rics & (depth < Modes

: = inside ? 1 + 1,0 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); D)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (dd

= * diffuse = true;

efl + refr)) && (depth < MAXDEPIN

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly, closel,
if;

radiance = SampleLight(&rand, I, 81, 81) e.x + radiance.y + radiance.z) > 0) 88 (0.000

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Small /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

/INFOMOV/ Optimization & Vectorization

J. Bikker - Sep-Nov 2019 - Lecture 2: "Low Level"

Welcome!





Home

⁺\$

 \sim



t a = nt - nc, b = n t Tr = 1 - (R0 + (1 r) R = (D = nnt - N

* diffuse; = true;

fl + refr)) && (dept

D, N); refl * E * diffuse; = true;

AXDEPTH)



v = true; at brdfPdf = EvaluateDif at3 factor = diffuse * I

at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

1

Mike Conley @mconley@mastodon.social @mike_conley · 13h Are you interested in helping us to make Firefox mind-meltingly fast?

43

仚

ר_

Come work with me!

careers.mozilla.org/position/gh/17...

11 11

Jonathan Adamczewski Retweeted

2 1 21

27





Senior Firefox Performance Engineer

Apply for this job

Engineering: Open Positions Team: Engineering **Cloud Operations Engineer** Engineering Manager, Desktop Firefox Locations: Frontend Remote Canada, Toronto, Vancouver Engineering Manager, WebAssembly Firefox Security Student Worker Why join Mozilla engineering? (Werkstudent) 🚸 Slack | Ebilkill | Paral... 😰 lecture2 - low level.... \land 🖮 🌈 🗘 💿 Nop Pro - I... Senior Firefox Perf... M ENG 09:30 **INFOMOV** 🖳 #optimization-and...

rics & (depth < Modes

: = inside ? 1 + 1,0 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); D)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (dd

= * diffuse = true;

efl + refr)) && (depth < MAXDEPIN

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly, closel,
if;

radiance = SampleLight(&rand, I, 81, 81) e.x + radiance.y + radiance.z) > 0) 88 (0.000

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Small /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

/INFOMOV/ Optimization & Vectorization

J. Bikker - Sep-Nov 2019 - Lecture 2: "Low Level"

Welcome!



Previously in INFOMOV...

Consistent Approach

: = inside 7 1 : ... ht = nt / nc, ddh os2t = 1.0f - nnt 0, N); >)

nt a = nt - nc, b = 00 nt Tr = 1 - (R0 + (1 - 80 r) R = (D = nnt - N = (0

* diffuse; = true;

: fl + refr)) && (depth < MAXDEP

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly close
f;
radiance = SampleLight(&rand, I, &L, &L
ext + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, a<u>t3 fact</u>or = diffuse * INVPI;

at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf) * (

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdi urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

(0.)	Determine optimization requirements
1.	Profile: determine hotspots
2.	Analyze hotspots: determine scalability
3.	Apply high level optimizations to hotspots
4.	Profile again.
5.	Parallelize
6.	Use GPGPU
7.	Profile again.
8.	Apply low level optimizations to hotspots
9.	Repeat steps 7 and 8 until time runs out
10.	Report.



tics & (depth < Moose)

: = inside ? | | | ht = nt / nc, ddn os2t = 1.0f - nmt O, N); 0)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - RC) fr) R = (D = nnt - N = (dd)

= * diffuse = true;

. efl + refr)) && (depth < MAXDEDIU

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (double)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- The Cost of a Line of Code
- CPU Architecture: Instruction Pipeline
- Data Types and Their Cost
- Rules of Engagement



Instruction Cost

tics ≰j(depth < MOCD

: = inside 7 1 1 1 ht = nt / nc, ddn bs2t = 1.0f - nnt 7 2, N); 3)

at a = nt - nc, b = n at Tr = 1 - (R0 + (1 - nc fr) R = (D = nnt - N - (3

= * diffuse; = true;

-=fl + refr)) && (depth < //

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffue estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, 2.x + radiance.y + radiance.<u>7) > 0) 20</u>

w = true; at brdfPdf = EvaluateDiffuse(L, N) * at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sov /ive)

```
,
t33 brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, south
urvive;
.pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

What is the 'cost' of a multiply?

starttimer();
float x = 0;
for(int i = 0; i < 1000000; i++) x *= y;
stoptimer();</pre>

- Actual measured operations:
 - timer operations;
 - initializing 'x' and 'i';
 - comparing 'i' to 1000000 (x 1000000);
 - increasing 'i' (x 100000);
 - jump instruction to start of loop (x 100000).
 - Compiler outsmarts us!
 - No work at all unless we use x
 - x += 1000000 * y

Better solution:

- Create an arbitrary loop
- Measure time with and without the instruction we want to time



Instruction Cost

nics ≹j(depth < NOCCS

= inside ? 1 ; 1 ; ht = nt / nc, ddn bs2t = 1.0f - nnt 0, N); ∂)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (ddr

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Pst at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Soc /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

What is the 'cost' of a multiply?

float x = 0, y = 0.1f; unsigned int i = 0, j = 0x28929227; for(int k = 0; k < ITERATIONS; k++)</pre>

// ensure we feed our line with fresh data
x += y, y *= 1.01f;
// integer operations to free up fp execution units
i += j, j ^= 0x17737352, i >>= 1, j /= 28763;
// operation to be timed
if (with) x *= y;
// integer operations to free up fp execution units
i += j, j ^= 0x17737352, i >>= 1, j /= 28763;

dummy = x + (float)i;

	Generate Graph of Include Files	
X	Show Call Stack on Code Map	Ctrl+Shift+`
ta	Insert Snippet	Ctrl+K, Ctrl+X
ta	Surround With	Ctrl+K, Ctrl+S
Ξ	Peek Definition	Alt+F12
1	Go To Definition	F12
÷≣	Go To Declaration	Ctrl+Alt+F12
	Find All References	
- 72	View Call Hierarchy	Ctrl+K, Ctrl+T
	Toggle Header / Code File	Ctrl+K, Ctrl+O
	Intel Advisor XE 2015	•
	Breakpoint	•
60	Add Watch	
60	Add Parallel Watch	
60	QuickWatch	Shift+F9
	Pin To Source	
	View Array	
→	Show Next Statement	Alt+Num *
k	Run To Cursor	Ctrl+F10
-le-	Run Flagged Threads To Cursor	
1	Set Next Statement	Ctrl+Shift+F10
Ģ	Go To Disassembly	
ж	Cut	Ctrl+X
ŋ	Сору	Ctrl+C
ĉ	Paste	Ctrl+V
	Outlining	•
	Intel Compiler	•
	Options	Ctrl+1
	Source Control	•

Instruction Cost

tics & (depth < Modes

: = inside 7 1 1 1 nt = nt / nc, ddn ss2t = 1.0f - nnt), N);))

nt a = nt - nc, b = nt nt Tr = 1 - (R0 + (1 - R0 nr) R = (D ⁺ nnt - N - (d

= * diffuse; = true;

. fl + refr)) && (depth < MAXDEPTH

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diff
estimation - doing it properly,
if;
radiance = SampleLight(&rand, I, 8
2.x + radiance.y + radiance.z) > 0)

w = true; at brdfPdf = EvaluateDiffuse(

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpd urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

x86 assembly in 5 minutes

Modern CPUs still run x86 machine code, based on Intel's 1978 8086 processor. The original processor was 16-bit, and had 8 'general purpose' 16-bit registers*:

AX ('accumulator register')	AH, AL (8-bit)	EAX (32-bit)	RAX (64-bit)
BX ('base register')	BH, BL	EBX	RBX
CX ('counter register')	CH, CL	ECX	RCX
DX ('data register')	DH, DL	EDX	RDX
BP ('base pointer')		EBP	RBP
SI ('source index')		ESI	RSI
DI ('destination index')		EDI	RDI
SP ('stack pointer')		ESP	RSP
		st0st7	R8R15
		XMM0XMM7	XMM0XMM15
			YMM0YMM15
* More info: <u>http://www.swansontec</u>	.com/sregisters.html		ZMM0ZMM31





Instruction Cost

x86 assembly in 5 minutes:

Typical assembler:

dec ecx

fld st0

faddp

jnz loop

loop:

at a = nt - nc, b

efl + refr)) && (depth < MAX

), N); refl * E * diffuse; = true;

(AXDEPTH)

survive = SurvivalProbability(diff radiance = SampleLight(&rand, I, &L,

v = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf) at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely foll /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &p urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true:

mov eax, [0x1008FFA0] // read from address into register // shift eax 5 bits to the right shr eax, 5 add registers, store in eax add eax, edx decrement ecx // jump if not zero fld [esi] // load from address [esi] onto FPU // duplicate top float add top two values, push result

More on x86 assembler: http://www.cs.virginia.edu/~evans/cs216/guides/x86.html A bit more on floating point assembler: https://www.cs.uaf.edu/2007/fall/cs301/lecture/11_12_floating_asm.html



Instruction Cost

What is the 'cost' of a <u>multiply</u>?

float x = 0, y = 0.1f;unsigned int i = 0, j = 0x28929227;for(int k = 0; k < ITERATIONS; k++)</pre>

at a = nt

), N); refl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability if; radiance = SampleLight(&rand e.x + radiance.y + radiance.z)

v = true: at brdfPdf = EvaluateDiffus at3 factor = diffuse * INV* at weight = Mis2(directPdd briffing y = x + (float)i;

E * ((weight * cosThetaOut) / directPd

andom walk - done properly, closely follow /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, & urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true:



Instruction Cost

nics & (depth < Moccos

:= inside 7 1 1 1 0 nt = nt / nc, ddn us2t = 1.0f - nnt D, N); 2)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N ⁻ (dd)

= * diffuse; = true;

efl + refr)) && (depth < MODEP)

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly
f;
radiance = SampleLight(&rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (doing

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (();

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdi urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

What is the 'cost' of a multiply?

Observations:

- Compiler reorganizes code
- Compiler cleverly evades division
- Loop counter *decreases*
- Presence of integer instructions affects timing (to the point where the mul is free)

But also:

It is really hard to measure the cost of a line of code.



Instruction Cost

ics & (depth < NoCCO

: = inside ? 1 1 1 1 ht = nt / nc, ddh 4 ps2t = 1.0f - nnt 4 D, N); 2)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - Rc Fr) R = (D * nnt - N * (d)

* diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse) estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &l) 2.x + radiance.y + radiance.z) > 0) && ()

w = true; at brdfPdf = EvaluateDiffuse(L, N at3 factor = diffuse * INVPI;

ht weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, Bpdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

What is the 'cost' of a single instruction?

Cost is highly dependent on the surrounding instructions, and many other factors. However, there is a 'cost ranking':

bit shifts simple arithmetic, logical operands multiplication division

sin, cos, tan, pow, exp

<< >>

+ - & | ^

*

sqrt

This ranking is generally true for any processor (including GPUs).



INFOMOV – Lecture 2	– "Low Level"	Instruction	Operands	Ops	Latency	Reciprocal throughput	Execution unit	Notes	14
Instruction (Cost	Arithmetic instructions ADD, SUB ADD, SUB ADD, SUB ADC, SBB	s r,r/i r,m m,r r,r/i	1 1 1	1 1 7 1	1/3 1/2 2,5 1/3	ALU ALU, AGU ALU, AGU ALU		
<pre>ics (depth < MUXDEFTH) = inside ? 1 + 1.2* t = nt / nc, den = 40 s2t = 1.0f - nst = mt , N);) t a = nt - nc, b = nt = mt t Tr = 1 - (R0 + (1 - R0) r) R = (D * nnt - N * (ddm) * diffuse; = true;</pre>	AMD K7	ADC, SBB ADC, SBB CMP F CMP INC, DEC, NEG INC, DEC, NEG AAA, AAS DAA DAA DAS AAD AAD F AAM F MUL, IMUL F	lath SQRT SIN COS SINCOS PTAN PATAN SCALE XTRACT 2XM1 YL2X YL2XP1		. 1	1 1/2 1 44 90 51 90 76 10 46 10 72 16 5 7 8 49 63	35 12 -100 -100 0-150 0-200 0-170 8 11 27 126 147	FMUL	
<pre>fl + refr)) && (depth < MAXDEPTH) , N); efl * E * diffuse; = true; AXDEPTH) urvive = SurvivalProbability(diffuse estimation - doing it properly, closed f; adiance = SampleLight(&rand, I, &L, &light()</pre>	1999	MUL, IMUL MUL, IMUL IMUL IMUL IMUL IMUL IMUL DIV DIV DIV IDIV IDIV	r16/m16 r32/m32 r16,r16/m16 r32,r32/m32 r16,(r16),i r32,(r32),i r16,m16,i r32,m32,i r8/m8 r16/m16 r32/m32 r8 r16 r16 r32 m8	3 3 2 2 2 2 3 3 32 47 79 41 56 88 42	3 4 3 4 5 24 24 24 40 17 25 41	2 3 2 2,5 1 2 2 2 2 2 3 23 40 17 25 41	ALU0_1 ALU0_1 ALU0 ALU0 ALU0 ALU0 ALU0 ALU0 ALU ALU ALU ALU ALU ALU	dx=4	
ts brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, a rvive; pdf; = E * brdf * (dot(N, R) / pdf); ion = toue;		IDIV	m16 m32	57 89	25 41	25	ALU		A A A A A A A A A A A A A A A A A A A

Instruction Cost

), N); refl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability(diffus radiance = SampleLight(&rand, I, &L e.x + radiance.y + radiance.z) > 0) 88

v = true; at brdfPdf = EvaluateDiffuse(L, N) * F at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, clos $\mathsf{Note}:\mathsf{Tw}$ /ive) execut at3 brdf = SampleDiffuse(diffuse, go to different execution pipes urvive;

at;						
= E	* brdf	(dot(N, R		pdf);	
on =	= true:					

MD	Jaguar	
	2013	

MD Jaguar	
2013	
o micro-operations can simultaneously if they	

nstruction	Operan	ds	Ops	Latency	Reciproc throughp	al ut	Execution pipe	on	No	otes		
Arithmetic instructions	S										1	
DD, SUB	r,r/i		1	1	0.5		10/1					
DD, SUB	r,m		1		1							
DD, SUB	m,r		1	6	1							
DC, SBR	r r/i		1	1	1		I0/1			I		
DC, Math												
DC, FSQRT				1	35		35		FP1			
MP FLDPI, etc.				1			1		FP0			
MP FSIN				4-44	30-139	3	30-151	F	P0, FP1			
NC, D FCOS				11-51	38-93			F	P0, FP1			
NC, D FSINCOS				11-76	55-122	5	55-180	F	P0, FP1			
AA FPTAN				11-45	55-177	5	55-177	F	P0, FP1			
AS FPATAN				9-75	44-167	4	14-167	FI	P0, FP1			
FSCALE				5	27		FF		P0, FP1			
AS FXTRACT				7	9		6	F	P0, FP1			
F2XM1				8	32-37			FI	P0, FP1			
AM FYL2X				8-51	30-120	3	30-120	F	P0, FP1			
IUL. I FYL2XP1				61	~160		~160	FI	P0, FP1			
IUL, IMUL	r16/m1	6	3	3	3	_	10					
IUL, IMUL	r32/m3	2	2	3	2		10					
IUL, IMUL	r64/m6	4	2	6	5		10					
MUL	r16,r16/n	n16	1	3	1		10					
MUL	r32,r32/n	n32	1	3	1		10					
MUL	r64,r64/n	n64	1	6	4		10					
MUL	r16,(r16	i),i	2	4	1		10					
MUL	r32,(r32	?),i	1	3	1		10					
MUL	r64,(r64	-),i	1	6	4		10					
DIV	r8/m8		1	11-14	11-14		10					
DIV	r16/m1	6	2	12-19	12-19		10					
DIV	r32/m3	2	2	12-27	12-27		10					
DIV	r64/m6	4	2	12-43	12-43		10					4
DIV	r8/m8		1	11-14	11-14		10					R A
DIV	r16/m1	6	2	12-19	12-19		10					1S
DIV	r32/m3	2	2	12-27	12-27		10					
DIV	r64/m6	4	2	12-43	12-43		10					



Instruction Cost

Intel Silvermont

), N); refl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability(diffu radiance = SampleLight(&rand, I, &L, e.x + radiance.y + radiance.z) > 0) 8

v = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

Note: This is a low-power /ive) processor (ATOM class).

2014

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

	Operands	pops	Unit	Latency	throughput	Remarks		
Arithmetic instructions								
ADD SUB	r,r/i	1	IP0/1	1	1/2			
ADD SUB	r,m	1	IP0/1, Mem		1			
ADD SUB	m,r/i	1	IP0/1, Mem	6	1			
ADC SBB	r,r/i	1	IP0/1	2	2			
ADC SBB	r,m	1			2			
ADC SBB	m,r/i	1		6	2			
CMP	r,r/i	1	IP0/1	1	1/2			
CMP	m,r/i	1			1			
INC DEC	· · ·	1 1	1 IP0/1	۱ _۱ ۹	1/2	atonov to flag	-2	
NEG NOT Math								
INC DEC FSCALE			27			66		
AAA FXTRACT			15		20	20		
AAS FSQRT			1		13-40	13-40		
DAA FSIN FCOS			18		40-170	40-170		
DAS FSINCOS			110		40-170			
AAD F2XM1			9		39-90			
AAM EVL2X			34		80-140			
MULIMUL EVLOYE1			61		154			
MULIMUL FTLZAPT			101		154			
MULIMUL FPTAN			101		45-200			
MULIMUL FPATAN			63		85-190			
IMUL	r16,r16	2	IP0	4	4			
IMUL	r32,r32	1	IP0	3	1			
IMUL	r64,r64	1	IP0	5	2			
IMUL	r16,r16,i	2	IP0	4	4			
IMUL	r32,r32,i	1	IP0	3	1			
IMUL	r64,r64,i	1	IP0	5	2			
MUL IMUL	m8	3	IP0					
MUL IMUL	m16	5	IP0					
MUL IMUL	m32	4	IP0					
MUL IMUL	m64	4	IP0	14				
DIV	r/m8	9	IP0, FP0	24	19			
DIV	r/m16	12	IP0, FP0	25-29	19-23			
DIV	r/m32	12	IP0, FP0	25-39	19-31			nº
DIV	r/m 64	23	IP0, FP0	34-94	25-94			s in the second se
IDIV	r/m8	26	IP0, FP0	24-35	25			
IDIV	r/m16	29	IP0, FP0	37-41	30-32			5
IDIV	r/m32	29	IP0, FP0	29-46	29-38			25/2
IDIV	r/m64	44	IP0, FP0	47-107	47-107			

16

Instruction Cost

nics & (depth < ™00

c = inside ? 1 : . . ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); ⊅)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - RC Fr) R = (D = nnt - N = (d)

* diffuse; = true:

. :fl + refr)) && (depth < MAXDE

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && closed

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvice at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (radi

andom walk - done properly, closely following Sec. /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Intel Skylake 2015

ADD SUB		r,r/i	1	1	p0156	5	1	0.25				17
ADD SUB		r,m	1	2	p0156 p	23		0.5				
ADD SUB		m,r/i	2	4	2p0156 2p2	37 p4	5	1				
ADC SBB		r,r/i	1	1	p06		1	1				
ADC SBB		r,m	2	2	p06 p2	3		1				
ADC SBB		m,r/i	4	6	3p0156 2p2	37 p4	5	2				
CMP		r,r/i	1	1	p0156	6	1	0.25				
CMP		m,r/i	1	2	p0156 p	23	1	0.5				
INC DEC NE NOT	G	r	1	1	p0156	6	1	0.25				
INC DEC NO	т	m	3	4	p0156 2p2	37 p4	5-6	1				
NEG		m	2	4	p0156 2p2	37 p4	5-6	1				
AAA			2	2	p1 p56	6	4		not 64	bit		
AAS	1		^	2	4 - 67		1			L 1		
DAA DAS	AA DAS Math											
AAD	FSC	ALE			27	27	·			130	130	
AAM	FXT	RACT			17	17	·			11	11	
MUL IMU	FSC	RT			1	1		p0	1	4-21	4-7	
MUL IML	FSI	N			53-105				50	0-120		
MUL IMU	FCC)S			53-105				50	0-130		
MUL IMU	FSI	VCOS			55-120				5	5-150		
MUL IMU	F2X	M1			16-90				6	5-80		
MUL IMU	FYL	2X			40-100					103		
MUL IMU	FYL	2XP1			56					77		
MUL IML	FPT	AN			40-112				14	0-160		
IMUL	FPA	TAN			30-160				10	0-160		
IMUL	<u> </u>	r.m	1	2	p1 p23	3		1				
IMUL		r16,r16,i	2	2	p1 p015	56	4	1				
IMUL		r32,r32,i	1	1	p1		3	1				
IMUL		r64,r64,i	1	1	p1		3	1				
IMUL		r16,m16,i	2	3	p1 p0156	p23		1				
IMUL		r32,m32,i	1	2	p1 p23	3		1			×.	194
IMUL		r64,m64,i	1	2	p1 p23	3		1			s.	
MULX		r32,r32,r32	3	3	p1 2p05	56	4	1	BMI	2		S S
MULX		r32,r32,m32	3	4	p1 2p056	p23		1	BMI	2	1	
MULX		r64,r64,r64	2	2	p1 p5	-	4	1	BMI	2	1	NI. AN
						1	I					

Instruction Cost

What is the 'cost' of a single instruction?

The cost of a single instruction depends on a number of factors:

- The arithmetic complexity (sqrt > add);
- Whether the operands are in register or memory;
- The size of the operand (16 / 64 bit is often slightly slower);
- Whether we need the answer immediately or not (latency);
- Whether we work on signed or unsigned integers (DIV/IDIV).

D, N); refl * E * diffuse; = true;

efl + refr)) && (depth)

AXDEPTH)

/ive)

at a = nt

survive = SurvivalProbability(diffuse estimation - doing it properly, closely, adiance = SampleLight(&rand, I, &L, &light() e.x + radiance.y + radiance.z) > 0) && (doing) w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvive at brdfPdf = EvaluateDiffuse(L, N) * Psurvive at stat factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (radia andom walk - done properly, closely following set andom walk - done properly, closely following set

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, #pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

On top of that, certain instructions can be executed simultaneously.

tics & (depth < Moose)

: = inside ? | | | ht = nt / nc, ddn os2t = 1.0f - nmt O, N); 0)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - RC) fr) R = (D = nnt - N = (dd)

= * diffuse = true;

. efl + refr)) && (depth < MAXDEDIU

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (double)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- The Cost of a Line of Code
- CPU Architecture: Instruction Pipeline
- Data Types and Their Cost
- Rules of Engagement



Pipeline

CPU Instruction Pipeline

nics & (depth < Morran

: = inside ? | ht = nt / nc, ddn hs2t = 1.0f - nnt n), N);))

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N = (dd)

= * diffuse; = true;

efl + refr)) && (depth < MAXDE

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light 2.x + radiance.y + radiance.z) > <u>0) &&</u>

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

CPI = 4

Fetch

3.

4.

Decode

Execute

Writeback

andom walk - done properly, closely foll /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Get the instruction from RAM The byte code is decoded The instruction is executed

Instruction execution is typically divided in four phases:

The results are written to RAM/registers



fldz xor ecx, ecx

fld dword ptr ds:[405290h] mov edx, 28929227h fld dword ptr ds: [40528Ch] push esi mov esi, 0C350h add ecx, edx mov eax, 91D2A969h xor edx, 17737352h shr ecx, 1 mul eax, edx fld st(1) faddp st(3), st mov eax, 91D2A969h shr edx, 0Eh add ecx, edx fmul st(1),st <u>xor edx, 17737352h</u> shr ecx, 1 mul eax, edx shr edx, 0Eh dec esi jne tobetimed<0>+1Fh



nics ≹j(depth < NOCC

: = inside } 1 () 1 ht = nt / nc, ddn () ps2t = 1.0f - nnt () p, N); ∂)

at a = nt - nc, b = Nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D - nnt - N - (100

= * diffuse; = true;

-:fl + refr)) && (depth < NAXDEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light 2.x + radiance.y + radiance.z) > 0) && closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psu at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sovi /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

CPU Instruction Pipeline

For each of the stages, different parts of the CPU are active. To use its transistors more efficiently, a modern processor overlaps these phases in a *pipeline*.



At the same clock speed, we get four times the throughput (CPI = IPC = 1).



tics **(depth** < MODE=

: = inside ? 1 + ... ht = nt / nc, ddn bs2t = 1.0f - nnt 0, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N = (d0)

= * diffuse; = true;

. efl + refr)) && (depth < NACOS

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, If; radiance = SampleLight(&rand, I, &L, &lin e.x + radiance.y + radiance.z) > 0) && (d)

w = true; at brdfPdf = EvaluateDiffuse(L,) at3 factor = diffuse * INVPI; at weight = Mic3(directOdf _ brdf)

at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely /ive)

```
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apdi )
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

CPU Instruction Pipeline

Maximum clockspeed is determined by the most complex of the four stages. For higher clockspeeds, it is advantageous to increase the number of stages (thereby reducing the complexity of each individual stage).

			_	_								
			E	E	E							
				Е	Е	Е						
					Е	Е	Е					
						F	F	F				
						_	-	-	_			
							E	E	E			
								Е	Е	E		

Obviously, 'execution' of different instructions requires different functionality.

Superpipelining allows higher clockspeeds and thus higher throughput, but it also increases the latency of individual instructions.

Stages

7	PowerPC G4e
8	Cortex-A9
10	Athlon
12	Pentium Pro/II/III, Athlon 64
14	Core 2, Apple A7/A8
14/19	Core i2/i3 Sandy Bridge
16	PowerPC G5, Core i*1 Nehalem
18	Bulldozer, Steamroller
20	Pentium 4
31	Pentium 4E Prescott



CPU Instruction Pipeline

E

nics & (depth < AvxCC+⊺

= inside 7 1 0 1... ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); 3)

at a = nt - nc, b = Nt = Nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (30)

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEP⊺

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly.closed
if;
radiance = SampleLight(%rand, I, %L, %l);
e.x + radiance.y + radiance.z) > 0) %% (0)

v = true; at brdfPdf = EvaluateDiffuse(L, N) = at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf) = ()

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdio urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true: Here, one execution unit handles floats; one handles integer; one handles memory operations.

Since the execution logic is typically the most complex part, we might just as well duplicate the other parts:

Different execution units for different (classes of) instructions:





ics & (depth < ₩0000

: = inside 7 1 0 1.0 ht = nt / nc, ddn = 0 ps2t = 1.0f - nnt = nn D, N); 3)

ut a = nt - nc, b = nt nt Tr = 1 - (R0 + (1 - R0 Tr) R = (D [#] nnt - N ^{*} (ddm -

= * diffuse; = true;

efl + refr)) && (depth < MAXDE

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &lig
2.x + radiance.y + radiance.z) > 0) && Closed

w = true; at brdfPdf = EvaluateDiffuse(L, N) P: at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely follo /ive)

, t33 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pdf urvive; .pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

CPU Instruction Pipeline

IPC = 3 (or: ILP = 3)

This leads to the *superscalar* processor, which can execute multiple instructions in the same clock cycle, assuming not all instruction require the same execution logic.





Pipeline

145

z = inside ? 1 ht = nt / nc, ddh bs2t = 1.0f - nnt D, N); ∂)

at a = nt - nc, b = (R0 + (1 - R0))at Tr = 1 - (R0 + (1 - R0)Tr) R = (D^{-1} nnt - N

= * diffuse = true;

efl + refr)) && (depth < MODEPTI

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly, close
f;
radiance = SampleLight(&rand, I, &L,)
e.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pourvis at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * ()

andom walk - done properly, closely following Sec. /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

CPU Instruction Pipeline

Using a pipeline has consequences. Consider the following situation:



Here, the second instruction needs the result of the first, which is available one clock tick too late. As a consequence, the pipeline stalls briefly.



Pipeline

tics & **(dept**h < Mode

: = inside ? 1 + 1 ... ht = nt / nc, ddn - 1 ... bs2t = 1.0f - nnt - nn 0, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (d0n)

= * diffuse = true;

efl + refr)) && (depth < MODEPTI

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed df; radiance = SampleLight(&rand, I, &L, &L e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) * P at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf) * (rail

andom walk - done properly, closely following Smil /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true:

CPU Instruction Pipeline

Using a pipeline has consequences. Consider the following situation:



a = b * c; jump if a is not zero

In this scenario, a conditional jump makes it hard for the CPU to determine what to feed into the pipeline after the jump.



rics & (depth < Monos

: = inside 7 1 1 1 0 ht = nt / nc, ddn 0 0 ss2t = 1.0f - nnt 0 n 2, N); ≥)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - Rc) Fr) R = (D = nnt - N = (dom

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPIN

), N); refl * E * diffuse; = true;

(AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, & .x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (

andom walk - done properly, closely following sour /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, %pdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

CPU Instruction Pipeline - Digest

For a more elaborate explanation of the pipeline, see this document: http://www.lighterra.com/papers/modernmicroprocessors

Or check this very detailed study of the Nehalem architecture:

The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms, Thomadakis, 2011.

For now:

- A compiler reorganizes code to prevent latencies
- Feeding mixed code provides the compiler with sufficient opportunities for shuffling Branching issues need to be prevented manually



tics & (depth < Moose)

: = inside ? | | | ht = nt / nc, ddn os2t = 1.0f - nmt O, N); 0)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - RC) fr) R = (D = nnt - N = (dd)

= * diffuse = true;

. efl + refr)) && (depth < MAXDEDIU

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (double)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- The Cost of a Line of Code
- CPU Architecture: Instruction Pipeline
- Data Types and Their Cost
- Rules of Engagement



Data Types



(*note: -1 = 0xfffffff*) at weight = Mis2(directPdf, brdfPdf) E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely follo /ive)

at cosThetaOut = dot(N, L);

), N);

AXDEPTH)

v = true;

if;

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, & urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Data Types

Data types in C++

float

Value:

: * diffuse; = true;

fl + refr)) && (depth <)

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffu estimation - doing it properly, is if; radiance = SampleLight(&rand, I, &L 2.x + radiance.y + radiance.z) > 0)

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Pst at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sec /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Size: 32 bit (4 bytes)

Exponent:8 bit; $-127 \dots 128$ Mantissa:23 bit; $0 \dots 2^{23} - 1$

sign * mantissa * 2^exponent

Exercise: write a function that replaces array $a = \{0.5, 0.25, 0.125, 0.0625, ...\}$.

30

Data Types

nics & (depth < Mox⊡

: = inside ? 1 + 1 0 ht = nt / nc, ddn − bs2t = 1.0f - nnt 2, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Fr) R = (D ⁼ nnt - N ⁻ (ddn

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTII)

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closel
if;
radiance = SampleLight(&rand, I, &L, &L)
e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psu at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sm /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2*: More on <u>http://www.catb.org/esr/structure-packing</u> prvive; pdf;

pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

double char, unsigned char short, unsigned short LONG LONG LONG, __int64 bool

Data types in C++

Padding*:

};

struct Test

unsigned int u; bool flag;

sizeof(Test) is 8

64 bit (8 bytes) 8 bit 16 bit 32 bit (same as int) 64 bit 8 bit (!)

```
struct Test2
{
    double d;
    bool flag;
};
// sizeof( Test2 ) is 16
```


31

Data Types

ics & (depth < No:

c = inside ? | . . . ht = nt / nc, ddn os2t = 1.0f - nnt 2, N); 2)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (d

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTHIL

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly
ff;
radiance = SampleLight(&rand, I, &L, &I
e.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following or /ive)

. t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Data types in C++ - Conversions

Explicit:

float fpi = 3.141593; int pi = (int)(1024.0f * fpi);

Implicit:

struct Color { unsigned char a, r, g, b; }; Color bitmap[640 * 480]; for(int i = 0; i < 640 * 480; i++)</pre>

bitmap[i].r *= 0.5f; bitmap[i].g *= 0.5f; bitmap[i].b *= 0.5f;

// bitmap[i].r *= 0.5f;

	-
IOVZX	eax,byte ptr [ecx-1]
IOV	dword ptr [ebp-4],eax
ild	dword ptr [ebp-4]
nstcw	word ptr [ebp-2]
IOVZX	eax,word ptr [ebp-2]
r	eax,0C00h
IOV	dword ptr [ebp-8],eax
mul	st,st(1)
ldcw	word ptr [ebp-8]
istp	dword ptr [ebp-8]
IOVZX	eax,byte ptr [ebp-8]
IOV	byte ptr [ecx-1],al

Data Types

ics & (depth < No:

: = inside ? | : | : ht = nt / nc, ddn bs2t = 1.0f - nmt 2, N); 2)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - Rc Tr) R = (D = nnt - N = (d

= * diffuse; = true;

-:fl + refr)) && (depth < NACCEPTIC

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly close
if;
radiance = SampleLight(&rand, I, &L, &l
e.x + radiance.y + radiance.z) > 00 &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Ps at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following. /ive)

```
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

Data types in C++ - Conversions

Explicit:

float fpi = 3.141593; int pi = (int)(1024.0f * fpi);

Avoiding conversion:

```
struct Color { unsigned char a, r, g, b; };
Color bitmap[640 * 480];
for( int i = 0; i < 640 * 480; i++ )</pre>
```

bitmap[i].r >>= 1; bitmap[i].g >>= 1; bitmap[i].b >>= 1; // bitmap[i].r >>= 1; shr byte ptr [eax-1],1 // bitmap[i].g >>= 1; shr byte ptr [eax],1 // bitmap[i].b >>= 1; shr byte ptr [eax+1],1

Data Types

ics & (depth < ∧∞

: = inside ? 1 : 1 ∂ ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); ð)

at a = nt - nc, b = n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N - (0

= * diffuse = true;

efl + refr)) && (depth < NOODEPIII)

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &l, &lig
2.x + radiance.y + radiance.z) > 0) && ()

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) *

andom walk - done properly, closely following Sec. /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Data types in C++ - Conversions

Explicit:

float fpi = 3.141593; int pi = (int)(1024.0f * fpi);

Avoiding conversion (2):

struct Color { union { struct { unsigned char a, r, g, b; }; int argb; }; }; Color bitmap[640 * 480]; for(int i = 0; i < 640 * 480; i++)</pre>

bitmap[i].argb = (bitmap[i].argb >> 1) & 0x7f7f7f;

Data Types

Data types in C++ - Free interpretation

Trick: Cheaper float comparison

= * diffuse; = true;

fl + refr)) && (depth < NAX

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, close if; radiance = SampleLight(&rand, I, &L, e.x + radiance.y + radiance.z) > 0) &

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psu at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Source /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true:

union { float v1; unsigned int u1; }; union { float v2; unsigned int u2; };

bool smaller = (v1 < v2);

bool smaller = (u1 < u2); // same result, if signs of v1 and v2 are equal.

Data Types

Data types in C++ - Rolling your own

HDR color storage

tics & (depth < Moose)

: = inside ? | | | ht = nt / nc, ddn os2t = 1.0f - nmt O, N); 0)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - RC) fr) R = (D = nnt - N = (dd)

= * diffuse = true;

. efl + refr)) && (depth < MAXDEDIU

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (double)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- The Cost of a Line of Code
- CPU Architecture: Instruction Pipeline
- Data Types and Their Cost
- Rules of Engagement

nics & (depth < NACCS

at a = nt - nc, b = Mt = M at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (ddM

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPII

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light()
2.x + radiance.y + radiance.z) > 0) && closed

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (rad);

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Common Opportunities in Low-level Optimization

RULE 1: Avoid Costly Operations

- Replace multiplications by bitshifts, when possible
- Replace divisions by (reciprocal) multiplications
- Avoid sin, cos, sqrt

Common Opportunities in Low-level Optimization

Adapt previous results (interpolation, reprojection, ...)

RULE 2: Precalculate

Loop hoisting

Lookup tables

Reuse (partial) results

- at a = nt nc, b = nt = nt at Tr = 1 - (R0 + (1 - R0) Tr) R = (D = nnt - N = (d0)
- = * diffuse; = true;
- efl + refr)) && (depth < MAXDE
- D, N); refl * E * diffuse; = true;
- AXDEPTH)
- survive = SurvivalProbability(diffuse)
 estimation doing it properly, closed
 if;
 radiance = SampleLight(&rand, I, &L, &light)
 e.x + radiance.y + radiance.z) > 0) && closed
- w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurviva at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (rad
- andom walk done properly, closely following Sec. /ive)
- ; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Common Opportunities in Low-level Optimization

RULE 3: Pick the Right Data Type

- Avoid byte, short, double
- Use each data type as a 32/64 bit container that can be used at will
- Avoid conversions, especially to/from float
- Blend integer and float computations
- Combine calculations on small data using larger data

), N); refl * E * diffuse; = true;

AXDEPTH)

at a = nt

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed of
if;
radiance = SampleLight(&rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (double)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Poundive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (read);

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

40

Common Opportunities in Low-level Optimization

Use lookup tables to prevent conditional code

If all else fails: make conditional branches predictable

Try to split loops with conditional paths into multiple unconditional loops

RULE 4: Avoid Conditional Branches

if, while, ?, MIN/MAX

Use loop unrolling

- at a = nt nc, b = nt at Tr = 1 - (R0 + (1 - R0 Ir) R = (D = nnt - N = (dd)
- = * diffuse; = true;
- efl + refr)) && (depth < ≀
- D, N); refl * E * diffuse; = true;
- AXDEPTH)
- survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && (closed)
- w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvice at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (rad);
- andom walk done properly, closely following Source /ive)
- ; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

41

Rules of Engagement

char c = 'p';

int position = -1;

if (a[t] == c)

position = t;

Common Opportunities in Low-level Optimization

RULE 5: Early Out

char a[] = "abcdfghijklmnopqrstuvwxyz";

for (int t = 0; t < strlen(a); t++)

```
:= nt / nc, ddn
:2t = 1.0f - nnt
N );
```

```
- 000,
```

fl + refr)) && (depth < MAXDEPT

```
D, N );
refl * E * diffuse;
= true;
```

```
AXDEPTH)
```

```
survive = SurvivalProbability( diffuse
estimation - doing it properly, closed
Hf;
radiance = SampleLight( &rand, I,
e.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.y + radiance.z) > 0
& & closed
.x + radiance.z) >
```

```
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurviv
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (reight);
```

andom walk - done properly, closely following Sec. /ive)

```
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, dpdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

```
char a[] = "abcdfghijklmnopqrstuvwxyz";
char c = 'p';
int position = -1, len = strlen( a );
for ( int t = 0; t < len; t++ )
{
    if (a[t] == c)
    {
        position = t;
        break;
    }
}
```


Common Opportunities in Low-level Optimization

RULE 6: Use the Power of Two

- A multiplication / division by a power of two is a (cheap) bitshift
- A 2D array lookup is a multiplication too make 'width' a power of 2
- Dividing a circle in 256 or 512 works just as well as 360 (but it's faster)
- Bitmasking (for free modulo) requires powers of 2

. efl + refr)) && (depth < MACDEPIID

D, N); refl * E * diffuse; = true;

(AXDEPTH)

at a = nt

survive = SurvivalProbability(diffu .estimation - doing it properly, co if; radiance = SampleLight(&rand, I, &L .x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psures at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (r)

andom walk - done properly, closely following sour /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

1-2-4-8-16-32-64-128-256-512-1024-2048-4096-8192-16384-32768-65536

Be fluent with powers of 2 (up to 2^{16}); learn to go back and forth for these: $2^{9} = 512 = 2^{9}$. Practice counting from 0..31 on one hand in binary.

Common Opportunities in Low-level Optimization

RULE 7: Do Things Simultaneously

t = nt / nc, ddn s2t = 1.0f - nnt , N);)

at a = nt - nc, b = Nt at Tr = 1 - (R0 + (1 - R6) Fr) R = (D ⁺ nnt - N ⁻ (30

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPIII)

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light 2.x + radiance.y + radiance.z) > 0) && closed

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (Psi

andom walk - done properly, closely following Small /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true:

Use those cores

- An integer holds four bytes; use these for instruction level parallelism
- More on this later.

Common Opportunities in Low-level Optimization

- 1. Avoid Costly Operations
- 2. Precalculate
- 3. Pick the Right Data Type
- 4. Avoid Conditional Branches
- 5. Early Out
- 6. Use the Power of Two
- 7. Do Things Simultaneously

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) &&

v = true; t brdfPdf = EvaluateDiffuse(L, N) * Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (rad

andom walk - done properly, closely following Small /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

tics & (depth < Moose)

: = inside ? | | | ht = nt / nc, ddn os2t = 1.0f - nmt O, N); 0)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - RC) fr) R = (D = nnt - N = (dd)

= * diffuse = true;

. efl + refr)) && (depth < MAXDEDIU

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (double)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- The Cost of a Line of Code
- CPU Architecture: Instruction Pipeline
- Data Types and Their Cost
- Rules of Engagement

Practice

nics & (depth < MaxDea

: = inside ? 1 | 1.3 ht = nt / nc, ddn - 1 ps2t = 1.0f - nnt - 1 D, N); 3)

at a = nt - nc, b = nt = nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (dom

* diffuse; = true;

. :fl + refr)) && (depth < MAXDEP

D, N); refl * E * diffuse; = true;

(AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly
if;
radiance = SampleLight(&rand, I, &L, &l
e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI;

at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (r

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Get (from the website) project glassball.zip

Using low-level optimization, speed up this application.

- . Avoid Costly Operations
- 2. Precalculate
- 3. Pick the Right Data Type
- 4. Avoid Conditional Branches
- 5. Early Out
- 6. Use the Power of Two

Make sure functionality remains intact. Target: a 10x speedup (this should be easy).

tics € (depth < Modes

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - Rc) Γr) R = (D = nnt - N = (30)

= * diffuse; = true;

. :fl + refr)) && (depth < NOCCEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse); estimation - doing it properly, closed Hf;

radiance = SampleLight(&rand, I, &L 2.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (rad

andom walk - done properly, closely following Small /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

/INFOMOV/

END of "Low Level"

next lecture: "caching (1)"

