Лекции по компютърна графика

Спас Ташев

6 декември 2010 г.

Глава 1

Минимум сведения за езика PostScript

PostScript е език за програмиране с който може да се оформи дизайна на страницата за печат или за показване на екран.

Минимални необходими сведения:

PostScript - овските файлове са обикновени текстови файлове, обикновено с наставка .ps, която не е задължителна, и те могат да се генерират на ръка или програмно.

Коментари могат да се слагат навсякъде със символа за процент и след него до края на реда всичко е коментар.

Много често първиат ред на файла започва със символа %! (процент последван от удивителен знак), което не е задължително.

Командата **showpage** форсира принтера да отпечати приготвената страница.

Софтуарът с който може да си види резултатът е **Ghostview** и обикновено може да се намери в директория

C : \ProgramFiles\Ghostgum\gsview\gsview32.exe.

PATH

path е множество от линии и криви. След като **path** е описана, тя може да бъде изчертана чрез командата **stroke** или запълнена ако е област чрез командата **fill**.

PostScript използва **stack** (LIFO (Last In First Out) - стек) за пазене на програмата и данните. **PostScript** - овския интерпретатор запазва **PostScript** - овската програма в стека и я изпълнява. Например операторът за умножение **mul**, който изисква 2 операнда (аргумента) изглежда така

```
11  10  mul
```

Интерпретаторът ще постави 11 в стека, после 10 в стека, тогава операторът **mul** ще извади първо 10 от стека, после 11 , ще ги умножи и ще постави в стека резултата (в случая 110).

Координатна система

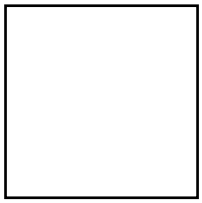
По подразбиране координатната система е с център в долния ляв ъгъл на екрана. X-координатата е насочена надясно а Y-координатата нагоре. Единиците за измерване са points (pt) като 1pt $= \frac{1}{72}$ от инча (приблизително). Ще използваме често факта 1 inch = 72pt. Координатната система може да се промени като се използват операторите

scale , rotate, translate.

Основни команди за чертаене:

Пример 1. Нека да начертаем квадрат със страна 1 инч и долния му ляв ъгъл да е в точката с координати $x = 400,\ y = 500$. На езика на **PostScript** това може да се реализира с операторите:

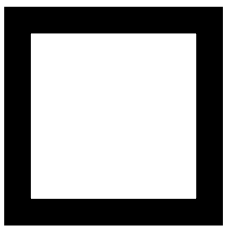
```
newpath
400 500 moveto
472 500 lineto
472 572 lineto
400 572 lineto
400 500 lineto
stroke
showpage
```



Операторът **newpath** занулява старата **path**, ако е имало такава, и декларира нова **path** . Резултатът от втоия оператор - **400 500 moveto** е че курсосър (въображаемия) отива в точка с координати (400, 500). Точката (400,500) става текуща. Третият оператор - **472 500 lineto** добавя сегмент към текущата пътека **path** от текущата точка до точката (472,500). И така нататък. Операторът **stroke** начертана така построената пътека. Пътеката става видима. Последния оператор (който не е задължителен) праща на принтера текущата страница (за нас така построената пътека). Резултатът ще изглежда квадрат показан вдясно.

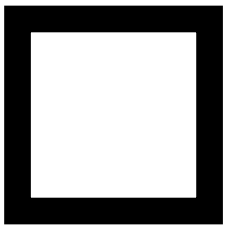
Пример 2. Да направим малка промяна в горния пример. Изтриването на оператора **showpage** няма да промени дисплея. Добавили сме нов операторе **10 setlinewidth** , който указва дебелината на линията която се чертае. В случая сме указали $10pt$.

```
newpath
400 500 moveto
472 500 lineto
472 572 lineto
400 572 lineto
400 500 lineto
10 setlinewidth
stroke
```



Пример 3. „Изядения“ долен ляв ъгъл на горния пример може да се отстрани например като заменим 4-я оператор **lineto** с оператора **closepath**, който свързва последната текуща точка с първата в текущата пътека.

newpath
400 500 moveto
472 500 lineto
472 572 lineto
400 572 lineto
closepath
10 setlinewidth
stroke

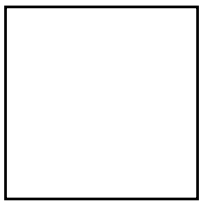


Пример 4. Съществуват и следната модификаци на операторите **moveto** и **lineto**: **rmoveto** и **rlineto**. Например ако въведем

a b rmoveto

то курсорът ще се премести в точка с координати $(\mathbf{x}+\mathbf{a}, \mathbf{y}+\mathbf{b})$, където $(\mathbf{x}, \mathbf{y})$ са координатите на текущата точка. Горния квадрат може да се изчертае и по следния начин:

newpath
400 500 moveto
72 0 rlineto
0 72 rlineto
-72 0 rlineto
closepath
stroke



Квадратът може да бъде запълнен с оттенаци на сивото като вместо

stroke

използваме операторът

fill

Преди него може да се постави оттенък на сивото:

0.6 setgray fill

или може да се зададе цвят чрез

r g b setrgbcolor fill

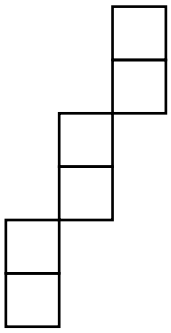
където **(r,g,b)** са оттенъци на червеното, зеленото и синьото. Променливите са в интервала [0,1].

Пример за извеждане на текст.
Ще запълним квадрата със сиво и ше изведем текст - „example 5“

```
newpath
400 500 moveto
72 0 rlineto
0 72 rlineto
-72 0 rlineto
closepath
0.5 setgray
fill
/Times-Roman findfont
12 scalefont
setfont
newpath
400 490 moveto
(example 5) show
```

Пример 6.
Искаме да начертаем много квадрати (които ще имитират пиксели) със страна h и които да са отместени един от друг на разстояние кратно на h. Първо ще дефинираме константа h а след това ще напишем процедура за чертаене на квадрат с име **setpixel** и накрая ще начертаем няколко квадрата, които да имитират една права линия.

```
/h 20 def
/setpixel {
      newpath
      moveto
      h -2 div h -2 div rmoveto
      h 0 rlineto
      0 h rlineto
      h -1 mul 0 rlineto
      closepath
      stroke
} def
400 500 setpixel
400 520 setpixel
420 540 setpixel
420 560 setpixel
440 580 setpixel
440 600 setpixel
/Times-Roman findfont
12 scalefont setfont
300 20 moveto
(example 1) show
```



example 1

Първият оператор дефинира константа h със стойност 20pt. От втори ред до 11-и дефинирана функция **setpixel**. На 12 ред тя е извикана със параметри 400 и 500, като тези параметри се отнасят до 4-и ред за оператора **moveto**, който във функцията е без параметри. h -2 div на 5-и ред означават, че стойността на h, която е 20 ще се раздели на -2 и резултата -10 ще е първия параметър на rmoveto. И другия параметър е -10. Тогава 5-и ред всъщност е еквивалентен на -10 -10 rmoveto, което ще премести курсора от текущата му позиция, (400,500) в точката (390,490). По нататък е ясно какво става. В този пример квадрата дефиниран със setpixel има център пресечната точка на диагоналите му. Може да се избере центърът да е долния ляв ъгъл на квадрата или някой друг ъгъл. В примерите координатите на точките бяха цели числа, което не е задължително. Ако са дробни интерпретаторът ги закръглява.

Повече подробности свързани със PostScrit-а могат да се намерят в интернет.

Глава 2

Растеризиране на примитиви

Растеризиране на примитиви

Примитиви са най-прости криви които се използвата за чертаене на фигури на екрана. например отсечка, окръжност, дъга от окръжност, елипса, символи.

Под растеризиране на отсечка например, се разбира следното: Предполагаме че е зададена отсечка (сегмент) с крайни точки $A = (a_1, a_2)$ и $B = (b_1, b_2)$ дефинирани в правоъгълна координатна система. Целочислените точки, т.е. точките с координати цели числа, ще ги наричаме пиксели. Избирането на онези пиксели, които са най-близо до точките от отсечката се нарича растериация на отсечката, а избраните пиксели ще ги наричаме растер на отсечката.

Правоъгълника в който варират пикселите се нарича "растерен дисплей"или само "дисплей"или "екран". За нас това ще част от екрана на монитора. Ние ще използваме софтуар GhostView за визуализиране на растера. Екрана на този софтуар на моя компютър е правоъгълник с размери 494 x 852 пиксела. Кординатната система е с център долния ляв ъгъл на екрана, X расте отляво на дясно в граници [0,494] а Y расте отдолу нагоре в интервала [0,852].

Алгоритмите за чертаене на отсечка трябва да изпълнявата някои изисквания като:

1. Растеризираната отсечка трябва да прилича на отсечка, крайните точки трябва да са избани акуратно.

2. Осфетеността на сегмента трябва да не зависи от мястото и ориентацията.

3. Сегмента трябва да се изчертава бързо.

2.1 Алгоритъм на Брезенхам за растеризиране на отсечка

Този алгоритъм търси да избере най-оптималните пиксели, които да представят отсечката. В зависимост от наклона на отсечката едната от координатите X или Y се променя с h (h ще е за нас големината на страната на пиксела, или разстоянието между пикселите, като в реалния случай $h = 1$) , а другата координата или се променя с h или не, в зависимост от разстоянието между отсечката и най близкия пиксел от вертикалната права $X = x + h$. Това разстояние ще го наричаме грешка. Този алгоритъм така е конструиран, че се проверява само знака на грешката. На фигурата по-долу е начертана част от грида на пикселите и 4 пиксела (черните точки) които приближават наклонената отсечка с наклон 3/8. При наклон на отсечката m такъв, че $0 \leq m \leq 0.5$, на всяка стъпка x ще се увеличава с h, а y ше се увеличи с h или не, в зависимост от знака на грешката e. Ако отсечката започва от пиксела означен с 1 и с координати (x, y), то този пиксел трябва да бъде избран. Грешката е 0, но тъй като ще се интерисуваме само от знака на грешката, то задаваме първоначална стойност на грешката $e = -h/2$. След това x се увеличава с h. Увеличаваме грешката с mh т.е. $e = e + mh$. За

нашия пример грешката става $-1/8h$. Пак е отрицателна и Y координата не се променя, т.е. от пикселите от вертикалната права 2 трябва да изберем черния пиксел. На следващата стъпка $X-$координата нараства пак с h. Грешката нараства с mh и става $2/8h$. Сега вече грешката е положителна и Y координата нараства с h, т.е. $y = y + h$. На вертикала 3 това е черния пиксел. Преди да продължим нататък трябва да инициализираме грешката наново. Това го правим като извадим h от нея, т.е. $e = e - h$.

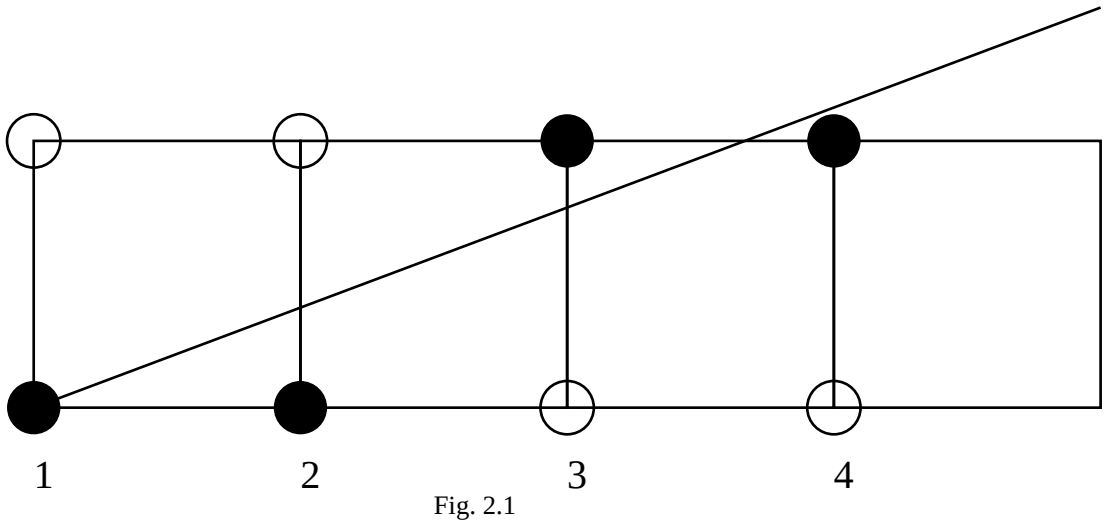


Fig. 2.1

Тъй като проверяваме само знака на грешката, за да смятаме само с цели числа, от инициализирането и от промяната на грешката, равенствата

$$\begin{aligned} e &= -\frac{h}{2} \\ e &= e + \frac{\Delta_y}{\Delta_x}h, \quad m = \frac{\Delta_y}{\Delta_x} \\ e &= e - h \end{aligned}$$

можем да ги умножим с $2\Delta_x$ и да положим $\bar{e} = 2\Delta_x$. Ще имаме

$$\begin{aligned} 2e\Delta_x &= -\Delta_x h \\ 2e\Delta_x &= 2e\Delta_x + 2\Delta_y h \\ 2e\Delta_x &= 2e\Delta_x - 2\Delta_x h \end{aligned}$$

т.е.

$$\begin{aligned} \bar{e} &= -2\Delta_x h \\ \bar{e} &= \bar{e} + 2\Delta_y h \\ \bar{e} &= \bar{e} - 2\Delta_x h \end{aligned}$$

Алгоритъм на Брезенхам за I октант

Нека краищата на сегмента да са $(x1, y1)$ и $(x2, y2)$ и те да не са равни едновременно.

```
Bresenham.partial(x1,y1,x2,y2)
/* Инициализиране на променливите */
x=x1
y=y1
dx=x2-x1
dy=y2-y1
e=2*dy*h-dx*h
/* Главния цикъл */
while (x<x2)
     setpixel(x,y)
     while (e>0)
          y=y+h
          e=e-2*dx*h
     end while
     x=x+h
     e=e+2*dy*h
end while
```

Ако наклонът на сегмента по абсолютна стойност е по-голям от $1/2$ то триабва Y-координатата да се променя на всяака стъпка, а за X-координатата трябва да се приложи правилото на Брезенхам. От следващата диаграма се вижда лесно и как ще изглежда алгоритъма в общия случай.

Алгоритъм на Брезенхам в общия случай

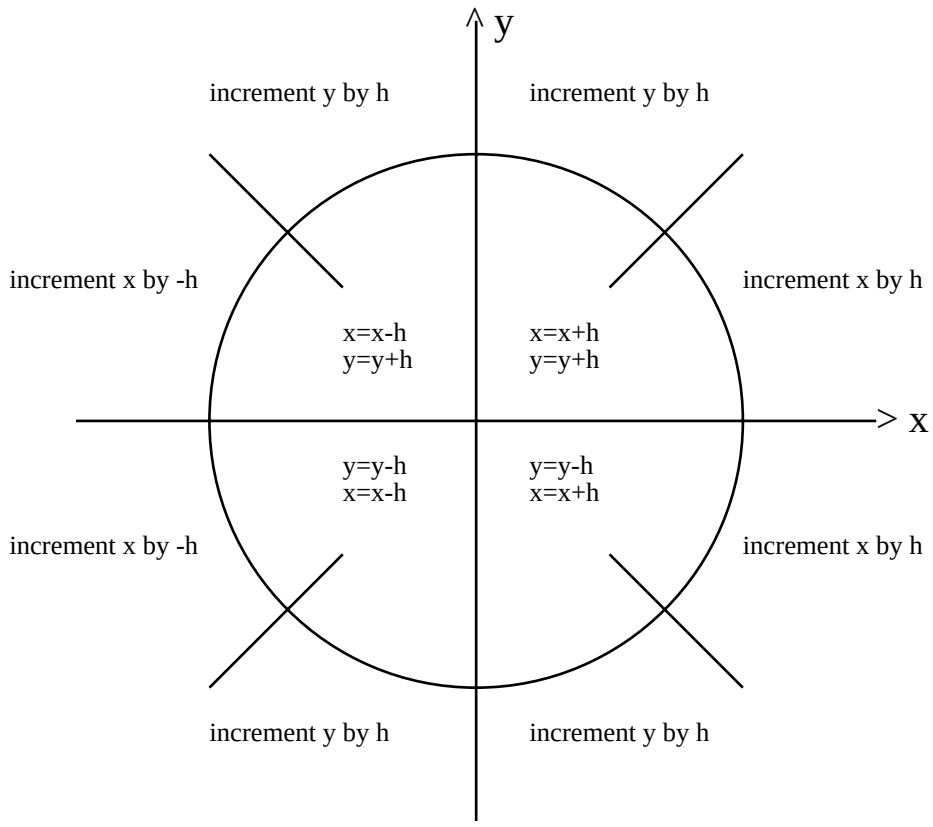


Fig. 2.2

```
Bresenham.general(x1,y1,x2,y2)
/* Инициализиране на променливите */
x=x1
y=y1
dx=abs(x2-x1)
dy=abs(y2-y1)
s1=sign(x2-x1)
s2=sign(y2-y1)
/* размяна на dx и dy в зависимост от наклона */
if dy > dx then
    temp=dx
    dx=dy
    dy=temp
    interchange=1
else
    interchange=0
end if
/* Инициализиране на грешката */
e=2*dy*h-dx*h
/* Главния цикъл */
while (x<x2)
    setpixel(x,y)
    while (e>0)
        if (interchange = 1) then
            x=x+s1*h
        else
            y=y+s2*h
        end if
        e=e-2*dx*h
    end while
    if (interchange = 1) then
        y=y+s2*h
    else
        x=x+s1*h
    end if
    e=e+2*dy*h
end while
```

2.2 Алгоритъм на средната точка за растеризиране на отсечка

Нека отсечката да е зададена с наклон в интервала $[0, 1/2]$ и с краища $A = (x_1, y_1)$ и $B = (x_2, y_2)$. Да запишем уравнението на правата през точките A и B във вида

$$f(x, y) \equiv ax + by + c = 0, \text{ където } a = -dy, b = dx, c = x_1y_2 - x_2y_1, \ dx = x_2 - x_1, \ dy = y_2 - y_1.$$

Нека да си припомним, че векторът с така определените координати $N = (a, b) = (-dy, dx)$ е перпендикулярен на правата и посоката му е такава, че стойността на f в точка от по-

луравнината в която той сочи е положителна. Този факт се използва в метода на средната точка.

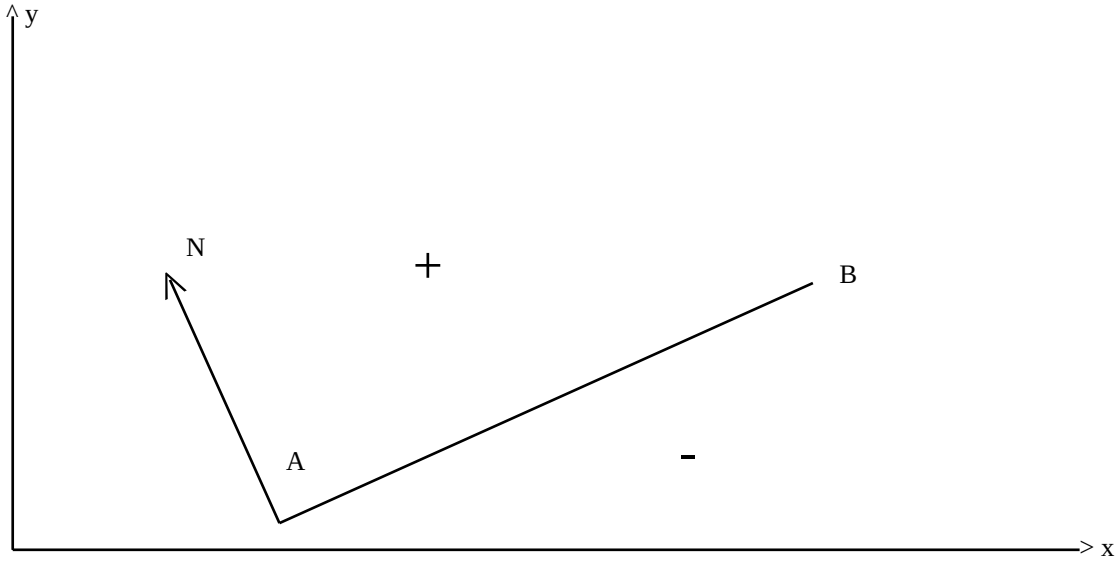


Fig. 2.3

Нека текущата точка да е $P = (x, y)$ и на следващата стъпка $x = x + h$, да трябва да избираме измежду хоризонталната точка $H = (x+h, y)$ и диагоналната точка $D = (x+h, y+h)$. За целта ще гледаме дали грешката $E = f(M)$ е положителна или отрицателна. Ако $E < 0$, то нашата права е пресякла вертикалната права през точката $x + h$ над средната точка на точките H и D и следователно пиксела D е по-близо до правата и него избираме. В другия случай $E \leq 0$ избираме H. Трябва да намерим новата грешка:

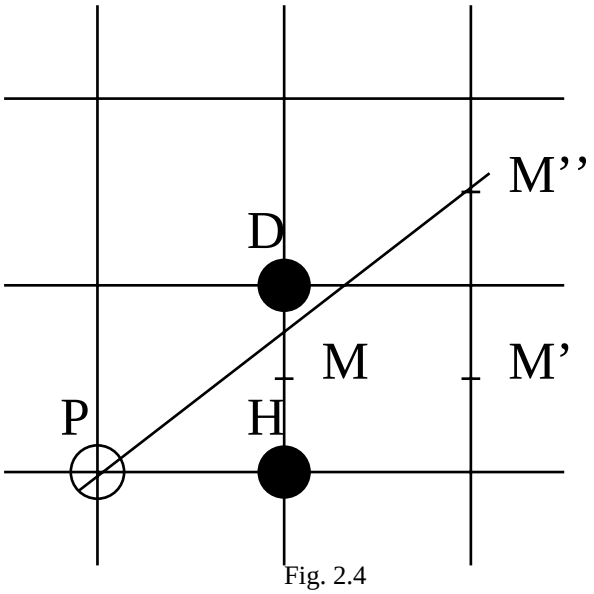


Fig. 2.4

Ако е избрана точка H то новата грешка ще бъде

$$E_H = f(M') = f(M) + f(M') - f(M) = E_P + [a(x+2h) + b(y+\frac{h}{2}) + c] -$$
$$[a(x+h) + b(y+\frac{h}{2}) + c] = E_P + ah$$

т.е.

$$E = E - dx.h \tag{2.1}$$

Ако е избрана точка D то:

$$E_D = f(M'') = f(M) + f(M'') - f(M) = E + [a(x+2h) + b(y+h+\frac{h}{2}) + c] -$$
$$[a(x+h) + b(y+\frac{h}{2}) + c] = E_P + ah + bh$$

т.е.

$$E = E - dx.h + dy.h \tag{2.2}$$

В началото инициализираме

$$x = x_1,\ y = y_1,\ E = -dy(x_1+h) + dx(y_1+\frac{h}{2}).$$

За да работим само с цели числа можем да умножим всички пресмятания с 2, т.е. да работим с нова грешка $\overline{(E)} = 2E$.

2.3 Алгоритъм на Брезенхам за растеризиране на окръжност

Както и за отсечка така и за окръжност, алгоритмите на Брезенхам са едни от най-ефективните. Ще предполагаме че окръжността е центрирана в началото на координатната система и че радиусът и R е цяло число (в нашия случай R е кратен на h.) Да забележим, че е достатъчно само един октант да бъде растеризиран за да получим пълния растер. Наистина да растеризираме само във втори октант. Да стартираме от пиксела с координати $(0, R)$ и да се движим по посока на часовата стрелка. Тогава y е намаляваща функция на x. Ако стигнем до $y = x$, то поради симетрията на всеки пиксел от втори октант с координати (x, y) ще отговарят още 7 пиксела от останалите октанти а именно пикселите с координати
$(y, x), (-y, x), (x, -y), (-x, -y), (-y, -x), (-x, y)$. Нека уравнението на окръжността да е

$$f(x, y) \equiv x^2 + y^2 - R^2 = 0.$$

Да забележим, че за точки вътре в окръжността, тази функция е отрицателна, а за външни точки - положителна.

Да предположим, че на текущата стъпка се намираме в точка $P = (x, y)$, от втори октант и че ще се движим по посока на часовата стрелка, т.е. на следващата стъпка трябва да изберем един от следващите 3 пиксела $H = (x+h, y),\ D = (x+h, y-h),\ V = (x, y-h)$.

Да означим грешката за текущата точка $P = (x, y)$ с Δ_P, където

$$\Delta_P = f(D) = (x+h)^2 + (y-h)^2 - R^2.$$

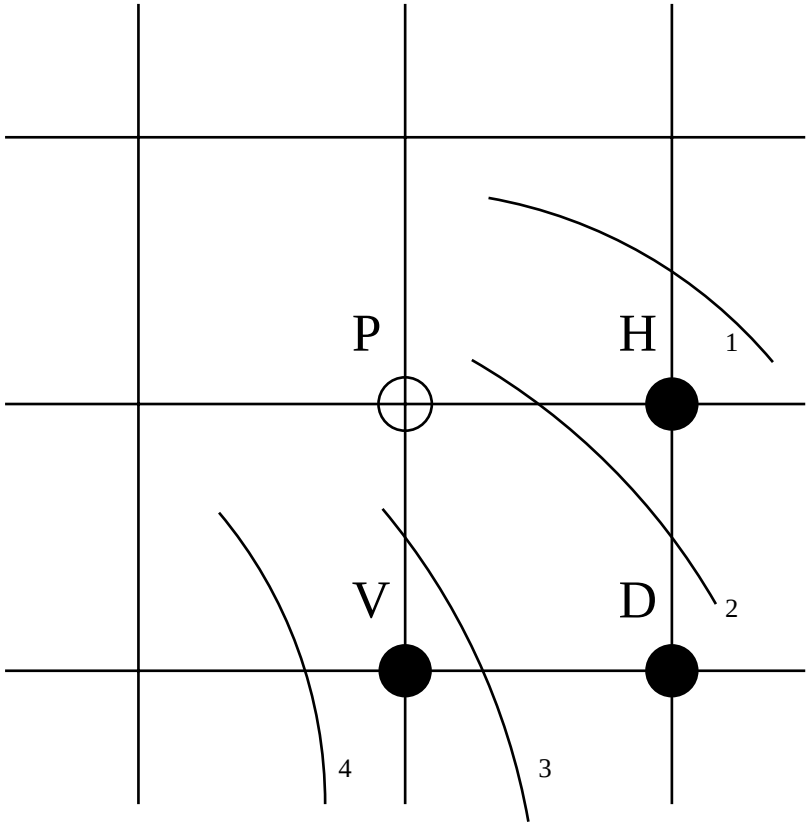


Fig. 2.5

Да забележим, че вземаме стойността на f в диагоналната точка на P.

Разглеждаме знака на грешката.

1. Ако $\Delta_P < 0$, то диагоналната точка D е вътрешна за окръжността и трябва да избираме измежду точките D и H коя е по-близо да окръжността, и нея ще причислим към растера. На картинката това са случаите 1 и 2. За да решим коя точка да изберем да разгледаме величината

$$\delta_P = |f(H)| - |f(D)|.$$

1.1. Ако $\delta_P < 0$, ($|f(D)| > |f(H)|$) то точката H е по-близо до окръжността и избираме нея. В случай, че 1.2. $\delta_P > 0$, то т. D е по-близо до окръжността и избираме D, а при $\delta_P = 0$, избираме без значение коя точка, например D.

И така в случая $\Delta_P < 0$, имаме

1.$\Delta_P < 0$
  1.1. $\delta_P < 0$, избираме т. H
  1.2. $\delta_P \geq 0$, избираме т. D

Да разгледаме по-подробно отделните случаи. Нека да имаме 1.1. Ако окръжността е пресякла грида на пикселите както случай 2 от картинката, то т. H е външна а т. D е вътрешна за окраъжността. В този случай, като се освободим от абсолютната стойност, имаме

$$\delta_P = f(H) + f(D) = 2f(D) + f(H) - f(D) = 2\Delta_P + (x+h)^2 + y^2 - (x+h)^2 - (y-h)^2$$

т.е.

$$\delta_P = 2\Delta_P + 2yh - h^2. \tag{2.3}$$

Ако имаме случай 1 от картинката, то двете точки са вътрешни, но по-близо до окръжността е т. H и нея избираме за растера. В този случай $\delta_P < 0$, тъй като

$$\delta_P = -f(H) + f(D) = h(h - 2y)$$

и $y \geq h$, т.е. т. H е избрана по същия критерий 1.1. С други думи, в случай 1.1. и за двата случая 1 и 2 от картинката при $\delta_P \leq 0$, избираме H.

Нека сега $\Delta_P > 0$. Тогава точка D е външна и трябва да избираме измежду точки D и V. Да разгледаме друга величина

$$\epsilon_P = |f(D)| - |f(V)|.$$

Разсъждавайки както по-горе, стигаме до извода

2.$\Delta_P > 0$
2.1. $\epsilon_P \leq 0$, избираме т. D
2.2. $\epsilon_P > 0$, избираме т. V

В случай 2.2 ще имаме

$$\epsilon_P = f(D) + f(V) = 2f(D) + f(V) - f(D) = 2\Delta_P - 2xh - h^2,$$

т.е.

$$\epsilon_P = 2\Delta_P - 2xh - h^2. \tag{2.4}$$

Казаното може още да се упрости: От случай 1.1. и равенство (**??**) следва, че при

$$\Delta_p < -yh + \frac{1}{2}h^2$$

трябва да изберем точка H. Също така от случай 2.1 и равенство (**??**) следва, че при

$$\Delta_p > xh + \frac{1}{2}h^2$$

трябва да изберем точка V, а в останалите случаи

$$-yh + \frac{1}{2}h^2 \leq \Delta_p \leq xh + \frac{1}{2}h^2$$

трябва да изберем точка D.

Независимо коя точка сме избрали, трябва да пресметнем грешката в тази точка, т.е. трябва да пресметнем $\Delta_H, \Delta_D, \Delta_V$. Имаме

$$\Delta_H = (x + 2h)^2 + (y - h)^2 - R^2 = (x + h)^2 + (y - h)^2 - R^2 + (x + 2h)^2 - (x + h)^2 = \Delta_P + 2(x + h)h + h^2 = \Delta_P + 2xh + 3h^2.$$

$$\Delta_D = (x + 2h)^2 + (y - 2h)^2 - R^2 = (x + h)^2 + (y - h)^2 - R^2 + (x + 2h)^2 - (x + h)^2 + (y - 2h)^2 - (y - h)^2 = \Delta_P + 2(x + h)h + h^2 - 2(y - h)h + h^2 = \Delta_P + 2xh - 2yh + 6h^2.$$

$$\Delta_V = (x + h)^2 + (y - 2h)^2 - R^2 = (x + h)^2 + (y - h)^2 - R^2 + (y - 2h)^2 - (y - h)^2 = \Delta_P - 2(y - h)h + h^2 = \Delta_P - 2yh + 3h^2.$$

Тъй като ще стартираме с точка $(0, R)$, то зараждаме в началото

$$\Delta = (0+h)^2 + (R-h)^2 - R^2 = 2h(h-R).$$

От тук и по-горните формули се вижда, че вместо с Δ можем да работим с $\bar{\Delta} = \frac{\Delta}{h}$. Ето и примерна програма за този алгоритъм и за първи квадрант.

```
Brezenham.central.circle(R)
/* инициализиране на променливите */
x=0
y=R;
Δ = 2h(h-R);
while (y > 0) /* или (y >= x) - за втори октант */
   setpixel(x,y)
   if Δ < 0 then
     δ = 2Δ + 2yh - h²
     if δ < 0 then call mh(x,y,Δ) else call md(x,y,Δ)
   else if Δ ≥ 0 then
     ϵ = 2Δ- 2xh -h²
     if ϵ > 0 then call mv(x,y,Δ) else call md(x,y,Δ)
end while (y > 0)
subroutine mh(x,y,Δ) begin
   x=x+h
   Δ = Δ + 2xh + 3h²
end /* mh */
subroutine mv(x,y,Δ) begin
   y=y-h
   Δ = Δ − 2yh + 3h²
end /* mv */
subroutine md(x,y,Δ) begin
   x=x+h; y=y-h
   Δ = Δ + 2xh − 2yh + 6h²
end /* md */
```

2.4 Алгоритъм на средната точка за растеризиране на окръжност

Нека уравнението на окръжността да е както преди:

$$f(x, y) \equiv x^2 + y^2 - R^2 = 0.$$

Нека да разгледаме за разнообразие растеризирането само в I октант. Ще стартираме от точката с координати $(R, 0)$ и ще се движим по посока обратна на часовата стрелка докато $y < x$. Ако текущата точка е $P = (x, y)$, то кандидати за точки от растера в I квадрант са вертикалната точка $V = (x, y+h)$ и диагоналната точка $D = (x-h, y+h)$, отбелязани с черен кръг на фигурата.

Този алгоритъм работи когато трябва да избираме измежду две точки както е в случая. Ще тестваме функцията f в средната точка на точките V и D, а тя е $M = (x - \frac{h}{2}, y + h)$. Ако

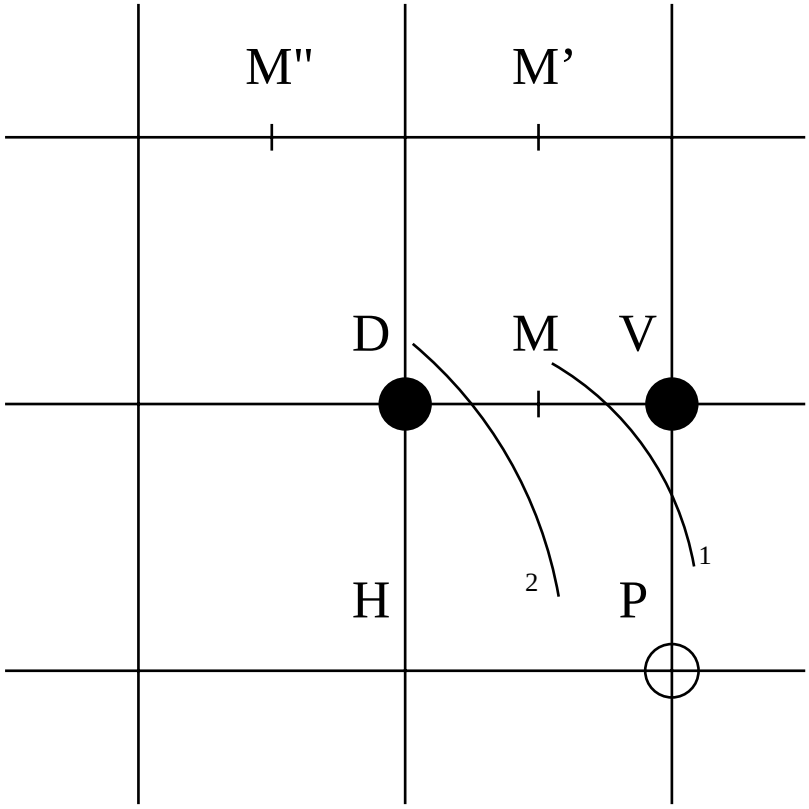


Fig. 2.6

$f(M) < 0$, т.е. точката M е вътрешна за окражността, (случай 1 на фигурата) то избираме точка V, иначе избираме точка D. Ако $f(M) = 0$, избираме например точка D. Нека грешката е

$$E_P = f(M)$$

и нека тя е отрицателна т.е. избрали сме точка V. Тогава

$$E_V = f(M) + f(M') - f(M) = f(M) + [(x - \frac{h}{2})^2 + (y + 2h)^2 - R^2] - [(x - \frac{h}{2})^2 + (y + h)^2 - R^2] =$$
$$E_P + (y + h + h)^2 - (y + h)^2 = E_P + 2(y + h)h + h^2 = E_P + 2yh + 3h^2,$$

т.е.

$$E_V = E_P + d_P^V, \quad \text{където } d_P^V = 2yh + 3h^2. \tag{2.5}$$

Ако сме избрали точката D, т.е. $E_P > 0$ то новата грешка ще е

$$E_D = f(M) + f(M'') - f(M) = f(M) + [(x - \frac{3h}{2})^2 + (y + 2h)^2 - R^2] - [(x - \frac{h}{2})^2 + (y + h)^2 - R^2] =$$
$$E_P + (x - \frac{h}{2} - h)^2 - (x - \frac{h}{2})^2 + (y + h + h)^2 - (y + h)^2 = E_P - 2h(x - \frac{h}{2}) + 2(y + h)h + 2h^2 =$$
$$E_P - 2xh + 2yh + 5h^2$$

т.е.

$$E_D = E_P + d_P^D, \quad \text{където } d_P^D = -2xh + 2yh + 5h^2. \tag{2.6}$$

В началото имаме $P = (R - \frac{h}{2}, 0)$ и трябва да заредим

$$E_P = f(R - \frac{h}{2}, 0) = -Rh\frac{h^2}{4}.$$

Вижда се, че вместо E_P можем да работим с E_P/h. Тогава ще имаме формулите

$$\bar{E}_P = -R + \frac{h}{4}. \tag{2.7}$$

$$\bar{E}_V = \bar{E}_P + \bar{d}_P^V, \quad \text{където } \bar{d}_P^V = 2y + 3h. \tag{2.8}$$

$$\bar{E}_D = \bar{E}_P + \bar{d}_P^D, \quad \text{където } \bar{d}_P^D = -2x + 2y + 5h. \tag{2.9}$$

2.5 Алгоритъм с крайни разлики от II ред за растеризиране на окръжност

Този алгоритъм е малка модификация на алгоритъма на средната точка. Трябва да пресметнем поправките на d_P^V и d_P^D когато от точка P отиваме в точка V или точка D от последната картинка.

1. От точка P отиваме в точка V.

Нека $V = (x', y') = (x, y + h)$. От (**??**) и (**??**) имаме

$$d_V^V = 2y' + 3h = 2(y + h) + 3h = 2y + 3h + 2h = d_P^V + 2h \tag{2.10}$$

и

$$d_V^D = -2x' + 2y' + 5h = -2x + 2(y + h) + 5h = -2x + 2y + 5h + 2h = d_P^D + 2h. \tag{2.11}$$

2. От точка P отиваме в точка D.

Нека $D = (x'', y'') = (x - h, y + h)$. От (**??**) и (**??**) имаме

$$d_D^V = 2y'' + 3h = 2(y + h) + 3h = 2y + 3h + 2h = d_P^V + 2h \tag{2.12}$$

и

$$d_D^D = -2x'' + 2y'' + 5h = -2(x - h) + 2(y + h) + 5h = -2x + 2y + 5h + 4h = d_P^D + 4h. \tag{2.13}$$

Вижда се, че поправките са или $2h$ или $4h$, които са константи (не зависят от координатите на точките) и това са точно вторите крайни разлики на функцията $f(x, y)$ със стъпка h. За алгоритъма трябва да въведем 2 нови променливе например d^V и d^D и трябва да ги инициализираме по формулите (**??**) и (**??**) :

$$(x, y) = (R, 0), \quad d^V = 3h, \quad d^D = -2R + 5h.$$

2.6 Растеризиране на дъга от окръжност

Ще предполагаме, че окръжността е центрирана с даден радиус R и дъгата започва от ъгъл α зададен в градуси и завършва в ъгъл β - пак в градуси. Дъгата е описана по посока обратна на часовата стрелка. Да не забравяме, че пикселите са квадрати със страна h. Уравнението на окръжността е

$$f(x, y) \equiv x^2 + y^2 - R^2 = 0.$$

Първо да намерим пиксела от растера на окръжността от който да тръгнем и пиксела в който да спрем.

Да разгледаме една точка $P = (x, y)$ в I квадрант от множеството на пикселите (x и y са кратни на h). Ако използваме тази точка за апроксимираща точка от растера, то грешката се задава с

$$E_P = f(P).$$

Нека от точка $P = (x, y)$ да се преместим в съседна точка $P' = (x + \epsilon h, y)$, където $\epsilon = \pm 1$. Грешката в P' е

$$E_{P'} = f(P') = f(P) + f(P') - f(P) = E_P + [(x + \epsilon h)^2 + y^2 - R^2] - [x^2 + y^2 - R^2] = E_P + 2\epsilon hx + h^2$$

Аналогично при преместмане по y т.е. от точка P в точка $P'' = (x, y + \delta h)$ за грешката имаме

$$E_{P''} = E_P + 2\delta hy + h^2,\ \epsilon = \pm 1.$$

Ще правим стъпка по по тази координатна ос по която се получава по-малка грешка. Ако например се намираме в I квадрант и вътре в окръжността, то тогава $\epsilon = 1$ и $\delta = 1$. Неравенството

$$|F(P')| < |f(P'')|$$

е еквивалентно на

$$-[(x + h)^2 + y^2 - R^2] < -[x^2 + (y + h)^2 - R^2]$$

това е еквивалентно на

$$x > y.$$

От тук имаме заключението, че ако точките P' и P'' се намират вътре в окръжността и в I октант то по-малка е грешката за точката която се намира по-надясно (x координатата и е по-голяма). Това и геометрично е очевидно. По аналогичен начин се вижда накъде трябва да се предвижваме за да се доближим до окръжността, когато се намираме в другите октанти и съответно вътре или вън за окръжността.

Ако началния ъгъл е α, то разумно е да вземен $x = R\cos\alpha, y = R\sin\alpha$ и тези стойности да ги закръглим до кратно на h. Мажем да използваме и горните разсъждения за да уточним пиксела от растера на окръжността.

Ако вече сме уточнили крайните пиксели (началния и крайния) то можем да се движим по посока обратна на часовата стрелка като прилагаме правилото на Брезенхам или правилото на средната точка. Например алгоритъма на Брезенхам може да се прилага като имаме предвид в кой квадрант се намираме. Нарстванията по x и y може да се види от таблицата

квадрант	incx	incy
I	-1	1
II	-1	-1
III	1	-1
IV	1	1

Грешката, която беше стойността на функцията в диагоналната точка сега ще стане

$$\Delta = f(D) = (x + incx.h)^2 + (y + incy.h)^2 - R^2$$

Ако означим

$$ddx = 2incx.h.x + h^2,\ ddy = 2incy.h.y + h^2$$

то грешката се променя по правилото:
$P \longmapsto H,\ \ \Delta = \Delta + ddx$
$P \longmapsto V,\ \ \Delta = \Delta + ddy$
$P \longmapsto D,\ \ \Delta = \Delta + ddx + ddy$

Проверките за I квадрант сега са :

Ако $\Delta < 0$, то избираме измежду D и V в зависимост от знака на $\epsilon = f(D) + f(V)$. Ако $\epsilon < 0$ избираме V иначе избираме D

Ако $\Delta > 0$, то избираме измежду D и H в зависимост от знака на $\delta = f(D) + f(H)$. Ако $\delta > 0$ избираме H иначе избираме D

Проверките за II квадрант са (в известен смисъл противоположни):

Ако $\Delta > 0$, то избираме измежду D и V в зависимост от знака на $\epsilon = f(D) + f(V)$. Ако $\epsilon > 0$ избираме V иначе избираме D

Ако $\Delta < 0$, то избираме измежду D и H в зависимост от знака на $\delta = f(D) + f(H)$. Ако $\delta < 0$ избираме H иначе избираме D

Поправките за δ и ϵ са:

$$\delta = 2f(D) + F(H) - f(D) = 2\Delta + [(x + incx.h)^2 + y^2 - R^2] - [(x + incx.h)^2 + (y + incy.h)^2 - R^2] = 2\Delta - ddy$$

$$\epsilon = 2f(D) + F(V) - f(D) = 2\Delta + [(x)^2 + (y + incy.h)^2 - R^2] - [(x + incx.h)^2 + (y + incy.h)^2 - R^2] = 2\Delta - ddx$$

2.7 Алгоритъм на Брезенхам за растеризиране на елипса

Ще предполагаме, че елипсата е зададена с нормално уравнение

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

Както и при окръжността ще използваме уравнението на елипсата във вида

$$f(x, y) \equiv b^2x^2 + a^2y^2 - a^2b^2 = 0.$$

Ще растеризираме само в I квадрант. В останалите квадранти пикселите се получават като се използва симетрията.

Точката от графиката на елипсата, където наклонът е -1, разделя I квадрант на 2 области (на картинката сме ги означили с 1. и 2.) и в зависимост от това в коя област се намираме, ще избираме от 2 пиксела и ще може да приложим алгоритъма на средната точка. Ако се намираме в област 1. и се движим по посока обратна на часовата стрелка, то ако текущия пиксел е $P = (x, y)$, ще избираме измежду вертикалния пиксел $V = (x, y + h)$ и диагоналния $D = (x - h, y + h)$, Виж Fig 2.8. Докато в област 2. ще избираме измежду хоризонталния $H = (x - h, y)$ и диагоналния $D = (x - h, y + h)$ пиксел.

1. Нека стартираме от точка $(a, 0)$, движим се в посока обратна на часовата стрелка и не напускаме област 1. Грешката за текущия пиксел (x, y) сега е

$$E = E_P = f(M) = b^2(x - \frac{h}{2})^2 + a^2(y + h)^2 - a^2b^2.$$

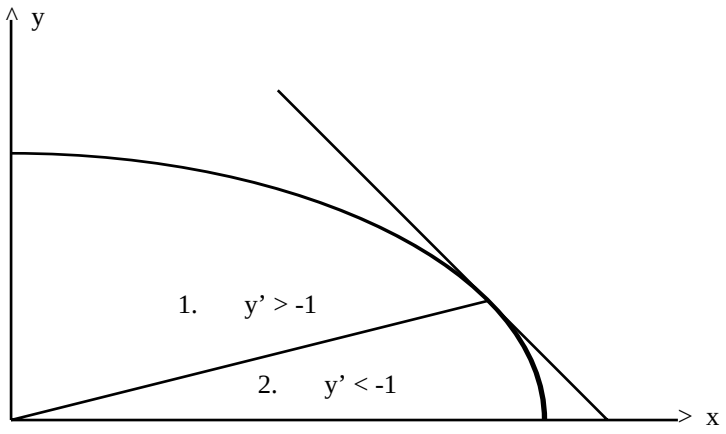


Fig. 2.7

Ако елипсата е минала както линия 1 от картинката, то очевидно ще трябва да изберем точка V. В този случай точка M е вътрешна за елипсата и $E < 0$. Критерия за избор на точка ще бъде знака на грешката. Ако $E < 0$ избираме точка V в противния случай избираме точка D.

1.1. $E < 0$ и е избрана е точка V.

Новата грешка ще бъде

$$E = E_V = f(M') = f(M) + f(M') - f(M) = E_P + [b^2(x - \frac{h}{2})^2 + a^2(y + 2h)^2 - a^2b^2] -$$
$$[b^2(x - \frac{h}{2})^2 + a^2(y + h)^2 - a^2b^2] = E + a^2h(2y + 3h)$$

1.2. $E > 0$ и е избрана е точка D.

Новата грешка ще бъде

$$E = E_D = f(M'') = f(M) + f(M'') - f(M) = E_P + [b^2(x - \frac{h}{2} - h)^2 + a^2(y + 2h)^2 - a^2b^2] -$$
$$[b^2(x - \frac{h}{2})^2 + a^2(y + h)^2 - a^2b^2] = E - 2b^2h(x - h) + a^2h(2y + 3h)$$

2. Нека се намираме в област 2. Сега вече избираме измежду точките H и D. Сега, ако текущия пиксел е пак P то грешката е

$$E = E_P = f(N).$$

2.1. Нека $E > 0$ и избрана е точка H.

Новата грешка ще бъде

$$E = E_H = f(N') = f(N) + f(N') - f(N) = E_P + [b^2(x - 2h)^2 + a^2(y + \frac{h}{2})^2 - a^2b^2] -$$
$$[b^2(x - h)^2 + a^2(y + \frac{h}{2})^2 - a^2b^2] = E - 2b^2h(x - h)$$

2.2. Нека $E < 0$ и избрана е точка D.

Новата грешка ще бъде

$$E = E_D = f(N'') = f(N) + f(N'') - f(N) = E_P + [b^2(x - 2h)^2 + a^2(y + \frac{h}{2} + h)^2 - a^2b^2] -$$
$$[b^2(x - h)^2 + a^2(y + \frac{h}{2})^2 - a^2b^2] = E - 2b^2h(x - h) + a^2h(2y + 3h).$$

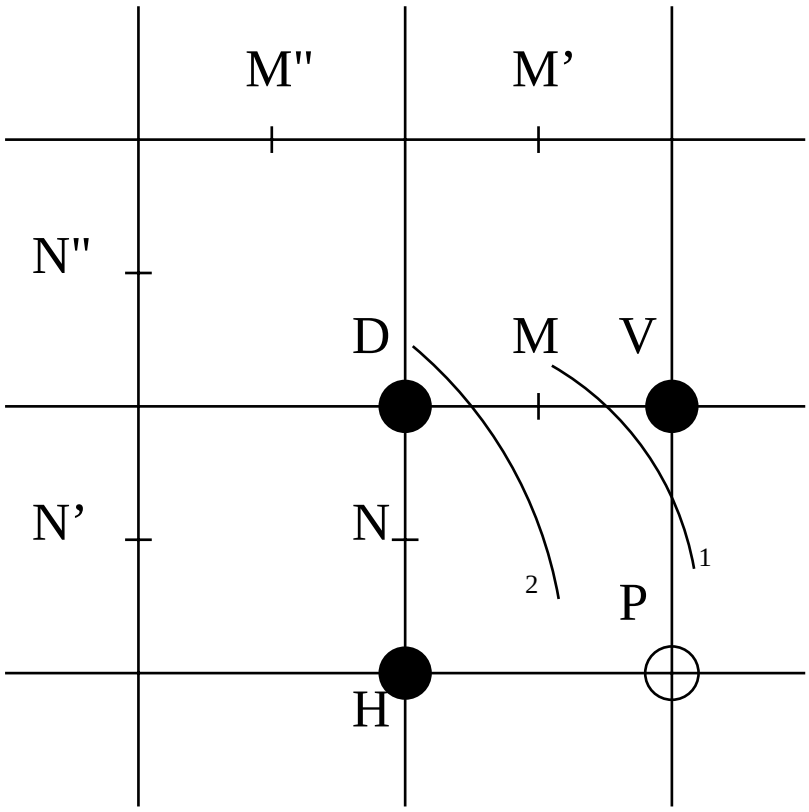


Fig. 2.8

Преминаването от област 1. в област 2. става в точката в която $y' = -1$. От уравнението имаме

$$df = 2b^2xdx + 2a^2ydy = 0$$

От тук имаме

$$\frac{dy}{dx} = -\frac{b^2x}{a^2y} = -1.$$

Тъй като се намираме в I квадрант, то това ще е точката за която

$$b^2x = a^2y$$

Но като вземем в предвид, че смятаме грешката в точката $(x - h/2, y + h)$, то проверката ще е

$$b^2(x - \frac{h}{2}) \leq a^2(y + h).$$

Глава 3

Запълване на области

Областите от пиксели биват 4-свързани и 8-свързани. 4-свързана е такава област, че всеки 2 точки от нея могат да бъдат свързани с последователност от съседни пиксели. Съседни са тези които са разположени един до друг или един над друг. Ако съседни са не само по норизонтала и вертикала, но и по диагонал, то областта е 8-свързана. Освен това областите се делят на вътрешноопределена и граничноопределена. Всяка област се задава спрямо фиксиран пиксел P. Вътрешноопределена област е максимално свързано множество от пиксели имащи стойността на P. Ако стойността на пиксела P е $OldValue$, то пикселите по границата на областта имат стойност различна от $OldValue$. Алгоритмите за запълване на такива области често се наричат $flood - fill$ алгоритми. Областите които са зададени чрез стойността на граничните пиксели се наричат граничноопределени. Това е максимално свързано множество от съседни пиксели, съдържащо пиксела P, всеки от които има стойност различна от граничната - $BoundaryValue$. Ето примерна програма, много неефективна, за запълване на вътрешноопределена област.

```
Flood_fill_4(x,y,OldValue,NewValue)
int x,y,OldValue,NewValue;
      if GetPixel(x,y) == OldValue {
        PutPixel(x,y,NewValue);
        Flood_fill_4(x-1,y,OldValue,NewValue);
        Flood_fill_4(x+1,y,OldValue,NewValue);
        Flood_fill_4(x,y-1,OldValue,NewValue);
        Flood_fill_4(x,y+1,OldValue,NewValue);
      }
```

Може да се направи оптимизация към рекурсивното обръщение, но ние ще разгледаме един много по ефективен алгоритъм за запълване на многоъгълник.

3.1 Алгоритъм на сканиращия ред за запълване

Идеята на алгоритъма е да се движим по всички хоризонтални прави (сканиране) със стъпка h и на всяка права да запълним пикселите принадлежащи на областта. Областта ще предполагаме, че е многоъгълник, което не е ограничение, със зададени върхове $P_1, P_2, ..., P_n$. Ребрата на многоъгълника са отсечките съединяващи 2 съсдни върха (последното ребро е отсечката която съединява последния връх с първия.) Най-напред да направим таблица на ребрата (TP). За целта

1. Отстраняваме хоризонталните ребра
2. За всяко ребро образуваме

Y_{max} - това е по-голямата $y-$ координата на крайните му точки

Y_{min} - това е по-малката $y-$ координата на крайните му точки

X_{min} - това е $x-$ координата на точката за която $y-$координатата е Y_{min}

$IncX$ - изменението на x при нарастването на y с 1.

3. Горната информация се подрежда в таблица на ребрата (ТР). В тази таблица за всяко $y := 0, 1, 2, ...N$, има множество от тройки числа $(Y_{max}, X_{min}, IncX)$, за които $Y_{min} = y$, като тези тройки са сортирани по X_{min}.

$IncX$ ще сметнем по следния начин: Ако правата през точките (x_1, y_1) и (x_2, y_2) има уравнение

$$x = k(y - y_1) + x_1, \text{ където } k = \frac{dx}{dy} = \frac{x_2 - x_1}{y_2 - y_1},$$

($dy \neq 0$ защото сме отстранили хоризонталните ребра) то $IncX = k$.

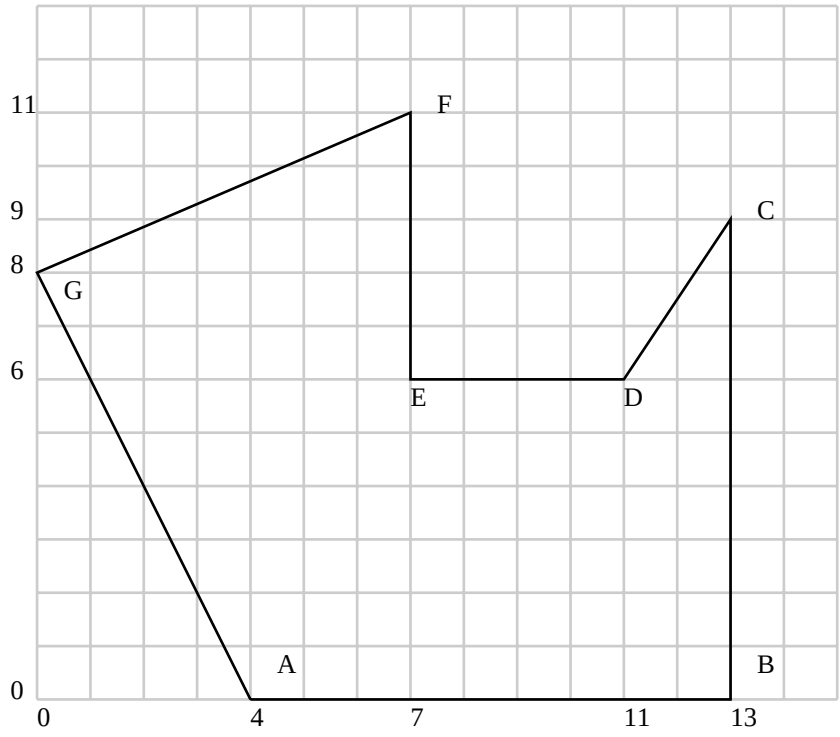


Fig. 3.1

За примера на Fig. 3.1 таблицата на ребрата ще се състои от ребрата BC, CD, EF, FG, GA. За всяко ребро ще трябва да определим числата:

$BC:\ (Y_{max} = 9,\ Y_{min} = 0,\ X_{min} = 13,\ IncX = 0)$

$CD:\ (Y_{max} = 9,\ Y_{min} = 6,\ X_{min} = 11,\ IncX = \frac{2}{3})$

$EF:\ (Y_{max} = 11,\ Y_{min} = 6,\ X_{min} = 7,\ IncX = 0)$

$FG:\ (Y_{max} = 11,\ Y_{min} = 8,\ X_{min} = 0,\ IncX = \frac{7}{3})$

$CD:\ (Y_{max} = 8,\ Y_{min} = 0,\ X_{min} = 4,\ IncX = -\frac{4}{8})$

ТР ще изглежда така:

...
$y = 12 \longrightarrow \{\oslash\}$
...
$y = 8 \longrightarrow \{FG = (11, 0, \frac{7}{3})\}$
$y = 7 \longrightarrow \{\oslash\}$
$y = 6 \longrightarrow \{EF = (11, 7, 0)\}$
$y = 5 \longrightarrow \{\oslash\}$
...
$y = 1 \longrightarrow \{\oslash\}$
$y = 0 \longrightarrow \{GA = (8, 4, -\frac{1}{2}),\ BC = (9, 13, 0)\}$

Тази таблица се използва за поддържане на САР (списък на активните ребра). Това е така да се каже текущото състояние. За всеки текущо обработван ред от растера в САР се съдържат тези ребра от ТР, които имат пресечни точки с този ред. Данните са подобни на тези от ТР:

Y_{max} - най-голямата $y-$ координата на реброто (същото както в ТР)
X - това е $x-$ координата на пресечната точка на реброто с реда
$IncX$ - същото както в ТР.

За сканиращия ред $y = 0$, САР ще се състои от 2 тройки числа:

$$CAP(y = 0) :\ (8, 4, -0.5),\ (9, 13, 0)\}. \tag{3.1}$$

Това са онези ребра от ТР, които имат Y_{min} равно на текущото y, в случая 0. Забелязва се, че броят на активните ребра е четен. За примера той 2. Понеже тройките числа в САР са сортирани по X, то пикселите с първа координата от X на първата тройка (нечетен номер на трройка) до X на втората тройка (четен номер на трройка) ще са от областта и тях ще трябва да оцветим. На следващата стъпка $y = y + 1 = 1$ Ще трябва да се направи промяна в САР само за $X-$ вете. Понеже y е нараснал с 1, то X ще се промени по формулата $X = X + IncX$. За примера:

$$CAP(y = 1) :\ (8, 3.5, -0.5),\ (9, 13, 0)\}. \tag{3.2}$$

Така ще продължим до $y = 6$. В ТР има 2 ребра с $Y_{min} = 6$. Тях ще ги добавим към САР и ще сортираме пак по X. Ще получим

$$CAP(y = 6) :\ (8, 1, -0.5),\ (11, 7, 0)\ (9, 11, 2/3)\ (9, 13, 0)\}. \tag{3.3}$$

Пикселите от областта са тези с първа координата от интервалите [1,7] и [11,13]. На следващата стъпка ще имаме

$$CAP(y = 7) :\ (8, 0.5, -0.5),\ (11, 7, 0)\ (9, 35/3, 2/3)\ (9, 13, 0)\}. \tag{3.4}$$

При следващата стъпка $y = 8$, ще трябва да отстраним реброто GA от САР (защото то има Y_{max} равно на текущото y, т.е. $Y_{max} = 8$. Но ще трябва да добавим реброто FG за което Y_{min} е равно на текущото y. И т.н.

Алгоритъма ще изглежда така:

1. Построяваме ТР за многоъгълника.
2. Инициализираме САР като празен списък.
3. Инициализираме y като най-малкото y в ТР.
4. Повтаряме докато ТР и САР станат празни.

4.1. Добавяме към CAP онези ребра от TP с Y_{min} равно на текушото y. Сортираме CAP по X.

4.1. Запълваме пикселите между всяка двойка от нечетни-четни пресечни точки от CAP.

4.1. Увеличаваме y с 1.

4.1. Изтриваме от CAP онези за които $Y_{max} = y$.

4.1. За всяко ребро от CAP изменяме X с $IncX$.

Глава 4

Геометрични трансформации

В компютърната графика под геометрични трансформации обикновинно се разбира такива трансформации на точки в равнината или в тримерното пространство: $P' = f(P)$ такива че

1. За всяка точка P' съществува точка P, за която $P' = f(P)$ и
2. Различни точки P_1 и P_2 се изобразяват в различни, т.е. $f(P_1) \neq f(P_2)$.

Такива са например трансформациите: транслация, ротация, мащабиране, хомотетия, симетрия и др. Те имат вида

$$\begin{aligned} x' &= a_{11}x + a_{12}y + a_{13}z + b_1 \\ y' &= a_{21}x + a_{22}y + a_{23}z + b_2 \\ z' &= a_{31}x + a_{32}y + a_{33}z + b_3 \end{aligned}$$

Такива трансформации се наричат още афинни.

4.1 Някои трансформации в равнината

Да разгледаме трансформации в равнината, които имат вида

$$\begin{aligned} x' &= ax + cy \\ y' &= bx + dy. \end{aligned}$$

При означенията $P = (x, y),\ P' = (x', y'),\ T = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ Горната трансформация може да се запише накратко като $P' = PT$.

За да разберем значението на параметрите на матрицата T нека най-напред да положим $b = 0, c = 0$. Трансформацията става $x' = ax,\ y' = dy$. Такава трансформация се нарича **мащабиране**. Числата a и d са мащабните множители. Ако $a = -1$ или $d = -1$ или и двата множители са равни на -1, то трансформациите се наричат **симетрия**. Да отбележим, че ако всички точки на даден обект са подложени на такава трансформация то площа на обекта се променя с множител $|ad|$, което е и стойността на детерминантата на T. Примерно, ако всички точки на дадена фигура се подложат на такава трансформация с някоя от следните магрици

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix},\ B = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix},\ C = \begin{pmatrix} 0.001 & 0 \\ 0 & 1 \end{pmatrix},\ D = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix},$$

то при матрицата A, y координатите ще се умножат с 2, при B, ще имаме симетрия относно правата $x = 0$, при C, x координатите ще се свият до 0.001 от тяхната първоначална стойност и при D фигурата ще се свие до вертикална права.

Следващата стъпка ще е да положим $a = 1, d = 1$ (няма мащабиране или симетрия) и да видим ефекта на другите 2 параметри. Сега трансформацията има вида $x' = x+cy,\ y' = bx+y$. Нека първо $b = 1$ и $c = 0$ и да забележим, че точките $A = (1,0), B = (3,0), C = (1,1)D = (3,1)$ отиват в $A' = (1,1), B' = (3,3), C' = (1,2), D' = (3,4)$, т.е. правоъгълника $ABCD$ отива в успоредника $A'B'C'D'$. Такава трансформация се нарича **"shearing"**. Аналогично се получава и когато $b = 0$ и $c = 1$.

Следващата важна трансформация е **ротация**. Нека точката $P = (x, y)$, се завърта около началото на координатната система на ъгъл α и отива в точка P'. Ако $P' = PR$, то искаме да намерим елементите на матрицата на въртене $R = R(\alpha)$. Ще запишем точките в полярни координати. Нека $x = \rho\cos\theta,\ y = \rho\sin\theta$. Тогава

$$x' = \rho\cos(\theta+\alpha) = \rho(\cos\theta\cos\alpha - \sin\theta\sin\alpha) = x\cos\alpha - y\sin\alpha$$
$$y' = \rho\sin(\theta+\alpha) = \rho(\sin\theta\cos\alpha + \cos\theta\sin\alpha) = x\sin\alpha + y\cos\alpha.$$

От тук следва

$$R(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

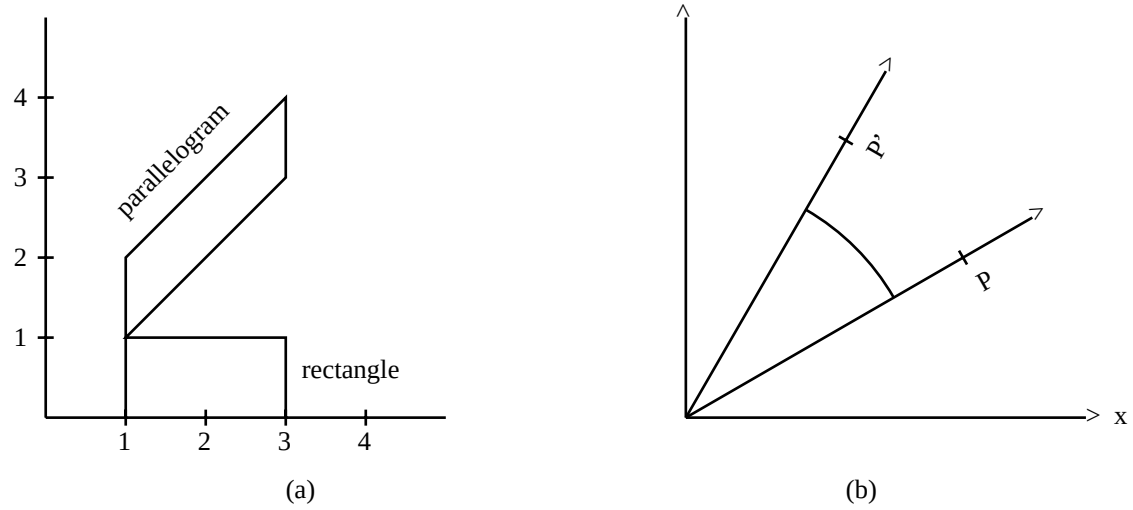


Fig. 4.1

Ако $\cos\theta \neq 0$, то матрицата на въртене може да се представи като

$$R(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} = \begin{pmatrix} \cos\alpha & 0 \\ 0 & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} 1 & \operatorname{tg}\alpha \\ \operatorname{tg}\alpha & 1 \end{pmatrix},$$

което показва че ротацията е комбинация от мащабиране и "shearing".

Матриците на въртене имат следните характерни свойства: Скаларното умножение на стълб със себе дава 1, а скалрното умножение на два различни стълба е 0. Това е вярно и за редовете. Друго характерно свойство е, че транспонираната матрица съвпада с обратната и. Такива матрици (с реални елементи) се наричат ортонормални или матрици на въртене.

С разгледаните матрици не може да се опише една много важна трансформация - транслация а е желателно всички необходими трансформации да се описват като произведение на точка с матрици. Един елегантен начин за целта е да се разглеждат хомогенни координати.

4.1.1 Хомогенни координати

Правилото за представяне на точка $P = (x, y)$ в хомогенни координати е:

1. За да трансформираме координатите (x, y) в хомогенни, добавяме 1 за трета координата, т.е. $(x, y) \Rightarrow (x, y, 1)$.

2. За да трансформираме хомогенни координати (a, b, c) в точка (x, y), делим първите 2 на третата, т.е. $(a, b, c) \Rightarrow (a/c, b/c)$. Ако $c = 0$, то $(a, b, 0)$ представлява вектор с координати (a, b).

Вижда се че точката (x, y) има много хомогенни представяния (ax, ay, a), където $a \neq 0$. Ако a пробягва числата $[0, \infty)$, то тройката (ax, ay, a) пробягва точките от лъча определен от вектора $(x, y, 1)$ в тримерното пространство. За да намерим реалното положение на точката (ax, ay, a) в равнината трябва да намерим пресечната точка на този лъч с равнината $z = 1$.

С хомогенни координате транслация с вектор $t = (t_x, t_y) : x' = x + t_x, y' = y + t_y$ може да се запише по следния начин

$$P' = PT \Longleftrightarrow (x', y', 1) = (x, y, 1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}.$$

В общия случай трансформацията с хомогенни координати ще изглежда по следния начин

$$P' = PT = (x, y, 1) \begin{pmatrix} a & c & p \\ b & d & q \\ t_x & t_y & 1 \end{pmatrix} = (ax + by + t_x, cx + dy + t_y, px + qy + 1). \tag{4.1}$$

От тук

$$x' = \frac{ax + by + t_x}{px + qy + 1}, \quad y' = \frac{cx + dy + t_y}{px + qy + 1}.$$

Да видим каква е ролята на параметрите p и q. Да вземем най-простия случай, когато матрицата е

$$\begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix},$$

т.е. няма мащабиране, няма транслация, няма ротация.

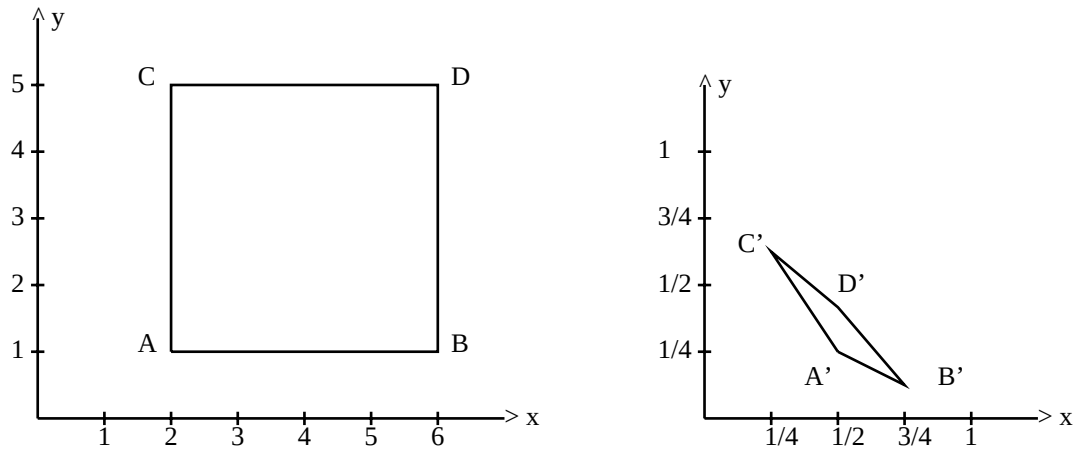


Fig. 4.2

В този случай точките A, B, C, D се трансформират в A', B', C', D', по схемата

$$\begin{aligned} A = (2,1) &\to (2,1,4) \to (1/2, 1/4) = A', \\ B = (6,1) &\to (6,1,8) \to (3/4, 1/8) = B', \\ C = (2,5) &\to (2,5,8) \to (1/4, 5/8) = C', \\ D = (6,5) &\to (6,5,12) \to (1/2, 5/12) = D'. \end{aligned}$$

Оригиналните точки образуват квадрат. Трансформираните точки също образуват квадрат но гледан под различен ъгъл, гледани в перспектива. Квадратът в равнината xOy се трансформира в квадрат но в друга равнина (Fig 4.2). Такава трансформация се нарича **проекция** и е полезна в тримерното пространство.

4.1.2 Умножаване на трансформации

Матричният запис на трансформациите е много удобен, когато трябва да направим последователно няколко трансформации. Тогава просто трябва да умножим матриците им спазвайки реда на прилагането им. Да обърнем внимание, че умножаването на матрици не е комутативно. Трябва да знаем и матриците на обратните трансформации, което не е проблем. Например ето за основните трансформации обратните им:

$$\textbf{Мащабиране: } A = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; A^{-1} = \begin{pmatrix} 1/a & 0 & 0 \\ 0 & 1/b & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; a \neq 0, \; b \neq 0.$$

$$\textbf{Симетрия: } A = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; A^{-1} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; a = \pm 1, \; b = \pm 1.$$

$$\textbf{Shearing: } A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

$$\textbf{Транслация: } A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{pmatrix}, \; A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{pmatrix}.$$

Нека за пример да искаме да приложим последователно симетрия относно правата Oy последвана от мащабиране на y и ротация на $45°$, $(\cos 45 \approx 0.707)$. Тогава ще умножим съответните матрици и ще имаме накрая матрицата на трите трансформации:

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.707 & -0.707 & 0 \\ 0.707 & 0.707 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -0.707 & 0.707 & 0 \\ 1.414 & 1.414 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Вижда се че хомогенния запис тук е излишен. Това е така защото всички трансформации се извършват около началото. Но ако искаме да напрвим въртене около точка P, например, която не е началото на координатната система O, то ще трябва да транслираме точката в началото после да направим ротацията и накрая да върнем точката на мястото и.

Пример: Симетрия относно правата $y = x + 1$

Тази права има наклон 1, значи тя сключва ъгъл $45°$ със оста x и пресича правата y в точката $y = 1$. Първо ще транслираме точката (0,1) в началото, после ще направим ротация на

45°, следва симетрия относно правата x, после обратна ротация и накрая обратна транслация. Ако означим $a = \cos(45°) = \sin(45°)$, то имаме тогава

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} a & -a & 0 \\ a & a & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a & a & 0 \\ -a & a & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}.$$

Пример: Симетрия относно правата $ax + by + c = 0$
Нека точката $P = (x, y)$ и точката $P' = (x', y')$ са симетрични относно правата

$$l: \ ax + by + c = 0. \tag{4.2}$$

Тогава уравнението на правата през двете точки е

$$m: \ -bx + ay + d = 0. \tag{4.3}$$

И точката P' лежи на тази права :

$$m: \ -bx' + ay' + d = 0. \tag{4.4}$$

Като извадим последните 2 уравнения имаме

$$-b(x - x') + a(y - y') = 0. \tag{4.5}$$

Средната точка $M = ((x + x')/2, (y + y')/2)$ лежи на правата l :

$$a\frac{x + x'}{2} + b\frac{y + y'}{2} + c = 0. \tag{4.6}$$

От последните 2 уравнения намираме P' :

$$x' = \frac{(b^2 - a^2)x - 2aby - 2ac}{a^2 + b^2}, \ y' = \frac{(a^2 - b^2)y - 2abx - 2bc}{a^2 + b^2}. \tag{4.7}$$

От тук лесно се вижда, че матрицата на тази симетрия има вида

$$T = \begin{pmatrix} b^2 - a^2 & -2ab & 0 \\ -2ab & a^2 - b^2 & 0 \\ -2ac & -2bc & a^2 + b^2 \end{pmatrix}.$$

Лесно се вижда, че $det(T) = -(a^2 + b^2)^3$. Ако правата беше зададена с нормално уравнение $(a^2 + b^2 = 1)$, то $det(T) = -1$, откъдето следва, че трансформацията е чиста симетрия.

4.1.3 Подобие

Матрицата на подобие има вида

$$T = \begin{pmatrix} a & c & 0 \\ -c & a & 0 \\ m & n & 1 \end{pmatrix}.$$

Лесно се пресмята връзката между разстоянията между 2 точки $P_1 = (x_1, y_1)$ и $P_2 = (x_2, y_2)$ и разстоянието между техните образи $P_1' = P_1T$ и $P_2' = P_2T$:

$$|P_1P_2| = \sqrt{a^2 + c^2}|P_1'P_2'|, \text{ където } |P_1P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Вижда се, че коефициента на подобие е $k = \sqrt{a^2 + c^2}$.

4.2 Трансформации в тримерното пространство

Да припомним, че правоъгълната координатна система в тримерното пространство, която се състои от трите взаимно перпендикулярни оси x, y, z, може да бъде дясно ориентирана или ляво ориентирана. Дясно ориентираната можем да познаем например като си представим, че палеца на дясната ни ръка сочи по посока на оста x, показалеца сочи по посока на оста y а средния пръст ще сочи по посока на оста z. Същия тест важи за ляво ориентираната координатна система, но за лявата ръка. Разликата между двете системи е важна, когато проектираме тримерен обект върху двумерен екран. В компютърната графика екрана обикновено е разположен в равнината xOy така, че долния ляв ъгъл да съвпада с началото на координатната система, оста x да сочи надясно, оста y да сочи нагоре а оста z да сочи от наблюдателя към екрана (наблюдателя стои пред екрана). Обекта който се проектира на екрана е разположен зад екрана, т.е. в полупространството с положителна z координата. В този случай координатната система е ляво ориентирана.

Ще използваме следните означения. Точка P ще трансформираме с матрица T в точка P' по по схемата $P' = PT$. (Има значение дали ще вземаме PT или TP, затова да обърнем внимание, че първо вземаме точката а после матрицата.) Точката $P = (x, y, z)$ ще я представим с хомогенни координати $P = (x, y, z, 1)$ и ще я трансформираме с матрицата T в точка $P' = (X, Y, Z, H) = (x', y', z', 1)$, където $x' = X/H, y' = Y/H, z' = Z/H$ и матрицата T нека има вида

$$T = \begin{pmatrix} a & b & c & p \\ d & e & f & q \\ h & i & j & r \\ l & m & n & s \end{pmatrix}.$$

Сега последния стълб на матрицата $(p, q, r, s)^T \neq (0, 0, 0, 1)^T$ за разлика от двумерния случай и той се използва при проекциите. Матрицата (3x3) в горния ляв ъгъл, както в равнината, влияе за мащабирането (елементите a,e,j), за "shearing"(елементите b,c,f,d,h,i) и за ротация (всичките и елементи). Числата от последния ред l,m,n се използват за транслацията и остават като нови числата от последниа стълб p,q,r,s. За да видим как s влияе на трансформацията, нека T=diag(1,1,1,s). (Елементите на T са 0 с изключение на главния диагонал който е (1,1,1,s).) Тогава $P' = (x, y, z, s) = (x/s, y/s, z/s, 1)$. Следователно параметъра s мащабира едновременно трите координати с множител 1/s. Вижда се, че матрицата T е еквивалентна на матрицата diag(1/s,1/s,1/s,1).

Транслацията е директно обобщение на двумерния случай. Нека да означим матрицата на транслация с вектор $t = (t_x, t_y, t_z)$ с $T(t)$:

$$T(t) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix}.$$

Мащабиране или хомотетия относно началото по трите координатни оси се извършва с матрицата $S(s) = diag(s),\ s = (s_x, s_y, s_z, 1)$. Ако искаме да направим хомотетия относно точка $P_0 = (x_0, y_0, z_0)$, ще трябва да направим трансформация, която е композиция (умножение) на 3 трансформации: 1. Транслация на P_0 в началото - $T(-t)$, където $t = (x_0, y_0, z_0)$; 2. Хомотетия S(s); 3. Обратна транслация на първата $T(t)$.

Shearring. Има 6 основни "shearing"трансформации: $H_{xz}(s)$, $H_{zx}(s)$, $H_{yz}(s)$, $H_{zy}(s)$, $H_{xy}(s)$, $H_{yx}(s)$. Първият индекс означава коя координата се променя ($shear$) а втория индекс казва

коя е координатата която променя първата. Например

$$H_{xz}(s) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; H_{zx}(s) = \begin{pmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

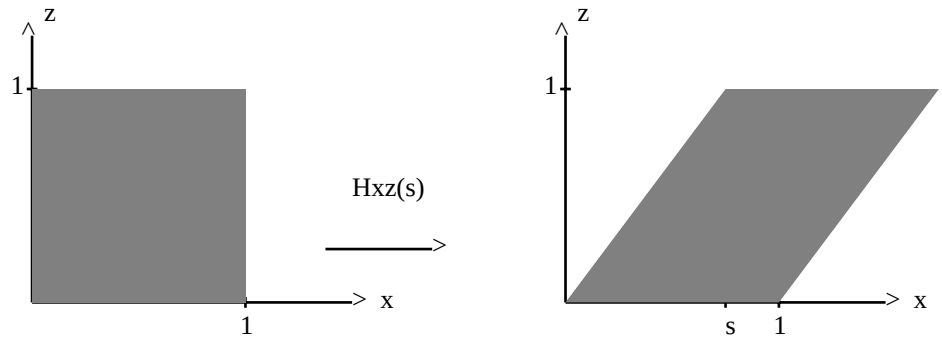


Fig. 4.3

На фигурата се вижда как трансформацията $H_{xz}(s)$ премества квадрат. z координатата влияе на x координатата и трансформераната точка ще има координати:

$$x' = x + sz, \quad y' = y, \quad z' = z.$$

4.2.1 Симетрия относно равнина

Да намерим симетричната точка на дадена точка относно координатните равнини xy, yz, zx е лесно. Трябва само да сменим знака на една от координатите на точката. За произволна равнина нещата са малко по сложни. За целта: (1) Ще си припомним уравнение на равнина; (2) Как да намерим разстояние от точка до равнина и (3) Как да намерим симетричната точка. Уравнение на равнина както знаем има вида

$$Ax + By + Cz + D = 0$$

Ако са дадени 3 точки $P_i = (x_i, y_i, z_i), \; i = 1, 2, 3$, които не лежат на една права, то уравнението на равнината минаваща през тези точки може да се запише във вида

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0.$$

Ако развием датарминантата по първия ред се вижда на колко са равни коефициентите A, B, C, D. Можем да напишем и параметрично уравнение на равнината като $P(u, v) = P_1 + ur + vs$, където $r = P_2 - P_1, \; s = P_3 - P_1$.

Ако е дадена точка $P_1 = (x_1, y_1, z_1)$ от равнината и нормален вектор $N = (N_x, N_y, N_z)$ към равнината то уравнението на равнината можем да намерим по следния начин: Нека

$P = (x, y, z)$ е произвална точка от равнинатал Векторът $P - P_1$ е от равнината и следователно е перпендикулярен на N. Скаларното произведение на тези два вектора ще е 0 и това е уравнението на равнината: $N(P - P_1) = 0$. или още: $NP = s$, където $s = NP_1$. В този случай $(A, B, C) = (N_x, N_y, N_z)$.

Нека сега да е дадена точка $P = (x, y, z)$ вън от равнината, зададена с уравнение $Ax + By + Cz + D = 0$. Искаме да намерим разстоянието от дадената точка до равнината. Нека $P_0 = (x_0, y_0, z_0)$ е произволна точка от равнината. Ако проектираме векторът $P - P_0 = (x - x_0, y - y_0, z - z_0)$ върху нормалата към равнината $N = (A, B, C)$, то големината на тази проекция ще е разстоянието от точката P до равнината. Проекцията е равна на

$$\frac{|A(x - x_0) + B(y - y_0) + C(z - z_0)|}{\sqrt{A^2 + B^2 + C^2}} = \frac{|Ax + By + Cz + D|}{\sqrt{A^2 + B^2 + C^2}}.$$

Можем да вземем този израз без абсолютната стойност в числителя и тогава разстоянието ще е ориентираното разстояние от точката да равнината. Равнината дели пространството но 2 части. Ако точката се намира в онази част от пространството в която сочи векторът N то разстоянието ще е положително, а в другата част - отрицателно.

И накрая да намерим симетричната точка P' на точката $P = (x, y, z)$ относно равнината зададена с уравнение $Ax + By + Cz + D = 0$. Нека ориентираното разстояние от точката P до равнината да е

$$d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}}.$$

Нека единичния вектор перпендикулярен на равнината да е

$$N = \frac{1}{\sqrt{A^2 + B^2 + C^2}}(A, B, C).$$

Тогава

$$P' = P - 2dN = P - 2\frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}}(A, B, C).$$

4.2.2 Ротация

Въртене около координатните оси в тримерното пространство е директно обобщение на въртенето в равнината. Матриците

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

извършват въртене около осите z, y, x съответно. Например да вземем първата матрица. Третия вектор стълб на тази матрица е единичния вектор (0,0,1,0) и следователно умножаването на точка (x,y,z,1) с тази матрица няма да промени z координатата и. Другите 2 координати ще се променят както при ротацията в равнината с ъгъл θ. По трудно е да познаем посоката на въртене. За целта е удобно да вземем ъгъла на въртене да е 90°. Точката (1,0,0) отива в (0,-1,0) и (0,1,0) отива в (1,0,0). Ако гледаме по посока на остта z то се вижда, че въртенето е обратно на часовата стрелка. За втората матрица обаче по същия начин се вижда че въртенето е по посока на часовата стрелка. Третата матрица е като първата. За да имат трите матрици една

и съща посока - например по посока на часовата стрелка избираме

$$R_z(\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.8}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.9}$$

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.10}$$

4.2.3 Ротация в общия случай

Разгледаните ротации са важни частни случаи но не решават въпроса в общия случай. На практика трябва да знаем как да осъществим каква да е ротация. Проблемът е следния:

Точка P трябва да я завъртим на ъгъл θ около дадена ос.

Да обърнем внимание, че остта се състои от начална точка, например P_0 и посока. Ще предполагаме че началната точка съвпада с началото на координатната система. Ако не е така, ще трябва ротацията да я предшествуваме от транслация на P_0 до началото, после ще осъществим ротацията и накрая ще направим обратна транслация. И така, остта около която ще завъртим точката P на ъгъл θ, минава през началото и има посока на единичния вектор **u**.

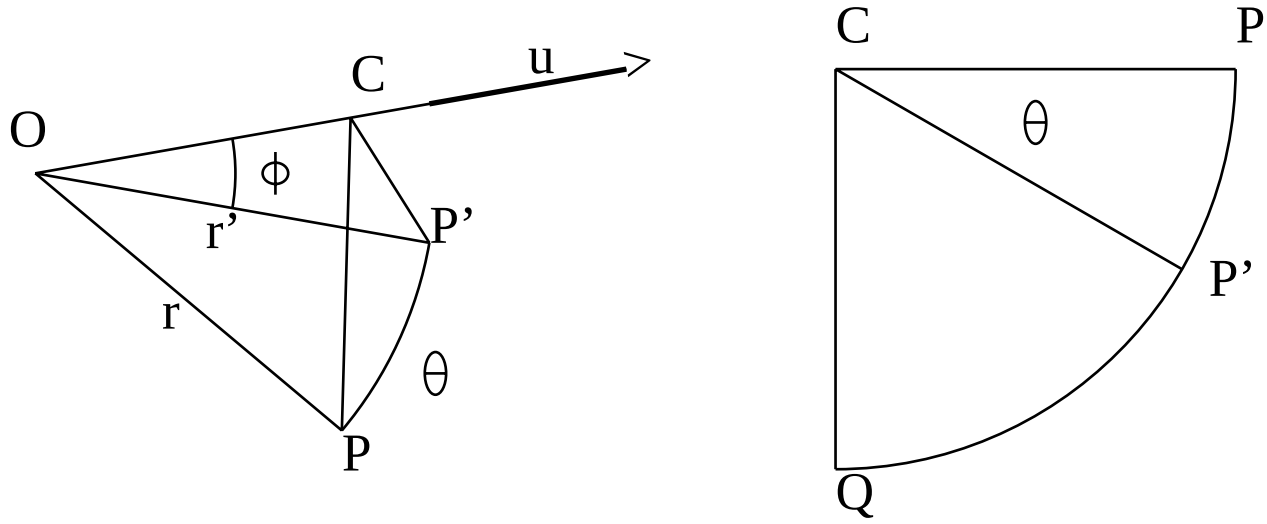


Fig. 4.4

Нека точката P да отива в точката P'. Нека още r и r' са радиус векторите на точките P и P'. Проекциите на тези вектори върху остта съвпадат и нека това да е векторът $\overrightarrow{OC}$. Големината на този вектор е скаларното произведение $(r.u)$. Тогава $\overrightarrow{OC} = (r.u)u$. От тук имаме $\overrightarrow{CP} = r - (r.u)u$ и големината на този вектор е $|CP| = |r - (r.u)u|$. От друга страна от чертежа се вижда, че $|\overrightarrow{CP}| = |r|\sin\phi$. Тъй като $|u| = 1$, то $|r\sin\phi| = |u \times r|$. Така получихме

$|\overrightarrow{CP}| = |r-(r.u)u| = |u\times r|$. На картинката вдясно е показано как изглежда, чертежа завъртян така, че векторът **u** е перпендикулярен на екрана и $\overrightarrow{CP}$ сочи надяасно. Векторът $\overrightarrow{CQ}$, който е по посока на $u \times r$ е перпендикулярен на $\overrightarrow{CP}$. Това е така защото векторното произведение $u \times r$ е перпендикулярно на r и u , а $\overrightarrow{CP}$ е от тяхната равнина.

Тъй като

$$\overrightarrow{CP'} = \overrightarrow{CP}\cos\theta + \overrightarrow{CQ}\sin\theta = \cos\theta[r-(r.u)u] + \sin\theta(u\times r)$$

и $r' = \overrightarrow{OC} + \overrightarrow{CP'}$, то

$$r' = (r.u)u + \cos\theta[r-(r.u)u] + \sin\theta(u\times r). \tag{4.11}$$

Последното равенство можем да запишем и по следния начин

$$r' = r(u^T u) + \cos\theta r - \cos\theta r(u^T u) + \sin\theta rU, \tag{4.12}$$

където

$$U = \begin{pmatrix} 0 & u_z & -u_y \\ -u_z & 0 & u_x \\ u_y & -u_x & 0 \end{pmatrix}.$$

Получихме, че $r' = rM$, където $M = (u^T u) + (I - (u^T u))\cos\theta + \sin\theta U$.

Тук използвахме $(r.u)u = r(u^T u)$, което следва от (**??**) и още $(u \times r) = rU$, което следва от (**??**).

Ротацията на ъгъл $-\theta$ е обратна на горната и има матрица M^T, т.е. ако искаме да завъртим радиус векторът r на ъгъл $-\theta$, то $r' = rM^T$. това лесно се проверява.

4.3 Трансформация на нормалата

Често е необходимо заедно с даден обект да се трансформира и нормалата към някоя от страните му. Ако M е матрицата с която трансформираме някоя от страните (или стените) на обекта, то със същата матрица можем да трансформираме и нормалата. В общия случай обаче така трансформираната нормала няма да е перпендикулярна на страната. Може да се докаже, че ако трансформираме нормалата с матрицата

$$N = (M^{-1})^T$$

то трансформираната нормала ще е перпендикулярна на съответната страна, която се трансформира с матрицата M.

В много от случаите не е необходимо да търсим обратната на M. Например когато матрицата M е ротация то обратната съвпада с транспонираната и матрицата N съвпада с матрицата M. При транслация нормалите се трансформират правилно, което е очевидно. Обикновенно нормалата е нормализирана (дължината и е 1), но след тези трансформации не е необходимо да нормализираме нормалата (да направим дължината и 1) защото дължините не се променят при ротация и транслация. Ако обаче в матрицата M има и мащабиране, то посоката на нормалата се запазва и ще трябва само да ренормализираме тронсформераната нормала.

В общия случай ще трябва да търсим обратната матрица, въпреки че може да си упростим сметките като вземем само транспонираната 3x3 матрица на горния ляв ъгъл на адюнгираните количества на елементите.

4.4 Кватерниони

С кватернионите (Вж. Добавление Б) може лесно да се обясни произволна ротация в тримерното пространство. В равнината ротацията се получаваше с умножаване с матрица на въртене, която имаше вида

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} c & d \\ -d & c \end{pmatrix}.$$

Умножаването на точка (a,b) с такава матрица:

$$(a,b)\begin{pmatrix} c & d \\ -d & c \end{pmatrix} = (ac-bd, ad+bc)$$

е еквивалентно на умножаването на комплексните числа $a+ib$ и $c+id$. С кватернионите това може да се пренесе в тримерния случай по следния начин. Да завъртим точка P на ъгъл θ около ост през началото, имаща посока определена от вектора $\mathbf{v}$, първо да определим кватерниона $\mathbf{q} = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\mathbf{u}]$, където $\mathbf{u} = \mathbf{v}/|\mathbf{v}|$ е единичен вектор по посока на остта. Тогава ротацията се определя от произведението на кватернионите $\mathbf{q}$. $[0,\mathbf{P}].\mathbf{q^{-1}}$. Ще отбележим, че $\mathbf{q}$ е единичен кватернион тъй като $\sin^2\theta/2 + \cos^2\theta/2 = 1$. Тогава $\mathbf{q^{-1}} = \mathbf{q}^* = [\cos\frac{\theta}{\mathbf{2}}, -\sin\frac{\theta}{\mathbf{2}}\mathbf{u}]$.

Упражнение: Да се докаже, че $\mathbf{q}$. $[0,\mathbf{P}].\mathbf{q^{-1}}$ е еквивалентно на (**??**).

Упътване. Да се използва 1) Правилото за умножение на кватерниони: Ако $\mathbf{q_1} = [s_1, \mathbf{v_1}]$, $\mathbf{q_2} = [s_2, \mathbf{v_2}]$ то

$$\mathbf{q_1}.\mathbf{q_2} = [\mathbf{s_1}.\mathbf{s_2} - \mathbf{v_1}.\mathbf{v_2}, \mathbf{s_1}\mathbf{v_2} + \mathbf{s_2}\mathbf{v_1} + \mathbf{v_1}\times\mathbf{v_2}]$$

2) Следното равенство за векторното произведение, когато $\mathbf{u}$ е единичен вектор:

$$(u\times P)\times u = P - (u.P)u$$

3) Тригонометричните тъждества: $\cos\theta = \cos^2\theta/2 - \sin^2\theta/2,\ \sin\theta = 2\sin\theta/2\cos\theta/2$.

Последователното прилагане на ротации ако използваме кватерниони става лесно. Например ако $\mathbf{q_1}$ и $\mathbf{q_2}$ са два единични кватерниона, отговарящи на 2 ротации, то произведението на двете ротации има за кватернион $\mathbf{q_2}\mathbf{q_1}$. Това следва от

$$q_2(q_1[0,P]q_1^{-1})q_2^{-1} = q_2q_1[0,P]q_1^{-1}q_2^{-1} = q_2q_1[0,P](q_2q_1)^{-1}.$$

4.5 Ортографическа проекция

Характерно свойство на тази проекция е че тя проектира успоредни прави в успоредни. Следната матрица е най прост пример на матрица на такаа проекция.

$$O = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Нека P'=PO. Тогава ако P = (x,y,z) то P' = (x,y,0). Следователно точките се проектират в равнината z=0, като проекцията е ортогонална. Очевидно тази трансформация не е обратима. Това следва и от това че det(O) =0. Друг проблем е, че точките с положителна z координата и с отрицателна z координата се проектират в една и съща точка. За да избегнем последното е полезно да ограничим изменението на z (а и $x-$ координатите и $y-$ координатите също) в

някакъв интервал например между n (near plane) и f (far plane). Това е целта на следващата трансформация.

Паралелепипед със страни успоредни на координатните равнини ще означаваме чрез AABB (Axis-Aligned Bounding Box). Ще намерим трансформацията, която трансформира произволен паралелепипед AABB в единичен куб, центриран в началото. Такъв куб ще наричаме каноничен. Нека паралелепипеда да е определен със страни (l,r,b,t,n,f) които означения идват от left, right, bottom, top, near и far равнини. (Лявата равнина има уравнение $x = l$, дясната равнина има уравнение $x = r$ и т.н.)

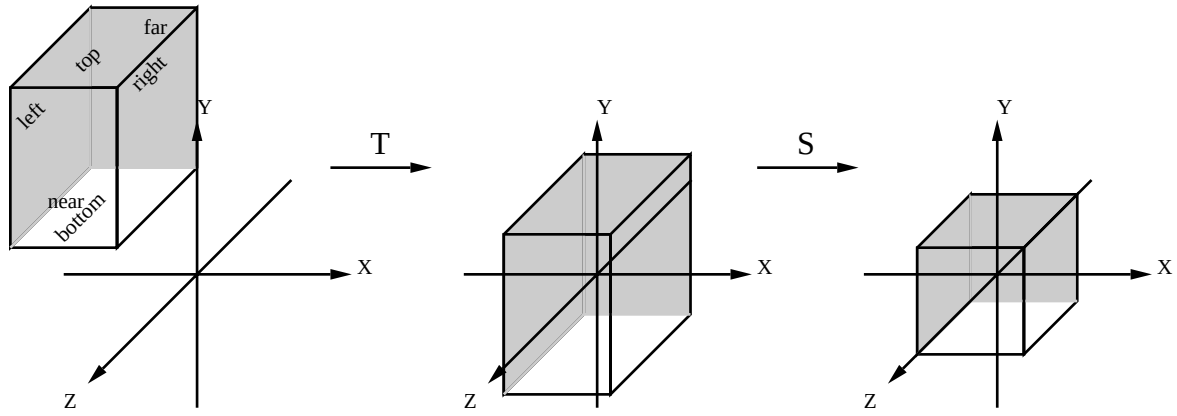


Fig. 6.6

По дефиниция минималния ъгъл на AABB е (l, b, n), а максималния - (r, t, f). Ще отбележим, че $n > f$, защото предполагаме че наблюдателя гледа в отрицателна посока по остта Oz. Това противоречи в известен смисъл на нашите възприятия. Би трябвало $n < f$ (на по-близката равнина да отговаря по-малко число). В OpenGL например $n < f$, като и при тях координатните оси са ориентирани по същия начин.

Единичният куб със страни успоредни на координатните оси (каноничния куб) има минимален ъгъл $(-1, -1, -1)$ и максимален ъгъл $(1, 1, 1)$. Трансформация която трансформира AABB паралелепипед в каноничния куб е полезна в много случаи. Например при отсичане спрямо AABB пресмятанията са облекчени след такава трансформация. Ако означим търсената трансформация с P_o, то очевидно $P_o = TS$, както е показано на фигурата. Да напишем матрицата на тази трансформация:

$$P_o = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{l+r}{2} & -\frac{b+t}{2} & -\frac{n+f}{2} & 1 \end{pmatrix} \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ -\frac{l+r}{r-l} & -\frac{b+t}{t-b} & -\frac{n+f}{f-n} & 1 \end{pmatrix} \tag{4.13}$$

В компютърната графика ляво ориентираната координатна система е по-често срещаната. Това е случая когато ако камерата (или наблюдателя) е пред екрана и гледайки екрана остта X сочи надясно, остта Y сочи нагоре и остта Z сочи към камерата. Тъй като f е по-малка от n, то ортографическата проекция винаги съдържа отразяване. За да видим това нека изходния паралелепипед да съвпада с единичния куб. Тогава $(l, b, n) = (-1, -1, 1)$ и $(r, t, f) = (1, 1, -1)$.

Като заместим в P_o получаваме

$$P_o = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

което е матрица на отразяване. Тази матрица превръща дясно ориентирана координатна система в ляво ориентирана.

Някои системи, като DirectX например, трансформират така че $z \in [0, 1]$ вместо $z \in [-1, 1]$. Тогава ще трябва матрицата P_o да я умножим отляво с матрица на мащабиране и транслация:

$$M_{st} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -0.5 & 0 \\ 0 & 0 & -0.5 & 1 \end{pmatrix},$$

Така че ортографическата матрица използвана в DirectX е

$$P_o = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ -\frac{l+r}{r-l} & -\frac{b+t}{t-b} & -\frac{n}{f-n} & 1 \end{pmatrix}.$$

4.6 Перспективна проекция

Тази проекция дава по-добри визуални резултати отколкото ортографическата. Успоредните линии след проектирането не са вече успоредни. По далечните предмети се проектират в помалки. Перспективната проекция се среща в почти всички графически пакети.

Равнината в която ще проектираме ще бъде $z = -d,\ d > 0$. Нека камерата да е в началото на координатната система. Искаме да проектираме точка $P = (Px, Py.Pz)$ в точка $Q = (Qx, Qy, -d),\ d > 0$.

От подобието намираме

$$\frac{Q_x}{P_x} = -\frac{d}{P_z} \Longleftrightarrow Q_x = -d\frac{P_x}{P_z}.$$

Аналогично намираме

$$Q_y = -d\frac{P_y}{P_z}.$$

От тук следва, че матрицата на перспективната проекция ще бъде

$$P_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/d \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.14}$$

Действително имаме

$$PP_p = (P_x, P_y, P_z, 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/d \\ 0 & 0 & 0 & 0 \end{pmatrix} = (P_x, P_y, P_z, \frac{-P_z}{d}) \Longleftrightarrow (-d\frac{P_x}{P_z}, -d\frac{P_y}{P_z}, -d, 1) = Q.$$

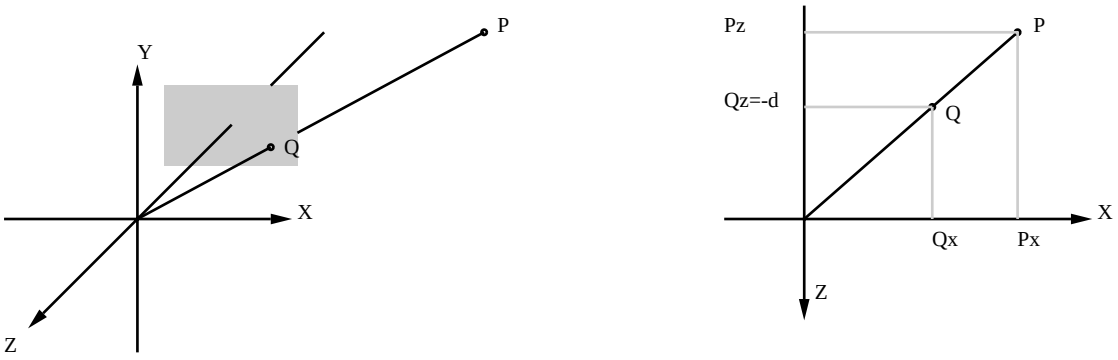


Fig. 6.7

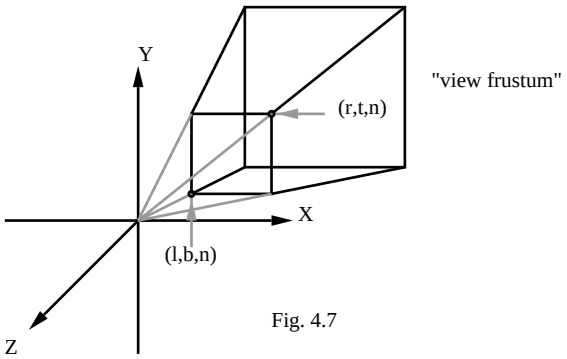


Fig. 4.7

Както при ортографическата проекция, така и тук има трансформация която привежда пресечената пирамида ("view frustum") в каноничния куб (Fig. 4.7).

Нека "view frustum" да започва от $z = n$ и да завършва в $z = f$, $0 > n > f$. Правоъгълника в равнината $z = n$ има минимален ъгъл (l, b, n) и максимален ъгъл (r, t, n).

Параметрита (l, r, b, t, n, f) определят "view frustum" на камерата. Хоризонталното полезрение на камерата се определя от ъгъла между лявата и дясната страна на "view frustum". Също вертикалното полезрение се определя от ъгъла между долната и горната страна. Колкото е по-голямо полезрението толкова повече камерата вижда. За "view frustum" може да се случи $l \neq -r$ или $b \neq -t$ и тогава пресечената пирамида е несиметрична. Ако означим с ϕ хоризонталния ъгъл на полезрението, с w ширината на предмета измерена по права перпендикулярна на правата от камерата към предмета и с d разстоянието от камерата до предмета, то връзката между тези параметри е

$$\phi = 2\text{arctg}\,\frac{w}{2d}$$

Матрицата която трансформира "view frustum" в единичния каноничен куб има вида

$$P = \begin{pmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & \frac{f+n}{f-n} & 1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{pmatrix}. \tag{4.15}$$

Точката $Q' = QP$ ще има четвърта координата Q'_w в общия случай различна от 1 и ще трябва да приведем хомогенните координати в обикновени за да получим координатите на точката Q'. След това може да се прави отсичане например.

Глава 5

Отсичане (Clipping)

Под отсичане (отрязване) се разбира показване на част от обектите които са вътре или вън от дадена област, която се нарича област на отсичане. На фигура 6.1 са показани няколко

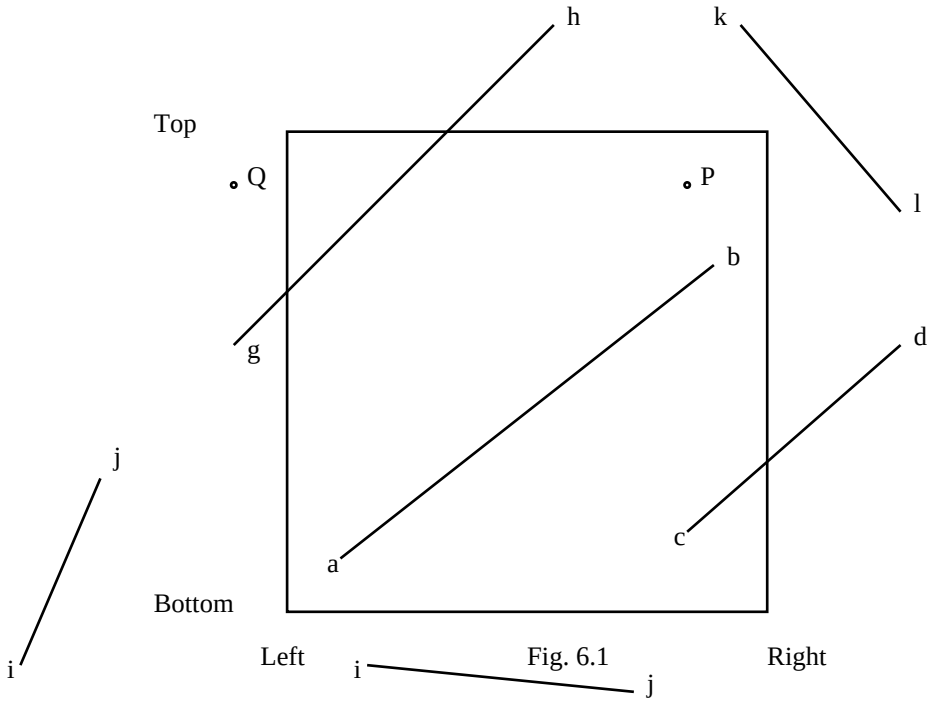


Fig. 6.1

отсечки и точки и прозореца за отсичане който е със страни успоредни на координатните оси и които са означени с Left, Right, Top , Bottom. Целта на алгоритмите за отсичане е да се определи кои точки , отсечки или части от отсечки лежат в прозореца за отсичане. Те ще се визуализират. Останалите точки или отсечки ще се отхвърлят. Тъй като има много такива отсечки за обработване при визуализиране на обекти, то ефикасността на алгоритмите е от съществено значение. В много случаи голяма част от примитивите (тук отсечките) се отхвърлят тъй като са невидими. Такива са например точка Q, отсечка ij. Други се приемат (видими са) - отсечка ab, гочка P, а трета част трябва да се обработят преди да се покажат. Такива са например отсечките cd, gh, kl.

5.1 Един елементарен алгоритъм

Ще казваме, че точка (x, y) е видима (вътре е в прозореца) ако

$$x_L \leq x \leq x_R \text{ and } y_B \leq y \leq y_T$$

Приемаме, че ако точката лежи на някоя от страните на прозореца то тя е видима.

Отсечка на която двата края са вътре в прозореца е видима като ab например. Ако краищата са вън от прозореца, това не означава че отсечката се отхвърля - примерно отсечката cd. Но ако двата края са наляво от лявата страна на прозореца - ($x_A < x_L$ и $x_B < x_L$) то такава отсечка се отхвърля или двата края са надясно от Right или двата края са надолу от Bottom или нагоре от Top. Отсечката gh е частично видима и тя трябва да се обработи (да се намерят пресечните точки със страните на прозореца).

Visibility algorithm Begin
a и b са краищата на отсечката с компоненти x и y
За всяка отсечка:
Visisbility = True
If $x_a < x_L$ and $x_b < x_L$ Then Visibility = False
If $x_a > x_R$ and $x_b > x_R$ Then Visibility = False
If $y_a > y_T$ and $y_b > x_T$ Then Visibility = False
If $y_a < y_B$ and $y_b < y_B$ Then Visibility = False
If Visibiliti <> False Then
Ако някоя координата на краищата е извън прозореца
то отсечката е частично видима
If $x_a < x_L$ or $x_a > x_R$ Then Visibility = Partial
If $x_b < x_L$ or $x_b > x_R$ Then Visibility = Partial
If $y_a < y_B$ or $y_a > y_T$ Then Visibility = Partial
If $y_b < y_B$ or $y_b > y_T$ Then Visibility = Partial
End If
If Visibility = Partial Then
Да се намерят точките на пресичане ако има такива
End If
If Visibility = True Then
Отсечката е изцяло видима. Да се начертае.
End If
End

5.2 Код на точка

Един много ефикасен тест за проверка дали дадена точка принадлежи на прозореца за отсичане или не принадлежи е предложен от Dan Cohen и Ivan Sutherland. Той се състои в следното. На всяка точка (x, y) се съпоставя 4-битов код по правилото:

1 бит = 1 ако точката е наляво от прозореца ($x < x_L$), иначе = 0
2 бит = 1 ако точката е надясно от прозореца ($x > x_R$), иначе = 0
3 бит = 1 ако точката е под прозореца ($y < y_B$), иначе = 0
4 бит = 1 ако точката е над прозореца ($y > y_T$), иначе = 0

Ако прекараме прави през страните на прозореца, те ще разделят равнината на 9 области и за всяка област кода на точка от тази област е показан на фигурата:

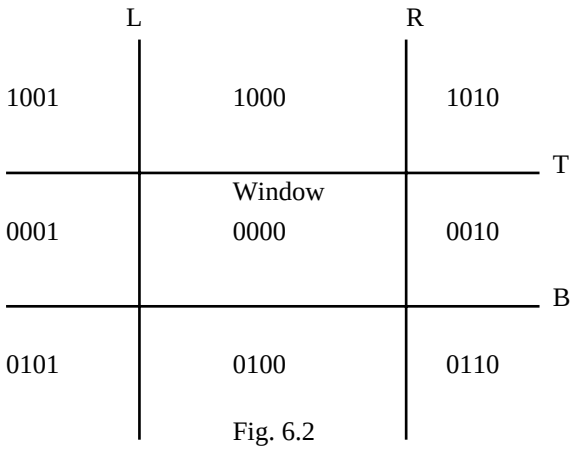


Fig. 6.2

Вижда се, че ако кодовете на двата краища на отсечката са равни на 0 то отсечката е изцяло видима. Ако побитовото събиране на кодовете на краишата е различно от нула то отсечката е изцяло невидима. Тези два теста ще наричаме тривиални. След като преминем през тези два теста остават да се обработят онези отсечки, които имат пресечни точки със страните на прозореца.

Нека Pcode да е 1x4 масив за единия край на отсечката. Примерен код за него е:

subroutine Endpoint(P,Window;Pcode)

P=(x,y), Window е масив съдържащ x_L, x_R, y_T, y_B,
Pcode е 1x4 масив съдържаш кода на P
If $x < x_L$ Then Pcode(4) = 1 Else Pcode(4) =0
If $x > x_R$ Then Pcode(3) = 1 Else Pcode(3) =0
If $y < y_B$ Then Pcode(2) = 1 Else Pcode(2) =0
If $y > y_T$ Then Pcode(1) = 1 Else Pcode(1) =0

Return

На езика Си това може да се запише компактно по следния начин

$$\text{Pcode} = (y > y_T) << 3 + (y < Y_B) << 2 + (x > x_R) << 1 + (x < x_L)$$

като ще трябва променливата Pcode да се обработва побитово по нататък в програмата.

Намирането на пресечните точки на отсечката със страните на прозореца може да се осъществи например по следния начин:

Нека (x_1, y_1) и (x_2, y_2) са краищата на отсечката. Нека правата през двете точки да е с уравнение

$$y = m(x - x_1) + y_1, \quad \text{или } y = m(x - x_2) + y_2, \quad \text{където } m = \frac{y_2 - y_1}{x_2 - x_1}$$

Пресечните точки със страните на прозореца са

$$\begin{aligned}
\text{Left} &: x = x_L, y = m(x_L - x_1) + y_1 && m \neq \infty \\
\text{Right} &: x = x_R, y = m(x_R - x_1) + y_1 && m \neq \infty \\
\text{Top} &: y = y_T, x = \frac{1}{m}(y_T - y_1) + x_1 && m \neq 0 \\
\text{Bottom} &: y = y_B, x = \frac{1}{m}(y_B - y_1) + x_1 && m \neq 0
\end{aligned}$$

Случаите $m = \infty$ $(x_2 = x_1)$ и $m = 0$ $(y_2 = y_1)$ трябва да се разгледат отделно.

5.3 Алгоритъм на Cohen-Sutherland за отсичане в равнината

Алгоритъмът на Cohen-Sutherland използва кода на точка описан по горе за тривиално приемане или отхвърляне на отсечката. Ако отсечката не може тривиално да се приеме или отхвърли, то се намира пресечна точка със една от страните на прозореца и отсечката се разделя на 2 отсечки. Алгоритъма така е построен така, че едната подотсечка се отхвърля и остава другата, за която процедурата се повтаря. За целта се приема за първи край на отсечката онзи който е извън прозореца. В противен случай се разменят двата края на отсечката.

Cohen-Sutherland алгоритъм за отсичане в равнината

Window е масив съдържащ страните на прозореца x_L, x_R, y_B, y_T

$P1 = (x1, y1), P2 = (x2, y2)$ са краищата на отсечката с кодове P1code, P2code

Iflag (за наклона) = -1 ако $m = \infty$, 0 ако $m = 0$, +1 в другите случаи

Vflag (за видимост) = yes (видима), no (невидима), partial (за обработване)

begin

```
call Endpoint(P1,Window;P1code)
call Endpoint(P2,Window;P2code)
call Sum(P1code;Sum1)
call Sum(P2code;Sum2)
call Visible(P1code,P2code,Sum1,Sum2;Vflag)
    /* Тривиални случаи */
    if Vflag = yes then { чертаем отсечката и край }
    if Vflag = no then { край }
    /* Нетривиални случаи */
    Iflag=1 { инициализиране на Iflag }
    if (x1 == x2) then
        Iflag=-1 { вертикална отсечка }
    else if (y1 == y2) then
        Iflag=0 { хоризонтална отсечка }
    else
        Slope = (y2 − y1)/(x2 − x1)
    end if
    while Vflag = partial
        for i=1 to 4 { за всяка страна на прозореца }
            if P1code(5-i) <> P2code(5-i) then
                /* Ако P1 е вътре в прозореца разменяме точките */
                if P1code(5-i) = 0 then
                    Temp = P1 ; P1=P2 ; P2 = Temp
                    Temp = P1code; P1code=P2code; P2code=Temp
                    Temp =Sum1; Sum1=Sum2; Sum2=Temp
                end if
                if (Iflag <> -1) and (i ≤ 2) then /* лява и дясна страна */
                    y1 = Slope * (Window(i)-x1) + y1
                    x1=Window(i)
                    call Endpoin(P1,Window;P1code)
                    call Sum(P1code;Sum1)
                end if
```

```
                    if (Iflag <> 0) and (i > 2) then /* долна и горна страна */
                        if (Iflag <> -1) then /* не вертикална страна */
                            x1 = (1/Slope) * (Window(i) - y1) + x1
                        end if
                        y1 = Window(i) /* ако отсечката е вертикална x1 не се променя */
                        call Endpoin(P1,Window;P1code)
                        call Sum(P1code;Sum1)
                    end if
                    call Visible(P1code,P2code,Sum1,Sum2; Vflag)
                    if Vflag=yes then /* чертаем отсечката и край */
                    if Vflag=no then /* не чертаем отсечката и край */
                end if
            next i
        end while
end
subroutine Visible(P1code,P2code,Sum1,Sum2; Vflag)
        /* да предположим, че отсечката е частично видима */
        Vflag = Patial
        /* проверка за тотално видима отсечка */
        if (Sum1=0) and (Sum2=0) then
            Vflag = yes
        else /* проверка за тривиално невидима */
            call Logical(P1code,P2code;Inter)
            if (Inter <> 0) then Vflag = no
        end if
return
subroutine Sum(Pcode; Sum)
        Sum=0
        for i=1 to 4
            Sum=Sum+Pcode(i)
        next i
return
subroutine Logical(P1code,P2code; Inter)
        Inter=0
        for i=1 to 4
            Inter = Inter + Integer((P1code+P2code)/2)
        next i
return
```

5.4 Алгоритъм на средната точка

В алгоритъма на Cohen-Sutherland се налага за всяка страна на прозореца да се пресмята пресечната точка. В алгоритъма на средната точка се избягват тези изчисления. В този алгоритъм отсечката се дели наполовина и за двете половинки се прилага теста за приемане или отхвърляане и т.н. Деление на 2 е бърза операция (използва се shift) и освен това алгоритъма позволяава паралелно смятане. В някои графични пакети този алгоритъм а хардуарно реализиран.

5.5 Алгоритъм на Cyrus-Beck за изпъкнали области

Нека прозореца за отсичане да е изпъкнала област R. Например изпъкналия многоъгалник FGHKLE на Fig. 5.3

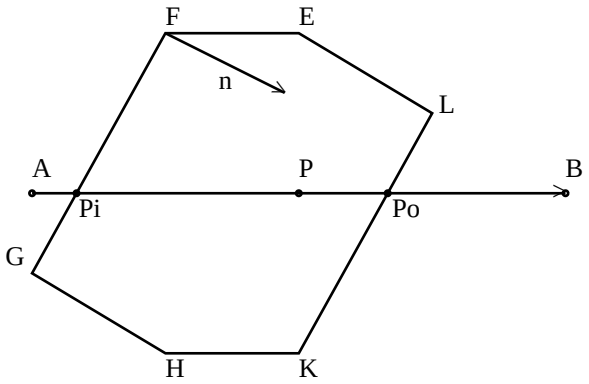


Fig. 6.3

Нека отсечката която ще отсичаме да е AB. Ще използваме параметричен запис на отсечката

$$P = P(t) = A + (B - A)t, \quad 0 \leq t \leq 1. \tag{5.1}$$

Това уравнение е еквивалентно на

$$x_P = x_A + (x_B - x_A)t \tag{5.2}$$

$$y_P = y_A + (y_B - y_A)t \tag{5.3}$$

Разбира се, в уравнението (**??**) t може да се мени от $-\infty$ до $+\infty$ и тогава ще получим цялата права минаваща през точките A и B. Отсечката AB може да пресече изпъкналата област R в 2 точки, както е на фигурата, може да се допира до R (във връх), и тогава приемаме че пресича областта в 1 точка, а може и да не пресича областта и тогава тя е невидима за областта. Нека отсечката пресича областта в 2 точки. Когато t се мени от 0 до 1 точката P пробягва отсечката от A до B и нека t_i да е онова $t \in [0, 1]$ за което P влиза в областта R, P_{in}, а t_o да отговаря на точката в която P излиза от областта, P_{out}. За изпъкнала област тези точки са най-много 2. Разбира се че $0 \leq t_i \leq t_o \leq 1$. За всяака страна на областта нека $\mathbf{n}$ да е вътрешната нормала за областта към тази страна. За страната FG тази нормала е отбелязана. Да припомним как може да намерим тази нормала. Нека $F = (x_F, y_F),\ G = (x_G, y_G)$. Нека $dx = x_G - x_F,\ dy = y_G - y_F$. Тогава векторът $\overrightarrow{FG} = (dx, dy)$ и пеппендикулярния му вектор, който сочи към областта е $\mathbf{n} = (-\mathrm{dy}, \mathrm{dx})$.

Нека сега да видим как можем да намерим пресечната точка на страната FG, например, с отсечката AB. Нека $\mathbf{n}$ да е вътрешната нормала на страната FG. Нека P да е произволна точка от AB. Да разгледаме скаларното произведение

$$S = \overrightarrow{FP}.\mathbf{n} = |\overrightarrow{FP}||\mathbf{n}|\cos\theta.$$

То ще е неотрицателно за точки P от областта R (P от полуравнината където сочи нормалата), ще е равно на 0 когато P лежи на правата FG и ще е отрицателно ако P е от другата полуравнина. Тъй като

$$\overrightarrow{FP}.\mathbf{n} = (\mathbf{P} - \mathbf{F}).\mathbf{n} = (\mathbf{A} + (\mathbf{B} - \mathbf{A})\mathrm{t} - \mathbf{F}).\mathbf{n} = (\mathbf{A} - \mathbf{F}).\mathbf{n} + \mathrm{t}((\mathbf{B} - \mathbf{A}).\mathbf{n}) = -\overrightarrow{AF}.\mathbf{n} + \mathrm{t}\overrightarrow{AB}.\mathbf{n}$$

то от $S = 0$ за t имаме уравнението

$$\mathrm{t}\overrightarrow{AB}.\mathbf{n} = \overrightarrow{AF}.\mathbf{n} \tag{5.4}$$

От тук:

(1) ако коефициента на t е различен от нула намираме

$$t \equiv t_i = \frac{\overrightarrow{AF}.\mathbf{n}}{\overrightarrow{AB}.\mathbf{n}}.$$

Така намереното t за страната FG ще отговаря на точката P_i. За страната KL обаче, намереното t :

$$t \equiv t_o = \frac{\overrightarrow{AK}.\mathbf{n}}{\overrightarrow{AB}.\mathbf{n}}$$

ще отговаря на точката P_o. Как да разграничим двете намерени стойности? Да забележим, че за страните EF и HK $\overrightarrow{AB}.\mathbf{n} = \mathbf{0}$, тъй като нормалите на двете страни са перпендикулярни на AB. Тези страни са успоредни на AB и те нямат пресечни точки с AB. За страните FG и GH скаларното произведение $\overrightarrow{AB}.\mathbf{n}$ е положително (защо?). От двете стойности за t , получени за тези страни, ще трябва да изберем по-голямата и тя ще отговаря на P_{in}. За другите две страни KL и LE скаларното произведение $\overrightarrow{AB}.\mathbf{n}$ е отрицателно (защо?). От двете стойности за t , получени за тези страни, ще трябва да изберем по-малката и тя ще отговаря на P_{out}.

(2) Ако коефициента пред t в (**??**) е равен на нула, то или отсечката AB и съответната страна са успоредни или отсечката AB се изражда в точка (A=B). Когато са успоредни, не намираме t и минаваме на следващата страна от областта или ако лежат на една права и имат общи точки трябва да намерим t_{in} и t_{out}. Когато A=B, трябва да проверим дали точката е от областта (видима е) или е вън от областта (невидима е). Това следва от знака на $w = \overrightarrow{AF}.\mathbf{n}$. Ако $w > 0$ то A е вън от областта, ако $w = 0$ то A лежи на страната и приемаме, че е от областта, и накрая ако $w < 0$ то A е вътрешна за областта.

5.6 Алгоритъм на Liang-Barsky за отсичане в равнината

Този алгоритъм е частен случай на алгоритъма на Сайръс-Бек. Ще предполагаме, че областта за отсичане е правоъгълник със страни успоредни на координатните оси. Нека страните му са зададени чрез правите през x_L, x_R, y_B, y_T. Областта за отсичане R се задава чрез

$$x_L \le x \le x_R$$
$$y_B \le y \le y_T$$

Ако отсечката която ще отсичаме е зададена с краища $A = (x_1, y_1)$ и $B = (x_2, y_2)$ и с параметрично уравнение

$$x(t) = x_1 + (x_2 - x_1)t, \quad t \in [0, 1]$$
$$y(t) = y_1 + (y_2 - y_1)t, \quad t \in [0, 1],$$

то като заместим в горните неравенства ще имаме

$$x_L \le x_1 + tdx \le x_R, \; dx = (x_2 - x_1)$$
$$y_B \le y_1 + tdy \le y_T, \; dy = (y_2 - y_1)$$

Последните неравенства да запишем във вида

$$\begin{aligned} -tdx &\le x_1 - x_L, & tdx &\le x_R - x_1 \\ -tdy &\le y_1 - y_B, & tdy &\le y_T - y_1 \end{aligned}$$

Всяко от тези неравенства има вида

$$td_i \le q_i,\ i = 1, 2, 3, 4, \tag{5.5}$$

където

$$\begin{aligned} d_1 &= -dx, & d_2 &= dx, & d_3 &= -dy, & d_4 &= dy \\ q_1 &= x_1 - x_L, & q_2 &= x_R - x_1, & q_3 &= y_1 - y_B, & q_4 &= y_T - y_1 \end{aligned}$$

и $i = 1, 2, 3, 4$ се отнасят съответно за лявата, дясната, долната и горната страна на отсичащия прозорец.

Вижда, че ако $q_i < 0$, то точката A е извън областта R. Ако някое $d_i = 0$, то отсечката е успоредна на съответната страна на R. Комбинация от $q_i < 0$ и $d_i = 0$ означава, че отсечката е невидима и може тривиално да бъде отхвърлена. От (**??**) следва, че пресечната точка на AB със $i-$тата страна на R ще има $t_i = q_i/d_i$ и нека пресечната точка да е I_i. Ако AB не е успоредна на страната и $t_i \notin [0, 1]$, то тази пресечна точка я отхвърляме. Ако има няколко $t_i \in [0, 1]$, то избираме най-голямото измежду онези за които $d_i < 0$ и най-малкото от онези за които $d_i > 0$. Така избраните ще отговарят на P_{in} и P_{out}.

Liang-Barsky двумерен алгоритъм за отсичане

$A = (x_1, y_1), B = (x_2, y_2)$ са краищата на отсечката
t_i, t_o са параметрите на входната и изходната точка
x_L, x_R, y_B, y_T са страните на прозореца

```
function clipt(d,q;t_i,t_o)
    /* намира max на долните параметри и min на горните */
    visible = true
    if (d=0) and (q<0) then /* отсечката е тривиално невидима */
        visible = false
    else if (d<0) then /* търсим max */
        t=q/d
        if (t > t_o) then
            visible = false
        else if (t > t_i) then
            t_i = t
        end if
    else if (d>0) then /* търсим min */
        t=q/d
        if (t < t_i) then
            visible=false
        if (t < t_o)
            t_o = t
        end if
    end if
    return(visible)
end function
```

```
/* start algorithm */
    t_i = 0, t_o = 1, dx = x_2 − x_1
    if clipt(-dx,x_1 − x_L; t_i, t_o) = true then /* лява страна */
      if clipt(dx,x_R − x_1; t_i, t_o) = true then /* дясна страна */
        dy=y_2 − y_1
        if clipt(-dy,y_1 − y_B; t_i, t_o) = true then /* долна страна */
          if clipt(dy,y_T − y_1; t_i, t_o) = true then /* горна страна */
            if (t_o < 1) then
              x_2 = x_1 + t_o * dx
              y_2 = y_1 + t_o * dy
            end if
            if (t_i > 0) then
              x_1 = x_1 + t_i * dx
              y_1 = y_1 + t_i * dy
            end if
            /* чертаем AB : x1 y1 moveto x2 y2 lineto */
          end if
        end if
      end if
    end if
end algorithm
```

5.7 Алгоритъм на Nicholl-Lee-Nicholl за отсичане в равнината

Този алгоритъм се отнася само за прозорец на отсичане който е със страни успоредни на координатните оси и само за равнината (няма обобщение в R^3). Счита се, че той е по-ефективен от другите алгоритми. Има по-малко сравнения и няма излишни деления.

Равнината се разделя на 3 вида области от отсичащият прозорец, както е показано на Fig. 5.4 Ако отсечката която ще отсичаме е AB, то A може да принадлези на една от трите

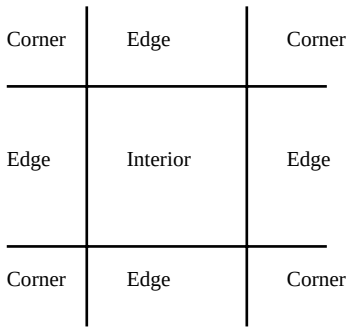


Fig. 6.4

области (Interior, Edge, Corner) а другия край на отсечката в една от 9 области. Ако прекараме лъчи от точката A през ъглите на прозореца то те ще разделят равнината на 4 области. Да означим тези лъчи с името на ъглите през които те минават. На Fig 6.5 те са означени с (LB), (RB), (LT), (RT). Чрез сравняване на наклоните на отсечката AB и тези лъчи, лесно може да

се определи дали отсечката е невидима, или ако е частично видима, броя на пресчните точки със страните на прозореца и с кои страни има пресичане.

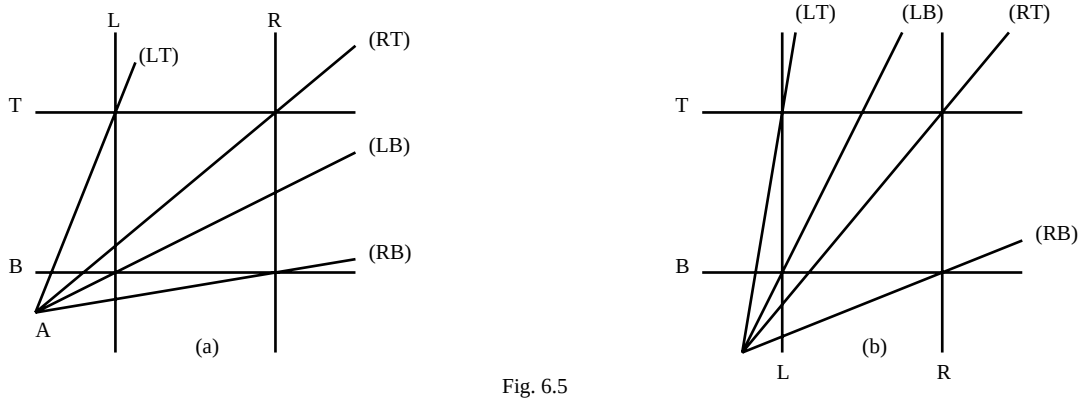


Fig. 6.5

Да разгледаме един от многото случаи. Точката A се намира в долния ляв ъгъл и точката B е външна за прозореца. Нека m е наклона на отсечката AB и нека наклоните на лъчите (правите които съдържат лъчите) са m_{LB}, m_{RB}, m_{LT}, m_{RT}. Тогава имаме следните релации:

if $m > m_{LT}$ then visible = false /* тривиално невидима отесечка */
if $m < m_{RB}$ then visible = false /* тривиално невидима отесечка */
if $m_{RB} \leq m \leq m_{LT}$ then visible = true /* 1 или 2 пресечни точки */
    if $m_{LT} = m$ then /* 1 пресечна точки */
    if $m_{LB} < m < m_{LT}$ then /* пресечни точки с Left и Top */
    if $m_{RT} < m \leq m_{LB}$ then /* пресечни точки с Bottom и Top */
    if $m_{RB} < m < m_{RT}$ then /* пресечни точки с Bottom и Right */
    if $m_{RB} = m$ then /* 1 пресечна точка */

За пресмятане на наклона трябва да извършим 2 изваждания и 1 деление, затова когато е възможно това трябва да се избягва. Ето примерна програма за този случай.

subroutine nln$(A, B, x_L, x_R, y_B, y_T)$
$A = (x_1, y_1)$, $B = (x_2, y_2)$, Точка B е вън от областта */
if $(x_1 < x_L)$ and $(y_1 < y_B)$ then /* A е в долния ляв ъгъл */
    if $(x_2 < x_L)$ then
        visible=false ; exit subroutine
    else if $(y_2 < y_B)$ then
        visible=false ; exit subroutine
    else /* може би частично видима */
        dx=$x_2 - x_1$; dy=$y_2 - y_1$
        dxLT=$x_L - x_1$; dyLT=$y_T - y_1$
        m=dy/dx; mLT=dyLT/dxLT
        if m>mLT then
            visible=false; exit subroutine /* наляво от ъгъл LT */
        else
          if (m=mLT) then
            $x_1 = x_2 = x_L$;
            $y_1 = y_2 = y_T$;
          else
            dxLB=dxLT; dyLB=$y_B - y_1$; mLB=dyLB/dxLB;
            if (mLB < m) and (m < mLT) then /* Left and Top */
              $x_2 = x_1$ + dyLT/m

$y_2 = y_T$
$x_1 = x_L$
$y_1 = y_1$+dxLT*m
else
dxRT=$x_R - x_1$; dyRT=dyLT; mRT=dyRT/dxRT;
if (mRT < m) and (m ≤ mLB) then /* Bottom and Top */
$x_2 = x_1$ +dyRT/m
$y_2 = y_T$
$x_1 = x_1$+dyLB/m
$y_1 = y_B$
else
dxRB=dxRT; dyRB=dyLB; mRB=dyRB/dxRB;
if (mRB < m) and (m < mRT) then /* Bottom and Right */
$y_2 = y_1$+dxRB*m
$x_2 = x_R$
$x_1 = x_1$ + dyRB/m
$y_1 = y_B$
else if (m=mRB) then /* отсечката минава през RB ъгъл */
$x_1 = x_2 = x_R$
$y_1 = y_2 = y_B$
else if (m<mRB) then
visible=false; exit subroutine
end if /* end Bottom and Right */
end if /* end Bottom and Top */
end if /* end Left and Top */
end if /* end LT Corner Intersection */
end if /* passes left of LT */
end if /* $x_2 < x_L$ */
end if /* B в долния ляв ъгъл */

end nln subroutine

Процедури подобно на горната трябва да се напишат и за всички останали случаи.

5.8 Външно и вътрешно отсичане

Разгледаните алгоритми за отсичане досега можем да ги наречем още вътрешно отсичане, защото показваме онази част на отсечката която е вътрешна по отношение на отсичащия прозорец. Но може да искаме да се покаже на дисплея онази част на отсечката, която не принадлежи на прозореца, която е външна по отношение на отсичащия прозорец, и тогава ще наричаме това отсичане външно. Тези отсичания са актуални особено когато показваме много прозорци и обекта трябва да се отсече по отношение на едни прозорци вътрешно, а по отношение на други - външно. Друг пример когато се налага външно отсичане е когато отсичаме по отношение на прозорец, който не е изпъкнал. Тогава може прозореца да се раздели на няколко изпъкнали прозореца и да се отсича външно и вътрешно спрямо различните прозорци.

5.9 Проверка за изпъкналост на многоъгълник и намиране на нормалата

Необходимо е да имаме алгоритъм за познаване дали един многоъгълник е изпъкнал. Ще използваме следния тест за изпъкналост. Двумерен полигон е изпъкнал ако отсечката съединяваща 2 точки от границата му изцяо принадлежи на полигона. Предполагаме че границата принадлежи на полигона. На практика ще използваме знака на векторното произведение на съседните страни на многоъгълника. Ако знака на всички векторни произведения на съседни страни е един и същ, то многоъгълника е изпъкнал. Ако има различни знаци, то многоъгълника не е изпъкнал. Ако всички векторни произведения са нули, то върховете лежат на една права.

Да припомним, че ако $V_i = (x_i, y_i)$, $V_{i+1} = (x_{i+1}, y_{i+1})$ и $V_{i+2} = (x_{i+2}, y_{i+2})$ са 3 съседни върха на многоъгълника то

$$\overrightarrow{V_iV_{i+1}} \times \overrightarrow{V_{i+1}V_{i+2}} = [(x_{i+1} - x_i)(y_{i+2} - y_{i+1}) - (x_{i+2} - x_{i+1})(y_{i+1} - y_i)]\mathbf{k},$$

където $\mathbf{k}$ е единичен вектор перпендикулярен на равнината през точките V_i, V_{i+1}, V_{i+2}. Когато говорим за знака на векторното произведение имаме предвид знака на $(x_{i+1} - x_i)(y_{i+2} - y_{i+1}) - (x_{i+2} - x_{i+1})(y_{i+1} - y_i)$. Ако обикаляме многоъгълника по посока обратна на часовата стрелка (многоъгълника е отляво на $\overrightarrow{V_iV_{i+1}}$) то нормален вектор (вътрешна нормала) към страната V_iV_{i+1}, който сочи към многоъгълника е $\mathbf{n} = (-(y_{i+1} - y_i), (x_{i+1} - x_i))$.

Може да използваме и следния тест за посоката на нормалата. Ако $\mathbf{n}$ е нормалата към страната V_i, V_{i+1} то ако скаларното произведение $\overrightarrow{V_iV_{i+2}}.\mathbf{n}$ е положително, то нормалата е вътрешна, иначе е външна.

Ще опишем една процедура за тестване за изпъкналост използваща транслация и ротация.

1. За всеки връх V_i на полигона транслираме полигона така че върха да отиде в началото.
2. Да се завърти полигона така че V_{i+1} да отиде в положителната част на оста Ox.
3. Проверяваме знака на y-компонентата на V_{i+2}.
4. Ако всички y-компоненти на V_{i+2} имат еднакви знаци, полигонът е изпъкнал. Иначе е вдлъбнат.
5. Ако V_{i+2} връх има нулева y-компонента, то V_i, V_{i+1}, V_{i+2} са колинеарни.
6. За всяка страна на изпъкналия полигон вътрешната нормала има 0 първа компонента и втора компонента равна на знака на y-компонентата на V_{i+2}.
7. За определяне на оригиналната нормала само обратна ротация е необходима.

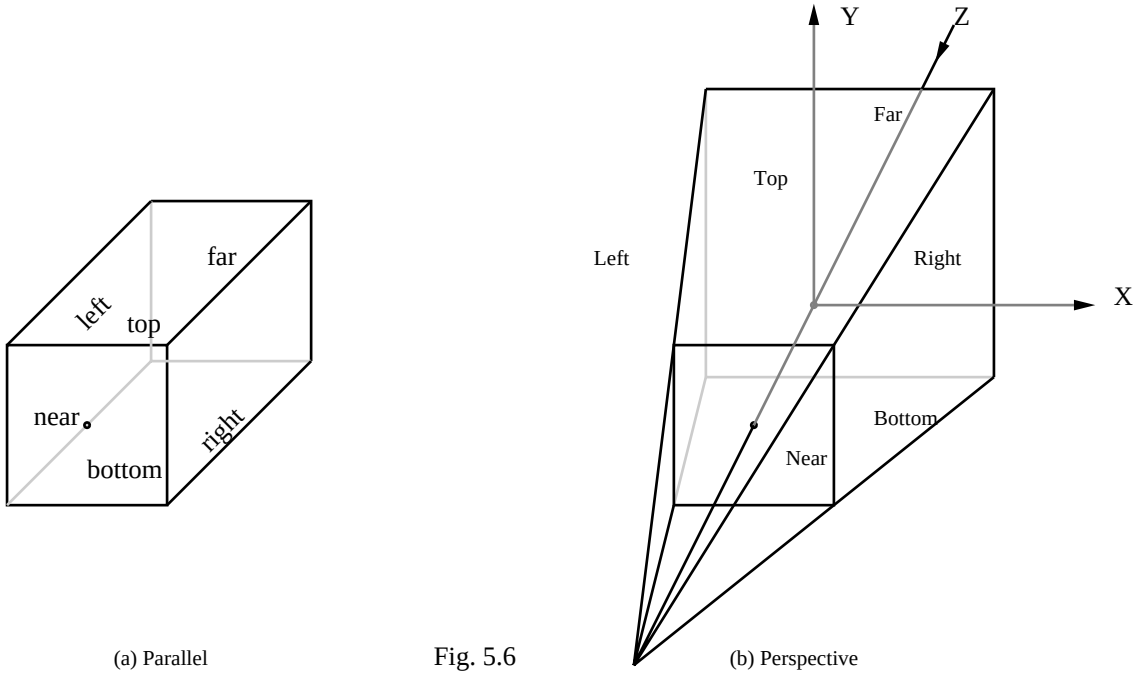
Описаната процедура с малки изменения позволяава да се раздели вдлъбнат полигон на изпъкнали:

1. За всеки връх V_i на полигона транслираме полигона така че върха да отиде в началото.
2. Да се завърти полигона по посока на часовата стрелка, така че V_{i+1} да отиде в положителната част на оста Ox.
3. Проверяваме знака на y-компонентата на V_{i+2}. Ако той е неотрицателен, то полигона е изпъкнал по отношение на тази страна. Ако знака е отрицателен да се разцепи.
4. Разцепване на полигона. Търси се връх след V_{i+2} с неотрицателна y-компонента. Нека той да е V_{i+j}. Полигонът се разцепва на два. Единия е $V_{i+1}, V_{i+2}, ..., V_{i+j}, V_i$. Другия е $V_i, V_{i+1}, V_{i+j}, ..., V_i$.

Този алгоритъм не е оптимален. Освен това не работи вярно ако полигона има самопресичания.

5.10 Отсичане в пространството

Преди да разглеждаме алгоритми в R^3 за отсичане ще дискутираме вида на отсичащата област. Най-често отсичащите области са две: едната е правоъгълен паралелепипед който се използва за ортографическа проекция и отрязана пирамида ("view frustum") използвана за перспективна проекция. Тези области са показани на Fig. 5.6. Тези области имат 6 страни лява, дясна,



(a) Parallel

Fig. 5.6

(b) Perspective

долна, горна, близка и далечна, които са означени с l(eft),r(ight),b(ottom),t(op),n(ear),f(ar). Както в двумерния случай така и в тримерния на всяка точка $P = (P_x, P_y, P_z)$ ще съпоставим код - 6 битов EndPoint код, който е 0 или 1 в зависимост от:

EndPoint(6)=1 ако точката е наляво от лявата страна иначе EndPoint(6)=0
EndPoint(5)=1 ако точката е надясно от дясната страна иначе EndPoint(5)=0
EndPoint(4)=1 ако точката е под долната страна иначе EndPoint(4)=0
EndPoint(3)=1 ако точката е над горната страна иначе EndPoint(3)=0
EndPoint(2)=1 ако точката е пред предната страна иначе EndPoint(2)=0
EndPoint(1)=1 ако точката е зад задната страна иначе EndPoint(1)=0

По същия начин както в равнината имаме тест за тривиално видима отсечка или тривиално невидима. Тривиално видима е тази отсечка за която кода е 0 и за двата края. Тривиално невидима е тази отсечка за която побитовото събиране на кодовете на краищата и е различно от нула. Алгоритъма на Choen-Sutherland за отсичане в равнината се обобщава тривиално и в пространството за горните обеми, особено ако обемът е каноничния куб. Може преди да извършим отсичането да трансформираме правоъгълния паралелепид или "view frustum" в каноничния единичен куб (виж проекции) и след това да извършим отсичането, което значително се упростява. Когато областта на отсичане е "view frustum" ще опишем директно как може да се определи кода на точката.

Ще предполагаме, че центърът на камерата и центърът на "view frustum" съвпадат с остта Z, както е показано на Fig. 5.6. (b). Изглед отгоре е показан на Fig. 5.7.

Уравнението на правата която представлява дясната равнина на "view frustum" е

$$x = \frac{z - Z_{cp}}{Z_f - Z_{cp}} X_r = z\alpha_1 + \alpha_2, \text{ където } \alpha_1 = \frac{X_r}{Z_f - Z_{cp}}, \ \alpha_2 = -\alpha_1 Z_{cp}.$$

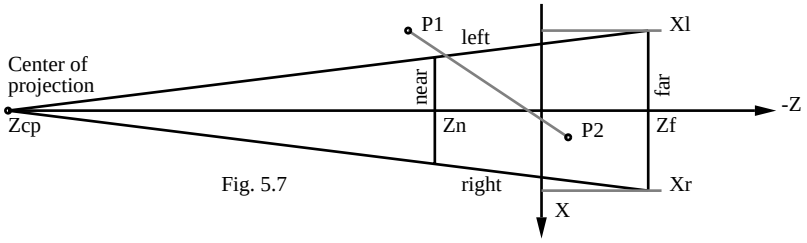


Fig. 5.7

Уравнението на тази равнина се използва за да определим дали точката $P = (x, y, z)$ е наляво, върху или надясно от равнината. Нека

$$F_r(x, y, z) \equiv F_r = x - z\alpha_1 - \alpha_2.$$

Тогава теста за дясната равнина е

Ако $F_r > 0$ то P е надясно от дясната равнина
Ако $F_r = 0$ то P лежи в дясната равнината
Ако $F_r < 0$ то P е наляво от дясната равнина

Аналогично ако означим

$$F_l = x - z\beta_1 - \beta_2,\ F_t = y - z\gamma_1 - \gamma_2,\ F_b = y - z\delta_1 - \delta_2,\ F_n = z - z_n,\ F_f = z - z_f,$$

където

$$\beta_1 = \frac{X_l}{Z_f - Z_{cp}},\ \beta_2 = -\beta_1 Z_{cp},\ \gamma_1 = \frac{Y_t}{Z_f - Z_{cp}},\ \gamma_2 = -\gamma_1 Z_{cp}, \delta_1 = \frac{Y_b}{Z_f - Z_{cp}},\ \delta_2 = -\delta_1 Z_{cp},$$

то теста за другите равнини е:

Ако $F_l > 0$ то P е надясно от лявата равнина
Ако $F_l = 0$ то P лежи в лявата равнина
Ако $F_l < 0$ то P е наляво от лявата равнина

Ако $F_t > 0$ то P е нагоре от горната равнина
Ако $F_t = 0$ то P лежи в горната равнина
Ако $F_t < 0$ то P е под горната равнина

Ако $F_b > 0$ то P е нагоре от долната равнина
Ако $F_b = 0$ то P лежи в долната равнина
Ако $F_b < 0$ то P е под долната равнина

Ако $F_n > 0$ то P е пред близката равнина
Ако $F_n = 0$ то P лежи в близката равнина
Ако $F_n < 0$ то P е зад близката равнина

Ако $F_f > 0$ то P е пред далечната равнина
Ако $F_f = 0$ то P лежи в далечната равнина
Ако $F_f < 0$ то P е зад далечната равнина

Когато $Z_{cp} \to \infty$, то "view frustum" приближава правоъгълния паралелепипед и тестовите функции клонят към съответните на паралелепипеда. При това ако точката е зад центъра на

проектирането, кода и може да не клони към правилния код. Liang и Barsky предлагат ако точката е зад центъра на проекция да се разменят битовете отговарящи на лявата и дясната страна и горната и долната страна.

Ето как ще изглежда процедурата EndPoint за "view frustum".

Subroutine EndPoint(P,Window,Pcode)

$P = (P_x, P_y, P_z)$ Window=$(x_l, x_r, y_b, y_t, z_n, z_f, z_{cp})$ Pcode = Pcode(i), i=1,2,...,6.

/* Пресмятане на $\alpha_i, \beta_i, \gamma_i$ */

$\alpha_1 = x_r/(z_f - z_{cp})$

$\alpha_2 = -\alpha_1 z_{cp}$

$\beta_1 = x_l/(z_f - z_{cp})$

$\beta_2 = -\beta_1 z_{cp}$

$\gamma_1 = y_t/(z_f - z_{cp})$

$\gamma_2 = -\gamma_1 z_{cp}$

$\delta_1 = y_b/(z_f - z_{cp})$

$\delta_2 = -\delta_1 z_{cp}$

/* определяне на кода */

if $P_x - P_z\beta_1 - \beta_2 < 0$ then Pcode(6)=1 else Pcode(6)=0

if $P_x - P_z\alpha_1 - \alpha_2 > 0$ then Pcode(5)=1 else Pcode(5)=0

if $P_y - P_z\delta_1 - \delta_2 < 0$ then Pcode(4)=1 else Pcode(4)=0

if $P_y - P_z\gamma_1 - \gamma_2 > 0$ then Pcode(3)=1 else Pcode(3)=0

if $P_z - z_n > 0$ then Pcode(2)=1 else Pcode(2)=0

if $P_z - z_f < 0$ then Pcode(1)=1 else Pcode(1)=0

return

5.11 Cyrus-Beck алгоритъм в тримерния случай

Този алгоритъм е както в равнината, но с малки изменения. Отсичащата област трябва да е изпъкнал полиедър.

Пример 1. Отсичане спрямо единичния куб.

Следваме алгоритъма в равнината. Нека отсечката, която ще отсичаме да е с краища $A = (-2, -1, 1/2)$ и $B = (3/2, 3/2, -1/2)$. Единичният куб ще бъде определен чрез $(x_l, x_r, y_b, y_t, z_n, z_f) = (-1, 1, -1, 1, 1, -1)$. Лесно се проверява, че вътрешните нормали към страните на куба са:

$$\begin{array}{rrcrcr} \text{Top:} & n_t & = & -j & = & [\ 0 \quad -1 \quad 0] \\ \text{Bottom:} & n_b & = & j & = & [\ 0 \quad 1 \quad 0] \\ \text{Right:} & n_r & = & -i & = & [\ -1 \quad 0 \quad 0] \\ \text{Left:} & n_l & = & i & = & [\ 1 \quad 0 \quad 0] \\ \text{Near:} & n_n & = & -k & = & [\ 0 \quad 0 \quad -1] \\ \text{Far:} & n_f & = & k & = & [\ 0 \quad 0 \quad 1] \end{array}$$

Точки от текущата равнина ще изберем вместо 6 (за всяка равнина по 1) само 2, а именно една обща точка за равнините Top, Right Near: U=(1,1,1) и една за равнините Bottom, Left, Far: V=(-1,-1,-1). Векторът $D = \overrightarrow{AB}$ ще има координати $D = B - A = (7/2, 5/2, -1)$. Също така $\overrightarrow{AU} = (3, 2, 1/2)$, $\overrightarrow{AV} = (1, 0, -3/2)$.

За лявата страна ще имаме: F=(-1,-1,-1); D.$\mathbf{n_l}$ =(7/2,5/2,-1).(1,0,0)=7/2>0; $\overrightarrow{AV}.\mathbf{n_l}$=1. От равенството

$$tD.\mathbf{n_l} = \overrightarrow{AV}.\mathbf{n_l}$$

намираме $t_l = 2/7$. Другите страни за които $D.\mathbf{n_l} > \mathbf{0}$ са долната с $t_b = 0$ и близката с $t_n = -1/2$. От трите t_l, t_b, t_n вземаме най голямото $t_{in} = t_l = 2/7$.

За другите 3 страни съответните t са $t_t = 4/5, t_r = 6/7, t_f = 3/2$. Най малкото от тях е $t_{out} = t_t = 4/5$.

За t_{in} и t_{out} съответните точки от отсечката AB са

$$P(t_{in}) = A + t_{in}(B - A) = (-1, -2/7, 3/14), \text{ и } P(t_{out}) = (4/5, 1, -3/10).$$

Точката $P(t_{in})$ е пресечната точка на отсечката AB с лявата страна, а точката $P(t_{out})$ с горната страна.

Пример 2. Отсичане спрямо перспективния обем ("view frustum").

Нека отсечката, която ще отсичаме да е същата $A = (-2, -1, 1/2)$ и $B = (3/2, 3/2, -1/2)$. Единичният обем да бъде определен чрез $(x_l, x_r, y_b, y_t, z_n, z_f) = (-1, 1, -1, 1, 1, -1)$, с център на проекцията $z_{cp} = 5$. Виж Fig. 5.6 (b) и Fig. 5.7. Вътрешните нормали към близката и далечна страна са очевидни. За останалите страни, които имат обща точка центърът на проекция, ще изберем векторите от центърът на проекция до върховете на близката страна на обема:

$$V_1 = (1, 1, -5),\ V_2 = (-1, 1, -5),\ V_3 = (-1, -1, -5),\ V_4 = (1, -1, -5)$$

и след това ще вземе тяхните векторни произведения за да получим вътрешните нормали на страните. Имаме

$$\begin{array}{rrcrcr} \text{Top:} & n_t & = & V_1 \times V_2 & = & [\ 0 \quad -10 \quad -2] \\ \text{Left:} & n_l & = & V_2 \times V_3 & = & [\ 10 \quad 0 \quad -2] \\ \text{Bottom:} & n_b & = & V_3 \times V_4 & = & [\ 0 \quad 10 \quad -2] \\ \text{Right:} & n_r & = & V_4 \times V_1 & = & [\ -10 \quad 0 \quad -2] \\ \text{Near:} & n_n & = & -k & = & [\ 0 \quad 0 \quad -1] \\ \text{Far:} & n_f & = & k & = & [\ 0 \quad 0 \quad 1] \end{array}$$

За текущи точки за страните left,right,top, bottom избираме общата им точка $F_l = F_r = F_t = F_b = (0, 0, 5)$, а за страните near и far центровете им $F_n = (0, 0, 1)$, $F_f = (0, 0, -1)$. След това следваме алгоритъма. Накрая ще получим пресечните точки на AB с left и top страни на обема (координатите са закръглени):

$$P_{in} = (-0.961, -0.258, 0.203),\ P_{out} = (0.891, 1.065, -0.323).$$

От двата примера става ясен алгоритъма и за произволем изпъкнал обем.

5.12 Liang-Barsky алгоритъм в тримерния случай

Двумерният алгоритъм на Liang-Barsky лесно се обобщава и за тримерния случай. Нека пирамидата сега да е зададена с ъгли на полезрението θ_x и θ_y. Ъгълът θ_x е ъгъла между лявата и дясната страна на "view frustum", като началото е във центъра на проекцията. Аналогично θ_y е ъгъла между долната и горната страна на "view frustum". На Fig. 5.8 е показано сечение в равнината $y = 0$, минаващо през остта Z и перпендикулярно на близката страна. Ще опишем вътрешността на ъгъла чрез ъгъла θ_x. Очевидно за фиксирано $z > 0$, $-x_e \leq x \leq x_e$, където $x_e = z\ \mathrm{tg}\,(\theta_x/2)$. Неравенствата можем да запишем и като

$$-z \leq x \coth \frac{\theta_x}{2} \leq z.$$

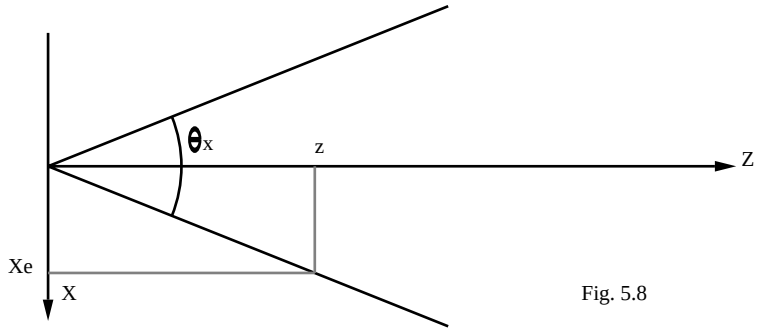


Fig. 5.8

Аналогично и за променливата y ще имаме

$$-z \leq y \coth \frac{\theta_y}{2} \leq z.$$

Правим смяна на променливите

$$x = \coth \frac{\theta_x}{2},\ y = \coth \frac{\theta_y}{2},$$

При тази смяна конуса ще се опише с неравенствата

$$-z \leq x \leq z$$
$$-z \leq y \leq z$$

Нека отсечката AB, която ще отсичаме има параметрично уравнение $P = A + t(B - A)$, което покоординатно може да се запише като

$$\begin{array}{lll} x(t) & = x_A + t(x_B - x_A) & = x_A + dx.t \\ y(t) & = y_A + t(x_B - x_A) & = y_A + dy.t \\ z(t) & = z_A + t(z_B - z_A) & = z_A + dz.t \end{array} \qquad 0 \leq t \leq 1$$

Като заместим в горните неравенства ще получим

$$\begin{array}{llll} -(dx + dz)t & \leq x_A + z_A & (dx - dz)t & \leq z_A - x_A \\ -(dy + dz)t & \leq y_A + z_A & (dy - dz)t & \leq z_A - y_A \end{array}$$

Както в равнината така и сега тези неравенства ще запишем като

$$d_i t \leq q_i, \quad i = 1, 2, 3, 4, \tag{5.6}$$

където

$$\begin{array}{llll} d_1 & = -(dx + dz) & q_1 & = z_A + x_A \\ d_2 & = (dx - dz) & q_2 & = z_A - x_A \\ d_3 & = -(dy + dz) & q_3 & = z_A + y_A \\ d_4 & = (dy - dz) & q_4 & = z_A - x_A \end{array}$$

и left, right, bottom, top страна съответстват на i=1,2,3,4 съответно.

Описания конус ще стане пресечена пирамида, ако ограничим изменението на z. Искаме

$$n \leq z \leq f$$

Като заместим параметичното представяне на отсечката в тези неравенства ще получим

$$\begin{aligned} -dz.t &\leq z_A - n \\ dz.t &\leq f - z_A \end{aligned}$$

които ще ги запишем във вида

$$\begin{aligned} d_5 t &\leq q_5, \quad \text{където } d_5 = -dz \quad q_5 = z_A - n, \\ d_6 t &\leq q_6, \quad \text{където } d_6 = dz \quad q_6 = f - z_A. \end{aligned}$$

Пресечната точка на отсечката с $i-$тата страна на пресечената пирамида има за параметър $t_i = q_i/d_i$. Ако $q_i \geq 0$, то A лежи от видимата страна на страната. Ако $q_i < 0$, то A лежи от невидимата страна на страната. Ако $d_i = 0$ и $q_i < 0$, то отсечката е откъм невидимата страна на областта и е успоредна на страната, значи е тривиално невидима. Псевдо код на алгоритъма ще изглежда по следния начин:

Liang-Barsky R^3 алгоритъм
$A = (x_A, y_A, z_A)$ и $B = (x_B, y_B, z_B)$ са краищата на отсечката
t_{in} и t_{out} са стойностите на параметъра за входната и изходната точка
x_l, x_r, y_b, y_t, n, f са страните на пресечената пирамида
clipt функцията е същата както в R^2
begin
$t_{in} = 0,\ t_{out} = 1$
$dx = x_B - x_A,\ dz = z_B - z_A$
if clipt$(-dx - dz, x_A + z_A, t_{in}, t_{out}) =$ true then
if clipt$(dx - dz, z_A - x_A, t_{in}, t_{out}) =$ true then
$dy = y_B - y_A$
if *clipt*$(-dx - dz, y_A + z_A, t_{in}, t_{out}) =$ true then
if *clipt*$(dy - dz, z_A - x_A, t_{in}, t_{out}) =$ true then
if clipt$(-dz, z_A - n, t_{in}, t_{out}) =$ true then
if clipt$(dz, f - z_A, t_{in}, t_{out}) =$ true then
if $(t_{out} < 1)$ then
$x_B = x_A + t_{out} * dx$
$y_B = y_A + t_{out} * dy$
$z_B = z_A + t_{out} * dz$
end if
if $(t_{in} > 0)$ then
$x_A = x_A + t_{in} * dx$
$y_A = y_A + t_{in} * dy$
$z_A = z_A + t_{in} * dz$
end if
/* draw AB */
end if
end if
end if
end if
end if
end if
end

5.13 Отсичане с хомогенни координати

Трябва да се внимава когато работим с хомогенни координати и когато се използва перспективна трансформация . Проблема е, че дадена равнина не разделя винаги дадена права на 2 части, една откъм видимата страна на равнината и една от невидимата страна. Когато се движим от единия край на отсечката към другия и край трансформираната отсечка може да мине през ∞ точка и да се върне в другата точка, така че да не се разделя на 2 части.

Сайръс-Бек алгоритъм

Алгоритмите на Сайръс-Бек и Лянг-Барски коректно отсичат когато отсечката е изцяло пред центъра на проекция. Но ако отсечката минава зад центъра на проекция, то тези алгоритми отхвърлят отсечката като невидима, независимо че тя е частично видима. На практика трябва да се направи отсичане преди перспективната трансформация. Ще разгледаме 2 примера за целта.

Сайръс-Бек алгоритъм за отсечка с краища от двете страни на центъра на проекция

Нека да отсечем отсечката AB, (A=(0,1,6), B=(0,-1,-6)) с единичен куб определен с $(x_l, x_r, y_b, y_t, n, f) = (-1, 1, -1, 1, -1, 1)$ и с център на проекция $C_{pr} = (0, 0, 5)$. Очевидно отсечката пресича куба, като краищата и са от различни страни на C_{pr}.

Да разгледаме два случая:

1. Нека да направим първо перспективна проекция с матрица

$$P_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{d} \\ 0 & 0 & 0 & 1, \end{pmatrix}$$

където за този пример $d = 5$. Лесно се съобразява, че с тази матрица точка с хомогенни координати $Q = (x, y, z, 1)$ се проектира в хомогенната равнина $h = 1$ и с център на проекция $C_{pr} = (0, 0, d)$ в точка $Q' = (\frac{dx}{d-z}, \frac{dy}{d-z}, \frac{dz}{d-z}, 1)$. Точките от равнината $z = d$ отиват в безкрайната точка. Тогава ако отсечката пресича тази равнина то когато точка се движи от единия край на отсечката към другия край, трансформираните точки ще отиват първо в безкрайната точка когато се приближаваме към равнината $z = d$ след като я преминем ще се връщат от безкрайната точка към другия край.

Намираме $A' = A.P_p = (0, 1, 6, -1/5)$ и $B' = B.P_p = (0, -1, -6, 11/5)$. Забеляазва се, че координатите на трансформираните точки имат противни знаци, което е индикатор, че правата през AB минава през (навива се около) ∞ точка. Това означава, че когато проектираме върху равнината $h = 1$ (хомогенната равнина) правата тръгва от точка A, отива в ∞ точка и се връща в B. Като разделим на хомогенната координата получаваме

$$A' = (0, -5, -30) \qquad B' = (0, -\frac{5}{11}, -\frac{30}{11})$$

И двете точки са зад отсичащия обем. Тогава алгоритъма отхвърля отсечката като невидима.

2. Сега първо да отсечем AB спрямо единичния куб и след това да направим перспективна трансформация.

Следвайки алгоритъма ще трябва да решаваме уравнения от вида

$$t(\mathbf{w}.\mathbf{n}) = (\mathbf{D}.\mathbf{n}),$$

където $\mathbf{n}$ е вътрешната нормала за текущата страна, $\mathbf{w} = \overrightarrow{AF}.\mathbf{n}$, F е точка от текущата страна, и $D = \overrightarrow{AB}$. За всяка страна тези стойности и съответните $t-$та са дадени в следната таблица:

страна	n	F	w	w.n	D.n	t_l	t_u
Top	(0,-1,0)	(1,1,1)	(1,0,-5)	0	2	0	
Bottom	(0,1,0)	(-1,-1,-1)	(-1,-2,-7)	-2	-2		1
Left	(1,0,0)	(-1,-1,-1)	(-1,-2,-7)	-1	0		
Right	(-1,0,0)	(1,1,1)	(1,0,-5)	-5	0		
Near	(0,0,-1)	(1,1,1)	(1,0,-5)	-5	12	5/12	
Far	(0,0,1)	(-1,-1,-1)	(-1,-2,-7)	7	-12		7/12

t_{in} и t_{out} намираме от

$$t_{in} = \max\{t_l : \ D.n > 0\} = \frac{5}{12} \qquad t_{out} = \min\{t_u : \ D.n < 0\} = \frac{7}{12}$$

$$P_{in} = (0, \frac{1}{6}, 1) \qquad P_{out} = (0, -\frac{1}{6}, -1)$$

Като трансформираме тези точки в перспективното пространство намираме

$$A' = P_{in}.P_p = (0, \frac{5}{24}, \frac{5}{4}, 1), \qquad B' = P_{out}.P_p = (0, -\frac{5}{36}, -\frac{5}{6}, 1)$$

Това е коректния резултат.

5.14 Тест за изпъкналост и намиране на вътрешна нормала

Тежниката за тест в равнината за изпъкналост на полигон и за вътрешна нормала може да се приложи и в тримерното пространство за тримерен обем. За всяка страна на обема:

Транслиране обема така, че един от върховете на полигона описващ страната да съвпадне с началото на координатната система.

Завъртаме около началото, така че една от двете страни с връх началото да съвпадне с една от координатните оси, например Ox.

Завъртаме около тази координатна ос, така че страната на полиедъра да съвпадне с една от координатните равнини, например $z = 0$.

Тест за знака на координатните компоненти перпендикулярни на избраната равнина на всички върхове на полиедъра (ако избраната равнина е $z = 0$, то тестваме z координатите на върховете)

Ако всички върхове имат един и същ знак то полиедъра е изпъкнал по отношение на текущата страна. Ако обема е изпъкнал по отношение на всяка от страните, то той е изпъкнал.

Ако всички компоненти са нули то полиедъра лежи в една равнина и не е тримерен а двумерен.

За всяка изпъкнала страна вътрешната нормала има координати нули и знака на координатните компоненти, перпендикулярни на на равнината в която страната лежи. Това е по отношение на завъртяната координатна система. За да определим координатите на оригиналната нормала, трябва да извършим само обратна ротация.

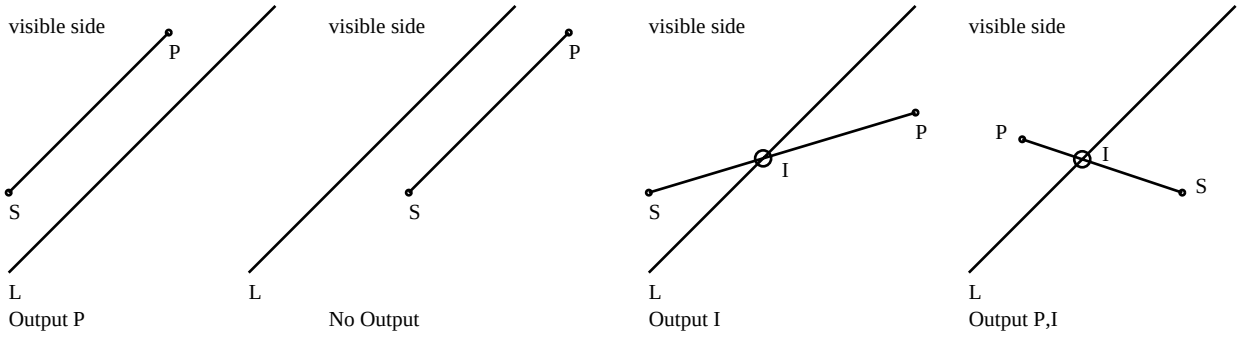
Ако полиедъра не е изпъкнал, той може да се раздели на изпъкнали по аналогичен начин както в равнината.

5.15 Отсичане на полигон в равнината. Алгоритъм на Sutherland-Hodgman.

До тук разглеждахме алгоритми за отсичане на отсечка. Полигонът може да се разглежда като множество от отсечки и всяка от тях да се отсича с някой от алгоритмите разгледани до сега. Но ако полигонът е затворен то отсеченият може да се окаже отворен (няколко отсечки, които не са свързани). Желателно е когато полигонът е затворен и отсечения да е такъв или да е множество от такива.

Идеята на алгоритъма на Съдърланд-Ходжман е първо да се отсече полигона относно една страна на прозореца, след което да се повтори това отсичане за другите страни. Полигонът обикновено е зададен с върховете си: $P_1, P_2, ..., P_n$, които задават страните му: $P_1P_2, P_2P_3, ..., P_nP_1$. Изходът на алгоритъма е множество от върхове, които са от видимата страна на прозореца.

Да видим как да отсечем полигона относно една страна. Нека например тази страна да лежи на права $L-$отсичаща права. Тази права разделя равнината на видима полуравнина (там където е прозореца) и невидима полуравнина. Достатъчно е да обясним как да отсечем една страна (отсечка) на полигона от отсичащата права, тъй като полигонът се състои от много отсечки. Ако всеки връх на полигона P, без първия, разглеждаме като край на отсечката с начало прдхождащия връх, (отсечката е SP) то има 4 възможни случая за отсечката SP и отсичащата права L, както е показано на фигурата. Ако отсечката SP е от видимата страна то



извеждаме само P. S е била изведена на предишната стъпка. Ако отсечката SP е от невидимата страна то не извеждаме точка.

Ако отсечката SP е частично видима, то тя или влиза или излиза от видимата страна. Ако SP напуска видимата страна то трябва да намерим пресечната точка I и да я изведем. Ако SP навлиза във видимата страна то трябва да намерим пресечната точка I и да изведем I и P.

Вижда се че за описание на алгоритъма трябва да знаем как да определим местоположението на точка спрямо права и как да намерим пресечната точка на две отсечки. Когато обикаляма полигона обратно на часовата стрелка то вътрешността на полигона е отляво, а ако обикаляме полигона по посока на часовата стрелка то полигона е отдясно на текущата страна.

Ако имаме права през точките P_i и P_{i+1}, като видима е дясната полуравнина на правата когато се движим от P_i към P_{i+1}, и точка P то има 3 начина да определим положението на точката спрямо правата (в коя полуравнина спрямо правата е точката) : 1) В зависимост от знака на скаларното произведение на нормален вектор към правата и вектор от някоя точка на правата до точка P. 2) В зависимост от знака на числото получено когато заместим координатите на точката в уравнението на правата. 3) В зависимост от знака на векторното

произведение

$$P_iP_{i+1} \times P_iP = (x_{i+1} - x_i)(y - y_i) - (x - x_i)(y_{i+1} - y_i),$$

където $P_i = (x_i, y_i)$ $P_{i+1} = (x_{i+1}, y_{i+1})$ $P = (x, y)$. Ако знака е положителен, нула или отрицателен, то точката е отляво, върху или отдясно на правата с посока от P_i към P_{i+1}.

Намиране на пресечната точка на отсечките P_1P_2 и P_3P_4. Има различни начини, като един от тях използващ параметрично представяне на отсечките е следния: Да представим параметрично отсечките

$$\begin{aligned} P(s) &= P_1 + s(P_2 - P_1) \qquad & 0 \le s \le 1, \\ P(t) &= P_3 + t(P_4 - P_3) \qquad & 0 \le t \le 1 \end{aligned}$$

Пресечната точка е онази за която $P(s) = P(t)$, т.е. $x(s) = x(t)$ и $y(s) = y(t)$. Това са 2 уравнения с дде неизвестни s, t. Ако системата няма рашение, то отсечките са успоредни, ако има решение но някой от параметрите излиза от интервала [0,1] то отсечките не се пресичат.

Алгоритъм на Sutherland-Hodgman за отсичане на полигон
P е масив от точките на входния полигон, Nin е брой на върховете
Q е масив от точките на изходния полигон, Nout е брой на върховете
W е масив от точките на отсичащия прозорец, Nw-1 е брой на върховете
Върховете на всички полигони се обикалят по посока обратна на часовата стрелка
begin

for i=1 to N_w - 1 /* за всяка страна на W */

Nout=0; Q=0;

for j=1 to Nin -1 /* отсичане на всяка страна на полигона */

if j=1 then /* първата точка е специална */

$F = P_j; S = P_j;$

call Visible(S, W_i, W_{i+1};Svisible)

if Svisible ≥ 0 then call Output(S,Nout,Q) ; end if

end if

call Cross(S, P_{j+1}, W_i, W_{i+1}; Spcross) /* проверка дали страната на полигона пресича страната на прозореца*/

if (Spcros = no) then $S = P_{j+1}$ else

/* намиране на пресечната точка и извеждането и */

call Intersect(S, P_j, W_i, W_{i+1};Pintersect)

call Output(Pintersect,Nout;Q)

$S = P_{j+1}$

end if

/* проверка дали втората точка (сега S) е видима */

call Visible(S, W_i, W_{i+1};Svisible)

if Svisible ≥ 0 then call Output(S,Nout,Q) ; end if

next j

/* Завършваме със страната P_nP_1 */

if Nout = 0 then /* ако няма Output прекратяваме цикъла по страните на W */

```
            P=Q; Nin=Nout;
        else
            /* проверяваме дали страната на полигона пресича страната на прозореца */
            call Cross(S, F, W_i, W_{i+1}; Spcross)
            if Spcross=0 then
                P=Q; Nin=Nout;
            else
                /* има пресичане; да намерим пресечната точка*/
                call Intersect(S, F, W_i, W_{i+1};Pintersect)
                call Output(Pintersect,Nout;Q)
                Nin=Nout
            end if
        end if
    next i
end
```

Процедура за пресичане на страните на полигона със страните на отсичащия прозорец

```
subroutine Cross(Start,Point,W1,W2;Spcross)
    call Visible(Start,W1,W2;Pvisible)  проверяваме дали първата точка е видима *
    Pvisible1=Pvisible
    call Visible(Point,W1,W2;Pvisible)  проверяваме дали втората точка е видима *
    Pvisible2=Pvisible
    /* страна на полигона която започва или завършва върху страна на прозореца, се третира
    като страна, която не пресича страната на прозореца. Такава точка е била изведена на
    предишната стъпка /*
    if (Pvisible1 < 0) and (Pvisible2 > 0) or (Pvisible1 > 0) and (Pvisible2 < 0) then
        Spcross = yes
    else
        Spcross = no
    end if
return

subroutine Visible(P,P1,P2;Pvisible)
/*
if Pvisible < 0 then P is to the left (visible)
```

if Pvisible = 0 then P is on the edge P_1P_2
if Pvisible > 0 then P is to the right (invisible)
използваме знака на векторното произведение на P_1P_2 и P_1P
*/
Pvisible = - sign[$(x_2 - x_1)(y - y_1)$ - $(x - x_1)(y_2 - y_1)$]
return

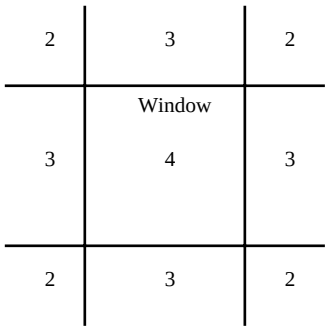
subroutine Intersect(S,P,P1,P2;Pintersect)
/*
Сигурно е че има пресичане. Нека
$S = (x_S, y_S)$, $P = (x_P, y_P)$, $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$
Решаваме системата $t(P_2 - P_1) - s(P - S) = S - P$
Pintersect=$P_1 + t(P_2 - P_1)$
*/
Det=$(x_2 - x_1) * (y_S - y_P) - (y_2 - y_1) * (x_S - x_P)$
$t = (x_S - x_1) * (y_S - y_P) - (y_2 - y_1) * (x_S - x_P)$
t=t/Det
Pintersec = $P_1 + t(P_2 - P_1)$
return

subroutine Output(Pout,Nout,Q)
Nout=Nout+1
Q(Nout)=Pout
return
Този алгоритъм тривиално се обобщава и за тримерното пространство.

5.16 Алгоритъм на Liang-Barsky за отсичане на полигон от прозорец в равнината

Liang-Barsky са разработили нов алгоритъм за отсичане на полигон. Той е особено ефективен за правоъгълен отсичащ прозорец. Тестовете показват че този алгоритъм е два пъти по бърз от алгоритъма на Съдърланд-Ходжман. Всяка страна на отсичащият прозорец разделя

Fig. 5.10



равнината на две полуравнини. Тази полуравнина, която съдържа отсичащия прозорец ще я наричаме видима (вътрешна) а другата невидима (външна). Четирите страни на прозореца разделят равнината на 9 области както е показано на фигурата. По-големия номер на областта индикира, че тя е от видимата страна по отношение на съседната област с по-малък

номер. Само Window (прозореца) се намира във видимата полуравнина на всички страни на прозореца.

Първо ще разгледаме случая, когато страната на полигона P_iP_{i+1} не е успоредна на някоя страна на прозореца (не е хоризонтална или вертикална). С L_i означаваме

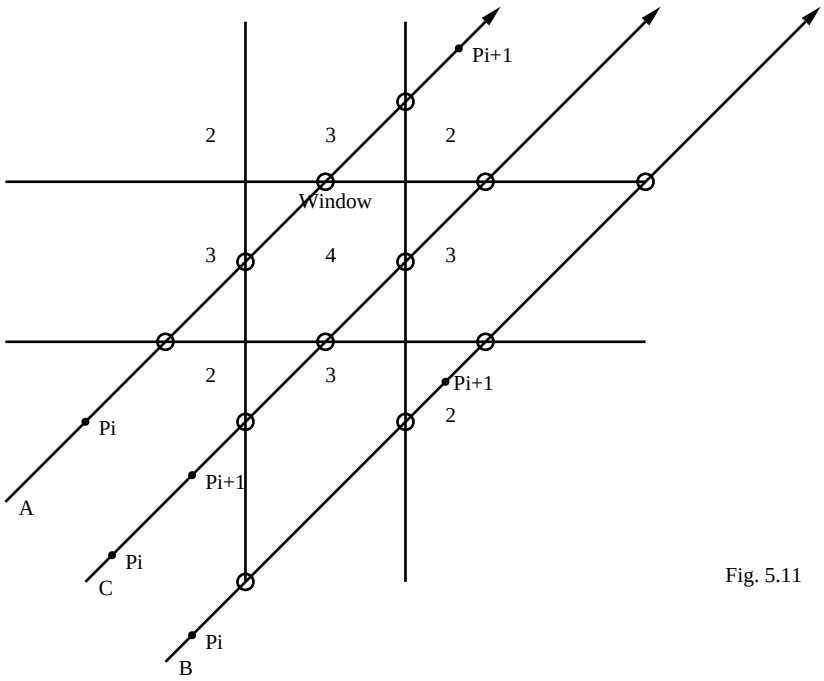


Fig. 5.11

правата съдържаща страната P_iP_{i+1}. Посоката на тази права е от P_i към P_{i+1}. При направеното предположение L_i пресича всяка страна на прозореца. Нещо повече, правата L_i идва от ъглова област (област с номер 2) и завършва пак в ъглова област (диагонално противоположна на първата) както е показано на фигурата. Пресечните точки на правата L_i със страните на прозореца са отбелязани с малки кръгчета. Първата пресечна точка със страна на прозореца е такава, че преминаваме от невидима област към видима и такава пресечна точка ще я наричаме **входяща** пресечна точка. Последната пресечна точка е такава, че правата напуска видима област и навлиза в невидима област и такава пресечна точка ще я наричаме **изходяща**. Средните пресечни точки може да са в ред входяща-изходяща точка както са правите при A и B или изходяща-входяща точка както е при правата C.

Ако междинните пресечни точки са в ред входяща-изходяща точка то отсечката P_iP_{i+1} може да е видима, невидима или частично видима в зависимост от разположението на точките по правата.

Ако междинните пресечни точки са в ред изходяща-входяща точка то отсечката P_iP_{i+1} е невидима.

Как видима, частично видима или невидима страна P_iP_{i+1} на полигона, влияе на изходния полигон зависи и от другите страни на полигона. Ако други страни на полигона влизат в прозореца от различни страни на прозореца, то ще е необходимо да се включи към изходния полигон един или повече върхове на отсичащия прозорец. Тук е разликата между отсичане на отсечка и отсичане на полигон за този алгоритъм. Лиянг и Барски наричат такива върхове **turning vertex** (повратни върхове).

Необходимо условие за съществуване на повратен връх е да съществува за съответната права входяща пресечна точка, която да е извън прозореца. Това условие не е достатъчно. Когато това условие е изпълнено алгоритъма добавя повратния връх, най-близък до входящия

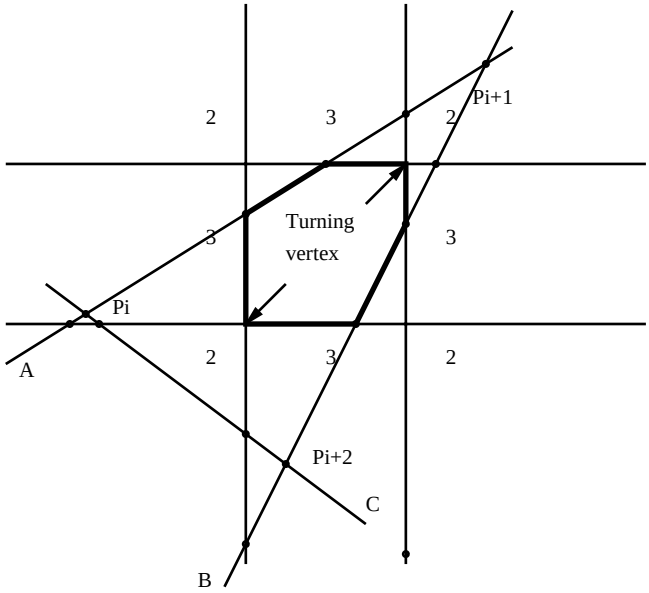


Fig. 5.12

връх, към изходния полигон. Но така повече върхове на прозореца може да се добавят към изходния полигон. Това е недостатък на този алгоритъм.

Описание на алгоритъма. По-удобно е отсечката от полигона P_iP_{i+1} да се запише параметрично:

$$\begin{aligned} x &= x_i + t.dx \qquad \text{където } dx_i = x_{i+1} - x_i \\ y &= y_i + t.dy \qquad \text{където } dy_i = y_{i+1} - y_i \end{aligned}$$

Нека отсичащия прозорец да е зададен с неравенствата

$$\begin{aligned} x_L \le x \le x_R \\ y_B \le y \le y_T \end{aligned}$$

Ще използваме индексите e и l от *entering* и *leaving* за входна и изходна пресечна точка. От горните равенства и неравенства за параметъра t намираме

$$\begin{aligned} t_{x_e} &= (x_e - x_i)/dx_i \qquad dx_i \neq 0 \\ t_{x_l} &= (x_l - x_i)/dx_i \qquad dx_i \neq 0 \\ t_{y_e} &= (y_e - y_i)/dy_i \qquad dy_i \neq 0 \\ t_{y_l} &= (y_l - y_i)/dy_i \qquad dy_i \neq 0 \end{aligned}$$

Тук за определените стойности имаме още

$$x_e = \begin{cases} x_L, & dx_i > 0 \\ x_R, & dx_i \leq 0 \end{cases}$$
$$x_l = \begin{cases} x_R, & dx_i > 0 \\ x_L, & dx_i \leq 0 \end{cases}$$
$$y_e = \begin{cases} y_B, & dy_i > 0 \\ y_T, & dy_i \leq 0 \end{cases}$$
$$y_l = \begin{cases} y_T, & dy_i > 0 \\ y_B, & dy_i \leq 0 \end{cases}$$

От тук се вижда, че параметъра за първата и втората входни и изходни пресечни точки е равен на

$$t_{e_1} = \min(t_{x_e}, t_{y_e})$$
$$t_{e_2} = \max(t_{x_e}, t_{y_e})$$
$$t_{l_1} = \min(t_{x_l}, t_{y_l})$$
$$t_{l_2} = \max(t_{x_l}, t_{y_l})$$

За отсечката P_iP_{i+1} има 3 условия, които нямат отношение към изходния полигон:

1. Отсечката свършва преди първата входна пресечна точка ($1 < t_{e_1}$). (Права A на Fig. 5.12)

2. Отсечката започва преди първата входна пресечна точка ($0 \geq t_{e_1}$) и свършва преди втората входна пресечна точка ($1 < t_{e_2}$). (Права B на Fig. 5.12)

3. Отсечката започва след втората входна пресечна точка ($0 \geq t_{e_2}$) и след първата изходна пресечна точка ($0 \geq t_{l_1}$). (Права C на Fig. 5.12)

Има също 3 условия, които имат съществено отношение към изходния полигон:

4. Отсечката изцяло или частично лежи в прозореца ако втората входна пресечна точка ($0 \geq t_{e_2}$) е преди първата изходна пресечна точка ($t_{e_2} \leq t_{l_1}$) и P_i лежи на L_i преди първата изходна пресечна точка ($0 \geq t_{l_1}$), и P_{i+1} лежи на L_i след втората входна пресечна точка ($0 \geq t_{e_2}$). (Прави D, E и H на Fig. 5.12)

   Ако отсечката е частично видима, намира се пресечната точка и се извежда за подходяща стойност на параметъра и подходяща координата. Другата координата е някоя страна на прозореца.

Следващите 2 условия намират и повратните върхове и ги извеждат в изходния полигон. Това са случаите когато входната пресечна точка лежи на отсечката P_iP_{i+1} и е вън от прозореца:

5. Ако първата входна пресечна точка лежи на P_iP_{i+1} ($0 < t_{e_1} \leq 1$), тогава повратен връх е (x_e, y_e) (Права G на Fig. 5.12)

6. Ако втората входна пресечна точка лежи на P_iP_{i+1} $(0 < t_{e_2} \leq 1)$ и е извън прозореца $(t_{l_1} < t_{e_2})$, тогава повратен връх е или (x_e, y_l), когато втората входна пресечна точка лежи на вертикална страна на прозореца $(t_{x_e} > t_{y_e})$ или (x_l, y_e) когато втората входна пресечна точка лежи на хоризонтална страна на прозореца $(t_{y_e} > t_{x_e})$. (Права I на Fig. 5.12)

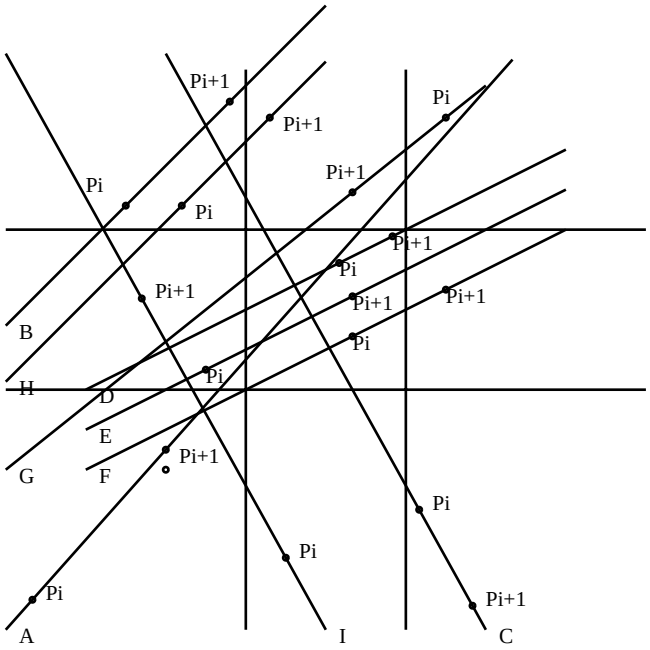


Fig. 5.13

Тези условия са дадени в следващата таблица.

N	Условие	Следствие
1	$1 < t_{e_1}$	няма
2	$0 \geq t_{e_1}$ и $1 < t_{e_2}$	няма
3	$0 \geq t_{e_2}$ и $0 \geq t_{l_1}$	няма
4	$t_{e_2} \leq t_{l_1}$ и $0 \leq t_{l_1}$ и $1 \geq t_{e_2}$	видима отсечка
5	$0 \leq t_{e_1} \leq 1$	повратен връх (x_e, y_e)
6	$0 \leq t_{e_1} \leq 1$ и $t_{l_1} < t_{e_2}$	повратен връх (x_e, y_l) ако $t_{x_e} > t_{y_e}$ или повратен връх (x_l, y_e) ако $t_{y_e} > t_{x_e}$

Условията, с изключение на 5, са независими едно от друго. Ако е изпълнено условие 5 може от условие 4, ако отсечката пресича прозореца, видима порция от нея да се образува, или ако отсечката е изцяло извън прозореца допълнителен повратен връх да се образува по условие 6. Условията 4 и 6 не могат да се случат едновременно.

Хоризонтални и вертикални отсечки: Тези отсечки лесно се проверяват. Ако $dx = 0$ или $dy = 0$ то отсечката е вертикална или хоризонтална.

Нека отсечката е почти хоризонтална. (Прави E и F на Fig. 5.12) За такива отсечки имаме

$$t_{y_e} \leq t_{x_e} \leq t_{x_l} \leq t_{y_l}$$

Ако такава отсечка клони към хоризонтална, то $dy \to 0$ и $t_{y_e} \to -\infty$ и $t_{y_l} \to \infty$. Така че хоризонталната права се характеризира с

$$-\infty \leq t_{x_e} \leq t_{x_l} \leq \infty$$

Аналогично, вертикалната отсечка се характеризира с

$$-\infty \leq t_{y_e} \leq t_{y_l} \leq \infty$$

За почти вертикална или хоризонтална права имаме (Права B)

$$t_{y_e} \leq t_{y_l} \leq t_{x_e} \leq t_{x_l} \text{ или } t_{x_e} \leq t_{x_l} \leq t_{y_e} \leq t_{y_l}$$

в зависимост от посоката на правата. Ако правата е хоризонтална то t_{y_e} и t_{y_l} стават ∞ и неравенствата стават

$$-\infty \leq -\infty \leq t_{x_e} \leq t_{x_l} \text{ или } t_{x_e} \leq t_{x_l} \leq \infty \leq \infty$$

в зависимост от посоката на правата.

Аналогично за вертикална права имаме

$$t_{y_e} \leq t_{y_l} \leq \infty \leq \infty \text{ или } -\infty \leq -\infty \leq t_{y_e} \leq t_{y_l}$$

в зависимост от посоката на правата.

Структура на алгоритъма

За всяка страна на полигона P_iP_{i+1} :

да се определи посоката на отсечката и дали тя е хоризонтална, вертикална или друга
да се определи реда на пресечните точки с прозореца
да се определи стойността на t за входните точки с прозореца и
стойността на t за първата изходна пресечна точка
анализ на страните
if $1 \geq t_{e_1}$ then /* условия 2, 3, 4 или 6 и не 1 */
  if $0 < t_{e_1}$ then /* условие 5 - повратен връх */
    call output(повратен връх)
  end if
  if $1 \geq t_{e_2}$ then /* условия 3 или 4 или 6 и не 1 или 2*/
  /* определяне на t за втора изходна пресечна точка, има output */
    if $0 < t_{e_2}$ or $0 < t_{l_1}$ then /* условия 4 или 6 не 3 */
      if $t_{e_2} \leq t_{l_1}$ then /* условие 4 */
        call output(подходяща пресечна точка)
      else /* свършва условие 4 и започва условие 6 */
        call output(подходящ повратен връх)
      end if /* свършва условие 6 */
    end if /* свършват условия 4 или 6 */
  end if /* свършват условия 3 или 4 или 6 */
end if /* свършват условия 2 или 3 или 4 или 6 */
next i

Псевдокод на Лианг-Барски алгоритъм за отсичане на полигон

subroutine lbpolyclipt(Nin,x,y,W;Nout,Q)
Nin - брой върхове на входния полигон
x - масив с първите координати на входния полигон
y - масив с вторите координати на входния полигон

W = (x_L, x_R, y_B, y_T)
Nout - брой върхове на изходния полигон
Q - масив с върхове на изходния полигон
Всички върхове се обикалят по посока обратна на часовата стрелка
Предполагаме че ∞ се представя с някое реално число, което не се среща в програмата
Nout=0
x(Nin+1) = x(1)
y(Nin+1) = y(1) /* затваряме полигона */
for i=1 to Nin /* започва главния цикъл */
  $dx = x_{i+1} - x_i$
  $dy = y_{i+1} - y_i$
  if dx>0 then /* не е вертикална отсечка */
  $x_e = x_L$ /* диагонална отсечка с посока отляво надясно */
    $x_l = x_R$
  else if dx < 0 then /* диагонална отсечка с посока отдясно наляво */
    $x_e = x_R$
    $x_l = x_L$
  else /* вертикална отсечка */
    if dy>0 then
      $y_e = y_B$
      $y_l = y_T$
    else
      $y_e = y_T$
      $y_l = y_B$
    end if
  end if
  if dy>0 then /* не е хоризонтална отсечка */
    $y_e = y_B$ /* диагонална отсечка с посока отдолу нагоре */
    $y_l = y_T$
  else if dy<0 then /* диагонална отсечка с посока отгоре надолу */
    $y_e = y_T$
    $y_l = y_B$
  else /* хоризонтална отсечка */
    if dx>0 then
      $x_e = x_L$
      $x_l = x_R$
    else
      $x_e = x_R$
      $x_l = x_L$
    end if
  end if
  /* определяме t за входни пресечни точки със страните на прозореца */
  /* и t за първата изходна пресечна точка */
  if $dx \neq 0$ then
    $t_{e_x} = (x_e - x_i)/dx$ /* диагонална права */
  else
    $t_{e_x} = -\infty$ /* вертикална права */
  end if

```
if dy ≠ 0
  t_ey = (y_e − y_i)/dy        /* диагонална права */
else
  t_ey = −∞        /* хоризонтална права */
end if
if t_ex < t_ey then        /* първа входна точка x после y */
  t_e1 = t_ex
  t_e2 = t_ey
else
  t_e1 = t_ey
  t_e2 = t_ex
end if
        /* анализиране на отсечката */
if 1 ≥ t_e1 then        /* условия 2 или 3 или 4 или 6 и не 1 */
  if 0 < t_e1 then        /* условие 5 - повратен връх */
    call output(x_e, y_e,Nout,Q)
  end if
  if 1 ≥ t_e2 then        /* условия 3 или 4 или 6 и не 1 или 2 */
            /* определяне t на втората изходна точка */
    if dx ≠ 0 then        /* диагонална права */
      t_lx = (x_l − x_i)/dx
    else        /* вертикална права */
      if (x_L ≤ x_i) and (x_i ≤ x_R) then        /* P_i inside */
        t_lx = ∞
      else        /* P_i outside */
        t_lx = −∞
      end if
    end if
    if dy ≠ 0 then        /* диагонална права */
      t_ly = (y_l − y_i)/dy
    else        /* хоризонтална права */
      if (y_B ≤ y_i) and (y_i ≤ y_T) then        /* P_i inside */
        t_ly = ∞
      else        /* P_i outside */
        t_ly = −∞
      end if
    end if
    if t_lx < t_ly then        /* първа изходна точка в x */
      t_l1 = t_lx
    else        /* първи изход в y */
      t_l1 = t_ly
    end if
          /* има изход и трябва да го намерим */
    if 0 < t_e2 or 0 < t_l1 then        /* условия 4 или 6 не 3 */
      if t_e2 ≤ t_l1 then        /* условия 4 - невидима отсечка */
        if 0 < t_e2 then        /* P_i е извън прозореца */
          if t_ex > t_ey then        /* вертикална граница */
            call output(x_e, y_i + t_ex ∗ dy;Nout,Q)
```

```
          else       /* хоризонтална граница */
            call output(x_i + t_{e_y} * dx, y_e;Nout,Q)
          end if
        end if
        if 1 > t_{l_1} then       /* P_{i+1} е извън прозореца */
          if t_{l_x} < t_{l_y} then       /* вертикална граница */
            call output(x_l, y_i + t_{l_x} * dy;Nout,Q)
          else       /* хоризонтална граница */
            call output(x_i + t_{l_y} * dx, y_l;Nout,Q)
          end if
        else       /* P_{i+1} е вътре в прозореца */
          call output(x_{i+1}, y_{i+1};Nout,Q)
        end if       /* край на условие 4 */
      else       /* условие 6 - повратен връх */
        if t_{e_x} > t_{e_y} then       /* втори вход в x */
          call output(x_e, y_l;Nout,Q)
        else       /* втори вход в y */
          call output(x_l, y_e;Nout,Q)
        end if
      end if       /* край на условие 6 */
    end if       /* край на условия 4 или 6 */
  end if       /* край на условия 3 или 4 или 6 */
end if       /* край на условия 2 или 3 или 4 или 6 */
next i       /* край на отсечка P_iP_{i+1} */
return
```

5.17 Отсичане на вдлъбнат полигон. Алгоритъм на Weiler-Atherton.

В този алгоритъм двата полигона (този който ще отсичаме и този който отсича) могат да бъдат вдлъбнати. Даже допуска се полигоните да имат дупки. Полигона който ще отсичаме ще го бележим с S, (Subject) а отсичащия полигон ще го бележим с C (Clip). Пресечните точки на двата полигона ще бележим с I (Intersection). Върховете на двата полигона ще обикаляме в посока, така че вътрешността да остава вдясно. Посоката е на часовата стрелка за външната граница и обратно на часовата стрелка за дупките, ако има такива. Границите на S полигона и C полигона могат да не се пресичат или да се пресичат. Ако се пресичат, то точките на пресичане се появяват по двойки. Едната пресечна точка е когато страната на S полигона навлиза в C полигона (входна точка) и другата точка е когато S полигона напуска C полигона (изходна точка). Алгоритъма стартира от входна точка. Тя се извежда в изходния полигон. След това се следва границата на S полигона по посока на часовата стрелка докато се стигне до пресечна точка с C полигона. Извеждат се всички върхове на S полигона през които се е минало. След това се следва границата на C полигона пак по посока на часовата стрелка докато се намери пресечна точка със S полигона. Извеждат се върховете на C полигона през които се е минало. И така докато се върнем в началната точка. От примерите алгоритъма става съвсем ясен.

Формално описание на алгоритъма:

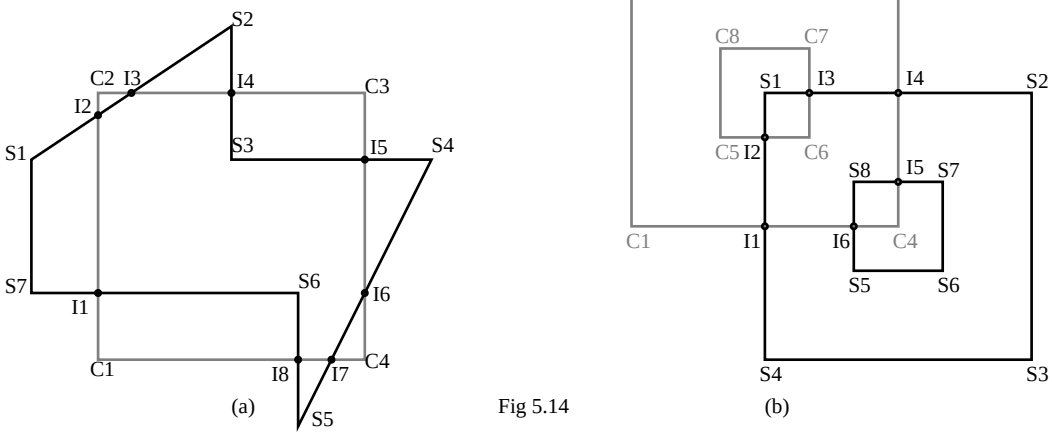


Fig 5.14

1. Определяне на пресечните точки в обектния полигон и отсичащия полигон

   Да се добавят пресечните точки към върховете на S полигона и C полигона. Да се установи съответствие между пресечните точки в двете множества на дадения полигон и отсичашия полигон, така че лесно да се намира една и съща пресечна точка от едното множество и другото множество.

2. Обработване на границите на полигона

   Създават се две работни множества от върхове на полигоните. Едното за върховете които са вътрешни за C полигона и другото за външните върхове. Да се игнорират тези върхове от отсичащия полигон които са външни за обектния полигон. Границата на отсичащия полигон, която е вътрешна за обектния полигон, оформя дупки в обектния полигон. Да се копира граница в подподящите създадени множества.

3. Създаване на 2 множества от пресечни точки

   Едното множество е за входните пресечни точки а другото за изходните пресечни точки. Входна пресечна точка и изходна пресечна точка вървят по двойки. Може да се създаде едно множество от всички последователни пресечни точки и тези с четните номера ще са от едното множество, например входното, а с нечетните номера от другото.

4. Извършване на отсичането

   (a) Намиране на вътрешен полигон (Полигон, който е вътрешен за отсичащия полигон).

      i. Намираме една пресечна точка от входното множество на предметния полигон. Ако няма такава процеса свършва.

      ii. Обикаляме отпред назад множеството от върхове на предметния полигон и пресечни точки докато стигнем до пресечна точка. Копираме множеството на обходени върхове във вътрешния полигон.

      iii. Скачаме на същата пресечна точка от множестмото на отсичашия полигон.

      iv. Обикаляме отпред назад множеството от върхове на отсичащия полигон и пресечни точки докато стигнем до пресечна точка. Копираме множеството на обходени върхове във вътрешния полигон.

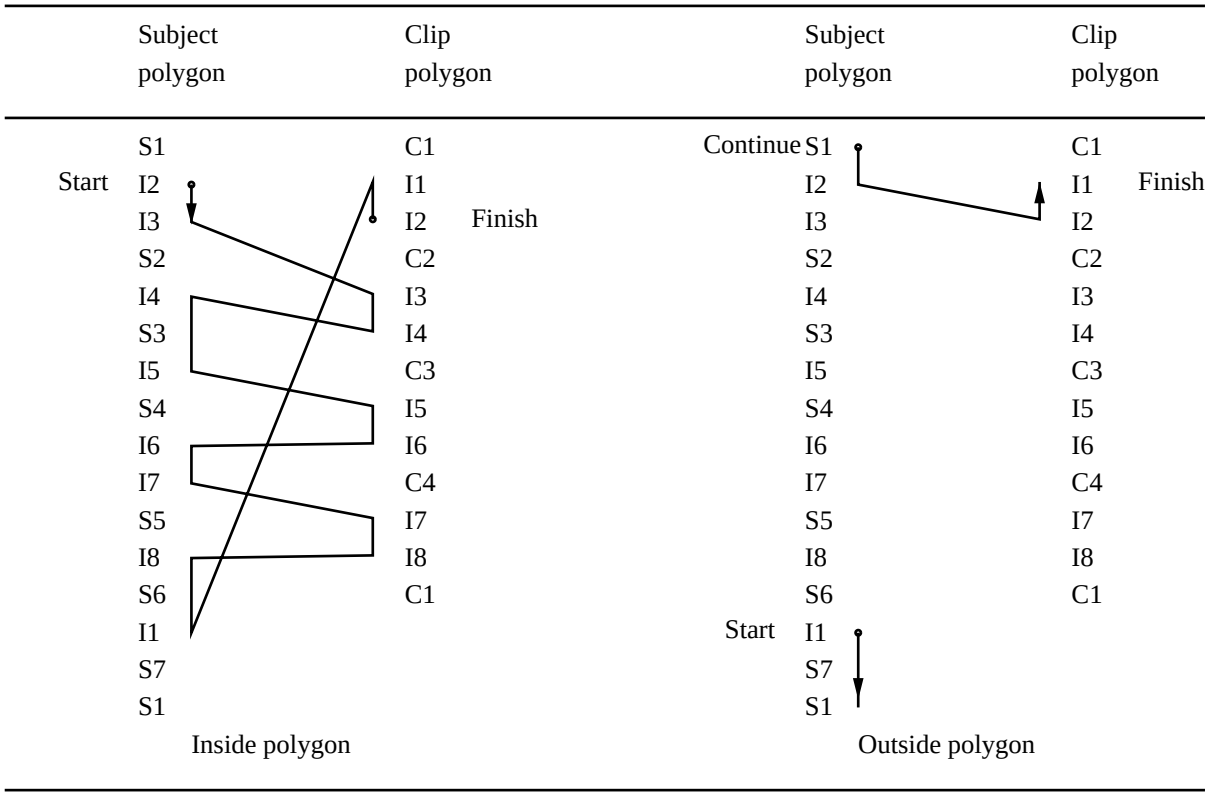
v. Скачаме обратно на същата пресечна точка от множестмото на обектния полигон.

vi. Повтаряме докато стигнем до началната точка.

(б) Намиране на външен полигон. Процедурата е същата, само че стартираме от изходна пресечна точка и множеството от върхове на отсичащия полигон обикаляме в обратна посока.

(в) Да се добавят и дупките ако има такива. Да не забравяме, че те се обикалят в обратна посока на външната граница.

Пример 1. Да разгледаме полигона на Fig. 5.14 (a). Отсичащият прозорец е квадрата чиито върхове са означени с C_i. Пресечните точки I_2, I_4, I_6, I_8 образуват входното множество а I_1, I_3, I_5, I_7 - изходното. Следвайки алгоритъма, вътрешния полигон се формира от върхо-



вете по непрекъснатата линия на ляватата страна на таблицата, започвайки от входна точка например I_2.

$$I_2 I_3 I_4 S_3 I_5 I_6 I_7 I_8 S_6 I_1 I_2$$

Ако тръгнем от друга входна точка ще получим пак същия полигон.

За да намерим външен полигон, ще стартираме от изходна точка например I_1. Следвайки стрелките вдясно на фигурата, като не забравяме, че С полигона обикаляме в обратен ред, намираме върховете

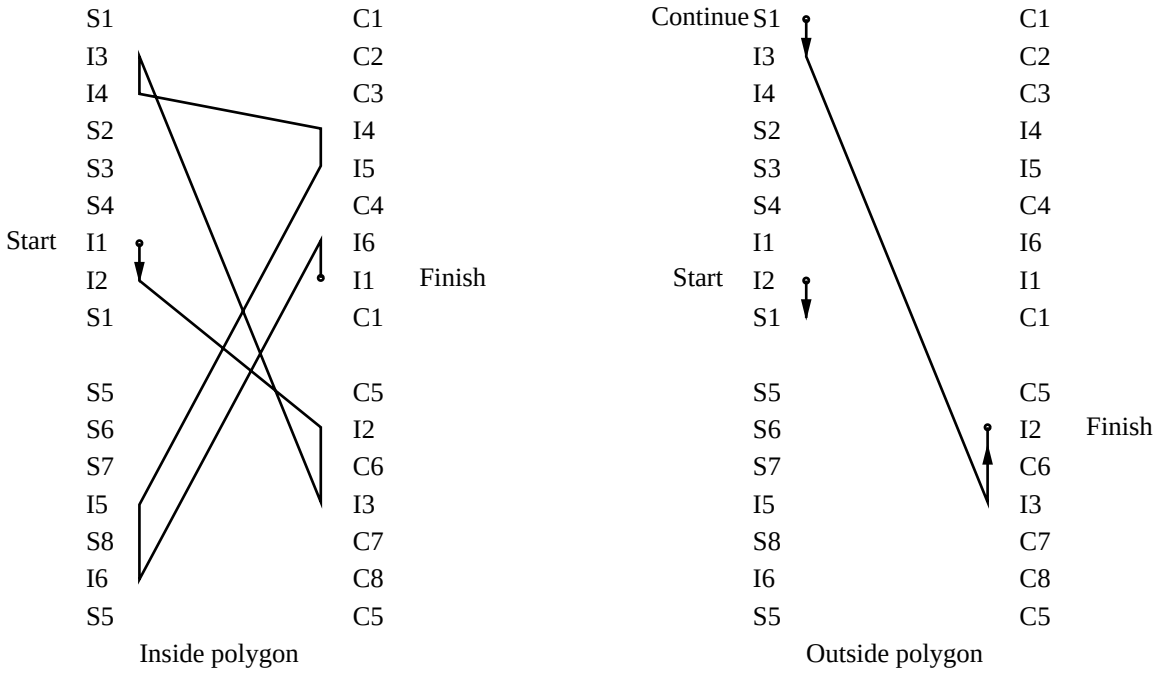
$$I_1 S_7 S_1 I_2 I_1$$

Ако стартираме от друга изходна точка, например I_5, ще намерим друг изходен полигон

$$I_5 S_4 I_6$$

Както се вижда на фигурата има 4 външни полигона.

Пример 2. Да разгледаме полигона на Fig. 5.14 (b). И двата полигона имат дупки. От стрелките на следващата таблица става ясно кои са изходните полигони.



Ако стартираме от I_1 и следваме стрелките отляво, ще получим вътрешно отсечен полигон:

$$I_1 I_2 C_6 I_3 I_4 I_5 S_8 I_6 I_1$$

Ако стартираме от I_2 и следваме стрелките отаясно, ще получим външно отсечен полигон:

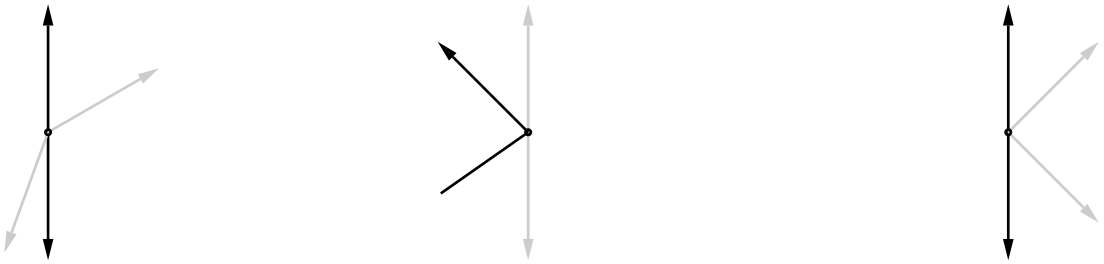
$$I_2 S_1 I_3 C_6 I_2$$

Да забележим, че обектния полигон има 2 гранични листи и всяка се обикаля индивидуално. Преминаване от листата на външната граница към тази на вътрешната става когато има скок от обектния полигон към отсичащия я обратно.

Ако стартираме от I_4 или I_6 ще получим външно отсечен полигон:

$$I_2 S_2 S_3 S_4 I_1 I_6 S_5 S_6 S_7 I_5 I_4$$

Специални случаи

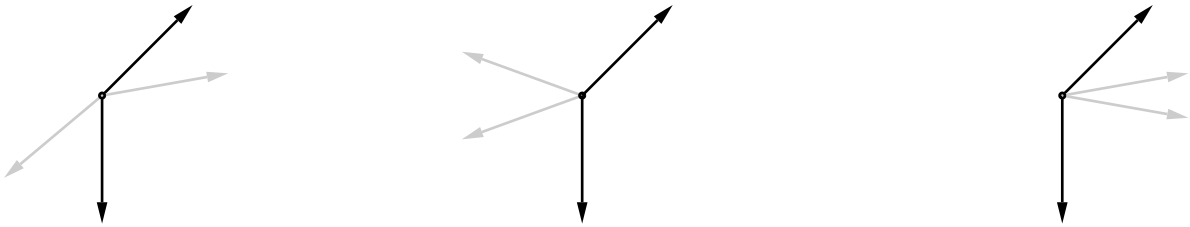


(a) (b) (c)

Fig. 5.15

За да работи алгоритъмът на Уайлър-Атертон вярно трябва да се обърне внимание на някои случаи при определяне на точките на пресичане. Пресечни точки могат да се случат в 1 връх на полигона (Фиг. 5.15) или в 2 върха (Фиг. 5.16)

Стрелките показват посоката на страните на полигона. Черните линии се отнасят за обектния полигон а червените линии за отсичащия полигон. В случаите (a) на 2-те фигури има пресечна точка, докато в другите случаи (b) и (c) няма пресичане, и такива точки не трябва да се слагат в листите на полигоните.



(a) (b) (c)

Fig. 5.16

Може да се случи част от някои страни на отсичащия и обектния полигон да съвпадат както е показано на Fig. 5.17.

В случай (a) страните се допират но няма пресичане. В случай (b) страните се допират, има 2 пресечни точки и те са отбелязани. В случаите (c) и (d) има по 1 пресечна точка, както е отбелязано.

Може да се случи обектния полигон и отсичащия полигон да нямат общи точки. Такива полигони се наричат свободни. Ето някои правила за такива случаи.

Ако обектния полигон е вътрешен за отсичащия полигон то да се добавят върховете му към вътрешната листа

Ако обектния полигон е външен за отсичащия полигон то да се добавят върховете му към външната листа

Ако обектния полигон има дупка, която е вътрешна за отсичащия полигон то да се добавят върховете и към вътрешната листа

Ако обектния полигон има дупка, която е външна за отсичащия полигон то да се добавят върховете и към външната листа

Ако отсичащия полигон е вътрешен за обектния полигон то да се добавят върховете му към вътрешната листа

Ако отсичащия полигон е външен за обектния полигон то да се добавят върховете му към външната листа в обратен ред

Ако C полигона има свободна дупка, която е вътрешна за S полигона то да се добавят върховете и към вътрешната листа

(a) (b) (c) (d)

Fig. 5.17

Глава 6

Видими повърхнини

Проблемът за визуализиране на повърхнини е един от трудните в компютърната графика. Тук трябва да се определят линии, страни, повърхнини, обеми които са видими за наблюдателя, който се намира в определена точка в пространството.

Алгоритмите могат да се разглеждата в координатната система, където са описани самите обекти. Тогава те са прецизни и са приложими за инженерни изследвания. Но алгоритмите могат да се разглеждат и в координатната система на дисплея. Тогава прецизността е тази на дисплея. Грубо казано това е точността на 1 пиксел. Обикновено броят на пикселите е 1280x1024. Теоретично работата за първия вид алгоритми е n^2, а за втория вид е nN, където n е броят на обектите на сцената, а N е броят на пикселите.

6.1 Алгоритъм на плаващия хоризонт

Този алгоритъм най-често се прилага за визуализиране на графиката на функция от вида

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

Тъй като визуализирането на графиката има обикновено принципно значение, то този алгоритъм се представя в координатната система на дисплея. Идеята на алгоритъма е да се сведе тримерния проблем до двумерен чрез пресичане на графиката със серия от успоредни равнини на една от координатните равнини. Обикновено горното уравнение се решава по отношение на една от променливите, (ако е възможно) например

$$y = f(x, z) \quad \text{или} \quad x = g(y, z)$$

където z е константа за всяка от успоредните равнини.

Горен хоризонт

Повърхнината се представя от серия криви във всяка от успоредните равнини. Предполага се, че сечението на графиката на дадената функция с равнина успоредна на Oxy е графика на еднозначна функция. Успоредните равнини са сортирани по $z : z = z_0, z_1, ..., z_{max}$, $z_0 < z_1 < ... < z_{max}$. Във всяка равнина се определя кривата $y = f(x, z_k)$. В такъв случай алгоритъма изглежда приблизително така:

Ако за дадена стойност на x стойността на y от кривата е по-голяма от стойността на y за предишната крива във същата точка x, тогава точката е видима в противен случай е невидима. Това е показано на Fig. 6.1.

Образува се масив, който да съдържа най-голямата стойност на y за всяко x. Стойностите на този масив представляват текущия хоризонт. Практически алгоритъма е едномерен.

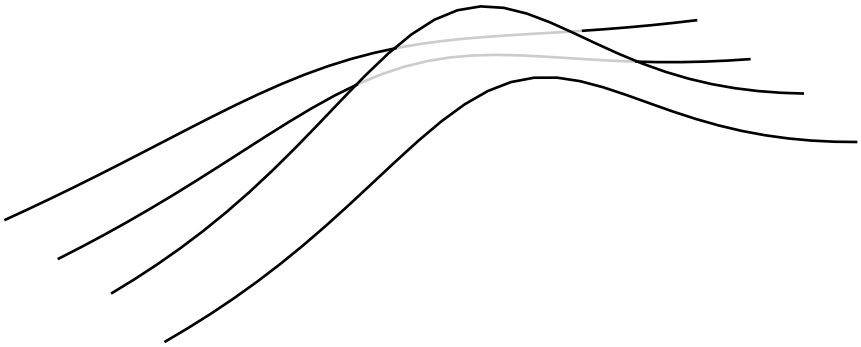


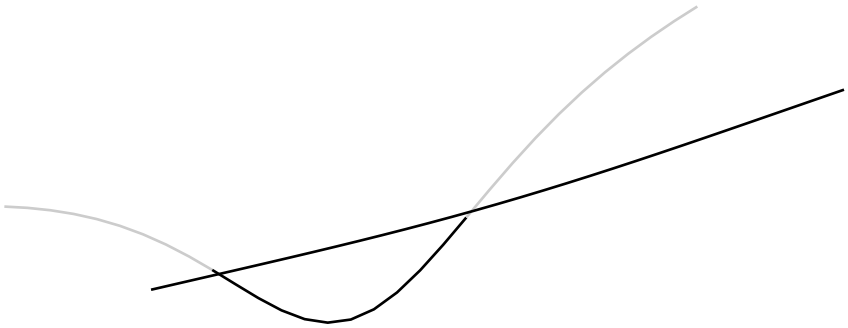
Fig. 6.01

Долен хоризонт

Аналогично на горния хоризонт, ако текущата крива някъде попадне под някоя от предните криви, то тази част на кривата също ще е видима от долу на картинката. Затова е необходимо да се въведе още един масив за долния хоризонт. тогава алгоритъма ще изглежда така:

Ако за дадена стойност на x стойността на y от кривата е по-голяма от най-голямата стойност на y от предишните криви, или е по-малка от най-малката стойност на y от предишните криви, тогава точката (x, y) е видима в противен случай е невидима.

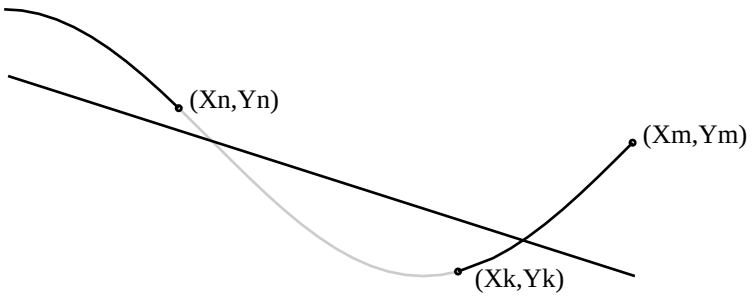
Fig. 6.02



Интерполиране

До тук алгоритъмът предполага, че стойността $y = f(x, z_k)$ може да се пресметне за всяка стойност на x. Ако обаче това не може да стане, то не могат да се попълнят масивите за горния и долния хоризонт. За да се попълнят тези масиви се използва линейна интерполация между известните данни. Но това често води до резултат, който не е достатъчно прецизен. Например на Fig. 6.02 се вижда, че точките на пресичане на двете криви не са достатъчно прецизно определени. Да предположим, че попълването става след проверката за видимост. Тогава ако текущата линия минава от видима точка (x_n, y_n) към невидима (x_k, y_k), то линията между двете точки става невидима по алгоритъма и не се чертае. Също така ако минаваме от невидима точка (x_k, y_k) към видима (x_m, y_m), то линията от (x_k, y_k) до (x_m, y_m) се чертае. Трябва да се определи пресечната точка по-акуратно.

Fig. 6.03



Един начин за това е x да се увеличава с 1 (в координатната система на дисплея) започвайки от x_n и завършвайки с x_k, а y да се увеличава с Δy. За всяка точка $(x_n+s, y_n+s\Delta y),\ s=1,2,...$ се проверява дали е видима или не и в зависимост от това тя се чертае или не.

Друг начин е да се представят параметрично отсечките с краища (x_n, y_n) и (x_k, y_k) от текущата линия и (x_n, y_n^p) и (x_k, y_k^p) от предишната линия (на фигура 6.03 - правата линия) и да се намери пресечната точка (x_i, y_i). След като направим пресмятанията се получава

$$x_i = x_n - \frac{\Delta x(y_n^p - y_n)}{\Delta y^p - \Delta y}, \quad y_i = y_n + m(x_i - x_n),$$

където

$$\Delta x = x_k - x_n, \quad \Delta y^p = y_k^p - y_n^p, \quad \Delta y = y_k - y_n, \quad m = \frac{y_k - y_n}{\Delta x}.$$

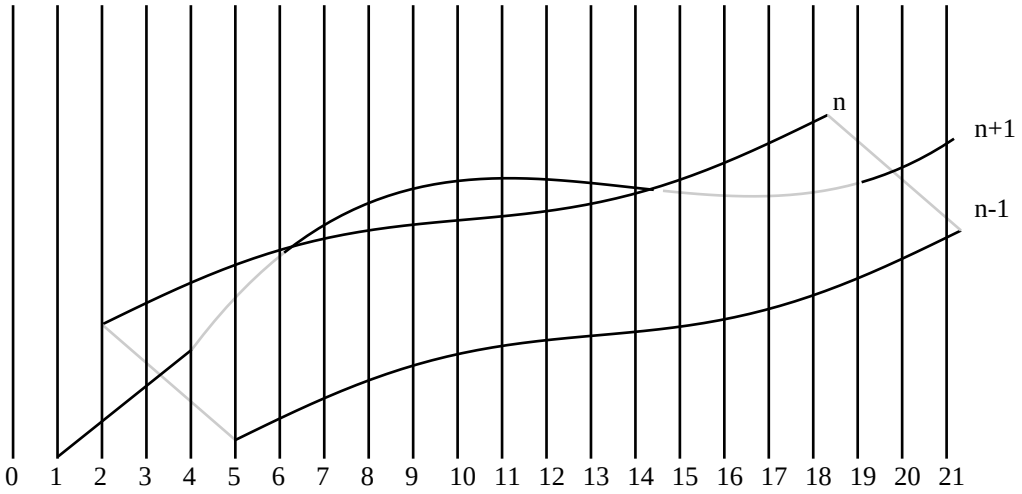
След като намерим пресечната точка (x_i, y_i), то

1) ако точката (x_n, y_n) е видима, а точката (x_k, y_k) е невидима то чертаем отсечката с краища (x_n, y_n) и (x_i, y_i),

2) ако точката (x_n, y_n) е невидима, а точката (x_k, y_k) е видима то чертаем отсечката с краища (x_i, y_i) и (x_k, y_k).

Назъбени краища

Алгоритъма има проблем, когато краищата на една от линиите излиза извън краищата на предишните линии. На фигурата е показан този ефект. Текущата линия е означена с $n+1$.



След като сме обработили $n-1$ и $n-$тата линия, горния хоризонт съдържа началните данни за $x=0,1$, данните от линия n за $x=2,3,...,18$ и данните от линия $n-1$ за $x=19,20,21$. Долния хоризонт съдържа началните данни за $x=0,1$, даннита от линя n за $x=2,3,4$ и даннита от линя $n-1$ за $x=5,6,...,21$. Когато обработваме линия $n+1$, то следвайки алгоритъма излиза че тя е видима за $x=4$. Това е показано с по-тъмната линия за x от 1 до 4. Аналогичен ефект е показан в другия край за $x=19$. Ефекта от това е че левия край на картината ще е назъбен. Решението за левия край е да се въведат стойности от светлата линия между позиции 2 и 5 на x за горния и долния хоризонт. Трябва да направим следното за:

Ляв край

Ако P_n е първата точка на първата линия, то тя да се запази като P_{n-1} и продължаваме;

Иначе: Ако P_n е първата точка на линия след първата, създаваме линия от P_{n-1} до P_n.

Обработваме двата хоризонта и запазваме P_n като P_{n-1}.

Десен край

Ако P_n е последната точка на първата линия, то тя да се запази като P_{n-1} и продължаваме;

Иначе: Ако P_n е последната точка на линия след първата, създаваме линия от P_{n-1} до P_n.

Обработваме двата хоризонта и запазваме P_n като P_{n-1}.

Aliasing

Ако функцията съдържа много стръмни и близки страни (подобно на комини) то алгоритъма дава грешни резултати (може да пропусне комините). Едно от решенията за такива случаи е да се намали стъпката по x (да се увеличи точността).

Cross-hatching

Под "Cross-hatching" се разбира чертаене както на линиите $y = f(x, z),\ z$ константа, така и на линиите $y = f(x, z),\ x$ константа. Така се получава по-пълна представа за графиката на функцията. На пръв поглед изглежда, че това е като да се направи суперпозиция на два алгоритъма - единия когато z е константа и другия когато x е константа. Но не е точно така. На практика след като се обработи кривата $y = f(x, z),\ z = z_1$, се обработват кривите $y = f(x, z),\ x$ константа, между z_1 и z_2 и след това се обработва кривата $y = f(x, z),\ z = z_2$. Когато се прави cross-hatching назъбване на краищата не се прави (не е необходимо).

Псевдокод на алгоритъма (floating horizon algorithm)

subroutine fhalg(Xmin,Xmax,Ymin,Ymax,Xsamples,Zsamples)

Hscr - брой пиксели по X направление
Vscr - брой пиксели по Y направление
Upper(1:Hscr) - масив за горния хоризонт
Lower(1:Hscr) - масив за долния хоризонт
y е текущата стойност на функцията $y = f(x, z)$ при фиксирано z
Cflag - флаг за видимост на текущата точка
Pflag - флаг за видимост на предишната точка
0 - невидима точка
1 - видима точка над горния хоризонт
-1 - видима точка под долния хоризонт
Draw - чертае видима линия между зададените точки
Xmin,Xmax - минимална и максимална x координата
Zmin,Zmax - минимална и максимална z координата
Zinc - стъпка по z между отделните сечения

```
Инициализиране на променливите:
Zinc=(Zmax-Zmin)/Zsamples
Upper = 0
Lower = Vscr
Стартираме с най-близката равнина
for z=Zmax to Zmin step -Zinc
    инициилизиране на предишните x и y : Xprev, Yprev
    Xprev=Xmin
    Yprev=f(Xprev,z)
    call Transform(Xprev,Yprev,z) /* viewing transformation */
    call Visibility(Xprev,Yprev,Upper,Lower;Pflag)
    /* за всяка точка от кривата y = f(x,z) */
    for xr=Xmin to Xmax step (Xmax-Xmin)/Xsamples
        x=xr
        y=f(xr,z)
```

```
            call Transform(x,y,z)
            call Visibility(x,y,Upper,Lower;Cflag) /* тест за видимост */
            /* чертаене на кривата и попълване на хоризонтите */
            call Drawline(Xprev,Yprev,Pflag,x,y,Cflag;Upper,Lower)
        next xr
        /* чертаене на ортогоналните траектории */
        /* следващите оператори може да се пропуснат както е на Fig. 6.05 */
        if z-Zinc >= Zmin then
            for xr=Xmin to Xmax step (Xmax-Xmin)/Xsamples
                Xprev=xr
                Yprev=f(Xprev,z)
                call Transform(Xprev,Yprev,z)
                call Visibility(Xprev,Yprev,Upper,Lower;Pflag) /* тест за видимост */
                for hz= z to z-Zinc step (Zmin-Zmax)/Xsamples
                    x=xr
                    y=f(x,hz)
                    call Transform(x,y,hz)
                    call Visibility(x,y,Upper,Lower;Cflag) /* тест за видимост */
                    call Drawline(Xprev,Yprev,Pflag,x,y,Cflag;Upper,Lower)
                next hz
            next xr
        end if
    next z
end sub
```

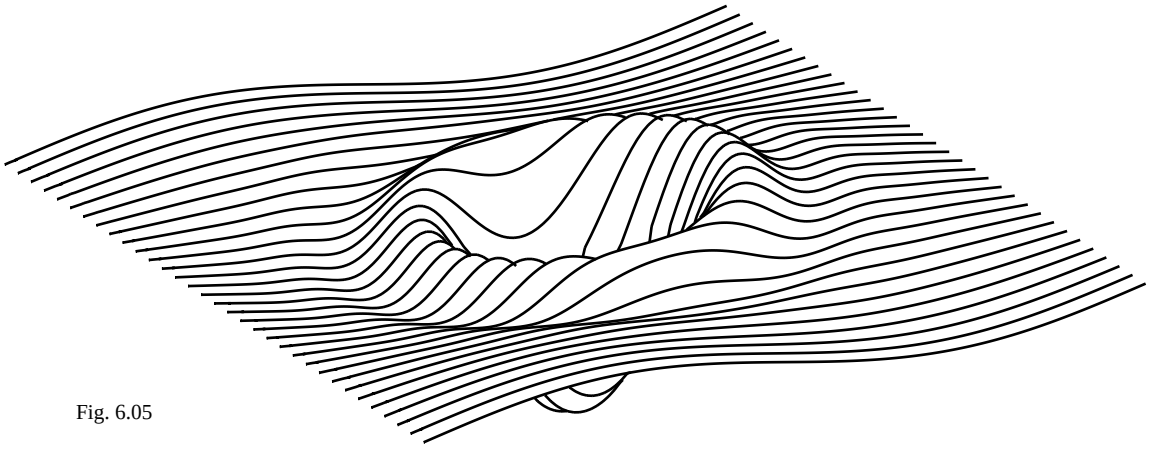


Fig. 6.05

На фигурата е начертана графиката на функцията

$$y = \frac{1}{5}\sin(x)\cos(z) - \frac{3}{2}\cos\frac{7a}{4}e^{-a}, \quad a = (x-\pi)^2 + (z-\pi)^2, \quad x, z \in [0, 2\pi]$$

по описания алгоритъм, без ортогоналните траектории. Точките (x, y) са мащабирани с коефициент 50 и са транслирани с вектор (50,125). Кординатата z също е умножена с 50.

Функцията Transform(x,y,z) първо завърта точката (x,y,z) около остта X на ъгъл $\frac{\pi}{8}$, после около остта Y на ъгъл $\frac{\pi}{6}$ и накрая проектира точката върху равнината $z = 0$ с център на

проекция ∞. Матрицата на трансформация е

$$T = \begin{pmatrix} 0.8660 & 0.1913 & 0 & 0 \\ 0 & 0.9239 & 0 & 0 \\ 0.5 & -0.3314 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Другите процедури са:

```
subroutine Drawline(Xprev,Yprev,Pflag,x,y,Cflag;Upper,Lower)
    /* попълва хоризонтите и чертае линията */
    if (0<=x) and (x<=Hscr) and (0<=y) and (y<=Vscr) and
       (0<=Xprev) and (Xprev<=Hscr) and (0<=Yprev) and (Yprev<=Vscr) then
       if Cflag==Pflag then
          if (Cflag==1) or (Cflag == -1) then
             call Draw(Xprev,Yprev,x,y)
             call Horizon(Xprev,Yprev,x,y;Upper,Lower)
          end if
       else
          if (Cflag==0) then
             if (Pflag==1) then
                call Intersect(Xprev,Yprev,x,y,Upper;Xi,Yi)
             else
                call Intersect(Xprev,Yprev,x,y,Lower;Xi,Yi)
             end if
                call Draw(Xprev,Yprev,Xi,Yi)
                call Horizon(Xprev,Yprev,Xi,Yi;Upper,Lower)
          else
             if (Cflag==1) then
                if (Pflag==0) then
                  call Intersect(Xprev,Yprev,x,y,Upper;Xi,Yi)
                  call Draw(Xi,Yi,x,y)
                  call Horizon(Xi,Yi,x,y;Upper,Lower)
                else
                  call Intersect(Xprev,Yprev,x,y,Lower;Xi,Yi)
                  call Draw(Xprev,Yprev,Xi,Yi)
                  call Horizon(Xprev,Yprev,Xi,Yi;Upper,Lower)
                  call Intersect(Xprev,Yprev,x,y,Upper;Xi,Yi)
                  call Draw(Xi,Yi,x,y)
                  call Horizon(Xi,Yi,x,y;Upper,Lower)
                end if
             else
                if (Pflag==0) then
                  call Intersect(Xprev,Yprev,x,y,Lower;Xi,Yi)
                  call Draw(Xi,Yi,x,y)
                  call Horizon(Xi,Yi,x,y;Upper,Lower)
                else
                  call Intersect(Xprev,Yprev,x,y,Upper;Xi,Yi)
                  call Draw(Xprev,Yprev,Xi,Yi)
```

```
                        call Horizon(Xprev,Yprev,Xi,Yi;Upper,Lower)
                        call Intersect(Xprev,Yprev,x,y,Lower;Xi,Yi)
                        call Draw(Xi,Yi,x,y)
                        call Horizon(Xi,Yi,x,y;Upper,Lower)
                    end if
                end if
            end if
        end if
    end if
end sub

subroutine Visibility(x,y,Upper,Lower;Cflag)
    /* определя видимост на точка спрямо хоризонтите */
    /* ако точката лежи на хоризонтите тя се определя като видима */
    /* предполага се че x е цяла променлива */
    /* Cflag = 0 - невидима */
    /* Cflag = 1 - видима над горния хоризонт */
    /* Cflag = -1 - видима под долния хоризонт */
    if (0<=x) and (x<Hscr) and (0<=y) and (y<Vscr) then
        if (y<Upper(x)) and (y>Lower(x)) then Cflag=0
        if (y>=Upper(x)) then Cflag=1
        if (y<=Lower(x)) then Cflag=-1
    else
        Cflag=0
    end if
end sub

subroutine Horizon(x1,y1,x2,y2;Upper,Lower)
    /* използва се линейна интерполация */
    if (x1==x2) then
        Upper(x2)=max(Upper(x2),y2)
        Lower(x2)=min(Lower(x2),y2)
    else
        slope=(y2-y1)/(x2-x1)
        for x=x1 to x2 step sign(x2-x1)
            y=y1+slope*(x-x1)
            Upper(x) = max(Upper(x),y)
            Lower(x) = min(Lower(x),y)
        next x
    end if
end sub

subroutine Intersect(x1,y1,x2,y2,Array;Xi,Yi)
    /* намира пресечната точка на двете отсечки */
    /* предполага се че x1 и x2 са цели */
    /* Inc - използва се за посоката на правата */
    Inc=-1
    if (x1<x2) then Inc=1
    if (x1=x2) then
```

```
        Xi=x2;
        Yi=Array(x2)
    else
        slope = (y2-y1)/(x2-x1)
        Xi = x1 + Inc
        Yi = y1 + Slope*Inc
        Ysign = sign(Yi - Array(Xi))
        Csign=Ysign
        while (Csign = Ysign) and (Xi*Inc < x2*Inc)
            Xi = Xi + Inc
            Yi = Yi + Slope*Inc
            Csign = sign(Yi - Array(Xi))
        end while
        if abs(Yi - slope*Inc - Array(Xi-Inc)) < abs(Yi - Array(Xi)) then
            Xi = Xi - Inc
            Yi = Yi - slope*Inc
        end if
        if (abs(slope) > 1) and (Array(Xi) <> Array(0)) then
            Yi = Array(Xi)
        end if
    end if
end sub
```

Глава 7

Криви и повърхнини представени чрез криви

Криви в равнината и повърхнини представяни чрез криви се задават с уравнения. Това е предимство за по-бърза обработка на такива обекти в компютърната графика. Тези представяния имат по компактна форма отколкото представяне чрез полигони. Обектите описани с криви са по-гладки, имат възможност да се обработват по лесно и по бързо, а също пести се памят когато обектите са зададени с функция.

7.1 Параметрично зададени криви

Точките от кривата се задават с формула по която за всяко $t \in [a, b]$ се пресмята $P = P(t)$ (пресмятат се координатите на точката P: x(t), y(t), (z(t) ако точката е в тримерното пространство)). За функцията $P(t)$ се иска да е непрекъсната и тогава казваме, че $P(t) \in C_0$. В много от случаите се иска тази функция да е гладка (да има непрекъсната тангента) и тогава $P(t) \in C_1$. Може да се иска да имаме по-висока гладкост $P(t) \in C_k,\ k > 1$.

7.1.1 Криви на Безие

Нека са дадени 2 точки в равнината P_0 и P_1. При линейна интерполация на тези две точки получаваме точките по правата между тях:

$$P^1(t) = P_0 + t(P_1 - P_0) = (1-t)P_0 + tP_1, \quad t \in [0, 1].$$

Това е най-простата крива на Безие - от 1-ви ред. Нека сега да имаме 3 точки P_0, P_1 и P_2. За фиксирано $t \in [0, 1]$ първо да интерполираме между P_0 и P_1 - ще имаме точка $P_0^1(t)$, после да интерполираме между P_1 и P_2 - ще имаме точка $P_1^1(t)$ и накрая да интерполираме между $P_0^1(t)$ и $P_1^1(t)$ за да получим точка $P^2(t)$. Ще имаме

$$P_0^1(t) = (1-t)P_0 + tP_1;\ P_1^1(t) = (1-t)P_1 + tP_2;$$

$$P^2(t) = (1-t)P_0^1(t) + tP_1^1(t) = (1-t)^2P_0 + 2(1-t)tP_1 + t^2P_2,\ t \in [0, 1]. \tag{7.1}$$

Така получената крива се нарича крива на Безие от втори ред. Ако продължим по същия начин ще стигнем до крива на Безие от ред n :

$$P^n(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k P_k,\ t \in [0, 1]. \tag{7.2}$$

Глава 8

Приложение

8.1 Приложение А

Ако $P = (P_x, P_y, P_z)$, $Q = (Q_x, Q_y, Q_z)$ и $R = (R_x, R_y, R_z)$ то

$$(P \cdot Q)R = (P_xQ_x + P_yQ_Y + P_zQ_z)(R_x, R_y, R_z) =$$
$$((P_xQ_x + P_yQ_y + P_zQ_z)R_x, (P_xQ_x + P_yQ_Y + P_zQ_z)R_y, (P_xQ_x + P_yQ_Y + P_zQ_z)R_z) =$$
$$(Q_x, Q_y, Q_z)\begin{pmatrix} P_xR_x & P_xR_y & P_xR_z \\ P_yR_x & P_xR_y & P_xR_z \\ P_zR_x & P_xR_y & P_xR_z \end{pmatrix} = (P_x, P_y, P_z)\begin{pmatrix} Q_xR_x & Q_xR_y & Q_xR_z \\ Q_yR_x & Q_xR_y & Q_xR_z \\ Q_zR_x & Q_xR_y & Q_xR_z \end{pmatrix}$$

От тук следва

$$(P \cdot Q)R = Q(P^TR) = P(Q^TR), \tag{8.1}$$

където P^TR и Q^TR са горните матрици.

Векторното произведение също може да се запише като произведение на матрици:

$$P \times Q = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ P_x & P_y & P_z \\ Q_x & Q_y & Q_z \end{vmatrix} = (P_yQ_z - P_zQ_y, -P_xQ_z + P_zQ_x, P_xQ_y - P_yQ_x)$$

т.е.

$$P \times Q = (Q_x, Q_y, Q_z)\begin{pmatrix} 0 & P_z & -P_y \\ -P_z & 0 & P_x \\ P_y & -P_x & 0 \end{pmatrix} = (P_x, P_y, P_z)\begin{pmatrix} 0 & -Q_z & Q_y \\ Q_z & 0 & -Q_x \\ -Q_y & Q_x & 0 \end{pmatrix} \tag{8.2}$$

8.2 Приложение Б. Кватерниони

Кватернион $\mathbf{q}$ може да се представи като матрица (2 x 2) от комплексни числа:

$$q = \begin{pmatrix} z & w \\ -\bar{w} & \bar{z} \end{pmatrix} = \begin{pmatrix} a+ib & c+id \\ -c+id & a-ib \end{pmatrix}, \tag{8.3}$$

където $z = a + ib,\ w = c + id$ са комплексни числа и a,b,c,d са реални числа. По аналогия с комплексните числа (z=a1+bi), кватернионът може да се запише и като:

$$q = aU + bI + cJ + dK, \tag{8.4}$$

където матриците U, I, J, K са:

$$U = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ I = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix},\ J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},\ K = \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}. \tag{8.5}$$

Да обърнем внимание, че единичната матрица при тези означения, е U а не I. Лесно се проверяват равенствата

$$I^2 = -U,\ j^2 = -U,\ k^2 = -U,$$

откъдето се вижда, че матриците $I,\ J,\ K$ са решение на матричното уравнение $X^2 = -U$. Можем да кажем, че матриците $I,\ J,\ K$ са корен квадратен от минус единичната матрица. (При комплексните числа i е корен квадратен от -1.)

На кватернионите може да се гледа и като елементи на 4-мерно векторно пространство с базис:

$$i = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix}, j = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, k = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, 1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

За кватернионите са изпълнени следните равенства

$$i^2 = j^2 = k^2 = -1,\ ij = -ji = k,\ jk = -kj = i,\ ki = -ik = j.$$

Осемте кватерниона: $\pm 1,\ \pm i,\ \pm j,\ \pm k$ образуват група по отношение на операцията умножение.

Кватернионите могат да се интерпретират като наредена двойка от число и тримерен вектор : $\mathbf{q} = [w, \mathbf{v}]$, където w е число и $\mathbf{v}$=(x,y,z) е тримерен вектор. Може да използваме и записа $\mathbf{q}$ = w + x**i** + y**j** +z**k**.

Кватернионите **i**, **j**, **k** се наричат още имагинерни единици. По аналогия с комплексните числа, числото w се нарича реална част на кватерниона, а векторът **v** се нарича имагинерна част. За имагинерната част можем да използваме обикновените операции с вектори - събиране, умножение с число, скаларно умножение, векторно умножение и др.

Спрегнатия кватернион се дава с $\mathbf{q}^*$ = w - x**i** - y**j** - z**k**.

Сума и **разлика** на кватерниони е очевидна:

$\mathbf{q_1} \pm \mathbf{q_2} = (w_1 \pm w_2) + (x_1 \pm x_2)\mathbf{i} + (y_1 \pm y_2)\mathbf{j} + (z_1 \pm z_2)\mathbf{k}$.

Произведението обаче не е очевидно:

$\mathbf{q_1}.\mathbf{q_2} = (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) + (w_1x_2 + w_2x_1 + y_1z_2 - z_1y_2)\ \mathbf{i} +$
$(w_1y_2 + w_2y_1 + z_1x_2 - x_1z_2)\ \mathbf{j} + (w_1z_2 + w_2z_1 + x_1y_2 - y_1x_2)\ \mathbf{k} =$
$[(s_1s_2 - \mathbf{v_1} \cdot \mathbf{v_2})\ ,\ s_1\ \mathbf{v_2} + s_2\ \mathbf{v_1} + \mathbf{v_1} \times \mathbf{v_2}]$.

Тук в реалната част участва скаларното произведение на $\mathbf{v_1}$ и $\mathbf{v_2}$, а в имагинерната част участва векторното произведение на тези вектори.

Произведението е асоциативно но не е комутативно. Но произведението на кватернион с число е комутативно.

Нормата се дефинира като:

$|\mathbf{q}| = |[\mathrm{w},\mathbf{v}]| = \sqrt{\mathbf{q}\mathbf{q}^*} = \sqrt{\mathbf{q}^*\mathbf{q}} = \sqrt{w^2 + \mathbf{v}.\mathbf{v}} = \sqrt{w^2 + x^2 + y^2 + z^2}$.

Лесно се проверява, че нормата е мултипликативна: $|\mathbf{q_1}\mathbf{q_2}| = |\mathbf{q_1}|.|\mathbf{q_2}|$

Единичен кватернион е онзи за който нормата му е 1 : $[1, \mathbf{0}]$.

Реципрочният кватернион се дава с

$$\mathbf{q^{-1}} = \frac{\mathbf{q}^*}{\mathbf{q}\mathbf{q}^*} = \frac{\mathbf{q}^*}{|\mathbf{q}|^\mathbf{2}}$$

Някои правила при операции с кватерниони. Нека $\mathbf{p}$, $\mathbf{q}$ и $\mathbf{r}$ са кватерниони а s и t са числа. Следните операции са лесно проверими:

$$(\mathbf{q}^*)^* = \mathbf{q}$$

$$(\mathbf{q}+\mathbf{r})^* = \mathbf{q}^* + \mathbf{r}^*$$

$$(\mathbf{qr})^* = \mathbf{r}^*\mathbf{q}^*$$

$$|(\mathbf{q})^*| = |\mathbf{q}|$$

$$|(\mathbf{q})\mathbf{r}| = |\mathbf{q}||\mathbf{r}|$$

$$\mathbf{p}(\mathrm{s}\mathbf{q} + \mathrm{t}\mathbf{r}) = \mathrm{s}\mathbf{pq} + \mathrm{t}\mathbf{pr}$$

$$\mathbf{p}(\mathbf{qr}) = (\mathbf{pq})\mathbf{r}$$

Нека $\mathbf{q}$ = [w,$\mathbf{v}$] е единичен кватернион, т.е. $w^2 + \mathbf{v}.\mathbf{v} = 1$. Нека векторът $\mathbf{v}$= s $\mathbf{u}$, където $\mathbf{u}$ е единичен вектор, т.е. $\mathbf{u}.\mathbf{u}$=1. Тогава имаме $w^2 + s^2 = 1$ и следователно съществува ъгъл ϕ, такъв че $w = \cos\phi,\ s = \sin\phi$. Тогава единичният кватернион $\mathbf{q}$ може да се запише по следния начин

$$\mathbf{q} = [\cos\phi, \sin\phi\mathbf{u}].$$

В равнината единичен вектор може дда се запише като $\cos\phi + i\sin\phi = e^{i\phi}$. По аналогия единичен кватернион ще записваме като

$$\mathbf{q} = [\cos\phi, \sin\phi\mathbf{u}] = \cos\phi + \sin\phi\mathbf{u} = \mathbf{e}^{\phi\mathbf{u}}.$$

Можем да логаритмуваме и степенуваме единични кватениони по формулите

$$\log\mathbf{q} = \log\mathbf{e}^{\phi\mathbf{u}} = \phi\mathbf{u},$$

$$\mathbf{q}^{\mathrm{t}} = \mathbf{e}^{\phi\mathrm{t}\mathbf{u}} = \cos(\phi\mathrm{t}) + \sin(\phi\mathrm{t})\mathbf{u}.$$