rics & (depth < ∧∞coo

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

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

= * diffuse = true;

efl + refr)) && (depth < MAXDED

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed) ff; and are - Sampleicht(Sample - Sample)

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

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) (real

andom walk - done properly, closely following Smool /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 1: "Introduction"

Welcome!



ics & (depth < Modelin

: = inside ? 1 + 1 4 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (00)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPID

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 e.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) (1860)

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

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

Today's Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling



Why?

ics (depth < Modean

= inside ? 1 0 1 0 t = nt / nc, ddn s2t = 1.0f - nnt , N);)

at a = nt - nc, b = 11 at Tr = 1 - (R0 + (1 - 10 Some problems require the supercomputer of the future.





E * ((weight * cosThetaOut) / directPdf)

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

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



Why?

fics ↓ (depth < MoxDennie t = inside ? 1 ht = nt / nc, ddn ss2t = 1.0f - nnt -5, N);

= * diffuse; = true; -

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

AXDEPTH)

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

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) * (r);

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

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

AlphaGo Parallel, ELO rating 3140 Running on 1202 CPUs, 176 GPUs

Some problems require the supercomputer of the future.

Anything that depends on Moore's Law and time to become feasible.







Why?

Games want to raise the bar.

• More, better, faster. Also: be scalable.



E * ((weight * cosThetaOut) / directPdf)

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

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

EXCLUSIVE

Why?

Introduction

nics & (depth < PV000000

: = inside ? 1 ; 1 ; 1 ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); ∂)

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

* diffuse; = true;

. :fl + refr)) && (depi

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

AXDEPTH)

survive = SurvivalProbabi estimation - doing it pro af; radiance = SampleLight(& O e.x + radiance.y + radiance

w = true; at brdfPdf = EvaluateDiff(at3 factor = diffuse * INN at weight = Mis2(directPd at cosThetaOut = dot(N, 1 E * ((weight * cosThetaOut) / directPd

andom walk - done properly, closely folic /ive)

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

Some software needs to run on pretty weak hardware.

Limited CPU, limited RAM (limited controls).







Why?

•••	< > Q Search
G Home	
Browse	Local Files
((0)) Radio	5 songs, 23 min
YOUR LIBRARY	PLAY
Made For You	
Recently Played	Q Filter
Songs	TITLE 🔨
Albums	Don't Stop The Sandman
Artists	It's the Right Time
Stations	
Local Files	Photograph (Live)
Videos	Roll You In The Hurricane
Podcasts	Shook Me Like A Prayer
* ((weight * cosT	hetaOut) / directPdf)

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



Some software should not use 90% of your CPU.

Leave room for other applications, be invisible.





Why?

fics ≹ (depth < Modes

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

= * diffuse = true;

. fl + refr)) && (depth

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

AXDEPTH)

survive = SurvivalProbab estimation - doing it p if; radiance = SampleLight(e.x + radiance.y + radia

w = true; at brdfPdf = EvaluateDiff at3 factor = diffuse * IN at weight = Mis2(directF at cosThetaOut = dot(N, E * ((weight * cosThetaOut)

andom walk - done properly, closely fol /ive)

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

Sometimes the cheapest / lowest power CPU is the best.

• What is the lowest end CPU this will still run on? Can we go lower?









INFOMOV – Lecture 1 – "Introduction"

Introduction

at a = nt

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

AXDEPTH)

survive = SurvivalProbability(diff. if; radiance = SampleLight(&rand, I,)

e.x + radiance.y + radiance.z) > 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) / directPd

andom walk - done properly, closely -/ive)

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

Why?

Waiting is annoying.

- Turning on your digital camera
- Getting a train ticking at the vending machine
- Copying files to a USB stick
- Windows updates

...

...

Copying 30,379 items from Desktop to Copy_test 95% complete Name: Untitled Game (v02).fm

Time remaining: About 30 seconds Items remaining: 1,649 (154 MB)

Fewer details

95% complete ×



Working on updates

11% complete

Don't turn off your computer





What is optimization?

Part of it is:

- INFOB3CC Concurrency
 - INFONW Computerarchitectuur en netwerken
 - INFOB3TC Talen en compilers

= true; = true;

at Tr = 1

- efl + refr)) && (depth < MAXDEPTI
- D, N); refl * E * diffuse; = true;

AXDEPTH)

- survive = SurvivalProba estimation - doing it if; radiance = SampleLight(e.x + radiance.y + radi
- w = true; at brdfPdf = EvaluateDif at3 factor = diffuse * I at weight = Mis2(direct ..., at cosThetaOut = dot(N, L);
- E * ((weight * cosThetaOut) / directPdf) * (rac

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

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

And of course: any course that deals with improving existing algorithms.

Specific purpose of INFOMOV:

- To gain understanding of performance aspects of the hardware we use;
- To gain an intuition for what affects performance;
- To learn to apply a structured process to improve performance.





What is optimization?

Think like a CPU

- Instruction pipelines
- Latencies
- Dependencies
- Bandwidth
- Cycles
- Floating point versus integer
- SIMD

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

), N);

AXDEPTH)

refl * E * diffuse;

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvivo 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)

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



What is optimization?

Work smarter, not harder: algorithm scalability

- Big O
- Research: not reinventing the wheel
- Data characteristics & algorithm choice
- STL, Boost: Trust No One
- As accurate as necessary (but not more)
- Balancing accuracy, speed and memory

AXDEPTH)

), N);

= true;

refl * E * diffuse;

at Tr = 1

survive = SurvivalProbability(diff f: radiance = SampleLight(&rand, I, e.x + radiance.y + radiance.z) > 0 v = true: at brdfPdf = EvaluateDiffuse(| Instruction pipelines at3 fa at wei Latencies at cos Dependencies Bandwidth /ive) Cycles • Floating point versus integer at3 bro irvive SIMD pdf;

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

TRUST NO ONE



INFOMOV – Lecture 1 – "Introduction"

Introduction

What is optimization?

Memory hierarchy: caches

- Cache architecture
- Cache lines
- Hits, misses and collisions
- **Eviction policies**
- Prefetching
- **Cache-oblivious**
- Data-centric programming

survive = SurvivalProbability(diff f: radiance = SampleLight(&rand, I .x + radiance.y + radiance.z) >

at Tr = 1

= true:

), N);

= true;

(AXDEPTH)

sion = true:

refl * E * diffuse;



- Instruction pipelines
- Latencies Dependencies
- Bandwidth
- Cycles
- Floating point versus integer
- Big O RUST NO ONE
 - Research: not reinventing the wheel Data characteristics & algorithm choice STL: Trust No One As accurate as necessary (but not more)
 - Balancing accuracy, speed and memory





What is optimization?

Don't assume, measure

- Profilers
- Interpreting profiling data
- Instrumentation
- Steering optimization effort

- Bottlenecks

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

(AXDEPTH)

pdf;

sion = true

at Tr = 1

- survive = SurvivalProbability(diff f: radiance = SampleLight(&rand, I e.x + radiance.y + radiance.z) >
- v = true; at brdfPdf = EvaluateDiffuse(at3 fa at wei at cos /ive) at3 bro irvive

1 = E * brdf * (dot(N, R) / pdf);

- Instruction pipelines
- Latencies
- Dependencies Bandwidth
- Cycles
- Floating point versus integer
- SIMD



- Big O
- Research: not reinventing the wheel
- Data characteristics & algorithm choice STL: Trust No One
 - As accurate as necessary (but not more)
 - Balancing accuracy, speed and memory

d .	Analysis Target A Analysis Type	spots viewpoint (<u>chang</u> e 🖾 Collection Log 🗈 Sun	nmary 🛷 Bottom-up 🛷	Caller/Callee	🗞 Top-down Tree 📧 Platfo		D
Gro	ouping: Task Domain / Task Ty	/pe / Task Duration Type	/ Function / Call Stack	ĸ	×.	× 0	1. 1.
Tas	k Domain / Task Type / Task	CPU Time 🔻 🚿	Instructions Retired	CPI Rate	CPU Frequency Ratio	Task Time	- ^
V L	JE4Domain	0.441s	1,796,600,000	0.861	1.352	13.549)s
	FDrawSceneCommand	0.030s	127,400,000	0.857	1.400	0.030)s
Þ	RenderViewFamily	0.028s 🚦	122,200,000	0.851	1.429	0.029)s
>	FDeferredShadingSceneRer	0.028s 🚦	122,200,000	0.851	1.429	0.029)s
Þ	FEngineLoopTick	0.019s	72,800,000	0.893	1.316	0.064	ls
Þ	FrameTime	0.019s	72,800,000	0.893	1.316	0.063	3s
۶	GameEngine Tick	0.009s	46,800,000	0.889	1.778	0.011	ls
Þ	FDeferredShadingSceneRer	0.008s	33,800,000	0.923	1.500	0.008	3s
Þ	FQueuedThread::Run.WaitF	0.008s	2,600,000	9.000	1.125	4.188	3s
Þ	InitViews	0.008s	33,800,000	0.923	1.500	0.008	3s
Þ	Game thread tick wait time	0.008s	08s 23,400,000 1.000		1.125	0.051	ls 🗸
٢	· · · · · · · · · · · · · · · · · · ·	< Contraction of the second se					>
	p: + - r r		20850ms		✓ Thread		⊽ ^
sad	Thread (TID: 181916)				A Runni	ng	
置 Thread (TID: 170200)					CPU 1	ime od Ouedea	
	Thread (TID: 173608)	And a			Spin a	na Overne. CLK LINH	4.5
	Thread (TID: 171728)				✓ Task	out_onti	725
	Thread (TID: 170044)	The second	e ver	VVV			





- Cache architecture
- **Cache** lines
- Hits, misses and collisions
- **Eviction policies**
- Prefetching
- Cache-oblivious





What is optimization? – Project Management

Keeping code maintainable

- Pareto principle / 80-20 rule: roughly 80% of the effects are caused by 20% of the causes.
- 1% of the code takes 99% of the time.

"The curse of premature optimization"

- Optimization, rule 1: "Don't do it".
- Rule 2 (for experts only!), "Don't do it yet".

Optimization as a deliberate process

Get predictable gains using a consistent approach.





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

= * diffuse; = true;

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

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) * 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 Sour /ive)

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

What is optimization?

"Perceived Performance"

-), N); refl * E * diffuse;

AXDEPTH)

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

andom walk - done properly, closely followi /ive)

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

- Wait for user input 1.
- Respond to user input *as quickly as possible* 2.
- Execute requested operation. 3.





At the end of this course:

You will know how to speed up critical code by a factor 2.5x to 25x (and more).

- You will be able to do this to virtually any program*.
- Your understanding of higher-level optimization approaches will increase.
- You will be able to apply these principles to new / alien hardware.
- You will have a more intimate relationship with your computer.

In other words:

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

efl + refr)) && (depth <

AXDEPTH)

at a = nt

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

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvivo 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 Sacial /ive) * disclaimer: 'tl

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

We will talk a lot about the 'C' in O(N).

* disclaimer: 'that has not been optimized by an expert'.



ics & (depth < Modelin

: = inside ? 1 + 1 4 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (00)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPID

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 e.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) (1860)

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

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

Today's Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling



ics ≰(depth < Notis-

: = inside ? 1 : 1 ht = nt / nc, ddn ps2t = 1.0f - nmt D, N); ∂)

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

= * diffuse; = true;

efl + refr)) {

D, N); refl * E * diffus⊄ = true;

(AXDEPTH)

survive = Surviva estimation - doin Hf; radiance = Sample e.x + radiance.y -

w = true; at brdfPdf = Evalu at3 factor = diffu at weight = Mis2(at cosThetaOut = (E * ((weight * cosmetaout)

andom walk - done properly, closel ⁄ive)

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









Lecturer

Jacco Bikker <u>j.bikker@uu.nl</u> Room 4.24 BBG

ics & (depth < ≥0000

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

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

= * diffuse; = true;

efl + refr)) && (depth < H

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

AXDEPTH)

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

w = 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 Sec /ive)

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

Course Layout

8 weeks + exam week:

- 2 lectures per week (for exceptions: see website)
- 1 guest lecture (I hope)
- Lectures start at 09:00...
- Working class PART 1 starts at 09:00, lecture at 10:00. 🙂
- Working class PART 2 starts at 12:00.

Assessment:

- 2 assignments (25% each, individual or pairs);
- 1 final assignment (50%, individual or pairs);
- 1 final theory exam (individual).

Before we start today's lecture, I would like to discuss the anoying habit of some of you of arriving late every single time





tics & (depth < ⊅0000

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

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPID

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

AXDEPTH)

survive = SurvivalProbability(diffu .estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L 2.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 Soul /ive)

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

Prerequisites

C++ English

Hardware / software

You'll need access to a computer with a CPU that supports SSE2 and OpenCL. Obtaining VTune (Intel CPU) or CodeXL (AMD CPU) is beneficial (VTune is free for students). We will use Visual Studio 2017/19 (community edition).

Other tools will (also) be free.



ics & (depth < Mo⊙

c = inside ? 1 ()) ht = nt / nc, ddn >s2t = 1.0f - nnt >, N); >)

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

= * diffuse; = true;

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

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

AXDEPTH)

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

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

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

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

Literature

No book! But that doesn't mean you won't be reading.

Main documents:

Agner Fog, 2004-2019, "Optimizing Software in C++" (also see his website: <u>http://agner.org</u>)

Ulrich Drepper, 2007, "What Every Programmer Should Know About Memory"

You are encouraged to do research into specific topics of interest yourself, and to report on this in class.





OptmzdSummaries™



: = inside } | | | | | ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); ∂)

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

= * diffuse; = true;

. :fl + refr)) && (depth ≪ MAXDEDI

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

AXDEPTH)

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

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) * (red

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:



New: overview of the lecture material, for some lectures (goal is a full set by next year).

These will become available on the website.





at a = nt - nc,

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

AXDEPTH)

survive = SurvivalProbability(diffu if; 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)

andom walk - done properly, closely follo /ive)

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

Audience

Any computer science student (with a slight bias towards games)

Make sure <u>you</u> get as much as possible out of this course. This automatically includes a free pass.





ics & (depth < Modelin

: = inside ? 1 + 1 4 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (00)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPID

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 e.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) (1860)

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

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

Today's Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling



Consistent Approach

(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 / vectorize / use GPGPU
- 6. Profile again.
- 7. Apply low level optimizations to hotspots
- 8. Repeat step 6 and 7 until time runs out
- 9. Report.

if; radiance = SampleLight(&rand, I, &L, &ligh e.x + radiance.y + radiance.z) > 0) && (doub

survive = SurvivalProbability(diff)

), N);

= true;

AXDEPTH)

refl * E * diffuse;

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvis 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, 8pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

at a = ni

), N);

= true;

(AXDEPTH)

v = true:

/ive)

f:

efl + refr)) && (depth <

survive = SurvivalProbability(diff

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

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

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely foll

at cosThetaOut = dot(N, L);

refl * E * diffuse;

Consistent Approach

(0.) Determine optimization requirements

- Target hardware (or range of hardware)
- Target performance
- Time available for optimization
- Constraints related to maintainability / portability
- 1. Profile: determine hotspots
- 2. Analyze hotspots: determine scalability
- 3. Apply high level optimizations to hotspots
 - Profile again.

4.

5.

6.

8.

9.

....

- Parallelize / vectorize / use GPGPU
- Profile again.
- 7. Apply low level optimizations to hotspots
 - Repeat steps 6 and 7 until time runs out
 - Report.

From here on, we will assume that:

- the code is 'done' (feature complete);
- a speed improvement is required;
- we have a finite amount of time for this.



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

Consistent Approach

(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 / vectorize / use GPGPU
- 6. Profile again.
- 7. Apply low level optimizations to hotspots
- 8. Repeat steps 6 and 7 until time runs out
- 9. Report.

if; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (dou

survive = SurvivalProbability(diff)

), N);

= true;

AXDEPTH)

refl * E * diffuse;

w = 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)

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

7.

8.

9.

Overview

at a = nt

), N);

= true;

(AXDEPTH)

v = true;

/ive)

if;

refl * E * diffuse;

survive = SurvivalProbability(diff

radiance = SampleLight(&rand, I, &l

e.x + radiance.y + radiance.z) > 0)

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 follo

Consistent Approach

- (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 / use GPGPU
- 6. Profile again.
 - Apply low level optimizations to hotspots
 - caching, data-centric programming,
 - removing superfluous functionality and precision,
 - aligning data to cache lines, vectorization,
 - checking compiler output, fixed point arithmetic,
 - ...

Repeat steps 6 and 7 until time runs out Report.

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

29

INFOMOV – Lecture 1 – "Introduction"

Overview

ics (depth < Modes

: = inside ? | : . . . ht = nt / nc, ddn ss2t = 1.0f - nnt), N); >)

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

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPT

D, 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) 24

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psur 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, apdf 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 / vectorize / use GPGPU
- 6. Profile again.
- 7. Apply low level optimizations to hotspots
- 8. Repeat steps 6 and 7 until time runs out
 - Report.

9.



Profiling

Basic Low Level

Cache & Memory

Data-centric

Compilers

Fixed-point Arithmetic

CPU architecture

SIMD





nics & (depth < Mo⊙SA

: = inside ? l : . . ht = nt / nc, ddn os2t = 1.0f - nnt 2, N); 3)

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

= * diffuse; = true;

efl + refr)) && (depth < M/

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) 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 Small /ive)

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

Assembler

In this course, we will <u>not</u> write assembler:

- It takes a pro to outperform the compiler
- You will be fighting the compiler
- You will have to redo the optimization for every target processor
- Maintainability will be zero.







nics & (depth < ≥voces

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

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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 && (dot)

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) * (ref

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:

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%." (Donald Knuth)



nics & (depth < Modes

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Ir) R = (D = nnt - N = (00)

= * diffuse = true;

efl + refr)) && (depth < MONDEPTIO

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

AXDEPTH)

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)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true: "A significant improvement in performance can often be achieved by solving only the actual problem and removing extraneous functionality." (Wikipedia)



nics **& (dept**h < Mode

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

at a = nt - nc, b = 00 at Tr = 1 - (R0 + (1 - 80 Fr) R = (D ⁼ nnt - N ⁼ 000

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPID

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

AXDEPTH)

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

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) * (set)

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

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

"More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity." (W.A. Wulff)



34







nics & (depth < Mode

: = inside } | ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); ⊅)

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

= * diffuse = true;

efl + refr)) && (depth < MAXDEDIN

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 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + radiance.z) > 0) && Closed 2.x + radiance.y + rad

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 S /ive)

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



"Deer Charles,

In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them (...).

One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.

Therefore, one should attend INFOMOV and learn from it.

Love, Ada."



ics & (depth < Modelin

: = inside ? 1 + 1 4 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (00)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPID

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 e.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) (1860)

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

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

Today's Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling



Consistent Approach

(0.) Determine optimization requirements

- 1. Profile: determine hotspots
- 2. Analyze hotspots: determine scalability
- 3. Apply high level optimizations to hotspots
- 4. Profile again.
- . Parallelize
- 5. Use GPGPU
- Profile again.
- 3. Apply low level optimizations to hotspots
- Repeat steps 7 and 8 until time runs out

10. Report.

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

survive = SurvivalProbability(diff)

v = true;

at a = nt

), N);

= true;

AXDEPTH)

f:

refl * E * diffuse;

- at brdfPdf = EvaluateDiffuse(L, N) Provide at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) ();
- andom walk done properly, closely following Soci vive)

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

Do you actually need to speed it up? By how much?

Things to consider:

- You have a finite amount of time for this
- You don't want to break anything
- You don't want to reduce maintainability
- → Focus on 'low hanging fruit' typically a small portion of the code.



38

at a = nt

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

AXDEPTH)

survive = SurvivalProbability(diff. radiance = SampleLight(&rand, I, &L e.x + radiance.y + radiance.z) > 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 follow /ive)

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

Consistent Approach

(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.

Don't trust your intuition

- Not even when optimizing your own code.
- *Especially* not when you are proficient at optimizing.

Blind changes may *reduce* the performance of the code.

Needless to say: *use version* control.



INFOMOV – Lecture 1 – "Introduction"

Never Assume

ics 6 (depth < MaxDem

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

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

(AXDEPTH)

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

w = 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 sold /ive)

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

Profiling

Measuring application performance

- Using external tools
- Using timers in the code

Measurements:

How much time is spent were? (inclusive / exclusive, cycles, percentage)

3437.0

- How often is each function called?
- Low level behavior: stalls / latencies, branch mispredictions, occupation, ...
- Performance over time: lag, spikes, stutter





nics & (depth < ™0000

: = 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 Fr) R = (D = nnt - N - (00)

= * diffuse; = true;

efl + refr)) && (depth < MODEPTI

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly Hf; radiance = SampleLight(&rand, I, &L, &list)

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) * (radi

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:

What if the goal is to have a 10x larger army in your RTS?

 $O(n^2)$ Don't just measure performance, measure *scalability*?(n⁴) O(n)O(√n) Time O(log n) 0(1) Data Input (Space)

nics & (depth < Monos

: = inside ? 1 ()) ht = nt / nc, ddn () s52t = 1.0f - nnt ()), N); ∂)

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

= * diffuse; = true;

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

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

(AXDEPTH)

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

w = 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 Soul /ive)

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

Profiling – getting accurate results

A profiler needs information about your code: this is typically available in *debug* builds.

However:

Debug builds have very different performance characteristics, for many reasons. We need to profile in *release* mode.

Enabling debug information in release mode in Visual Studio:

- Properties >> C/C++ >> General >> Debug information format
- Properties >> Linker >> Debugging >> Generate Debug Info

Differences between debug and release configurations

In debug:

- your code is not optimized
- debug info is added to the executable
- variables are initialized
- memory blocks are padded with guard bytes
- array bounds are checked

In release:

code may be reordered

IMPORTANT:

It makes very little sense to optimize in debug mode.



), N);

if;

/ive)

urvive; pdf;





), N);

= true;

(AXDEPTH)

v = true;

/ive)

urvive; pdf;

sion = true:



INFOMOV – Lecture 1 – "Introduction"



INFOMOV – Lecture 1 – "Introduction"



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

sion = true:

_2013 - Mici	rosoft Visual Studio (Ad	dministrator)						7 D	_ = >
W PROJEC	CT BUILD DEBUG	NSIGHT TOOLS CODE-BUILDER	ANAL	YZE WINDOW HELP Rele	ease – Win32		- II I	. G G ∖	Sign in 📲
- - - - - - - - - - -	Template 150701(3).vsp	→ × Template 150701(2).vsp						g	game.cpp 🖮 🖂
ō - "	← → Current View:	Function Details		- T T = 🦖 👇	💾 📬 🔲				
orer (🔎 -	Tmpl8::Game::Sim	wlate							
4.00a_2013'	Template.exe								
l Dependenc	-					_			
e.txt	Calling functions		\rightarrow	Current function		→	Called functions		
рр		F0 70/		Simulate	67.	4%		Bottom of Stack	
CDD	TICK	53./%							
.h	Toit	13.8%		Function Body	67.4%	•			
e.cpp		13.0 %							
e.n cpp	Related Views: Caller/	Callee Functions					Performance metric:	Inclusive Samples %	
.h	Function Code View								
	D:\ACTIVE\water\game								
		// simulation step 1	- move	, 200					
		drop[i].pos += drop[i].pos -	- prev_pos;					
		<pre>// simulation step 2</pre>	- apply	y gravity					
		drop[i].pos += gravit	y * 0.2	25f;					
		<pre>// simulation step 3</pre>	- satis	sfy constrains					INCHING-
		<pre>for (int step = 0; s </pre>	tep < 3	3; step++)					i tost stra
		۱ // simulation ste		satisfy constraints - e	vade other drops				TWO DAYS AND THE REAL OF THE R
	5.4 %	for (int j = i +	1; j∢	<pre>dropcount; j++)</pre>					1000A.S
		{							
	25.8 %	float dist =	length((drop[i].pos - drop[j].	pos);				MALINE WORKS
	33.7 %	{	KUPKADI	105 2))					
	1.9 %	vec3 dire	ction =	= normalize(drop[i].pos	; - drop[j].pos)				
	0.5 %	drop[i].p	os += c	direction * (DROPRADIUS	* 2 - dist) * 0.	02f;			
		drop[j].p	os -= c	direction * (DROPRADIUS	* 2 - dist) * 0.0	02f;			مصالبته ۲ ال
		// simulation ste	p 3b -	satisfv constraints - e	vade walls				
$i \cap P_1$	rofiler 🛛	<pre>if (drop[i].pos.y</pre>	> 20)	drop[i].pos.y = 19.99f	<pre>- drop[i].pos.z</pre>	* 0.0001f			100-
	lonici	<pre>if (drop[i].pos.x</pre>	< -20)) drop[i].pos.x = -19.99	of + drop[i].pos.	z * 0.000	1f;		
		<pre>if (drop[i].pos.x</pre>	> 20)	<pre>drop[i].pos.x = 19.99f</pre>	<pre>- drop[i].pos.z</pre>	* 0.0001f			
		if (drop[i].pos.z	< -20)) drop[i].pos.z = -19.99	of + drop[i].pos.	z * 0.000	1f;		
	0.1 %	if (drop[i].pos.z	> 20)	drop[i].pos.z = 19.99f	- drop[i].pos.z	* 0.0001f			
	100 % - 4	}	_					_	•
						la 1	Col 1	Ch 1	TNIS
						LIT I	COLT		СРИ



Tools

ics & (depth < ≫occorr

z = inside | | it = nt / nc, ddn os2t = 1.0f - nnt D, N); D)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N - (000

= * diffuse; = true;

-•fl + refr)) && (denth s

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly df; radiance = SampleLight(&rand, I, &. & .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) * (reference);

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

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

very Sleepy CS - C:\Users\Jacco\Ap	ppData\Local\	Temp\F8AF.tm	IP							_
File View Help										
Functions		-	-				Averages Call Stacks	Filters		
Name	Exdu	Inclusive	% Exclusive	% Inclusive	Module	Source File	Called From			
Tmpl8::Game::Simulate	9.47s	9.47s	62.76%	62.76%	water	d:\water\game.	Name	Samples	V Calls	Module
Tmpl8::Game::SmoothWater	2.01s	2.01s	13.34%	13.34%	water	d:\water\game.	Tmpl8::Game::Tick	8.70s	91.88%	water
Tmpl8::Game::DrawTriangle	1.33s	1.33s	8.80%	8.80%	water	d:\water\game.	Tmpl8::Game::Init	0.775	8,12%	water
Tmpl8::Game::RenderZSprites	1.19s	1.19s	7.86%	7.86%	water	d:\water\game.				
Tmpl8::Game::RenderWaterSurface	0.43s	1.76s	2.87%	11.67%	water	d:\water\game.				
Tmpl8::Surface::Clear	0.11s	0.11s	0.75%	0.75%	water	d:\water\surfac				
Tmpl8::Game::RenderDebugInfo	0.03s	0.05s	0.19%	0.32%	water	d:\water\game.				
Tmpl8::Surface::Plot	0.02s	0.02s	0.13%	0.13%	water	d:\water\surfac				
Tmpl8::Game::DownScale	0.02s	0.02s	0.10%	0.10%	water	d:\water\game.				
Tmpl8::Game::TimeSmooth	0.00s	0.00s	0.02%	0.02%	water	d:\water\game.				
Tmpl8::Surface::AddLine	0.00s	0.00s	0.01%	0.01%	water	d:\water\surface				
swap	0.00s	0.25s	0.01%	1.66%	water	d:\water\templa				
[006ADCD0]	0.00s	0.00s	0.00%	0.01%	water					
tmainCRTStartup	0.00s	15.08s	0.00%	99.93%	water	f:\dd\vctools\crt				
SDL_main	0.00s	15.03s	0.00%	99.57%	water	d:\water\templa				
Tmpl8::Game::Tick	0.00s	13.88s	0.00%	91.96%	water	d:\water\game.				
Tmpl8::Game::DrawBoat	0.00s	0.00s	0.00%	0.01%	water	d:\water\game.				
Tmpl8::Game::GlowLine	0.00s	0.00s	0.00%	0.01%	water	d:\water\game.				
for (int	step = 0; s	tep < 3; ste	p++)							
i //	simulation	sten 3a - sa	tisfy constr	aints - eva	le other	drops				
0.70s for	(int j = :	i + 1; j < D	ROPCOUNT; j+	+)		41095	Child Calls			
{	-	-	_			T	Name	Samples	 % Calls 	Module
2.78s	float di	ist = (drop[i].pos - droj TUS * 200	p[j].pos).Le	ength();	<u> </u>				
0.015	{		100 2,7							
0.33s	· · ·	vector3 dire	ction = (drop	p[i].pos - d	lrop[j].p	os).Normalizec				
0.11s	c	irop[i].pos	+= direction	 (DROPRADI 	US * 2 -	dist) * 0.021				
0.02=	, c	irop[j].pos	-= direction	 (DROPRADI 	US * 2 -	dist) * 0.021				
	1									
/зісеру – 7/2	simulation :	step 3b - sa	tisfy constr	aints - eva	de walls					
if	(drop[i].pos	s.y > 20) dr	op[i].pos.y	= 19.99f - 0	drop[i].p	os.z * 0.0001				
if	(drop[i].pos	s.x < -20) d	rop[i].pos.x	= -19.99f ·	+ drop[i]	.pos.z * 0.00				
11	(drop[1].pos	s.x > 20) ar	op[1].pos.x :	= 19.991 - (= -19 99f -	irop[1].p H drop[i]	05.2 * 0.0001				
	(drop[i] po	= 7 > 20) dr	on[i] nos z :	= 19 99f - /	Aron[i] n					
i.e. i.e.						•	1			
i۴										
Source file: d: \water\game.cpp					Line 97					



Tools

ics & (depth < MoxDani

c = inside 7 1 1 1 1 ht = nt / nc, ddh bs2t = 1.0f - nnt 7 7 0, N); 8)

at a = nt - nc, b = 0 at Tr = 1 - (R0 + (1 - R0) Tr) R = (D = nnt - N - (10)

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

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

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 following 300 /ive)

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





AMD

Tools

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

E * ((weight * cosThetaOut) / directPdf)

/ive)

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

MyApp - CodeXL Profile I	Mode (CPU: Time-based Sampling)		10.00					1000	- • ×
File Edit View Debug	Profile Tools Window Help)							
			T. 👩 🛳 🛍 🗖	Work Item X:	▼ Y:	v 7: v	NO. Q. ON R	0000	1 1 5 0
		CPU: Sep 24, 2012 02:	25-28					00000	
		Ci O, SCP 2 1, 2012 02.	23.30						
Sep 24, 2012 02:06:	Profile Overview 🔀 🛛 Call Cha	in 🔀							
Sep 24, 2012 02:11:	Process: 7736 Display S	System Libraries Functions							
Sep 24, 2012 02:13: Sep 24, 2012 02:17:	Functions								
Sep 24, 2012 02:18: Sep 24, 2012 02:20:	Function (153 functions, 15 sh	own) # of Paths	Path Samples	Avg. Samples per Path	Self Samples	Deep Samples	% of Deep Samples	Source File	Module
Sep 24, 2012 02:21:	pow	39	211	5.4	25	211	58% n	nath.h(498)	MyApp.exe =
Sep 24, 2012 02:22:	mainCRTStartup	58	202	3.5		202	56% c	rtexe.c(361)	MyApp.exe
Sep 24, 2012 02:24:	tmainCRTStartup	58	202	3.5		202	56% c	rtexe.c(378)	MyApp.exe
Sep 24, 2012 02:25:	main	57	201	3.5		201	56% n	nyapp.cpp(10)	MyApp.exe
	Worker::doWork	57	201	3.5	4	201	56% v	vorker.cpp(6)	MyApp.exe
	_Pow_int <double></double>	16	152	9.5	142	152	42% n	nath.h(484)	MyApp.exe
	SemiWorker::doAsyncWor	k 51	137	2.7		137	38% s	emiworker.cpp(43)	MyApp.exe
	SemiWorker::Calc	35	118	3.4	6	118	33% s	emiworker.cpp(28)	MyApp.exe 🔻
	Immediate Ancestors and Children Parents F	n of function: "pow" unc Samples Deep	Samples % of Deep 9	Samples	Self + Children	Func Sam	ples Deep Sample	es % of Deep Samp	oles
	Worker::doWork	4	201	46%	_Pow_int <dout< td=""><td>ole></td><td>142 15</td><td>52</td><td>44%</td></dout<>	ole>	142 15	52	44%
	SemiWorker::Calc	б	118	27%	(self)		25 (25 sel	lf)	7%
	Paths containing function: pow					~			
	Function				Self Samples	Downstream Samples			Downstream 🔺 Samples %
	SemiWorker::doAsyncWorker	k				83			39% ≡
	▲ SemiWorker::Calc					83			39%
	⊳ pow				12	71			34%
	mainCRTStartup					128			61%
) CodeXL	4tmainCRTStartup					128			61%
Gedenia									
							_		_



INFOMOV – Lecture 1 – "Introduction"

Never Assume

nics & (depth < Note:

= inside / l ht = nt / nc, ddn os2t = 1.0f - nnt O, N); 3)

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPIL

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

AXDEPTH)

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:

Take-away:

Never assume. Profiling *always* steers optimization.

Optimize in release mode. Enable debug info during this process. Don't forget to turn it off before distribution.





Profiler Output

nics & (depth < Monnes

t = inside } | | | | | ht = nt / nc, ddn | | bs2t = 1.0f - nnt | n D, N); ð)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Ir) R = (D = nnt - N = (00)

= * diffuse = true;

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

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

AXDEPTH)

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

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 Source /ive)

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





INFOMOV – Lecture 1 – "Introduction"

Profiler Output

), N);

AXDEPTH)

v = true;

/ive)

urvive;

sion = true:

refl * E * diffuse;

survive = SurvivalProbabil

radiance = SampleLight(&r e.x + radiance.y + radianc

at brdfPdf = EvaluateDiffu: at3 factor = diffuse * INVi at weight = Mis2(directPd' at cosThetaOut = dot(N, L E * ((weight * cosThetaOu'

at3 brdf = SampleDiffuse(

pdf; n = E * brdf * (dot(N, R

File view Help											
Functions									Averages	Call Stacks	Filters
Name	Exclu	 Inclusive 	% Exclusive	% Inclusive	Module	Source File	Sourc	Address 🔺	Called From		
Tmpl8::Game::Simulate	3.06s	3.06s	67.89%	67.89%	Template	f:\projects\water\game.cpp	97	0x401763	Name		Samples
Tmpl8::Game::SmoothWater	0.47s	0.47s	10.54%	10.54%	Template	f:\projects\water\game.cpp	206	0x401c20	Tmpl8::G	ame::Tick	2,175
Tmpl8::Game::RenderZSprites	0.32s	0.32s	7.18%	7.18%	Template	f:\projects\water\game.cpp	170	0x401a59	Tmpl8::G	ame::Init	0.89s
Tmpl8::Game::DrawTriangle	0.32s	0.32s	7.14%	7.14%	Template	f:\projects\water\game.cpp	119	0x402a6			0.055
Tmpl8::Game::RenderWaterSurface	0.11s	0.44s	2.52%	9.66%	Template	f:\projects\water\game.cpp	278	0x40262c			
Tmpl8::Surface::Plot	0.01s	0.01s	0.31%	0.31%	Template	f:\projects\water\surface.cpp	177	0x40391(
Tmpl8::Surface::Clear	0.01s	0.01s	0.29%	0.29%	Template	f:\projects\water\surface.cpp	76	0x4036f7			
[0062F8B6]	0.01s	0.01s	0.13%	0.13%	Template		0	0x62f8b6			
[0062F8C0]	0.01s	0.01s	0.13%	0.13%	Template		0	0x62f8c0			
Tmpl8::Game::RenderDebugInfo	0.01s	0.02s	0.13%	0.44%	Template	f:\projects\water\game.cpp	308	0x40299:			
zbuffer	0.00s	0.01s	0.00%	0.13%	Template	[unknown]	0	0x4090f8			
tmainCRTStartup	0.00s	4.51s	0.00%	100.00%	Template	f:\dd\vctools\crt\crtw32\dllstuf	618	0x405e0f			
SDL_main	0.00s	4.49s	0.00%	99.58%	Template	f:\projects\water\template.cpp	252	0x404cc5			
Tmpl8::Game::Tick	0.00s	3.44s	0.00%	76.32%	Template	f:\projects\water\game.cpp	320	0x402c6c			
Tmpl8::Game::Init	0.00s	0.89s	0.00%	19.81%	Template	f:\projects\water\game.cpp	49	0x40146			
WinMain	0.00s	4.51s	0.00%	100.00%	Template	x: \projects \sdl \src \main \windo	177	0x4010c7			
main	0.00s	4.51s	0.00%	100.00%	Template	x: \projects \sdl \src \main \windo	140	0x40101(
Source Log // simula drop[i].p // simula drop[i].y // simula for (int	tion step os += drop tion step os += grav tion step step = 0;	1 - move [i].pos - pre 2 - apply gra ity * 0.25f; 3 - satisfy c step < 3; st	v_pos; vity onstrains ep++)						Child Calls		
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: 0.86s 1.68s 0.08s	tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d {	<pre>1 - move [i].pos - pre 2 - apply gra stity * 0.25f; 3 - satisfy c step < 3; st n step 3a - s; = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[i].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - eva +) s - drop[j]. ze(drop[i]. * (DROPRAD] * (DROPRAD)	de other dr .pos); pos - drop US * 2 - d US * 2 - d	<pre>cops ([j].pos); ist) * 0.02f; ist) * 0.02f;</pre>			Child Calls Name		Sampl
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: 0.86s 1.68s 0.08s	<pre>tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d { } }</pre>	<pre>1 - move [i].pos - pre 2 - apply gra ity * 0.25f; 3 - satisfy c step < 3; st n step 3a - s; = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[i].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - eva +) s - drop[j]. * (DROPRAD] * (DROPRAD]	de other dr .pos); pos - drop US * 2 - d US * 2 - d	<pre>cops (j].pos); ist) * 0.02f; ist) * 0.02f;</pre>			Child Calls Name		Sampl
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.86s 1.68s 0.08s	<pre>tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d { }</pre>	<pre>1 - move [i].pos - pre 2 - apply gra ity * 0.25f; 3 - satisfy c step < 3; st n step 3a - s = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[j].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - eva +) s - drop[j]. ze(drop[i]. * (DROPRAD] * (DROPRAD]	de other dr .pos); pos - drop US * 2 - d US * 2 - d	<pre>cops (j].pos); ist) * 0.02f; ist) * 0.02f;</pre>		×	Child Calls Name		Sampl
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: 0.86s 1.68s 0.08s 4 source file: f:\projects\water\game.cpp	<pre>tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d { }</pre>	<pre>1 - move [i].pos - pre 2 - apply gra sity * 0.25f; 3 - satisfy c step < 3; st n step 3a - s = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[i].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - evad +) s - drop[j]. ze(drop[i]. * (DROPRAD) * (DROPRAD)	de other dr .pos); pos - drop US * 2 - d US * 2 - d	<pre>cops ([j].pos); (ist) * 0.02f; ist) * 0.02f; [Line 25</pre>		× •	Child Calls Name		Sampl
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: 0.86s 1.68s 0.08s 4 source file: f:\projects\water\game.cpp	<pre>tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d { }</pre>	<pre>1 - move [[].pos - pre 2 - apply gra sity * 0.25f; 3 - satisfy c step < 3; st n step 3a - s = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[i].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - evad +) s - drop[j]: ze(drop[i] * (DROPRAD) * (DROPRAD)	de other dr .pos); pos - drop .US * 2 - d .US * 2 - d	cops ([j].pos); (ist) * 0.02f; (ist) * 0.02f; Line 25		×	Child Calls Name I		Samp
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: 0.86s 1.68s 0.08s 4 Source file: f:\projects\water\game.cpp	<pre>tion step os += drop tion step os += grav tion step = 0; simulatio r (int j float if (d { } }</pre>	<pre>1 - move [[].pos - pre 2 - apply gra sity * 0.25f; 3 - satisfy c step < 3; st n step 3a - s = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[j].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - eva +) s - drop[j]. * (DROPRAD) * (DROPRAD)	de other dr .pos); pos - drop US * 2 - d US * 2 - d	<pre>cops (j].pos); ist) * 0.02f; ist) * 0.02f;</pre>		•	Child Calls Name I		Samp
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: 0.86s 1.68s 0.08s 4 Source file: f:\projects\water\game.cpp	<pre>tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d { } }</pre>	<pre>1 - move [i].pos - pre 2 - apply gra ity * 0.25f; 3 - satisfy c step < 3; st a step 3a - s; = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[i].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - evad +) s - drop[j]. ze(drop[i]. * (DROPRAD]	de other dr .pos); US * 2 - d US * 2 - d	rops (j].pos); (ist) * 0.02f; (ist) * 0.02f; (Line 25		× •	Child Calls Name		Sampl
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: (0.86s 1.68s 0.08s 4 Source file: f:\projects\water\game.cpp	<pre>tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d { }</pre>	<pre>1 - move [i].pos - pre 2 - apply gra sity * 0.25f; 3 - satisfy c step < 3; st a step 3; st = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[i].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - evad +) s - drop[j]. ze(drop[i]. * (DROPRAD) * (DROPRAD)	de other dr .pos); pos - drop US * 2 - d US * 2 - d	:ops (j].pos); (ist) * 0.02f; (ist) * 0.02f; Line 25		× •	Child Calls Name		Sampl
Source Log // simula drop[i].p // simula drop[i].p // simula for (int { 0.44s fo: 0.86s 1.68s 0.08s 4 K. apd1	<pre>tion step os += drop tion step os += grav tion step step = 0; simulatio r (int j float if (d { } }</pre>	<pre>1 - move [i].pos - pre 2 - apply gra ity * 0.25f; 3 - satisfy c step < 3; st n step 3a - s; = i + 1; j < 1 dist = lengtl ist < (DROPRAI vec3 directi drop[i].pos</pre>	<pre>v_pos; vity onstrains ep++) atisfy constr DROPCOUNT; j+ h(drop[i].po DIUS * 2)) ion = normali += direction -= direction</pre>	aints - evad +) s - drop[j]. ze(drop[i]. * (DROPRAD]	de other dr .pos); pos - drop US * 2 - d US * 2 - d	cops (j].pos); (ist) * 0.02f; (ist) * 0.02f; (Line 25		× •	Child Calls Name		Sampl



Profiler Output

ics (depth < Marine

= inside } 1 1 1 1 ht = nt / nc, ddn bs2t = 1.0f - nnt ∩ n D, N); ≫)

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPIN

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

AXDEPTH)

```
survive = SurvivalProbability( diffuse )
estimation - doing it properly closed
f;
radiance = SampleLight( &rand, I, &L, &Light
e.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)

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

Profiling – Results

Game::Simulate Game::SmoothWater Game::RenderZSprites Game::Tick

Running \sim 3 seconds, we spent 0.86s on this line:

float dist = length(drop[i].pos - drop[j].pos);

67.89%

10.54%

7.18%

0.00%

67.89%

10.54%

7.18%

76.32%

and 1.68s on this line:

if (dist < (DROPRADIUS * 2))</pre>

🚷 Very Sleepy CS - C:\Users\Jacco\App	Data\Local\Te	mp\B916.tmp		
File View Help				
Functions				
Name	Exdu 👻	Inclusive	% Exclusive	% Indusive
Tmpl8::Game::Simulate	3.06s	3.06s	67.89%	67.89%
Tmpl8::Game::SmoothWater	0.47s	0.47s	10.54%	10.54%
Tmpl8::Game::RenderZSprites	0.32s	0.32s	7.18%	7.18%
Tmpl8::Game::DrawTriangle	0.32s	0.32s	7.14%	7.14%
Tmpl8::Game::RenderWaterSurface	0.11s	0.44s	2.52%	9.66%
Tmpl8::Surface::Plot	0.01s	0.01s	0.31%	0.31%
Tmpl8::Surface::Clear	0.01s	0.01s	0.29%	0.29%
[0062F8B6]	0.01s	0.01s	0.13%	0.13%
[0062F8C0]	0.01s	0.01s	0.13%	0.13%
Tmpl8::Game::RenderDebugInfo	0.01s	0.02s	0.13%	0.44%
zbuffer	0.00s	0.01s	0.00%	0.13%
tmainCRTStartup	0.00s	4.51s	0.00%	100.00%
SDL_main	0.00s	4.49s	0.00%	99.58%
Tmpl8::Game::Tick	0.00s	3.44s	0.00%	76.32%
Tmpl8::Game::Init	0.00s	0.89s	0.00%	19.81%
WinMain	0.00s	4.51s	0.00%	100.00%
main	0.00s	4.51s	0.00%	100.00%
<u> </u>				
Source Log				
Source Log				
// simulatio	on step 1 - :	move		
// simulatio	<pre>+= drop[1]. on step 2 -</pre>	apply gravit	pos, tv	
drop[i].pos	+= gravity	* 0.25f;	-1	
// simulatio	on step 3 -	satisfy con	strains	
for (int st	tep = 0; ste	p < 3; step	++)	
{	imulation st	on 3a - eat	iefu constra	inte - en
0.44s for	(int i = i	ep sa - sat. + 1; i < DR(DPCOUNT; i++	-)
{				
0.86s	float dis	t = length(drop[i].pos	- drop[j
1.68s	if (dist	< (DROPRADIU	JS * 2))	
0.08=	1	-3 direction	= normaliz	e (drop[i
	dro	op[i].pos +=	direction	 (DROPRA
	dro	op[j].pos -=	direction	 (DROPRA
	}			
Source file: f:\projects\water\game.cpp				



INFOMOV – Lecture 1 – "Introduction"

Profiler Output

Profiling – finding hotspots

The profiler allows you to quickly find the parts of your program that take most time.

But:

- Mind debug versus release;
- The profiler doesn't tell you why a function is costly
- The profiler doesn't report scalability
- There is no 'cost over time' information

AXDEPTH)

), N);

= true;

= true:

efl + refr)) && (dec

refl * E * diffuse;

- survive = SurvivalProbability(diffus estimation - doing it properly close If; radiance = SampleLight(&rand, I, &L, e.x + radiance.y + radiance.z) > 0) &
- 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, closely follo /ive)

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

- → Scalability analysis requires running the program with different work sets (i.e., change N in O(N)).
- Determining why a section takes a lot of time requires more in-depth knowledge.
 - Solving the performance issue requires even more in-depth knowledge.





Profiler Output

	Solution Explorer $\bullet \Psi \times$	r003ge 🗘 🗙 game.cpp
	○○☆ ७・≈ ๗ ₪ "	Seneral Exploration General Exploration viewpoint (<u>change</u>) ③
	Search Solution Explorer (Ctrl+;)	🕢 ⊕ Analysis Target 🙏 Analysis Type 📟 Collection Log 🗓 Summary 🥵 Bottom-up 🗳 PMU Events 🔛 Tasks and Frames 🚯 game.cpp 🚯 game.cpp
	Solution 'tmpl84.00a_2013' (1 proj	💫 Elapsed Time: 🔍 11.571s 🗎
	 Template 	Paused Time: O 3.150s
	Am r000hs	
	Am r001ge	
4	Am r002ths	<u>Instructions Repred:</u> 7,236,010,854
depth a MANDER	Am r003ge	<u>CPI Rate:</u> 0 0.513
- Area	🔺 💁 Template	⊗ Filled Pipeline Slots: [®]
= inside) 1 1 1 1	External Dependencies	
nt = nt / nc, ddn	_readme.txt	%RetiredPipelineSlotsIssueTextAll
os2t = 1.0f - nat	++ game.cpp	General Retirement: 0 0.767
), N);	B game.h	Inside the program. A uDospect-instruction ratio of 1 is expected. If the Retirement value for non-vectorized code is high, consider vectorizing own code to reduce instructions and hence
ð)	++ surface.cpp	this bucket.
	Image: Surface.h	Microcode Sequencer: © 0.007
at a = nt - nc, b = nt - n	++ template.cpp	⊗ Bad Speculation: 0.107
at Tr = 1 - (R0 + (1 - R0	Image: Image: Image: book i	A significant proportion of pipeline slots containing useful work are being cancelled. This can be caused by mispredicting branches or by machine clears.
(ddi) R = (D = nnt - N = (ddi)	++ threads.cpp	Branch Mispredict: [©] 0.000
	Image:	Machine Clears: 0 0.107
* dittuse;		A significant portion of execution time is spent handling machine clears. Examine the MACHINE CLEARS events to determine the specific cause.
= true;		🔗 Unfilled Pipeline Slots (Stalls): 🔍
		⊗ Back-End Bound: 0 0.070
fl i pofo)) 22 (dooth a Niver		Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where
TIT TETT)) aa (depen s maabaan		or stalls due to the overloaded divider unit are examples of back-end bound issues.
). N):		Memory Bound: [™] 0.000
refl * F * diffuse:		⊙ Core Bound: 0.493
= true:		C Front-End Bound: 0 0.049
		CPU Usage Histogram
4AXDEPTH)		This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.
survive = SurvivalProbability(diffuse)		15
estimation - doing it properly, closed		e /.ss
1†;		등 5s 호
radiance = SampleLight(&rand, 1, 81, 11)		
.x + radiance.y + radiance.z) > 0) as (die		
- true.		36 2
at brdfPdf = EvaluateDiffuse(L_N) = Psie		
at3 factor = diffuse * INVPT:		1.55
at weight = Mis2(directPdf, brdfPdf):		
at cosThetaOut = dot(N, L);		
<pre>E * ((weight * cosThetaOut) / directPdf) *</pre>		
		Simultaneously Utilized Logical CPUs
andom walk - done properly, closely following		
/ive)		
at3 brdf = SampleDittuse(diffuse, N, r1, r2.		
irvive;		
sion = true:		



INFOMOV – Lecture 1 – "Introduction"

Profiler Output

);))

nt a = nt - nc, b = nt nt Tr = 1 - (R0 + (1 - R0 ir) R = (D = nnt - N = (000

* diffuse; = true;

fl + refr)) && (death c P

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &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)

andom walk - done properly, closely follo /ive)

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

So. Lin.▲	Source	A Clockticks	Instructions Retired	CPI Rate	Filled Pipeline Slots		Unfilled Pipeline Slo	
					≫	≫	>	X
					Retiring	Bad Spec	Back-End Bound	Fr. Bo.
129								
130	// Game::Simulate: do Verlet physics simulation on the particles							
131	void Game::Simulate()							
132	{							
133	ENTER(SIMULATE);							
134	<pre>for (int i = 0; i < DROPCOUNT; i++)</pre>							
135	{							
136	gravity = vec3(sin(angle * PI / 180), cos(angle * PI / 180), 0);							
137	<pre>vec3 prev_pos = drop[i].prev_pos;</pre>							
138	<pre>drop[i].prev_pos = drop[i].pos;</pre>							
139	// simulation step 1 - move							
140	<pre>drop[i].pos += drop[i].pos - prev_pos;</pre>							
141	<pre>// simulation step 2 - apply gravity</pre>							
142	drop[i].pos += gravity * 0.25f;							
143	<pre>// simulation step 3 - satisfy constrains</pre>							
144	<pre>for (int step = 0; step < 3; step++)</pre>	0	2,000,003	0.000	0.000	0.000	1.000	0.00
145	{							
146	// simulation step 3a - satisfy constraints - evade other drops							
147	<pre>for (int j = i + 1; j < DROPCOUNT; j++)</pre>	128,000,192	148,000,222	0.865	0.703	0.000	0.824	0.05
148	{							_
149	<pre>float dist = length(drop[i].pos - drop[j].pos);</pre>	780,001,170	208,000,312	3.750	1.000	0.000	0.231	0.01
150	if (dist < (DROPRADIUS * 2))	1,042,001,563	2,818,004,227	0.370	0.115	0.446	0.381	0.05
151								
152	<pre>vec3 direction = normalize(drop[i].pos - drop[j].pos);</pre>	64,000,096	86,000,129	0.744	0.820	0.000	0.180	0.00
153	<pre>drop[i].pos += direction * (DROPRADIUS * 2 - dist) * 0.02f;</pre>	22,000,033	0		0.000	1.000	0.000	1.00
154	<pre>drop[j].pos -= direction * (DROPRADIUS * 2 - dist) * 0.02f;</pre>	6,000,009	2,000,003	3.000	0.000	0.000	1.000	0.00
155	}							
156	}							
157	// simulation step 3b - satisfy constraints - evade walls							
156	ir (arop[i], pos.y > 20) arop[i], pos.y = 19.991 - arop[i], pos.z = 0.000							
159	ir (drop[i].pos.x < -20) drop[i].pos.x = -19.99r + drop[i].pos.z * 0.0							
160	ir (arop[i], pos.x > 20) arop[i], pos.x = 19.991 - arop[i], pos.z = 0.000							
161	ir (arop[i], pos.z < -20) arop[i], pos.z = -19.991 + arop[i], pos.z = 0.00							
162	11 (arop[1].pos.z > 20) arop[1].pos.z = 19.991 - arop[1].pos.z = 0.000							
164								
165								
165	LEAVE (SIRULAIE);							
100				-				<u></u>

56

INFOMOV – Lecture 1 – "Introduction"

Profiler Output

Take-away:

Free, vendor-agnostic profilers tell you where time is spent in your program (but not *why*).

Vendor-specific tools provide a wealth of information, but generally require knowledge about the hardware processes.

Stalls are generally not vendorspecific and will be similar on similar hardware.

Just timing information is often sufficient to make an educated guess towards improvements.

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

= true:

), N);

(AXDEPTH)

v = true:

refl * E * diffuse;

survive = SurvivalProbability(dif

radiance = SampleLight(&rand, I

e.x + radiance.y + radiance.z) > (

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

E * ((weight * cosThetaOut) / directPdf)

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









Generic Profiler Downsides

- No 'performance over time' measurements
- Requires inclusion of debug information (including source code)
- Not real-time
- Not very intuitive

Using a custom in-app profiler we can drastically improve our profiling information.

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) = Psurvivo 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)

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



nics & (depth < Mox00

c = inside / l ht = nt / nc, ddh os2t = 1.0f - nnt 0, N); 3)

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

= * diffuse; = true;

-:**fl + refr)) && (death < MAXD**

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

AXDEPTH)

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

w = true; at brdfPdf = EvaluateDiffuse(L, N) * at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf) at cosThetaMt= A @ C A a f f E * ((weight * cosThetaOut) / directPd

andom walk - done properly, closely folloring /ive)

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

Hineonaft Beta 1,9 Prenetease 2 (118 fps, 2 ohunk updates) C: 979/5498, F: 1912, O: 9, E: 2517 E: 1/251, B: 9, I: 259 P: 9, T: All: 251 ServerChunkCache: 979 Drop: 9

x: <u>12,393761495692818</u> y: 7962000000476837 z: 231,78292536730885 f: 3

Seed: 5130996236305320162

666666666



Used memory: 41% (412HB) of 989HB fillocated memory: 199% (989HB)

level 82.58% textures 11.06%

777 0.23% gameRenderer 0.07% keyboard 0.03% gameMode 0.02% stats 0.01% particles 0.01%

centerChunkSource 0.0% levelRenderer 0.0%

nics & (depth < NAXOSA

: = inside 7 1 1 1 1 ht = nt / nc, ddh bs2t = 1.0f - nnt 7 0, N); 8)

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

= * diffuse = true;

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

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

AXDEPTH)

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

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvivation at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf) at cosThetaOut = dot(NUNPEALENGINE 3 E * ((weight * cosThetaOut) / directPdf)

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

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





at3 brdf = SampleDiffuse(diffuse, N, r

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

urvive;

	MainThread						
	RenderThread					1	
				NY WAY			-
	Worker 0			2167	ST.		
	Worker 1			The second			
	Worker 2				1		
	Worker 3						
	hite allowed A				40	ि सेंग्रे स-सः विस्	<u>्</u>
	worker 4				3-48-9		
	Worker 5	12.			NUS-	1. A.	5
	Worker 6			A Cast			7.51
- aittuse; true;	Worker 7	1	-	als at	10	1	
	MainThread 13	5.02 ms	JobName (Num Invocati	ons) TimeE	xecuted(MS) Ti	Lme
1 + refr)) && (depth < MODEPTH	BLOOM_GEN 0 BOKEH DOF C	0.04 ms OMPOSE 00.02 m	AsyncDIP AsyncOctreeUpdate	(0): (0);	0.00	0.00	0.
N); fl * E * diffuse:	COMPOSITION CPartManage	00.04 ms	CRenderProxy Render CWaterVolume Render	(35):	0.10 0.15	0.00	0.
• true;	CSystem::Up	date() 00.16 m	CheckOcclusion CommandBufferExecute	(1);	0.09	0,00	G. 0,
	CombineColo	rGradingWithCo	ComputeVertices CreateSubsetRenderMes CrvAsyncMemony	(/4): (0): (0):	0.58	0.00	0.
	DEFERRED_CO	CALS 00.89 ms	FinalizeRendItems FinalizeShadowRendIte	(2); ms (2);	0.10 0.04	0.21	0.
<pre>irvive = SurvivalProbability(diffu stimation - doing it properly, clo</pre>	DEFERRED_LI DEPTH READB	GHTS 00.44 ms ACK 00.20 ms	InvokeShadowMapRender MNRM_SortActiveInstar MivStreamsKernel	Jobs(11): ces (1):	0.31	0,00	0.
; adiance = SampleLight(&rand, I, &L	DRAWSTRINGW	01.20 ms	NavigationGeneration PVRNCullSamples	(); (26);	9.66 9.06	0.00	G. 0.
x + radiance.y + radiance.z) > 0)	EYE OVERLAY	00,00 ms	PVRNUpdateDeform PVRNUpdateSpines	(); (21);	0.79	0,00	0.
= true; : brdfPdf = EvaluateDiffuse(L. N.)	EntitySyste	m::Update 00.3	PrepareOcclusion Rast PrepareOcclusion Rast	teriz(1):	0.00 0.00 0.00	0.00	0.
3 factor = diffuse * INVPI;	FRUSTUM 0 0	ALS 00.04 Ms 0.10 ms	PrepareOcclusion Rep PrepareOcclusion Rep	ojec(16): ojec(16):	0.49 0.27	0.00	0.
cosThetaOut = dot(N, L);	FRUSTUM 1 0 FRUSTUM 2 0	0,22 ms	RenderContent SkinningTransformatic	(117); onsCo(2);	0.85 0.00 0.00	0.00 0.00	6) 0.
: * ((weight * cosThetaOut) / direc	FRUSTUM 3 0 GENERAL 00	1.10 ms	SortRendItems StreamInflateBlock	(9); (9);	0.16 0.00	0.00	0.
idom walk - done properly, closely	CENEDAL DW			MM .	28 V-6 14	and the second second	1.4

And in case of the local division of the loc

hit(MS) AVG(MS) TexStrmUpdate (.0): 0.00 UpdateParticles (34): 0.40 WaterUbdate (1): 0.29

WaterUpdate (1): 0,29 0,00 0,29 2110 inflate (0): 0.00 0,09 0.00

CryEngine

nics & (depth < NOC

t = inside 7 1 1 1 0 nt = nt / nc, ddn ns2t = 1.0f - nnt ⊂ n 2, N); 8)

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

= * diffuse = true;

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

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N) Pourviv 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, 8R, 8p; urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



StarCraft II



nics & (depth < MaxCos

c = inside / l 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 = (300

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPID

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

AXDEPTH)

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) * (reference);

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

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



StarCraft II



Take-away:

In-app profiling provides advantages over external profilers:

- You get real-time information, which is easily associated with what is going on in the app;
- You can measure statistics that are not available to the profiler;
- You can present the data in a form that is also useful to people not familiar with the intricacies of the profiler.





x.x + radiance.y + radiance.z) > 0) && (...) w = true; at brdfPdf = EvaluateDiffuse(L, N) * Pour at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L);

survive = SurvivalProbability(diff

radiance = SampleLight(&rand, I,

= true:

), N);

= true;

(AXDEPTH)

refl * E * diffuse;

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

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:

Considerations

Custom timers: what to measure?

- Time spent in your code
- 'Wall clock time'
- Cycles

In what quantities?

- A millisecond is a *long time*
- Averaged / smoothed values are easier to read
- Relative performance may be better

The impact of measurements:

- .x + radiance.y + radiance.z) > 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)

survive = SurvivalProbability(diff)

radiance = SampleLight(&rand, I, &

at a = nt

), N);

= true;

(AXDEPTH)

f:

refl * E * diffuse;

andom walk - done properly, closely follo /ive)

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

- Especially relevant for brief snippets of code
- Logging is expensive!

This is what you can control Including file I/O, library calls, ... CPU-independent (but: rate may change)



5

Considerations

at a = nt

efl + refr)) && (depth

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

(AXDEPTH)

survive = SurvivalProbability(diff) if; 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) andom walk - done properly, closely follow

/ive)

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

Consistent Approach

 $(\mathbf{0})$ **Determine optimization requirements**

- Profile: determine hotspots 1.
- Analyze hotspots: determine scalability Ζ.
- Apply high level optimizations to hotspots 2
- 4. Profile again.
 - Parallelize / vectorize / use GPGPU
- Profile again. 6.
- Apply low level optimizations to hotspots 7.
- Repeat steps 6 and 7 until time runs out 8.
- 9. Report.



And Finally:

Profiling:

Without it, no optimization – we need to *know*

How to profile: tools, custom timers, CPU + GPU

What to profile: realistically (release!), raw performance, scalability *(but also: cache misses, pipelining, branch prediction)*

Keep in mind: profiling takes time too.

Repeated profiling: things change, if you're doing it right. Stay informed.

if; adiance = SampleLight(&rand, I, &L, &li) e.x + radiance.y + radiance.z) > 0) && (d) w = true; at brdfPdf = EvaluateDiffuse(L, N) = Psu at2 factor = diffuse * TAN/PT.

survive = SurvivalProbability(diff.

at a = nt

), N);

AXDEPTH)

efl + refr)) && (depth

refl * E * diffuse; = true;

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

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: 69

nics & (depth < MAXDE

: = inside ? | ht = nt / nc, ddn bs2t = 1.0f - nnt m D, N); 3)

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

= * diffuse = true;

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

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

AXDEPTH)

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

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 "Introduction"

next lecture: "Low Level"

