Diderot: A DSL for image analysis

Diderot is a domain specific language for image analysis and scientific visualization

[CHIW PLDI]

Diderot: A Parallel DSL for Image Analysis and Visualization

Charisee Chiw Gordon Kindlmann John Reppy Lamont Samuels Nick Seltzer**
University of Chicago
{cchiw,glk,jhr,larmore,nseltzer}@cs.uchicago.edu

Abstract

Research scientists and medical professionals use imaging technology, such as *computed tomography* (CT) and *magnetic resonance imaging* (MRI) to measure a wide variety of biological and physical objects. The increasing sophistication of imaging technology creates demand for equally sophisticated computational techniques to analyze and visualize the image data. Analysis and visualization codes are often crafted for a specific experiment or set of images, ... cal value of imaging, the quantities and types of data from modern imaging hardware also strain the capabilities of established tools for image processing. Scientists working with the new imaging techniques and applications are the best positioned to create and evaluate methods of image data processing, but they do not have the programming background to create the efficient parallel implementations required to handle the quantity of data involved.

In this paper, we present a domain-specific language (DSL), ...

Diderot: A DSL for image analysis

Diderot is a domain specific language for image analysis and scientific visualization

- Provide a notation can express statements in the domain space

[CHIW PLDI]

Diderot: A Parallel DSL for Image Analysis and Visualization

Charisee Chiw Gordon Kindlmann John Reppy Lamont Samuels Nick Seltzer**
University of Chicago
{cchiw,glk,jhr,larmore,nseltzer}@cs.uchicago.edu

Abstract

Research scientists and medical professionals use imaging technology, such as computed tomography (CT) and magnetic resonance imaging (MRI) to measure a wide variety of biological and physical objects. The increasing sophistication of imaging technology creates demand for equally sophisticated computational techniques to analyze and visualize the image data. Analysis and visualization codes are often crafted for a specific experiment or set of images, ed value of imaging, the quantities and types of data from modern imaging hardware also strain the capabilities of established tools for image processing. Scientists working with the new imaging techniques and applications are the best positioned to create and evaluate methods of image data processing, but they do not have the programming background to create the efficient parallel implementations required to handle the quantity of data involved.

In this paper, we present a domain-specific language (DSL),

Diderot: A DSL for image analysis

Diderot is a domain specific language for image analysis and scientific visualization

- Provide a notation can express statements in the domain space
- Efficient execution of expensive computations

[CHIW PLDI]

Diderot: A Parallel DSL for Image Analysis and Visualization

Charisee Chiw Gordon Kindlmann John Reppy Lamont Samuels Nick Seltzer**
University of Chicago
{cchiw,glk,jhr,larmore,nseltzer}@cs.uchicago.edu

Abstract

Research scientists and medical professionals use imaging technology, such as computed tomography (CT) and magnetic resonance imaging (MRI) to measure a wide variety of biological and physical objects. The increasing sophistication of imaging technology creates demand for equally sophisticated computational techniques to analyze and visualize the image data. Analysis and visualization codes are often crafted for a specific experiment or set of images, ed value of imaging, the quantities and types of data from modern imaging hardware also strain the capabilities of established tools for image processing. Scientists working with the new imaging techniques and applications are the best positioned to create and evaluate methods of image data processing, but they do not have the programming background to create the efficient parallel implementations required to handle the quantity of data involved.

In this paper, we present a domain-specific language (DSL),

Diderot: A DSL for image analysis

Diderot is a domain specific language for image analysis and scientific visualization

- Provide a notation can express statements in the domain space
- Efficient execution of expensive computations
- Enable cutting-edge research

[CHIW PLDI]

Diderot: A Parallel DSL for Image Analysis and Visualization

Charisee Chiw Gordon Kindlmann John Reppy Lamont Samuels Nick Seltzer**
University of Chicago
{cchiw,glk,jhr,larmore,nseltzer}@cs.uchicago.edu

Abstract

Research scientists and medical professionals use imaging technology, such as computed tomography (CT) and magnetic resonance imaging (MRI) to measure a wide variety of biological and physical objects. The increasing sophistication of imaging technology creates demand for equally sophisticated computational techniques to analyze and visualize the image data. Analysis and visualization codes are often crafted for a specific experiment or set of images, ... cal value of imaging, the quantities and types of data from modern imaging hardware also strain the capabilities of established tools for image processing. Scientists working with the new imaging techniques and applications are the best positioned to create and evaluate methods of image data processing, but they do not have the programming background to create the efficient parallel implementations required to handle the quantity of data involved.

In this paper, we present a domain-specific language (DSL), ...

Diderot: A DSL for image analysis

Diderot: A DSL for image analysis

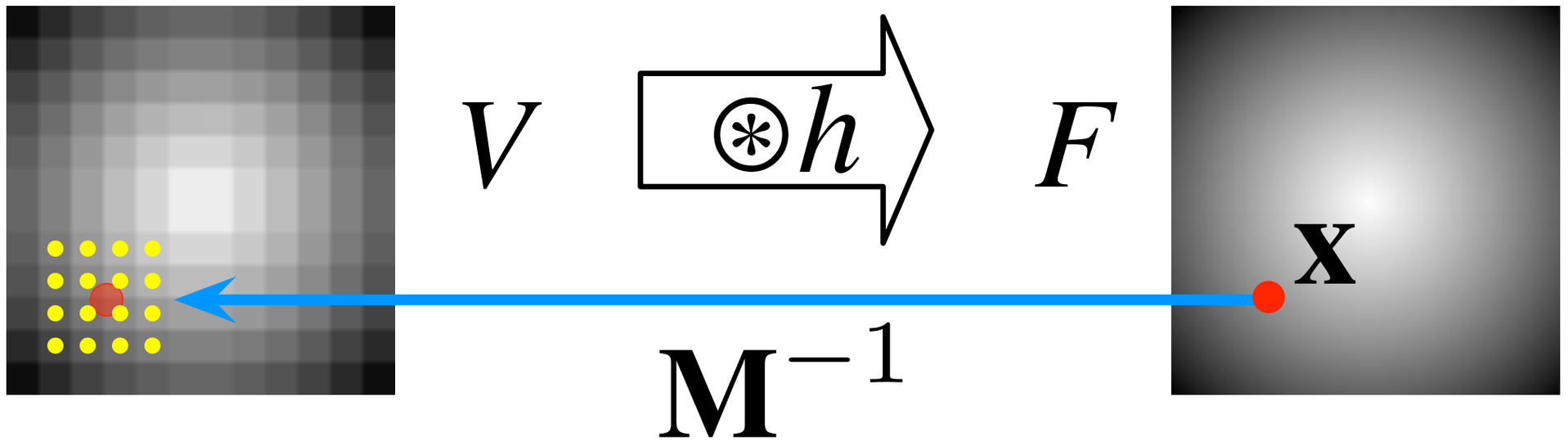
- Ease transition of ideas from whiteboard to code

Diderot: A DSL for image analysis

- Ease transition of ideas from whiteboard to code
- Tensor based IR (EIN) enables vis algorithms

Diderot: A DSL for image analysis

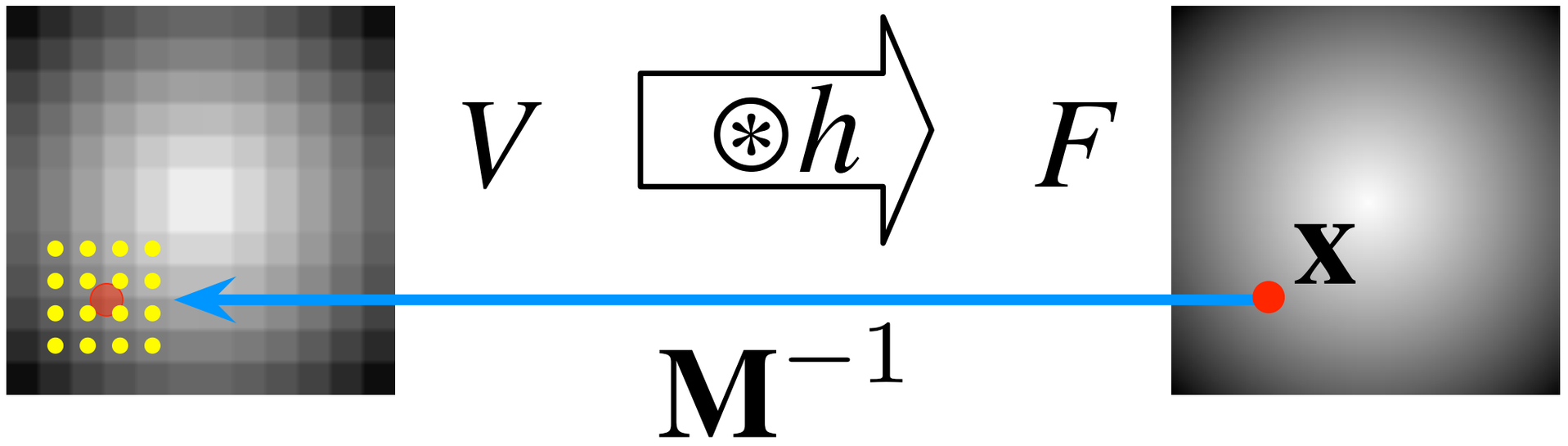
- Ease transition of ideas from whiteboard to code
- Tensor based IR (EIN) enables vis algorithms
- Turn discrete data into a continuous field



Discrete image data *Continuous field*

Diderot: A DSL for image analysis

- Ease transition of ideas from whiteboard to code
- Tensor based IR (EIN) enables vis algorithms
- Turn discrete data into a continuous field



Discrete image data *Continuous field*

$$F(x) = \sum_{i=1-s}^{s} \sum_{j=1-s}^{s} V[n+ < i, j >] h(f_x - i) h(f_y - j)$$

Roadmap

Roadmap

1. Developing of the Diderot Language

Roadmap

1. Developing of the Diderot Language
2. Application of the language to a new domain: finite element data

Roadmap

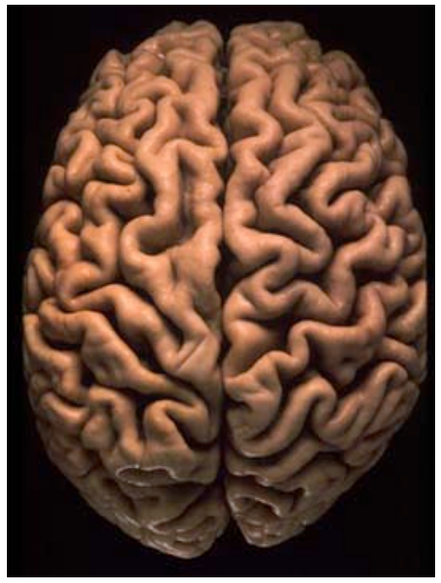
0. Image analysis and scientific visualization
1. Developing of the Diderot Language
2. Application of the language to a new domain: finite element data

Why image analysis is important

- Scientists need software tools to extract structure from many kinds of image data.
- Creating new visualization programs is part of the experimental process.
- Visualization helps explore data.

Why image analysis is important

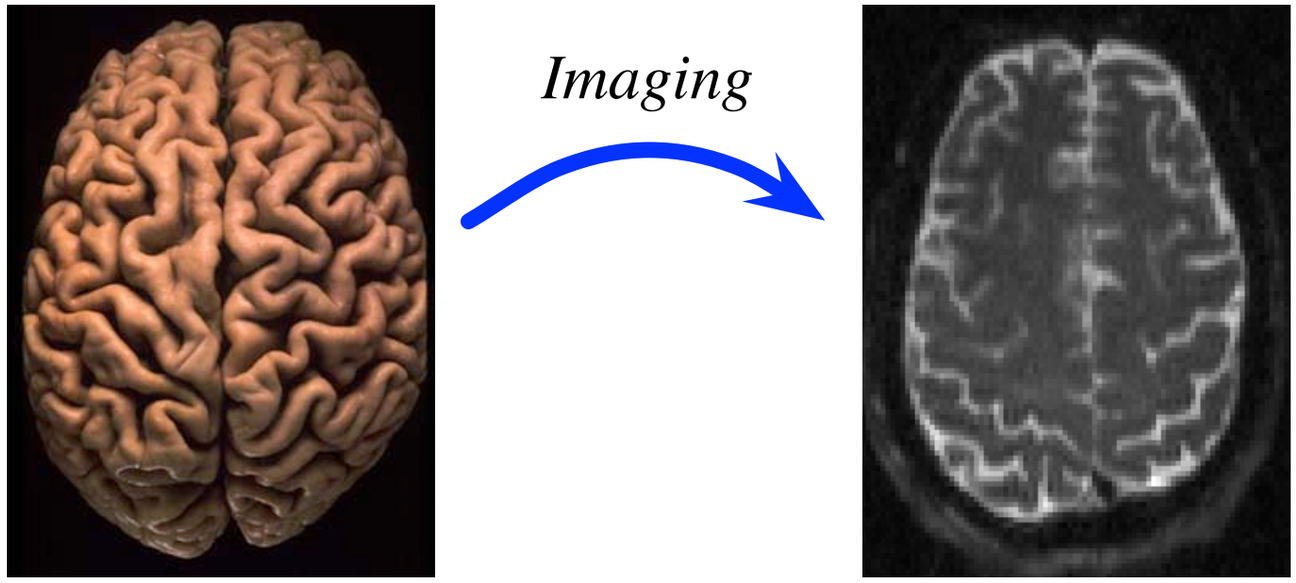
- Scientists need software tools to extract structure from many kinds of image data.
- Creating new visualization programs is part of the experimental process.
- Visualization helps explore data.



Physical object

Why image analysis is important

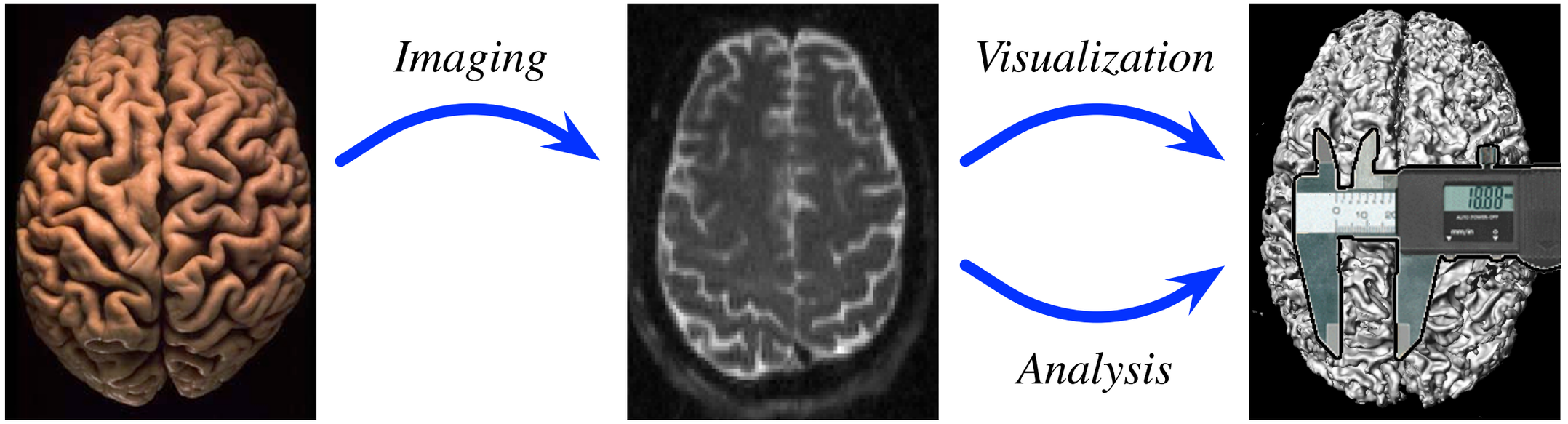
- Scientists need software tools to extract structure from many kinds of image data.
- Creating new visualization programs is part of the experimental process.
- Visualization helps explore data.



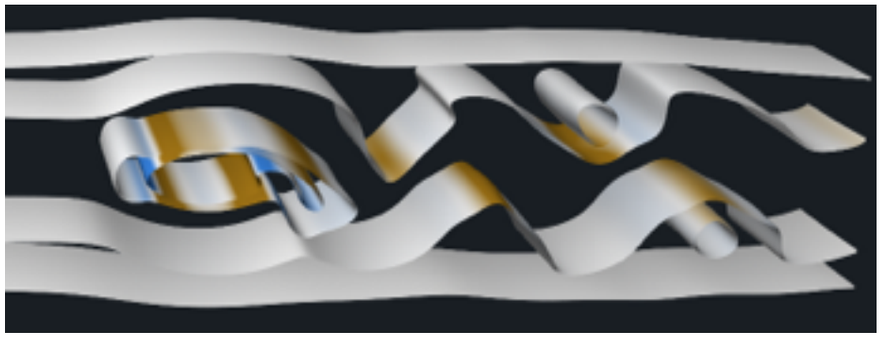
Physical object **Image data**

Why image analysis is important

- Scientists need software tools to extract structure from many kinds of image data.
- Creating new visualization programs is part of the experimental process.
- Visualization helps explore data.

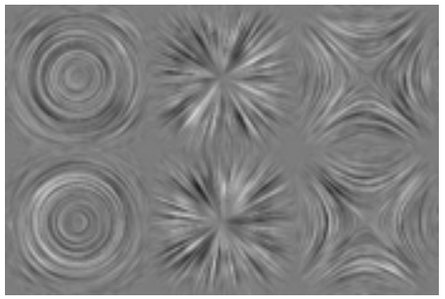


Visualization Applications

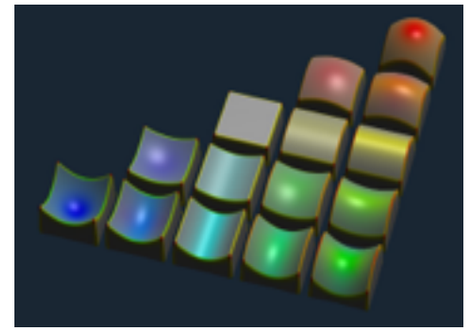


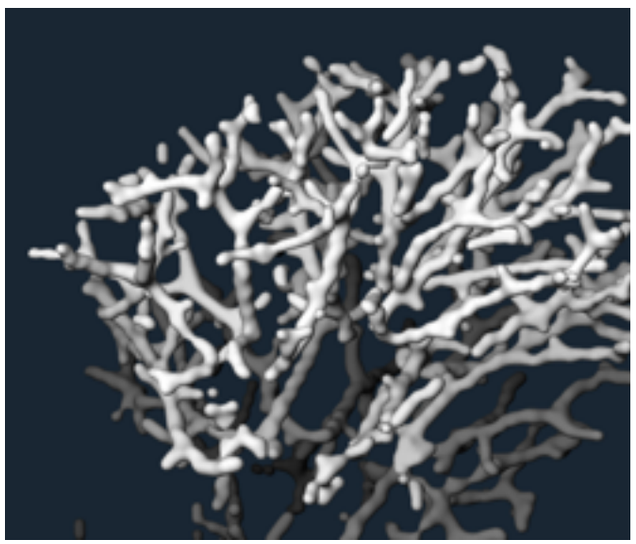
Fluid flow

Diderot ^ The guy

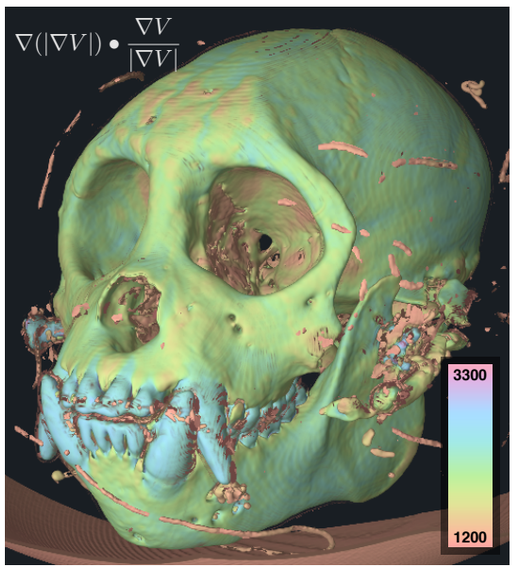


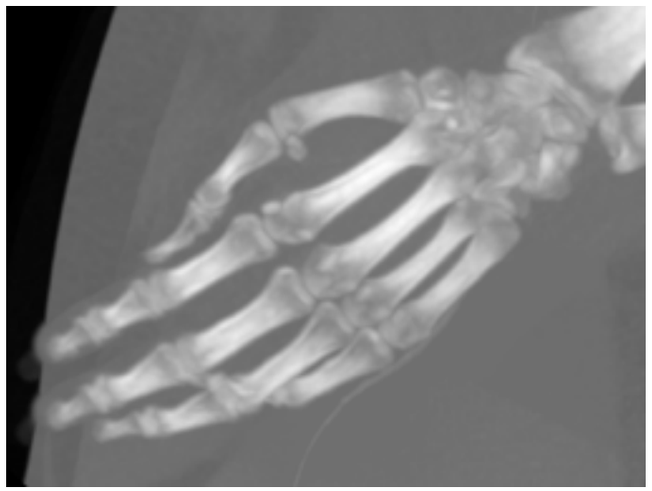
Synthetic data

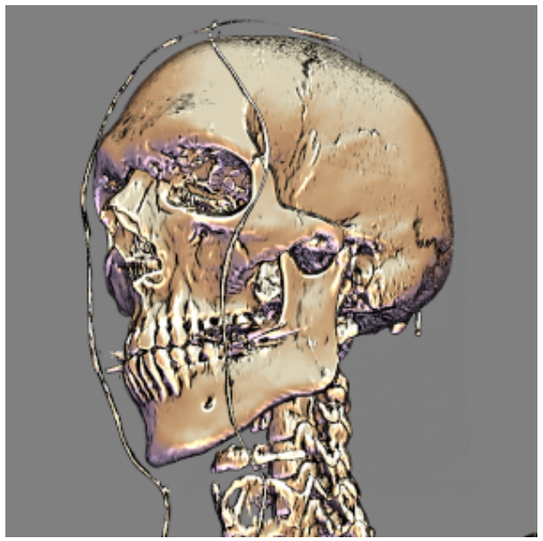




Lung data





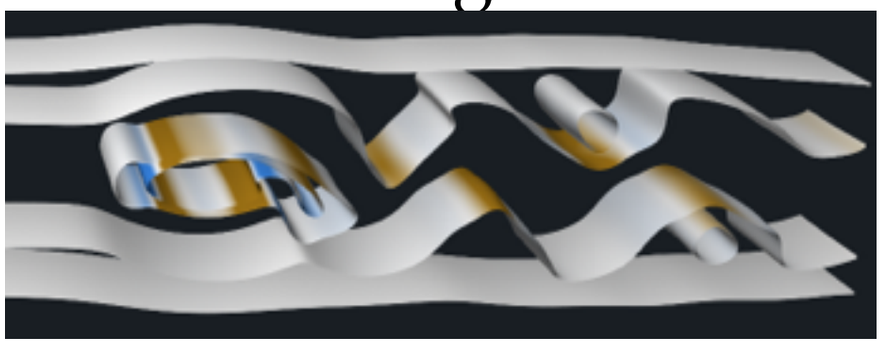


X-Ray data and CT Scans

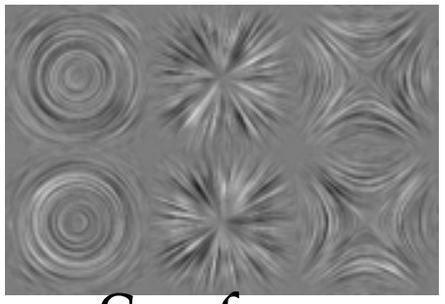
Visualization Applications

Visually explore data and compute features

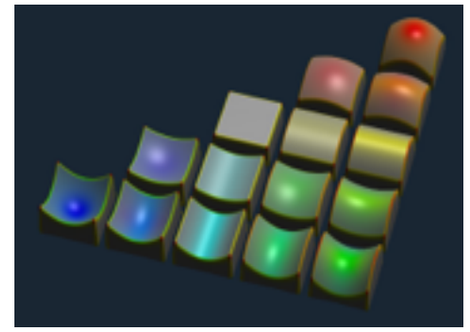
Flow Magnitude



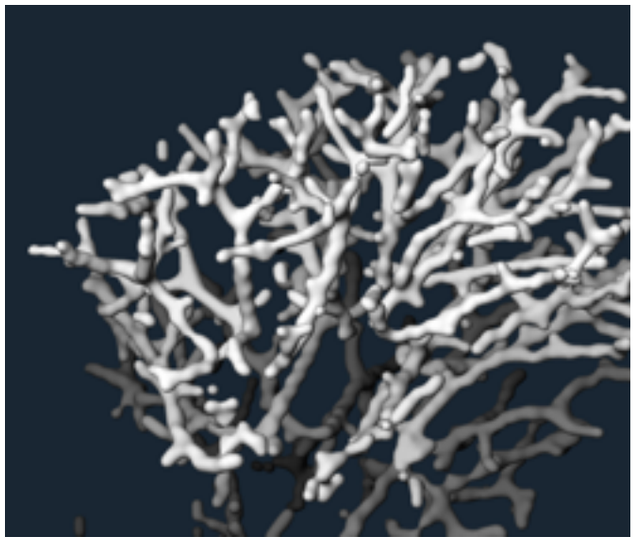
LIC



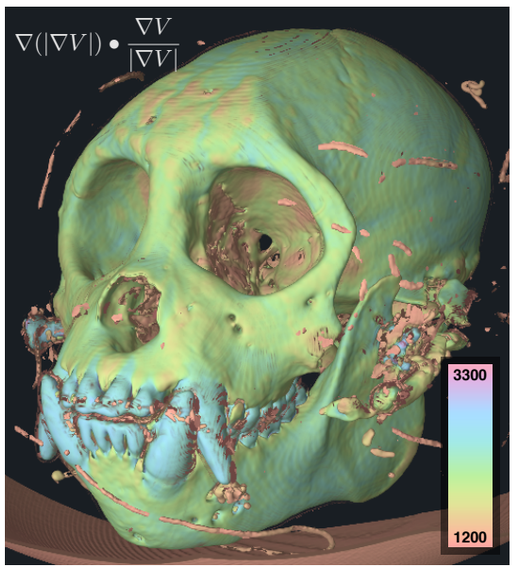
Surface Curvature



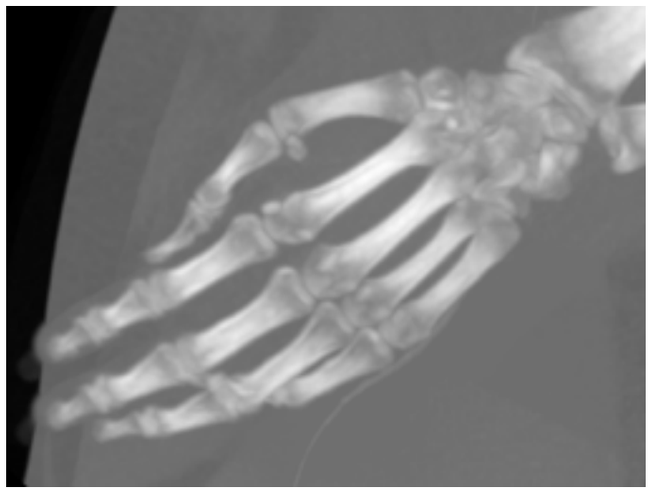
Canny Edges



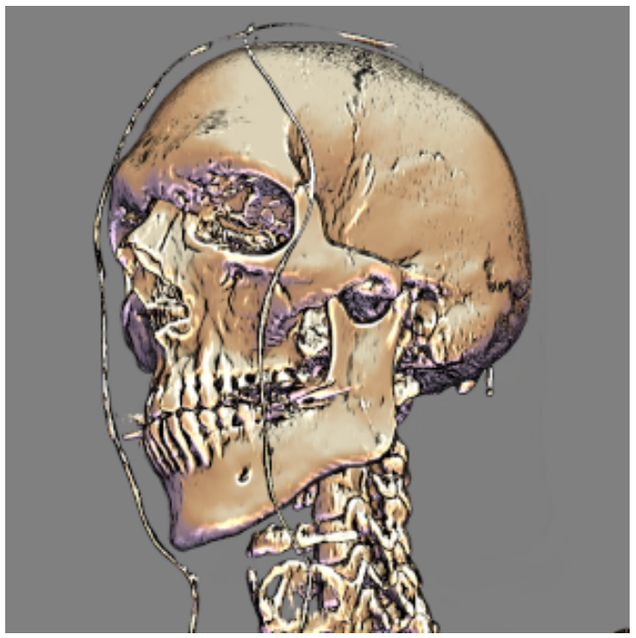
Ridge Detection



Isocontour Surface MIP

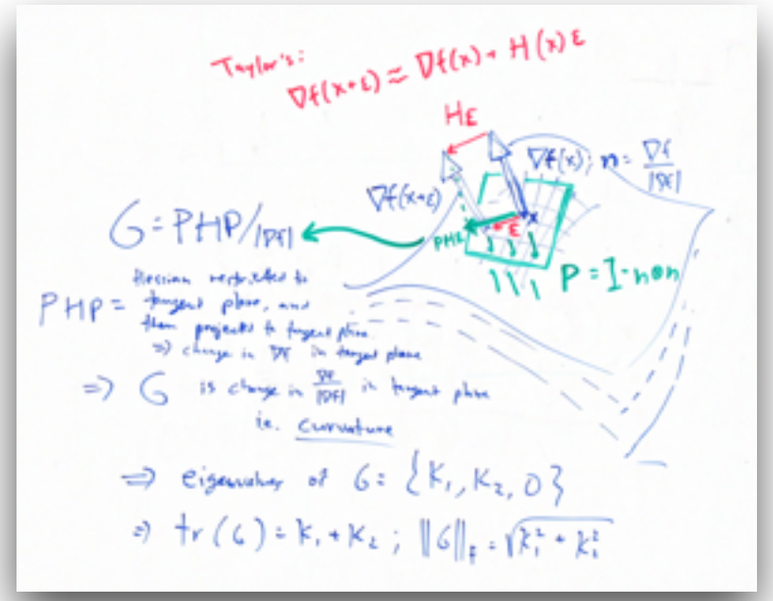


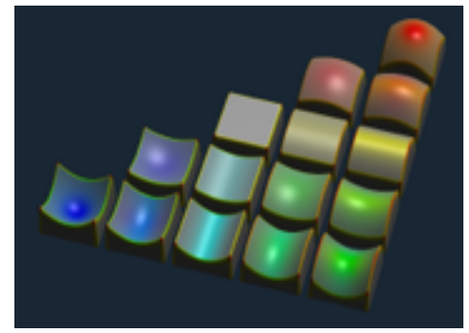
Volume rendering



Challenges

- What do I want to implement?



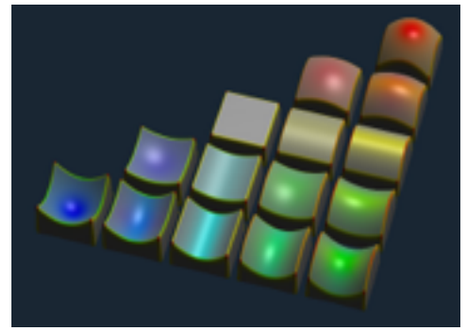


Visualization concept **Implementation** **Image**

Challenges

- What do I want to implement?

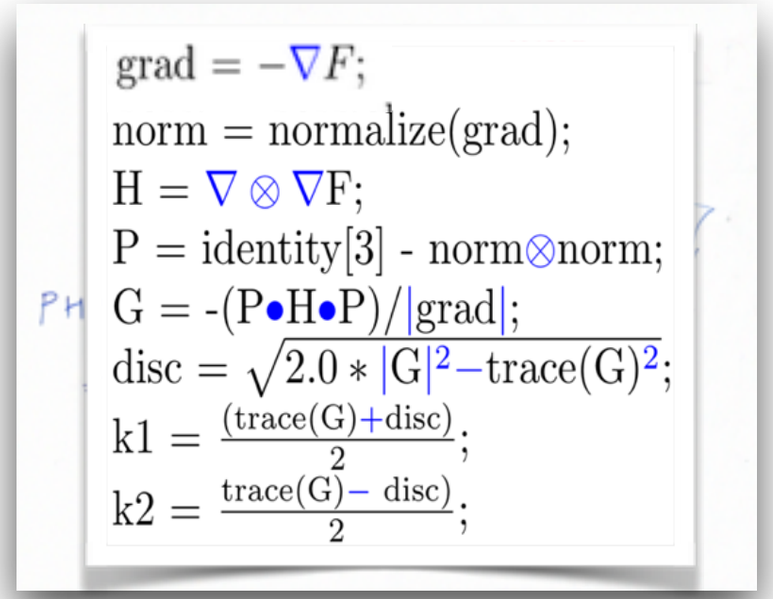
$$
\begin{aligned}
&\text{grad} = -\nabla F;\\
&\text{norm} = \text{normalize}(\text{grad});\\
&\text{H} = \nabla \otimes \nabla \text{F};\\
&\text{P} = \text{identity}[3] - \text{norm} \otimes \text{norm};\\
&\text{G} = -(\text{P} \bullet \text{H} \bullet \text{P})/|\text{grad}|;\\
&\text{disc} = \sqrt{2.0 * |\text{G}|^2 - \text{trace}(\text{G})^2};\\
&\text{k1} = \frac{(\text{trace}(\text{G}) + \text{disc})}{2};\\
&\text{k2} = \frac{\text{trace}(\text{G}) - \text{disc})}{2};
\end{aligned}
$$

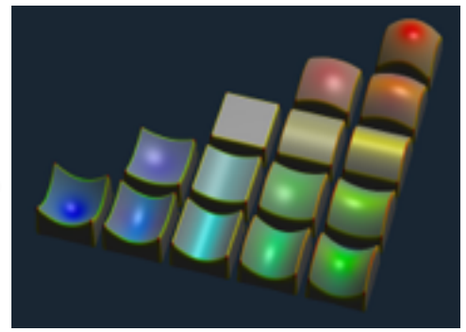


Visualization concept **Implementation** **Image**

Challenges

- What do I want to implement?
- How do I implement this?

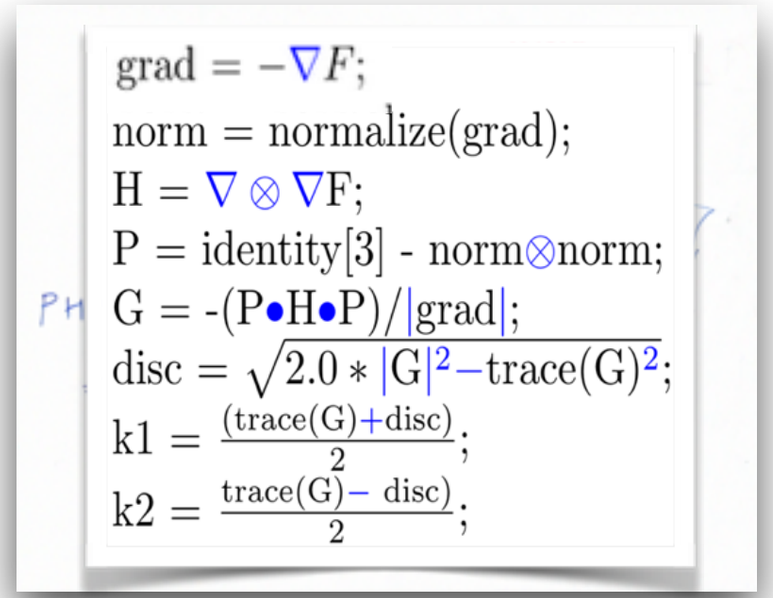


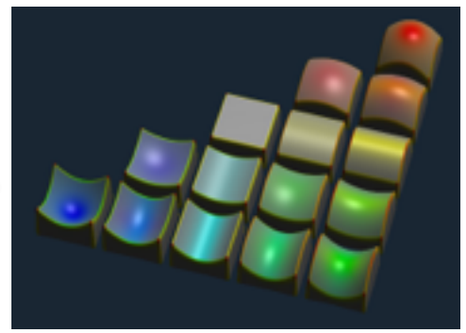


Visualization concept **Implementation** **Image**

Challenges

- What do I want to implement?
- How do I implement this?
- How do I implement it efficiently?



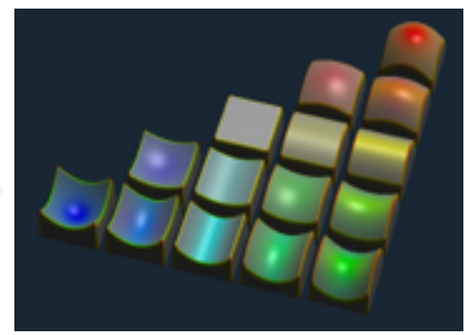


Visualization concept **Implementation** **Image**

Programmability: From whiteboard to code

Surface Curvature

$$\text{grad} = -\nabla F;$$
$$\text{norm} = \text{normalize}(\text{grad});$$
$$\text{H} = \nabla \otimes \nabla \text{F};$$
$$\text{P} = \text{identity}[3] - \text{norm} \otimes \text{norm};$$
$$\text{G} = -(\text{P} \bullet \text{H} \bullet \text{P})/|\text{grad}|;$$
$$\text{disc} = \sqrt{2.0 * |\text{G}|^2 - \text{trace}(\text{G})^2};$$
$$\text{k1} = \frac{(\text{trace}(\text{G}) + \text{disc})}{2};$$
$$\text{k2} = \frac{\text{trace}(\text{G}) - \text{disc})}{2};$$



Visualization concept

Diderot code

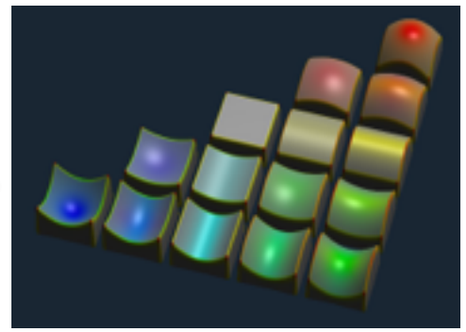
Image

Programmability: From whiteboard to code

Surface Curvature

$$\text{grad} = -\nabla F;$$
$$\text{norm} = \text{normalize}(\text{grad});$$
$$\text{H} = \nabla \otimes \nabla \text{F};$$
$$\text{P} = \text{identity}[3] - \text{norm} \otimes \text{norm};$$
$$\text{G} = -(\text{P} \bullet \text{H} \bullet \text{P})/|\text{grad}|;$$
$$\text{disc} = \sqrt{2.0 * |\text{G}|^2 - \text{trace}(\text{G})^2};$$
$$\text{k1} = \frac{(\text{trace}(\text{G}) + \text{disc})}{2};$$
$$\text{k2} = \frac{(\text{trace}(\text{G}) - \text{disc})}{2};$$

```
vec3 grad = -∇F (pos);
vec3 norm = normalize(grad);
tensor[3,3] H = ∇ ⊗ ∇ F (pos);
tensor[3,3] P = identity[3] - norm⊗norm;
tensor[3,3] G = -(P•H•P)/|grad|;
real disc = sqrt(2.0 * |G|^2 - trace(G)^2);
real k1 = (trace(G) + disc)/2.0;
real k2 = (trace(G) - disc)/2.0;
```

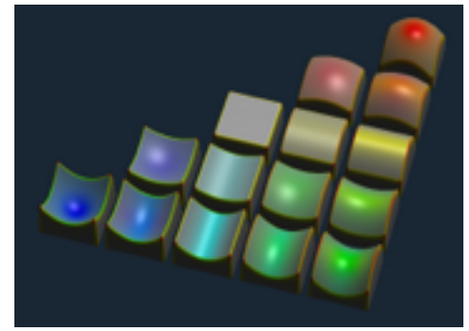


Visualization concept | **Diderot code** | **Image**

Diderot program

```
field#4(3)[] F = bspln5 ⊛ image("quad-patches.nrrd");
field#0(2)[3] RGB = tent ⊛ image("2d-bow.nrrd");
...
strand RayCast (int ui, int vi) {
  ...
  update
   ...
  vec3 grad = -∇F (pos);
  vec3 norm = normalize(grad);
  tensor[3,3] H = ∇⊗∇ F (pos);
  tensor[3,3] P = identity[3] - norm⊗norm;
  tensor[3,3] G = -(P•H•P)/|grad|;
  real disc = sqrt(2.0 * |G|^2 - trace(G)^2);
  real k1 = (trace(G) + disc)/2.0;
  real k2 = (trace(G) - disc)/2.0;
  vec3 matRGB = ...RGB([...k1, ...k2]);
  real out = ...
  }
...
}
```

Surface Curvature



Diderot program

```
field#4(3)[] F = bspln5 ⊛ image("quad-patches.nrrd");
field#0(2)[3] RGB = tent ⊛ image("2d-bow.nrrd");
...
strand RayCast (int ui, int vi) {
  ...
  update
   ...
  vec3 grad = -∇F (pos);
  vec3 norm = normalize(grad);
  tensor[3,3] H = ∇ ⊗ ∇ F (pos);
  tensor[3,3] P = identity[3] - norm⊗norm;
  tensor[3,3] G = -(P•H•P)/|grad|;
  real disc = sqrt(2.0 * |G|^2 - trace(G)^2);
  real k1 = (trace(G) + disc)/2.0;
  real k2 = (trace(G) - disc)/2.0;
  vec3 matRGB = ...RGB([...k1, ...k2]);
  real out = ...
  }
...
}
```

Tensor: reals, vectors, matrices,…

Tensor operators: additional, subtraction, multiplication

Diderot program

```
field#4(3)[] F = bspln5 ⊛ image("quad-patches.nrrd");
field#0(2)[3] RGB = tent ⊛ image("2d-bow.nrrd");
...
strand RayCast (int ui, int vi) {
  ...
  update
   ...
  vec3 grad = -∇F (pos);
  vec3 norm = normalize(grad);
  tensor[3,3] H = ∇ ⊗ ∇ F (pos);
  tensor[3,3] P = identity[3] - norm⊗norm;
  tensor[3,3] G = -(P•H•P)/|grad|;
  real disc = sqrt(2.0 * |G|^2 - trace(G)^2);
  real k1 = (trace(G) + disc)/2.0;
  real k2 = (trace(G) - disc)/2.0;
  vec3 matRGB = ...RGB([...k1, ...k2]);
  real out = ...
  }
...
}
```

Tensor: reals, vectors, matrices,…

Tensor operators: additional, subtraction, multiplication

Diderot program

```
field#4(3)[] F = bspln5 ⊛ image("quad-patches.nrrd");
field#0(2)[3] RGB = tent ⊛ image("2d-bow.nrrd");
...
strand RayCast (int ui, int vi) {
  ...
  update
   ...
  vec3 grad = -∇F (pos);
  vec3 norm = normalize(grad);
  tensor[3,3] H = ∇ ⊗ ∇ F (pos);
  tensor[3,3] P = identity[3] - norm⊗norm;
  tensor[3,3] G = -(P•H•P)/|grad|;
  real disc = sqrt(2.0 ∗ |G|² - trace(G)²);
  real k1 = (trace(G) + disc)/2.0;
  real k2 = (trace(G) - disc)/2.0;
  vec3 matRGB = ...RGB([...k1, ...k2]);
  real out = ...
  }
...
}
```

Field: function from d-dimensional space to tensors

Probe: fields are probed at a position to produce a tensor

Derivative: rate of change

Diderot program

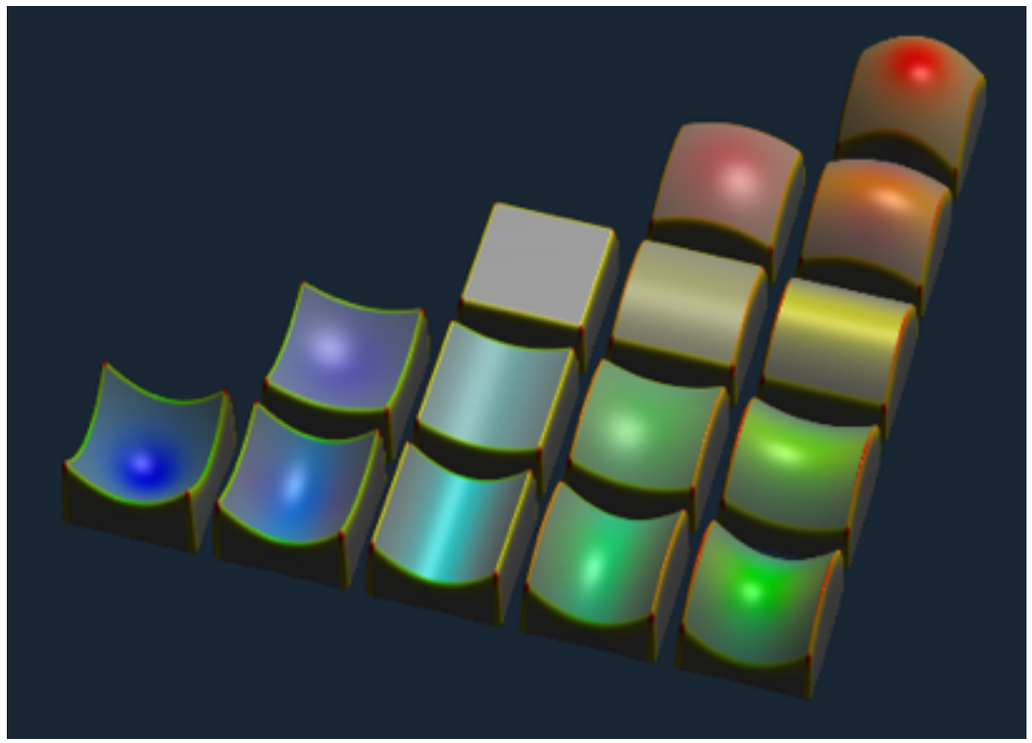
```
field#4(3)[] F = bspln5 ⊛ image("quad-patches.nrrd");
field#0(2)[3] RGB = tent ⊛ image("2d-bow.nrrd");
...
strand RayCast (int ui, int vi) {
  ...
  update
   ...
  vec3 grad = -∇F (pos);
  vec3 norm = normalize(grad);
  tensor[3,3] H = ∇⊗∇F (pos);
  tensor[3,3] P = identity[3] - norm⊗norm;
  tensor[3,3] G = -(P•H•P)/|grad|;
  real disc = sqrt(2.0 * |G|^2 - trace(G)^2);
  real k1 = (trace(G) + disc)/2.0;
  real k2 = (trace(G) - disc)/2.0;
  vec3 matRGB = ...RGB([...k1, ...k2]);
  real out = ...
  }
...
}
```

Field: function from d-dimensional space to tensors

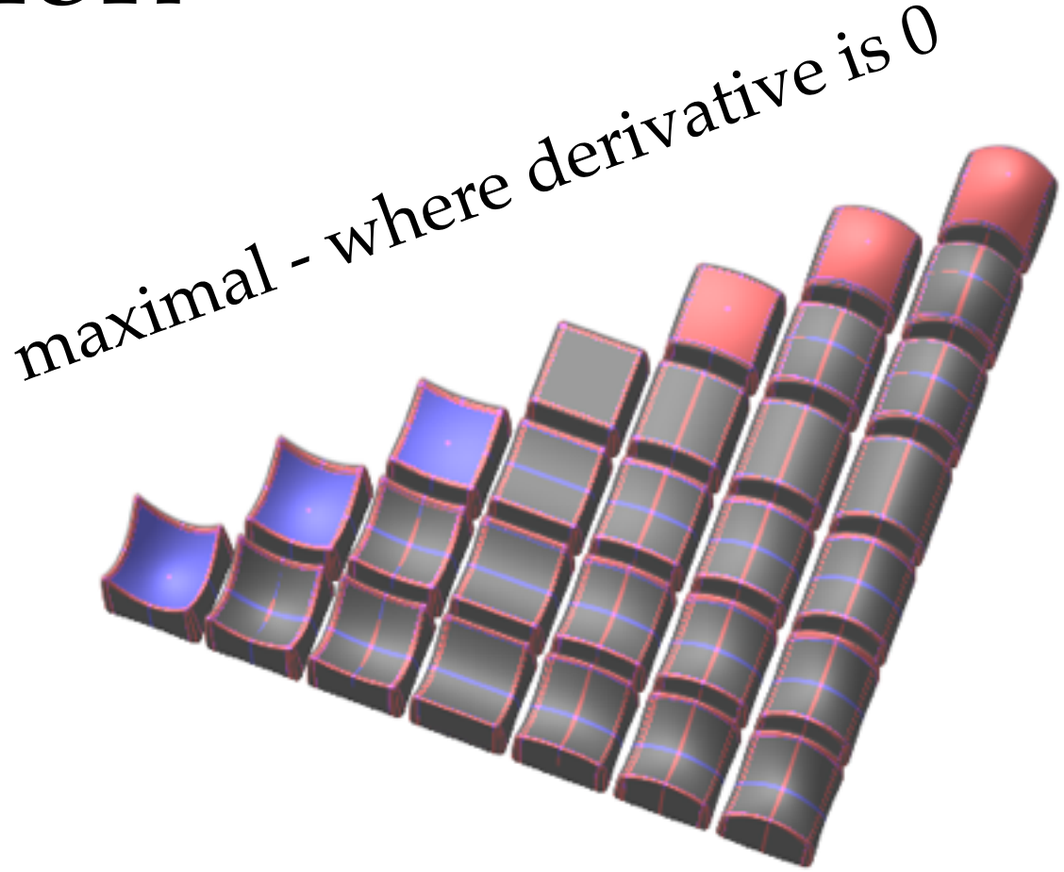
Probe: fields are probed at a position to produce a tensor

Derivative: rate of change

Motivation



Curvature



Crest Lines are places where the surface curvature is maximal along the **curvature** direction.

Motivation

```
field#4(3)[] F = bspln5 ⊛ image("quad-patches.nrrd");
field#0(2)[3] RGB = tent ⊛ image("2d-bow.nrrd");
...
strand RayCast (int ui, int vi) {
  ...
  update
   ...
  vec3 grad = -∇F (pos);
  vec3 norm = normalize(grad);
  tensor[3,3] H = ∇ ⊗ ∇ F (pos);
  tensor[3,3] P = identity[3] - norm⊗norm;
  tensor[3,3] G = -(P•H•P)/|grad|;
  real disc = sqrt(2.0 * |G|^2 - trace(G)^2);
  real k1 = (trace(G) + disc)/2.0;
  real k2 = (trace(G) - disc)/2.0;
  vec3 matRGB = ...RGB([...k1, ...k2]);
  real out = ...∇ ⊗ ∇k1...
  }
...
}
```

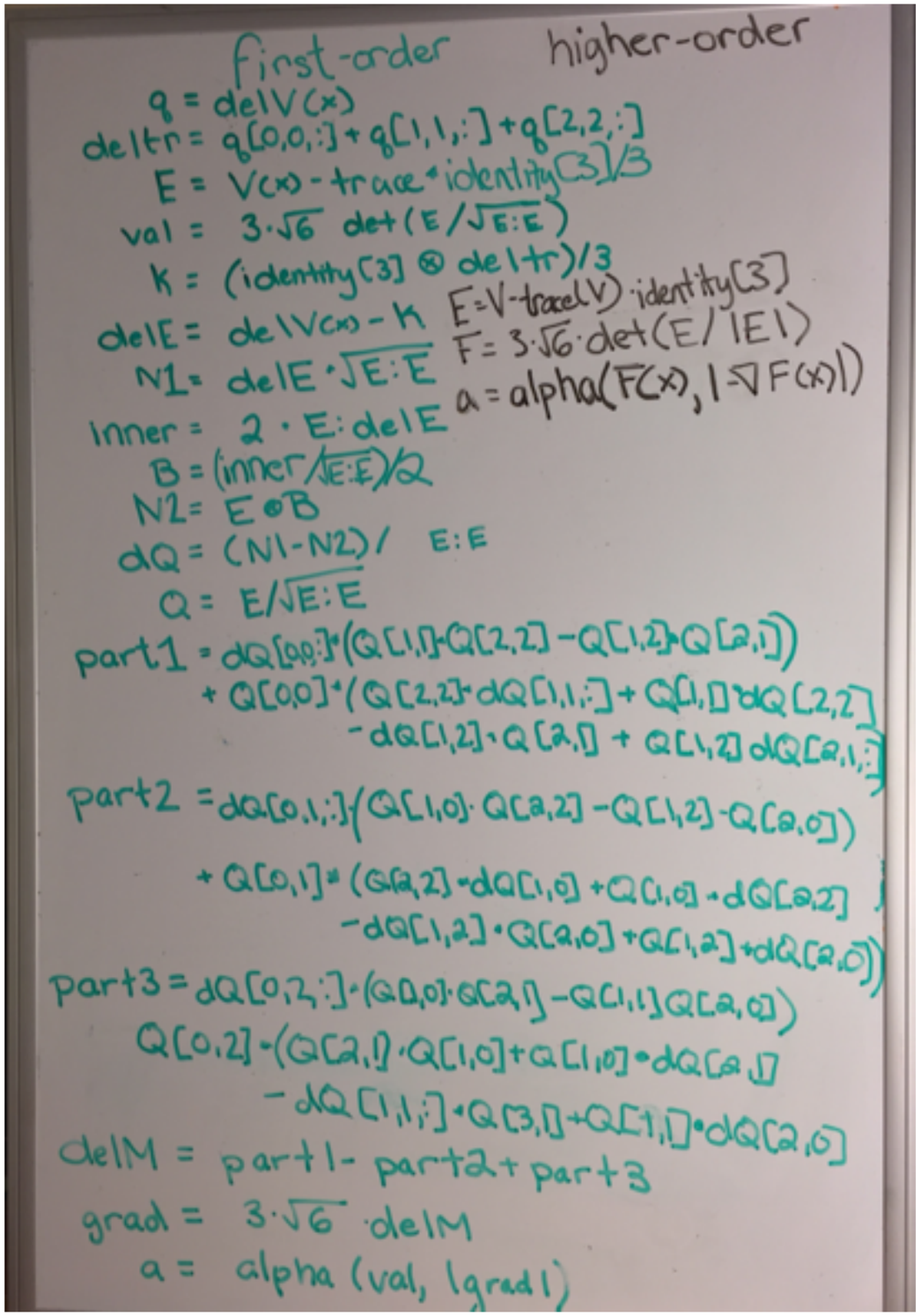
Directly add line

Motivation

```
field#4(3)[] F = bspln5 ⊛ image("quad-patches.nrrd");
field#0(2)[3] RGB = tent ⊛ image("2d-bow.nrrd");
...
strand RayCast (int ui, int vi) {
  ...
  update
   ...
  vec3 grad = -∇F (pos);
  vec3 norm = normalize(grad);
  tensor[3,3] H = ∇ ⊗ ∇ F (pos);
  tensor[3,3] P = identity[3] - norm⊗norm;
  tensor[3,3] G = -(P•H•P)/|grad|;
  real disc = sqrt(2.0 * |G|^2 - trace(G)^2);
  real k1 = (trace(G) + disc)/2.0;
  real k2 = (trace(G) - disc)/2.0;
  vec3 matRGB = ...RGB([...k1, ...k2]);
  real out = ...∇ ⊗ ∇k1..
  }
...
}
```

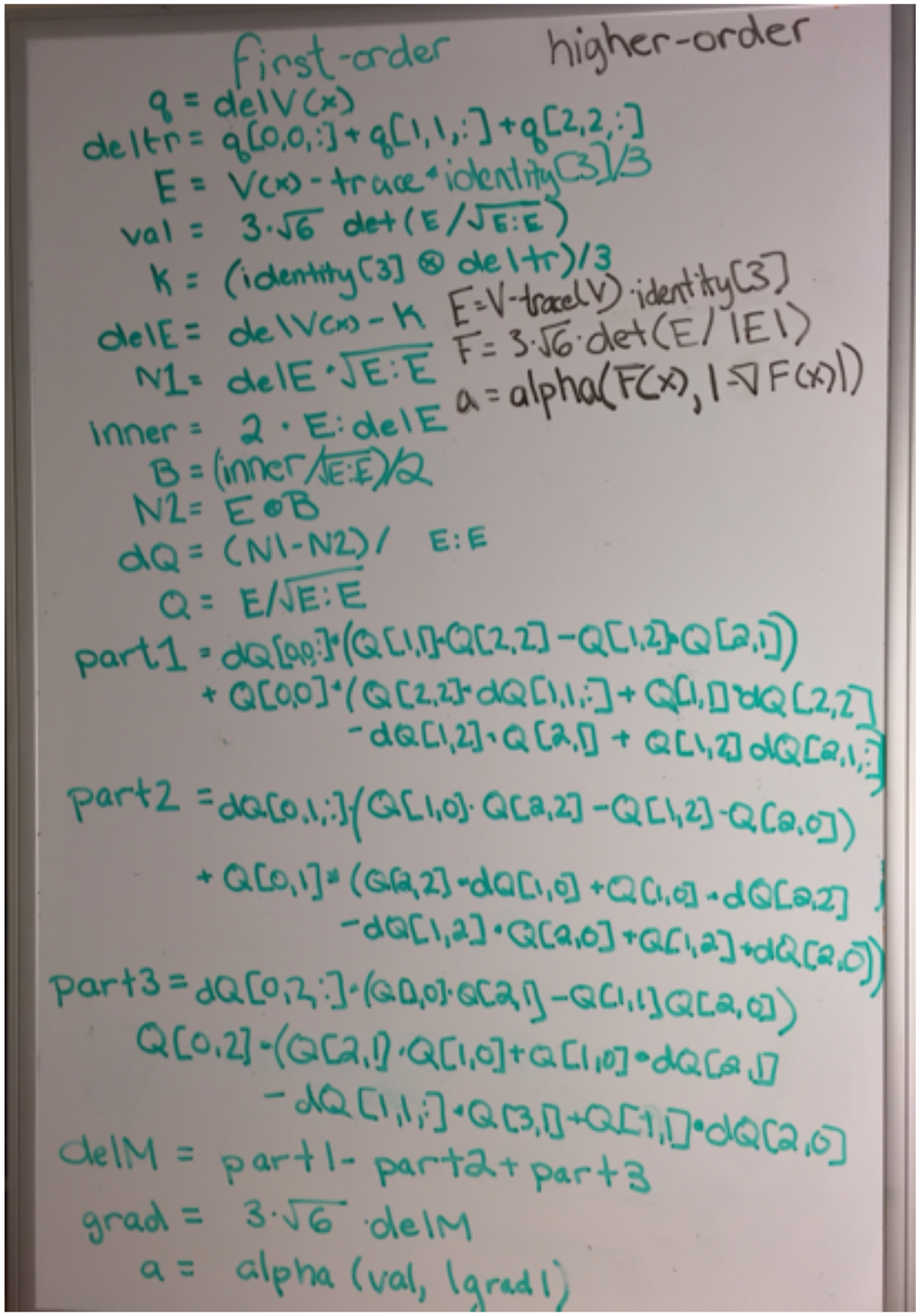
Not permitted

Motivation


first-order
higher-order
q = delV(x)
deltr = q[0,0,:] + q[1,1,:] + q[2,2,:]
E = V(x) - trace * identity[3]/3
val = 3·√6 det(E/√E:E)
K = (identity[3] ⊗ deltr)/3
delE = delV(x) - K
N1 = delE · √E:E
inner = 2 · E:delE
B = (inner/√E:E)/2
N2 = E ∘ B
dQ = (N1-N2)/ E:E
Q = E/√E:E
F=V-trace(V)·identity[3]
F = 3·√6·det(E/|E|)
a = alpha(F(x), 1-∇F(x))
part1 = dQ[0,0,:]·(Q[1,1]·Q[2,2] - Q[1,2]·Q[2,1])
+ Q[0,0]·(Q[2,2]·dQ[1,1,:] + Q[1,1]·dQ[2,2]
- dQ[1,2]·Q[2,1] + Q[1,2]·dQ[2,1,:])
part2 = dQ[0,1,:]·(Q[1,0]·Q[2,2] - Q[1,2]·Q[2,0])
+ Q[0,1]·(Q[2,2]·dQ[1,0] + Q[1,0]·dQ[2,2]
- dQ[1,2]·Q[2,0] + Q[1,2]·dQ[2,0])
part3 = dQ[0,2,:]·(Q[1,0]·Q[2,1] - Q[1,1]·Q[2,0])
Q[0,2]·(Q[2,1]·Q[1,0] + Q[1,0]·dQ[2,1]
- dQ[1,1,:]·Q[3,1] + Q[1,1]·dQ[2,0]
delM = part1 - part2 + part3
grad = 3·√6 ·delM
a = alpha (val, |grad|)

Motivation

The compiler, not the programmer, should do the work.



Implementing Expressivity

- Motivated our need for higher-level notation
- Compilation of tensor calculus to C
- Our intermediate representation, EIN

EIN Intermediate Representation

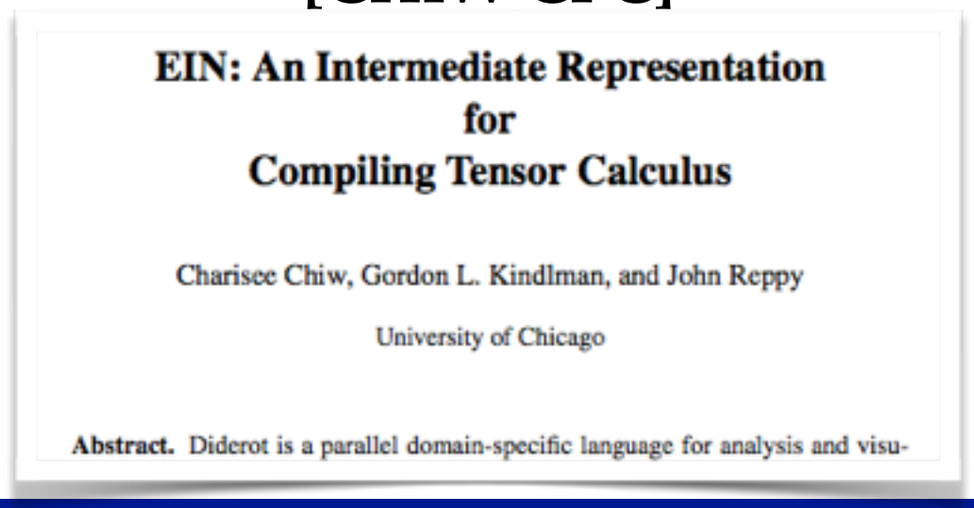
- Our intermediate representation, EIN based on Einstein Index Notation
- Concise way to represent operations on and between tensors and tensor fields

EIN Intermediate Representation

- Our intermediate representation, EIN based on Einstein Index Notation
- Concise way to represent operations on and between tensors and tensor fields

Our design and implementation →

[CHIW CPC]



EIN: An Intermediate Representation for Compiling Tensor Calculus

Charisee Chiw, Gordon L. Kindlman, and John Reppy

University of Chicago

Abstract. Diderot is a parallel domain-specific language for analysis and visu-

EIN Intermediate Representation

The matrix M is represented in EIN with two variable indices

Matrix M $\begin{bmatrix} M_{11} M_{12} \\ M_{21} M_{22} \end{bmatrix}$

EIN Intermediate Representation

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

EIN Intermediate Representation

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

EIN Intermediate Representation

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

EIN Intermediate Representation

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

Add diagonal components $trace(M) = \sum_i M_{ii}$ Summation

Single entry $M_{12} = M_{12}$ Set indices to constants

Select a column $\begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix} = M_{i1}$ Mix of different indices

EIN Intermediate Representation

- Multiply two vectors together

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

EIN Intermediate Representation

- Multiply two vectors together

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

EIN Intermediate Representation

- Multiply two vectors together

real

$$\sum_i A_i B_i$$

dot product

vector

$$A_i B_i$$

component-wise multiplication

matrix

$$A_i B_j$$

outer product

- Express more complicated computations

$$\lambda F \left\langle \sum_{ijk} F_{0i} F_{1j} F_{2k} \mathcal{E}_{ijk} \right\rangle$$

determinant

$$\lambda F \left\langle \frac{\sum_k F_{kk} \delta_{ij} - F_{ij}}{F_{00} F_{11} - F_{01} F_{10}} \right\rangle_{ij}$$

inverse

EIN Intermediate Representation

- Multiply two vectors together

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

- Express more complicated computations

$$\lambda F \left\langle \sum_{ijk} F_{0i} F_{1j} F_{2k} \mathcal{E}_{ijk} \right\rangle$$

determinant

$$\lambda F \left\langle \frac{\sum_k F_{kk} \delta_{ij} - F_{ij}}{F_{00} F_{11} - F_{01} F_{10}} \right\rangle_{ij}$$

inverse

An EIN term can concisely represent tensor math

EIN Intermediate Representation

Consider the computation

EIN Intermediate Representation

Consider the computation

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

EIN Intermediate Representation

Consider the computation

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

Diderot expression is represented in compiler as

$$M = \text{Outer}(a, b)$$
$$t = \text{Trace}(M)$$

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

EIN Intermediate Representation

Consider the computation

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

Diderot expression is represented in compiler as

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

substitution of the definition of M for X yields

$$t = \text{Trace}(\text{Outer}(\text{a}, \text{b})) \qquad t = (\lambda\,(A, B)\langle \textstyle\sum_i A_i B_i \rangle)(a, b)$$

EIN Intermediate Representation

Consider the computation

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

Diderot expression is represented in compiler as

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

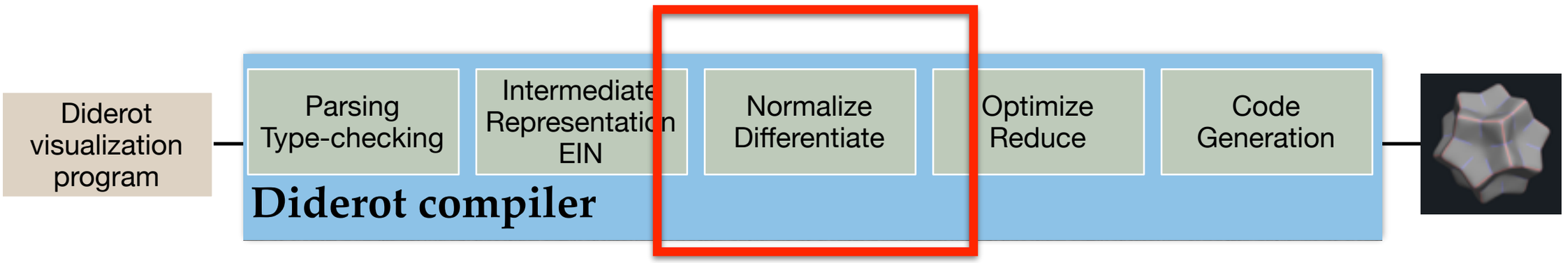
substitution of the definition of M for X yields

$$t = \text{Trace}(\text{Outer}(\text{a}, \text{b})) \qquad t = (\lambda\,(A, B)\langle \textstyle\sum_i A_i B_i \rangle)(a, b)$$

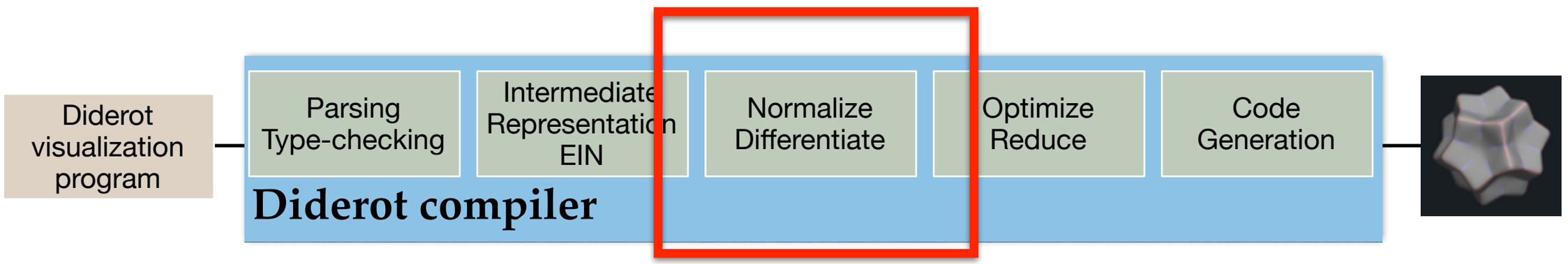
Replaces the ad hoc rewrite rule:

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

Compiling Diderot From Tensor Calculus to C

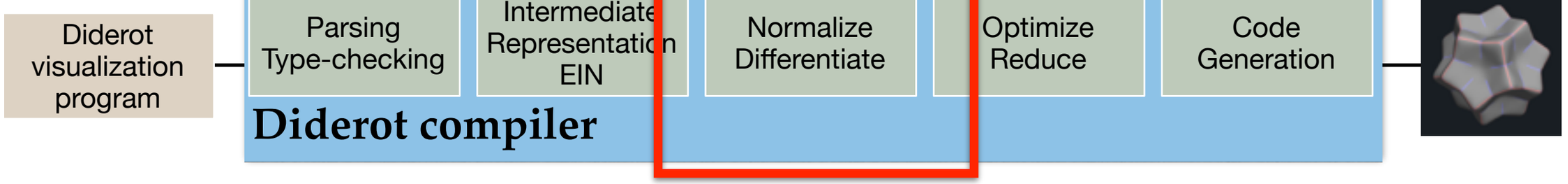


Compiling Diderot From Tensor Calculus to C



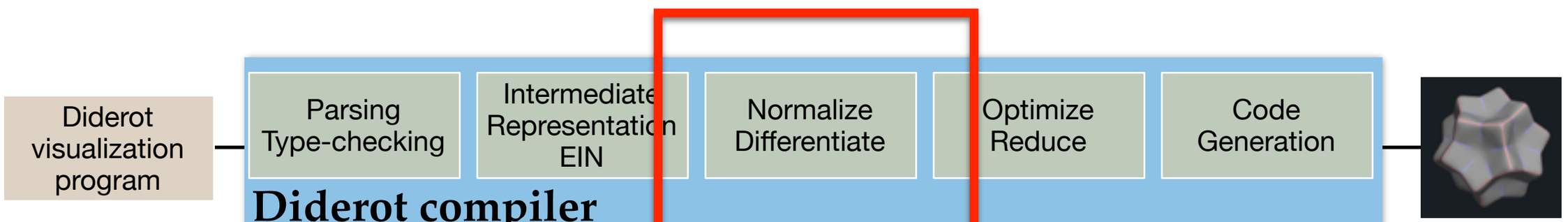
$$\nabla|\nabla F|(x)$$

Compiling Diderot From Tensor Calculus to C



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

Compiling Diderot From Tensor Calculus to C



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

Compilation Issue

Diderot visualization program

Parsing
Type-checking

Intermediate Representation
EIN

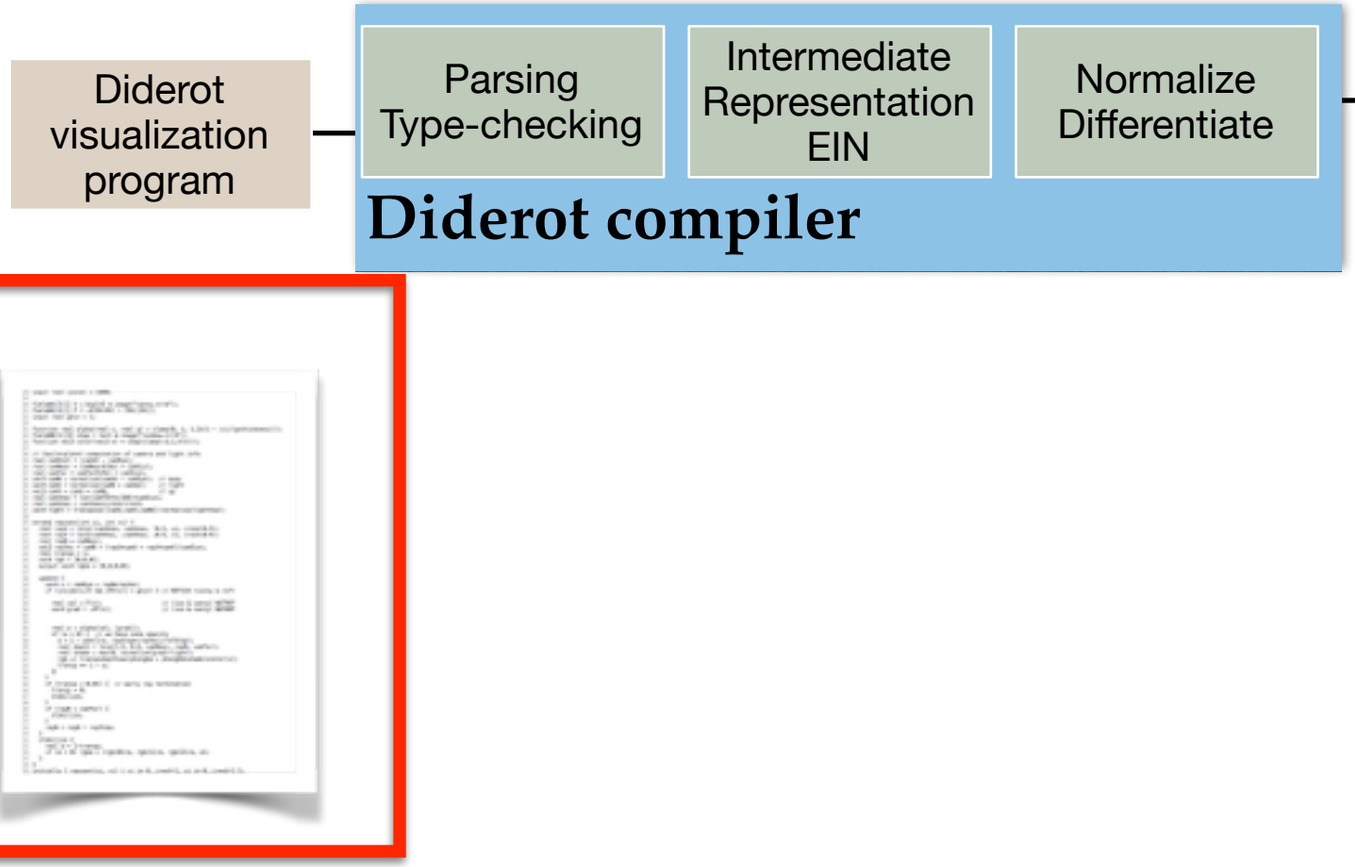
Normalize
Differentiate

Diderot compiler

Compilation Issue

Programs could not compile or took too long to compile

Compilation Issue

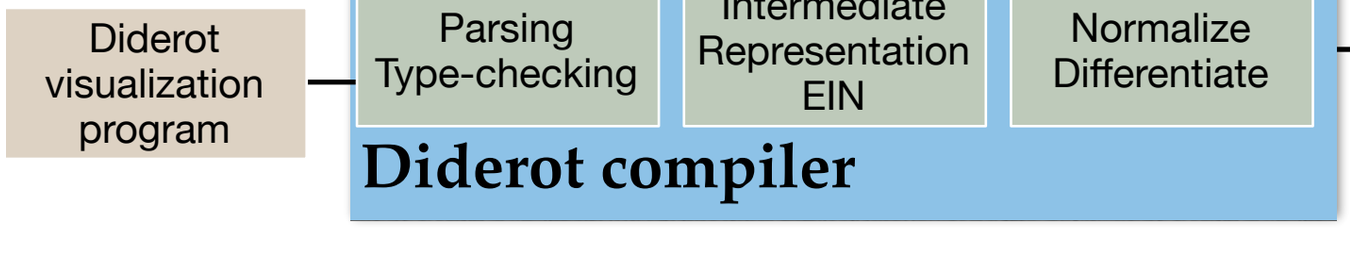


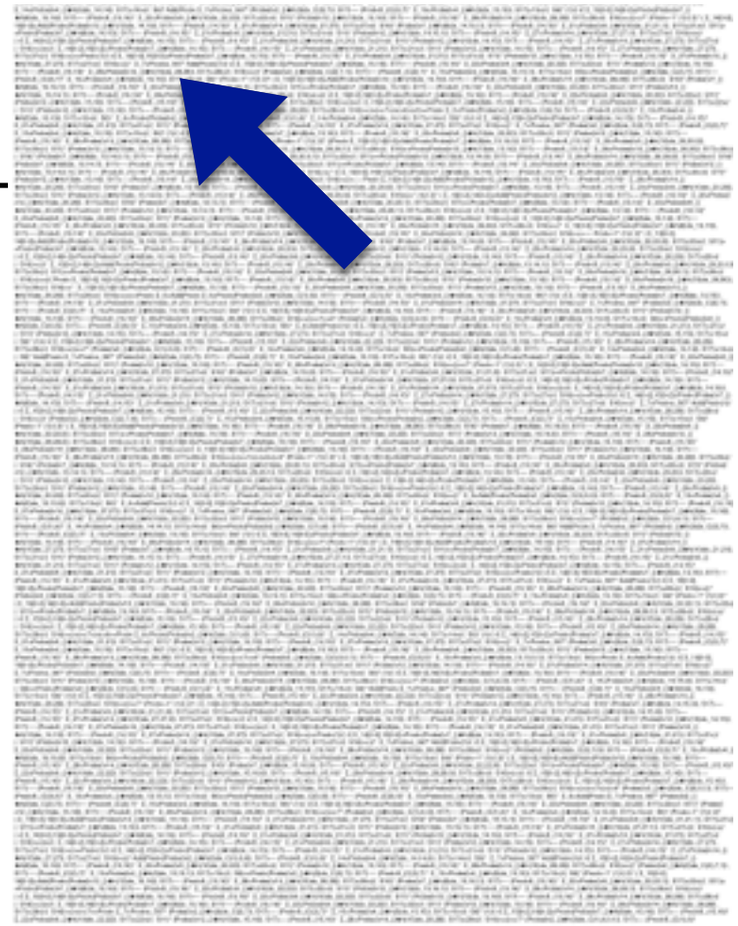
Compilation Issue

Programs could not compile or took too long to compile

Compilation Issue

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

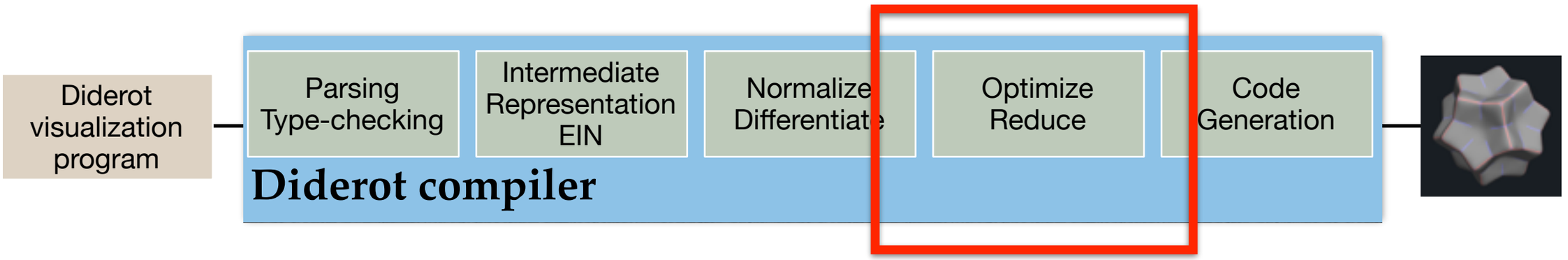




Programs could not compile or took too long to compile

Compilation Issue

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



Phase that enables effective compilation of tensor math

Compilation Issue

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

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

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

Split

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

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

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

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

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

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

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

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

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

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

Split

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

Slice

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

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

Write programs with higher-order operations

- Diderot is designed to represent general tensor math

Write programs with higher-order operations

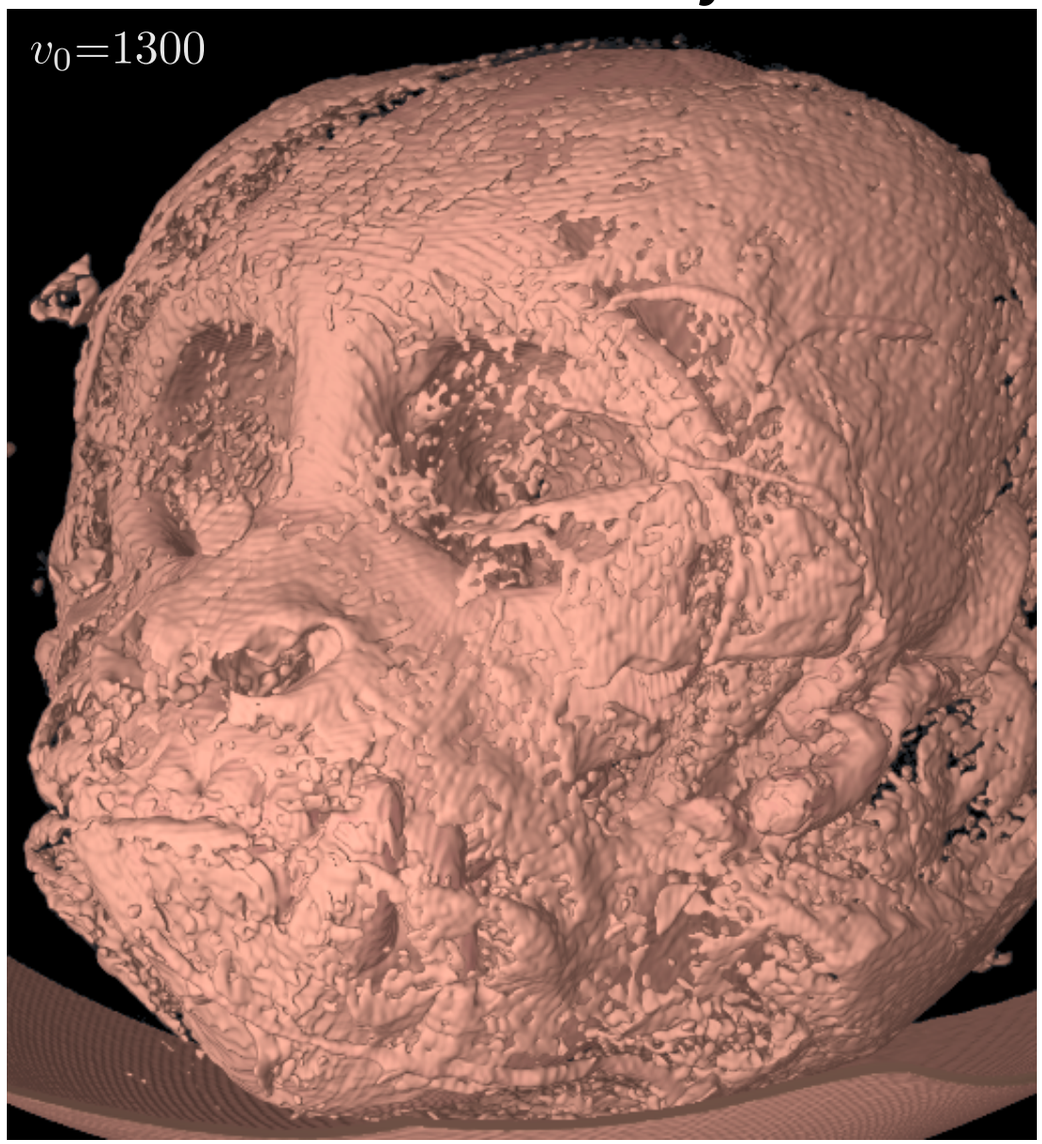
- Diderot is designed to represent general tensor math
- We rely on the expressivity of the language to support the implementation of visualization ideas.

Write programs with higher-order operations

- Diderot is designed to represent general tensor math
- We rely on the expressivity of the language to support the implementation of visualization ideas.
- Push development of compiler to enable sophisticated visualizations

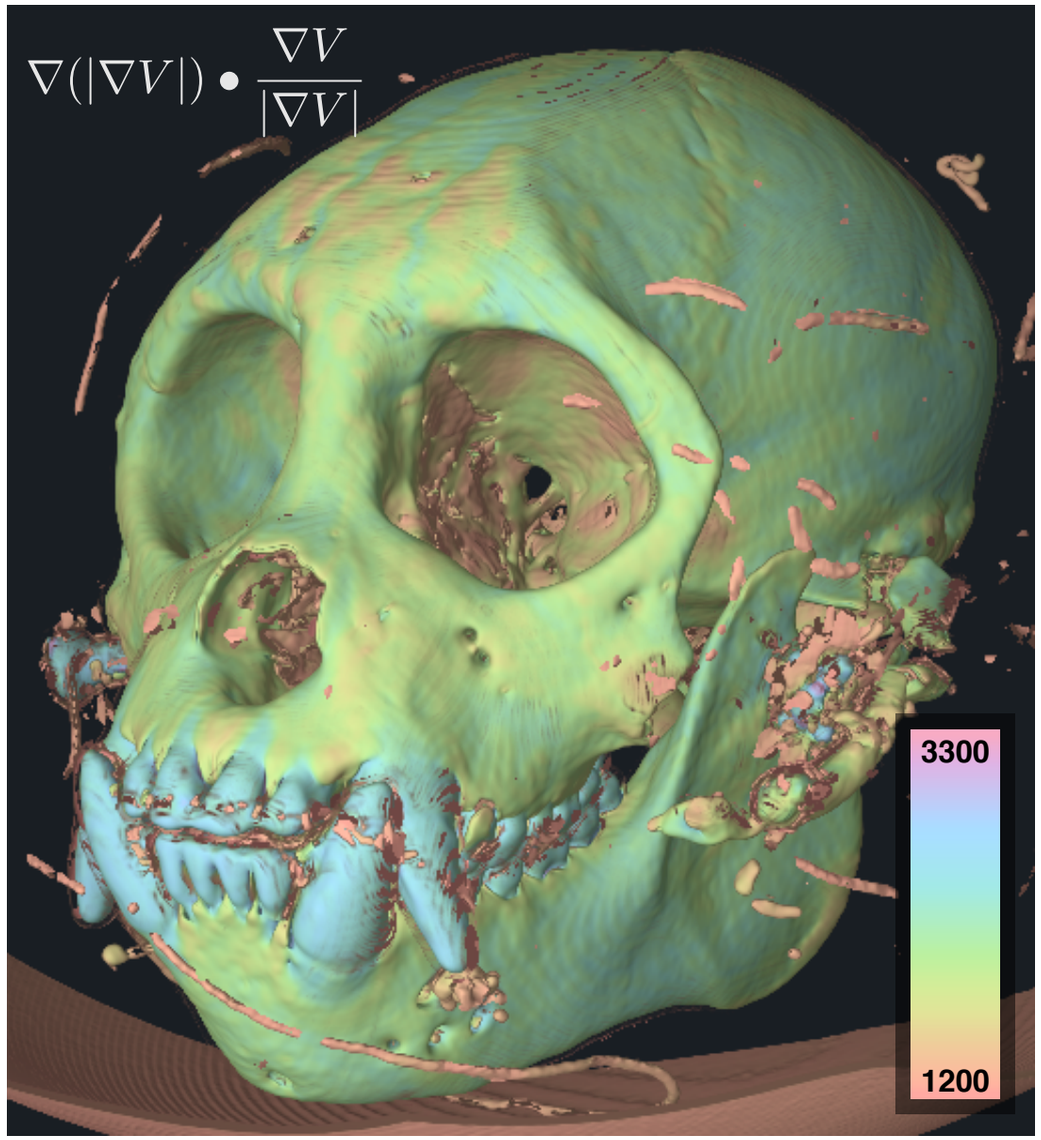
Results: Sophisticated Vis

Previously:



Iso-Surface

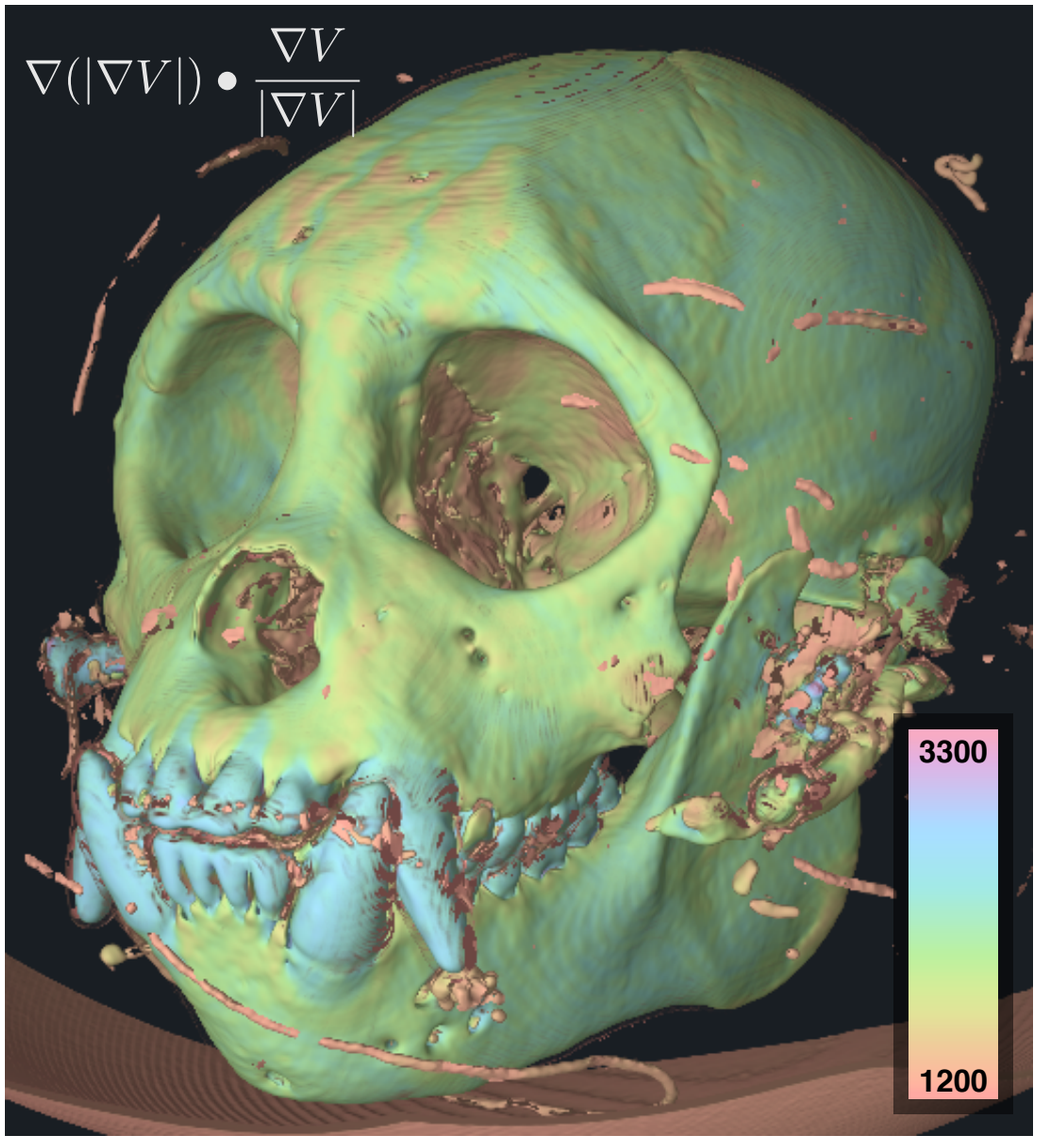
Now:



Canny Edges

Data provided by Callum Ross (U of Chicago)

Results: Sophisticated Vis

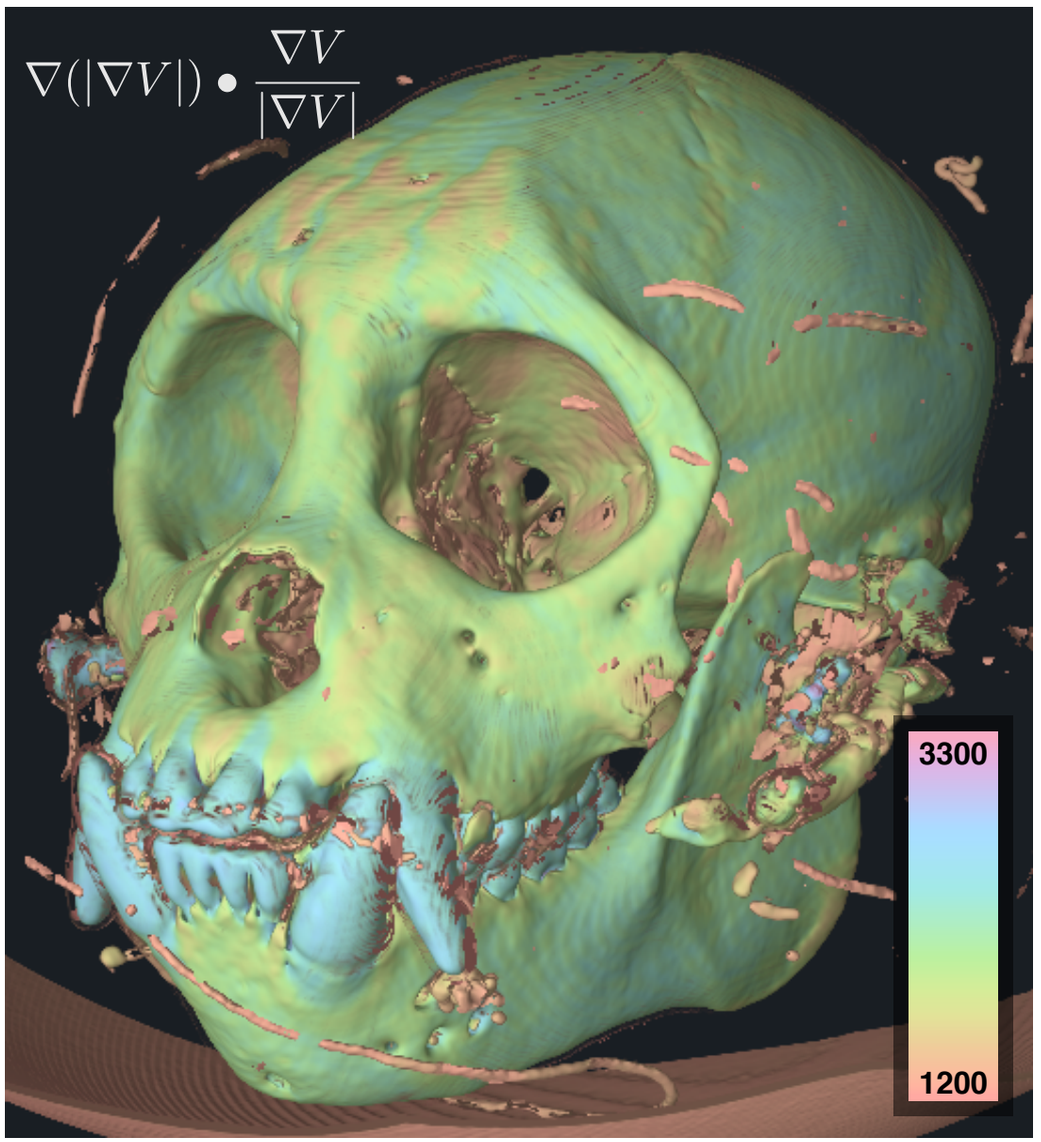


Canny Edges

Results: Sophisticated Vis

Inline high level syntax into Diderot code

```
−∇(|∇F|) • ∇F/|∇F|;
```



Canny Edges

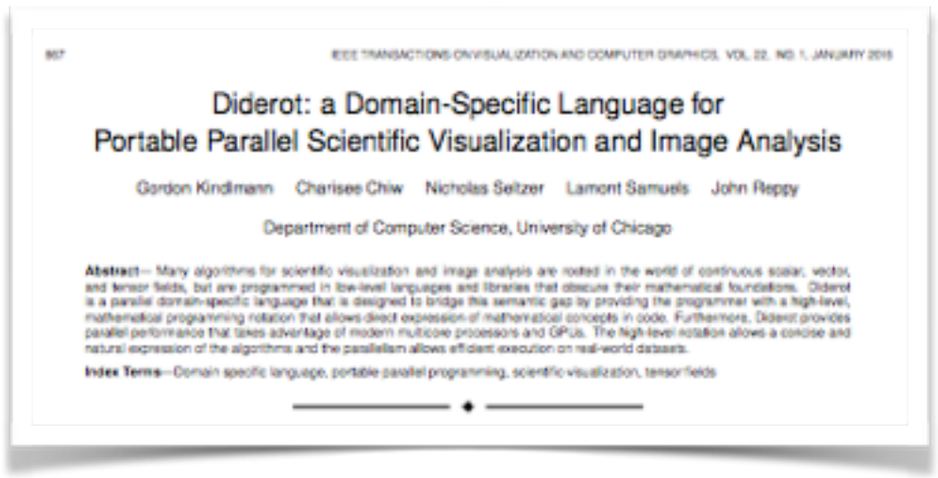
Results: Sophisticated Vis

Inline high level syntax into Diderot code

```
−∇(|∇F|) • ∇F/|∇F|;
```

Users focused on visualization

[KindImann VIS]



IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 22, NO. 1, JANUARY 2016

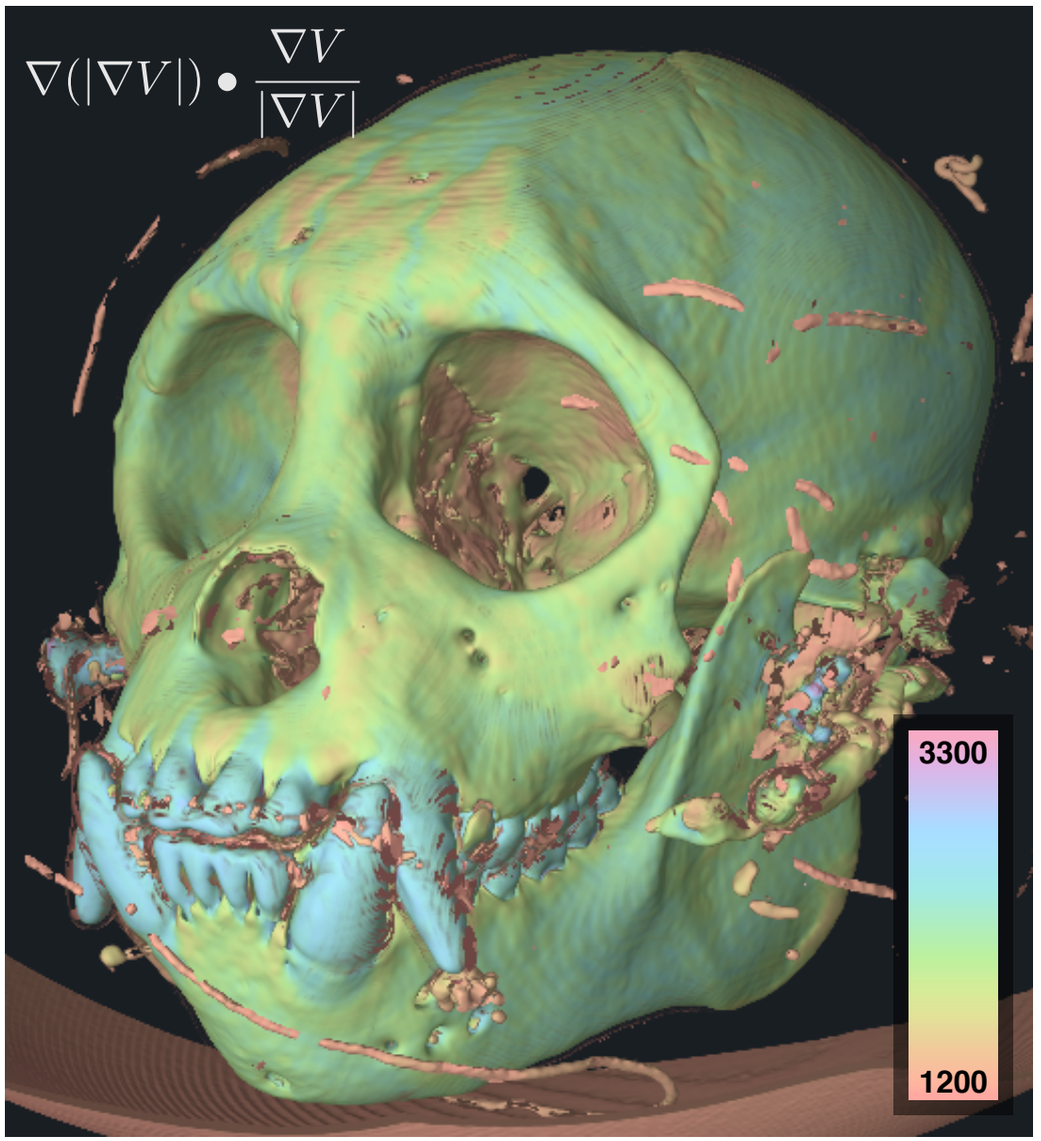
Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis

Gordon Kindlmann Charisee Chiw Nicholas Seltzer Lamont Samuels John Reppy

Department of Computer Science, University of Chicago

Abstract— Many algorithms for scientific visualization and image analysis are rooted in the world of continuous scalar, vector, and tensor fields, but are programmed in low-level languages and libraries that obscure their mathematical foundations. Diderot is a parallel domain-specific language that is designed to bridge this semantic gap by providing the programmer with a high-level, mathematical programming notation that allows direct expression of mathematical concepts in code. Furthermore, Diderot provides parallel performance that takes advantage of modern multicore processors and GPUs. The high-level notation allows a concise and natural expression of the algorithms and the parallelism allows efficient execution on real-world datasets.

Index Terms—Domain specific language, portable parallel programming, scientific visualization, tensor fields



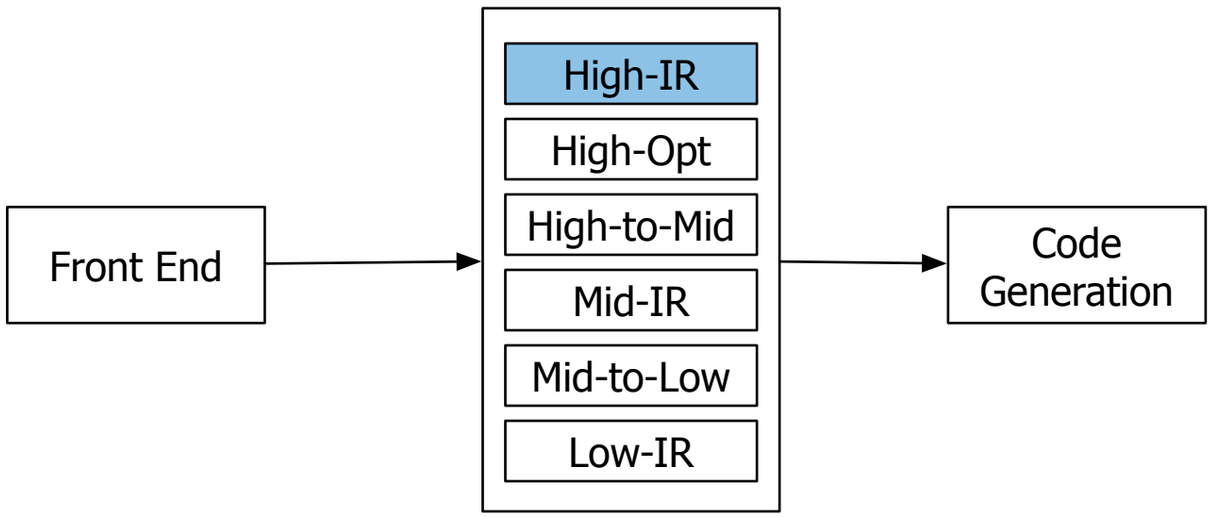
Canny Edges

Extra Slides

EIN

EIN IR

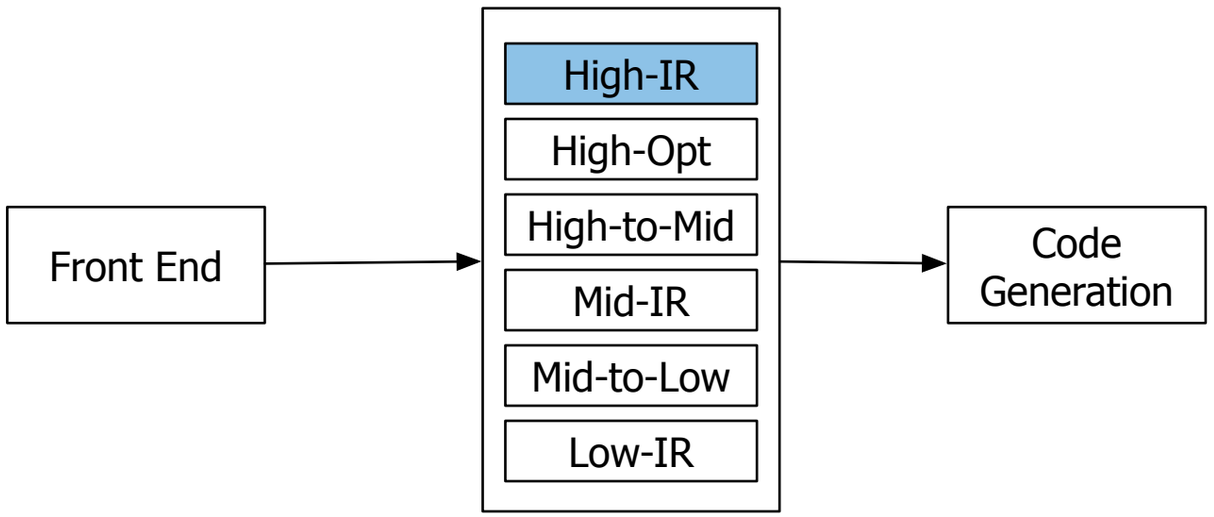
IR is in Static single assignment (SSA) form



Replace our fixed set of direct-style operators with an expressive language

EIN IR

IR is in Static single assignment (SSA) form

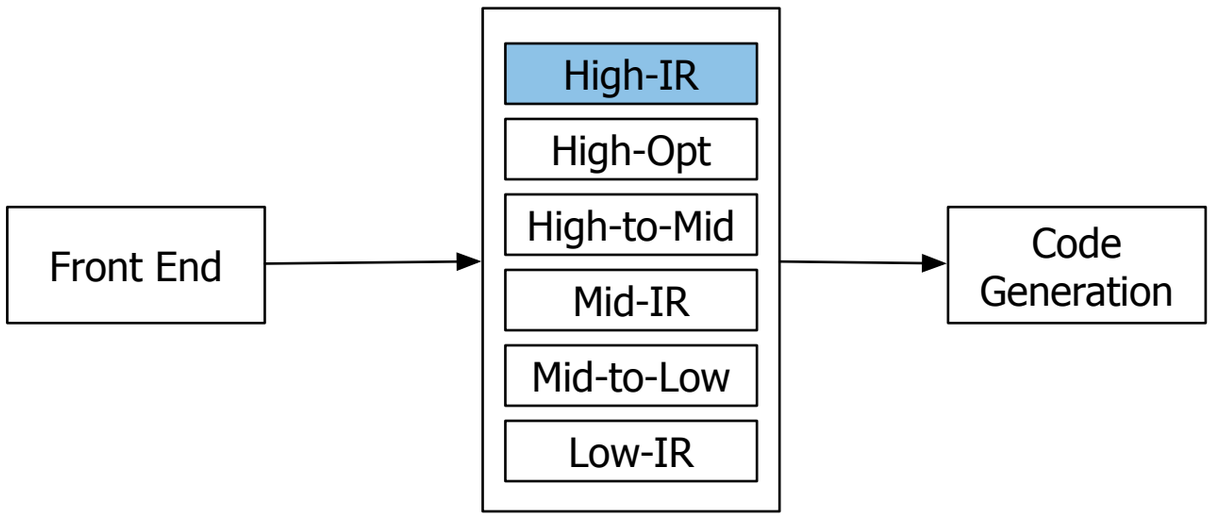


Replace our fixed set of direct-style operators with an expressive language

$$\mathbf{E} = \lambda\, \bar{x} \langle e \rangle_{\sigma}$$

EIN IR

IR is in Static single assignment (SSA) form



Replace our fixed set of direct-style operators with an expressive language

$$\mathbf{E} = \lambda\, \bar{x} \langle e \rangle_{\sigma}$$

whose semantics are specified by the expression e where $\bar{x}$ are tensor and field parameters, and σ determines the shape of the result.

EIN IR

$$\mathbf{E} \;=\; \lambda\, \bar{x} \langle e \rangle_{\sigma} \qquad \text{EIN Operator}$$

EIN IR

$$\mathbf{E} \;=\; \lambda\, \bar{x} \langle e \rangle_{\sigma} \qquad \text{EIN Operator}$$

- Provides a concise and expressive internal representation

EIN IR

- Provides a concise and expressive internal representation
- Terms include fields, images, permutation

$$
\begin{array}{rcll}
\mathbf{E} & = & \lambda\, \bar{x} \langle e \rangle_{\sigma} & \text{EIN Operator} \\
e & ::= & T_{\alpha}, A_{\alpha}, B_{\alpha} & \text{Tensor} \\
 & | & F_{\alpha}, G_{\alpha} & \text{Field} \\
 & | & \delta_{ij} & \text{Kronecker deltas} \\
 & | & \mathcal{E}_{ij}, \mathcal{E}_{ijk} & \text{Levi-Civita tensor}
\end{array}
$$

EIN IR

- Provides a concise and expressive internal representation
- Terms include fields, images, permutation
- Operators include differentiation, summation, domain-specific

$$
\begin{array}{rcll}
\mathbf{E} & = & \lambda\, \bar{x} \langle e \rangle_{\sigma} & \text{EIN Operator} \\
e & ::= & T_{\alpha}, A_{\alpha}, B_{\alpha} & \text{Tensor} \\
 & | & F_{\alpha}, G_{\alpha} & \text{Field} \\
 & | & \delta_{ij} & \text{Kronecker deltas} \\
 & | & \mathcal{E}_{ij}, \mathcal{E}_{ijk} & \text{Levi-Civita tensor} \\
 & | & \sum_{\sigma} e & \text{Summation} \\
 & | & \frac{\partial}{\partial x_{\alpha}} \diamond e & \text{Derivative of } e \\
 & | & e_1 @ e_2 & \text{Probe of a field} \\
 & | & \mathbf{lift}(e) & \text{Lift tensor } e \text{ to a field} \\
 & | & V_{\alpha} \circledast H^{\beta} & \text{Convolution} \\
 & | & \sqrt{e}, -e, \exp(e), e^n, \text{sine}(e) & \text{Unary operators} \\
 & | & e + e, e - e, \frac{e}{e}, ee & \text{Binary operators} \\
\sigma & = & \ldots & \text{index mapping}
\end{array}
$$

EIN IR

- Provides a concise and expressive internal representation
- Terms include fields, images, permutation
- Operators include differentiation, summation, domain-specific
- Index is bound to variable or constant

$$
\begin{array}{rcll}
\mathbf{E} & = & \lambda\, \bar{x} \langle e \rangle_{\sigma} & \text{EIN Operator} \\
e & ::= & T_{\alpha}, A_{\alpha}, B_{\alpha} & \text{Tensor} \\
 & | & F_{\alpha}, G_{\alpha} & \text{Field} \\
 & | & \delta_{ij} & \text{Kronecker deltas} \\
 & | & \mathcal{E}_{ij}, \mathcal{E}_{ijk} & \text{Levi-Civita tensor} \\
 & | & \sum_{\sigma} e & \text{Summation} \\
 & | & \frac{\partial}{\partial x_{\alpha}} \diamond e & \text{Derivative of } e \\
 & | & e_1 @ e_2 & \text{Probe of a field} \\
 & | & \mathbf{lift}(e) & \text{Lift tensor } e \text{ to a field} \\
 & | & V_{\alpha} \circledast H^{\beta} & \text{Convolution} \\
 & | & \sqrt{e}, -e, \exp(e), e^n, \text{sine}(e) & \text{Unary operators} \\
 & | & e + e, e - e, \frac{e}{e}, ee & \text{Binary operators} \\
i, j, k & \in & \textsc{IndexVar} & \text{Variable index} \\
n & \in & \textsc{ConstVar} = \mathbb{N} & \text{Constant index} \\
\mu & \in & \textsc{IndexVar} \cup \textsc{ConstVar} & \text{Single index} \\
\alpha, \beta, \gamma & = & \bar{\mu} & \text{Sequence of indices} \\
\sigma & = & (\mathbb{Z} \times \textsc{IndexVar} \times \mathbb{Z})^* & \text{index mapping}
\end{array}
$$

Compilation Issue

$$y = \lambda Params \langle e_{ij} e_{1k} - e_{0i} e_{jk} + e_{ij} e_{k1} \rangle_{ijk} (args)$$

Compilation Issue

$$y = \lambda Params \langle e_{ij} e_{1k} - e_{0i} e_{jk} + e_{ij} e_{k1} \rangle_{ijk} (args)$$

- Consider example: $e = f \otimes (a + b)$

Compilation Issue

$$y = \lambda F, A, B \left\langle (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{1l}(A_{lk} + B_{lk})) - (\sum_l F_{0l}(A_{li} + B_{li}))(\sum_l F_{jl}(A_{lk} + B_{lk})) + (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{kl}(A_{l1} + B_{l1})) \right\rangle_{ijk} (f, a, b)$$

- Consider example: $e = f \otimes (a + b)$

Compilation Issue

$$y = \lambda F, A, B \left\langle (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{1l}(A_{lk} + B_{lk})) - (\sum_l F_{0l}(A_{li} + B_{li}))(\sum_l F_{jl}(A_{lk} + B_{lk})) + (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{kl}(A_{l1} + B_{l1})) \right\rangle_{ijk} (f, a, b)$$

- Consider example: $e = f \otimes (a + b)$
- Pull it out and create a new operator

$$z = \lambda F, A, B, x \left\langle \sum_k F_{ik}(A_{kj} + B_{kj}) \right\rangle_{ij} (f, a, b, x)$$

Compilation Issue

$$y = \lambda F, A, B \left\langle (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{1l}(A_{lk} + B_{lk})) - (\sum_l F_{0l}(A_{li} + B_{li}))(\sum_l F_{jl}(A_{lk} + B_{lk})) + (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{kl}(A_{l1} + B_{l1})) \right\rangle_{ijk} (f, a, b)$$

- Consider example: $e = f \otimes (a + b)$
- Pull it out and create a new operator

$$z = \lambda F, A, B, x \left\langle \sum_k F_{ik}(A_{kj} + B_{kj}) \right\rangle_{ij} (f, a, b, x)$$

- Replace subexpression with tensor term

$$y' = \lambda F, A, B, Z \left\langle Z_{ij}(\sum_l F_{1l}(A_{lk} + B_{lk})) - (\sum_l F_{0l}(A_{li} + B_{li}))Z_{jk} + Z_{ij}(\sum_l F_{kl}(A_{l1} + B_{l1})) \right\rangle_{ijk} (f, a, b, z)$$

Compilation Issue

$$y = \lambda F, A, B \left\langle (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{1l}(A_{lk} + B_{lk})) - (\sum_l F_{0l}(A_{li} + B_{li}))(\sum_l F_{jl}(A_{lk} + B_{lk})) + (\sum_l F_{il}(A_{lj} + B_{lj}))(\sum_l F_{kl}(A_{l1} + B_{l1})) \right\rangle_{ijk} (f, a, b)$$

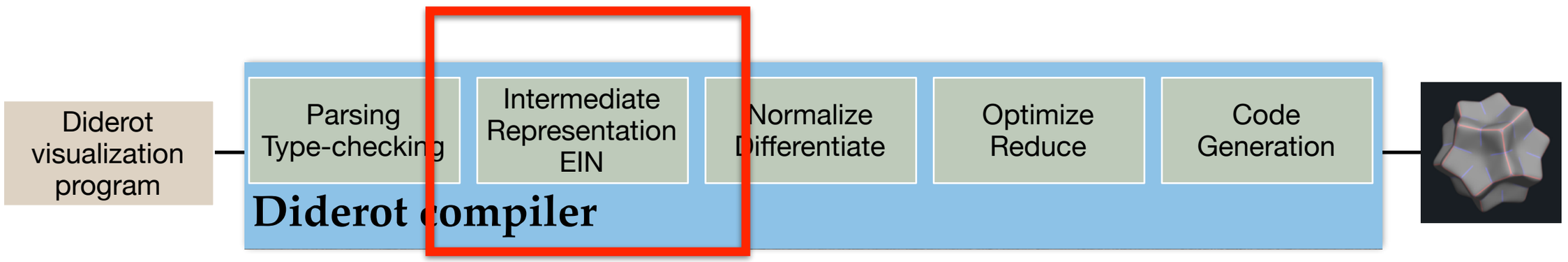
- Consider example: $e = f \otimes (a + b)$
- Pull it out and create a new operator

$$z = \lambda F, A, B, x \left\langle \sum_k F_{ik}(A_{kj} + B_{kj}) \right\rangle_{ij} (f, a, b, x)$$

- Replace subexpression with tensor term

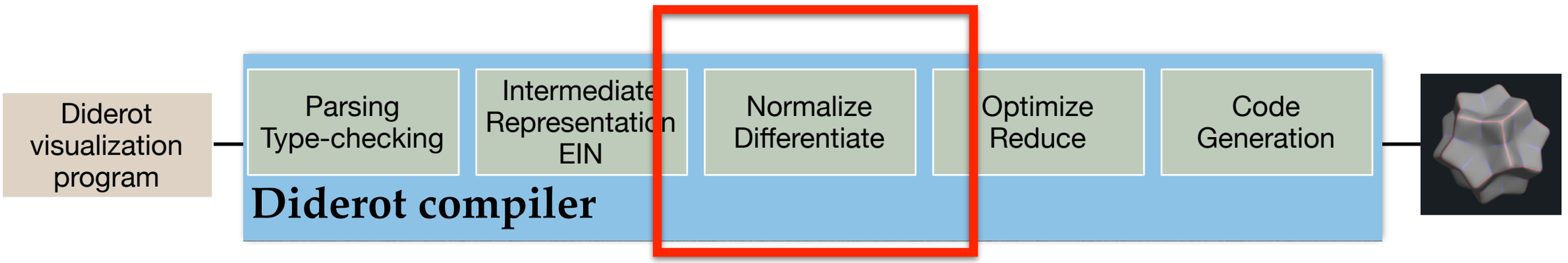
$$y' = \lambda F, A, B, Z \left\langle Z_{ij}(\sum_l F_{1l}(A_{lk} + B_{lk})) - (\sum_l F_{0l}(A_{li} + B_{li}))Z_{jk} + Z_{ij}(\sum_l F_{kl}(A_{l1} + B_{l1})) \right\rangle_{ijk} (f, a, b, z)$$

EIN Intermediate Representation

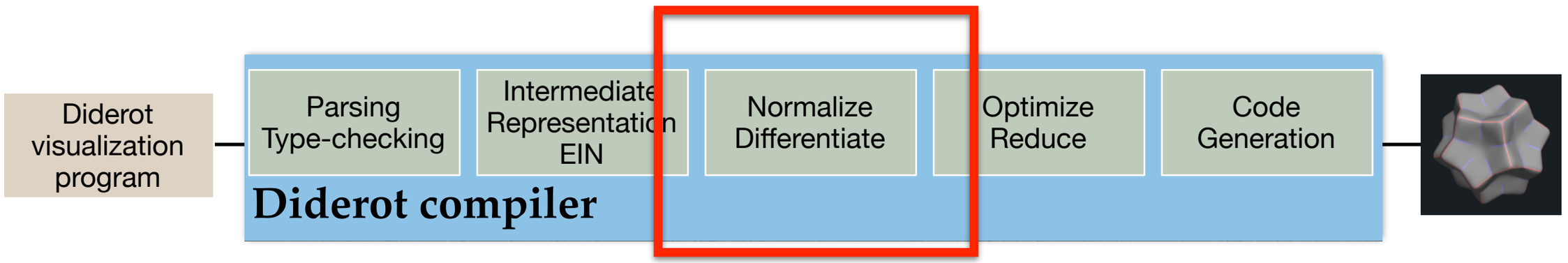


Phase that represents tensor math

Compiling Diderot From Tensor Calculus to C

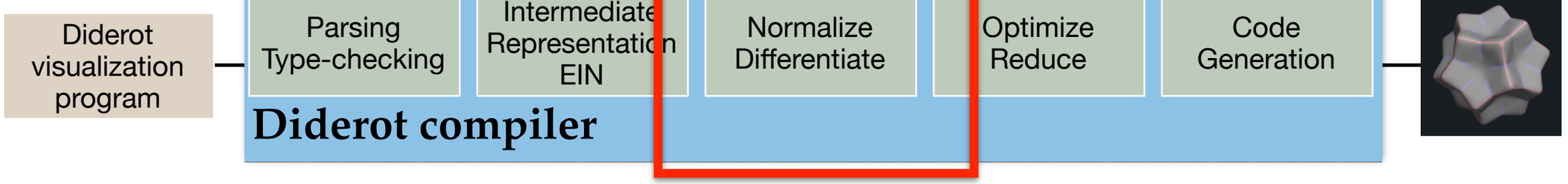


Compiling Diderot
From Tensor Calculus to C



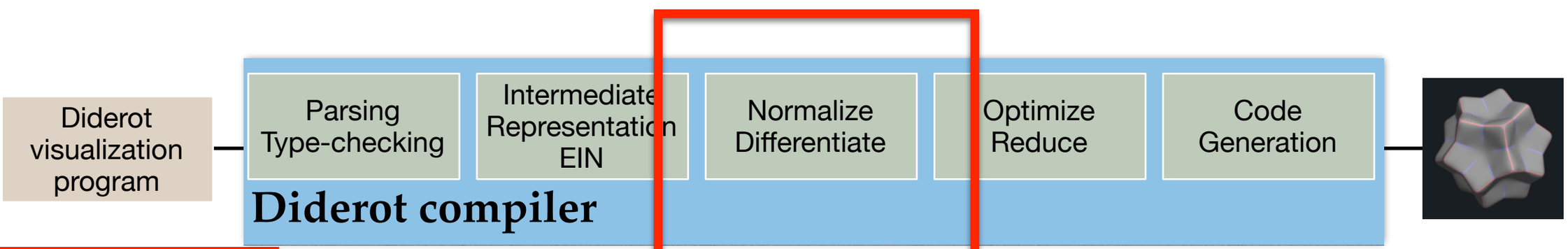
$$\nabla|\nabla F|(x)$$

Compiling Diderot From Tensor Calculus to C



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

Compiling Diderot From Tensor Calculus to C



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

Compilation Issue

Diderot visualization program

Parsing
Type-checking

Intermediate Representation
EIN

Normalize
Differentiate

Diderot compiler

Compilation Issue

Programs could not compile or took too long to compile

Compilation Issue

Diderot visualization program

Parsing
Type-checking

Intermediate Representation
EIN

Normalize
Differentiate

Diderot compiler

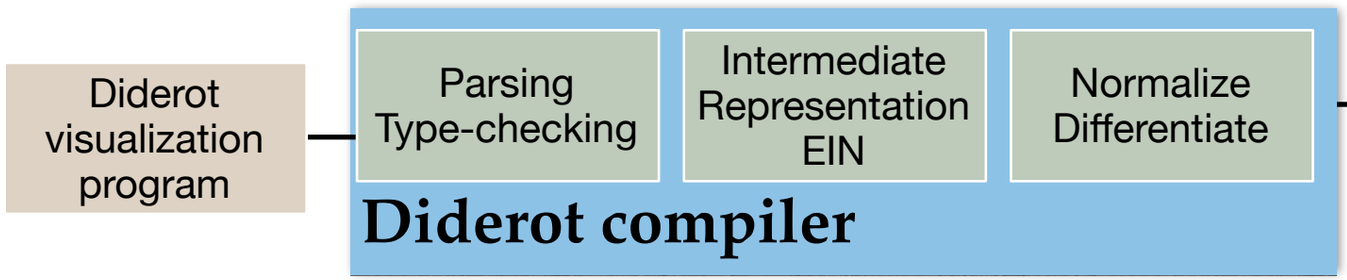
Compilation Issue

Programs could not compile or took too long to compile

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$

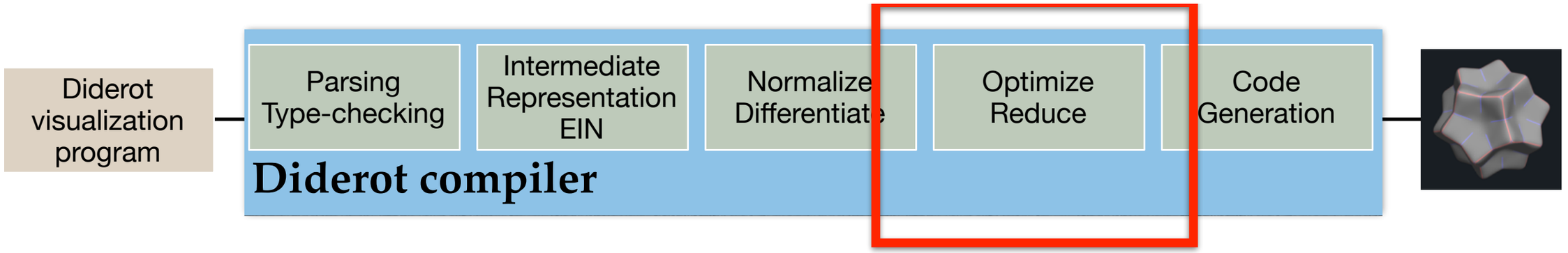
Issue



Programs could not compile or took too long to compile

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$



Phase that enables effective compilation of tensor math

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$

To reduce the size of the IR we want to

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$

To reduce the size of the IR we want to

- enable traditional compiler approaches (cse)

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$

To reduce the size of the IR we want to

- enable traditional compiler approaches (cse)

Terms can be syntactically different even when they

- are doing the same thing

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$

To reduce the size of the IR we want to

- enable traditional compiler approaches (cse)

Terms can be syntactically different even when they

- are doing the same thing
- create some of the same computations later

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$

To reduce the size of the IR we want to

- enable traditional compiler approaches (cse)

Terms can be syntactically different even when they

- are doing the same thing
- create some of the same computations later
- not the same at all

Compilation Issue

$$e_{ij}e_{1k} - e_{0i}e_{jk} + e_{ij}e_{k1}$$

To reduce the size of the IR we want to

- enable traditional compiler approaches (cse)

Terms can be syntactically different even when they

- are doing the same thing
- create some of the same computations later
- not the same at all

Our design and implementation →

[CHIW CPC]

EIN: An Intermediate Representation for Compiling Tensor Calculus

Charisee Chiw, Gordon L. Kindlman, and John Reppy

University of Chicago

Abstract. Diderot is a parallel domain-specific language for analysis and visu-