UNIVERSITEIT
VAN
AMSTERDAM

Factorization, Join (and Meet) of Blades

Efficient algorithms for factorization of blades and and computing the join of blades.

Daniel Fontijne
University of Amsterdam
fontijne@science.uva.nl

UNIVERSITEIT VAN AMSTERDAM

Motivation

Blade factorization: $\mathbf{B}_k = \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$.

UNIVERSITEIT VAN AMSTERDAM

Motivation

Blade factorization: $\mathbf{B}_k = \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$.

Applications of blade factorization:

- Conversion to other (LA-compatible) representations.
- As a building block of other algorithms.

UNIVERSITEIT VAN AMSTERDAM

Motivation

Blade factorization: $\mathbf{B}_k = \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$.

Applications of blade factorization:

- Conversion to other (LA-compatible) representations.
- As a building block of other algorithms.

The join: $\mathbf{A} \cup \mathbf{B}$ is the union of $\mathbf{A}$ and $\mathbf{B}$.

Motivation

Universiteit van Amsterdam

Blade factorization: $\mathbf{B}_k = \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$.

Applications of blade factorization:

- Conversion to other (LA-compatible) representations.
- As a building block of other algorithms.

The join: $\mathbf{A} \cup \mathbf{B}$ is the union of $\mathbf{A}$ and $\mathbf{B}$.

Applications of the join:

- True union of subspaces.
- Computing the meet.

In my implementation the join is interwoven with factorization, so factorization must be discussed first.

UNIVERSITEIT VAN AMSTERDAM

Blade Representation

The algorithms in this talk are based on the *additive presentation*. Blades are represented as a sum of basis blades.

Example of basis for 3-D space:

$$\{\ \underbrace{1}_{\textit{grade 0}}\ ,\ \underbrace{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3}_{\textit{grade 1}}\ ,\ \underbrace{\mathbf{e}_1 \wedge \mathbf{e}_2,\ \mathbf{e}_2 \wedge \mathbf{e}_3,\ \mathbf{e}_1 \wedge \mathbf{e}_3}_{\textit{grade 2}}\ ,\ \underbrace{\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3}_{\textit{grade 3}}\}.$$

UNIVERSITEIT VAN AMSTERDAM

Example of FastFactorization

Suppose our input blade is:

$$\mathbf{B} = 1.0\,\mathbf{e}_1{\wedge}\mathbf{e}_2{\wedge}\mathbf{e}_3 - 0.5\,\mathbf{e}_1{\wedge}\mathbf{e}_3{\wedge}\mathbf{e}_4 + 0.25\,\mathbf{e}_2{\wedge}\mathbf{e}_3{\wedge}\mathbf{e}_4 - 0.75\,\mathbf{e}_1{\wedge}\mathbf{e}_2{\wedge}\mathbf{e}_4.$$

FastFactorization factorizes this to:

$$\begin{aligned}
\mathbf{b}_1 &= 1.0\,\mathbf{e}_1 && + 0.25\,\mathbf{e}_4,\\
\mathbf{b}_2 &= 1.0\,\mathbf{e}_2 && + 0.5\,\mathbf{e}_4,\\
\mathbf{b}_3 &= 1.0\,\mathbf{e}_3 && - 0.75\,\mathbf{e}_4,
\end{aligned}$$

such that $\mathbf{B} = \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \mathbf{b}_3$.

UNIVERSITEIT VAN AMSTERDAM

Example of FastFactorization

Suppose our input blade is:

$$\mathbf{B} = 1.0\,\mathbf{e}_1\wedge\mathbf{e}_2\wedge\mathbf{e}_3 - 0.5\,\mathbf{e}_1\wedge\mathbf{e}_3\wedge\mathbf{e}_4 + 0.25\,\mathbf{e}_2\wedge\mathbf{e}_3\wedge\mathbf{e}_4 - 0.75\,\mathbf{e}_1\wedge\mathbf{e}_2\wedge\mathbf{e}_4.$$

FastFactorization factorizes this to:

$$\begin{aligned}
\mathbf{b}_1 &= 1.0\,\mathbf{e}_1 && + 0.25\,\mathbf{e}_4,\\
\mathbf{b}_2 &= 1.0\,\mathbf{e}_2 && + 0.5\,\mathbf{e}_4,\\
\mathbf{b}_3 &= 1.0\,\mathbf{e}_3 && - 0.75\,\mathbf{e}_4,
\end{aligned}$$

such that $\mathbf{B} = \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \mathbf{b}_3$.

The coordinates of the factors are $\pm$ the coordinates of the input blade! How does this work?

UNIVERSITEIT VAN AMSTERDAM

Basic Factorization Algorithm

Algorithm $Factorization(\mathrm{B})$:

UNIVERSITEIT VAN AMSTERDAM

Basic Factorization Algorithm

Algorithm $Factorization(\mathbf{B})$:

1. Find the largest basis blade $\mathbf{F}$ in the representation of $\mathbf{B}$. I.e., $\mathbf{F} = \mathbf{e}_i \wedge \mathbf{e}_j \wedge \ldots \wedge \mathbf{e}_k$.

UNIVERSITEIT VAN AMSTERDAM

Basic Factorization Algorithm

Algorithm $Factorization(\mathbf{B})$:

1. Find the largest basis blade $\mathbf{F}$ in the representation of $\mathbf{B}$.
   I.e., $\mathbf{F} = \mathbf{e}_i \wedge \mathbf{e}_j \wedge \ldots \wedge \mathbf{e}_k$.
2. Project the basis vectors of $\mathbf{F}$ onto $\mathbf{B}$.
   Use orthogonal projection: $\mathbf{b}_i = (\mathbf{e}_i \rfloor \mathbf{B}) \rfloor \mathbf{B}^{-1}$.
   The $\mathbf{b}_i$ will be independent.

Basic Factorization Algorithm

Algorithm $Factorization(\mathbf{B})$:

1. Find the largest basis blade $\mathbf{F}$ in the representation of $\mathbf{B}$.
   I.e., $\mathbf{F} = \mathbf{e}_i \wedge \mathbf{e}_j \wedge \ldots \wedge \mathbf{e}_k$.
2. Project the basis vectors of $\mathbf{F}$ onto $\mathbf{B}$.
   Use orthogonal projection: $\mathbf{b}_i = (\mathbf{e}_i \rfloor \mathbf{B}) \rfloor \mathbf{B}^{-1}$.
   The $\mathbf{b}_i$ will be independent.
3. Compute the scale β such that $\mathbf{B} = \beta\, \mathbf{b}_i \wedge \mathbf{b}_j \wedge \ldots \wedge \mathbf{b}_k$.

UNIVERSITEIT VAN AMSTERDAM

Basic Factorization Algorithm

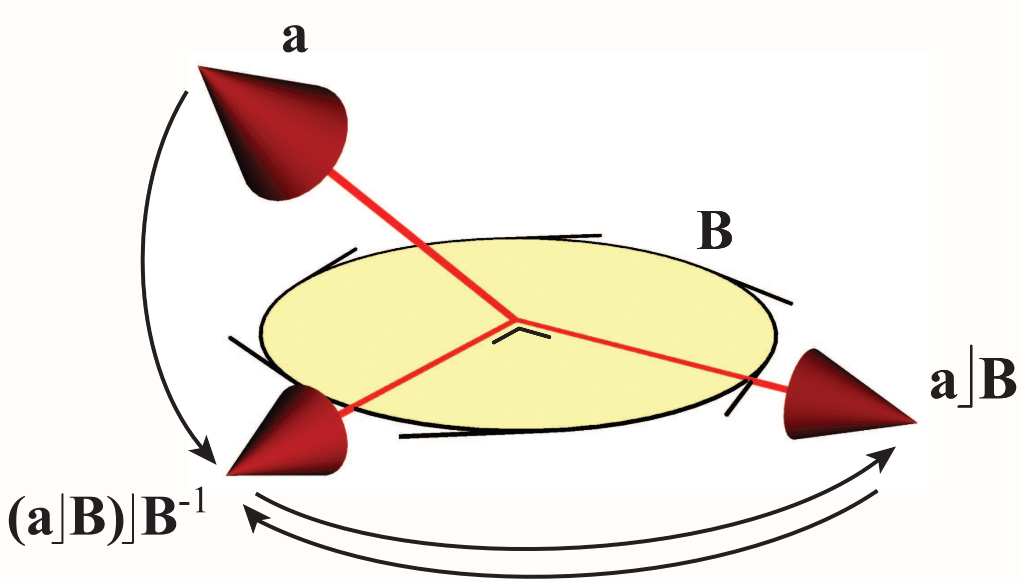
Algorithm $Factorization(\mathbf{B})$:

1. Find the largest basis blade $\mathbf{F}$ in the representation of $\mathbf{B}$.
   I.e., $\mathbf{F} = \mathbf{e}_i \wedge \mathbf{e}_j \wedge \ldots \wedge \mathbf{e}_k$.
2. Project the basis vectors of $\mathbf{F}$ onto $\mathbf{B}$.
   Use orthogonal projection: $\mathbf{b}_i = (\mathbf{e}_i \rfloor \mathbf{B}) \rfloor \mathbf{B}^{-1}$.
   The $\mathbf{b}_i$ will be independent.
3. Compute the scale β such that $\mathbf{B} = \beta\, \mathbf{b}_i \wedge \mathbf{b}_j \wedge \ldots \wedge \mathbf{b}_k$.

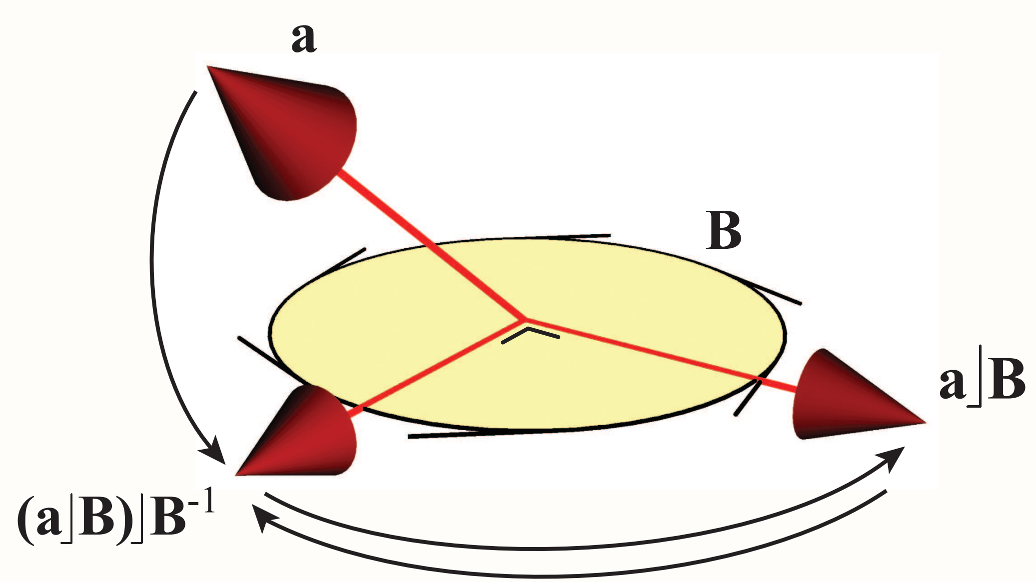
This works but is a bit slow (in our implementation, $50\times$ to $100\times$ slower than a simple bilinear outer product).
$\rightarrow$The projection is expensive!

UNIVERSITEIT
VAN
AMSTERDAM

Orthogonal Projection Shortcut



Orthogonal Projection Shortcut



Instead of doing a true projection $\mathbf{b}_i = (\mathbf{e}_i \rfloor \mathbf{B}) \rfloor \mathbf{B}^{-1}$,
we do a 'pseudo projection' $\mathbf{b}_i = (\mathbf{e}_i \rfloor \mathbf{F}) \rfloor \mathbf{B}^{-1}$.

The pseudo projection is computationally cheap because it amounts to simply selecting coordinates from $\mathbf{B}$.

FastFactorization Algorithm

Algorithm $\mathrm{FastFactorization}(\mathbf{B})$:

UNIVERSITEIT VAN AMSTERDAM

FastFactorization Algorithm

Algorithm $\mathrm{FastFactorization}(\mathbf{B})$:

Let $\mathbf{B}$ be a k-blade, with $1 < k < n$.
The algorithm computes a factorization
$\mathbf{B} = \beta\, \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$, where β is a scalar:

UNIVERSITEIT VAN AMSTERDAM

FastFactorization Algorithm

Algorithm $\mathrm{FastFactorization}(\mathbf{B})$:

Let $\mathbf{B}$ be a k-blade, with $1 < k < n$.
The algorithm computes a factorization
$\mathbf{B} = \beta\, \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$, where β is a scalar:

1. Find the basis blade $\mathbf{F} = \mathbf{f}_1 \wedge \mathbf{f}_2 \wedge \ldots \wedge \mathbf{f}_k$ to which the absolute largest coordinate of $\mathbf{B}$ refers. The $\mathbf{f}_i$ are basis vectors. Let β be the coordinate that refers to $\mathbf{F}$.

UNIVERSITEIT VAN AMSTERDAM

FastFactorization Algorithm

Algorithm $\mathrm{FastFactorization}(\mathbf{B})$:

Let $\mathbf{B}$ be a k-blade, with $1 < k < n$.
The algorithm computes a factorization
$\mathbf{B} = \beta\,\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$, where β is a scalar:

1. Find the basis blade $\mathbf{F} = \mathbf{f}_1 \wedge \mathbf{f}_2 \wedge \ldots \wedge \mathbf{f}_k$ to which the absolute largest coordinate of $\mathbf{B}$ refers. The $\mathbf{f}_i$ are basis vectors. Let β be the coordinate that refers to $\mathbf{F}$.
2. Compute $\mathbf{B}_s = \mathbf{B}/\beta$.

UNIVERSITEIT VAN AMSTERDAM

FastFactorization Algorithm

Algorithm $\mathrm{FastFactorization}(\mathbf{B})$:

Let $\mathbf{B}$ be a k-blade, with $1 < k < n$.
The algorithm computes a factorization
$\mathbf{B} = \beta\, \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$, where β is a scalar:

1. Find the basis blade $\mathbf{F} = \mathbf{f}_1 \wedge \mathbf{f}_2 \wedge \ldots \wedge \mathbf{f}_k$ to which the absolute largest coordinate of $\mathbf{B}$ refers. The $\mathbf{f}_i$ are basis vectors. Let β be the coordinate that refers to $\mathbf{F}$.
2. Compute $\mathbf{B}_s = \mathbf{B}/\beta$.
3. For each $\mathbf{f}_i$ compute: $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}_s$.

UNIVERSITEIT VAN AMSTERDAM

FastFactorization Algorithm

Algorithm $\mathrm{FastFactorization}(\mathbf{B})$:

Let $\mathbf{B}$ be a k-blade, with $1 < k < n$.
The algorithm computes a factorization
$\mathbf{B} = \beta\, \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \ldots \wedge \mathbf{b}_k$, where β is a scalar:

1. Find the basis blade $\mathbf{F} = \mathbf{f}_1 \wedge \mathbf{f}_2 \wedge \ldots \wedge \mathbf{f}_k$ to which the absolute largest coordinate of $\mathbf{B}$ refers. The $\mathbf{f}_i$ are basis vectors. Let β be the coordinate that refers to $\mathbf{F}$.
2. Compute $\mathbf{B}_s = \mathbf{B}/\beta$.
3. For each $\mathbf{f}_i$ compute: $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}_s$.

Because the k vectors $\mathbf{b}_i$ are linearly independent and all contained in $\mathbf{B}$, they must form a factorization of $\mathbf{B}_s$.

UNIVERSITEIT VAN AMSTERDAM

FastFactorization ‘Proof’

(The full proof in the paper).
Again, suppose our input blade is:

$$\mathbf{B} = 1.0\,\mathbf{e}_1\wedge\mathbf{e}_2\wedge\mathbf{e}_3 - 0.5\,\mathbf{e}_1\wedge\mathbf{e}_3\wedge\mathbf{e}_4 + 0.25\,\mathbf{e}_2\wedge\mathbf{e}_3\wedge\mathbf{e}_4 - 0.75\,\mathbf{e}_1\wedge\mathbf{e}_2\wedge\mathbf{e}_4.$$

Then $\mathbf{F} = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3$, and the factors are:

$$\begin{array}{rcllll} \mathbf{b}_1 & = & 1.0\,\mathbf{e}_1 & & & +\,0.25\,\mathbf{e}_4, \\ \mathbf{b}_2 & = & & 1.0\,\mathbf{e}_2 & & +\,0.5\,\mathbf{e}_4, \\ \mathbf{b}_3 & = & & & 1.0\,\mathbf{e}_3 & -\,0.75\,\mathbf{e}_4. \end{array}$$

The diagonal typesetting of $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ should make it obvious that the $\mathbf{b}_i$ are linearly independent.

UNIVERSITEIT VAN AMSTERDAM

FastFactorization Code Generation

We used code generation to implement FastFactorization.

One function was generated for each valid combination of basis blade and grade.

Example of a generated function:

```
void factorE234grade3(const float *B, float **b) {
  b[2][0] = B[0];
  b[1][0] = -B[1];
  b[0][0] = B[2];
  b[0][1] = b[1][2] = b[2][3] = B[3];
  b[2][4] = B[6];
  b[1][4] = -B[8];
  b[0][4] = B[9];
  b[0][2] = b[0][3] = b[1][1] = b[1][3] = b[2][1] = b[2][2] = 0.0f;
}
```

UNIVERSITEIT VAN AMSTERDAM

FastFactorization Implementation

The full FastFactorization implementation amounts to:

- Filter out trivial special cases (hand written).
- Find largest coordinate / basis blade (hand written).
- Call the appropriate factorization function (generated) via a lookup table .

FastFactorization Benchmarks

Benchmark: Factorize millions of random blades.

Used one CPU on a Core2Duo 1.83Ghz.
Compiled using VS2005.

n	3	4	5	6
factorizations per second	15M	9.2M	5.2M	2.8M
relative to O.P.	5.1×	5.1×	3.4×	3.8×

UNIVERSITEIT VAN AMSTERDAM

The Join (and the Meet)

The join $\mathbf{A} \cup \mathbf{B}$ is the union of $\mathbf{A}$ and $\mathbf{B}$.

The join is a non-linear product, for example in general
$\mathbf{A} \cup (\mathbf{B} + \mathbf{C}) \neq \mathbf{A} \cup \mathbf{B} + \mathbf{A} \cup \mathbf{C}$.

UNIVERSITEIT VAN AMSTERDAM

The Join (and the Meet)

The join $\mathbf{A} \cup \mathbf{B}$ is the union of $\mathbf{A}$ and $\mathbf{B}$.

The join is a non-linear product, for example in general $\mathbf{A} \cup (\mathbf{B} + \mathbf{C}) \neq \mathbf{A} \cup \mathbf{B} + \mathbf{A} \cup \mathbf{C}$.

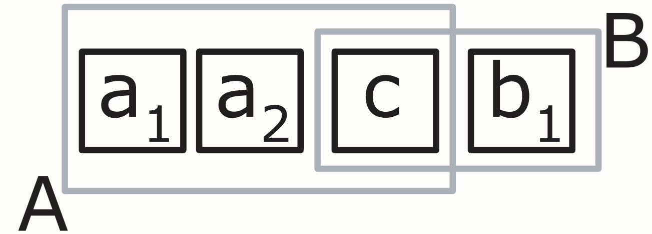
The meet $\mathbf{A} \cap \mathbf{B}$ can be (most?) efficiently computed from the join using $\mathbf{A} \cap \mathbf{B} = (\mathbf{B} \rfloor (\mathbf{A} \cup \mathbf{B})^{-1}) \rfloor \mathbf{A}$.

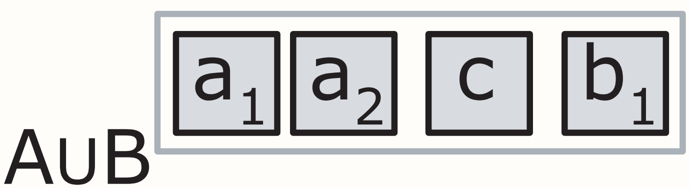
Universiteit van Amsterdam

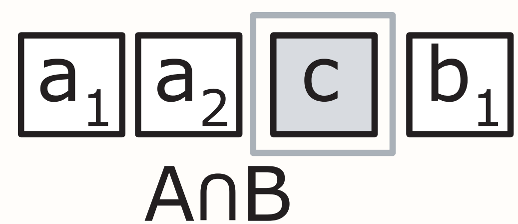
The Join, Meet and Delta Product Illustrated

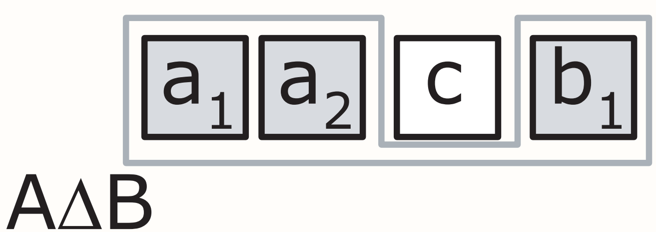
$A = a_1 \wedge a_2 \wedge c$

$B = c \wedge b_1$









UNIVERSITEIT
VAN
AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.

UNIVERSITEIT VAN AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.

UNIVERSITEIT
VAN
AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.
4. Find the largest basis blade term $\mathbf{F}$ in $\mathbf{B}$.

UNIVERSITEIT VAN AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.
4. Find the largest basis blade term $\mathbf{F}$ in $\mathbf{B}$.
5. While $\mathrm{grade}(\mathbf{J}) \neq n$ and not all basis vectors $\mathbf{f}_i$ in $\mathbf{F}$ have been tried:

UNIVERSITEIT VAN AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.
4. Find the largest basis blade term $\mathbf{F}$ in $\mathbf{B}$.
5. While $\mathrm{grade}(\mathbf{J}) \neq n$ and not all basis vectors $\mathbf{f}_i$ in $\mathbf{F}$ have been tried:
   (a) Take basis vector $\mathbf{f}_i$ in $\mathbf{F}$ which has not been tried yet.

UNIVERSITEIT VAN AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.
4. Find the largest basis blade term $\mathbf{F}$ in $\mathbf{B}$.
5. While $\mathrm{grade}(\mathbf{J}) \neq n$ and not all basis vectors $\mathbf{f}_i$ in $\mathbf{F}$ have been tried:
   (a) Take basis vector $\mathbf{f}_i$ in $\mathbf{F}$ which has not been tried yet.
   (b) Compute $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}$.

UNIVERSITEIT VAN AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.
4. Find the largest basis blade term $\mathbf{F}$ in $\mathbf{B}$.
5. While $\mathrm{grade}(\mathbf{J}) \neq n$ and not all basis vectors $\mathbf{f}_i$ in $\mathbf{F}$ have been tried:
   (a) Take basis vector $\mathbf{f}_i$ in $\mathbf{F}$ which has not been tried yet.
   (b) Compute $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}$.
   (c) Compute $\mathbf{H} = \mathbf{J} \wedge \mathrm{unit}(\mathbf{b}_i)$.

UNIVERSITEIT VAN AMSTERDAM

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.
4. Find the largest basis blade term $\mathbf{F}$ in $\mathbf{B}$.
5. While $\mathrm{grade}(\mathbf{J}) \neq n$ and not all basis vectors $\mathbf{f}_i$ in $\mathbf{F}$ have been tried:
   (a) Take basis vector $\mathbf{f}_i$ in $\mathbf{F}$ which has not been tried yet.
   (b) Compute $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}$.
   (c) Compute $\mathbf{H} = \mathbf{J} \wedge \mathrm{unit}(\mathbf{b}_i)$.
   (d) If $(\|\mathbf{H}\| \geq \epsilon)$ set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{H})$.

The FastJoin Algorithm

Algorithm $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$:

1. Filter out trivial cases.
2. Swap $\mathbf{A}$ and $\mathbf{B}$ such that $\mathrm{grade}(\mathbf{A}) \geq \mathrm{grade}(\mathbf{B})$.
3. Set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{A})$.
4. Find the largest basis blade term $\mathbf{F}$ in $\mathbf{B}$.
5. While $\mathrm{grade}(\mathbf{J}) \neq n$ and not all basis vectors $\mathbf{f}_i$ in $\mathbf{F}$ have been tried:
   (a) Take basis vector $\mathbf{f}_i$ in $\mathbf{F}$ which has not been tried yet.
   (b) Compute $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}$.
   (c) Compute $\mathbf{H} = \mathbf{J} \wedge \mathrm{unit}(\mathbf{b}_i)$.
   (d) If $(\|\mathbf{H}\| \geq \epsilon)$ set $\mathbf{J} \leftarrow \mathrm{unit}(\mathbf{H})$.
6. Return $\mathbf{J}$.

UNIVERSITEIT
VAN
AMSTERDAM

Limitations of the FastJoin Algorithm

Limitations of the FastJoin algorithm:

- Grade stability.

Limitations of the FastJoin Algorithm

Limitations of the FastJoin algorithm:

- Grade stability.
- Numerical stability.

UNIVERSITEIT VAN AMSTERDAM

Limitations of the FastJoin Algorithm

Limitations of the FastJoin algorithm:

- Grade stability.
- Numerical stability.

The StableFastJoin algorithm (next slide) solves both problems.

Limitations of the FastJoin Algorithm

Limitations of the FastJoin algorithm:

- Grade stability.
- Numerical stability.

The StableFastJoin algorithm (next slide) solves both problems.

The delta product Δ (geometric symmetric difference) is used:

$$\mathrm{grade}(\mathbf{A} \cup \mathbf{B}) = \frac{\mathrm{grade}(\mathbf{A}) + \mathrm{grade}(\mathbf{B}) + \mathrm{grade}(\mathbf{A}\Delta\mathbf{B})}{2}.$$

UNIVERSITEIT VAN AMSTERDAM

The StableFastJoin Algorithm

Algorithm $\mathrm{StableFastJoin}(\mathbf{A}, \mathbf{B}, \epsilon, \delta)$:

Start with steps 1-5 of $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$.

UNIVERSITEIT VAN AMSTERDAM

The StableFastJoin Algorithm

Algorithm $\mathrm{StableFastJoin}(\mathbf{A}, \mathbf{B}, \epsilon, \delta)$:

Start with steps 1-5 of $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$.

6. If $(\mathrm{grade}(\mathbf{J}) = n)$
or $(\mathrm{grade}(\mathbf{J}) = \mathrm{grade}(\mathbf{A}) + \mathrm{grade}(\mathbf{B}))$,
return $\mathbf{J}$. Otherwise:

Universiteit van Amsterdam

The StableFastJoin Algorithm

Algorithm $\mathrm{StableFastJoin}(\mathbf{A}, \mathbf{B}, \epsilon, \delta)$:

Start with steps 1-5 of $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$.

6. If $(\mathrm{grade}(\mathbf{J}) = n)$
   or $(\mathrm{grade}(\mathbf{J}) = \mathrm{grade}(\mathbf{A}) + \mathrm{grade}(\mathbf{B}))$,
   return $\mathbf{J}$. Otherwise:
7. Compute $\mathrm{grade}(\mathbf{A} \cup \mathbf{B})$ using the delta product.

UNIVERSITEIT VAN AMSTERDAM

The StableFastJoin Algorithm

Algorithm $\mathrm{StableFastJoin}(\mathbf{A}, \mathbf{B}, \epsilon, \delta)$:

Start with steps 1-5 of $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$.

6. If $(\mathrm{grade}(\mathbf{J}) = n)$
   or $(\mathrm{grade}(\mathbf{J}) = \mathrm{grade}(\mathbf{A}) + \mathrm{grade}(\mathbf{B}))$,
   return $\mathbf{J}$. Otherwise:
7. Compute $\mathrm{grade}(\mathbf{A} \cup \mathbf{B})$ using the delta product.
8. While $(\mathrm{grade}(\mathbf{J}) < \mathrm{grade}(\mathbf{A} \cup \mathbf{B}))$
   (a) For all valid i, compute $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}$.
   Set $\mathbf{b}_m$ to that $\mathbf{b}_i$ which leads to the largest $\|\mathbf{J} \wedge \mathbf{b}_i\|$.
   (b) Update $\mathbf{J} \leftarrow \mathbf{J} \wedge \mathbf{b}_m$.

UNIVERSITEIT VAN AMSTERDAM

The StableFastJoin Algorithm

Algorithm $\mathrm{StableFastJoin}(\mathbf{A}, \mathbf{B}, \epsilon, \delta)$:

Start with steps 1-5 of $\mathrm{FastJoin}(\mathbf{A}, \mathbf{B}, \epsilon)$.

6. If $(\mathrm{grade}(\mathbf{J}) = n)$ or $(\mathrm{grade}(\mathbf{J}) = \mathrm{grade}(\mathbf{A}) + \mathrm{grade}(\mathbf{B}))$, return $\mathbf{J}$. Otherwise:
7. Compute $\mathrm{grade}(\mathbf{A} \cup \mathbf{B})$ using the delta product.
8. While $(\mathrm{grade}(\mathbf{J}) < \mathrm{grade}(\mathbf{A} \cup \mathbf{B}))$
   (a) For all valid i, compute $\mathbf{b}_i = (\mathbf{f}_i \rfloor \mathbf{F}^{-1}) \rfloor \mathbf{B}$.
   Set $\mathbf{b}_m$ to that $\mathbf{b}_i$ which leads to the largest $\|\mathbf{J} \wedge \mathbf{b}_i\|$.
   (b) Update $\mathbf{J} \leftarrow \mathbf{J} \wedge \mathbf{b}_m$.
9. Return $\mathbf{J}$.

UNIVERSITEIT VAN AMSTERDAM

FastJoin Implementation

Code generation is used to generate the core of FastJoin.
Example of a generated function (step 5b/5c of algorithm):

```
void factorAndOuterProductE35G3(const float *J, const float *B, float *H) {
  H[0] = J[3] * B[5] - J[2] * B[6] + J[0] * B[9];
  H[1] = J[6] * B[5] - J[5] * B[6];
  H[2] = J[8] * B[5] - J[7] * B[6] - J[4] * B[9];
  H[3] = J[9] * B[5] - J[5] * B[9];
  H[4] = J[9] * B[6] - J[6] * B[9];
  return B[5] * B[5] + B[6] * B[6] + B[9] * B[9];
}
```

UNIVERSITEIT
VAN
AMSTERDAM

FastJoin Implementation

Code generation is used to generate the core of FastJoin.
Example of a generated function (step 5b/5c of algorithm):

```
void factorAndOuterProductE35G3(const float *J, const float *B, float *H) {
  H[0] = J[3] * B[5] - J[2] * B[6] + J[0] * B[9];
  H[1] = J[6] * B[5] - J[5] * B[6];
  H[2] = J[8] * B[5] - J[7] * B[6] - J[4] * B[9];
  H[3] = J[9] * B[5] - J[5] * B[9];
  H[4] = J[9] * B[6] - J[6] * B[9];
  return B[5] * B[5] + B[6] * B[6] + B[9] * B[9];
}
```

Implementation of the delta product is optimized using code generation and lazy evaluation.

UNIVERSITEIT VAN AMSTERDAM

FastJoin Implementation

Code generation is used to generate the core of FastJoin.
Example of a generated function (step 5b/5c of algorithm):

```
void factorAndOuterProductE35G3(const float *J, const float *B, float *H) {
  H[0] = J[3] * B[5] - J[2] * B[6] + J[0] * B[9];
  H[1] = J[6] * B[5] - J[5] * B[6];
  H[2] = J[8] * B[5] - J[7] * B[6] - J[4] * B[9];
  H[3] = J[9] * B[5] - J[5] * B[9];
  H[4] = J[9] * B[6] - J[6] * B[9];
  return B[5] * B[5] + B[6] * B[6] + B[9] * B[9];
}
```

Implementation of the delta product is optimized using code generation and lazy evaluation.

The approach is limited to 6-D due to code size!
Above 6-D a conventional (hand-written) approach can be used (about $2\times$ slower).

UNIVERSITEIT VAN AMSTERDAM

FastJoin Benchmarks

Benchmark: Compute the join of millions of random blades. Pairs of blades were generated such that they shared a common factor of a random grade.

Used one CPU on a Core2Duo 1.83Ghz.
Compiled using VS2005.

UNIVERSITEIT VAN AMSTERDAM

FastJoin Benchmarks

Benchmark: Compute the join of millions of random blades. Pairs of blades were generated such that they shared a common factor of a random grade.

Used one CPU on a Core2Duo 1.83Ghz.
Compiled using VS2005.

n	3	4	5	6
FastJoin (absolute)	7.4M	5.4M	3.1M	1.8M
FastJoin (relative)	9.8×	8.7×	5.8×	6.4×
StableFastJoin (absolute)	7.4M	5.2M	2.6M	1.6M
StableFastJoin (relative)	9.8×	9.1×	7.0×	6.8×
Gram-Schmidt (relative)	12×	12×	7.9×	8.0×

UNIVERSITEIT
VAN
AMSTERDAM

The Meet

The meet can be directedly computed by factorizing the dual of the delta product.

UNIVERSITEIT VAN AMSTERDAM

The Meet

The meet can be directedly computed by factorizing the dual of the delta product.

But:
-Expensive full evaluation of delta product is always required.
-Generated code is larger.

UNIVERSITEIT VAN AMSTERDAM

Discussion / Summary

- Fastest possible factorization algorithm?

UNIVERSITEIT VAN AMSTERDAM

Discussion / Summary

- Fastest possible factorization algorithm?
- The join $10\times$ faster and still numerically stable. Still some possibility for improvement.

UNIVERSITEIT VAN AMSTERDAM

Discussion / Summary

- Fastest possible factorization algorithm?
- The join $10\times$ faster and still numerically stable. Still some possibility for improvement.