



Windows Phone 7 Performance Best Practices

Allan Murphy
Senior Software Development Engineer
Advanced Technology Group
Microsoft



Hardware Overview

- Hardware
 - 1GHz Qualcomm Snapdragon processor
 - ARM instruction set
 - 480x800 display
 - At least DX9 capable GPU
- Software
 - .NET Compact Framework
 - Lighter weight runtime compared to Windows
 - Simpler JITter and garbage collection



Dev Tools Overview

- Visual Studio plus XNA Game Studio
 - Full version
 - Free VS C# Express
 - Both include WP7 emulator
- Windows Phone 7 application choice
 - XNA Game Studio
 - Silverlight
- Useful additions...
 - Expression Blend 4
 - Windows tools – including CLRProfiler & PIX For Windows



Development Options

- Silverlight
 - Better for simpler interaction, page based presentation
- XNA GS
 - Better for traditional game scenarios
 - Not great for complex static display, lists, etc
- LIVE services
 - Achievements, leaderboards, gamerscore, avatars
 - Part of managed portfolio. working with publishers
 - If you have a great demo or finished game contact:
MoGameSub@Microsoft.com



Garbage Collection *The Price of Freedom*



Garbage Collection Overview

- Perhaps the most pleasant part of managed
 - Don't scoff at its seductive power
- But with great power...
 - ...comes great responsibility
- The A#1 performance problem on WP7:

Intermittent stalls due to garbage collection



Garbage Collection In-Depth

- CF GC runs:
 - After every cumulative 1Mb of memory allocated
 - On fail to allocate
 - On application calling `GC.Collect()`
- CF GC mark and sweep process:
 - Scans entire heap, following live references
 - Mark all objects reachable from live references
 - When done, free everything not marked
 - Sometimes, compact the heap



Garbage Collection Performance

- Size of Heap
 - Sheer amount of elements to look through
- Complexity Of Heap
 - Tangled webs of references take longer
 - Flat arrays of simple types require no work
 - Your case will be in between
- Rule of thumb: 10ms per Mb of total size
 - 5Mb heap, garbage at 100k/s -> 50ms pause every 10s
 - 50Mb heap, garbage at 1Mb/s -> 500ms pause every 1sec



Sources of Garbage

- String handling
 - Strings in .NET are immutable
 - Creating a new string creates garbage
 - `ToString()`, `String.Format()`
- Boxing
- Using reference types for temporary values
 - Eg in math routines
- `foreach()` loops with certain types
- LINQ extension methods create garbage
 - `IEnumerable`, calls with delegates



Garbage Collection Best Practice

- Keep your heap simple
 - Keep your heap small
 - Work with value types where you can
 - Avoid methods using 'object'
 - Use generics instead - `DoThing<T>(T thing)`
 - Force garbage collection when convenient
 - Start / end of level
 - Game pause
 - Save / load
- ```
GC.Collect();
```



*Low Frame Rate  
When You Try To Give Too Much*



## Being Sure

- Poor frame rate?
- Initial triage on bottleneck
  - Is it the CPU or GPU?
- 2 simple checks
  - 1) Draw scene with Update() disabled
  - 2) Update game with Draw() disabled
- Which is quicker?
  - 1 -> CPU bound
  - 2 -> GPU bound



## Ways Of Measuring

- Instrument your code
  - `System.Diagnostics.Stopwatch`
    - 10m ticks per second
  - `Start()` & `Stop()` around `Update()` and `Draw()`
- FPS counter, timing bar and onscreen counts
  - Included in PerformanceMeasuring sample
  - GameDebugTools also includes neat console
- Windows build of your game



## Windows Build of Your Game

- Easy to do
  - A few clicks
  - Adjust your controls & UI, maybe
- Leverage Windows performance tools
  - Remember relative timings matter, absolute will differ
- Particularly useful tools
  - Visual Studio 2010 Profiler
  - CLRProfiler
  - PIX For Windows



## CPU Performance Bound

- So. It's the CPU, eh.
  - Coarse mechanism...
    - ...yet more `System.Diagnostics.Stopwatch`
      - Be aware that times are distorted with debugger connected
      - Display on screen or log to file or through http port
  - Profile your Windows build
    - Instrumented build profile will give call counts
    - Times will not be accurate
    - Different JIT compiler, different processor, different memory, different \*\*
    - Relative information is still good



## Common CPU Pitfalls

- Property accesses are function calls (remember?)
  - Use public fields in performance critical areas
- For value types:
  - Returning type is slower than out parameter
  - Passing by value is slower than ref
- 2D array access is unpleasantly slow
  - Instead of `a[x,y]` use `a[y][x]` or `a[x + (y * width)]`
- `double` is faster than `float`
  - All internal math is 64 bit, so 32 bit causes conversions



## GPU Performance Bound

- Could my problem be total pixels rendered?
  - Could be heavy use of most expensive effects
  - Could be too much overdraw
  - > Check by dropping resolution
 

```
graphics.PreferredBackBufferWidth = 200;
graphics.PreferredBackBufferHeight = 120;
```
  - Now filling 16 times less pixels
- Hardware scaler takes the strain
  - Is the game faster?
  - Ok, now figure out how to reduce overall pixel cost



### GPU Performance Bound

- In detail – PIX For Windows
  - Free with DirectX SDK
- You can spot:
  - Unnecessary geometry
  - Packing too many texels or vertices into too few pixels
  - Overdraw
  - Unnecessary clears
  - Warnings about slow operations
  - Relative draw call timing
  - Texture size and format selection



### Summary

- Garbage collector
  - Keep heap small and simple
  - Force collect at major game transitions
- Be aware of common performance pitfalls
  - Know when your game is CPU or GPU bound
- Profile on multiple platforms
  - CPU: CLRProfiler, Visual Studio 2010 profiler, timers
  - GPU: PIX for Windows



*Thank you for listening*

Q&A

