Tony Albrecht

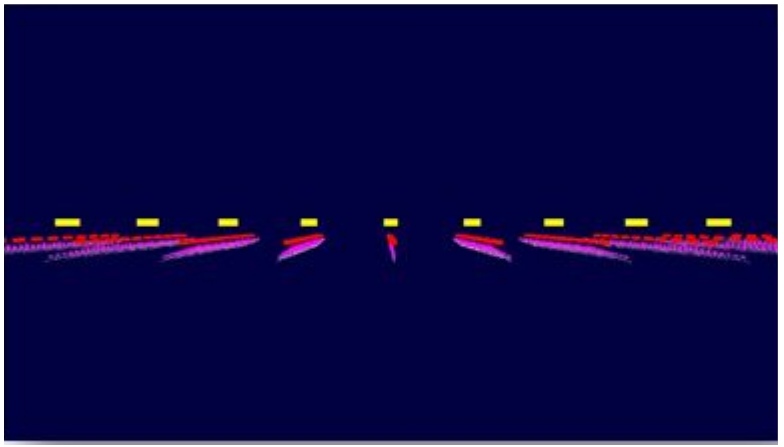
Riot Games

Pitfalls of Object Oriented Programming - Revisited

@TonyAlbrecht

Pitfalls of Object Oriented Programming - 2009

- Investigated the performance of a simple OO scenetree.
- Ran on PlayStation 3.
- Used Sony tools for profiling.
- PS3 had a limited CPU.



- Original: http://overbyte.com.au/misc/Pitfalls2009.pdf

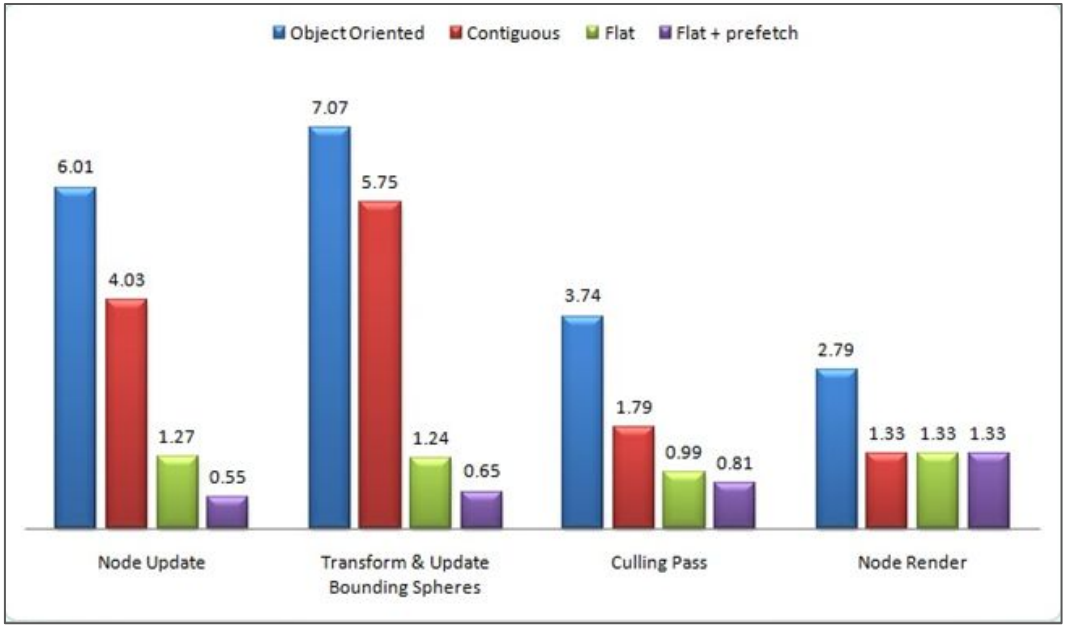
Pitfalls 2009

Start: 19.2ms

Data reorg: 12.9ms

Linear traversal: 4.8ms

Prefetching: 3.3ms



8 years later...

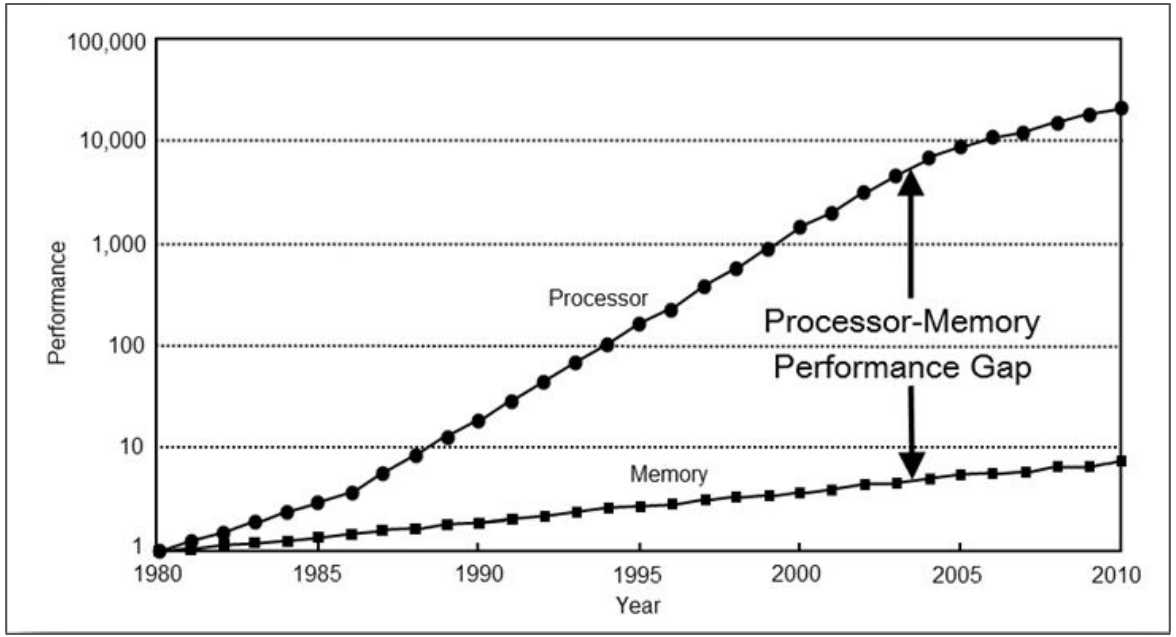
- Do we still need to care about data as much?
- What about branching?
- Prefetching?
- Virtuals?
- Can’t the compiler optimise it?


The compiler will **not** tidy your room!

“The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry.”

-Henry Petroski

Random Memory Accesses are slow



Computer architecture: a quantitative approach

By John L. Hennessy, David A. Patterson, Andrea C. Arpaci-Dusseau

Caches

- Number of levels, types and speeds depend on your platform:
  - L1 ~ cycles
  - L2 ~ 10s to 100s of cycles
  - L3 ~ 100s to thousands of cycles
- Fairly dumb - they store data for access until evicted.
- **CPUs will try to predict where the next access will be.**

How does the CPU prefetch?

- Linearly.
  - Uniform stride/direction.
- Multiple streams can be active at once.
  - But only a limited number of them.

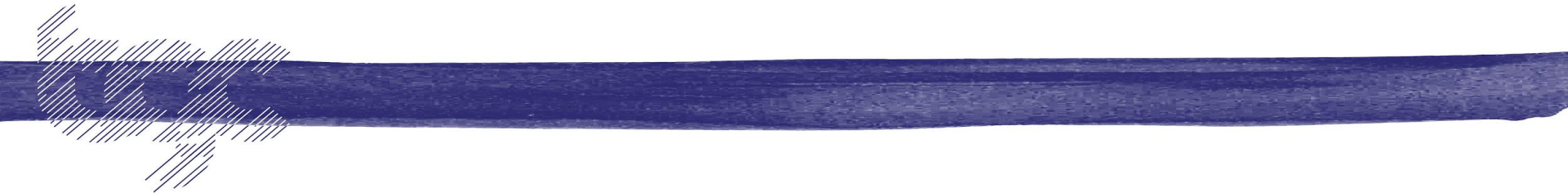
A smart programmer will take advantage of this.

So, memory access is slow?

If what I’m saying is true, we should be able to observe and measure it.

Then, as we change code and data, we can measure the changes in performance.

This is not an ideological argument. This is science.



STAND BACK

I'M GOING TO ~~TRY~~ DO

SCIENCE

Performance measurement?

- Profilers
  - Instrumented
  - Sampling
  - Special

A quick note on units:

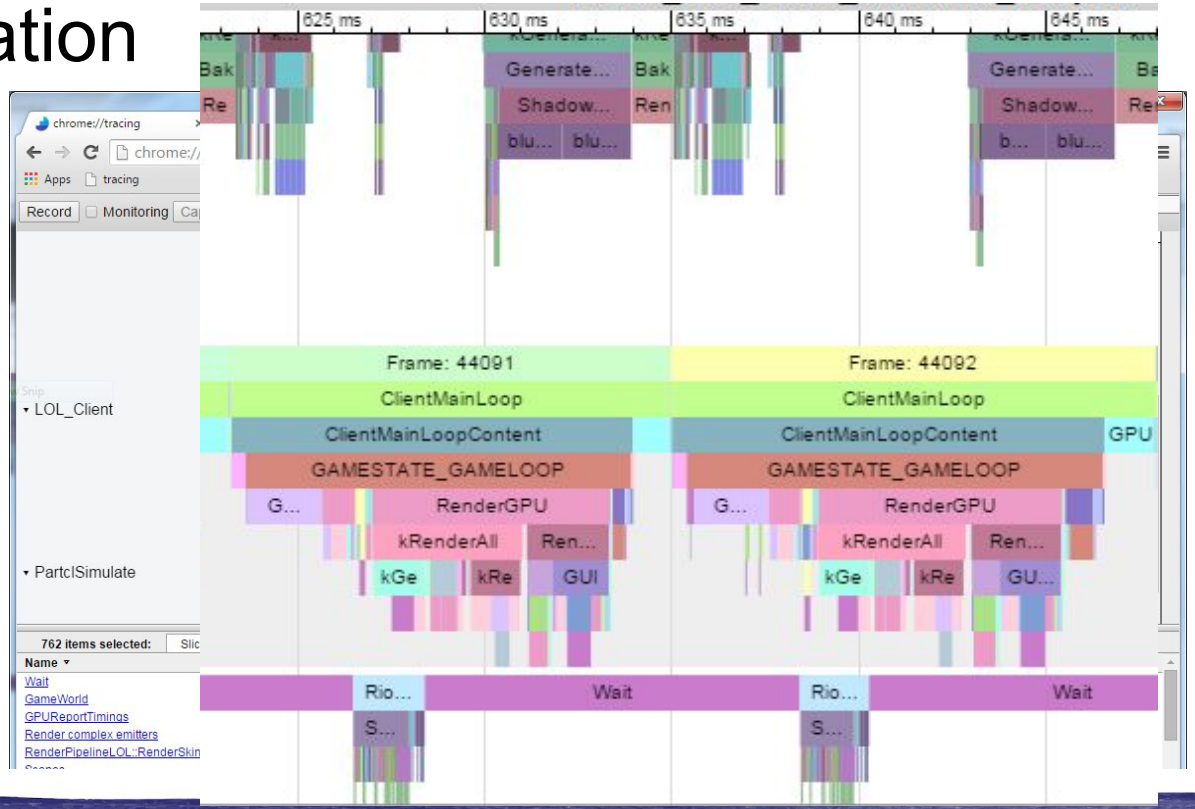
- Never use Frames Per Second to measure performance.
- FPS is a relative measurement.
- For example: How much faster is “20fps faster”?
- That depends...
  - 60fps -> 80fps = 4.16ms improvement per frame
  - 20fps -> 40fps = 25ms improvement per frame

Instrumented profiling

- Manually mark up sections to profile
  - Record unique ID
  - Start time
  - End time
- Visualise it

```
48  void Node::Update()
49  {
50      FT_PROFILE_FN
51      for(Object* obj : mObjects)
52      {
53          obj->Update();
54      }
55  }
56
```

Visualisation



Instrumented Profilers

Pros

- Fantastic for detecting spikes
- Provides a visual sense of performance characteristics
- Top-down view

Cons

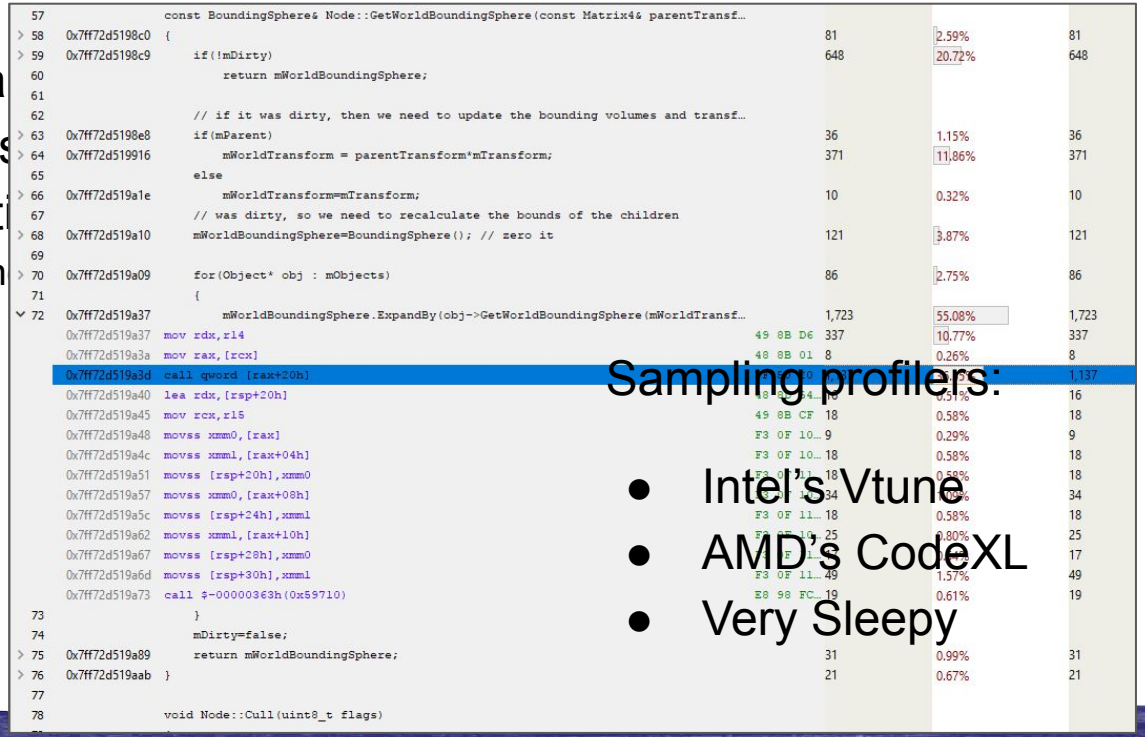
- Intrusive
- Won’t tell you which lines are slow

Examples:

- RAD Game Tool’s Telemetry
- Write your own - visualise with chrome://tracing
- Use mine (when I release it)

Sampling profilers

- Rapidly sa
- Then reass
- Slow funct
  - Slow lin
- Bottom up



```
57                  const BoundingSphere& Node::GetWorldBoundingSphere(const Matrix4& parentTransf…
58  0x7ff72d5198c0  {                                                                  81   2.59%  81
59  0x7ff72d5198c9      if(!mDirty)                                                    648  20.72%  648
60                          return mWorldBoundingSphere;
61
62                      // if it was dirty, then we need to update the bounding volumes and transf…
63  0x7ff72d5198e8      if(mParent)                                                    36   1.15%  36
64  0x7ff72d519916          mWorldTransform = parentTransform*mTransform;              371  11.86%  371
65                      else
66  0x7ff72d519a1e          mWorldTransform=mTransform;                            10   0.32%  10
67                      // was dirty, so we need to recalculate the bounds of the children
68  0x7ff72d519a10      mWorldBoundingSphere=BoundingSphere(); // zero it            121  3.87%  121
69
70  0x7ff72d519a09      for(Object* obj : mObjects)                                    86   2.75%  86
71                      {
72  0x7ff72d519a37          mWorldBoundingSphere.ExpandBy(obj->GetWorldBoundingSphere(mWorldTransf…  1,723  55.08%  1,723
    0x7ff72d519a37  mov rdx,r14                          49 8B D6  337   10.77%  337
    0x7ff72d519a3a  mov rax,[rcx]                        48 8B 01  8     0.26%  8
    0x7ff72d519a3d  call qword [rax+20h]                 FF 50 20  1,137 36.35%  1,137
    0x7ff72d519a40  lea rdx,[rsp+20h]                    48 8D 54…  16   0.51%  16
    0x7ff72d519a45  mov rcx,r15                          49 8B CF  18    0.58%  18
    0x7ff72d519a48  movss xmm0,[rax]                     F3 0F 10…  9    0.29%  9
    0x7ff72d519a4c  movss xmm1,[rax+04h]                 F3 0F 10…  18   0.58%  18
    0x7ff72d519a51  movss [rsp+20h],xmm0                 F3 0F 11…  18   0.58%  18
    0x7ff72d519a57  movss xmm0,[rax+08h]                 F3 0F 10…  34   1.09%  34
    0x7ff72d519a5c  movss [rsp+24h],xmm1                 F3 0F 11…  18   0.58%  18
    0x7ff72d519a62  movss xmm1,[rax+10h]                 F3 0F 10…  25   0.80%  25
    0x7ff72d519a67  movss [rsp+28h],xmm0                 F3 0F 11…  17   0.54%  17
    0x7ff72d519a6d  movss [rsp+30h],xmm1                 F3 0F 11…  49   1.57%  49
    0x7ff72d519a73  call $-00000363h(0x59710)            E8 98 FC…  19   0.61%  19
73                      }
74                      mDirty=false;
75  0x7ff72d519a89      return mWorldBoundingSphere;                                   31   0.99%  31
76  0x7ff72d519aab  }                                                                  21   0.67%  21
77
78                  void Node::Cull(uint8_t flags)
```

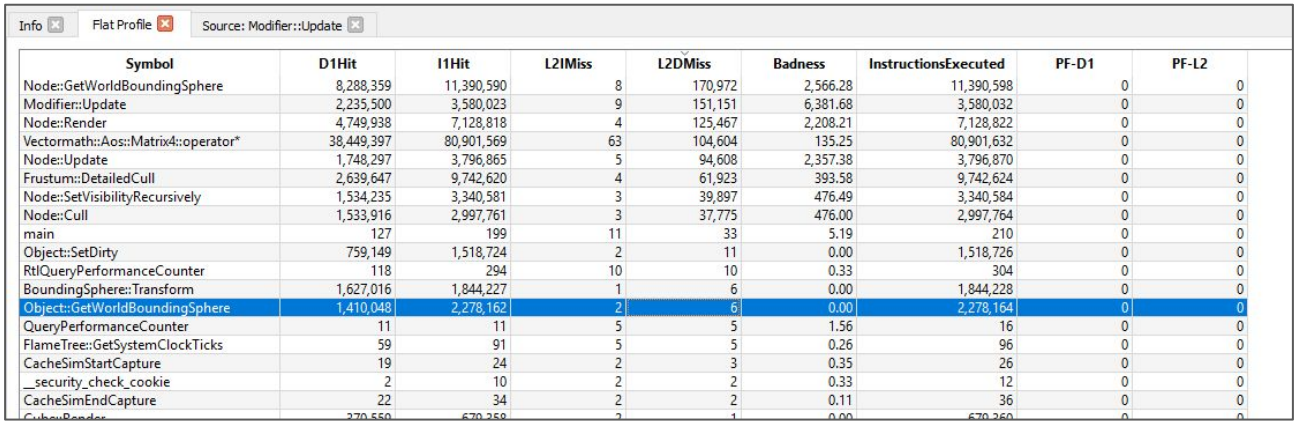
Sampling profilers:

- Intel’s Vtune
- AMD’s CodeXL
- Very Sleepy

Specialised Profilers

Extract particular information from a process

- CPU specific perf counters
  - AMD/Intel profilers
- CacheSim
  - https://github.com/InsomniacGames/ig-cachesim



Info | Flat Profile | Source: Modifier::Update

Symbol	D1Hit	I1Hit	L2IMiss	L2DMiss	Badness	InstructionsExecuted	PF-D1	PF-L2
Node::GetWorldBoundingSphere	8,288,359	11,390,590	8	170,972	2,566.28	11,390,598	0	0
Modifier::Update	2,235,500	3,580,023	9	151,151	6,381.68	3,580,032	0	0
Node::Render	4,749,938	7,128,818	4	125,467	2,208.21	7,128,822	0	0
Vectormath::Aos::Matrix4::operator*	38,449,397	80,901,569	63	104,604	135.25	80,901,632	0	0
Node::Update	1,748,297	3,796,865	5	94,608	2,357.38	3,796,870	0	0
Frustum::DetailedCull	2,639,647	9,742,620	4	61,923	393.58	9,742,624	0	0
Node::SetVisibilityRecursively	1,534,235	3,340,581	3	39,897	476.49	3,340,584	0	0
Node::Cull	1,533,916	2,997,761	3	37,775	476.00	2,997,764	0	0
main	127	199	11	33	5.19	210	0	0
Object::SetDirty	759,149	1,518,724	2	11	0.00	1,518,726	0	0
RtlQueryPerformanceCounter	118	294	10	10	0.33	304	0	0
BoundingSphere::Transform	1,627,016	1,844,227	1	6	0.00	1,844,228	0	0
Object::GetWorldBoundingSphere	1,410,048	2,278,162	2	6	0.00	2,278,164	0	0
QueryPerformanceCounter	11	11	5	5	1.56	16	0	0
FlameTree::GetSystemClockTicks	59	91	5	5	0.26	96	0	0
CacheSimStartCapture	19	24	2	3	0.35	26	0	0
__security_check_cookie	2	10	2	2	0.33	12	0	0
CacheSimEndCapture	22	34	2	2	0.11	36	0	0

When optimising

- You want a deterministic test case (if possible)
  - Otherwise, be aware of iterative variation
  - Run test case multiple times and compare

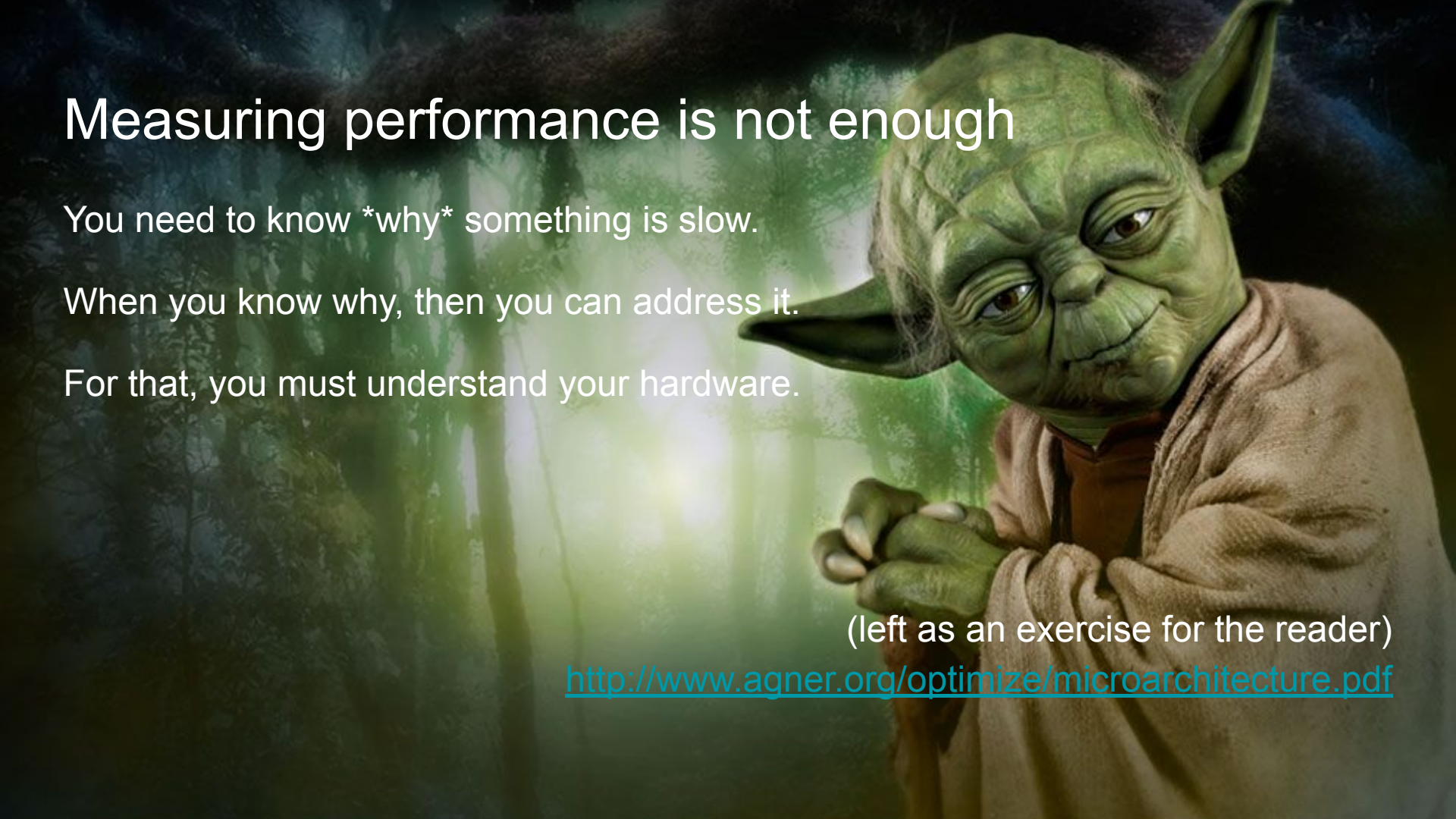
- USE THE COMPILER OPTIONS!!
  - Learn what the different compiler options do.
  - Experiment and Profile!

Measuring performance is not enough

You need to know *why* something is slow.

When you know why, then you can address it.

For that, you must understand your hardware.



(left as an exercise for the reader)

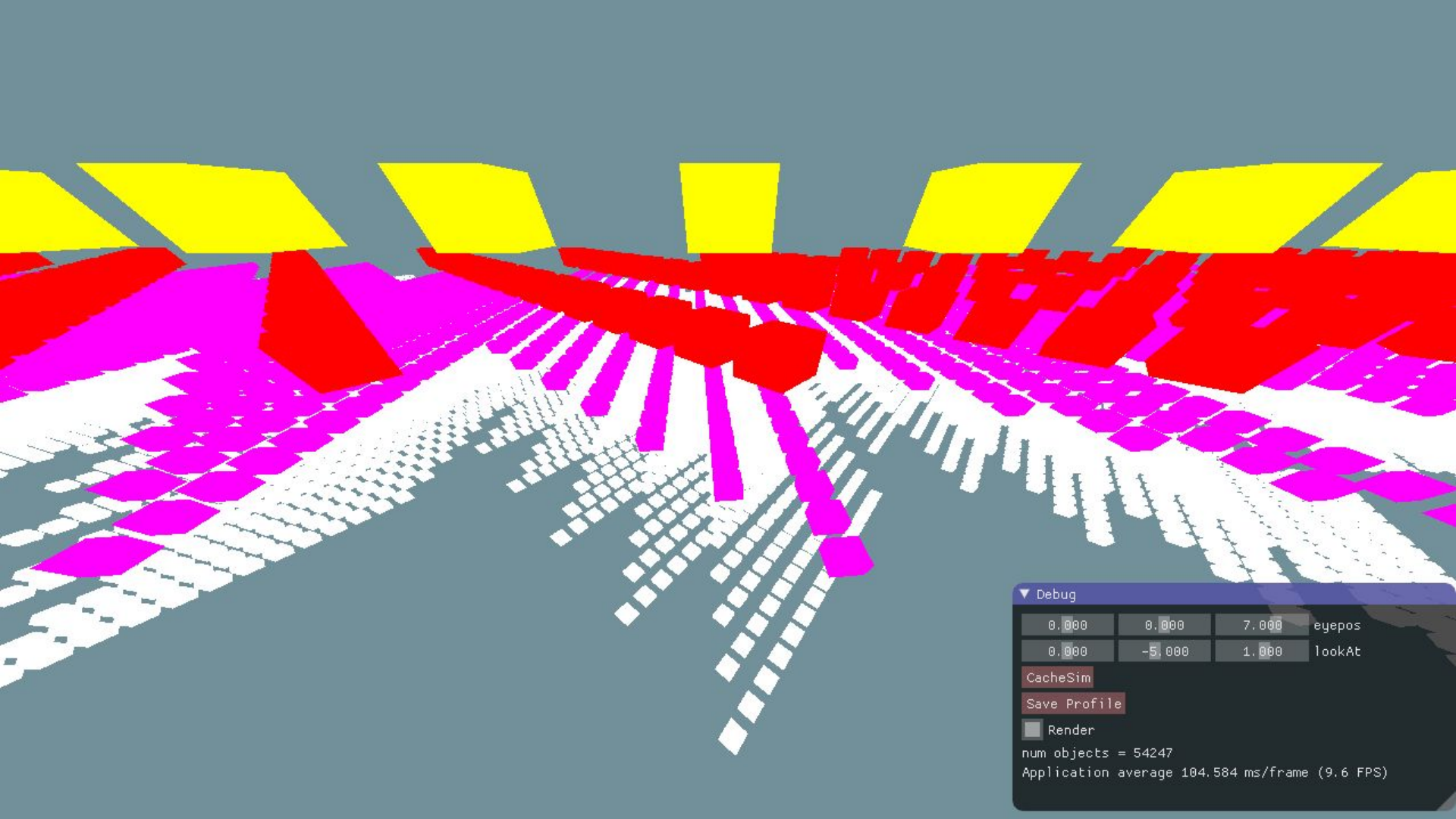
http://www.agner.org/optimize/microarchitecture.pdf

The Test Case

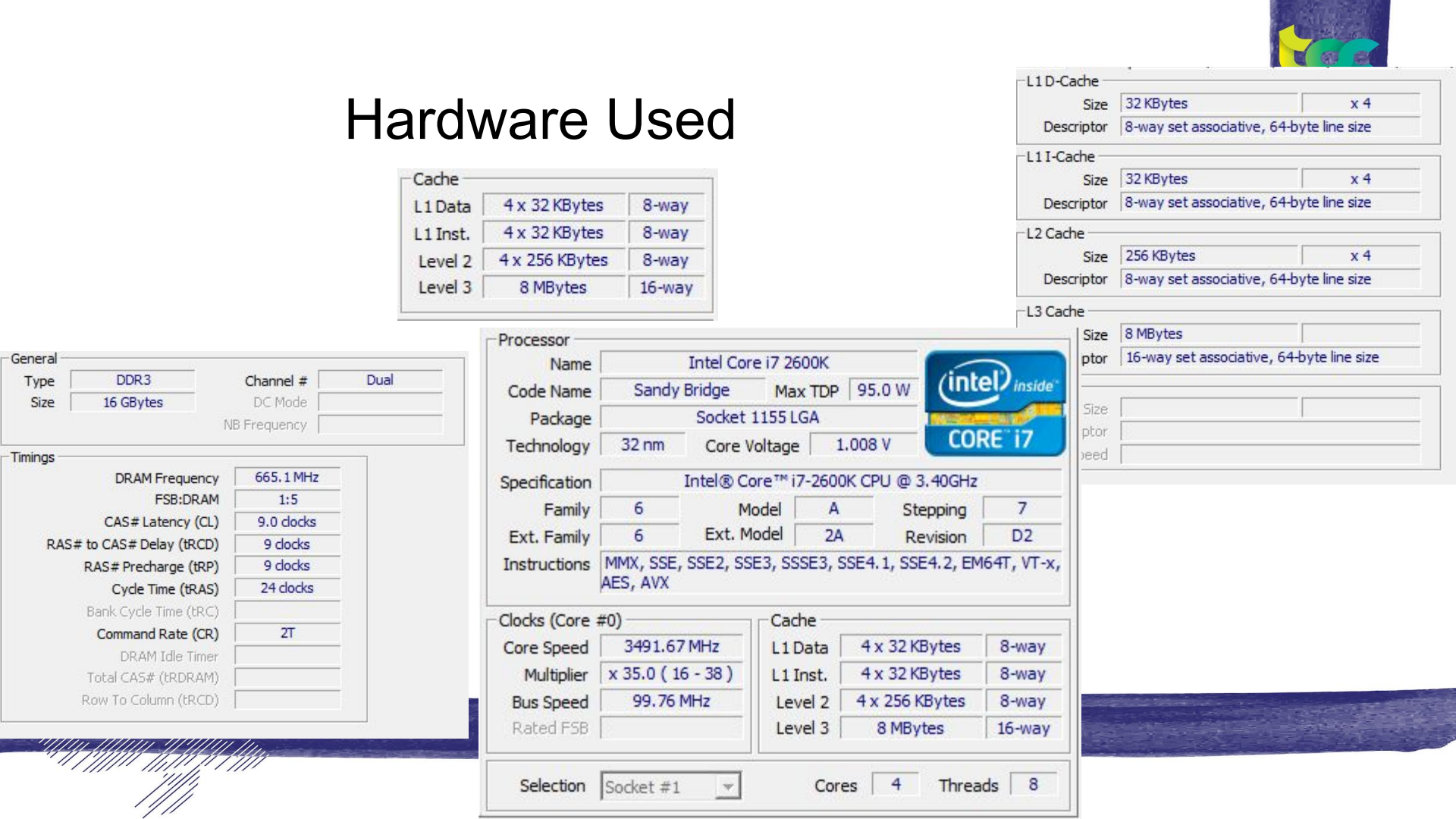
Basically the same code as the 2009 Pitfalls talk, but with more.
55,000 objects instead of 11,000.

Animates, culls and renders a scenetree.

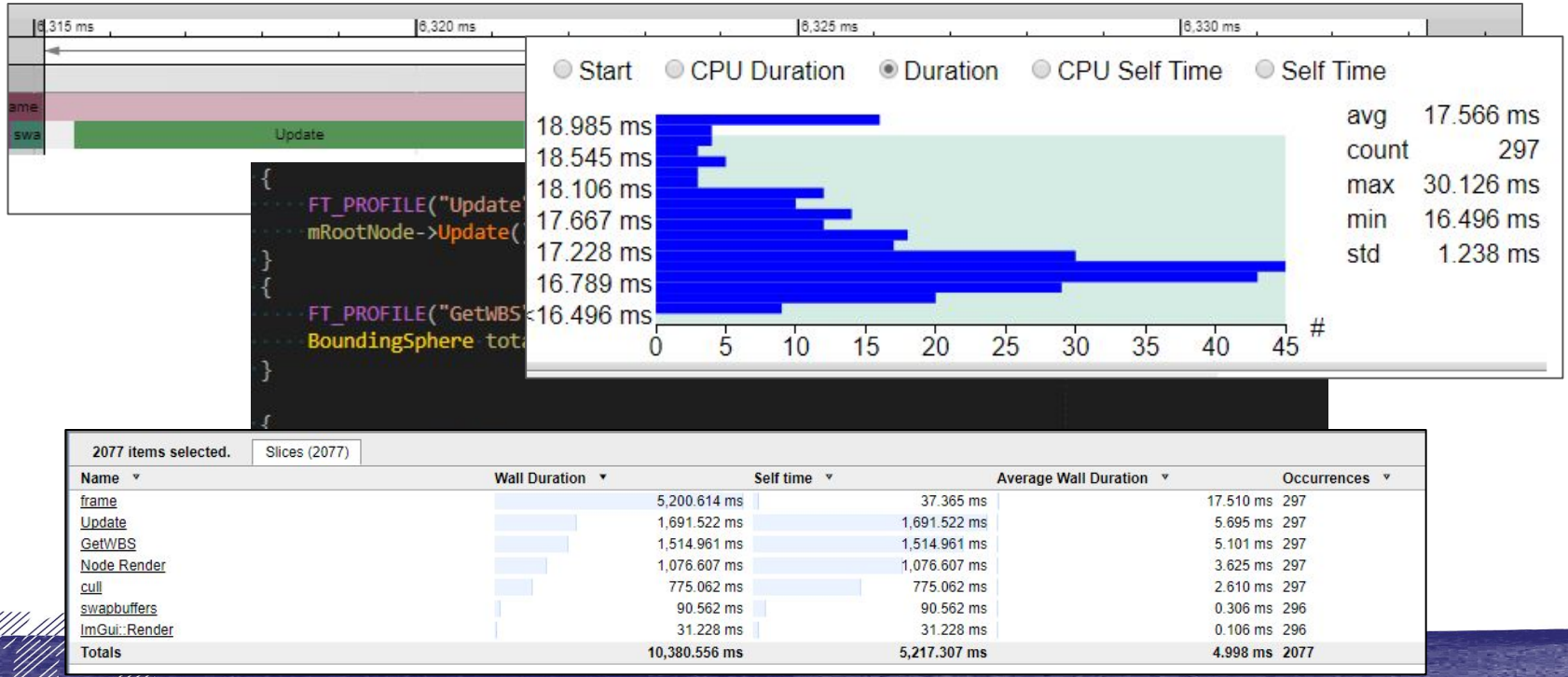
- FREE 3rd party libs/applications:
  - dear imgui: https://github.com/ocornut/imgui
  - Vectormath from Bullet: http://bulletphysics.org/
  - Chrome Tracing for perf vis: chrome://tracing
  - CodeXL:http://gpuopen.com/compute-product/codexl/


Debug
0.000
0.000
7.000
eyepos
0.000
-5.000
1.000
lookAt
CacheSim
Save Profile
Render
num objects = 54247
Application average 104.584 ms/frame (9.6 FPS)

Hardware Used



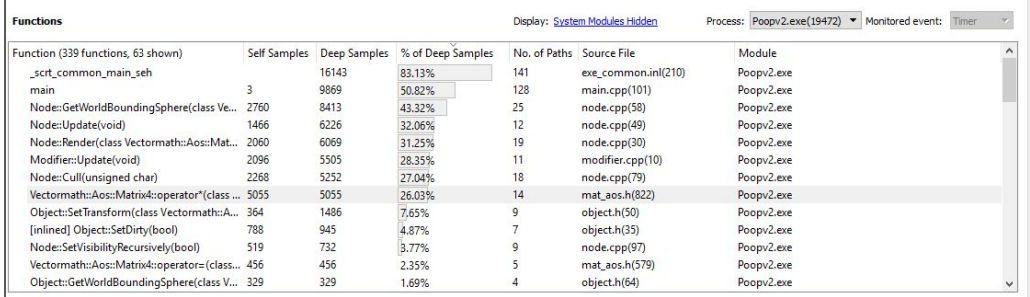
Here’s a single instrumented frame

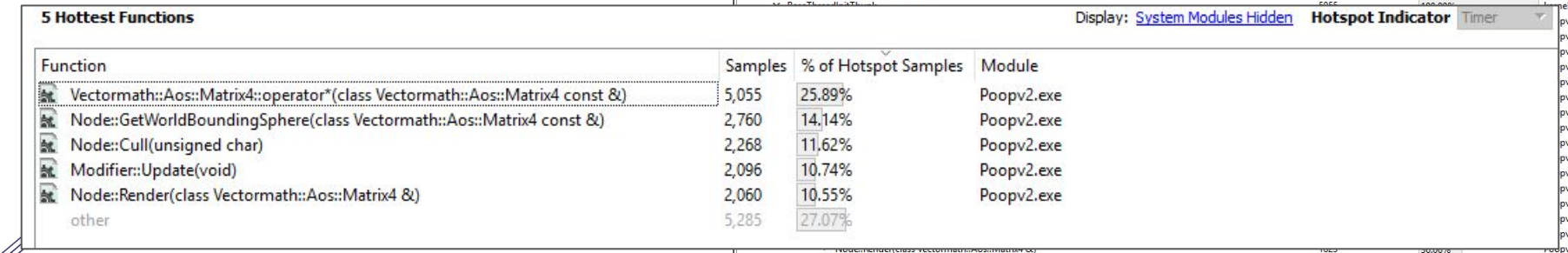


2077 items selected. Slices (2077)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
frame	5,200.614 ms	37.365 ms	17.510 ms	297
Update	1,691.522 ms	1,691.522 ms	5.695 ms	297
GetWBS	1,514.961 ms	1,514.961 ms	5.101 ms	297
Node Render	1,076.607 ms	1,076.607 ms	3.625 ms	297
cull	775.062 ms	775.062 ms	2.610 ms	297
swapbuffers	90.562 ms	90.562 ms	0.306 ms	296
ImGui::Render	31.228 ms	31.228 ms	0.106 ms	296
Totals	**10,380.556 ms**	**5,217.307 ms**	**4.998 ms**	**2077**

Sampling profiler

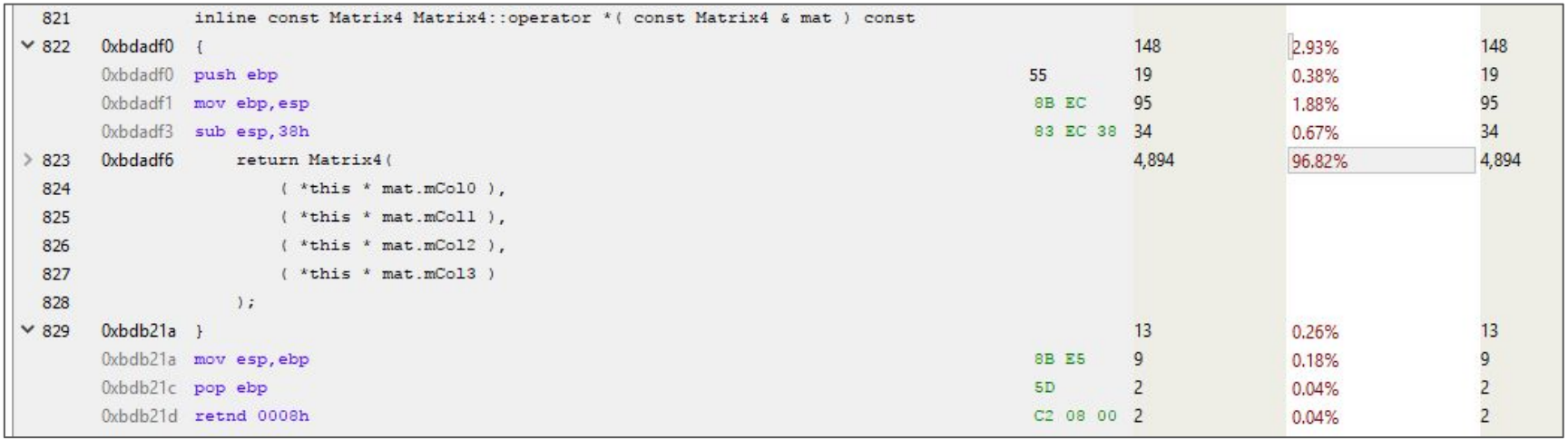




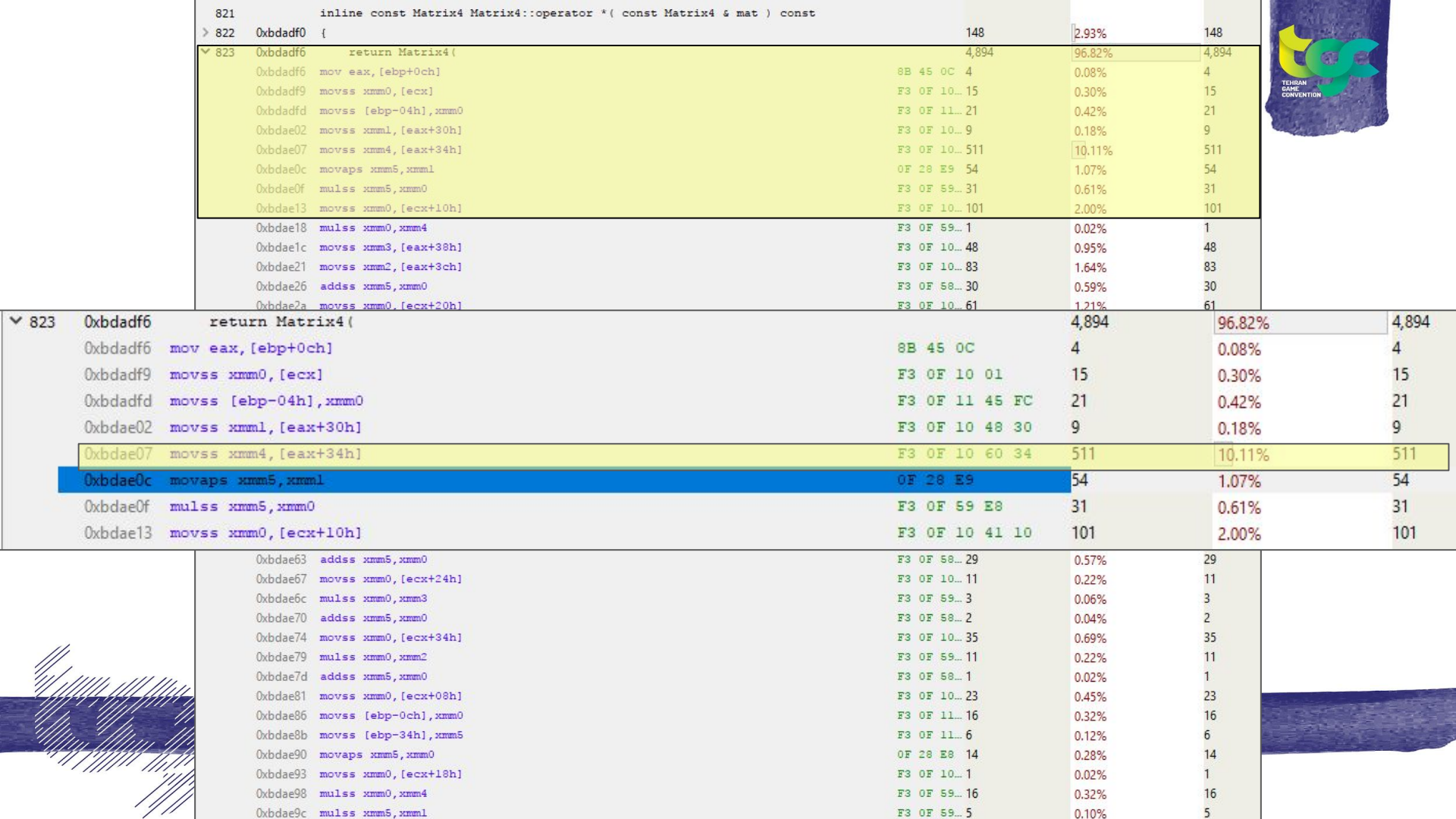
_scrt_common_main_seh		6226	100.00%
Node::Update(void)	1466	4760	100.00%
Modifier::Update(void)	1314	3409	75.86%
Vectormath::Aos::Matrix4::operator*(class Vectormath::Aos::Matrix4 const &)	1591		25.55%
Modifier::Update(void)		782	12.56%
Modifier::Update(void)	373	409	12.56%
Modifier::Update(void)	409		6.57%
Object::SetTransform(class Vectormath::Aos::Matrix4 &)	63	966	16.53%
[inlined] Object::SetDirty(bool)	669	1	10.76%
[inlined] Object::SetDirty(bool)	1		0.02%
Object::SetTransform(class Vectormath::Aos::Matrix4 &)		296	4.75%
Object::SetTransform(class Vectormath::Aos::Matrix4 &)	296		4.75%
Object::SetDirty(bool)	7		0.11%
Vectormath::Aos::Matrix4::operator*(class Vectormath::Aos::Matrix4 const &)	32		0.51%
Object::SetTransform(class Vectormath::Aos::Matrix4 &)	5		0.08%

inline const Matrix4 Matrix4::operator *()

```
821             inline const Matrix4 Matrix4::operator *( const Matrix4 & mat ) const
822 0xbdadf0  {                                                                          148     2.93%   148
823 0xbdadf6        return Matrix4(                                                     4,894   96.82%  4,894
824                     ( *this * mat.mCol0 ),
825                     ( *this * mat.mCol1 ),
826                     ( *this * mat.mCol2 ),
827                     ( *this * mat.mCol3 )
828                 );
829 0xbdb21a  }                                                                         13      0.26%   13
830
```



821		inline const Matrix4 Matrix4::operator *(const Matrix4 & mat) const					
822	0xbdadf0	{			148	2.93%	148
	0xbdadf0	push ebp	55		19	0.38%	19
	0xbdadf1	mov ebp,esp	8B EC		95	1.88%	95
	0xbdadf3	sub esp,38h	83 EC 38		34	0.67%	34
823	0xbdadf6	return Matrix4(			4,894	96.82%	4,894
824		(*this * mat.mCol0),					
825		(*this * mat.mCol1),					
826		(*this * mat.mCol2),					
827		(*this * mat.mCol3)					
828		);					
829	0xbdb21a	}			13	0.26%	13
	0xbdb21a	mov esp,ebp	8B E5		9	0.18%	9
	0xbdb21c	pop ebp	5D		2	0.04%	2
	0xbdb21d	retnd 0008h	C2 08 00		2	0.04%	2


821 inline const Matrix4 Matrix4::operator *(const Matrix4 & mat) const
822 0xbdadf0 {
148 2.93% 148
823 0xbdadf6 return Matrix4(
4,894 96.82% 4,894
0xbdadf6 mov eax,[ebp+0ch] 8B 45 0C 4 0.08% 4
0xbdadf9 movss xmm0,[ecx] F3 0F 10... 15 0.30% 15
0xbdadfd movss [ebp-04h],xmm0 F3 0F 11... 21 0.42% 21
0xbdae02 movss xmm1,[eax+30h] F3 0F 10... 9 0.18% 9
0xbdae07 movss xmm4,[eax+34h] F3 0F 10... 511 10.11% 511
0xbdae0c movaps xmm5,xmm1 0F 28 E9 54 1.07% 54
0xbdae0f mulss xmm5,xmm0 F3 0F 59... 31 0.61% 31
0xbdae13 movss xmm0,[ecx+10h] F3 0F 10... 101 2.00% 101
0xbdae18 mulss xmm0,xmm4 F3 0F 59... 1 0.02% 1
0xbdae1c movss xmm3,[eax+38h] F3 0F 10... 48 0.95% 48
0xbdae21 movss xmm2,[eax+3ch] F3 0F 10... 83 1.64% 83
0xbdae26 addss xmm5,xmm0 F3 0F 58... 30 0.59% 30
0xbdae2a movss xmm0,[ecx+20h] F3 0F 10... 61 1.21% 61
823 0xbdadf6 return Matrix4(4,894 96.82% 4,894
0xbdadf6 mov eax,[ebp+0ch] 8B 45 0C 4 0.08% 4
0xbdadf9 movss xmm0,[ecx] F3 0F 10 01 15 0.30% 15
0xbdadfd movss [ebp-04h],xmm0 F3 0F 11 45 FC 21 0.42% 21
0xbdae02 movss xmm1,[eax+30h] F3 0F 10 48 30 9 0.18% 9
0xbdae07 movss xmm4,[eax+34h] F3 0F 10 60 34 511 10.11% 511
0xbdae0c movaps xmm5,xmm1 0F 28 E9 54 1.07% 54
0xbdae0f mulss xmm5,xmm0 F3 0F 59 E8 31 0.61% 31
0xbdae13 movss xmm0,[ecx+10h] F3 0F 10 41 10 101 2.00% 101
0xbdae63 addss xmm5,xmm0 F3 0F 58... 29 0.57% 29
0xbdae67 movss xmm0,[ecx+24h] F3 0F 10... 11 0.22% 11
0xbdae6c mulss xmm0,xmm3 F3 0F 59... 3 0.06% 3
0xbdae70 addss xmm5,xmm0 F3 0F 58... 2 0.04% 2
0xbdae74 movss xmm0,[ecx+34h] F3 0F 10... 35 0.69% 35
0xbdae79 mulss xmm0,xmm2 F3 0F 59... 11 0.22% 11
0xbdae7d addss xmm5,xmm0 F3 0F 58... 1 0.02% 1
0xbdae81 movss xmm0,[ecx+08h] F3 0F 10... 23 0.45% 23
0xbdae86 movss [ebp-0ch],xmm0 F3 0F 11... 16 0.32% 16
0xbdae8b movss [ebp-34h],xmm5 F3 0F 11... 6 0.12% 6
0xbdae90 movaps xmm5,xmm0 0F 28 E8 14 0.28% 14
0xbdae93 movss xmm0,[ecx+18h] F3 0F 10... 1 0.02% 1
0xbdae98 mulss xmm0,xmm4 F3 0F 59... 16 0.32% 16
0xbdae9c mulss xmm5,xmm1 F3 0F 59... 5 0.10% 5

Cache miss!

- An L3 cache miss is of the order of a few 100 cycles. (200-300?)
- A hit is around 40 cycles

- Average instruction takes 1 to 14 cycles (atomics can be 30+cycles)
- And they can pipeline…
- An L3 Cache miss is equivalent to potentially 100s of instructions.

Let’s take a step back...

- Be careful not to get caught up in micro-optimisation.
- Take the time to understand the big picture.
- Algorithmic optimisations can provide dramatic performance boosts.
- For this example, let’s assume that it’s algorithmically perfect
  - It’s not.

What co

```
Drawable* geo1 = new Cube("cube1", size[0], 0xffff00ff);
Drawable* geo2 = new Cube("cube2", size[1], 0xff0000ff);
Drawable* geo3 = new Cube("cube3", size[2], 0xff00ffff);
Drawable* geo4 = new Cube("cube4", size[3], 0xffffffff);

Modifier* rotatery = new Modifier("RotateY");
Matrix4 roty = Matrix4::rotationY(0.0f);
rotatery->SetTransform(roty);
RootNode->AddObject(rotatery);

Modifier* rotaterz = new Modifier("RotaterZ");
Matrix4 rotz = Matrix4::rotationY(0.022f);
rotaterz->SetTransform(rotz);
RootNode->AddObject(rotaterz);

for (int i = 0; i < level[0]; i++)
{
    Node* node1 = new Node("Node1");
    node1->AddObject(geo1);
    Matrix4 pos1 = Matrix4::translation(Vector3(size[0] * 6 * (i - level[0] / 2), 0, 0));
    node1->SetTransform(pos1);
    RootNode->AddObject(node1);
    rotatery->AddObject(node1);

    for (int j = 0; j < level[1]; j++)
    {
        Node* node2 = new Node("Node2");
        node2->AddObject(geo2);
        node1->AddObject(node2);

        Matrix4 pos2 = Matrix4::translation(Vector3(size[1] * 3 * (j - level[1] / 2), -size[0] * 2, 0));
        node2->SetTransform(pos2);
        rotaterz->AddObject(node2);
        for (int k = 0; k < level[2]; k++)
```

Object Class

```
class Object
{
    virtual void Render(Matrix4& parentTransform) = 0;
    virtual void Update() = 0;
    void SetDirty(bool dirty);
    void SetParent(Object* parent);
    void SetTransform(Matrix4& transform)
    inline const Matrix4 GetTransform() const { return mTransform; }
    inline const Matrix4 GetWorldTransform() const { return mWorldTransform; }

    virtual const BoundingSphere& GetBoundingSphere();
    virtual const BoundingSphere& GetWorldBoundingSphere(const Matrix4& parentTransform);
    virtual void SetVisibilityRecursively(bool visibility);
    virtual void Cull(uint8_t flags);
protected:
    Matrix4 mTransform;
    Matrix4 mWorldTransform;
    BoundingSphere mBoundingSphere;
    BoundingSphere mWorldBoundingSphere;
    bool m_IsVisible = true;
    const char* mName;
    bool mDirty = true;
    Object* mParent;
};
```

Modifiers

- Hold a vector of Objects
- And a Matrix4
- Call Update() to multiply all its Objects by its transform.

```
class Modifier : public Object
{
public:
    Modifier(const char* name) : Object(name) {}
    virtual ~Modifier() {};
    virtual void Render(Matrix4& parentTransform) override {};
    virtual void Update() override;

    void AddObject(Object* obj);
protected:
    std::vector<Object*> mObjects;
};
```

```
void Modifier::AddObject( Object* obj )
{
    mObjects.push_back(obj);
}

void Modifier::Update()
{
    for(Object* obj : mObjects)
    {
        Matrix4 mat = obj->GetTransform();
        mat = mTransform*mat;
        obj->SetTransform(mat);
    }
}
```

Nodes

```cpp
class Node : public Object
{
public:
    Node();
    Node(const char* name);
    virtual ~Node();

    void AddObject(Object* obj);
    virtual void Render(Matrix4& parentTransform) override;
    virtual void Update() override;
    virtual const BoundingSphere& GetWorldBoundingSphere(const Matrix4& parentTra
    void SetVisibilityRecursively(bool visibility) override;
    void Cull(uint8_t flags) override;
protected:
    std::vector<Object*> mObjects;

    static int s_NumNodes;
};
```

```cpp
void Node::Update()
{
    FT_PROFILE_FN
    for(Object* obj : mObjects)
    {
        obj->Update();
    }
}
```

Back to the Cache miss

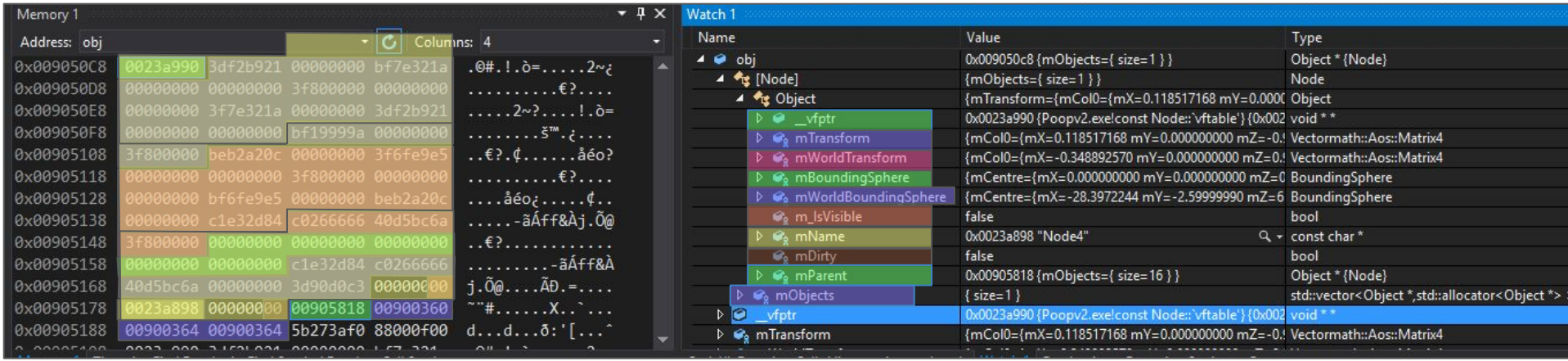
- Why is Matrix4::operator*() the bottleneck?

```
void Modifier::Update()
{
    for(Object* obj : mObjects)
    {
        Matrix4 mat = obj->GetTransform();
        mat = mTransform*mat;
        obj->SetTransform(mat);
    }
}
```

Where Object is

```
protected:
    Matrix4 mTransform;
    Matrix4 mWorldTransform;
    BoundingSphere mBoundingSphere;
    BoundingSphere mWorldBoundingSphere;
    bool m_IsVisible = true;
    const char* mName;
    bool mDirty = true;
    Object* mParent;
};
```

Memory layout for Nodes



Node size = 200 bytes

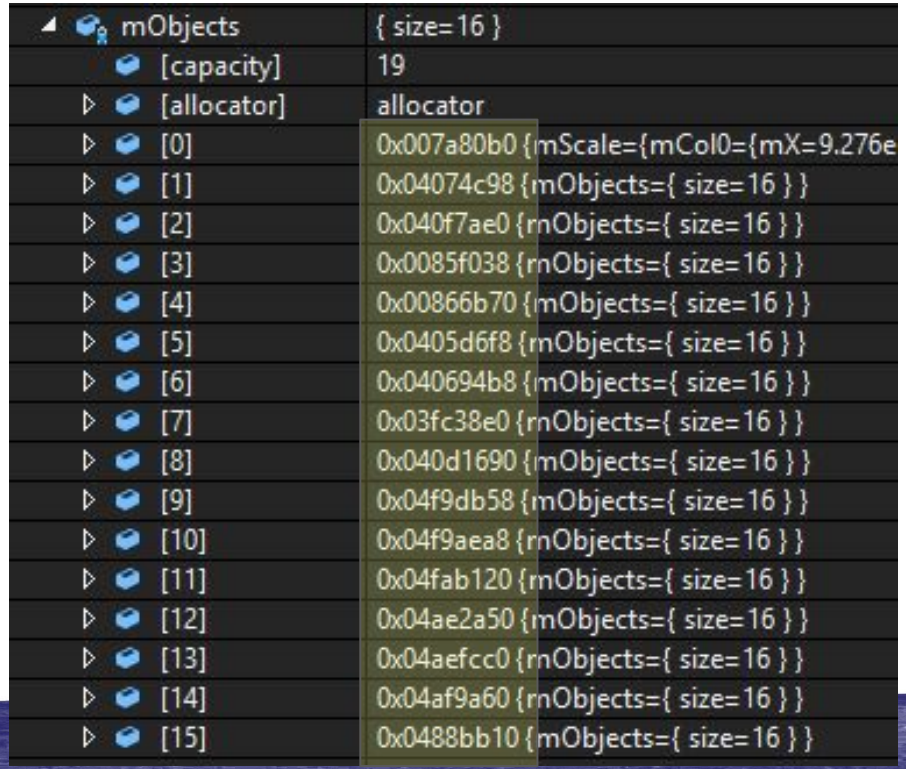
Object size = 188 bytes

Modifer::Update()

Iterates through all its objects.

```
void Modifier::Update()
{
    for(Object* obj : mObjects)
    {
        Matrix4 mat = obj->GetTransform();
        mat = mTransform*mat;
        obj->SetTransform(mat);
    }
}
```

Which are scattered throughout memory.



How do we remove this bottleneck?

- Do less.
- Use less memory.
- Minimise load stalls by making memory access contiguous.
- Or, use prefetching to tell the CPU where the upcoming data will be.
  - Tricky. Pointer chasing, pre-emptive loads, messy code...

- Better off working **with** the HW.

How do we fix it?

Force homogeneous, temporally coherent data to be contiguous

- Memory Pool Managers
- Overload new

“Don’t be clever, be clear”

A simple allocator

```
Manager<Matrix4>          gTransformManager;
Manager<Matrix4>          gWorldTransformManager;
Manager<BoundingSphere>   gBSManager;
Manager<BoundingSphere>   gWorldBSManager;
Manager<Node>             gNodeManager;
```

```
mTransform = gTransformManager.Alloc();
mWorldTransform = gWorldTransformManager.Alloc();
mBoundingSphere = gBSManager.Alloc();
mWorldBoundingSphere = gWorldBSManager.Alloc();
```

```
Matrix4* mTransform              = nullptr;
Matrix4* mWorldTransform         = nullptr;
BoundingSphere* mBoundingSphere  = nullptr;
BoundingSphere* mWorldBoundingSphere = nullptr;
const char* mName                = nullptr;
bool mDirty                      = true;
bool m_IsVisible                 = true;
Object* mParent;
```

sizeof(Node) = 44, sizeof(Object) = 32
(was 200 and 188)

Let’s look at the memory layout now

this	0x0a013350 {mObjects={ size=15 } }
Object	{mTransform=0x03c66170 {mCol0={mX=
mObjects	{ size=15 }
[capacity]	19
[allocator]	allocator
[0]	0x07269040 {mObjects={ size=16 } }
[1]	0x07269080 {mObjects={ size=16 } }
[2]	0x072690c0 {mObjects={ size=16 } }
[3]	0x07269100 {mObjects={ size=16 } }
[4]	0x07269140 {mObjects={ size=16 } }
[5]	0x07269180 {mObjects={ size=16 } }
[6]	0x072691c0 {mObjects={ size=16 } }
[7]	0x07269200 {mObjects={ size=16 } }
[8]	0x07269240 {mObjects={ size=16 } }
[9]	0x07269280 {mObjects={ size=16 } }
[10]	0x072692c0 {mObjects={ size=16 } }
[11]	0x07269300 {mObjects={ size=16 } }
[12]	0x07269340 {mObjects={ size=16 } }
[13]	0x07269380 {mObjects={ size=16 } }
[14]	0x072693c0 {mObjects={ size=16 } }

(((this)->mObjects)[0])->mTransform	0x03c661f0 {
(((this)->mObjects)[1])->mTransform	0x03c66230
(((this)->mObjects)[2])->mTransform	0x03c66270
(((this)->mObjects)[3])->mTransform	0x03c662b0
(((this)->mObjects)[4])->mTransform	0x03c662f0 {
(((this)->mObjects)[5])->mTransform	0x03c66330
(((this)->mObjects)[6])->mTransform	0x03c66370
(((this)->mObjects)[7])->mTransform	0x03c663b0
(((this)->mObjects)[8])->mTransform	0x03c663f0 {
(((this)->mObjects)[9])->mTransform	0x03c66430
(((this)->mObjects)[10])->mTransform	0x03c66470
(((this)->mObjects)[11])->mTransform	0x03c664b0
(((this)->mObjects)[12])->mTransform	0x03c664f0 {
(((this)->mObjects)[13])->mTransform	0x03c66530
(((this)->mObjects)[14])->mTransform	0x03c66570

Now, measure performance...

Previously…

2077 items selected. Slices (2077)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
frame	5,200.614 ms	37.365 ms	17.510 ms	297
Update	1,691.522 ms	1,691.522 ms	5.695 ms	297
GetWBS	1,514.961 ms	1,514.961 ms	5.101 ms	297
Node Render	1,076.607 ms	1,076.607 ms	3.625 ms	297
cull	775.062 ms	775.062 ms	2.610 ms	297
swapbuffers	90.562 ms	90.562 ms	0.306 ms	296
ImGui::Render	31.228 ms	31.228 ms	0.106 ms	296
Totals	**10,380.556 ms**	**5,217.307 ms**	**4.998 ms**	**2077**

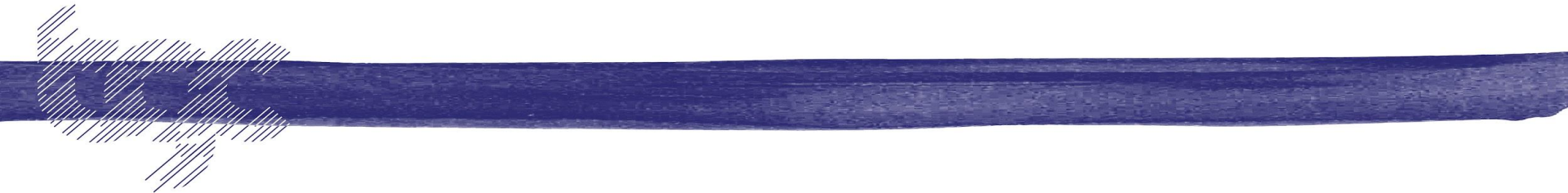
Now…

6482 items selected. Slices (6482)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
frame	8,664.484 ms	120.255 ms	9.347 ms	927
GetWBS	4,741.308 ms	4,741.308 ms	5.120 ms	926
Update	1,441.532 ms	1,441.532 ms	1.557 ms	926
Node Render	1,380.743 ms	1,380.743 ms	1.491 ms	926
cull	591.396 ms	591.396 ms	0.639 ms	925
swapbuffers	283.919 ms	283.919 ms	0.307 ms	926
ImGui::Render	95.686 ms	95.686 ms	0.103 ms	926
Totals	**17,199.068 ms**	**8,654.839 ms**	**2.653 ms**	**6482**

17.5ms -> 9.5ms

No functional code changes.



Now, measure performance...

Previously…

2077 items selected. Slices (2077)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
frame	5,200.614 ms	37.365 ms	17.510 ms	297
Update	1,691.522 ms	1,691.522 ms	5.695 ms	297
GetWBS	1,514.961 ms	1,514.961 ms	5.101 ms	297
Node Render	1,076.607 ms	1,076.607 ms	3.625 ms	297
cull	775.062 ms	775.062 ms	2.610 ms	297
swapbuffers	90.562 ms	90.562 ms	0.306 ms	296
ImGui::Render	31.228 ms	31.228 ms	0.106 ms	296
Totals	**10,380.556 ms**	**5,217.307 ms**	**4.998 ms**	**2077**

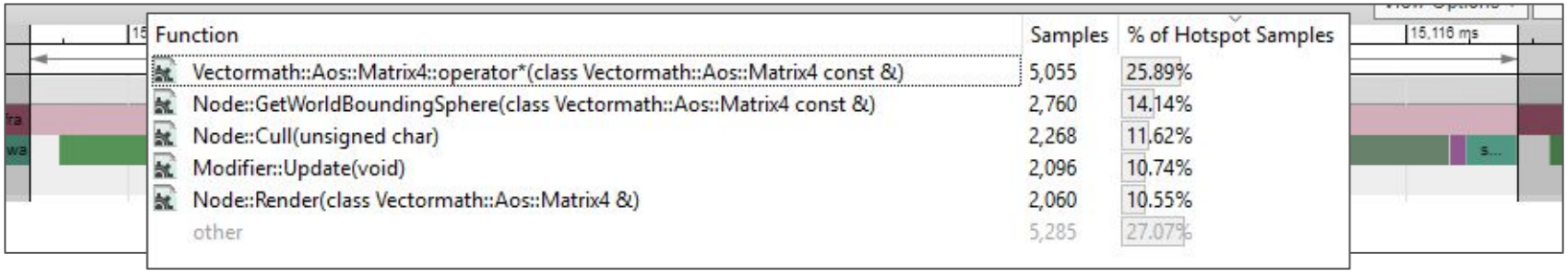
Now…

6482 items selected. Slices (6482)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
frame	8,664.484 ms	120.255 ms	9.347 ms	927
GetWBS	4,741.308 ms	4,741.308 ms	5.120 ms	926
Update	1,441.532 ms	1,441.532 ms	1.557 ms	926
Node Render	1,380.743 ms	1,380.743 ms	1.491 ms	926
cull	591.396 ms	591.396 ms	0.639 ms	925
swapbuffers	283.919 ms	283.919 ms	0.307 ms	926
ImGui::Render	95.686 ms	95.686 ms	0.103 ms	926
Totals	**17,199.068 ms**	**8,654.839 ms**	**2.653 ms**	**6482**

Where are the bottlenecks now?

Previous



Function	Samples	% of Hotspot Samples
Vectormath::Aos::Matrix4::operator*(class Vectormath::Aos::Matrix4 const &)	5,055	25.89%
Node::GetWorldBoundingSphere(class Vectormath::Aos::Matrix4 const &)	2,760	14.14%
Node::Cull(unsigned char)	2,268	11.62%
Modifier::Update(void)	2,096	10.74%
Node::Render(class Vectormath::Aos::Matrix4 &)	2,060	10.55%
other	5,285	27.07%

New

Function	Samples	% of Hotspot Samples
Vectormath::Aos::Matrix4::operator*(class Vectormath::Aos::Matrix4 const &)	11,596	45.55%
Node::Render(class Vectormath::Aos::Matrix4 const &)	2,267	8.91%
Node::GetWorldBoundingSphere(class Vectormath::Aos::Matrix4 const &)	1,905	7.48%
Node::SetVisibilityRecursively(bool)	1,440	5.66%
Modifier::Update(void)	1,296	5.09%
other	6,952	27.31%

Line	Address	Source Code	Code Bytes	Hotspot S	% of Hots	Timer
821		inline const Matrix4 Matrix4::operator *(const Matrix4 & mat …				
822	0x31a660	{		268	2.31%	268
	0x31a660	push ebp	55	54	0.47%	54
	0x31a661	mov ebp,esp	8B EC	176	1.52%	176
	0x31a663	sub esp,38h	83 EC 38	38	0.33%	38
823	0x31a666	return Matrix4(		11,287	97.34%	11,287
	0x31a666	mov			0.01%	1
	0x31a669	movs			0.72%	84
	0x31a66d	movs			0.19%	22
	0x31a672	movss xmm1,[eax+30h]	F3 0F 10 48 30	107	0.92%	107
	0x31a677	movss xmm4,[eax+34h]	F3 0F 10 60 34	704	6.07%	704
	0x31a67c	movaps xmm5,xmm1	0F 28 E9	90	0.78%	90
	0x31a67f	mulss xmm5,xmm0	F3 0F 59 E8	93	0.80%	93
	0x31a683	movss xmm0,[ecx+10h]	F3 0F 10 41 10	381	3.29%	381
	0x31a688	mulss xmm0,xm	0F 59 C4	1	0.01%	1
	0x31a68c	movss xmm3,[eax+	F3 0F 10 58 38	70	0.60%	70
	0x31a691	movss xmm2,[eax+3ch]	F3 0F 10 50 3C			
	0x31a696	addss xmm5,xmm0	F3 0F 58 E8	75	0.65%	75
	0x31a69a	movss xmm0,[ecx+20h]	F3 0F 10 41 20	163	1.41%	163
	0x31a69f	mulss xmm0,xmm3	F3 0F 59 C3	19	0.16%	19
	0x31a6a3	addss xmm5,xmm0	F3 0F 58 E8	4	0.03%	4
	0x31a6a7	movss xmm0,[ecx+30h]	F3 0F 10 41 30	259	2.23%	259
	0x31a6ac	mulss xmm0,xmm2	F3 0F 59 C2	2	0.02%	2
	0x31a6b0	addss xmm5,xmm0	F3 0F 58 E8	14	0.12%	14

Enable Parallel Code Generation	
Enable Enhanced Instruction Set	**Streaming SIMD Extensions 2 (/arch:SSE2)**
Floating Point Model	Precise (/fp:precise)

Where is my SIMD?

Recompile and profile with SIMD

```
#include <GL/gl3w.h>
#include <GLFW/glfw3.h>
#define USE_SSE
#include "vmInclude.h"
```

9.5ms -> 6.2ms

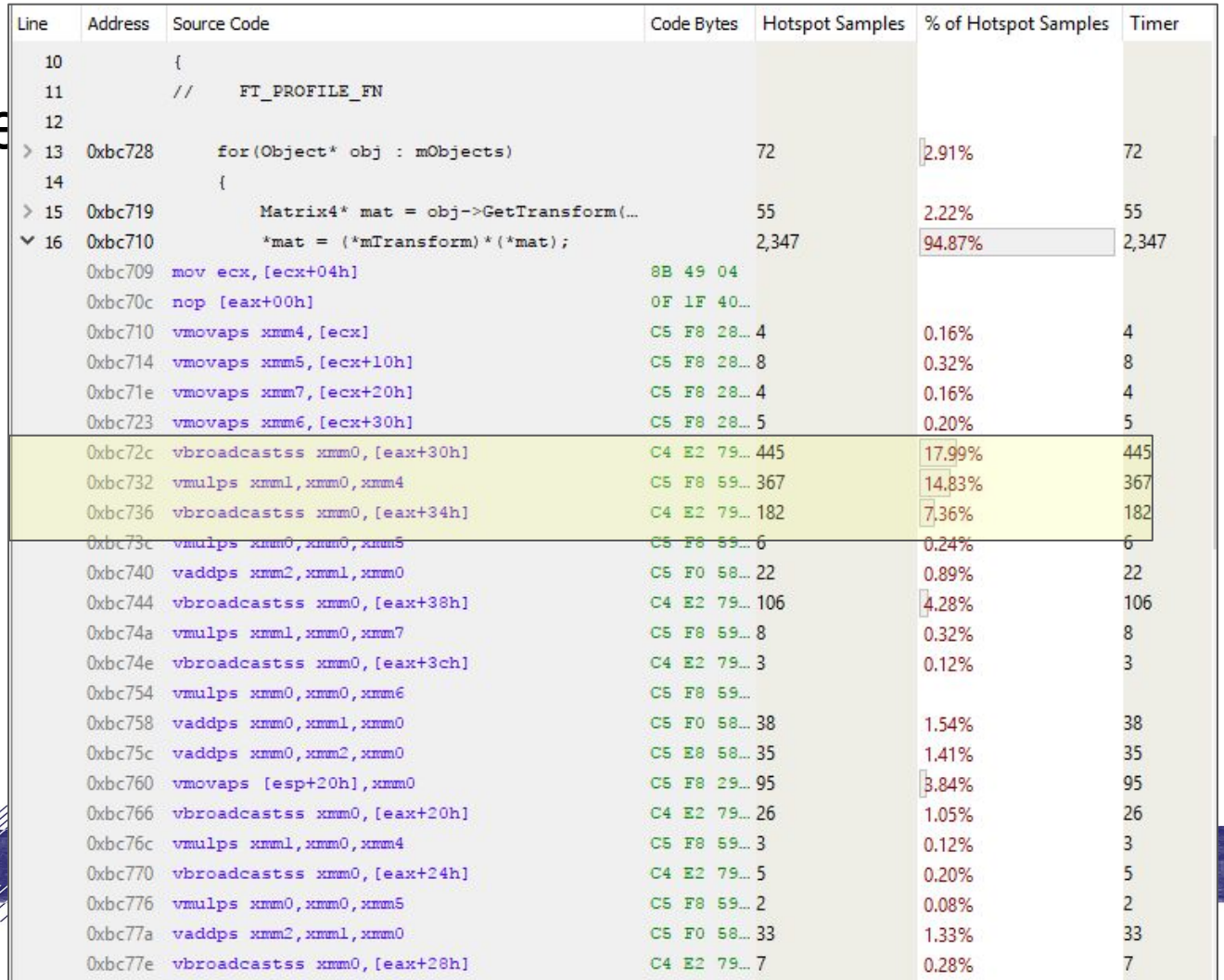
Name	Wall Duration	Self time	Average Wall Duration	Occurrences
frame	100,800.493 ms	1,223.651 ms	6.210 ms	16231
GetWBS	38,329.471 ms	38,329.471 ms	2.362 ms	16229
Node Render	23,318.794 ms	23,318.794 ms	1.437 ms	16230
Update	20,829.673 ms	20,829.673 ms	1.283 ms	16230
cull	10,347.799 ms	10,347.799 ms	0.638 ms	16230
swapbuffers	5,121.740 ms	5,121.740 ms	0.316 ms	16230
ImGui::Render	1,621.210 ms	1,621.210 ms	0.100 ms	16230
Totals	**200,369.180 ms**	**100,792.338 ms**	**1.764 ms**	**113610**

Sampling profile

Function	Samples	% of Hotspot Samples	Module
Node::GetWorldBoundingSphere(class Vectormath::Aos::Matrix4 const &)	4,883	28.13%	Poopv2.exe
Node::Render(class Vectormath::Aos::Matrix4 const &)	3,664	21.10%	Poopv2.exe
Modifier::Update(void)	2,474	14.25%	Poopv2.exe
Node::SetVisibilityRecursively(bool)	1,511	8.70%	Poopv2.exe
Node::Update(void)	1,160	6.68%	Poopv2.exe
other	3,669	21.13%	

- Matrix multiply has disappeared!
  - It’s now small enough to be inlined.

Modifie



Line	Address	Source Code	Code Bytes	Hotspot Samples	% of Hotspot Samples	Timer
10		{				
11		// FT_PROFILE_FN				
12						
13	0xbc728	for(Object* obj : mObjects)		72	2.91%	72
14		{				
15	0xbc719	Matrix4* mat = obj->GetTransform(…		55	2.22%	55
16	0xbc710	*mat = (*mTransform)*(*mat);		2,347	94.87%	2,347
	0xbc709	mov ecx,[ecx+04h]	8B 49 04			
	0xbc70c	nop [eax+00h]	0F 1F 40…			
	0xbc710	vmovaps xmm4,[ecx]	C5 F8 28…	4	0.16%	4
	0xbc714	vmovaps xmm5,[ecx+10h]	C5 F8 28…	8	0.32%	8
	0xbc71e	vmovaps xmm7,[ecx+20h]	C5 F8 28…	4	0.16%	4
	0xbc723	vmovaps xmm6,[ecx+30h]	C5 F8 28…	5	0.20%	5
	0xbc72c	vbroadcastss xmm0,[eax+30h]	C4 E2 79…	445	17.99%	445
	0xbc732	vmulps xmm1,xmm0,xmm4	C5 F8 59…	367	14.83%	367
	0xbc736	vbroadcastss xmm0,[eax+34h]	C4 E2 79…	182	7.36%	182
	0xbc73c	vmulps xmm0,xmm0,xmm5	C5 F8 59…	6	0.24%	6
	0xbc740	vaddps xmm2,xmm1,xmm0	C5 F0 58…	22	0.89%	22
	0xbc744	vbroadcastss xmm0,[eax+38h]	C4 E2 79…	106	4.28%	106
	0xbc74a	vmulps xmm1,xmm0,xmm7	C5 F8 59…	8	0.32%	8
	0xbc74e	vbroadcastss xmm0,[eax+3ch]	C4 E2 79…	3	0.12%	3
	0xbc754	vmulps xmm0,xmm0,xmm6	C5 F8 59…			
	0xbc758	vaddps xmm0,xmm1,xmm0	C5 F0 58…	38	1.54%	38
	0xbc75c	vaddps xmm0,xmm2,xmm0	C5 E8 58…	35	1.41%	35
	0xbc760	vmovaps [esp+20h],xmm0	C5 F8 29…	95	3.84%	95
	0xbc766	vbroadcastss xmm0,[eax+20h]	C4 E2 79…	26	1.05%	26
	0xbc76c	vmulps xmm1,xmm0,xmm4	C5 F8 59…	3	0.12%	3
	0xbc770	vbroadcastss xmm0,[eax+24h]	C4 E2 79…	5	0.20%	5
	0xbc776	vmulps xmm0,xmm0,xmm5	C5 F8 59…	2	0.08%	2
	0xbc77a	vaddps xmm2,xmm1,xmm0	C5 F0 58…	33	1.33%	33
	0xbc77e	vbroadcastss xmm0,[eax+28h]	C4 E2 79…	7	0.28%	7

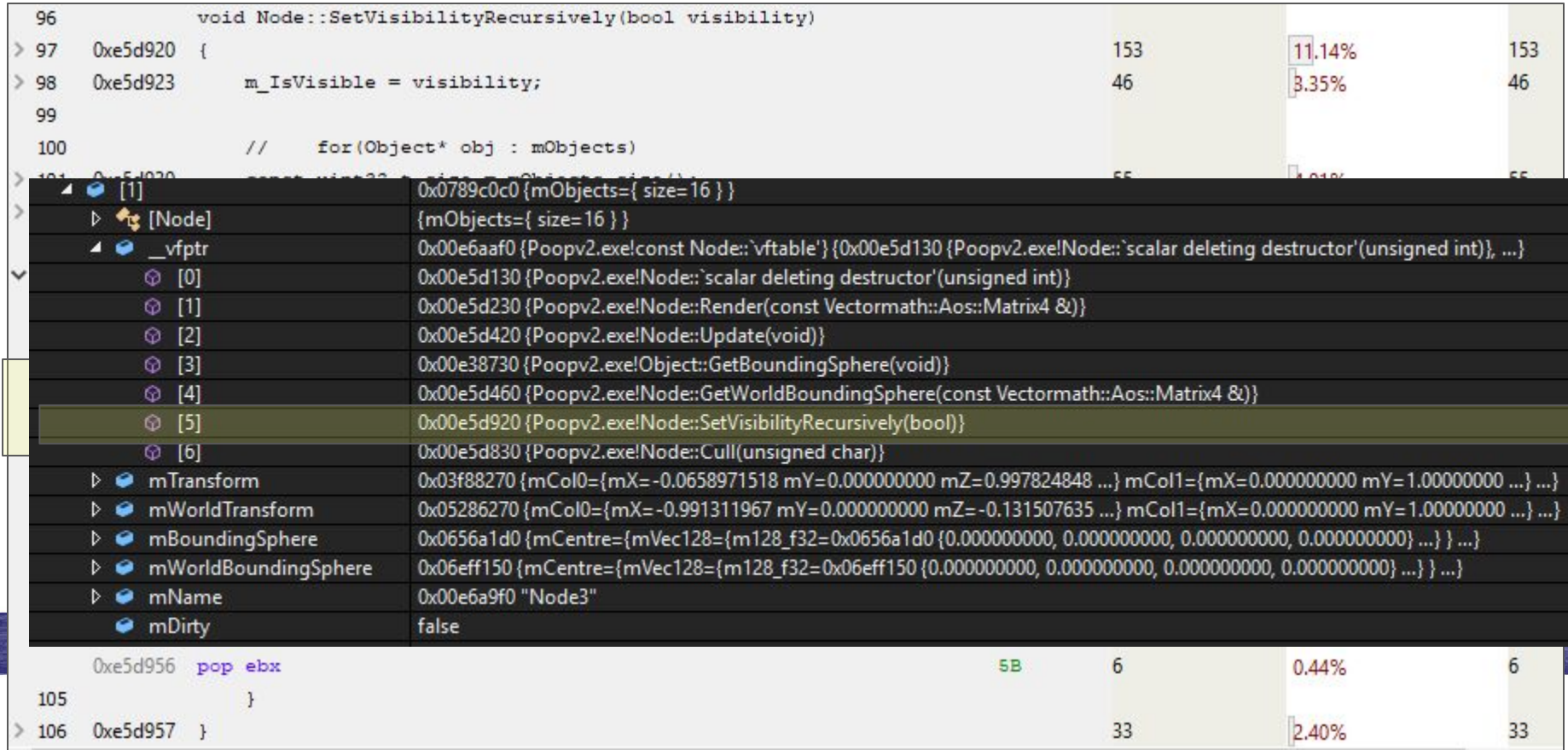
Virtual function overhead

- This was a big issue on PS3.
- Let’s look at SetVisibilityRecursively()

```
void Node::SetVisibilityRecursively(bool visibility)
{
    m_IsVisible = visibility;

    //    for(Object* obj : mObjects)
    const uint32_t size = mObjects.size();
    for (uint32_t i = 0; i < size; i++)
    {
        mObjects[i]->SetVisibilityRecursively(visibility);
    }
}
```

```
virtual void SetVisibilityRecursively(bool visibility)
{
    m_IsVisible = visibility;
}
```


96 void Node::SetVisibilityRecursively(bool visibility)
97 0xe5d920 {
153 11.14% 153
98 0xe5d923 m_IsVisible = visibility;
46 3.35% 46
99
100 // for(Object* obj : mObjects)
[1] 0x0789c0c0 {mObjects={ size=16 } }
[Node] {mObjects={ size=16 } }
__vfptr 0x00e6aaf0 {Poopv2.exe!const Node::`vftable'} {0x00e5d130 {Poopv2.exe!Node::`scalar deleting destructor'(unsigned int)}, ...}
[0] 0x00e5d130 {Poopv2.exe!Node::`scalar deleting destructor'(unsigned int)}
[1] 0x00e5d230 {Poopv2.exe!Node::Render(const Vectormath::Aos::Matrix4 &)}
[2] 0x00e5d420 {Poopv2.exe!Node::Update(void)}
[3] 0x00e38730 {Poopv2.exe!Object::GetBoundingSphere(void)}
[4] 0x00e5d460 {Poopv2.exe!Node::GetWorldBoundingSphere(const Vectormath::Aos::Matrix4 &)}
[5] 0x00e5d920 {Poopv2.exe!Node::SetVisibilityRecursively(bool)}
[6] 0x00e5d830 {Poopv2.exe!Node::Cull(unsigned char)}
mTransform 0x03f88270 {mCol0={mX=-0.0658971518 mY=0.000000000 mZ=0.997824848 ...} mCol1={mX=0.000000000 mY=1.00000000 ...} ...}
mWorldTransform 0x05286270 {mCol0={mX=-0.991311967 mY=0.000000000 mZ=-0.131507635 ...} mCol1={mX=0.000000000 mY=1.00000000 ...} ...}
mBoundingSphere 0x0656a1d0 {mCentre={mVec128={m128_f32=0x0656a1d0 {0.000000000, 0.000000000, 0.000000000, 0.000000000} ...} } ...}
mWorldBoundingSphere 0x06eff150 {mCentre={mVec128={m128_f32=0x06eff150 {0.000000000, 0.000000000, 0.000000000, 0.000000000} ...} } ...}
mName 0x00e6a9f0 "Node3"
mDirty false
0xe5d956 pop ebx 5B 6 0.44% 6
105 }
106 0xe5d957 } 33 2.40% 33

De-inheriting everything

- Decoupled Node from Object
- Changed code to addNode and addObject etc
- Nodes looped over Objects and Nodes separately
- No virtuals!
- How fast?

6.2ms -> 7.6ms

Ah, wat?

- Suspect better branch prediction.
- From asm - not much worse than function call overhead.
- Extra code for looping over nodes and objects broke cache coherence.
- Worthy of further inspection.

“Assume nothing, test everything”


QUESTION
EVERYTHING
WHY?

Prefetching?

- Prefetching is complicated
- Hard to get significant perf improvements
- The HW does a pretty good job if you keep your access patterns simple

Summary

- 17.5ms -> 6.2ms
- No functional changes
  - Memory layout (9.5ms)
  - SIMD use
- Could go even faster
  - 2009 talk reduced everything to flat arrays
  - But, at the cost of flexibility and readability.

Optimisation Process

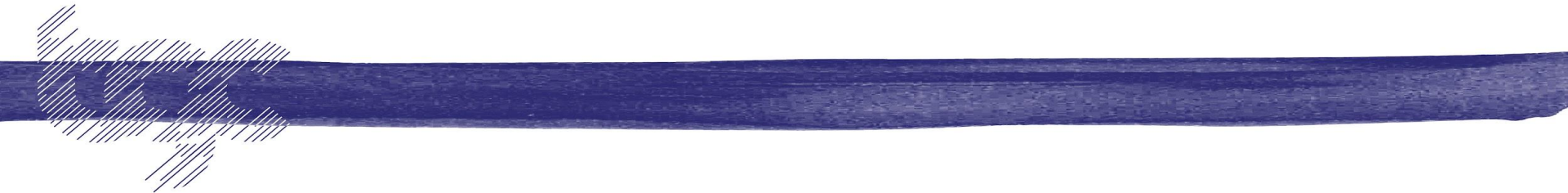
1. Understand your problem.
2. Is there a better algorithm?
3. Can you call it less (or in a different thread)?
4. Understand your data access patterns.
    - Optimise for temporal coherence.
    - Side effect: Easier to parallelise!
5. Then, instruction level optimisation.

Obfuscation by Optimisation

When optimising, aim for simplicity.

Simple code is easy to understand, easy to maintain.

Weigh up costs of complex, highly optimised code - it can be brittle and costly to maintain. Will often be throw away, but can be necessary.



So, is OO bad?

- Encapsulation by
  - logic/function

    vs
  - data

- OO used with foresight
  - Fast
  - Simple
  - Maintainable
- OO used without care
  - Slow
  - Complex
  - Unmaintainable
  - Unoptimisable.

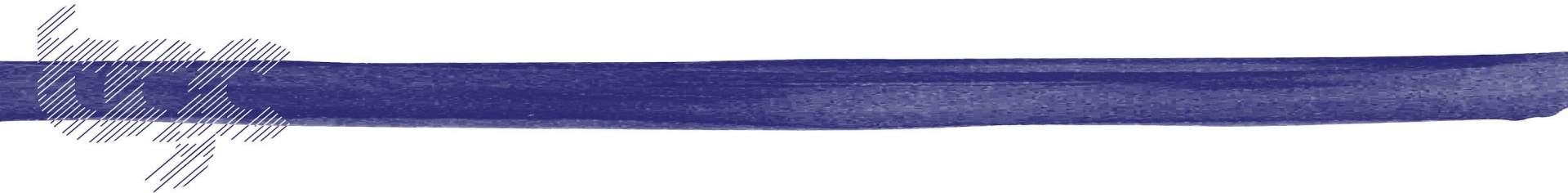
The Language is not your platform

You are not building something to run in C++

You are building something to run on some hardware.

Your language is an abstraction of the HW.

If you need it to run fast, build with the HW in mind.



END

پایان