```cpp
#include "ChartControl.h"
#include <wx/settings.h>
#include <wx/graphics.h>
#include <wx/dcbuffer.h>

// wxFULL_REPAINT_ON_RESIZE needed for Windows
ChartControl::ChartControl(wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size) :
    wxWindow(parent, id, pos, size, wxFULL_REPAINT_ON_RESIZE)
{
    this->SetBackgroundStyle(wxBG_STYLE_PAINT); // needed for windows

    this->Bind(wxEVT_PAINT, &ChartControl::OnPaint, this);
}

void ChartControl::Set(const wxString& title_, const wxString& lt, const wxString& rt, const wxString& lb, const wxString& rb) {
    title = title_;
    tlt = lt;
    trt = rt;
    tlb = lb;
    trb = rb;
}

void ChartControl::Set(const wxString& title_, const wxString& lt) {
    title = title_;
    tlt = lt;
}

void ChartControl::Set(
    const std::vector<double>& values_ltx,
    const std::vector<double>& values_lty,
    const std::vector<double>& values_rbx,
    const std::vector<double>& values_rby) {
    ltx = values_ltx;
    lty = values_lty;
```

```
35      rbx = values_rbx;
36      rby = values_rby;
37      quad = true;
38  }
39
40  void ChartControl::Set(
41      const std::vector<double>& values_ltx,
42      const std::vector<double>& values_lty) {
43      ltx = values_ltx;
44      lty = values_lty;
45      quad = false;
46  }
47
48  void ChartControl::OnPaint(wxPaintEvent& evt)
49  {
50      wxAutoBufferedPaintDC dc(this);
51      dc.Clear();
52      wxGraphicsContext* gc = wxGraphicsContext::Create(dc);
53
54      if (gc)
55      {
56          wxFont titleFont = wxFont(wxNORMAL_FONT->GetPointSize() * 1.5,
57              wxFONTFAMILY_DEFAULT, wxFONTSTYLE_NORMAL, wxFONTWEIGHT_BOLD);
58
59          gc->SetFont(titleFont, wxSystemSettings::GetAppearance().IsDark() ? *wxWHITE : *wxBLACK);
60
61          double tw, th;
62          gc->GetTextExtent(this->title, &tw, &th);
63          const double titleTopBottomMinimumMargin = this->FromDIP(10);
64          wxRect2DDouble chartArea{ 0.0,0.0,static_cast<double>(GetSize().GetWidth()), static_cast<double>
              (GetSize().GetHeight()) };
65          const double marginX = chartArea.GetSize().GetWidth() / 8.0;
66          const double marginTop = std::max(chartArea.GetSize().GetHeight() / 8.0,
              titleTopBottomMinimumMargin * 2.0 + th);
67          const double marginBottom = chartArea.GetSize().GetHeight() / 8.0;
```

```
68          chartArea.Inset(marginX, marginTop, marginX, marginBottom);
69
70          gc->DrawText(this->title, marginX + (chartArea.GetSize().GetWidth() - tw) / 2.0,
              titleTopBottomMinimumMargin);
71
72          wxFont subtitleFont = wxFont(wxNORMAL_FONT->GetPointSize(),
73              wxFONTFAMILY_DEFAULT, wxFONTSTYLE_NORMAL, wxFONTWEIGHT_MEDIUM);
74
75          gc->SetFont(subtitleFont, wxSystemSettings::GetAppearance().IsDark() ? *wxWHITE : *wxBLACK);
76
77          if (quad) {
78              auto halfWidth = static_cast<double>(GetSize().GetWidth()) / 2.0;
79              auto halfHeight = static_cast<double>(GetSize().GetHeight()) / 2.0;
80
81              wxRect2DDouble leftTopArea{ marginX / 2.0, th, halfWidth, halfHeight };
82              DrawComponent(dc, gc, leftTopArea, tlt, ltx, lty, false, true);
83
84              wxRect2DDouble rightTopArea{ halfWidth, th, halfWidth, halfHeight };
85              DrawComponent(dc, gc, rightTopArea, trt, rbx, lty, false, false);
86
87              wxRect2DDouble leftBottomArea{ marginX / 2.0, halfHeight, halfWidth, halfHeight };
88              DrawComponent(dc, gc, leftBottomArea, tlb, rbx, rby, true, true);
89
90              wxRect2DDouble rightBottomArea{ halfWidth, halfHeight, halfWidth, halfHeight };
91              DrawComponent(dc, gc, rightBottomArea, trb, rbx, rby, true, false);
92          }
93          else {
94              auto Width = static_cast<double>(GetSize().GetWidth());
95              auto Height = static_cast<double>(GetSize().GetHeight());
96
97              wxRect2DDouble leftTopArea{ 0.0, th, Width, Height };
98              DrawComponent(dc, gc, leftTopArea, tlt, ltx, lty, true, true);
99          }
100
101         delete gc;
```

```
102     }
103 }
104
105 void ChartControl::DrawComponent(
106     wxAutoBufferedPaintDC& dc,
107     wxGraphicsContext* gc,
108     wxRect2DDouble chartArea,
109     const wxString& chartTitle,
110     const std::vector<double>& x,
111     const std::vector<double>& y,
112     bool drawXLabels,
113     bool drawYLabels
114 ) {
115
116     auto values = y;
117     double tw, th;
118     gc->GetTextExtent(this->title, &tw, &th);
119     const double titleTopBottomMinimumMargin = this->FromDIP(10);
120
121     const double marginX = chartArea.GetSize().GetWidth() / 8.0;
122     const double marginTop = std::max(chartArea.GetSize().GetHeight() / 8.0, titleTopBottomMinimumMargin *
        2.0 + th);
123     const double marginBottom = chartArea.GetSize().GetHeight() / 8.0;
124     double labelsToChartAreaMargin = this->FromDIP(10);
125
126     chartArea.Inset(marginX, marginTop, marginX, marginBottom);
127
128     wxFont subtitleFont = wxFont(wxNORMAL_FONT->GetPointSize(),
129         wxFONTFAMILY_DEFAULT, wxFONTSTYLE_NORMAL, wxFONTWEIGHT_BOLD);
130
131     gc->SetFont(subtitleFont, wxSystemSettings::GetAppearance().IsDark() ? *wxWHITE : *wxBLACK);
132
133     gc->DrawText(chartTitle, chartArea.GetPosition().m_x + (chartArea.GetSize().GetWidth() - tw) / 2.0,
       chartArea.GetPosition().m_y - (marginTop) / 2.0 );
134
```

```
135     wxAffineMatrix2D normalizedToChartArea{};
136     normalizedToChartArea.Translate(chartArea.GetLeft(), chartArea.GetTop());
137     normalizedToChartArea.Scale(chartArea.m_width, chartArea.m_height);
138
139     // vertical axis
140     double lowValue = *std::min_element(values.begin(), values.end());
141     double highValue = *std::max_element(values.begin(), values.end());
142
143     const auto& [segmentCount, rangeLow, rangeHigh] = calculateChartSegmentCountAndRange(lowValue,
          highValue);
144
145     double yValueSpan = rangeHigh - rangeLow;
146     lowValue = rangeLow;
147     highValue = rangeHigh;
148
149     double yLinesCount = segmentCount + 1;
150
151     wxAffineMatrix2D normalizedToValueY{};
152     normalizedToValueY.Translate(0, highValue);
153     normalizedToValueY.Scale(1, -1);
154     normalizedToValueY.Scale(static_cast<double>(values.size() - 1), yValueSpan);
155
156     gc->SetPen(wxPen(wxColor(128, 128, 128)));
157     gc->SetFont(*wxNORMAL_FONT, wxSystemSettings::GetAppearance().IsDark() ? *wxWHITE : *wxBLACK);
158
159     for (int i = 0; i < yLinesCount; i++)
160     {
161         double normalizedLineY = static_cast<double>(i) / (yLinesCount - 1);
162
163         auto lineStartPoint = normalizedToChartArea.TransformPoint({ 0, normalizedLineY });
164         auto lineEndPoint = normalizedToChartArea.TransformPoint({ 1, normalizedLineY });
165
166         wxPoint2DDouble linePoints[] = { lineStartPoint, lineEndPoint };
167         gc->StrokeLines(2, linePoints);
168
```

```
169         if (drawYLabels) {
170             double valueAtLineY = normalizedToValueY.TransformPoint({ 0, normalizedLineY }).m_y;
171
172             auto text = wxString::Format("%.2f", valueAtLineY);
173             text = wxControl::Ellipsize(text, dc, wxELLIPSIZE_MIDDLE, chartArea.GetLeft() -
                  labelsToChartAreaMargin);
174
175             double tw, th;
176             gc->GetTextExtent(text, &tw, &th);
177             gc->DrawText(text, chartArea.GetLeft() - labelsToChartAreaMargin - tw, lineStartPoint.m_y -
                  th / 2.0);
178         }
179     }
180
181     // horizontal axis
182     values = x;
183     lowValue = *std::min_element(values.begin(), values.end());
184     highValue = *std::max_element(values.begin(), values.end());
185
186     const auto& [xsegmentCount, xrangeLow, xrangeHigh] = calculateChartSegmentCountAndRange(lowValue,
          highValue);
187
188     double xValueSpan = xrangeHigh - xrangeLow;
189     lowValue = xrangeLow;
190     highValue = xrangeHigh;
191
192     double xLinesCount = xsegmentCount + 1;
193
194     wxAffineMatrix2D normalizedToValueX;
195     normalizedToValueX.Translate(highValue, 0);
196     normalizedToValueX.Scale(-1, 1);
197     normalizedToValueX.Scale(xValueSpan, static_cast<double>(values.size() - 1));
198
199     gc->SetPen(wxPen(wxColor(128, 128, 128)));
200     gc->SetFont(*wxNORMAL_FONT, wxSystemSettings::GetAppearance().IsDark() ? *wxWHITE : *wxBLACK);
```

```
201
202     for (int i = 0; i < xLinesCount; i++)
203     {
204         double normalizedLineX = static_cast<double>(i) / (xLinesCount - 1);
205
206         auto lineStartPoint = normalizedToChartArea.TransformPoint({ normalizedLineX, 0 });
207         auto lineEndPoint = normalizedToChartArea.TransformPoint({ normalizedLineX, 1 });
208
209         wxPoint2DDouble linePoints[] = { lineStartPoint, lineEndPoint };
210         gc->StrokeLines(2, linePoints);
211
212         if (drawXLabels) {
213             double valueAtLineX = normalizedToValueX.TransformPoint({ normalizedLineX, 0 }).m_x;
214
215             auto text = wxString::Format("%.2f", valueAtLineX);
216             text = wxControl::Ellipsize(text, dc, wxELLIPSIZE_MIDDLE, chartArea.GetLeft() -
                  labelsToChartAreaMargin);
217
218             double tw, th;
219             gc->GetTextExtent(text, &tw, &th);
220             gc->DrawText(text, lineStartPoint.m_x - tw / 2.0, chartArea.GetLeftBottom().m_y + th / 2.0);
221         }
222     }
223
224     wxPoint2DDouble leftHLinePoints[] = {
225         normalizedToChartArea.TransformPoint({0, 0}),
226         normalizedToChartArea.TransformPoint({0, 1}) };
227
228     wxPoint2DDouble rightHLinePoints[] = {
229         normalizedToChartArea.TransformPoint({1, 0}),
230         normalizedToChartArea.TransformPoint({1, 1}) };
231
232     gc->StrokeLines(2, leftHLinePoints);
233     gc->StrokeLines(2, rightHLinePoints);
234
```

```
235     // line plot
236     wxPoint2DDouble* pointArray = new wxPoint2DDouble[values.size()];
237
238     wxAffineMatrix2D valueToNormalized = normalizedToValueY;
239     valueToNormalized.Invert();
240     wxAffineMatrix2D valueToChartArea = normalizedToChartArea;
241     valueToChartArea.Concat(valueToNormalized);
242
243     for (int i = 0; i < values.size(); i++)
244     {
245         pointArray[i] = valueToChartArea.TransformPoint({ x[i], y[i]});
246     }
247
248     gc->SetPen(wxPen(wxSystemSettings::GetAppearance().IsDark() ? *wxCYAN : *wxBLUE, 3));
249     gc->StrokeLines(values.size(), pointArray);
250
251     delete[] pointArray;
252 }
253
254 std::tuple<int, double, double> ChartControl::calculateChartSegmentCountAndRange(double origLow, double
      origHigh)
255 {
256     constexpr double rangeMults[] = {0.2, 0.25, 0.5, 1.0, 2.0, 2.5, 5.0};
257     constexpr int maxSegments = 6;
258
259     double magnitude = std::floor(std::log10(origHigh - origLow));
260
261     for (auto r : rangeMults)
262     {
263         double stepSize = r * std::pow(10.0, magnitude);
264         double low = std::floor(origLow / stepSize) * stepSize;
265         double high = std::ceil(origHigh / stepSize) * stepSize;
266
267         int segments = round((high - low) / stepSize);
268
```

```
269             if (segments <= maxSegments)
270             {
271                 return std::make_tuple(segments, low, high);
272             }
273         }
274 
275         // return some defaults in case rangeMults and maxSegments are mismatched
276         return std::make_tuple(10, origLow, origHigh);
277     }
```