Gamefest
MICROSOFT GAME TECHNOLOGY CONFERENCE 2007
Take it to the Next Level
Understanding XNA Framework Performance

Shawn Hargreaves
Software Development Engineer
XNA Community Game Platform
Microsoft
Contents

Graphics
- Offload to the GPU
- Understand Xbox 360 system calls
- SpriteBatch, Effects, Renderstates

Math

Multithreading

Profiling tools
OFFLOAD TO THE GPU
The GPU Is a Powerful Beastie

- Offload tasks from CPU to GPU
- Consider GPU instancing
- Particle 3D sample ([http://creators.xna.com](http://creators.xna.com))
  - GPU effect with low CPU overhead
UNDERSTAND SYSTEM CALLS
User programs cannot directly access hardware

Windows Architecture

Operating System (supervisor mode) -> Graphics Driver -> Graphics Hardware

Game Executable (user mode) -> D3D -> D3DX
Xbox Architecture

Consoles typically just run everything directly in supervisor mode

- No mode transitions = reduced overhead
- Small batches less expensive than on Windows
Xbox 360 Architecture

Xbox 360 hypervisor enforces security

- Hypervisor ensures only signed memory pages can execute
- Games are signed during certification

If only signed code can execute, how is a dynamically jitted runtime even possible?
Managed code cannot directly call D3D or D3DX.

User to supervisor transitions are expensive:
- 4 microseconds per system call

Command buffer batches up API calls.
Batchable APIs

*These APIs are currently batched into a single system call*

Assigning to:
- VertexShader
- PixelShader
- VertexDeclaration
- IndexBuffer
- RenderState
- SamplerStates
- Textures
- DepthStencilBuffer
- Viewport
- ScissorRectangle
- ClipPlanes
- Effect.CurrentTechnique

Calling:
- Effect Begin/End
- EffectPass Begin/End
- Effect.CommitChanges
- EffectParameter.SetValue
- VertexStream.SetSource
- Set*ShaderConstant
- StateBlock Capture/Apply
- SetRenderTarget
- Draw[Indexed]Primitives
- DrawUser[Indexed]Primitives
  - *If the primitive count is small*
- Clear
- Resolve
Nasty Unbatchable APIs

These APIs currently require one system call each

- Present
- Creating or destroying graphics resources
- *.SetData, *.GetData
- DrawUser[Indexed]Primitives
  - If the primitive count is large
- Reading from:
  - VertexShader
  - PixelShader
  - RenderState
  - SamplerStates
  - Textures
  - Get*ShaderConstant
  - EffectParameter.GetValue
Cached Managed State

These can be read without any system call at all

- DisplayMode
- Viewport
- VertexDeclaration
- VertexStream
- IndexBuffer
- Effect.CurrentTechnique
SPRITEBATCH, EFFECTS, RENDERSTATES
## Speedy Sprites

*SpriteBatch is well optimized*

- Draw many sprites inside one Begin/End pair
- If possible, use SpriteSortMode.Immediate
  - Draw in texture order
  - Use sprite sheets to combine multiple tiles or animation frames into a single texture
- Otherwise, use SpriteSortMode.Texture

<table>
<thead>
<tr>
<th>Description</th>
<th>SpriteSortMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 batches, one sprite in each (<em>please don’t do this!</em>!)</td>
<td>Immediate: 34 ms, Deferred: 34 ms, Texture: 34 ms</td>
</tr>
<tr>
<td>One batch, 1000 sprites, all using the same texture</td>
<td>Immediate: 0.6 ms, Deferred: 0.7 ms, Texture: 1.8 ms</td>
</tr>
<tr>
<td>One batch, 1000 sprites, alternating between two different textures</td>
<td>Immediate: 11.5 ms, Deferred: 11.6 ms, Texture: 1.9 ms</td>
</tr>
</tbody>
</table>
Efficient Effects

Two different ways to think about effects

- **Effect = shader**
  - One effect instance per shader algorithm
  - Material parameters are stored elsewhere

- **Effect = material**
  - One effect instance per unique material, created from an original archetype effect using `Effect.Clone`
  - Material parameters are stored directly inside the effect
  - The Content Pipeline does this by default

<table>
<thead>
<tr>
<th>Operation</th>
<th>Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin/End on a cloned instance of BasicEffect</td>
<td>9.2</td>
</tr>
<tr>
<td>Using <code>EffectParameter.SetValue</code> and <code>CommitChanges</code> to update a shared BasicEffect instance</td>
<td>19.2</td>
</tr>
<tr>
<td>Using <code>EffectParameterBlock.Apply</code> to update a shared BasicEffect instance</td>
<td>19.6</td>
</tr>
</tbody>
</table>
Rapid Renderstates

- Assigning directly to managed state properties is fastest
- Using dummy effect passes to manage state can be convenient, but not faster
- State blocks are particularly slow on Xbox 360
  - Do not specify `SaveStateMode.SaveState` when calling `SpriteBatch.Begin` or `Effect.Begin`

<table>
<thead>
<tr>
<th>Method</th>
<th>Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assigning directly to renderstates</td>
<td>4.8</td>
</tr>
<tr>
<td>Using a dummy effect pass</td>
<td>5.4</td>
</tr>
<tr>
<td>Using a <code>StateBlock</code></td>
<td>34.5</td>
</tr>
</tbody>
</table>
Math Performance

Simple things you can do to help the JIT
- Pass vector + matrix arguments by reference
- Manually inline performance critical routines

But these optimizations:
- Can affect readability
- May not be necessary in the future
Math Performance

A particular example

class Particle
{
    public Vector3 Position;
    public Vector3 Velocity;

    const float Friction = 0.9f;

    public void Update()
    {
        Position += Velocity;
        Velocity *= Friction;
    }
}

- Updates per second: 3380000
Math Performance
Pass structures by reference

public void Update()
{
    Position += Velocity;
    Velocity *= Friction;

    Vector3.Add(ref Position, ref Velocity, out Position);
    Vector3.Multiply(ref Velocity, Friction, out Velocity);
}

Updates per second: 5540000 (x1.6)
Math Performance
Manually inline computations

public void Update()
{
    Position += Velocity;
    Velocity *= Friction;

    Position.X += Velocity.X;
    Position.Y += Velocity.Y;
    Position.Z += Velocity.Z;

    Velocity.X *= Friction;
    Velocity.Y *= Friction;
    Velocity.Z *= Friction;
}

Updates per second: 12840000 (x3.8)
Math Performance

- XNA Framework math library is heavily inlined
  
  ```csharp
  // These alternatives perform identically
  Position = Vector3.Add(Position, Velocity);
  Position += Velocity;
  ```

- Constructors can be manually inlined
  
  ```csharp
  Position = new Vector3(23, 42, -1);
  Position = new Vector3();
  Position.X = 23;
  Position.Y = 42;
  Position.Z = -1;
  ```
MULTITHREADING
Run, Thread, Run!

- Xbox 360 has three independent CPU cores
  - CPU horsepower is idle if you have fewer than three parallel threads

- Xbox 360 does not automatically schedule threads across multiple cores
  - You must explicitly assign threads to cores
  - Current Xbox 360 ThreadPool is not optimized
Reentering the Framework

- **GraphicsDevice is somewhat thread-safe**
  - Cannot render from more than one thread at a time
  - Can create resources and SetData while another thread renders

- **ContentManager is not thread-safe**
  - Ok to have multiple instances, but only one per thread

- **Input is not threadable**
  - Windows games must read input on the main game thread

- **Audio and networking are thread-safe**
PROFILING TOOLS
Profiling on Xbox 360

XNA Framework Remote Performance Monitor for Xbox 360

- Provides basic garbage collector information
- Can tell if you have a GC problem, but not usually enough to diagnose the cause
- Shows the number of system calls
- Not much help for identifying computational bottlenecks
# Profiling on Xbox 360

![XNA Framework Remote Performance Monitor for Xbox 360](image)

## GC

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Value</th>
<th>Delta</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes Collected By GC</td>
<td>38716504</td>
<td>1093480</td>
<td></td>
<td>The count of bytes collected by the Garbage Collector.</td>
</tr>
<tr>
<td>Garbage Collections (GC)</td>
<td>36</td>
<td>1</td>
<td></td>
<td>The number of times the Garbage Collector has run.</td>
</tr>
<tr>
<td>GC Compactions</td>
<td>1</td>
<td>0</td>
<td></td>
<td>The number of times the Garbage Collector has compacted the heap.</td>
</tr>
<tr>
<td>Managed Bytes In Use After GC</td>
<td>559896</td>
<td>-1296</td>
<td></td>
<td>The number of live objects after the last Garbage Collection.</td>
</tr>
<tr>
<td>Managed Bytes Allocated</td>
<td>39461816</td>
<td>256200</td>
<td></td>
<td>The count of bytes allocated by the Garbage Collector.</td>
</tr>
<tr>
<td>Managed Objects Allocated</td>
<td>2106407</td>
<td>14030</td>
<td></td>
<td>The count of objects allocated by the Garbage Collector.</td>
</tr>
<tr>
<td>Bytes of String Objects Allocated</td>
<td>456768</td>
<td>1952</td>
<td></td>
<td>The count of bytes of string objects allocated by the Garbage Collector.</td>
</tr>
<tr>
<td>Managed String Objects Allocated</td>
<td>24998</td>
<td>183</td>
<td></td>
<td>The number of managed string objects allocated by the Garbage Collector.</td>
</tr>
<tr>
<td>Code Pitchings</td>
<td>0</td>
<td>0</td>
<td></td>
<td>The number of times the Garbage Collect has pitched JIT compiled code.</td>
</tr>
<tr>
<td>Objects Finalized</td>
<td>3</td>
<td>0</td>
<td></td>
<td>The count of objects for which a finalizer have been run.</td>
</tr>
<tr>
<td>Objects Not Moved by Compactor</td>
<td>50</td>
<td>0</td>
<td></td>
<td>The count of the objects that could not be moved by the Garbage Collector.</td>
</tr>
<tr>
<td>Boxed Value Types</td>
<td>2024573</td>
<td>13481</td>
<td></td>
<td>The number of value types that have been boxed.</td>
</tr>
<tr>
<td>Objects on Finalizer Queue</td>
<td>0</td>
<td>0</td>
<td></td>
<td>The number of objects on the finalizer Queue.</td>
</tr>
<tr>
<td>GC Latency Time (ms)</td>
<td>119</td>
<td>3</td>
<td></td>
<td>The total time (in milliseconds) that the Garbage Collector has taken.</td>
</tr>
<tr>
<td>Calls to GC.Collect</td>
<td>0</td>
<td>0</td>
<td></td>
<td>The number of times the application has called the GC.Collect() method.</td>
</tr>
<tr>
<td>Objects Moved by Compactor</td>
<td>1723</td>
<td>0</td>
<td></td>
<td>The count of objects moved by the Garbage Collector during a compaction.</td>
</tr>
<tr>
<td>Pinned Objects</td>
<td>9</td>
<td>0</td>
<td></td>
<td>The count of pinned objects encountered while performing a Garbage Collect.</td>
</tr>
</tbody>
</table>

## Generics

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Value</th>
<th>Delta</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed Methods Loaded</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>The count of unique generic methods that have been loaded across the session.</td>
</tr>
<tr>
<td>Closed Methods Loaded per Definition</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>The maximum number of unique generic methods created for a given type.</td>
</tr>
</tbody>
</table>
Profiling on Windows

- Inference to the rescue!
  - The XNA Framework is similar enough on both platforms that measurements taken on Windows are also applicable to the Xbox 360 version of your game.

- There are many great Windows perf tools
  - The CLR Profiler for garbage collection issues
  - Sampling profilers: Visual Studio Team System, ANTS, NProf, Optimizeit, VTune
Profiling on Windows
CLR Profiler for the .NET Framework 2.0
Recommendations

Graphics
- Offload to the GPU
- Understand Xbox 360 system calls
- Choose an appropriate SpriteSortMode
- Avoid StateBlock

Optimize math where necessary
Take advantage of multiple threads
Profile on both Xbox and Windows
Gamefest 2007
MICROSOFT GAME TECHNOLOGY CONFERENCE

http://www.xna.com

© 2007 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.