

DTU



30/11/23

# Real-Time Visual and Machine Learning Systems

# Types

- GPU Hand-in
  - Any Questions?
  - Common problems?
    - Bind groups and unused elements
- Analysis Hand-in
  - Any Questions?
- Projects
  - Any Questions?

# Types

Integers and floats

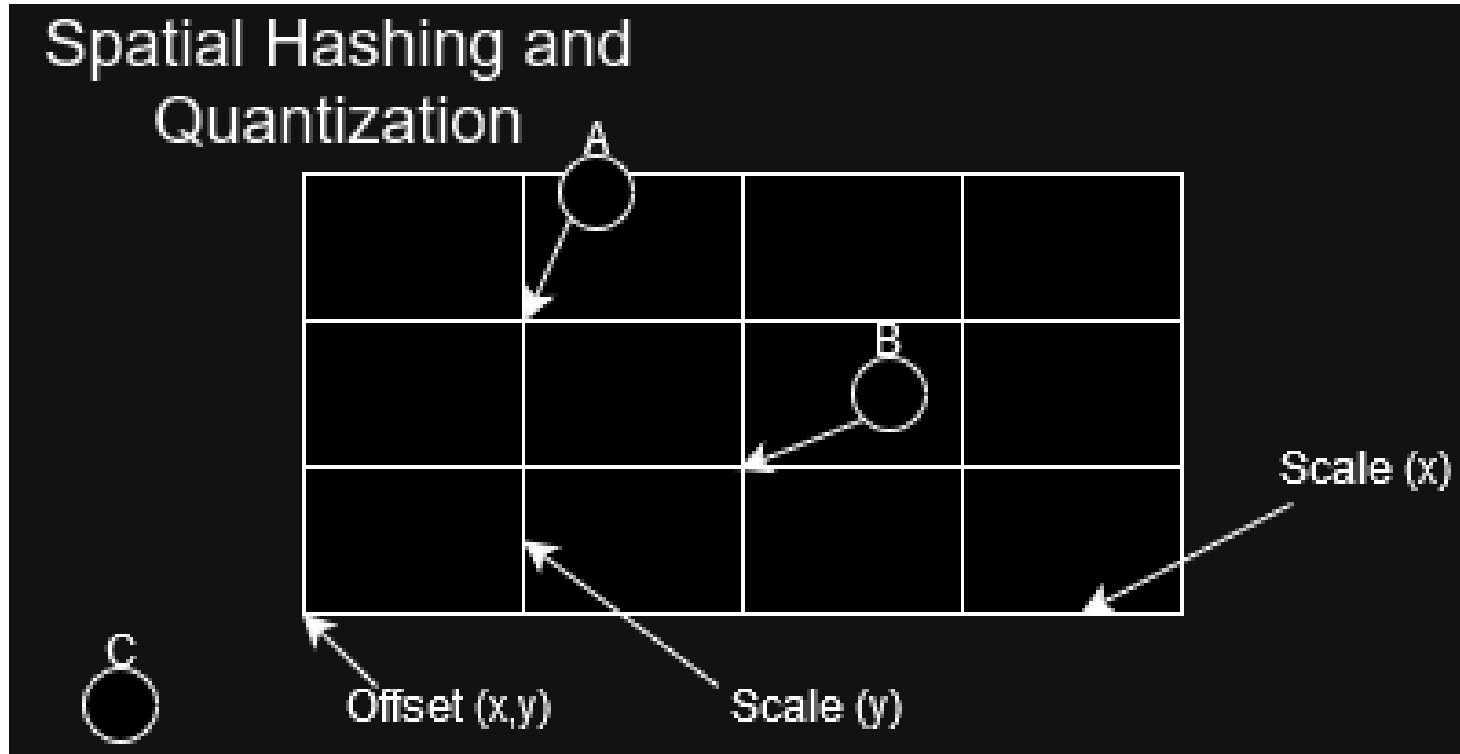
You know what these are, otherwise, read the text

# Types - Precision

- Floating point numbers – small to large and sorting
- Pairwise additions
- Every calculation results in an error
- Fused multiplication-addition results in 1 error instead of 2
- It is almost impossible for two floats to be exactly equal

# Types - Precision

- We can quantize floats to integers
- Integers compress much better than floats and can be sorted in an exciting number of ways
- Integers allow us to do bit tricks! Yay!



# Types - Precision

- Spatial hashing is basically quantization, some shifts and some additions

```
// This would be given for this specific grid
// If we aren't too bothered by the loss of precision we can
// multiply by the inverse of the scaling term.
let x_scale: f32 = 5.0f;
let x_offset: f32 = 0.32f;

let y_scale: f32 = 4.0f;
let y_offset: f32 = 3.50f;

let z_scale: f32 = 2.0f;
let z_offset: f32 = 0.42f;

let x: f32 = 7.2f;
let y: f32 = 5.5f;
let z: f32 = 2.2f;

let x_quantized: u64 = ((x - x_offset) / x_scale) as u64;
let y_quantized: u64 = ((y - y_offset) / y_scale) as u64;
let z_quantized: u64 = ((z - z_offset) / z_scale) as u64;

let x_hash: u64 = (x_quantized & 0xFFFFF) << 40;
let y_hash: u64 = (y_quantized & 0xFFFFF) << 20;
let z_hash: u64 = z_quantized & 0xFFFFF;

let hash: u64 = x_hash | y_hash | z_hash;
```

# Types - Sorting

- `.sort()` and floats/ordering in Rust
- Quantizing floats to integers
- Sorting usually also has performance implications and is usually a good idea in preprocessing



# Types – Morton Codes

- An encoding, not a sorting method, but we can change the way numbers are sorted by encoding them differently and shuffling bits around

# Types – Sorting – Morton

		x		0		1	
		00	01	10	11		
y		00	0000	0001	0100	0101	
		01	0010	0011	0110	0111	
1	10	1000	1001	1100	1101		
	11	1010	1011	1110	1111		

(a)

node	bits	index
-----:	:-----:	:-----
0,0,0	000	0
0,0,1	001	1
0,1,0	010	2
0,1,1	011	3
1,0,0	100	4
1,0,1	101	5
1,1,0	110	6
1,1,1	111	7

(b)

# Types – Sorting – Morton

```
let x: u32 = 8;
let y: u32 = 1;
let z: u32 = 2;

let component_count: u32 = 3;
let bits_of_precision: u32 = 10;

let mut encoded: u32 = 0;
let mut mask: u32 = 1;
for index in 0..bits_of_precision {
    encoded |= ((x & (1 << index)) << component_count * index);
    encoded |= ((y & (1 << index)) << (component_count * index + 1));
    encoded |= ((z & (1 << index)) << (component_count * index + 2));
}
```

# Types – Sorting – Morton

		x		0		1	
		y		00	01	10	11
0	00	0	0	0000	0001	0100	0101
	01			0010	0011	0110	0111
1	10	1	0	1000	1001	1100	1101
	11			1010	1011	1110	1111

(a)

node	bits	index
-----:	:-----:	:-----
0,0,0	000	0
0,0,1	001	1
0,1,0	010	2
0,1,1	011	3
1,0,0	100	4
1,0,1	101	5
1,1,0	110	6
1,1,1	111	7

(b)

# Types – Sorting – Radix

- Radix sort!
- There are many ways to sort, but Radix has known GPU implementations to do so in a single shader dispatch

1	2	1
0	0	1
4	3	2
0	2	3
5	6	4
0	4	5
7	8	8

0	0	1
1	2	1
0	2	3
4	3	2
0	4	5
5	6	4
7	8	8

0	0	1
0	2	3
0	4	5
1	2	1
4	3	2
5	6	4
7	8	8

sorting the integers according to units, tens and hundreds place digits

# Types - Compression

- Optimize your data for a more advanced algorithm to do the heavy lifting
- Avoid floats and strings if you can, especially floats represented as strings
- Point cloud compression case:
  - Hierarchical data structure and spatial hashing
  - Quantification (16-bits)
  - Bit nulling
  - Sorting
  - Delta encoding
  - Gzip
  - Transfer
  - Un-gzip
  - Transfer to gpu
  - Dequantify in shader

# Types – Energy Usage

**Table VI** ENERGY CONSUMPTION OF ARITHMETIC OPERATIONS ON THE INTEL XEON X5670 TEST SYSTEM

<b>Workload</b>	<b>operations per 16 Byte</b>	<b><i>biop</i></b>	<b><i>E<sub>calc</sub></i></b>
add pi	2 (64 Bit)	8 Byte/op	428 pJ/op
mul_pi	2 (64 Bit)	8 Byte/op	476 pJ/op
add_pd	2 (64 Bit)	8 Byte/op	319 pJ/op
mul_pd	2 (64 Bit)	8 Byte/op	387 pJ/op
mul+add _pd	4 (64 Bit)	4 Byte/op	464 pJ/op
add ps	4 (32 Bit)	4 Byte/op	111 pJ/op
mul_ps	4 (32 Bit)	4 Byte/op	164 pJ/op

# Types – Energy Usage

**Table V** COMPARISON OF THE ENERGY CONSUMPTION OF DATA TRANSFERS (movAP s), AMD OPTERON 2435 AND INTEL XEON X5670

Location	$E_{t,.}$ AMD	$E_{trans}$ Intel	Factor
Li	105 pJ/Byte	64 pJ/Byte	1.6
L2	357 pJ/Byte	121 pJ/Byte	2.9
L3	654 pJ/Byte	254 pJ/Byte	2.6
RAM	3590 pJ/Byte	1250 pJ/Byte	2.8



# Profilers

- Timing
  - System monitoring (Task manager, htop, nvidia-smi)
  - General profilers (Visual Studio, perf, tracy)
  - System specific (AMD, Intel VTune)
  - GPU specific (nsight-compute, Renderdoc, AMD)
  - App specific (PyTorch profiler)
- 
- Disk is hard to profile, system monitors are likely the way to go

Too many profilers to go into too much detail, do the [exercise](#)!