

COVFIE: Generalised Compositional Vector Fields

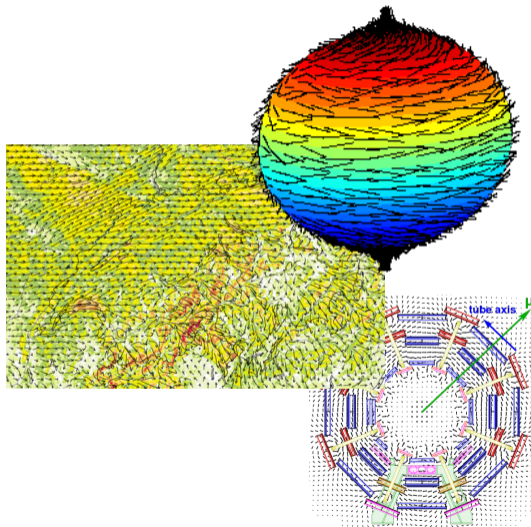
Stephen Nicholas Swatman^{1,2}

Tuesday, September 27, 2022

¹University of Amsterdam ²CERN

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 COVFILE, the **compositional vector field** library
- Goal is to...
 - support arbitrary vector fields...
 - across many devices...
 - with high performance...
 - in a simple way...
 - through composition.

The screenshot shows the GitHub repository page for 'acts-project/covfile'. The repository is public and has 219 commits. The commit history shows a recent update to version 0.4.0. The file list includes folders like 'glib/workflows', 'benchmarks', 'cmake', 'docs', 'examples', 'extra/layers', 'lib', 'tests', and files like '.clang-format', 'glibignore', 'CMakeLists.txt', 'LICENSE', and 'README.md'. The README.md file is open, showing the repository name 'covfile' and a progress bar for supported languages: C++ (30.3%), CMake (6.0%), Code (2.0%), and Haskell (1.7%). Logos for 'PARALLEL COMPUTING SYSTEMS' and 'ats' are visible. The README text describes 'covfile' as a co-processor vector field library consisting of a header-only C++ library, benchmarks, and a test suite.

Composition

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

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

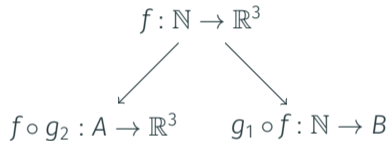
Composition

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

$$\begin{array}{ccc} f : \mathbb{N} \rightarrow \mathbb{R}^3 & & \\ & \searrow & \\ & & g_1 \circ f : \mathbb{N} \rightarrow B \end{array}$$

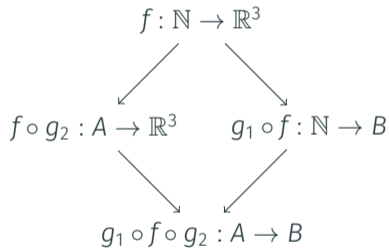
Composition

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



Composition

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



Composition

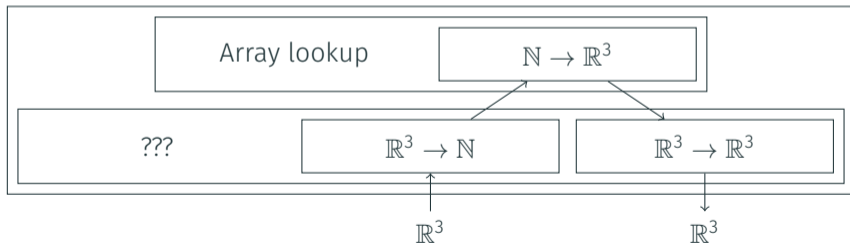
- Let's consider the simplest vector field f : \mathbb{R}^3 in memory
- We can turn this into an $\mathbb{N} \rightarrow B$ field by covariantly composing a function $g_1 : \mathbb{R}^3 \rightarrow B...$
- ...into an $A \rightarrow \mathbb{R}^3$ field by contravariantly composing a function $g_2 : A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



Source: Dario Sanches

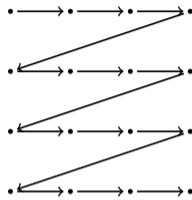
Example: ATLAS

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

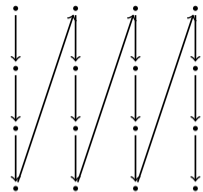


3D Layouts

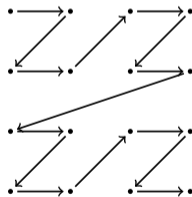
- Primitive backends are intentionally simple
- Add functional and non-functional properties on a need-to-have basis
- Functionally, we need three-dimensional access
- Non-functionally, layouts matter (caching!)



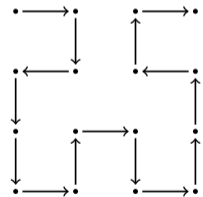
Row-major order



Column-major order



Morton order

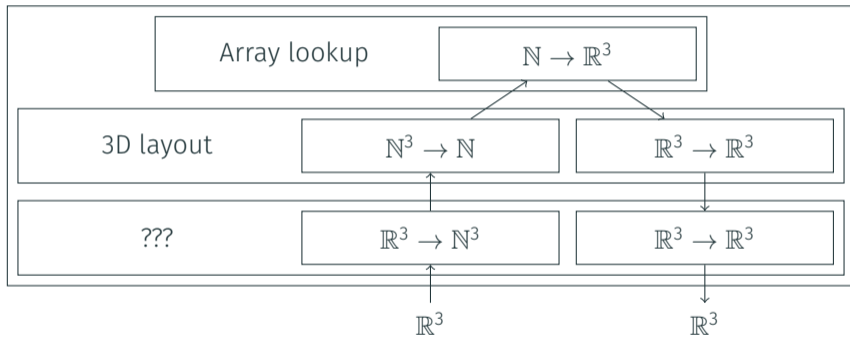


Hilbert order

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

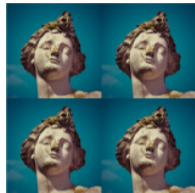
```
1 using field_t = covfie::layout::morton<  
2     ulong2 ,  
3     covfie::storage::array<float2 >  
4 >;
```

Example: ATLAS



Bounds checking

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



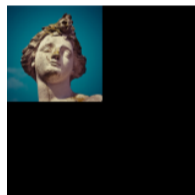
Repeat



Mirror



Clamp

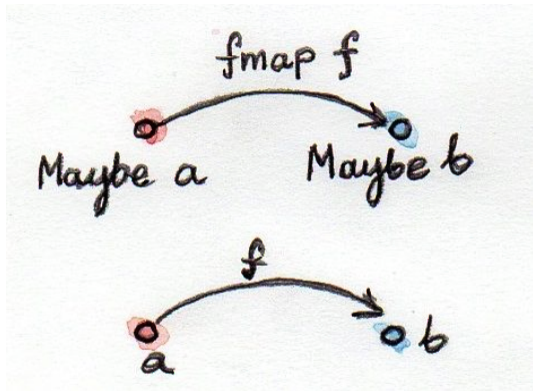


Default

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 $F(a) = a + \mathbb{1}$

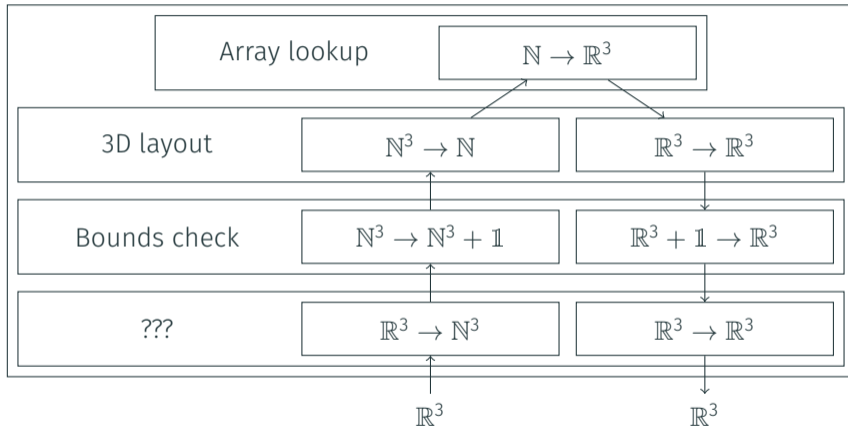


Source: Bartosz Milewski

- Let's compose a bounds checking transformer:

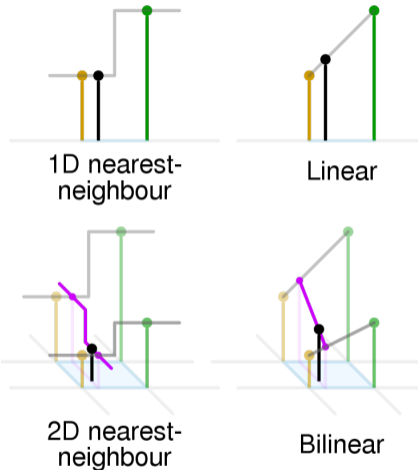
```
1 using field_t = covfie::boundary::def<  
2     covfie::layout::morton<  
3         ulong2,  
4         covfie::storage::array<float2>  
5     >  
6 >;
```

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

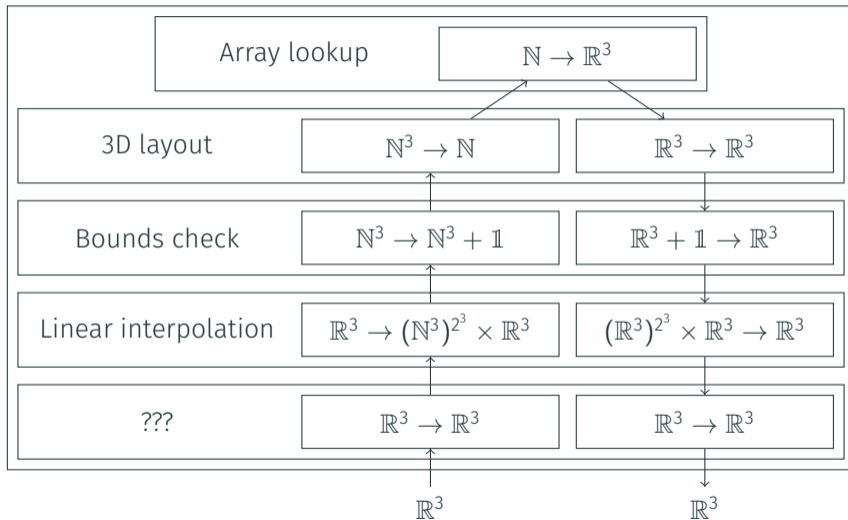


Source: Cmglee

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

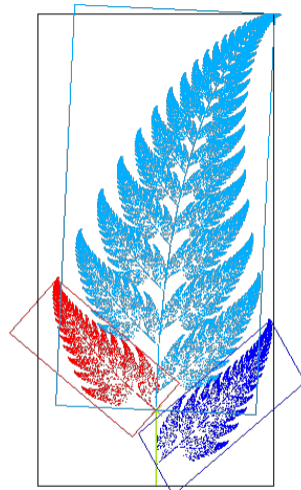
```
1 using field_t = covfie::interpolator::linear <
2     covfie::boundary::def <
3         covfie::layout::morton <
4             ulong2 ,
5             covfie::storage::array <float2 >
6         >
7     >
8 >;
```

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



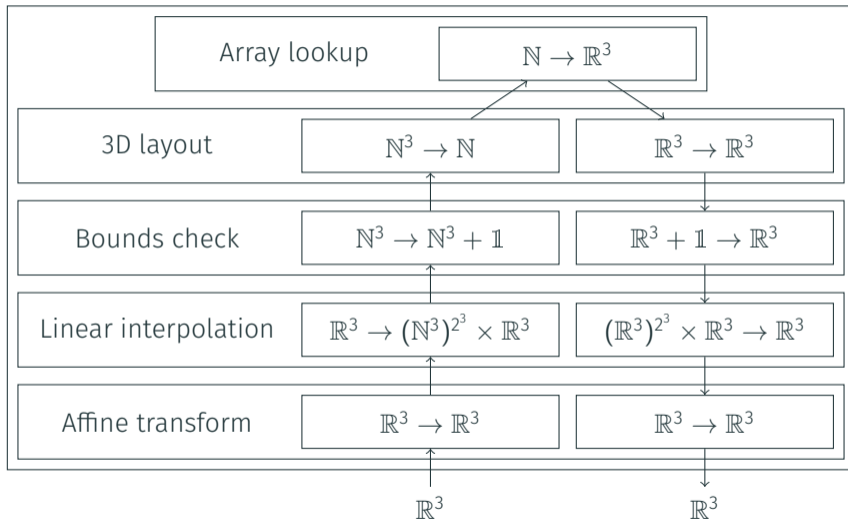
Source: Antônio Miguel de Campos

Interpolation

- Thereby we arrive at our final type:

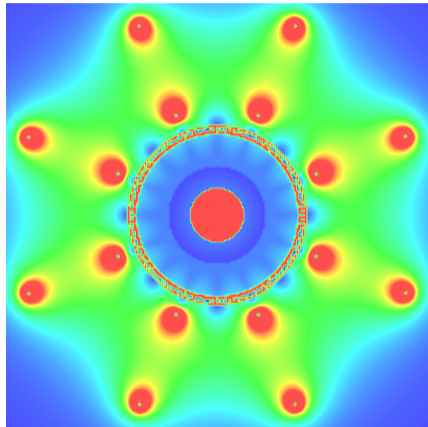
```
1 using field_t = covfie::transformation::affine <
2     covfie::interpolator::linear <
3         covfie::boundary::def<
4             covfie::layout::morton<
5                 ulong2,
6                 covfie::storage::array<float2>
7             >
8         >
9     >
10 >;
```

Example: ATLAS



Rendering

```
1 field_t::view_t fv(f);
2
3 for (std::size_t x = 0; x < im_size[1]; ++x) {
4     for (std::size_t y = 0; y < im_size[0]; ++y) {
5         field_t::view_t::output_t p = fv.at(x, y);
6
7         img[im_size[1] * y + x] = static_cast<char>(
8             255.f *
9             std::min(std::sqrt(mag(p)), 1.0f)
10        );
11     }
12 }
```



List of Primitive Backends

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

List of Backend Transformers

Name	Type
PITCHED	$\prod_{n:\mathbb{N}} \prod_T \text{Type}(\mathbb{N}^n \rightarrow \mathbb{N}) \times (T \rightarrow T)$
MORTON	$\prod_{n:\mathbb{N}} \prod_T \text{Type}(\mathbb{N}^n \rightarrow \mathbb{N}) \times (T \rightarrow T)$
HILBERT	$\prod_T \text{Type}(\mathbb{N}^2 \rightarrow \mathbb{N}) \times (T \rightarrow T)$
SHUFFLE	$\prod_{n:\mathbb{N}} \prod_{\rho:\mathfrak{S}_n} \prod_{S,T} \text{Type}(S^n \rightarrow S^n) \times (T \rightarrow T)$
DEFAULT	$\prod_{S,T} \text{Type}(S \rightarrow S + \mathbb{1}) \times (T + \mathbb{1} \rightarrow T)$
WRAP	$\prod_{S,T} \text{Type}(S \rightarrow S) \times (T \rightarrow T)$
NEAREST	$\prod_{n:\mathbb{N}} \prod_T \text{Type}(\mathbb{R}^n \rightarrow \mathbb{N}^n) \times (T \rightarrow T)$
LINEAR	$\prod_{n:\mathbb{N}} (\mathbb{R}^n \rightarrow (\mathbb{N}^n)^{2^n} \times \mathbb{R}^n) \times ((\mathbb{R}^n)^{2^n} \times \mathbb{R}^3 \rightarrow \mathbb{R}^n)$
AFFINE	$\prod_{n:\mathbb{N}} \prod_T \text{Type}(\mathbb{R}^n \rightarrow \mathbb{R}^n) \times (T \rightarrow T)$

Moving this to a GPU

CPU

```
1 using field_t = covfie::interpolator::linear<
2   covfie::boundary::def<
3     covfie::layout::morton<
4       ulong2,
5       covfie::storage::array<
6         float2
7     >
8   >
9 >
10 >;
```

GPU

```
1 using field_t = covfie::interpolator::linear<
2   covfie::boundary::def<
3     covfie::layout::morton<
4       ulong2,
5       covfie::storage::cuda::array<
6         float2
7     >
8   >
9 >
10 >;
```

Rendering on a GPU

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

```
1  __global__ void render(  
2      typename field_t::view_t vf, char * out,  
3      uint width, uint height, float z  
4  ) {  
5      int x = blockDim.x * blockIdx.x + threadIdx.x;  
6      int y = blockDim.y * blockIdx.y + threadIdx.y;  
7  
8      if (x < width && y < height) {  
9          float fx = x / static_cast<float>(width);  
10         float fy = y / static_cast<float>(height);  
11  
12         typename field_t::output_t p = vf.at(fx, fy, z);  
13         out[height * x + y] = static_cast<char>(  
14             255.f *  
15             std::min(mag(p), 1.0f)  
16         );  
17     }  
18 }
```

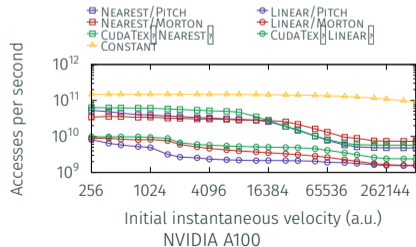
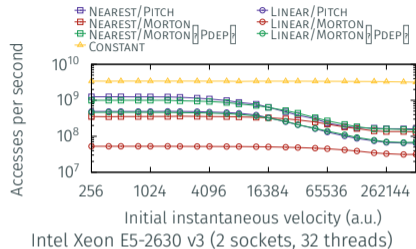
Texture “Memory”

- 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 COVIE backend



Benchmarking

- COVIE 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



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!](mailto:stephen.nicholas.swatman@cern.ch)

acts-project / covfie Public

Code Issues Pull requests Actions Security Insights

main 1 branch 3 tags

Go to file Add file Code

stephenwat Increment version to 0.4.0 ✓ Insect 13 days ago 219 commits

github/workflows	Add support for building with CUDA in CI	5 months ago
benchmarks	Change Lorentz-Euler execution parameters	last month
cmake	Add optional flag to silence compiler messages	13 days ago
docs	Add optional flag to silence compiler messages	13 days ago
examples	Remove mentions of ATLAS experiment	last month
extraLayers	Remove layout throughput analysis	last month
lib	Add optional flag to silence compiler messages	13 days ago
tests	Expand generic CUDA tests	last month
.clang-format	Improve clang-format include sorting	4 months ago
gignore	Add basic Dorygen support	5 months ago
CMakeLists.txt	Increment version to 0.4.0	13 days ago
LICENSE	Initial commit	5 months ago
README.md	Update README examples to new vectors	last month

README.md

covfie

Build tests passing Code Checks passing Apps passing Score 99.2.0 Issues Open Pull requests Open

Latest activity 4 weeks Contributors 1 Last commit 3 months Subscribers 0

covfie (pronounced coffee) is a co-processor vector field library. covfie consists of two main components; the first is the header-only C++ library, which can be used by scientific applications using CUDA or other programming platforms. The second is a set of benchmarks which can be used to quantify the computational performance of all the different vector field implementations covfie provides. Arguably, the test suite constitutes a third component.

About: A library for storing interpolatable vector fields on co-processors

Releases: 3 logs Create a new release

Packages: No packages published Publish your first package

Languages: C++ 99.2%, CMake 0.0%, Code 2.0%, Haskell 1.7%