Schwarcz Tibor

BEVEZETÉS A SZÁMÍTÓGÉPI GRAFIKÁBA

mobiDIÁK könyvtár

Schwarcz Tibor

BEVEZETÉS A SZÁMÍTÓGÉPI GRAFIKÁBA

mobiDIÁK könyvtár

SOROZATSZERKESZTŐ

Fazekas István

Schwarcz Tibor

BEVEZETÉS A SZÁMÍTÓGÉPI GRAFIKÁBA

Jegyzet

Első kiadás

mobiDIÁK könyvtár
Debreceni Egyetem

Lektor: Dr. Kovács Emőd

mobiDIÁK könyvtár

Debreceni Egyetem

Informatikai Intézet

4010 Debrecen, Pf. 12.

Hungary

http://mobidiak.inf.unideb.hu/

A mű „A mobiDIÁK önszervező mobil portál” (IKTA, OMFB-00373/2003) és a „GNU Iterátor, a legújabb generációs portál szoftver” (ITEM, 50/2003) projektek keretében készült.

1 Történeti áttekintés, grafikus hardwe eszközök, szabványok

1.1 A számítógépi grafika történetének néhány kiemelkedő eseménye:

- 1950. A Whirlwind Computer által kifejlesztett első valósidejű grafikus megjelenítő
- 1963. IVAN E. SUTHERLAND 'Sketchpad' rajzoló rendszerének kifejlesztése. Az első „on line” működő grafikus rendszer.
- 1964. GM DAC rendszer-az első grafikus konzol, grafikus parancsokat is értelmező rendszer. Fénytoll bevezetése.
- 1964 Az első CAD rendszer kifejlesztése az IBM és GM közös projektjében.
- 1965 Az egér bevezetése (fából és műanyagból készítette Douglas Engelbart.)

- 1973 Sharp (Japán) kifejlesztetee az LCD (Liquid Crystal Display) monitort. 20 év kellett az elterjedéséhez.
- 1974 A Phillips cég első videó-telefonja.
- 1975 Az első fraktállal kapcsolatos publikációja Mandelbrotnak.

- 1977 A személy számítógépek korszakának kezdete. Megalakul a Microsoft cég.
- 1980 A PC-k elterjedése beépített raszter grafikával (IBM,APPLE) bit-térképek (pixel vagy pel), desktop-felület, window manager elterjedése.

1.2 Néhány lehetséges felhasználói terület:

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

1.3 Grafikus output eszközök:

1.3.1 Monitorok

A monitorok működési elv szerint lehetnek

- raszteres
- vonalrajzoló monitorok.

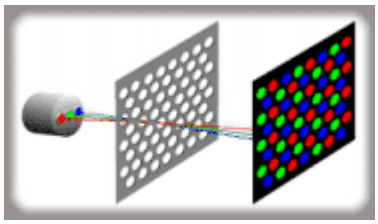
1.3.1.1 A monitorok által alkotott kép minőséget meghatározó legfontosabb műszaki paraméterek

- a felbontás megadja a raszteres képen megjeleníthető pixelek számat. Pl.: 800*600, 1024*768, 1280*1024, stb.
- képátmérő: pl.: 14, 17, 21 inch, stb.
- kép-pontátmérő: a képernyőn beszínezhető pixelek nagyságát adja meg.
- a videosáv-szélesség az elektronika állapotváltozásainak maximális számát adja meg másodpercenként, és így meghatározza a másodpercenként kirajzolható pixelek számát is.
- a képfrissítési frekvencia megadja a másodpercenként kirajzolt teljes képernyők számát.
- képkirajzolási üzemmód szerint interlace vagy noninterlace. Az előbbi frissítési lépésenként a pixelsorok közül egyszerre csak a páros majd a páratlan sorokat rajzolja ki, az utóbbi folyamatosan rajzolja ki a sorokat.

1.3.1.2 A raszteres display működési elve

Egyik fajtája a katódsugárcsöves képernyő. Ebben a képernyő belső oldalán van egy foszforréteg, amit a megfelelő helyen elektronágyú meglő, és az ott felvillanó

fényfolt lesz a pixel. Az elektronágyúk elektronsugarat bocsátanak ki, amelyet vízszintes és függőleges irányban elektromágnesek segítségével eltérítenek.Színes képernyő esetén 3 alapszínből épül fel egy pixel. (RGB)



Az LCD (Liquid Crystal Display, folyadékkristályos kijelző) monitorok. Működési elvük: a hátulról érkező fehér fényt az adott képpontban elhelyezkedő olyan szerkezet ereszti át vagy zárja el, amely a fényáteresztő képességét az áramerősség függvényében változtja

1.3.1.3 Monitor és videokártya típusok

Legfontosabb jellemzője a frissítési frekvencia, szinmélység, felbontás, melyek nem független adatok.

Egy pixel szininformációjának tárolására rendelkezésre álló bitek száma határozza meg a színek számát, azaz a rendelkezésre álló videomemória nagysága határozza meg a maximális felbontást is.

1 bit, 2 szín

4 bit 16 szín

8 bit, 256 szín

16 bit, 65536 szín = High color

24 bit, 16 millió szín. = True Color

Időrendi kialakulás:

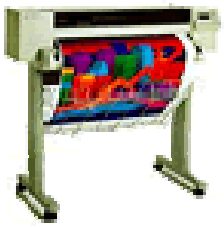
1980-ban CGA : 320 * 200-as felbontásnál 4 szín , 640*200-as felbontásnál 2 szín

- 1984-ban MDA/HERKULES : 720*348-as felbontásnál 2 szín
- 1984-ban EGA : 640*350-as felbontásnál 16 szín
- 1987-ban VGA : 640*480-as felbontásnál 16 szín, 320*200-as felbontásnál 256 szín
- 1990-ban SVGA VESA szabvány

640*480	256 szín
800*600	32K
1024*768	64K 164 ezer szín
1280*1024	16M 16 millió szín
1600*1200	TRUE COLOR

1.3.2 Rajzgépek

A rajzgépek egy íróhegyet vezetnek a papíron. A rajz a toll két egymásra merőleges, X-Y irányú mozgásának eredőjeként jön létre. Síkplotterek esetében a rajzlapot egy táblán rögzítik, mely fölött az írócsúcs két, egymásra merőleges irányban mozog. A görgős papírmozgatású rajzgépnél a toll csak egy irányban mozog, a rá merőleges irányú vezérlést görgők végzik, behúzva a rajzlapot a megfelelő helyzetbe.



Egy plottert a következő jellemzők alapján minősíthetünk:

- befogható rajzlap mérete

- rajzlap anyaga (papír, film, pausz, műanyag)

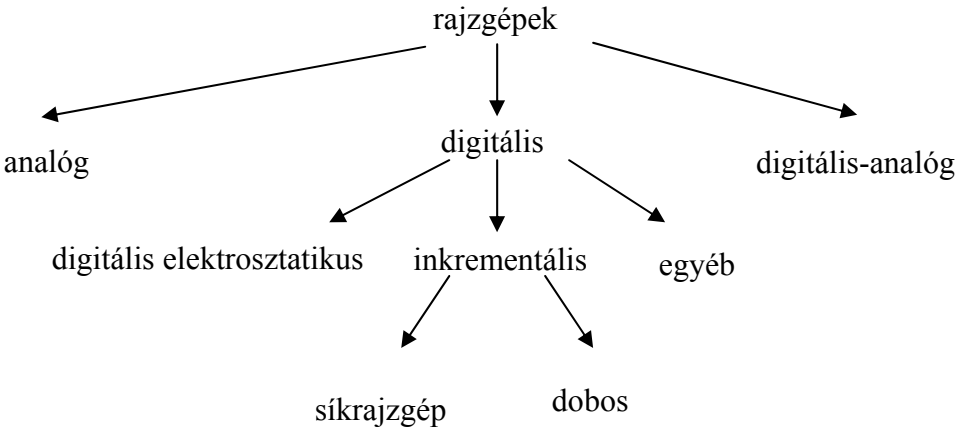
- tollak száma (színek, különböző vonalvastagságok változtathatósága)

- használható tollfajta (meghatározza a rajz minőségét; lehet: tustoll, golyóstoll, rostirón, kerámiahegyű toll);

- gyorsulás (a toll nyugalmi helyzetből indulva mennyi idő alatt éri el a maximális sebességét);

- tengelyirányú tollsebesség (ez a toll maximális rajzolási sebessége);

- pontosság ;



1.3.3 Egyéb grafikus eszközök:

fénytoll

egér

pozícionáló-gomb

scanner

1.4 A legfontosabb grafikus szabványok:

3D Core Graphics System –alacsony szintű, eszközfüggetlen grafikus rendszer (1977)

SRGP-Simple Raster Graphics Package

GKS: Graphical Kernel System -2D az első hivatalos grafikai szabvány (1985)

GKS 3D kiterjesztése (1988)

PHIGS Programmer's Hierarchical Interactive Graphic System (1988)

PHIGS PLUS (ANSI/ISO 1992)

1.4.1 Rasztergrafikus szabvány (SRGP)

Ezt a szabványt egyszerű raszter-grafikus programcsomagnak, vagy az angol megnevezése után rövidítve SRGP-nek (Simple Raster Graphic Package) nevezik. Az SRGP tulajdonképpen a legalapvetőbb raszter-grafikus funkciókra, megvalósító programegységekre vonatkozó szabvány. Az SRGP legfontosabb alkotóelemei a raszter-grafikus primitívek, melyek vonalak, ellipszisek, sokszögek és szövegek lehetnek. A primitívek tulajdonképpen képelem generátorok, melyeket felhasználásukkor kell felparaméterezni (például kör esetén a középpont és sugár megadásával). Az SRGP egyes program-realizációi a szokásos raszter-grafikus funkcionalitást biztosítják.

A Borland cég úgynevezett BGI grafikájának felépítése:

- Primitívek:
- egyenesek,
- poligonok,
- körök,
- ellipszisek,
- szöveg

Konstansok és típusok

Konstansok

- Bar3D Constants
- BitBlt Operators
- Clipping Constants
- Color Constants Graphics
- Colors for the 8514
- Fill Pattern Constants
- Graphics Drivers
- Graphics Modes for Each Driver
- Justification Constants
- Line-Style and Width Constants
- Text-Style Constants

Típusok

- ArcCoordsType
- FillPatternType
- FillSettingsType
- Memory Pointers
- LineSettingsType
- PaletteType
- PointType
- TextSettingsTyp
- ViewPortType

ATTRIBÚTUMOK

Színek:

```
procedure SetColor(Color: Word);
```

Sötét színek:

(Tinta & Papír)

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

Világos színek:

(Tinta)

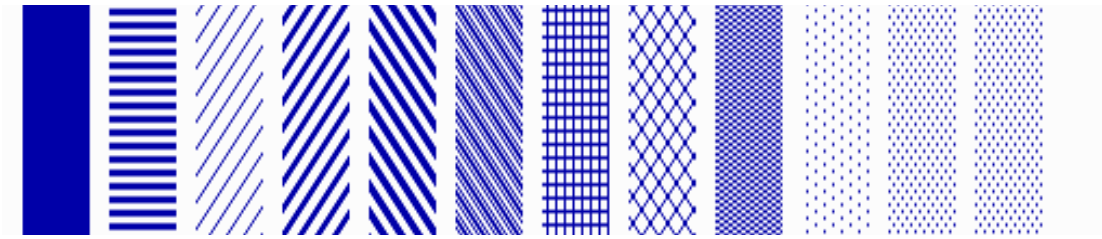
- DarkGray 8
- LightBlue 9
- LightGreen 10
- LightCyan 11
- LightRed 12
- LightMagenta 13
- Yellow 14
- White 15

Kitöltések és minták:

```
Procedure SetFillStyle(Pattern: Word; Color: Word);
```

Konstans	Érték	Jelentés
• EmptyFill	0	Háttérszín
• SolidFill	1	Tintaszín
• 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:



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

1.4.2 Vektorgrafikus szabványok

1.4.2.1 A GKS és a GKS-3D:

Az első nemzetközi grafikai szabvány a német kezdeményezés alapján kidolgozott grafikus magrendszer (Graphical Kernel system = GKS), mely a 2D-s vektorgrafikára vonatkozott. A célkitűzések és követelmények, melyeket a GKS szabvánnyal el akartak érni a következők voltak:

- egységes illesztési helyet meghatározni a grafikus rendszerek és az egyedi alkalmazások között,
- a felhasználástól független, számítástechnikailag hatékonyan realizálható könyvtár definiálása a vektorgrafikus rendszerek fontosabb funkcióira,
- a grafika területén minél több általánosítható követelményt lefedni a szabvánnyal, elválasztania grafikus rendszerek alap- (ún. mag) funkcióit a magasabb szintű modellezési funkcióktól, a szabvánnyal egy fejlesztési irányt mutatni a készülékgyártók számára.

Az egyre teljesítőképesebb hardverek eredményezték a GKS szabvány továbbfejlesztését. Így jött létre a GKS-3D szabvány, melyben a GKS-t 3D eljárásokkal és funkciókkal bővítették. A GKS szabványt több hardver és operációs rendszer platformon is megvalósították. Ezek számítógépes megjelenési formájukat tekintve legtöbbször egy alprogram könyvtárban öltenek testet. Ezért a grafikus szoftverfejlesztők a GKS-t tulajdonképpen egy felhasználói programozói interfész (APl) formájában látják.

A GKS a GKS-3D funkciókkal kiegészítve a vektorgrafika következő területeit fedi le:

- színkezelés (színpaletta definiálása),
- transzformációk (vetítés, 3D-s ablak definiálás stb.),
- grafikus primitívek (sokszög, kitöltött terület, 2 és 3D-s szöveg stb.),
- koordináta-rendszer, skálázás, rácsok,
- objektumok definiálása (kör, téglatest, ellipszoid stb.),

- térgörbék és felületek definiálása (kontúrvonalak, háromszög közelítésű felületek,
- függvénnyel megadott felület háromszög-közelítéssel stb.),
- láthatósági eljárások (huzalvázas megjelenítés, árnyalt felületek, poligonok rendezése, fedettség szerint stb.).

Ezeket a funkciókat a programozó a GKS könyvtárban lévő programok felhívásával érheti el, mely alprogramokat fordítást követően bekapcsol a programjába.

1.4.2.2 A PHIGS, PHIGS+ és a PEX:

A PHIGS a szabvány angol elnevezésének: Programmers Hierarchical Interactive Graphics System rövidítéséből származik. Ezek szerint a PHIGS programozók hierarchikus, interaktív grafikus rendszere. A PHIGS szabvánnyal főként a tervezőszoftverek egységesítését szerették volna elérni, a norma funkcionalitását tekintve döntően megegyezik a GKS-3D-vel. A lényeges különbségeket a szabvány neve is kifejezi: a PHIGS támogatja a vektorgrafikus objektumok közötti hierarchikus kapcsolatok felépítését, ezeket egy modell-adatbankban az ún. CSS-be (Central Structure Storage) a központi struktúra-tárolóba integrálja. Az adatbázis-szerkezet a szabványban úgy került megfogalmazásra, hogy a programok a modelltér műveleteket interaktív módon is képesek legyenek végrehajtani. A PHIGS szabvány - melynek kapcsolata a legfontosabb programnyelvekkel, így például FORTRAN, ADA, C is kidolgozásra került - a programozó számára egy hatékony eszközt biztosított a 3D-s objektumok adatbázisban való feldolgozásához. Az 1990-es évek elejére viszont a grafikus hardver teljesítményének növekedése és a realisztikus képelőállítást biztosító renderelés igénye egyaránt szükségessé tette a szabvány továbbfejlesztését. A PHIGS+ szabványt 1992 végén adták ki. Ez már a túl absztrakt huzalvázas megjelenítést meghaladva kitér a különböző megvilágítási és árnyalási modellekre és eljárásokra is, és beépíti a szabványba a térbeli görbék és felületek modellezésének legújabb eredményeit. A PHIGS a PHIGS+ bővítéssel a következő elemekből épül fel:

- A primitívek egyik csoportja geometriai, ide tartoznak a szokásos poligonok, poliéderek kitöltött felületek, NURBS stb. Emellett a szabvány ismeri a szöveges és raszteres primitívek mellett a különböző mennyiségi (matematikai, statisztikai) primitíveket és a speciális szervezési primitíveket is (például egy úthálózat gráfja). A primitívekhez egyedi és csoporttulajdonságok is hozzárendelhetők, például színmodellek.
- A primitívekből tárgymodelleket állíthatunk össze és ezeket struktúrákba szervezhetjük. A CSS lehetővé teszi a primitívek, tárgymodellek és struktúrák adatbázisszerű feldolgozását.
- A CSS objektumaira a szokásos adatbázis-műveletek (visszakeresés, létesítés, törlés, csoportosítás) mellett végrehajthatjuk a vektorgrafikában szokásos transzformációkat is (skálázás, mozgatás, koordináta-rendszer választása stb.).
- A modelltér jeleneteit különböző nézőpontokból ábrázolhatjuk, ehhez a szabvány biztosítja a szükséges 3D-2D-s vetítéseket és kivágást.
- A megjelenítéshez a PHIGS ismeri a láthatósági és megvilágítási és árnyalási modelleket.
- A felhasználóval való kapcsolattartás eszközeként alkalmazhatók a lokátor eszközök, a poligon rajzolás, a kijelölés, értékadás stb.

A PEX (PHIGS Extension on X) az X-Windows-System bővítése a PHIGS-hez, illetve a PHIGS+ -hoz. Ez egy illesztési helyet ad meg a PHIGS és a XII között, ami lehetővé teszi az X szerver szolgáltatások igénybevételét a PHIGS-en belül. A PEX által támogatott elemek: képernyő, pixeles bitmap-ek, billentyűzet és egér, és CSS.

2 Alapvető rasztergrafikai algoritmusok

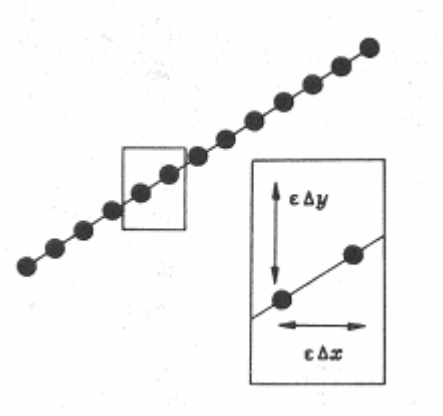
A modellezés során arra a kérdésre keressük a választ, hogy hogyan lehet folytonos geometriai alakzatokat képpontokkal közelíteni. Azokat az algoritmusokat, melyek erre megadják a választ, digitális differenciaelemző (DDA) algoritmusoknak nevezzük

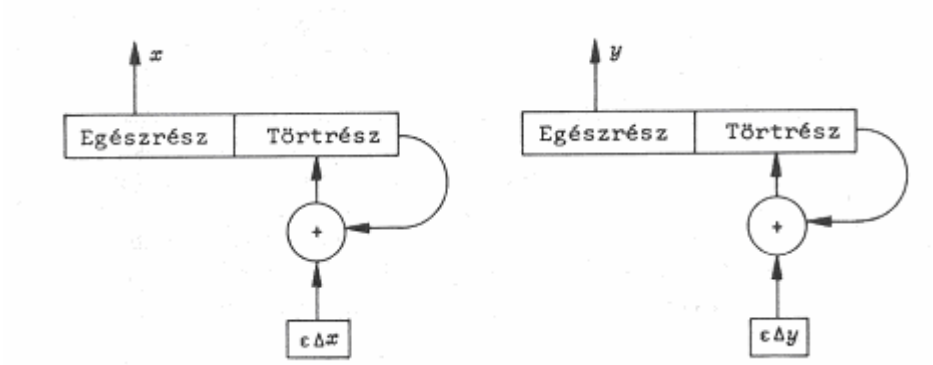
2.1 Egyenes szakasz rajzolása

2.1.1 Szimmetrikus DDA

Az egyenes $\mathbf{r}(t) = \mathbf{r_0} + t\mathbf{v}$ skalár-vektror előállításán alapszik. A szakasz meghatározó $A(x_1, y_1)$ és $B(x_2, y_2)$ pontokból képezzük a $\Delta x = x_2 - x_1$ és $\Delta y = y_2 - y_1$ különbségeket. Meghatározzuk az $\varepsilon = 2^{-n}$ értéket, ahol $2^{n-1} \le \max(|\Delta x|, |\Delta y|) < 2^n$ és egy x és y regiszter

tartalmát növeljük az $\varepsilon\Delta x$ és $\varepsilon\Delta y$ értékekkel. Az egész túlcsordulásoknál megjelenítjük a pontot.





2.1.2 Egyszerű DDA

Ekkor az x vagy y irányban egyesével lépkedünk, azaz a $\varepsilon\Delta x = 1$ vagy $\varepsilon\Delta y = 1$

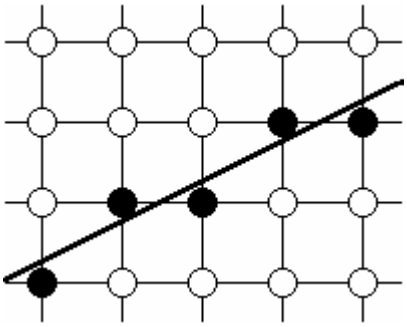
Csak egy regisztert használunk, és egész túlcsordulás esetén lépünk a megfelelő irányba.

2.1.3 Midpoint algoritmus

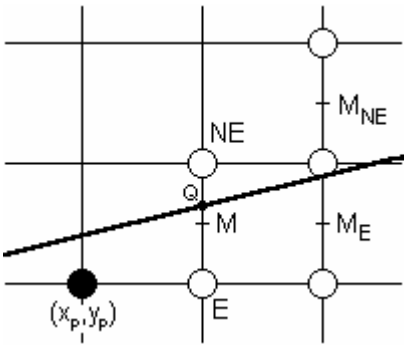
Bárhogyan is származtassuk az egyenest, az egyenlete:

$$ax+by+c=0$$

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



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<=m<=1. Az egyenest balról jobbra haladva rajzoljuk ki. A kitöltött körök a megvilágított pixeleket jelentik.



Legyen az éppen megjelenített pont $P(x_p,y_p)$, ekkor a következő 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 elhelyezkedő felezőpont (M) segítségével történik. Ha az egyenes az M pont felett halad el, akkor az NE pontot jelenítjük meg, különben az E-t. Az M pont helyzetét analitikusan határozzuk meg. Az egyenes egyenletét az $F(x,y) = ax + by + c = 0$ formában tekintjük, ahol

$$a = (y_2 - y_1),$$
$$b = -(x_2 - x_1), \text{ és}$$
$$c = (y_2 - y_1)x_1 - (x_2 - x_1)y_1$$

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:

$$d = F(M) = F(x_p + 1, y_p + \frac{1}{2}) = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

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 a régi értékéből számoljuk ki. Jelölje ezt d_{old}, az új érteket d_{new}. Ekkor a d_{new} függ az új pont meg választásától. Ha az E pontot választjuk, akkor

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

azaz ekkor d-t $\Delta_E = d_{new} - d_{old} = a$ -val kell növelni. Ha az NE pontot választjuk, akkor

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

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 meg kell határoznunk 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:

$$d_0 = F(x_1 + 1, y_1 + \frac{1}{2}) = ax_1 + by_1 + c + a + \frac{b}{2} = F(x_1, y_1) + a + \frac{b}{2},$$

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:

$$F(x, y) = 2\left(ax + by + c\right)$$

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

$$d_0 = 2a + b,$$

$$\Delta_E = 2a, \text{ és}$$

$$\Delta_{NE} = 2(a + b)$$

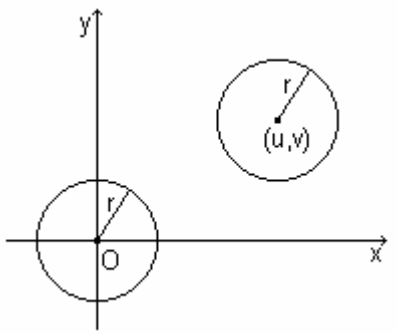
egyenleteket kapjuk, minden más változatlan. Az iterációs lépést addig kell ismételni, amíg az utolsó pontot is ki nem rajzoljuk.

A rutin kódja:

```
procedure Line(x1,y1,x2,y2:integer);
var
  a,b,x,y,d,deltaE,deltaNE:integer;
begin
  a:=y1-y2;
  b:=x2-x1;
  d:=2*a+b; { d kezdőértéke }
  deltaE:=2*a; { d növekménye E esetén }
  deltaNE:=2*(a+b); { és NE esetén }
  x:=x1; { a kezdőpont }
  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;
```

2.2 Kör rajzolása

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



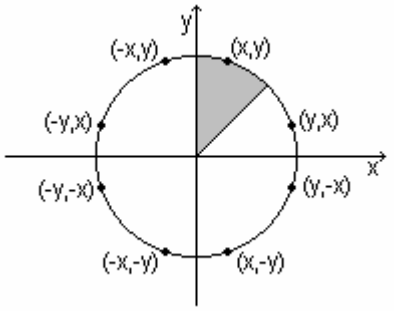
Ha rögzítünk egy [x, y] koordináta-rendszert, akkor az origó középpontú, r sugarú kör egyenlete: $x^2 + y^2 = r^2$. Ebből pedig könnyen levezethető az (u, v) középpontú, r sugarú kör egyenlete: $(x-u)^2 + (y-v)^2 = r^2$. Az egyenletekben xy-os tag nem szerepel, és a négyzetes tagok együtthatója megegyezik. Az utóbbi egyenletet átrendezve a következő összefüggést kapjuk: $F(x,y) = (x-u)^2 + (y-v)^2 - r^2 = 0$. Az $F(x,y)$ függvénybe a körre illeszkedő pontok koordinátáit helyettesítve nulla értéket kapunk. Az (x_1,y_1) pont akkor és csak akkor belső pont, ha $F(x_1,y_1)<0$ és a (x_2,y_2) pont akkor és csakis akkor külső pont, ha $F(x_2,y_2)>0$

2.2.1 Midpoint algoritmus

Szeretnénk meghatározni egy adott (u,v) középpontú és r sugarú kör pontjait. Az egyik lehetséges megoldás, ami eszünkbe juthat a trigonometrikus függvényeket használja az $x = r\cos(t) + u, y = r\sin(t) + v$ összefüggések segítségével határozza meg a kör pontjait. Számunkra ez a módszer most nem megfelelő, mert a trigonometrikus függvények kiszámolása, ami valós aritmetikát követel meg, túlságosan sok időt 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 megfelelő módszereket, nézzünk meg egy igen hatékony algoritmust és annak egy gyorsítását. Ez az algoritmus az egyenes rajzolásnál tárgyalt midpoint algoritmus továbbfejlesztése.

Nyolcas szimmetria elve:

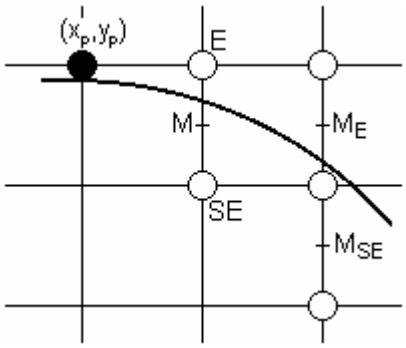
Tekintsünk egy origó középpontú kört. Ha egy (x,y) pont rajta van a körön, akkor könnyen meghatározhatunk három másik pontot, ami szintén rajta van a körön.



Ezért ha meghatározzuk a kör egy megfelelő 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 essen. Ha egy nem origó középpontú kört akarunk rajzolni (az esetek többségében ez teljesül), akkor koordináta transzformációt alkalmazunk. A koordináta-rendszer origóját a kör (u,v) középpontjába visszük. Vagyis a kör pontjait úgy határozzuk meg, mintha a középpontja az origóban lenne, de kirajzolás előtt a pontokat az (u,v) vektorral eltoljuk, s így a kívánt helyzetű kört kapjuk

Elsőrendű differenciák módszere

Az elmondottak alapján a midpoint algoritmus origó középpontú kört feltételez, és csak a kitüntetett nyolcad körív pontjait határozza meg.



Legyen az aktuális kivilágított pixel $P(x_p,y_p)$, az elméleti körhöz legközelebb eső pont. A módszer ugyanaz, mint a vonalrajzoló algoritmus esetében: a következő pixelt két pont (E,SE) közül kell kiválasztani. A kiszámolt körív minden pontjában a kör érintőjének meredeksége -1 és 0 között van. Ezáltal a következő 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 felezőpontját M-mel.

Ha a körív az M pont felett halad el, akkor (a körív megválasztása miatt) az M a kör belső pontja, azaz $F(M)<0$. Megmutatható, hogy ebben az esetben az E pont van közelebb a körhöz, így ekkor E-t választjuk, különben az SE pont van közelebb a körhöz és SE-t választjuk. Jelöljük d-vel az F(M) értékét:

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

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

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

lesz a *d* új értéke, vagyis

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

- Ha *d>=0*, akkor az SE-t választjuk, és ekkor

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

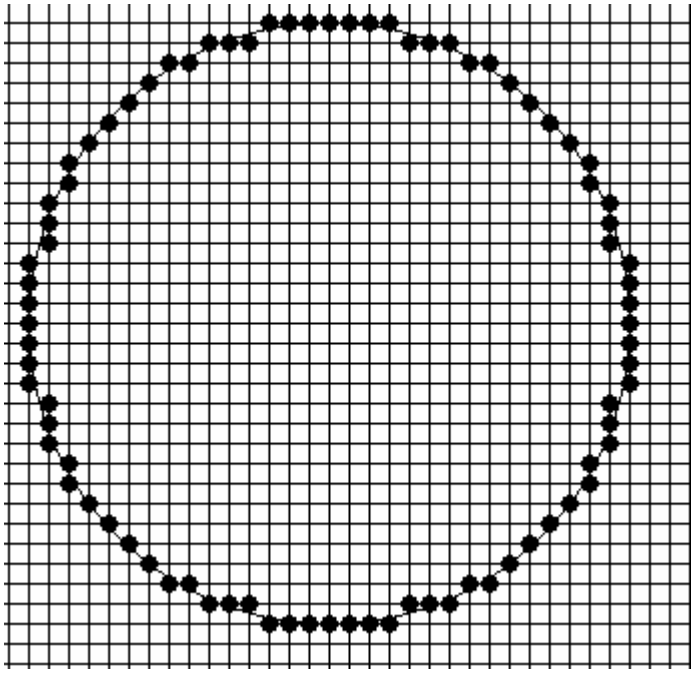
vagyis

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

Vegyük észre, hogy míg az egyenes rajzolásánál a ΔE, ΔSE elsőrendű differenciák konstans értékek voltak, most a ΔE és ΔSE az xp,yp lineáris függvénye. Ez azt jelenti, hogy minden egyes lépésben a ΔE és ΔSE értékeket (még az aktuális pont koordinátái alapján) újra kell számolni.Először meg kell határoznunk a kezdeti értékeket. Az algoritmus a (0,r) pontból indul, így: d0 = F(1,r-1/2)= 5/4-r .

Látható, hogy ekkor d0 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, ΔE és ΔSE is egészek d' mindig egész lesz, így egyszerűen tekinthetjük a d'<0 feltételt.

```
procedure CircleFirst(u,v,r:integer);
var
  xp,yp,d:integer;
begin
  xp:=0; { kezdő 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;
```



2.3 Vágási algoritmusok

2.3.1 Cohen-Sutherland vágóalgoritmus (vágás téglalapra):

A leghatékonyabb a Cohen-Sutherland vágóalgoritmus. Ez a síkot 9 részre osztja. A középső 9-ed a képernyő (ablak). Egy négyjegyű bináris kód minden ponthoz hozzárendelhető

1001	1000	1010
0001	0000	0010
0101	0100	0110

A 0. bit egyes, ha a pont balra esik a baloldali képernyőéltől.

Az első bit egyes, ha a pont jobbra esik a jobboldali képernyőéltől.

A második bit egyes, ha a pont az alsóíél alatt van.

A harmadik bit egyes, ha a pont a felső él felett van.

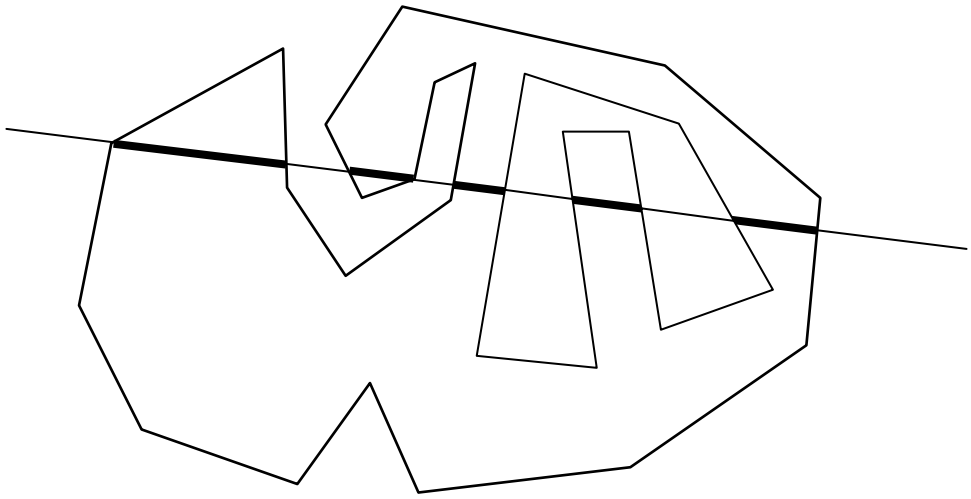
Ha a szakasz 2 végpontjához rendelt kód csupa nulla, akkor a szakasz a képernyőn belül van. Ha a két kódnak van olyan bitje, hogy azonos helyiértéken egyes van, akkor az a szakasz eldobható, mert a képen kívülre esik. Ha a szakasz két végpontja közül valamelyik kódjában van egyes, akkor az egyes helyi értékének megfelelő képernyőéllel el kell metszeni a szakaszt, és a metszéspontra módosítani a szakasz végpontját, és az új kódot újból meg kell vizsgálni. Ezt addig folytatjuk, amíg a szakasz végpontjaihoz rendelt kódok csupa nullák nem lesznek. Előnyei: hatékony, ha nagy valószínűséggel kívül esnek a szakaszok a képernyőn; jól általánosítható 3D-ben, itt 6 bitesek a kódok. Hátránya az, hogy a poligon ablakra nem általánosítható.

2.3.2 Szakaszvágása konvex poligonra:

A vizsgált egyenes egyenletébe behelyettesítjük a poligon csúcsainak koordinátáit, és tároljuk az eredmény előjelét.. Ha mindenhol azonos az előjel, akkor az egyenes a poligonon kívül esik. Ha a kapott előjelsorozat egymást követő elemei különböznek, akkor a két csúcs által meghatározott szakaszt metszi a vágandó szakasz egyenese. Csak két helyen lehet előjelváltás, mivel konvex a poligon. Kiszámoljuk a szakasz egyenesének és a poligonnak a metszéspontjait, majd valamely (nem nulla) koordináta szerint rendezzük a négy pontot. Megrajzoljuk a megfelelő két pont között a szakaszt.

2.3.3 Vágás konkáv poligonra:

Behelyettesítjük a poligont alkotó csúcsok koordinátáit a vágandó szakasz egyenesének egyenletébe. A kapott értékek előjeleit figyelve az előjelváltások esetén az aktuális poligon él metszi a szakasz egyenesét. Kiszámoljuk a szakasz egyenesének és a poligonnak a metszéspontjait, és ezeket x (vagy y) koordináta szerint rendezzük. Majd beszúrjuk a szakasz két végpontját a rendezett sorba, és megnézzük, hogy hol van a két végpont a metszéspontokhoz képest. A szakasz egyik végpontjától számítva a páratlan és páros metszéspontok közötti szakaszokat kell rajzolni, a két végpont között. Ha az egyenes párhuzamos az y tengellyel, akkor y szerint kell rendeznünk.

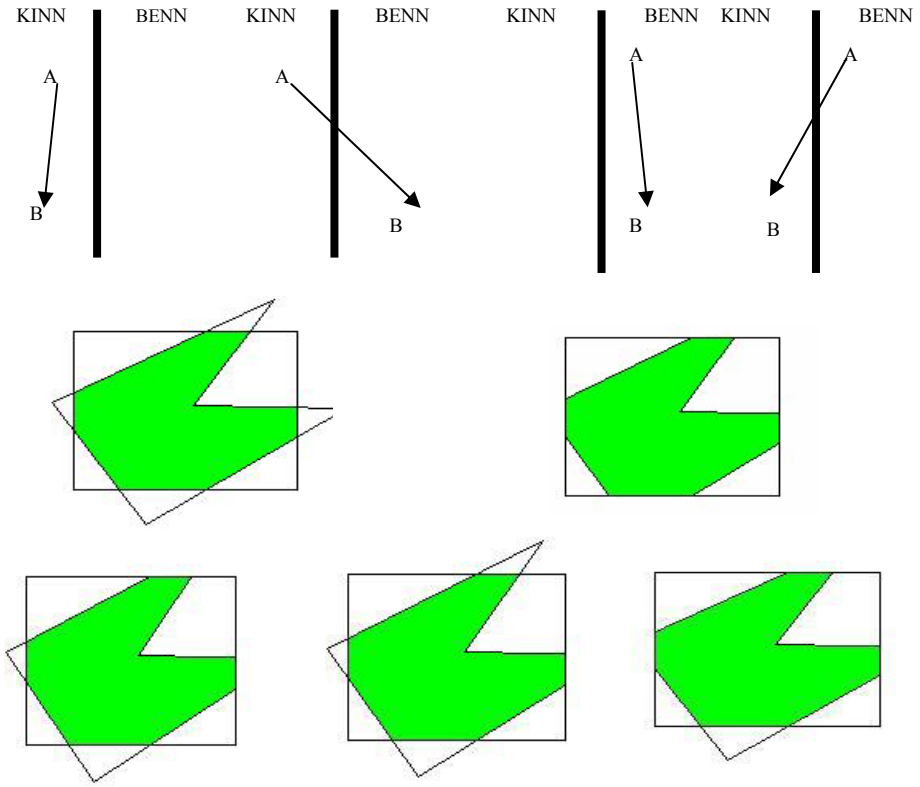


2.3.4 Sudherland-Hodgman poligon vágó algoritmus téglalap alakú ablakra

Nem elég csak a poligon éleit vágni, mert úgy csak az élek egy halmazát kapjuk meg, ami nem poligon. Az ablak négy élével egymás után elvágjuk az alakzatot. Minden ablak élre vonatkozóan egymásután létrehozunk az eredeti éleket sorba véve egy új listát.

A poligon minden AB (irányított) élére vonatkozóan az alábbi esetek lehetségesek:

1. Mindkét csúcs kívül van: nincs output
2. Mindkét csúcs benn van: B csúcs felkerül a listára (Ha nem az első élről van szó, A már rajta volt)
3. A bennt van, B kinn: Az ablak él és AB metszéspontja felkerül a listára
4. B benn van, A kinn: először AB és az ablak él metszéspontja, majd B is felkerül a listára.



Konkáv alakzat esetén az is előfordulhat, hogy több poligon lesz a vágás végeredménye.

2.4 Kitöltés

Egy kitöltendő zárt alakzat (poligon) kétféle módon lehet megadva. Az alakzatot határoló „görbe” –azaz szomszédos pixelek sorozata ismert, a háttérszíntől eltérő határszín által, vagy a poligon csúcsainak koordinátái ismertek. Ezek alapján kell a belső pixeleket megtalálni, melyeket színezni akarunk.

2.4.1 Színinformáción alapuló eljárások

2.4.1.1 Él flag módszer

Határszínnel adott zárt tartományon dolgozunk. Vízszintes scan-line mentén határszínű pixelhez érve állítunk egy flag-et. Ha true az értéke benn vagyunk, egyébként kinn. A vízszintes élek esetén a flag-et nem állítjuk, ennek az esetnek a kezelése külön megoldandó.

Pseudo kód:

```
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;
```

2.4.1.2 Rekurzív módszer

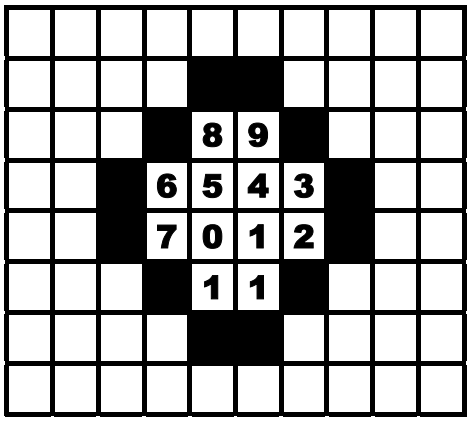
Háttérszínű, zárt tartomány színezésére való. Bemenő paraméterként egy belső pixelt kell megadni. Rekurzívan megvizsgálja a szomszédos pixeleket, és amelyik háttérszínű, kiszínezi. A rekurzió miatt nagy stack igénye van, és viszonylag lassú.

A kód:

```
flood_fill(x,y)
if (getpixel(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;
```

Vagy közvetlen memóriacímekre való hivatkozással

```
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;
```



2.4.2 Csúcsaival adott poligon kitöltése

2.4.2.1 Téglalap kitöltése

Legegyszerűbb egy a koordináta-tengelyekkel párhuzamos élű téglalap kitöltése, ekkor ugyanis egy dupla ciklussal, melyek a 2-2 párhuzamos határoló élek között futnak, egyszerű megtalálni a belső pixeleket. Az egymással közös élben érintkező téglalapoknál problémát jelent, hogy a két téglalap fillezésének sorrendjétől függően más színű lesz a közös él darab, és fellép egy „szőrösödési" hatás. A probléma kezelésére megoldás, hogy a belső pixel definícióját úgy adjuk meg, hogy a déli és nyugati határoló élt belső élnek, míg az északi és keleti élt külsőnek tekintjük.

2.4.2.2 Poligon-fillező módszer

Általános módszer ebben az esetben is az él-flag módszer.

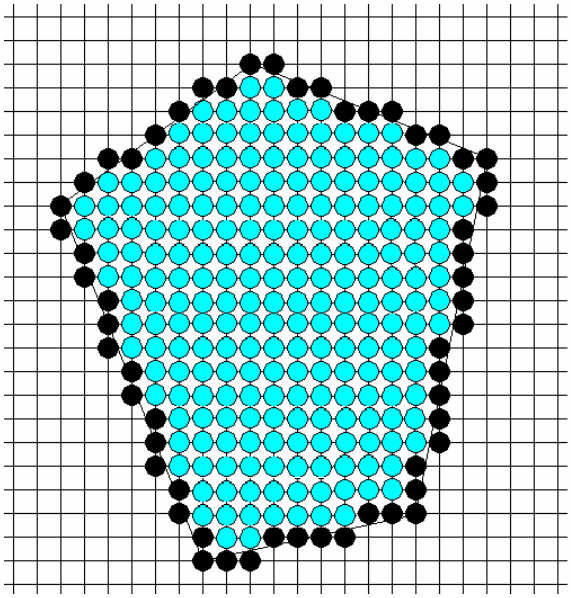
A legdélibb és legészakibb csúcsok között indított minden vízszintes scan-line-ra az alábbi lépéseket tesszük:

1. A scan-line metszéspontjainak meghatározása a poligon minden élével.

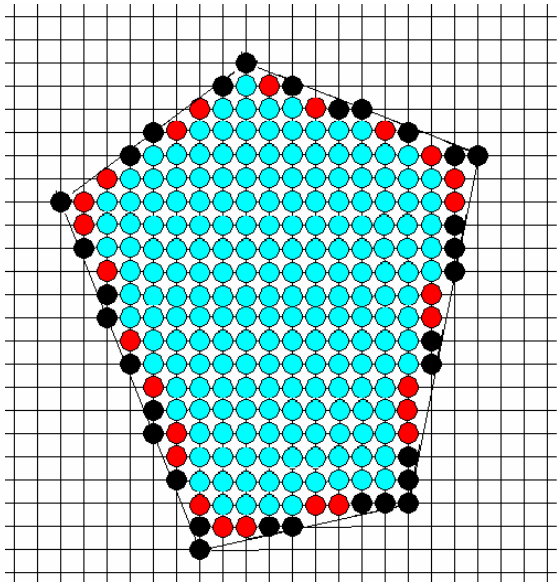
2. A metszéspontok rendezése x koordináta szerint.

3. Egy paritás bitet használva azon pixelek kitöltése a metszéspontok között, melyek a poligon belsejében fekszenek

A poligon csúcsai meg vannak adva egész koordinátákkal. A feladat itt is a belső pixelek meghatározása. A poligon éleit pl. a middlepoint algoritmussal meg lehet határozni. Kérdés, hogy az egyenes rajzoló által generált pixelek közül melyeket tekintsük belső és melyeket külső pixeleknek. Erre az a megoldás, hogy megkülönböztetünk úgynevezett baloldali és jobb oldali éleket, aszerint hogy páratlanadik vagy párosadik metszéspontról van szó. Baloldali él esetén felfelé, jobb oldali él esetén lefelé kerekítünk.. Azaz kerekített midle-pointot alkalmazunk



A midle-point szerinti szélső pixelek



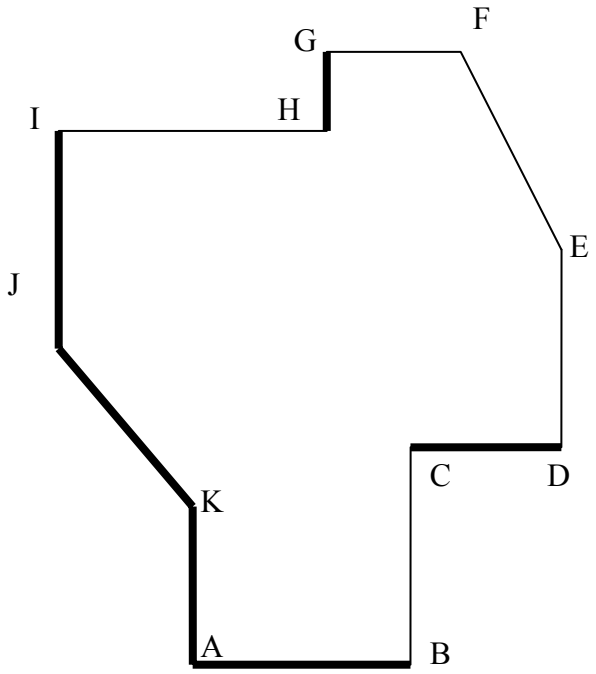
Az új „szélső” pixelek

Felmerülő problémák:

- Az egész koordinátájú metszéspontok esetén belső vagy külső pixelként tekintsük a metszéspontot?
- Mi a teendő ha csúcsponton halad át a scin-line? (A flag kétszer kapcsolna!)
- Mi a teendő vízszintes élek esetén?

Megoldások:

- Baloldali élnél belsőként, jobb oldali élnél külsőként definiáljuk. (Mint a téglalapnál)
- A paritásbitet csak a déli csúcs esetén állítjuk.
- A déli élet belsőként, az északi élet külsőként definiáljuk. Ez automatikusan bekövetkezik az előző pont alapján.



Algoritmus:

Az algoritmus során használunk egy éltáblát (ET), melynek tulajdonságai:

- az éltábla annyi elemű, ahány scanline-unk van
- a tábla elemei listák, melyekben azon élekről van adat, melyek az adott scanline-t metszik
- a vízszintes lista élei xmin koordinátájuk szerint vannak rendezve az adatok az élekről: az y koordináták az élek alacsonyabb csúcsának y koordinátája, az ymax az él maximális y koordinátája, az xmin az él alacsonyabb csúcsának az x koordinátája
- 1/m az él meredeksége.

C-ben a megfelelő adatstruktúra a következő:

```
typedef struct ETstruct{
       int y;
       struct ETLstruct *etl;
       struct ETstruct *next;
      }ET;                // eltabla

typedef struct ETLstruct{
       int x1,y2;
       double m;
       struct ETLstruct *next;
      }ETL;                    // eltablahoz tartozo
                               //ellista

typedef struct AETstruct{
       double x;
       int y2;
       double m;
       struct AETstruct *next;
       }AET;               // aktiv eltabla
```

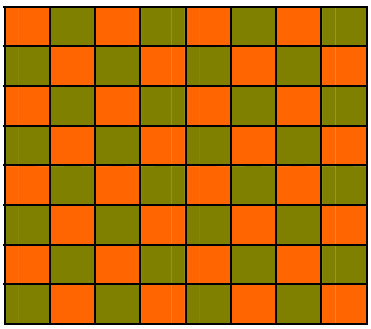
Van egy aktív éltábla (AET) is hasonló tulajdonságokkal, csak ebben nem annyi elem van, ahány scan-line. Ez a tábla az algoritmus folyamán mindig változó

Az algoritmus lépései:

1. Töltsük fel az ET listát.
2. Legyen y az ET lista első elemének az y-ja.
3. Inicializáljuk üresnek az AET listát.
4. Ismételjük a következőket, amíg az ET és AET listák üresek nem lesznek:
   4.1. Tegyük az AET listába azokat az éleket, amelyekre y= ymin , majd rendezzük az AET-ben lévő éleket az x koordináta szerint.
   4.2. Rajzoljuk ki az y scan-line-t, az AET-ben lévő x koordináta-párok között, figyelembe véve a paritást.
   4.3. y:=y+1
   4.4. Távolítsuk el azokat az éleket az AET-ből, amelyekre y= ymax.
   4.5. Minden nem függőleges AET-beli élre x:=x+1/m

2.4.3 Kitöltés mintával

Az alakzatokat különböző mintákkal is ki lehet tölteni. Ehhez egy n*m-es (általában 8*8-as) kitöltő kép szükséges, amely színinformációkat tartalmaz.

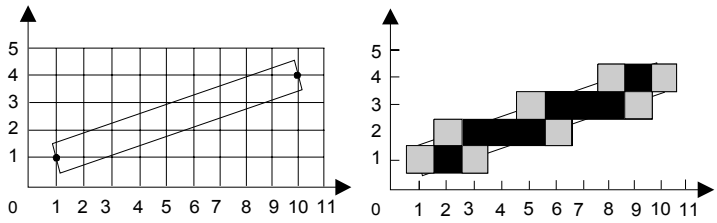


Módszerek:

1. A mintát gondolatban fölmásoljuk a teljes képernyőre (0,0)-tól, és ahol az alakzat van, ott átlátszóvá tesszük a képernyőt. Ekkor a minta a képernyő pixeljeihez van rendelve, tehát ha az objektum mozog, a minta nem fog vele együtt mozogni. Ha a minta egy mxn-es tömbben van tárolva, akkor az alakzat egy (x,y) koordinátájú pixelének színe a mintát tároló táblázat (x mod m, y mod n) elemének megfelelő színű lesz.
2. A minta egy meghatározott pixelét az alakzat egy pixeléhez kell hozzárendelni, így eltolásnál a minta az alakzattal mozog. (Ha a koordináta rendszert is hozzárendeljük, akkor még forgatható is lesz.)

2.5 Élsimítás (Anti-aliasing)

A szakasz rajzoló alap algoritmusok a ferde egyeneseket töredezetten ábrázolják. Az anit-aliasing módszer a ferde vonalak töredezettség mentes képének és a látványnak a javítására szolgál. A megtalált pixeleken túl különböző színintenzitással további pixelek vesznek részt az egyenes kirajzolásánál, így enyhíteni lehet az egyenes töredezettségét. Az elméleti szakaszra egy egy pixel szélességű téglalap tartományt fektetünk. Ha egy pixelnek van a téglalaptartománnyal közös része, akkor megfelelő szinintezitással kirajzoljuk.



2.5.1 Súlyozatlan antialiasing:

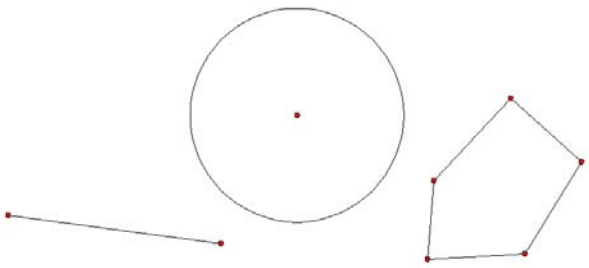
Itt azt nézzük, hogy az adott pixelnek hány százaléka van lefedve a téglalappal, azaz a szinintenzitás mértéke területarányos.

2.5.2 súlyozott antialiasing:

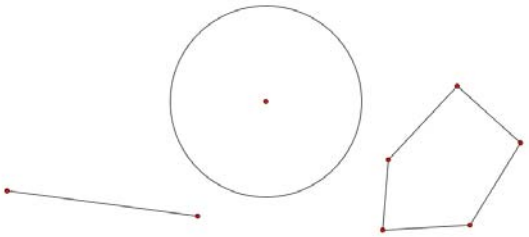
A pixeleket kör alakúnak feltételezzük, és egységnyi magasságú kúpokat illesztünk feléjük. A téglalaptartományra pedig ugyancsak egységnyi magasságú hasábot fektetünk, és a szinintenzitást a térfogatarányoknak megfelelően állítjuk be. Ezzel a módszerrel nem csak a lefedettség mértéke, hanem a pixelnek az elméleti egyenestől való távolsága is figyelembe vevődik.

2.5.3 Super-sampling

Ez az elsimításnak egy másik módszere, melynek lényege, hogy az éleken elhelyezkedő pixeleket felbontjuk 4*4=16 darab további részre, melyeket subpixeleknek nevezünk. Ezek nem valódi képpontok. Ez azt jelenti, hogy a raszteres képpontokat elvileg egy nagyobb felbontásnak megfelelően számítjuk ki. Egy megjelenített pixel színe vagy szürkeségértéke ezt követően a hozzátartozó részpixelekhez rendelt értékek átlagaként kerül kiszámításra.



Elsimítás nélkül



Élsimítással

3 Tanszformációk

A $\mathbf{r}(x, y, z) \rightarrow \mathbf{r}'(x', y', z')$ hozzárendelést 3D-s transzformációnak nevezzük, ahol

$$x' = f_1(x, y, z)$$
$$y' = f_2(x, y, z)$$
$$z' = f_3(x, y, z)$$

Az **r** vektornak megfelelő pontokat tárgypontoknak, az **r'** vektoroknak megfelelő pontokat pedig képpontoknak nevezzük. A számítógépes grafikában két olyan transzformációval találkozunk, amelyek matematikailag teljesen azonos formában írhatók le, ugyanakkor lényegüket tekintve eltérőek. Ezek a koordináta-transzformáció és a pont-transzformáció.

Koordináta-transzformációról akkor beszélünk, ha a tárgypont egy új koordináta-rendszerre vonatkozó koordinátáit határozzuk meg, a régiek ismeretében. Ilyenkor tehát a vizsgált tárgy változatlan, csupán nézőpontunkat változtatjuk meg. A grafikus objektum változatlan marad. Ilyenre például akkor van szükség, ha a nézőpontunkat változtatjuk 3D-ben.

Pont-transzformációról akkor beszélünk, ha a grafikus objektum pontjaihoz új pontokat rendelünk hozzá. A hozzárendelés módjától függően a tárgyalandó transzformációk lehetnek egybevágósági, hasonlósági, affin vagy projektív transzformációk. A dimenzió vesztéssel járó transzformációkat leképezéseknek nevezzük.

A térbeli tárgyaknak a számítógépes grafikában való feldolgozása során koordináta- és pont-transzformációt is alkalmazunk. Előbbi jellegzetesen a tárgyhoz képest elfoglalt nézőpontunk változtatása esetén szükséges, utóbbi pedig a testek különböző mozgatásainak, nagyításának, vetítésének matematikai leírására szolgál. Koordináta-transzformációt végezve a transzformáció kölcsönösen egyértelmű, hiszen új koordináták is maradéktalanul őrzik az eredeti információtartalmat. A pont-transzformációknál viszont előfordulhat, hogy a tárgy és a kép pontjainak kapcsolata nem mindig kölcsönösen egyértelmű. Gondoljunk például a 3D-s modelltér leképzésére 2D-s nézetre. Ez tipikus esete az elfajuló transzformációnak.

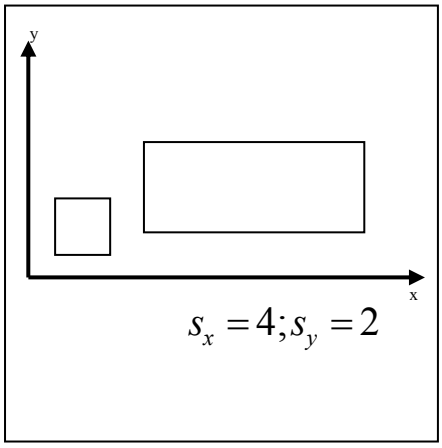
A transzformációk egységes kezelése miatt úgynevezett homogén koordinátákat vezetünk be. A 3 dimenziós projektív teret a valós 4 dimenziós vektortérbe ágyazzuk be, a projektív tér pontjainak reprezentálása a zérus vektortól különböző 4 dimenziós vektortér vektorainak egy-egy osztályával történik. Egy osztályba tartoznak a lineárisan függő vektorok, vagy másképp az egymással arányos számnégyesek ugyanazt a pontot reprezentálják. Ezeket a koordinátákat nevezzük homogén koordinátáknak. Ha a pont negyedik koordinátája nem zérus, akkor a pont $\mathbb{E}^3$-nak is pontja, egyébként a pontot végtelen távoli pontnak nevezzük. A végesben lévő pontok inhomogén koordinátáit megkapjuk, ha a negyedik koordinátával elosztjuk a pont első három koordinátáját: $y_i = \frac{x_i}{x_4}$, ahol $x_4 \neq 0$. A síkbeli homogén koordinátákat hasonlóan értelmezzük, a pontokat és az egyeneseket valós számhármasok egy-egy osztályával jellemezzük.

3.1 2D-S Transzformációk

3.2 Eltolás

Inhomogén alak:

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



mátrix alakban:

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

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

Homogén alak:

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},\ \mathbf{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},\ \mathbf{T} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3.2.1 Forgatás origó körül

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

mátrix alakban:

Inhomogén alak :

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

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

Homogén alak :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3.2.2 Skálázás

Inhomogén alak:

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

mátrix alakban:

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

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

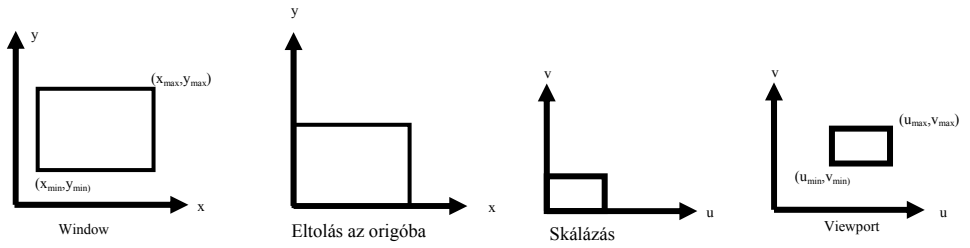
Homogén alak:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Ha s_x=s_y, akkor hasonlóságról, egyébként affinitásról beszélünk.

3.2.3 Window to viewport-transzformáció

A valós világ leképezése a rajzolási területre. A kép generálásához fel kell vennünk egy ablakot az (u,v) síkban, melyen keresztül a 3D-s modellteret látjuk. Mindazon objektumok, melyek képe az (u,v) síkra vetítve az ablakon kívül esik, a jelenet képén nem fog szerepelni. Azaz egy megjelenítendő szakasznak csak a képernyőn lévő részét kell kirajzolni. A vágás előtt egy vetítés történik az (u,v) síkra. Első lépés a világ koordináta-rendszerbeli rajzterület eltolása az origóba, majd a két rajzterület megfelelő élei arányának megfelelően skálázás és visszatolás. Ami kilóg a window-ból, az a viewport-ból is ki fog, azaz vágni kell.



Az Window origóba való eltolásának mátrixa: $\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix}$.

A skálázás: $\mathbf{S} = \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Visszatolás a Viewportba: $\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{bmatrix}$

Az eredményt e 3 mátrix szorzata adja:

$$\mathbf{P}' = (\mathbf{T}_2\mathbf{S}\mathbf{T}_1) \bullet \mathbf{P} =$$

$$\begin{bmatrix} \frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}} & 0 & -x_{\min}\frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}}+u_{\min} \\ 0 & \frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}} & -y_{\min}\frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}}+v_{\min} \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} (x-x_{\min})\frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}}+u_{\min} \\ (y-y_{\min})\frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}}+v_{\min} \\ 1 \end{bmatrix}$$

3.3 Térbeli pont-transzformációk

A térbeli pont transzformációk mátrix reprezentációban homogén koordinátákat használva adjuk meg.

3.3.1 Egybevágósági transzformációk (mérettartó transzformációk):

3.3.1.1 Eltolás

Az eltolást a $\mathbf{d}(d_x, d_y, d_z)$ vektorral adjuk meg. Az eltolást leíró mátrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.1.2 Forgatás α szöggel

A térbeli forgatás nem pont körül, hanem egyenes körül értelmezett. Az alábbi mátrixok a koordináta tengelyek körüli forgatásokat írják le.

- x tengely körül

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- y tengely körül $$M = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- z tengely körül $$M = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.1.3 Tükrözés az {x,y} síkra

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.2 Hasonlósági transzformációk (arányosságtartó transzformációk):

3.3.2.1 Kicsinyítés, nagyítás origó középponttal:

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ha λ nagyobb mint egy, akkor nagyítás, ha pedig kisebb mint egy, akkor kicsinyítés történik.

3.3.3 Affin transzformációk:

3.3.3.1 Skálázás

Origóból történő, tengelyek menti nyújtás.

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \nu & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.3.2 Nyírás

Adott egy origón átmenő fix sík **n** normálvektorával, egy a síkkal párhuzamos **t** irány, valamint egy λ>0 valós szám. A hozzárendelés:

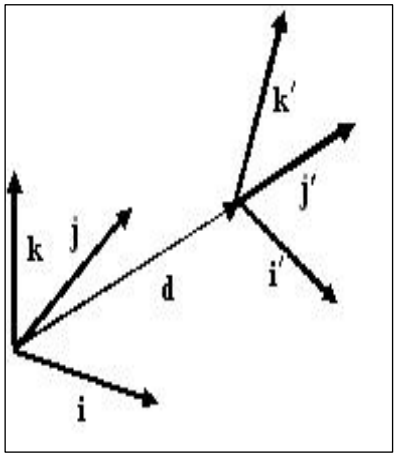
$$\mathbf{P}' = \mathbf{P} + \lambda \bullet d\mathbf{t} = \mathbf{P} + \lambda \bullet (\mathbf{np})\mathbf{t}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3.4 Koordináta-transzformációk

A 3D-s euklideszi térben egy K koordináta-rendszerről egy K' koordináta-rendszerre térünk át és feltételezzük, hogy ortonormált Descartes-féle koordinátarendszerekről van szó. Jelölje **i, j, k** illetve **i′, j′, k′** a két koordinátarendszer egységvektorait, valamint **d** mutasson az eredeti rendszer kezdőpontjából az új rendszer kezdőpontjába. Először meghatározzuk egy pont eredeti rendszerbeli koordinátát, ha ismerjük az új rendszerben a koordinátákat.

Az **i′, j′, k′** vektorok **i, j, k** rendszerbeli koordinátáit rendre jelölje $i_x^{'}, i_y^{'}, i_z^{'}, j_x^{'}, j_y^{'}, j_z^{'}, k_x^{'}, k_y^{'}, k_z^{'}$.



$\mathbf{p} = \mathbf{d} + \mathbf{p}' = \mathbf{d} + x'\mathbf{i}' + y'\mathbf{j}' + z'\mathbf{k}'$, vagy mátrix reprezentációban:

$$\begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ 1 \end{bmatrix} = \begin{bmatrix} i_x' & j_x' & k_x' & d_x \\ i_y' & j_y' & k_y' & d_y \\ i_z' & j_z' & k_z' & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{x}' \\ \mathrm{y}' \\ \mathrm{z}' \\ 1 \end{bmatrix}$$

Ha ez eredeti rendszerben ismertek egy pont koordinátái, és az új rendszerbeli előállítás szükséges, akkor $\mathbf{p}' = \mathbf{p} - \mathbf{d}$ egyenlőségből indulunk ki. Egy pont valamely koordinátája az adott tengelyhez tartozó normál egységvektornak és a pontba mutató vektornak a skaláris szorzataként áll elő. Tehát:

$$\begin{aligned} x' &= \mathbf{p}'\mathbf{i}' = (\mathbf{p} - \mathbf{d})\mathbf{i}' = \mathbf{p}\mathbf{i}' - \mathbf{d}\mathbf{i}' \\ y' &= \mathbf{p}'\mathbf{j}' = (\mathbf{p} - \mathbf{d})\mathbf{j}' = \mathbf{p}\mathbf{j}' - \mathbf{d}\mathbf{j}' \\ z' &= \mathbf{p}'\mathbf{k}' = (\mathbf{p} - \mathbf{d})\mathbf{k}' = \mathbf{p}\mathbf{k}' - \mathbf{d}\mathbf{k}' \end{aligned}$$

Mátrixreprezentációban az eredmény:

$$\begin{bmatrix} \mathrm{x}' \\ \mathrm{y}' \\ \mathrm{z}' \\ 1 \end{bmatrix} = \begin{bmatrix} i_x' & i_y' & i_z' & -\mathbf{d}\mathbf{i}' \\ j_x' & j_y' & j_z' & -\mathbf{d}\mathbf{j}' \\ k_x' & k_y' & k_z' & -\mathbf{d}\mathbf{k}' \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ 1 \end{bmatrix} = \begin{bmatrix} i_x' & i_y' & i_z' & -d_x i_x' - d_y i_y' - d_z i_z' \\ j_x' & j_y' & j_z' & -d_x j_x' - d_y j_y' - d_z j_z' \\ k_x' & k_y' & k_z' & -d_x k_x' - d_y k_y' - d_z k_z' \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ 1 \end{bmatrix}$$

4 Paraméteres görbék

A térbeli görbék modellezésének legfontosabb eszköze a paraméteres vektor függvény, mely alkalmazásával a síkbeli és térbeli görbéket hasonló módon írhatjuk le. Ez egy $t \to \mathbf{r}(t),\ t \in [a,b]$ paraméteres skalár-vektor függvény megadását jelenti, ahol [a,b] a paramétertartomány. Az egyenlet komponensekben:

$$\begin{aligned} x &= x(t) \\ y &= y(t) \\ z &= z(t) \end{aligned}$$

Síkgörbék esetében természetesen csak két komponens szükséges. Ezek szerint minden egyes t konkrét számértéket behelyettesítve, a függvény egy olyan konkrét r vektort állít elő, mely a térgörbe egy pontjára mutat. Az r(t) függvényről általában feltételezzük, hogy kölcsönösen egyértelmű és folytonos leképzés, azaz egy konkrét t_0 értékhez egy és csak egy $\mathbf{r}_0$ vektort rendelünk hozzá. Azt is feltételezzük, hogy t-szerint folytonosan differenciálható és deriváltja nem nulla. Az utóbbi kikötés szemléletesen azt jelenti, hogy a görbén nincsenek csúcsok, hegyes részek és az érintővektort a görbe bármely pontjában meg lehet húzni. Az érintővektort az $\mathbf{r}$ deriválásával kapjuk, azaz az $x(t), y(t), z(t)$ skalárfüggvény komponensekből képzett differenciálhányadosából.

4.1 Interpoláció

A térbeli görbék közül azok kiválasztását, melyek a tér előre megadott $P_0, P_1, ..., P_n$ pontjain áthaladnak, egy interpolációs feladat megoldásának nevezzük. A síkbeli görbék esetében legyen adott a $P_0, P_1, ..., P_n$ pontsorozat az $\mathbf{r_0}, \mathbf{r}_1, ..., \mathbf{r}_n$ vektorokkal. Keressük azt az $t \to \mathbf{r}(t),\ t \in [a,b]$ skalár- vektor függvényt, mely kielégíti a következő feltételt: találhatóak olyan $t_0, t_1, ..., t_n$ paraméterértékek, hogy

$$\begin{aligned} \mathbf{r_0} &= \mathbf{r}(t_0) \\ \mathbf{r}_1 &= \mathbf{r}(t_1) \\ &\vdots \\ \mathbf{r}_n &= \mathbf{r}(t_n) \end{aligned}$$

teljesül. Ebben az esetben az $\mathbf{r}(t)$ skalár-vektor függvényt interpolációs görbének, $P_0, P_1, ..., P_n$ pontokat pedig az interpolációs görbe kontrollpontjainak nevezzük.

4.2 Approximáció

Ekkor egy görbecsaládból azt a görbét választjuk ki, mely az előre megadott $P_0, P_1, ..., P_n$ térbeli pontokat megközelíti. Az approximációs görbe általában nem

halad át a megadott kontrollpontokon, hanem a kontrollpontok valamilyen módon hatnak a görbe alakjára.

Ahhoz, hogy az interpolációs vagy approximációs eljárás a gyakorlatban is használható legyen, valamilyen módon szűkítenünk kell a szóba jöhető végtelen sok $\mathbf{r} = \mathbf{r}(t)$ skalár-vektor függvény halmazát. Ezért az összes lehetséges $\mathbf{r} = \mathbf{r}(t)$ függvények közül csak a polinomokat vesszük figyelembe, azaz az

$$x(t) = a_n t^n + a_{n-1} t^{n-1} + ... + a_0$$
$$y(t) = b_n t^n + b_{n-1} t^{n-1} + ... + b_0$$
$$z(t) = c_n t^n + c_{n-1} t^{n-1} + ... + c_0$$

alakú függvények között keressük a feladatnak megfelelőket. Minél kisebb a polinom fokszáma, annál kevesebb művelettel számíthatjuk ki a függvényértékeket, de túl alacsony fokszámú poligont nem választhatunk, mivel akkor a bonyolultabb görbéket és felületeket nem lehetne jól közelíteni. Másodfokú polinom esetén:

$$x(t) = a_2 t^2 + a_1 t + a_0$$
$$y(t) = b_2 t^2 + b_1 t + b_0$$
$$z(t) = c_2 t^2 + c_1 t + c_0$$

Ha másodfokúak a koordinátafüggvények, akkor a generált görbe síkbeli lesz. Ezért a térgörbék és felületek modellezésére a legalább harmadfokú polinomokat választották. Harmadfokú polinomokkal modellezhetők olyan geometriai tulajdonságok is, mint az önmetszés, csúcspont vagy az inflexiós pont.

4.3 Harmadrendű paraméteres görbék

Általános alakban a koordináta függvények az alábbiak:

$$x(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0$$
$$y(t) = b_3 t^3 + b_2 t^2 + b_1 t + b_0$$
$$z(t) = c_3 t^3 + c_2 t^2 + c_1 t + c_0$$

Vezessük be az alábbi jelöléseket:

$$\mathbf{C} = \begin{bmatrix} a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 \\ c_3 & c_2 & c_1 & c_0 \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$$\mathbf{Q}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \mathbf{C} \bullet \mathbf{T}$$

Célunk általában a C mátrix meghatározása. A különböző típusú interpolációk és approximációk esetén eltérő geometriai jellemzőkkel rendelkező görbéket állítunk elő, természetesen különböző meghatározó paraméterekkel. Az ismeretlenek száma 12 (vagy 8, ha síkban vagyunk). Ehhez 4 geometriai adat tartozhat, ezek általában pontok, de lehetnek előírt érintők is. Jelölje **G** a geometria adatokból álló sorvektort, azaz

$$\mathbf{G} = \begin{bmatrix} \mathbf{G_1} & \mathbf{G}_2 & \mathbf{G}_3 & \mathbf{G}_4 \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \end{bmatrix}$$

A **C** matrixot **GM** alakban keressük, ahol **M** 4x4-s valós elemű mátrix. Így a görbe általános alakja:

$$\mathbf{Q}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \mathbf{C} \bullet \mathbf{T} = \mathbf{G} \bullet \mathbf{M} \bullet \mathbf{T}$$

Az $\mathbf{B}(t)=\mathbf{MT}$ harmadfokú polinomokat súlyfüggvényeknek nevezzük. A görbe valamely t_0 paraméterű pontjában az érintő vektor egyszerűen származtatható:

$$\mathbf{Q}'(t) = \begin{bmatrix} x'(t) \\ y'(t) \\ z'(t) \end{bmatrix} = \mathbf{C} \bullet \mathbf{T}' = \mathbf{G} \bullet \mathbf{M} \bullet \mathbf{T}' = \mathbf{G} \bullet \mathbf{M} \bullet \begin{bmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{bmatrix}$$

4.4 Matematikai folytonosságok

- C0 matematikai folytonosság:
  Ebben az esetben a két görbe darabnak az illesztési pontban lehet törése, de hézagmentesen csatlakozik a két rész.
- C1 matematikai folytonosság:
  A két ívdarabnak az érintője is megegyezik, azaz koordinátafüggvényeik első deriváltja a csatlakozási pontban egyenlő.
- C2 matematikai folytonosság:
  Ebben az esetben a két ívdarabnak a csatlakozási pontban az érintője és a görbülete is megegyezik. Azaz a koordinátafüggvények első és második deriváltjának értéke is azonos.
- G1 geometriai folytonosság:
  Ebben az esetben a két ívdarabnak a csatlakozási pontban nem kell azonos deriváltakkal rendelkeznie. Csak az érintő egyenes azonos, de a derivált vektor nagysága és előjele ellenkező lehet.

4.5 Hermite-interpoláció (3. rendű)

Legyen adva 4 geometriai adat (pontok vagy érintők) $\mathbf{G} = \begin{bmatrix} \mathbf{G_1} & \mathbf{G}_2 & \mathbf{G}_3 & \mathbf{G}_4 \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \end{bmatrix}$ és a t_1, t_2, t_3, t_4 skalárok.
Keressük az M 4x4-es mátrixot, melyre teljesedik:

$$\mathbf{G}_1 = \mathbf{G} \bullet \mathbf{M} \bullet \mathbf{T}_1$$
$$\mathbf{G}_2 = \mathbf{G} \bullet \mathbf{M} \bullet \mathbf{T}_2$$
$$\mathbf{G}_3 = \mathbf{G} \bullet \mathbf{M} \bullet \mathbf{T}_3$$
$$\mathbf{G}_4 = \mathbf{G} \bullet \mathbf{M} \bullet \mathbf{T}_4$$

vagy röviden

$\mathbf{G} = \mathbf{G} \bullet \mathbf{M} \bullet \begin{bmatrix} \mathbf{T}_1 & \mathbf{T}_2 & \mathbf{T}_3 & \mathbf{T}_4 \end{bmatrix}$, ahol $\mathbf{T}_1\ \mathbf{T}_2\ \mathbf{T}_3\ \mathbf{T}_4$ oszlopvektorok, értékük vagy a

$$\mathbf{T} = \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \text{ vagy a } \mathbf{T}' = \begin{bmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{bmatrix}$$

képlet alapján számolandó, aszerint, hogy pontról vagy előírt érintőről van szó.

A $\mathbf{G} = \mathbf{G}\bullet\mathbf{M}\bullet\left[\mathbf{T}_1\ \mathbf{T}_2\ \mathbf{T}_3\ \mathbf{T}_4\right]$ egyenlőség akkor állhat fenn, ha

$$\mathbf{M}\bullet\left[\mathbf{T}_1\ \mathbf{T}_2\ \mathbf{T}_3\ \mathbf{T}_4\right] = \mathbf{E}, \text{azaz}\ \mathbf{M} = \left[\mathbf{T}_1\ \mathbf{T}_2\ \mathbf{T}_3\ \mathbf{T}_4\right]^{-1}$$

Konkrét esetek:

4.5.1 4 pontra illeszkedő Hermit-ív

Adottak a $\mathbf{G} = \left[\mathbf{P}_1\ \ \mathbf{P}_2\ \ \mathbf{P}_3\ \ \mathbf{P}_4\right]$ pontok, valamint előírjuk, hogy a $t_1 = -1, t_2 = 0, t_3 = 1, t_4 = 2$ paraméterekre:

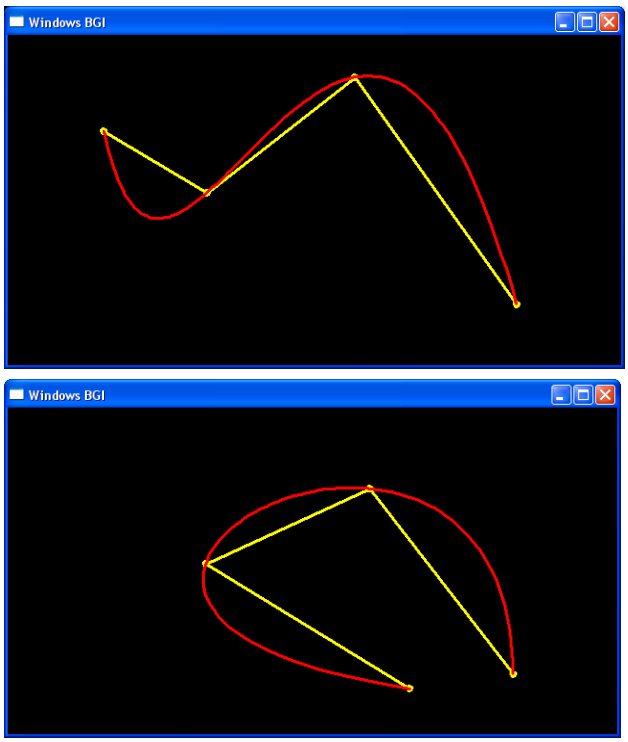
$$\mathbf{Q}(-1) = P_1, \mathbf{Q}(0) = P_2, \mathbf{Q}(1) = P_3, \mathbf{Q}(2) = P_4$$

Ekkor:

$$\mathbf{M} = \begin{bmatrix} -1 & 0 & 1 & 8 \\ 1 & 0 & 1 & 4 \\ -1 & 0 & 1 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & -\frac{1}{3} & 0 \\ \frac{1}{2} & -1 & -\frac{1}{2} & 1 \\ -\frac{1}{2} & \frac{1}{2} & 1 & 0 \\ \frac{1}{6} & 0 & -\frac{1}{6} & 0 \end{bmatrix}$$

$$\mathbf{Q}(t) = \left[\mathbf{P}_1\ \ \mathbf{P}_2\ \ \mathbf{P}_3\ \ \mathbf{P}_4\right]\bullet\begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & -\frac{1}{3} & 0 \\ \frac{1}{2} & -1 & -\frac{1}{2} & 1 \\ -\frac{1}{2} & \frac{1}{2} & 1 & 0 \\ \frac{1}{6} & 0 & -\frac{1}{6} & 0 \end{bmatrix}\bullet\begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} =$$

$$\mathbf{P}_1\bullet(-\frac{1}{6}t^3 + \frac{1}{2}t^2 - \frac{1}{3}t) + \mathbf{P}_2\bullet(\frac{1}{2}t^3 - t^2 - \frac{1}{2}t + 1) + \mathbf{P}_3\bullet(-\frac{1}{2}t^3 + \frac{1}{2}t^2 + t) + \mathbf{P}_4\bullet(\frac{1}{6}t^3 - \frac{1}{6}t),\ t \in [-1, 2]$$



4.5.2 3 pontra illeszkedő Hermit-ív, ha adott az első pontban az érintő:

$\mathbf{G} = [\mathbf{P}_1 \quad \mathbf{P}_2 \quad \mathbf{P}_3 \quad \mathbf{R}_1]$ valamint legyen $t_1 = -1, t_2 = 0, t_3 = 1$. Ekkor

$$\mathbf{M} = \begin{bmatrix} -1 & 0 & 1 & 3 \\ 1 & 0 & 1 & -2 \\ -1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{3}{4} & \frac{1}{2} & -\frac{5}{4} & 0 \\ -1 & -1 & 1 & 1 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \end{bmatrix}$$

$$\mathbf{Q}(t) = [\mathbf{P}_1 \quad \mathbf{P}_2 \quad \mathbf{P}_3 \quad \mathbf{R}_4] \bullet \begin{bmatrix} \frac{3}{4} & \frac{1}{2} & -\frac{5}{4} & 0 \\ -1 & -1 & 1 & 1 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \end{bmatrix} \bullet \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} =$$

$$\mathbf{P}_1 \bullet (\frac{3}{4}t^3 + \frac{1}{2}t^2 - \frac{5}{4}t) + \mathbf{P}_2 \bullet (-t^3 - t^2 + t + 1) + \mathbf{P}_3 \bullet (\frac{1}{4}t^3 + \frac{1}{2}t^2 + \frac{1}{4}t) + \mathbf{R}_4 \bullet (\frac{1}{2}t^3 - \frac{1}{2}t),\ t \in [-1,1]$$

4.5.3 2 pontra illeszkedő Hermit-ív, ha adottak az érintők is:

$$\mathbf{G}_H = [\mathbf{P}_1 \quad \mathbf{P}_2 \quad \mathbf{R}_1 \quad \mathbf{R}_2] \text{ valamint legyen } t_1 = 0, t_2 = 1. \text{ Ekkor}$$

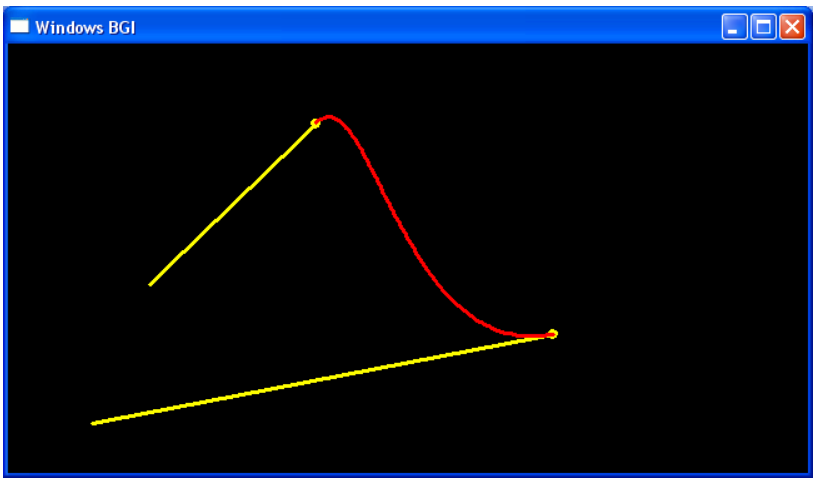
$$\mathbf{M}_H = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

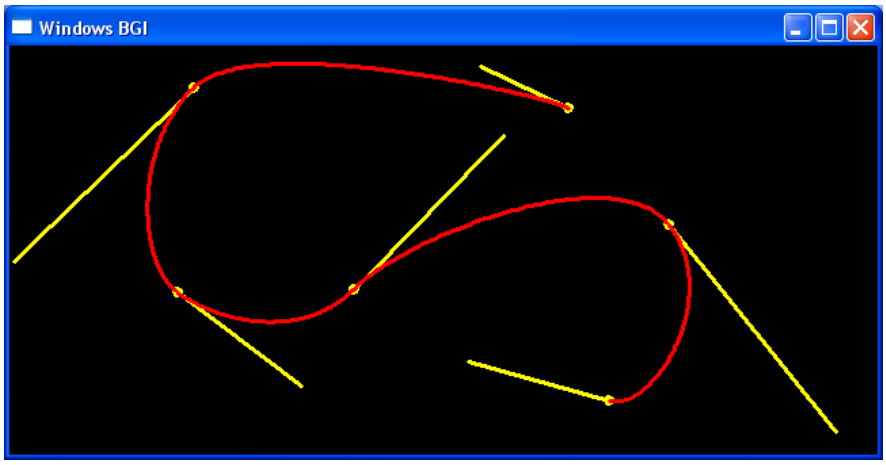
$$\mathbf{Q}(t) = [\mathbf{P}_1 \quad \mathbf{P}_2 \quad \mathbf{R}_1 \quad \mathbf{R}_2] \bullet \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} =$$

$$\mathbf{P}_1 \bullet (2t^3 - 3t^2 + 1) + \mathbf{P}_2 \bullet (-2t^3 + 3t^2) + \mathbf{R}_1 \bullet (t^3 - 2t^2 + t) + \mathbf{R}_2 \bullet (t^3 - t^2),\ t \in [0,1]$$

Ha 2 helyett n darab pontot és érintőt adunk meg, akkor a Hermite-ívekből egy egész görbét tudunk előállítani. Ekkor az előző eljárást sorban el kell végeznünk az egymás után következő pontpárokra. Ekkor a görbe elsőrendbeli folytonossága a kapcsolódási pontokban automatikusan érvényesül, mivel az i. ív végérintője megegyezik az (i+1). ív kezdőérintőjével (i=1, … , n-1).

A Hermite-íveknek a számítógépes grafikában történő alkalmazásában gondot okoz, hogy ezek a paramétertartomány affin transzformációira nem invariánsak.





Hermite-ívek összekapcsolása

4.6 Bézier-approximációs ívek

A harmadrendű Bézier íveket a két pontjával és két érintőjével adott Hermite ívre vezetjük vissza.

A Hermit-ívtől eltérően itt az érintővektorok helyett is pontokat adunk meg, melyek befolyásolják a görbe alakját, bár a görbe nem megy át ezeken a pontokon.

Legyen adva a Bézier ív 4 kontrollpontja, $\mathbf{G}_B = [\mathbf{P}_1 \quad \mathbf{P}_2 \quad \mathbf{P}_3 \quad \mathbf{P}_4]$. Előállítjuk azt a Hermite-ívet, mely meghatározó adatai közül a két adott pont $\mathbf{P}_1$ és $\mathbf{P}_4$,és a hozzájuk tartozó érintők pedig a $\overrightarrow{\mathbf{P}_1\mathbf{P}_2}$ valamint a $\overrightarrow{\mathbf{P}_3\mathbf{P}_4}$ vektorok háromszorosa, azaz $3(\overrightarrow{\mathbf{P}_2} - \overrightarrow{\mathbf{P}_1})$illeteve $3(\overrightarrow{\mathbf{P}_4} - \overrightarrow{\mathbf{P}_3})$. A két mátrix között tehát a következő összefüggés van:

$$\mathbf{G}_H = \mathbf{G}_B \bullet \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix},$$

$$\mathbf{Q}(t) = \mathbf{G}_B \bullet \mathbf{M}_B \bullet \mathbf{T} = \mathbf{G}_H \bullet \mathbf{M}_H \bullet \mathbf{T} = \mathbf{G}_B \bullet \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} \bullet \mathbf{M}_H \bullet \mathbf{T},$$

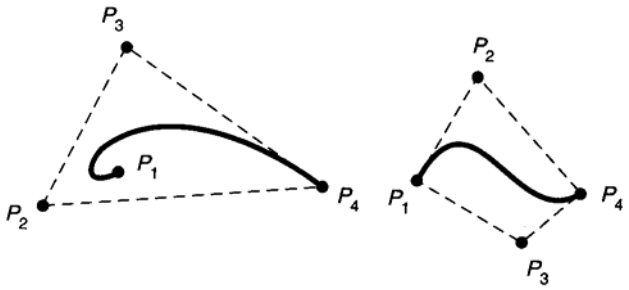
$$\mathbf{M}_B = \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} \bullet \mathbf{M}_H = \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} \bullet \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

A Bézier görbe előállításában szereplő $\mathbf{M_BT}$ szorzat tulajdonképpen négy harmadfokú polinom, melyek az úgynevezett Bernstein-féle polinomok, n=3 esetben. Ez alapján a Bézier görbe így is felírható:

$$\mathbf{Q}(t) = \mathbf{G}_B \bullet \mathbf{M}_B \bullet \mathbf{T} = \mathbf{G}_B \bullet \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} =$$

$$\mathbf{P}_1 \bullet (1-t)^3 + \mathbf{P}_2 \bullet 3(1-t)^2 t + \mathbf{P}_3 \bullet 3(1-t)t^2 + \mathbf{P}_4 \bullet t^3$$

A Bézier-ívek tartópontjai közül kettő $\mathbf{P}_1$ és $\mathbf{P}_4$ a görbén helyezkedik el, a $\mathbf{P_2}$ $\mathbf{P_3}$ pedig a görbe két végpontjába húzott érintőkön található. A görbe íve mindig a $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$, $\mathbf{P_4}$, kontrollpontok által meghatározott négyszög belsejében helyezkedik el.



4.6.1 Harmadrendű Bézier görbék csatolása.

Legyenek $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$, $\mathbf{A_4}$ és $\mathbf{B_1}$,$\mathbf{B_2}$,$\mathbf{B_3}$, $\mathbf{B_4}$ kontrollpontok és $\mathbf{Q_1}$(t), $\mathbf{Q_2}$ (t), t∈[0,1] a megfelő Bézier-szegmensek. Képezve $\mathbf{Q}(t)$ *t*-szerinti deriváltjait és helyettesítve a t=0 illetve t=1 értékeket az alábbi eredményt kapjuk:

- Nulladrendű folytonosság esetén: $\mathbf{Q_1}$ (1)= $\mathbf{Q_2}$ (0), azaz $\mathbf{A_4}$=$\mathbf{B_1}$.
- Elsőrendű folytonosság esetén:

  $$\frac{d}{dt}\mathbf{Q}_1(1) = \frac{d}{dt}\mathbf{Q}_2(0), \text{azaz } \mathbf{A_4 - A_3 = B_2 - B_1}$$

- Másodrendű folytonosság esetén:

$$\frac{d^2}{dt^2}\mathbf{Q}_1(1) = \frac{d^2}{dt^2}\mathbf{Q}_2(0), \text{ azaz } \mathbf{(A_4 - A_3) - (A_3 - A_2) = (B_3 - B_2) - (B_2 - B_1)}$$

4.7 B-spline görbe előállítása

Legyen adott a $\mathbf{P_0}, \mathbf{P_1}, ..., \mathbf{P_n}$ .pontsorozat, azaz n+1 pont. A kontrolpontok közül rendre tekintjük az egymást követő négyet, azaz az n-3 pontnégyest. Keresünk négy harmadfokú súlyfüggvényt, melyek eleget tesznek a következő feltételnek: az egymást követő pontnégyesekhez tartozó ívek csatlakozzanak, a csatlakozási pontban első és másodrendben legyenek folytonosak.

Az i. ívet jelöljük $\mathbf{Q}_i$-vel, mely ív függjön a t paramétertől, ahol $t \in [0,1], i = 3,4,...,n$. A $\mathbf{Q}_i$ ívet meghatározó geometriai adatok tehát $\mathbf{G}_i = \begin{bmatrix} \mathbf{P}_{i-3} & \mathbf{P}_{i-2} & \mathbf{P}_{i-1} & \mathbf{P}_i \end{bmatrix}$. A keresett súlyfüggvényeket (harmadfokú polinomokat) jelölje $X_0(t), X_1(t), X_2(t), X_3(t)$. Az i. szegmens tehát az alábbi alakú: $\mathbf{Q}_i(t) = \mathbf{P}_{i-3}X_0(t) + \mathbf{P}_{i-2}X_1(t) + \mathbf{P}_{i-1}X_2(t) + \mathbf{P}_iX_3(t) \quad i = 3,4...,n \quad t \in [0,1]$.
Mivel $\mathbf{Q}_i(1) = \mathbf{Q}_{i+1}(0), i = 3,4...,n-1$,ezért

$$X_0(1) = X_3(0) = 0$$
$$X_1(1) = X_0(0)$$
$$X_2(1) = X_1(0)$$
$$X_3(1) = X_2(0)$$

Az elsőrendű és másodrendű csatlakozásra tekintettel, hasonlóan:

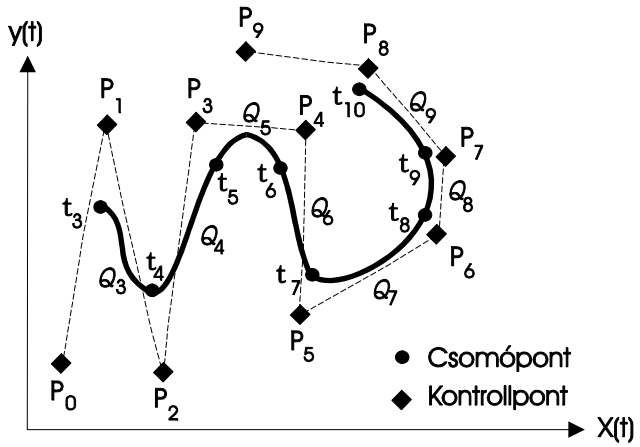
$$\dot{X}_0(1) = \dot{X}_3(0) = 0$$
$$\dot{X}_1(1) = \dot{X}_0(0)$$
$$\dot{X}_2(1) = \dot{X}_1(0)$$
$$\dot{X}_3(1) = \dot{X}_2(0)$$

és

$$\ddot{X}_0(1) = \ddot{X}_3(0) = 0$$
$$\ddot{X}_1(1) = \ddot{X}_0(0)$$
$$\ddot{X}_2(1) = \ddot{X}_1(0)$$
$$\ddot{X}_3(1) = \ddot{X}_2(0)$$

Ez összesen 15 egyenlet, 16 ismeretlennel. Hozzávesszük a koordináta transzformációval szembeni invarianciát biztosító Cauchy-egyenletet:

$$X_0(t) + X_1(t) + X_2(t) + X_3(t) \equiv 1$$



Az egyenletrendszert megoldva a keresett súlyfüggvények:

$$X_0(t) = \frac{1}{6}(1-t)^3$$

$$X_1(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)$$

$$X_2(t) = \frac{1}{6}(-3t^3 + 3t^2 + 3t + 1)$$

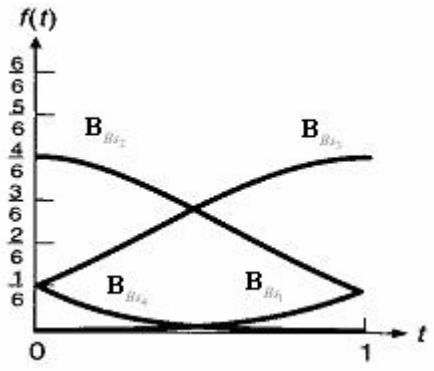
$$X_3(t) = \frac{1}{6}t^3$$

Mátrix-reprezentációban az eredmény:

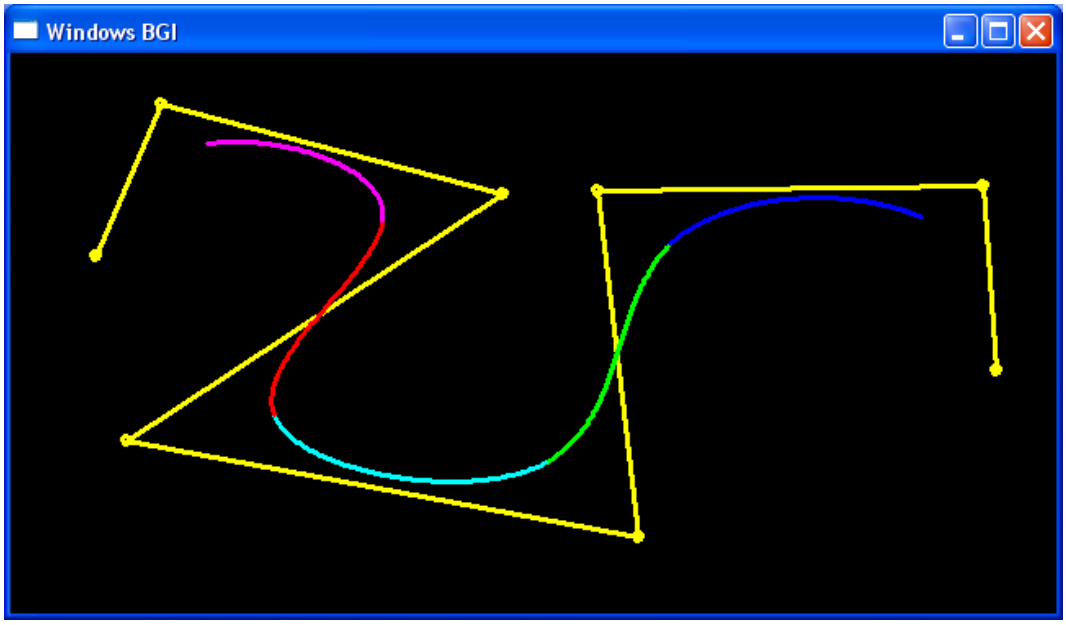
$$\mathbf{Q}(t) = \mathbf{G}_{Bz} \bullet \mathbf{M}_{Bz} \bullet \mathbf{T} = \mathbf{G}_{Bz} \bullet \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} =$$

$$\frac{1}{6}(\mathbf{P}_1 \bullet (1-t)^3 + \mathbf{P}_2 \bullet (3t^3 - 6t^2 + 4t) + \mathbf{P}_3 \bullet (-3t^3 + 3t^2 + 3t + 1) + \mathbf{P}_4 \bullet t^3)$$

Az $\mathbf{B}_{Bs_1} = \frac{1}{6}(1-t)^3, \mathbf{B}_{Bs_2} = \frac{1}{6}(3t^3 - 6t^2 + 4t), \mathbf{B}_{Bs_3} = \frac{1}{6}(-3t^3 + 3t^2 + 3t + 1), \mathbf{B}_{Bs_4} = \frac{1}{6}t^3$ súlyfüggvényeket koordinátarendszerben ábrázolva:



Az alábbi ábrán az egymást követő szegmenseket eltérő színnel rajzoltuk.



4.8 Beziér-görbe előállítása Berstein polinommal

Legyen n>0, i=0,1,…n. Berstein polinomnak nevezzük a következő alakú n-ed fokú polinomokat:

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

A Berstein polinomokra igaz a következő azonosság:

$$\sum_{i=0}^{n} B_i^n(t) = 1,\ t \in \mathbf{R}$$

Ennek igazolása a binomiális tétel alapján nyilvánvaló:

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

Legyen adott a $\mathbf{P_0}, \mathbf{P_1}, ..., \mathbf{P_n}$ pontsorozat. A $\mathbf{Q}(t) = \sum_{i=0}^{n} \mathbf{P_i} B_i^n(t),\ t \in [0,1]$ görbét Beziér-görbének nevezzük. A harmadrendű Beziér görbe speciális esetként adódik.

A Bézier ívek kontrollpontjai affin transzformációjával szemben invariánsak. Ez azt jelenti, hogy például a görbe mozgatásakor elegendő a kontrollpontokat transzformálni, mellyel igen sok számítás megtakarítható. A Bézier-ívek a paramétertartomány affin transzformációira is invariánsak. A Bézier-ívek globálisan változtathatók, azaz ha a kontrollpontok közül egyet elmozgatunk, az az egész görbére kihat. A görbe áthalad a kezdő és végponton.

4.8.1 A kontrollpontok multiplicitása

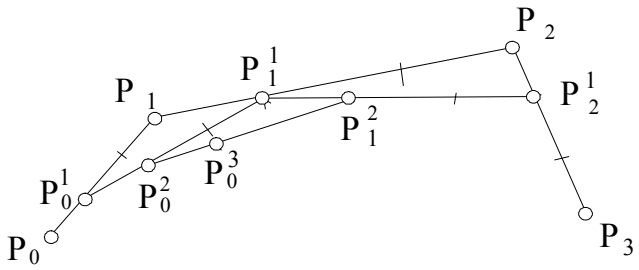
A $\mathbf{P_0}, \mathbf{P_1}, ..., \mathbf{P_n}$ közül k egymást követő pont megegyezhet, azaz $\mathbf{P}_i = \mathbf{P}_{i+1} = ... = \mathbf{P}_{i+k-1}, (i > 0,\ i \le n-k+1)$. Ekkor azt mondjuk, hogy a $\mathbf{P}_i$ kontrollpont multiplicitása k. A görbe alakulásában az ilyen pontok nagyobb súllyal vesznek részt.

4.8.2 Bézier-görbe előállítása de Casteljau-algoritmussal

Legyen adva egy n-edrendű Bézier-ív a $\mathbf{P_0}, \mathbf{P_1}, ..., \mathbf{P_n}$ kontrollpontokkal. Definiáljuk a következő rekurzív összefüggést:

$$\mathbf{P}_i^r(t) = (1-t)\mathbf{P}_i^{r-1} + t\mathbf{P}_{i+1}^{r-1}, \text{ahol}$$
$$r = 1, 2, \ldots, n; \quad \text{és} \quad i = 0, 1, \ldots, n - r,$$
$$\text{valamint } \mathbf{P}_i^0 = \mathbf{P}_i, \; i = 0, 1, \ldots, n$$

Ezzel a rekurzív képlettel kapott $\mathbf{P}_i^r(t)$ pont a t paraméterértékhez tartozó görbepont, amit a gyakorlatban szakaszok adott arányú felosztásával kapunk meg. Vagyis a $\mathbf{P}_i^r(t)$-nek megfelelő pont a $\mathbf{P}_i^{r-1}(t)$ és $\mathbf{P}_{i+1}^{r-1}(t)$-nek megfelelő pontokat összekötő szakasz egy belső pontja, melyet az 1-t és t súlyok jelölnek ki. Vagyis például t=0.5-nél a kontroll poligon oldalainak felezőpontjait kötjük össze, majd a kapott szakaszok felezőpontjait, stb.. n=4 esetben a rekurziónak a 3. lépésnél van vége.



Egy konkrét t_0 értéket és négy kontrollpontot választva a harmadik felosztás már görbepontot eredményez. Természetesen belátható, hogy ez az előállítás ugyanazt a görbét eredményezi, mint a Berstein-polinomokkal való számolás.

$\mathbf{P}_0$

$\mathbf{P}_0^1$

$\mathbf{P}_1$ $\mathbf{P}_0^2$

$\mathbf{P}_1^1$ $\mathbf{P}_0^3$

$\mathbf{P}_2$ $\mathbf{P}_1^2$

$\mathbf{P}_2^1$

$\mathbf{P}_3$

A Bézier-görbe a megadott pontok közül átmegy a kezdő- és végponton, azaz interpolálja azokat. Ezen pontokban az érintő egyenese megegyezik a kontrollpoligon első, illetve utolsó szakaszának egyenesével. Azt, hogy a görbe mennyire jól aproximálja az adott pontokat, az úgynevezett konvex burok tulajdonság mutatja.

Egy ponthalmaz konvex burkán az ezen pontokat tartalmazó konvex ponthalmazok metszetét értjük, lényegében a legkisebb olyan konvex alakzatot, melyben mindegyik pont benne van. A konvex burok tulajdonság azt mondja ki, hogy a görbe sosem hagyja el az őt definiáló kontrollpontok konvex burkát.

4.8.3 Interpoláló Bézier-görbe előállítása

Adottak a $\mathbf{P_0}, \mathbf{P_1}, ..., \mathbf{P_n}$ kontrollpontok, valamint a $t_0, t_1, ..., t_n$ (különböző) paraméterek. Határozzuk meg a $\mathbf{B_0}, \mathbf{B_1}, ..., \mathbf{B_n}$ pontokat, hogy az általuk előállított $\mathbf{Q}(t) = \sum_{i=0}^{n} \mathbf{B_i} B_i^n(t),\ t \in [0,1]$ Bézier görbe áthaladjon az adott pontokon, azaz:

$$\mathbf{P}_j = \sum_{i=0}^{n} \mathbf{B_i} B_i^n(t_j),\ j = 0,1,...,n$$

Ugyanez mátrix reprezentációban:

$$\begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ . \\ . \\ . \\ \mathbf{P}_n \end{bmatrix} = \begin{bmatrix} B_0^n(t_0) & B_1^n(t_0) & . & . & . & B_n^n(t_0) \\ B_0^n(t_1) & B_1^n(t_1) & . & . & . & B_n^n(t_1) \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & . \\ B_0^n(t_n) & B_1^n(t_n) & . & . & . & B_n^n(t_n) \end{bmatrix} \bullet \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \\ . \\ . \\ . \\ \mathbf{B}_n \end{bmatrix}$$

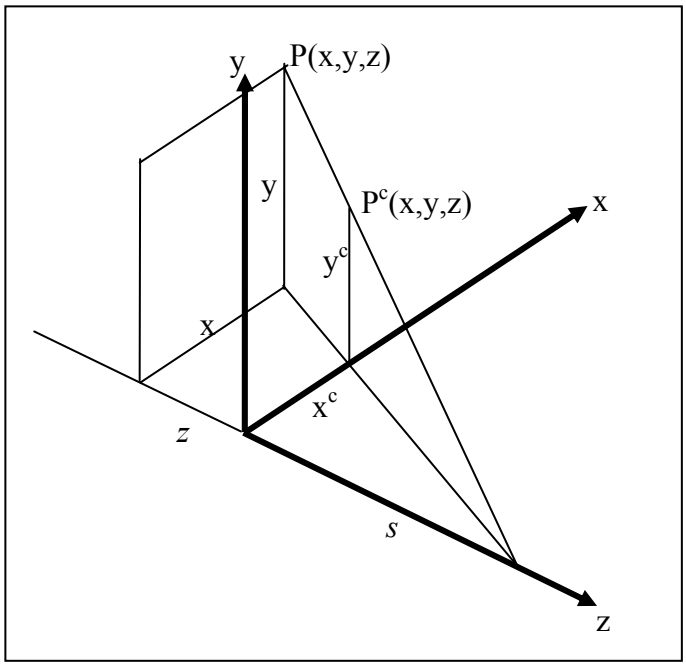
A fenti inhomogén lineáris egyenletrendszert (minden koordinátára vonatkozóan) megoldva kapjuk a keresett pontokat. A lineáris egyenletrendszer mindig megoldható, mivel igazolható, hogy a rendszerdeterminánsnem nulla.

5 Tér síkra való leképezései

Ha raszteres képpé konvertálva a 3D-s tárgyakat meg akarjuk jeleníteni a monitor képernyőjén, akkor ezeket a 3D-s modelltérből egy 2D-s nézetre kell leképezni. Ezért a számítógépes grafikában kiemelt jelentőségű transzformációk a vetítések. Vetítésnek nevezzük azokat a dimenzióveszteséggel járó pont-transzformációkat, amelyeknél a képpont és a neki megfelelő tárgypont egy egyenesen helyezkedik el. A tárgy- és képpontokon áthaladó egyenest vetítősugárnak nevezzük. A vetítés eredménye a vetület, ami a képsíkon képződik. Az egyes tárgypontok képe a vetítősugarak döféspontja a képsíkkal. A vetítés két alaptípusa a párhuzamos és a középpontos vetítés. Párhuzamos vetítésről beszélünk, ha a vetítősugarak egymással párhuzamosak. Ha ezen kívül a vetítősugarak még merőlegesek is a képsíkra, akkor merőleges a vetítés, egyébként pedig a ferde vetítés elnevezést használjuk. Középpontos vetítés esetén a vetítősugarak mindegyike áthalad a vetítési középponton, a centrumon. Perspektivikus hatás elsősorban a tárgy és a centrum és a pont távolságától függ. Ha ez a távolság minden határon túl nő, a középpontos vetítés párhuzamos vetítésbe megy át.

5.1 Centrális vetítés

A képsík legyen a Descartes-féle koordinátarendszer {x,y} koordináta síkja, a vetítési centrumot helyezzük a z-tengelyre, az origótól s távolságra. A



megfelelő hasonló háromszögekből: $x^c = x\frac{s}{s-z}; y^c = y\frac{s}{s-z};$

Ugyan ez homogén koordinátákkal mátrix-reprezentációban:

$$\begin{bmatrix} x^c \\ y^c \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{s} & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1-\frac{z}{s} \end{bmatrix} = \begin{bmatrix} x\frac{s}{s-z} \\ y\frac{s}{s-z} \\ 0 \\ 1 \end{bmatrix}$$

5.2 Axonometria

Megadjuk a képsíkon a tengelykereszt képét.

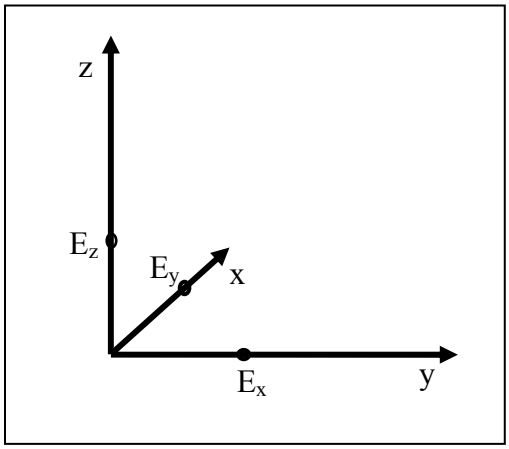
$$E_x'(a_{11}, a_{21}), E_y'(a_{12}, a_{22}), E_z'(a_{13}, a_{23})$$

$$P(x, y, z) \to P'(u, v)$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + a_{13}z \\ a_{21}x + a_{22}y + a_{23}z \end{bmatrix}$$

5.2.1 Kavalier axonometria:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -q\cos\alpha & 1 & 0 \\ -q\sin\alpha & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



- q: rövidülés a x tengelyen

- α: x tengely képének y tengely képével bezárt szöge

5.3 Párhuzamos vetítés

Legyen a vetítés iránya adott a vektorral. Ekkor:

$$P' = P + \lambda \mathbf{v} \text{ alapján}$$

$$x' = x + \lambda v_x, y' = y + \lambda v_y, z' = z + \lambda v_z = 0, \text{amiből}$$

$$\lambda = -\frac{z}{v_z} \rightarrow x' = x - \frac{z}{v_z} v_x, y' = y - \frac{z}{v_z} v_y$$

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

6 Felületek

A térbeli felületeket kétparaméteres r=r(u,v) skalár-vektor fügvényekkel modellezhetjük, ahol $u \in [u_1, u_2]$ és $v \in [v_1, v_2]$ paramétertartománynak. Ez komponensekben felírva:

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

Minden egyes konkrét *u* és *v* paraméterérték a felület egy pontjához vezető helyvektort határoz meg. A térbeli felületeket paramétervonalaikkal ábrázolhatjuk. Azaz megadjuk a felületen azon a pontok halmazát, melyek egy konkrét *u* és *v* értékhez tartoznak. Ez egy térgörbe lesz, amit paramétervonalnak nevezünk.

A vektorgrafikus objektumok képét csak egyenes szakaszokból összerakott alakzatokból tudjuk összeállítani. Ezért a görbék és felületek modellezésében fontos szerepet töltenek be az egymáshoz kapcsolódó egyenes szakaszokból felépített poligonok és a sokszöglapokból álló térbeli alakzatok, a poliéderek, mivel a térbeli görbéket mindig poligonokból, a felületeket pedig sokszöglapokból felépített alakzatokkal közelítjük.

Ha a modellezni kívánt felület $\mathbf{r} = \mathbf{r}(u,v)$ skalár-vektor egyenlete adott, akkor az u illetve v irányú paramétervonalak megjelenítése két-két egymásba ágyazott ciklussal történhet.

6.1 Coons-foltok (felület interpoláció)

Adott 4, egymást páronként metsző térgörbe, melyeknek ismerjük a skalár-vektoros előállítását. Ezek a görbék legyenek:

$$\mathbf{a}_1(u), \mathbf{a}_2(u), u \in [0,1] \text{ és}$$
$$\mathbf{b}_1(v), \mathbf{b}_2(v), v \in [0,1]$$

Ezek egy térbeli négyszöget alkotnak, amelyre illeszteni akarjuk az $\mathbf{r}(u,v)$ felületet, ahol $u,v \in [0,1]$. A felületet úgy kell meghatározni, hogy a határokon pontosan a határoló görbéket adja vissza. Vagyis:

$$\mathbf{r}(u,0) = \mathbf{a}_1(u)$$
$$\mathbf{r}(u,1) = \mathbf{a}_2(u)$$
$$\mathbf{r}(0,v) = \mathbf{b}_1(v)$$
$$\mathbf{r}(1,v) = \mathbf{b}_2(v)$$

A coons-foltot 3 felületből állítjuk elő. Ebből kettő az $\mathbf{a}_1$ és $\mathbf{a}_2$, valamint a $\mathbf{b}_1$ és $\mathbf{b}_2$ által meghatározott vonalfelület, a harmadik pedig egy nyeregfelület.

Vonalfelületek leírása:

Adott $\mathbf{a}_1(u)$ és $\mathbf{a}_2(u)$ térgörbe, melyek ugyanazon az intervallumon kell hogy legyenek definiálva, ez általában: $u \in [0,1]$, de tetszőleges [a,b] intervallum is lehet. A két görbe adott u értékekhez tartozó pontjait kötjük össze, azaz a paramétervonalak egyenes szakaszok. A két térgörbe által kifeszített $\mathbf{I}_\mathbf{a}(u,v)$, ahol $u,v \in [0,1]$ felület egyenesekből áll és igaz rá, hogy

$$\mathbf{I}_{\mathbf{a}}(u,0)=\mathbf{a}_1(u),\ \text{és}\ \mathbf{I}_{\mathbf{a}}(u,1)=\mathbf{a}_2(u)$$

A felület az $\mathbf{a}_1(u)$ és $\mathbf{a}_2(u)$ térgörbék lineáris interpolációja, melynek egyenlete:

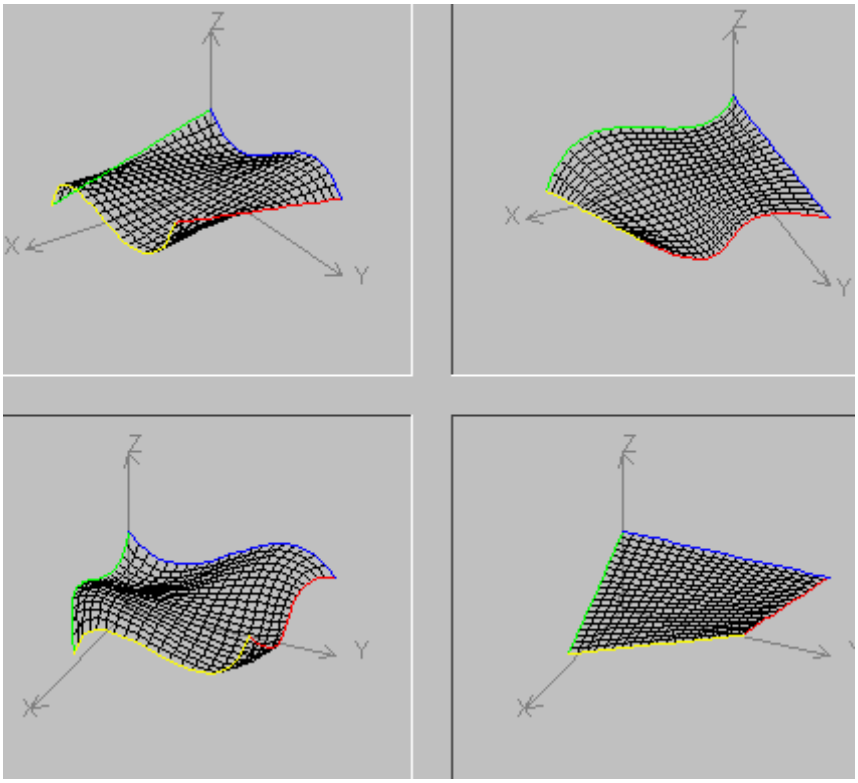
$$\mathbf{I}_{\mathbf{a}}(u,v)=(1-v)\mathbf{a}_1(u)+v\mathbf{a}_2(u)$$

A felület az $\mathbf{a}_1(u)$ és $\mathbf{a}_2(u)$, egymással szemben fekvő két görbét interpolálja, de a másik két határoló görbén $\mathbf{b}_1(v),\mathbf{b}_2(v)$-n nyilván nem halad át.A $\mathbf{b}_1(v),\mathbf{b}_2(v)$ által meghatározott vonalfelület hasonlóképpen állítható elő. Ennek egyenlete: $\mathbf{I}_{\mathbf{b}}(u,v)=(1-u)\mathbf{b}_1(v)+u\mathbf{b}_2(v)$

A négy görbe metszéspontjai négy pontot határoznak meg a térben, melyek meghatároznak egy nyeregfelületet, a kitérő egyenesek affin pontsorai megfelelő elemeinek összekötése által. A négy metszéspont bilineáris interpolációja:

$$\mathbf{I}_{\mathbf{ab}}(u,v)=\begin{bmatrix}1-u & u\end{bmatrix}\begin{bmatrix}\mathbf{r}(0,0) & \mathbf{r}(0,1)\\ \mathbf{r}(1,0) & \mathbf{r}(1,1)\end{bmatrix}\begin{bmatrix}1-v\\ v\end{bmatrix}$$

A coons-foltot a 3 felületből úgy kapjuk meg, hogy a két vonalfelület összegéből kivonjuk a nyeregfelületet: $\mathbf{r}(u,v)=\mathbf{I}_{\mathbf{a}}(u,v)+\mathbf{I}_{\mathbf{b}}(u,v)-\mathbf{I}_{\mathbf{ab}}(u,v)$



Azaz:

$$\mathbf{r}(u,v) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{r}(0,v) \\ \mathbf{r}(1,v) \end{bmatrix} + \begin{bmatrix} \mathbf{r}(u,0) & \mathbf{r}(u,1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} - \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{r}(0,0) & \mathbf{r}(0,1) \\ \mathbf{r}(1,0) & \mathbf{r}(1,1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}$$

6.2 Bikubikus felületek

A harmadrendű paraméteres görbék $\mathbf{Q}(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}^T = \mathbf{G} \bullet \mathbf{M} \bullet \mathbf{T}$ előállításából indulunk ki. **Q**-t sorvektorként tekintve:

$$\mathbf{Q}(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix} = \mathbf{T}^T \bullet \mathbf{M}^T \bullet \mathbf{G}^T .$$

G-t magát is harmadfokú paraméteres függvényként állítjuk elő, akkor bikubikus felületet kapunk.

$$\mathbf{r}(u,v) = \mathbf{U} \cdot \mathbf{M}^T \cdot \mathbf{G}(v) = \mathbf{U} \cdot \mathbf{M}^T \cdot \begin{bmatrix} \mathbf{G}_1(v) \\ \mathbf{G}_2(v) \\ \mathbf{G}_3(v) \\ \mathbf{G}_4(v) \end{bmatrix}, \text{ ahol } \mathbf{U} = \begin{bmatrix} u^3 u^2 u 1 \end{bmatrix}^T$$

$$\text{és } \mathbf{G}_i(v) = \mathbf{V}^T \cdot \mathbf{M}^T \cdot \mathbf{G}_i, \text{ ahol } \mathbf{G}_i = \begin{bmatrix} g_{i1} g_{i2} g_{i3} g_{i4} \end{bmatrix}^T, \mathbf{V} = \begin{bmatrix} v^3 v^2 v 1 \end{bmatrix}^T$$

$$(\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C})^T = \mathbf{C}^T \mathbf{B}^T \mathbf{A}^T \text{ alapján}$$

$$\mathbf{G}_i(v) = \mathbf{G}_i^T \cdot \mathbf{M} \cdot \mathbf{V} = \begin{bmatrix} g_{i1} g_{i2} g_{i3} g_{i4} \end{bmatrix} \cdot \mathbf{M} \cdot \mathbf{V}$$

$$\mathbf{r}(u,v) = \mathbf{U}^T \cdot \mathbf{M}^T \cdot \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot \mathbf{M} \cdot \mathbf{V} \text{ vagy}$$

$$\mathbf{r}(u,v) = \mathbf{U}^T \cdot \mathbf{M}^T \cdot \mathbf{G} \cdot \mathbf{M} \cdot \mathbf{V}$$

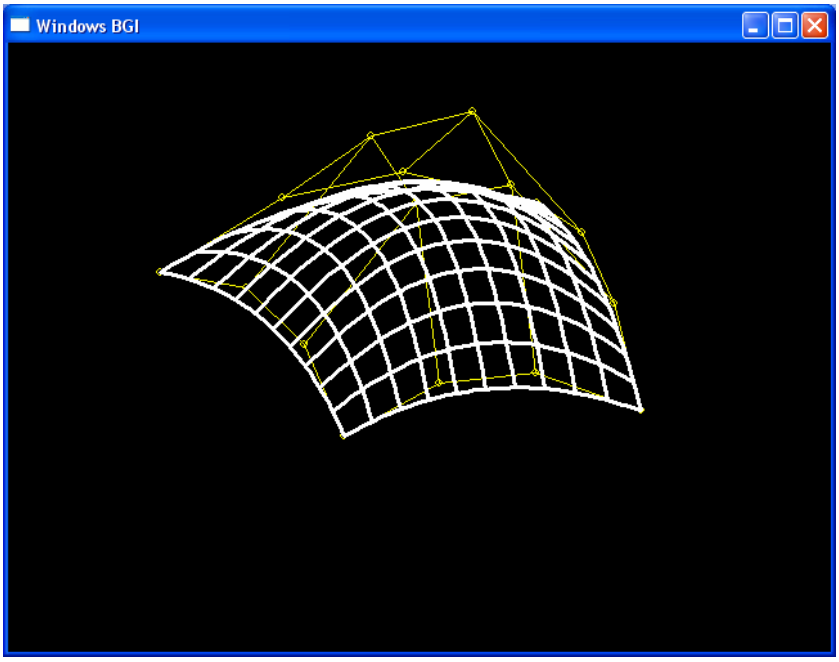
$$x(u,v) = \mathbf{U}^T \cdot \mathbf{M}^T \cdot \mathbf{G}_x \cdot \mathbf{M} \cdot \mathbf{V}$$

$$y(u,v) = \mathbf{U}^T \cdot \mathbf{M}^T \cdot \mathbf{G}_y \cdot \mathbf{M} \cdot \mathbf{V}$$

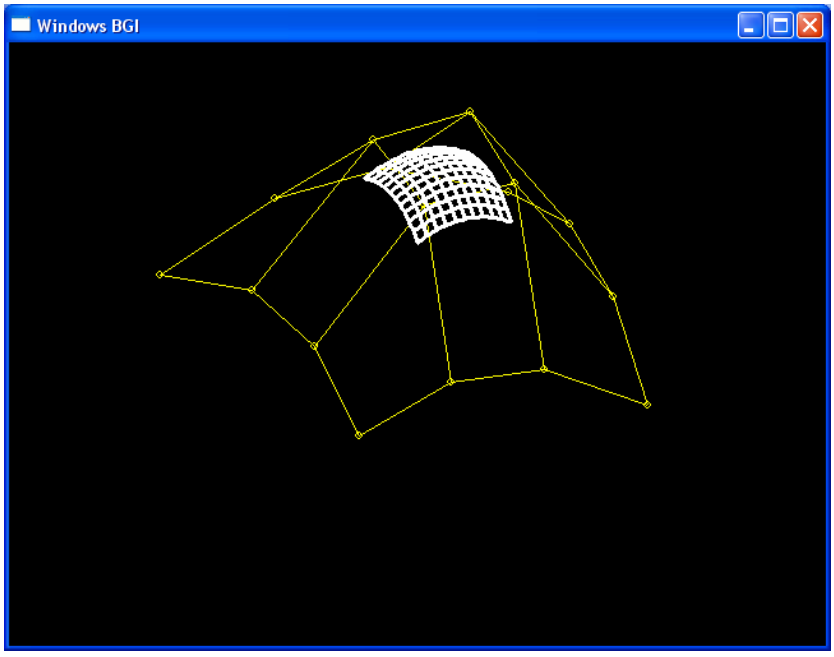
$$z(u,v) = \mathbf{U}^T \cdot \mathbf{M}^T \cdot \mathbf{G}_z \cdot \mathbf{M} \cdot \mathbf{V}$$

A fenti módon előálló felületek közül a B-spline és Bezier felületek esetében a geometriai adatok 16 kontrolpontot jelentenek. Hasonló a helyzet, ha 16 pontot interpoláló Hermite-féle felületet állítunk elő. A két pont és két érintő alapján előálló Hermite ívhez tartozó alapmátrix esetében a geometriai adatok jelentése

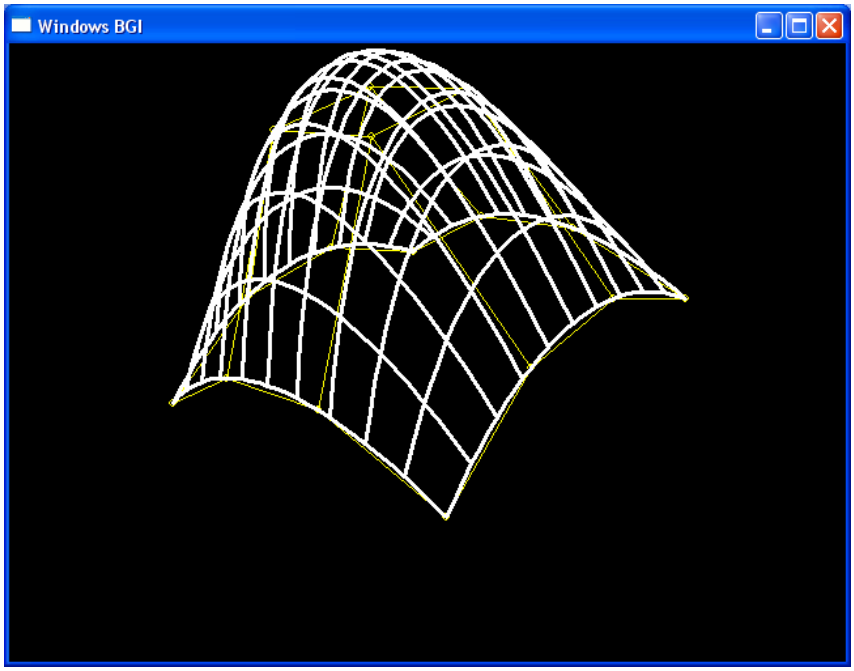
összetettebb. A felület ábrázolásakor érdemes a $\mathbf{M}^T \cdot \mathbf{G} \cdot \mathbf{M}$ szorzatot előre, a szükséges dupla cikluson kívül kiszámítani. Mindkét irányú paramétervonalat egy-egy egymásba ágyazott ciklussal számítjuk ki.



Bezier felület



B-spline felület



Hermite felület 16 pontra

6.2.1 Hermite felület

Induljunk ki a Hermite-görbe alapmátrixából, mely két pont két érintő adatokhoz tartozik.

$$x(u,v) = \mathbf{U}^T \cdot \mathbf{M}_H^T \cdot \mathbf{G}_{H_x}(v) = \mathbf{U}^T \cdot \mathbf{M}_H^T \cdot \begin{bmatrix} \mathbf{P_1}(v) \\ \mathbf{P_4}(v) \\ \mathbf{R_1}(v) \\ \mathbf{R_4}(v) \end{bmatrix}_x ,$$

$$\text{ahol } \mathbf{U} = \left[u^3\, u^2\, u\, 1\right]^T, \mathbf{V} = \left[v^3\, v^2\, v1\right]^T$$

A $\mathbf{P_1}(v), \mathbf{P_4}(v), \mathbf{R_1}(v), \mathbf{R_4}(v)$
függvények előállítása (x komponenseik):

$$\mathbf{P}_{1x}(v) = \left[g_{11} \quad g_{12} \quad g_{13} \quad g_{14}\right]_x \cdot \mathbf{M}_H \cdot \mathbf{V}$$
$$\mathbf{P}_{2_x}(v) = \left[g_{21} \quad g_{22} \quad g_{23} \quad g_{24}\right]_x \cdot \mathbf{M}_H \cdot \mathbf{V}$$
$$\mathbf{R}_{1x}(v) = \left[g_{31} \quad g_{32} \quad g_{33} \quad g_{34}\right]_x \cdot \mathbf{M}_H \cdot \mathbf{V}$$
$$\mathbf{R}_{4x}(v) = \left[g_{41} \quad g_{42} \quad g_{43} \quad g_{44}\right]_x \cdot \mathbf{M}_H \cdot \mathbf{V}$$

$$\begin{bmatrix} \mathbf{P_1}(v) \\ \mathbf{P_4}(v) \\ \mathbf{R_1}(v) \\ \mathbf{R_4}(v) \end{bmatrix}_x = \mathbf{G}_{H_x} \cdot \mathbf{M}_H \cdot \mathbf{V}, \text{ ahol } \mathbf{G}_{H_x} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}_x$$

$$\begin{bmatrix} \mathbf{P_1}(v) \\ \mathbf{P_4}(v) \\ \mathbf{R_1}(v) \\ \mathbf{R_4}(v) \end{bmatrix}_x = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}_x \mathbf{M}_H \cdot \mathbf{V} = \mathbf{G}_{H_x} \mathbf{M}_H \cdot \mathbf{V}$$

$$x(u,v) = \mathbf{U}^T \cdot \mathbf{M}_H^T \cdot \mathbf{G}_{H_x} \cdot \mathbf{M}_H \cdot \mathbf{V}$$

$$\mathbf{G}_{H_x} = \begin{bmatrix} x(0,0) & x(0,1) & \frac{\partial}{\partial v}x(0,0) & \frac{\partial}{\partial v}x(0,1) \\ x(1,0) & x(1,1) & \frac{\partial}{\partial v}x(1,0) & \frac{\partial}{\partial v}x(1,1) \\ \frac{\partial}{\partial u}x(0,0) & \frac{\partial}{\partial u}x(0,1) & \frac{\partial}{\partial u \partial v}x(0,0) & \frac{\partial}{\partial u \partial v}x(0,1) \\ \frac{\partial}{\partial u}x(1,0) & \frac{\partial}{\partial u}x(1,1) & \frac{\partial}{\partial u \partial v}x(1,0) & \frac{\partial}{\partial u \partial v}x(1,1) \end{bmatrix}$$

A $\mathbf{G}_H$ első oszlopában szereplő adatok határozzák meg az u irányú paramétervonalat v=0 mellett. Az első sorvektor hasonlóan a v irányú paramétervonalat határozza meg. Fontos szerepe van a felület alakjának alakításában a jobb alsó 2x2 kettes mátrixnak. Ezek az úgynevezett twist vektorok, melyeknek a Hermite-féle foltok egymáshoz illesztésében van szerepük.

6.2.2 Felületi normálisok

Általában gyakorta szükséges a felületek valamely pontjában a felületi normálisok kiszámítása. A felületi normálist a paramétervonalak adott pontbeli érintőinek vektoriális szorzataként számítjuk ki.

$$\frac{\partial}{\partial u}\mathbf{r}(u,v)=\frac{\partial}{\partial u}(\mathbf{U}^T\cdot\mathbf{M}^T\cdot\mathbf{G}\cdot\mathbf{M}\ \cdot\mathbf{V})=\frac{\partial}{\partial u}(\mathbf{U}^T)\cdot\mathbf{M}^T\cdot\mathbf{G}\cdot\mathbf{M}\ \cdot\mathbf{V}=$$

$$\left[3u^2\ 2u\,1\ 0\right]\cdot\mathbf{M}^T\cdot\mathbf{G}\cdot\mathbf{M}\ \cdot\mathbf{V}=\left[x_u\,y_u\,z_u\right]$$

$$\frac{\partial}{\partial v}\mathbf{r}(u,v)=\frac{\partial}{\partial v}(\mathbf{U}^T\cdot\mathbf{M}^T\cdot\mathbf{G}\cdot\mathbf{M}\ \cdot\mathbf{V})=\mathbf{U}^T\cdot\mathbf{M}^T\cdot\mathbf{G}\cdot\mathbf{M}\ \cdot\frac{\partial}{\partial u}(\mathbf{V})=$$

$$\mathbf{U}^T\cdot\mathbf{M}^T\cdot\mathbf{G}\cdot\mathbf{M}\ \cdot\begin{bmatrix}3v^2\\2v\\1\\0\end{bmatrix}=\left[x_v\,y_v\,z_v\right]$$

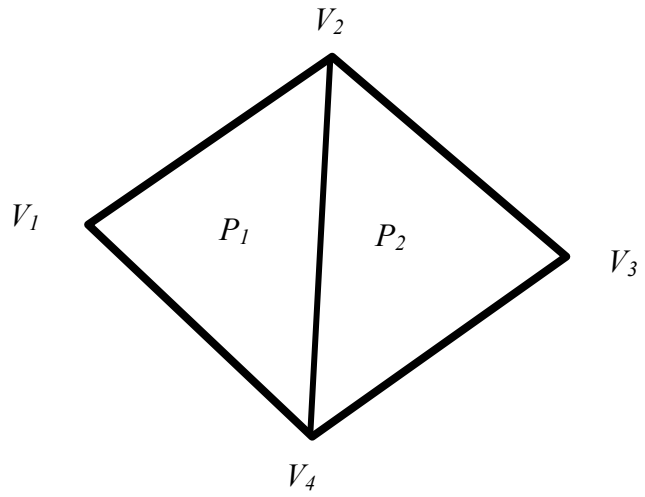
$$\frac{\partial}{\partial u}\mathbf{r}(u,v)\times\frac{\partial}{\partial v}\mathbf{r}(u,v)=\left[\left(y_u z_v-y_v z_u\right)\ \ \left(z_u x_v-z_v x_u\right)\ \ \left(x_u y_v-x_v y_u\right)\right]$$

6.3 Poligonokkal határolt felületek

6.3.1 Explicit reprezentáció

Minden poligont csúcsainak koordinátáival adunk meg, az egymásra következő csúcsok, az első és az utolsó adják az éleket. Redundáns adattárolás, mivel egy csúcs annyiszor szerepel, ahány poligonhoz tartozik. Ha egy *P* poligonnak *n* csúcsa van, akkor az alábbi módon tároljuk a csúcsokat:

$$P=\left(\left(x_1\quad y_1\quad z_1\right),\left(x_2\quad y_2\quad z_2\right),...,\left(x_n\quad y_n\quad z_n\right)\right)$$



A fenti ábrához tartozó adatszerkezetben két poligont adunk meg, mindegyikben 3-3 csúcsot sorolunk fel. A V_2 és V_4 csúcsok kétszer szerepelnek a listában.

6.3.2 Mutatók a csúcslistába

Két listában tároljuk az adatokat. Az egyik lista a csúcsok listája, melyben megadjuk az összes csúcsot. Ezek az úgynevezett geometriai adatok. A másik listában felsoroljuk a poligonokat, ahol a listaeelemek mutatók (vagy esetleg indexek) és a csúcslista megfelelő elemeire mutatnak. Ezek a topológiai adatok.

Csúcsok listája:

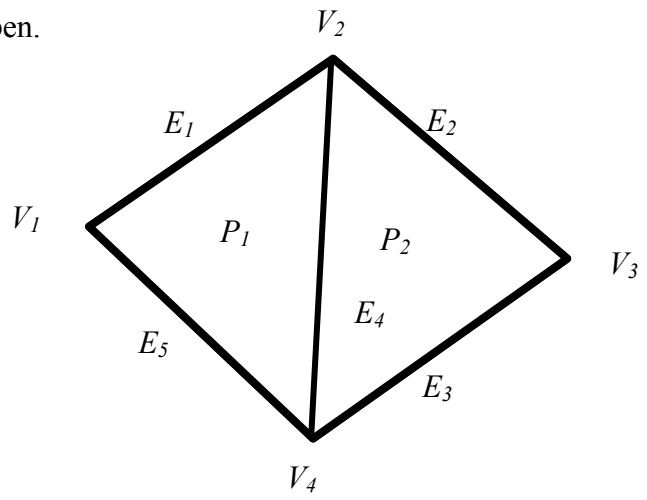
$$V = \left(\left(x_1 \quad y_1 \quad z_1 \right), \left(x_2 \quad y_2 \quad z_2 \right), ..., \left(x_n \quad y_n \quad z_n \right) \right)$$

Poligonok listája (pl. k darab poligon estén):

$$P_1 = \left(i_1^1, i_2^1, ..., i_{m_1}^1 \right)$$
$$P_2 = \left(i_1^2, i_2^2, ..., i_{m_2}^2 \right)$$
$$\vdots$$
$$P_k = \left(i_1^k, i_2^k, ..., i_{m_k}^k \right)$$

6.3.3 Mutatók az éllistába

A csúcslistán kívül megadunk egy éllistát és a poligonok listáját is. Az élek listájában két mutató a csúcslista 1-1 elemére mutat, melyek az él két végpontját határozza meg. Másik két mutató pedig a poligonok listájába mutat, arra a legfeljebb két poligonra, melyek az adott élben találkoznak. Szélső él esetén az egyik mutató null is lehet. A struktúra akkor is hasonló, ha több mint két poligon találkozik egy élben.



A poligonok listájában szereplő mutatók az éllistára mutatnak, a körüljárásnak megfelelően felsorolva az adott poligonhoz tartozó éleket.

A fenti ábrához tartozó struktúra az alábbi:

$$V=(V_1,V_2,V_3,V_4)=((x_1 \quad y_1 \quad z_1),(x_2 \quad y_2 \quad z_2),...,(x_4 \quad y_4 \quad z_4))$$
$$E_1=(1,2,1,\lambda)$$
$$E_2=(2,3,2,\lambda)$$
$$E_3=(3,4,2,\lambda)$$
$$E_4=(4,2,1,2)$$
$$E_5=(4,1,1,\lambda)$$
$$P_1=(1,4,5)$$
$$P_2=(2,3,4)$$

Az élek esetén az első két index a csúcslistába, a második kettő pedig a poligonok listájába mutat.

Általánosan, ha *n* csúcs, *k* poligon és *m* él van:

$$V=((x_1 \quad y_1 \quad z_1),(x_2 \quad y_2 \quad z_2),...,(x_n \quad y_n \quad z_n))$$
$$E_i=(V_1^i,V_2^i,P_1^i,P_2^i),i=1,2,..,m$$
$$P_j=(E_1^j,E_2^j,...,E_{p_j}^j),j=1,2,..,k$$

Bármely poligon éleinek a száma eltérő lehet, ebben az esetben dinamikus adatszerkezetre van szükség. Ha egyező élszámú poligonokról van szó, (pl háromszögek) akkor az adatszerkezet egyszerűbb is lehet.

7 Testmodellezés

7.1 Reguralizált halmazműveletek

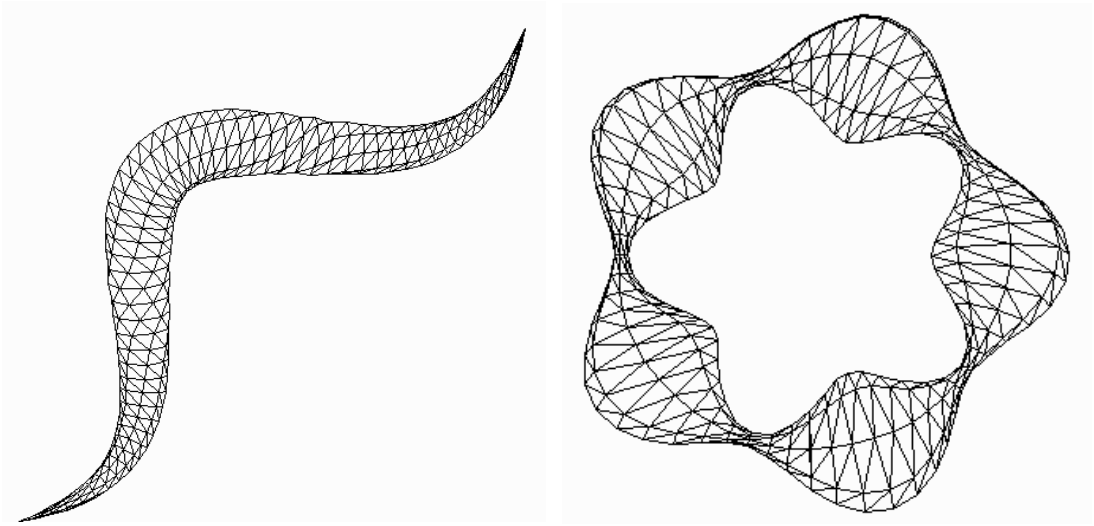
Testek modellezésekor előfordulhat, hogy a szokásos értelemben vett halmazműveletek eredménye nem test lesz. Például poliéderek esetén metszet vagy különbség képzéskor az eredmény lehet egy lap is. Ezt elkerülendő, bevezetjük a regularizált halmazművelet fogalmát, amit a * szimbólummal jelölünk, és a következő módon értelmezünk:

$$Aop^{*}B = \overline{\mathrm{int}\left(AopB\right)}$$

Az *op* szimbólum jelöli az unió, metszet, különbség képzés valamelyikét, *int* pedig a belső pontok halmazát. Azaz vesszük a szokásos halmazműveletet, az eredmény belső pontjait, majd a lezártat.

7.2 Sepert testek (SWEEP representation)

Mozgassunk a térben egy objektumot folytonosan valamilyen pályán. A mozgás során az objektum egy térrészt súrol, ami egy új objektumként fogható fel. A legegyszerűbb eset az, amikor egy 2d-s zárt tartományt mozgatunk például egy szakasz mentén önmagával párhuzamosan, ekkor hengert (poligon mozgatása esetén hasábot) kapunk. Hasonlóan felületek is előállíthatók, ekkor nyílt alakzatot (görbét) mozgatunk. A mozgó objektumot 2d-s esetben generátor görbének, a mozgás pályáját vezérgörbének nevezzük. Megengedett a generátor görbe alakváltozása is. A mozgás gyakran kötődik a vezérgörbe kísérő triéderéhez, ami a görbe egy pontjában értelmezett ortonormált bázis. Az érintő vektor, fő normális és binormális vektorok alkotják. Bizonyos görbék esetén (pl. splinok) könnyen számítható.

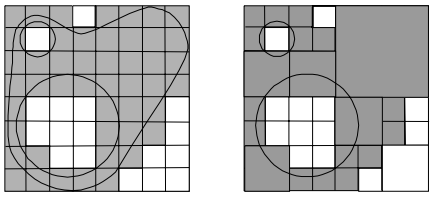


7.3 Felületmodell (Boundary representation, B-rep)

A legismertebb és leggyakrabban használt módszer testek modellezésére. A 6.3 fejezetben leírt adatstruktúra alapján poliédereket definiálunk, az által, hogy megadjuk a határoló poligonjaikat. A „görbült” testeket gyakran elég kicsi poligonokkal közelítjük. Elvileg a B-rep adatstruktúrában megengedett nem síkfelületek kezelése is. A leggyakoribb a háromszögekre való bontás, mert ezek kezelése gyors. Itt említjük meg a közönséges egyszerű poliéderekre vonatkozó Euler formulát, miszerint $l + c = e + 2$, ahol l a lapok, c a csúcsok és e az élek számát jelenti. Ennek teljesülése a legtöbb modellező rendszer esetében szükséges.

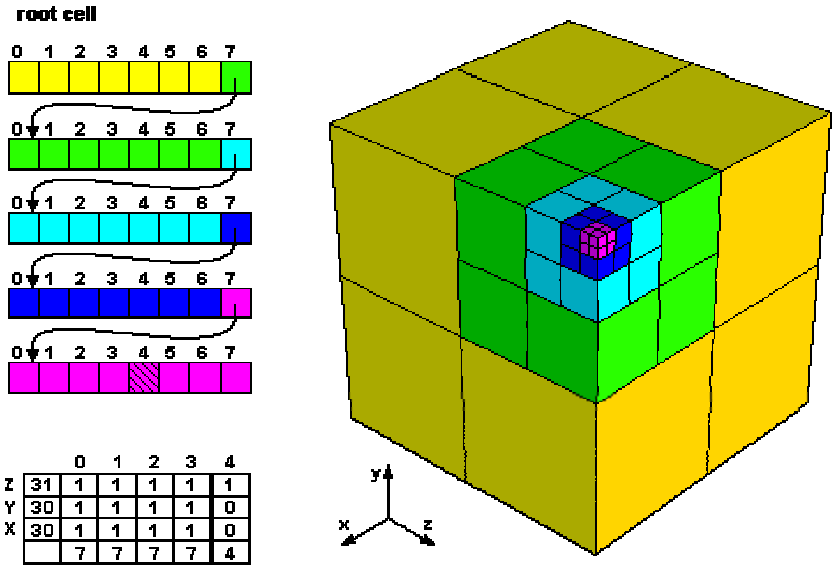
7.4 Cella módszerek (Volume visualisation)

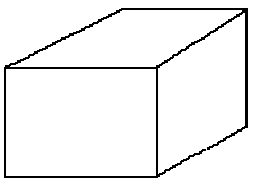
Egységkockákból épül fel a test. Ha a test az egységkocka térfogatának több mint a felét elfoglalja, akkor azt a kockát hozzávesszük a testhez. Előnye, hogy a halmazműveletek nagyon egyszerűen kezelhetőek. Az egységkockák elnevezése *voxel.* A legegyszerűbb adatszerkezet a voxelek tárolására egy háromdimenziós tömb, melynek elemei igaz-hamis értékek lehetnek. Hátrány a nagy memória igény, a megfelelő pontosság eléréséhez kis egységkockákat kell megadnunk. Optimalizálni lehet az adatstruktúrát az Octree felépítéssel. Ez egy speciális fa. Az első csomópont a gyökér cella, mely 8 elemből áll. Az elemek mindegyike egy másik 8 elemű blokkra mutathat. Ez folytatható mindaddig, amíg a szükséges mélységet el nem értük. Az utolsó szint a levélelemek szintje ahol a voxeleket tároljuk.



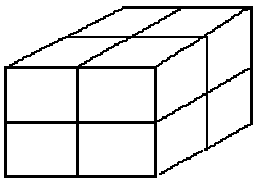
A fenti ábrán 2d-ben szemléltetjük egy objektum két módszerszerinti felbontását. Az a ábrán minden voxel tárolódik, a b ábrát egy quadtree-vel adhatjuk meg. Csak akkor végzünk további felbontást, ha szükséges.

A háromdimenziós adatstruktúrát a következő ábra szemlélteti:

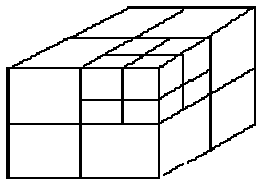




(root)



(1 level)



(2 levels)

7.5 Térfogat modell (CSG: Constructive Solid Geometry)

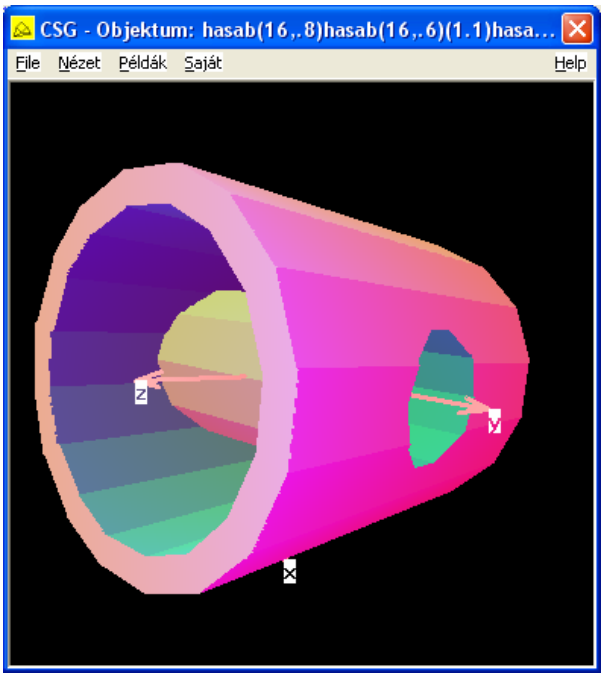
F(x, y, z) típusú implicit egyenletekkel primitívek vannak megadva, melyeken (reguralizált) halmazműveleteket végezve létrehozhatók más testek.

A legtöbb rendszer az alábbi primitíveket használja:

- Gúla
- Hasáb
- Gömb
- Kúp
- Henger
- Tórusz

A primitiveken különböző transzformációk végezhetőek, pl. eltolás, forgatás, skálázás stb. Az alábbi ábra transzformált hasábok uniója és különbségeképpen áll elő.

hasab(16,.8)hasab(16,.6)(1.1)hasab(10,.4){1.57,,}//hasab(10,.3){1.57,,}/



8 Láthatósági algoritmusok

A modelltér objektumaihoz tartozó látható élek és felületek meghatározása, valamint a takart élek és felületek eltávolítása a cél. Ennek lényege, hogy adott 3 dimenziós objektumok és a nézőpont esetén el kell döntenünk, hogy mi látható az alakzatokból a vetítés középpontjából vagy irányából nézve. Ennek eldöntését segítő algoritmusokat takart vonal, illetve takart felület (hidden-line, hidden-surface) meghatározó eljárásoknak nevezik.

8.1 Hátsó lapok eltávolítása (back-face culling)

A zárt, sokszögekből felépülő objektumokat teljesen körbezárják a felületüket alkotó poligonok. Ha a poligonok normálvektorait úgy állítjuk be, hogy az objektumból kifelé mutassanak, akkor azok a poligonok, melyek normálvektorai nem a néző félé mutatnak, biztosan takarva lesznek az objektum közelebbi felületei által. Ezeket a takarásba kerülő, úgymond hátrafelé néző poligonokat back-face poligonoknak nevezzük és az objektumokat leíró adatbázisból való eltávolításukat eredményező eljárást pedig back-face culling-nak. Analóg módon az előre néző poligonokat front-face-nek nevezzük. A back-face poligonok könnyen azonosíthatók kifelé mutató normálvektoruk és lap egy pontjából a centrumba mutató vektor skalárszorzatának vizsgálatával: a negatív érték hátrafelé néző poligonra utal, hiszen ekkor a két vektor szöge nagyobb a derékszögnél. Ha az ábrázolandó jelenet egyetlen, konvex poliéderből áll, akkor a látható felületek meghatározása back-face culling műveletre egyszerűsödik. Más esetekben további vizsgálatok szükségesek, hiszen a front-face poligonok is takarhatják egymást. Egy poligoniális objektumot metsző vetítősugár pontosan annyi back-face poligonon megy át, mint ahány front-face poligonon. A back-face poligonok eltávolítása tehát éppen megfelezi a képpontosság algoritmusa által egy-egy pixel esetén megvizsgálandó poligonok számát. Átlagos esetben a jeleneteket alkotó poligonok nagyjából fele hátrafelé néz, így a back-face culling a tárgypontosság algoritmusok által vizsgálandó poligonok számát is nagyjából megfelezi.

A konvex poliéderek láthatóság szerinti megjelenítéséhez elegendő a hátsó lapokat eltávolítani.

8.2 Robert-féle algoritmus

Konvex poliéderekből álló objektum takarási viszonyinak meghatározására szolgál. Minden konvex objektum hátsó lapjait eltávolítjuk. Meghatározzuk a konvex poliéderek kontúrjait a képen. A kontúrok takarási viszonyainak figyelembevételével megvágjuk a poligonokat (vagy éleket), ha szükséges.

8.3 Listaprioritásos algoritmusok

8.3.1 Mélységi rendező algoritmus (festő algoritmus, Newell, 1972)

Ennek a tárgypontosság algoritmusnak az az alapgondolata, hogy a megjelenítendő objektumokat a nézőponttól való távolság függvényében sorba kell állítani. Az algoritmus a helyes takarási viszonyokat úgy alakítja ki, hogy a nézőhöz közelebb eső objektum képe felülírja a távolabbi objektum képét. Ennek legegyszerűbb változata az úgynevezett festő algoritmus, mely a képfestés technikájához hasonlóan a távolabbi objektumokra ráfesti a közelebbieket. Ezek az algoritmusok általában az objektumok poligonfelületekkel, legtöbbször háromszögekkel való közelítését tételezik fel. Ha egy háromszög z irányban egyértelműen takarja a másikat, akkor a megjelenítésnél ennek képével felül kell írni a távolabbi háromszöget. Ha két háromszög mindegyike takarja a másikat, akkor ezeket úgy kell felbontani részháromszögekre, hogy a takarási viszonyok egyértelműek legyenek A mélység rendező algoritmusokat általában valamilyen képpontosság eljárással együtt alkalmazzák. Ekkor az objektumok sorba rendezése és szükségszerinti feldarabolását követően a pixeles megjelenítés például egy z-bufferrel kombinált scan-line algoritmussal történhet.

Lépések:

- Rendezzük a poligonokat a legkisebb (legtávolabbi pontjaik) z koordinátáik szerint
- ha kell, a poligonok darabolásával az átlapolásokat szüntessük meg
- hátulról előre haladva scan konverzióval fessük ki a poligonokat

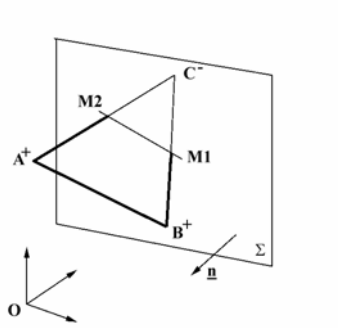
8.3.2 Bináris térfelosztási fa (BSP)

A háromdimenziós objektumok tetszőleges nézőponthoz tartozó megjelenítésére használt algoritmusok közül az. egyik leghatékonyabb a bináris térfelosztó fákat (BSP fák) alkalmazó eljárás. Bár az algoritmusnak az objektumok térbeli helyzetét feltáró, előfeldolgozó része meglehetősen időigényes, később, új nézőpont definiálásakor gyorsan képes előállítani az új képet, hiszen ekkor már csak a frame-bufferbe kell konvertálnia a takaráshelyes objektumokat. A BSP fa algoritmusok azon alapulnak, hogy az ábrázolandó jeleneteket úgy is tekinthetjük, mint objektumcsoportok gyűjteményét. Ha található olyan sík, amely elválaszt egymástól két objektumcsoportot, akkor az a csoport, amely oldalán a nézőpont is van, eltakarhatja a másik csoportot, de a másik csoport által sohasem kerülhet fedésbe. Minden egyes objektumcsoport rekurzívan tovább osztható, ha megfelelő szeparáló síkok találhatók. A jelenet felosztását reprezentálhatjuk olyan bináris fával, melynek gyökere az elsőként választott szeparáló síknál van. A fa belső csomópontjai a szeparáló síkokat jelentik, a fa levelei pedig a felosztás során keletkezett térrészeket. Minden térrészhez hozzárendelhetjük az objektumcsoportok olyan sorrendjét, amely az adott térrészben elhelyezkedő nézőponthoz tartozó helyes takarási viszonyokat rögzíti a megjelenítés során. Bár az algoritmus helyes működése nem függ attól, hogy hogyan választjuk meg a szeparáló síkjainkat, a síkok által végrehajtandó vágások száma azonban jelentősen befolyásolja az algoritmus időigényét. A BSP fa algoritmusok, a mélységi-rendező algoritmusokhoz hasonlóan az objektumok vágását és rendezését tárgypontosság módszerekkel végzik, képpontosság technikákhoz pedig csak a megjelenítés során nyúlnak. Az itt elvégzett objektum felosztásokat, ellentétben a mélységi-rendező algoritmusokkal, csak akkor kell újra elvégezni, ha a jelenet megváltozik. Másként fogalmazva, a nézőpont megváltoztatása esetén nem kell a BSP fát módosítani.

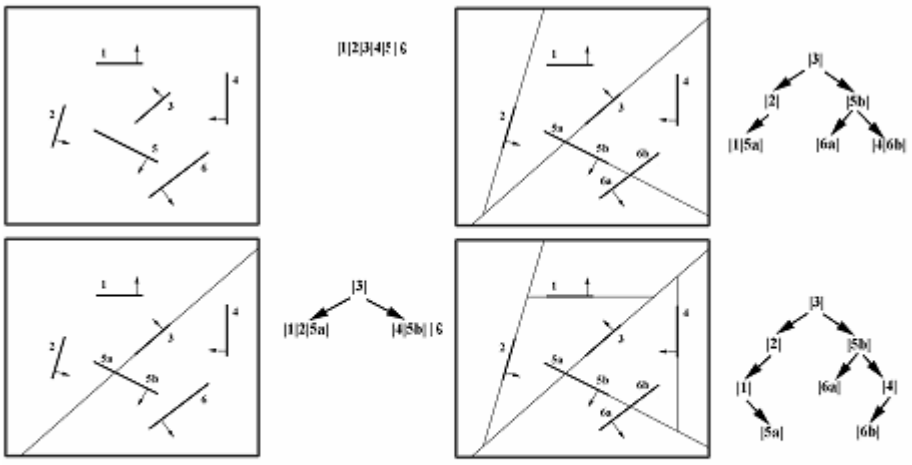
8.3.2.1 A BSP fa elkészítése

Rendelkezésünkre án a feldolgozandó lapok egy listája.

1. Válasszunk ki egy alkalmas lapot a listából. Ez lesz a fa gyökéreleme (gyökérlap).
2. Ha nem maradt több lap (egyelemű volt a lista), akkor kész vagyunk.
3. Majd a maradék lapokat két újabb listába rendezzük aszerint, hogy:
    - A gyökérlap által generált pozitív féltérben helyezkedik-e el -front listába tesszük.
    - - A gyökérlap által generált negatív féltérben helyezkedik-e el -back listába tesszük.
    - A gyökérlap síkja vágja a vizsgált lapot- ekkor a vizsgált lapot vágjuk a gyökérlap síkjával és a -front ill. back listába tesszük ezeket.



4. A két új listára végrehajtjuk az 1.-4. pontokat. Kész.

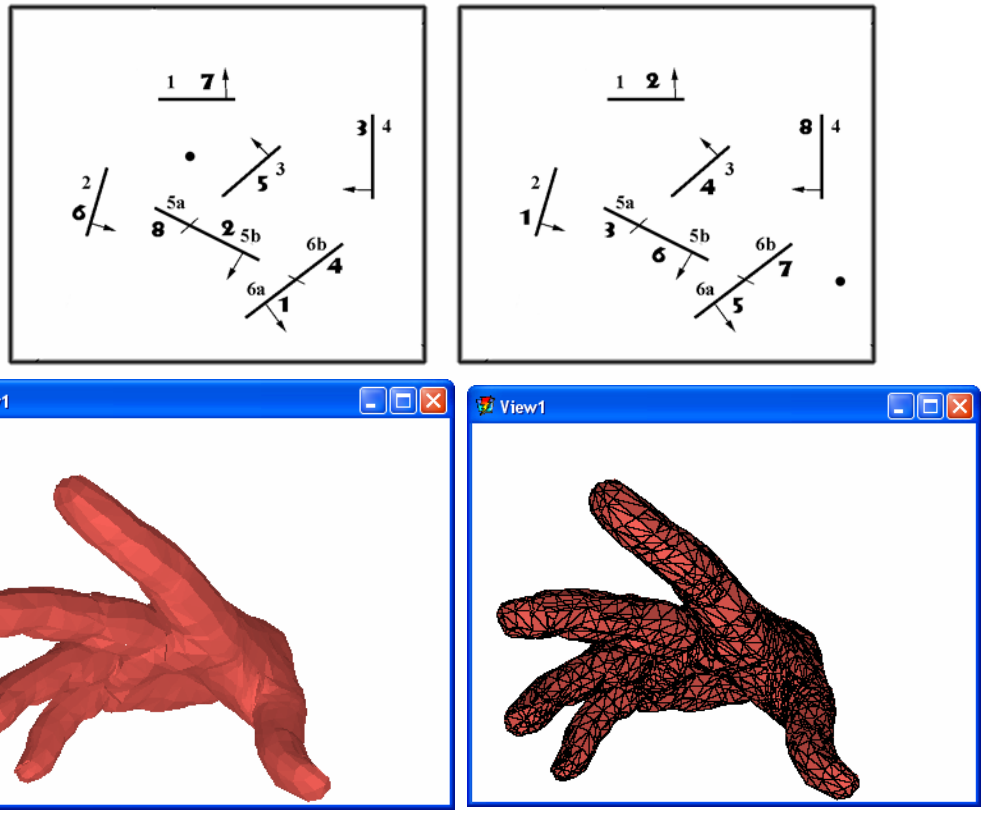


8.3.2.2 A BSP fa bejárása

1. Kiszámoljuk a korábban megismert módszer szerint, hogy a nézőpont a gyökérlap melyik oldalára esik.
2. Ha a gyökérlap által generált pozitív féltérbe esik, akkor:
    - Bejárjuk a back ágat, ha van.
    - Kirajzoljuk az aktuális gyökérlapot.
    - Bejárjuk a front ágat, ha van.
    - Kész.
3. Egyébként:

    - Bejárjuk a front ágat, ha van.
    - Kirajzoljuk az aktuális gyökérlapot.
    - Bejárjuk a back ágat, ha van.
    - Kész.

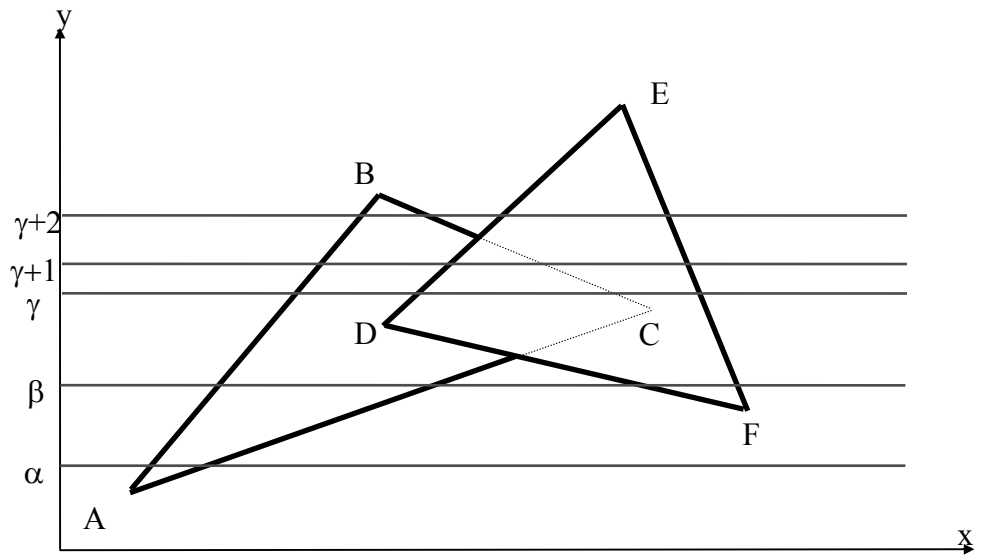
Bonyolultsága O(n), ahol n a BSP fa elemeinek száma. A korábbi példa két különböző nézőpont esetén (A vastagon szedett számok a kirajzolási sorrendet jelentik, a fekete kör pedig a nézőpont):



8.4 Scan-line algoritmusok

A scan-line algoritmusok képpontosság műveleteket használva, pixelsorokként készítik a képet, és a poligonok (háromszögek), frame-bufferbe történő, soronkénti konvertálásának módszerén alapulnak. Az algoritmus, a jelenetet felépítő objektumokról gyűjtött információkat táblázatokban tárolja: a képsíkra vetített, nem vízszintes éleket az éltáblázat, a poligonok fontosabb paramétereit a poligon táblázat, az éppen vizsgált pixelsorhoz tartozó éleket az aktív éltáblázat tartalmazza. Egy-egy pixelsor vizsgálatakor az él táblázat azok az elemei, melyek metszik a sort, áttöltődnek az aktív éltáblázatba a pixelsor és a háromszögek közös részét szegmenseknek nevezzük. Ha a pixelsor egy szegmense csak egy háromszöghöz tartozik, akkor ezt egyszerűen meg kell csak jeleníteni. Ha egy pixel több szegmenshez tartozik, akkor a megfelelő háromszögek mélységi vizsgálatával kell eldönteni, hogy melyik háromszög felületi pontját kell kirajzolni. Ebből látható a scan-line algoritmusoknak az a nagy előnye, hogy az objektumok közötti geometriai összefüggéseket figyelembe véve a látható pixelek meghatározásához szükséges tesztelésekből a feleslegeseket képes kiszűrni. Így például, ha két egymást követő pixelsorhoz tartozó aktív élek és ezek sorrendje megegyezik, akkor nem történt változás az objektumok takarási viszonyaiban, így nem szükséges a háromszögek újabb mélységi vizsgálata.

A levetített poligonok minden nem vízszintes élére vonatkozóan ET létrehozása, a kisebb y koordináta szerint rendezetten.



ET

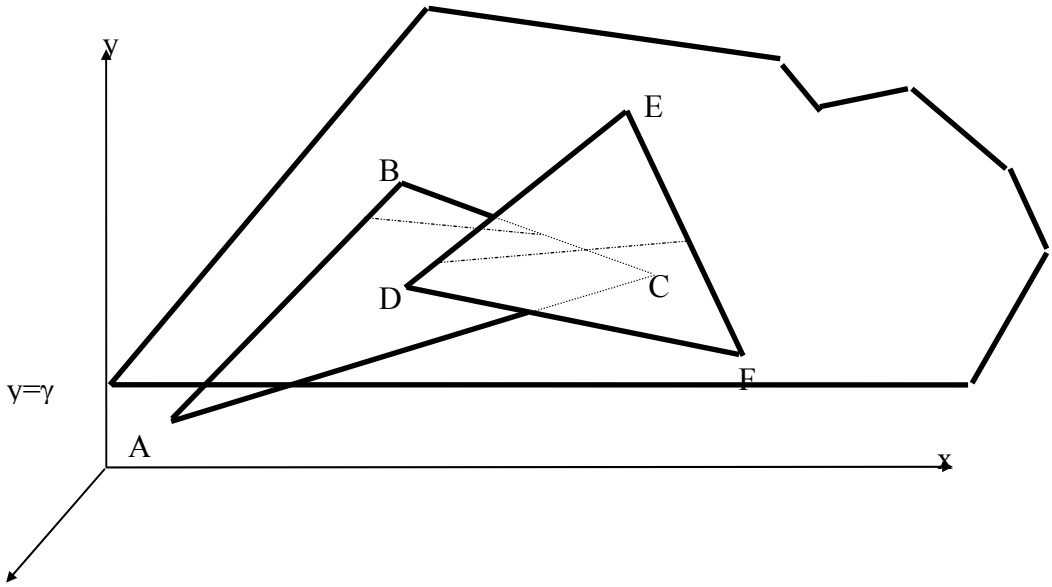
x	ymax	Δx	ID	•→

PT

ID	Sík egyenlet	Szin inf.	Flag

AET:

α	AB	AC		
β	AB	AC	FD	FE
γ,γ+1	AB	DE	CB	FE
γ	AB	CB	DE	FE



8.5 A z-buffer vagy mélység tároló algoritmus

Az algoritmus két tárolóterületet használ:

frame-buffert, mely a képernyő pixeleihez rendelt színértékeket tárolja, induló feltöltése a képernyő háttérszíne,

z-buffert, mely az egyes pixelekhez rendelt z értékeket tárolja a normalizált látótérből, kezdeti értéke a hátsó vágósík z koordinátája.

Ezt követően a jelenet minden egyes normalizált látótérbeli objektumának minden levetített lapjára a következő algoritmus kerül végrehajtásra:

A lap minden (x, y) koordinátájú belső pixeléhez tartozó vetítősugárhoz kiszámoljuk a laphoz tartozó z értéket, majd ha z értéke nagyobb mint egy korábban a z-bufferben letárolt érték (azaz a pont közelebb van a nézőponthoz), akkor ezzel felülírjuk a korábban letárolt értéket a z-bufferben, és egyúttal a neki megfelelő színértékkel, felülírjuk a frame-buffer (x, y) koordinátájú tárolóhelyét. Az alapalgoritmus szempontjából közömbös, hogy az objektumokat, illetve a raszterpontokat milyen sorrendben teszteljük. Ha a levetített lapok összes belső (x, y) képpontjára végrehajtjuk az eljárást, akkor ennek végén a frame-bufferben a kívánt képet kapjuk. A raszterpontok száma a korszerűbb monitorok esetén milliós nagyságrendű. Ennek megfelelő számú z-érték tesztelést kell végezni az algoritmus végrehajtása során és ezzel arányos az eljárás memóriaigénye is. Vegyük észre, hogy a z-buffer algoritmus a modelltér elemeinek formájától független, így háromszög, poliéder vagy görbült felületek láthatóságának megállapítására egyaránt alkalmas. Alkalmazhatóságának egyetlen feltétele, hogy az objektumok felületi pontjaiban a nézőponttól való z távolság és az árnyalási információk (szín, megvilágítás, textúrák) meghatározhatók legyenek. Az algoritmus jelentős erőforrásigénye, valamint tiszta formájában való alkalmazásánál egyes további eljárások (anti-aliasing, az átlátszóság stb.) elvi megvalósíthatatlansága (ugyanis a z-bufferben csak egy z értéket tároljunk) miatt a z-buffert általában más eljárásokkal kombinálva szokták alkalmazni.

Az algoritmus Pseudo-kódja:

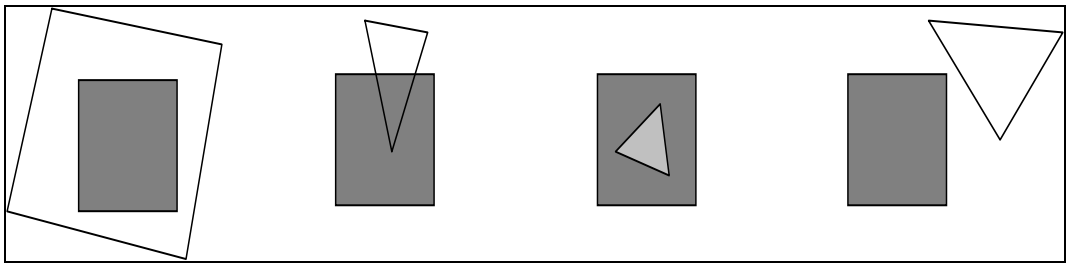
```
procdure zBuffer;
var
     pz:integer;
begin
     for y:=0 to YMAX do
          for x:=0 to XMAX do
          begin
                    WritePixel(x,y,HATTER_SZIN);
                    WriteZ(x,y,0);
               end
     for minden poligon do
          for minden pixelje a poligon képének do
               begin
pz:=a poligon (x,y) pontjának z koordinátája;
if pz>=ReadZ(x,y) then
     begin
          WriteZ(x,y,pz)
          WritePixel(x,y,szin)
     end;
               end;
end;
```

8.6 Terület-felosztásos algoritmus

A területfelosztó algoritmusok alapgondolata, hogy néhány esetben (például a képen csak egy objektumot kell ábrázolni) nagyon egyszerűen megállapítható a képernyőn megjelenítendő kép, eltért a bonyolultabb takarási vizsgálatokat a kép területének kisebb részekre való rekurzív, felosztásával vezessük vissza az egyszerűen kezelhető esetekre. A képfelosztás történhet a raszteres képernyőkoordináta-rendszerben, illetve a vektoros látótér koordináta-rendszerben.

A legismertebb algoritmus ebben a csoportban Warnock (1969) algoritmusa.

A rekurzió után maradó lehetséges helyzetek:



1. Minden poligon a vizsgált területen kívül esik. Háttérszinnel befestjük a területet.
2. Csak egy metsző vagy tartalmazott poligon van a vizsgált területen. Először a területet kitöltjük háttérszínnel, majd a poligon területen belüli részét scan-konvertáljuk.
3. Csak egy, a vizsgált területet teljesen tartalmazó poligon van. Befestjük a területet a poligonnak megfelelő színnel.
4. Több mint egy poligonnak van közös része a tartománnyal, de van olyan, a taromány teljesen tartalmazó poligon, amely a többi előtt van.

A rekurzív felosztás egybevágó téglalapokra vagy egy alkalmas belső pont választásával négy nem egybevágó téglalapra történhet. Ez utóbbi esetben a pont alkalmas választásával az algoritmus hatékonysága jelentősen növelhető.

8.7 Sugárkövetéses algoritmusok (raytracing)

A sugárkövetéses (raytracing) algoritmusok a felületek láthatóságát a nézőpontból kiinduló, képzeletbeli fénysugarak követésével határozzák meg. Tipikus képpontosság algoritmusok. Működésükhöz a nézőpontot és egy tetszőleges vetítési síkon felvett ablakot kell definiálni. Az ablak téglalap alakú területét felosztó szabályos rács a képernyő pixeleinek, felel meg. A raytracing algoritmussal a nézőpontból vetítősugarat indítunk az ablak minden egyes pixelén keresztül a jelenet objektumai felé. Az adott pixel színét a sugár által a nézőponthoz legközelebb metszett objektum színe határozza meg. Minden raytracing algoritmussal szemben támasztott legfontosabb követelmény, hogy képes legyen fénysugarak és objektumok metszéspontjait megtalálni. A feladat megoldásához a fénysugarat a nézőpont és egy pixel által definiált vektor segítségével paraméteres egyenletével formában írjuk fel. Ha a nézőpont a

$C(x_0, y_0, z_0)$, a vizsgált képpont pedig a $P(x_1, y_1, z_1)$, akkor a rájuk illeszkedő sugár bármely *(x, y, z)* pontja megadható az egyenes paraméteres egyenletével:

$$x = x_0 + t(x_1 - x_0),\ y = y_0 + t(y_1 - y_0),\ z = z_0 + t(z_1 - z_0)$$
$$\Delta x = x_1 - x_0,\ \Delta y = y_1 - y_0,\ \Delta z = z_1 - z_0$$
$$x = x_0 + t\Delta x,\ y = y_0 + \Delta y,\ z = z_0 + \Delta z$$

A t paraméter 0 és 1 közé eső értékei a nézőpont és a vizsgált pixel közé eső fénysugárpontokat definiálják, negatív értékek esetén a nézőpont mögött vagyunk, míg 1-nél nagyobb t értékekre a képsík túlsó oldalára kerülünk. A fénysugár és az objektumok metszéspontjainak meghatározásához minden egyes, a jelenetet alkotó objektumtípust matematikailag le kell írnunk.

Pl gömb esetében a metszéspont az alábbi *t*-re másodfokú egyenlet alapján számítható::

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2, \text{ behellyettesítve } x,\ y,\ \text{és } z\text{-t:}$$
$$(\Delta x + \Delta y + \Delta z)^2 t^2 + 2t\left[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)\right] +$$
$$(x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$$

Poligon esetén a metszéspont egyszerűbben számolható::

A poligon síkjának egyenlete: $Ax + By + Cz + D = 0$. Behelyettesítés után t-re adódik: $t = \dfrac{Ax_0 + By_0 + Cz_0 + D}{A\Delta x + B\Delta y + C\Delta z}$, hacsak $A\Delta x + B\Delta y + C\Delta z \neq 0$.

A z-buffer algoritmushoz hasonlóan a raytracing-nek is megvan az az előnyös tulajdonsága, hogy metszéseket csak objektumok és fénysugarak között kell elvégeznünk, objektumok egymás közti helyzetének közvetlen meghatározására nincs szükség. A raytracing algoritmus legegyszerűbb (egyben legdrágább) változata a pixelekhez tartozó sugarakkal elmetszi a jelenetben szereplő összes objektumot. Egy száz objektumból álló jelentről készítendő 1024x1024-es felbontású kép tehát több mint 100 millió metszési műveletet igényel! Nem meglepő tehát, hogy az átlagos jeleneteket ábrázoló képek előállítási idejének döntő része is a metszéspontok meghatározására fordítódhat. Ezért a raytracing algoritmusok időigényének csökkentését célzó kutatások fő iránya a metszések

hatékonyságának növelése, illetve a felesleges metszési műveletek teljes elkerülése.

Az algoritmus pszeudó-kódja:

```
for minden scan-line-ra a képsíkon do
     for minden pixelre a scan line-on do
     begin
          a centrumból a pixelhez vezető sugár
          meghatározása;
          for minden megjelenítendő objektumára a
          térrésznek do
                         if van metszéspont és közelebb
                         van, mint az előző then
                              feljegyezni a
                              metszéspontot és az
                              objektumot,
                              a pixel színét beállítani

     end;
```

8.7.1 Rekurzív sugárkövetés

A rekurzív raytracing a számítógépes grafika fotorealisztikus képábrázolásának legelterjedtebb algoritmusa, legintenzívebben fejlődő területe. Felhasználják a filmiparban, a reklámkészítésben, a videoclip-gyártásban, tervek fotorealisztikus bemutatásában, sőt a művészetben is. A raytracing algoritmussal készített képek jellegzetességei könnyen felismerhetők.

Ezek:

- objektumok egymásra vetett árnyékai,
- többszörös tükröződések,
- átlátszóság kezelése fénytöréssel.

A felületi pontoknak megfeleltetett pixel színének kiszámításához általános esetben több típusú másodlagos sugarat is nyomon kell követni. Ezek lehetnek a visszatükröződött sugarak, megtörő sugarak, és a fényforrással összekötő sugarak. Ha a fényforrással összekötő sugarak (ezekből annyit kell indítani, ahány fényforrás van definiálva a modelltérben), melyeket árnyéksugaraknak is neveznek, beleütköznek valamilyen objektumba mielőtt a fényforrást elérnék,

akkor a felületi pont árnyékban van és a megfelelő fényforrást ki kell hagyni a felületi pontot árnyaló megvilágítási egyenletből.

8.7.2 A raytracing műveletigényét csökkentő eljárások

8.7.2.1 Backward Raytracing

Mivel a fényforrásokból kiinduló fénysugarak többsége olyan, hogy nem jut el az ablakon keresztül a nézőpontba, jelentősen csökken a számításigény, ha ezekkel nem foglalkozunk. Ezért az algoritmus csak azokat a fénysugarakat veszi figyelembe, melyek az ablak celláin keresztül e jutnak a nézőpontba. Ezt úgy éri el, hogy a fény terjedésével ellentétben nem a fényforrásokból kiinduló sugarakat kezdi vizsgálni, hanem azokat a nézőpontból indított vetítősugarakat elemzi, melyek visszafelé eljutnak a fényforrásba. Ezt a megoldást "hátrafelé történő sugárkövetésnek", vagy Backward Raytracing-nek nevezik.

8.7.2.2 Bounding Volumes

A raytracing legműveletigényesebb része a vetítősugarak és az objektumok metszéspontjainak kiszámítása, célszerű ezek számát tehát csökkenteni. Ezt célozza a raytracing sorári a befoglaló testek alkalmazása. Ennek során legtöbbször gömböket használunk fel. Ez azt jelenti, hogy olyan legkisebb méretű gömböket definiálunk a modelltérben, melyek objektumainkat teljes mértékben tartalmazzák. A befoglaló gömbökkel úgy tudjuk csökkenteni a metszéspontok számításának mennyiségét, hogy csak azokat a vetítő sugarakat vesszük figyelembe a metszéspontszámításnál, melyek a befoglaló gömböket is metszik.

8.7.2.3 A sugarak transzformálása z tengellyel párhuzamos helyzetbe.

A térpontjait egy megfelelő projektív transzformácóval olyan helyzetbe hozzuk, hogy a sugarak párhuzamosak legyenek a z-tengellyel. Ha az 5.1 fejezetbeli speciális centrális vetítésből indulunk ki, akkor a megfelelő transzformációs mátrix az alábbi lesz:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{s} & 1 \end{bmatrix}$$

Az **A** mátrixú transzformáció után a pontok képei a z-koordinátáik elhagyásával előállnak. Az objektumok és a speciális vetítősugarak kölcsönös helyzetének vizsgálata nagyban leegyszerűsödik, sok teszt 2d-ben elvégezhető.

9 Színelméleti alapok

- Színérzet:A fény elektromágneses hullám spektrális tulajdonságai által keltett hatás
- Tristimulus: a beérkező fényenergiát leíró skalár érték
- Szintér: A lehetséges színérzetek alkotta három dimenziós tér

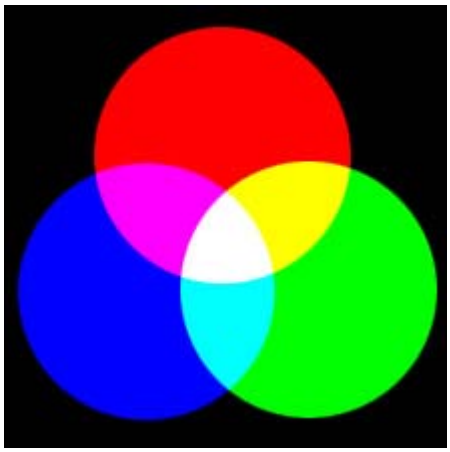
Pl.piros (red), zöld (green) és kék (blue) színérzetet okozó hullámhosszakból álló bázis:

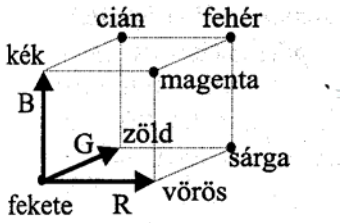
$$\lambda_{red} = 700nm, \lambda_{green} = 561nm, \lambda_{blue} = 436nm,$$

- színillesztő függvények: Egy λ hullámhosszú monokromatikus fénynyaláb keltette ekvivalens színérzetet előállító (r(λ), g(λ) és b(λ)) függvények (fiziológiai mérések eredménye)

9.1 RGB színrendszer:

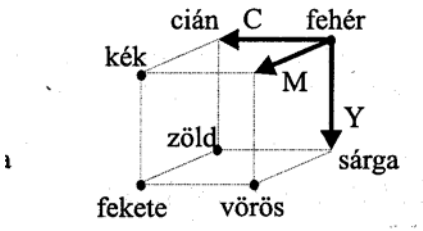
Egy képpont a piros, a kék és a zöld 256-256-256 féle árnyalatából áll össze, összesen 16 millió színárnyalattal. 24 biten tárolja az információt. Ez additív színrendszer, tehát a három alapszín egyforma keverése fehér, hiányuk fekete színt eredményez. Ezeket a színeket használja minden elektronikus kivetítőeszköz (monitor, kivetítő).





9.2 CMY színrendszer:

Az RGB rendszer alap színeinek kiegészítő színeit használja. Elsősorban a nyomtatóknál alkalmazzák ezt a színrendszert. Ez úgynevezett szubtraktív színrendszer. Egy képpont a türkiz (Cyan), a bíbor (Magenta) a sárga (Yellow) (másodlagos alapszínek) árnyalataiból áll össze. A színek hiánya fehéret eredményez.

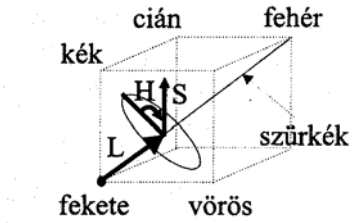


9.3 CMYK (vagy 32 Bit Color):

Az előző szírendszer kiegészül a fekete (blacK) 256*4 féle árnyalatával. 32 biten (4 byte) tárolja az információt. 4,3 milliárd árnyalata lehet egy képpontnak.

9.4 HLS szinrendszer:

Színárnyalat-fényesség-telítettség hármas határozza meg a színt.



9.5 Egyéb lehetőségek

Black and White: Egy képpontnak két állapota van, fekete és fehér. Egy képpont állapotának rögzítése 1 bitet igényel.

16 Color: 16 megadott színe lehet egy képpontnak, 4 biten tárolja az információt. Ilyen pl. a BGI grafika.

Grayscale: Egy képpont a szürke 256 árnyalatával rendelkezhet, 8 biten tárolja az információt.

256 Color: 256 megadott színe lehet egy képpontnak, 8 biten tárolja az információt. (Régebben weblapok képeihez ajánlották ezt a színtartomány)

10 Egyszerű megvilágítás és árnyalási technikák

Fotorealisztikus képek előállítása nagyon nagy számításigényű probléma. Minden pixelben meg kell határozni az objektum színét, ami bonyolult szinmodell esetén sok időt jelent. A számítások egyszerűsítésére megalkotott módszerek közül ismertetünk néhányat.

10.1 Szórt háttérvilágítás (ambient light)

Ekkor az objektumok egyenletesen, minden irányból kapnak fényt. Hatása a nappali fényviszonyoknak felel meg, ha nincs napsütés. Ebben a modellben nincs fényforrás, az objektumok "saját" fényüket bocsátják ki. Ez megfelel annak, hogy a jelenetet egy irányfüggetlen, szórt fény világítja meg. Plasztikus képek előállítására nem alkalmas.

10.2 Diffúz fényvisszaverõdés (diffuse light)

A diffúz fényvisszaverődés a matt felületek jellemzője. Ekkor a megvilágított felület minden irányban ugyanannyi fényt ver vissza.

10.3 Fényvisszaverődés (specular light)

A sima felületekre általában az a jellemző, hogy rajtuk fényes foltokat is látunk, melyek helye nézőpontunkkal együtt változik. Ezek a felületek bizonyos irányokban visszatükrözik a fényforrásokat. A fényforrás távolságától és irányától is függő komponens, a spekuláris visszaverődésből származó színt adja meg.

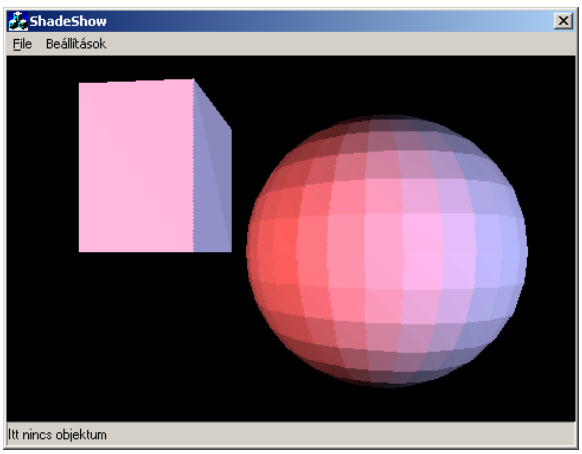
10.4 Konstans árnyalás (Flat shading)

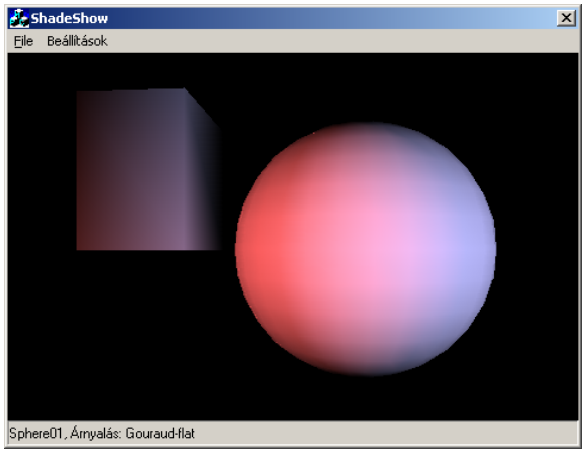
Síklapok árnyalására a leggyorsabb módszer. Feltételezi, hogy a fényforrás végtelentávoli, így a sík lap minden pontjába azonos szögben érkezik. A legegyszerűbb esetben tehát a beeső fénysugár és a felületi normális szögének függvényében határozzuk meg a szinintenzitást, minél közelebbi van a szög a derékszöghöz, annál intenzívebb árnyalatot rendelünk a ponthoz. A szín tehát csak laponként számolandó.

10.5 Gouraud féle árnyalás

Elsősorban poligonok árnyalására kidolgozott eljárás. A poligon minden csúcsában meghatározunk egy vektort, melynek koordinátái az adott csúcsban találkozó lapokhoz tartozó normálvektorok megfelelő koordinátáinak számtani közepe. A színmodellnek megfelelően kiszámítjuk a színeket a csúcsokban. Először az poligon éleinek színét interpoláljuk, majd a poligon belső pixeleinek

szinét határozzuk meg lineáris interpolációval. Érdemes összekötni a módszert a polligon fillezésre vonatkozó algoritmussal (2.4 fejezet).





Flat shading Gouraud

10.6 Phong árnyalás

A Goruaud árnyalástól abban különbözik, hogy magát a normálvektorokat határozza meg lineáris interpolációval, a színek kiszámítása a színmodellnek megfelelően pixelenként történik.