rics & (depth < Notos

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

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

= * diffuse = true;

efl + refr)) && (depth < MAXDEP

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

AXDEPTH)

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

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

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

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

/INFOMOV/ Optimization & Vectorization

J. Bikker - Sep-Nov 2019 - Lecture 12: "Cache-Oblivious"

Welcome!



nics & (depth < Notice⊺

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

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

= * diffuse; = true;

-:fl + 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)
2.x + radiance.y + radiance.z) > 0) & closed

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

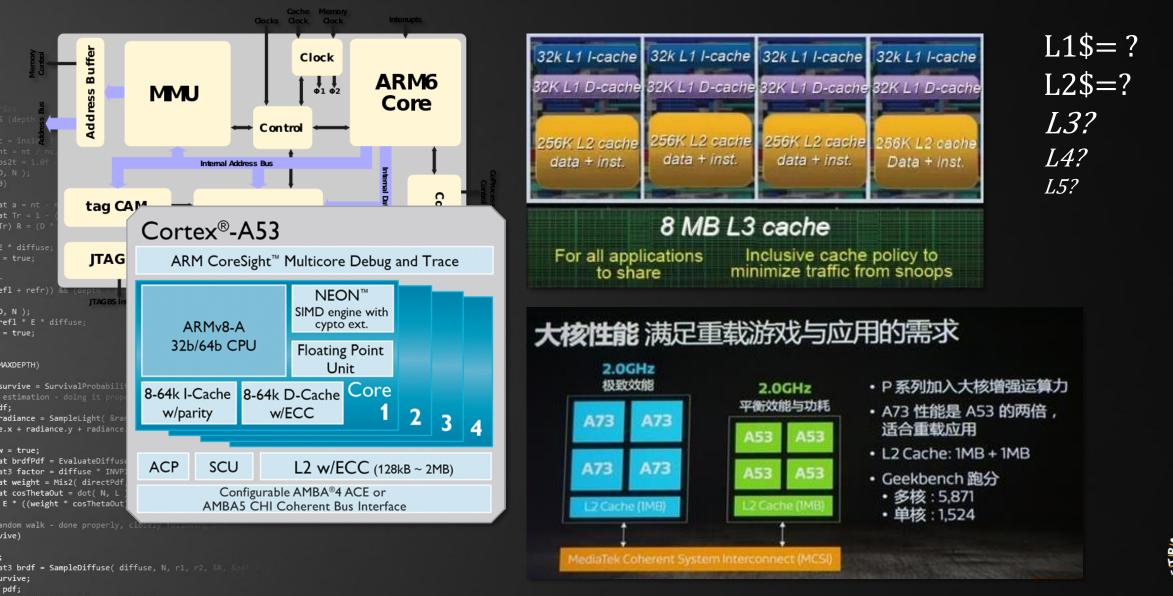
andom walk - done properly, closely following Sec. /ive)

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

Today's Agenda:

- Introduction
- The Idealized Cache Model
- Divide and Conquer
- Sorting
- Digest





f:

nics & (depth < MANDERT

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

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPT

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)

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

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely foll /ive)

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

Dealing with Different Architectures

Modern hardware is not uniform

- Number of cache levels
- Cache sizes and cache line size
- Associativity, replacement strategy, bandwidth, latency...

Programs should ideally run for different parameters

- Works if we determine the parameters at runtime
- (or perhaps a few important ones)
- Or we just ignore the details. *(i.e., what we do in practice)*

Programs are executed on unpredictable configurations

- Generic portable software libraries
 - Code running in the browser

Dealing with Different Architectures

Modern hardware is not uniform

Number of cache levels
 Cache sizes and cache line size
 Associativity, replacement strategy, bandwidth, late diffuses

 $\Pr_{\text{fight}} = \Pr_{\text{fight}} \operatorname{should}_{\text{fight}} \operatorname{should}_{\text{fight}}$

Works if we determine the parameters at runtime

■ (or perhaps a few important ones)

surve or we just ignore the details. (i.e., what we do in estimation - doing it properly and the details.

weight = Mis2(diflettPdft brdf1) costdetheLL pollable SoftWare libraries * ((weight * costhetaOut) / directPdf)

Code running in the browser

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



OBLIVIOUS

Nice Day for a walk



Introduction

a **cache-oblivious algorithm** is an algorithm designed to *take advantage of a CPU cache without having the size of the cache (or the length of the cache lines, etc.) as an explicit parameter.*

An **optimal cache-oblivious algorithm** is a cache-oblivious algorithm that *uses the cache optimally.*

A cache-oblivious algorithm is effective on *all levels of the memory hierarchy, simultaneously.*

Can we get the benefits of cache-aware code without knowing the details of the cache?



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

efl + refr)) && (depth

survive = SurvivalProbability(dif

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

refl * E * diffuse;

), N);

AXDEPTH)

v = true:

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

, H33 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pd urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

People

tics &j(depth < MAXDE

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

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

= * diffuse; = true;

efl + refr)) && (depth < NACED

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

AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly. 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 Small /ive)

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

Cache-Oblivious Algorithms. Harald Prokop, Master thesis, MIT, 1999. Cache-Oblivious Algorithms. Frigo, Leierson, Prokop, Ramachandran, 1999. Cache Oblivious Distribution Sweeping. Brodal, Stølting. Lecture notes, 2002. Cache-Oblivious Algorithms and Data Structures. Brodal, SWAT 2004.





survive = SurvivalProbability(diffuence estimation - doing it properly, closed ff; radiance = SampleLight(&rand, I, &L, &light)

e.x + radiance.y + radiance.z) > 0) 28 (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)

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;

Cache-oblivious data structures and algorithms:

Optimizing an application *without knowing hardware details.*

Hi, I don't care. Thanks.



nics & (depth < Notice⊺

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

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

= * diffuse; = true;

-:fl + 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)
2.x + radiance.y + radiance.z) > 0) & closed

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

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

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

Today's Agenda:

- Introduction
- The Idealized Cache Model
- Divide and Conquer
- Sorting
- Digest



Cache Model

T

at Tr = 1

), N);

= true;

(AXDEPTH)

v = true:

lf:

refl * E * diffuse;

survive = SurvivalProbability

radiance = SampleLight(&rand,

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

at brdfPdf = EvaluateDiffuse(| at3 factor = diffuse * INVPI;

at cosThetaOut = dot(N, L);

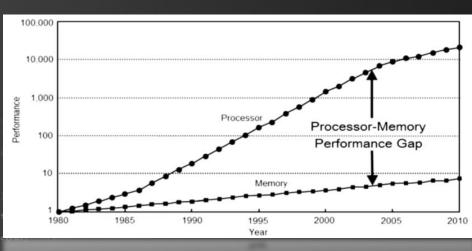
at weight = Mis2(directPdf, brdf)

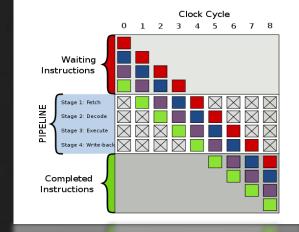
E * ((weight * cosThetaOut) / di

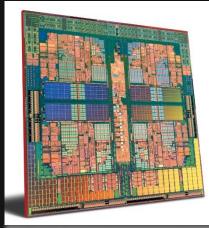
Previously in INFOMOV:

Estimating algorithm cost:

- 1. Algorithmic Complexity : O(N), $O(N^2)$, $O(N \log N)$, ...
- 2. Cyclomatic Complexity* (or: Conditional Complexity)
- 3. Amdahl's Law / Work-Span Model
- 4. Cache Effectiveness









andom walk - done properly, closely follo /ive)

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

Cache Model

The External-Memory Model

Assumptions*:

- Transfers happen in blocks of B elements.
- The cache stores M elements, in M/B blocks.
- The block count is substantial.
- A cache miss results in transfer of 1 block. If the cache was full, a second transfer occurs (eviction).



efl + refr)) && (dept)

AXDEPTH)

at Tr = 1 -

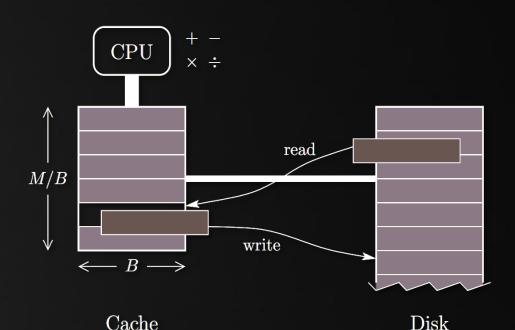
survive = SurvivalProbability(diffuse estimation - doing it properly, close if; radiance = SampleLight(&rand, I, &L, / e.x + radiance.y + radiance.z) > 0) &&

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

andom walk - done properly, closely foll /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1 urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true: The complexity of an algorithm is (solely) measured as the number of cache misses.

*: Cache-Oblivious Algorithms. Prokop, 1999. MIT Master Thesis. For a digest, read: <u>http://erikdemaine.org/papers/BRICS2002/paper.pdf</u>





Cache Model

at a = nt

), N);

= true;

(AXDEPTH)

v = true;

/ive)

urvive;

sion = true:

if;

efl + refr)) && (depth)

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);

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

E * ((weight * cosThetaOut) / directPdf) andom walk - done properly, closely fol

refl * E * diffuse;

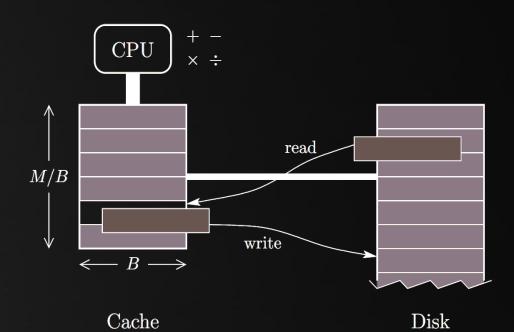
The Cache-Oblivious Model

Assumptions*:

- Transfers happen in blocks of D clements.
- <u>The cache stores M elements, in M/B blocks.</u>
- The block count is substantial.
- A cache miss results in transfer of 1 block. If the cache was full, a second transfer occurs (eviction).
- The cache is fully associative.
- The replacement policy is optimal.



For a digest, read: <u>http://erikdemaine.org/papers/BRICS2002/paper.pdf</u>





at3 brdf = SampleDiffuse(diffuse, N, r1,

Cache Model

The Cache-Oblivious Model

Example:

at a = ni at Tr = 1 - (R0

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

(AXDEPTH)

survive = SurvivalProbability(di radiance = SampleLight(&rand, .x + radiance.v + radiance.z)

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

andom walk - done properly, closely foll /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 1 urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true

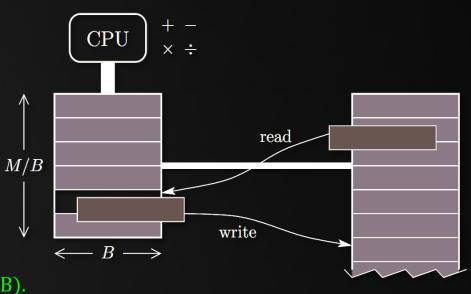
Calculating the sum of an array of *N* integers has an algorithmic complexity O(N).

In the external-memory model, the complexity is: [N/B] (i.e.: ceil(M/B).

(note: this assumes alignment, which requires knowledge about B).

The cache-oblivious algorithm cannot assume specific values for M or B. We therefore get: [N/B]+1.

(note: one extra block, because of alignment) (note: we do use B in the analysis, but not in the algorithm.) (note: the complexity is identical to [N/B] for $N = \infty$.)





Disk



Cache Model

nics & (depth < Motos

: = inside ? 1 1 1 1 ht = nt / nc, ddn - 1 552t = 1.0f - nnt - nh 2, N); 8)

at a = nt - nc, b = nt 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 close )
f;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (do
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 foll
```

```
/ive)
```

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

The Cache-Oblivious Model

And now for an actually useful example...

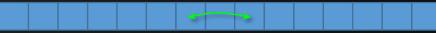
void Reverse(int* values, int N)
{
 // ...?

- Easy to do with a temporary array.
- Cache-oblivious algorithm*:

for(int i = 0; i < N/2; i++) { swap(values[i], values[N-1-i]);</pre>

(note: requires as many block access as a single scan.)

*: Programming Pearls, 2nd edition. Jon Bentley, 2000.





nics & (depth < Modean

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

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

= * diffuse; = true;

-:fl + 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)
2.x + radiance.y + radiance.z) > 0) & closed

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

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

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

Today's Agenda:

- Introduction
- The Idealized Cache Model
- Divide and Conquer
- Sorting
- Digest



Tree

nics & (depth < MADS≣1

: = inside ? 1 : . . . ht = nt / nc, ddn hs2t = 1.0f - nnt h, N); a)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (300

= * diffuse = true;

efl + refr)) && (depth < MAXDE

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

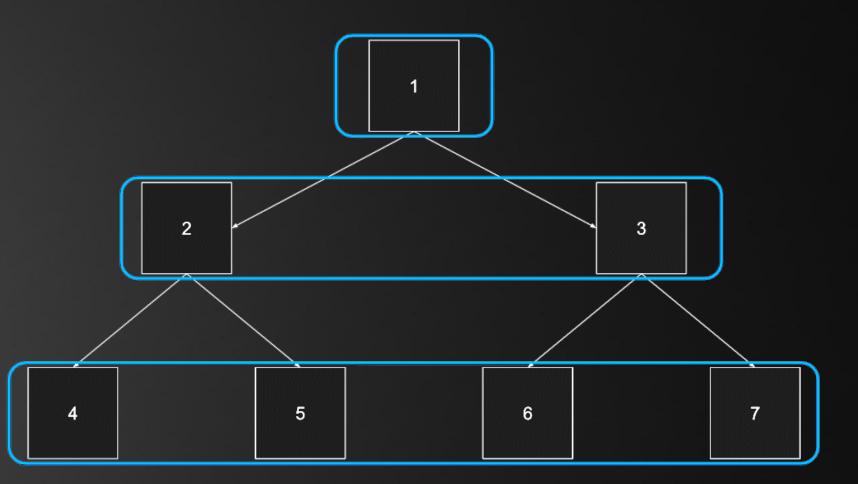
AXDEPTH)

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

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) * (radi

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

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





Tree

nics & (depth < Monosa

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

= * diffuse = true:

efl + refr)) && (depth < MAXDEPT

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

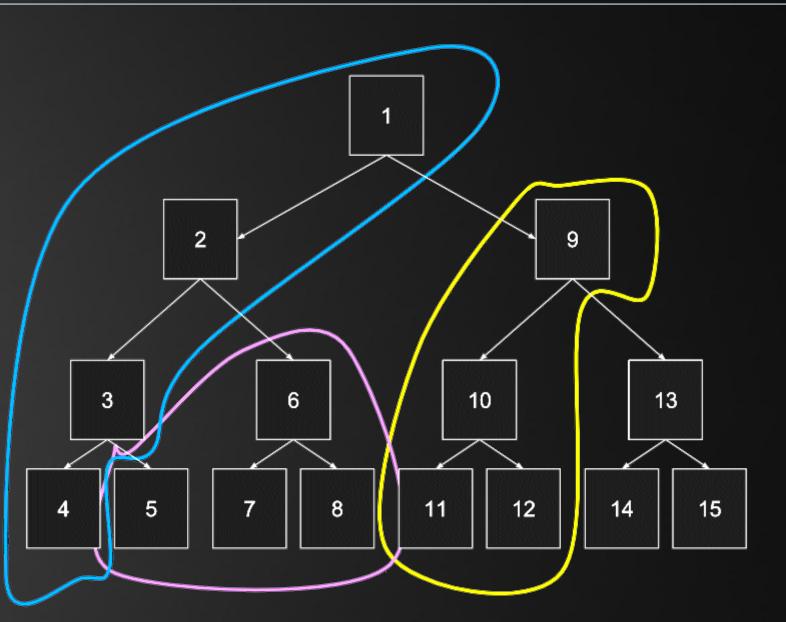
AXDEPTH)

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

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)

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

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





Tree

nics & (depth < NOCCO⊓

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

= * diffuse; = true;

. **.**

), N); •efl * E * diffus = true;

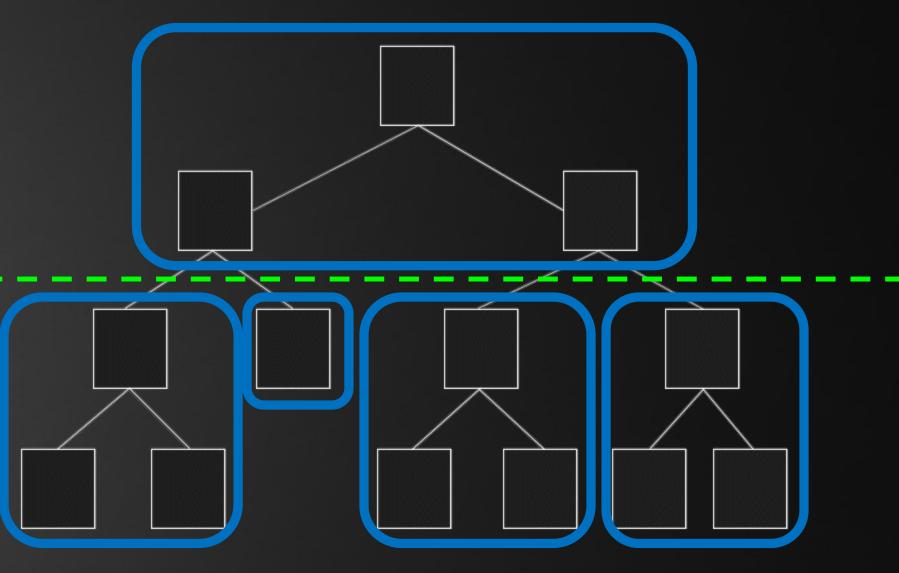
AXDEPTH)

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

v = 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 30 /ive)

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





Tree

nics & (depth < NACCSS

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

at a = nt - nc, b = NC = at Tr = 1 - (RO + (1 - RO Tr) R = (D [#] nnt - N ⁼ (dd)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEDIDI

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

AXDEPTH)

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

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

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

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

log

log

Comparisons

Breadth-first tree:

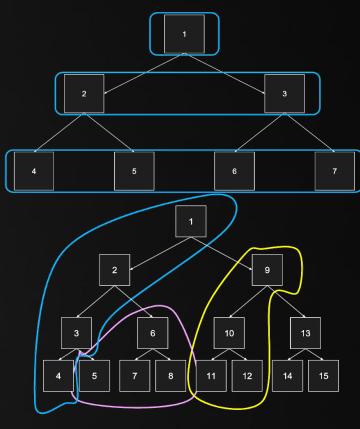
Going down in the tree, every step will access a different block. Expected accesses is $\log_2 N$. (e.g. 16 for N=65536)

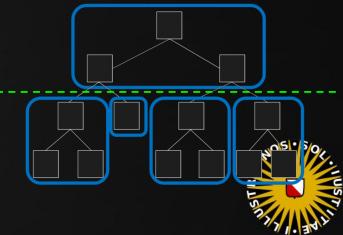
Depth-first tree:

Although left branches are efficient, every right branch requires a different block.

Cache-oblivious layout:

$$\frac{N}{R} = \log_B N.$$
 (e.g. 4 for N=65536, B=16)





Tree

ics & (depth < ≫0000

: = inside ? 1 : 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 = (d

= * diffuse; = true;

• efl + refr)) && (depth < MAXO

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

AXDEPTH)

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

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

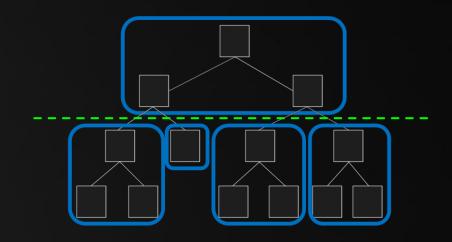
andom walk - done properly, closely following Sec. /ive)

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

The Cache-Oblivious Tree

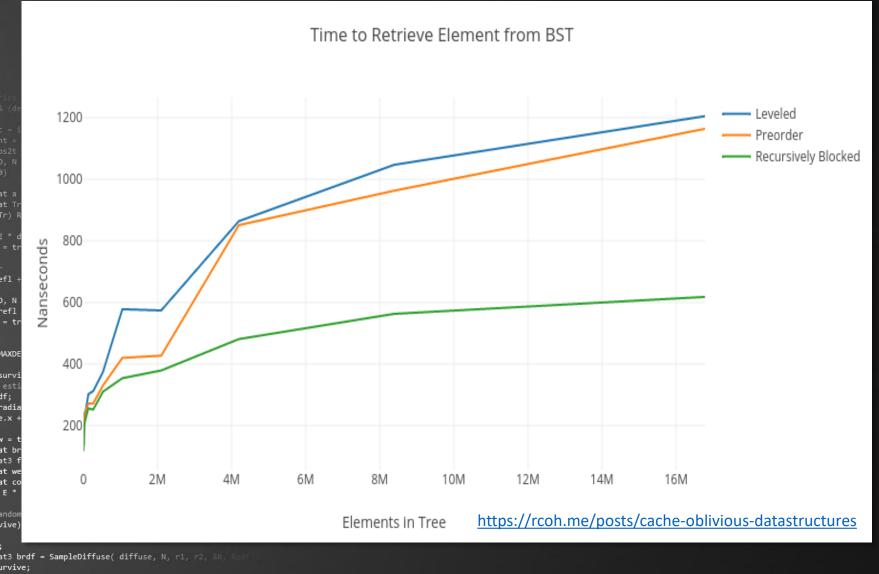
Algorithm:

- 1. Split the tree vertically, at level $\frac{1}{2}\log(N)$.
 - (where N is the number of leaf nodes)
- 2. The top now contains \sqrt{N} elements.
- 3. Produce five subtrees and process these recursively.





Tree



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

nics & (depth < Modean

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

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

= * diffuse; = true;

-:fl + 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)
2.x + radiance.y + radiance.z) > 0) & closed

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

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

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

Today's Agenda:

- Introduction
- The Idealized Cache Model
- Divide and Conquer
- Sorting
- Digest



Sort

), N); refl * E * diffuse;

AXDEPTH)

v = true;

MergeSort



urvive;

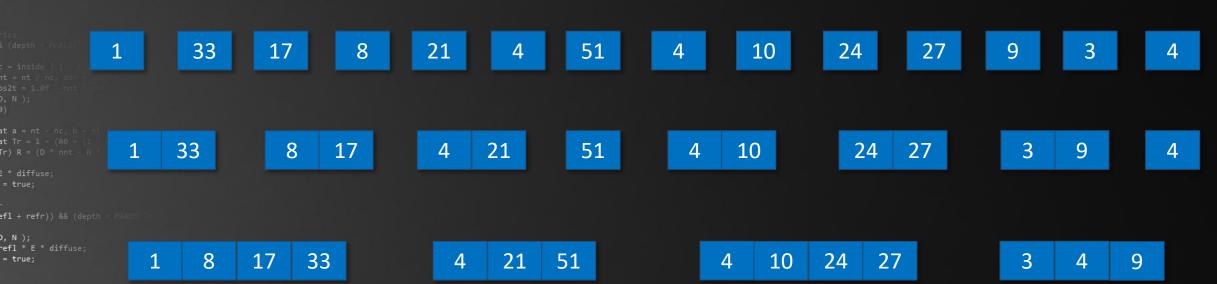
/ive)

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

US/7/2

MergeSort

Sort



(AXDEPTH)

survive = SurvivalProbability(diffue)
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) 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, 8pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

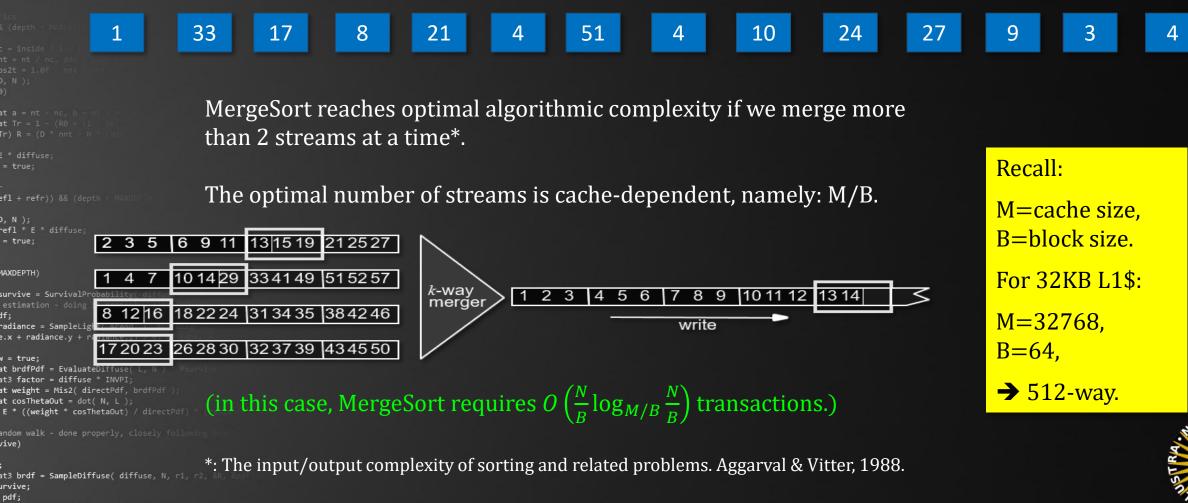
Merging two buffers A[] and B[] to C[]:

*C = *A < *B ? *A++ : *B++



MergeSort

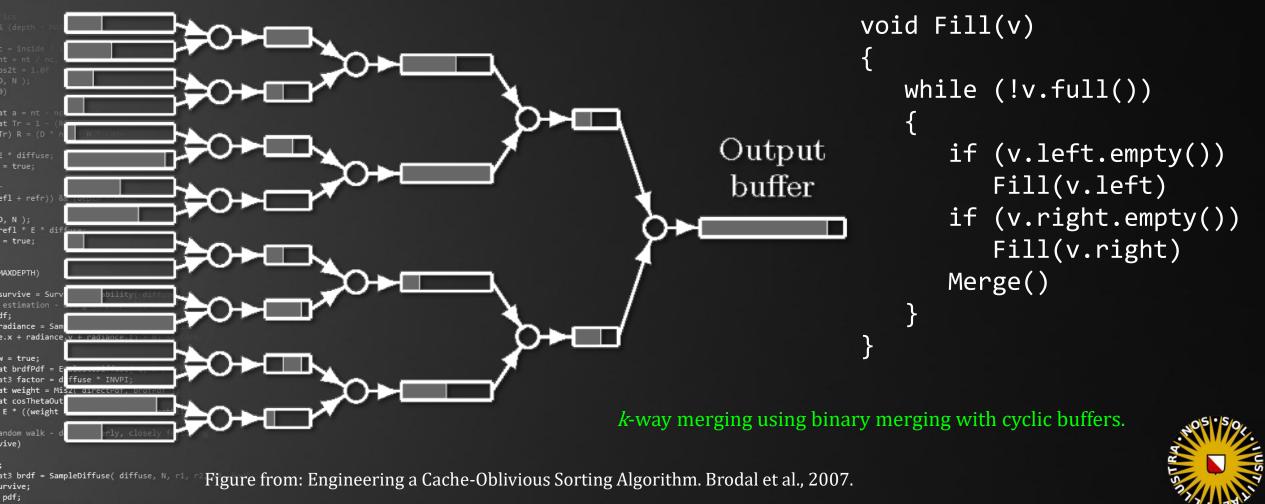
Sort



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

Sort

FunnelSort (the "lazy" variety)



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

Sort

nics & (depth < Notes

: = inside ? 1 : 1 .0 ht = nt / nc, ddn ps2t = 1.0f - nnt 2, N); 3)

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDED

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

AXDEPTH)

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

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 some /ive)

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

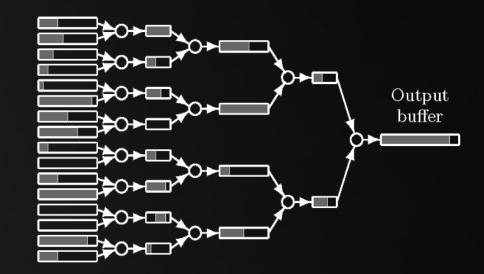
FunnelSort (the "lazy" variety)

How:

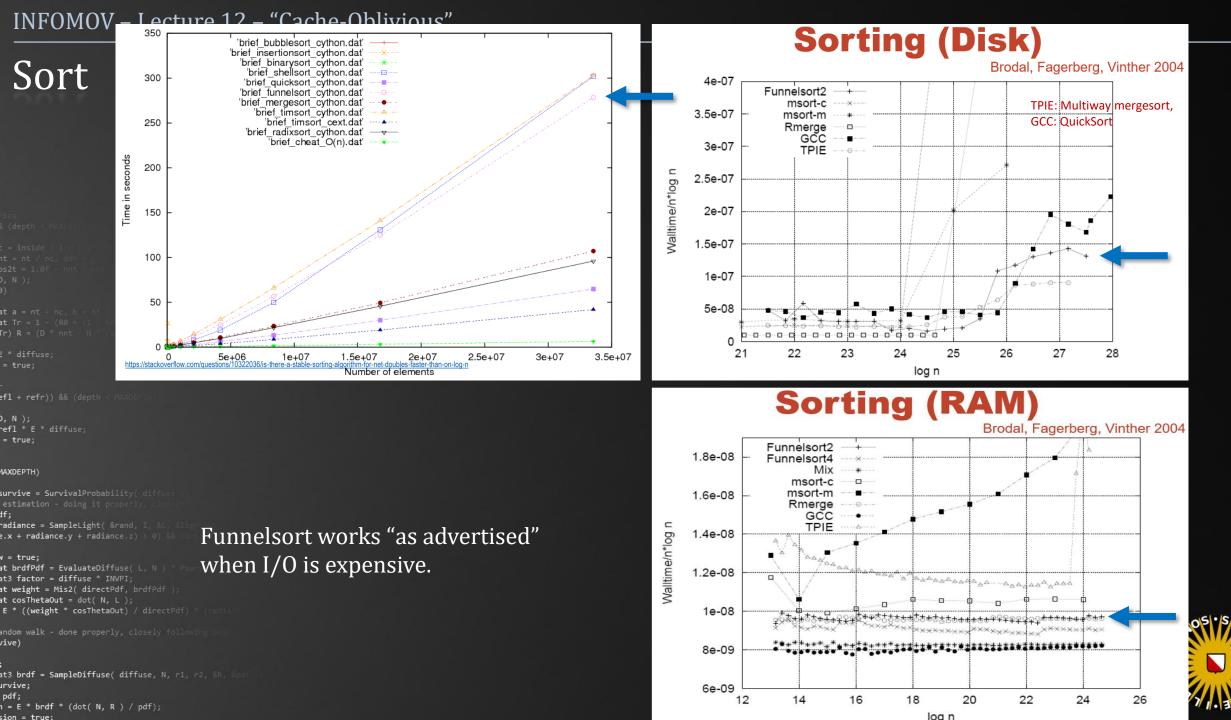
• Split the input into $N^{\frac{1}{3}}$ ("cube root") sets of $N^{\frac{2}{3}}$ elements.

(so: 1000 becomes 10 sets of 100; 512 becomes 8 sets of 64, 8 becomes 2 sets of 4.)

- Recurse.
- Merge the $N^{\frac{1}{3}}$ sorted sequences using an $k = N^{\frac{1}{3}}$ merger.
 - The *k*-merger suspends work whenever there is sufficient output.







nics & (depth < Modean

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

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

= * diffuse; = true;

-:fl + 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)
2.x + radiance.y + radiance.z) > 0) & closed

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

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

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

Today's Agenda:

- Introduction
- The Idealized Cache Model
- Divide and Conquer
- Sorting
- Digest



ics & (depth < Notes

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

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEDII

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)

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 Source /ive)

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

Cache-Oblivious Concepts

Data structures:

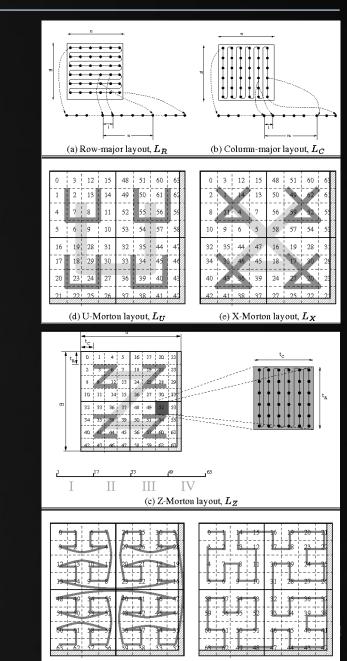
Linear array – operated on using a scan.
 (works for the most basic cases, but also Bentley's Reverse)

2. Recursive subdivision

(not discussed in this lecture, but covered before)

3. Cache-Oblivious tree layout

(I wish I knew about that one before)



nics & (depth < Moccor

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

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

= * diffuse; = true;

• efl + refr)) && (depth < MAXDE

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

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly
f;
vadiance = SampleLight(&rand, I, &L, &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 fo. /ive)

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

Cache-Oblivious Concepts

Algorithms:

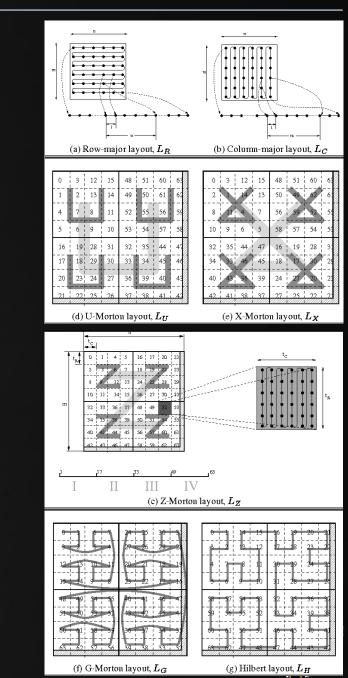
- Often trivially following from data structures.
- Sorting only fast for expensive I/O.

Note the overlap with:

- Data oriented design
- Data-parallel algorithms
- Streaming algorithms

(although there are differences too)

And appreciate the attention to memory cost.



nics & (depth < MADDar

: = inside ? 1 0 0 0 ht = nt / nc, ddn 0 0 ps2t = 1.0f - nnt 0 2, N); 3)

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

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPTION

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

AXDEPTH)

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

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 Soci /ive)

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

Cache-Oblivious Concepts

Original question:

Can we get the benefits of cache-aware code without knowing the details of the cache?

IMHO:

- Yes, to some extend.
- But we were not really taking into account cache size anyway
- Nor the specifics of the eviction policy
 - And it seems silly not to anticipate a reasonable 'B' (e.g. for alignment)





at a = nt

), N);

= true;

(AXDEPTH)

v = true;

if;

efl + refr)) && (depth <

survive = SurvivalProbability(diff

radiance = SampleLight(&rand, I, . e.x + radiance.y + radiance<u>.z) > 0)</u>

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

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

E * ((weight * cosThetaOut) / directPdf)

refl * E * diffuse;

Cache-Oblivious Concepts

Further reading

"& Cache-Oblivious Algorithms (Updated)" <u>qstuff.blogspot.com/2010/06/cache-oblivious-algorithms.html</u>

Cache-Oblivious R-Trees: www.win.tue.nl/~mdberg/Papers/co-rtree.pdf

Cache-Oblivious hashing: https://www.itu.dk/people/pagh/papers/cohash.pdf

Cache-Oblivious FFT:

https://www.csd.uwo.ca/~moreno/CS433-CS9624/Resources/Implementing_FFTs_in_Practice.pdf

Cache-Oblivious mesh layouts (and other graphics-related CO topics): http://gamma.cs.unc.edu/COL/

andom walk - done properly, closely following Sastin /ive) ; ;;

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

33



nics & (depth < Modean

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

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

= * diffuse; = true;

-:fl + 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)
2.x + radiance.y + radiance.z) > 0) & closed

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

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

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

Today's Agenda:

- Introduction
- The Idealized Cache Model
- Divide and Conquer
- Sorting
- Digest





at Tr Tr) R E * di

= true;

efl + ref

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

AXDEPTH)

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

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

THE END IS NEAR

w = true; at brdfPdf = EvaluateDiffuse(<u>L, N)</u>

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

/INFOMOV/

END of "Low Level"

next lecture: "Snippets & Multi-Threading"

