SIMD Ray Tracing Tips

Toshiya Hachisuka

SIMD

- Single Instruction Multiple Data
- Perform the same operation on multiple data at a time

- Example: addition of vectors

  - Non-SIMD

    ```
    c.x = a.x + b.x; c.y = a.y + b.y;
    c.z = a.z + b.z; c.w = a.w + b.w;
    ```

  - SIMD

    ```
    c = a + b
    ```

SIMD

- One of the keys to achieve high performance in ray tracing
- Mostly 4-way SIMD
  - Can be x4 faster (but difficult to achieve)
  - SSE (x86), Altivec (PowerPC)
- Only basic operations
  - Add, Subtract, Multiply etc.
  - No Dot, Cross
- ...and it is fun to code!

AoS and SoA

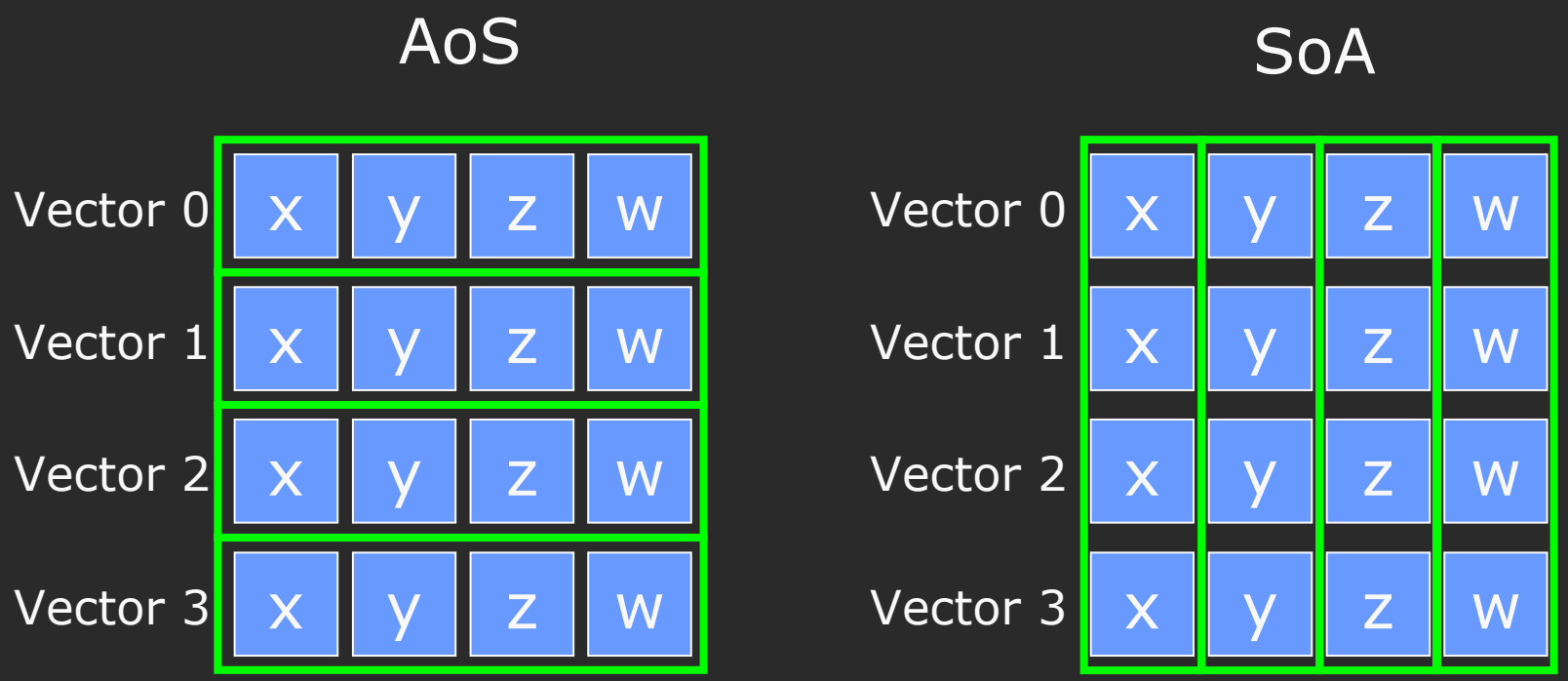
- Array of Structure
- Each vector is a SIMD variable

- Vector 0 (V[0])
  - x, y, z, w
- Vector 1 (V[1])
  - x, y, z, w
- Vector 2 (V[2])
  - x, y, z, w
- Vector 3 (V[3])
  - x, y, z, w

- Structure of Array
- Each array of elements of 4 vectors is a SIMD variable

- X elements (vx)
  - vx[0], vx[1], vx[2], vx[3]
- Y elements (vy)
  - vy[0], vy[1], vy[2], vy[3]
- Z elements (vz)
  - vz[0], vz[1], vz[2], vz[3]
- W elements (vw)
  - vw[0], vw[1], vw[2], vw[3]

AoS and SoA



single SIMD variable

AoS and SoA

- Pitfall1:
  Well… since AoS (a single SIMD variable as a single vector) sounds natural, let's use it
- Actually, you will prefer SoA in the end
  - vx[4], vy[4], vz[4], vw[4]
- We will see the reason very soon

Using SIMD Efficiently

- Pitfall2:
  Replacing a Vector class by SIMD variable. For instance, use SIMD add in vector addition.

- It is not efficient, beecause of additional copies

- Use SIMD locally
  - e.g., write SIMD ray-triangle intersection

- Always try to use all 4-elements in SIMD
  - AoS is wasteful for 3-elements vectors

SIMD Ray(s)-Triangle(s) Intersection(s)

- 2 Ways to utilize SIMD

- 4 Rays – Single Triangle Intersection
  - Need to bundle 4 rays
  - Not trivial, but the state of the art

- Single Ray – 4 Triangles Intersection
  - Need to bundle 4 triangles
  - Trivial (just use 4 triangles in a leaf node of BVH)

Dot Product in SIMD

- Remember:
  - a.x * b.x + a.y * b.y + a.z * b.z
- In AoS:
  - vd0 = a0 * b0; vd1 = a1 * b1;
    vd2 = a2 * b2; vd3 = a3 * b3;
  - dots[0] = vd0.x + vd0.y + vd0.z
    dots[1] = vd1.x + vd1.y + vd1.z
    dots[2] = vd2.x + vd2.y + vd2.z
    dots[3] = vd3.x + vd3.y + vd3.z
- In SoA:
  - dots = ax * bx + ay * by + az * bz
  - Note that ax = (a0.x, a1.x, a2.x, a3.x)

Cross Product in SIMD

- Remember:
  - (ay*bz – az*by, az*bx – ax*bz, ax*by – ay*bx)
- In AoS:
  - $!#)%#$’!)%’#!
  - Need shuffle operations (high latency)
  - (ax, ay, az, aw) ⟶ (ay, az, ax, aw) etc.
- In SoA:
  - CrossX = ay*bz – az*by;
  - CrossY = az*bx – ax*bz;
  - CrossZ = ax*by – ay*bx;
  - Straightforward

Branching in SIMD

- No branch instructions
- Solution: Bit operations

- Example:
  - c = (min(a.x,b.x), ….)

- In SIMD:
  - isAltB = (a < b) // returns (a.x < b.x, …)
  - c = (a & isAltB) + (b & ~isAltB)

Branching in SIMD

- Example:
  - a = b + c;
  - d = e + f;
  - If (a > 0) {d = a * d};
- In SIMD:
  - a = b + c;
  - d = e + f;
  - isAgt0 = (a > 0)
  - d = ((a * d) & isAgt0) + (d & ~isAgt0)
  - Note that we compute a * d always

Practical (Annoying) Issues

- In general, all addresses of SIMD data should be 16-byte aligned
  - Address should be 0x1234560
  - Usually they are not
- Solution depends on implementation
- Search the Internet
  - It is FAQ

Summary

- SIMD is the key for high performance
- Use SoA (structure of array), not AoS
  - vx[4], vy[4], vz[4], vw[4]
  - Simply replacing Vector by SIMD is not efficient
- Use masking instead of branching
- Do single ray-4 triangles intersections
- Be aware of the address alignment

More on SIMD

- Is SIMD useful only for ray-triangle intersection?
- Ray-triangle intersection is not the only one

- Use SIMD for the ray traversal of BVH
  - Obvious idea: 4-ary BVH instead of binary BVH
  - Each node has 4 child nodes
  - Two levels of splits to construct 4 child nodes
  - 4 ray-AABBs intersections at a time

- … and many more (shading, BVH construction etc)
  Use wisely!

Useful Resources

- The article about SIMD ray tracing by Intel
  - http://softwarecommunity.intel.com/articles/eng/2658.htm
  - Contains SIMD traversal as well (4 rays)

- "Optimizing Ray-Triangle Intersection via Automated Search" by Kensler and Shirley
  - http://www.cs.utah.edu/~aek/research/triangle.pdf
  - Perhaps the fastest SIMD ray-triangle intersection code

- "Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays" by Dammertz et al.
  - http://www.uniulm.de/fileadmin/website_uni_ulm/iui.inst.100/institut/Papers/QBVH.pdf
  - Latest implementation of 4-ary BVH