How to improve performance and memory usage with a large number of polygons on a canvas?

Asked 16 years, 2 months ago Modified 15 years, 1 month ago

Viewed 4k times

5

The project I'm working on needs to render an ESRI shape file, which can have a large number of polygons/shapes. When I add all these polygons, lines, points, etc to the canvas I'm using, it gets really slow.

To draw the shapes on the map, I'm creating a Path object, and setting it's Data property to a StreamGeometry. I originally used a Polygon, but according to MSDN a StreamGeometry is much lighter weight.

How can I improve the performance? Would converting the finished product to a Bitmap, or VisualBrush help? Is there a more efficient way of to render all these shapes onto the canvas?

EDIT: I forgot to mention that this needs to be able to work in a Partial-Trust XBAP.

wpf performance

Share

Improve this question

Follow

edited Sep 29, 2008 at 20:08

asked Sep 29, 2008 at 19:03

Dylan

2,422 ● 6 ● 27 ● 34

8 Answers

Sorted by: Highest score (default)

6

No need to resort to GDI, you just need to move down a layer in the WPF API's and combine your geometry into fewer visuals. Pablo Fermicola has some useful information about picking which layer to use depending on your performance needs.

I've managed to get excellent performance using DrawingVisual and the DrawingContext class.

Share Improve this answer

Follow

answered Sep 29, 2008 at 20:27

Samuel Jack

33.2k ● 18 ● 120 ● 155

DrawingVisual with DrawingContext works fast and well. Also check out GeometryDrawing and StreamGeometry if you are in a situation where it make sense to save and re-use parts of your geometry. (DrawingContext can only update the visual as a whole, whereas StreamGeometryContext updates a Geometry that may be a small portion of the whole DrawingVisual) – Ray Burns Nov 19, 2009 at 10:07

3

You could try using GDI+ or direct x instead of WPF.

I did a similar project (rendering Map data via WPF), for an MSDN magazine article I wrote about a year ago.

It was just something I wrote rather quickly (the article + the app took about 1 week), and was designed to just be a "this is something neat you can do", and so I didn't focus on performance much. In any case, I ran into the same problem. Once the number of polygons on the canvas got large, the rendering performance started to suffer. For example, resizing the window would involve about a 1/2 second delay or so.

A lot of this has to do with the overhead of WPF. It's primarily designed as an GUI toolkit, rather than a high performance graphic engine. That means it focuses on feature richness over efficent rendering. With smaller numbers of objects (what is likely to be found in most GUI applications), the performance is pretty good and the databinding, animation, and declarative style features (and all the event routing and other things they require) can really come in handy.

When drawing a map, however, the sheer volume of data can cause all those neat data binding features to cause perf problems, as you seem well aware.

If you don't need declarative styling and animation, then I would just eliminate WPF, and use GDI+ to draw the map your self. It basicly involves setting up the right transformation matrix to get the map to draw onto a control surface, and then iterating through all the polygons and calling a bunch of DrawPolygon methods.

To enable interaction you will have to write your own hit testing code, and you will have to redraw the map anytime the form is resized. You will also have to hand code any animations or style changes or things like that you want to do (such as highlighting regions when the mouse is over them). But, it shouldn't be that difficult to write that code. I would imaging you could do it in about 1.5 weeks.

Doing it that way, however, should improve performance, because it would come down to doing only about 20-30K vector transformations, which doesn't take much processor power on most modern CPU's. You could also look into using Direct X, which would allow you to take advantage of the GPU as well, which could give an even bigger performance boost.

Share Improve this answer Follow

answered Sep 29, 2008 at 19:53

Scott Wisniewski
25k ● 8 ● 62 ● 90

---

1 Depending on what you are doing, WPF's GeometryDrawing and StreamGeometry classes can actually be much faster than GDI+, since it uses Direct3D in retained mode. GDI+ works well for static images, but not so well for images that are mostly static but with some changes. GDI+ also does not perform as well in remote desktop scenarios. In general I would recommend using the WPF low level drawing layers over the GDI+ ones in most cases because all the transforms

are taken care of for you and you don't have to deal with air space and similar issues. – Ray Burns Nov 19, 2009 at 10:02

As far as DirectX goes, it can be faster than WPF in the local scenario (though impossible in the remote), but it is much more difficult to code to than StreamGeometry or GDI+. Also it can't run in an XBAP. – Ray Burns Nov 19, 2009 at 10:04

GDI+ has a global lock for most operations so it can only be doing one thing at a time within a process. Since WPF can do many things at once this alone can give performance gains if drawing multiple images at once. – user310988 Sep 6, 2012 at 8:20

1

You may be limited by the performance of the canvas widget. If you have the option of using a 3rd party toolkit look at QT. It has a high performance canvas widget that is designed to render complex shapes quickly. This has already been used for at least one GIS application, so it has some track record in this space.

You can wrap QT as an ActiveX control or use the Qyoto/Kimono .Net bindings to interface to a .Net application. Troll Tech have just revamped their website and I can't find the downloadable demo they used to have there but it shows the QGraphicsView widget rendering a very large vector drawing and zooming out in real time.

Share Improve this answer Follow

answered Sep 29, 2008 at 19:27

ConcernedOfTunbridge Wells

66.5k ● 15 ● 148 ● 198

Thanks for the link to QT, not sure if it will work for my implementation, because I need to deploying as an XBAP. – Dylan Sep 29, 2008 at 19:41

QT also has a browser plugin facility - if you hunt around the developer support area you should be able to find it. You might want to edit your original post to clarify the XBAP requirement. – ConcernedOfTunbridgeWells Sep 29, 2008 at 19:47

0

You are about to discover the motivation behind GPUs and insanely overclocked and cooled-down graphics cards. And double-buffering.

And: converting the thing to a bitmap: what's the difference to rendering it?

After all you have n objects that /somehow/ have to get rendered (unless you can figure out which objects are hidden behind other objects but that doesn't help you much since you'd have to look at n! object relations).

Besides: maybe it pays off to leave the orthodox OOP approach and go procedural instead.

Share Improve this answer Follow

answered Sep 29, 2008 at 19:18

xmjx
1,153 ● 1 ● 7 ● 18

@xmjx

0

> And: converting the thing to a bitmap: what's the difference to rendering it?

My thought here was that my slowdown could be caused by having n number of objects on a canvas, versus 1 bitmap. That said, I don't know the performance, or what happens in the background when converting a canvas with n number of objects to a Bitmap.

I would prefer not to use a Bitmap, so that I can allow the user to interact (modify/delte) with the shape/polygons.

Share Improve this answer Follow

answered Sep 29, 2008 at 19:23
Dylan
2,422 ● 6 ● 27 ● 34

0

If performance is an issue then you want to avoid Visual Brushes. These brushes are best used to project some visible part of the screen to another place on the screen. They cause a large performance hit because these brushes automatically update when the brush's content changes.

Any usage of this brush will be software rendered, and will not be able to take advantage of the hardware capabilities of your graphic card.

[More info from MSDN on VisualBrush]()

Share Improve this answer

answered Nov 26, 2008 at 15:25

Follow

NigelTufnel
101 ● 1 ● 5

0

Use DrawingGroup and add/remove Drawing objects to solve this. See my answer to a similar question about DrawingVisual performance with Opacity=0. I think my answer actually fits this question better.

Share Improve this answer

Follow

edited May 23, 2017 at 11:44

Community Bot
1 ● 1

answered Apr 4, 2009 at 4:14

eesh
1,414 ● 2 ● 14 ● 25

0

if you do not have to show ALL polygons at once (e.g. due to zooming, panning etc), check out the virtualizing canvas sample here

Share Improve this answer

Follow

answered Nov 18, 2009 at 9:04

Joachim Kerschbaumer
9,841 ● 7 ● 51 ● 84