High Performance 3D Games on Windows Phone 7 Series

Shawn Hargreaves
Microsoft Corporation
Blog: http://shawnhargreaves.com
Twitter: @ShawnHargreave
Consistent sets of hardware capabilities defined by Microsoft

- One resolution at launch
- Second resolution added later
- Same touch input
- Consistent processor / GPU
- Same available RAM
- Optional keyboard
The CPU
The Evolution of Programming

Low level control
Straight to the metal
Raw performance tuning

High level abstraction
Rely on compiler and runtime
Developer productivity
for (EntityList::iterator it = entities.begin(); it != entities.end(); it++)
{
    ICollidable* col = dynamic_cast<ICollidable*>(*it);
    if (col)
        pendingCollisions.push_back(new CollisionInfo(col));
}
Why C# r0x0rz

- Powerful and expressive
- Type safety reduces hard-to-track-down bugs
- Reflection
- Initializer syntax
- Blazingly fast compiles
- Great tooling (IntelliSense)
- Similar enough to C that learning and porting are easy
.NET on Windows

Usually within a few percent of native performance

Awesome generational garbage collection

Performance shootout: Raymond Chen vs. Rico Mariani
http://blogs.msdn.com/ricom/archive/2005/05/10/416151.aspx
.NET on Xbox 360

Significant delta between managed and native

.NET Compact Framework
Simplistic mark-and-sweep garbage collection

Xbox is not a general purpose computer
Unforgiving in-order CPU architecture
Requires custom VMX instructions for optimal math perf
Security architecture poses challenges for jitted code
.NET on Windows Phone 7 Series

In between Windows and Xbox 360

.NET Compact Framework
Keep an eye on garbage collection!

ARMv7 CPU
More forgiving toward jitted code
ARM jitter is more mature than PPC
Ways To Call Code

- Instance method
- Virtual method
- Interface
- Delegate / event
- Reflection
C++ allows independent choice of

- Data type
  - The memory in which a type lives (placement new)
  - How a type instance is referenced (T, T*, T&, const T&)

.NET types dictate their allocation and usage semantics

- Value types
  - int, bool, struct, Vector3
- Reference types
  - class, array, string, delegate, boxed value types
A Popular Myth

Oft-repeated wisdom

- Value types live on the stack
- Reference types live on the heap

That is subtly incorrect

- Value types live wherever they are declared
- Reference types have two pieces
  - Memory allocated from the heap
  - A pointer to this heap memory
By default, prefer class over struct.

Use struct for things that are:
- Small (<= 16 bytes)
- Short lived

Pass large structs by reference:

```csharp
Matrix a, b, c;
c = Matrix.Multiply(a, b); // copies 192 bytes!
Matrix.Multiply(ref a, ref b, out c);
```
### Garbage collection is not optional
Can’t have type safety without automatic memory management

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>.NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate</td>
<td>Initially fast, becoming slower as fragmentation increases</td>
<td>Very fast, apart from periodic garbage collections</td>
</tr>
<tr>
<td>Free</td>
<td>Fast</td>
<td>Instantaneous</td>
</tr>
<tr>
<td>Fragmentation</td>
<td>Increases over time</td>
<td>None</td>
</tr>
<tr>
<td>Cache coherency</td>
<td>Requires custom allocators</td>
<td>Things allocated close in time are also close in physical location</td>
</tr>
</tbody>
</table>
Mark and Sweep

1. Triggered per megabyte of allocation
2. Starts with root references (stack variables, statics)
3. Recursively follows all references to see what other objects can be reached
4. Anything we didn’t reach must be garbage
5. Compacts the heap, sliding live objects down to fill holes
Two Ways To Keep GC Happy

Make it run **Less Often**

- If you never allocate, GC will never run
- Use object pools

Make it **Finish Quickly**

- Collection time is proportional to how many object references must be traversed
- Simple heap = fast collection
- Use value types and integer handles
Explicitly forces a garbage collection

Don’t call every frame!

Use wisely to give yourself more headroom
After loading
During pauses in gameplay
Avoiding Allocation

Beware of boxing

string vs. StringBuilder

Use WeakReference to track GC frequency

Use CLR Profiler on Windows
See Cullen Waters talk: “Development and Debugging Tools for Windows Phone 7 Series”

Use .NET Reflector to peek behind the curtain
http://www.red-gate.com/products/reflector/
foreach is Syntactic Sugar

```csharp
foreach (var x in collection) DoStuff(x);

becomes:

var enumerator = collection.GetEnumerator();
while (enumerator.MoveNext()) DoStuff(enumerator.Current);
```

Is the enumerator a value type?
Array, List<T>, and most XNA types are fine
Some collection types create garbage
IEnumerator<Action> Think()
{
    while (true)
    {
        Heading = ChooseRandomDirection();

        while (ElapsedTime < 23)
        {
            yield return Action.WalkForward;
        }

        yield return Action.LookAround;

        if (Vector3.Distance(Position, Enemy.Position) < 42)
        {
            yield return Action.LoadWeapon;
            yield return Action.FireWeapon;
        }
    }
}
Iterators Are Compiler Magic

[CompilerGenerated]
private sealed class <Think>d__0 : IEnumerable<Action>
{
    private int <>1__state;
    private Action <>2__current;

    private bool MoveNext()
    {
        switch (this.<>1__state)
        {
            case 0:
                this.<>1__state = -1;
                break;

            case 1:
                goto Label_0073;

            case 2:
                this.<>1__state = -1;
                if (Vector3.Distance(this.<>4__this.Position,
                    ...
                )
                    break;
                this.<>2__current = Action.LoadWeapon;
                this.<>1__state = 3;
                return true;

            default:
                return false;
        }
        this.<>4__this.Heading = ChooseRandomDirection();
        while (this.<>4__this.ElapsedTime < 23f)
        {
            this.<>2__current = Action.WalkForward;
            this.<>1__state = 1;
            return true;
        }
        Label_0073:
        this.<>1__state = -1;
        this.<>2__current = Action.FireWeapon;
        this.<>1__state = 4;
        return true;

            case 3:
                this.<>1__state = -1;
                this.<>2__current = Action.FireWeapon;
                this.<>1__state = 4;
                return true;

            case 4:
                this.<>1__state = -1;
                break;

                default:
                    return false;
        }
        this.<>4__this.Heading = ChooseRandomDirection();
        while (this.<>4__this.ElapsedTime < 23f)
        {
            this.<>2__current = Action.WalkForward;
            this.<>1__state = 1;
            return true;
        }
        Label_0073:
        this.<>1__state = -1;
        this.<>2__current = Action.LookAround;
        this.<>1__state = 2;
        return true;
    }
}
The GPU
Five Configurable Effects

- BasicEffect
- DualTextureEffect
- AlphaTestEffect
- SkinnedEffect
- EnvironmentMapEffect

Plus hardware accelerated 2D sprite drawing
BasicEffect

- 0-3 directional lights
- Blinn-Phong shading
- Optional texture
- Optional fog
- Optional vertex color

<table>
<thead>
<tr>
<th></th>
<th>Vertex Cost</th>
<th>Pixel Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>No lighting</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>One vertex light</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>Three vertex lights</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>Three pixel lights</td>
<td>18</td>
<td>50</td>
</tr>
<tr>
<td>+ Texture</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td>+ Fog</td>
<td>+4</td>
<td>+2</td>
</tr>
</tbody>
</table>
DualTextureEffect

- For lightmaps, detail textures, decals
- Blends two textures
- Separate texture coordinates
- Modulate 2X combine mode (A*B*2)
- Good visuals at low pixel cost

<table>
<thead>
<tr>
<th></th>
<th>Vertex Cost</th>
<th>Pixel Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two Textures</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>+ Fog</td>
<td>+4</td>
<td>+2</td>
</tr>
</tbody>
</table>
**AlphaTestEffect**

- For billboards and imposters
- Adds alpha test operations (pixel kill)
- Standard blending is free with all effects
- Only need alpha test if you want to disable depth/stencil writes

<table>
<thead>
<tr>
<th>Condition</th>
<th>Vertex Cost</th>
<th>Pixel Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;, &lt;=, &gt;=, &gt;</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>==, !=</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>+ Fog</td>
<td>+4</td>
<td>+2</td>
</tr>
</tbody>
</table>
SkinnedEffect

- For animated models and instancing
- Game code animates bones on CPU
- Vertex skinning performed by GPU
- Up to 72 bones
- One, two, or four weights per vertex

<table>
<thead>
<tr>
<th>Lights</th>
<th>Vertex Cost</th>
<th>Pixel Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>One vertex light</td>
<td>55</td>
<td>4</td>
</tr>
<tr>
<td>Three vertex lights</td>
<td>75</td>
<td>4</td>
</tr>
<tr>
<td>Three pixel lights</td>
<td>33</td>
<td>51</td>
</tr>
<tr>
<td>Two bones</td>
<td>+7</td>
<td>+0</td>
</tr>
<tr>
<td>Four bones</td>
<td>+13</td>
<td>+0</td>
</tr>
<tr>
<td>Fog</td>
<td>+0</td>
<td>+2</td>
</tr>
</tbody>
</table>
**EnvironmentMapEffect**

- **Oooh, shiny!**
- **Diffuse texture + cube environment map**
- **Cheap way to fake many complex lights**
- **Fresnel term simulates behavior when light reaches a surface and some reflects, some penetrates**

<table>
<thead>
<tr>
<th></th>
<th>Vertex Cost</th>
<th>Pixel Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>One light</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>Three lights</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>+ Fresnel</td>
<td>+7</td>
<td>+0</td>
</tr>
<tr>
<td>+ Specular</td>
<td>+0</td>
<td>+2</td>
</tr>
<tr>
<td>+ Fog</td>
<td>+0</td>
<td>+2</td>
</tr>
</tbody>
</table>
A Balancing Act

Framercate

Pixel Cost

Number of Pixels
Balancing Framerate

- 30 hz refresh rate
- No point updating faster than the display!

Game.TargetElapsedTime = TimeSpan.FromSeconds(1f / 30);
Balancing Pixel Cost

Pixel Cost

- Prefer cheaper effects
- Minimize overdraw
  - Many known algorithms:
    - Distance, frustum, BSP, sort front to back
    - Implement "overdraw xray mode"
      - Draw untextured with additive blending
      - Brighter areas indicate overdraw
Balancing Number of Pixels

- 800x480 is 25% more pixels than Xbox 1
  - Great for text
  - Too many pixels for intensive games
  - 800x480 = 384,000 pixels
  - 600x360 = 216,000 pixels (56%)
- Dedicated hardware scaler
- Does not consume any GPU
- Higher quality than bilinear upsampling
Scaler Demo
<table>
<thead>
<tr>
<th>Avoid</th>
<th>Prefer</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RenderTargetUsage.PreserveContents</code></td>
<td><code>RenderTargetUsage.DiscardContents</code></td>
</tr>
<tr>
<td><code>device.BlendState = new BlendState {...};</code></td>
<td>// At startup</td>
</tr>
<tr>
<td></td>
<td>static BlendState myState = new BlendState {...};</td>
</tr>
<tr>
<td></td>
<td>// Per frame</td>
</tr>
<tr>
<td></td>
<td>Device.BlendState = myState;</td>
</tr>
<tr>
<td><code>VertexBuffer.SetData(...)</code></td>
<td><code>device.DrawUserPrimitives(...);</code></td>
</tr>
<tr>
<td></td>
<td>// or</td>
</tr>
</tbody>
</table>
|                                                | DynamicVertexBuffer.SetData(...,
|                                                | SetDataOptions.NoOverwrite);               |
Great performance comes from great knowledge

Understand

- Value types vs. reference types
- Garbage collection
- C# compiler magic (foreach, iterator methods, closures)
- Cost of the different graphical effect options

Actions

- Use CLR Profiler and .NET Reflector
- Render smaller than display resolution, rely on scaler