PERFORMANCE

- Rene Damm
- Kim Steen Riber

WHO WE ARE

- René Damm
  Core engine developer @ Unity

- Kim Steen Riber
  Core engine lead developer @ Unity

unity

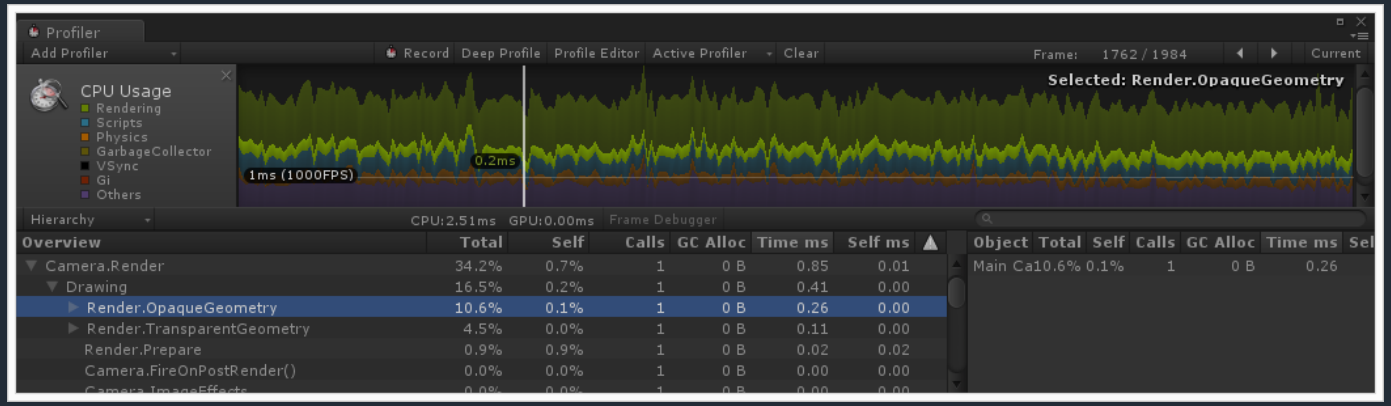
OPTIMIZING YOUR GAME

- FPS
  - CPU (Gamecode, Physics, Skinning, Particles, …)
  - GPU (Drawcalls, Shader usage, Image effects, …)
- Hiccups
  - Spikes in framerate (e.g. GC.Collect, Physics rebuild)
  - Loading
- Memory
  - Maintaining small runtime memory on device
  - Avoid GC Hiccups by reducing memory activity
  - Leak detection

UNITY PROFILER

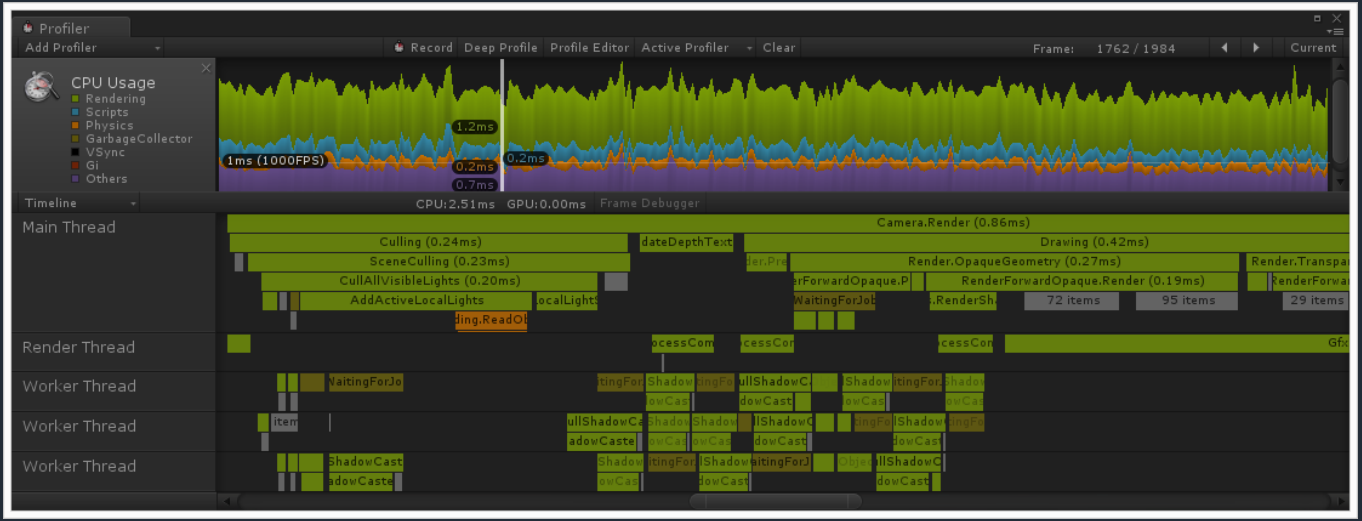
- CPU
  - Hierarchy and Timeline profiler
- GPU
  - Frame debugger
- Audio
- Physics
- Memory usage
  - Detailed memory view
  - Memory reference view

CPU PROFILER



- Cpu time consumption for methods
- Mono memory activity
- Remote Profiling of you game running on target device
- Deep profile (editor only)
  - Large overhead, only usable for very small scenes

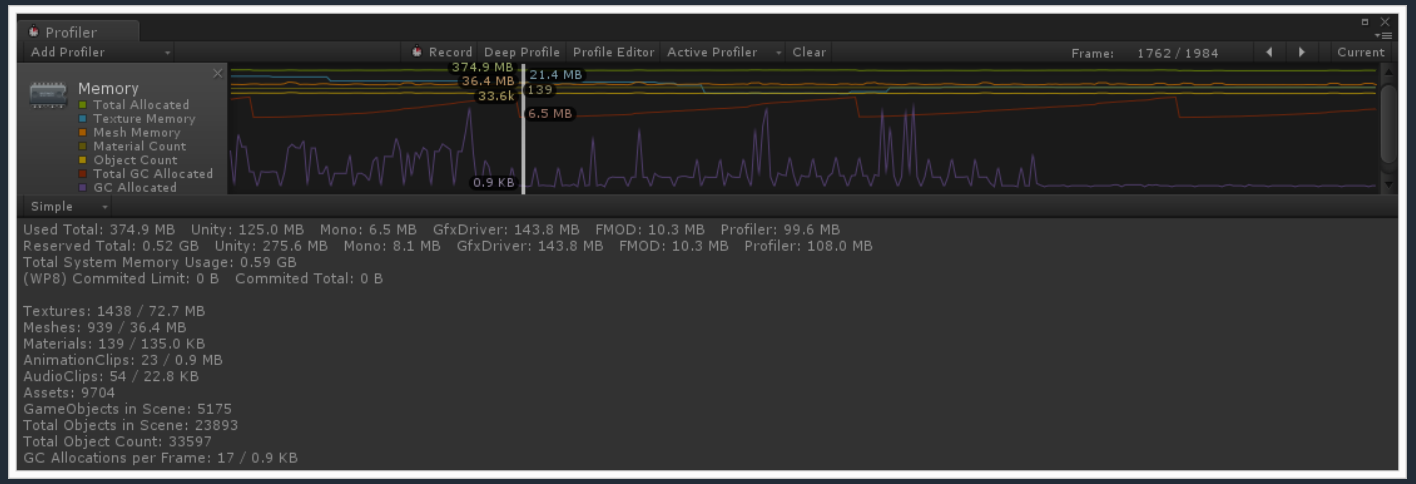
CPU PROFILER - TIMELINE



- Better overview of multithreaded tasks
- Scheduling and stall investigation

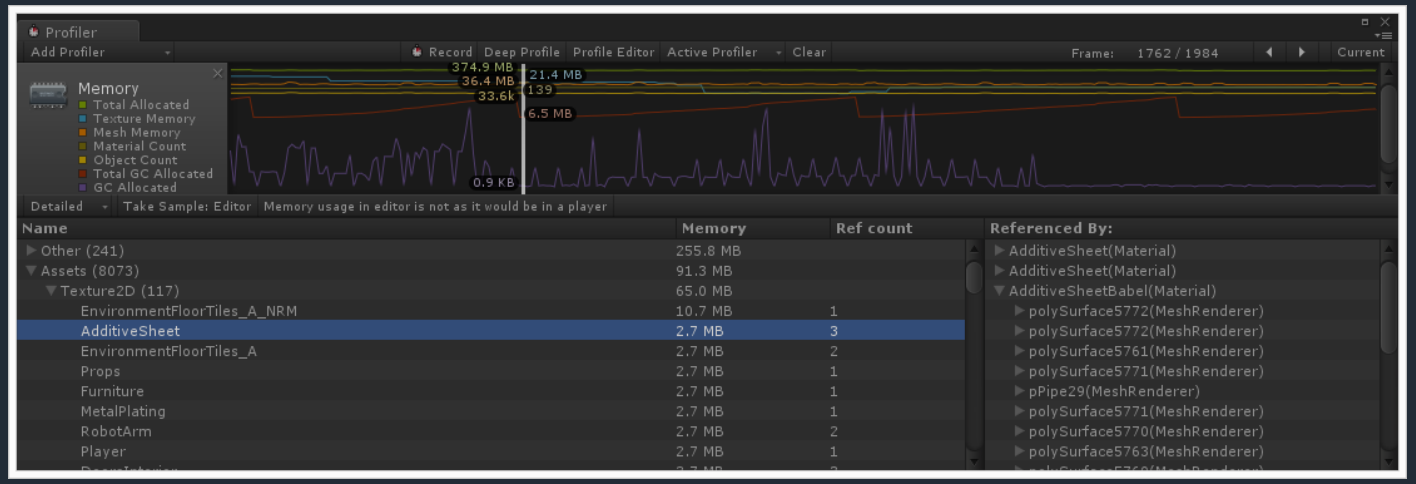
unity

MEMORY PROFILER



- Unity reserves chunks of memory from the os
- Mono allocates from a reserved heap
- GfxMemory is an estimate

MEMORY PROFILER



- Detail view is taken as a snapshot (too expensive for per frame)
- Reference view See where an object is referenced – Good for leak hunting

LOADING.

LOADING..

LOADING...

LOADING DATA

- Kinds of data
  - Scenes
  - Assets
  - Resources, StreamingAssets, WWW, etc.
- Load through Unity or custom
- Load synchronously or asynchronously

LOADING – ASYNCHRONOUSLY

- Application.LoadLevel*Async
- AssetBundle.Load*Async
- 5.0: Resources.LoadAsync
- “Batch” size still matters
  - Bulk activation step on main thread
- 5.0: Textures continue to load in background

LOADING SCENES – MEMORY

- Current scene unloaded only after new scene has been (pre)loaded on background thread
- Assets unloaded only after new scene is entirely loaded
- Solution: Load empty scene before loading “real” scene
  - 5.x: Multi-scene loading API with explicit unloading control

LOADING – SERIALIZATION

- Generally pretty fast
- Slow path if data from older version
- .NET reflection only hit for first object
  - Some platforms use “serialization weaver” (WP8)
- Future
  - More blobified data
  - Memory mapping

LOADING – ASSETBUNDLES

- Optimize player size
- Optimize build time
- Some (small) memory overhead per bundle while loaded
  - Use WWW.LoadFromCacheOrDownload

UNLOADING

- 5.0: Bulk of deletion work moved off main thread
- Managed wrappers stick around until next GC
- Assets stick around until next asset GC
- Place GCs strategically
  - Resources.UnloadUnusedAssets / GC.Collect
- Resources.Unload
- Be mindful of references accumulating in statically referenced objects

GC PERFORMANCE

- Boehm GC scans entire heap and follows all potential references (Mark and sweep)
- Scales with heap usage
- Scattered indirections cause cache thrashing

GC OPTIMIZATIONS

- Reduce the work GC has to do
  - Optimize code by ensuring fewer references
- Reduce the frequency for GC to run
  - Avoid creating garbage

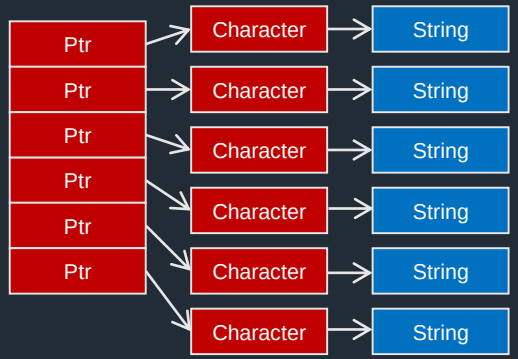
REDUCE GC WORK

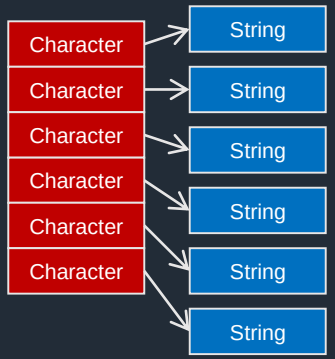
- Pack data with indirections tightly, or remove indirections
  - Structs instead of classes
- Contiguous memory layout for cache friendliness
- Split data - Separate value types from indirection data
  - Float, Vector3, Structs of value types
  - Strings, Classes

EXAMPLE: ARRAY OF CHARACTERS

```
class Character
{
    public String Name;
    public Vector3 Pos;
}
Character[] characters;
```

```
struct Character
{
    public String Name;
    public Vector3 Pos;
}
Character[] characters;
```





unity

RESULTS

- GC for array length of 1.000.000 characters on MBP
  - Class with randomized memory locations: 35ms
  - Class with linear memory: 20ms
  - Structs: 10ms
  - Struct and no String: 0.18ms

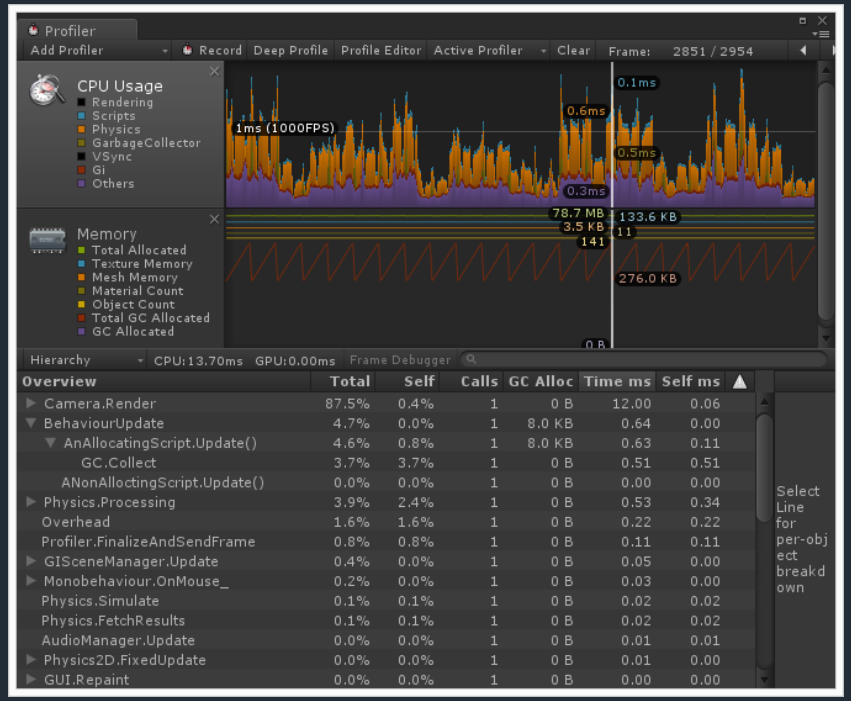
MINIMIZING MANAGED ALLOCATIONS

- Reuse temporary buffers
  - If buffers for data processing are needed every frame, allocate the buffer once and reuse
- Allocate pools of reusable objects
- Use structs instead of classes
  - Structs are placed on the stack, while classes uses the heap
- Don’t use OnGUI - Use the New UI System
  - Even empty OnGUI calls are very memory intensive

AVOIDING ALLOCATIONS

```
public class AnAllocatingScript : MonoBehaviour {
    class WorkData {
        public int value;
    }
    WorkData CreateWorkData() {
        return new WorkData();
    }
    void ProcessWorkData(WorkData data) {
        int[] TempWorkBuffer = new int[2*1024];
        TempWorkBuffer[0] = data.value;
    }
    void Update () {
        ProcessWorkData(CreateWorkData());
    }
}
```

```
public class ANonAllocatingScript : MonoBehaviour {
    struct WorkData {
        public int value;
    }
    WorkData CreateWorkData() {
        return new WorkData();
    }
    int[] TempWorkBuffer = new int[2*1024];
    void ProcessWorkData(WorkData data) {
        TempWorkBuffer[0] = data.value;
    }
    void Update () {
        ProcessWorkData(CreateWorkData());
    }
}
```



unity

POOLING

- Goals
  - Eliminate GC churn
  - Eliminate Instantiate/Destroy overhead
- Problems
  - Still need to instantiate to “warm up” pool
  - Adds overhead to total GC size

POOLING – HOW?

- Avoid data structures that incur GC overhead
- Warm up at load-time, if necessary
- Choose least costly activation
  - SetActive()
  - .enabled for specific components
  - Just move off-screen
- 5.0: Instantiation and activation times greatly reduced

GENERAL PERFORMANCE

- Avoid GameObject.Find and relatives
  - Use direct linkage or managers
  - Retain GameObject and component references
- Avoid excessive managed/native transitions
- Not everything needs to be a MonoBehaviour
- Deep hierarchies are costly

FUTURE

- Empty out main thread
- Further improve streaming
- Native performance and SIMD on all platforms
- Improve performance of transform hierarchy
- More profiling tools

CONCLUSION

- Load asynchronously to reduce wait times
- Be aware of memory usage patterns
- Reuse objects to lower GC pressure
- Cache results to avoid repeated work
- Use the profiler to identify issues

OTHER RESOURCES

- Scripting Behind the Scenes
  - Jonathan Chambers - Unite 2013
- Mobile Performance Poor Man's Tips And Tricks
  - Tautvydas and Zbignev - Unite 2014
- Optimizing unity games
  - Aleksandr Dolbilov - JoyCraft Games - Google IO 2014