Article

Performance Considerations for

Windows Phone 7

By Kris Gray

Contents

[Introduction 3](#_Toc274647355)

[Windows Phone 7 Performance Considerations 3](#_Toc274647356)

[Content Loading 3](#_Toc274647357)

[Memory Management 4](#_Toc274647358)

[Measuring Performance 5](#_Toc274647359)

[Conclusion 6](#_Toc274647360)

# Introduction

This article describes ways to reduce memory, techniques for allocating memory that reduce or eliminate garbage collection, and profiling tools to help identify the memory bottlenecks. It results from practical experience gained by the Advanced Technology Group at Microsoft when analyzing and assisting customers with Windows Phone 7 games.

# Windows Phone 7 Performance Considerations

XNA Game Studio 4.0 supports multiplatform game development using C#. Code generated by Game Studio runs with little or no change on three platforms: Window Phone 7, Windows, and Xbox 360. Opportunities for performance tuning exist, even for Windows Phone.

Windows Phone 7 does not use the same processing power as a Windows PC or an Xbox 360. Why not? Because for a phone:

* The CPU is slower.
* There is less RAM.
* The GPU is less powerful.
* The screen is smaller.
* There are strict power requirements since it has its own battery.

If you have a C++ game development background, you may need to think differently about memory management for a game developed with C# managed code.

A game developed in managed code by using Game Studio runs on the .NET Compact Framework runtime. The framework performs automatic garbage collection, a powerful technique for cleaning up objects in memory that are no longer used. Many C++ developers may be unaccustomed to this approach to memory management. While garbage collection simplifies the process for reclaiming unused memory, there are approaches to writing C# code that makes executing garbage collection on a Windows phone more efficient.

This article identifies two areas in which you may be able to improve performance.

* Content Loading – Load game assets more efficiently.
* Memory Management – Adjust your code development style to avoid unnecessary garbage generation and collection.

# Content Loading

Rich, detailed content is an important part of gameplay. Loading, processing and rendering detailed textures and models, and loading and processing audio affects performance. Here are tips for using game content to improve performance.

* **Load only the content needed by the game.** The smaller the content, the faster loading occurs. Generally, the access data rate (memory bus speed as opposed to the CPU speed) hinders performance and creates a bottleneck.
* **Reduce the size of textures**. A large phone screen resolution typically is 800 x 480. Because only a certain level of detail is seen at this resolution, consider reducing your texture sizes. DXT compressed textures must be sized to a power of two, so combine compressed textures into a sprite sheet to conserve space that otherwise might be wasted.
* **Use DXT to compress textures**. Encoded or compressed content is smaller, sometimes dramatically smaller. This reduces loading time.
* **Load what you can at startup.** At startup, a game usually presents a startup screen. As the screen loads, other content could be loaded or data preprocessed so that gameplay is not impacted.
* **Use a custom processor to load textures**. Converting textures to the native format of 32-bit RGBA used by Game Studio improves texture save and load times. At design time, use a content pipeline custom processor to convert existing textures.
* **Take advantage of idle CPU time.** The CPU often is underutilized while your game awaits user input. Take advantage of the idle time to load content.

# Memory Management

Memory management generally refers to wisely using available memory. Wise usage depends on many things, including how memory is allocated for new objects or how it is reclaimed for objects no longer in use. The intention of good memory management is to use the least amount of memory possible and to reduce memory fragmentation. In general, the less memory you use, the less likely your game will have a performance problem caused by memory management.

Automatic garbage collection is a form of memory management. Garbage collection:

* Attempts to reclaim memory occupied by objects no longer used by the program.
* Works on objects such as textures, buffers, and reference type variables created by your game.
* Does not manage other resources such as network sockets, database handles, user interaction windows, and file and device descriptors.

Game performance can be impacted by automatic garbage collection if it occurs at an inconvenient time, such as during gameplay. Here are some suggestions for designing your game to minimize the need for automatic garbage collection:

* **Reduce the amount of memory needed**. Reduce the number or size of objects. Perhaps you cannot see the detail, or as many objects on the phone that⎯on a larger display⎯look great. Reduce texture sizes to the level of detail needed for the device's resolution.
* **Allocate memory for Reusable Objects**. Allocate a set of reusable objects and reinitialize them as needed. Reusable objects never need to be freed, and reduce the number of additional allocations. This makes garbage collection less frequent or unnecessary. Be careful when using reusable objects; allocating memory for pre-allocated and unused objects wastes memory.
* **Use more value types and fewer reference types**. Value types do not get collected; use a value type for complex cohesive data rather than allocating reference types. Collision data, for example, typically is stored as a tree of reference types. The tree can be optimized dramatically by flattening it into an array of value types, encoding the tree structure as integer indices. Exercise caution when optimizing value types; over optimization can cause problems by burdening the stack with excessive allocations.
* **Allocate memory in chunks smaller than 1 MB**. The garbage collector in the Windows Phone 7 common language runtime (CLR) is nongenerational. It uses a mark-and-sweep algorithm that occurs when allocations exceed 1 MB. This means the garbage collector runs each time you hit the 1 MB mark. During garbage collection, all other threads stop.
* **Reduce the complexity of data structures**. Consider grouping data types in a structure based on their frequency of update. Grouping increases the likelihood that data access is more efficient.
* **Use an array of value types to guarantee objects are located close to each other**. An array of value types is always one contiguous chunk of memory. Conversely, an array of reference types holds a series of pointers to the actual location of the allocated memory. In addition, memory allocated in a batch is not guaranteed to be contiguous because automatic heap compaction can move an object during the life of the process.
* **Manually call GC.Collect() when the timing is right**. There are times⎯ like during a level loading screen⎯when you can run garbage collection without affecting its use. This resets the 1 MB allocation threshold.

# Measuring Performance

There several tools available for analyzing the .NET garbage collection heap. One tool available from Microsoft is the CLR Profiler. The tool is free and recommended for profiling memory. It is available for download at <http://www.microsoft.com/downloads/en/details.aspx?FamilyId=A362781C-3870-43BE-8926-862B40AA0CD0&displaylang=en>.

**To launch the profiler**

1. Download the CLR Profiler, and then extract the files.
2. Navigate to CLRProfiler\Binaries\x86 (or x64), and then run CLRProfiler.exe.
3. Click **Start Application**, and then select your application or game.

The profiler generates a graph that displays information about all objects in the garbage collection heap, and a hierarchy of references related to these structures.

The CLR Profiler does not run on Windows Phone 7, so you must create a Windows version of your game. By using Game Studio tools, this task is easy.

**To create a profile**

1. Open Visual Studio Express.
2. In the solution, add a New Project for Windows.
3. Add the game code to it, and then build it.

Profiling on Windows shows the same heap activity as does profiling on Windows Phone 7. If you have questions, a good way to get answers is through the XNA Community Forums. Here is an example of one such forum that answered questions about using the CLR Profiler with XNA 4.0: <http://forums.xna.com/forums/t/50361.aspx>

As an alternative to the free CLR Profiler, some versions of Visual Studio 2010 (like Premium and Ultimate) include a built-in profiler. The Advanced Technology Group at Microsoft recommends using this profiler because it generates reports that contain more actionable detail, such as the game code that produces the most allocations and the game code that allocated the most memory.

For more information about profiling in versions of Visual Studio 2010 such as Premium and Ultimate, visit

<http://www.nachmore.com/2010/profiling-silverlight-4-with-visual-studio-2010/> and

<http://blogs.msdn.com/b/oren/archive/2010/06/03/profile-silverlight-4-from-the-vs-2010-ui-premium-amp-ultimate-only.aspx>.

# Conclusion

Using a profiler is the only one way to measure performance. The best performance for a Windows Phone 7 game is achieved by following two essential practices:

* Keep overall memory use under 90 MB.
* Minimize or eliminate garbage collection.