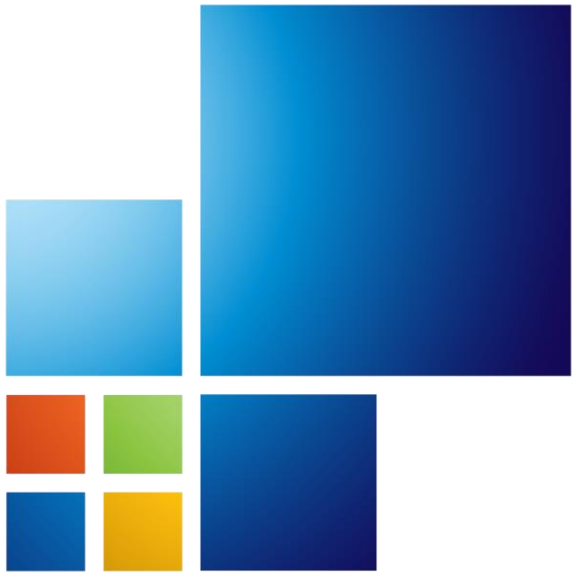


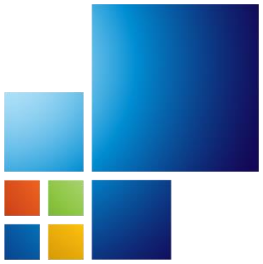
High Performance 3D Games on Windows Phone 7 Series

Shawn Hargreaves
Microsoft Corporation
Blog: http://shawnhargreaves.com
Twitter: @ShawnHargreave

Windows Phone 7 Series Hardware

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

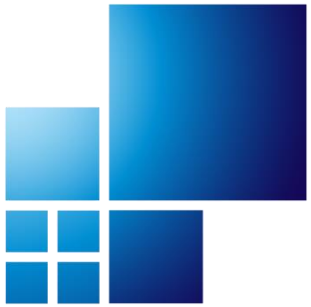
Straight to the metal

Raw performance tuning

High level abstraction

Rely on compiler and runtime

Developer productivity



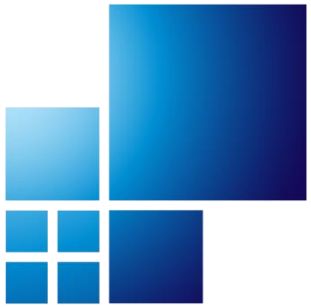
C++ Interview Question

```
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

C#

- 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

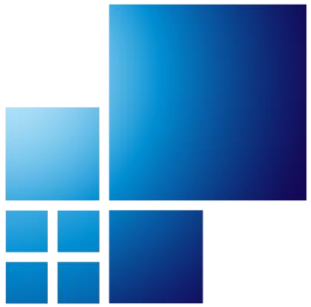


.NET on Windows

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

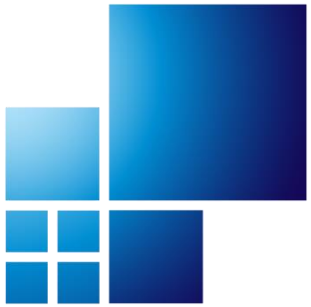
Simplistic mark-and-sweep garbage collection

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

Microsoft® .NET

Windows Phone 7 Series

In between Windows and Xbox 360

.NET Compact Framework

Keep an eye on garbage collection!

ARMv7 CPU

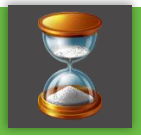
More forgiving toward jitted code

ARM jitter is more mature than PPC

Ways To Call Code

Instance method

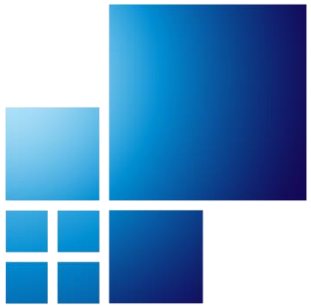
Virtual method



Interface

Delegate / event

Reflection



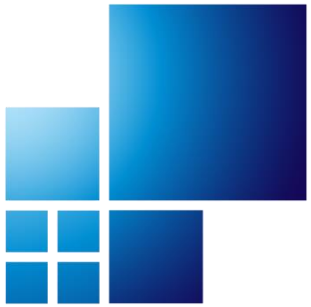
Choose Your Own Address

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

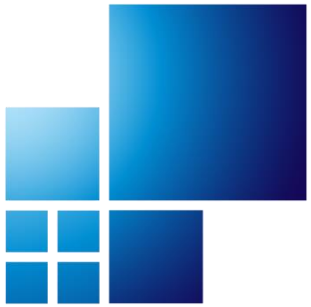
- Value types live on the stack
- Reference types live on the heap

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



class vs. struct

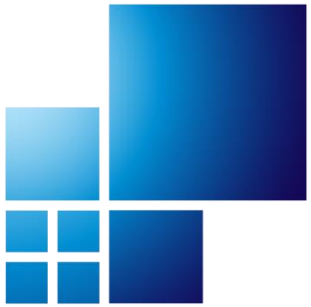
By default,
prefer class over struct

Use struct for things that are

- Small (<= 16 bytes)
- Short lived

Pass large structs by reference

```
Matrix a, b, c;
c = Matrix.Multiply(a, b); // copies 192 bytes!
Matrix.Multiply(ref a, ref b, out c);
```

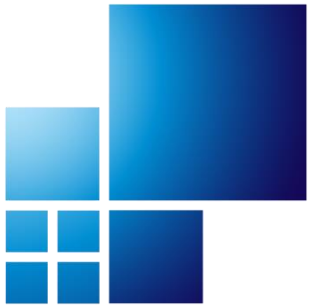


Memory Management

Garbage collection is not optional

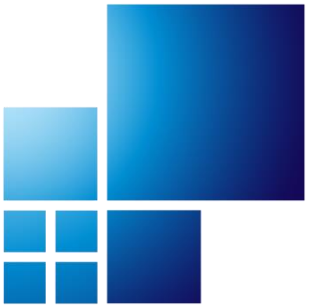
Can't have type safety without automatic memory management

	C++	.NET
Allocate	Initially fast, becoming slower as fragmentation increases	Very fast, apart from periodic garbage collections
Free	Fast	Instantaneous
Fragmentation	Increases over time	None
Cache coherency	Requires custom allocators	Things allocated close in time are also close in physical location



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

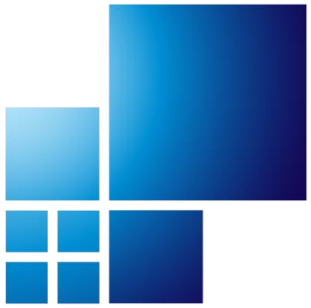
Use value types and integer handles

GC.Collect

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

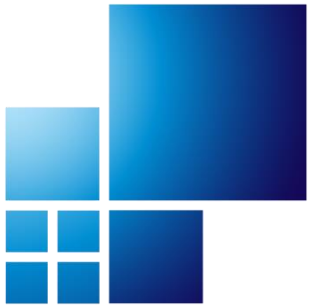
Beware of boxing

string vs. StringBuilder

Use WeakReference to track GC frequency
http://blogs.msdn.com/shawnhar/archive/2007/10/12/monitoring-the-garbage-collector.aspx

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/



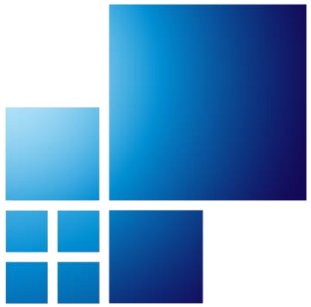
foreach is Syntactic Sugar

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

    becomes:

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

Is the enumerator a value type?

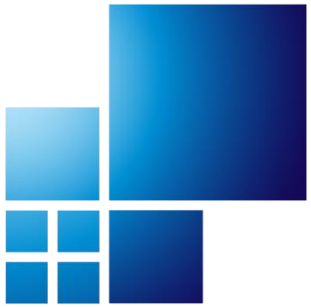
Array, List<T>, and most XNA types are fine

Some collection types create garbage



Iterator Methods

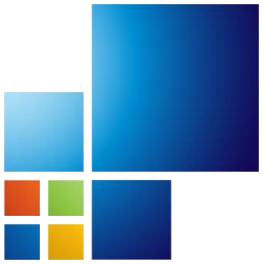
```
IEnumerable<Action> Think()
{
    while (true)
    {
        Heading = ChooseRandomDirection();

        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;
```

```
            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`

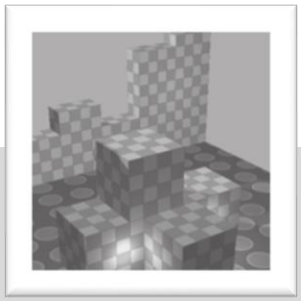
`SkinnedEffect`

`EnvironmentMapEffect`

Plus hardware accelerated 2D sprite drawing

BasicEffect

`BasicEffect`



`DualTextureEffect`

`AlphaTestEffect`

`SkinnedEffect`

`EnvironmentMapEffect`

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

	Vertex Cost	Pixel Cost
No lighting	5	1
One vertex light	40	1
Three vertex lights	60	1
Three pixel lights	18	50
+ *Texture*	+1	+2
+ *Fog*	+4	+2

DualTextureEffect


BasicEffect


DualTextureEffect


AlphaTestEffect


SkinnedEffect

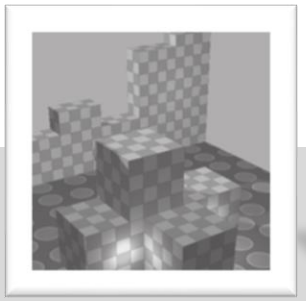

EnvironmentMapEffect

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

	Vertex Cost	Pixel Cost
Two Textures	7	6
+ Fog	*+4*	*+2*

AlphaTestEffect


BasicEffect


DualTextureEffect


AlphaTestEffect


SkinnedEffect

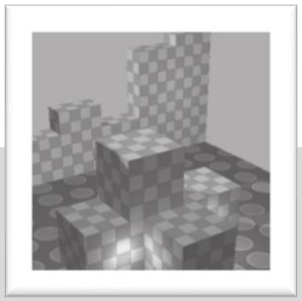

EnvironmentMapEffect

- 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

	Vertex Cost	Pixel Cost
<, <=, >=, >	6	6
==, !=	6	10
+ Fog	*+4*	*+2*

SkinnedEffect


BasicEffect


DualTextureEffect


AlphaTestEffect


SkinnedEffect

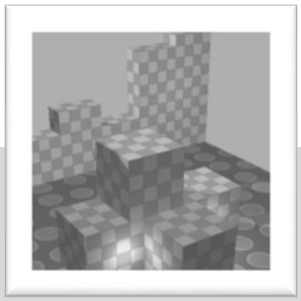

EnvironmentMapEffect

- 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

	Vertex Cost	Pixel Cost
One vertex light	55	4
Three vertex lights	75	4
Three pixel lights	33	51
+ Two bones	*+7*	*+0*
+ Four bones	*+13*	*+0*
+ Fog	*+0*	*+2*

EnvironmentMapEffect


BasicEffect


DualTextureEffect

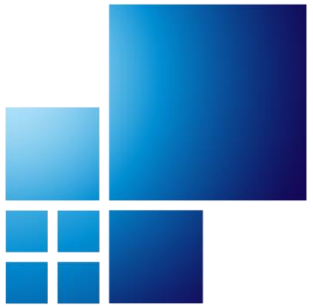

AlphaTestEffect


SkinnedEffect


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

	Vertex Cost	Pixel Cost
One light	32	6
Three lights	36	6
+ *Fresnel*	+7	+0
+ *Specular*	+0	+2
+ *Fog*	+0	+2

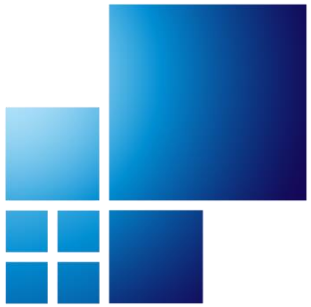


A Balancing Act

Framerate

Pixel Cost

Number of Pixels

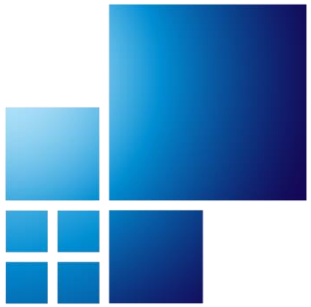


Balancing Framerate

Framerate

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

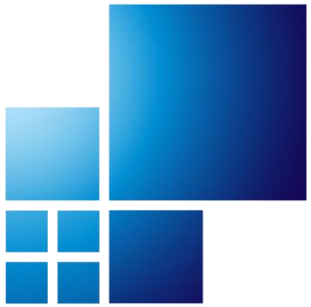
```
Game.TargetElapsedTime =
     TimeSpan.FromSeconds(1f / 30);
```



Balancing Pixel Cost

Pixel Cost

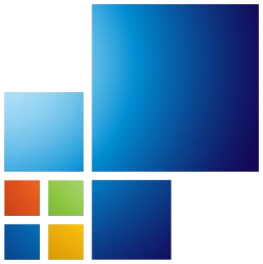
- Prefer cheaper effects
- Minimize overdraw
  - Many known algorithms:
    - Distance, frustum, BSP, sort front to back
  - Implement “overdraw xray mode”
    - Draw untextured with additive blending
    - Brighter areas indicate overdraw



Balancing Number of Pixels

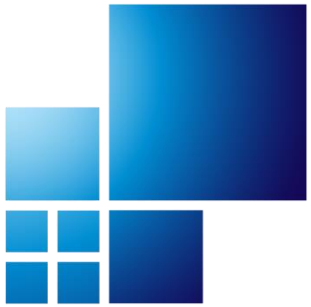
- 800x480 is 25% more pixels than Xbox 1
  - Great for text
  - Too many pixels for intensive games
  - 800x480 = 384,000 pixels
  - 600x360 = 216,000 pixels (56%)
- Dedicated hardware scaler
- Does not consume any GPU
- Higher quality than bilinear upsampling

Number of Pixels



Demo

Scaler Demo

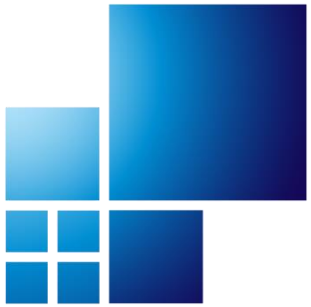


XNA Framework API Cheat Sheet

Avoid	Prefer
`RenderTargetUsage.PreserveContents`	`RenderTargetUsage.DiscardContents`

`device.BlendState = new BlendState {…};`	`// At startup`
	`static BlendState myState = new BlendState {…};`
	
	`// Per frame`
	`Device.BlendState = myState;`

`VertexBuffer.SetData(…)`	`device.DrawUserPrimitives(…);`
	
	`// or`
	
	`DynamicVertexBuffer.SetData(…,`
	`SetDataOptions.NoOverwrite);`



Summary

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