

€ unity

Optimizing Mobile Applications





Mark



lan

€ unity

About This Talk

- Getting Good Data
- General Best Practices
- Common Problems & Solutions
 - Memory Usage
 - CPU Performance





Use The Best Tools

- iOS: Instruments
- Android: VTune, Snapdragon Profiler
- Unity Editor
 - Timeline
- 5.3: Memory Profiler

Instruments!

- Free, included with XCode
- Works perfectly with Unity IL2CPP builds
- Best tool for mobile CPU profiling
- Best tool for startup time profiling

Instruments! (2)

• Instructions on how to run it:

 http://blogs.unity3d.com/2016/02/01/profiling-withinstruments/

Instruments CPU Profiler: Startup Time

Oetails	O Details							
Running	Running Time~ Self (ms) Symbol Name							
617.0ms	64.0%	0.0		▼Main Thread 0x13124a				
615.0ms	63.7%	71.0	п	▼main ProductName				
241.0ms	25.0%	0.0	1	w-[UnityAppController(Rendering) repaintDisplayLink] ProductName				
238.0ms	24.6%	0.0	1	▶UnityRepaint ProductName				
3.0ms	0.3%	0.0	1	-[UnityAppController(Rendering) repaint] ProductName				
197.0ms	20.4%	0.0	П	▼-[UnityAppController startUnity:] ProductName				
174.0ms	18.0%	0.0	1	VunityInitApplicationGraphics ProductName				
168.0ms	17.4%	0.0	л	▼PlayerInitEngineGraphics(bool) ProductName				
161.0ms	16.7%	0.0	1	PlayerLoadGlobalManagers(char const*, char const*, unsigned int) ProductName				
7.0ms	0.7%	0.0	л	InitializeEngineGraphics(bool) ProductName				
4.0ms	0.4%	4.0	1	UnityCoreMotionStart ProductName				
2.0ms	0.2%	0.0	1	InitializeGfxDevice(unsigned int) ProductName				
14.0ms	1.4%	1.0	1	-[UnityAppController(ViewHandling) showGameUI] ProductName				
9.0ms	0.9%	0.0	1	► UnityLoadApplication ProductName				
104.0ms	10.7%	5.0	1	▶-[UnityAppController application:didFinishLaunchingWithOptions:] ProductName				
2.0ms	0.2%	0.0	1	▶UnityParseCommandLine ProductName				
2.0ms	0.2%	1.0	1	GLOBAL_sub_l_Notifications.mm ProductName				

Instruments CPU Profiler: Runtime

119.0ms	47.0%	0.0	▼Main Thread 0x13124a				
119.0ms	47.0%	13.0	▼main ProductName				
106.0ms	41.8%	0.0	Tepaint [UnityAppController(Rendering) repaintDisplayLink] ProductName				
106.0ms	41.8%	0.0	VunityRepaint ProductName				
98.0ms	38.7%	0.0	UnityPlayerLoopImpl(bool) ProductName				
98.0ms	38.7%	0.0	PlayerLoop(bool, bool, IHookEvent*) ProductName				
60.0ms	23.7%	0.0	PlayerRender(bool) ProductName				
23.0ms	9.0%	0.0	PhysicsManager::FixedUpdate() ProductName				
8.0ms	3.1%	0.0	PlayerConnection::PollListenMode() ProductName				
2.0ms	0.7%	0.0	EnlightenRuntimeManager::Update() ProductName				
2.0ms	0.7%	0.0	AudioModule::Update() ProductName				
1.0ms	0.3%	0.0	PreloadManager::UpdatePreloading() ProductName				
1.0ms	0.3%	0.0	PlayerSendFrameComplete() ProductName				
1.0ms	0.3%	1.0	GetDelayedCallManager() ProductName				
4.0ms	1.5%	0.0	UnityUpdateJoystickData ProductName				
2.0ms	0.7%	0.0	metal::StartFrame(RenderSurfaceBase*, RenderSurfaceBase*) ProductName				
2.0ms	0.7%	2.0	UnityEndFrame ProductName				

Instruments: Reading the PlayerLoop

- PlayerRender
 - Draw calls, batching, <u>OnWillRender</u> & image effect callbacks

- BaseBehaviourManager::CommonUpdate¹
 - <u>Update</u>, <u>FixedUpdate</u> and <u>LateUpdate</u> callbacks

Instruments: Reading the PlayerLoop (2)

- Ul::CanvasManager (several methods)
 - UI canvas rebatching, text mesh generation, etc.
- DelayedCallManager::Update
 - Resumed coroutines
- PhysicsManager::FixedUpdate
 - PhysX simulation, <u>OnCollision*</u> and <u>OnTrigger*</u> callbacks

Instruments: Examining a Callback

3536.0ms	12.8%	0.0	▼Main Thread 0x13aee5				
3535.0ms	12.8%	635.0 🗖	▼main sampleassets				
2870.0ms	10.4%	10.0 👤	v-[UnityAppController(Rendering) repaintDisplayLink] sampleassets				
2860.0ms	10.3%	116.0 🗖	▼UnityRepaint sampleassets				
2744.0ms	9.9%	0.0 🗖	VUnityPlayerLoopImpl(bool) sampleassets				
2744.0ms	9.9%	20.0 🗖	PlayerLoop(bool, bool, IHookEvent*) sampleassets				
1662.0ms	6.0%	4.0 🗖	void BaseBehaviourManager::CommonUpdate <behaviourmanager>() sampleassets</behaviourmanager>				
1649.0ms	5.9%	0.0 🗖	▼UpdateBehaviour sampleassets				
1642.0ms	5.9%	0.0 🗖	MonoBehaviour::CallUpdateMethod(int) sampleassets				
1636.0ms	5.9%	4.0 🗖	TallMethodIfAvailable sampleassets				
1630.0ms	5.9%	0.0 🗖	ScriptingInvocationNoArgs::Invoke() sampleasets				
1630.0ms	5.9%	2.0 🗖	ScriptingInvocationNoArgs::Invoke(ScriptingException**) sampleassets				
1623.0ms	5.8%	1.0 🗖	▼0xc6f744 sampleassets				
1616.0ms	5.8%	1.0 🗖	RuntimeInvoker_Void_t2779279689(MethodInfo const*, void*, void**) sampleassets				
1350.0ms	4.8%	6.0 🗖	EventSystem_Update_m242895889 sampleassets				
74.0ms	0.2%	3.0 🗖	WaypointProgressTracker_Update_m3046203301 sampleassets				
72.0ms	0.2%	4.0 🗖	JetParticleEffect_Update_m2014536237 sampleassets				
56.0ms	0.2%	2.0 🗖	AeroplaneAudio_Update_m3409313906 sampleassets				
39.0ms	0.1%	1.0 🗖	AeroplaneControlSurfaceAnimator_Update_m2151891663 sampleassets				
20.0ms	0.0%	1.0 🗖	LandingGear_Update_m658550403 sampleassets				
3.0ms	0.0%	0.0 💶	▶Platformer2DUserControl_Update_m3525060864 sampleassets				
1.0ms	0.0%	1.0 🗖	LevelReset_Update_m1325509725 sampleassets				

€unity

Instruments: Examining a Coroutine

O Details	$\rangle \equiv$	Call Tree		Call Tree	Or delayed ∅				
Running	Time≁	Self (ms)		Symbol Nar	ne				
366.0ms	16.3%	0.0			ead 0x1324a0				
366.0ms	16.3%	340.0	л	▼main	sampleassets				
26.0ms	1.1%	1.0		▼-[Unit	yAppController(Rendering) repaintDisplayLink] sampleassets				
25.0ms	1.1%	15.0	п	▼Unit	tyRepaint sampleassets				
10.0ms	0.4%	0.0		∀U	nityPlayerLoopImpl(bool) sampleassets				
10.0ms	0.4%	3.0		V	PlayerLoop(bool, bool, IHookEvent*) sampleassets				
5.0ms	0.2%	1.0	п		DelayedCallManager::Update(int) sampleassets				
3.0ms	0.1%	0.0	п		▼Coroutine::Run() sampleassets				
2.0ms	0.0%	0.0	П		Coroutine::InvokeMoveNext(ScriptingException**) sampleassets				
2.0ms	0.0%	0.0			▼bool ScriptingInvocation::Invoke <bool>(ScriptingException**, bool) sampleassets</bool>				
2.0ms	0.0%	0.0			▼ScriptingInvocation::Invoke(ScriptingException**, bool) sampleassets				
2.0ms	0.0%	0.0	П		▼0xc47744 sampleassets				
2.0ms	0.0%	0.0	1		RuntimeInvoker_Boolean_t211005341(MethodInfo const*, void*, void**) sampleas				
2.0ms	0.0%	0.0			U3CDoBobCycleU3Ec_Iterator6_MoveNext_m405508410 sampleassets				
1.0ms	0.0%	0.0	п		▶Coroutine::ProcessCoroutineCurrent() sampleassets				
1.0ms	0.0%	1.0	П		GetTimeManager() sampleassets				
2.0ms	0.0%	0.0	1		PlayerSendFrameComplete() sampleassets				

Instruments: Coroutines (2)

- Coroutine execution is split between two places:
 - The method where the coroutine was started.
 - i.e. where StartCoroutine() was called
 - DelayedCallManager
- StartCoroutine runs all code until the first yield
- DelayedCalledManager runs the rest

Instruments: Identifying Asset Loads

▼void BaseBehaviourManager::CommonUpdate <behaviourmanager>() sampleassets</behaviourmanager>
VUpdateBehaviour sampleassets
WMonoBehaviour::CallUpdateMethod(int) sampleassets
TallMethodifAvailable sampleassets
ScriptingInvocationNoArgs::Invoke() sampleasets
ScriptingInvocationNoArgs::Invoke(ScriptingException**) sampleassets
▼0xc47744 sampleassets
RuntimeInvoker_Void_t2779279689(MethodInfo const*, void*, void**) sampleassets
ParticleSceneControls_Update_m2783660318 sampleassets
VObject_Instantiate_m2255090103 sampleassets
Object_Internal_InstantiateSingle_m3047563925 sampleassets
VObject_INTERNAL_CALL_Internal_InstantiateSingle_m1201424140 sampleassets
Volject_CUSTOM_INTERNAL_CALL_Internal_InstantiateSingle(ReadOnlyScriptingObjectOfType <object>, Vector3f const</object>
▼InstantiateObject(Object&, Vector3f const&, Quaternionf const&) sampleassets
InstantiateObject(Object&, Vector3f const&, Quaternionf const&, vector_map <int, int,="" std::1::less<int="">, stl_allocate</int,>
CloneObjectImpl(Object*, vector_map <int, int,="" std::1::less<int="">, stl_allocator<std::1::pair<int, int="">, (MemLabel)</std::1::pair<int,></int,>
▼ParticleSystem::VirtualRedirectTransfer(StreamedBinaryRead <false>&) sampleassets</false>
void ParticleSystem::Transfer <streamedbinaryread<false> >(StreamedBinaryRead<false>&) sampleassets 4</false></streamedbinaryread<false>



5.3 Memory Profiler

demoryProfilers			•	••
			Load Snapshot	
Mesh 7.7 Ma	Forc 0.7 MB RenderTexture 3.8 MB	Cobernago Andrea Audio Manager 1.5 M8		
Techare20 47.2 MB				

5.3 Memory Profiler

- Download code from Bitbucket
 - <u>http://bitbucket.org/Unity-Technologies/MemoryProfiler/</u>
- Drop into an *Editor* folder inside Assets
- In Unity Editor: Window > MemoryProfilerWindow
- Connect Unity Profiler via Profiler Window
- Click "Take Snapshot"

5.3 Memory Profiler: Duplicated Textures



€unity

5.3 Memory Profiler: Duplicated Textures



Same Texture, Different Instances



5.3 Memory Profiler

				•••
Texture2D 1.2 MB	Mesh AudioManager 0.8 M8	Mach these Gener AudioManager 0.8 MB 0.5 MB		
RenderTexture 3.8 MB	Render Texture 1.0 MB	Render/Texture	is DontDestroyOnLoad False is Persistent False is Manager False hidef Tags HideAndDontSave hidef Tags 4002000 References: Referenced by: This is a root because: the DontDebatTesisedAsset hidefTag is set on M builtin resources set this flag. Users can also set	False False HideAndDontSave 4002000 Asset hideflag is set on this object. Unity's 5 Rog. User's Can also set the flag themselves





Asset Auditing: Preventing Mistakes

- Developers are people (arguably)
- People make mistakes
- Mistakes cost dev time

• Write tools to prevent common, costly errors

Asset Auditing: Common Errors

- Insane texture sizes
- Asset compression
- Improper Avatar/Rig settings

• Different rules for different parts of project

Asset Auditing: HOWTO

public class AssetAuditExample : AssetPostprocessor {

```
public void OnPreprocessTexture() {
    // ...
}
```

```
public void OnPreprocessModel() {
    // ...
}
```

Asset Auditing: HOWTO (2)

• AssetPostprocessor classes receive callbacks on import

• Implement *OnPreprocess** methods

• Apply your project's rules to *assetImporter* instance

Asset Auditing: HOWTO (3)

public class ReadOnlyModelPostprocessor : AssetPostprocessor {

public void OnPreprocessModel() {
 ModelImporter modelImporter = (ModelImporter)assetImporter;
 if(modelImporter.isReadable) {
 modelImporter.isReadable = false;
 modelImporter.SaveAndReimport();
 }
}

Common Rules: Textures

- Make sure Read/Write is disabled
- Disable mipmaps if possible
- Make sure textures are Compressed
- Ensure sizes aren't too large
 - 2048x2048 or 1024x1024 for UI atlases
 - 512x512 or smaller for model textures

Common Rules: Models

- Make sure Read/Write is disabled
- Disable rig on non-character models
- Copy avatars for characters with shared rigs
- Enable mesh compression

Side Note: Mesh Renderer

Do you need these on?

- Mesh Renderer
 Cast Shadows
 Receive Shadows
 Materials
 Use Light Probes
 Reflection Probes
 - Anchor Override

		0	۵,
	On		
	\checkmark		
	Blend Probes		
	None (Transform)		0

Common Rules: Audio

- MP3 compression on iOS
- Vorbis compression on Android
- "Force Mono" for mobile games
- Set bitrate as low as possible

Memory in Unity

€ unity



🚭 unity

More memory is allocated when requested by code. int[] someNumbers = new int[2048];



Garbage collector runs periodically, looks for unused objects. Unused objects are deleted.

GC.Collect();

int[] Array	int[]	int[] Arra	ay int[] Array	int]] Array	
List	float[] a	array	string		string

Holes are not filled. This is Memory Fragmentation.



€unity
Managed Memory: How It Works

When there isn't enough space for new objects...



Managed Memory: How It Works

The heap expands.

int[] Array

int[] Array	int[]			int[] Array	
List	float[] ai	rray	string		string

Managed Memory: Problems

• In Unity, the heap <u>only</u> expands. It <u>never</u> shrinks.

• iOS & Android still care about reserved pages.

• Detail: Unused blocks of the heap remain reserved, but are paged out of the working set.

Managed Memory: Problems (2)

- Temporary memory allocations are <u>really</u> bad.
- 1 kilobyte of allocation per frame, 60 FPS
 - = 60 kilobytes per second of allocation
- If GC runs once per minute (BAD for framerate)...
- <u>3600 kilobytes of memory needed!</u>

Tracking Managed Memory Allocations

Use Unity Profiler. Sort by "GC Alloc" column.

When user can interact with app, stay as close to zero as possible.

(During loading, allocations aren't as bad.)

	CPU:1.02ms	GPU:0.00r is	Frame Debu
erview		Total	GC Alloc
BehaviourUpdate		6.4%	368 B
ParticleSceneControls.Update()		0.8%	368 B
AutoMoveAndRotate.Update()		1.3%	0 B
EventSystem.Update()		2.8%	0 B
LevelReset.Update()		0.0%	0 B
PlatformSpecificContent.Update()		0.0%	0 B
GameView.GetMainGameViewRenderRect		1.7%	32 B
AudioManager.Update		0.9%	0 B
Camera.Render		29.7%	0 B
Canvas.RenderOverlays		0.7%	0 B
Canvas.SendWillRenderCanvases()		0.6%	0 B
Cleanup Unused Cached Data		0.0%	0 B
GlobalEventQueue		0.0%	0 B
GUI.Repaint		0.2%	0 B
GUIUtility.SetSkin()		0.6%	0 B
HandleUtility.SetViewInfo()		0.0%	0 B

Memory Conservation

- Reuse Collections (Lists, HashSets, etc.)
- Avoid string concatenation
 - Reuse StringBuilders to compose strings
- Avoid closures & anonymous methods

Memory Conservation: Boxing

- Happens when passing a value type as a reference type.
- Value is temporarily allocated on the heap.

Example: int x = 1; object y = new object(); y.Equals(x); // Boxes "x" onto the heap

Memory Conservation: Boxing (2)

- Also happens when using enums as Dictionary keys
- Example: enum MyEnum { a, b, c }; var myDictionary = new Dictionary<MyEnum, object>();

myDictionary.Add(MyEnum.a, new object()); // Boxes value "MyEnum.a"

• Workaround: Implement IEqualityComparer class

Memory Conservation: Foreach

- Allocates a Enumerator when loop begins
- Specific to Unity's version of Mono

• Just don't use it.

Memory Conservation: Unity APIs

• If a Unity API returns an array, it allocates a new copy.

• Every time it is accessed.

• Even if the values do not change.



Memory Conservation: Unity APIs (2)

• This code allocates many Touch[] arrays.

```
for ( int i = 0; i < Input.touches.Length; i++ )
{
   Touch touch = Input.touches[i];
   // ...</pre>
```

Memory Conservation: Unity APIs (3)

• This code allocates only one copy of the Touch[] array.

```
Touch[] touches = Input.touches;
for ( int i = 0; i < touches.Length; i++ )
{
Touch touch = touches[i];
// ...
}
```

Memory Conservation: Unity APIs (4)

• Some APIs have allocationless versions.

```
int touchCount = Input.touchCount;
for ( int i = 0; i < touchCount; i++ )
{
   Touch touch = Input.GetTouch(i);
   // ...
```

CPU Performance Tips: Loading



XML, JSON & other text formats

- Parsing text is very slow.
- Avoid parsers built on Reflection extremely slow.
 - In 5.3: Use Unity's <u>JsonUtility</u> class!

• Three strategies to speed up data parsing.

XML/JSON: Reduce Workload

- Strategy 1: Don't parse text.
- Bake text data to binary
 - Use ScriptableObject

- Useful for data that does not change often.
 - e.g. Game design parameters

XML/JSON: Reduce Workload (2)

• Strategy 2: Do less work.

- Split data into smaller chunks.
- Parse only the parts that are needed.
- Cache parsing results for later reuse.

XML/JSON: Reduce Workload (3)

• Strategy 3: Threads.

- Pure C# types only.
- No Unity Assets (ScriptableObjects, Textures, etc.)
- Be VERY careful.

The Resources Folder

- An index of Resources is loaded at startup.
- Cannot be avoided or deferred.

• Solution: Move assets from Resources to Asset Bundles.

CPU Performance Tips: Runtime



Easy: Material/Animator/Shader Properties

- Never address Material, Shader, or Animator properties by name.
- Internally, hashes the property name into an integer.
- Don't do this: material.SetColor("_Color", Color.white); animator.SetTrigger("attack");

Cached Material/Animator/Shader Properties

• Do hashing at startup, cache results and reuse them.

static readonly int material_Color = Shader.PropertyToID("_Color");
static readonly int anim_Attack = Animator.StringToHash("attack");

material.SetColor(material_Color, Color.white);
animator.SetTrigger(anim_Attack);



• So irritating we had to mention it twice.

• Instruments: Search for "::Box"

Instruments: Identifying Boxing

4886.0ms	11.1%	0.0	∀Main Thread 0x1324a0			
4886.0ms	11.1%	2886.0	▼main sampleassets			
1997.0ms	4.5%	12.0	▼-[UnityAppController(Rendering) repaintDisplayLink] sampleassets			
1985.0ms	4.5%	288.0	∀UnityRepaint sampleassets			
1697.0ms	3.8%	2.0	▼UnityPlayerLoopImpl(bool) sampleassets			
1695.0ms	3.8%	30.0 🖡	PlayerLoop(bool, bool, IHookEvent*) sampleassets			
1016.0ms	2.3%	5.0 🗾	PlayerRender(bool) sampleassets			
575.0ms	1.3%	23.0 🗾	PhysicsManager::FixedUpdate() sampleassets			
35.0ms	0.0%	1.0	void BaseBehaviourManager::CommonUpdate <fixedbehaviourmanager>() sampleassets</fixedbehaviourmanager>			
26.0ms	0.0%	2.0	UI::CanvasManager::WillRenderCanvases() sampleassets			
8.0ms	0.0%	5.0	▼DelayedCallManager::Update(int) sampleassets			
3.0ms	0.0%	0.0 🗾	Coroutine::Run() sampleassets			
3.0ms	0.0%	0.0 🗾	Coroutine::InvokeMoveNext(ScriptingException**) sampleassets			
3.0ms	0.0%	0.0 🗾	vbool ScriptingInvocation::Invoke <bool>(ScriptingException**, bool) sampleassets</bool>			
3.0ms	0.0%	0.0	ScriptingInvocation::Invoke(ScriptingException**, bool) sampleassets			
3.0ms	0.0%	0.0 🗾	T0xc47744 sampleassets			
3.0ms	0.0%	1.0	RuntimeInvoker_Boolean_t211005341(MethodInfo const*, void*, void**) sampleas			
1.0ms	0.0%	0.0 🗾	Box(TypeInfo*, void*) sampleassets			
1.0ms	0.0%	0.0	>U3CStartU3Ec_Iterator0_MoveNext_m916801450_gshared sampleassets			
5.0ms	0.0%	4.0	PreloadManager::UpdatePreloading() sampleassets			
3.0ms	0.0%	0.0	►-[UnityAppController startUnity:] sampleassets			
153.0ms	0.3%	0.0	▶MemoryManager::ThreadInitialize 0x1324cc			





"

Some people, when confronted with a problem, think "I know, I'll use regular expressions."

Now they have two problems.

"

- JWZ

Regular Expressions

• Avoid RegExps whenever possible

• RegExps generate *tons* of temporary garbage

• Regex.Match("foo", "bar") = <u>5 kilobytes</u>

Precompile the RegExps

- Don't: Regex.Match(myString, "foo");
 - 5 kilobyte allocation

- Do:
 - var myRegExp = new Regex("foo"); myRegExp.Match(myString);
 - 320 byte allocation

String Comparisons

- Locale coercion is slow
 - 'e' matches the 'æ' ligature in the en-US culture
 - String.Equals("encyclopedia", "encyclopædia")

- Ordinal comparison compares bytes
 - myString.Equals(otherString, StringComparison.Ordinal);

The String Library is Just Slow

- Do not use String.StartsWith, String.EndsWith
- 10-100x slower than a hand-coded bytewise replacement

• Why?



€junity

A Tale of Two Calls

int Accum { get; set; }
Accum = 0;

int accum = 0; int len = myList.Count;





THERE ARE FOUR LIGHTS!!!

€unity

```
int accum = 0;
int len = myList.Count;
                             List::get_Value
for(int i = 0;
   i < len;</pre>
   i++) {
 accum += myList[i];
}
```
Timings for 100,000 Iterations

```
int Accum { get; set; }
Accum = 0;
for(int i = 0;
    i < myList.Count;
    i++) {
    Accum += myList[i];
}</pre>
```

324 milliseconds

128 milliseconds

Remember...

• Unity does minimal, if any, inlining.

- A call in the source is a callvirt in the IL.
- A callvirt in the IL is probably a call in the binary.

• Property accessors are <u>always</u> method calls.

Trivial properties are bad news

- Vector3.zero
 - get { return new Vector3(0,0,0); }
- Quaternion.identity
 - get { return new Quaternion(0, 0, 0, 1); }
- For simple types: make a **const**
- For complex types: make a **static readonly**

So are trivial methods.

- Quaternion.Set, Vector3.Scale
 - Do the math and assign the variables.

- Transform.Translate, Transform.Rotate?
 - Assign the position/rotation.

Remember...

Profile before optimizing.

• Apply these techniques only when needed.

Thanks for listening!

