GDV Algorithmen und Beispielprogramme

Helge Janicke

5. Januar 2002

Inhaltsverzeichnis

1 Bresenham Linie

Betrachtet werden Geraden, die durch den Koordinaten-Ursprung laufen und eine Steigung kleiner als 1 besitzen.
Definition der Geradengleichung:

$$Y = \frac{\overbrace{y_2 - y_1}^{\delta y}}{\underbrace{x_2 - x_1}_{\delta x}} x \tag{1}$$

Bresenham berechnet nicht den Funktionswert Y für ein bestimmtes x, sondern bestimmt die Fortschreitung in y-Richtung für eine Fortschreitung in x-Richtung (flacher Fall, steigend).
Die Bedingung für das Fortschreiten ist, daß der nächste zu zeichnende y-Wert y dem echten y-Wert Y näher kommt als der aktuelle. Diese Bedingung lautet:

$$\begin{aligned} y + 0.5 &< Y & (2)\\ 2y + 1 &< 2Y & (3)\\ 2y + 1 - 2Y &< 0 & (4) \end{aligned}$$

Die Ungleichung wurde so erweitert, das mit ganzen Zahlen gerechnet werden kann. Dies macht sich in der Ausführungsgeschwindigkeit auf dem Computer bemerkbar.
Wir setzen nun Gleichung 1 in das Ergebnis der Umformung ein:

$$\begin{aligned} 2y + 1 - 2\frac{\delta y}{\delta x} x &< 0 & (5)\\ \underbrace{2y\,\delta x + \delta x - 2\delta y\, x}_{h} &< 0 & (6) \end{aligned}$$

Hiermit lassen sich nun die Anfangsbedingungen für h ablesen. Es gilt:

$$\begin{aligned} (x|y) = (0|0) &\implies h = \delta x & (7)\\ x \to x + 1 &\implies h \to h - 2\delta y & (8)\\ y \to y + 1 &\implies h \to h + 2\delta x & (9) \end{aligned}$$

Daraus folgt, daß wenn entlang der x-Achse durchlaufen wird für den flachen, steigenden Fall bei jeder Fortschreitung des x-Wertes um 1 der Wert von h um $2\delta y$ dekrementiert. Sinkt h jedoch unter 0, so wird der y-Wert erhöht und der Wert von h um $2\delta x$ inkrementiert. Die 8 verschiedenen Fälle lassen sich analog behandeln. Eine elegante Art ist im Algorithmus mit den Beträgen zu arbeiten und die Fortschreitung (x und y) über eine Variable zu steuern. Der nachfolgende Quellcode arbeitet nach diesem Muster.

```
c------------------------------------------------------------- gd_line
c
c  gd_line(x1,y1, x2,y2, lt)
c
c  start point          (x1 | y1)
c  end point            (x2 | y2)
c  line type            lt         NOT IMPLEMENTED
c---------------------------------------------------------------------

      subroutine gd_line(x1,y1, x2,y2, lt)

      integer x1,y1, x2,y2, lt

      integer dx, dy, ax, ay, xx, yy, h, i, imax

c dummy fuer lt
      if (lt .lt. 0) write (*,*)

      xx = x1
      yy = y1

      dx = x2 - x1
      dy = y2 - y1

      ax = abs(dx)
      ay = abs(dy)

      if (dy .gt. 0) then
                         dy = 1
                     else
                         dy = -1
                     endif

      if (dx .gt. 0) then
                         dx = 1
                     else
                         dx = -1
                     endif

      call gd_set (xx, yy)

      if (ax .ge. ay) then
                           imax = ax
                           h    = ax
                           ax   = ax + ax
                           ay   = ay + ay

                           do i=1, imax
                                   h  = h - ay
                                   xx = xx + dx
                                   if (h .lt. 0) then
                                                      yy = yy + dy
                                                      h  = h + ax
                                                  endif
                                   call gd_set(xx, yy)
                           enddo
                       else
                           imay = ay
                           h    = ay
                           ay   = ay + ay
                           ax   = ax + ax

                           do i=1, imay
                                   h  = h - ax
                                   yy = yy + dy
                                   if (h .lt. 0) then
                                                      xx = xx + dx
                                                      h  = h + ay
                                                  endif
                                   call gd_set(xx, yy)
                           enddo
                       endif
      end
```

2 Bresenham Kreis

Analog zur Linie läßt sich der Bresenham Algorithmus auch für den Kreis betrachten. Hierbei unterteilt man den Vollkreis in 8 Segmente. Bei der theoretischen Betrachtung beschränken wir uns auf den zweiten Sektor des ersten Quadranten (45° - 90°). Wir betrachten analog zur Linie die Fortschreitung. Wir beginnen beim Zeichnen des Kreises mit dem Punkt P(xm,ym+r). Wobei P(xm, ym) den Mittelpunkt, und r den Radius des Kreises darstellt. Für die Berechnung eines Kreises gilt:

$$Y^2 = r^2 - x^2 \tag{10}$$

Die Fortschreitungsbedingung läßt sich für den Bresenhamalgorithmus analog zur Linie betrachten. Die Frage ist, ab wann (für ein fortschreitendes x) der y-Wert erniedrigt werden muß. Die Bedingung lautet:

$$y - 0.5 > Y \tag{11}$$
$$2y - 1 > 2Y \tag{12}$$

Da wir den Fall für den ersten Quadranten und den Kreismittelpunkt (0|0) betrachten ist y, x und Y immer größer 0. Wir dürfen also die Gleichung quadrieren, um besser die Kreisgleichung (1) einsetzen zu können.

$$4y^2 - 4y + 1 > 4Y^2 \tag{13}$$

Durch Einsetzen der ersten Gleichung erhält man:

$$4y^2 - 4y + 1 > 4(r^2 - x^2) \tag{14}$$

Um eine einfachere Betrachtung durchführen zu können vernachlässigen wir die 1 als Summand. Die 1 wirkt sich als Summand nicht für die Beschreibung der Fortschreitung aus. Der Vorteil liegt nun darin die Gleichung durch -4 dividieren zu können.

$$y^2 - y > r^2 - x^2 \tag{15}$$
$$\underbrace{-y^2 + y + r^2 - x^2}_{h} < 0 \tag{16}$$

Sinkt h nun unter 0, so muß auch der y-Wert um 1 erniedrigt werden. Betrachten wir nun wieder die Fortschreitungsregeln:

$$(x|y) = (0|r) \implies h = \delta r \tag{17}$$
$$x \to x + 1 \implies h \to -2x - 1 \tag{18}$$
$$y \to y - 1 \implies h \to h + 2y - 2 \tag{19}$$

Mit dieser Fortschreitungsregel kann man den Kreis-Sektor beschreiben. Es ist günstig alle 8 Sektoren gleichzeitig zu zeichnen, da der Kreis punktsymetrisch zu seinem Mittelpunkt ist. Ein Problem liegt darin keine Punkte doppelt zu zeichnen. Hier ist mit Vorsicht zu testen, ob dies auch gelingt (Testausgabe von $(x|y)$ Paaren). Die folgende Seite zeigt eine Implementation des Algorithmus.

```
c------------------------------------------------------------- gd_circle
c gd_circle(xm,ym, r)
c
c Kreismittelpunkt  (xm | ym)
c Radius            r
c
c--------------------------------------------------------------------

      subroutine gd_circle(xm,ym, r)

      integer xm,ym, r

      integer ix, iy, h

      ix = 0
      iy = r
      h  = r

      call gd_set(xm    , ym + r)
      call gd_set(xm - r, ym    )
      call gd_set(xm    , ym - r)
      call gd_set(xm + r, ym    )

 1    if (iy .le. ix) GOTO 9

          h  = h - ix
          ix = ix + 1
          h  = h - ix

          if (h .le. 0) then
                            iy = iy - 1
                            h  = h + iy + iy
                        endif

          if (ix .gt. iy) GOTO 9
          call gd_set(xm + ix, ym + iy)
          call gd_set(xm - iy, ym + ix)
          call gd_set(xm - ix, ym - iy)
          call gd_set(xm + iy, ym - ix)

          if (ix .eq. iy) GOTO 9
          call gd_set(xm - ix, ym + iy)
          call gd_set(xm - iy, ym - ix)
          call gd_set(xm + ix, ym - iy)
          call gd_set(xm + iy, ym + ix)

      GOTO 1
 9    continue
      end
```

3 Gram-Schmidt-Verfahren zur Orthonormierung von Vektoren

Gegeben sei eine Basis im Vektorraum V durch $\{\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n}\}$ Gesucht ist eine orthonormierte Basis $\{\vec{u_1}, \vec{u_2}, \ldots, \vec{u_n}\}$. Das Gram-Schmidt-Verfahren stellt einen Algorithmus dar, der genau diese Aufgabe löst. Man gehe folgendermaßen vor:

$$\vec{u_1} = \frac{\vec{v_1}}{|\vec{v_1}|} \tag{20}$$

$$\vec{w_2} = \vec{v_2} - (\vec{v_2} * \vec{u_1})\,\vec{u_1} \tag{21}$$

$$\vec{u_2} = \frac{\vec{w_2}}{|\vec{w_2}|} \tag{22}$$

$$\vec{w_3} = \vec{v_3} - (\vec{v_3} * \vec{u_1})\,\vec{u_1} - (\vec{v_3} * \vec{u_2})\,\vec{u_2} \tag{23}$$

$$\vec{u_3} = \frac{\vec{w_3}}{|\vec{w_3}|} \tag{24}$$

$$\ldots \text{ bis } \vec{u_n} \tag{25}$$

Gilt für einen orthogonalisierten Vektor ($|\vec{w_i}| = 0$) so muß der Algorithmus abgebrochen werden. Desweiteren empfiehlt es sich gegenzurechnen, wie groß der Fehler bei der Orthonormierung ist und diesen Wert als Fehlercode zurückzuliefern. Dazu wird nach der Orthonormierung des Vektors das Skalarprodukt zu allen bereits orthonormierten Vektoren gebildet. Das Ergebnis stellt den Cosinus des Winkels dar. Der Betrag der betragsgrößten Abweichung wird gemerkt.

Bei einem Abbruch gibt der Rückgabewert der Funktion den negierten Index des Vektors an, für den die Orthonormierung gescheitert ist.
Bei gelungener Orthonormierung wird der 10er-Logarithmus der betragsgrößten Abweichung zurückgegeben. Ist die betragsmäßig größte Abweichung = 0, so wird der Wert 100 zurückgeliefert. Je größer der Rückgabewert, desto genauer ist das Ergebnis der Orthonormierung. Werte größer als 13 sind wüschenswert. Die folgende Seite zeigt eine Implementierung des Algorithmus.

```
c ------------------------------------------------------------ RN_VEKTOR
c   n: Dimension, v w: Vektor
c
c F Rn_V_sqr    (n, v)    := |v|*|v|
c F Rn_V_len    (n, v)    := |v|
c F Rn_V_skalp  (n, v, w) := v * w (Skalarprodukt)
c F Rn_V_normier(n, v, W) := |v|, W := v / |v|
c                            |v| = 0 ==> W unverändert.
c
c s Rn_V_add    (n, v, w, x)  X := v + w
c s Rn_V_sub    (n, v, w, x)  X := v - w
c
c s Rn_rV_add   (n, v, r, w, x) r: double, X := v + r*w
c s Rn_rV_sub   (n, v, r, w, x) r: double, X := v - r*w
c
c F Rn_V_angle_c(n, v, w) := Winkel(v, w) über arc_cos.
c
c F Rn_V_ortho(L,n,m, VV, Rn_V_skalp, Rn_V_normier, Rn_V_sub) := k
c                 VV(L,m)
c       orthogonalisiert  m Vektoren VV im Rn; Funktionsparameter s.o.
c
c           k <  0: Abbruch, Vektor V(-k) = Nullvektor
c           k >= 0: k = -log10(Orthogonalisierungsfehler)
c ------------------------------------------------------------------------

      function Rn_V_ortho(L, n, m, A, skalp,normier,sub)

      implicit none

      integer L, n, m, Rn_V_ortho
      double precision A(L, m)

      integer i, j
      double precision d, sp,sm, skalp,normier,sub
      external                   skalp,normier,sub

      sm = 0

      do i = 1, m
                 do j = 1, i-1
                      sp = skalp  (n, A(1,i), A(1,j))
                      call sub    (n, A(1,i), sp, A(1,j), A(1,i))
                 end do

                      d =  normier(n, A(1,i), A(1,i))
                 if (d .eq. 0) goto 9

                 do j = 1, i-1
                      sp = skalp(n, A(1,i), A(1,j))
                      sm = max(sm, abs(sp))
                 end do
      end do

      if ( sm .eq. 0 ) then
                           i = -100
                       else
                           sm = log10(sm)
                           i  = nint (sm)
                       endif

9     Rn_V_ortho = -i

      end
```

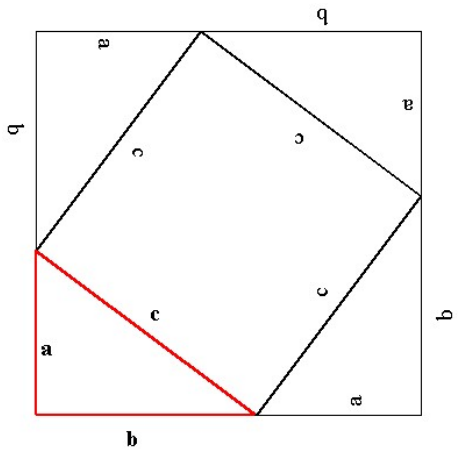
4 Euklid

4.1 Satz des Pythagoras

$$A^2 + B^2 = C^2 \tag{26}$$

Die Summe der Katheten-Quadrate ist gleich dem Quadrat der Hypothenuse.

Beweis:



$$\begin{aligned} c^2 &= 4 * \frac{ab}{2} \\ c^2 &= 2ab \\ c^2 - 2ab &= (a+b)^2 \\ c^2 &= a^2 + b^2 \end{aligned}$$

Abbildung 1: Satz des Pythagoras