

```
...
    & (depth < MAXDEPTH)
...
    c = inside / 4.0;
    nt = nt / nc;
    cos2t = 1.0f - nt * nt;
    D, N );
)
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * (dot
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, M, &light
e.x + radiance.y + radiance.z) > 0) && (nt > 0)
...
v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPT;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
...
random walk - done properly, closely following
survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
```

/INFOMOV/

Optimization & Vectorization

J. Bikker - Sep-Nov 2018 - Lecture 1: "Introduction"

Welcome!



```
...
    && (depth < MAXDEPTH)
{
    // Inside of a sphere
    Vec r = inside / R;
    Vec nt = nt / nc, ndd = ndd * ndd;
    double r2t = 1.0f - nt * ndd;
    Vec t = Vec(0, 0, 0);
    Vec N = N;
    double a = nt - nc, b = nt + nc;
    double r1 = 1 - (RR + 1) * RR;
    double r2 = (D * nnt - N * ndd);
    Vec E * diffuse;
    bool = true;
    ...
    refl + refr)) && (depth < MAXDEPTH)
{
    Vec N;
    refl * E * diffuse;
    bool = true;
    ...
    MAXDEPTH)
{
    survive = SurvivalProbability( diffuse, N );
    // estimation - doing it properly, close to Monte Carlo
    if(
    radiance = SampleLight( &rand, E, N, &light );
    Vec r = Vec( r.x + radiance.x + radiance.z ) > 0) && (depth <
    bool = true;
    Vec brdfPdf = EvaluateDiffuse( L, N ) * survive;
    double factor = diffuse * INVPT;
    Vec weight = Mts2( directPdf, brdfPdf );
    double cosThetaOut = dot( N, L );
    Vec E * ((weight * cosThetaOut) / directPdf) * radiance;
    // random walk - done properly, closely following Monte Carlo
    survive)
{
    Vec brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    Vec pdf;
    Vec r = E * brdf * (dot( N, R ) / pdf);
    bool = true;
    ...
}
```

Today's Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling



Introduction

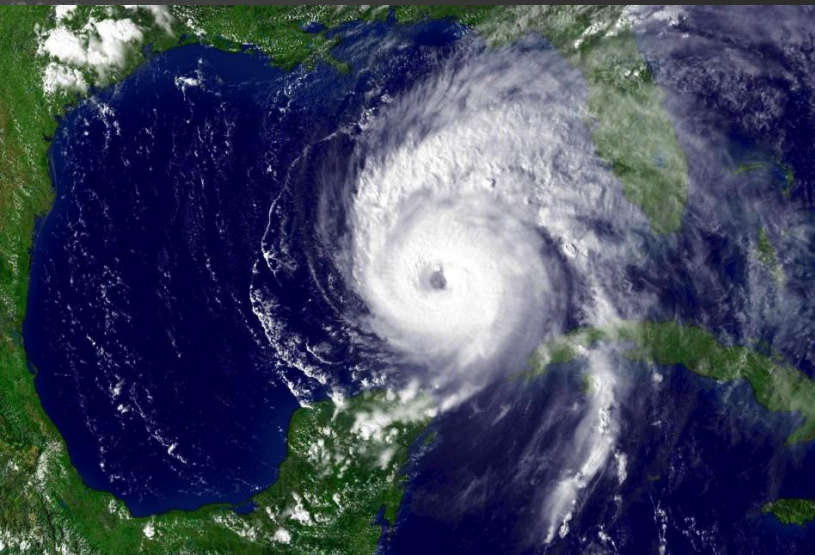
Why?

Some problems require the supercomputer of the future.

```

...
(depth + 1)
...
inside / ...
nt / ...
os2t = 1.0f / ...
...
N );
...
)
...
at a = nt - nc, b = nt
at Tr = 1 - (R0 + 1) * ...
Tr) R =
...
dif
= true
...
efl + n
...
D, N );
refl *
= true
...
MAXDEPTH
...
survive
estima
if;
radiance
e.x + n
...
w = true
at brdf
at3 fac
at weig
at cosine
E * ((weight * cosThetaOut) / direction
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, AR, app
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



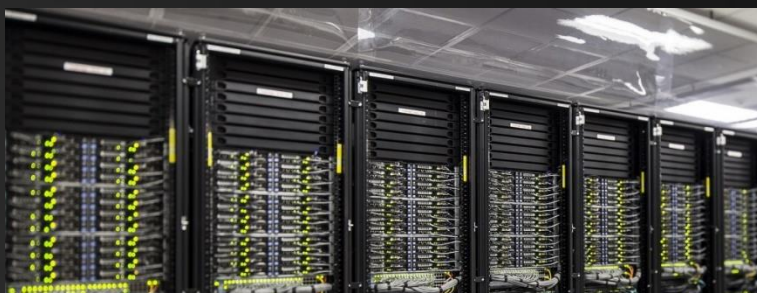
Introduction

Why?

Some problems require the supercomputer of the future.

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

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

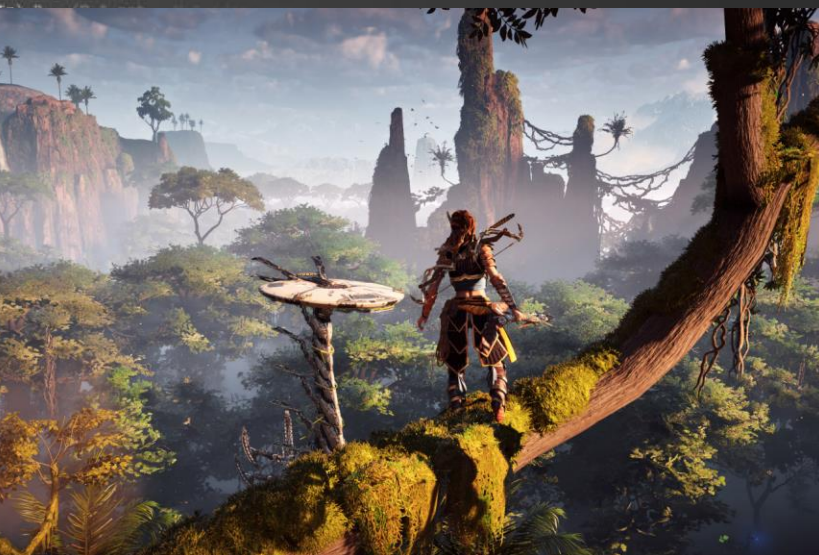


Introduction

Why?

Games want to raise the bar.

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



```
E * ((weight * cosThetaOut) / directPDF);  
  
random walk - done properly, closely follow  
(ive)
```

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, RR, spp  
urvive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
ision = true;
```

FUTUREMARK
CORPORATION

WORK IN PROGRESS
WWW.3DMARK.COM
3DMARK
VANTAGE



Introduction

Why?

Some software needs to run on pretty weak hardware.

- Limited CPU, limited RAM (limited controls).



Introduction

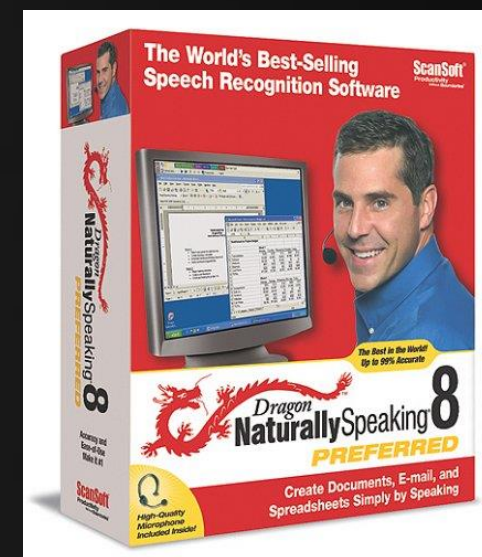
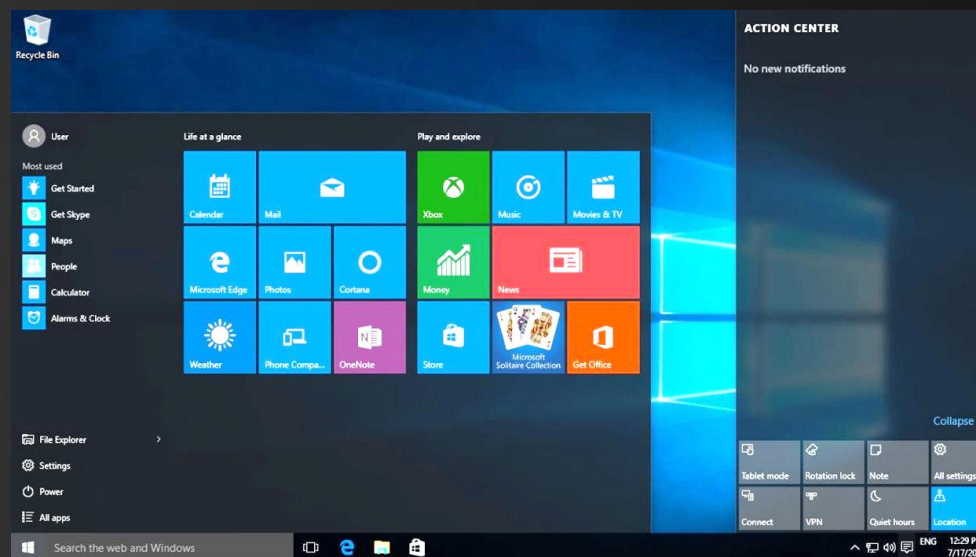
Why?

Some software should not use 90% of your CPU.

- Leave room for other applications, be invisible.

```

ice
(depth + MAXDEPTH)
...
+ inside / ...
nt = nt / ...
os2t = 1.0f * ...
D, N );
...
at a = nt - nc, b = ...
at Tr = 1 - (R0 + I) * ...
Tr) R = (D * nnt - N ...
...
E * diffuse;
= true;
...
efl + refr) && (depth <
...
D, N );
refl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbabl
estimation - doing it pr
ff;
...
radiance = SampleLight(
.e.x + radiance.y + radian
...
v = true;
at brdfPdf = EvaluateDiff
at3 factor = diffuse * IV
at weight = Mis2( directP
at cosThetaOut = dot( N,
E * ((weight * cosThetaOut) / directPdf)
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, spon
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;
    
```



Introduction

Why?

Sometimes the cheapest / lowest power CPU is the best.

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



Introduction

Why?

Some things are done so frequently, they must be efficient.

- Memory manager
- Garbage collector
- JIT compiler
- Compilers in general
- Image processing in Photoshop
- OS startup / resume
- ...
- ...



```

...
    & (depth < MAXDEPTH)
...
    c = inside / 2.0;
    nt = nt / nc;
    cos2t = 1.0f - nt * nt;
    D, N );
    )
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 + R0);
    (Tr) R = (D * nnt - N * (dot
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
    MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, E, M, Allg
    e.x + radiance.y + radiance.z) > 0) && (nt
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    t3 factor = diffuse * INVPI;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance
...
    random walk - done properly, closely following
    (survive)
...
    t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```



Introduction

What is optimization?

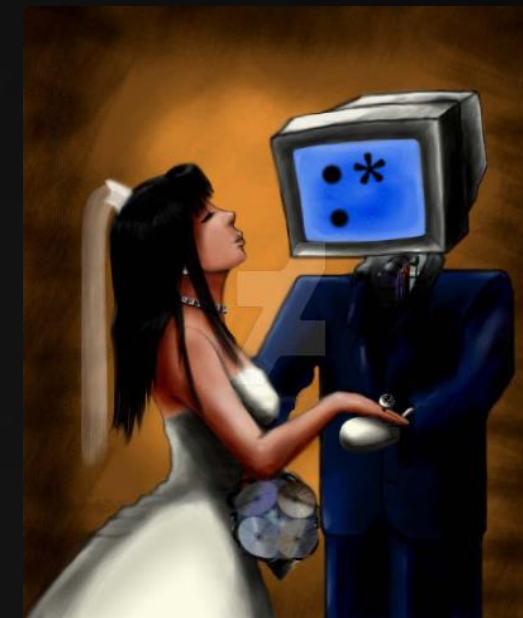
Part of it is:

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

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.



Introduction

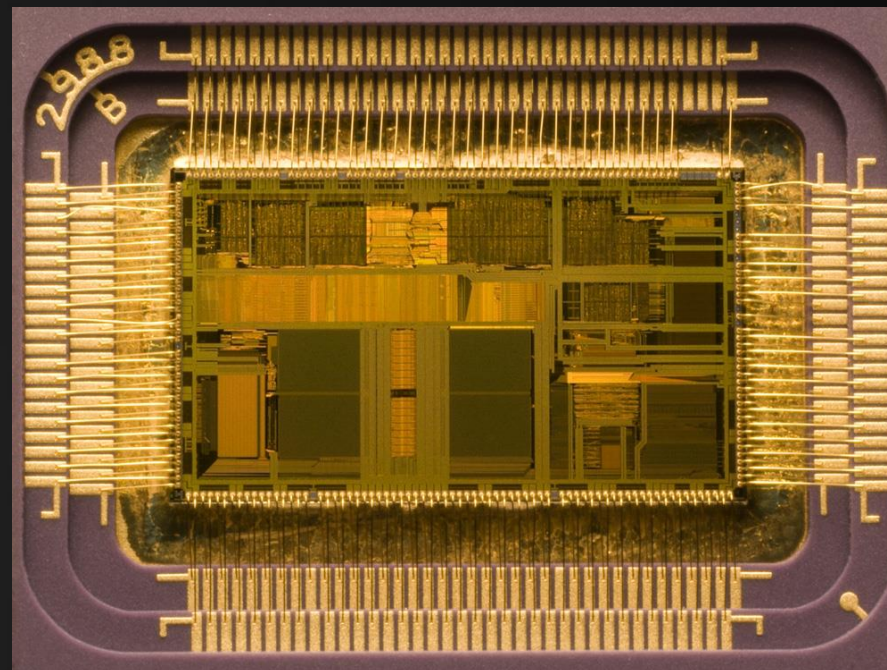
What is optimization?

Think like a CPU

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

```

...
    & (depth < MAXDEPTH)
...
    t = inside / 2;
    nt = nt / 2;
    cos2t = 1.0f - cos(2 * t);
    D, N );
}
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, E, M, &light
    e.x + radiance.y + radiance.z) > 0) && (nt
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance
...
    random walk - done properly, closely following
    survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
    
```



Introduction

What is optimization?

Work smarter, not harder: algorithm scalability

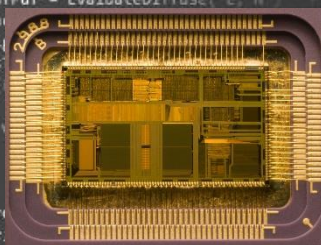
- Big O
- Research: not reinventing the wheel
- Data characteristics & algorithm choice
- STL: Trust No One
- As accurate as necessary (but not more)
- Balancing accuracy, speed and memory



```

...
    & (depth < MAXDEPTH)
...
    c = inside / n;
    nt = nt / n;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + R1 + R2);
    Tr) R = (D * nnt - N * (dot(
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse;
    estimation = doing it properly, dista
    if;
    radiance = SampleLight( &rand, E, M, &light
    e.x + radiance.y + radiance.z) > 0) && (ent
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Pa
    t3 fa
    at wei
    at cos
    E * (
...
    andom
    rive)
...
    t3 br
    urvive
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
    
```

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

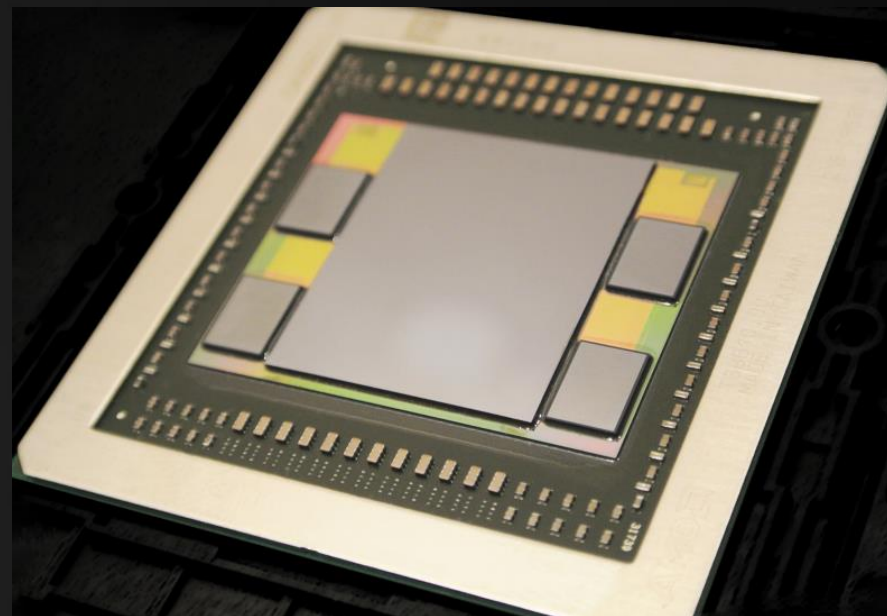


Introduction

What is optimization?

Memory hierarchy: caches

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



```

...
    & (depth < MAXDEPTH)
...
    c = inside / a;
    nt = nt / nc;
    cos2t = 1.0f - cos2t;
    D, N );
    )
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + 1) * R;
    Tr) R = (D * nnt - N * (D
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse;
    estimation = doing it properly, dist
    if;
    radiance = SampleLight( &rand, r, M, Allg
    e.x + radiance.y + radiance.z) > 0) && (ent
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * P;
    t3 fa
    at wei
    at cos
    E * (
...
    rand
    ve)
...
    t3 br
    urvive
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```

- 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

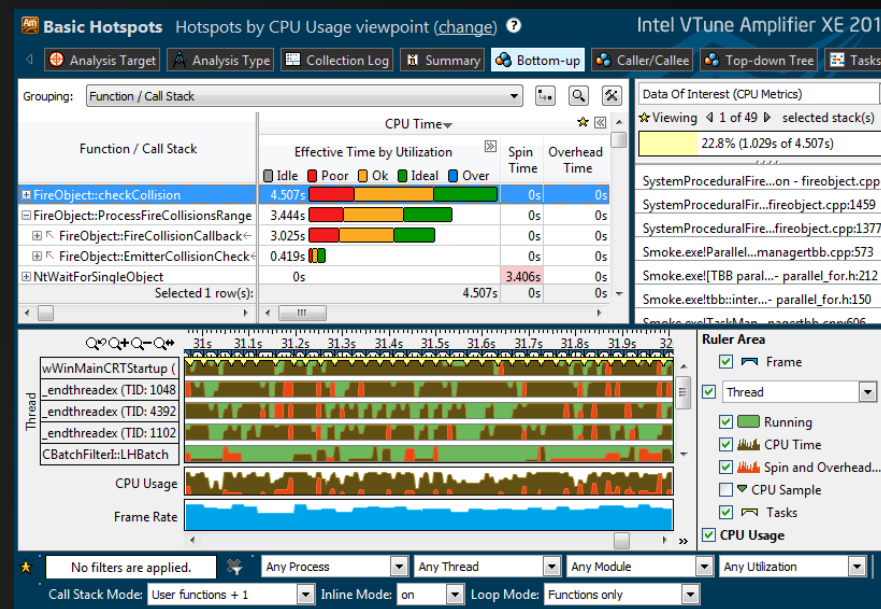


Introduction

What is optimization?

Don't assume, measure

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



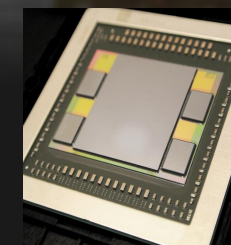
```

...
    (depth < MAXDEPTH)
...
    inside / ...
    nt = nt / ...
    os2t = 1.0f / ...
    D, N );
...
    at a = nt - nc, b = nt - ...
    at Tr = 1 - (R0 + I1 - R0)
    Tr) R = (D * nnt - N * ...
...
    E * diffuse;
    = true;
...
    fl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    efl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse
    estimation - doing it properly, dist
    if;
    radiance = SampleLight( &rand, E, M, Allig
    e.x + radiance.y + radiance.z) > 0) && (ent
...
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * ...
    at3 fai
    at wei
    at cos
    E * (
...
    random
    ive)
...
    at3 br
    urvive
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```

- 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



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



Introduction

What is optimization? – Project Management

Keeping code maintainable

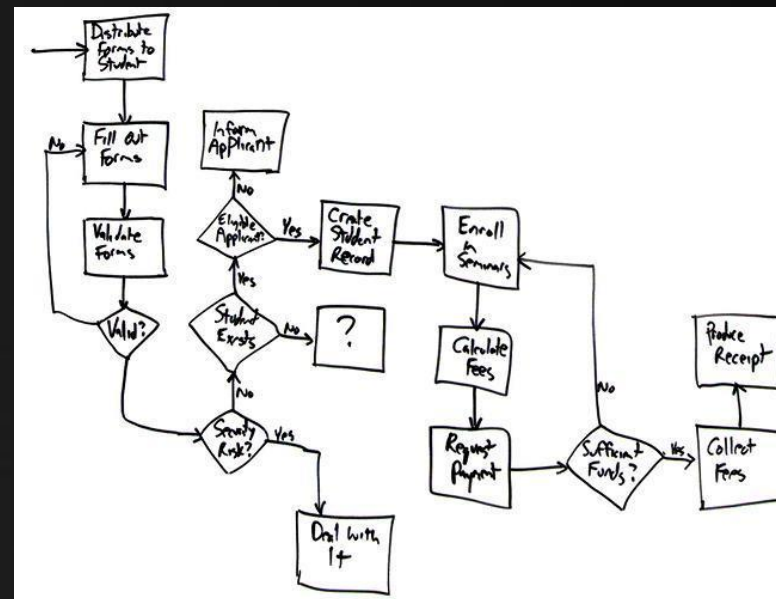
- Pareto principle / 80-20 rule: roughly 80% of the effects are caused by 20% of the causes.
- 1% of the code takes 99% of the time.

“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.*



Introduction

What is optimization?

“Perceived Performance”

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

```

ice
    & (depth < MAXDEPTH)
    {
        r = inside / 2.0;
        nt = nt / nc;
        ds2t = 1.0f - nt;
        D, N );
    }
}

at a = nt - nc, b = nt - nc;
at Tr = 1 - (R0 + I1 + R0);
Tr) R = (D * nnt - N * (ds2t

E * diffuse;
= true;

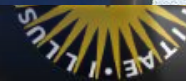
refl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffuse, r);
estimation - doing it properly, close
if;
radiance = SampleLight( @rand, r, N, @light
e.x + radiance.y + radiance.z) > 0) && (nt > 0)
w = true;
at brdfPdf = EvaluateDiffuse( L, N) * survive;
at3 factor = diffuse * INVPI;
at weight = MIs2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * radiance

andom walk - done properly, closely following
ive)

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pot
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ion = true;
    
```



Introduction

At the end of this course:

You will know how to speed up critical code by a factor 2x to 10x (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:

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

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

```

...
    & (depth < MAXDEPTH)
...
    c = inside / 2;
    nt = nt / nc;
    cos2t = 1.0f / (1.0f + cos2t);
    D, N );
)
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly,
    if;
    radiance = SampleLight( &rand, E, M,
    e.x + radiance.y + radiance.z) > 0) && (nt
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    t3 factor = diffuse * INVPI;
    at weight = MIs2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
...
random walk - done properly, closely following
    vive)
...
    t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



Formalities

Lecturer

Jacco Bikker

j.bikker@uu.nl

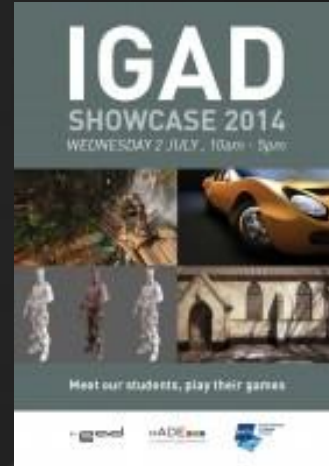
Room 4.24 BBG



```

ice
k (depth + MAXDEPTH)
+ inside / ...
nt = nt / ...
os2t = 1.0f - ...
D, N );
)
at a = nt - nc, b = nt - ...
at Tr = 1 - (R0 + I1 + R0)
Tr) R = (D * nnt - N * ...
E * diffuse;
= true;
efl + refr)) && (
D, N );
efl * E * diffuse
= true;
MAXDEPTH)
survive = Survival
estimation - doin
if;
radiance = Sample
e.x + radiance.y +
v = true;
at brdfPdf = Evalu
at3 factor = diffu
at weight = Mis2(
at cosThetaOut = c
E * ((weight * cosme
random walk - done properly, closely following the
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, AR, spot
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ion = true;

```



Formalities

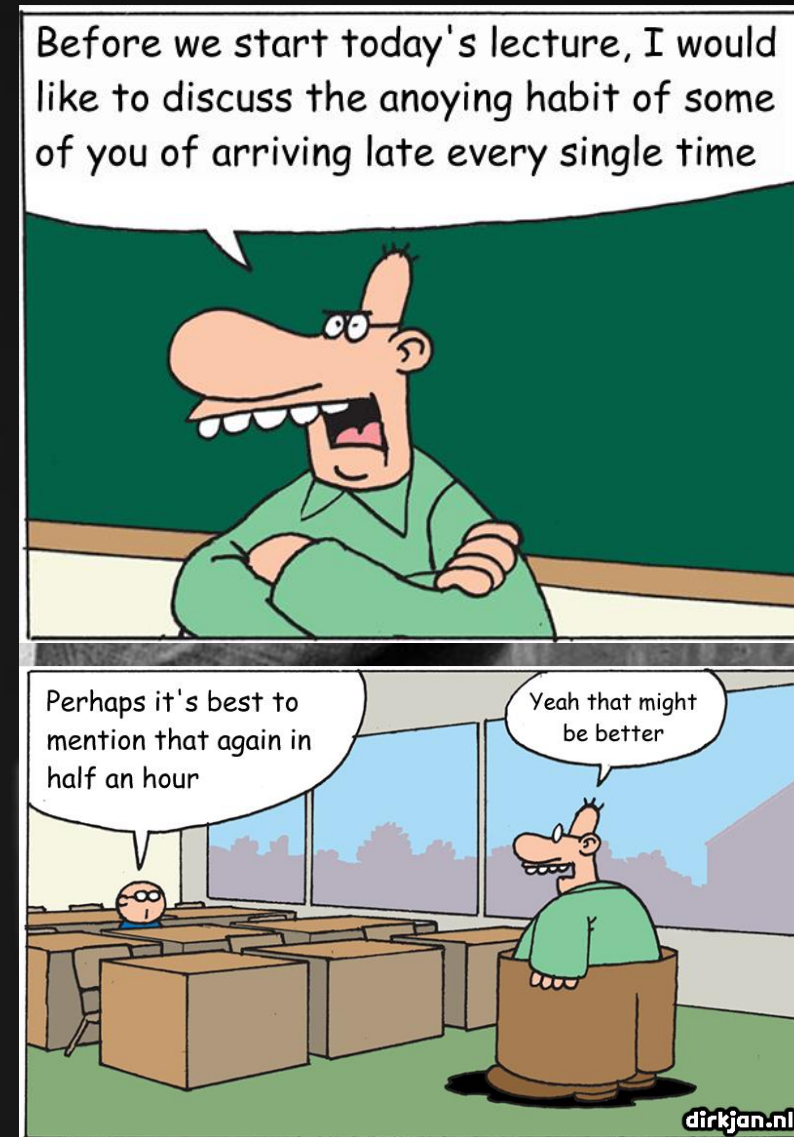
Course Layout

9 weeks + exam week:

- 2 lectures per week (for exceptions: see website)
- 1 guest lecture
- Lectures start at 09:00...
- Working class starts at 09:00, lecture at 11:00. ☺

Assessment:

- 3 assignments (20% each, individual or pairs);
- 1 final assignment (40%, individual or pairs);
- 1 final theory exam.



Formalities

Prerequisites

C++
English

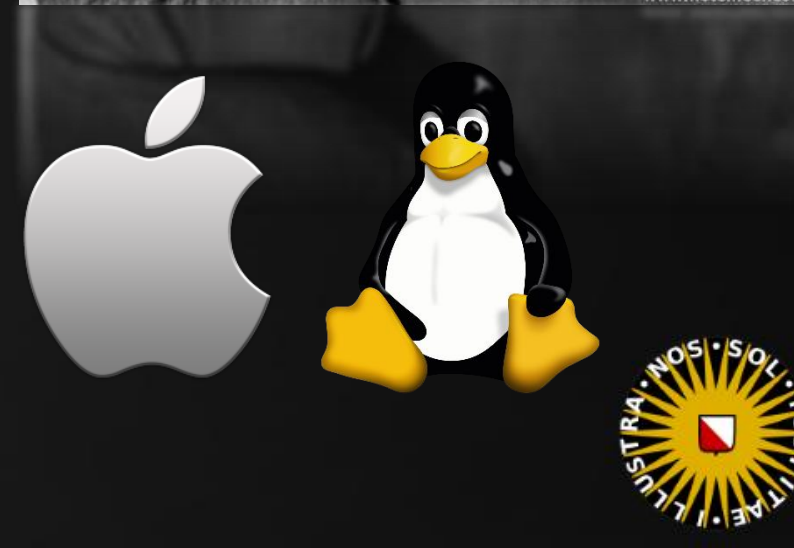
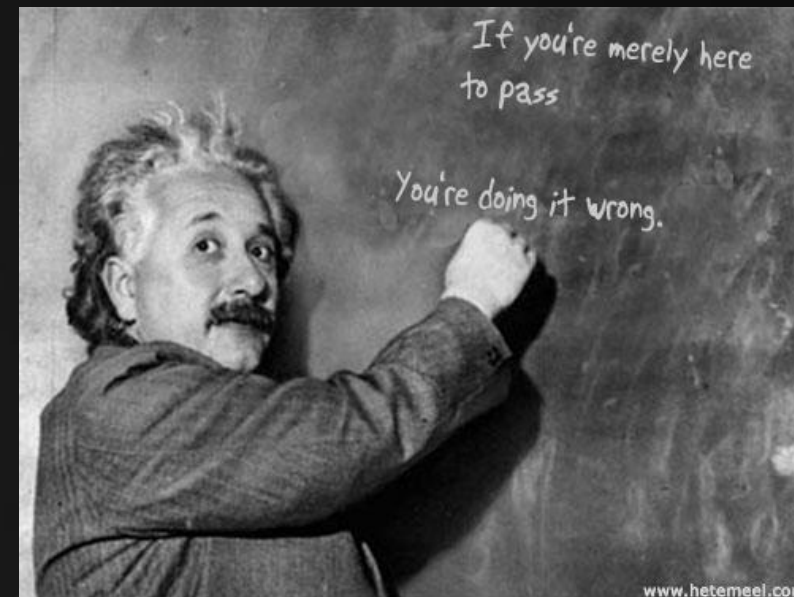
Hardware / software

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 to try for 30 days).

We will use Visual Studio 2017 (community edition).

Other tools will (also) be free.



Formalities

Literature

No book!

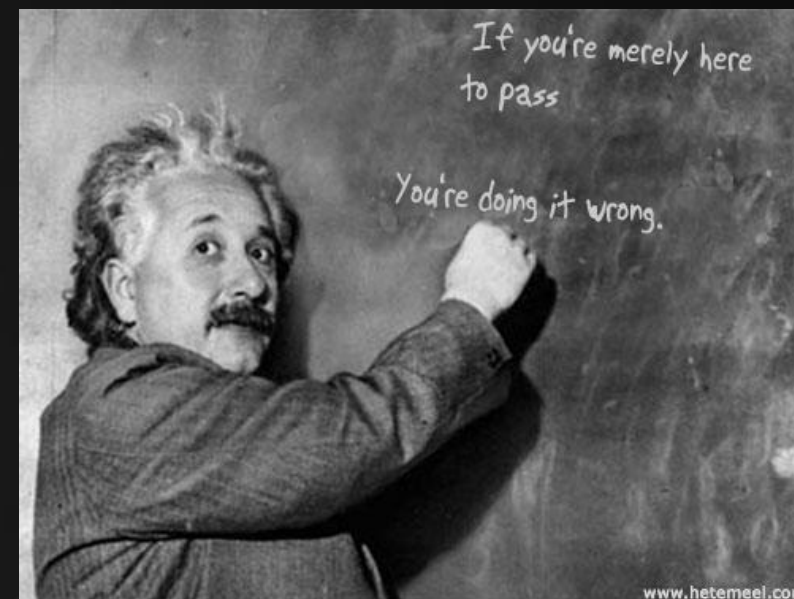
But that doesn't mean you won't be reading.

Main documents:

Agner Fog, 2004-2018, “Optimizing Software in C++”
(also see his website: <http://agner.org>)

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.



```

...
    & (depth < MAXDEPTH)
...
    c = inside / 2;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
}
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + 1) * R;
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, directly
    if;
    radiance = SampleLight( &rand, E, M, &light
    e.x + radiance.y + radiance.z) > 0) && (refl
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    at3 factor = diffuse * INVPT;
    at weight = MIs2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
...
    random walk - done properly, closely following the
    rive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    pdf;
    r = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```

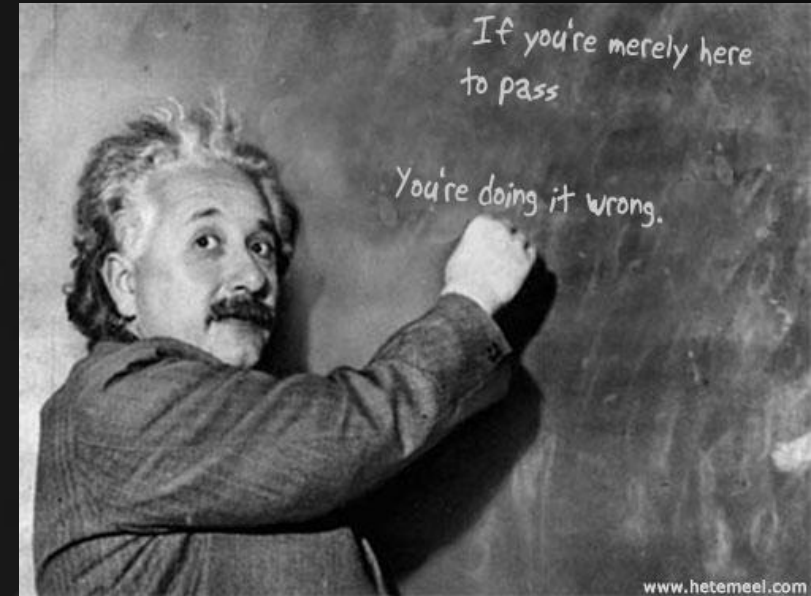


Formalities

Audience

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

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



www.hetemeel.com



```
...
    && (depth < MAXDEPTH)
{
    // Inside of a sphere
    Vec r = inside / R;
    Vec nt = nt / nc, ndd = ndd * ndd;
    double r2t = 1.0f - nt * ndd;
    Vec t = (D > N ? Vec(1, 0, 0) : Vec(0, 1, 0));
    Vec R = (D * nnt - N * (dlt > 0 ? 1 : -1));
    Vec E * diffuse;
    bool = true;
    ...
    refl + refr)) && (depth < MAXDEPTH)
{
    Vec D, N );
    refl * E * diffuse;
    = true;
    ...
    MAXDEPTH)
{
    survive = SurvivalProbability( diffuse, r );
    // estimation - doing it properly, close to Monte Carlo
    if(
    radiance = SampleLight( &rand, E, M, &light,
    e.x + radiance.y + radiance.z) > 0) && (nt > 0)
    v = true;
    Vec brdfPdf = EvaluateDiffuse( L, N ) * survive;
    Vec3 factor = diffuse * INVPT;
    Vec weight = Mts2( directPdf, brdfPdf );
    Vec cosThetaOut = dot( N, L );
    Vec E * ((weight * cosThetaOut) / directPdf) * radiance;
    ...
    random walk - done properly, closely following Monte Carlo
    survive)
{
    Vec3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    Vec pdf;
    Vec r = E * brdf * (dot( N, R ) / pdf);
    Vec3 = true;
    ...
}
```

Today's Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling



Overview

Consistent Approach

```

...
    & (depth < MAXDEPTH)
...
    c = inside ? 0 : 0;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + 1) * R;
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly,
if;
radiance = SampleLight( &rand, E, M,
e.x + radiance.y + radiance.z) > 0) && (nt
v = true;
at brdfPdf = EvaluateDiffuse( L, N) * survive;
st3 factor = diffuse * INVPI;
at weight = Mts2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * radiance
...
random walk - done properly, closely following
ive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ion = 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 step 6 and 7 until time runs out
- 9. Report.



Overview

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
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.

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.

```

...
    & (depth < MAXDEPTH)
...
    c = inside ? 0 : 0;
    nt = nt / nc;
    r1 = 1.0f - cos(2 * M_PI * r1);
    D, N );
    )
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 + R0);
    (Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly, close
if;
radiance = SampleLight( &rand, E, M, Allg
e.x + radiance.y + radiance.z) > 0) && (refr
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * num
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



Overview

Consistent Approach

```

...
    & (depth < MAXDEPTH)
...
    c = inside / N;
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
)
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 - R0);
    Tr) R = (D * nnt - N * (a0
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly,
if;
radiance = SampleLight( &rand, I, M,
e.x + radiance.y + radiance.z) > 0) && (nnt >
v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
...
random walk - done properly, closely following
survive)
;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = 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
- 9. Report.



Overview

Consistent Approach

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (inside + outside);
    nt = nt / nc;
    ns2t = 1.0f / (ns2t + 1.0f);
    D, N );
...
    at a = nt - nc, b = nt - ns;
    at Tr = 1 - (R0 + I1 + R1);
    Tr) R = (D * nnt - N * (a0
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly,
if;
radiance = SampleLight( &rand, I, M, &light
e.x + radiance.y + radiance.z) > 0) && (int
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    t3 factor = diffuse * INVPI;
    at weight = MIs2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) *
...
random walk - done properly, closely following
survive)
...
    t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = 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 / use GPGPU
 6. Profile again.
 7. 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,
 - ...
 8. Repeat steps 6 and 7 until time runs out
 9. Report.



Overview

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.

Profiling

High Level

Basic Low Level

Cache & Memory

Data-centric

Compilers

Fixed-point Arithmetic

CPU architecture

SIMD

GPGPU



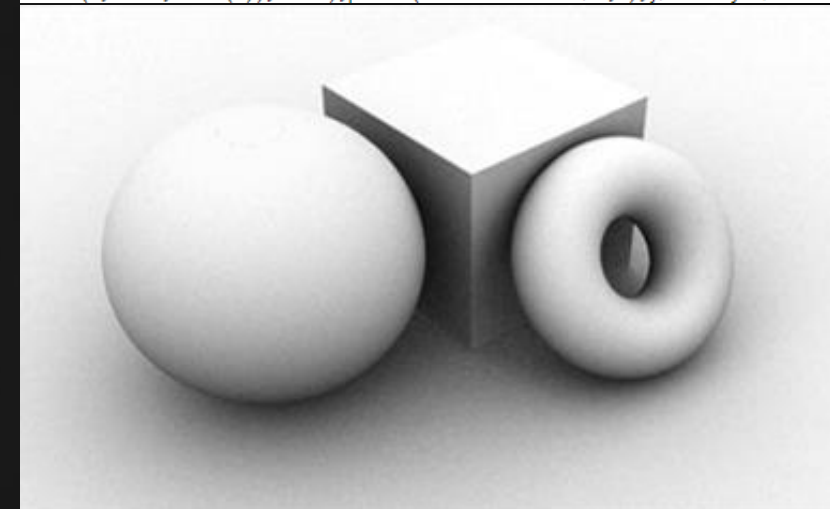
Overview

Assembler

In this course, we will not 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.

```
typedef struct{double x,y,z;}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1.,8.,8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,8,1.,7,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>1e-7&&u<tmin?best:s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=1
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)=1)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black)):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){printf("%d %d\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);/*minray!*/
```



Quotes

“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)

```

ice
(depth < MAXDEPTH)
{
    if (inside)
    {
        nt = nt / nc;
        cos2t = 1.0f / (nt * nt);
        D, N );
    }
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * (a
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    -
    MAXDEPTH)
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, I, M, &light
    e.x + radiance.y + radiance.z) > 0) && (nt > 0)
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPT;
    at weight = MIs2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
    random walk - done properly, closely following
    survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



Quotes

“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)

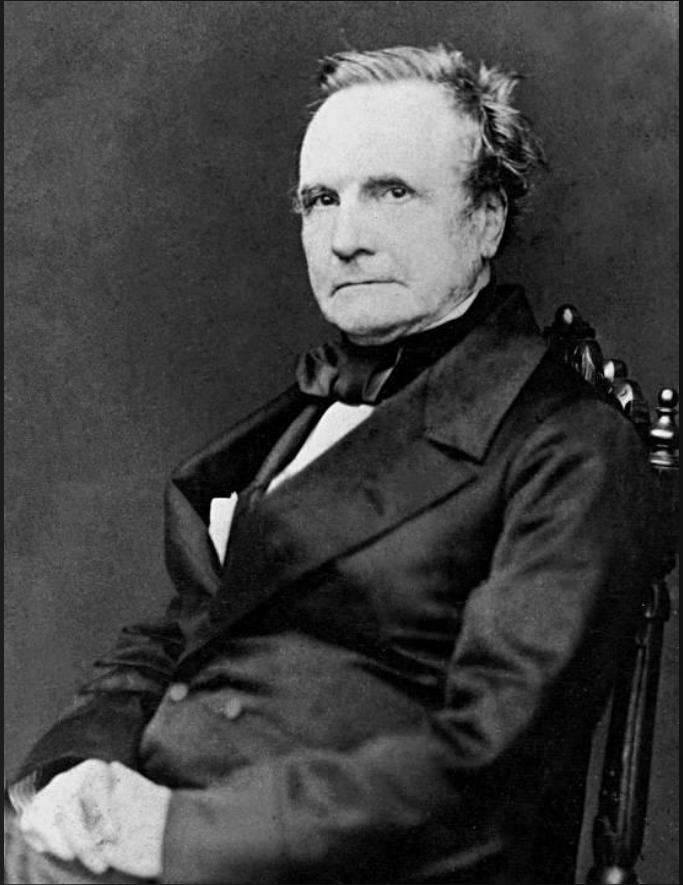
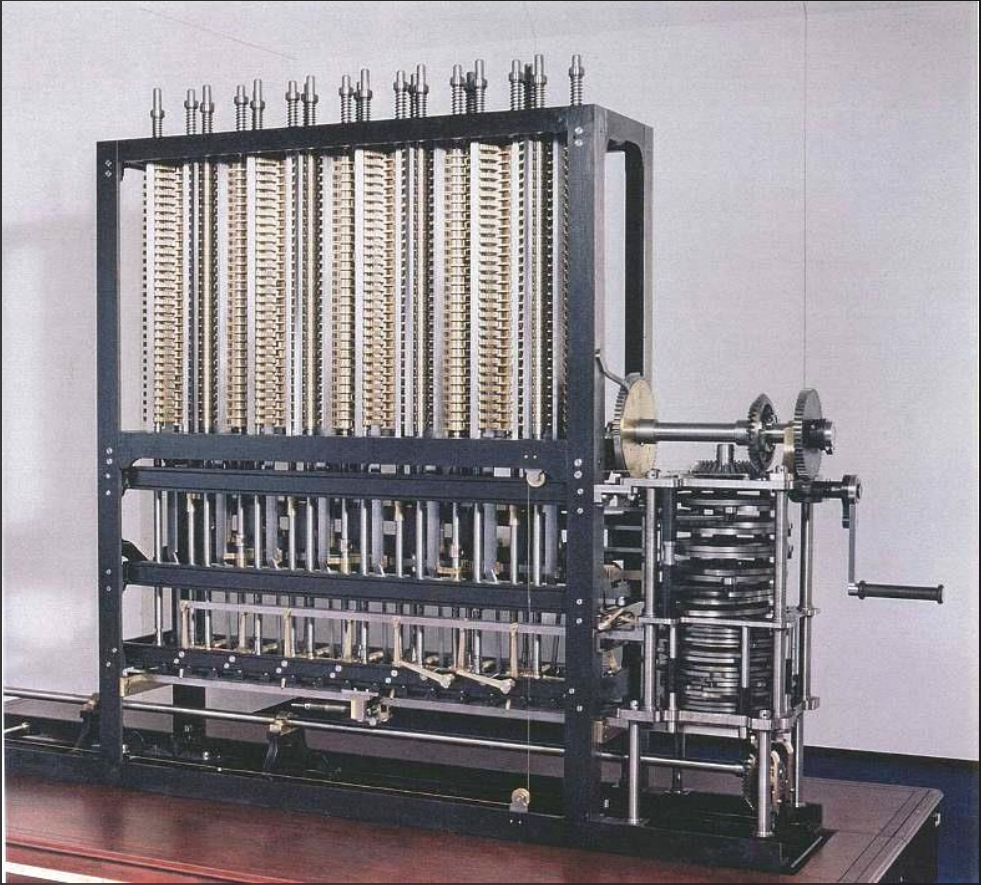


Quotes

```

ice
& (depth < MAXDEPTH)
{
    c = inside / 0.5;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
}
at a = nt - nc, b = nt - nc;
at Tr = 1 - (R0 + I1 + R0);
Tr) R = (D * nnt - N * abs
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
efl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely following the
if;
radiance = SampleLight( &rand, L, M, &light
e.x + radiance.y + radiance.z) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
at3 factor = diffuse * INVPI;
at weight = MIs2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rad
random walk - done properly, closely following the
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```



Quotes



“Dear 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.”



```
...
    && (depth < MAXDEPTH)
{
    // Inside / Outside
    bool inside = (dot(N, L) > 0);
    int nt = nt / nr; // 1000000000
    double r1 = 1.0f - exp(-nt * area);
    double r2 = 1.0f - exp(-nt * area);
    Vec D, N;
    ...
}

// Inside / Outside
int a = nt - nr, b = nt - nr;
double Tr = 1 - (Rr + (1 - Rr) * Rr);
Vec R = (D * nnt - N * (dot(D, N)));

// Diffuse
Vec E * diffuse;
...
}

// Refractive Index
double refl + refr) && (depth < MAXDEPTH)
{
    Vec D, N;
    double refl * E * diffuse;
    ...
}

// Survival Probability
double survive = SurvivalProbability( diffuse, N );
// Estimation - doing it properly, close to Monte Carlo
if(
    radiance = SampleLight( &rand, E, N, &light );
    e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
{
    Vec v = true;
    double brdfPdf = EvaluateDiffuse( L, N ) * survive;
    double factor = diffuse * INVPT;
    double weight = Mts2( directPdf, brdfPdf );
    double cosThetaOut = dot( N, L );
    Vec E * ((weight * cosThetaOut) / directPdf) * radiance;
}

// Random walk - done properly, closely following Monte Carlo
double survive)
{
    double brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot );
    survive;
    double pdf;
    Vec r = E * brdf * (dot( N, R ) / pdf);
    survive = true;
}

```

Today's Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling



Never Assume

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.*

```

...
    & (depth < MAXDEPTH)
...
    c = inside / b;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (Rb + 1) - Rb;
    Tr) R = (D * nnt - N * (ab
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly,
if;
radiance = SampleLight( &rand, E, M, Allge
e.x + radiance.y + radiance.z) > 0) && (nt
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    t3 factor = diffuse * INVPT;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
    t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



Never Assume

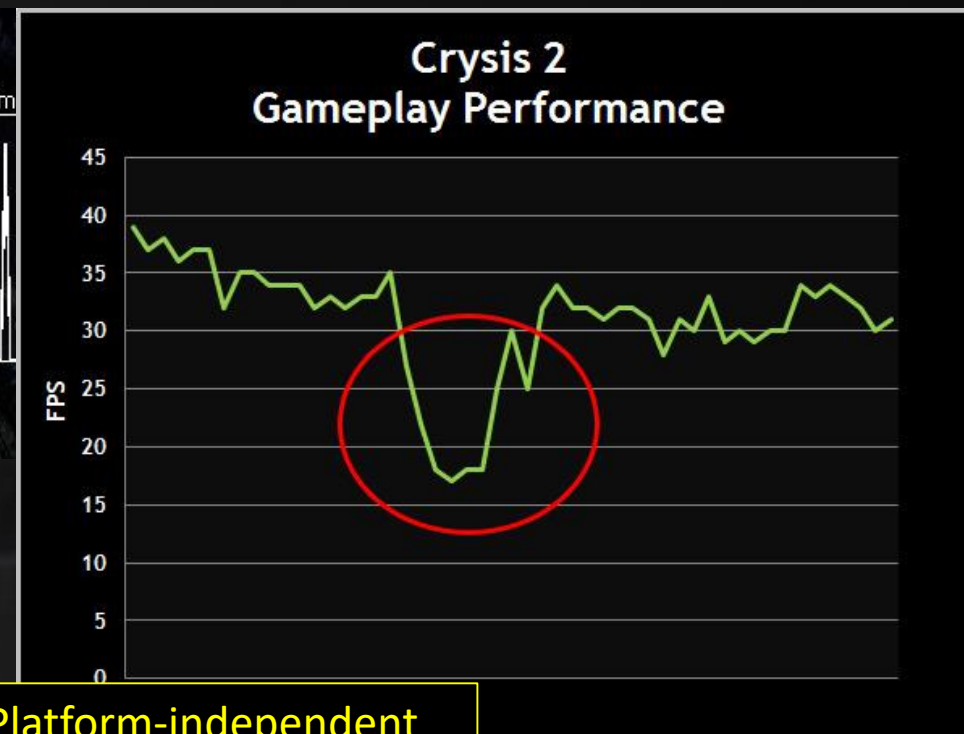
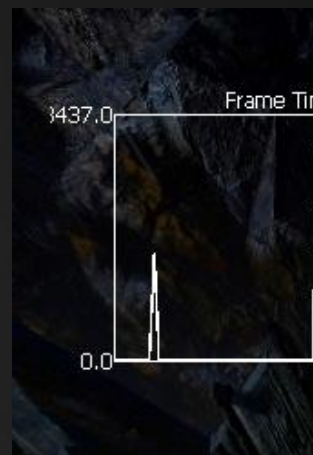
Profiling

Measuring application performance

- Using external tools
- Using timers in the code

Measurements:

- How much time is spent were? (inclusive / exclusive, cycles, percentage)
- How often is each function called?
- Low level behavior: stalls / latencies, branch mispredictions, occupation, ...
- Performance over time: lag, spikes, stutter



Platform-independent

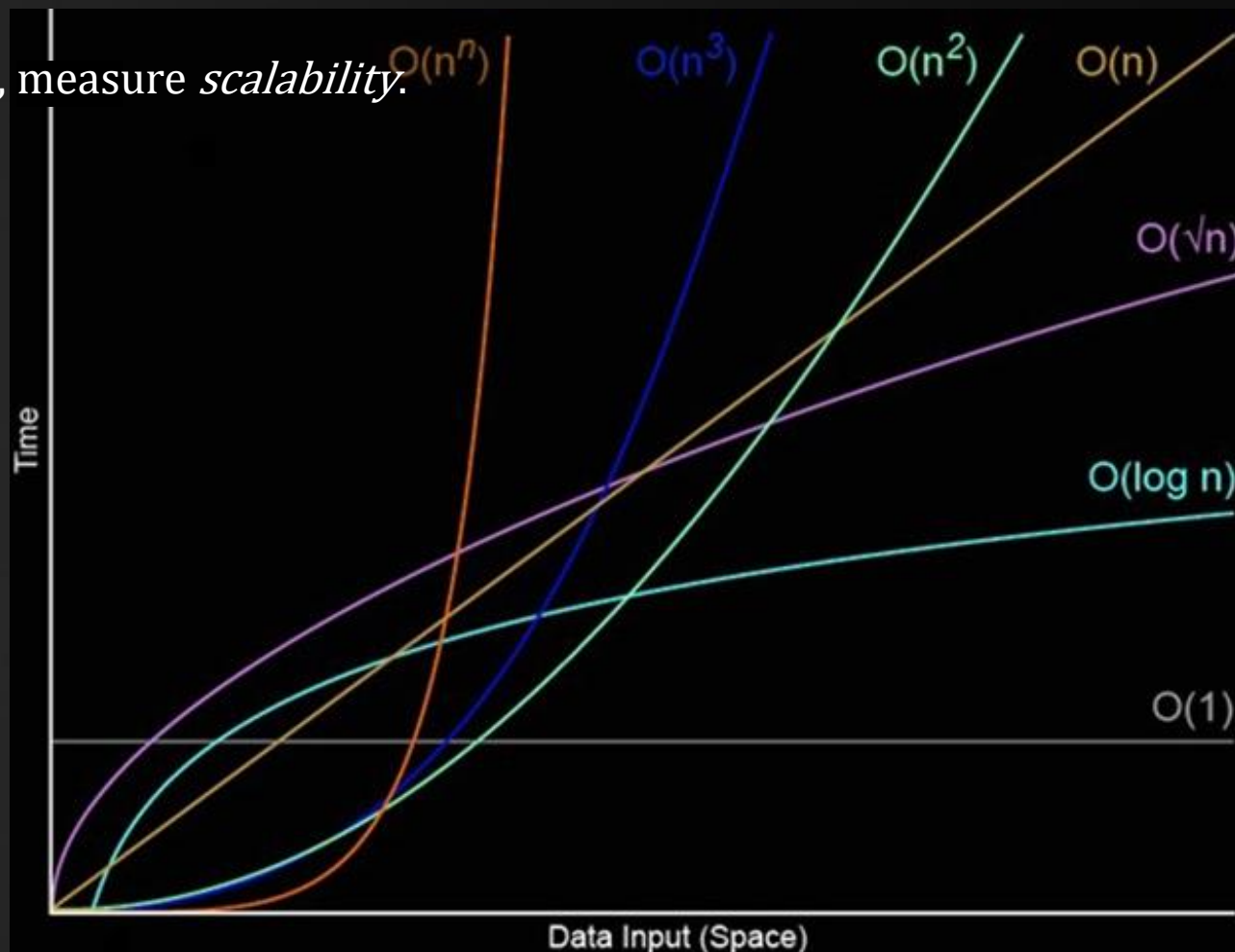
Platform-dependent



Never Assume

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

Don't just measure performance, measure *scalability*.



```

ice
(depth < MAXDEPTH)
    & (depth < MAXDEPTH)
    {
        c = inside / 4;
        nt = nt / nc;
        ns2t = 1.0f - ns2;
        D, N );
    }
}

at a = nt - nc, b = nt - nc;
at Tr = 1 - (R0 + I1 + R0);
Tr) R = (D * nnt - N * (d0

E * diffuse;
= true;

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

MAXDEPTH)

survive = SurvivalProbability( diffuse,
estimation - doing it properly, close
if;
radiance = SampleLight( &rand, E, M, Allg
e.x + radiance.y + radiance.z) > 0) && (nt
w = true;
at brdfPdf = EvaluateDiffuse( L, N) * survive;
st3 factor = diffuse * INVPT;
at weight = Mls2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * radiance

random walk - done properly, closely following
ive)

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
tion = true;
    
```



Never Assume

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.



Never Assume

The screenshot shows the Microsoft Visual Studio IDE with a C++ project named 'tpl83.00c'. The main editor displays the source code for 'game.cpp', which includes headers like 'string.h', 'stdlib.h', 'template.h', 'surface.h', and 'game.h'. A 'Template Property Pages' dialog box is open, showing the configuration for the 'Active(Relase)' configuration on the 'Active(Win32)' platform. The 'Debug Information Format' is set to 'Program Database (/ZI)'. The 'C/C++' section is expanded, and the 'Debug Information Format' dropdown is open, showing options like 'None', 'C7 compatible (/Z7)', 'Program Database (/ZI)', and 'Program Database for Edit And Continue (/ZI)'. The 'Debug Information Format' section at the bottom of the dialog explains that this setting specifies the type of debugging information generated by the compiler and must be matched with linker settings.



Never Assume

The screenshot shows the Visual Studio IDE with the 'Template Property Pages' dialog box open. The dialog is for the 'Active(Relase)' configuration on the 'Active(Win32)' platform. The 'Generate Debug Info' property is highlighted, and its value is 'Yes (/DEBUG)'. Other properties like 'Generate Program Database File' and 'Strip Private Symbols' are also visible. The 'Release' button in the top menu bar is circled in red.

Property	Value
Generate Debug Info	Yes (/DEBUG)
Generate Program Database File	No
Strip Private Symbols	Yes (/DEBUG)
Generate Map File	<inherit from parent or project defaults>
Map File Name	
Map Exports	No
Debuggable Assembly	



Tools

The screenshot shows the Visual Studio Profiler interface. The top pane displays the 'Function Details' view for the 'Simulate' function. It shows a call graph with 'Calling functions' (Tick at 53.7%, Init at 13.8%) and 'Called functions' (Bottom of Stack). The 'Current function' pane shows 'Simulate' at 67.4% and 'Function Body' at 67.4%. The bottom pane shows the 'Function Code View' for 'D:\ACTIVE\water\game.cpp', with a call stack on the right. A white box with the text 'Visual Studio Profiler' is overlaid on the bottom left of the code view.

Function	Percentage
Tick	53.7%
Init	13.8%
Simulate	67.4%
Function Body	67.4%

```

D:\ACTIVE\water\game.cpp
drop[i].prev_pos = drop[i].pos;
// simulation step 1 - move
drop[i].pos += drop[i].pos - prev_pos;
// simulation step 2 - apply gravity
drop[i].pos += gravity * 0.25f;
// simulation step 3 - satisfy constrains
for ( int step = 0; step < 3; step++ )
{
    // simulation step 3a - satisfy constraints - evade other drops
    5.4 % for ( int j = i + 1; j < DROPCOUNT; j++ )
    {
        25.8 % float dist = length( drop[i].pos - drop[j].pos );
        33.7 % if (dist < (DROPRADIUS * 2))
        {
            1.9 % vec3 direction = normalize( drop[i].pos - drop[j].pos );
            0.5 % drop[i].pos += direction * (DROPRADIUS * 2 - dist) * 0.02f;
            drop[j].pos -= direction * (DROPRADIUS * 2 - dist) * 0.02f;
        }
    }
    // simulation step 3b - satisfy constraints - evade walls
    if (drop[i].pos.y > 20) drop[i].pos.y = 19.99f - drop[i].pos.z * 0.0001f;
    if (drop[i].pos.x < -20) drop[i].pos.x = -19.99f + drop[i].pos.z * 0.0001f;
    if (drop[i].pos.x > 20) drop[i].pos.x = 19.99f - drop[i].pos.z * 0.0001f;
    if (drop[i].pos.z < -20) drop[i].pos.z = -19.99f + drop[i].pos.z * 0.0001f;
    0.1 % if (drop[i].pos.z > 20) drop[i].pos.z = 19.99f - drop[i].pos.z * 0.0001f;
}
    
```



Tools

The screenshot shows the Visual Studio performance profiler for a project named 'Very Sleepy CS'. The main window displays a list of functions with columns for Name, Exclusion, Inclusion, % Exclusive, % Inclusive, Module, and Source File. The function 'Tpl8::Game::Simulate' is highlighted, showing it is the most significant function in the profile.

Name	Exclu...	Inclusive	% Exclusive	% Inclusive	Module	Source File
Tpl8::Game::Simulate		9.47s	62.76%	62.76%	water	d:\water\game...
Tpl8::Game::SmoothWater		2.01s	13.34%	13.34%	water	d:\water\game...
Tpl8::Game::DrawTriangle		1.33s	8.80%	8.80%	water	d:\water\game...
Tpl8::Game::RenderZSprites		1.19s	7.86%	7.86%	water	d:\water\game...
Tpl8::Game::RenderWaterSurface		0.43s	2.87%	11.67%	water	d:\water\game...
Tpl8::Surface::Clear		0.11s	0.75%	0.75%	water	d:\water\surfac...
Tpl8::Game::RenderDebugInfo		0.03s	0.19%	0.32%	water	d:\water\game...
Tpl8::Surface::Plot		0.02s	0.13%	0.13%	water	d:\water\surfac...
Tpl8::Game::DownScale		0.02s	0.10%	0.10%	water	d:\water\game...
Tpl8::Game::TimeSmooth		0.00s	0.02%	0.02%	water	d:\water\game...
Tpl8::Surface::AddLine		0.00s	0.01%	0.01%	water	d:\water\surfac...
swap		0.00s	0.01%	1.66%	water	d:\water\templa...
[006ADCD0]		0.00s	0.00%	0.01%	water	water
__tmainCRTStartup		0.00s	0.00%	99.93%	water	f:\dd\vctools\cr...
SDL_main		0.00s	0.00%	99.57%	water	d:\water\templa...
Tpl8::Game::Tick		0.00s	0.00%	91.96%	water	d:\water\game...
Tpl8::Game::DrawBoat		0.00s	0.00%	0.01%	water	d:\water\game...
Tpl8::Game::GlowLine		0.00s	0.00%	0.01%	water	d:\water\game...

The 'Averages' pane on the right shows the following data:

Name	Samples	% Calls	Module
Tpl8::Game::Tick	8.70s	91.88%	water
Tpl8::Game::Init	0.77s	8.12%	water

The 'Source' pane shows the following code snippet:

```

for ( int step = 0; step < 3; step++ )
{
    // simulation step 3a - satisfy constraints - evade other drops
    for ( int j = i + 1; j < DROPCOUNT; j++ )
    {
        float dist = (drop[i].pos - drop[j].pos).Length();
        if (dist < (DROPRADIUS * 2))
        {
            vector3 direction = (drop[i].pos - drop[j].pos).Normalized;
            drop[i].pos += direction * (DROPRADIUS * 2 - dist) * 0.02f;
            drop[j].pos -= direction * (DROPRADIUS * 2 - dist) * 0.02f;
        }
    }
    // simulation step 3b - satisfy constraints - evade walls
    if (drop[i].pos.y > 20) drop[i].pos.y = 19.99f - drop[i].pos.z * 0.0001f;
    if (drop[i].pos.x < -20) drop[i].pos.x = -19.99f + drop[i].pos.z * 0.0001f;
    if (drop[i].pos.x > 20) drop[i].pos.x = 19.99f - drop[i].pos.z * 0.0001f;
    if (drop[i].pos.z < -20) drop[i].pos.z = -19.99f + drop[i].pos.z * 0.0001f;
    if (drop[i].pos.z > 20) drop[i].pos.z = 19.99f - drop[i].pos.z * 0.0001f;
}
    
```

A call stack pane on the right shows 'Child Calls' with columns for Name, Samples, % Calls, and Module.

VerySleepy



Tools

Basic Hotspots Hotspots by CPU Usage viewpoint (change) ? Intel VTune Amplifier XE 2015

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks

Grouping: Function / Call Stack

Function / Call Stack	CPU Time					Spin Time	Overhead Time
	Effective Time by Utilization						
	Idle	Poor	Ok	Ideal	Over		
FireObject::checkCollision	4.507s					0s	0s
FireObject::ProcessFireCollisionsRange	3.444s					0s	0s
FireObject::FireCollisionCallback<	3.025s					0s	0s
FireObject::EmitterCollisionCheck<	0.419s					0s	0s
NtWaitForSingleObject	0s					3.406s	0s
Selected 1 row(s):					4.507s	0s	0s

Data Of Interest (CPU Metrics)

★ Viewing 1 of 49 selected stack(s)

22.8% (1.029s of 4.507s)

SystemProceduralFire...on - fireobject.cpp

SystemProceduralFir...fireobject.cpp:1459

SystemProceduralFire...fireobject.cpp:1377

Smoke.exe!Parallel...managertbb.cpp:573

Smoke.exe!TBB paral... - parallel_for.h:212

Smoke.exe!tbb::inter... - parallel_for.h:150

Smoke.exe!TaskMan...nagertbb.cpp:606

Thread

wWinMainCRTStartup (

_endthreadex (TID: 1048

_endthreadex (TID: 4392

_endthreadex (TID: 1102

CBatchFilter::LHBatch

CPU Usage

Frame Rate

Ruler Area

- Frame
- Thread
- Running
- CPU Time
- Spin and Overhead...
- CPU Sample
- Tasks
- CPU Usage

No filters are applied. Any Process | Any Thread | Any Module | Any Utilization

Call Stack Mode: User functions + 1 | Inline Mode: on | Loop Mode: Functions only

Intel VTune



Tools

MyApp - CodeXL | Profile Mode (CPU: Time-based Sampling)

CodeXL Profile Session: CPU: Sep 24, 2012 02:25:38

Process: 7736

Functions (153 functions, 15 shown)

Function	# of Paths	Path Samples	Avg. Samples per Path	Self Samples	Deep Samples	% of Deep Samples	Source File	Module
pow	39	211	5.4	25	211	58%	math.h(498)	MyApp.exe
mainCRTStartup	58	202	3.5		202	56%	crtexe.c(361)	MyApp.exe
__tmainCRTStartup	58	202	3.5		202	56%	crtexe.c(378)	MyApp.exe
main	57	201	3.5		201	56%	myapp.cpp(10)	MyApp.exe
Worker::doWork	57	201	3.5	4	201	56%	worker.cpp(6)	MyApp.exe
_Pow_int<double>	16	152	9.5	142	152	42%	math.h(484)	MyApp.exe
SemiWorker::doAsyncWork	51	137	2.7		137	38%	semiworker.cpp(43)	MyApp.exe
SemiWorker::Calc	35	118	3.4	6	118	33%	semiworker.cpp(28)	MyApp.exe

Immediate Ancestors and Children of function: "pow"

Parents	Func Samples	Deep Samples	% of Deep Samples	Self + Children	Func Samples	Deep Samples	% of Deep Samples
Worker::doWork	4	201	46%	_Pow_int<double>	142	152	44%
SemiWorker::Calc	6	118	27%	(self)	25	(25 self)	7%

Paths containing function: pow

Function	Self Samples	Downstream Samples	Downstream Samples %
▲ SemiWorker::doAsyncWork		83	39%
▲ SemiWorker::Calc		83	39%
▷ pow	12	71	34%
▲ mainCRTStartup		128	61%
▲ __tmainCRTStartup		128	61%
▲ main		128	61%

AMD CodeXL



Never Assume

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.

```

...
    & (depth < MAXDEPTH)
...
    + inside / ...
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
)
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * ...
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse, ...
    estimation - doing it properly, ...
    if;
    radiance = SampleLight( &rand, I, M, ...
    e.x + radiance.y + radiance.z) > 0) && (nt > 0)
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
...
random walk - done properly, closely following ...
    survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```

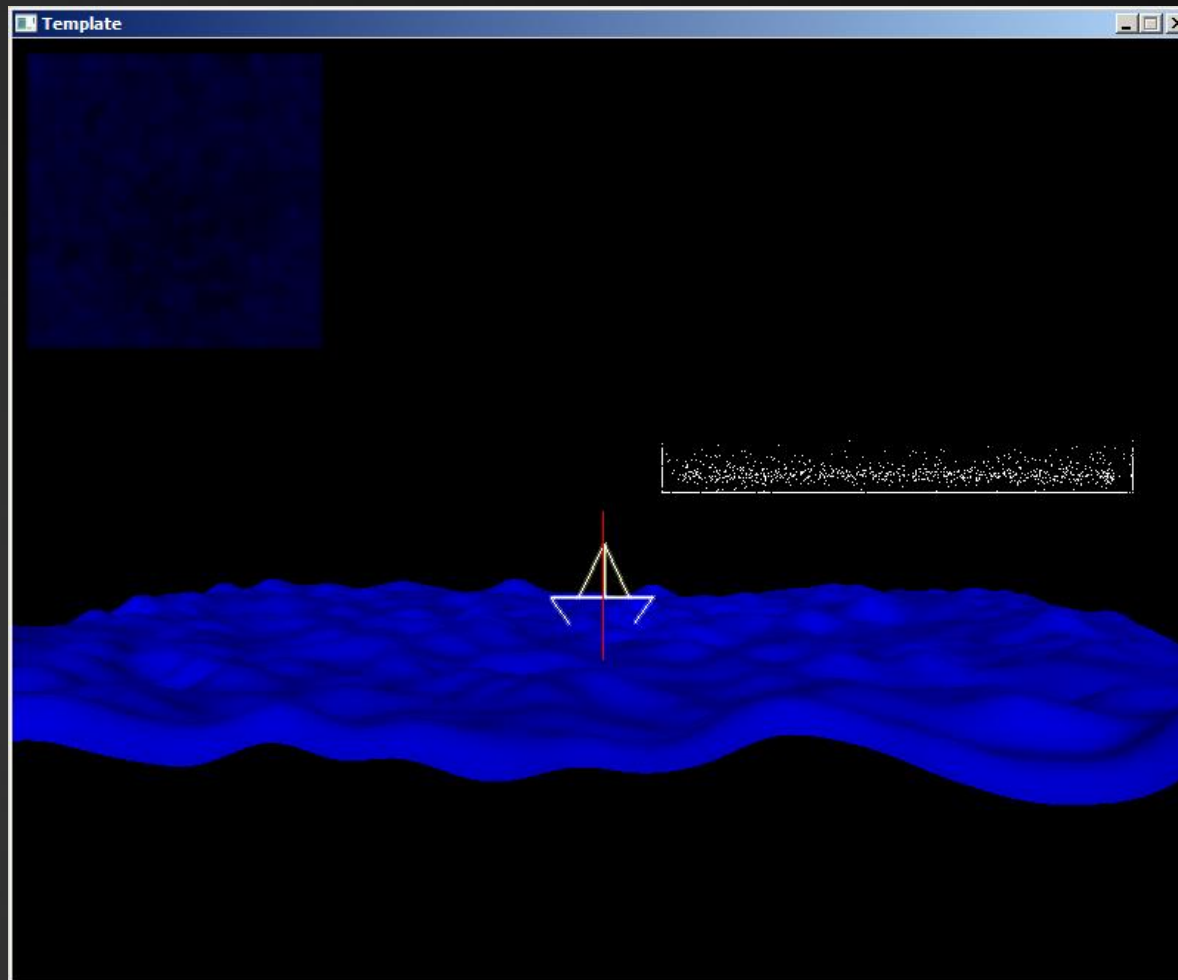


Profiler Output

```

...
    & (depth < MAXDEPTH)
...
    c = inside / (n * d);
    nt = nt / nc; ddt = d * d;
    cos2t = 1.0f - nt * nt;
    D, N );
}
...
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * ddc)
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, I, M, Allg
    e.x + radiance.y + radiance.z) && (nt > 0)
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    st3 factor = diffuse * INVPI;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance
...
    random walk - done properly, closely following
    ve)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



Profiler Output

Very Sleepy CS - C:\Users\Jacco\AppData\Local\Temp\B916.tmp

File View Help

Functions

Name	Exclu...	Inclusive	% Exclusive	% Inclusive	Module	Source File	Sourc...	Address
Tmpl8::Game::Simulate	3.06s	3.06s	67.89%	67.89%	Template	f:\projects\water\game.cpp	97	0x401761
Tmpl8::Game::SmoothWater	0.47s	0.47s	10.54%	10.54%	Template	f:\projects\water\game.cpp	206	0x401c20
Tmpl8::Game::RenderZSprites	0.32s	0.32s	7.18%	7.18%	Template	f:\projects\water\game.cpp	170	0x401a59
Tmpl8::Game::DrawTriangle	0.32s	0.32s	7.14%	7.14%	Template	f:\projects\water\game.cpp	119	0x402a6e
Tmpl8::Game::RenderWaterSurface	0.11s	0.44s	2.52%	9.66%	Template	f:\projects\water\game.cpp	278	0x40262e
Tmpl8::Surface::Plot	0.01s	0.01s	0.31%	0.31%	Template	f:\projects\water\surface.cpp	177	0x403910
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	0x40299f
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\dlstuf...	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	0x40146c
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	0x40101f

Averages Call Stacks Filters

Called From

Name	Samples	% Calls
Tmpl8::Game::Tick	2.17s	70.83%
Tmpl8::Game::Init	0.89s	29.17%

Child Calls

Name	Samples	% Calls
------	---------	---------

Source Log

```

// simulation step 1 - move
drop[i].pos += drop[i].pos - prev_pos;
// simulation step 2 - apply gravity
drop[i].pos += gravity * 0.25f;
// simulation step 3 - satisfy constraints
for ( int step = 0; step < 3; step++ )
{
    // simulation step 3a - satisfy constraints - evade other drops
    for ( int j = i + 1; j < DROPCOUNT; j++ )
    {
        float dist = length( drop[i].pos - drop[j].pos );
        if ( dist < ( DROPRADIUS * 2 ) )
        {
            vec3 direction = normalize( drop[i].pos - drop[j].pos );
            drop[i].pos += direction * ( DROPRADIUS * 2 - dist ) * 0.02f;
            drop[j].pos -= direction * ( DROPRADIUS * 2 - dist ) * 0.02f;
        }
    }
}

```

Source file: f:\projects\water\game.cpp Line 25



Profiler Output

Profiling – Results

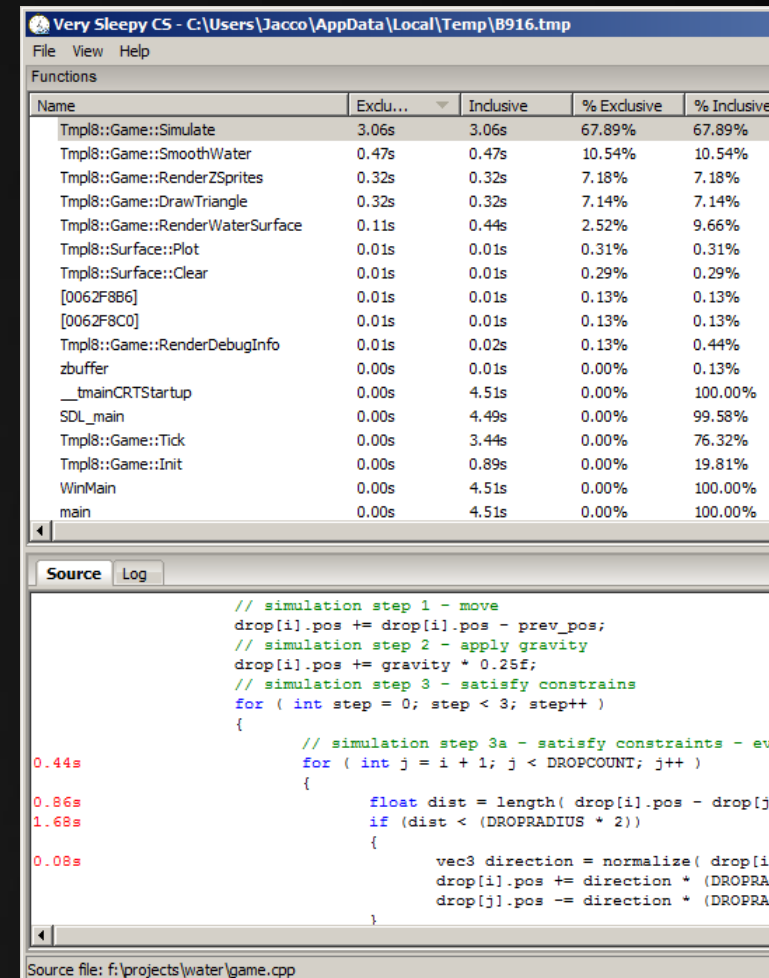
Game::Simulate	67.89%	67.89%
Game::SmoothWater	10.54%	10.54%
Game::RenderZSprites	7.18%	7.18%
Game::Tick	0.00%	76.32%

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

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

and 1.68s on this line:

```
if (dist < (DROPRADIUS * 2))
```



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



➔ 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.

```

ice
    & (depth < MAXDEPTH)
    {
        // ...
        t = inside / (n * n);
        nt = nt / nc;
        cos2t = 1.0f - nt;
        D, N );
    }
}

at a = nt - nc, b = nt - nc;
at Tr = 1 - (R0 + I1 + R0);
Tr) R = (D * nnt - N * t);

E * diffuse;
= true;

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

MAXDEPTH)

survive = SurvivalProbability( diffuse,
estimation - doing it properly, close
if;
radiance = SampleLight( &rand, E, M, Allg
e.x + radiance.y + radiance.z) > 0) && (refl
y = true;
at brdfPdf = EvaluateDiffuse( L, N) * Survive;
st3 factor = diffuse * INVPI;
at weight = Mls2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) *

andom walk - done properly, closely following the
ive)

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



Profiler Output

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'tmp184.00a_2013' (1 proj)

- Amplifier XE Results
 - Template
 - r000hs
 - r001ge
 - r002ths
 - r003ge
 - Template
 - External Dependencies
 - _readme.txt
 - game.cpp
 - game.h
 - surface.cpp
 - surface.h
 - template.cpp
 - template.h
 - threads.cpp
 - threads.h

r003ge game.cpp

General Exploration viewpoint (change) ?

Analysis Target Analysis Type Collection Log Summary Bottom-up PMU Events Tasks and Frames game.cpp game.cpp

Elapsed Time: 11.571s

- Paused Time: 3.150s
- Clockticks: 3,710,005,565
- Instructions Retired: 7,236,010,854
- CPI Rate: 0.513

Filled Pipeline Slots: 0.774

- Retiring: 0.774**
%RetiredPipelineSlotsIssueTextAll
General Retirement: 0.767
This metric represents a fraction of slots during which CPU was retiring uOps not originated from the Microcode Sequencer. This correlates with the total number of instructions executed by the program. A uOps-per-Instruction ratio of 1 is expected. If the Retirement value for non-vectorized code is high, consider vectorizing your code to reduce instructions and hence this bucket.
Microcode Sequencer: 0.007
- Bad Speculation: 0.107**
A significant proportion of pipeline slots containing useful work are being cancelled. This can be caused by mispredicting branches or by machine clears.
Branch Mispredict: 0.000
Machine Clears: 0.107
A significant portion of execution time is spent handling machine clears. Examine the MACHINE_CLEAR events to determine the specific cause.

Unfilled Pipeline Slots (Stalls): 0.070

- Back-End Bound: 0.070**
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 the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.
Memory Bound: 0.000
Core Bound: 0.493
- Front-End Bound: 0.049**

CPU Usage Histogram

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.

Simultaneously Utilized Logical CPUs	Category
0	Idle
1	Poor
2	Poor
3	Poor
4	Poor
5	Ok
6	Ok
7	Ok
8	Ideal
9	Ideal



Profiler Output

So. Lin. ▲	Source	Clockticks	Instructions Retired	CPI Rate	Filled Pipeline Slots		Unfilled Pipeline Slo ...	
					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	for (int i = 0; i < DROP_COUNT; i++)							
135	{							
136	gravity = vec3(sin(angle * PI / 180), cos(angle * PI / 180), 0);							
137	vec3 prev_pos = drop[i].prev_pos;							
138	drop[i].prev_pos = drop[i].pos;							
139	// simulation step 1 - move							
140	drop[i].pos += drop[i].pos - prev_pos;							
141	// simulation step 2 - apply gravity							
142	drop[i].pos += gravity * 0.25f;							
143	// simulation step 3 - satisfy constraints							
144	for (int step = 0; step < 3; step++)	0	2,000,003	0.000	0.000	0.000	1.000	0.000
145	{							
146	// simulation step 3a - satisfy constraints - evade other drops							
147	for (int j = i + 1; j < DROP_COUNT; j++)	128,000,192	148,000,222	0.865	0.703	0.000	0.824	0.059
148	{							
149	float dist = length(drop[i].pos - drop[j].pos);	780,001,170	208,000,312	3.750	1.000	0.000	0.231	0.010
150	if (dist < (DROPRADIUS * 2))	1,042,001,563	2,818,004,227	0.370	0.115	0.446	0.381	0.058
151	{							
152	vec3 direction = normalize(drop[i].pos - drop[j].pos);	64,000,096	86,000,129	0.744	0.820	0.000	0.180	0.000
153	drop[i].pos += direction * (DROPRADIUS * 2 - dist) * 0.02f;	22,000,033	0		0.000	1.000	0.000	1.000
154	drop[j].pos -= direction * (DROPRADIUS * 2 - dist) * 0.02f;	6,000,009	2,000,003	3.000	0.000	0.000	1.000	0.000
155	}							
156	}							
157	// simulation step 3b - satisfy constraints - evade walls							
158	if (drop[i].pos.y > 20) drop[i].pos.y = 19.99f - drop[i].pos.z * 0.000							
159	if (drop[i].pos.x < -20) drop[i].pos.x = -19.99f + drop[i].pos.z * 0.0							
160	if (drop[i].pos.x > 20) drop[i].pos.x = 19.99f - drop[i].pos.z * 0.000							
161	if (drop[i].pos.z < -20) drop[i].pos.z = -19.99f + drop[i].pos.z * 0.0							
162	if (drop[i].pos.z > 20) drop[i].pos.z = 19.99f - drop[i].pos.z * 0.000							
163	}							
164	}							
165	LEAVE(SIMULATE);							
166	}							



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 vendor-specific and will be similar on similar hardware.

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



```

...
    & (depth < MAXDEPTH)
...
    c = inside / n;
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
}
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I) * R0;
    Tr) R = (D * nnt - N * c);
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, E, M, Allg
    e.x + radiance.y + radiance.z) > 0) && (ref
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * naur
    at3 factor = diffuse * INVPT;
    at weight = MIs2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
    random walk - done properly, closely following the
    rive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

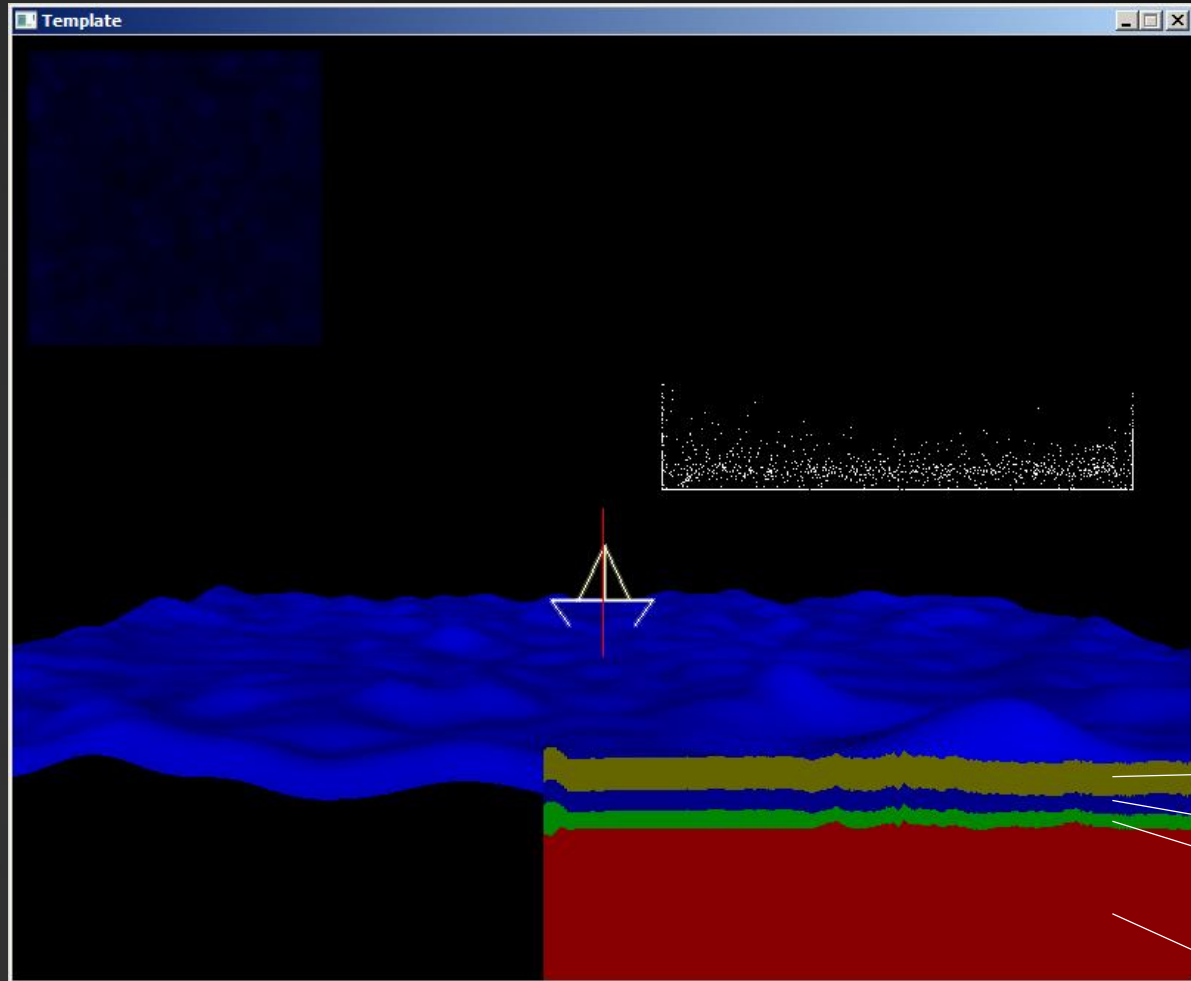
```



Custom Profiling

```

ice
    & (depth < MAXDEPTH)
    {
        r = inside / (inside + outside);
        nt = nt / nc;
        ns2t = 1.0f - ns2;
        D, N );
    }
    {
        at a = nt - nc, b = nt - ns;
        at Tr = 1 - (R0 + I1 + R0);
        Tr) R = (D * nnt - N * (D0
    }
    E * diffuse;
    = true;
    }
    {
        refl + refr) && (depth < MAXDEPTH)
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, I, M, Allg
    e.x + radiance.y + radiance.z) > 0) && (n
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance
    random walk - done properly, closely following
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    r = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```



- smoothwater
- rendersprite
- drawtriangle
- simulate



Custom Profiling

Minecraft Beta 1.9 Prerelease 2 (118 fps, 2 chunk updates) Used memory: 41% (412MB) of 989MB
 C: 979/5408, F: 1912, O: 0, E: 2517 Allocated memory: 190% (989MB)
 E: 1/251, B: 0, I: 259
 P: 0, T: All: 251
 ServerChunkCache: 979 Drop: 0

X: -12.898761435882818
 Y: 79.620000000476897
 Z: 231.78292536730885
 F: 3

Seed: 5130996236305320162

level 82.58%
 textures 11.06%
 animateTick 5.61%
 pick 0.31%
 ??? 0.23%
 gameRenderer 0.07%
 keyboard 0.03%
 gameMode 0.02%
 stats 0.01%
 particles 0.01%
 mouse 0.0%
 gui 0.0%
 centerChunkSource 0.0%
 levelRenderer 0.0%

Minecraft



Custom Profiling

```

...
    & (depth < MAXDEPTH)
...
    c = inside / b;
    nt = nt / nc;
    cos2t = 1.0f - nt * nt;
    D, N );
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, die
    if;
    radiance = SampleLight( &rand, I, M, Allg
    e.x + radiance.y + radiance.z) > 0) && (n
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * Survive
    at3 factor = diffuse * INVPT;
    at weight = MIs2( directPdf, brdfPdf )
    at cosThetaOut = dot( N, R )
    E * ((weight * cosThetaOut) / directPdf
...
    random walk - done properly, closely following the
    (survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```

UnrealEngine 3
 FPS: 54.7 (Avg over 20 frames) | Tris/Frame: 623/34 | Time: 601.9s | Speed: 11 | Shaders Used (Modified) | RS Used (Modified)

Paused

LIGHTING NEEDS TO BE REBUILT
 Rendering frozen...

2T VT CT VS PS BU IB RT DSS MSC

DrawPrim Count Avg Batch

Driver Time (ms) GPU Idle (ms) Driver Sleeping (ms) Frame Time (ms)

ACP (MB) VID (MB)

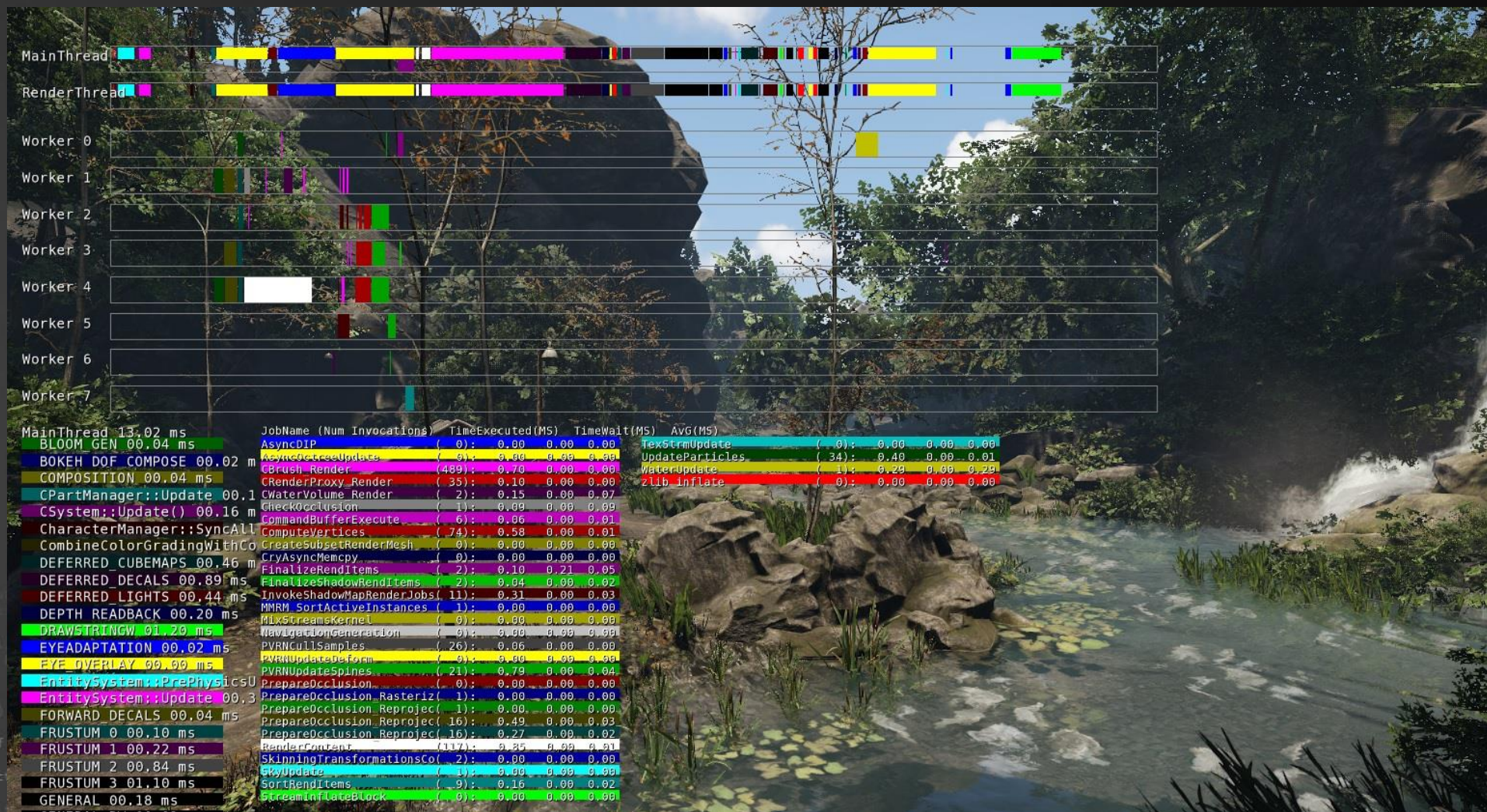
Performance Dashboard | Debug Console | Frame Debugger | Frame Profiler

Copyright © 2007 Epic Games, Inc. Cary, N.C., USA. ALL RIGHTS RESERVED. Epic, Unreal, and Circle U logo are registered trademarks of Epic Games, Inc. in the United States of America and elsewhere.

UnrealEngine 3



Custom Profiling



CryEngine



Custom Profiling

```

...
    & (depth < MAXDEPTH)
...
    c = inside / 4.0;
    nt = nt / nc;
    cos2t = 1.0f - nt * nt;
    D, N );
)
...
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + I1 + R0);
    Tr) R = (D * nnt - N * (dot
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, I, M, Allge
    e.x + radiance.y + radiance.z) > 0) && (nt
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    t3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance
...
    random walk - done properly, closely following
    survive)
...
    t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



StarCraft II



Custom Profiling

```

...
}

void RayTrace(
    Ray * r,
    int depth, int MAXDEPTH,
    Vec * inside, Vec * outside,
    Vec * nt, Vec * ndo, Vec * n,
    Vec * ns2t = 1.0f, Vec * ns2n,
    Vec * D, Vec * N );

...

Vec * a = nt - nc, b = nt - nc;
Vec * Tr = 1 - (RB * (1 + RB));
Vec * R = (D * nnt - N * (dot(
...
Vec * E * diffuse;
Vec * refl = true;

Vec * refl + refr) && (depth < MAXDEPTH);

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

Vec * MAXDEPTH);

Vec * survive = SurvivalProbability( diffuse,
Vec * estimation - doing it properly, close
if;
Vec * radiance = SampleLight( &rand, E, M, Allg
Vec * e.x + radiance.y + radiance.z) > 0) && (nt
Vec * w = true;
Vec * at brdfPdf = EvaluateDiffuse( L, N ) * P
Vec * st3 factor = diffuse * INVPI;
Vec * at weight = Mis2( directPdf, brdfPdf );
Vec * at cosThetaOut = dot( N, L );
Vec * E * ((weight * cosThetaOut) / directPdf) * (radiance
Vec * random walk - done properly, closely following
Vec * survive);

Vec * at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
Vec * survive;
Vec * pdf;
Vec * n = E * brdf * (dot( N, R ) / pdf);
Vec * n - true;
    
```



StarCraft II



Considerations

Custom timers: what to measure?

- Time spent in your code
- ‘Wall clock time’
- Cycles

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

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:

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

```

...
    & (depth < MAXDEPTH)
...
    c = inside / 2;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
}

...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + I) - R0;
    Tr) R = (D * nnt - N * (dot
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, I, M);
    e.x + radiance.y + radiance.z) > 0) && (depth
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * num
    t3 factor = diffuse * INVPT;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radi
...
    random walk - done properly, closely following
    rive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



Considerations

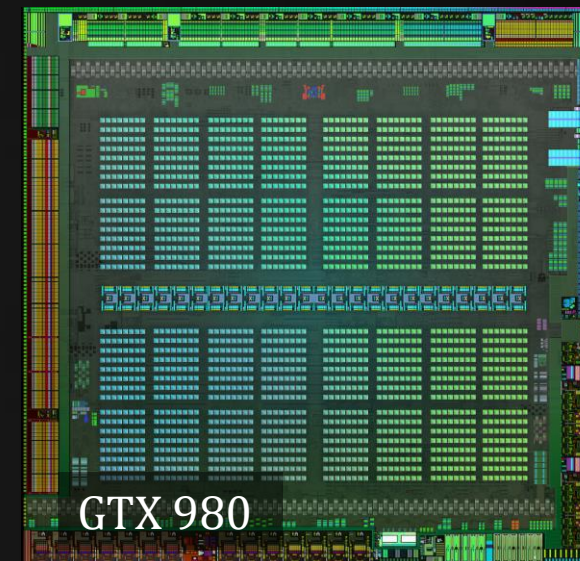
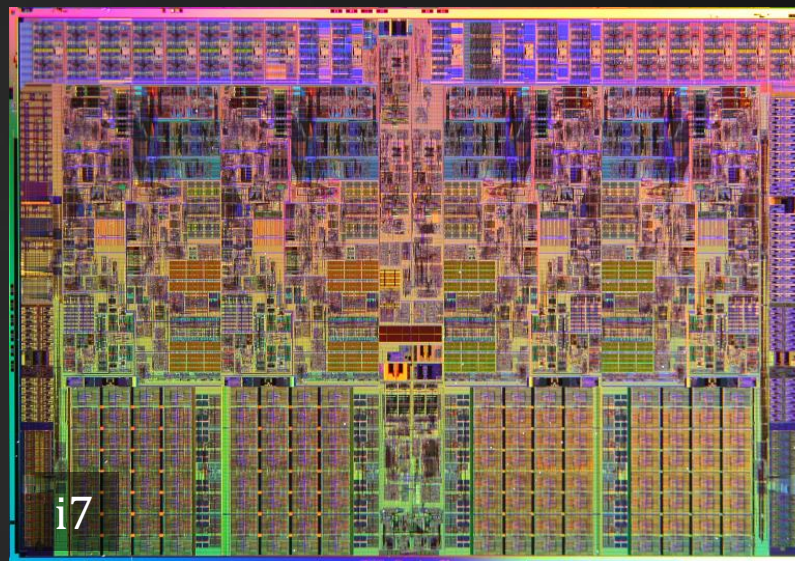
What if the CPU isn't the problem?

A modern system consists of:

- A multicore CPU;
- A many-core GPU.

If one of them is idling, your application isn't running at maximum performance.

Move work, even if the other processor can't do it as efficient.



```

...
k (depth * MAXDEPTH);
...
inside / ...
nt = nt / ...
os2t = 1.0f / ...
D, N );
...
)
...
at a = nt - nc, b = nt - ...
at Tr = 1 - (R0 + I1 + ...
Tr) R = (D * nnt - N * ...
...
E * diffuse;
= true;
...
efl + refr)) && (depth < MAXDEPTH);
...
D, N );
efl * E * diffuse;
= true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse, ...
estimation - doing it properly, ...
if;
radiance = SampleLight( &rand, E, M, ...
e.x + radiance.y + radiance.z) * ...
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * ...
st3 factor = diffuse * INWPT;
at weight = Mix2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * ...
...
random walk - done properly, closely following ...
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```



Considerations

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.

```

...
    & (depth < MAXDEPTH)
...
    c = inside / 2;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + 1) - R0;
    Tr) R = (D * nnt - N * (ab
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly,
if;
    radiance = Samplelight( &rand, E, M, Allg
    e.x + radiance.y + radiance.z) > 0) && (nt
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    t3 factor = diffuse * INVPT;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
    t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, spot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



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.

```

...
    & (depth < MAXDEPTH)
...
    c = inside ? 0 : 0.5;
    nt = nt / nc;
    cos2t = 1.0f - cos2t;
    D, N );
}
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (RR + LL + BB);
    Tr) R = (D * nnt - N * (bb
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, close
if;
radiance = SampleLight( &rand, E, M, &light
e.x + radiance.y + radiance.z) > 0) && (nt
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N) * survive;
    t3 factor = diffuse * INVPT;
    at weight = Mts2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
    t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pot
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```



