Rewriting with an index-based intermediate representation

Charisee Chiw and John Reppy

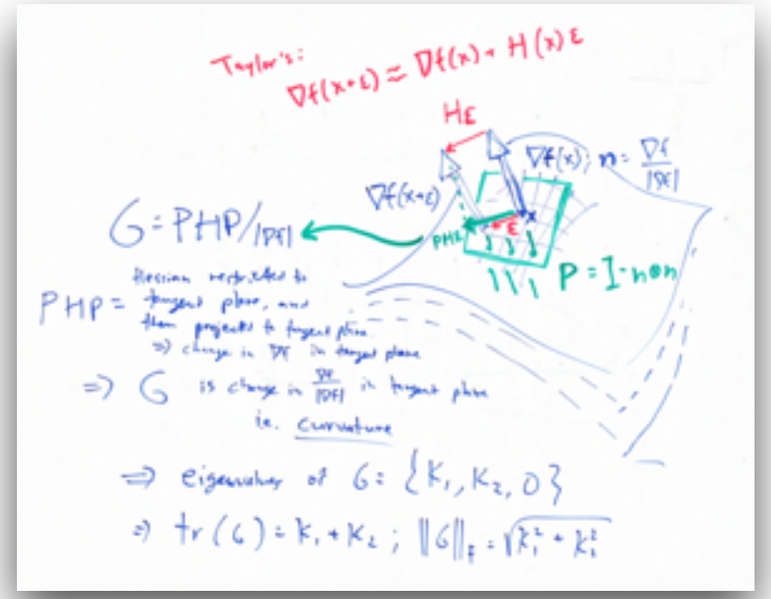
THE UNIVERSITY OF CHICAGO
COMPUTER SCIENCE

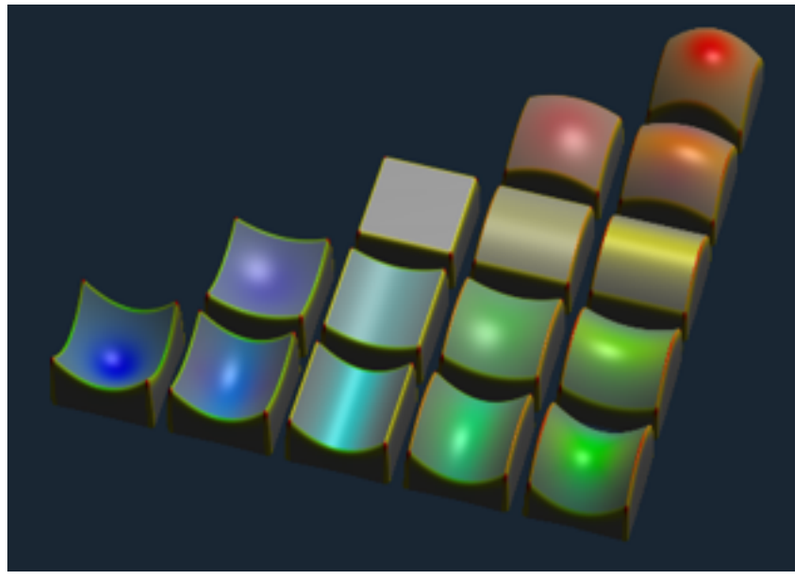
Roadmap

- Domain Space
- EIN operators as a compact representation of loops over tensor indices (used for tensor math)
- Index-based optimizations
- Potential for loop optimization

Domain

- Diderot; DSL for scivis and image analysis [PLDI'12]
- Scientists use algorithms to explore image space and understand data
- Expressive language for tensor and field computations

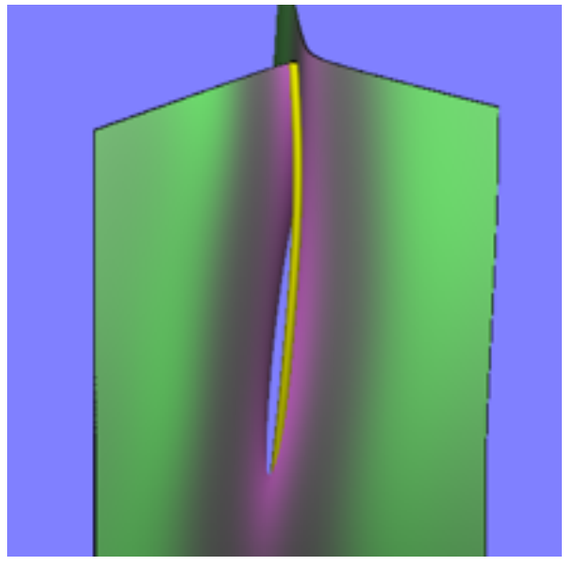


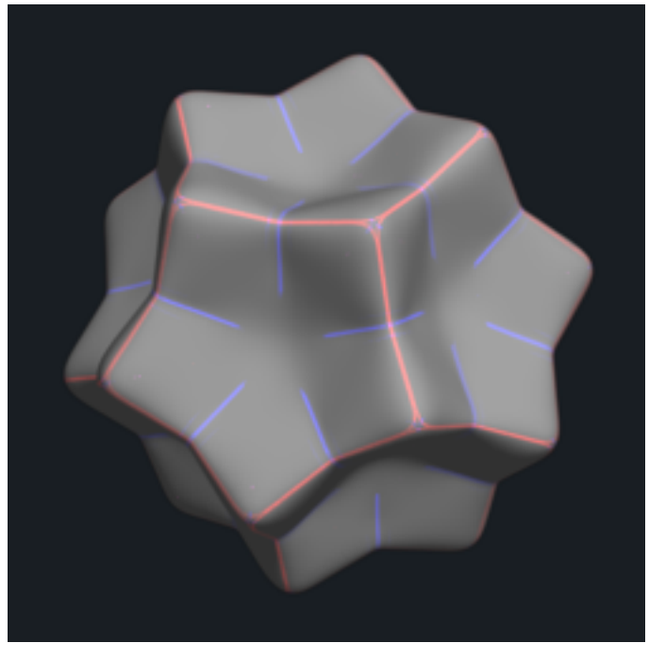


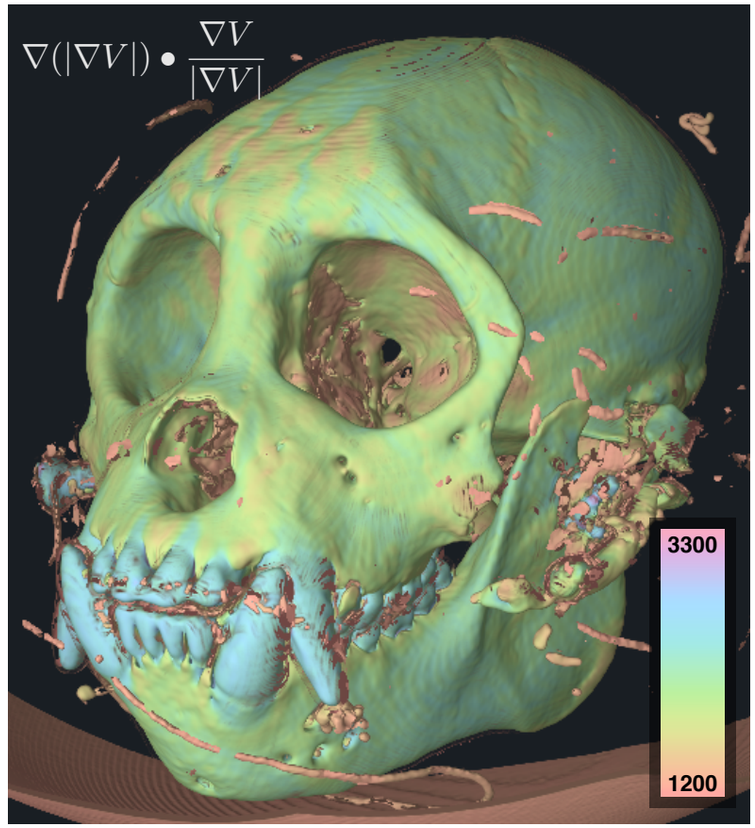
Domain

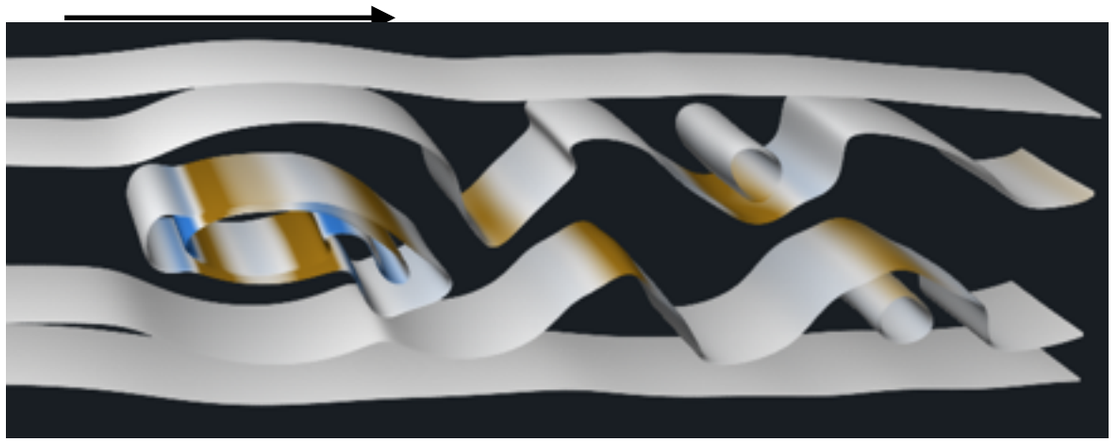
Sophisticated Visualization algorithms

Algorithm in this space require a lot of tensor math









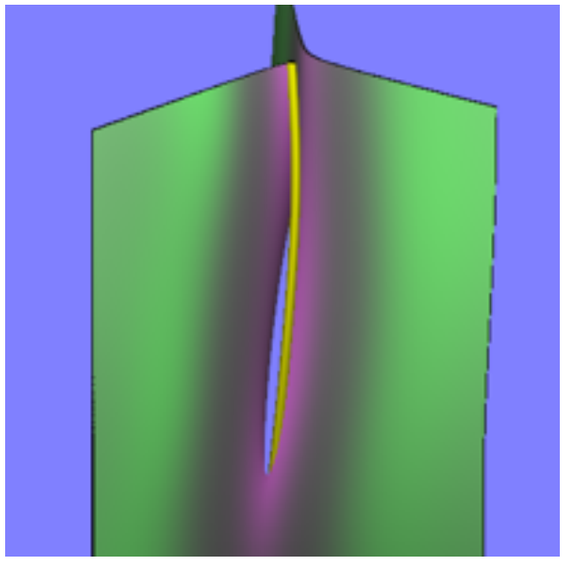
Provided by GK

[VIS ’15, submitted EuroVis’18]

Domain

Implementing Sophisticated Visualization algorithms

Visualization



computation

E = $\nabla \otimes \nabla$F - identity[3]*trace($\nabla \otimes \nabla$ F)/3;

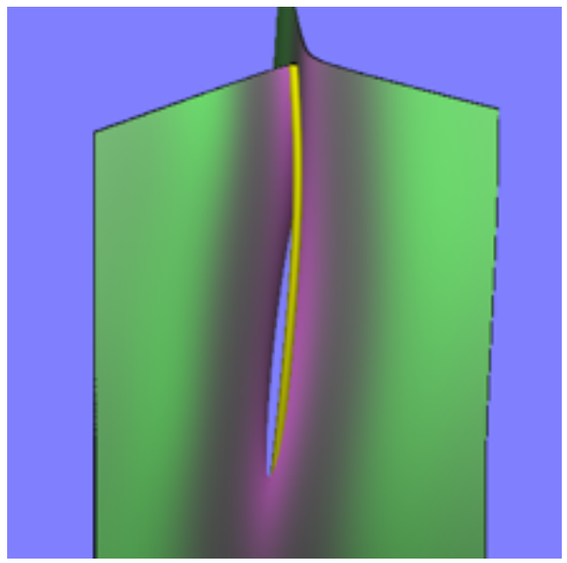
D = 3 $\sqrt{6} * det(E/|E|)$;

T = $\nabla \otimes \nabla$ D

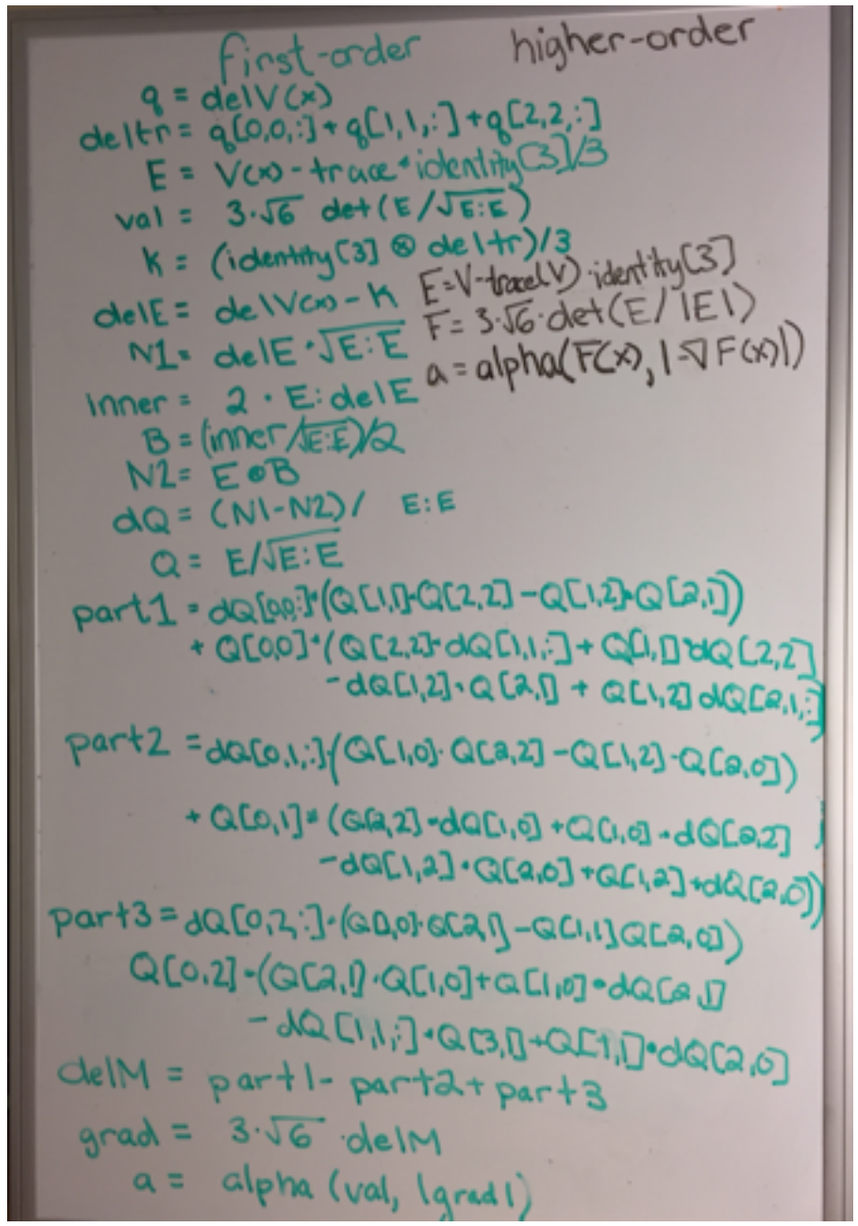
Domain

Implementing Sophisticated Visualization algorithms

Visualization



User does derivations



computation

$E = \nabla \otimes \nabla F$ - identity[3]*trace($\nabla \otimes \nabla F$)/3;

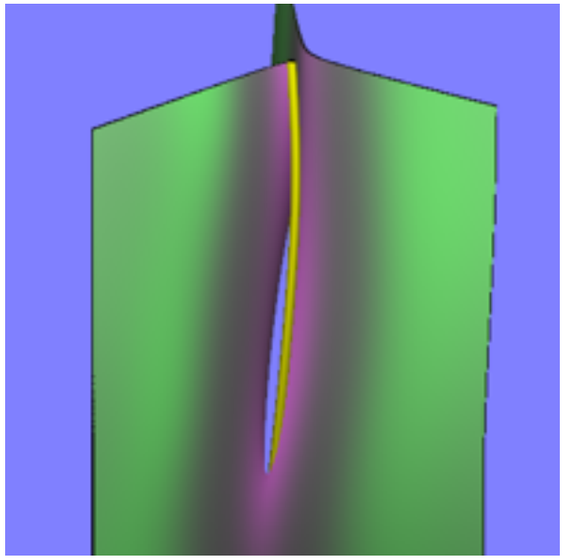
$D = 3\sqrt{6} * det(E/|E|)$;

$T = \nabla \otimes \nabla D$

Domain

Implementing Sophisticated Visualization algorithms

Visualization



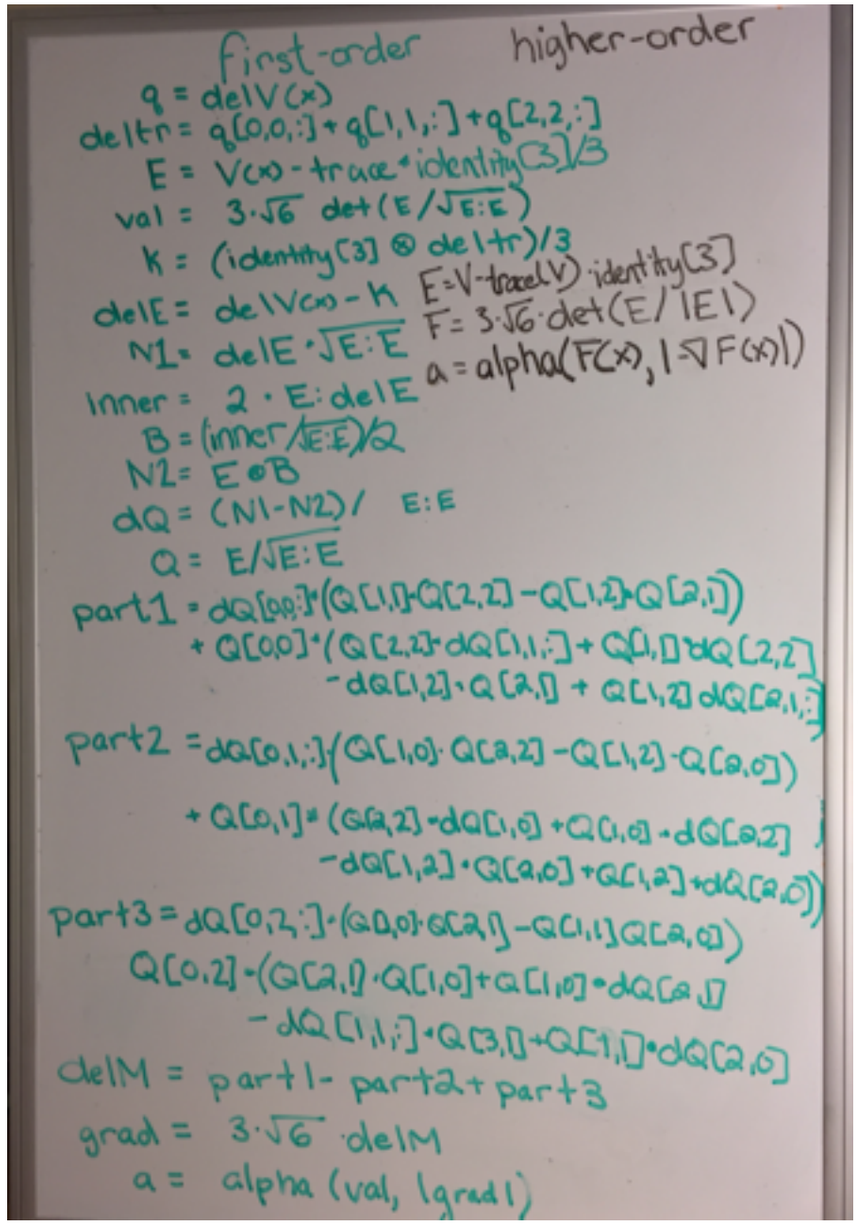
computation

$E = \nabla \otimes \nabla F - identity[3]*trace(\nabla \otimes \nabla F)/3;$

$D = 3\sqrt{6} * det(E/|E|);$

$T = \nabla \otimes \nabla D$

User does derivations

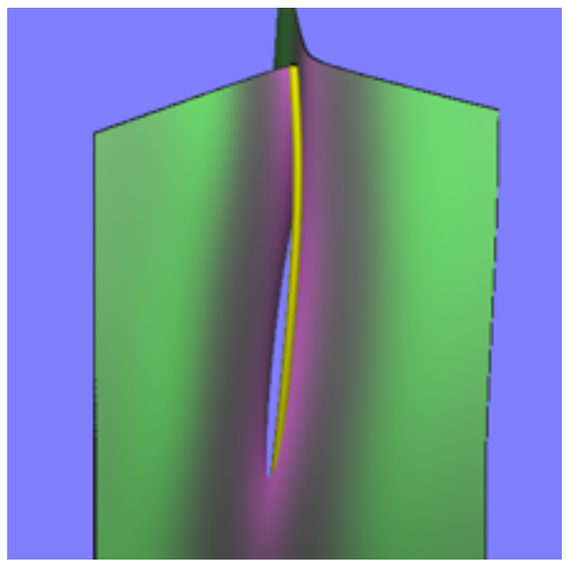


Diderot Program ‘12

Domain

Implementing Sophisticated Visualization algorithms

Visualization



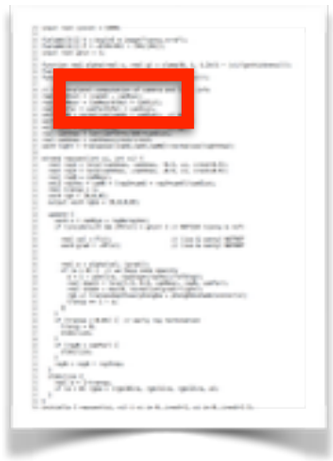
computation

E = $\nabla \otimes \nabla$F - identity[3]*trace($\nabla \otimes \nabla$ F)/3;

D = 3 $\sqrt{6} * det(E/|E|)$;

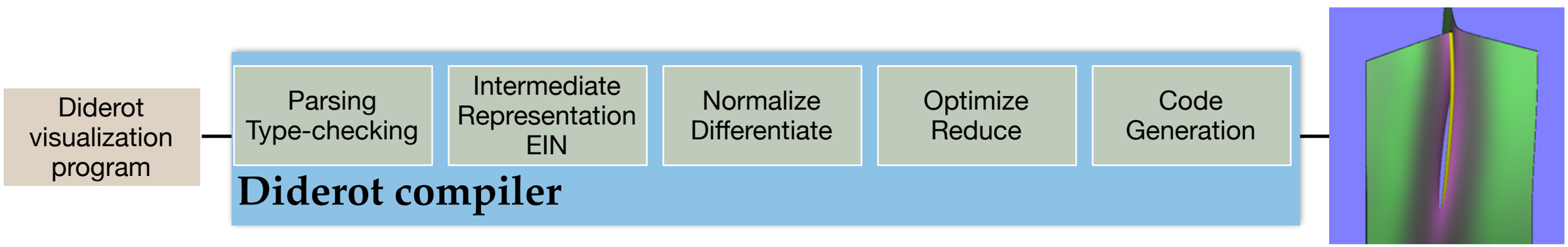
T = $\nabla \otimes \nabla$ D

New and improved Diderot program



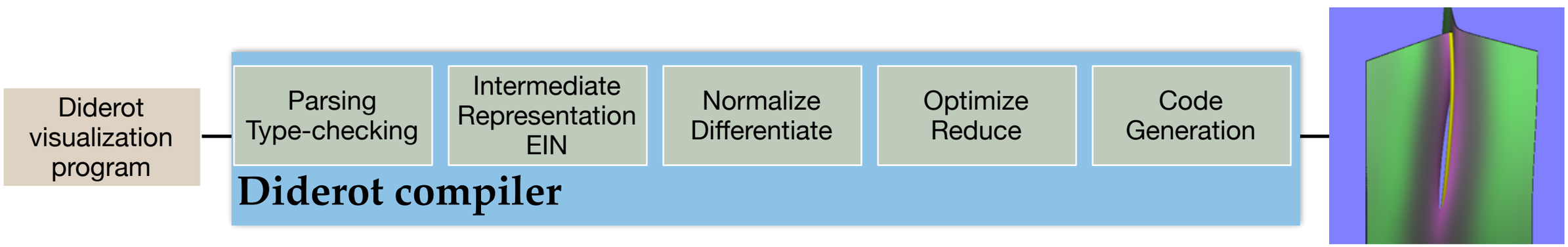
fourth derivatives

From Tensor calculus to C



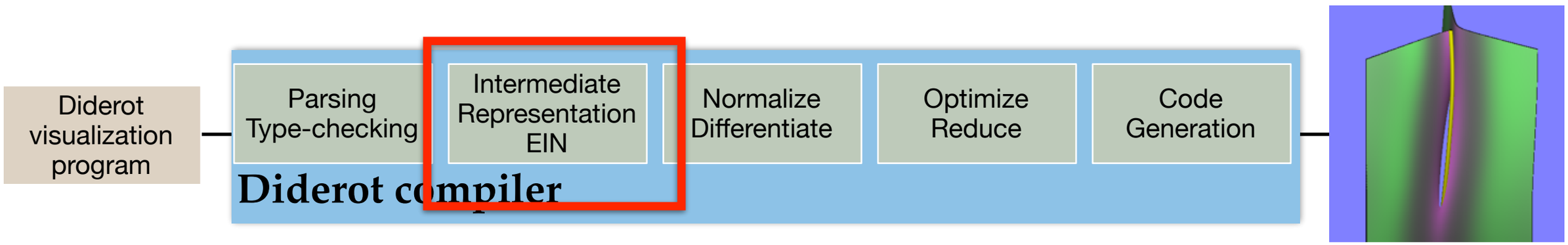
From Tensor calculus to C

- Diderot offers ability to write computations at a high level



From Tensor calculus to C

- Diderot offers ability to write computations at a high level
- Enabled by design and implementation of EIN IR [Masters '14]



Rewriting with EIN

- Hybrid design that embeds expression trees into a normalized SSA representation

$$X \;\; Y \;\; Z$$

Rewriting with EIN

- Hybrid design that embeds expression trees into a normalized SSA representation

$$X_\alpha Y_\beta Z_\beta$$

- Indices on tensor and field argument illustrate sampling

Rewriting with EIN

- Hybrid design that embeds expression trees into a normalized SSA representation

$$\sum_{\beta} X_{\alpha} Y_{\beta} Z_{\beta}$$

- Indices on tensor and field argument illustrate sampling
- Summation operator represents a loop nest that will be unrolled later

Rewriting with EIN

- Hybrid design that embeds expression trees into a normalized SSA representation

$$\sum_{\beta} X_{\alpha} Y_{\beta} Z_{\beta} \rightarrow X_{\alpha} \sum_{\beta} Y_{\beta} Z_{\beta}$$

- Indices on tensor and field argument illustrate sampling
- Summation operator represents a loop nest that will be unrolled later
- Moving terms outside can avoid subsequent code duplication

Intermediate Representation: EIN

The matrix M is represented in EIN with two variable indices

Matrix M $\begin{bmatrix} M_{11} M_{12} \\ M_{21} M_{22} \end{bmatrix} = M_{ij}$ EIN Term

Intermediate Representation: EIN

The matrix M is represented in EIN with two variable indices

Matrix M $\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = M_{ij}$ EIN Term

Switch row and column $\begin{bmatrix} M_{11} & M_{21} \\ M_{12} & M_{22} \end{bmatrix} = M_{ji}$ Switch the order

Intermediate Representation: EIN

The matrix M is represented in EIN with two variable indices

Matrix M $\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = M_{ij}$ EIN Term

Switch row and column $\begin{bmatrix} M_{11} & M_{21} \\ M_{12} & M_{22} \end{bmatrix} = M_{ji}$ Switch the order

Intermediate Representation: EIN

The matrix M is represented in EIN with two variable indices

Matrix M	$\begin{bmatrix} M_{11} M_{12} \\ M_{21} M_{22} \end{bmatrix} = M_{ij}$	EIN Term
Switch row and column	$\begin{bmatrix} M_{11} M_{21} \\ M_{12} M_{22} \end{bmatrix} = M_{ji}$	Switch the order
Select a column	$\begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix} = M_{i1}$	Mix of different indices
Single entry	$M_{12} = M_{12}$	Set indices to constants
Add diagonal components	$trace(M) = \sum_i M_{ii}$	Summation

Intermediate Representation: EIN

The matrix M is represented in EIN with two variable indices

Matrix M $\begin{bmatrix} M_{11} M_{12} \\ M_{21} M_{22} \end{bmatrix} = M_{ij}$ EIN Term

Intermediate Representation: EIN

The matrix M is represented in EIN with two variable indices

Matrix M $\begin{bmatrix} M_{11} M_{12} \\ M_{21} M_{22} \end{bmatrix} = M_{ij}$ EIN Term

Matrix Inverse $\mathbf{inv(M)} = \frac{\sum_k M_{kk}\delta_{ij} - M_{ij}}{M_{00}M_{11} - M_{01}M_{10}}$

Matrix Determinant $\mathbf{det(M)} = \sum_{ijk} M_{0i}M_{1j}M_{2k}\mathcal{E}_{ijk}$

Intermediate Representation: EIN

- Operation between terms

$$\sum_i A_i B_i \qquad A_i B_i \qquad A_i B_j$$

Intermediate Representation: EIN

- Operation between terms

real	**vector**	**matrix**
$\sum_i A_i B_i$	$A_i B_i$	$A_i B_j$
dot product	component-wise multiplication	outer product

Intermediate Representation: EIN

- Operation between terms

real

$$\sum_i A_i B_i$$

dot product

vector

$$A_i B_i$$

component-wise multiplication

matrix

$$A_i B_j$$

outer product

- Can apply mix of different operators to each other:
  composition, curl, concatenation, Hessian, Transpose, Double dot product

$$\nabla(F \circ G)$$

$$\nabla \times$$

$$Cons[T[:0], T[1:]]$$

$$\nabla \otimes \nabla((A : B) \times C)$$

Intermediate Representation: EIN

- Operation between terms

real

$$\sum_i A_i B_i$$

dot product

vector

$$A_i B_i$$

component-wise multiplication

matrix

$$A_i B_j$$

outer product

- Can apply mix of different operators to each other:
composition, curl, concatenation, Hessian, Transpose, Double dot product

$$\nabla(F \circ$$

EIN supports a rich language for tensor and field computations

$$\nabla \times$$

$$Cons[T[:0], T[1:]]$$

$$\nabla \otimes \nabla((A:B) \times C)$$

Intermediate Representation: EIN

Consider the computation

$$t = \texttt{trace}(a \otimes b)$$

Intermediate Representation: EIN

Consider the computation

$$t = \texttt{trace}(a \otimes b)$$

Diderot expression is represented in compiler as

$$M = (\lambda\,(A, B)\langle A_i B_j \rangle_{ij})(a, b)$$
$$t = (\lambda\, T\langle \textstyle\sum_i T_{ii} \rangle)(M)$$

Intermediate Representation: EIN

Consider the computation

$$t = \texttt{trace}(a \otimes b)$$

Diderot expression is represented in compiler as

$$M = (\lambda\,(A, B)\langle A_i B_j \rangle_{ij})(a, b)$$
$$t = (\lambda\, T\langle \textstyle\sum_i T_{ii} \rangle)(M)$$

substitution of the definition of M for X yields

$$t = (\lambda\,(A, B)\langle \textstyle\sum_i A_i B_i \rangle)(a, b)$$

Intermediate Representation: EIN

Consider the computation

$$t = \texttt{trace}(a \otimes b)$$

Diderot expression is represented in compiler as

$$M = (\lambda\,(A, B)\langle A_i B_j \rangle_{ij})(a, b)$$
$$t = (\lambda\, T\langle \textstyle\sum_i T_{ii} \rangle)(M)$$

substitution of the definition of M for X yields

$$t = (\lambda\,(A, B)\langle \textstyle\sum_i A_i B_i \rangle)(a, b)$$

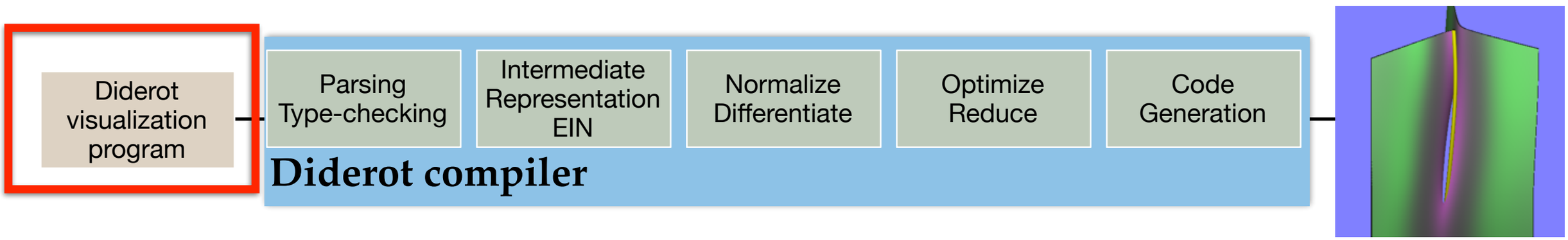
Replaces the ad hoc rewrite rule:

$$\mathrm{Trace}(\mathrm{Outer}(a, b)) \Rightarrow \mathrm{Dot}(a, b)$$

From Tensor calculus to C

Core Computation in visualization program

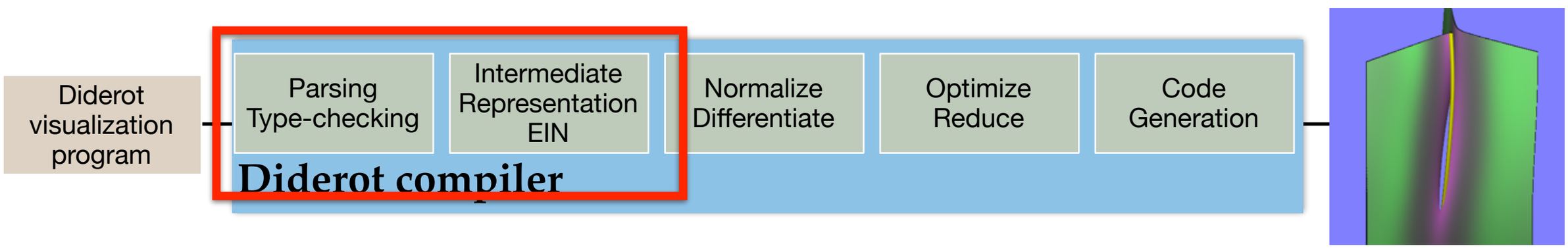
E = $\nabla \otimes \nabla$F - identity[3]*trace($\nabla \otimes \nabla$ F)/3;
D = 3 $\sqrt{6} * det(E/|E|)$;
T = $\nabla \otimes \nabla$ D



From Tensor calculus to C

Core Computation in visualization program

E = $\nabla \otimes \nabla$F - identity[3]*trace($\nabla \otimes \nabla$ F)/3;
D = 3 $\sqrt{6} * det(E/|E|)$;
T = $\nabla \otimes \nabla$ D



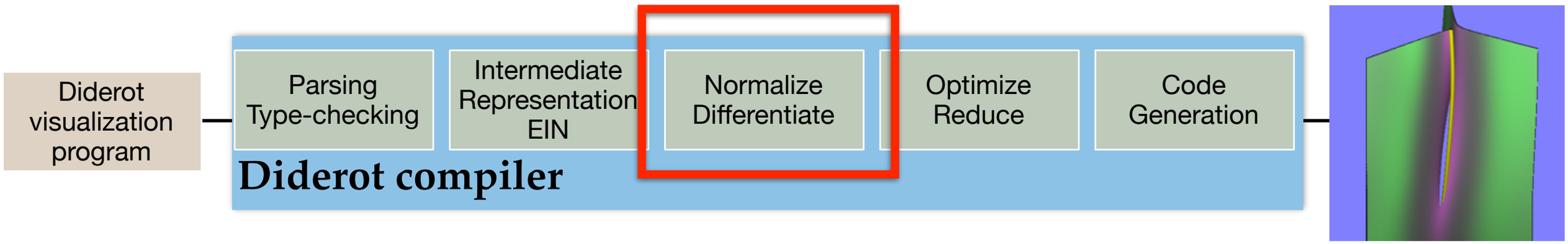
From Tensor calculus to C

Core Computation in visualization program

E = $\nabla \otimes \nabla$F - identity[3]*trace($\nabla \otimes \nabla$ F)/3;
D = 3 $\sqrt{6} * det(E/|E|)$;
T = $\nabla \otimes \nabla$ D

Necessary Rewriting

$$\nabla|\nabla F|(x) \longrightarrow \frac{1}{2}\frac{(\nabla\otimes\nabla F(p)\bullet\nabla F(p))+(\nabla F(p)\bullet\nabla\otimes\nabla F(p))}{\sqrt{\nabla F(p)\bullet\nabla F(p)}}$$



From Tensor calculus to C

Core Computation in visualization program

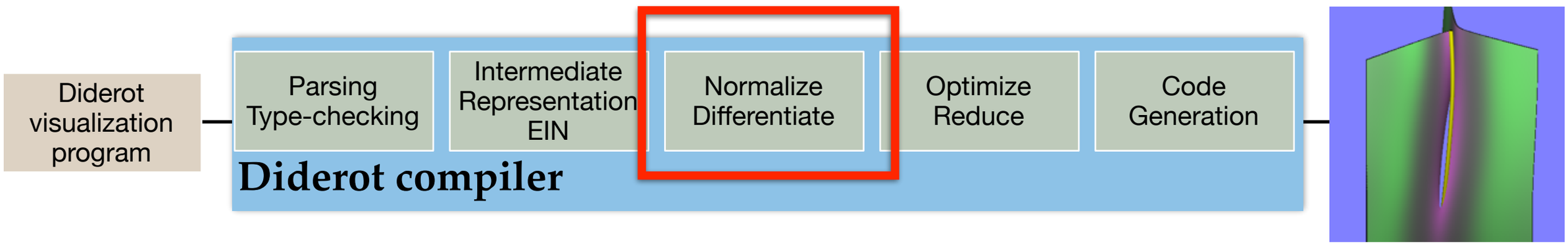
E = $\nabla \otimes \nabla$F - identity[3]*trace($\nabla \otimes \nabla$ F)/3;
D = 3 $\sqrt{6} * det(E/|E|)$;
T = $\nabla \otimes \nabla$ D

Necessary Rewriting

$$\nabla|\nabla F|(x) \longrightarrow \frac{1}{2}\frac{(\nabla\otimes\nabla F(p)\bullet\nabla F(p))+(\nabla F(p)\bullet\nabla\otimes\nabla F(p))}{\sqrt{\nabla F(p)\bullet\nabla F(p)}}$$

The Issue

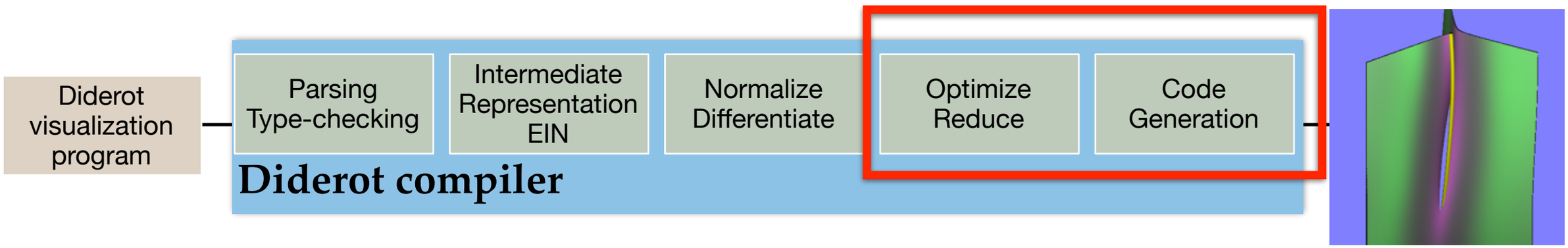
- Rich Language + necessary rewriting = bulky obtuse IR
- It becomes difficult to compile and generate efficient code



From Tensor calculus to C

Index Based Rewriting

- Shift invariant terms
- Split EIN computations into SSA representation based on EIN structure
- Examine Structure and vectorize



1. Shift invariant terms

User writes high-level computation

$$out = (s*(v \otimes x)) \bullet u$$

1. Shift invariant terms

User writes high-level computation

$$\text{out} = (s*(v \otimes x)) \bullet u$$

Internally represented with EIN operator

$$\text{out} = \left\langle \sum_j (s v_i x_j u_j) \right\rangle_i$$

1. Shift invariant terms

User writes high-level computation

$$\text{out} = (s*(v \otimes x)) \bullet u$$

Internally represented with EIN operator

$$\text{out} = \left\langle \sum_j (s v_i x_j u_j) \right\rangle_i \overset{\text{Shift}}{\Rightarrow} \left\langle s v_i \sum_j (x_j u_j) \right\rangle_i$$

1. Shift invariant terms

User writes high-level computation

$$\text{out} = (s*(v \otimes x)) \bullet u$$

Internally represented with EIN operator

$$\text{out} = \left\langle \sum_j (s v_i x_j u_j) \right\rangle_i \overset{\text{Shift}}{\Rightarrow} \left\langle s v_i \sum_j (x_j u_j) \right\rangle_i$$

Operators are split apart based on EIN structure

$$\overset{\text{Split}}{\Rightarrow} \begin{aligned} \text{out} &= \langle s v_i d \rangle_i \\ d &= \left\langle \sum_j x_j u_j \right\rangle \end{aligned}$$

1. Shift invariant terms

User writes high-level computation

$$\text{out} = (s*(v \otimes x)) \bullet u$$

Internally represented with EIN operator

$$\text{out} = \left\langle \sum_j (s v_i x_j u_j) \right\rangle_i \overset{\text{Shift}}{\Rightarrow} \left\langle s v_i \sum_j (x_j u_j) \right\rangle_i$$

Operators are split apart based on EIN structure

$$\overset{\text{Split}}{\Rightarrow} \quad \begin{aligned} \text{out} &= \langle s v_i d \rangle_i \\ d &= \left\langle \sum_j x_j u_j \right\rangle \end{aligned}$$

vector operation

1. Shift invariant terms

User writes high-level computation

Replace a direct-style rewrite

$$\text{out} = (s*(v \otimes x)) \bullet u \rightarrow s*(v*(x \bullet u))$$

Internally represented with EIN operator

$$\text{out} = \left\langle \sum_j (s v_i x_j u_j) \right\rangle_i \overset{\text{Shift}}{\Rightarrow} \left\langle s v_i \sum_j (x_j u_j) \right\rangle_i$$

Operators are split apart based on EIN structure

$$\overset{\text{Split}}{\Rightarrow} \quad \begin{aligned} \text{out} &= \langle s v_i d \rangle_i \\ d &= \left\langle \sum_j x_j u_j \right\rangle \end{aligned}$$

vector operation

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$

User writes high-level computation

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$ User writes high-level computation

$\Rightarrow \sum_i (\sum_{jk} \mathcal{E}_{ijk} a_j b_k)(\sum_{lm} \mathcal{E}_{ilm} c_l d_m)$ EIN operator

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$

User writes high-level computation

$$\Rightarrow \sum_{i}\Big(\sum_{jk}\mathcal{E}_{ijk}a_j b_k\Big)\Big(\sum_{lm}\mathcal{E}_{ilm}c_l d_m\Big)$$

EIN operator

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$

User writes high-level computation

$$\Rightarrow \sum_{i} (\sum_{jk} \mathcal{E}_{ijk} a_j b_k)(\sum_{lm} \mathcal{E}_{ilm} c_l d_m)$$

EIN operator

$$\Rightarrow (\sum_{jklm} \delta_{jl}\delta_{km} a_j b_k c_l d_m) - \sum_{jklm} (\delta_{jm}\delta_{kl} a_j b_k c_l d_m)$$

Epsilons with shared index

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$

User writes high-level computation

$\Rightarrow \sum_{i} (\sum_{jk} \mathcal{E}_{ijk} a_j b_k)(\sum_{lm} \mathcal{E}_{ilm} c_l d_m)$

EIN operator

$\Rightarrow (\sum_{jklm} \delta_{jl}\delta_{km} a_j b_k c_l d_m) - \sum_{jklm} (\delta_{jm}\delta_{kl} a_j b_k c_l d_m)$

Epsilons with shared index

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$ User writes high-level computation

$\Rightarrow \sum_{i} (\sum_{jk} \mathcal{E}_{ijk} a_j b_k)(\sum_{lm} \mathcal{E}_{ilm} c_l d_m)$ EIN operator

$\Rightarrow (\sum_{jklm} \delta_{jl}\delta_{km} a_j b_k c_l d_m) - \sum_{jklm} (\delta_{jm}\delta_{kl} a_j b_k c_l d_m)$ Epsilons with shared index

$\Rightarrow (\sum_{jk} a_j b_k c_j d_k) - \sum_{jk} (a_j b_k c_k d_j)$ Delta Reduction

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$ User writes high-level computation

$\Rightarrow \sum_{i}(\sum_{jk} \mathcal{E}_{ijk} a_j b_k)(\sum_{lm} \mathcal{E}_{ilm} c_l d_m)$ EIN operator

$\Rightarrow (\sum_{jklm} \delta_{jl}\delta_{km} a_j b_k c_l d_m) - \sum_{jklm} (\delta_{jm}\delta_{kl} a_j b_k c_l d_m)$ Epsilons with shared index

$\Rightarrow (\sum_{jk} a_j b_k c_j d_k) - \sum_{jk} (a_j b_k c_k d_j)$ Delta Reduction

Shift

$\Rightarrow (\sum_{j} a_j c_j)(\sum_{k} b_k d_k) - (\sum_{j} a_j d_j)(\sum_{k} b_k c_k)$ Summation binding

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d)$ User writes high-level computation

$\Rightarrow \sum_{i} (\sum_{jk} \mathcal{E}_{ijk} a_j b_k)(\sum_{lm} \mathcal{E}_{ilm} c_l d_m)$ EIN operator

$\Rightarrow (\sum_{jklm} \delta_{jl}\delta_{km} a_j b_k c_l d_m) - \sum_{jklm} (\delta_{jm}\delta_{kl} a_j b_k c_l d_m)$ Epsilons with shared index

$\Rightarrow (\sum_{jk} a_j b_k c_j d_k) - \sum_{jk} (a_j b_k c_k d_j)$ Delta Reduction

Shift

$\Rightarrow (\sum_{j} a_j c_j)(\sum_{k} b_k d_k) - (\sum_{j} a_j d_j)(\sum_{k} b_k c_k)$ Summation binding

vector operation

2. Separate a large loop

Rewrites allow us to eliminate summations

(i.e., effectively reduce the loop-nesting depth)

$(a \times b) \bullet (c \times d) \Rightarrow (a \bullet c)(b \bullet d) - (a \bullet d)(b \bullet c)$ User writes high-level computation

Replace a direct-style rewrite

$\Rightarrow \sum_{i} (\sum_{jk} \mathcal{E}_{ijk} a_j b_k)(\sum_{lm} \mathcal{E}_{ilm} c_l d_m)$ EIN operator

$\Rightarrow (\sum_{jklm} \delta_{jl}\delta_{km} a_j b_k c_l d_m) - \sum_{jklm} (\delta_{jm}\delta_{kl} a_j b_k c_l d_m)$ Epsilons with shared index

$\Rightarrow (\sum_{jk} a_j b_k c_j d_k) - \sum_{jk} (a_j b_k c_k d_j)$ Delta Reduction

Shift

$\Rightarrow (\sum_{j} a_j c_j)(\sum_{k} b_k d_k) - (\sum_{j} a_j d_j)(\sum_{k} b_k c_k)$ Summation binding

vector operation

3. Outer loop decision

User writes high-level computation (c • d*a)• b

3. Outer loop decision

User writes high-level computation (c • d*a)• b

Internally represented with EIN operator

$$e = \sum_{ij} AB_i C_j D_{ji}$$

3. Outer loop decision

User writes high-level computation (c • d*a)• b

Internally represented with EIN operator

$$e = \sum_{ij} AB_i C_j D_{ji}$$

Two possible rewrites. Which expression is best?

$$e \xrightarrow{R2} A \sum_j C_j \sum_i D_{ji} B_i \qquad e \xrightarrow{R1} A \sum_i (B_i \sum_j C_j D_{ji})$$

3. Outer loop decision

User writes high-level computation (c • d*a)• b

Internally represented with EIN operator

$$e = \sum_{ij} AB_i C_j D_{ji}$$

Two possible rewrites. Which expression is best?

$$e \xrightarrow{R2} A \sum_j C_j \sum_i \underline{D_{ji} B_i}$$

Row Projection

$$e \xrightarrow{R1} A \sum_i (B_i \sum_j \underline{C_j D_{ji}})$$

Column Projection

3. Outer loop decision

User writes high-level computation (c • d*a)• b

Internally represented with EIN operator

$$e = \sum_{ij} AB_i C_j D_{ji}$$

Two possible rewrites. Which expression is best?

$$e \xrightarrow{R2} A \sum_j C_j \sum_i \underline{D_{ji} B_i}$$

Row Projection

$$e \xrightarrow{R1} A \sum_i (B_i \sum_j \underline{C_j D_{ji}})$$

Column Projection

Split

$$t_0 = \lambda(f,d)\left\langle \sum_i D_{ji} B_i \right\rangle_j \text{(f,d)}$$

$$\text{out} = \lambda(c,e,t)\left\langle A \sum_j C_j t_j \right\rangle \text{(e,}t_0\text{)}$$

3. Outer loop decision

User writes high-level computation (c • d*a)• b

Internally represented with EIN operator

$$e = \sum_{ij} AB_i C_j D_{ji}$$

Two possible rewrites. Which expression is best?

$$e \xrightarrow{R2} A \sum_j C_j \sum_i \underline{D_{ji} B_i}$$

Row Projection

$$e \xrightarrow{R1} A \sum_i (B_i \sum_j \underline{C_j D_{ji}})$$

Column Projection

Split

$$t_0 = \lambda(f,d)\left\langle \underline{\textstyle\sum_i D_{ji} B_i} \right\rangle_j \text{(f,d)}$$

$$\text{out} = \lambda(c,e,t)\left\langle \underline{A \textstyle\sum_j C_j t_j} \right\rangle \text{(e,}t_0\text{)}$$

vector operation

The big question

- Can we expand on this approach?
- Is splitting based on the EIN structure the best thing?
- Could we take a step back and look at bigger mathematical structure?
- Performance?

Conclusion

- EIN operators as a compact representation of loops over tensor indices (used for tensor math)
- Invariant terms can be moved in/out of loops with simple pattern matching, rewriting, and analysis
- In the future we want to examine the entire computation earlier in the computation

Thank you

Github Page
https://github.com/Diderot-Language/examples

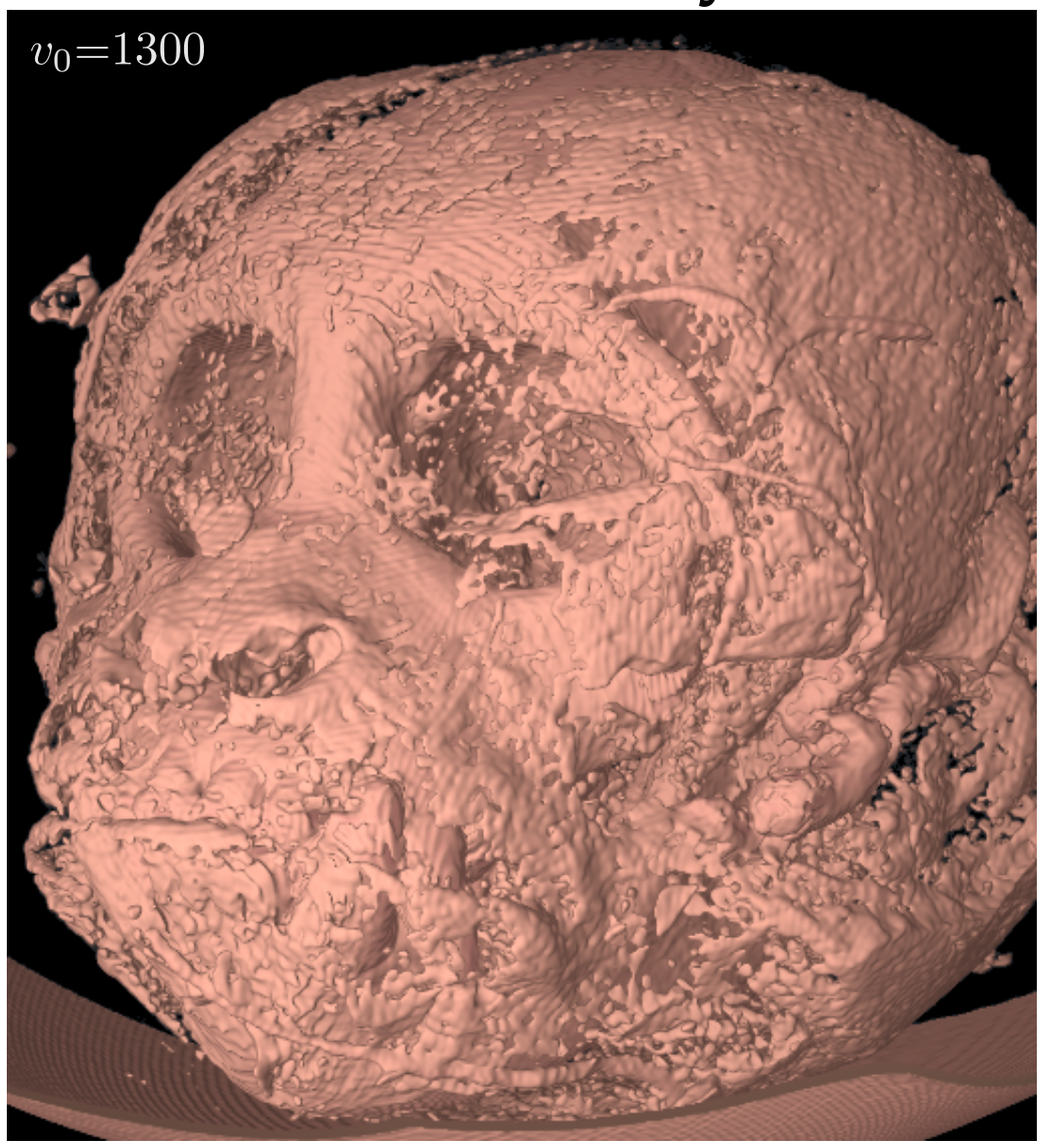
Diderot
http://diderot-language.cs.uchicago.edu/

Acknowledgement: This material is based upon work supported by the NSF under Grants CCF-1446412 and CCF-1564298

Extra Slides

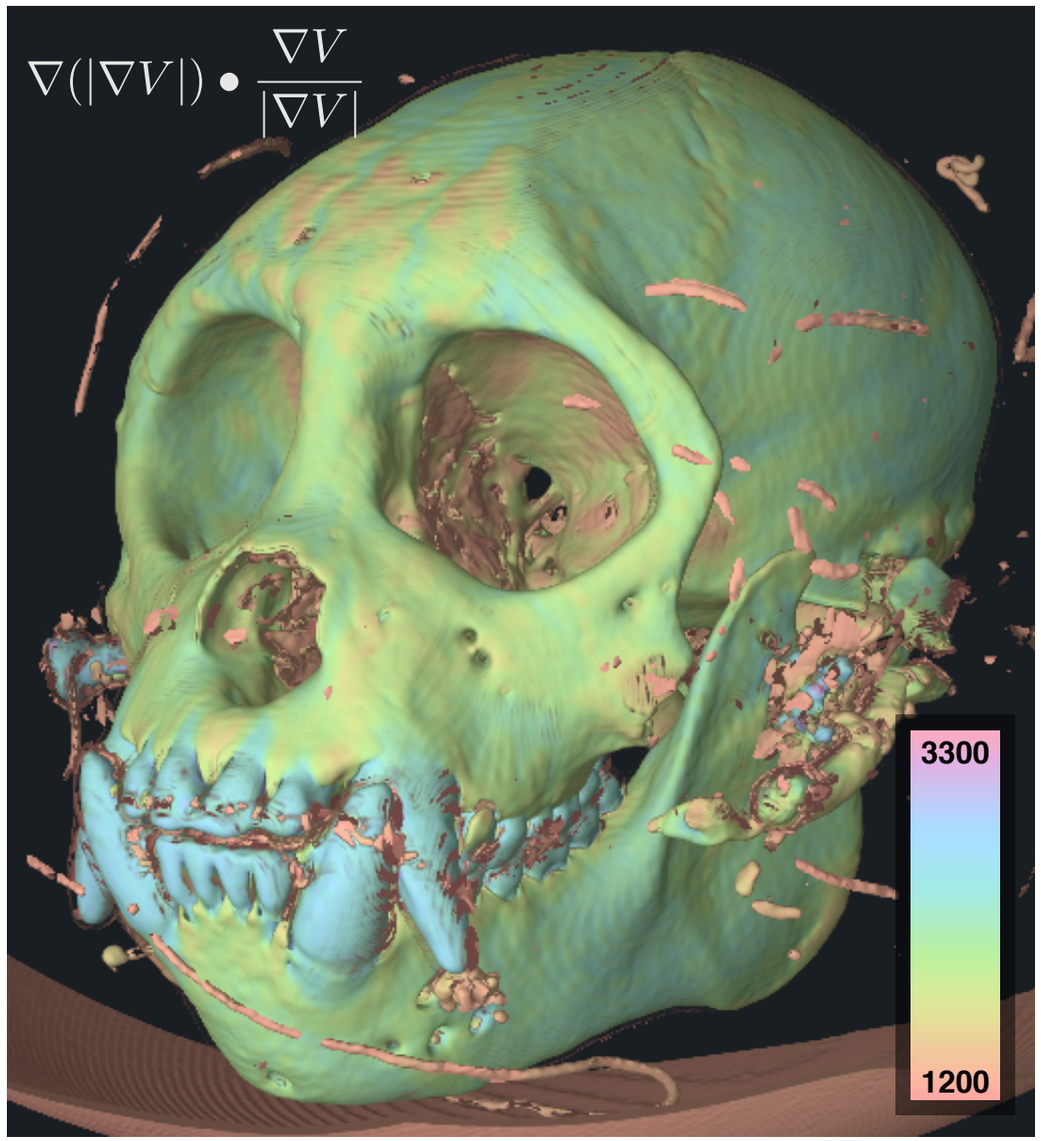
Results: Sophisticated Vis

Previously:



Iso-Surface

Now:



Canny Edges

Data provided by Callum Ross (U of Chicago)

Index based optimizations

Rewrites allow us to eliminate summations (i.e., effectively reduce the loop-nesting depth)

$$\nabla \bullet \nabla \times F \Rightarrow 0$$

$$\sum_{jk} \mathcal{E}_{ijk} \left(\frac{\partial}{\partial x_{ij}} \diamond e \right) \Rightarrow 0$$

$$\text{Trace}(u \otimes v) \rightarrow u \bullet v$$

$$(a \times b) \bullet (c \times d) \Rightarrow (a \bullet c)(b \bullet d) - (a \bullet d)(b \bullet c)$$

Compilation Issue

$$\mathrm{y} = \lambda F, n \left\langle F_{ij}(n) F_{1k}(n) - F_{0i}(n) F_{jk}(n) + F_{ij}(n) F_{k1}(n) \right\rangle_{ijk} (\mathrm{f,n})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \left\langle F_{ij}(n) F_{1k}(n) - F_{0i}(n) F_{jk}(n) + F_{ij}(n) F_{k1}(n) \right\rangle_{ijk} (\mathrm{f,n})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \langle F_{ij}(n) F_{1k}(n) - F_{0i}(n) F_{jk}(n) + F_{ij}(n) F_{k1}(n) \rangle_{ijk} (\mathrm{f}, \mathrm{n})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \langle F_{ij}(n) F_{1k}(n) - F_{0i}(n) F_{jk}(n) + F_{ij}(n) F_{k1}(n) \rangle_{ijk} (\mathrm{f,n})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \left\langle \underline{F_{ij}(n)} \underline{F_{1k}(n)} - \underline{F_{0i}(n)} \underline{F_{jk}(n)} + F_{ij}(n) \underline{F_{k1}(n)} \right\rangle_{ijk} (\mathrm{f,n})$$

Split

$$\mathrm{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$
$$\mathrm{v}_2 = \lambda F, n \left\langle F_{1i}(n) \right\rangle_{i} (\mathrm{f,n})$$
$$\mathrm{v}_3 = \lambda F, n \left\langle F_{0i}(n) \right\rangle_{i} (\mathrm{f,n})$$
$$\mathrm{v}_4 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$
$$\mathrm{v}_5 = \mathrm{v}_1$$
$$\mathrm{v}_6 = \lambda F, n \left\langle F_{i1}(n) \right\rangle_{i} (\mathrm{f,n})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \left\langle \underline{F_{ij}(n)} \underline{F_{1k}(n)} - \underline{F_{0i}(n)} \underline{F_{jk}(n)} + F_{ij}(n) \underline{F_{k1}(n)} \right\rangle_{ijk} (\mathrm{f,n})$$

Split

$$\mathrm{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$

$$\mathrm{v}_2 = \lambda F, n \left\langle F_{1i}(n) \right\rangle_{i} (\mathrm{f,n})$$

$$\mathrm{v}_3 = \lambda F, n \left\langle F_{0i}(n) \right\rangle_{i} (\mathrm{f,n})$$

$$\mathrm{v}_4 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$

$$\mathrm{v}_5 = \mathrm{v}_1$$

$$\mathrm{v}_6 = \lambda F, n \left\langle F_{i1}(n) \right\rangle_{i} (\mathrm{f,n})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \left\langle F_{ij}(n) F_{1k}(n) - F_{0i}(n) F_{jk}(n) + F_{ij}(n) F_{k1}(n) \right\rangle_{ijk} (\mathrm{f,n})$$

Split

$$\mathrm{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$

$$\mathrm{v}_2 = \lambda F, n \left\langle F_{1i}(n) \right\rangle_{i} (\mathrm{f,n})$$

$$\mathrm{v}_3 = \lambda F, n \left\langle F_{0i}(n) \right\rangle_{i} (\mathrm{f,n})$$

$$\mathrm{v}_4 = \mathrm{v}_1$$

$$\mathrm{v}_5 = \mathrm{v}_1$$

$$\mathrm{v}_6 = \lambda F, n \left\langle F_{i1}(n) \right\rangle_{i} (\mathrm{f,n})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \left\langle \underline{F_{ij}(n)} \underline{F_{1k}(n)} - \underline{F_{0i}(n)} F_{jk}(n) + F_{ij}(n) \underline{F_{k1}(n)} \right\rangle_{ijk} (\mathrm{f,n})$$

Split

$$\mathrm{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$
$$\mathrm{v}_2 = \lambda F, n \left\langle F_{1i}(n) \right\rangle_{i} (\mathrm{f,n})$$
$$\mathrm{v}_3 = \lambda F, n \left\langle F_{0i}(n) \right\rangle_{i} (\mathrm{f,n})$$
$$\mathrm{v}_4 = \mathrm{v}_1$$
$$\mathrm{v}_5 = \mathrm{v}_1$$
$$\mathrm{v}_6 = \lambda F, n \left\langle F_{i1}(n) \right\rangle_{i} (\mathrm{f,n})$$

Slice

$$\mathrm{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$
$$\mathrm{v}_2 \rightarrow \lambda T \left\langle T_{1i} \right\rangle_{i} (\mathrm{t})$$
$$\mathrm{v}_3 \rightarrow \lambda T \left\langle T_{0i} \right\rangle_{i} (\mathrm{t})$$
$$\mathrm{v}_4 = \mathrm{v}_1$$
$$\mathrm{v}_5 = \mathrm{v}_1$$
$$\mathrm{v}_6 \rightarrow \lambda T \left\langle T_{i1} \right\rangle_{i} (\mathrm{t})$$

Compilation Issue

Field Terms

$$\mathrm{y} = \lambda F, n \left\langle F_{ij}(n) F_{1k}(n) - F_{0i}(n) F_{jk}(n) + F_{ij}(n) F_{k1}(n) \right\rangle_{ijk} (\mathrm{f,n})$$

Split

$$\mathrm{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$
$$\mathrm{v}_2 = \lambda F, n \left\langle F_{1i}(n) \right\rangle_{i} (\mathrm{f,n})$$
$$\mathrm{v}_3 = \lambda F, n \left\langle F_{0i}(n) \right\rangle_{i} (\mathrm{f,n})$$
$$\mathrm{v}_4 = \mathrm{v}_1$$
$$\mathrm{v}_5 = \mathrm{v}_1$$
$$\mathrm{v}_6 = \lambda F, n \left\langle F_{i1}(n) \right\rangle_{i} (\mathrm{f,n})$$

Slice

$$\mathrm{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathrm{f,n})$$
$$\mathrm{v}_2 \rightarrow \lambda T \left\langle T_{1i} \right\rangle_{i} (\mathrm{t})$$
$$\mathrm{v}_3 \rightarrow \lambda T \left\langle T_{0i} \right\rangle_{i} (\mathrm{t})$$
$$\mathrm{v}_4 = \mathrm{v}_1$$
$$\mathrm{v}_5 = \mathrm{v}_1$$
$$\mathrm{v}_6 \rightarrow \lambda T \left\langle T_{i1} \right\rangle_{i} (\mathrm{t})$$

Compilation Issue

Field Terms

$$\mathbf{y} = \lambda F, n \left\langle F_{ij}(n) F_{1k}(n) - F_{0i}(n) F_{jk}(n) + F_{ij}(n) F_{k1}(n) \right\rangle_{ijk} (\mathbf{f},\mathbf{n})$$

Split

$$\mathbf{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathbf{f},\mathbf{n})$$
$$\mathbf{v}_2 = \lambda F, n \left\langle F_{1i}(n) \right\rangle_{i} (\mathbf{f},\mathbf{n})$$
$$\mathbf{v}_3 = \lambda F, n \left\langle F_{0i}(n) \right\rangle_{i} (\mathbf{f},\mathbf{n})$$
$$\mathbf{v}_4 = \mathbf{v}_1$$
$$\mathbf{v}_5 = \mathbf{v}_1$$
$$\mathbf{v}_6 = \lambda F, n \left\langle F_{i1}(n) \right\rangle_{i} (\mathbf{f},\mathbf{n})$$

Slice

$$\mathbf{v}_1 = \lambda F, n \left\langle F_{ij}(n) \right\rangle_{ij} (\mathbf{f},\mathbf{n})$$
$$\mathbf{v}_2 \rightarrow \lambda T \left\langle T_{1i} \right\rangle_{i} (\mathbf{t})$$
$$\mathbf{v}_3 \rightarrow \lambda T \left\langle T_{0i} \right\rangle_{i} (\mathbf{t})$$
$$\mathbf{v}_4 = \mathbf{v}_1$$
$$\mathbf{v}_5 = \mathbf{v}_1$$
$$\mathbf{v}_6 \rightarrow \lambda T \left\langle T_{i1} \right\rangle_{i} (\mathbf{t})$$

$$\mathbf{y} = \lambda T, B, C, D \left\langle T_{ij} B_k - C_i T_{jk} + T_{ij} D_k \right\rangle_{ijk} (\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3,\mathbf{v}_6)$$