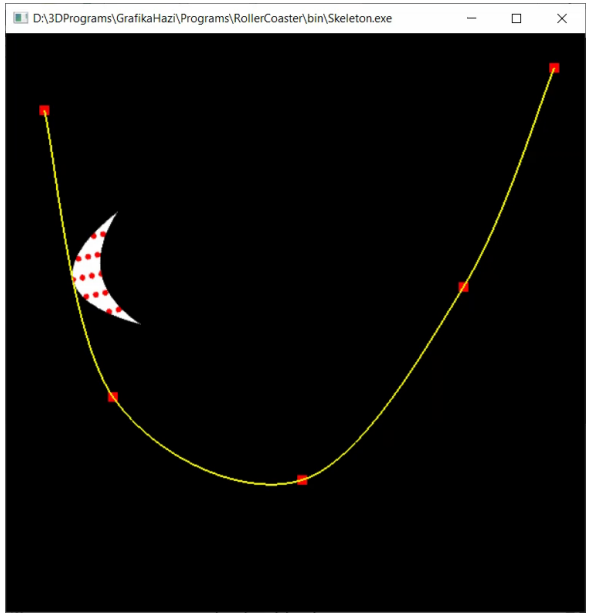
"Photographers don't take pictures. They create images."

Mark Denman

2D képszintézis

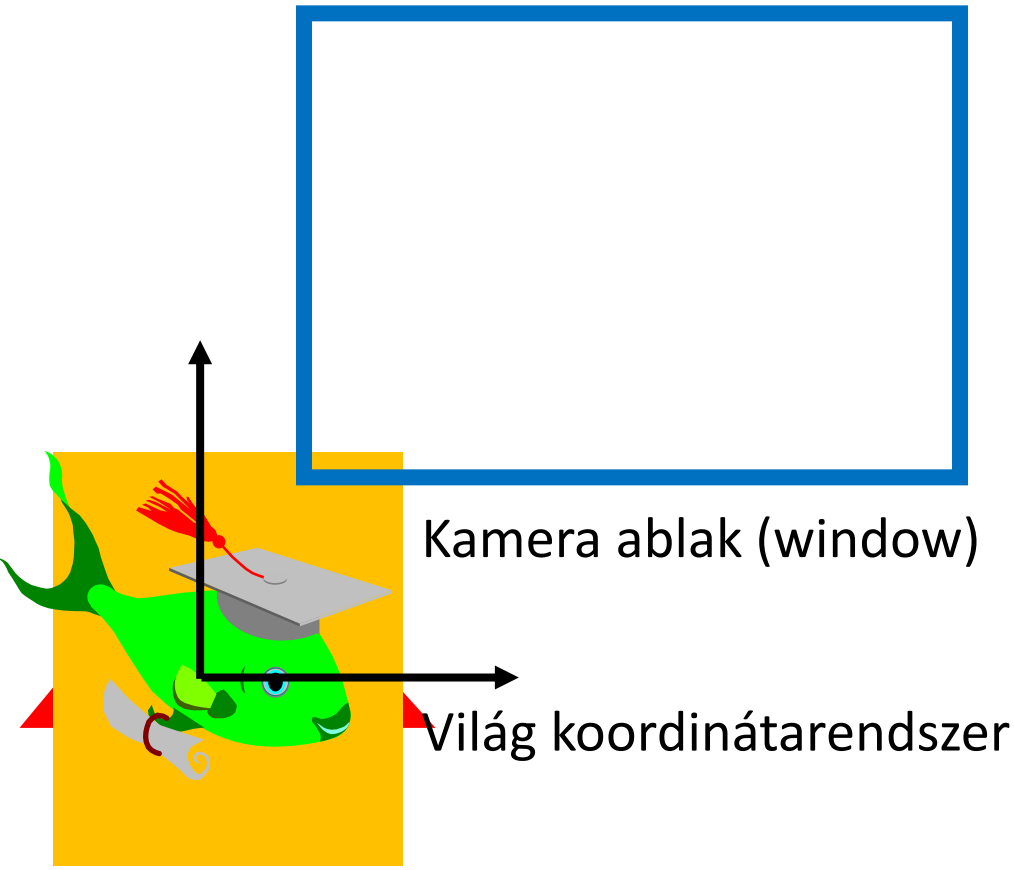
Szirmay-Kalos László

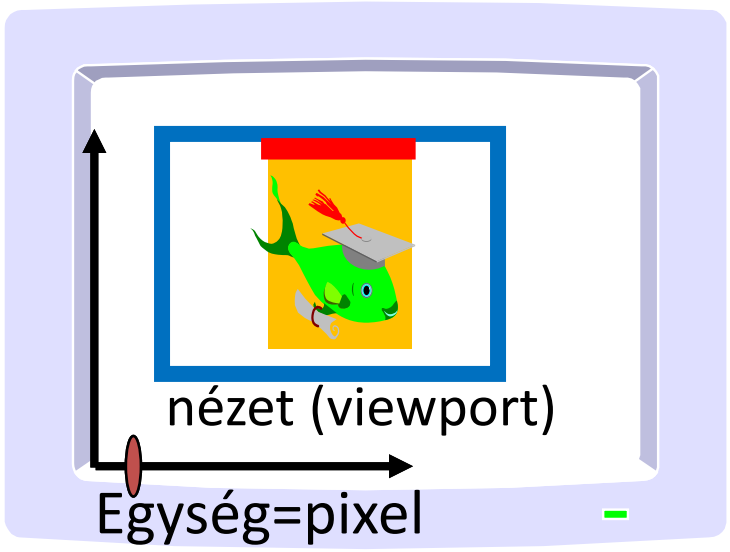


2D képszintézis

Modell

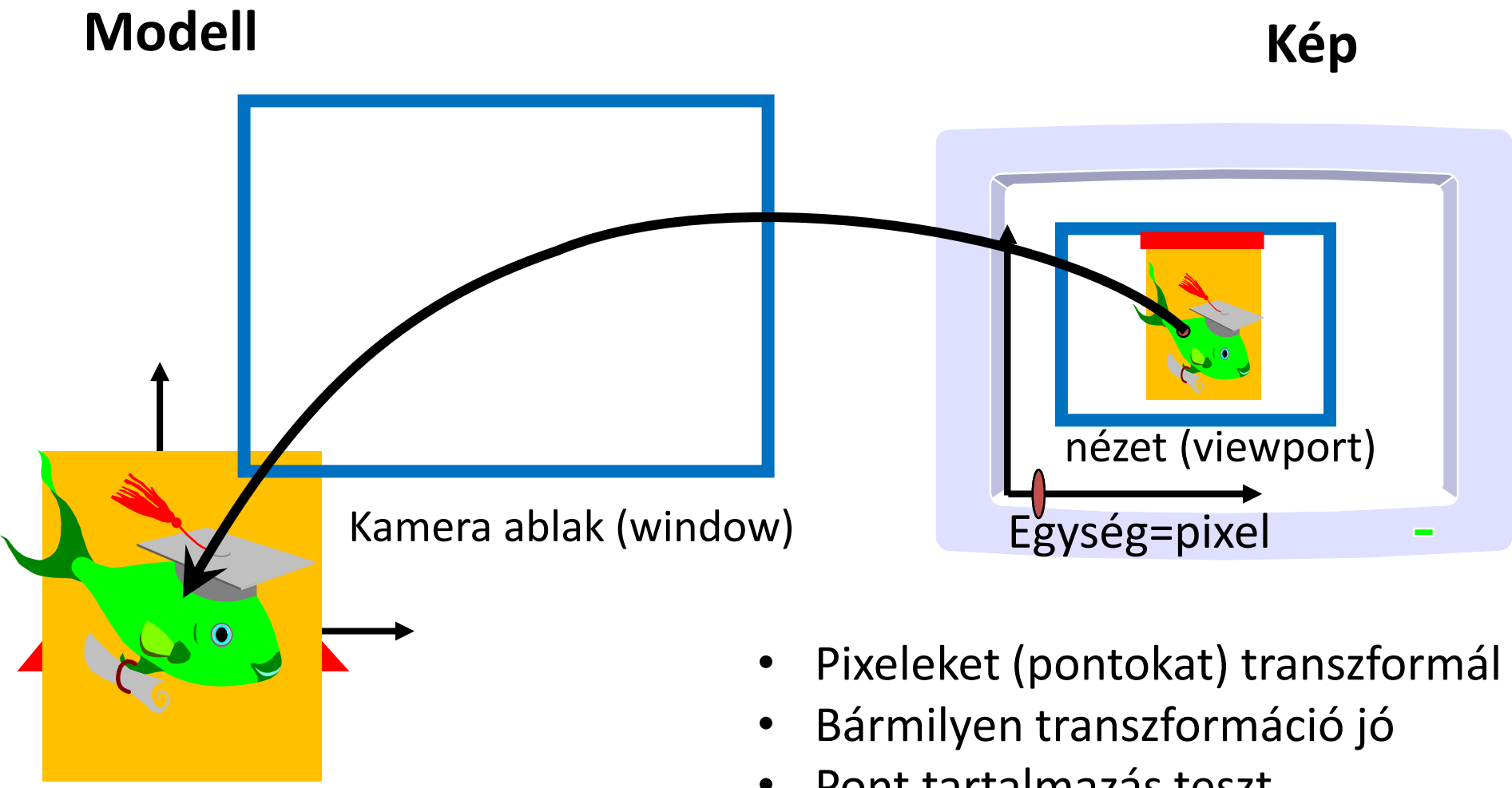
Kép





Saját színnel rajzolás

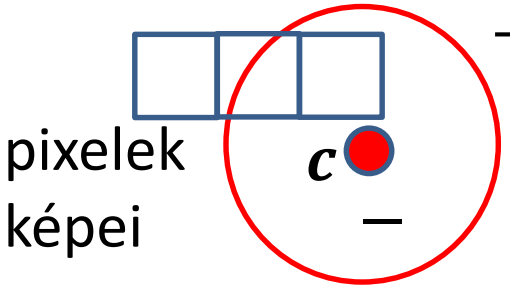
Pixel vezérelt 2D képszintézis



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

Pixel vezérelt megközelítés: Tartalmazás (objektum, pont)

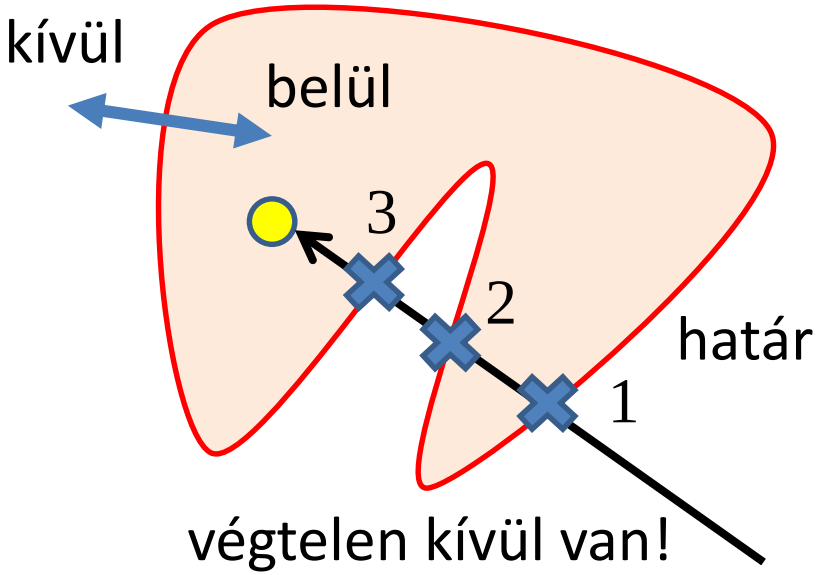
- Határ implicit görbe:

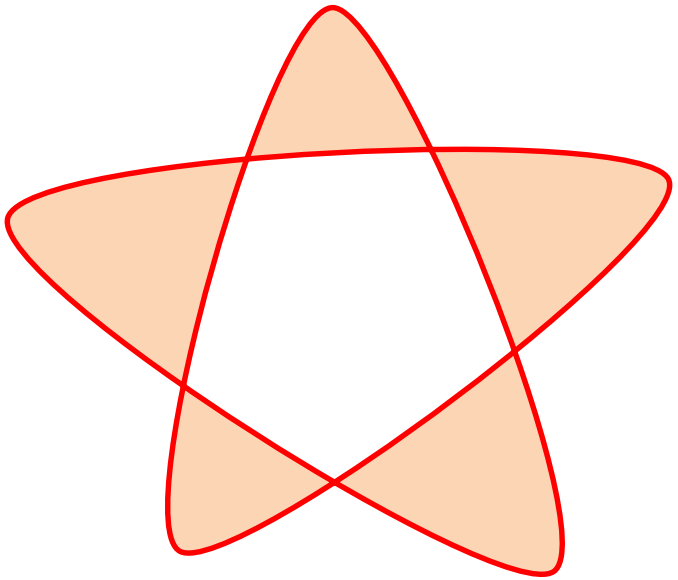


$$f(\boldsymbol{r}) = |\boldsymbol{r} - \boldsymbol{c}|^2 - R^2$$

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

- Határ parametrikus görbe:





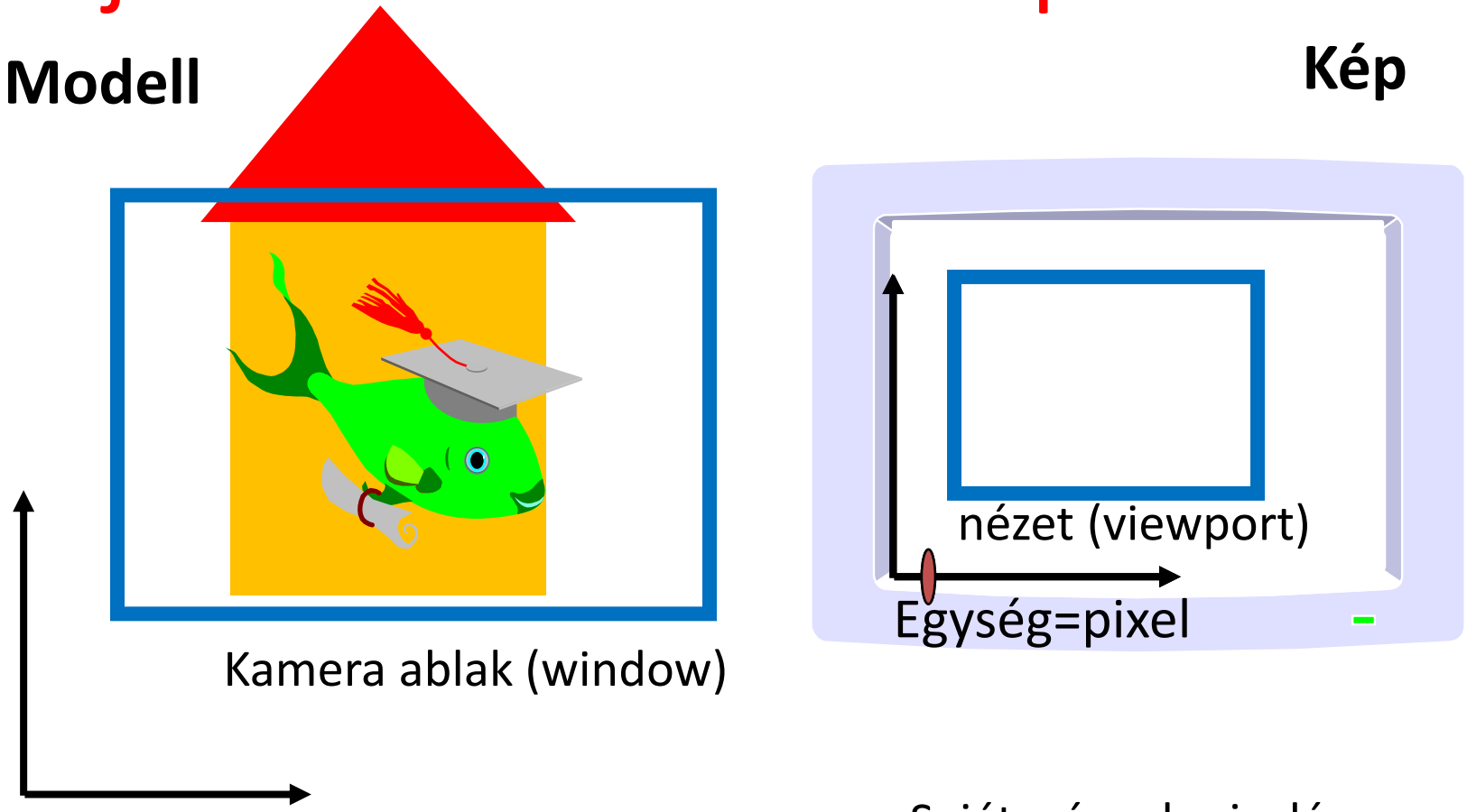
Pixel vezérelt rendering

```
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

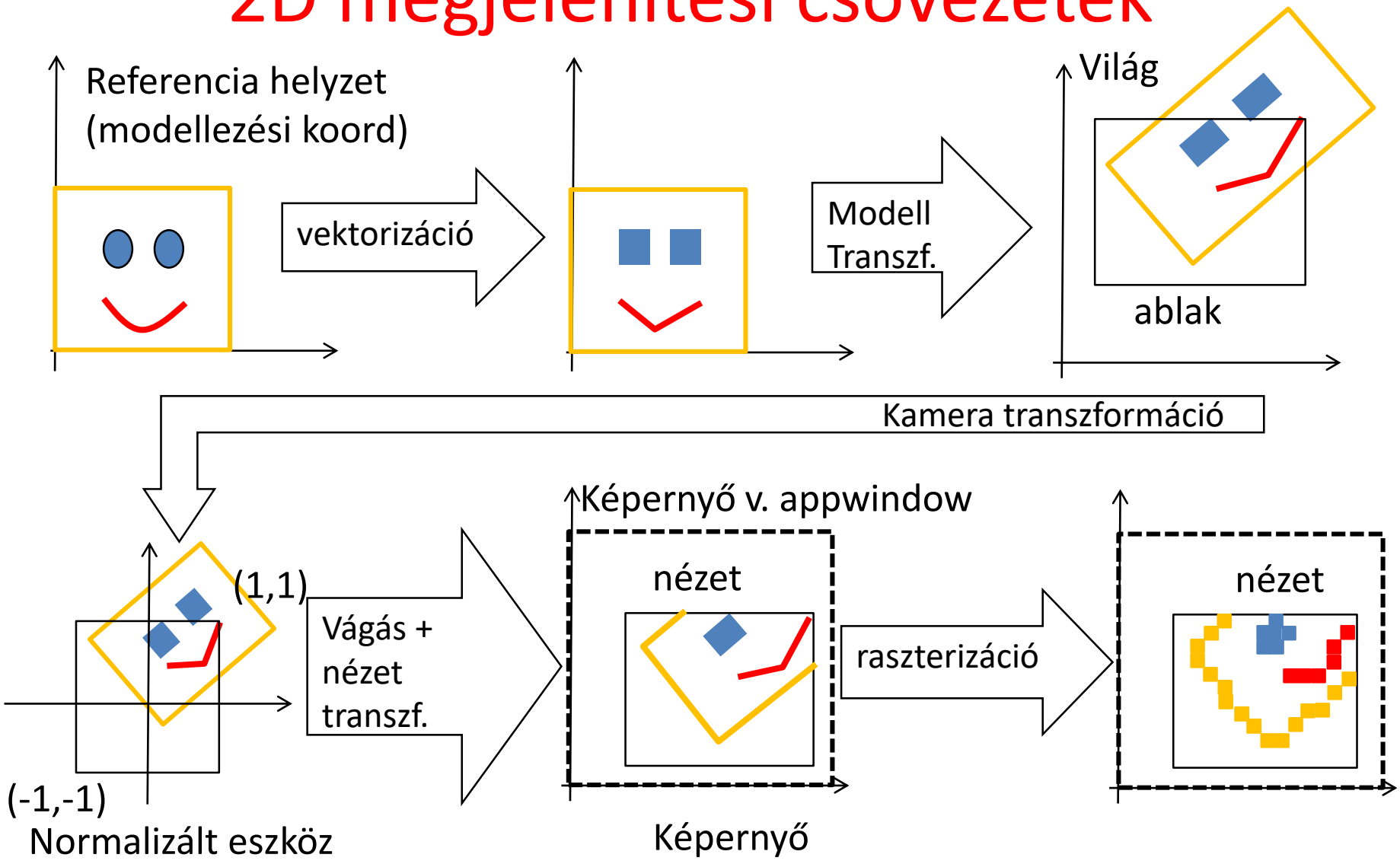
```
class Scene {                       // virtual world
   list<Object *> objs;             // objects with decreasing priority
   Object *picked = nullptr;        // selected for operation
public:
   void Add(Object * o) { objects.push_front(o); picked = o; }
   void Pick(int pX, int pY) { // pX, pY: 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



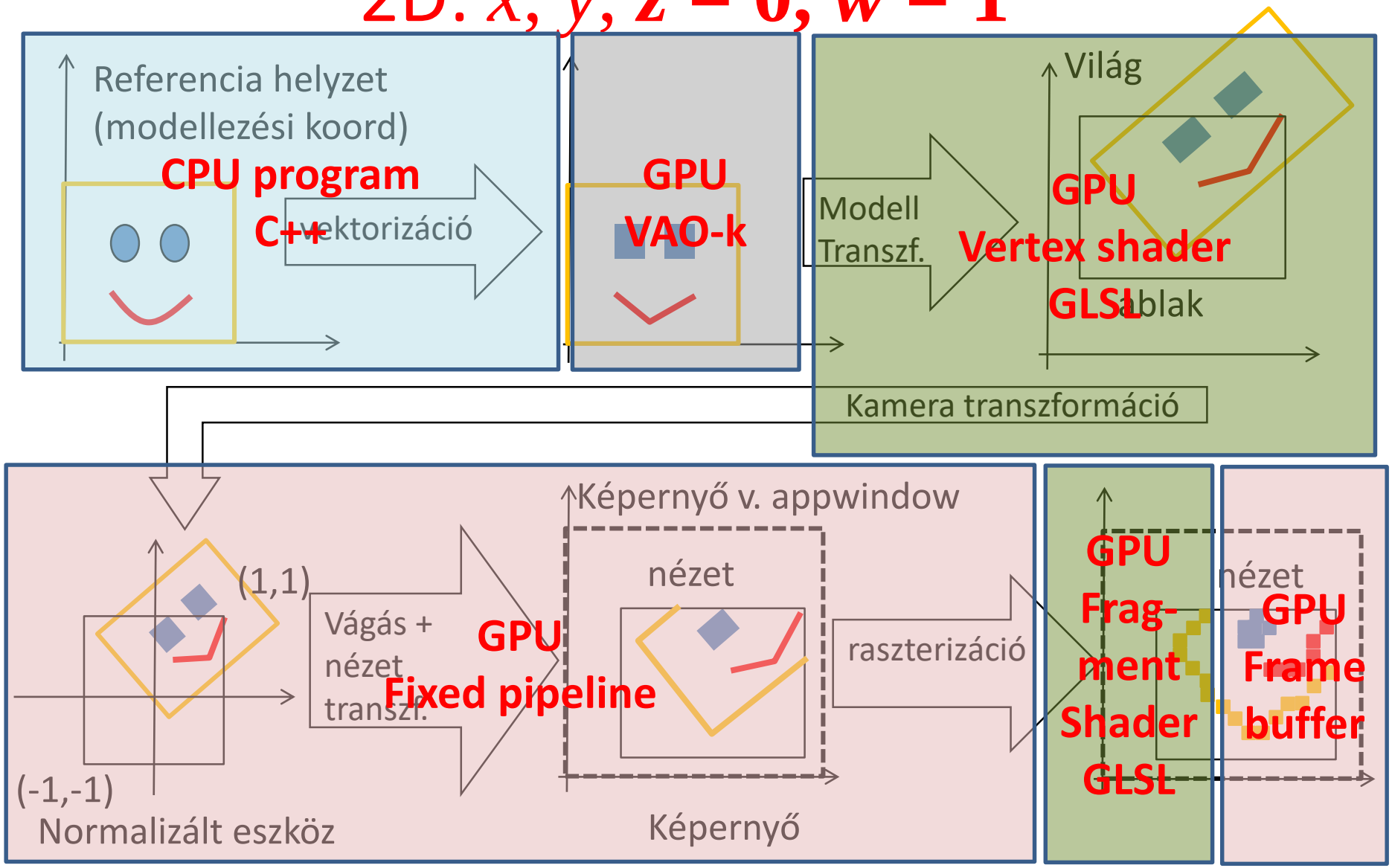
Saját színnel rajzolás
a kis prioritásúakkal kezdve

Objektum vezérelt megközelítés: 2D megjelenítési csővezeték



GPU megjelenítési csővezeték

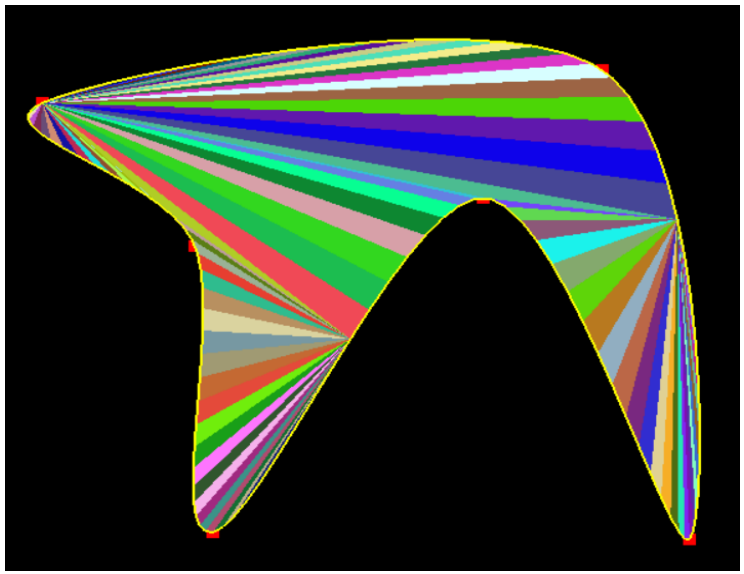
2D: $x, y, z = 0, w = 1$



"μή μου τοὺς κύκλους τάραττε."
Ἀρχιμήδης

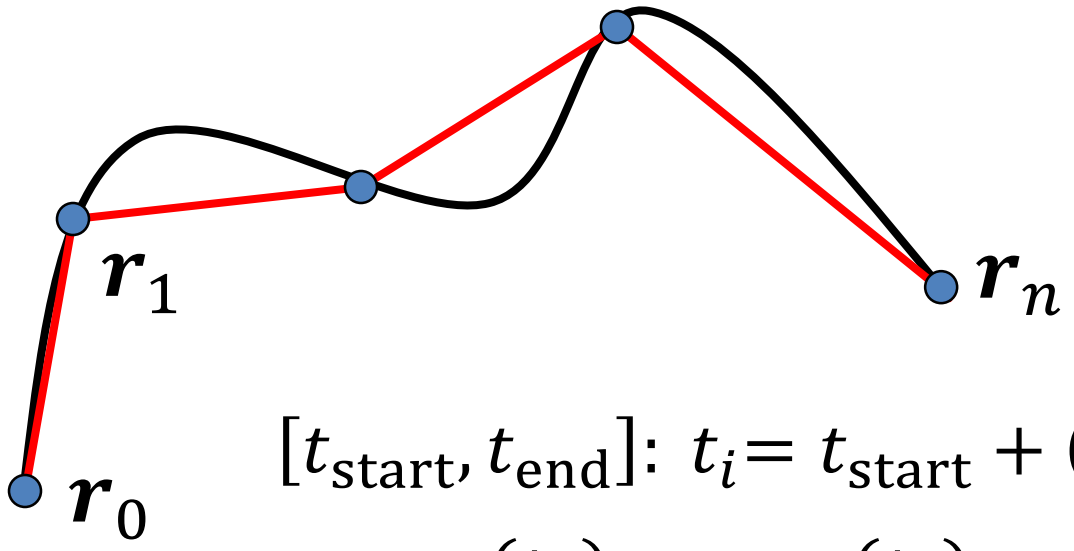
2D képszintézis
2. Vektorizáció és háromszögesítés

Szirmay-Kalos László



Vektorizáció (CPU)

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



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

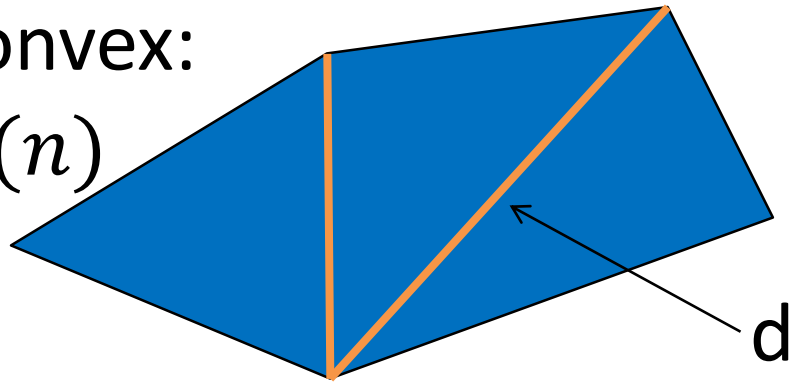
$\boldsymbol{r}_0 = \boldsymbol{r}(t_0), \boldsymbol{r}_1 = \boldsymbol{r}(t_1), \dots, \boldsymbol{r}_n = \boldsymbol{r}(t_n)$

		Hw érdekében
Görbe	→ nyílt töröttvonal	→ szakaszok
Terület határa	→ zárt töröttvonal = poligon	→ háromszögek

Poligon háromszögekre bontása

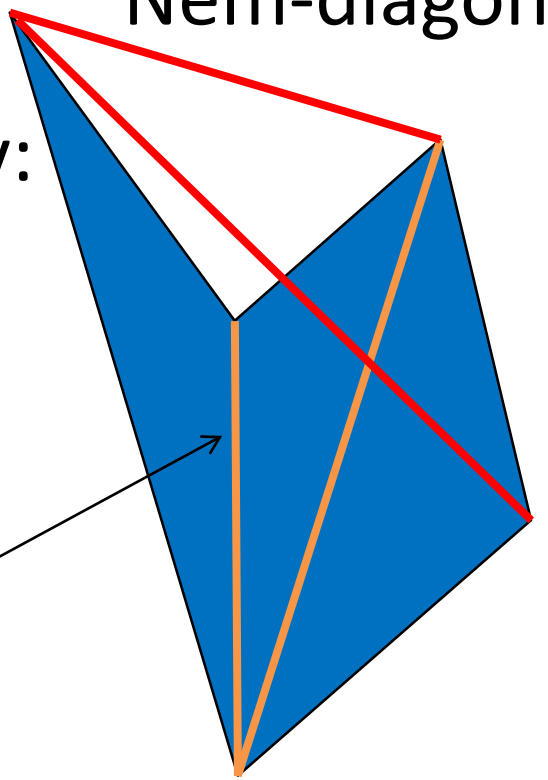
Diagonál mentén értelmes vágni, mert az csökkenti a csúcspontok számát!

Konvex:
$O(n)$



Nem-diagonál

Konkáv:
$O(n^4)$



Tétel: Minden 4+ csúcsú **<u>egyszerű</u>** sokszögnek van diagonálja, azaz mindegyik felbontható diagonálok mentén.

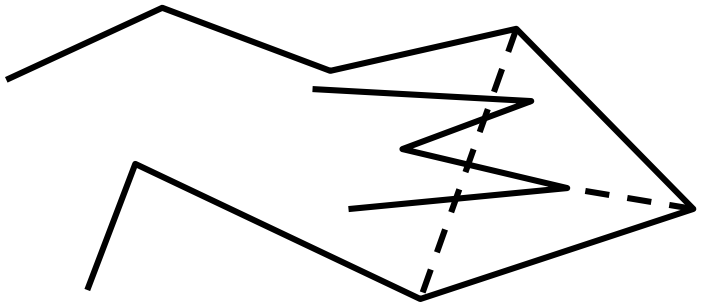
Dia
me
szá

Ko
$O($

Egyszerű sokszög

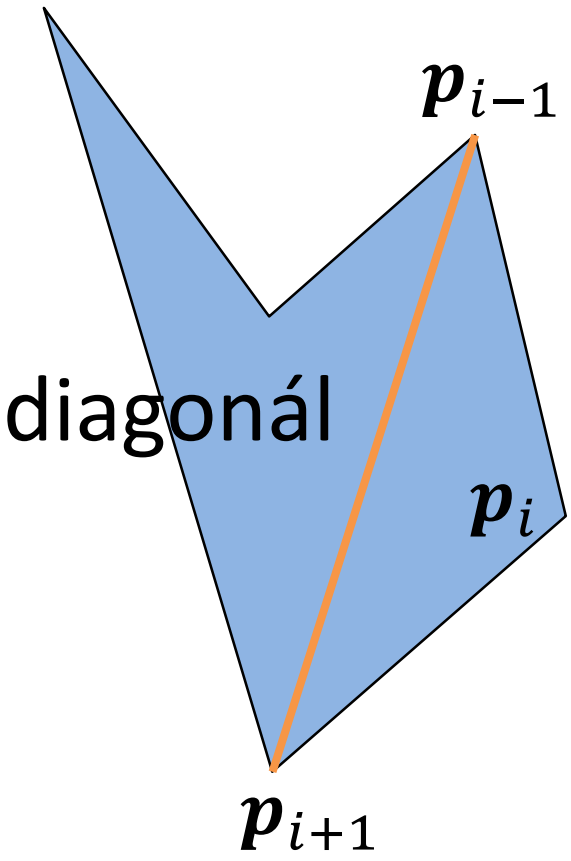
Nem egyszerű sokszög

Tétel: Minden 4+ csúcsú **<u>egyszerű</u>** sokszögnek van diagonálja, azaz mindegyik felbontható diagonálok mentén.



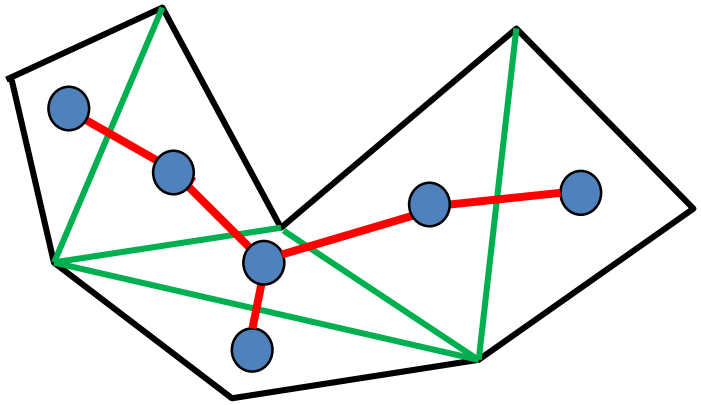
Konvex csúcs

Fül



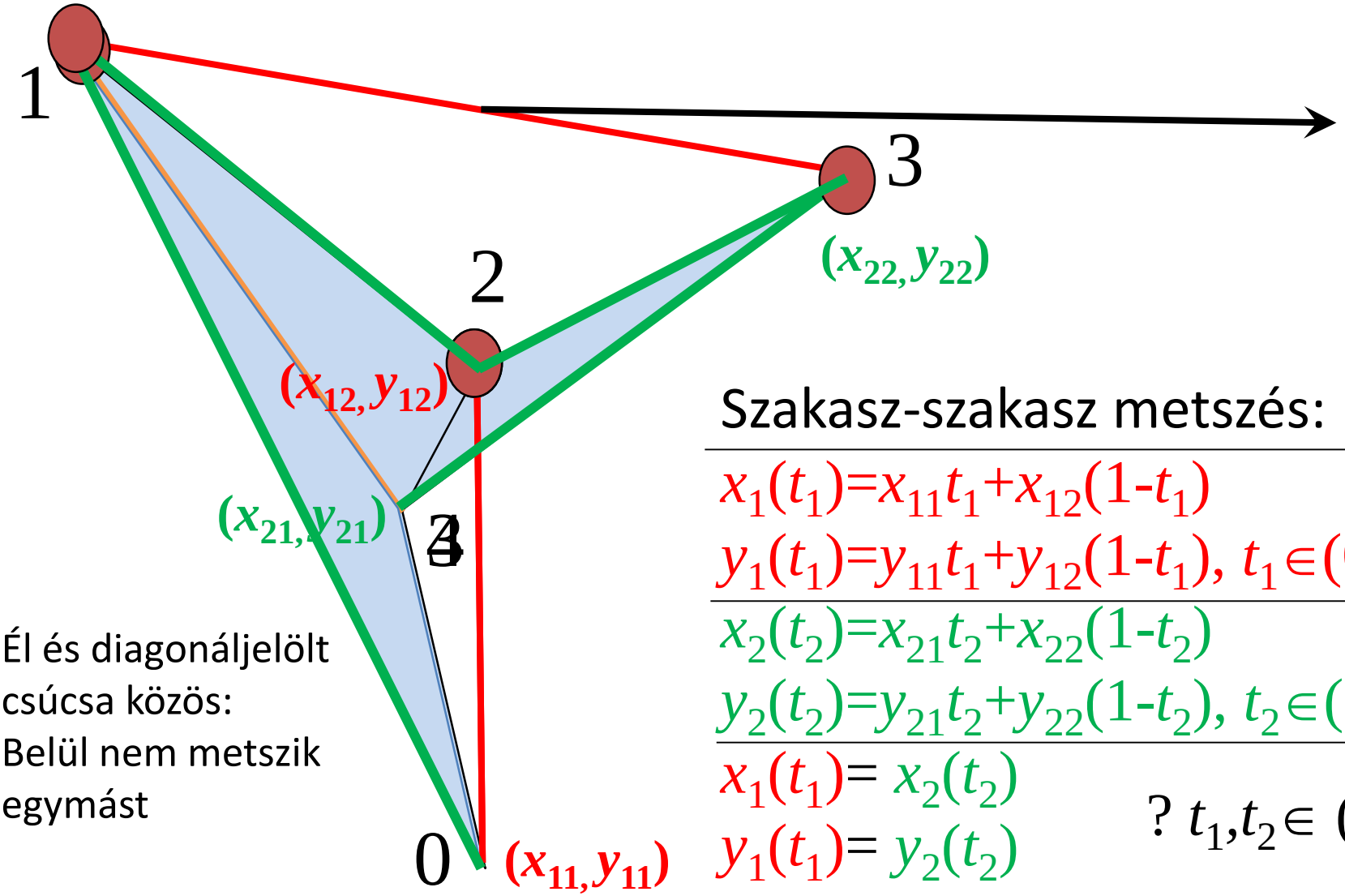
- p_i fül, ha $p_{i-1} \leftrightarrow p_{i+1}$ diagonál
- Fül levágható!
- **Fülvágás**: keress fület és nyissz!
- $O(n^3)$

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



„Minden fának van legalább két levele.”

Fülvágó algoritmus: $O(n^3)$



Él és diagonáljelölt csúcsa közös:
Belül nem metszik egymást

Szakasz-szakasz metszés:

$x_1(t_1)=x_{11}t_1+x_{12}(1-t_1)$

$y_1(t_1)=y_{11}t_1+y_{12}(1-t_1),\ t_1\in(0,1)$

$x_2(t_2)=x_{21}t_2+x_{22}(1-t_2)$

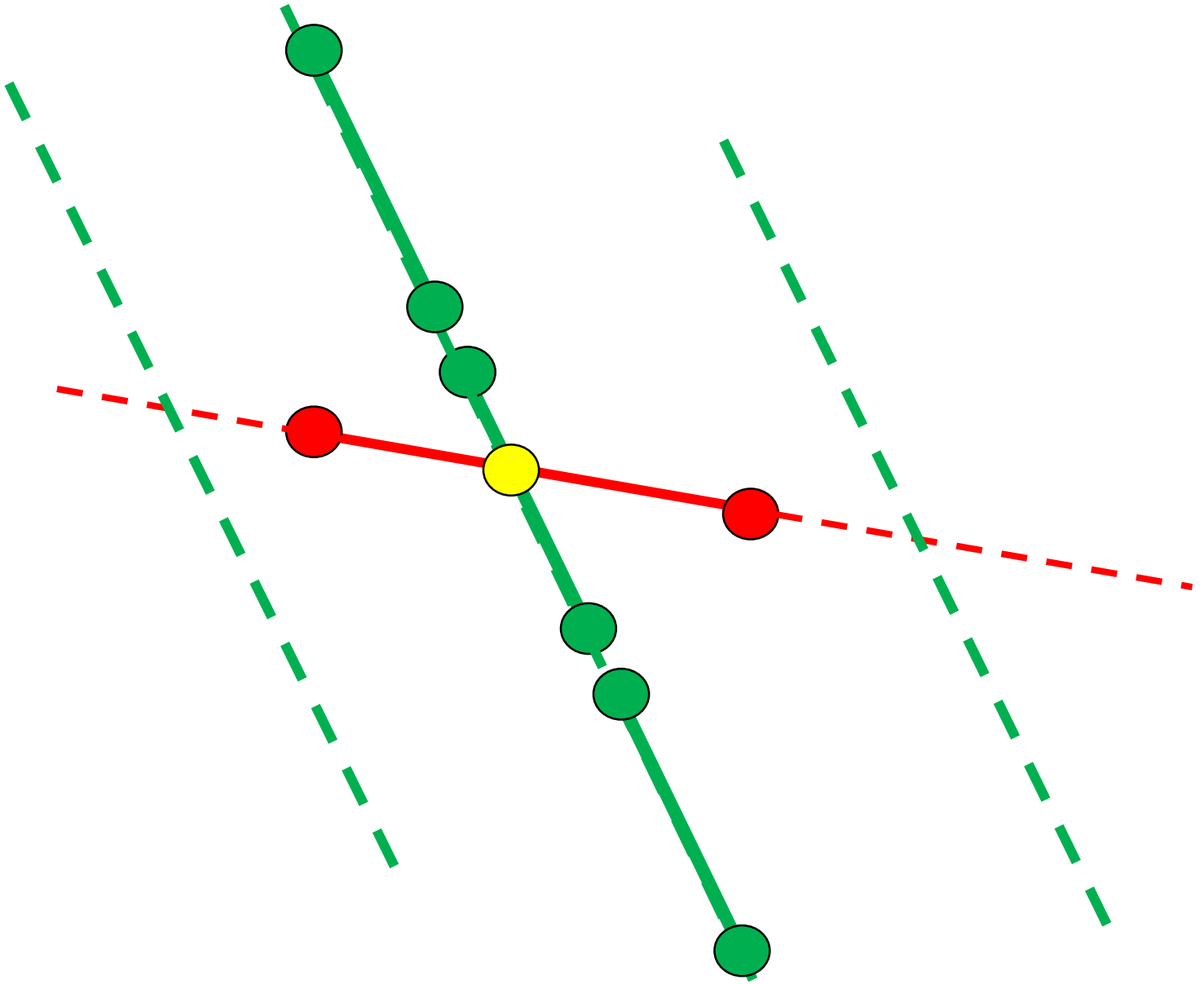
$y_2(t_2)=y_{21}t_2+y_{22}(1-t_2),\ t_2\in(0,1)$

$x_1(t_1)= x_2(t_2)$

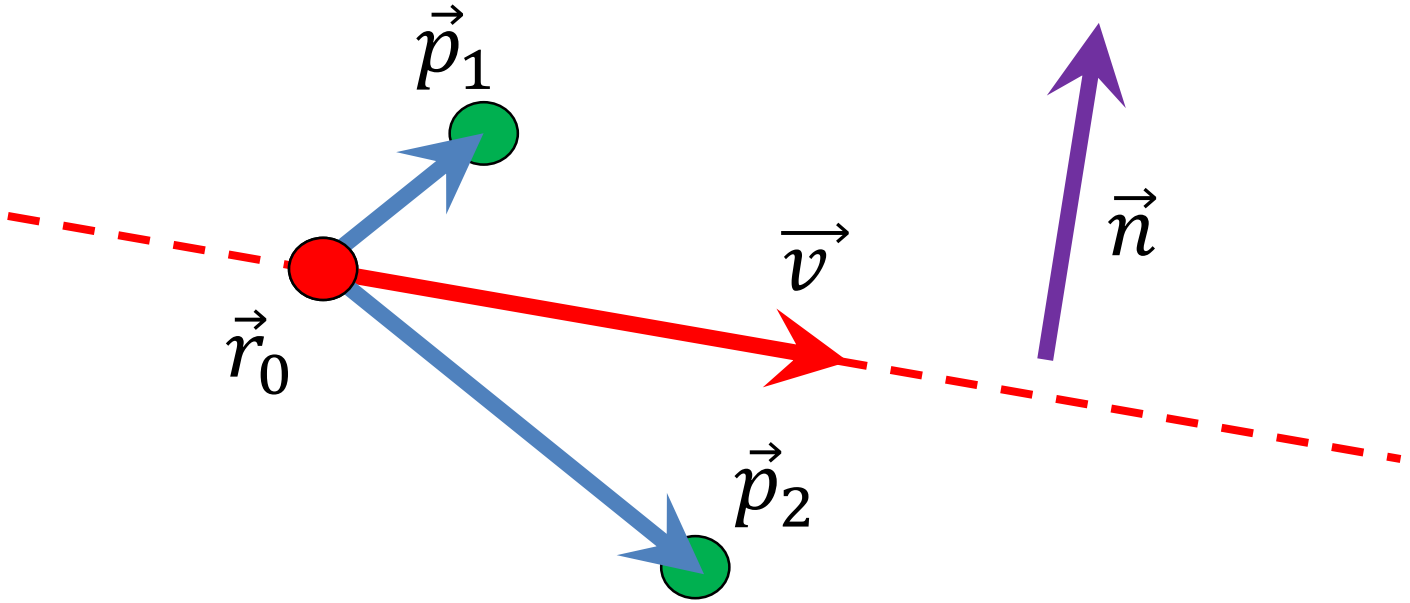
$y_1(t_1)= y_2(t_2)$

? $t_1,t_2\in(0,1)$

Szakasz-szakasz metszés egyszerűbben



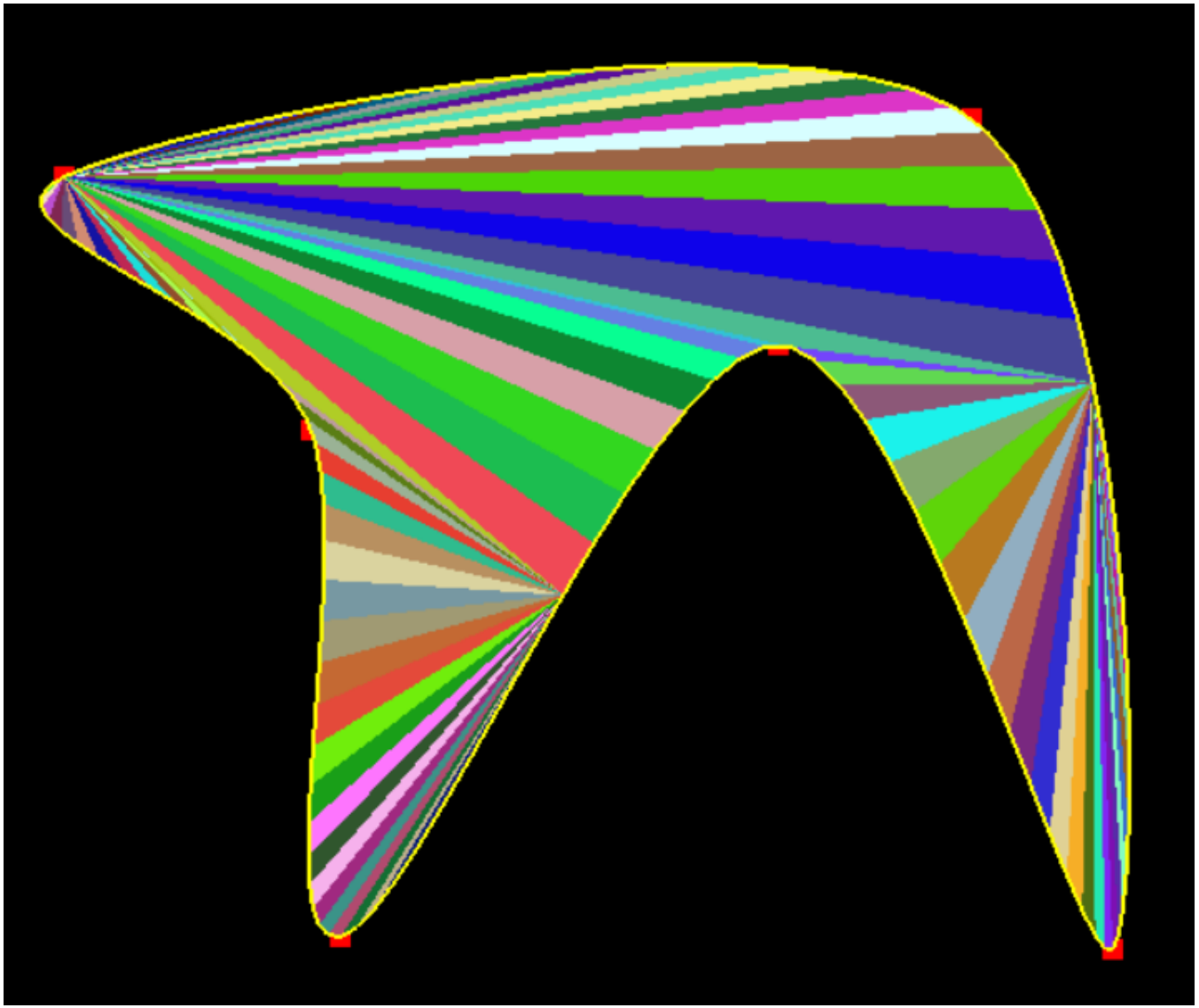
Két pont az egyenes két oldalán van



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

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

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:

$$[x_{\text{world}}, y_{\text{world}}, z_{\text{world}}, w_{\text{world}}] = [x_{\text{model}}, y_{\text{model}}, z_{\text{model}}, 1] \cdot \boldsymbol{T}_{4\times4}$$

- Speciális eset: 2D affin modellezési transzformáció:

$$\boldsymbol{T}_{4\times4} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & * & 0 \\ v_x & v_y & 0 & 1 \end{bmatrix}$$

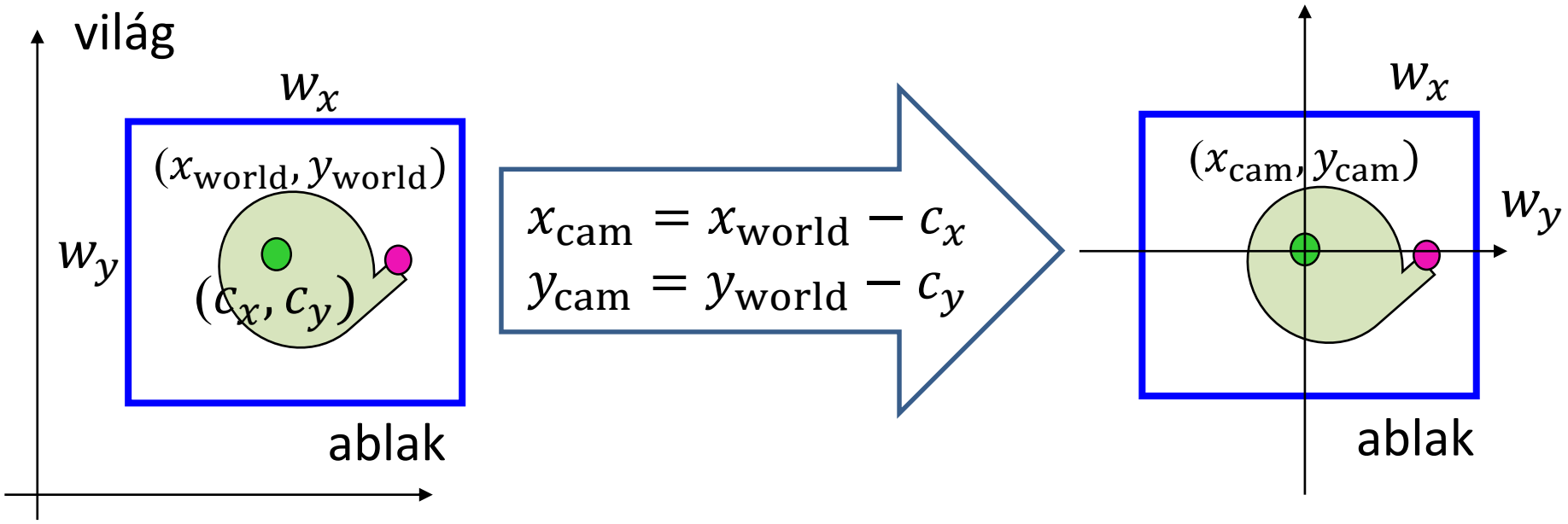
mat4 osztály

```
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”

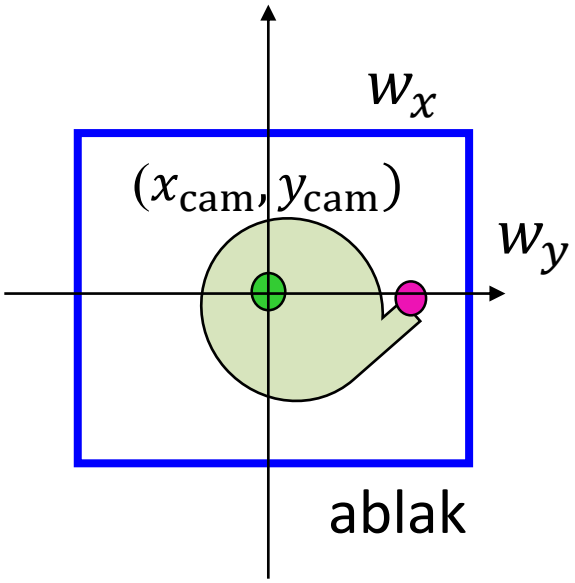
```
inline mat4 TranslateMatrix(vec2 t) {
	return mat4( vec4(1,   0,   0,  0),
	             vec4(0,   1,   0,  0),
	             vec4(0,   0,   1,  0),
	             vec4(t.x, t.y, 0,  1));
}

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 közepe az origóba

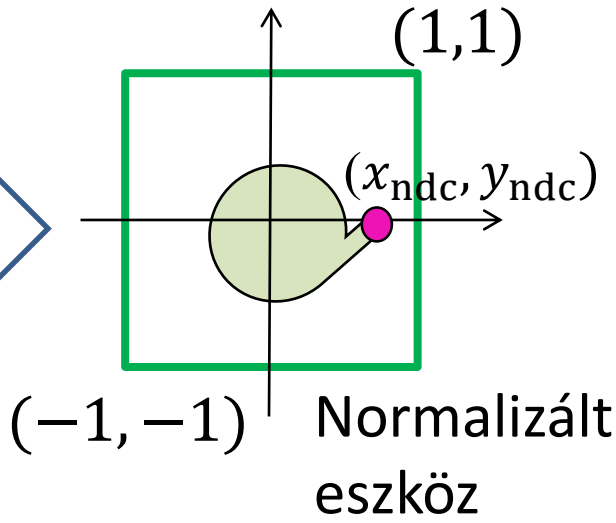


$$[x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}}, 1] = [x_{\text{world}}, y_{\text{world}}, z_{\text{world}}, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -c_x & -c_y & 0 & 1 \end{bmatrix}$$

Projekció: P()
Kameraablak a (-1, -1)-(1, 1) négyzetbe



$$x_{\text{ndc}} = x_{\text{cam}} \cdot 2/w_x$$
$$y_{\text{ndc}} = y_{\text{cam}} \cdot 2/w_y$$



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

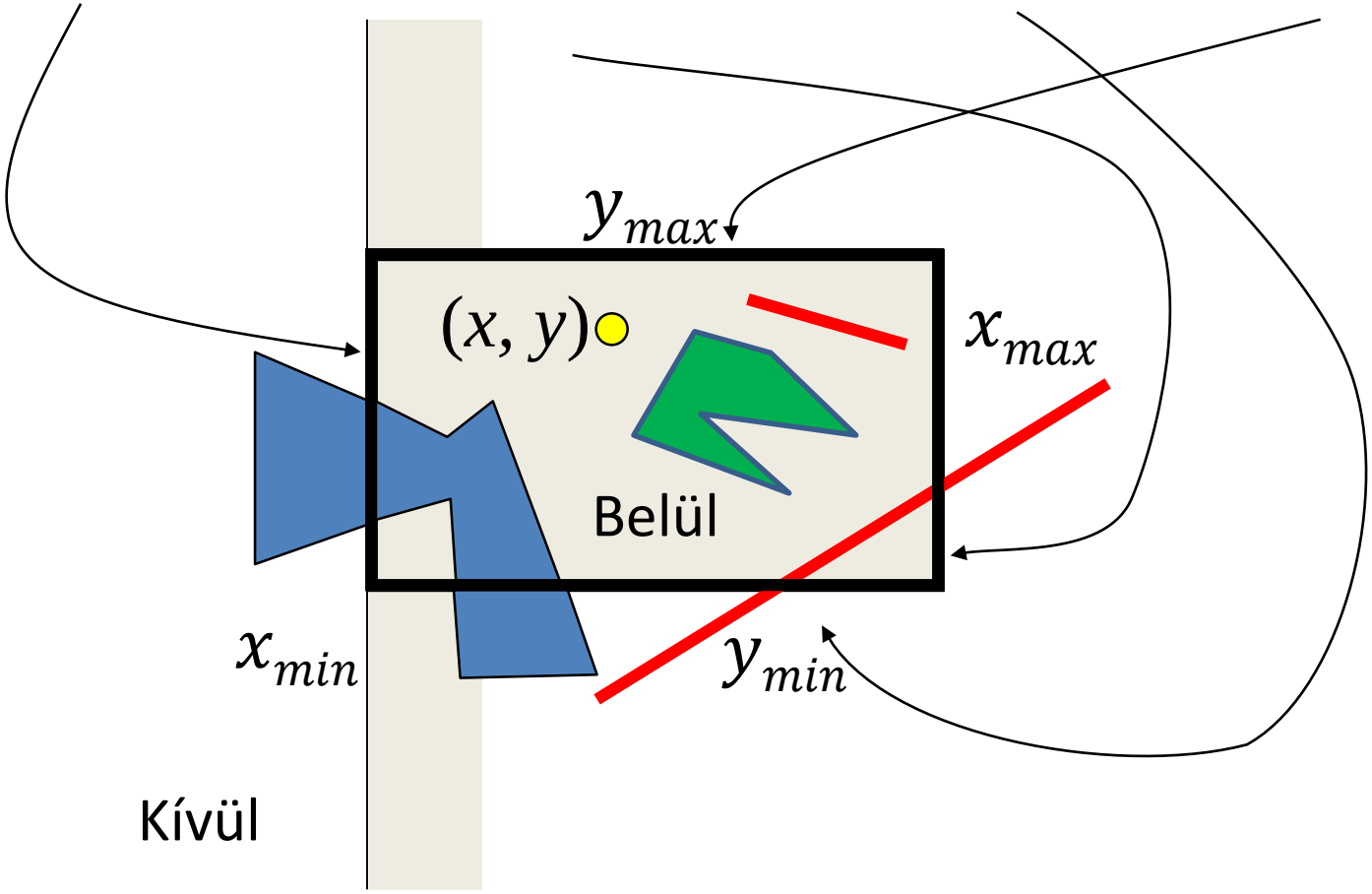
2D kamera

```
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; }
};
```

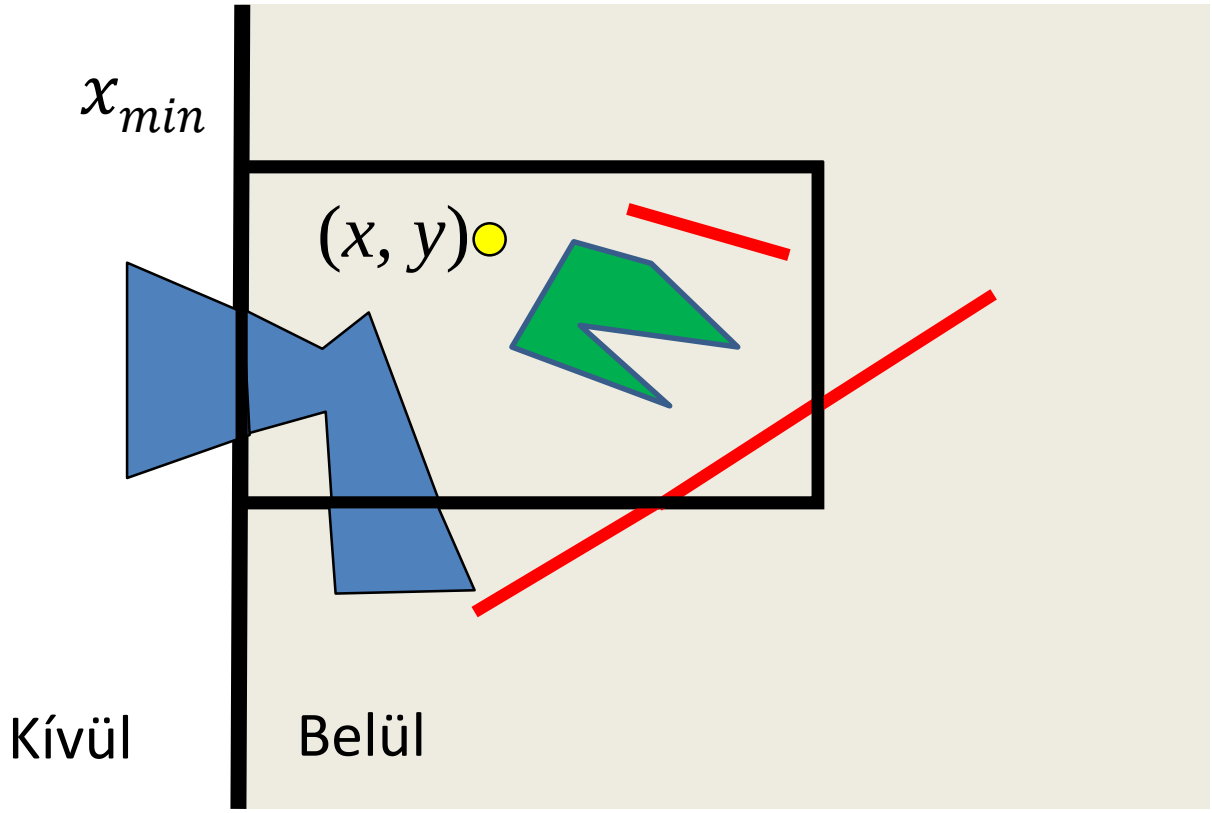
2D vágás

Pont vágás:

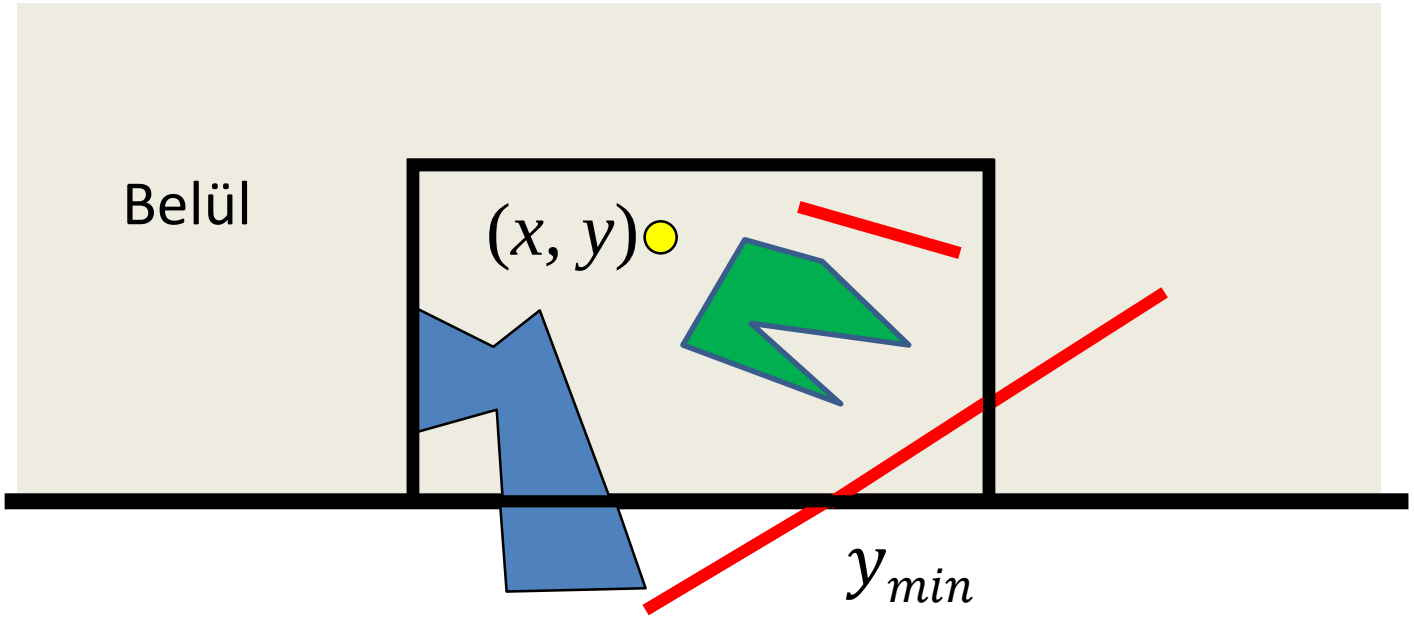
$x > x_{min} = -1,\ x < x_{max} = +1,\quad y > y_{min} = -1,\ y < y_{max} = +1$



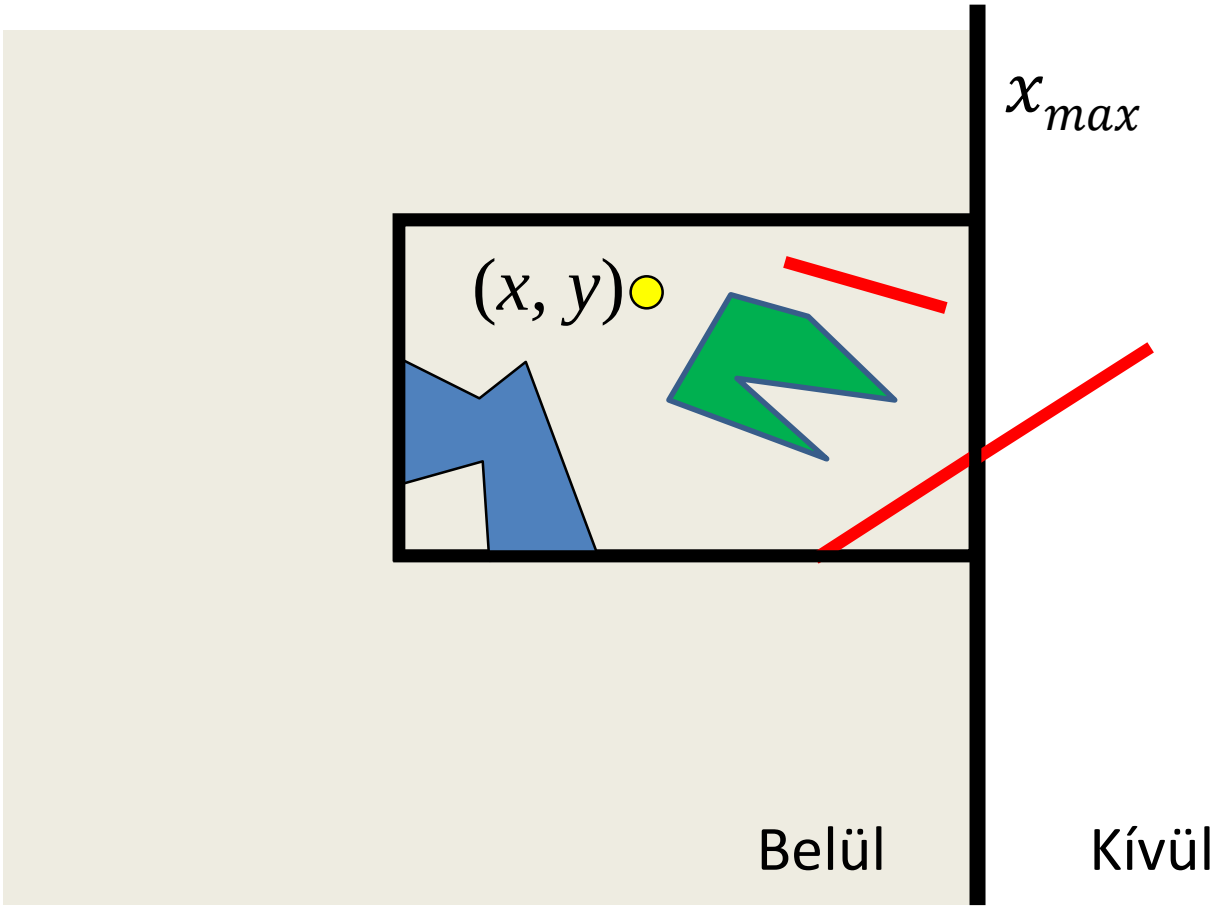
Vágás



Vágás

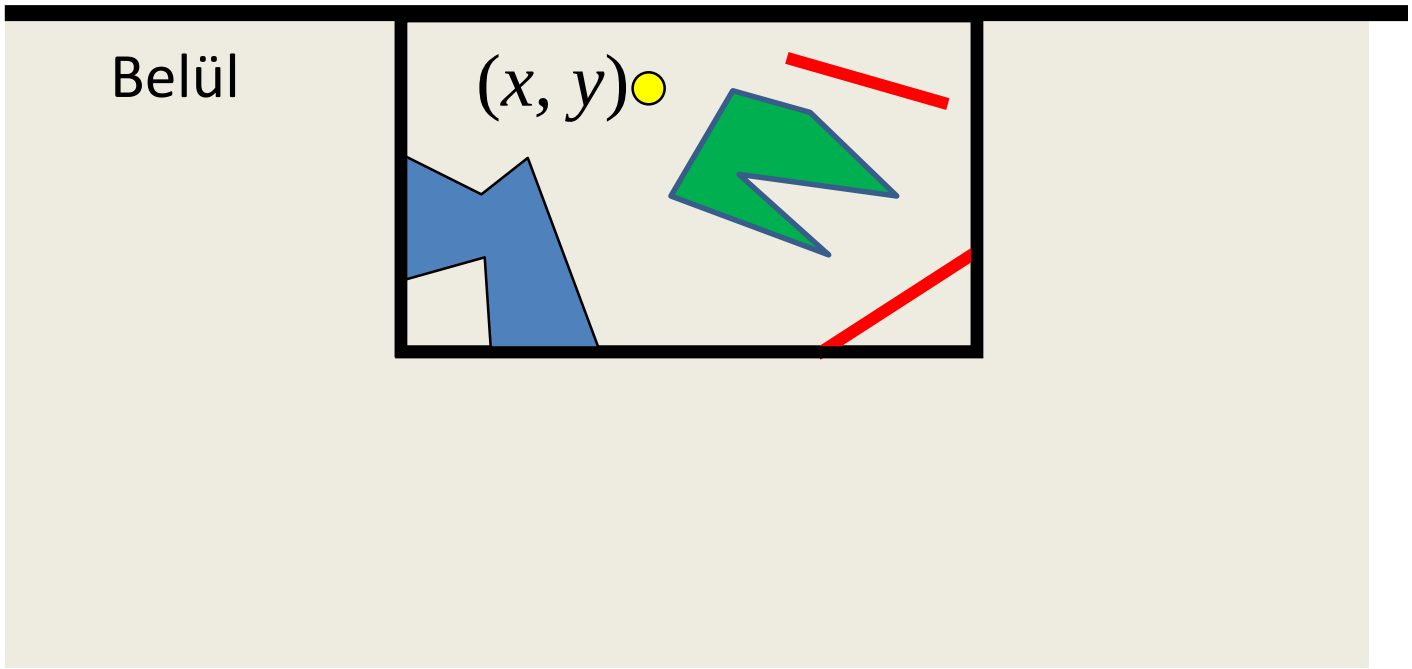


Vágás

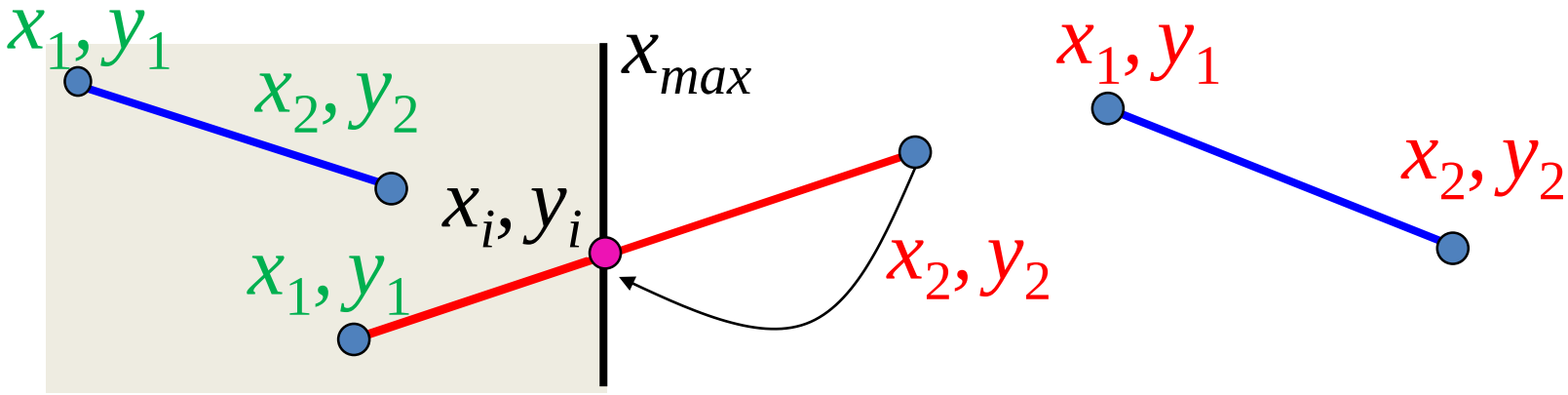


Vágás

Kívül



2D szakasz vágás: $x < x_{max}$



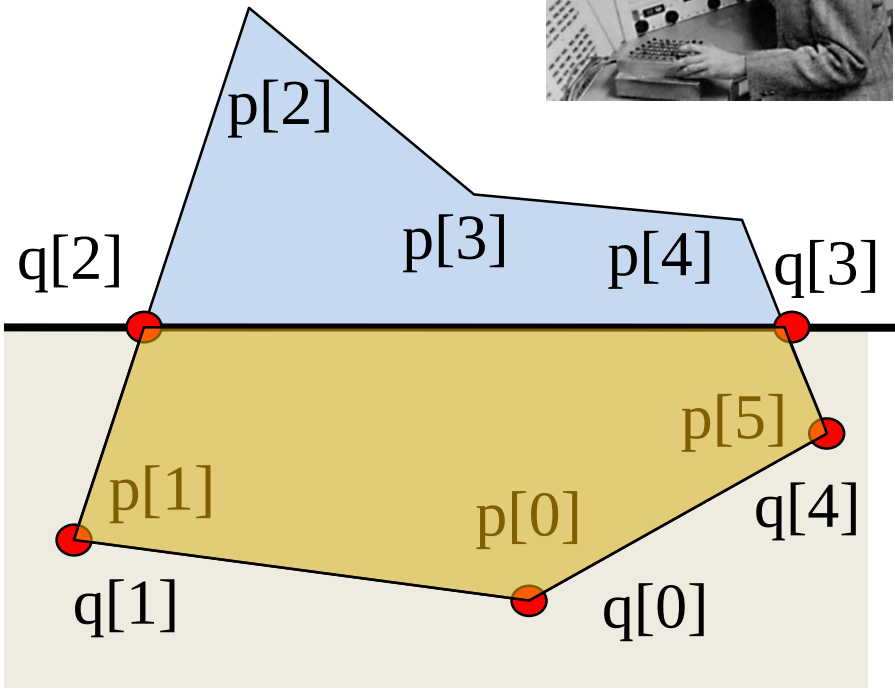
$$x(t) = x_1 + (x_2 - x_1)t, \quad y(t) = y_1 + (y_2 - y_1)t$$

$$x = x_{max}$$

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

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

(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

$-w < X < w$
$-w < Y < w$
$-w < Z < w$

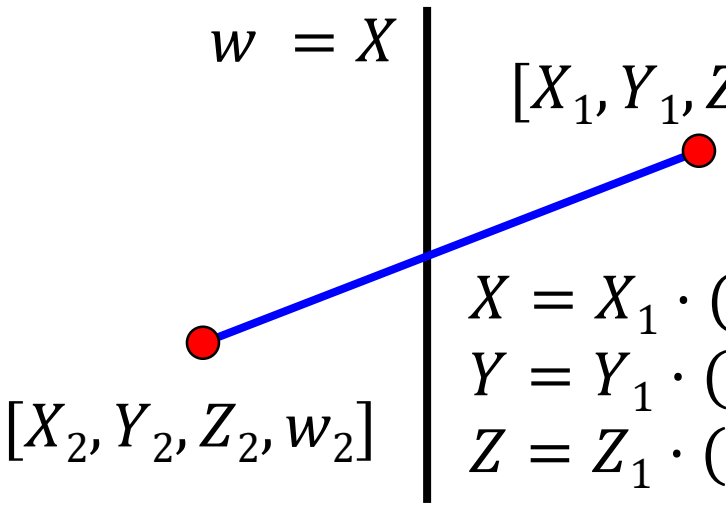
$-w > X > w$
$-w > Y > w$
$-w > Z > w$

3D szakasz/poligon vágás homogén koordinátákban (GPU)

$$-w < X < w$$
$$-w < Y < w$$
$$-w < Z < w$$

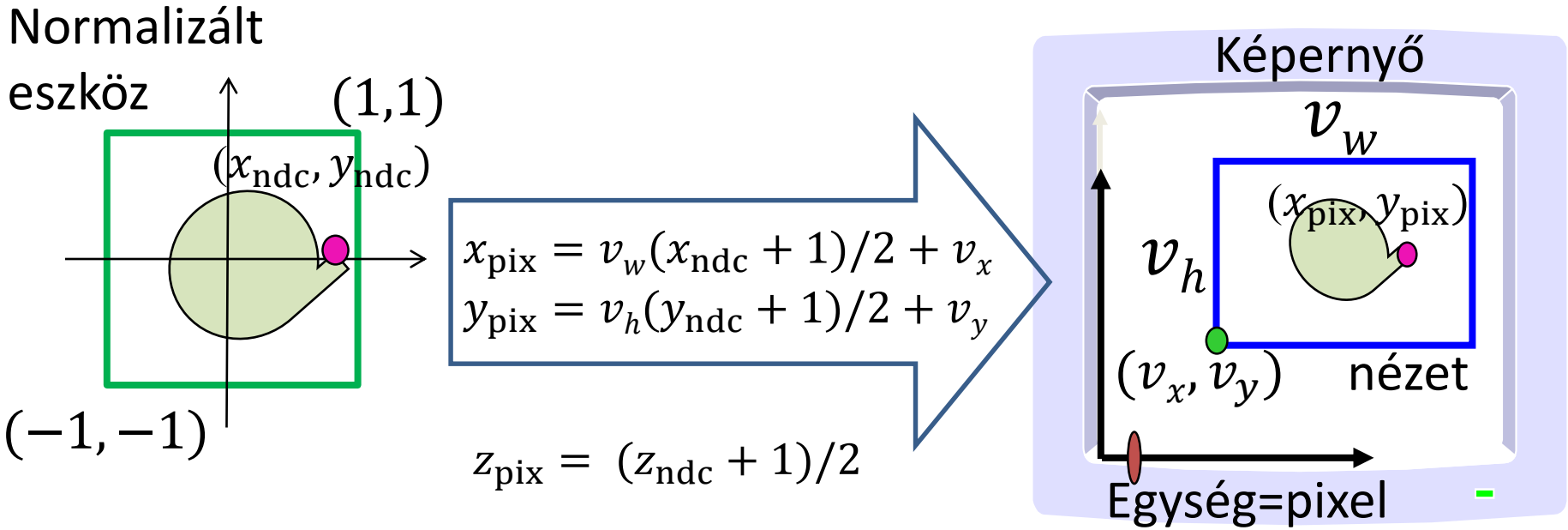
$$w = w_1 \cdot (1-t) + w_2 \cdot t =$$
$$= X = X_1 \cdot (1-t) + X_2 \cdot t$$

$$t = \ldots$$



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

Viewport transzformáció:
Normalizáltból képernyő koordinátákba (GPU)

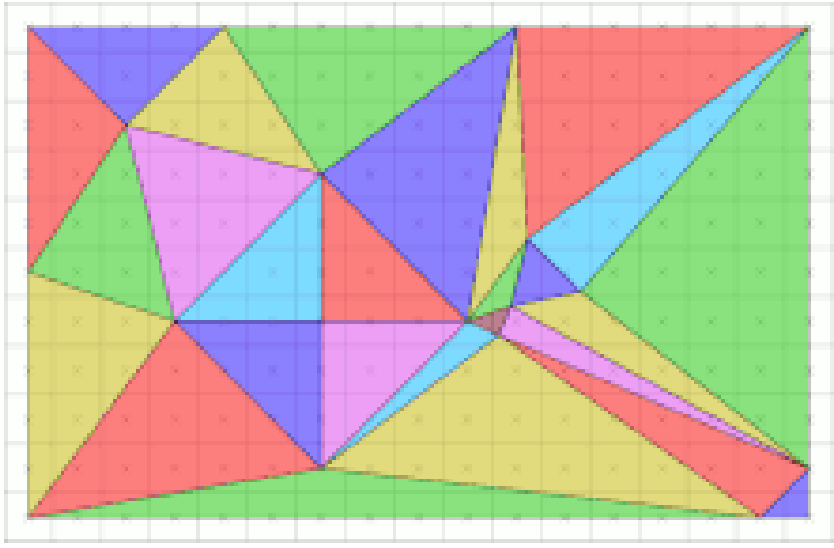


“The computer was born to solve problems that did not exist before.”

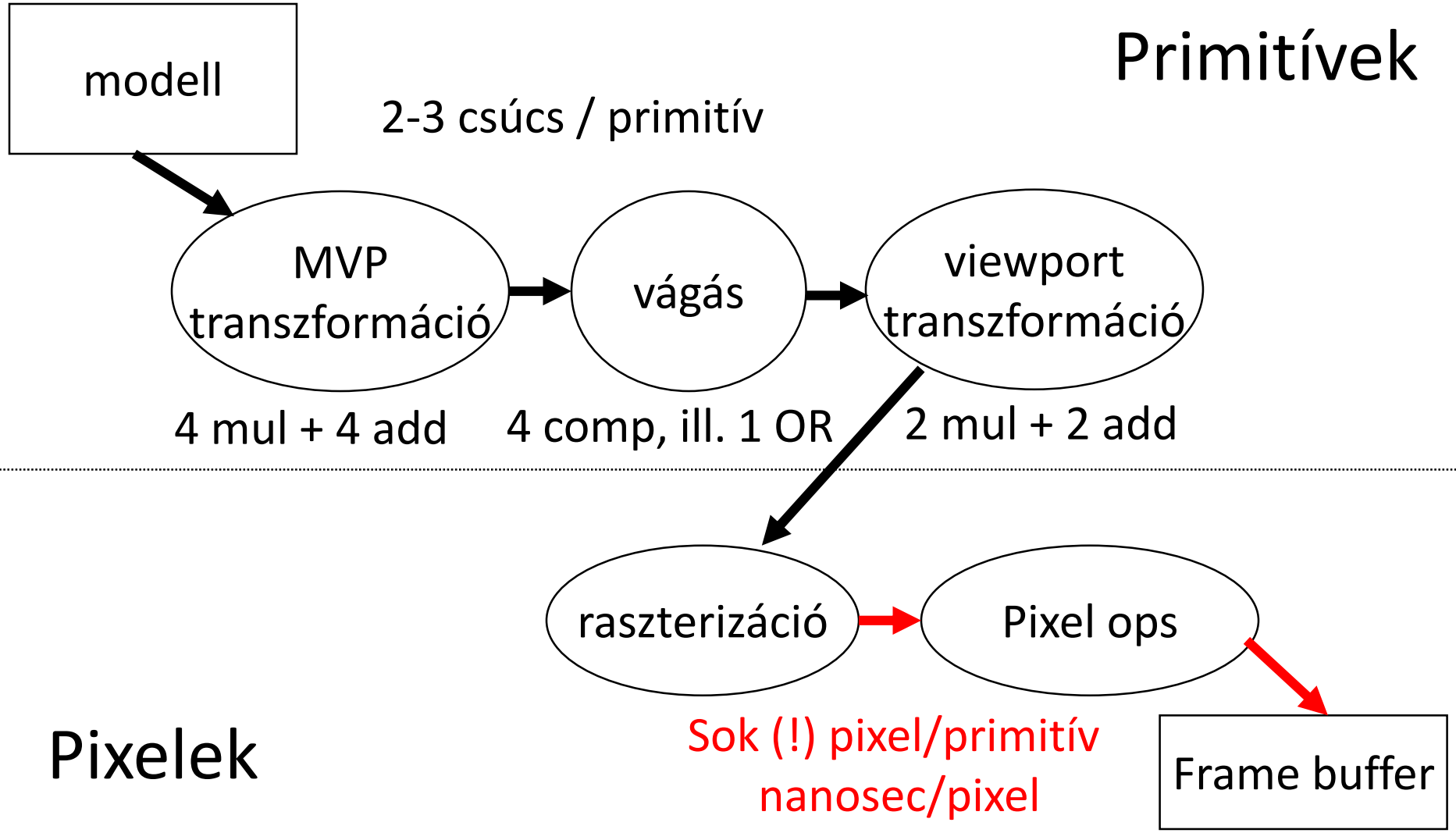
Bill Gates

2D képszintézis
4. Rasterizáció

Szirmay-Kalos László



Raszterizáció (GPU)

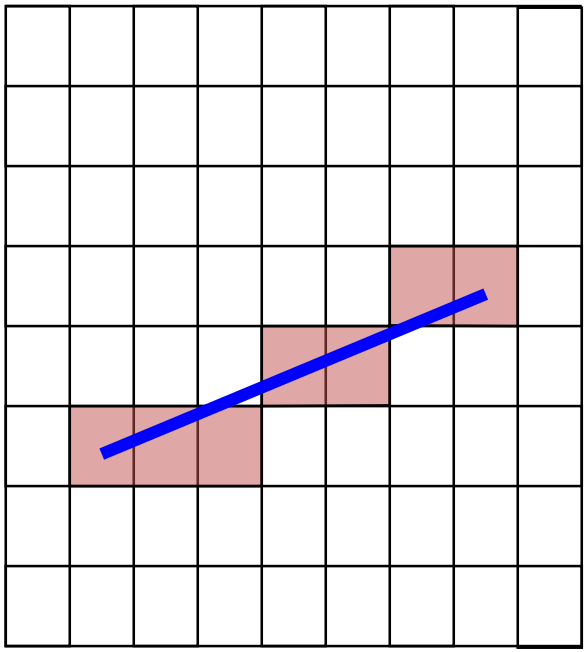


Pont rajzolás

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:

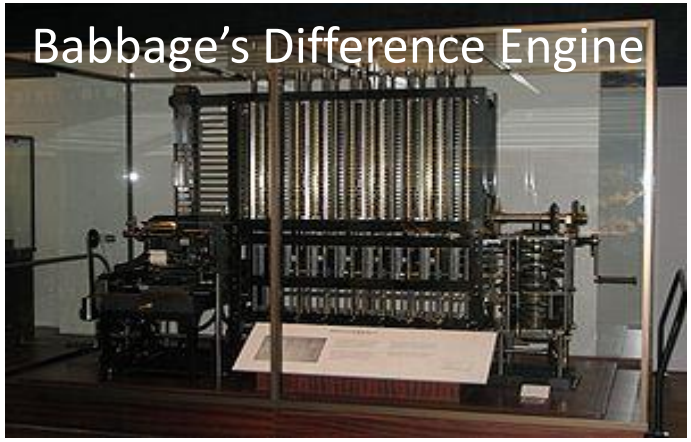
$$y = mx + b$$

x2 > x1, |x2-x1| >= |y2-y1| 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

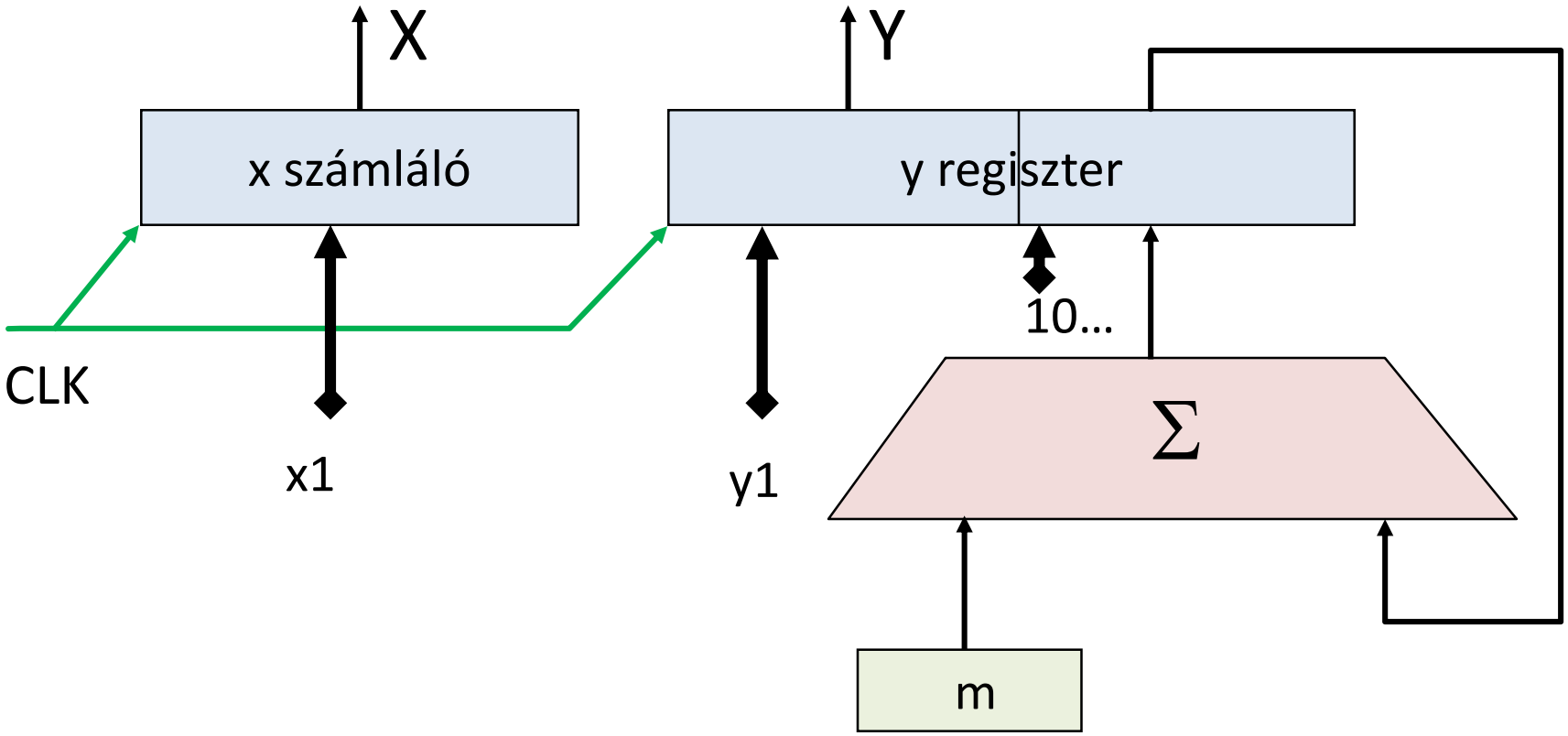
$$y(x) = mx + b = y(x-1) + m$$



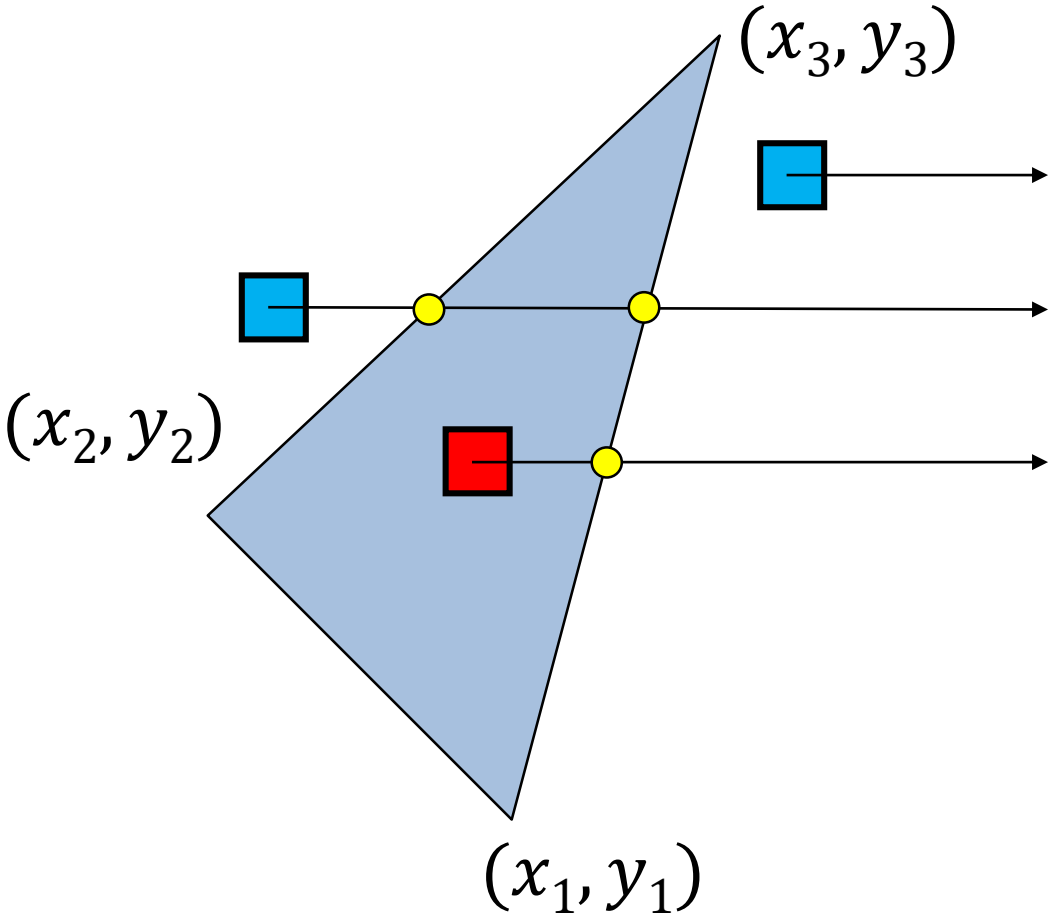

```
LineFloat (short x1, short y1,
           short x2, short y2) {
    float m = (float)(y2-y1)/(x2-x1);
    float y = y1;
    for(short x = x1; x <= x2; x++) {
        short Y = round(y);
        write(x, Y, color);
        y = y+m;
    }
}
```

```
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;
    }
}
```

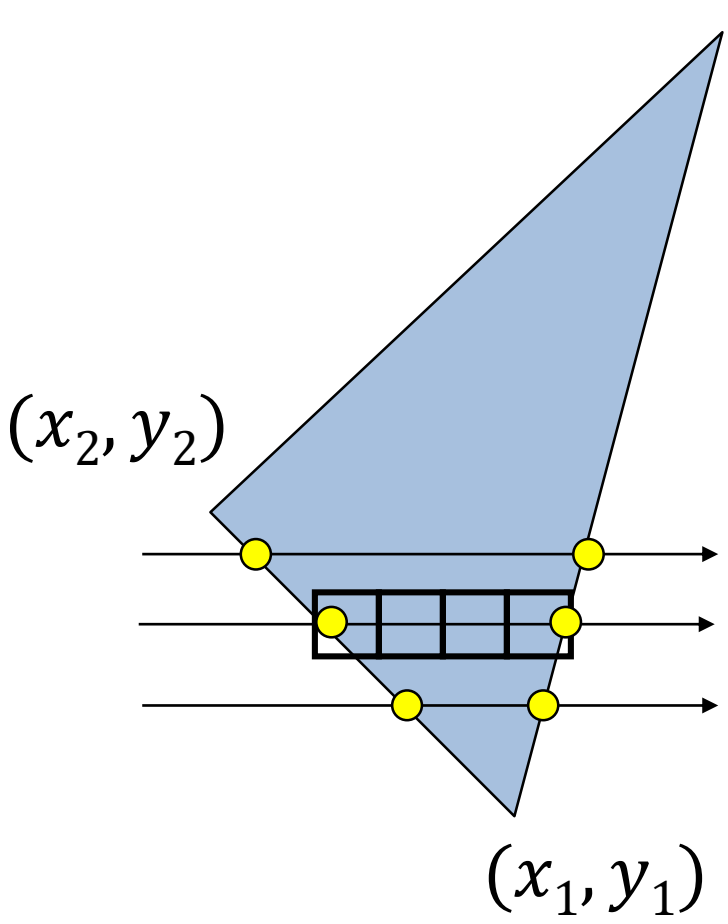
DDA szakaszrajzoló hardver



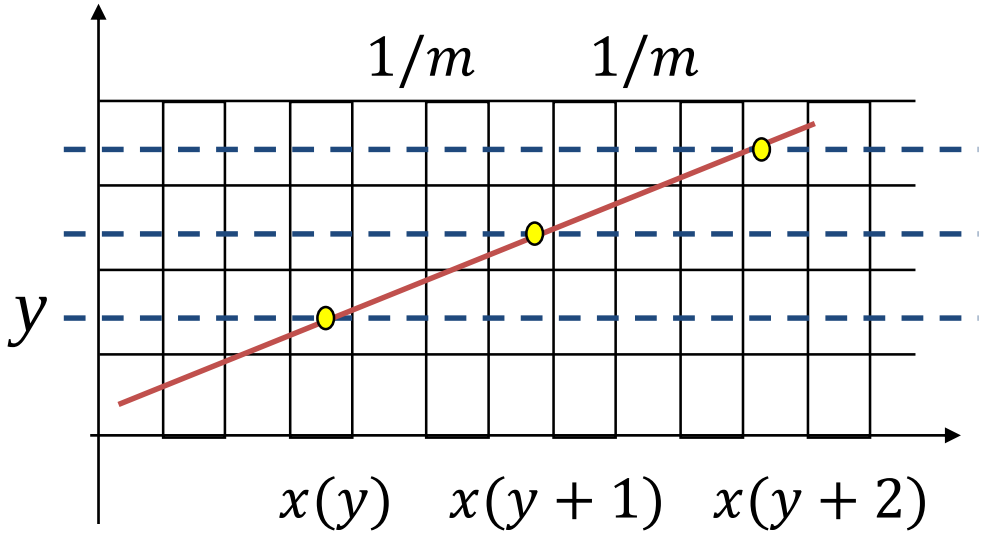
Naív háromszög kitöltés



Háromszög kitöltés



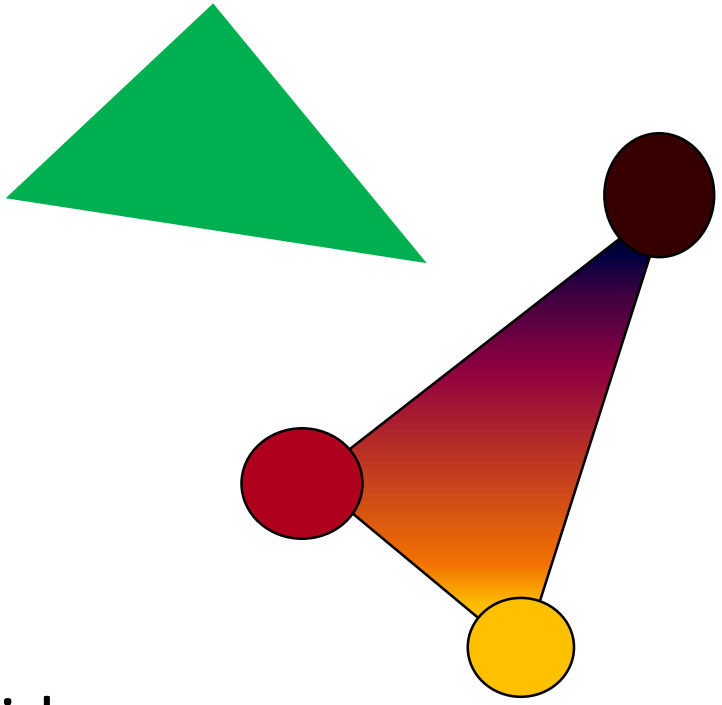
$$y = mx + b$$



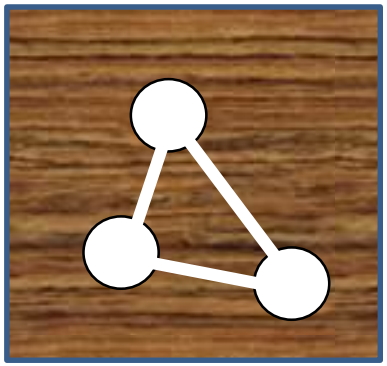
$$x(y+1) = x(y) + 1/m$$

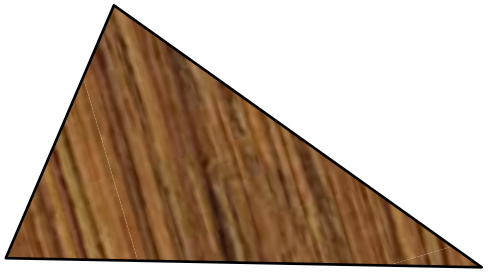
Milyen színű legyen a pixel?

- Uniform az egész objektum



- Csúcspont tulajdonságokból interpolált szín
- Pl. Textúrázás (2D):
  Csúcspont textúrakoordináták interpolációja a pixelekre, majd textúra kiolvasás.





Ellenőrző kérdések

- Bizonyítsa be, hogy bármely 4+ csúcsú sokszögnek van diagonálja!
- Bizonyítsa be a kétfül tételt!
- 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 koordináta-tengelyekkel 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?