SZÁMÍTÓGÉPI GRAFIKA

Javasolt irodalom:

- W.M NEWMAN-R.F. SPROULL : Interaktív számítógépes grafika
- KOMÁROMI IMRE: Számítógépes grafika
- JUHÁSZ IMRE: Számítógépi geometria és grafika
- SZABÓ JÓZSEF: Számítógépi grafika
- FOLEY-VAN DAM -FEINERHUGHES-PHILLIPS:Introduction to the computer graphics
- DONALD HEARN-M. PAULINE BAKER:Computer Graphics

1. Néhány lehetséges felhasználói terület:

- Felhasználói felületek (Pl. Windows 3.1, stb)
- Interaktív diagrammok, hisztogrammok (2D vagy 3D)
- Térképészet
- Orvostudomány
- Tervezés (AutoCAD)
- Multimédia rendszerek
- Tudományos kísérletek eredményeinek megjelenítése, animácó

2. A számítógépi grafika rövid története

1950. CRT display -Whirlwind Computer
1963. SUTHERLAND 'Sketchpad' rajzoló rendszer
1964. GM DAC rendszer
1980- A PC-k elterjedése beépített raszter grafikával (IBM,APPLE)
bit-térképek, pixel vagy pel
desktop-felület
window manager

3. Grafikus output eszközök

a. DISPLAY - MONITOR tipusok működési elv szerint
Raszteres, katódsugárcsöves monitorok:

i. INTERLACE	páros-páratlan sorok
ii. NONINTERLACE	soronként

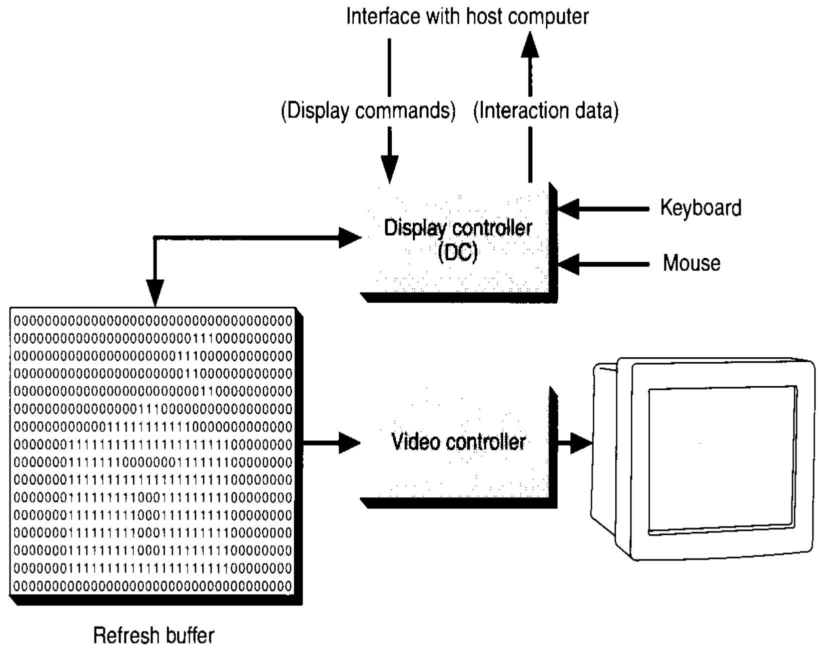
Monitor és videokártya típusok

1980	CGA	320x200 4 szin, 640x200 2 szin
1984	MDA/HERKULES	720x348 2 szin
1984	EGA	640x350 16 szin
1987	VGA	640x480 16 szin, 320x200 256 szin
1990	SVGA	VESA szabvány
		640x480 256 szin 800x600 32 K 1024x768 64 K 164 ezer szin 1280x1024 16 M 16 millió szin 1600x1200 TRUE COLOR

b. Rajzgépek

c. Egyéb grafikus eszközök - digitalizálók
- fénytoll
- egér
- pozicionáló gömb
- scanner

A raszteres display működési elve



A szoftverek hordozhatósága és a grafikus szabványok

3D Core Graphics System -1977
ANSI-American National Standards Institute
ISO-International Standards Organization
GKS: Graphical Kernel System -2D az első hivatalos grafikai szabvány 1985
GKS 3D kiterjesztése 1988
PHIGS Programmer's Hierarchical Interactive Graphic System (ANSI sz. 1988)
PHIGS PLUS (ANSI/ISO 1992)

SRGP Simple Raster Graphic
Primitivek: egyenesek,poligonok,körök,ellipszisek,szöveg

BP70

GRAPH.TPU Konstansok és Típusok

Konstansok	Típusok
Bar3D Constants	ArcCoordsType
BitBlt Operators	FillPatternType
Clipping Constants	FillSettingsType
Color Constants	Graphics Memory Pointers
Colors for the 8514	LineSettingsType
Fill Pattern Constants	PaletteType
Graphics Drivers	PointType
Graphics Modes for Each Driver	TextSettingsType
Justification Constants	ViewPortType
Line-Style and Width Constants	
Text-Style Constants	

PointType
```
PointType = record
        X, Y : integer;
end;
```

procedure Line(x1, y1, x2, y2: Integer);

Példa:

```
uses Crt, Graph;
var Gd, Gm: Integer;
begin
Gd := Detect;
InitGraph(Gd, Gm, ' ');
if GraphResult <> grOk then
        Halt(1);
Randomize;
repeat
        Line(Random(200), Random(200), Rand
until KeyPressed;
Readln;
CloseGraph;
end.
```

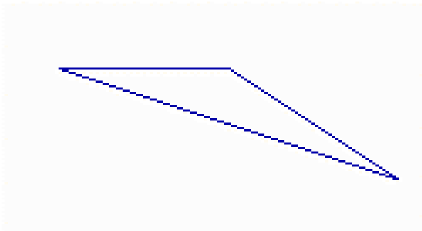
Vagy:
procedure LineP(P1, P2: PoinType);

procedure DrawPoly(NumPoints: Word; var PolyPoints);

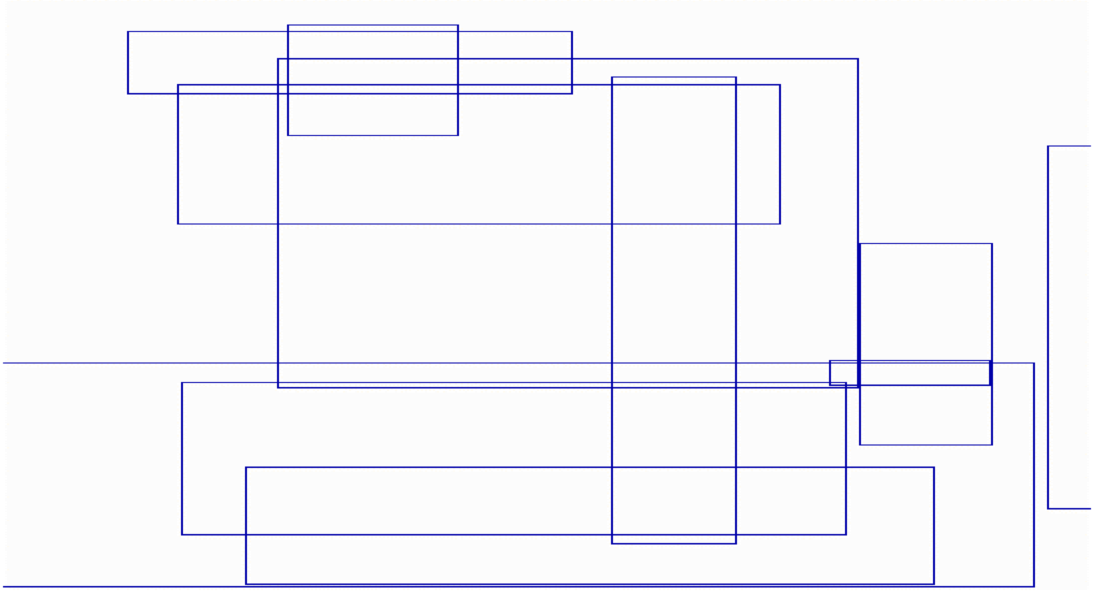
Példa:

```
uses Graph;

  const
   Triangle: array[1..4] of PointType = ((X: 50; Y: 100), (X: 100; Y:100),
     (X: 150; Y: 150), (X:  50; Y: 100));
  var Gd, Gm: Integer;
  begin
   Gd := Detect;
   InitGraph(Gd, Gm, ' ');
   if GraphResult <> grOk then
     Halt(1);
   DrawPoly(SizeOf(Triangle) div SizeOf(PointType), Triangle);{ 4 }
   Readln;
   CloseGraph;
  end. uses Graph;
```

```
{Rectangl.PAS}

{Sample code for the Rectangle procedure.}

uses Crt, Graph;

var
 GraphDriver, GraphMode: Integer;
 X1, Y1, X2, Y2: Integer;
 ch : Char;
begin
 GraphDriver := Detect;
 InitGraph(GraphDriver, GraphMode, ' ');
 if GraphResult<> grOk then
   Halt(1);
 Randomize;
 SetColor(Blue);
 SetBkColor(White);
 repeat
         X1 := Random(GetMaxX);
         Y1 := Random(GetMaxY);
         X2 := Random(GetMaxX - X1) + X1;
         Y2 := Random(GetMaxY - Y1) + Y1;
         ch := Readkey;
         Rectangle(X1, Y1, X2, Y2);
 until ch='Q';
 CloseGraph;
```



end.

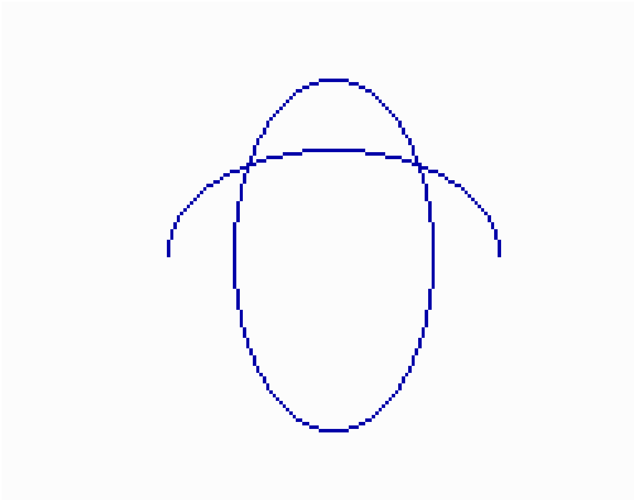


procedure Circle(X,Y: Integer; Radius: Word);

procedure Ellipse(X, Y: Integer; StAngle, EndAngle: Word; XRadius, YRadius: Word);

Példa:

```
uses Graph;

var Gd, Gm: Integer;
begin
 Gd := Detect;
 InitGraph(Gd, Gm, ' ');
 SetColor(Blue);
 SetBkColor(White);
 if GraphResult <> grOk then
   Halt(1);
 Ellipse(100, 100, 0, 360, 30, 50);
 Ellipse(100, 100, 0, 180, 50, 30);
 Readln;
 CloseGraph;
end.
```



Attributumok

a. Vonal fajták és vastagságok

procedure SetLineStyle(LineStyle: Word; Pattern: Word; Thickness: Word);

Line Styles:

- SolidLn 0
- DottedLn 1
- CenterLn 2
- DashedLn 3
- UserBitLn 4 (User-defined line style)

Line Widths:

- NormWidth 1
- ThickWidth 3

b. Szinek

procedure SetColor(Color: Word);

Sötét szinek:
(Tinta & Papír)

- Black 0
- Blue 1
- Green 2
- Cyan 3
- Red 4
- Magenta 5
- Brown 6
- LightGray 7

Világos szinek:
(Tinta)

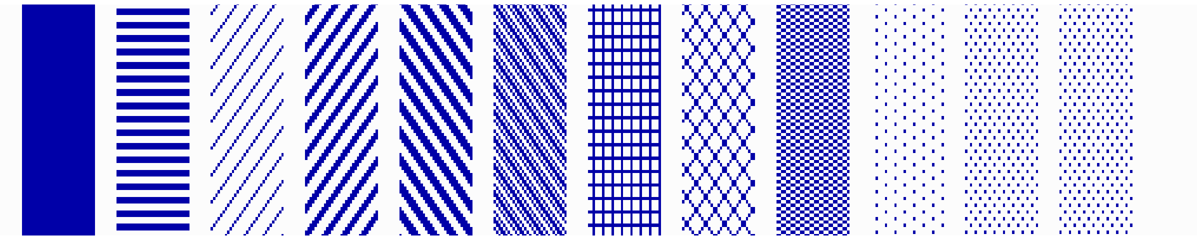
- DarkGray 8
- LightBlue 9
- LightGreen 10
- LightCyan 11
- LightRed 12
- LightMagenta 13
- Yellow 14
- White 15

c. Kitöltések és minták

procedure SetFillStyle(Pattern: Word; Color: Word);

Konstans	Érték	Jelentés
• EmptyFill	0	Háttérszin
• SolidFill	1	Tintaszin
• LineFill	2	---- kitöltés
• LtSlashFill	3	/// kitöltés
• SlashFill	4	/// sűrű kitöltés
• BkSlashFill	5	\sűrű kitöltés
• LtBkSlashFill	6	\ kitöltés
• HatchFill	7	Négyzetrács kitöltés
• XhatchFill	8	Négyzetrács kitöltés
• InterleaveFill	9	Interleaving line
• WideDotFill	10	Widely spaced dot
• CloseDotFill	11	Closely spaced dot
• UserFill	12	User-defined fill

Minták:



Grafikus szoftwer csomaggal szemben támasztott elvárások

- Egyszerűség
- Következesség
- Teljesség
- Bolondbiztos
- Undo funkció

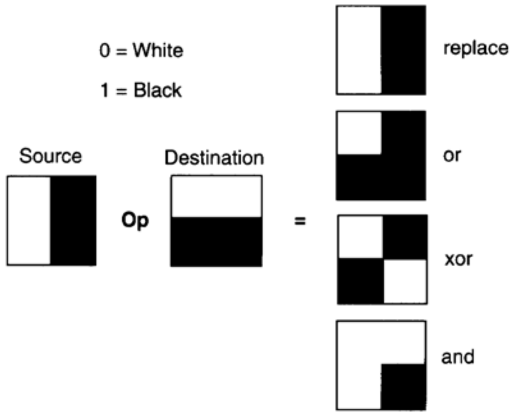
A raszter grafika néhány jellemzűje:

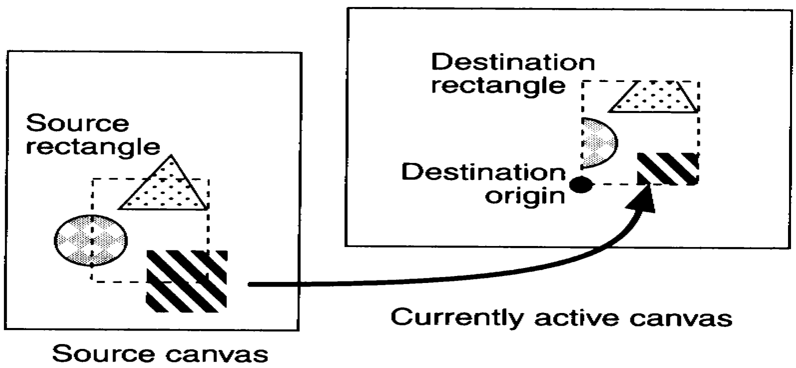
1. Canvanses-Vásznak

Példa:

```
uses Graph;

 var
  Gd, Gm: Integer;
  P: Pointer;
  Size: Word;
 begin
  Gd := Detect;
  InitGraph(Gd, Gm, ' ');
  if GraphResult <> grOk then
    Halt(1);
  Bar(0, 0, GetMaxX, GetMaxY);
  Size := ImageSize(10, 20, 30, 40);
  GetMem(P, Size);   { Allocate memory on heap }
  GetImage(10, 20, 30, 40, P^);
  Readln;
  ClearDevice;
  PutImage(100, 100, P^, NormalPut);
  Readln;
  CloseGraph;
 end.
```

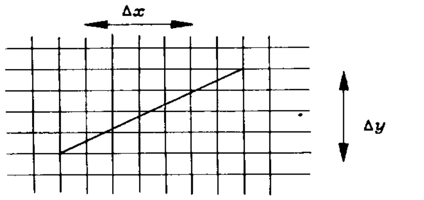
Írás módok:



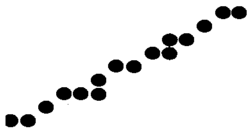


Az alapvető raszteres algoritmusok:

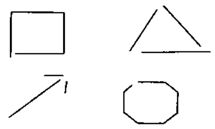
1. Egyenes rajzolása



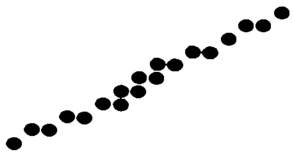
2.3. ábra. Előfordulhat, hogy két rácspontot összekötő egyenesszakasz egyetlen további rácsponton sem halad át



2.4. ábra. Egy gyenge vonalrajzoló algoritmus eredménye

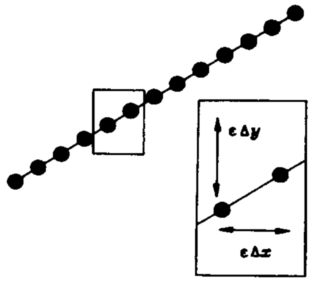


2.5. ábra. Egymással pontatlanul csatlakozó egyenesszakaszokkal rajzolt szimbólumok

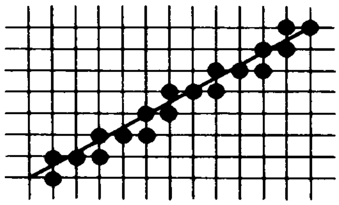


2.6. ábra. A pontok sűrűsödése által okozott egyenlőtlen vonalfedettség

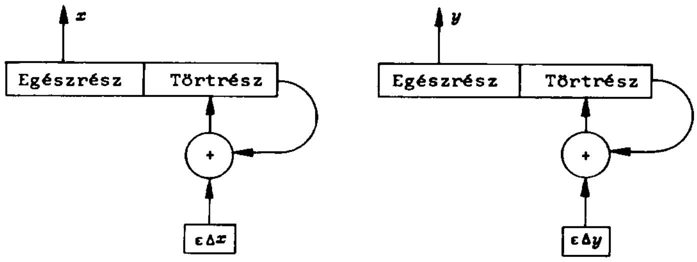
Szimmetrikus DDA (digitális differenciál analizátor



2.7. ábra. Ideális növekményes módszer egyenes generálásra



2.8. ábra. Szimmetrikus DDA-val rajzolt egyenes



2.9. ábra. A szimmetrikus DDA elrendezése

Egyszerű DDA

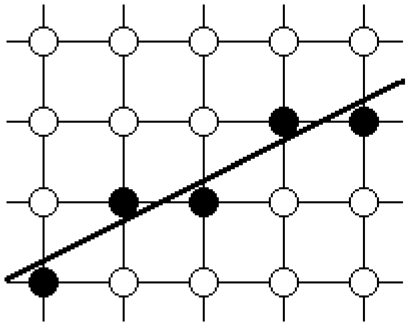
Midpoint algoritmus

Bárhogyan is származtassuk az egyenest, az egyenlete:

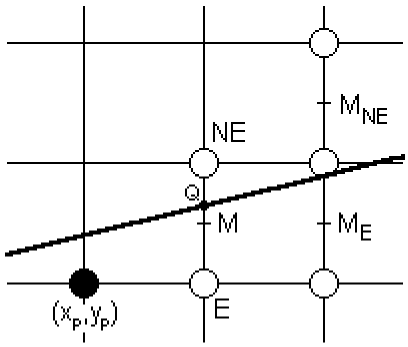
$$ax + by + c = 0$$

alakra hozható, ahol a és b egyszerre nem lehet nulla.

A kitöltött körök a megvilágított pixeleket jelentik.



Legyenek az egyenest meghatározó pontok P_1 (x_1 , y_1) és P_2 (x_2 , y_2). Az algoritmus ismertetetéséhez tegyük fel, hogy a meredekség $0 \le m \le 1$. Más meredekség esetén a megfelelõ változtatásokkal az algoritmus hasonlóan mûködik.



Az egyenest balról jobbra haladva rajzoljuk ki. Legyen az éppen megjelenített pont P (x_p , y_p) (3. ábra), ekkor a következõ megrajzolandó raszterpont az E (East) és az NE (North East) pontok valamelyike lehet. Közülük azt a pontot kell kigyújtani, amelyik közelebb van az elméleti egyeneshez. A választás a két rácspont között elhelyezkedõ felezõpont* (M) segítségével történik. Ha az egyenes az M pont felett halad el, akkor az NE pontot jelenítjük, különben az E-t. Az M pont helyzetét analitikusan határozzuk meg. Az egyenes egyenletét az

$$F (x , y) = a x + b y + c = 0$$

formában tekintjük, ahol $a = (y_1 - y_2)$, $b = (x_2 - x_1)$, és $c = (y_2 - y_1) x_1 - (x_2 - x_1) y_1$. Feltehetjük, hogy b pozitív, - különben a pontokat felcseréljük -, ezért $F(x , y) > 0$, ha az (x , y) pont az egyenes felett helyezkedik el, illetve $F(x , y) < 0$, ha az egyenes alatt. Jelöljük az M-hez tartozó függvényértéket d-vel:

$$d = F(M) = F (x_p + 1 , y_p + 1/2) = a (x_p + 1) + b (y_p + 1/2) + c.$$

Ha $d < 0$, akkor NE-t választjuk, ha $d > 0$, akkor E-t, ha $d = 0$, akkor választhatjuk bármelyiket, de megegyezés szerint E-t választjuk. Ezután d új értékét - inkrementális módon - a régi értékébõl számoljuk ki. Jelölje ezt d_{old}, az új értéket d_{new}. Ekkor a d_{new} függ az új pont meg választásától.

- Ha az E pontot választjuk, akkor

$d_{new} = F(M_E) = F (x_p + 2 , y_p + 1/2) = a (x_p + 2) + b (y_p + 1/2) + c = d_{old} + a$,

azaz ekkor d-t

$$\Delta_E = d_{new} - d_{old} = a$$

-val kell növelni.

- Ha az NE pontot választjuk, akkor

$d_{new} = F(M_{NE}) = F(x_p + 2 , y_p + 3/2) = a (x_p + 2) + b (y_p + 3/2) + c = d_{old} + a + b$.

Ekkor d-t

$$\Delta_{NE} = d_{new} - d_{old} = a + b$$

-vel növeljük.

Most már ha ismerjük x_p , y_p és d aktuális értékét, akkor tovább tudunk lépni, meg tudjuk határozni az újabb értékeket. Az elinduláshoz határozzuk meg a kezdeti értékeket! Az első kirajzolandó pont a P_1 (x_1 , y_1), azaz

$$(x_0 , y_0) := (x_1, y_1),$$

ekkor a d kezdő értéke:

$d_0 = F (x_1 + 1 , y_1 + 1/2) = a x_1 + b y_1 + c + b/2 = F (x_1 , y_1) + a + b/2$, de a P_1 (x_1 , y_1) pont rajta van az egyenesen, így

$$d_0 = a + b/2.$$

Ahhoz, hogy a kettővel való osztást elkerüljük definiáljuk át az F (x , y) függvényt:

$$F (x , y) = 2 (ax + by + c).$$

Ezt megtehetjük, mert csak a d előjelére van szükség és a 2-vel való szorzás ezt nem változtatja meg. Ekkor

$$d_0 = 2a + b,\ \ \Delta_E = 2a,\ \text{ és }\ \Delta_{NE} = 2\,(a + b)$$

-t kapjuk, minden más változatlan.

Az iterációs lépést addig kell ismételni, amíg az utolsó pontot is ki nem rajzoljuk.

```
procedure
Line(x1,y1,x2,y2:integer);
var

a,b,x,y,d,deltaE,deltaNE:inte
ger;
begin
  a:=y1-y2;
  b:=x2-x1;
  d:=2*a+b; { d kezdõértéke }
  deltaE:=2*a; { d növekménye
E esetén }
  deltaNE:=2*(a+b);  {  és  NE
esetén }
  x:=x1; { a kezdõpont }
  y:=y1; { koordinátái }
  WritePixel(x,y);
  while x<x2 do begin
    if d>=0 then begin { E }
      d:=d+deltaE;
      x:=x+1;
    end
    else begin        { NE }
      d:=d+deltaNE;
      x:=x+1;
      y:=y+1;
    end;
    WritePixel(x,y);
  end; { while }
end;
```

3. Kör scan-konverziója

3.1. Matematikai háttér

A kör azon pontok halmaza a síkban, amelyek egy adott, a síkra illeszkedõ C ponttól egyenlõ (r > 0) távolságra vannak. A C pontot a kör középpontjának, az r távolságot a kör sugarának nevezzük. Egy pontot a kör belsõ (illetve külsõ) pontjának nevezünk, ha a pont távolsága a kör középpontjától kisebb (illetve nagyobb) a kör sugaránál.

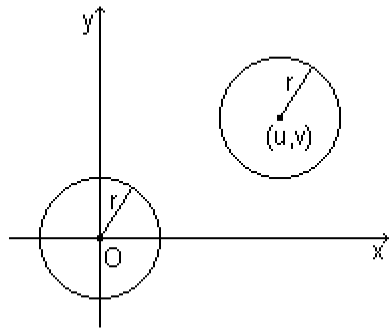
Ha rögzítünk egy [x , y] koordinátarendszert, akkor az origó középpontú, r sugarú kör egyenlete:

$$x^2 + y^2 = r^2$$

Ebbõl pedig könnyen levezethetõ, az (u , v) középpontú, r sugarú kör egyenlete:

$$(x - u)^2 + (y - v)^2 = r^2$$

Az egyenletekben xy-os tag nem szerepel és a négyzetes tagok együtthatója megegyezik. Az utóbbi egyenletet átrendezve a következõ összefüggést kapjuk:



5. ábra

$$F (x , y) = (x - u)^2 + (y - v)^2 - r^2 = 0$$

Az F (x , y) függvénybe a körre illeszkedõ pontok koordinátáit helyettesítve nulla értéket kapunk. Az (x_1 , y_1) pont akkor és csakis akkor belsõ pont, ha

$$F (x_1 , y_1) < 0$$

és az (x_2 , y_2) pont akkor és csakis akkor külsõ pont, ha

$$F (x_2 , y_2) > 0.$$

3.2. Kör rajzolás midpoint algoritmussal

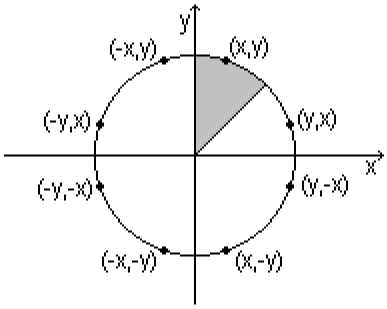
Szeretnénk meghatározni egy adott (u , v) középpontú és r sugarú kör pontjait. Az egyik lehetséges megoldás, ami eszünkbe juthat a trigonometrikus függvényeket használja, az:

$$x = u + r \cos t \quad \text{és} \quad y = v + r \sin t$$

összefüggések segítségével határozza meg a kör pontjait. Számunkra ez a módszer most nem megfelelõ, mert a trigonometrikus függvények kiszámolása, ami valós aritmetikát követel meg, túlságosan sok idõt vesz igénybe. Rekurziót alkalmazva lehet valamelyest gyorsítani az algoritmuson, de az így kapott algoritmus sem elég gyors, és a vonalrajzolásra megfogalmazott követelményeinknek sem tesz eleget. Semmi sem garantálja azt, hogy a vonal vastagsága egyenletes és folytonos lesz. De ahelyett, hogy számba vennénk a számunkra nem megfelelõ módszereket, nézzünk meg egy igen hatékony algoritmust és annak egy gyorsítását. Ez az algoritmus az egyenes rajzolásnál tárgyalt midpoint algoritmus továbbfejlesztése.

3.3. Nyolcas szimmetria elve

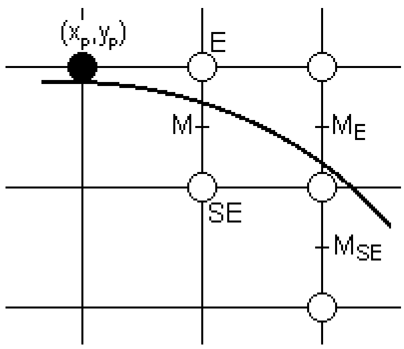
Mielõtt az algoritmus leírásába kezdenénk, nézzünk meg egy a szakirodalomban nyolcas szimmetriaként emlegetett tényt. Tekintsünk egy origó középpontú kört. Ha egy (x , y) pont rajta van a körön, akkor könnyen meghatározhatunk hét (speciális esetben három) másik pontot, ami szintén rajta van a körön (6. ábra). Ezért ha meghatározzuk a kör egy megfelelõ nyolcadának pontjait (pl. az ábrán satírozott részhez tartozó körív pontjait), akkor tulajdonképpen a teljes kört is meghatároztuk. Ezt kihasználva az algoritmus gyorsítható. Egyedüli feltétel az, hogy a kör középpontja az origóba essék. Ha egy nem origó középpontú kört akarunk rajzolni (az esetek többségében ez teljesül), akkor koordináta transzformációt alkalmazunk. A koordinátarendszer origóját a kör (u , v) középpontjába visszük. Másképpen mondva a kör pontjait úgy határozzuk meg, mintha a középpontja az origóban lenne, de kirajzolás elõtt a pontokat az (u , v) vektorral eltoljuk, s így a kívánt helyzetû kört kapjuk.



6. ábra

3.4. Elsõrendû differenciák módszere

Az elmondottak alapján a midpoint algoritmus origó középpontú kört feltételez, és csak a 7. ábrán látható kitüntetett nyolcad körív pontjait határozza meg. Legyen az aktuális kivilágított pixel P (x_p , y_p), az elméleti körhöz legközelebb esõ pont. A módszer ugyanaz, mint a vonalrajzoló algoritmus esetében: a következõ pixelt két pont (E, SE) közül kell kiválasztani (7. ábra). A kiszámolt körív minden pontjában a kör érintõjének meredeksége -1 és 0 között van. Ezáltal a következõ kirajzolandó pont az ($x_p + 1$, y_p), vagy az ($x_p + 1$, $y_p - 1$) lehet. Jelöljük az E, SE pontok által meghatározott szakasz felezõpontját M-mel. Ha a körív az M pont felett halad el, akkor (a körív megválasztása miatt) az M a kör belsõ pontja, azaz F (M) < 0. Megmutatható, hogy ebben az esetben az E pont van közelebb a körhöz, így ekkor E-t



7. ábra

választjuk, különben az SE pont van közelebb a körhöz és SE-t választjuk. Jelöljük d-vel az F(M) értékét:

$$d = d_{old} = F (M) = F (x_p + 1 , y_p - 1/2) = (x_p + 1)^2 + (y_p - 1/2)^2 - r^2.$$

- Ha $d < 0$, akkor az E-t választjuk, és ekkor:

$d_{new} = F (M_E) = F (x_p + 2 , y_p - 1/2) = (x_p + 2)^2 + (y_p - 1/2)^2 - r^2 = d_{old} + (2x_p + 3)$

lesz a d új értéke, vagyis

$$\Delta_E = d_{new} - d_{old} = 2x_p + 3.$$

- Ha $d \geq 0$, akkor az SE-t választjuk, és ekkor:

$$d_{new} = F (M_{SE}) = F (x_p + 2 , y_p - 3/2) = (x_p + 2)^2 + (y_p - 3/2)^2 - r^2 =$$
$$= d_{old} + (2 (x_p - y_p) + 5)$$

vagyis

$$\Delta_{SE} = 2 (x_p - y_p) + 5.$$

Vegyük észre, hogy míg az egyenes rajzolásánál a Δ_E, Δ_{NE} elsõrendû differenciák konstans értékek voltak, most a Δ_E és Δ_{SE} az x_p, y_p lineáris függvénye. Ez azt jelenti, hogy minden egyes lépésben a Δ_E és Δ_{SE} értékeket (még az aktuális pont koordinátái alapján) újra kell számolni. Szerencsére a kifejezések egyszerûek és egész aritmetikában számolhatók.

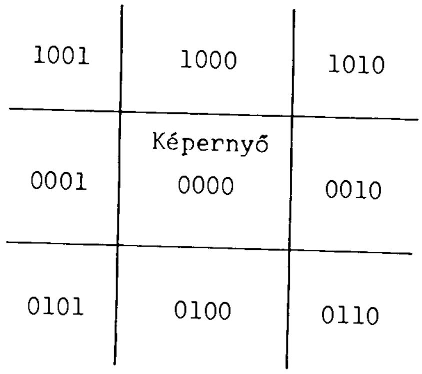
Mielõtt belekezdenénk az algoritmus kódjának leírásába, meg kell határoznunk a kezdeti étékeket. Az algoritmus a (0, r) pontból indul, így $d_0 = F (1 , r - 1/2) = 5/4 - r$. Látható, hogy ekkor d_0 nem egész. Ahhoz, hogy egészekkel tudjunk számolni d helyett tekintsük a $d' = d - 1/4$ változót. Így $d'_0 = 1 - r$. Ekkor a $d < 0$ feltétel helyett $d' < -1/4$ feltételt kell vizsgálni, viszont ez is valós aritmetikát feltételez. Mivel d'_0, Δ_E és Δ_{SE} is egészek d' mindig egész lesz, így egyszerûen tekinthetjük a $d' < 0$ feltételt. Az algoritmus a 8. ábrán látható.

```
procedure CircleFirst(u,v,r:integer);
var
  xp,yp,d:integer;
begin
  xp:=0; { kezdõ értékek }
  yp:=r;
  d:=1-r;
  CirclePoints(u,v,xp,yp);
  while yp>xp do begin
    if d<0 then begin { E }
      d:=d+xp*2+3;
      xp:=xp+1;
    end
    else begin { SE }
      d:=d+(xp-yp)*2+5;
      xp:=xp+1;
      yp:=yp-1;
    end;
    CirclePoints(u,v,xp,yp);
  end;
end;
```

8. ábra

/ Cohen Sutherland vagóalgoritmus

Első bit: a pont a bal oldali képernyőéltől balra esik.
Második bit: a pont a jobb oldali éltől jobbra esik.
Harmadik bit: a pont az alsó él alatt van.
Negyedik bit: a pont a felső él felett van.



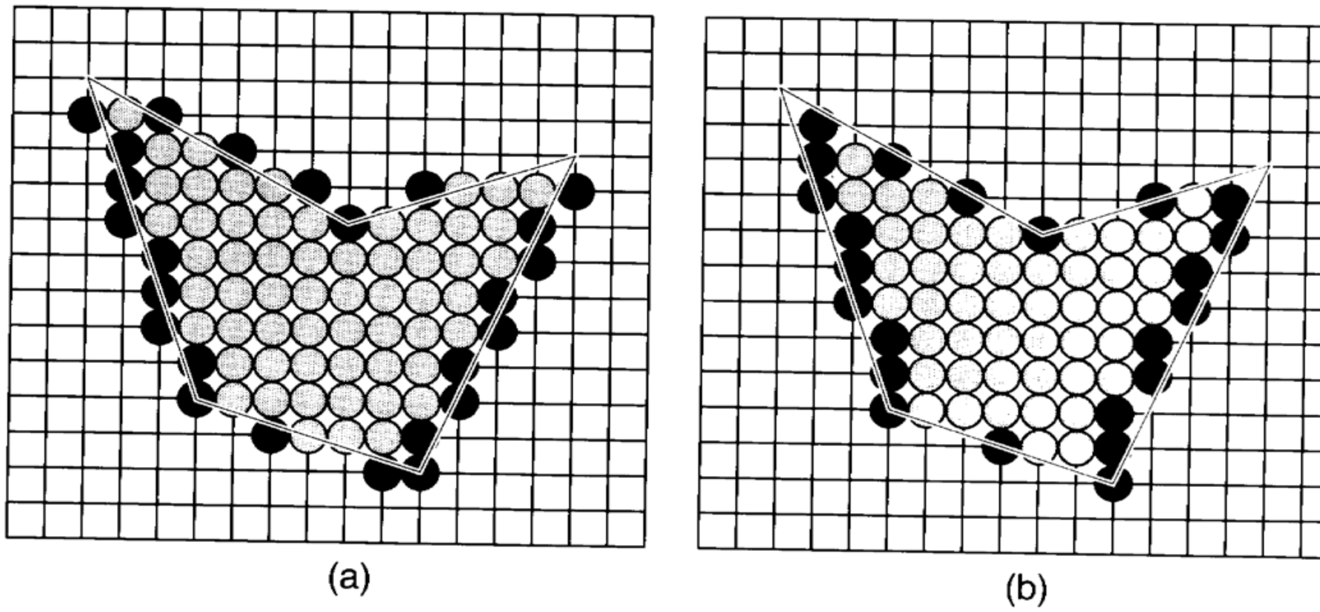
5.5. ábra. A képernyő szélei által meghatározott kilenc tartomány és a hozzájuk rendelt végpontkód

```
//  Vágás téglalapra

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>

struct pont
{
	long x,y,z;
};

long xbal, xjobb, yalso, yfelso;

int kod(long x, long y);
void vagas(pont A, pont B);
pont metsz(pont p, pont q);

void main()
{
	pont a, b;
	int c = 1;
	int gd = DETECT, gm;

	initgraph(&gd, &gm, "");
	xbal = (getmaxx() + 1) / 4;
	xjobb = (getmaxx() + 1) / 4 * 3;
	yalso = (getmaxy() + 1) / 4 * 3;
	yfelso = (getmaxy() + 1) / 4;
	randomize();
  setcolor(LIGHTGRAY);
```

```
        rectangle(xbal, yfelso, xjobb, yalso);
        while (c != 27)
                {
                        a.x =random(getmaxx()) +1;
                        a.y =random(getmaxy()) +1;
                        a.z = 1;
                        b.x =random(getmaxx()) +1;
                        b.y =random(getmaxy()) +1;
                        b.z = 1;
                        setcolor(GREEN);
                        line(a.x, a.y, b.x, b.y);
                        vagas(a, b);
                        c = getch();
                }
        closegraph();
}

int kod(pont A)
{
        int code;
        if (A.x < xbal) code = 8;           //  1000
        else if (A.x > xjobb) code = 4;     //  0100
                else code = 0;              //  0000
        if (A.y > yalso) code |= 2;         //  0010
        else if (A.y < yfelso) code |= 1;   //  0001
                else code;
        return (code);
}

void vagas(pont A, pont B)
{
        int codea, codeb;
        pont ab, xb = {1, 0, -1 * xbal}, xj = {1, 0, -1 * xjobb}, ya = {0, 1,
-1 * yalso}, yf = {0, 1, -1 * yfelso};

        ab = metsz(A, B);
        codea = kod(A);
        codeb = kod(B);
        setcolor(WHITE);
        if (codea == codeb  && !codea)
                {
                        line(A.x, A.y, B.x, B.y);
                        return;
                }
        if (codea == codeb || (codea & codeb)) return;
        while (codea || codeb)
                {
                        if (codea)
```

```
                {
                        if (codea & 8) A = metsz(ab, xb);
                        else if (codea & 4) A = metsz(ab, xj);
                                else if (codea & 2) A = metsz(ab,
ya);
                                        else if (codea & 1) A =
metsz(ab, yf);
                        A.x /= A.z;
                        A.y /= A.z;
                        A.z /= A.z;
                        codea = kod(A);
                }
        else if (codeb)
                {
                        if (codeb & 8) B = metsz(ab, xb);
                        else if (codeb & 4) B = metsz(ab, xj);
                                else if (codeb & 2) B = metsz(ab,
ya);
                                        else if (codeb & 1) B =
metsz(ab, yf);
                        B.x /= B.z;
                        B.y /= B.z;
                        B.z /= B.z;
                        codeb = kod(B);
                }
        if (codea == codeb  && !codea)
                {
                        line(A.x, A.y, B.x, B.y);
                        return;
                }
        if (codea == codeb || (codea & codeb)) return;
    }
}

pont metsz(pont p, pont q)
{
    pont r;
    r.x = p.y * q.z - p.z * q.y;
    r.y = p.z * q.x - p.x * q.z;
    r.z = p.x * q.y - p.y * q.x;
    return (r);
}
```

KITÖLTÉS



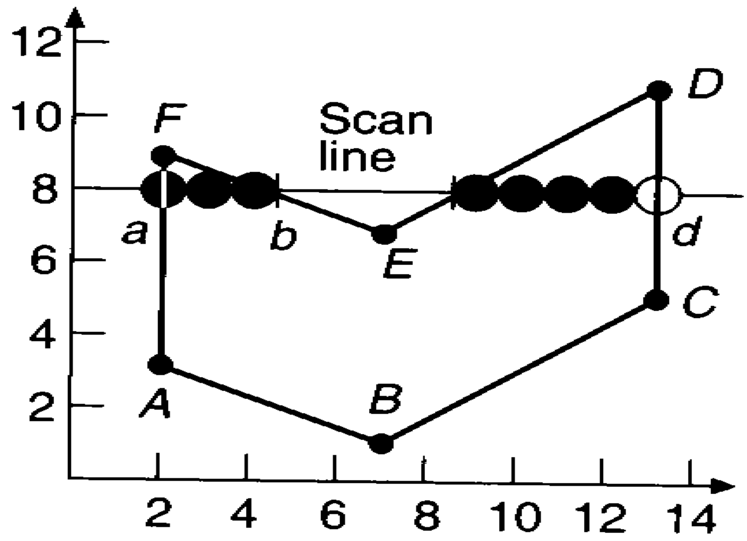
● SZÉLSŐ PIXEL ○ TOVÁBBI PIXELEK

A kitöltési algoritmus lépései:

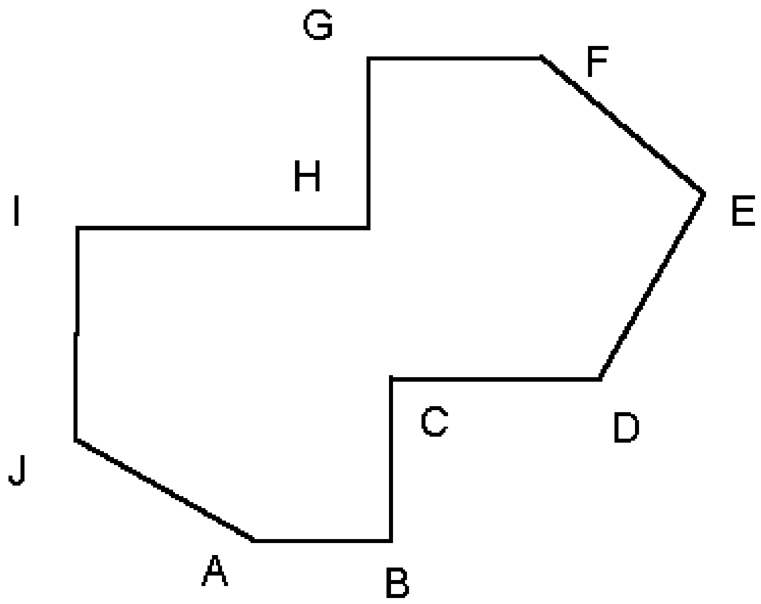
1. A scan-line metszéspontjainak meghatározása a polygon minden élével.
2. A metszéspontok rendezése x kordináta szerint.
3. Azon pixelek kitöltése a metszéspontok között, melyek a poligon belsejében fekszenek, használva egy paritás bitet.

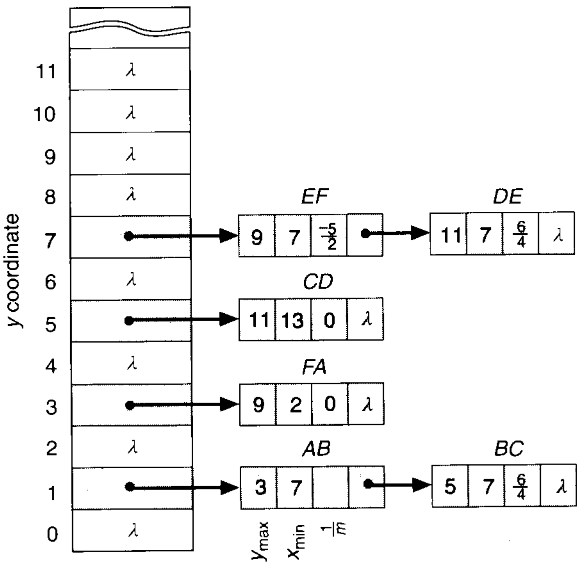
Problémák:

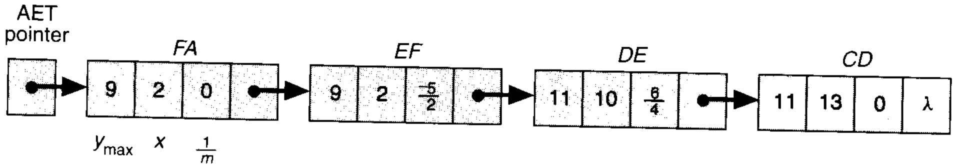
3.1 Nem egész kordinátáju metszéspont esetén hogyan állapítható meg, hogy melyik oldalon lévő pixel tartozik a poligon belsejébe?
3.2 Hogyan kezelhetők az egész koordinátáju metszéspontok?
3.3 Hogyan kezelhetőek a 3.2 beli pontok vizszintes él esetén?
3.4 Hogyan kezelhetőek a 3.2 beli pontok közös él esetén?



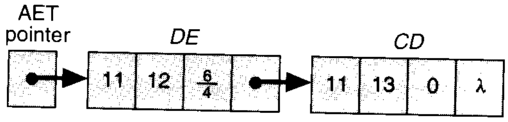
VIZSZINTES ÉLEK






AET
pointer
FA
9
2
0
EF
9
2
$\frac{-5}{2}$
DE
11
10
$\frac{6}{4}$
CD
11
13
0
λ
y_{max}
x
$\frac{1}{m}$

(a)


AET
pointer
DE
11
12
$\frac{6}{4}$
CD
11
13
0
λ

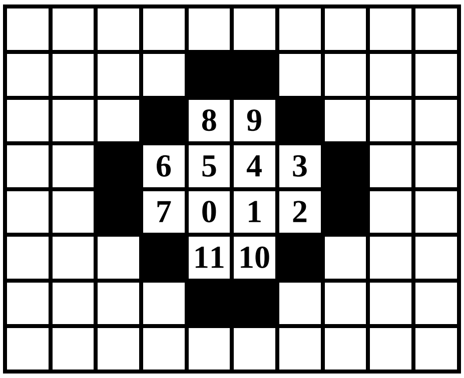
(b)

Él flag módszer

```
……..
for y:=ymin to ymax
        for x=xmin to xmax do
                begin
                        if ( getpixel(x,y)=hatarszin ) flag:=!flag;
                        if (flag) putpixel(x,y,szin);
                end;
……..
```

Rekurzív módszer

```
flood_fill(x,y)
if (getpixe(x,y) == hatterszin
        begin
                putpixel(x,y,szin)
                flood_fill(x+1,y);
                flood_fill(x-1,y);
                flood_fill(x,y+1);
                flood_fill(x,y-1);
        end;

flood_fill(cim)
if (read_pixel(cim) == hatterszin
        begin
                write_pixel(cim,szin)
                flood_fill(cim+1);
                flood_fill(cim-1);
                flood_fill(cim+M);
                flood_fill(cim-M);
        end;
```



Transzformációk

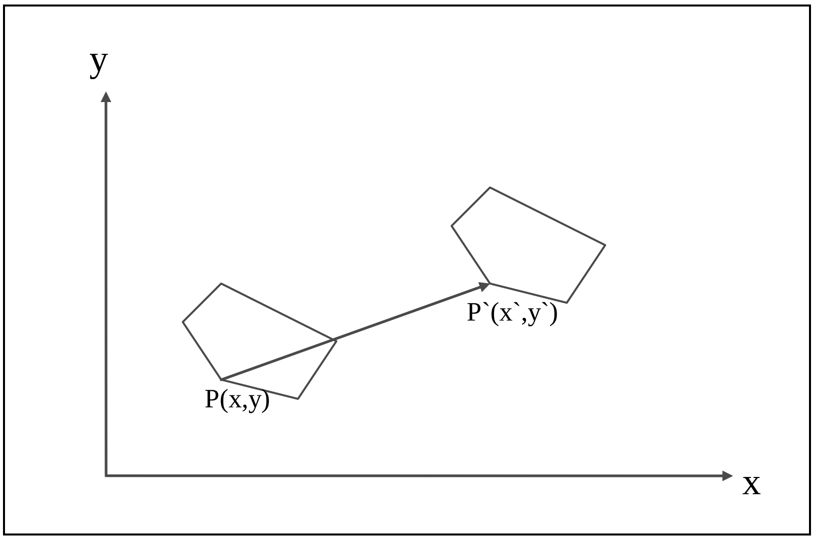
Eltolás

$$x' = x + d_x, \quad y' = y + d_y$$

mátrix alakban:

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \ \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \ \mathbf{T} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{P} + \mathbf{T}$$



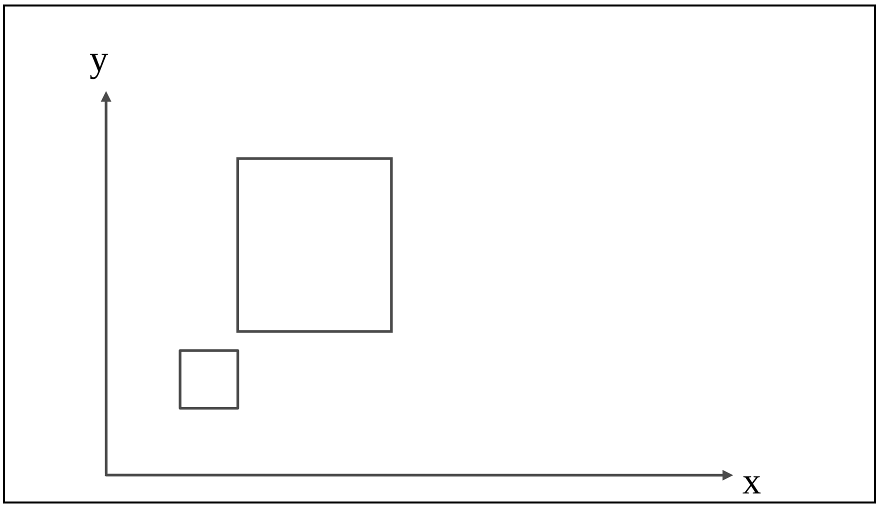
Skálázás

$$x' = s_x x, \quad y' = s_y y$$

mátrix alakban:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$



Forgatás:

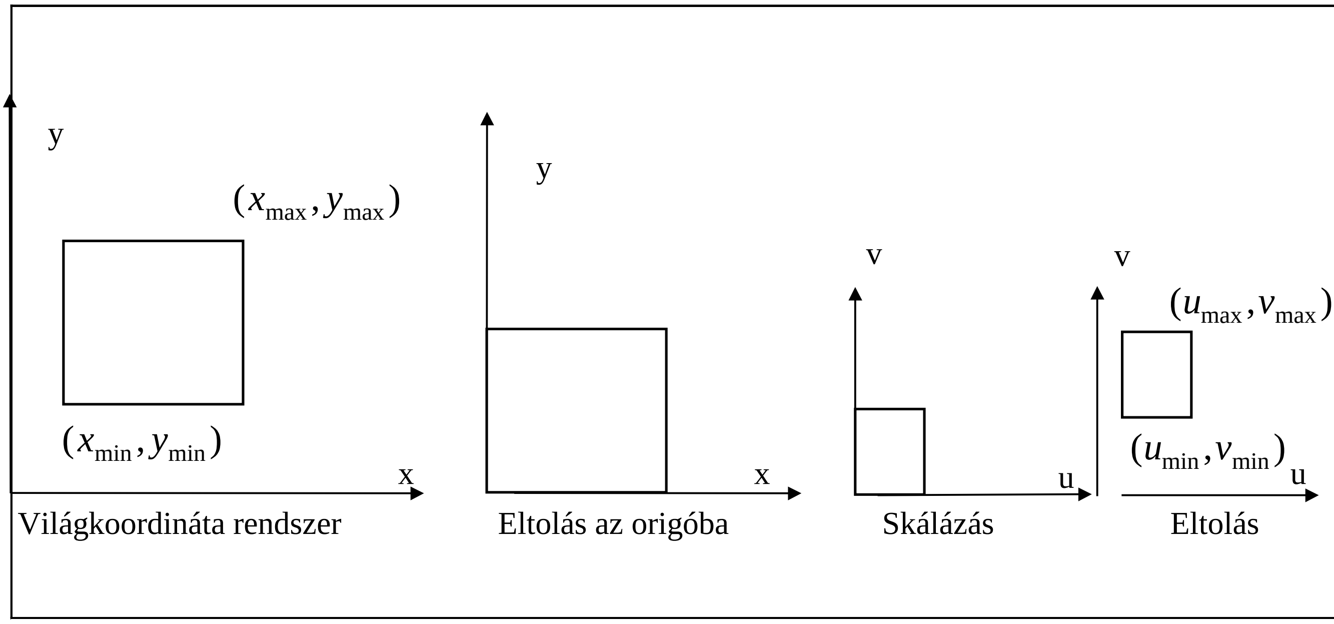
$$x' = x \cdot \cos(\theta) - y \cdot \sin(\theta), \quad y' = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

mátrix alakban:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$

Window to Viewport transzformáció {swinview.pas}



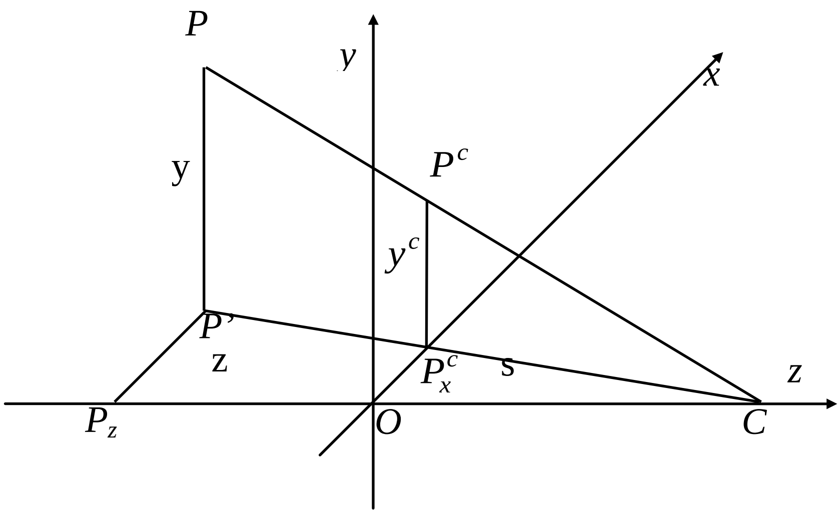
$$\mathbf{M}_{wv} = \mathbf{T}(\mathbf{u}_{min}, \mathbf{v}_{min}) \cdot \mathbf{S}\left(\frac{\mathbf{u}_{max} - \mathbf{u}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}}, \frac{\mathbf{v}_{max} - \mathbf{v}_{min}}{\mathbf{y}_{max} - \mathbf{y}_{min}}\right) \cdot \mathbf{T}(-\mathbf{x}_{min}, -\mathbf{y}_{min}) =$$

$$\begin{pmatrix} 1 & 0 & \mathbf{u}_{min} \\ 0 & 1 & \mathbf{v}_{min} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{\mathbf{u}_{max} - \mathbf{u}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}} & 0 & 0 \\ 0 & \frac{\mathbf{v}_{max} - \mathbf{v}_{min}}{\mathbf{y}_{max} - \mathbf{y}_{min}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -\mathbf{x}_{min} \\ 0 & 1 & -\mathbf{y}_{min} \\ 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} \frac{\mathbf{u}_{max} - \mathbf{u}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}} & 0 & -\mathbf{x}_{min} \cdot \frac{\mathbf{u}_{max} - \mathbf{u}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}} + \mathbf{u}_{min} \\ 0 & \frac{\mathbf{v}_{max} - \mathbf{v}_{min}}{\mathbf{y}_{max} - \mathbf{y}_{min}} & -\mathbf{y}_{min} \cdot \frac{\mathbf{v}_{max} - \mathbf{v}_{min}}{\mathbf{y}_{max} - \mathbf{y}_{min}} + \mathbf{v}_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P}' = \left((\mathbf{x} - \mathbf{x}_{min}) \cdot \frac{\mathbf{u}_{max} - \mathbf{u}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}} + \mathbf{u}_{min} \;,\; (\mathbf{y} - \mathbf{y}_{min}) \cdot \frac{\mathbf{v}_{max} - \mathbf{v}_{min}}{\mathbf{y}_{max} - \mathbf{y}_{min}} + \mathbf{v}_{min} \;,\; 1 \right)$$

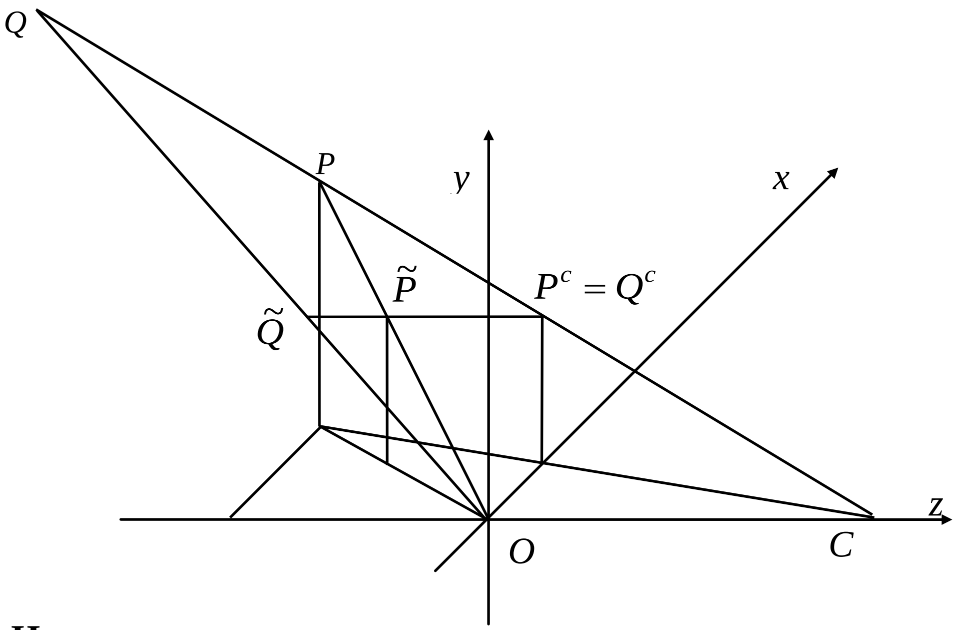
Centrális vetítés



$$\frac{x^c}{x} = \frac{s}{s-z}; \quad \frac{s}{s-z} = \frac{P_x^c C}{P'C}; \quad \frac{P_x^c C}{P'C} = \frac{y^c}{y}$$

$$x^c = x \cdot \frac{s}{s-z}; \quad y^c = y \cdot \frac{s}{s-z}; \quad z^c = 0$$

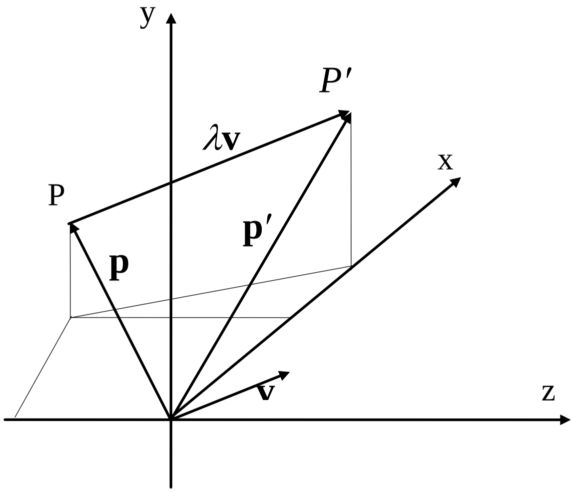
Centrális vetület előállítása merőleges vetületként

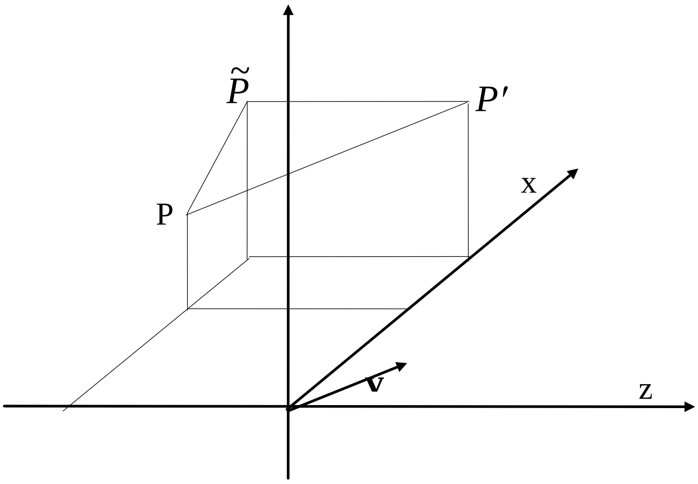


Ha z≠s

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{s} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1-\frac{z}{s} \end{pmatrix} = \begin{pmatrix} x\frac{s}{s-z} \\ y\frac{s}{s-z} \\ z\frac{s}{s-z} \\ 1 \end{pmatrix}; P \to P^c: \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{s} & 1 \end{pmatrix}$$

PÁRHUZAMOS VETÍTÉS





$$\mathbf{p}' = \mathbf{p} + \lambda \cdot \mathbf{v} \quad \lambda \in R$$

$$x' = x + \lambda v_x = x - \frac{v_x}{v_z} z,$$

$$y' = y + \lambda v_y = y - \frac{v_y}{v_z} z$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\tilde{x} = x - \frac{v_x}{v_z} z = x',$$

$$\tilde{y} = y - \frac{v_y}{v_z} z = y',$$

$$\tilde{z} = z$$

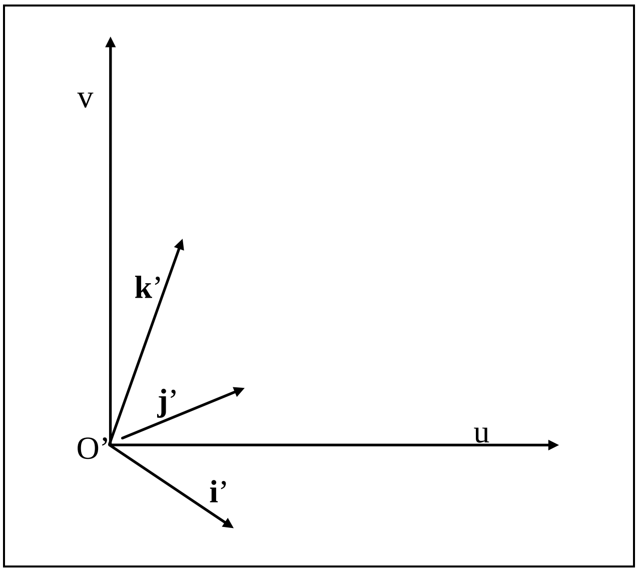
$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

AXONOMETRIA

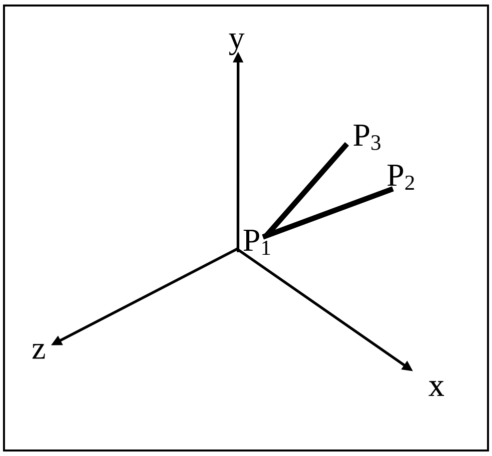
$$\mathbf{p} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

$$\mathbf{p}' = x\mathbf{i}' + y\mathbf{j}' + z\mathbf{k}'$$

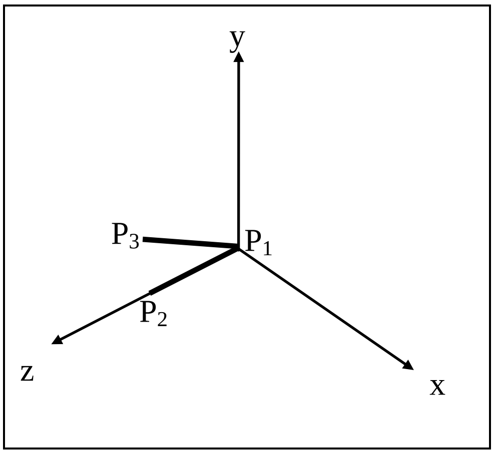
$$\begin{bmatrix} u \\ v \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{i}'_u & \mathbf{j}'_u & \mathbf{k}'_u \\ \mathbf{i}'_v & \mathbf{j}'_v & \mathbf{k}'_v \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



PONTTRANSZFORMÁCIÓK SZORZATA



Kiindulási pozició



Végső pozició

Lépések:

1. P_1 eltolása az origóba
2. P_1P_2 beforgatása y körül az $\{y,z\}$ koordináta síkba
3. P_1P_2 forgatása x körül a z tengelyre
4. P_1P_2 forgatása z körül az $\{y,z\}$ síkba

1.

$$\mathbf{T}(-x_1,-y_1,-z_1)=\begin{pmatrix}1 & 0 & 0 & -x_1\\ 0 & 1 & 0 & -y_1\\ 0 & 0 & 1 & -z_1\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$P_1'=\mathbf{T}(-x_1,-y_1,-z_1)\,P_1=\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\quad P_2'=\mathbf{T}(-x_1,-y_1,-z_1)\,P_2=\begin{pmatrix}x_2-x_1\\ y_2-y_1\\ z_2-z_1\\ 1\end{pmatrix}$$

$$P_3'=\mathbf{T}(-x_1,-y_1,-z_1)\,P_3=\begin{pmatrix}x_3-x_1\\ y_3-y_1\\ z_3-z_1\\ 1\end{pmatrix}$$

2.

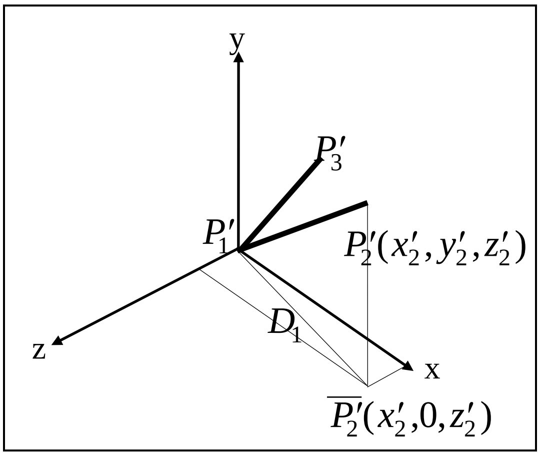
$$\text{a forgatás szöge:} -(90 - \theta) = \theta - 90$$

$$\cos(\theta - 90) = \sin\theta = \frac{z_2'}{D_1} = \frac{z_2 - z_1}{D_1}$$

$$\sin(\theta - 90) = -\cos\theta = -\frac{x_2'}{D_1} = \frac{x_2 - x_1}{D_1}, \text{ahol}$$

$$D_1 = \sqrt{(z_2')^2 + (x_2')^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2}$$

$$P'' = \mathbf{R}_y(\theta - 90) \cdot P_2' = \left[0 \; y_2 - y_1 \;\; D_1 \;\; 1\right]^T$$



3.

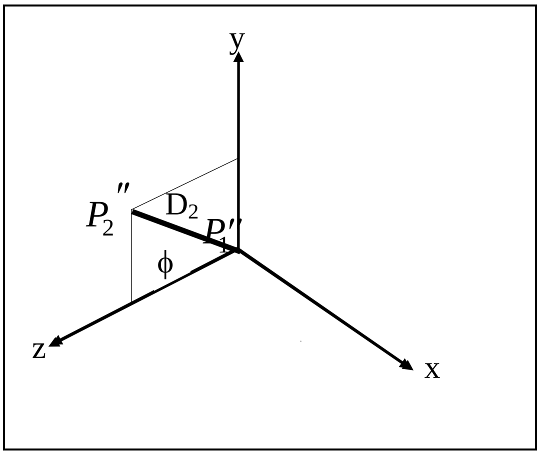
$$\text{a forgatás szöge:} \phi$$

$$\cos\phi = \frac{z_2''}{D_2}, \quad \sin\phi = \frac{y_2''}{D_2}$$

$$D_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

$$P_2''' = \mathbf{R}_x(\phi) \cdot P_2'' = \mathbf{R}_x(\phi) \cdot \mathbf{R}_y(\theta - 90) \cdot P_2' =$$

$$\mathbf{R}_x(\phi) \cdot \mathbf{R}_y(\theta - 90) \cdot T \cdot P_2 = \left[0 \;\; 0 \;\; D_2 \;\; 1\right]^T$$



4.

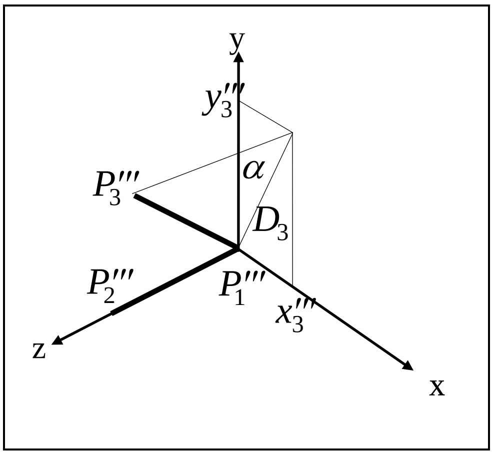
$$\text{a forgatás szöge:} \alpha$$

$$P_3''' = \left[x_3''' \; y_3''' \; z_3''' \; 1\right]^T =$$

$$\mathbf{R}_x(\phi) \cdot \mathbf{R}_y(\theta - 90) \cdot \mathbf{T}\left(-x_1, -y_1, -z_1\right) \cdot P_3$$

$$\cos\alpha = \frac{y_3'''}{D_3}, \quad \sin\alpha = \frac{y_3'''}{D_3} \quad D_3 = \sqrt{(x_3''')^2 + (y_3''')^2}$$

$$\mathbf{R}_z(\alpha) \cdot \mathbf{R}_x(\phi) \cdot \mathbf{R}_y(\theta - 90) \cdot \mathbf{T}\left(-x_1, -y_1, -z_1\right)$$



KOORDINÁTA TRANSZFORMÁCIÓ A TÉRBEN

$$\mathbf{p} = \mathbf{d} + \mathbf{p}' = x'\mathbf{i}' + y'\mathbf{j}' + z'\mathbf{k}' + \mathbf{d}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{i}' & \mathbf{j}' & \mathbf{k}' & \mathbf{d} \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & d_x \\ i'_y & j'_y & k'_y & d_y \\ i'_z & j'_z & k'_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{p} - \mathbf{d}$$

$$x' = \mathbf{p}'\mathbf{i}' = (\mathbf{p} - \mathbf{d})\mathbf{i}' = \mathbf{p}\mathbf{i}' - \mathbf{d}\mathbf{i}',$$

$$y' = \mathbf{p}'\mathbf{j}' = (\mathbf{p} - \mathbf{d})\mathbf{j}' = \mathbf{p}\mathbf{j}' - \mathbf{d}\mathbf{j}',$$

$$z' = \mathbf{p}'\mathbf{k}' = (\mathbf{p} - \mathbf{d})\mathbf{k}' = \mathbf{p}\mathbf{k}' - \mathbf{d}\mathbf{k}'$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & i'_y & i'_z & -\mathbf{d}\mathbf{i}' \\ j'_x & j'_y & j'_z & -\mathbf{d}\mathbf{j}' \\ k'_x & k'_y & k'_z & -\mathbf{d}\mathbf{k}' \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$