

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.2f * nc)
    {
        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) * ddn);
    (Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &light,
    e.x + radiance.y + radiance.z) > 0) && (ace)
    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;
}
```

/INFOMOV/ Optimization & Vectorization

J. Bikker - Sep-Nov 2019 - Lecture 4: “Caching (2)”

Welcome!



```
ics
& (depth < MAXDEPTH) {
    if ( ! inside ) {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    if ( 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) && (acc < 0.0001)
    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)*



1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

```
Survive = SurvivalProbability( diffuse /  
estimation - doing it properly, closely follow  
diff;  
radiance = SampleLight( &rand, I, &L, &R,  
.x + radiance.y + radiance.z) > 0) && 1;  
  
w = true;  
at brdfPdf = EvaluateDiffuse( L, N ) * Pdf;  
at3 factor = diffuse * INWPI;  
at weight = Mis2( directPdf, brdfPdf );  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / directPdf)  
  
random walk - done properly, closely follow  
(survive)  
  
;  
at3 brdf = SampleDiffuse( diffuse, N, r1,  
survive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
ission = true;
```

Three types of cache:

Direct mapped

N-set associative

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

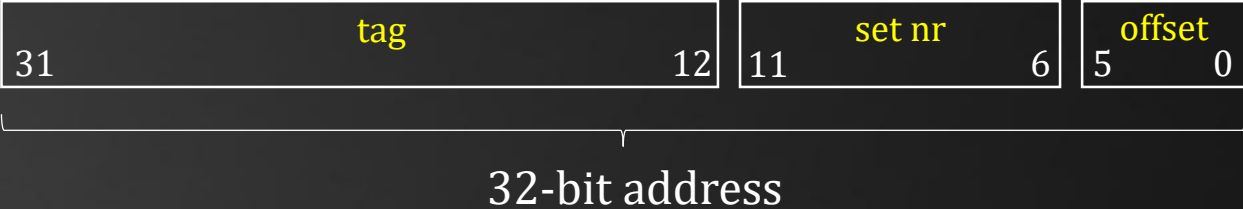
Example:

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



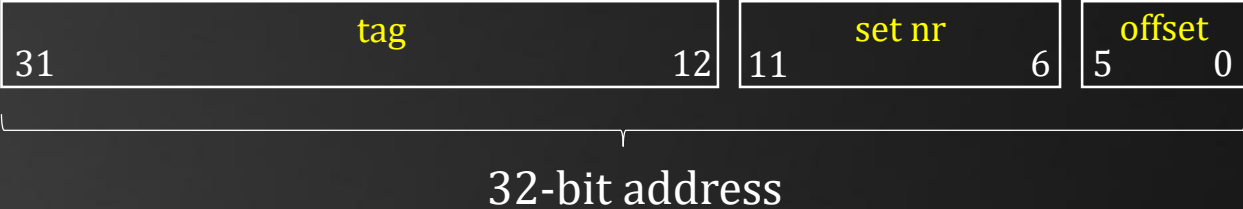
Recap

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



Recap

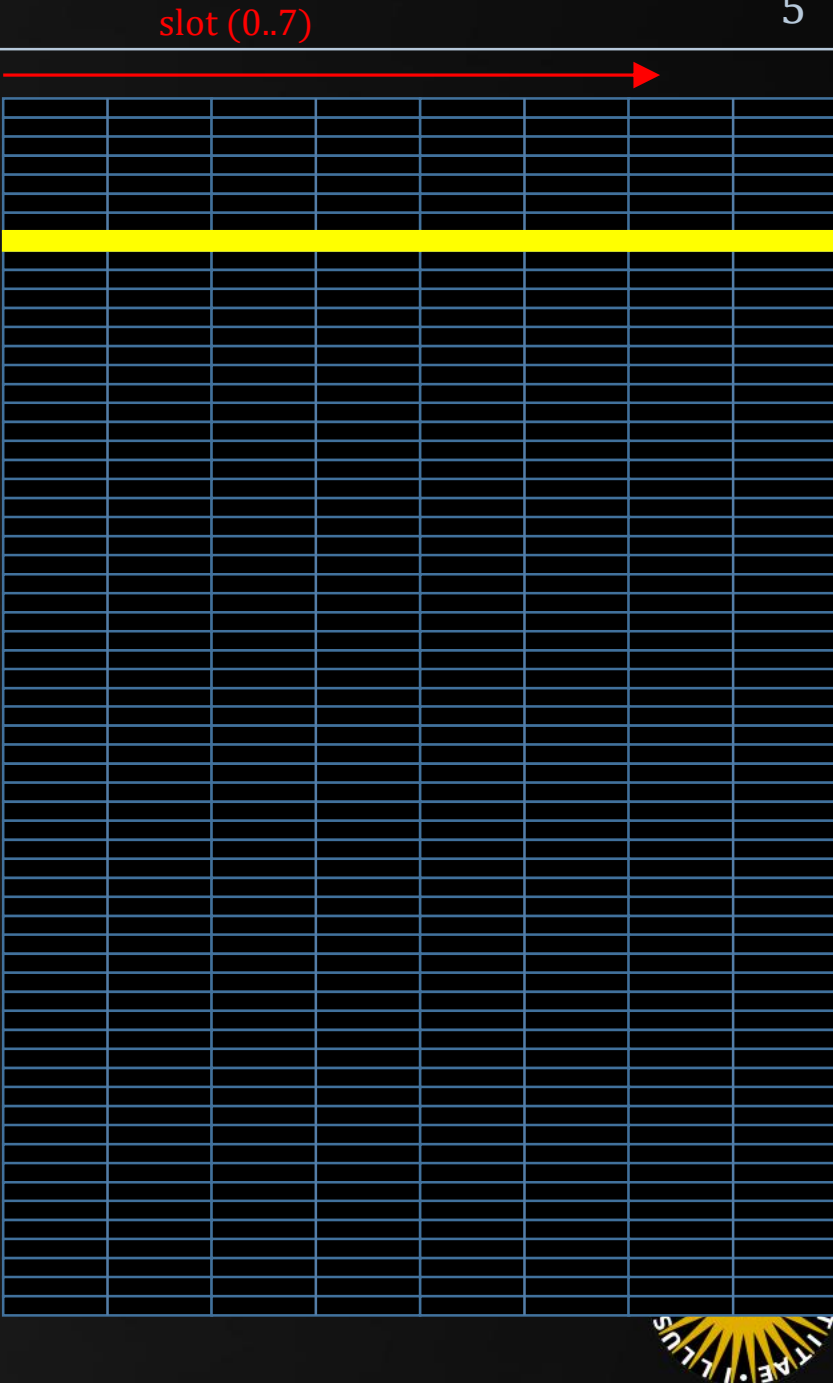
32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes



Examples:

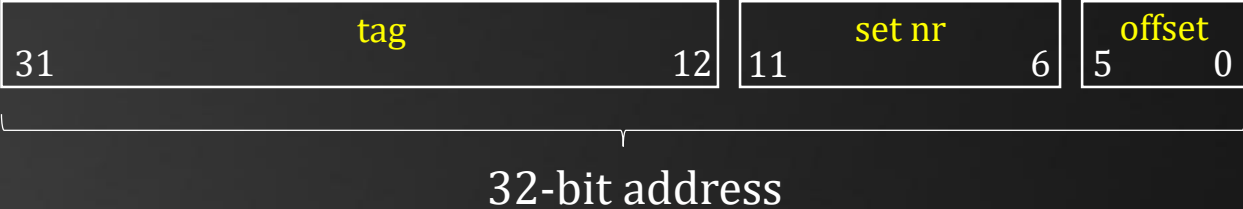
0x00001234	0001	001000	110100
0x00008234	1000	001000	110100
0x00006234	0110	001000	110100
0x0000A234	1010	001000	110100
0x0000A240	1010	001001	000000
0x0000F234	1111	001000	110100

set: 0..63 (6 bit)



Recap

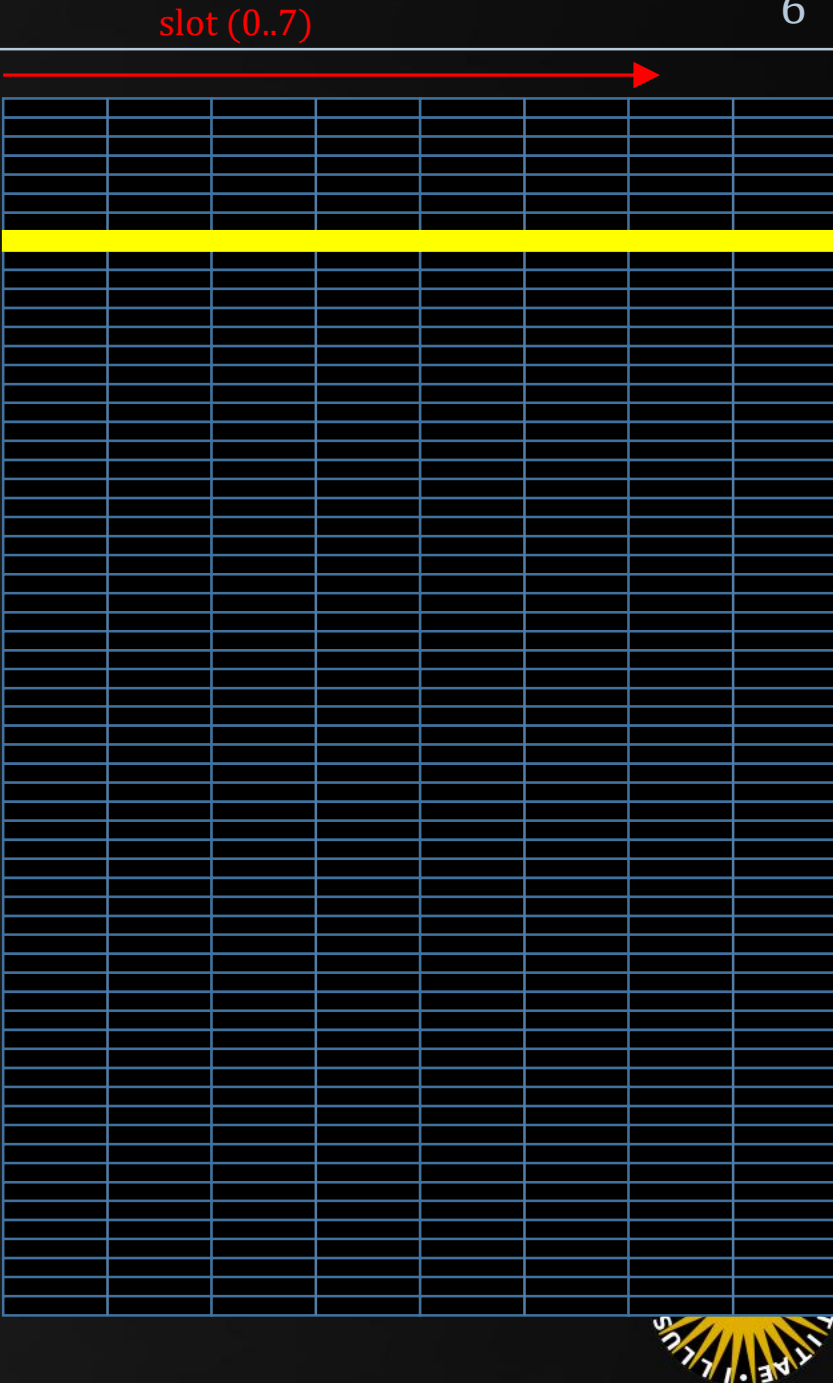
32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes



Examples:

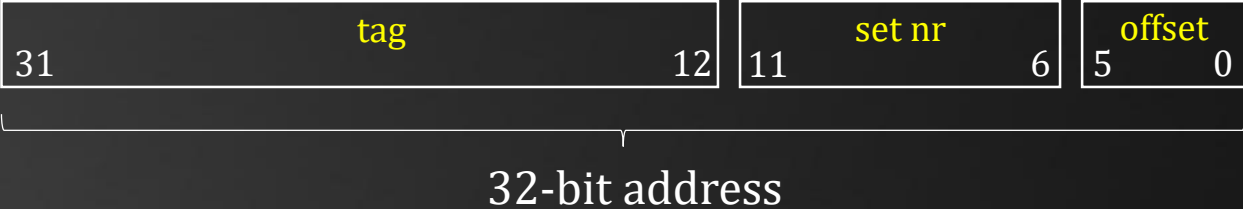
0x00001234	0001	001000	110100
0x00008234	1000	001000	110100
0x00006234	0110	001000	110100
0x0000A234	1010	001000	110100
0x0000A240	1010	001001	000000
0x0000F234	1111	001000	110100

set: 0..63 (6 bit)



Recap

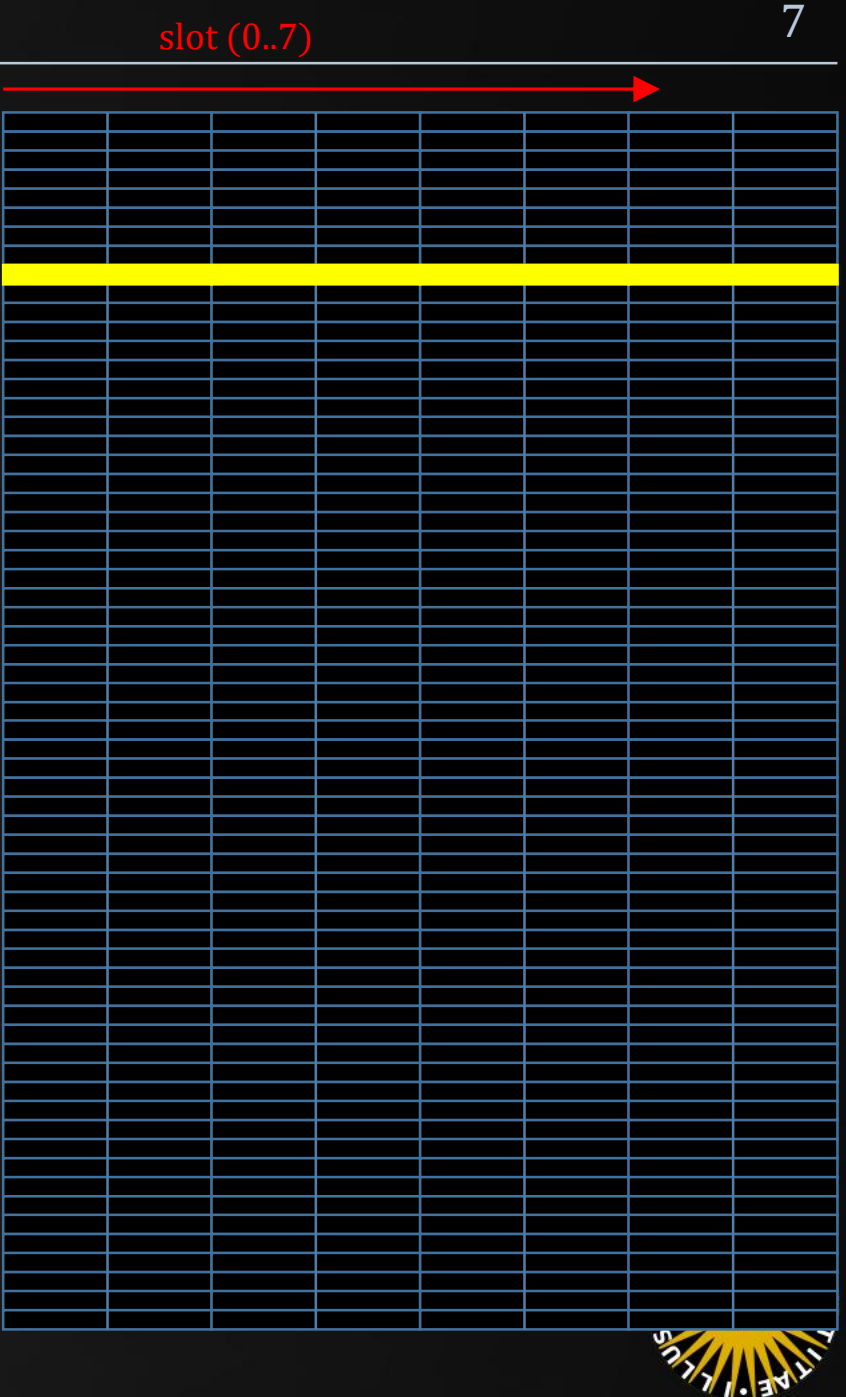
32KB, 8-way set-associative, 64 bytes per cache line:
64 sets of 512 bytes



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.



Considering the Cache

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



```
ics
& (depth < MAXDEPTH) {
    if ( ! inside ) {
        nt = nt / nc, ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    if ( a = nt - nc, b = nt * nc, c =
    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
    df;
    radiance = SampleLight( &rand, I, &L, &light,
    e.x + radiance.y + radiance.z) > 0) && (acc < 0.001) {
        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 SurvivalProb
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf, &
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```

Today's Agenda:

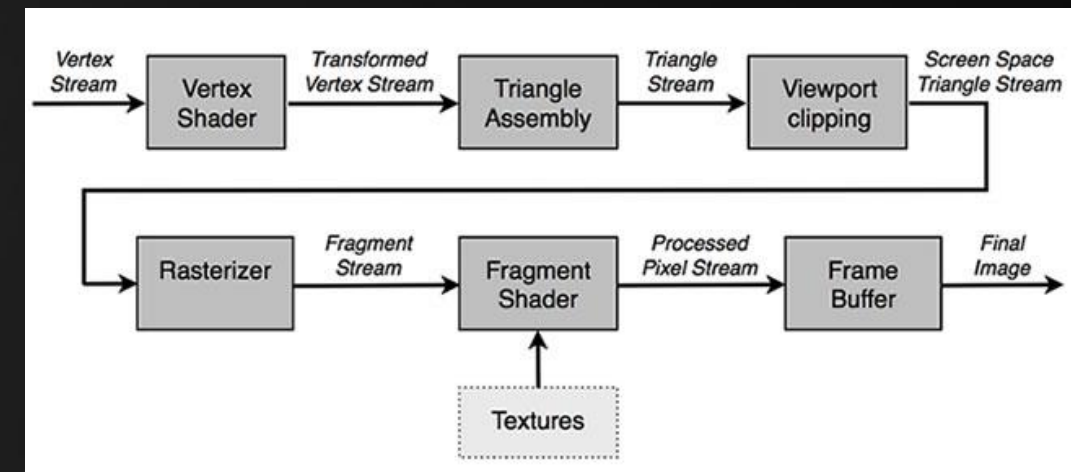
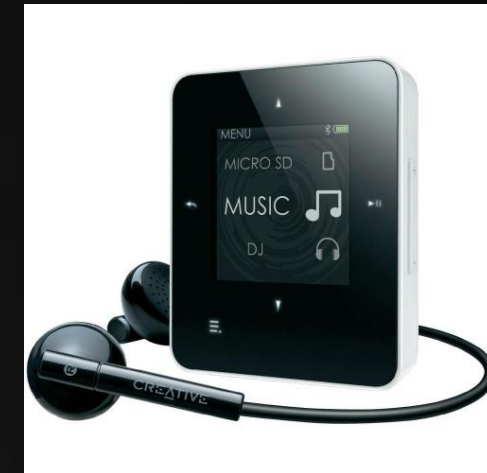
- Caching: Recap
- Data Locality
- Alignment
- False Sharing
- A Handy Guide *(to Pleasing the Cache)*



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

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.0f - nt)
    {
        nt = nt / nc; ddn = ddn * 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 * (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) && (acc + radiance.x +
    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,
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```

- 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

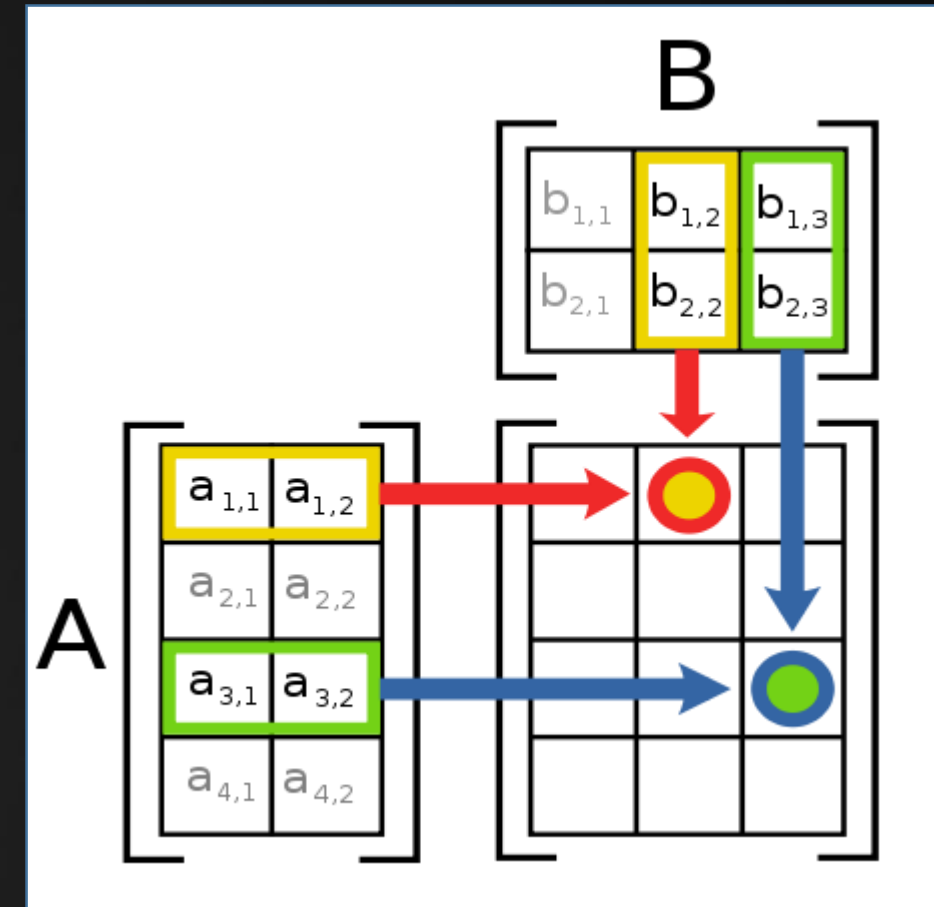
Reusing data

Ideal pattern:

- load data sequentially.

Typical pattern:

- whatever the algorithm dictates.*



Data Locality

Example: rotozooming

```

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

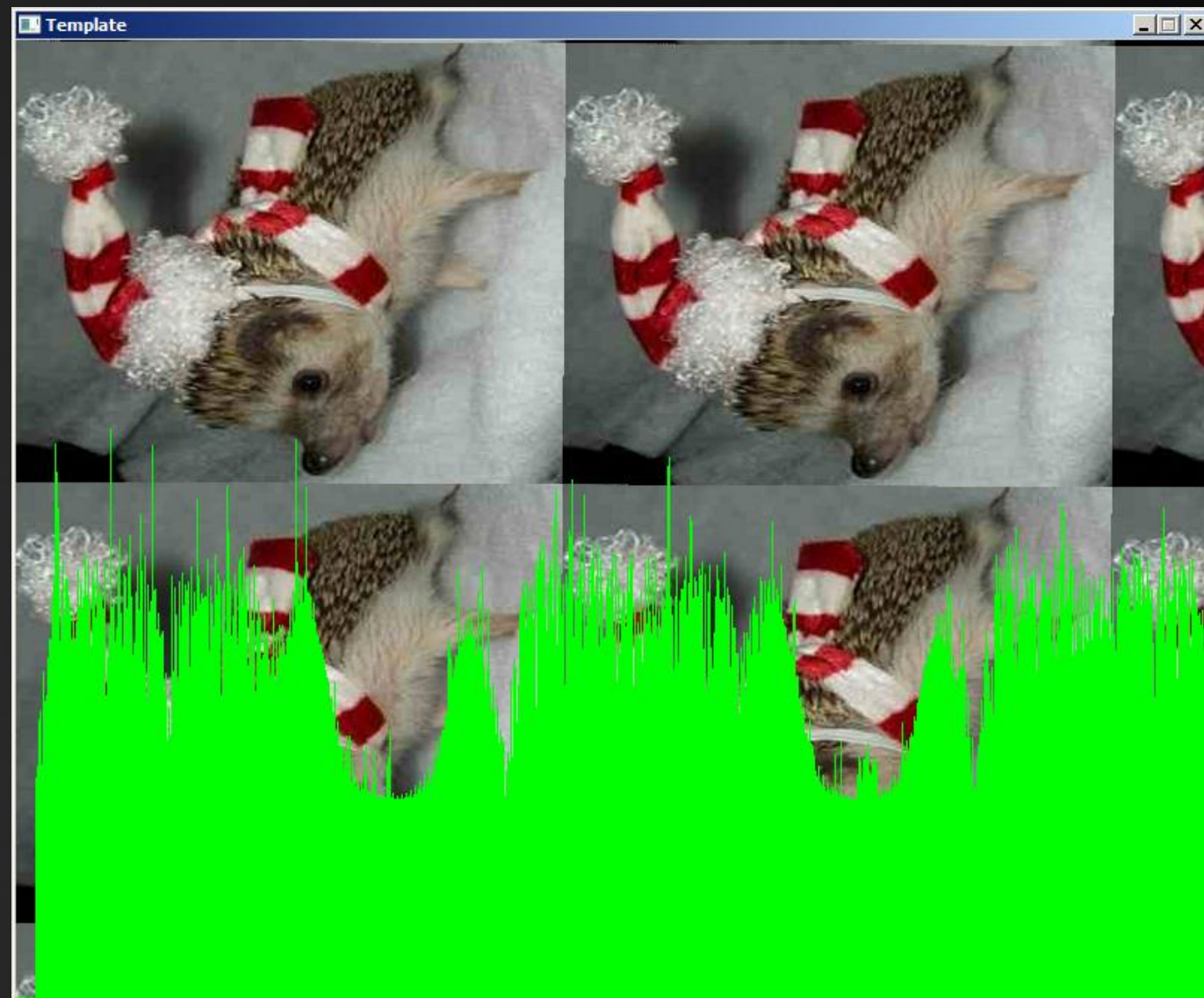


Data Locality

Example: rotozooming

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0.2)
    {
        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;
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &align,
    e.x + radiance.y + radiance.z) > 0) && (ace)
    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;
    
```



Data Locality

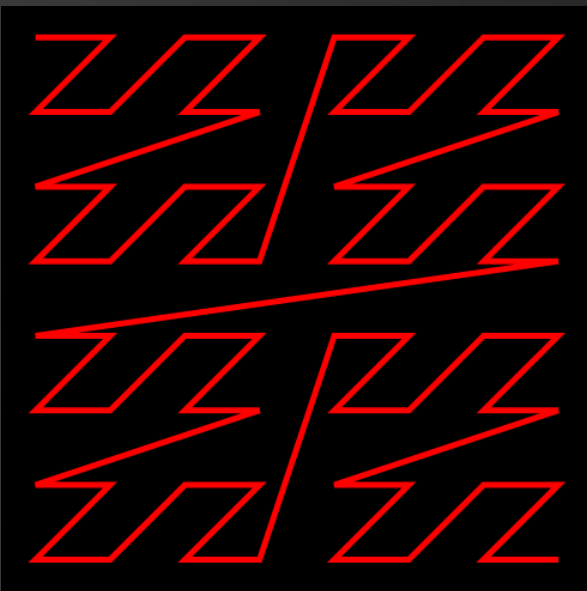
Example: rotozooming

Improving data locality: z-order / Morton curve

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        pos2t = 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;
    }
    {
        refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    {
        MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely
        df;
        radiance = SampleLight( &rand, I, &L, &align
        e.x + radiance.y + radiance.z) > 0) && (do
    }
    {
        w = true;
        at brdfPdf = EvaluateDiffuse( L, N ) * Psun
        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;
    }
}

```



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



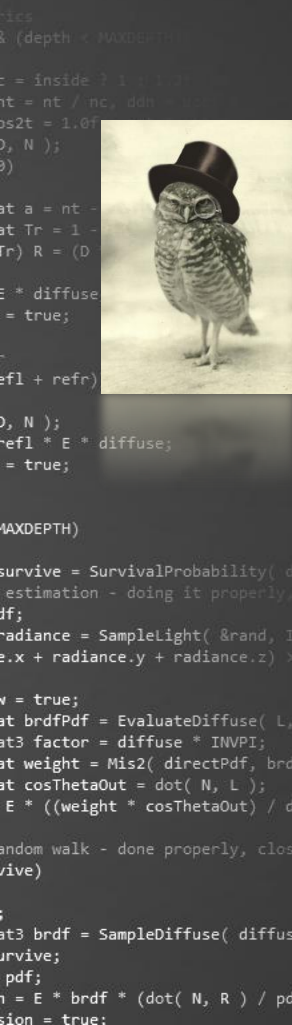
Data Locality

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 : 1.2f) {
        nt = nt / nc, ddn = ddn * nc;
        rnt = 1.0f - nnt * nnt;
        D, N );
    }
    if (a = nt - nc, b = nt * nc, c =
    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
    df;
    radiance = SampleLight( &rand, I, &L, &light,
    e.x + radiance.y + radiance.z) > 0) && (acc < 0.001) {
        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 SurvivalProb
    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 ) {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    if ( 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) && (acc < 0.0001)
    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)*



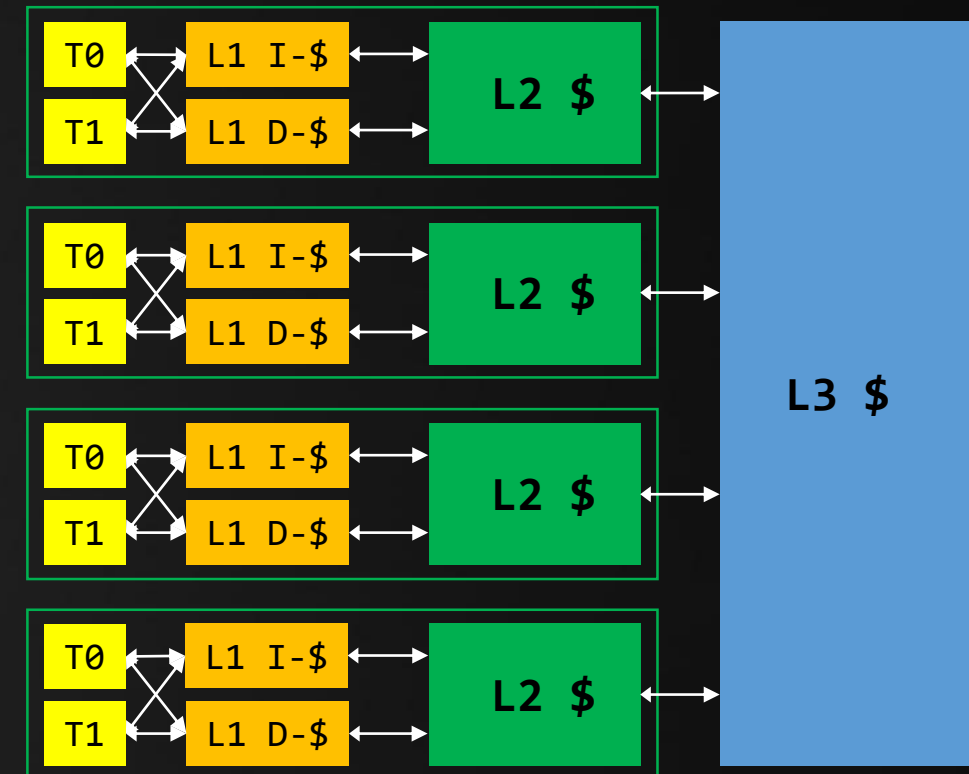
False Sharing

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 ) {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    if ( a = nt - nc, b = nt * nc, c =
    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
    df;
    radiance = SampleLight( &rand, I, &L, &light,
    e.x + radiance.y + radiance.z) > 0) && (acc < 0.0001)
    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 Small's
    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

Unions will never go in registers

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

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)
{
    if (inside ? 1 : 1.0f - nnt)
    {
        nt = nt / nc; ddn = ddn / 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 * (ddn
    E * diffuse;
    = true;
    efl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &align,
    e.x + radiance.y + radiance.z) > 0) && (acc < 1.0f)
    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 S&S
    vive)
    ;
    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)
{
    if (inside ? 1 : 0.2f)
    {
        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 following
    df;
    radiance = SampleLight( &rand, I, &L, &light;
    e.x + radiance.y + radiance.z) > 0) && (accu
    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 S&S;
    vive)
    ;
    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.



How to Please the Cache

Or: “how to evade RAM”

5. Be informed

Use the profiler!



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




```
ics
& (depth < MAXDEPTH) {
    if (inside ? 1 : 1.2f) {
        nt = nt / nc, ddn = ddn * nc;
        rnt = 1.0f - nnt * nnt;
        D, N );
    }
    if (a = nt - nc, b = nt * nc, c =
    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) && (acc < 0.001f) {
        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 SurvivalProb
    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)”



/INFOMOV/

Practical

```
ics
& (depth < MAXDEPTH) {
    if (nt < 0) {
        nt = inside ? 1 : 1.2f;
        nt = nt / nc; ddn = dot(N, N);
        r = 1.0f - nnt * nnt;
        D, N );
    }
    if (a < 0) {
        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;
    -
    refl + refr)) && (depth < MAXDEPTH) {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following 3-sphere model
    if;
    radiance = SampleLight( &rand, I, &L, &light, &N, &D, &N );
    e.x + radiance.y + radiance.z > 0) && (ace) {
        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.x + radiance.y + radiance.z);
    }
    random walk - done properly, closely following 3-sphere model
    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 );
    }

    if ( 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) && (ace)
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 3e
vive)

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

1. Timing the Rotozoomer



Data Locality

Example: rotozooming

```

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

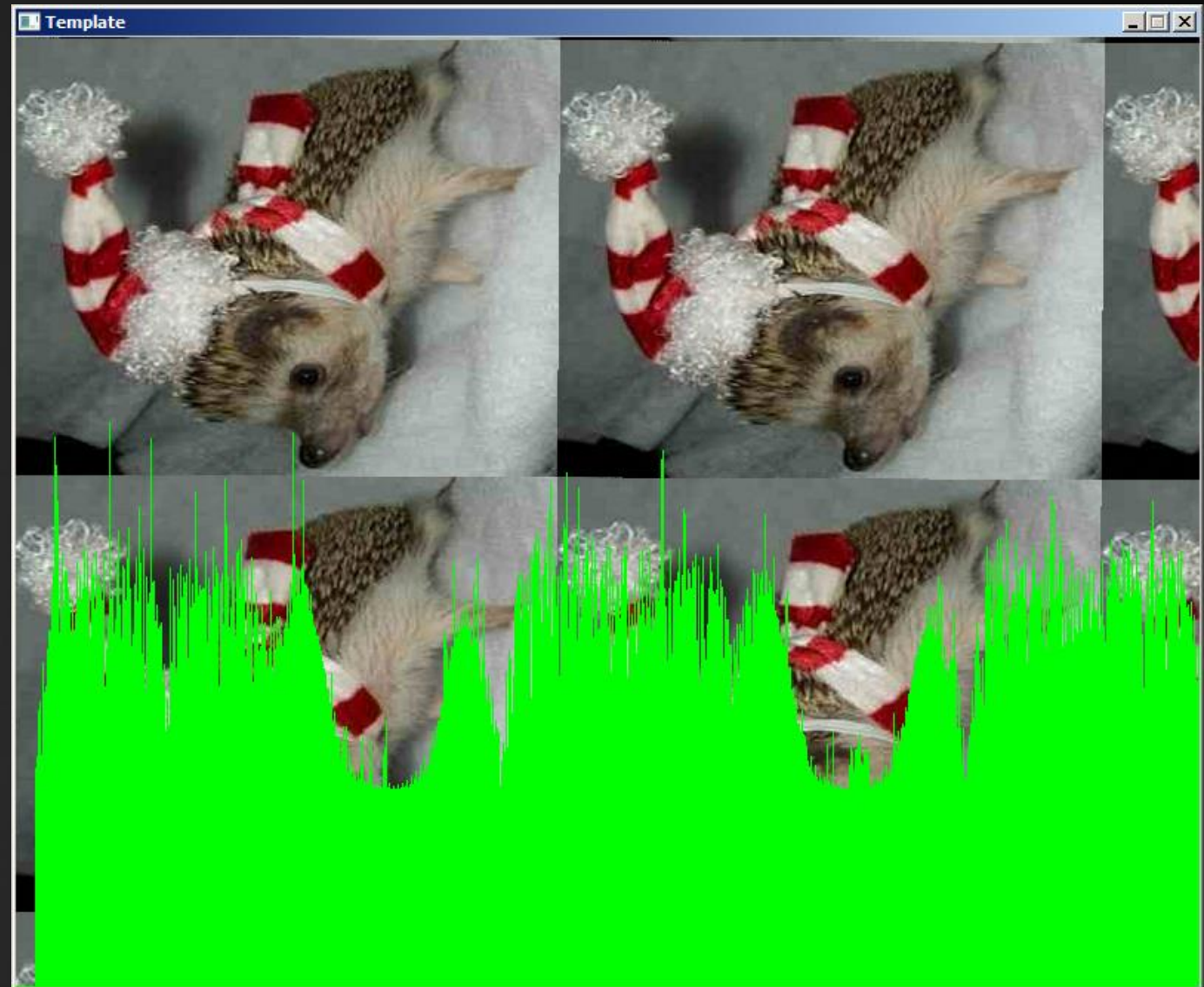


Data Locality

Example: rotozooming

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0.2)
    {
        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 *
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &align,
    e.x + radiance.y + radiance.z) > 0) && (acc < 0.001)
    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;
    
```




```

k (depth < MAXDEPTH);
{
    nc = inside ? 1 : 1.0f;
    nt = nt / nc;
    ddn = dot(N, N);
    cos2t = 1.0f - nt * nt;
    r = sqrt(cos2t);
    D = (D, N);
    0);
}

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

E * diffuse;
= true;

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

MAXDEPTH)

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

```

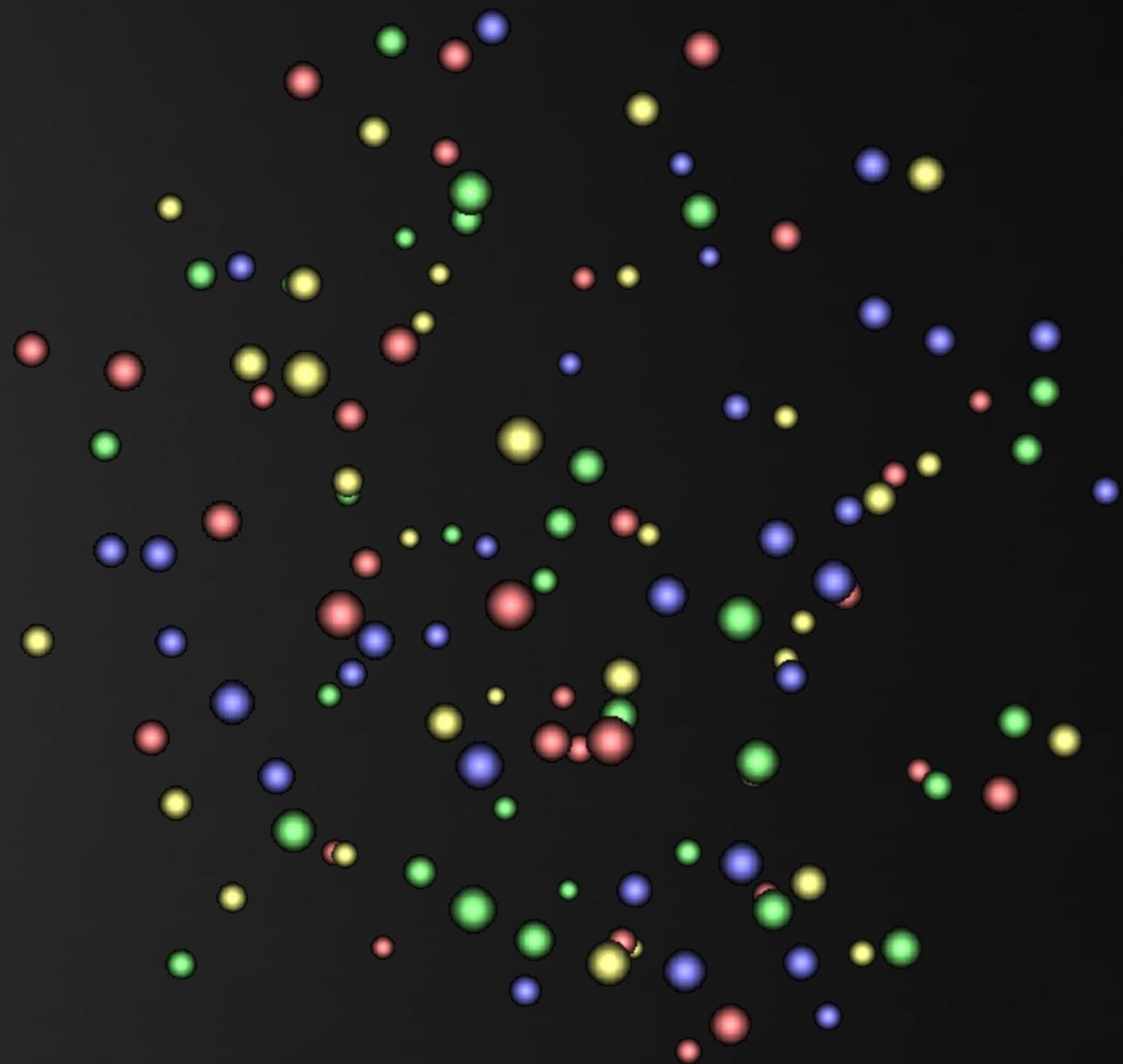


Data Locality

Example: dotcloud

```

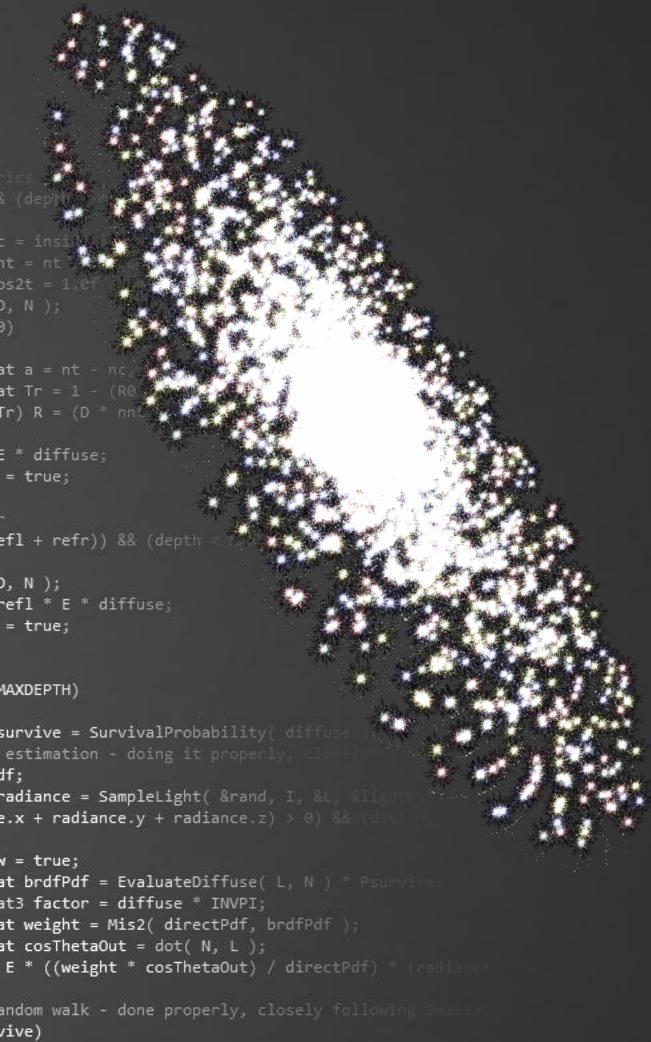
ics
& (depth < MAXDEPTH) {
    if (inside ? 1 : 1.2f * nnt) {
        nt = nt / nc; ddn = ddn * nc;
        ps2t = 1.0f - nnt * nnt;
        D, N );
    }
    if (a = nt - nc; b = nt + nc; c = nt - nc;
    at Tr = 1 - (R0 + (1 - R0) * nnt);
    Tr) R = (D * nnt - N * (ddn * nnt));
    E * diffuse;
    = true;
    -
    efl + refr)) && (depth < MAXDEPTH) {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following 3-sphere
    df;
    radiance = SampleLight( &rand, I, &L, &light, &N );
    e.x + radiance.y + radiance.z) > 0) && (ace) {
        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 3-sphere
    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 ) {
        nt = nt / nc, ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    )
    at a = nt - nc, b = nt + nc, c = 0;
    at Tr = 1 - (R0 + (1 - R0) * r);
    Tr) 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 following 3-sphere
    if;
    radiance = SampleLight( &rand, I, &L, &light, &N, &N );
    e.x + radiance.y + radiance.z) > 0) && (acc < 0.0001) {
        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.x + radiance.y + radiance.z);
    }
    random walk - done properly, closely following 3-sphere
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf, &N );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```

3. LUTs Gone Wrong





Source	Assembly	Assembly grouping: Address		
...	▲	Source	🔥 Clockticks	Instructions Retired
474		#if 0		
475		Pixel p0 = src[x0 + y0 * m_Pitch];		
476		Pixel p1 = src[x1 + y0 * m_Pitch];		
477		Pixel p2 = src[x0 + y1 * m_Pitch];		
478		Pixel p3 = src[x1 + y1 * m_Pitch];		
479		Pixel scaledp0 = ScaleColor(p0, (int)(w0 * 255.9f));		
480		Pixel scaledp1 = ScaleColor(p1, (int)(w1 * 255.9f));		
481		Pixel scaledp2 = ScaleColor(p2, (int)(w2 * 255.9f));		
482		Pixel scaledp3 = ScaleColor(p3, (int)(w3 * 255.9f));		
483		#else		
484		Pixel scaledp0 = m_Surface->m_Scaled[(int)(w0 * 255.9f)][x0	870,400,000	384,000,000
485		Pixel scaledp1 = m_Surface->m_Scaled[(int)(w1 * 255.9f)][x1	4,406,400,000	3,267,200,000
486		Pixel scaledp2 = m_Surface->m_Scaled[(int)(w2 * 255.9f)][x0		
487		Pixel scaledp3 = m_Surface->m_Scaled[(int)(w3 * 255.9f)][x1	582,400,000	521,600,000
488		#endif		
489		Pixel color = scaledp0 + scaledp1 + scaledp2 + scaledp3;	7,881,600,000	8,633,600,000
490		a_Target->GetBuffer()[i + j * a_Target->GetPitch()] = AddBl	7,468,800,000	4,371,200,000
491		}		
492		}		
493				
494		void Sprite::InitializeStartData()		
495		{		
496		for (unsigned int f = 0; f < m_NumFrames; ++f)		
497		{		
498		m_Start[f] = new unsigned int[m_Height];		
499		for (int y = 0; y < m_Height; ++y)		
500		{		
501		m_Start[f][y] = m_Width;		
502		Pixel* addr = GetBuffer() + f * m_Width + y * m_Pitch;		
503		for (int x = 0; x < m_Width; ++x)		
504		{		
505		if (addr[x])		
506		{		
507		m_Start[f][y] = x;		
508		break;		
509		}		
510		}		
511		}		
512		}		
513		}		
514				
515		Font::Font(const char *a_File, const char *a_Chars)		
516		{		
517		m_Surface = new Surface(a_File);		
518		Pixel* b = m_Surface->GetBuffer();		
519		int w = m_Surface->GetWidth();		
520		int h = m_Surface->GetHeight();		
521		unsigned int charp = 0, start = 0;		

Address ▲	Sour...	Assembly	Locators		
			rd Bound	Bad Speculation	Back-End B
					Memory Bound
0x140006d00	484	addss xmm0, xmm6	0.0%	0.0%	0.0%
0x140006d04	487	imul r10d, ecx	0.0%	0.1%	0.0%
0x140006d08	487	imul r11d, r8d	0.0%	0.1%	0.3%
0x140006d0c	487	addss xmm0, xmm4			
0x140006d10	487	subss xmm1, xmm0	0.0%	0.0%	
0x140006d14	487	movss xmm0, dword ptr [rip+0x307c]	0.0%	0.0%	0.0%
0x140006d1c	487	muls xmm4, xmm0	0.0%	0.0%	0.7%
0x140006d20	487	muls xmm6, xmm0	0.0%	0.0%	0.0%
0x140006d24	487	cvtts2si eax, xmm4			
0x140006d28	487	movssd rcx, eax	0.0%	0.0%	0.5%
0x140006d2b	487	lea eax, ptr [r10+rdi*1]	0.0%	0.0%	0.0%
0x140006d2f	487	movssd r9, eax			
0x140006d32	487	cvtts2si eax, xmm6	0.0%	0.0%	0.0%
0x140006d36	489	mov r8, qword ptr [rbx+rcx*8]	0.0%	0.0%	0.2%
0x140006d3a	489	movssd rcx, eax	0.0%	0.0%	0.0%
0x140006d3d	489	lea eax, ptr [r11+rdi*1]	0.0%	0.0%	0.0%
0x140006d41	489	mov edi, dword ptr [r8+r9*4]	0.0%	0.0%	0.3%
0x140006d45	489	movssd rdx, eax	0.0%	0.0%	19.0%
0x140006d48	489	mov rax, qword ptr [rbx+rcx*8]	0.0%	0.0%	0.2%
0x140006d4c	489	add edi, dword ptr [rax+rdx*4]	0.0%	0.0%	0.0%
0x140006d4f	485	muls xmm1, xmm0	0.0%	0.0%	12.7%
0x140006d53	485	muls xmm7, xmm0	0.0%	0.0%	0.0%
0x140006d57	487	cvtts2si eax, xmm1			
0x140006d5b	487	xorps xmm1, xmm1	0.0%	0.0%	0.5%
0x140006d5e	487	movssd rcx, eax			
0x140006d61	487	lea eax, ptr [r10+rsi*1]	0.0%	0.0%	0.0%
0x140006d65	490	mov r10, qword ptr [r13]	0.0%	0.0%	
0x140006d69	490	movssd rdx, eax	0.0%	0.0%	0.3%
0x140006d6c	490	mov rax, qword ptr [rbx+rcx*8]	0.0%	0.0%	0.0%
0x140006d70	490	add edi, dword ptr [rax+rdx*4]	0.0%	0.0%	0.0%
0x140006d73	490	cvtts2si eax, xmm7	0.0%	0.8%	9.4%
0x140006d77	490	movssd rcx, eax	0.0%	0.1%	0.3%
0x140006d7a	490	lea eax, ptr [r11+rsi*1]			
0x140006d7e	490	movssd rdx, eax	0.0%	0.0%	0.0%
0x140006d81	490	mov rax, qword ptr [rbx+rcx*8]	0.0%	0.0%	0.3%
0x140006d85	490	add edi, dword ptr [rax+rdx*4]			
0x140006d88	490	mov eax, dword ptr [r13+0x10]	0.0%	0.2%	6.3%
0x140006d8c	490	imul eax, r15d	0.0%	0.1%	0.0%
0x140006d90	490	inc r15d	0.0%	0.0%	0.2%
0x140006d93	490	add eax, ebp	0.0%	0.0%	0.0%
0x140006d95	490	movssd r11, eax	0.0%	0.0%	0.2%
0x140006d98	490	mov eax, edi	0.0%	0.0%	0.0%
0x140006d9a	490	and eax, 0xff0000	0.0%	0.0%	0.0%
0x140006d9f	490	mov ecx, dword ptr [r10+r11*4]			
0x140006da3	490	mov r8d, ecx	0.0%	0.0%	0.0%
0x140006da6	490	mov r9d, ecx	0.0%	0.0%	0.2%



/INFOMOV/

End of Practical

