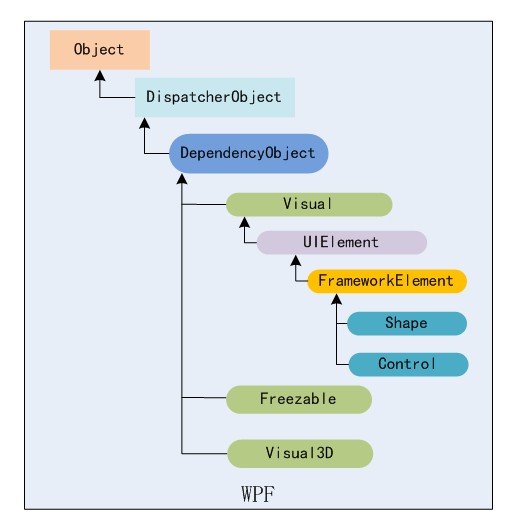
# WPF 绘制性能

## 继承关系



除了UIElement，Visual抽象类还有一下几个子类：

ViewPort3DVisual：用于显示3D内容

ContainerVisual：包含可视化对象的基本容器

**DrawingVisual**：最常用的派生类；**增加了支持“绘制” 希望放置到可视化对象中的图形内容的功能**。实际就是增加了DrawingContext RenderOpen()方法，支持“绘制”的功能，绘制希望放置到可视化对象中的图形内容。

**但是为了在屏幕上实际显示可视化内容 ，仅仅定义可视化对象还是不够的，必须借助功能完备的WPF元素，WPF元素将可视化对象添加到可视化树中。**

**即继承一个UIElement、FrameworkElement或其子类，封装可视化对象，以便其在屏幕显示。**

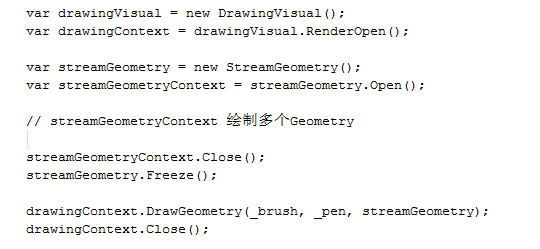


## 如何提高性能？

1. **使用一个WPF元素封装一个包含List<Visual>的对象**

一个WPF元素中可以包含成千上万个Visual元素，使用DrawingContext.DrawXxx()绘制各个Visual；最终返回一个WPF元素用于界面显示。

1. **使用StreamGeometry提高性能**



1. **使用WriteableBitmap（即使用GDI+）**

适用于处理需要操作单个像素的应用程序，或者需要处理来自外部硬件设备（如网络摄像机）的原始图像数据的应用程序。

## Question

1. **为什么使用Visual性能优于WPF元素**

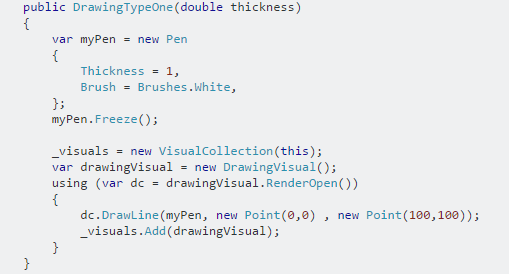
从继承层次上看，WPF元素添加了额外的功能，因此相对更重；

箱子里装100个带盒子的手机和箱子里装不带盒子的手机，前者肯定拿得慢。

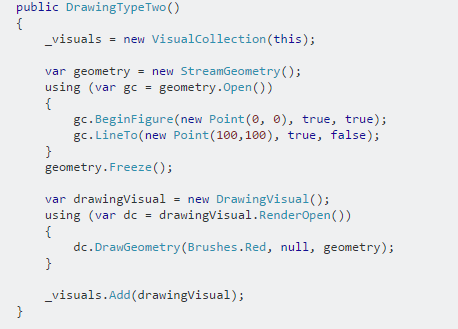
1. **为什么使用StreamGeometry性能优于使用DrawingContext**

比方使用DrawingContext绘制多个Rect，实际会生成多个RectGeometry对象，而使用StreamGeometry可以绘制复杂对象，最终只生成一个Geometry对象，因此性能更好

1. **为什么单个StreamGeometry性能优于单次调用DrawingContext.** **DrawRectangle**
2. 使用DrawingContext. DrawRectangle ()



1. 使用StreamGeometry



从上面两个例子上看，使用StreamGeometry似乎又做了一层封装，应该更“重”，更耗时间，但实际情况不是如此

在WPF中，调用DrawingContext. DrawLine()方法实际上是在内部创建了一个LineGeometry对象，调用DrawRectangle()方法，就内部创建了一个RectagleGeometry对象

在WPF中，最底层的图形的绘制级别是Geometry。而**RectagleGeometry**、**LineGeometry和StreamGeometry**都是Geometry的一个子类。

**StreamGeometry的效率非常之高，因为他是直接保存字节流绘制数据的**。

***StreamGeometry****is light-weight alternative to****PathGeometry****for creating geometric shapes: it does not support databinding, animation, or modification. Use a****StreamGeometry****when you need to describe a complex geometry but do not want the overhead of supporting databinding, animation, or modification. For example, because of its efficiency, the****StreamGeometry****class is a good choice for describing adorners.*

StreamGeometry is all about being able to describe geometry segments (e.g., line’s, bezier’s, etc.) using the least amount of memory possible. The functionality provided by StreamGeometry is nearly identical to PathGeometry’s/PathFigure’s/\*Segment’s, except that **StreamGeometry’s can’t be modified after they are created**.

By forfeiting your ability to modify the geometry, **you get a representation that uses much less memory**. It does this by internally representing each segment **using a byte stream**. Conversely, PathGeometry supports a richer (and more expensive) API by representing geometry as an editable tree/graph of Freezable CLR objects. Not only does this save memory, it can save significant CPU cycles that would be spent instantiating, constructing, and hooking up the PathGeometry. This is particularly true for larger Geometry’s.

**结论：**

RectGeometry是简单的Geometry对象，用于绘制矩形；而StreamGeometry是PathGeometry的轻量级版本，用于绘制复杂的图像。

按道理来说，应该RectGeometry性能更好。但是实际发现使用StreamGeometry具有更优的性能，猜测是因为StreamGeometry绘制好后就不允许修改，在这方面的牺牲导致其可以使用更少的内存（实际使用字节流存储），因此性能更好。**(在进一步的Demo测试发现，还是看情况，复杂对象用StreamGeometry性能会好，如果就是简单对象，RectGeometry更好)**

1. 使用WriteableBitmap（即使用GDI+）是否优于以上两种方法？

**确实是**，

**理论上**以上两种方式是基于DirectX 技术的，支持硬件加速，不应该比GDI/GDI+技术性能差很多；不过实际测试发现差距还是很明显。

**原因可以参考一下两个链接**

<http://stackoverflow.com/questions/8713864/high-performance-graphics-using-the-wpf-visual-layer>

<https://jeremiahmorrill.wordpress.com/2011/02/14/a-critical-deep-dive-into-the-wpf-rendering-system/>

另外，WriteableBitmap本身太简单，提供的API很少，对于复杂图像处理使用有难度，有一个开源的控件再这个基础上做了扩展（不仅支持WPF，还支持WinRT Windows Phone 8 ,Sliverlight），一方面可以保证性能，一方面还有易用性

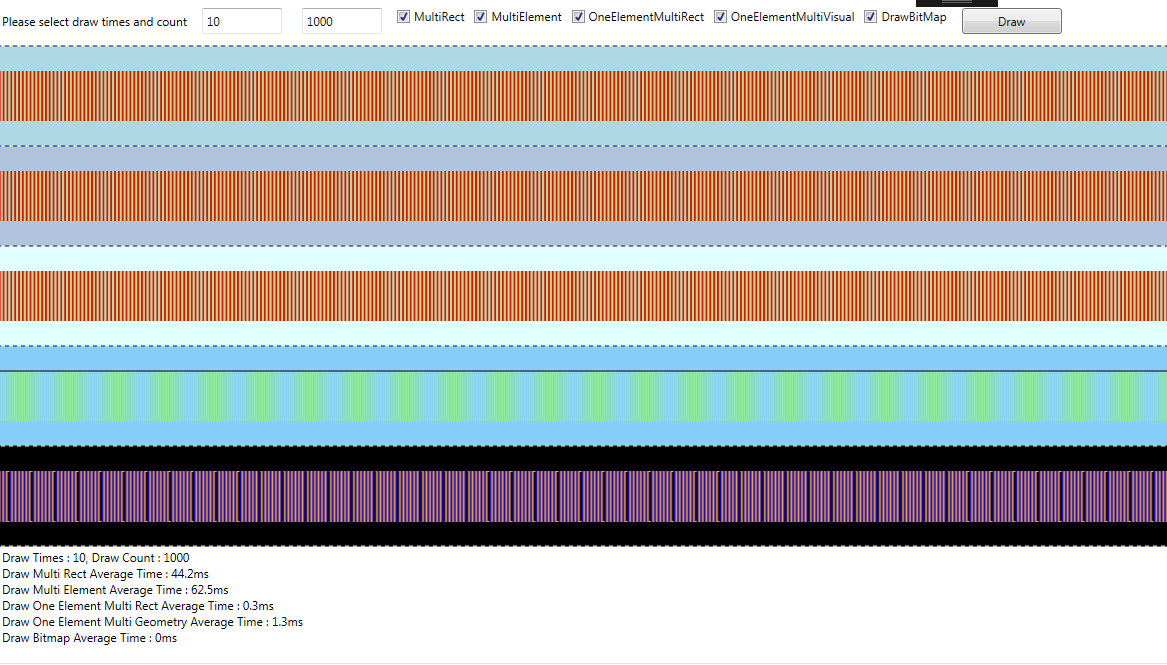
其代码在如下路径：

<https://github.com/teichgraf/WriteableBitmapEx.git>

简单测试了一下，性能确实可以。

# WPF Draw Demo Result

Source Code: <https://github.com/LaiHotchner/WPF.git>



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Draw Times 10** | **1000** | **2000** | **5000** | **10000** |
| Draw Multi Rect Average Time | 44.2ms | 85.2ms | 254.2ms | 768.3ms |
| Draw Multi Element Average Time | 62.5ms | 143.6ms | 472.1ms | 1833.6ms |
| Draw One Element Multi Rect Average Time | 0.3ms | 2.2ms | 2.8ms | 4.6ms |
| Draw One Element Multi Geometry Average Time | 1.3ms | 2.3ms | 5.3ms | 19.9ms |
| Draw Bitmap Average Time | 0ms | 1.2ms | 2.8ms | 7.3ms |

**从结果上看，使用Bitmap 性能有很大的提高；并在在缩放窗口时，在元素超过1000个 的时候，使用前几种方式明显整个程序很卡，而Bitmap不存在这个问题。**