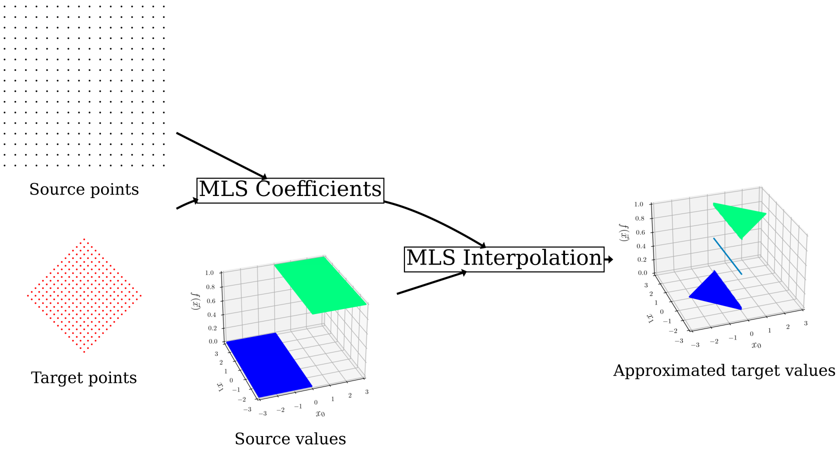
KUG 2023 - Moving least squares

Yohann Bosqued

December 14, 2023

Moving least squares

Principle


Source points
MLS Coefficients
Target points
MLS Interpolation
Source values
Approximated target values

Definitions

- Let $A(\vec{x})$ be a polynomial, then $A(\vec{x}) = p(\vec{x}).a^T = \langle p(\vec{x})|a \rangle$ with $p(\vec{x})$, the polynomial basis taken at $\vec{x}$.
  $p(\begin{bmatrix} x & y \end{bmatrix}) = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 \end{bmatrix}$
- Let ϕ be a compactly supported radial basis function. Its value is only dependant on the input's norm and is supported on a compact set. Its goal is to select points and give each an influence on the final result.
  $\phi(\vec{x}) = \begin{cases} (1 - \|\vec{x}\|)^2 & \text{if } \|\vec{x}\| \leq 1 \\ 0 & \text{else} \end{cases}$

Weighted and moving least squares

Let f be a function from $\mathbb{R}^d$ to $\mathbb{R}$ and a set of source points S. The goal of the weighted least squares is to find the polynomial G minimizing $\|G - f\|_{\phi,2}$. Using the polynomial basis, finding g minimizing $\| \langle p(\cdot)|g\rangle - f\|_{\phi,2}$.

$$\begin{aligned} g^T &= \left[P^T \Phi P\right]^{-1} P^T \Phi F^T \\ f(\vec{u}) &\approx \left\langle p(\vec{u})|g \right\rangle \\ &\approx p(\vec{u}) \left[P^T \Phi P\right]^{-1} P^T \Phi F^T \end{aligned}$$

Weighted and moving least squares

Let f be a function from $\mathbb{R}^d$ to $\mathbb{R}$ and a set of source points S. The goal of the weighted least squares is to find the polynomial G minimizing $\|G - f\|_{\phi,2}$. Using the polynomial basis, finding g minimizing $\| \langle p(\cdot)|g\rangle - f\|_{\phi,2}$.

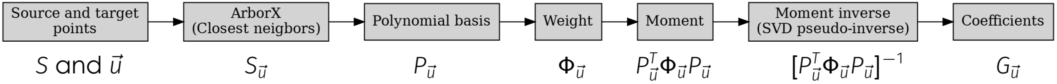
$$
\begin{aligned}
g^T &= \left[P^T \Phi P\right]^{-1} P^T \Phi F^T \\
f(\vec{u}) &\approx \left\langle p(\vec{u}) | g \right\rangle \\
&\approx p(\vec{u}) \left[P^T \Phi P\right]^{-1} P^T \Phi F^T
\end{aligned}
$$

$$
\begin{aligned}
g_{\vec{u}}^T &= \left[P_{\vec{u}}^T \Phi_{\vec{u}} P_{\vec{u}}\right]^{-1} P_{\vec{u}}^T \Phi_{\vec{u}} F_{\vec{u}}^T \\
f(\vec{u}) &\approx \left\langle p(\vec{u}) | g_{\vec{u}} \right\rangle \\
&\approx p(\vec{u}) \left[P_{\vec{u}}^T \Phi_{\vec{u}} P_{\vec{u}}\right]^{-1} P_{\vec{u}}^T \Phi_{\vec{u}} F_{\vec{u}}^T
\end{aligned}
$$

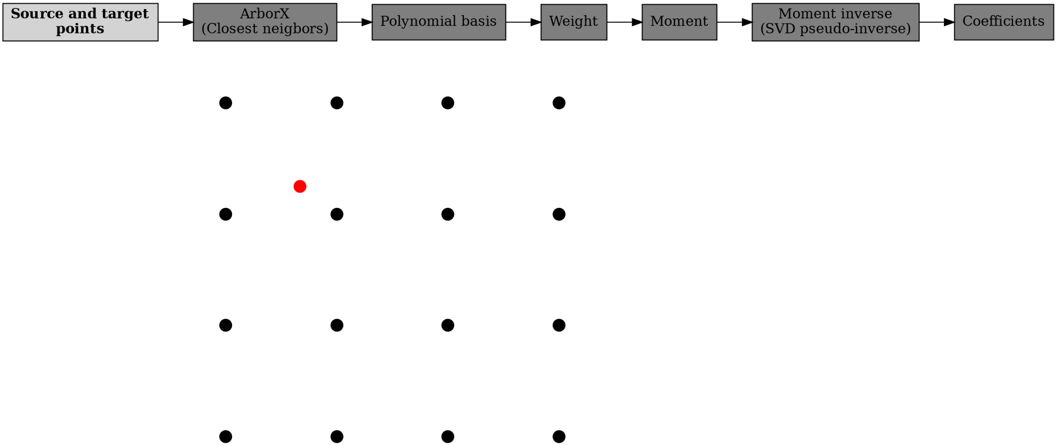
$$
\begin{aligned}
f(\vec{u}) &\approx \langle G_{\vec{u}} | F_{\vec{u}} \rangle \\
G_{\vec{u}} &= p(\vec{u}) \left[P_{\vec{u}}^T \Phi_{\vec{u}} P_{\vec{u}}\right]^{-1} P_{\vec{u}}^T \Phi_{\vec{u}}
\end{aligned}
$$

What needs to be done

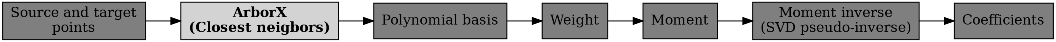
- From a set of points S and a target point $\vec{u}$:
  - Find the closest set of neighboring points $S_{\vec{u}}$
  - Compute and save the coefficients $\left(G_{\vec{u}} = p(\vec{u}) \left[P_{\vec{u}}^T \Phi_{\vec{u}} P_{\vec{u}}\right]^{-1} P_{\vec{u}}^T \Phi_{\vec{u}}\right)$
  - Now ready to make approximations!
- Coefficients and approximation are computed in batches of target points

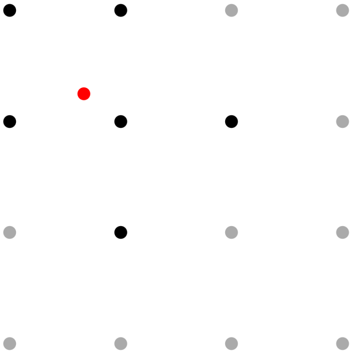


Example (Coefficients)



Example (Coefficients)

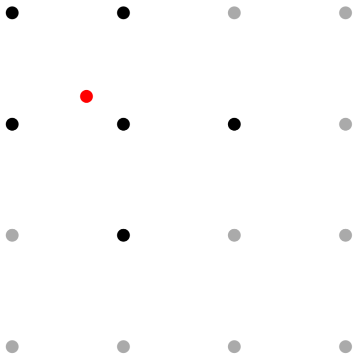




- 6 neighbors per target point

Example (Coefficients)

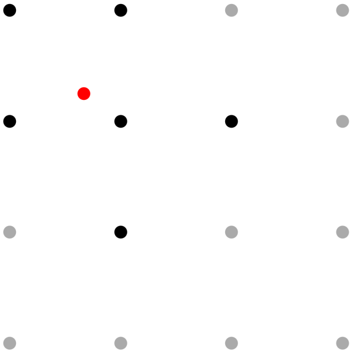
Source and target points → ArborX (Closest neigbors) → **Polynomial basis** → **Weight** → Moment → Moment inverse (SVD pseudo-inverse) → Coefficients



- Linear poly basis (size 3)
- $P_{\vec{u}}$: 6×3
- $\Phi_{\vec{u}}$: 6×6

Example (Coefficients)

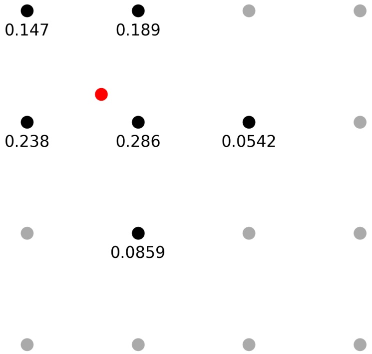
Source and target points → ArborX (Closest neigbors) → Polynomial basis → Weight → **Moment** → **Moment inverse (SVD pseudo-inverse)** → Coefficients



- $M_{\vec{u}}$: 3×3
- $M_{\vec{u}}^{-1}$: 3×3

Example (Coefficients)

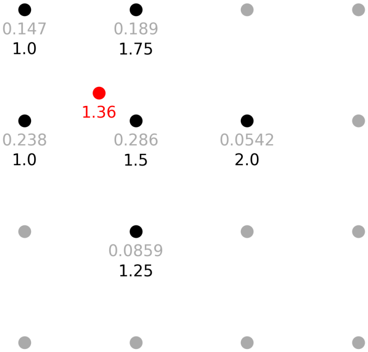
Source and target points → ArborX (Closest neigbors) → Polynomial basis → Weight → Moment → Moment inverse (SVD pseudo-inverse) → **Coefficients**



- $G_{\vec{u}}$: 6×1

Example (Approximation)

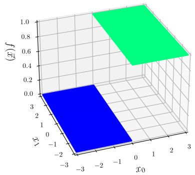
$$f(\vec{x}) = \frac{x_0 x_1}{4} + 1$$



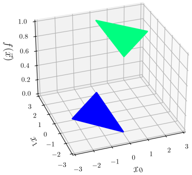
$$\begin{bmatrix} 0.147 \\ 0.189 \\ 0.238 \\ 0.286 \\ 0.054 \\ 0.086 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1.75 \\ 1 \\ 1.5 \\ 2.0 \\ 1.25 \end{bmatrix}$$

Example (Larger scale)

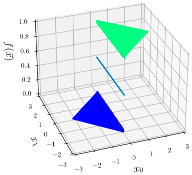
$$f(\vec{x}) = \frac{\mathrm{sgn}(x_0)+1}{2}$$



Source values



Real target values



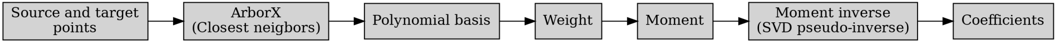
Approximated values

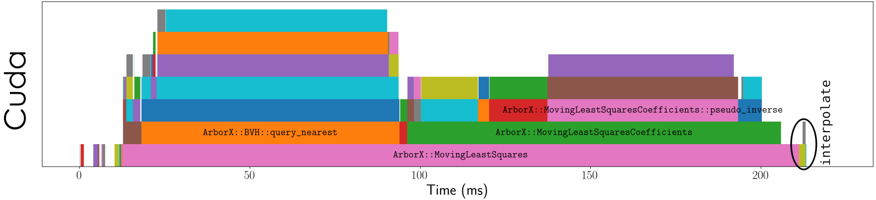
Interface

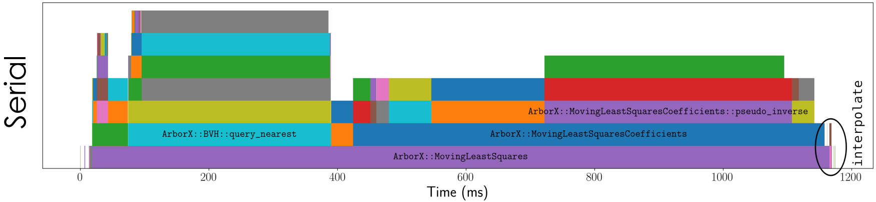
```
Kokkos::View<Point*, MemorySpace> source_points;
Kokkos::View<Point*, MemorySpace> target_points;
Kokkos::View<double*, MemorySpace> source_values;
Kokkos::View<double*, MemorySpace> approx_values;

// ...

MovingLeastSquares<MemorySpace> mls(exec_space, source_points, target_points);
mls.interpolate(exec_space, source_values, approx_values);
// source_values = ...
mls.interpolate(exec_space, source_values, approx_values);
```

Performance

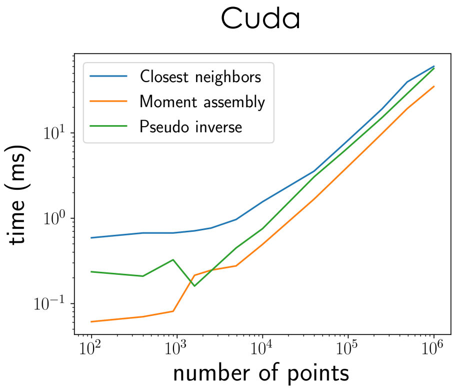
Performance (1M points)

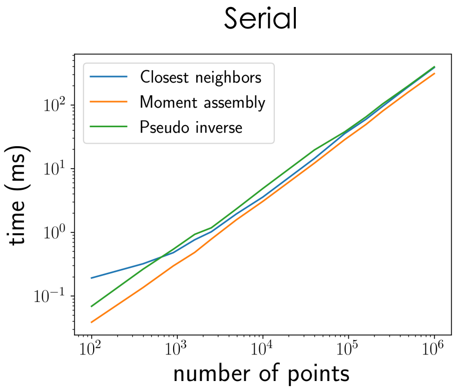




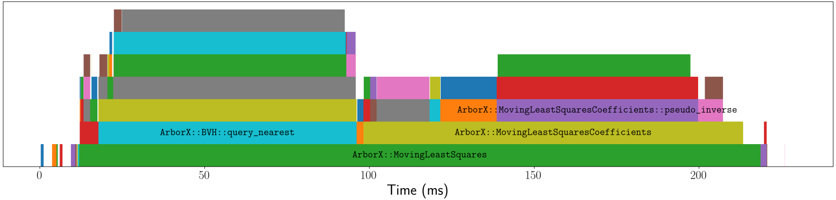


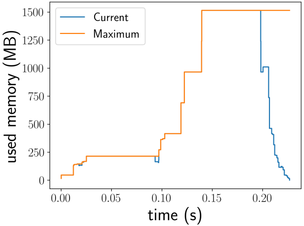
Performance





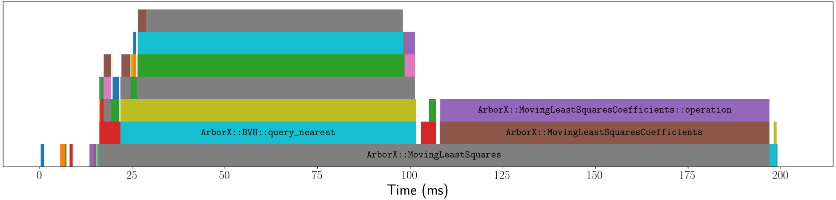
Performance (Update!)

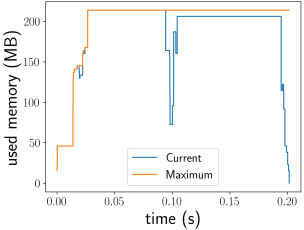




- No scratch pad use, global memory only
- Multiple kernels, heavy use of `MDRangePolicy`

Performance (Update!)





- Scratch pad for temporary data
- Single kernel, parallelized with `TeamPolicy` (One target per thread)
- Originally 1200B/target, now down to 960B/target with extra work

OAK RIDGE National Laboratory

Learning Kokkos

Cursus and previous internships

- Parallelization
- OpenMP
- MPI
- …

- CUDA
- OpenMP Target
- GPU offloading
- …

What I used to do

```
#pragma omp target distribute teams parallel reduce op(+:...) is_device_ptr(...)
for (int i = 0; i < N; i++) {
  // ...
}
```

and manual memory management

What I do now

```
Kokkos::parallel_reduce(Kokkos::RangePolicy<ExecutionSpace>(space, 0, N),
KOKKOS_LAMBDA (int i, double& loc) {
  // ...
}, /**/);
```

and smart memory management (with `Views`!)

What will I do

Questions?