Real-time charting in TeeChart VCL

Occasionally you want to plot data at the moment it was measured or generated? You want to plot it in real-time. An ideal real-time plotting would mean data would be plotted at the moment it’s generated. Of course this is only idealization in real life you’ll be looking for methods which will improve plotting speed and thus minimize delay between data being measured and data being plotted. This article discusses several methods you can use to ensure data will be plotted as fast as possible.

Since in real-time charting speed is the name of the game, we’ll try to use different tricks to speed up plotting time. This involves:

- choosing best series type,
- speeding-up plot drawing time by disabling all fancy features you don’t need in real-time charting,
- pre-processing your data,
- choosing the best method to populate series with data.

Choose correct series type

You should use TFastLineSeries, TPointSeries, or if you’re plotting histograms, THistogramSeries or TVolumeSeries. The preferred choice, if possible, is TFastLineSeries. This series is missing some “regular” TLineSeries properties, but it’s a lot faster when it comes to drawing simple $y=y(x)$ functions.

Additionally, TFastLineSeries introduces several properties for fast drawing. These include:

- The DrawAllPoints boolean property, default value True. Normally chart size is limited to a fixed number of screen pixels. This means that if, for example, you have 1.000.000 points, they will inevitably “share” the same screen pixel coordinate (in horizontal, vertical or both directions). Drawing an algorithm will then plot multiple points with different real x,y coordinates at the same screen coordinate. After multiple calls to drawing the algorithm and waste of cpu time you’ll end up with a single painted screen pixel. In this case a reasonable thing to do is group the points with the same x screen pixel coordinate and replace them with two points (group minimum and maximum values). The end result will visually be the same as drawing all the points in the group. But it will be a lot faster, especially if there are lots of points per group. Setting DrawAllPoints to False does precisely that: the internal algorithm processes data and draws only non-repeated (group) points. Using this trick you can plot millions of points in realtime with little fuss.

- The FastPen property, default value False. But if you set it to True, fastline series will use solid width=1 pen for drawing. Using this trick works only on Windows 2000, XP and 2003 operating systems.

- Series Delete method. The Delete method now includes a second parameter which controls how many points will be deleted from a series. This allows fast delete of multiple points in a single call, which is much faster than deleting multiple points using a loop.

- Series AutoRepaint property, default value True, meaning adding new value will result in all values being repaint ed again. But if you set this property to False, new points will be painted as they are added to a series, without redrawing the whole chart.

Additional speed up can be achieved by setting Series.XValues.Order property to clNone, meaning no internal sorting will be performed on data.

Disable/hide some chart elements

Each element drawn on a chart somewhat increases chart drawing time. So it makes good sense to hide all chart elements you don’t need. This includes chart legend, chart title, chart frame. Additionally you might want to manually define chart axis increments and axis ranges and thus avoid internal calculation algorithms being executed too often. Also, you could use a new TChartAxes.FastCal property introduced in TeeChart v6.

Setting this property to True might give you an additional decrease in drawing time. In the example below we’ll be setting up a chart for fast realtime drawing. We’ll try to include all the properties/methods mentioned above:

```pascal
// Prepare chart for maximum speed:
with Chart1 do
begin
```
ClipPoints := False;
Title.Visible := False;
Legend.Visible := False;
LeftAxis.Axis.Width := 1;
BottomAxis.Axis.Width := 1;
BottomAxis.RoundFirstLabel := False;
View3D := False;
end;

// Number of points we'll be displaying
MaxPoints := 10000;
// Number of points deleted when scrolling chart
ScrollPoints := 5000;

// Prepare series.
// Disable AutoRepaint and X Order

// AutoRepaint=False means "real-time" drawing mode.
// Points are displayed just after adding them,
// without redrawing the whole chart.
Series1.AutoRepaint := False;

// Set Ordering to none, to increment speed when adding points
Series1.XValues.Order := loNone;

// Initialize axis scales
// we're assuming left axis values are within [0,1000]
Chart1.LeftAxis.SetMinMax(0, 10000);
Chart1.BottomAxis.SetMinMax(1, MaxPoints);

// Speed tips:

// When using only a single thread, disable locking:
Series1.LinePen.OwnerCriticalSection := nil;

// For Windows NT, 2000 and XP only:
// Speed realtime painting with solid pens of width 1.
Series1.FastPen := True;

// Set axis calculations in "fast mode".
// Note: For Windows Me and 98 might produce bad drawings when
// chart zoom is very big.
Chart1.Axes.FastCalc := True;

We've set up the chart and series for fast real-time plotting. Next on our task list is selecting the best method to populate the series with data points.

**Populate series with data**

The easiest solution is to use the AddXY method to add points to a series. A big advantage of this method is that it's very simple to use. This is the preferred method for adding points if you're plotting at real-time and the number of points shown doesn't exceed a couple of thousand. Together with TChartSeries.Delete method it provides a powerful method to do real-time plotting. The following two routines are used in one of the TeeChart examples to perform real-time scrolling of the chart. First the routine adds new points to the series, second a routine scrolls points as new data is added and deletes old unnecessary points:

// Adds a new random point to Series
Procedure RealTimeAdd(Series:TChartSeries);
var XValue,YValue : Double;
begin
  if Series.Count=0 then // First random point
    begin
      YValue:=Random(10000);
      XValue:=1;
    end
  else
    begin
      // Next random point
      YValue:=Series.YValues.Last+Random(10)-4.5;
      XValue:=Series.XValues.Last+1;
    end;
  // Add new point
  Series.AddXY(XValue,YValue);
end;

// When the chart is filled with points, this procedure
// deletes and scrolls points to the left.
Procedure DoScrollPoints(Series: TChartSeries);
var tmp,tmpMin,tmpMax : Double;
begin
  // Delete multiple points with a single call.
  // Much faster than deleting points using a loop.
  Series.Delete(0,ScrollPoints);

  // Scroll horizontal bottom axis
  tmp := Series.XValues.Last;
  Series.GetHorizAxis..SetMinMax(tmp-MaxPoints+ScrollPoints,tmp+ScrollPoints);

  // Scroll vertical left axis
  tmpMin := Series.YValues.MinValue;
  tmpMax = Series.YValues.MaxValue;
  Series.GetVertAxis.SetMinMax(tmpMin-tmpMin/5,tmpMax+tmpMax/5);

  // Do chart repaint after deleting and scrolling
  Application.ProcessMessages;
end;

Another way of adding a large number of points is to use direct dynamic arrays. In this case we're bypassing
the AddXY method and accessing the Series x,y values directly, and thus avoid AddXY method overhead. Here
is the code you can use to populate a series with a large number of points:

Var X,Y : Array of Double; // TChartValues
t : Integer;
Num : Integer;
begin
  { 1M points }
  Num:= 1000000;

  { allocate our custom arrays }
  SetLength(X,Num);
  SetLength(Y,Num);
{ fill data in our custom arrays }
X[0]:=0;
Y[0]:=Random(10000);
for t:=1 to Num-1 do
begin
X[t]:=t;
Y[t]:=Y[t-1]+Random(101)-50;
end;

{ set our X array }  
With Series1.XValues do
begin
  Value:=TChartValues(X);   { <-- the array }  
  Count:=Num;               { <-- number of points }  
  Modified:=True;           { <-- recalculate min and max }  
end;

{ set our Y array }  
With Series1.YValues do
begin
  Value:=TChartValues(Y);   
  Count:=Num;               
  Modified:=True;           
end;

{ Show data }  
Series1.Repaint;

In the example above we first generated some data for the a series and then assigned the data arrays directly to the a series XValues.Value and YValues.Value arrays. Please note that we had to manually define the XValues.Count and YValues.Count properties.

Conclusion

In TeeChart real-time plotting involves selecting the correct series type, setting some series and chart properties and using the appropriate method to populate a series with data. Actual coding might vary with the actual data you're trying to plot, but basic ideas outlined in this article can still be used with good results.

Note: All these features are also demonstrated in the TeeChart VCL demo. Especially, check all items under the "All Features -> Speed" tree node.