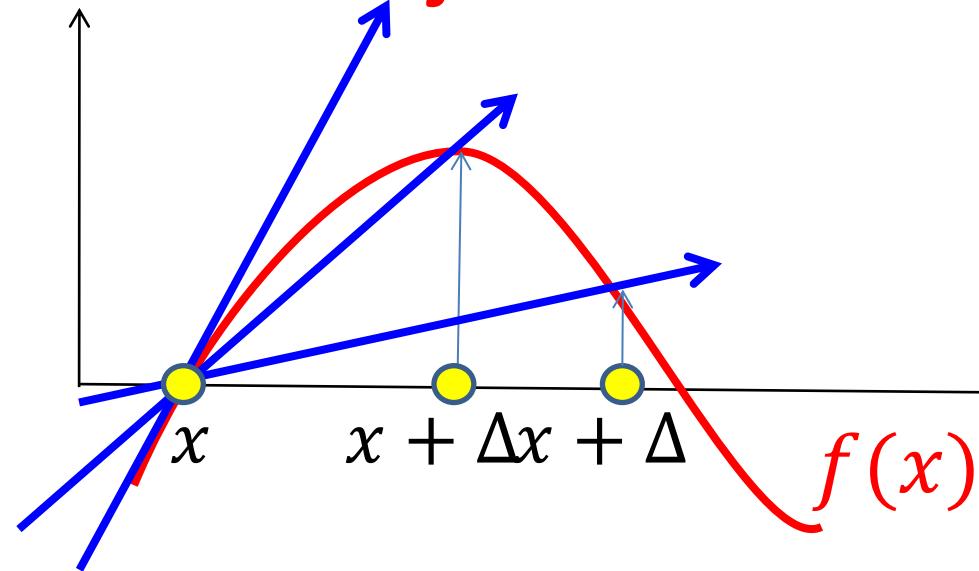


# Hogyan NE deriváljunk!

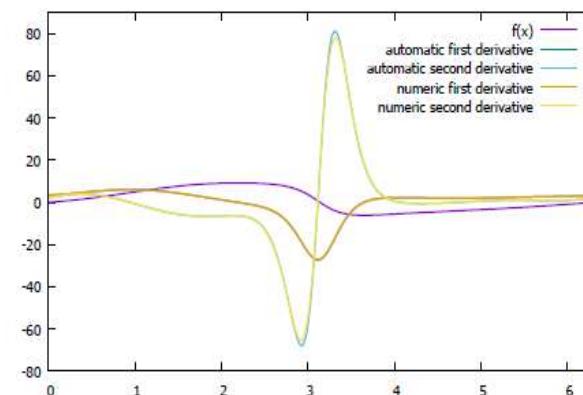
$$f'(x) \approx \frac{f(x + \Delta) - f(x)}{\Delta}$$

Kivonás:  

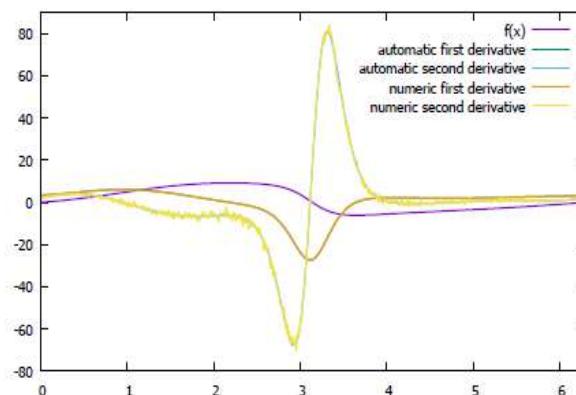
$$\begin{array}{r} 1234.569????? \\ -1234.567????? \\ \hline 0.002????? \end{array} \quad \begin{array}{l} 7 \text{ értékes jegy} \\ 7 \text{ értékes jegy} \\ 1 \text{ értékes jegy} \end{array}$$



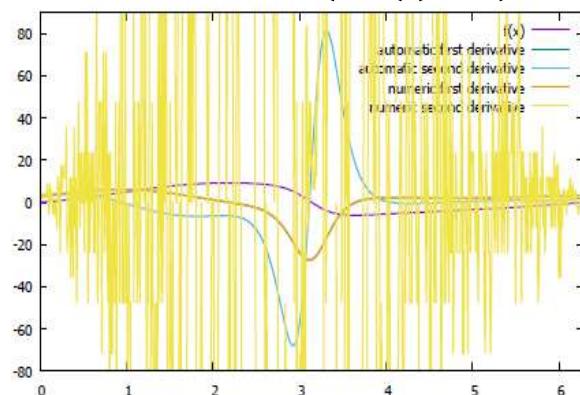
$$\frac{\sin(t)(\sin(t) + 3)4}{\tan(\cos(t) + 2)}$$



$$\Delta = 0.1$$



$$\Delta = 0.001$$



$$\Delta = 0.0001$$



# (William) Clifford algebra: Hiperszám

- Tanítsuk meg a C++-t deriválni (is)!

függvény derivált

- Hiperszám:  $z = x + yi$ , ahol

$-i^2 = -1$ : komplex szám;  $i^2 = 1$  : hiperbolikus szám;

$-i^2 = 0$ : duális szám, a deriváláshoz ez kell

összeg/  
különbség

függvény össz/kül

össz/kül deriváltja

$$(x_1 + y_1 i) \pm (x_2 + y_2 i) = (x_1 \pm x_2) + (y_1 \pm y_2) i$$

szorzat

függvény szorzat

szorzat deriváltja

$$(x_1 + y_1 i) \cdot (x_2 + y_2 i) = (x_1 x_2) + (x_1 y_2 + y_1 x_2) i + (y_1 y_2) i^2$$

hányados

függvény hányados

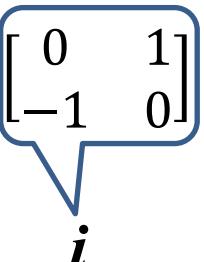
hányados  
deriváltja

$$\frac{x_1 + y_1 i}{x_2 + y_2 i} = \frac{(x_1 + y_1 i)(x_2 - y_2 i)}{(x_2 + y_2 i)(x_2 - y_2 i)} = \frac{x_1 x_2 + (y_1 x_2 - x_1 y_2) i - (y_1 y_2) i^2}{x_2^2 - y_2^2 i^2} = \frac{x_1}{x_2} + \frac{y_1 x_2 - x_1 y_2}{x_2^2} i$$

# Mi az az $i$ ?

## Komplex szám

$$z = x + yi$$

$$z = \begin{bmatrix} x & y \\ -y & x \end{bmatrix} = x \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + y \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$


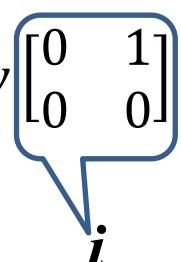
$i$

$$i \cdot i = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = -\mathbf{1} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Lehet osztani  $i$ -vel.

## Duális szám

$$z = x + yi$$

$$z = \begin{bmatrix} x & y \\ 0 & x \end{bmatrix} = x \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + y \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$


$i$

$$i \cdot i = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \mathbf{0} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Nem lehet osztani  $i$ -vel.

# Duális szám osztály

```
struct Dnum {  
    float f, d; // function and derivative values  
  
    Dnum(float f0, float d0 = 0) { // constant' = 0  
        f = f0, d = d0;  
    }  
  
    Dnum operator+(Dnum r) { return Dnum(f + r.f, d + r.d); }  
    Dnum operator-(Dnum r) { return Dnum(f - r.f, d - r.d); }  
    Dnum operator*(Dnum r) {  
        return Dnum(f * r.f, f * r.d + d * r.f);  
    }  
    Dnum operator/(Dnum r) {  
        return Dnum(f / r.f, (d * r.f - f * r.d) / r.f / r.f);  
    }  
};
```

# Duális szám alkalmazása

- Deriválás nélkül:

```
float t = value;  
float F = t * a / (t * t + b);
```

- Deriválással együtt:

```
Dnum F = Dnum(t,1) * Dnum(a,0) /  
        (Dnum(t,1) * Dnum(t,1) + Dnum(b,0));
```

- Deriválással együtt szebben, kihasználva a default paraméterezést a konstruktorban:

```
Dnum t(value, 1);  
Dnum F = t * a / (t * t + b);
```

# Elemi függvény

```
float F, x, y, a;  
...  
F = 3 * t + a * sin(t);
```

```
struct Dnum {  
    float f, d; // function and derivative values  
    Dnum(float f0, float d0 = 0) { f = f0, d = d0; }  
    ...  
};  
  
Dnum Sin(float t) { return Dnum(sinf(t), cosf(t)); }  
Dnum Cos(float t) { return Dnum(cosf(t), -sinf(t)); }  
...
```

# Összetett függvény

```
float F, x, y, a;
```

```
...
```

```
F = 3 * t + a * sin(t) + cos(y * log(t) + 2);
```

$$\frac{df(g(t))}{dt} = \frac{df}{dg} \cdot \frac{dg}{dt}$$

```
struct Dnum {
    float f, d; // function and derivative values
    Dnum(float f0, float d0 = 0) { f = f0, d = d0; }
    ...
};

Dnum Sin(Dnum g) { return Dnum(sinf(g.f), cosf(g.f) * g.d); }
Dnum Cos(Dnum g) { return Dnum(cosf(g.f), -sinf(g.f) * g.d); }
Dnum Tan(Dnum g) { return Sin(g)/Cos(g); }
Dnum Log(Dnum g) { return Dnum(logf(g.f), 1/g.f * g.d); }
Dnum Exp(Dnum g) { return Dnum(expf(g.f), expf(g.f) * g.d); }
Dnum Pow(Dnum g, float n) {
    return Dnum(powf(g.f, n), n * powf(g.f, n - 1) * g.d);
}
```

# Többváltozós függvények

```
template<class T> struct Dnum {
    float f; // function value
    T d; // derivatives
    Dnum(float f0, T d0 = T(0)) { f = f0, d = d0; }
    Dnum operator+(Dnum r) { return Dnum(f+r.f, d+r.d); }
    Dnum operator*(Dnum r) { return Dnum(f*r.f, f*r.d + d*r.f); }
    Dnum operator/(Dnum r) {
        return Dnum(f / r.f, (d * r.f - f * r.d) / r.f / r.f);
    }
};

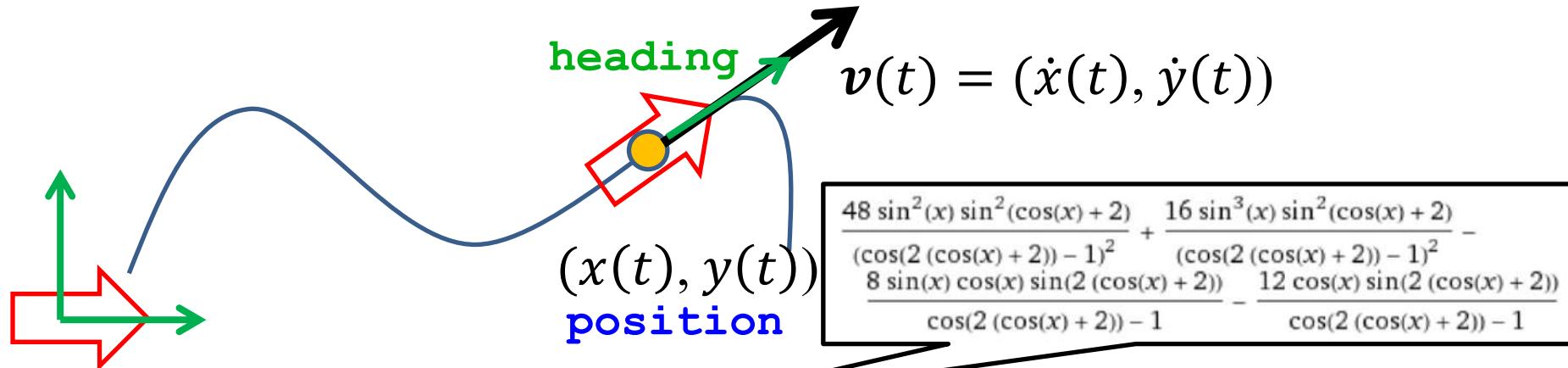
template<class T> Dnum<T> Exp(Dnum<T> g) {
    return Dnum<T>(expf(g.f), expf(g.f) * g.d);
}
```

## F(x,y,z) gradiense:

```
float x, y, z;
Dnum<vec3> X(x, vec3(1,0,0)), Y(y, vec3(0,1,0)), Z(z, vec3(0,0,1));
Dnum<vec3> F = X*X/a + Y*Y/b + Z*Z/c - 1;
vec3 grad = F.d;
```



# Mire jó? Pálya animáció



Pálya:  $x(t) = \frac{\sin(t)(\sin(t)+3)^4}{\tan(\cos(t)+2)}$ ,  $y(t) = \frac{(\cos(\sin(t))^8+1)^{12+2}}{(\sin(t)\sin(t))^3+2}$



```
void Animate(float tt) {
    Dnum t(tt, 1);
    Dnum x = Sin(t)*(Sin(t)+3)*4 / (Tan(Cos(t))+2);
    Dnum y = (Cos(Sin(t)*8+1)*12+2) / (Pow(Sin(t)*Sin(t), 3)+2);
    vec2 position(x.f, y.f), velocity(x.d, y.d);
    vec2 heading = normalize(velocity);
    Draw(position, heading);
}
```

# Automatikus deriválás

- Komplex számhoz hasonló osztály (duális szám, Clifford algebra)
  - Valós rész a függvényérték
  - Imaginárius rész, a derivált értéke
- Operátor overload-dal a deriválási szabályok
- Elég csak a függvényt leprogramozni, a derivált automatikusan számolódik

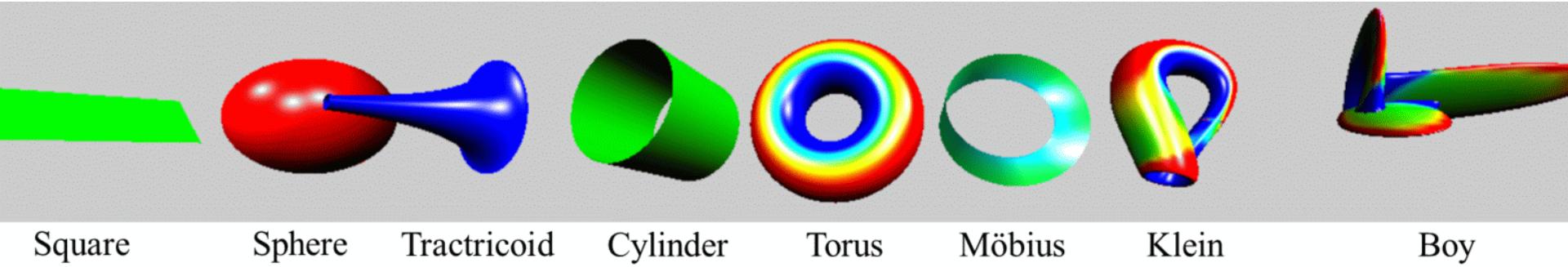
*“Wahrlich es ist nicht das Wissen, sondern das Lernen, nicht das Besitzen, sondern das Erwerben, nicht das Da-Seyn, sondern das Hinkommen, was den grössten Genuss gewährt.”*

*Carl Friedrich Gauß*

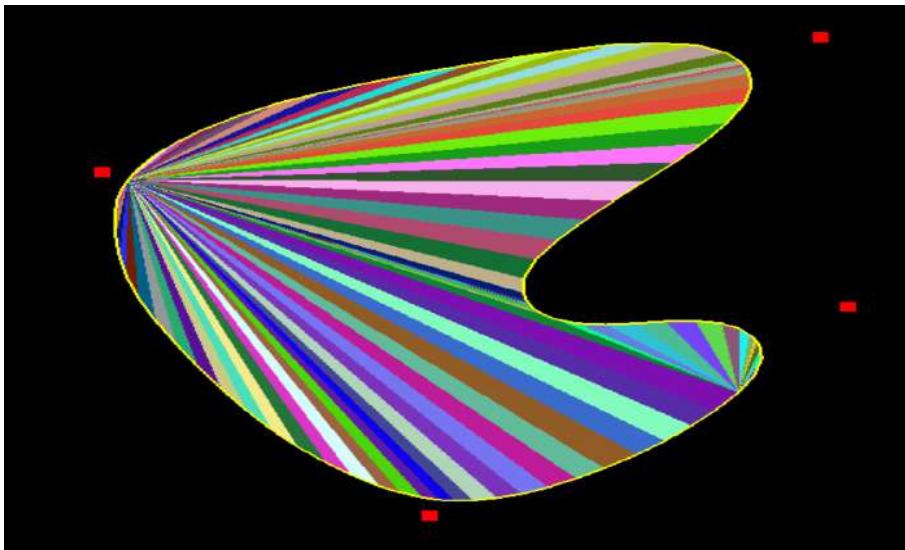
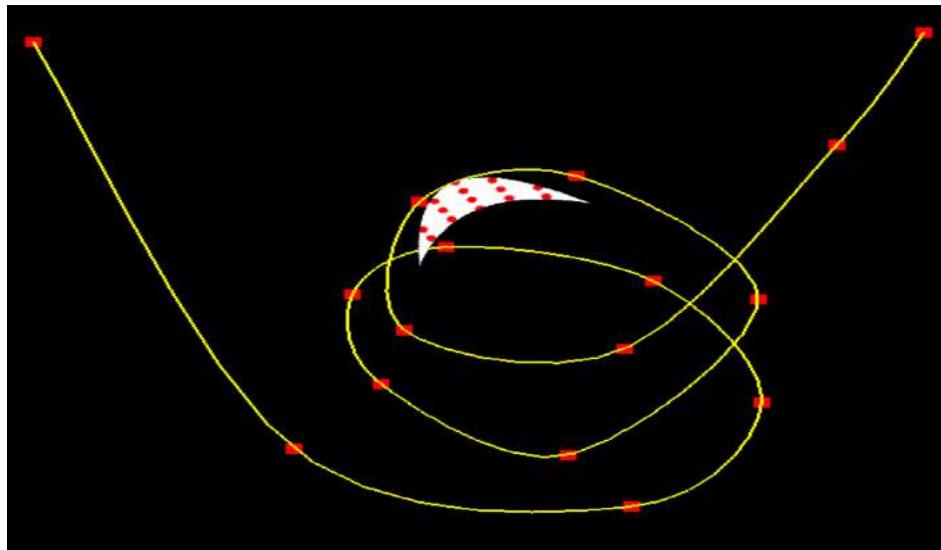
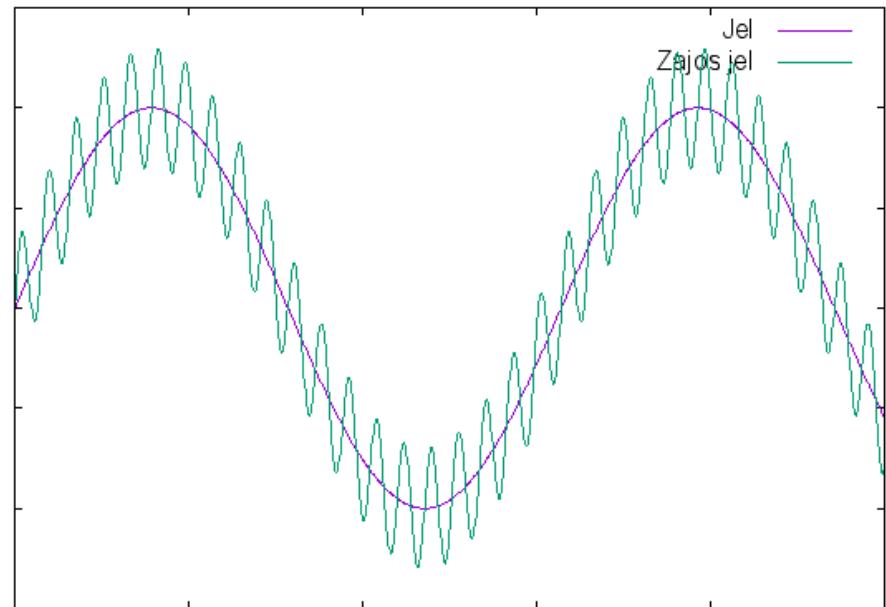
# Geometriák és algebrák

## 3. Differenciálgeometria

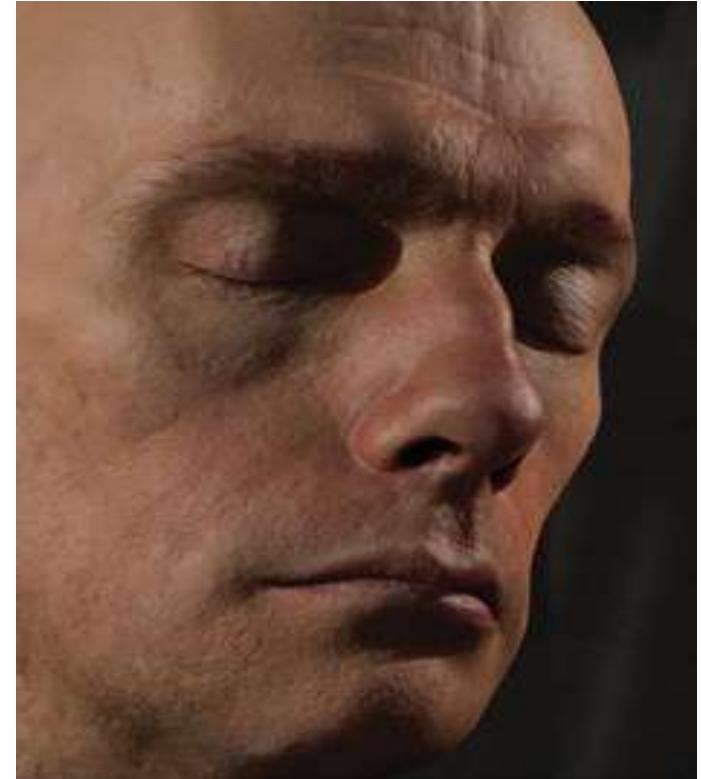
Szirmay-Kalos László



# Mire jó: Szimuláció, szűrés, ...



# Mire jó: árnyalás, normál vektor



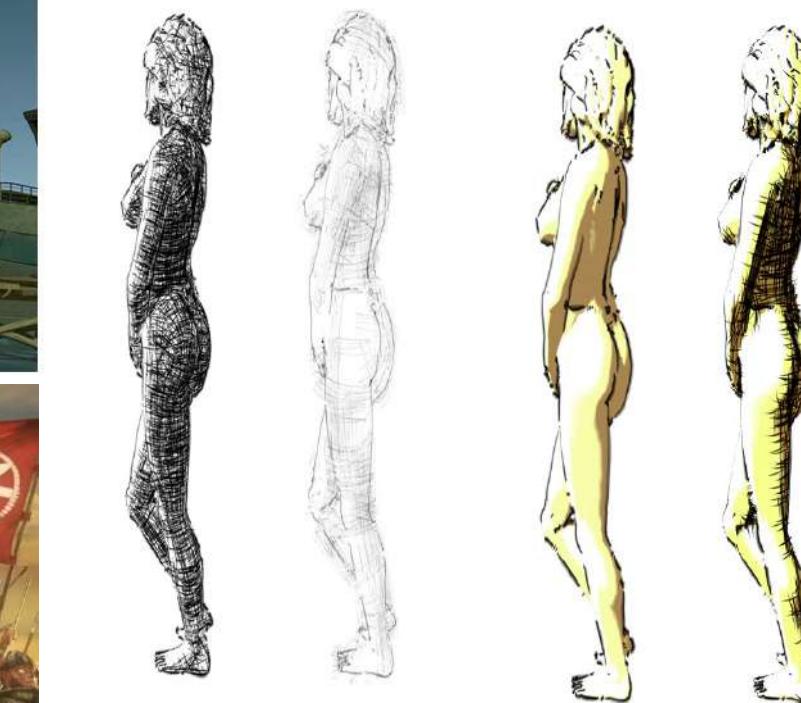
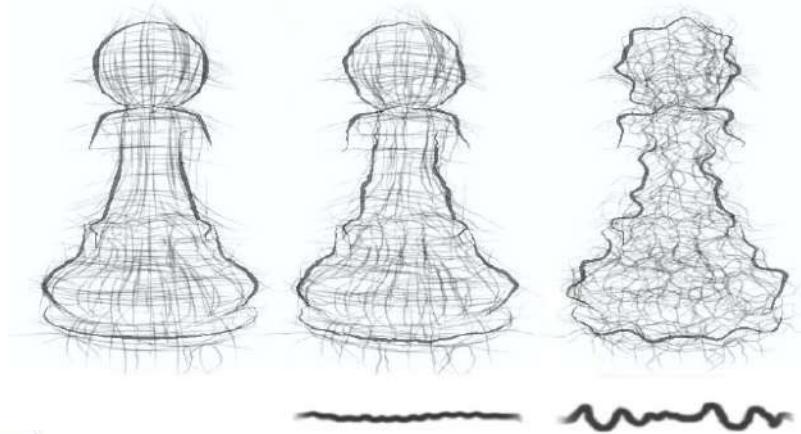
# Mire jó: Illusztratív képszintézis



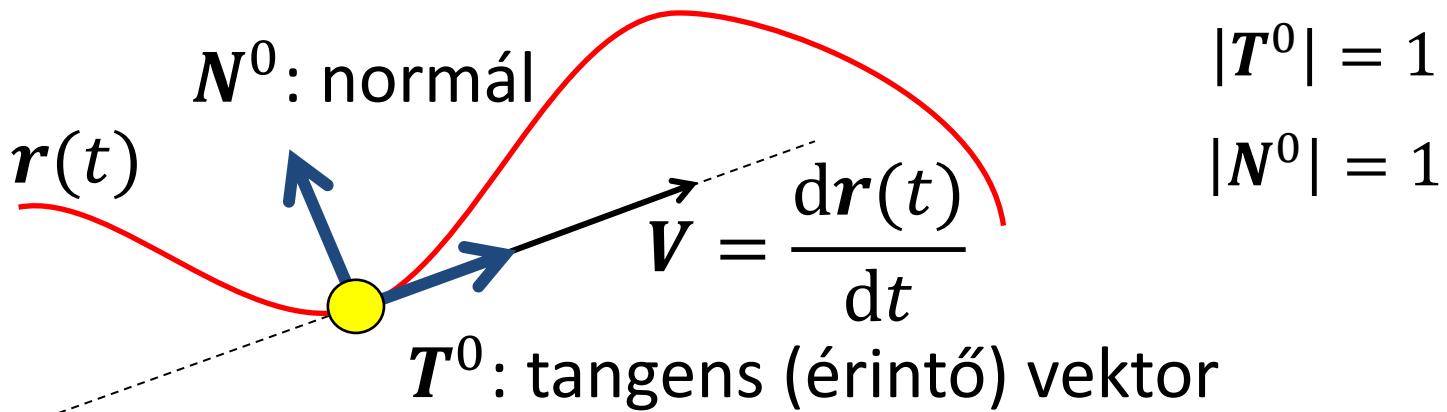
Greenscreen RGBZ footage (zLense Rig)



Shaded Scene



# Síkgörbék érintője és normálvektora



$$|T^0| = 1$$

$$|N^0| = 1$$

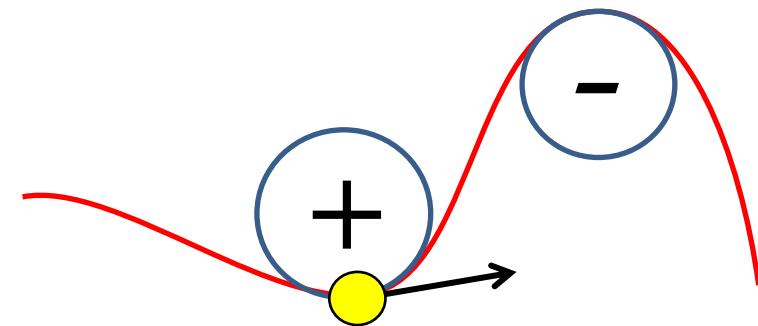
$T^0$ : tangens (érintő) vektor

- A **görbe = mozgás**, azaz a hely időfüggvénye, de a dinamika nem érdekes, ezért valós idő helyett más növekvő paramétert is jó, ami lényegében a hely pályamenti koordinátája.
- Az **érintő iránya** a sebességvektor, ami a pálya időszerinti **első deriváltja**, azaz elsőrendű közelítés (egyenes):

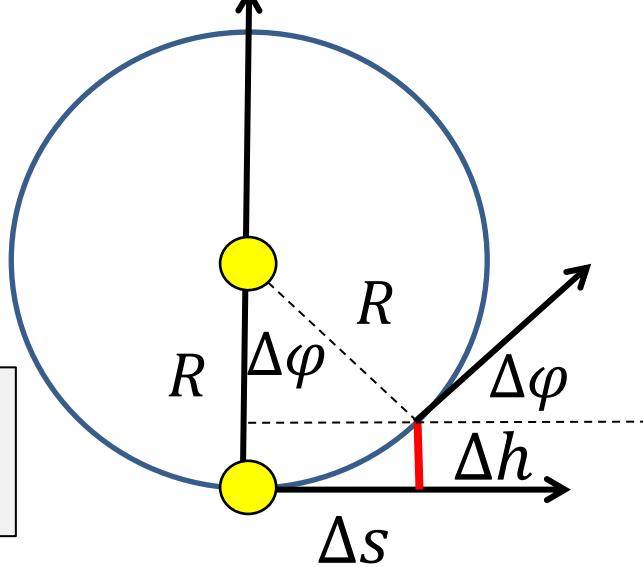
$$\mathbf{r}(t + \Delta t) \approx \mathbf{r}(t) + \mathbf{r}'(t)\Delta t$$

- Normálvektor érintőre merőleges:  $(N_x, N_y) = (-T_y, T_x)$ .
- Gyorsulás normál irányú, ha tangenciális zérus, azaz  $|\mathbf{v}| = \text{állandó}$ .

# Görbék görbülete: $\kappa$



$$\kappa = \frac{1}{R} = \frac{2\Delta h}{\Delta s^2}$$



- Egységsebességű mozgás centripetális gyorsulás:  $a_{cp} = \frac{v^2}{R}$
- (Másodrendben) simulókör sugarának reciproka
- Az érintő elfordulása kis lépésnél:  $\Delta\varphi \approx \sin(\Delta\varphi) = \frac{\Delta s}{R}$
- Érintőtől távolodás kis lépésnél:

$$\Delta h = R - \sqrt{R^2 - \Delta s^2} \Rightarrow \Delta h(0) = 0, \quad \Delta h'(0) = 0, \quad \Delta h''(0) = \frac{1}{R}$$

$$\Delta h = \Delta h(0) + \Delta h'(0)\Delta s + \frac{\Delta h''(0)}{2}\Delta s^2 + o(\Delta s^2) = \frac{\Delta s^2}{2R} + o(s^2)$$

# Görbület számítás

- Másodrendű Taylor közelítés:

$$\mathbf{r}(\tau + \Delta\tau) \approx \mathbf{r}(\tau) + \mathbf{r}'(\tau)\Delta\tau + \frac{\mathbf{r}''(\tau)}{2}\Delta\tau^2$$

- Lépéshossz négyzet (**metrika**):

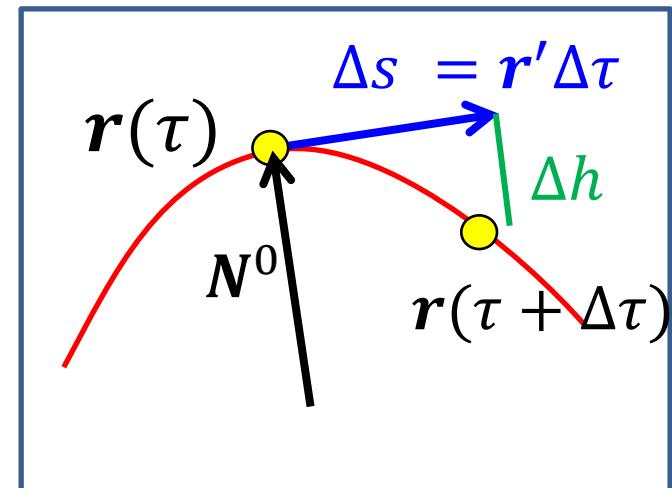
$$\Delta s^2 = |\mathbf{r}(\tau + \Delta\tau) - \mathbf{r}(\tau)|^2 = \mathbf{r}'^2 \Delta\tau^2 + o(\Delta\tau^2)$$

- Merőleges eltávolodás az érintőtől

$$\Delta h \approx N^0 \cdot \frac{\mathbf{r}''(\tau)}{2} \Delta\tau^2$$

- Görbület:

$$\kappa = \frac{2\Delta h}{\Delta s^2} = \frac{N^0 \cdot \mathbf{r}''}{\mathbf{r}'^2}$$



Metrikus tensor  
(I. fundamentális forma)

II. fundamentális forma

I. fundamentális forma

# Példa: Kör görbülete

- Parametrikus egyenlet:

$$\mathbf{r}(\tau) = (c_x + R \cos(\tau), c_y + R \sin(\tau))$$

- Első és második derivált:

$$\mathbf{r}'(\tau) = (-R \sin(\tau), R \cos(\tau)),$$

$$\mathbf{r}''(\tau) = (-R \cos(\tau), -R \sin(\tau)),$$

- Metrikus tenzor:

$$|\mathbf{r}'|^2 = \mathbf{r}' \cdot \mathbf{r}' = R^2$$

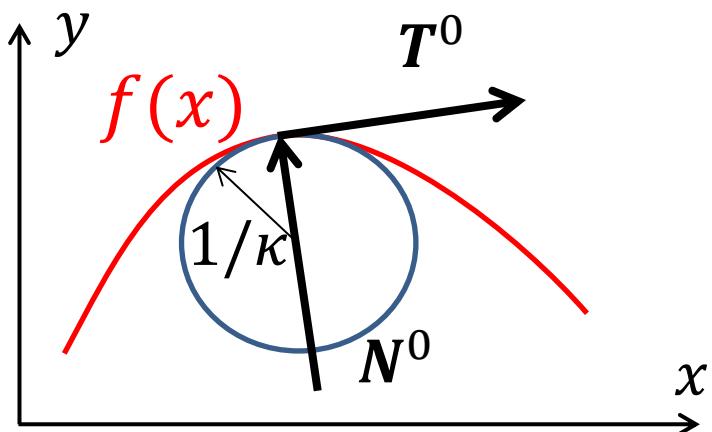
- Egységsebességű mozgás sebesség és normálvektora:

$$\mathbf{T}^0 = \frac{\mathbf{r}'}{|\mathbf{r}'|} = (-\sin(\tau), \cos(\tau)), \quad \mathbf{N}^0 = (-\cos(\tau), -\sin(\tau))$$

- Görbület:

$$\kappa = \frac{\mathbf{r}'' \cdot \mathbf{N}^0}{|\mathbf{r}'|^2} = \frac{(-R \cos(\tau), -R \sin(\tau)) \cdot (-\cos(\tau), -\sin(\tau))}{R^2} = 1/R$$

# Explicit egyenletű görbüék



Visszavezetés parametrikus esetre:

- $\tau = x$
- $\mathbf{r}(\tau) = (x, f(x))$
- $\mathbf{r}'(\tau) = (1, f')$
- $|\mathbf{r}'| = \sqrt{1 + f'^2}$
- $\mathbf{r}''(\tau) = (0, f'')$

- Érintő és normálvektor:

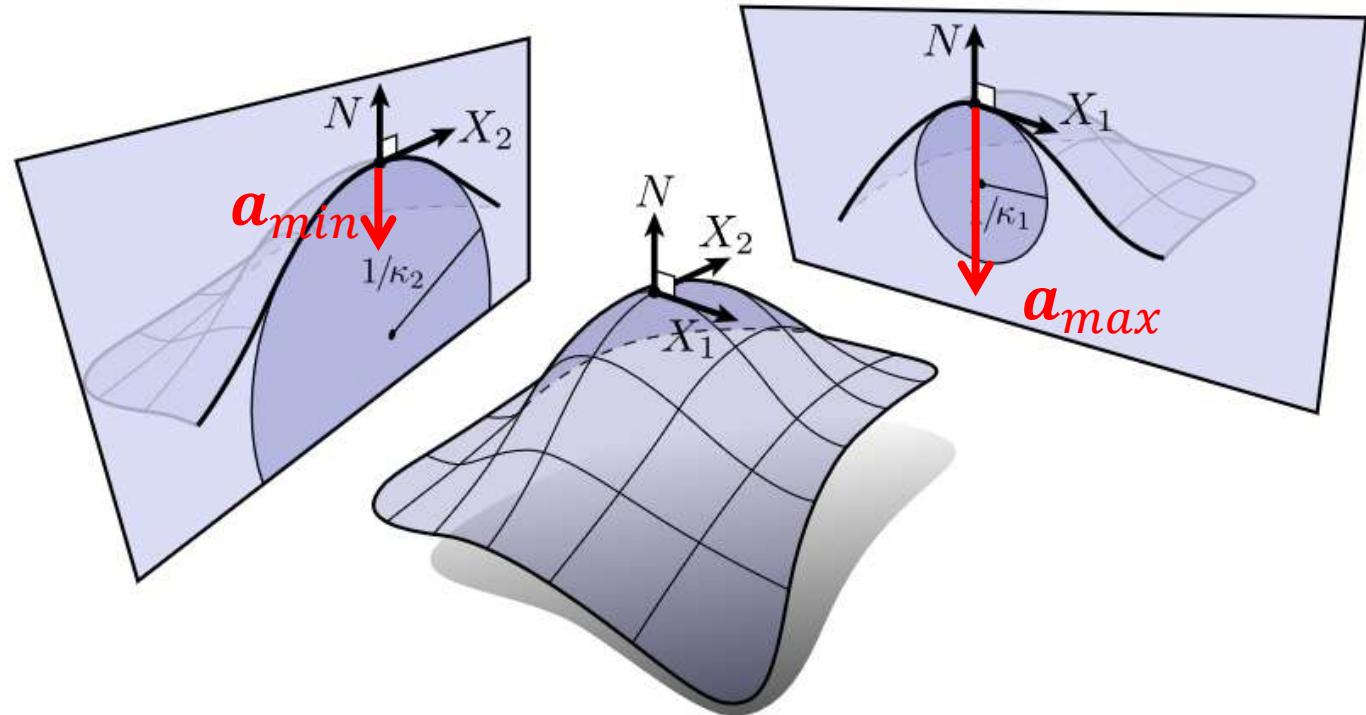
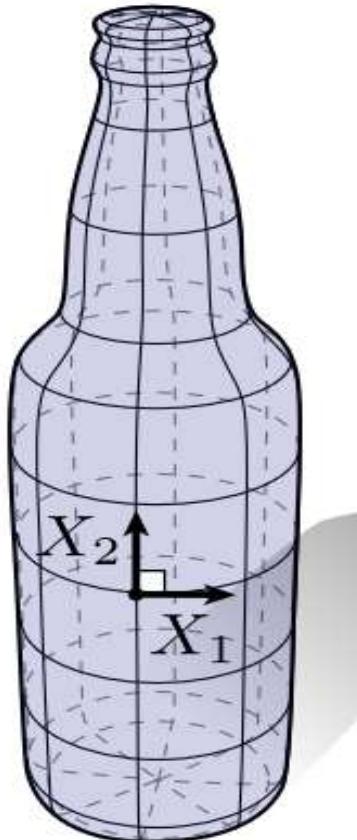
$$\mathbf{T}^0 = \frac{\mathbf{r}'}{|\mathbf{r}'|} = \frac{(1, f')}{\sqrt{1+f'^2}}, \quad \mathbf{N}^0 = \frac{(-f', 1)}{\sqrt{1+f'^2}}$$

- Görbület:

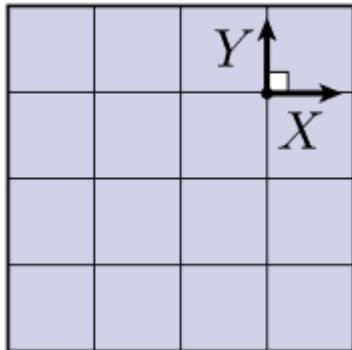
$$\kappa = \frac{\mathbf{N}^0 \cdot \mathbf{r}''}{|\mathbf{r}'|^2} = \frac{f''}{(1+f'^2)^{3/2}}$$



# Felületek: fő görbületi irányok és Gauss görbület

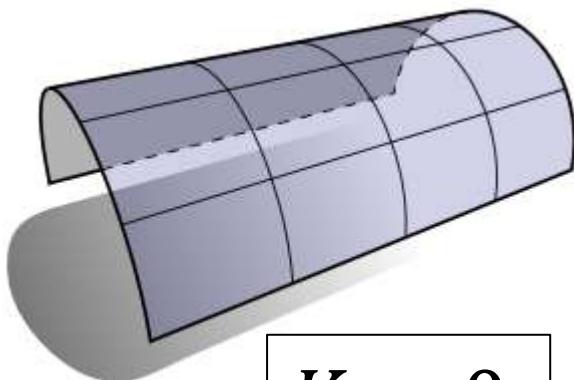


$$K = \kappa_1 \kappa_2 = a_{min} \cdot a_{max}$$



# Gauss görbület:

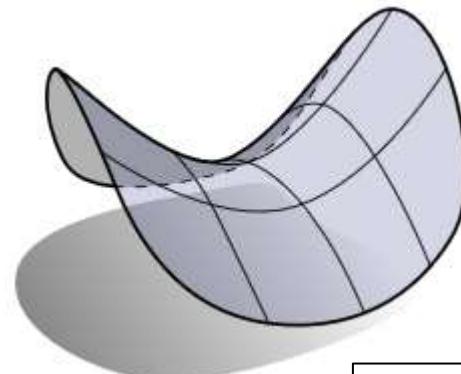
$$K = \kappa_1 \kappa_2$$



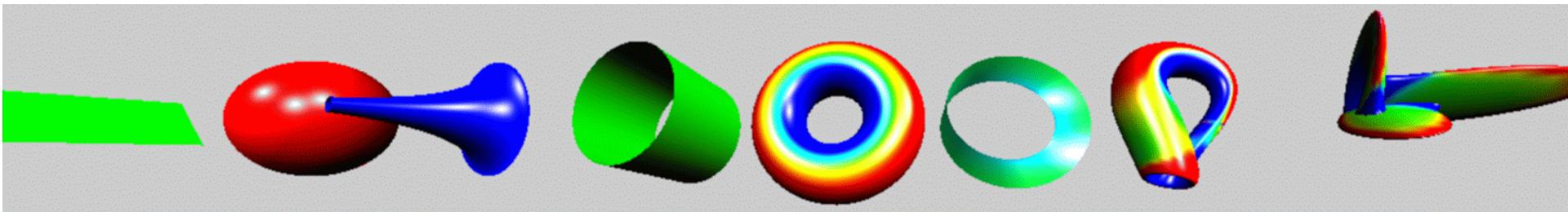
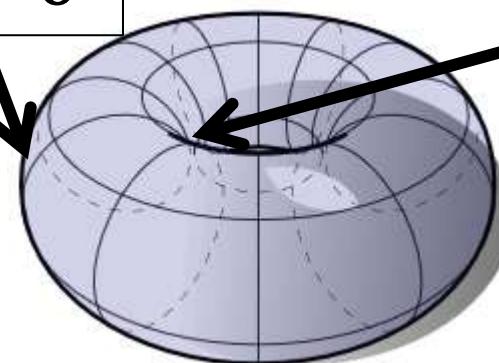
$$K = 0$$



$$K > 0$$



$$K < 0$$



Square

Sphere

Tractricoid

Cylinder

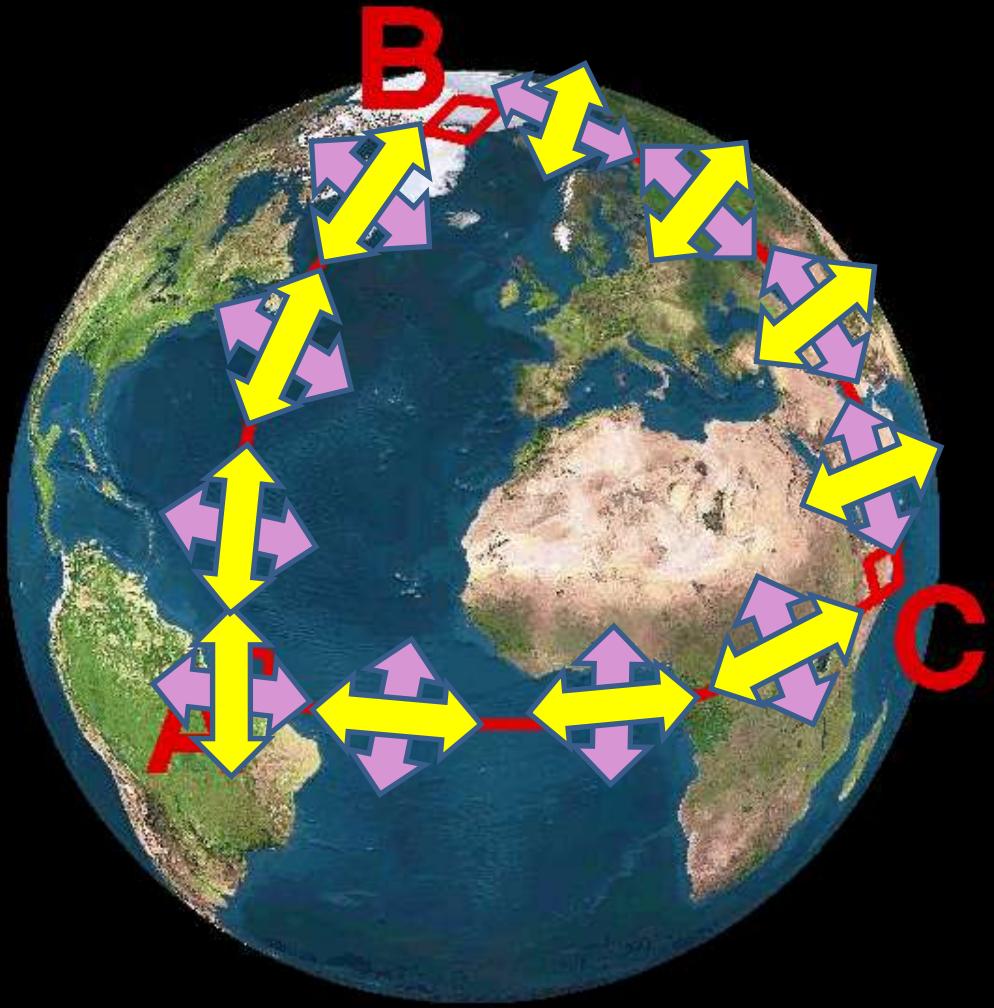
Torus

Möbius

Klein

Boy

# Párhuzamos transzport (Holonomy)



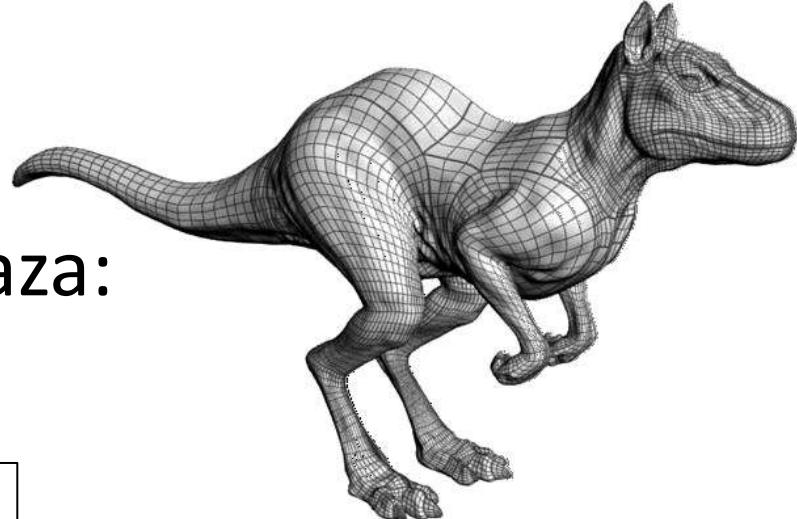
Párhuzamos transzport  
utáni szögváltozás

$$K = \frac{\Delta\varphi}{\Delta A} = \frac{\pi/2}{4R^2\pi/8} = \frac{1}{R^2}$$

Körbejárt terület

Előjel pozitív, ha a  
körbejárás és  
irányváltozás azonos

# Felületek



Felület a 3D tér 2D részhalmaza:

– Explicit:

$$z = h(x, y)$$

– Implicit:

$$f(x, y, z) = 0$$

– gömb:  $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - R^2 = 0$

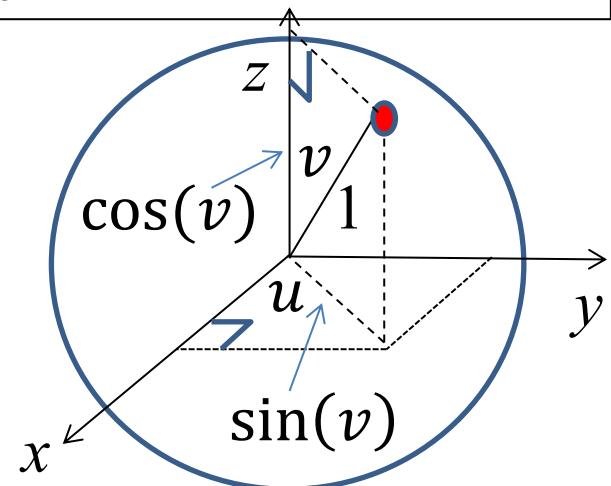
– sík:  $ax + by + cz + d = 0$

– Parametrikus:

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

– gömb:

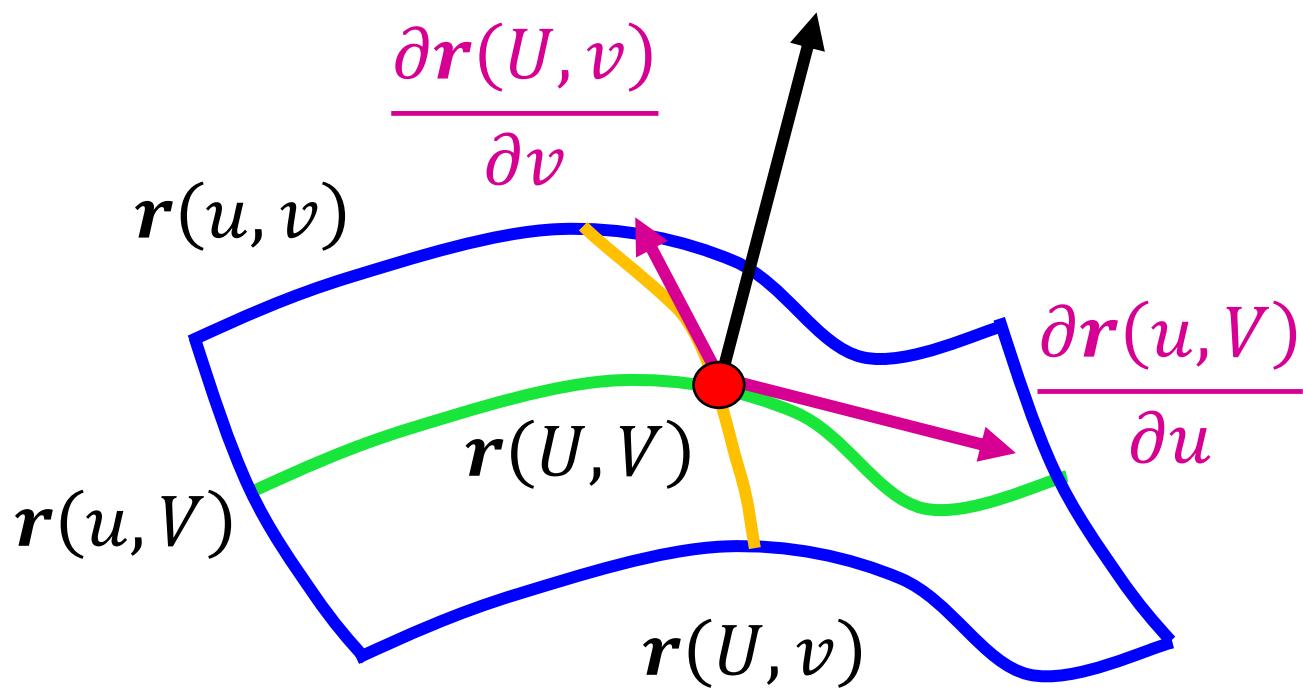
$$\begin{aligned}x(u, v) &= c_x + R \cos(u) \sin(v) \\y(u, v) &= c_y + R \sin(u) \sin(v) \\z(u, v) &= c_z + R \cos(v) \\u &\in [0, 2\pi), v \in [0, \pi)\end{aligned}$$



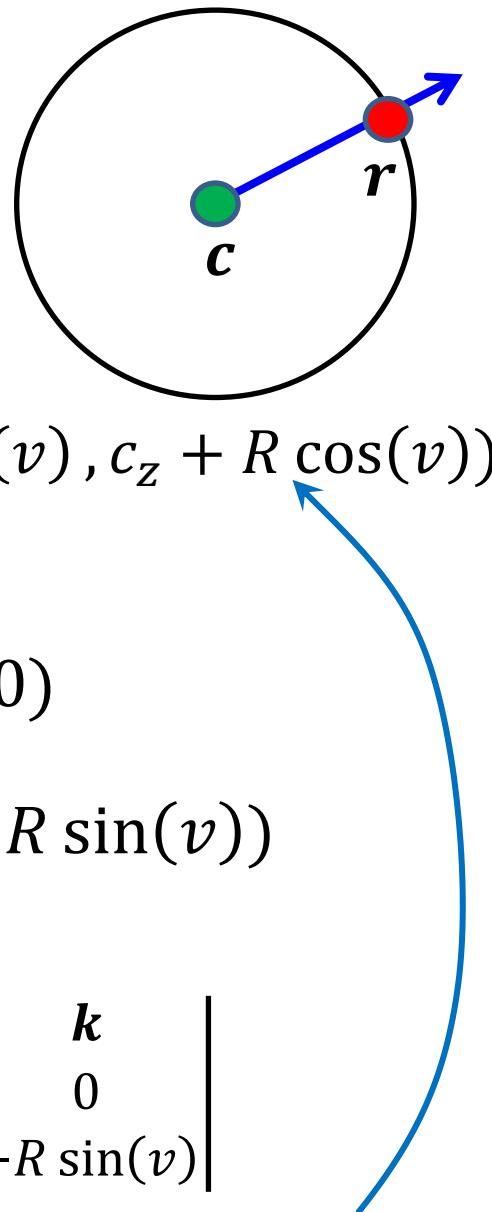
# Parametrikus felületek normálvektora

$$\mathbf{N}(U, V) = \frac{\partial \mathbf{r}(u, v)}{\partial u} \times \frac{\partial \mathbf{r}(u, v)}{\partial v}$$

$$u = U \\ v = V$$



# Példa: A gömb normálvektora



- Egy parametrikus egyenlet:

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

- Parciális deriváltak:

$$\frac{\partial \mathbf{r}}{\partial u} = (-R \sin(u) \sin(v), R \cos(u) \sin(v), 0)$$

$$\frac{\partial \mathbf{r}}{\partial v} = (R \cos(u) \cos(v), R \sin(u) \cos(v), -R \sin(v))$$

- Vektoriális szorzat:

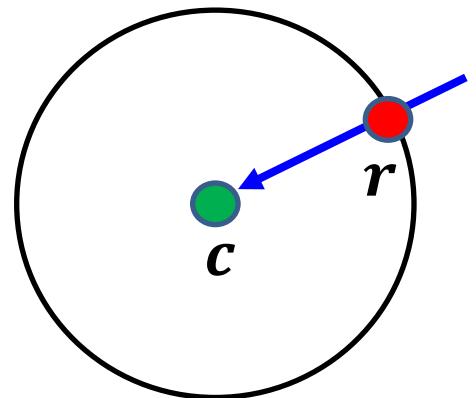
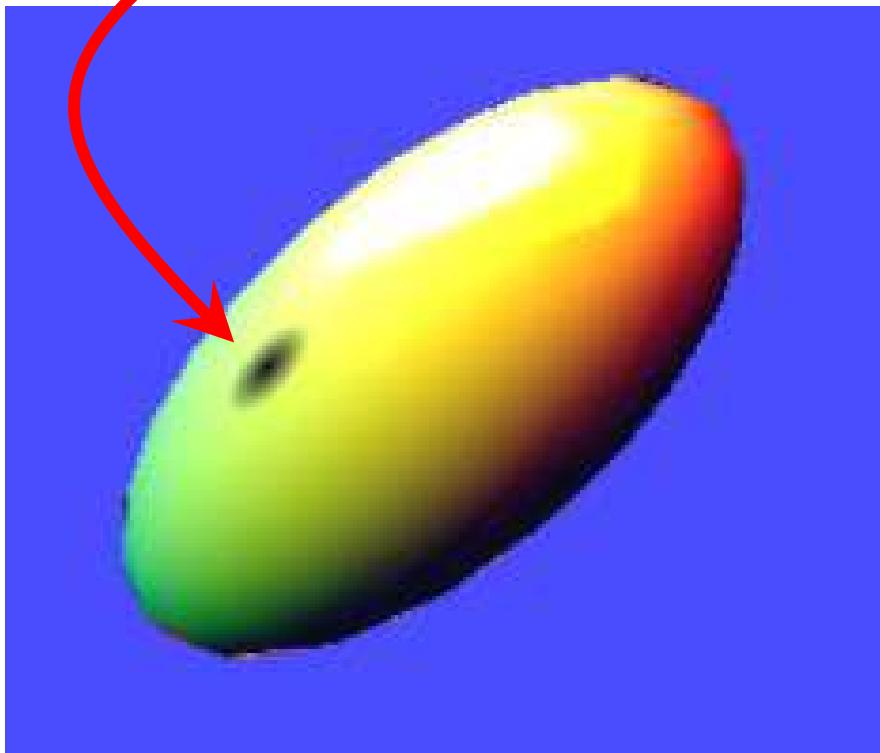
$$\mathbf{N} = \frac{\partial \mathbf{r}}{\partial u} \times \frac{\partial \mathbf{r}}{\partial v} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -R \sin(u) \sin(v) & R \cos(u) \sin(v) & 0 \\ R \cos(u) \cos(v) & R \sin(u) \cos(v) & -R \sin(v) \end{vmatrix}$$

$$= R \sin(v) (-R \cos(u) \sin(v), -R \sin(u) \sin(v), -R \cos(v))$$

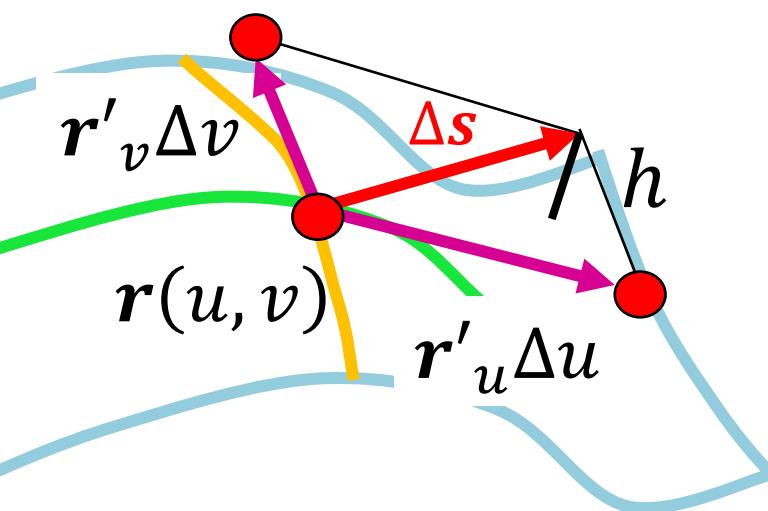
$$= R \sin(v) (\mathbf{c} - \mathbf{r}(u, v))$$

# Szingularitás

$$N = R \sin(\nu)(c - r(u, \nu))$$



# Metrikus tenzor, I. fundamentális forma



Mekkorát lépünk **a síkon**, ha  $u$   $\Delta u$ -val,  $v$  pedig  $\Delta v$ -vel változik?

$$\mathbf{r}(u + \Delta u, v + \Delta v) - \mathbf{r}(u, v) \approx$$

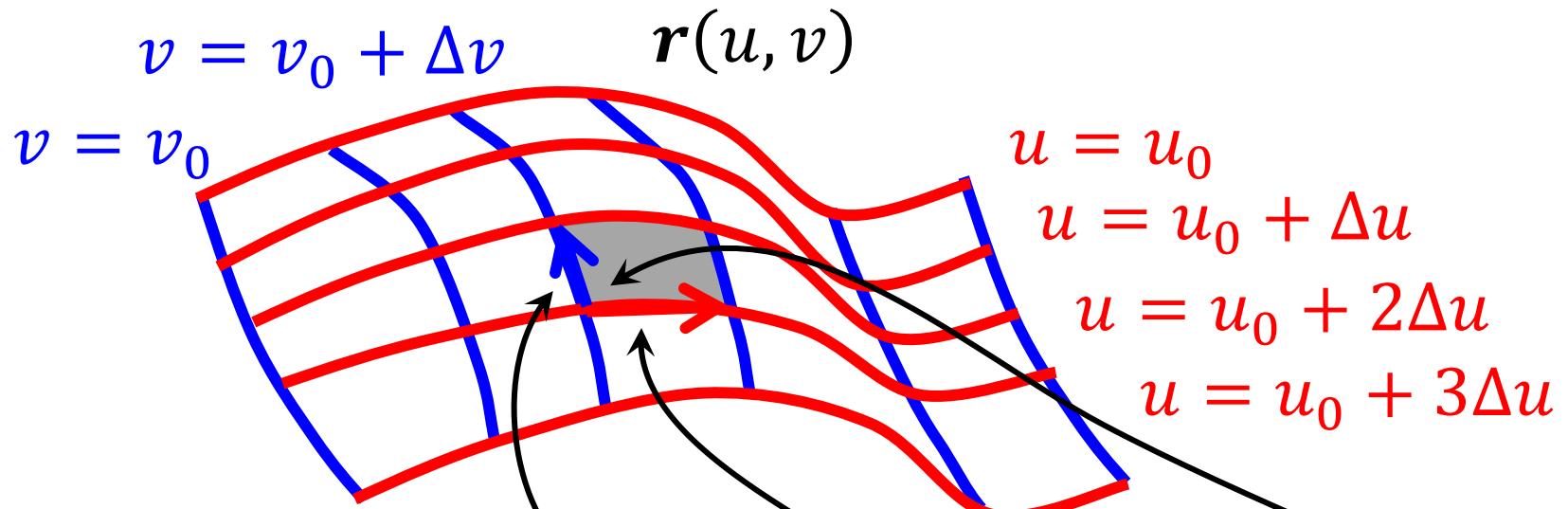
$$\underbrace{\mathbf{r}'_u \Delta u + \mathbf{r}'_v \Delta v}_{\Delta s} + \frac{1}{2} (\mathbf{r}''_{uu} \Delta u^2 + 2\mathbf{r}''_{uv} \Delta u \Delta v + \mathbf{r}''_{vv} \Delta v^2)$$

$$\Delta s^2 = \Delta s \cdot \Delta s = \mathbf{r}'_u^2 \Delta u^2 + 2\mathbf{r}'_u \cdot \mathbf{r}'_v \Delta u \Delta v + \mathbf{r}'_v^2 \Delta v^2$$

$$\Delta s^2 = [\Delta u \quad \Delta v] \underbrace{\begin{bmatrix} \mathbf{r}'_u \cdot \mathbf{r}'_u & \mathbf{r}'_u \cdot \mathbf{r}'_v \\ \mathbf{r}'_u \cdot \mathbf{r}'_v & \mathbf{r}'_v \cdot \mathbf{r}'_v \end{bmatrix}}_{g} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

$g$ : metrikus tenzor,  $I$ : első fundamentális forma  
Szimmetrikus, pozitív definit 2x2-es mátrix

# Metrikus tensor jelentése



$$I = \begin{bmatrix} r'_u \cdot r'_u & r'_u \cdot r'_v \\ r'_u \cdot r'_v & r'_v \cdot r'_v \end{bmatrix} = \begin{bmatrix} |r'_u|^2 & |r'_u||r'_v|\cos(\alpha) \\ |r'_u||r'_v|\cos(\alpha) & |r'_v|^2 \end{bmatrix}$$

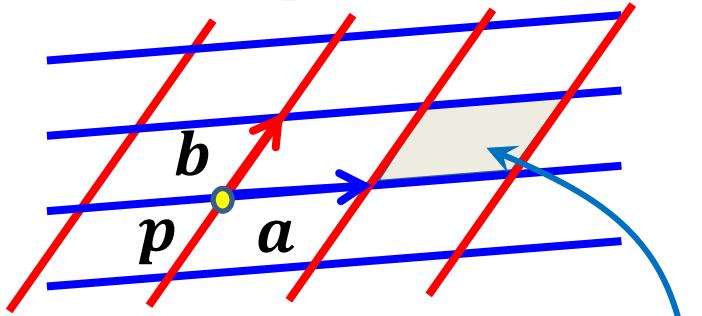
Cella területének négyzete:

$$\begin{aligned} \det(I) &= |r'_u|^2 |r'_v|^2 (1 - \cos^2(\alpha)) = |r'_u|^2 |r'_v|^2 \sin^2(\alpha) \\ &= |r'_u \times r'_v|^2 \end{aligned}$$

# Példa: Sík I. fundamentális forma

- Egy parametrikus egyenlet:

$$\mathbf{r}(u, v) = \mathbf{p} + \mathbf{a}u + \mathbf{b}v$$



- Parciális deriváltak:

$$\frac{\partial \mathbf{r}}{\partial u} = \mathbf{a}$$

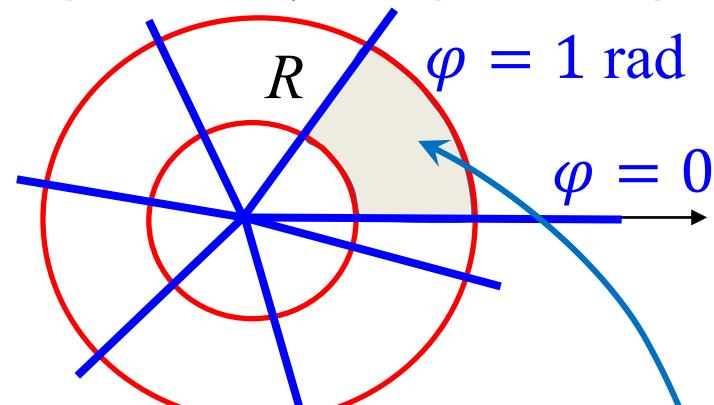
$$\det = \text{area}^2$$

$$\frac{\partial \mathbf{r}}{\partial v} = \mathbf{b}$$

- Metrikus tenzor:

$$I = \begin{bmatrix} \mathbf{a} \cdot \mathbf{a} & \mathbf{a} \cdot \mathbf{b} \\ \mathbf{a} \cdot \mathbf{b} & \mathbf{b} \cdot \mathbf{b} \end{bmatrix}$$

$$\mathbf{r}(R, \varphi) = R(\cos(\varphi), \sin(\varphi))$$



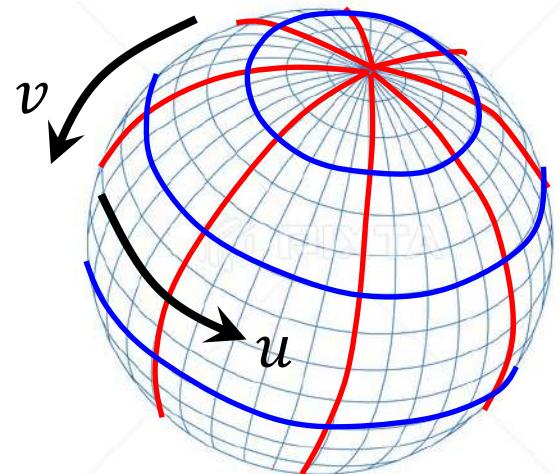
$$\frac{\partial \mathbf{r}}{\partial R} = (\cos(\varphi), \sin(\varphi))$$

$$\frac{\partial \mathbf{r}}{\partial \varphi} = R(-\sin(\varphi), \cos(\varphi))$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & R^2 \end{bmatrix}$$

$$\det = \text{area}^2$$

# Példa: Gömb I. fundamentális forma



- Egy parametrikus egyenlet:

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

- Parciális deriváltak:

$$\mathbf{r}'_u = (-R \sin(u) \sin(v), R \cos(u) \sin(v), 0)$$

$$\mathbf{r}'_v = (R \cos(u) \cos(v), R \sin(u) \cos(v), -R \sin(v))$$

- Metrikus tenzor:

$$I = \begin{bmatrix} \mathbf{r}'_u \cdot \mathbf{r}'_u & \mathbf{r}'_u \cdot \mathbf{r}'_v \\ \mathbf{r}'_u \cdot \mathbf{r}'_v & \mathbf{r}'_v \cdot \mathbf{r}'_v \end{bmatrix} = \begin{bmatrix} R^2 \sin^2(v) & 0 \\ 0 & R^2 \end{bmatrix}$$

# Mire jó példa: sebesség a gömbfelületen

- Pálya paramétertérben:

$$u = at + u_0 \quad v = bt + v_0$$

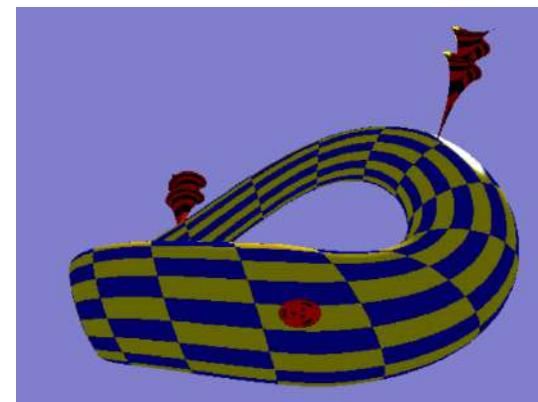
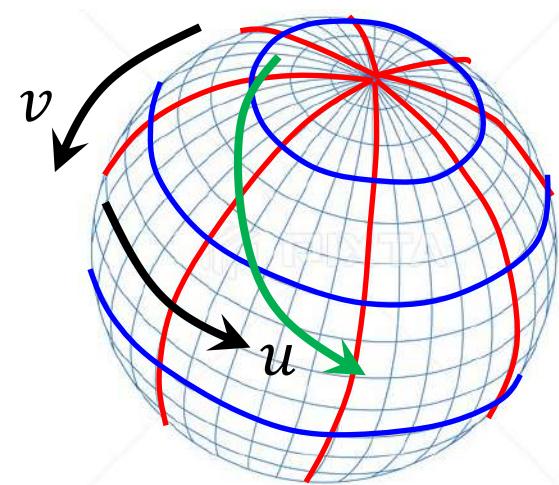
- Sebesség paramétertérben:

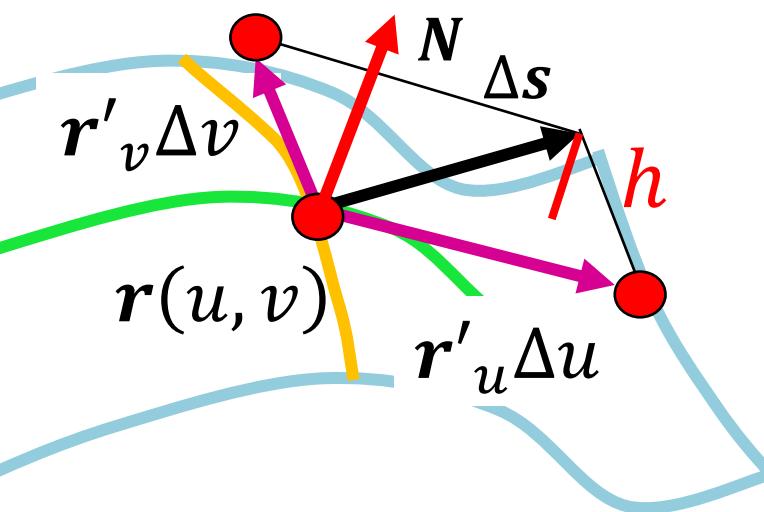
$$du = a \, dt, \quad dv = b \, dt$$

- Sebesség a felületen:

$$\frac{ds}{dt} = \frac{\sqrt{R^2 \sin^2(v) \, du^2 + R^2 \, dv^2}}{dt} = R \sqrt{\sin^2(v) \, a^2 + b^2}$$

- Mik az egyenesek (geodézikus vonalak) a felületen?





## II. fundamentális forma

Mennyire távolodunk a síktól, ha  $u$   $\Delta u$ -val,  $v$  pedig  $\Delta v$ -vel változik?

$$\mathbf{r}(u + \Delta u, v + \Delta v) - \mathbf{r}(u, v) \approx$$

$$\mathbf{r}'_u \Delta u + \mathbf{r}'_v \Delta v + \frac{1}{2} (\mathbf{r}''_{uu} \Delta u^2 + 2\mathbf{r}''_{uv} \Delta u \Delta v + \mathbf{r}''_{vv} \Delta v^2)$$

A normálvektorral szorozva a síktól való távolság marad:

$$h(\Delta u, \Delta v) = \frac{1}{2} (N \cdot \mathbf{r}''_{uu} \Delta u^2 + 2N \cdot \mathbf{r}''_{uv} \Delta u \Delta v + N \cdot \mathbf{r}''_{vv} \Delta v^2)$$

$$h(\Delta u, \Delta v) = \frac{1}{2} [\Delta u \quad \Delta v] \begin{bmatrix} N \cdot \mathbf{r}''_{uu} & N \cdot \mathbf{r}''_{uv} \\ N \cdot \mathbf{r}''_{uv} & N \cdot \mathbf{r}''_{vv} \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

*II: második fundamentális forma  
Szimmetrikus 2x2-es mátrix*

# Példa: Sík II. fundamentális forma

- Egy parametrikus egyenlet:

$$\mathbf{r}(u, v) = \mathbf{p} + \mathbf{a}u + \mathbf{b}v$$

$$\mathbf{r}(R, \varphi) = R(\cos(\varphi), \sin(\varphi), 0)$$

- Parciális deriváltak:

$$\mathbf{r}''_{uu} = 0$$

$$\mathbf{r}''_{vv} = 0$$

$$\mathbf{r}''_{uv} = 0$$

$$\mathbf{N} = \mathbf{a} \times \mathbf{b} / |\mathbf{a} \times \mathbf{b}|$$

$$\mathbf{r}''_{RR} = 0$$

$$\mathbf{r}''_{\varphi\varphi} = R(-\cos(\varphi), -\sin(\varphi), 0)$$

$$\mathbf{r}''_{R\varphi} = (-\sin(\varphi), \cos(\varphi), 0)$$

$$\mathbf{N} = (0, 0, 1)$$

- II. fundamentális forma

$$II = \begin{bmatrix} \mathbf{N} \cdot \mathbf{r}''_{uu} & \mathbf{N} \cdot \mathbf{r}''_{uv} \\ \mathbf{N} \cdot \mathbf{r}''_{uv} & \mathbf{N} \cdot \mathbf{r}''_{vv} \end{bmatrix} = 0 \quad II = 0$$

# Példa: Gömb

## II. fundamentális forma

- Egy parametrikus egyenlet:

$$\mathbf{r}(u, v) = (R \cos(u) \sin(v), R \sin(u) \sin(v), R \cos(v))$$

- Parciális deriváltak:

$$\mathbf{r}''_{uu} = (-R \cos(u) \sin(v), -R \sin(u) \sin(v), 0)$$

$$\mathbf{r}''_{vv} = (-R \cos(u) \sin(v), -R \sin(u) \sin(v), -R \cos(v))$$

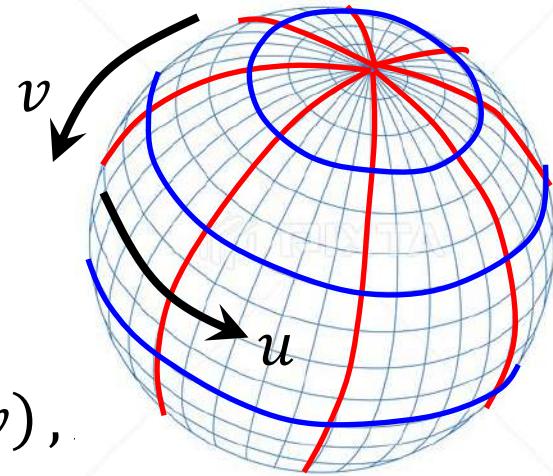
$$\mathbf{r}''_{uv} = (-R \sin(u) \cos(v), R \cos(u) \cos(v), 0)$$

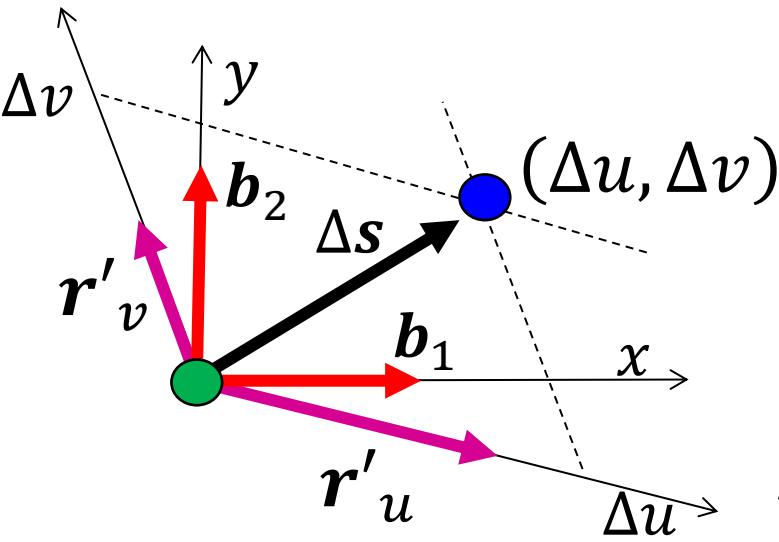
$$\mathbf{N}^0 = (-\cos(u) \sin(v), -\sin(u) \sin(v), -\cos(v))$$

- II. fundamentális forma:

$$II = \begin{bmatrix} \mathbf{N}^0 \cdot \mathbf{r}''_{uu} & \mathbf{N}^0 \cdot \mathbf{r}''_{uv} \\ \mathbf{N}^0 \cdot \mathbf{r}''_{uv} & \mathbf{N}^0 \cdot \mathbf{r}''_{vv} \end{bmatrix} = \begin{bmatrix} R \sin^2(v) & 0 \\ 0 & R \end{bmatrix}$$

- Görbület  $\mathbf{r}'_u$  irányban:  $\kappa = \frac{2\Delta h}{\Delta s^2} = \frac{\mathbf{N}^0 \cdot \mathbf{r}''_{uu}}{\|\mathbf{r}'_u\|^2} = \frac{R \sin^2(v)}{R^2 \sin^2(v)} = \frac{1}{R}$





## Bázis normalizálás

$$\Delta s^2 = [\Delta u \quad \Delta v] I \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

$$\Delta s^2 \neq \Delta u^2 + \Delta v^2$$

Az  $r'_u$  és  $r'_v$  nem egységhosszúak és nem merőlegesek.

- Új ortonormált bázis  $\mathbf{b}_1, \mathbf{b}_2$ :

$$\Delta s = r'_u \Delta u + r'_v \Delta v = \mathbf{b}_1 x + \mathbf{b}_2 y \quad // \cdot \mathbf{b}_1, \mathbf{b}_2$$

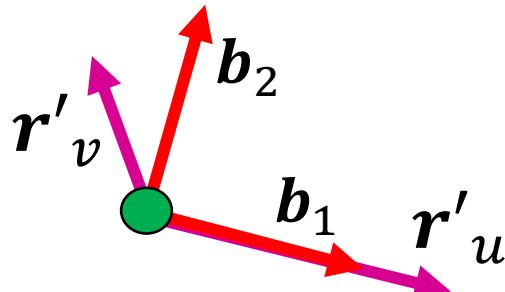
- Bázistranszformáció:

$$r'_u \cdot \mathbf{b}_1 \Delta u + r'_v \cdot \mathbf{b}_1 \Delta v = x$$

$$r'_u \cdot \mathbf{b}_2 \Delta u + r'_v \cdot \mathbf{b}_2 \Delta v = y$$

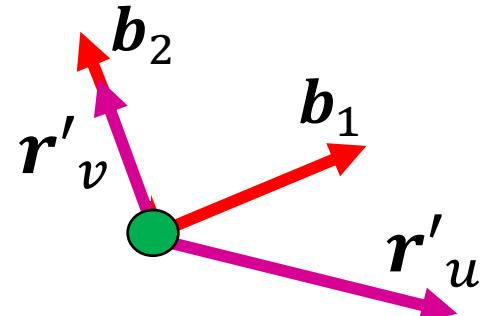
$$B = \begin{bmatrix} r'_u \cdot \mathbf{b}_1 & r'_v \cdot \mathbf{b}_1 \\ r'_u \cdot \mathbf{b}_2 & r'_v \cdot \mathbf{b}_2 \end{bmatrix}, \quad B \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad B^{-1} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

Legyen  $B = \begin{bmatrix} r'_u \cdot b_1 & r'_v \cdot b_1 \\ r'_u \cdot b_2 & r'_v \cdot b_2 \end{bmatrix}$  szimmetrikus



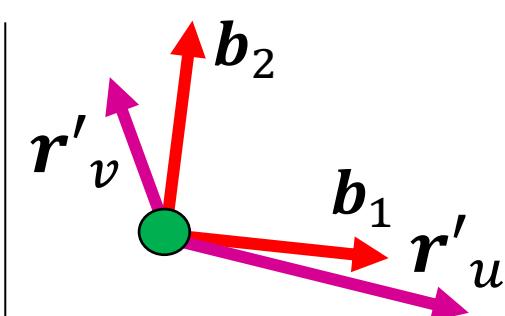
$$b_1 = r'_u / |r'_u|$$

$$\begin{bmatrix} |r'_u| & r'_v \cdot r'_u / |r'_u| \\ 0 & * \end{bmatrix}$$



$$b_2 = r'_v / |r'_v|$$

$$\begin{bmatrix} * & 0 \\ r'_u \cdot r'_v / |r'_v| & |r'_v| \end{bmatrix}$$



$$B = \begin{bmatrix} C & D \\ D & E \end{bmatrix}$$

- A két szélső esetben az egyik diagonálon kívüli elem 0 a másik pedig az  $r'_u \cdot r'_v$  szerinti előjelű, a másik esetben fordítva.
- Átmenetkor az egyik zérusról indul, a másik szemből zérusra érkezik, tehát kell lennie olyan helyzetnek, amikor megegyeznek.

# Metrikus tenzor a normalizált bázisban

- Bázis transzformáció:  $\mathbf{B}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$
- Metrika:  
$$\Delta s^2 = [\Delta u \quad \Delta v] \mathbf{I} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = [x \quad y] (\mathbf{B}^{-1})^T \cdot \mathbf{I} \cdot \mathbf{B}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} =$$
- Mivel  $\mathbf{B}$  és  $\mathbf{I}$  szimmetrikus:  
$$= [x \quad y] \cdot \mathbf{I} \cdot (\mathbf{B}^2)^{-1} \begin{bmatrix} x \\ y \end{bmatrix} = x^2 + y^2, \text{ ha } \boxed{\mathbf{B}^2 = \mathbf{I}}$$

# Alakoperátor (Shape operator)

- Érintősíkban ortonormált bázis:  $\mathbf{B}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$

$$h = \frac{1}{2} [\Delta u \quad \Delta v] \mathbf{II} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \frac{1}{2} [x \quad y] (\mathbf{B}^{-1})^T \cdot \mathbf{II} \cdot \mathbf{B}^{-1} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Mivel szimmetrikus mátrixok, az alakoperátor:  
 $(\mathbf{B}^{-1})^T \cdot \mathbf{II} \cdot \mathbf{B}^{-1} = \mathbf{II} \cdot (\mathbf{B}^2)^{-1} = \boxed{\mathbf{II} \cdot \mathbf{I}^{-1} = \mathbf{S}}$
- Érintősíktól távolodás alakoperátorral:

$$h = \frac{1}{2} [x \quad y] \mathbf{S} \begin{bmatrix} x \\ y \end{bmatrix}$$

# A gömb alakoperátora

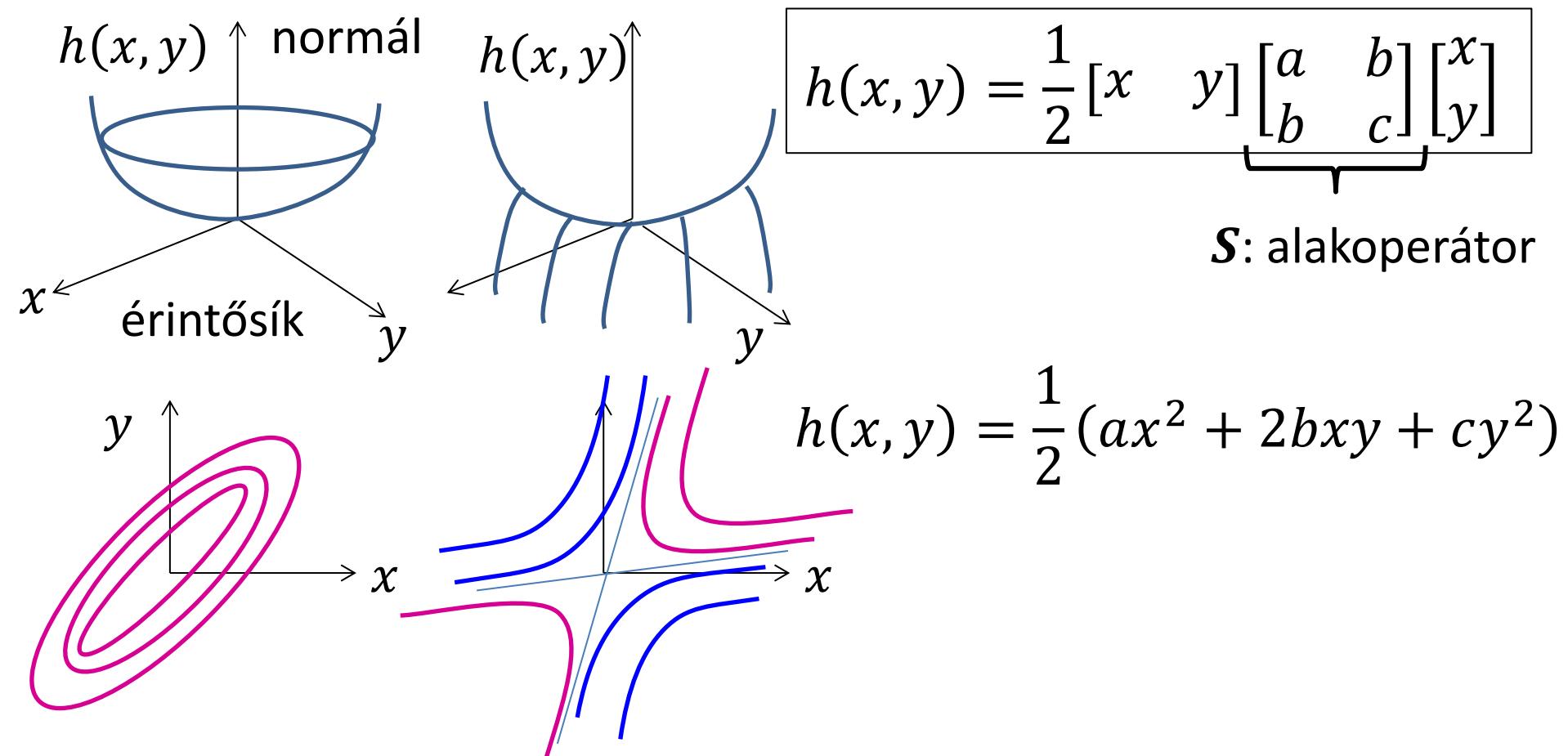
$$S = II \cdot I^{-1}$$

$$= \begin{bmatrix} R\sin^2(\nu) & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} R^2\sin^2(\nu) & 0 \\ 0 & R^2 \end{bmatrix}^{-1}$$

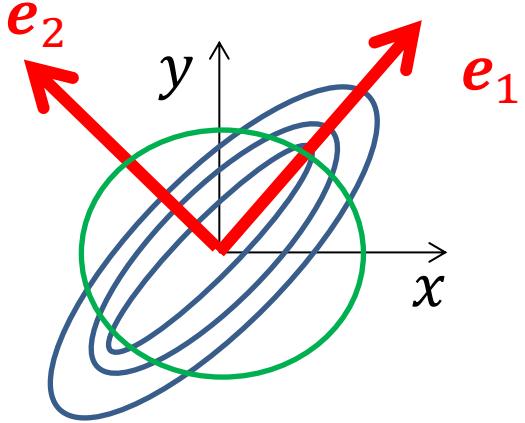
$$= \begin{bmatrix} R\sin^2(\nu) & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} 1/(R^2\sin^2(\nu)) & 0 \\ 0 & 1/R^2 \end{bmatrix}$$

$$= \begin{bmatrix} 1/R & 0 \\ 0 & 1/R \end{bmatrix}$$

# Érintősíktól távolság alakoperátorral



Szintvonalak:  $h(x, y) = \text{állandó}$



## Fő görbületi irányok

A szintvonalak ellipszisek vagy hiperbolák, a legsűrűbb és a legritkább szintvonalak a merőleges főtengelyek irányába vannak.

- Köv: A fő görbületi irányok mindenkorábban egymásra merőlegesek.
- Meghatározás: Milyen  $x, y$ -ra van szélsőértéke a  $h(x, y)$ -nak az  $x^2 + y^2 = s^2$  feltétel (**körbemegyünk**) mellett?

$$H(x, y, \lambda) = \frac{1}{2}(ax^2 + 2bxy + cy^2) - \frac{\lambda}{2}(x^2 + y^2 - s^2)$$

szélsőértéke (Lagrange multiplikátor):

$$\frac{\partial H}{\partial x} = ax + by - \lambda x = 0$$

$$\frac{\partial H}{\partial y} = bx + cy - \lambda y = 0$$

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_S = \lambda \begin{bmatrix} x \\ y \end{bmatrix}$$

# Fő görbületi irányok és görbületek

- A  $e_{1,2}$  fő görbületi irányok az alakoperátor sajátvektorai

- Görbe az  $e$  fő irányban (egységvektor):  $[x \quad y] = e^T s$

$$h(x(s), y(s)) = \frac{1}{2} [x \quad y] S \begin{bmatrix} x \\ y \end{bmatrix} = \frac{s^2}{2} e^T S e = \frac{s^2}{2} \lambda e^T e = \frac{s^2}{2} \lambda$$

- Principális görbületek az alakoperátor sajátértékei:

$$\kappa_{1,2} = \frac{2h_{1,2}}{s^2} = \lambda_{1,2}$$

- Gauss görbület: A fő görbületek (sajátértékek) szorzata. Egy mátrix sajátértékeinek szorzata = determinánsa:

$$K = \det(S) = \det(II) \det(I^{-1}) = \det(II) / \det(I)$$

# Lineáris algebrai tanulságok

- Tetszőleges  $S$ -hez (szimmetrikus mátrix) találtunk merőleges főgörbületi irányokat (sajátvektor) és görbületeket (valós sajátérték).
- minden szimmetrikus  $A$  mátrixnak van egymásra merőleges  $e_i$  sajátvektor rendszere (ortonormált bázis),  $\lambda_i$  sajátértékei pedig valósak:  $Ae_i = \lambda_i e_i$

- Diagonizálás:  $A[e_1, \dots, e_D] = [e_1, \dots, e_D] \begin{bmatrix} \lambda_1 & \Lambda & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_D \end{bmatrix}$

$$A = U\Lambda U^{-1} = U\Lambda U^T$$

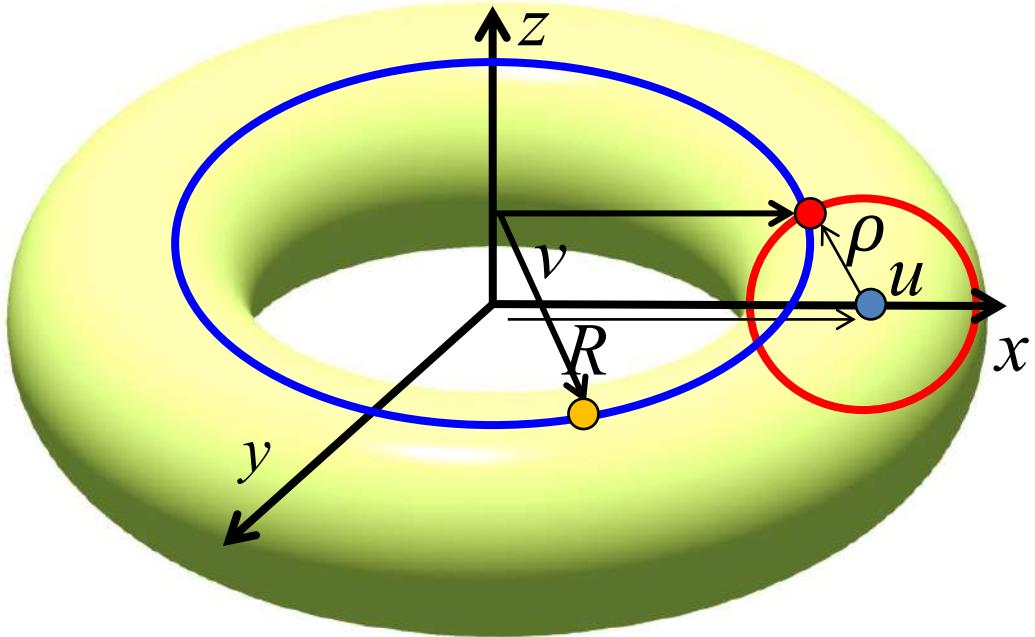
- Sajátértékek szorzata:  $\det(A) = \det(U)\det(\Lambda)\det(U^{-1}) = \det(\Lambda)$
- Mátrix függvények:  $A^n = U\Lambda U^{-1} U\Lambda U^{-1} \dots U\Lambda U^{-1} = U\Lambda^n U^{-1}$

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2}x^2 + \dots$$

$$f(A) = f(0)\mathbf{1} + f'(0)A + \frac{f''(0)}{2}A^2 + \dots = U \begin{bmatrix} f(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f(\lambda_D) \end{bmatrix} U^{-1}$$

# Gömb görbülete

- Alakoperátor:  $\begin{bmatrix} 1/R & 0 \\ 0 & 1/R \end{bmatrix}$
- Sajátértékek:  $\lambda_{1,2} = 1/R$
- Sajátvektor egyenlet:  $\begin{bmatrix} 1/R & 0 \\ 0 & 1/R \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 1/R \begin{bmatrix} x \\ y \end{bmatrix}$   
Bármi fő görbületi irány.
- Gauss görbület:
  - Principális görbületek szorzata:  $K = \lambda_1 \lambda_2 = \frac{1}{R^2}$
  - Alakoperátor determinánsa:  $K = \det\left(\begin{bmatrix} 1/R & 0 \\ 0 & 1/R \end{bmatrix}\right) = \frac{1}{R^2}$
  - Fundamentális formák determinánsainak hányadosa:  
$$K = \det\left(\begin{bmatrix} R\sin^2(\nu) & 0 \\ 0 & R \end{bmatrix}\right) / \det\left(\begin{bmatrix} R^2\sin^2(\nu) & 0 \\ 0 & R^2 \end{bmatrix}\right) = \frac{1}{R^2}$$



Példa: Tórusz  
első deriváltak és  
normál vektor

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

$$\mathbf{r}'_u = \rho [-\sin(u) \cos(v), -\sin(u) \sin(v), \cos(u)]$$

$$\mathbf{r}'_v = (R + \rho \cos(u))[-\sin(v), \cos(v), 0]$$

$$\begin{aligned} \mathbf{N} &= \frac{\partial \mathbf{r}}{\partial u} \times \frac{\partial \mathbf{r}}{\partial v} = (R + \rho \cos(u))\rho \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -\sin(u) \cos(v) & -\sin(u) \sin(v) & \cos(u) \\ -\sin(v) & \cos(v) & 0 \end{vmatrix} \\ &= -(R + \rho \cos(u))\rho [\cos(v) \cos(u), \sin(v) \cos(u), \sin(u)] \end{aligned}$$

$$\mathbf{N}^0 = -[\cos(v) \cos(u), \sin(v) \cos(u), \sin(u)]$$

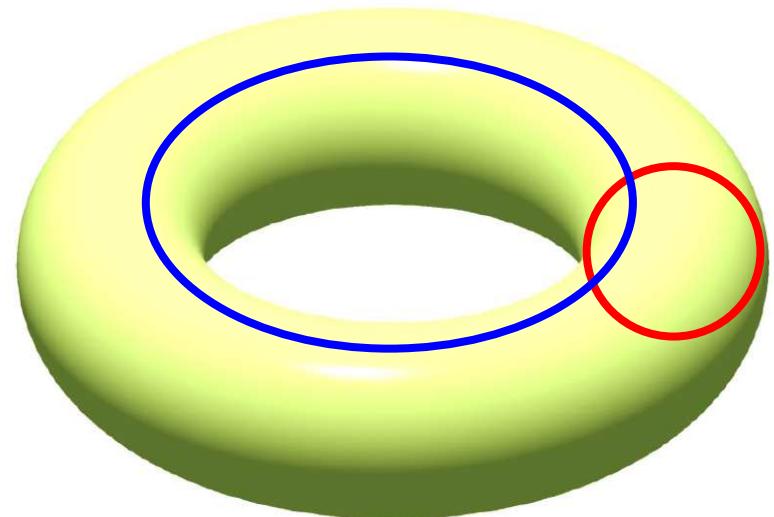
# Tórusz: I. FF és bázis transzformáció

$$\mathbf{r}'_u = \rho [-\sin(u) \cos(v), -\sin(u) \sin(v), \cos(u)]$$

$$\mathbf{r}'_v = (R + \rho \cos(u))[-\sin(v), \cos(v), 0]$$

$$I = \begin{bmatrix} \mathbf{r}'_u \cdot \mathbf{r}'_u & \mathbf{r}'_u \cdot \mathbf{r}'_v \\ \mathbf{r}'_u \cdot \mathbf{r}'_v & \mathbf{r}'_v \cdot \mathbf{r}'_v \end{bmatrix} = \begin{bmatrix} \rho^2 & 0 \\ 0 & (R + \rho \cos(u))^2 \end{bmatrix}$$

$$B = \sqrt{I} = \begin{bmatrix} \rho & 0 \\ 0 & R + \rho \cos(u) \end{bmatrix}$$



# Tórusz: II. FF, alakoperátor

$$\mathbf{r}'_u = \rho [-\sin(u) \cos(v), -\sin(u) \sin(v), \cos(u)]$$

$$\mathbf{r}'_v = (R + \rho \cos(u))[-\sin(v), \cos(v), 0]$$

$$\mathbf{N}^0 = -[\cos(v) \cos(u), \sin(v) \cos(u), \sin(u)]$$

---

$$\mathbf{r}''_{uu} = -\rho [\cos(u) \cos(v), \cos(u) \sin(v), \sin(u)]$$

$$\mathbf{r}''_{vv} = -(R + \rho \cos(u))[\cos(v), \sin(v), 0]$$

$$\mathbf{r}''_{uv} = \rho [\sin(u) \sin(v), -\sin(u) \cos(v), 0]$$

---

$$II = \begin{bmatrix} \mathbf{N}^0 \cdot \mathbf{r}''_{uu} & \mathbf{N}^0 \cdot \mathbf{r}''_{uv} \\ \mathbf{N}^0 \cdot \mathbf{r}''_{uv} & \mathbf{N}^0 \cdot \mathbf{r}''_{vv} \end{bmatrix} = \begin{bmatrix} \rho & 0 \\ 0 & (R + \rho \cos(u)) \cos(u) \end{bmatrix}$$

$$S = II \cdot I^{-1} = \begin{bmatrix} \rho & 0 \\ 0 & (R + \rho \cos(u)) \cos(u) \end{bmatrix} \begin{bmatrix} \rho^2 & 0 \\ 0 & (R + \rho \cos(u))^2 \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} 1/\rho & 0 \\ 0 & \cos(u)/(R + \rho \cos(u)) \end{bmatrix}$$

# Fő görbületek és Gauss görbület

$$\mathbf{B} = \sqrt{I} = \begin{bmatrix} 0 \\ 0 & R + \rho \cos(u) \end{bmatrix}$$

---


$$s \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} 1/\rho & 0 \\ 0 & \cos(u)/(R + \rho \cos(u)) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}$$


---

$$\lambda_1 = \frac{1}{\rho}, \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \mathbf{B}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/\rho \\ 0 \end{bmatrix}$$

$$\mathbf{d}_1 = \mathbf{r}'_u \Delta u + \mathbf{r}'_v \Delta v = \mathbf{r}'_u / \rho$$

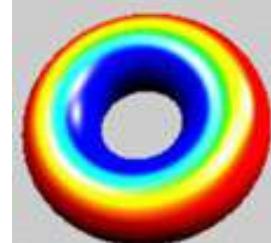

---

$$\lambda_2 = \frac{\cos(u)}{R + \rho \cos(u)}, \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \mathbf{B}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/(R + \rho \cos(u)) \end{bmatrix}$$

$$\mathbf{d}_2 = \mathbf{r}'_u \Delta u + \mathbf{r}'_v \Delta v = \mathbf{r}'_v / (R + \rho \cos(u))$$

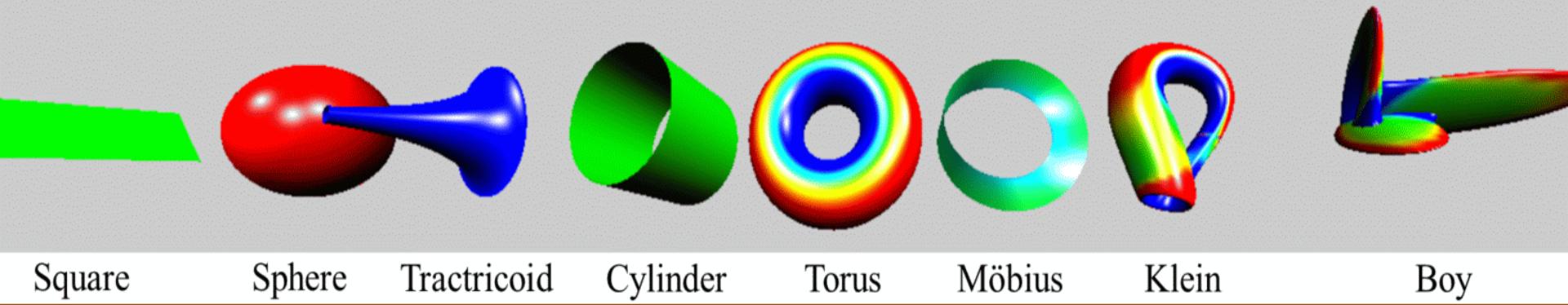

---

$$K = \lambda_1 \lambda_2 = \frac{\cos(u)}{\rho(R + \rho \cos(u))}$$



# Program: $u, v \rightarrow K$

```
Dnum U = Dnum(u, 1, 0), V = Dnum(v, 0, 1), X, Y, Z;  
eval(U, V, X, Y, Z); // U,V -> X, Y, Z  
  
vec3 ru = vec3(X(1, 0), Y(1, 0), Z(1, 0)),  
vec3 rv = vec3(X(0, 1), Y(0, 1), Z(0, 1));  
vec3 normal = normalize(cross(ru, rv));  
// I fundamental form  
float E = dot(ru, ru), F = dot(ru, rv), G = dot(rv, rv);  
  
vec3 ruu = vec3(X(2, 0), Y(2, 0), Z(2, 0)),  
vec3 ruv = vec3(X(1, 1), Y(1, 1), Z(1, 1));  
vec3 rvv = vec3(X(0, 2), Y(0, 2), Z(0, 2));  
// II Fundamental form  
float L = dot(normal, ruu), M = dot(normal, ruv), N = dot(normal, rvv);  
float curvature = (L * N - M * M) / (E * G - F * F); // Gauss curvature K  
  
void Sphere :: eval(Dnum U, Dnum V, Dnum X, Dnum Y, Dnum Z) {  
    X = Cos(U) * Sin(V) * R; Y = Sin(U) * Sin(V) * R; Z = Cos(V) * R;  
}
```



Square

Sphere

Tractricoid

Cylinder

Torus

Möbius

Klein

Boy

### Torus:

$$X = (\cos(U) * r + R) * \cos(V); \quad Y = (\cos(U) * r + R) * \sin(V); \quad Z = \sin(U) * r;$$

### Tractricoid:

$$X = \cos(V) / \cosh(U); \quad Y = \sin(V) / \cosh(U); \quad Z = U - \tanh(U);$$

### Cylinder:

$$X = \cos(U); \quad Y = \sin(U); \quad Z = V;$$

### Möbius:

$$X = (\cos(U) * V + R) * \cos(U * 2); \quad Y = (\cos(U) * V + R) * \sin(U * 2); \quad Z = \sin(U) * V;$$

### Klein:

$$\text{Dnum } a = \cos(U) * (\sin(U) + 1) * 3, b = \sin(U) * 8, c = (\cos(U) * (-1) + 2);$$

$$X = a + c * ((U(0, 0) > M_PI) ? \cos(V + M_PI) : \cos(U) * \cos(V));$$

$$Y = b + ((U(0, 0) > M_PI) ? 0 : c * \sin(U) * \cos(V));$$

$$Z = c * \sin(V);$$



# Van integrálgeometria is?

- Igen van: Geometriai valószínűsséggel foglalkozik
- Pontokat szórunk el egyenletesen a síkon (vagy térben). Mi a valószínűsége, hogy egy pont egy adott alakzat belsejében van?  
Válasz: arányos a területtel (terfogattal)
- Egyeneseket szórunk el egyenletesen a síkon (vagy térben). Mi a valószínűsége, hogy egy egyenes egy adott konvex sokszöget (konvex testet) metsz?  
Válasz: arányos a kerülettel (felünnel)
- Alkalmazás: Milyen adatstruktúrában kell felületeket tárolni, hogy hatékonyan megmondhassuk, hogy egy félegyenes melyiket metszi először.

*"A semmiből egy új, más világot teremtettem."*

*Bolyai János*

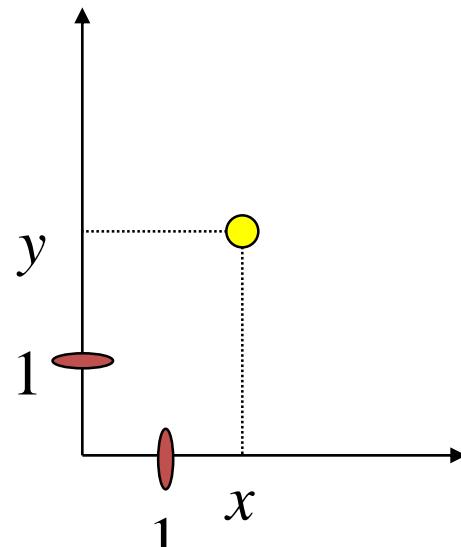
# Geometriai modellezés

## 1. Pontok és klasszikus görbék

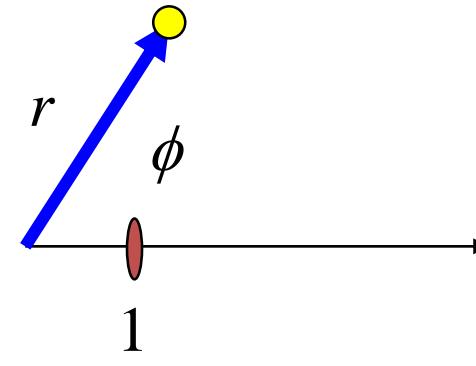
Szirmay-Kalos László



# Pontok definíciója koordinátarendszerrel



Descartes

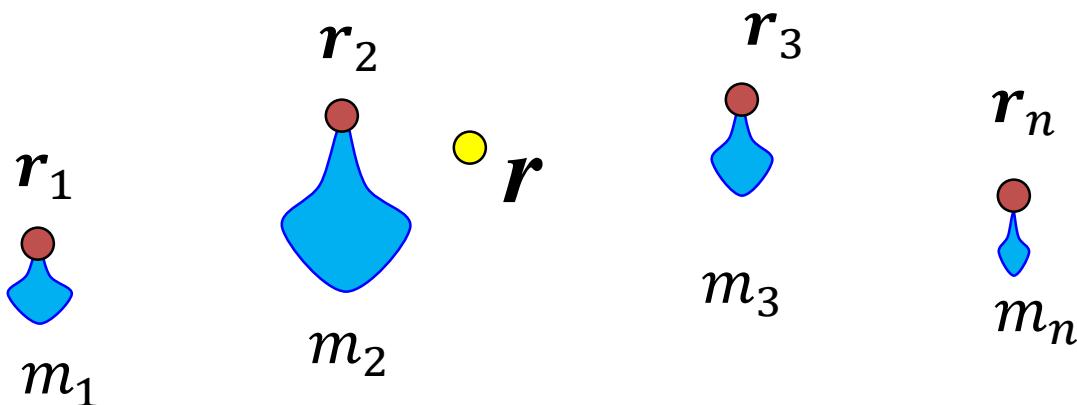


Polár

## Számokkal!

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

# Baricentrikus (homogén) koordináták



Forgatónyomaték zérus

$$\sum_i (\mathbf{r}_i - \mathbf{r}) \times m_i \mathbf{g} = 0$$

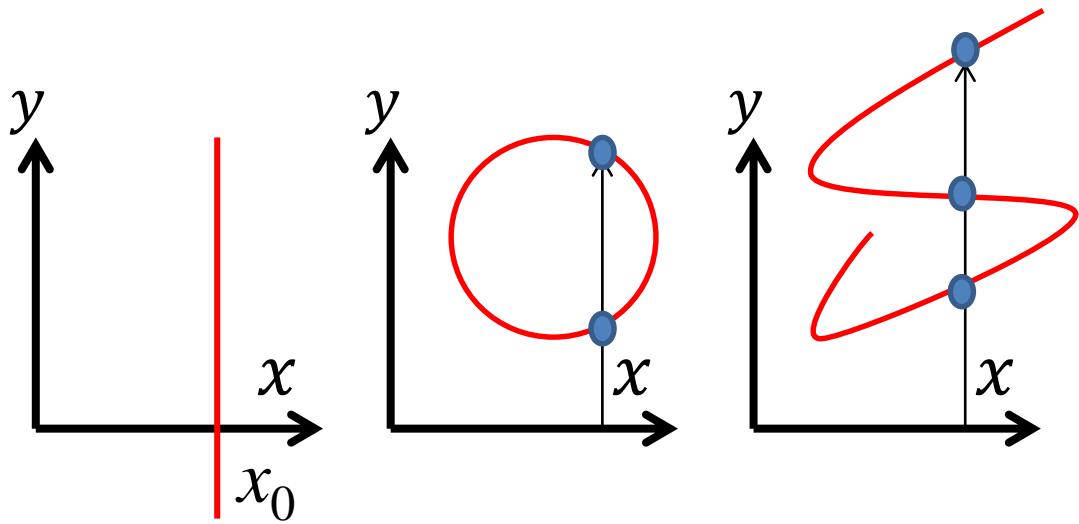
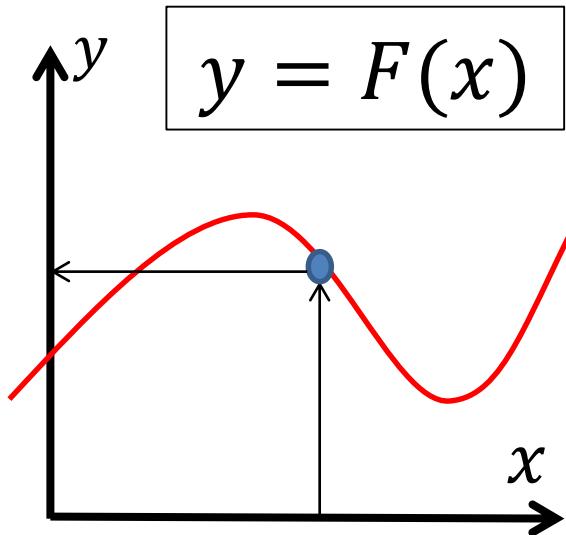
$$\mathbf{r} = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i}$$

- $\mathbf{r}$  az  $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n$  pontok kombinációja
- 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



# Görbék: 1D ponthalmazok

## Explicit egyenlet



2D egyenes:

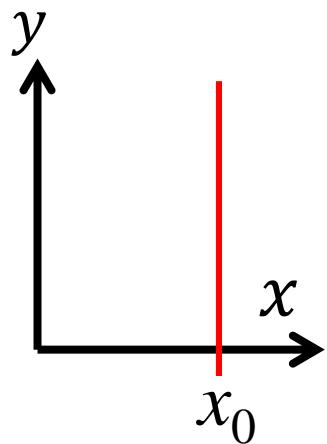
$$y = mx + b$$

Nem jó:

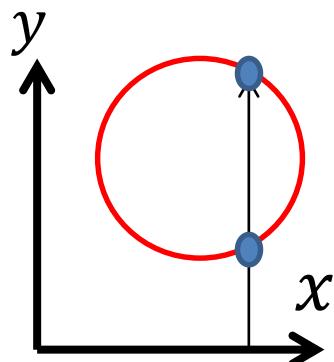
Egy  $x$ -hez nem pontosan egy  $y$

# Görbe: Implicit egyenlet

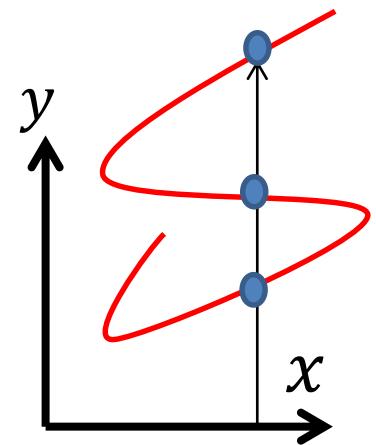
$$f(x, y) = 0 \text{ vagy } f(\mathbf{r}) = 0$$



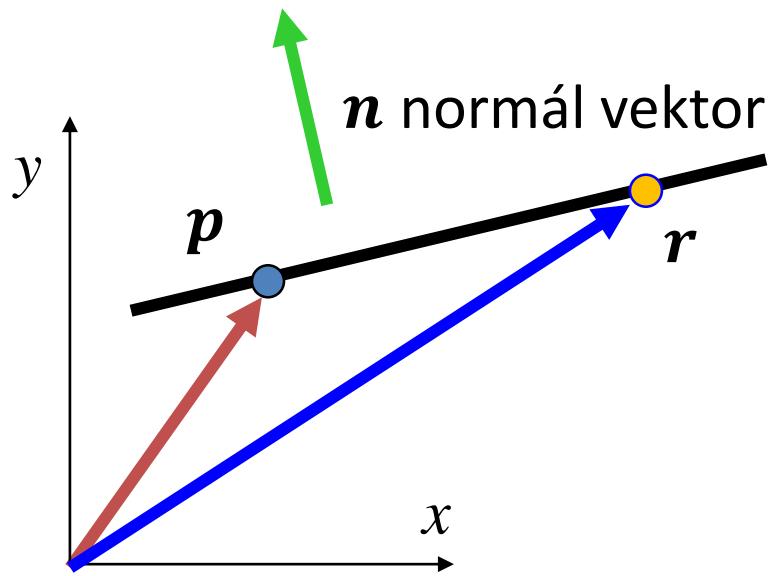
$$x - x_0 = 0$$



$$(x - c_x)^2 + (y - c_y)^2 - R^2 = 0$$



# 2D egyenes implicit egyenlete



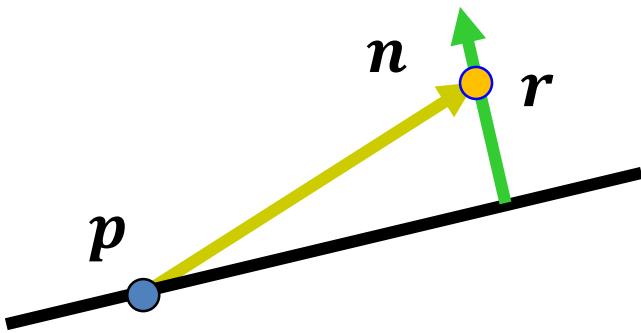
$$\mathbf{n} \cdot (\mathbf{r} - \mathbf{p}) = 0$$

$$n_x(x - p_x) + n_y(y - p_y) = 0$$

$$ax + by + c = 0$$

$$\text{Ez is: } (ax + by + c)^2 = 0$$

2D egyenestől mért távolság:



$\mathbf{n} \cdot (\mathbf{r} - \mathbf{p}) =$  Vetület  $\mathbf{n}$ -re  $\times$  az  $\mathbf{n}$  hossza

Ha  $\mathbf{n}$  egységvektor:

$\mathbf{n} \cdot (\mathbf{r} - \mathbf{p}) =$  az előjeles távolság!

# Kvadratikus görbék

- **Kör:** Azon  $\mathbf{r}(x,y)$  pontok mértani helye, amelyek a  $\mathbf{c}(c_x, c_y)$  középponttól  $R$  távolságra vannak:  $|\mathbf{r} - \mathbf{c}| = R \leftrightarrow (\mathbf{r} - \mathbf{c})^2 - R^2 = 0 \leftrightarrow (x - c_x)^2 + (y - c_y)^2 - R^2 = 0$
- **Ellipszis:** Azon  $\mathbf{r}$  pontok, amelyek a  $f_1$  és  $f_2$  fókuszpontuktól mért távolság összege állandó  $C$ :  $|\mathbf{r} - \mathbf{f}_1| + |\mathbf{r} - \mathbf{f}_2| = C$
- **Hiperbola:** Azon  $\mathbf{r}$  pontok, amelyek a  $f_1$  és  $f_2$  fókuszpontuktól mért távolság különbsége állandó  $C$ :  $|\mathbf{r} - \mathbf{f}_1| - |\mathbf{r} - \mathbf{f}_2| = C$
- **Parabola:** Azon  $\mathbf{r}$  pontok, amelyek az  $f$  fókuszponttól mért távolsága megegyezik az  $\mathbf{n}$  normálvektorú és  $\mathbf{p}$  helyvektorú egyenestől mért távolsággal:  $|\mathbf{r} - \mathbf{f}| = |\mathbf{n}^0 \cdot (\mathbf{r} - \mathbf{p})|$

# Kvadratikus görbék = kvadratikus alak

- Implicit függvény négyzetgyökök nélkül:

$$f(x, y) = a_{11}x^2 + a_{22}y^2 + 2a_{12}xy + 2a_{13}x + 2a_{23}y + a_{33} = 0$$

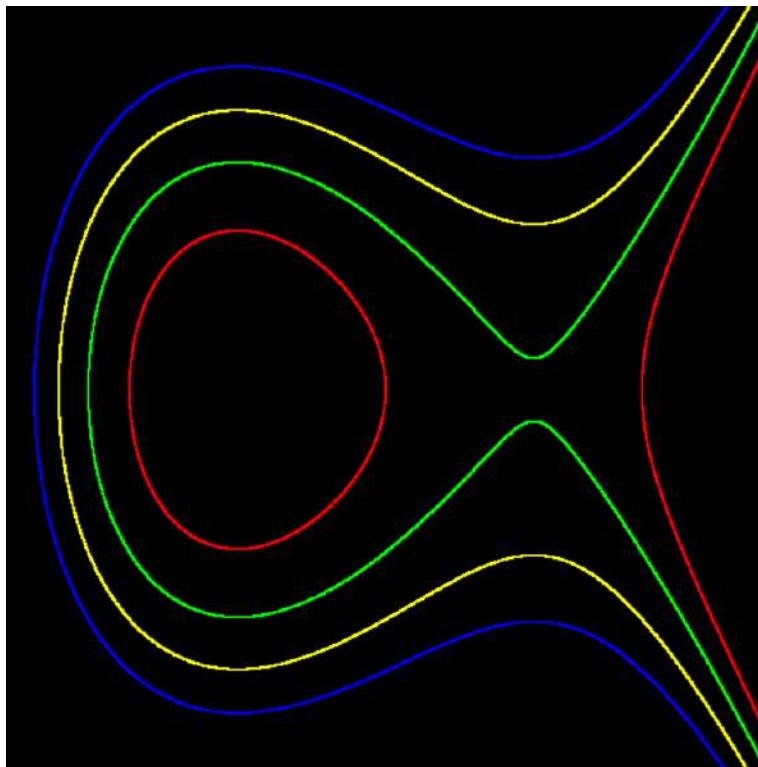
- Mátrixszal:

$$f(x, y) = [x, y, 1] \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

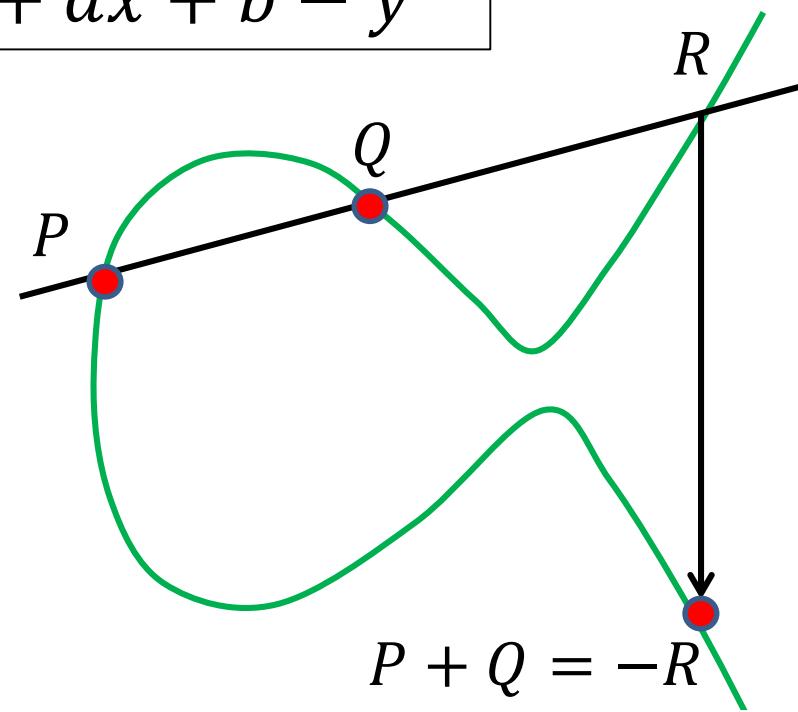


# Elliptikus görbék

$$f(x, y) = x^3 + ax + b - y^2$$



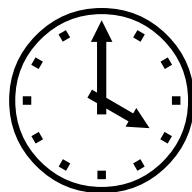
$$a = -1, b = 0, \dots, 1.6$$



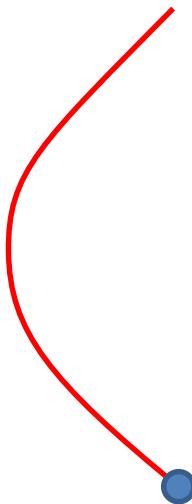
Ezzel az összeadással **csoporthoz**,  
Nem vezet ki a racionális számokból

Alkalmazás: Kriptográfia, számelmélet (Fermat téTEL bizonyítás)

# Görbe: Paraméteres egyenlet



$t$



$$x = x(t), y = y(t), z = z(t)$$

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

3D egyenes:

$$\begin{aligned}x(t) &= x_0 + v_x t \\y(t) &= y_0 + v_y t \\z(t) &= z_0 + v_z t \\t &\in (-\infty, +\infty)\end{aligned}$$

Kör:

$$\begin{aligned}x(t) &= c_x + R \cos(t) \\y(t) &= c_y + R \sin(t) \\t &\in [0, 2\pi)\end{aligned}$$

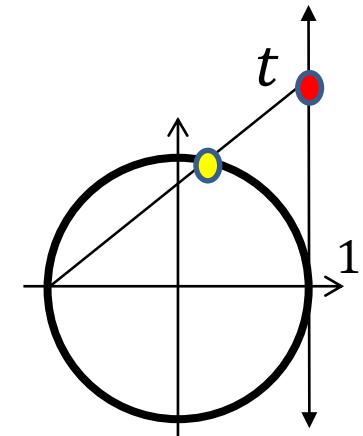
# Klasszikus görbék

- **Kör:**

$$x(t) = \frac{(4-t^2)}{(4+t^2)} \quad t \in (-\infty, +\infty)$$

$$y(t) = \frac{4t}{(4+t^2)}$$

Pitagoraszi  
számhármások



- **Cikloisz:**

$$x(t) = t - \sin t$$

$$y(t) = 1 - \cos t$$



- **Tractrix:**

$$x(t) = \operatorname{sech} t$$

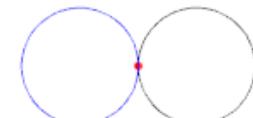
$$y(t) = t - \tanh t$$



- **Kardioid:**

$$x(t) = (1 - \cos t) \cos t$$

$$y(t) = (1 - \cos t) \sin t$$



# Pontok és klasszikus görbék

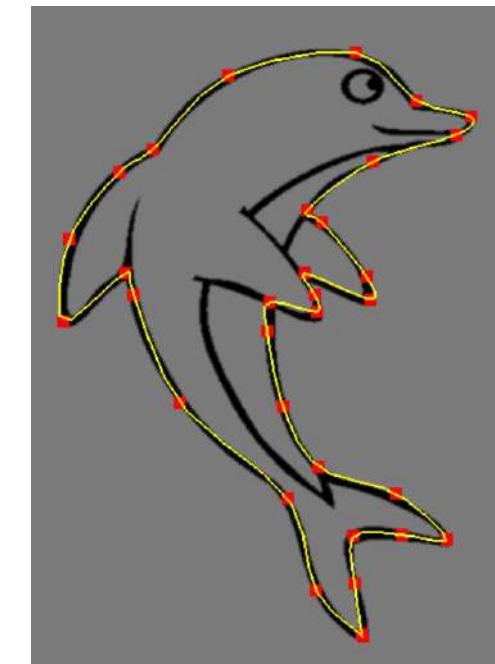
- Ponthoz koordinátarendszer kell
  - Descartes, Baricentrikus
- Görbéhez egyenlet kell
  - Az explicit ritkán használható
  - Az implicit feltételeket fogalmaz meg a pontokra
    - Azon pontok halmaza, amelyekben ... pontok távolságra ( $|p - q|$ ), ... merőleges ( $\mathbf{d} \cdot \mathbf{v} = 0$ ), ... párhuzamos ( $\mathbf{d} \times \mathbf{v} = 0$ ), ... vetülete ( $\mathbf{v}^0 \cdot \mathbf{r}$ ).
  - A paraméteres mozgásként fogalmazza meg a görbét



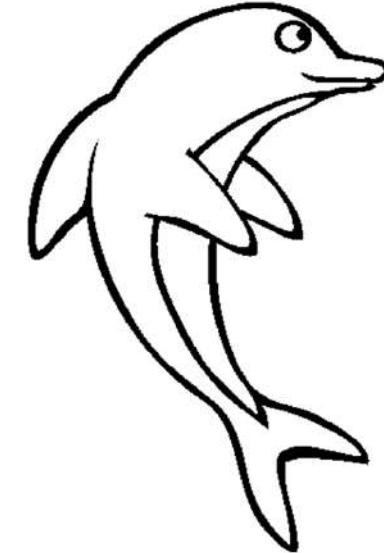
# Geometriai modellezés

## 2. Szabadformájú görbék

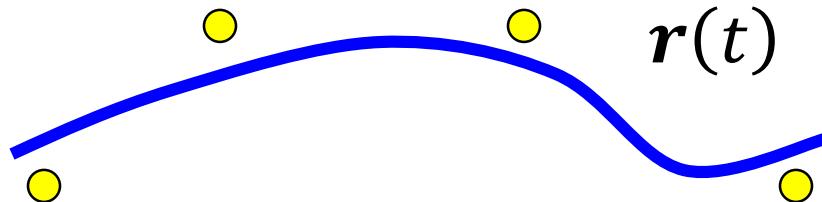
Szirmay-Kalos László



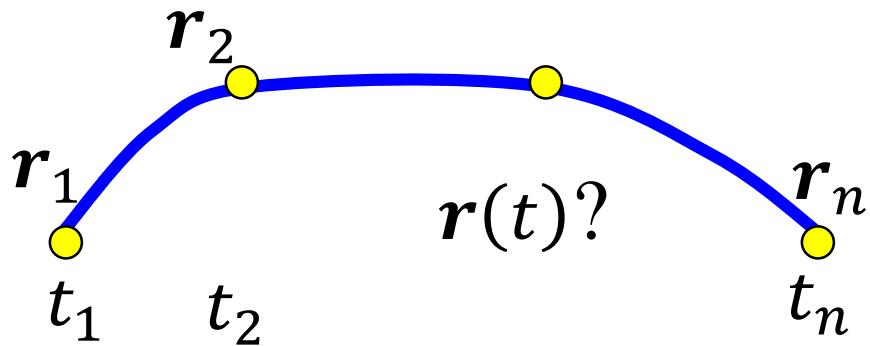
# Szabadformájú görbék



- Definíció vezérlőpontokkal



- Polinom:  $x(t) = \sum_i a_i t^i, y(t) = \sum_i b_i t^i, z(t) = \dots$
- Polinom együtthatók:
  - Kövesse a vezérlőpontokat: **Interpoláció/Approximáció**
  - Természetesség:**  $C^2$  folytonosság
  - Szépség:** kis görbületváltozás indokolatlan hullámzás nélkül
  - Független legyen a koordinátarendszertől (súlypont)
  - Lokális vezérelhetőség



## (Giuseppe) Lagrange interpoláció



- Keresd:  $\mathbf{r}(t) = (\sum_i a_i t^i, \sum_i b_i t^i, \sum_i c_i t^i)$ , amelyre  $\mathbf{r}(t_1) = \mathbf{r}_1$ ,  $\mathbf{r}(t_2) = \mathbf{r}_2$ , ...,  $\mathbf{r}(t_n) = \mathbf{r}_n$
- Hányad fokú a polinom?  $n - 1$
- Megoldás:

$$\mathbf{r}(t) = \sum_i L_i(t) \mathbf{r}_i \quad \rightarrow \quad \mathbf{r}(\mathbf{t}_k) = \sum_i L_i(\mathbf{t}_k) \mathbf{r}_i = \mathbf{r}_k$$

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

$$L_i(\mathbf{t}_k) = \frac{\prod_{j \neq i} (\mathbf{t}_k - t_j)}{\prod_{j \neq i} (t_i - t_j)} \begin{cases} 1 \text{ ha } i = k \\ 0 \text{ ha } i \neq k \end{cases}$$

# LagrangeCurve

```
class LagrangeCurve {
```

```
    vector<vec3> cps; // control pts  
    vector<float> ts; // knots
```

```
    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(vec3 cp) {
```

```
        float ti = cps.size(); // or something better
```

```
        cps.push_back(cp); ts.push_back(ti);
```

```
}
```

```
    vec3 r(float t) {
```

```
        vec3 rt(0, 0, 0);
```

```
        for(int i=; i < cps.size(); i++) rt += cps[i] * L(i,t);
```

```
        return rt;
```

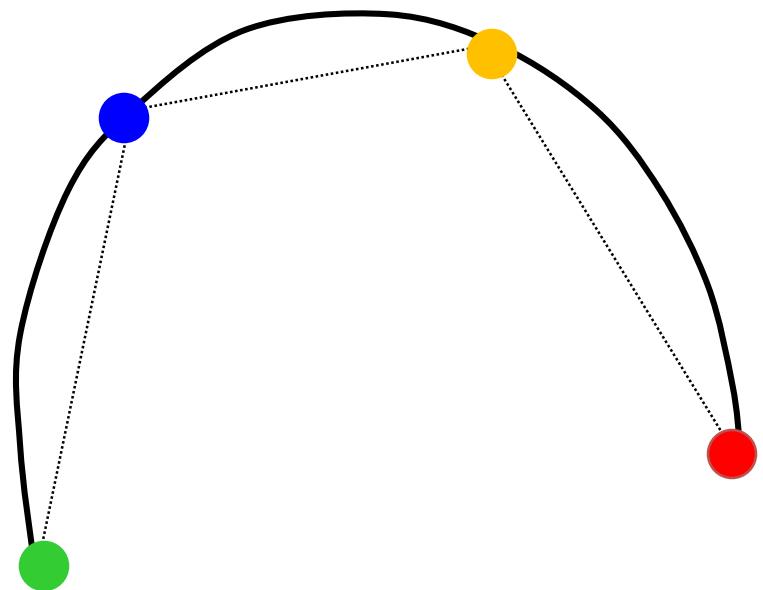
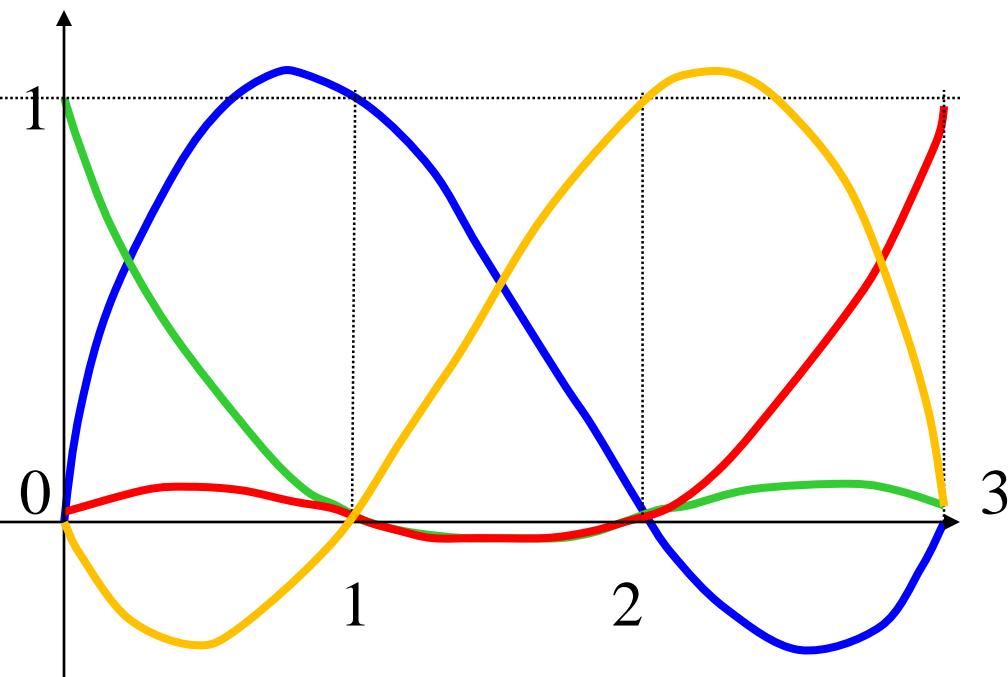
```
}
```

```
};
```

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

$$\mathbf{r}(t) = \sum_i L_i(t) \mathbf{r}_i$$

# Lagrange interpoláció bázisfüggvényei

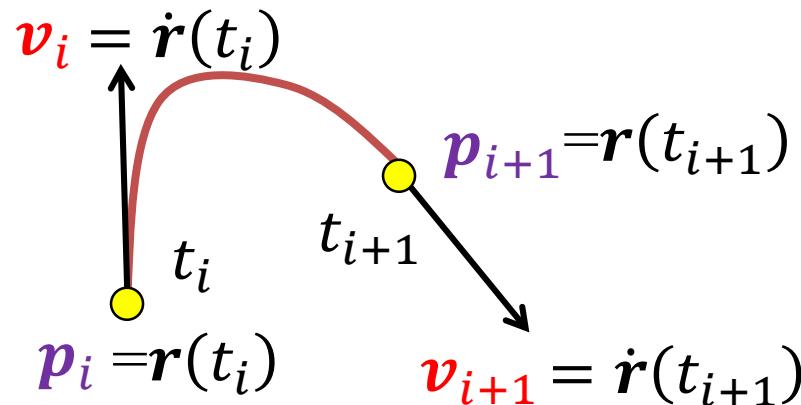


$$L_i(t) = \frac{\prod_{j \neq i} (t - t_j)}{\prod_{j \neq i} (t_i - t_j)}$$

$$\mathbf{r}(t) = \sum_i L_i(t) \mathbf{r}_i$$



# (Charles) Hermite interpoláció



$$\begin{aligned}a_0 &= p_i \\a_1 &= v_i \\a_2 &= \frac{3(p_{i+1} - p_i)}{(t_{i+1} - t_i)^2} - \frac{(v_{i+1} + 2v_i)}{t_{i+1} - t_i} \\a_3 &= \frac{2(p_i - p_{i+1})}{(t_{i+1} - t_i)^3} + \frac{(v_{i+1} + v_i)}{(t_{i+1} - t_i)^2}\end{aligned}$$

- $\mathbf{r}(t) = a_3(t - t_i)^3 + a_2(t - t_i)^2 + a_1(t - t_i) + a_0$
- $\dot{\mathbf{r}}(t) = 3a_3(t - t_i)^2 + 2a_2(t - t_i) + a_1$

$$\mathbf{r}(t_i) = a_0 = p_i$$

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

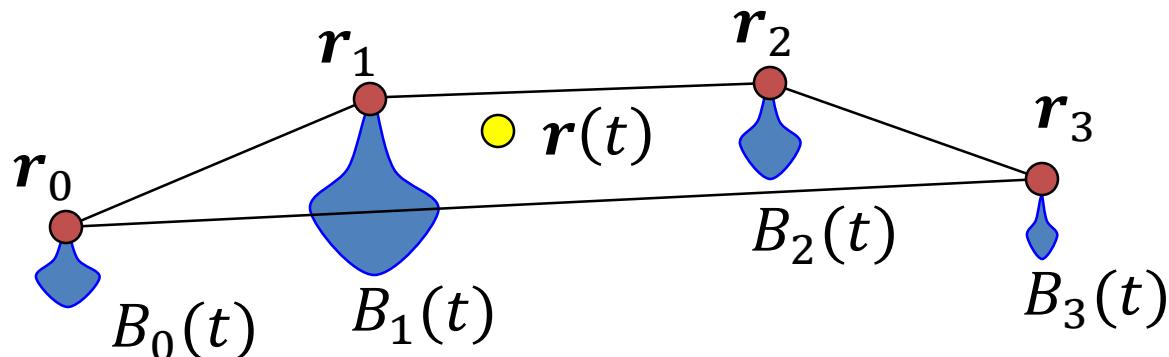
$$\dot{\mathbf{r}}(t_i) = a_1 = v_i$$

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

# (Pierre) Bézier approximáció



- Keresd:  $\mathbf{r}(t) = \sum_i B_i(t) \mathbf{r}_i$ 
  - $B_i(t)$ : ne oszcilláljon
  - Konvex burok tulajdonság
  - $B_i(t) \geq 0, \quad \sum_i B_i(t) = 1$



# (Сергéй Натáнович) Bernstein polinomok



Newton binomiális téTEL

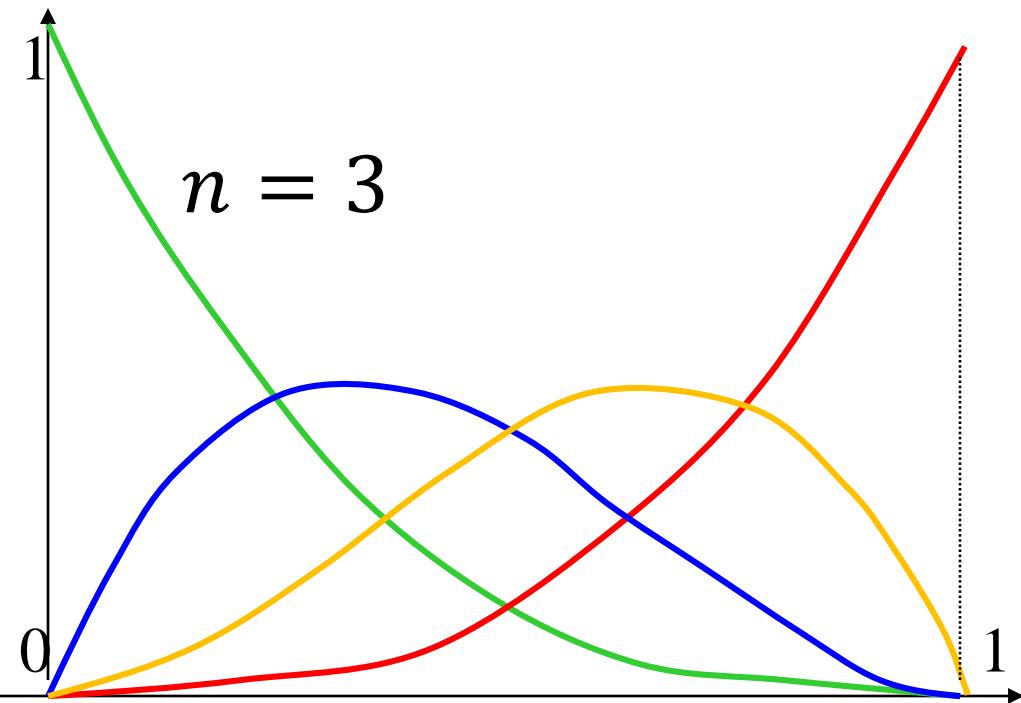
$$1^n = (t + (1 - t))^n = \sum_{i=0}^n \binom{n}{i} t^i (1 - t)^{n-i}$$

$B_i(t)$

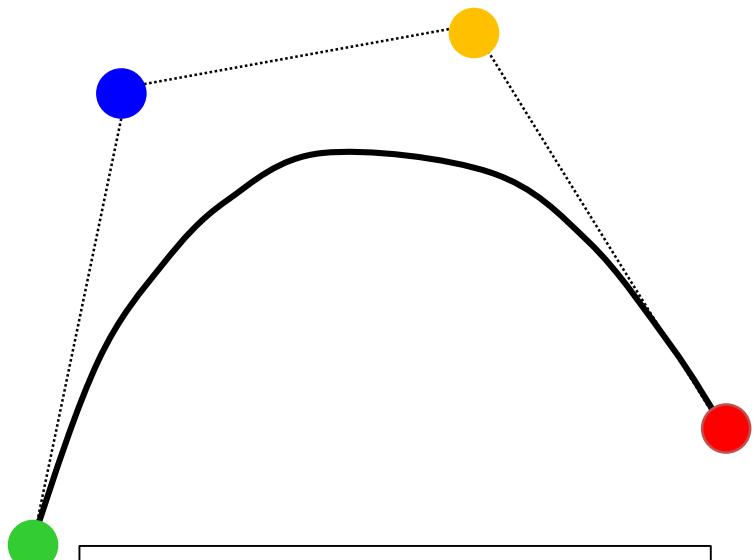
A diagram showing a bracket under the term  $\binom{n}{i} t^i (1 - t)^{n-i}$  from the binomial expansion, which is then labeled  $B_i(t)$  below it.

$B_i(t) \geq 0, \sum_i B_i(t) = 1 : \text{OK}$

# Bézier approximáció



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



$$\mathbf{r}(t) = \sum_{i=0}^n B_i(t) \mathbf{r}_i$$

# BezierCurve

```
class BezierCurve {  
    vector<vec3> cps;// control pts  
    float B(int i, float t) {  
        int n = cps.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, n-i);  
    }  
public:  
    void AddControlPoint(vec3 cp) { cps.push_back(cp); }  
  
    vec3 r(float t) {  
        vec3 rt(0, 0, 0);  
        for(int i=0; i < cps.size(); i++) rt += cps[i] * B(i,t);  
        return rt;  
    }  
};
```

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

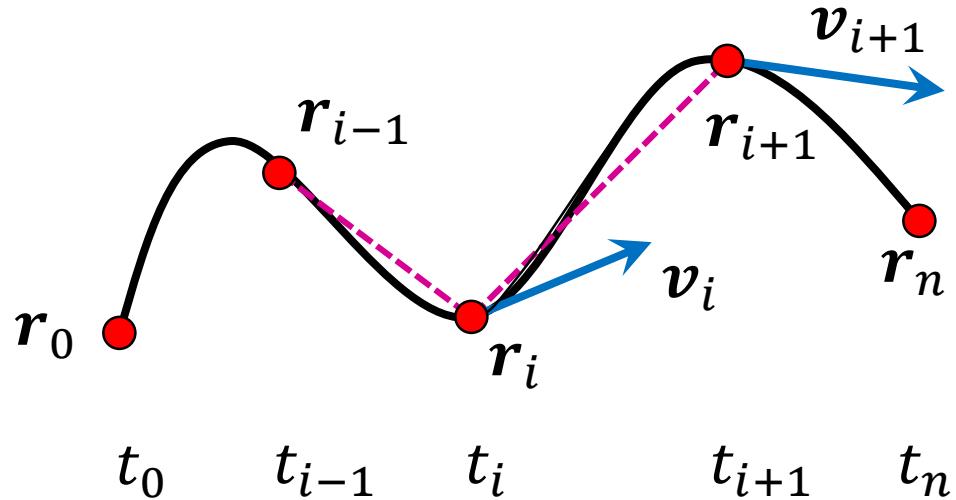
$$\mathbf{r}(t) = \sum_{i=0}^n B_i(t) \mathbf{r}_i$$



# Catmull-Rom spline



- minden két vezérlőpont közé egy Hermite
- $C^1$  simaság: a sebesség is legyen közös két egymás utánira
- közelítő  $C^2$  simaság: A közös sebességet úgy válaszd meg, hogy a gyorsulás is közelítőleg folytonos legyen



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

# CatmullRom

```
class CatmullRom {  
    vector<vec3> cps; // control points  
    vector<float> ts; // parameter (knot) values  
  
    vec3 Hermite( vec3 p0, vec3 v0, float t0,  
                  vec3 p1, vec3 v1, float t1, float t ) {  
  
        r(t) = a3(t - t0)3 + a2(t - t0)2 + a1(t - t0) + a0  
    }  
  
public:  
    void AddControlPoint(vec3 cp, float t) { ... }  
  
    vec3 r(float t) {  
        for(int i = 0; i < cps.size() - 1; i++)  
            if (ts[i] <= t && t <= ts[i+1]) {  
                vec3 v0 = ..., v1 = ...;  
                return Hermite(cps[i], v0, ts[i],  
                               cps[i+1], v1, ts[i+1], t);  
            }  
    }  
};
```

$$\begin{aligned} a_0 &= \mathbf{p}_i, \quad a_1 = \mathbf{v}_i \\ a_2 &= \frac{3(\mathbf{p}_{i+1} - \mathbf{p}_i)}{(t_{i+1} - t_i)^2} - \frac{(\mathbf{v}_{i+1} + 2\mathbf{v}_i)}{t_{i+1} - t_i} \\ a_3 &= \frac{2(\mathbf{p}_i - \mathbf{p}_{i+1})}{(t_{i+1} - t_i)^3} + \frac{(\mathbf{v}_{i+1} + \mathbf{v}_i)}{(t_{i+1} - t_i)^2} \end{aligned}$$

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



# CatmullRom, 2. verzió

```
class CatmullRom {  
    vector<vec3> cps;           // control points  
    vector<float> ts;          // parameter (knot) values  
  
    vec3 Seg(vec3 p_1, float t_1, vec3 p0, float t0,  
             vec3 p1, float t1, vec3 p2, float t2, float t){  
        float c_1 = ..., c0 = ..., c1 = ..., c2 = ...; // t_i, t  
        return p_1 * c_1 + p0 * c0 + p1 * c1 + p2 * c2;  
    }  
  
public:  
    void AddControlPoint(vec3 cp, float t) { ... }  
  
    vec3 r(float t) {  
        for(int i = 0; i < cps.size() - 1; i++)  
            if (ts[i] <= t && t <= ts[i+1]) {  
                // Túlcímzést lekezelni!  
                return Seg(cps[i-1],ts[i-1], cps[i],ts[i],  
                           cps[i+1],ts[i+1], cps[i+2],ts[i+2], t);  
            }  
    }  
};
```



# Szabadformájú görbék

- Paraméteres egyenlet (mozgás), polinomok
- Kontrolpontokkal definiáljuk (approximációs, interpolációs)
- Görbe = kontrolpontok kombinációja (súlypont)
- Görbe tulajdonságait a súlyfüggvények határozzák meg
  - Folytonosság ( $C^0, C^1, C^2$ )
  - Konvex burok: súlyfüggvények nem negatívak
  - Lokális vezérelhetőség: súlyfüggvények a tartomány egy részében nem zérus értékűek

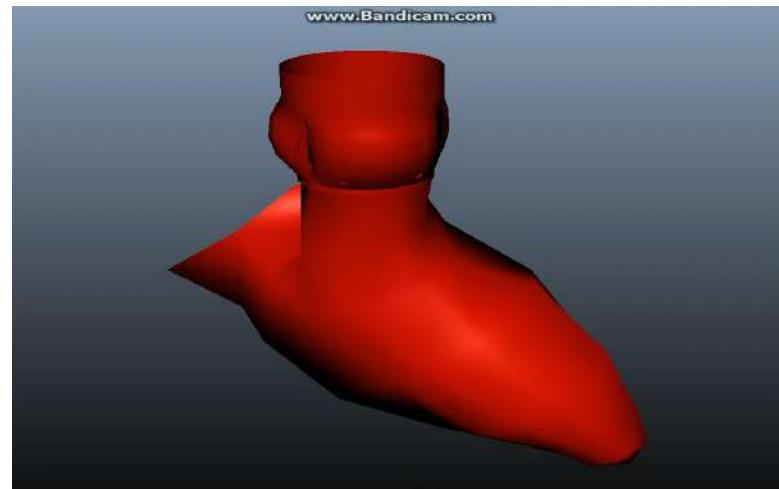
*"La semplicità è la sofisticazione finale."*

*Leonardo da Vinci*

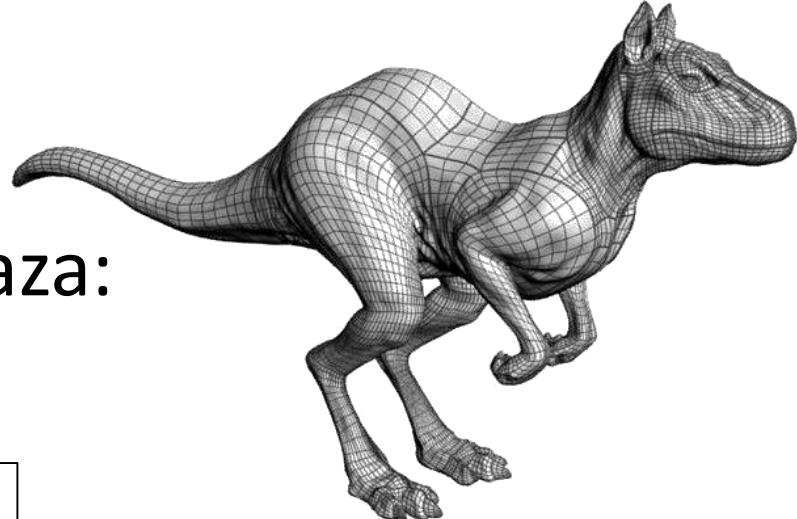
# Geometriai modellezés

## 3. Felületek

Szirmay-Kalos László



# Felületek



Felület a 3D tér 2D részhalmaza:

– Explicit:

$$z = h(x, y)$$

– Implicit:

$$f(x, y, z) = 0$$

– gömb:  $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - R^2 = 0$

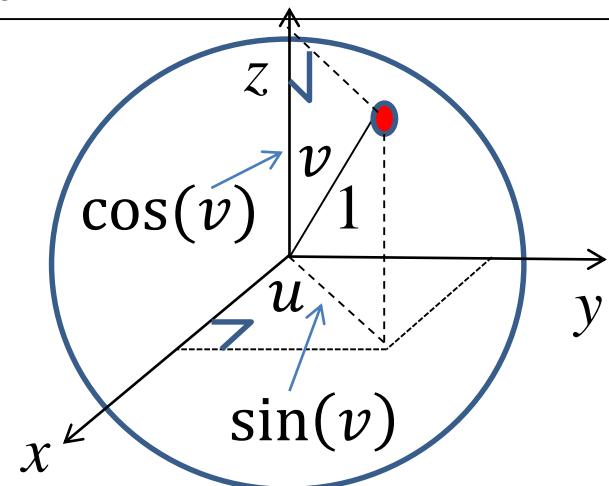
– sík:  $ax + by + cz + d = 0$

– Parametrikus:

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

– gömb:

$$\begin{aligned}x(u, v) &= c_x + R \cos(u) \sin(v) \\y(u, v) &= c_y + R \sin(u) \sin(v) \\z(u, v) &= c_z + R \cos(v) \\u &\in [0, 2\pi), v \in [0, \pi)\end{aligned}$$



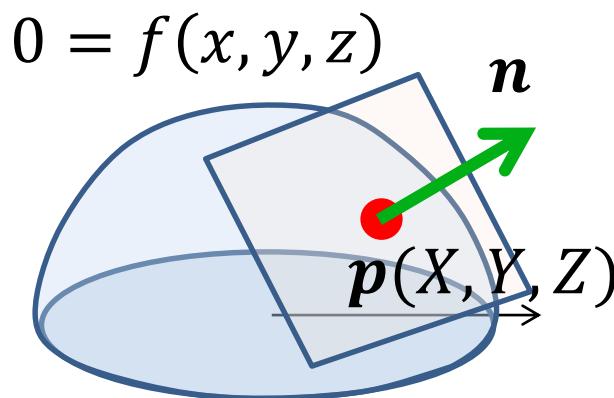
# Implicit felületek normálvektora

$$\text{Normál vektor} = \text{grad } f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

$$0 = f(x, y, z)$$

$$= f(X + (x - X), Y + (y - Y), Z + (z - Z))$$

$$\approx \cancel{f(X, Y, Z)} + \frac{\partial f}{\partial x}(x - X) + \frac{\partial f}{\partial y}(y - Y) + \frac{\partial f}{\partial z}(z - Z)$$



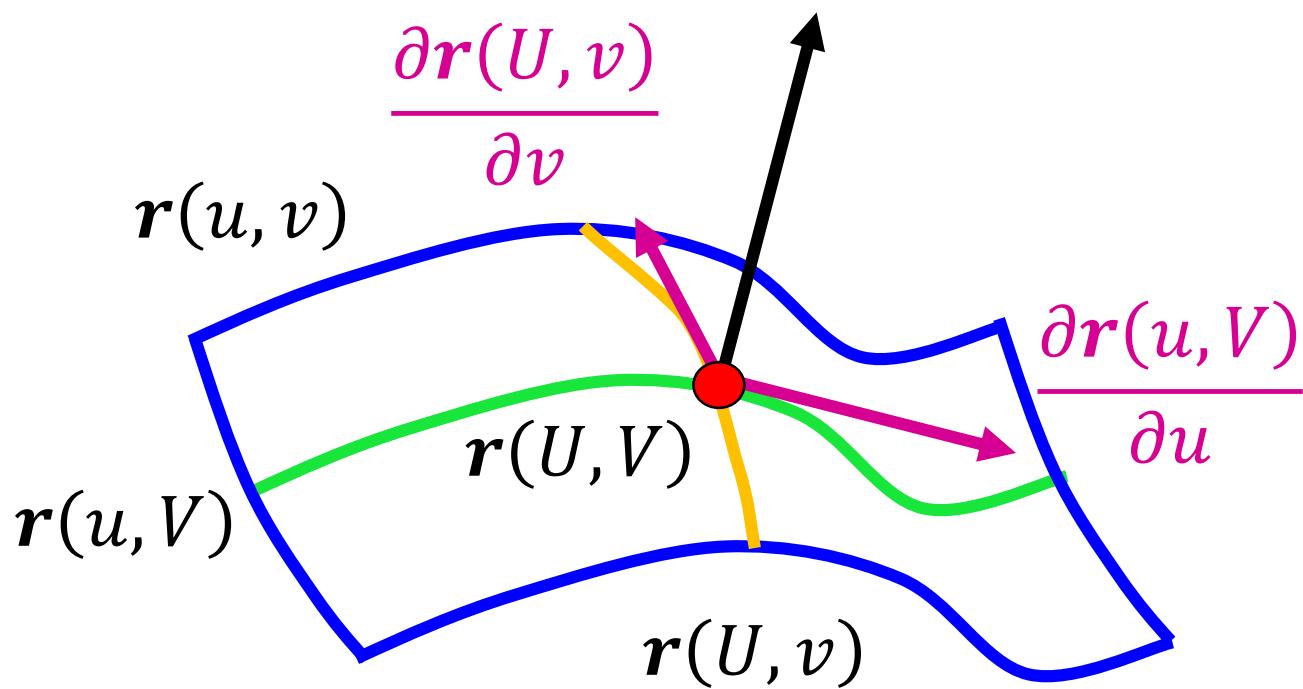
$$\left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \cdot (x - X, y - Y, z - Z) = 0$$

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

# Parametrikus felületek normálvektora

$$\mathbf{N}(U, V) = \frac{\partial \mathbf{r}(u, v)}{\partial u} \times \frac{\partial \mathbf{r}(u, v)}{\partial v}$$

$$u = U \\ v = V$$

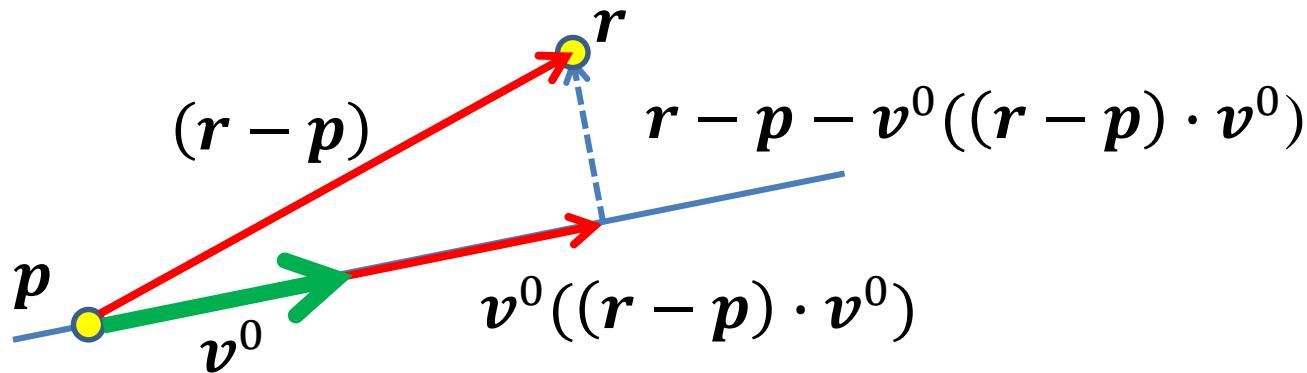


# Implicit kvadratikus felületek

- **Gömb:** Azon  $\mathbf{r}(x,y)$  pontok mértani helye, amelyek a  $\mathbf{c}(c_x, c_y)$  középponttól  $R$  távolságra vannak:  $|\mathbf{r} - \mathbf{c}| = R$
- **Henger:** Azon  $\mathbf{r}$  pontok, amelyek a  $\mathbf{v}^0$  irányvektorú és  $\mathbf{p}$  helyvektorú egyenestől mért távolsága  $R$ :  $|\mathbf{r} - \mathbf{p} - \mathbf{v}^0((\mathbf{r} - \mathbf{p}) \cdot \mathbf{v}^0)| = R$

# Implicit kvadratikus felületek

- **Gömb:** Azon  $\mathbf{r}(x,y)$  pontok mértani helye, amelyek a  $\mathbf{c}(c_x, c_y)$  középponttól  $R$  távolságra vannak:  $|\mathbf{r} - \mathbf{c}| = R$
- **Henger:** Azon  $\mathbf{r}$  pontok, amelyek a  $\mathbf{v}^0$  irányvektorú és  $\mathbf{p}$  helyvektorú egyenestől mért távolsága  $R$ :  $|\mathbf{r} - \mathbf{p} - \mathbf{v}^0((\mathbf{r} - \mathbf{p}) \cdot \mathbf{v}^0)| = R$



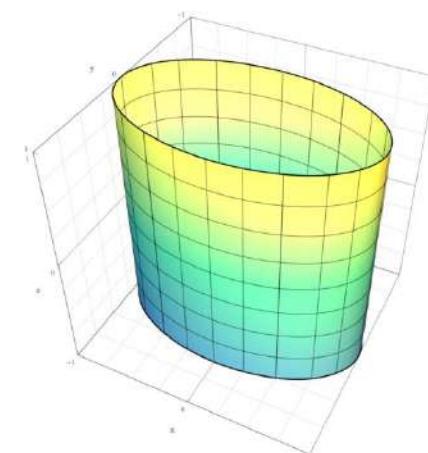
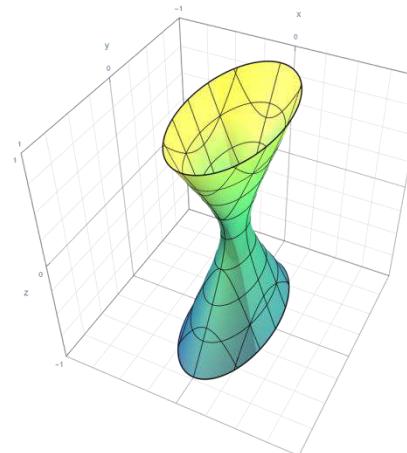
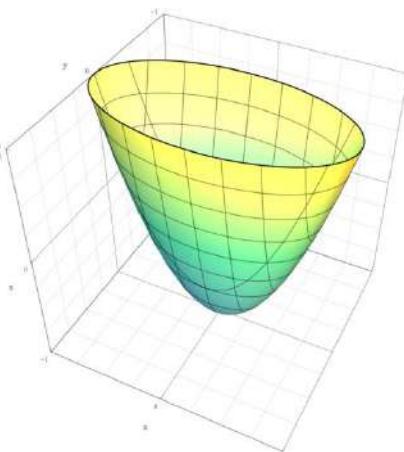
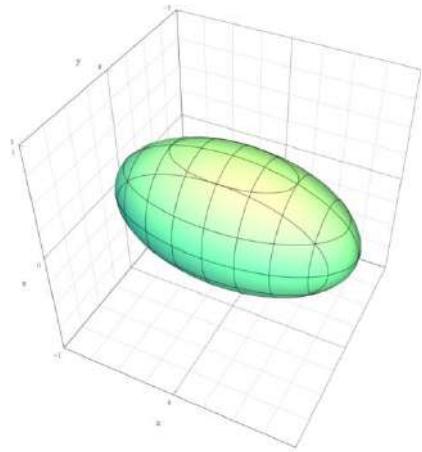
# Implicit kvadratikus felületek

- **Gömb:** Azon  $\mathbf{r}(x,y)$  pontok mértani helye, amelyek a  $\mathbf{c}(c_x, c_y)$  középponttól  $R$  távolságra vannak:  $|\mathbf{r} - \mathbf{c}| = R$
- **Henger:** Azon  $\mathbf{r}$  pontok, amelyek a  $\mathbf{v}^0$  irányvektorú és  $\mathbf{p}$  helyvektorú egyenestől mért távolsága  $R$ :  $|\mathbf{r} - \mathbf{p} - \mathbf{v}^0((\mathbf{r} - \mathbf{p}) \cdot \mathbf{v}^0)| = R$
- **Ellipszoid:** Azon  $\mathbf{r}$  pontok, amelyek a  $\mathbf{f}_1$  és  $\mathbf{f}_2$  fókuszpontuktól mért távolság összege állandó  $C$ :  $|\mathbf{r} - \mathbf{f}_1| + |\mathbf{r} - \mathbf{f}_2| = C$
- **Hiperboloid:** Azon  $\mathbf{r}$  pontok, melyek a  $\mathbf{f}_1$  és  $\mathbf{f}_2$  fókuszpontuktól mért távolság különbsége állandó  $C$ :  $|\mathbf{r} - \mathbf{f}_1| - |\mathbf{r} - \mathbf{f}_2| = C$
- **Paraboloid:** Azon  $\mathbf{r}$  pontok, amelyek az  $\mathbf{f}$  fókuszponttól mért távolsága megegyezik az  $\mathbf{n}$  normálvektorú és  $\mathbf{p}$  helyvektorú síktól mért távolsággal:  $|\mathbf{r} - \mathbf{f}| = |\mathbf{n}^0 \cdot (\mathbf{r} - \mathbf{p})|$

# Implicit kvadratikus felületek

$$f(x, y, z) = [x, y, z, 1] \mathbf{Q} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

Mindig felírható úgy is,  
hogy  $\mathbf{Q}$  szimmetrikus



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

Ellipszoid

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

Paraboloid

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

Hiperboloid

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

Elliptikus  
henger

# Kvadratikus objektum

$$f(x, y, z) = [x, y, z, 1] \mathbf{Q} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
$$\frac{\partial f}{\partial x} = [1, 0, 0, 0] \mathbf{Q} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + [x, y, z, 1] \mathbf{Q} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
$$= [x, y, z, 1] \mathbf{Q} \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
struct Quadrics {
    mat4 Q; // symmetric matrix

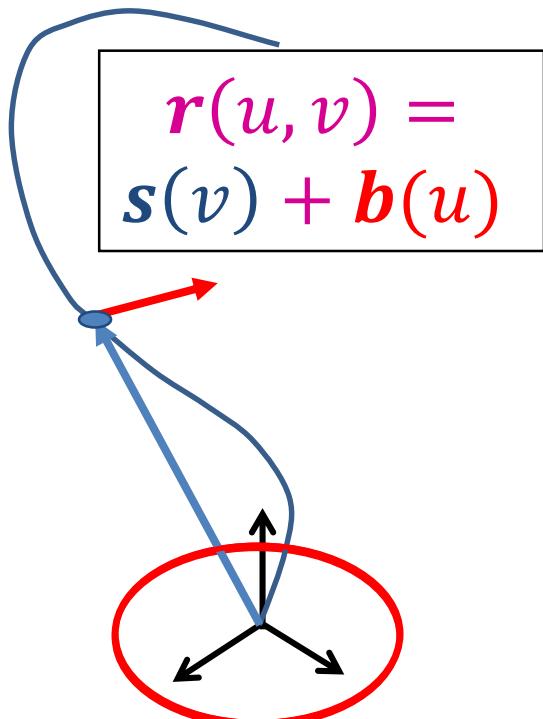
    float f(vec4 r) { // r.w = 1
        return dot(r * Q, r);
    }

    vec3 gradf(vec4 r) { // r.w = 1
        vec4 g = r * Q * 2;
        return vec3(g.x, g.y, g.z);
    }
};
```

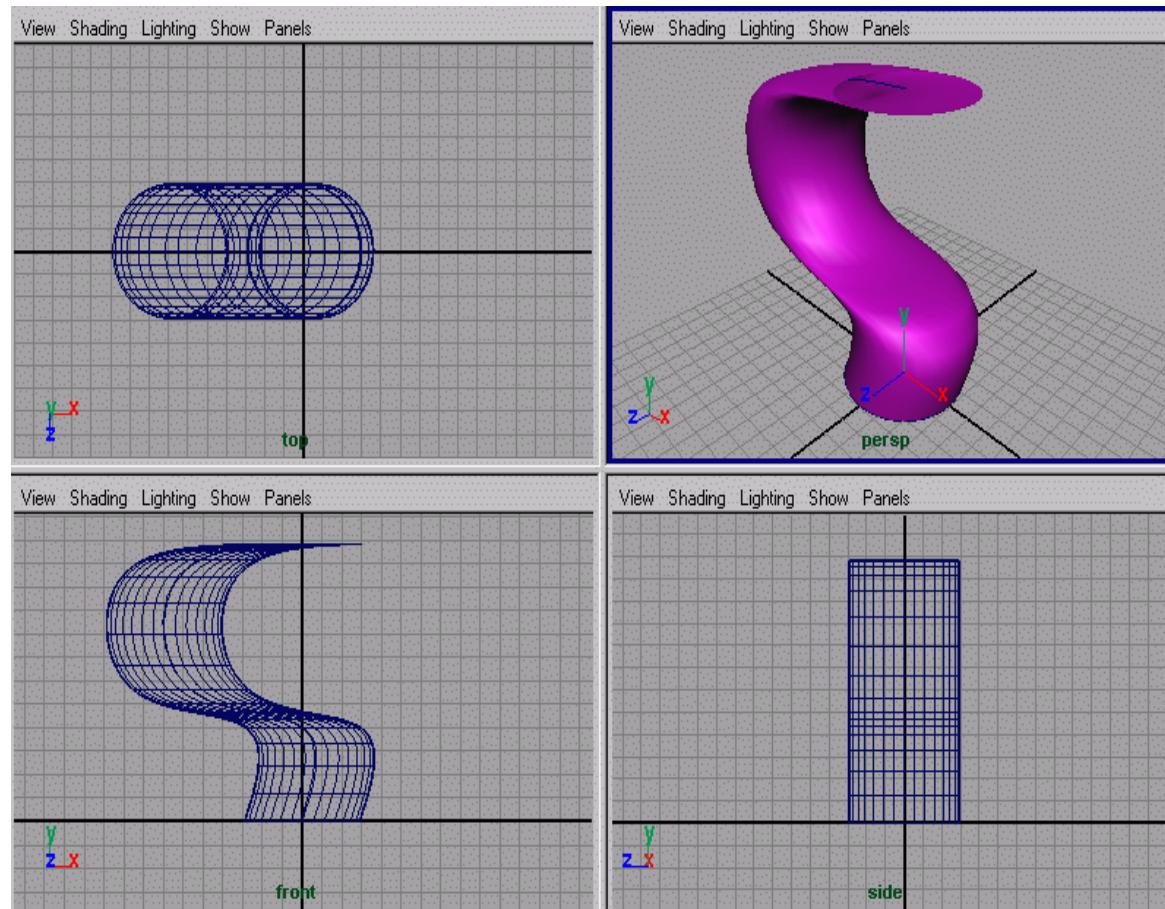


# Parametrikus felületek: Kihúzás

$s(v)$ : gerinc

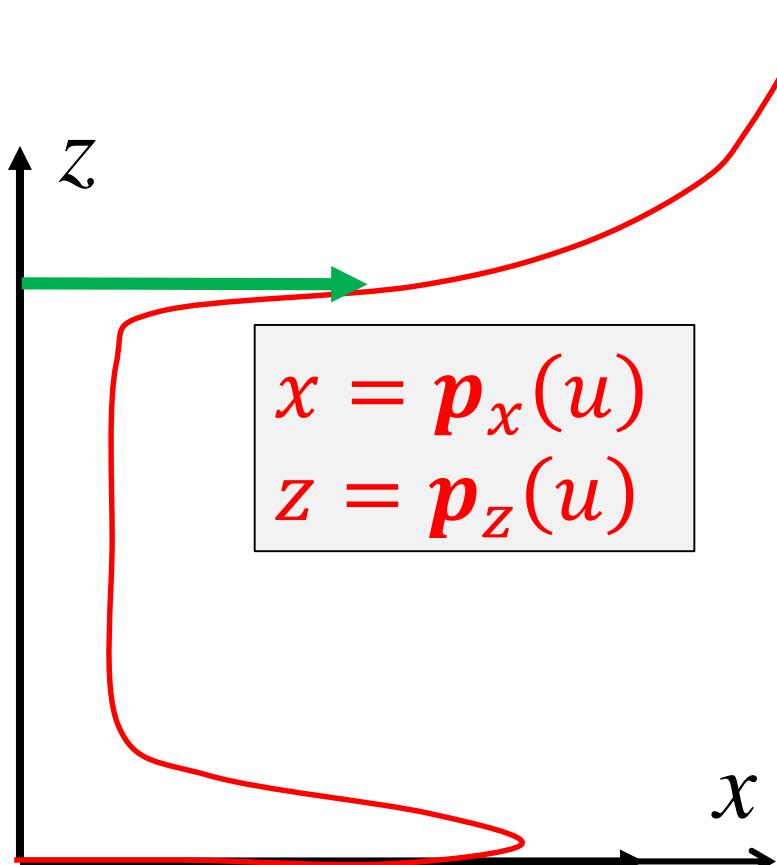


$b(u)$ : profil

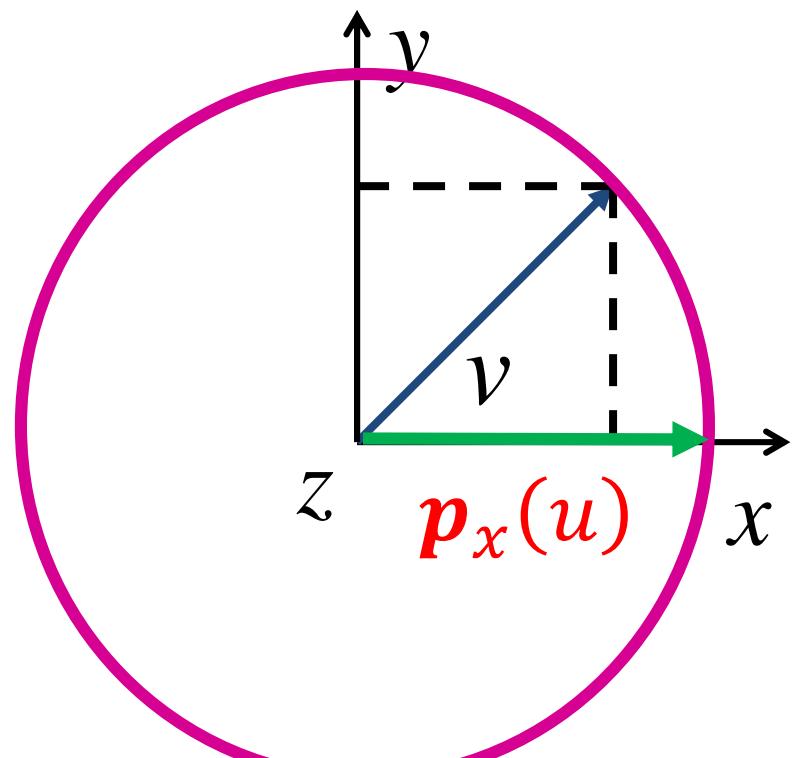


# Parametrikus felületek: Forgalás

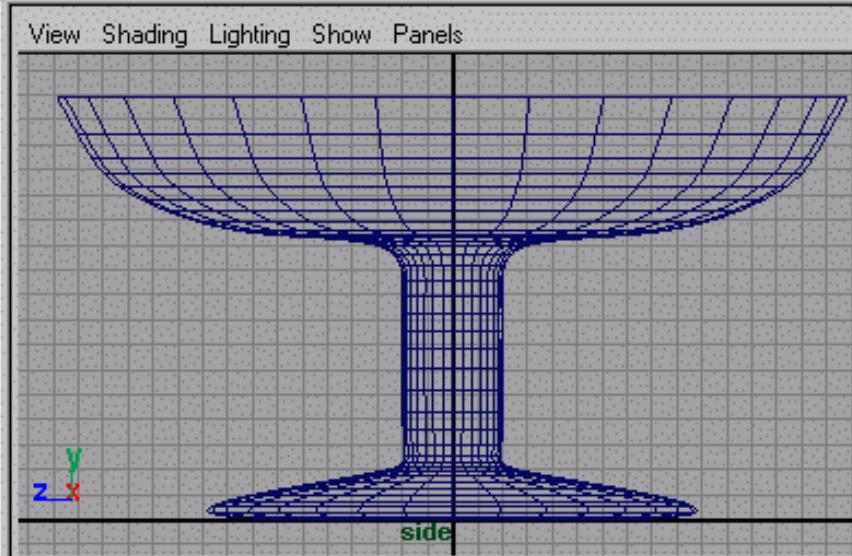
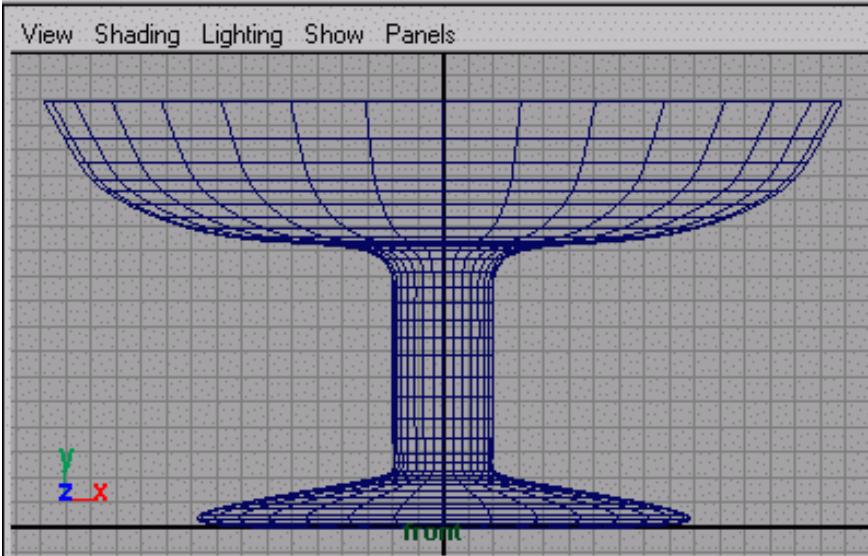
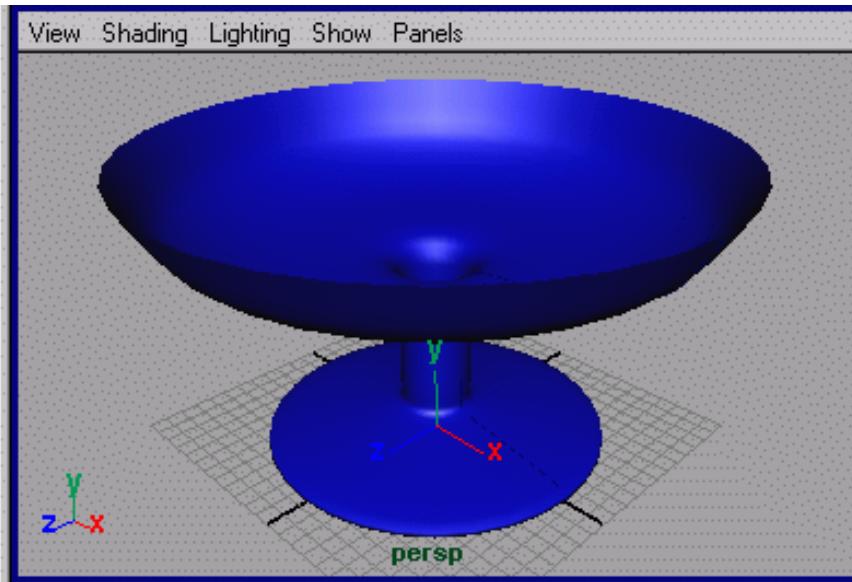
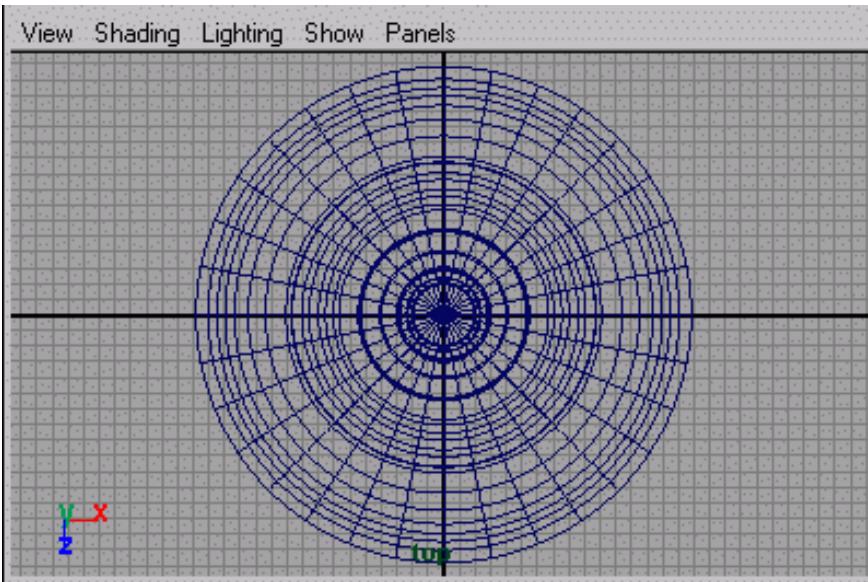
Oldalnézet



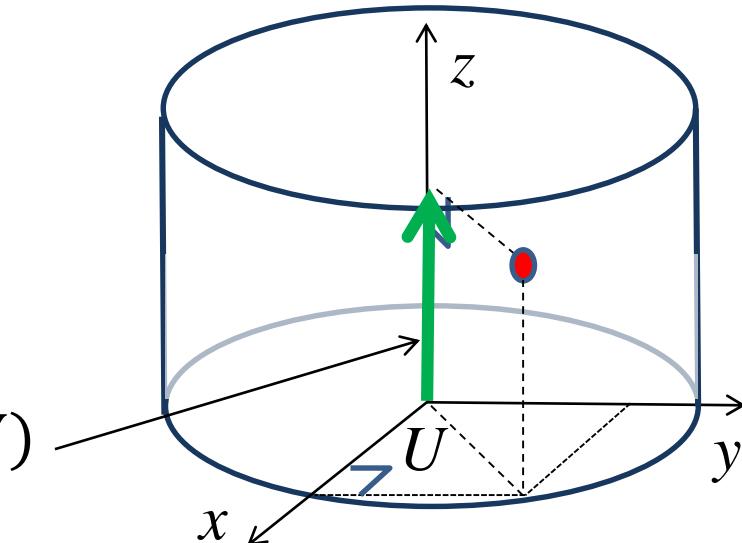
Felülnézet



# Forgatás



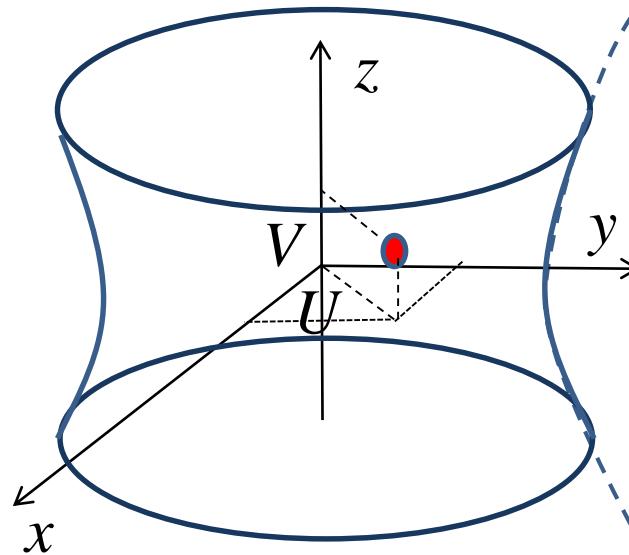
# Henger



$$\begin{aligned}x(U, V) &= r \cos(U) \\y(U, V) &= r \sin(U) \\z(U, V) &= V \\U &\in [0, 2\pi], \quad V \in [0, h]\end{aligned}$$

```
void eval(float u, float v, vec3& point, vec3& normal) {
    float U = u * 2 * M_PI, V = v * height;
    vec3 base(cos(U) * r, sin(U) * r, 0), spine(0, 0, v);
    point = base + spine;
    normal = base;
```

# Hiperboloid

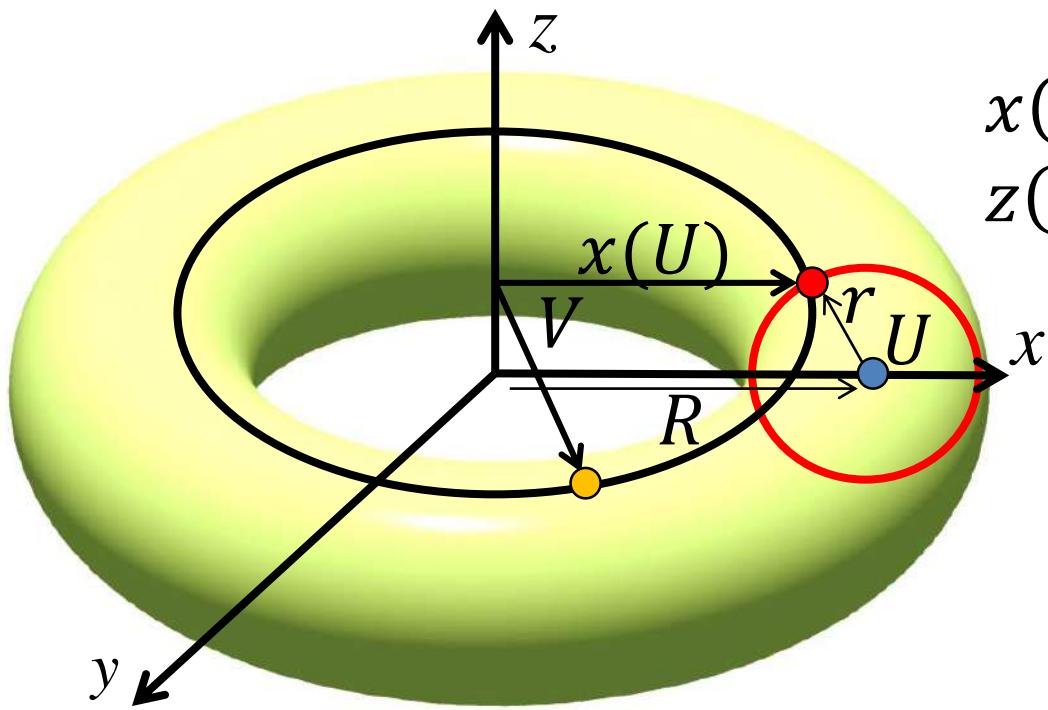


$$\begin{aligned}x(U, V) &= r \cosh(U) \cos(V) \\y(U, V) &= r \cosh(U) \sin(V) \\z(U, V) &= \sinh(U) \\V \in [0, 2\pi], \quad U \in [-h/2, h/2]\end{aligned}$$

$$\mathbf{p}(U) = (r \cosh(U), 0, \sinh(U))$$

```
void eval(float u, float v, vec3& point, vec3& normal){  
    float U = (v - 0.5f) * h, V = u * 2 * M_PI;  
    float shu=sinh(U), chu=cosh(U), cv=cos(V), sv=sin(V);  
    point = vec3(r * chu * cv, r * chu * sv, shu);  
    vec3 drdU(r * shu * cv, r * shu * sv, chu);  
    vec3 drdV(r * chu * (-sv), r * chu * cv, 0);  
    normal = cross(drdU, drdV);  
}
```

# Tórusz

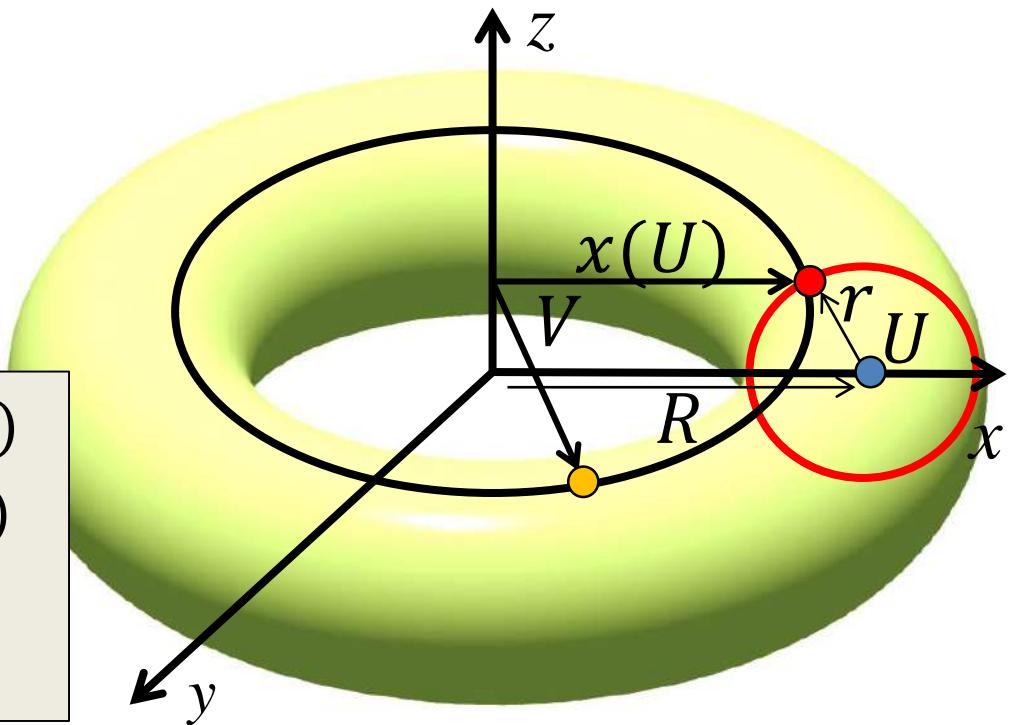


$$x(U) = R + r \cos(U)$$
$$z(U) = r \sin(U)$$

$$x(U, V) = (R + r \cos(U)) \cos(V)$$
$$y(U, V) = (R + r \cos(U)) \sin(V)$$
$$z(U, V) = r \sin(U)$$
$$U \in [0, 2\pi], \quad V \in [0, 2\pi]$$

# Tórusz

$$x(U, V) = (R + r \cos(U)) \cos(V)$$
$$y(U, V) = (R + r \cos(U)) \sin(V)$$
$$z(U, V) = r \sin(U)$$
$$U \in [0, 2\pi], \quad V \in [0, 2\pi]$$

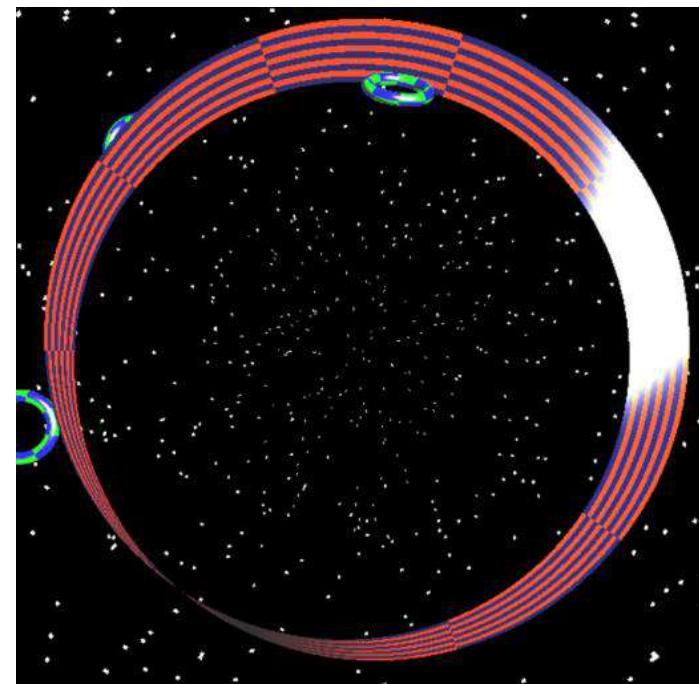


```
typedef Dnum<vec2> Dnum2;

void eval(float u, float v, vec3& point, vec3& normal){
    Dnum2 U(u*2*M_PI, vec2(1,0)), V(v*2*M_PI, vec2(0,1));
    Dnum2 D = Cos(U) * r + R;
    Dnum2 X = D * Cos(V), Y = D * Sin(V), Z = Sin(U) * r;
    point = vec3(X.f, Y.f, Z.f);
    vec3 drdU(X.d.x,Y.d.x,Z.d.x), drdV(X.d.y,Y.d.y,Z.d.y);
    normal = cross(drdU, drdV);
}
```

# Möbius

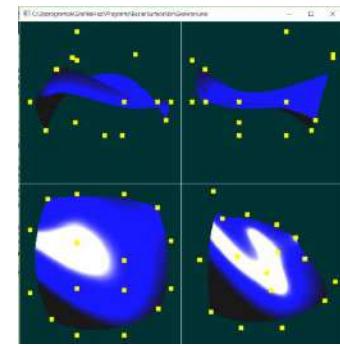
$$\begin{aligned}x(U, V) &= (R + V \cos(U)) \cos(2U) \\y(U, V) &= (R + V \cos(U)) \sin(2U) \\z(U, V) &= V \sin(U) \\U &\in [0, \pi], \quad V \in [-w/2, w/2]\end{aligned}$$



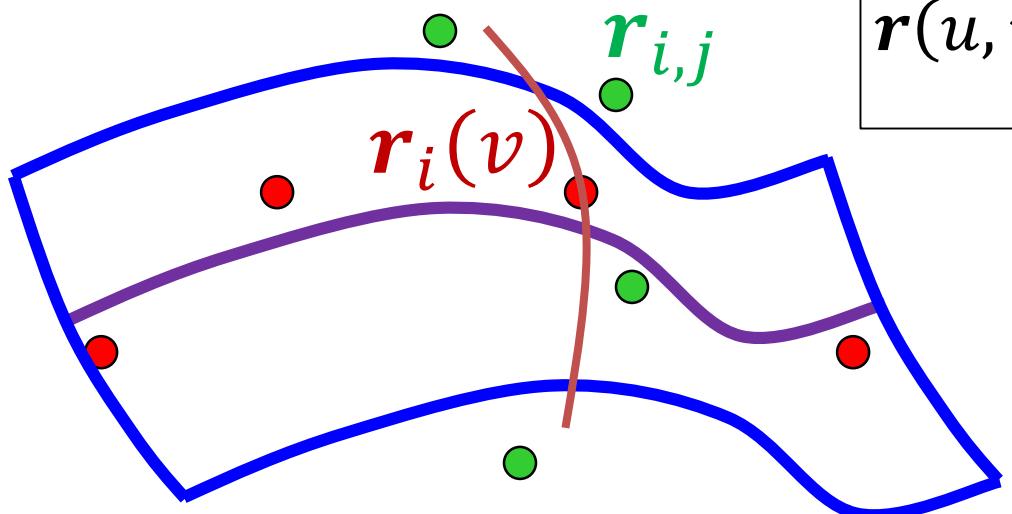
```
typedef Dnum<vec2> Dnum2;

void eval(float u, float v, vec3& point, vec3& normal){
    Dnum2 U(u*M_PI, vec2(1,0)), V((v-0.5)*w, vec2(0,1));
    Dnum2 X = (Cos(U) * V + R) * Cos(U * 2);
    Dnum2 Y = (Cos(U) * V + R) * Sin(U * 2);
    Dnum2 Z = Sin(U) * V;
    point = vec3(X.f, Y.f, Z.f);
    vec3 drdU(X.d.x,Y.d.x,Z.d.x), drdV(X.d.y,Y.d.y,Z.d.y);
    normal = cross(drdU, drdV);
}
```

# Szabadformájú felület

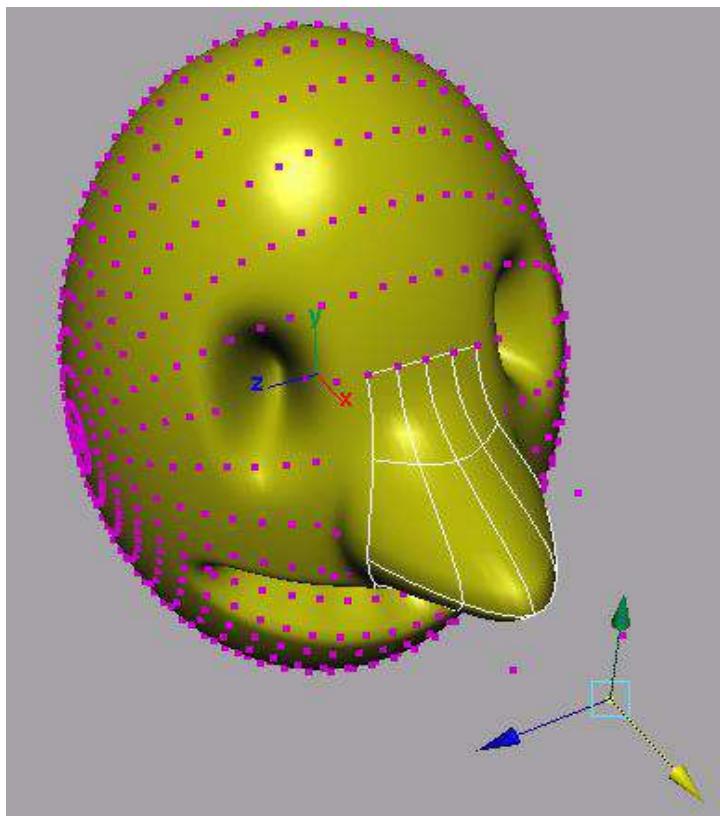


Definíció kontroll pontokkal:

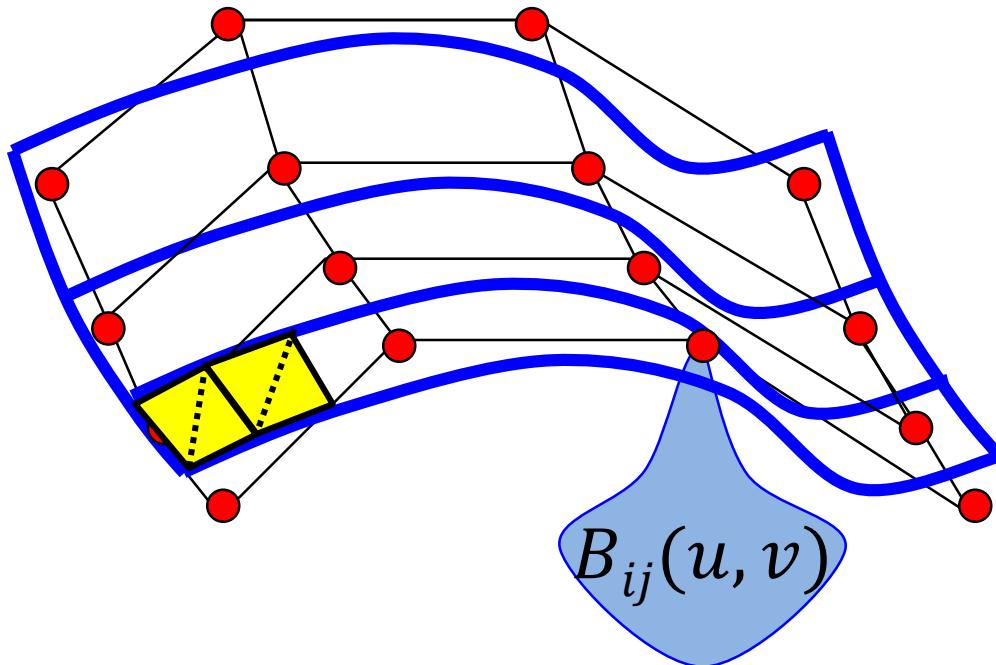


$$\mathbf{r}(u, v) = \sum_i \sum_j B_j(v) B_i(u) \mathbf{r}_{i,j}$$

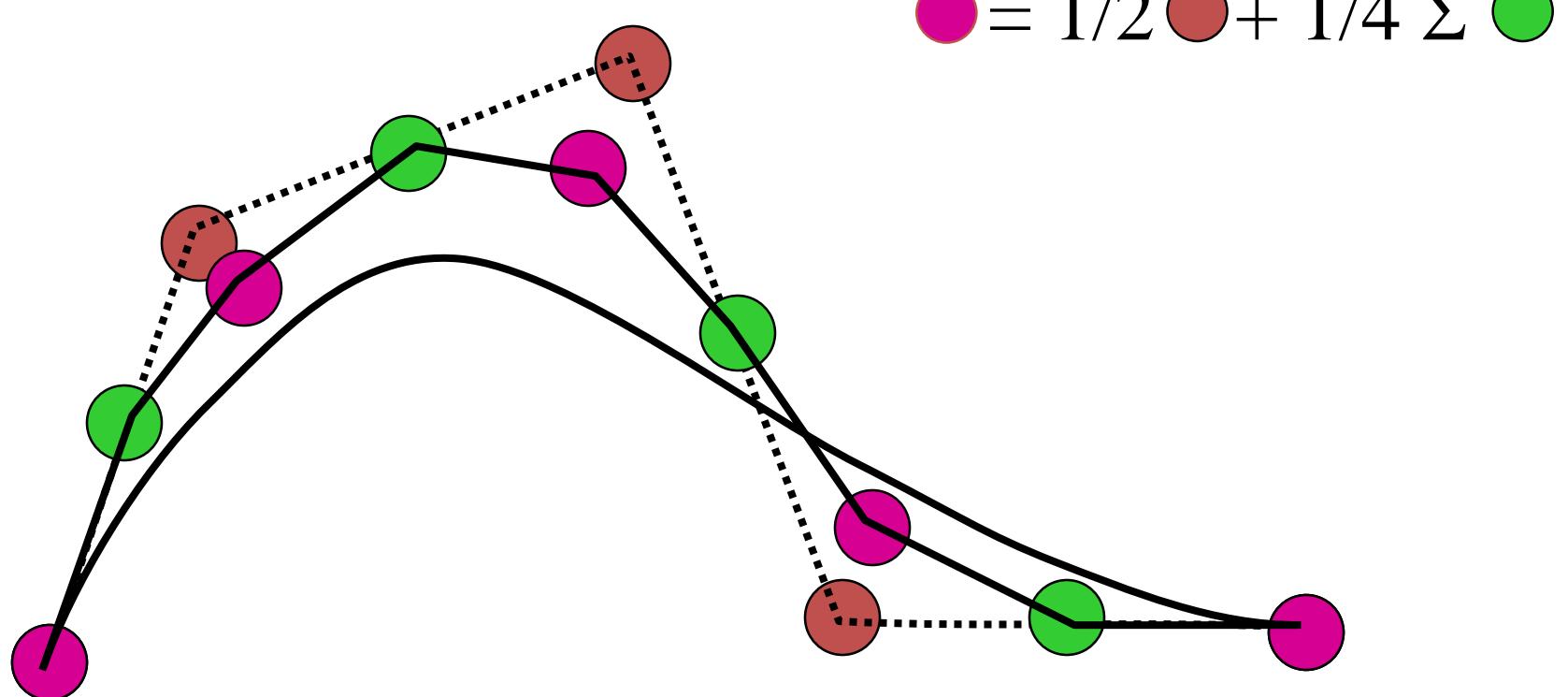
$$\mathbf{r}(u, v) = \mathbf{r}_v(u) = \sum_i B_i(u) \mathbf{r}_i(v)$$
$$\mathbf{r}_i(v) = \sum_j B_j(v) \mathbf{r}_{i,j}$$



# Poligonháló finomítása



# Subdivision görbék





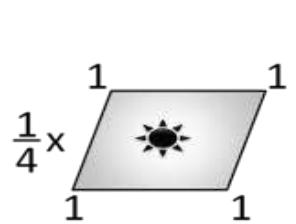
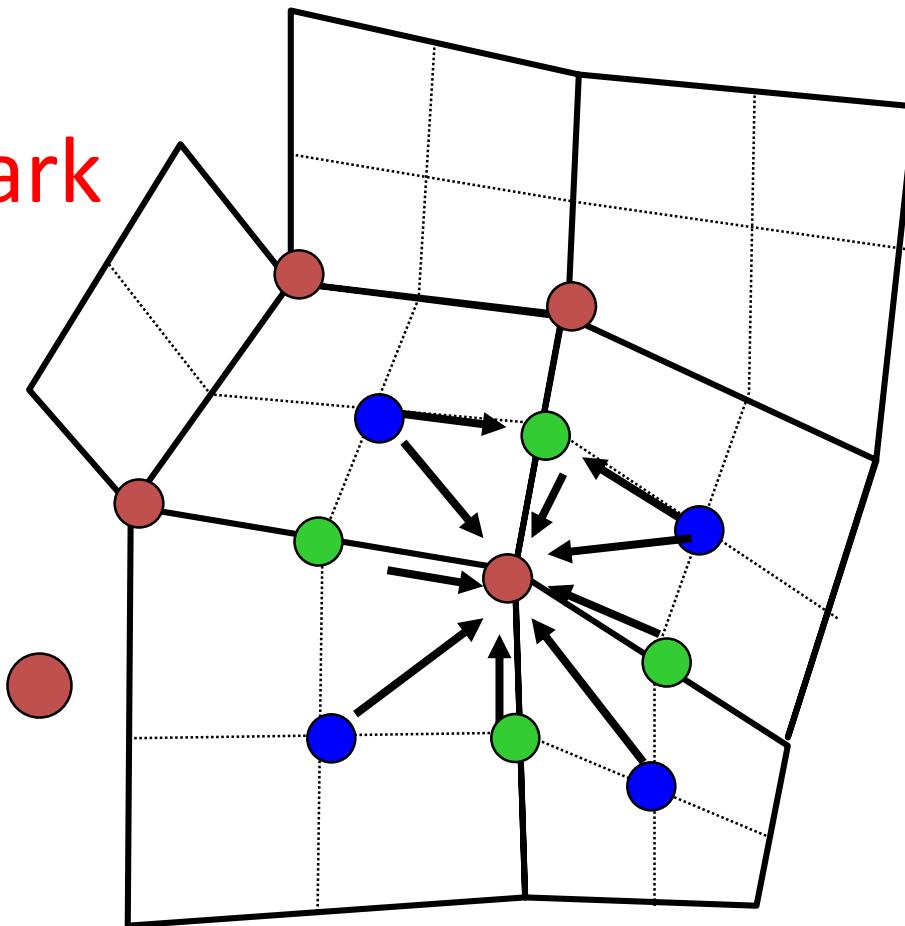
# (Edwin) Catmull- (James) Clark subdivision felület

$$\bullet = 1/4 \sum \text{red circles}$$

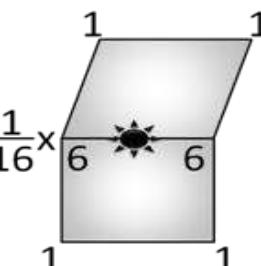
$$i = 1/2 \sum \text{red circles}$$

$$\text{red circle} = 1/v^2 \sum \bullet + 2/v^2 \sum i + (v-3)/v \text{ red circle}$$

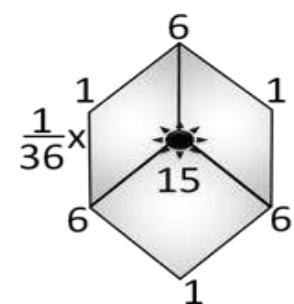
$$\text{green circle} = 1/4 \sum \bullet + 1/2 i$$



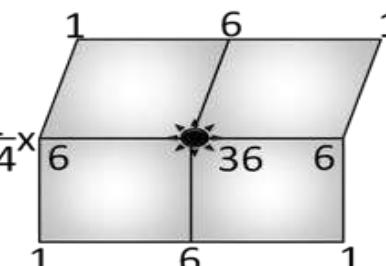
Face Point



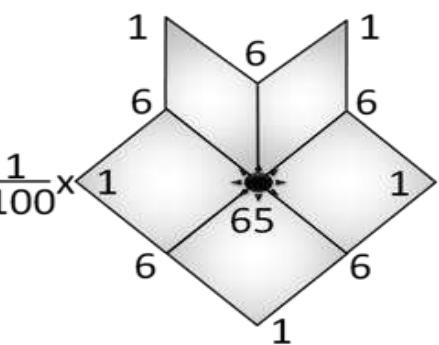
Edge Point



Valence 3 Vertex

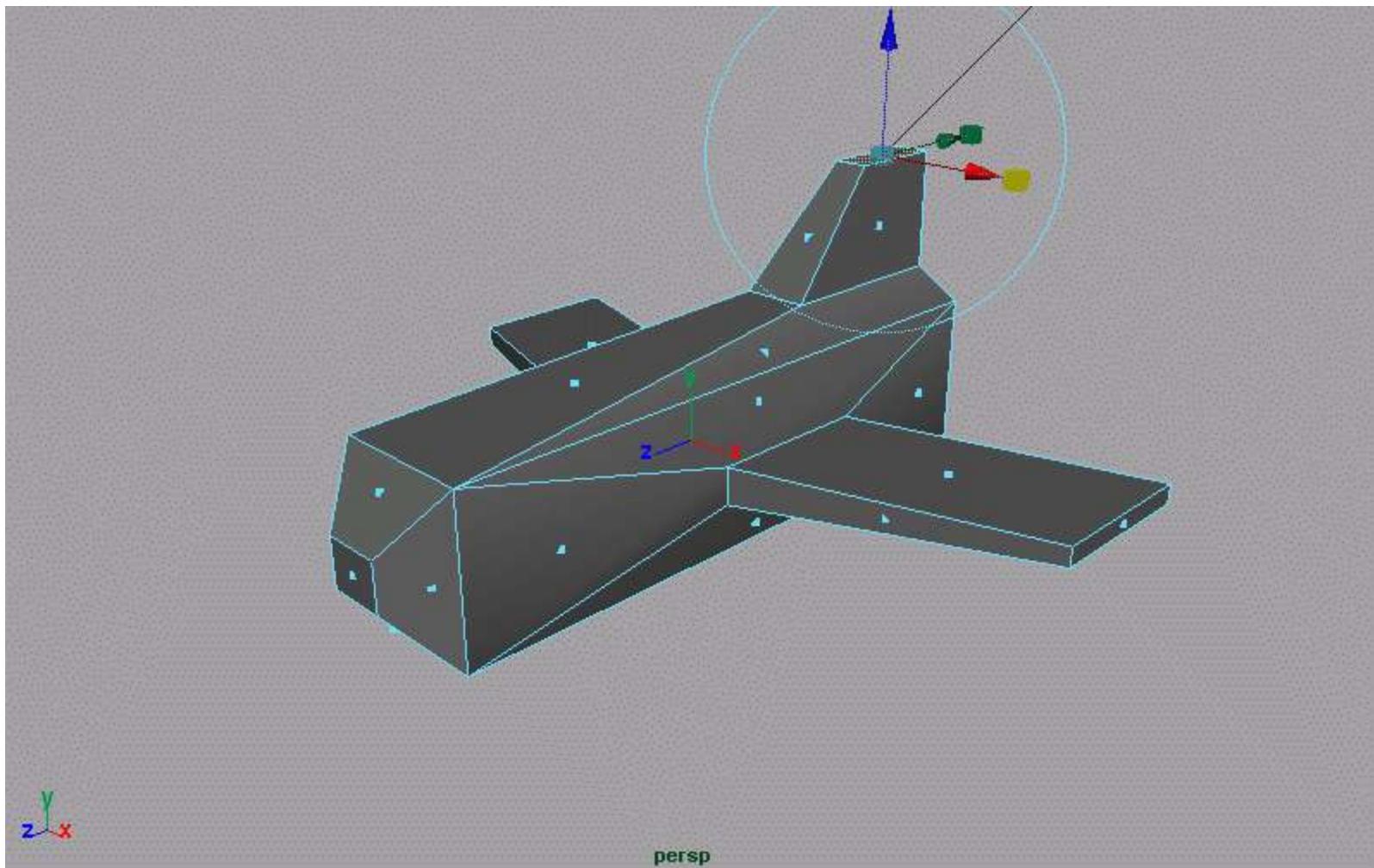


Valence 4 Vertex

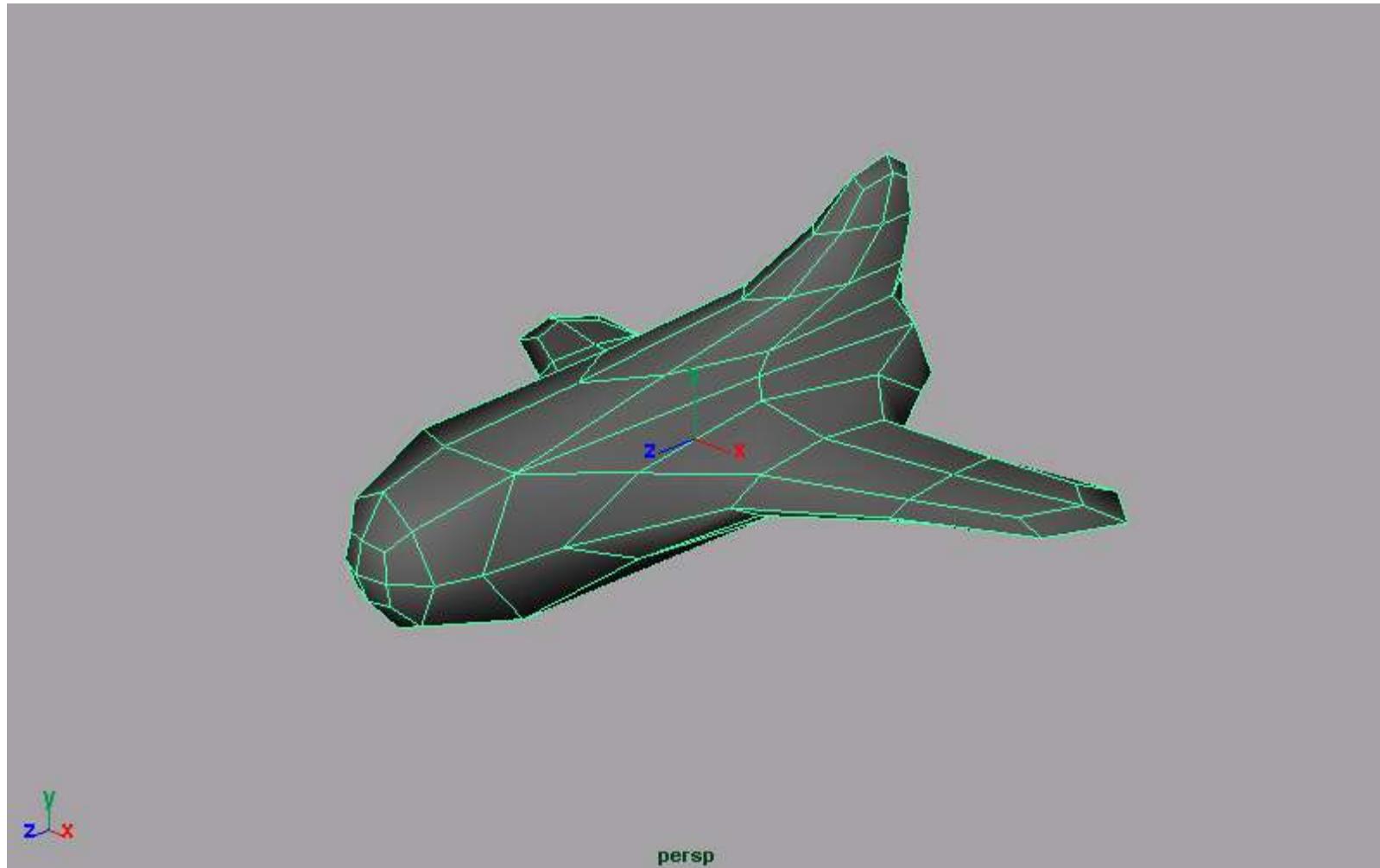


Valence 5 Vertex

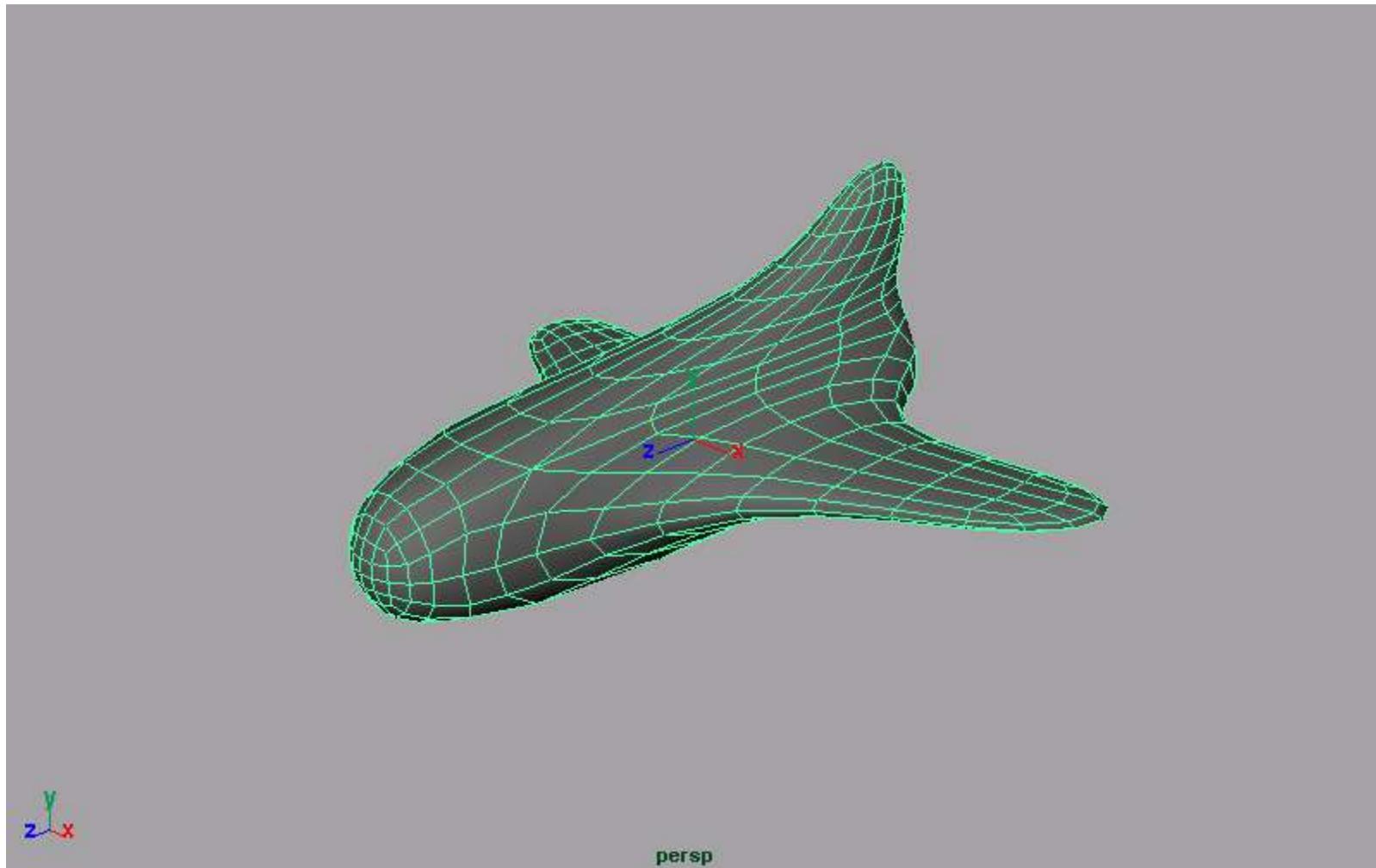
# Durva poligon modell



# Subdivision 1

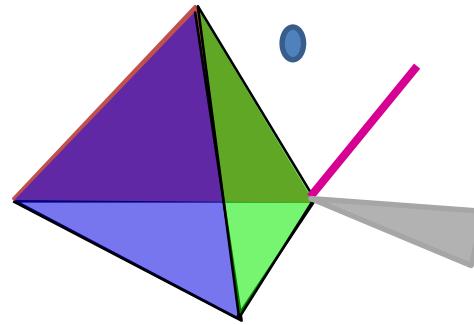
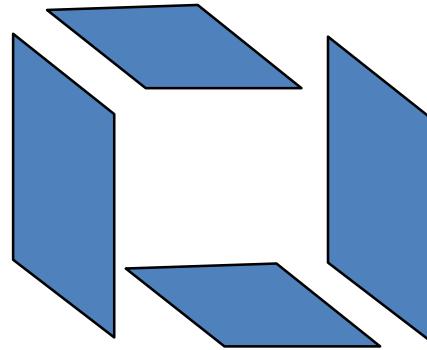


# Subdivision 2



# B-rep: határfelületek megadása

- Test = érvényes (létrehozható): ne legyenek alacsonyabb dimenziós elfajuló részek: minden határpont környezetében kell belső pontnak is lennie.



- Topológiai érvényesség:
  - Élek (2,3,...) csúcspontban találkoznak
  - Egy él két lapot választ el, és nem metsz élt
  - Egy lapot él és csúcs sorozat határol
  - A felület nem metszi saját magát
  - **Euler-Poincaré téTEL:**  $\boxed{\text{csúcs} - \text{él} + \text{lap} = 2(\text{db-lyuk})}$

# (Leonhard) Euler operátorok

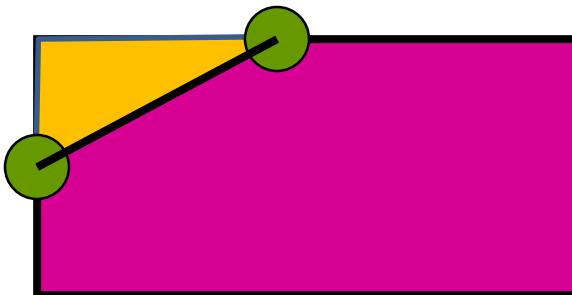
- Lap kihúzás



Csúcsok Lapok Élek

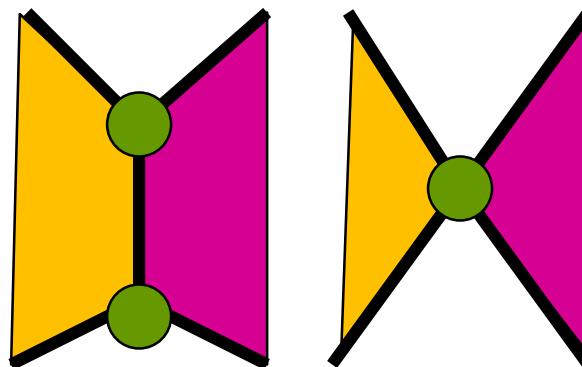
$$+4 \quad +4 = +8$$

- Lap felvágás



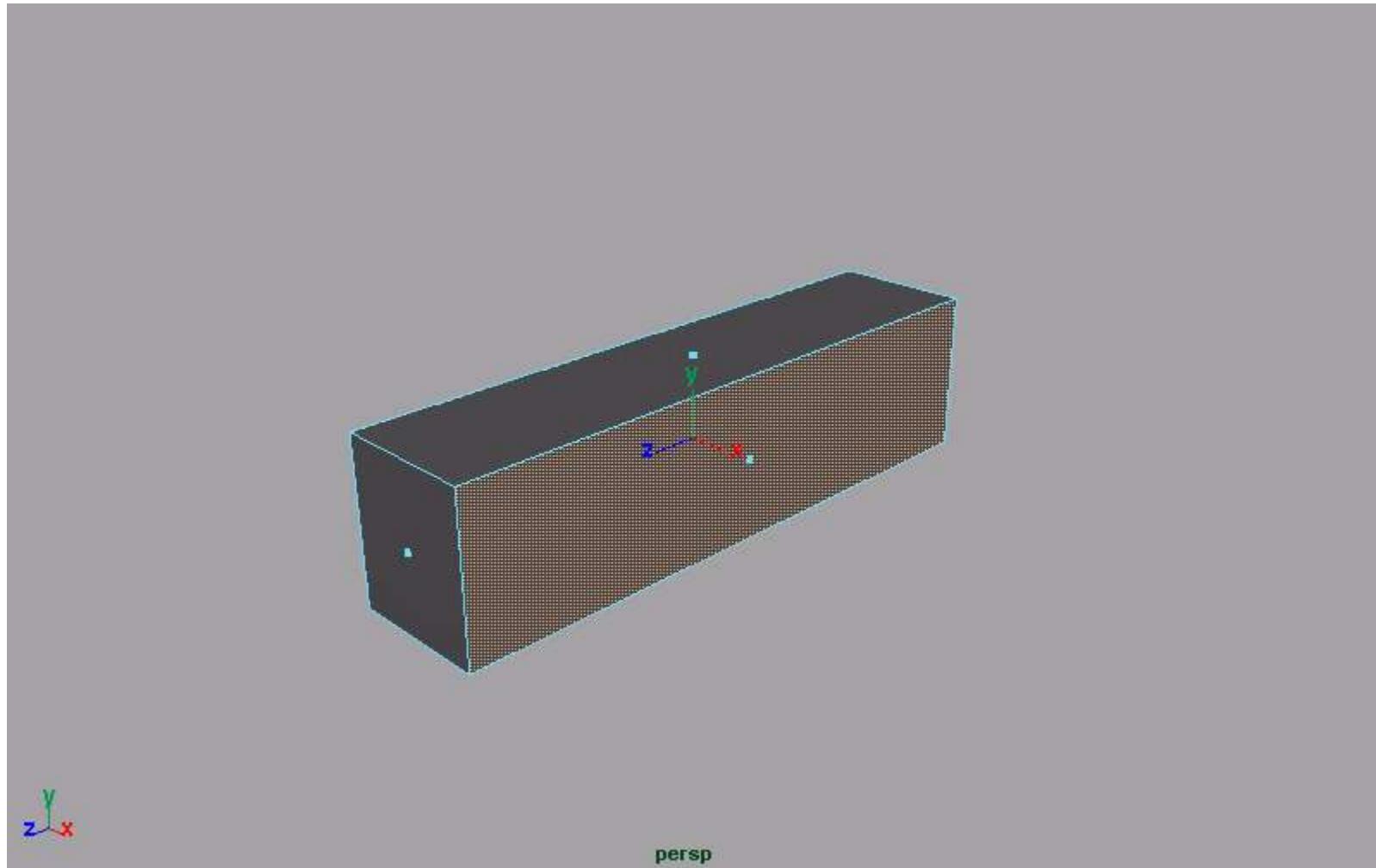
$$+2 \quad +1 = +3$$

- Él törlés
- Csúcs szétvágás

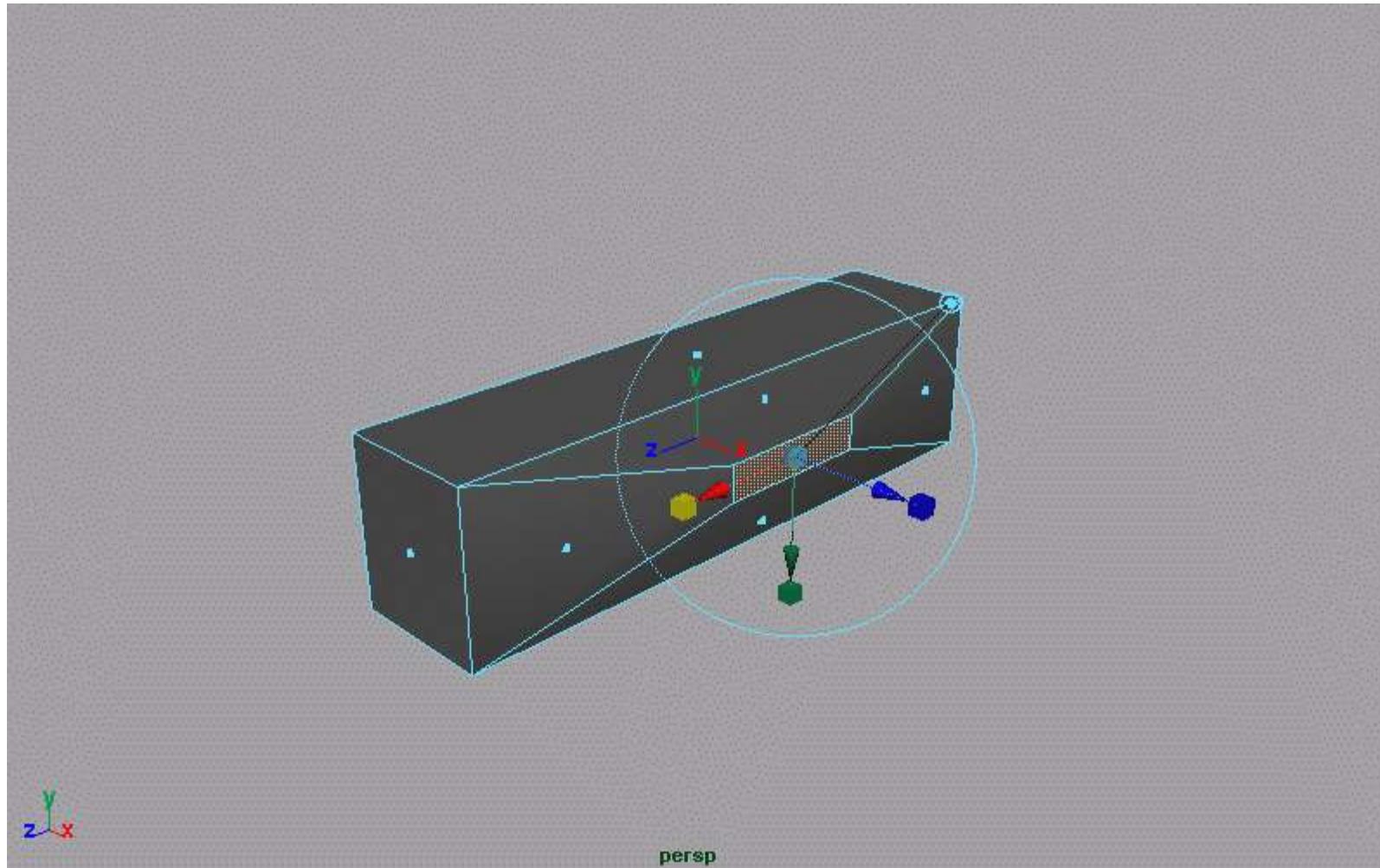


$$\begin{matrix} -1 \\ +1 \end{matrix} \quad \begin{matrix} 0 \\ 0 \end{matrix} = \begin{matrix} -1 \\ +1 \end{matrix}$$

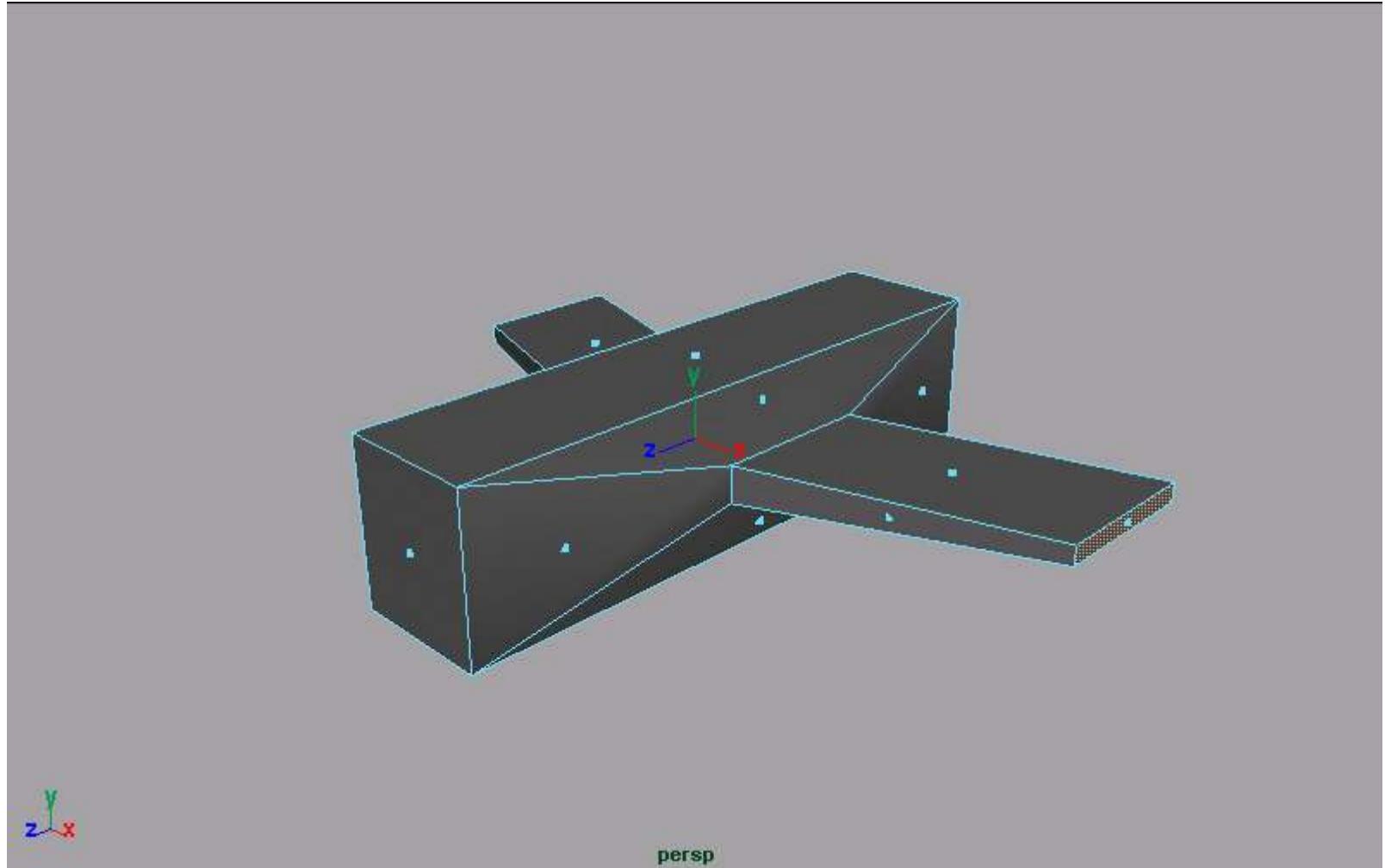
# Kezdet: érvényes téglatest



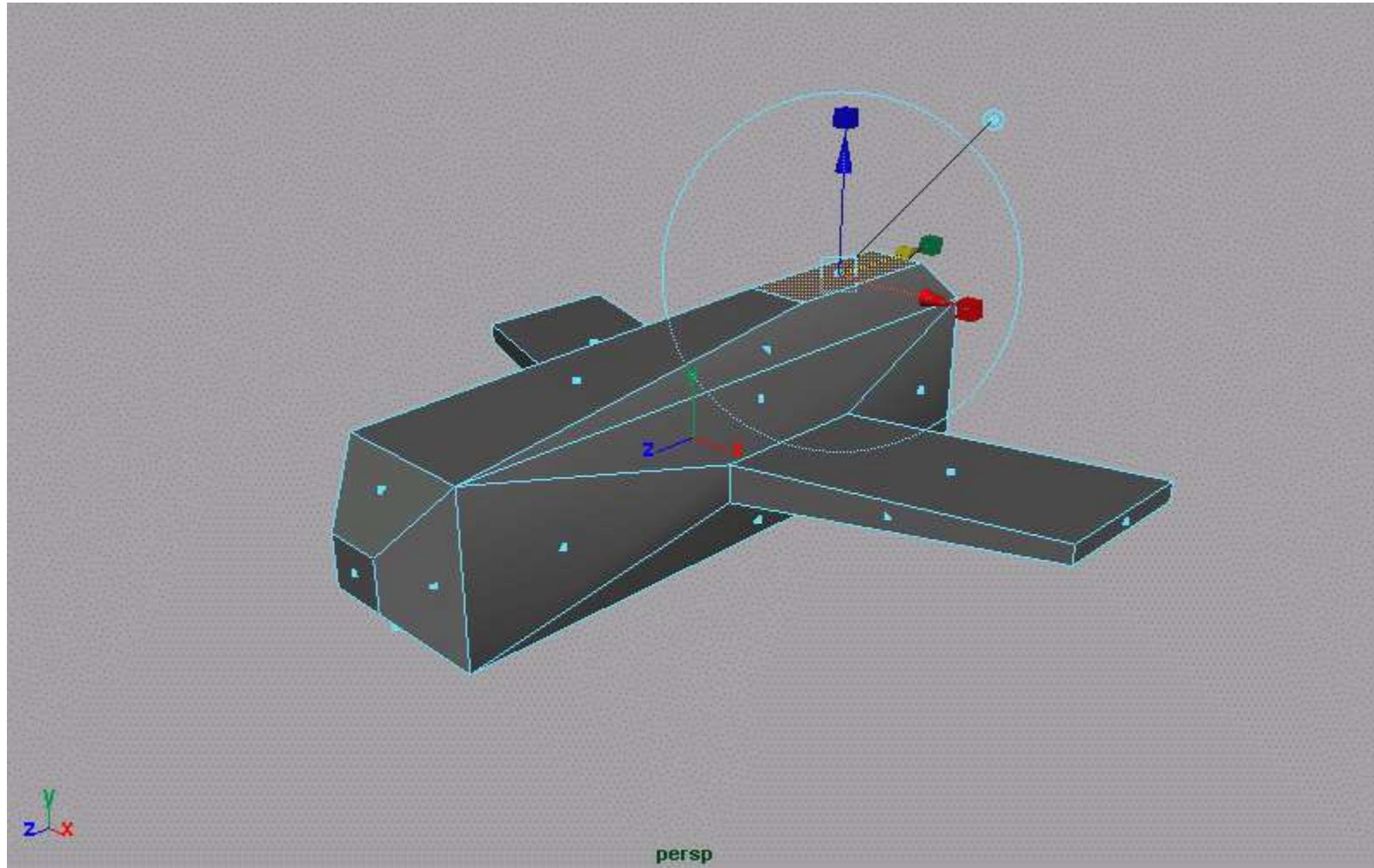
# Lap kihúzás



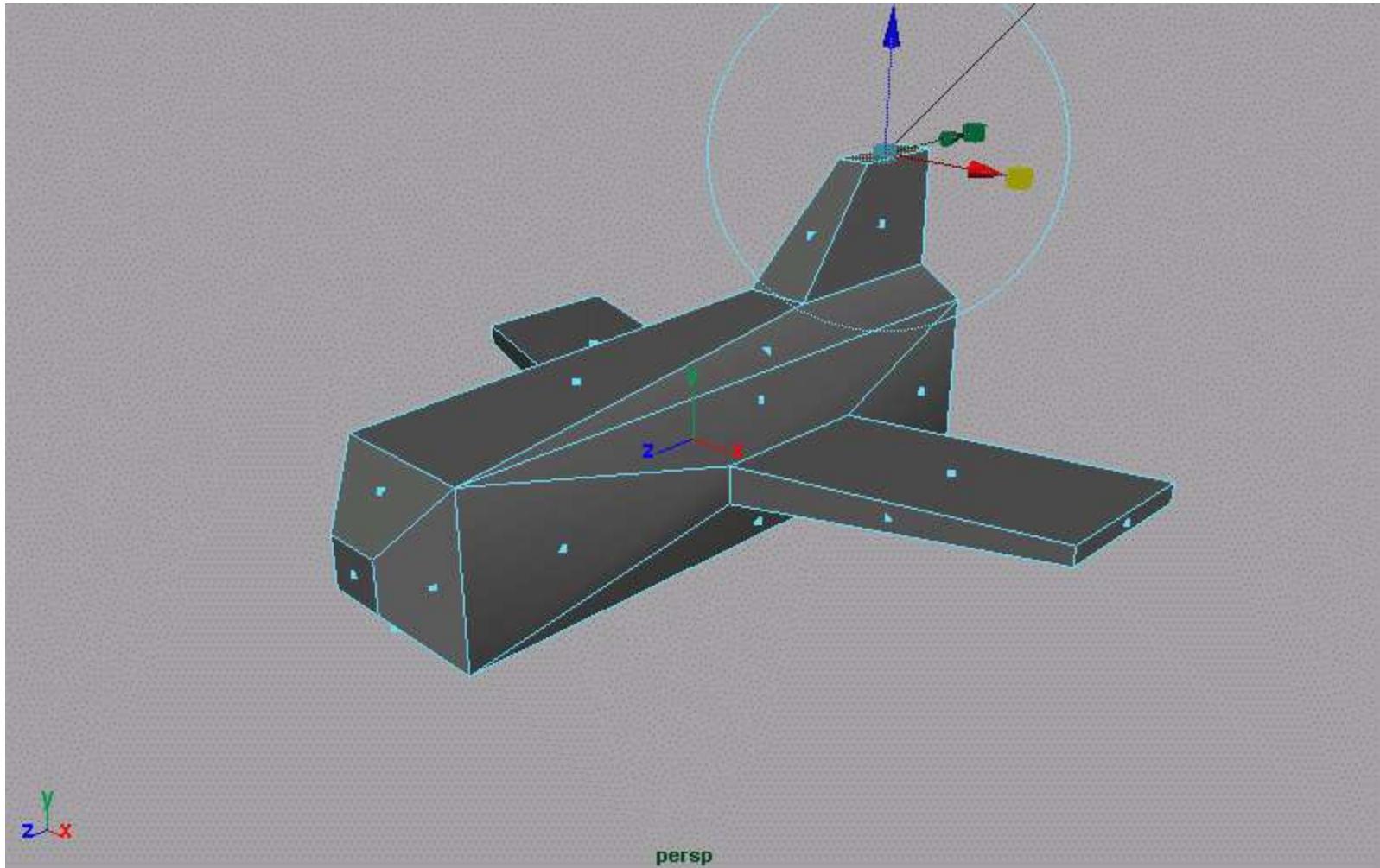
# Lap kihúzás



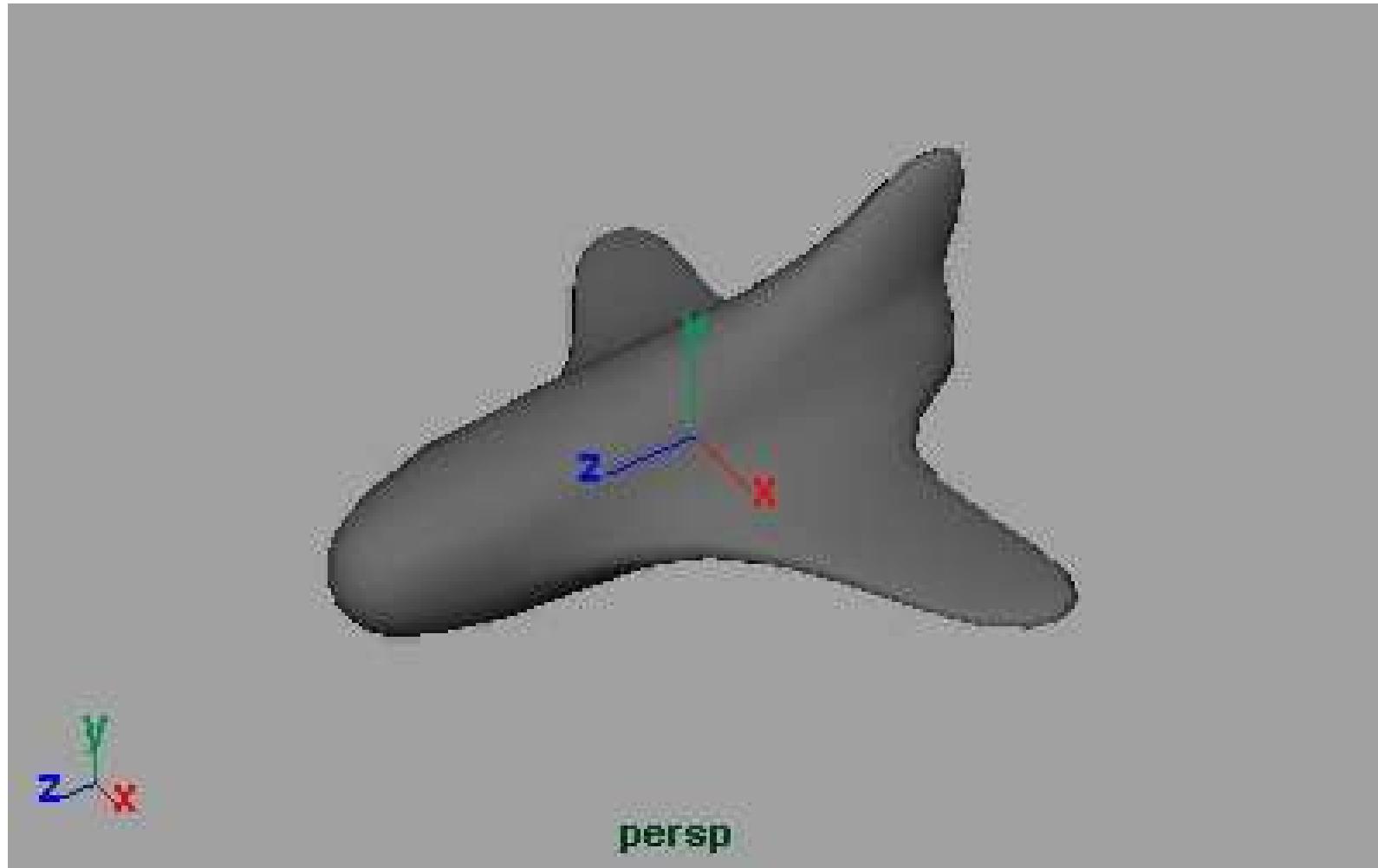
# Lap kihúzás



# Lap kihúzás



# Subdivision simítás



# Felület modellezés

- Explicit = magasságmező ritkán használható
- Implicit: geometriai definíció vektoralgebrai fordítása
  - Implicit felület normálvektor = gradiens
- Paraméteres: paraméteres görbékre vagy a súlypont analógiára vezetjük vissza
  - Paraméteres felület normálvektora = parciális deriváltak vektoriális szorzata
- Felosztott: háromszögháló

# Ellenőrző kérdések

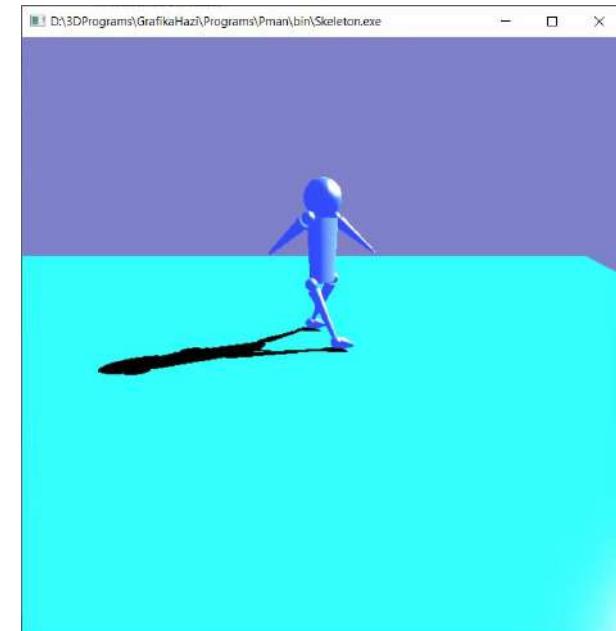
- Bizonyítsa be, hogy a Lagrange bázisfüggvények összege 1 (pl. teljes indukció)
- Bizonyítsa be, hogy a görbék koordinátarendszer függetlensége megköveteli, hogy a bázisfüggvények összege 1 legyen.
- Írja fel az egyenletes Catmull-Rom spline bázisfüggvényeit!
- Írja fel a tórusz paraméteres és implicit egyenleteit.
- Implementáljon PowerPoint-szerű szabadformájú görbét!
- Soroljon fel minél több Euler operátort!
- *Tervezzen adatstruktúrát egy poliéderhez! Hogyan implementálható azon a Catmull-Clark Subdivision?*
- Adja meg az egyenes másodfokú egyenletét!
- Általánosítsa az Euler tételet több darabból álló és lyukas objektumokra.
- Írjon óraprogramot: számjegyek animált Catmull-Rom spline-ok.
- Írjon programot, amely pontokkal adott függvényt interpolál, integrál és differenciál.
- Animálja végig a sebesség és gyorsulás vektorokat a Bézier, Lagrange és Catmull-Rom görbéken.

”Τὰ πάντα ῥεῖ καὶ οὐδὲν μένει.”  
Ἡράκλειτος

# Transzformációk

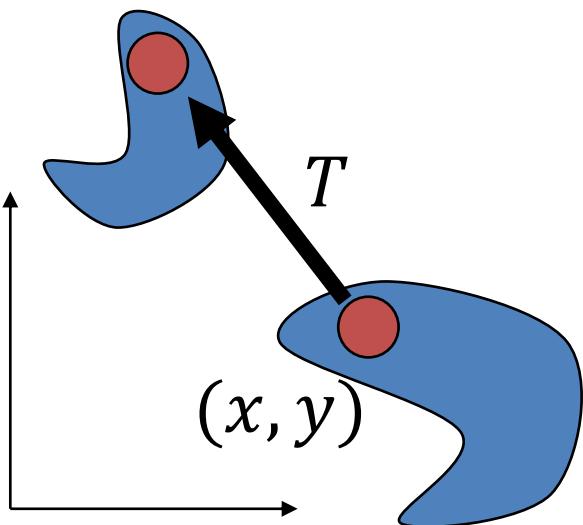
## 1. Affin transzformációk

Szirmay-Kalos László



# Transzformációk

$$(x', y') = T(x, y)$$



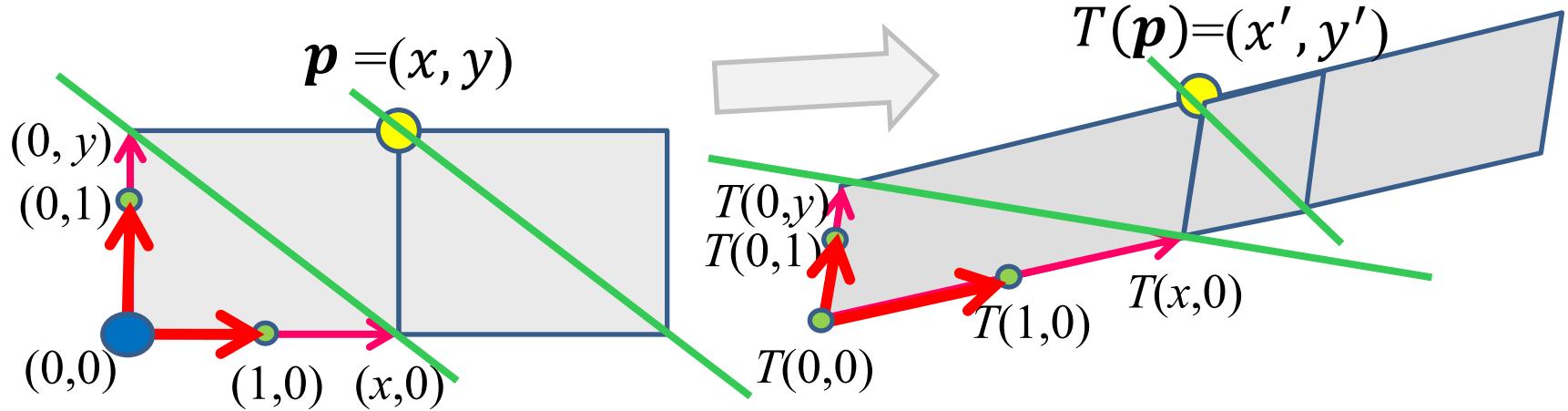
- Ponthoz pontot rendel
- Tönkre tehetik reprezentációt és az egyenletet
- Korlátozzuk a transzformációkat, hogy az **egyenes (szakasz)** és **sík (háromszög)** megmaradjon

- **Affin transzformációk**

- Párhuzamos egyenes tartó
- Eltolás, elforgatás, nyújtás, nyírás, tükrözés, ...

# Affin transzformációk

Egyenest-egyenesbe, párhuzamosokat megtartja

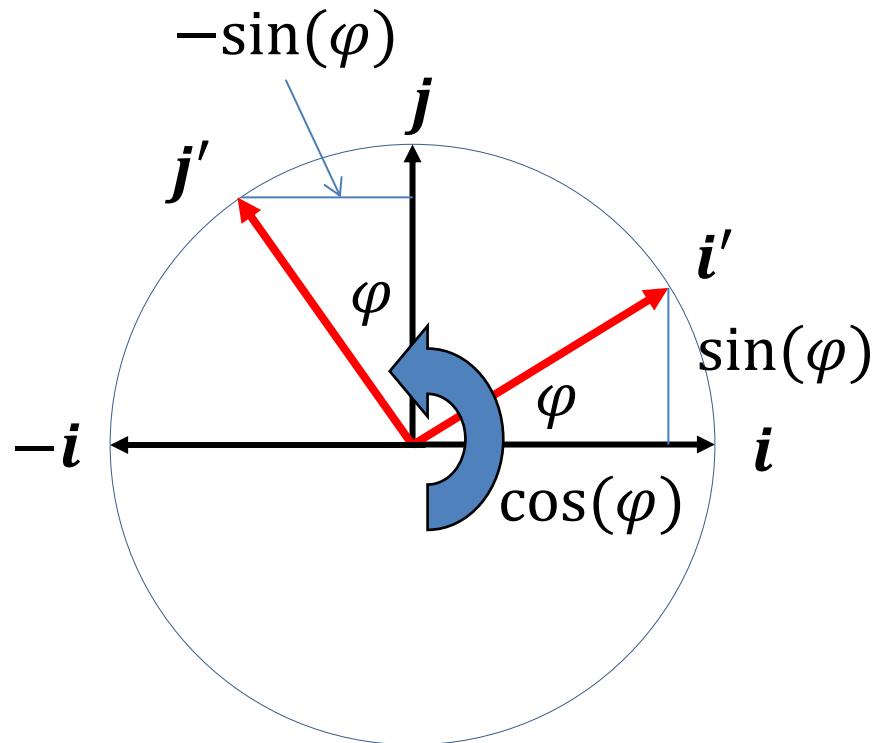


$$\begin{aligned}T(\mathbf{p}) &= T(0,0) + (T(x,0) - T(0,0)) + (T(0,y) - T(0,0)) \\&= T(0,0) + x(T(1,0) - T(0,0)) + y(T(0,1) - T(0,0)) \\&= \mathbf{o}' + x\mathbf{i}' + y\mathbf{j}'\end{aligned}$$

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} \mathbf{i}'_x & \mathbf{i}'_y & 0 \\ \mathbf{j}'_x & \mathbf{j}'_y & 0 \\ \mathbf{o}'_x & \mathbf{o}'_y & 1 \end{bmatrix}$$

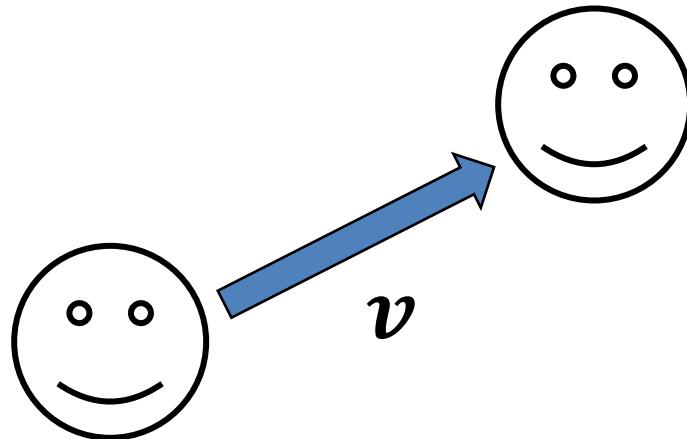
$$\begin{aligned}x' &= ax + by + c \\y' &= dx + ey + f\end{aligned}$$

# 2D forgatás



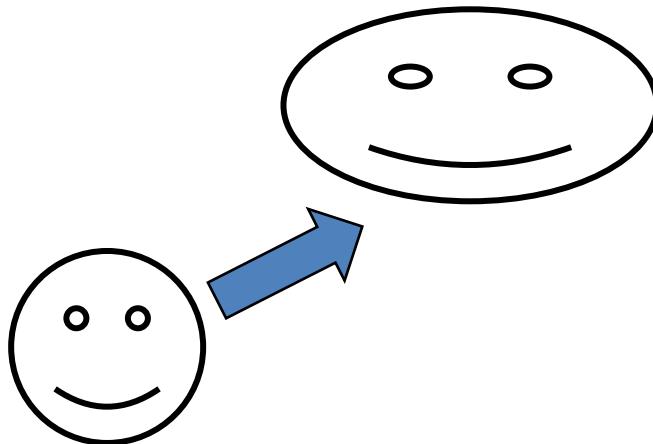
$$[x', y', 1] = [x, y, 1] \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# 3D eltolás



$$[x', y', z', 1] = [x, y, z, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ v_x & v_y & v_z & 1 \end{bmatrix}$$

# 3D skálázás

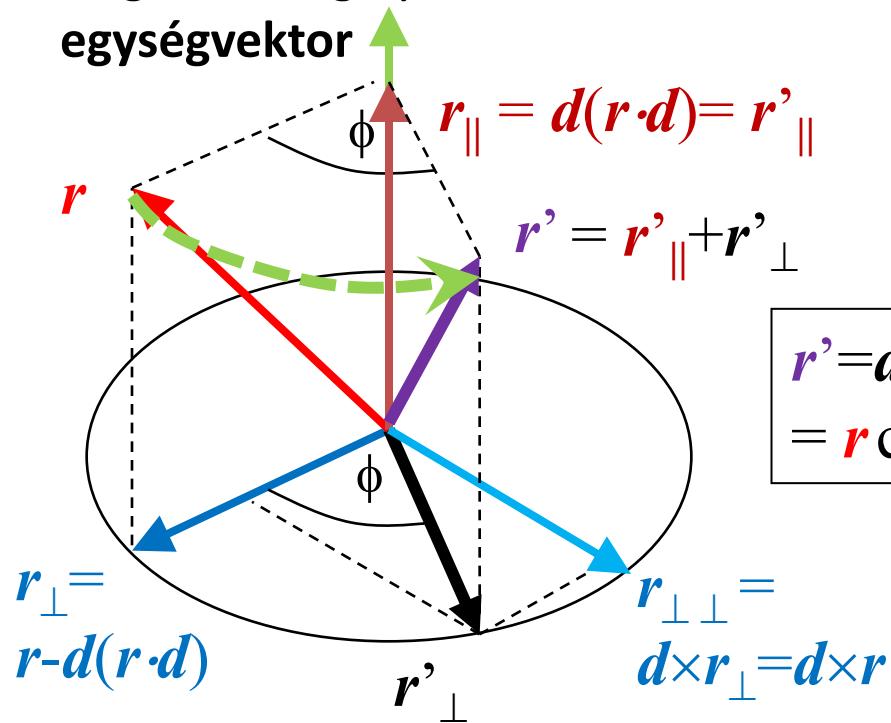


$$[x', y', z', 1] = [x, y, z, 1] \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Origón átmenő $d$ tengely körüli forgatás: (Olinde) Rodrigues formula



$d$ : forgatási tengely,  
egységvektor



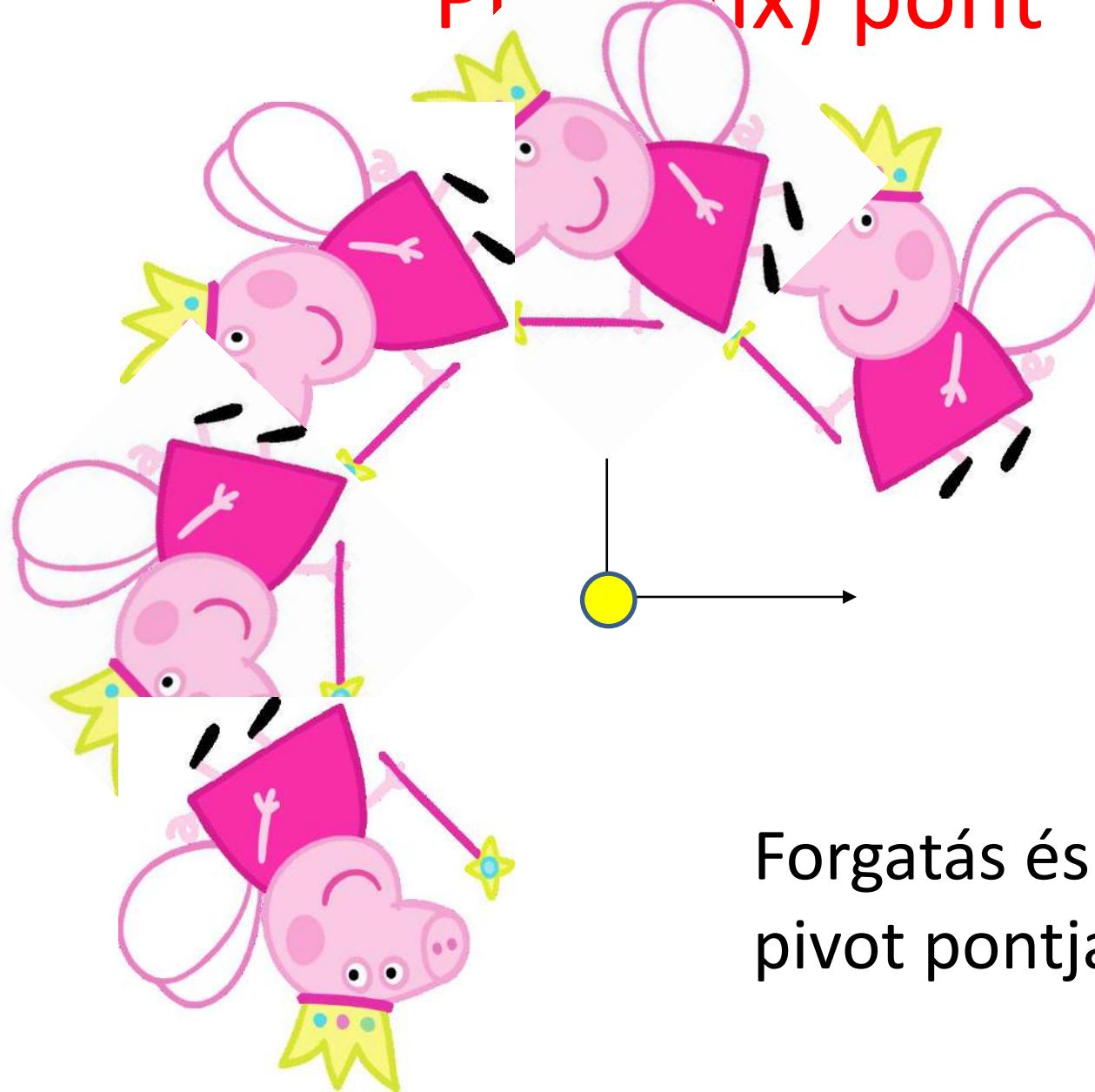
$$\begin{aligned} r' &= d(r \cdot d) + (r - d(r \cdot d))\cos(\phi) + d \times r \sin(\phi) \\ &= r \cos(\phi) + d(r \cdot d)(1 - \cos(\phi)) + d \times r \sin(\phi) \end{aligned}$$

Mátrix sorai: hova kerül az origó és az  $i, j, k$ ?

$$(0,0,0) \rightarrow (0,0,0)$$

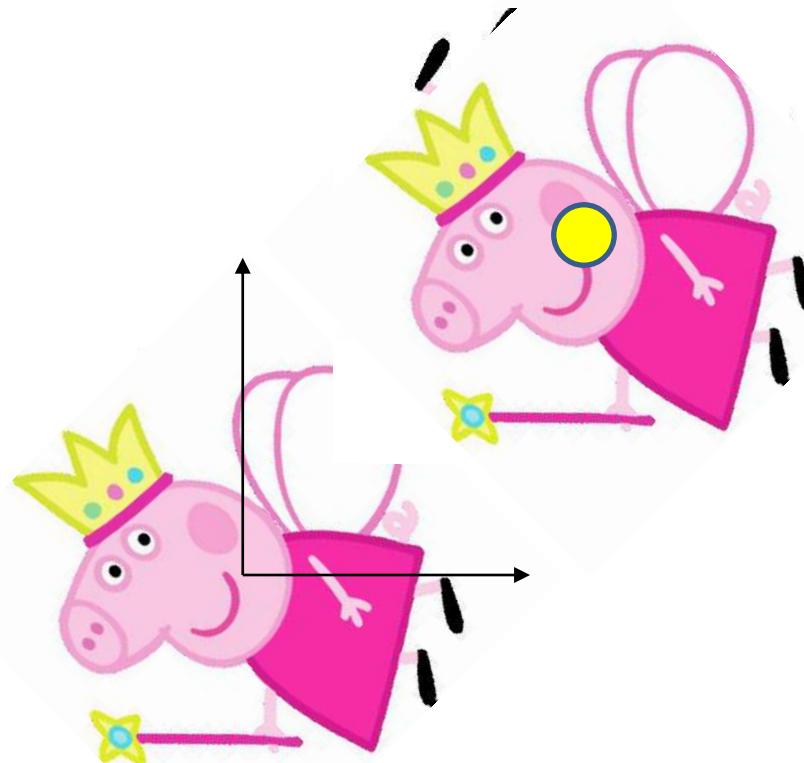
$$\begin{aligned} (1,0,0) \rightarrow (1,0,0)C + (\mathbf{d}_x, \mathbf{d}_y, \mathbf{d}_z)\mathbf{d}_x(1-C) + (\mathbf{d}_x, \mathbf{d}_y, \mathbf{d}_z) \times (1,0,0)S = \\ (C + \mathbf{d}_x^2(1-C), \mathbf{d}_y\mathbf{d}_x(1-C) + \mathbf{d}_zS, \mathbf{d}_z\mathbf{d}_x(1-C) - \mathbf{d}_yS) \end{aligned}$$

Pivot (fix) pont



Forgatás és skálázás  
pivot pontja az origó

# Pivot (fix) pont



1. Pivot az origóba
2. Forgatás vagy skálázás az origó körül
3. Origó vissza

# Egybevágósági transzformációk

- Affin:

$$[x', y', z', 1] = [x, y, z, 1] \begin{bmatrix} i'_x & i'_x & i'_z & 0 \\ j'_x & j'_y & j'_z & 0 \\ k'_x & k'_y & k'_z & 0 \\ o'_x & o'_y & o'_z & 1 \end{bmatrix}$$

- Bázisvektorok egységnnyi hossza és egymásra merőlegessége megmarad:

$$\begin{aligned} i' \cdot i' &= 1, & j' \cdot j' &= 1, & k' \cdot k' &= 1 \\ i' \cdot j' &= 0, & j' \cdot k' &= 0, & k' \cdot i' &= 0 \end{aligned}$$

- Determináns: +1 vagy -1
- Példák: Eltolás, forgatás, tükrözés
- Ellenpélda: skálázás, nyírás

# Affin transzformációk

- Párhuzamos egyeneseket párhuzamos egyenesekbe képeznek le.
- Szokásos transzformációk (eltolás, forgatás, skálázás, nyírás, tükrözés) ilyenek.
- Mátrixszorzás ambiens koordinátákra, ahol a negyedik oszlop  $[0,0,0,1]^T$
- Eltolásnak nincs pivot (fix) pontja, a többinek az origó. Ha mást szeretnénk, akkor eltolás, transzformáció, visszatolás.

”μὴ εῖναι βασιλικὴν ἀτραπὸν  
ἐπὶ γεωμετρίαν”

Εύκλείδης

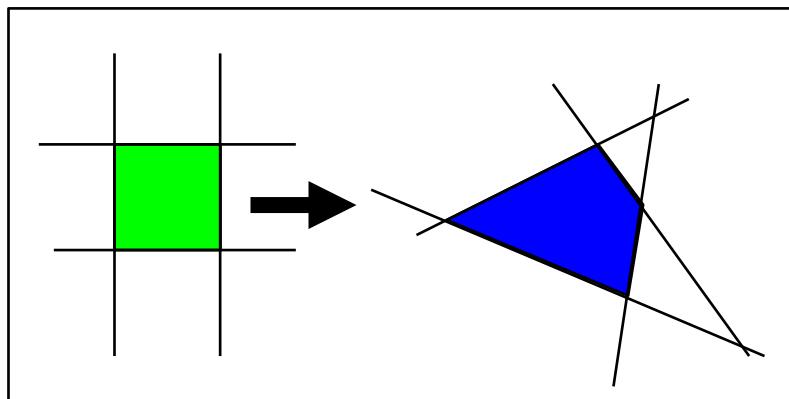
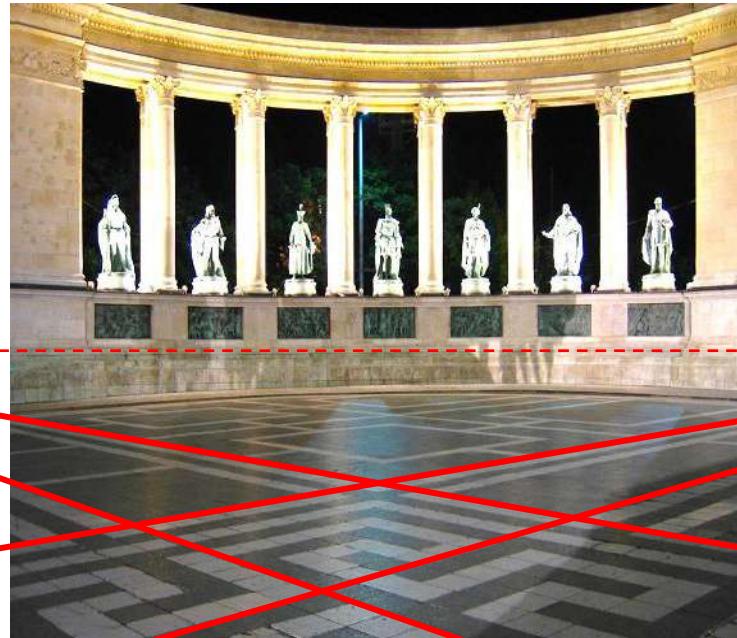
# Transzformációk

## 2. Projektív geometria

Szirmay-Kalos László

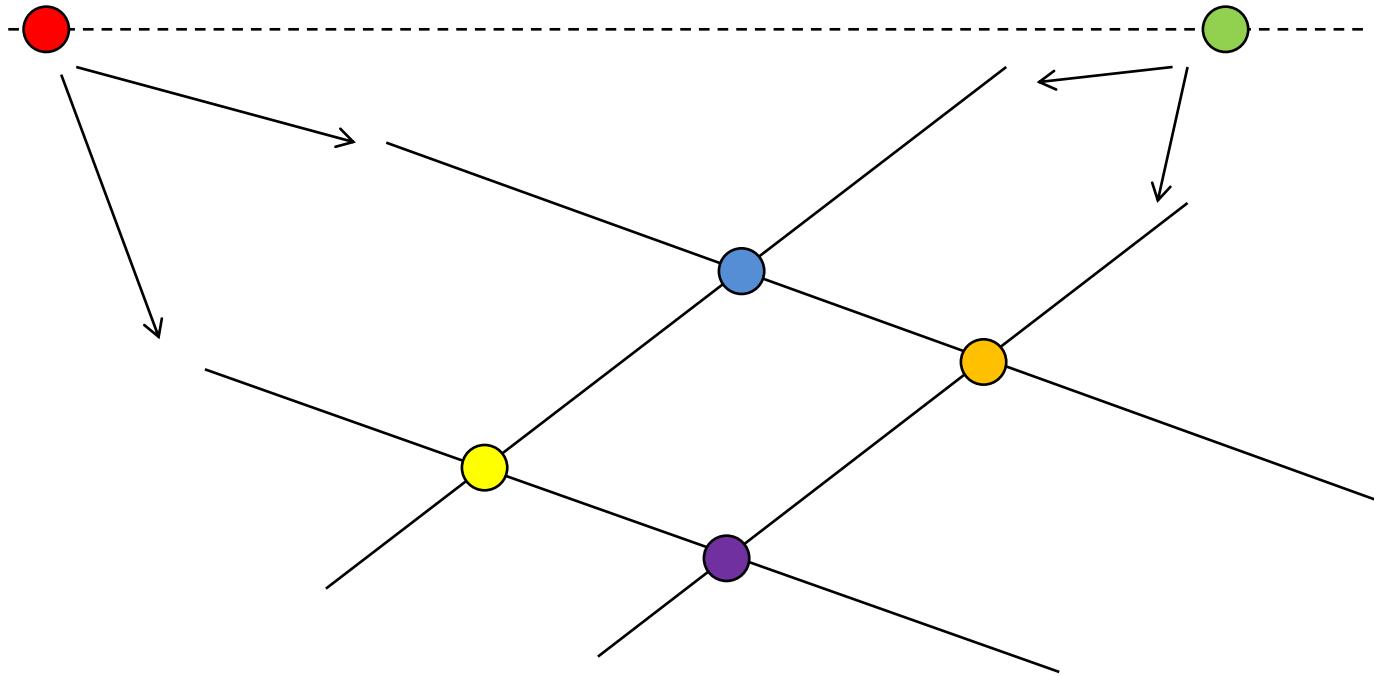


# Perspektíva



- Egyenest egyenesbe képezi le
- Nem párhuzamostartó, azaz nem affin
- Euklideszi geometria lyukas

# Euklideszi → Projektív sík



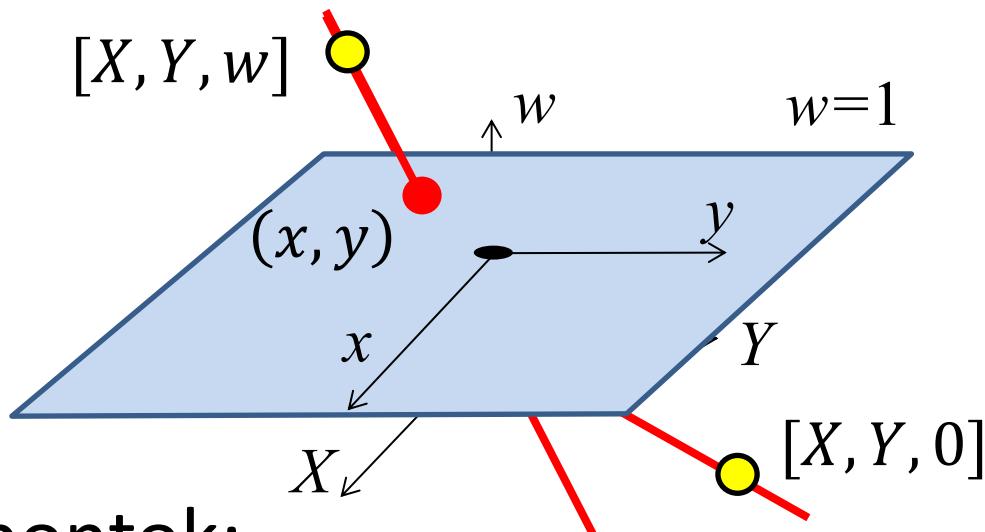
## Euklideszi

- 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.

## Projektív

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

# Projektív sík analitikus geometriája



Euklideszi pontok:

$$(x, y) \rightarrow [x, y, 1] \sim [x \cdot w, y \cdot w, w] = [X, Y, w]$$

Homogén osztás:  $x = \frac{X}{w}, y = \frac{Y}{w}$

$$w \neq 0$$

Ideális pontok:

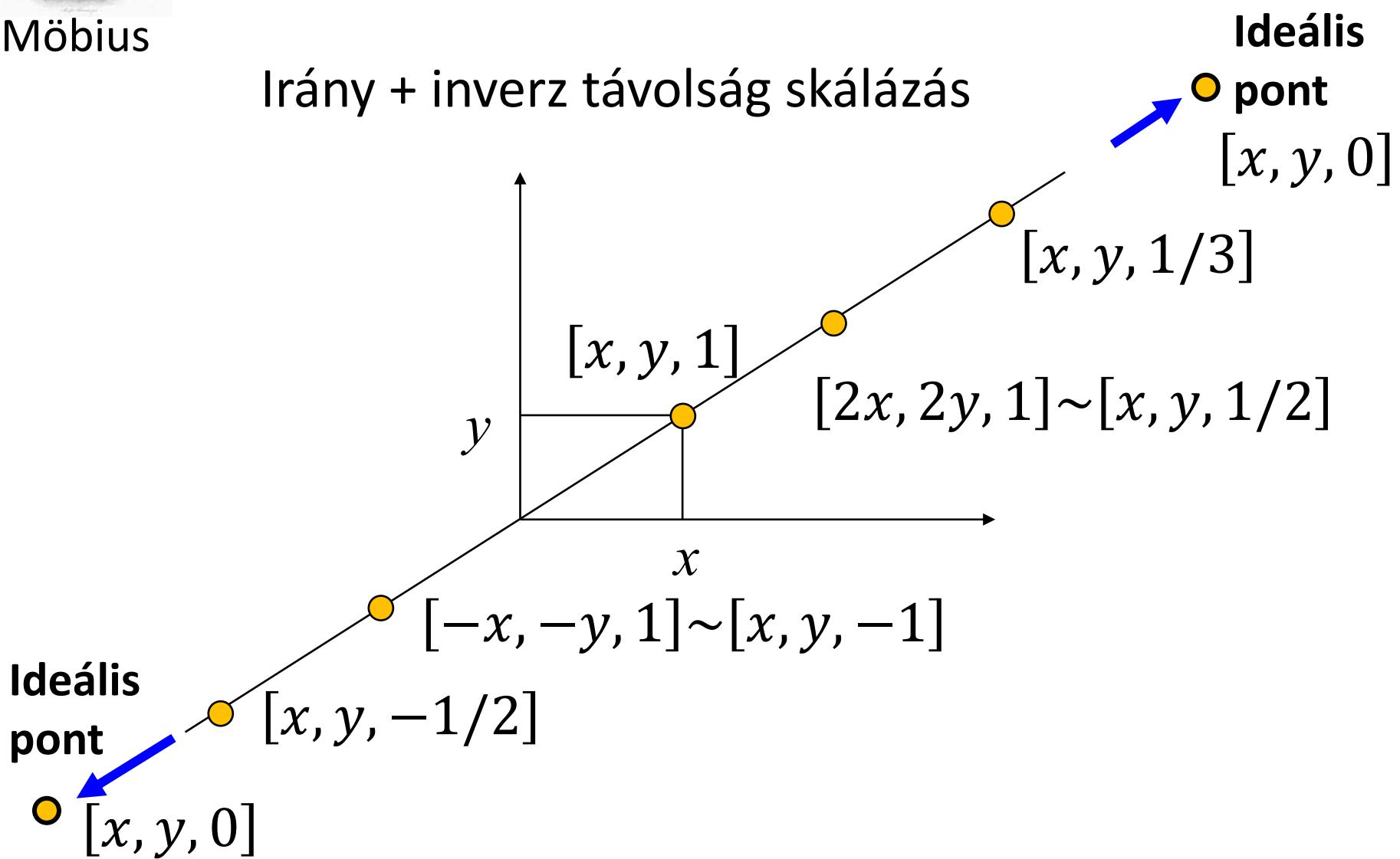
$$[X, Y, 0]$$



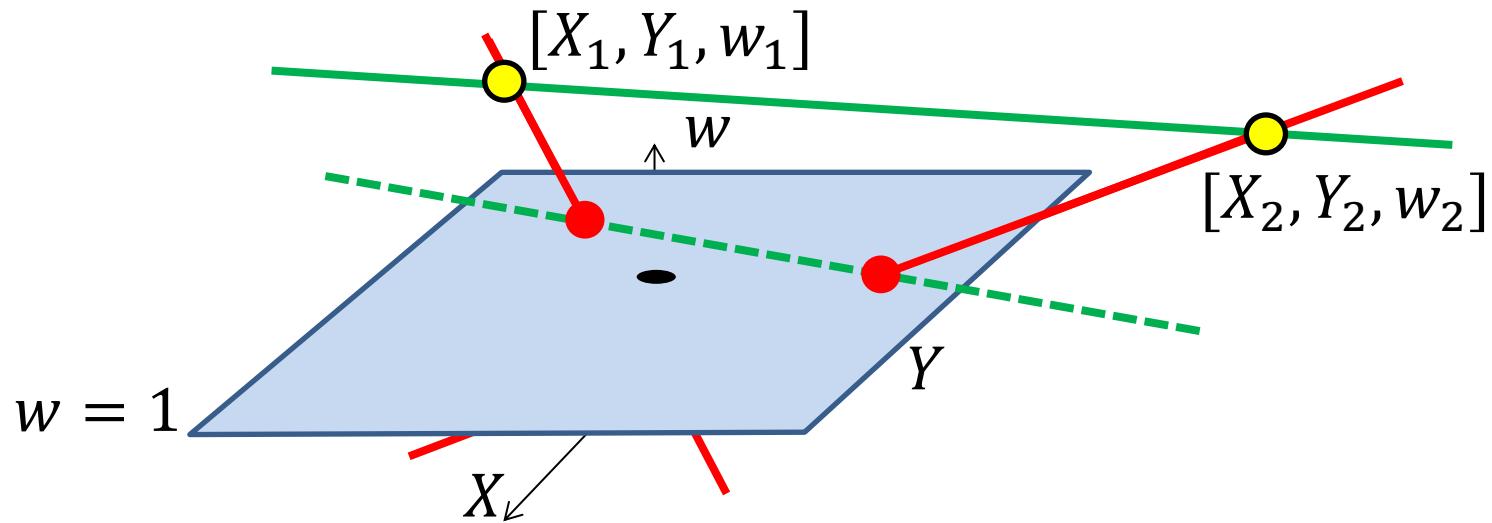
Möbius

# Homogén koordináták

Irány + inverz távolság skálázás



# Projektív egyenes parametrikus egyenlete



$$[X(t), Y(t), w(t)] = [X_1, Y_1, w_1](1 - t) + [X_2, Y_2, w_2]t$$

# Egyenes implicit egyenlete

Euklideszi egyenes, Descartes koordináták:

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

Euklideszi egyenes, homogén koordináták:

$$n_x X/w + n_y Y/w + d = 0 \quad w \neq 0$$

Projektív egyenes:

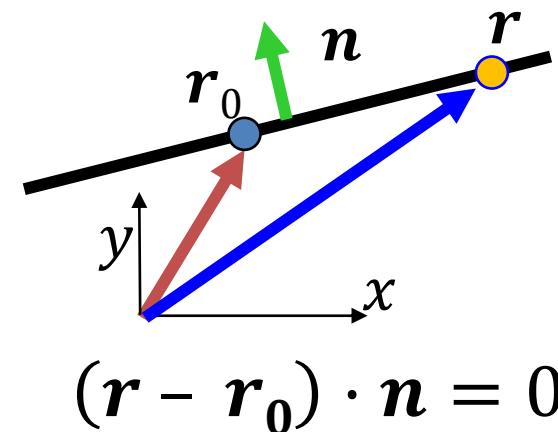
$$n_x X + n_y Y + dw = 0$$

$$w \neq 0$$

$$[X, Y, w] \begin{bmatrix} n_x \\ n_y \\ d \end{bmatrix} = 0$$

Pont: sorvektor

Egyenes:  
oszlopvektor



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

# Dualitás

- Ha egy téTEL pontok és egyenesek viszonyáról szól, akkor a pont és egyenes felcserélhetők benne.
- 2D pont: 3 elemű sorvektor, homogén
  - 2D projektív sík pont  $\sim$  3D euklideszi tér origót metsző egyenes
- 2D egyenes: 3 elemű oszlopvektor, homogén
  - 2D projektív sík egyenes  $\sim$  3D euklideszi tér origót metsző egyenes
- **2D egyenes egyenlete:**

$p \cdot l = 0$

  - 2D pont 3D vektora merőleges a 2D egyenes 3D vektorára

# Metszés és illeszkedés

- $p_1$  és  $p_2$  pontra illeszkedő  $\mathbf{l}$  egyenes:

$$\mathbf{p}_1 \cdot \mathbf{l} = 0, \quad \mathbf{p}_2 \cdot \mathbf{l} = 0 \rightarrow \mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$$

$$\mathbf{l} \perp \mathbf{p}_1 \quad \mathbf{l} \perp \mathbf{p}_2$$

- $l_1$  és  $l_2$  egyenes  $\mathbf{p}$  metszéspontja:

$$\mathbf{p} \cdot \mathbf{l}_1 = 0, \quad \mathbf{p} \cdot \mathbf{l}_2 = 0 \rightarrow \mathbf{p} = \mathbf{l}_1 \times \mathbf{l}_2$$

$$\mathbf{p} \perp \mathbf{l}_1 \quad \mathbf{p} \perp \mathbf{l}_2$$

- $p$  ponton átmenő  $L$  egyenesre merőleges  $\mathbf{l}$  egyenes ( $L^* = L$ -ből a w törlése)

$$\mathbf{p} \cdot \mathbf{l} = 0, \quad \mathbf{l}_X L_X + \mathbf{l}_Y L_Y = \mathbf{l} \cdot L^* = 0 \rightarrow \mathbf{l} = \mathbf{p} \times L^*$$

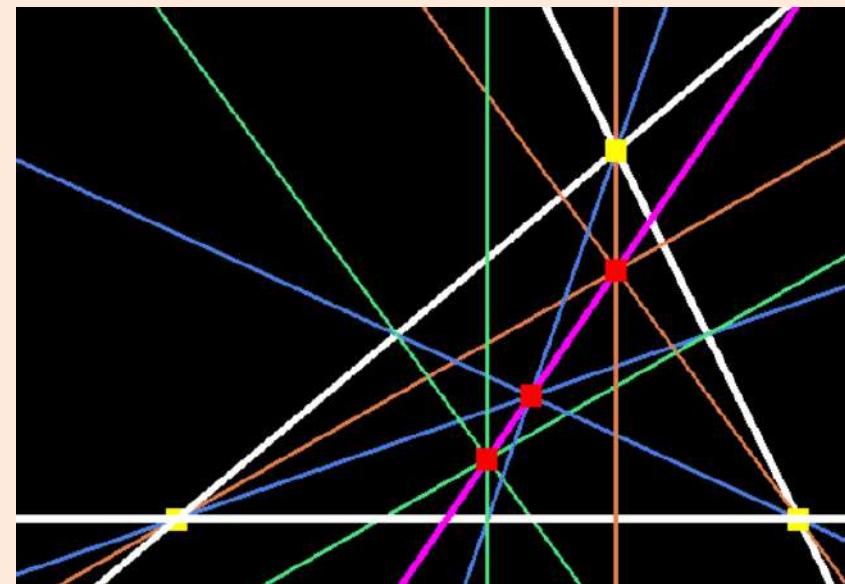


# Euler egyenes

```
vec3 v[0]=vec3(x0,y0,1), v[1]=vec3(x1,y1,1), v[2]=vec3(x2,y2,1);
// midpoints
m[2] = v[0] + v[1];
m[0] = v[1] + v[2];
m[1] = v[2] + v[0];
// edges
e[0] = cross(v[1], v[2]);
e[1] = cross(v[2], v[0]);
e[2] = cross(v[0], v[1]);

for (int i = 0; i < 3; i++) {
    median[i] = cross(v[i], m[i]);
    altitude[i] = cross(v[i], vec3(e[i].x, e[i].y, 0));
    bisector[i] = cross(m[i], vec3(e[i].x, e[i].y, 0));
}
centroid = cross(median[0], median[1]);
orthocenter = cross(altitude[0], altitude[1]);
circumcenter = cross(bisector[0], bisector[1]);

euler = cross(centroid, circumcenter); // Euler's line
```



# Projektív tér homogén koordinátákkal

- Euklideszi pontok:

$$(x, y, z) \rightarrow [x, y, z, 1] \sim [x \cdot w, y \cdot w, z \cdot w, w] = [X, Y, Z, w]$$

$$\text{Homogén osztás: } x = \frac{X}{w}, \quad y = \frac{Y}{w}, \quad z = \frac{Z}{w}$$

- Ideális pontok:  $[X, Y, Z, 0]$

- Egyenes paraméteres egyenlete:

$$[X(t), Y(t), Z(t), w(t)] = [X_1, Y_1, Z_1, w_1](1 - t) + [X_2, Y_2, Z_2, w_2]t$$

- Sík implicit egyenlete:

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

# Homogén lineáris transzformációk

## Homogén koordinátavektor szorzása mátrixszal

- Affin transzformációkat is tartalmazza

- 2D transzformáció  $3 \times 3$  mátrix

$$[X', Y', w'] = [X, Y, w] \cdot T_{3 \times 3}$$

- 3D transzformáció  $4 \times 4$  mátrix

$$[X', Y', Z', w'] = [X, Y, Z, w] \cdot T_{4 \times 4}$$

- Transzformációk konkatenációja: Asszociatív

$$\begin{aligned}[X', Y', Z', w'] &= (\dots ([X, Y, Z, w] \cdot T_1) \cdot T_2) \dots \cdot T_n \\ &= [X, Y, Z, w] \cdot (T_1 \cdot T_2 \cdot \dots \cdot T_n) \\ &= [X, Y, Z, w] \cdot T\end{aligned}$$

# Homogén lineáris transzformációk tulajdonságai

- Egyenest egyenesbe, kombinációkat kombinációkba, konvex kombinációkat konvex kombinációkba
- Ha nem invertálható, akkor elfajulás lehetséges

## Példa: egyenest egyenesbe

$$[X(t), Y(t), Z(t), w(t)] = [X_1, Y_1, Z_1, w_1]t + [X_2, Y_2, Z_2, w_2](1 - t)$$

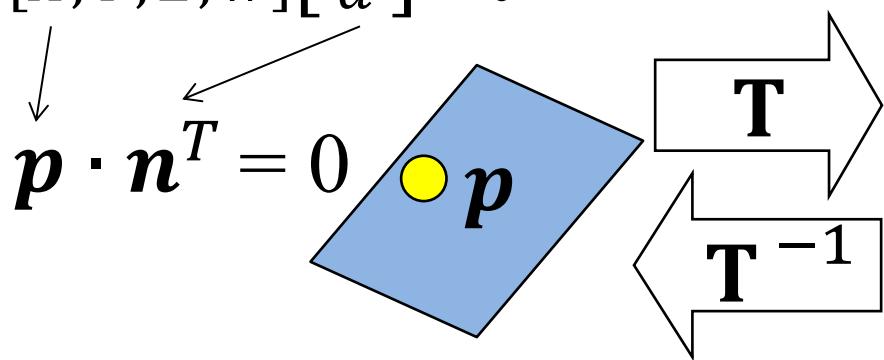
$$\mathbf{p}(t) = \mathbf{p}_1 t + \mathbf{p}_2 (1 - t) \quad // \cdot \mathbf{T}$$

$$\mathbf{p}^*(t) = (\mathbf{p}_1 \cdot \mathbf{T})t + (\mathbf{p}_2 \cdot \mathbf{T})(1 - t)$$

$$\mathbf{p}^*(t) = \mathbf{p}_1^* t + \mathbf{p}_2^* (1 - t)$$

# Invertálható homogén lineáris transzformációk: síkot síkba

$$[X, Y, Z, w] \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} = 0$$



Sík transzformáltja:

$$\mathbf{n}^{*T} = \mathbf{T}^{-1} \cdot \mathbf{n}^T$$

$$p^* = p \cdot \mathbf{T}$$

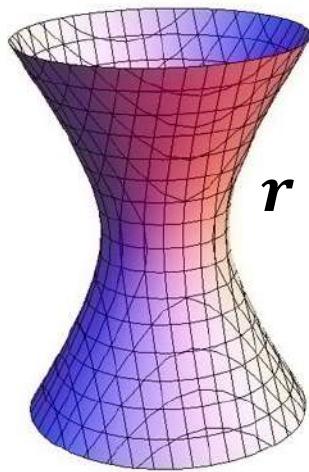
$$p^* \cdot \mathbf{T}^{-1} = p$$

$$(p^* \cdot \mathbf{T}^{-1}) \cdot \mathbf{n}^T = 0$$

$$p^* \cdot (\mathbf{T}^{-1} \cdot \mathbf{n}^T) = 0$$

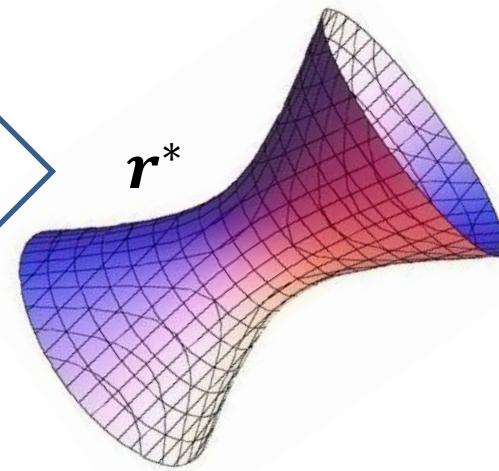
$$p^* \cdot \mathbf{n}^{*T} = 0$$

# Implicit felületek transzformációja



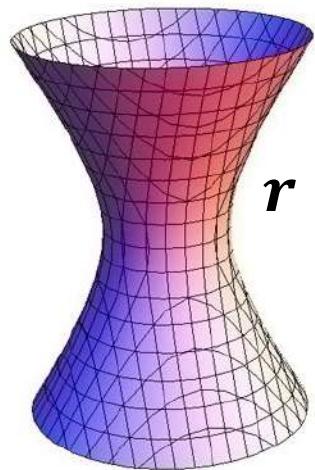
$$f(\mathbf{r}) = 0$$

$$\boxed{\mathbf{r}^* = T(\mathbf{r})}$$

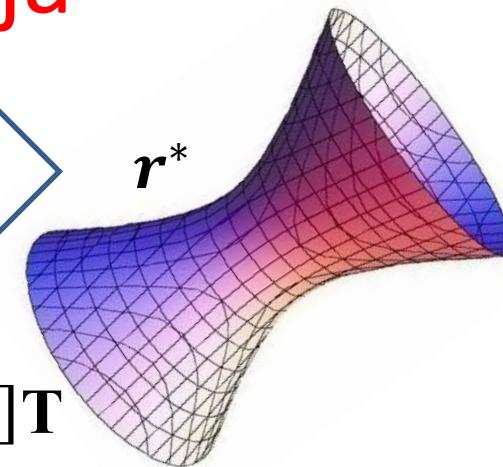


$$f^*(\mathbf{r}^*) = 0 \Leftrightarrow \boxed{f(T^{-1}(\mathbf{r}^*)) = 0}$$

# Kvadratikus felületek homogén lineáris transzformációja



$$\mathbf{r}^* = T(\mathbf{r})$$



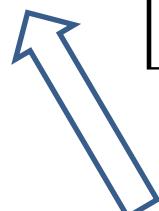
$$[X^*, Y^*, Z^*, w^*] = [X, Y, Z, w] \mathbf{T}$$

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

$$f^*(\mathbf{r}^*) = 0 \Leftrightarrow f(T^{-1}(\mathbf{r}^*)) = 0$$

$$f(\mathbf{r}) = x^2 + y^2 - z^2 - 1 = 0$$

$$[X, Y, Z, w] \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -1 & \\ & & & -1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} = 0$$



$$[X^*, Y^*, Z^*, w^*] \mathbf{T}^{-1}$$

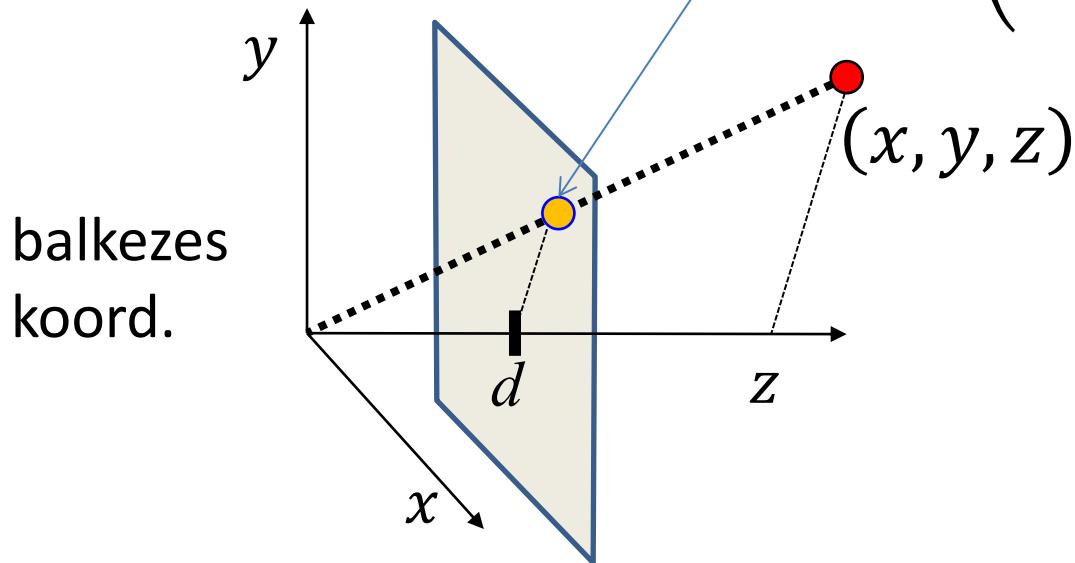
$$[X^*, Y^*, Z^*, w^*] \mathbf{T}^{-1} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -1 & \\ & & & -1 \end{bmatrix} (\mathbf{T}^{-1})^T \begin{bmatrix} X^* \\ Y^* \\ Z^* \\ w^* \end{bmatrix} = 0$$

$$[X^*, Y^*, Z^*, w^*] \mathbf{Q} \begin{bmatrix} X^* \\ Y^* \\ Z^* \\ w^* \end{bmatrix} = 0$$

Homogén lineáris transzformációk kvadratikus felületet kvadratikus felületre képeznek le.

# Középpontos vetítés (projekció)

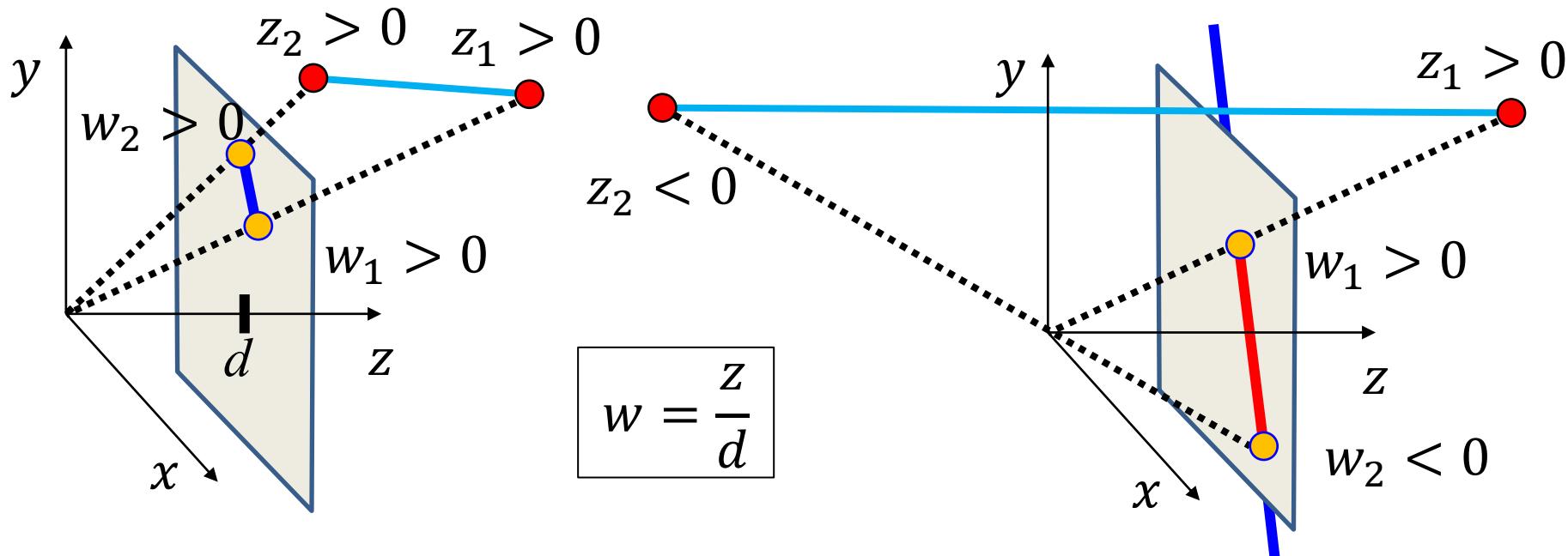
$$(x', y', z') = \left( x \frac{d}{z}, y \frac{d}{z}, d \right)$$



$$[x', y', z', 1] = \left[ x \frac{d}{z}, y \frac{d}{z}, d, 1 \right] \sim \left[ x, y, z, \frac{z}{d} \right]$$

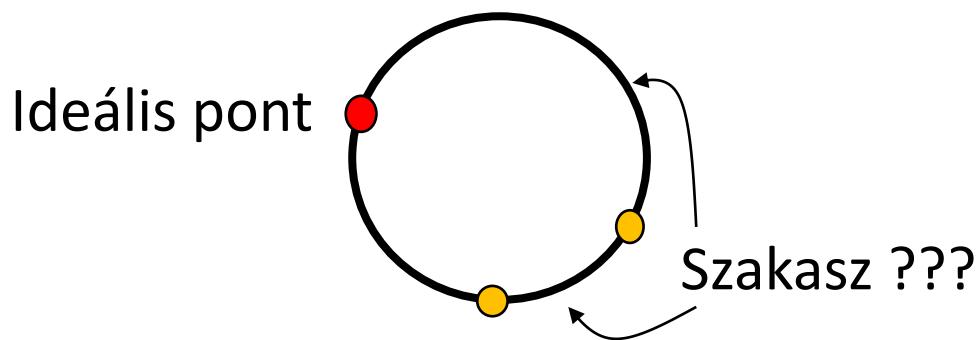
$$[x, y, z, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{bmatrix} = \left[ x, y, z, \frac{z}{d} \right] \sim [x', y', z', 1]$$

# Átfordulási probléma



## Projektív egyenes (topológia)

$$[X(t), Y(t), Z(t), w(t)] = [X_1, Y_1, Z_1, w_1]t + [X_2, Y_2, Z_2, w_2](1 - t)$$



# Projektív geometria

- Euklideszi geometria a középpontos vetítésre lyukas
- Más geometria kell: projektív geometria, amely a végtelent is tartalmazza és nincsenek párhuzamosok
- Projektív geometriához a Descartes koordináták nem jó: Möbius homogén koordinátái
- Projektív síkban az egyenes és a pont egymás duálisai
- Homogén lineáris transzformációk = homogén koordináták egy a transzformációs mátrix szorzata
- Ha homogén koordinátákat használunk, akkor a projektív térben mozgunk (veszély!) átfordulási probléma.

$$i^2 = j^2 = k^2 = ijk = -1$$

*William Rowan Hamilton*

# Kvaterniák

Szirmay-Kalos László

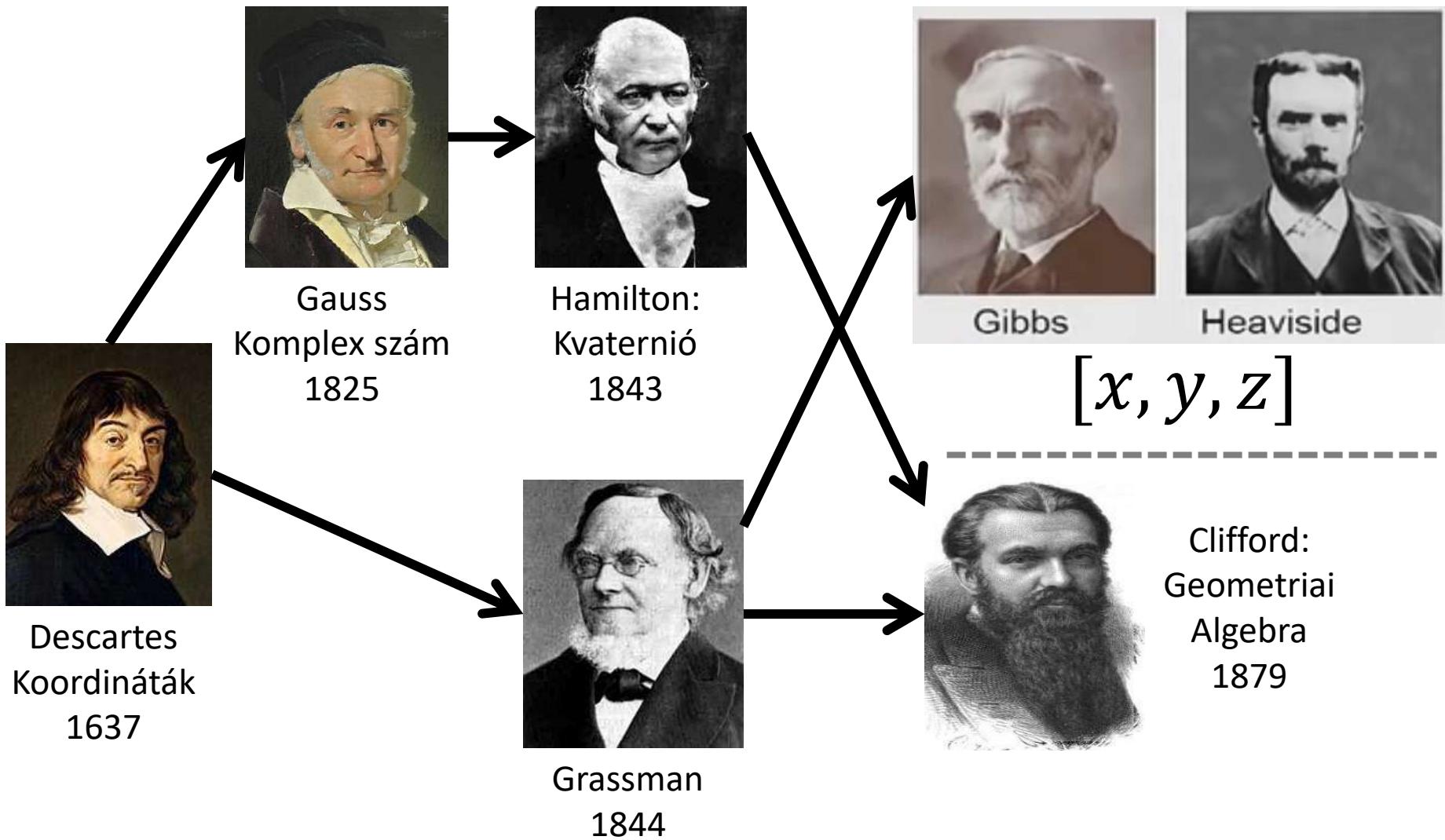


# 2D geometria = vektor algebra

- Pont:  $p = [x, y, 1]$
- Vektor:  $v = [x, y, 0]$
- Eltolás:  $p' = p + v$
- Eltolás, forgatás, skálázás:

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} a & e & 0 \\ b & f & 0 \\ c & d & 1 \end{bmatrix}$$

# Vektor háború



# 2D geometria = komplex szám

- Pont:

$$z_p = x_p + y_p i = Re^{i\alpha} = R \cos \alpha + iR \sin \alpha$$

- Eltolás:  $z_t = x_t + y_t i$

$$z_p' = z_p + z_t$$

- Irányfüggetlen skálázás:  $z_s = s$

$$z_p' = z_p \cdot z_s$$

- Forgatva nyújtás:  $z_r = x_r + y_r i = se^{i\varphi}$

$$z_p' = z_p \cdot z_r = R_s \cdot e^{i(\alpha+\varphi)}$$

- Forgatás = egység abszolút értékű komplex szám

# Komplex számok algebrája

```
struct Complex {
    float x, y;

    Complex(float x0, float y0) { x = x0, y = y0; }

    Complex operator+(Complex r) {
        return Complex(x + r.x, y + r.y);
    }

    Complex operator-(Complex r) {
        return Complex(x - r.x, y - r.y);
    }

    Complex operator*(Complex r) {
        return Complex(x * r.x - y * r.y, x * r.y + y * r.x);
    }

    Complex operator/(Complex r) {
        float l = r.x * r.x + r.y * r.y;
        return (*this) * Complex(r.x / l, -r.y / l); // conjugate
    }

};

Complex Polar(float r, float phi) { // Constructor
    return Complex(r * cosf(phi), r * sinf(phi));
}
```

# 2D transzformációk komplex számokkal

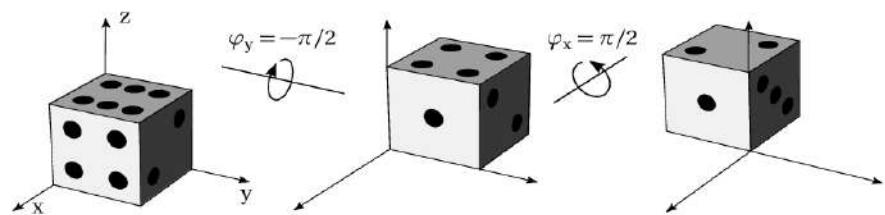
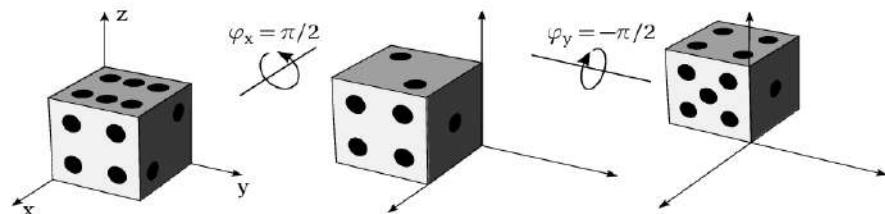
A **p** pontot az **(1,-1)** pivot pont körül nyújtsuk 2-szeresére és forgassuk el t-vel, majd toljuk el a **(2, 3)** vektorral és végül nyújtsuk az origó körül 0.8-szorosára és forgassuk  $-t/2$ -radiánnal:

```
Complex p, tp;  
Complex pivot(1, -1);  
tp = (((p - pivot) * Polar(2,t) + pivot) + Complex(2, 3))  
    * Polar(0.8, -t/2);
```



# Működik 3D-ben?

- $z = x + yi + zj$
- Összeadás és irányfüggetlen skálázás OK
- Forgatás mint szorzás? Tulajdonságok:
  - Asszociatív, összeadásra disztributív (biz: mátrix)
  - Nem kommutatív
  - Invertálható

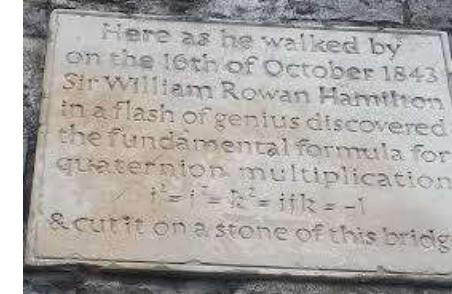


$$- i^2 = ?, \quad j^2 = ?, \quad ij = ?, \quad ji = ?$$



# (Sir William Rowan) Hamilton

## Kvaternió: 4D komplex szám



- $\mathbf{q} = [s, \mathbf{x}, \mathbf{y}, \mathbf{z}] = [s, \mathbf{d}] = s + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$
- $\mathbf{q}_1 + \mathbf{q}_2 = [s_1 + s_2, \mathbf{x}_1 + \mathbf{x}_2, \mathbf{y}_1 + \mathbf{y}_2, \mathbf{z}_1 + \mathbf{z}_2]$
- $a\mathbf{q} = \mathbf{q}a = [as, ax, ay, az]$
- $|\mathbf{q}| = \sqrt{s^2 + x^2 + y^2 + z^2}$
- Szorzás:  
$$[s_1, \mathbf{d}_1] \cdot [s_2, \mathbf{d}_2] = [s_1s_2 - \mathbf{d}_1 \cdot \mathbf{d}_2, s_1\mathbf{d}_2 + s_2\mathbf{d}_1 + \mathbf{d}_1 \times \mathbf{d}_2]$$
  - $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$
  - $\mathbf{ij} = \mathbf{k}, \mathbf{ji} = -\mathbf{k}, \mathbf{jk} = \mathbf{i}, \mathbf{kj} = -\mathbf{i}, \mathbf{ki} = \mathbf{j}, \mathbf{ik} = -\mathbf{j}$
  - Szorzás asszociatív, de nem kommutatív,
  - Összeadásra disztributív
  - Van egységelem:  $[1,0,0,0]$
  - Van inverz:  $\mathbf{q}^{-1} = [s, -\mathbf{d}] / |\mathbf{q}|^2, \quad \mathbf{q}^{-1} \cdot \mathbf{q} = \mathbf{q} \cdot \mathbf{q}^{-1} = [1,0,0,0]$

# Kvaternió = forgatás $\alpha$ szöggel az origón átmenő $d$ irányú tengely körül

$$q = [\cos(\alpha/2), \mathbf{d} \sin(\alpha/2)], \quad |\mathbf{d}| = 1$$

$$q \cdot [0, \mathbf{u}] \cdot q^{-1} = [0, \mathbf{v}], \quad \mathbf{v} \text{ az } \mathbf{u} \text{ elforgatottja a } d \text{ körül } \alpha\text{-val}$$

Rodriguez:  $\mathbf{v} = \mathbf{u} \cos(\alpha) + \mathbf{d}(\mathbf{u} \cdot \mathbf{d})(1 - \cos(\alpha)) + \mathbf{d} \times \mathbf{u} \sin(\alpha)$

Bizonyítás  $d$  merőleges  $u$  esetre (párhuzamos  $\rightarrow$  HF):

Rodriguez:  $\mathbf{v} = \mathbf{u} \cos(\alpha) + \mathbf{d} \times \mathbf{u} \sin(\alpha)$

Kvaternió:



$$[\cos(\alpha/2), \mathbf{d} \sin(\alpha/2)] \cdot [0, \mathbf{u}] = [0, \mathbf{u} \cos(\alpha/2) + \mathbf{d} \times \mathbf{u} \sin(\alpha/2)] = [0, \mathbf{u}^*]$$

$$[0, \mathbf{u}^*] \cdot [\cos(\alpha/2), -\mathbf{d} \sin(\alpha/2)] = [0, \mathbf{u}^* \cos(\alpha/2) - \mathbf{u}^* \times \mathbf{d} \sin(\alpha/2)]$$

$$[s_1, \mathbf{d}_1] \cdot [s_2, \mathbf{d}_2] = [s_1 s_2 - \mathbf{d}_1 \cdot \mathbf{d}_2, s_1 \mathbf{d}_2 + s_2 \mathbf{d}_1 + \mathbf{d}_1 \times \mathbf{d}_2]$$

# Példa

Az  $\mathbf{u} = (1, 0, 0)$  forgatása a  $\mathbf{d} = (0, 0, 1)$  körül  $\alpha$  szöggel:

$$\begin{aligned}\mathbf{q} &= [\cos(\alpha/2), 0, 0, \sin(\alpha/2)] \cdot [0, \mathbf{u}] \cdot [\cos(\alpha/2), 0, 0, -\sin(\alpha/2)] \\ &= (\cos(\alpha/2) + \sin(\alpha/2)\mathbf{k}) \cdot \mathbf{i} \cdot (\cos(\alpha/2) - \sin(\alpha/2)\mathbf{k}) \\ &= (\cos(\alpha/2)\mathbf{i} + \sin(\alpha/2)\mathbf{j}) \cdot (\cos(\alpha/2) - \sin(\alpha/2)\mathbf{k}) \\ &= (\cos^2(\alpha/2)\mathbf{i} + \sin(\alpha/2)\cos(\alpha/2)\mathbf{j} + \cos(\alpha/2)\sin(\alpha/2)\mathbf{j} - \sin^2(\alpha/2)\mathbf{i}) \\ &= (\cos^2(\alpha/2) - \sin^2(\alpha/2))\mathbf{i} + 2\sin(\alpha/2)\cos(\alpha/2)\mathbf{j} \\ &= \cos(\alpha)\mathbf{i} + \sin(\alpha)\mathbf{j}\end{aligned}$$

# Implementáció: $q = s + xi + yj + zk = \text{vec4}$

```
struct vec4 {  
    float x, y, z, w;  
    ...  
};  
  
vec4 qmul(vec4 q1, vec4 q2) { // kvaternió szorzás  
    vec3 d1(q1.x, q1.y, q1.z), d2(q2.x, q2.y, q2.z);  
    return vec4(d2 * q1.w + d1 * q2.w + cross(d1, d2),  
               q1.w * q2.w - dot(d1, d2));  
}  
  
vec4 quaternion(float ang, vec3 axis) { // konstruálás  
    vec3 d = normalize(axis) * sinf(ang / 2);  
    return vec4(d.x, d.y, d.z, cosf(ang / 2));  
}  
  
vec3 Rotate(vec3 u, vec4 q) {  
    vec4 qinv(-q.x, -q.y, -q.z, q.w); // conjugate  
    vec4 qr = qmul(qmul(q, vec4(u.x, u.y, u.z, 0)), qinv);  
    return vec3(qr.x, qr.y, qr.z);  
}
```

# GPU shader programozás GLSL nyelven

```
uniform vec4 q;      // quaternion as uniform variable
in vec3 u;          // Varying input: vertex

vec4 qmul(vec4 q1, vec4 q2) {
    vec3 d1 = q1.xyz, d2 = q2.xyz;
    return vec4(d2 * q1.w + d1 * q2.w + cross(d1, d2),
                q1.w * q2.w - dot(d1, d2));
}

void main() { // vertex shader program
    vec4 qinv = vec4(-q.xyz, q.w); // conjugate
    vec3 v = qmul(qmul(q, vec4(u, 0)), qinv).xyz;
    gl_Position = vec4(v, 1);
}
```



# Kvaterniók

- 2D geometriát a komplex számok is megalapozzák algebrailag
- 3D általánosítás a forgatás asszociativitása és invertálhatósága miatt nem megy közvetlenül
- 4D-ben viszont OK: kvaterniók
- Forgatás = két kvaternió szorzás
- Kvaternió = vec4 speciális szorzással

*“For geometry, you know, is the gate of science, and the gate is so low and small that we can only enter it as a little child.”*

*William Kingdon Clifford*



# Geometriai (Clifford) algebra

Szirmay-Kalos László

# Geometriához használt algebrák állatkertje

## Alakzatok

2D

- Pont:  $[x, y]$ ,  $[x, y, w]$ , komplex szám
- Egyenes:  $ax + by + c = 0$
- Metrika: skaláris szorzás, külső szorzás

## Mozgások (motorok)

- Eltolás: vektor, komplex szám
- **Forgatás, eltolás, skálázás, tükröz, nyírás, vetítés, ...:** 3x3-as mátrix
- 2D forgatva nyújtás: komplex sz.
- Deriválás: duális szám

3D

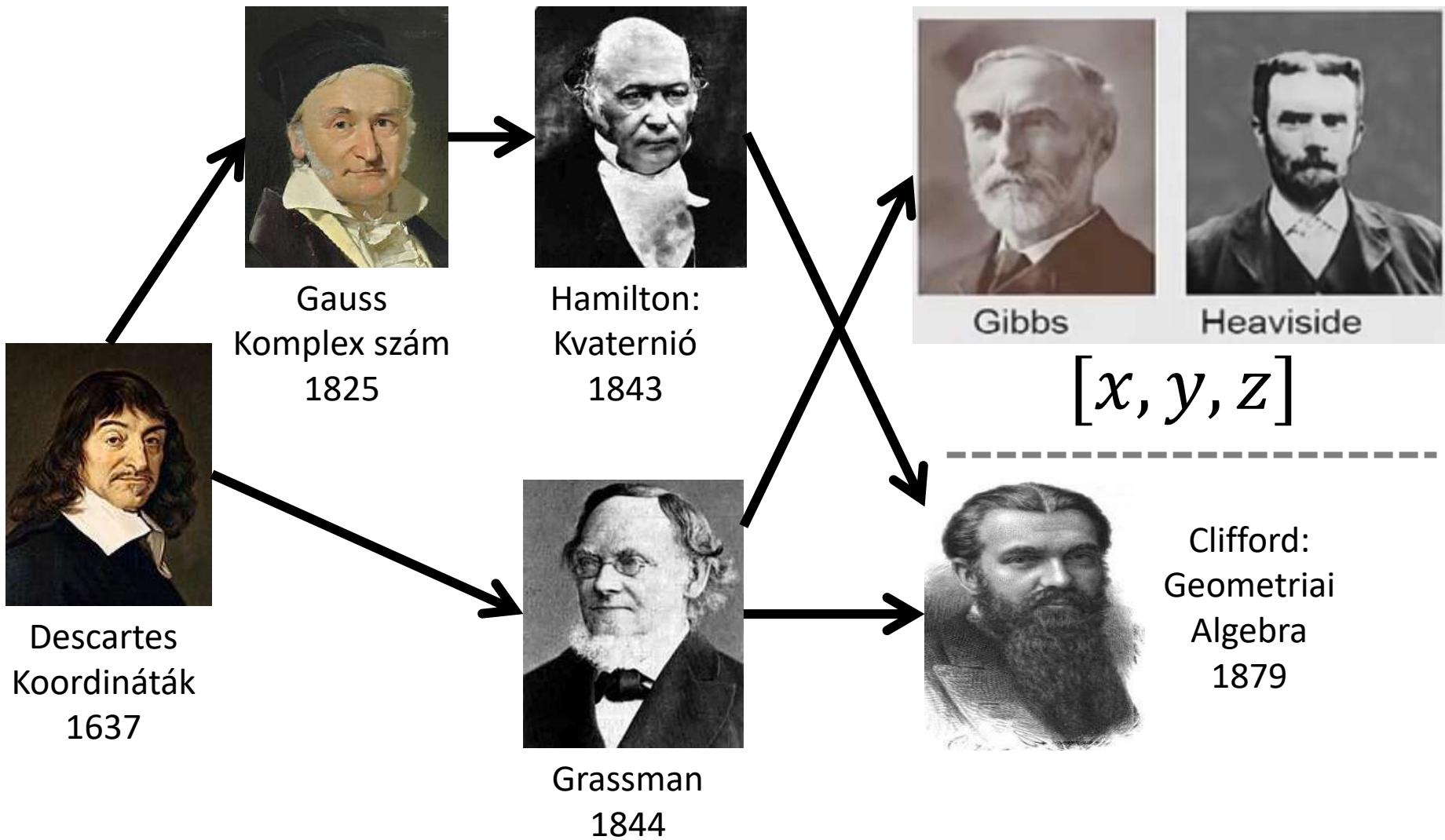
- Pont:  $[x, y, z]$ ,  $[x, y, z, w]$
- Sík:  $ax + by + cz + d = 0$
- Metrika: skaláris szorzás, külső szorzás, vektoriális szorzás

- Eltolás: vektor
- Forgatás, skálázás, tükrözés, nyírás, vetítés, perspektíva, ...: 4x4-es mátrix
- Origón átmentő tengely körül 3D forgatás: kvaternió

# Vektor osztás (csoport)

- Szorzatok: skalár/vektor szorzás nem asszociatív, semelyik sem invertálható (független egyenletek száma kisebb, mint a vektor koordinátáinak száma)
  - $\nu \cdot a = b \Rightarrow |\nu||a| \cos(\alpha) = b$
  - $\nu \times a = b \Rightarrow \nu$  síkja ismert és  $|\nu||a| \sin(\alpha) = |b|$
  - A (skaláris + külső) invertálható volna
- Megoldás: **Geometriai (Clifford) algebra**
  - Az irányított szakasz (=vektor) mellett, irányított terület (=bivektor), térfogat (=trivektor), stb. is részt vesznek.
  - 2D-ben 4 elemű bázis: skalár + 2 vektor + 1 bivektor
  - 3D-ben 8 elemű: (skalár + 3 vektor + 3 bivektor + 1 trivektor)
- Speciális esetek:
  - Komplex szám, Duális szám, Kvaternió, ...

# Vektor háború



# Geometriai számok és szorzat

- Vektor:  $\boldsymbol{v} = x\mathbf{e}_1 + y\mathbf{e}_2$       Ortogonális:  $\mathbf{e}_1 \cdot \mathbf{e}_2 = 0$
- Szorzás: asszociatív, disztributív, invertálható
- Kapcsolat a valós számokkal:  $\boldsymbol{v}\boldsymbol{v}$  legyen valós

$$\begin{aligned}\boldsymbol{v}\boldsymbol{v} &= (x\mathbf{e}_1 + y\mathbf{e}_2)(x\mathbf{e}_0 + y\mathbf{e}_1) \\ &= x^2\mathbf{e}_1\mathbf{e}_1 + y^2\mathbf{e}_2\mathbf{e}_2 + xy(\mathbf{e}_1\mathbf{e}_2 + \mathbf{e}_2\mathbf{e}_1)\end{aligned}$$

- Bázis (geometriai számok):  $\mathbf{e}_k\mathbf{e}_k = 1, \quad 0, \quad -1?$
- Antiszimmetrikus:  $\mathbf{e}_{12} = \mathbf{e}_1\mathbf{e}_2 = -\mathbf{e}_2\mathbf{e}_1 = -\mathbf{e}_{21}$

- Geometriai (Clifford) szorzat:

$$\begin{aligned}\boldsymbol{v}_1\boldsymbol{v}_2 &= (x_1\mathbf{e}_1 + y_1\mathbf{e}_2)(x_2\mathbf{e}_1 + y_2\mathbf{e}_2) \\ &= x_1x_2 + y_1y_2 + (x_1y_2 - x_2y_1)\mathbf{e}_1\mathbf{e}_2\end{aligned}$$

$$\boldsymbol{v}_1\boldsymbol{v}_2 = \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 + \boldsymbol{v}_1 \wedge \boldsymbol{v}_2$$

Inverz:

$$\boldsymbol{v}^{-1} = \frac{\boldsymbol{v}}{\boldsymbol{v}\boldsymbol{v}}$$

# 2D geometriai algebra

0 dim      1 dim      2 dim

- Multivektor:  $\mathbf{V} = s + \underline{x\mathbf{e}_1 + y\mathbf{e}_2} + B\mathbf{e}_{12}$
- Összeadás, skálázás a szokásos módon
- Szorzás (asszociatív, disztributív, nemkommutatív):
  - skalár és bármi: a szokásos
  - Pszeudó-skalár és pszeudó-skalár:

$$\mathbf{e}_{12}\mathbf{e}_{12} = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_1\mathbf{e}_2 = -\mathbf{e}_1\mathbf{e}_1\mathbf{e}_2\mathbf{e}_2 = -1$$

-1                  1    1  
                      \underbrace{\hspace{1cm}}\_{\mathbf{e}\_1} \underbrace{\hspace{1cm}}\_{\mathbf{e}\_2}

Ha  $x = y = 0$ , akkor a komplex számokat kapjuk:  $\mathbf{e}_{12} = I$

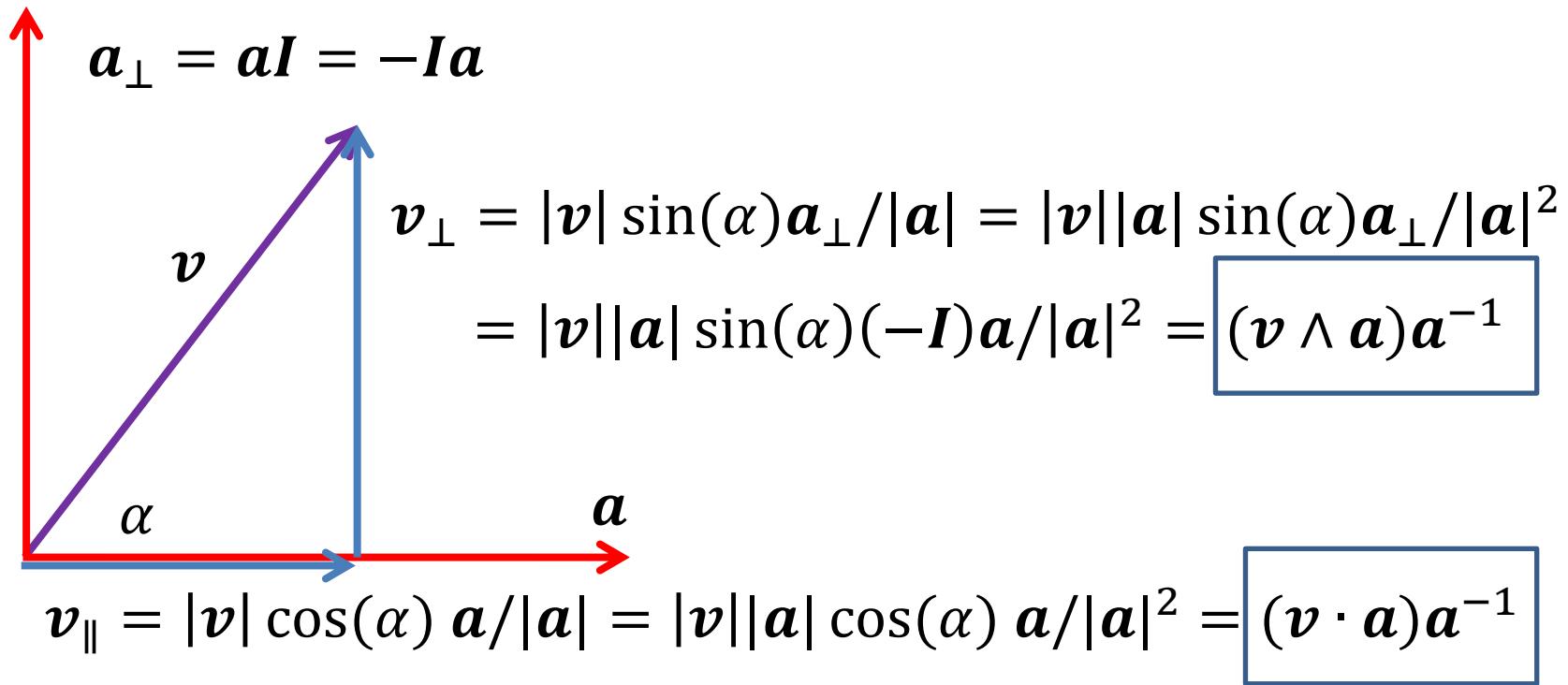
- Pszeudó-skalárral jobbról/balról:  $\pm 90^\circ$ -os forgatás
- $$(\mathbf{x}\mathbf{e}_1 + \mathbf{y}\mathbf{e}_2)\mathbf{e}_{12} = \mathbf{x}\mathbf{e}_1\mathbf{e}_1\mathbf{e}_2 + \mathbf{y}\mathbf{e}_2\mathbf{e}_1\mathbf{e}_2 = -\mathbf{y}\mathbf{e}_1 + \mathbf{x}\mathbf{e}_2$$

# Szorzótábla

	1	$e_1$	$e_2$	$I$
1	1	$e_1$	$e_2$	$I$
$e_1$	$e_1$	1	$I$	$e_2$
$e_2$	$e_2$	$-I$	1	$-e_1$
$I$	$I$	$-e_2$	$e_1$	-1

$$\mathbf{V} = s + x\mathbf{e}_1 + y\mathbf{e}_2 + BI$$

# Projection és Rejection



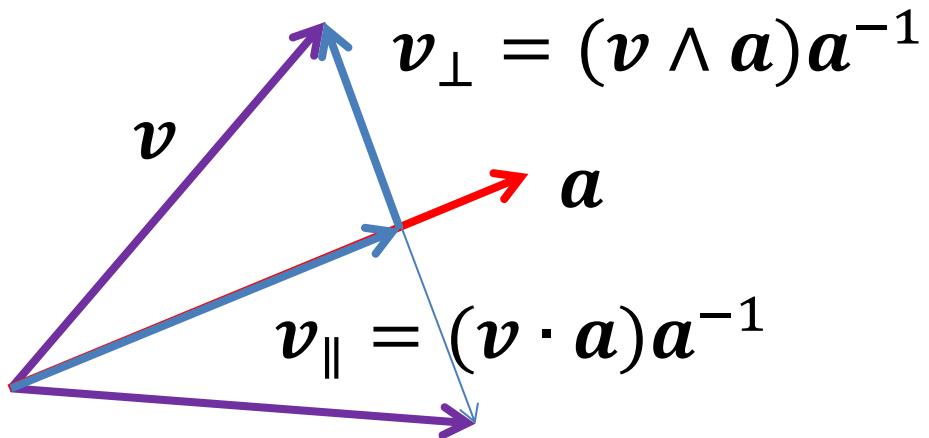
Vektorokra:

$$va = v \cdot a + v \wedge a \Rightarrow$$

$$v \cdot a = (va + av)/2$$

$$v \wedge a = (va - av)/2$$

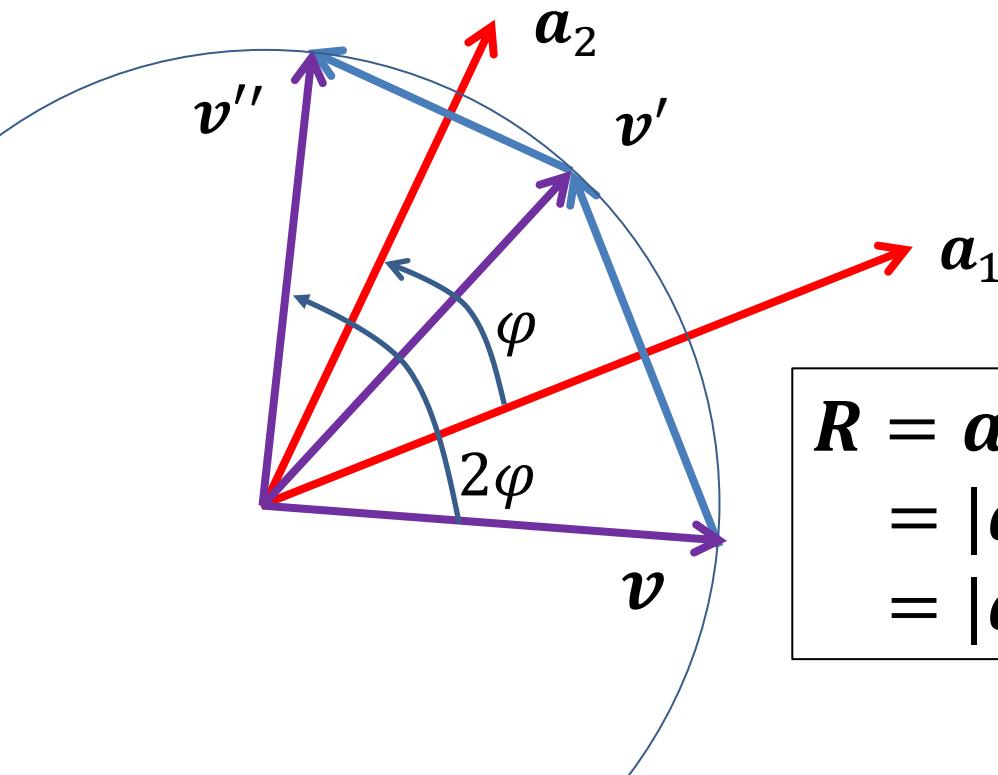
# Tükörözés és szendvics



$$\begin{aligned}\mathbf{v}' &= \mathbf{v}_{\parallel} - \mathbf{v}_{\perp} = (\mathbf{v} \cdot \mathbf{a})\mathbf{a}^{-1} - (\mathbf{v} \wedge \mathbf{a})\mathbf{a}^{-1} \\ &= (\mathbf{v} \cdot \mathbf{a} - \mathbf{v} \wedge \mathbf{a})\mathbf{a}^{-1} = (\mathbf{a} \cdot \mathbf{v} + \mathbf{a} \wedge \mathbf{v})\mathbf{a}^{-1}\end{aligned}$$

$$\boxed{\mathbf{v}' = \mathbf{a} \mathbf{v} \mathbf{a}^{-1}}$$

# Forgatás = 2 tükrözés



$$\begin{aligned} R &= a_1 a_2 \\ &= |a_1| |a_2| (\cos(\varphi) + \sin(\varphi) I) \\ &= |a_1| |a_2| \exp(\varphi I) \end{aligned}$$

$$v'' = a_2 v' a_2^{-1} = a_2 a_1 v' a_1^{-1} a_2^{-1} = (a_2 a_1) v (a_2 a_1)^{-1}$$

$$v'' = R v R^{-1} = (\cos(\varphi) - \sin(\varphi) I) v (\cos(\varphi) + \sin(\varphi) I)$$

# Exponentiation

- A transzformációk szorzással működnek (multiplikatív csoport):  $(\mathbf{R}_2 \mathbf{R}_1) \mathbf{v} (\mathbf{R}_2 \mathbf{R}_1)^{-1}$
- Rotor:  $\mathbf{R}(\varphi_2 + \varphi_1) = \mathbf{R}(\varphi_2) \mathbf{R}(\varphi_1)$
- Hatványfüggvény:  $\mathbf{R}(\varphi) = a^\varphi = \exp(b\varphi)$
- Ha  $\varphi = \pi$ , akkor  $(-\mathbf{I}) \mathbf{v} \mathbf{I} \Rightarrow \mathbf{R}(\varphi) = \exp(\mathbf{I}\varphi)$

# 3 dimenzió

- Vektor:  $\boldsymbol{v} = x\boldsymbol{e}_1 + y\boldsymbol{e}_2 + z\boldsymbol{e}_3$
- Geometriai (Clifford) szorzat:

$$\boldsymbol{v}_1 \boldsymbol{v}_2 = \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 + \boldsymbol{v}_1 \wedge \boldsymbol{v}_2 \quad \text{Inverz: } \boldsymbol{v}^{-1} = \frac{\boldsymbol{v}}{\boldsymbol{v}\boldsymbol{v}}$$

- Ortogonális bázis:  $\boldsymbol{e}_1 \cdot \boldsymbol{e}_2 = \boldsymbol{e}_2 \cdot \boldsymbol{e}_3 = \boldsymbol{e}_3 \cdot \boldsymbol{e}_1 = 0$
- Antiszimmetrikus:  $i \neq j$

$$\boldsymbol{e}_{ij} = \boldsymbol{e}_i \boldsymbol{e}_j = \boldsymbol{e}_i \wedge \boldsymbol{e}_j = -\boldsymbol{e}_j \wedge \boldsymbol{e}_i = -\boldsymbol{e}_j \boldsymbol{e}_i = -\boldsymbol{e}_{ji}$$

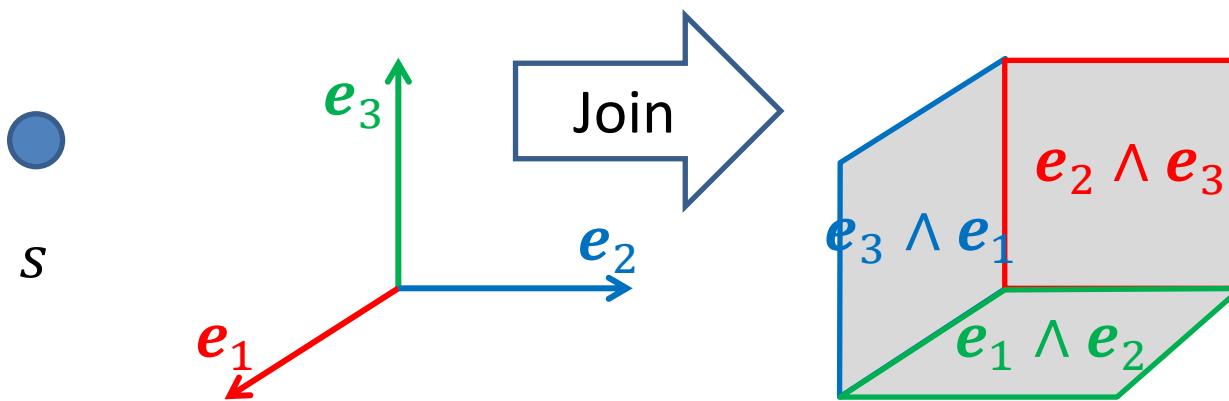
- Trivektor:  $\boldsymbol{I} = \boldsymbol{e}_1 \boldsymbol{e}_2 \boldsymbol{e}_3$
- Geometriai számok:  $\boldsymbol{e}_k \boldsymbol{e}_k = 1$

$$(\boldsymbol{e}_{ij})^2 = \boldsymbol{e}_i \boldsymbol{e}_j \boldsymbol{e}_i \boldsymbol{e}_j = -1$$

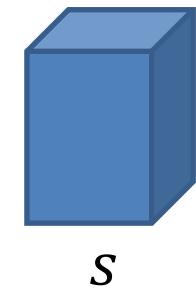
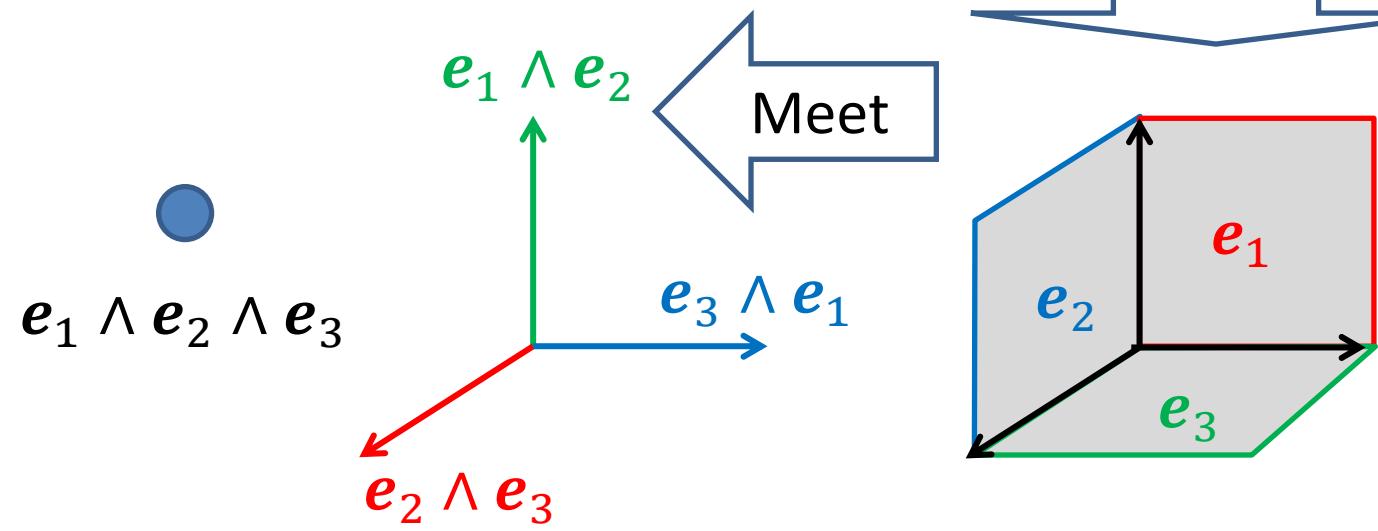
$$\boldsymbol{I}^2 = \boldsymbol{e}_1 \boldsymbol{e}_2 \boldsymbol{e}_3 \boldsymbol{e}_1 \boldsymbol{e}_2 \boldsymbol{e}_3 = \boldsymbol{e}_1 \boldsymbol{e}_1 \boldsymbol{e}_2 \boldsymbol{e}_3 \boldsymbol{e}_2 \boldsymbol{e}_3 = -1$$

# Dualitás

Pont alapú



$$e_1 \wedge e_2 \wedge e_3$$



$S$

Sík alapú

## 2D geometria: 3D-be ágyazott sík-alapú modell

- Sík-alapú: vektor = tükrözés, sík az invariánsa  
bivektor = forgatás, az egyenes az invariánsa
- Beágyazás: Eltolást is szeretnénk (origó nem része)

– Egyenes:  $ax + by + c = 0 \Rightarrow \mathbf{l} = a\mathbf{e}_1 + b\mathbf{e}_2 + c\mathbf{e}_3$

– Pont:  $\mathbf{p} = X\mathbf{e}_2\mathbf{e}_3 + Y\mathbf{e}_1\mathbf{e}_3 + w\mathbf{e}_1\mathbf{e}_2$

– Geometriai számok:  $\mathbf{e}_1\mathbf{e}_1 = \mathbf{e}_2\mathbf{e}_2 = 1$ ,  $\mathbf{e}_3\mathbf{e}_3 = 0$

– 1. Ok: két egyenes szöge:

$$\mathbf{l}_1 \cdot \mathbf{l}_2 = a_1 a_2 \mathbf{e}_1^2 + b_1 b_2 \mathbf{e}_2^2 + c_1 c_2 \mathbf{e}_3^2$$

– 2. Ok: Eltolás

$$(\mathbf{e}_{12})^2 = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_1\mathbf{e}_2 = -1 \Rightarrow \text{egy fajta forgatás}$$

$$(\mathbf{e}_{13})^2 = \mathbf{e}_1\mathbf{e}_3\mathbf{e}_1\mathbf{e}_3 = (\mathbf{e}_{23})^2 = 0 \Rightarrow \text{két fajta eltolás}$$

# Ellenőrző kérdések

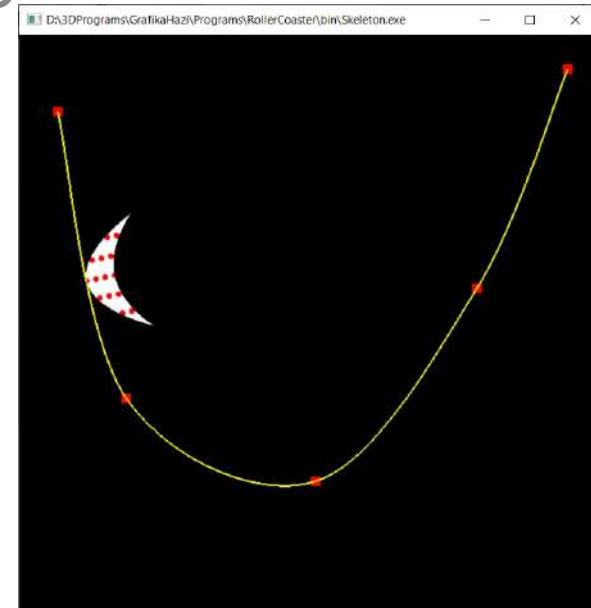
- Bizonyítsa be, hogy ha a transzformált  $x'$ ,  $y'$  koordináták az eredeti  $x$ ,  $y$ -nak lineáris függvényei, akkor a transzformáció egyenest egyenesbe képez le és a párhuzamos egyeneseket megtartja!
- Lehet-e egy affin transzformációnak olyan mátrixa, ahol az utolsó oszlop nem  $[0, 0, 0, 1]$ ?
- Írja fel az adott irányú, origón átmenő tengely körül alfa szöggel forgató transzformáció mátrixát!
- Írja fel a vektoriális szorzást mátrixművelettel?
- Írja fel egy síkra merőlegesen vetítő, illetve centrálisan vetítő transzformációk mátrixait!
- Hogyan oldható fel az átfordulási probléma?
- Milyen alakzat az összes ideális pontot tartalmazó halmaz?
- Írja fel egy parabola egyenletét a projektív síkon!
- Határozza meg két párhuzamos egyenes metszéspontjának (homogén) koordinátáit a projektív síkon!
- Adjon meg transzformációt, amely egy adott háromszöget egy másik adott háromszögbe képez le!
- *Adjon meg transzformációt, amely egy konvex négyyszöget egy konvex négyiszögbe képez le! Mi történik, ha a négyiszög nem konvex?*

*"Photographers don't take pictures.  
They create images."*

*Mark Denman*

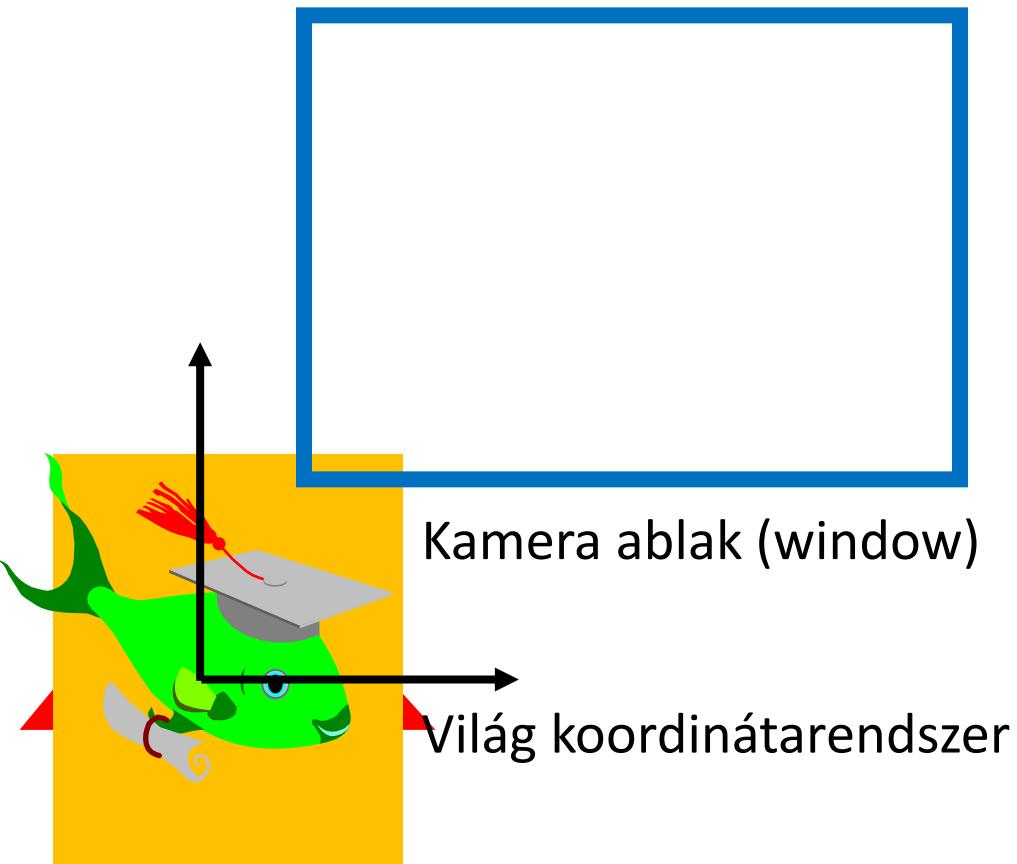
# 2D képszintézis

Szirmay-Kalos László

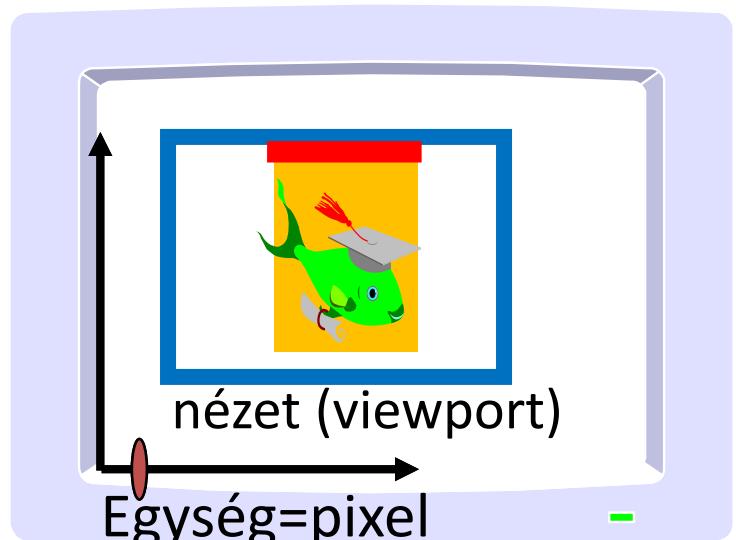


# 2D képszintézis

Modell



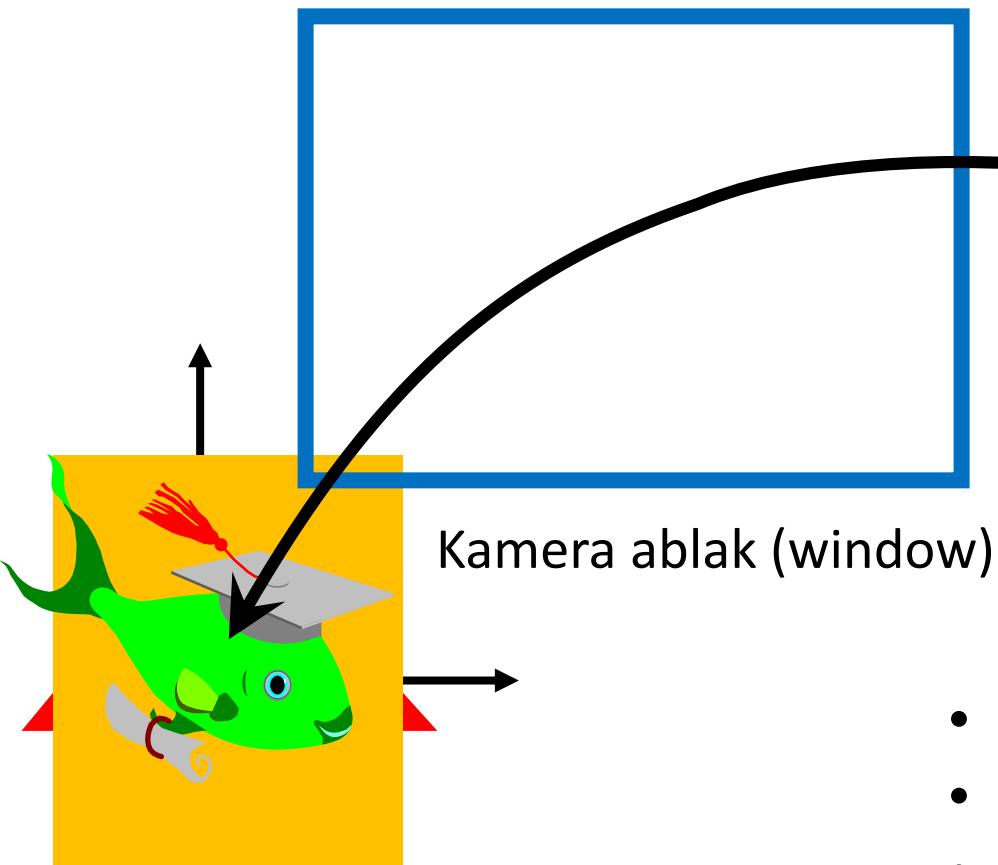
Kép



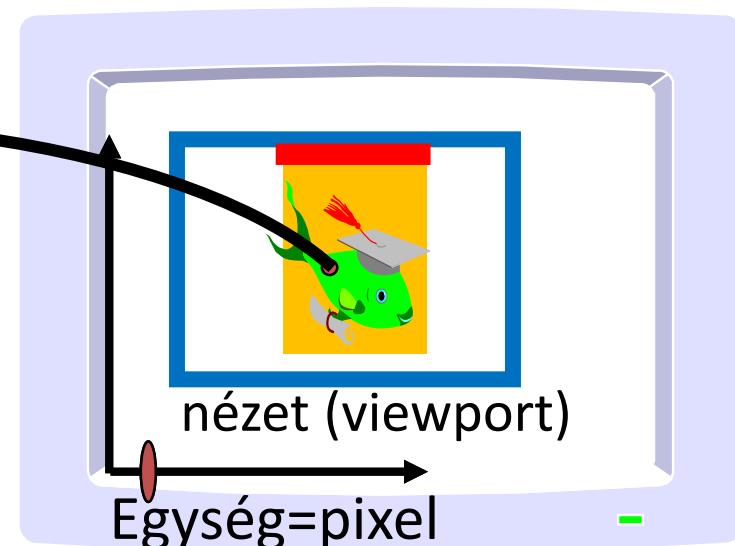
Saját színnel rajzolás

# Pixel vezérelt 2D képszintézis

Modell



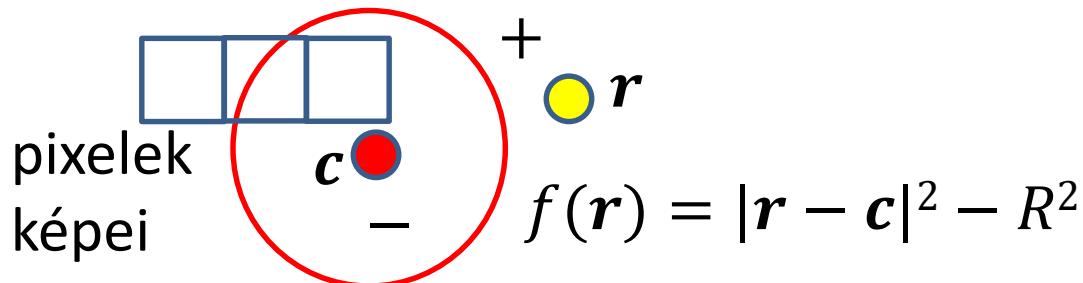
Kép



- Pixeleket (pontokat) transzformál
- Bármilyen transzformáció jó
- Pont tartalmazás teszt
- Lassú

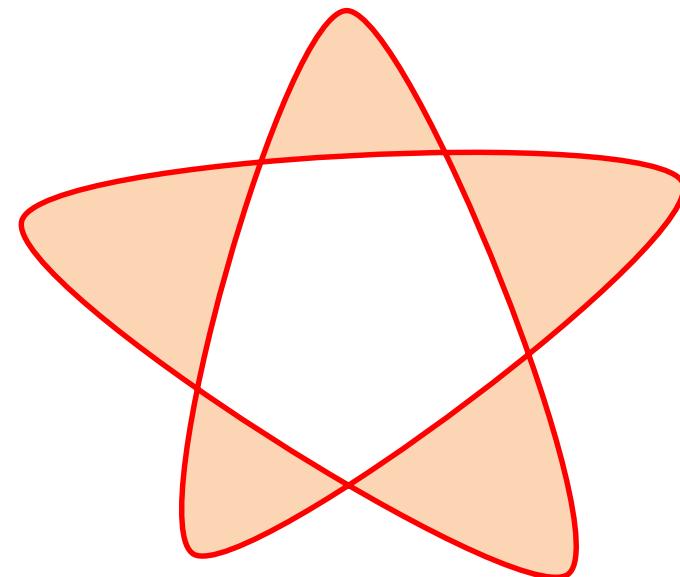
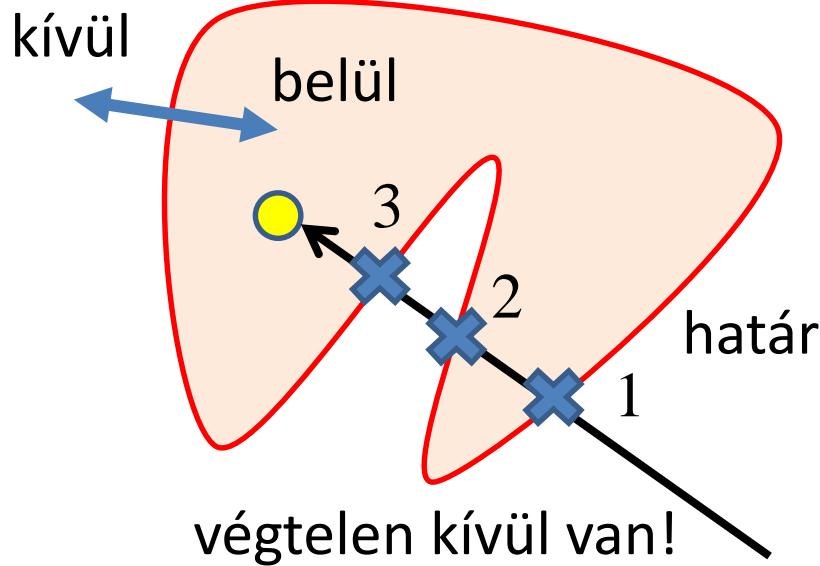
# Pixel vezérelt megközelítés: Tartalmazás (objektum, pont)

- Határ implicit görbe:



$> 0$  : egyik oldalon  
 $f(x, y) = 0$  : határon  
 $< 0$  : másik oldalon  
(ált. belül)

- Határ parametrikus görbe:



# Pixel vezérelt rendering

```
struct Object { // base class
    vec3 color;
    virtual bool In(vec2 r) = 0; // containment test
};

struct Circle : Object {
    vec2 center;
    float R;
    bool In(vec2 r) { return (dot(r-center, r-center)-R*R < 0); }
};

struct HalfPlane : Object {
    vec2 r0, n; // position vec, normal vec
    bool In(vec2 r) { return (dot(r-r0, n) < 0); }
};

struct GeneralEllipse : Object {
    vec2 f1, f2;
    float C;
    bool In(vec2 r) { return (length(r-f1) + length(r-f2) < C); }
};

struct Parabola : Object {
    vec2 f, r0, n; // f=focus, (r0,n)=directrix line, n=unit vec
    bool In(vec2 r) { return (fabs(dot(r-r0, n)) > length(r-f)); }
};
```

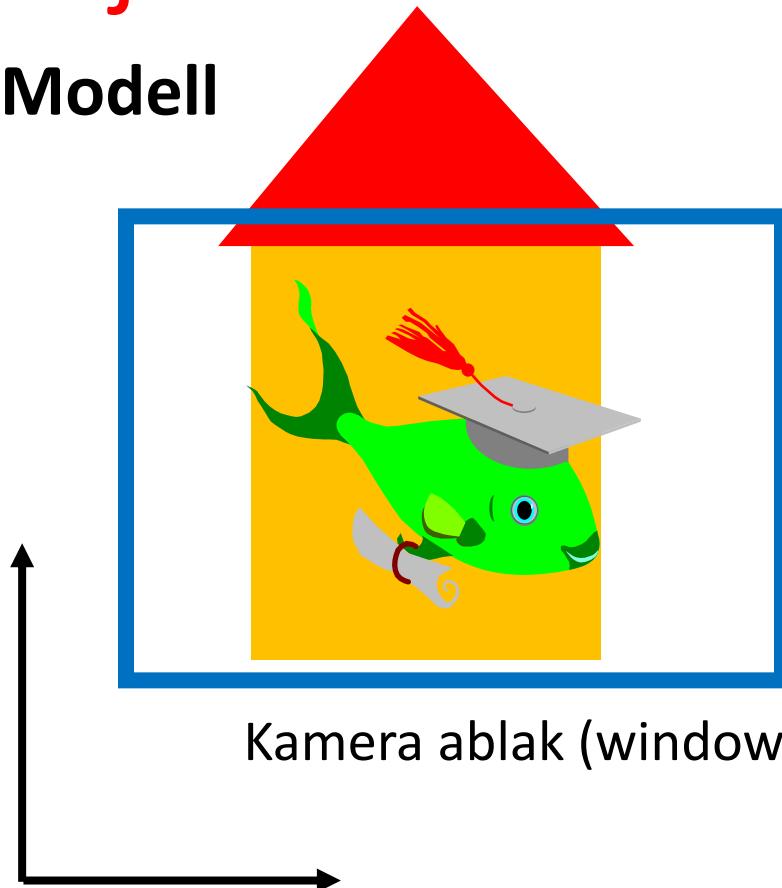
# Pixel vezérelt rendering



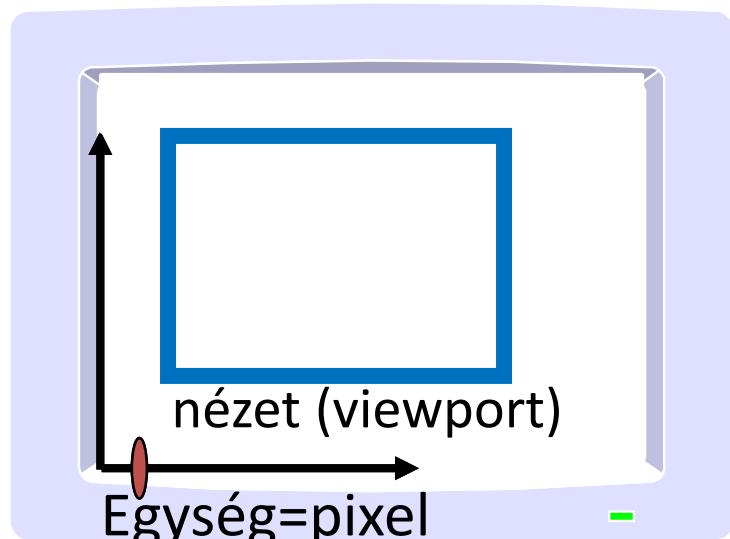
```
class Scene {                                // virtual world
    list<Object *> objs;                  // objects with decreasing priority
    Object *picked = nullptr;   // selected for operation
public:
    void Add(Object * o) { objects.push_front(o); picked = o; }
    void Pick(int pX, int pY) { // pX, pY: pixel coordinates
        vec2 wPoint = Viewport2Window(pX, pY); // transform to world
        picked = nullptr;
        for(auto o : objs) if (o->In(wPoint)) { picked = o; return; }
    }
    void BringToFront() {
        if (picked) { // move to the front of the priority list
            objs.erase(find(objs.begin(), objs.end(), picked));
            objs.push_front(picked);
        }
    }
    void Render() {
        for(int pX=0; pX<xmax; pX++) for(int pY=0; pY<ymax; pY++) {
            vec2 wPoint = Viewport2Window(pX, pY); // wPoint.x=a*pX+b*pY+c
            for(auto o : objs) // object covers the pixel
                if (o->In(wPoint)) { image[pY][pX] = o->color; break; }
        }
    }
};
```

# Objektum vezérelt 2D képszintézis

Modell

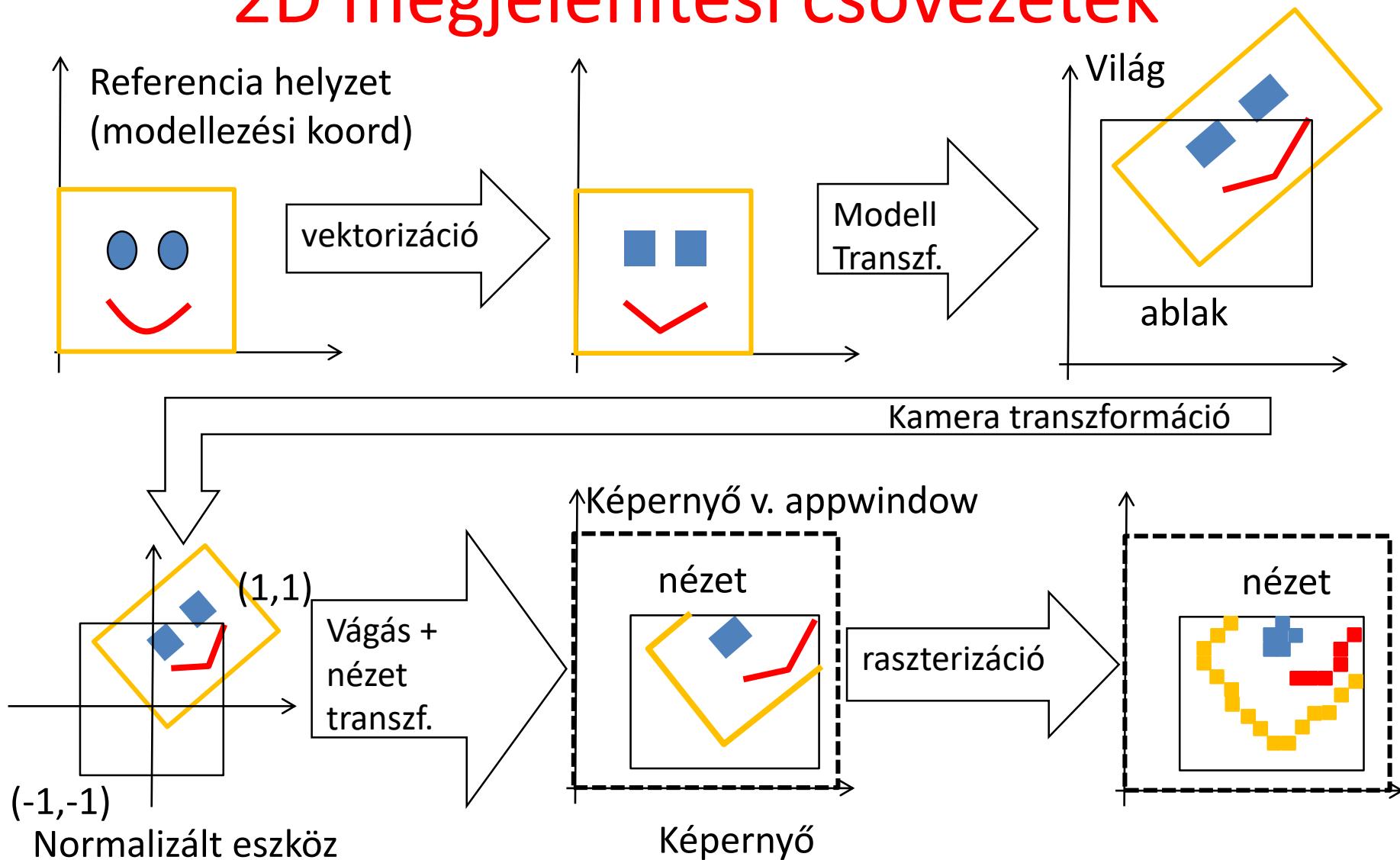


Kép



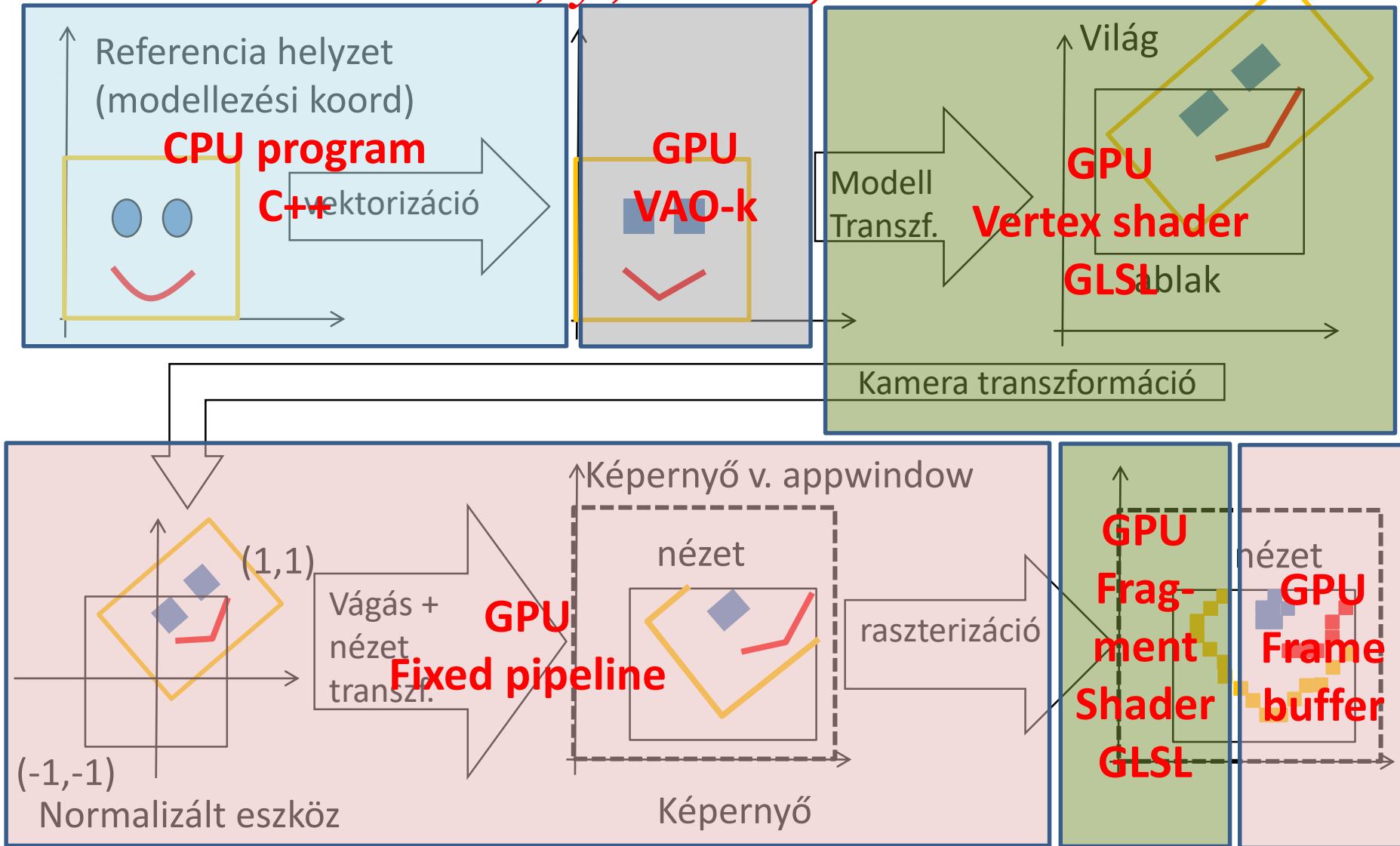
Saját színnel rajzolás  
a kis prioritásúakkal kezdve

# Objektum vezérelt megközelítés: 2D megjelenítési csővezeték



# GPU megjelenítési csővezeték

2D:  $x, y, z = 0, w = 1$

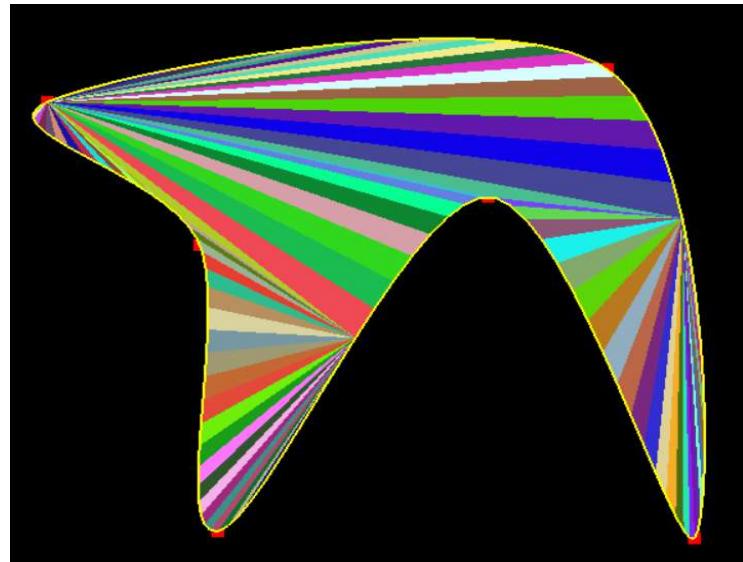


"μή μου τοὺς κύκλους τάραττε."  
Ἀρχιμήδης

# 2D képszintézis

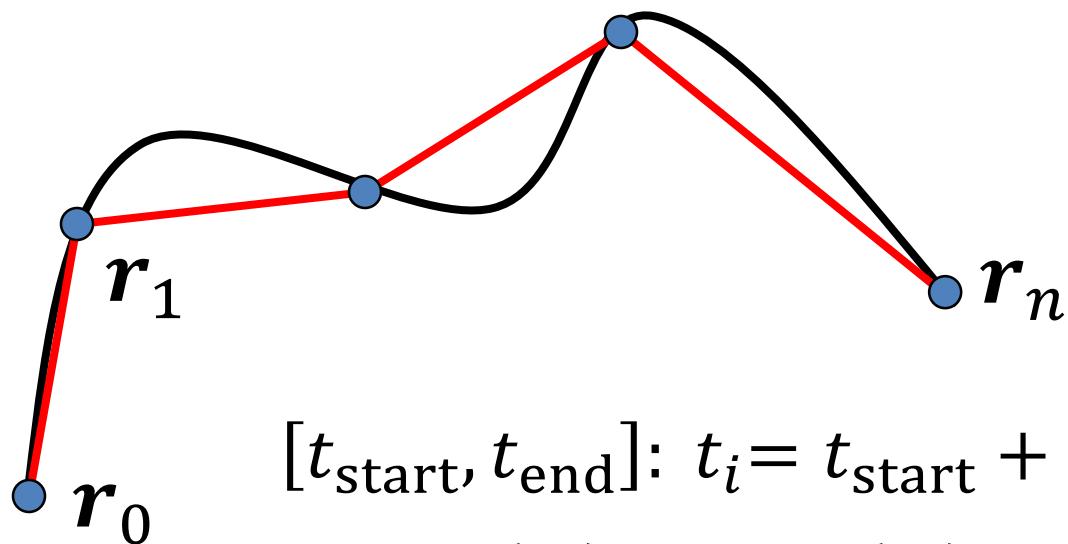
## 2. Vektorizáció és háromszögesítés

Szirmay-Kalos László



# Vektorizáció (CPU)

$$\mathbf{r}(t), t \in [t_{\text{start}}, t_{\text{end}}]$$



$$[t_{\text{start}}, t_{\text{end}}]: t_i = t_{\text{start}} + (t_{\text{end}} - t_{\text{start}})i/n$$

$$\mathbf{r}_0 = \mathbf{r}(t_0), \mathbf{r}_1 = \mathbf{r}(t_1), \dots, \mathbf{r}_n = \mathbf{r}(t_n)$$

Görbe

→ nyílt törötvonal

Hw érdekében

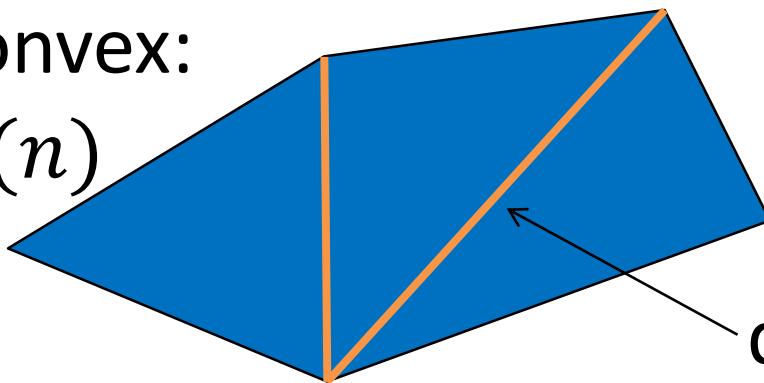
→ szakaszok

Terület határa → zárt törötvonal = poligon → háromszögek

# Poligon háromszögekre bontása

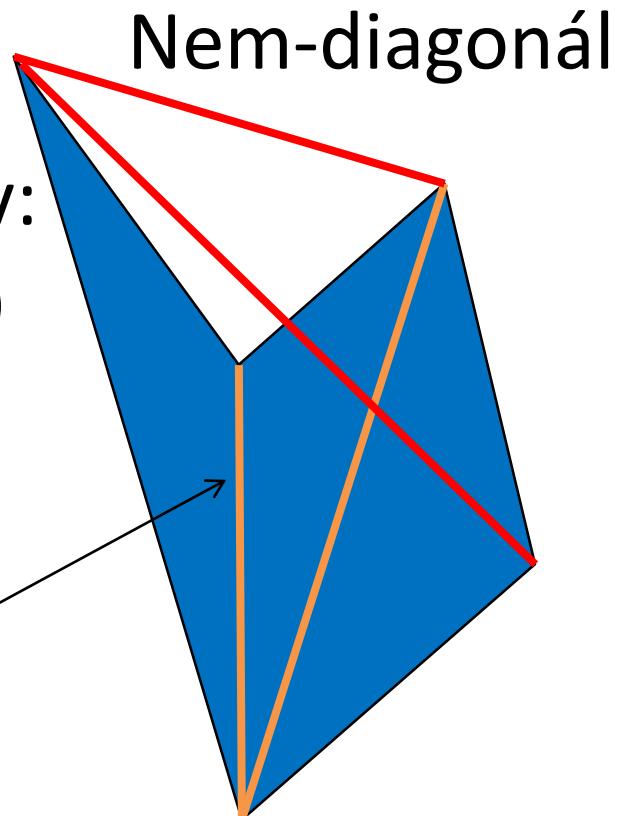
Diagonál mentén értelmes vágni,  
mert az csökkenti a csúcspontok  
számát!

Konvex:  
 $O(n)$



diagonál

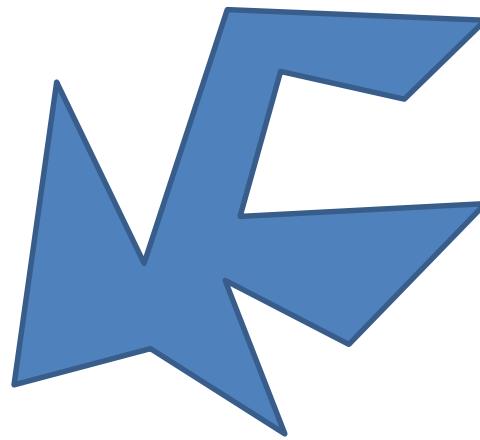
Konkáv:  
 $O(n^4)$



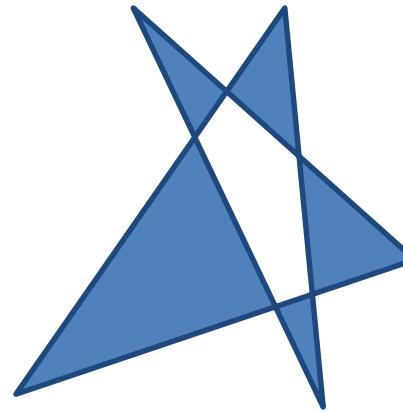
Nem-diagonál

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

Dia  
me  
szá  
Ko  
0

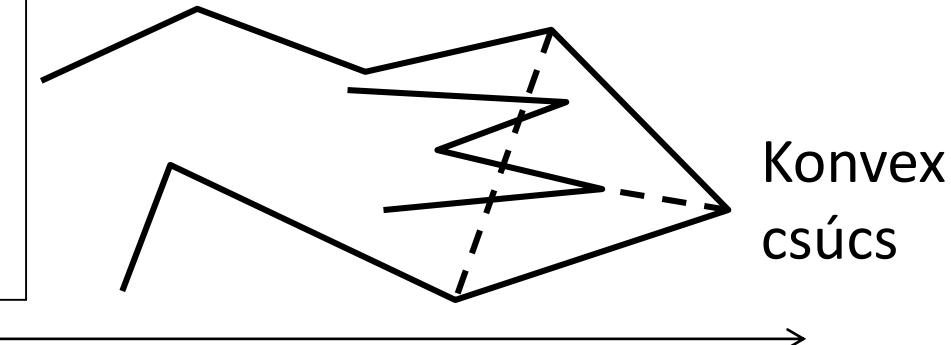


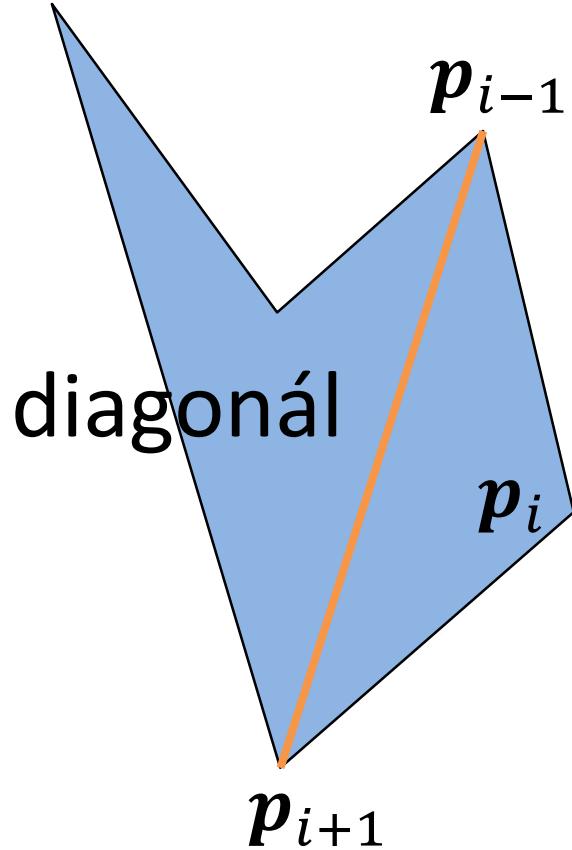
Egyszerű sokszög



Nem egyszerű sokszög

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

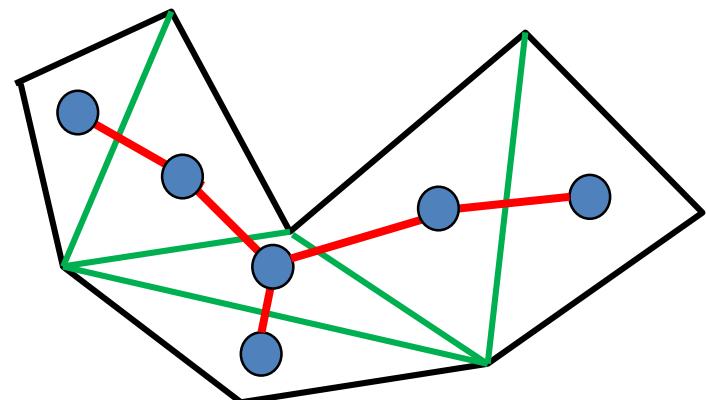




**Két fül tétele:** minden legalább 4 csúcsú egyszerű sokszögnek van legalább 2 füle.

## Fül

- $p_i$  fül, ha  $p_{i-1} \leftrightarrow p_{i+1}$  diagonál
- Fül levágható!
- **Fülvágás:** keress fület és nyissz!
- $O(n^3)$

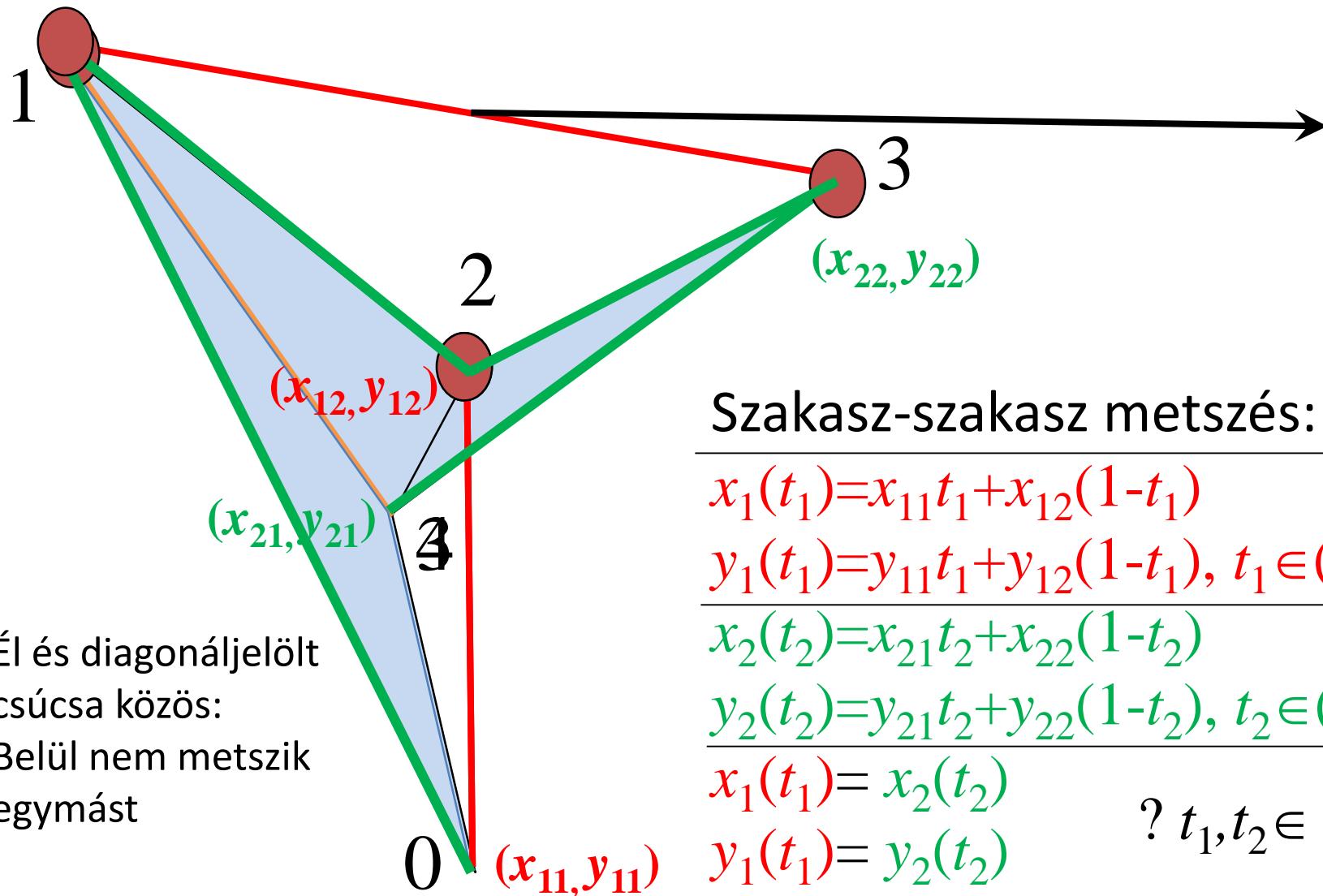


„Minden fának van legalább két levele.”

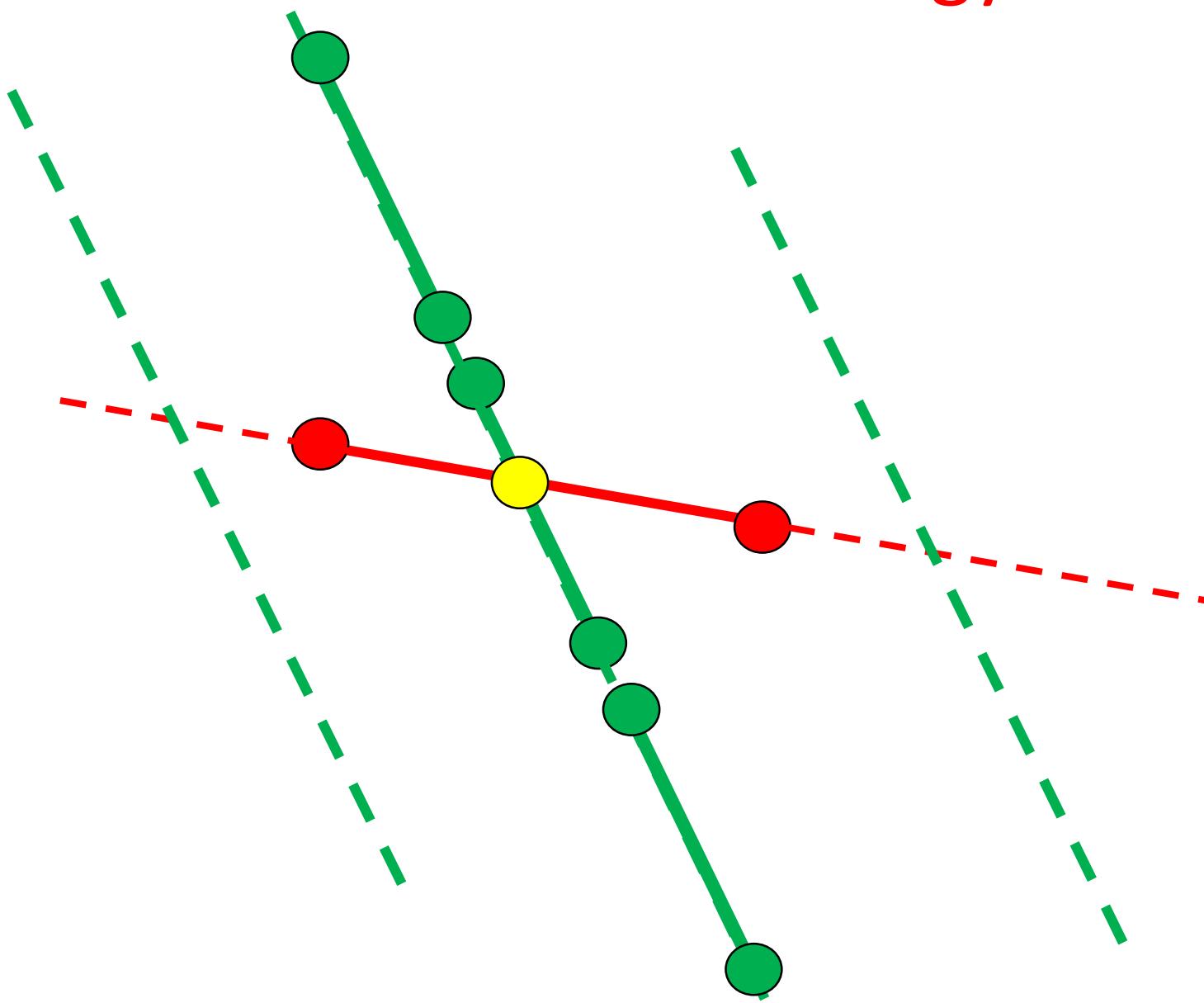




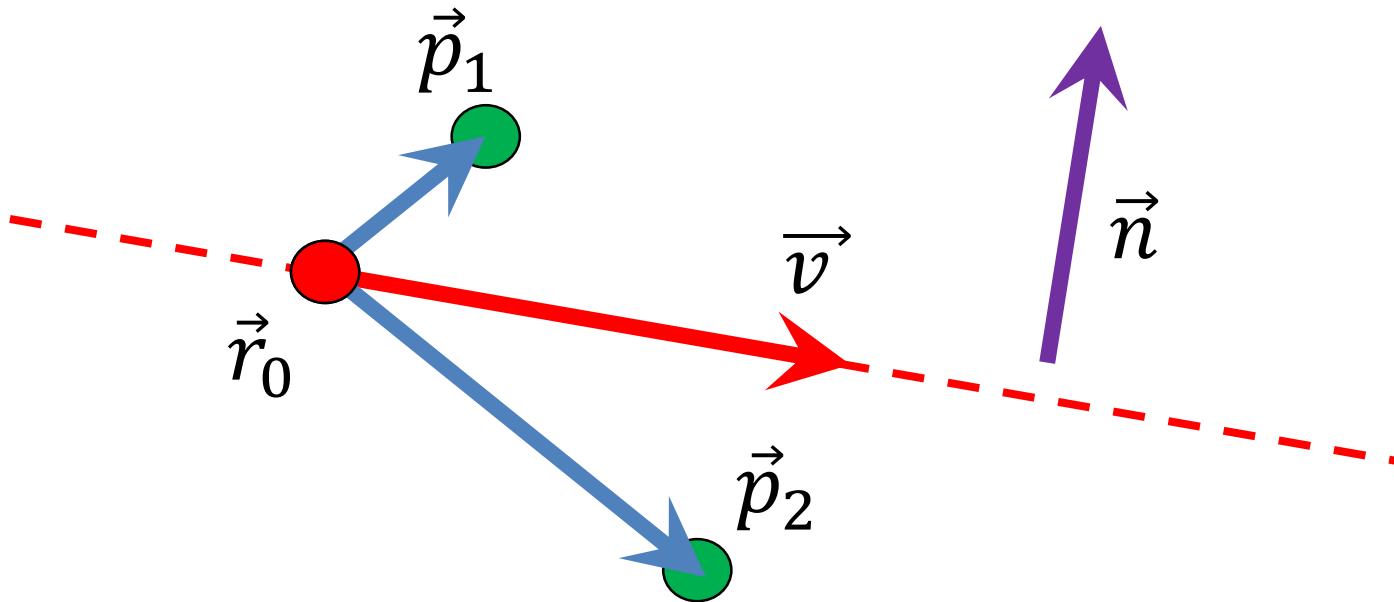
# Fülvágó algoritmus: $O(n^3)$



# Szakasz-szakasz metszés egyszerűbben

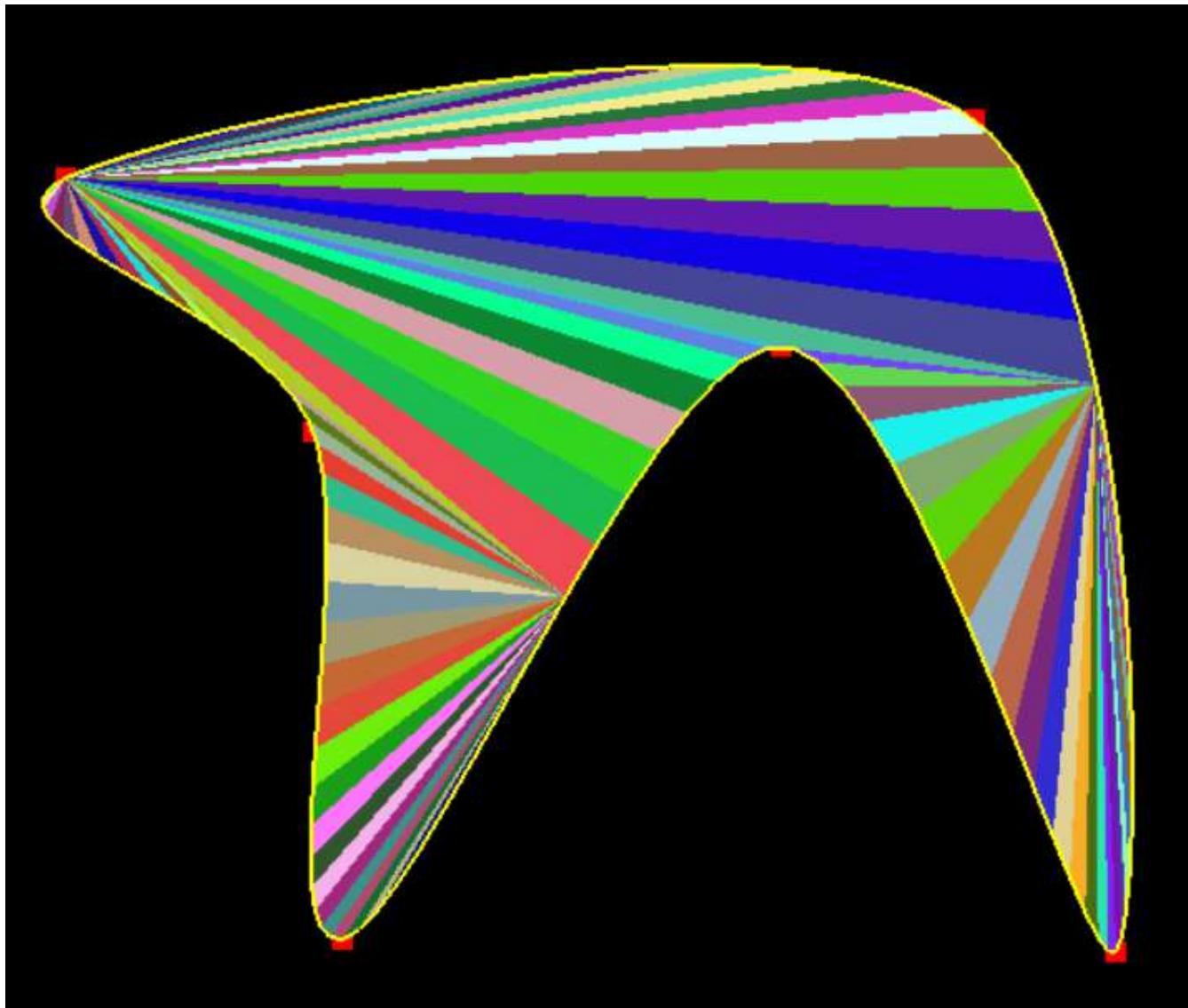


# Két pont az egyenes két oldalán van



1.  $(\vec{n} \cdot (\vec{p}_1 - \vec{r}_0)) (\vec{n} \cdot (\vec{p}_2 - \vec{r}_0)) < 0$
2.  $(\vec{v} \times (\vec{p}_1 - \vec{r}_0)) \cdot (\vec{v} \times (\vec{p}_2 - \vec{r}_0)) < 0$

# Fülvágás eredmények



"μή μου τοὺς κύκλους τάραττε."

Ἀρχιμήδης

## 2D képszintézis

### 3. Transzformációk és vágás

Szirmay-Kalos László



# Modellezési transzformáció

- Mátrixokat a CPU-n számítjuk, a transzformációt a GPU hajtja végre
- Homogén lineáris transzformáció:

$$[x_{\text{world}}, y_{\text{world}}, z_{\text{world}}, w_{\text{world}}] = [x_{\text{model}}, y_{\text{model}}, z_{\text{model}}, 1] \cdot T_{4 \times 4}$$

- Speciális eset: 2D affin modellezési transzformáció:

$$T_{4 \times 4} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & * & 0 \\ v_x & v_y & 0 & 1 \end{bmatrix}$$

# mat4 osztály

```
struct mat4 { // row-major matrix 4x4
    vec4 rows[4];

    mat4(vec4& it, vec4& jt, vec4& kt, vec4& ot) {
        rows[0]=it; rows[1]=jt; rows[2]=kt; rows[3]=ot;
    }
    vec4& operator[](int i) { return rows[i]; }
};

inline vec4 operator*(vec4& v, mat4& m) {
    return v.x*m[0] + v.y*m[1] + v.z*m[2] + v.w*m[3];
}

inline mat4 operator*(mat4& ml, mat4& mr) {
    mat4 res;
    for (int i = 0; i < 4; i++)
        res.rows[i] = ml.rows[i] * mr;
    return res;
}
```

# mat4 „konstruktorok”

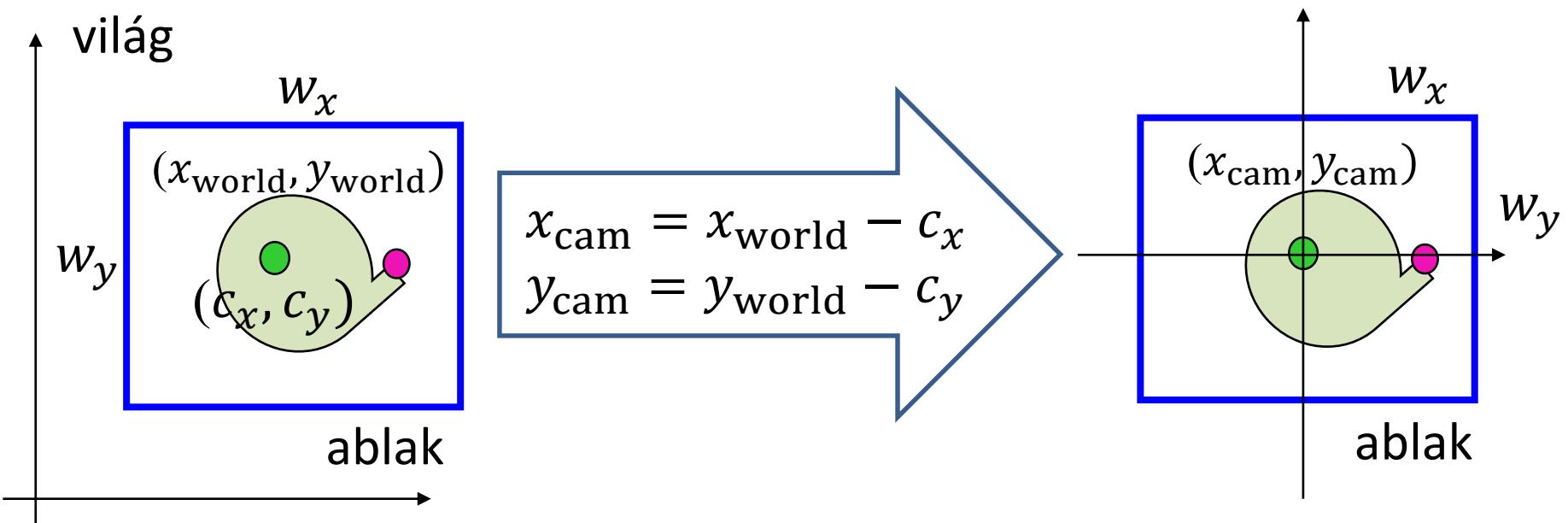
```
inline mat4 TranslateMatrix(vec2 t) {
    return mat4( vec4(1, 0, 0, 0),
                 vec4(0, 1, 0, 0),
                 vec4(0, 0, 1, 0),
                 vec4(t.x, t.y, 0, 1));
}

inline mat4 ScaleMatrix(vec2 s) {
    return mat4( vec4(s.x, 0, 0, 0),
                 vec4(0, s.y, 0, 0),
                 vec4(0, 0, 1, 0),
                 vec4(0, 0, 0, 1));
}

inline mat4 RotationMatrix(float fi) {
    return mat4( vec4( cos(fi), sin(fi), 0, 0),
                 vec4(-sin(fi), cos(fi), 0, 0),
                 vec4(0, 0, 1, 0),
                 vec4(0, 0, 0, 1));
}
```

# View transzformáció: V()

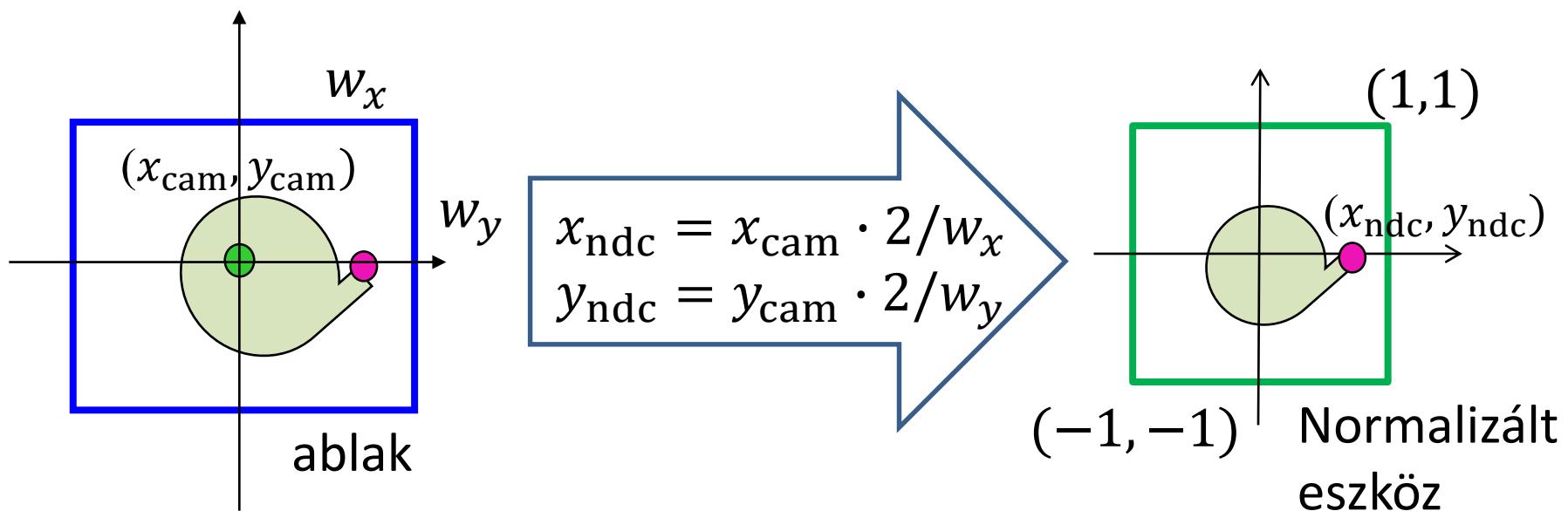
## Kameraablak középe az origóba



$$[x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}}, 1] = [x_{\text{world}}, y_{\text{world}}, z_{\text{world}}, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -c_x & -c_y & 0 & 1 \end{bmatrix}$$

# Projekció: P()

Kameraablak a  $(-1, -1)$ - $(1, 1)$  négyzetbe



$$[x_{\text{ndc}}, y_{\text{ndc}}, z_{\text{ndc}}, 1] = [x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}}, 1] \begin{bmatrix} 2/w_x & 0 & 0 & 0 \\ 0 & 2/w_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

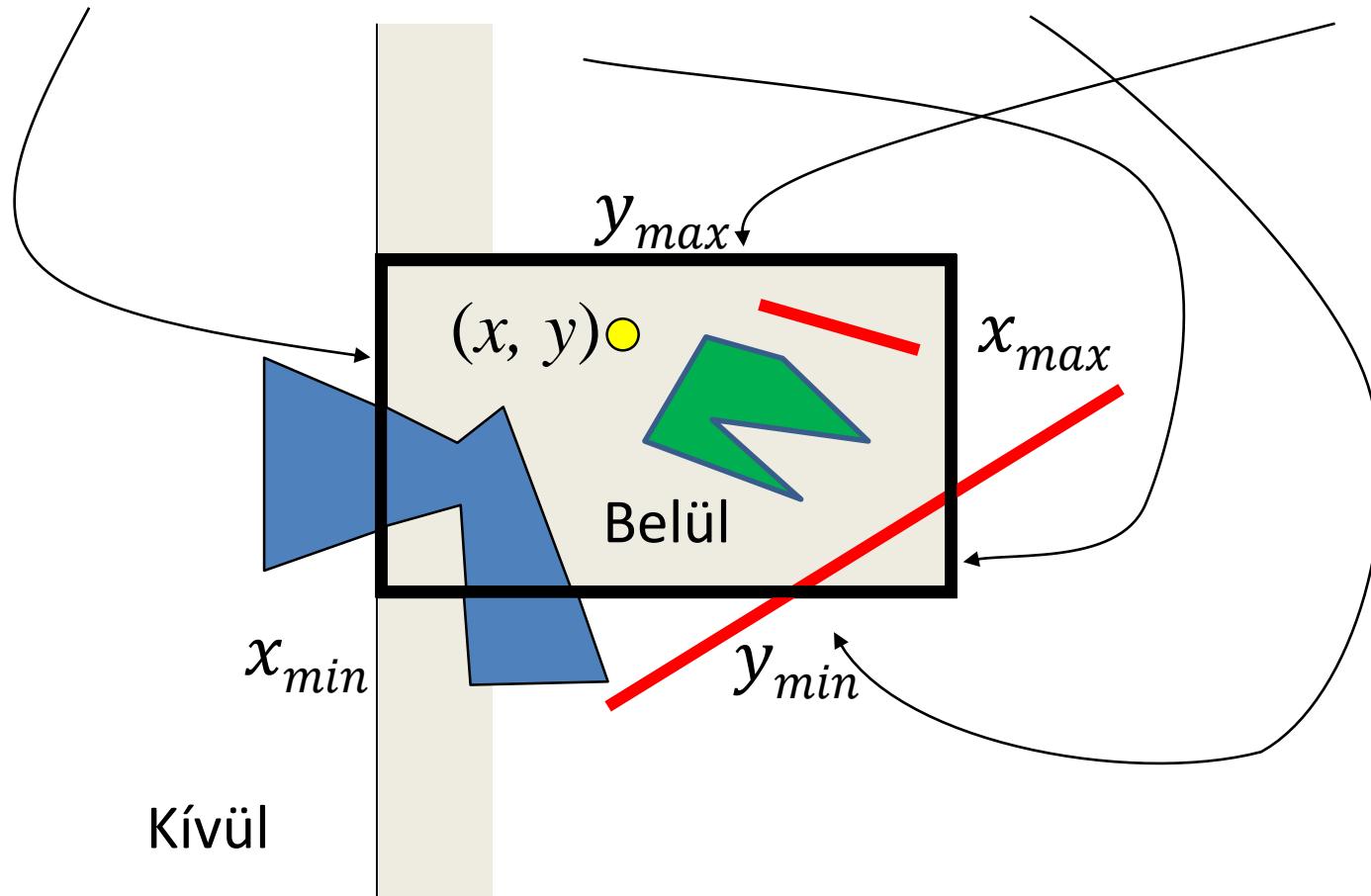
# 2D kamera

```
class Camera2D {  
    vec2 wCenter;// center in world coords  
    vec2 wSize; // width and height in world coords  
  
public:  
    mat4 V() { return TranslateMatrix(-wCenter); }  
  
    mat4 P() { // projection matrix  
        return ScaleMatrix(vec2(2/wSize.x, 2/wSize.y));  
    }  
    mat4 Vinv() { // inverse view matrix  
        return TranslateMatrix(wCenter);  
    }  
    mat4 Pinv() { // inverse projection matrix  
        return ScaleMatrix(vec2(wSize.x/2, wSize.y/2));  
    }  
    void Zoom(float s) { wSize = wSize * s; }  
    void Pan(vec2 t) { wCenter = wCenter + t; }  
};
```

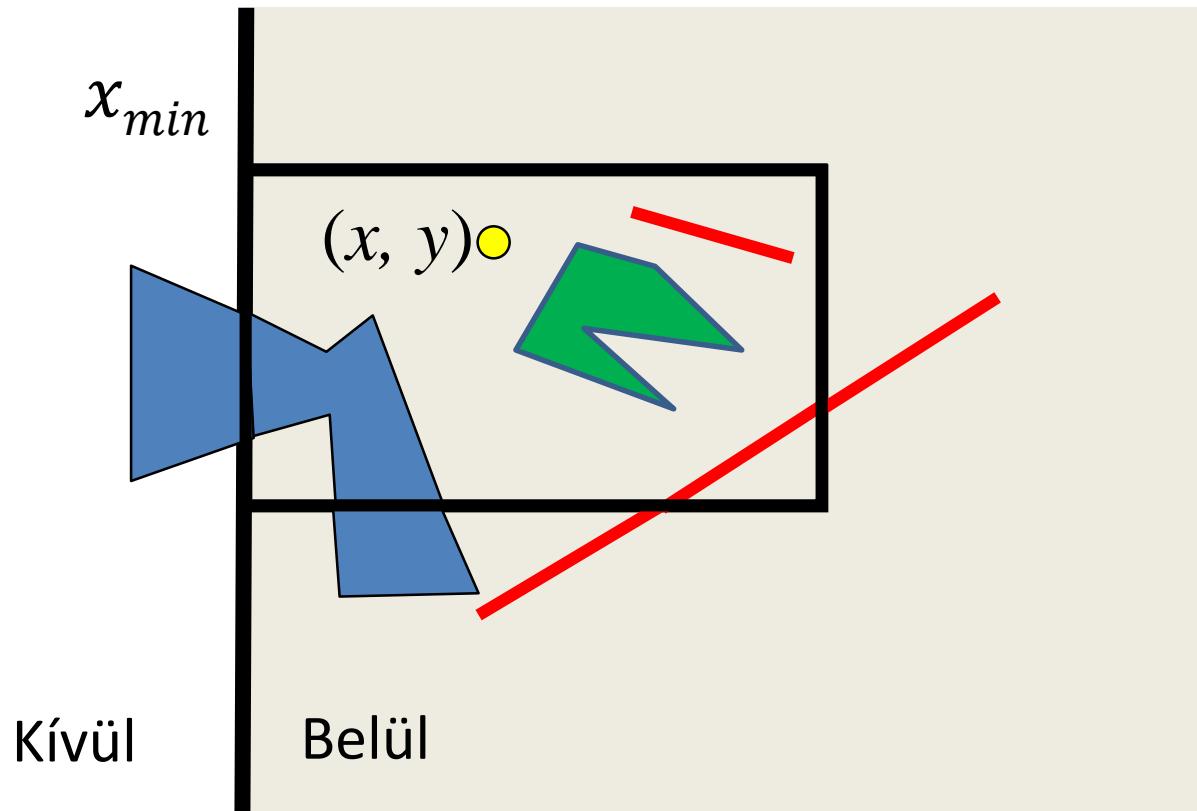
# 2D vágás

Pont vágás:

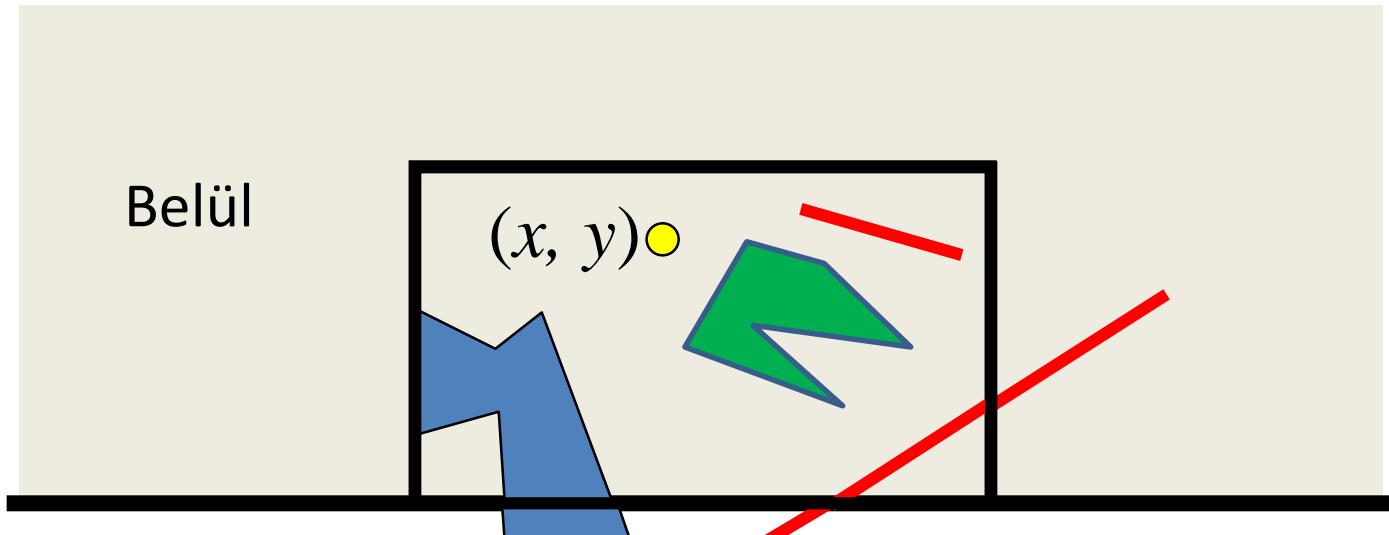
$$x > x_{min} = -1, \quad x < x_{max} = +1, \quad y > y_{min} = -1, \quad y < y_{max} = +1$$



# Vágás

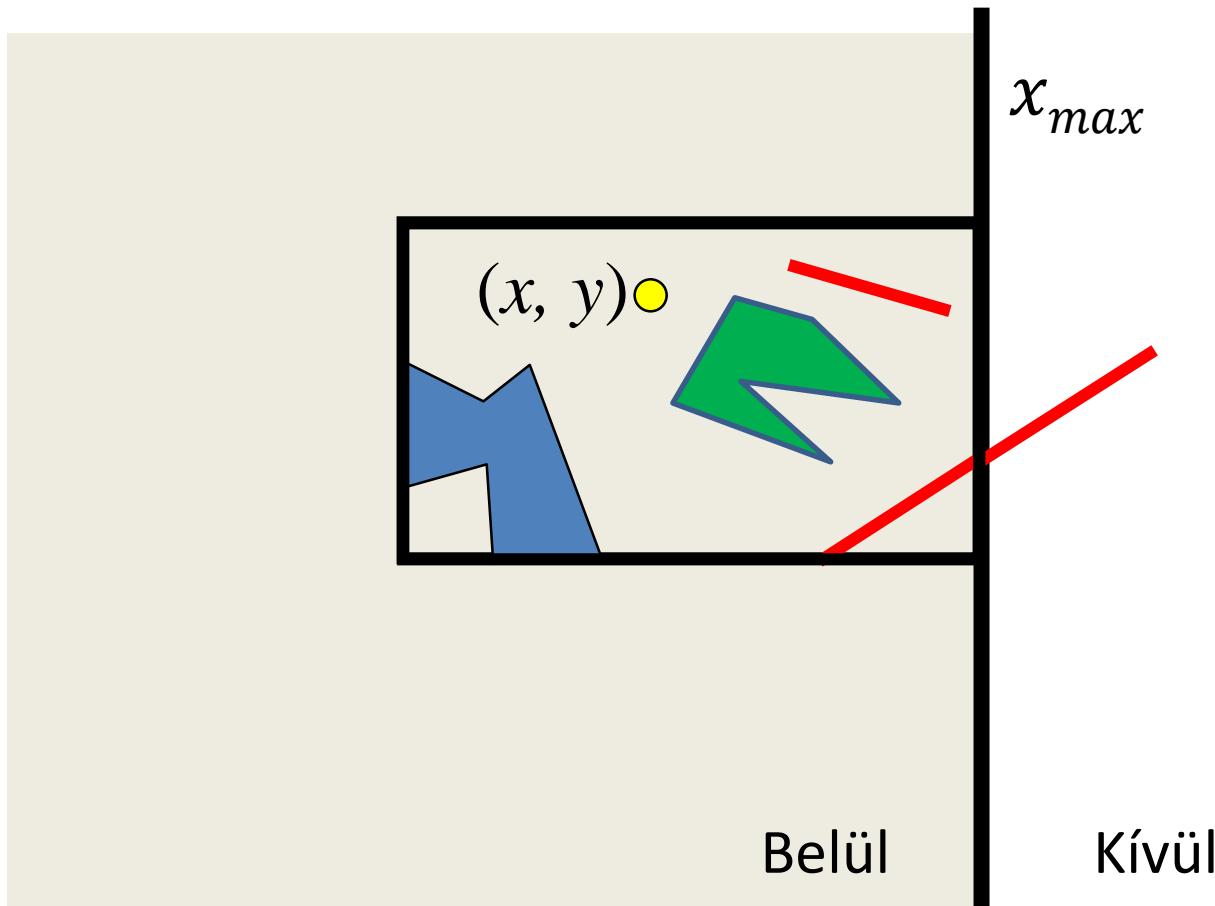


# Vágás



Kívül

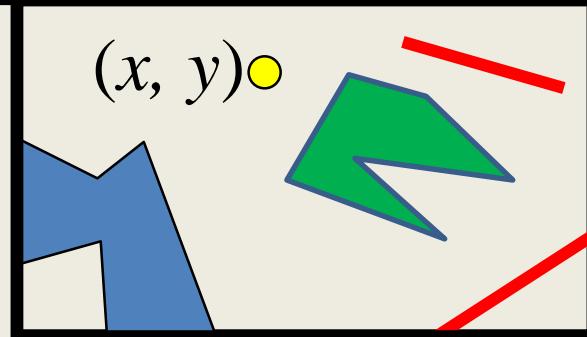
# Vágás



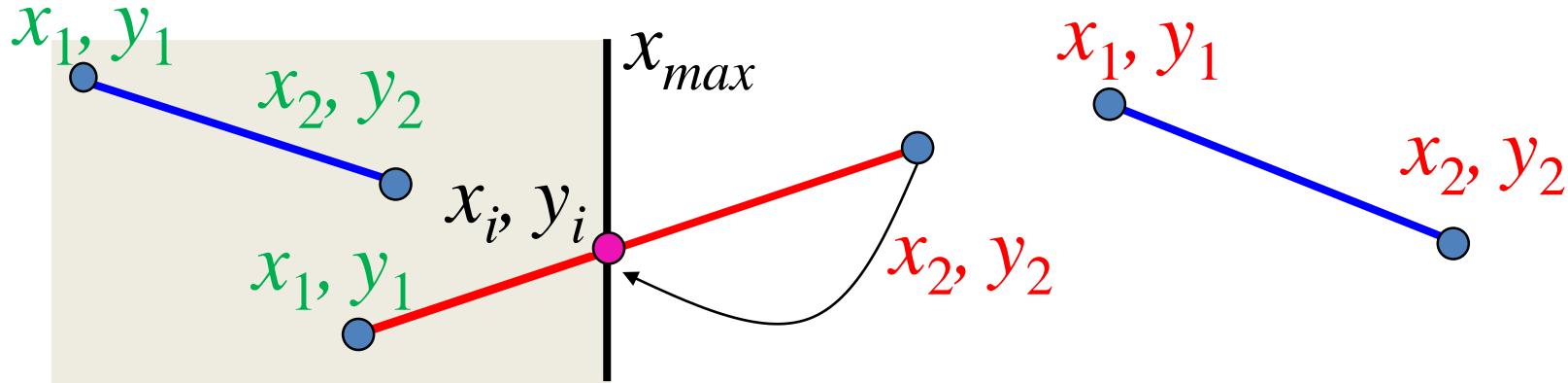
# Vágás

Kívül

Belül



## 2D szakasz vágás: $x < x_{max}$



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

$$x = x_{max}$$

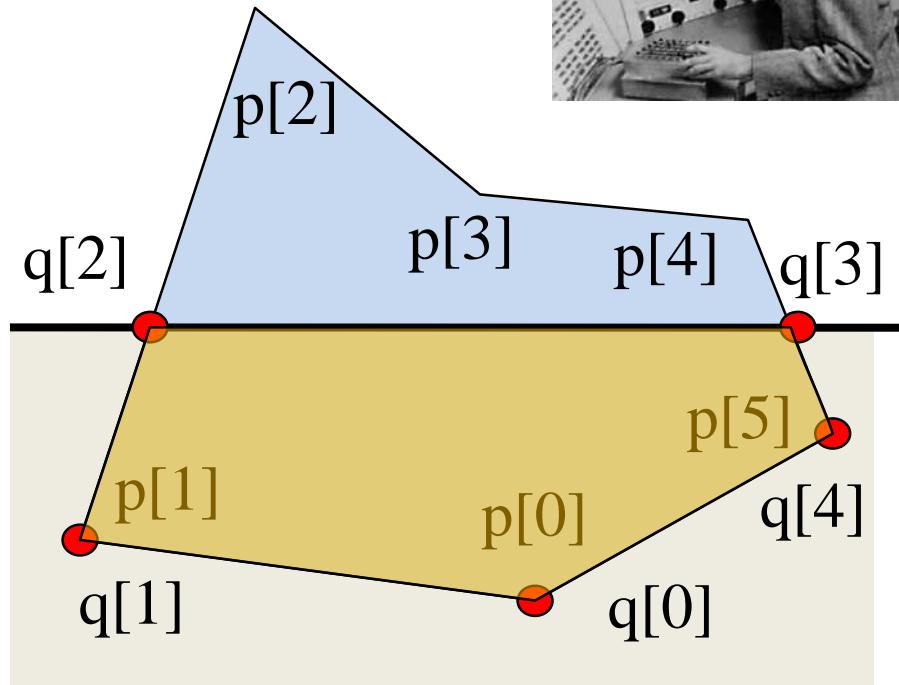
Metszés:  $x_{max} = x_1 + (x_2 - x_1)t \iff t = (x_{max} - x_1) / (x_2 - x_1)$

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

# (Ivan) Sutherland-(Gary) Hodgman 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

# 3D vágás homogén koordinátákban (GPU)

$$x(t) = x_1 + (x_2 - x_1)t$$

$$y(t) = y_1 + (y_2 - y_1)t$$

$$-1 = x_{min} < x < x_{max} = 1$$

$$-1 = y_{min} < y < y_{max} = 1$$

Cél:

$$-1 < x = X/w < 1$$
$$-1 < y = Y/w < 1$$
$$-1 < z = Z/w < 1$$

$$w > 0$$

$$w < 0$$

GPU csak ezt  
csinálja meg

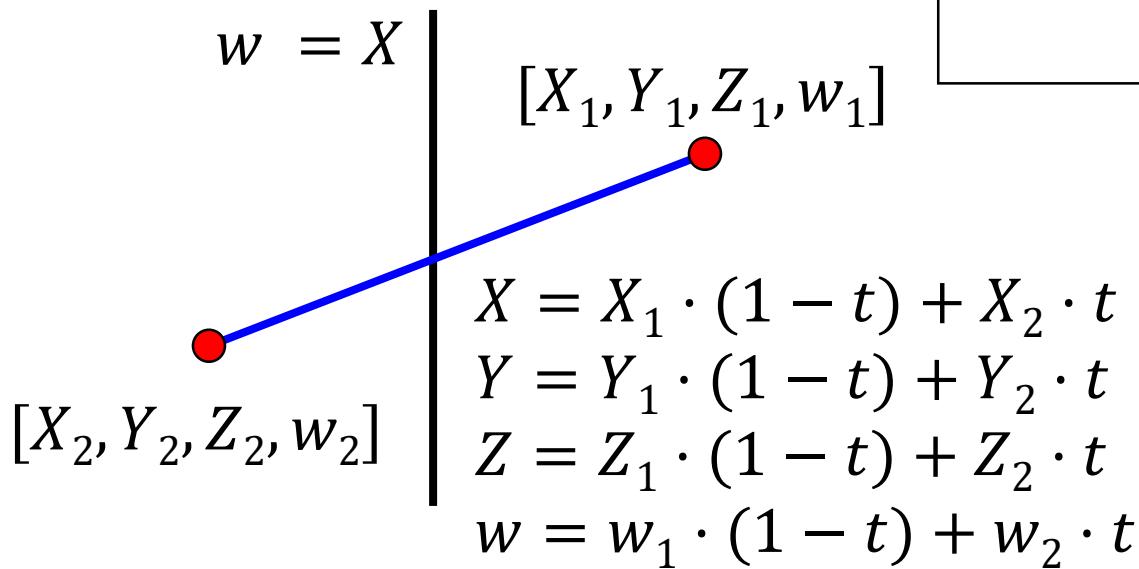
$$\begin{aligned} -w &< X < w \\ -w &< Y < w \\ -w &< Z < w \end{aligned}$$

$$\begin{aligned} -w &> X > w \\ -w &> Y > w \\ -w &> Z > w \end{aligned}$$

# 3D szakasz/poligon vágás homogén koordinátákban (GPU)

$$\begin{aligned} -w &< X < w \\ -w &< Y < w \\ -w &< Z < w \end{aligned}$$

$$\begin{aligned} w &= w_1 \cdot (1 - t) + w_2 \cdot t = \\ &= X = X_1 \cdot (1 - t) + X_2 \cdot t \end{aligned}$$

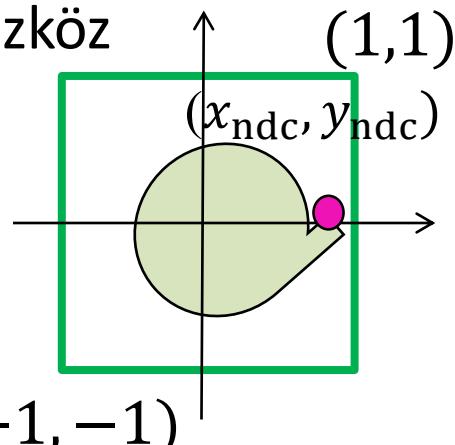


$$t = \dots$$

# Viewport transzformáció: Normalizáltból képernyő koordinátákba (GPU)

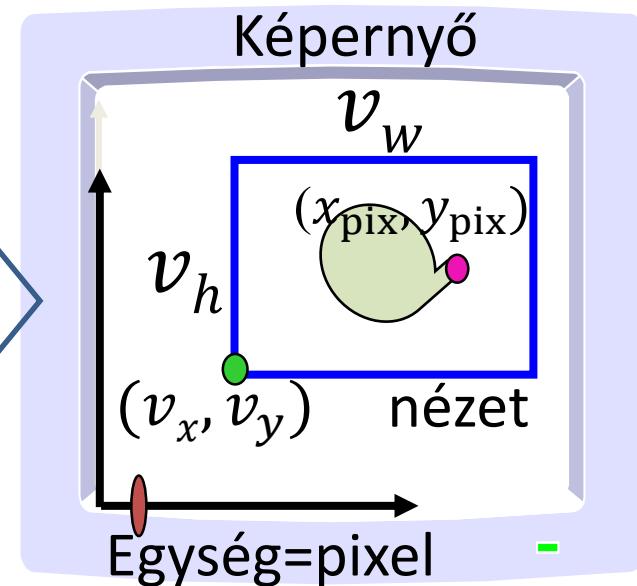
Normalizált

eszköz



$$x_{pix} = v_w(x_{ndc} + 1)/2 + v_x$$
$$y_{pix} = v_h(y_{ndc} + 1)/2 + v_y$$

$$z_{pix} = (z_{ndc} + 1)/2$$



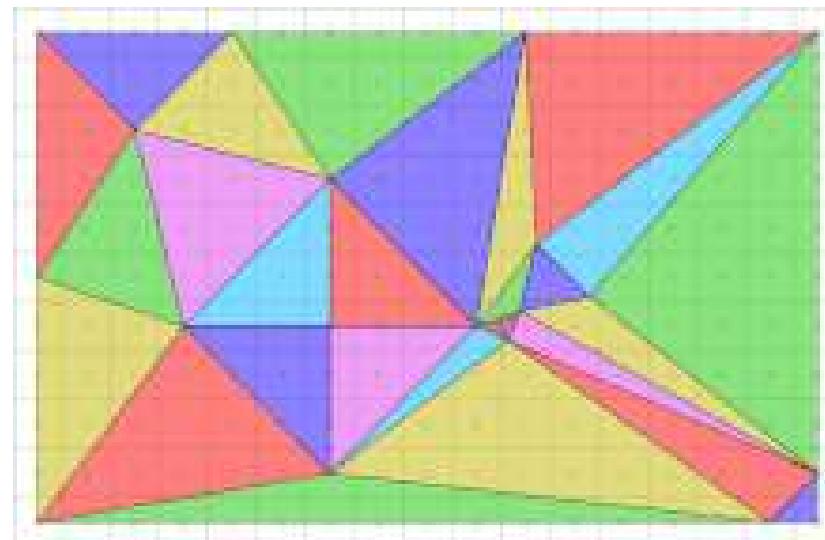
*“The computer was born to solve  
problems that did not exist before.”*

*Bill Gates*

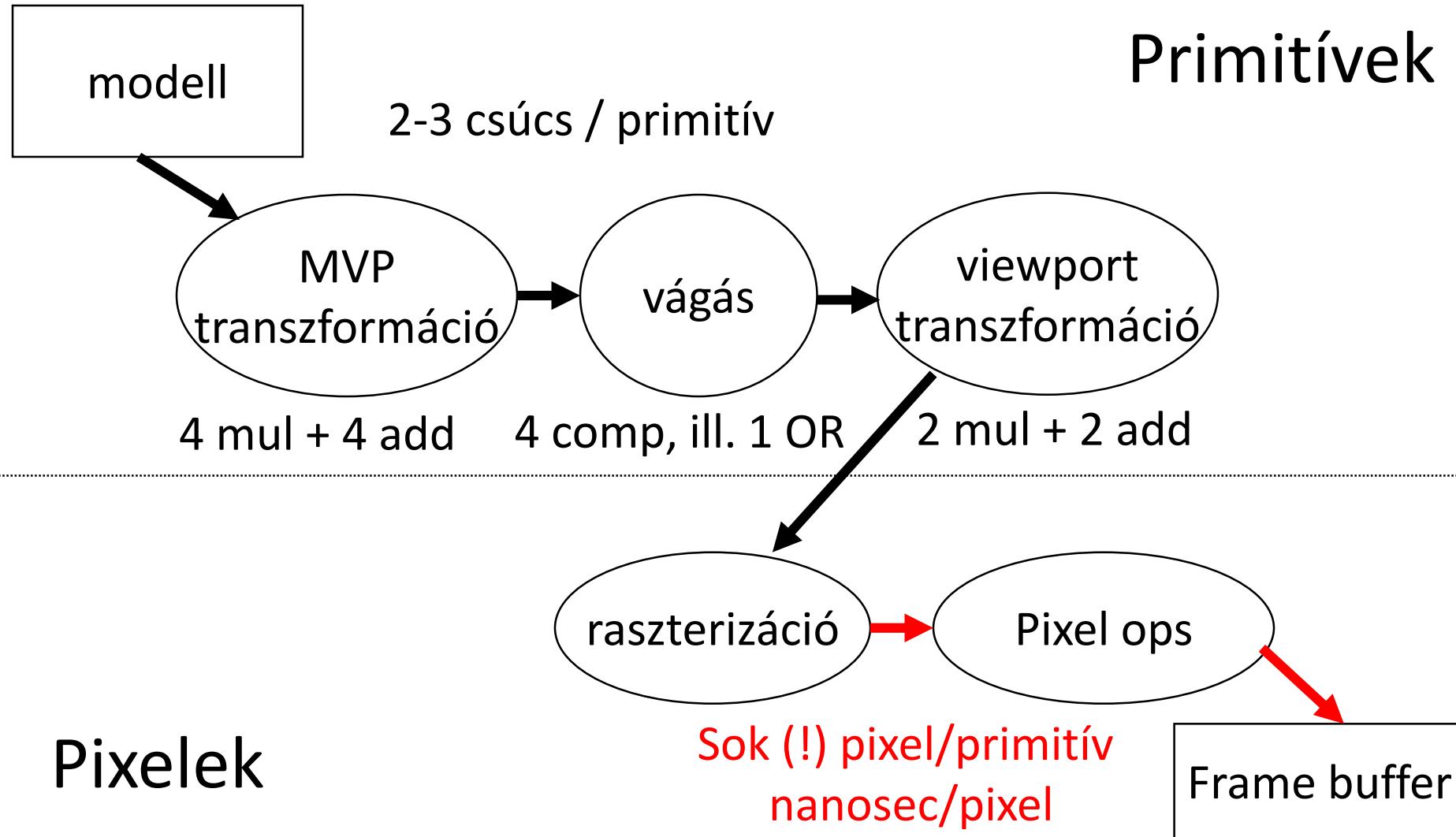
# 2D képszintézis

## 4. Rasterizáció

Szirmay-Kalos László



# Raszterizáció (GPU)

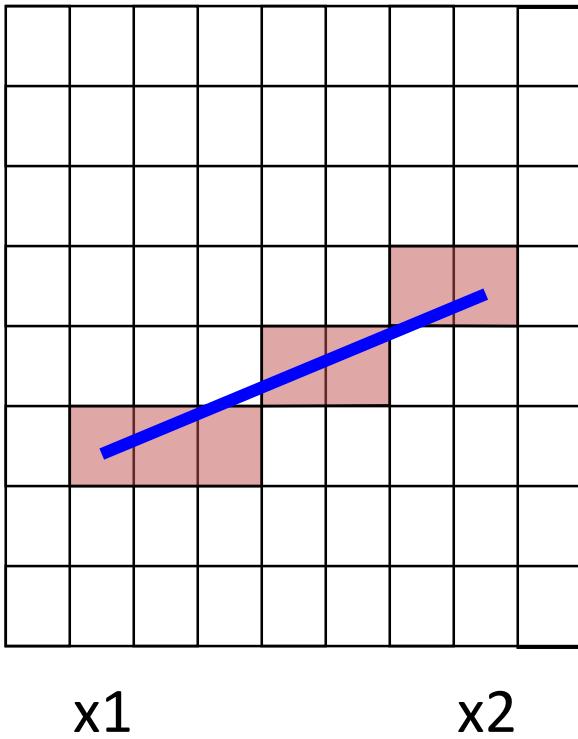


# Pont rajzolás

A pont „**kicsi**” és van jellemző helye:

- A kiszínezett tartomány is legyen kicsi
- A legkisebb dolog, amit át lehet színezni a pixel
- Színezzünk ki egy (vagy néhány) pixelt, amely legközelebb van a ponthoz
- Pixelkoordináták egészek
- **Legközelebbi pixel** = koordináták kerekítése

# Szakasz rajzolás



Egyenes „**vékony**” és **összefüggő**.  
Pontjai kielégítik az egyenletét:

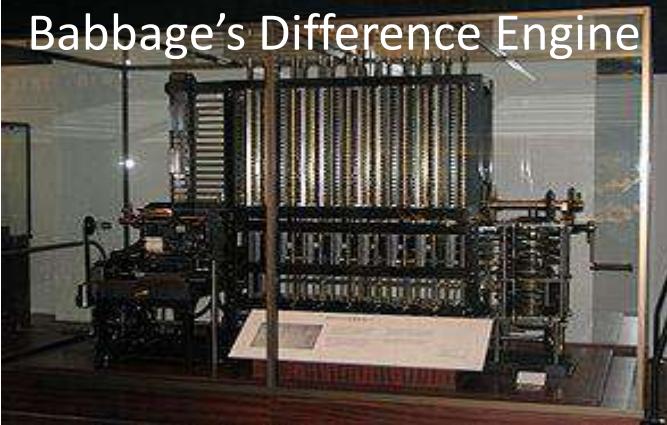
$$y = mx + b$$

$x_2 > x_1$ ,  $|x_2-x_1| \geq |y_2-y_1|$  típusú  
szakasz rajzolása:

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

# Inkrementális elv és fixpontos program

$$y(x) = mx + b = y(x - 1) + m$$

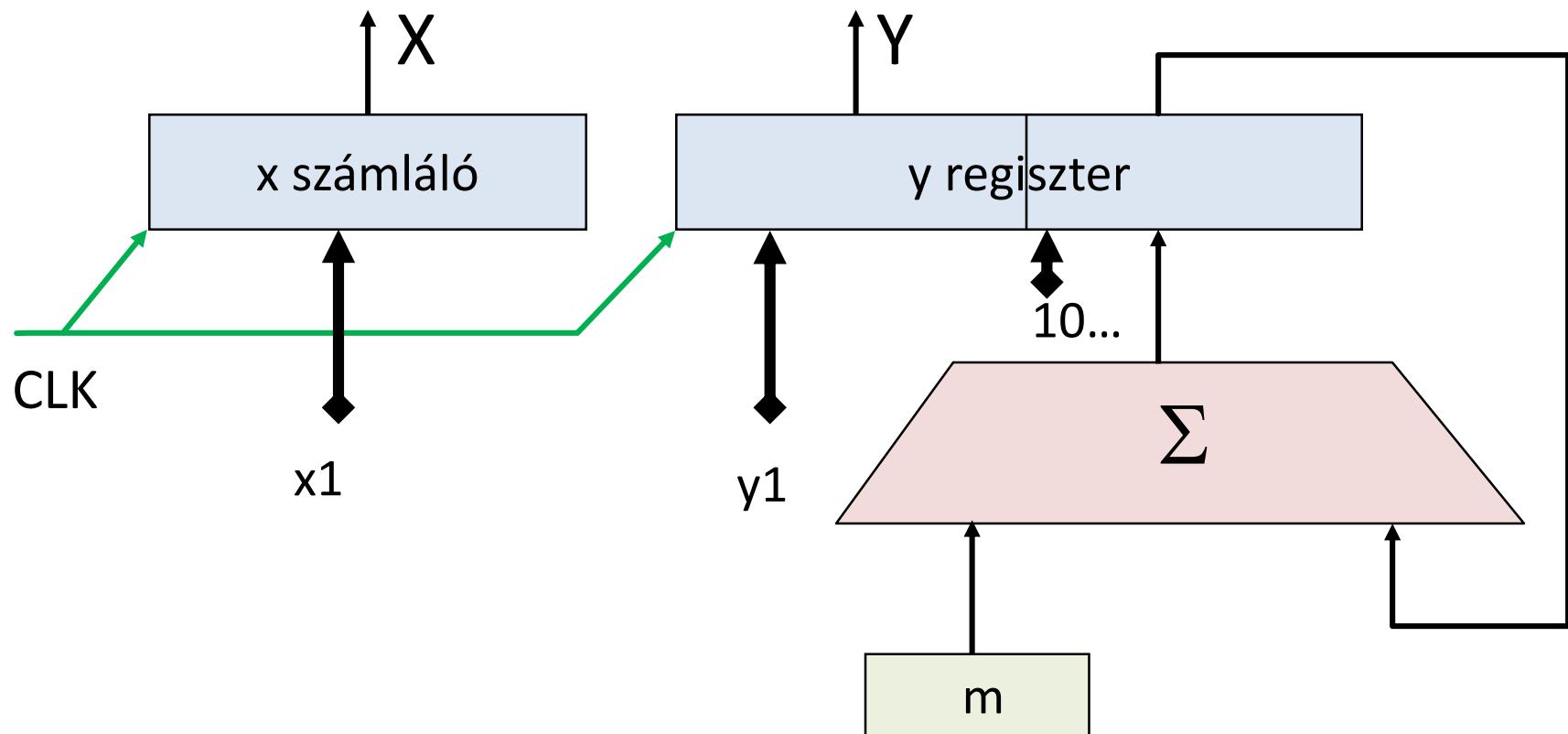


```
LineFloat (short x1, short y1,  
          short x2, short y2) {  
  
    float m = (float)(y2-y1)/(x2-x1);  
    float y = y1;  
    for(short x = x1; x <= x2; x++) {  
        short Y = round(y);  
        write(x, Y, color);  
        y = y+m;  
    }  
}
```

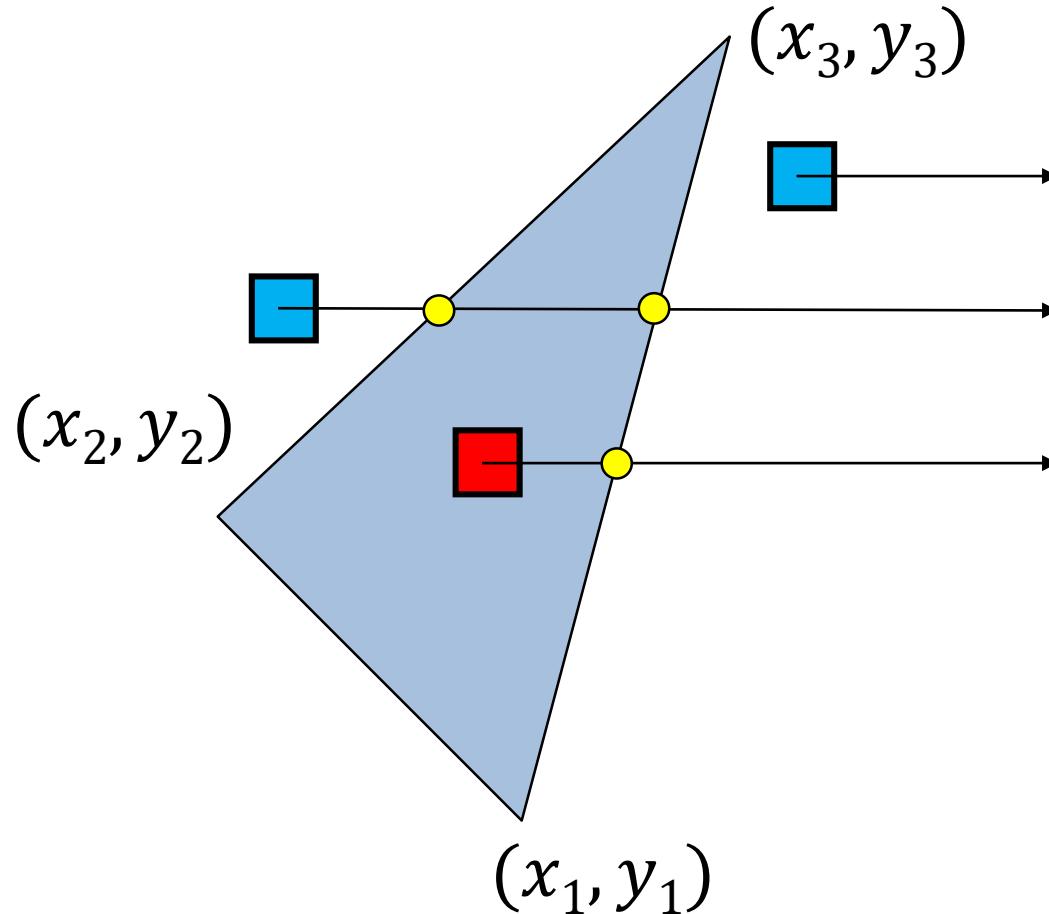
```
const int T=12; // fractional bits
```

```
LineFix (short x1, short y1,  
          short x2, short y2) {  
  
    int m = ((y2 - y1)<<T)/(x2 - x1);  
    int y = (y1<<T)+(1<<(T-1)); // +0.5  
    for(short x = x1; x <= x2; x++) {  
        short Y = y >> T; // trunc  
        write(x, Y, color);  
        y = y+m;  
    }  
}
```

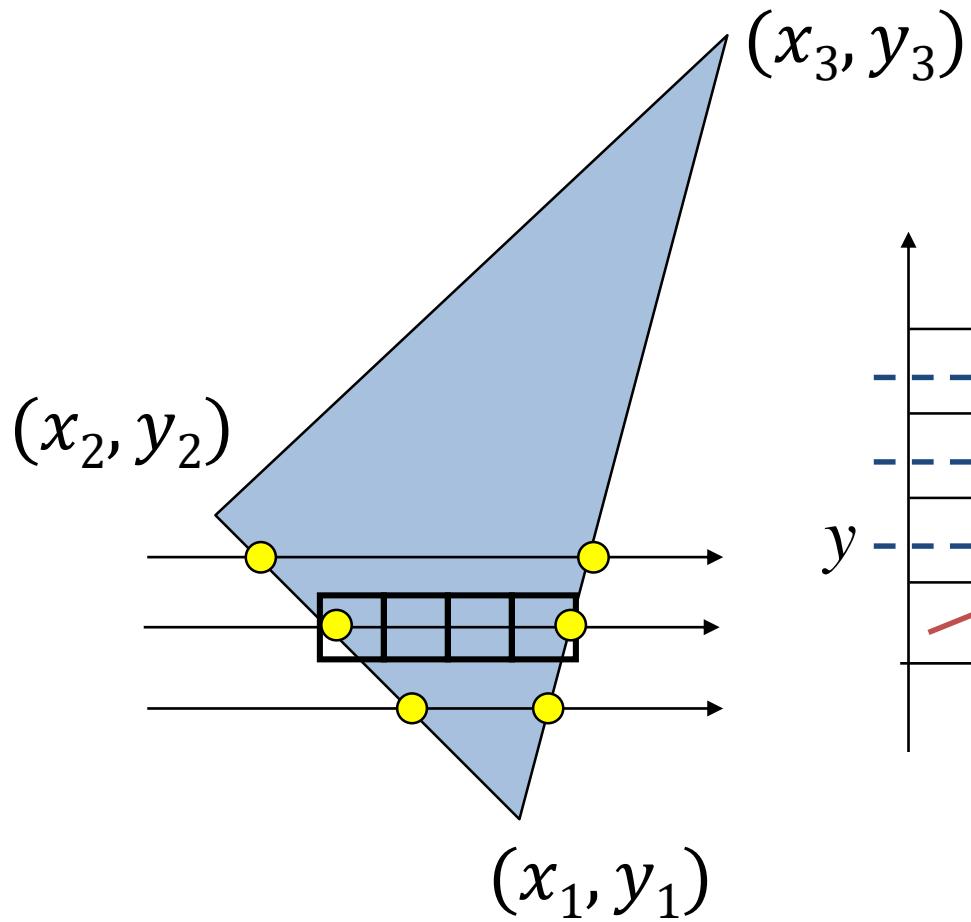
# DDA szakaszrajzoló hardver



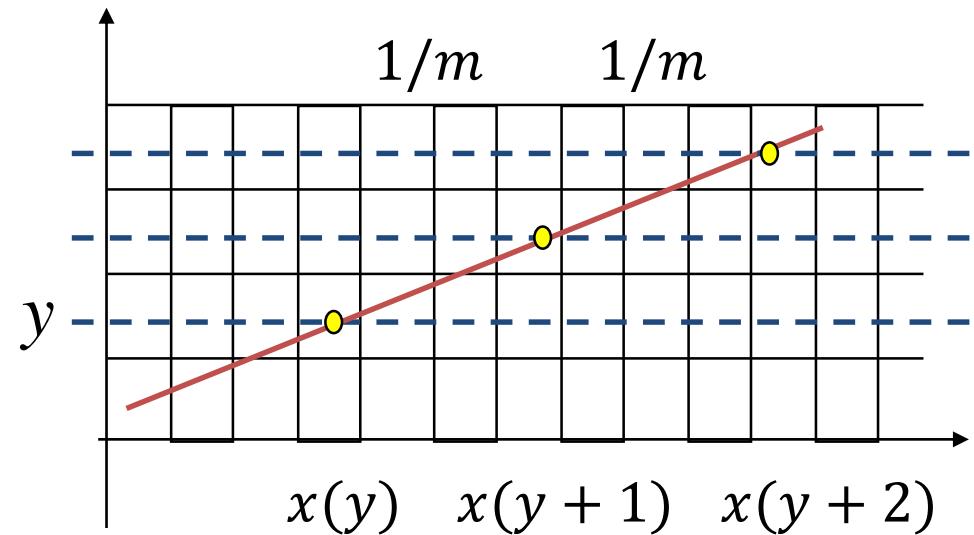
# Naív háromszög kitöltés



# Háromszög kitöltés



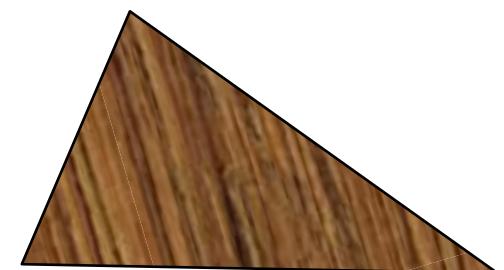
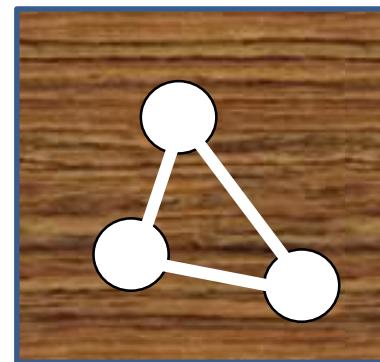
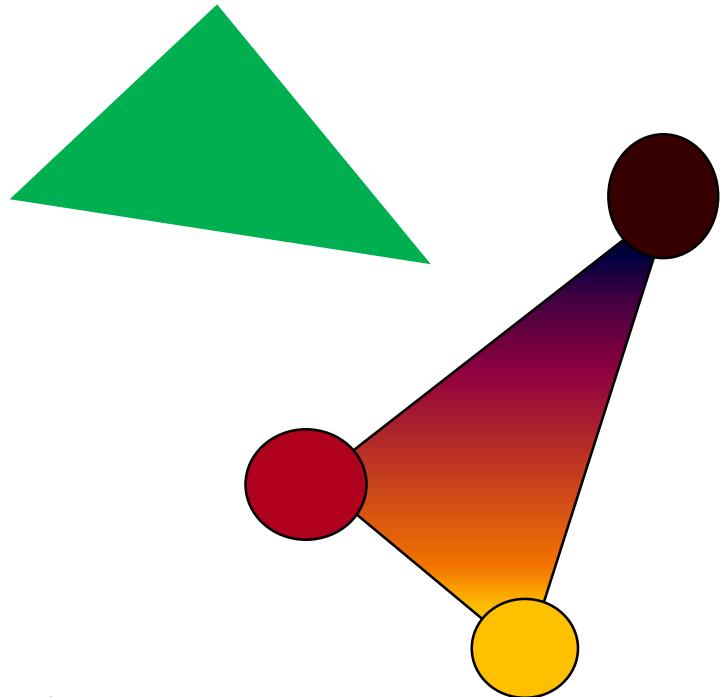
$$y = mx + b$$



$$x(y + 1) = x(y) + 1/m$$

# Milyen színű legyen a pixel?

- Uniform az egész objektum
- Csúcspont tulajdonságokból interpolált szín
- Pl. Textúrázás (2D):  
Csúcspont textúrakoordináták interpolációja a pixelekre, majd textúra olvasás.



# Ellenőrző kérdések

- Bizonyítsa be, hogy bármely 4+ csúcsú sokszögnek van diagonálja!
- Bizonyítsa be a kétfül tételeit!
- Van értelme egy kört raszterizáló algoritmusnak?
- Írjon sokszögkitöltő algoritmust, amely nem egyszerű (határ önmagát metszi és több határ is van) sokszögeket is ki tud tölteni.
- Implementálja a vágás és raszterizálás algoritmusait!
- Írjon programot, amely eldönti, hogy egy koordinátatengelyekkel párhuzamos téglalap tartalmaz-e egy szakaszból vagy egy sokszögből valamennyit?
- Adja meg egy 2D szerkesztő (pl. egyszerűsített Powerpoint) osztálydiagramját.
- Mi az értelme a normalizált eszköz-koordinátarendszer bevezetésének?

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

*In practice, there is."*

*Benjamin Brewster*

# Grafikus hardver/szoftver alapok

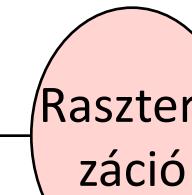
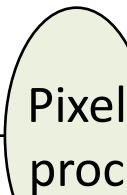
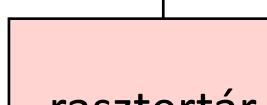
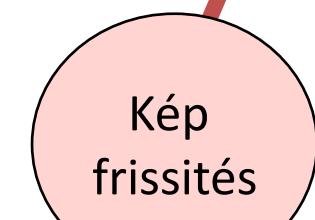
## 1. Építőelemek

Szirmay-Kalos László



# Interaktív rendszer: Funktionális modell

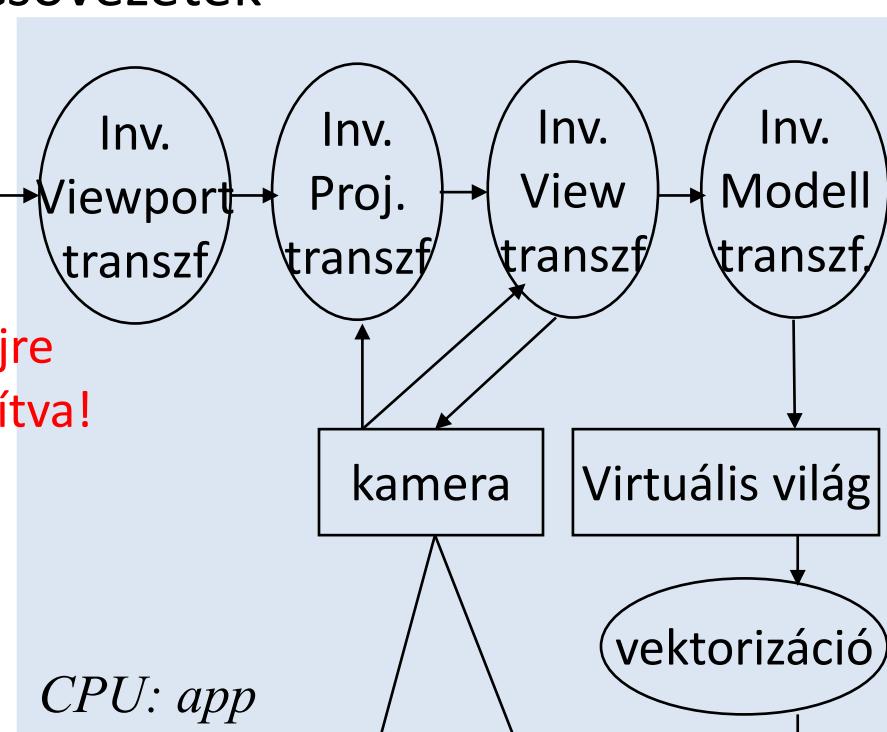
Éppen belemerül  
a virtuális világba



*fix hw*



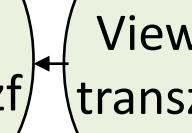
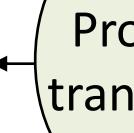
*CPU: app*



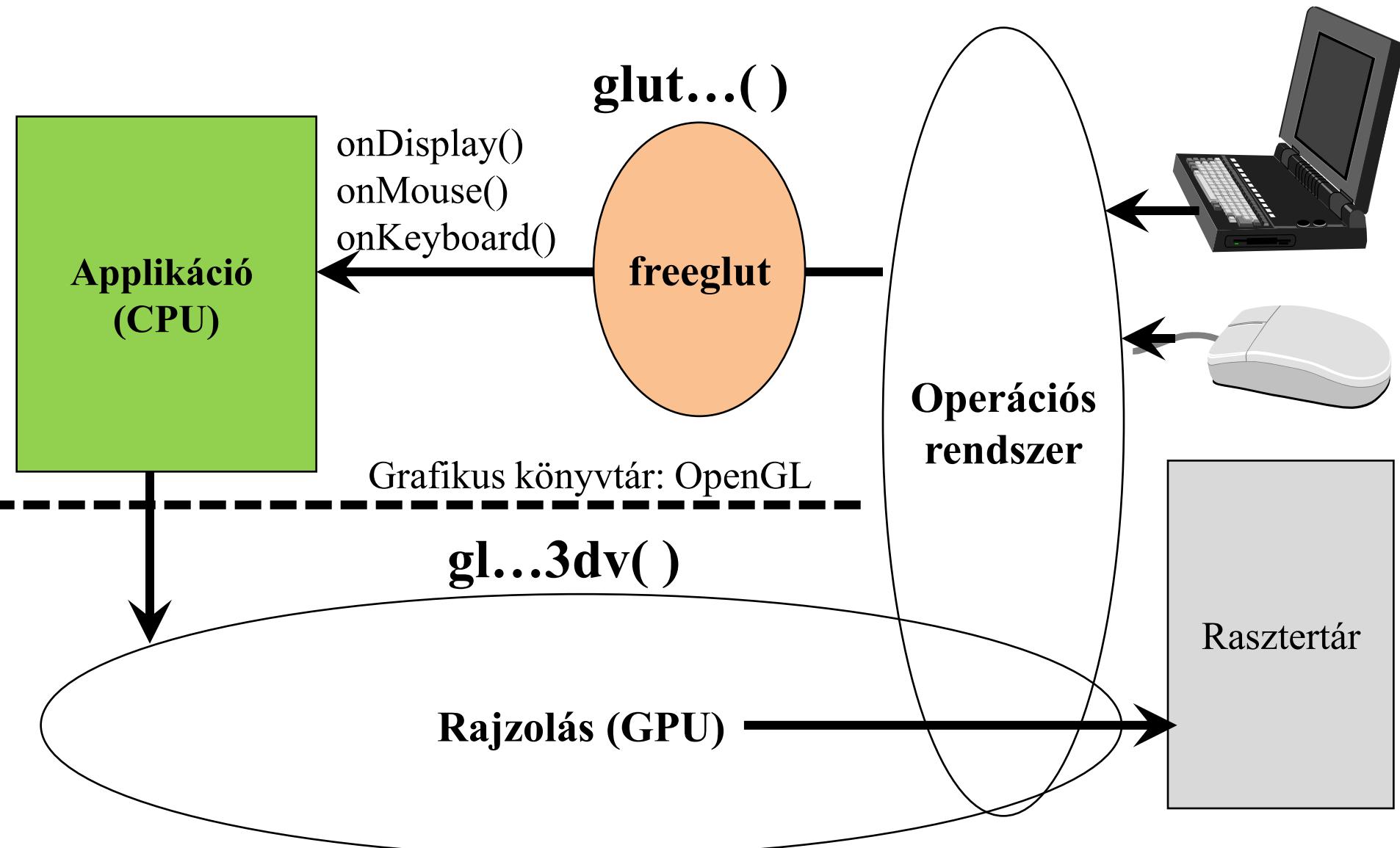
*pixel shader+blending*

Kimeneti csővezeték: GPU

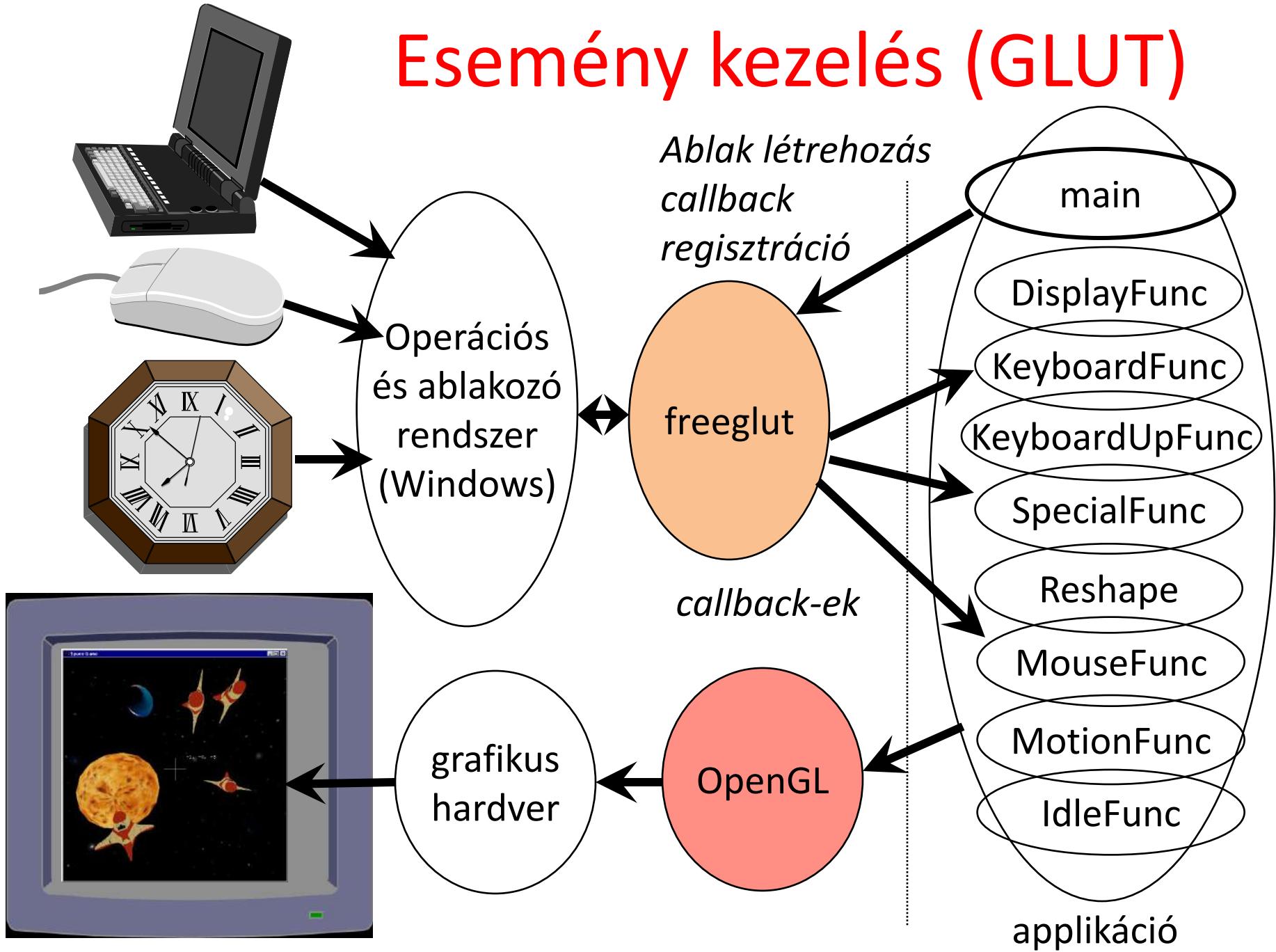
*vertex shader*



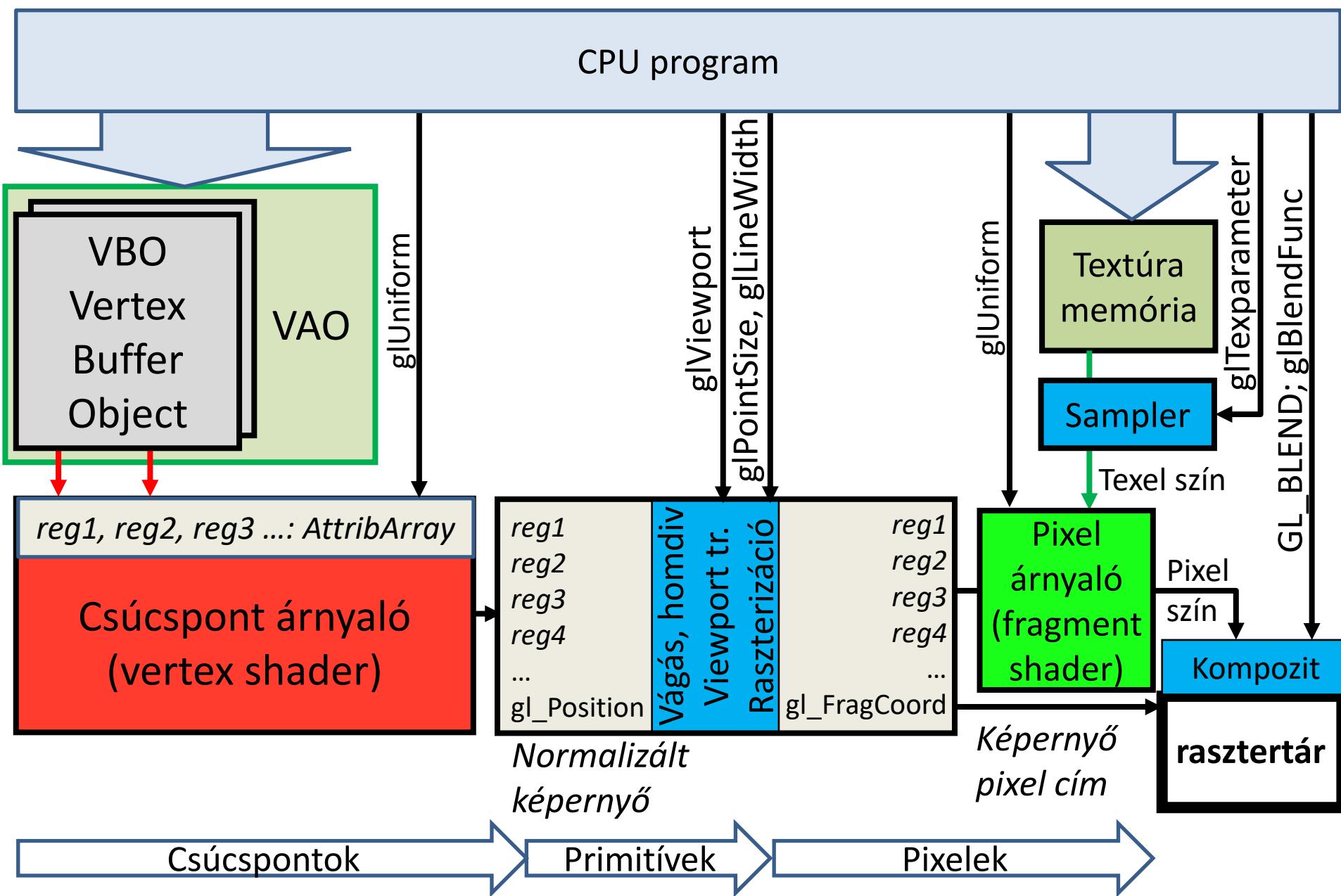
# Szoftver architektúra



# Esemény kezelés (GLUT)

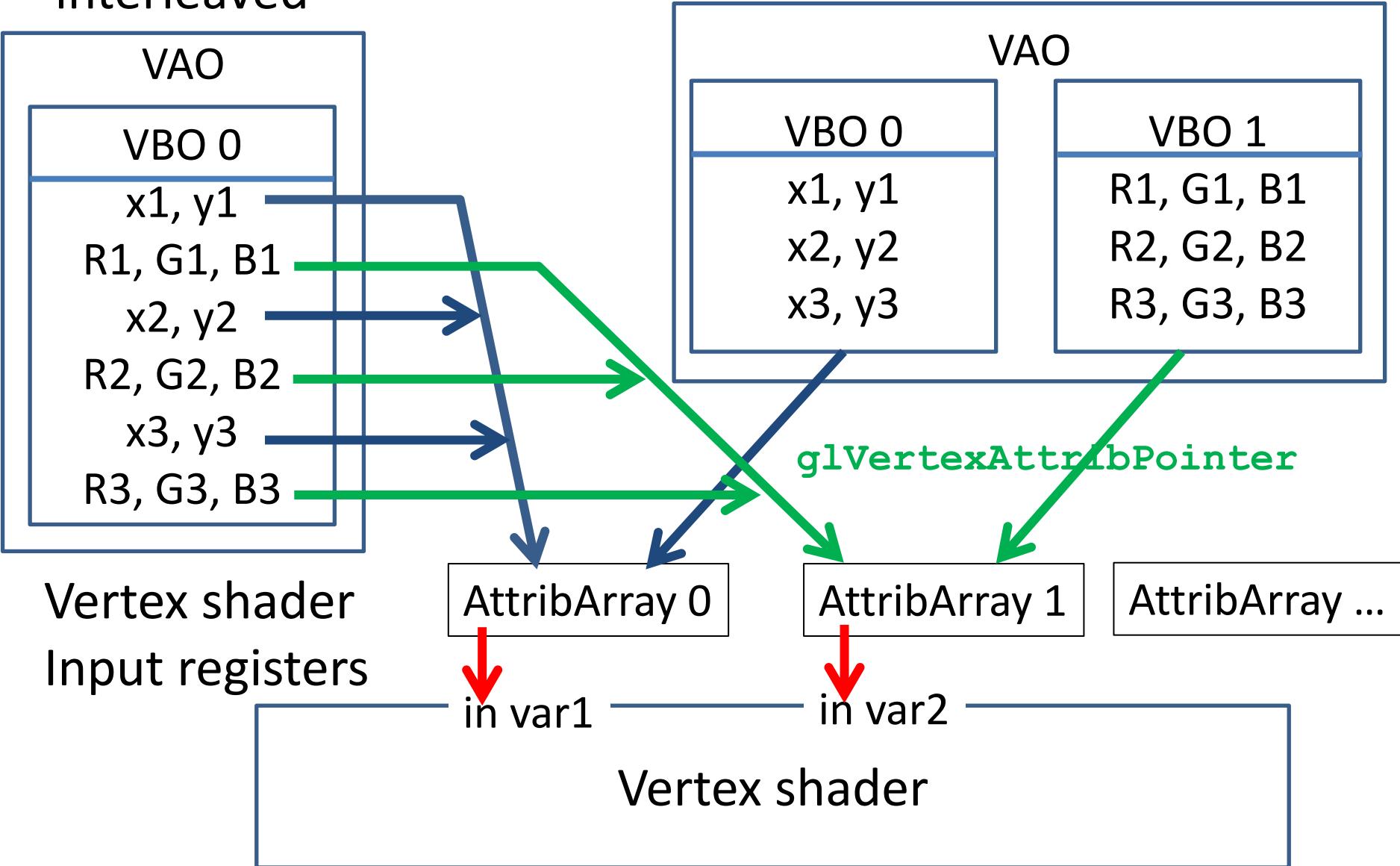


# OpenGL 3.3 ... 4.6 (Modern OpenGL)



# Csúcspont adatfolyamok

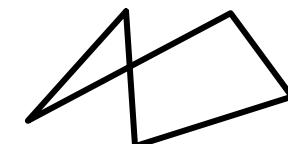
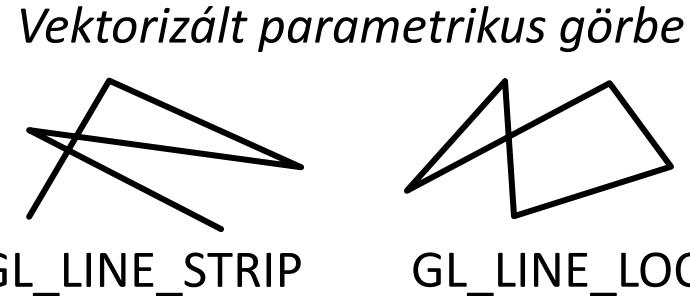
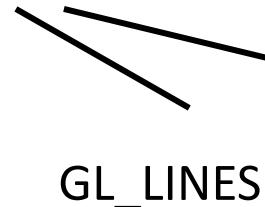
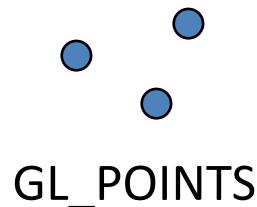
interleaved



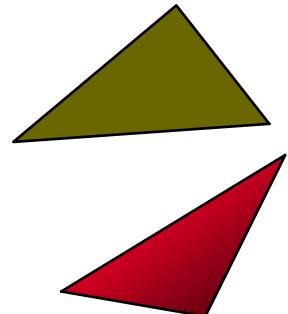
# Rajzolás: `glDrawArrays`

## OpenGL primitívek

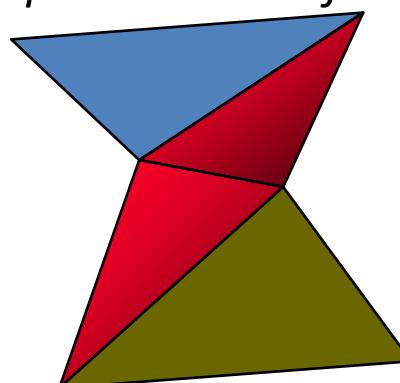
```
glBindVertexArray(vao);  
glDrawArrays(primitiveType, startIdx, numOfElements);
```



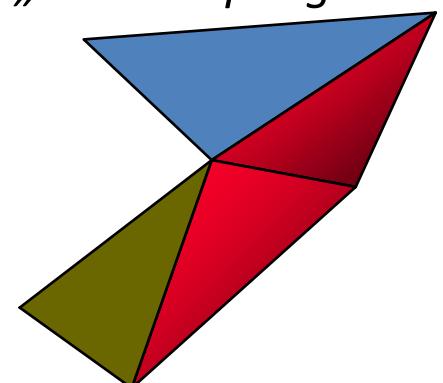
Fülvágó kimenete



Tesszellált 3D  
parametrikus felület



„Konvex” poligon



# OpenGL állapotgép

## Gagyi grafikus könyvtár

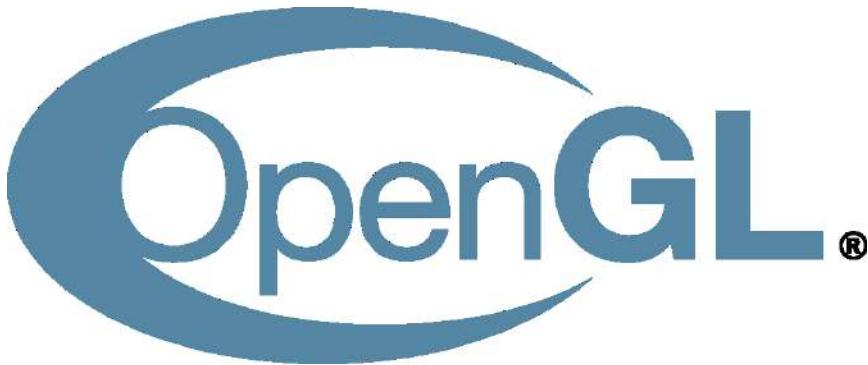
```
fillOval(x1,y1,x2,y2, texture, color, width,...);
```

## OpenGL

```
glPointSize(3);
glLineWidth(5);
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBindTexture(GL_TEXTURE_2D, textureId);

glBufferData(GL_ARRAY_BUFFER, 10, v, GL_STATIC_DRAW);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, ...);

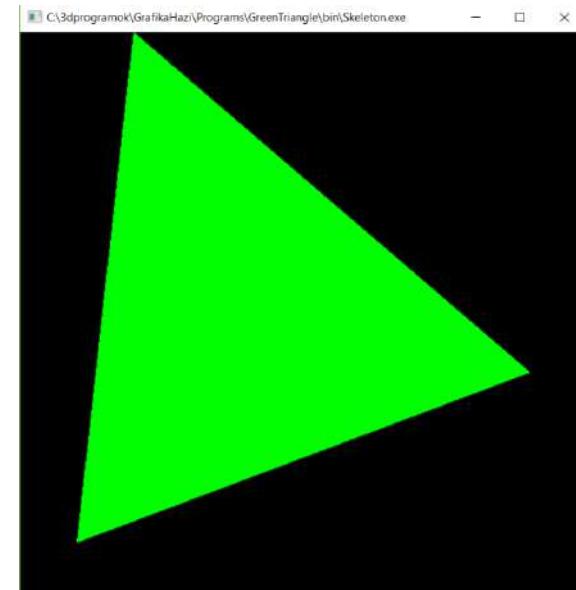
glDrawArrays(GL_TRIANGLES, 0, 3); // Mind! !!
```



# Grafikus hardver/szoftver alapok

## 2. Helló OpenGL/GLSL/GLUT

Szirmay-Kalos László



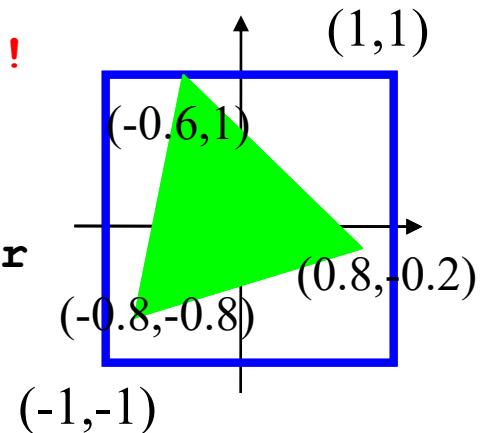
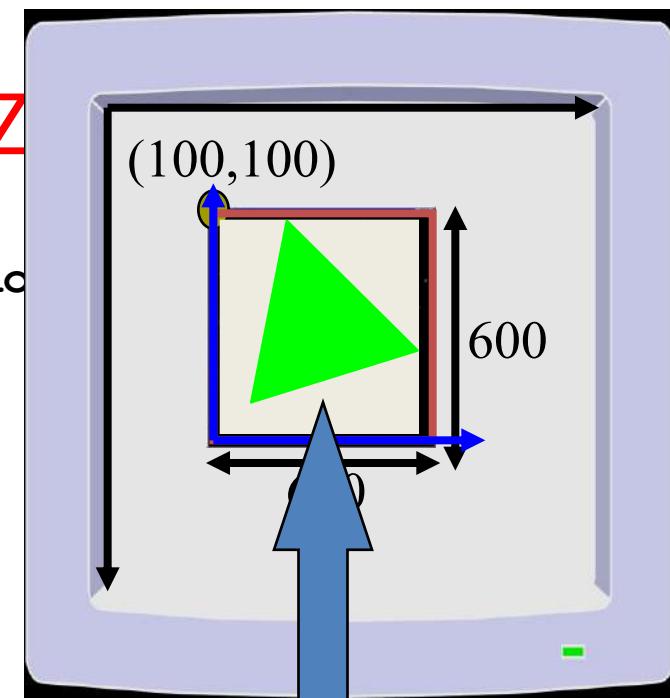
# Az első OpenGL programom: Z

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

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

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

    glutDisplayFunc(onDisplay); //event handler
    glutMainLoop(); // message loop
    return 1;
}
```

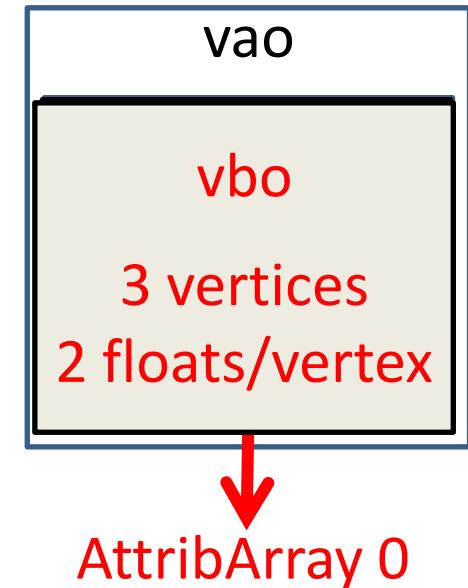


# onInitialization()

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

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

    unsigned int vbo;// vertex buffer object
    glGenBuffers(1, &vbo); // Generate 1 buffer
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    // Geometry with 24 bytes (6 floats or 3 x 2 coordinates)
    float vertices[] = {-0.8,-0.8, -0.6,1.0, 0.8,-0.2};
    glBufferData(GL_ARRAY_BUFFER, // Copy to GPU target
                 sizeof(vertices), // # bytes
                 vertices,           // address
                 GL_STATIC_DRAW); // we do not change later
    glEnableVertexAttribArray(0); // AttribArray 0
    glVertexAttribPointer(0, // vbo -> AttribArray 0
                          2, GL_FLOAT, GL_FALSE, // two floats/attrib, not fixed-point
                          0, NULL);             // stride, offset: tightly packed
```



```
#version 330
precision highp float;

uniform mat4 MVP;
layout(location = 0) in vec2 vp;
void main() {
    gl_Position = vec4(vp.x,vp.y,0,1) * MVP;
}

static const char * vertSource = R"( ... )";
// vagy:
// static const char * vertSource = FileToString("vertex.gls1");

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

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

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

glBindFragDataLocation(shaderProgram, 0, "outColor");

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

C++11

```

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

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

```

```

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

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

```

```

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

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

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

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

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

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

```

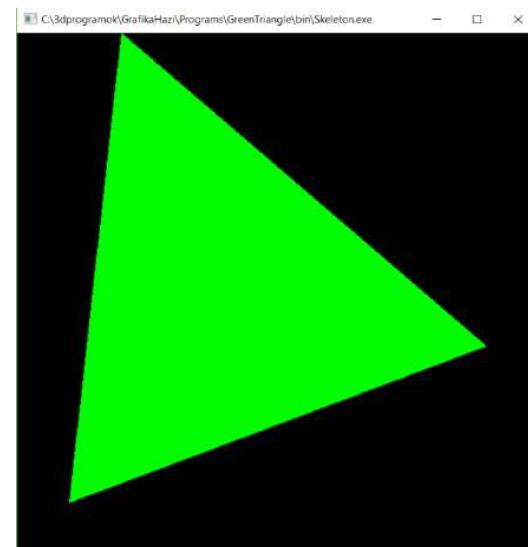
row-major



# Grafikus hardver/szoftver alapok

## Program: Keret és zöld háromszög

Szirmay-Kalos László



# OpenGL starters' kit: Shader programs

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

```
class GPUProgram {  
    ...  
    bool create(char* vertShader, char * fragShader,  
               char * OutputName, char * geomShader = 0);  
    void Use();  
    void setUniform(...);  
};
```

# OpenGL starters' kit

## Erőforrás létrehozás, feltöltés és aktivizálás

- `glGen[VertexArrays|Buffers|Textures]()` ;
- `glBind[VertexArray|Buffer|Texture]()` ;
- `glBufferData()` ;

## Bufferek vertex shader bemeneti regiszterekhez kötése

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

## Rajzolás és csővezeték management

- `glDrawArrays()` vao bufferek felrajzolása
- `glClearColor()` háttér törlési szín
- `glClear()` háttér törlés
- `glViewport()` fénykép méret
- `glPointSize()` pont méret
- `glLineWidth()` vonal vastagság

# framework.h

```
include: <stdio.h>, <stdlib.h>, <math.h>, <vector>, <string>
        if windows <windows.h>
        <GL/glew.h>, <GL/freeglut.h> // must be downloaded

const unsigned int windowWidth = 600, windowHeight = 600;

struct vec2;
struct vec3;
struct vec4;
struct mat4;

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

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

# framework.cpp

```
#include "framework.h"

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

int main(int argc, char * argv[]) {
    glutInit(&argc, argv); glutInitContextVersion(3, 3);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutCreateWindow(argv[0]);

    glewExperimental = true; glewInit();

    onInitialization();
    glutDisplayFunc(onDisplay); // Register event handlers
    glutMouseFunc(onMouse);
    glutIdleFunc(onIdle);
    glutKeyboardFunc(onKeyboard);
    glutKeyboardUpFunc(onKeyboardUp);
    glutMotionFunc(onMouseMotion);
    glutMainLoop(); return 1;
}
```

# Skeleton.cpp

```
#include "framework.h"

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

GPUProgram gpuProgram; // vertex and fragment shaders

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

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

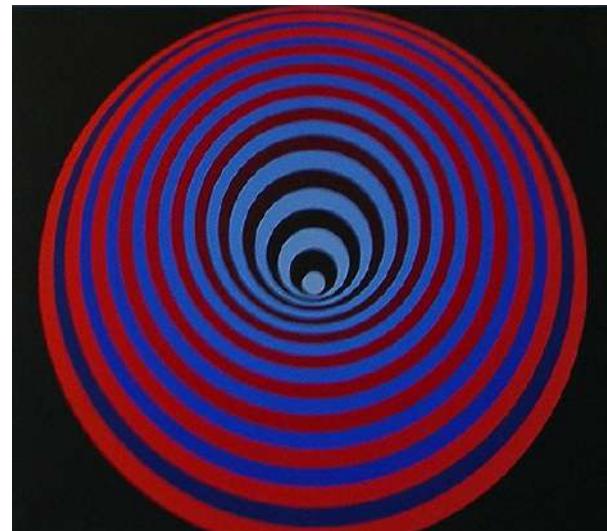
void onKeyboard(unsigned char key, int pX, int pY) { ... }
void onKeyboardUp(unsigned char key, int pX, int pY) { ... }
void onMouseMotion(int pX, int pY) { ... }
void onMouse(int button, int state, int pX, int pY) { ... }
void onIdle() { ... }
```



# Grafikus hardver/szoftver alapok

## Program: Vasarely festmény

Szirmay-Kalos László

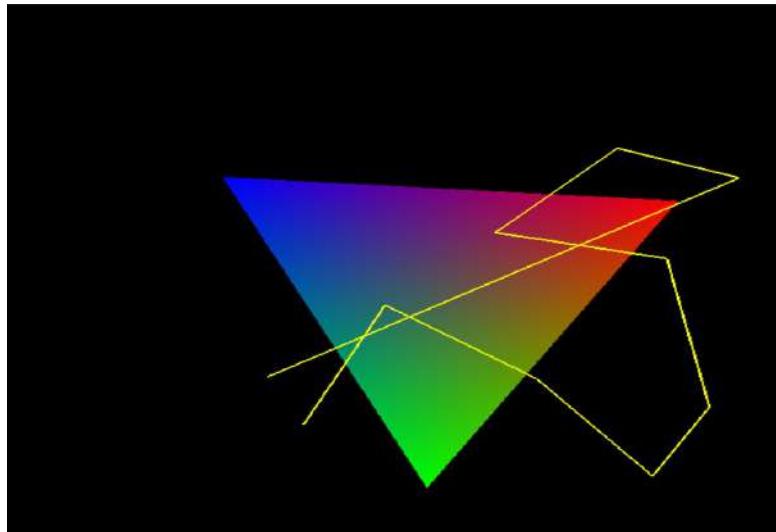




# Grafikus hardver/szoftver alapok

## Program: Animáció és interakció

Szirmay-Kalos László

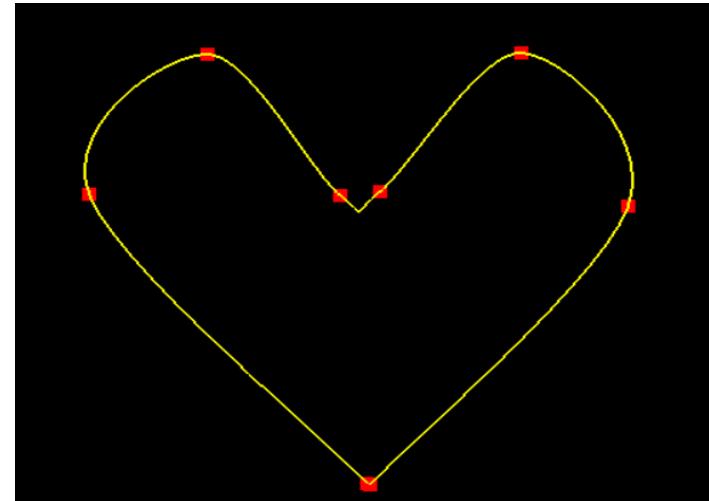




# Grafikus hardver/szoftver alapok

## Program: Görbeszerkesztő

Szirmay-Kalos László

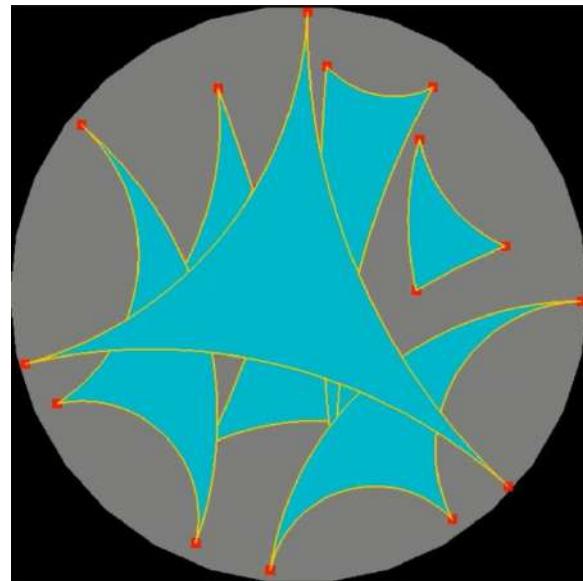




# Grafikus hardver/szoftver alapok

## Program: Hiperbolikus háromszögek

Szirmay-Kalos László

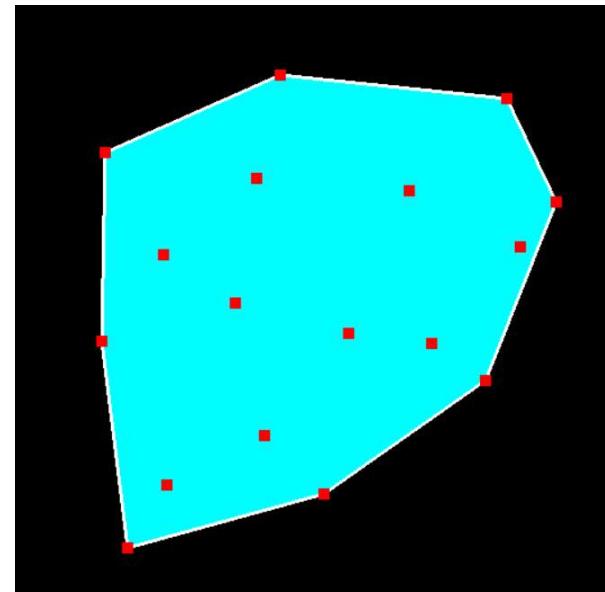




# Grafikus hardver/szoftver alapok

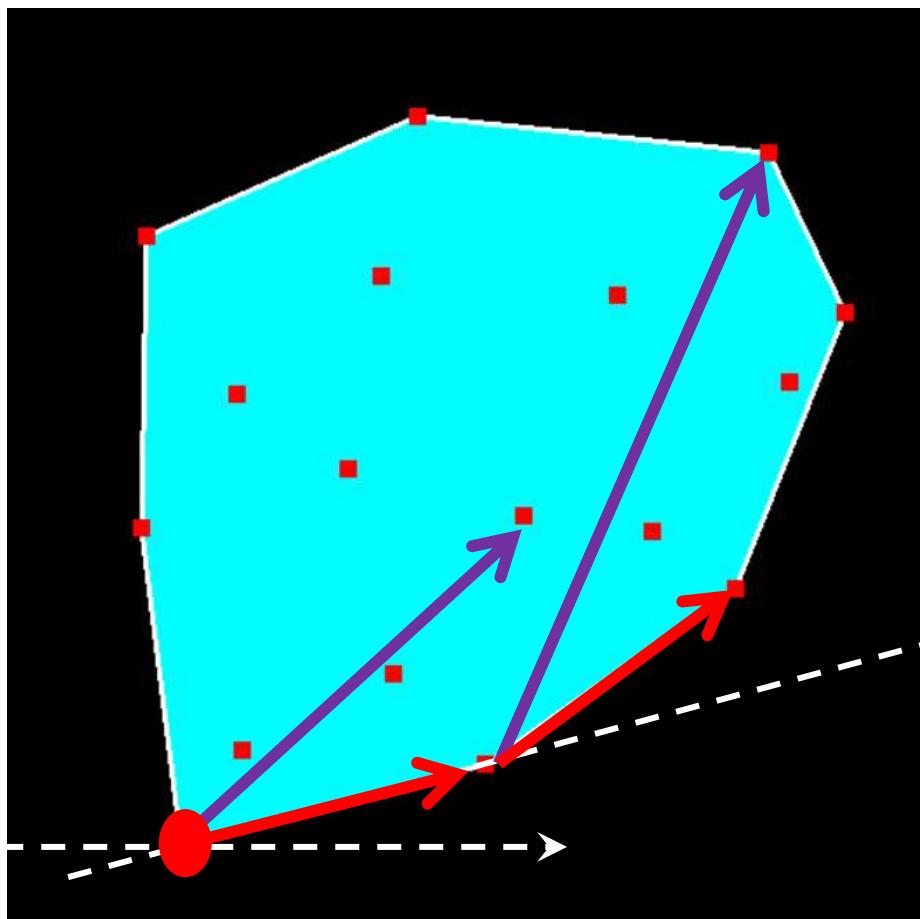
## Program: Konvex burok, interakció

Szirmay-Kalos László



# Konvex burok

- Minimális konvex ponthalmaz, ami az adott pontokat tartalmazza



Legalsó pontból indulunk, a kezdő irány balról jobbra.

```
While (vissza nem érünk) {  
    Következő pont,  
    amelyhez minimálisat  
    kell fordulni.  
}
```

# Csúcspont és pixel árnyalók

## Vertex shader:

```
layout(location = 0) in vec2 vertexPosition;  
  
void main() {  
    gl_Position = vec4(vertexPosition, 0, 1);  
}
```

## Fragment shader:

```
uniform vec3 color;  
out vec4 fragmentColor;  
  
void main() {  
    fragmentColor = vec4(color, 1);  
}
```

# Object

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

Object() {
    glGenVertexArrays(1, &vao); glBindVertexArray(vao);
    glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, NULL);
}

void updateGPU() {
    glBindVertexArray(vao); glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, vtx.size() * sizeof(vec2),
                 &vtx[0], GL_DYNAMIC_DRAW);
}

void Draw(int type, vec3 color) {
    if (vertices.size() > 0) {
        glBindVertexArray(vao);
        gpuProgram.setUniform(color, "color");
        glDrawArrays(type, 0, vertices.size());
    }
}
};

};
```

# Convex hull

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

# Convex hull előállítás

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

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

# Virtuális világ és megjelenítése

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

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

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

# Controller

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

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

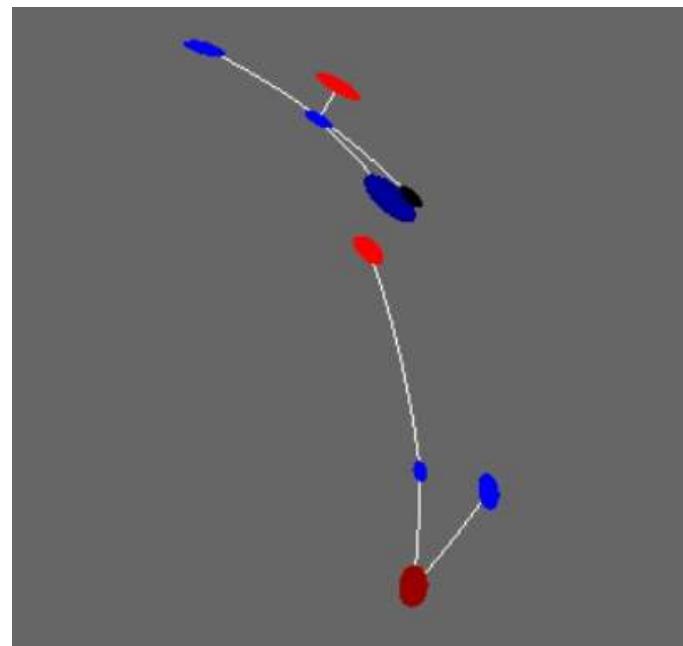
void onMouseMotion(int pX, int pY) {
    if (pickedPoint) {
        *pickedPoint = vec2(PixelToNDC(pX, pY));
        hull->update(); glutPostRedisplay(); // redraw
    }
}
```



# Grafikus hardver/szoftver alapok

## Program: Molekula dokkolás

Szirmay-Kalos László



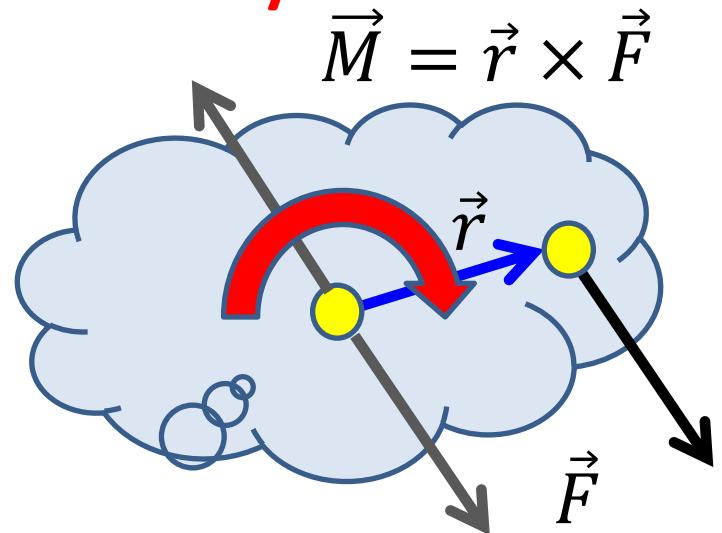
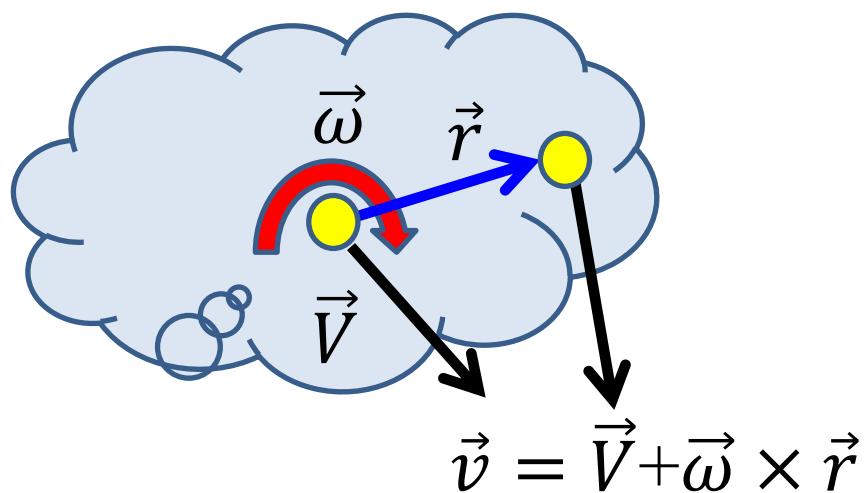
# Feladatleírás

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

# Molekula

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

# 2D merevtest kinematika/dinamika



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

$$\frac{d\omega}{dt} = \frac{\sum M}{\theta}$$

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

**Erők:**

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

# Dinamikai szimuláció

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

$$\frac{d\omega}{dt} = \frac{\sum M}{\theta}$$

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

for( $t = 0; t < T; t += dt$ ) { // onIdle  
  for each node  $i$  {

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

$$\sum M = \dots$$

$$\mathbf{v}_i += \sum \vec{F} / m \cdot dt$$

$$\mathbf{r}_i += \mathbf{v}_i \cdot dt$$

$$\omega_i += \sum M / \theta \cdot dt$$

$$\alpha_i += \omega_i \cdot dt$$

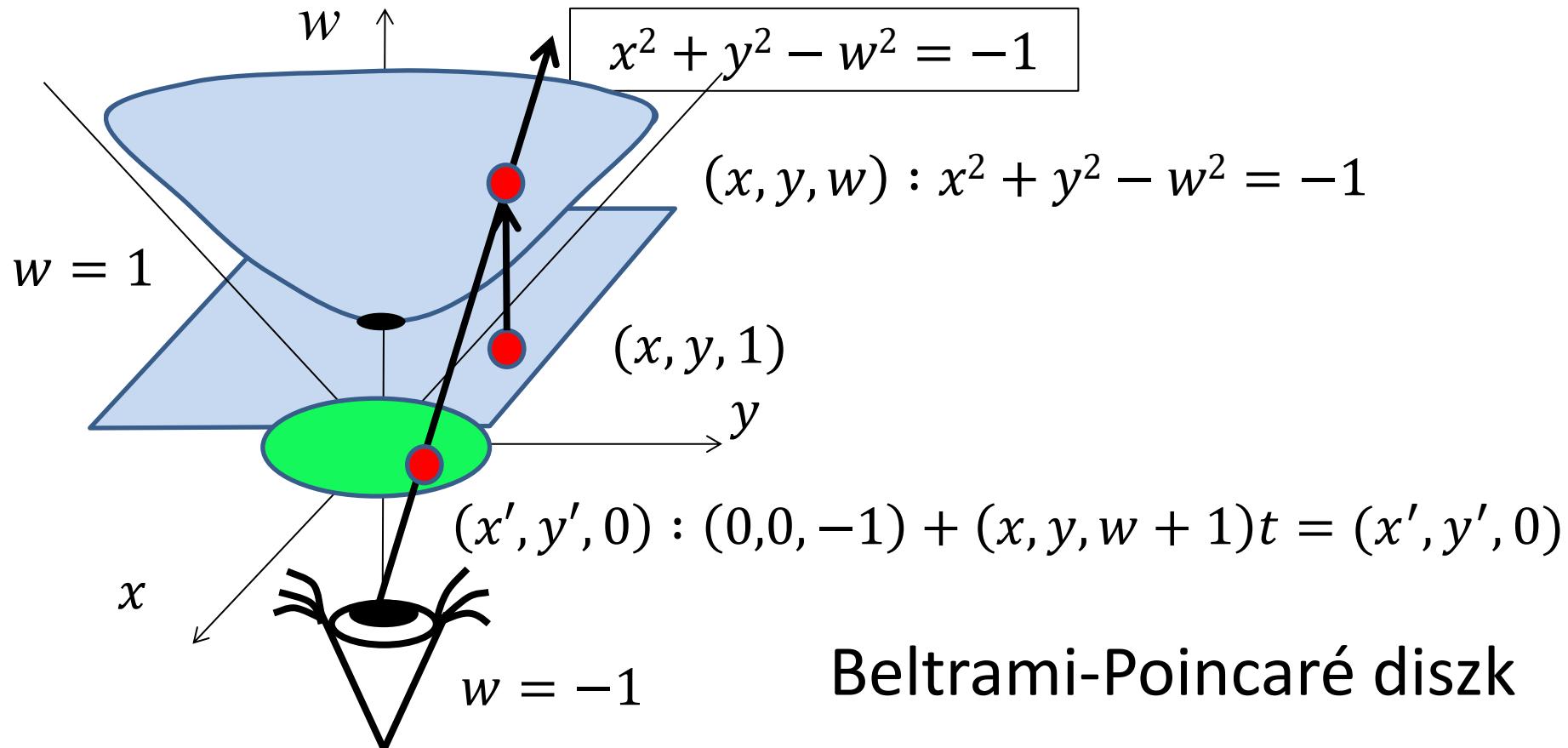
}

}

# Nézeti V és P transzformáció (vertex shader)

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

**P:** a teljes síkot az origó középpontú egységsugarú körbe viszi

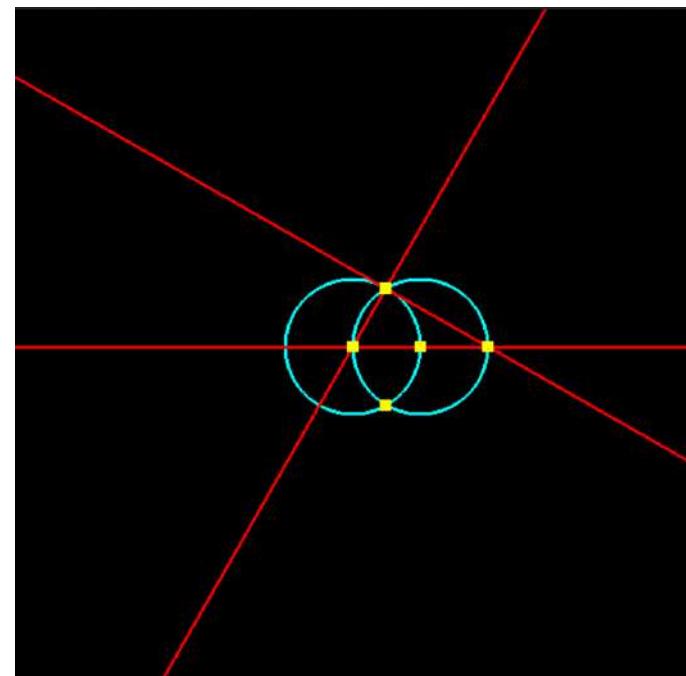




# Grafikus hardver/szoftver alapok

## Program: Körző és vonalzó

Szirmay-Kalos László



# Világ koordináták

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

# Modell

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

# Pont, egyenes, kör

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

# Kontroller: forgatókönyvek

## Körző befogása:

's'

klikk egy létező pont1-re

klikk egy létező pont2-re

Művelet = körző befogás, 1. pont jön

pont1 fehér Nem 1. pont jön

Sugár = |pont2 – pont1|, point1 eredeti szín

## Kör rajzolás a befogott sugárral:

'c'

klikk egy létező pont1-re

Művelet = kör rajzolás, 1. pont jön

kör felvétel: középpont = pont1, kör sugár = Sugár

## Egyenes rajzolás:

'l'

klikk egy létező pont1-re

klikk egy létező pont2-re

Művelet = egyenes rajzolás, 1. pont jön

pont1 fehér Nem 1. pont jön

egyenes felvétel: pont1, pont2; point1 eredeti szín

## Metszéspont:

'i'

klikk egy egyenesre vagy körre

klikk egy egyenesre vagy körre

Művelet = metszéspont, 1. primitív jön

primitív1 fehér Nem 1. primitív jön

primitív2, primitív1 eredeti szín

metszéspontok számítása és felvétele

# Események szerinti átrendezés

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

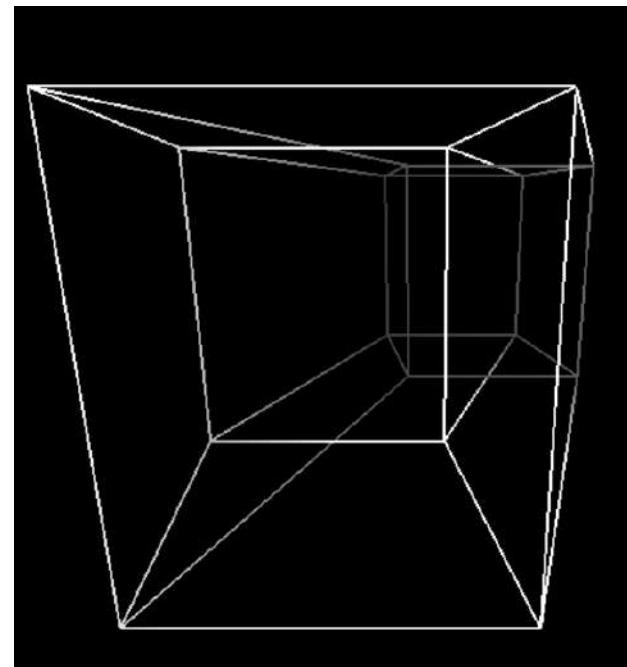
void onMouse(int button, int button_state, int pX, int pY) {
    if (button==GLUT_LEFT_BUTTON && button_state==GLUT_DOWN) {
        vec2 pickPoint = InputPipelineTranszformáció (pX, pY);
        switch (művelet) {
            case ...
            case ...
        }
        glutPostRedisplay();      //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



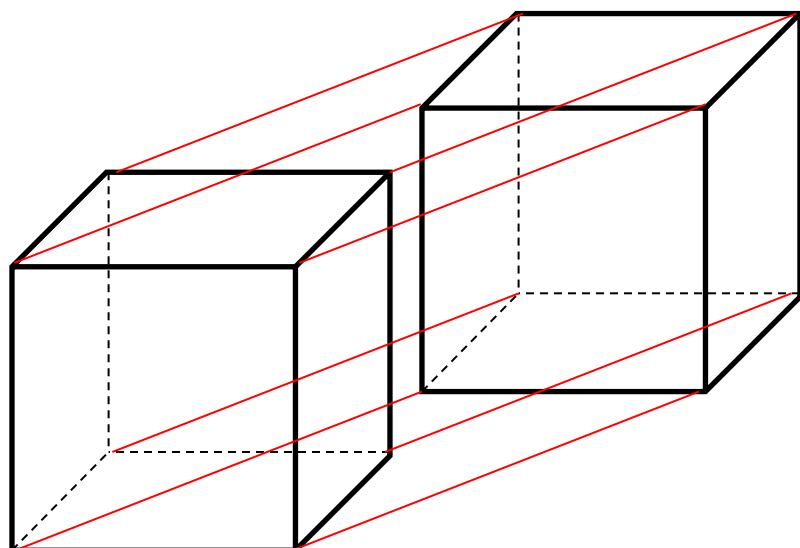
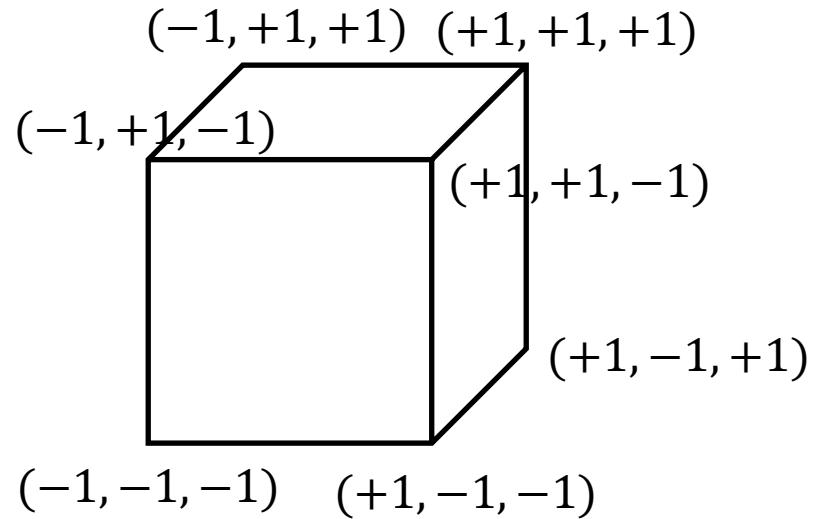
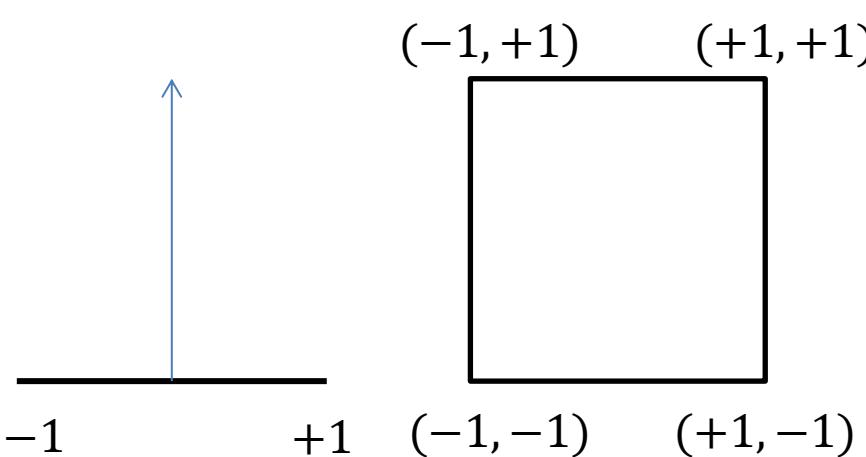
# Grafikus hardver/szoftver alapok

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

Szirmay-Kalos László



# N-dimenziós „kocka”



## Tesseract:

- Csúcsok:  $2^4 = 16$  darab  $(\pm 1, \pm 1, \pm 1, \pm 1)$
- Élek:  $\binom{4}{1} 2^3 = 32$  darab 1 csúcsból 4, 1 Hamming-ra
- Lap:  $\binom{4}{2} 2^2 = 24$
- Határoló 3D test:  $\binom{4}{3} 2 = 8$

```
#include "framework.h"

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

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

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

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

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

# Tesseract objektum

```
class Tesseract {
    const int D = 4;
    const int maxcode = (1 << D) - 1;
    unsigned int vao, vbo; // vertex array object id
    vector<float> vtx; // vertices of the object
    mat4 Rotate; // Transformation matrix

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

# Élek a GPU-ra

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

# Animáció és rajzolás

```
void Tesseract::Animate(float t) {
    Rotate = mat4(1, 0, 0, 0,
                  0, 1, 0, 0,
                  0, 0, cos(t), sin(t),
                  0, 0,-sin(t), cos(t));
    gpuProgram.setUniform(Rotate, "R");
}

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

# Vertex és fragment árnyalók

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

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

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

```
in float depthCue;
out vec4 fragColor;

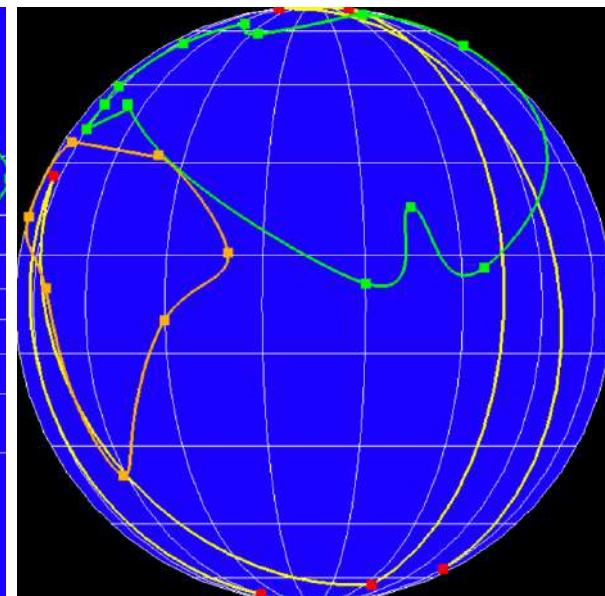
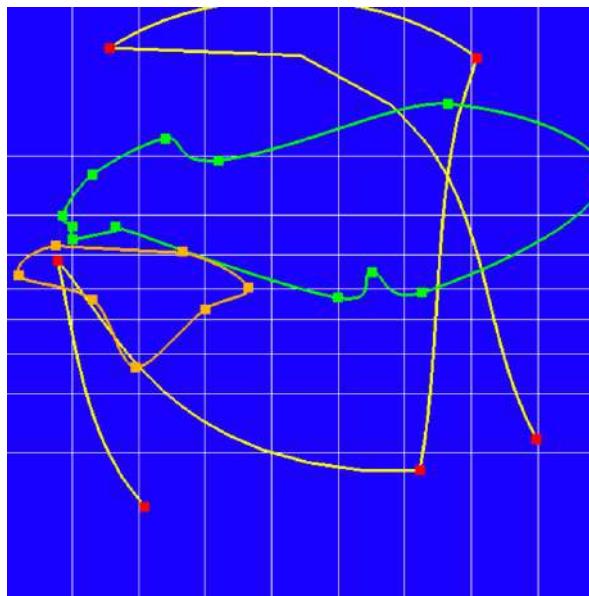
void main() {
    fragColor = vec4(depthCue, depthCue, depthCue, 1);
}
```



# Grafikus hardver/szoftver alapok

## Program: Mercator Útvonaltervező

Szirmay-Kalos  
László





# Specifikáció

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

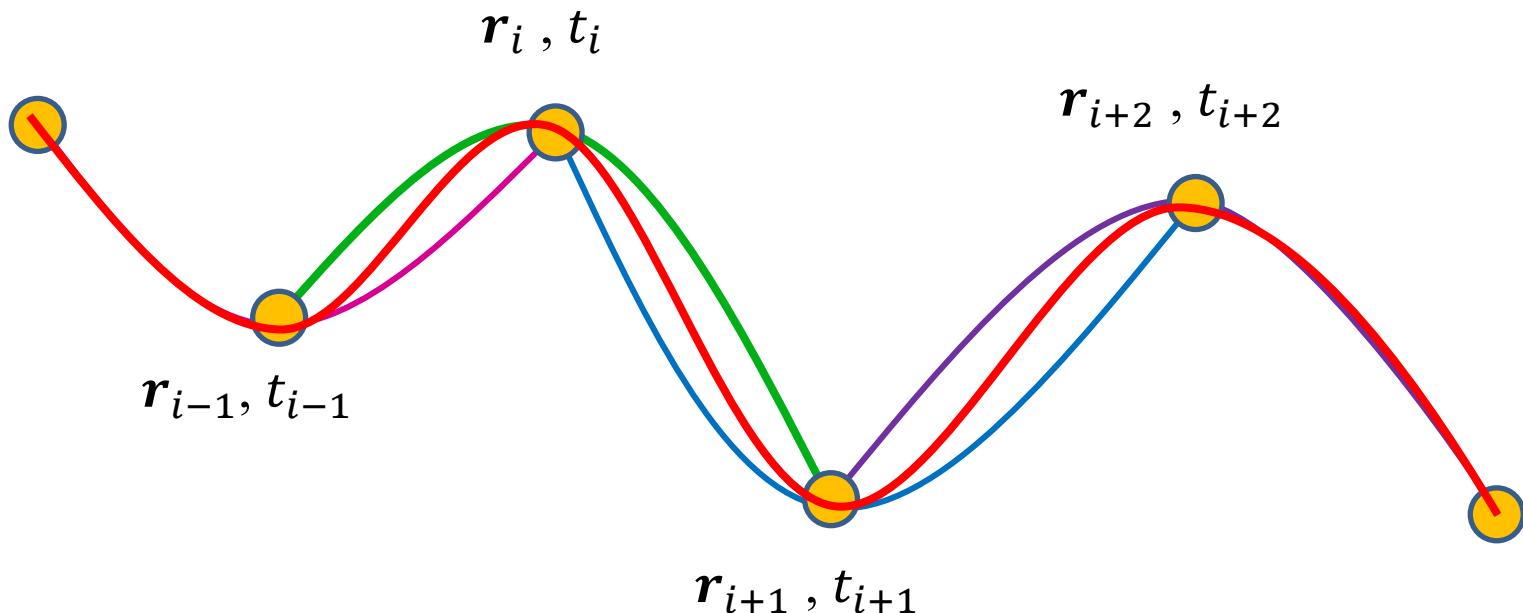
A útvonal állomásait az egér bal gomb lenyomásával bármely(!) nézetben kijelölhetjük. Az állomásokat a rendszer a gömbön sárga legrövidebb utakkal köti össze, és kiírja printf-fel az állomás szélesség-hosszúság koordinátáit és az utolsó állomástól mért hosszát (a föld sugara 6371 km).

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

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

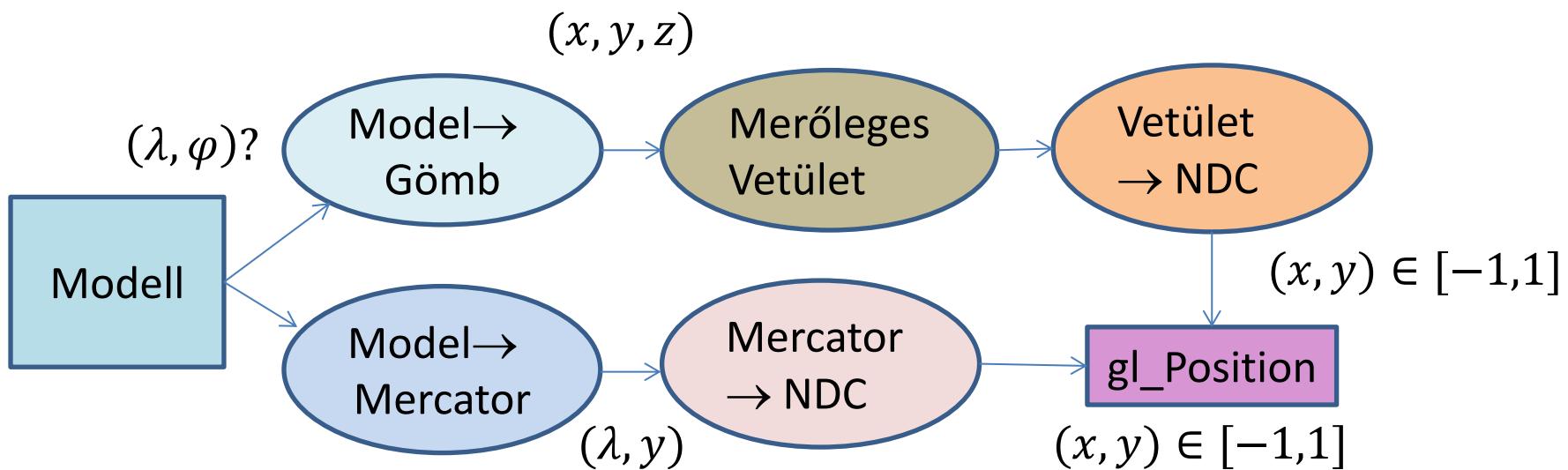
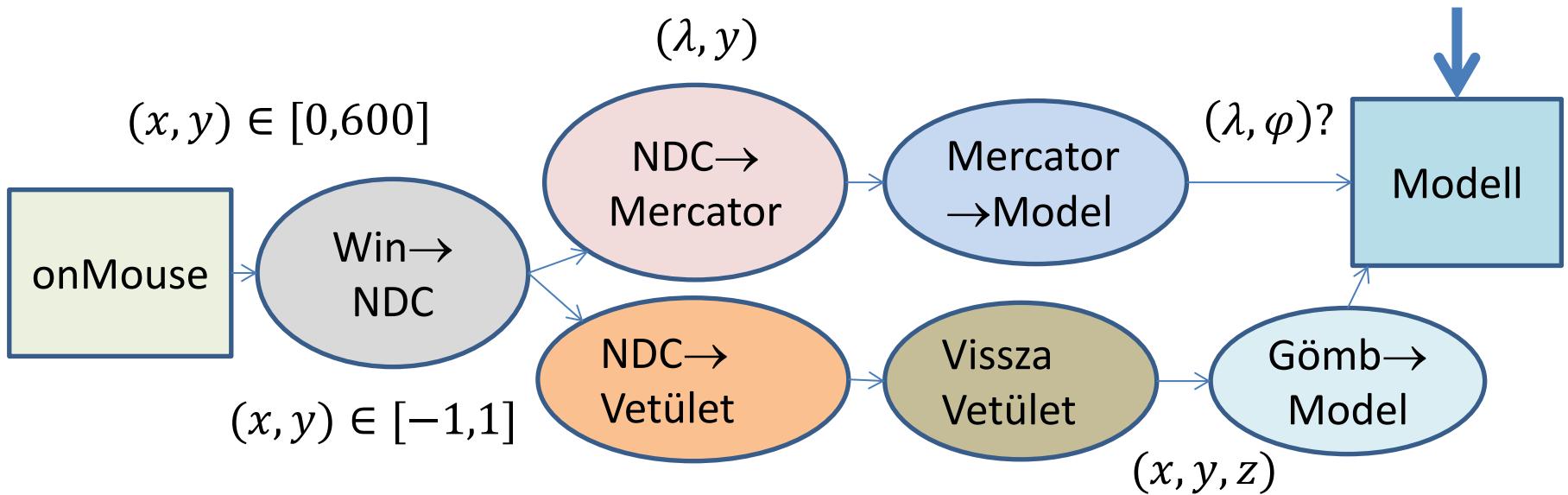
Afrika kontrollpontjai: (33, -5), (17, -16), (3, 6), (-35, 19), (-3, 40), (10, 53), (30, 33)

# O-spline



- Másodfokú szegmens:  $s_i(t) = a_i(t - t_i)^2 + b_i(t - t_i) + c_i$
- Összemosás:  $r(t) = (s_i(t)(t_{i+1} - t) + s_{i+1}(t)(t - t_i))/(t_{i+1} - t_i)$
- Ciklikus: A 0 előtt a legutolsó áll, a legutolsó után a 0.

# Modell koordinátarendszer?



# Objektumok

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

# Feladatok

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

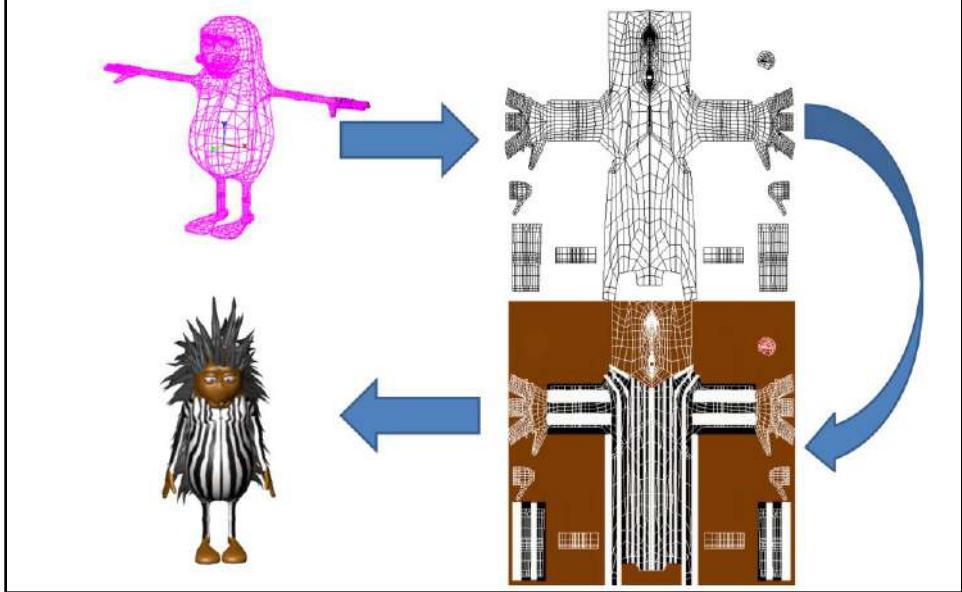
*"Everything must be made as simple as possible. But not simpler."*  
Albert Einstein

## 2D textúrázás

Szirmay-Kalos László



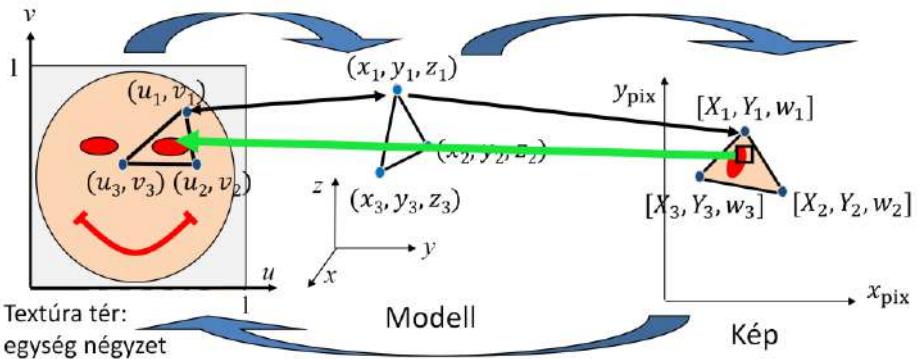
## 2D textúrázás



## 2D Textúrázás

$$[X, Y, w] = [u, v, 1] \cdot P$$

$$(x_{\text{pix}}, y_{\text{pix}}) = [X/w, Y/w]$$



$$[u/w, v/w, 1/w] = [x_{\text{pix}}, y_{\text{pix}}, 1] \cdot P^{-1}$$

$$[U, V, W] \rightarrow u = U/W, v = V/W$$

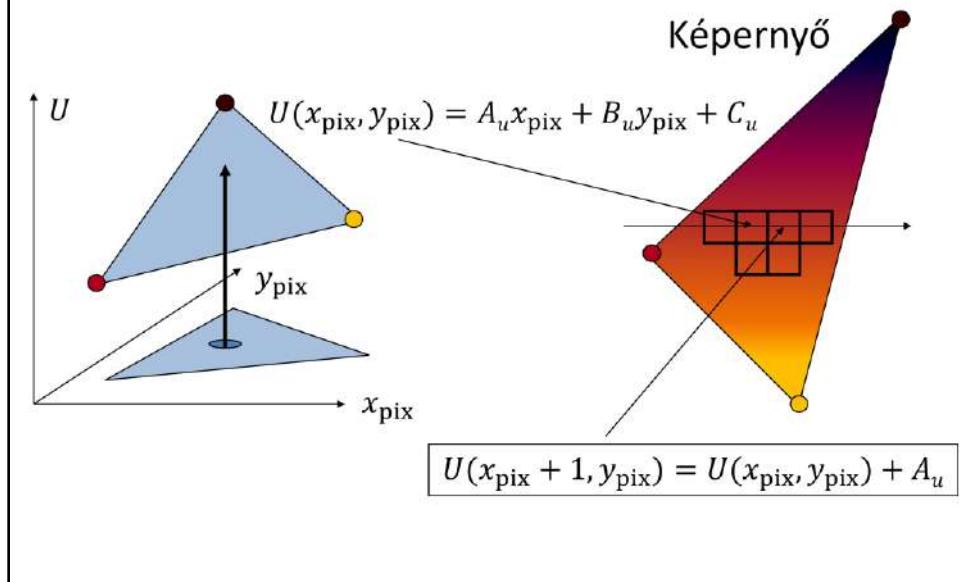
2D texture mapping can be imagined as wallpapering. We have a wallpaper that defines the image or the function of a given material property. This image is defined in texture space as a unit rectangle. The wallpapering process will cover the region with this image. This usually implies the distortion of the texture image. To execute texturing, we have to find a correspondence between the region and the 2D texture space. After vectorization, the region is a triangle mesh, so for each triangle, we have to identify a 2D texture space triangle, which will be painted onto the model triangle. The definition of this correspondence is called **parameterization**. A triangle can be parameterized with an affine transformation ( $x, y$  are linear functions of  $u, v$ ). This can be seen by counting the number of unknowns in a pair of linear functions (6) and counting the number of constraints for an affine transformation to map one given triangle onto another given triangle (this is also 6). So for unknown  $a_x, b_x, c_x, a_y, b_y, c_y$  parameters, we can establish a system of equations where the number of unknowns is the same as the number of equations.

Screen space coordinates ( $X, Y$ ) are obtained with affine or to be general with homogeneous linear transformation from  $x, y$ . Thus the mapping between texture space and screen space is also a homogeneous linear transformation:

The triangle is rasterized in screen space. When a pixel is processed, texture

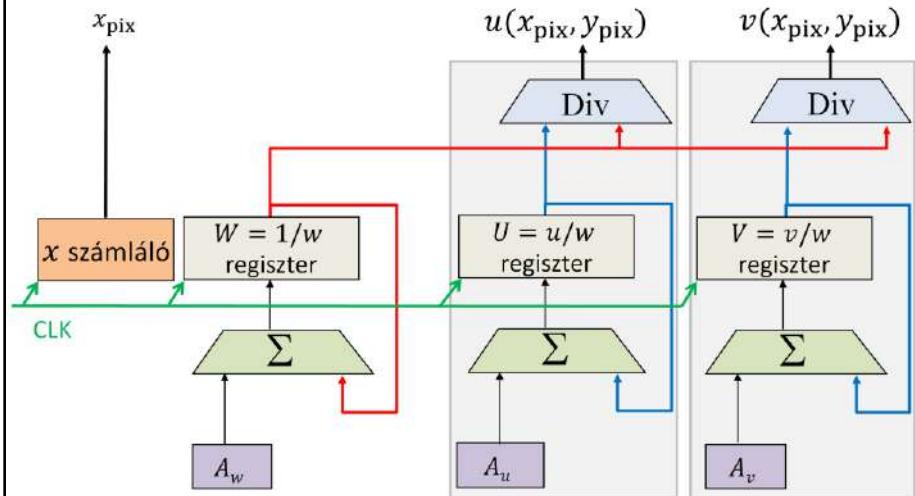
coordinate pair  $u, v$  must be determined from pixel coordinates  $X, Y$ , which requires the execution of the inverse of this homogeneous linear transformation. If we need the result in Cartesian coordinates, homogeneous division is also needed. In other words, homogeneous coordinates  $[U, V, H]$  depend linearly on pixel coordinates  $X$  and  $Y$ , and then Cartesian texture coordinates are obtained as  $u = U/H, v = V/H$ .

## Lineáris interpoláció



The crucial problem is then the linear interpolation of the color, properties in general, or texture coordinates for internal pixels. This interpolation must be fast as we have just a few nanoseconds for each pixel. Let us denote the interpolated property by  $U$ . Linear interpolation means that  $U$  is a linear function of pixel coordinates  $X, Y$ . Linear function  $U(X, Y) = A_u X + B_u Y + C_u$  evaluation would require two multiplications and two additions. This can be reduced to a single addition if we use the incremental concept and focus on the difference between the  $U$  values of the current and previous pixels in this scanline:  $U(X+1, Y) = U(X, Y) + A_u$ .

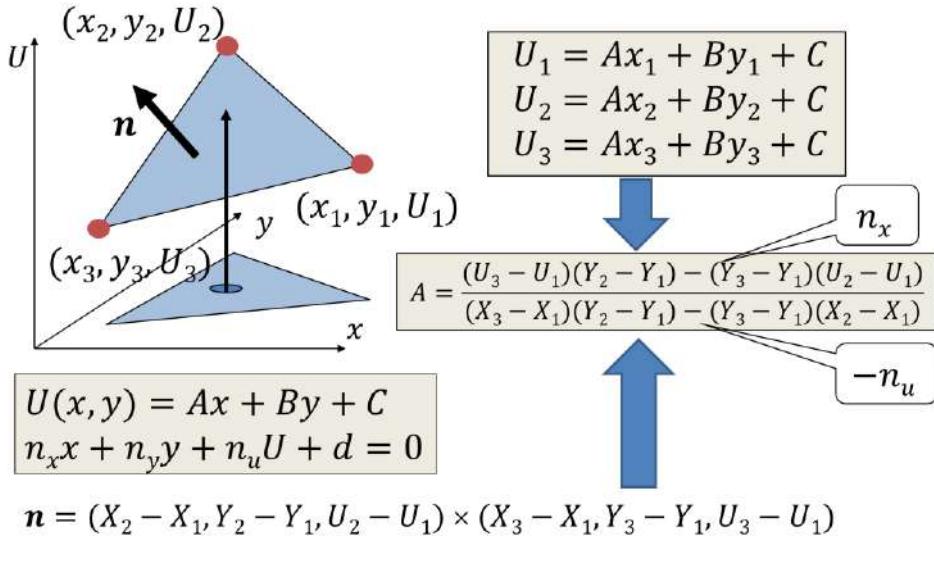
## Interpolációs hardver



Such an incremental algorithm is easy to be implemented directly in hardware. A counter increments its value to generate coordinate X for every clock cycle. A register that stores the actual U coordinate in fixed point, non-integer format (note that increment  $A$  is usually not an integer). This register is updated with the sum of its previous value and  $A$  for every clock cycle. For a texture coordinate, we execute linear interpolation for the three homogeneous coordinates  $[U, V, H]$ .

From these, the u, v Cartesian texture coordinates are obtained by two additional division.

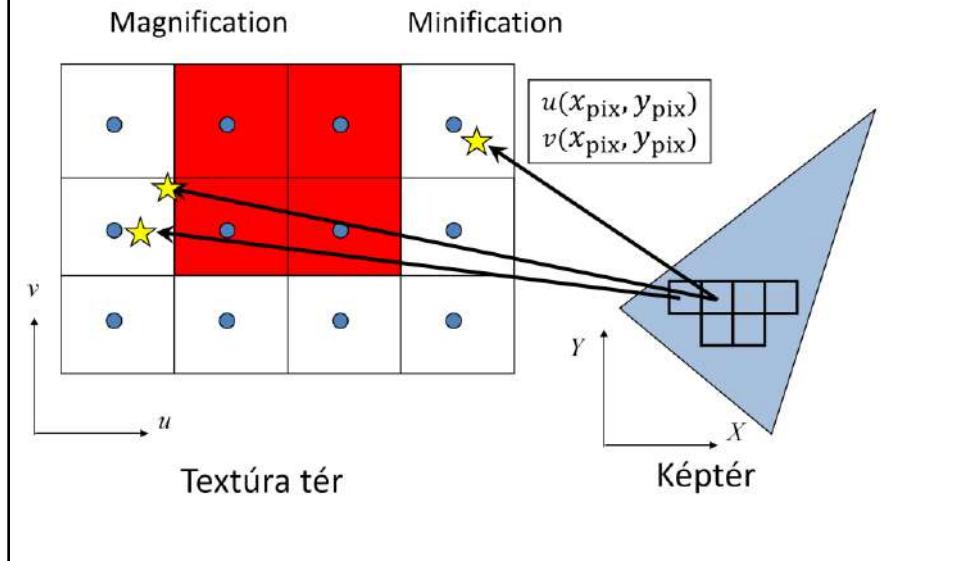
## Triangle setup



The final problem is how increment  $A$  is calculated. One way of determining it is to satisfy the interpolation constraints at the three vertices. It means that substituting the  $X, Y$  coordinates of the vertices into the linear expression, we should obtain the value to be interpolated at the three vertices. This is a system of linear equations for unknown  $A, B, C$ .

The other way is based on the recognition that we work with the plane equation where  $X, Y, U$  coordinates are multiplied by the coordinates of the plane's normal. The normal vector, in turn, can be calculated as a cross product of the edge vectors.

## Textúra szűrés (GL\_NEAREST)

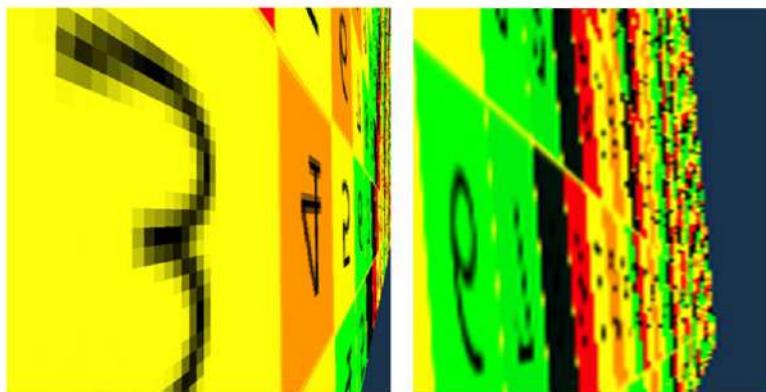


Rasterization visits pixels inside the projection of the triangle and maps the center of the pixel from screen space to texture space to look up the texture color. This mapping will result in a point that is in between the texel centers.

Suppose that we step onto the next pixel. Consequently, the transformed  $u$ ,  $v$  texture coordinates also change. This change can be smaller than a texel or bigger than a texel. Looking at this situation from the other direction, the first case is when a texel covers multiple pixels. This is called **magnification**. The second case is when many texels are mapped to a single pixel and the pixel color is determined by that texel which is lucky enough to be mapped onto the pixel center. This case is called **minification**.

From signal processing point of view, the texture can be considered as a signal, which is sampled at the pixel centers. In the optimal case, one texel corresponds to one pixel. In case of magnification, texels will be large, which is oversampling. In case of minification, texels are too small and are undersampled and the result will be a mess or noise.

## Textúratér és képtér kapcsolata



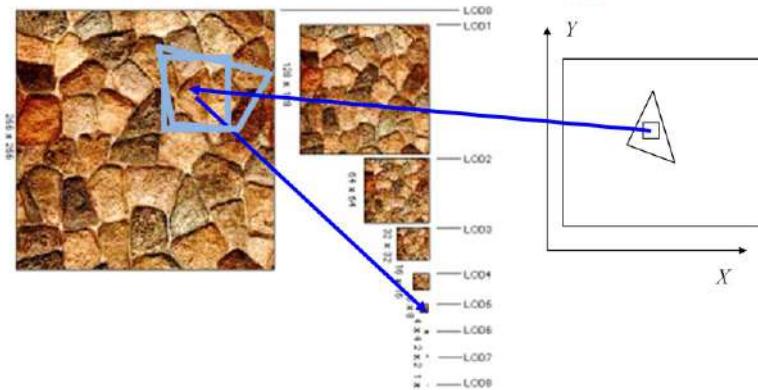
Magnification

Minification

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

In case of oversampling or undersampling artifacts occur. Magnification makes the individual texels obvious. Minification converts the pattern to a noise. When these images were rendered, the pixel is colored with the texel covering its center. This sampling strategy is called **GL\_NEAREST** in OpenGL. The sampling strategy can be set with the **glTexParameter** function separately for the minification (**GL\_TEXTURE\_MIN\_FILTER**) and magnification (**GL\_TEXTURE\_MAG\_FILTER**).

## Mip-map (multum in parvo) Minification-ra jó

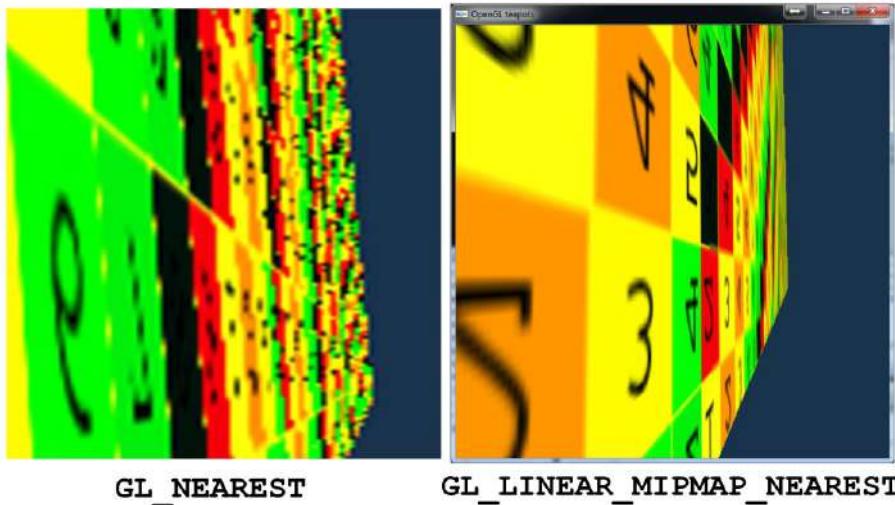


```
a) glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  
                   GL_LINEAR_MIPMAP_NEAREST); // Mip-mapping  
  
b) glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  
                   GL_LINEAR_MIPMAP_LINEAR); // Tri-linear filtering
```

The solution for the undersampling problem is filtering. Instead of mapping just the center of the pixel rectangle must be mapped to texture space at least approximately, and the average of texel colors in this region should be returned as a color. However, it would be too time consuming.

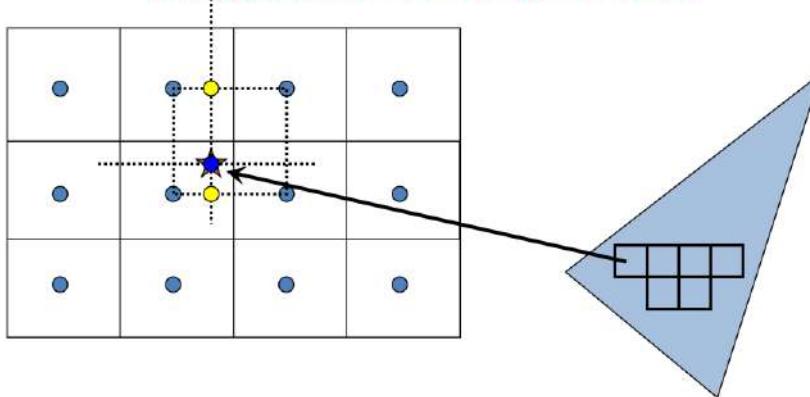
One efficient approximation is to prepare the texture not only in its original resolution, but also in half resolution, quarter resolution, etc. where a texel represents the average color of a square of the original texel. During rasterization, OpenGL estimates the magnification factor, and looks up the appropriate version of filtered, downsample texture. The collection of the original and downsampled textures is called mip-map.

## Mip-map (GL\_LINEAR\_MIPMAP\_...)



Indeed, with mip-mapping undersampling artifacts disappear. Mip-mapping can be selected with **GL\_LINEAR\_MIPMAP\_NEAREST** texture sampling strategy.

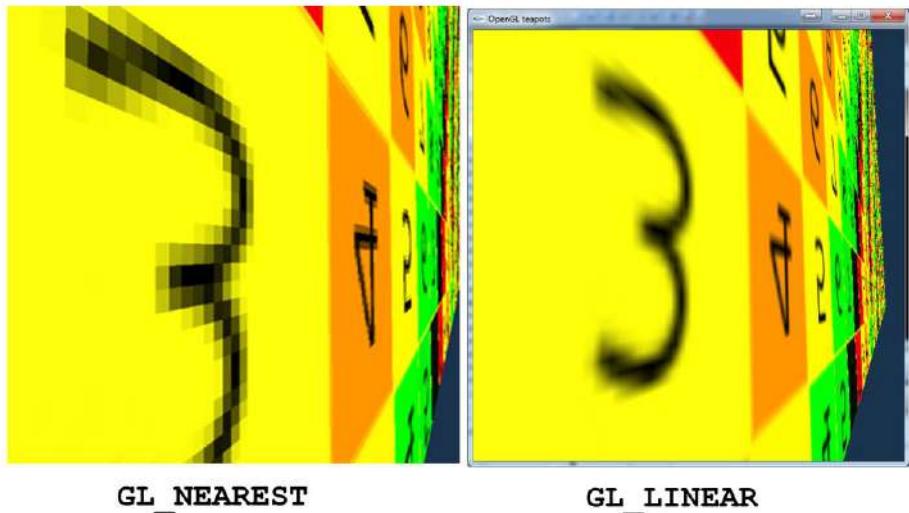
## Bi-linear textúra szűrés (GL\_LINEAR) Magnification-ra igazán jó



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

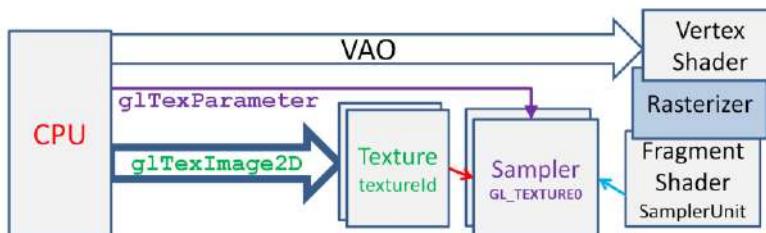
There is another simpler filtering scheme, which is particularly good for magnification, i.e. to reduce oversampling artifacts. In this method, when a pixel center is mapped to texture space, not only the closest texel is obtained but the four closest ones, and the filtered color is computed as the bi-linear interpolation of their colors.

## Bi-linear filtering (GL\_LINEAR)



As these images demonstrate, bi-linear filtering smears the ugly texel edges.

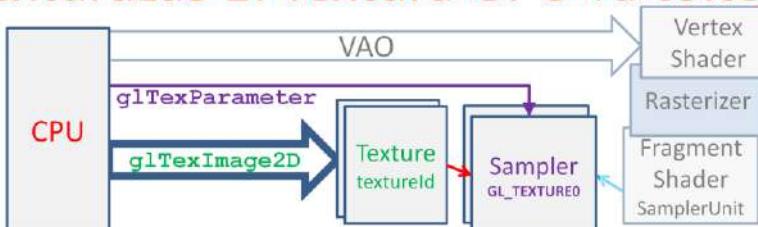
## Textúrázás a GPU-n



If we wish to use texturing, four steps should be executed. In this slide we show how to upload an image to the GPU that is to be used as a 2D texture. Additionally, texture sampling/filtering is also specified, which is nearest neighbor in case of minification and bi-linear filtering in case of magnification.

Note that OpenGL is an output library, so it gives no help to create or read a texture from a file. This is the programmers' responsibility, which should be done in the LoadImage function.

## Textúrázás 1: Textúra GPU-ra töltése

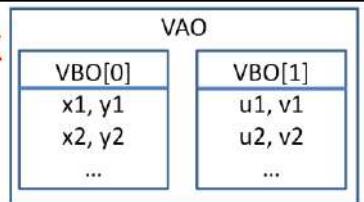


```
unsigned int textureId;  
  
void UploadTexture(int width, int height, vector<vec4>& image) {  
    glGenTextures(1, &textureId);  
    glBindTexture(GL_TEXTURE_2D, textureId);      // binding  
    Mip-map szint      célfórmátum      Border  
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,  
                GL_RGBA, GL_FLOAT, &image[0]); //Texture -> GPU  
    forrás  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
}
```

If we wish to use texturing, four steps should be executed. In this slide we show how to upload an image to the GPU that is to be used as a 2D texture. Additionally, texture sampling/filtering is also specified, which is nearest neighbor in case of minification and bi-linear filtering in case of magnification.

Note that OpenGL is an output library, so it gives no help to create or read a texture from a file. This is the programmers' responsibility, which should be done in the LoadImage function.

## Textúrázás 2: Objektumok felszerelése textúra koordinátákkal



```
glGenVertexArrays(1, &vao);
 glBindVertexArray(vao);

 glGenBuffers(2, vbo); // Generate 2 vertex buffer objects

 // vertex coordinates: vbo[0] -> Attrib Array 0 -> vertices
 glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
 float vtxs[] = {x1, y1, x2, y2, ...};
 glBufferData(GL_ARRAY_BUFFER, sizeof(vtxs), vtxs, GL_STATIC_DRAW);
 glEnableVertexAttribArray(0);
 glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, NULL);

 // vertex coordinates: vbo[1] -> Attrib Array 1 -> uvs
 glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
 float uvs[] = {u1, v1, u2, v2, ...};
 glBufferData(GL_ARRAY_BUFFER, sizeof(uvs), uvs, GL_STATIC_DRAW);
 glEnableVertexAttribArray(1);
 glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, NULL);
```

The second step equips the object to be textured with texture coordinates or so called uvs. That is, for every vertex we also specify a texture space point from where the color or data should be fetched if this point is rendered.

In the shown program or VAO has two VBOs, one stores the modeling space Cartesian coordinates of the points and the second the texture space coordinates of the same points. We direct modeling space coordinates to input register 0 and texture space coordinates to input register 1.

## Textúrázás 3: Vertex és Pixel Shader

```
layout(location = 0) in vec2 vtxPos;
layout(location = 1) in vec2 vtxUV;

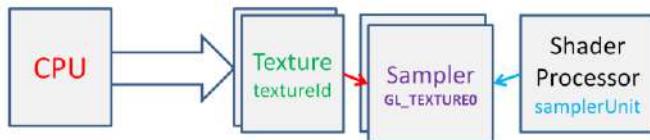
out vec2 texcoord; ——————→ Raszterizáció  
void main() {
    gl_Position = vec4(vtxPos, 0, 1) * MVP;
    texcoord = vtxUV;
    ...
}

uniform sampler2D samplerUnit;
in vec2 texcoord; ← Interpoláció
out vec4 fragmentColor;

void main() {
    fragmentColor = texture(samplerUnit, texcoord);
}
```

The vertex shader gets the location of the point and also the uv associated with this point. The location is transformed to normalized device space, the texture coordinate is only copied by the vertex shader. The fixed function hardware interpolates the texture coordinates and the fragment shader looks up the texture memory via the sampler unit.

## Textúrázás 4: Aktív textúra és sampler



```
unsigned int textureId;

void Draw( ) {
    int sampler = 0; // which sampler unit should be used

    int location = glGetUniformLocation(shaderProg, "samplerUnit");
    glUniform1i(location, sampler);

    glActiveTexture(GL_TEXTURE0 + sampler); // = GL_TEXTURE0
    glBindTexture(GL_TEXTURE_2D, textureId);

    glBindVertexArray(vao);
    glDrawArrays(GL_TRIANGLES, 0, nVtx);
}
```



When the object is rendered, we should select the active texture and also connect the shader processor to this texture via a sampler unit. The samplerUnit variable of the shader program gets the id of the sampler unit (zero in this example). With the glActiveTexture and glBindTexture calls, we also establish the connection between the texture in the texture memory and the sampler unit.



## 2D textúrázás

### 1. Program: Textúrázott négyzet

Szirmay-Kalos László



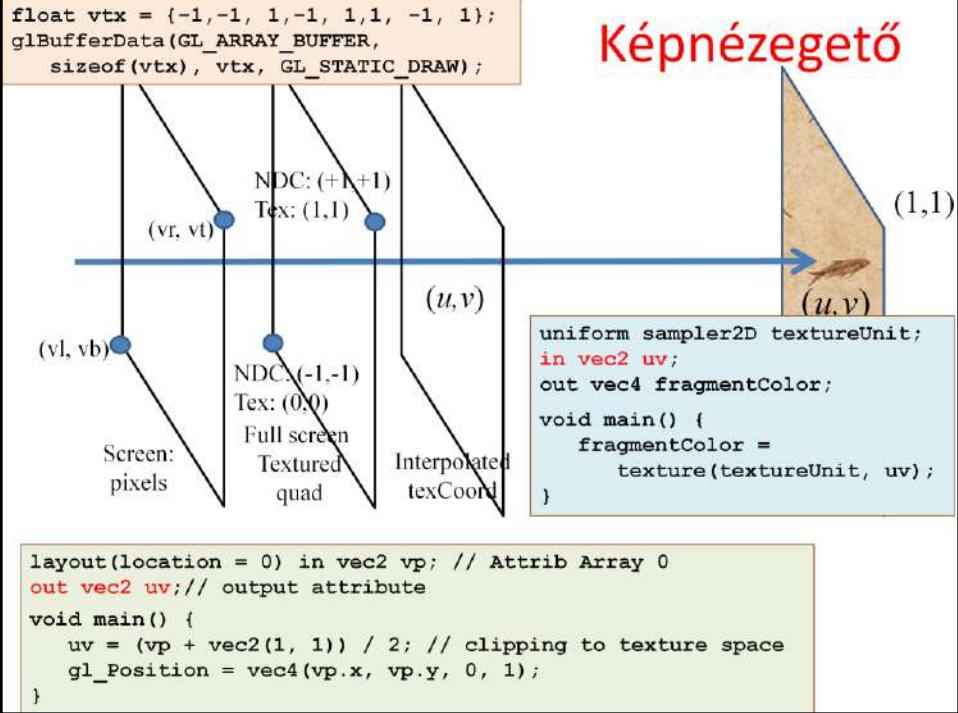
"The message of this lecture is that black holes ain't as black as they are painted. So if you feel you are in a black hole, don't give up – there's a way out."

Stephen Hawking

## Képnézegető és nemlineáris 2D képeffektusok

Szirmay-Kalos László

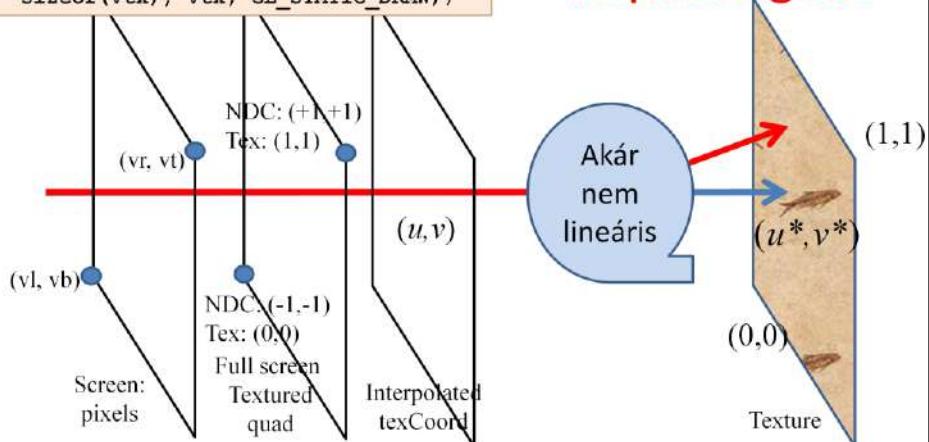




Let us implement an image viewer program. To copy an image into the raster memory, we should draw a full viewport sized quad textured with the image. In normalized device space, the full viewport quad has corners  $(-1,-1)$ ,  $(1,-1)$ ,  $(1,1)$ ,  $(-1,1)$ , which are stored in the vbo. From these, the texture coordinates  $(0,0)$ ,  $(1,0)$ ,  $(1,1)$ ,  $(0,1)$  addressing the four corners of the texture rectangle are computed by the vertex shader. The fragment shader looks up the texture with the interpolated uv.

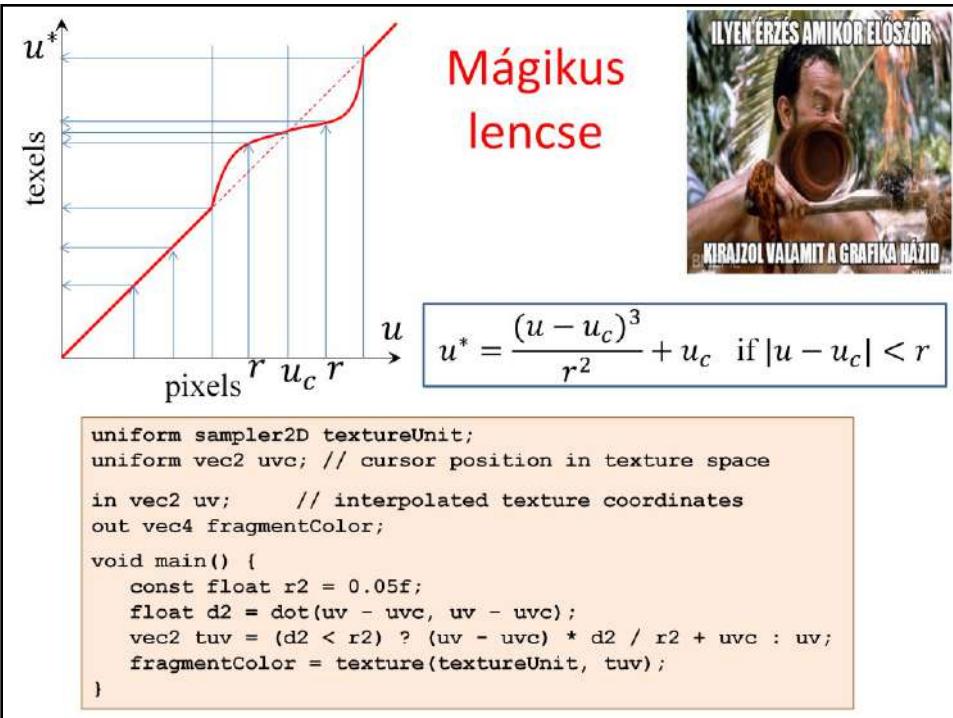
```
float vtx = {-1,-1, 1,-1, 1,1, -1, 1};
glBufferData(GL_ARRAY_BUFFER,
             sizeof(vtx), vtx, GL_STATIC_DRAW);
```

## Képnézegető



```
layout(location = 0) in vec2 vp; // Attrib Array 0
out vec2 uv; // output attribute
void main() {
    uv = (vp + vec2(1, 1)) / 2; // clipping to texture space
    gl_Position = vec4(vp.x, vp.y, 0, 1);
}
```

If we modify the interpolated texture coordinate by some function, exciting image effects can be produced.



Normally, the texture is fetched at interpolated texture coordinate  $uv$ , but now we shall transform it to  $tuv$  and get the texture at  $tuv$ . The transformation  $uv$  to  $tuv$  uses the texture space location of the cursor to make the effect interactive.

Note that if  $tuv$  is the translation of  $uv$ , then the effect corresponds to the translation of the image into the opposite direction. Similarly, if  $tuv$  is the rotated version of  $uv$ , we observe the effect as rotating the image backwards. Generally, the inverse of the transformation of  $uv$  to  $tuv$  will be applied to the image.

Note also that we can use non-linear transformations as well since finite number of texel centers are transformed. In this demo we modify the originally linear dependence of  $tuv$  (or  $u^*$ ) on  $uv$  and insert a non-linear cubic segment. Close to  $u_c$ , the slope of the function is less than 1, i.e. when we step to the next pixel, the corresponding movement in the texture is less than normal. Here a texel is seen on more pixels, thus the image is magnified. When the slope is greater than 1, the image is contracted. So the seen effect is what a lens would produce.

# Örvény: Swirl



```
uniform sampler2D textureUnit;
uniform vec2 uvc; // cursor position in texture space

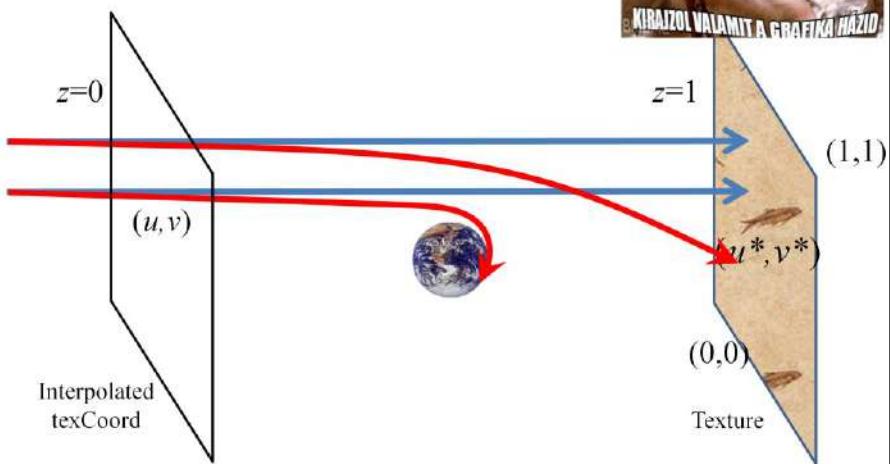
in vec2 uv;      // interpolated texture coordinates
out vec4 fragmentColor;

void main() {
    const float a = 8, alpha = 15;
    float ang = a * exp( -alpha * length(uv - uvc) );
    mat2 rotMat = mat2( cos(ang), sin(ang),
                        -sin(ang), cos(ang) );
    vec2 tuv = (uv - uvc) * rotMat + uvc;
    fragmentColor = texture(textureUnit, tuv);
}
```

A swirl is a rotation where the angle (or speed) of the rotation grows towards the center of the swirl. A rotation is a mat2 matrix multiplication where the pivot point is the current cursor location, uvc. To control the angle of the rotation, we compute the distance from the center and use the  $a \cdot \exp(-\alpha \cdot x)$  function to obtain smaller angles farther from the center.



## Gravitáció (fekete lyukak)



In this effect, we assume that a large mass but small size object, e.g. a black hole is in front of the image, which distorts space and also lines of sight.

**Ekvivalencia elv**

$$\Delta d = \frac{g}{2} (\Delta t)^2 = \frac{fM}{2r^2 c^2} (\Delta s)^2 = \frac{r_0}{4r^2} (\Delta s)^2$$

$\frac{r_0}{2}$  : Schwarzschild rádiusz

```
void main() {
    const float r0 = 0.09f, ds = 0.001f;
    vec3 p = vec3(uv,0), dir = vec3(0,0,1), blackhole = vec3(uvc,0.5f);
    float r2 = dot(blackhole - p, blackhole - p);
    while (p.z < 1 && r2 > r0 * r0) {
        p += dir * ds;
        r2 = dot(blackhole - p, blackhole - p);
        vec3 gDir = (blackhole - p)/sqrt(r2); // gravity direction
        dir = normalize(dir * ds + gDir * r0 / r2 / 4 * ds * ds);
    }
    if (p.z >= 1) fragmentColor = texture(textureUnit,vec2(p.x,p.y));
    else           fragmentColor = vec4(0, 0, 0, 1);
}
```

The amount of distortion can be computed exploiting the equivalence principle that states that gravity and acceleration are the same and cannot be distinguished. We can easily determine the amount of light bending when our room is accelerating in a direction. Then, the same formula is applied when the room stands still but a black hole distorts the space.

## Hullám: Wave

The diagram shows a cross-section of water with a wave. A vertical line represents the water surface. A horizontal line at the bottom represents the ground. A blue rectangular area represents a texture. A point **uvc** is on the texture, and a point **uv** is on the water surface directly above it. The distance between them is labeled **d**. A point **tuv** is on the water surface to the right of **uv**, at a height labeled **waterDepth=1**. A green line segment from **uvc** to **uv** is labeled **waveDist=c\*time**. The angle between the vertical and this line is **angIn**. A red line segment from **uv** to **tuv** is labeled **angRefr**. The angle between the vertical and this line is **angRefr**. The diagram also shows the angle **angIn-angRefr**.

```

uniform float time;
const float PI = 3.14159265, n = 1.33, c = 0.1, aMax = 0.1;
void main() {
    float d = length(uv - uvc), waveDist = c * time;
    if (abs(d - waveDist) < waveWidth) {
        float angIn = aMax/waveDist * sin((waveDist-d)/waveWidth*PI);
        float angRefr = asin(sin(angIn)/n);
        vec2 dir = (uv - uvc)/d;
        vec2 tuv = uv + dir * tan(angIn - angRefr) * waterDepth;
        fragmentColor = texture(textureUnit, tuv);
    } else {
        fragmentColor = texture(textureUnit, uv);
    }
}

```

In this final demo, we assume that image is covered by water and its surface is distorted by a wave started in a single point and moving in all directions with speed  $c$ . The width of the wave is  $waveWidth$ , its amplitude is decreased with the travelled distance. The distortion is computed by applying the Snells refraction law on the water surface.

*"Why do two colors, put one next to the other, sing? Can one really explain this? no. Just as one can never learn how to paint."*

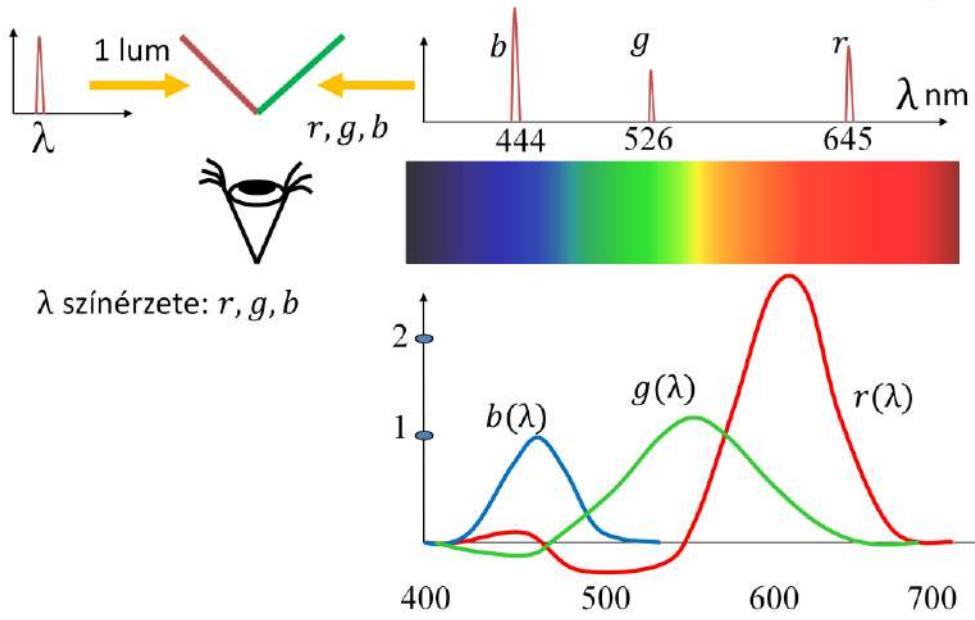
Pablo Picasso

# Színek

Szirmay-Kalos László



## Színérzékelés: monokromatikus fény



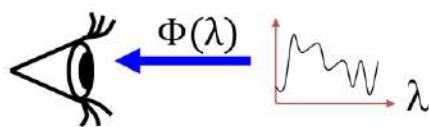
Light is an electromagnetic wave, color is just an illusion created by the human eye and the brain. As the eye is a poor spectrometer, we can cheat it with a different spectrum, the eye cannot tell the difference. This fact is exploited by displays, which can emit light just around three wavelengths. So the task is to convert the computed spectrum to the intensities of the three lamps associated with a pixel. To solve this, we should understand how the illusion of color is created. As the illusion is deep in our brain, we can use only subjective comparative experiments to find out what color means.

In our experiment, we have two white sheets, the first is illuminated by a unit radiance monochromatic light beam of wavelength  $\lambda$ , the other is by three lamps of controllable intensities and of wavelengths, say, 444, 526, 645 nanometers, which could be seen as red, green and blue (we could choose other reference wavelengths as well, they just have to be far enough; this particular selection is justified by the fact that there exist materials that emit light on these wavelengths). A human observer sits in front of the two white sheets and his task is to control the intensities of the three lamps in order to eliminate any perceived difference between the two sheets. If it happens, the monochromatic light and the controlled three component light provide the same color and are called metamers. If the same experiment is repeated in many discrete wavelengths, three color matching functions can be obtained, which are going to be our model of the human color perception.

Note that the red and the green matching function have negative parts as well, which means, for example, that the 500 nm light can be matched only if some red is added to it.

In the second experiment we can try to match two, three, etc. component light beams and beams of non-unit intensity. We will come to the conclusion that the corresponding r,g,b values of polychromatic light are the sums of the r,g,b primaries of the monochromatic components, and also that if the intensity of the beam is not unit, then the r,g,b intensities should also be multiplied by the same factor. This means that colors are linear objects.

## Színérzékelés: polikromatikus fény (Hermann) Grassmann törvények



$$R = \int \Phi(\lambda)r(\lambda)d\lambda$$

$$G = \int \Phi(\lambda)g(\lambda)d\lambda$$

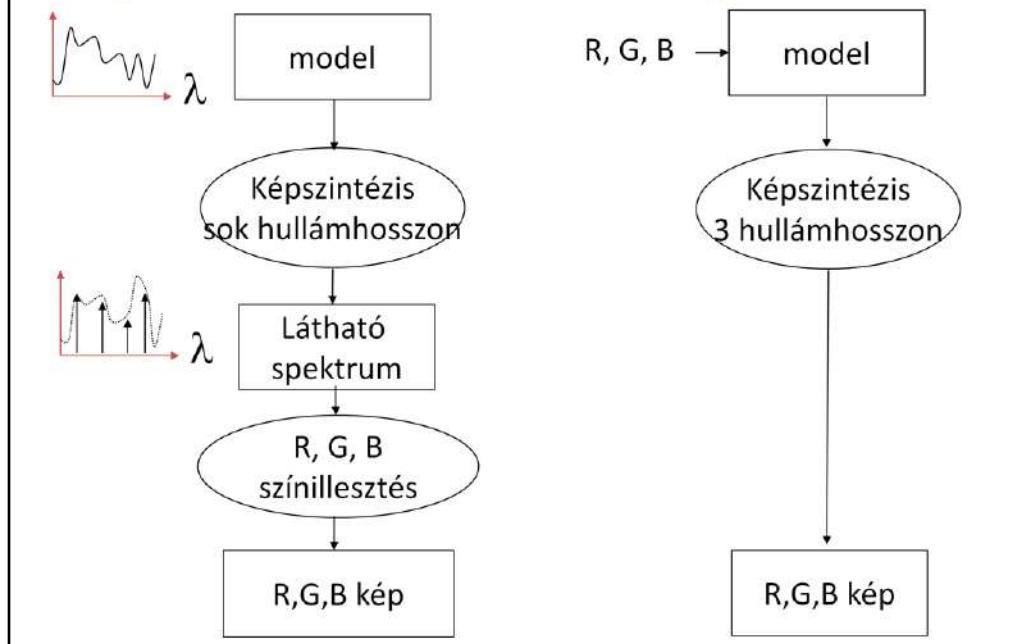
$$B = \int \Phi(\lambda)b(\lambda)d\lambda$$

### A színérzet lineáris (Grassmann törvények)

- Kétszer akkora spektrumhoz kétszer akkora  $R, G, B$  tartozik
- Különböző spektrumok összegéhez az  $R, G, B$ -k összege tartozik

Based on these experiments, we can establish the Grasmann laws of color science. Any spectrum can be matched with three primaries by weighting the monochromatic components by the color matching functions and adding (integrating) different monochromatic components.

## Spektrális versus RGB képszintézis



A physically plausible simulation would be executed on many wavelengths (note that wavelengths can be handled independently) resulting in a visible spectrum. The final step of rendering is the conversion of this spectrum to red, green, blue intensities, which can be set in the frame buffer, and ultimately on the display.

However, in many cases, we use an approximation. We assume that light sources emit light directly on the wavelengths of the red, green, blue. Thus, we can immediately get the r,g,b, values without any integration. Note, however, that the rendering process is not linear since the products of radiance values and BRDFs are computed, so this simpler option is just an approximation.

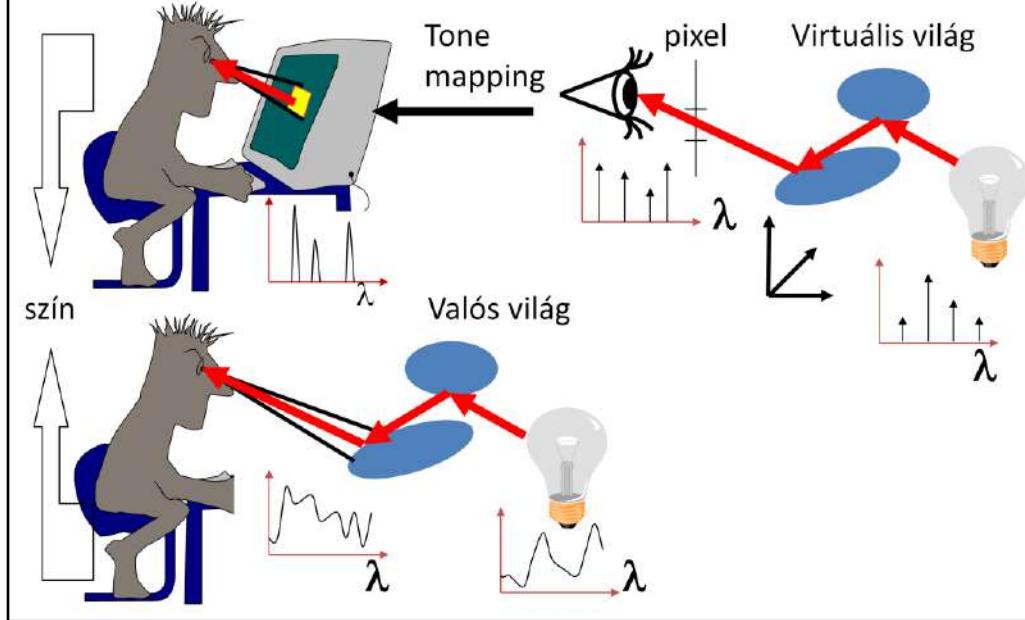
*"Science is either physics or stamp collecting."*  
*Ernest Rutherford*

## 3D képszintézis fizikai alapmodellje

Szirmay-Kalos László



## 3D képszintézis



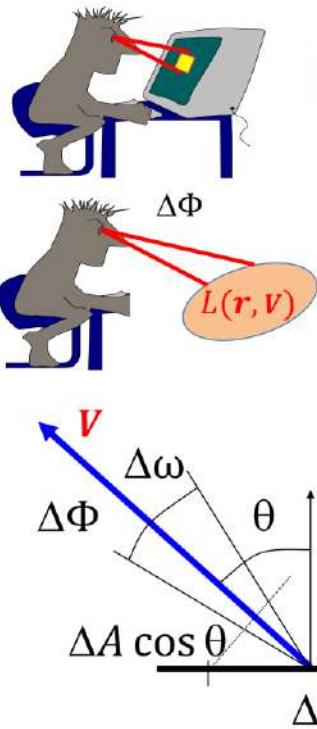
In order to compute the image, the power arriving at the eye from the solid angle of each pixel needs to be determined on different wavelengths.

We establish a virtual world model in the computer memory, where the user is represented by a single eye position and the display by a window rectangle. Then we compute the power going through the pixel toward the eye on different wavelengths, which results in a power spectrum.

If we can get the display to emit the same photons (i.e. the same number and of the same frequency), then the illusion of watching the virtual world can be created. Note that the human eye is sensitive only to the number and frequency of photons, but is not to the source of the photons, we would not be able to distinguish the real world from a virtual one shown on the computer screen.

As the human eye can be cheated with red, green, and blue colors, it is enough if the display emits light on these wavelengths. The last step of rendering is the conversion of the calculated spectrum to displayable red, green and blue intensities, which is called tone mapping. If we compute the light transfer only on these wavelengths, then this step can be omitted and the resulting spectrum can be used directly to control the monitor.

One crucial question is what exactly should be computed that describes the strength of the light intensity and when the pixel is controlled accordingly, provides the same color perception as the surface. Note that the pixel is at a different distance than the visible surface. The orientations of the display surface and of the visible surface are also different. The total emitted power would definitely be not good since it would mean less photons for the eye for farther sources.



## Radiancia (sugársűrűség)

Egységnnyi vetített terület által egységnnyi térszögbe sugárzott teljesítmény [Watt/sr/m<sup>2</sup>]:

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

We should work with power density instead of the power, that is computed with respect to the solid angle in which the light is emitted and with respect to the size of the projected surface. The density computed as the power divided by the projected surface and the solid angle of emission is called the radiance.

Experience and an important theorem state that if two surfaces have the same radiance, then they look identical no matter whether they are at a different distance or have different orientation. The proof is based on that if in a solid angle the eye would gather the same number of photons, i.e. energy, then it would not be able to distinguish the source surfaces. Let us compute this power for two surfaces that are seen in the same solid angle and have the same radiance.

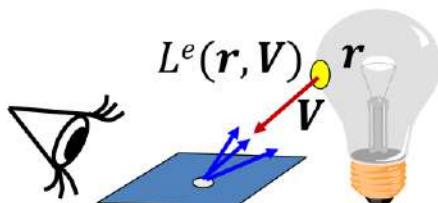
If the surface is closer, then its real area is smaller, but the solid angle in which the pupil of the eye can be reached from this surface is larger. Both the solid angle and the surface changes with the square of the distance and the two factors compensate each other. If the surface is not perpendicular to the viewing direction, then the surface seen in a given solid angle is larger, but the cosine factor will be proportionally smaller, so again we see no difference.

So, the conclusion is that we should compute the radiance of a surface and set the pixel of the display to have the same radiance. Then the two surfaces will be identical for the eye.

The fact that surfaces having the same radiance but at different distances look similar can also be interpreted as that the radiance does not change along a ray.

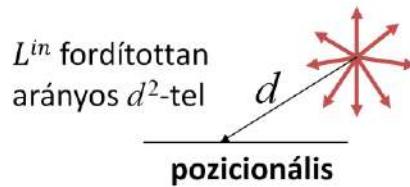
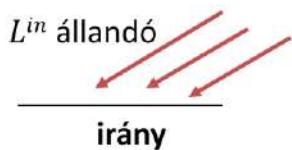
# Fényforrások

- Geometria+sugársűrűség:



- 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



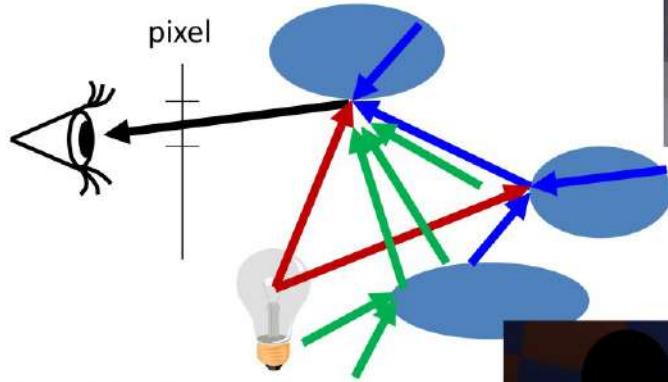
Real light sources are defined by their emission radiance,  $L^e$ . When the reflected radiance of a point is considered, the contribution of all those light source points should be added which are visible from the point of interest. This means integration. Thus, we often prefer abstract light source models, that can illuminate a surface just from a single direction, which saves integration.

In case of directional light sources, the radiance is constant everywhere, so is the illumination direction. In other words, the illumination rays are parallel. The Sun is an example for directional light source if we are on the Earth.

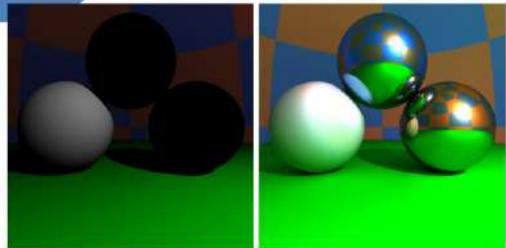
For point light sources, the illumination direction points from the location of the source to the illuminated point. The radiance decreases with the square of the distance.

If we ignore the dependence of the radiance on the distance, directional light sources can be considered as point sources being at infinity.

## Képszintézis



lokális illumináció  
rekurzív sugárkövetés  
globális illumináció



Rendering requires the determination of the surface that is visible through a pixel, then the computation of the radiance of this surface in the direction of the eye. There are different tradeoffs between accuracy of the light transport computation and the speed of the computation.

In the local illumination setting, when the radiance of a surface is calculated, we consider only the direct contribution of the light sources and ignore all indirect illumination.

In recursive ray tracing, indirect illumination is computed only for smooth surfaces, in the ideal reflection and refraction directions.

In the global illumination model, indirect illumination is taken into account for rough surfaces as well. In engineering applications we need global illumination solutions since only these provide predictable results. However, in games and real time systems, local illumination or recursive ray tracing will also be acceptable.

## Fénynél a hullámhosszok külön kezelhetők



- Relativisztikus tömeg és impulzus kicsi:  
 $m=E/c^2 = hf/c^2$ ,  $I=hf/c$
- A foton energia (hullámhossz) nem változik rugalmas ütközésnél
- Elnyelődési valószínűség energiafüggő

In computer graphics we consider photons in the visible wavelength range, roughly from 300 to 700 nanometer wavelengths. A photon has zero rest mass, otherwise it would not be able to fly with the speed of light. However, it has non-zero energy and impulse. The energy is proportional to frequency  $f$  of the light as stated by Einstein who invented this law when examining the photoelectric effect. He got his Nobel prize for this and not for the theory of relativity. Using the equivalence of the energy and mass, which was also published by Einstein as a short paper in 1905, we can assign a relativistic weight to the photon as the Planck constant  $h$  multiplied by the frequency and divided by the square of the speed of light.

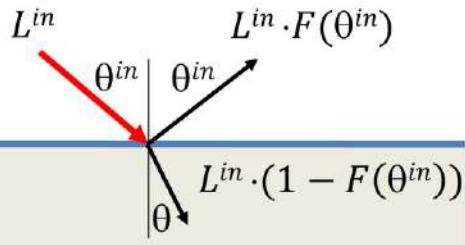
If  $f$  is small, then this relativistic mass is small. When photons meet a material, photons collide or scatter by the electrons or less probably with the core of atoms. For photons belonging to the visible spectrum, the relativistic mass of the photon is much smaller than the mass of the electron, thus a photon bounces off the electron like a ball bounces off from a rigid wall or a billiard ball bounces off from the edge of the table. If the collision is elastic, then the photon energy is preserved and the electron does not change its energy level.

If the collision is inelastic, then the energy of the photons is absorbed by the electron, this is the photoelectric effect, and the number of photons gets smaller. The probability of inelastic scattering, i.e. the albedo associated with a collision is energy dependent.

Summarizing when photons meet electrons, their number may get smaller but their energy level and consequently their frequency remain the same. This is the reason that in computer graphics wavelengths or frequencies are handled independently.



## Sima felület: Fresnel egyenlet



Snellius –  
Descartes:

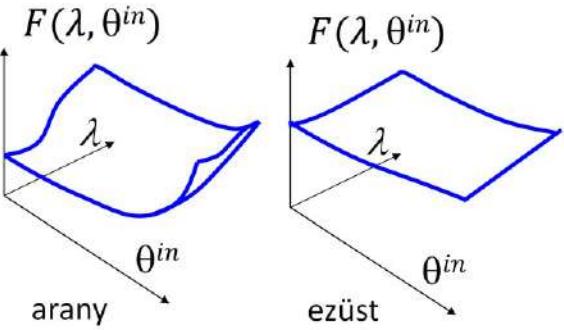
$$n = \frac{\sin \theta^{in}}{\sin \theta}$$

$n$ : a törésmutató, sebességarány  
 $\kappa$ : kioltási tényező

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

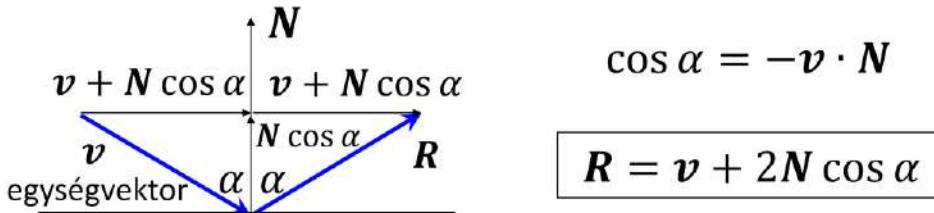
The simplest arrangement for the light transfer is a single plane that separates the space into two half spaces of different materials. According to the laws of geometric optics, the illumination ray is broken to a reflection ray meeting the reflection law and a refraction ray obeying the Snell's law of refraction. Here  $n$  is the index of refraction, which expresses the ratio of speeds of light outside and inside the material. The Fresnel equations define the amount of reflected energy (i.e. the probability that a photon is reflected). The Fresnel function can be calculated from index of refraction  $n$ , extinction  $k$ , which describes how quickly the electromagnetic wave diminishes inside the material, incident angle  $\theta^{in}$  and refraction angle  $\theta$ . The extinction is negligible for non-metals. Here we show a simplified Fresnel term, which is accurate enough for our purposes.

## Fresnel függvény



The Fresnel function depends on the wavelength and on the incident angle. When we see an object, we can observe surfaces of many different orientations, so we perceive the Fresnel function as a whole. This is why we can distinguish gold from copper although both of them are yellow.

## Tükörirány



```
vec3 reflect(vec3 V, vec3 N) {
    return V - N * dot(N, V) * 2;
};
```

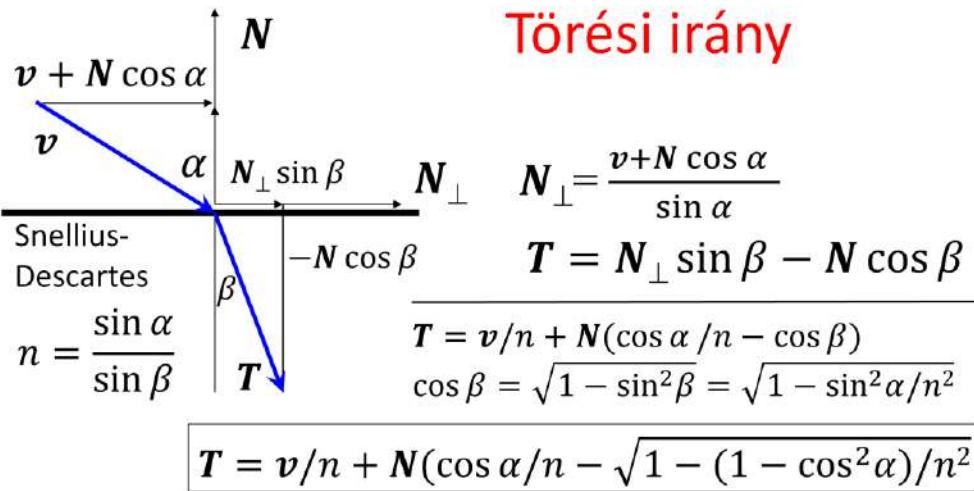
```
vec3 Fresnel(vec3 V, vec3 N) {
    float cosa = -dot(V, N);
    vec3 one(1, 1, 1);
    vec3 F0 = ((n-one)*(n-one) + kappa*kappa) /
        ((n+one)*(n+one) + kappa*kappa);
    return F0 + (one - F0) * pow(1-cosa, 5);
}
```

$$F_0 = \frac{(n-1)^2 + \kappa^2}{(n+1)^2 + \kappa^2}$$

To render smooth surfaces, we should compute the ideal reflection direction. Assume that incident direction  $v$  and surface normal  $N$  are unit length vectors.

Incident direction  $v$  is decomposed to a component parallel to the normal and a component that is perpendicular to it. Then, the reflection direction is built up from these two components.

For the amount of reflected light, we also need the Fresnel function for this direction, which is the direct implementation of our approximating formula.



```
vec3 refract(vec3 V, vec3 N, float ns) {
    float cosa = -dot(V, N);
    float disc = 1 - (1-cosa*cosa)/ns/ns; // scalar n
    if (disc < 0) return vec3(0, 0, 0);
    return V/ns + N * (cosa/ns - sqrt(disc));
}
```

The refraction direction calculation is also similar. The refraction direction  $v_t$  is expressed as a combination of the normal vector and a vector that is perpendicular to the normal,  $N_{\text{perpendicular}}$ . These vectors should be combined with weights  $\cos(\beta)$  and  $\sin(\beta)$  where  $\beta$  is the refraction angle.

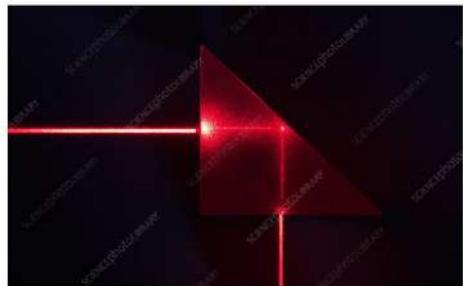
$N_{\text{perpendicular}}$  is expressed from  $v+N \cos(\alpha)$  by dividing it with its length  $\sin(\alpha)$ .

Then  $\sin(\beta)/\sin(\alpha)$  is replaced by the reciprocal of the index of refraction. Function  $\cos(\beta)$  is expressed with  $\sin(\beta)$ , for which we apply the same trick.

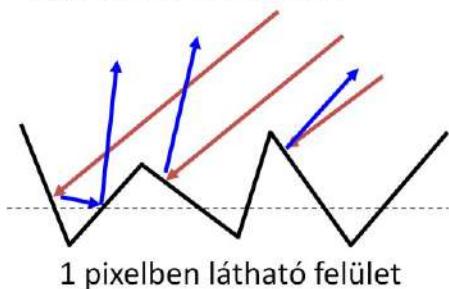
Finally,  $\sin(\alpha)$  is expressed from  $\cos(\alpha)$ , which is available as the dot product of the surface normal and the incident direction.

The final formula for refraction direction  $T$  involves a square root, where the discriminator in the square root can be negative if relative refraction index  $n$  is less than 1. This relative refraction index is less than 1 if the ray arrives at the boundary from the optically denser medium, for example, when the ray is originally in water and meets the surface separating water from air. Negative discriminator indicates that there is no refraction direction because the light gets fully reflected. This situation is called the total internal reflection and the threshold angle when it first occurs is the Brewster angle. The picture in the left upper corner shows a swimmer (László Cseh, olympic silver medalist) who is below the water while the camera is also at the bottom of the pool. The water surface acts as a perfect mirror reflecting the back of the swimmer and also the bottom of the pool.

## Teljes belső visszaverődés

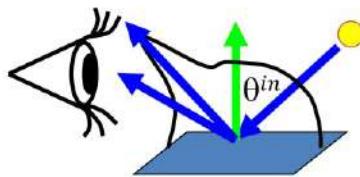


Mikroszkópikus modell:

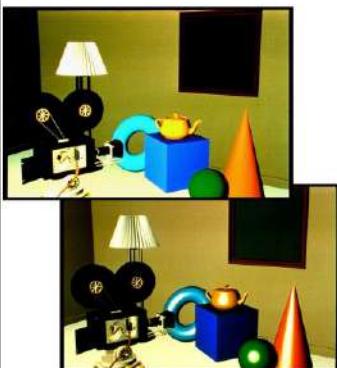


1 pixelben látható felület

## Rücskös felületek



Mi: viselkedésileg érvényes modell



Surfaces are usually not perfectly smooth, so they reflect light not just in the ideal reflection direction but practically in all possible directions. On microscopic scale, we can imagine these rough surfaces as a random collection of ideal mirror microfacets that reflect light according to their random orientation.

As we see not a single microfacet in a pixel, but a large collection of them, we perceive the average radiance reflected by this collection.

Photons may have a single scattering on these microfacets when the average is maximum around the ideal reflection direction of the mean surface. On the other hand, photons may get scattered multiple times, when they “forget” their original direction, so the reflection lobe will be roughly uniform.

Instead of following a probabilistic reasoning, for the sake of simplicity, we handle these rough surfaces as a black-box, i.e. empirical model. That is, we describe the behavior of the surface based on everyday experience without any structural analysis. By experience, we say that a rough surface reflects light into all directions, but more light is reflected into the neighborhood of the ideal reflection direction.

## Fény-felület kölcsönhatás

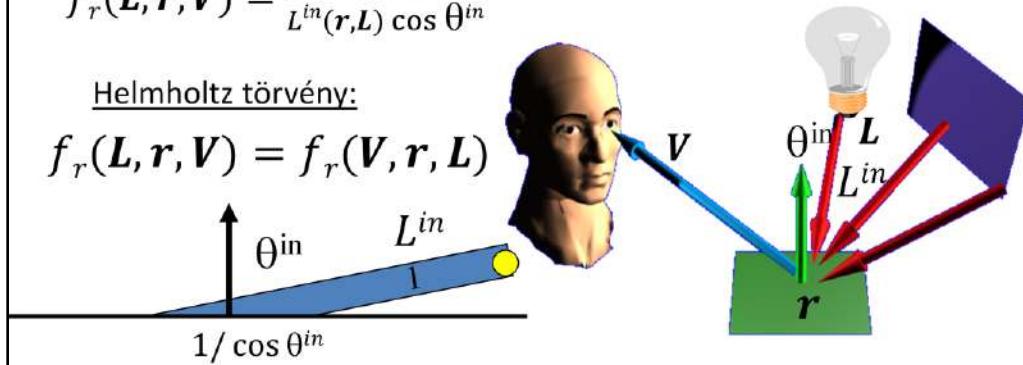
Radiancia = Irradiancia · BRDF

$$L(\mathbf{r}, \mathbf{V}) = \underbrace{L^{in}(\mathbf{r}, \mathbf{L}) \cdot \cos \theta^{in}}_{\text{Irradiancia}} \cdot f_r(\mathbf{L}, \mathbf{r}, \mathbf{V}) \quad \text{BRDF}$$

$$f_r(\mathbf{L}, \mathbf{r}, \mathbf{V}) \stackrel{\text{def}}{=} \frac{L(\mathbf{r}, \mathbf{V})}{L^{in}(\mathbf{r}, \mathbf{L}) \cos \theta^{in}} \quad \mathbf{L} - \text{ból } \mathbf{V}-\text{be vert arány}$$

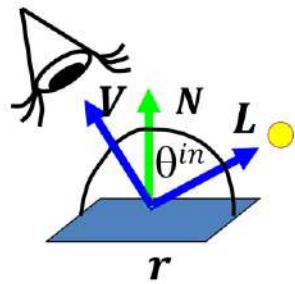
Helmholtz törvény:

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



The reflected radiance of a surface depends on the irradiance and the likelihood of the reflection. The irradiance is the incident radiance and a geometric factor that expresses that the illumination is weaker if the light arrives from a non-perpendicular direction since a unit cross section light beam illuminates a larger surface on which the photons are distributed. The likelihood of reflection, or the ration of the reflected radiance and the incident irradiance is expressed by the Bi-directional Reflectance Distribution Function (BRDF)  $f_r$ . In real life, BRDFs are symmetric, which is referred to as the Helmholtz's reciprocity principle. However, in computer graphics, the BRDF is just a formula, so we can work with non-symmetric BRDFs as well, but these cannot exist in real life.

## Diffúz visszaverődés



- Radiancia = Irradiancia · BRDF
- A nézeti iránytól független
- A BRDF a nézeti iránytól független
- Helmholtz: a BRDF a megvilágítási iránytól is független
- A BRDF irányfüggetlen:  
$$f_r(\mathbf{L}, \mathbf{r}, \mathbf{V}) = k_d(\mathbf{r}, \lambda)$$
- Diffúz visszaverődés = nagyon rücskös
  - sokszoros fény-anyag kölcsönhatás
  - színes!

Our first model is for very rough surfaces where all photons get reflected multiple times. Such materials (snow, sand, wall, chalk, cloth etc.) have a matte look, they look the same from all viewing directions. Thus, the radiance, which equals to the incident radiance times the BRDF times the geometry factor, is independent of the viewing direction. Incident radiance and the geometry term are already independent of the viewing direction, thus the BRDF must also be independent of the viewing direction. According to Helmholtz reciprocity, if the BRDF is independent of the viewing direction, it must be independent of the illumination direction as well, so the BRDF is direction independent.

Diffuse surfaces correspond to very rough surfaces where a photon collides many times. The Fresnel depends on the wavelength, which is strong for metals and weak for non-metals. Even if a single reflection changes the spectrum just a little, multiple reflections amplify this effect, so the final reflected light will have a modified spectrum. Diffuse reflection is primarily responsible for the “own color” of the surface.



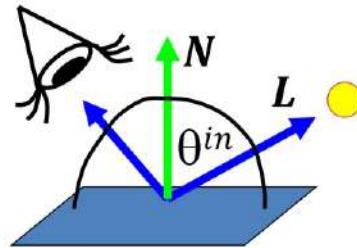
## (Johann Heinrich) Lambert törvény

Pont/irány fényforrásra válasz

BRDF irányfüggetlen, DE a sugársűrűség függ a megvilágítási iránytól

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

$$\cos \theta^{in} = \mathbf{N} \cdot \mathbf{L}$$



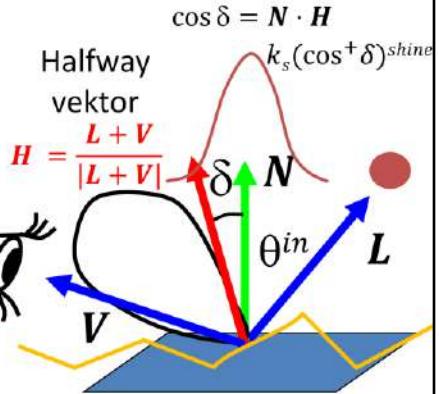
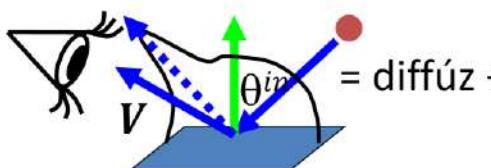
The reflected radiance is the incident radiance times the BRDF, which is constant now, and the geometry factor. So for diffuse surfaces, the reflected radiance is proportional to the cosine of the orientation angle. This cosine can be computed as a dot product of the unit surface normal and the unit illumination direction.

If the cosine is negative, i.e. the angle between the surface normal and the illumination direction is greater than 90 degrees, then the illumination is blocked by the object whose surface is considered. In such cases, the negative value is replaced by zero.



## Spekuláris visszaverődés (Phong (Bui) - (Jim) Blinn modell)

Nézeti irányfüggő



$$\begin{aligned}
 L^{ref} &= L^{in} k_d \cos^+ \theta^{in} + L^{in} k_s (\cos^+ \delta)^{\text{shine}} \\
 &= L^{in} \left( k_d + k_s \frac{(\cos^+ \delta)^{\text{shine}}}{\cos^+ \theta^{in}} \right) \cos^+ \theta^{in}
 \end{aligned}$$

Shiny, glossy or specular surfaces also reflect the light in all possible directions, but they look differently from different viewing directions. We can observe the blurred reflection of the light sources, thus they reflect more light close to the ideal reflection direction.

We model such surfaces as a combination of diffuse reflection where the radiance is constant and a specular reflection where the radiance is great around the ideal reflection direction. According to the microfacet model, diffuse reflection is caused by multiple light microfacet interaction while specular reflection is the result of a single light microfacet interaction. In order to model the specular reflection lobe, we need a function that is maximum at the reflection direction and decreases in a controllable way if the viewing direction gets farther from the reflection direction. Phong and Blinn proposed the  $\cos^{\text{shininess}}(\delta)$  function where delta is the angle between the macroscopic surface normal and the microfacet normal. The shininess exponent defines how shiny the surface is.

## Phong-Blinn modell nem tökéletes

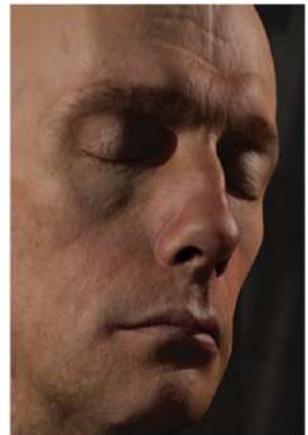
- Még a K épületet sem tudja éjszaka (aranyhíd)



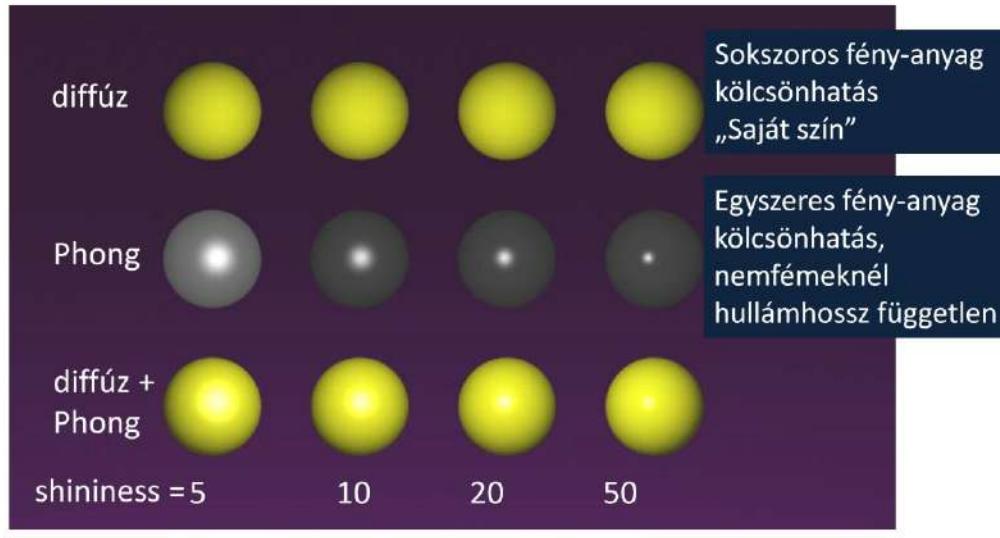
- Nem szimmetrikus (séríti a fizikát)

Pl. javítás:

$$L^{ref} = L^{in} \left( k_d + k_s \frac{(\cos^+ \delta)^{shine}}{(\mathbf{L} + \mathbf{V})^2} \right) \cos^+ \theta^{in}$$

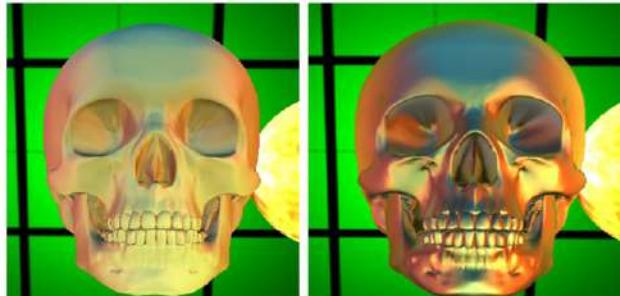


## Diffúz+Phong anyagok



Diffuse reflection simulates multiple light-surface interaction and is colored. Specular reflection is the model of the single light-surface interaction and it is proportional to the Fresnel function. For non metals, the wavelength dependence of the Fresnel is moderate, so for non metals the specular reflection is said to be "white".

## Diffúz + spekuláris visszaverődés



```
vec3 shade(vec3 N, vec3 V, vec3 L, vec3 inRad) {
    float cosTheta = dot(N, L); // unit vecs
    if(cosTheta < 0) return vec3(0,0,0); // self shadow
    vec3 diffuseRad = inRad * kd * cosTheta; // diffuse
    vec3 H = normalize(V + L);
    float cosDelta = dot(N, H);
    if(cosDelta < 0) return diffuseRad;
    return diffuseRad + inRad * ks * pow(cosDelta, shine);
}
```

To simulate a rough material, we need a **shade** function that computes the reflected radiance for incident radiance **inRad**, incident direction **lightDir**, surface normal vector **normal**, and view direction **viewDir**. Since the reflected and incident radiances are spectra, we use **vec3** type for them that can be different on the wavelength of red, green and blue. The function first computes the geometry factor **cosTheta**. If this geometry factor is negative, then the back of the surface is illuminated, so the object itself occludes the light source, so the reflected radiance is zero. Then, the diffuse radiance is computed, which is increased by the glossy term if **cosDelta** is not negative.

## Fényelnyelő



Diagram showing a cross-section of a medium with area  $A=1$ . A yellow arrow represents a photon moving through the medium. Blue circles represent particles. Arrows indicate interactions between photons and particles.

$$ds$$
$$A=1$$
$$L(s + ds) = L(s) - L(s)\sigma(s)ds$$
$$\frac{dL(s)}{ds} = -L(s)\sigma(s)$$
$$\int_{L(0)}^{L(S)} \frac{dL}{L} = - \int_0^S \sigma(s)ds$$
$$\ln(L(S)) - \ln(L(0)) = - \int_0^S \sigma(s)ds$$

$\sigma(s)ds = P\{\text{ütközés}\}$

Hatáskeresztmetszet,  
alias kioltási tényező  
[1/m]: egységfüggő!

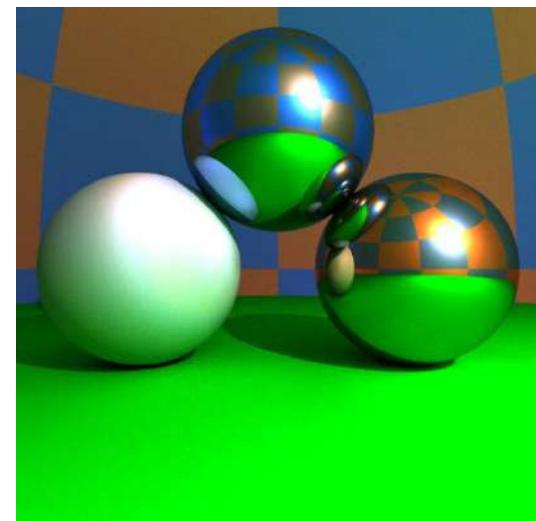
$$L(S) = L(0)e^{- \int_0^S \sigma(s)ds}$$
$$\approx L(0)e^{- \sum_i \sigma(s_i)\Delta s}$$

*“As technology advances, the rendering time remains constant.”*

*Jim Blinn*

# Sugárkövetés: ray-casting, ray-tracing, path-tracing

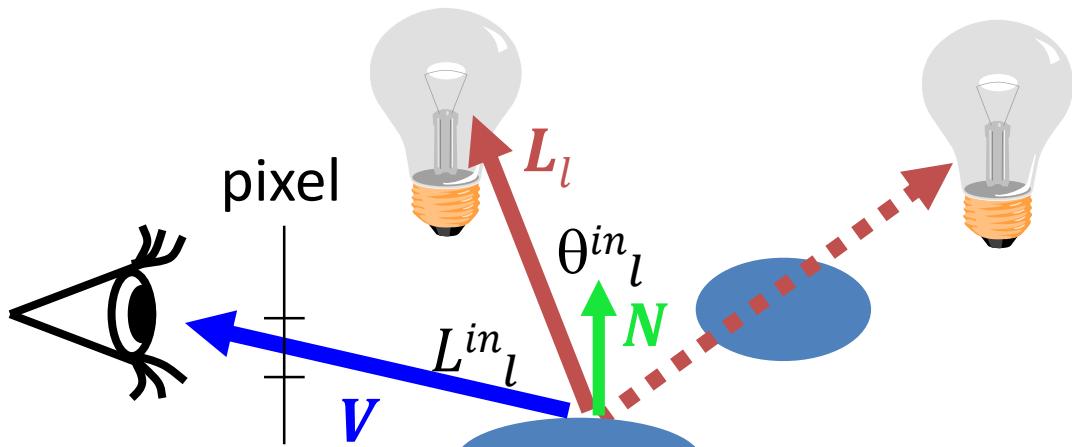
Szirmay-Kalos László



# Lokális illumináció: rücskös felületek, absztrakt fényforrások

Csak absztrakt  
fényforrások  
direkt megvilágítása

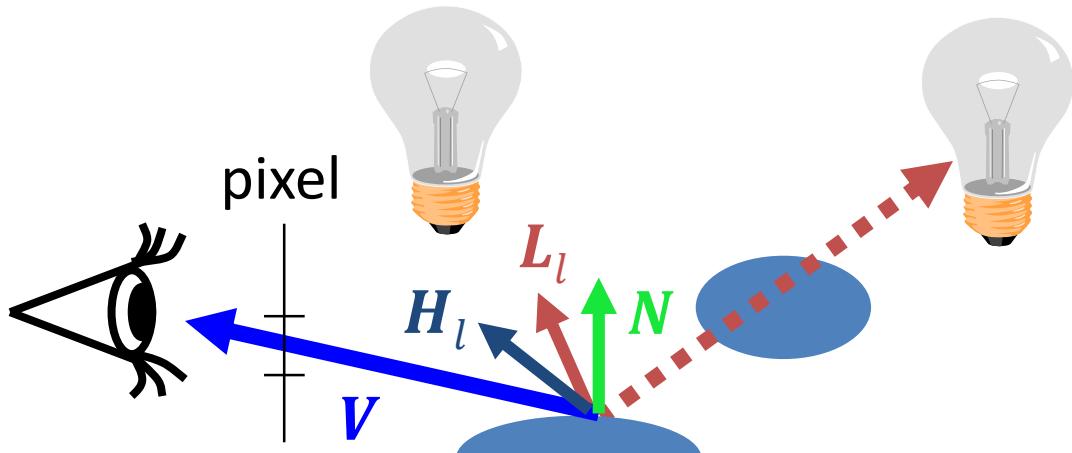
$$L(\mathbf{V}) \approx \sum_l L^{in}_l * f_r(\mathbf{L}_l, \mathbf{N}, \mathbf{V}) \cdot \cos^+ \theta^{in}_l$$



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

# Lokális illumináció: rücskös felületek, absztrakt fényforrások

Csak absztrakt  
fényforrások  
direkt megvilágítása



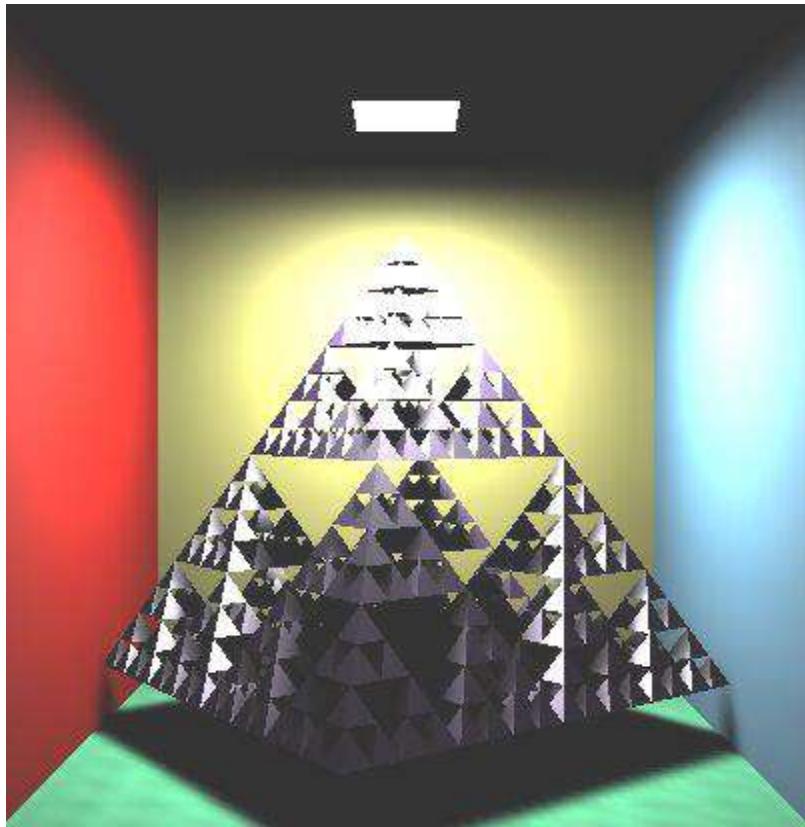
$$L(\mathbf{V}) \approx \sum_l L^{in}_l * \{k_d \cdot (\mathbf{L}_l \bullet \mathbf{N})^+ + k_s \cdot ((\mathbf{H}_l \bullet \mathbf{N})^+)^{shine}\}$$

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

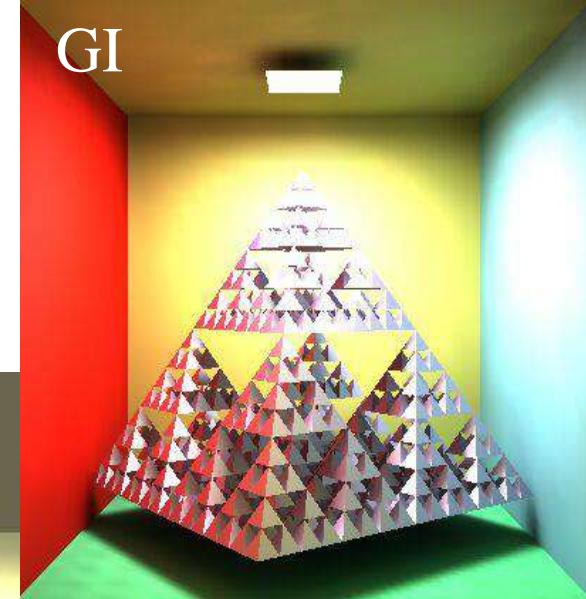
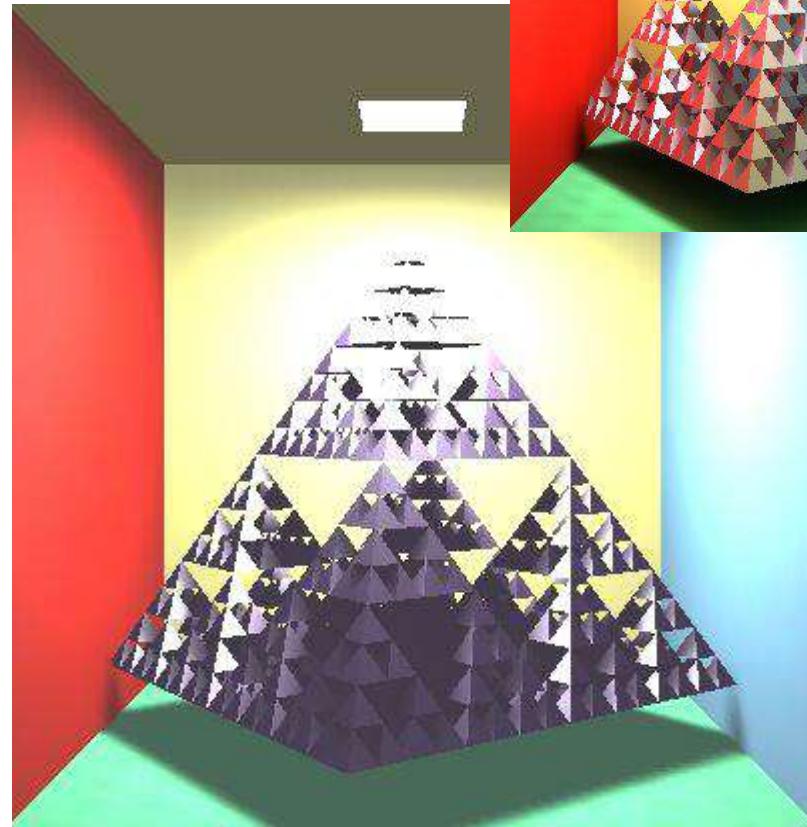
GI

# Ambiens tag

Lokális illumináció

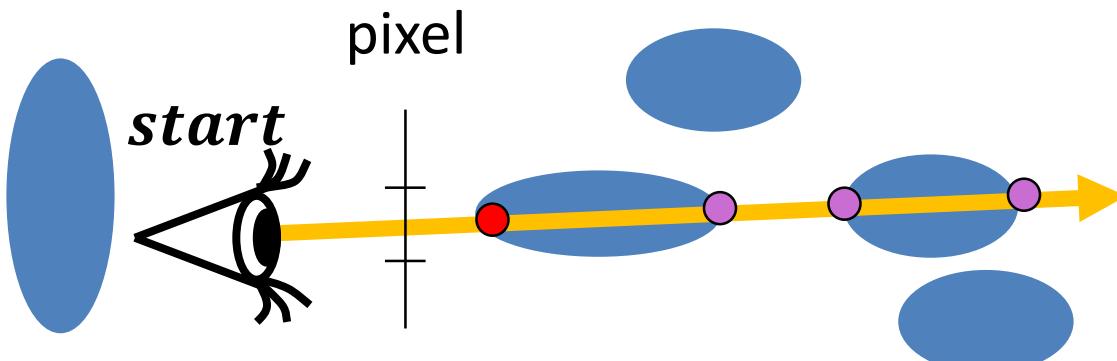


+ ambiens tag



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

# Láthatóság



$$ray(t) = start + dir \cdot t, \quad t > 0$$

```
struct Ray {  
    vec3 start;  
    vec3 dir; //egységvektor  
    bool out; //kívül? vagy ior  
};  
  
struct Hit {  
    float t;  
    vec3 position;  
    vec3 normal;  
    Material* material;  
    Hit() { t = -1; }  
};
```

```
Hit firstIntersect(Ray ray) {  
    Hit bestHit;  
    for(Intersectable * obj : objects) {  
        Hit hit = obj->intersect(ray); // hit.t<0 ha nincs metszés  
        if(hit.t > 0 && (bestHit.t < 0 || hit.t < bestHit.t))  
            bestHit = hit;  
    }  
    if (dot(ray.dir, bestHit.normal) > 0) bestHit.normal *= -1;  
    return bestHit;  
}
```

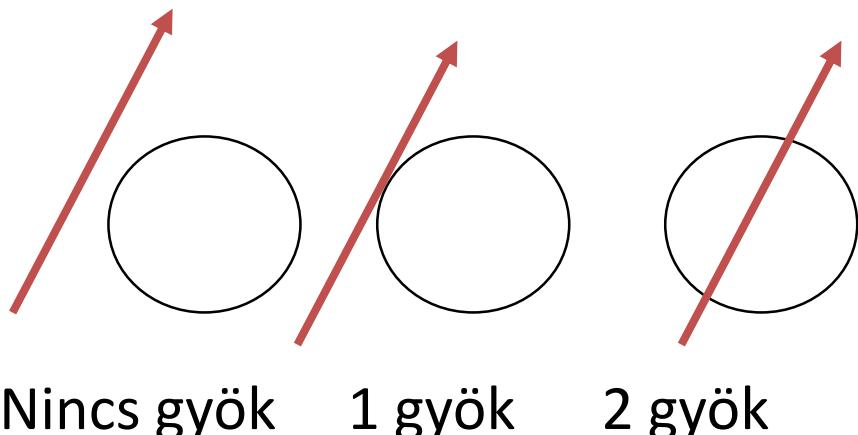
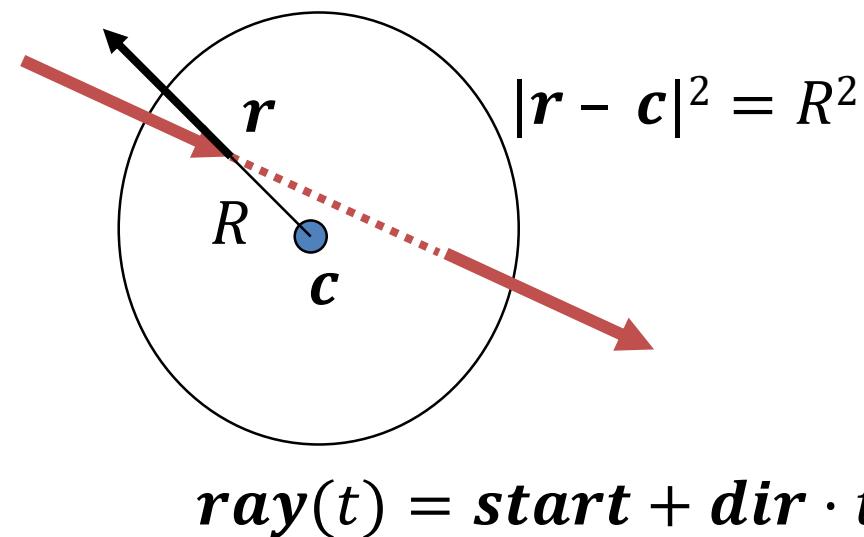
Nézzen felénk!

$$\text{if } (N \bullet dir > 0) \{ N = -N; \}$$

# Objektum = Intersectable

```
struct Intersectable {
    Material* material;
    virtual Hit intersect(const Ray& ray) = 0;
};
```

# Metszéspont számítás gömbbel



$$|ray(t) - c|^2 = (start + dir \cdot t - c) \cdot (start + dir \cdot t - c) = R^2$$

$$(dir \cdot dir)t^2 + 2((start - c) \cdot dir)t + (start - c) \cdot (start - c) - R^2 = 0$$

Wanted: a pozitív megoldások közül a kisebb

Felületi normális:  $N = (ray(t) - c)/R$

# Sphere mint Intersectable

```
class Sphere : public Intersectable {
    vec3 center;
    float radius;
public:
    Hit intersect(const Ray& ray) {
        Hit hit;
        vec3 dist = ray.start - center;
        float a = dot(ray.dir, ray.dir);
        float b = dot(dist, ray.dir) * 2;
        float c = dot(dist, dist) - radius * radius;
        float discr = b * b - 4 * a * c;
        if (discr < 0) return hit; else discr = sqrtf(discr);
        float t1 = (-b + discr)/2/a, t2 = (-b - discr)/2/a;
        if (t1 <= 0) return hit; // t1 >= t2 for sure
        hit.t = (t2 > 0) ? t2 : t1;
        hit.position = ray.start + ray.dir * hit.t;
        hit.normal = (hit.position - center)/radius;
        hit.material = material;
        return hit;
    }
};
```

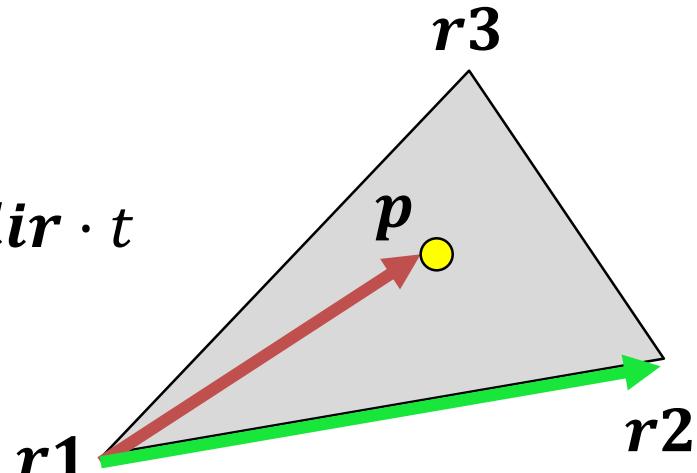
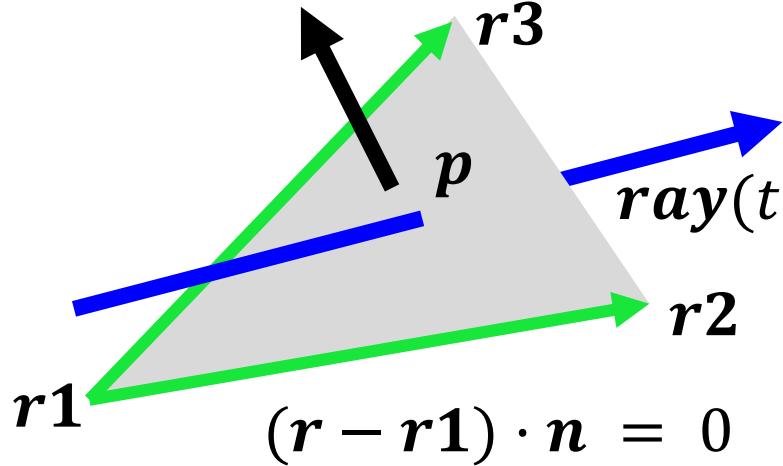
# Implicit felületek

- A felület pontjai:  $f(\mathbf{r}) = 0$
  - Sugár:  $\mathbf{ray}(t) = \mathbf{s} + \mathbf{d} \cdot t$
  - Metszés paraméter  $t^*$ :  $f(\mathbf{s} + \mathbf{d} \cdot t^*) = 0$
  - Metszéspont:  $\mathbf{r}^* = \mathbf{ray}(t^*) = \mathbf{s} + \mathbf{d} \cdot t^*$
  - Normálvektor =  $\nabla f(\mathbf{r}^*)$
- 

- Kvadratikus felületek példa:  $f(\mathbf{r}) = [\mathbf{r}, 1] \cdot \mathbf{Q} \cdot [\mathbf{r}, 1]^T = 0$
- Metszés paraméter:  
 $[\mathbf{s} + \mathbf{d} \cdot t^*, 1] \cdot \mathbf{Q} \cdot [\mathbf{s} + \mathbf{d} \cdot t^*, 1]^T = 0$   
 $[\mathbf{d}, 0] \cdot \mathbf{Q} \cdot [\mathbf{d}, 0]^T (t^*)^2 + 2[\mathbf{s}, 1] \cdot \mathbf{Q} \cdot [\mathbf{d}, 0]^T t^* + [\mathbf{s}, 1] \cdot \mathbf{Q} \cdot [\mathbf{s}, 1]^T = 0$
- Normálvektor:  $\nabla f(\mathbf{r}^*) = \mathbf{Q} \cdot [\mathbf{r}^*, 1]^T$  első három koord.

# Háromszög

$$\mathbf{n} = (\mathbf{r}_2 - \mathbf{r}_1) \times (\mathbf{r}_3 - \mathbf{r}_1)$$



1. Síkmetszés:  $(\mathbf{ray}(t) - \mathbf{r}_1) \cdot \mathbf{n} = 0$ ,  
 $t > 0$

$$t = \frac{(\mathbf{r}_1 - \mathbf{start}) \cdot \mathbf{n}}{\mathbf{dir} \cdot \mathbf{n}}$$

2. A metszéspont a háromszögön belül van-e?

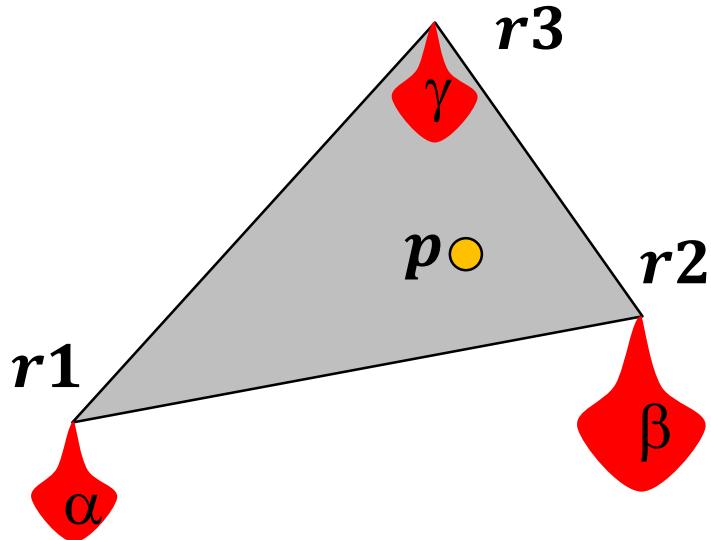
$$((\mathbf{r}_2 - \mathbf{r}_1) \times (\mathbf{p} - \mathbf{r}_1)) \cdot \mathbf{n} > 0$$

$$((\mathbf{r}_3 - \mathbf{r}_2) \times (\mathbf{p} - \mathbf{r}_2)) \cdot \mathbf{n} > 0$$

$$((\mathbf{r}_1 - \mathbf{r}_3) \times (\mathbf{p} - \mathbf{r}_3)) \cdot \mathbf{n} > 0$$

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

# Háromszög metszés baricentrikus koordinátákban



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

$$\begin{aligned} \alpha + \beta + \gamma &= 1 && \text{1 skalár egyenlet} \\ \alpha, \beta, \gamma &\geq 0 \end{aligned}$$

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

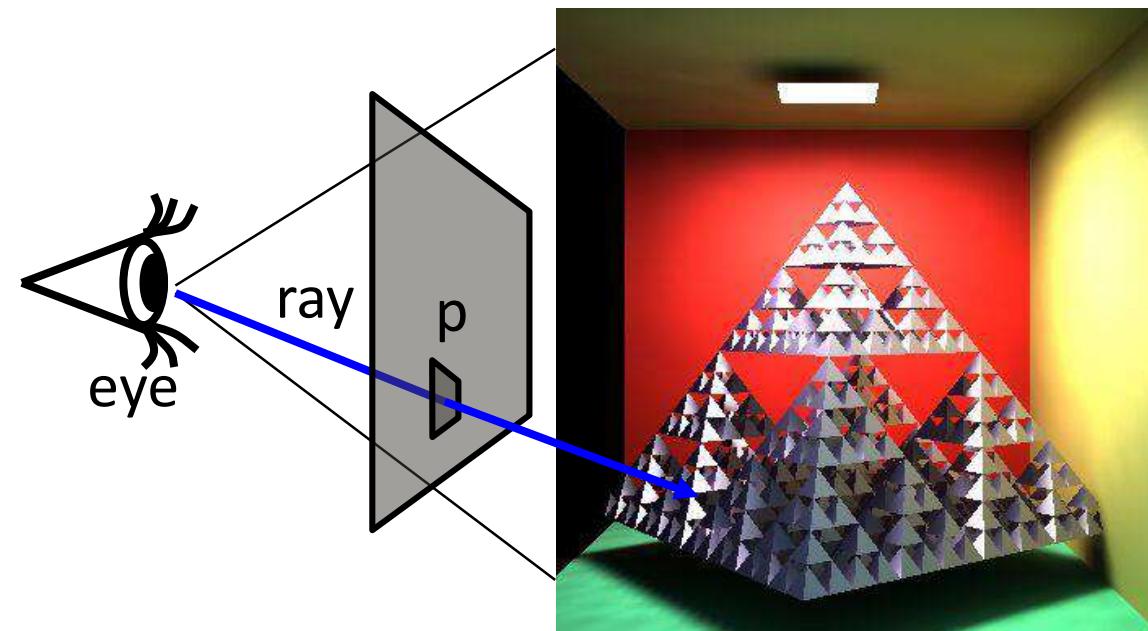
$$start + dir \cdot t = \alpha \cdot r1 + \beta \cdot r2 + \gamma \cdot r3 \quad \text{3 skalár egyenlet}$$

4 ismeretlen:  $\alpha, \beta, \gamma, t$

A megoldás után ellenőrzés, hogy minden nem negatív-e

# Sugárkövetés: Render

Virtuális világ: szem+ablak



Render( )

for each pixel  $p$

    Ray  $r = \text{getRay}( \text{eye} \rightarrow \text{pixel } p )$

    color = **trace(ray)**

    WritePixel( $p$ , color)

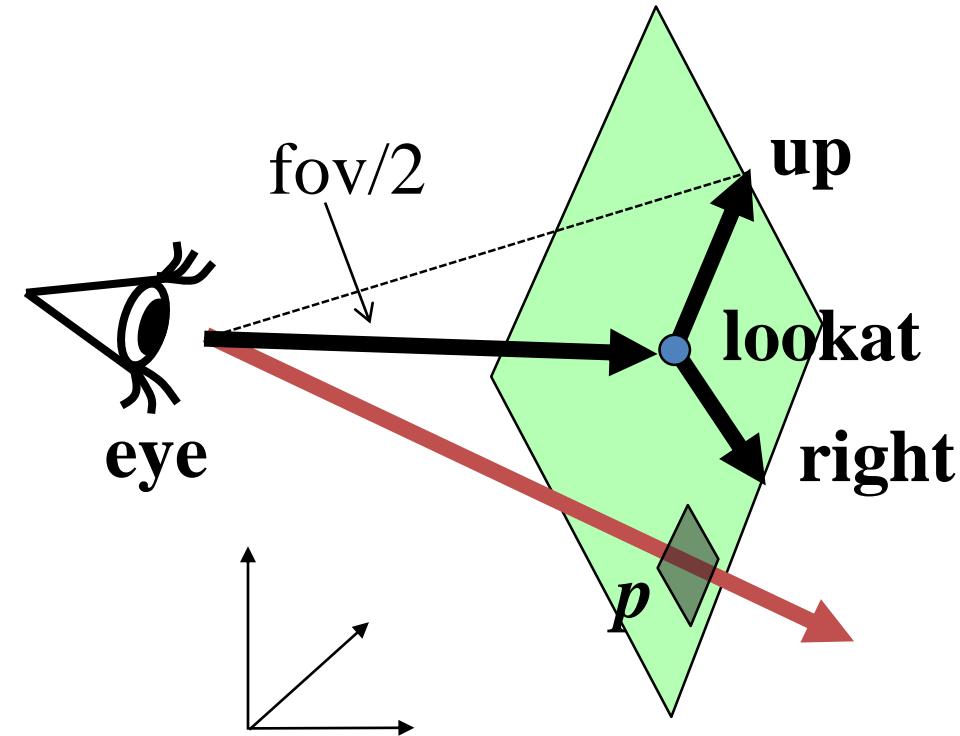
endfor

end

Valós világ: néző+képernyő



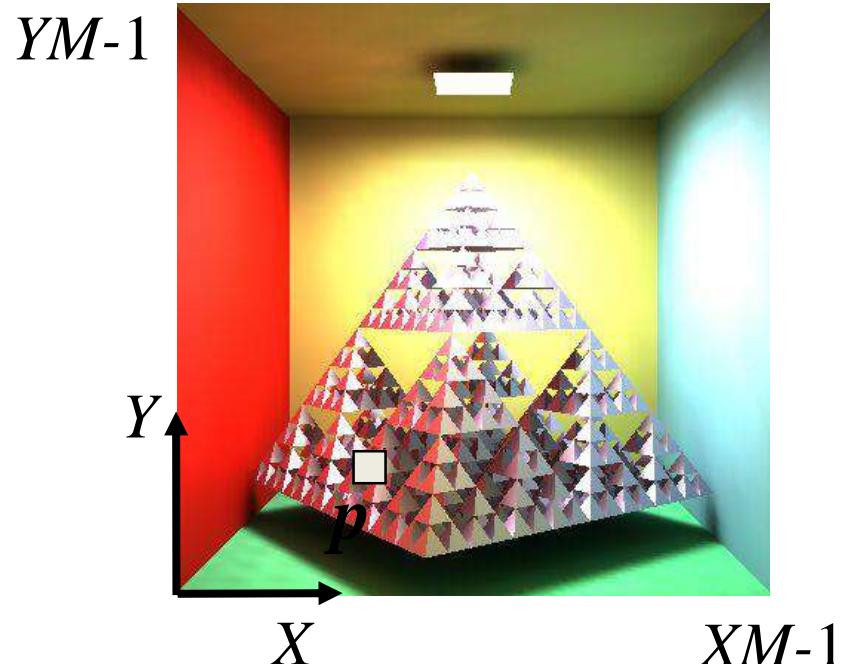
# Kamera: getRay



$$p = \text{lookat} + \alpha \cdot \text{right} + \beta \cdot \text{up},$$

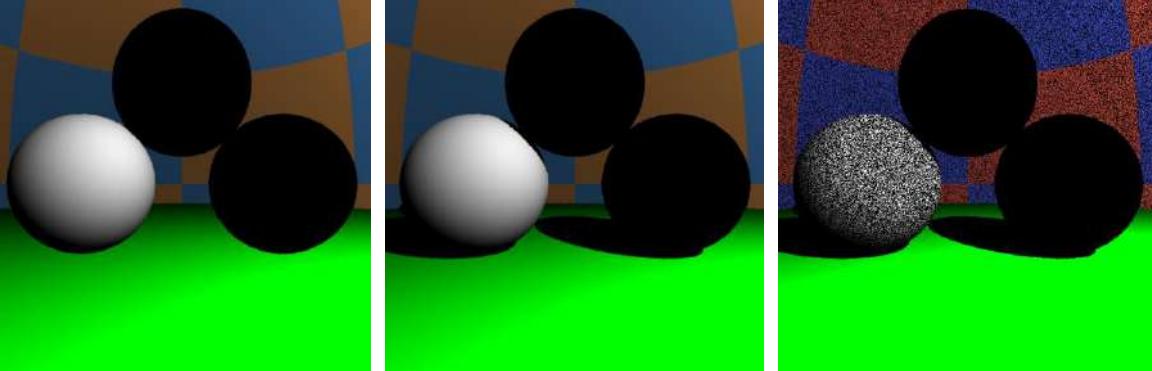
$$= \text{lookat} + (2(X+0.5)/XM-1) \cdot \text{right} + (2(Y+0.5)/YM-1) \cdot \text{up}$$

Ray: start = eye, dir = p - eye



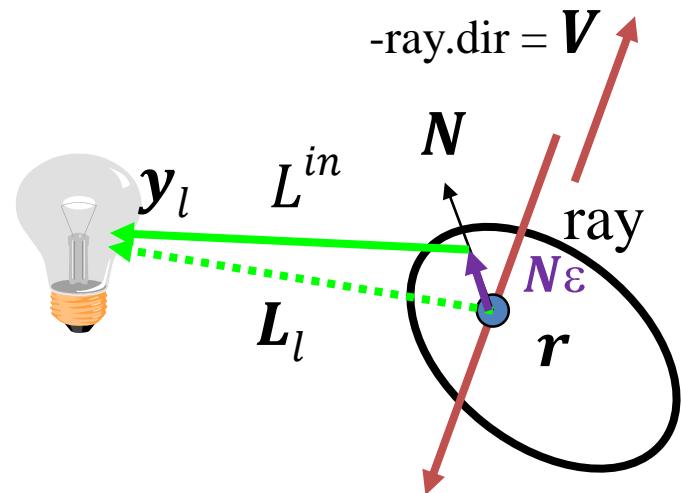
Normalizált eszköz koordináták

$\alpha, \beta$  in  $[-1,1]$



No Shadow

Shadow

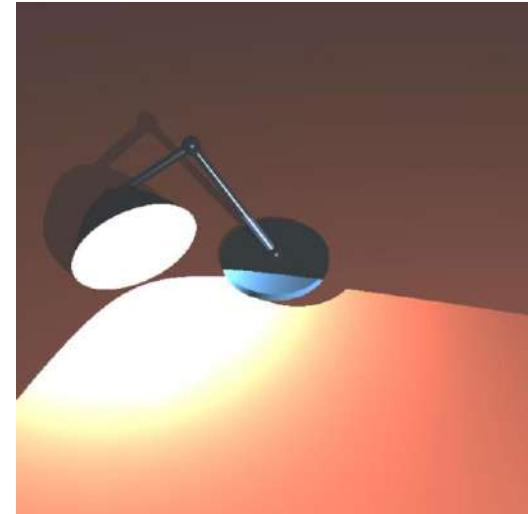
 $\varepsilon = 0$ : acne

```

DirectLight()
vec3 trace(Ray ray) {
    Hit hit = firstIntersect(ray);
    if(hit.t < 0) return La; // nothing
    [r, N, ka, kd, ks, shine] ← hit;
    vec3 outRad = ka * La;
    for(each light source l) {
        Ray shadowRay(r + Nε, Ll);
        Hit shadowHit = firstIntersect(shadowRay);
        if(shadowHit.t < 0 || shadowHit.t > |r - yl| )
            outRad +=  $L^{in}_l \cdot \{k_d \cdot (L_l \bullet N)^+ + k_s \cdot ((H_l \bullet N)^+)^{shine}\}$ 
    }
    return outRad;
}

```

## 2. házi: Luxo Grandpa



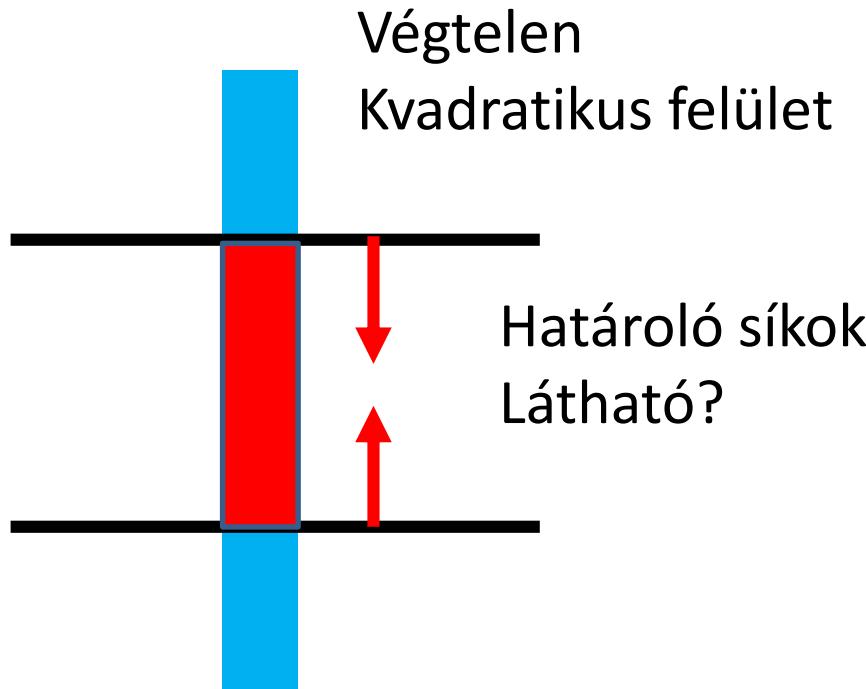
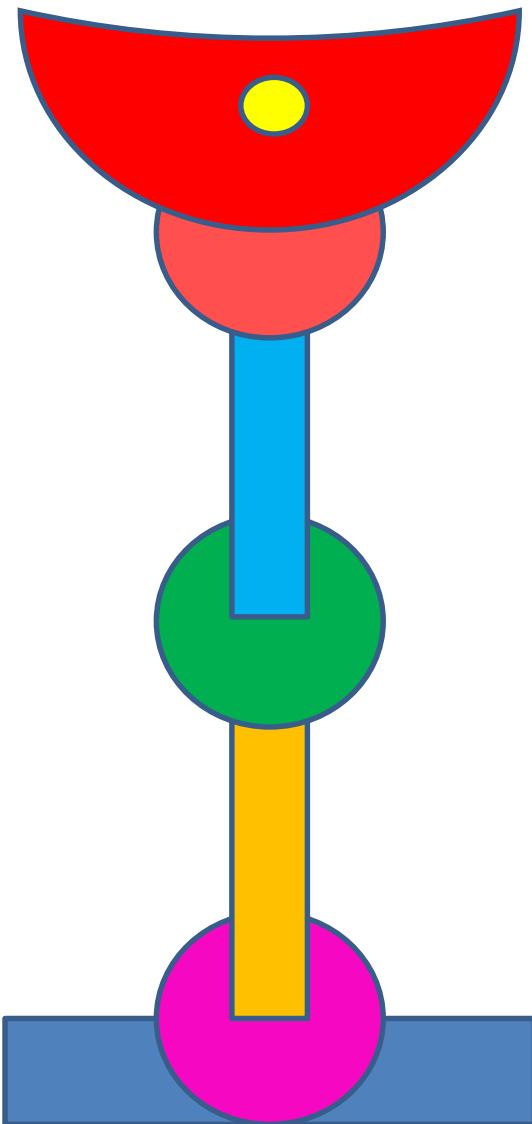
Luxo Junior-ból *Luxo Grandpa (LG)* lett, ezért új programot szentelünk neki.

LG egy síkon áll, a szerkezete (alulról felfelé): henger talp, gömbcsukló1, henger rúd1, gömbcsukló2, henger rúd2, gömbcsukló3, paraboloid, amelynek fókuszpontjában egy pontfényforrás ül.

A gömbcsuklók a koordinátatengelyektől eltérő tengelyek mentén folyamatosan elfordulnak. A szereplőket még egy pontfényforrás és ambiens fény világítja meg. A kamera forog LG körül.

A szereplők diffúz-spekuláris típusú rücskös anyagúak.

# Luxo Grandpa



Végtelen  
Kvadratikus felület

Hatóroló síkok  
Látható?

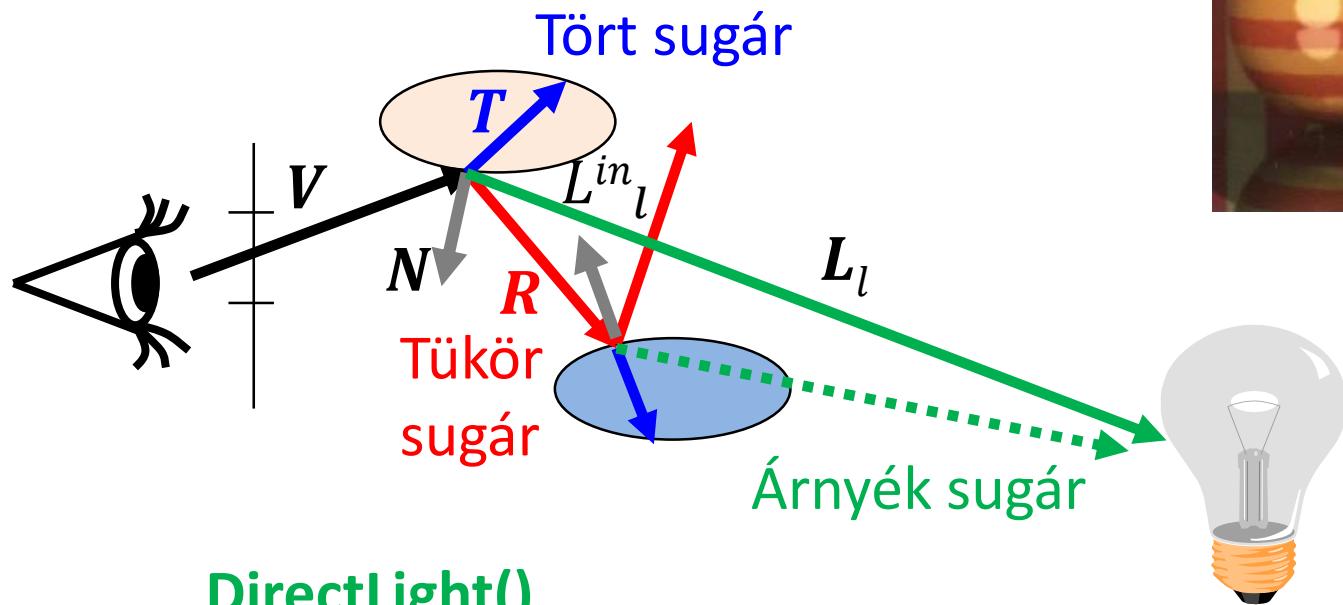
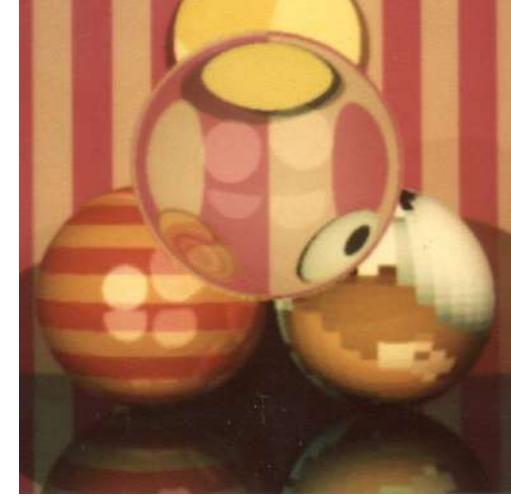
## Feladatok:

Kvadratikus felület és síkok

- Hierarchikus transzformálása
- Metszése
- Normálvektora

Fókuszpont hierarchikus transzformálása

# Rekurzív sugárkövetés



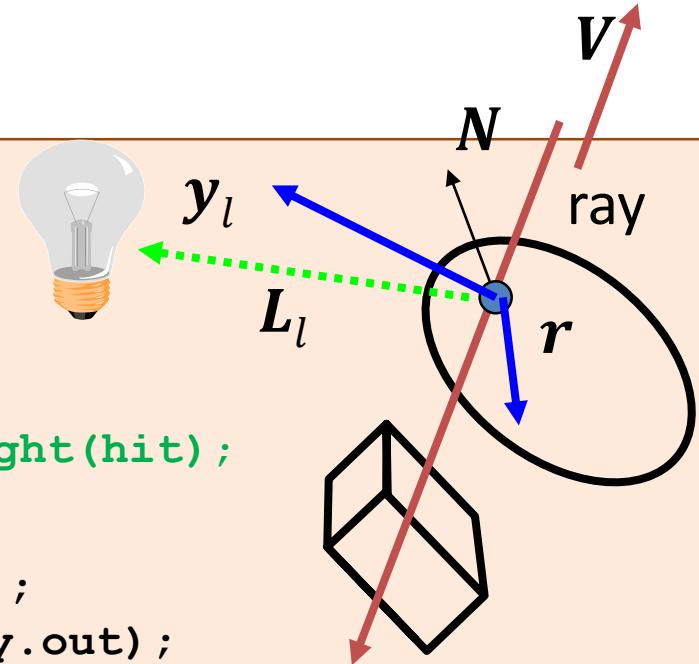
**DirectLight()**

$$L(V) \approx \frac{k_a * L_a + \sum_l L^{in}_l * \{k_d \cdot (L_l \bullet N)^+ + k_s \cdot ((H_l \bullet N)^+)^{shine}\}}{F(V \bullet N) * L^{in}(R) + (1 - F(V \bullet N)) * L^{in}(T)}$$

Fresnel  
 Tükör irányból érkező fény  
 1-Fresnel  
 Törési irányból érkező fény

# trace

```
vec3 trace(Ray ray) {  
  
    Hit hit = firstIntersect(ray);  
    if(hit.t < 0) return L_a; // nothing  
    vec3 outRad(0, 0, 0);  
    if(hit.material->rough) outRad = DirectLight(hit);  
  
    if(hit.material->reflective){  
        vec3 reflectionDir = reflect(ray.dir,N);  
        Ray reflectRay(r + Nε, reflectionDir, ray.out);  
        outRad += trace(reflectRay)*Fresnel(ray.dir,N);  
    }  
  
    if(hit.material->refractive) {  
        ior = (ray.out) ? n.x : 1/n.x;  
        vec3 refractionDir = refract(ray.dir,N,ior);  
        if (length(refractionDir) > 0) {  
            Ray refractRay(r - Nε, refractionDir, !ray.out);  
            outRad += trace(refractRay)*(vec3(1,1,1)-Fresnel(ray.dir,N));  
        }  
    }  
  
    return outRad;  
}
```

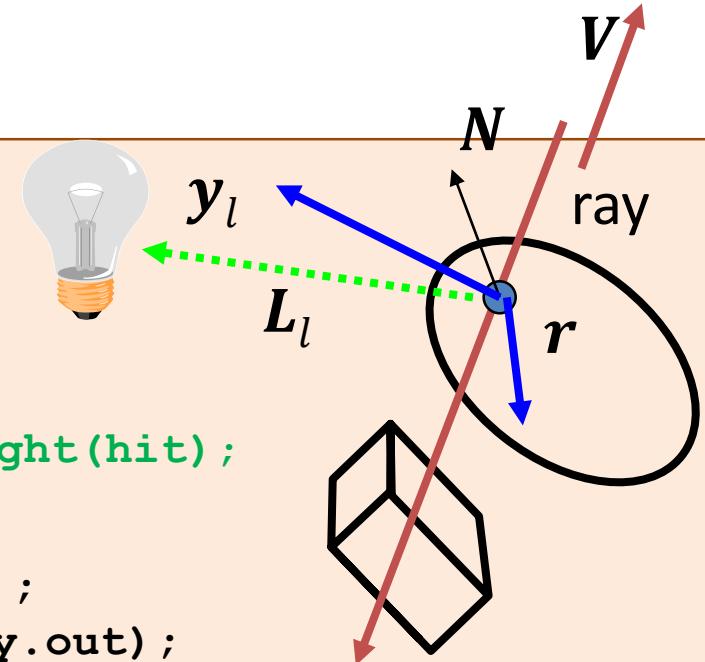


# trace

```
vec3 trace(Ray ray, int d=0) {
    if (d > maxdepth) return L_a;
    Hit hit = firstIntersect(ray);
    if(hit.t < 0) return L_a; // nothing
    vec3 outRad(0, 0, 0);
    if(hit.material->rough) outRad = DirectLight(hit);

    if(hit.material->reflective) {
        vec3 reflectionDir = reflect(ray.dir,N);
        Ray reflectRay(r + Nε, reflectionDir, ray.out);
        outRad += trace(reflectRay,d+1)*Fresnel(ray.dir,N);
    }

    if(hit.material->refractive) {
        ior = (ray.out) ? n.x : 1/n.x;
        vec3 refractionDir = refract(ray.dir,N,ior);
        if (length(refractionDir) > 0) {
            Ray refractRay(r - Nε, refractionDir, !ray.out);
            outRad += trace(refractRay,d+1)*(vec3(1,1,1)-Fresnel(ray.dir,N))
        }
    }
    return outRad;
}
```





# Paul Heckbert névjegye



```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02}; struct sphere{ vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9, .05,.2,.85,0.,1.7,-1.8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,1.,.8,.8, 1.,.3,.7,0.,.0,.1.2,3.,-6.,15.,1.,.8,1.,.7,0.,.0,0.,.6,1.5,-3.,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

## Material

n, kappa  
ka, kd, ks, shine  
reflective?  
refractive?  
rough?

reflect()  
refract()  
Fresnel ()  
shade()

## Intersectable

Hit *intersect()*

heterogeneous  
collection  
objects

## Scene

La  
build()  
render()  
firstIntersect()  
trace()

## Sphere

center, radius  
intersect()

## Mesh

intersect()

## Camera

eye, lookat,  
up, right,  
XM, YM  
getRay()

## Light

pos, Lout, type  
getLightDir()  
getInRad()  
getDist()

CPU

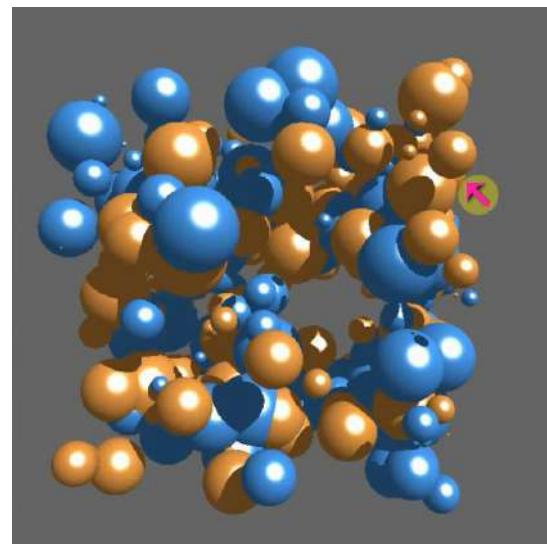
GPU



# Sugárkövetés

## 1. Program: Ray casting

Szirmay-Kalos László

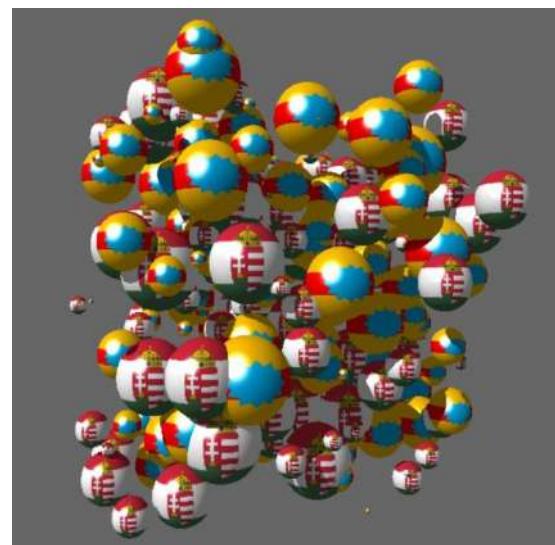




## Sugárkövetés

# 2. Program: Ray casting textúrázással

Szirmay-Kalos László

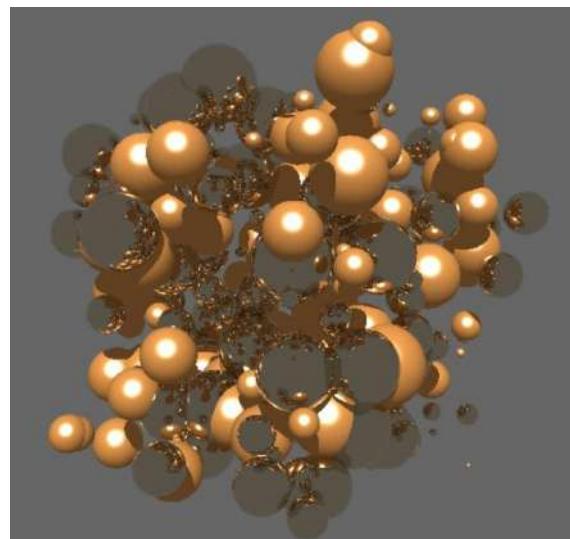




## Sugárkövetés

### 3. Program: Rekurzív sugárkövetés

Szirmay-Kalos László





## Sugárkövetés

# 4. Program: Napfénycső szimulátor

Szirmay-Kalos László

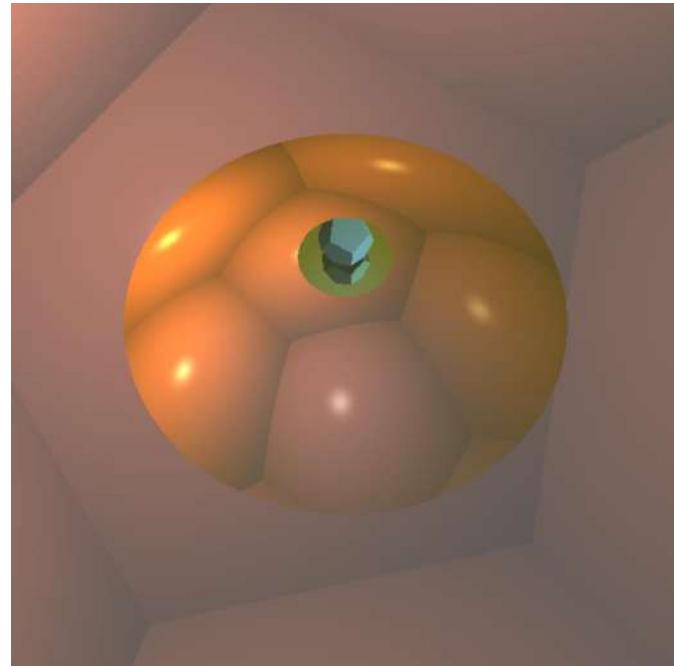




# GPU sugárkövetés

## 5. Program: Mirascope

Szirmay-Kalos László





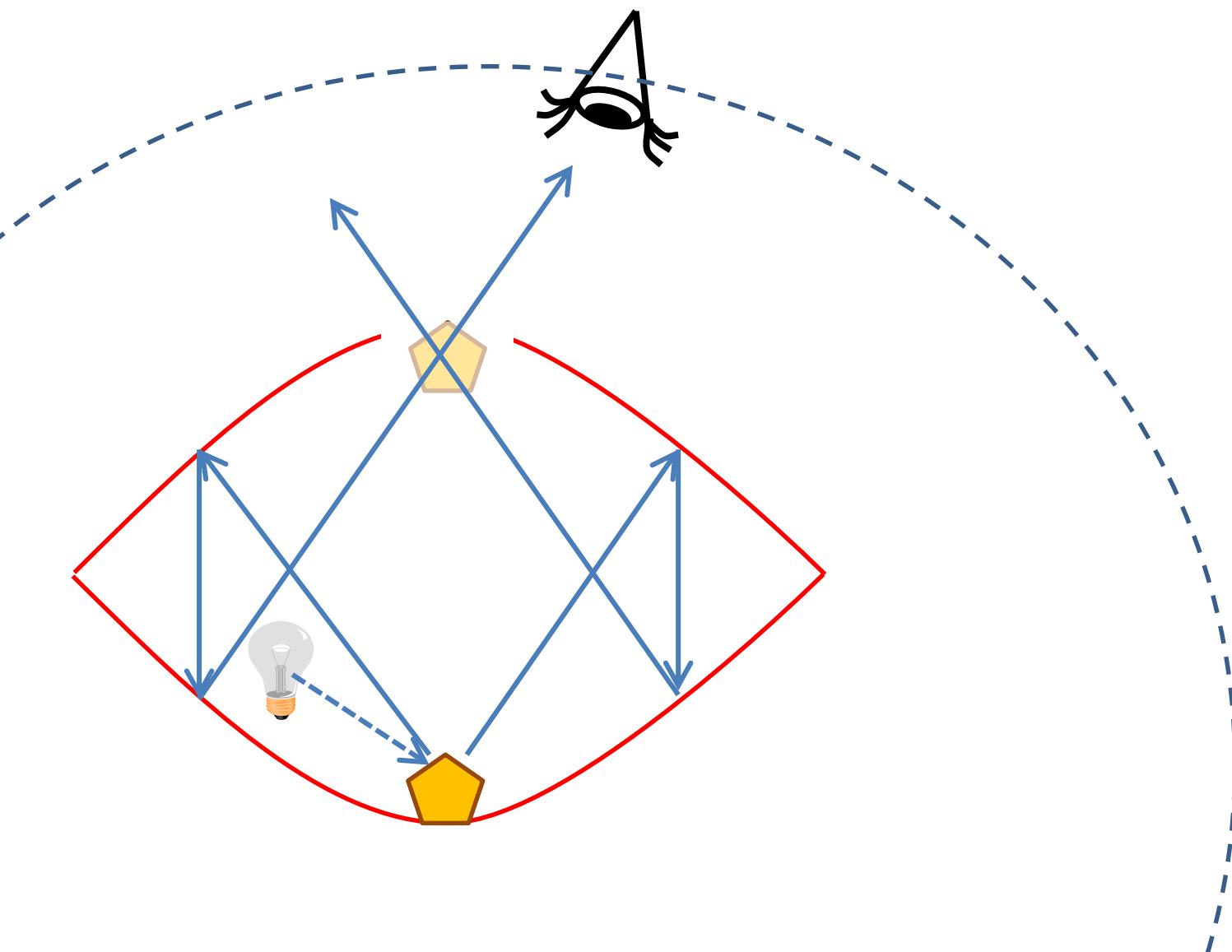
# Specifikáció

Készítsen **mirascope szimulátort**. A mirascope két ugyanolyan, egymásra tett és szembe fordított paraboloid tükrő, ahol az egyik fókuszpontja a másik aljára esik és a felsőn lyuk van. A mi mirascope-unk aranyból van és tükrő lévén optikailag sima. A mirascope az alsó aljára tett tárgyat a felső lyukban megjeleníti. A szimulátorban a tárgy diffúz-spekuláris anyagú dodekaéder, amit a mirascope belsejében lévő pont fényforrás és az egész térben jelen lévő ambiens forrás világít meg. A néző és a mirascope egy Platon-i szabályos test geometriájú szoba belsejében szemlélődik.

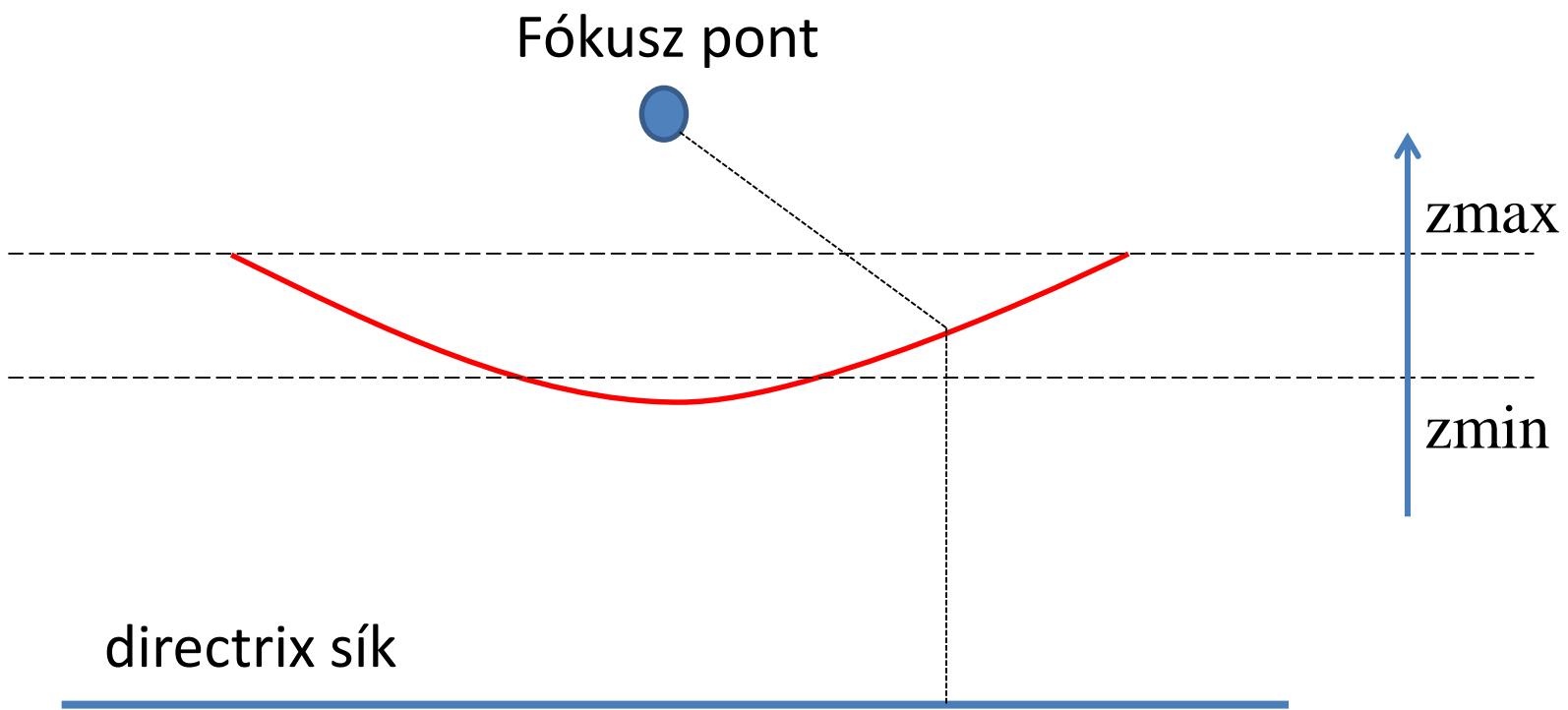
A virtuális kamerán kívülről, felülről és oldalról néz rá a mirascope-ra. Az 'f' lenyomásával a szempozíció feljebb kerül, az 'F' hatására lejjebb úgy, hogy atávolsága a mirascope aljától állandó legyen.

Az arany törésmutatója és kioltási tényezője az r,g,b hullámhosszokon:  
 $n/k: 0.17/3.1, 0.35/2.7, 1.5/1.9$

# Mirascope



# Paraboloid

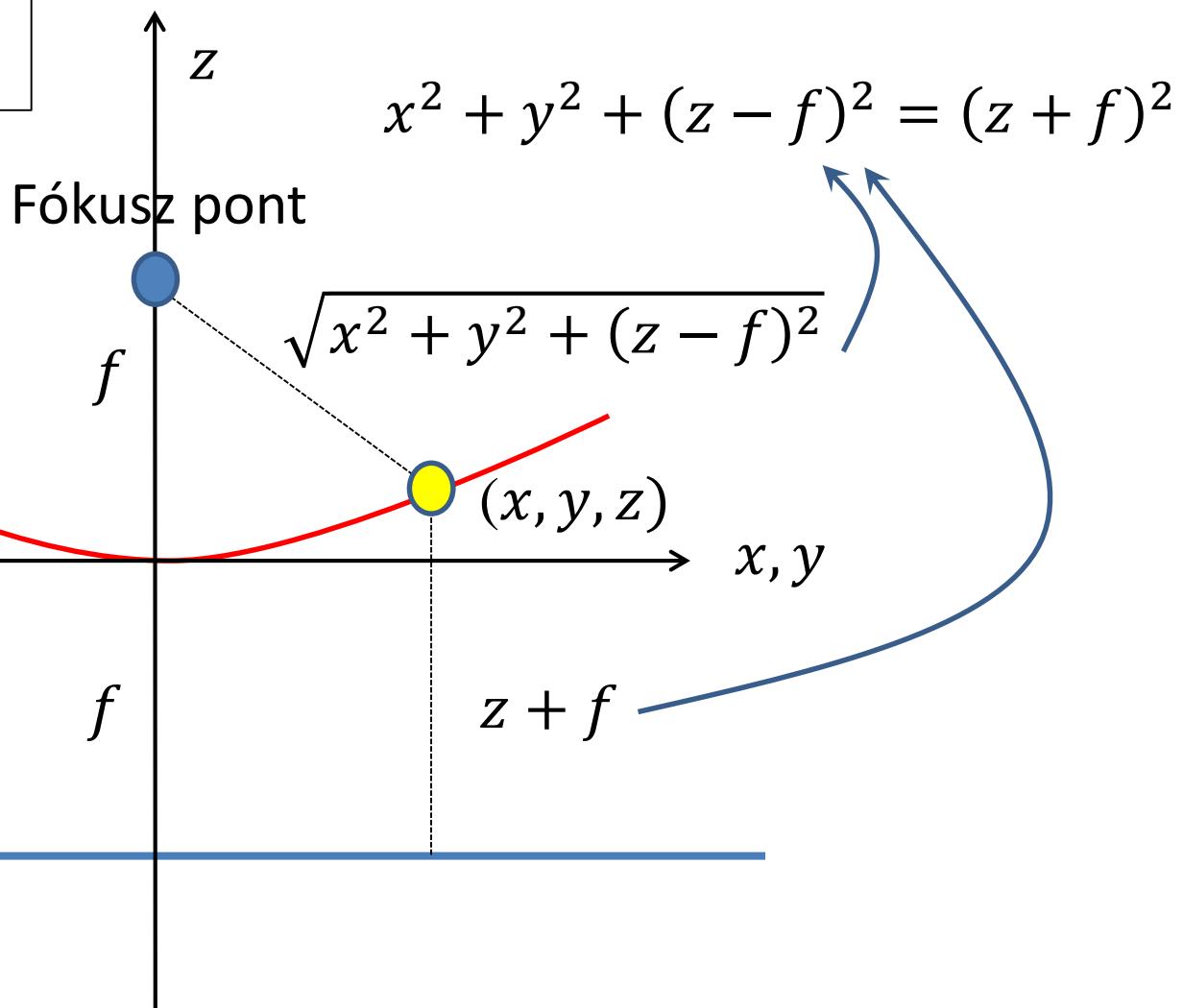


# Paraboloid

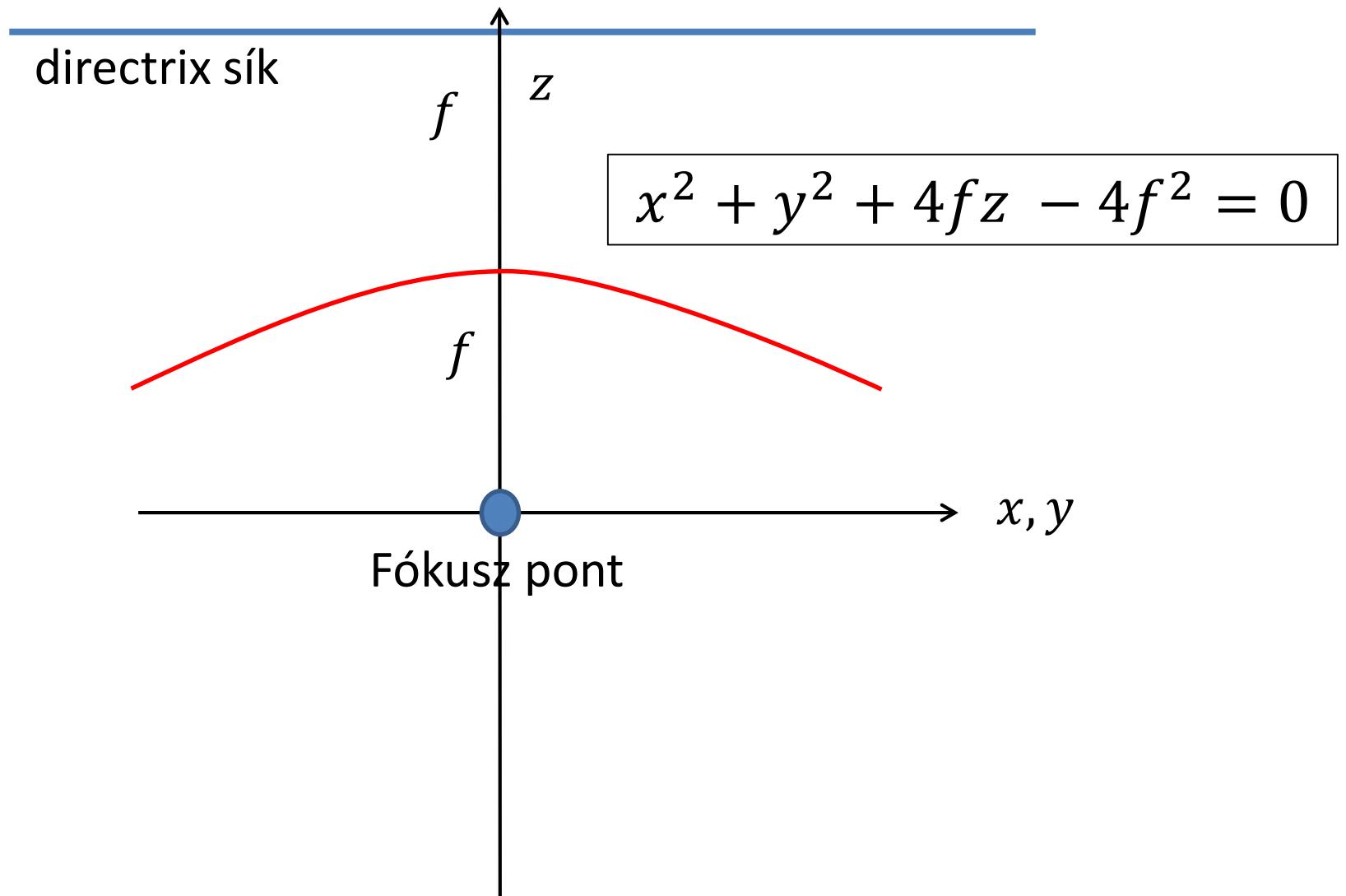
$$x^2 + y^2 - 4fz = 0$$

Gradiens:

$$\mathbf{n} = (2x, 2y, -4f)$$



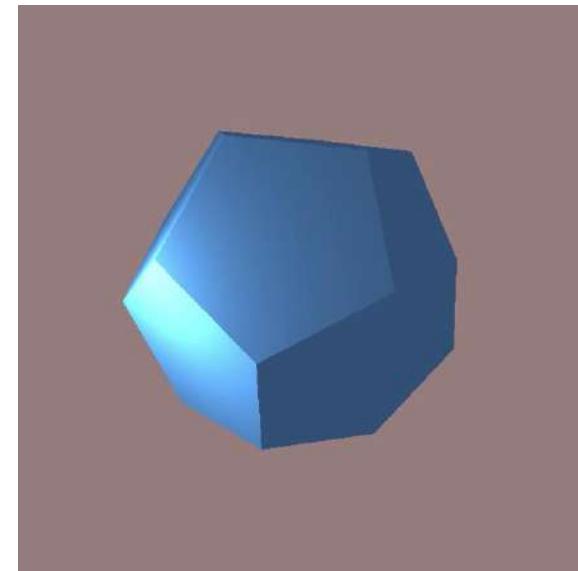
# Paraboloid



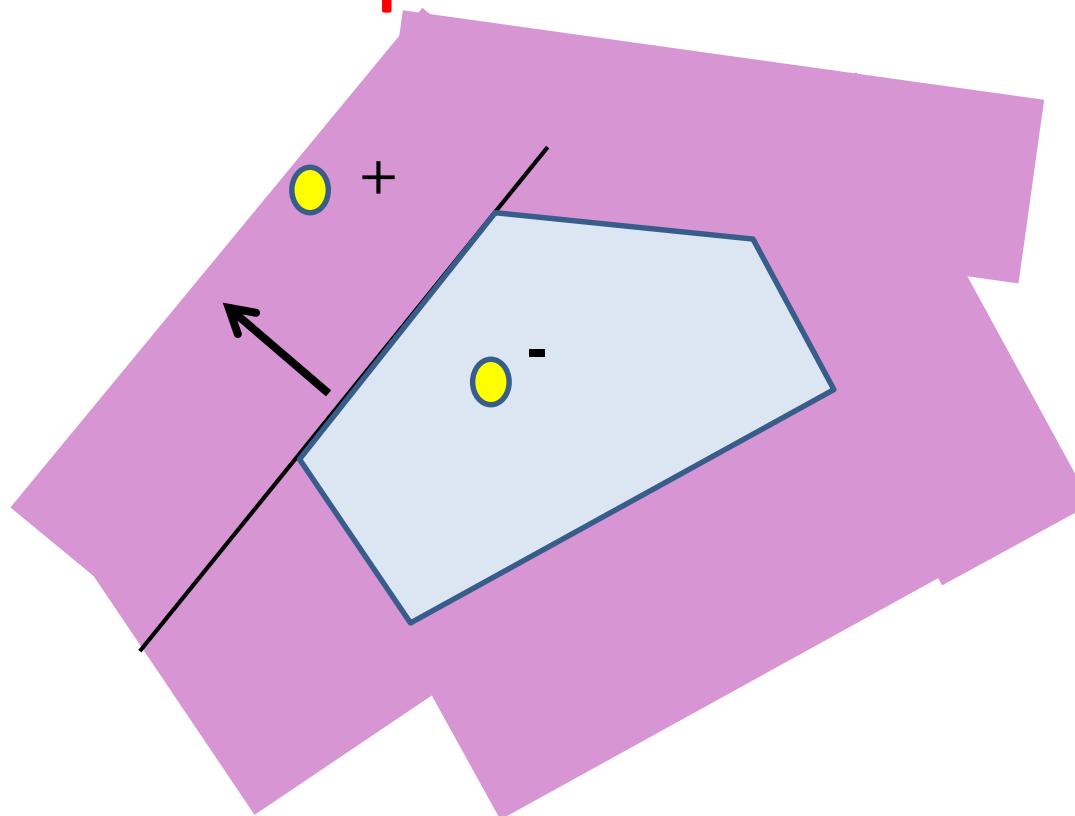
```
v 0 0.618 1.618
v 0 -0.618 1.618
v 0 -0.618 -1.618
v 0 0.618 -1.618
v 1.618 0 0.618
v -1.618 0 0.618
v -1.618 0 -0.618
v 1.618 0 -0.618
v 0.618 1.618 0
v -0.618 1.618 0
v -0.618 -1.618 0
v 0.618 -1.618 0
v 1 1 1
v -1 1 1
v -1 -1 1
v 1 -1 1
v 1 -1 -1
v 1 1 -1
v -1 1 -1
v -1 -1 -1
```

# Dodekaéder: OBJ

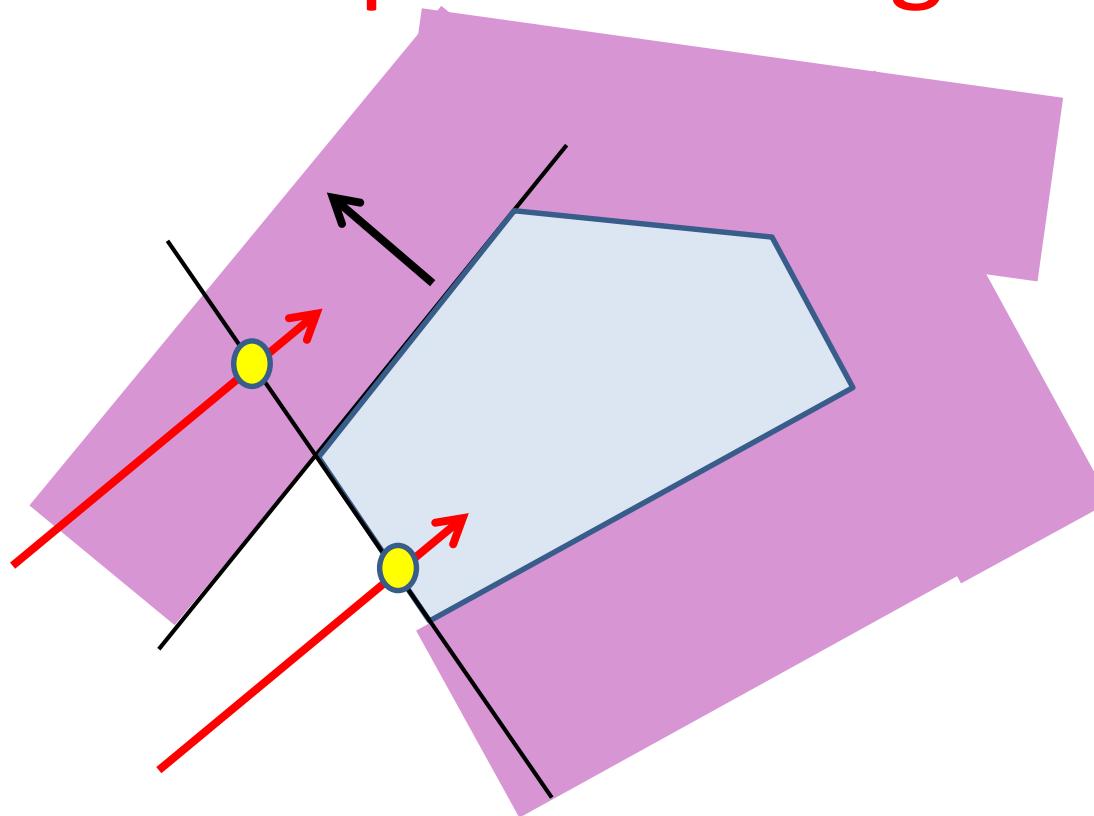
```
f 1 2 16 5 13
f 1 13 9 10 14
f 1 14 6 15 2
f 2 15 11 12 16
f 3 4 18 8 17
f 3 17 12 11 20
f 3 20 7 19 4
f 19 10 9 18 4
f 16 12 17 8 5
f 5 8 18 9 13
f 14 10 19 7 6
f 6 7 20 11 15
```



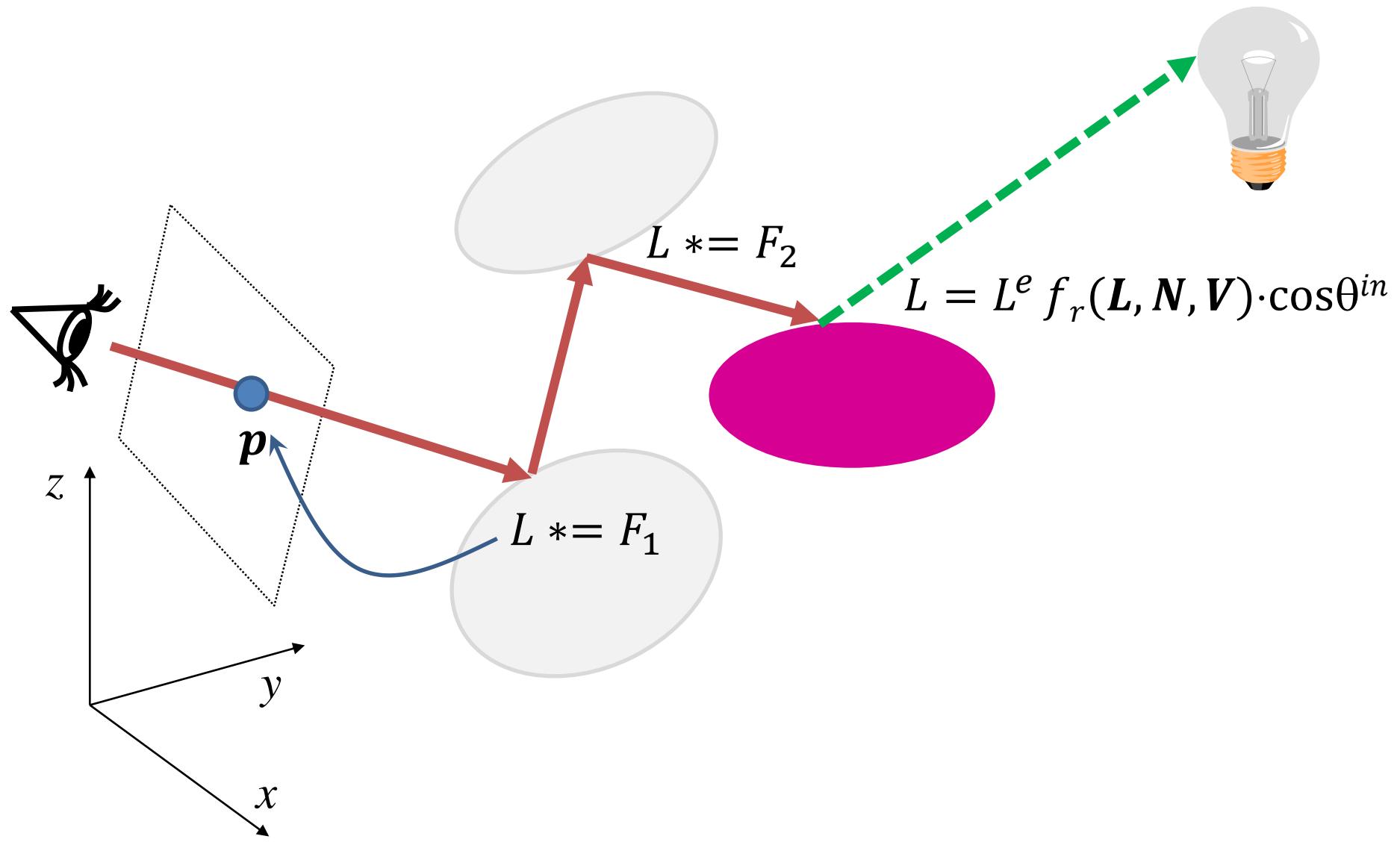
# Konvex poliéder tartalmazás



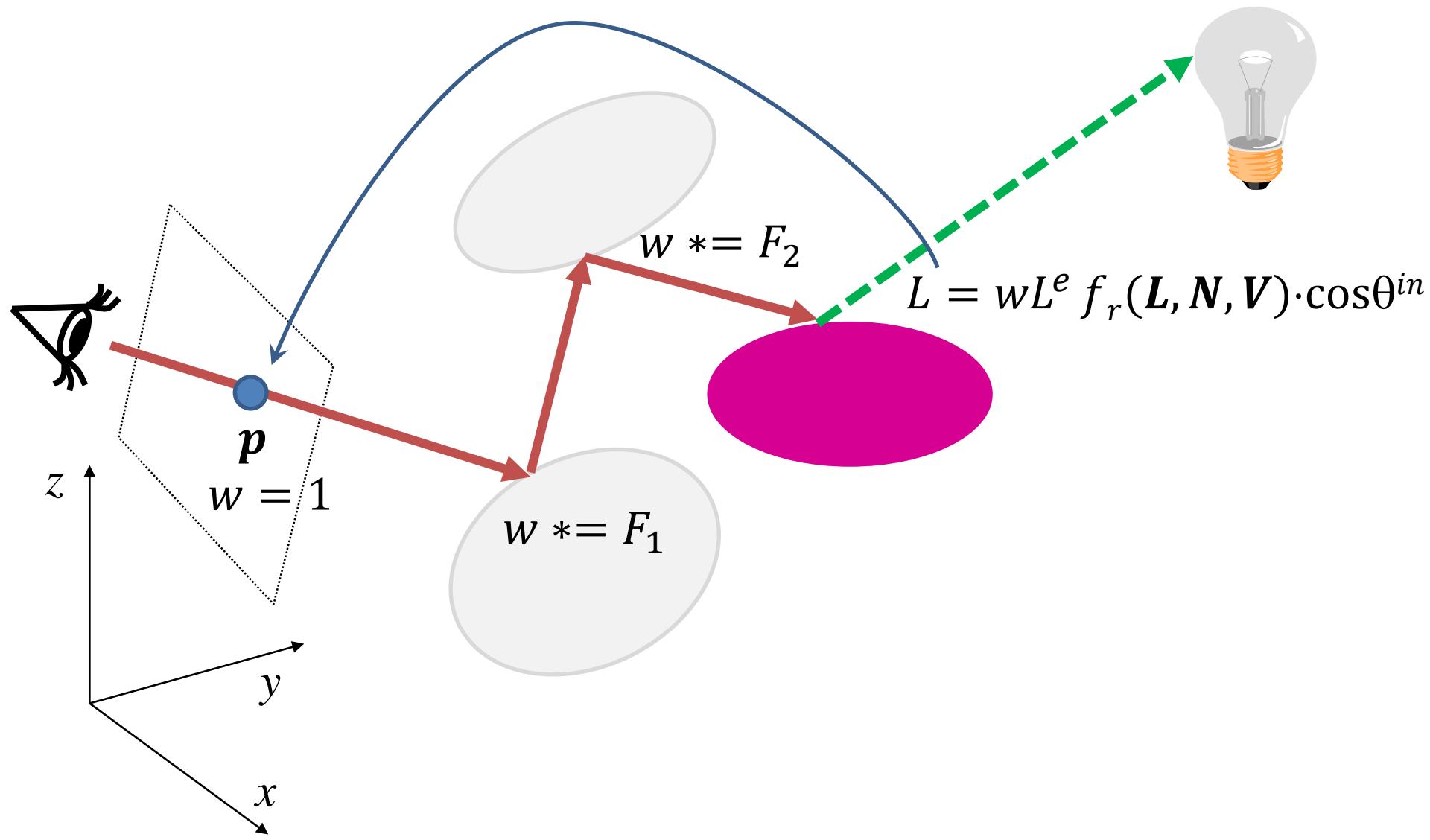
# Konvex poliéder sugárkövetés



# CPU sugárkövetés



# GPU sugárkövetés



# Feladatok

- Sugár-paraboloid metszés, figyelembe véve a korlátozást
- Sugár-dodekaéder, sugár-platoni metszés (poligon vagy sík)
- Tükörirány számítás, Fresnel
- Diffúz-spekuláris sugársűrűség számítás
- Kamera vezérlés

*“Bad programmers worry about the code. Good programmes worry about data structures.”*

*Linus Torvalds*

# Térpartícionáló adatstruktúrák

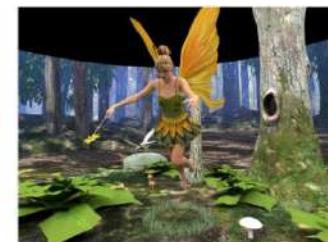
Szirmay-Kalos László



(a) CONFERENCE (282K triangles)



(b) CRYTEK SPONZA (262K triangles)



(c) FAIRY (174K triangles)



(d) HAIRBALL (2.9M triangles)

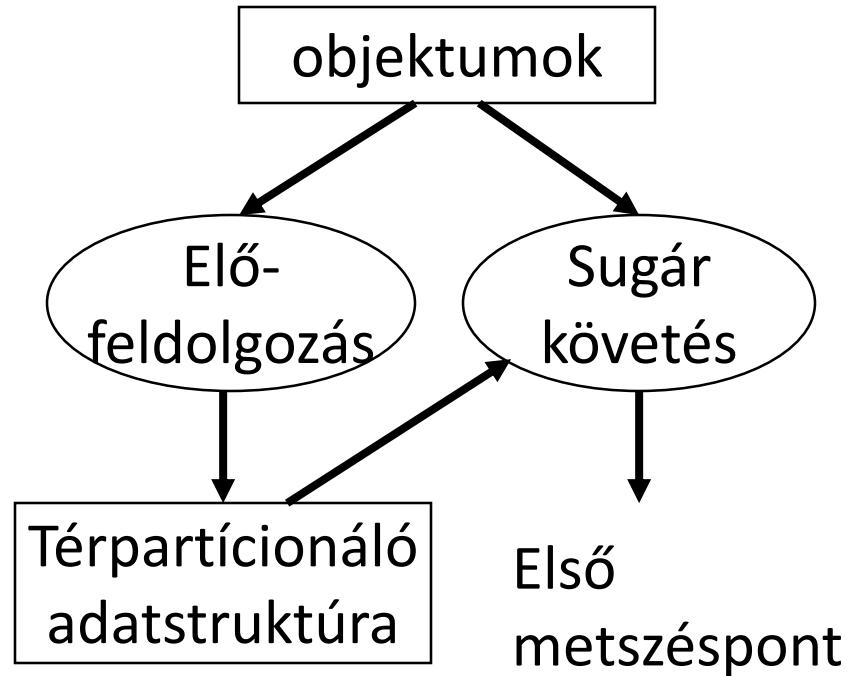
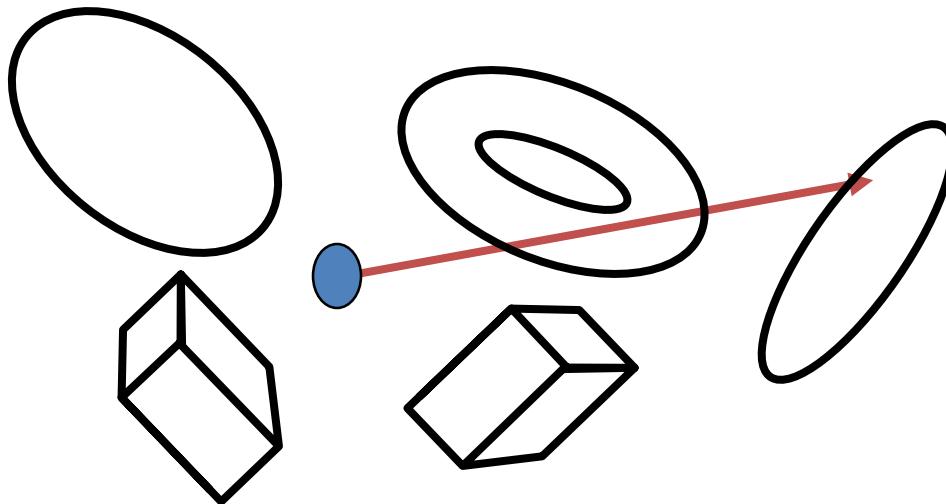


(e) POWER PLANT (12.7M triangles)



(f) SAN MIGUEL (10.5M triangles)

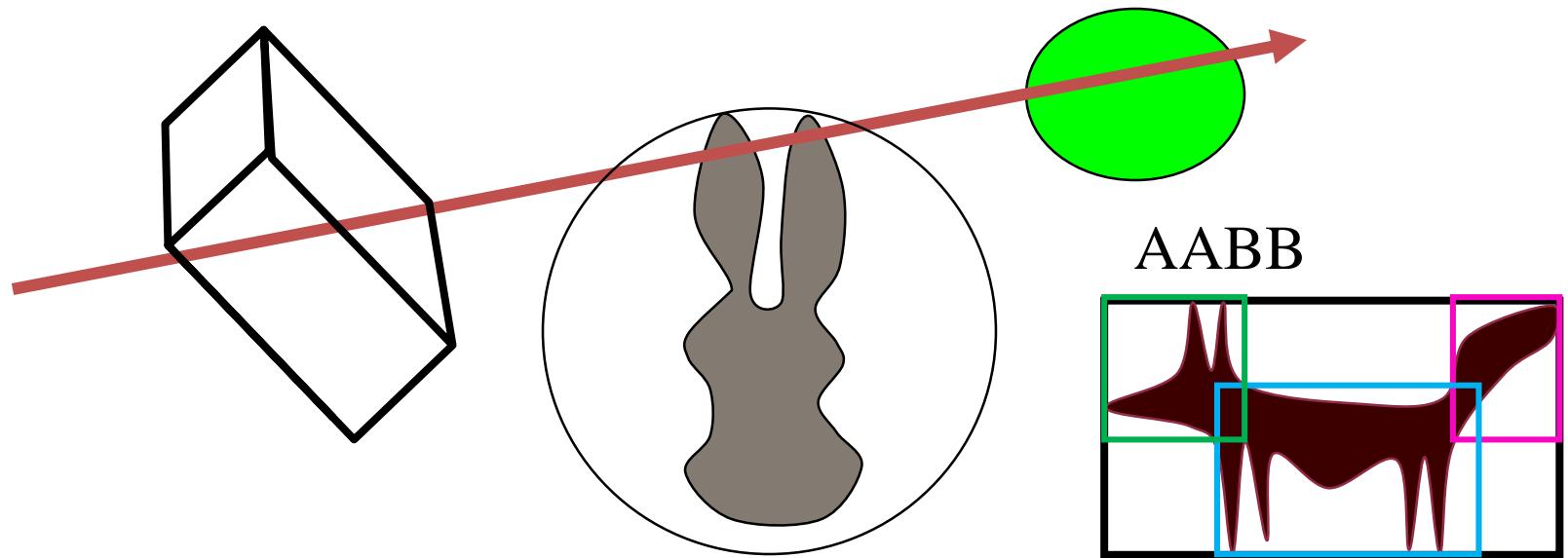
# Térpartícionáló módszerek



Adatstruktúra:

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

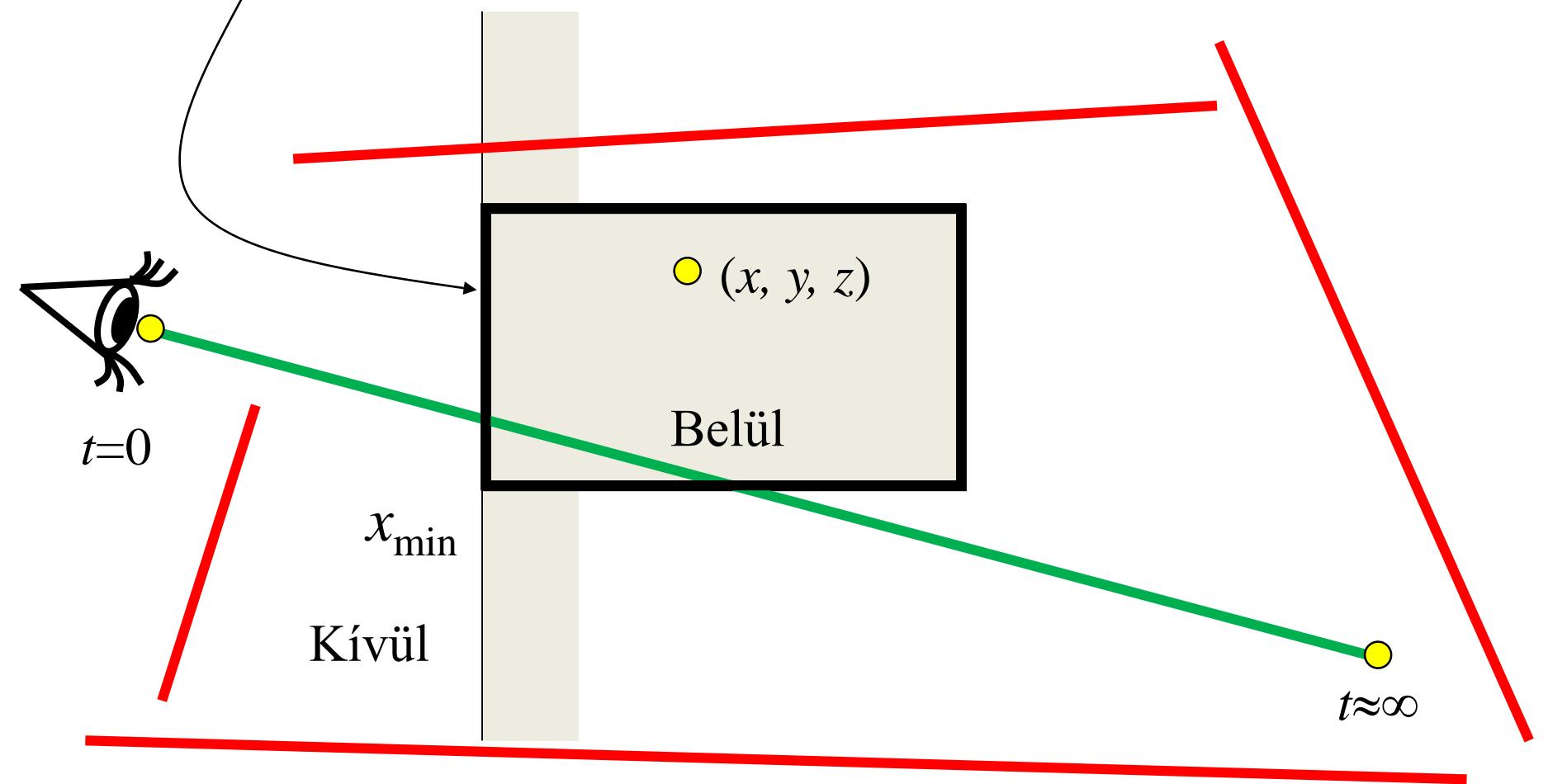
# Befoglaló térfogat (Bounding Volume)



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

# AABB metszés vágással

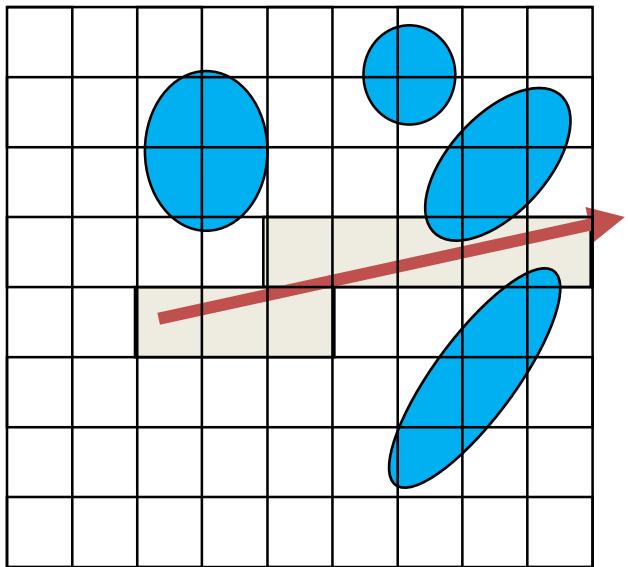
$x > x_{\min}$     $x < x_{\max}$     $y > y_{\min}$     $y < y_{\max}$     $z > z_{\min}$     $z < z_{\max}$



# Reguláris térháló

## Előfeldolgozás:

Minden cellára a metszett objektumok  
komplexitás:  $O(n \cdot c) = O(n^2)$



## Sugárkövetés:

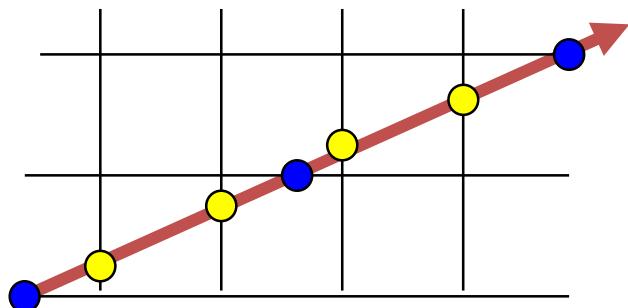
FOR each cell of the line // line drawing

Metszés a cellában lévőkkel

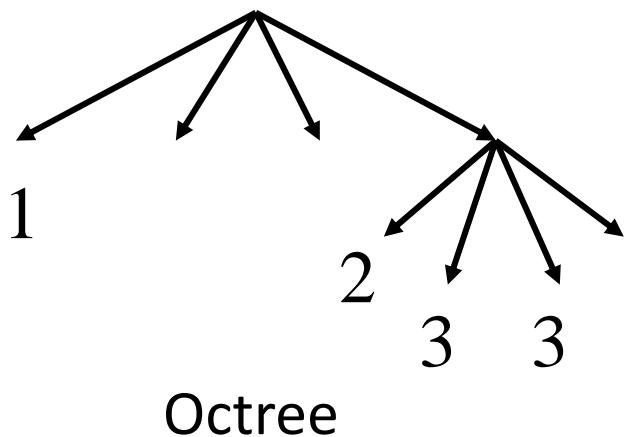
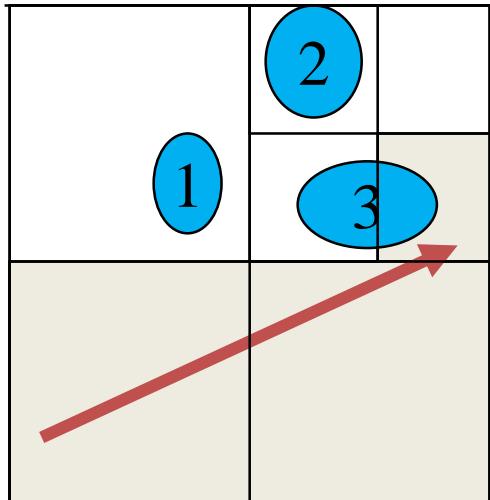
IF van metszés RETURN

ENDFOR

átlagos eset komplexitás:  $O(1)$



# Nyolcas (oktális) fa



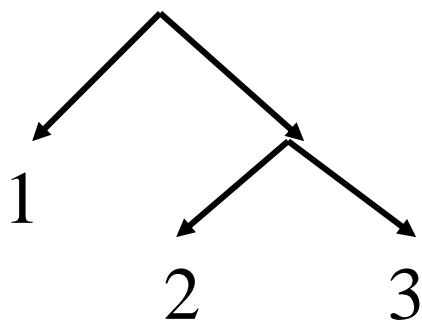
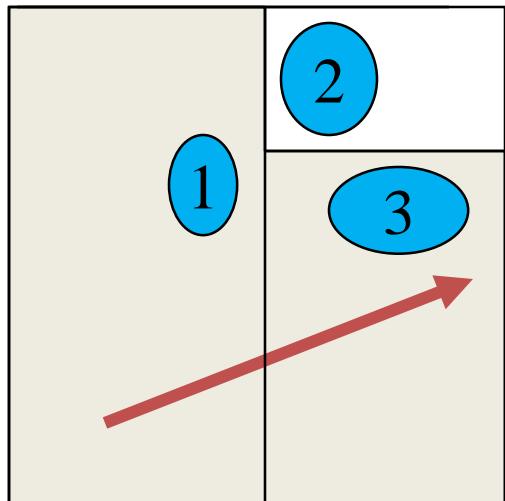
## Faépítés( cella ):

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

## Sugárkövetés:

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

# Binary Space Partitioning fa (kd-fa)



kd-tree

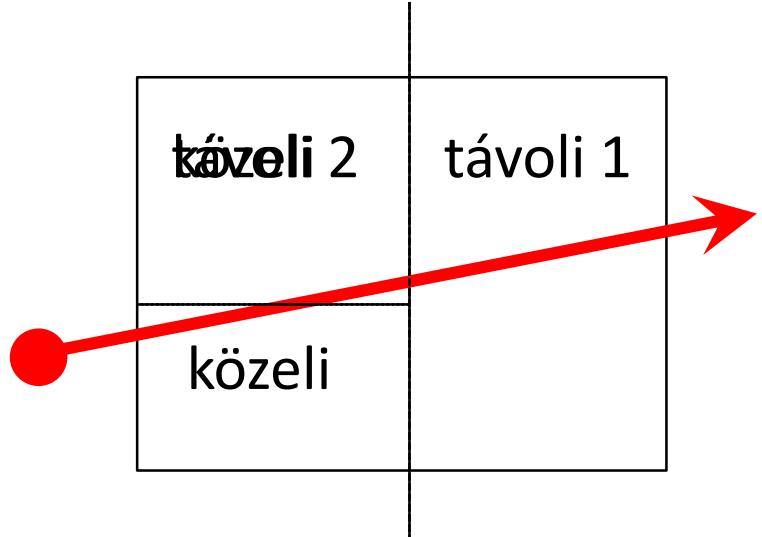
## Faépítés( cella ):

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

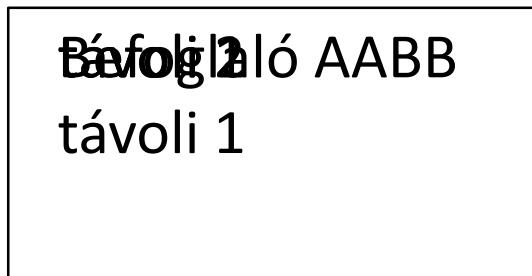
## Sugárkövetés:

```
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
```



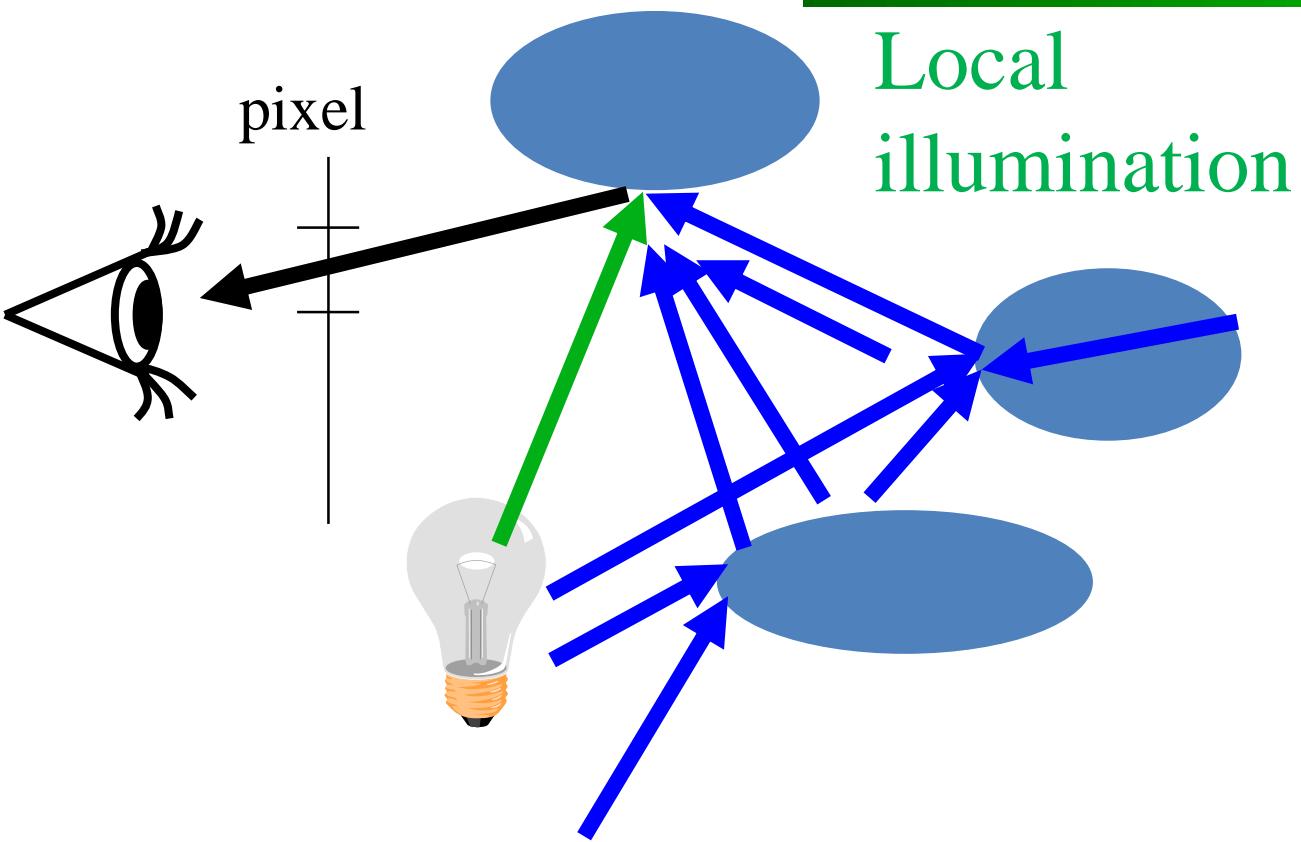
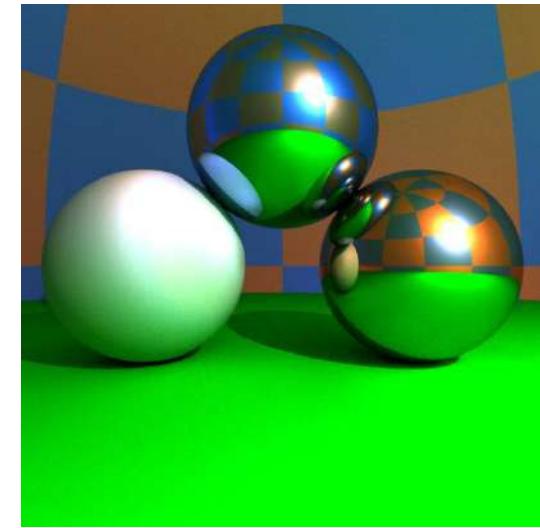
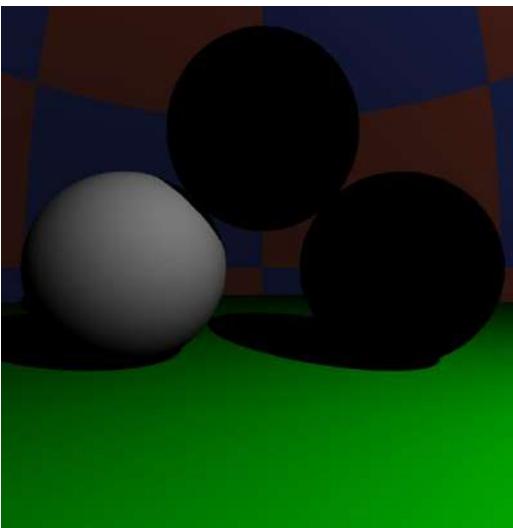
*“Úgy szeretnék integrálni.”*  
Kockaéder

# Globális illumináció: path tracing

Szirmay-Kalos László



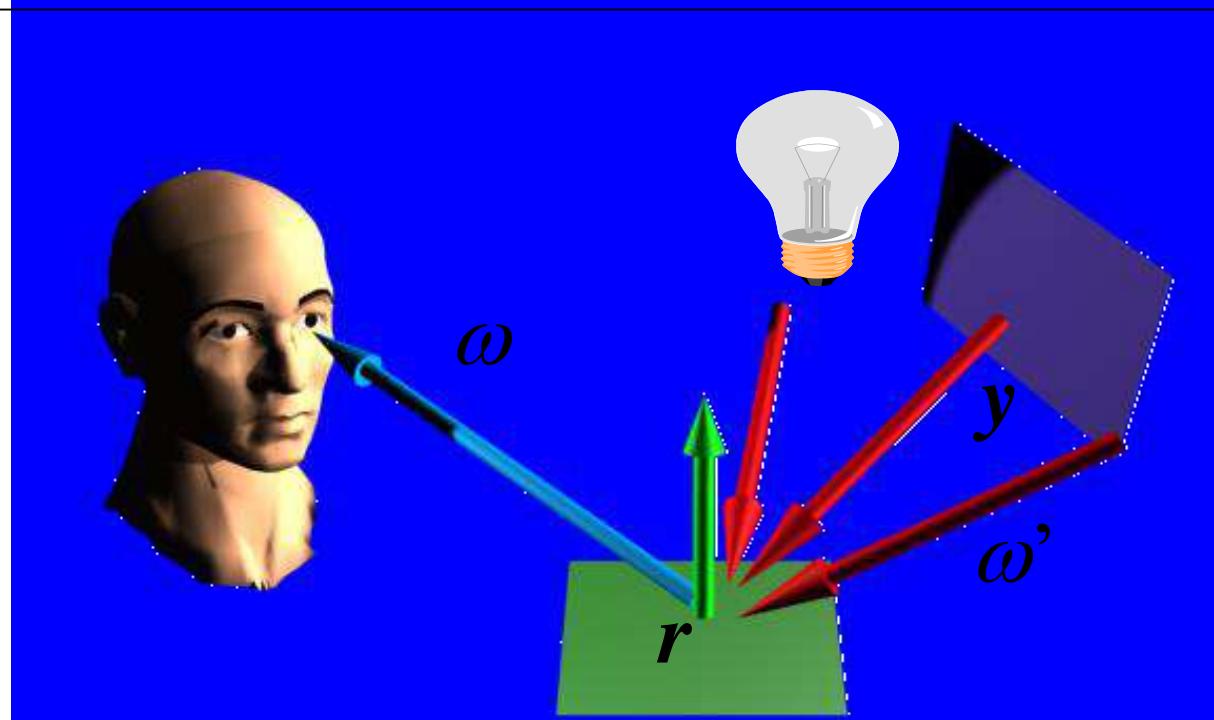
# Képszintézis



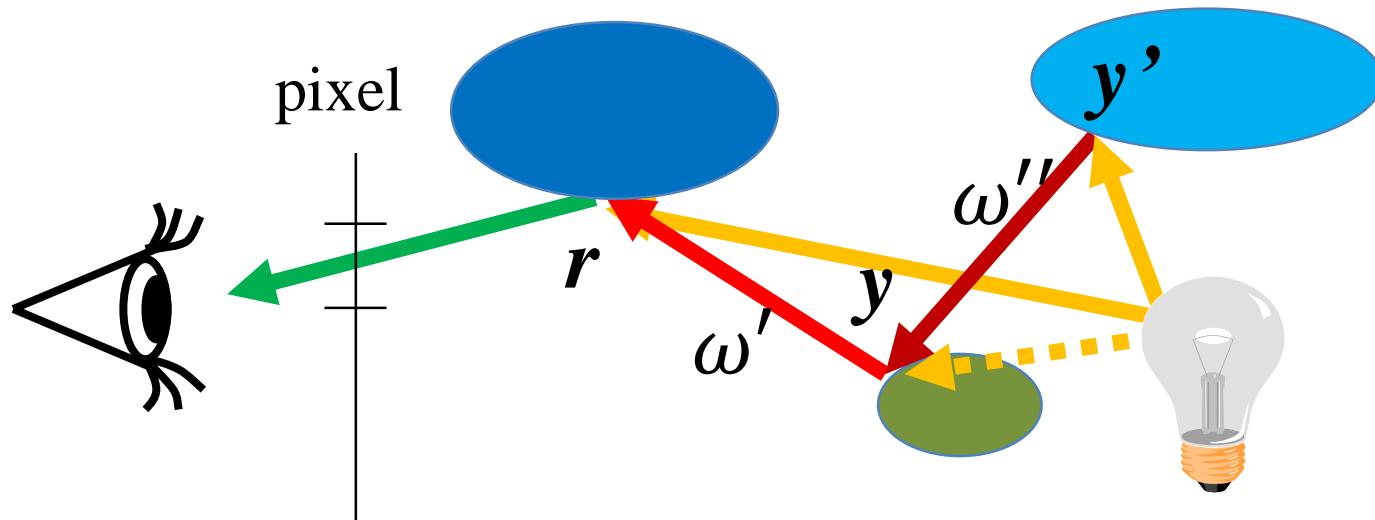
# Rendering equation

$$\text{OutRad} = \text{DirectLight} + \sum \text{InRad} * \text{Reflection}$$

$$L(\mathbf{r}, \omega) = D(\mathbf{r}, \omega) + \int_{\Omega} L(\mathbf{y}, \omega') R(\omega, \omega') d\omega'$$



# A megoldás integrálok sorozata

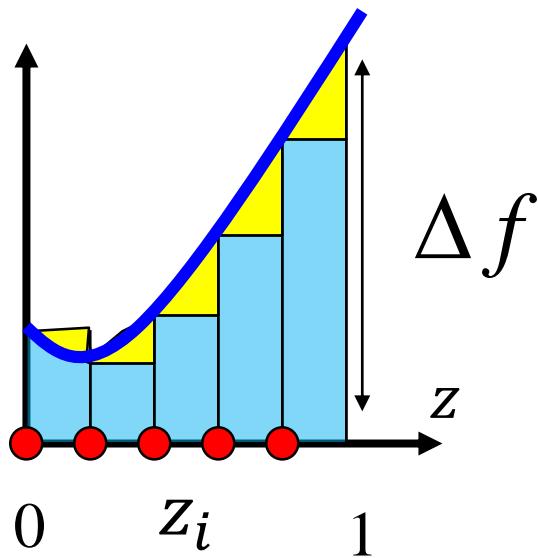


$$L(y', \omega'') = D(y', \omega'') + \int_{\Omega''} L(y'', \omega''') R(\omega'', \omega''') d\omega'''$$

$$L(y, \omega') = D(y, \omega') + \int_{\Omega'} L(y', \omega'') R(\omega', \omega'') d\omega''$$

$$L(r, \omega) = D(r, \omega) + \int_{\Omega} L(y, \omega') R(\omega, \omega') d\omega'$$

# Numerikus integrálás



$$\int_0^1 f(z) dz \approx \frac{1}{M} \sum_{i=1}^M f(z_i)$$

$M$  minta

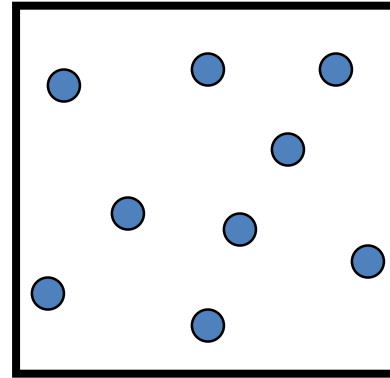
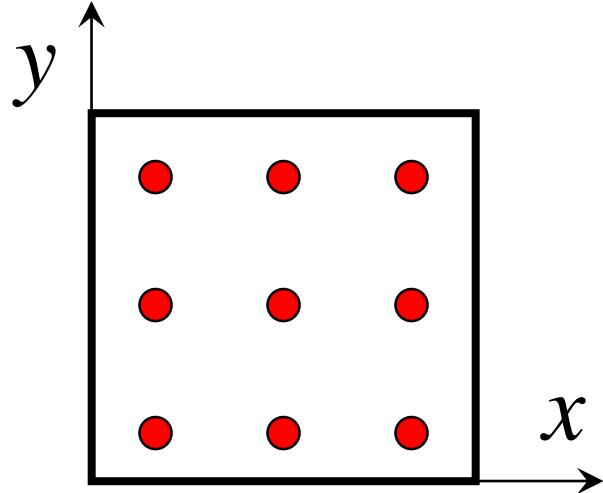
Átlagos  
magasság

Alap

Háromszögek  
száma

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

# Magasabb dimenziók: Megátkozva



Véletlen minták

$n = \sqrt{M}$  minta egy dimenzióban

Hiba 2-dim =  $O(n^{-1}) = O(M^{-0.5})$

Hiba  $D$ -dim =  $O(n^{-1}) = O(M^{-1/D})$

# Monte Carlo integrálás: Integrál = várható érték

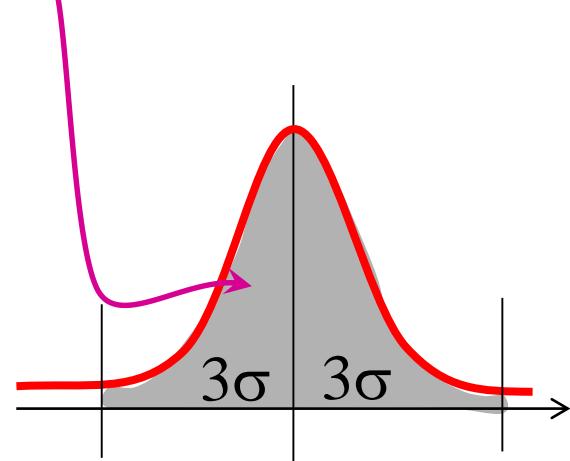
$$\int f(z)dz = \int \frac{f(z)}{p(z)} p(z)dz = E\left[\frac{f(z)}{p(z)}\right] \approx \frac{1}{M} \sum_{i=1}^M \frac{f(z_i)}{p(z_i)}$$

A becslő egy valószínűségi változó:

$$\text{Variancia} = \sigma^2 = D^2 \left[ \frac{f(z)}{p(z)} \right] \frac{1}{M}$$

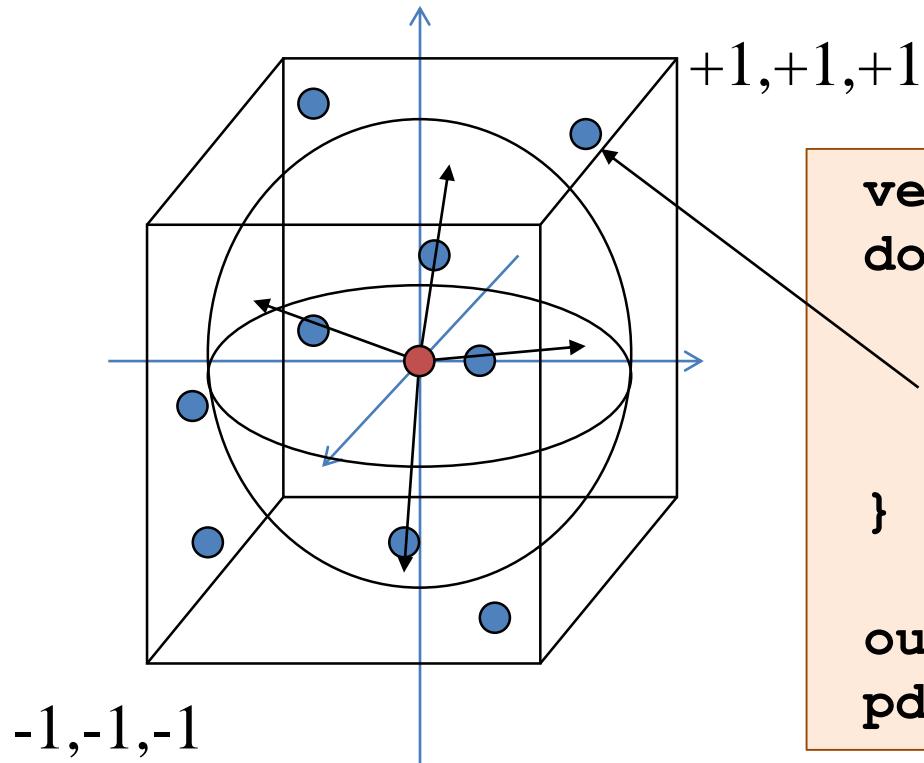
3 × szóráson belül  
99.7% konfidenciával

$$\text{Error} < 3\sigma = \frac{3}{\sqrt{M}} D \left[ \frac{f(z)}{p(z)} \right]$$



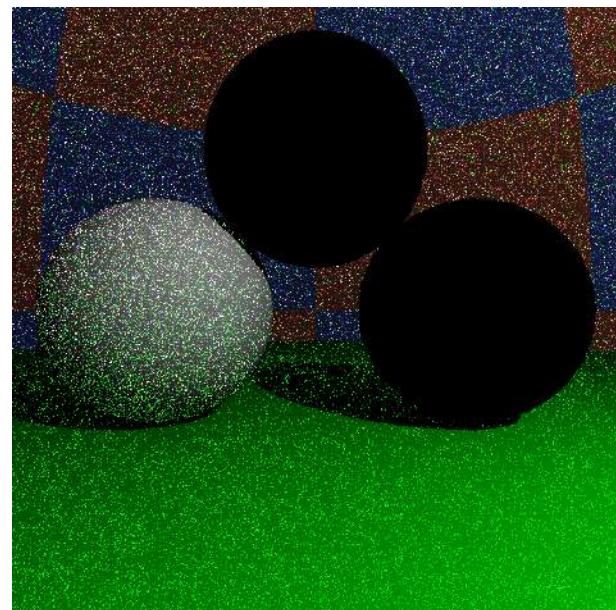
# Irányok generálása egyenletes valószínűsséggel

```
float random() { return (float)rand()/RAND_MAX; }
```

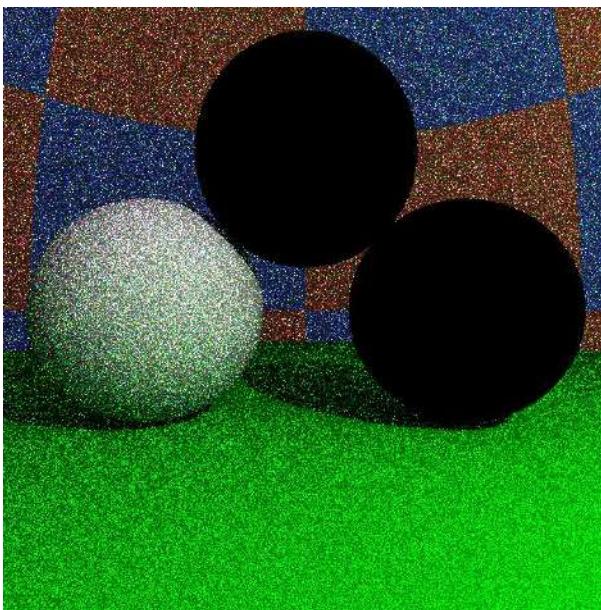


```
vec3 p;  
do {  
    p.x = 2*random()-1;  
    p.y = 2*random()-1;  
    p.z = 2*random()-1;  
} while(dot(p, p) > 1);  
  
outDir = normalize(p);  
pdf = 1.0/4.0/M_PI;
```

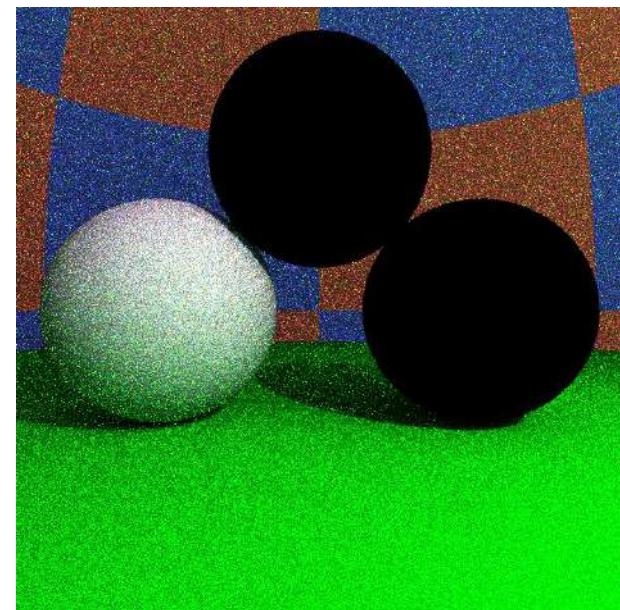
# Uniform



1 sample/pixel

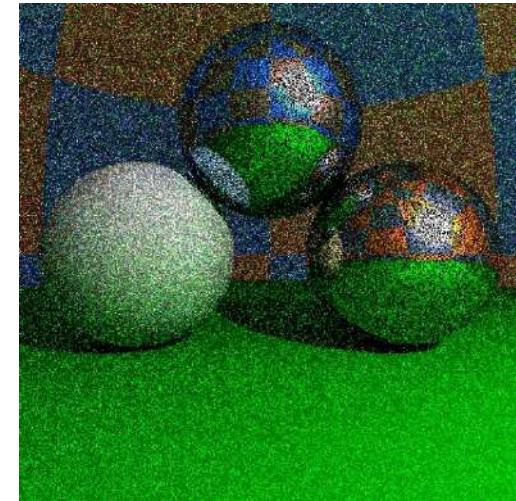


10 samples/pixel



100 samples/pixel

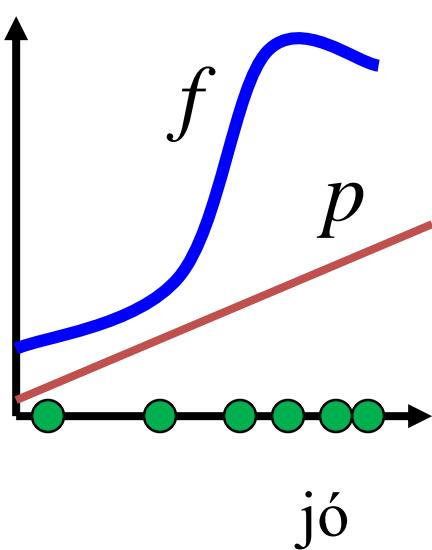
# Fontosság szerinti mintavétel



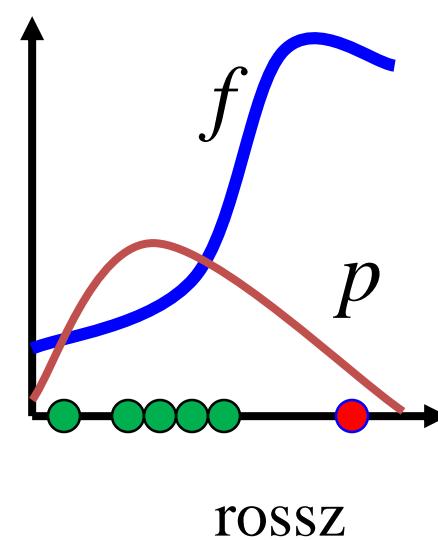
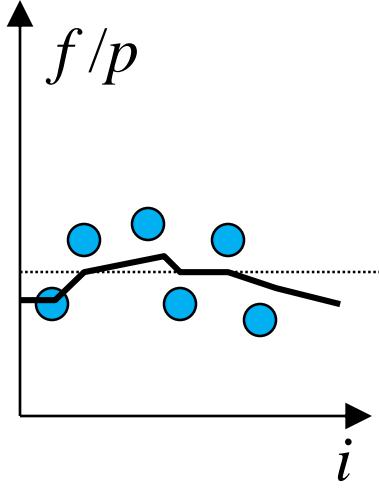
$$\text{Becslő: } \frac{1}{M} \sum_{i=1}^M \frac{f(z_i)}{p(z_i)}$$

$f(z)/p(z)$  legyen lapos

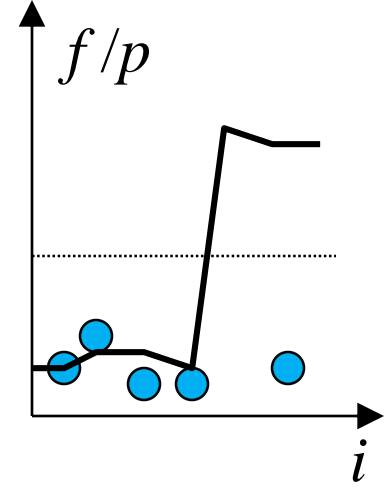
Ahol  $f$  nagy,  $p$  is legyen nagy



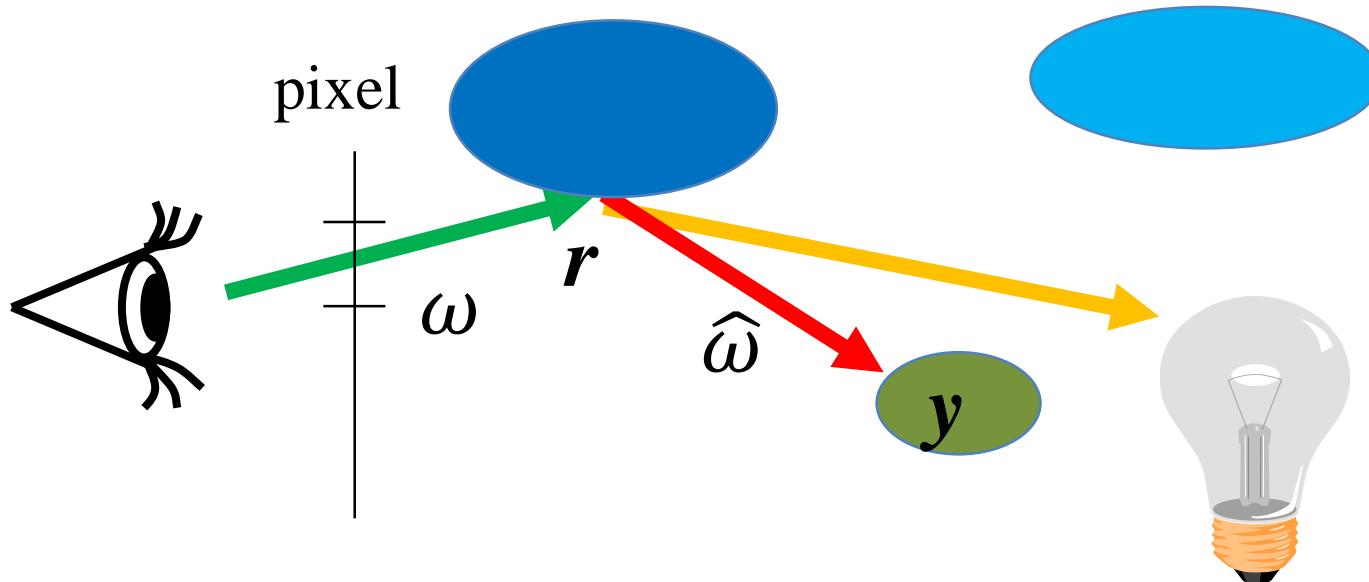
hasonló  $f/p$  hozzájárulások



Ritka, nagy  $f/p$  hozzájárulások



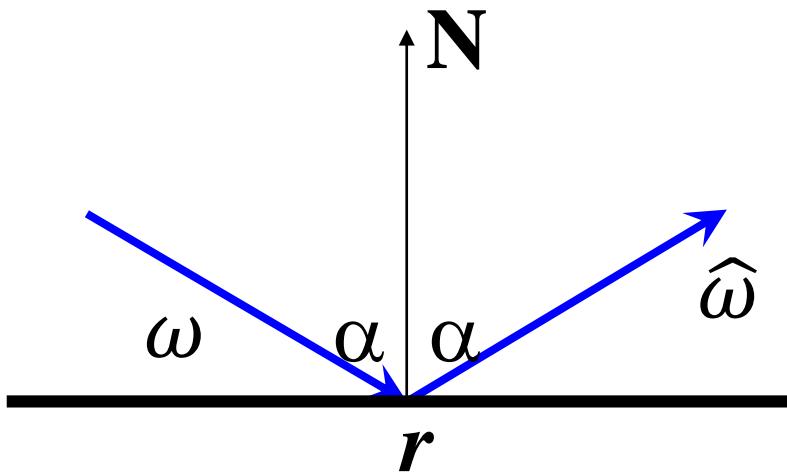
# Véletlen bolyongás



$$\int_{\Omega} L(\mathbf{y}, \omega') R(\omega, \omega') d\omega' \approx \frac{L(\mathbf{y}, \hat{\omega}) R(\omega, \hat{\omega})}{p(\hat{\omega})}$$

- $p(\hat{\omega})$ -nak az integrandusra kell hasonlítania (fontosság)
- A  $L(\mathbf{y}, \omega')$  nem ismerjük, mert éppen számoljuk
- $p(\hat{\omega})$  legyen arányos  $R(\omega, \hat{\omega})$

# Tükör: Irány diszkrét eloszlású (Diract-delta)

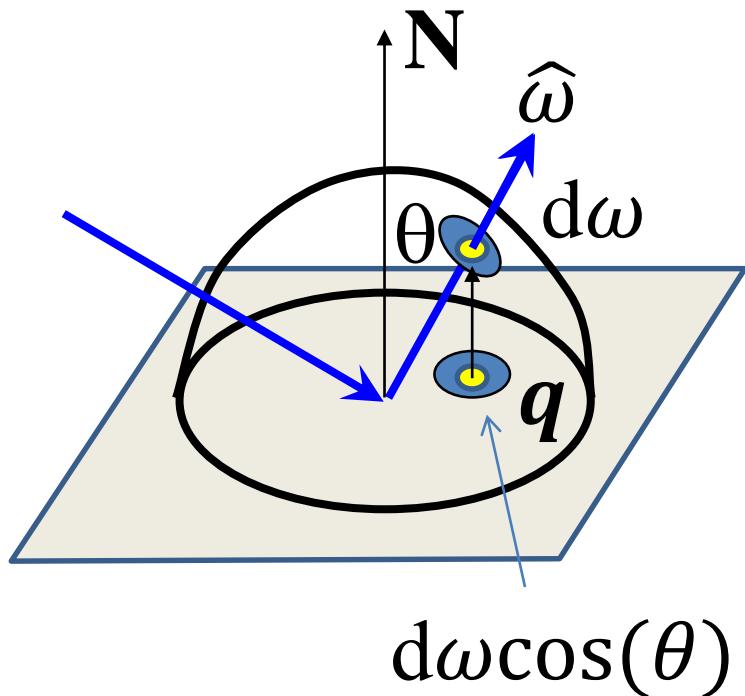


$$\cos \alpha = -(\omega \cdot \mathbf{N})$$

$$\hat{\omega} = \omega + 2\mathbf{N} \cos \alpha$$

```
float SampleMirror(vec3 N, vec3 inDir, vec3& outDir) {  
    outDir = inDir - N * dot(N, inDir) * 2.0f;  
    return 1; // pdf  
}
```

# Diffúz: Irány cos eloszlással

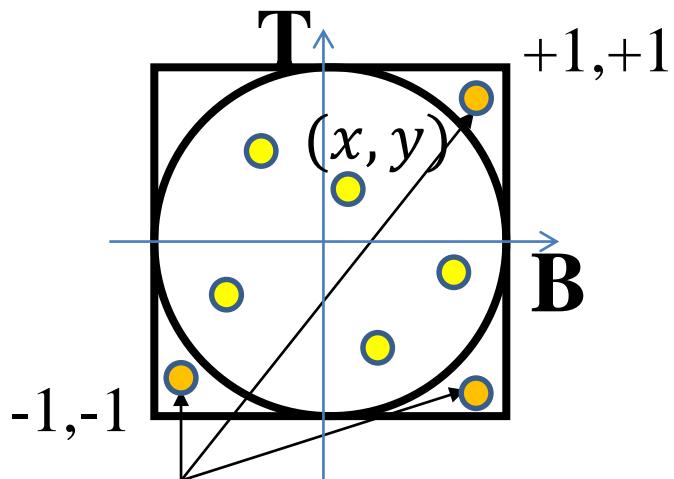


$$\begin{aligned} p(\hat{\omega}) d\omega &= \text{Prob}\{\hat{\omega} \text{ in } d\omega\} \\ &= \text{Prob}\{q \text{ in } d\omega \cos(\theta)\} \\ &= \frac{d\omega \cos(\theta)}{\text{kör területe}} \quad \text{Ha } q \text{ egyenletes eloszlású} \end{aligned}$$

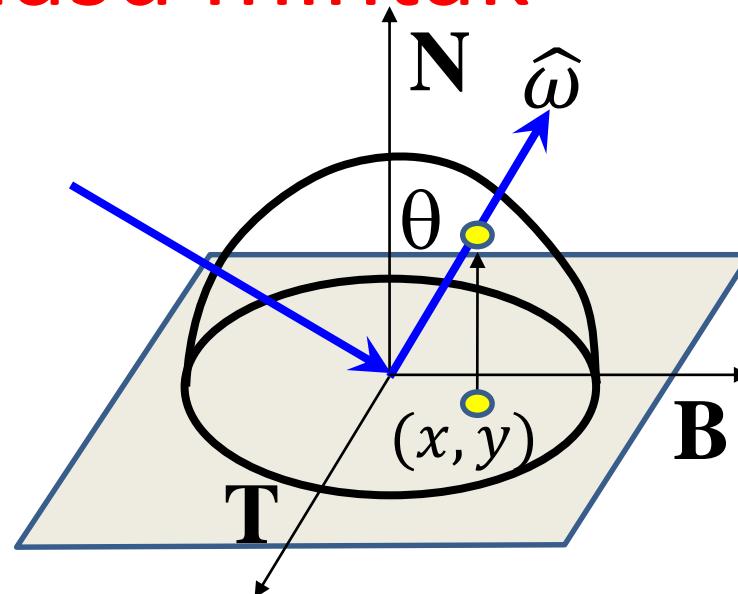
$$= \frac{d\omega \cos(\theta)}{\pi}$$

$$p(\hat{\omega}) = \frac{\cos(\theta)}{\pi}$$

# Cos eloszlású minták



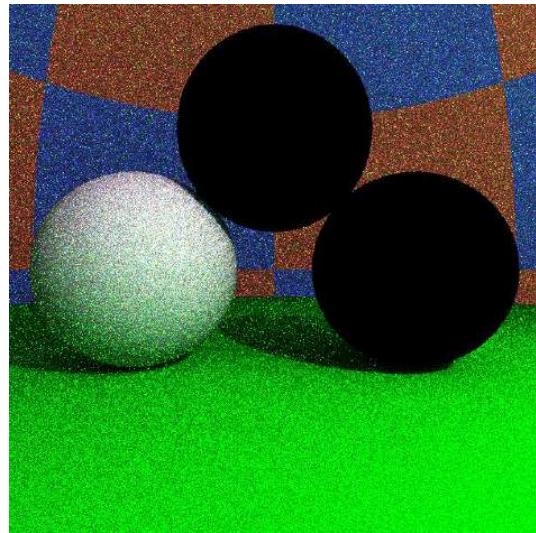
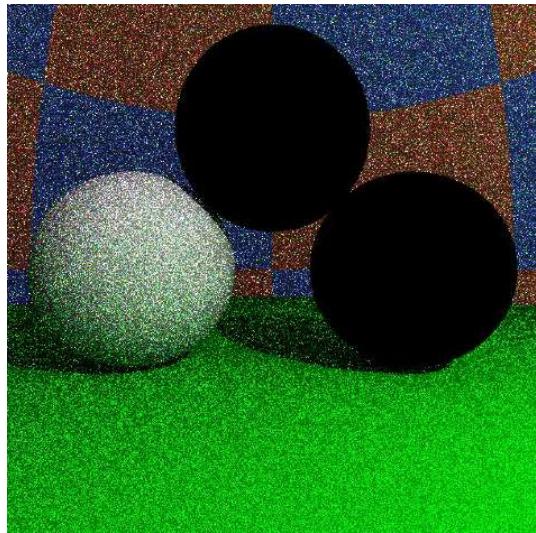
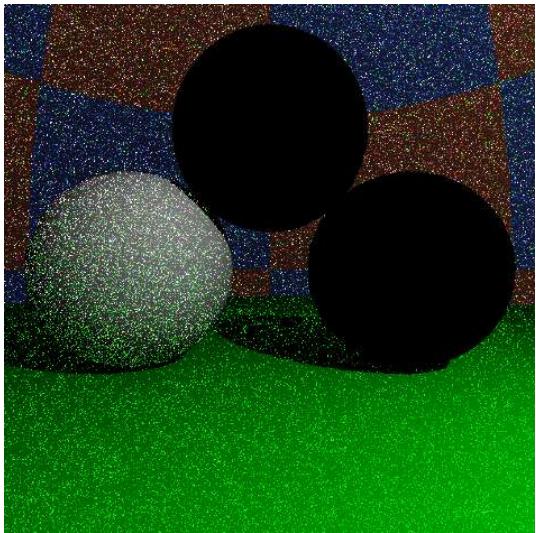
Elvetett minták



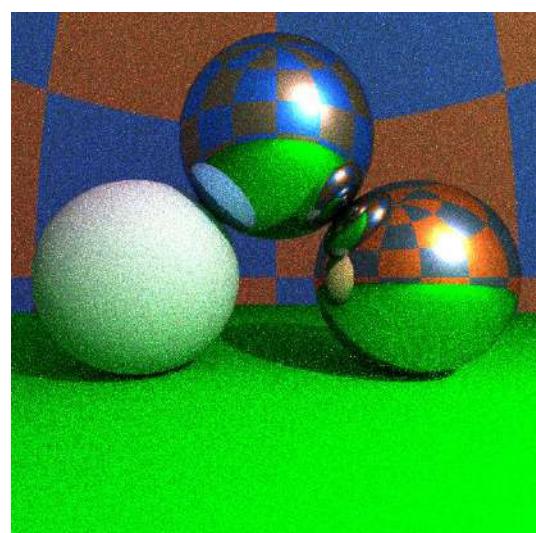
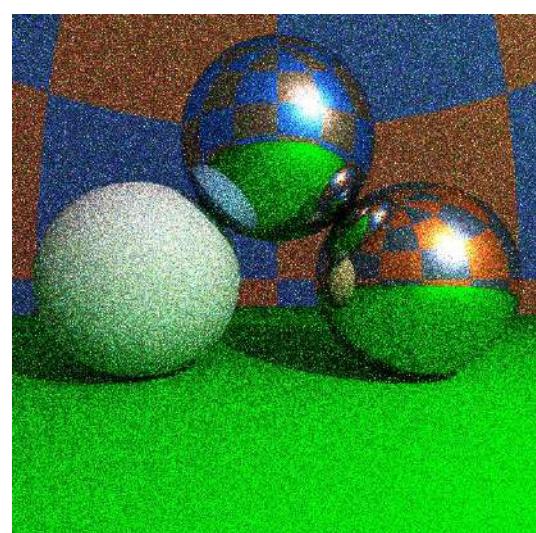
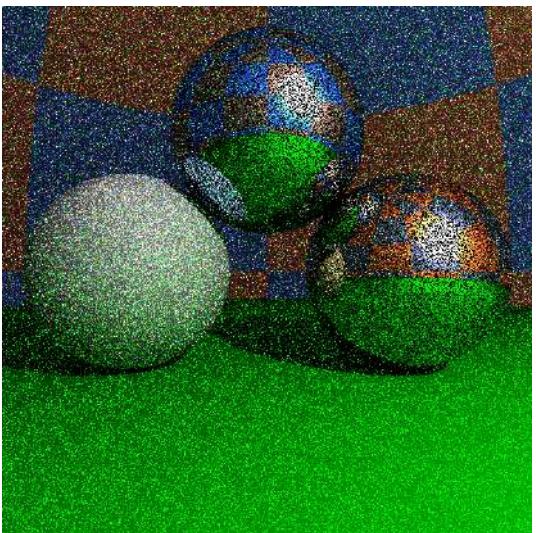
```
float SampleDiffuse(vec3 N, vec3 inDir, vec3& outDir) {  
    vec3 T = normalize(cross(N, vec3(1,0,0)));  
    vec3 B = cross(N, T);  
    float x, y, z;  
    do { x = 2*random()-1; y = 2*random()-1; }  
        while(x*x + y*y > 1); // reject if not in circle  
  
    z = sqrtf(1 - x*x - y*y); // project to hemisphere  
    outDir = T * x + B * y + N * z;  
    return z/M_PI; // pdf  
}
```

# Fontosság szerinti mintavétel

egyenletes



$\cos +$  diszkrét



1 sample/pixel

10 samples/pixel

100 samples/pixel

# Megállás és visszaverődés típus: Orosz rulett

1. Monte Carlo integrál:

$$\int_{\Omega} L(\mathbf{y}, \omega') R(\omega, \omega') d\omega' = \mathbf{E} \left[ \frac{L(\mathbf{y}, \hat{\omega}) k_d \cos^+(\theta)}{p_d(\hat{\omega})} \right] + \mathbf{E} \left[ \frac{L(\mathbf{y}, \hat{\omega}) F \delta(\omega_m, \hat{\omega})}{p_m(\hat{\omega})} \right]$$

$$R_d + R_m$$

$$\mathbf{E}[L_d]$$

$$\mathbf{E}[L_m]$$

2. Számold  $\mathbf{E}[L_d]$ -t  $s_d$  valószínűsséggel,  $\mathbf{E}[L_m]$ -t  $s_m$  valószínűsséggel, egyébként 0
3. Kompenzálj  $s$ -sel osztással

Várható érték OK:

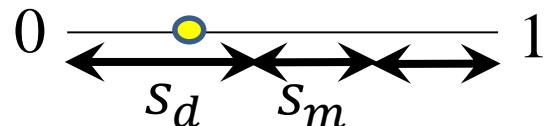
$$s_d \mathbf{E}[L_d/s_d] + s_m \mathbf{E}[L_m/s_m] + (1 - s_d - s_m)0 = \mathbf{E}[L_d] + \mathbf{E}[L_m]$$

# A visszaverődés típusának kiválasztása

$$\frac{L_d}{s_d} = \frac{L(\mathbf{y}, \hat{\omega}) k_d \cos^+(\theta)}{p_d(\hat{\omega}) s_d} = L(\mathbf{y}, \hat{\omega}) \frac{k_d \cos^+(\theta)}{\frac{\cos(\theta)}{\pi} s_d} = L(\mathbf{y}, \hat{\omega}) \frac{k_d \pi}{s_d}$$

$$\frac{L_m}{s_m} = \frac{L(\mathbf{y}, \hat{\omega}) F \delta(\omega_m, \hat{\omega})}{p_m(\hat{\omega}) s_m} = L(\mathbf{y}, \hat{\omega}) \frac{F}{s_m}$$

- Tükörirány valószínűsége:  $s_m = F$  luminance
- Diffúz irány valószínűsége:  $s_d = k_d \pi$  luminance
- Megállás valószínűsége:  $1 - s_m - s_d$



# Path Tracer

```
vec3 trace(Ray ray, int depth = 0) {  
    Hit hit = firstIntersect(ray);  
    if (hit.t < 0 || depth >= maxdepth) return vec3(0,0,0);  
    [r, N, kd, n, k] ← hit;  
    vec3 outRad = DirectLight(hit);  
    rnd = random();           // Russian roulette  
    sd = Luminance(kd π); sm = Luminance(Fresnel(ray.dir, N));  
    if (rnd < sd) { // diffuse  
        pdf = SampleDiffuse(N, ray.dir, outDir);  
        inRad = trace(Ray(r + Nε, outDir), depth+1);  
        outRad += inRad * kd * dot(N, outDir) / pdf / sd;  
    } else if (rnd < sd + sm) { // mirror  
        pdf = SampleMirror(N, ray.dir, outDir);  
        inRad = trace(Ray(r + Nε, outDir), depth+1);  
        outRad += inRad * Fresnel(ray.dir, N) / pdf / sm;  
    }  
    return outRad;  
}
```

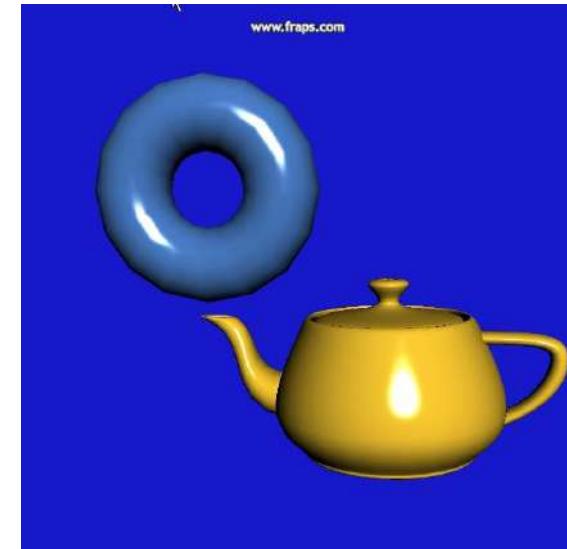
*“All problems in computer graphics can  
be solved with a matrix inversion.”*

*Jim Blinn*

# Inkrementális 3D képszintézis

## 1. Bevezetés

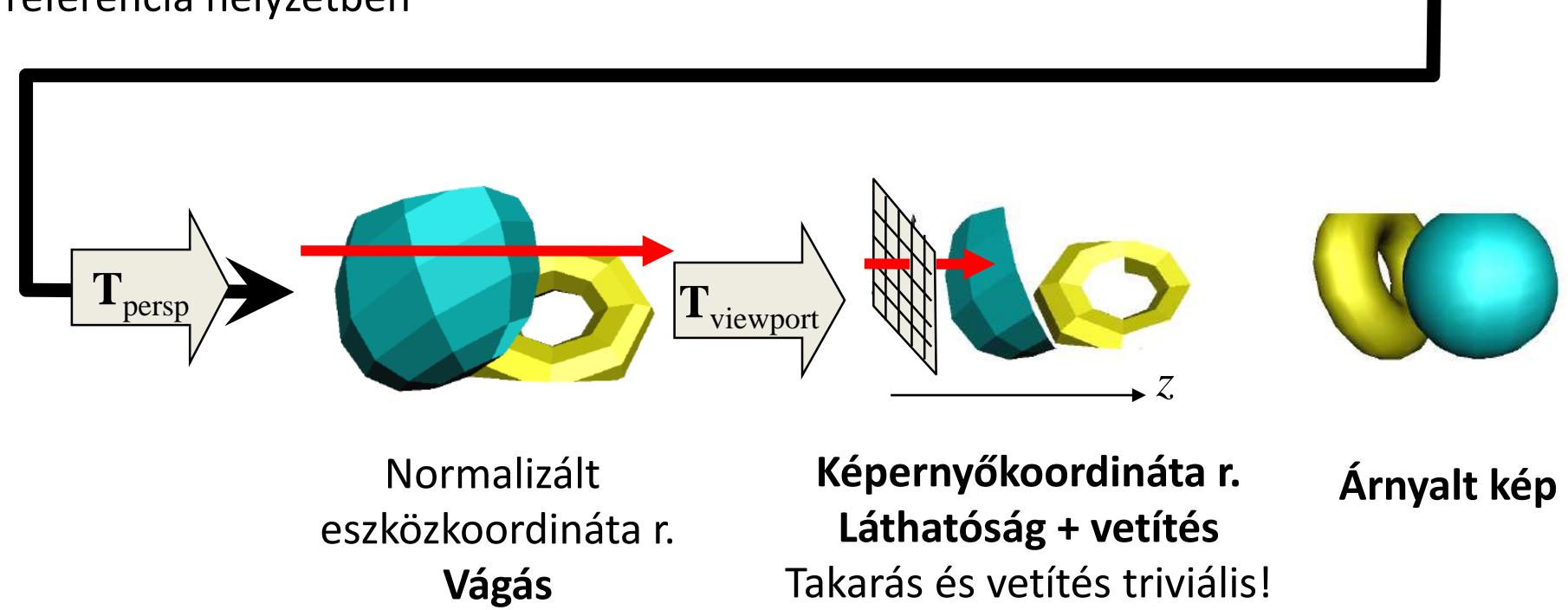
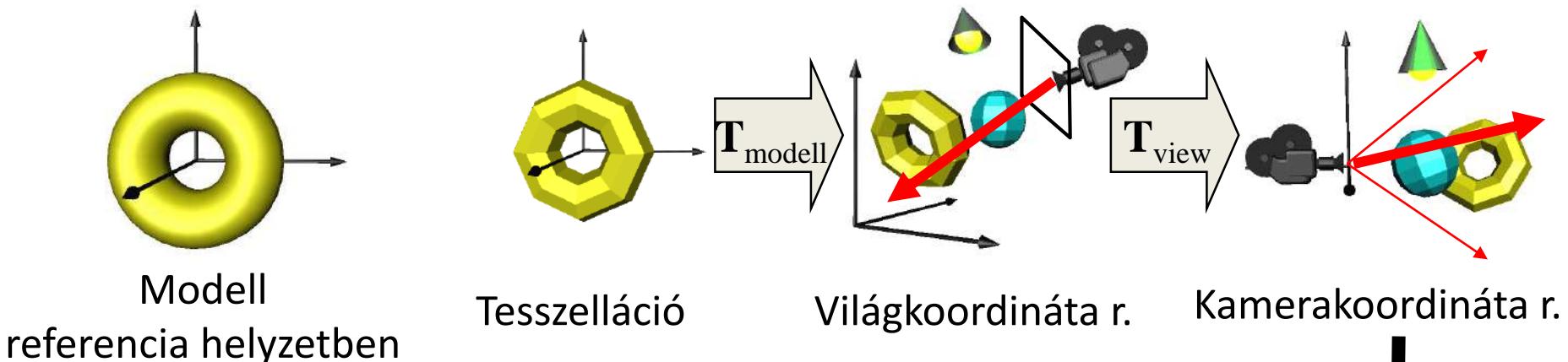
Szirmay-Kalos László



# Inkrementális képszintézis

- Objektum vezérelt megközelítés
  - Cél a sebesség és a hw támogathatóság
- 
- koherencia: oldjuk meg nagyobb egységekre
  - feleslegesen ne számoljunk: vágás
  - transzformációk: minden feladathoz megfelelő koordinátarendszert
    - vágni, transzformálni nem lehet akármit: tesszelláció

# 3D inkrementális képszintézis



*“Order is repetition of units.*

*Chaos is multiplicity without rhythm.”*

*Maurits Cornelis Escher*

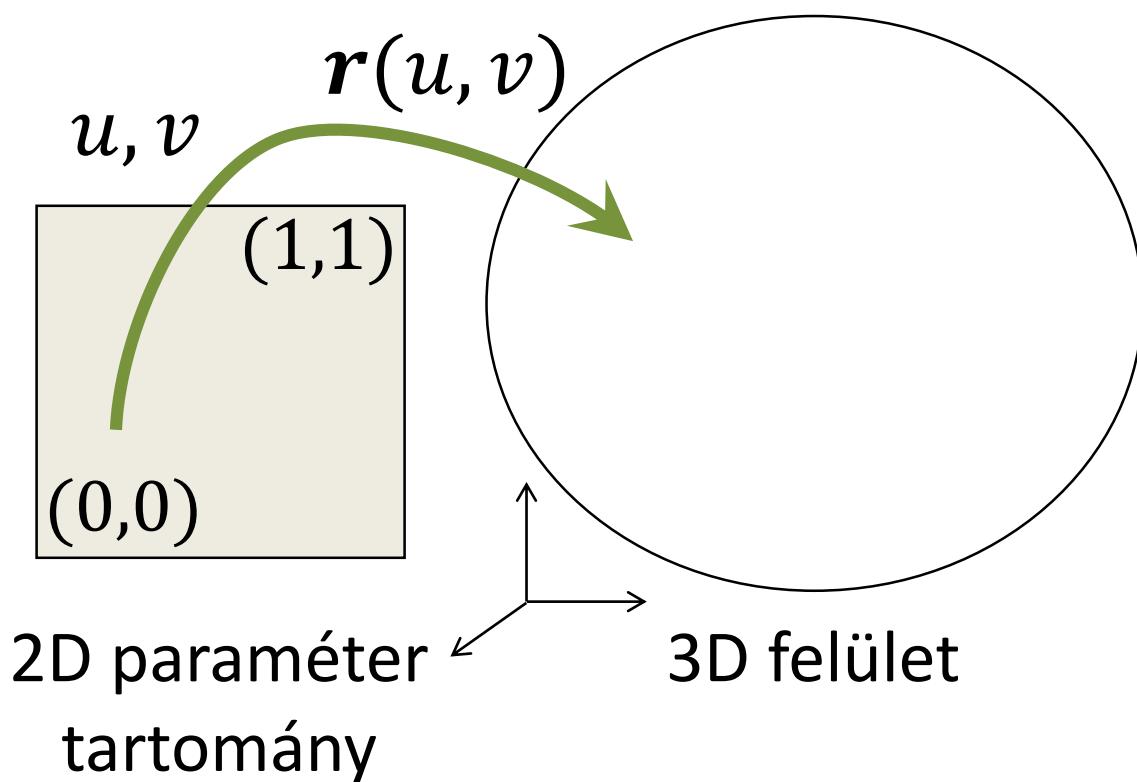
# Inkrementális 3D képszintézis

## 2. Geometria a GPU-nak

Szirmay-Kalos László

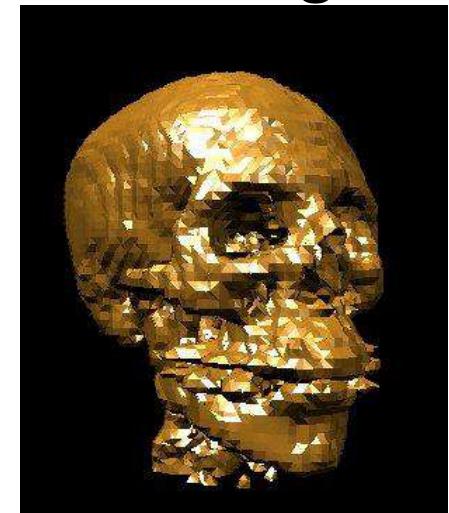
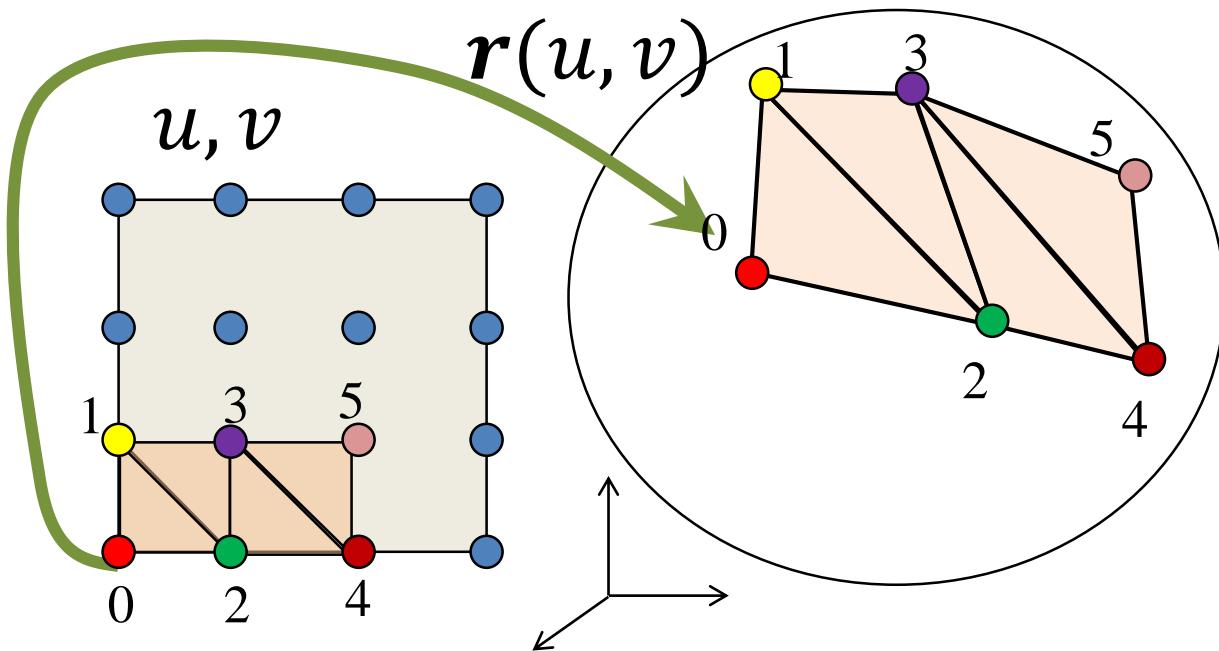


# Parametrikus felületek tesszellációja



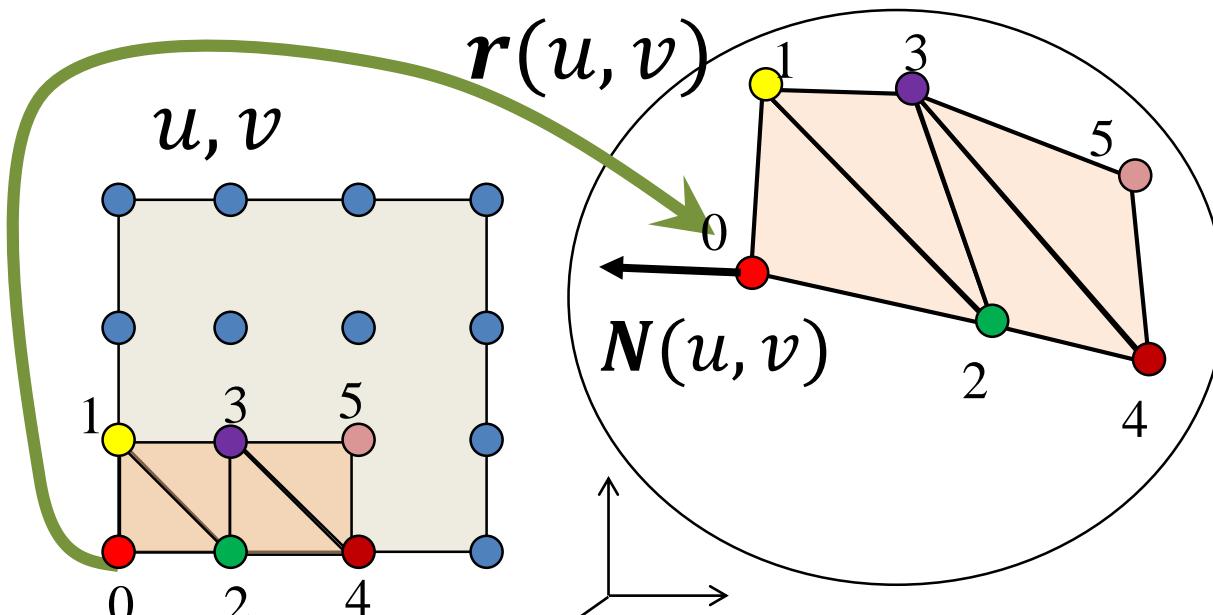
# Parametrikus felületek tesszellációja

„Paramétertérben szomszédos” pontokból háromszögek



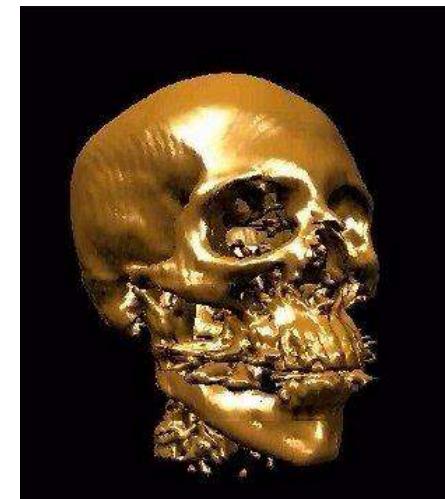
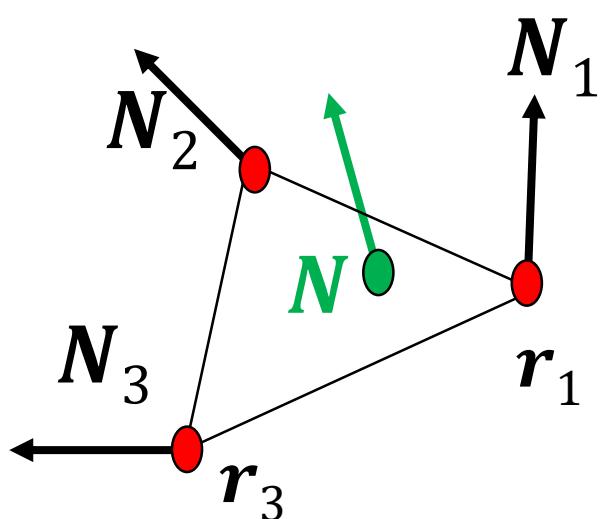
# Parametrikus felületek tesszellációja

„Paramétertérben szomszédos” pontokból háromszögek



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

Árnyaló normálok

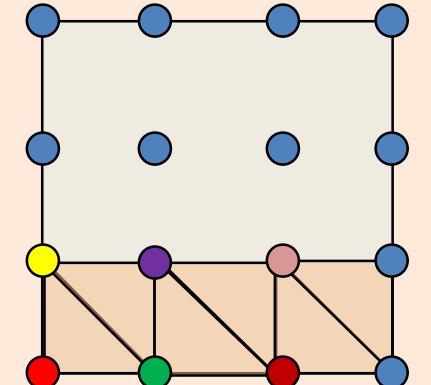


# Objektumok az GPU-nak

```
class Geometry {  
protected:  
    unsigned int vao, vbo;  
public:  
    Geometry() {  
        glGenVertexArrays(1, &vao);  
        glBindVertexArray(vao);  
        glGenBuffers(1, &vbo);  
        glBindBuffer(GL_ARRAY_BUFFER, vbo);  
    }  
    virtual void Draw() = 0;  
    ~Geometry() {  
        glDeleteBuffers(1, &vbo);  
        glDeleteVertexArrays(1, &vao);  
    }  
};
```

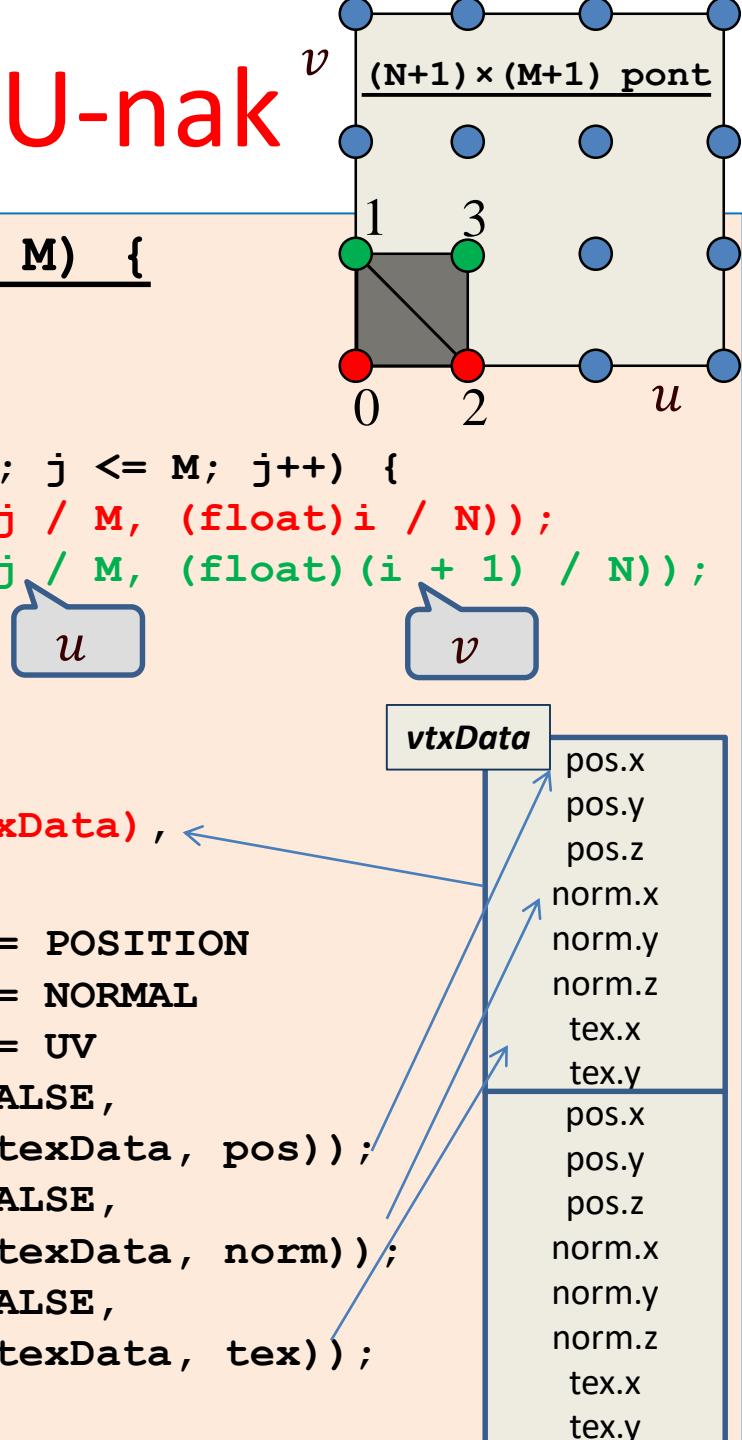
# Parametrikus felületek GPU-nak

```
class ParamSurface : public Geometry {
    unsigned int nVtxStrip, nStrips;
    virtual void eval(float u, float v, vec3& pos, vec3& norm)=0;
    struct VertexData {
        vec3 pos, norm;
        vec2 tex;
    };
    virtual VertexData GenVertexData(float u, float v) {
        VertexData vd;
        vd.tex = vec2(u, v); eval(u, v, vd.pos, vd.norm);
        return vd;
    }
public:
    void Create(int N, int M);
    void Draw() {
        glBindVertexArray(vao);
        for (int i = 0; i < nStrips; i++)
            glDrawArrays(GL_TRIANGLE_STRIP, i*nVtxStrip, nVtxStrip);
    }
};
```

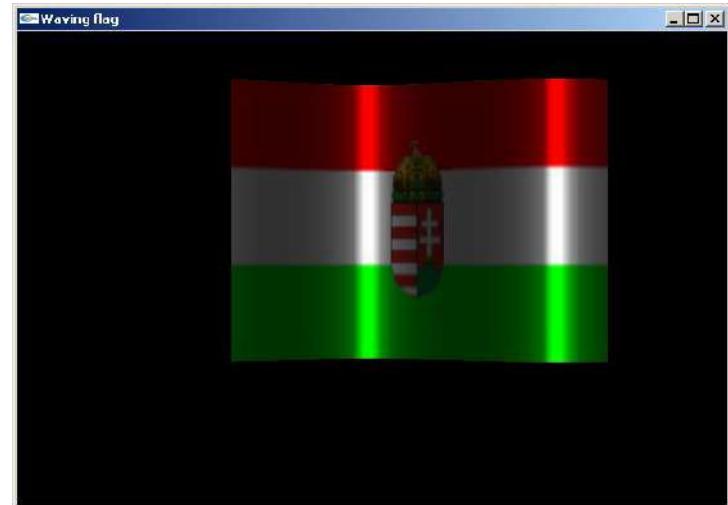
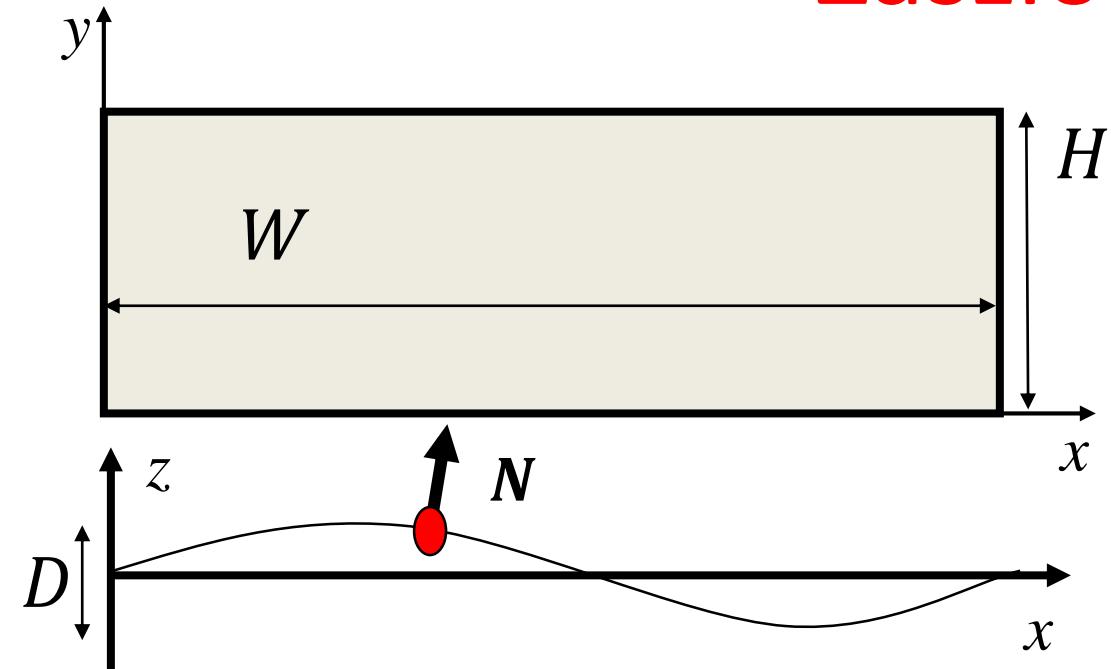


# Parametrikus felület GPU-nak

```
void ParamSurface::Create(int N, int M) {  
    nVtxStrip = (M + 1) * 2;  
    nStrips = N;  
    vector<VertexData> vtxData; // CPU-n  
    for (int i = 0; i < N; i++) for (int j = 0; j <= M; j++) {  
        vtxData.push_back(GenVertexData((float)j / M, (float)i / N));  
        vtxData.push_back(GenVertexData((float)j / M, (float)(i + 1) / N));  
    }  
    glBindVertexArray(vao);  
    glBindBuffer(GL_ARRAY_BUFFER, vbo);  
    glBufferData(GL_ARRAY_BUFFER,  
                 vtxData.size() * sizeof(VertexData),  
                 &vtxData[0], GL_STATIC_DRAW);  
  
    glEnableVertexAttribArray(0); // AttArr 0 = POSITION  
    glEnableVertexAttribArray(1); // AttArr 1 = NORMAL  
    glEnableVertexAttribArray(2); // AttArr 2 = UV  
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,  
                          sizeof(VertexData), (void*)offsetof(VertexData, pos));  
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,  
                          sizeof(VertexData), (void*)offsetof(VertexData, norm));  
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE,  
                          sizeof(VertexData), (void*)offsetof(VertexData, tex));  
}
```



# Zászló

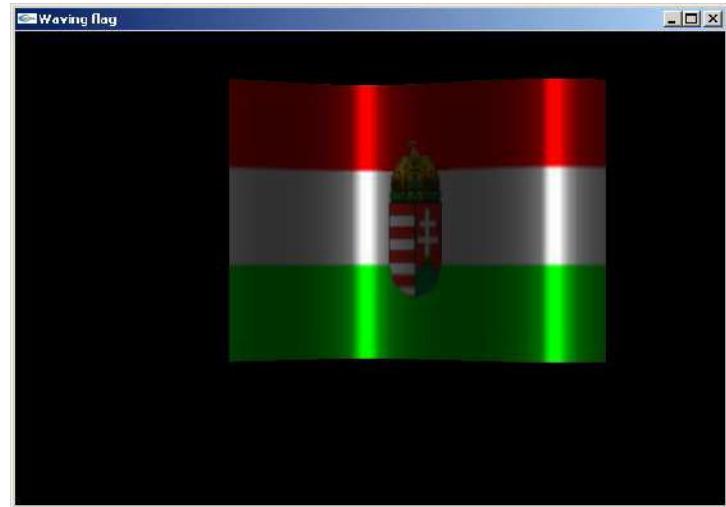


$$\mathbf{r}(u, v) = [uW, \quad vH, \quad D\sin(Ku + \alpha)]$$

$$\begin{aligned} \partial \mathbf{r} / \partial u &= \begin{vmatrix} i & j & k \\ W, & 0, & K D \cos(Ku + \alpha) \\ 0, & H, & 0 \end{vmatrix} \\ \partial \mathbf{r} / \partial v &= \begin{vmatrix} i & j & k \\ W, & 0, & K D \cos(Ku + \alpha) \\ 0, & H, & 0 \end{vmatrix} \end{aligned}$$

$$\mathbf{N}(u, v) = \partial \mathbf{r} / \partial u \times \partial \mathbf{r} / \partial v = [-HKD \cos(Ku + \alpha), 0, WH]$$

# Zászló



```
class Flag : public ParamSurface {
    float W, H, D, K, phase;
public:
    void eval(float u, float v, vec3& pos, vec3& norm) {
        float angle = u * K + phase;
        pos = vec3(u * W, v * H, D * sin(angle));
        norm = vec3(-K * D * cos(angle), 0, W);
    }
};
```

# Geometria a GPU-nak

- A GPU egy VAO-ba szervezett VBO-kban háromszögeket vár, amit a CPU program legkönnyebben paraméteres felületek tesszellációjával készíthet el.
- Elsődlegesen a paraméterteret tesszelláljuk.
- A csúcspontokhoz árnyaló normálisok és textúra koordináták is tartozhatnak.
- A normálvektor a parciális deriváltak vektoriális szorzata (lásd: Felület modellezés és Automatikus deriválás fejezetek).
- A tesszellált háromszögháló kompakt tárolásához találták ki a GL\_TRIANGLE\_STRIP-et.

*“The Matrix is everywhere. It is all around us. A prison for your mind.”*

*Morpheus*

# Inkrementális 3D képszintézis

## 3. Transzformációk

Szirmay-Kalos László



# Transzformációk

Modellezési transzformáció:

$$[\mathbf{r}, 1] \mathbf{T}_{Model} = [\mathbf{r}_{world}, 1]$$

$$\mathbf{T}_{Model}^{-1} [\mathbf{N}, 0]^T = [\mathbf{N}_{world}, d]^T$$

Kamera transzformáció:

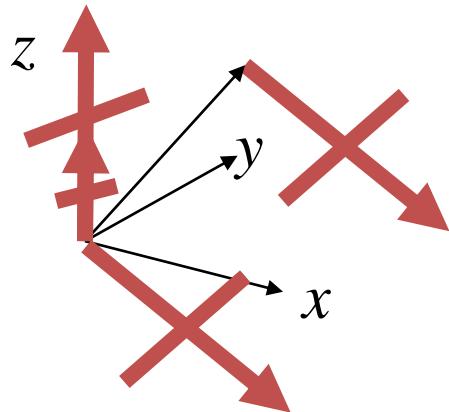
$$[\mathbf{r}_{world}, 1] \mathbf{T}_{View} = [\mathbf{r}_{camera}, 1]$$

Projektív transzformáció:

$$[\mathbf{r}_{camera}, 1] \mathbf{T}_{Proj} = [\mathbf{r}_{ndc}w, w]$$

MVP transzformáció:  $\mathbf{T}_{Model} \mathbf{T}_{View} \mathbf{T}_{Proj} = \mathbf{T}_{MVP}$

# Modellezési transzformáció



1. skálázás:  $s_x, s_y, s_z$
2. orientáció:  $d, \varphi$
3. pozíció:  $v$

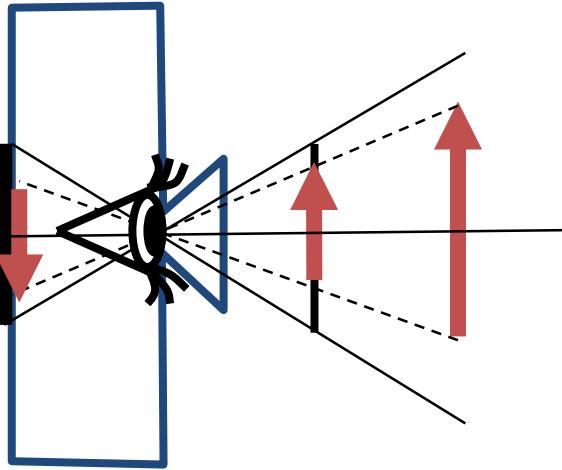
$$T_{4 \times 4} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i'_x & i'_y & i'_z & 0 \\ j'_x & j'_y & j'_z & 0 \\ k'_x & k'_y & k'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ v_x & v_y & v_z & 1 \end{bmatrix}$$

$$\mathbf{r}' = \mathbf{r} \cos(\varphi) + d(\mathbf{r} \cdot \mathbf{d})(1 - \cos(\varphi)) + \mathbf{d} \times \mathbf{r} \sin(\varphi)$$

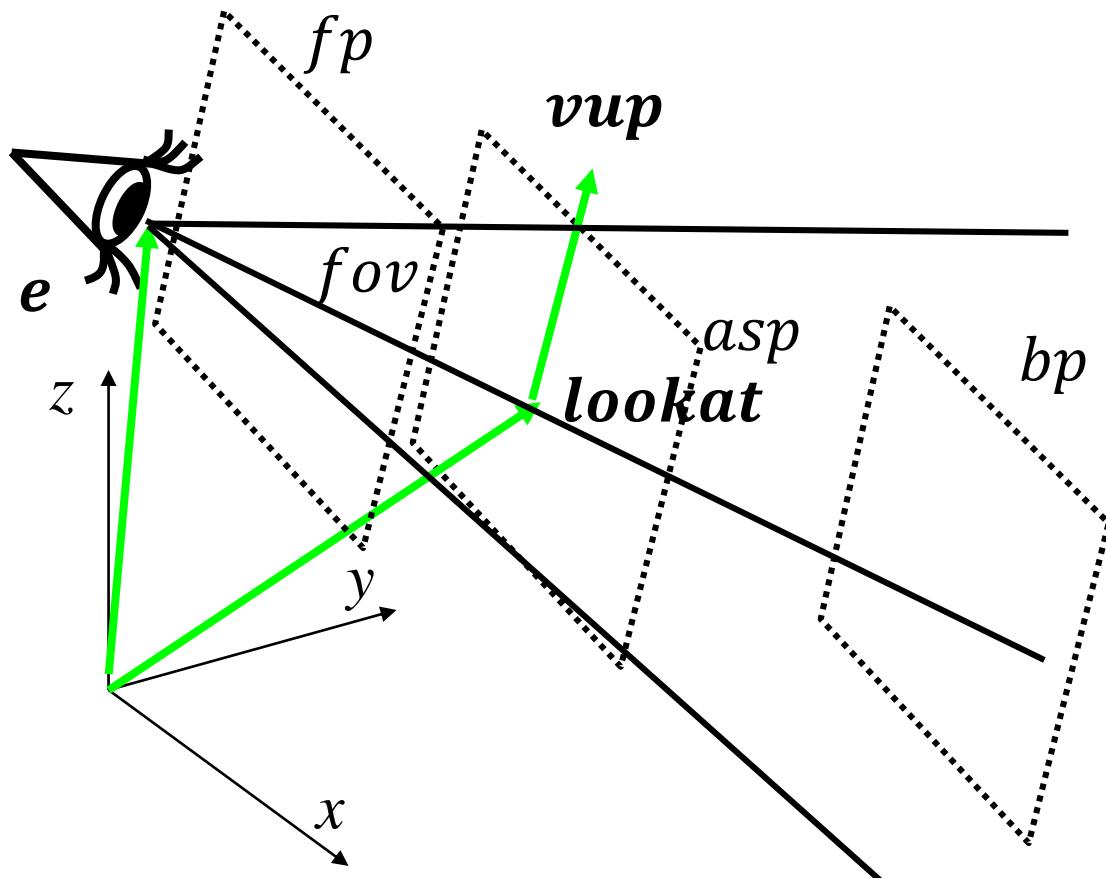
# Kamera modell



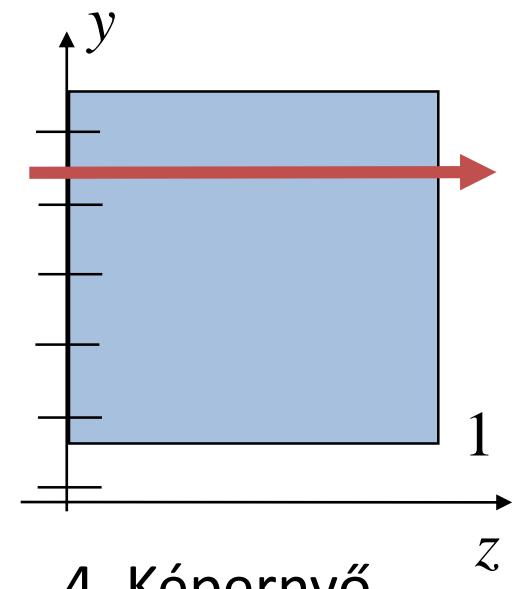
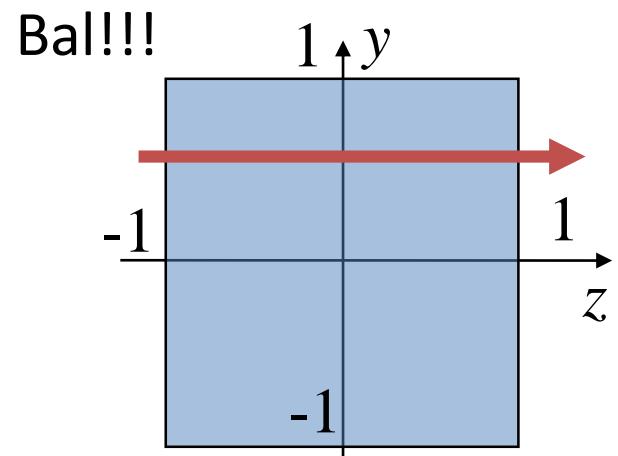
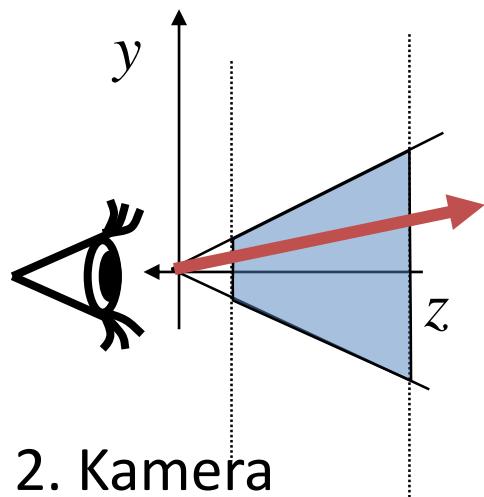
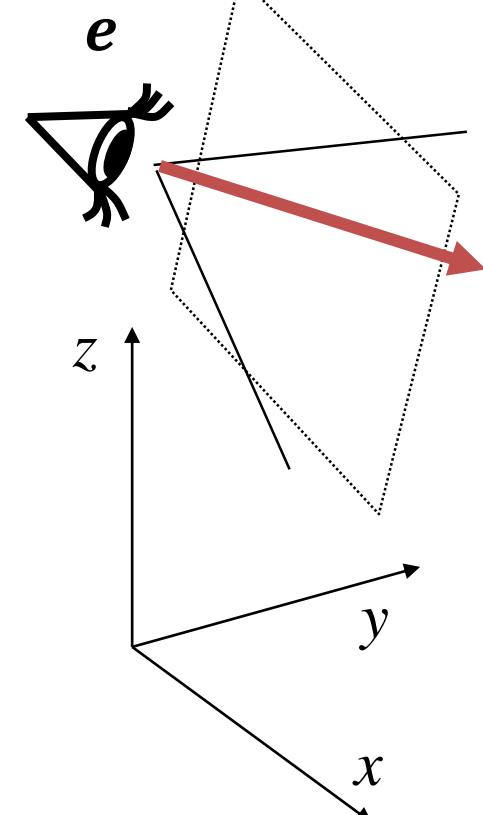
Nézeti  
téglalap

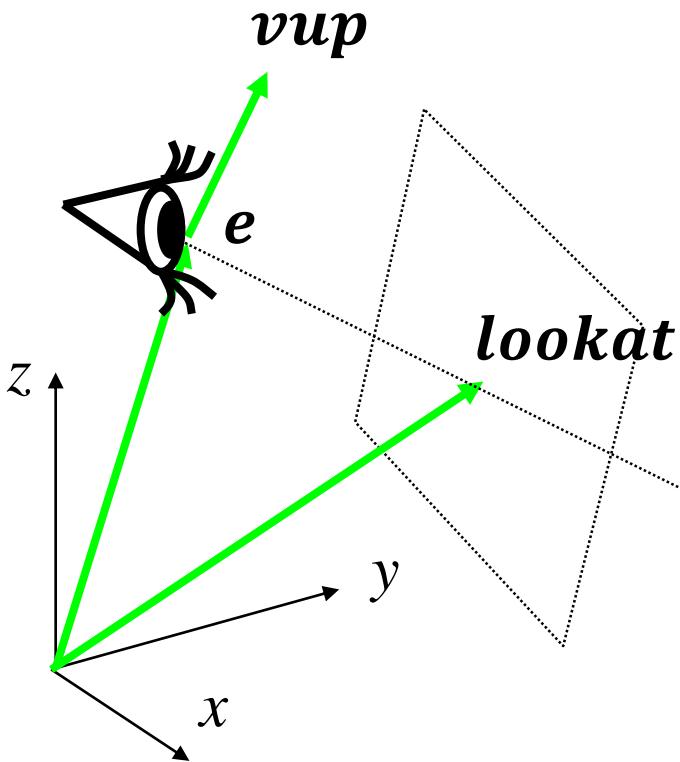


Mi: Camera obscura



# Világból a képernyőre





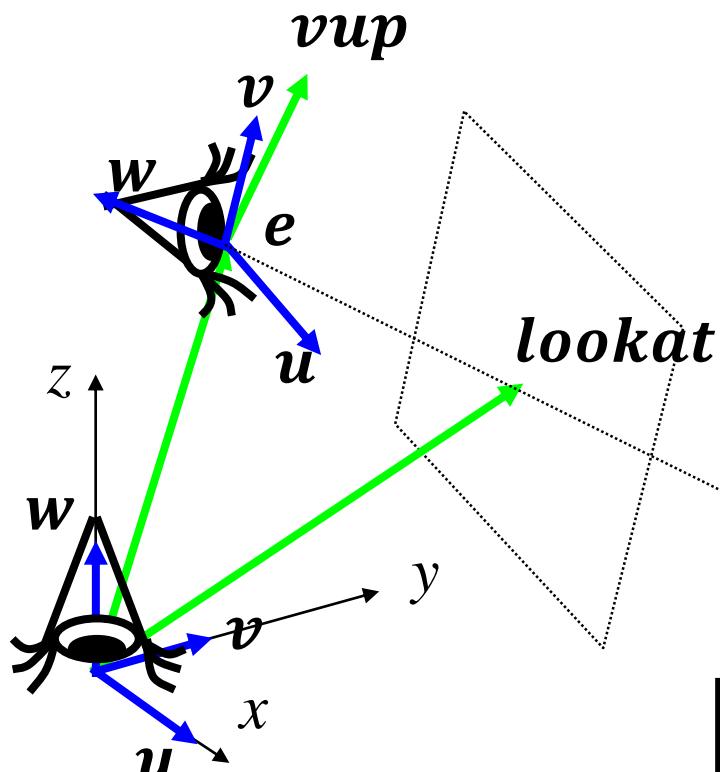
# View transformáció

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

$$u = (vup \times w) / |w \times vup|$$

$$v = w \times u$$

# View transformáció

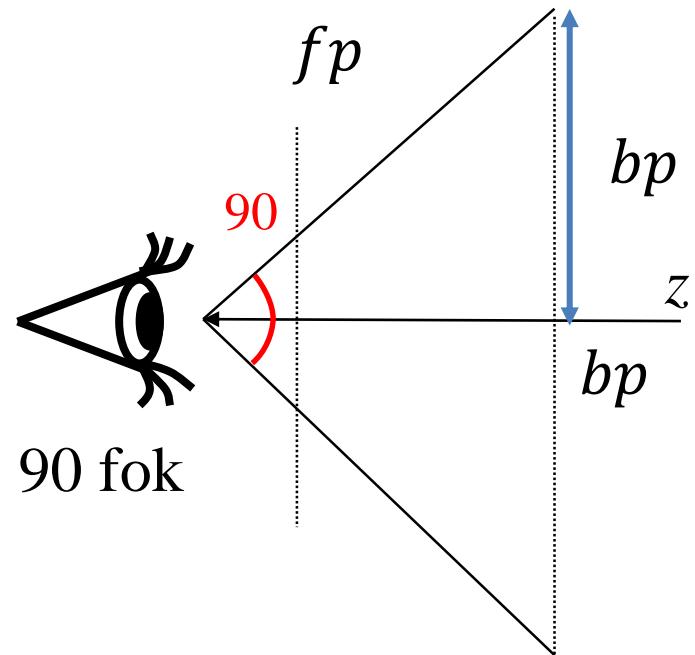
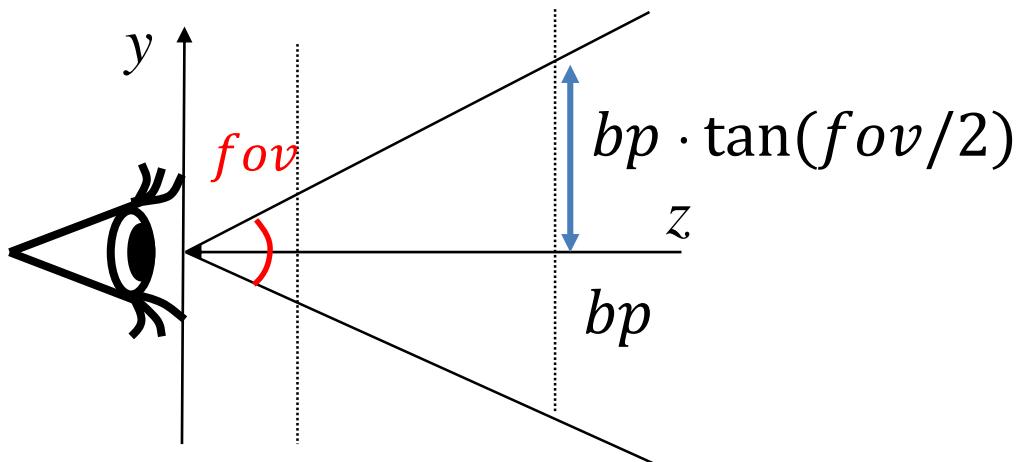


$$[x_c, y_c, z_c, 1] = [x, y, z, 1]$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -e_x & -e_y & -e_z & 1 \end{bmatrix} \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

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

# Látószög normalizálás



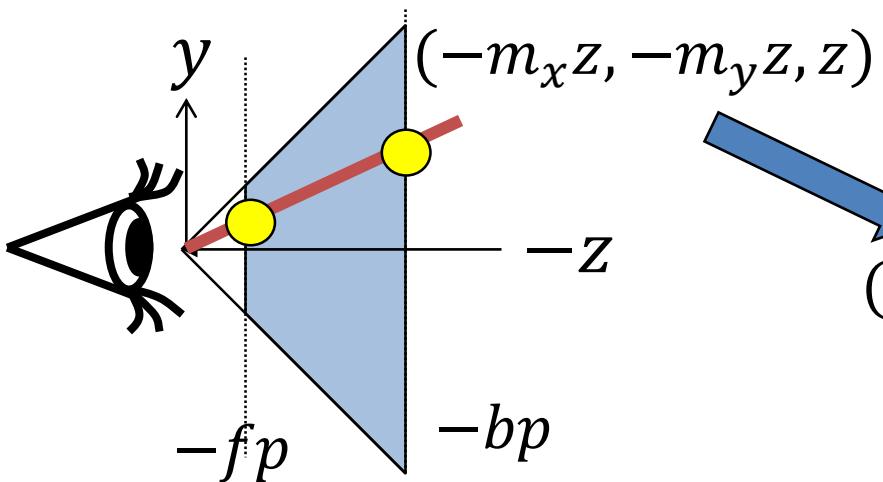
$$\begin{bmatrix} s_x & & \\ & s_y & \\ & & 1 \end{bmatrix} = \begin{bmatrix} 1/(\tan(fov/2) \text{ asp}) & 0 & 0 & 0 \\ 0 & 1/\tan(fov/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2D egyenes explicit egyenlete:

$$y = mx + b$$

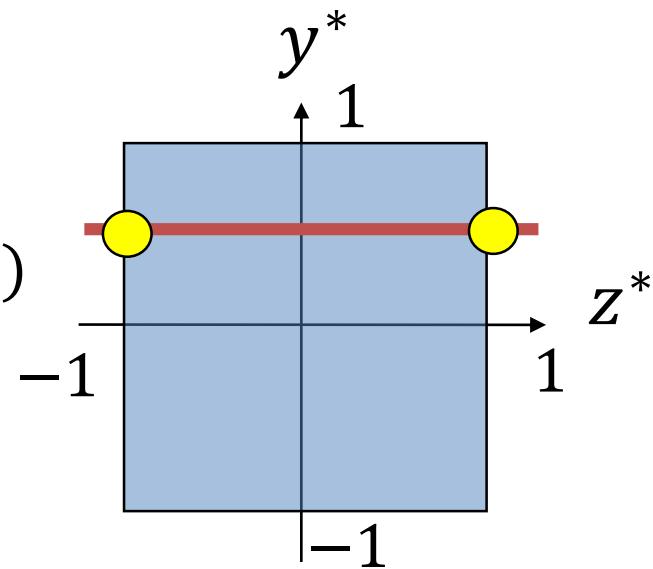
Origón átmenő:  $y = mx$

Vízszintes:  $y = b$



Normalizált kamera

## Normalizálás utáni perspektív transzformáció

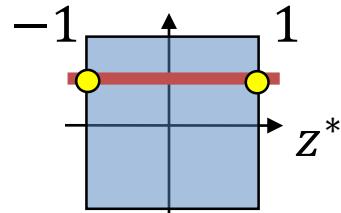
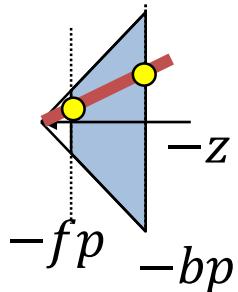


Normalizált képernyő: NDC

$$(-m_x z, -m_y z, z) \rightarrow (m_x, m_y, z^*)$$

$$[-m_x z, -m_y z, z, 1] \rightarrow [m_x, m_y, z^*, 1] \sim [-m_x z, -m_y z, -z z^*, -z]$$

# Perspektív transzformáció



$$T_{Persp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & -1 \\ 0 & 0 & \beta & 0 \end{bmatrix}$$

$$[-m_x z, -m_y z, z, 1] \cdot [-m_x z, -m_y z, -zz^*, -z]$$

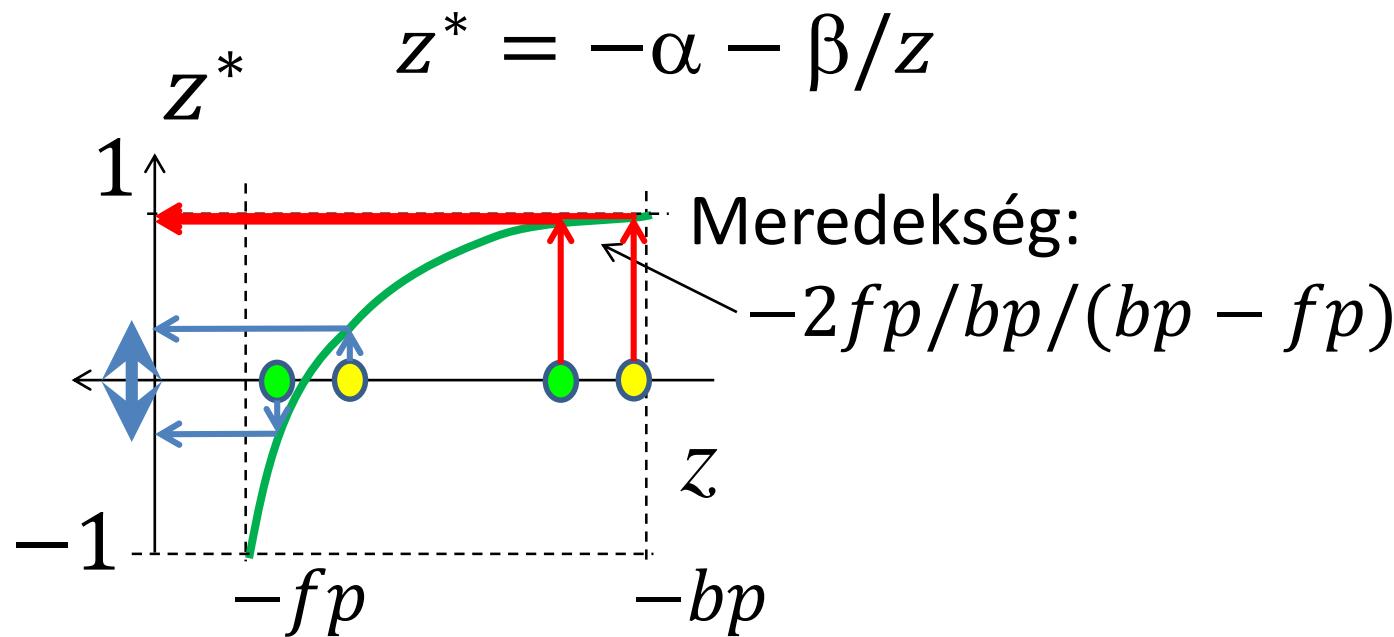
$$-zz^* = \alpha z + \beta \quad \rightarrow$$

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

$$\begin{aligned} fp(-1) &= \alpha(-fp) + \beta \\ bp(1) &= \alpha(-bp) + \beta \end{aligned}$$

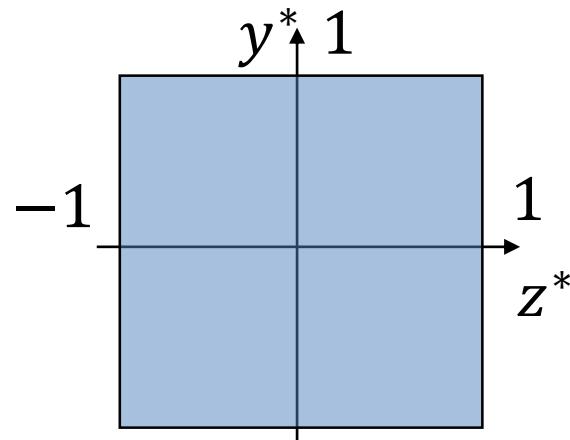
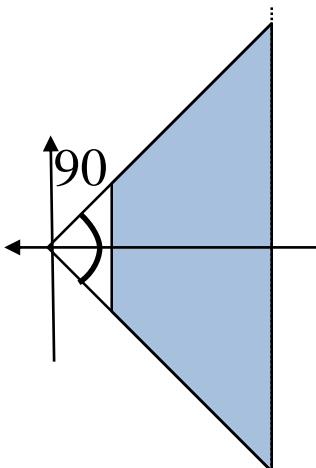
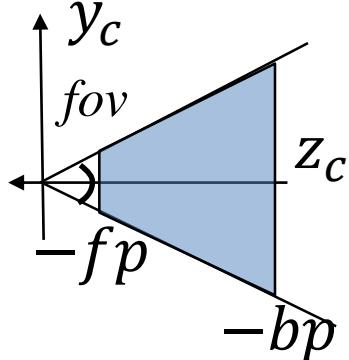
$$\begin{aligned} \alpha &= -(fp + bp)/(bp - fp) \\ \beta &= -2fp \cdot bp/(bp - fp) \end{aligned}$$

# Z-fighting



$fp/bp$  nem lehet kicsi!

# Projekció (perspektív) transzformáció



$$T_{Proj} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & -(fp + bp)/(bp - fp) & -1 \\ 0 & 0 & -2fp \cdot bp/(bp - fp) & 0 \end{bmatrix}$$

$$[x_c, y_c, z_c, 1] \cdot [X, Y, Z, w]$$

Perspektív torzítás = homogén osztás

$$(x^*, y^*, z^*) = (X/w, Y/w, Z/w)$$

$w = -z_c$

# Camera osztály

```
class Camera {
    vec3 wEye, wLookat, wVup; // extrinsic parameters
    float fov, asp, fp, bp;   // intrinsic parameters
public:
    mat4 V() { // view matrix
        vec3 w = normalize(wEye - wLookat);
        vec3 u = normalize(cross(wVup, w));
        vec3 v = cross(w, u);
        return TranslateMatrix(-wEye) * mat4( 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);
    }
    mat4 P() { // projection matrix
        float sy = 1/tanf(fov/2);
        return mat4(sy/asp, 0,         0,         0,
                    0,         sy,         0,         0,
                    0,         0, -(fp+bp)/(bp-fp), -1,
                    0,         0, -2*fp*bp/(bp-fp),  0);
    }
};
```

# Transzformációk előkészítése a CPU-n

```
void Draw() {
    mat4 M = ScaleMatrix(scale) *
              RotationMatrix(rotAng, rotAxis) *
              TranslateMatrix(pos);
    mat4 Minv = TranslateMatrix(-pos) *
                RotationMatrix(-rotAngle, rotAxis) *
                ScaleMatrix(1/scale);

    mat4 MVP = M * camera.V() * camera.P();

    shader->setUniform(M, "M");
    shader->setUniform(Minv, "Minv");
    shader->setUniform(MVP, "MVP");

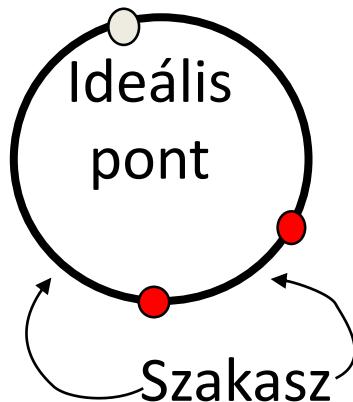
    glBindVertexArray(vao);
    glDrawArrays( . . . );
}
```

# Transzformációk végrehajtása a GPU vertex árnyalójában

```
uniform mat4 M, Minv, MVP;  
layout(location = 0) in vec3 vtxPos;  
layout(location = 1) in vec3 vtxNorm;  
  
out vec4 color;  
  
void main() {  
    gl_Position = vec4(vtxPos, 1) * MVP;  
    vec4 wPos = vec4(vtxPos, 1) * M;  
    vec4 wNormal = Minv * vec4(vtxNorm, 0);  
    color = Illumination(wPos, wNormal);  
}
```

# 3D vágás homogén koordinátákban (GPU)

## Homogén koordinátákban kell vágni



$$[X(t), Y(t), Z(t), w(t)] =$$

$$[X_1, Y_1, Z_1, w_1](1 - t) + [X_2, Y_2, Z_2, w_2]t$$

$$\begin{aligned} -1 < x = X/w < 1 \\ -1 < y = Y/w < 1 \\ -1 < z = Z/w < 1 \end{aligned}$$

$$w > 0$$

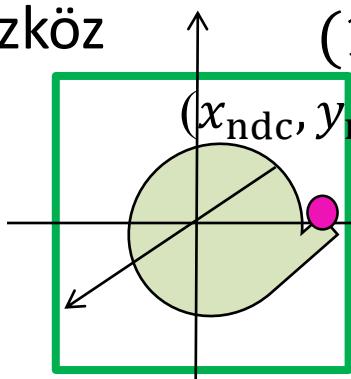
$$\begin{aligned} -w < X < w \\ -w < Y < w \\ -w < Z < w \end{aligned}$$

Mert  $w = -z_c > 0$  a szem előtt

# Viewport transzformáció: Normalizáltból képernyő koordinátákba (GPU)

Normalizált

eszköz



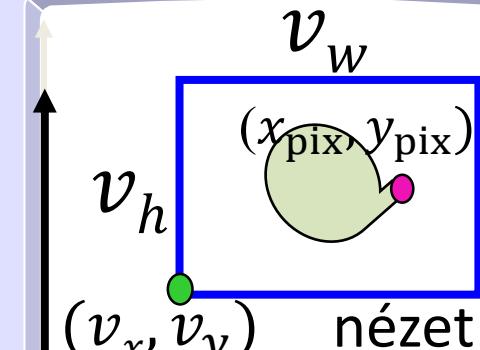
(1,1,1)  
 $(x_{ndc}, y_{ndc}, z_{ndc})$

$$x_{pix} = v_w(x_{ndc} + 1)/2 + v_x$$
$$y_{pix} = v_h(y_{ndc} + 1)/2 + v_y$$

(-1, -1, -1)

$$z_{pix} = (z_{ndc} + 1)/2$$

Képernyő



$v_w$   
 $(x_{pix}, y_{pix})$

$v_h$   
 $(v_x, v_y)$

nézet

Egység=pixel

-

# Transzformációs csővezeték

- Transzformációk 4x4-es mátrixok szorzata:
  - Modell, Modell-inverz, MVP
- A CPU-n az egyes transzformációkat külön-külön számítjuk ki, majd a szorzatot adjuk át a csúcspont árnyalónak.
- A transzformációk homogén koordinátákat transzformálnak, a vágást is homogén koordinátákban kell megcsinálni.
- A vágás után visszatérhetünk Descartes koordinátákhoz.

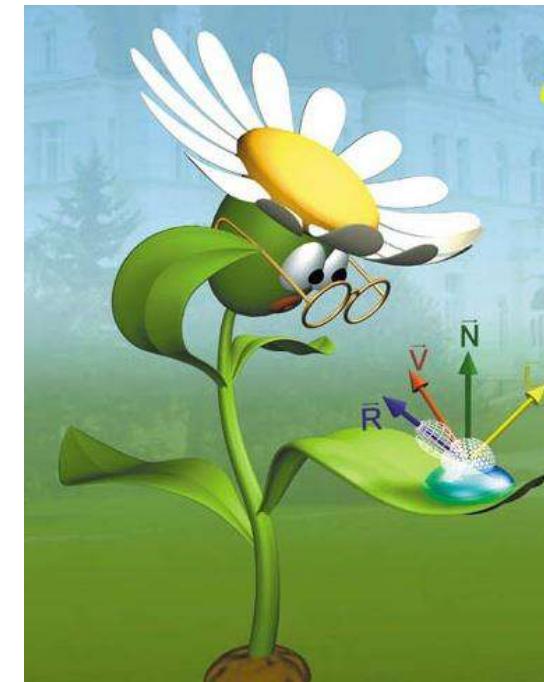
*"If you can't make it good,  
at least make it look good."*

*Bill Gates*

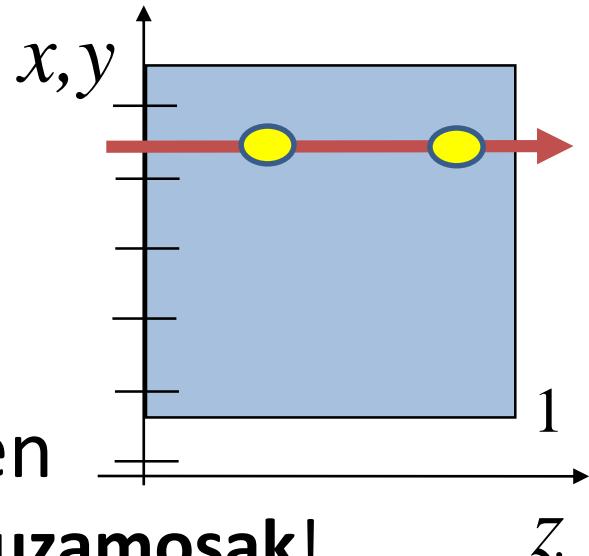
# Inkrementális 3D képszintézis

## 4. Láthatóság és árnyalás

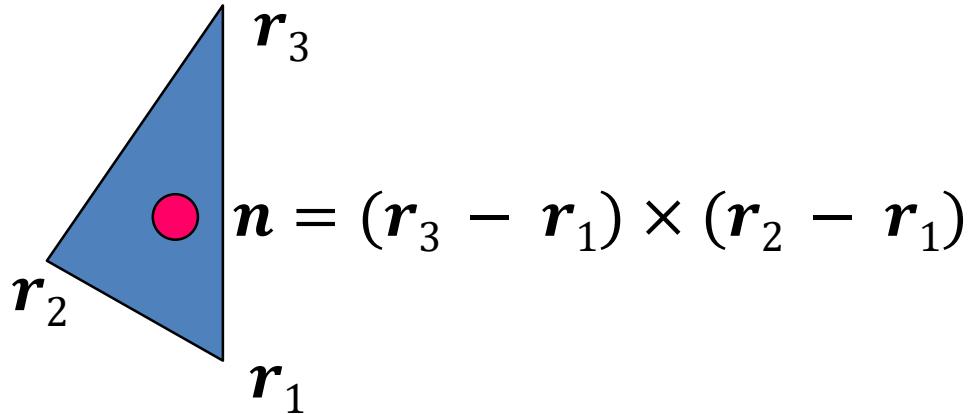
Szirmay-Kalos László



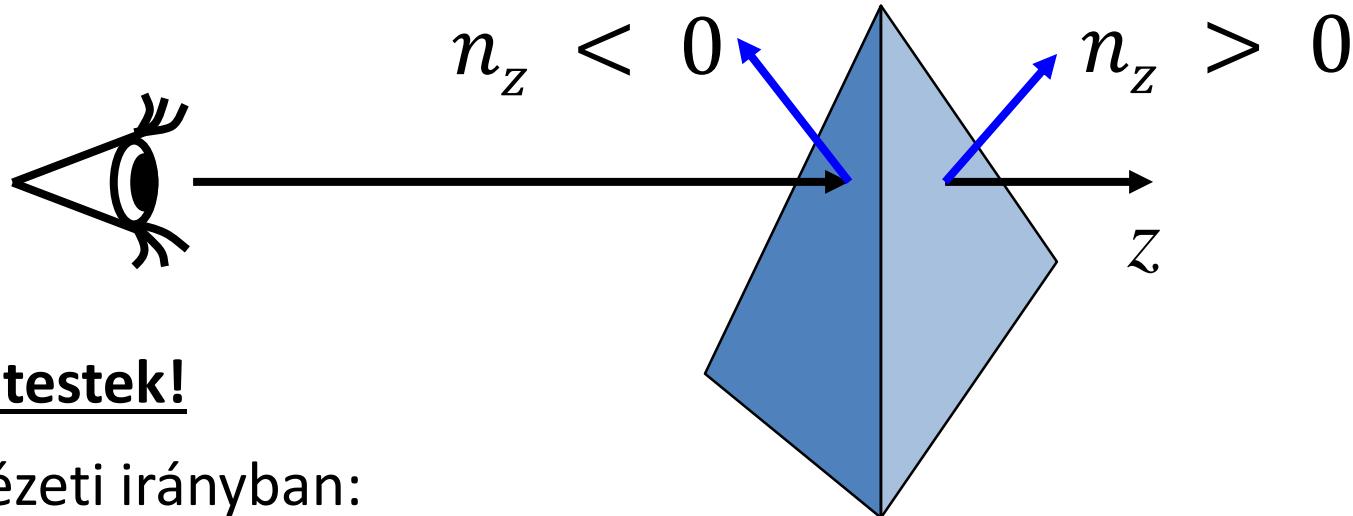
# Takarás



- Képernyő koordinátarendszerben
  - **vetítősugarak a  $z$  tengellyel párhuzamosak!**
  - Sugárparaméter =  $z$  koordináta
  - $(x, y, z)$  pont az  $(x, y)$  pixelben látszik
- Objektumtér algoritmusok (folytonos):
  - láthatóság számítás nem függ a felbontástól
- Képtér algoritmusok (diszkrét):
  - mi látszik egy pixelben
  - Sugárkövetés ilyen volt!



## Hátsólab eldobás: back-face culling



### Valódi 3D testek!

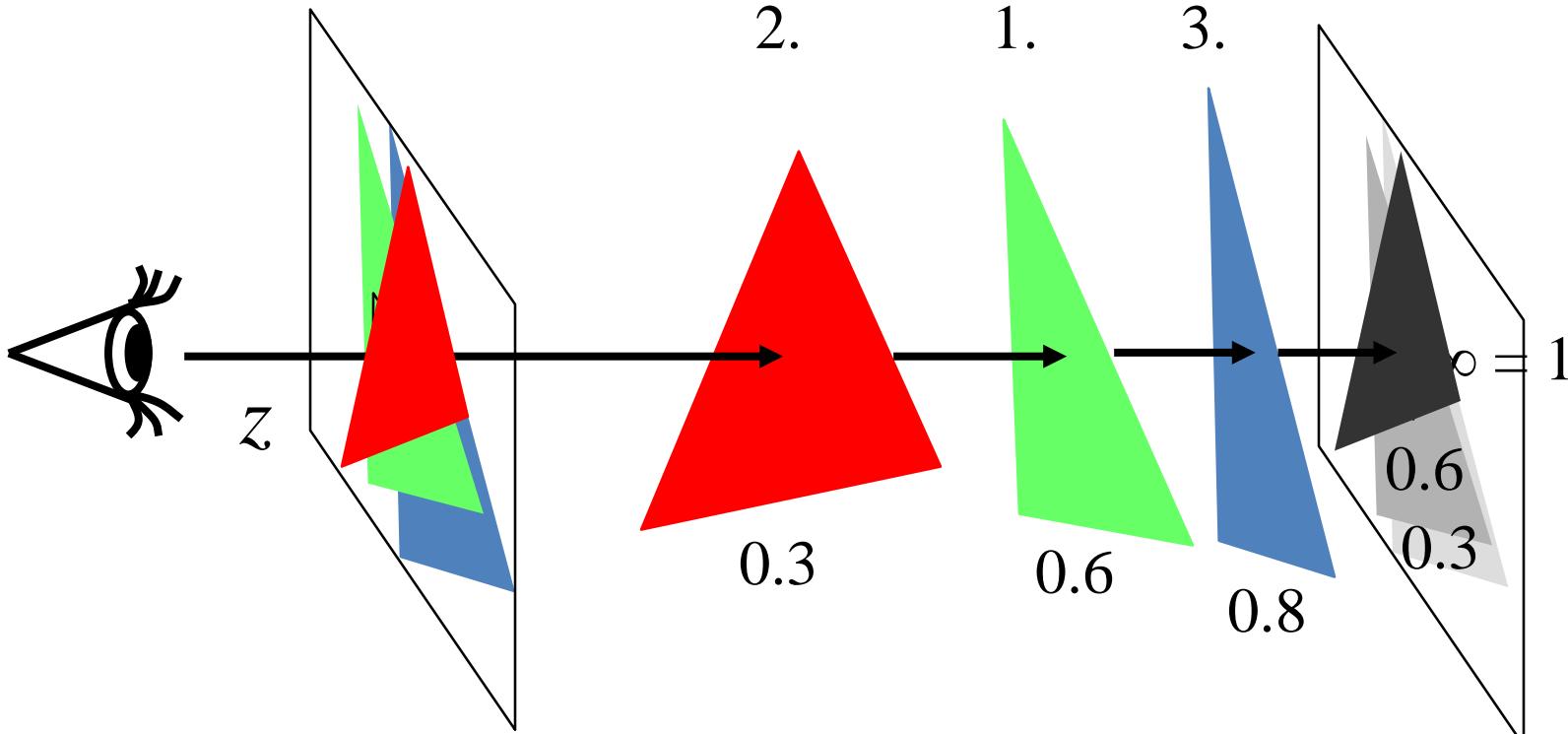
Lapok a nézeti irányban:

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

Feltételezés:

**Ha kívülről, akkor csúcsok óramutatóval megegyező körüljárásúak**

# Z-buffer algoritmus

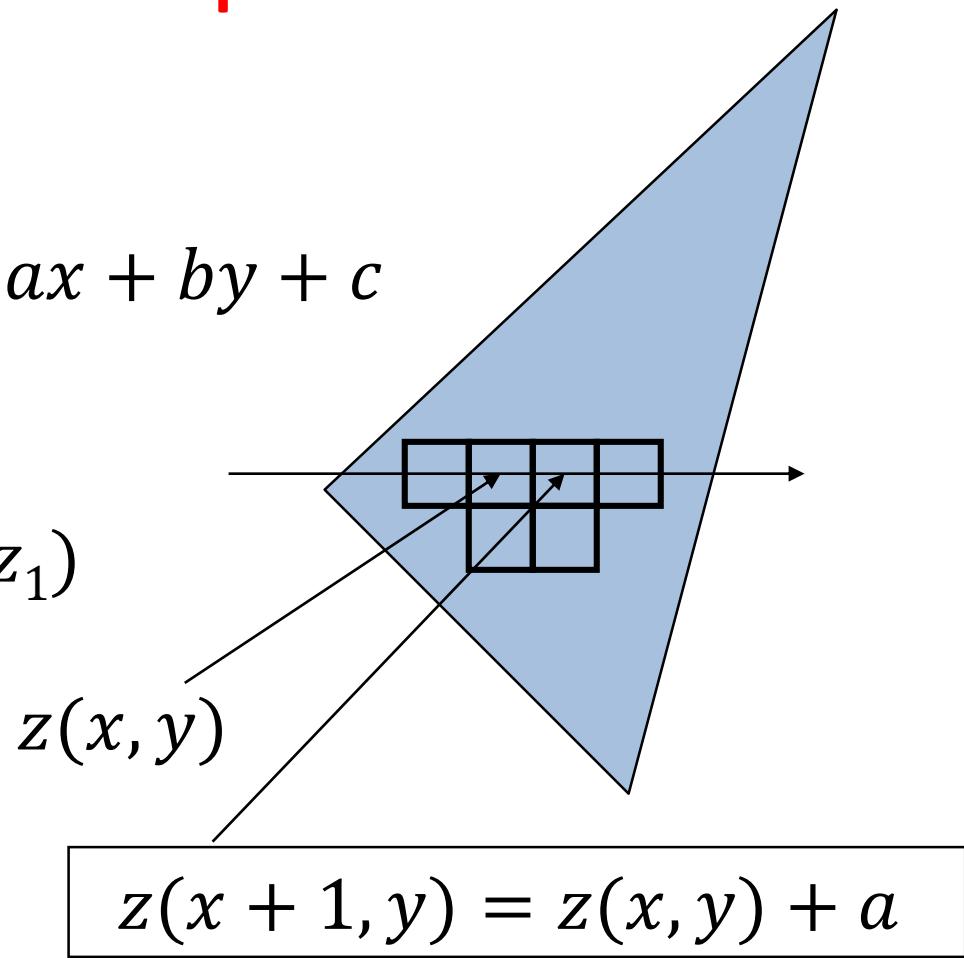
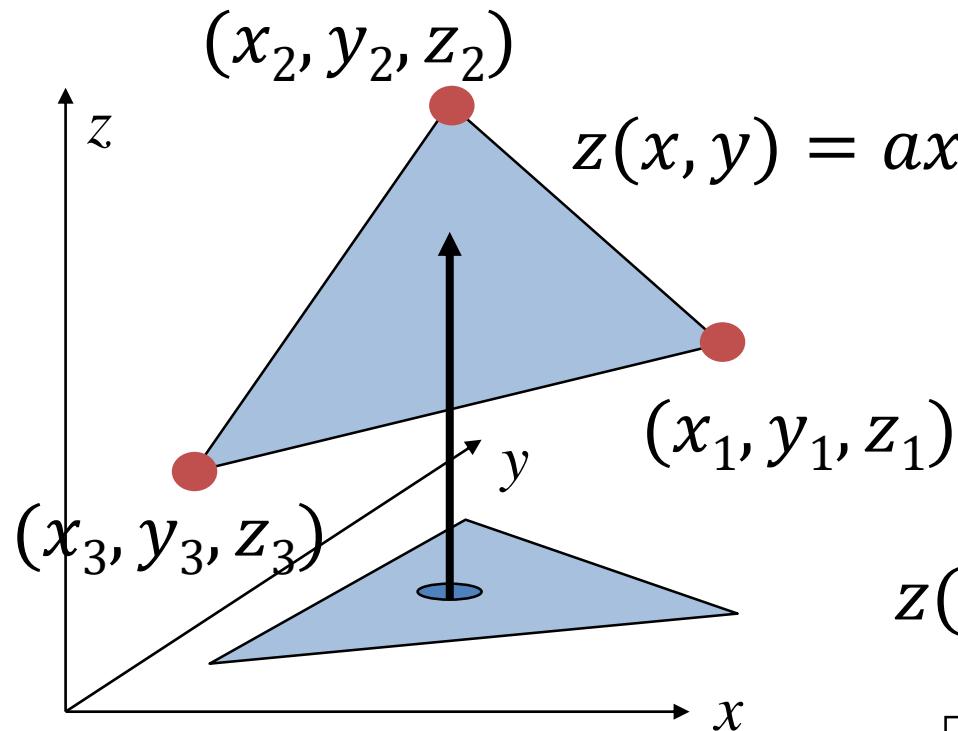


Szín buffer

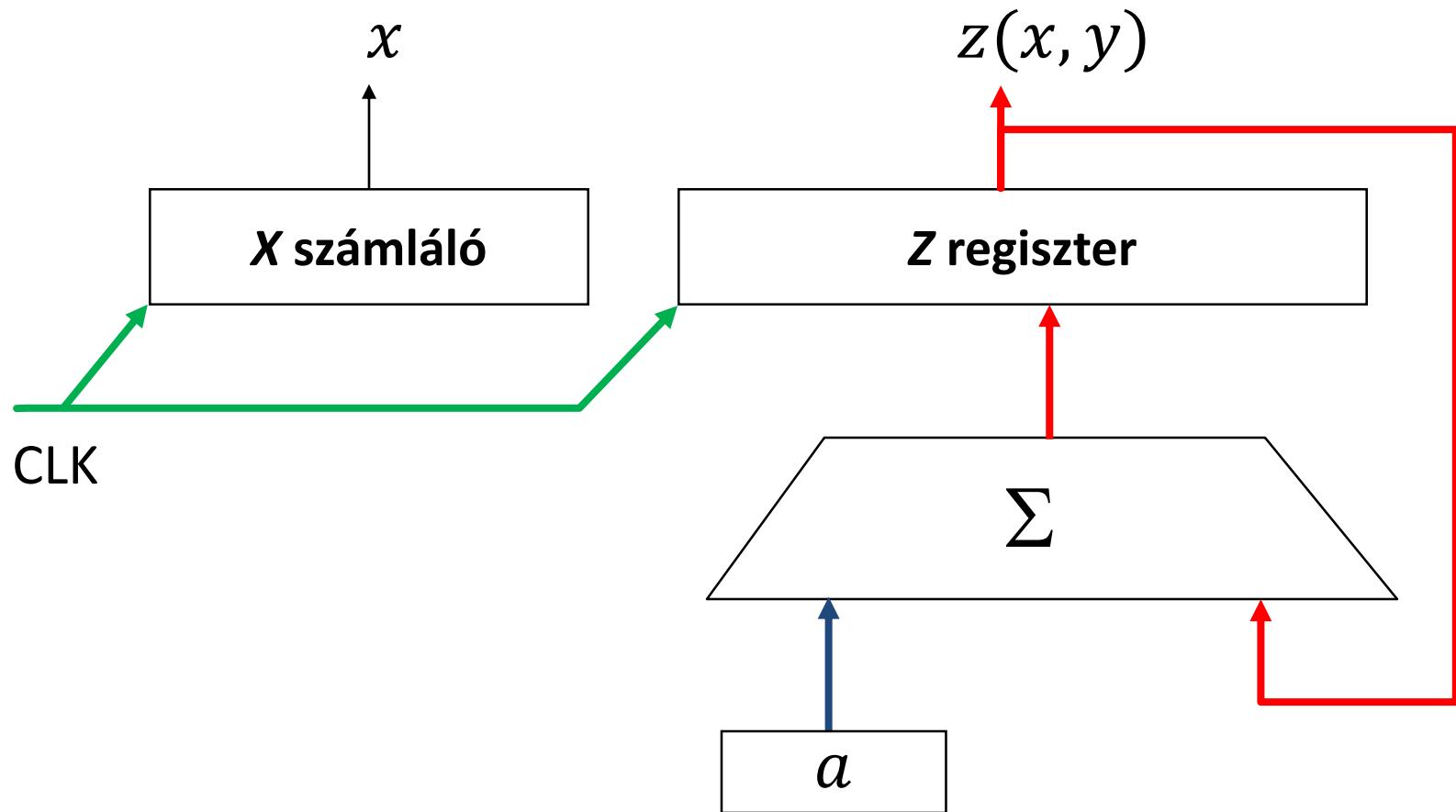
Mélység buffer  
Z-buffer

= sugárparaméterek  
pixelenként

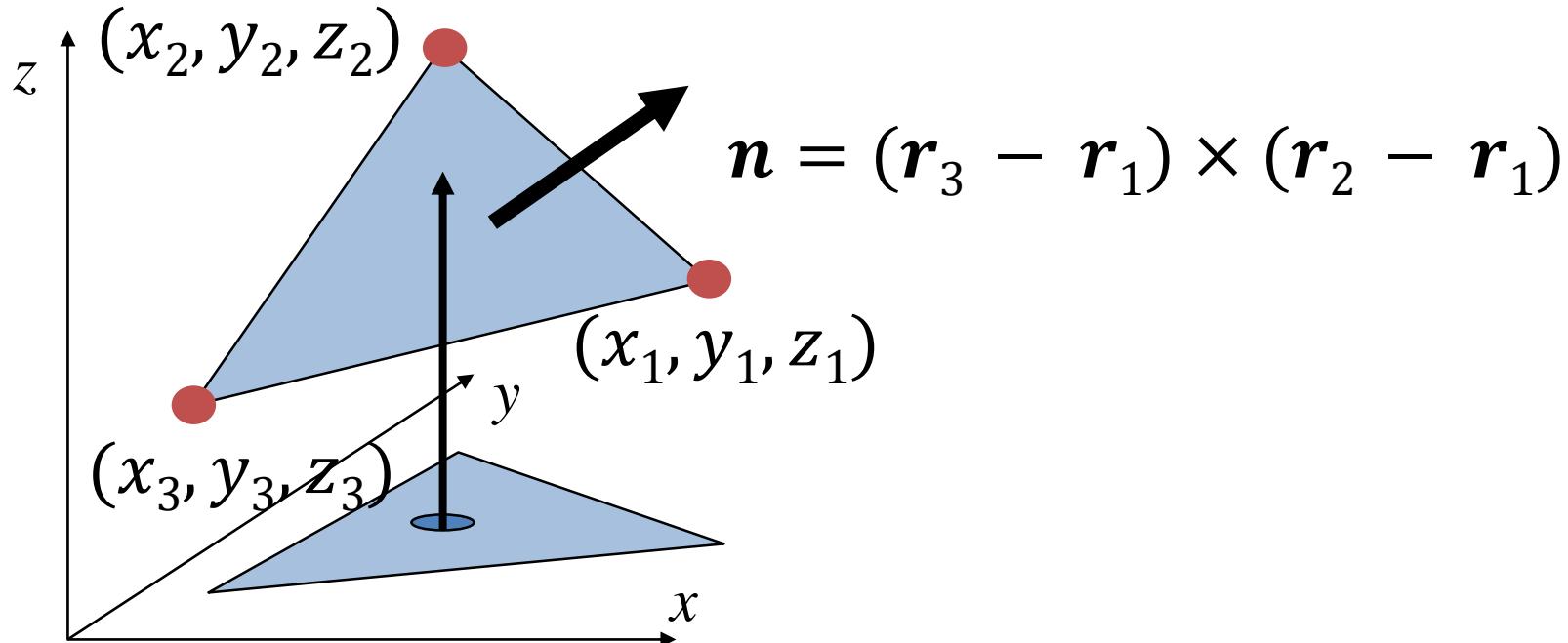
# Z: lineáris interpoláció



# Z-interpolációs hardver



# Triangle setup



$$z(x, y) = ax + by + c$$
$$n_x x + n_y y + n_z z + d = 0$$



$$a = \frac{-n_x}{n_z}$$

# Takarás OpenGL-ben

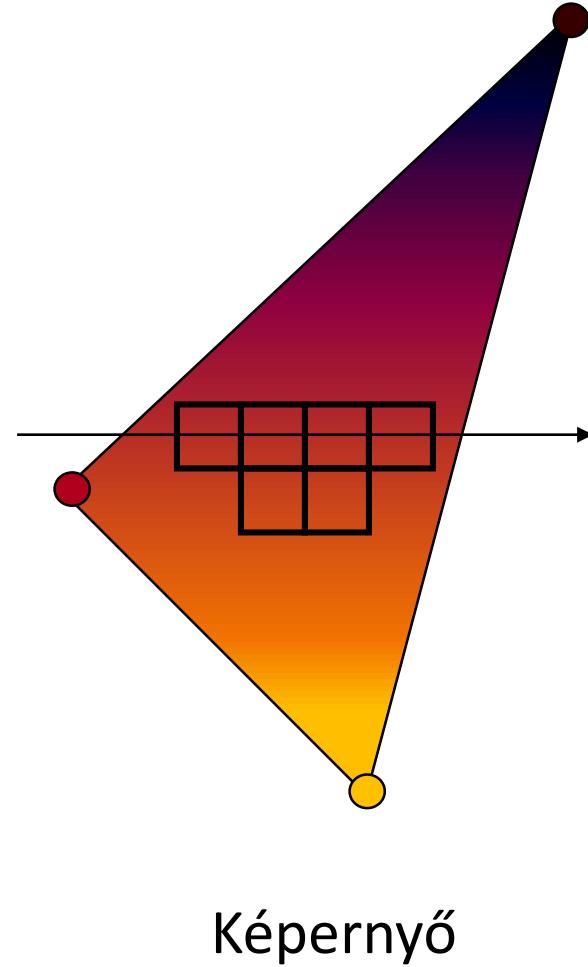
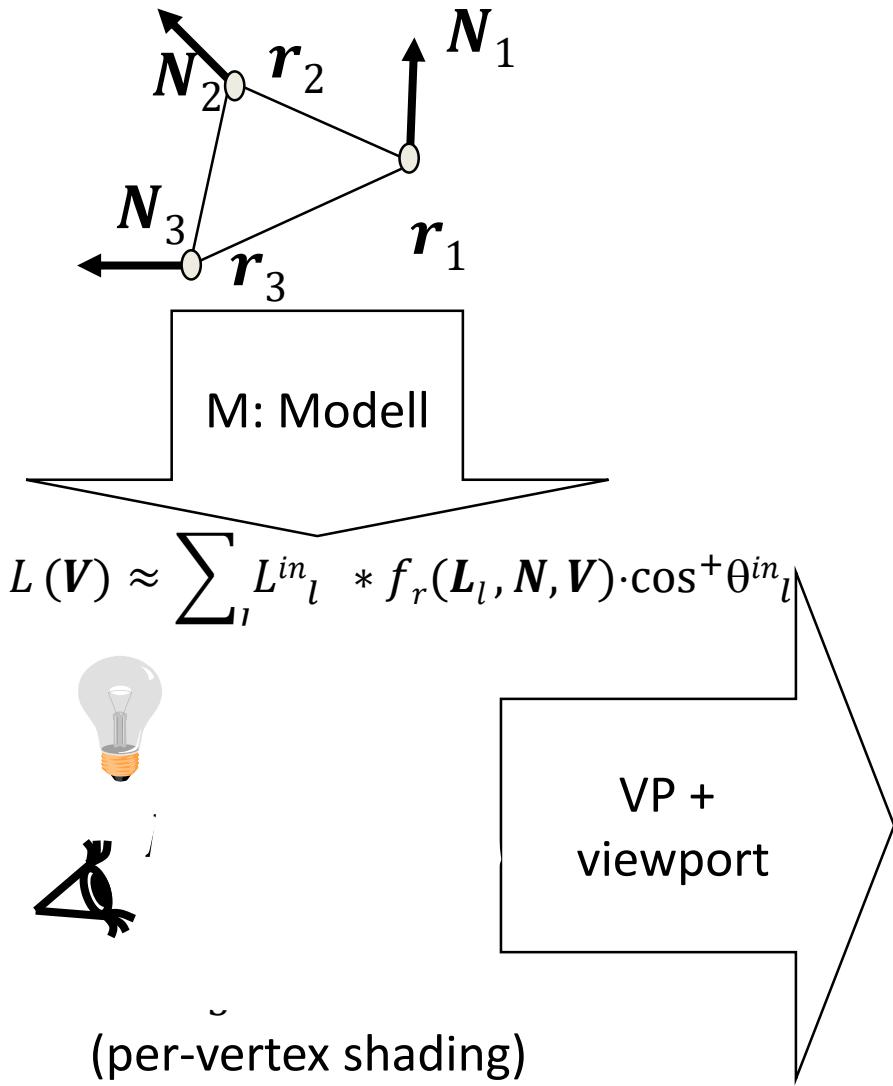
```
int main(int argc, char * argv[]) {  
    ...  
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE |  
                        GLUT_DEPTH);  
    glEnable(GL_DEPTH_TEST); // z-buffer is on  
    glDisable(GL_CULL_FACE); // backface culling is off  
    ...  
}  
  
void onDisplay() {  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
  
rajzolás...  
  
    glutSwapBuffers(); // exchange the two buffers  
}
```

# Árnyalás: Lokális illumináció árnyék nélkül

$$L(\mathbf{V}) \approx \sum_l L^{in}_l * f_r(\mathbf{L}_l, \mathbf{N}, \mathbf{V}) \cdot \cos^+ \theta^{in}_l$$

- Koherencia: ne minden pixelenként
- **Csúcspontronként**: belül az  $L$  „szín” interpolációja:  
Gouraud árnyalás (per-vertex shading)
- **Pixelenként**: belül a Normál (View, Light) vektort interpoláljuk:  
Phong árnyalás (per-pixel shading)

# Per-vertex (Gouraud) árnyalás



# Per-vertex shading: Vertex shader

```
uniform mat4 MVP, M, Minv; // MVP, Model, Model-inverse
uniform vec4 kd, ks, ka; // diffuse, specular, ambient ref
uniform float shine; // shininess for specular ref
uniform vec4 La, Le; // ambient and point sources
uniform vec4 wLiPos; // pos of light source in world
uniform vec3 wEye; // pos of eye in world

layout(location = 0) in vec3 vtxPos; // pos in modeling space
layout(location = 1) in vec3 vtxNorm; // normal in modeling space
out vec4 color; // computed vertex color

void main() {
    gl_Position = vec4(vtxPos, 1) * MVP; // to NDC
    vec4 wPos = vec4(vtxPos, 1) * M;
    vec3 L = normalize( wLiPos.xyz/wLiPos.w - wPos.xyz/wPos.w );
    vec3 V = normalize(wEye - wPos.xyz/wPos.w);
    vec4 wNormal = Minv * vec4(vtxNorm, 0);
    vec3 N = normalize(wNormal.xyz);
    vec3 H = normalize(L + V);
    float cost = max(dot(N, L), 0), cosd = max(dot(N, H), 0);
    color = ka * La + (kd * cost + ks * pow(cosd, shine)) * Le;
}
```

# Per-vertex shading: Pixel shader

```
in vec4 color;           // interpolated color of vertex shader
out vec4 fragmentColor; // output goes to frame buffer

void main() {
    fragmentColor = color;
}
```

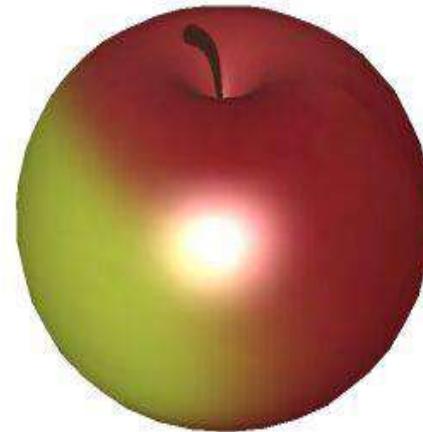


# (Henri) Gouraud árnyalás problémái

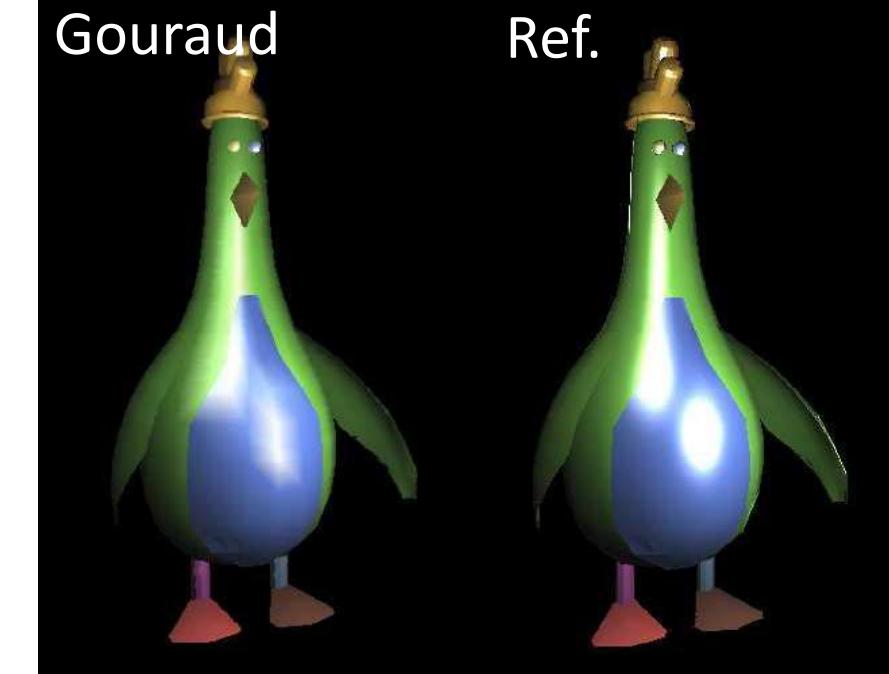
Gouraud



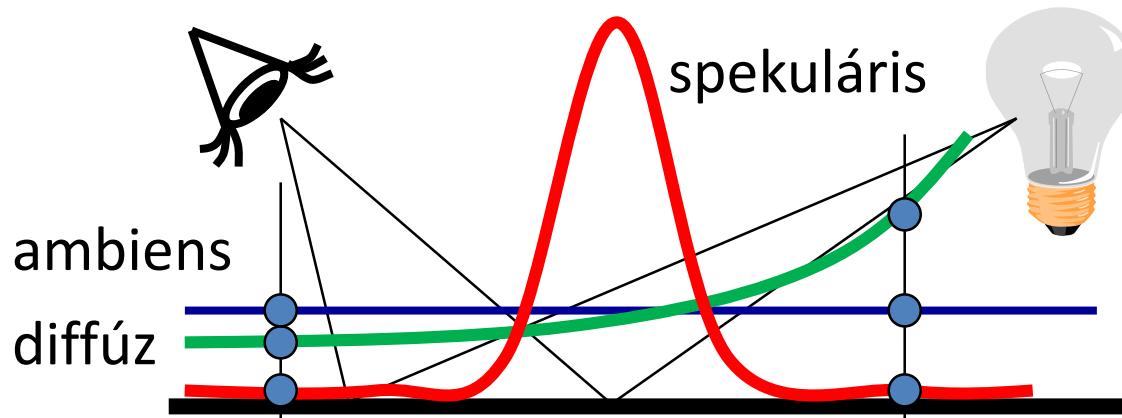
Ref.



Gouraud



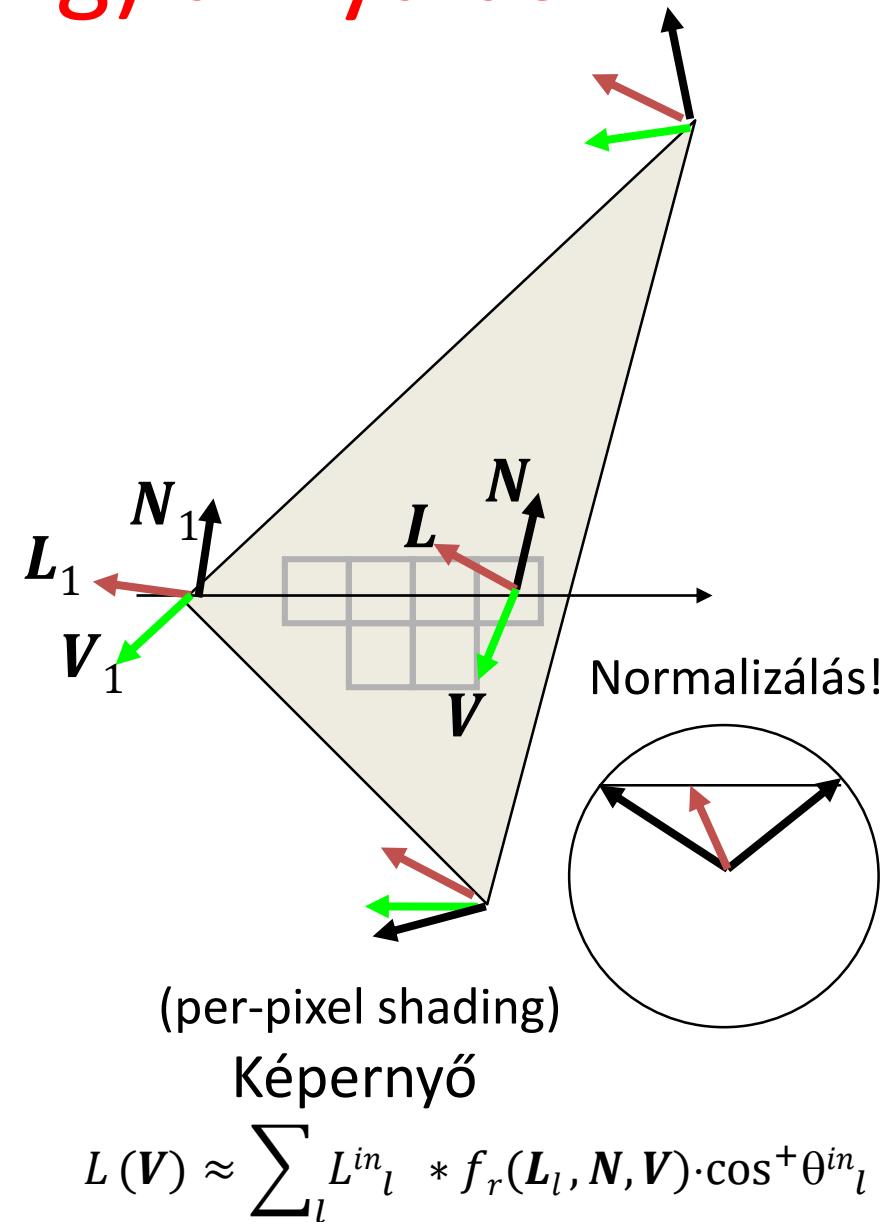
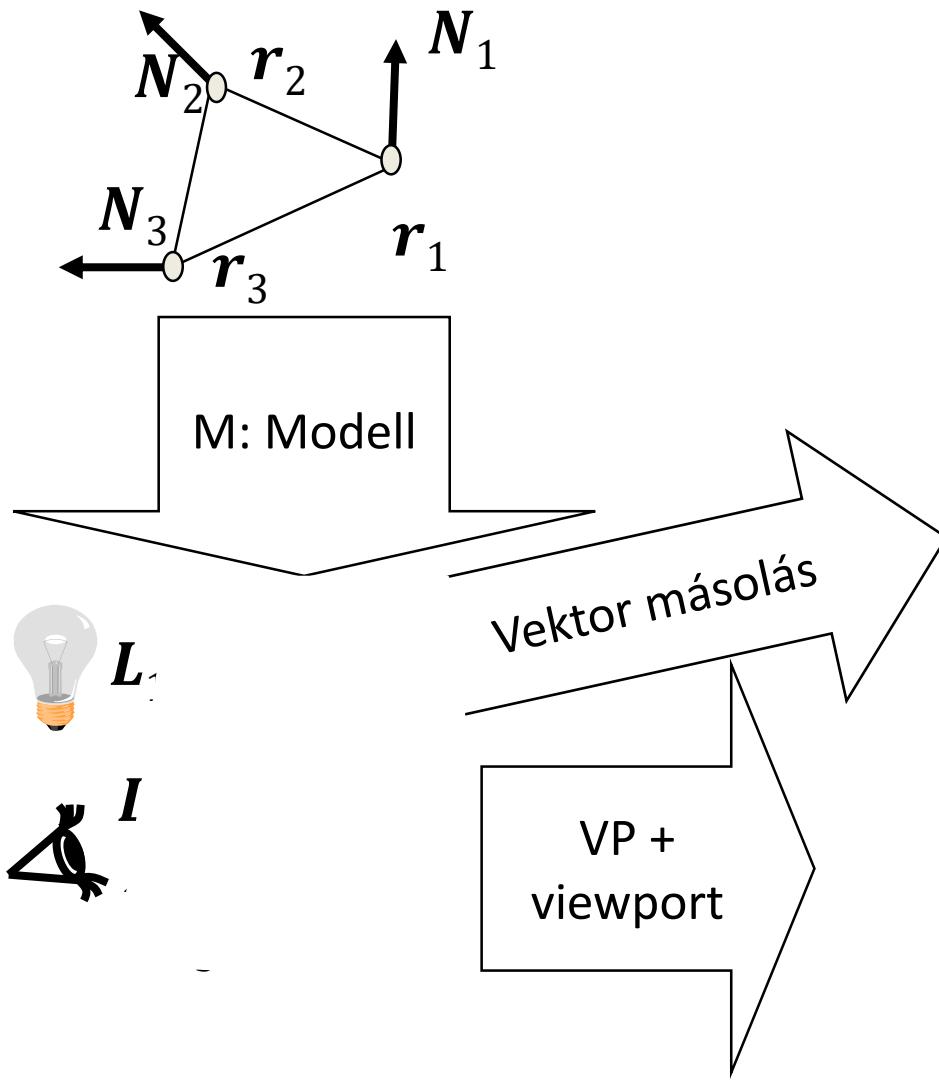
Ref.



## További bajok:

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

# Per-pixel (Phong) árnyalás



# Per-pixel shading: Vertex shader

```
uniform mat4    MVP, M, Minv; // MVP, Model, Model-inverse
uniform vec4    wLiPos;       // pos of light source
uniform vec3    wEye;        // pos of eye

layout(location = 0) in vec3 vtxPos; // pos in model sp
layout(location = 1) in vec3 vtxNorm;// normal in mod sp

out vec3 wNormal;           // normal in world space
out vec3 wView;             // view in world space
out vec3 wLight;            // light dir in world space

void main() {
    gl_Position = vec4(vtxPos, 1) * MVP; // to NDC

    vec4 wPos = vec4(vtxPos, 1) * M;
    wLight = wLiPos.xyz/wLiPos.w - wPos.xyz/wPos.w;
    wView = wEye - wPos.xyz/wPos.w;
    wNormal = (Minv * vec4(vtxNorm, 0)).xyz;
}
```

# Per-pixel shading: Pixel shader

```
uniform vec3 kd, ks, ka; // diffuse, specular, ambient ref
uniform float shine; // shininess for specular ref
uniform vec3 La, Le; // ambient and point source rad

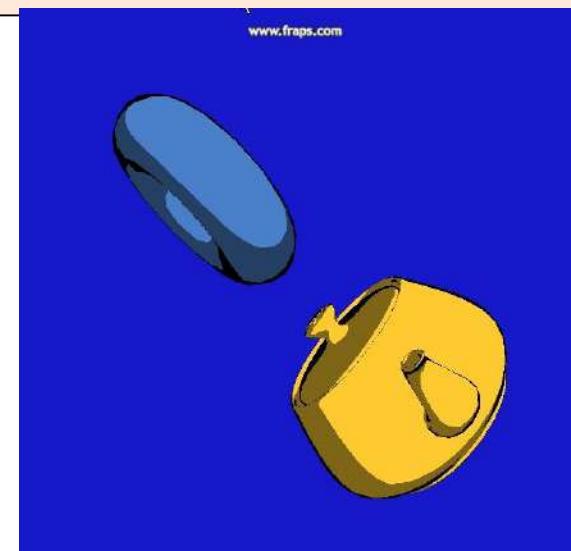
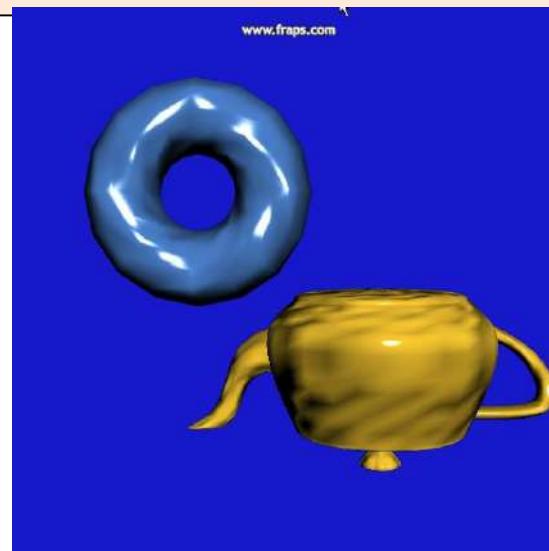
in vec3 wNormal; // interpolated world sp normal
in vec3 wView; // interpolated world sp view
in vec3 wLight; // interpolated world sp illum dir
out vec4 fragmentColor; // output goes to frame buffer

void main() {
    vec3 N = normalize(wNormal);
    vec3 V = normalize(wView);
    vec3 L = normalize(wLight);
    vec3 H = normalize(L + V);
    float cost = max(dot(N,L), 0), cosd = max(dot(N,H), 0);
    vec3 color = ka * La +
                 (kd * cost + ks * pow(cosd,shine)) * Le;
    fragmentColor = vec4(color, 1);
}
```

# NPR: Non-Photorealistic Rendering

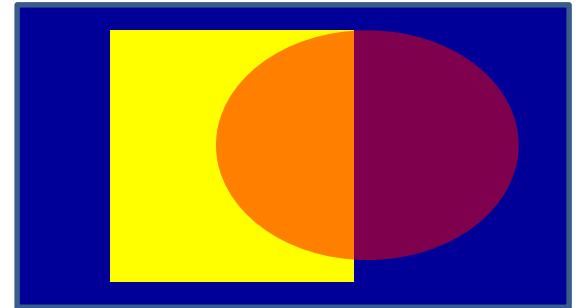
```
uniform vec3 kd;           // diffuse ref
in  vec3 wNormal, wView, wLight; // interpolated
out vec4 fragmentColor;     // output goes to frame buffer

void main() {
    vec3 N = normalize(wNormal);
    vec3 V = normalize(wView);
    vec3 L = normalize(wLight);
    float y = (dot(N, L) > 0.5) ? 1 : 0.5;
    if (abs(dot(N, V)) < 0.2) fragmentColor = vec4(0, 0, 0, 1);
    else                         fragmentColor = vec4(y * kd, 1);
}
```

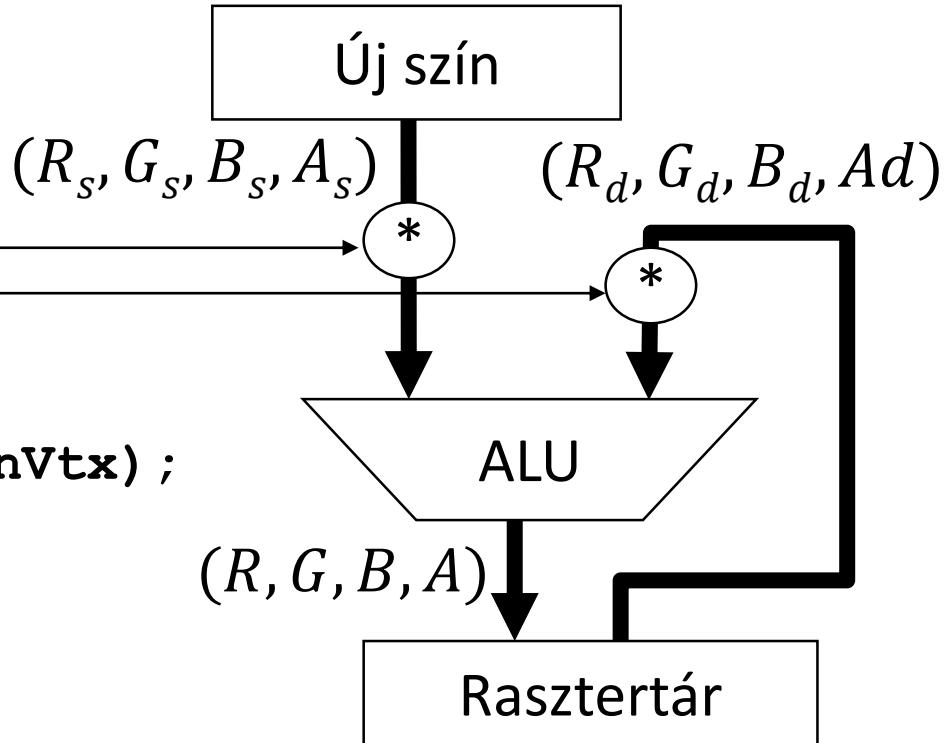


# Kompozitálás és átlátszóság

## Sorrend számít!



```
glEnable(GL_BLEND);  
  
glBlendFunc(  
    GL_SRC_ALPHA, _____  
    GL_ONE_MINUS_SRC_ALPHA _____  
);  
  
glDrawArrays(GL_TRIANGLES, 0, nVtx);  
  
glDisable(GL_BLEND);
```



# Inkrementális képszintézis csővezeték

- Azért transzformáltunk, hogy a láthatósági feladatot és a vetítést képernyő koordinátarendszerben oldassuk meg
  - Triviális hátsó lap eldobás
  - Z-buffer algoritmus
  - Vetítés = z eldobása, 3D háromszög = 2D háromszög
- Per-pixel árnyalás:
  - Vektorokat csúcspontonként számítjuk és pixelekre interpoláljuk
  - Illuminációs képlet pixelenként

### 3. házi: Luxo Grandpa

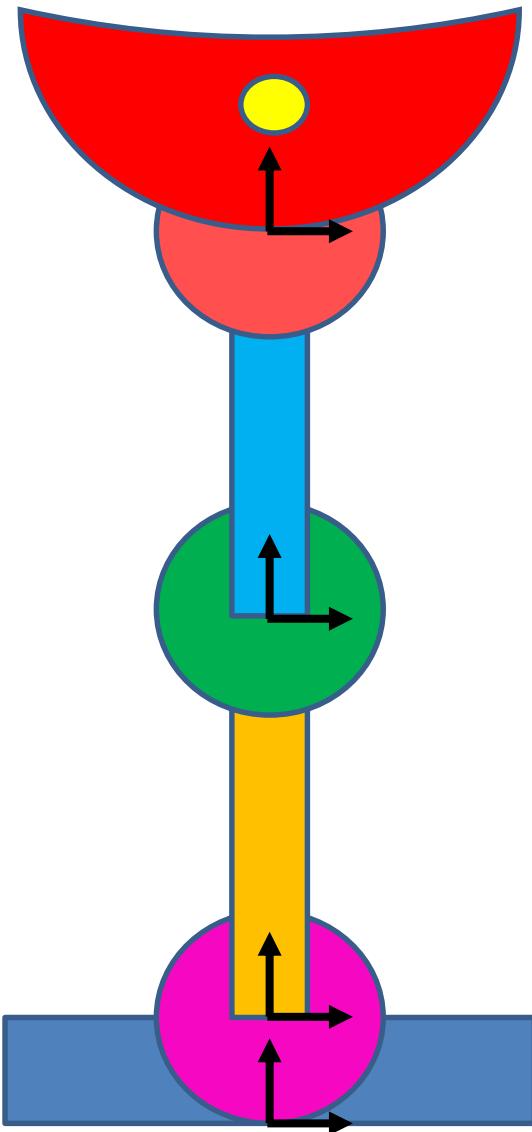
Luxo Junior-ból *Luxo Grandpa (LG)* lett, ezért új **inkrementális képszintézis programot** szentelünk neki.

LG egy síkon áll, a szerkezete (alulról felfelé): henger talp, gömbcsukló1, henger rúd1, gömbcsukló2, henger rúd2, gömbcsukló3, paraboloid, amelynek fókuszpontjában egy pontfényforrás ül.

A gömbcsuklók a koordinátatengelyektől eltérő tengelyek mentén folyamatosan elfordulnak. A szereplőket még egy pontfényforrás és ambiens fény világítja meg. A kamera forog LG körül. **Árnyékot csak a paraboloid vet a fókuszpontjában lévő pontforrásra.**

A szereplők diffúz-spekuláris típusú rücskös anyagúak.

# Luxo Grandpa



## Feladatok:

Téglalap, paraboloid, henger, gömb

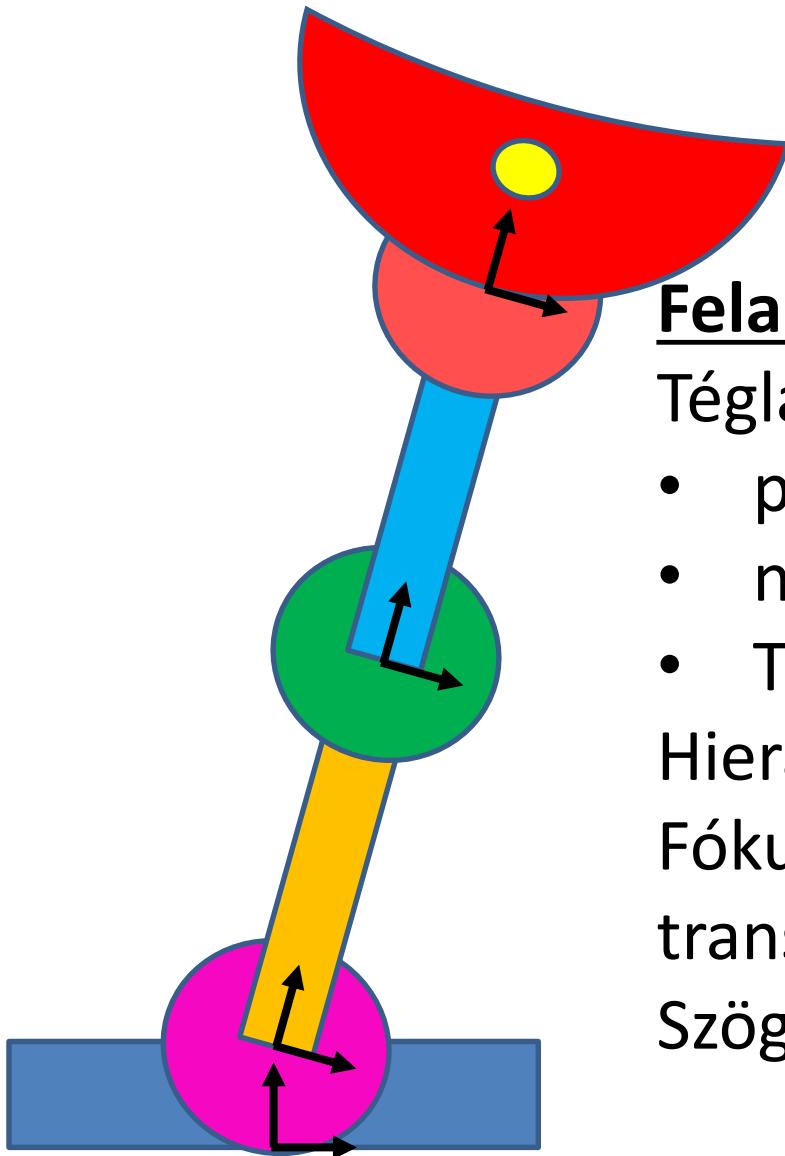
- parametrikus egyenlete
- normálvektora
- Tesszellációja

Hierarchikus transzformálás.

Fókuszpont és tengely hierarchikus  
transzformálása

Szög vizsgálat a lámpa hatásánál

# Luxo Grandpa



## Feladatok:

Téglalap, paraboloid, henger, gömb

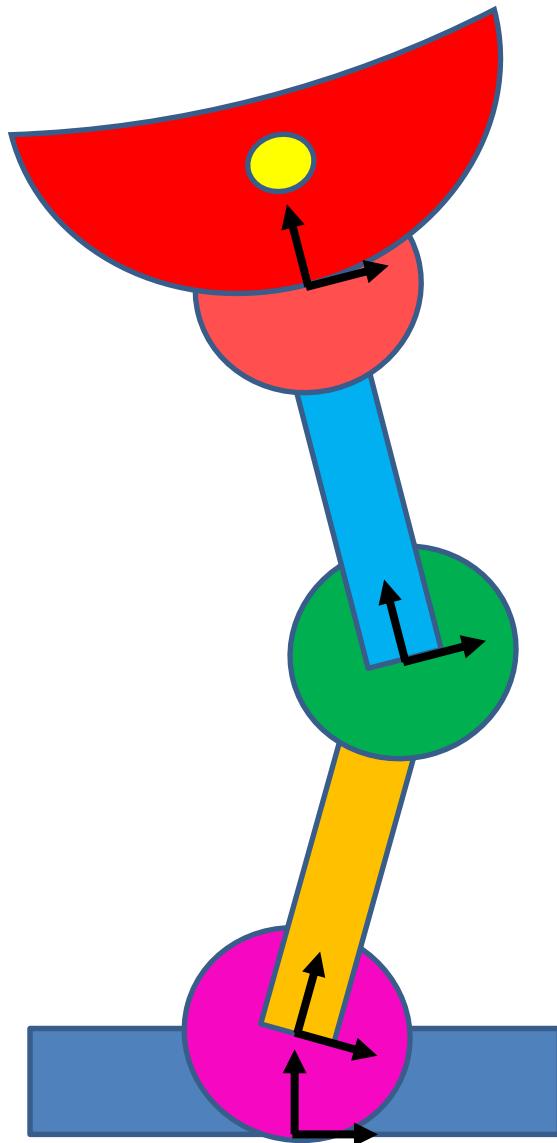
- parametrikus egyenlete
- normálvektora
- Tesszellációja

Hierarchikus transzformálás.

Fókuszpont és tengely hierarchikus  
transzformálása

Szög vizsgálat a lámpa hatásánál

# Luxo Grandpa



## Feladatok:

Téglalap, paraboloid, henger, gömb

- parametrikus egyenlete
- normálvektora
- Tesszellációja

Hierarchikus transzformálás.

Fókuszpont és tengely hierarchikus  
transzformálása

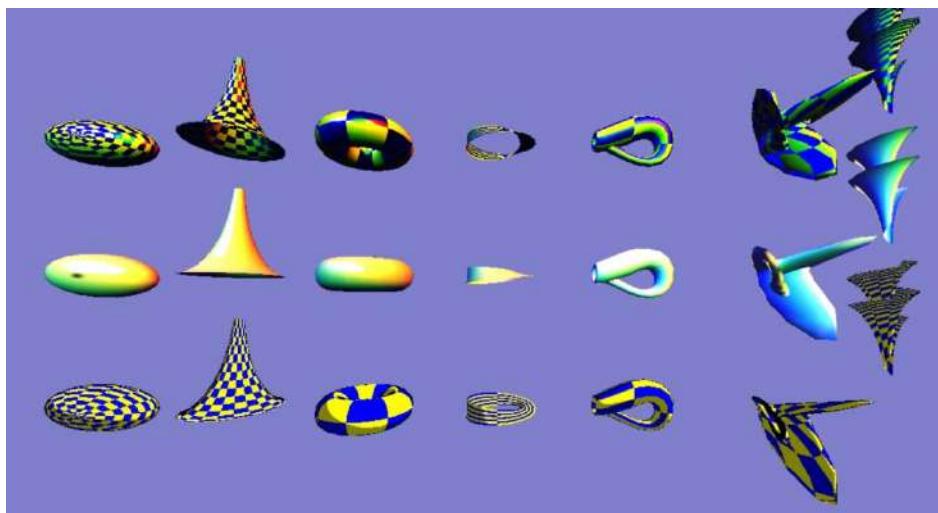
Szög vizsgálat a lámpa hatásánál

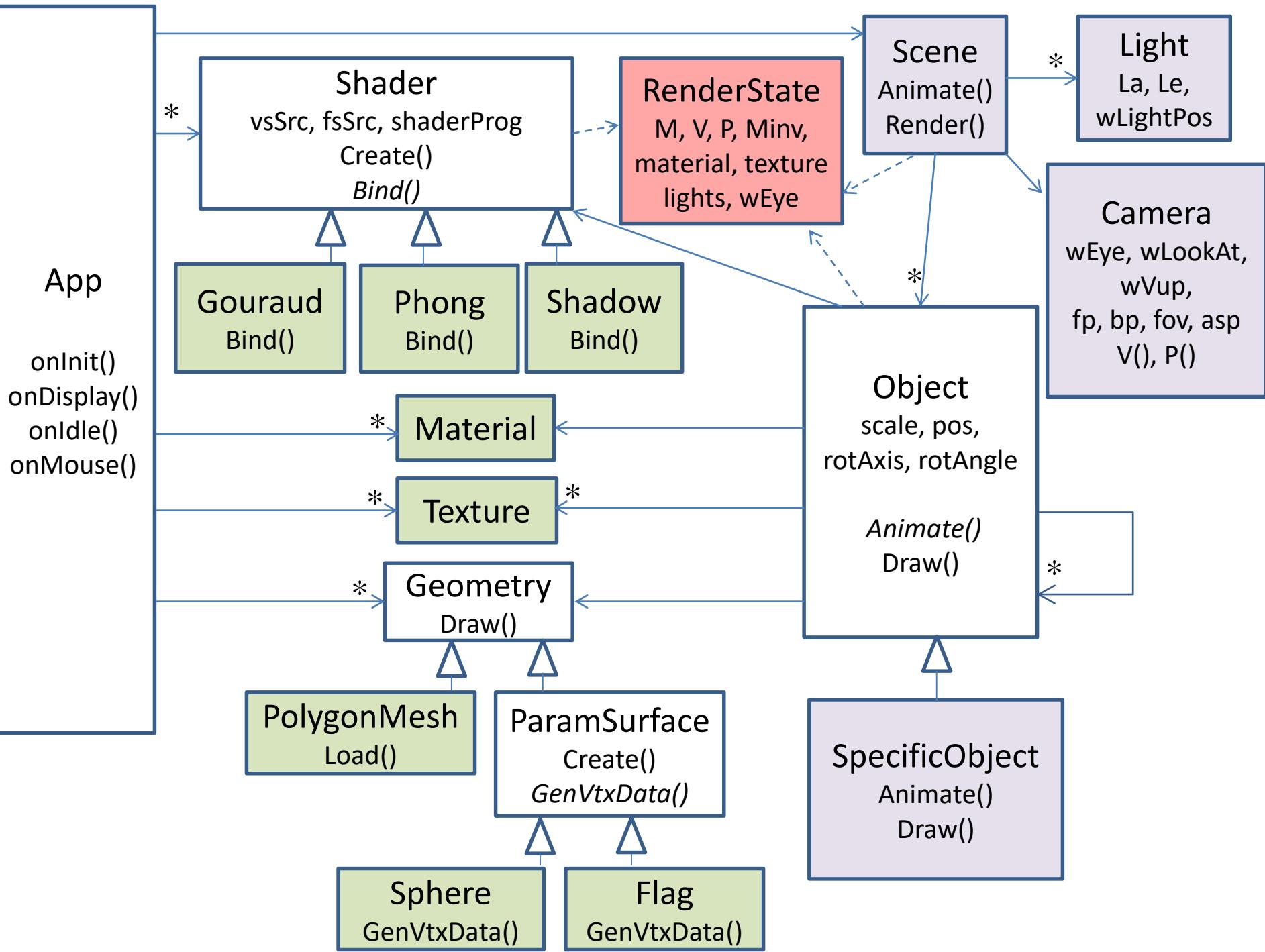


# Inkrementális 3D képszintézis

## 5. Program: 3D motorka

Szirmay-Kalos László





# Scene

```
class Scene {
    Camera camera;
    vector<Object *> objects;
    vector<Light *> lights;
public:
    void Render() {
        RenderState state;
        state.M = state.Minv = UnitMatrix();
        state.wEye = camera.wEye;
        state.V = camera.V();
        state.P = camera.P();
        state.lights = lights;
        for (Object * obj : objects) obj->Draw(state);
    }
    void Animate(float ts, float te) {
        for (Object * obj : objects) obj->Animate(ts, te);
    }
};
```

# Object

```
class Object {  
    Shader * shader;  
    Material * material;  
    Texture * texture;  
    Geometry * geometry;  
    vec3 scale, pos, rotAxis;  
    float rotAngle;  
    vector<Object *> children;  
  
public:  
    virtual void Draw(RenderState state) {  
        state.M = Scale(scale.x, scale.y, scale.z) *  
            Rotate(rotAngle, rotAxis.x, rotAxis.y, rotAxis.z) *  
            Translate(pos.x, pos.y, pos.z) * state.M;  
        state.Minv = state.Minv *  
            Translate(-pos.x, -pos.y, -pos.z) *  
            Rotate(-rotAngle, rotAxis.x, rotAxis.y, rotAxis.z) *  
            Scale(1/scale.x, 1/scale.y, 1/scale.z);  
        state.material = material; state.texture = texture;  
        shader->Bind(state); // uniform változók beállítása  
        geometry->Draw(); // háromszögek végigmennek a pipeline-on  
        for (Object * child : children) child->Draw(state);  
    }  
    virtual void Animate(float ts, float te) {}  
};
```

Érték szerinti paraméterátadás:  
Objektumok nem zavarják egymást

# Shader

```
struct Shader {  
    unsigned int shaderProg;  
  
    void Create(const char * vsSrc,  
                const char * fsSrc, const char * fsOuput) {  
        unsigned int vs = glCreateShader(GL_VERTEX_SHADER);  
        glShaderSource(vs, 1, &vsSrc, NULL); glCompileShader(vs);  
        unsigned int fs = glCreateShader(GL_FRAGMENT_SHADER);  
        glShaderSource(fs, 1, &fsSrc, NULL); glCompileShader(fs);  
        shaderProg = glCreateProgram();  
        glAttachShader(shaderProg, vs);  
        glAttachShader(shaderProg, fs);  
        glBindFragDataLocation(shaderProg, 0, fsOuput);  
        glLinkProgram(shaderProg);  
    }  
    virtual void Bind(RenderState& state) {  
        glUseProgram(shaderProg);  
    }  
};
```



# ShadowShader

```
class ShadowShader : public Shader {  
    const char * vsSrc = R"(  
        uniform mat4 MVP;  
        layout(location = 0) in vec3 vtxPos;  
        void main() { gl_Position = vec4(vtxPos, 1) * MVP; }  
    )";  
  
    const char * fsSrc = R"(  
        out vec4 fragmentColor;  
        void main() { fragmentColor = vec4(0, 0, 0, 1); }  
    )";  
  
public:  
    ShadowShader() {  
        Create(vsSrc, fsSrc, "fragmentColor");  
    }  
  
    void Bind(RenderState& state) {  
        glUseProgram(shaderProg);  
        mat4 MVP = state.M * state.V * state.P;  
        MVP.SetUniform(shaderProg, "MVP");  
    }  
};
```

*“The only legitimate use of a computer is to play games.”*

*Eugene Jarvis*

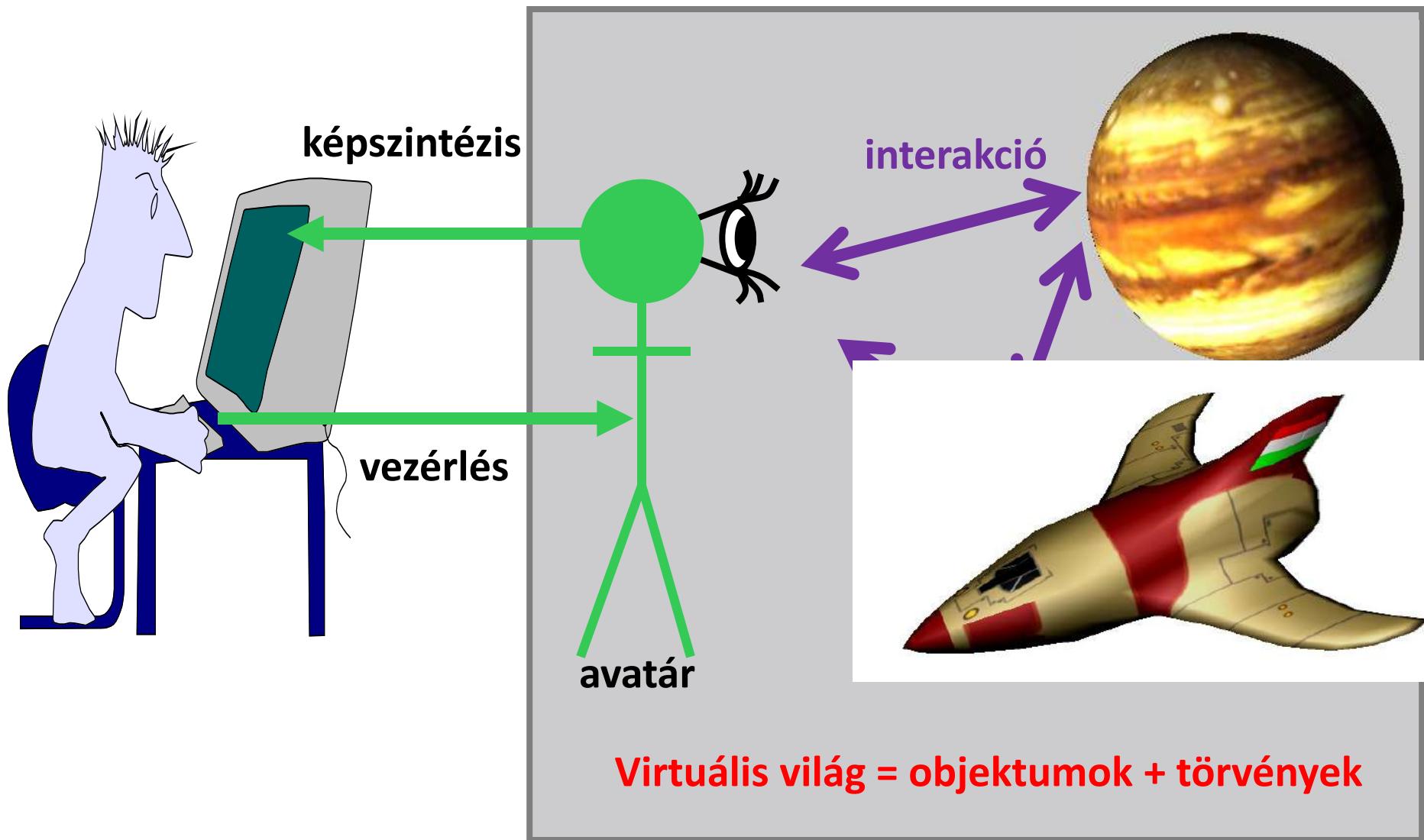
# Játékfejlesztés

Szirmay-Kalos  
László





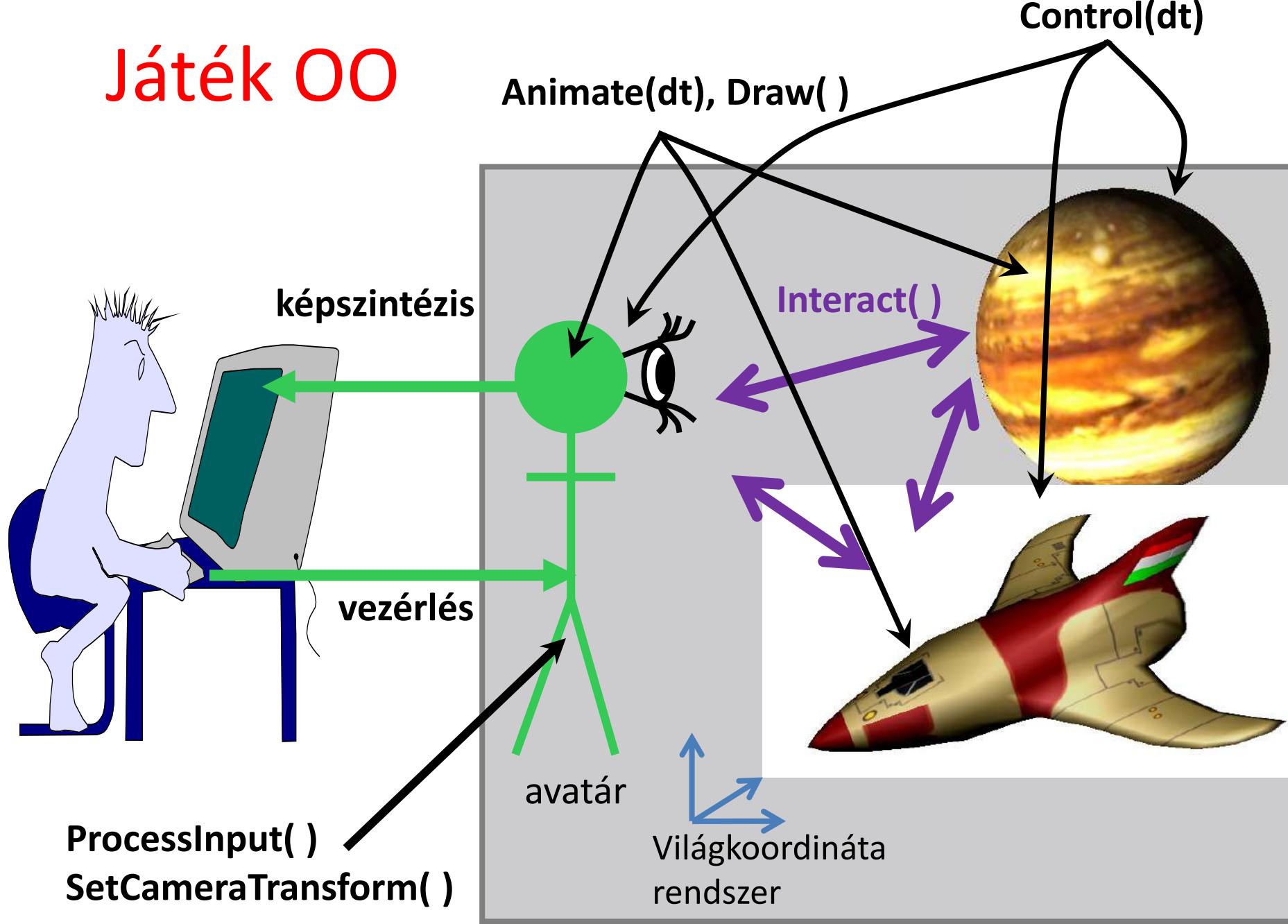
# Virtuális valóság



# Játékok feladatai

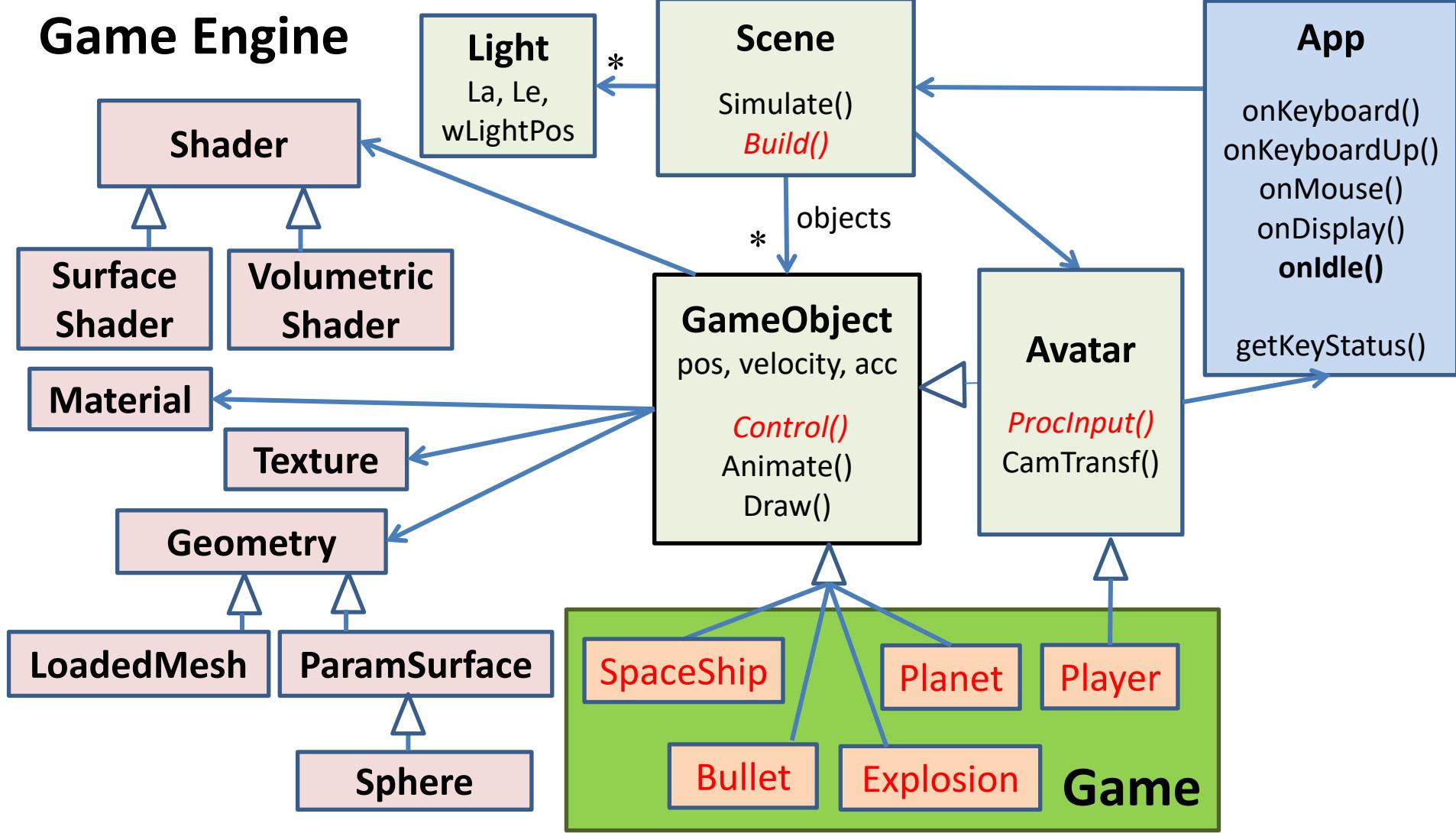
- Képszintézis az avatár nézőpontjából
- Az avatár vezérlése a beviteli eszközökkel (keyboard, mouse, Wii, gépi látás, Kinect, stb.)
- Az „intelligens” objektumok vezérlése (AI)
- A fizikai világ szimulációja

# Játék OO



# Osztálydiagram

## Game Engine



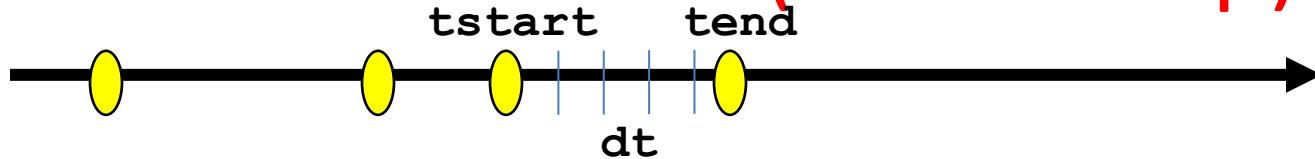
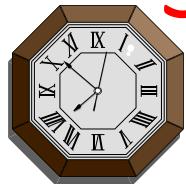
# Játékobjektum (GameObject)

```
class GameObject {  
protected:  
    Shader * shader;  
    Material * material;  
    Texture * texture;  
    Geometry * geometry;  
    vec3 pos, velocity, acceleration;  
public:  
    GameObject(Shader* s, Material* m,  
               Texture* t, Geometry* g) { ... }  
    virtual void Control(float dt) { }  
    virtual void Animate(float dt) { }  
    virtual void Draw(RenderState state) { }  
};
```

## Virtuális világ

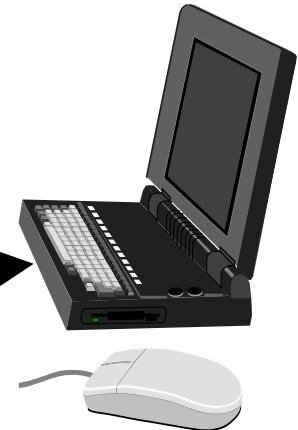
```
vector<GameObject *> objects;
```

# Szimulációs hurok (Game loop)



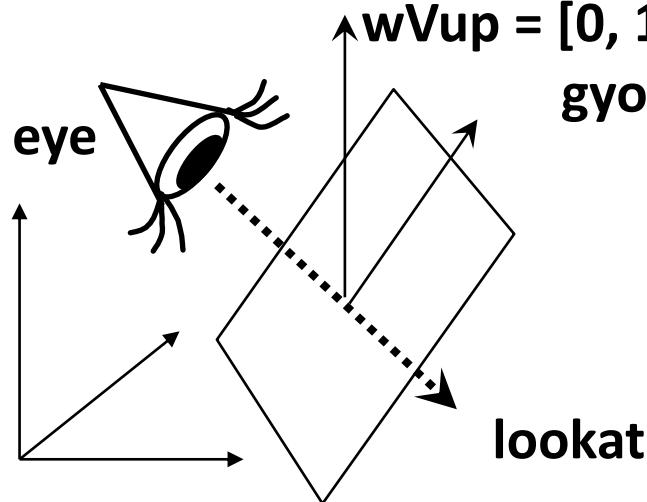
```
void onIdle ( ) {    // idle call back
    static float tend = 0;
    float tstart = tend;
    tend = glutGet(GLUT_ELAPSED_TIME)/1000.0f;
    avatar->ProcessInput( );
    for(float t = tstart; t < tend; t += dt) {
        float Dt = min(dt, tend - t);
        for (GameObject * obj : objects) obj->Control(Dt);
        for (GameObject * obj : objects) obj->Animate(Dt);
    }
    glutPostRedisplay();
}

void onDisplay(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    avatar->SetCameraTransform(state);
    for (GameObject * obj : objects) obj->Draw(state);
    glutSwapBuffers( );
}
```



# Avatar

```
struct Avatar : public GameObject {  
    virtual void ProcessInput() { }  
    virtual vec3 wVup() { return vec3(0, 1, 0); }  
    void SetCameraTransform(RenderState& state) {  
        Camera camera(pos, pos + velocity, wVup);  
        state.V() = camera.V();  
        state.P() = camera.P();  
    }  
};
```



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

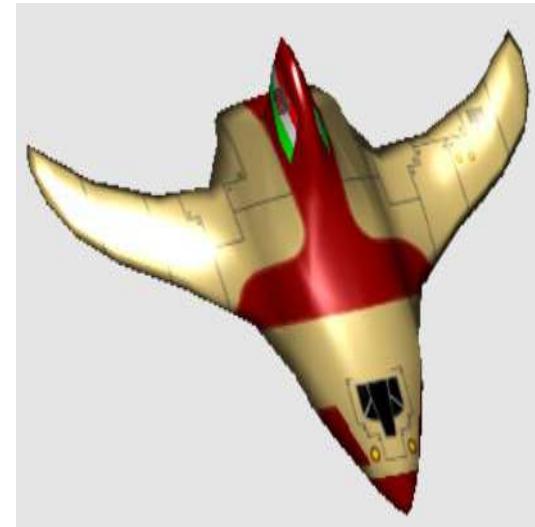
*"Be van fejezve a nagy mű, igen.  
A gép forog, az alkotó pihen.  
Évmilliókig eljár tengelyén,  
Míg egy kerékfogát ujítni kell."*

*Madách Imre*

# Játékfejlesztés

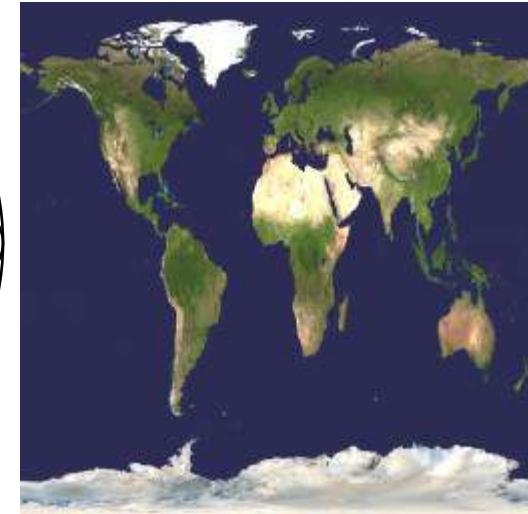
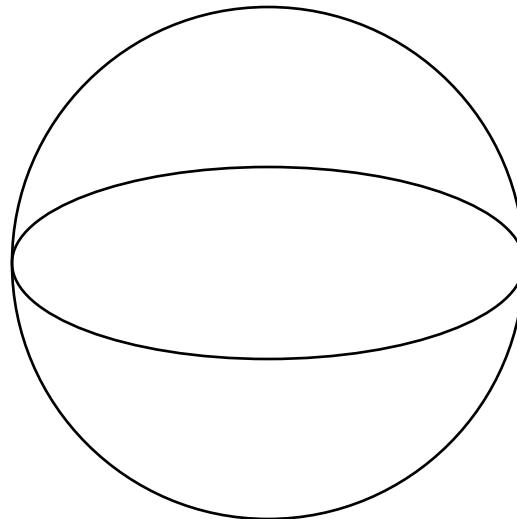
## 2. Játékobjektumok

Szirmay-Kalos László



# Bolygó: Planet

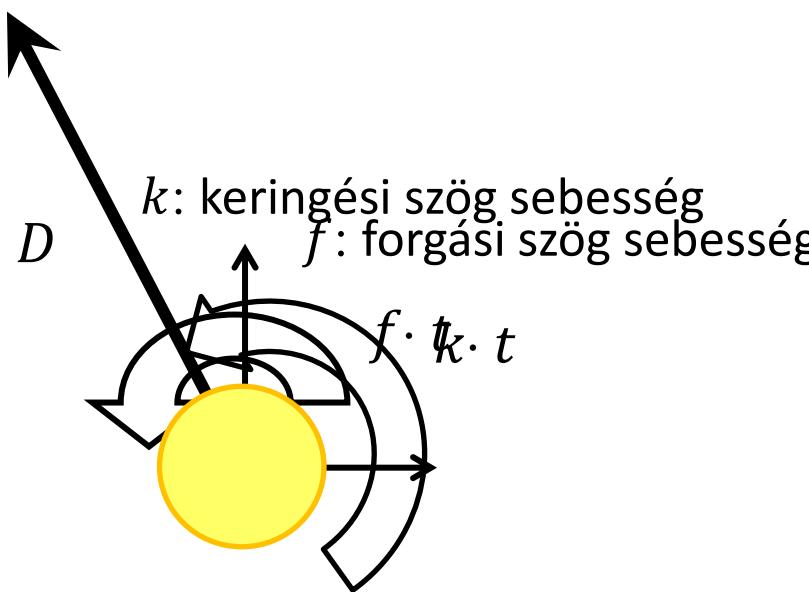
- Geometria: gömb
- Textúra
- Fizikai vagy
  - Tájékozódik majd követi a gravitációs törvényt
- Képletanimáció:
  - „beégetett pálya”
  - Többiek érdektelenek
  - Nincs respektált törvény





Animate:

## A Föld forog és kering a Nap körül



$$M = \begin{bmatrix} \cos(f \cdot t) & \sin(f \cdot t) & 0 & 0 \\ -\sin(f \cdot t) & \cos(f \cdot t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(23^\circ) & \sin(23^\circ) & 0 \\ 0 & -\sin(23^\circ) & \cos(23^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D \cos(k \cdot t) & D \sin(k \cdot t) & 0 & 1 \end{bmatrix}$$

# Planet

```
const vec3 X(1,0,0), Z(0,0,1);

class Planet : public GameObject {
    float rotAng = 0, revAng = 0; // animation state
    float rotVel, revVel, tilt, D; // animation parameter
public:
    Planet(Shader* s, Material* m, Texture* t, Sphere* s,
           float rot, float rev, float ti, float d) :
        GameObject(s,m,t,s), rotVel(rot), revVel(rev), tilt(ti), D(d) {}

    void Animate(float dt) {
        rotAng += rotVel * dt; revAng += revVel * dt;
    }

    void Draw(RenderState state) {
        vec3 p = vec3(cos(revAng), sin(revAng), 0) * D;
        state.M = RotationMatrix(rotAng, Z) * RotationMatrix(tilt, X) *
                  TranslateMatrix(p);
        state.Minv = TranslateMatrix(-p) * RotationMatrix(-tilt, X) *
                     RotationMatrix(-rotAng, Z);
        state.material = material; state.texture = texture;
        shader->Bind(state);
        geometry->Draw();
    }
};
```

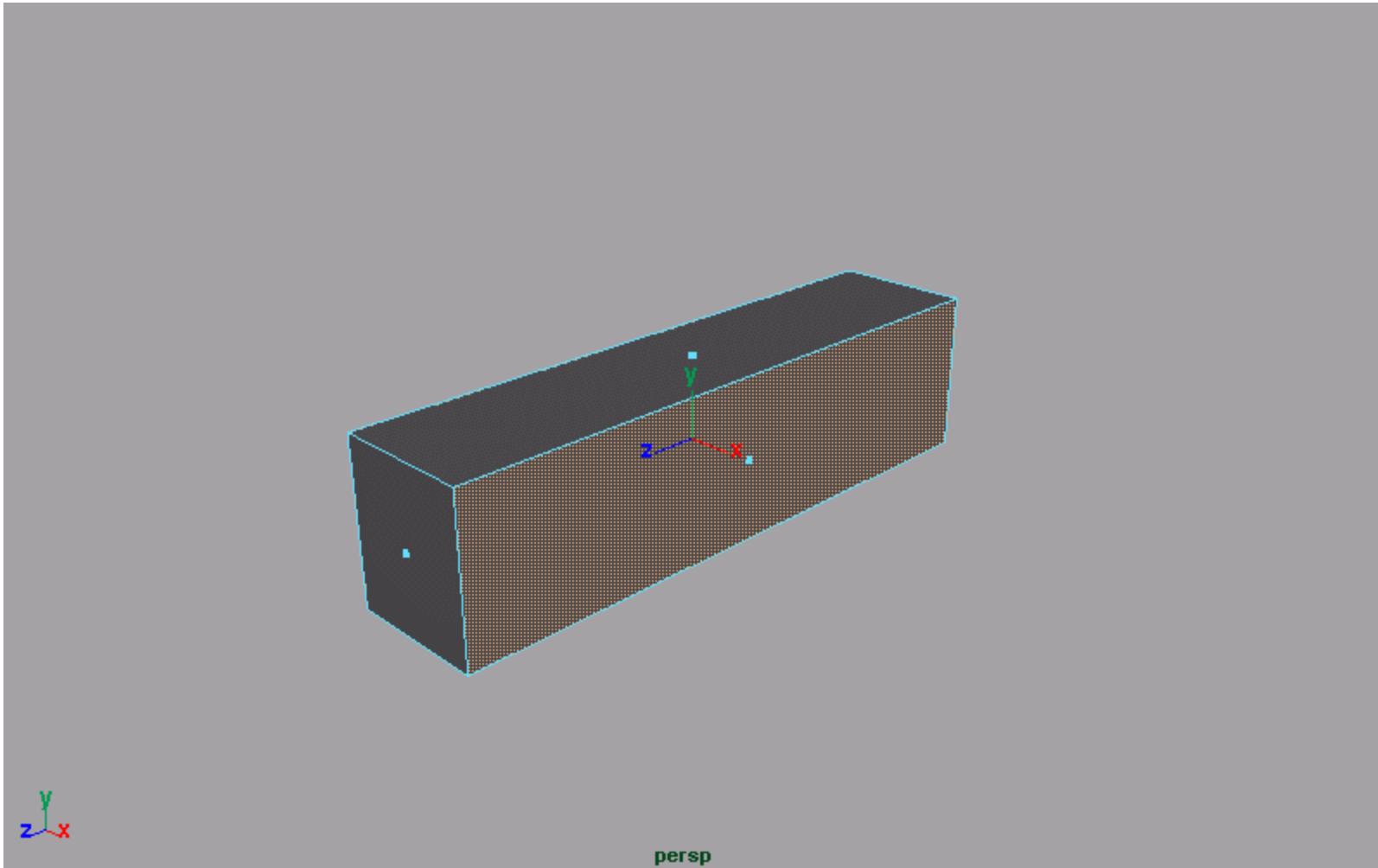
# Az Úrhajó

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

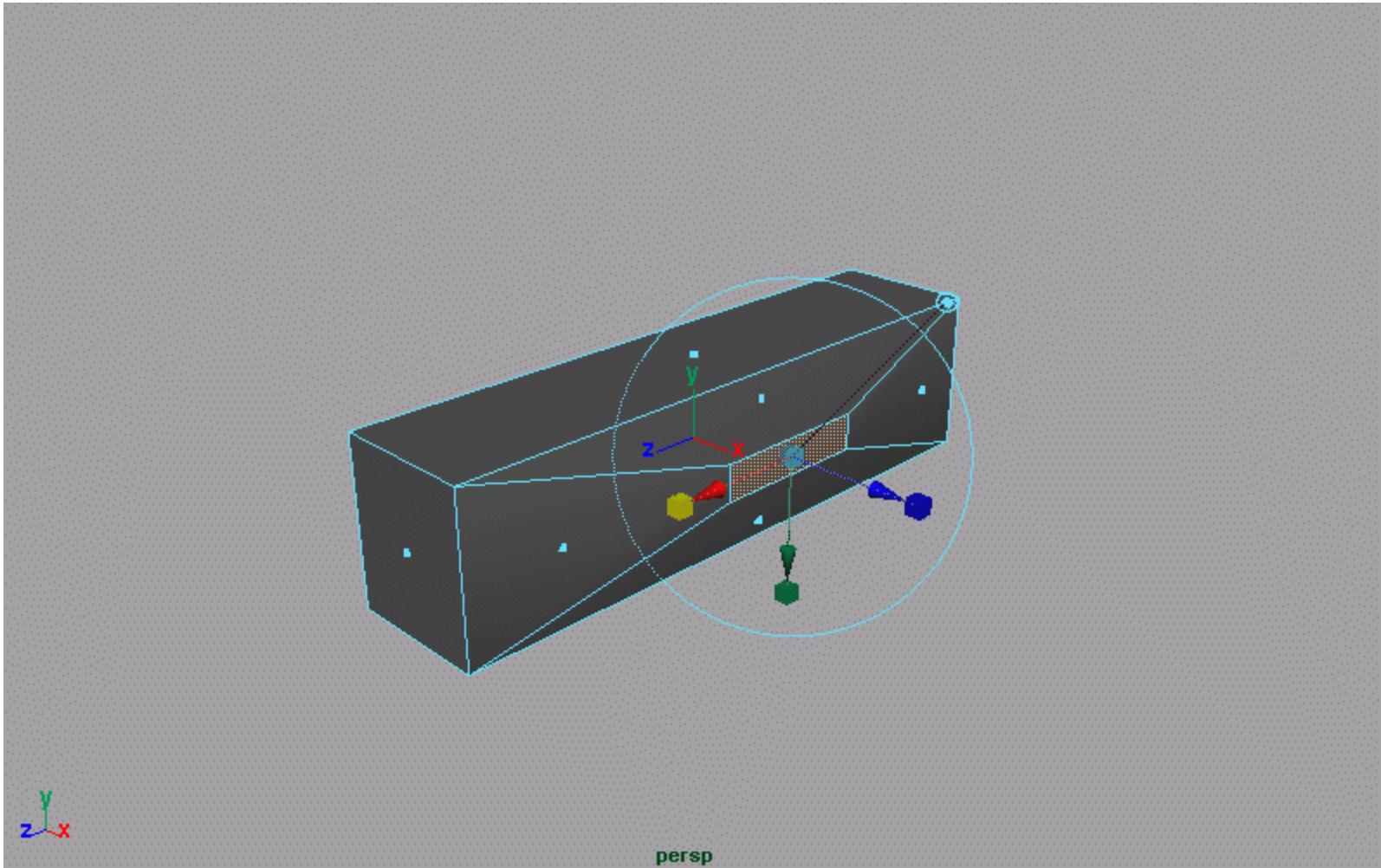


# Úrhajó geometria

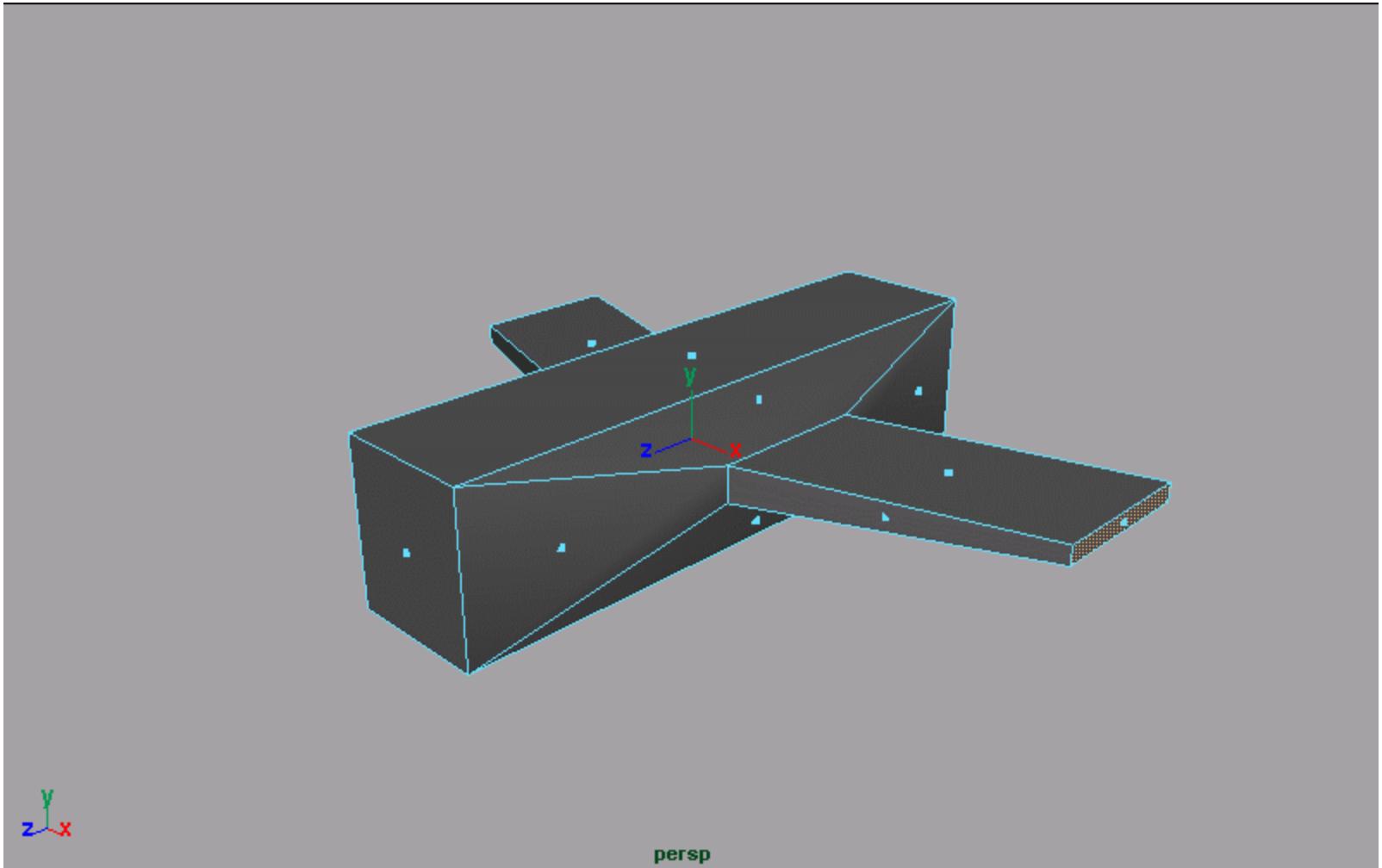
Euler műveletek: csúcs + lap = él + 2



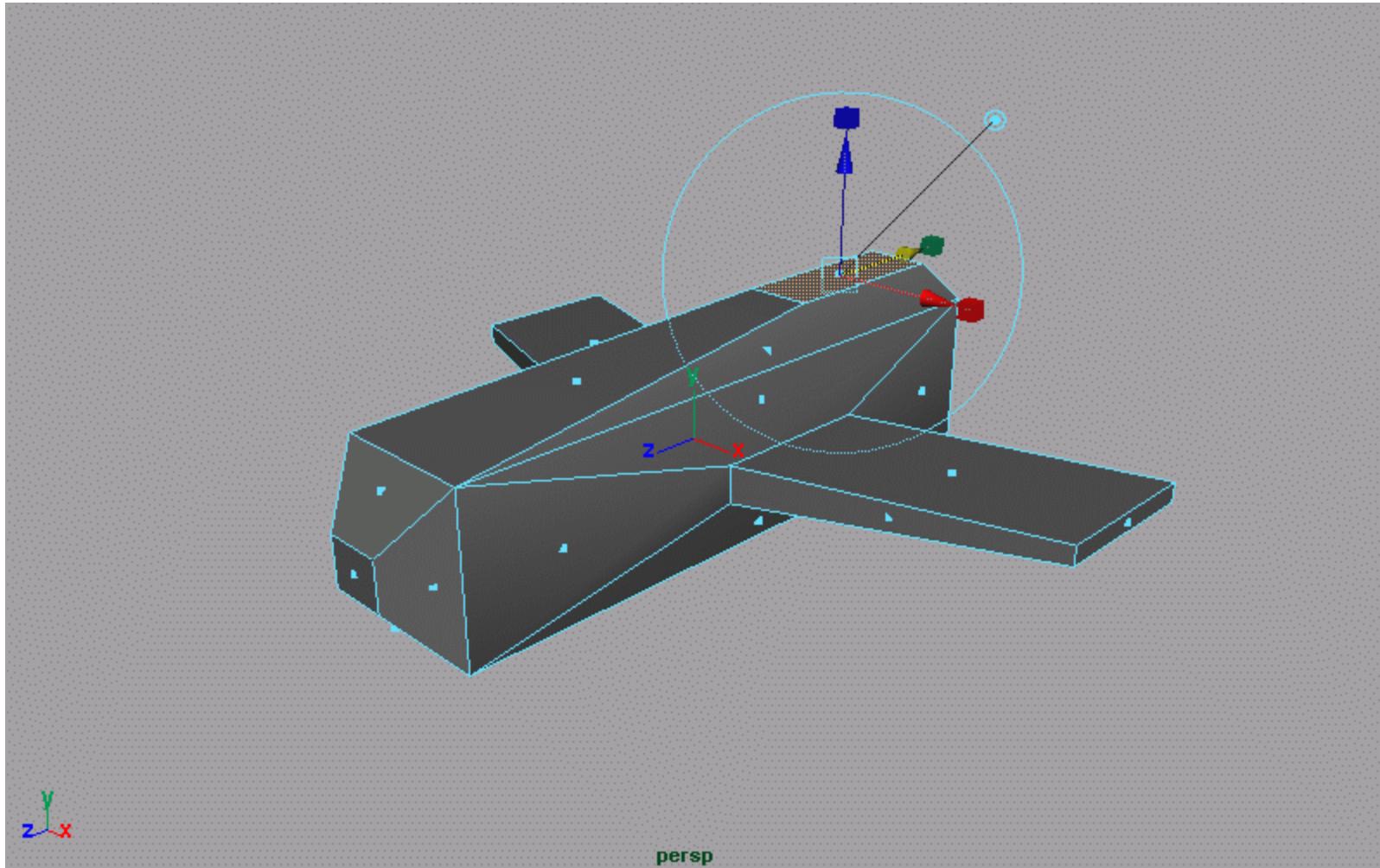
# Úrhajó geometria



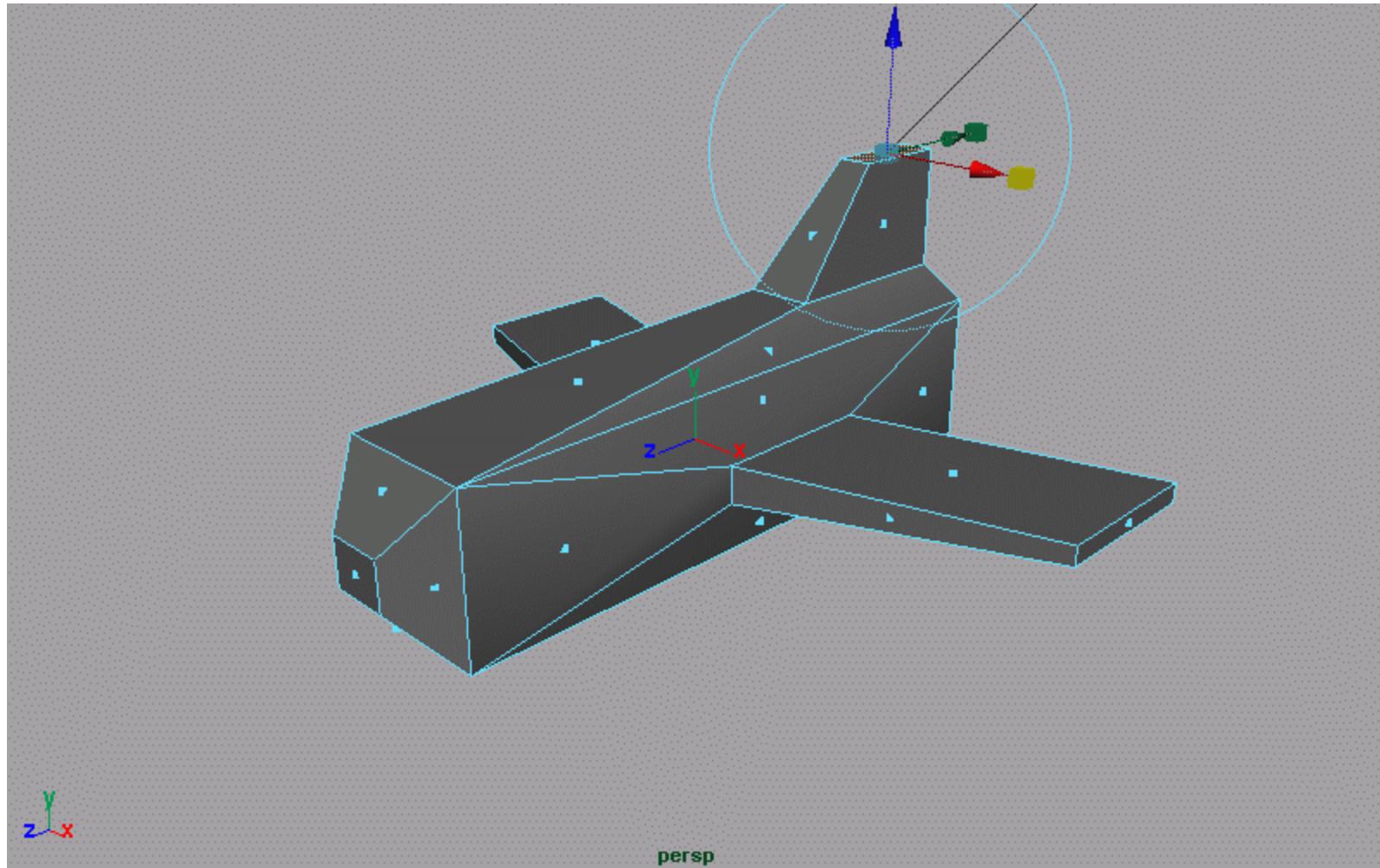
# Úrhajó geometria



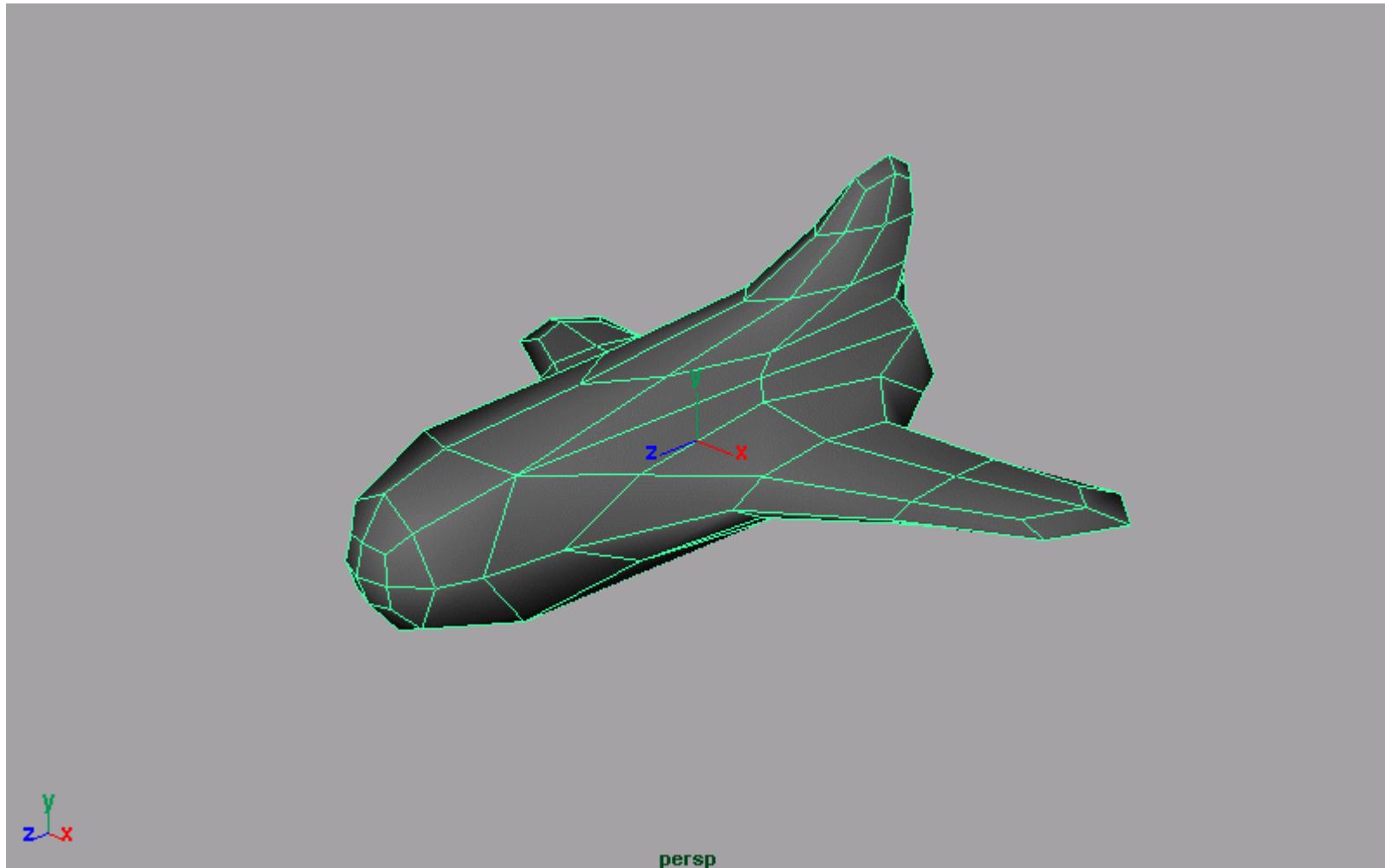
# Úrhajó geometria



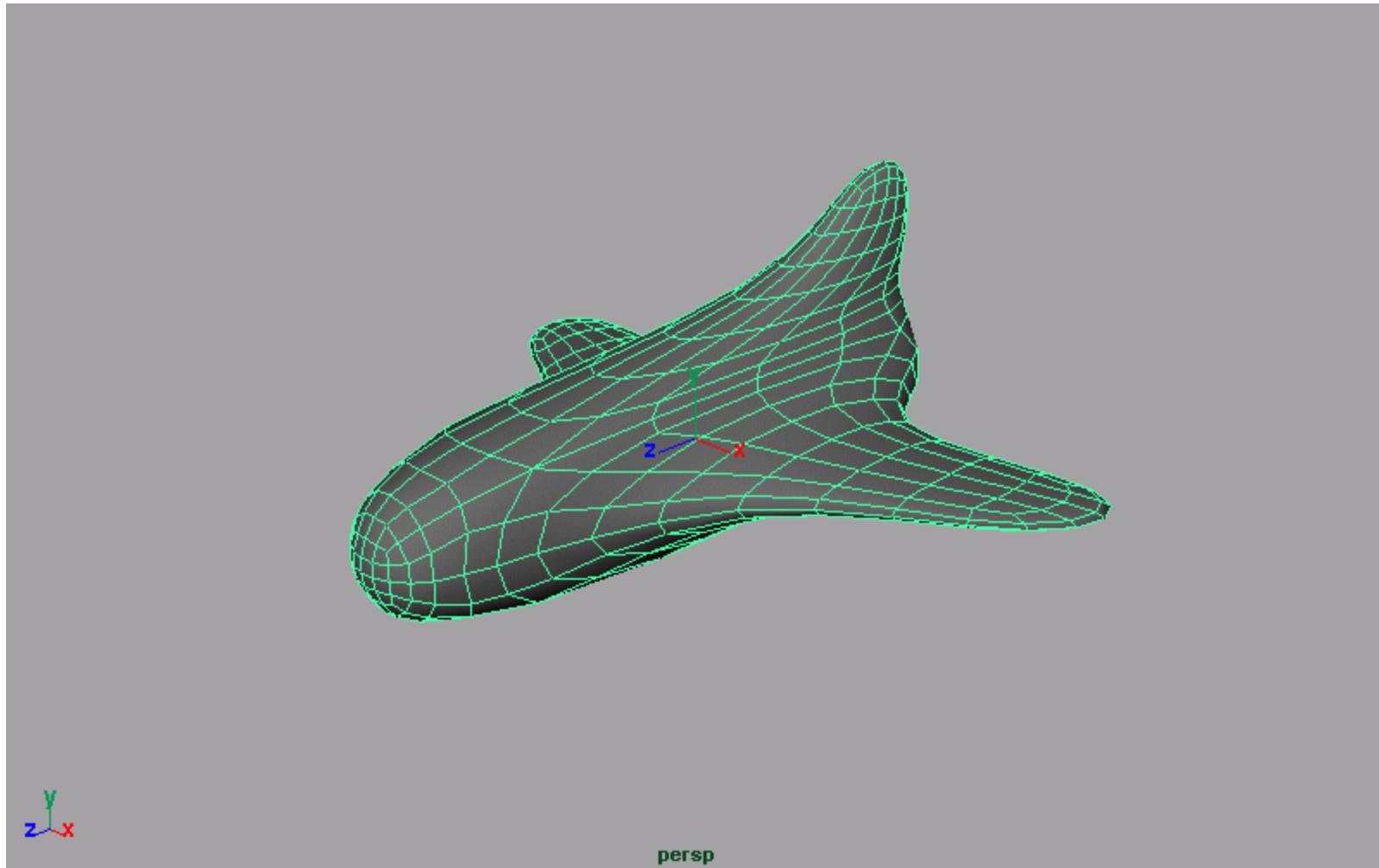
# Úrhajó geometria



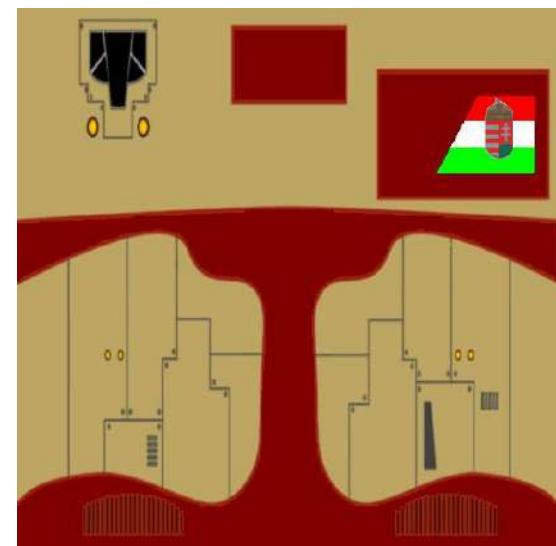
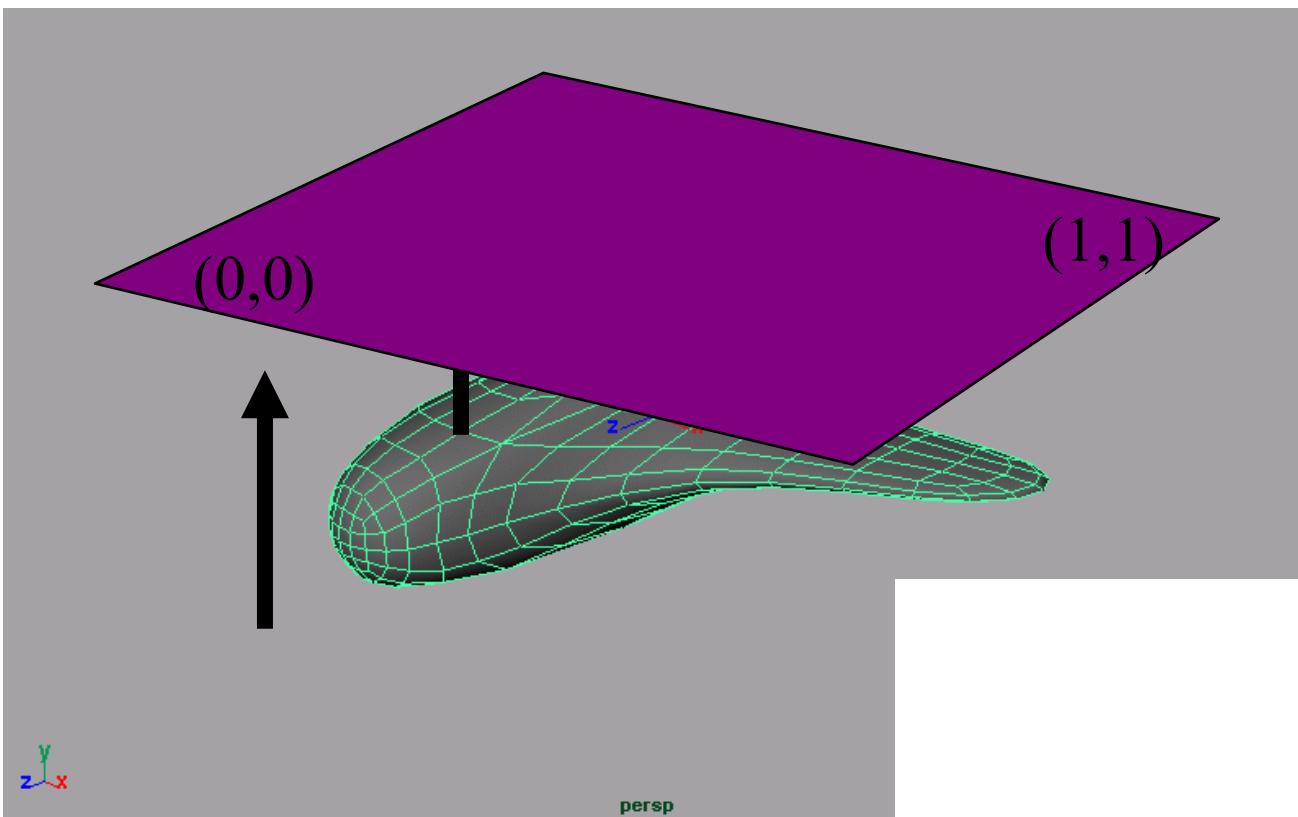
# Úrhajó geometria: Catmull-Clark subdivision



# Úrhajó geometria: Catmull-Clark subdivision



# Textúrához paraméterezés



# OBJ fájlformátum

```
v -0.708698 -0.679666 2.277417
v 0.708698 -0.679666 2.277417
v -0.735419 0.754681 2.256846

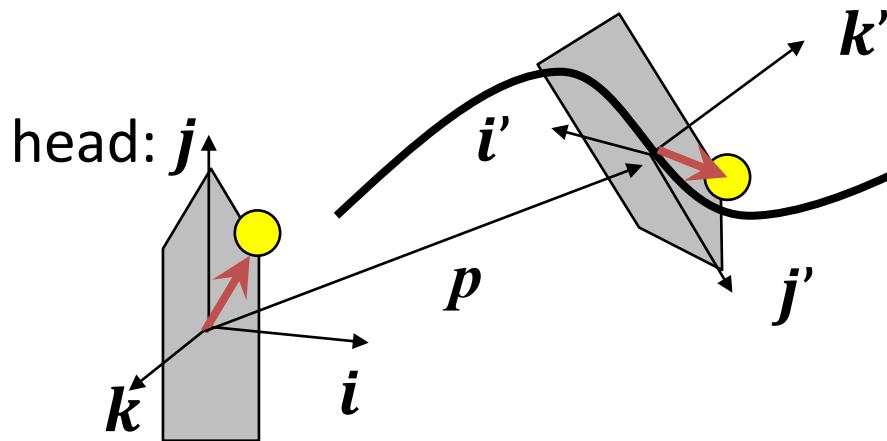
...
vt 0.510655 0.078673
vt 0.509594 0.070000
vt 0.496429 0.079059

...
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: Frenet frame + Newton 2



$$M = \begin{bmatrix} i' & 0 \\ j' & 0 \\ k' & 0 \\ p & 1 \end{bmatrix}$$



Orientáció, nem ortonormált:

$$\begin{aligned} j' &= v, \\ k^* &= k'(1 - \alpha) + a\alpha, \\ i' &= j' \times k^* \end{aligned}$$

Gram-Schmidt ortogonalizáció:

$$\begin{aligned} j' &= v/|v|, \\ i' &= j' \times k^*/|j' \times k^*|, \\ k' &= i' \times j' \end{aligned}$$

Dinamikai szimuláció:

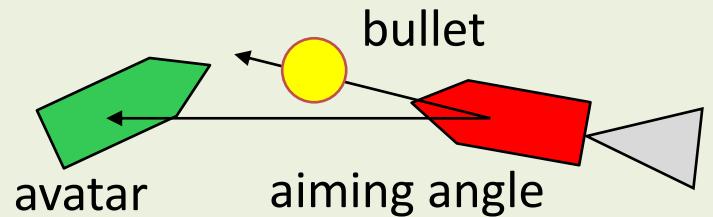
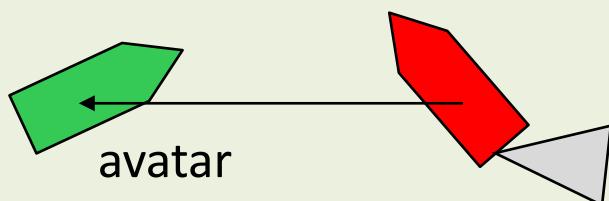
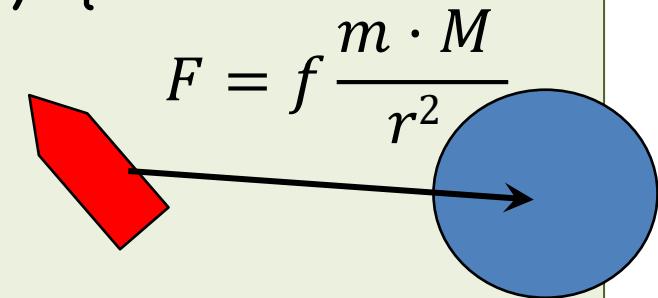
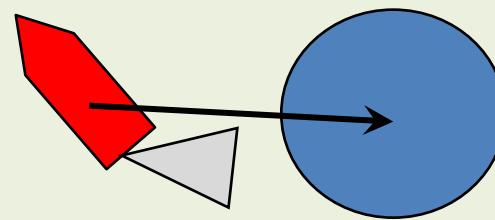
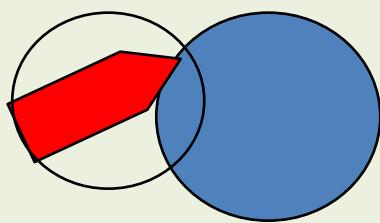
$$F = ma, \quad a = \frac{dv}{dt}, \quad v = \frac{dp}{dt}$$



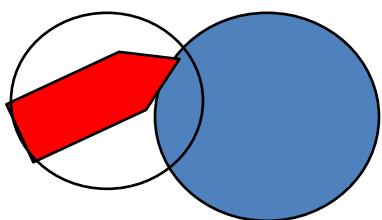
$$a = \frac{\textcolor{red}{F}}{m}, \quad v += a dt, \quad p += v dt$$

# Ship :: Control

```
void Ship :: Control( float dt ) {
    force = vec3(0, 0, 0);
    for (GameObject * obj : objects) {
        if (dynamic_cast<Planet*>(obj)) {
            }
        if (dynamic_cast<Avatar*>(obj)) {
            }
    }
}
```

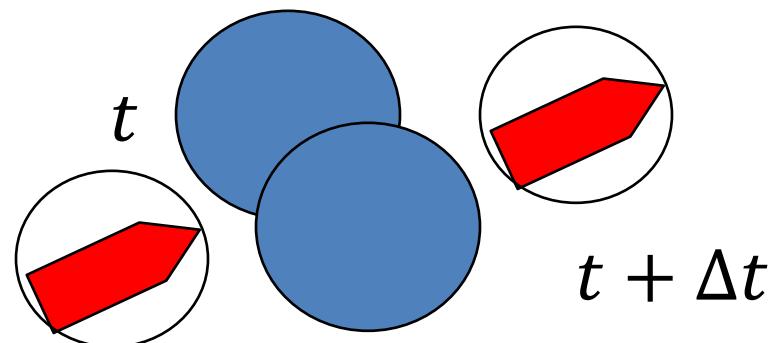


# Ütközésdetektálás: lassú objektumok



adott  $t$

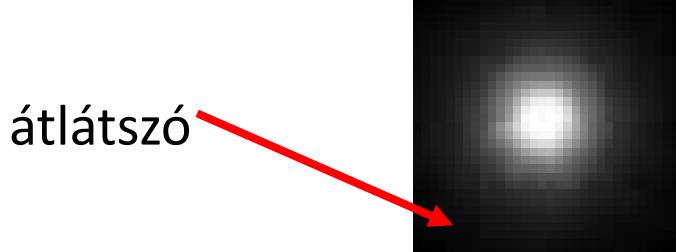
Probléma, ha az objektum gyors



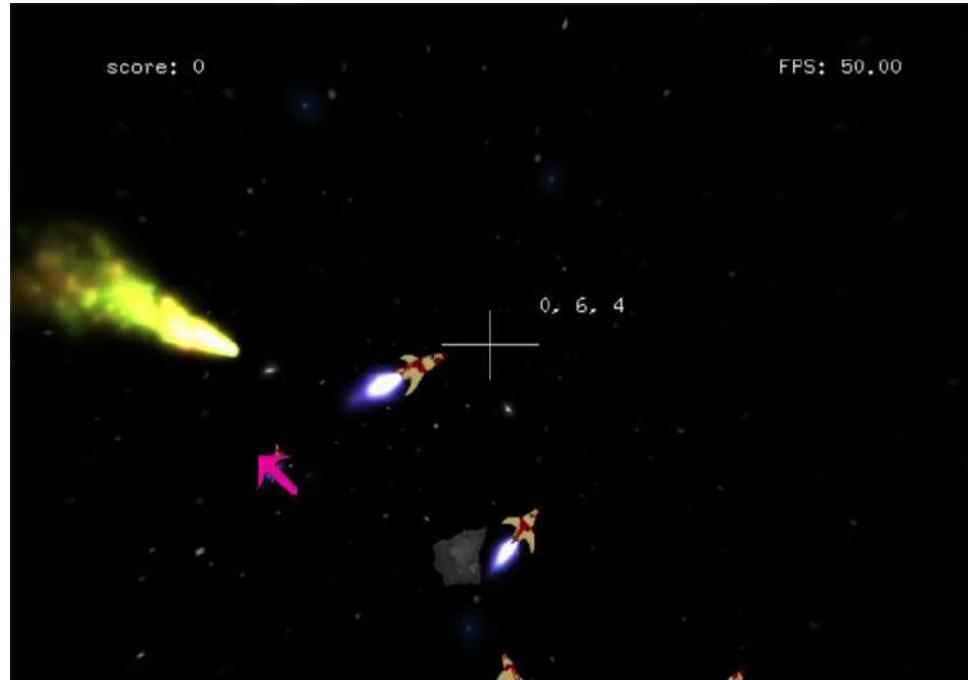
```
dist = length(obj1.pos - obj2.pos)
minDist = obj1.BoundingRadius() + obj2.BoundingRadius()
if (dist < minDist) Collision!
```

# Foton torpedó

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

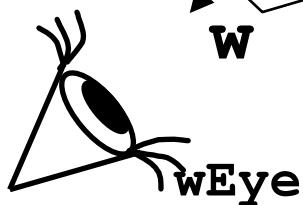
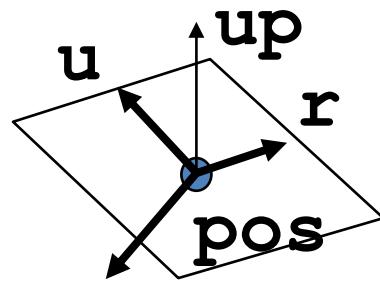
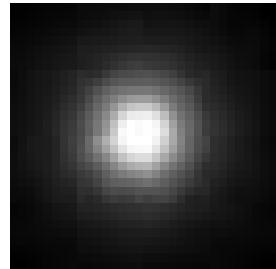
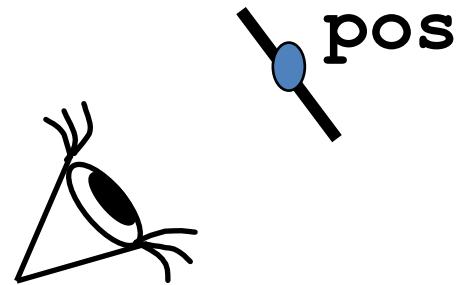
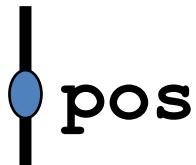
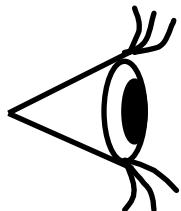
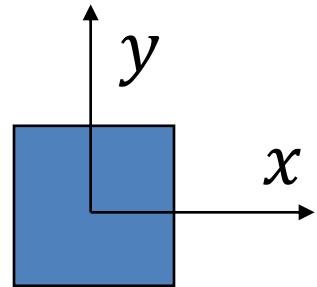


- Ütközésdetektálás = gyors mozgás



# Billboard

Egyetlen fél átlátszó textúra egy téglalapon



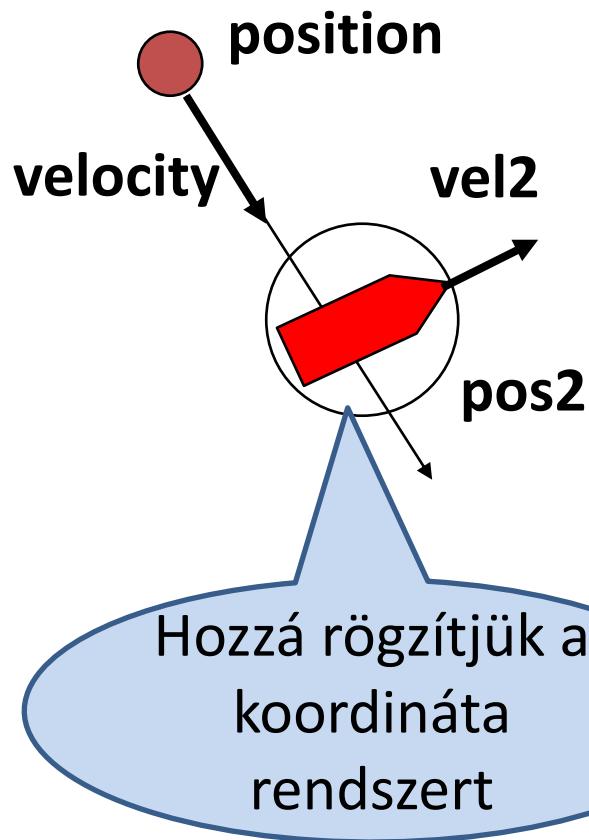
```
vec3 w = wEye - pos;  
vec3 r = cross(w, up);  
vec3 u = cross(r, w);  
r = normalize(r) * size;  
u = normalize(u) * size;
```



# Billboard

```
void Bullet :: Draw(RenderState state) {
    vec3 up = vec3(0, 1, 0);
    vec3 w = state.wEye - pos;
    vec3 r = cross(w, up);
    vec3 u = cross(r, w);
    r = normalize(r) * size;
    u = normalize(u) * size;
    glEnable(GL_BLEND);           // átlátszóság
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    state.M = mat4(r.x,   r.y,   r.z,   0,
                   u.x,   u.y,   u.z,   0,
                   0,     0,     1,     0,
                   pos.x, pos.y, pos.z, 1);
    shader->Bind(state);
    geometry->Draw();
    glDisable(GL_BLEND);
}
```

# Gyors (folytonos) ütközés detektálás



$$\text{rel\_pos} = \text{position} - \text{pos2}$$

$$\text{rel\_velocity} = \text{velocity} - \text{vel2}$$

$$\text{Ray: } \text{rel\_pos} + \text{rel\_velocity} \cdot t$$

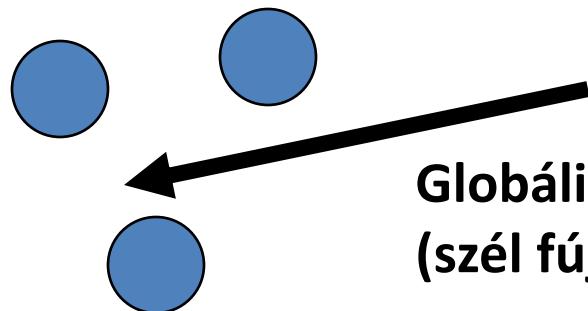
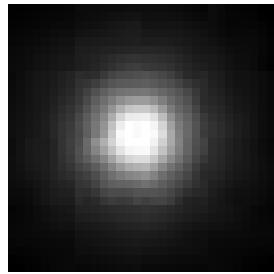
If (ray intersects bounding sphere first  
&&  $t_{\text{intersect}} < dt$ ) ***Collision!***

# Robbanás

- Nagyon komplex geometria
- Hasonló kinézet minden irányból
- Plakátgyűjtemény
- Részecske rendszer

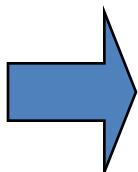


# Részecske rendszerek



Globális erőtér  
(szél fújja a füstöt)

Véletlen  
Kezdeti  
értékek



**pos:**  
**velocity:**  
**acceleration:**  
**lifetime**  
**age:**  
  
**size, dsize:**  
**weight, dweight:**  
**color, dcolor:**

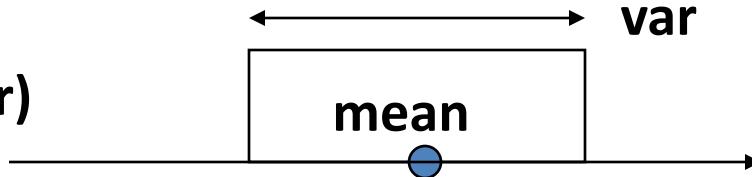
**pos += velocity \* dt**  
**velocity += acceleration \* dt**  
**acceleration = force / weight**  
  
**age += dt; if (age > lifetime) Kill();**  
  
**size += dsize \* dt;**  
**weight += dweight \* dt**  
**color += dcolor \* dt**





# Robbanás paraméterei

Rand(mean, var)



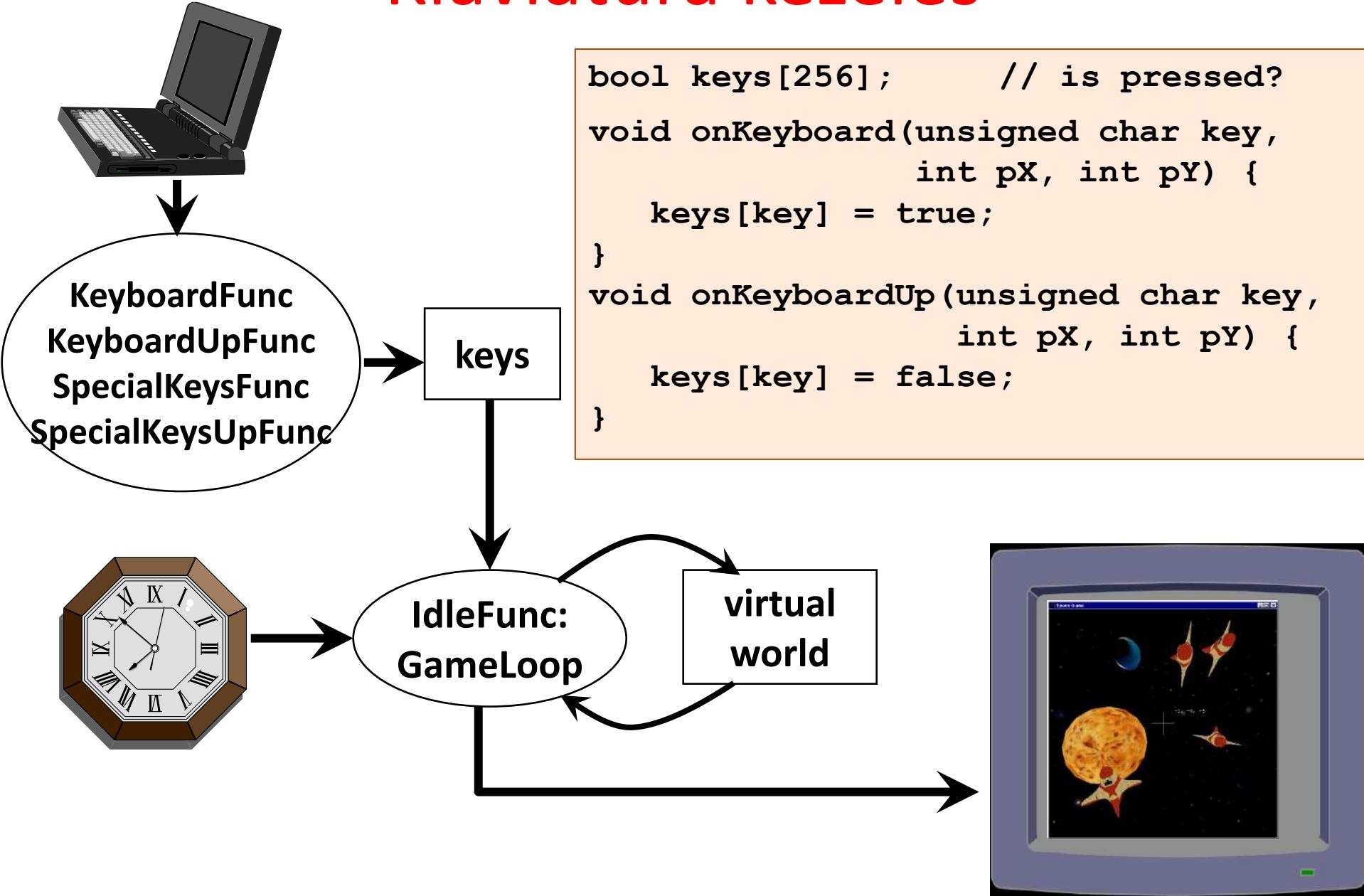
```
pos = center;                                // kezdetben fókuszált
lifetime = Rand(2, 1);

size = 0.001;                                 // kezdetben kicsi
dszie = Rand(0.5, 0.25) / lifetime;

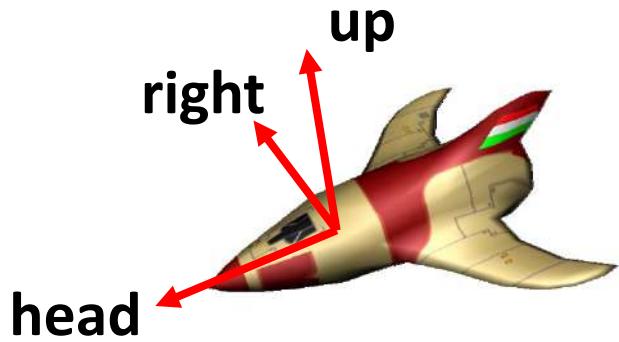
velocity = Vector(Rand(0,0.4), Rand(0,0.4), Rand(0,0.4));
acceleration = Vector(Rand(0,1), Rand(0,1), Rand(0,1));

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

# Klaviatúra kezelés



# Player



```
class Player : public Avatar, public Ship {
    void ProcessInput() {
        if ( app.keys[ ' ' ] ) // Fire!
            objects.push_back( new Bullet( pos, velocity ) );

        // Kormányzás: az avatár koordinátarendszerében!
        vec3 head = normalize( velocity );
        vec3 right = normalize( cross( wVup(), head ) );
        vec3 up = cross( head, right );

        if (app.keys[KEY_UP])      force -= up;
        if (app.keys[KEY_DOWN])   force += up;
        if (app.keys[KEY_LEFT])   force -= right;
        if (app.keys[KEY_RIGHT])  force += right;
    }
};
```

*“Clouds are not spheres, mountains are not cones,  
coastlines are not circles, and bark is not smooth,  
nor does lightning travel in a straight line.”*

*Benoit Mandelbrot*

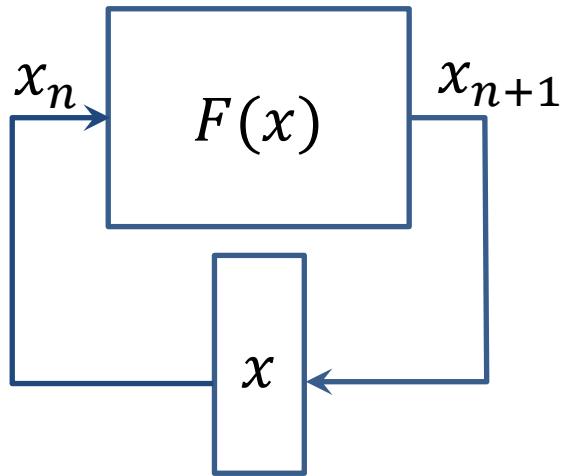
# Fraktálok és káosz

## 1. Fraktális dimenzió

Szirmay-Kalos László



# A valóság (természet) szimulálható?



Ha  $F$  és  $x_0$  csak közelítőleg ismert,  
akkor  $x_n$  is csak közelítés lesz, de ha  
pontosítjuk  $F$ -t és  $x_0$ -t, akkor a hiba  
zérushoz tart.



# A valóság (természet) metrikus?

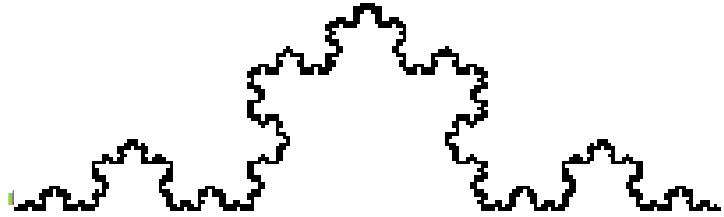
- Idáig a virtuális világ = euklideszi, gömbi, hiperbolikus
  - „Sima” egyenesre/síkra épít
    - Kicsiben mindenki lineáris: **Skálafüggőség**, differenciálható
  - Méret = lefedés: Hossz (1D) = szakasz; felület (2D) = négyzet; térfogat (3D) = kocka
  - Dimenzió: koordináták minimális száma; görbe (1D):  $r(t)$ ; felület (2D):  $r(u, v)$ ; **milyen méret értelmes?**



- **Természet**
  - **Skálafüggetlen**, azaz közelről olyan, mint messziről
  - Nem lesz kicsiben sem lineáris
  - Nem differenciálható
  - Szakasz, négyzet, kocka lefedés nem működik (dimenzió?)



# (Helge von) Koch görbe: hossz végtelen

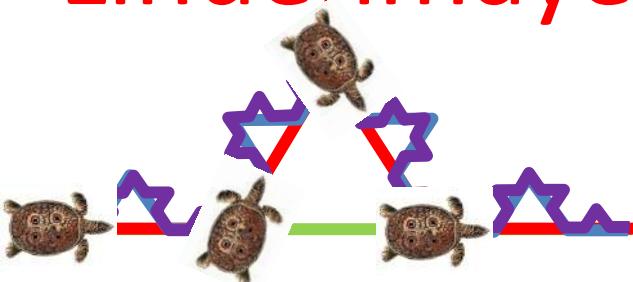
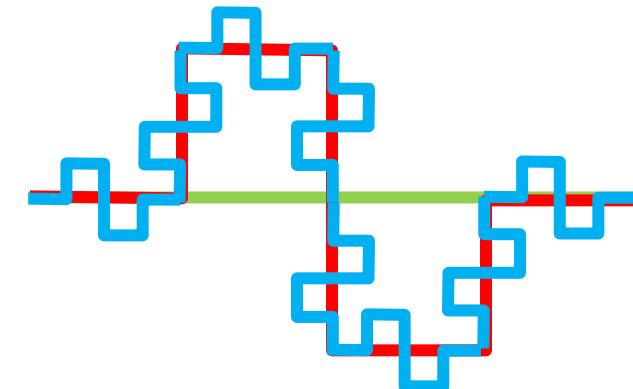
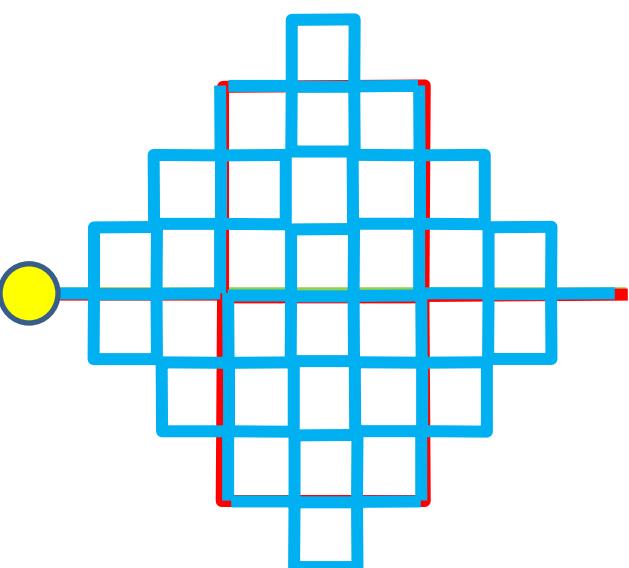


$$l_n = l_0 \left(\frac{4}{3}\right)^n \rightarrow \infty$$

- Véges tartományban végtelen hosszú:
  - Dimenzió  $> 1$
- Területe zérus
  - Dimenzió  $< 2$
- Folytonos
- Sehol sem differenciálható (tüskés)
- Önhasonló



# Lindenmayer (Arisztid) rendszerek


$$F \rightarrow F + 60F - 120F + 60F$$

$$F \rightarrow F + 90F - 90F - 90FF + 90F + 90F - 90F$$

$$F \rightarrow F + 90F - 90F - 90F - 90F + 90F + 90F + 90F - 90F$$

Peano/Hilbert térkitöltő görbe

# (Felix) Hausdorff dimenzió önhasonló objektumokra



$r = 1/2$  kicsinyítés

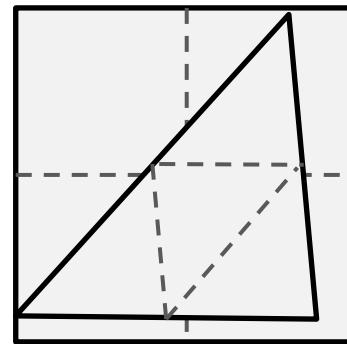
$$N = 1$$



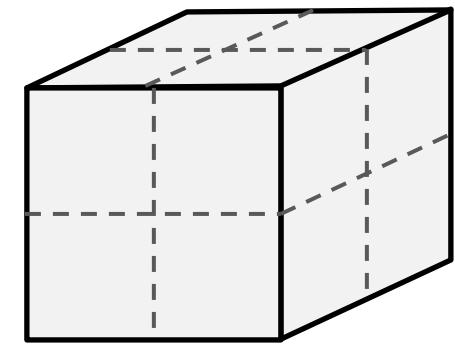
$$N = 2$$



$$N = 4$$



$$N = 8$$

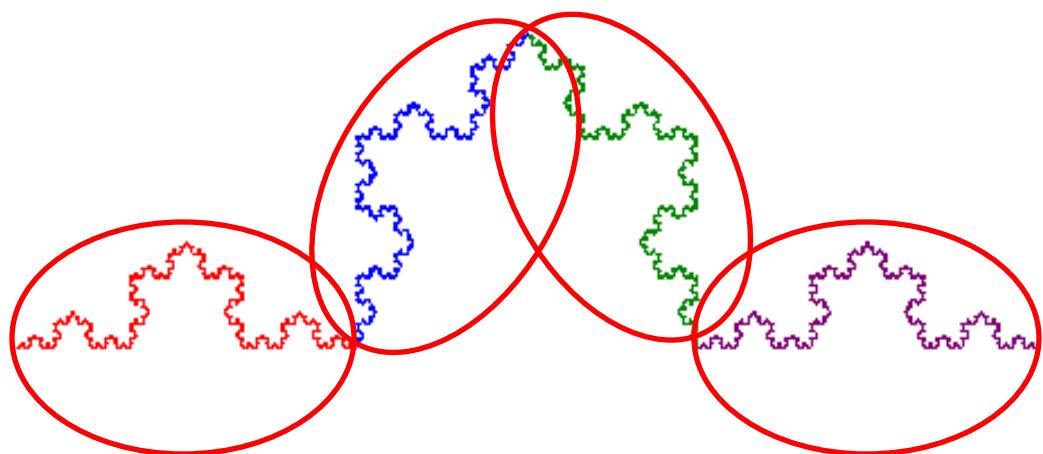


$$N = 1/r^D$$



$$D = \frac{\log(N)}{\log(1/r)}$$

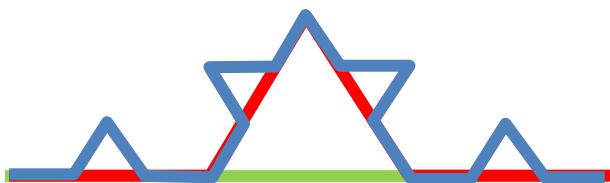
# Koch görbe Hausdorff dimenziója



$$\begin{aligned}r &= 1/3 \\N &= 4\end{aligned}$$

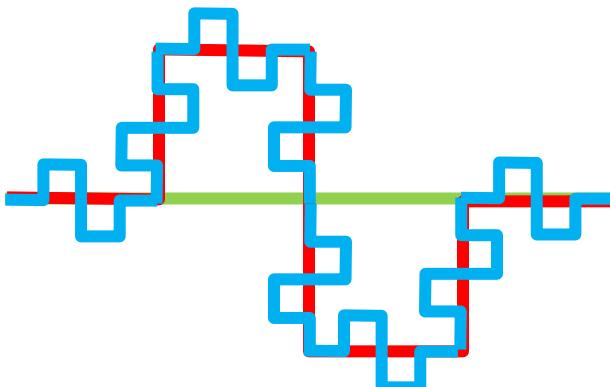
$$D = \frac{\log(4)}{\log(3)} \approx 1.26$$

# Lindenmayer (Arisztid) rendszerek



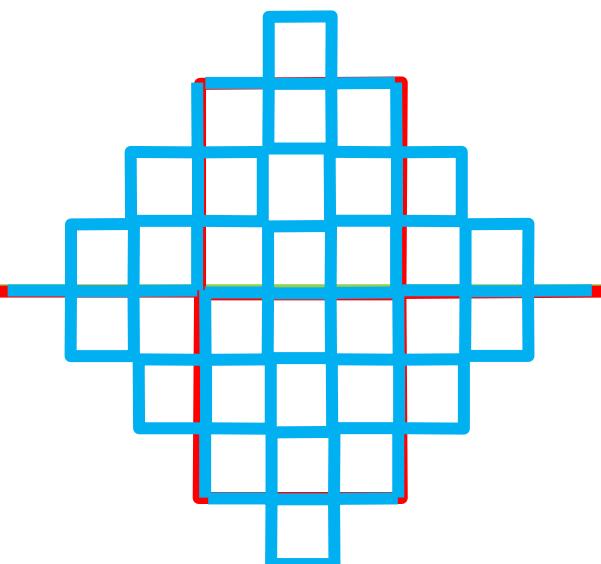
$$F \rightarrow F + 60F - 120F + 60F$$

$$r = 1/3, N = 4, D = \log(4)/\log(3) = 1.26$$



$$F \rightarrow F + 90F - 90F - 90FF + 90F + 90F - 90F$$

$$r = 1/4, N = 8, D = \log(8)/\log(4) = 1.5$$

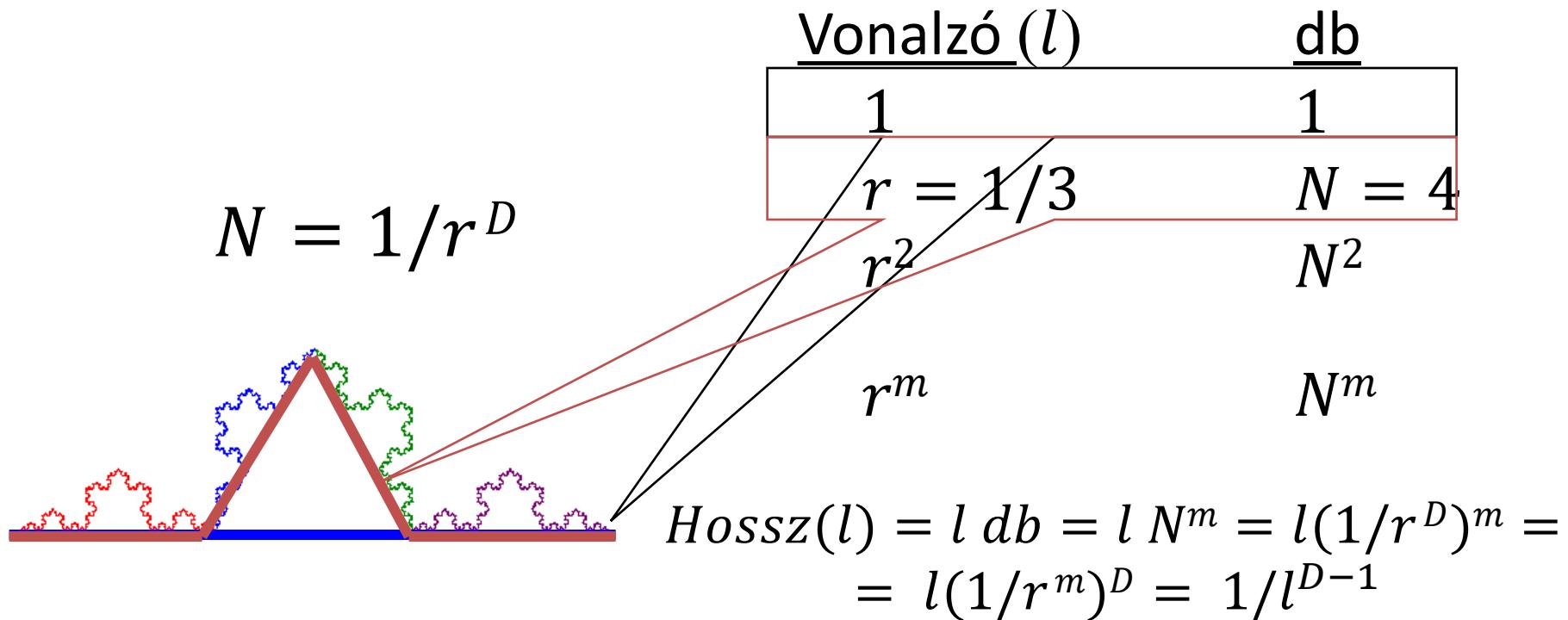


$$F \rightarrow F + 90F - 90F - 90F - 90F + 90F + 90F + 90F - 90F$$

$$r = 1/3, N = 9, D = \log(9)/\log(3) = 2$$

Peano/Hilbert térkitöltő görbe

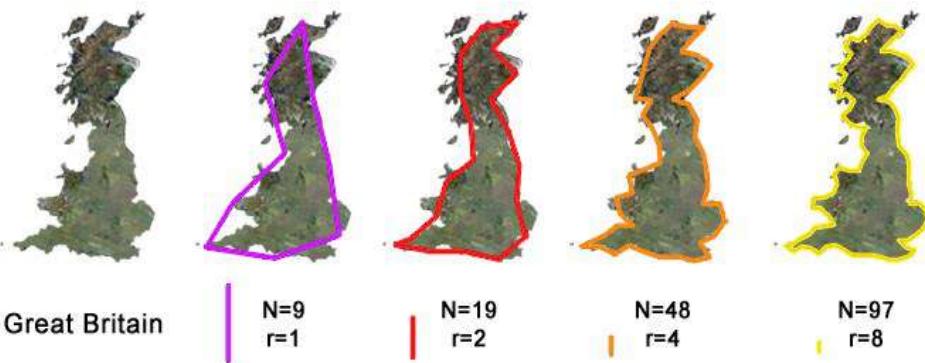
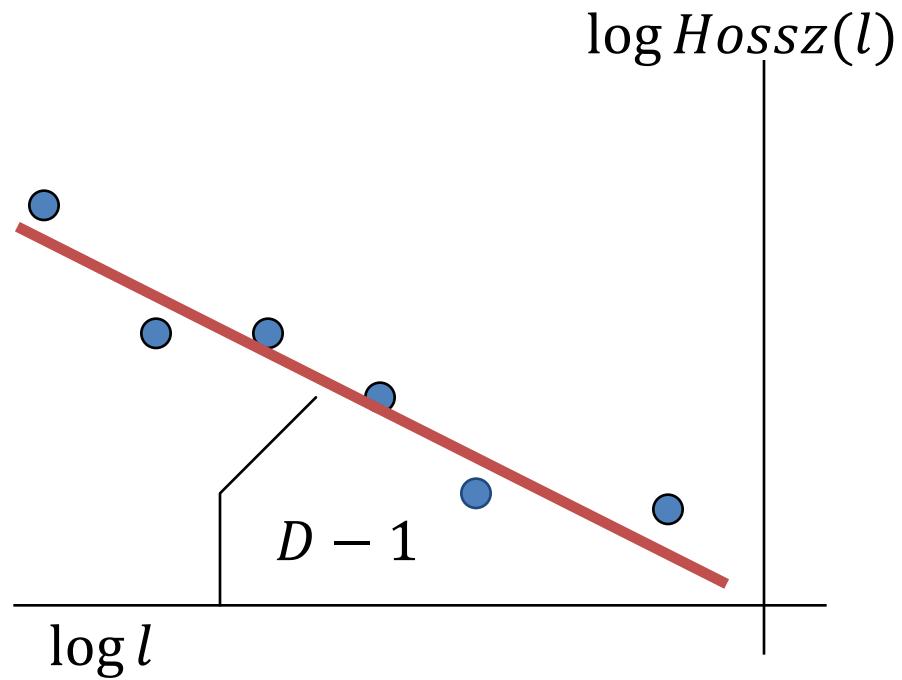
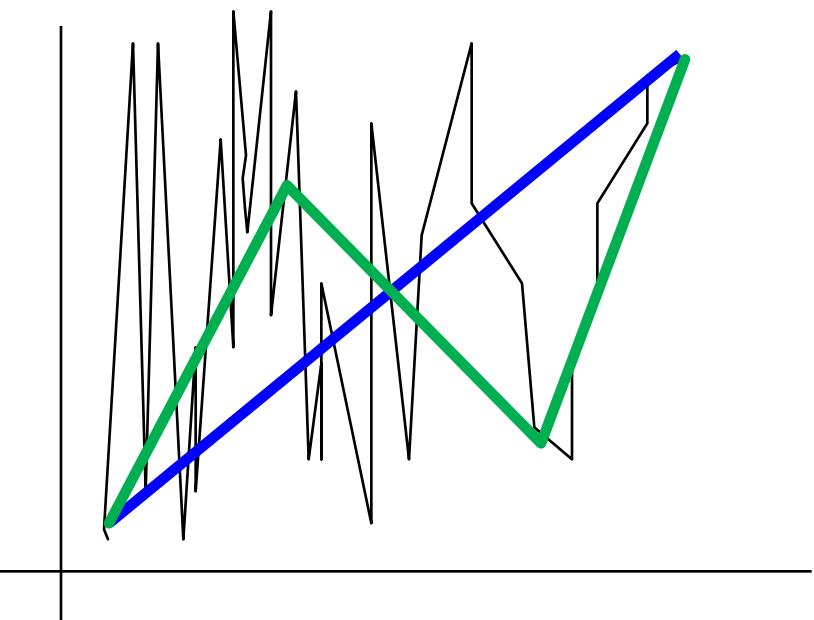
# Nem önhasonló objektumok: vonalzó dimenzió



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

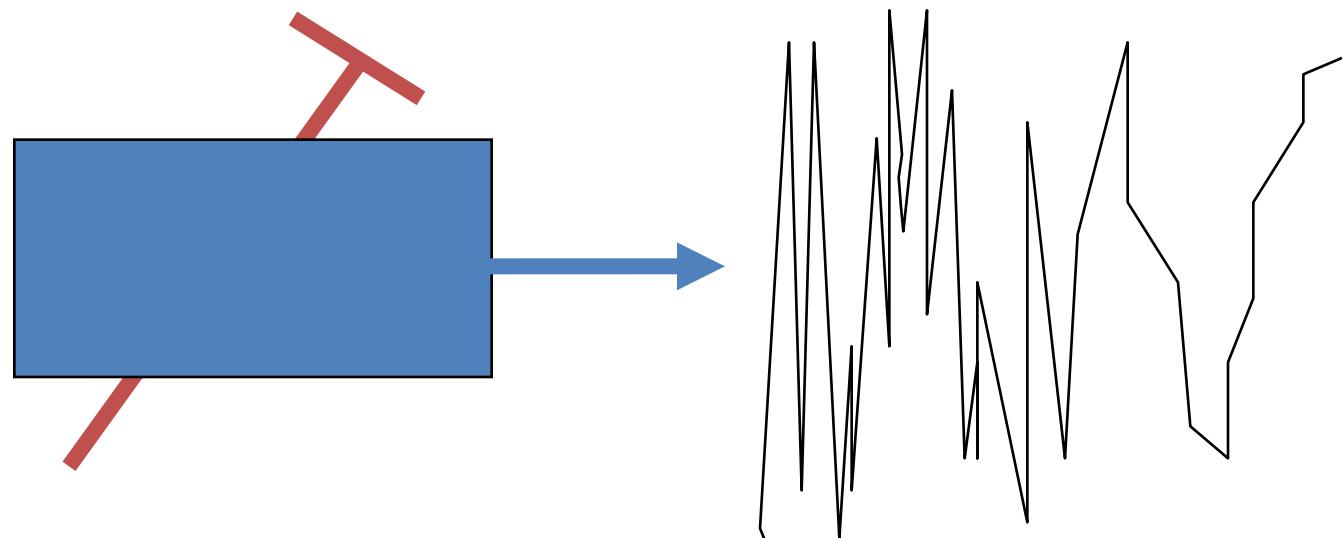
# Dimenziómérés = hosszmérés

EEG görbe



Alkalmazás:  
Természetes objektumok  
elkülönítése és kategorizálása

# Fraktálok előállítása

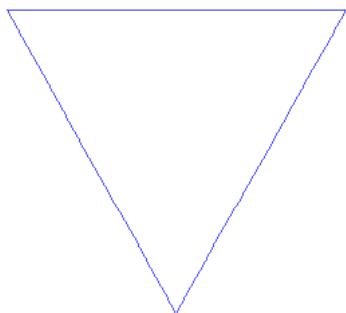


## Matematikai gépek (algoritmusok):

- L-rendszerék
- Fraktális ( $1/f$ ) zaj: Brown mozgás, Perlin zaj
- Kaotikus dinamikus rendszerek (véletlenszám generátor)

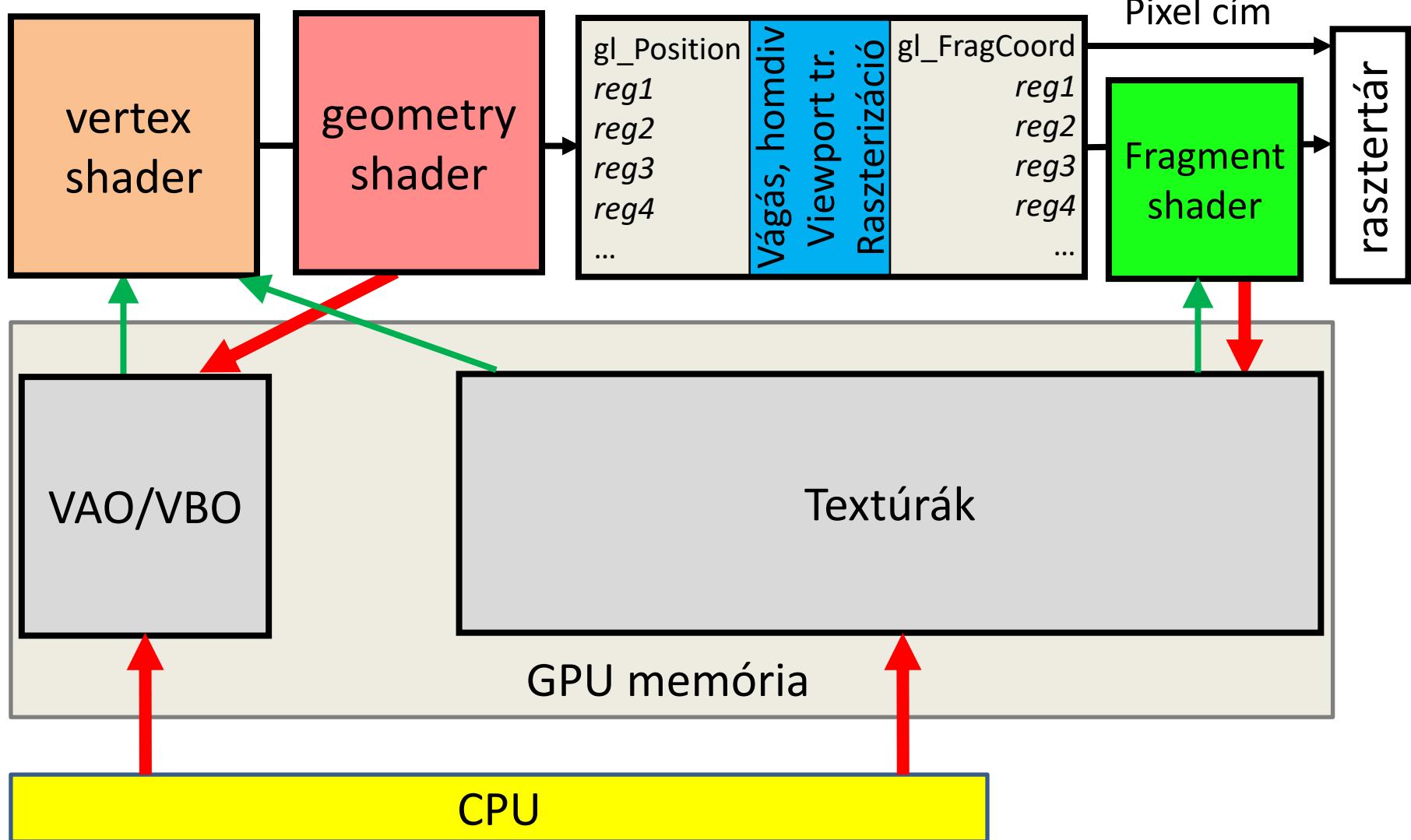
# Lindenmayer rendszerek

$F \rightarrow F+60F-120F+60F$

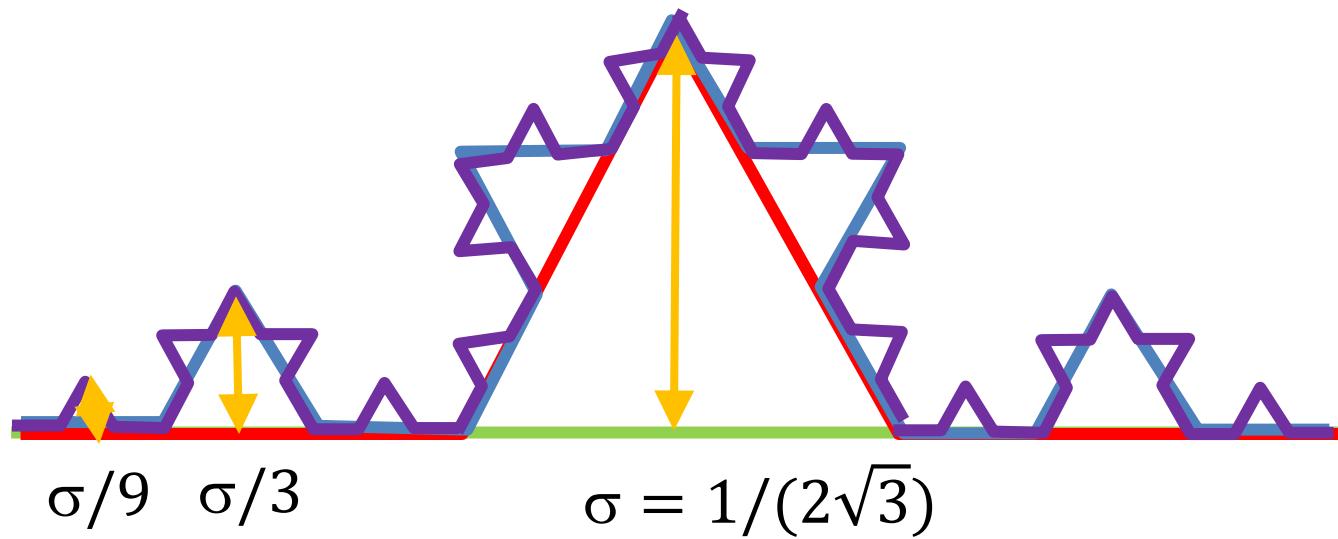


# Geometry shader

csúcspontok → primitívek → pixelek



# Fraktális zaj



Skála

$$N = 4$$

$$N^2 = 16$$

$$N^m = 4^m$$

Ré  $\overset{?}{\epsilon}$  et

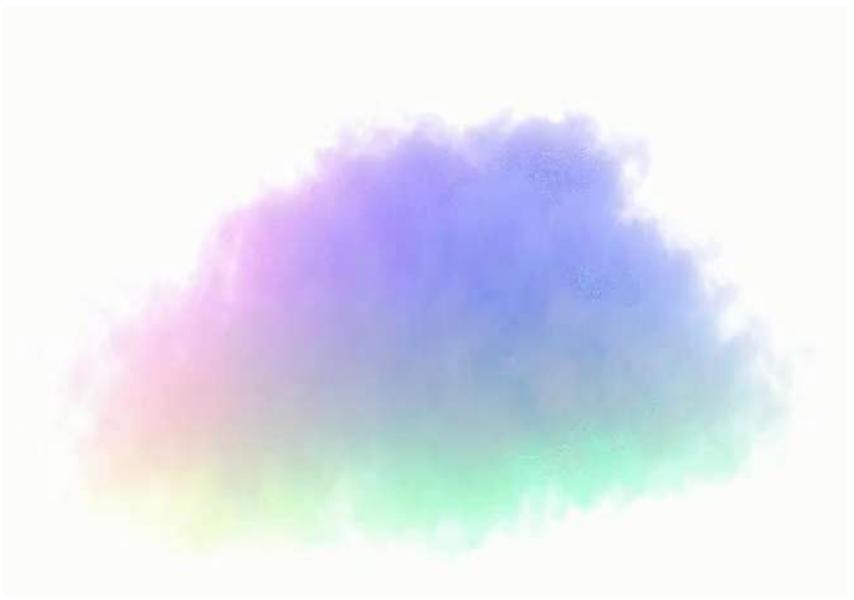
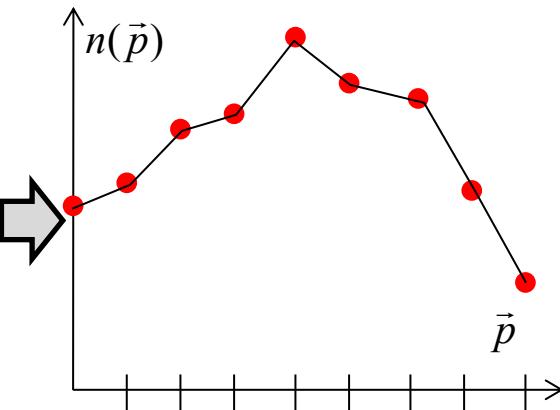
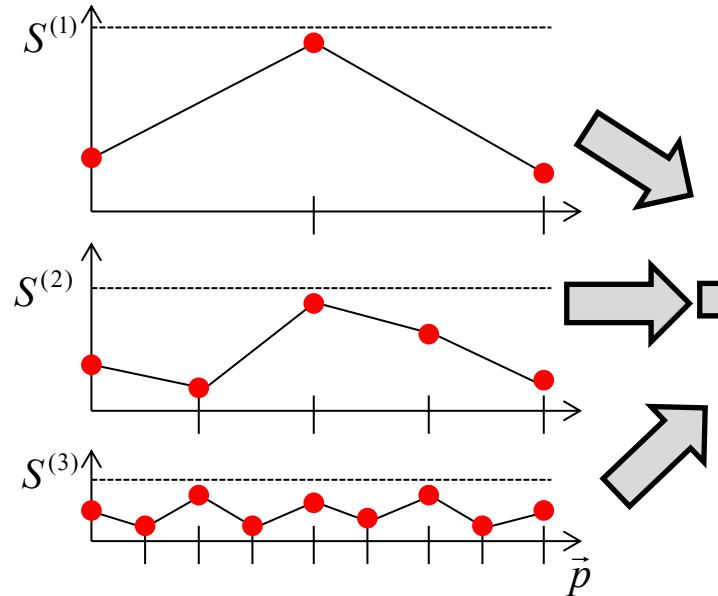
$$\sigma/3$$

$$\dots$$
  
$$\sigma/3^{m-1}$$



szórás

# (Ken) Perlin zaj



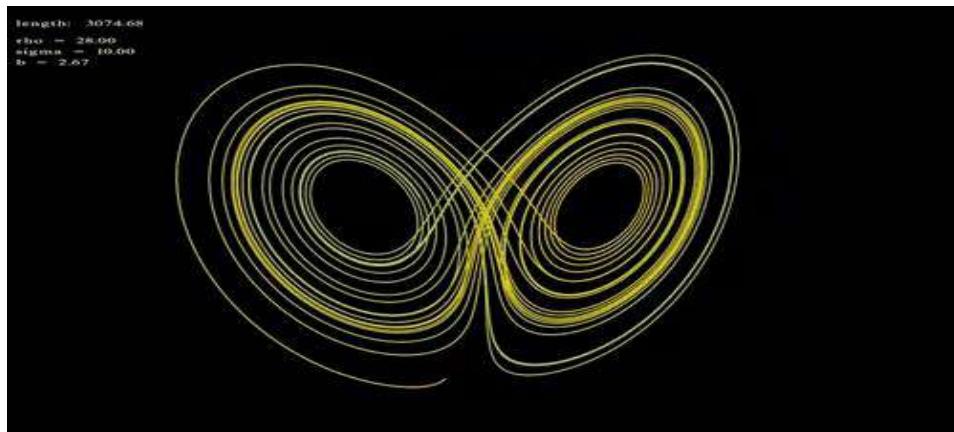
*"Invention, it must be humbly admitted,  
does not consist in creating out of void  
but out of chaos."*

*Mary Shelly*

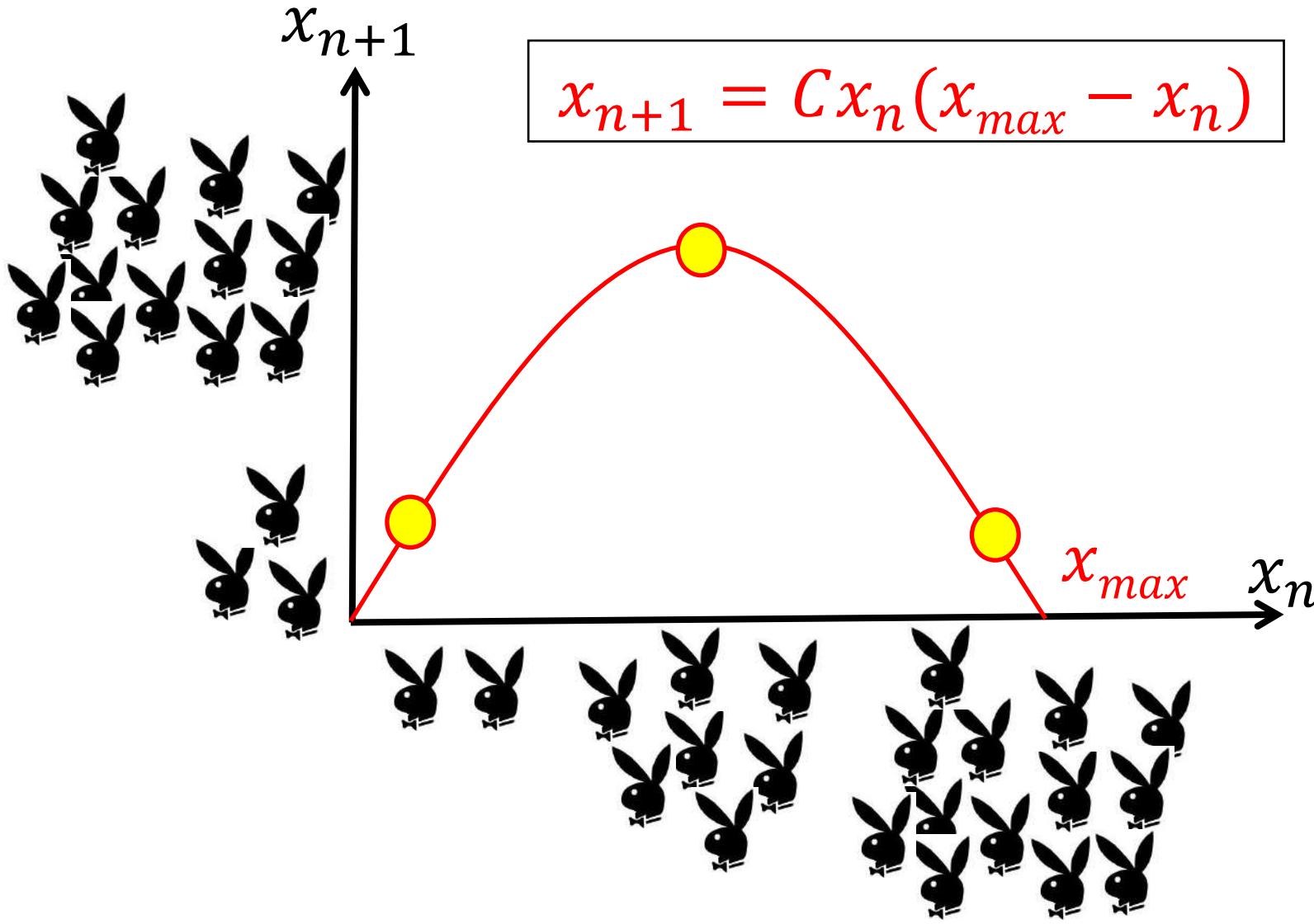
# Fraktálok és káosz

## 2. Káosz

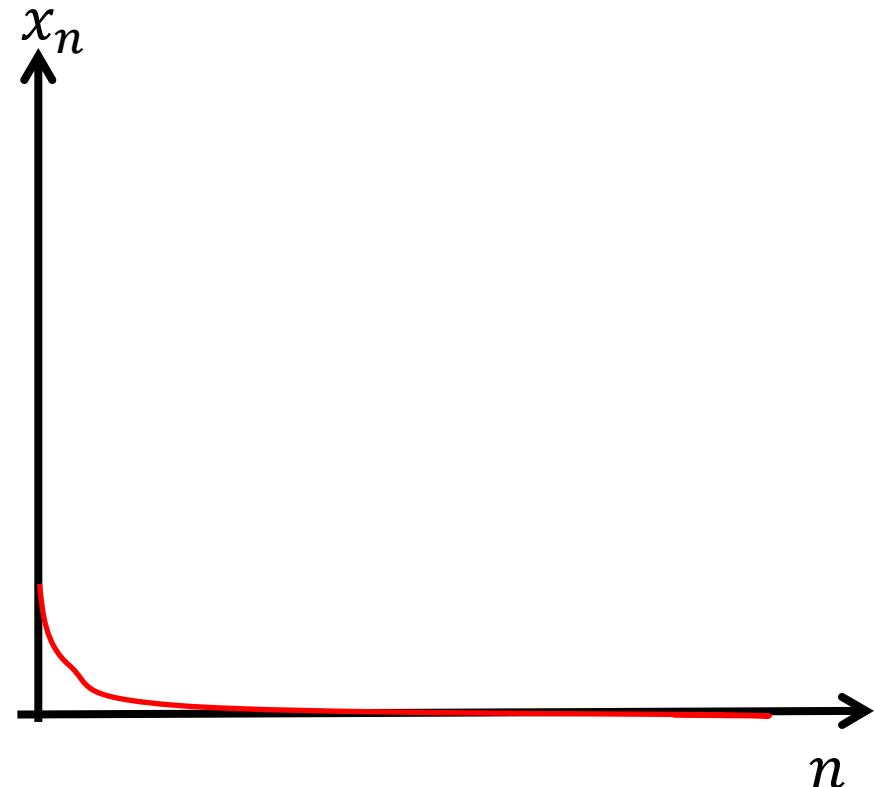
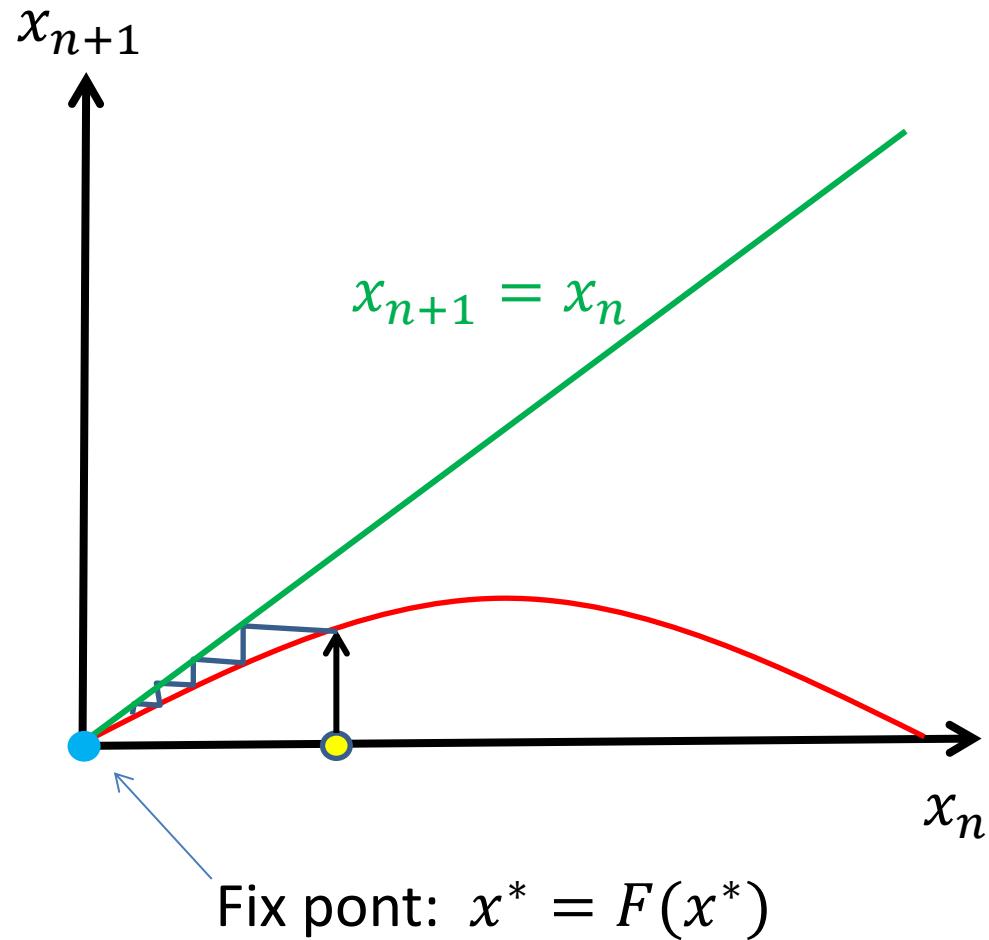
Szirmay-Kalos László



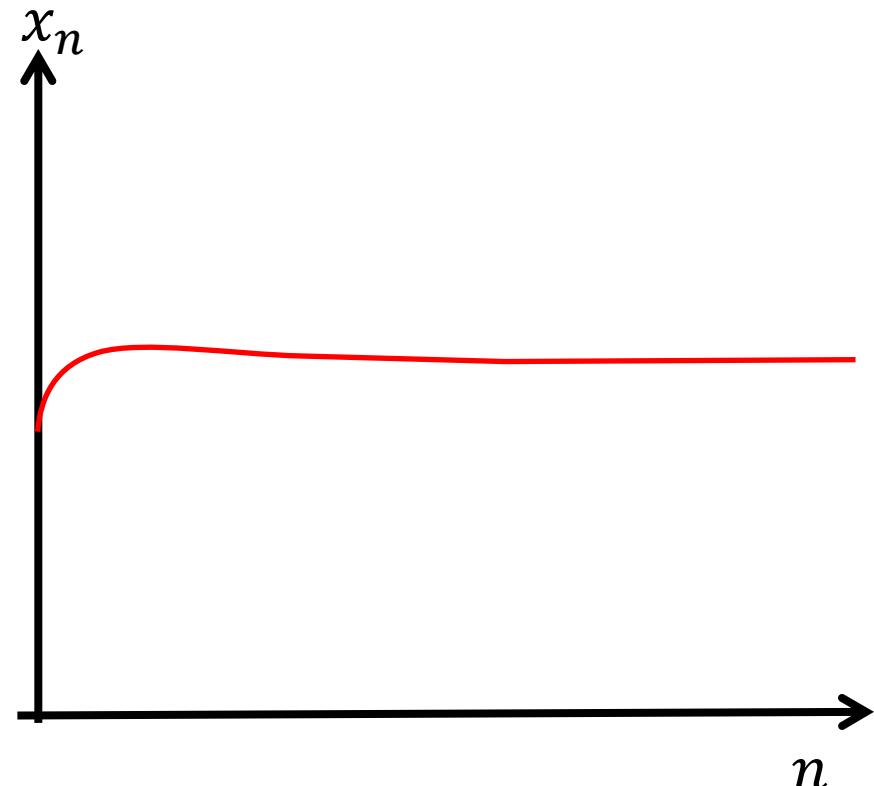
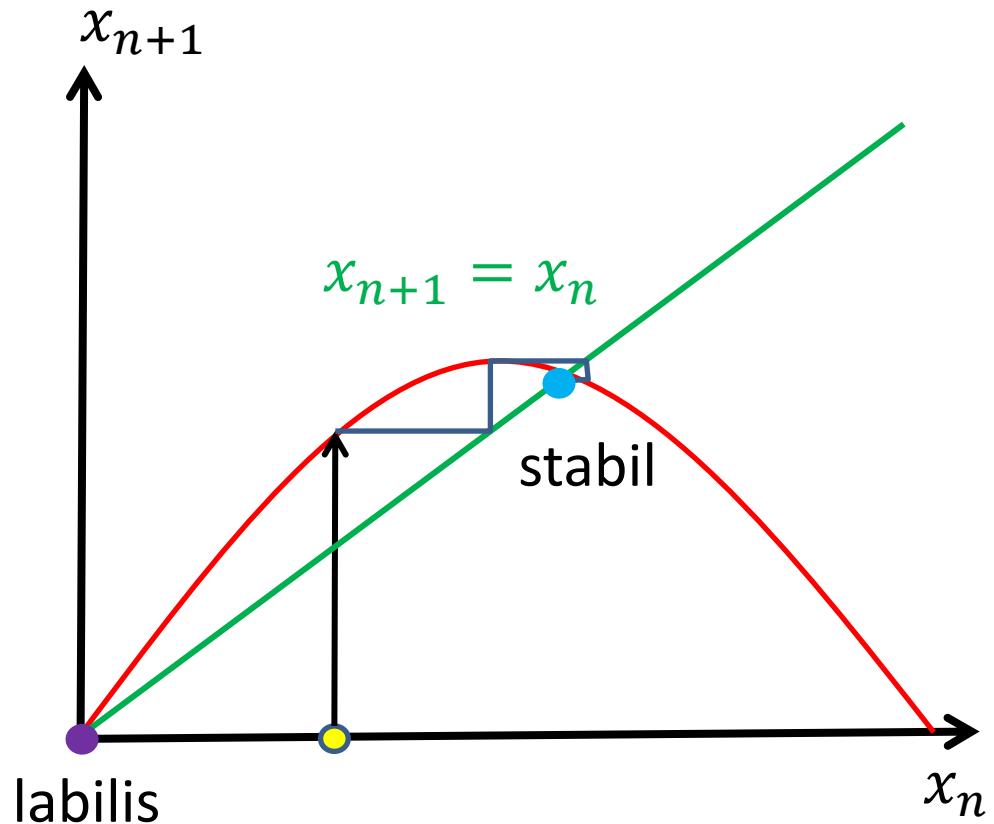
# Nyulak szigete



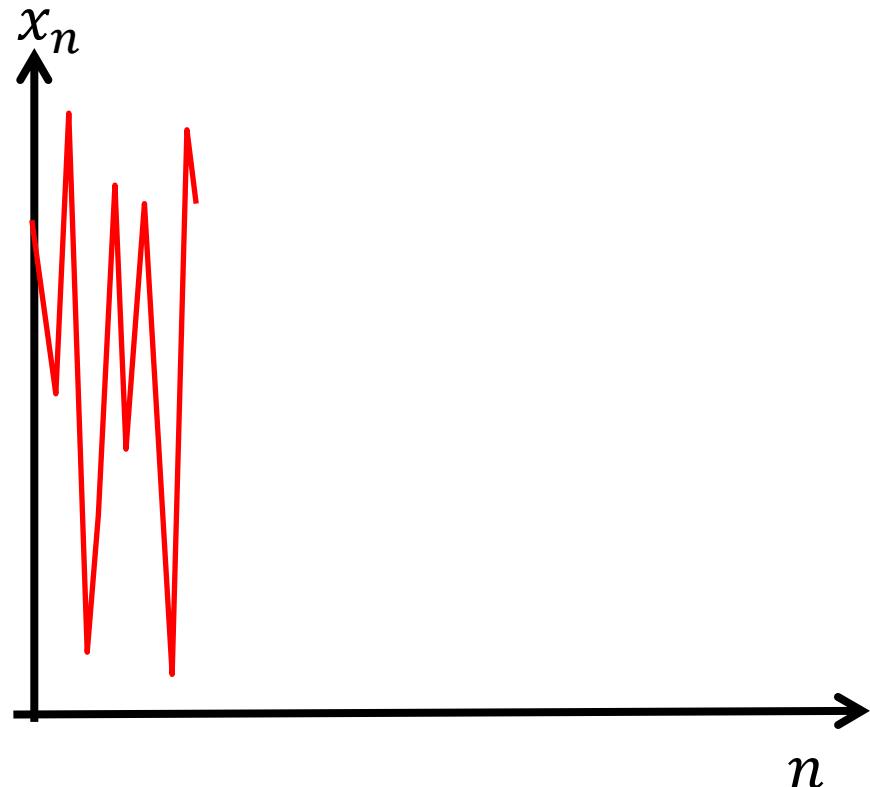
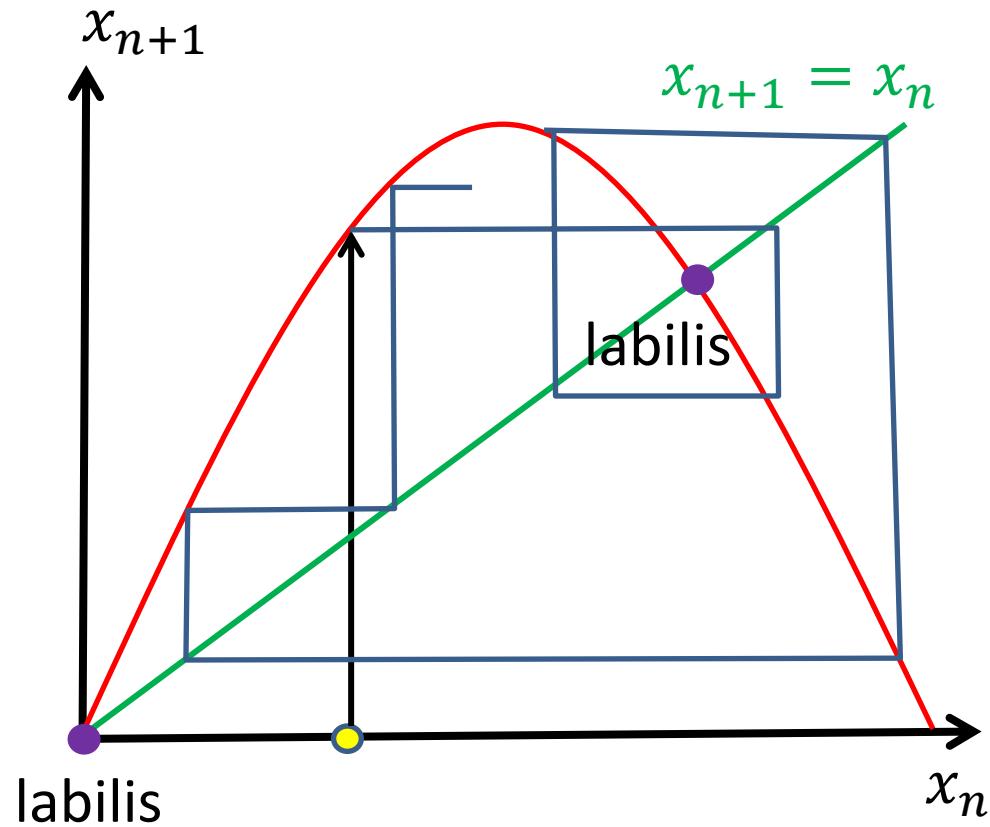
# Nyulak kis C értékre: $x_{n+1} = Cx_n(x_{max} - x_n)$



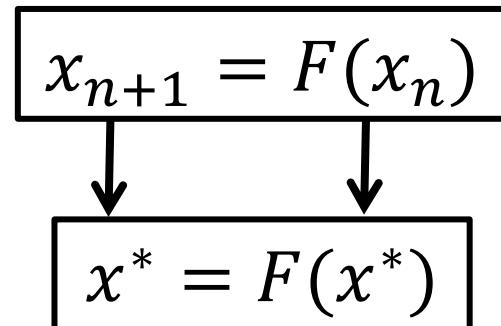
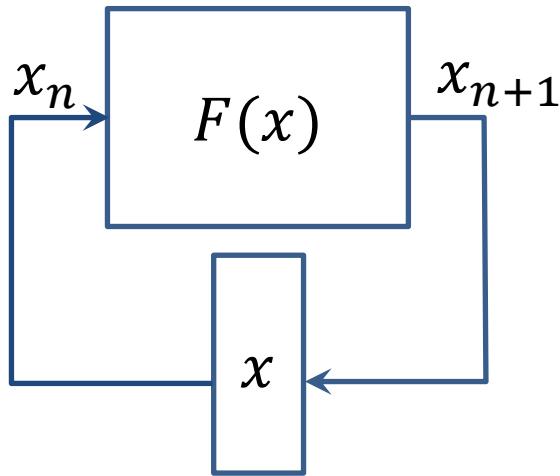
Közepes  $C$  értékre:  $x_{n+1} = Cx_n(x_{max} - x_n)$



Nagy  $C$  értékre:  $x_{n+1} = Cx_n(x_{\max} - x_n)$



# Iterált függvények, fix pont



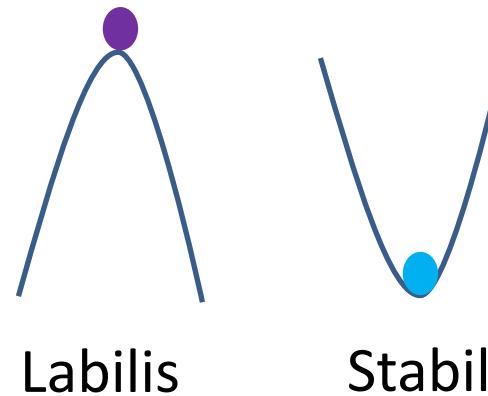
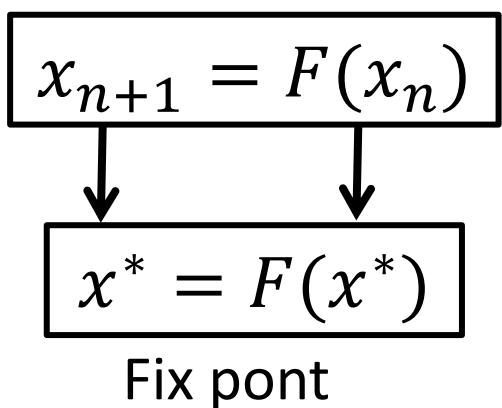
Fix pont

Viselkedés (trajektória):

- $F(x)$
- $x_0$

$$\begin{aligned}x_1 &= F(x_0) \\x_2 &= F(x_1) = F(F(x_0)) \\x_3 &= F(x_2) = F(F(F(x_0))) \\&\vdots\end{aligned}$$

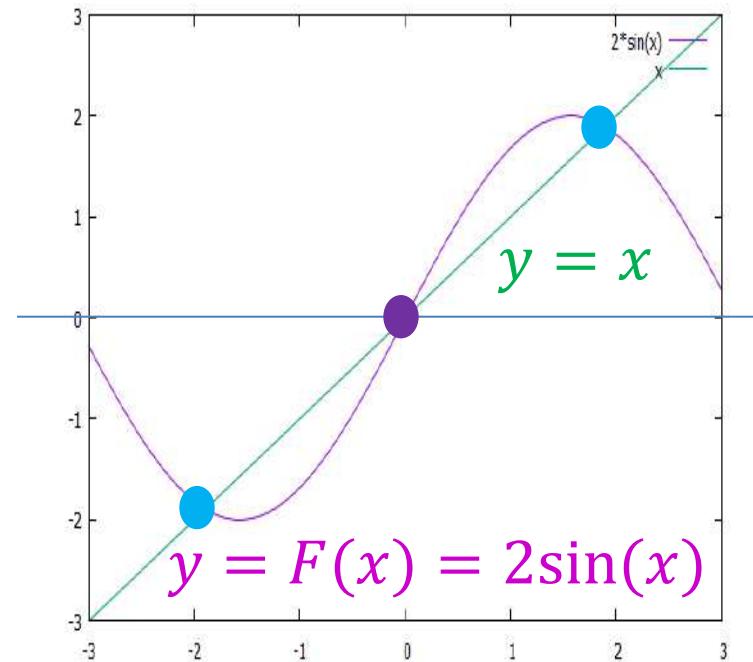
# Iterált függvények, fix pont stabilitás



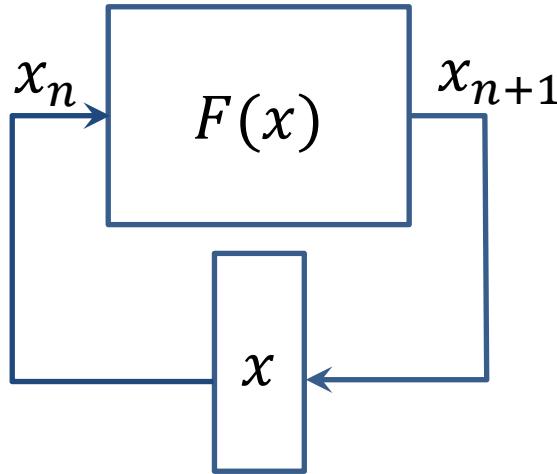
$$\begin{aligned} x^* + \Delta x_{n+1} &= F(x^* + \Delta x_n) \\ &\approx F(x^*) + F'(x^*) \cdot \Delta x_n \end{aligned}$$

$$|\Delta x_{n+1}| \approx |F'(x^*)| \cdot |\Delta x_n|$$

Stabilitás:  $|F'(x^*)| < 1$



# A valóság (természet) szimulálható?



Ha  $F$  és  $x_0$  csak közelítőleg ismert, akkor  $x_n$  is csak közelítés lesz, de ha pontosítjuk  $F$ -t és  $x_0$ -t, akkor a hiba zérushoz tart?

$$x_1 = F(x_0) = \tilde{F}(\tilde{x}_0) + \tilde{F}'(\tilde{x}_0)\Delta x_0 + \Delta F(\tilde{x}_0)$$

$$x_2 = F(x_1) = \tilde{F}(\tilde{F}(\tilde{x}_0)) + \tilde{F}'(\tilde{x}_1)\tilde{F}'(\tilde{x}_0)\Delta x_0 + \tilde{F}'(\tilde{x}_1)\Delta F(\tilde{x}_0) + \Delta F(\tilde{x}_1)$$

...

$$x_n = F(x_{n-1}) = \tilde{F}(\tilde{F} \dots (\tilde{x}_0)) + \tilde{F}'(\tilde{x}_{n-1}) \dots \tilde{F}'(\tilde{x}_1)\tilde{F}'(\tilde{x}_0)\Delta x_0 +$$

$$\tilde{F}'(\tilde{x}_{n-1}) \dots \tilde{F}'(\tilde{x}_0)\Delta F(\tilde{x}_0) +$$

$$\tilde{F}'(\tilde{x}_{n-1}) \dots \Delta F(\tilde{x}_1) +$$

...

$$\Delta F(\tilde{x}_{n-1})$$



Akkumulált hiba:

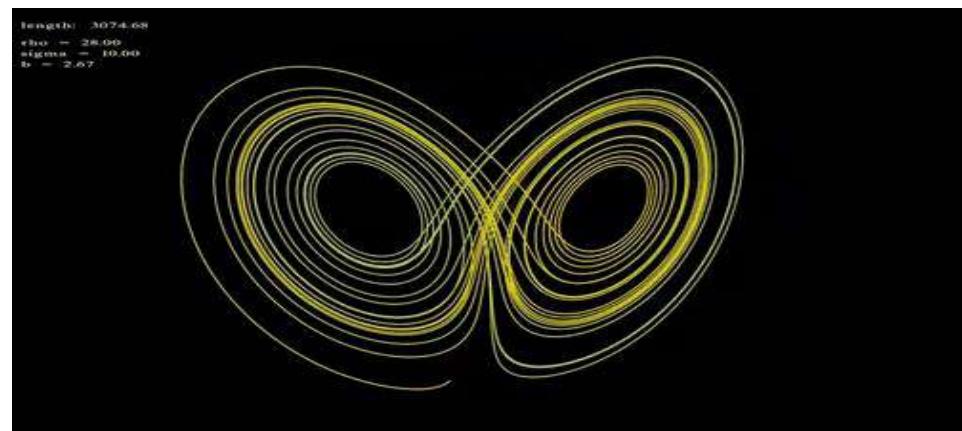
Ha  $|\tilde{F}'| > 1$ , akkor divergál,  
hiába tart  $\Delta x_0$  és  $\Delta F$  zérushoz

# Káosz

A rendszer determinisztikus, de

- Kezdeti állapot hatása eltűnik
  - Kis perturbáció nagyon eltérő viselkedéshez vezet
- Auto-korrelációs függvény zérushoz tart
  - Korábbi állapot gyengén befolyásolja a sokkal későbbit
  - Megjósolhatatlanság
- Teljesítmény sűrűség spektrum nem tart zérushoz
  - Nagy frekvencia

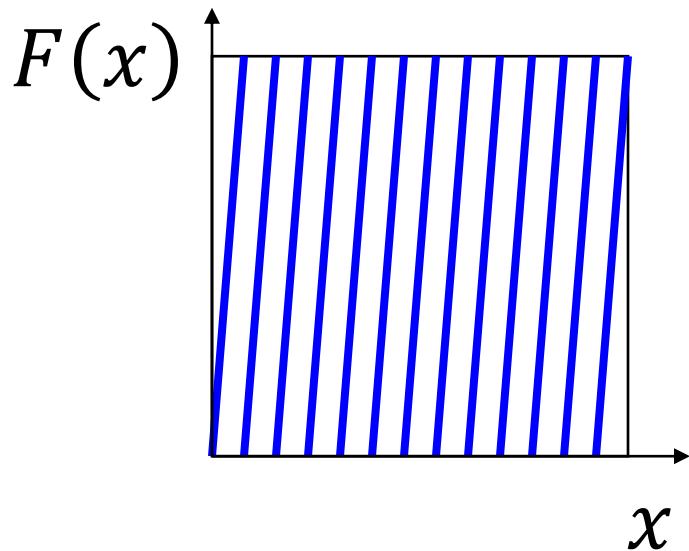
Edward Lorenz



# Pszeudó véletlenszám generátor

```
static uint x = 3;  
  
void seed(uint s) { x = s; }  
  
uint rand( ) {  
    x = F(x);  
    return x;  
}
```

- $x_{n+1} = F(x_n)$
- $|F'(x)| > 1$ ,  
nagy és állandó



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

- $\{z\} = z$  tört része
- $g$  nagy

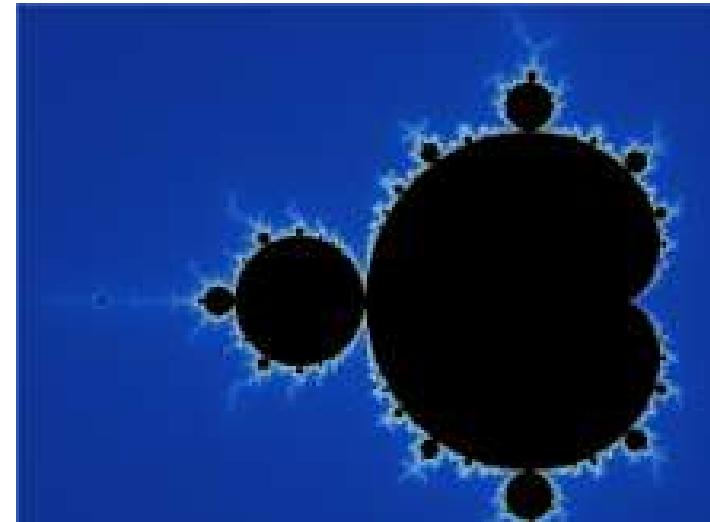
*"There's no sense in being precise when you  
don't even know what you're talking about."*

Neumann János

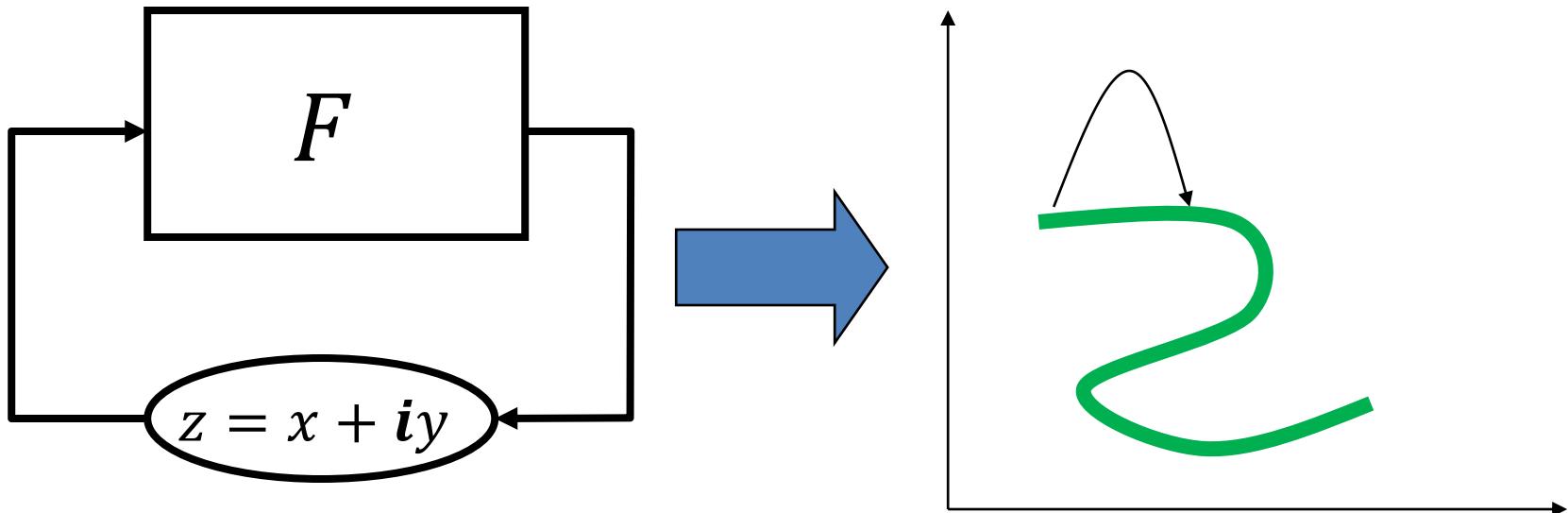
# Fraktálok és káosz

## 3. Kaotikus rendszerek a síkon

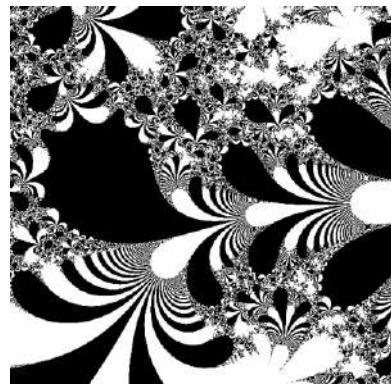
Szirmay-Kalos László



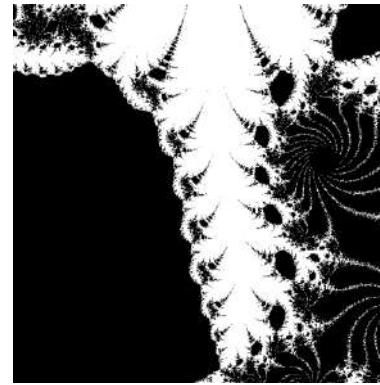
# Kaotikus rendszerek a síkon



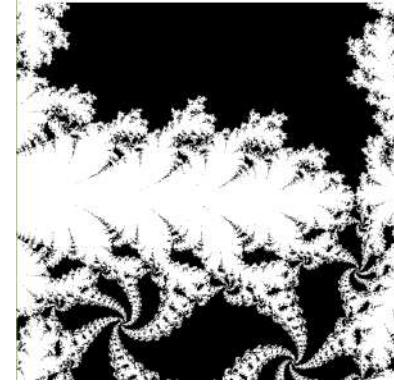
$$F(z) = z^2 + c$$



$$F(z) = e^z + c$$

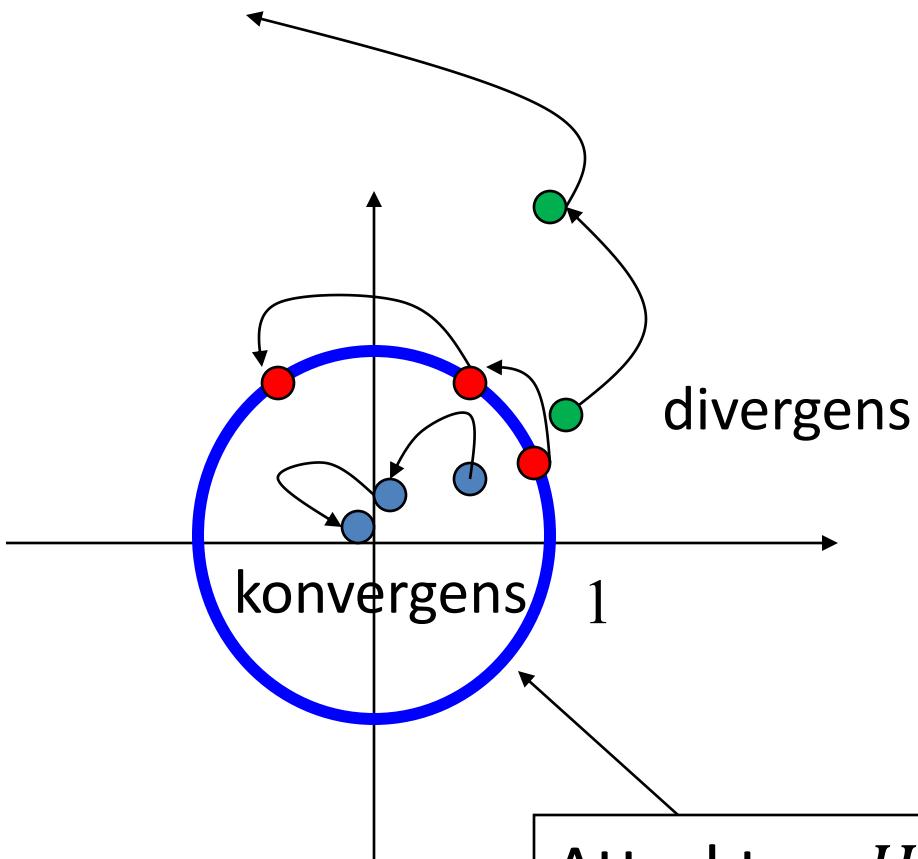


$$F(z) = \cos(z + c)$$



$$F(z) = \sinh(z + c)$$

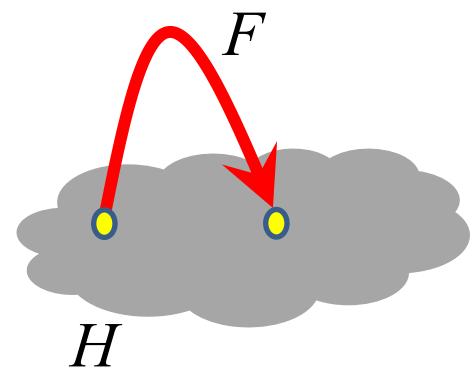
$$F: z \rightarrow z^2$$



$$z = re^{i\varphi}$$

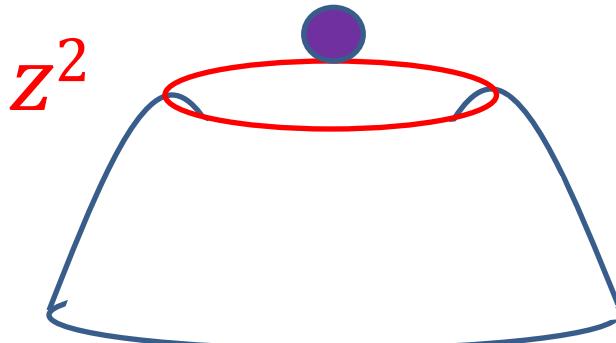
$$r \rightarrow r^2$$

$$\varphi \rightarrow 2\varphi$$

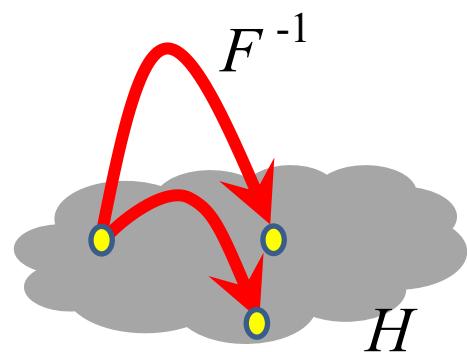


# Attraktor felrajzolása

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



# Inverz iterációs módszer



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


---

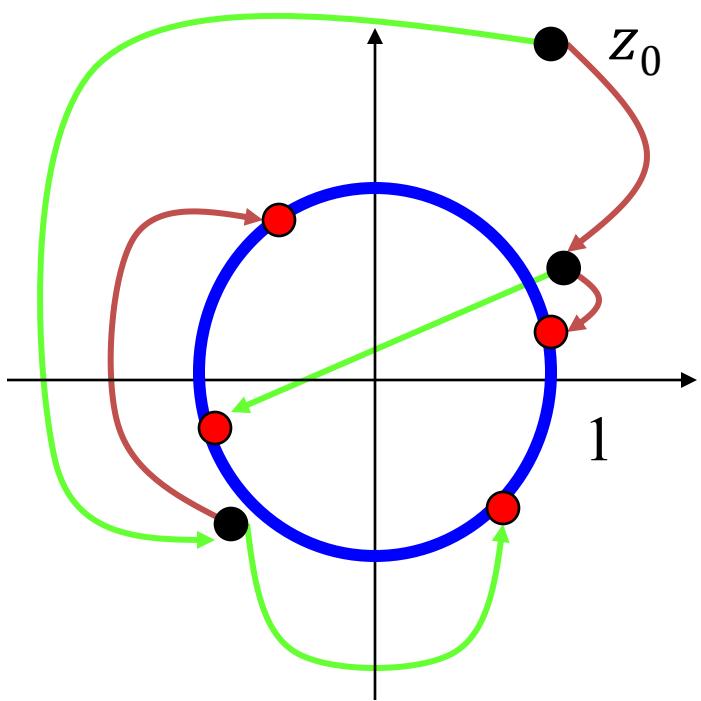
$$z_{n+1} = z_n^2$$

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

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

$$\varphi_{n+1} = \varphi_n/2 + \pi\{0|1\}$$


---



Ha  $n$  nagy:

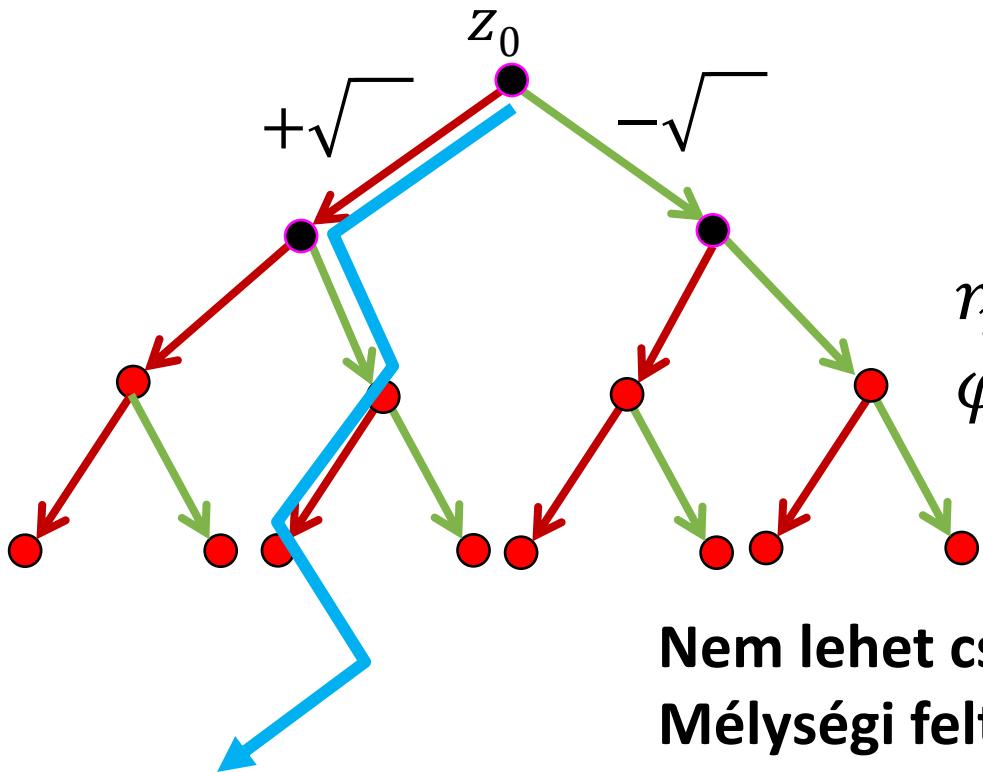
$$r_n = \sqrt[2^n]{r_0} \approx 1$$

$$\varphi_n = \frac{\varphi_0}{2^n} + \pi\{0|1\}.\{0|1\}\{0|1\} \dots$$

$$\approx \pi\{0|1\}.\{0|1\}\{0|1\} \dots$$

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

# Többértékű leképzés: Bolyongás



$$r_n \approx 1$$

$$\varphi_n \approx \pi \{0|1\}. \{0|1\} \{0|1\} \dots$$

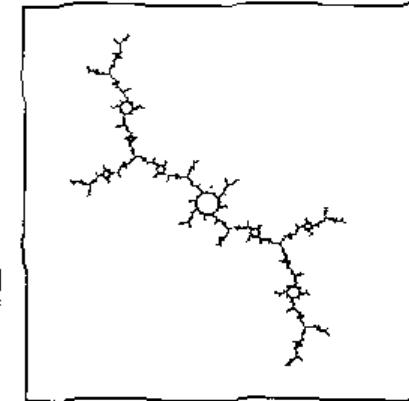
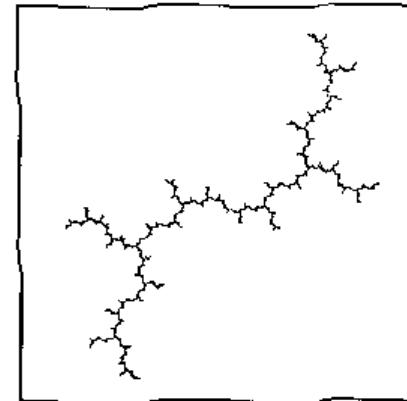
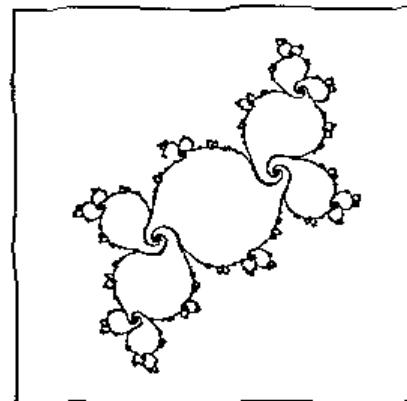
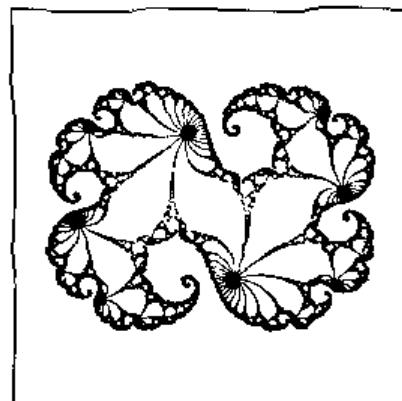
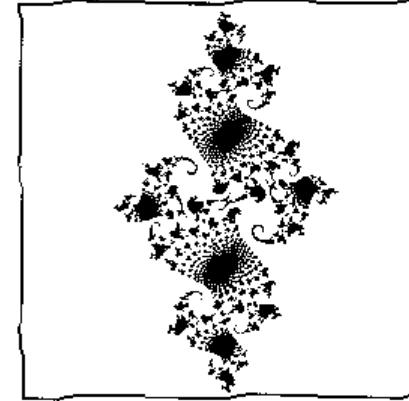
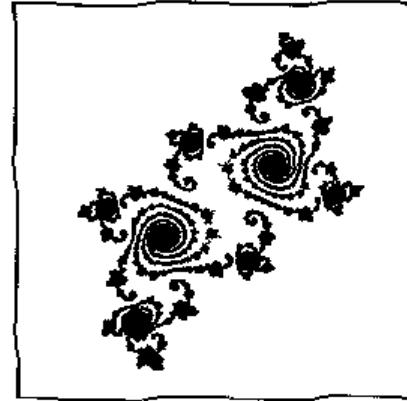
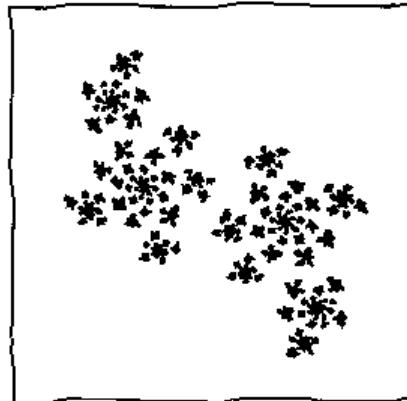
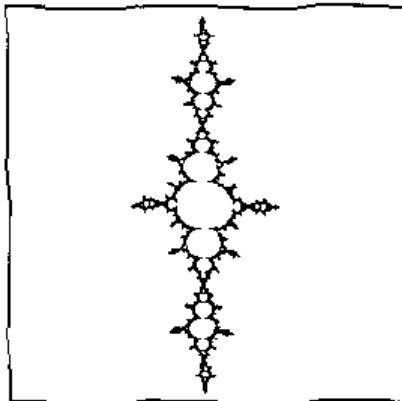
$n \quad n-1 \quad n-2$

Nem lehet csak egy értékkel dolgozni ???  
Mélységi feltárás szélességi helyett???

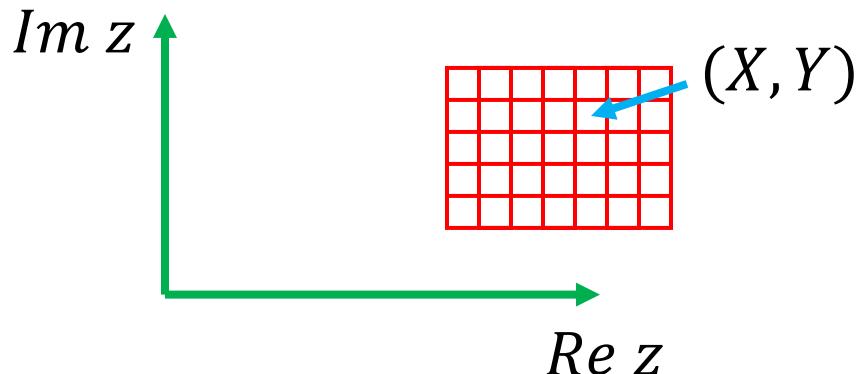
- Csak a  $+$ :  $\varphi_n \approx 0.00 \dots \cdot \pi = 0$
- Csak a  $-$ :  $\varphi_n \approx 1.1 \dots \cdot \pi = 2\pi \sim 0$
- Felváltva:  $\varphi_{2n} \approx 1.010 \dots \cdot \pi = \frac{4\pi}{3}$ ;  $\varphi_{2n+1} \approx 0.1010 \dots \cdot \pi = \frac{2\pi}{3}$
- **Véletlenszerűen!**

# (Gaston) Julia halmaz

$$F: z \rightarrow z^2 + c$$



# Kitöltött Julia halmaz: algoritmus



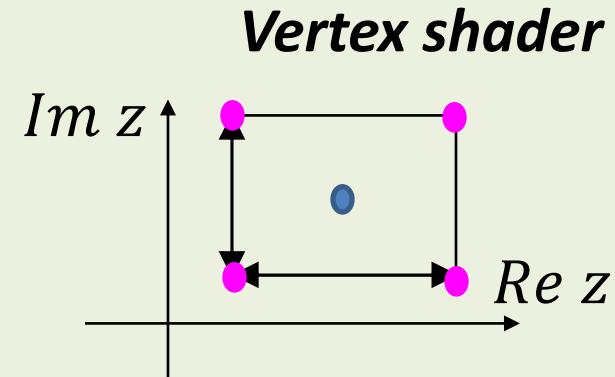
```
FilledJulia( Complex c ) {
    for(Y = 0; Y < Ymax; ++Y) {
        for(X = 0; X < Xmax; ++X) {
            Complex z = ViewportWindow(X, Y);
            for(n = 0; n < infinity; ++n) z = z2 + c
            image[Y][X] = (|z| < infinity) ? black : white;
        }
    }
}
```

# GPU implementáció

```
float cVtx[] = { -1, -1, 1, -1, 1, 1, -1, 1 }; CPU program
```

```
glBufferData(GL_ARRAY_BUFFER, sizeof(cVtx), cVtx, GL_STATIC_DRAW);  
...  
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

```
uniform vec2 cameraCenter, cameraSize;  
layout(location = 0) in vec2 cVertex;  
out vec2 z0;  
  
void main() {  
    gl_Position = vec4(cVertex, 0, 1);  
    z0 = cVertex * cameraSize/2 + cameraCenter;  
}
```



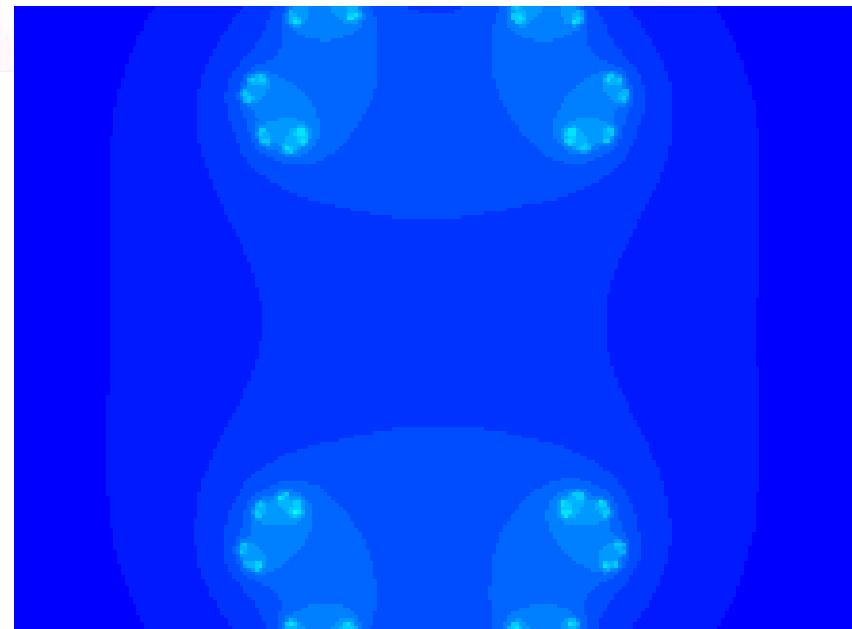
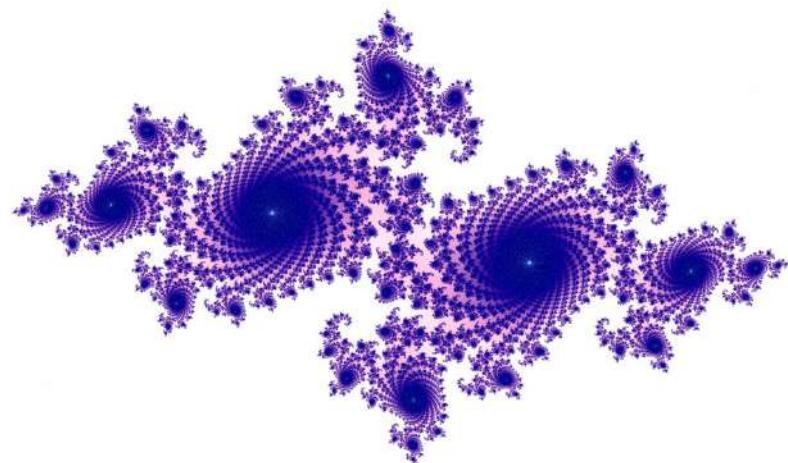
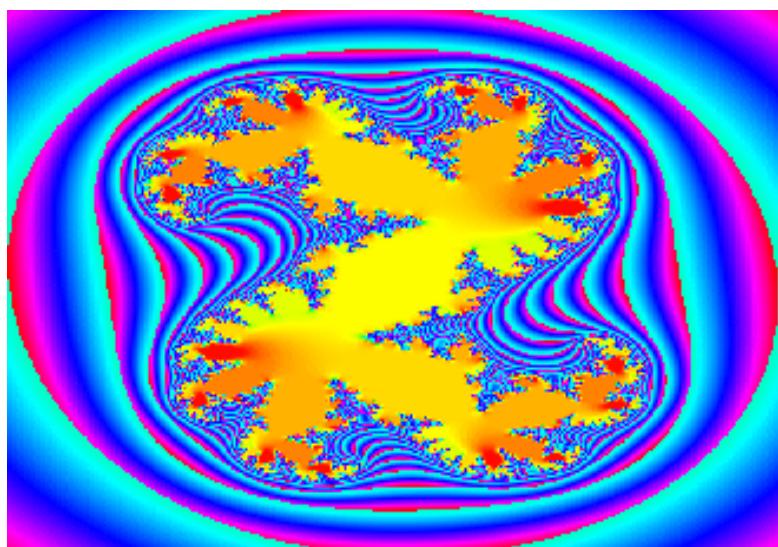
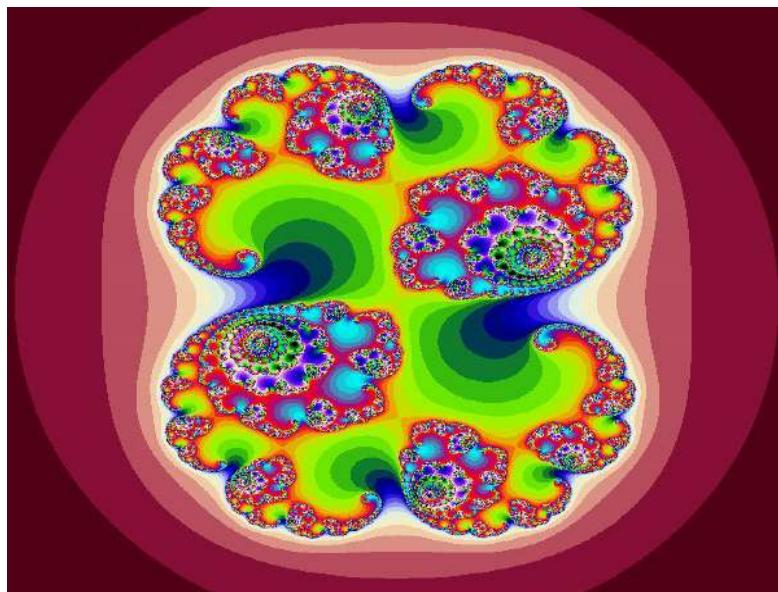
```
uniform vec2 c;  
in vec2 z0;  
out vec4 fragCol;  
  
void main() {  
    vec2 z = z0;  
    for(int i=0; i<1000; i++) z = vec2(z.x*z.x-z.y*z.y, 2*z.x*z.y) + c;  
    fragCol = (dot(z,z) < 100) ? vec4(0, 0, 0, 1) : vec4(1, 1, 1, 1);  
}
```

*Fragment shader*

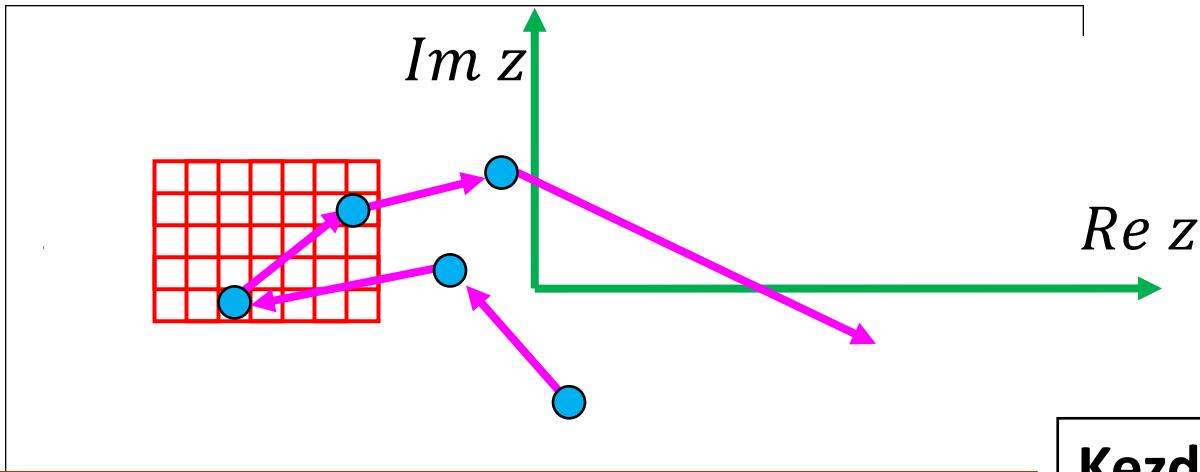
$$z = z^2 + c$$



# Kitöltött Julia halmaz: kép



# Julia halmaz inverz iterációval

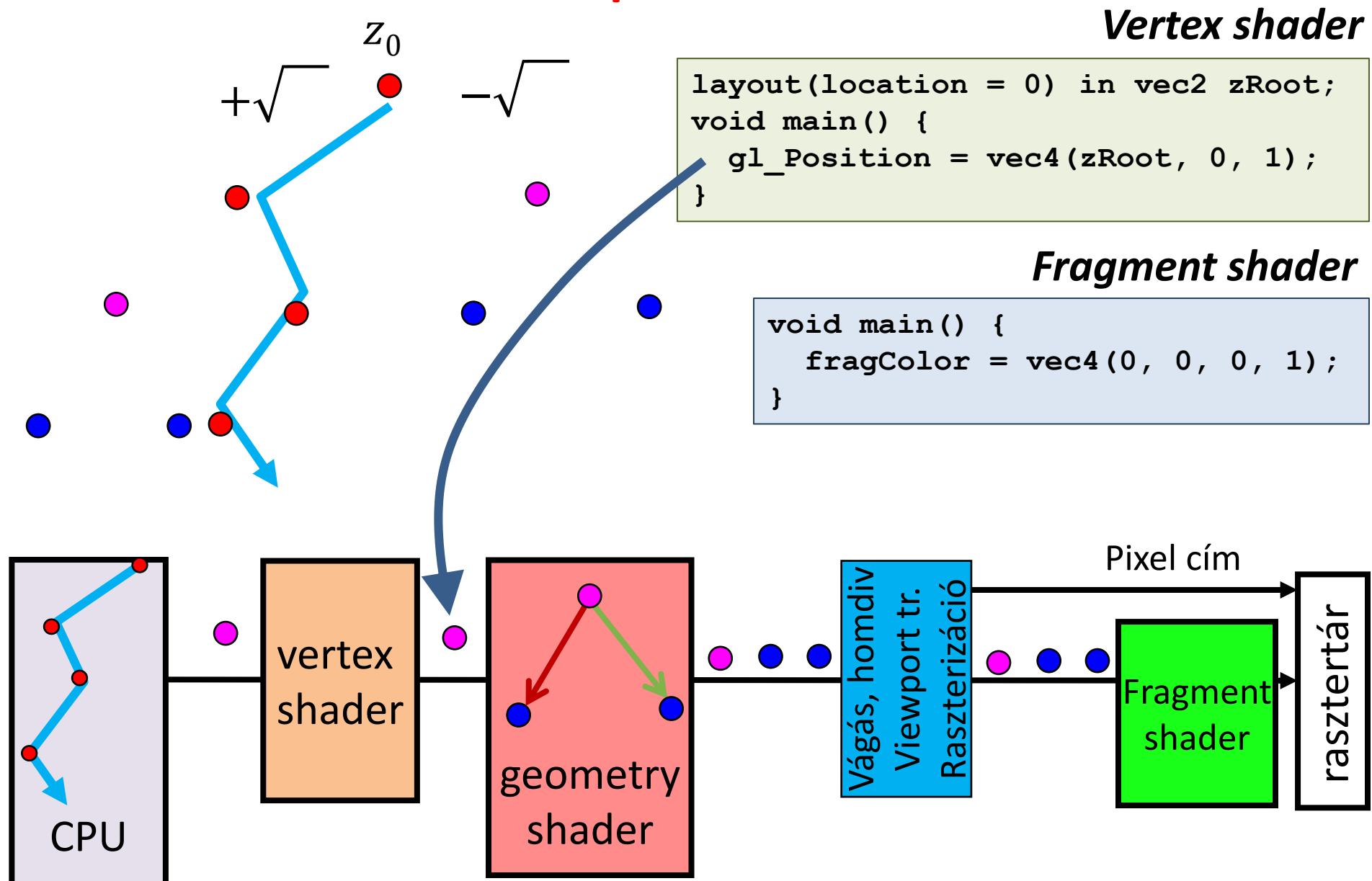


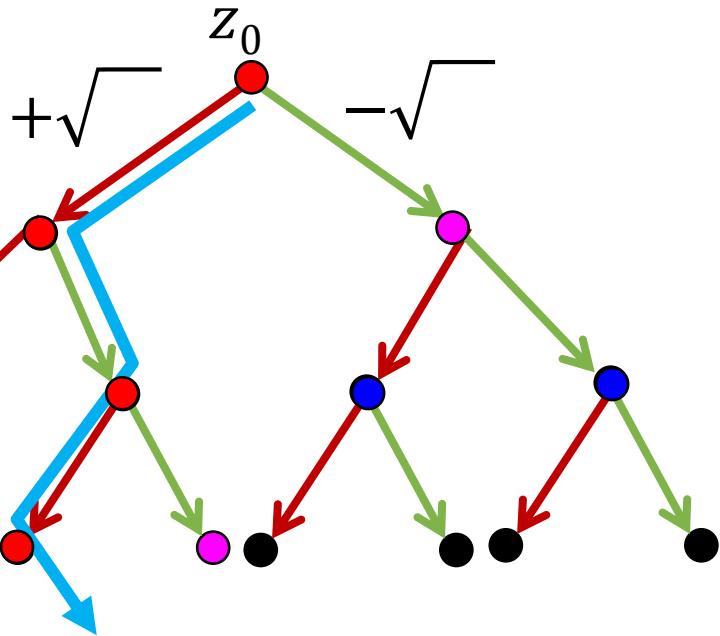
```
Julia( Complex c ) {  
    z = Kezdeti érték választás ←  
    for(n = 0; n < infinity; ++n) {  
        if (InWindow( z ) {  
            pixel = WindowViewport( z );  
            image[pixel] = black;  
        }  
        z =  $\sqrt{z - c}$  ←  
        if (random( ) > 0.5) z = -z;  
    }  
}
```

Kezdeti z érték:  
 $z = z^2 + c$  gyöke

Inverz:  
 $F(z) = z^2 + c$   
 $F^{-1}(z) = \pm\sqrt{z - c}$

# GPU implementáció





# CPU program

Kezdeti z érték:  

$$z^2 = z - c$$
 gyöke

```

vec2 z = vec2(0.5, 0) + sqrtComplex(vec2(0.25 - c.x, -c.y));

for (int p = 0; p < nPackets; p++) {
    vec2 vtx[nSeeds];
    for (int i = 0; i < nSeeds; i++) {
        z = sqrtComplex(z - c) * (rand() & 1 ? 1 : -1);
        vtx[i] = -z;
    }
    glBufferData(GL_ARRAY_BUFFER, sizeof(vtx), vtx,
                GL_DYNAMIC_DRAW);
    glDrawArrays(GL_POINTS, 0, nSeeds);
}

```

# Geometry shader



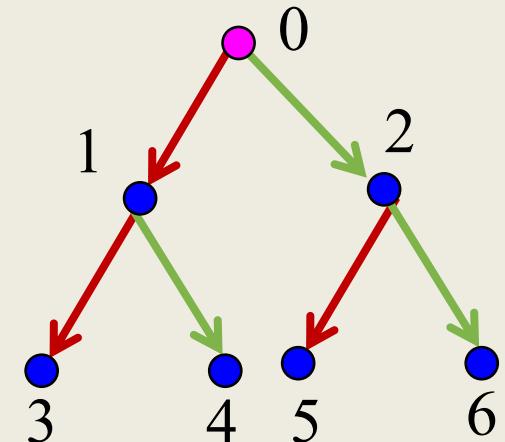
```
uniform vec2 cameraCenter, cameraSize, c;

layout(points) in;
layout(points, max_vertices = 63) out;

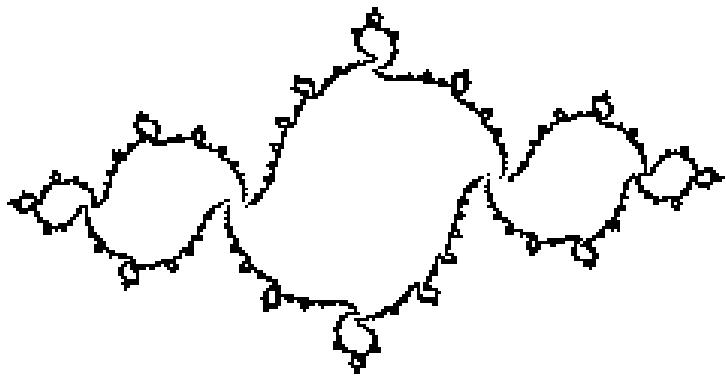
vec2 sqrtComplex(vec2 z) {
    float r = length(z), phi = atan(z.y, z.x);
    return vec2(cos(phi/2), sin(phi/2)) * sqrt(r);
}

void main() {
    vec2 zs[63];
    zs[0] = gl_in[0].gl_Position.xy;
    gl_Position = vec4((zs[0]-cameraCenter)/(cameraSize/2),0,1);
    EmitVertex();

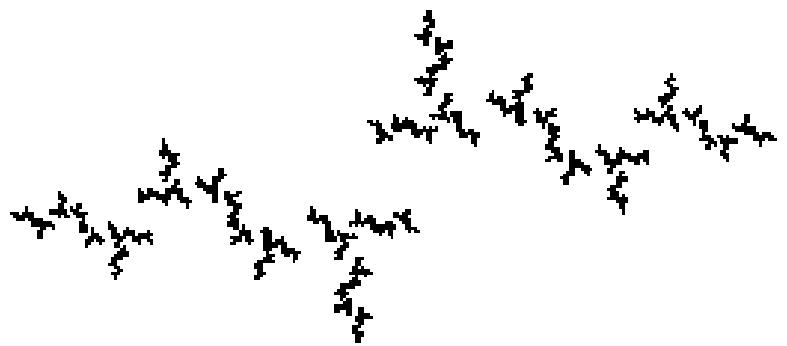
    for(int i = 0; i < 63/2; i++) {
        vec2 z = sqrtComplex(zs[i] - c);
        for(int j = 1; j <= 2; j++) {
            zs[2 * i + j] = z;
            gl_Position = vec4((z-cameraCenter)/(cameraSize/2),0,1);
            EmitVertex();
            z = -z;
        }
    }
    EndPrimitive();
}
```



# Julia halmaz összefüggése



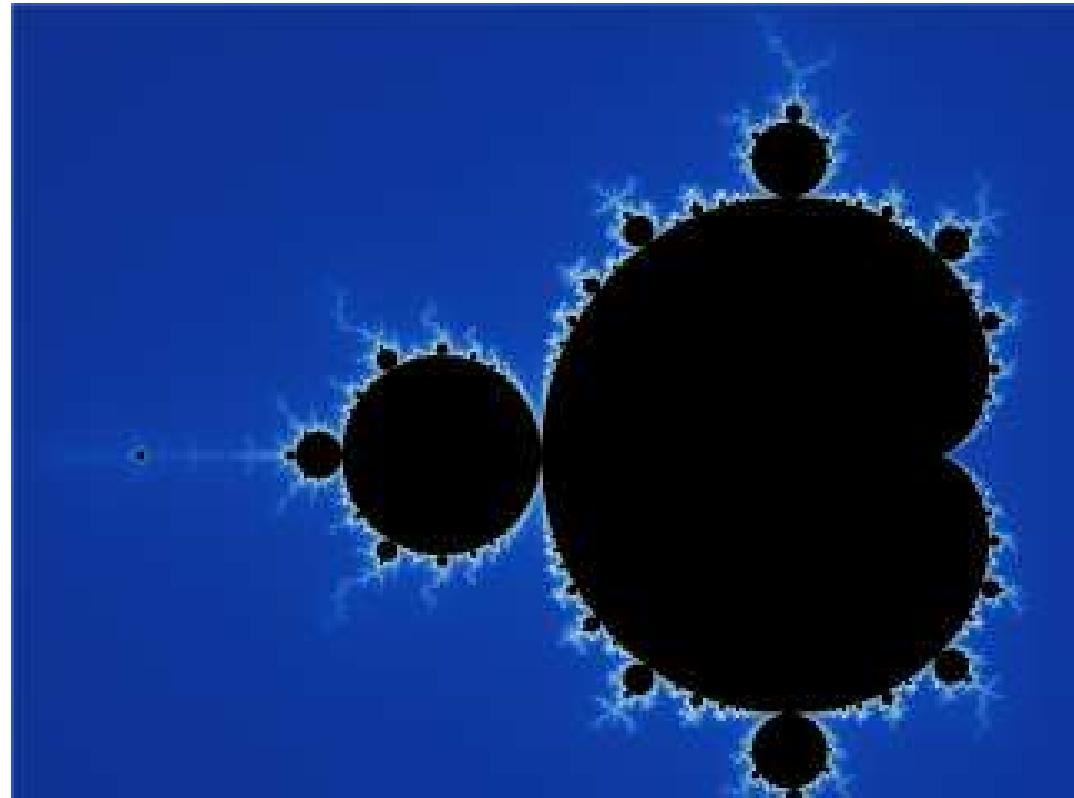
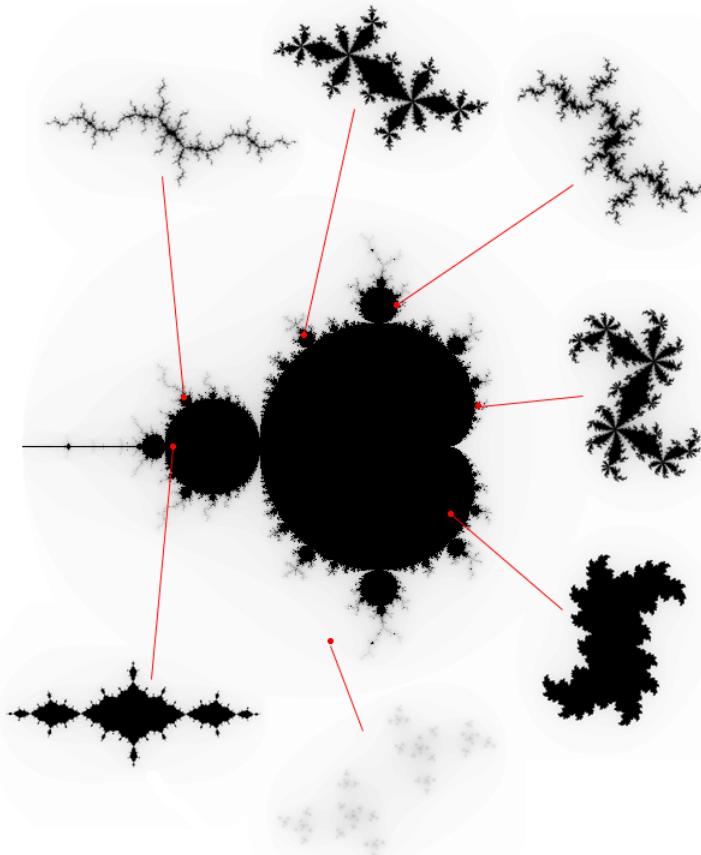
Összefüggő



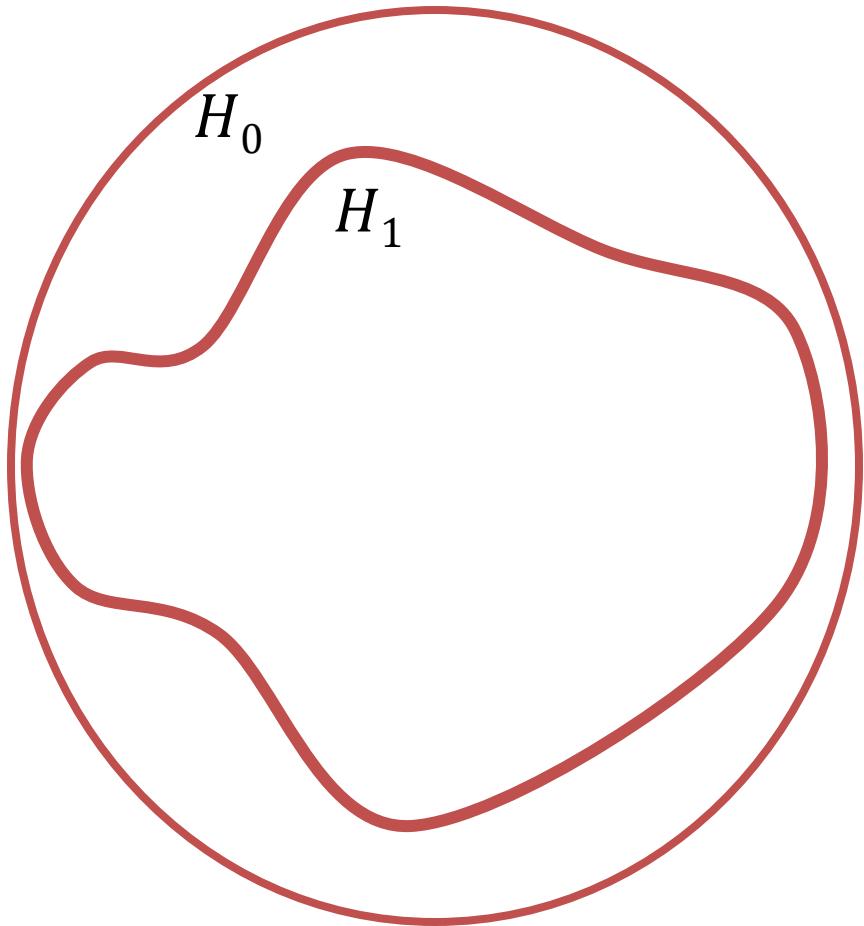
Nem összefüggő,  
Cantor féle halmaz

# (Benoit) Mandelbrot halmaz

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

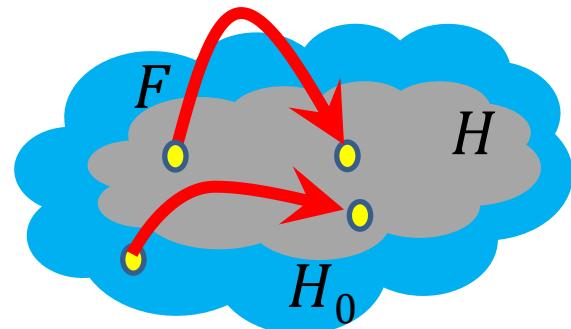


# Julia halmaz összefüggősége



$$F(z) = \pm\sqrt{z - c}$$

$$H = F(H)$$



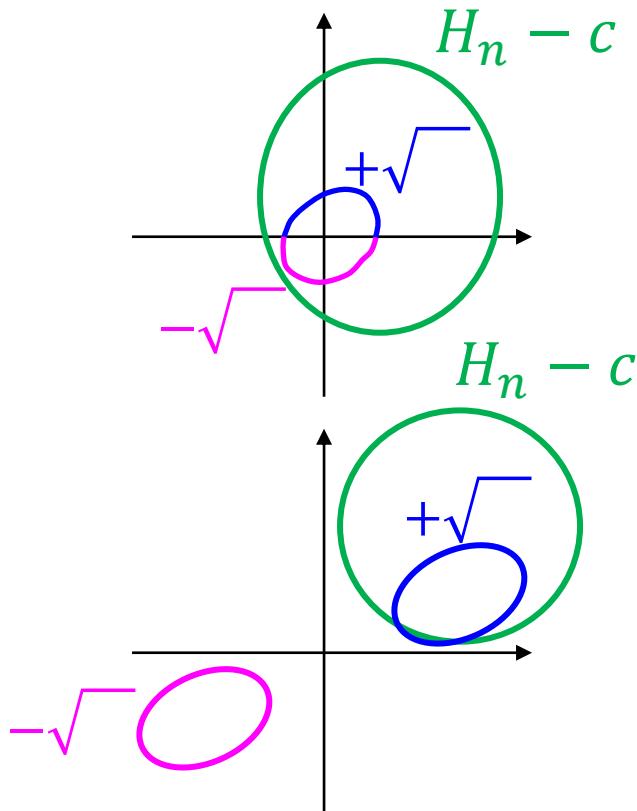
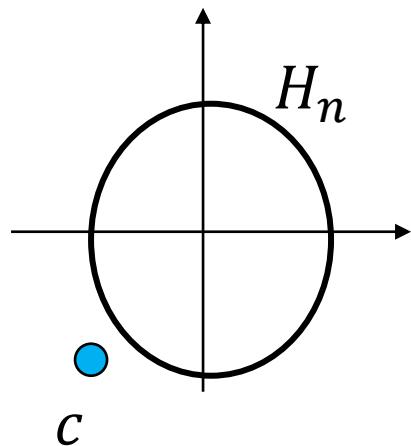
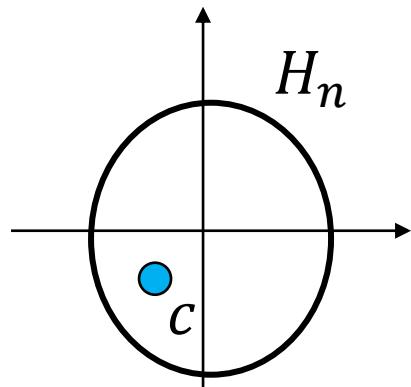
$$H_1 = F(H_0)$$

$$H_2 = F(H_1)$$

$$H_3 = F(H_2)$$

...

# Julia halmaz összefüggősége



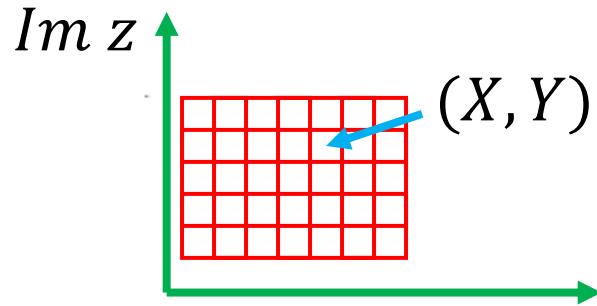
$$H_{n+1} = \pm\sqrt{H_n - c}$$

# (Benoit) Mandelbrot halmaz

Azon  $c$  komplex számok halmaza, amelyekre:

- A  $z \rightarrow z^2 + c$  Julia halmaza összefüggő.
- A  $c$  a Julia halmaz attraktorában vagy konvergens tartományában van.
- A  $z_{n+1} = z_n^2 + c$  iteráció a  $c$ -ből indítva nem divergens.

# Mandelbrot halmaz rajzolás



```
Mandelbrot ( ) {
    for(Y = 0; Y < Ymax; ++Y) {
        for(X = 0; X < Xmax; ++X) {
            Complex c = ViewportWindow(X, Y);
            Complex z = c;
            for(n = 0; n < infinity; ++n) z = z2 + c
            image[Y][X] = (|z| < infinity) ? black : white;
        }
    }
}
```

$|z| < 2$

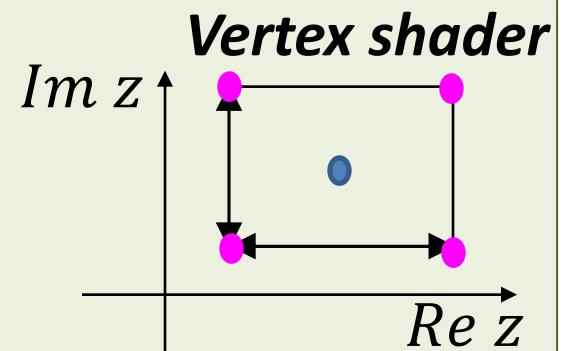


# GPU implementáció

```
float cVtx[] = { -1, -1, 1, -1, 1, 1, -1, 1 };
glBufferData(GL_ARRAY_BUFFER, sizeof(cVtx), cVtx, GL_STATIC_DRAW);
...
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

*CPU program*

```
uniform vec2 cameraCenter, cameraSize;
layout(location = 0) in vec2 cVertex;
out vec2 c;
void main() {
    gl_Position = vec4(cVertex, 0, 1);
    c = cVertex * cameraSize/2 + cameraCenter;
}
```

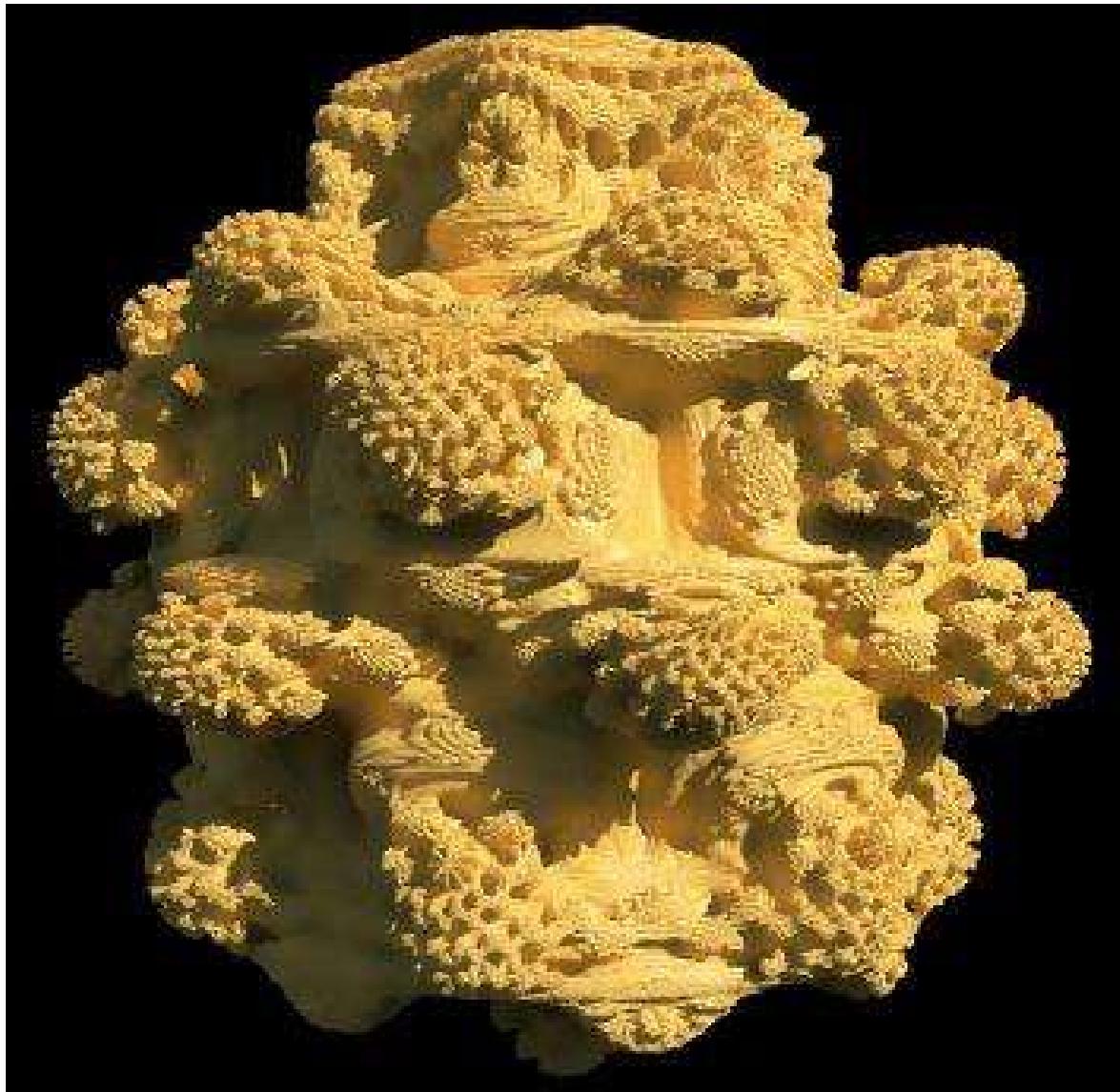


```
in vec2 c;
out vec4 fragCol;
const int nIteration = 1000;
void main() {
    vec2 z = c;
    for(int i = 0; i < nIteration; i++) {
        z = vec2(z.x * z.x - z.y * z.y, 2 * z.x * z.y) + c;
        if (dot(z, z) > 4) break;
    }
    fragCol = (i == nIteration) ? vec4(0, 0, 0, 1) : vec4(1, 1, 1, 1);
}
```

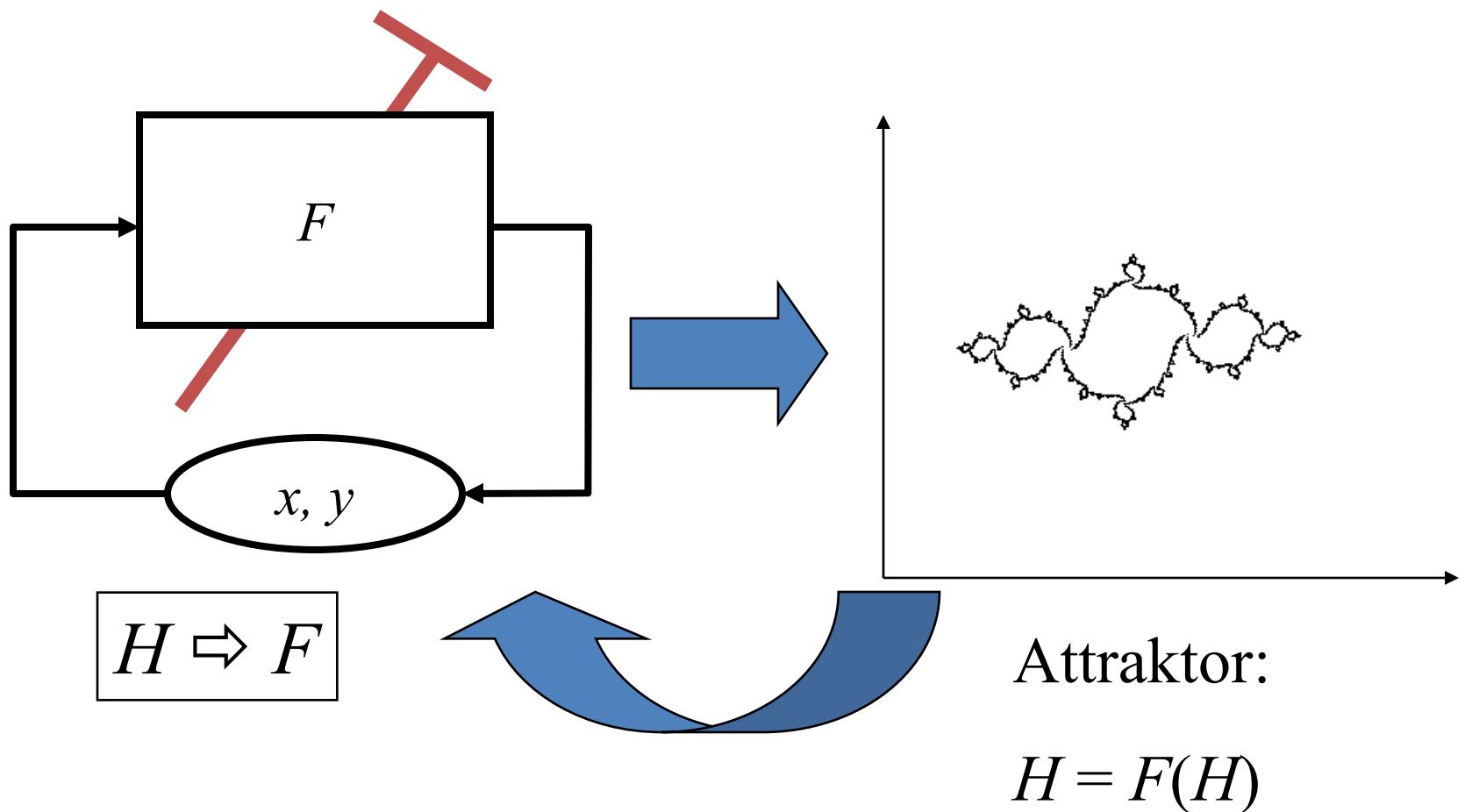
*Fragment shader*

$$z = z^2 + c$$

# Mandelbulb



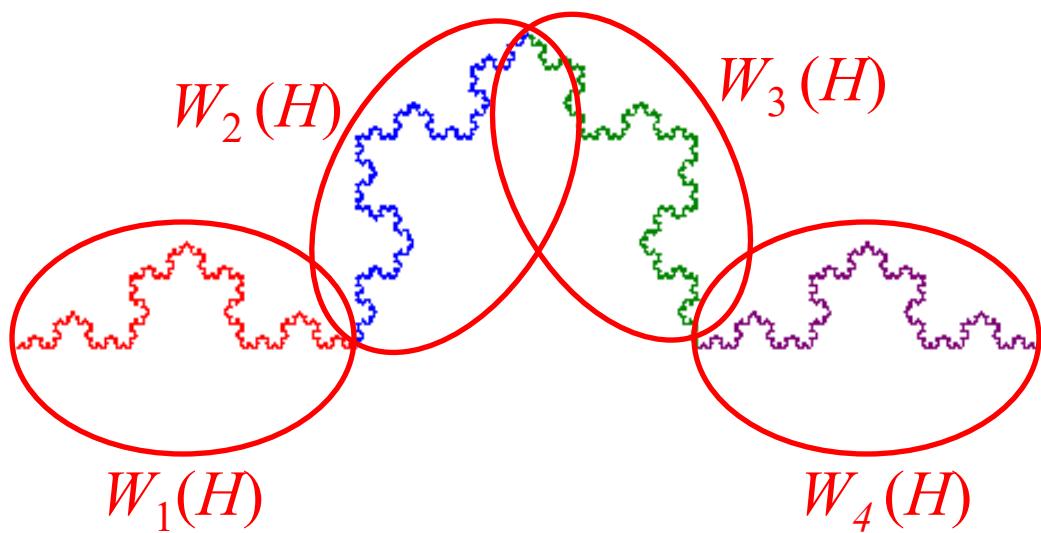
# Inverz feladat: IFS modellezés



$F$ : szabadon vezérelhető, legyen stabil attraktora

# Melyik függvény attraktora a Koch görbe?

Attraktor:  $H = F(H) = W_1(H) \cup W_2(H) \cup W_3(H) \cup W_4(H)$



$$W_k(x,y) = [x,y] \cdot A_k + q_k$$

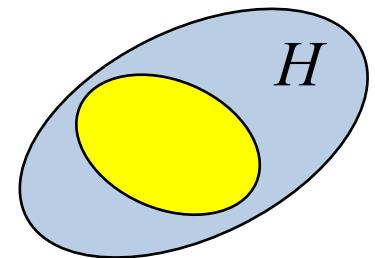
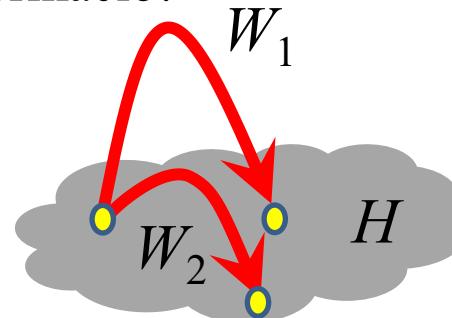
# $F$ : többértékű lineáris leképzés

Nem feltétlenül hasonlósági transzformáció!

Nem csak önhasonló objektumok.

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

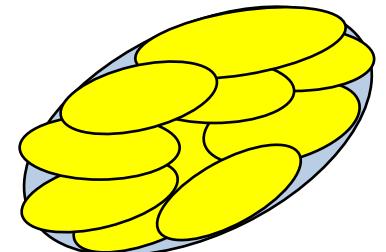
$$W_k(x,y) = [x,y] \cdot A_k + q_k$$



Stabilitás = kontrakció  
 $|A$  sajátértékei| < 1

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

$$H = F(H)$$



Kollázs:

lefedés a kicsinyített változatokkal  
Nem feltétlenül diszkjunkt

# IFS rajzolás: iterációs algoritmus

## IFSDraw()

Legyen  $[x,y] = [x,y] A_1 + q_1$  megoldása a kezdő  $[x,y]$   
**FOR**  $i = 0$  TO “infinity” **DO**

**IF** InWindow( $x, y$ )

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

        Write( $X, Y, color$ );

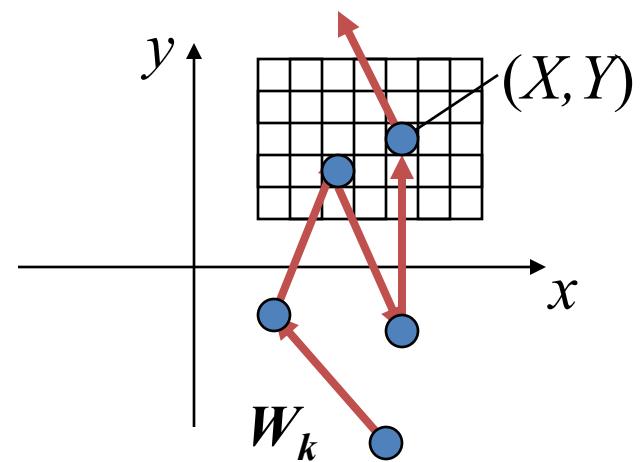
**ENDIF**

    Válassz  $k$ -t  $p_k$  valószínűsséggel

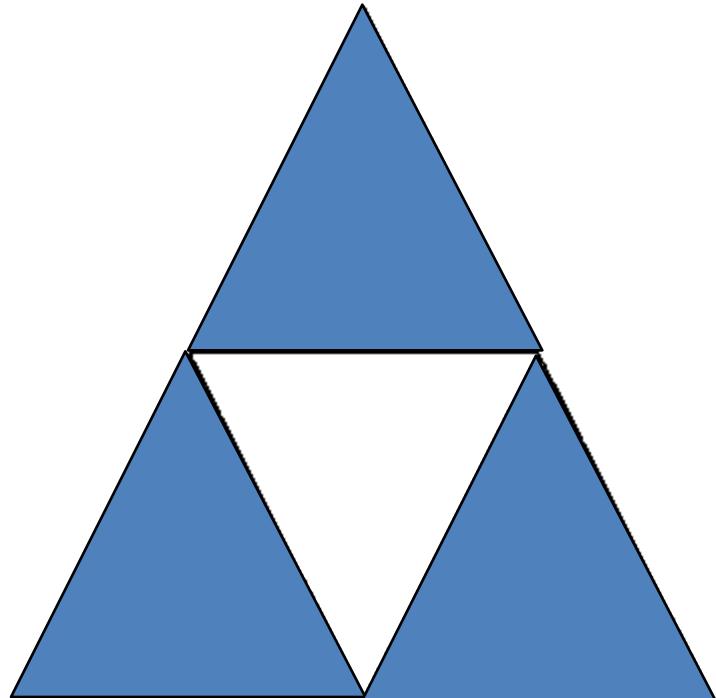
$[x,y] = [x,y] A_k + q_k$

**ENDFOR**

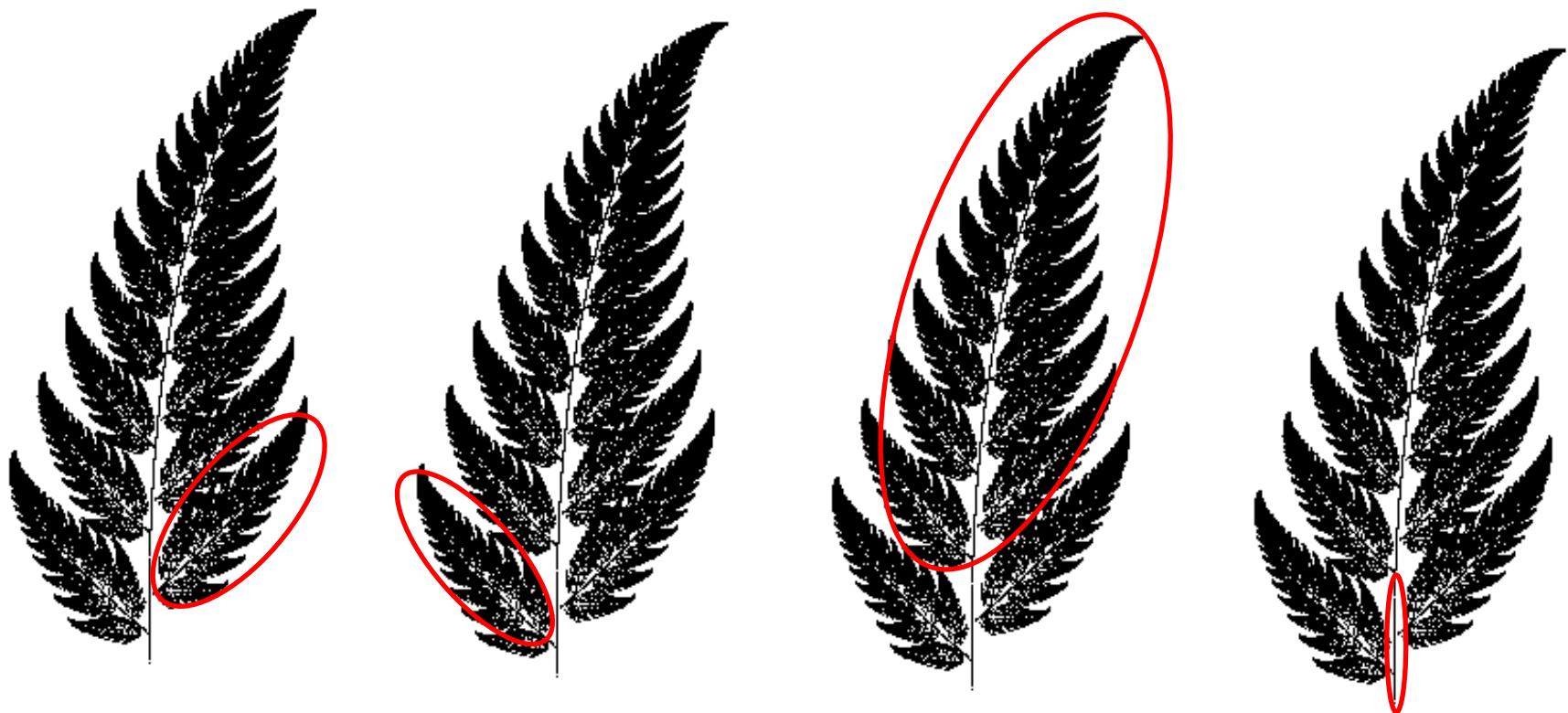
**END**



# Egyszerű IFS-ek



# IFS modellezés



# IFS képek

