ACAT 2022

COVFIE: a compositional library for heterogeneous vector fields

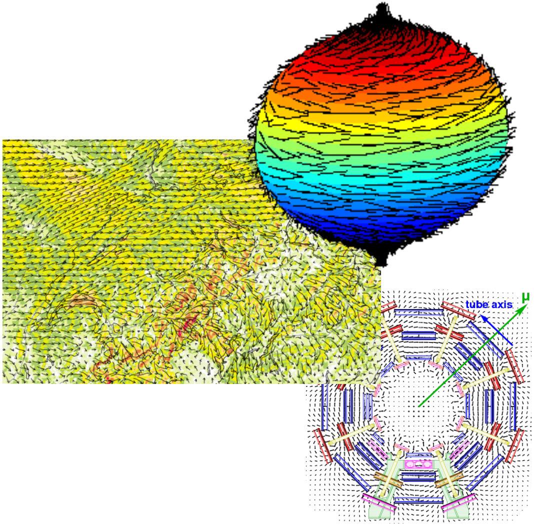
Stephen Nicholas Swatman[1,2], Andreas Salzburger[2], Attila Krasznahorkay[2]

Wednesday, October 26, 2022

[1]University of Amsterdam [2]CERN

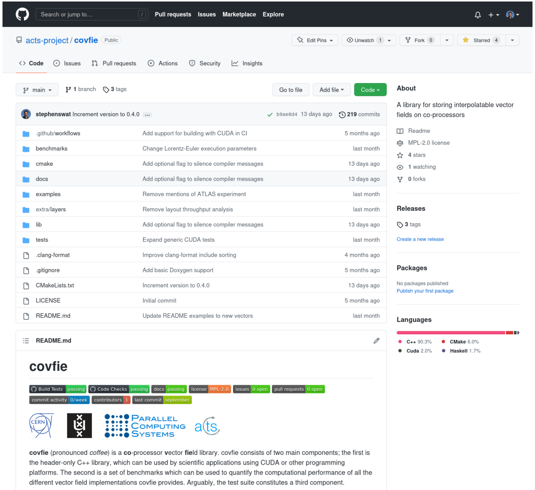
Introduction

- Vector fields are used everywhere
  - Oceanography
  - Atmospheric science
  - High-energy physics
- Different functional and non-functional properties
- Many non-unified different implementations of fields



Introduction

- Enter COVFIE, the **co**mpositional **v**ector **fie**ld library
- Goal is to...
  - support arbitrary vector fields...
  - across many devices...
  - with high performance...
  - in a simple way...
  - through composition.



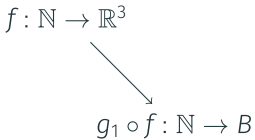
Composition

- Let's consider the simplest vector field *f*: $\mathbb{R}^3$ in memory

$$f : \mathbb{N} \to \mathbb{R}^3$$

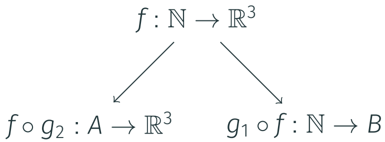
Composition

- Let's consider the simplest vector field f: $\mathbb{R}^3$ in memory
- We can turn this into an $\mathbb{N} \to B$ field by covariantly composing a function $g_1 : \mathbb{R}^3 \to B$...



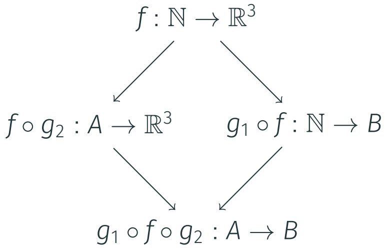
Composition

- Let's consider the simplest vector field f: $\mathbb{R}^3$ in memory
- We can turn this into an $\mathbb{N} \to B$ field by covariantly composing a function $g_1 : \mathbb{R}^3 \to B$...
- ...into an $A \to \mathbb{R}^3$ field by contravariantly composing a function $g_2 : A \to \mathbb{R}^3$



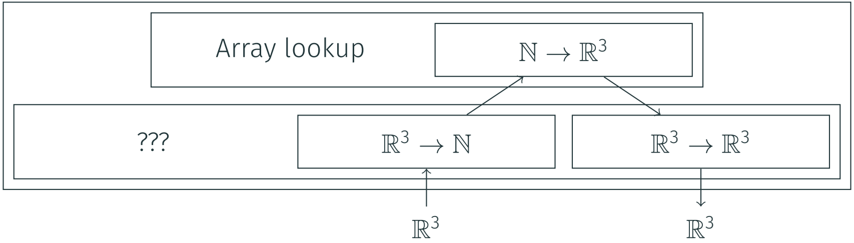
Composition

- Let's consider the simplest vector field f: $\mathbb{R}^3$ in memory
- We can turn this into an $\mathbb{N} \to B$ field by covariantly composing a function $g_1 : \mathbb{R}^3 \to B$...
- ...into an $A \to \mathbb{R}^3$ field by contravariantly composing a function $g_2 : A \to \mathbb{R}^3$
- *Transformer* $t = \langle g_1, g_2 \rangle$ lets us build any field



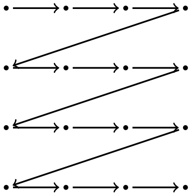
Example: ATLAS

- Let's consider the ATLAS magnetic field, which we can build using:
  - A primitive storage backend $\mathbb{N} \to \mathbb{R}^3$: our memory
  - ...and a transformer $\langle \mathbb{R}^3 \to \mathbb{N}, \mathbb{R}^3 \to \mathbb{R}^3 \rangle$
- But this is too complex, let's decompose!

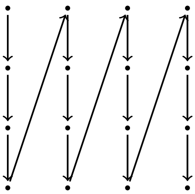


3D Layouts

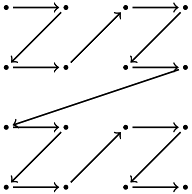
- Primitive backends are intentionally simple
- Add functional and non-functional properties on a need-to-have basis
- Functionally, we need three-dimensional access
- Non-functionally, layouts matter (caching!)



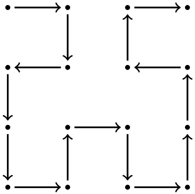
Row-major order



Column-major order



Morton order



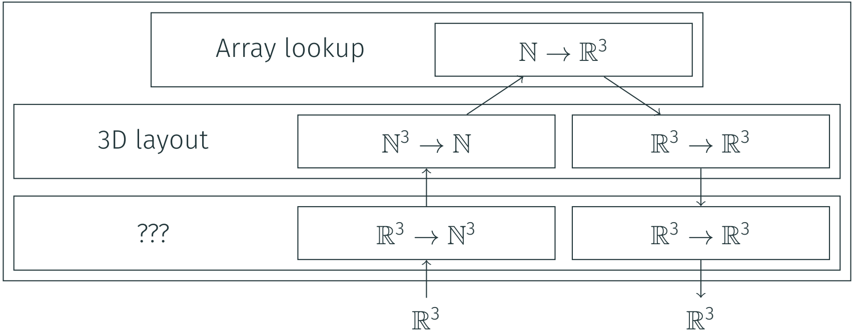
Hilbert order

3D Layouts

- Let’s pick a Morton curve and compose it with our primitive backend:

```
using field_t = covfie::layout::morton<
        ulong2,
        covfie::storage::array<float2>
>;
```

Example: ATLAS



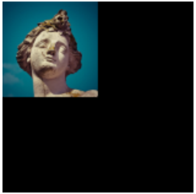
Bounds checking

- LDMX talk yesterday: behaviour at B-field bounds is important
- Borrow schemes from computer graphics:
  - Repeat
  - Mirror
  - Clamp
  - Default

Repeat

Mirror

Clamp

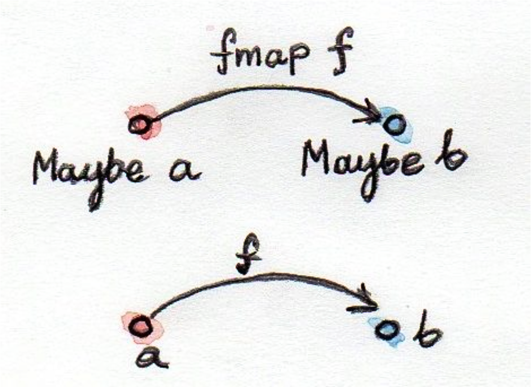


Default

Source: Julien Delezenne

Bounds Checking

- Let's return the zero vector $\vec{0}$ for our of bounds errors
- Problem: out-of-bounds-ness is only known contravariantly...
- ...and we need this information covariantly
- How do we pass this information around?
- Wrap our contravariant output and covariant input in an optional functor $F(a) = a + \mathbb{1}$



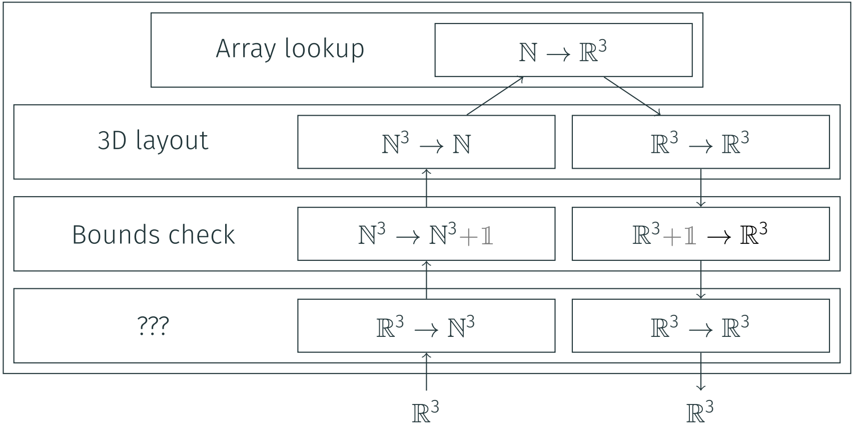
Source: Bartosz Milewski

Bounds Checking

- Let's compose a bounds checking transformer:

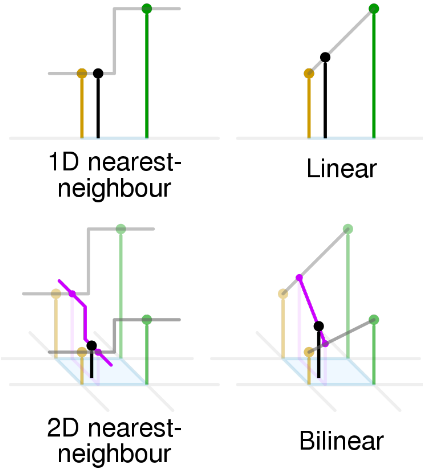
```
using field_t = covfie::boundary::def<
    covfie::layout::morton<
        ulong2,
        covfie::storage::array<float2>
    >
>;
```

Example: ATLAS



Interpolation

- Our field is currently indexed using natural numbers: not much use for a B-field
- Need an interpolation method to go from real coordinates to natural
  - Nearest neighbour
  - *n*-linear



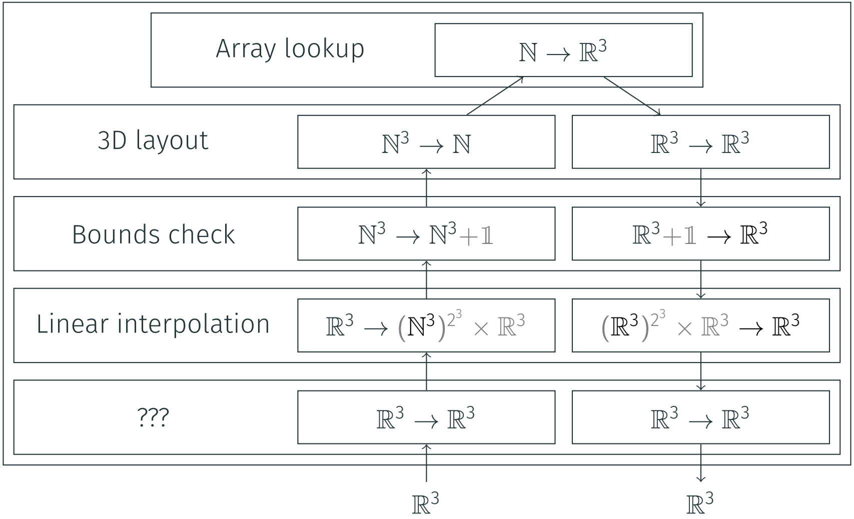
Source: Cmglee

Interpolation

- Let's add a trilinear interpolator to our vector field:

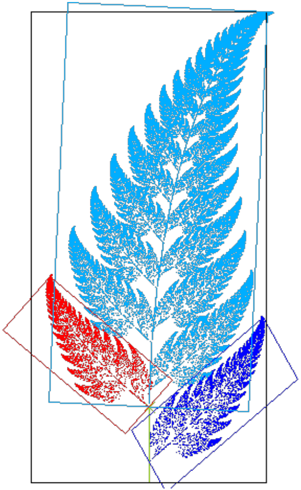
```
using field_t = covfie::interpolator::linear<
    covfie::boundary::def<
        covfie::layout::morton<
            ulong2,
            covfie::storage::array<float2>
        >
    >
>;
```

Example: ATLAS



Transformations

- Right now, coordinate $(0, 0, 0)$ is at the top left corner of the field…
- …not in the center of the detector
- No problem: just compose an affine transformation!
- Takes arbitrary $(n + 1) \times (n + 1)$ transformation matrix

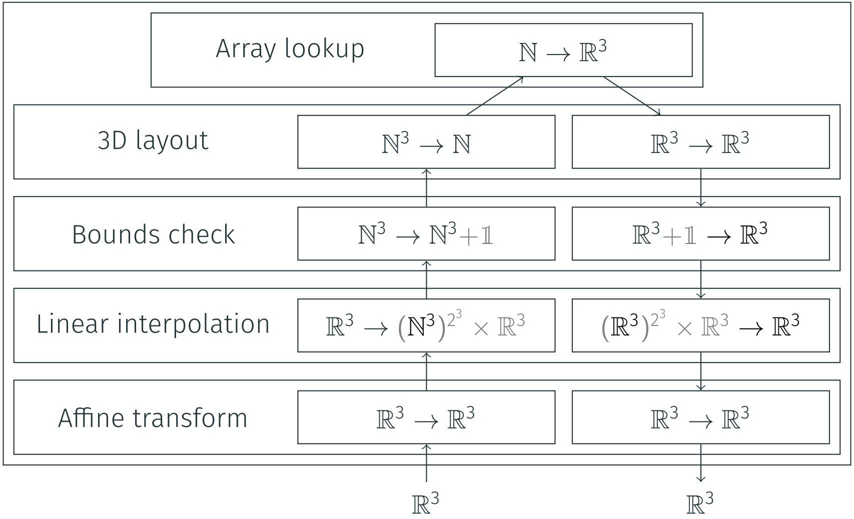


Source: António Miguel de Campos

Interpolation

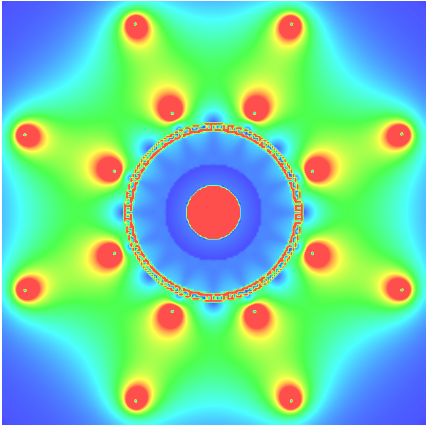
- Thereby we arrive at our final type:

```
using field_t = covfie::transformation::affine<
    covfie::interpolator::linear<
        covfie::boundary::def<
            covfie::layout::morton<
                ulong2,
                covfie::storage::array<float2>
            >
        >
    >
>;
```



Array lookup
$\mathbb{N} \to \mathbb{R}^3$
3D layout
$\mathbb{N}^3 \to \mathbb{N}$
$\mathbb{R}^3 \to \mathbb{R}^3$
Bounds check
$\mathbb{N}^3 \to \mathbb{N}^3 + \mathbb{1}$
$\mathbb{R}^3 + \mathbb{1} \to \mathbb{R}^3$
Linear interpolation
$\mathbb{R}^3 \to (\mathbb{N}^3)^{2^3} \times \mathbb{R}^3$
$(\mathbb{R}^3)^{2^3} \times \mathbb{R}^3 \to \mathbb{R}^3$
Affine transform
$\mathbb{R}^3 \to \mathbb{R}^3$
$\mathbb{R}^3 \to \mathbb{R}^3$
$\mathbb{R}^3$
$\mathbb{R}^3$

```cpp
field_t::view_t fv(f);

for (std::size_t x = 0; x < im_size[1]; ++x) {
    for (std::size_t y = 0; y < im_size[0]; ++y) {
        field_t::view_t::output_t p = fv.at(x, y);

        img[im_size[1] * y + x] = static_cast<char>(
            255.f *
            std::min(std::sqrt(mag(p), 1.0f)
        );
    }
}
```



List of Primitive Backends

Name	Platform	Field type
ARRAY	CPU	$\prod_{T:\mathcal{V}} \mathbb{N} \to T$
CUDAARRAY	CUDA	$\prod_{T:\mathcal{V}} \mathbb{N} \to T$
CUDAPITCH	CUDA	$\prod_{d:[\![1,3]\!]} \prod_{T:\mathcal{V}} \mathbb{N}^d \to T$
CUDATEX	CUDA	$\prod_{d:[\![1,3]\!]} \prod_{d':[\![1,4]\!]} \mathbb{R}^d \to \mathbb{R}^{d'}$
ANALYTIC	CPU	$\prod_{S,T:\mathcal{V}} S \to T$
CONSTANT	Any	$\prod_{S,T:\mathcal{V}} S \to T$

List of Backend Transformers

Name	Type
Pitched	$\prod_{n:\mathbb{N}} \prod_{T\ \mathtt{Type}} (\mathbb{N}^n \to \mathbb{N}) \times (T \to T)$
Morton	$\prod_{n:\mathbb{N}} \prod_{T\ \mathtt{Type}} (\mathbb{N}^n \to \mathbb{N}) \times (T \to T)$
Hilbert	$\prod_{T\ \mathtt{Type}} (\mathbb{N}^2 \to \mathbb{N}) \times (T \to T)$
Shuffle	$\prod_{n:\mathbb{N}} \prod_{p:\mathfrak{S}_n} \prod_{S,T\ \mathtt{Type}} (S^n \to S^n) \times (T \to T)$
Default	$\prod_{S,T\ \mathtt{Type}} (S \to S + \mathbb{1}) \times (T + \mathbb{1} \to T)$
Wrap	$\prod_{S,T\ \mathtt{Type}} (S \to S) \times (T \to T)$
Nearest	$\prod_{n:\mathbb{N}} \prod_{T\ \mathtt{Type}} (\mathbb{R}^n \to \mathbb{N}^n) \times (T \to T)$
Linear	$\prod_{n:\mathbb{N}} (\mathbb{R}^n \to (\mathbb{N}^n)^{2^n} \times \mathbb{R}^n) \times ((\mathbb{R}^n)^{2^n} \times \mathbb{R}^3 \to \mathbb{R}^n)$
Affine	$\prod_{n:\mathbb{N}} \prod_{T\ \mathtt{Type}} (\mathbb{R}^n \to \mathbb{R}^n) \times (T \to T)$

Moving this to a GPU

CPU

```
using field_t = covfie::interpolator::linear<
    covfie::boundary::def<
        covfie::layout::morton<
            ulong2,
            covfie::storage::array<
                float2
            >
        >
    >
>;
```

GPU

```
using field_t = covfie::interpolator::linear<
    covfie::boundary::def<
        covfie::layout::morton<
            ulong2,
            covfie::storage::cuda::array<
                float2
            >
        >
    >
>;
```

- Type conversion includes movement of the data to or from the GPU

Rendering on a GPU

- Field types are convertible
- Simply put one into your CUDA kernel: no hassle!

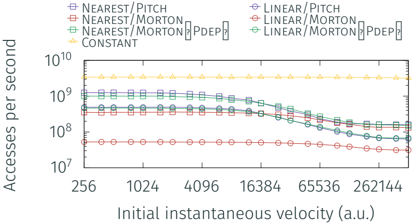
```
__global__ void render(
    typename field_t::view_t vf, char * out,
    uint width, uint height, float z
) {
    int x = blockDim.x * blockIdx.x + threadIdx.x;
    int y = blockDim.y * blockIdx.y + threadIdx.y;

    if (x < width && y < height) {
        float fx = x / static_cast<float>(width);
        float fy = y / static_cast<float>(height);

        typename field_t::output_t p = vf.at(fx, fy, z);
        out[height * x + y] = static_cast<char>(
            255.f *
            std::min(mag(p), 1.0f)
        );
    }
}
```

Texture “Memory”

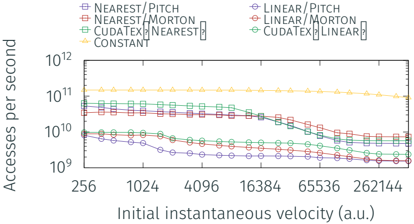
- Primitive backends need not be as simple as an array
- GPUs support special texture “memory”
  - It’s just global memory with storage layouts, separate caching, and hardware acceleration
- In some cases, may provide performance benefits
- Entirely encapsulated as primitive COVFIE backend

Benchmarking

- covfie also comes with a benchmark suite
- Set of mini-apps with different access patterns
- Includes RK1 and RK4 propagation of agents
- Fun results for depth-first RK4 propagation of 65 536 agents



Intel Xeon E5-2630 v3 (2 sockets, 32 threads)



NVIDIA A100

Conclusions

- COVFIE is available right now to try!
- Header-only, documented, and easy to use
- *https://github.com/acts-project/covfie*
- Please get in contact stephen.nicholas.swatman@cern.ch!