```cpp
#include <algorithm>
#include <cmath>
#include <iostream>
#include <numeric>
#include <vector>

#include <cppqt.h>

using namespace std;

void drawAntialiasedWideLine(Drawing& pic, DPoint2D p1, DPoint2D p2,
                             double w, int f = 4, int colour = 0)
// zeichnet eine Linie von p1 nach p2 der Dicke w, der Algorithmus verwendet
// zur Berechnung ein f-fach in x- und y-Richtung verfeinertes Raster
{
//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
    if (p1.y > p2.y)
        swap(p1, p2);                          // dann ist p2 der obere Punkt


  DPoint2D p11, p12, p21, p22;           // Eckpunkte des Linienrechtecks im
                                         // feineren Raster
  DPoint2D nv(p2.y - p1.y, p1.x - p2.x); // Normalenvektor (zeigt erstmal immer
                                         // nach rechts bzw. im Extremfall nach
                                         // oben oder unten)

  if (nv.y < 0)
    nv = -1.0 * nv;                      // dann zeigt nv nicht nach unten

  nv = nv / norm(nv);                    // normiere nv



  // Eckpunkte mittels Normalenvektor berechnen
```

```
35   p11 = static_cast<double>(f) * (p1 - 0.5 * w * nv);
36   p12 = static_cast<double>(f) * (p1 + 0.5 * w * nv);
37   p21 = static_cast<double>(f) * (p2 - 0.5 * w * nv);
38   p22 = static_cast<double>(f) * (p2 + 0.5 * w * nv);
39   // nv zeigt immer nach oben, deshalb liegt p11 unterhalb von p12 und
40   // p21 unterhalb von p22.
41
42
43   //ymin un ymax in f-fach Raster
44   double yminf = p11.y;
45   double ymaxf = p22.y;
46   //x-Bereich im Original-Raster
47   int xmax = static_cast<int>(ceil(max(p12.x / f, p21.x / f)));
48   // Anzahl Zeilen im feinen Raster
49   // Selbst für ymaxf == 0 wird immernoch eine Zeile benötigt.
50   int numrows = static_cast<int>(ceil(ymaxf)) + 1;
51
52
53   // Linker und rechter Rand der Linie im feinen Raster.<<<
54   //Initialisiere so, dass die Ränder sukzessive aktualisiert werden können,
55   //denn es ist nicht bekannt ob eine Seite des Rechtecks „links“ bzw.
56   // „rechts“ liegt.
57   vector<int> linkerrand(numrows, numeric_limits<int>::max());   //aller_werte=MAX_INT
58   vector<int> rechterrand(numrows, -1);
59   int y;
60   double x;
61   double einsdurchm;
62
63
64
65   // Bestimme nun anhand der 4 Rechteckseiten für jede Bildzeile im feinen
66   // Raster den x-Bereich.
67   // Ob eine Seite des Rechtecks „links“ bzw. „rechts“ liegt, ist unbekannt.
68   // Im Allgemeinen werden die Bereiche [p11.y,p12.y], [p11.y,p21.y],
```

```
69   // [p21.y,p22.y] und [p12.y, p22.y] überlappen.
70   if (p1.x != p2.x)
71     {
72       // Bereich zwischen p11.y und p12.y
73       einsdurchm = (p11.x - p12.x) / (p11.y - p12.y);
74       x = p11.x + (ceil(p11.y) - p11.y) * einsdurchm;
75       for (y = static_cast<int>(ceil(p11.y)); y <= floor(p12.y); ++y)
76         {
77           linkerrand[y] = min(linkerrand[y], static_cast<int>(round(x)));  //MAX vs x
78           rechterrand[y] = max(rechterrand[y], static_cast<int>(round(x)));
79           x += einsdurchm;
80         }
81
82       // Bereich zwischen p21.y und p22.y
83       einsdurchm = (p21.x - p22.x) / (p21.y - p22.y);
84       x = p21.x + (ceil(p21.y) - p21.y) * einsdurchm;
85       for (y = static_cast<int>(ceil(p21.y)); y <= floor(p22.y); ++y)
86         {
87           linkerrand[y] = min(linkerrand[y], static_cast<int>(round(x)));
88           rechterrand[y] = max(rechterrand[y], static_cast<int>(round(x)));
89           x += einsdurchm;
90         }
91     }
92
93
94
95   if (p1.y != p2.y)
96     {
97       // Bereich zwischen p11.y und p21.y
98       einsdurchm = (p11.x - p21.x) / (p11.y - p21.y);
99       x = p11.x + (ceil(p11.y) - p11.y) * einsdurchm;
100      for (y = static_cast<int>(ceil(p11.y)); y <= floor(p21.y); ++y)
101        {
102          linkerrand[y] = min(linkerrand[y], static_cast<int>(round(x)));
```

```
103           rechterrand[y] = max(rechterrand[y], static_cast<int>(round(x)));
104           x += einsdurchm;
105         }
106       // Bereich zwischen p12.y und p22.y
107       einsdurchm = (p12.x - p22.x) / (p12.y - p22.y);
108       x = p12.x + (ceil(p12.y) - p12.y) * einsdurchm;
109       for (y = static_cast<int>(ceil(p12.y)); y <= floor(p22.y); ++y)
110         {
111           linkerrand[y] = min(linkerrand[y], static_cast<int>(round(x)));
112           rechterrand[y] = max(rechterrand[y], static_cast<int>(round(x)));
113           x += einsdurchm;
114         }
115     }
116
117
118
119   // y-Bereich im Original-Raster
120   int ymin = static_cast<int>(round(1.0 / f * yminf));
121   int ymax = static_cast<int>(round(1.0 / f * ymaxf));
122   // Für jedes Pixel der ursprünglichen Pixelzeile, summiere die
123   // Intensitäten innerhalb dieser Zeile.
124   // Ein Pixel mehr, für Rechenungenauigkeiten.
125   vector<int> xx(xmax + 1, 0);
126   int xxmin, xxmax;
127   int xi, xf, xfend, z;
128
129   // Schleife über die Zeilen des Originalbildes
130   for (y = ymin; y <= ymax; ++y)
131     {
132       // Enthält später den linken bzw. rechten Rand der zugehörigen
133       // echten Pixelzeile.
134       xxmin = numeric_limits<int>::max();
135       xxmax = -1;
136       // Schleife über die f Zeilen des feineren Rasters pro echter Zeile
```

```
137       for (z = 0; z < f && f * y + z <= ymaxf; ++z)
138         {
139           // Finde den vorher berechneten Rand der Subpixelzeile
140           xf = linkerrand[f * y + z];
141           xfend = rechterrand[f * y + z];
142           // Wenn mindestens ein Subpixel
143           if (xf <= xfend)
144             {
145               // bestimme kleinstes und größtes x in Subpixeln
146               xxmin = min(xf, xxmin);
147               xxmax = max(xfend, xxmax);
148               // Addiere die Pixel des feineren Bildes auf. Laufe dazu über
149               // die Subpixelzeile und addiere im Eintrag, der zum Originalpixel
150               // im Vektor xx gehört.
151               for ( ; xf <= xfend; ++xf)
152                 xx[xf / f] += 1;
153             }
154         }
155       // Wenn Zeile nicht leer
156       if (xxmin <= xxmax)
157         // Schleife über den zu zeichnenden Bereich in Zeile y, in echten Pixeln
158         for (xi = xxmin / f; xi <= xxmax / f; xi++)
159           // Schleife über den zu zeichnenden Bereich in Zeile y
160           {
161             // bestimme die Intensität des Pixels
162             // 0  ↦ 255 (weiß)
163             // f² ↦ col (die gewünschte Farbe der Linie)
164             double n = 255;
165             double m = (colour - 255.0) / (f * f);
166             pic.drawPoint(xi, y, static_cast<int>(round( m * xx[xi] + n )),
167                           false);
168             // lösche den Intensitätswert für die nächste Zeile
169             xx[xi] = 0;
170           }
```

```
171     }
172 //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
173 }
174
175 int maindraw( )
176 {
177   Drawing pic1(170, 170);
178
179   pic1.show( );
180
181   DPoint2D p1, p2;
182   double width;
183   int f;
184
185   cout << "Dicke, Raster: ";
186   cin >> width >> f;
187
188   for (int theta = 0; theta <= 90; theta += 9)
189     {
190       p1 = DPoint2D(10,10);
191       p2 = p1 + 150.0 * DPoint2D(sin(theta * M_PI / 180.0),
192                                  cos(theta * M_PI / 180.0));
193       drawAntialiasedWideLine(pic1, p1, p2, width, f, 0);
194     }
195
196   cin.get( );
197   cout << "Return zum Vergrößern" << endl;
198   cin.get( );
199   pic1.setZoom(4);
200
201   IOThread::waitForWindow(60);
202
203   return 0;
204 }
```

```
205
```