

```
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) * ddn);
    (Fr) 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 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;
```

/INFOMOV/

Optimization & Vectorization

J. Bikker - Sep-Nov 2018 - Lecture 4: "Caching (2)"

Welcome!



```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 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 * 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) && (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
vive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

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



Recap

Refresher:

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)
{
    int inside = 1;
    int nt = nt / nc;
    double ddn = ddn * (1.0f - nnt);
    double s2t = 1.0f - nnt;
    double D, N;
    double a = nt - nc;
    double b = nt * nc;
    double Tr = 1 - (R0 + (1 - R0) * nnt);
    double R = (D * nnt - N * (ddn * nnt));
    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
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    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*



```

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) * R);
Tr) R = (D * nnt - N * (ddn * nnt));

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 < MAXDEPTH)
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

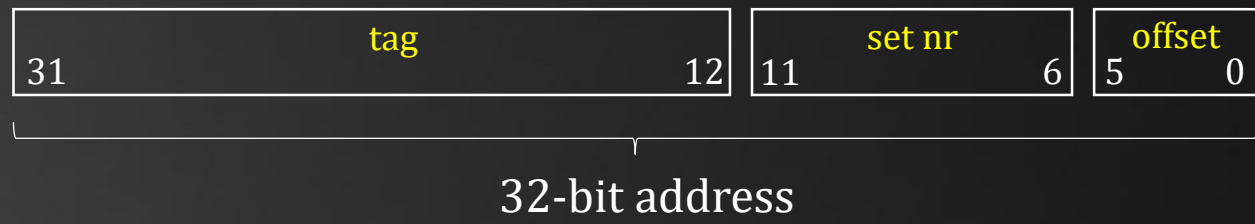
random walk - done properly, closely following
ive)

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



Recap

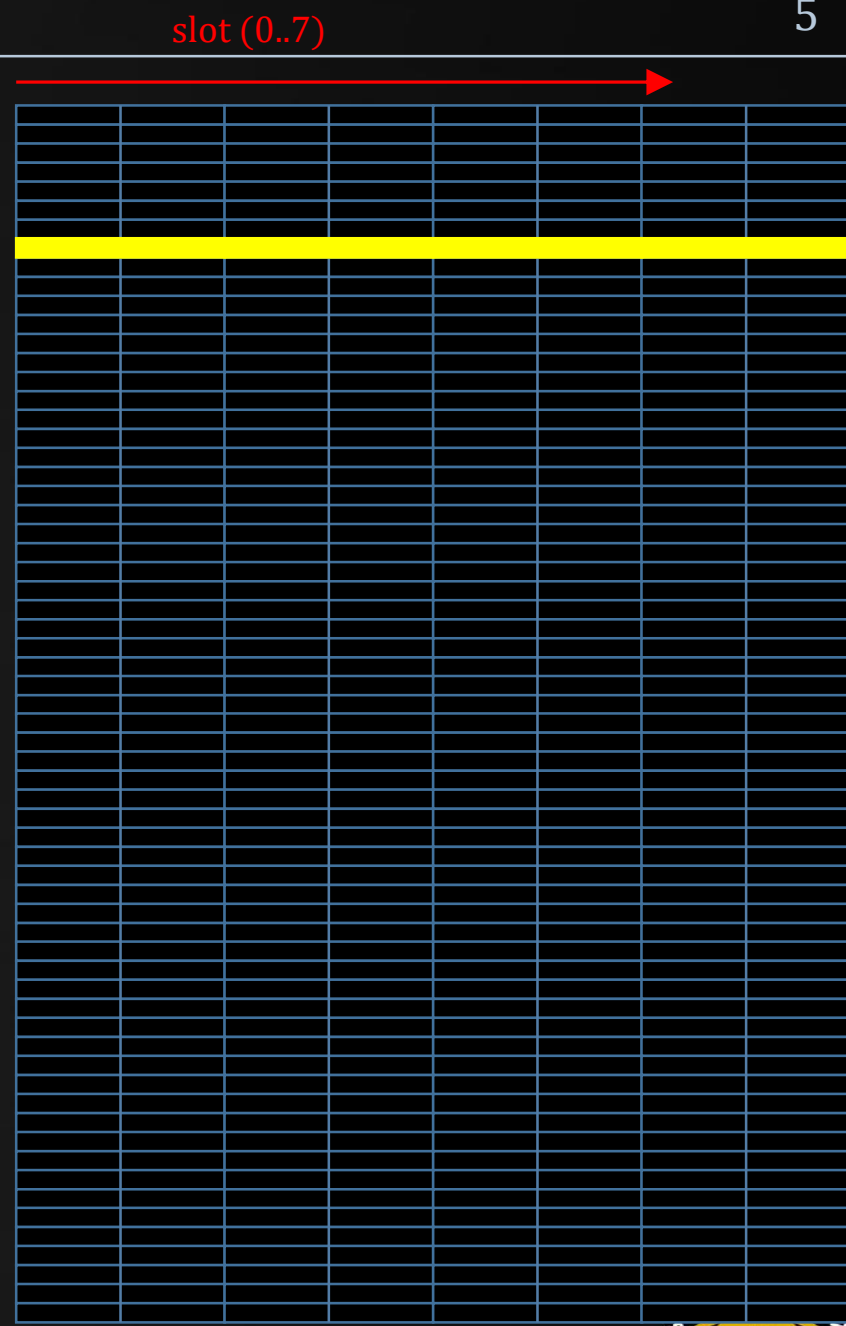
*32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes*



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)



Recap

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

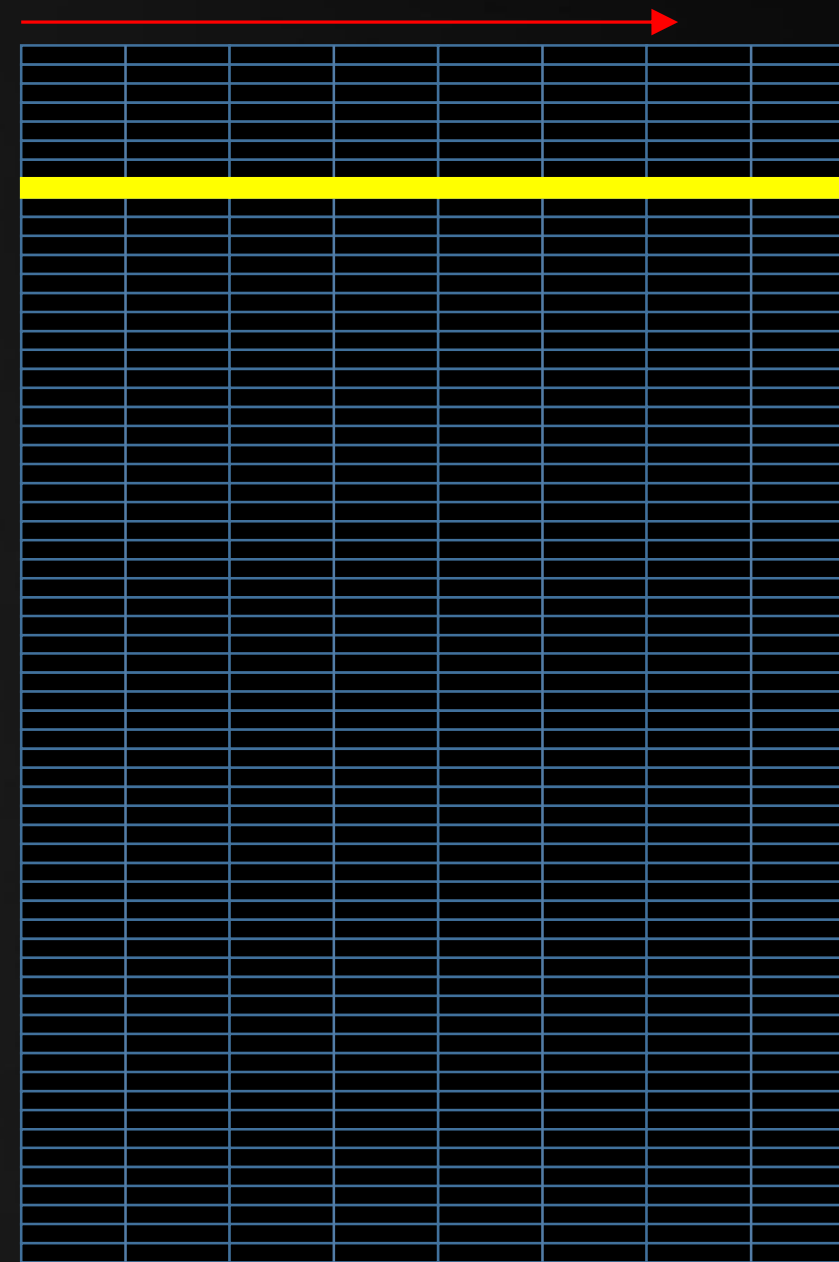


Examples:

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

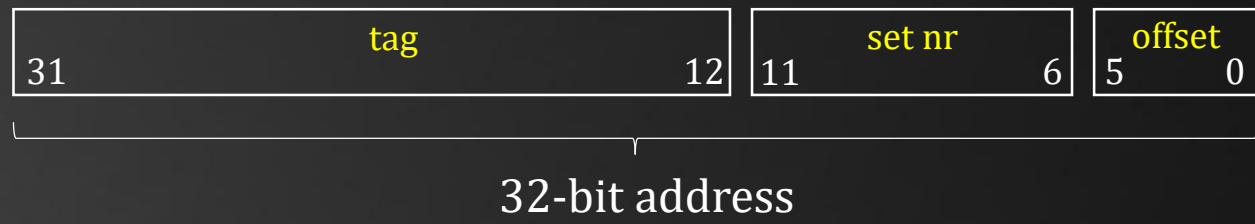
set: 0..63 (6 bit)

slot (0..7)



Recap

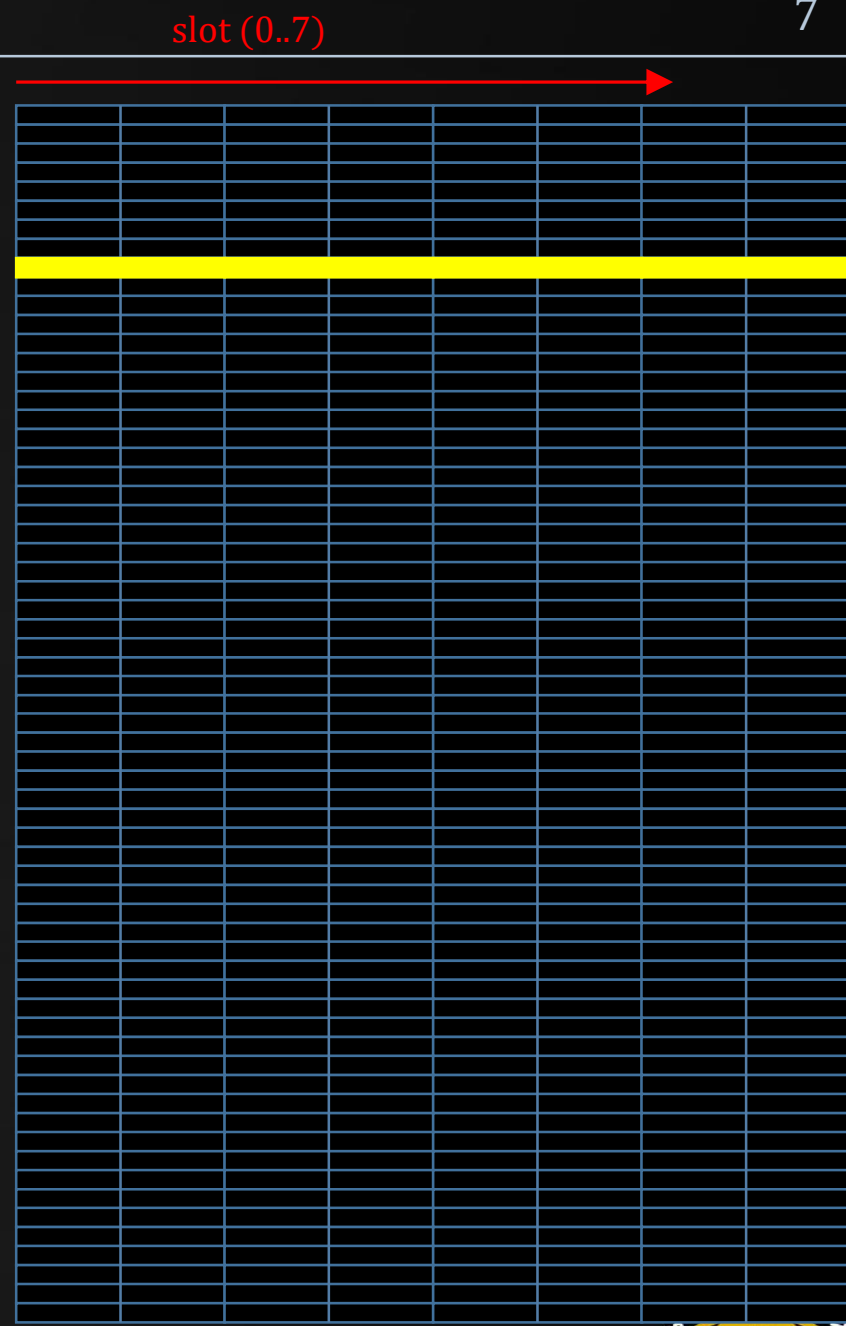
*32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes*



Examples:

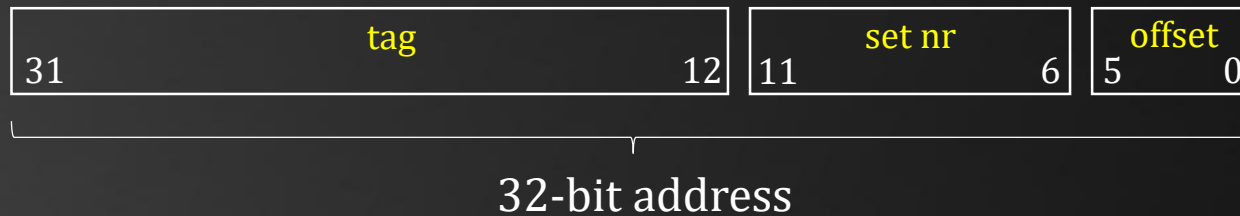
0x00001234	0001	001000	110100
0x00008234	1000	001000	110100
0x00006234	0110	001000	110100
0x0000A234	1010	001000	110100
0x0000A240	1010	001001	000000
0x0000F234	1111	001000	110100

set: 0..63 (6 bit)



Recap

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



Theoretical consequence:

- Address 0, 4096, 8192, ... map to the same set (which holds max. 8 addresses)
- consider `int value[1024][1024]`:
 - `value[0,1,2...][x]` map to the same set
 - querying this array vertically:
 - will quickly result in evictions
 - will use only 512 bytes of your cache



Recap

64 bytes per cache line

Theoretical consequence:

- If address X is pulled into the cache, so is $(X+1 \dots X+63)$.

Example*:

```
int arr = new int[64 * 1024 * 1024];
// loop 1
for( int i = 0; i < 64 * 1024 * 1024; i++ ) arr[i] *= 3;
// loop 2
for( int i = 0; i < 64 * 1024 * 1024; i += 16 ) arr[i] *= 3;
```

Which one takes longer to execute?

*: <http://igoro.com/archive/gallery-of-processor-cache-effects>



Recap

64 bytes per cache line

Theoretical consequence:

- If address X is removed from cache, so is $(X+1 \dots X+63)$.
- If the object you’re querying straddles the cache line boundary, you may suffer not one but *two* cache misses.

Example:

```
struct Pixel { float r, g, b; }; // 12 bytes
Pixel screen[768][1024];
```

Assuming pixel (0,0) is aligned to a cache line boundary, the offsets in memory of pixels (0,1..5) are 12, 24, 36, 48, 60, Walking column 5 will be very expensive.



Recap

Considering the Cache

- Size
- Cache line size and alignment
- Aliasing
- Sharing
- Access patterns

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        ns2t = 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;

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) && (occl

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
vive)

;
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 ? 1 : 0)
    {
        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) * ddn);
Tr) R = (D * nnt - N * (ddn * 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) && (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
vive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

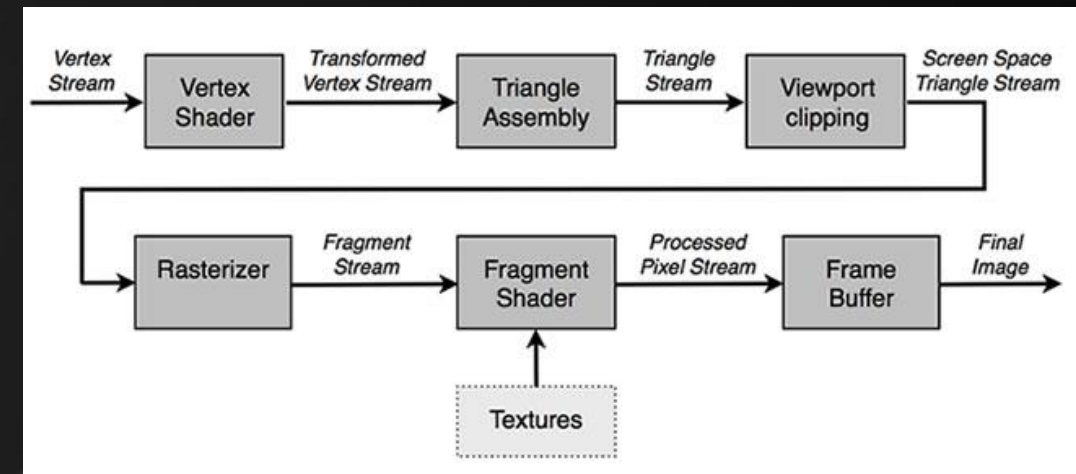
- Caching: Recap
- Data Locality
- Alignment
- False Sharing
- A Handy Guide *(to Pleasing the Cache)*



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 CD every night → disk



Data Locality

Reusing data

Ideal pattern:

- load data once, operate on it, discard.

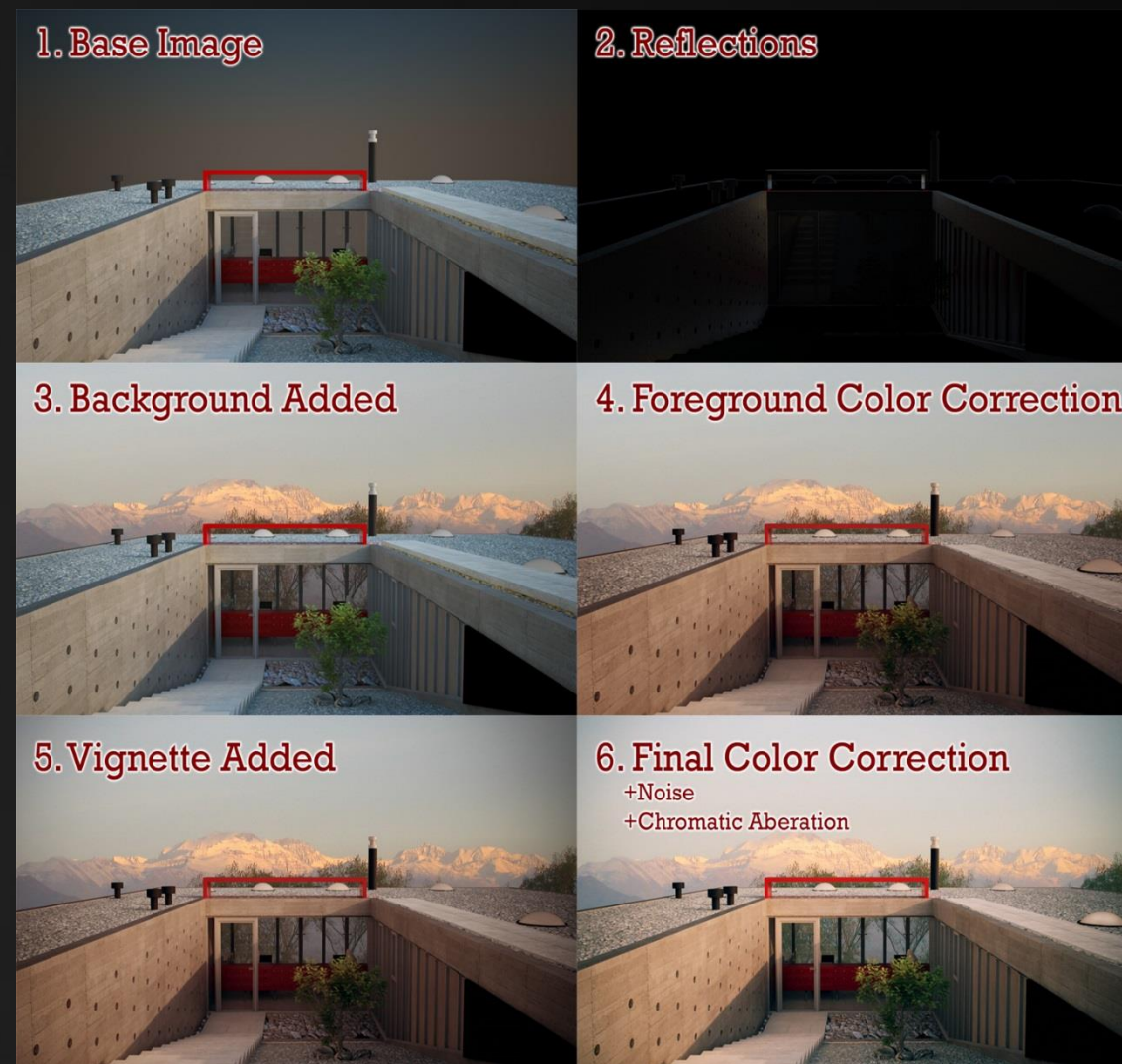
Typical pattern:

- operate on data using algorithm 1, then using algorithm 2, ...

Note:

GPUs typically follow the ideal pattern.

(more on that later)



Data Locality

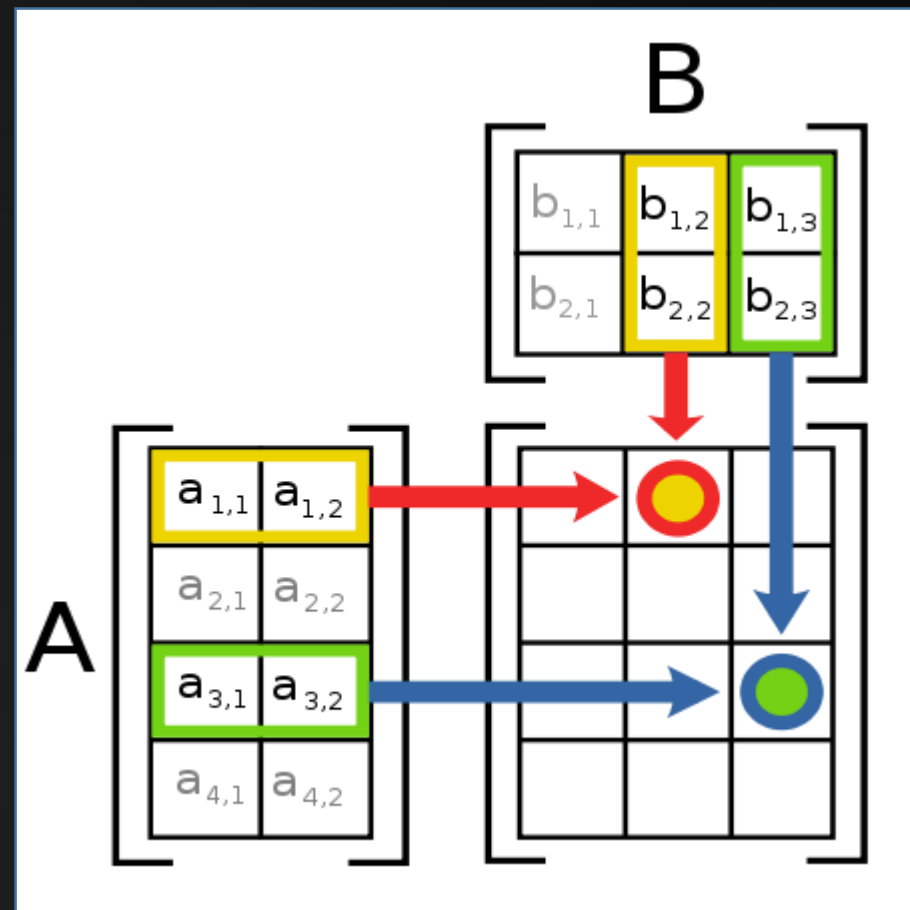
Reusing data

Ideal pattern:

- load data sequentially.

Typical pattern:

- *whatever the algorithm dictates.*



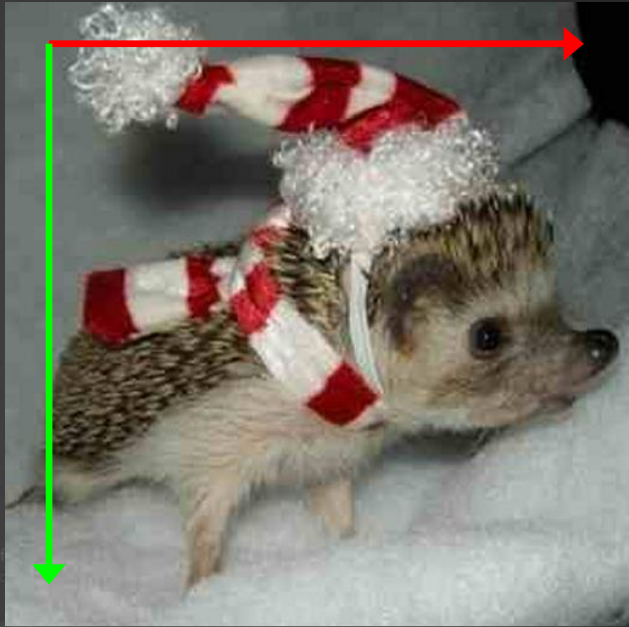
Data Locality

Example: rotozooming

```

ics
& (depth < MAXDEPTH)
= inside ? 1 : 0;
nt = nt / nc; ddn = ddn /
os2t = 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;
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) &&
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 Saito's
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Spdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



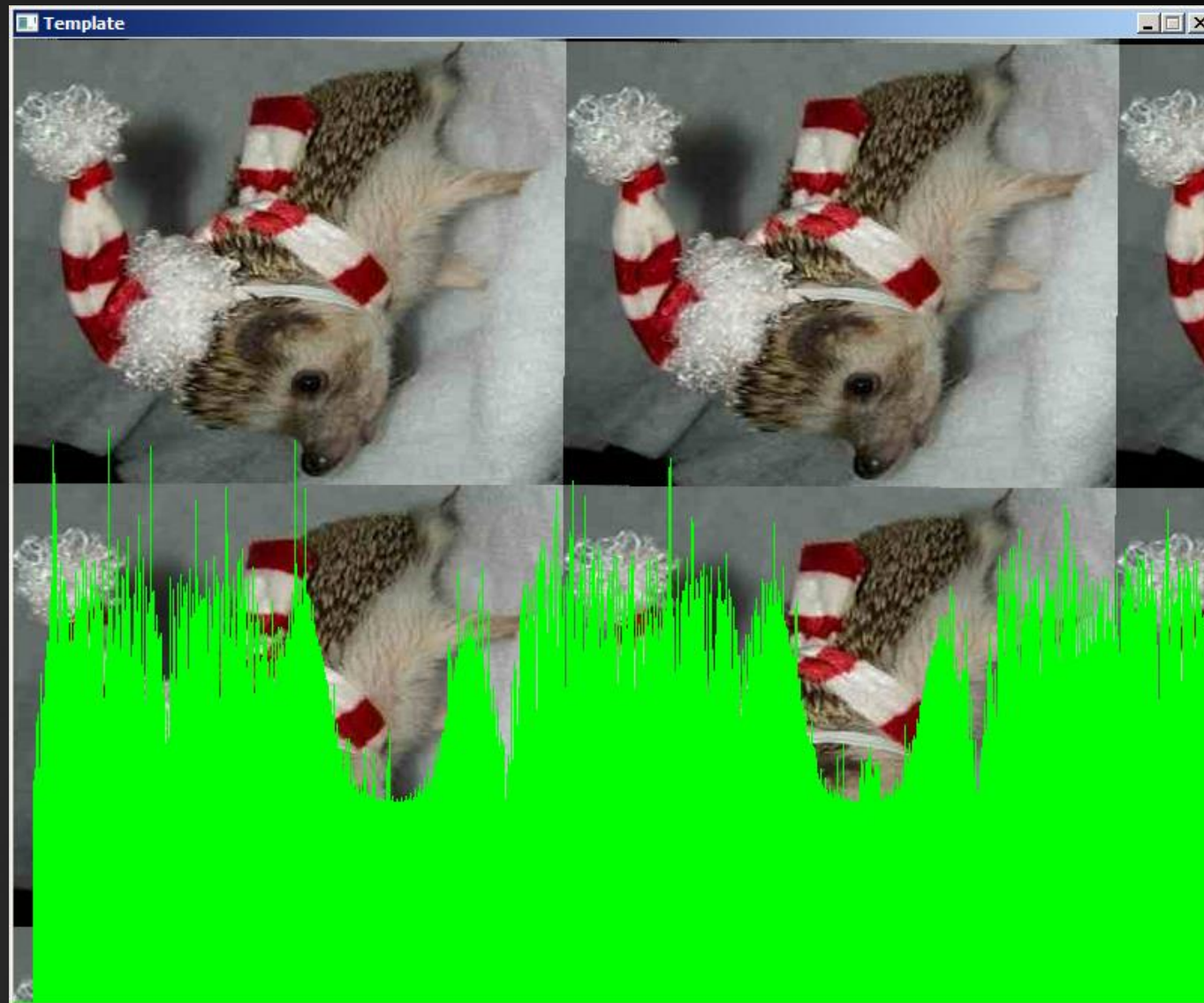
Data Locality

Example: rotozooming

```

ics
& (depth < MAXDEPTH)
    if (inside ? 1 : 0) {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * nnt;
        D, N );
        0)
        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 following
        if;
        radiance = SampleLight( &rand, I, &L, &light;
        e.x + radiance.y + radiance.z) > 0) && (cosThetaOut > 0)
        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

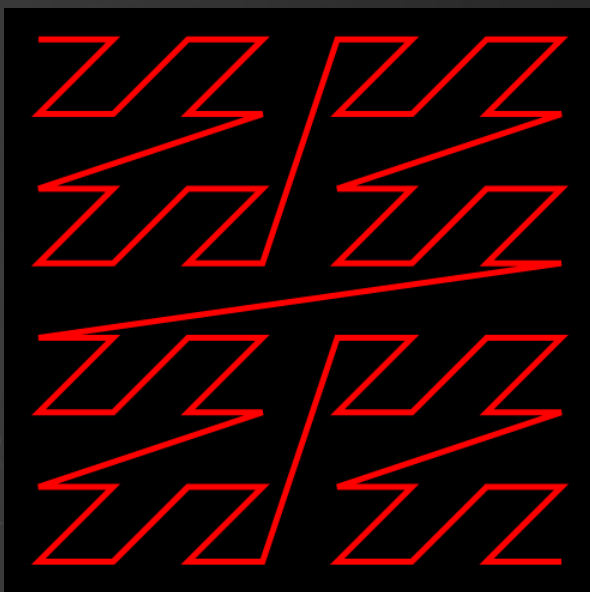
Example: rotozooming

Improving data locality: z-order / Morton curve

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * 0.5;
        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;
efl + 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) && (cos
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) * Psur
random walk - done properly, closely following
vive)
;
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

M = 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)
{
    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,
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (cos
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
vive)
;
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 ? 1 : 0)
    {
        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) * ddn);
Tr) R = (D * nnt - N * (ddn * 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) && (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
vive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

- Caching: Recap
- Data Locality
- Alignment
- False Sharing
- 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 { ... };
```



```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        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) * ddn);
Tr) R = (D * nnt - N * (ddn * nnt));

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

Today's Agenda:

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



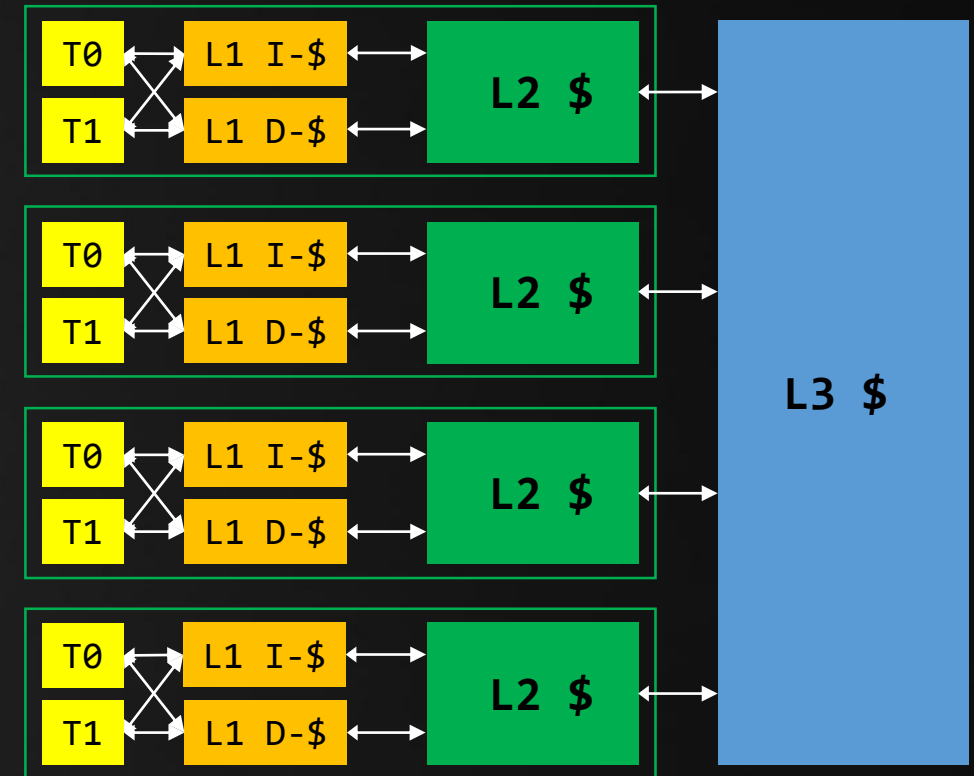
False Sharing

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)
{
    if (inside ? 1 : 0)
    {
        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) * ddn);
Tr) R = (D * nnt - N * (ddn * 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) && (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
vive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

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



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

```

ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn * nc;
ps2t = 1.0f - nnt * (ddn *
D, N );
0)
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,
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 Section 3.2.1
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf,
urvive;
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) && (rand < 0.5);
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 * nnt;
D, N );
0)
at a = nt - nc, b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * nnt);
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() <
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
vive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Easy Steps

How to Please the Cache

Or: “how to evade RAM”

4. Advanced tricks

Prefetch

Use a prefetch thread

Use streaming writes

Separate mutable / immutable data

```

ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn * ddn;
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 * (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
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;
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 );
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
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.



```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 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 * 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) && (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
vive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

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



/INFOMOV/

END of “Caching (2)”

next lecture: “SIMD (1)”

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        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 * ddn));

    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, &L, &light);
    e.x + radiance.y + radiance.z) > 0) && (cosThetaOut > 0)
    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;
}
```

