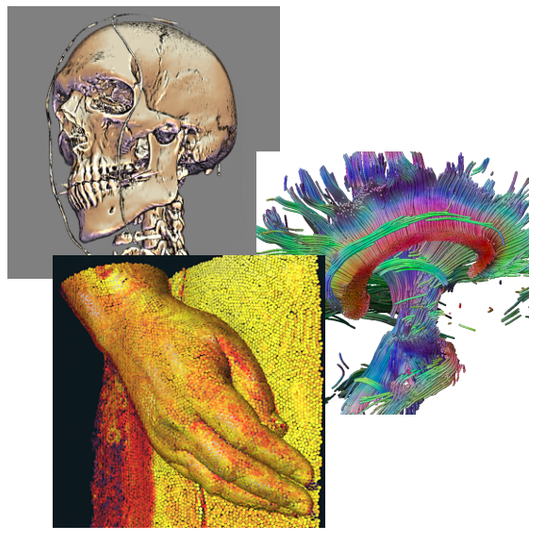
Addressing Domain-Specific Computational Needs with a New Programming Language

Charisee Chiw



THE UNIVERSITY OF CHICAGO
COMPUTER SCIENCE

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, ... 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
- 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, 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),

Roadmap

Roadmap

1. Implementing expressivity

Roadmap

1. Implementing expressivity
2. Automated Testing of compiler implementation

Roadmap

1. Implementing expressivity
2. Automated Testing of compiler implementation
3. Extension to a new domain

Roadmap

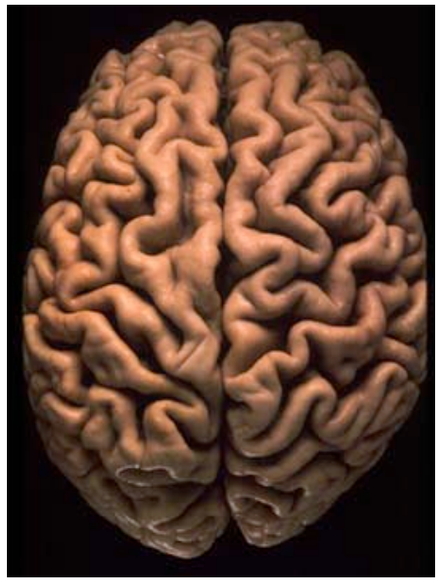
0. Image analysis and scientific visualization
1. Implementing expressivity
2. Automated Testing of compiler implementation
3. Extension to a new domain

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

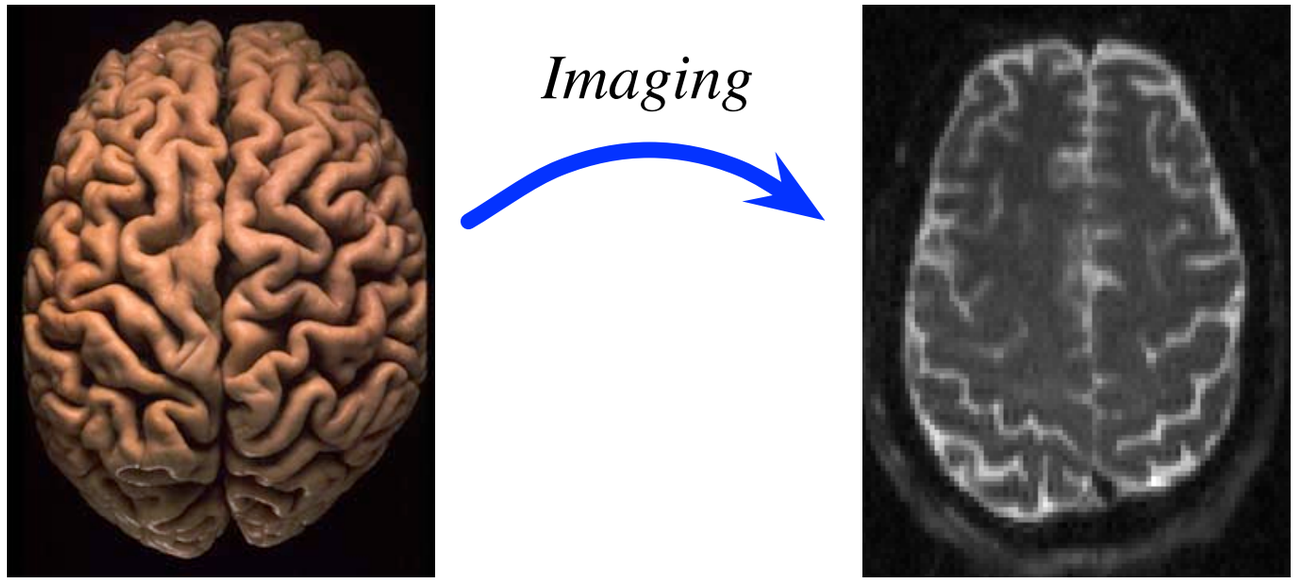
- Scientists need software tools to extract structure from many kinds of image data.
- Creating new visualization programs is part of the experimental process.
- Visualization helps explore data.



Physical object

Why image analysis is important

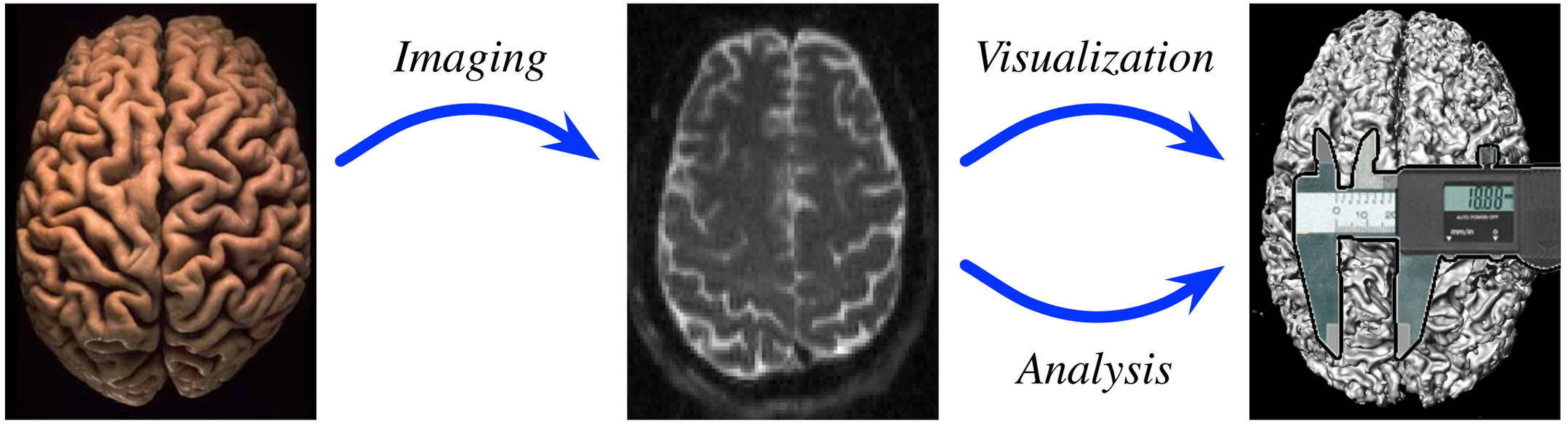
- Scientists need software tools to extract structure from many kinds of image data.
- Creating new visualization programs is part of the experimental process.
- Visualization helps explore data.



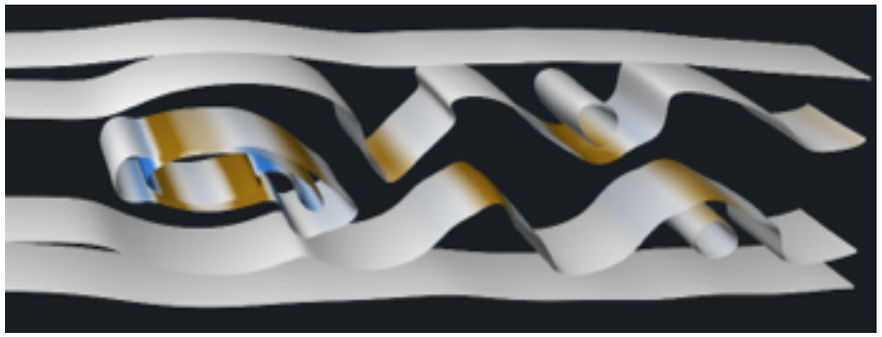
Physical object **Image data**

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.

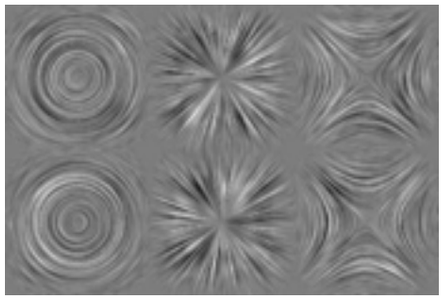


Visualization Applications

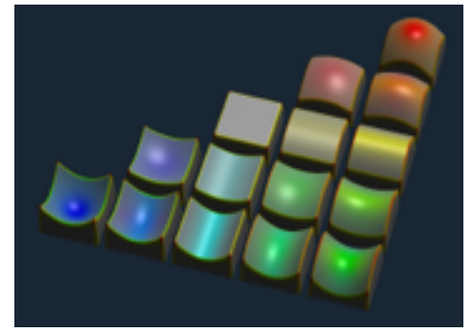


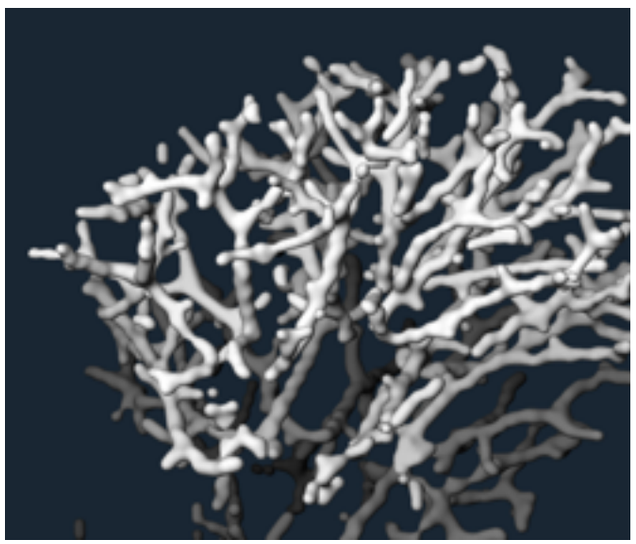
Fluid flow

Diderot ^ The guy

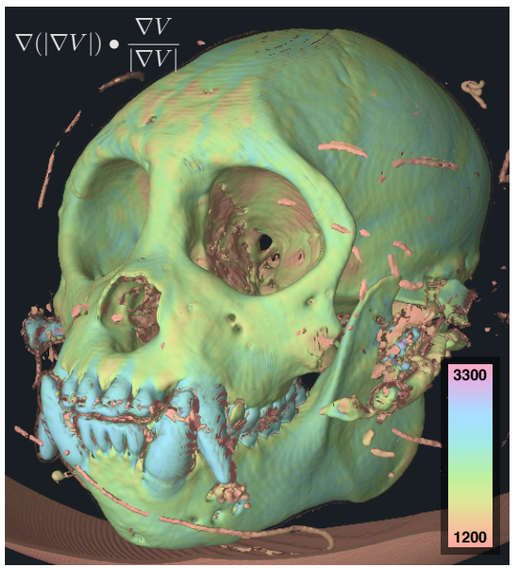


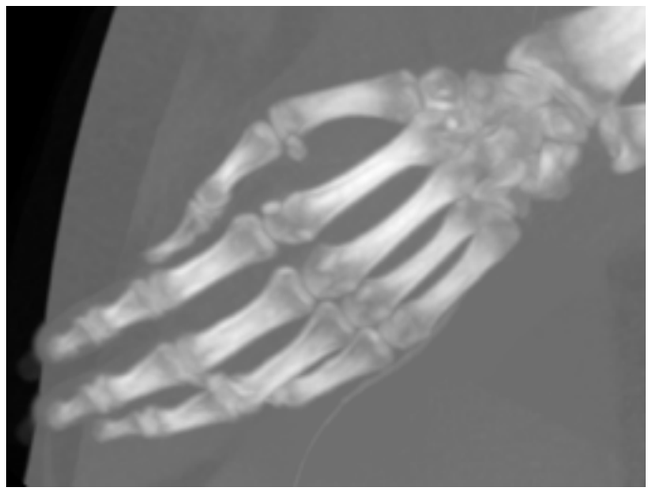
Synthetic data

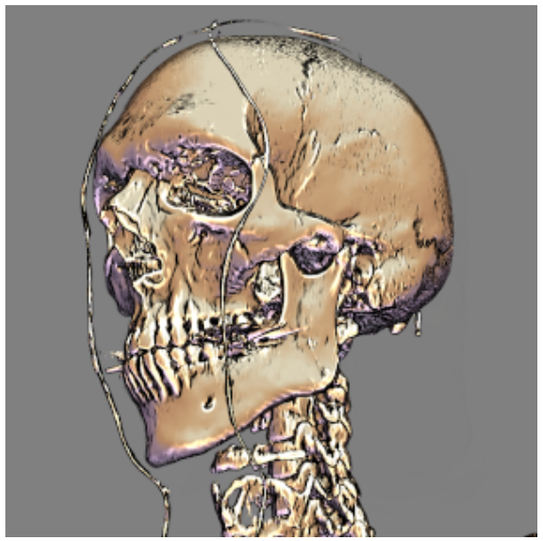




Lung data





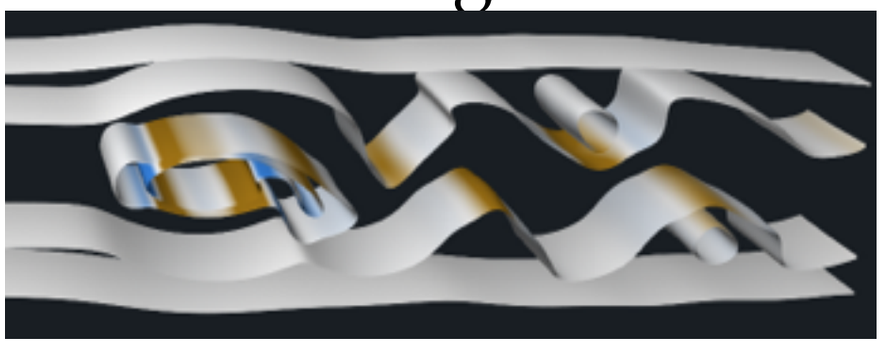


X-Ray data and CT Scans

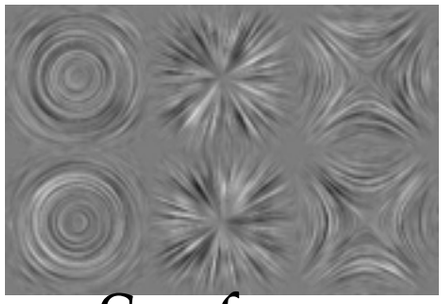
Visualization Applications

Visually explore data and compute features

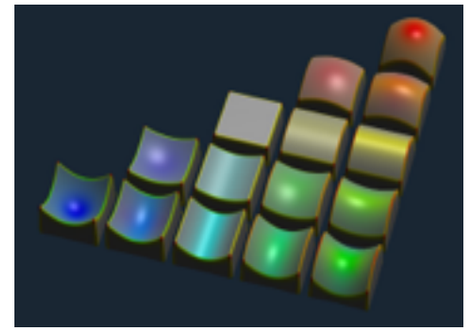
Flow Magnitude



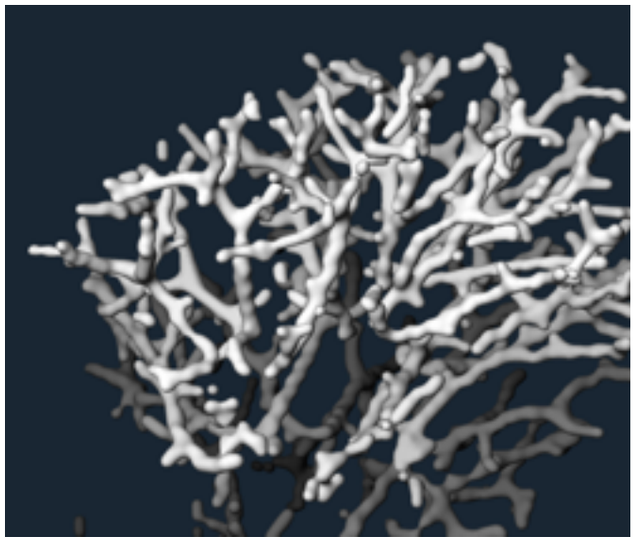
LIC



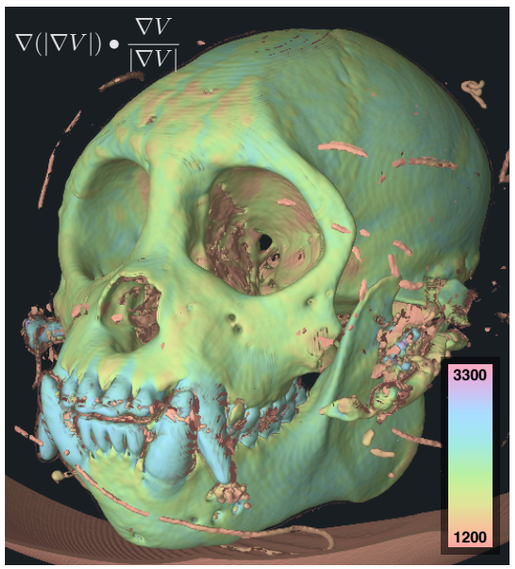
Surface Curvature



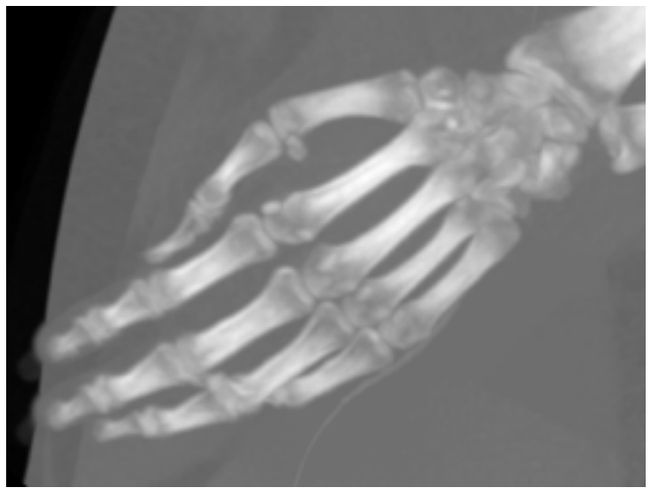
Canny Edges



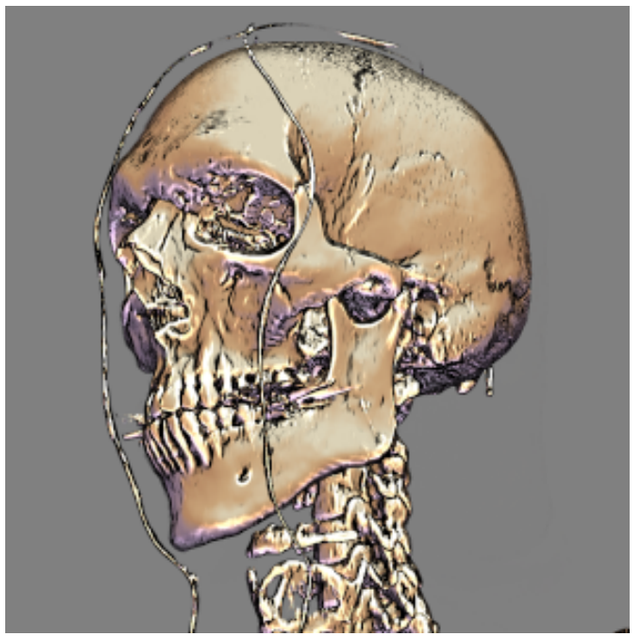
Ridge Detection



Isocontour Surface MIP



Volume rendering



Types and definitions

Tensor: reals, vectors, matrices,… 9

Tensor operators: additional, subtraction, multiplication 9+3

Types and definitions

Tensor: reals, vectors, matrices,… 9

Tensor operators: additional, subtraction, multiplication 9+3

Field: function from d-dimensional space to tensors F(x) = x^2

Probe: fields are probed at a position to produce a tensor F(3) = 9

Types and definitions

Tensor: reals, vectors, matrices,… 9

Tensor operators: additional, subtraction, multiplication 9+3

Field: function from d-dimensional space to tensors F(x) = x^2

Probe: fields are probed at a position to produce a tensor F(3) = 9

Derivative: rate of change ∇F = 2x

Applying the derivative operator ∇(F+G) => ∇F+ ∇G

Types and definitions

Tensor: reals, vectors, matrices,… 9

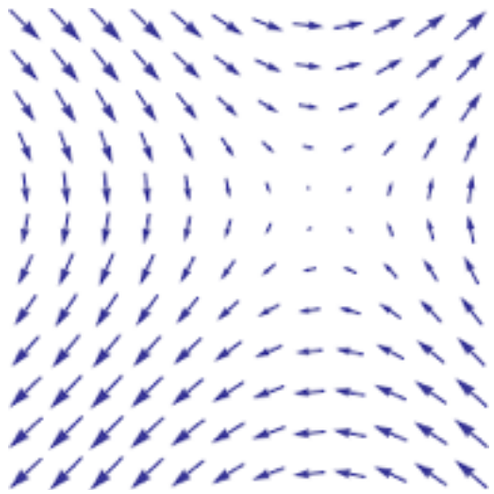
Tensor operators: additional, subtraction, multiplication 9+3

Field: function from d-dimensional space to tensors F(x) = x^2

Probe: fields are probed at a position to produce a tensor F(3) = 9

Derivative: rate of change ∇F = 2x

Applying the derivative operator ∇(F+G) => ∇F+ ∇G



vector field is a type of field

vector is a type of tensor

Types and definitions

Tensor: reals, vectors, matrices,… 9

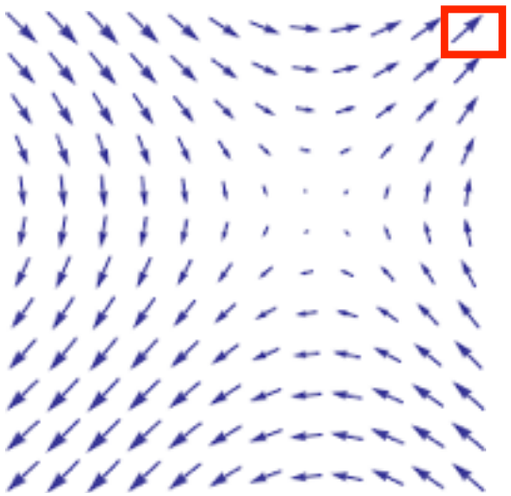
Tensor operators: additional, subtraction, multiplication 9+3

Field: function from d-dimensional space to tensors F(x) = x^2

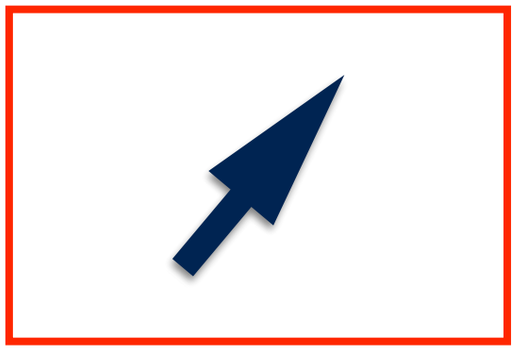
Probe: fields are probed at a position to produce a tensor F(3) = 9

Derivative: rate of change ∇F = 2x

Applying the derivative operator ∇(F+G) => ∇F+ ∇G



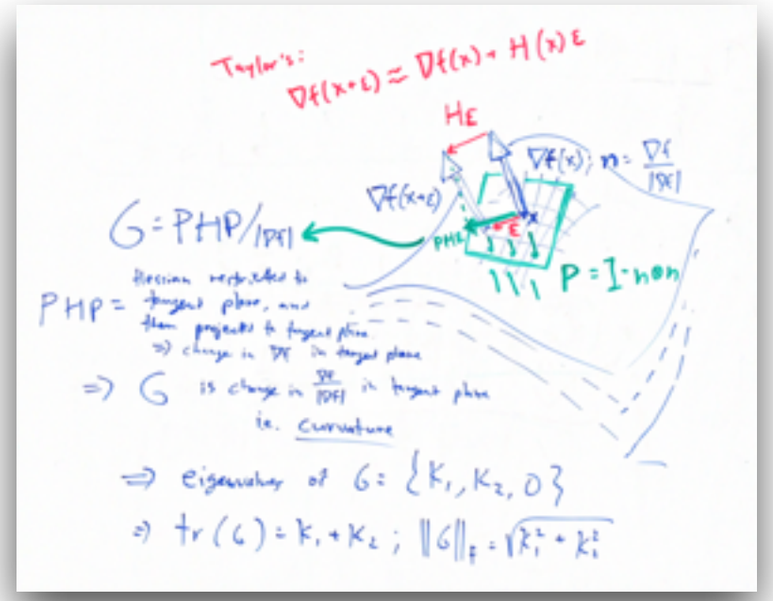
vector field is a type of field

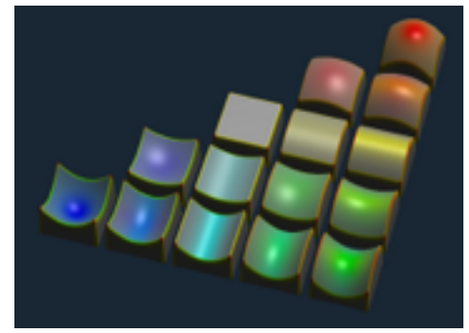


vector is a type of tensor

Challenges

- What do I want to implement?



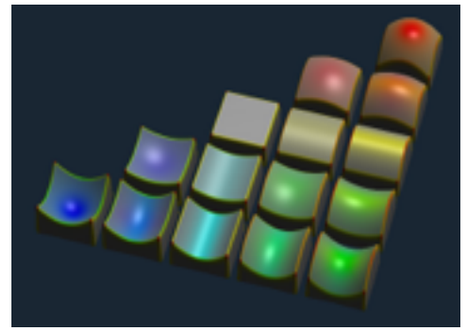


Visualization concept **Implementation** **Image**

Challenges

- What do I want to implement?

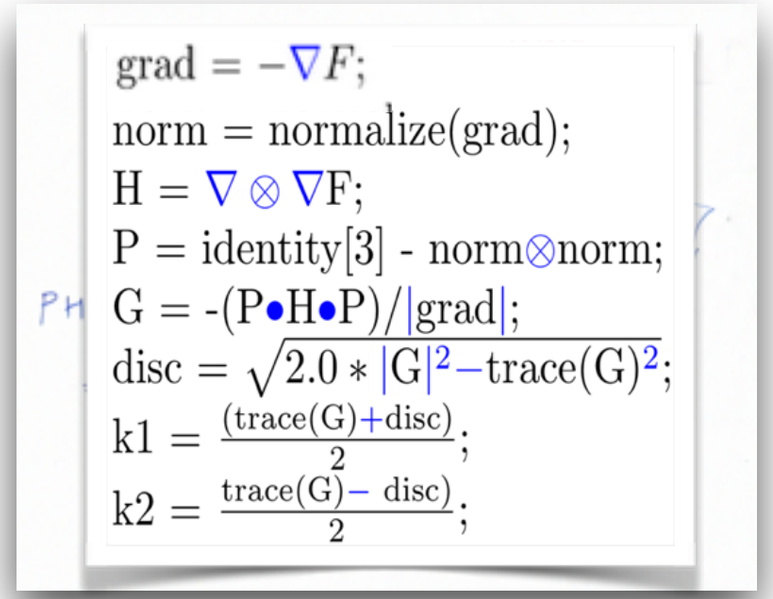
$$
\begin{aligned}
&\text{grad} = -\nabla F;\\
&\text{norm} = \text{normalize}(\text{grad});\\
&\text{H} = \nabla \otimes \nabla \text{F};\\
&\text{P} = \text{identity}[3] - \text{norm} \otimes \text{norm};\\
&\text{G} = -(\text{P} \bullet \text{H} \bullet \text{P})/|\text{grad}|;\\
&\text{disc} = \sqrt{2.0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

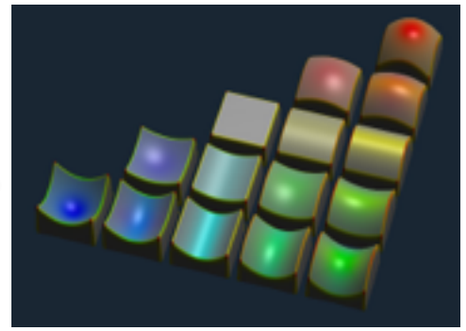


Visualization concept **Implementation** **Image**

Challenges

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

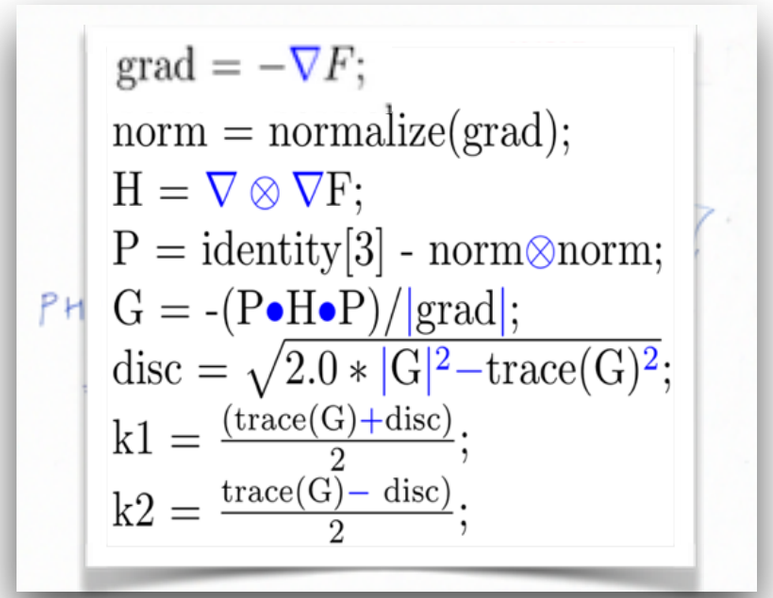


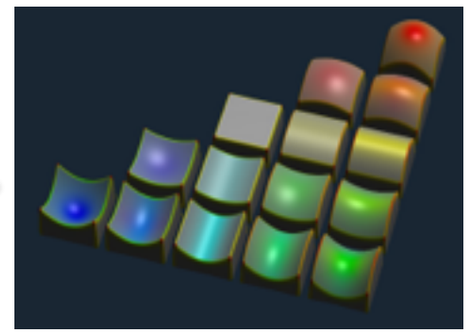


Visualization concept **Implementation** **Image**

Challenges

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



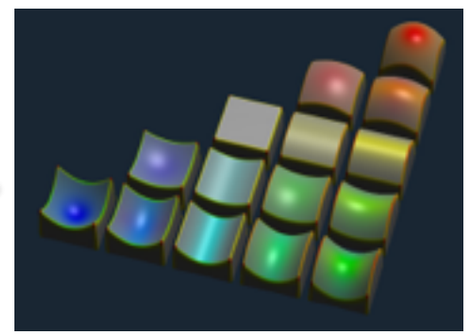


Visualization concept **Implementation** **Image**

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

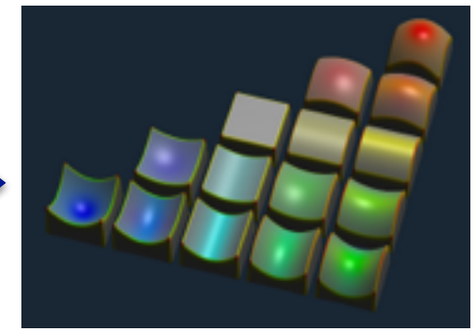
Image

Programmability: From whiteboard to code

Surface Curvature

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

```
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;
```

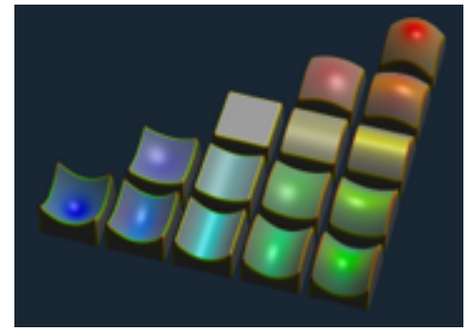


Visualization concept **Diderot code** **Image**

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

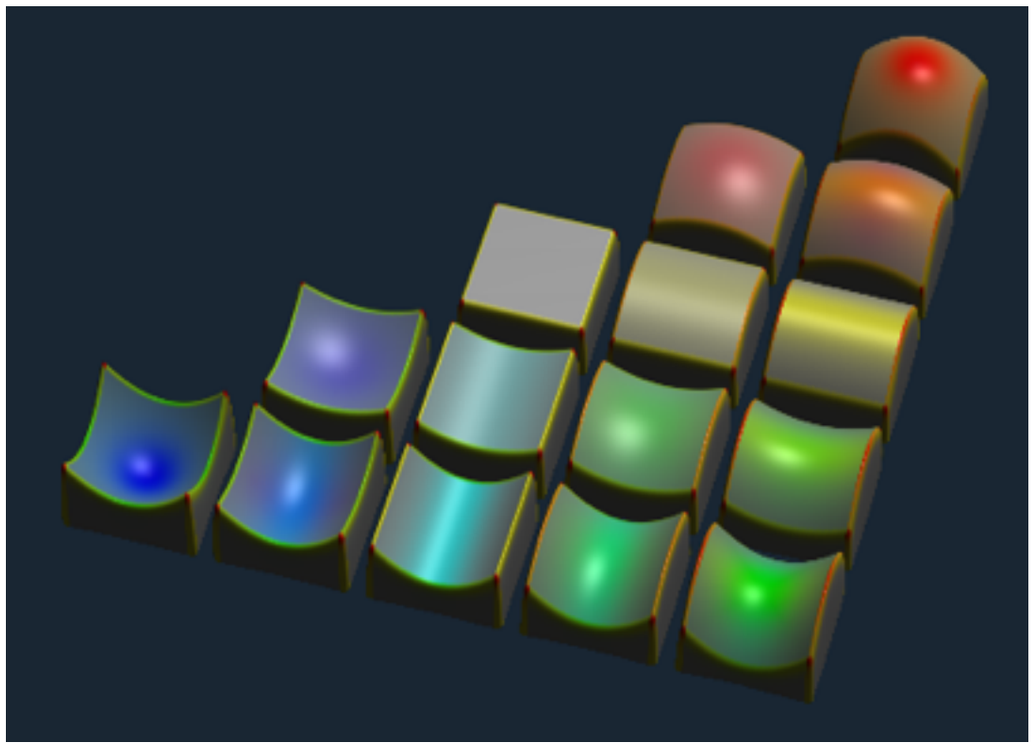
- Language restrictions make it difficult to implement new ideas

Motivation

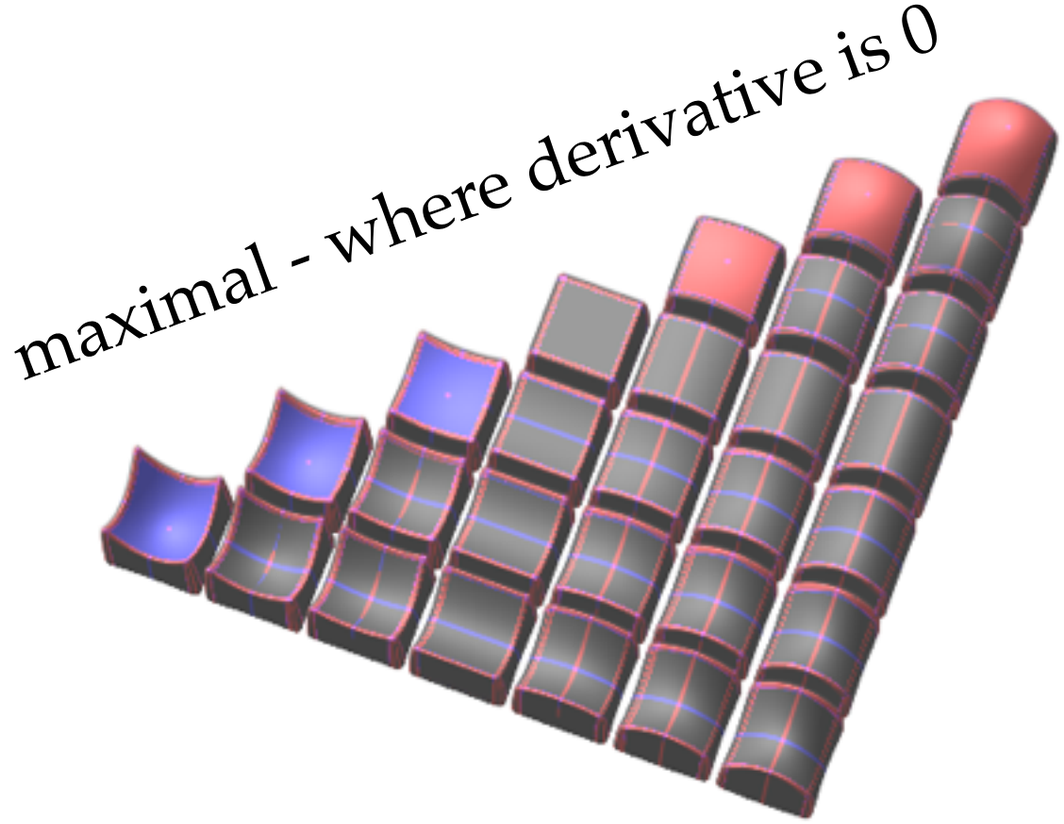
- Language restrictions make it difficult to implement new ideas
- Curvature and Crest Lines are related visualization properties

Motivation

- Language restrictions make it difficult to implement new ideas
- Curvature and Crest Lines are related visualization properties



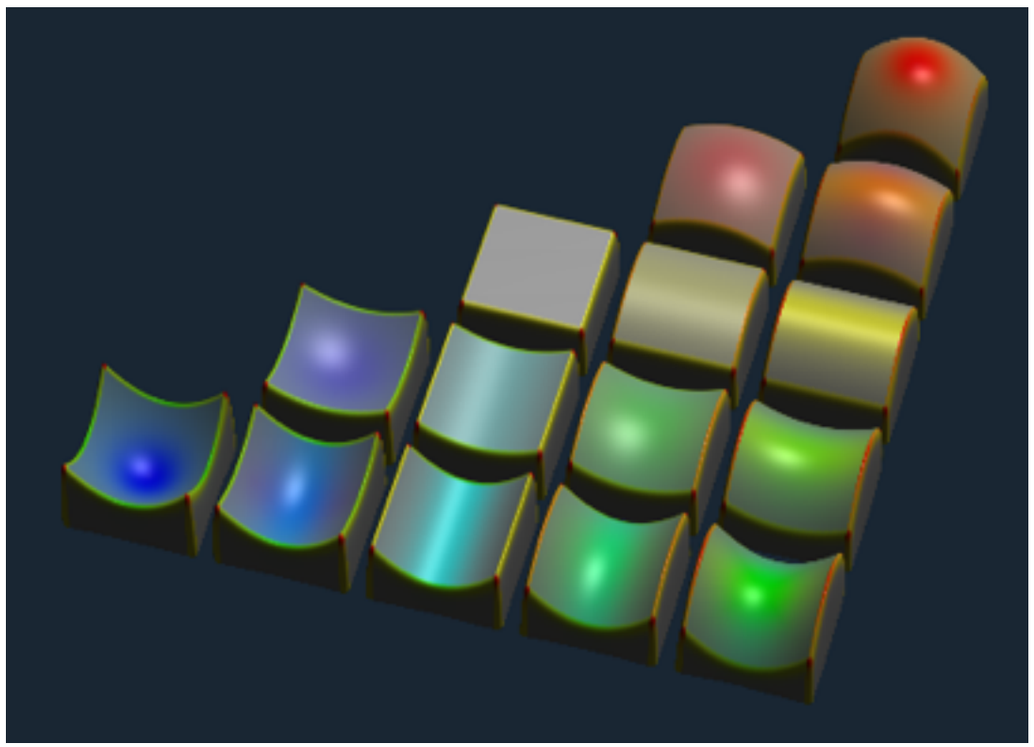
Curvature



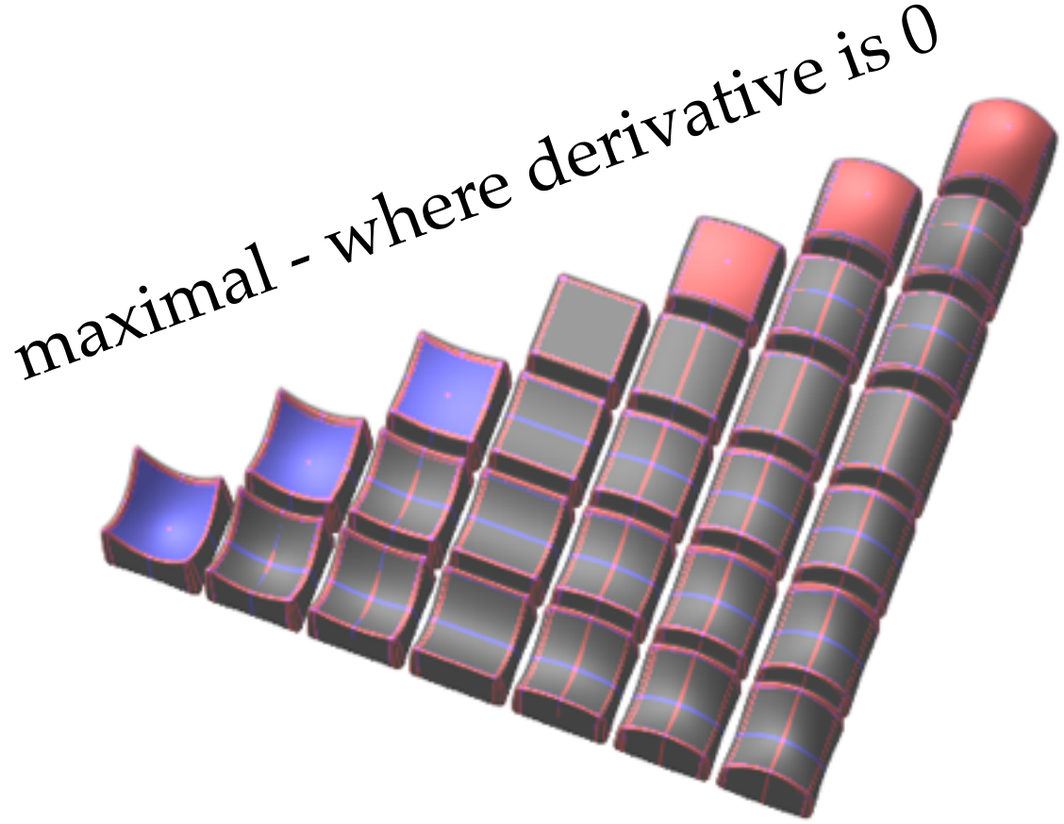
Crest Lines are places where the surface curvature is maximal along the **curvature** direction.

Motivation

- Language restrictions make it difficult to implement new ideas
- Curvature and Crest Lines are related visualization properties
- Should be able to build code



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...
  }
...
}
```

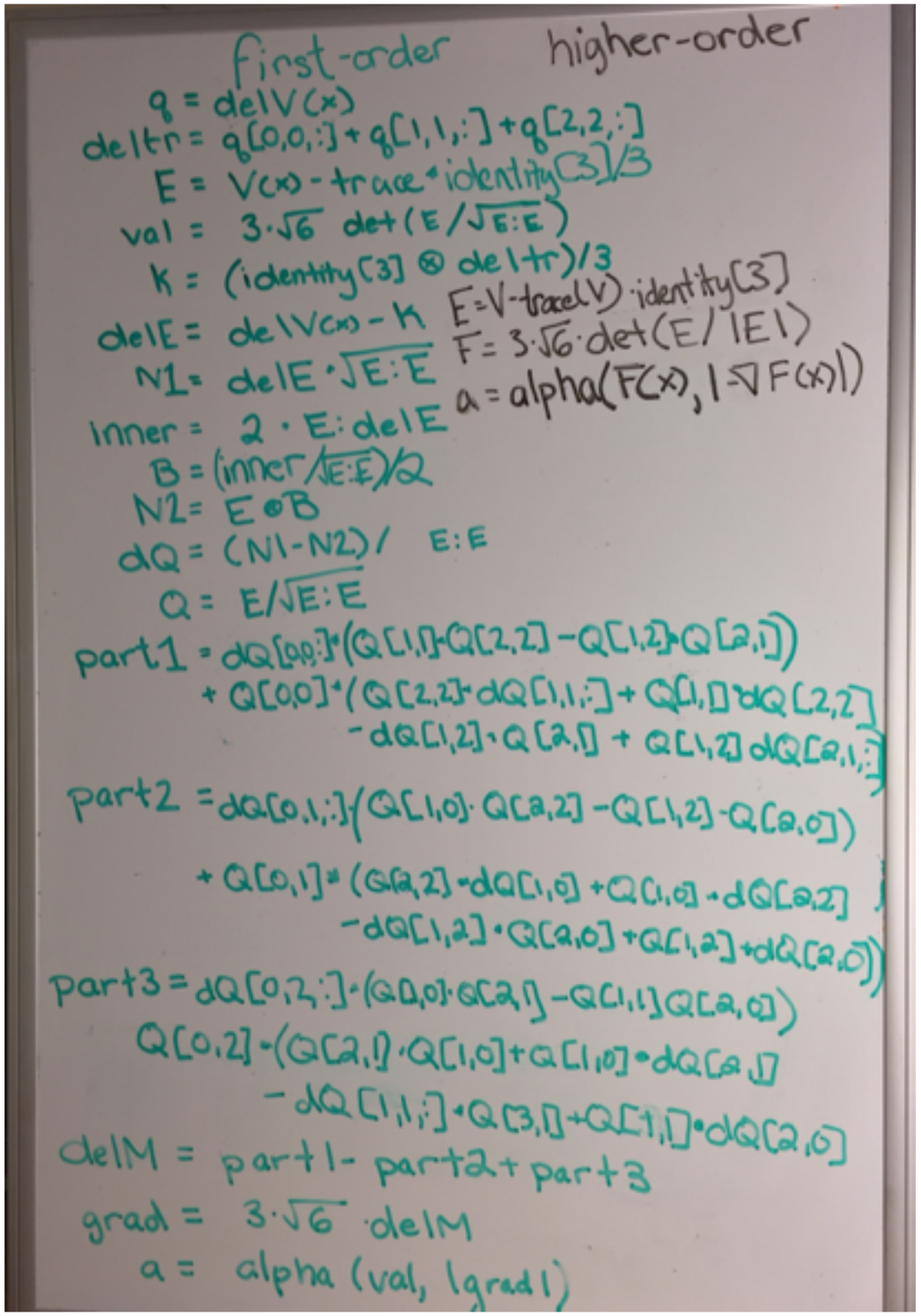
Directly add line

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..
  }
...
}
```

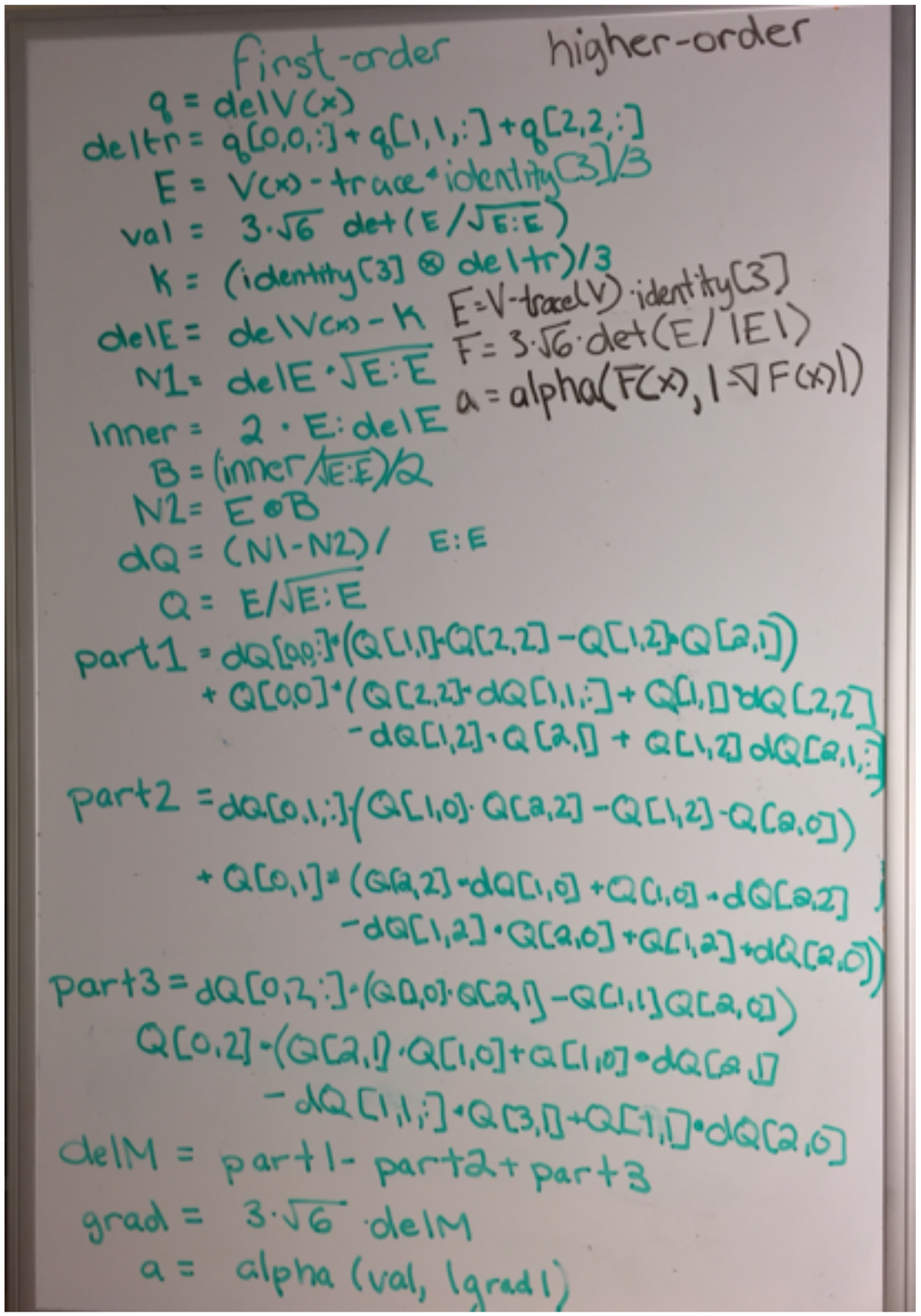
Not permitted

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.



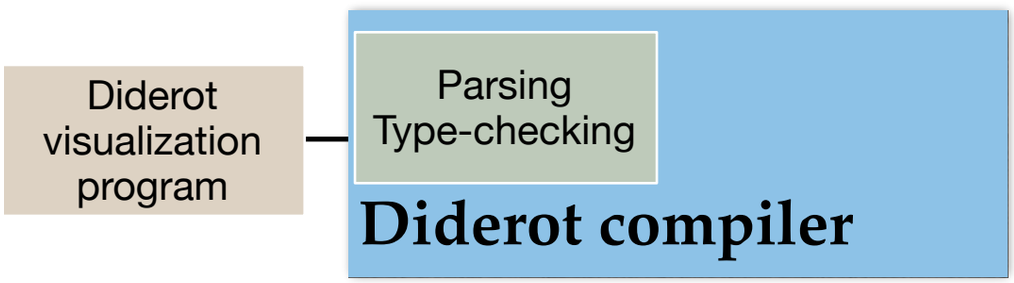
Implementing Expressivity

- Motivated our need for higher-level notation
- Overview of Diderot compiler
- Then focus on two stages

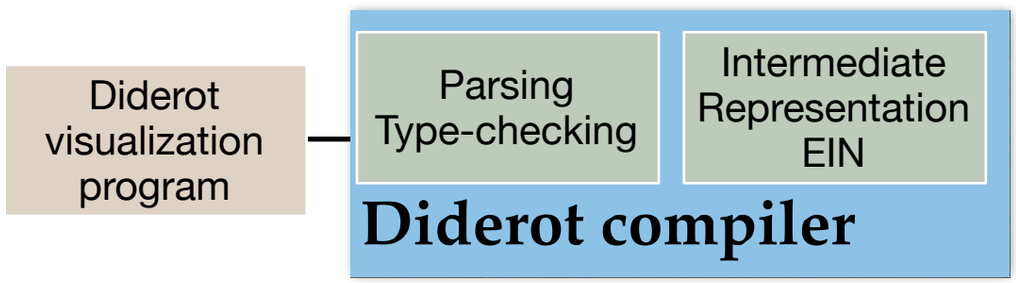
Implementing Expressivity

Diderot visualization program

Implementing Expressivity

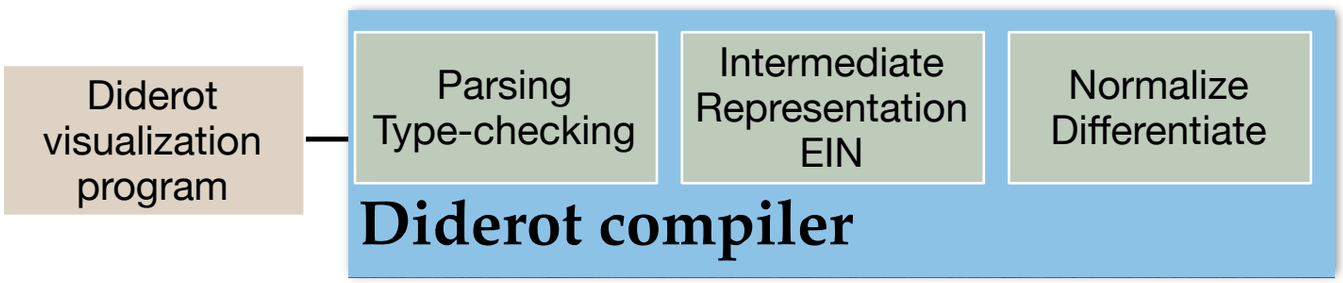


Implementing Expressivity

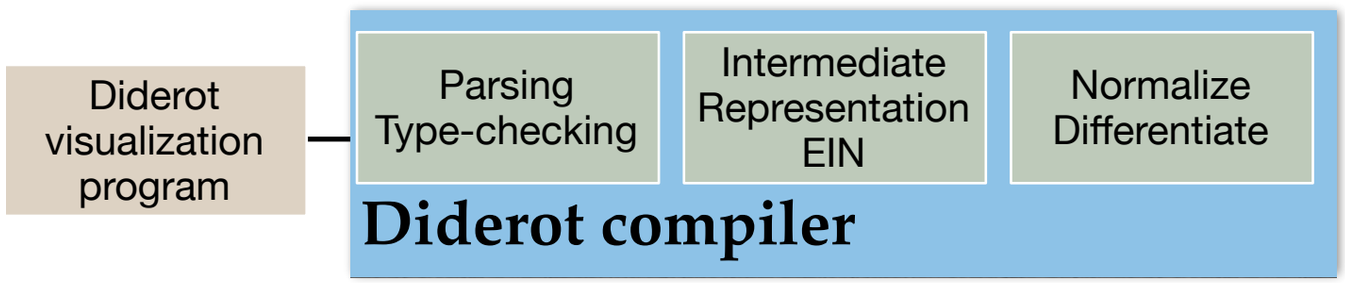


Phase that represents tensor math

Implementing Expressivity



Implementing Expressivity

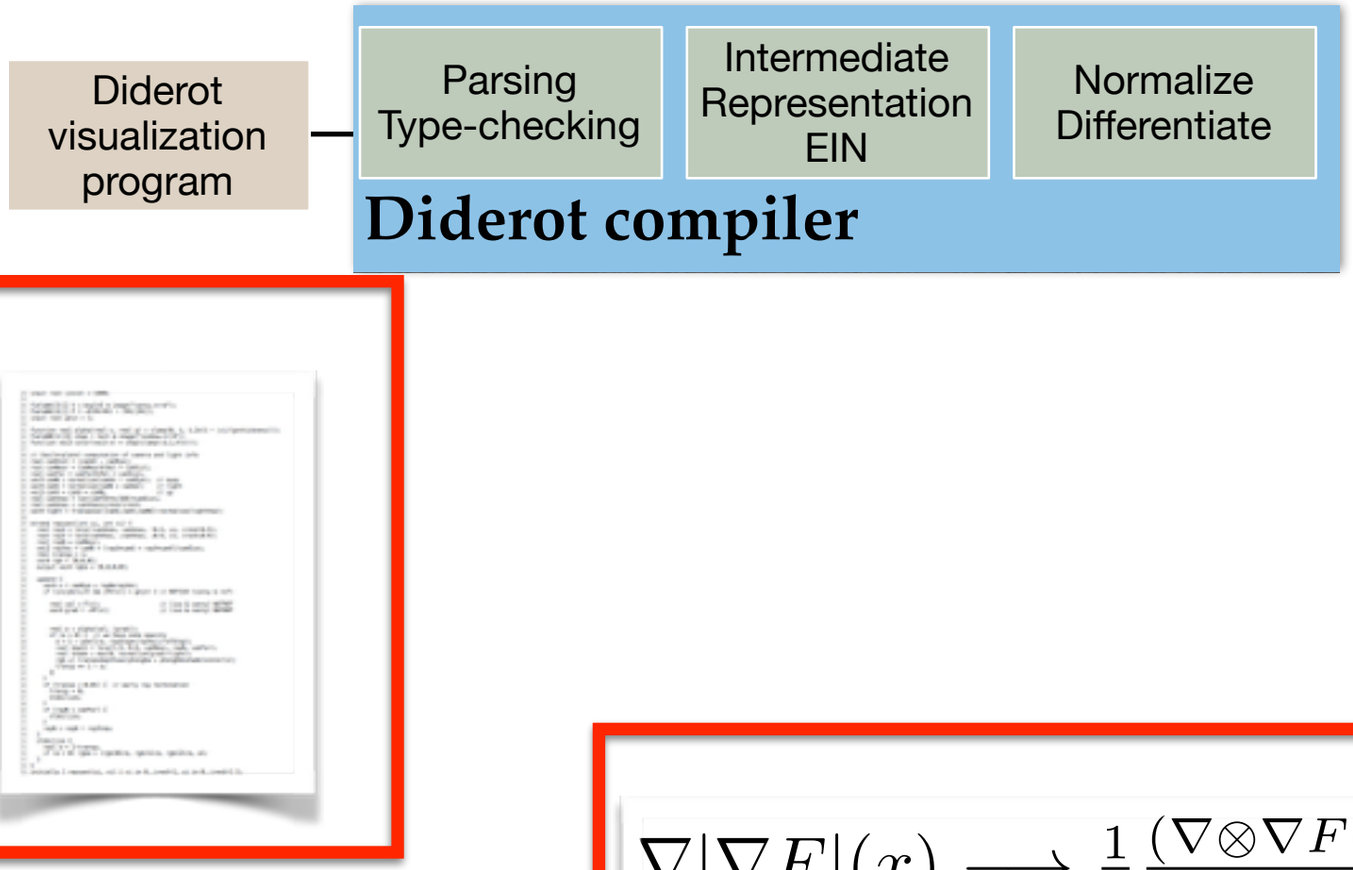


$$\nabla|\nabla F|(x)$$

Implementing Expressivity

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

Implementing Expressivity



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

Implementing Expressivity

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

Implementing Expressivity

Diderot visualization program

Parsing
Type-checking

Intermediate Representation
EIN

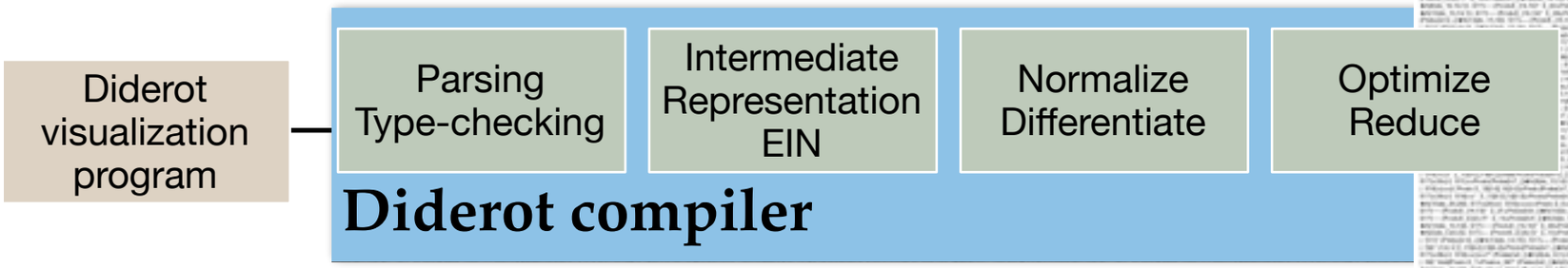
Normalize
Differentiate

Diderot compiler

Compilation Issue

Programs could not compile or took too long to compile

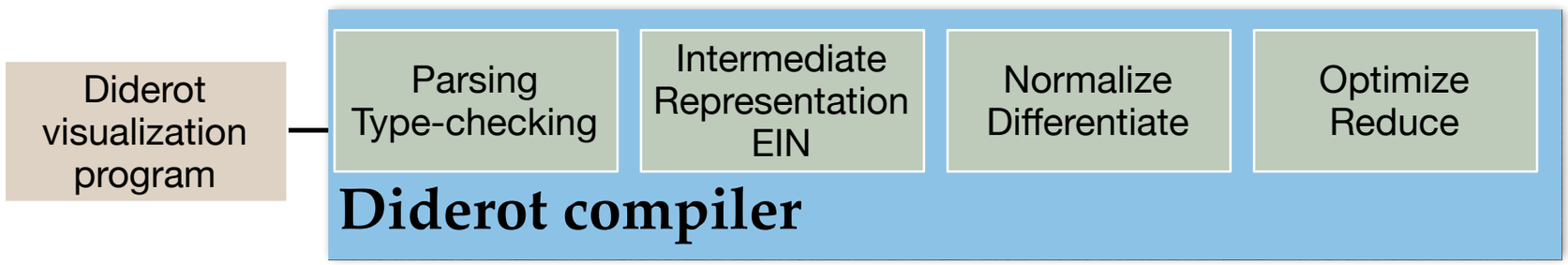
Implementing Expressivity



Compilation Issue

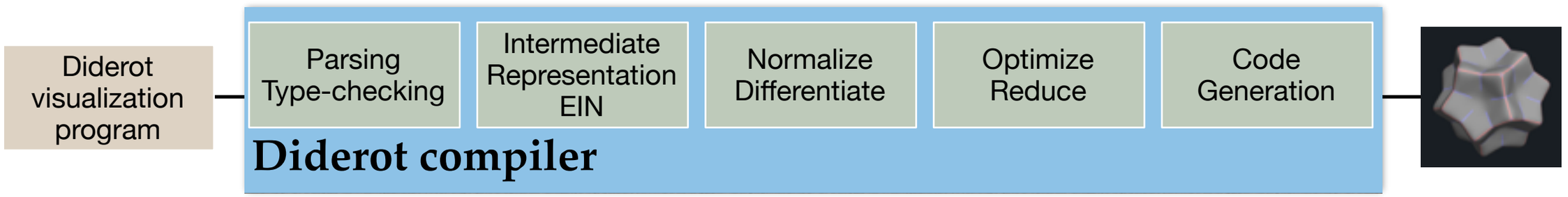
Programs could not compile or took too long to compile

Implementing Expressivity

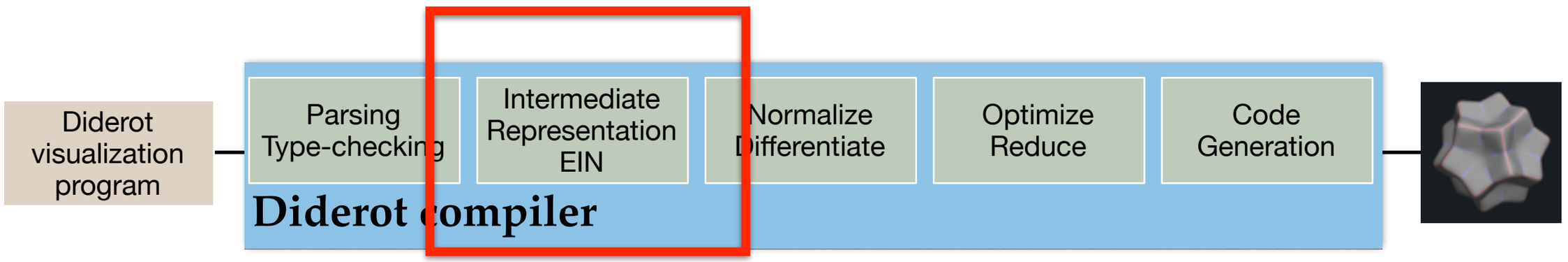


Phase that enables effective compilation of tensor math

Implementing Expressivity



EIN Intermediate Representation



Phase that represents tensor math

EIN Intermediate Representation

- EIN represents tensors and fields

EIN Intermediate Representation

- EIN represents tensors and fields

$$M_{ij} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

EIN Term Matrix M

EIN Intermediate Representation

- EIN represents tensors and fields

$$M_{\boxed{ij}} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

EIN Term Matrix M

EIN Intermediate Representation

- EIN represents tensors and fields

$$M_{\text{ij}} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

EIN Term — Matrix M

- EIN represents operators on tensor and fields

$$M_{\text{ji}} = \begin{bmatrix} M_{11} & M_{21} \\ M_{12} & M_{22} \end{bmatrix}$$

Switch row and column

$$M_{11} = M_{11}$$

Single entry

$$\sum M_{\text{ii}} = M_{11} + M_{22}$$

Add diagonal components

$$M_{\text{i1}} = \begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix}$$

Select a column

EIN Intermediate Representation

- EIN represents tensors and fields

$$M_{\text{ij}} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

EIN Term — Matrix M

- EIN represents operators on tensor and fields

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

$M_{\text{11}} = M_{11}$ Single entry

$\sum M_{\text{ii}} = M_{11} + M_{22}$ Add diagonal components

$M_{\text{i1}} = \begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix}$ Select a column

EIN Intermediate Representation

- EIN represents tensors and fields

$$M_{\text{ij}} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

EIN Term — Matrix M

- EIN represents operators on tensor and fields

$$M_{\text{ji}} = \begin{bmatrix} M_{11} & M_{21} \\ M_{12} & M_{22} \end{bmatrix}$$

Switch row and column

$$M_{11} = M_{11}$$

Single entry

$$\sum M_{\text{ii}} = M_{11} + M_{22}$$

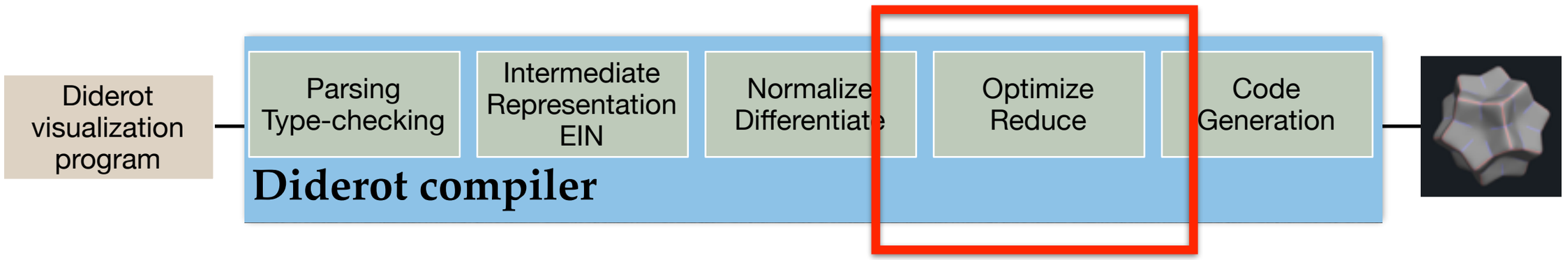
Add diagonal components

$$M_{\text{i1}} = \begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix}$$

Select a column

An EIN term can concisely represent tensor math

Implementing Expressivity



Phase that enables effective compilation of tensor math

Compilation Issue

Phase that enables effective compilation of tensor math

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)
- maintain mathematical integrity

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)
- maintain mathematical integrity

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)
- maintain mathematical integrity

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

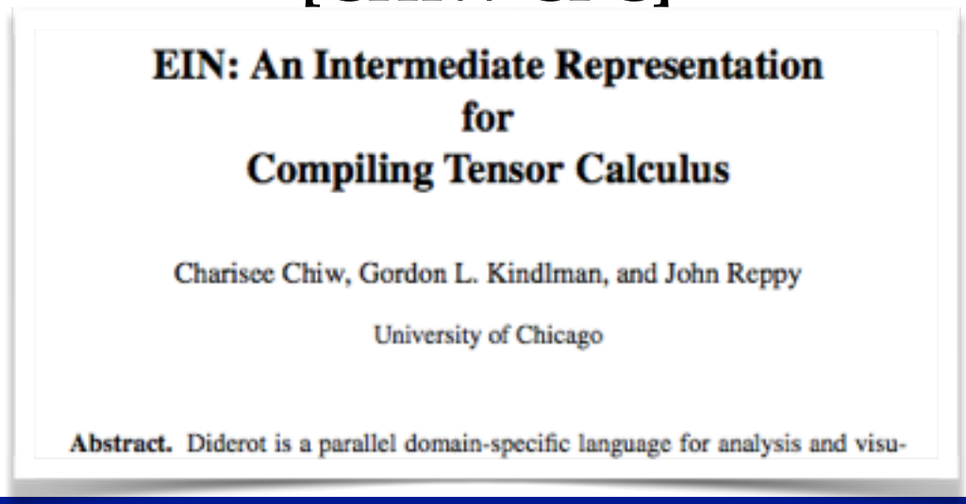
- enable traditional compiler approaches (cse)
- maintain mathematical integrity

Terms can be syntactically different even when they

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

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-

Write programs with higher-order operations

- Diderot is designed to represent general tensor math

Write programs with higher-order operations

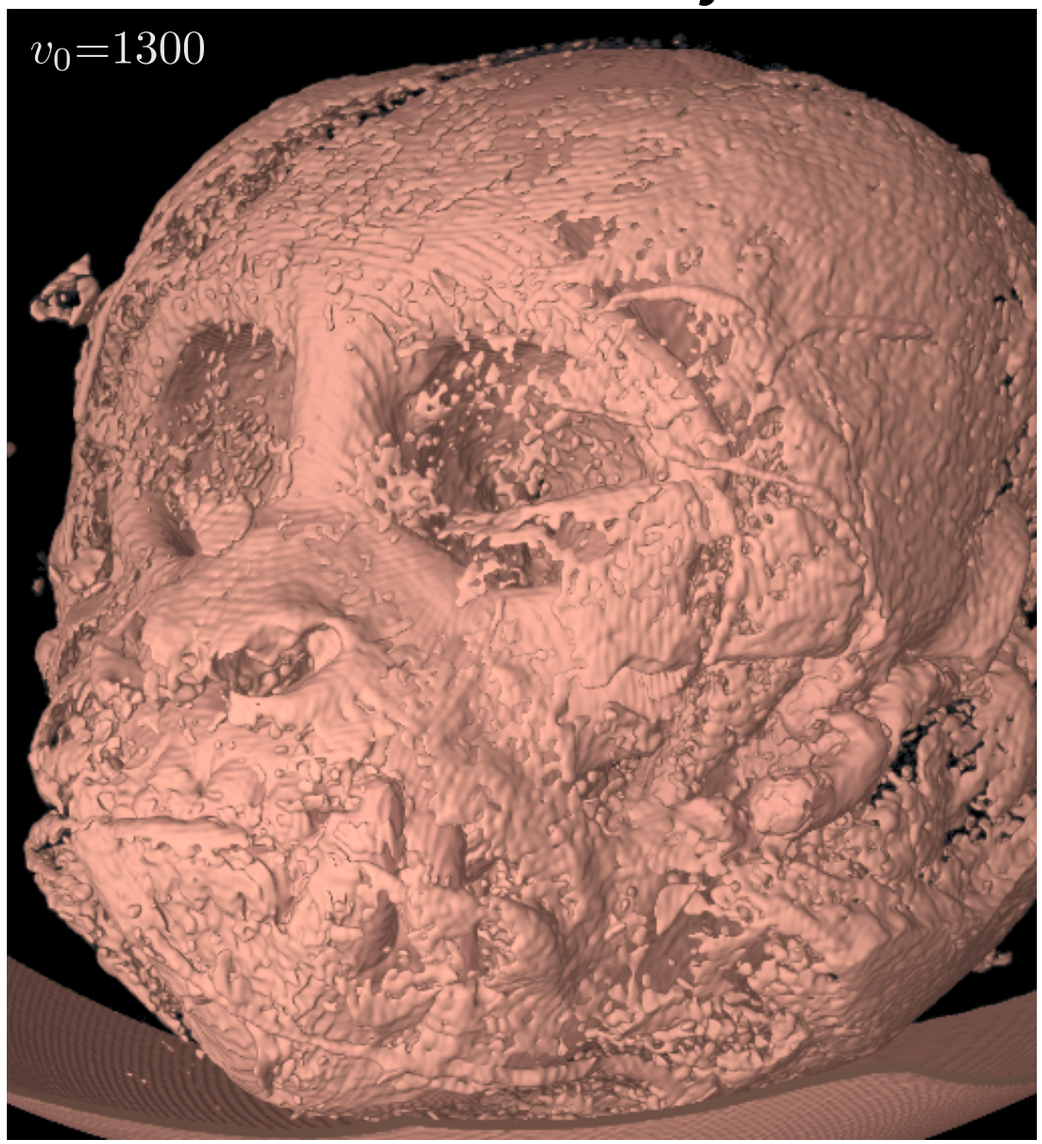
- Diderot is designed to represent general tensor math
- We rely on the expressivity of the language to support the implementation of visualization ideas.

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

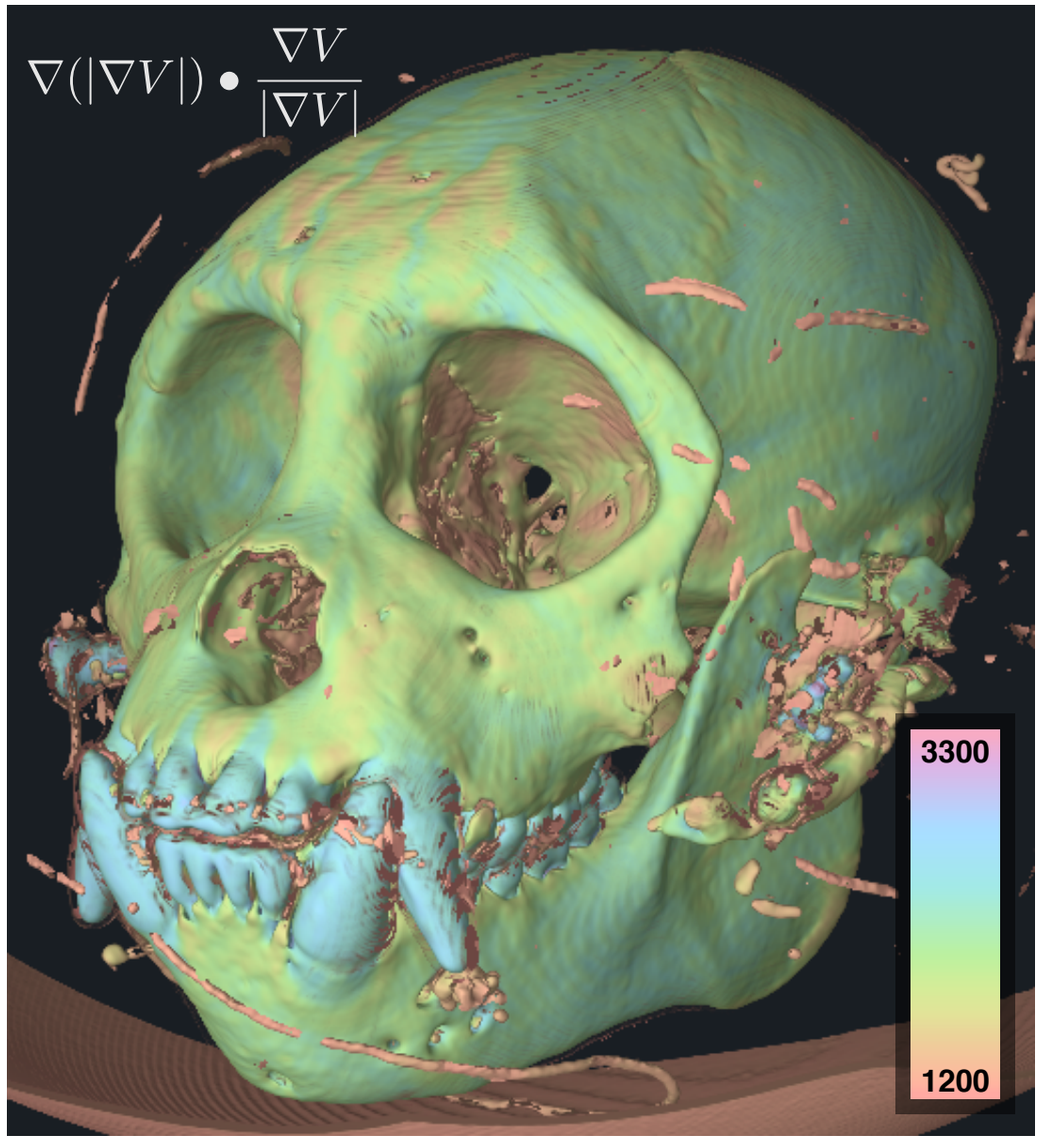
Results: Sophisticated Vis

Previously:



Iso-Surface

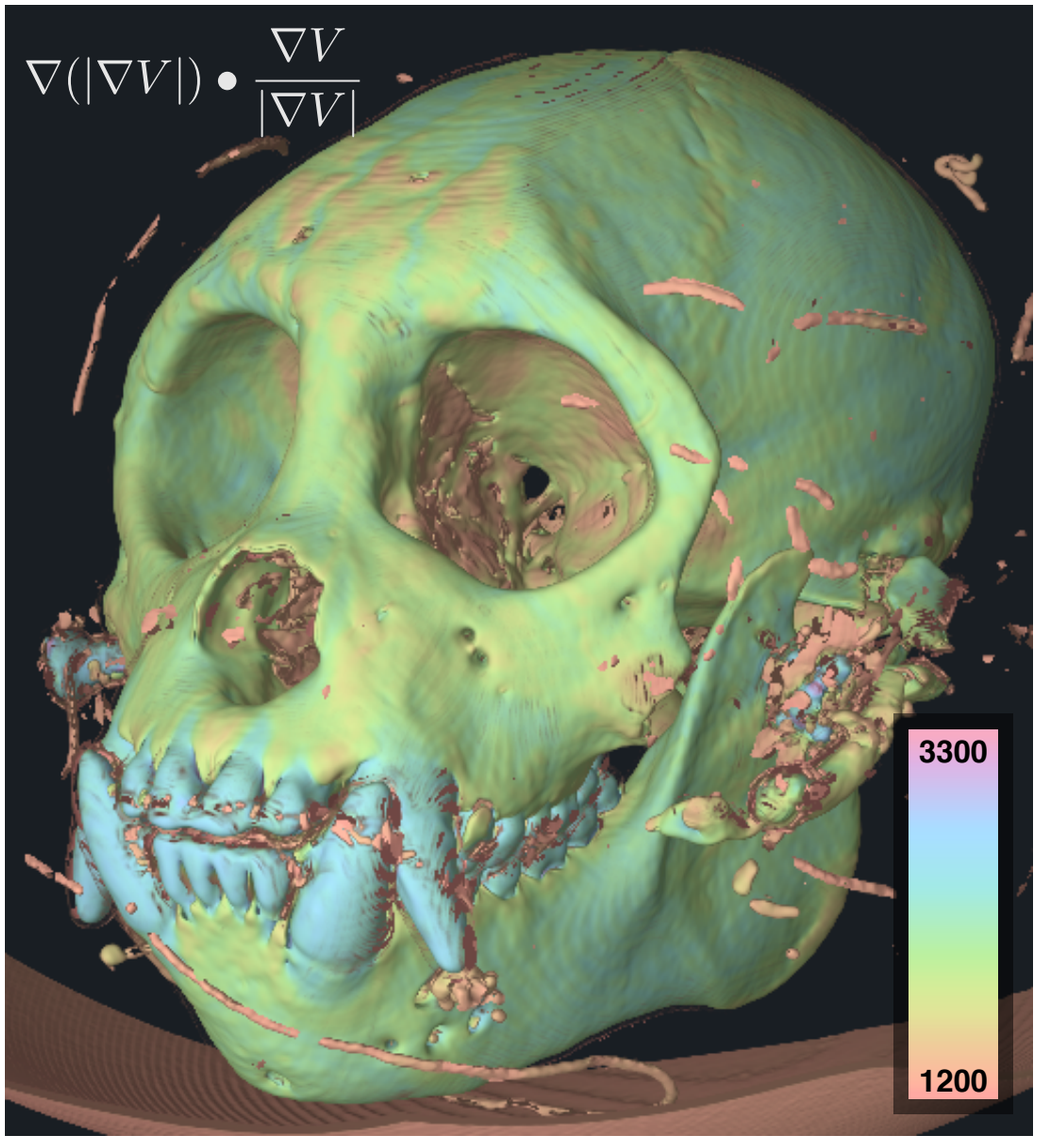
Now:



Canny Edges

Data provided by Callum Ross (U of Chicago)

Results: Sophisticated Vis

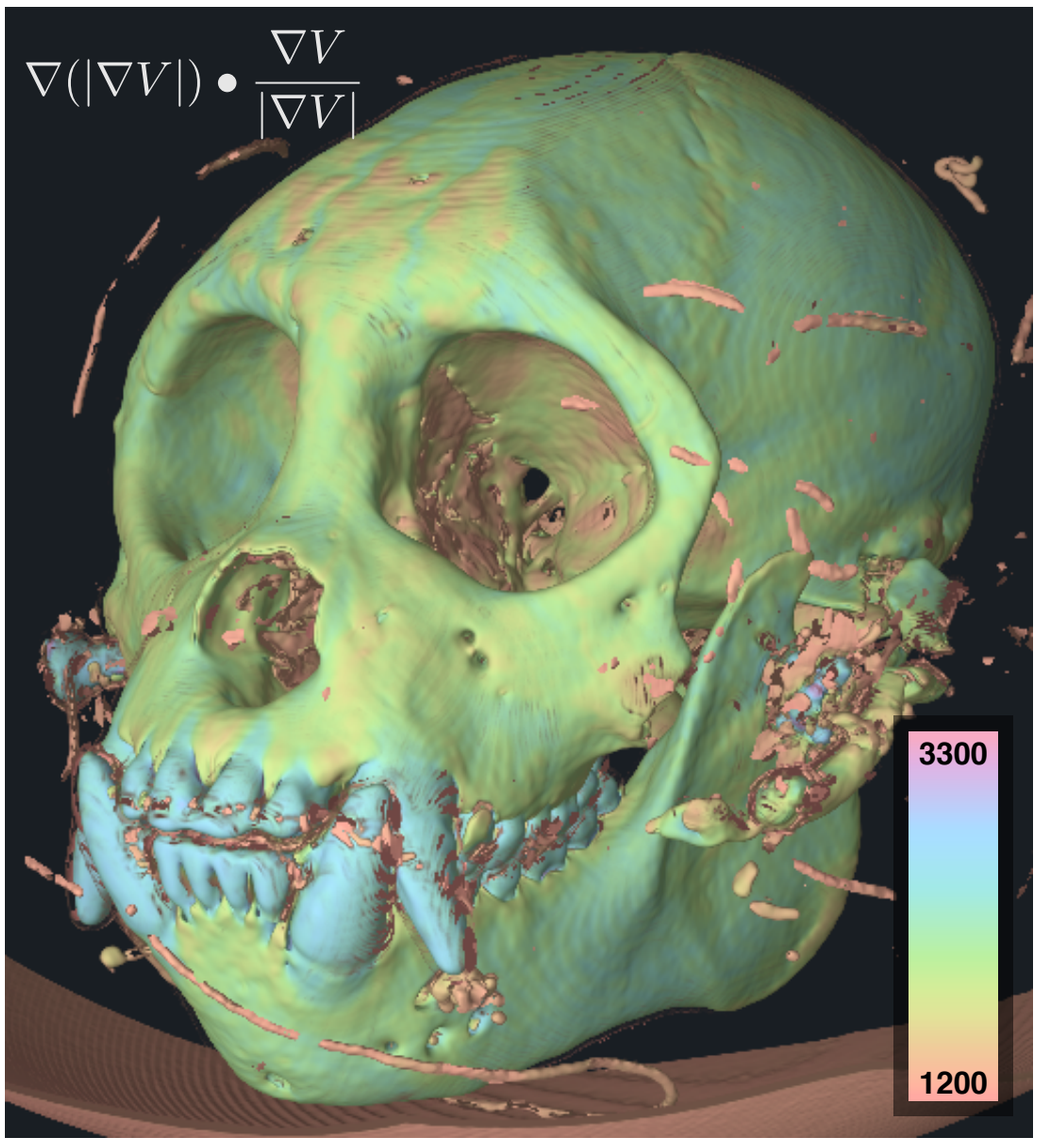


Canny Edges

Results: Sophisticated Vis

Inline high level syntax into Diderot code

```
−∇(|∇F|) • ∇F/|∇F|;
```



Canny Edges

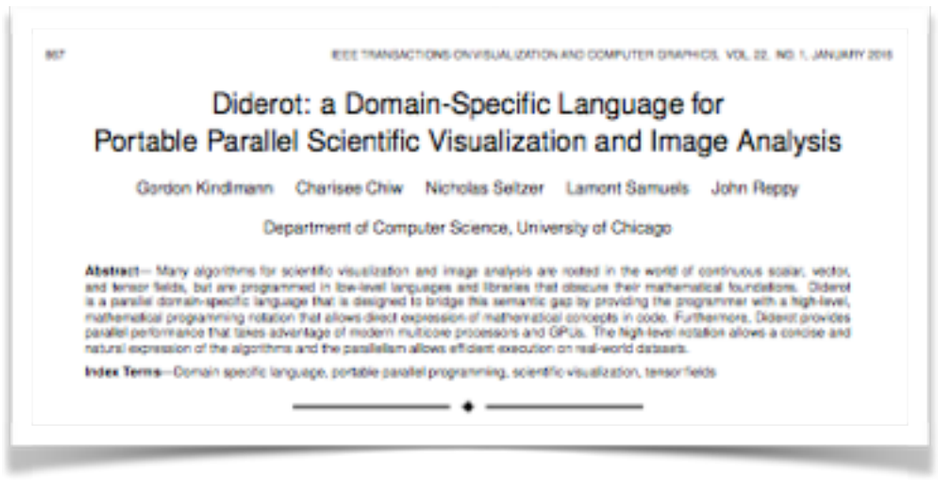
Results: Sophisticated Vis

Inline high level syntax into Diderot code

```
−∇(|∇F|) • ∇F/|∇F|;
```

Users focused on visualization

[KindImann VIS]



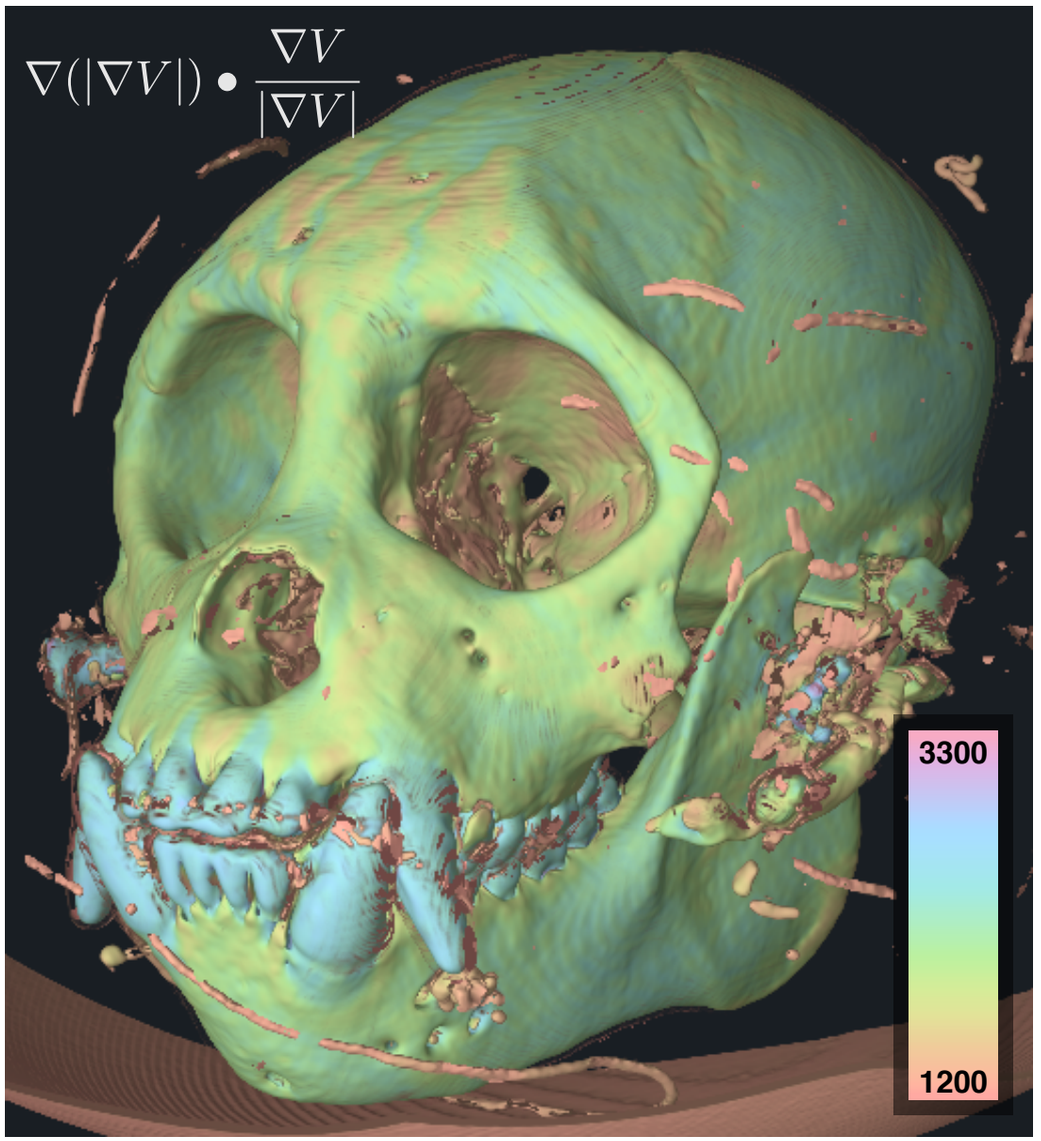
Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis

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

Domain Specific Languages and Tools

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

Firedrake

Finite element method

SPIRAL

Digital Signal Processing

Chemistry

Microsoft SQL Server

Databases

Cryptography

HTML

Web design

Unified Form Language (UFL)

Solve PDEs

Diderot

Diderot Image analysis

Domain Specific Languages and Tools

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

Firedrake

Big Matrix computations

TCE

SPIRAL

Digital Signal Processing
Transformation

Microsoft SQL Server

Databases

Cryptography: assurance and security

HTML

Web design

Unified Form Language (UFL)

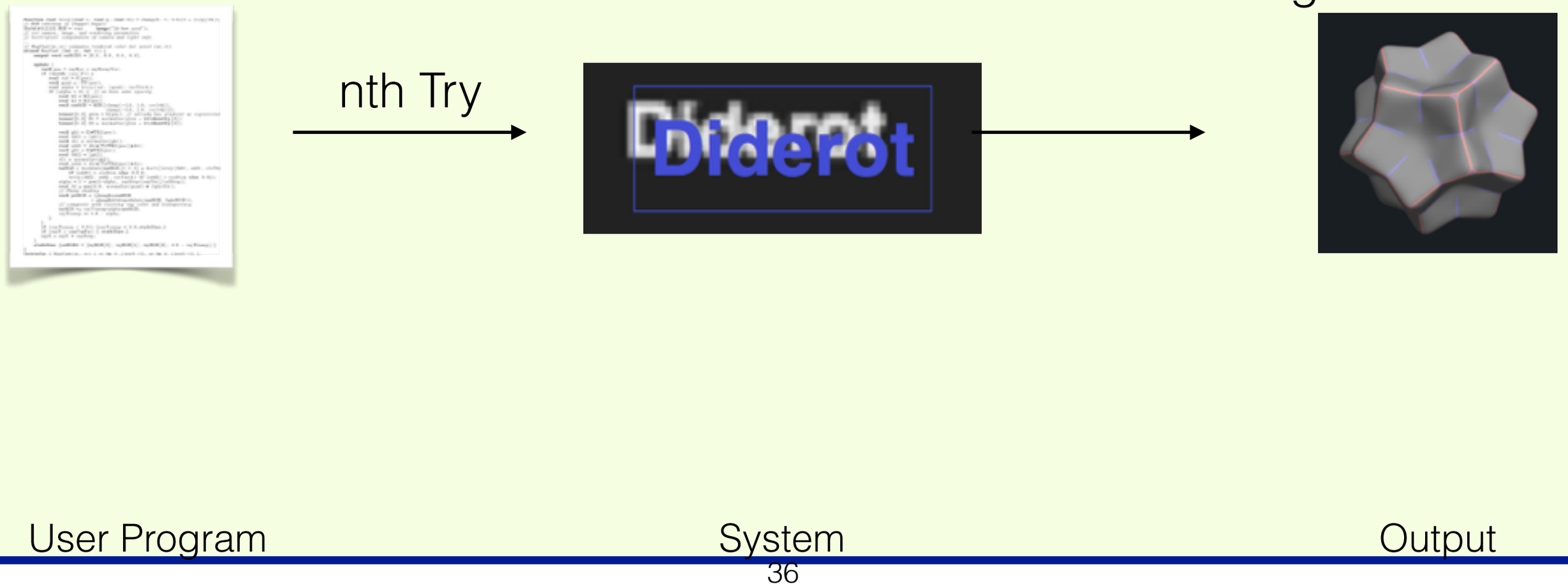
Hands over to form compiler

Diderot

Vis and general tensor math

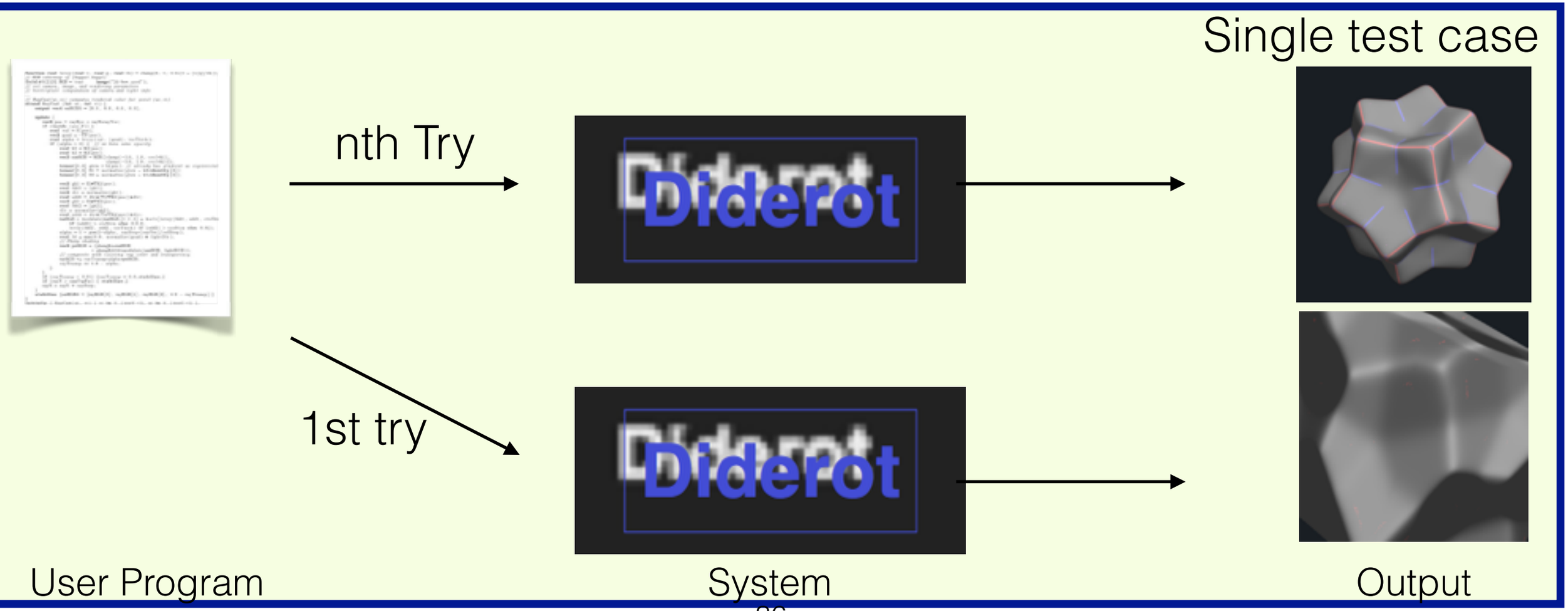
Return to Crest Line example

We need to test our language implementation



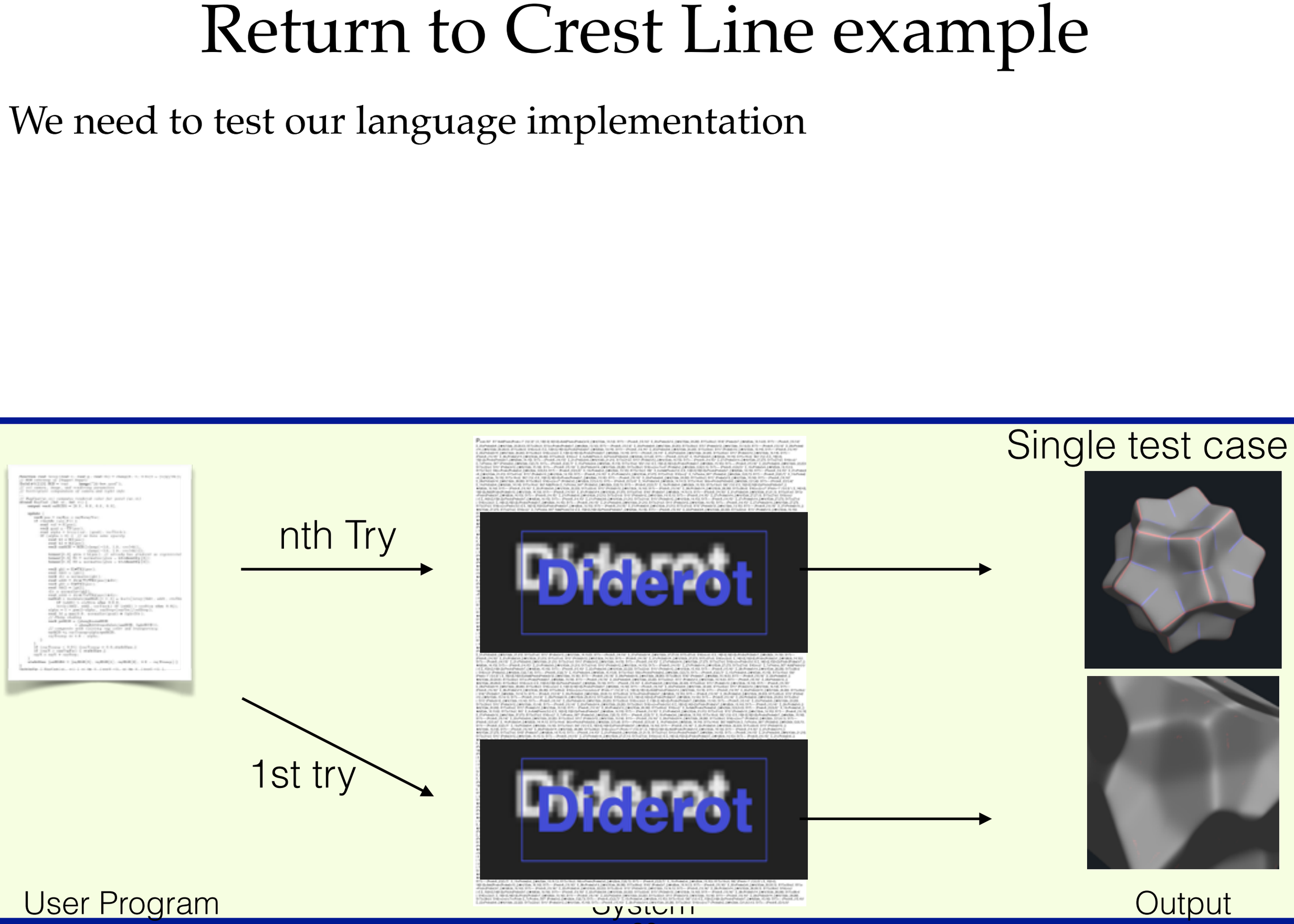
Return to Crest Line example

We need to test our language implementation



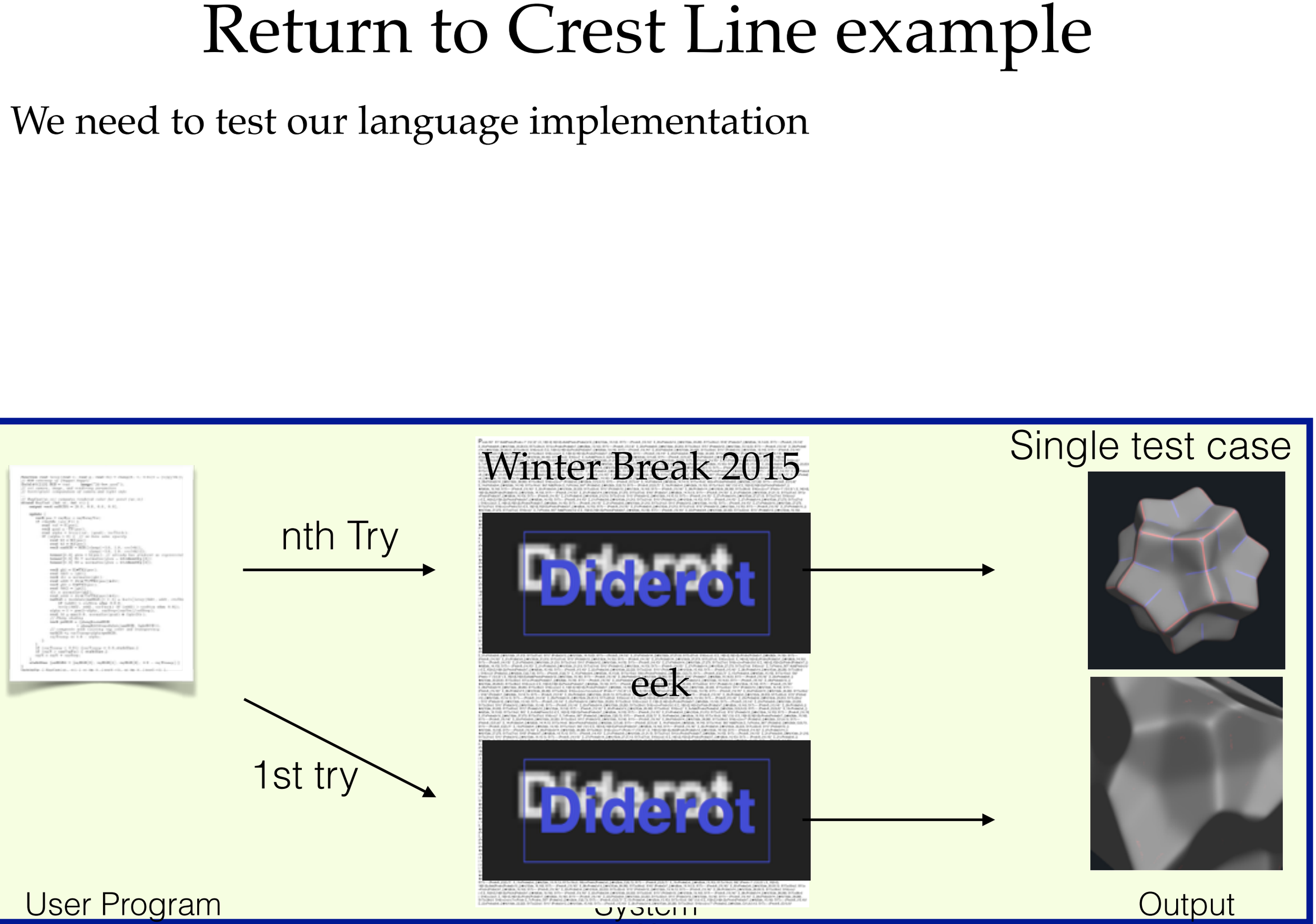
Return to Crest Line example

We need to test our language implementation



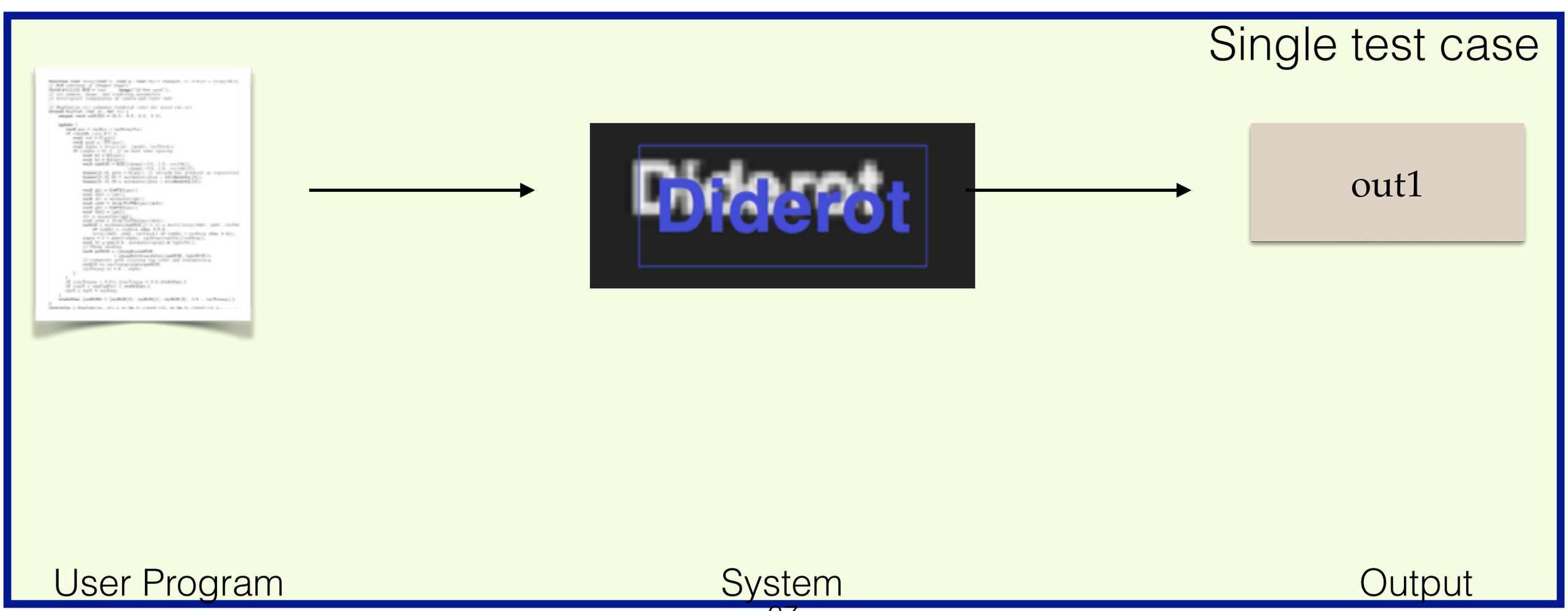
Return to Crest Line example

We need to test our language implementation



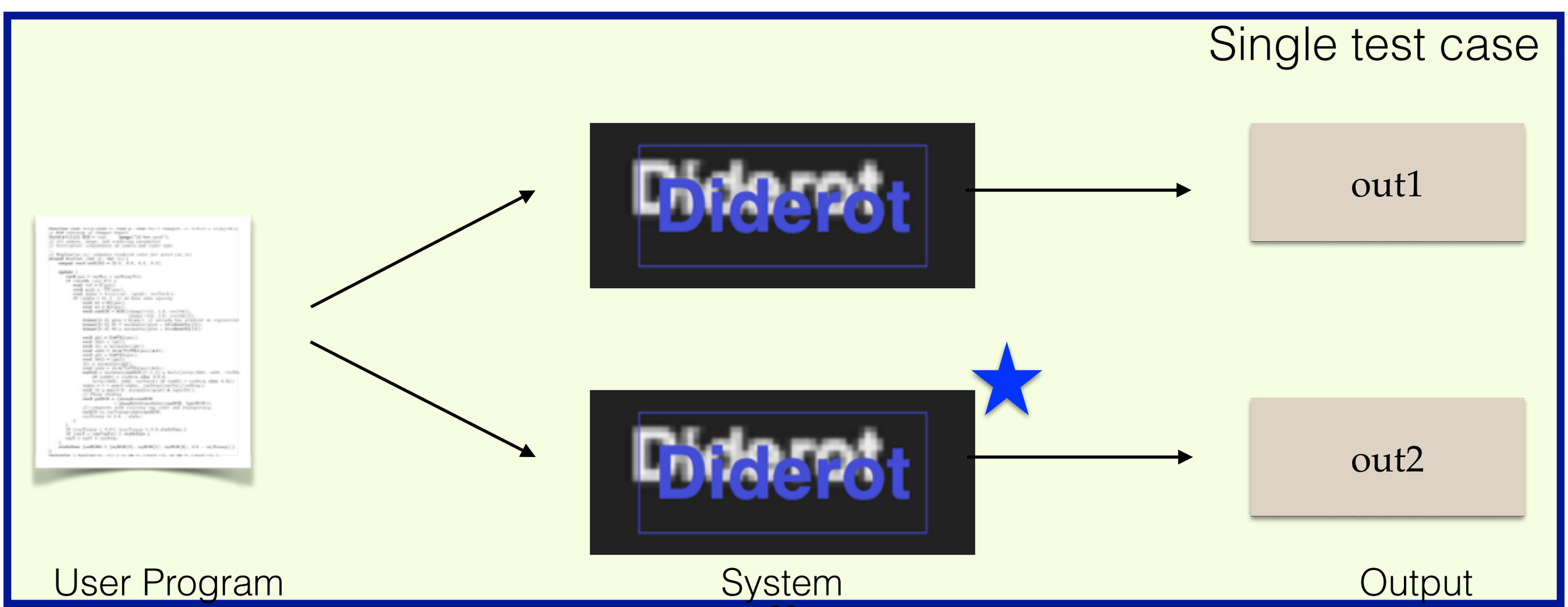
Software testing

- **Different Optimization Levels** [Chen]



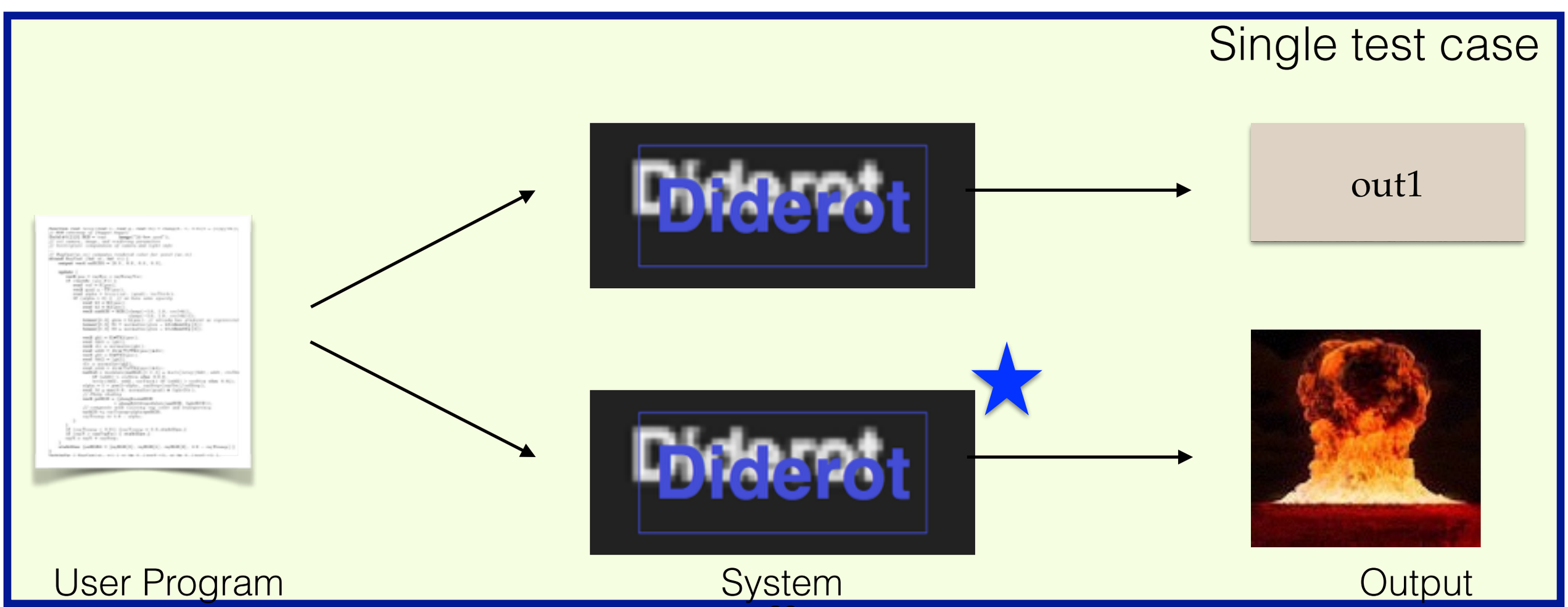
Software testing

- **Different Optimization Levels** [Chen]



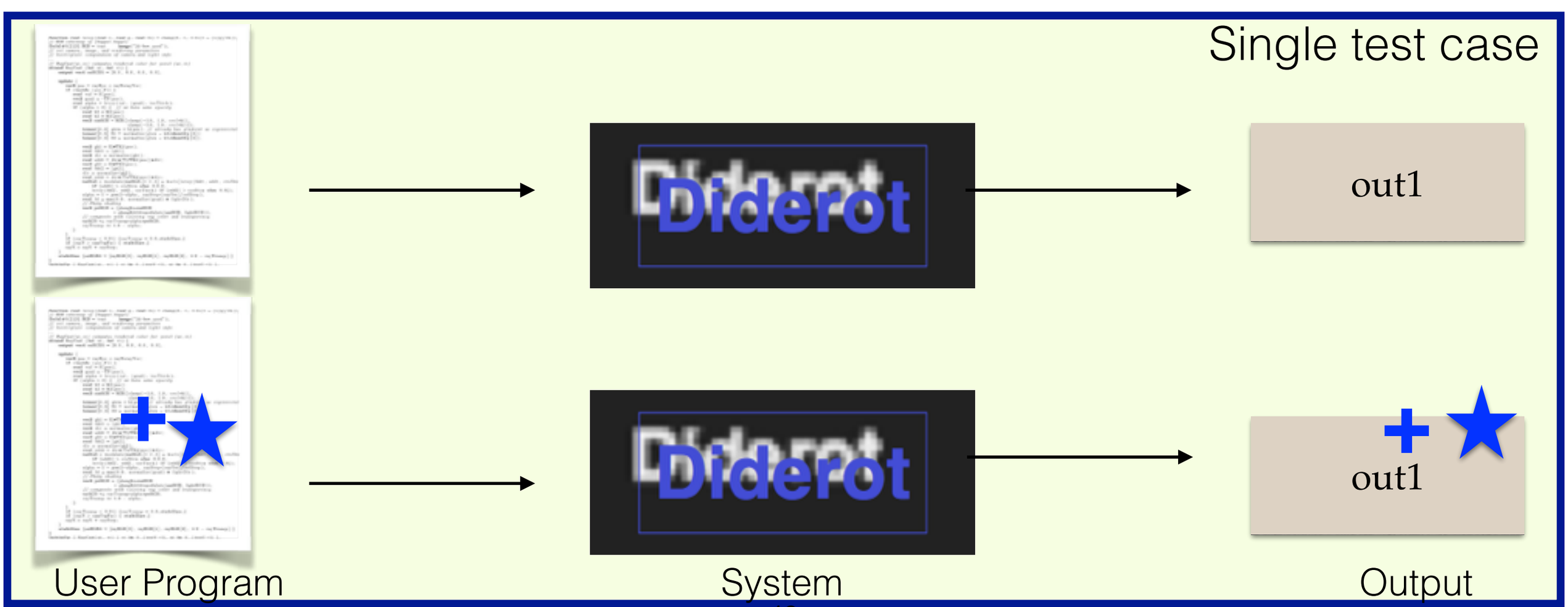
Software testing

- **Different Optimization Levels** [Chen]



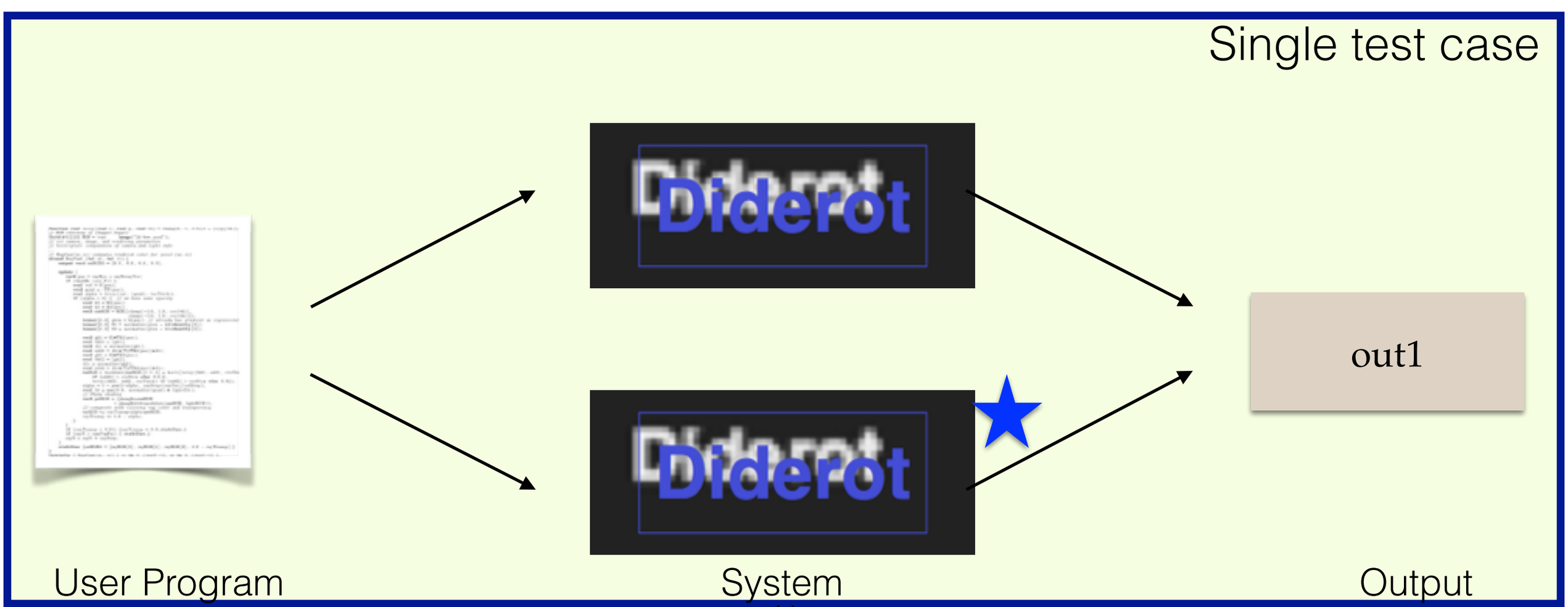
Software testing

- Different Optimization Levels [Chen]
- **Metamorphic Testing** [Donaldson et.al'16, Tao et.al.' 10]



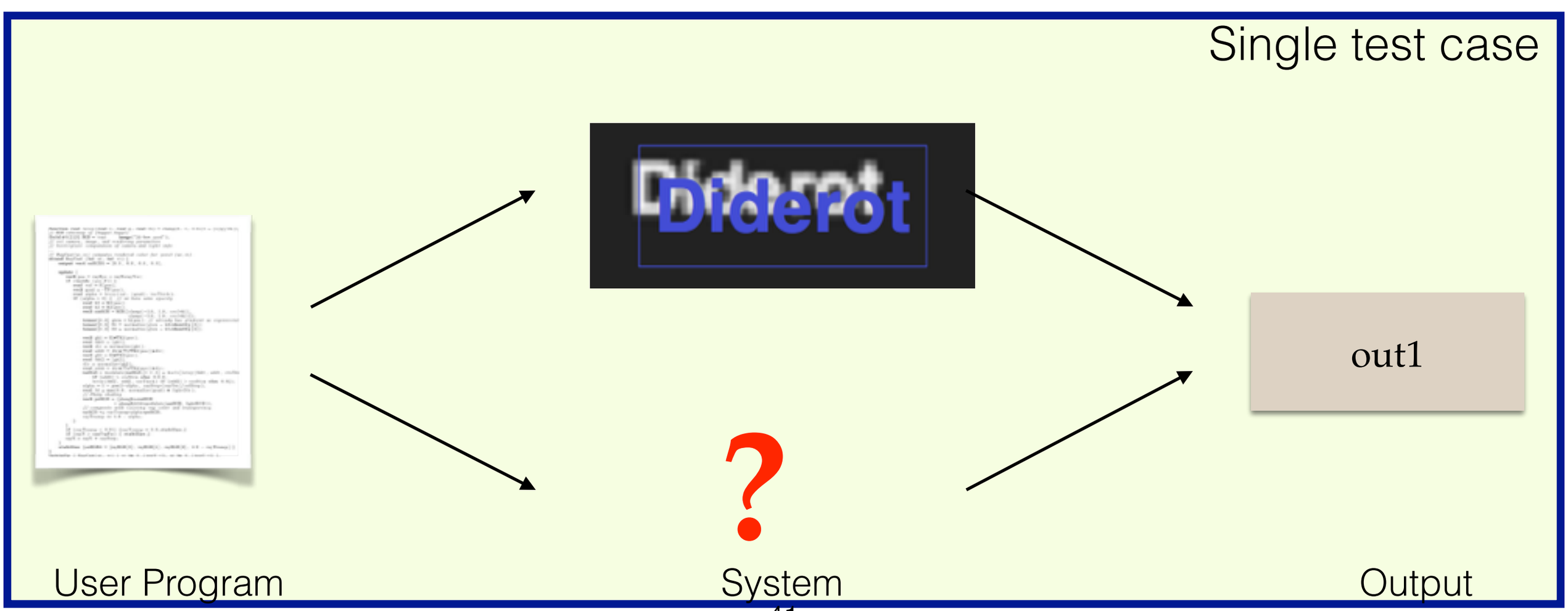
Software testing

- Different Optimization Levels [Chen]
- Metamorphic Testing [Donaldson et.al'16, Tao et.al.' 10]
- **Differential Testing** [Yang et.al'11, Lindig'05, McKeeman'98]



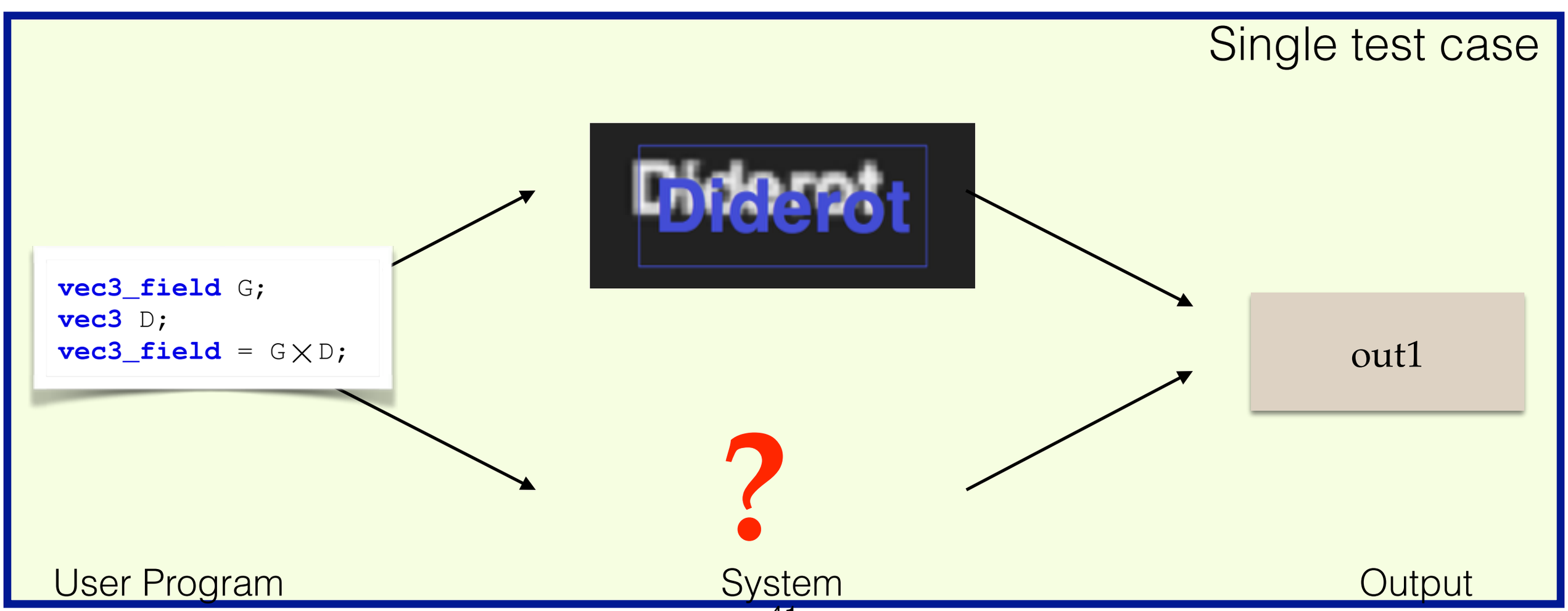
Software testing

- Different Optimization Levels [Chen]
- Metamorphic Testing [Donaldson et.al'16, Tao et.al.' 10]
- **Differential Testing** [Yang et.al'11, Lindig'05, McKeeman'98]



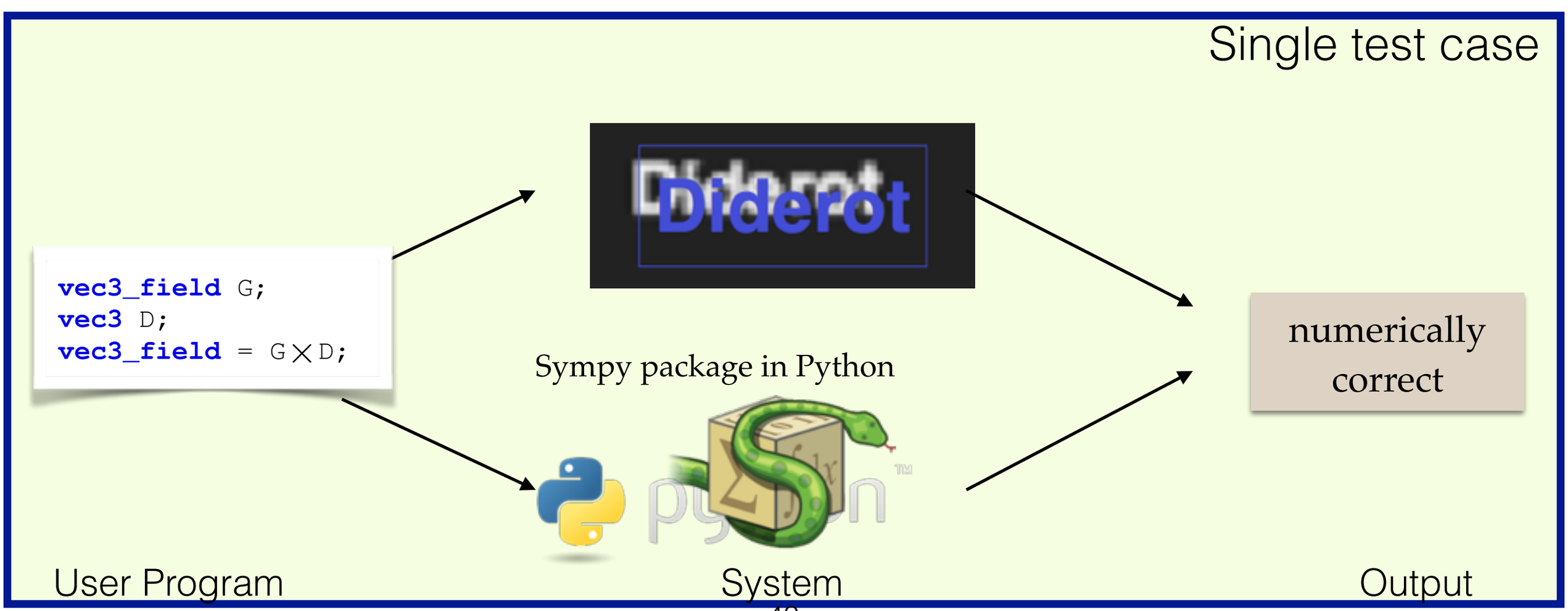
Software testing

- Different Optimization Levels [Chen]
- Metamorphic Testing [Donaldson et.al'16, Tao et.al.' 10]
- **Differential Testing** [Yang et.al'11, Lindig'05, McKeeman'98]



Software testing

- Different Optimization Levels [Chen]
- Metamorphic Testing [Donaldson et.al'16, Tao et.al.' 10]
- Differential Testing [Yang et.al'11, Lindig'05, McKeeman'98]
- **DATm: Diderot's Automated Testing model** [Chiw '17]



DATm: Diderot's Automated Testing model

- Manual construction of test infeasible

Single test case

```
vec3_field G;
vec3 D;
vec3_field = G×D;
```

Diderot

Sympy package in Python

numerically correct

User Program | System | Output

DATm: Diderot's Automated Testing model

- Manual construction of test infeasible
- DATm automatically creates hundred of thousands of tests

Single test case

```
vec3_field G;
vec3 D;
vec3_field = G×D;
```

Diderot

Sympy package in Python

numerically correct

User Program | System | Output

DATm: Diderot's Automated Testing model

- Manual construction of test infeasible
- DATm automatically creates hundred of thousands of tests
- Successful at finding bugs

Single test case

```
vec3_field G;
vec3 D;
vec3_field = G×D;
```

Diderot

Sympy package in Python

numerically correct

User Program | System | Output

DATm: Diderot's Automated Testing model

Generate test cases

Check operator "x" with different argument types

- [ ] OK
- [ ] Type error
- [ ] Compiler error
- [ ] Numerical error

DATm: Diderot's Automated Testing model

Generate test cases

Check operator "x" with different argument types

- [ ] OK
- [ ] Type error
- [ ] Compiler error
- [ ] Numerical error

```
vec2 A;
vec2 B;
vec2 out = A×B;
```

```
vec3 C;
vec3 D;
vec3 out = C×D;
```

```
vec2_field E;
vec2_field F;
vec2_field = E×F;
```

```
vec3_field G;
vec3_field H;
vec3_field = G×H;
```

```
vec2 A;
vec2_field F;
vec2_field = A×F;
```

```
vec3_field G;
vec3 D;
vec3_field = G×D;
```

```
vec2_field E;
vec2 B;
vec2_field = E×B;
```

```
vec3 C;
vec3_field H;
vec3_field = C×H;
```

DATm: Diderot's Automated Testing model

Generate test cases

```
vec2 A;
vec2 B;
vec2 out = A×B;
```

```
vec3 C;
vec3 D;
vec3 out = C×D;
```

```
vec2_field E;
vec2_field F;
vec2_field = E×F;
```

```
vec3_field G;
vec3_field H;
vec3_field = G×H;
```

```
vec2 A;
vec2_field F;
vec2_field = A×F;
```

```
vec3_field G;
vec3 D;
vec3_field = G×D;
```

Check operator "x" with different argument types

```
vec2_field E;
vec2 B;
vec2_field = E×B;
```

```
vec3 C;
vec3_field H;
vec3_field = C×H;
```

- [ ] OK
- [x] Type error
- [ ] Compiler error
- [ ] Numerical error

DATm: Diderot's Automated Testing model

Generate test cases

Check combination of operators

- [ ] OK
- [ ] Type error
- [ ] Compiler error
- [ ] Numerical error

DATm: Diderot's Automated Testing model

Generate test cases

Check combination of operators

- ☐ OK
- ☐ Type error
- ☐ Compiler error
- ☐ Numerical error

```
mat3_field F;
mat3_field G;
mat3_field out  modulate (F,G);
```

```
mat3_field F;
mat3_field G;
real_field out = trace ( modulate (F,G));
```

```
mat3_field F;
mat3_field G;
real_field out = trace (-modulate (F,G));
```

DATm: Diderot's Automated Testing model

Generate test cases

Check combination of operators

- [ ] OK
- [ ] Type error
- [x] Compiler error
- [ ] Numerical error

```
mat3_field F;
mat3_field G;
mat3_field out  modulate (F,G);
```

```
mat3_field F;
mat3_field G;
real_field out = trace ( modulate (F,G));
```

```
mat3_field F;
mat3_field G;
real_field out = trace (-modulate (F,G));
```

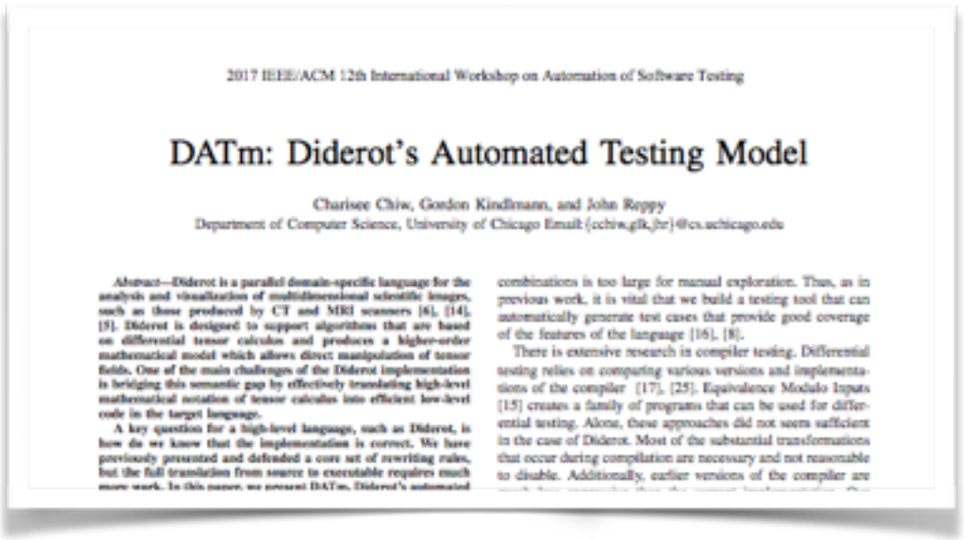
DATm: Diderot's Automated Testing model

Generate test cases


```
mat3_field F;
mat3_field G;
mat3_field out  modulate (F,G);
```

```
mat3_field F;
mat3_field G;
real_field out = trace ( modulate (F,G));
```

Check combination of operators



2017 IEEE/ACM 12th International Workshop on Automation of Software Testing

DATm: Diderot's Automated Testing Model

Charisee Chiw, Gordon Kindlmann, and John Reppy
Department of Computer Science, University of Chicago Email:{cchiw,glk,jhr}@cs.uchicago.edu

Abstract—Diderot is a parallel domain-specific language for the analysis and visualization of multidimensional scientific images, such as those produced by CT and MRI scanners [6], [14], [5]. Diderot is designed to support algorithms that are based on differential tensor calculus and produces a higher-order mathematical model which allows direct manipulation of tensor fields. One of the main challenges of the Diderot implementation is bridging this semantic gap by effectively translating high-level mathematical notation of tensor calculus into efficient low-level code in the target language.

A key question for a high-level language, such as Diderot, is how do we know that the implementation is correct. We have previously presented and defended a core set of rewriting rules, but the full translation from source to executable requires much

combinations is too large for manual exploration. Thus, as in previous work, it is vital that we build a testing tool that can automatically generate test cases that provide good coverage of the features of the language [16], [8].

There is extensive research in compiler testing. Differential testing relies on comparing various versions and implementations of the compiler [17], [25]. Equivalence Modulo Inputs [15] creates a family of programs that can be used for differential testing. Alone, these approaches did not seem sufficient in the case of Diderot. Most of the substantial transformations that occur during compilation are necessary and not reasonable to disable. Additionally, earlier versions of the compiler are

```
mat3_field F;
mat3_field G;
real_field out = trace (-modulate (F,G));
```

[CHIW ICSE-AST]

What's next?

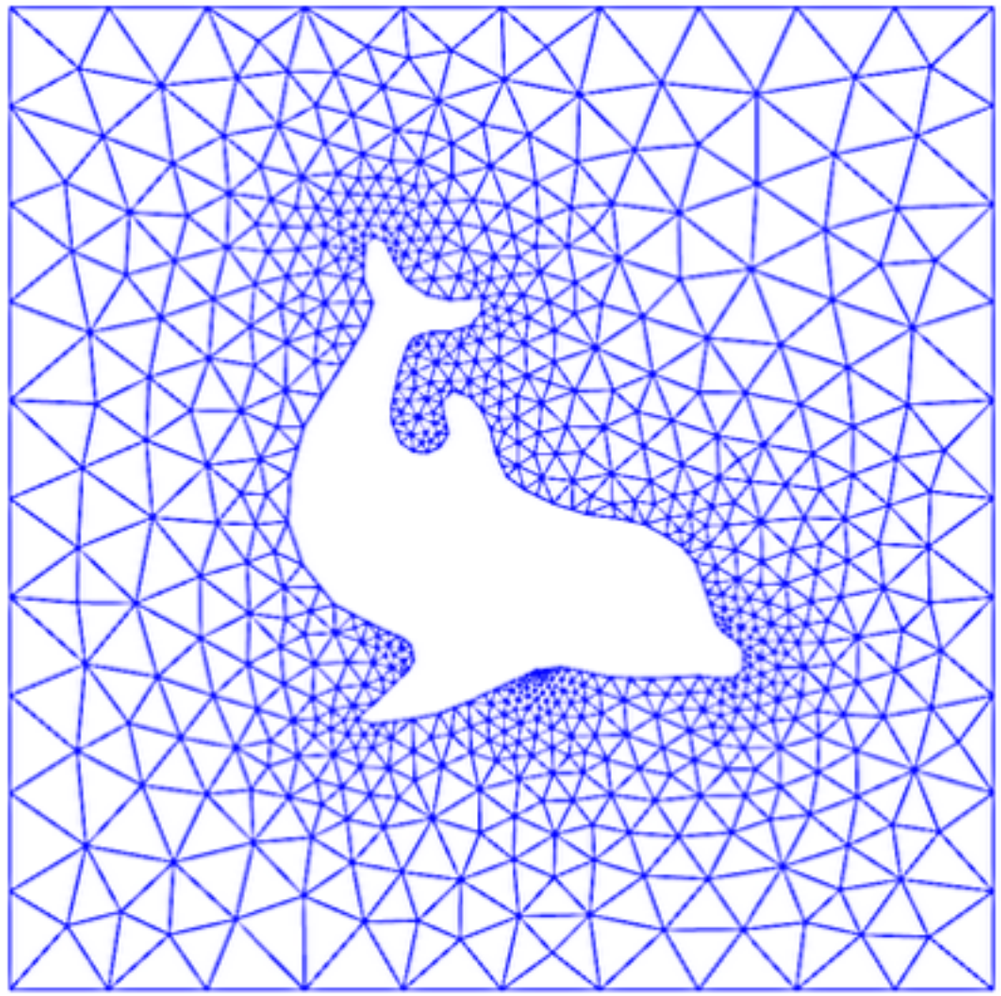
- Shown that we can address domain specific needs

What's next?

- Shown that we can address domain specific needs
- New domain focus: scientific computing

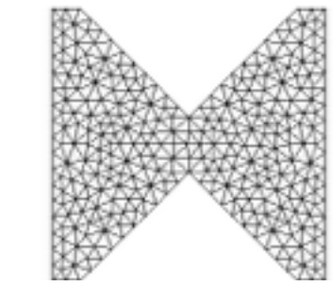
Scientific computing domain

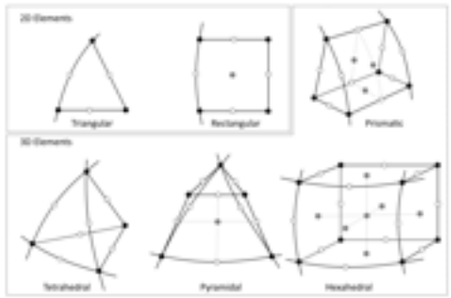
- Partial differential equations (PDEs) model a range of problems.

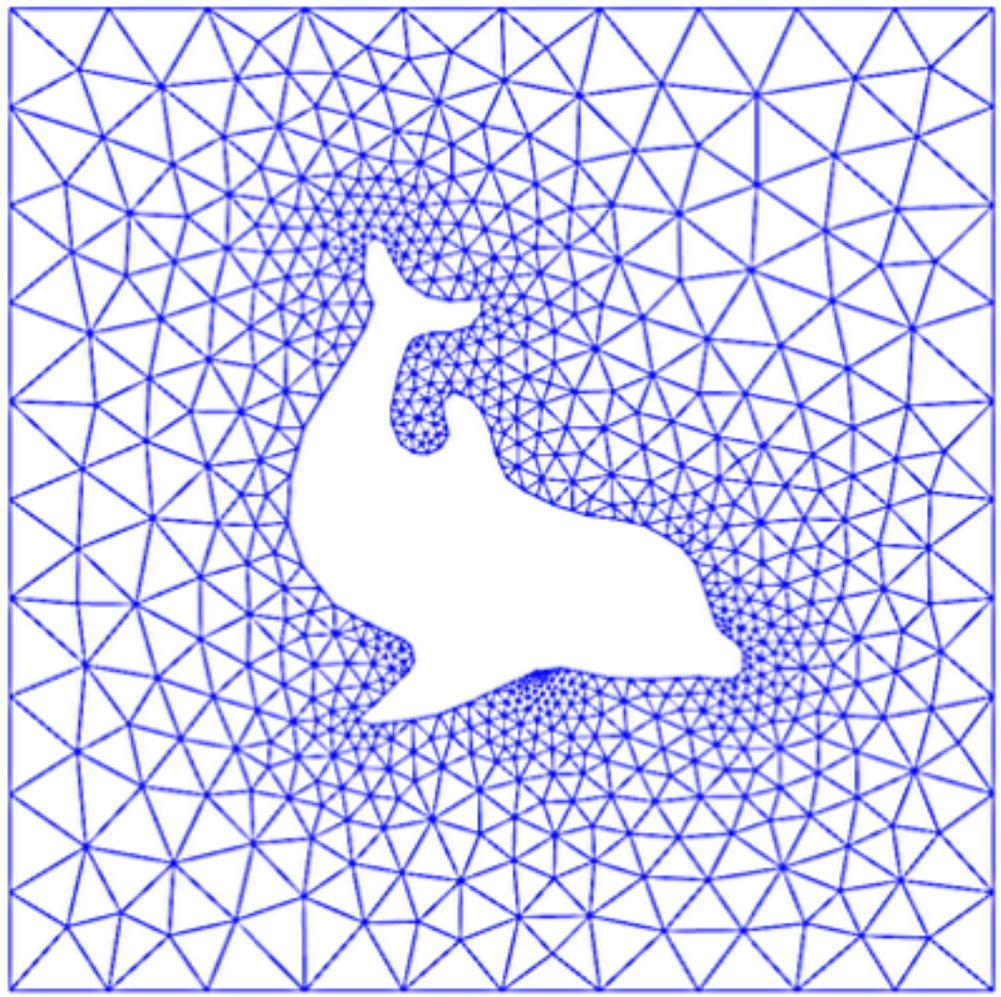


Scientific computing domain

- Partial differential equations (PDEs) model a range of problems.
- Various software tools are used solve PDEs





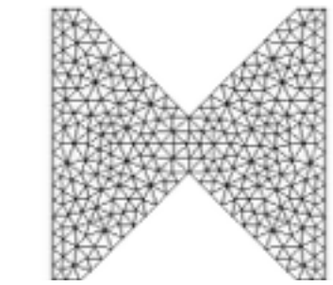


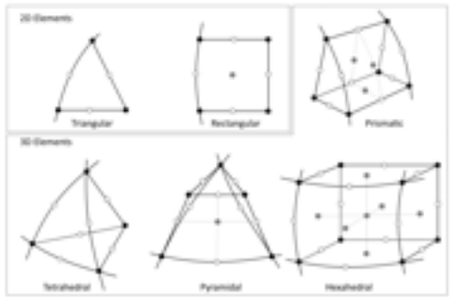
gmsh

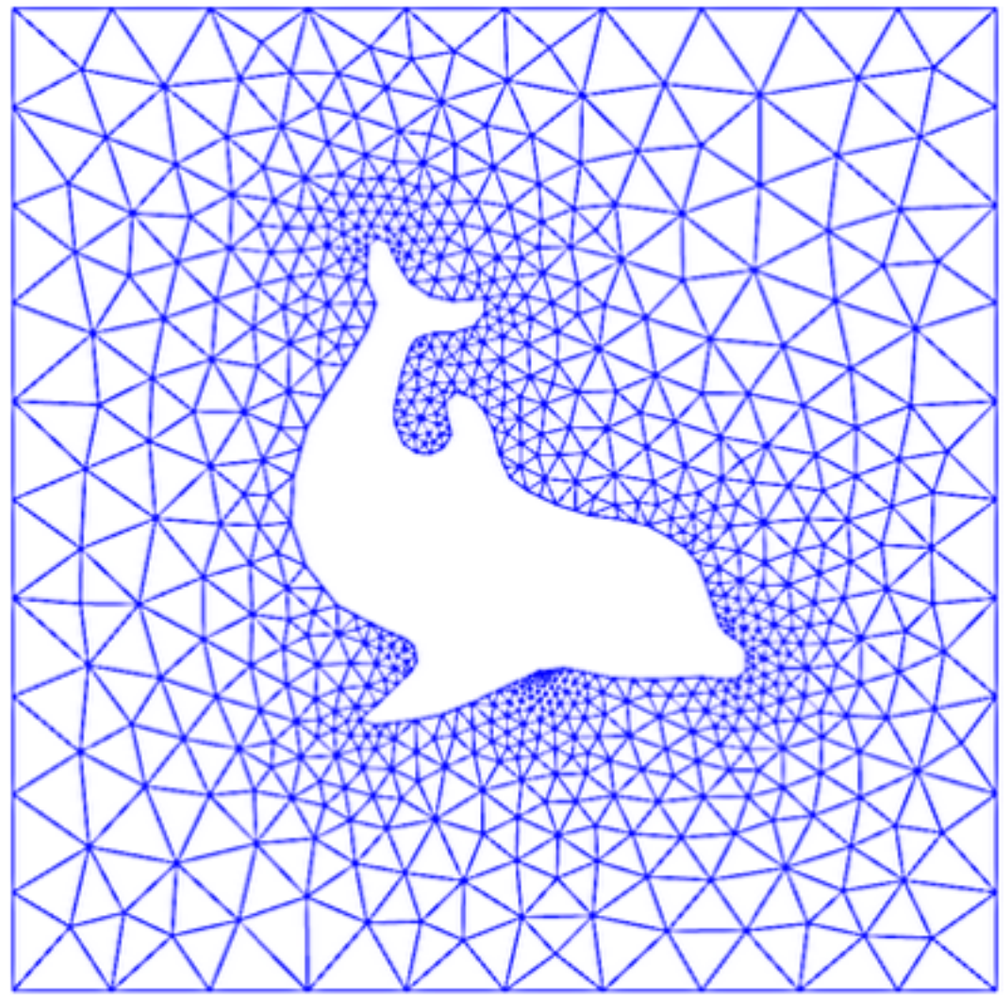
FENICS PROJECT

Scientific computing domain

- Partial differential equations (PDEs) model a range of problems.
- Various software tools are used solve PDEs
- Tools know PDEs, not Vis.







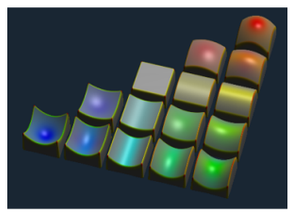
gmsh

FENICS PROJECT

Firedrake

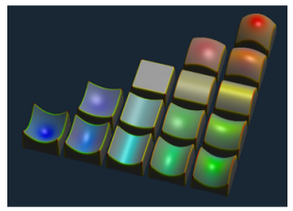
First approach

Connect Diderot and Firedrake with call back functions



```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
  if (inside(pos,F)){
    real out = F(pos);
    tensor[d] grad = G(pos);
    tensor[d,d] hess = H(pos);
  }
```

First approach



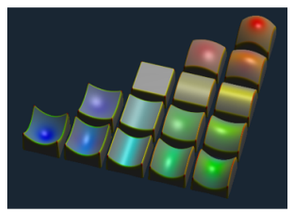
```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
if (inside(pos,F)){
  real out = F(pos);
  tensor[d] grad = G(pos);
  tensor[d,d] hess = H(pos);
}
```

inside_Field()

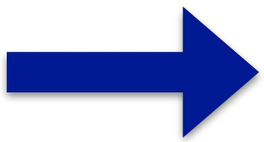
- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position

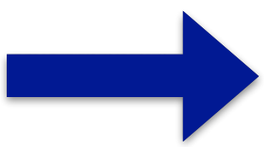
Firedrake

First approach



```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
  if (inside(pos,F)){
    real out = F(pos);
    tensor[d] grad = G(pos);
    tensor[d,d] hess = H(pos);
  }
```





inside_Field()

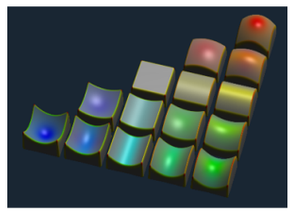
- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position

evaluate_Field()

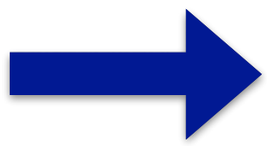
- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position

- get data coordinates
- compute weights
- get sbasis function
- evaluate basis functions

First approach



```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
  if (inside(pos,F)){
    real out = F(pos);
    tensor[d] grad = G(pos);
    tensor[d,d] hess = H(pos);
  }
```



inside_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position

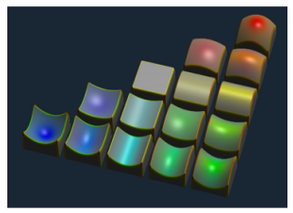
evaluate_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- get data coordinates
- compute weights
- get sbasis function
- evaluate basis functions

gradient_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- check cell
- use maps
- get basis Jacobian
- compute Jacobian inverse
- get data coordinates
- compute weights
- get derivative basis function values
- evaluate basis functions

First approach



```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
  if (inside(pos,F)){
    real out = F(pos);
    tensor[d] grad = G(pos);
    tensor[d,d] hess = H(pos);
  }
```

inside_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position

evaluate_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- get data coordinates
- compute weights
- get sbasis function
- evaluate basis functions

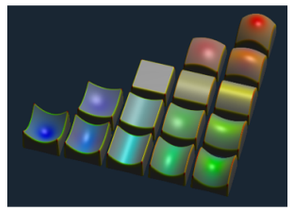
gradient_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- check cell
- use maps
- get basis Jacobian
- compute Jacobian inverse
- get data coordinates
- compute weights
- get derivative basis function values
- evaluate basis functions

H(pos)

First approach

Diderot



```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
  if (inside(pos,F)){
    real out = F(pos);
    tensor[d] grad = G(pos);
    tensor[d,d] hess = H(pos);
  }
```

Firedrake

inside_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position

evaluate_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- get data coordinates
- compute weights
- get sbasis function
- evaluate basis functions

gradient_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- check cell
- use maps
- get basis Jacobian
- compute Jacobian inverse
- get data coordinates
- compute weights
- get derivative basis function values
- evaluate basis functions

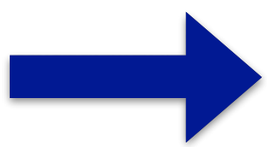
H(pos)

?

First approach

can't apply visualization

```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
  if (inside(pos,F)){
    real out = F(pos);
    tensor[d] grad = G(pos);
    tensor[d,d] hess = H(pos);
  }
```



inside_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position

evaluate_Field()

- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- get data coordinates
- compute weights
- get sbasis function
- evaluate basis functions

gradient_Field()

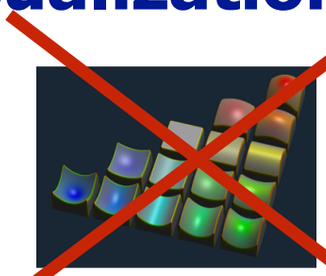
- get maps
- iterate over cells
- for each cell create a map
- get gbasis functions
- compute Jacobian inverse
- translate position
- check cell
- use maps
- get basis Jacobian
- compute Jacobian inverse
- get data coordinates
- compute weights
- get derivative basis function values
- evaluate basis functions

H(pos)

?

First approach

can't apply visualization



```
input fem#k(d)[] F;
field#k(d)[d] G = -∇F;
field#k(d)[d,d] H = -∇ ⊗ ∇F;
tensor[d] pos;
...
  if (inside(pos,F)){
    real out = F(pos);
    tensor[d] grad = G(pos);
    tensor[d,d] hess = H(pos);
  }
```

redundancy and missing optimizations

inside_Field()	gradient_Field()	evaluate_Field()	evaluate_Field()
get maps	get maps	get maps	check cell
iterate over cells	iterate over cells	iterate over cells	use maps
for each cell create a map	for each cell create a map	for each cell create a map	get basis Jacobian
get gbasis functions	get gbasis functions	get gbasis functions	compute Jacobian inverse
compute Jacobian inverse	compute Jacobian inverse	compute Jacobian inverse	get data coordinates
translate position	translate position	translate position	compute weights
		get data coordinates	get derivative basis function values
		compute weights	evaluate basis functions
		get sbasis function	
		evaluate basis functions	

H(pos)

?

The long term plan

- We want to expand the number of Diderot users by enhancing our understand of fields
- Build collaborations with experts in this domain
- Enable new informed visualization
- Domain-specific optimizations to improve run time

Conclusion

My work has been around enabling the expressivity in the Diderot language so that the user

- can write intuitive code
- compile programs with complicated tensor math
- believe in the correctness in the compiler
- and visualize different types of data

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

Questions?