SIGGRAPH ASIA 2021 TOKYO

“Heart”

I❤️LA: Compilable Markdown for Linear Algebra

Yong Li

GEORGE MASON UNIVERSITY

CraGL Creativity and Graphics Lab

Shoaib Kamil

Adobe

Alec Jacobson

University of Toronto

Adobe

Yotam Gingold

GEORGE MASON UNIVERSITY

CraGL Creativity and Graphics Lab

Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI

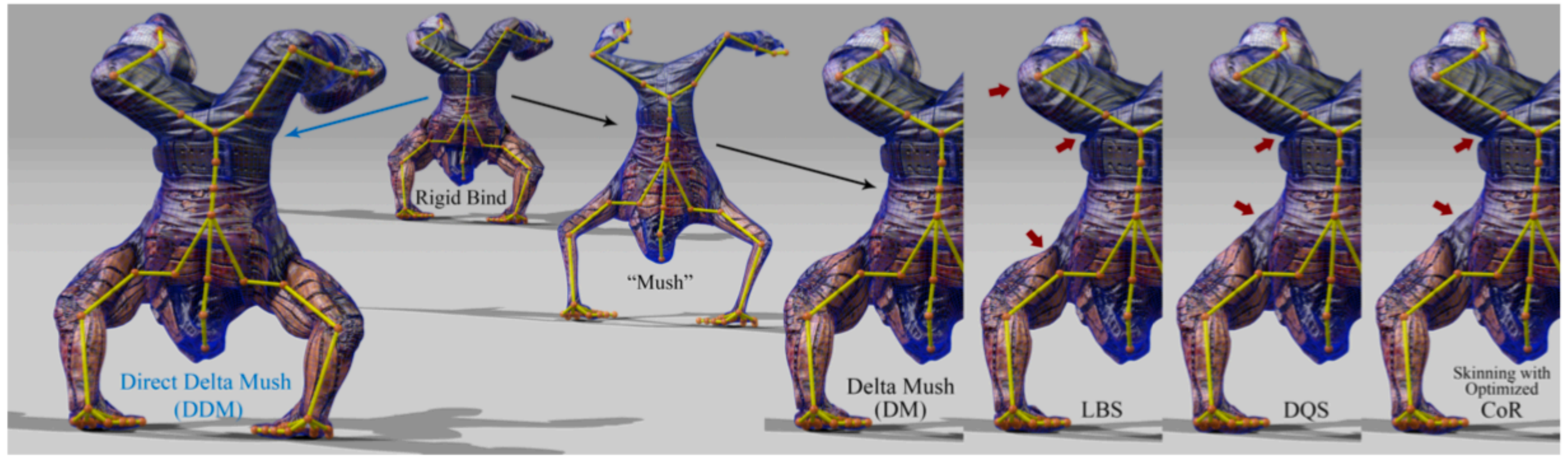


Fig. 1. The skinned model (left) is produced directly from the "unrigged" rigid bind model using our Direct Delta Mush algorithm. DDM can produce equivalent results to the Delta Mush algorithm but uses a direct local computation rather than the iterated global "mush" runtime smoothing of DM. The DM and DDM algorithms both provide greatly simplified authoring. They do not have the bulge and cleft artifacts common to other methods, which are prominent in the under-arm and hip regions (respectively) in this example (red arrows). DDM offers further advantages over DM, as described in the paper.

A significant fraction of the world's population have experienced virtual characters through games and movies, and the possibility of online VR social experiences may greatly extend this audience. At present, the skin deformation for interactive and real-time characters is typically computed using geometric skinning methods. These methods are efficient and simple to implement, but obtaining quality results requires considerable manual "rigging" effort involving trial-and-error weight painting, the addition of virtual helper bones, etc. The recently introduced Delta Mush algorithm largely solves this rig authoring problem, but its iterative computational approach has prevented direct adoption in real-time engines.

This paper introduces Direct Delta Mush, a new algorithm that simultaneously improves on the efficiency and control of Delta Mush while generalizing previous algorithms. Specifically, we derive a direct rather than iterative algorithm that has the same ballpark computational form as some previous geometric weight blending algorithms. Straightforward variants of the algorithm are then proposed to further optimize computational and storage cost with insignificant quality losses. These variants are equivalent to special cases of several previous skinning algorithms.

Our algorithm simultaneously satisfies the goals of reasonable efficiency, quality, and ease of authoring. Further, its explicit decomposition of rotational and translational effects allows independent control over bending versus twisting deformation, as well as a skin sliding effect.

Authors' addresses: Binh Huy Le, SEED - Electronic Arts, Redwood City, CA, bbinh85@gmail.com; JP Lewis, Google AI, San Francisco, CA, noisebrain@gmail.com.

0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

CCS Concepts: • **Computing methodologies** → **Animation**.

Additional Key Words and Phrases: skinning, skeletal animation, delta mush, real time, deformation, character animation

ACM Reference Format:
Binh Huy Le and JP Lewis. 2019. Direct Delta Mush Skinning and Variants. *ACM Trans. Graph.* 38, 4, Article 113 (July 2019), 13 pages. https://doi.org/10.1145/3306346.3322982

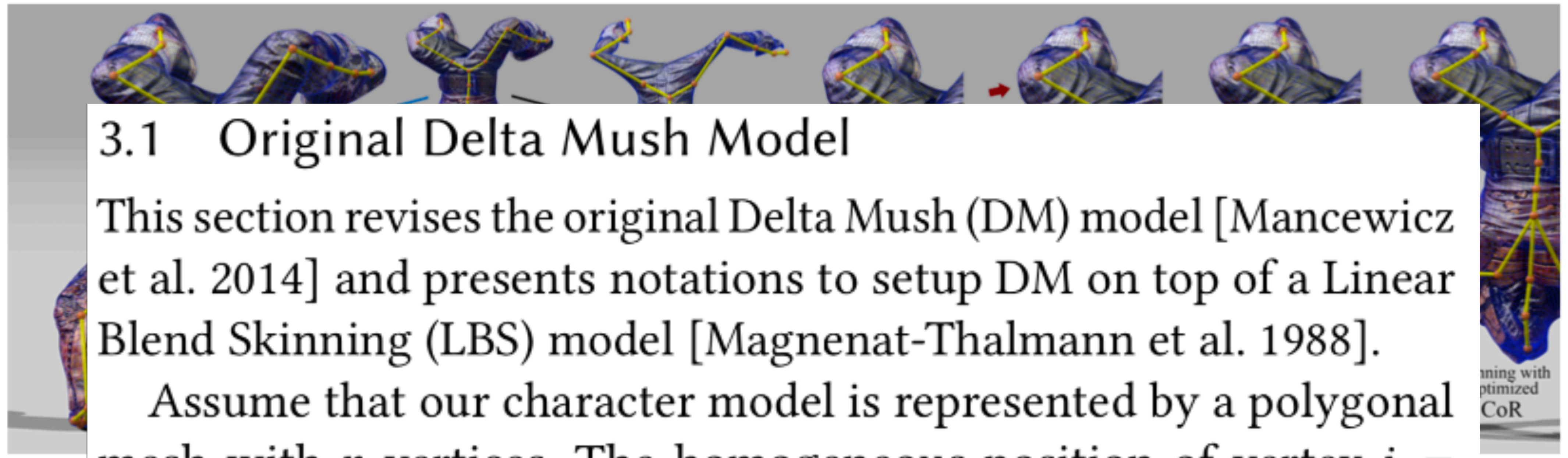
1 INTRODUCTION

Typically characters are the main focus of any movie or game. Major characters are often humans or animals, and thus are articulated models with rigid bones underlying deformable flesh and skin. Other objects in the scene such as trees can deform and may also be represented with a similar underlying approach. A key focus in all these cases is getting the deformation right.

A character deformation method suitable for games and interactive applications such as animation should have the following characteristics: (1) speed, (2) quality, (3) simplicity of setup and authoring. Existing approaches to character deformation can be very broadly classified into geometric skinning and simulation approaches. Simulation approaches produce the highest quality but may be less suitable in terms of criteria (1) and (3). Regarding speed, simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI



3.1 Original Delta Mush Model

This section revises the original Delta Mush (DM) model [Mancewicz et al. 2014] and presents notations to setup DM on top of a Linear Blend Skinning (LBS) model [Magnenat-Thalmann et al. 1988].

Assume that our character model is represented by a polygonal mesh with n vertices. The homogeneous position of vertex $i = 1..n$ at the rest pose is $\mathbf{u}_i \in \mathbb{R}^4$, where the 4^{th} component is 1. For convenience, we concatenate all vectors $\mathbf{u}_i$ to a matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_n] \in \mathbb{R}^{4\times n}$.

The deformation of $\mathbf{U}$ is driven by a LBS model with m bones. The transformation of bone $j = 1..m$ is $\mathbf{M}_j \in \mathbb{R}^{4\times 4}$. Let w_{ij} be the weight of bone j on vertex i. The weights are required to be affine, i.e. $\sum_{j=1}^{m} w_{ij} = 1, \forall i$. Non-negativity and sparseness have no effect on our formulation. Note that we use the LBS model for the sake of generality, but rigid binding is more common in practice, i.e. each vertex is only assigned to one bone ($w_{ij} \in \{0, 1\}$). The skinned geometry $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_n] \in \mathbb{R}^{4\times n}$ is computed as:

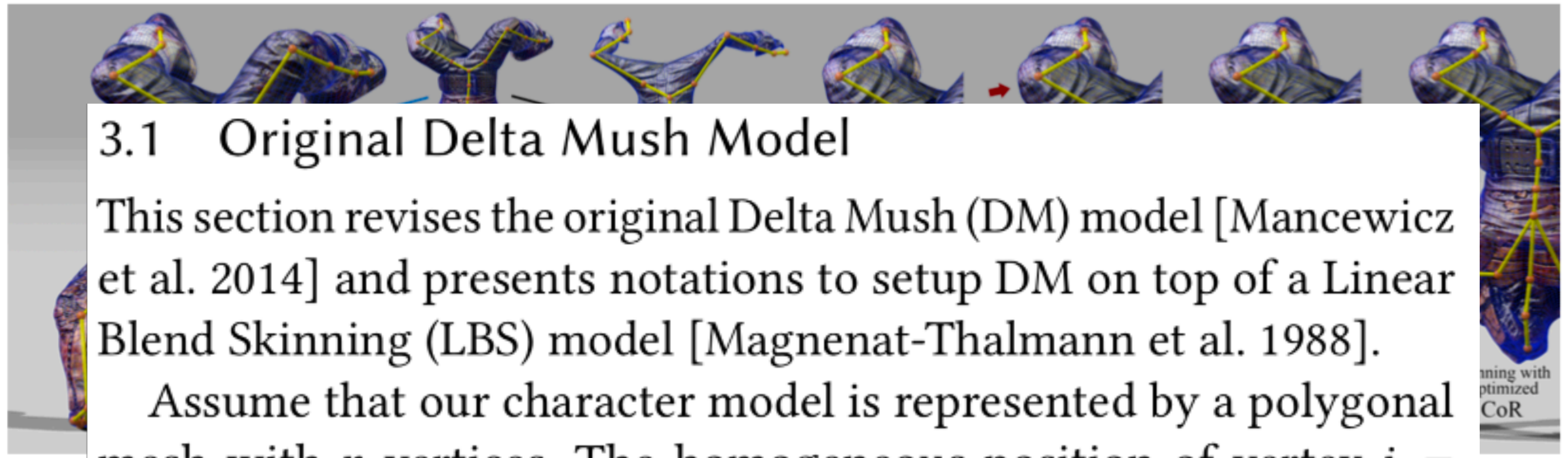
$$\mathbf{v}_i = \sum_{j=1}^{m} w_{ij}\mathbf{M}_j\mathbf{u}_i \ , \ i = 1..n \tag{1}$$

simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI



3.1 Original Delta Mush Model

This section revises the original Delta Mush (DM) model [Mancewicz et al. 2014] and presents notations to setup DM on top of a Linear Blend Skinning (LBS) model [Magnenat-Thalmann et al. 1988].

Assume that our character model is represented by a polygonal mesh with n vertices. The homogeneous position of vertex $i = 1..n$ at the rest pose is $\mathbf{u}_i \in \mathbb{R}^4$, where the 4^{th} component is 1. For convenience, we concatenate all vectors $\mathbf{u}_i$ to a matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_n] \in \mathbb{R}^{4 \times n}$.

The deformation of $\mathbf{U}$ is driven by a LBS model with m bones. The transformation of bone $j = 1..m$ is $\mathbf{M}_j \in \mathbb{R}^{4\times 4}$. Let w_{ij} be the weight of bone j on vertex i. The weights are required to be affine, i.e. $\sum_{j=1}^{m} w_{ij} = 1, \forall i$. Non-negativity and sparseness have no effect on our formulation. Note that we use the LBS model for the sake of generality, but rigid binding is more common in practice, i.e. each vertex is only assigned to one bone ($w_{ij} \in \{0, 1\}$). The skinned geometry $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_n] \in \mathbb{R}^{4\times n}$ is computed as:

$$\mathbf{v}_i = \sum_{j=1}^{m} w_{ij}\mathbf{M}_j\mathbf{u}_i \ , \ i = 1..n \tag{1}$$

simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

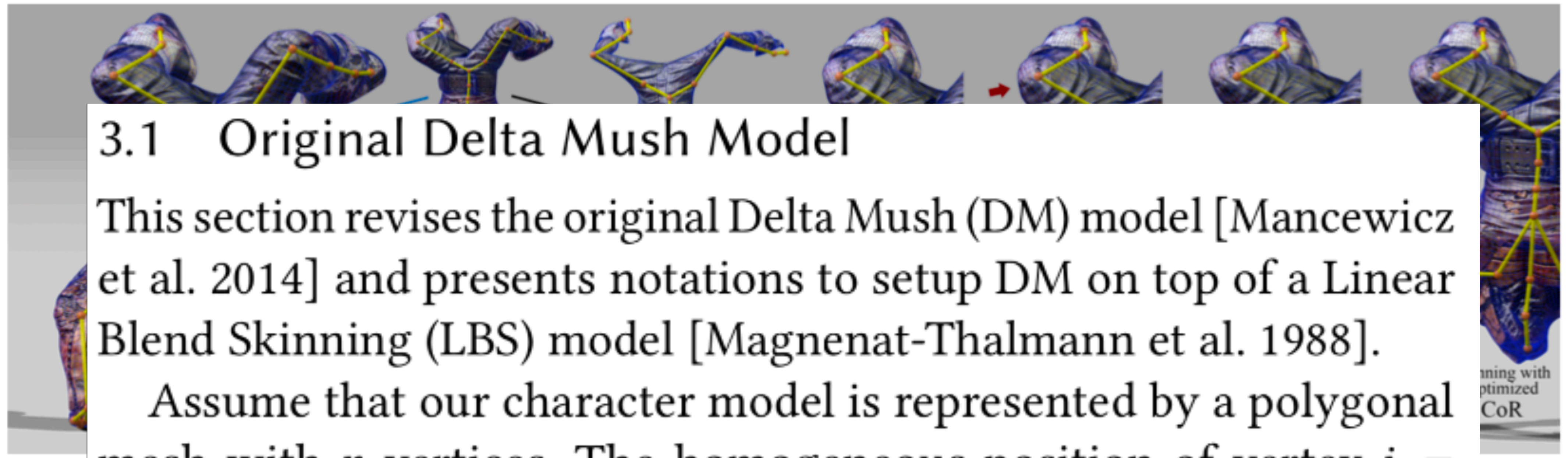
0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

Eigen_Test 〉 Eigen_Test 〉 main.cpp 〉 No Selection

```cpp
#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <iostream>

void get_skinned_geometry(const Eigen::MatrixXd & w,
                          const std::vector<Eigen::Matrix<double, 4, 4>> & M,
                          const Eigen::MatrixXd & U,
                          Eigen::MatrixXd & V)
{
    V.resize(4, U.cols());
    for (int i = 0; i < U.cols(); i++) {
        for (int j = 0; j < w.cols(); j++) {
            V.col(i) += w(i, j) * M[i] * U.col(i);
        }
    }
}

```

Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI



3.1 Original Delta Mush Model

This section revises the original Delta Mush (DM) model [Mancewicz et al. 2014] and presents notations to setup DM on top of a Linear Blend Skinning (LBS) model [Magnenat-Thalmann et al. 1988].

Assume that our character model is represented by a polygonal mesh with n vertices. The homogeneous position of vertex $i = 1..n$ at the rest pose is $\mathbf{u}_i \in \mathbb{R}^4$, where the 4^{th} component is 1. For convenience, we concatenate all vectors $\mathbf{u}_i$ to a matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_n] \in \mathbb{R}^{4\times n}$.

The deformation of $\mathbf{U}$ is driven by a LBS model with m bones. The transformation of bone $j = 1..m$ is $\mathbf{M}_j \in \mathbb{R}^{4\times 4}$. Let w_{ij} be the weight of bone j on vertex i. The weights are required to be affine, i.e. $\sum_{j=1}^{m} w_{ij} = 1, \forall i$. Non-negativity and sparseness have no effect on our formulation. Note that we use the LBS model for the sake of generality, but rigid binding is more common in practice, i.e. each vertex is only assigned to one bone ($w_{ij} \in \{0, 1\}$). The skinned geometry $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_n] \in \mathbb{R}^{4\times n}$ is computed as:

$$\mathbf{v}_i = \sum_{j=1}^{m} w_{ij}\mathbf{M}_j\mathbf{u}_i \ , \ i = 1..n \qquad (1)$$

0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

Eigen_Test 〉 Eigen_Test 〉 main.cpp 〉 No Selection

```
#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <iostream>

void get_skinned_geometry(const Eigen::MatrixXd & w,
                          const std::vector<Eigen::Matrix<double, 4, 4>> & M,
                          const Eigen::MatrixXd & U,
                          Eigen::MatrixXd & V)
{
    V.resize(4, U.cols());
    for (int i = 0; i < U.cols(); i++) {
        for (int j = 0; j < w.cols(); j++) {
            V.col(i) += w(i, j) * M[i] * U.col(i);
        }
    }
}
```

i should be j

Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI

3.1 Original Delta Mush Model

This section revises the original Delta Mush (DM) model [Mancewicz et al. 2014] and presents notations to setup DM on top of a Linear Blend Skinning (LBS) model [Magnenat-Thalmann et al. 1988].

Assume that our character model is represented by a polygonal mesh with n vertices. The homogeneous position of vertex $i = 1..n$ at the rest pose is $\mathbf{u}_i \in \mathbb{R}^4$, where the 4^{th} component is 1. For convenience, we concatenate all vectors $\mathbf{u}_i$ to a matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_n] \in \mathbb{R}^{4\times n}$.

The deformation of $\mathbf{U}$ is driven by a LBS model with m bones. The transformation of bone $j = 1..m$ is $\mathbf{M}_j \in \mathbb{R}^{4\times 4}$. Let w_{ij} be the weight of bone j on vertex i. The weights are required to be affine, i.e. $\sum_{j=1}^{m} w_{ij} = 1, \forall i$. Non-negativity and sparseness have no effect on our formulation. Note that we use the LBS model for the sake of generality, but rigid binding is more common in practice, i.e. each vertex is only assigned to one bone ($w_{ij} \in \{0, 1\}$). The skinned geometry $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_n] \in \mathbb{R}^{4\times n}$ is computed as:

$$\mathbf{v}_i = \sum_{j=1}^{m} w_{ij}\mathbf{M}_j\mathbf{u}_i \ , \ i = 1..n \qquad (1)$$

0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

Eigen_Test 〉 Eigen_Test 〉 main.cpp 〉 No Selection

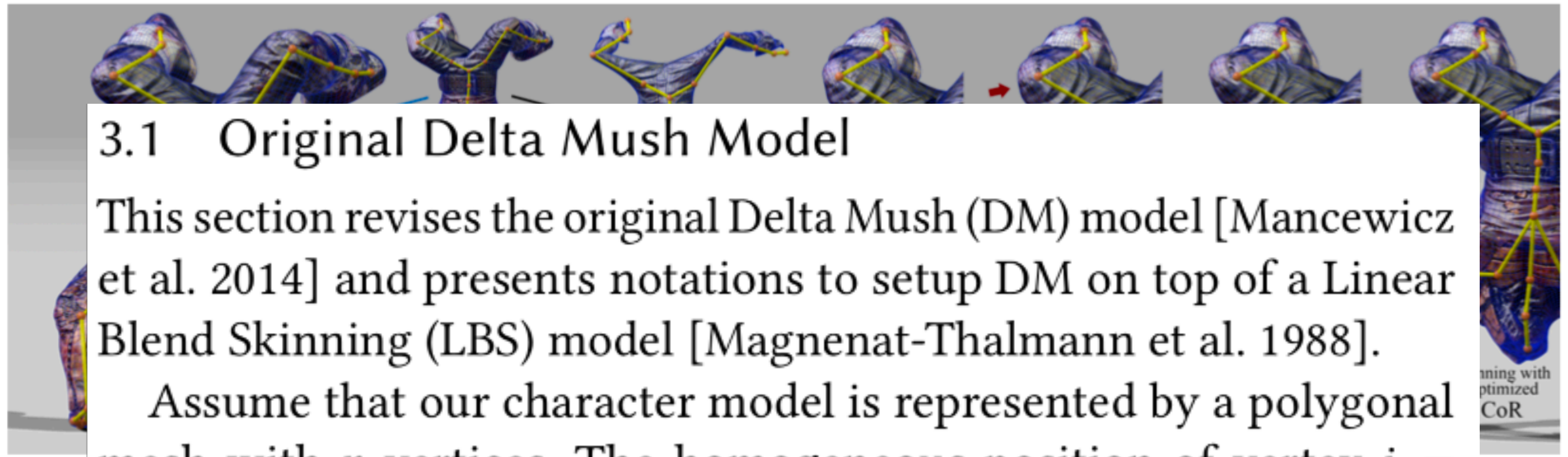
```cpp
#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <iostream>

void get_skinned_geometry(const Eigen::MatrixXd & w,
                          const std::vector<Eigen::Matrix<double, 4, 4>> & M,
                          const Eigen::MatrixXd & U,
                          Eigen::MatrixXd & V)
{
    V.resize(4, U.cols());
    for (int i = 0; i < U.cols(); i++) {
        for (int j = 0; j < w.cols(); j++) {
            V.col(i) += w(i, j) * M[i] * U.col(i);
        }
    }
}
```

i should be j

```python
import numpy as np


def get_skinned_geometry(w, M, U):
    v = np.zeros((4, U.shape[1]))
    for i in range(0, U.shape[1]):
        for j in range(0, w.shape[1]):
            v[:, i] += w[i, j] * M[j] @ U[:, i]
    return v
```

Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI



3.1 Original Delta Mush Model

This section revises the original Delta Mush (DM) model [Mancewicz et al. 2014] and presents notations to setup DM on top of a Linear Blend Skinning (LBS) model [Magnenat-Thalmann et al. 1988].

Assume that our character model is represented by a polygonal mesh with n vertices. The homogeneous position of vertex $i = 1..n$ at the rest pose is $\mathbf{u}_i \in \mathbb{R}^4$, where the 4^{th} component is 1. For convenience, we concatenate all vectors $\mathbf{u}_i$ to a matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_n] \in \mathbb{R}^{4\times n}$.

The deformation of $\mathbf{U}$ is driven by a LBS model with m bones. The transformation of bone $j = 1..m$ is $\mathbf{M}_j \in \mathbb{R}^{4\times 4}$. Let w_{ij} be the weight of bone j on vertex i. The weights are required to be affine, i.e. $\sum_{j=1}^{m} w_{ij} = 1, \forall i$. Non-negativity and sparseness have no effect on our formulation. Note that we use the LBS model for the sake of generality, but rigid binding is more common in practice, i.e. each vertex is only assigned to one bone ($w_{ij} \in \{0, 1\}$). The skinned geometry $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_n] \in \mathbb{R}^{4\times n}$ is computed as

$$\mathbf{v}_i = \sum_{j=1}^{m} w_{ij}\mathbf{M}_j\mathbf{u}_i \ , \ i = 1..n \qquad (1)$$

0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

Eigen_Test 〉 Eigen_Test 〉 main.cpp 〉 No Selection

```cpp
#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <iostream>

void get_skinned_geometry(const Eigen::MatrixXd & w,
                          const std::vector<Eigen::Matrix<double, 4, 4>> & M,
                          const Eigen::MatrixXd & U,
                          Eigen::MatrixXd & V)
{
    V.resize(4, U.cols());
    for (int i = 0; i < U.cols(); i++) {
        for (int j = 0; j < w.cols(); j++) {
            V.col(i) += w(i, j) * M[i] * U.col(i);
        }
    }
}
```

i should be j

```python
import numpy as np


def get_skinned_geometry(w, M, U):
    v = np.zeros((4, U.shape[1]))
    for i in range(0, U.shape[1]):
        for j in range(0, w.shape[1]):
            v[:, i] += w[i, j] * M[j] @ U[:, i]
    return v
```

Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI

3.1 Original Delta Mush Model

This section revises the original Delta Mush (DM) model [Mancewicz et al. 2014] and presents notations to setup DM on top of a Linear Blend Skinning (LBS) model [Magnenat-Thalmann et al. 1988].

Assume that our character model is represented by a polygonal mesh with n vertices. The homogeneous position of vertex $i = 1..n$ at the rest pose is $\mathbf{u}_i \in \mathbb{R}^4$, where the 4^{th} component is 1. For convenience, we concatenate all vectors $\mathbf{u}_i$ to a matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_n] \in \mathbb{R}^{4\times n}$.

The deformation of $\mathbf{U}$ is driven by a LBS model with m bones. The transformation of bone $j = 1..m$ is $\mathbf{M}_j \in \mathbb{R}^{4\times 4}$. Let w_{ij} be the weight of bone j on vertex i. The weights are required to be affi[…], i.e. $\sum_{j=1}^{m} w_{ij} = 1, \forall i$. Non-negativity and sparseness have no e[…]ect on our formulation. Note that we use the LBS model for the […]ke of generality, but rigid binding is more common in practice […]e. each vertex is only assigned to one bone ($w_{ij} \in \{0, 1\}$). Th[…] skinned geometry $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_n] \in \mathbb{R}^{4\times n}$ is computed as

$$\mathbf{v}_i = \sum_{j=1}^{m} w_{ij}\mathbf{M}_j\mathbf{u}_i \ , \ i = 1..n \qquad (1)$$

simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

4 AN EIGENANALYSIS OF I_5

4.1 The Eigensystem of I_5

We will now show that the eigensystem of any energy expressed solely in terms of I_5 can be written down in closed form. The I_5 invariant can be written in several forms,

$$I_5 = \|\mathbf{F}\mathbf{a}\|_2^2 = \mathbf{a}^T\mathbf{C}\mathbf{a} = \operatorname{tr}(\mathbf{C}\mathbf{A}), \qquad (5)$$

where $\mathbf{A} = \mathbf{a}\mathbf{a}^T$ and $\|\cdot\|_2^2$ denotes the squared Euclidean norm. The PK1 and Hessian in 3D are

$$\frac{\partial I_5}{\partial \mathbf{F}} = 2\mathbf{F}\mathbf{A} \qquad (6)$$

$$\frac{\partial^2 I_5}{\partial \mathbf{f}^2} = 2\begin{bmatrix} \mathbf{A}_{00}\mathbf{I}_{3\times3} & \mathbf{A}_{01}\mathbf{I}_{3\times3} & \mathbf{A}_{02}\mathbf{I}_{3\times3} \\ \mathbf{A}_{10}\mathbf{I}_{3\times3} & \mathbf{A}_{11}\mathbf{I}_{3\times3} & \mathbf{A}_{11}\mathbf{I}_{3\times3} \\ \mathbf{A}_{20}\mathbf{I}_{3\times3} & \mathbf{A}_{21}\mathbf{I}_{3\times3} & \mathbf{A}_{22}\mathbf{I}_{3\times3} \end{bmatrix} = 2\mathbf{H}_5, \qquad (7)$$

where $\mathbf{I}_{3\times3}$ is a 3×3 identity matrix, and $\mathbf{A}_{ij}$ is the (i, j) scalar entry of $\mathbf{A}$. (Appendix A shows the matrix explicitly.) Since Eqn. 7 is constant in $\mathbf{a}$, it is straightforward to state its eigensystem in closed form. In 3D, it contains three identical non-zero eigenvalues, $\lambda_{0,1,2} = 2\|\mathbf{a}\|_2^2$, and since fiber directions are usually normalized, this simplifies

```
A_ij = { 1 if (i,j) ∈ E
         0 otherwise
D_ii = ∑_j A_ij
L = D⁻¹(D-A)

where

E ∈ {ℤ×ℤ}
A ∈ ℝ^(n×n)
n ∈ ℤ
```

$$A_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$D_{i,i} = \sum_j A_{i,j}$$

$$L = D^{-1}(D - A)$$

$$where$$

$$E \in \{\mathbb{Z} \times \mathbb{Z}\}$$

$$A \in \mathbb{R}^{n \times n}$$

$$n \in \mathbb{Z}$$

C++

```cpp
/*
A_ij = { 1 if (i,j) ∈ E
       |    0 otherwise
D_ii = ∑_j A_ij
L = D⁻¹(D-A)

where

E ∈ {ℤ×ℤ}
A ∈ ℝ^(nxn)
n ∈ ℤ
*/
#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <iostream>
#include <set>

struct myExpressionResultType {
    Eigen::SparseMatrix<double> A;
    Eigen::SparseMatrix<double> D;
    Eigen::SparseMatrix<double> L;
    myExpressionResultType(const Ei
            const Eigen::SparseMa
            const Eigen::SparseMa
    : A(A),
    D(D),
    L(L)
    {}
};
```

Python

```python
"""
A_ij = { 1 if (i,j) ∈ E
       |    0 otherwise
D_ii = ∑_j A_ij
L = D⁻¹(D-A)

where

E ∈ {ℤ×ℤ}
A ∈ ℝ^(nxn)
n ∈ ℤ
"""
import numpy as np
import scipy
import scipy.linalg
from scipy import sparse
from scipy.integrate import quad
from scipy.optimize import minimize


class myExpressionResultType:
    def __init__( self, A, D, L):
        self.A = A
        self.D = D
        self.L = L


def myExpression(E, n):
    E = frozenset(E)

    assert all( len(el) == 2 for el
```

MATLAB

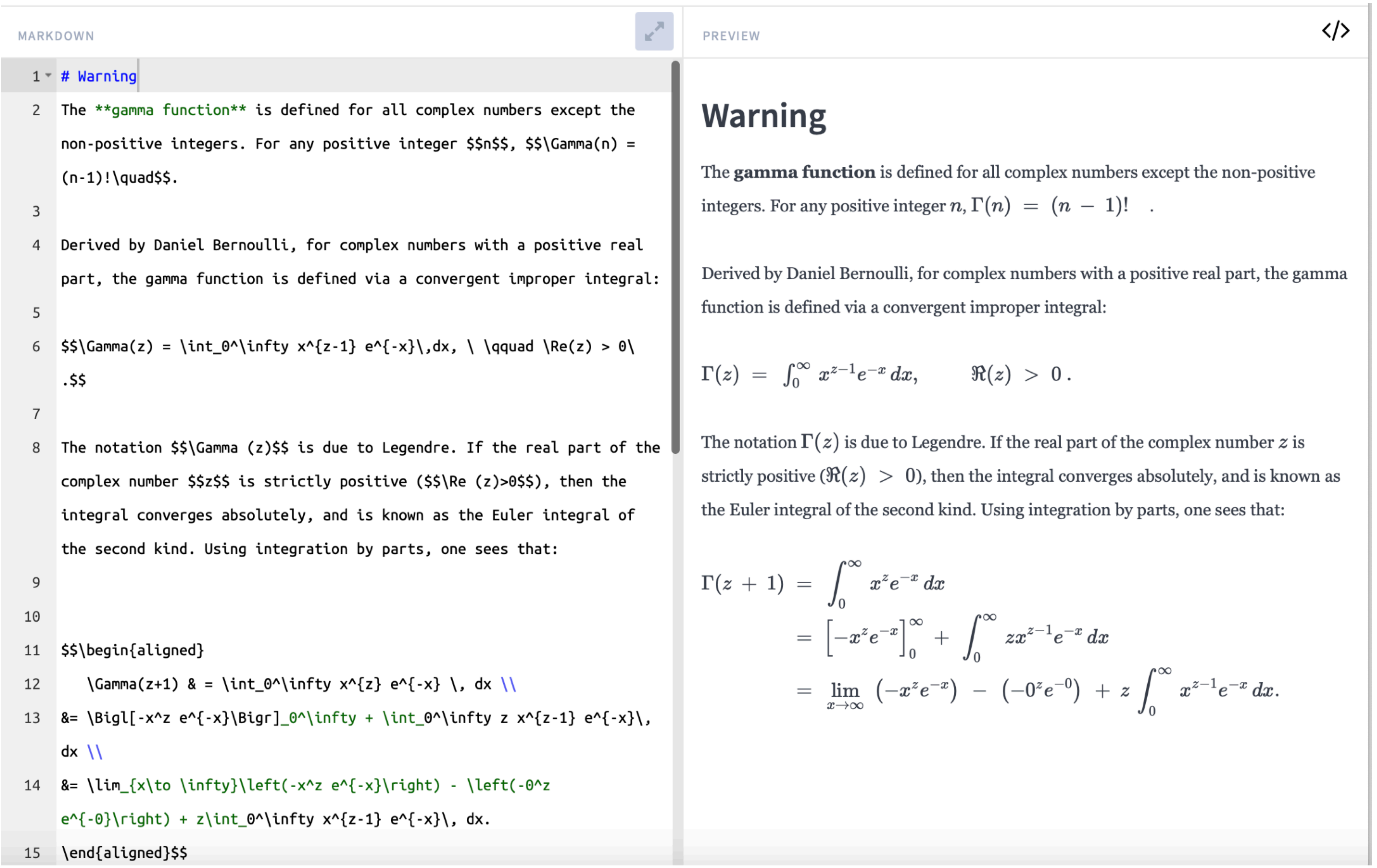
```matlab
function output = myExpression(E, n)
% output = myExpression(E, n)
%
%    A_ij = { 1 if (i,j) ∈ E
%             0 otherwise
%    D_ii = ∑_j A_ij
%    L = D⁻¹(D-A)
%
%    where
%
%    E ∈ {ℤ×ℤ}
%    A ∈ ℝ^(nxn)
%    n ∈ ℤ
    if nargin==0
        warning('generating random i
        [E, n] = generateRandomData(
    end
    function [E, n] = generateRandom
        n = randi(10);
        E = [];
        dim_2 = randi(10);
        for i = 1:dim_2
            E = [E;randi(10), randi(
        end
    end

    assert(size(E,2) == 2)
    assert(numel(n) == 1);

    Aij_0 = zeros(2,0);
    Avals_0 = zeros(1,0);
```

LaTeX

```latex
\documentclass[12pt]{article}
\usepackage{mathdots}
\usepackage[bb=boondox]{mathalfa}
\usepackage{mathtools}
\usepackage{amssymb}
\usepackage{libertine}
\DeclareMathOperator*{\argmax}{arg\
\DeclareMathOperator*{\argmin}{arg\
\usepackage[paperheight=8in,paperwi
\let\originalleft\left
\let\originalright\right
\renewcommand{\left}{\mathopen{}\mat
\renewcommand{\right}{\aftergroup\eg
\begin{document}

\begin{center}
\resizebox{\textwidth}{!}
{
\begin{minipage}[c]{\textwidth}
\begin{align*}
\mathit{A}_{\mathit{i}, \mathit{j}}
\mathit{D}_{\mathit{i},\mathit{i}} &
\mathit{L} & = \mathit{D}^{-1}\left(
\intertext{where}
\mathit{E} & \in \{\mathbb{Z} \times
\mathit{A} & \in \mathbb{R}^{ \mathi
\mathit{n} & \in \mathbb{Z} \\
\\
\end{align*}
\end{minipage}
}
```

Related work: Markup languages

- LaTeX [Goossens et al. 1994]
- Markdown [Gruber and Swartz 2004]
- AsciiMath [Jipsen 2005]
- MathML [W3C 2016]



Related work: DSLs for graphics

- [Perlin 1985]
- [Hanrahan and Lawson 1990]
- SafeGI [Ou and Pellacini 2010]
- Halide [Ragan-Kelley et al. 2012]
- VizGen [Yang et al. 2016]
- Simit [Kjolstad et al. 2016]
- Ebb [Bernstein et al. 2016]
- Opt [Devito et al. 2017]
- Slang [He et al. 2018]
- [Preussner 2018]
- Taichi [Hu et al. 2019]
- [Geisler et al. 2020]
- Penrose [Ye et al. 2020]
- TEG [Bangaru et al. 2021]

Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines

Jonathan Ragan-Kelley* Andrew Adams* Sylvain Paris† Marc Levoy‡ Saman Amarasinghe* Frédo Durand*

*MIT CSAIL †Adobe ‡Stanford University

Abstract

Using existing programming tools, writing high-performance image processing code requires sacrificing readability, portability, and modularity. We argue that this is a consequence of conflating what computations define the *algorithm*, with decisions about *storage* and the *order* of computation. We refer to these latter two concerns as the *schedule*, including choices of tiling, fusion, recomputation vs. storage, vectorization, and parallelism.

We propose a representation for feed-forward imaging pipelines that separates the algorithm from its schedule, enabling high-performance without sacrificing code clarity. This decoupling simplifies the algorithm specification: images and intermediate buffers become functions over an infinite integer domain, with no explicit storage or boundary conditions. Imaging pipelines are compositions of functions. Programmers separately specify scheduling strategies for the various functions composing the algorithm, which allows them to efficiently explore different optimizations without changing the algorithmic code.

We demonstrate the power of this representation by expressing a range of recent image processing applications in an embedded domain specific language called Halide, and compiling them for ARM, x86, and GPUs. Our compiler targets SIMD units, multiple cores, and complex memory hierarchies. We demonstrate that it can handle algorithms such as a camera raw pipeline, the bilateral grid, fast local Laplacian filtering, and image segmentation. The algorithms expressed in our language are both shorter and faster than state-of-the-art implementations.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Languages

Keywords: Image Processing, Compilers, Performance

Links: DL PDF WEB CODE

1 Introduction

Computational photography algorithms require highly efficient implementations to be used in practice, especially on power-constrained mobile devices. This is not a simple matter of programming in a low-level language like C. The performance difference between naive C and highly optimized C is often an order of magnitude. Unfortunately, optimization usually comes at the cost of programmer pain and code complexity, as computation must be reorganized to achieve memory efficiency and parallelism.

(a) Clean C++ : 9.94 ms per megapixel

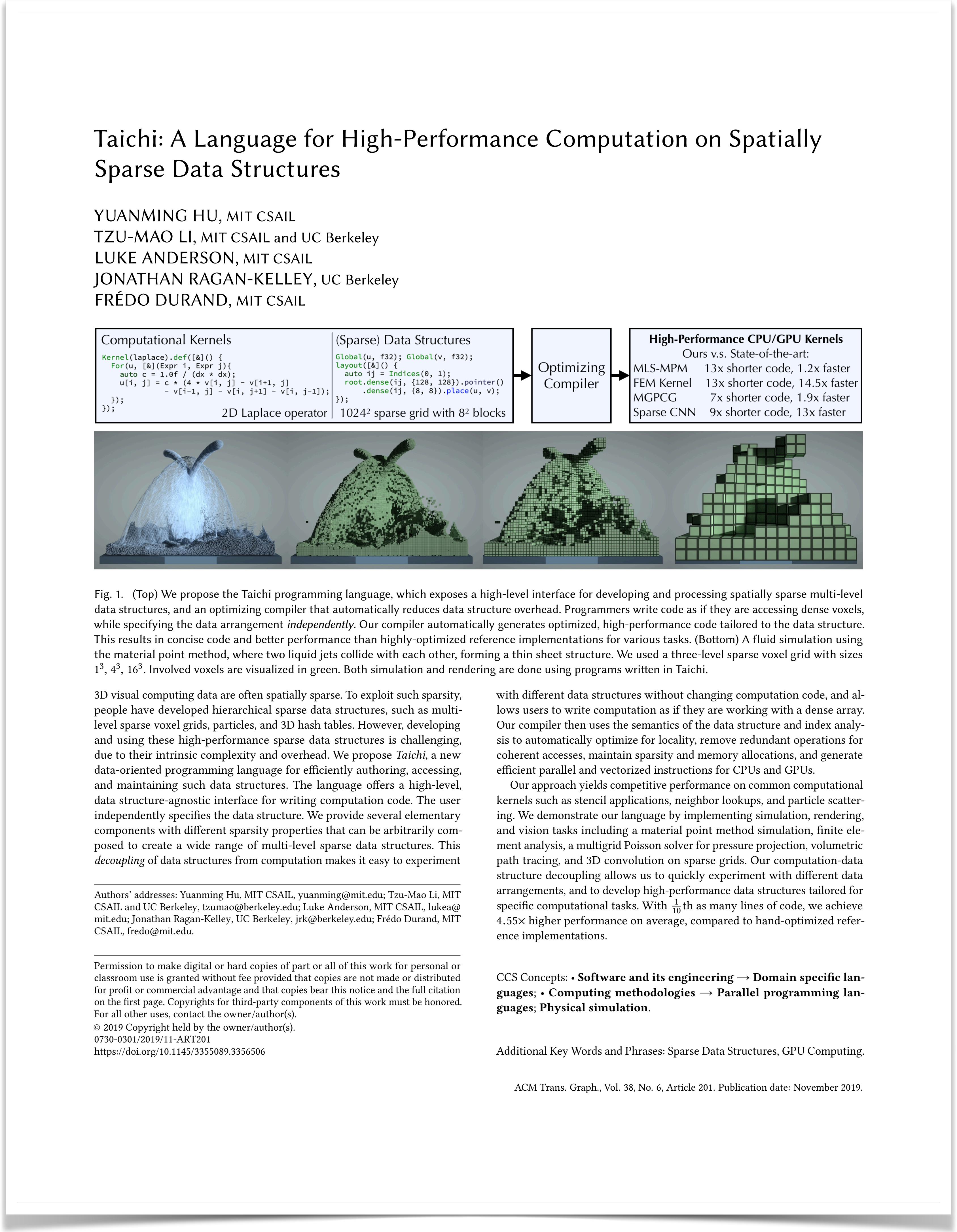
(b) Fast C++ (for x86) : 0.90 ms per megapixel

(c) Halide : 0.90 ms per megapixel

Figure 1: *The code at the top computes a 3×3 box filter using the composition of a 1×3 and a 3×1 box filter (a). Using vectorization, multithreading, tiling, and fusion, we can make this algorithm more than 10× faster on a quad-core x86 CPU (b). However, in doing so we've lost readability and portability. Our compiler separates the algorithm description from its schedule, achieving the same performance without making the same sacrifices (c). For the full details about how this test was carried out, see the supplemental material.*

Halide

[Ragan-Kelley et al. 2012]



Taichi: A Language for High-Performance Computation on Spatially Sparse Data Structures

YUANMING HU, MIT CSAIL
TZU-MAO LI, MIT CSAIL and UC Berkeley
LUKE ANDERSON, MIT CSAIL
JONATHAN RAGAN-KELLEY, UC Berkeley
FRÉDO DURAND, MIT CSAIL

Fig. 1. (Top) We propose the Taichi programming language, which exposes a high-level interface for developing and processing spatially sparse multi-level data structures, and an optimizing compiler that automatically reduces data structure overhead. Programmers write code as if they are accessing dense voxels, while specifying the data arrangement *independently*. Our compiler automatically generates optimized, high-performance code tailored to the data structure. This results in concise code and better performance than highly-optimized reference implementations for various tasks. (Bottom) A fluid simulation using the material point method, where two liquid jets collide with each other, forming a thin sheet structure. We used a three-level sparse voxel grid with sizes $1^3, 4^3, 16^3$. Involved voxels are visualized in green. Both simulation and rendering are done using programs written in Taichi.

3D visual computing data are often spatially sparse. To exploit such sparsity, people have developed hierarchical sparse data structures, such as multi-level sparse voxel grids, particles, and 3D hash tables. However, developing and using these high-performance sparse data structures is challenging, due to their intrinsic complexity and overhead. We propose *Taichi*, a new data-oriented programming language for efficiently authoring, accessing, and maintaining such data structures. The language offers a high-level, data structure-agnostic interface for writing computation code. The user independently specifies the data structure. We provide several elementary components with different sparsity properties that can be arbitrarily composed to create a wide range of multi-level sparse data structures. This *decoupling* of data structures from computation makes it easy to experiment with different data structures without changing computation code, and allows users to write computation as if they are working with a dense array. Our compiler then uses the semantics of the data structure and index analysis to automatically optimize for locality, remove redundant operations for coherent accesses, maintain sparsity and memory allocations, and generate efficient parallel and vectorized instructions for CPUs and GPUs.

Our approach yields competitive performance on common computational kernels such as stencil applications, neighbor lookups, and particle scattering. We demonstrate our language by implementing simulation, rendering, and vision tasks including a material point method simulation, finite element analysis, a multigrid Poisson solver for pressure projection, volumetric path tracing, and 3D convolution on sparse grids. Our computation-data structure decoupling allows us to quickly experiment with different data arrangements, and to develop high-performance data structures tailored for specific computational tasks. With $\frac{1}{10}$th as many lines of code, we achieve 4.55× higher performance on average, compared to hand-optimized reference implementations.

CCS Concepts: • **Software and its engineering → Domain specific languages;** • **Computing methodologies → Parallel programming languages; Physical simulation.**

Additional Key Words and Phrases: Sparse Data Structures, GPU Computing.

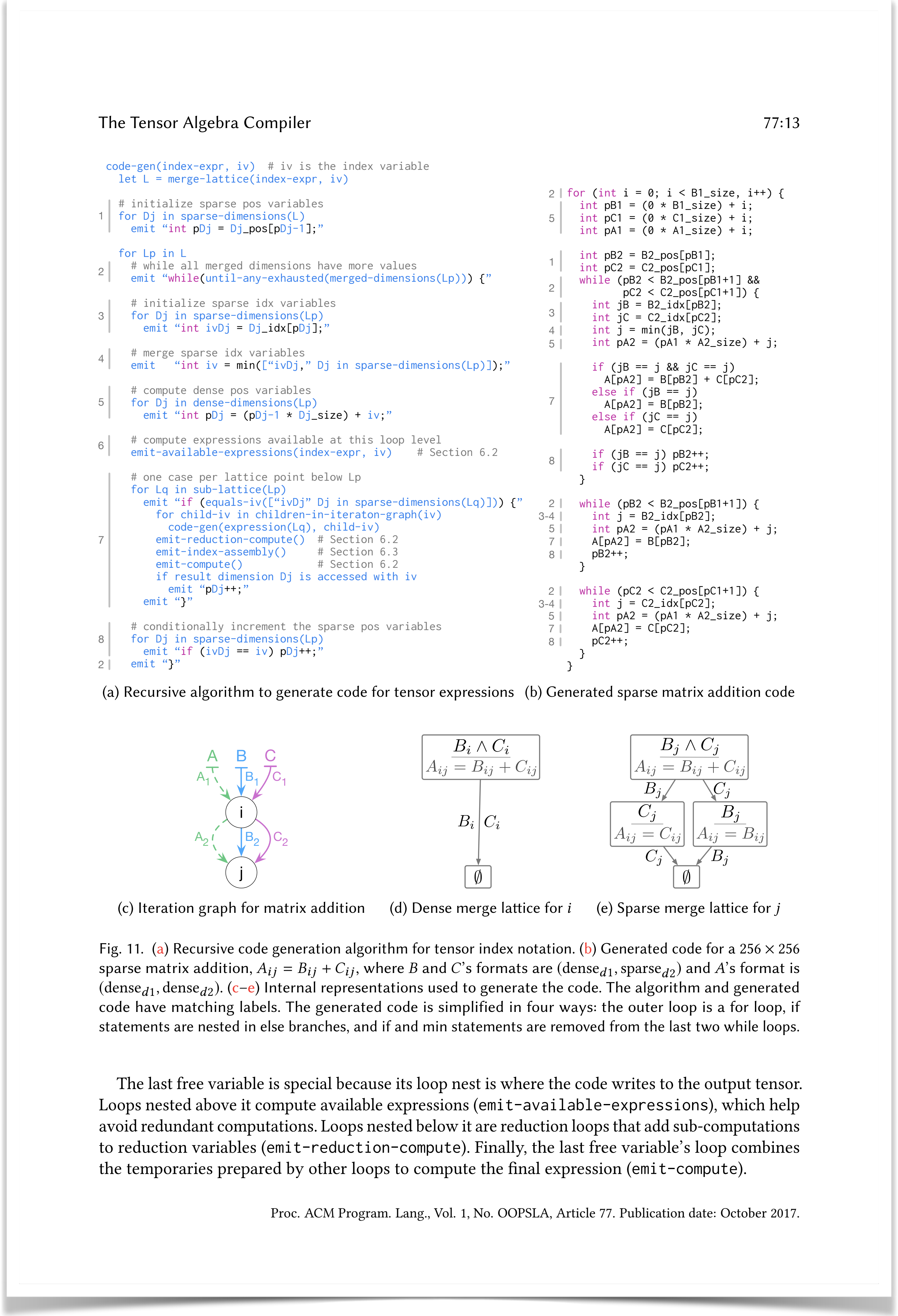
ACM Trans. Graph., Vol. 38, No. 6, Article 201. Publication date: November 2019.

Taichi

[Hu et al. 2019]

Related work: Languages for numerical computing

- YALMIP [Löfberg 2004]
- Fortress [Allen et al. 2005]
- APL [Iverson 2007]
- BLAC [Spampinato and Püschel 2014]
- Julia [Bezanson et al. 2017]
- TACO [Kjolstad et al. 2017]
- GENO [Laue et al. 2019]




```
  code-gen(index-expr, iv)  # iv is the index variable
    let L = merge-lattice(index-expr, iv)

    # initialize sparse pos variables
1 | for Dj in sparse-dimensions(L)
      emit "int pDj = Dj_pos[pDj-1];"

    for Lp in L
2 |   # while all merged dimensions have more values
      emit "while(until-any-exhausted(merged-dimensions(Lp))) {"

      # initialize sparse idx variables
3 |   for Dj in sparse-dimensions(Lp)
        emit "int ivDj = Dj_idx[pDj];"

4 |   # merge sparse idx variables
      emit   "int iv = min(["ivDj," Dj in sparse-dimensions(Lp)]);"

      # compute dense pos variables
5 |   for Dj in dense-dimensions(Lp)
        emit "int pDj = (pDj-1 * Dj_size) + iv;"

6 |   # compute expressions available at this loop level
      emit-available-expressions(index-expr, iv)    # Section 6.2

      # one case per lattice point below Lp
      for Lq in sub-lattice(Lp)
        emit "if (equals-iv(["ivDj" Dj in sparse-dimensions(Lq)])) {"
          for child-iv in children-in-iteraton-graph(iv)
            code-gen(expression(Lq), child-iv)
7 |       emit-reduction-compute()  # Section 6.2
          emit-index-assembly()     # Section 6.3
          emit-compute()            # Section 6.2
          if result dimension Dj is accessed with iv
            emit "pDj++;"
        emit "}"

      # conditionally increment the sparse pos variables
8 |   for Dj in sparse-dimensions(Lp)
        emit "if (ivDj == iv) pDj++;"
2 |   emit "}"
```

(a) Recursive algorithm to generate code for tensor expressions

```
  2 | for (int i = 0; i < B1_size, i++) {
        int pB1 = (0 * B1_size) + i;
  5 |   int pC1 = (0 * C1_size) + i;
        int pA1 = (0 * A1_size) + i;

  1 |   int pB2 = B2_pos[pB1];
        int pC2 = C2_pos[pC1];
  2 |   while (pB2 < B2_pos[pB1+1] &&
               pC2 < C2_pos[pC1+1]) {
  3 |     int jB = B2_idx[pB2];
          int jC = C2_idx[pC2];
  4 |     int j = min(jB, jC);
  5 |     int pA2 = (pA1 * A2_size) + j;

          if (jB == j && jC == j)
            A[pA2] = B[pB2] + C[pC2];
  7 |     else if (jB == j)
            A[pA2] = B[pB2];
          else if (jC == j)
            A[pA2] = C[pC2];

  8 |     if (jB == j) pB2++;
          if (jC == j) pC2++;
        }

  2 |   while (pB2 < B2_pos[pB1+1]) {
3-4 |     int j = B2_idx[pB2];
  5 |     int pA2 = (pA1 * A2_size) + j;
  7 |     A[pA2] = B[pB2];
  8 |     pB2++;
        }

  2 |   while (pC2 < C2_pos[pC1+1]) {
3-4 |     int j = C2_idx[pC2];
  5 |     int pA2 = (pA1 * A2_size) + j;
  7 |     A[pA2] = C[pC2];
  8 |     pC2++;
        }
      }
```

(b) Generated sparse matrix addition code

(c) Iteration graph for matrix addition (d) Dense merge lattice for i (e) Sparse merge lattice for j

Fig. 11. (a) Recursive code generation algorithm for tensor index notation. (b) Generated code for a 256×256 sparse matrix addition, $A_{ij} = B_{ij} + C_{ij}$, where B and C's formats are $(\text{dense}_{d1}, \text{sparse}_{d2})$ and A's format is $(\text{dense}_{d1}, \text{dense}_{d2})$. (c–e) Internal representations used to generate the code. The algorithm and generated code have matching labels. The generated code is simplified in four ways: the outer loop is a for loop, if statements are nested in else branches, and if and min statements are removed from the last two while loops.

The last free variable is special because its loop nest is where the code writes to the output tensor. Loops nested above it compute available expressions (`emit-available-expressions`), which help avoid redundant computations. Loops nested below it are reduction loops that add sub-computations to reduction variables (`emit-reduction-compute`). Finally, the last free variable's loop combines the temporaries prepared by other loops to compute the final expression (`emit-compute`).

TACO

[Kjolstad et al. 2017]

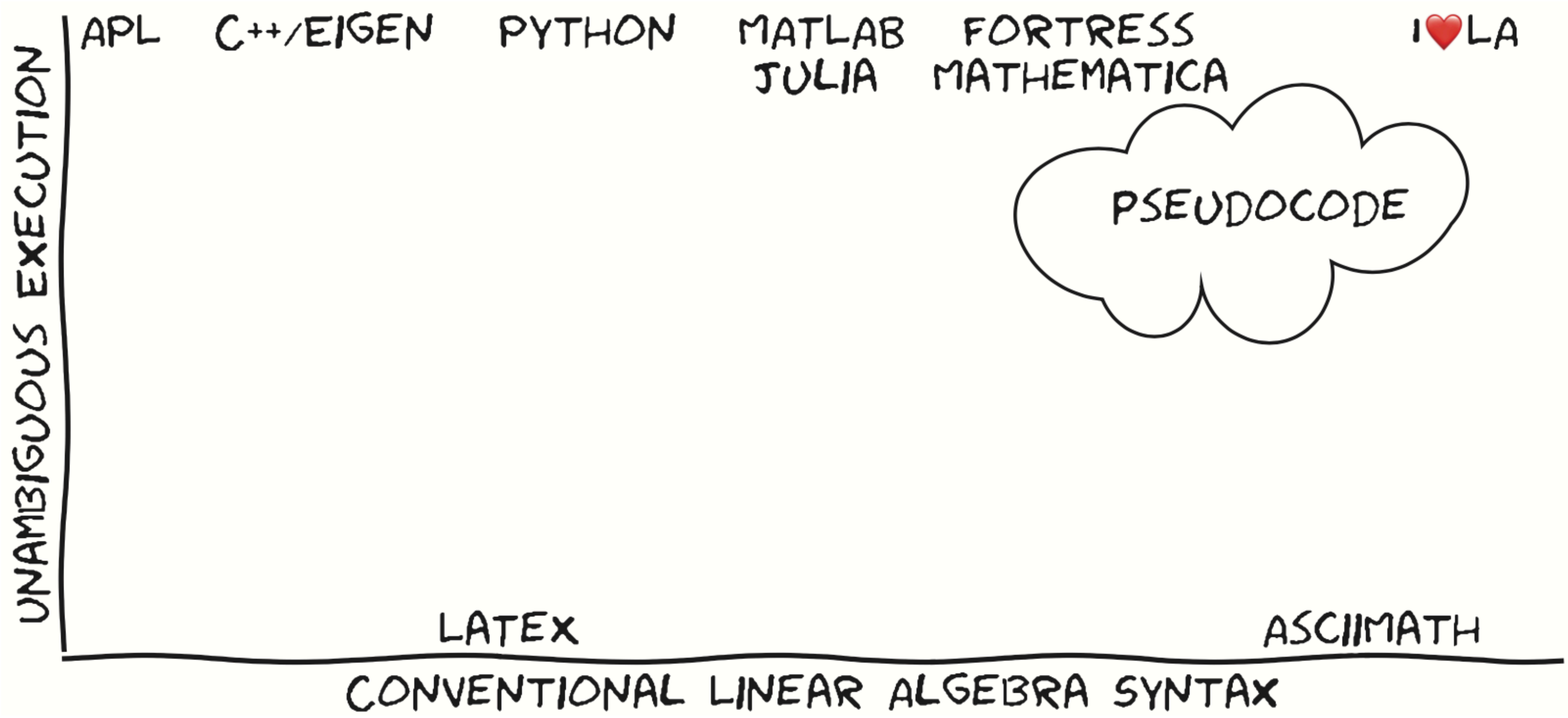
Related work: Languages for proof-checking

- Agda [Norell 2007]
- Lean [de Moura et al. 2015]
- Coq [Team 2021]

```
theorem le.antisymm : ∀ {a b : ℤ}, a ≤ b → b ≤ a → a = b :=
take a b : ℤ, assume (H₁ : a ≤ b) (H₂ : b ≤ a),
obtain (n : ℕ) (Hn : a + n = b), from le.elim H₁,
obtain (m : ℕ) (Hm : b + m = a), from le.elim H₂,
have H₃ : a + of_nat (n + m) = a + 0, from
... -- suppressed rest of the proof due to space limitations
have H₆ : n = 0, from nat.eq_zero_of_add_eq_zero_right H₅,
show a = b, from
  calc
    a = a + 0       : add_zero
      ... = a + n   : H₆
      ... = b       : Hn
```

Lean example

I❤️LA combines conventional syntax with unambiguous execution




SIGGRAPH 2019 papers

$$L_s = L_f + \lambda L_{KL} + \eta R, \tag{7}$$

where

$$L_{KL} = KL(q_\phi(z|x,y,c)||p_\phi(z|c)), \tag{8}$$

$$R = \sum_{i=1}^{k} ||h'_i - h_i||_2^2 + \sum_{i=1}^{k}\sum_{j=i+1}^{k} ||h'_{i,j} - h_{i,j}||_2^2, \tag{9}$$

$$B_i = \begin{bmatrix} \cos(\omega_i T) & \sin(\omega_i T) \\ -\sin(\omega_i T) & \cos(\omega_i T) \end{bmatrix}. \tag{15}$$

$$\mathbb{D}_i = \sum_{j=i-M,\ldots,i-1} W_j \cdot (\mathbb{D}_i + f^{j\to i}) + W_i \cdot \mathbb{D}_i \tag{8}$$

$$\frac{W}{\sqrt{N}} \leq \frac{80 z_{\min}}{S}. \tag{11}$$

$$m_{\mathbf{v}}^{\mathbf{i}} = E_O\left[\sum_{\vec{\mathbf{x}} \in \mathbb{P}_{\mathbf{v}}^{\mathbf{i},O}} \mu(\vec{\mathbf{x}})\right], \tag{14}$$

$$C_{\mathbf{v}_1,\mathbf{v}_2 2}^{\mathbf{i}_1,\mathbf{i}_2} = E_O\left[\sum_{\vec{\mathbf{x}}^1 \in \mathbb{P}_{\mathbf{v}_1}^{\mathbf{i}_1,O}, \vec{\mathbf{x}}^2 \in \mathbb{P}_{\mathbf{v}_2}^{\mathbf{i}_2,O}} \mu(\vec{\mathbf{x}}^1)\cdot\mu(\vec{\mathbf{x}}^2)^*\right] - m_{\mathbf{v}_1}^{\mathbf{i}_1}\cdot m_{\mathbf{v}_2}^{\mathbf{i}_2 *}. \tag{15}$$

$$\langle I\rangle_{\mathrm{MIS}} = \sum_{i=1}^{n}\frac{1}{n_i}\sum_{j=1}^{n_i} w_i(\overline{\mathbf{x}}_{i,j})\frac{f(\overline{\mathbf{x}}_{i,j})}{p_i(\overline{\mathbf{x}}_{i,j})}, \tag{42}$$

$$\mathcal{L}_{\mathrm{rec}} = \mathbb{E}_{\mathbf{p}_{i,j}\sim\mathcal{P}}\left[\|D(E_M(\mathbf{p}_{i,j}), E_S(\mathbf{p}_{i,j})) - \mathbf{p}_{i,j}\|^2\right].$$

$$\min_{0\leq\tau_i\leq\tau_{\max}} T(\tilde{\mathbf{x}}(\boldsymbol{\tau})) + \eta\sum_i \tau_i^p, \tag{7}$$

$$\min_{C} E_{\mathrm{sim}}(\mathcal{S},\mathcal{S}^c) = E_{\mathrm{sim}}(\mathcal{S},\Psi(\mathcal{M}^c)), \tag{1}$$

$$\mathcal{E}_{\mathrm{reg}}(\beta,\theta) = \omega_{\mathrm{shape}}E_{\mathrm{shape}}(\beta) + \omega_{\mathrm{pose}}E_{\mathrm{pose}}(\theta) \tag{7}$$

$$\omega_{\mathrm{temp}}E_{\mathrm{temp}}(\beta,\theta) + \omega_{\mathrm{coll}}E_{\mathrm{coll}}(\beta,\theta). \tag{8}$$

$$\Pr(D^+|\mathbf{x}) = (1+\exp(\alpha f(\mathbf{x})+\beta))^{-1}. \tag{3}$$

$$\frac{D(t,d_j)}{D'} + \gamma\frac{l_{outer}(S_i^j,t)}{l_{inner}(S_i^j,t)} \tag{9}$$

$$\langle I\rangle_{a,n} \approx \frac{f(\overline{\xi_n})I_a(\overline{\xi_n})}{p(\overline{\xi_n})}, \tag{16}$$

$$\begin{aligned}\mathrm{var}[\bar{\sigma}_t] &= \frac{1}{t^2}(\langle\tau(\mathbf{x},t)^2\rangle - \langle\tau(\mathbf{x},t)\rangle^2)\\ &= \frac{1}{t^2}\int_0^t\int_0^t \langle\sigma_\mu(\mathbf{x}')\sigma_\mu(\mathbf{x}'')\rangle \mathrm{d}t'\mathrm{d}t''\\ &= \frac{1}{t^2}\int_0^t\int_0^t \mathrm{cov}(\mathbf{x}',\mathbf{x}'')\mathrm{d}t'\mathrm{d}t''.\end{aligned} \tag{14}$$

$$\hat{\mathbf{I}} = \arg\min_{\mathbf{I}} \|\mathbf{J} - \Phi\mathbf{I}\|_2^2 + R(\mathbf{I}), \tag{9}$$

$$K_{ij} = \sum_{\widehat{C}\in\widehat{\mathcal{M}}}\int_{\mathrm{g}(\widehat{C})} \nabla\phi_i(\mathbf{x})\cdot\nabla\phi_j(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

$$\max_{\mathbf{q}_{1:T},\dot{\mathbf{q}}_{1:T},\boldsymbol{\tau}_{1:T}} \dot{x}_T\cdot\dot{y}_T,$$

$$\text{subject to}\quad \ddot{\mathbf{q}}_t = \mathbf{S}(\boldsymbol{\tau}_t,\mathbf{q}_t,\dot{\mathbf{q}}_t),\quad -\boldsymbol{\epsilon}\leq\mathbf{R}(\mathbf{q}_t,\dot{\mathbf{q}}_t,\boldsymbol{\tau}_t)\leq\boldsymbol{\epsilon},\quad \mathbf{q}_1 = \bar{\mathbf{q}};\ \ \dot{\mathbf{q}}_1 = \dot{\bar{\mathbf{q}}},$$

$$\mathcal{B}(p,q) = \left\{x\in\mathbb{R}^2 : d_{\mathcal{S}}(p,x) = d_{\mathcal{S}}(q,x)\right\} \tag{9}$$

$$\Psi(x,y,z,t) = \iiint \Phi(k_x,k_y,k_z)e^{2\pi i(k_x x+k_y y+k_z z-ft)}\,\mathrm{d}k_x\mathrm{d}k_y\mathrm{d}k_z. \tag{3}$$

$$\underset{\hat{\mathbf{w}},\boldsymbol{\delta}}{\arg\min}\ \frac{1}{2}\|\boldsymbol{\delta}\|^2 \quad \text{s.t.} \quad C(\hat{\mathbf{w}}+\boldsymbol{\delta}) = \mathbf{0},\ \ \|\hat{\mathbf{w}}_i\| = 1.$$

$$\begin{pmatrix} d_0^\top \\ B \\ C \end{pmatrix}\theta_e = K_c - \overline{K}_c.$$

$$\mathbb{E}_{I\sim\mathcal{D}}\left(||\tau(I) - \tau(I_{\mathrm{gt}})||^2\right), \tag{7}$$

$$\frac{\partial^2\Psi_{\mathrm{AA}}}{\partial\mathbf{f}} = \mu\left[\left(1 - \frac{\mathcal{S}(I_4)}{\sqrt{I_5}}\right)\mathbf{H}_5 + \frac{\mathcal{S}(I_4)}{I_5^{3/2}}\mathbf{f}_a\mathbf{f}_a^T\right]. \tag{36}$$

$$f = f(l,\dot{l},a) = a\cdot f_l(l)\cdot f_v(\dot{l}) + f_p(l),$$

$$\mathcal{Y}_{t_0}^{t+\Delta t}(\mathbf{x}) = \mathcal{Y}_{t_0}^t(\mathbf{x}) + \Delta t\mathbf{u}(\mathcal{Y}_{t_0}^t(\mathbf{x}),t)$$

$$\mathcal{H}[2g-1](z) = -\frac{2}{\pi}i\log(i\alpha+2\pi\mathcal{H}[d](z))$$

$$\overset{\text{Eq. (4)}}{\Rightarrow}\ (2g-1)(\varphi) = \frac{2}{\pi}\lim_{z\to\exp(i\varphi)}\Re(-i\log(i\alpha+2\pi\mathcal{H}[d](z)))$$

$$\Rightarrow\ g(\varphi) = \frac{1}{\pi}\arg\left(i\alpha + \lim_{z\to\exp(i\varphi)}2\pi\mathcal{H}[d](z)\right) + \frac{1}{2} \tag{5}$$

$$\varphi(\mathbf{x}) = \mathrm{atan2}\left(\sum_j a_j(\mathbf{x})\sin(\varphi_j), \sum_j a_j(\mathbf{x})\cos(\varphi_j)\right) \tag{9}$$

$$E_1(p,T) = \begin{cases} 0 & \text{for } p = p_{(i-1),T} \\ \infty & \text{for other } p\text{'s} \end{cases}. \tag{9}$$

$$\frac{\partial^2 I_1}{\partial\mathbf{f}^2} = \frac{\partial\mathbf{r}}{\partial\mathbf{f}} = \frac{2}{\sigma_1+\sigma_2}\mathbf{t}\mathbf{t}^\top, \tag{12}$$

$$\mathcal{L}_{train} = \mathcal{L}_{map} + \frac{1}{9}\mathcal{L}_{render},$$

$$\mathcal{E}_l(p) = \omega_l(L)\sum_{q\in N_4(p)} \|\overline{F}_{G_i}^L(p) - \overline{F}_{G_j}^L(p)\|^2 + \|\overline{F}_{G_i}^L(q) - \overline{F}_{G_j}^L(q)\|^2, \tag{14}$$

$$\Delta\tau_S = \frac{1}{\tau_2-\tau_1}\int_{\tau_1}^{\tau_2}\frac{||\Lambda_{\mathbf{n}}||}{||\mathbf{B}_{\tau_{max}}||}d\tau. \tag{16}$$

$$r_0 = \frac{d}{2\tan(0.5\arctan(\frac{d}{2f}))},$$

$$\mathbb{T}_t = \exp(t\log A)\exp(t\log S)\exp(t\log E),$$

$$\tau(x',y',t) = \left(\frac{2}{tc}\right)^4\iiint_\Omega v(x',y',x,y,z)\,\rho(x,y,z)\,(\omega\cdot n(x,y,z))$$

$$\left(2\sqrt{(x'-x)^2+(y'-y)^2+z^2} - tc\right)\,dx\,dy\,dz, \tag{2}$$

$$E_{align}(h,h^{i-1},\mathcal{F}) = \sum_{c\in C(\mathcal{F})} Flat(c)\frac{A(c)}{A(\mathcal{F})}\left(H(c,h) - Snap(c,h^{i-1})\right)^2$$

$$c_e(S) = \left\|\mathbf{v}_m^{2D} - \mathbf{v}_{m+1}^{2D}\right\|^2 - \left\|\mathbf{v}_m^{2D,2} - \mathbf{v}_{m+1}^{2D,2}\right\|^2.$$

$$I = \mathrm{CNN}(I_p, I_{\mathrm{dx}}, I_{\mathrm{dy}}, F_1, F_2, \ldots;\boldsymbol{\theta})$$

$$a'_r = a_r\frac{T(\mathbf{x}_j,\mathbf{x}_{j+1})\sigma_s(\mathbf{x}_{j+1})}{p_{\mathrm{d}}(d\,|\,\mathbf{x}_j,\omega_j)},$$

$$W_s(f_0,f_1) = \min_{\mathbf{P}}\sum_{i,j}(x_i-y_j)^2P_{i,j} \tag{10}$$

$$\text{s.t.}\quad P_{i,j}\geq 0,\ \forall i,j \tag{11}$$

$$\sum_{j=1}^{m} P_{i,j} = 1,\ \forall i \tag{12}$$

$$\ldots \leq 1,\ \forall j \tag{13}$$

$$D[p_{\mathbf{q}}(\mathbf{z}|\mathbf{x})\parallel p_{\mathbf{z}}(\mathbf{z}|\mathbf{x})] = E_{\mathbf{z}\sim\mathbf{q}(\mathbf{x})}\left[\log p_{\mathbf{q}}(\mathbf{z}|\mathbf{x}) - \log p_{\mathbf{z}}(\mathbf{z}|\mathbf{x})\right]$$

$$\mathcal{L}_t(\mathcal{G},\mathcal{D}_t) = \log(\mathcal{D}_t(r_{i-L}^i,\Delta_{i,l}(\mathbf{f}))) + \log(1-\mathcal{D}_t(r_{i-L}^i,\Delta_{i,l}(\mathbf{o}))). \tag{8}$$

$$\mathcal{L}_{\ell_1}(\mathbf{G}) = \|\mathbf{Y} - \mathbf{G}(\mathbf{X})\|_1. \tag{3}$$

$$-\nabla\times(\nabla\times\mathbf{u}_\Omega(i)) + \nabla(\nabla\cdot\mathbf{u}_\Omega(i)) = \mathbf{0},$$

$$\frac{1}{||\mathbf{x}_{\mathrm{o}} - \boldsymbol{p}_i(\mathbf{x}_{\mathrm{o}})||_2^2\cdot|1+\boldsymbol{c}_i^\top\boldsymbol{d}_{\mathrm{sun}}|^2+\epsilon}, \tag{1}$$

$$d_k = \hat{d} - \gamma\Delta + \gamma\frac{2\Delta(k-1)}{p-1}.$$

$$\sum_{ij\in\eta_p}\omega_{ij} = -\Phi_0(\eta_p)$$

$$vi(\tau,v) = \frac{1}{\xi}vi_i(\tau,v)vi_r(\tau,v) \tag{7}$$

$$vi_i(\tau,v) = \exp(-\beta_i^F\max(\gamma-\beta_i^T,0°))$$

$$vi_r(\tau,v) = \exp(-\beta_r^F\max(px(\tau,v)-\beta_r^T,0px)),$$

$$\mathcal{L}_{\mathrm{match}} \equiv ||(G'(z_e)-x)\odot(1-\mathrm{mask}_e)||_1, \tag{3}$$

$$\tilde{\chi}(x) = \frac{1}{2} + \left(\frac{1}{2}-\frac{\alpha}{4}\right)\frac{d(x)}{w(x)} + \frac{\alpha d^3(x)}{4w^3(x)} + O(|w(x)|). \tag{6}$$

$$s\tilde{\mathbf{a}} = \alpha\tilde{\mathbf{p}} + \beta\tilde{\mathbf{q}} + (1-\alpha-\beta)\tilde{\mathbf{r}} \tag{6}$$

$$\hat{\mathbf{x}}(u) = \underset{\mathbf{x}\in\mathcal{M}}{\mathrm{argmin}}\sum_{i=0}^{N_a-1}B_i(u)d(\mathbf{x},\mathbf{c}_i^p)^2 \tag{4}$$

$$Lu(\mathbf{x}) = g(\mathbf{x}), \tag{1}$$

$$e(n) = (1-b(n))|x(n)| + b(n)e(n-1) \tag{13}$$

$$\mu_{\mathrm{source}} = \sum_{r=1}^{R}\delta(\mathcal{R}_r) \tag{1}$$

$$\text{RMSE-s}(\hat{\mathbf{I}}_t,\mathbf{I}_t) = \min_\alpha\left\|\alpha\hat{\mathbf{I}}_t - \mathbf{I}_t\right\|_2 \tag{5}$$

$$e^{-t\mathcal{S}}(x,x) = \sum_{i=0}^{\infty}e^{-t\lambda_i}\psi_i(x)^2 \sim \sum_{k=0}^{\infty}a_k(x)t^{k-2} + \sum_{l=1}^{\infty}b_l(x)t^l\log t, \tag{1}$$

$$\mathbf{W}^{-1}(\mathbf{x}) = \sum_i\mathbf{A}_i(\mathbf{x})a_i(\mathbf{x}), \tag{5}$$

$$\text{with}\quad \mathbf{A}_i(\mathbf{x}) = \mathbf{R}_i(\mathbf{s}_i\circ(\mathbf{x}-\mathbf{t}_i)),\qquad a_i(\mathbf{x}) = \frac{w_i(\mathbf{A}_i(\mathbf{x}))}{\sum_j w_j(\mathbf{A}_j(\mathbf{x}))}, \tag{6}$$

$$(\mathbf{p}_0^i)'_{xy} = \begin{cases} (\mathbf{p}_0^i)_{xy} & \text{if } i = 1 \\ (\mathbf{p}_0^{i-1})'_{xy} + \eta E(\boldsymbol{\theta}_r)(\Delta\mathbf{p}_0^{i-1})_{xy} & \text{otherwise} \end{cases},$$

$$(\Delta\mathbf{p}_0^i)'_{xy} = \eta E(\boldsymbol{\theta}_r)(\Delta\mathbf{p}_0^i)_{xy},$$

$$(\Delta\mathbf{q}_0^i)' = \exp(\boldsymbol{\theta}_r)\,\mathbf{q}_0^i.$$

$$E_{\mathrm{centering}} = \frac{||p-b_{v_1}|| + ||q-b_{v_2}||}{w_{v_1}+w_{v_2}}.$$

$$\gamma\sum_{i=1}^{r}A_1(p_i)||X_{12}(p_i,:) - X_2(q_i)||_{M_2}^2 + A_2(q_i)||X_{21}(q_i,:) - X_1(p_i)||_{M_1}^2.$$

$$\min_C\sum_{i=1}^{N}||x_i + (R_i - I)C||^2.$$

$$E_i = \sum_k\left\|\sum_{j\in f(p_t^k)}\omega_t^j\hat{v}_t^j - \sum_{j\in f(p_{t-1}^k)}\omega_{t-1}^j\hat{v}_{t-1}^j\right\|^2 \tag{5}$$

$$\hat{R}_i = \mathbf{F}_\phi(H_i;i),\ \forall i\in C. \tag{4}$$

$$I_{pred}(i,j) = \begin{cases} 1, & w_{ij}\geq thres \\ 0, & w_{ij} < thres \end{cases} \tag{7}$$

$$\ell_b = \min(\ell_{\max},\beta\cdot a_b), \tag{9}$$

$$\varphi_{\mathrm{CR}}(t) = \begin{cases} \frac{3}{2}|t|^3 - \frac{5}{2}|t|^2 + 1 \text{ if } |t|\leq 1 \\ -\frac{1}{2}|t|^3 + \frac{5}{2}|t|^2 - 4|t| + 2 \text{ if } 1 < |t|\leq 2 \\ 0 \text{ otherwise.} \end{cases} \tag{6}$$

$$\begin{aligned}\min_{\mathbf{f},\mathbf{u}}\ &\sum_{j=1}^{N_p}\left\|\mathbf{A}_j\mathbf{W}_j^f(\mathbf{f}_{j^-}) - \mathbf{p}_j\right\|_2^2 + \sum_{j=1}^{N_p}\left\|\mathbf{A}_j\mathbf{W}_j^b(\mathbf{f}_{j^+}) - \mathbf{p}_j\right\|_2^2\\ &+\kappa_1\sum_{k=1}^{N_k-1}\|\nabla_T\mathbf{f}_k + \nabla_S\mathbf{f}_k\cdot\mathbf{u}_k\|_1\\ &+\sum_{k=1}^{N_k}\left[\kappa_2\|\nabla_S\mathbf{f}_k\|_{\mathrm{H}_\epsilon} + \kappa_3\|\nabla_T\mathbf{f}_k\|_2^2\right]\\ &+\sum_{k=1}^{N_k-1}\sum_{i=x,y,z}\left[\kappa_4\|\nabla_S\mathbf{u}_{k,i}\|_{\mathrm{H}_\tau} + \kappa_5\|\nabla_T\mathbf{u}_{k,i}\|_2^2\right],\end{aligned} \tag{7}$$

$$\underset{i}{\arg\max}\ \frac{L(\mathbf{f}_c,\mathbf{f}_{c_i})}{M} + \frac{[H(\mathbf{B}_s\odot\mathbf{g}_s,\mathbf{B}_s\odot\mathbf{g}_{l_i}) + H(\mathbf{B}_{l_i}\odot\mathbf{g}_s,\mathbf{B}_{l_i}\odot\mathbf{g}_{l_i})]}{2N}, \tag{8}$$

$$\begin{aligned} JMJ &= (I-KN^T)J\\ &= J - KN^TJ\\ &= J - K(JN)^T\\ &= J\end{aligned}$$

$$0\leq\begin{pmatrix}\mathbf{A} & \mathbf{J}^T & \mathbf{J}_t^T\\ \mathbf{J} & 0 & 0\\ \mathbf{J}_t & 0 & 0\end{pmatrix}\begin{pmatrix}\mathbf{v}\\ \boldsymbol{\lambda}\\ \boldsymbol{\gamma}\end{pmatrix} - \begin{pmatrix}\mathbf{b} - \mathbf{J}_t^T\boldsymbol{\gamma}'\\ \mathbf{c}_n\\ \mathbf{c}_t\end{pmatrix}\perp\begin{pmatrix}\mathbf{1}\\ \boldsymbol{\lambda}\\ \boldsymbol{\gamma}\end{pmatrix}\geq\mathbf{0}, \tag{20}$$

$$\left(\frac{4l_0M_c(1-r)\Psi^+}{\mathcal{G}} + M_c + \frac{1}{\Delta t}\right)c^{n+1} - (4l_0^2M_c)\nabla^2c^{n+1} = r, \tag{6}$$

$$\begin{cases} Z_1 = H_1^{0,0}\cup H_1^{0,1}\cup H_1^{0,2}\\ H_1^{0,0} = \{f\in\mathbb{R}^3\mid n_1^{0,0}\cdot f = 0 \text{ and } M_1^{0,0}f > 0\}\\ H_1^{0,1} = \{f\in\mathbb{R}^3\mid n_1^{0,1}\cdot f = 0 \text{ and } M_1^{0,1}f > 0\}\\ H_1^{0,2} = \{f\in\mathbb{R}^3\mid n_1^{0,2}\cdot f = 0 \text{ and } M_1^{0,2}f > 0\},\end{cases} \tag{8}$$

with $n_1^{0,0} = (1,0,0)$, $n_1^{0,1} = (w_1,w_2,0)$, $n_1^{0,2} = (w_1,w_2,w_3)$, and

$$M_1^{0,0} = \begin{pmatrix}0 & -1 & 0\\ 0 & -w_2 & -w_3\end{pmatrix},\ M_1^{0,1} = \begin{pmatrix}0 & 1 & 0\\ 0 & 0 & -1\end{pmatrix},\ M_1^{0,2} = \begin{pmatrix}0 & w_2 & w_3\\ 0 & 0 & 1\end{pmatrix}.$$

$$\mathcal{L} = \mathcal{L}_{TL} + \mathcal{L}_P$$

$$\begin{tikzcd} \mathbf{f} \arrow[r, "M^{-1}L"] \arrow[d, "P"'] & \bullet \arrow[d, "P"] \\ \bullet \arrow[r, "\widetilde{M}^{-1}\widetilde{L}"'] & \widetilde{\mathbf{f}} \end{tikzcd} \tag{2}$$

$$\widetilde{G} := -\int_{\mathbb{S}^1}\langle\gamma,g\rangle dm \tag{11}$$

$$f_{\mathrm{ir}}^{\mathrm{eff}}(\omega_i,\omega_o;\mathcal{G},f)A_{\mathcal{G}}(\omega_o) = f_{\mathrm{ir}}^{\mathrm{eff}}(\omega_o,\omega_i;\mathcal{G},f)A_{\mathcal{G}}(\omega_i). \tag{25}$$

$$\nabla_\theta D_{\mathrm{KL}}(p\parallel q;\theta) = -\nabla_\theta\int_\Omega p(x)\log q(x;\theta)\,\mathrm{d}x \tag{22}$$

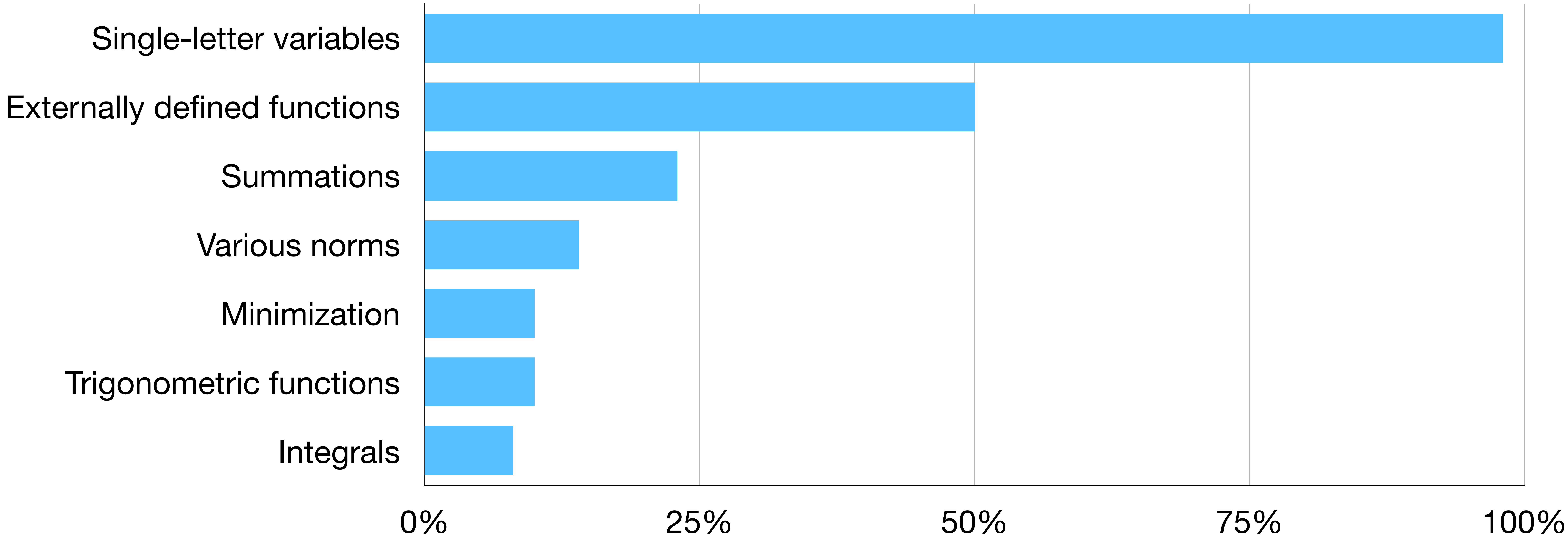
$$= \mathbb{E}\left[-\frac{p(X)}{q(X;\theta)}\nabla_\theta\log q(X;\theta)\right], \tag{23}$$

$$X^*,\theta^* = \underset{X,\theta}{\arg\max}[\log(P(Y|X,\theta))].$$

$$V_p^0\frac{\partial\Psi_{\star,p}}{\partial\mathbf{F}_{\star,p}}(\mathcal{Z}_\star(\mathbf{F}_{\star,p}^{\mathrm{tr}})) + \rho\mathbf{W}_{\star,p}^T\mathbf{W}_{\star,p}\mathbf{F}_{\star,p}^{\mathrm{tr}} = \mathcal{R}_{\star,p}$$

where $\mathcal{R}_{\star,p} = \rho\mathbf{W}_{\star,p}^T\mathbf{W}_{\star,p}(\mathbf{D}\delta\boldsymbol{v}^n + \boldsymbol{b})_p - \mathbf{W}_{\star,p}^T\boldsymbol{y}_{\star,p}^n.$

Analysis of all 1987 Equations at SIGGRAPH 2019



Single-letter variables (98%)

Single-letter variables (98%)

$$\mathcal{P}^* = \underset{\{\mathcal{P}\}}{\arg\min} \sum_{i=1}^{N} \mathcal{L}_{\text{TASK}}\left(f_{\text{ISP}}(\mathbf{I}_i; \mathcal{P}), \mathbf{T}_i\right) , \tag{1}$$

$$\mathbf{s} = (\bar{\boldsymbol{\phi}}^T, \boldsymbol{\omega}^T, \mathbf{D}^T, \mathbf{I}^T)^T, \tag{7}$$

$$C_t = \frac{\sum_k w_{t,k} \alpha_{t,k} C_{t,k}}{\sum_k w_{t,k} \alpha_{t,k}} . \tag{8}$$

$$\overline{T}(\mathbf{x}, \mathbf{y}) = e^{-\overline{\tau}(\mathbf{x},\mathbf{y})} = e^{-\int_0^y \overline{\mu}_{\mathrm{t}}(\mathbf{x} - s\omega)\, \mathrm{d}s} \tag{10}$$

$$f_{\mathcal{M}}(\mathbf{x}) = \sum_i w_i f(\mathbf{x}|\Theta_i) \tag{1}$$

$$\vec{p}_i(t) = f(\vec{q}(t), \vec{r}_i, \ell) \tag{1}$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{R}_z(-\psi) & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\psi} \end{bmatrix} - \begin{bmatrix} \mathbf{u} \\ \psi \end{bmatrix} \right). \tag{8}$$

$$\mathbf{L}(\theta, \phi) = \sum_{t \in \mathcal{T}} \left(\sum_{i \in C} \left\| \hat{R}_i^t - R_i^t \right\|_1 + \lambda \delta(z^t) \right), \tag{5}$$

Single-letter variables (98%)

$$\mathcal{P}^* = \underset{\{\mathcal{P}\}}{\arg\min} \sum_{i=1}^{N} \mathcal{L}_{\text{TASK}}\left(f_{\text{ISP}}(\mathbf{I}_i; \mathcal{P}), \mathbf{T}_i\right), \quad (1)$$

$$C_t = \frac{\sum_k w_{t,k}\alpha_{t,k}C_{t,k}}{\sum_k w_{t,k}\alpha_{t,k}}. \quad (8)$$

$$f_{\mathcal{M}}(\mathbf{x}) = \sum_i w_i f(\mathbf{x}|\Theta_i) \quad (1)$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{R}_z(-\psi) & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\psi} \end{bmatrix} - \begin{bmatrix} \mathbf{u} \\ \psi \end{bmatrix} \right), \quad (8)$$

$$\mathbf{s} = (\bar{\boldsymbol{\phi}}^T, \boldsymbol{\omega}^T, \mathbf{D}^T, \mathbf{I}^T)^T, \quad (7)$$

$$\overline{T}(\mathbf{x}, \mathbf{y}) = e^{-\overline{\tau}(\mathbf{x},\mathbf{y})} = e^{-\int_0^y \overline{\mu}_t(\mathbf{x}+s\omega)\,\mathrm{d}s} \quad (10)$$

$$\vec{p}_i(t) = f(\vec{q}(t), \vec{r}_i, \ell) \quad (1)$$

$$\mathbf{L}(\theta, \phi) = \sum_{t \in \mathcal{T}} \left(\sum_{i \in C} \|\hat{R}_i^t - R_i^t\|_1 + \lambda\delta(z^t) \right), \quad (5)$$

Single-letter variables (98%)

$$\mathcal{P}^* = \underset{\{\mathcal{P}\}}{\arg\min} \sum_{i=1}^{N} \mathcal{L}_{\text{TASK}}\left(f_{\text{ISP}}(\mathbf{I}_i;\mathcal{P}), \mathbf{T}_i\right) , \tag{1}$$

$$C_t = \frac{\sum_k w_{t,k}\alpha_{t,k}C_{t,k}}{\sum_k w_{t,k}\alpha_{t,k}} . \tag{8}$$

$$f_{\mathcal{M}}(\mathbf{x}) = \sum_i w_i f(\mathbf{x}|\Theta_i) \tag{1}$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{R}_z(-\psi) & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\psi} \end{bmatrix} - \begin{bmatrix} \mathbf{u} \\ \psi \end{bmatrix} \right) , \tag{8}$$

$$\mathbf{s} = (\bar{\phi}^T, \boldsymbol{\omega}^T, \mathbf{D}^T, \mathbf{I}^T)^T , \tag{7}$$

$$\overline{T}(\mathbf{x}, \mathbf{y}) = e^{-\overline{\tau}(\mathbf{x}, \mathbf{y})} = e^{-\int_0^y \overline{\mu}_t(\mathbf{x} - s\omega)\, ds} \tag{10}$$

$$\vec{p}_i(t) = f(\vec{q}(t), \vec{r}_i, \ell) \tag{1}$$

$$\mathbf{L}(\theta, \phi) = \sum_{t \in \mathcal{T}} \left(\sum_{i \in C} \|\hat{R}_i^t - R_i^t\|_1 + \lambda\delta(z^t) \right) , \tag{5}$$

Variables in I❤️LA

Variables in I❤️LA

```
ω = ABC
d̂ = xᵀωᵀωx

where
A ∈ ℝ^(3×n)
B ∈ ℝ^(n×m)
C ∈ ℝ^(m×2)
x ∈ ℝ²
```

Variables in I❤️LA

- Single-letter identifiers are encouraged

```
ω = ABC
d̂ = xᵀωᵀωx

where
A ∈ ℝ^(3×n)
B ∈ ℝ^(n×m)
C ∈ ℝ^(m×2)
x ∈ ℝ²
```

Variables in I❤️LA

- Single-letter identifiers are encouraged

- Juxtaposition is multiplication

```
ω = ABC
d̂ = xᵀωᵀωx

where
A ∈ ℝ^(3×n)
B ∈ ℝ^(n×m)
C ∈ ℝ^(m×2)
x ∈ ℝ²
```

Variables in I❤️LA

- Single-letter identifiers are encouraged

- Juxtaposition is multiplication

- Unicode

```
ω = ABC
d̂ = xᵀωᵀωx

where
A ∈ ℝ^(3×n)
B ∈ ℝ^(n×m)
C ∈ ℝ^(m×2)
x ∈ ℝ²
```

Variables in I❤️LA

- Single-letter identifiers are encouraged
- Juxtaposition is multiplication
- Unicode
- Variables cannot be re-defined

```
ω = ABC
d̂ = xᵀωᵀωx

where
A ∈ ℝ^(3×n)
B ∈ ℝ^(n×m)
C ∈ ℝ^(m×2)
x ∈ ℝ²
```

Variables in I❤️LA

- Single-letter identifiers are encouraged
- Juxtaposition is multiplication
- Unicode
- Variables cannot be re-defined
- Compatible matrix and vector dimensions are statically checked (compile-time, not run-time).

```
ω = ABC
d̂ = xᵀωᵀωx

where
A ∈ ℝ^(3×n)
B ∈ ℝ^(n×m)
C ∈ ℝ^(m×2)
x ∈ ℝ²
```

Matrices in I❤️LA

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
      Mᵀ 0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
     Mᵀ 0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
      3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
     Mᵀ  0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I  M+yxᵀ
     Mᵀ 0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I  M+yxᵀ
    Mᵀ 0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
        3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
     Mᵀ  0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
      3  k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
     Mᵀ  0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I  Mᵀ
     M  0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
      Mᵀ 0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

- Element-wise:

```
L_ij = M_ij + 7y_i
where
M ∈ ℝ^(m×n)
y ∈ ℝ^m
```

```
L_ij = { 1 if (i,j) ∈ E
         0 otherwise
L_ii = -∑_(j for j != i) L_i,j
where
E ∈ { ℤ×ℤ }
L ∈ ℝ^(n×n)
n ∈ ℤ
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
     3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
     Mᵀ  0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

- Element-wise:

```
L_ij = M_ij + 7y_i
where
M ∈ ℝ^(m×n)
y ∈ ℝ^m
```

```
L_ij = { 1 if (i,j) ∈ E
         0 otherwise
L_ii = -∑_(j for j != i) L_i,j
where
E ∈ { ℤ×ℤ }
L ∈ ℝ^(n×n)
n ∈ ℤ
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3  k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
     Mᵀ  0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

- Element-wise:

```
L_ij = M_ij + 7y_i
where
M ∈ ℝ^(m×n)
y ∈ ℝ^m
```

```
L_ij = { 1 if (i,j) ∈ E
         0 otherwise
L_ii = -∑_(j for j != i) L_i,j
where
E ∈ { ℤ×ℤ }
L ∈ ℝ^(n×n)
n ∈ ℤ
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
       3 k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
      Mᵀ 0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

- Element-wise:

```
L_ij = M_ij + 7y_i
where
M ∈ ℝ^(m×n)
y ∈ ℝ^m
```

```
L_ij = { 1 if (i,j) ∈ E
         0 otherwise
L_ii = -∑_(j for j != i) L_i,j
where
E ∈ { ℤ×ℤ }
L ∈ ℝ^(n×n)
n ∈ ℤ
```

Matrices in I❤️LA

- 2D matrix definitions:

```
L = [2a  0
      3  k+1]
where
a ∈ ℝ
k ∈ ℝ
```

```
L = [I   M+yxᵀ
     Mᵀ  0]
where
M ∈ ℝ^(m×n)
x ∈ ℝ^n
y ∈ ℝ^m
```

- Element-wise:

```
L_ij = M_ij + 7y_i
where
M ∈ ℝ^(m×n)
y ∈ ℝ^m
```

```
L_ij = { 1 if (i,j) ∈ E
         0 otherwise
L_ii = -∑_(j for j != i) L_i,j
where
E ∈ { ℤ×ℤ }
L ∈ ℝ^(n×n)
n ∈ ℤ
```

Externally defined functions (50%)

Externally defined functions (50%)

$$\mathbf{g} = \frac{\partial \mathbf{F}}{\partial \mathbf{u}}^T : \mathbf{P}(\mathbf{F}). \quad (1)$$

$$\mathcal{L}_{\text{relight}} = \mathbb{E}\left[w_2 \cdot \mathcal{P}(I^{\text{R}}, I^{\star})\right]. \quad (5)$$

$$E(x, x^t, v^t) = \frac{1}{2} x^T M x - x^T M x^p + h^2 W(x). \quad (3)$$

$$E_{E'} = \sum_{i \in I'} \sum_{f \in F_i} \frac{A(f)}{A'} \max(0, n_f^i \cdot d_i) \quad (2)$$

$$K_{ij} = \sum_{\widehat{C} \in \widehat{\mathcal{M}}} \int_{\mathbf{g}(\widehat{C})} \nabla \phi_i(\mathbf{x}) \cdot \nabla \phi_j(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \quad (1)$$

$$\varphi_{\mathcal{G} \to \mathcal{S}} = \max_{\mathbf{g} \in \mathcal{G}} \left[\min_{\mathbf{s} \in \mathcal{S}} \phi(\mathbf{s}, \mathbf{g})\right]. \quad (8)$$

$$M_i = \mathcal{T}(I_i). \quad (3)$$

$$\tilde{\chi}(x) = \frac{1}{2} + \left(\frac{1}{2} - \frac{\alpha}{4}\right) \frac{d(x)}{w(x)} + \frac{\alpha d^3(x)}{4 w^3(x)} + O(|w(x)|). \quad (6)$$

Externally defined functions (50%)

$$\mathbf{g} = \frac{\partial \mathbf{F}}{\partial \mathbf{u}}^{T} : \boxed{\mathbf{P}(\mathbf{F})} \qquad (1)$$

$$\mathcal{L}_{\text{relight}} = \mathbb{E}\left[w_2 \cdot \mathcal{P}(I^{\mathrm{R}}, I^{\star})\right]. \qquad (5)$$

$$E(x, x^t, v^t) = \frac{1}{2} x^T M x - x^T M x^p + h^2 W(x). \qquad (3)$$

$$E_{E'} = \sum_{i \in I'} \sum_{f \in F_i} \frac{A(f)}{A'} \max(0, n_f^i \cdot d_i) \qquad (2)$$

$$K_{ij} = \sum_{\widehat{C} \in \widehat{\mathcal{M}}} \int_{\mathbf{g}(\widehat{C})} \nabla \phi_i(\mathbf{x}) \cdot \nabla \phi_j(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \qquad (1)$$

$$\varphi_{\mathcal{G} \to \mathcal{S}} = \max_{\mathbf{g} \in \mathcal{G}} \left[\min_{\mathbf{s} \in \mathcal{S}} \phi(\mathbf{s}, \mathbf{g})\right]. \qquad (8)$$

$$M_i = \mathcal{T}(I_i). \qquad (3)$$

$$\tilde{\chi}(x) = \frac{1}{2} + \left(\frac{1}{2} - \frac{\alpha}{4}\right) \frac{d(x)}{w(x)} + \frac{\alpha d^3(x)}{4 w^3(x)} + O(|w(x)|). \qquad (6)$$

Externally defined functions (50%)

$$\mathbf{g} = \frac{\partial \mathbf{F}}{\partial \mathbf{u}}^{T} : \boxed{\mathbf{P}(\mathbf{F})} \qquad (1)$$

$$E(x, x^t, v^t) = \frac{1}{2} x^T M x - x^T M x^p + h^2 \boxed{W(x)}. \qquad (3)$$

$$K_{ij} = \sum_{\widehat{C} \in \widehat{M}} \int_{\mathbf{g}(\widehat{C})} \nabla \phi_i(\mathbf{x}) \cdot \nabla \phi_j(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \qquad (1)$$

$$M_i = \mathcal{T}(I_i). \qquad (3)$$

$$\mathcal{L}_{\mathrm{relight}} = \mathbb{E}\left[w_2 \cdot \mathcal{P}(I^{\mathrm{R}}, I^{\star}) \right]. \qquad (5)$$

$$E_{E'} = \sum_{i \in I'} \sum_{f \in F_i} \frac{A(f)}{A'} \max(0, n_f^i \cdot d_i) \qquad (2)$$

$$\varphi_{\mathcal{G} \to \mathcal{S}} = \max_{\mathbf{g} \in \mathcal{G}} \left[\min_{\mathbf{s} \in \mathcal{S}} \phi(\mathbf{s}, \mathbf{g}) \right]. \qquad (8)$$

$$\tilde{\chi}(x) = \frac{1}{2} + \left(\frac{1}{2} - \frac{\alpha}{4} \right) \frac{d(x)}{w(x)} + \frac{\alpha d^3(x)}{4 w^3(x)} + O(|w(x)|). \qquad (6)$$

Externally defined functions (50%)

$$\mathbf{g} = \frac{\partial \mathbf{F}}{\partial \mathbf{u}}^T : \boxed{\mathbf{P}(\mathbf{F})} \quad (1)$$

$$\mathcal{L}_{\text{relight}} = \mathbb{E}\left[w_2 \cdot \boxed{\mathcal{P}(I^{\mathrm{R}}, I^{\star})}\right]. \quad (5)$$

$$E(x, x^t, v^t) = \frac{1}{2} x^T M x - x^T M x^p + h^2 \boxed{W(x)}. \quad (3)$$

$$E_{E'} = \sum_{i \in I'} \sum_{f \in F_i} \frac{\boxed{A(f)}}{A'} \max(0, n_f^i \cdot d_i) \quad (2)$$

$$K_{ij} = \sum_{\widehat{C} \in \widehat{M}} \int_{\mathbf{g}(\widehat{C})} \nabla \phi_i(\mathbf{x}) \cdot \nabla \boxed{\phi_j(\mathbf{x})} \, \mathrm{d}\mathbf{x}, \quad (1)$$

$$\varphi_{\mathcal{G} \to \mathcal{S}} = \max_{\mathbf{g} \in \mathcal{G}} \left[\min_{\mathbf{s} \in \mathcal{S}} \boxed{\phi(\mathbf{s}, \mathbf{g})}\right]. \quad (8)$$

$$M_i = \boxed{\mathcal{T}(I_i)}. \quad (3)$$

$$\tilde{\chi}(x) = \frac{1}{2} + \left(\frac{1}{2} - \frac{\alpha}{4}\right) \boxed{\frac{d(x)}{w(x)}} + \frac{\alpha d^3(x)}{4 w^3(x)} + O(|w(x)|). \quad (6)$$

Trigonometric functions (10%)

Trigonometric functions (10%)

$$p(\boldsymbol{x}, t) = c_i(\boldsymbol{x}) \cos \omega_i t + d_i(\boldsymbol{x}) \sin \omega_i t, \quad (5)$$

$$= \sqrt{c_i^2(\boldsymbol{x}) + d_i^2(\boldsymbol{x})} \cos(\omega_i t + \varphi_i(\boldsymbol{x})) \quad (6)$$

$$E_M(\mathbf{R}, \mathbf{t}) = \left\| \arcsin\left(\frac{\mathbf{q}^\top \mathbf{t}_\times \mathbf{R}\mathbf{p}}{\|\mathbf{t}_\times \mathbf{R}\mathbf{p}\|} \right) \right\|_2 \quad (4)$$

$$B_i = \begin{bmatrix} \cos(\omega_i T) & \sin(\omega_i T) \\ -\sin(\omega_i T) & \cos(\omega_i T) \end{bmatrix}. \quad (15)$$

$$W_M D_I = W_N^M, \text{ where}$$

$$(W_N^M)_{i,j} = \frac{1}{2} \left(\frac{\mu^M(T_\alpha)}{\mu^N(T_\alpha)} \cot \alpha_{ij}^N + \frac{\mu^M(T_\beta)}{\mu^N(T_\beta)} \cot \beta_{ij}^N \right). \quad (9)$$

$$p_I(r, t) = \frac{k^2 \rho c D_S(t)}{4\pi r} cos\theta e^{-ikr} \quad (1)$$

$$c_p(\alpha, \epsilon) = \frac{K_p e \sec^2(\alpha + \epsilon)}{d_p \sec^2(\phi_{\alpha,\epsilon})} \quad (10)$$

$$h(\varphi) = \frac{1}{\pi} \arctan\left(\Re\lambda_0 + 2\Re \sum_{l=1}^{m} \lambda_l \exp(-il\varphi) \right) + \frac{1}{2}. \quad (11)$$

$$\alpha_{max} = \tan^{-1}\left(\frac{M w_m}{2d_f + D_e} \right), \quad (4)$$

Trigonometric functions (10%)

$$p(\boldsymbol{x}, t) = c_i(\boldsymbol{x}) \cos \omega_i t + d_i(\boldsymbol{x}) \boxed{\sin \omega_i t} \quad (5)$$

$$= \sqrt{c_i^2(\boldsymbol{x}) + d_i^2(\boldsymbol{x})} \cos(\omega_i t + \varphi_i(\boldsymbol{x})) \quad (6)$$

$$E_M(\mathbf{R}, \mathbf{t}) = \left\| \arcsin\left(\frac{\mathbf{q}^\top \mathbf{t}_\times \mathbf{R}\mathbf{p}}{\|\mathbf{t}_\times \mathbf{R}\mathbf{p}\|} \right) \right\|_2 \quad (4)$$

$$B_i = \begin{bmatrix} \cos(\omega_i T) & \boxed{\sin(\omega_i T)} \\ -\sin(\omega_i T) & \cos(\omega_i T) \end{bmatrix}. \quad (15)$$

$$W_M D_I = W_N^M, \text{ where}$$

$$(W_N^M)_{i,j} = \frac{1}{2}\left(\frac{\mu^M(T_\alpha)}{\mu^N(T_\alpha)} \cot \alpha_{ij}^N + \frac{\mu^M(T_\beta)}{\mu^N(T_\beta)} \cot \beta_{ij}^N \right). \quad (9)$$

$$p_I(r, t) = \frac{k^2 \rho c D_S(t)}{4\pi r} cos\theta e^{-ikr} \quad (1)$$

$$c_p(\alpha, \epsilon) = \frac{K_p e \sec^2(\alpha + \epsilon)}{d_p \sec^2(\phi_{\alpha,\epsilon})} \quad (10)$$

$$h(\varphi) = \frac{1}{\pi} \arctan\left(\Re\lambda_0 + 2\Re \sum_{l=1}^{m} \lambda_l \exp(-il\varphi) \right) + \frac{1}{2}. \quad (11)$$

$$\alpha_{max} = \tan^{-1}\left(\frac{M w_m}{2d_f + D_e} \right), \quad (4)$$

Trigonometric functions (10%)

$$p(\boldsymbol{x}, t) = c_i(\boldsymbol{x}) \boxed{\cos \omega_i t} + d_i(\boldsymbol{x}) \boxed{\sin \omega_i t} \tag{5}$$

$$= \sqrt{c_i^2(\boldsymbol{x}) + d_i^2(\boldsymbol{x})} \cos(\omega_i t + \varphi_i(\boldsymbol{x})) \tag{6}$$

$$E_M(\mathbf{R}, \mathbf{t}) = \left\| \arcsin \left(\frac{\mathbf{q}^\top \mathbf{t}_\times \mathbf{R}\mathbf{p}}{\|\mathbf{t}_\times \mathbf{R}\mathbf{p}\|} \right) \right\|_2 \tag{4}$$

$$B_i = \begin{bmatrix} \boxed{\cos(\omega_i T)} & \boxed{\sin(\omega_i T)} \\ -\sin(\omega_i T) & \cos(\omega_i T) \end{bmatrix}. \tag{15}$$

$$W_M D_I = W_N^M, \text{ where}$$

$$(W_N^M)_{i,j} = \frac{1}{2} \left(\frac{\mu^M(T_\alpha)}{\mu^N(T_\alpha)} \cot \alpha_{ij}^N + \frac{\mu^M(T_\beta)}{\mu^N(T_\beta)} \cot \beta_{ij}^N \right). \tag{9}$$

$$p_I(r, t) = \frac{k^2 \rho c D_S(t)}{4\pi r} \boxed{\cos\theta} e^{-ikr} \tag{1}$$

$$c_p(\alpha, \epsilon) = \frac{K_p e \sec^2(\alpha + \epsilon)}{d_p \sec^2(\phi_{\alpha,\epsilon})} \tag{10}$$

$$h(\varphi) = \frac{1}{\pi} \arctan \left(\Re \lambda_0 + 2\Re \sum_{l=1}^{m} \lambda_l \exp(-il\varphi) \right) + \frac{1}{2}. \tag{11}$$

$$\alpha_{max} = \tan^{-1} \left(\frac{M w_m}{2d_f + D_e} \right), \tag{4}$$

Trigonometric functions (10%)

$$p(\boldsymbol{x}, t) = c_i(\boldsymbol{x}) \boxed{\cos \omega_i t} + d_i(\boldsymbol{x}) \boxed{\sin \omega_i t} \quad (5)$$

$$= \sqrt{c_i^2(\boldsymbol{x}) + d_i^2(\boldsymbol{x})} \, \cos(\omega_i t + \varphi_i(\boldsymbol{x})) \quad (6)$$

$$E_M(\mathbf{R}, \mathbf{t}) = \left\| \boxed{\arcsin} \left(\frac{\mathbf{q}^\top \mathbf{t}_\times \mathbf{R} \mathbf{p}}{\| \mathbf{t}_\times \mathbf{R} \mathbf{p} \|} \right) \right\|_2 \quad (4)$$

$$B_i = \begin{bmatrix} \boxed{\cos(\omega_i T)} & \boxed{\sin(\omega_i T)} \\ -\sin(\omega_i T) & \cos(\omega_i T) \end{bmatrix}. \quad (15)$$

$$W_M D_I = W_N^M, \text{ where}$$

$$(W_N^M)_{i,j} = \frac{1}{2} \left(\frac{\mu^M(T_\alpha)}{\mu^N(T_\alpha)} \boxed{\cot \alpha_{ij}^N} + \frac{\mu^M(T_\beta)}{\mu^N(T_\beta)} \cot \beta_{ij}^N \right). \quad (9)$$

$$p_I(r, t) = \frac{k^2 \rho c D_S(t)}{4 \pi r} \boxed{\cos\theta} e^{-ikr} \quad (1)$$

$$c_p(\alpha, \epsilon) = \frac{K_p e \boxed{\sec^2} (\alpha + \epsilon)}{d_p \sec^2(\phi_{\alpha, \epsilon})} \quad (10)$$

$$h(\varphi) = \frac{1}{\pi} \boxed{\arctan} \left(\Re \lambda_0 + 2 \Re \sum_{l=1}^{m} \lambda_l \exp(-il\varphi) \right) + \frac{1}{2}. \quad (11)$$

$$\alpha_{max} = \boxed{\tan^{-1}} \left(\frac{M w_m}{2 d_f + D_e} \right), \quad (4)$$

Functions in I❤️LA

Functions in I❤️LA

- Externally defined functions

```
a = f(2) + g()
where
f ∈ ℝ → ℝ^(3x3)
g ∈ ∅ → ℝ^(3x3)
```

Functions in I❤️LA

- Externally defined functions

```
a = f(2) + g()
where
f ∈ ℝ → ℝ^(3x3)
g ∈ ∅ → ℝ^(3x3)
```

- Built-in Functions
  - Directly used
  - Import from built-in libraries: **trigonometric** and **linearalgebra**

```
from trigonometry: sin
c = sin(a) + exp(b)
where
a ∈ ℝ
b ∈ ℝ
```

Functions in I❤️LA

- Externally defined functions

```
a = f(2) + g()
where
f ∈ ℝ → ℝ^(3x3)
g ∈ ∅ → ℝ^(3x3)
```

- Built-in Functions
  - Directly used
  - Import from built-in libraries: **trigonometric** and **linearalgebra**

```
from trigonometry: sin
c = sin(a) + exp(b)
where
a ∈ ℝ
b ∈ ℝ
```

Summation (23%)

Summation (23%)

$$k_t(x,y) := \frac{e^{-d(x,y)^2/4t}}{(4\pi t)^{n/2}} j(x,y)^{-1/2} \left(1 + \sum_{i=1}^{\infty} t^i \Phi_i(x,y)\right). \quad (3)$$

$$\varphi_{ij_a} := \sum_{p=0}^{a-1} \tilde{\theta}_i^{j_p, j_{p+1}}. \quad (8)$$

$$W_p(f_0, f_1) = \min_{\mathbf{P}} \sum_{i,j} c(x_i, y_j) P_{i,j} \quad (5)$$

$$e \cdot k_0 + \sum_{j=1}^{4} k_j \cdot e^j, \quad (1)$$

$$L_s = \sum_t \left\| q_\lambda^t - q_\lambda^{t-1} \right\|_2^2, \quad (7)$$

$$L_r = \sum_t \sum_k \left\| \Pi q_k^t - u_k^t \right\|_2^2 c_k^t, \quad (5)$$

$$R = \sum_{i=1}^{k} ||h_i' - h_i||_2^2 + \sum_{i=1}^{k} \sum_{j=i+1}^{k} ||h_{i,j}' - h_{i,j}||_2^2, \quad (9)$$

$$\hat{\boldsymbol{\alpha}}_p = \sum_i w_{pi} \hat{\boldsymbol{\alpha}}_i. \quad (44)$$

Summation (23%)

$$k_t(x,y) := \frac{e^{-d(x,y)^2/4t}}{(4\pi t)^{n/2}} j(x,y)^{-1/2} \left(1 + \sum_{i=1}^{\infty} t^i \Phi_i(x,y)\right). \quad (3)$$

$$L_s = \boxed{\sum_t} \|q_\lambda^t - q_\lambda^{t-1}\|_2^2, \quad (7)$$

$$\varphi_{ij_a} := \sum_{p=0}^{a-1} \tilde{\theta}_i^{j_p, j_{p+1}}. \quad (8)$$

$$L_r = \boxed{\sum_t \sum_k} \|\Pi q_k^t - u_k^t\|_2^2 \, c_k^t, \quad (5)$$

$$W_p(f_0, f_1) = \min_{\mathbf{P}} \boxed{\sum_{i,j}} c(x_i, y_j) P_{i,j} \quad (5)$$

$$R = \sum_{i=1}^{k} ||h_i' - h_i||_2^2 + \sum_{i=1}^{k} \sum_{j=i+1}^{k} ||h_{i,j}' - h_{i,j}||_2^2, \quad (9)$$

$$e \cdot k_0 + \sum_{j=1}^{4} k_j \cdot e^j, \quad (1)$$

$$\hat{\alpha}_p = \boxed{\sum_i} w_{pi} \hat{\alpha}_i. \quad (44)$$

Summation (23%)

$$k_t(x,y) := \frac{e^{-d(x,y)^2/4t}}{(4\pi t)^{n/2}} j(x,y)^{-1/2} \left(1 + \sum_{i=1}^{\infty} t^i \Phi_i(x,y)\right). \qquad (3)$$

$$\varphi_{ij_a} := \sum_{p=0}^{a-1} \tilde{\theta}_i^{j_p, j_{p+1}}. \qquad (8)$$

$$W_p(f_0, f_1) = \min_{\mathbf{P}} \boxed{\sum_{i,j}} c(x_i, y_j) P_{i,j} \qquad (5)$$

$$e \cdot k_0 + \sum_{j=1}^{4} k_j \cdot e^j, \qquad (1)$$

$$L_s = \boxed{\sum_t} \|q_\lambda^t - q_\lambda^{t-1}\|_2^2, \qquad (7)$$

$$L_r = \boxed{\sum_t \sum_k} \|\Pi q_k^t - u_k^t\|_2^2 \, c_k^t, \qquad (5)$$

$$R = \sum_{i=1}^{k} ||h_i' - h_i||_2^2 + \boxed{\sum_{i=1}^{k} \sum_{j=i+1}^{k}} ||h_{i,j}' - h_{i,j}||_2^2, \qquad (9)$$

$$\hat{\boldsymbol{\alpha}}_p = \boxed{\sum_i} w_{pi} \hat{\boldsymbol{\alpha}}_i. \qquad (44)$$

Summation (23%)

$$k_t(x,y) := \boxed{\frac{e^{-d(x,y)^2/4t}}{(4\pi t)^{n/2}} j(x,y)^{-1/2}\left(1 + \sum_{i=1}^{\infty} t^i \Phi_i(x,y)\right)} \quad (3)$$

$$L_s = \boxed{\sum_t} \|q_\lambda^t - q_\lambda^{t-1}\|_2^2, \quad (7)$$

$$\varphi_{ij_a} := \sum_{p=0}^{a-1} \tilde{\theta}_i^{j_p, j_{p+1}}. \quad (8)$$

$$L_r = \boxed{\sum_t \sum_k} \|\Pi q_k^t - u_k^t\|_2^2 \, c_k^t, \quad (5)$$

$$W_p(f_0, f_1) = \min_{\mathbf{P}} \boxed{\sum_{i,j}} c(x_i, y_j) P_{i,j} \quad (5)$$

$$R = \sum_{i=1}^{k} ||h_i' - h_i||_2^2 + \boxed{\sum_{i=1}^{k} \sum_{j=i+1}^{k}} ||h_{i,j}' - h_{i,j}||_2^2, \quad (9)$$

$$\boxed{e \cdot k_0 + \sum_{j=1}^{4} k_j \cdot e^j,} \quad (1)$$

$$\hat{\boldsymbol{\alpha}}_p = \boxed{\sum_i} w_{pi} \hat{\boldsymbol{\alpha}}_i. \quad (44)$$

Summation is ambiguous

Summation is ambiguous

$$\sum_i a_i b_i + c$$

Summation is ambiguous

$$\sum_i a_i b_i + c$$

$$(\sum_i a_i b_i) + c$$

$$(\sum_i a_i b_i + c)$$

Summation is ambiguous

$$\sum_i a_i b_i + c$$

$$(\sum_i a_i b_i) + c$$

$$(\sum_i a_i b_i + c)$$

More complex equations

$$\sum_i a_i + c + \sum_j b_j$$

$$\sum_i a_i + c_i + \sum_j b_j$$

Complex summation formulas from SIGGRAPH 2019

Complex summation formulas from SIGGRAPH 2019

$$L_{\text{total}} = \sum_{(I^S, I^L, I^{L'}) \in \mathcal{A}} L_{\text{rend}}(I^S, I^L) + L_{\text{adjust\_rend}}(I^S, I^{L'}) + w_1 L_{\text{smooth}}(\mathbb{D}) \tag{6}$$

$$\begin{aligned} \langle F \rangle^{\text{CV}} &= \langle F \rangle + \textstyle\sum_{i=1}^{K} \gamma_i (G_i - \langle G_i \rangle) \\ &= \textstyle\sum_{i=1}^{K} \gamma_i G_i + \langle F \rangle - \sum_{i=1}^{K} \gamma_i \langle G_i \rangle \end{aligned} \tag{14}$$

$$C^{\mathbf{i}_1, \mathbf{i}_2}_{\mathbf{v}_1, \mathbf{v}_2} \approx \frac{1}{N} \sum_{n=1}^{N} u^{\mathbf{i}_1, O^n}_{\mathbf{v}_1} \cdot u^{\mathbf{i}_2, O^n\,*}_{\mathbf{v}_2} - m^{\mathbf{i}_1}_{\mathbf{v}_1} \cdot m^{\mathbf{i}_2\,*}_{\mathbf{v}_2}. \tag{6}$$

$$\tilde{u}(\boldsymbol{x}, k) = \sum_j a_{jk} F_k(\boldsymbol{x} - \boldsymbol{y}_j, k) + u_{\text{in}}(\boldsymbol{x}, k) \tag{22}$$

$$= -\sum_j a_{jk} \frac{i}{4} H_0^{(2)}(k \|\boldsymbol{x} - \boldsymbol{y}_j\|) + u_{\text{in}}(\boldsymbol{x}, k). \tag{23}$$

$$\begin{aligned} &\min_{\mathbf{a}} && \|\ddot{\mathbf{q}}_{\text{d}}(\mathbf{u}) - \ddot{\mathbf{q}}(\mathbf{a})\|^2 + w_{\text{reg}} \|\mathbf{a}\|^2 \\ &\text{subject to} && \mathbf{M}\ddot{\mathbf{q}} + \mathbf{c} = \sum_m \mathbf{J}_m^\top \mathbf{f}_m(a_m) + \mathbf{J}_{\text{c}}^\top \mathbf{f}_{\text{c}} + \tau_{\text{ext}} \\ &&& 0 \le a_m \le 1 \quad \text{for} \quad \forall m. \end{aligned} \tag{13}$$

$$\begin{aligned} \mathcal{E}_{symm} &\approx \sum_i \Big[\cos\theta\, (p_i - q_i) \cdot n_i + \\ &\qquad \cos\theta\, \big(\tilde{a} \times (p_i + q_i)\big) \cdot n_i \,+\, t \cdot n_i \Big]^2 \\ &= \sum_i \cos^2\theta \Big[(p_i - q_i) \cdot n_i + \\ &\qquad \big((p_i + q_i) \times n_i\big) \cdot \tilde{a} \,+\, n_i \cdot \tilde{t} \Big]^2, \end{aligned} \tag{9}$$

$$\begin{aligned} \mathcal{E}_{two\text{-}plane} = \sum_i &\Big[(\mathrm{R}p_i - \mathrm{R}^{-1} q_i + t) \cdot (\mathrm{R} n_{p,i}) \Big]^2 + \\ &\Big[(\mathrm{R}p_i - \mathrm{R}^{-1} q_i + t) \cdot (\mathrm{R}^{-1} n_{q,i}) \Big]^2. \end{aligned} \tag{14}$$

$$f_{\mathrm{s,g}}(\mathbf{x}) = \sum_i a_i \phi(\mathbf{x}, \mathbf{x}_i) + \sum_i \mathbf{b}_i^T D^{0,1} \phi(\mathbf{x}, \mathbf{x}_i) + \mathbf{c}^T \mathbf{x} + d \tag{3}$$

$$L(\hat{x}, x_\Gamma) = \sum_{j \in [1,s]} \Big(E_j(x_j) + \frac{1}{2} \|z_j - R_{\Gamma_j} x_\Gamma\|^2_{K_j} \Big). \tag{5}$$

$$\begin{aligned} \Sigma_J &= [\check{q}_J]^{-1} = \Big[\sum_i \alpha_i \check{q}_i \Big]^{-1} = \Big(\sum_i \alpha_i \Sigma_i^{-1} \Big)^{-1}, \\ \mu_J &= \Sigma_J \Big(\sum_i \alpha_i \bar{q}_i \Big) = \Big(\sum_i \alpha_i \Sigma_i^{-1} \Big)^{-1} \Big(\sum_i \alpha_i \Sigma_i^{-1} \mu_i \Big), \end{aligned} \tag{13}$$

Complex summation formulas from SIGGRAPH 2019

$$L_{\text{total}} = \sum_{(I^S, I^L, I^{L'}) \in \mathcal{A}} L_{\text{rend}}(I^S, I^L) + L_{\text{adjust\_rend}}(I^S, I^{L'}) + w_1 L_{\text{smooth}}(\mathbb{D}) \quad (6)$$

$$\begin{aligned} \langle F \rangle^{\text{CV}} &= \langle F \rangle + \textstyle\sum_{i=1}^{K} \gamma_i (G_i - \langle G_i \rangle) \\ &= \textstyle\sum_{i=1}^{K} \gamma_i G_i + \langle F \rangle - \sum_{i=1}^{K} \gamma_i \langle G_i \rangle \end{aligned} \quad (14)$$

$$C_{\mathbf{v}_1, \mathbf{v}_2}^{\mathbf{i}_1, \mathbf{i}_2} \approx \frac{1}{N} \sum_{n=1}^{N} u_{\mathbf{v}_1}^{\mathbf{i}_1, O^n} \cdot u_{\mathbf{v}_2}^{\mathbf{i}_2, O^n *} - m_{\mathbf{v}_1}^{\mathbf{i}_1} \cdot m_{\mathbf{v}_2}^{\mathbf{i}_2 *}. \quad (6)$$

$$\tilde{u}(\boldsymbol{x}, k) = \sum_j a_{jk} F_k(\boldsymbol{x} - \boldsymbol{y}_j, k) + u_{\text{in}}(\boldsymbol{x}, k) \quad (22)$$

$$= -\sum_j a_{jk} \frac{i}{4} H_0^{(2)}(k \|\boldsymbol{x} - \boldsymbol{y}_j\|) + u_{\text{in}}(\boldsymbol{x}, k). \quad (23)$$

$$\begin{aligned} \min_{\mathbf{a}} \quad & \|\ddot{\mathbf{q}}_{\text{d}}(\mathbf{u}) - \ddot{\mathbf{q}}(\mathbf{a})\|^2 + w_{\text{reg}} \|\mathbf{a}\|^2 \\ \text{subject to} \quad & \mathbf{M}\ddot{\mathbf{q}} + \mathbf{c} = \sum_m \mathbf{J}_m^\top \mathbf{f}_m(a_m) + \mathbf{J}_{\text{c}}^\top \mathbf{f}_{\text{c}} + \tau_{\text{ext}} \\ & 0 \le a_m \le 1 \quad \text{for} \quad \forall m. \end{aligned} \quad (13)$$

$$\begin{aligned} \mathcal{E}_{symm} &\approx \sum_i \Big[\cos\theta \, (p_i - q_i) \cdot n_i + \\ &\qquad \cos\theta \, \big(\tilde{a} \times (p_i + q_i)\big) \cdot n_i + t \cdot n_i \Big]^2 \\ &= \sum_i \cos^2\theta \Big[(p_i - q_i) \cdot n_i + \\ &\qquad \big((p_i + q_i) \times n_i\big) \cdot \tilde{a} + n_i \cdot \tilde{t} \Big]^2, \end{aligned} \quad (9)$$

$$\mathcal{E}_{two\text{-}plane} = \sum_i \Big[(\mathrm{R}p_i - \mathrm{R}^{-1} q_i + t) \cdot (\mathrm{R} n_{p,i}) \Big]^2 + \Big[(\mathrm{R}p_i - \mathrm{R}^{-1} q_i + t) \cdot (\mathrm{R}^{-1} n_{q,i}) \Big]^2. \quad (14)$$

$$f_{\mathrm{s,g}}(\mathbf{x}) = \sum_i a_i \phi(\mathbf{x}, \mathbf{x}_i) + \sum_i \mathbf{b}_i^T D^{0,1} \phi(\mathbf{x}, \mathbf{x}_i) + \mathbf{c}^T \mathbf{x} + d \quad (3)$$

$$L(\hat{x}, x_\Gamma) = \sum_{j \in [1,s]} \Big(E_j(x_j) + \frac{1}{2} \|z_j - R_{\Gamma_j} x_\Gamma\|_{K_j}^2 \Big). \quad (5)$$

$$\begin{aligned} \Sigma_J &= [\check{q}_J]^{-1} = \Big[\sum_i \alpha_i \check{q}_i \Big]^{-1} = \Big(\sum_i \alpha_i \Sigma_i^{-1} \Big)^{-1}, \\ \mu_J &= \Sigma_J \Big(\sum_i \alpha_i \bar{q}_i \Big) = \Big(\sum_i \alpha_i \Sigma_i^{-1} \Big)^{-1} \Big(\sum_i \alpha_i \Sigma_i^{-1} \mu_i \Big), \end{aligned} \quad (13)$$

Summation in I❤️LA

Summation in I❤️LA

- Conservative rather than greedy

$$\sum_i a_i b_i + c$$

$$\boxed{\left(\sum_i a_i b_i\right) + c}$$

$$\left(\sum_i a_i b_i + c\right)$$

Summation in I❤️LA

- Conservative rather than greedy

$$\sum_i a_i b_i + c$$

$$\boxed{(\sum_i a_i b_i) + c}$$

$$(\sum_i a_i b_i + c)$$

- Bounds inferred from index use

```
∑_i w_i T_i p
where
w_i ∈ ℝ
T_i ∈ ℝ^(4×4)
p ∈ ℝ⁴
```

Summation in I❤️LA

- Conservative rather than greedy

$$\sum_i a_i b_i + c$$

$$\boxed{\left(\sum_i a_i b_i\right) + c}$$

$$\left(\sum_i a_i b_i + c\right)$$

- Bounds inferred from index use

```
∑_i w_i T_i p
where
w_i ∈ ℝ
T_i ∈ ℝ^(4×4)
p ∈ ℝ^4
```

Summation in I❤️LA

- Conservative rather than greedy

$$\sum_i a_i b_i + c \rightarrow \boxed{(\sum_i a_i b_i) + c}$$

$$\sum_i a_i b_i + c \rightarrow (\sum_i a_i b_i + c)$$

- Bounds inferred from index use

```
∑_i w_i T_i p
where
w_i ∈ ℝ
T_i ∈ ℝ^(4×4)
p ∈ ℝ⁴
```

Various norms (14%)

Various norms (14%)

$$\iiint_\Omega \left(\frac{\rho}{\Delta t} \|\mathbf{u} - \mathbf{u}^*\|_2^2 + \mu \|\nabla \mathbf{u}\|_F^2 \right) dV, \tag{5}$$

$$r_k = ||\mathbf{A}\mathbf{x} - \mathbf{b}||_{\mathcal{K}_k} \tag{69}$$

$$\min_\alpha \| \sum_i \alpha_i E^{X_i} - E^V \|_F^2 + \tau \|\alpha\|_1, \tag{10}$$

$$\min_\alpha \|E^U C - C \sum_i \alpha_i E^{V_i}\|^2 + \tau \|\alpha\|_1. \tag{14}$$

$$E(\widetilde{\mathsf{L}}) = \|\mathsf{PM}^{-1}\mathsf{LI} - \widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{PI}\|_{\widetilde{\mathsf{M}}}^2, \tag{3}$$

$$\min_{\{\mathbf{a}_l, \mathbf{b}_l, \mathbf{c}_l\}} \|R - \sum_{l=1}^{\overline{m}} \mathbf{a}_l \otimes \mathbf{b}_l \otimes \mathbf{c}_l\|_{\mathcal{F}}^2 \tag{3}$$

$$\gamma \sum_{i=1}^{r} A_1(p_i) \|X_{12}(p_i, :) - X_2(q_i)\|_{M_2}^2 + A_2(q_i) \|X_{21}(q_i, :) - X_1(p_i)\|_{M_1}^2. \tag{13}$$

$$\mathbf{w}_j = \left(\mathbf{R}_j^W\right)^\top \left(\frac{\mathbf{P} - \mathbf{t}_j^W}{\|\mathbf{P} - \mathbf{t}_j^W\|_2} \right). \tag{13}$$

Various norms (14%)

$$\iiint_{\Omega} \left(\frac{\rho}{\Delta t} \boxed{\|\mathbf{u} - \mathbf{u}^*\|_2^2} + \mu \|\nabla \mathbf{u}\|_F^2 \right) dV, \tag{5}$$

$$E(\widetilde{\mathsf{L}}) = \|\mathsf{P}\mathsf{M}^{-1}\mathsf{L}\mathsf{I} - \widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{P}\mathsf{I}\|_{\widetilde{\mathsf{M}}}^2, \tag{3}$$

$$r_k = ||\mathbf{A}\mathbf{x} - \mathbf{b}||_{\mathcal{K}_k} \tag{69}$$

$$\min_{\{\mathbf{a}_l, \mathbf{b}_l, \mathbf{c}_l\}} \|R - \sum_{l=1}^{\overline{m}} \mathbf{a}_l \otimes \mathbf{b}_l \otimes \mathbf{c}_l\|_{\mathcal{F}}^2 \tag{3}$$

$$\min_{\alpha} \| \sum_i \alpha_i E^{X_i} - E^V \|_F^2 + \tau \|\alpha\|_1, \tag{10}$$

$$\gamma \sum_{i=1}^{r} A_1(p_i) \|X_{12}(p_i, :) - X_2(q_i)\|_{M_2}^2 + A_2(q_i) \|X_{21}(q_i, :) - X_1(p_i)\|_{M_1}^2. \tag{13}$$

$$\min_{\alpha} \|E^U C - C \sum_i \alpha_i E^{V_i}\|^2 + \tau \|\alpha\|_1. \tag{14}$$

$$\mathbf{w}_j = \left(\mathbf{R}_j^W\right)^\top \left(\frac{\mathbf{P} - \mathbf{t}_j^W}{\boxed{\|\mathbf{P} - \mathbf{t}_j^W\|_2}} \right). \tag{13}$$

Various norms (14%)

$$\iiint_\Omega \left(\frac{\rho}{\Delta t} \boxed{\|\mathbf{u} - \mathbf{u}^*\|_2^2} + \mu \|\nabla \mathbf{u}\|_F^2 \right) dV, \tag{5}$$

$$E(\widetilde{\mathsf{L}}) = \|\mathsf{PM}^{-1}\mathsf{LI} - \widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{PI}\|_{\widetilde{\mathsf{M}}}^2, \tag{3}$$

$$r_k = ||\mathbf{Ax} - \mathbf{b}||_{\mathcal{K}_k} \tag{69}$$

$$\min_{\{\mathbf{a}_l, \mathbf{b}_l, \mathbf{c}_l\}} \|R - \sum_{l=1}^{\overline{m}} \mathbf{a}_l \otimes \mathbf{b}_l \otimes \mathbf{c}_l\|_{\mathcal{F}}^2 \tag{3}$$

$$\min_\alpha \| \sum_i \alpha_i E^{X_i} - E^V \|_F^2 + \tau \boxed{\|\alpha\|_1}. \tag{10}$$

$$\gamma \sum_{i=1}^{r} A_1(p_i) \|X_{12}(p_i, :) - X_2(q_i)\|_{M_2}^2 + A_2(q_i) \|X_{21}(q_i, :) - X_1(p_i)\|_{M_1}^2. \tag{13}$$

$$\min_\alpha \|E^U C - C \sum_i \alpha_i E^{V_i}\|^2 + \tau \boxed{\|\alpha\|_1}. \tag{14}$$

$$\mathbf{w}_j = \left(\mathbf{R}_j^W\right)^\top \left(\frac{\mathbf{P} - \mathbf{t}_j^W}{\boxed{\|\mathbf{P} - \mathbf{t}_j^W\|_2}} \right). \tag{13}$$

Various norms (14%)

$$\iiint_{\Omega} \left(\frac{\rho}{\Delta t} \boxed{\|\mathbf{u} - \mathbf{u}^*\|_2^2} + \mu \boxed{\|\nabla \mathbf{u}\|_F^2} \right) dV, \tag{5}$$

$$E(\widetilde{\mathsf{L}}) = \|\mathsf{P}\mathsf{M}^{-1}\mathsf{L}\mathsf{I} - \widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{P}\mathsf{I}\|_{\widetilde{\mathsf{M}}}^2, \tag{3}$$

$$r_k = ||\mathbf{A}\mathbf{x} - \mathbf{b}||_{\mathcal{K}_k} \tag{69}$$

$$\min_{\{\mathbf{a}_l, \mathbf{b}_l, \mathbf{c}_l\}} \boxed{\|R - \sum_{l=1}^{m} \mathbf{a}_l \otimes \mathbf{b}_l \otimes \mathbf{c}_l\|_{\mathcal{F}}^2} \tag{3}$$

$$\min_{\alpha} \boxed{\| \sum_i \alpha_i E^{X_i} - E^V \|_F^2} + \tau \boxed{\|\alpha\|_1}. \tag{10}$$

$$\gamma \sum_{i=1}^{r} A_1(p_i) \|X_{12}(p_i, :) - X_2(q_i)\|_{M_2}^2 + A_2(q_i) \|X_{21}(q_i, :) - X_1(p_i)\|_{M_1}^2. \tag{13}$$

$$\min_{\alpha} \|E^U C - C \sum_i \alpha_i E^{V_i}\|^2 + \tau \boxed{\|\alpha\|_1}. \tag{14}$$

$$\mathbf{w}_j = \left(\mathbf{R}_j^W\right)^\top \left(\frac{\mathbf{P} - \mathbf{t}_j^W}{\boxed{\|\mathbf{P} - \mathbf{t}_j^W\|_2}} \right). \tag{13}$$

Various norms (14%)

$$\iiint_\Omega \left(\frac{\rho}{\Delta t} \boxed{\|\mathbf{u} - \mathbf{u}^*\|_2^2} + \mu \boxed{\|\nabla \mathbf{u}\|_F^2} \right) dV, \tag{5}$$

$$E(\widetilde{\mathsf{L}}) = \boxed{\|\mathsf{PM}^{-1}\mathsf{LI} - \widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{PI}\|_{\widetilde{\mathsf{M}}}^2}, \tag{3}$$

$$r_k = \boxed{||\mathbf{Ax} - \mathbf{b}||_{\mathcal{K}_k}} \tag{69}$$

$$\min_{\{\mathbf{a}_l, \mathbf{b}_l, \mathbf{c}_l\}} \boxed{\|R - \sum_{l=1}^{m} \mathbf{a}_l \otimes \mathbf{b}_l \otimes \mathbf{c}_l\|_{\mathcal{F}}^2} \tag{3}$$

$$\min_\alpha \boxed{\| \sum_i \alpha_i E^{X_i} - E^V \|_F^2} + \tau \boxed{\|\alpha\|_1}. \tag{10}$$

$$\gamma \sum_{i=1}^{r} A_1(p_i) \boxed{\|X_{12}(p_i,:) - X_2(q_i)\|_{M_2}^2} + A_2(q_i)\|X_{21}(q_i,:) - X_1(p_i)\|_{M_1}^2. \tag{13}$$

$$\min_\alpha \|E^U C - C \sum_i \alpha_i E^{V_i}\|^2 + \tau \boxed{\|\alpha\|_1}. \tag{14}$$

$$\mathbf{w}_j = \left(\mathbf{R}_j^W\right)^\top \left(\frac{\mathbf{P} - \mathbf{t}_j^W}{\boxed{\|\mathbf{P} - \mathbf{t}_j^W\|_2}} \right). \tag{13}$$

Norms in I❤️LA

```
a = ‖T‖₁ + ‖T‖
b = ‖T‖_∞ + ‖T‖_P
c = ‖P‖_* + ‖P‖_F
where
T: ℝ²: a vector
P: ℝ^(2×2): a matrix
```

I❤️LA compiler

I❤️LA compiler

I❤️LA
Source

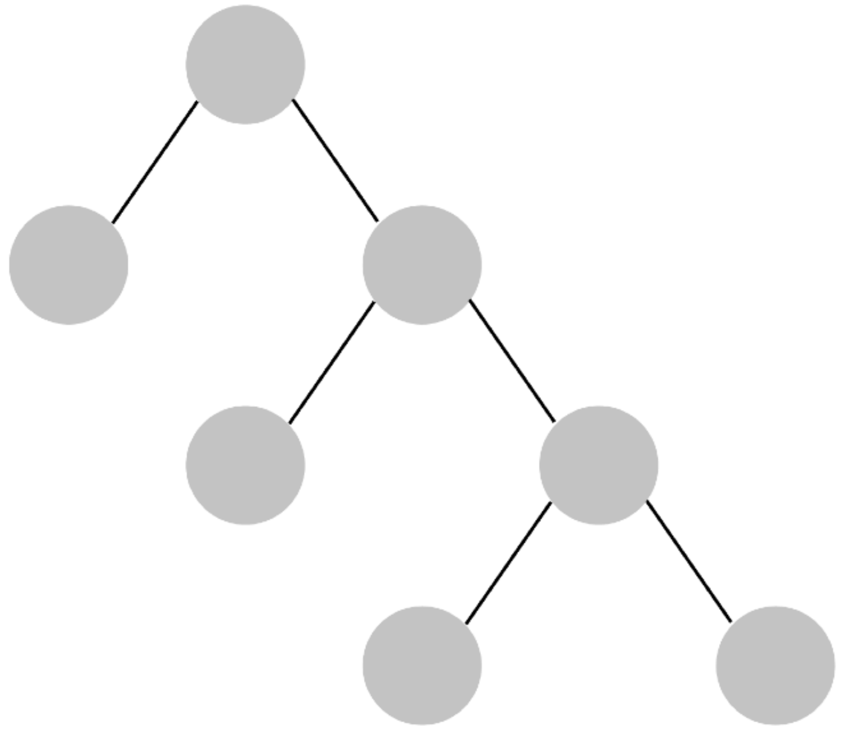
```
y = f(x)²

where
x ∈ ℝ²
f ∈ ℝ² → ℝ
```

I❤️LA compiler

I❤️LA
Source

```
y = f(x)²

where
x ∈ ℝ²
f ∈ ℝ² → ℝ
```

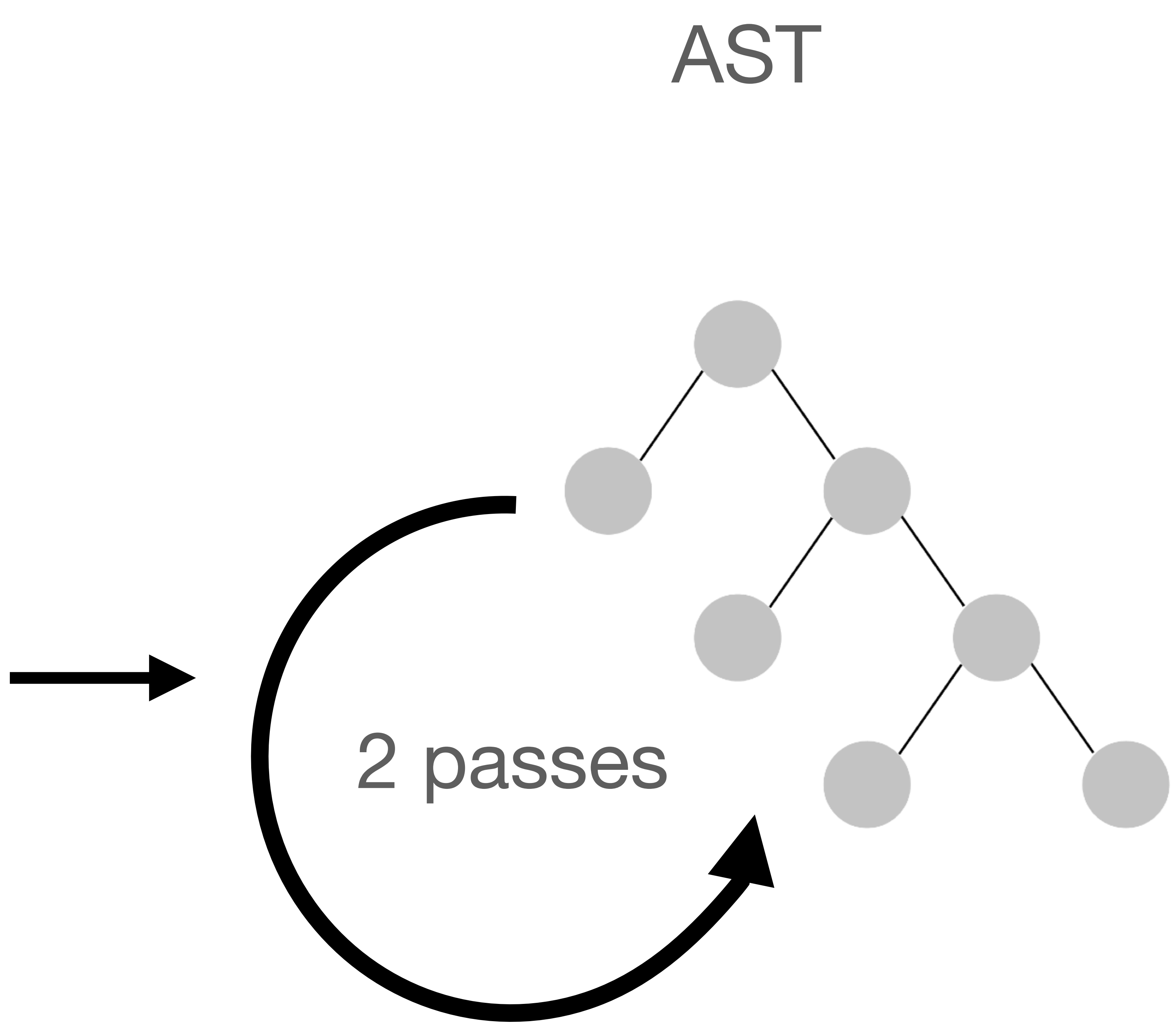
AST



I❤️LA compiler

I❤️LA
Source

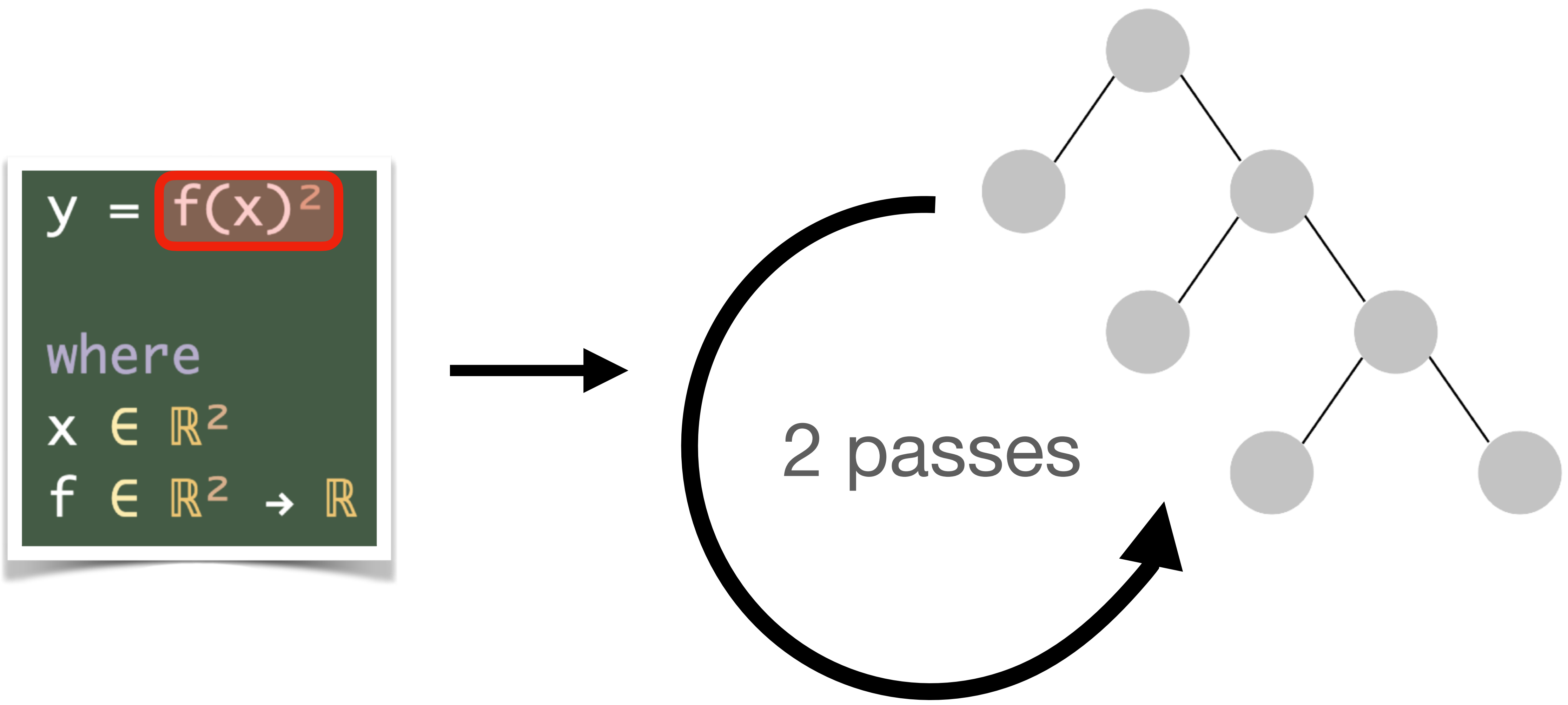
```
y = f(x)²

where
x ∈ ℝ²
f ∈ ℝ² → ℝ
```



I❤️LA compiler

I❤️LA Source

AST

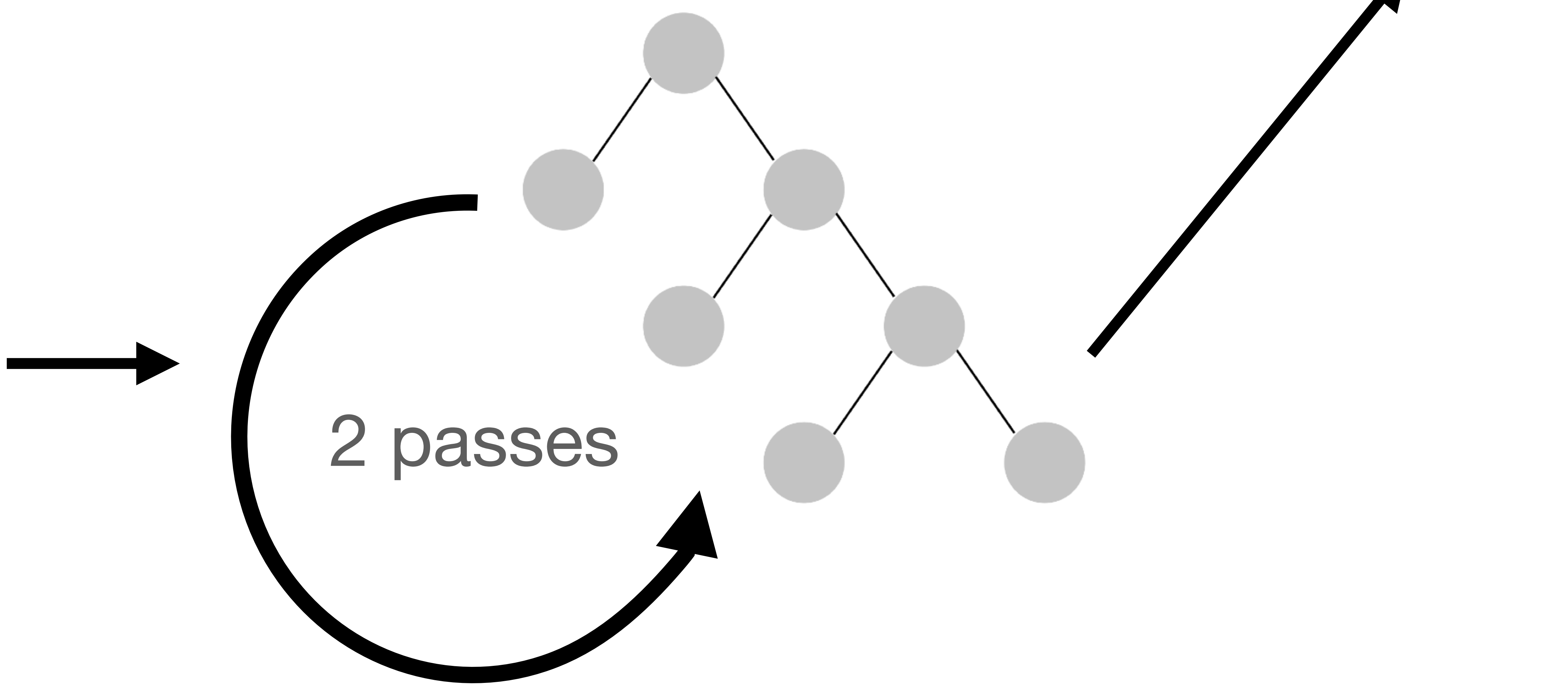


I❤️LA compiler

I❤️LA
Source

```
y = f(x)²

where
x ∈ ℝ²
f ∈ ℝ² → ℝ
```

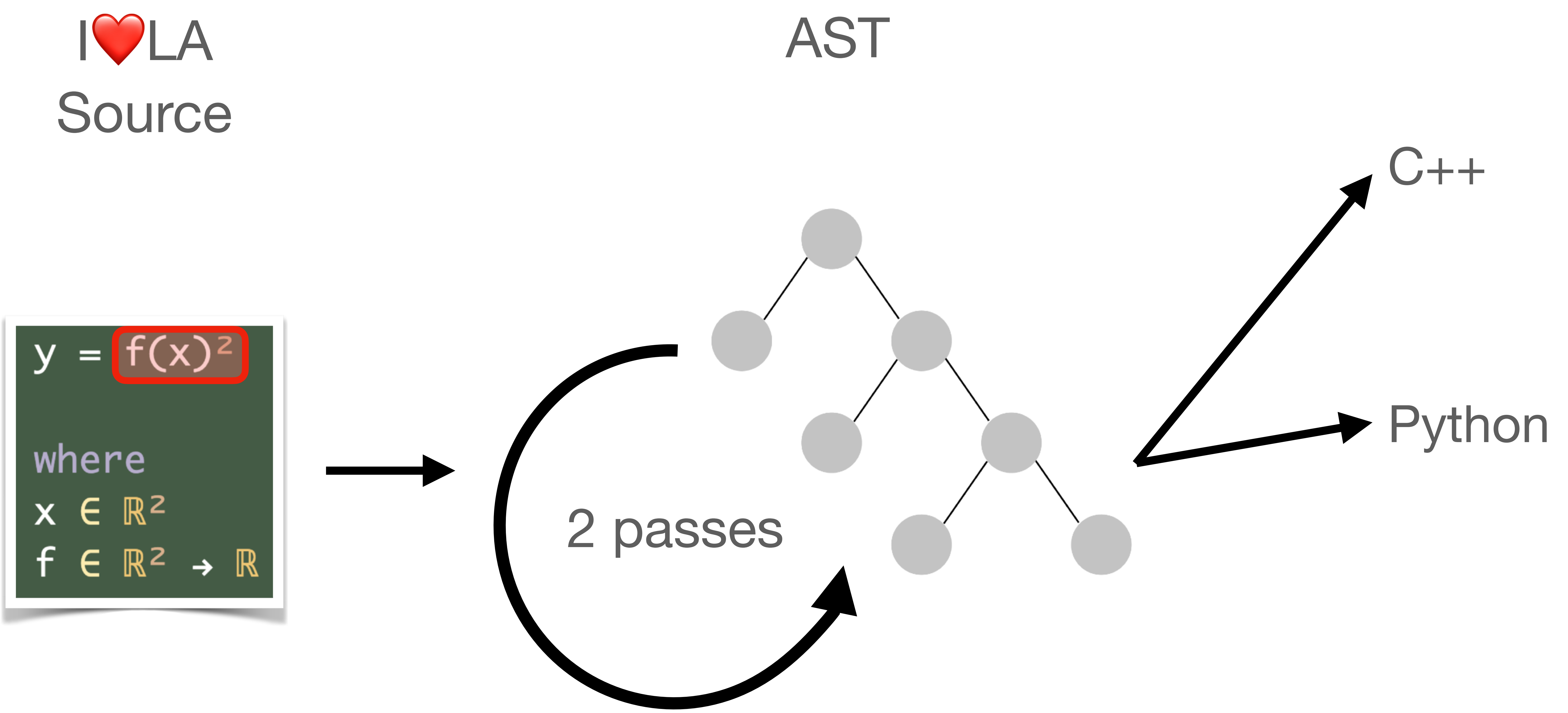
AST



Code
Generation

```
myExpressionResultType myExpression(
    const Eigen::Matrix<double, 2, 1> & x,
    const std::function<double(Eigen::Matrix<double, 2, 1>)> & f)
{
    double y = pow(f(x), 2);
    return myExpressionResultType(y);
}
```

I❤️LA compiler

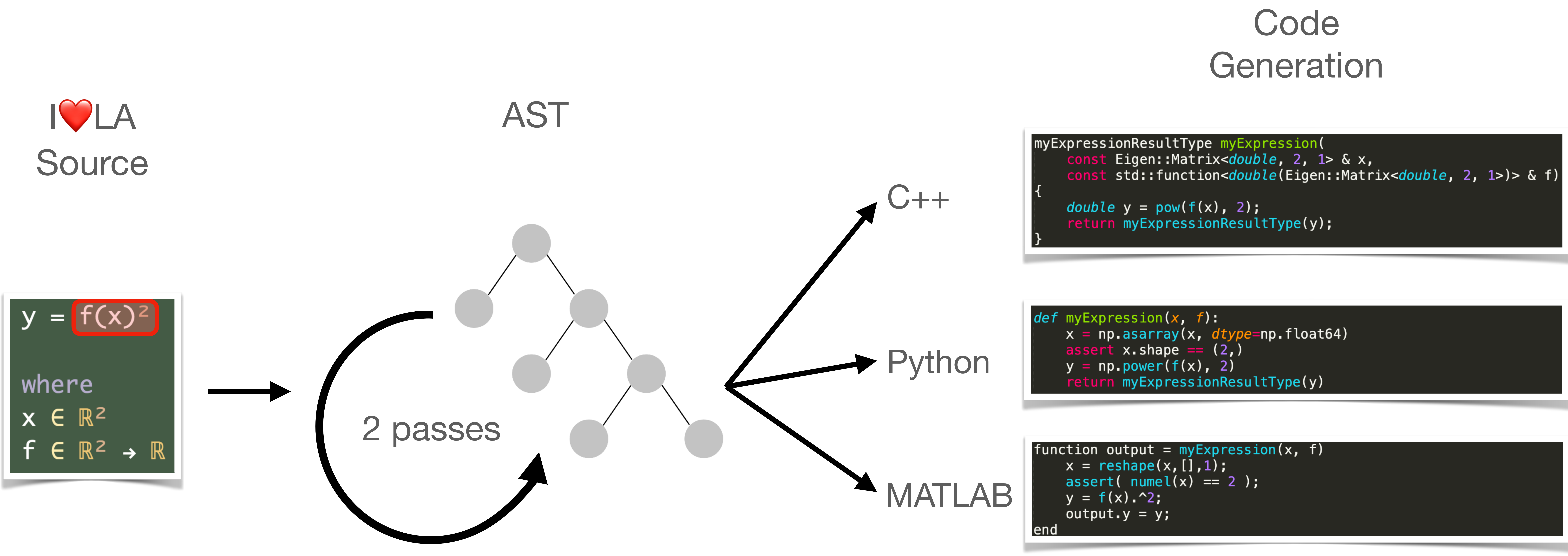


Code Generation

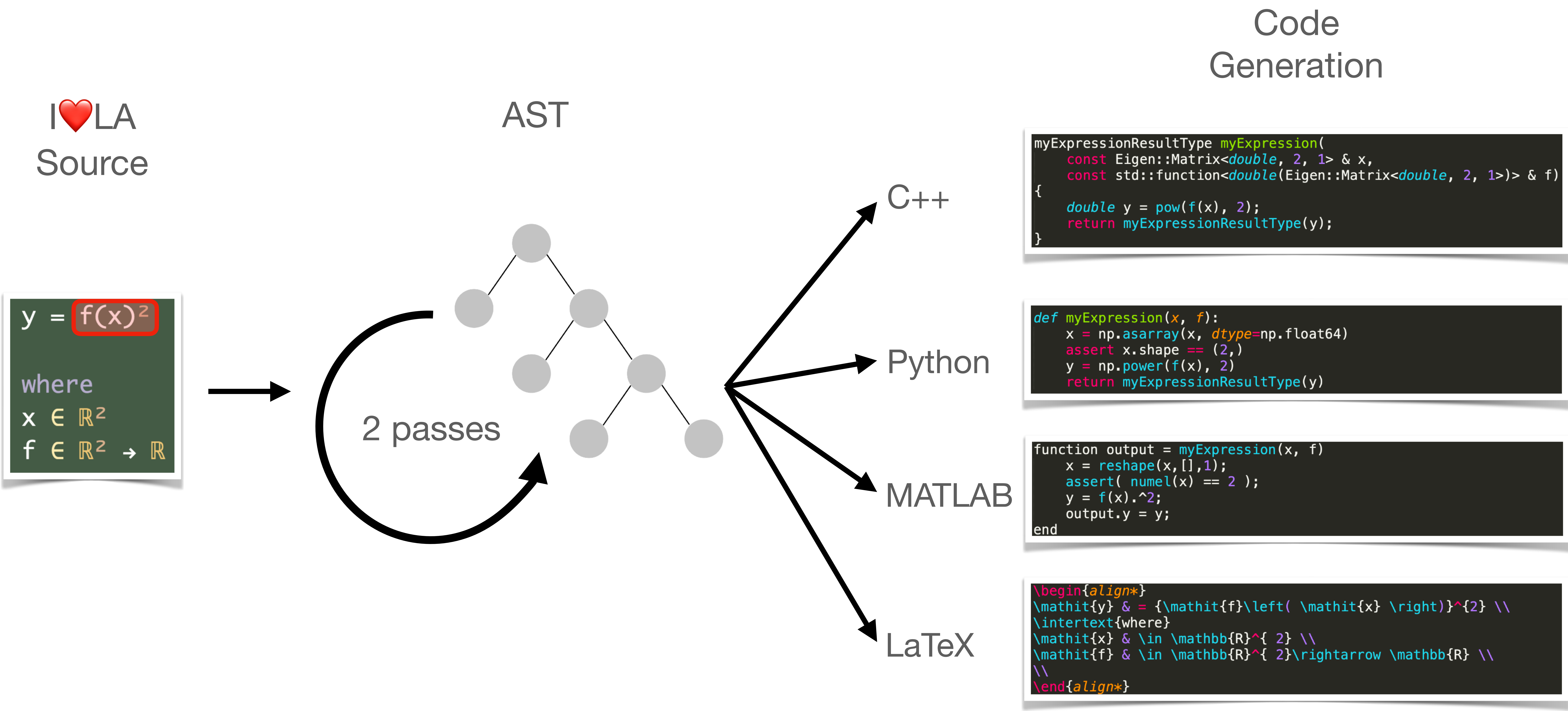
```cpp
myExpressionResultType myExpression(
    const Eigen::Matrix<double, 2, 1> & x,
    const std::function<double(Eigen::Matrix<double, 2, 1>)> & f)
{
    double y = pow(f(x), 2);
    return myExpressionResultType(y);
}
```

```python
def myExpression(x, f):
    x = np.asarray(x, dtype=np.float64)
    assert x.shape == (2,)
    y = np.power(f(x), 2)
    return myExpressionResultType(y)
```

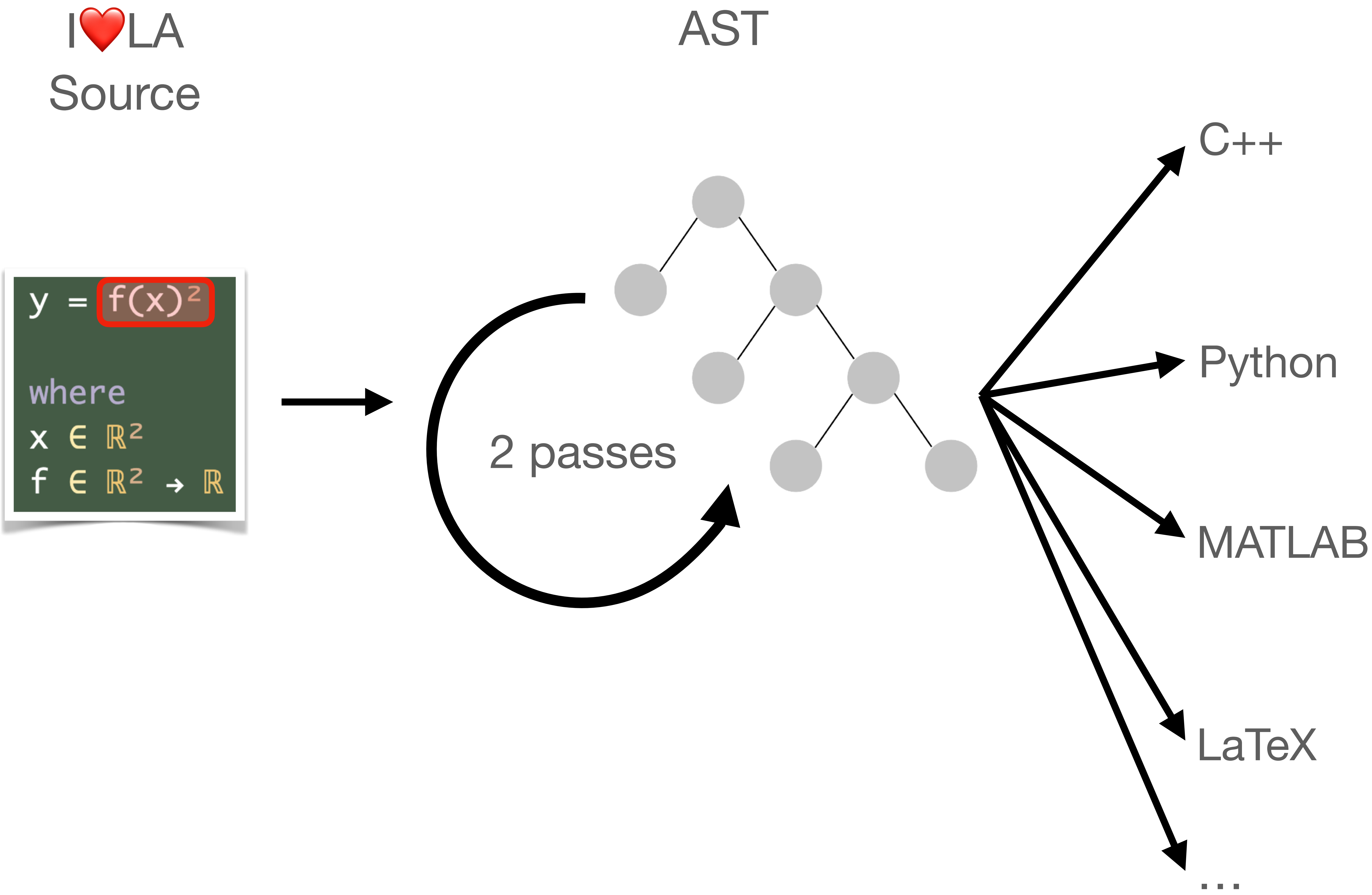
I❤️LA compiler



I❤️LA compiler



I❤️LA compiler



Code
Generation

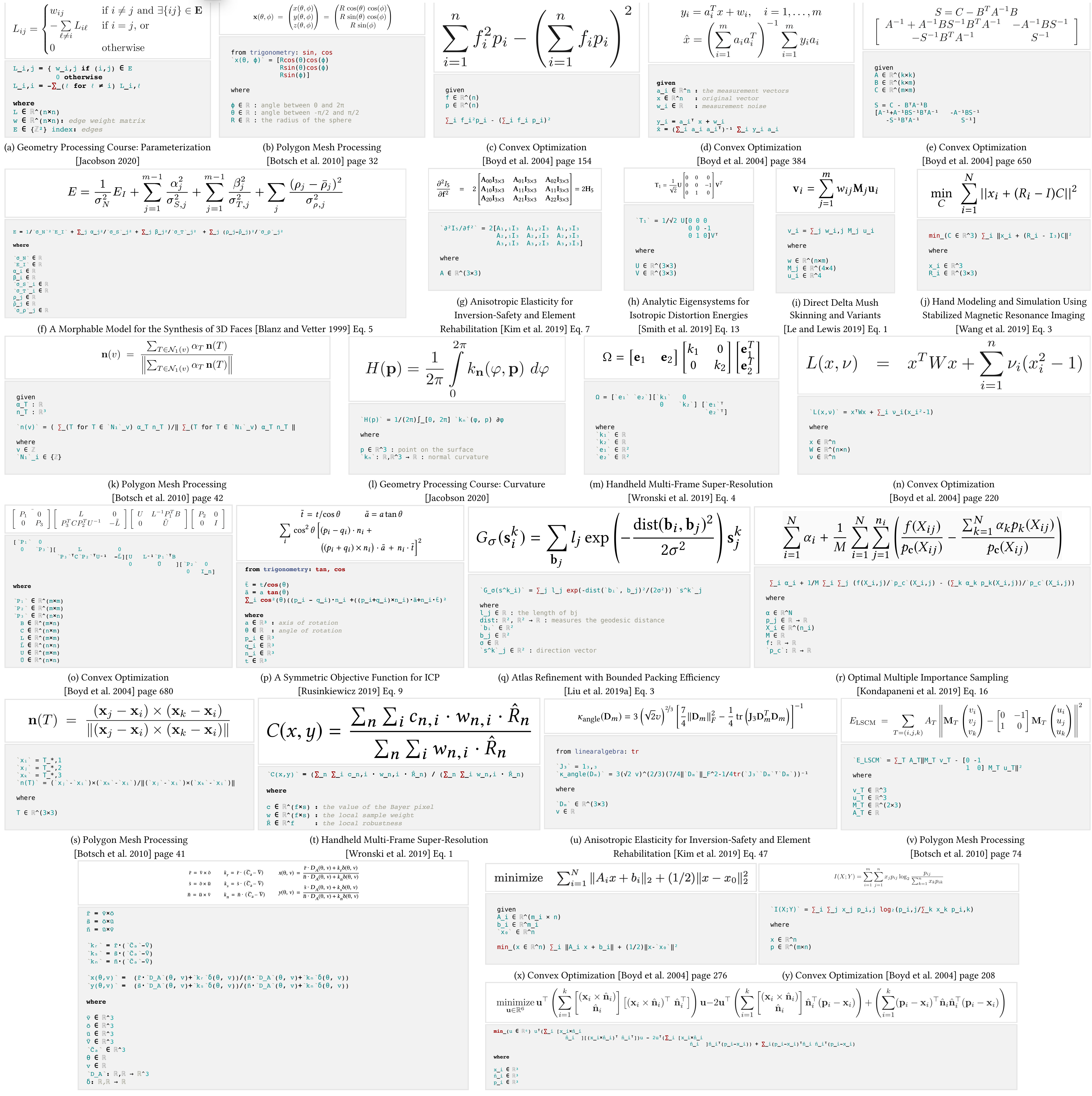
```cpp
myExpressionResultType myExpression(
    const Eigen::Matrix<double, 2, 1> & x,
    const std::function<double(Eigen::Matrix<double, 2, 1>)> & f)
{
    double y = pow(f(x), 2);
    return myExpressionResultType(y);
}
```

```python
def myExpression(x, f):
    x = np.asarray(x, dtype=np.float64)
    assert x.shape == (2,)
    y = np.power(f(x), 2)
    return myExpressionResultType(y)
```

```matlab
function output = myExpression(x, f)
    x = reshape(x,[],1);
    assert( numel(x) == 2 );
    y = f(x).^2;
    output.y = y;
end
```

```latex
\begin{align*}
\mathit{y} & = {\mathit{f}\left( \mathit{x} \right)}^{2} \\
\intertext{where}
\mathit{x} & \in \mathbb{R}^{ 2} \\
\mathit{f} & \in \mathbb{R}^{ 2}\rightarrow \mathbb{R} \\
\\
\end{align*}
```

Examples from the Wild



(a) Geometry Processing Course: Parameterization [Jacobson 2020]
(b) Polygon Mesh Processing [Botsch et al. 2010] page 32
(c) Convex Optimization [Boyd et al. 2004] page 154
(d) Convex Optimization [Boyd et al. 2004] page 384
(e) Convex Optimization [Boyd et al. 2004] page 650
(f) A Morphable Model for the Synthesis of 3D Faces [Blanz and Vetter 1999] Eq. 5
(g) Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation [Kim et al. 2019] Eq. 7
(h) Analytic Eigensystems for Isotropic Distortion Energies [Smith et al. 2019] Eq. 13
(i) Direct Delta Mush Skinning and Variants [Le and Lewis 2019] Eq. 1
(j) Hand Modeling and Simulation Using Stabilized Magnetic Resonance Imaging [Wang et al. 2019] Eq. 3
(k) Polygon Mesh Processing [Botsch et al. 2010] page 42
(l) Geometry Processing Course: Curvature [Jacobson 2020]
(m) Handheld Multi-Frame Super-Resolution [Wronski et al. 2019] Eq. 4
(n) Convex Optimization [Boyd et al. 2004] page 220
(o) Convex Optimization [Boyd et al. 2004] page 680
(p) A Symmetric Objective Function for ICP [Rusinkiewicz 2019] Eq. 9
(q) Atlas Refinement with Bounded Packing Efficiency [Liu et al. 2019a] Eq. 3
(r) Optimal Multiple Importance Sampling [Kondapaneni et al. 2019] Eq. 16
(s) Polygon Mesh Processing [Botsch et al. 2010] page 41
(t) Handheld Multi-Frame Super-Resolution [Wronski et al. 2019] Eq. 1
(u) Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation [Kim et al. 2019] Eq. 47
(v) Polygon Mesh Processing [Botsch et al. 2010] page 74
(w) Plenoptic Modeling: An Image-Based Rendering System [McMillan and Bishop 1995] Eq. 22
(x) Convex Optimization [Boyd et al. 2004] page 276
(y) Convex Optimization [Boyd et al. 2004] page 208
(z) Geometry Processing Course: Registration [Jacobson 2020]

Examples from the Wild

$$L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E} \\ -\sum_{\ell \neq i} L_{i\ell} & \text{if } i = j\text{, or} \\ 0 & \text{otherwise} \end{cases}$$

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

(a) Geometry Processing Course: Parameterization [Jacobson 2020]

$$\mathbf{x}(\theta, \phi) = \begin{pmatrix} x(\\ y(\\ z(\end{pmatrix}$$

```
from trigonometry:
`x(θ, ϕ)` = [Rcos(
             Rsin(
             Rsin(

where

ϕ ∈ ℝ : angle betw
θ ∈ ℝ : angle betw
R ∈ ℝ : the radius
```

(b) Polyg
[Botsch

Examples from the Wild

$$L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E} \\ -\sum_{\ell \neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}$$

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

$$L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E} \\ -\sum\limits_{\ell \neq i} L_{i\ell} & \text{if } i = j \text{, or} \\ 0 & \text{otherwise} \end{cases}$$

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

Geometry Processing: Parameterization

An example from Geometry Processing: Parameterization

The original equation:

We can try to remedy this by introducing a non-uniform weight or spring stiffness w_{ij} for each edge $\{i, j\}$:

$$\min_{\mathbf{U}} \sum_{\{i,j\}\in\mathbf{E}} w_{ij}\|\mathbf{u}_i - \mathbf{u}_j\|^2.$$

For example, we could weigh the distortion of shorter edges (on the 3D mesh) more than longer ones: $w_{ij} = 1/\|\mathbf{v}_i - \mathbf{v}_j\|$. See "Parametrization and smooth approximation of surface triangulations" [Floater 1996]. This will at best help tame ***length distortion***. The "shapes" (i.e., aspect ratios) of triangles will only be indirectly preserved. We need a way to discourage *area distortion* and *angle distortion*.

To do this, let's write the energy minimization problem above in matrix form:

$$\min_{\mathbf{U}} \frac{1}{2} \operatorname{tr}\left(\mathbf{U}^\mathsf{T}\mathbf{L}\mathbf{U}\right),$$

where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a sparse matrix with:

$$\boxed{L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E}, \\ -\sum\limits_{\ell\neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}}.$$

> **What's up with the $\operatorname{tr}()$ in the energy?**
>
> The degrees of freedom in our optimization are a collected in the *matrix* $\mathbf{U} \in \mathbb{R}^{n\times 2}$ with two columns. The energy is written as the trace of the quadratic form (a.k.a. matrix) $\mathbf{Q} \in \mathbb{R}^{n\times n}$ applied to $\mathbf{U}$. In effect, this is really applying $\mathbf{Q}$ to each column of $\mathbf{U}$ independently and summing the result:

$$\begin{aligned} \operatorname{tr}\left(\mathbf{U}^\mathsf{T}\mathbf{Q}\mathbf{U}\right) = & \\ & = \operatorname{tr}\left(\mathbf{U}^\mathsf{T}\mathbf{Q}\mathbf{U}\right) \\ & = \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^\mathsf{T} \\ \mathbf{U}_2^\mathsf{T} \end{bmatrix} \mathbf{Q}[\mathbf{U}_1 \mathbf{U}_2]\right) \\ & = \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^\mathsf{T}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_1^\mathsf{T}\mathbf{Q}\mathbf{U}_2 \\ \mathbf{U}_2^\mathsf{T}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_2^\mathsf{T}\mathbf{Q}\mathbf{U}_2 \end{bmatrix}\right) \\ & = \mathbf{U}_1^\mathsf{T}\mathbf{Q}\mathbf{U}_1 + \mathbf{U}_2^\mathsf{T}\mathbf{Q}\mathbf{U}_2. \end{aligned}$$

I❤️LA implementation:

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

Geometry Processing: Parameterization

An example from Geometry Processing: Parameterization

The original equation:

We can try to remedy this by introducing a non-uniform weight or spring stiffness w_{ij} for each edge $\{i,j\}$:

$$\min_{\mathbf{U}} \sum_{\{i,j\}\in\mathbf{E}} w_{ij}\|\mathbf{u}_i - \mathbf{u}_j\|^2.$$

For example, we could weigh the distortion of shorter edges (on the 3D mesh) more than longer ones: $w_{ij} = 1/\|\mathbf{v}_i - \mathbf{v}_j\|$. See "Parametrization and smooth approximation of surface triangulations" [Floater 1996]. This will at best help tame ***length distortion***. The "shapes" (i.e., aspect ratios) of triangles will only be indirectly preserved. We need a way to discourage *area distortion* and *angle distortion*.

To do this, let's write the energy minimization problem above in matrix form:

$$\min_{\mathbf{U}} \frac{1}{2} \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{L}\mathbf{U}\right),$$

where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a sparse matrix with:

$$\boxed{L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E}, \\ -\sum_{\ell\neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}}.$$

> **What's up with the $\operatorname{tr}()$ in the energy?**
>
> The degrees of freedom in our optimization are a collected in the *matrix* $\mathbf{U} \in \mathbb{R}^{n\times 2}$ with two columns. The energy is written as the trace of the quadratic form (a.k.a. matrix) $\mathbf{Q} \in \mathbb{R}^{n\times n}$ applied to $\mathbf{U}$. In effect, this is really applying $\mathbf{Q}$ to each column of $\mathbf{U}$ independently and summing the result:

$$\begin{aligned} \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) = & \\ &= \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) \\ &= \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^{\mathsf{T}} \\ \mathbf{U}_2^{\mathsf{T}} \end{bmatrix} \mathbf{Q}[\mathbf{U}_1 \mathbf{U}_2]\right) \\ &= \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2 \\ \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2 \end{bmatrix}\right) \\ &= \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 + \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2. \end{aligned}$$

I❤️LA implementation:

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

I❤️LA compiled to C++/Eigen:

```
  */
course_parameterizationResultType course_parameterization(
    const Eigen::MatrixXd & w,
    const std::set<std::tuple< int, int > > & E)
{
    const long n = w.cols();
    assert( w.rows() == n );

    Eigen::SparseMatrix<double> L(n, n);
    std::vector<Eigen::Triplet<double> > tripletList_L;
    for( int i=1; i<=n; i++){
        for( int j=1; j<=n; j++){
            if(E.find(std::tuple< int, int >(i-1, j-1)) != E.end()){
                tripletList_L.push_back(Eigen::Triplet<double>(i-1, j-1
            }
        }
    }
    L.setFromTriplets(tripletList_L.begin(), tripletList_L.end());

    for( int i=1; i<=n; i++){
        double sum_0 = 0;
```

Geometry Processing: Parameterization

An example from Geometry Processing: Parameterization

The original equation:

We can try to remedy this by introducing a non-uniform weight or spring stiffness w_{ij} for each edge $\{i,j\}$:

$$\min_{\mathbf{U}} \sum_{\{i,j\}\in\mathbf{E}} w_{ij}\|\mathbf{u}_i - \mathbf{u}_j\|^2.$$

For example, we could weigh the distortion of shorter edges (on the 3D mesh) more than longer ones: $w_{ij} = 1/\|\mathbf{v}_i - \mathbf{v}_j\|$. See "Parametrization and smooth approximation of surface triangulations" [Floater 1996]. This will at best help tame ***length distortion***. The "shapes" (i.e., aspect ratios) of triangles will only be indirectly preserved. We need a way to discourage *area distortion* and *angle distortion*.

To do this, let's write the energy minimization problem above in matrix form:

$$\min_{\mathbf{U}} \frac{1}{2}\operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{L}\mathbf{U}\right),$$

where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a sparse matrix with:

$$\boxed{L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E}, \\ -\sum_{\ell\neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}}.$$

> **What's up with the $\operatorname{tr}()$ in the energy?**
>
> The degrees of freedom in our optimization are a collected in the *matrix* $\mathbf{U} \in \mathbb{R}^{n\times 2}$ with two columns. The energy is written as the trace of the quadratic form (a.k.a. matrix) $\mathbf{Q} \in \mathbb{R}^{n\times n}$ applied to $\mathbf{U}$. In effect, this is really applying $\mathbf{Q}$ to each column of $\mathbf{U}$ independently and summing the result:

$$\begin{aligned}\operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) = & \\ & = \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) \\ & = \operatorname{tr}\left(\begin{bmatrix}\mathbf{U}_1^{\mathsf{T}} \\ \mathbf{U}_2^{\mathsf{T}}\end{bmatrix}\mathbf{Q}[\mathbf{U}_1\mathbf{U}_2]\right) \\ & = \operatorname{tr}\left(\begin{bmatrix}\mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2 \\ \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2\end{bmatrix}\right) \\ & = \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 + \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2.\end{aligned}$$

I❤️LA implementation:

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

I❤️LA compiled to C++/Eigen:

```cpp
  */
course_parameterizationResultType course_parameterization(
    const Eigen::MatrixXd & w,
    const std::set<std::tuple< int, int > > & E)
{
    const long n = w.cols();
    assert( w.rows() == n );

    Eigen::SparseMatrix<double> L(n, n);
    std::vector<Eigen::Triplet<double> > tripletList_L;
    for( int i=1; i<=n; i++){
        for( int j=1; j<=n; j++){
            if(E.find(std::tuple< int, int >(i-1, j-1)) != E.end()){
                tripletList_L.push_back(Eigen::Triplet<double>(i-1, j-1
            }
        }
    }
    L.setFromTriplets(tripletList_L.begin(), tripletList_L.end());

    for( int i=1; i<=n; i++){
        double sum_0 = 0;
```

I❤️LA compiled to Python/NumPy/SciPy:

```python
def course_parameterization(w, E):
    """
    :param :w : edge weight matrix
    :param :E : edges
    """
    w = np.asarray(w, dtype=np.float64)
    E = frozenset(E)

    n = w.shape[1]
    assert w.shape == (n, n)

    Lij_0 = []
    Lvals_0 = []
    for i in range(1, n+1):
        for j in range(1, n+1):
            if (i-1, j-1) in E:
                Lij_0.append((i-1, j-1))
                Lvals_0.append(w[i-1, j-1])
    sparse_0 = scipy.sparse.coo_matrix((Lvals_0, np.asarray(Lij_0).T), 
    L = sparse_0
    for i in range(1, n+1):
```

Geometry Processing: Parameterization

An example from Geometry Processing: Parameterization

The original equation:

We can try to remedy this by introducing a non-uniform weight or spring stiffness w_{ij} for each edge $\{i,j\}$:

$$\min_{\mathbf{U}} \sum_{\{i,j\}\in\mathbf{E}} w_{ij}\|\mathbf{u}_i - \mathbf{u}_j\|^2.$$

For example, we could weigh the distortion of shorter edges (on the 3D mesh) more than longer ones: $w_{ij} = 1/\|\mathbf{v}_i - \mathbf{v}_j\|$. See "Parametrization and smooth approximation of surface triangulations" [Floater 1996]. This will at best help tame ***length distortion***. The "shapes" (i.e., aspect ratios) of triangles will only be indirectly preserved. We need a way to discourage *area distortion* and *angle distortion*.

To do this, let's write the energy minimization problem above in matrix form:

$$\min_{\mathbf{U}} \frac{1}{2}\operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{L}\mathbf{U}\right),$$

where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a sparse matrix with:

$$\boxed{L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E}, \\ -\sum_{\ell\neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}}.$$

> **What's up with the $\operatorname{tr}()$ in the energy?**
>
> The degrees of freedom in our optimization are a collected in the *matrix* $\mathbf{U} \in \mathbb{R}^{n\times 2}$ with two columns. The energy is written as the trace of the quadratic form (a.k.a. matrix) $\mathbf{Q} \in \mathbb{R}^{n\times n}$ applied to $\mathbf{U}$. In effect, this is really applying $\mathbf{Q}$ to each column of $\mathbf{U}$ independently and summing the result:

$$\begin{aligned} \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) = & \\ &= \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) \\ &= \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^{\mathsf{T}} \\ \mathbf{U}_2^{\mathsf{T}} \end{bmatrix} \mathbf{Q}[\mathbf{U}_1 \mathbf{U}_2]\right) \\ &= \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2 \\ \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2 \end{bmatrix}\right) \\ &= \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 + \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2. \end{aligned}$$

I❤️LA implementation:

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

I❤️LA compiled to C++/Eigen:

```cpp
 */
course_parameterizationResultType course_parameterization(
    const Eigen::MatrixXd & w,
    const std::set<std::tuple< int, int > > & E)
{
    const long n = w.cols();
    assert( w.rows() == n );

    Eigen::SparseMatrix<double> L(n, n);
    std::vector<Eigen::Triplet<double> > tripletList_L;
    for( int i=1; i<=n; i++){
        for( int j=1; j<=n; j++){
            if(E.find(std::tuple< int, int >(i-1, j-1)) != E.end()){
                tripletList_L.push_back(Eigen::Triplet<double>(i-1, j-1
            }
        }
    }
    L.setFromTriplets(tripletList_L.begin(), tripletList_L.end());

    for( int i=1; i<=n; i++){
        double sum_0 = 0;
```

I❤️LA compiled to Python/NumPy/SciPy:

```python
def course_parameterization(w, E):
    """
    :param :w : edge weight matrix
    :param :E : edges
    """
    w = np.asarray(w, dtype=np.float64)
    E = frozenset(E)

    n = w.shape[1]
    assert w.shape == (n, n)

    Lij_0 = []
    Lvals_0 = []
    for i in range(1, n+1):
        for j in range(1, n+1):
            if (i-1, j-1) in E:
                Lij_0.append((i-1, j-1))
                Lvals_0.append(w[i-1, j-1])
    sparse_0 = scipy.sparse.coo_matrix((Lvals_0, np.asarray(Lij_0).T),
    L = sparse_0
    for i in range(1, n+1):
```

I❤️LA compiled to MATLAB:

```matlab
    Lvals_0 = zeros(1,0);
    for i = 1:n
        for j = 1:n
            if ismember([i, j],E,'rows')
                Lij_0(1:2,end+1) = [i;j];
                Lvals_0(end+1) = w(i, j);
            end
        end
    end
    sparse_0 = sparse(Lij_0(1,:),Lij_0(2,:),Lvals_0,n,n);
    L = sparse_0;
    for i = 1:n
        sum_0 = 0;
        for ell = 1:size(L,2)
            if ell ~= i
                sum_0 = sum_0 + L(i, ell);
            end
        end
        Lij_0(1:2,end+1) = [i;i];
```

Geometry Processing: Parameterization

An example from Geometry Processing: Parameterization

The original equation:

We can try to remedy this by introducing a non-uniform weight or spring stiffness w_{ij} for each edge $\{i,j\}$:

$$\min_{\mathbf{U}} \sum_{\{i,j\}\in\mathbf{E}} w_{ij}\|\mathbf{u}_i-\mathbf{u}_j\|^2.$$

For example, we could weigh the distortion of shorter edges (on the 3D mesh) more than longer ones: $w_{ij}=1/\|\mathbf{v}_i-\mathbf{v}_j\|$. See "Parametrization and smooth approximation of surface triangulations" [Floater 1996]. This will at best help tame ***length distortion***. The "shapes" (i.e., aspect ratios) of triangles will only be indirectly preserved. We need a way to discourage *area distortion* and *angle distortion*.

To do this, let's write the energy minimization problem above in matrix form:

$$\min_{\mathbf{U}} \frac{1}{2}\operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{L}\mathbf{U}\right),$$

where $\mathbf{L}\in\mathbb{R}^{n\times n}$ is a sparse matrix with:

$$\boxed{L_{ij}=\begin{cases} w_{ij} & \text{if } i\neq j \text{ and } \exists\{ij\}\in\mathbf{E},\\ -\sum_{\ell\neq i} L_{i\ell} & \text{if } i=j, \text{ or}\\ 0 & \text{otherwise}\end{cases}}$$

> **What's up with the $\operatorname{tr}()$ in the energy?**
>
> The degrees of freedom in our optimization are a collected in the *matrix* $\mathbf{U}\in\mathbb{R}^{n\times 2}$ with two columns. The energy is written as the trace of the quadratic form (a.k.a. matrix) $\mathbf{Q}\in\mathbb{R}^{n\times n}$ applied to $\mathbf{U}$. In effect, this is really applying $\mathbf{Q}$ to each column of $\mathbf{U}$ independently and summing the result:

$$\begin{aligned}\operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) &= \\ &= \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right)\\ &= \operatorname{tr}\left(\begin{bmatrix}\mathbf{U}_1^{\mathsf{T}}\\ \mathbf{U}_2^{\mathsf{T}}\end{bmatrix}\mathbf{Q}[\mathbf{U}_1\mathbf{U}_2]\right)\\ &= \operatorname{tr}\left(\begin{bmatrix}\mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2\\ \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2\end{bmatrix}\right)\\ &= \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1+\mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2.\end{aligned}$$

I❤️LA implementation:

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

I❤️LA LaTeX output:

$$L_{i,j}=\begin{cases} w_{i,j} & \text{if } (i,j)\in E\\ 0 & \text{otherwise}\end{cases}$$

$$L_{i,i}=-\sum_{\ell\neq i} L_{i,\ell}$$

where

$L\in\mathbb{R}^{n\times n}$

$w\in\mathbb{R}^{n\times n}$ edge weight matrix

$E\in\{\mathbb{Z}^2\}$ *index* edges

I❤️LA compiled to LaTeX:

```
\begin{center}
\resizebox{\textwidth}{!}
{
\begin{minipage}[c]{\textwidth}
\begin{align*}
\mathit{L}_{\mathit{i}, \mathit{j}} & = \begin{cases} \mathit{w
\mathit{L}_{\mathit{i}, \mathit{i}} & = -\sum_{\mathit{ℓ} ≠ \ma
\intertext{where}
\mathit{L} & \in \mathbb{R}^{ \mathit{n} \times \mathit{n} } \\
\mathit{w} & \in \mathbb{R}^{ \mathit{n} \times \mathit{n} } \t
\mathit{E} & \in \{\mathbb{Z}^{ 2 }\} \mathit{{ index}} \text{
\\
\end{align*}
\end{minipage}
}
\end{center}

\end{document}
```

I❤️LA compiled to C++/Eigen:

```cpp
 */
course_parameterizationResultType course_parameterization(
    const Eigen::MatrixXd & w,
    const std::set<std::tuple< int, int > > & E)
{
    const long n = w.cols();
    assert( w.rows() == n );

    Eigen::SparseMatrix<double> L(n, n);
    std::vector<Eigen::Triplet<double> > tripletList_L;
    for( int i=1; i<=n; i++){
        for( int j=1; j<=n; j++){
            if(E.find(std::tuple< int, int >(i-1, j-1)) != E.end()){
                tripletList_L.push_back(Eigen::Triplet<double>(i-1, j-1
            }
        }
    }
    L.setFromTriplets(tripletList_L.begin(), tripletList_L.end());

    for( int i=1; i<=n; i++){
        double sum_0 = 0;
```

I❤️LA compiled to Python/NumPy/SciPy:

```python
def course_parameterization(w, E):
    """
    :param :w : edge weight matrix
    :param :E : edges
    """
    w = np.asarray(w, dtype=np.float64)
    E = frozenset(E)

    n = w.shape[1]
    assert w.shape == (n, n)

    Lij_0 = []
    Lvals_0 = []
    for i in range(1, n+1):
        for j in range(1, n+1):
            if (i-1, j-1) in E:
                Lij_0.append((i-1, j-1))
                Lvals_0.append(w[i-1, j-1])
    sparse_0 = scipy.sparse.coo_matrix((Lvals_0, np.asarray(Lij_0).T),
    L = sparse_0
    for i in range(1, n+1):
```

I❤️LA compiled to MATLAB:

```matlab
    Lvals_0 = zeros(1,0);
    for i = 1:n
        for j = 1:n
            if ismember([i, j],E,'rows')
                Lij_0(1:2,end+1) = [i;j];
                Lvals_0(end+1) = w(i, j);
            end
        end
    end
    sparse_0 = sparse(Lij_0(1,:),Lij_0(2,:),Lvals_0,n,n);
    L = sparse_0;
    for i = 1:n
        sum_0 = 0;
        for ell = 1:size(L,2)
            if ell ~= i
                sum_0 = sum_0 + L(i, ell);
            end
        end
        Lij_0(1:2,end+1) = [i;i];
```

Geometry Processing: Parameterization

An example from Geometry Processing: Parameterization

The original equation:

We can try to remedy this by introducing a non-uniform weight or spring stiffness w_{ij} for each edge $\{i, j\}$:

$$\min_{\mathbf{U}} \sum_{\{i,j\}\in\mathbf{E}} w_{ij}\|\mathbf{u}_i - \mathbf{u}_j\|^2.$$

For example, we could weigh the distortion of shorter edges (on the 3D mesh) more than longer ones: $w_{ij} = 1/\|\mathbf{v}_i - \mathbf{v}_j\|$. See "Parametrization and smooth approximation of surface triangulations" [Floater 1996]. This will at best help tame ***length distortion***. The "shapes" (i.e., aspect ratios) of triangles will only be indirectly preserved. We need a way to discourage *area distortion* and *angle distortion*.

To do this, let's write the energy minimization problem above in matrix form:

$$\min_{\mathbf{U}} \frac{1}{2} \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{L}\mathbf{U}\right),$$

where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a sparse matrix with:

$$L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E} \\ -\sum\limits_{\ell\neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}.$$

What's up with the $\operatorname{tr}()$ in the energy?

The degrees of freedom in our optimization are a collected in the *matrix* $\mathbf{U} \in \mathbb{R}^{n\times 2}$ with two columns. The energy is written as the trace of the quadratic form (a.k.a. matrix) $\mathbf{Q} \in \mathbb{R}^{n\times n}$ applied to $\mathbf{U}$. In effect, this is really applying $\mathbf{Q}$ to each column of $\mathbf{U}$ independently and summing the result:

$$\begin{aligned} \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) = & \\ &= \operatorname{tr}\left(\mathbf{U}^{\mathsf{T}}\mathbf{Q}\mathbf{U}\right) \\ &= \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^{\mathsf{T}} \\ \mathbf{U}_2^{\mathsf{T}} \end{bmatrix} \mathbf{Q}[\mathbf{U}_1 \mathbf{U}_2]\right) \\ &= \operatorname{tr}\left(\begin{bmatrix} \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2 \\ \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 & \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2 \end{bmatrix}\right) \\ &= \mathbf{U}_1^{\mathsf{T}}\mathbf{Q}\mathbf{U}_1 + \mathbf{U}_2^{\mathsf{T}}\mathbf{Q}\mathbf{U}_2. \end{aligned}$$

I❤️LA implementation:

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

I❤️LA LaTeX output:

$$L_{i,j} = \begin{cases} w_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$L_{i,i} = -\sum_{\ell\neq i} L_{i,\ell}$$

where

$L \in \mathbb{R}^{n\times n}$

$w \in \mathbb{R}^{n\times n}$ edge weight matrix

$E \in \{\mathbb{Z}^2\}$ *index* edges

I❤️LA compiled to LaTeX:

```
\begin{center}
\resizebox{\textwidth}{!}
{
\begin{minipage}[c]{\textwidth}
\begin{align*}
\mathit{L}_{\mathit{i}, \mathit{j}} & = \begin{cases} \mathit{w
\mathit{L}_{\mathit{i}, \mathit{i}} & = -\sum_{\mathit{ℓ} ≠ \ma
\intertext{where}
\mathit{L} & \in \mathbb{R}^{ \mathit{n} \times \mathit{n} } \\
\mathit{w} & \in \mathbb{R}^{ \mathit{n} \times \mathit{n} } \t
\mathit{E} & \in \{\mathbb{Z}^{ 2 }\} \mathit{{ index}} \text{
\\
\end{align*}
\end{minipage}
}
\end{center}

\end{document}
```

I❤️LA compiled to C++/Eigen:

```cpp
 */
course_parameterizationResultType course_parameterization(
    const Eigen::MatrixXd & w,
    const std::set<std::tuple< int, int > > & E)
{
    const long n = w.cols();
    assert( w.rows() == n );

    Eigen::SparseMatrix<double> L(n, n);
    std::vector<Eigen::Triplet<double> > tripletList_L;
    for( int i=1; i<=n; i++){
        for( int j=1; j<=n; j++){
            if(E.find(std::tuple< int, int >(i-1, j-1)) != E.end()){
                tripletList_L.push_back(Eigen::Triplet<double>(i-1, j-1
            }
        }
    }
    L.setFromTriplets(tripletList_L.begin(), tripletList_L.end());

    for( int i=1; i<=n; i++){
        double sum_0 = 0;
```

I❤️LA compiled to Python/NumPy/SciPy:

```python
def course_parameterization(w, E):
    """
    :param :w : edge weight matrix
    :param :E : edges
    """
    w = np.asarray(w, dtype=np.float64)
    E = frozenset(E)

    n = w.shape[1]
    assert w.shape == (n, n)

    Lij_0 = []
    Lvals_0 = []
    for i in range(1, n+1):
        for j in range(1, n+1):
            if (i-1, j-1) in E:
                Lij_0.append((i-1, j-1))
                Lvals_0.append(w[i-1, j-1])
    sparse_0 = scipy.sparse.coo_matrix((Lvals_0, np.asarray(Lij_0).T),
    L = sparse_0
    for i in range(1, n+1):
```

I❤️LA compiled to MATLAB:

```matlab
    Lvals_0 = zeros(1,0);
    for i = 1:n
        for j = 1:n
            if ismember([i, j],E,'rows')
                Lij_0(1:2,end+1) = [i;j];
                Lvals_0(end+1) = w(i, j);
            end
        end
    end
    sparse_0 = sparse(Lij_0(1,:),Lij_0(2,:),Lvals_0,n,n);
    L = sparse_0;
    for i = 1:n
        sum_0 = 0;
        for ell = 1:size(L,2)
            if ell ~= i
                sum_0 = sum_0 + L(i, ell);
            end
        end
        Lij_0(1:2,end+1) = [i;i];
```

Examples from the Wild

$$L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists\{ij\} \in \mathbf{E} \\ -\sum_{\ell \neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}$$

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

Examples from the Wild

$$L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \text{ and } \exists \{ij\} \in \mathbf{E} \\ -\sum_{\ell \neq i} L_{i\ell} & \text{if } i = j, \text{ or} \\ 0 & \text{otherwise} \end{cases}$$

```
L_i,j = { w_i,j if (i,j) ∈ E
          0 otherwise
L_i,i = -∑_(ℓ for ℓ ≠ i) L_i,ℓ

where
L ∈ ℝ^(n×n)
w ∈ ℝ^(n×n): edge weight matrix
E ∈ {ℤ²} index: edges
```

(a) Geometry Processing Course: Parameterization [Jacobson 2020]

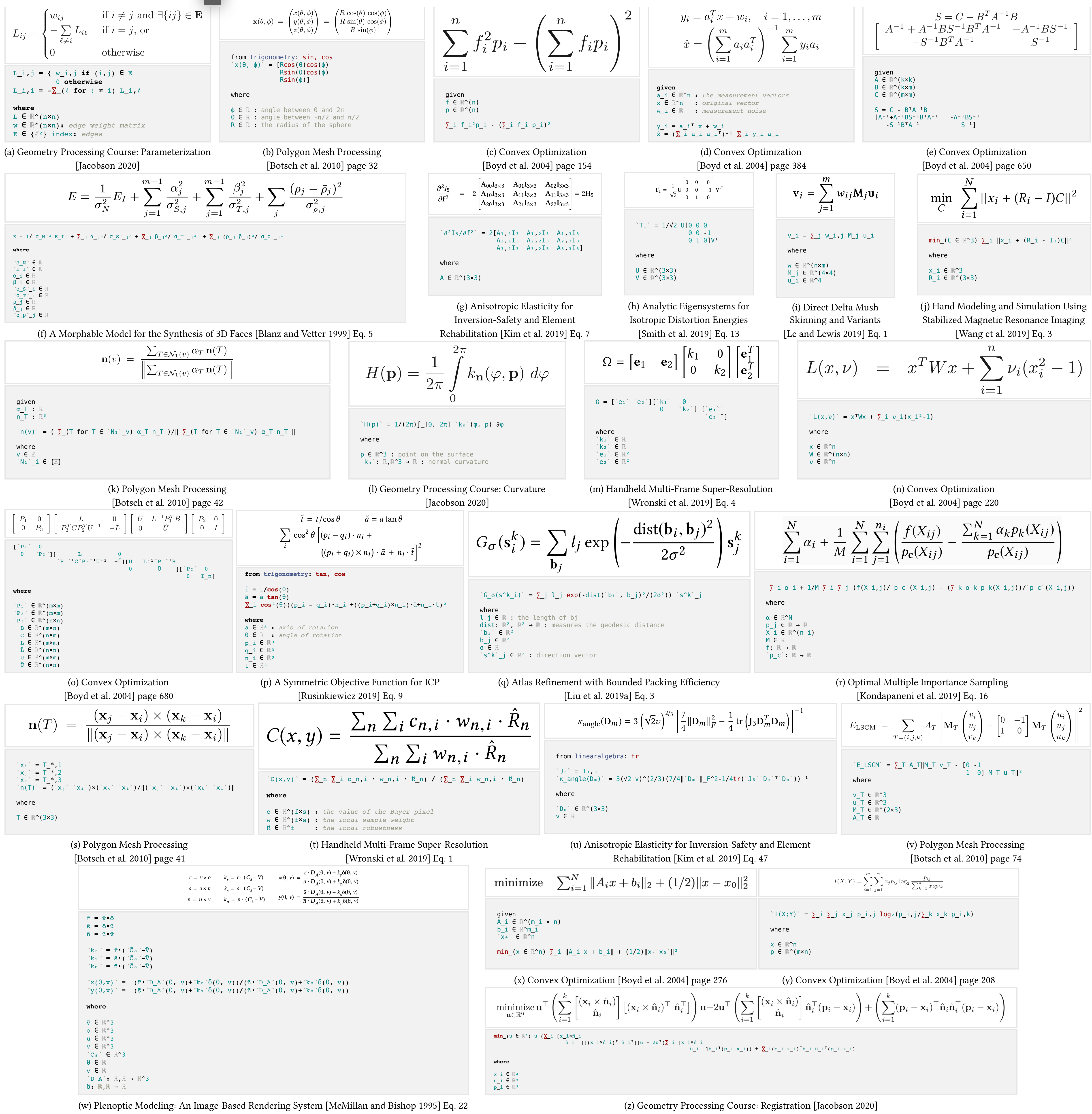
$$\mathbf{x}(\theta, \phi) = \begin{pmatrix} x(\\ y(\\ z(\end{pmatrix}$$

```
from trigonometry:
`x(θ, ϕ)` = [Rcos(
             Rsin(
             Rsin(

where

ϕ ∈ ℝ : angle betw
θ ∈ ℝ : angle betw
R ∈ ℝ : the radius
```

(b) Polyg

[Botsch

Examples from the Wild



(a) Geometry Processing Course: Parameterization [Jacobson 2020]

(b) Polygon Mesh Processing [Botsch et al. 2010] page 32

(c) Convex Optimization [Boyd et al. 2004] page 154

(d) Convex Optimization [Boyd et al. 2004] page 384

(e) Convex Optimization [Boyd et al. 2004] page 650

(f) A Morphable Model for the Synthesis of 3D Faces [Blanz and Vetter 1999] Eq. 5

(g) Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation [Kim et al. 2019] Eq. 7

(h) Analytic Eigensystems for Isotropic Distortion Energies [Smith et al. 2019] Eq. 13

(i) Direct Delta Mush Skinning and Variants [Le and Lewis 2019] Eq. 1

(j) Hand Modeling and Simulation Using Stabilized Magnetic Resonance Imaging [Wang et al. 2019] Eq. 3

(k) Polygon Mesh Processing [Botsch et al. 2010] page 42

(l) Geometry Processing Course: Curvature [Jacobson 2020]

(m) Handheld Multi-Frame Super-Resolution [Wronski et al. 2019] Eq. 4

(n) Convex Optimization [Boyd et al. 2004] page 220

(o) Convex Optimization [Boyd et al. 2004] page 680

(p) A Symmetric Objective Function for ICP [Rusinkiewicz 2019] Eq. 9

(q) Atlas Refinement with Bounded Packing Efficiency [Liu et al. 2019a] Eq. 3

(r) Optimal Multiple Importance Sampling [Kondapaneni et al. 2019] Eq. 16

(s) Polygon Mesh Processing [Botsch et al. 2010] page 41

(t) Handheld Multi-Frame Super-Resolution [Wronski et al. 2019] Eq. 1

(u) Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation [Kim et al. 2019] Eq. 47

(v) Polygon Mesh Processing [Botsch et al. 2010] page 74

(w) Plenoptic Modeling: An Image-Based Rendering System [McMillan and Bishop 1995] Eq. 22

(x) Convex Optimization [Boyd et al. 2004] page 276

(y) Convex Optimization [Boyd et al. 2004] page 208

(z) Geometry Processing Course: Registration [Jacobson 2020]

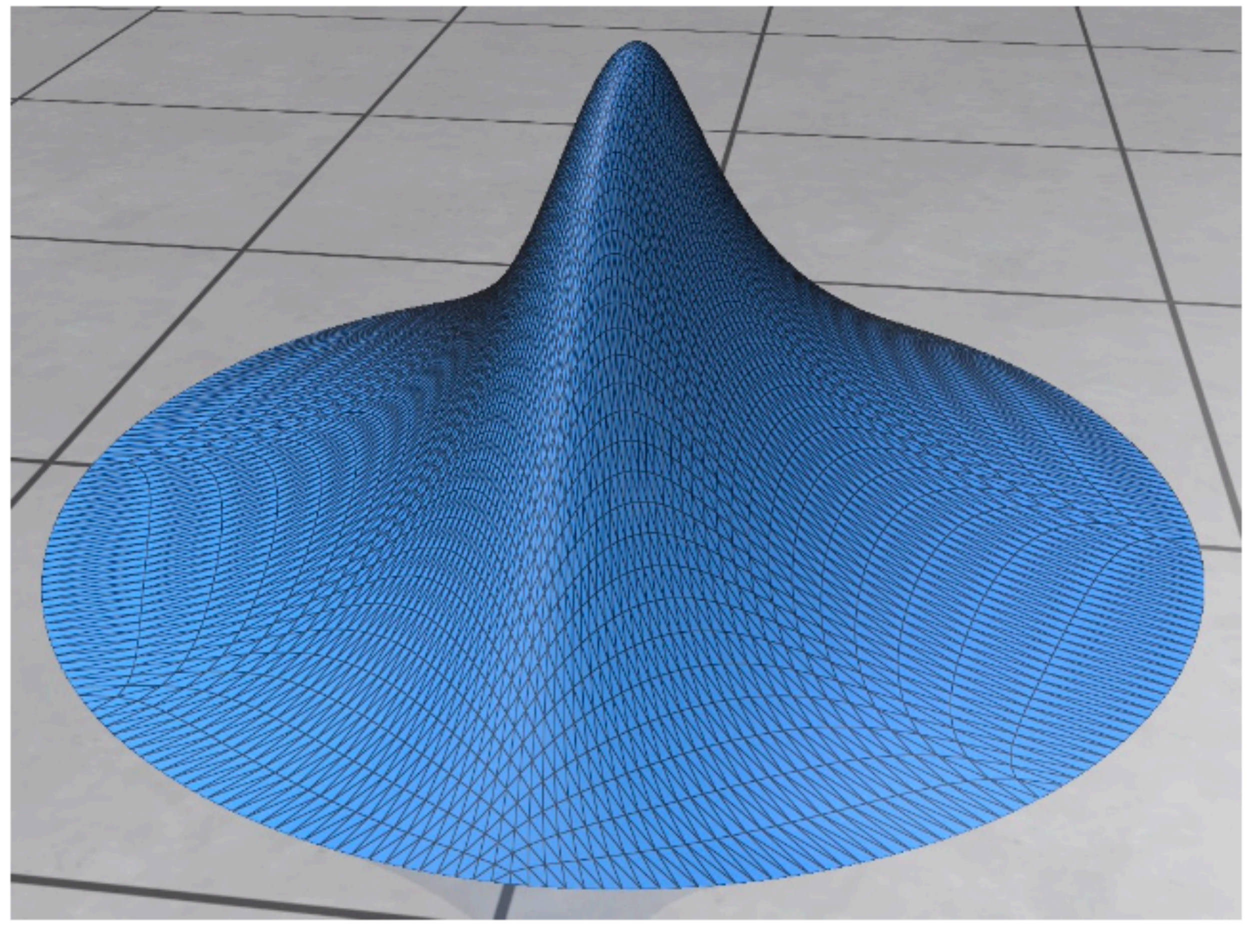
Integrating with Existing Code

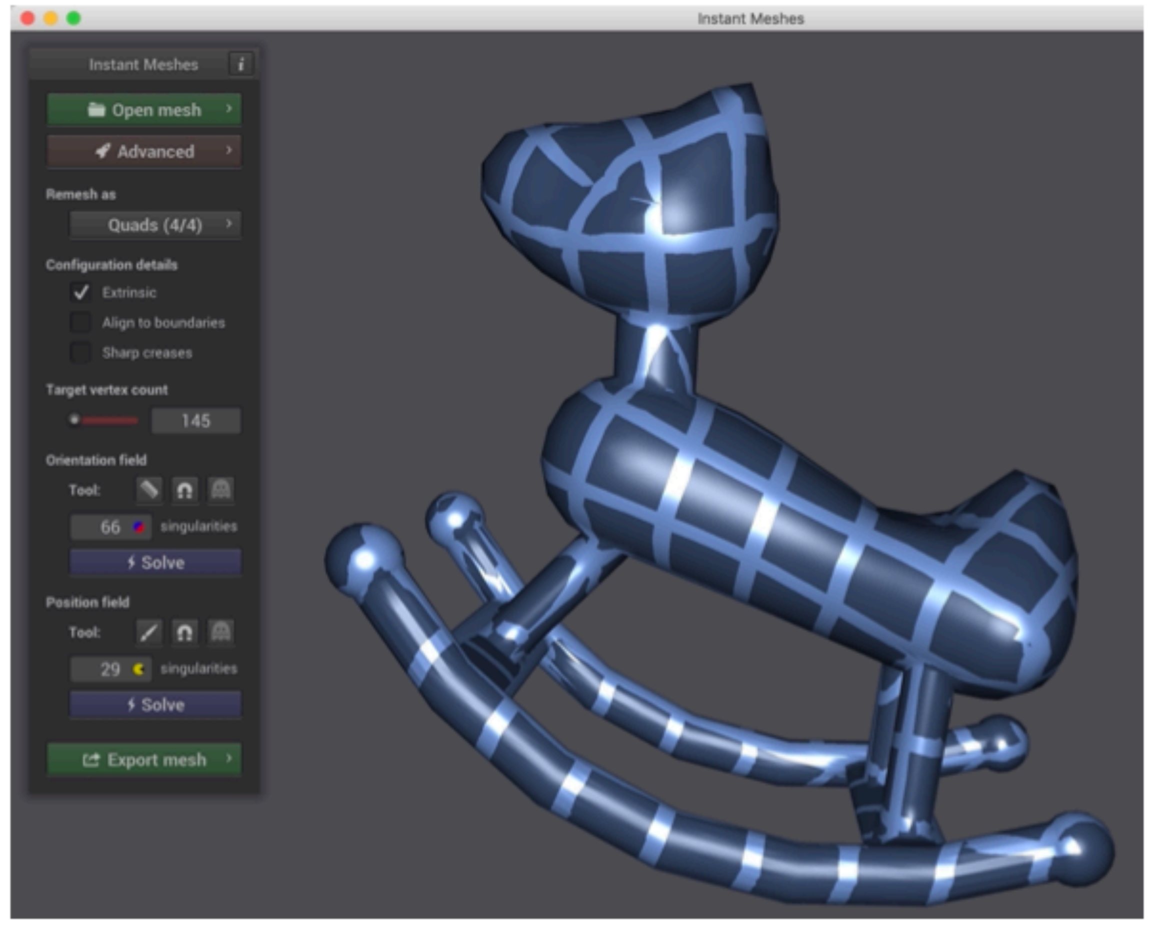
Integrating with Existing Code

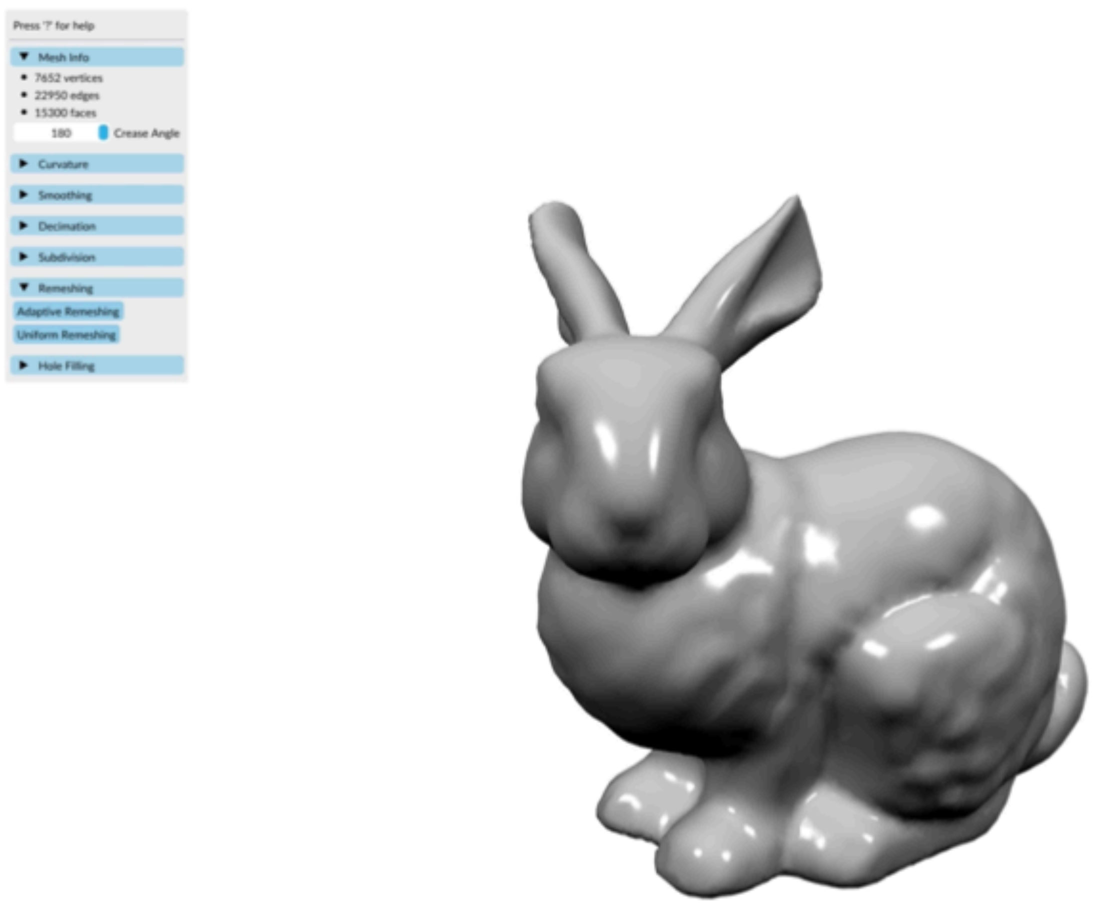
- Polygon Mesh Processing Library
- Conforming Weighted Delaunay Triangulations
- Properties of Laplace Operators for Tetrahedral Meshes_volume
- Properties of Laplace Operators for Tetrahedral Meshes_circumcenter
- Instant Field-Aligned Meshes
- Collision-Aware and Online Compression of Rigid Body Simulations via Integrated Error Minimization
- Frame Fields: Anisotropic and Non-Orthogonal Cross Fields
- Regularized Kelvinlets: Sculpting Brushes based on Fundamental Solutions of Elasticity
- Robust Inside-Outside Segmentation using Generalized Winding Numbers
- Gaussian-Product Subdivision Surfaces

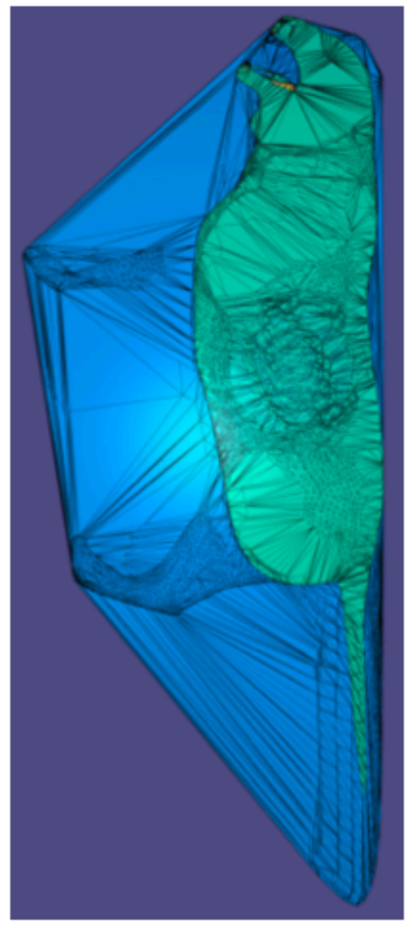
Integrating with Existing Code

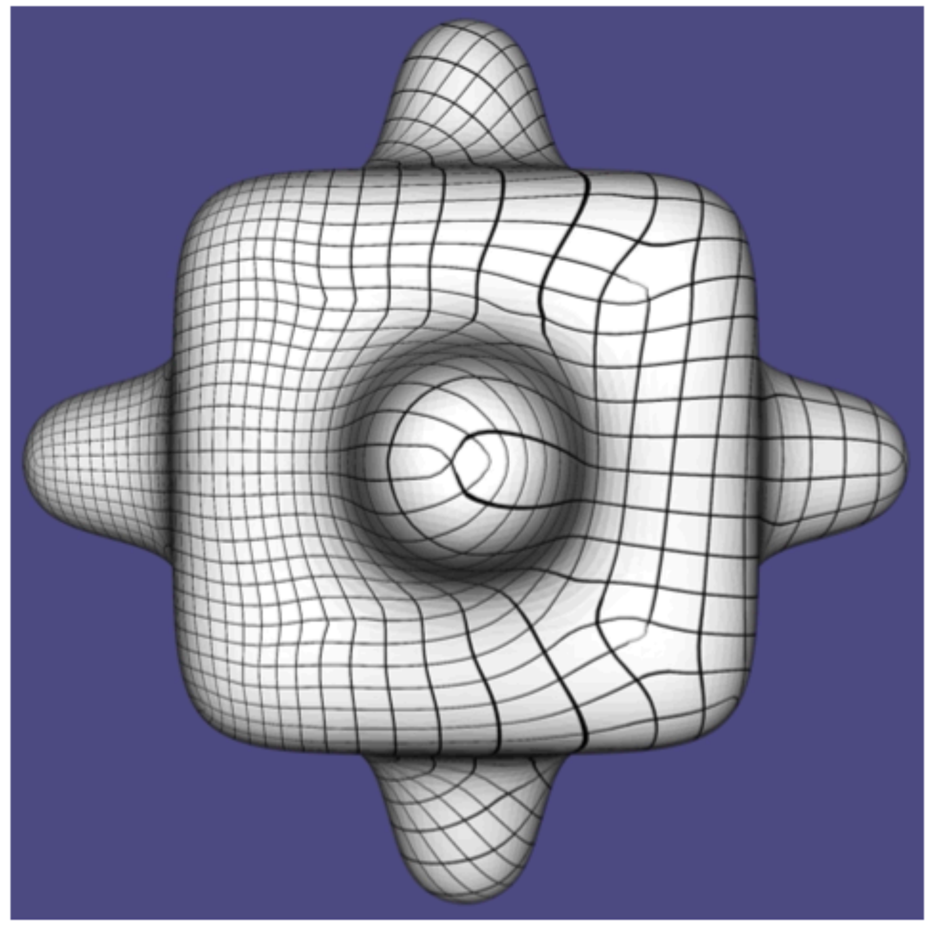
- Polygon Mesh Processing Library
- Conforming Weighted Delaunay Triangulations
- Properties of Laplace Operators for Tetrahedral Meshes_volume
- Properties of Laplace Operators for Tetrahedral Meshes_circumcenter
- Instant Field-Aligned Meshes
- Collision-Aware and Online Compression of Rigid Body Simulations via Integrated Error Minimization
- Frame Fields: Anisotropic and Non-Orthogonal Cross Fields
- Regularized Kelvinlets: Sculpting Brushes based on Fundamental Solutions of Elasticity
- Robust Inside-Outside Segmentation using Generalized Winding Numbers
- Gaussian-Product Subdivision Surfaces





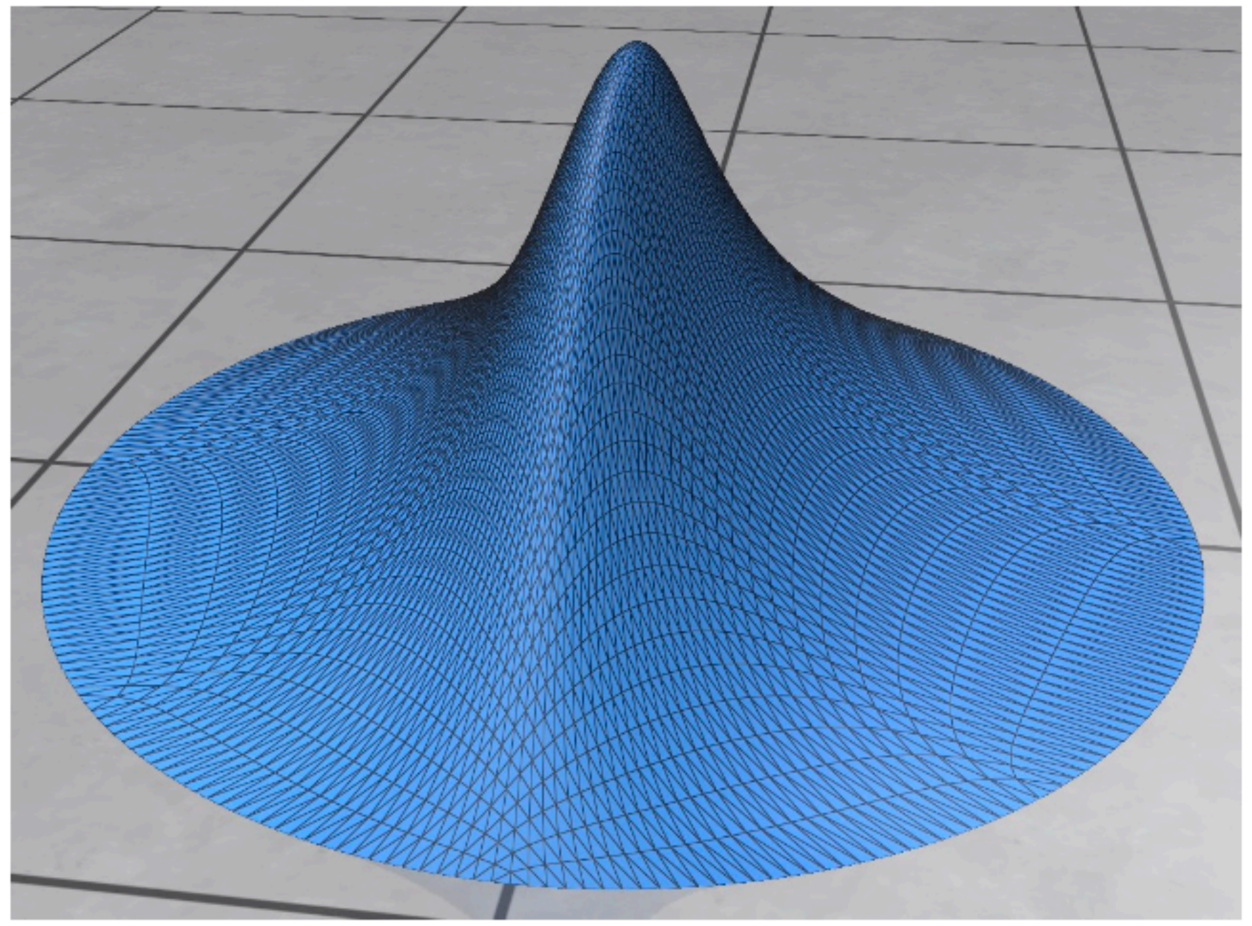


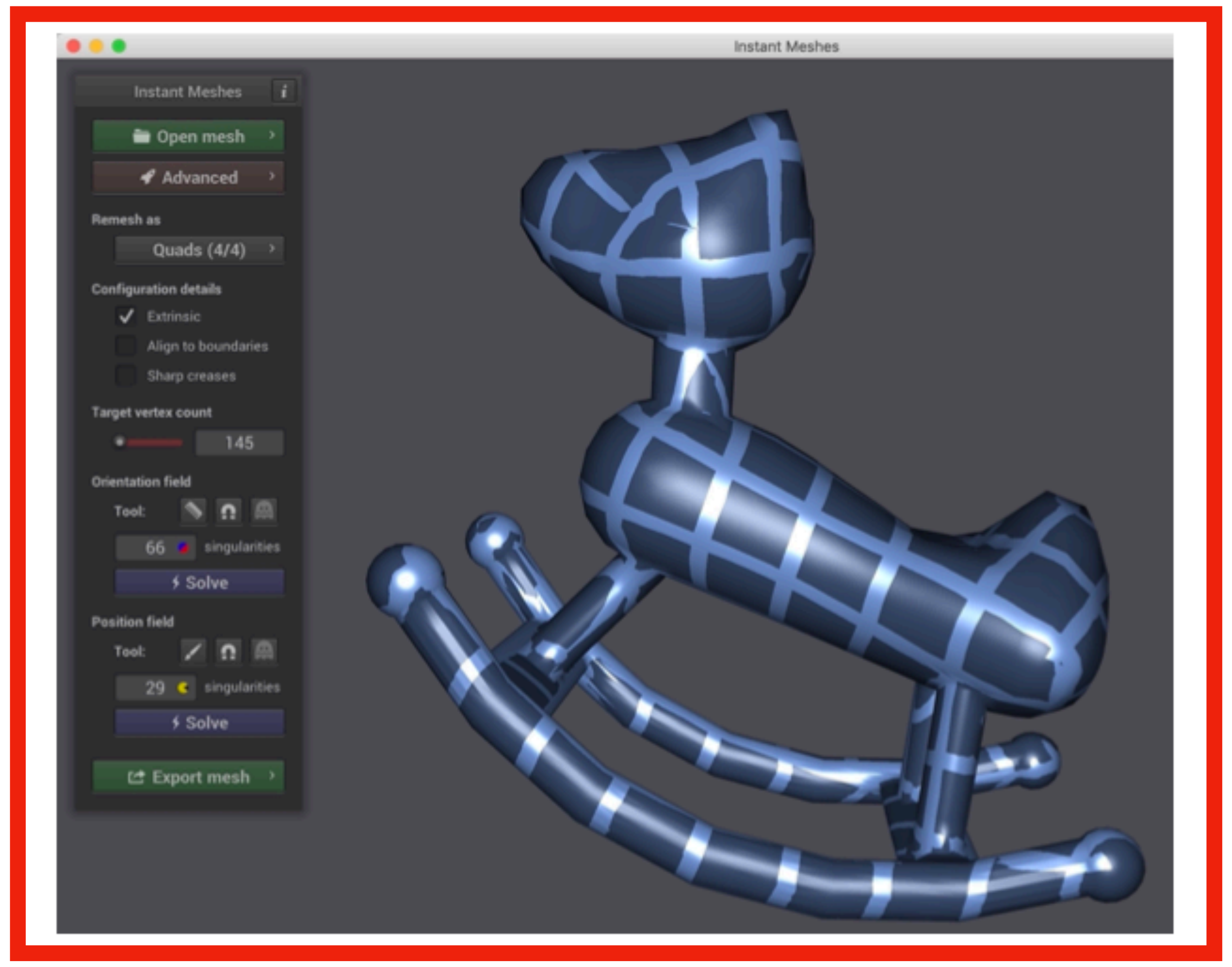


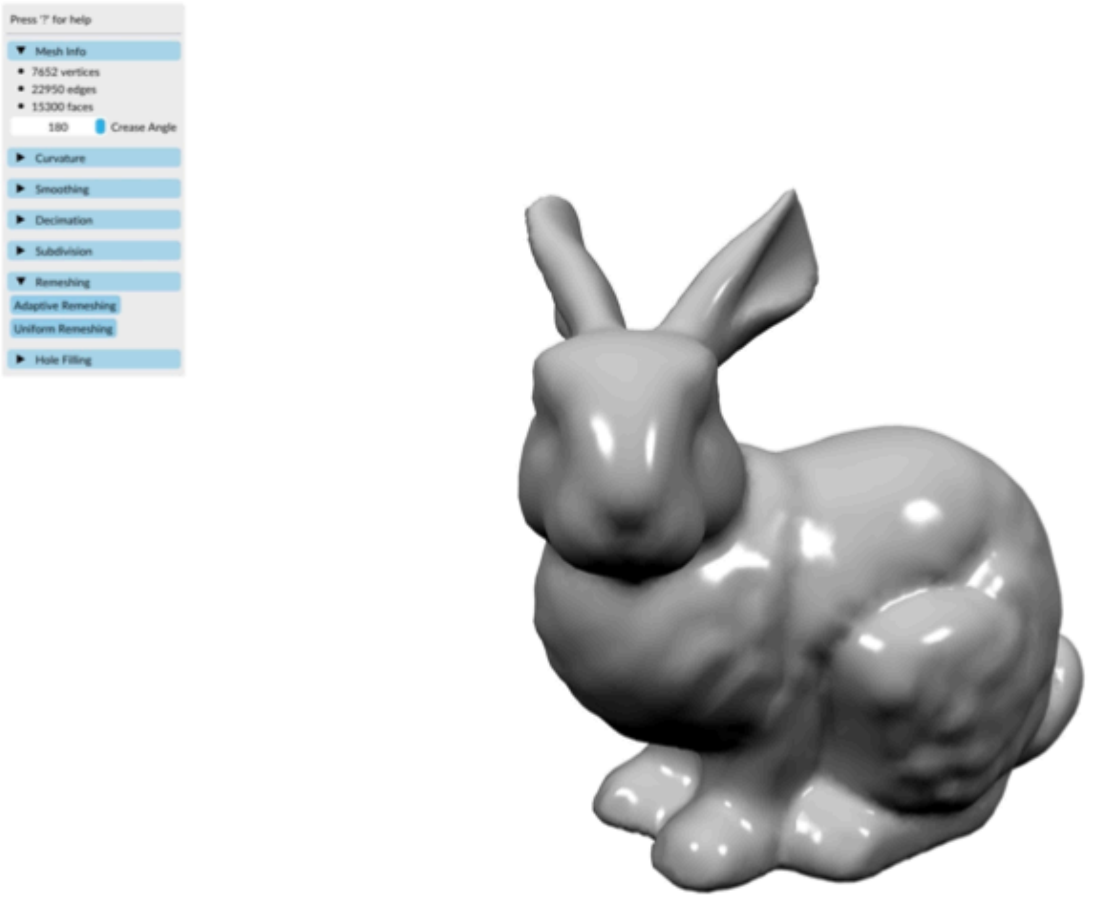


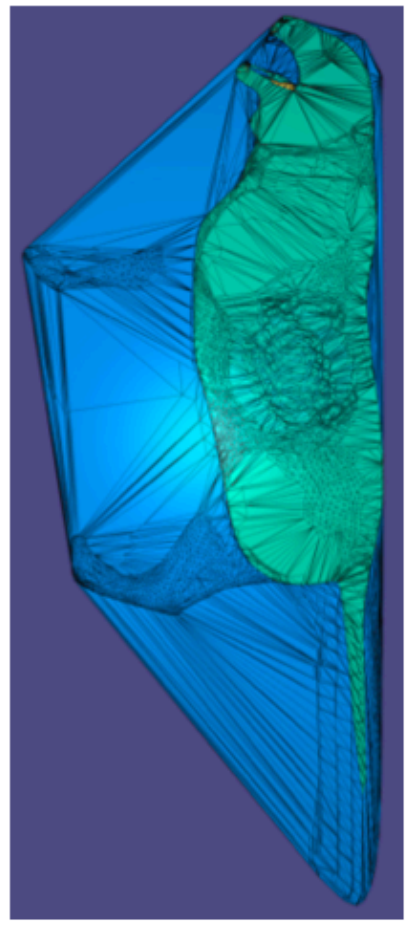
Integrating with Existing Code

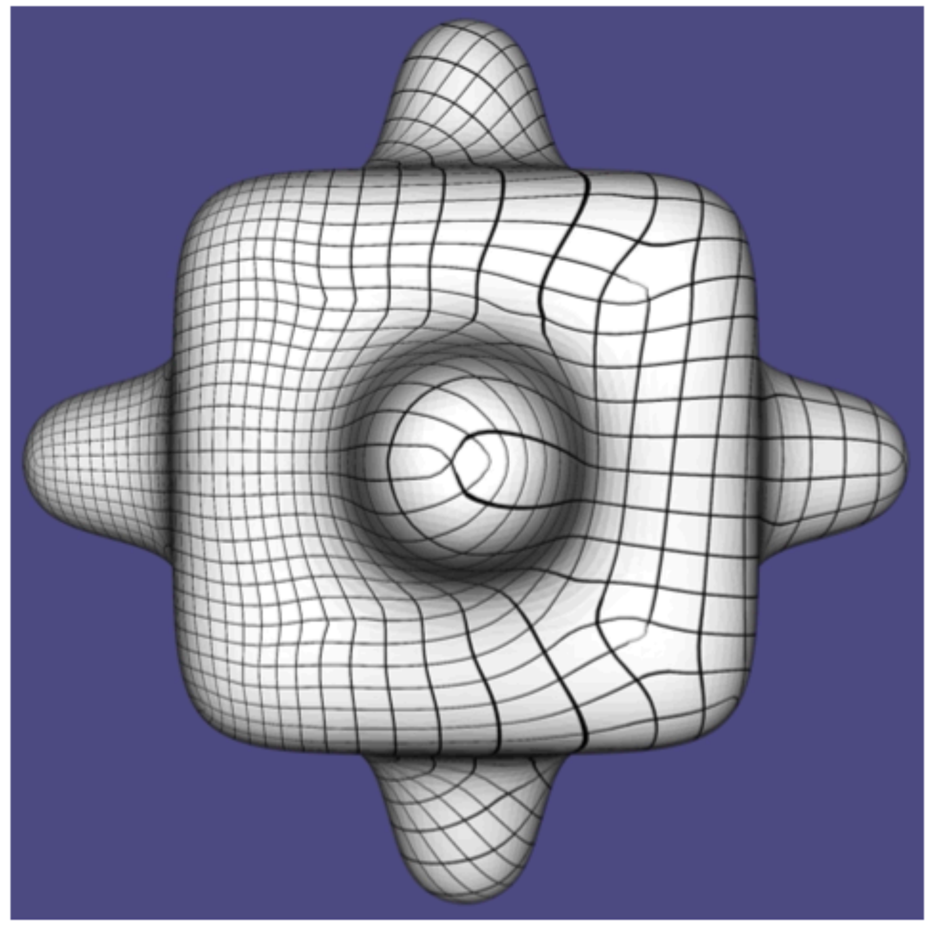
- Polygon Mesh Processing Library
- Conforming Weighted Delaunay Triangulations
- Properties of Laplace Operators for Tetrahedral Meshes_volume
- Properties of Laplace Operators for Tetrahedral Meshes_circumcenter
- Instant Field-Aligned Meshes
- Collision-Aware and Online Compression of Rigid Body Simulations via Integrated Error Minimization
- Frame Fields: Anisotropic and Non-Orthogonal Cross Fields
- Regularized Kelvinlets: Sculpting Brushes based on Fundamental Solutions of Elasticity
- Robust Inside-Outside Segmentation using Generalized Winding Numbers
- Gaussian-Product Subdivision Surfaces











Instant Field-Aligned Meshes

Integrating with existing code from Instant Field-Aligned Meshes.

Project URL: Instant Field-Aligned Meshes

The original formula:

Intermediate position. We define a position $\mathbf{q}_{ij}$ that minimizes the distance to vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ while being located in their respective tangent planes, i.e.:

$$\begin{aligned}&\text{minimize} \quad \|\mathbf{v}_i - \mathbf{q}_{ij}\|_2^2 + \|\mathbf{v}_j - \mathbf{q}_{ij}\|_2^2 \\ &\text{subject to} \quad \langle \mathbf{n}_i, \mathbf{q}_{ij} \rangle = \langle \mathbf{n}_i, \mathbf{v}_i \rangle \ \text{ and } \ \langle \mathbf{n}_j, \mathbf{q}_{ij} \rangle = \langle \mathbf{n}_j, \mathbf{v}_j \rangle.\end{aligned}$$

This constrained least-squares problem has a simple solution:

$$\mathbf{q}_{ij} = \tfrac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) - \tfrac{1}{4}(\lambda_i \mathbf{n}_i + \lambda_j \mathbf{n}_j),$$

where the Lagrange multiplier λ_i is

$$\lambda_i = \frac{2 \left\langle (\mathbf{n}_i + \langle \mathbf{n}_i, \mathbf{n}_j \rangle \mathbf{n}_j, \ \mathbf{v}_j - \mathbf{v}_i \right\rangle}{1 - \langle \mathbf{n}_i, \mathbf{n}_j \rangle^2 + \varepsilon},$$

and λ_j is defined analogously with i and j swapped. The parameter ε (set to 10^{-4} in our implementation) ensures that $\mathbf{q}_{ij}$ approximates the arithmetic mean of $\mathbf{v}_i$ and $\mathbf{v}_j$ when $\mathbf{n}_i \approx \mathbf{n}_j$.

Instant Field-Aligned Meshes

Integrating with existing code from Instant Field-Aligned Meshes.

Project URL: Instant Field-Aligned Meshes

The original formula:

Intermediate position. We define a position $\mathbf{q}_{ij}$ that minimizes the distance to vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ while being located in their respective tangent planes, i.e.:

$$\begin{aligned} &\text{minimize} \;\; \|\mathbf{v}_i - \mathbf{q}_{ij}\|_2^2 + \|\mathbf{v}_j - \mathbf{q}_{ij}\|_2^2 \\ &\text{subject to} \;\; \langle \mathbf{n}_i, \mathbf{q}_{ij} \rangle = \langle \mathbf{n}_i, \mathbf{v}_i \rangle \;\; \text{and} \;\; \langle \mathbf{n}_j, \mathbf{q}_{ij} \rangle = \langle \mathbf{n}_j, \mathbf{v}_j \rangle. \end{aligned}$$

This constrained least-squares problem has a simple solution:

$$\mathbf{q}_{ij} = \tfrac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) - \tfrac{1}{4}(\lambda_i \mathbf{n}_i + \lambda_j \mathbf{n}_j),$$

where the Lagrange multiplier λ_i is

$$\lambda_i = \frac{2\,\langle (\mathbf{n}_i + \langle \mathbf{n}_i, \mathbf{n}_j \rangle \mathbf{n}_j,\; \mathbf{v}_j - \mathbf{v}_i \rangle}{1 - \langle \mathbf{n}_i, \mathbf{n}_j \rangle^2 + \varepsilon},$$

The original source code:

field.cpp — instant-meshes

```cpp
    return q;
}

inline Vector3f middle_point(const Vector3f &p0, const Vector3f &n0, const Vector3f &p1, const Vector3f &n1) {
    /* How was this derived?
     *
     * Minimize \|x-p0\|^2 + \|x-p1\|^2, where
     * dot(n0, x) == dot(n0, p0)
     * dot(n1, x) == dot(n1, p1)
     *
     * -> Lagrange multipliers, set derivative = 0
     *  Use first 3 equalities to write x in terms of
     *  lambda_1 and lambda_2. Substitute that into the last
     *  two equations and solve for the lambdas. Finally,
     *  add a small epsilon term to avoid issues when n1=n2.
     */
    Float n0p0 = n0.dot(p0), n0p1 = n0.dot(p1),
          n1p0 = n1.dot(p0), n1p1 = n1.dot(p1),
          n0n1 = n0.dot(n1),
          denom = 1.0f / (1.0f - n0n1*n0n1 + 1e-4f),
          lambda_0 = 2.0f*(n0p1 - n0p0 - n0n1*(n1p0 - n1p1))*denom,
          lambda_1 = 2.0f*(n1p0 - n1p1 - n0n1*(n0p1 - n0p0))*denom;

    return 0.5f * (p0 + p1) - 0.25f * (n0 * lambda_0 + n1 * lambda_1);
}

std::pair<Vector3f, Vector3f> compat_orientation_intrinsic_2(
    const Vector3f &q0, const Vector3f &n0, const Vector3f &_q1, const Vector3f &n1) {
    const Vector3f q1 = rotate_vector_into_plane(_q1, n1, n0);
    return std::make_pair(q0, q1 * signum(q1.dot(q0)));
}
```

FOLDERS
- instant-meshes
  - build
  - ext
  - resources
  - src
    - aabb.h
    - adjacency.cpp
    - adjacency.h
    - batch.cpp
    - batch.h
    - bvh.cpp
    - bvh.h
    - cleanup.cpp
    - cleanup.h
    - common.h
    - dedge.cpp
    - dedge.h
    - diff.cpp
    - extract.cpp
    - extract.h
    - field.cpp

Instant Field-Aligned Meshes

Integrating with existing code from Instant Field-Aligned Meshes.

Project URL: Instant Field-Aligned Meshes

The original formula:

Intermediate position. We define a position $\mathbf{q}_{ij}$ that minimizes the distance to vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ while being located in their respective tangent planes, i.e.:

$$\begin{aligned}&\text{minimize} \;\; \|\mathbf{v}_i - \mathbf{q}_{ij}\|_2^2 + \|\mathbf{v}_j - \mathbf{q}_{ij}\|_2^2\\ &\text{subject to} \;\; \langle \mathbf{n}_i, \mathbf{q}_{ij}\rangle = \langle \mathbf{n}_i, \mathbf{v}_i\rangle \;\text{ and }\; \langle \mathbf{n}_j, \mathbf{q}_{ij}\rangle = \langle \mathbf{n}_j, \mathbf{v}_j\rangle.\end{aligned}$$

This constrained least-squares problem has a simple solution:

$$\mathbf{q}_{ij} = \tfrac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) - \tfrac{1}{4}(\lambda_i \mathbf{n}_i + \lambda_j \mathbf{n}_j),$$

where the Lagrange multiplier λ_i is

$$\lambda_i = \frac{2\left\langle (\mathbf{n}_i + \langle \mathbf{n}_i, \mathbf{n}_j\rangle \mathbf{n}_j,\; \mathbf{v}_j - \mathbf{v}_i\right\rangle}{1 - \langle \mathbf{n}_i, \mathbf{n}_j\rangle^2 + \varepsilon},$$

I ❤️ LA source code:

```
ϵ = 10^(-4)
`λᵢ` = (2⟨`nᵢ`+⟨`nᵢ`,`nⱼ`⟩`nⱼ`, `vⱼ`-`vᵢ`⟩)/(1-⟨`nᵢ`, `nⱼ`⟩²+ϵ)
`λⱼ` = (2⟨`nⱼ`+⟨`nⱼ`,`nᵢ`⟩`nᵢ`, `vᵢ`-`vⱼ`⟩)/(1-⟨`nⱼ`, `nᵢ`⟩²+ϵ)
`qᵢⱼ` = 1/2(`vᵢ`+`vⱼ`)-1/4(`λᵢ``nᵢ`+`λⱼ``nⱼ`)

where

`vᵢ` ∈ ℝ^3
`nᵢ` ∈ ℝ^3
`vⱼ` ∈ ℝ^3
`nⱼ` ∈ ℝ^3
```

The original source code:

field.cpp — instant-meshes

FOLDERS
- instant-meshes
  - build
  - ext
  - resources
  - src
    - aabb.h
    - adjacency.cpp
    - adjacency.h
    - batch.cpp
    - batch.h
    - bvh.cpp
    - bvh.h
    - cleanup.cpp
    - cleanup.h
    - common.h
    - dedge.cpp
    - dedge.h
    - diff.cpp
    - extract.cpp
    - extract.h
    - field.cpp

field.cpp

```cpp
    return q;
}

inline Vector3f middle_point(const Vector3f &p0, const Vector3f &n0, const Vector3f &p1, const Vector3f &n1) {
    /* How was this derived?
     *
     * Minimize \|x-p0\|^2 + \|x-p1\|^2, where
     * dot(n0, x) == dot(n0, p0)
     * dot(n1, x) == dot(n1, p1)
     *
     * -> Lagrange multipliers, set derivative = 0
     *  Use first 3 equalities to write x in terms of
     *  lambda_1 and lambda_2. Substitute that into the last
     *  two equations and solve for the lambdas. Finally,
     *  add a small epsilon term to avoid issues when n1=n2.
     */
    Float n0p0 = n0.dot(p0), n0p1 = n0.dot(p1),
          n1p0 = n1.dot(p0), n1p1 = n1.dot(p1),
          n0n1 = n0.dot(n1),
          denom = 1.0f / (1.0f - n0n1*n0n1 + 1e-4f),
          lambda_0 = 2.0f*(n0p1 - n0p0 - n0n1*(n1p0 - n1p1))*denom,
          lambda_1 = 2.0f*(n1p0 - n1p1 - n0n1*(n0p1 - n0p0))*denom;

    return 0.5f * (p0 + p1) - 0.25f * (n0 * lambda_0 + n1 * lambda_1);
}

std::pair<Vector3f, Vector3f> compat_orientation_intrinsic_2(
    const Vector3f &q0, const Vector3f &n0, const Vector3f &_q1, const Vector3f &n1) {
    const Vector3f q1 = rotate_vector_into_plane(_q1, n1, n0);
    return std::make_pair(q0, q1 * signum(q1.dot(q0)));
}
```

Instant Field-Aligned Meshes

Integrating with existing code from Instant Field-Aligned Meshes.

Project URL: [Instant Field-Aligned Meshes]

The original formula:

Intermediate position. We define a position $\mathbf{q}_{ij}$ that minimizes the distance to vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ while being located in their respective tangent planes, i.e.:

$$\begin{aligned} &\text{minimize} \quad \|\mathbf{v}_i - \mathbf{q}_{ij}\|_2^2 + \|\mathbf{v}_j - \mathbf{q}_{ij}\|_2^2 \\ &\text{subject to} \quad \langle \mathbf{n}_i, \mathbf{q}_{ij} \rangle = \langle \mathbf{n}_i, \mathbf{v}_i \rangle \ \text{ and } \ \langle \mathbf{n}_j, \mathbf{q}_{ij} \rangle = \langle \mathbf{n}_j, \mathbf{v}_j \rangle. \end{aligned}$$

This constrained least-squares problem has a simple solution:

$$\mathbf{q}_{ij} = \tfrac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) - \tfrac{1}{4}(\lambda_i \mathbf{n}_i + \lambda_j \mathbf{n}_j),$$

where the Lagrange multiplier λ_i is

$$\lambda_i = \frac{2\,\langle(\mathbf{n}_i + \langle \mathbf{n}_i, \mathbf{n}_j\rangle \mathbf{n}_j,\ \mathbf{v}_j - \mathbf{v}_i\rangle}{1 - \langle \mathbf{n}_i, \mathbf{n}_j\rangle^2 + \varepsilon},$$

I ❤️ LA source code:

```
ϵ = 10^(-4)
`λᵢ` = (2⟨`nᵢ`+⟨`nᵢ`,`nⱼ`⟩`nⱼ`, `vⱼ`-`vᵢ`⟩)/(1-⟨`nᵢ`, `nⱼ`⟩²+ϵ)
`λⱼ` = (2⟨`nⱼ`+⟨`nⱼ`,`nᵢ`⟩`nᵢ`, `vᵢ`-`vⱼ`⟩)/(1-⟨`nⱼ`, `nᵢ`⟩²+ϵ)
`qᵢⱼ` = 1/2(`vᵢ`+`vⱼ`)-1/4(`λᵢ``nᵢ`+`λⱼ``nⱼ`)

where

`v
`n
`v
`n
```

The original source code:

field.cpp — instant-meshes

FOLDERS
- instant-meshes
  - build
  - ext
  - resources
  - src
    - aabb.h
    - adjacency.cpp
    - adjacency.h
    - batch.cpp
    - batch.h
    - bvh.cpp
    - bvh.h
    - cleanup.cpp
    - cleanup.h
    - common.h
    - dedge.cpp
    - dedge.h
    - diff.cpp
    - extract.cpp
    - extract.h
    - field.cpp

field.cpp

```cpp
    return q;
}

inline Vector3f middle_point(const Vector3f &p0, const Vector3f &n0, const Vector3f &p1, const Vector3f &n1) {
    /* How was this derived?
     *
     * Minimize \|x-p0\|^2 + \|x-p1\|^2, where
     * dot(n0, x) == dot(n0, p0)
     * dot(n1, x) == dot(n1, p1)
     *
     * -> Lagrange multipliers, set derivative = 0
     *  Use first 3 equalities to write x in terms of
     *  lambda_1 and lambda_2. Substitute that into the last
     *  two equations and solve for the lambdas. Finally,
     *  add a small epsilon term to avoid issues when n1=n2.
     */
    Float n0p0 = n0.dot(p0), n0p1 = n0.dot(p1),
          n1p0 = n1.dot(p0), n1p1 = n1.dot(p1),
          n0n1 = n0.dot(n1),
          denom = 1.0f / (1.0f - n0n1*n0n1 + 1e-4f),
          lambda_0 = 2.0f*(n0p1 - n0p0 - n0n1*(n1p0 - n1p1))*denom,
          lambda_1 = 2.0f*(n1p0 - n1p1 - n0n1*(n0p1 - n0p0))*denom;

    return 0.5f * (p0 + p1) - 0.25f * (n0 * lambda_0 + n1 * lambda_1);
}

std::pair<Vector3f, Vector3f> compat_orientation_intrinsic_2(
    const Vector3f &q0, const Vector3f &n0, const Vector3f &_q1, const Vector3f &n1) {
    const Vector3f q1 = rotate_vector_into_plane(_q1, n1, n0);
    return std::make_pair(q0, q1 * signum(q1.dot(q0)));
}
```

The modified source code:

field.cpp — instant-meshes

FOLDERS
- instant-meshes
  - bb
  - build
  - ext
  - resources
  - src
    - aabb.h
    - adjacency.cpp
    - adjacency.h
    - batch.cpp
    - batch.h
    - bvh.cpp
    - bvh.h
    - cleanup.cpp
    - cleanup.h
    - common.h
    - dedge.cpp
    - dedge.h
    - diff.cpp
    - extract.cpp
    - extract.h
    - field.cpp
    - field.cpp.bkup
    - field.cpp.modified
    - field.h
    - glutil.cpp
    - glutil.h
    - gui_serializer.h
    - hierarchy.cpp
    - hierarchy.h
    - main.cpp
    - meshio.cpp
    - meshio.h
    - meshstats.cpp
    - meshstats.h
    - normal.cpp
    - normal.h

field.cpp

```cpp
}

struct myExpressionResultType {
    double ε;
    double λᵢ;
    double λⱼ;
    Eigen::Matrix<double, 3, 1> qᵢⱼ;
    myExpressionResultType(const double & ε,
                const double & λᵢ,
                const double & λⱼ,
                const Eigen::Matrix<double, 3, 1> & qᵢⱼ)
    : ε(ε),
    λᵢ(λᵢ),
    λⱼ(λⱼ),
    qᵢⱼ(qᵢⱼ)
    {}
};

myExpressionResultType myExpression(
    const Eigen::Matrix<double, 3, 1> & vᵢ,
    const Eigen::Matrix<double, 3, 1> & nᵢ,
    const Eigen::Matrix<double, 3, 1> & vⱼ,
    const Eigen::Matrix<double, 3, 1> & nⱼ)
{
    double ε = pow(10, (-4));

    double λᵢ = (2 * (nᵢ + (nᵢ).dot(nⱼ) * nⱼ).dot(vⱼ - vᵢ)) / double((1 - pow((nᵢ).dot(nⱼ), 2) + ε));

    double λⱼ = (2 * (nⱼ + (nⱼ).dot(nᵢ) * nᵢ).dot(vᵢ - vⱼ)) / double((1 - pow((nⱼ).dot(nᵢ), 2) + ε));

    Eigen::Matrix<double, 3, 1> qᵢⱼ = 1 / double(2) * (vᵢ + vⱼ) - 1 / double(4) * (λᵢ * nᵢ + λⱼ * nⱼ);

    return myExpressionResultType(ε, λᵢ, λⱼ, qᵢⱼ);
}

inline Vector3f middle_point(const Vector3f &p0, const Vector3f &n0, const Vector3f &p1, const Vector3f &n1) {
    Eigen::Matrix<double, 3, 1> ret = myExpression(p0.cast <double>() , n0.cast <double>(),
        p1.cast <double>(), n1.cast <double>() ).qᵢⱼ;
    return ret.cast <float>();
}

std::pair<Vector3f, Vector3f> compat_orientation_intrinsic_2(
    const Vector3f &q0, const Vector3f &n0, const Vector3f &_q1, const Vector3f &n1) {
    const Vector3f q1 = rotate_vector_into_plane(_q1, n1, n0);
    return std::make_pair(q0, q1 * signum(q1.dot(q0)));
}

std::pair<Vector3f, Vector3f> compat_orientation_intrinsic_4(
```

Instant Field-Aligned Meshes

Wenzel Jakob[1] Marco Tarini[2,3] Daniele Panozzo[1] Olga Sorkine-Hornung[1]

[1]ETH Zurich [2]CNR-ISTI [3]Università dell'Insubria

Figure 1: *Remeshing a scanned dragon with 13 million vertices into feature-aligned isotropic triangle and quad meshes with ∼80k vertices. From left to right, for both cases: visualizations of the orientation field, position field, and the output mesh (computed in 71.1 and 67.2 seconds, respectively). For the quad case, we optimize for a quad-dominant mesh at quarter resolution and subdivide once to obtain a pure quad mesh.*

Abstract

We present a novel approach to remesh a surface into an isotropic triangular or quad-dominant mesh using a unified local smoothing operator that optimizes both the edge orientations and vertex positions in the output mesh. Our algorithm produces meshes with high isotropy while naturally aligning and snapping edges to sharp features. The method is simple to implement and parallelize, and it can process a variety of input surface representations, such as point clouds, range scans and triangle meshes. Our full pipeline executes instantly (less than a second) on meshes with hundreds of thousands of faces, enabling new types of interactive workflows. Since our algorithm avoids any global optimization, and its key steps scale linearly with input size, we are able to process extremely large meshes and point clouds, with sizes exceeding several hundred

1 Introduction

Triangle and quad-dominant meshes are ubiquitously used in computer graphics and CAD applications to represent surfaces, either directly, or as the control grid for higher-order parametric or subdivision surfaces. With the introduction of T-splines [Sederberg et al. 2003] and Dyadic T-Mesh Subdivision [Kovacs et al. 2015], quad-dominant meshes with T-joints (T-meshes) now have similar properties and applications of pure quadrilateral meshes, while being more flexible and naturally supporting the flexible local refinement that is often desired in CAD applications. Meshing surfaces is a challenging problem, and a plethora of methods have been proposed in the past three decades to cope with the increasing quality and scalability requirements of modern applications [Owen 1998; Bommes et al. 2013a]. Semi-regular meshes, which have uniform element

```
l, const Vector3f &target_normal) {
Vector3f &p1, const Vector3f &n1) {
tor3f &n1) {
tor3f &n1) {
```

INSERT MODE, Line 9, Column 1
INSERT MODE, 12 characters selected

instant-meshes-SA-2015-jakob-et-al
Page 1 of 15

Instant Field-Aligned Meshes

Wenzel Jakob[1] Marco Tarini[2,3] Daniele Panozzo[1] Olga Sorkine-Hornung[1]

[1]ETH Zurich [2]CNR-ISTI [3]Università dell'Insubria

Figure 1: *Remeshing a scanned dragon with 13 million vertices into feature-aligned isotropic triangle and quad meshes with ∼80k vertices. From left to right, for both cases: visualizations of the orientation field, position field, and the output mesh (computed in 71.1 and 67.2 seconds, respectively). For the quad case, we optimize for a quad-dominant mesh at quarter resolution and subdivide once to obtain a pure quad mesh.*

Abstract

We present a novel approach to remesh a surface into an isotropic triangular or quad-dominant mesh using a unified local smoothing operator that optimizes both the edge orientations and vertex positions in the output mesh. Our algorithm produces meshes with high isotropy while naturally aligning and snapping edges to sharp features. The method is simple to implement and parallelize, and it can process a variety of input surface representations, such as point clouds, range scans and triangle meshes. Our full pipeline executes instantly (less than a second) on meshes with hundreds of thousands of faces, enabling new types of interactive workflows. Since our algorithm avoids any global optimization, and its key steps scale linearly with input size, we are able to process extremely large meshes and point clouds, with sizes exceeding several hundred

1 Introduction

Triangle and quad-dominant meshes are ubiquitously used in computer graphics and CAD applications to represent surfaces, either directly, or as the control grid for higher-order parametric or subdivision surfaces. With the introduction of T-splines [Sederberg et al. 2003] and Dyadic T-Mesh Subdivision [Kovacs et al. 2015], quad-dominant meshes with T-joints (T-meshes) now have similar properties and applications of pure quadrilateral meshes, while being more flexible and naturally supporting the flexible local refinement that is often desired in CAD applications. Meshing surfaces is a challenging problem, and a plethora of methods have been proposed in the past three decades to cope with the increasing quality and scalability requirements of modern applications [Owen 1998; Bommes et al. 2013a]. Semi-regular meshes, which have uniform element

```
, const Vector3f &target_normal) {
Vector3f &p1, const Vector3f &n1) {
or3f &n1) {
or3f &n1) {
```

INSERT MODE, Line 9, Column 1 INSERT MODE, 12 characters selected

Integrating with Existing Code

Source	Language	LoC (original)	LoC (I❤️LA)
Jacobson et al. [2013]	C++	31	8
Sieger and Botsch [2020]	C++	26	9
Alexa [2020]	C++	21	8
Preiner et al. [2019]	Python	15	9
Panozzo et al. [2014]	C++	14	5
Alexa et al. [2020]	C++	12	6
Jakob et al. [2015]	C++	7	4
De Goes and James [2017]	C++	6	1
Jeruzalski et al. [2018]	C++	5	2

Integrating with Existing Code

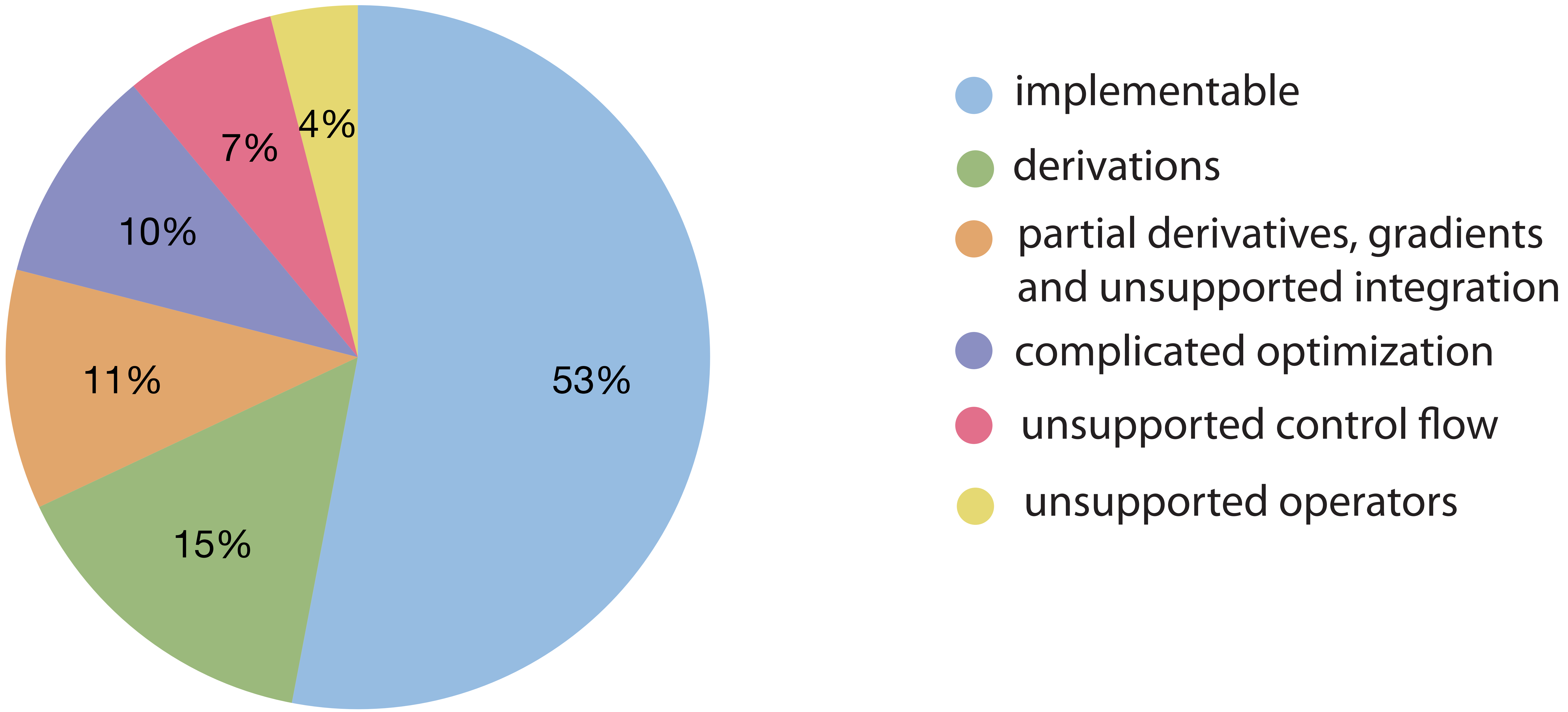
Source	Language	LoC (original)	LoC (I❤️LA)
Jacobson et al. [2013]	C++	31	8
Sieger and Botsch [2020]	C++	26	9
Alexa [2020]	C++	21	8
Preiner et al. [2019]	Python	15	9
Panozzo et al. [2014]	C++	14	5
Alexa et al. [2020]	C++	12	6
Jakob et al. [2015]	C++	7	4
De Goes and James [2017]	C++	6	1
Jeruzalski et al. [2018]	C++	5	2

Integrating with Existing Code

Source	Language	LoC (original)	LoC (I❤️LA)
Jacobson et al. [2013]	C++	31	8
Sieger and Botsch [2020]	C++	26	9
Alexa [2020]	C++	21	8
Preiner et al. [2019]	Python	15	9
Panozzo et al. [2014]	C++	14	5
Alexa et al. [2020]	C++	12	6
Jakob et al. [2015]	C++	7	4
De Goes and James [2017]	C++	6	1
Jeruzalski et al. [2018]	C++	5	2

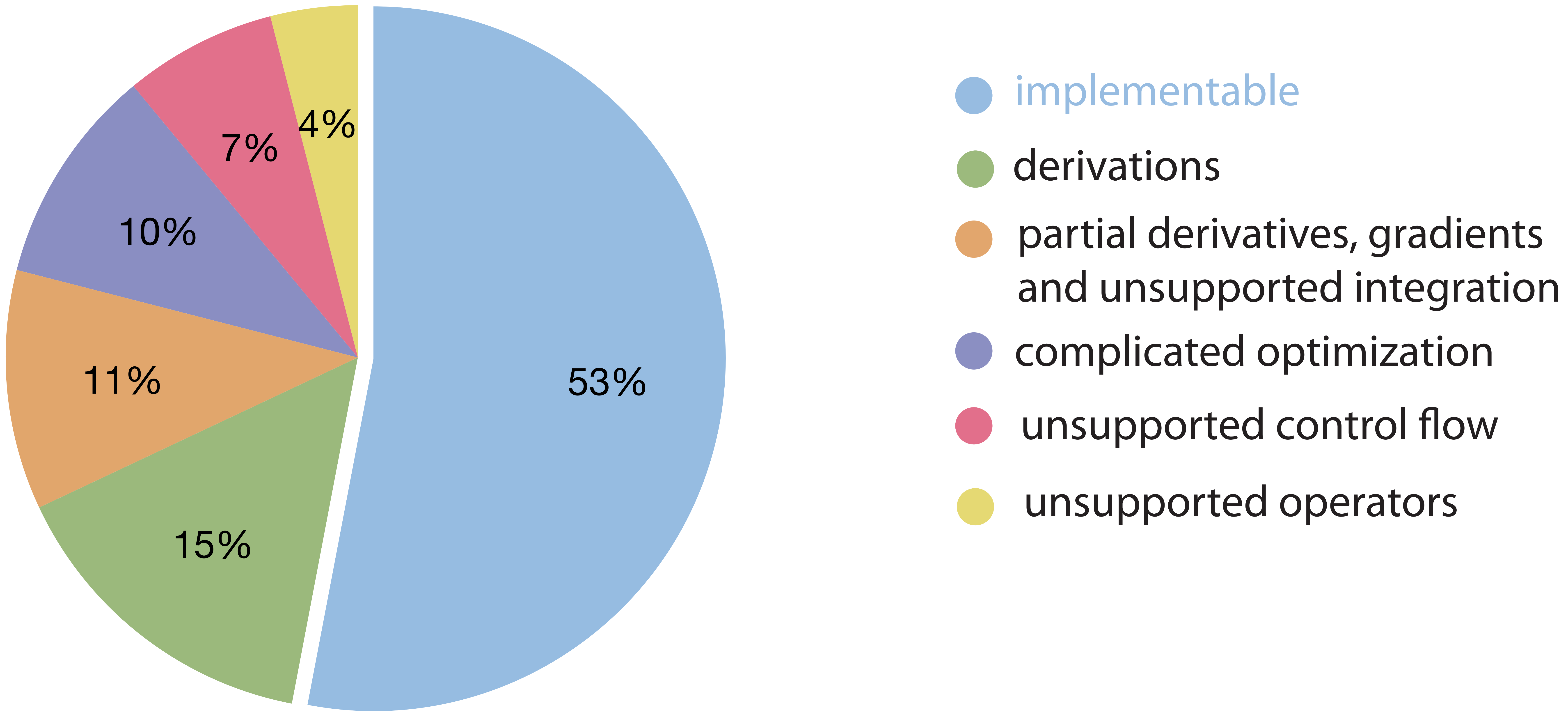
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



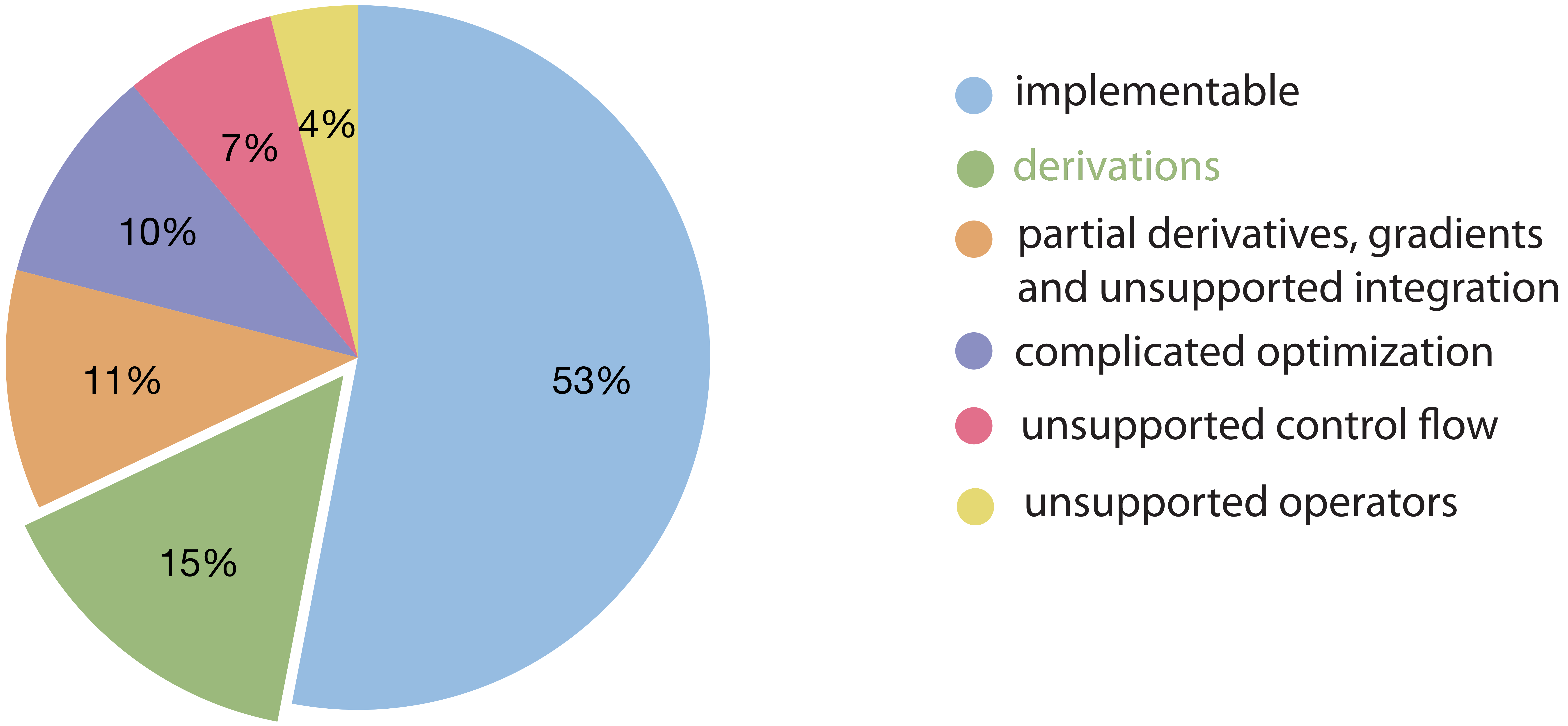
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



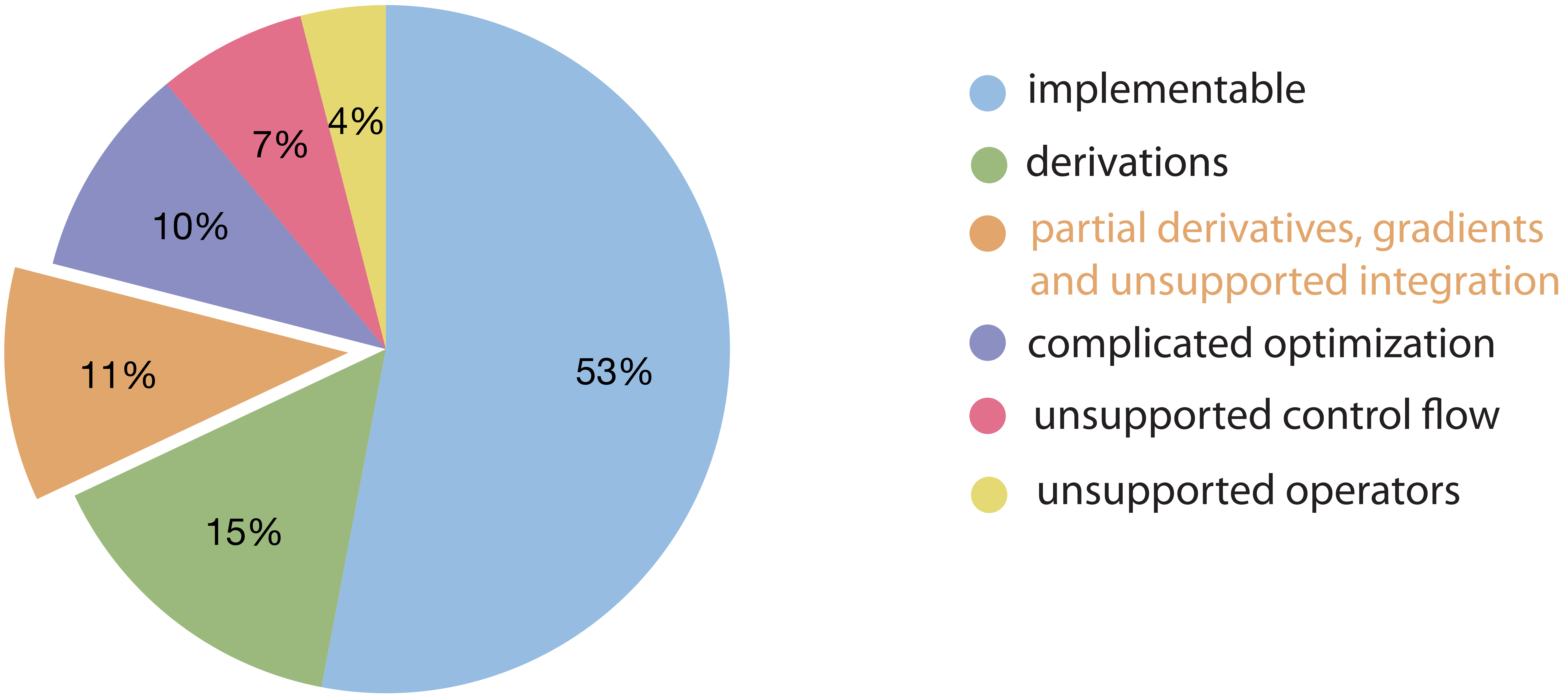
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



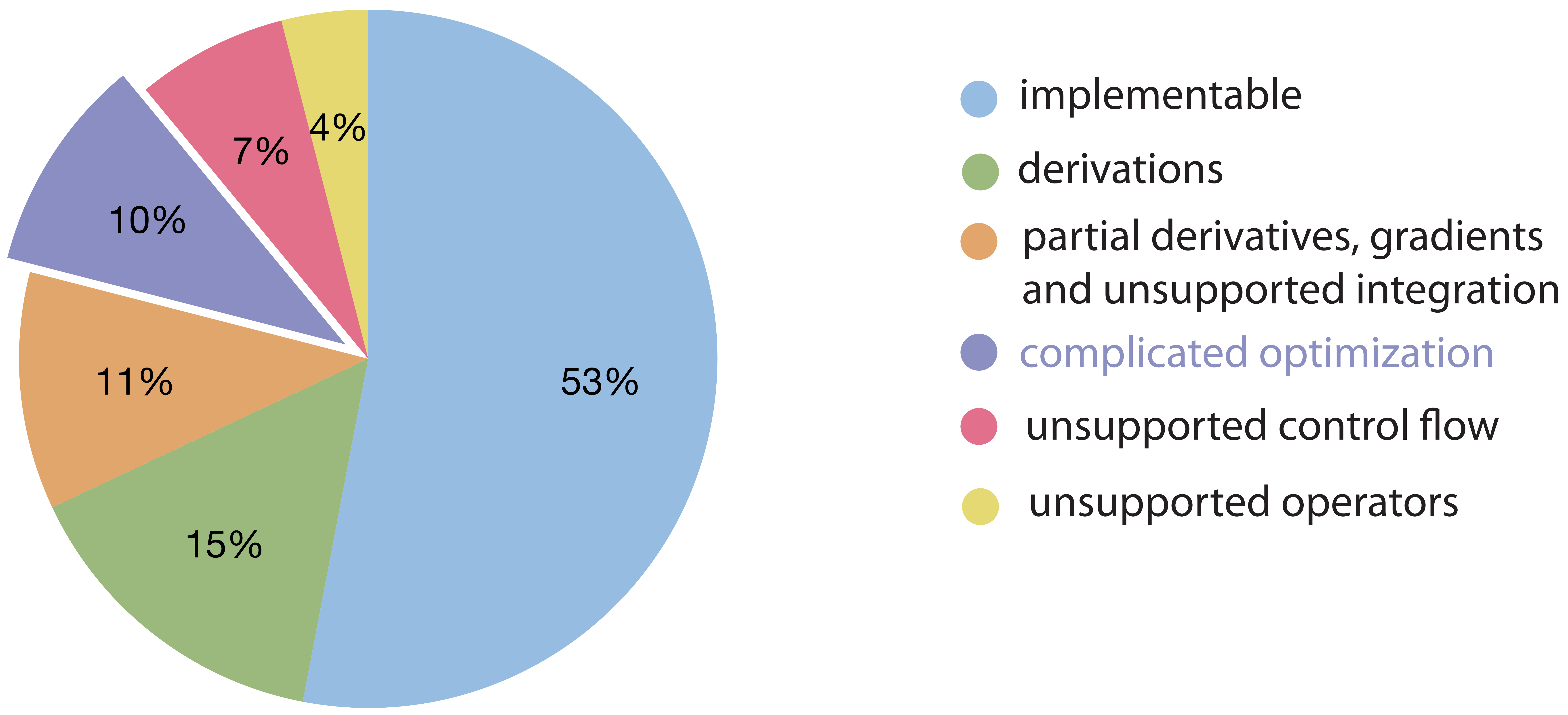
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



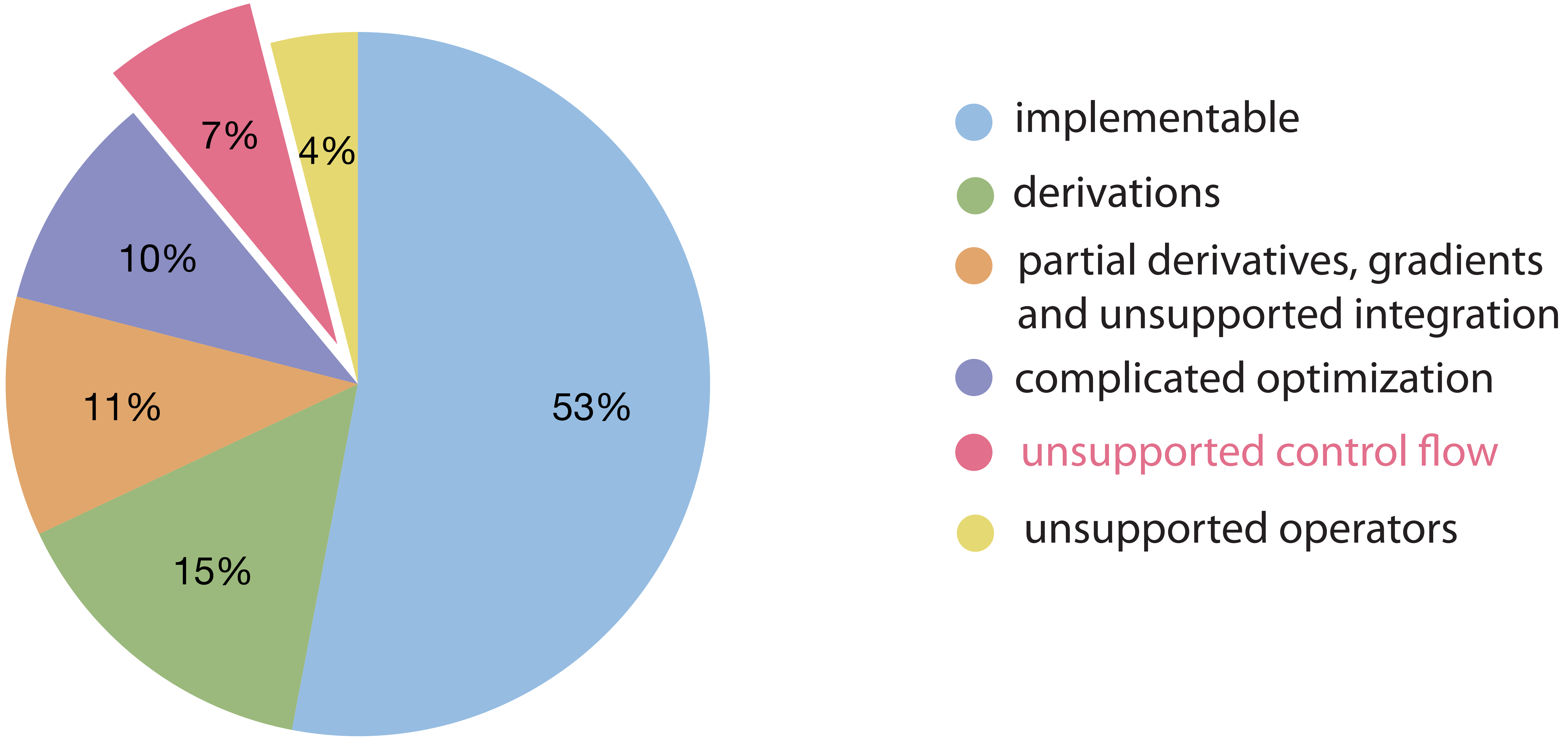
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



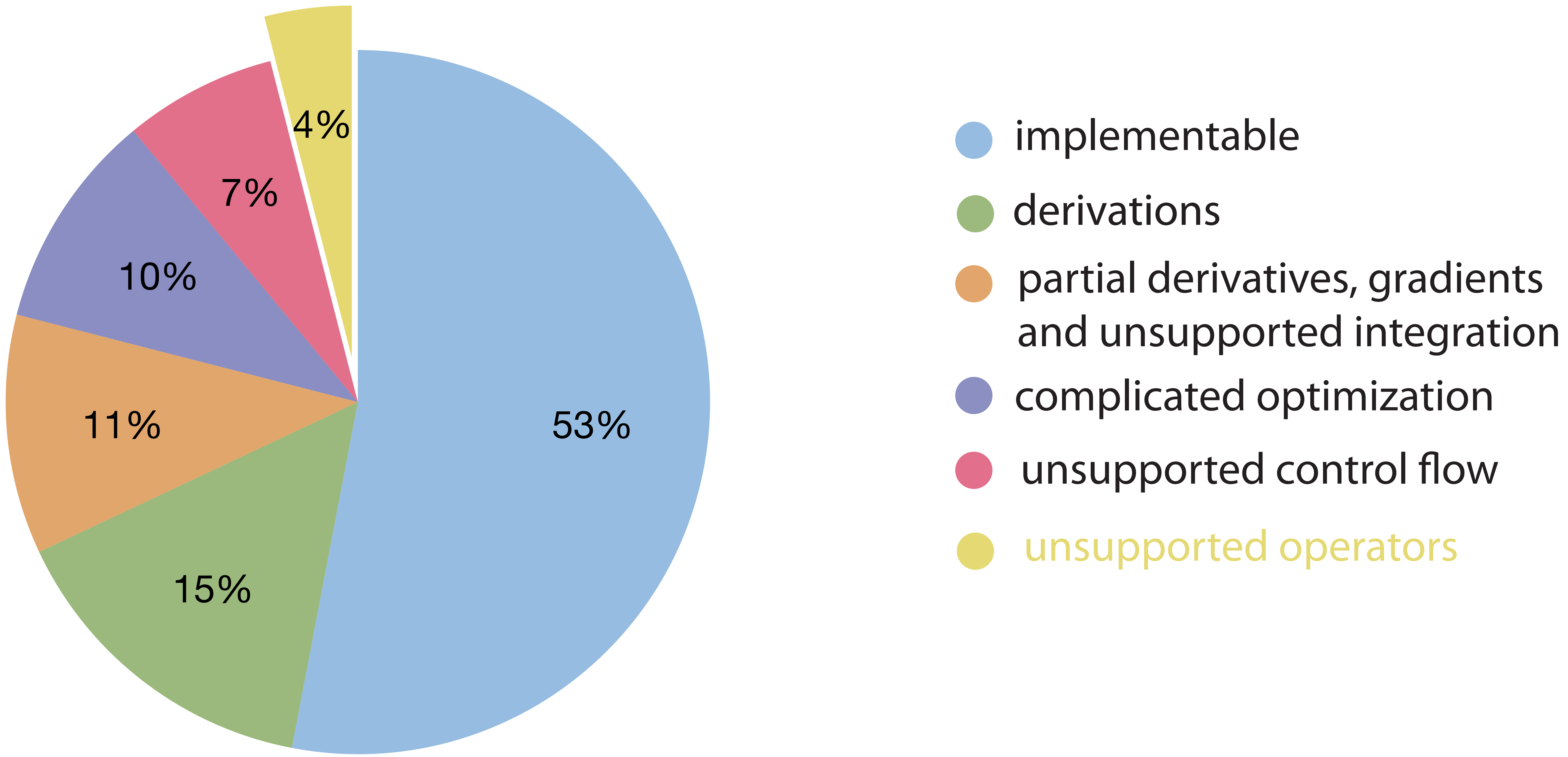
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



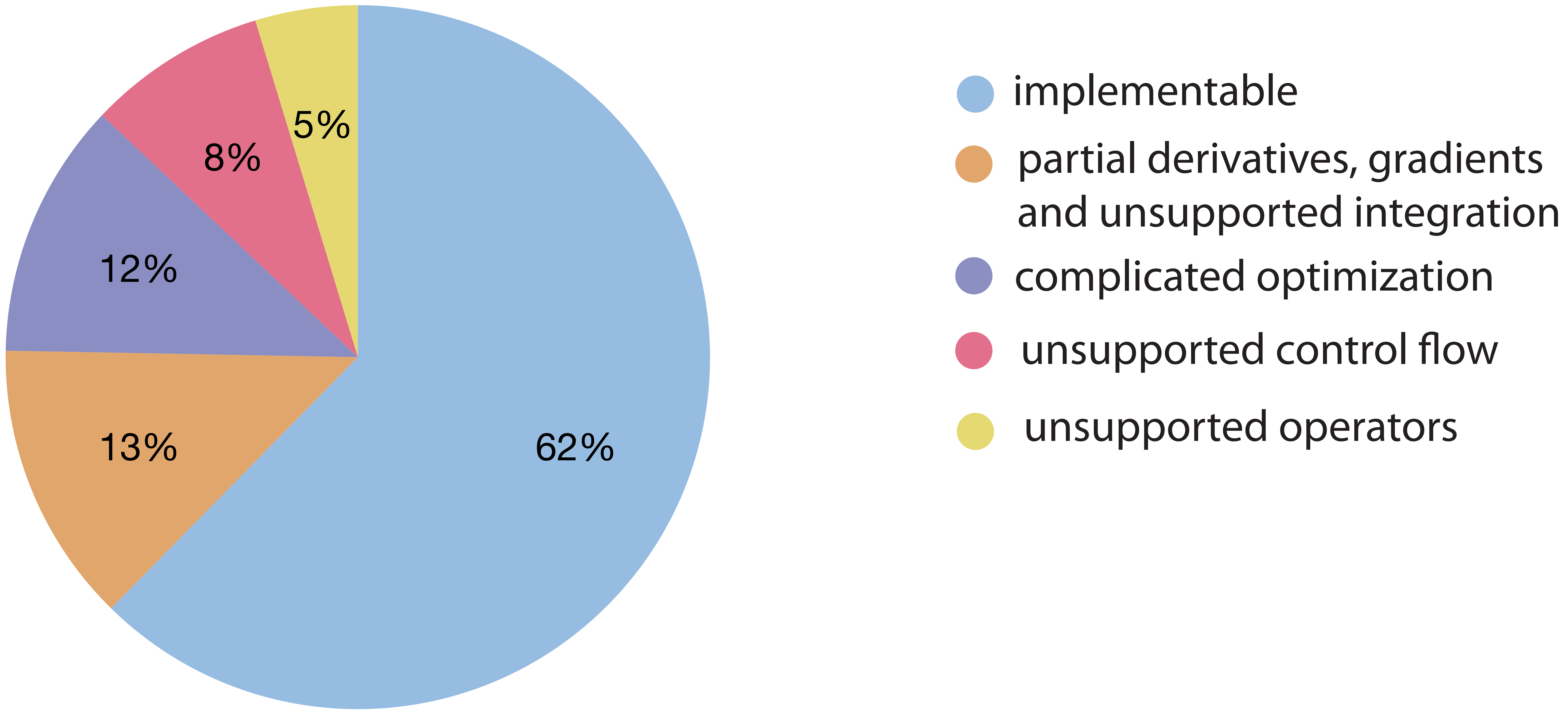
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



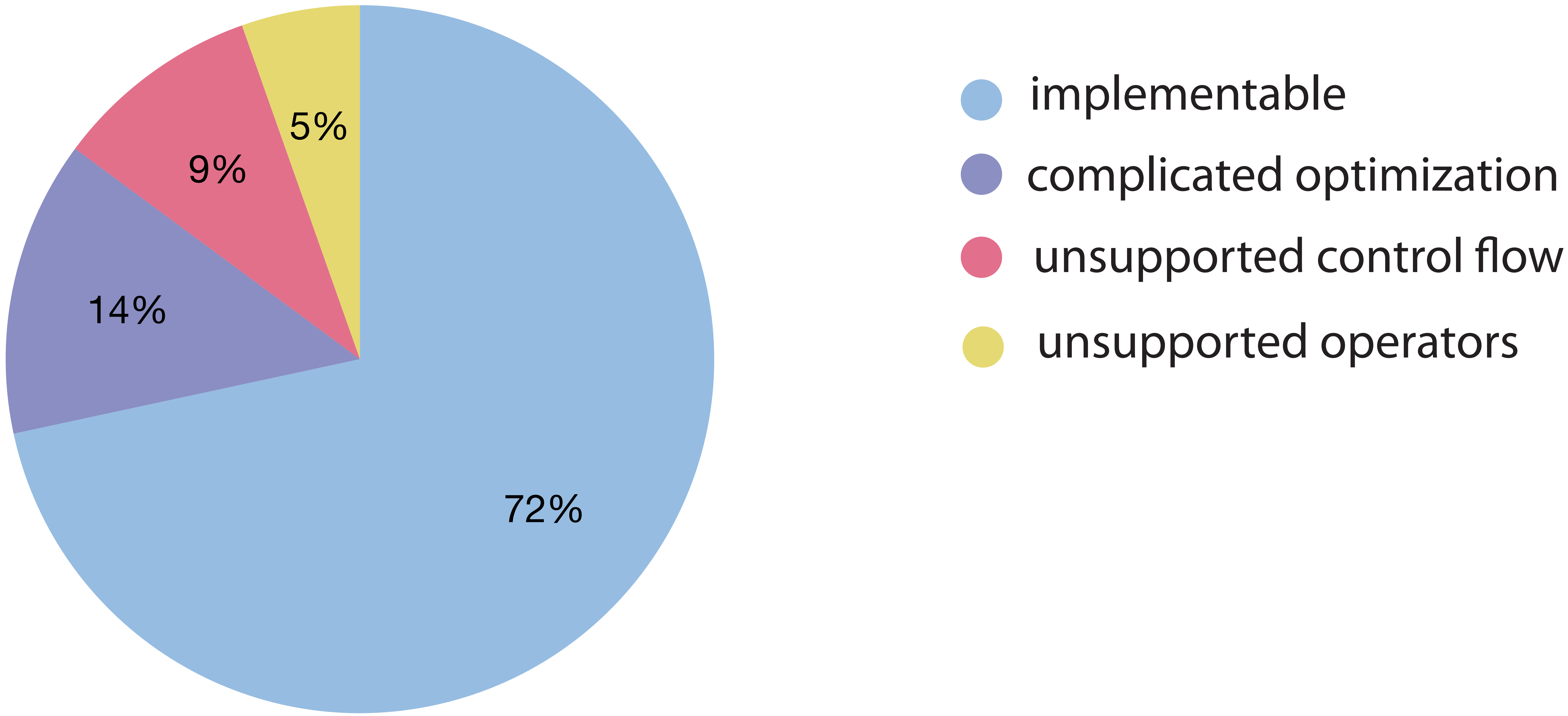
A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



A Statistical Estimate of Applicability

- We randomly sampled 100 of all 1987 numbered equations



User study

User study

- 8 CS PhD students

User study

- 8 CS PhD students
- Implemented 3 formula using both I❤️LA and their preferred programming environment (half C++/Eigen, half Python/NumPy)

User study

- 8 CS PhD students
- Implemented 3 formula using both I❤️LA and their preferred programming environment (half C++/Eigen, half Python/NumPy)

Simple

Given an $n \times n$ matrix A, an n-vector b, and a constant c, compute the quadratic form for an n-vector x:

$$x^T A x + b^T x + c$$

User study

- 8 CS PhD students
- Implemented 3 formula using both I❤️LA and their preferred programming environment (half C++/Eigen, half Python/NumPy)

Medium

Multiply a 3D vertex position v by a weighted average of 4×4 transformation matrices T_i. The corresponding weights are w_i. Assume the vertex position v is already in homogeneous coordinates, which is to say v is a 4-vector.

$$u = \sum_i w_i T_i v$$

User study

- 8 CS PhD students
- Implemented 3 formula using both I❤️LA and their preferred programming environment (half C++/Eigen, half Python/NumPy)

Complex

Create an edge-weighted adjacency matrix. Given a set of edges E for a graph of n vertices v_i, create the matrix:

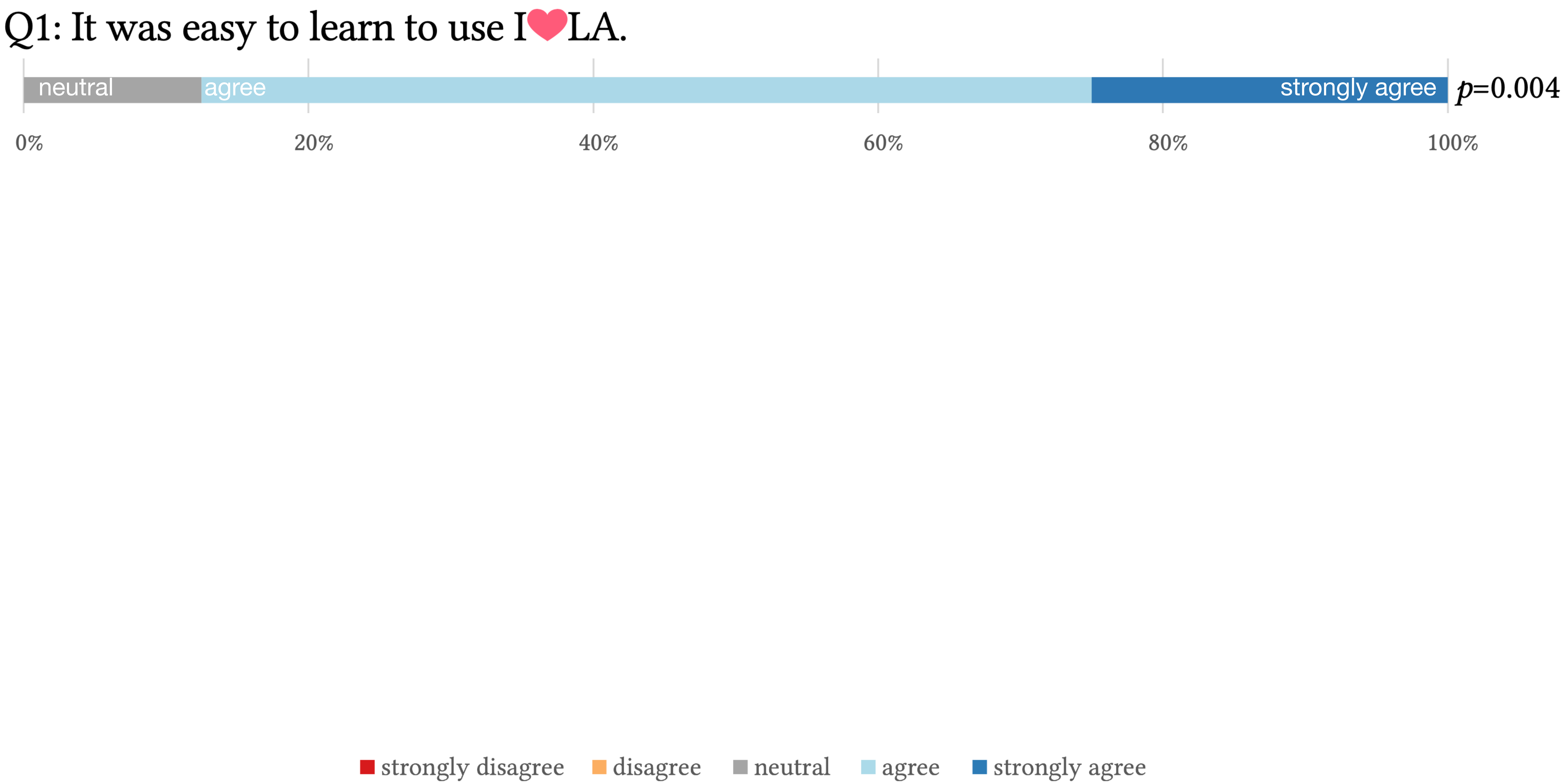
$$A_{ij} = \begin{cases} \frac{1}{\|v_i - v_j\|} & \text{if } i,j \in E \\ 0 & \text{otherwise} \end{cases}$$

Qualitative data from our user study

Qualitative data from our user study

strongly disagree disagree neutral agree strongly agree

Qualitative data from our user study



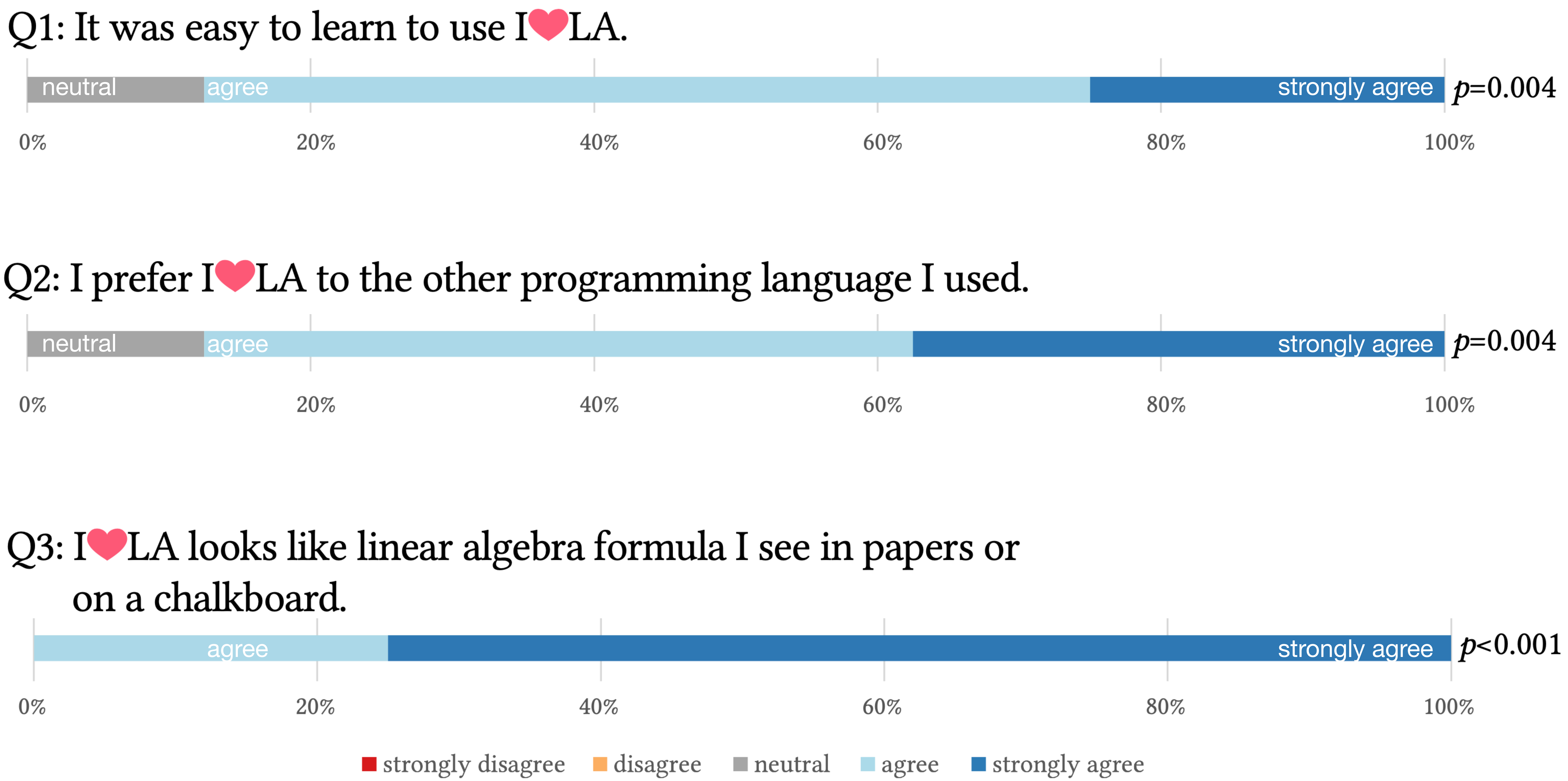
Qualitative data from our user study

Q1: It was easy to learn to use ILA.

Q2: I prefer ILA to the other programming language I used.

strongly disagree · disagree · neutral · agree · strongly agree

Qualitative data from our user study



User study observations and conclusions

User study observations and conclusions

- The average time for each task

	simple	medium	complex
I❤️LA (minutes)	10	9	12
Other (minutes)	4	6	12
Significance (p)	**0.005**	0.065	0.862

User study observations and conclusions

- The average time for each task

	simple	medium	complex
I❤️LA (minutes)	10	9	12
Other (minutes)	4	6	12
Significance (p)	**0.005**	0.065	0.862

- Users can accomplish a range of tasks in I❤️LA within 15 minutes

User study observations and conclusions

- The average time for each task

	simple	medium	complex
I❤️LA (minutes)	10	9	12
Other (minutes)	4	6	12
Significance (p)	**0.005**	0.065	0.862

- Users can accomplish a range of tasks in I❤️LA within 15 minutes

- Users perceive that I❤️LA looks similar to conventional math

Limitation: Unsupported equations

Limitation: Unsupported equations

- Unsupported operators

$$\Sigma_i^v = \operatorname{cov}\left(v_i \cup \mathcal{N}(i)\right) + \sigma_0^2 I, \tag{15}$$

Limitation: Unsupported equations

- Unsupported operators

$$\Sigma_i^v = \mathrm{cov}\left(v_i \cup \mathcal{N}(i)\right) + \sigma_0^2 I, \tag{15}$$

- Multiple conditions

$$w_{\mathrm{avr}}(q) = \frac{1}{\pi}\begin{cases} \frac{1}{40}(15q^3 - 36q^2 + 40) & 0 \le q < 1\,, \\ \frac{-3}{4q^3}\left(\frac{q^6}{6} - \frac{6q^5}{5} + 3q^4 - \frac{8q^3}{3} + \frac{1}{15}\right) & 1 \le q < 2\,, \\ \frac{3}{4q^3} & q \ge 2\,. \end{cases}$$

Limitation: Unsupported equations

- Unsupported operators

$$\Sigma_i^v = \operatorname{cov}\left(v_i \cup \mathcal{N}(i)\right) + \sigma_0^2 I, \qquad (15)$$

- Derivations

$$\sum_{j=1}^{n} P_{i,j} \leq f_0(x_i)\,,\ \forall i \qquad (7)$$

- Multiple conditions

$$w_{\mathrm{avr}}(q) = \frac{1}{\pi}\begin{cases} \frac{1}{40}(15q^3 - 36q^2 + 40) & 0 \leq q < 1\,, \\ \frac{-3}{4q^3}\left(\frac{q^6}{6} - \frac{6q^5}{5} + 3q^4 - \frac{8q^3}{3} + \frac{1}{15}\right) & 1 \leq q < 2\,, \\ \frac{3}{4q^3} & q \geq 2\,. \end{cases}$$

Limitation: Unsupported equations

- Unsupported operators

$$\Sigma_i^v = \operatorname{cov}\left(v_i \cup \mathcal{N}(i)\right) + \sigma_0^2 I, \qquad (15)$$

- Derivations

$$\sum_{j=1}^{n} P_{i,j} \leq f_0(x_i)\,,\ \forall i \qquad (7)$$

- Multiple conditions

$$w_{\mathrm{avr}}(q) = \frac{1}{\pi} \begin{cases} \frac{1}{40}(15q^3 - 36q^2 + 40) & 0 \leq q < 1\,, \\ \frac{-3}{4q^3}\left(\frac{q^6}{6} - \frac{6q^5}{5} + 3q^4 - \frac{8q^3}{3} + \frac{1}{15}\right) & 1 \leq q < 2\,, \\ \frac{3}{4q^3} & q \geq 2\,. \end{cases}$$

- Derivatives

$$\lambda_{1,2} = 2\frac{\partial \Psi_5}{\partial I_5}, \qquad (13)$$

Limitation: Unsupported equations

- Unsupported operators

$$\Sigma_i^v = \operatorname{cov}(v_i \underline{\cup} \mathcal{N}(i)) + \sigma_0^2 I, \qquad (15)$$

- Derivations

$$\sum_{j=1}^{n} P_{i,j} \leq f_0(x_i)\,, \forall i \qquad (7)$$

- Unsupported optimization

$$\mu(\mathbf{X}) = \underline{\max_{1\leq i\neq j\leq n}} \frac{|\mathbf{X}_{\cdot,i}^T \mathbf{X}_{\cdot,j}|}{\|\mathbf{X}_{\cdot,i}\|_2 \|\mathbf{X}_{\cdot,j}\|_2}. \qquad (20)$$

- Multiple conditions

$$w_{\mathrm{avr}}(q) = \frac{1}{\pi} \begin{cases} \frac{1}{40}(15q^3 - 36q^2 + 40) & 0 \leq q < 1\,, \\ \frac{-3}{4q^3}\left(\frac{q^6}{6} - \frac{6q^5}{5} + 3q^4 - \frac{8q^3}{3} + \frac{1}{15}\right) & 1 \leq q < 2\,, \\ \frac{3}{4q^3} & q \geq 2\,. \end{cases}$$

- Derivatives

$$\lambda_{1,2} = 2\frac{\partial \Psi_5}{\partial I_5}, \qquad (13)$$

Limitation: Unsupported equations

- Unsupported operators

$$\Sigma_i^v = \mathrm{cov}\left(v_i \cup \mathcal{N}(i)\right) + \sigma_0^2 I, \qquad (15)$$

- Derivations

$$\sum_{j=1}^{n} P_{i,j} \leq f_0(x_i), \forall i \qquad (7)$$

- Unsupported optimization

$$\mu(\mathbf{X}) = \max_{1 \leq i \neq j \leq n} \frac{|\mathbf{X}_{\cdot,i}^T \mathbf{X}_{\cdot,j}|}{\|\mathbf{X}_{\cdot,i}\|_2 \|\mathbf{X}_{\cdot,j}\|_2}. \qquad (20)$$

- Multiple conditions

$$w_{\mathrm{avr}}(q) = \frac{1}{\pi} \begin{cases} \frac{1}{40}(15q^3 - 36q^2 + 40) & 0 \leq q < 1, \\ \frac{-3}{4q^3}\left(\frac{q^6}{6} - \frac{6q^5}{5} + 3q^4 - \frac{8q^3}{3} + \frac{1}{15}\right) & 1 \leq q < 2, \\ \frac{3}{4q^3} & q \geq 2. \end{cases}$$

- Derivatives

$$\lambda_{1,2} = 2\frac{\partial \Psi_5}{\partial I_5}, \qquad (13)$$

- Others

$$\Psi(x,y,z,t) = \iiint \Phi(k_x, k_y, k_z) e^{2\pi i (k_x x + k_y y + k_z z - f t)}\, \mathrm{d}k_x \mathrm{d}k_y \mathrm{d}k_z\,. \quad (3)$$

Limitation: Unsupported equations

- Unsupported operators

$$\Sigma_i^v = \mathrm{cov}\left(v_i \cup \mathcal{N}(i)\right) + \sigma_0^2 I, \qquad (15)$$

- Derivations

$$\sum_{j=1}^{n} P_{i,j} \leq f_0(x_i)\,,\ \forall i \qquad (7)$$

- Unsupported optimization

$$\mu(\mathbf{X}) = \max_{1\leq i\neq j\leq n} \frac{|\mathbf{X}_{\cdot,i}^T \mathbf{X}_{\cdot,j}|}{\|\mathbf{X}_{\cdot,i}\|_2 \|\mathbf{X}_{\cdot,j}\|_2}. \qquad (20)$$

- Multiple conditions

$$w_{\mathrm{avr}}(q) = \frac{1}{\pi} \begin{cases} \frac{1}{40}(15q^3 - 36q^2 + 40) & 0 \leq q < 1\,, \\ \frac{-3}{4q^3}\left(\frac{q^6}{6} - \frac{6q^5}{5} + 3q^4 - \frac{8q^3}{3} + \frac{1}{15}\right) & 1 \leq q < 2\,, \\ \frac{3}{4q^3} & q \geq 2\,. \end{cases}$$

- Derivatives

$$\lambda_{1,2} = 2\frac{\partial \Psi_5}{\partial I_5}, \qquad (13)$$

- Others

$$\Psi(x,y,z,t) = \iiint \Phi(k_x, k_y, k_z) e^{2\pi i(k_x x + k_y y + k_z z - f t)}\, \mathrm{d}k_x \mathrm{d}k_y \mathrm{d}k_z\,. \qquad (3)$$

?

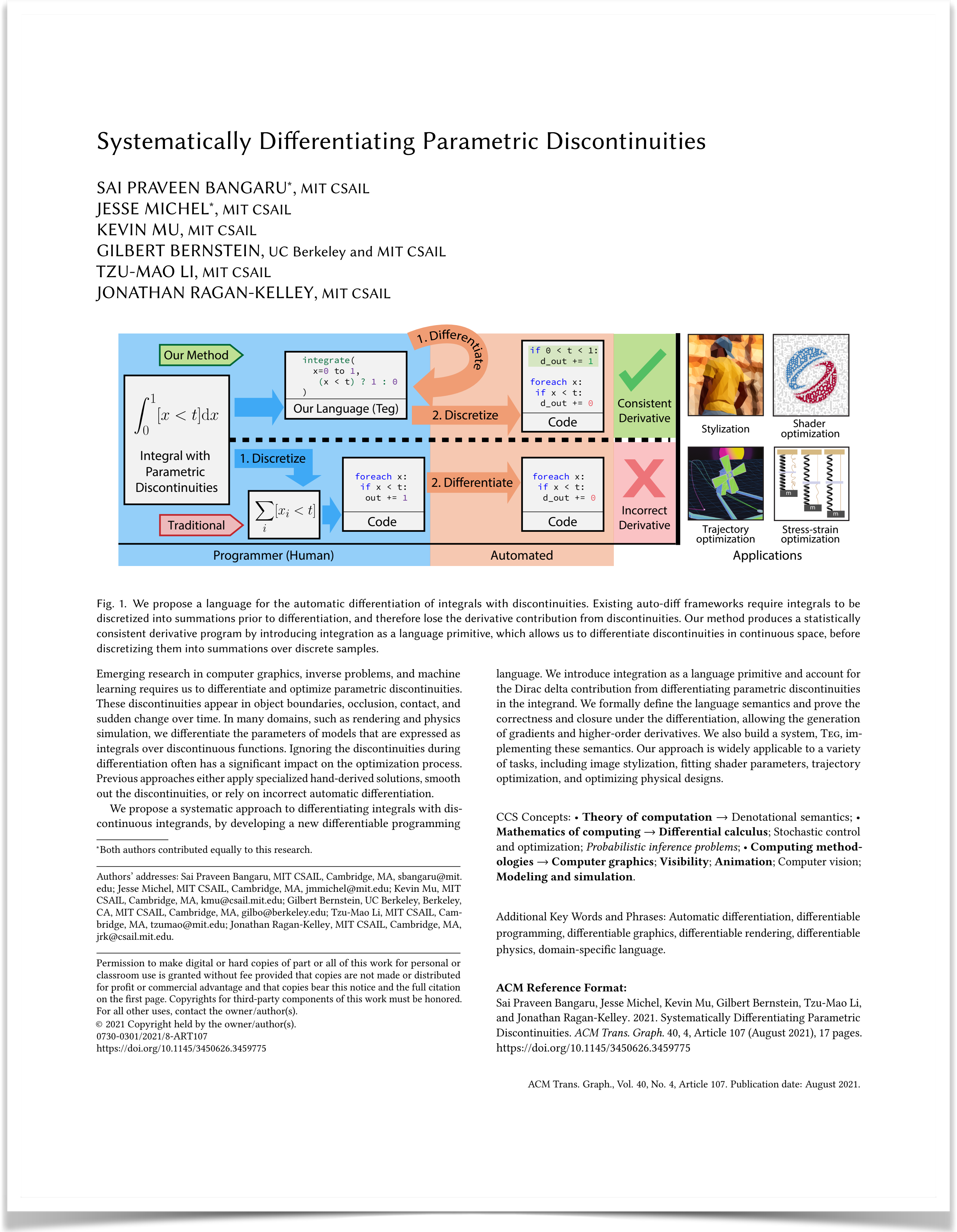
Future work

Future work

- Support more language features (tensors, automatic differentiation, integration, optimization)

Future work

- Support more language features (tensors, automatic differentiation, integration, optimization)



Systematically Differentiating Parametric Discontinuities

SAI PRAVEEN BANGARU*, MIT CSAIL
JESSE MICHEL*, MIT CSAIL
KEVIN MU, MIT CSAIL
GILBERT BERNSTEIN, UC Berkeley and MIT CSAIL
TZU-MAO LI, MIT CSAIL
JONATHAN RAGAN-KELLEY, MIT CSAIL

Fig. 1. We propose a language for the automatic differentiation of integrals with discontinuities. Existing auto-diff frameworks require integrals to be discretized into summations prior to differentiation, and therefore lose the derivative contribution from discontinuities. Our method produces a statistically consistent derivative program by introducing integration as a language primitive, which allows us to differentiate discontinuities in continuous space, before discretizing them into summations over discrete samples.

Emerging research in computer graphics, inverse problems, and machine learning requires us to differentiate and optimize parametric discontinuities. These discontinuities appear in object boundaries, occlusion, contact, and sudden change over time. In many domains, such as rendering and physics simulation, we differentiate the parameters of models that are expressed as integrals over discontinuous functions. Ignoring the discontinuities during differentiation often has a significant impact on the optimization process. Previous approaches either apply specialized hand-derived solutions, smooth out the discontinuities, or rely on incorrect automatic differentiation.

We propose a systematic approach to differentiating integrals with discontinuous integrands, by developing a new differentiable programming language. We introduce integration as a language primitive and account for the Dirac delta contribution from differentiating parametric discontinuities in the integrand. We formally define the language semantics and prove the correctness and closure under the differentiation, allowing the generation of gradients and higher-order derivatives. We also build a system, TEG, implementing these semantics. Our approach is widely applicable to a variety of tasks, including image stylization, fitting shader parameters, trajectory optimization, and optimizing physical designs.

CCS Concepts: • **Theory of computation** → Denotational semantics; • **Mathematics of computing** → **Differential calculus**; Stochastic control and optimization; *Probabilistic inference problems*; • **Computing methodologies** → **Computer graphics**; **Visibility**; **Animation**; Computer vision; **Modeling and simulation**.

Additional Key Words and Phrases: Automatic differentiation, differentiable programming, differentiable graphics, differentiable rendering, differentiable physics, domain-specific language.

ACM Reference Format:
Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically Differentiating Parametric Discontinuities. *ACM Trans. Graph.* 40, 4, Article 107 (August 2021), 17 pages. https://doi.org/10.1145/3450626.3459775

*Both authors contributed equally to this research.

Authors' addresses: Sai Praveen Bangaru, MIT CSAIL, Cambridge, MA, sbangaru@mit.edu; Jesse Michel, MIT CSAIL, Cambridge, MA, jmmichel@mit.edu; Kevin Mu, MIT CSAIL, Cambridge, MA, kmu@csail.mit.edu; Gilbert Bernstein, UC Berkeley, Berkeley, CA, MIT CSAIL, Cambridge, MA, gilbo@berkeley.edu; Tzu-Mao Li, MIT CSAIL, Cambridge, MA, tzumao@mit.edu; Jonathan Ragan-Kelley, MIT CSAIL, Cambridge, MA, jrk@csail.mit.edu.

0730-0301/2021/8-ART107
https://doi.org/10.1145/3450626.3459775

TEG [Bangaru et al. 2021]

Minimization (10%)

Minimization (10%)

$$\Phi(\mathbf{d}) = \min_{\theta} \frac{D_{in}(\theta, \mathbf{d})}{D_{out}(\theta, \mathbf{d})}. \tag{10}$$

$$\mathcal{P}_{opt} = \arg\min_{\mathcal{P}} \Delta\tau_{\mathcal{S}}(\mathcal{P}). \tag{17}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\mathcal{M}}{\operatorname{argmin}} \sum_{i} \xi_i d(\mathbf{x}, \mathbf{x}_i)^2. \tag{3}$$

$$\underset{\boldsymbol{\rho}}{\text{minimize}} \left\|\boldsymbol{\tau} - \mathbf{A}\boldsymbol{\rho}\right\|_2^2 + \Gamma\left(\boldsymbol{\rho}\right), \quad \text{s.t. } 0 \leq \boldsymbol{\rho} \tag{4}$$

$$\max_{\substack{a,\, \phi_1, \ldots, \phi_N \\ \text{s.t. } a^T a = 1}} \sum_{i=1}^{N} \operatorname{dot}\Big(p_i,\ \cos(\frac{\phi_i}{2}) + \sin(\frac{\phi_i}{2})a\Big). \tag{1}$$

$$(\hat{\mathbf{I}}, \hat{\mathbf{V}}) = \arg\min_{\mathbf{I},\mathbf{V}} \|\mathbf{J} - \mathbf{\Phi}\mathbf{I}\|_2^2 + R\left(\mathbf{V}\right) \quad \text{s.t.} \quad \mathbf{V} = \mathbf{I}. \tag{17}$$

$$\min_{C} \sum_{i=1}^{N} ||x_i + (R_i - I)C||^2. \tag{3}$$

$$\begin{gathered} \min f(\mathbf{v}) = \frac{1}{2}\mathbf{v}^T \mathbf{A}\mathbf{v} - \mathbf{v}^T \mathbf{b} \\ \text{subject to } \mathbf{J}\mathbf{v} \geq \mathbf{c}_n. \end{gathered} \tag{10}$$

Minimization (10%)

$$\Phi(\mathbf{d}) = \min_{\theta} \frac{D_{in}(\theta, \mathbf{d})}{D_{out}(\theta, \mathbf{d})}. \quad (10)$$

$$\mathcal{P}_{opt} = \arg\min_{\mathcal{P}} \Delta\tau_S(\mathcal{P}). \quad (17)$$

I❤️LA:

```
argmin_(x ∈ ℝ³) 1/2 xᵀQx + qᵀx
s.t.
‖x‖ > 1
where
Q ∈ ℝ^(3×3)
q ∈ ℝ³
```

$$\min_{C} \sum_{i=1}^{N} ||x_i + (R_i - I)C||^2. \quad (3)$$

$$\min f(\mathbf{v}) = \frac{1}{2}\mathbf{v}^T \mathbf{A}\mathbf{v} - \mathbf{v}^T \mathbf{b} \quad (10)$$
$$\text{subject to } \mathbf{J}\mathbf{v} \geq \mathbf{c}_n.$$

Integration (8%)

Integration (8%)

$$T_{\mathrm{ir}}(\boldsymbol{x}) = \frac{1}{C} \int_{\mathcal{H}^2} \int_{\mathcal{H}^2} R_{\mathrm{ir}}(\boldsymbol{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)\, \mathrm{d}\boldsymbol{\omega}_i\, \mathrm{d}\boldsymbol{\omega}_o, \qquad (22)$$

$$\widetilde{G} := -\int_{\mathbb{S}^1} \langle \gamma, g \rangle dm \qquad (11)$$

$$\gamma := \int_{-\pi}^{\pi} d(\varphi)\mathbf{c}(\varphi)\, \mathrm{d}\varphi \in \mathbb{C}^{m+1}, \qquad (1)$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}) = \int_{\partial\Omega} \int_{S^2} L_i(\mathbf{x}', \boldsymbol{\omega}')\, S(\mathbf{x}', \boldsymbol{\omega}', \mathbf{x}, \boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}'\, \mathrm{d}\mathbf{x}'. \qquad (3)$$

$$p(\boldsymbol{x}, k) = -\int \frac{i}{4} H_0^{(2)}(k||\boldsymbol{x} - \boldsymbol{y}||) g(\boldsymbol{y}, k)\, d\boldsymbol{y}. \qquad (14)$$

$$U(\mathbf{q}) = \int_V \Psi(\mathbf{s}(\mathbf{q}))dV, \qquad (81)$$

$$u_2(x, y) = \frac{e^{ikz}}{i\lambda z} \iint u_1(x', y')\, e^{\frac{ik}{2z}\left\{(x-x')^2 + (y-y')^2\right\}} dx'dy', \qquad (4)$$

$$\begin{aligned} \boldsymbol{f}_{s\to t} &= \int \boldsymbol{F}_{\mathrm{Kelvin}}\, \mathrm{d}\boldsymbol{r} = \mu_0 \int \boldsymbol{M}_t(\boldsymbol{r}) \cdot \nabla \boldsymbol{H}_s(\boldsymbol{r})\, \mathrm{d}\boldsymbol{r} \\ &= \mu_0 \boldsymbol{m}_t \cdot \int W(\boldsymbol{r} - \boldsymbol{r}_t, h) \nabla \boldsymbol{H}(\boldsymbol{r} - \boldsymbol{r}_s, \boldsymbol{m}_s)\, \mathrm{d}\boldsymbol{r}\,, \end{aligned} \qquad (17)$$

Integration (8%)

$$T_{\mathrm{ir}}(\boldsymbol{x}) = \frac{1}{C}\int_{\mathcal{H}^2}\int_{\mathcal{H}^2} R_{\mathrm{ir}}(\boldsymbol{x},\omega_i,\omega_o)\,\mathrm{d}\omega_i\,\mathrm{d}\omega_o, \tag{22}$$

$$\widetilde{G} := -\int_{\mathbb{S}^1}\langle\gamma, g\rangle dm \tag{11}$$

$$\gamma := \int_{-\pi}^{\pi} d(\varphi)\mathbf{c}(\varphi)\,\mathrm{d}\varphi \in \mathbb{C}^{m+1},$$

$$\ldots S(\mathrm{x}',\omega',\mathrm{x},\omega)\,\mathrm{d}\omega'\,\mathrm{dx}'. \tag{3}$$

I❤️LA:

```
∫_0^3 ∫_[1, 2] xy ∂x ∂y
```

$$p(\boldsymbol{x},k) = -\int \frac{i}{4}H_0^{(2)}(k\|\boldsymbol{x}-\boldsymbol{y}\|)g(\boldsymbol{y},k)\,d\boldsymbol{y}. \tag{14}$$

$$U(\mathrm{q}) = \int_V \Psi(\mathrm{s}(\mathrm{q}))dV, \tag{81}$$

$$u_2(x,y) = \frac{e^{ikz}}{i\lambda z}\iint u_1(x',y')\,e^{\frac{ik}{2z}\{(x-x')^2+(y-y')^2\}}dx'dy', \tag{4}$$

$$\begin{aligned} f_{s\to t} &= \int F_{\mathrm{Kelvin}}\,\mathrm{d}\boldsymbol{r} = \mu_0\int M_t(\boldsymbol{r})\cdot\nabla H_s(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r} \\ &= \mu_0\boldsymbol{m}_t\cdot\int W(\boldsymbol{r}-\boldsymbol{r}_t,h)\nabla H(\boldsymbol{r}-\boldsymbol{r}_s,\boldsymbol{m}_s)\,\mathrm{d}\boldsymbol{r}\,, \end{aligned} \tag{17}$$

Future work

Future work

- More output languages (PyTorch, TensorFlow, Julia, JavaScript, FORTRAN)

Future work

- More output languages (PyTorch, TensorFlow, Julia, JavaScript, FORTRAN)
- Analyze and support more fields (ML, robotics, CV, physics, scientific computing in general)

Future work

- More output languages (PyTorch, TensorFlow, Julia, JavaScript, FORTRAN)
- Analyze and support more fields (ML, robotics, CV, physics, scientific computing in general)
- Papers of the future

Future work

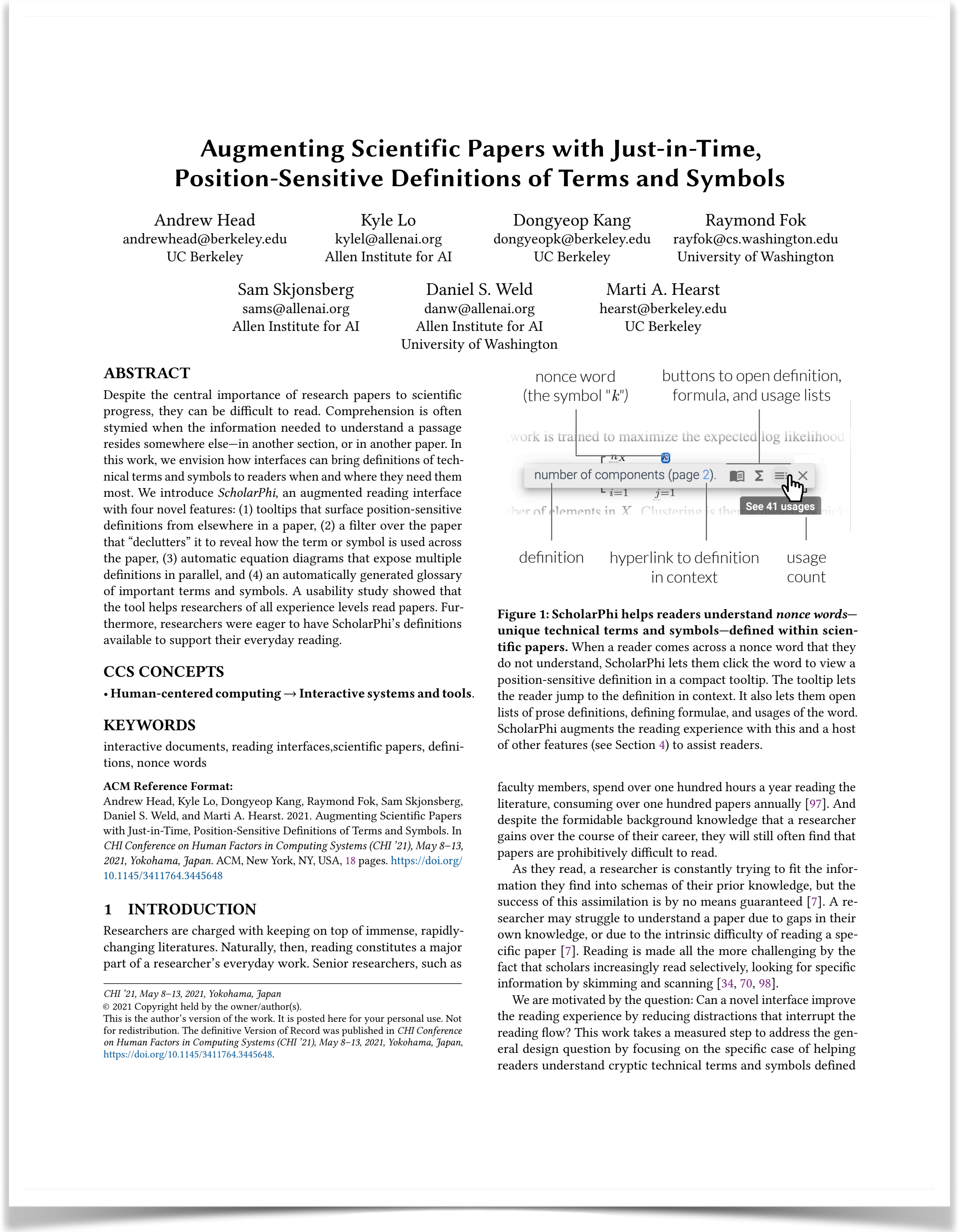
- More output languages (PyTorch, TensorFlow, Julia, JavaScript, FORTRAN)
- Analyze and support more fields (ML, robotics, CV, physics, scientific computing in general)
- Papers of the future
  - compile an entire paper into a library

Future work

- More output languages (PyTorch, TensorFlow, Julia, JavaScript, FORTRAN)
- Analyze and support more fields (ML, robotics, CV, physics, scientific computing in general)
- Papers of the future
  - compile an entire paper into a library
  - improve readability

Future work

- More output languages (PyTorch, TensorFlow, Julia, JavaScript, FORTRAN)
- Analyze and support more fields (ML, robotics, CV, physics, scientific computing in general)
- Papers of the future
    - compile an entire paper into a library
    - improve readability



Augmenting Scientific Papers with Just-in-Time, Position-Sensitive Definitions of Terms and Symbols

Andrew Head, andrewhead@berkeley.edu, UC Berkeley
Kyle Lo, kylel@allenai.org, Allen Institute for AI
Dongyeop Kang, dongyeopk@berkeley.edu, UC Berkeley
Raymond Fok, rayfok@cs.washington.edu, University of Washington
Sam Skjonsberg, sams@allenai.org, Allen Institute for AI
Daniel S. Weld, danw@allenai.org, Allen Institute for AI, University of Washington
Marti A. Hearst, hearst@berkeley.edu, UC Berkeley

ABSTRACT

Despite the central importance of research papers to scientific progress, they can be difficult to read. Comprehension is often stymied when the information needed to understand a passage resides somewhere else—in another section, or in another paper. In this work, we envision how interfaces can bring definitions of technical terms and symbols to readers when and where they need them most. We introduce *ScholarPhi*, an augmented reading interface with four novel features: (1) tooltips that surface position-sensitive definitions from elsewhere in a paper, (2) a filter over the paper that "declutters" it to reveal how the term or symbol is used across the paper, (3) automatic equation diagrams that expose multiple definitions in parallel, and (4) an automatically generated glossary of important terms and symbols. A usability study showed that the tool helps researchers of all experience levels read papers. Furthermore, researchers were eager to have ScholarPhi's definitions available to support their everyday reading.

CCS CONCEPTS

• **Human-centered computing → Interactive systems and tools**.

KEYWORDS

interactive documents, reading interfaces,scientific papers, definitions, nonce words

ACM Reference Format:
Andrew Head, Kyle Lo, Dongyeop Kang, Raymond Fok, Sam Skjonsberg, Daniel S. Weld, and Marti A. Hearst. 2021. Augmenting Scientific Papers with Just-in-Time, Position-Sensitive Definitions of Terms and Symbols. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3411764.3445648

1 INTRODUCTION

Researchers are charged with keeping on top of immense, rapidly-changing literatures. Naturally, then, reading constitutes a major part of a researcher's everyday work. Senior researchers, such as

CHI '21, May 8–13, 2021, Yokohama, Japan
© 2021 Copyright held by the owner/author(s).
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan*, https://doi.org/10.1145/3411764.3445648.

nonce word (the symbol "k")
buttons to open definition, formula, and usage lists
number of components (page 2).
See 41 usages
definition
hyperlink to definition in context
usage count

Figure 1: ScholarPhi helps readers understand *nonce words*—unique technical terms and symbols—defined within scientific papers. When a reader comes across a nonce word that they do not understand, ScholarPhi lets them click the word to view a position-sensitive definition in a compact tooltip. The tooltip lets the reader jump to the definition in context. It also lets them open lists of prose definitions, defining formulae, and usages of the word. ScholarPhi augments the reading experience with this and a host of other features (see Section 4) to assist readers.

faculty members, spend over one hundred hours a year reading the literature, consuming over one hundred papers annually [97]. And despite the formidable background knowledge that a researcher gains over the course of their career, they will still often find that papers are prohibitively difficult to read.

As they read, a researcher is constantly trying to fit the information they find into schemas of their prior knowledge, but the success of this assimilation is by no means guaranteed [7]. A researcher may struggle to understand a paper due to gaps in their own knowledge, or due to the intrinsic difficulty of reading a specific paper [7]. Reading is made all the more challenging by the fact that scholars increasingly read selectively, looking for specific information by skimming and scanning [34, 70, 98].

We are motivated by the question: Can a novel interface improve the reading experience by reducing distractions that interrupt the reading flow? This work takes a measured step to address the general design question by focusing on the specific case of helping readers understand cryptic technical terms and symbols defined

ScholarPhi [Head et al. 2021]

Conclusions

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly
- I❤️LA may reduce *translation loss* as ideas move from researchers to writers to readers to implementors

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly
- I❤️LA may reduce *translation loss* as ideas move from researchers to writers to readers to implementors

🗣️

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly
- I❤️LA may reduce *translation loss* as ideas move from researchers to writers to readers to implementors

🗣️ 👂

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly
- I❤️LA may reduce *translation loss* as ideas move from researchers to writers to readers to implementors

🗣️👂📝

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly
- I❤️LA may reduce *translation loss* as ideas move from researchers to writers to readers to implementors

🗣️ 👂 📝 📄

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly
- I❤️LA may reduce *translation loss* as ideas move from researchers to writers to readers to implementors

🗣️ 👂 📝 📄 👨‍💻

Conclusions

- I❤️LA has the potential to greatly benefit the scientific ecosystem
- I❤️LA makes it easy to try new ideas
- I❤️LA can be learned quickly
- I❤️LA may reduce *translation loss* as ideas move from researchers to writers to readers to implementors

🗣️ 👂 📝 📄 👨‍💻 = 🐞

Acknowledgments

Acknowledgments

• Anonymous reviewers for their suggestions

Acknowledgments

- Anonymous reviewers for their suggestions
- Towaki Takikawa for helpful feedback

Acknowledgments

- Anonymous reviewers for their suggestions
- Towaki Takikawa for helpful feedback
- Thomas LaToza for a discussion on evaluating programming languages

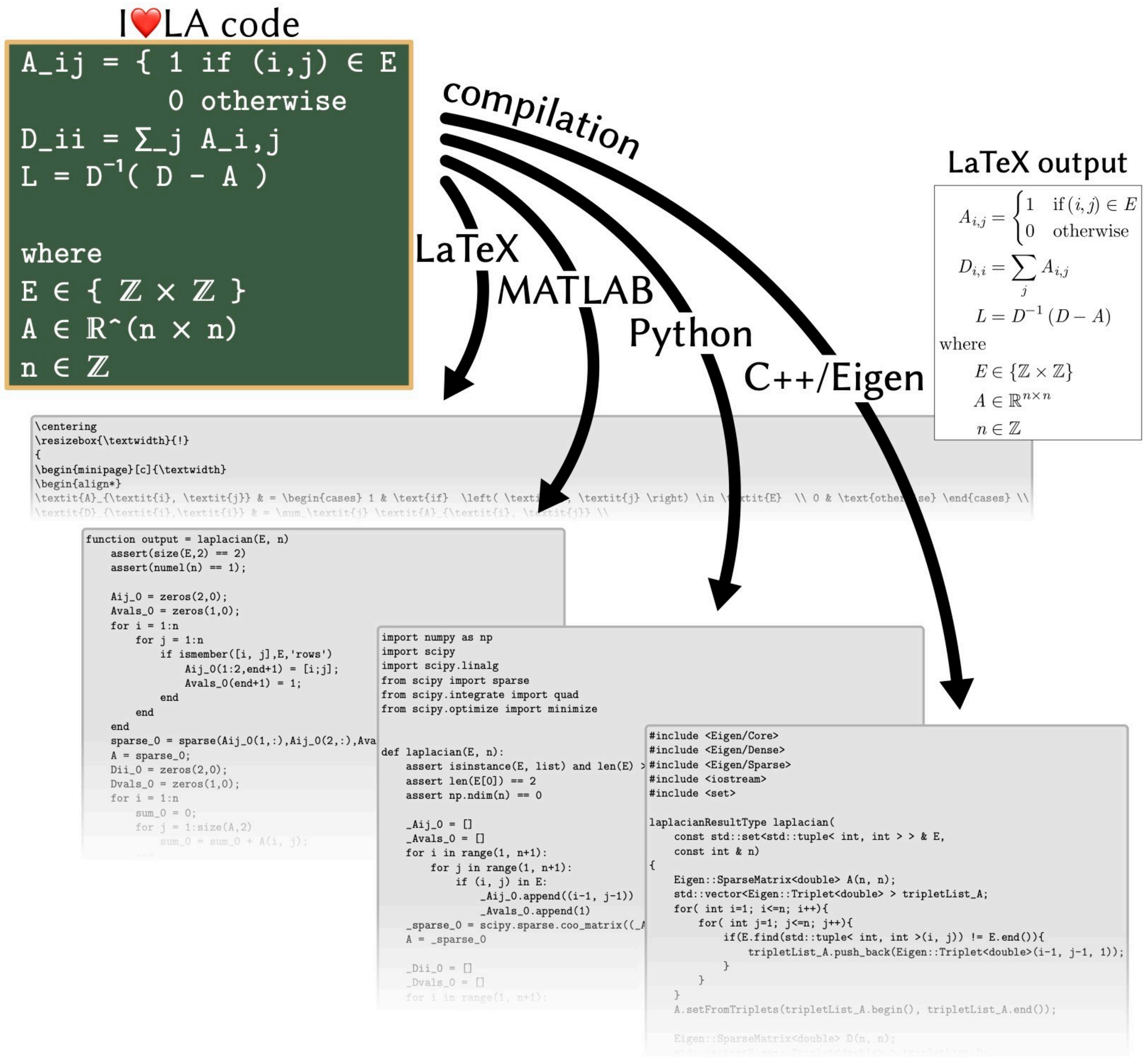
Acknowledgments

- Anonymous reviewers for their suggestions
- Towaki Takikawa for helpful feedback
- Thomas LaToza for a discussion on evaluating programming languages

Acknowledgments

- Anonymous reviewers for their suggestions
- Towaki Takikawa for helpful feedback
- Thomas LaToza for a discussion on evaluating programming languages

- Sponsors:
  - Canada Research Chairs Program
  - United States National Science Foundation (IIS-1453018)
  - Adobe



iheartla.github.io

yli69@gmu.edu