

```
ics
(depth < MAXDEPTH)
inside ? 1 : 0;
nt = nt / nc; ddn = ddn * ddn;
s2t = 1.0f - nnt * nnt;
D, N );
)
at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * ddn);
Tr) R = (D * nnt - N * (ddn *
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH);
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light;
e.x + radiance.y + radiance.z) > 0) && (survive);
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) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

/INFOMOV/

Optimization & Vectorization

J. Bikker - April - June 2024 - Lecture 6: "Caching (2)"

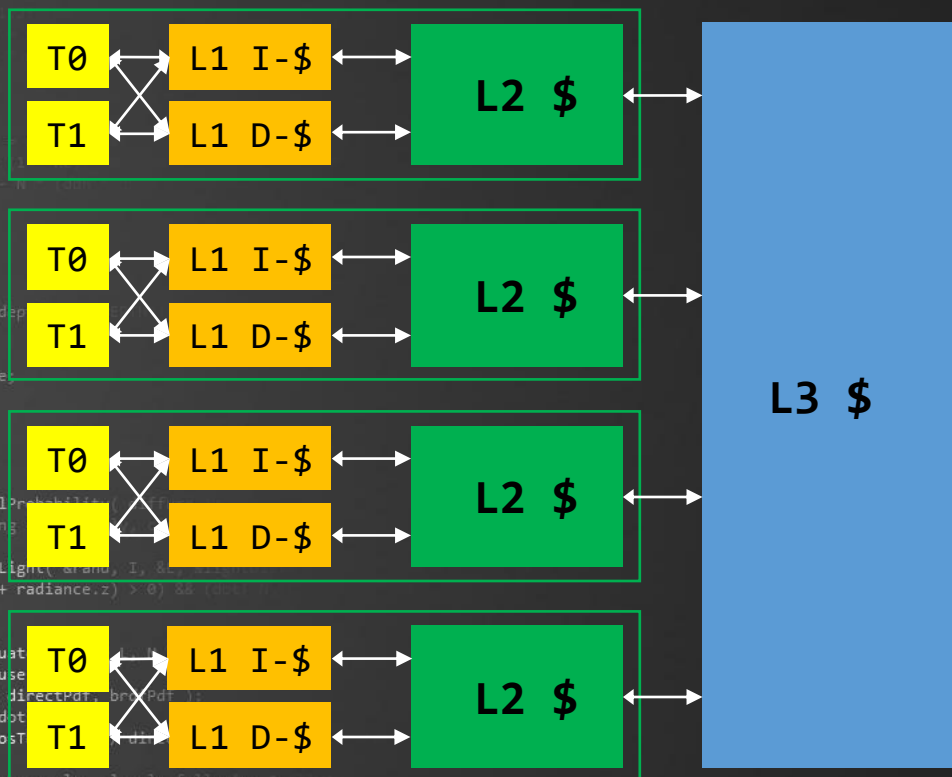
Welcome!



Recap

```

...
    &(depth < MAXDEPTH)
...
    int = nt / nc;
    float s2t = 1.0f / nnt;
    int D, N;
...
    at a = nt - nc;
    at Tr = 1 - (R0 +
    Tr) R = (D * nnt
...
    E * diffuse;
    = true;
...
    refl + refr) && (dep
...
    D, N);
    refl * E * diffuse;
    = true;
...
    MAXDEPTH)
...
    survive = Survival;
    estimation - doing
    if;
    radiance = SampleLight;
    e.x + radiance.y + radiance.z) > 0) && (survive)
...
    w = true;
    at brdfPdf = Evaluat
    at3 factor = diffuse
    at weight = Mis2( directPdf, brdfPdf);
    at cosThetaOut = dot
    E * ((weight * cosT
...
    random walk - done properly, closely following
    (survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```

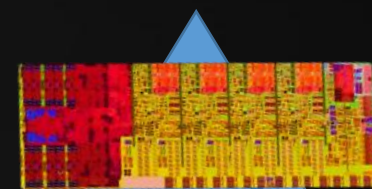


32KB I / 32KB D per core

256KB per core

8MB

x GB



registers:
0 cycles

level 1 cache: 4 cycles

level 2 cache: 11 cycles

level 3 cache: 39 cycles

RAM: 100+ cycles



Recap

Three types of cache

- Fully associative
- Direct mapped
- N-set associative

In an *N-set associative cache*, each memory address can be stored in *N* slots.

Example:

- 32KB, 8-way set-associative, 64 bytes per cache line: 64 sets of 512 bytes.

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }

    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (ddn *
    E * diffuse;
    = true;

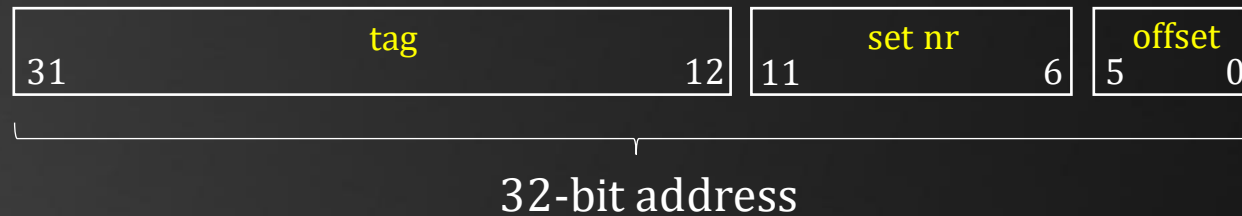
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;

MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (rand
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) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



Recap

32KB, 8-way set-associative, 64 bytes per cache line: 64 sets of 512 bytes



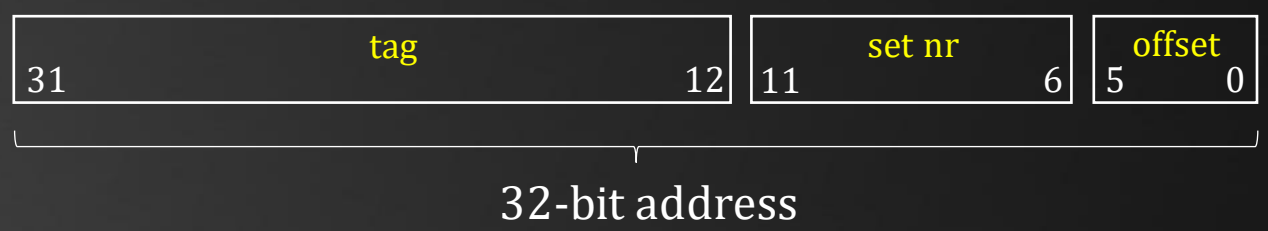
Fetching a byte at address a :

- offset = a & 63
- set = (a >> 6) & 63
- tag = a >> 12, range: 0 .. 2²⁰-1
- return cache[tag][0..7].data[offset]



Recap

*32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes*



Examples:

set: 001000b = 8
offset: 110100b = 52

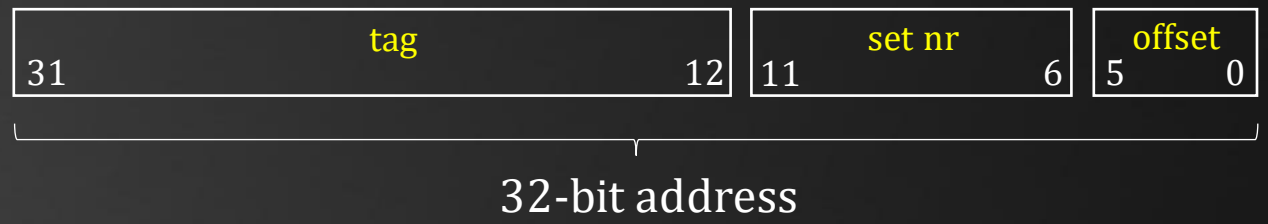
0x00001234	0001	001000	110100
0x00008234	1000	001000	110100
0x00006234	0110	001000	110100
0x0000A234	1010	001000	110100
0x0000A240	1010	001001	000000
0x0000F234	1111	001000	110100

set: 0..63 (6 bit)



Recap

*32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes*

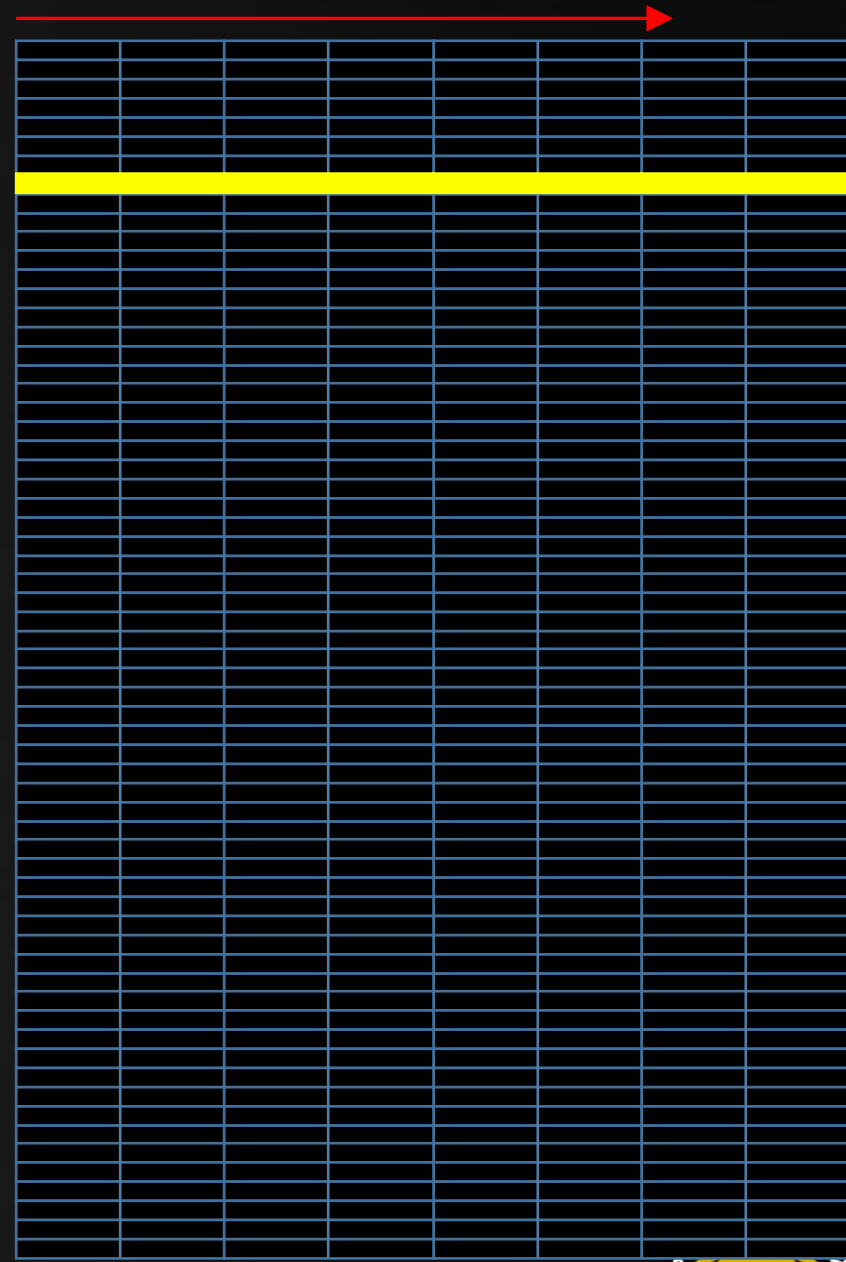


Examples:

0x00001234	0001	001000	110100
0x00008234	1000	001000	110100
0x00006234	0110	001000	110100
0x0000A234	1010	001000	110100
0x0000A240	1010	001001	000000
0x0000F234	1111	001000	110100

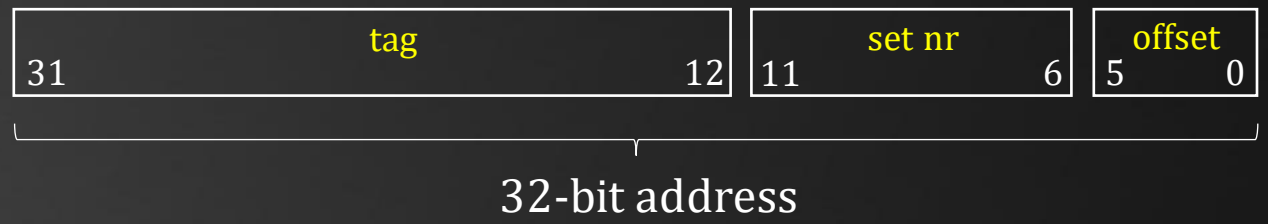
set: 0..63 (6 bit)

slot (0..7)



Recap

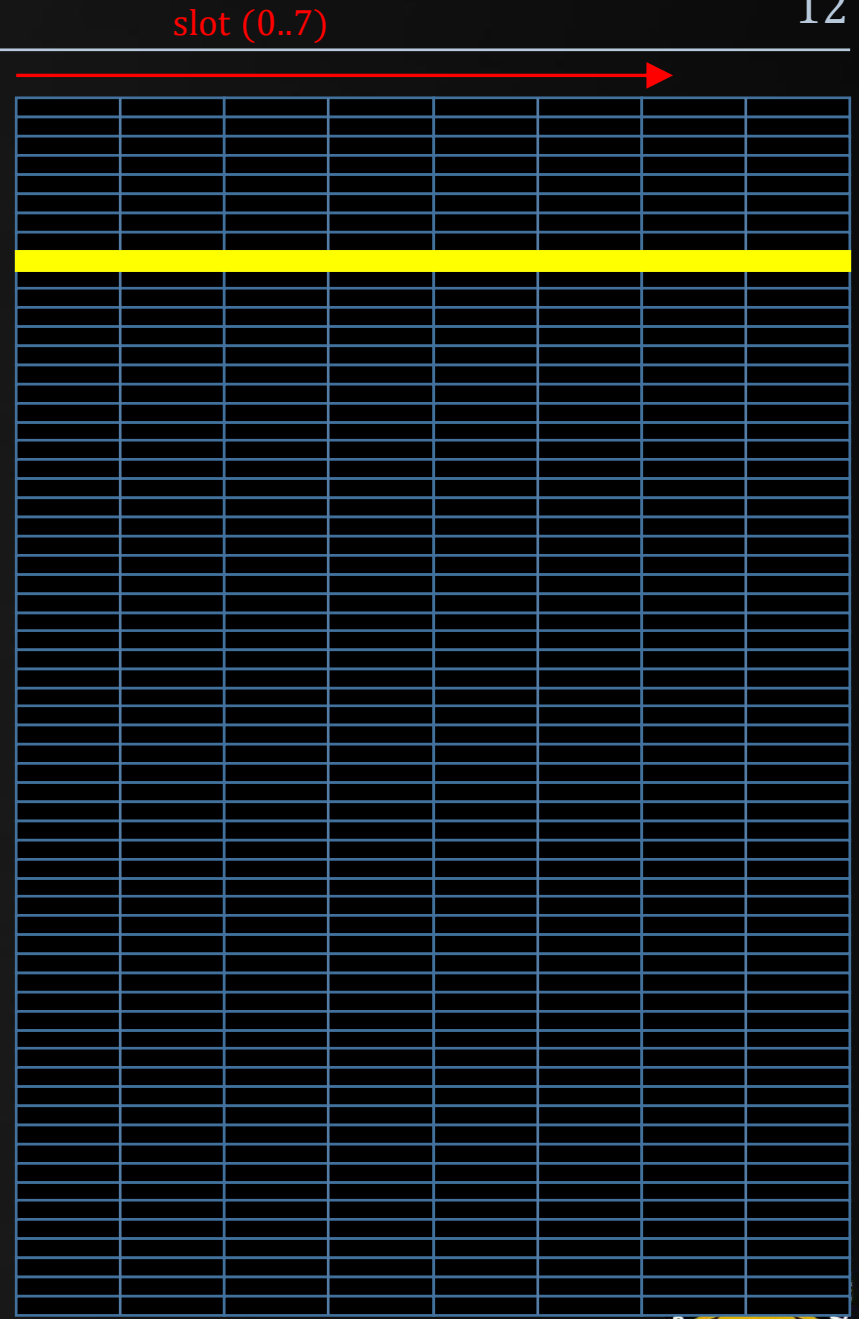
*32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes*



Examples:

0x00001234	0001	001000	110100
0x00008234	1000	001000	110100
0x00006234	0110	001000	110100
0x0000A234	1010	001000	110100
0x0000A240	1010	001001	000000
0x0000F234	1111	001000	110100

set: 0..63 (6 bit)




```
ics
& (depth < MAXDEPTH)
{
    if ( ! inside ) return 0;
    Vec nt = nc, ndn = n;
    Vec ns2t = 1.0f - nnt * nnt;
    Vec D, N );
    Vec a = nt - nc, b = nt * n;
    float Tr = 1 - (R0 + (1 - R0) * r);
    Vec R = (D * nnt - N * (Dn > 0 ? 1 : -1));
    Vec E * diffuse;
    Vec refl;
    Vec refl + refr)) && (depth < MAXDEPTH)
    Vec D, N );
    Vec refl * E * diffuse;
    Vec = true;
    Vec MAXDEPTH)
    Vec survive = SurvivalProbability( diffuse, r);
    Vec estimation - doing it properly, closely following
    Vec if;
    Vec radiance = SampleLight( &rand, I, &L, &light);
    Vec e.x + radiance.y + radiance.z) > 0) && (depth <
    Vec w = true;
    Vec at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    Vec at3 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 ;
    Vec at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    Vec survive;
    Vec pdf;
    Vec n = E * brdf * (dot( N, R ) / pdf);
    Vec sion = true;

```

Today's Agenda:

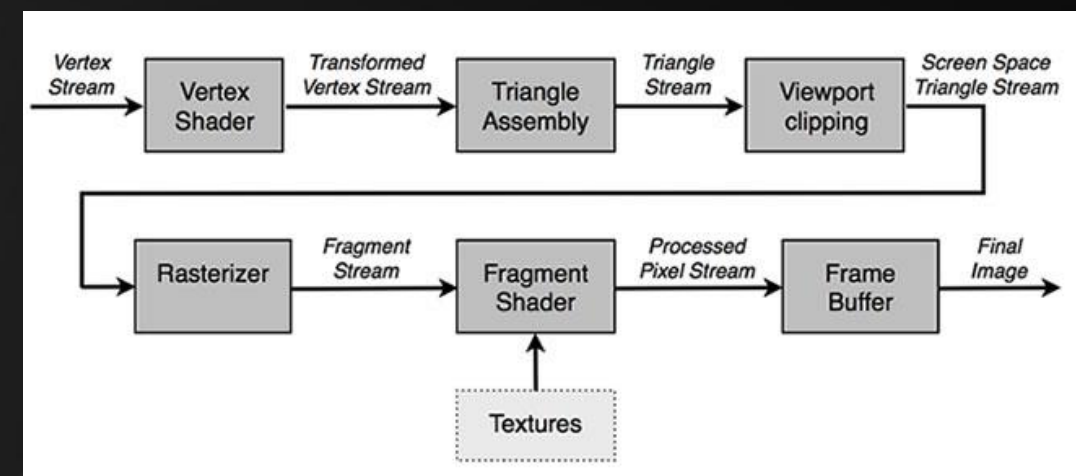
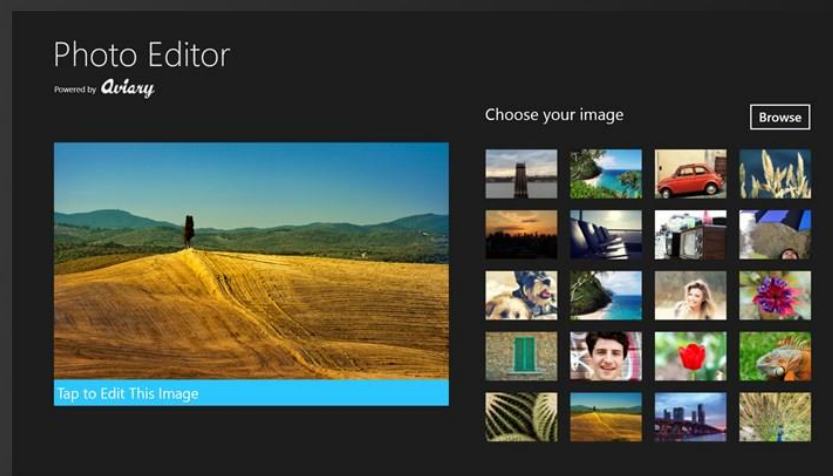
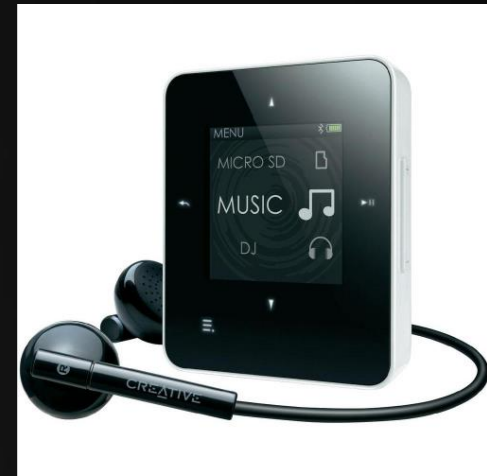
- Recap
- Data Locality
- Alignment
- False Sharing
- Experiments
- A Handy Guide *(to Pleasing the Cache)*



Data Locality

Why do Caches Work?

1. Because we tend to reuse data.
2. Because we tend to work on a small subset of our data.
3. Because we tend to operate on data in patterns.



Data Locality

Reusing data

- Very short term: variable ‘i’ being used intensively in a loop → register
- Short term: lookup table for square roots being used on every input element → L1 cache
- Mid-term: particles being updated every frame → L2, L3 cache
- Long term: sound effect being played ~ once a minute → RAM
- Very long term: playing the same game disk every night → disk

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        ps2t = 1.0f + nnt * ddn;
        D, N );
    }

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

E * diffuse;
= true;

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

MAXDEPTH)

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

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) * (radiance

random walk - done properly, closely following
ive)

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

```



Data Locality

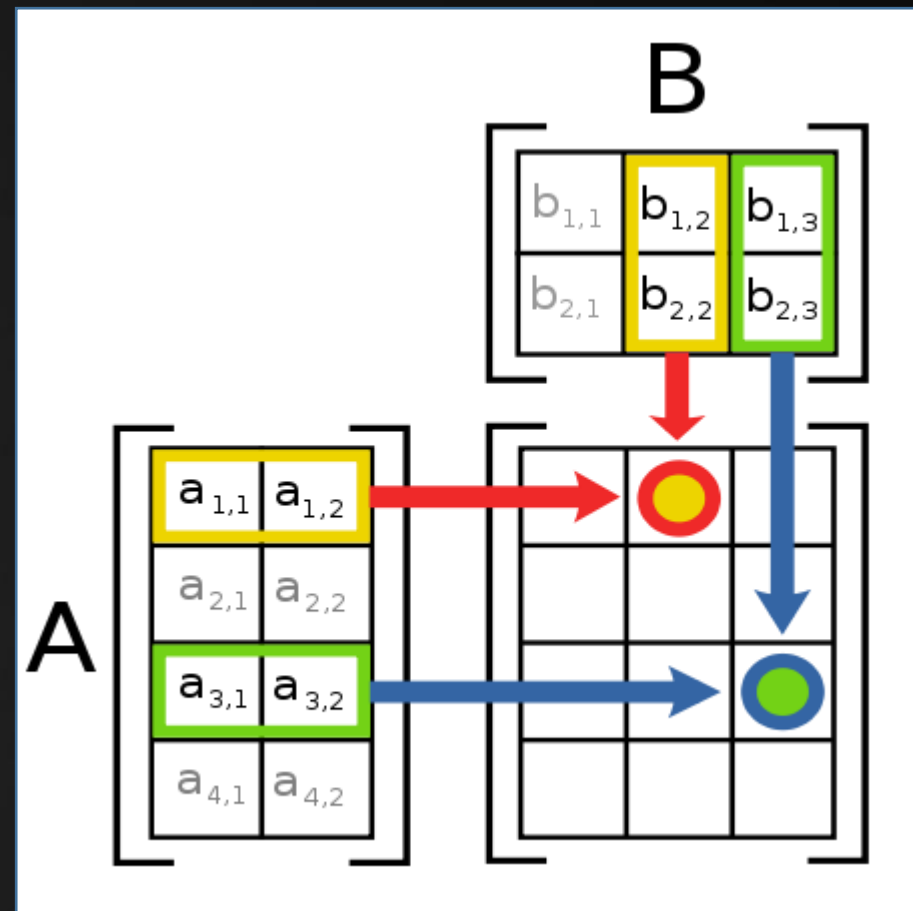
Reusing data

Ideal pattern:

- load data sequentially.

Typical pattern:

- *whatever the algorithm dictates.*



```

ics
& (depth < MAXDEPTH)
    c = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddt = ddt * nc;
    ps2t = 1.0f - nnt * nnt;
    D, N );
    )
at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (ddt
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &align, &
e.x + radiance.y + radiance.z > 0) && (rand < 1);
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) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



Data Locality

Example: rotozooming

```

ics
& (depth < MAXDEPTH)
c = inside ? 1.0f : 0.0f;
nt = nt / nc; ddn = ddn * ddn;
cos2t = 1.0f - nnt * ddn;
D, N );
0);
at a = nt - nc, b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * c);
Tr) R = (D * nnt - N * (ddn *
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH);
D, N );
refl * E * diffuse;
= true;
MAXDEPTH);
survive = SurvivalProbability( diff
estimation - doing it properly,
df;
radiance = SampleLight( &rand, I,
e.x + radiance.y + radiance.z) > 0.45;
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurface;
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 Saito's
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Spdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



Data Locality

Example: rotozooming

```

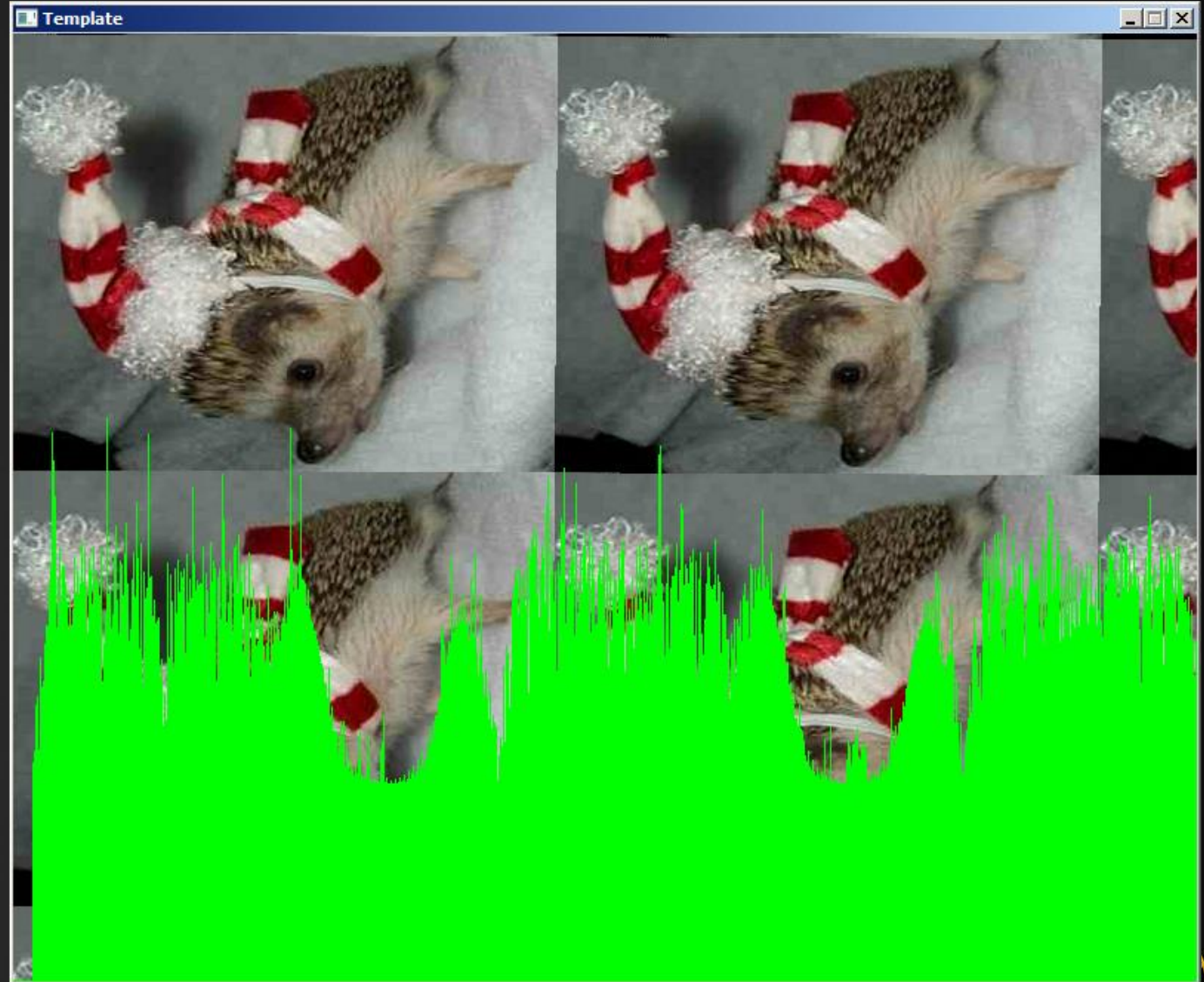
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
}

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

E * diffuse;
= true;

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

MAXDEPTH)
{
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &align );
    e.x + radiance.y + radiance.z > 0) && (depth <
    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) * (radiance
    random walk - done properly, closely following
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
    
```



Data Locality

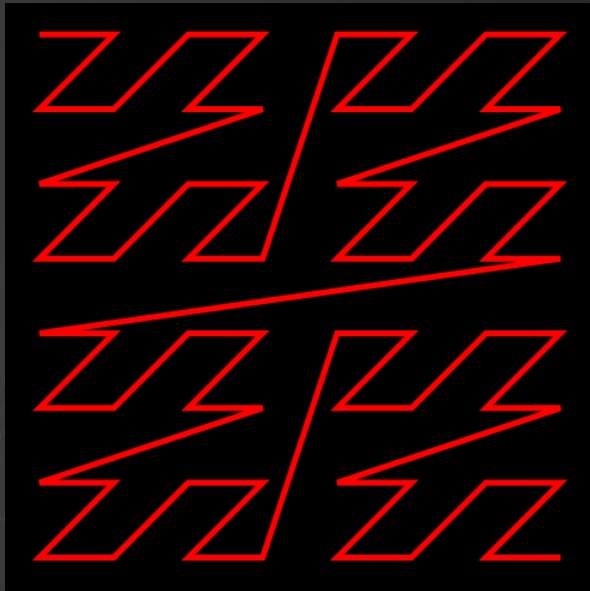
Example: rotozooming

Improving data locality: z-order / Morton curve

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
}

at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * R);
Tr) R = (D * nnt - N * (ddn * cos2t +
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse, r,
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light,
    e.x + radiance.y + radiance.z) > 0) && (cos
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Pdf;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * Pdf;
    random walk - done properly, closely following
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```



Method:

X = 1 1 0 0 0 1 0 1 1 0 1 1 0 1

Y = 1 0 1 1 0 1 1 0 1 0 1 1 1 0

address = 1101101000111001110011111001



Data Locality

Data Locality

Wikipedia:

Temporal Locality – “If at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.”

Spatial Locality – “If a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future.”



* More info: <http://gameprogrammingpatterns.com/data-locality.html>



Data Locality

Data Locality

How do we increase data locality?

Linear access – Sometimes as simple as swapping for loops *

Tiling – Example of working on a small subset of the data at a time.

Streaming – Operate on/with data until done.

Reducing data size – Smaller things are closer together.

How do trees/linked lists/hash tables fit into this?

* For an elaborate example see <https://www.cs.duke.edu/courses/cps104/spring11/lects/19-cache-sw2.pdf>

```

ics
& (depth < MAXDEPTH)
{
    int inside = 1;
    int nt = nt / nc;
    double s2t = 1.0f + nnt * nnt;
    double D, N );
}

at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * R);
Tr) R = (D * nnt - N * (D0 + D1 * s2t));

E * diffuse;
= true;

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

MAXDEPTH)

survive = SurvivalProbability( diffuse, r1, r2, &R, Spdf );
estimation - doing it properly, closely following Section 19.1.
if;
radiance = SampleLight( &rand, I, &L, &align );
e.x + radiance.y + radiance.z) > 0) && (case 1)
{
    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) * (radiance +
    diffuse * factor);
}

random walk - done properly, closely following Section 19.1.
ive)

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



```
ics
(depth < MAXDEPTH)
inside ? 1 : 0;
nt = nt / nc; ddn = u * u + v * v;
s2t = 1.0f - nnt * nnt;
D, N );
);
at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * r);
Tr) R = (D * nnt - N * (ddn > 0) ? 1 : -1);
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse, i);
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (depth <
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) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

- Recap
- Data Locality
- Alignment
- False Sharing
- Experiments
- A Handy Guide *(to Pleasing the Cache)*



Alignment

Cache line size and data alignment

What is wrong with this struct?

```
struct Particle
{
    float x, y, z;
    float vx, vy, vz;
    float mass;
};
// size: 28 bytes
```

Better:

```
struct Particle
{
    float x, y, z;
    float vx, vy, vz;
    float mass, dummy;
};
// size: 32 bytes
```

Note:

As soon as we read *any* field from a particle, the other fields are guaranteed to be in L1 cache.

If you update x, y and z in one loop, and vx, vy, vz in a second loop, it is better to merge the two loops.

Two particles will fit in a cache line (taking up 56 bytes).

The next particle will be in *two* cache lines.



Alignment

Cache line size and data alignment

What is wrong with this allocation?

```

struct Particle
{
    float x, y, z;
    float vx, vy, vz;
    float mass, dummy;
};
// size: 32 bytes
Particle particles[512];

```

Although two particles will fit in a cache line, we have no guarantee that the address of the first particle is a multiple of 64.

Note:

Is it bad if particles straddle a cache line boundary?

Not necessarily: if we read the array sequentially, we sometimes get 2, but sometimes 0 cache misses.

For random access, this is not a good idea.



Alignment

Cache line size and data alignment

Controlling the location in memory of arrays:

An address that is dividable by 64 has its lowest 6 bits set to zero. In hex: all addresses ending with 40, 80 and C0.

Enforcing this:

```
Particle* particles =
    _aligned_malloc(512 * sizeof( Particle ), 64);
```

Or:

```
__declspec(align(64)) struct Particle { ... };
```

```

...
    & (depth < MAXDEPTH)
...
    c = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = ddn * nc;
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * r);
    Tr) R = (D * nnt - N * (ddn
...
    E * diffuse;
    = true;
...
    efl + refr) && (depth < MAXDEPTH)
...
    D, N );
    efl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse, r);
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &align
e.x + radiance.y + radiance.z) > 0) && (rand
...
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) * (radiance
...
random walk - done properly, closely following
survive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



```
ics
& (depth < MAXDEPTH)
{
    if ( ! inside )
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
}

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

E * diffuse;
= true;

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

MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (depth <
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) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

- Recap
- Data Locality
- Alignment
- False Sharing
- Experiments
- A Handy Guide *(to Pleasing the Cache)*



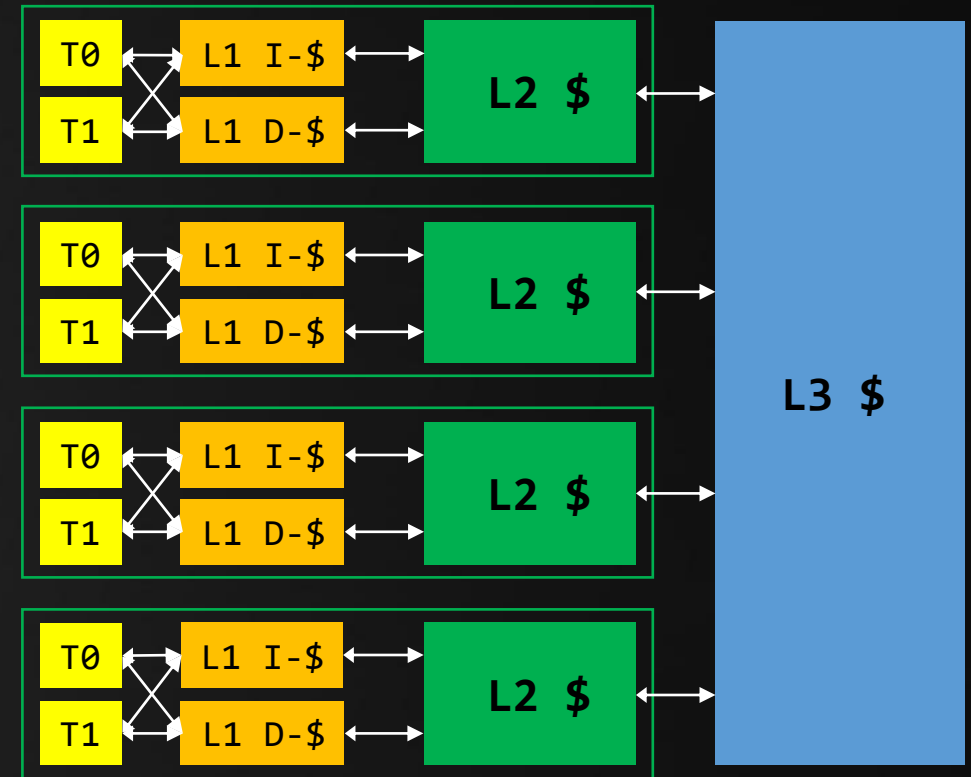
False Sharing

Multiple Cores using Caches

Two cores can hold copies of the same data.

Not as unlikely as you may think – Example:

```
byte data = new byte[COUNT];
for( int i = 0; i < COUNT; i++ )
    data[i] = rand() % 256;
// count byte values
int counter[256];
for( int i = 0; i < COUNT; i++ )
    counter[byteArray[i]]++;
```



```
ics
(depth < MAXDEPTH)
inside ? 1 : 0;
nt = nt / nc; ddn = sqrt(1 - nt);
s2t = 1.0f - nnt * ddn;
D, N );
);
at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * ddn);
Tr) R = (D * nnt - N * (ddn *
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse, 1);
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (depth <
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) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

- Recap
- Data Locality
- Alignment
- False Sharing
- Experiments
- A Handy Guide *(to Pleasing the Cache)*



Experiments

Cache Size

Basic test:

- random access
- increasing data size

Additional test:

- payload size
- data type

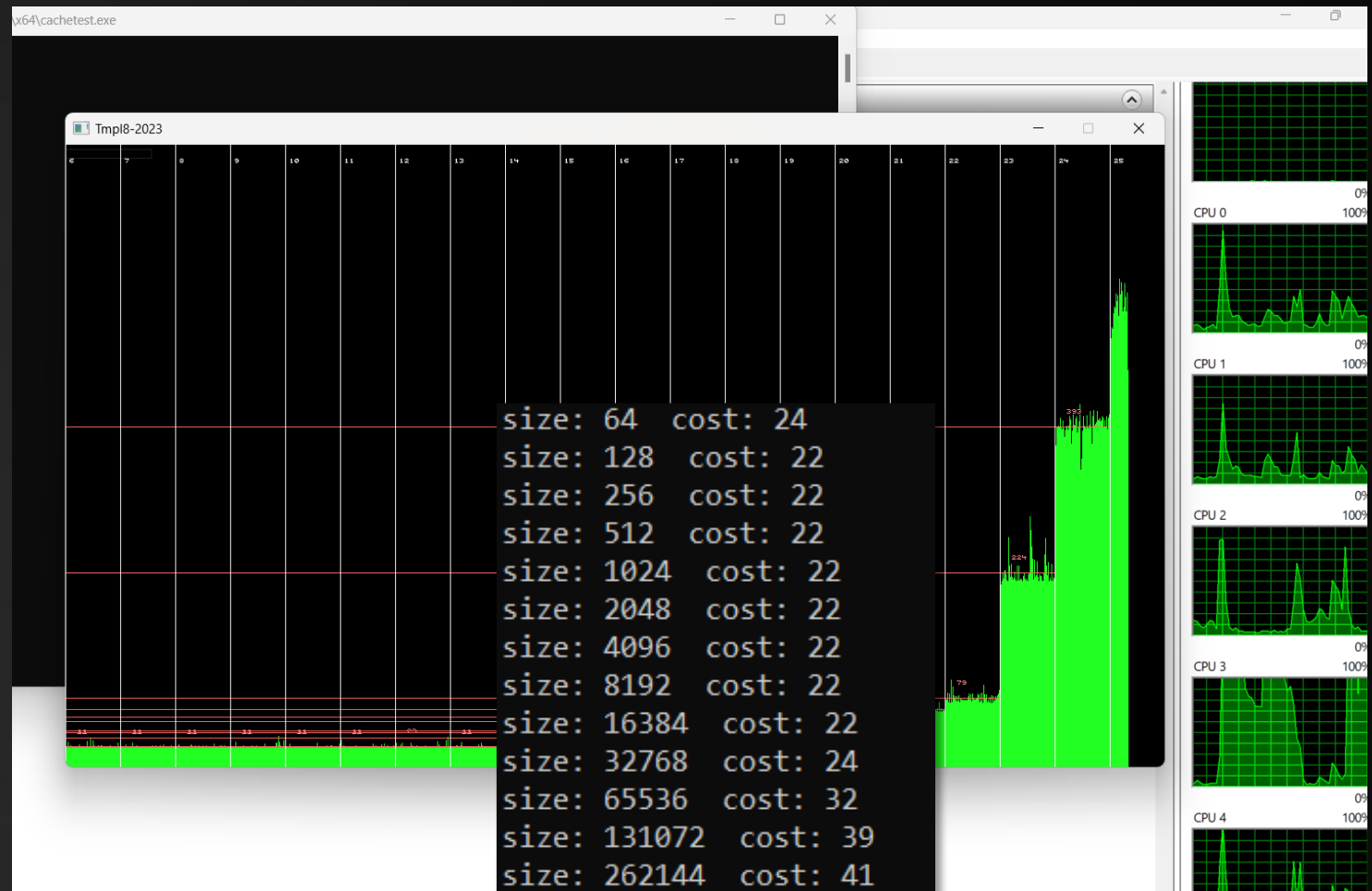
Additional test:

- straddling cache lines

```

ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn / d;
s2t = 1.0f - nnt * d;
D, N );
);
at a = nt - nc, b = nt * n;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (dd
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse, r,
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (rand
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) * (radiance
random walk - done properly, closely following 3e-10
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Experiments

Cache Size

Basic test:

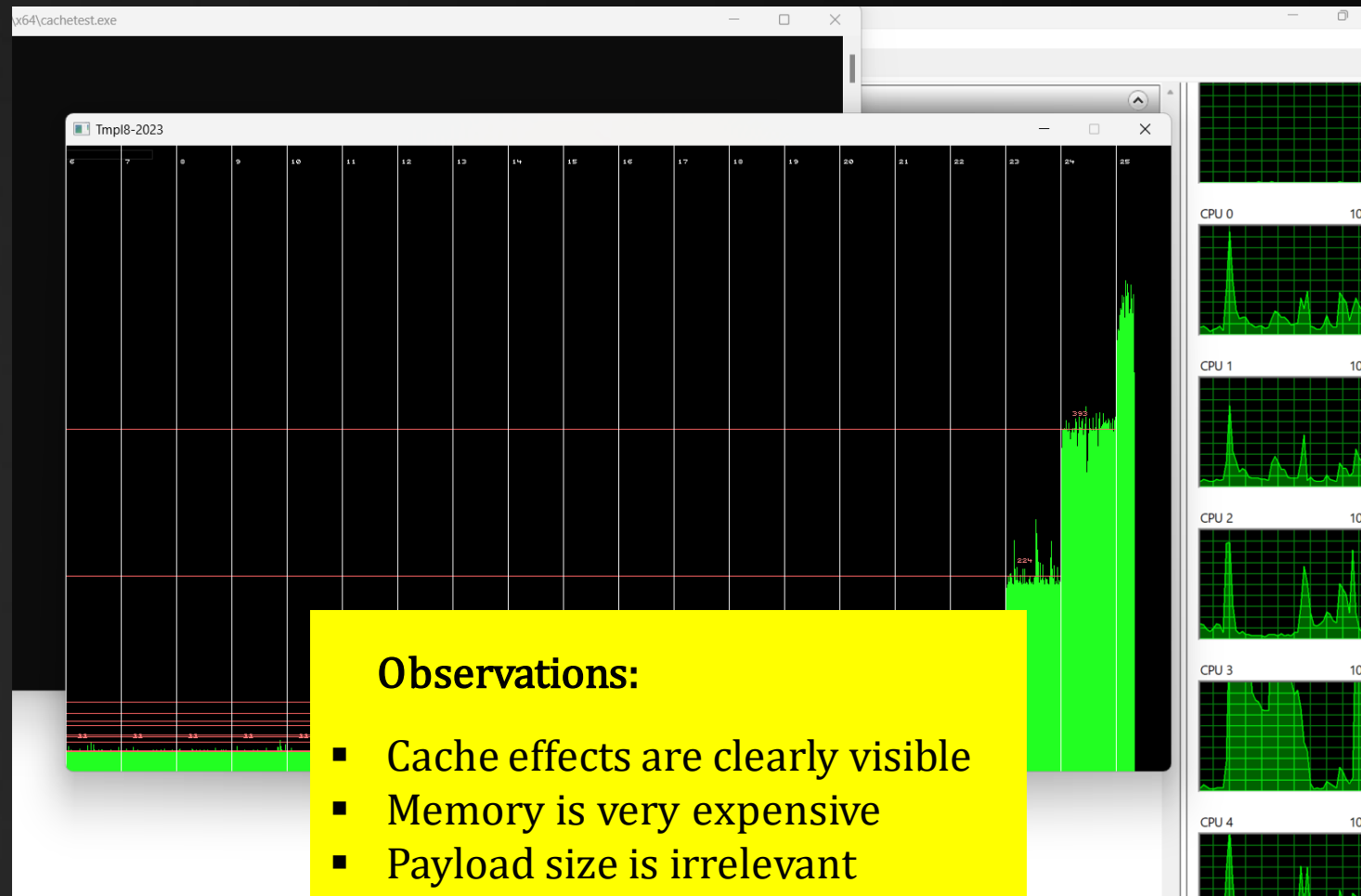
- random access
- increasing data size

Additional test:

- payload size
- data type

Additional test:

- straddling cache lines



Observations:

- Cache effects are clearly visible
- Memory is very expensive
- Payload size is irrelevant
- Datatype is irrelevant
- Straddling is irrelevant (!)
- ...



Experiments

Linear Data Access (1)

Horizontal versus vertical...

Experiment:

Getting it to work

- suspiciously fast...
- ...until we include i in the result. 😊

Changing access pattern

- running average
- swapping loops

```
static Payload table[1024][1024]; // 4MB
Timer t;
Payload sum;
for( int i = 0; i < 100; i++ )
{
    for (int x = 0; x < 1024; x++ )
    {
        for( int y = 0; y < 1024; y++ )
        {
            sum += table[y][x];
        }
    }
}
float elapsed = t.elapsed() * 1000;
```



Experiments

Linear Data Access

Horizontal versus vertical...

Experiment:

Getting it to work

- suspiciously fast...
- ...until we include i in the re...

Changing access pattern

- running average
- swapping loops

Observations:

- Compiler is clever
- Linear access matters!
- ...

```

...ics
& (depth < MAXDEPTH)
...
c = inside ? 1 : 0;
nt = nt / nc; add = add + c;
os2t = 1.0f - nnt * (add / D, N);
...
at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * Tr); R = (D * nnt - N * (add / D, N));
...
E * diffuse;
= true;
...
efl + refr) && (depth < MAXDEPTH)
...
D, N);
refl * E * diffuse;
= true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse, r1, r2, &R, Spdf );
estimation - doing it properly, closely following the
if;
radiance = SampleLight( &rand, I, &L, Align );
e.x + radiance.y + radiance.z) > 0) && (add / D, N);
...
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) * (radiance
...
random walk - done properly, closely following the
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Spdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```

```

static Payload table[1024][1024]; // 4MB
Timer t;
Payload sum;
for( int i = 0; i < 100; i++ )
{
    for (int x = 0; x < 1024; x++ )
    {
        for (int y = 0; y < 1024; y++ )
        {
            table[x][y] * 1000;
        }
    }
}
    
```

```

for (int x = 0; x < 1024; x++)
00000000140001090 mov     edx,400h
00000000140001095 nop     word ptr [rax+rax]
    {
        for (int y = 0; y < 1024; y++)
000000001400010A0 imul   eax,dword ptr [rcx],64h
000000001400010A3 add    rcx,4
000000001400010A7 add    edi,eax
000000001400010A9 sub    rdx,1
000000001400010AD jne    Tmpl8::Game::Tick+40h (01400010A0h)
        Payload sum;
        for (int i = 0; i < 100; i++)
000000001400010AF sub    r8,1
000000001400010B3 jne    Tmpl8::Game::Tick+30h (0140001090h)
            {
                sum += table[y][x];
            }
        }
    }
}
    
```



Experiments

False Sharing

Experiment:

- Counting, single-threaded
- Counting, multi-threaded

Observations:

- False sharing is no joke!
- Use per-thread counters.

<https://cdrdv2-public.intel.com/671363/vtune-tutorial-linux-identifying-false-sharing.pdf>

```
static int counters[256];
Timer t;
#pragma omp parallel for schedule(dynamic)
for( int chunk = 0; chunk < 16; chunk++ )
{
    Payload* chunkStart = mem + chunk * 1024 * 1024;
    for( int i = 0; i < 1024 * 1024; i++ )
    {
        Payload value = chunkStart[i];
        counters[value]++;
    }
}
float elapsed = t.elapsed() * 1000;
```

Common pitfall:

- Random number generator seed
- Any case of ‘why is this not scaling’.
- Solution: Use thread_local keyword.



Easy Steps

How to Please the Cache

Or: “how to evade RAM”

1. Keep your data in registers

Use fewer variables

Limit the scope of your variables

Pack multiple values in a single variable

Use floats and ints (they use different registers)

Compile for 64-bit (more registers)

Arrays will never go in registers

Unions technically can never go in registers

```

ics
& (depth < MAXDEPTH)
c = inside ? 1.0f : 0.0f;
nt = nt / nc; ddn = ddn * ddn;
ps2t = 1.0f - nnt; nnt = nnt * nnt;
D, N );
0)
at a = nt - nc, b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * nnt);
Tr) R = (D * nnt - N * (1 - nnt));
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse, nnt );
estimation - doing it properly, closely following the path
if;
radiance = SampleLight( &rand, I, &L, &light );
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
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) * (radiance.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
random walk - done properly, closely following the path
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```

Liefde is...



... hem het cadeau geven dat hij écht wil.



Easy Steps

How to Please the Cache

Or: “how to evade RAM”

2. Keep your data local

Read sequentially

Keep data small

Use tiling / Morton order

Fetch data once, work until done (streaming)

Reuse memory locations

```

ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn * ddn;
ps2t = 1.0f - nnt * nnt;
D, N );
0)
at a = nt - nc, b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * R);
Tr) R = (D * nnt - N * (ddn *
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse, i);
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (rand() < psurvive)
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) * (radiance
random walk - done properly, closely following
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```

Liefde is...



... hem het cadeau geven dat hij écht wil.



Easy Steps

How to Please the Cache

Or: “how to evade RAM”

3. Respect cache line boundaries

Use padding if needed

Don't pad for sequential access

Use aligned malloc / `__declspec align`

Assume 64-byte cache lines

```

ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn * nc;
ps2t = 1.0f - nnt * ddn;
D, N );
)
at a = nt - nc, b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (dd
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light;
e.x + radiance.y + radiance.z) > 0) && (rand() <
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) * (radiance
andom walk - done properly, closely following
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Liefde is...



... hem het cadeau geven dat hij écht wil.



Easy Steps

How to Please the Cache

Or: “how to evade RAM”

4. Advanced tricks

Prefetch

Use a prefetch thread (theoretical...)

Use *streaming writes*

Separate mutable / immutable data

```

ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn / d;
ps2t = 1.0f - nnt * nnt;
D, N );
0);
at a = nt - nc, b = nt * n;
at Tr = 1 - (R0 + (1 - R0)
Tr) R = (D * nnt - N * (dd
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light;
e.x + radiance.y + radiance.z) > 0) && (rand
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) * (radiance
andom walk - done properly, closely following
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Liefde is...



... hem het cadeau geven
dat hij écht wil.



Easy Steps

How to Please the Cache

Or: “how to evade RAM”

5. Be informed

Use the profiler!

```

ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn * ddn;
s2t = 1.0f - nnt * nnt;
D, N );
0);
at a = nt - nc, b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * r);
Tr) R = (D * nnt - N * (ddn *
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light;
e.x + radiance.y + radiance.z) > 0) && (rand <
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) * (radiance
random walk - done properly, closely following
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Liefde is...



... hem het cadeau geven
dat hij écht wil.




```
ics
& (depth < MAXDEPTH)
{
    if ( ! inside )
        return 0;
    int nt = nt / nc; ddn = sqrt(1 - nt);
    double r1 = 2 * M_PI * ddn; r2 = sqrt(1 - ddn * ddn);
    Vec D, N );
    Vec R = (D * nnt - N * (ddn * r1 + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse, I );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light,
    e.x + radiance.y + radiance.z) > 0) && (depth <
    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) * (radiance
    random walk - done properly, closely following
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf,
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```

Today's Agenda:

- Recap
- Data Locality
- Alignment
- False Sharing
- Experiments
- A Handy Guide *(to Pleasing the Cache)*



