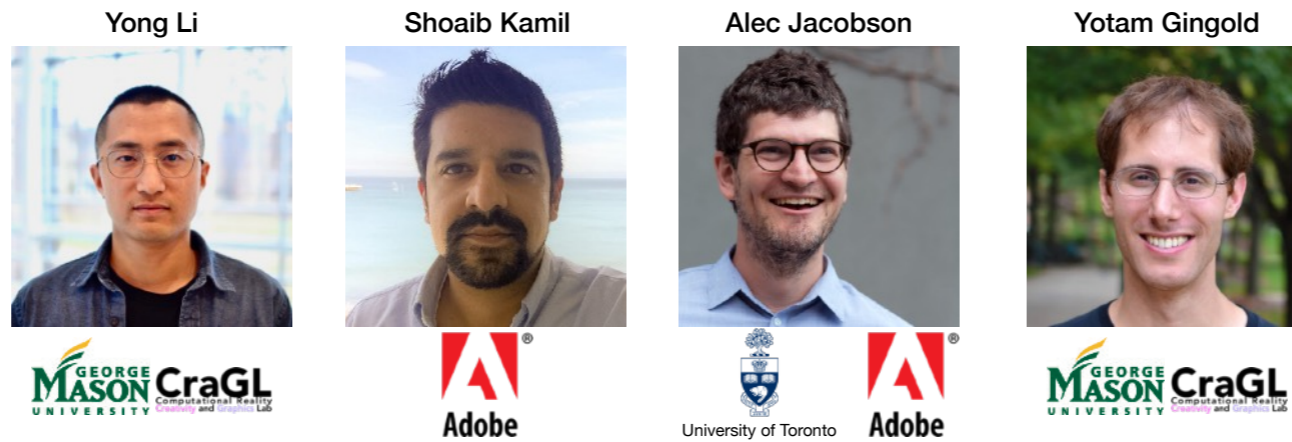


H♥rtDown: Document Processor for Executable Linear Algebra Papers



Thank you. This talk should be given by my PhD student Yong Li. He couldn't be here today for visa reasons.

%%%

Welcome to the talk, my name is Yong Li, I'm a PhD student in George Mason University.
We create a document processor called H♥rtDown for executable linear algebra papers.

This is a joint work with
Dr. Shoaib Kamil from Adobe Research,
Prof. Alec Jacobson from University of Toronto and Adobe Research,
And my advisor Prof. Yotam Gingold from George Mason University.

H♥rtDown



H♥rtDown is an environment for reading and writing scientific documents. Instead of writing formulas in latex, you write them in I♥LA.

```

4 # Surface Fairing
5 ♥: fairing
6
7 Surface fairing given boundary constraints depends on the order of the Laplacian. A simple graph Laplacian  $L$  can be written in terms of the adjacency matrix  $A$  and the degree matrix  $D$ . Those matrices can be derived purely from the the edges of the mesh  $E$ .
8 ```iheartla
9  $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 1 & \text{if } (j,i) \in E \\ 0 & \text{otherwise} \end{cases}$ 
10
11  $D_{ii} = \sum_j A_{ij}$ 
12  $L = D^{-1} (D - A)$ 
13 where
14  $E \in \{ \mathbb{Z} \times \mathbb{Z} \}$  index
15  $A \in \mathbb{R}^{(n \times n)}$ : The adjacency matrix
16  $n \in \mathbb{Z}$ : The number of mesh vertices
17 ```
18
19
20 We then solve a system of equations  $Lx = 0$  for free vertices to obtain the fair surface. We can write the fair mesh vertices  $V'$  directly given boundary constraints provided as a binary vector  $B$  with 1's for boundary vertices, a large scalar constraint weight  $w=10^6$ , and 3D vertices for the constrained mesh  $V$ :
21 ```iheartla
22 diag from linearalgebra
23
24  $V' = (L + w \text{diag}(B))^{-1} (w \text{diag}(B) V)$ 
25 where

```

H♥rtDown is an environment for reading and writing scientific documents. Instead of writing formulas in latex, you write them in I♥LA.

H♥rtDown Editor

```
1 ---
2 full_paper: False
3 ---
4 # Surface Fairing
5 w: fairing
6
7 Surface fairing given boundary constraints depends on the order of the Laplacian. A simple
8 class="def">graph Laplacian  $L$  can be written in terms of the adjacency matrix  $A$  and the
9 class="def">degree matrix  $D$ . Those matrices can be derived purely from the
10 class="def">edges of the mesh  $E$ .
11
12  $A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$ 
13  $D_{ii} = \sum_j A_{ij}$ 
14  $L = D - A$ 
15 where
16  $E \in \mathbb{Z}^2$ : index
17  $A \in \mathbb{R}^{(n \times n)}$ : The adjacency matrix
18  $n \in \mathbb{Z}$ : The number of mesh vertices
19 ---
20 We then solve a system of equations  $Lx = 0$  for free vertices to obtain the fair surface. We can write
21 class="def">the fair mesh vertices  $V$  directly given class="def">boundary constraints
22 provided as a binary vector  $b$  with 1's for boundary vertices, a large scalar class="def">
23 class="def">constraint weight  $w$ , and  $n$  vertices for the constrained mesh
24  $V \in \mathbb{R}^{(n \times 3)}$ .
25
26 class="def">diag from linearalgebra
27  $V = (L + w \text{diag}(b))^{-1} (w \text{diag}(b) V)$ 
28 where
29  $b \in \mathbb{R}^n$ 
30  $V \in \mathbb{R}^{(n \times 3)}$ 
31 ---
32 class="def">figure
33 class="def">python
34 from lib import *
35 import make_cylinder
36
37 # Load cylinder with n vertices
38 mesh = make_cylinder.make_cylinder( 10, 10 )
39 make_cylinder.save_obj( mesh, 'input.obj', clobber = True )
40 V = mesh.v
41 F = mesh.fv
42 n = len(V)
43
44 # Extract the mesh edges
45 edges = set()
46 for face in F:
47     for fvi in range(3):
48         v1, v2 = face[fvi], face[(fvi+1)%3]
49         edges.add( ( min(v1, v2), max(v1, v2) ) )
50
51 # The constraint vector is all vertices with  $z < 1/4$  or  $z > 3/4$ 
52 b = np.zeros( n, dtype = int )
53 b[ V[:,2] < 1/4 ] = 1
54 b[ V[:,2] > 3/4 ] = 1
55
56 # Rotate the top around the z axis by 90 degrees.
57 R = np.array([[ 1, 0, 0 ],
```

By compiling the math, H♥rtDown augments the paper with clickable definitions

```
simple <span  
x  $\$A\$$  and the <span  
span class="def">the  
  
face. We can write  
boundary constraints  
<span  
or the constrained mesh
```

By compiling the math, **H♥rtDown** augments the paper with clickable definitions

```

1  """
2  full_paper: False
3  """
4  # Surface Fairing
5  # fairing
6
7  Surface fairing given boundary constraints depends on the order of the Laplacian. A simple python
8  class def graph Laplacian l is l can be written in terms of the adjacency matrix A and the degree
9  matrix D. Those matrices can be derived purely from the edges of the mesh edges.
10
11  """
12  """
13  A_ij = 1 if (i,j) in E
14         1 if (j,i) in E
15         0 otherwise
16
17  D_ii = sum_j A_ij
18  L = D - A
19
20  where
21  E in [ (x,y) ] index
22  A in R^(n,n): The adjacency matrix
23  n in Z: The number of mesh vertices
24
25
26  We then solve a system of equations Lx = 0 for free vertices to obtain the fair surface. We can write
27  python class def fair the fair mesh vertices V directly given python class def boundary constraints
28  provided as a binary vector B with 1's for boundary vertices, a large scalar alpha
29  class def constraint weight w, and python class def 3D vertices for the constrained mesh
30  python class def
31
32  """
33  """
34  """
35  """
36  """
37  """
38  """
39  """
40  """
41  """
42  """
43  """
44  """
45  """
46  """
47  """
48  """
49  """
50  """
51  """
52  """
53  """
54  """
55  """
56  """

```

Missing descriptions for symbols:
fairing: D

Compile

1 Surface Fairing

Surface fairing given boundary constraints depends on the order of the Laplacian. A simple graph Laplacian L can be written in terms of the adjacency matrix A and the degree matrix D . Those matrices can be derived purely from the edges of the mesh E .

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

$$D_{ii} = \sum_j A_{ij}$$

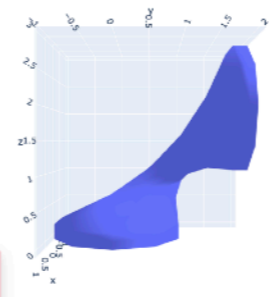
$$L = D - A$$

We then solve a system of equations $Lx = 0$ for free vertices to obtain the fair surface. We can write the fair mesh vertices V directly given boundary constraints provided as a binary vector B with 1's for boundary vertices, a large scalar constraint weight w , and 3D vertices for the constrained mesh V .

$$V = (L + w \text{diag}(B))^{-1} (w \text{diag}(B) V)$$

Glossary of fairing

- $A \in \mathbb{R}^{n \times n}$: The adjacency matrix
- $B \in \mathbb{Z}^n$: boundary constraints provided as a binary vector B with 1's for boundary vertices
- $D \in \mathbb{R}^{n \times n}$
- E set type: the edges of the mesh E
- $L \in \mathbb{R}^{n \times n}$: graph Laplacian
- L
- $V \in \mathbb{R}^{n \times 3}$: 3D vertices for the constrained mesh V
- $V \in \mathbb{R}^{n \times 3}$: the fair mesh vertices V
- $n \in \mathbb{Z}$: The number of mesh vertices
- $w \in \mathbb{R}$: constraint weight



Fairing the middle half of a cylinder.

and warns you when you've forgotten to describe a variable

HotDown Editor

```

1
2 full_paper: False
3
4 # Surface Fairing
5 # Fairing
6
7 Surface fairing given boundary constraints depends on the order of the Laplacian. A simple
8
9 class def graph Laplacian L:
10     """
11     graph Laplacian L:
12     """
13     def __init__(self, E):
14         self.E = E
15         self.L = L
16         self.D = D
17         self.L = D - A
18         self.n = n
19
20 We then solve a system of equations Lx = 0 for free vertices to obtain the fair surface. We can write
21
22 class def fair mesh vertices V:
23     """
24     fair mesh vertices V:
25     """
26     def __init__(self, E, B, w):
27         self.E = E
28         self.B = B
29         self.w = w
30         self.L = L
31         self.D = D
32         self.L = D - A
33         self.n = n
34
35 # Load cylinder with n vertices
36 mesh = make_cylinder.make_cylinder(10, 10)
37 mesh = mesh.v
38 V = mesh.v
39 F = mesh.fv
40 n = len(V)
41
42 # Extract the mesh edges
43 edges = set()
44 for face in F:
45     for fvs in range(3):
46         v1, v2 = face[fvs], face[(fvs+1)%3]
47         edges.add((min(v1, v2), max(v1, v2)))
48
49 # The constraint vector is all vertices with z < 1/4 or z > 3/4
50 B = np.zeros(n, dtype = int)
51 B[ V[:,2] < 1/4 ] = 1
52 B[ V[:,2] > 3/4 ] = 1
53
54 # Rotate the top around the z axis by 90 degrees.
55 B = np.array([1, 0, 0]).

```

1 Surface Fairing

Surface fairing given boundary constraints depends on the order of the Laplacian. A simple graph Laplacian L can be written in terms of the adjacency matrix A and the degree matrix D . These matrices can be derived purely from the edges of the mesh E .

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

$$D_i = \sum_j A_{ij}$$

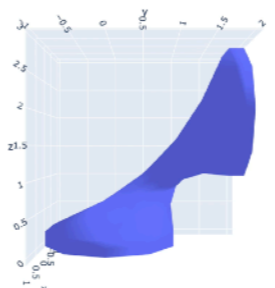
$$L = D - A$$

We then solve a system of equations $Lx = 0$ for free vertices to obtain the fair surface. We can write the fair mesh vertices V' directly given boundary constraints provided as a binary vector B with 1's for boundary vertices, a large scalar constraint weight $w = 10^5$, and 3D vertices for the constrained mesh V .

$$V' = (L + w \text{diag}(B))^{-1} (w \text{diag}(B)V) \quad (2)$$

Glossary of fairing

- $A \in \mathbb{R}^{n \times n}$: The adjacency matrix
- $B \in \mathbb{Z}^n$: boundary constraints provided as a binary vector B with 1's for boundary vertices
- $D \in \mathbb{R}^{n \times n}$: degree matrix
- E set type: the edges of the mesh
- $L \in \mathbb{R}^{n \times n}$: graph Laplacian
- $V \in \mathbb{R}^{n \times 3}$: 3D vertices for the constrained mesh
- $V' \in \mathbb{R}^{n \times 3}$: the fair mesh vertices
- $n \in \mathbb{Z}$: The number of mesh vertices
- $w \in \mathbb{R}$: constraint weight



Fairing the middle half of a cylinder.

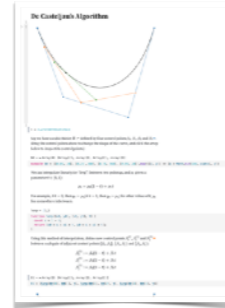
The compiled math can be used to generate figures.

Updating the formula will change both the typeset math and the figure.

Related Work

Literate programming environments

- Literate Programming [Knuth 1984]
- Markdown [Gruber and Swartz 2004]
- Notebooks [Amon 1988; Kery et al. 2018; Rule et al. 2018; Wolfram 1988]
- Pluto [Plas 2020]
- Observable [Bostock 2017]



Reactive documents and publishing

- Idyll [Conlen and Heer 2018]
- Tangle [Victor 2011]
- Distill [Team 2021]
- Authorea [Goodman et al. 2017]
- Nota [Crichton 2021]
- [Bonneel et al. 2020]
- ScholarPhi [Head et al. 2021]



Compilable math and augmentations

- Fortress [Allen et al. 2005]
- Lean [de Moura et al. 2015]
- Julia [Bezanson et al. 2017]
- [Alcock and Wilkinson 2011]
- [Dragunov and Herlocker 2003]
- [Head et al. 2021, 2022]
- Penrose [Ye et al. 2020]
- I♥LA [Li et al. 2021]

```
given
p_i ∈ ℝ³: points on lines
d_i ∈ ℝ³: unit directions along lines

P_i = ( I_3 - d_i d_i^T )
q = ( ∑_i P_i )⁻¹ ( ∑_i P_i p_i )
```

There is related work on ...

* Literate programming environments, which duplicate math and code. H♥rtDown avoids this duplication.

* H♥rtDown is related to reactive documents and publishing. H♥rtDown focuses on helping scientific document users correctly author, read, and experiment with mathematical formulas.

* We make use of languages for compiling math and ideas for augmenting math.

Design Goals

- Support **authoring, reading**, and making use of (**experimenting** with)
 - Correct and reproducible documents
 - Minimal authoring overhead
- **Ecological compatibility**
 - Don't change **what** authors put in papers (prose, math, figures, tables)
 - Minimal changes to **how** they write
 - Plain text documents

We have two design goals.

The first is to support authoring, reading, and making use of (experimenting with)

Correct and reproducible documents

With Minimal authoring overhead

The second is to provide ecological compatibility which means

We don't want to change/restrict what authors put in papers (prose, math, figures, tables)

And we want minimal changes to how they write, e.g.: We prefer plain text documents



To inform our design, we analyzed 156 papers from the SIGGRAPH North America 2020 Technical Papers program.

% collecting both quantitative and qualitative observations.

Formative Study

- All appear to be written using LaTeX.
- Observations:
 - I. Prose organizes the document, interleaved with math.
 - II. Math appears out of order. Symbols used before defined.

Our formative study found that

All papers appear to be written using LaTeX.

Other Observations include

- (I) Prose organizes the document. Mathematical expressions appear between paragraphs of prose or inline.
- (II) Math symbols are often used before they are defined, as determined by the prose.

Equations in papers often depend on each other.

Math appears out of order [Wronski et al. 2019]



Let's see a typical example from this paper. (# this paper is not from SIGGRAPH 2020, but it better demonstrates the dependence)

If we zoom in the six and seventh pages.

Math appears out of order [Wronski et al. 2019]

284 • Wronski et al.

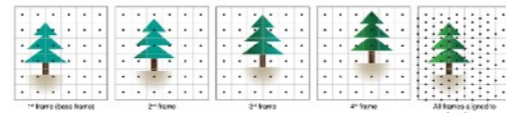


Fig. 4. Subpixel displacements from handfield motion. Illustration of a kernel of four frames with linear hand motion. Each frame is offset from the previous frame by half a pixel along the x-axis and a quarter pixel along the y-axis due to the hand motion. After alignment to the base frame, the pixel centers (black dots) uniformly cover the sampling grid (grey lines) at an increased density. In practice, the distribution is more uniform than in this simplified example.

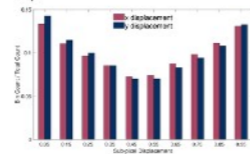


Fig. 5. Distribution of estimated subpixel displacements. Histogram of red and green displacements as computed by the alignment algorithm (Section 3.2). While the alignment process is based on edge-related values, we observe a different coverage of sampled values to maintain representation. Note that displacements in x and y are not correlated.



Fig. 6. Sparse data reconstruction with anisotropic kernels. A sparse sample of very blurry (i.e., $\sigma_{blur} = 0.15$ px) kernels on a real-world scene. For demonstration purposes, we represent samples corresponding to white RGB input patches instead of separate color channels. Kernel adaptation allows us to apply differentially anisotropic kernels through the filter to emphasize different features. The original kernel is aligned with the filter so we observe a blur gradient as being in k_x (red), and the k_y (blue) is used to enhance the resolution in the presence of details.

5.1 Kernel Reconstruction

The core of our algorithm is built on the idea of creating pixels of multiple raw Bayer frames as irregularly offset, aligned and registered measurements of three different underlying continuous pixels, one for each color channel of the Bayer mosaic. Though the color channels are often correlated, in the case of saturation colors like orange, red, green or blue which they are not. Using different spatial coverage, separate per-channel reconstruction allows us to recover the original high resolution signal even in these cases.

To produce the final output image we process all frames separately - for every output image pixel, we evaluate local contributions to the red, green and blue color channels from different input frames. Every input raw image pixel has a different k_x and k_y and contributes only to a specific output color channel. Local contributions are weighted; therefore, we accumulate weights, probabilities and weights. At the end of the pipeline, these probabilities are normalized. For each color channel, this can be expressed as:

$$C(x, y) = \frac{\sum_{i=1}^N w_i \cdot R_i}{\sum_{i=1}^N w_i \cdot R_i} \quad (1)$$

ACM Trans. Graph., Vol. 38, No. 4, Article 28. Publication date: July 2019.

where (x, y) are the pixel coordinates, the sum $\sum_{i=1}^N$ is over all one subsampling frames, $\sum_{i=1}^N$ is a sum over samples within a local neighborhood (in our case 3×3). C_{ij} denotes the value of the Bayer pixel at Bayer frame i and sample j , w_{ij} is the local sample weight and R_i is the local robustness (Section 3.2). In the case of the base frame, R is equal to 1 as it does not get aligned, and we have full confidence in its local sample values.

To compute the local pixel weights, we use local radial basis function kernels, similarly to the non-parametric kernel regression framework of Takida et al. [2016, 2017]. Unlike Takida et al., we don't alternate kernel basis function parameters at sparse sample positions. Instead, we evaluate them at the final resampling grid positions. Furthermore, we always look at the nine closest samples in a 3×3 neighborhood and use the same kernel function for all those samples. This allows for efficient parallel evaluation on a GPU. Using this 'softmax' approach every output pixel is independently processed only once per frame. This is similar to work



Fig. 7. Anisotropic Kernels. Left: When isotropic kernels ($k_{x,iso} = 1$, $k_{y,iso} = 1$), only a global mean will be used, small weight differences cause blurry edges along edges. Right: Anisotropic kernels ($k_{x,iso} = 0$, $k_{y,iso} = 1$) in the center.

Yu and Turk [2013], developed for fluid rendering. Two steps described in the following sections are: estimation of the kernel shape (Section 3.3) and robustness based sample contributions weighting (Section 5.2).

5.1.1 Local Anisotropic Merge Kernels. Given our problem formulation, kernel weights and kernel functions define the image quality of the final merged image. Kernels with wide spatial support produce noise-free and artifact-free, but blurry images, while kernels with very narrow support can produce sharp and detailed images. A natural choice for kernels used for signal reconstruction are Radial Basis Function kernels - in our case anisotropic Gaussian kernels. We can adjust the kernel shape to different local properties of the input frames: amounts of detail and the presence of edges (Figure 6). This is similar to kernel selection techniques used in other sparse data reconstruction applications [Takida et al. 2016, 2017; Yu and Turk 2013].

Specifically, we use a 2D normalized anisotropic Gaussian RBF

$$k_{i,j} = \exp\left(-\frac{1}{2} \mathbf{x}^T \mathbf{\Omega}^{-1} \mathbf{x}\right) \quad (2)$$

where $\mathbf{\Omega}$ is the kernel covariance matrix and \mathbf{x} is the offset vector of samples in the output image ($\mathbf{x} = (x_i - x_j, y_i - y_j)^T$).

One of the main motivations for using anisotropic kernels is that they decrease the algorithm's tolerance for small misalignments and uneven coverage around edges. Edges are ambiguous in the alignment procedure due to the aperture problem and result in alignment errors [Robinson and Sifkoff 2004] even frequently compared to non-edge regions of the image. Subpixel misalignment as well as a lack of sufficient sample coverage can manifest as jitter artifacts (Figure 9). By stretching the kernel along the edges, we can enforce the assignment of smaller weights to pixels not belonging to edges in the image.

5.1.2 Kernel Covariance Computation. We compute the kernel covariance matrix by analyzing every frame's local gradient structure. To improve runtime performance and resilience to image noise, we analyze gradients of half-resolution images formed by downsampling the original raw frames by a factor of two. To estimate a Bayer image containing different color channels, we create a single

Handfield Multi-Frame Super-Resolution • 287

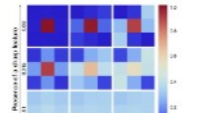


Fig. 8. Merge kernels. Heat of relative weights in different 3×3 sampling kernels as a function of local feature features.

pixel from a 2×2 Bayer quad by combining four different color channels together. This way, we can operate on single channel Bayer images and perform the computation of a quarter of the full resolution cost and with improved signal to noise ratio. To estimate local information about strength and direction of gradients, we use gradient structure tensor analysis [Bigun et al. 1991; Harris and Stephens 1988]:

$$\mathbf{\Omega} = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad (3)$$

where I_x and I_y are the local image gradients in horizontal and vertical directions, respectively. The image gradients are computed by finite forward differencing the luminance in a small 3×3 color window (going to four different horizontal and vertical gradient values). Eigenanalysis of the local structure tensor $\mathbf{\Omega}$ gives two orthogonal direction vectors $\mathbf{e}_1, \mathbf{e}_2$ and two associated eigenvalues

$$\mathbf{\Omega} = \mathbf{e}_1 \lambda_1 \mathbf{e}_1^T + \mathbf{e}_2 \lambda_2 \mathbf{e}_2^T \quad (4)$$

where λ_1 and λ_2 contain the desired tensor variance in either edge or orthogonal direction. We convert these values to robustly estimate super-resolution and denoising. We use the magnitude of the structure tensor's dominant eigenvalue λ_1 to derive the spatial support of the kernel and the trade-off between the super-resolution and denoising, where $\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}$ is used to derive the desired anisotropy of the kernels (Figure 8). The specific process we use to compute the final kernel covariance can be found in the supplemental material along with the tuning values. Since $\mathbf{\Omega}$ is computed at half of the Bayer image resolution, we upsample the kernel covariance values through bilinear sampling before computing the kernel weights.

5.2 Motion Robustness

Reliable alignment of an arbitrary sequence of images is extremely challenging - because of both theoretical [Robinson and Sifkoff 2004] and practical (available computational power) limitations. Even assuming the existence of a perfect registration algorithm, changes in scene and cameras can result in some areas of the photographed scene being superrepresented in many frames of the

ACM Trans. Graph., Vol. 38, No. 4, Article 28. Publication date: July 2019.

We can see that (*) equation one defines a function C that uses w which is (*) defined in the second equation, meanwhile, omega in the second equation is defined in (*) the fourth equation.

For each equation, there's a prose block (*) after the equation describing all the symbols in that equation

Formative Study

- All appear to be written using LaTeX.
- Observations:
 - I. Prose organizes the document, interleaved with math.
 - II. Math appears out of order. Symbols used before defined.
 - III. Symbols re-used in different contexts.

This is an excellent fit to the psychophysical data, with a mean absolute error of 0.24 (equivalent to 9.4%) between measured and predicted judder at the probed points. To present the reader with an error metric that relates to physical quantities, we also computed the mean error in the log-luminance domain (to avoid under representing errors in low-luminance conditions). Given N as the number of measured conditions, $O(i)$ being the observed means for each condition and $M(i)$ values predicted by our model, we calculate the error E as

$$E = \sum_{i=1}^N \frac{|\log(O(i)) - \log(M(i))|}{\log(O(i))} / N, \quad (2)$$

If we introduce the simplifying assumption that the critical flicker fusion rate (CFF) is linearly correlated through a factor M with judder-sensitivity, then we can obtain a log-luminance equivalence like the one queried in this experiment. Denoting F_a and F_b as the two frame rates and L_a, L_b as the luminances:

$$F_a = M * CFF(L_a) = M(a * \log(L_a) + b), \quad (4)$$

[Chapiro et al. 2019]

We also found that

(I) Symbols may be re-used, but the different context is clear to the reader.

For example, the M symbols have different meanings in these equations.

Formative Study

- All appear to be written using LaTeX.
- Observations:
 - I. Prose organizes the document, interleaved with math.
 - II. Math appears out of order. Symbols used before defined.
 - III. Symbols re-used in different contexts.
 - IV. Symbol appears in executable formulas and non-executable derivations.

We now consider the nine possible deformations \bar{u}_e^{ij} generated by setting $f = e_i$ and $g = e_j$ for every pair (i, j) , where the vectors $\{e_1, e_2, e_3\}$ form an orthonormal bases spanning \mathbb{R}^3 . Due to superposition, we can linearly combine \bar{u}_e^{ij} with scalar coefficients F_{ij} , and obtain a matrix-driven solution of (2) of the form

$$\bar{u}_e(r) = \sum_{ij} F_{ij} e_j \cdot \nabla(\mathcal{K}_e(r) e_i) = \nabla \mathcal{K}_e(r) : F, \quad (12)$$

where $F = [F_{ij}]$ is a 3×3 force matrix, and the symbol $:$ indicates the double contraction of F to the third-order tensor $\nabla \mathcal{K}_e(r)$, thus returning a vector. Similarly, we can write the body load that generates

By computing the spatial derivatives of u_e , we obtain the displacement field $\bar{u}_e(r)$ in terms of the force matrix F :

$$\bar{u}_e(r) = -a \left(\frac{1}{r_e^2} + \frac{3e^2}{2r_e^3} \right) Fr + b \left[\frac{1}{r_e^2} (F + F^t + \text{tr}(F)I) - \frac{3}{r_e^3} (r^t F r) I \right] r. \quad (14)$$

[De Goes and James 2017]

(I) A symbol may appear in both derivations and executable formulas.

For example, the equation 12 is the derivation for the function \mathbf{u} while equation 14 is executable.

Formative Study

- All appear to be written using LaTeX.
- Observations:
 - I. Prose organizes the document, interleaved with math.
 - II. Math appears out of order. Symbols used before defined.
 - III. Symbols re-used in different contexts.
 - IV. Symbol appears in executable formulas and non-executable derivations.
 - V. Symbols and functions appear with conditional assignment.
 - VI. Functions have a variety of implied semantics for parameters and pre-computed symbols.

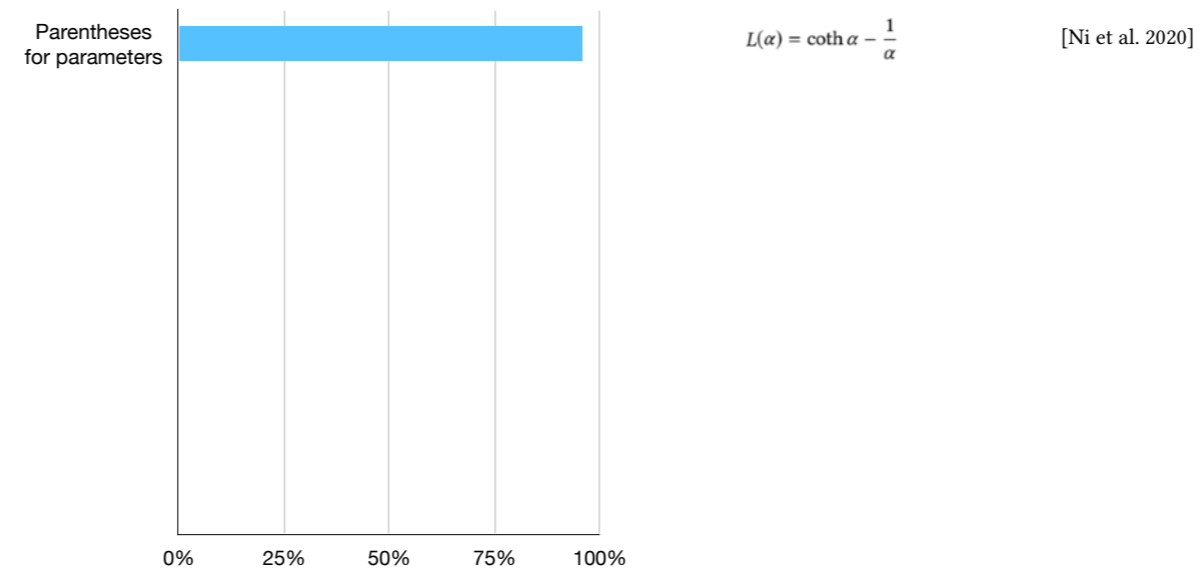
(I) Symbols and functions may be defined via conditional assignment, a simple form of control flow

(II) Functions make use of a variety of implied semantics for parameters

$f_1(y) = \begin{cases} -\frac{y^2}{c^2 k^2} + \frac{2y}{c_0 k}, & y \in (0, h \epsilon_v) \\ 1, & y \geq h \epsilon_v. \end{cases} \quad (13)$	$R(\lambda) = r_{as} + \sum_{k=0}^{\infty} r_{as} r_{sa} \left(r_{10}^2 e^{i\lambda \phi} \right)^k e^{i\lambda \phi} r_{sa} \quad (27)$	$\Psi(d) = \begin{cases} 1, & d = 0 \\ 1/n, & d > 0 \text{ and } d \leq h \\ 0, & d > h \end{cases} \quad (32)$	$E[(f)_{\text{SMS}}] = E[(f)_{\text{SMS}}]_{(t_1, x_1), \dots, (t_n, x_n)} \quad (33a)$
$\bar{V}_k(d) = \int_{\mathcal{X}^d} \gamma_k(d, \mathcal{V}, d) d\mathcal{V}. \quad (1)$	$E[L_{X_p}] = \int_0^\pi \cos \phi_1 \frac{\sin^{n-2} \phi_1 d\phi_1}{i_{n-2}} \quad (24)$	$\log(\alpha_T) = \lambda^{-1} \int_{V(T)} w(r) \log(\alpha_T)^2 dr, \quad (3)$	$E[(f)_{\text{SMS}}] = E[E[(f)_{\text{SMS}}]_{t_1, \dots, t_n} t_1, \dots, t_n}] \quad (33b)$
$O_{\text{num}}(l) = \sum_i^{H_i \times W_i} \hat{f}_{m_i}^l(l) \hat{f}_{m_i}^l(l). \quad (3)$	$\bar{b}(f, g) = \begin{cases} \frac{1}{2} \ (x_p - x_f) - (r_{t_p} - r_{t_f})\ _W^2 & \text{if } f - g = 1 \\ 0 & \text{otherwise.} \end{cases}$	$W_{\text{cloth}}(\lambda_1, \lambda_2) = \begin{cases} 0 & \lambda_1 < 1, \lambda_2 < 1 \\ W_{\text{SbVX}}(\lambda_1, \lambda_2, \lambda_1) & \lambda_1 \geq 1, \lambda_2 < \lambda_2(\lambda_1) \\ W_{\text{SbVX}}(\lambda_1, \lambda_2) & \text{otherwise.} \end{cases}$	$E[(f)_{\text{SMS}}] = E\left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33c)$
$\pi(\mathbf{a} \mathbf{s}) = \sum_{i \in \mathcal{E}} w_i(\mathbf{s}) \pi_i(\mathbf{a} \mathbf{s}), \quad w_i(\mathbf{s}) = \frac{\exp(g_i(\mathbf{s}))}{\sum_{i \in \mathcal{E}} \exp(g_i(\mathbf{s}))} \quad (8)$	$J_{RL}(\theta) = E \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \right]. \quad \Gamma_{n,b}(z) := \left(S(a, b) + z n(a, b) : z \in \left[-\frac{h(a, b)}{2}, \frac{h(a, b)}{2} \right] \right)$	$b(\alpha, \beta) := \begin{cases} \sum_{f=1}^n \ e_f \alpha \beta\ \left(1 + \sum_{i \in \alpha \cap \beta} \ \mathbf{x}_{f_i} - \bar{\mathbf{x}}_i\ \right) & \text{if } q_\alpha \neq q_\beta \\ 0 & \text{otherwise,} \end{cases} \quad (3)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \int_{\mathcal{X}} w(x, t_i) \frac{f(x)}{\rho(x t_i)} dx \right]_{t_1, \dots, t_n} = I. \quad (33d)$
$D(n, m) = \sum_{k \in \mathcal{K}} d(n+k, m+k) \quad (1)$	$F_{s \rightarrow f}(X_p) = \frac{\rho(\mathbf{u}_b - \mathbf{u}_s) - n}{\Delta t} \quad \mathcal{L}(H(\bar{x}), \tilde{H}(\bar{x})) = \frac{1}{2} \sum_{\bar{x}} (H(\bar{x}) - \tilde{H}(\bar{x}))^2 \quad (16)$	$c^{\text{im}}(\mathbf{v}^q, \mathbf{v}^h, \omega_p^q, \omega^h) = \ \mathbf{v}^q - \mathbf{v}^h\ ^2 + \ \omega_p^q - \omega^h\ ^2 \quad (5)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33e)$
$d(n, m) = w_1 \frac{d_k}{ F } + w_2 \frac{d_c}{ F } + w_3 \frac{d_{cc}}{ F } + w_4 d_{\text{root}} \quad (1)$	$a^{(n)}(\mathbf{x}, t) = \int_{\mathcal{X}} \frac{(2\pi)^{n/2}}{e^{-\ \mathbf{u}\ ^2/2}} H^{(n)}(\mathbf{u}) d\mathbf{u} \approx \sum_{i=0}^{q-1} f_i(\mathbf{x}, t) H^{(n)}(\mathbf{e}_i), \quad (7)$	$f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{pmatrix}, \quad f'(x) = \begin{pmatrix} \nabla f_1(x)^T \\ \vdots \\ \nabla f_n(x)^T \end{pmatrix} \quad (11)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33f)$
$\Psi(F^S, J^S) = \Psi^S(F^S) + \Psi^S(J^S), \quad (9)$	$E_{T, \omega}(\ell) = \frac{1}{2} \frac{\ \text{Det}(Y)\ }{\sqrt{\text{Det}(Y^T Y)}} \quad (2)$	$C_d(x) = \frac{1}{V_d(1)} \int_{-\infty}^x R_d(t) dt \quad \mathcal{L}_{\text{iso}}(\theta) = \ \mathbf{r} - f(U_n; \theta)\ _1$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33g)$
$JSD(P Q) = \frac{1}{2} D(P M) + \frac{1}{2} D(Q M) \quad (8)$	$\pi(a_{t+1} s_t, c_t) = \frac{1}{Z(s_t, c_t)} \prod_{i=1}^k \theta_i^{a_i} \quad (3)$	$\phi_H(\nabla \mathbf{u}) = \begin{cases} \frac{1}{2\alpha} (\ \nabla \mathbf{u}\ ^2, \ \nabla \mathbf{u}\ \leq \alpha \\ \ \nabla \mathbf{u}\ - \frac{\alpha}{2}, \ \nabla \mathbf{u}\ > \alpha \end{cases}$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33h)$
$d(\mathcal{L}, \ell) = \frac{1}{ \ell } \int_{\mathcal{L}} \min_{c_i \in \mathcal{L}} \text{dist}(\ell_i, \rho_{\theta}) d\rho_{\theta} \quad (2)$	$q^f(\mathbf{x}) = \sum_c q_c^f N_c^{f-1}(\mathbf{x}). \quad (28)$	$C(P) = \sum_{ij} f_{ij}(e_{ij}) + \sum_{ijk} f_{ijk}(e_{ij}, e_{jk}) + \omega_{ij} \sum_{k \neq i, j} f_{ik}(e_{ij}, e_{jk}, e_{ik}) + \sum_{ij} f_{imj}(e_{ij}) \quad (5)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33i)$
$Q(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos^2 2\theta & \sin 2\theta \cos 2\theta & -\sin 2\theta \\ 0 & \sin 2\theta \cos 2\theta & \sin^2 2\theta & \cos 2\theta \\ 0 & \sin 2\theta & -\cos 2\theta & 0 \end{bmatrix}$	$\bar{u}_f(x_k) = \begin{cases} g(x_k), & x_k \in \partial \Omega_c \\ \bar{u}_f(x_{k+1}) + B(x_k) f(y_k) G(x_k, y_k), & \text{otherwise.} \end{cases} \quad (8)$	$c_1(x) = D_1(x) K_1^{-1} \bar{x}, \quad (8)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33j)$
$b_3(x_3, y_3, \lambda) = \tilde{b}_3(x_3, y_3, \lambda) d(x_3, y_3) = \frac{h(x_3, y_3, \lambda)}{(j\lambda f)^2 A} \left(\frac{x_3 + y_3}{\lambda f}, \frac{y_3 + y_3}{\lambda f} \right) \frac{1}{v_0} \sum_{k=-\infty}^{\infty} \delta \left(x_3 - \frac{k}{v_0} \right)$	$f_s(S_n(\mathbf{y}), \mathbf{y}) = -\frac{1}{(n+1)!} (x^{(n+1)}, \mathbb{S}^{n+1} \mathbf{y}) + o(\mathbf{y} ^{n+1}), \quad (19)$	$\phi_H(\nabla \mathbf{u}) = \begin{cases} \frac{1}{2\alpha} (\ \nabla \mathbf{u}\ ^2, \ \nabla \mathbf{u}\ \leq \alpha \\ \ \nabla \mathbf{u}\ - \frac{\alpha}{2}, \ \nabla \mathbf{u}\ > \alpha \end{cases} \quad (16)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33k)$
$V[\ell_i] = \frac{1}{M} \sum_{p=1}^M V \left[\frac{w_i f}{p} \right] - \frac{1}{MN} \sum_{q=1}^M V \left[\frac{w_i f}{q} \right] + \frac{1}{N} V \left[\frac{w_i f}{q} \right] + \left(1 - \frac{1}{N} \right) V \left[\frac{1}{\rho(Z_i)} \int_{\mathcal{A}} w_i(g_2) f(g_2) d\rho(g) \right] - \frac{1}{M} \left(1 - \frac{1}{N} \right) \sum_{i=1}^M V \left[\frac{1}{\rho(Z_i)} \int_{\mathcal{A}} w_i(g_2) f(g_2) d\rho(g) \right]. \quad (9)$	$\text{Var}[\mathbf{x}^*] = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}^{-1} \cdot \text{Var}[y] \cdot \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}^{-1} = \sigma^2 \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}^{-1} \quad (56)$	$E(\Phi) = \int_{\mathcal{A}} \ \mathbb{1}\ _2^2 dA_{\mathcal{B}} + \int_{\mathcal{A}} \ \mathbb{1}^{-1}\ _2^2 dA_{\mathcal{A}} \quad (8)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33l)$
		$E(\mathbf{X}) = \sum_{i=1}^K \sum_{j=0}^{N-1} \lambda_j \sum_{m=0}^K \gamma_{ij}(f_m, v) \omega_{ij} \quad (14)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33m)$
		$t(j) = \begin{cases} p_k, & \text{if } \exists k : j = \Delta_{\Sigma}(k) \\ \{t_{\Psi_k, \Psi_{k+1}}\}(\psi(j)), & \text{else } \exists k : \Delta_{\Sigma}(k) < j < \Delta_{\Sigma}(k+1) \end{cases} \quad (14)$	$E[(f)_{\text{SMS}}] = E \left[\sum_{i=1}^n \sum_{j=1}^n w(x_i, t_i) \frac{f(x_i)}{\rho(x_i t_i)} \right]_{t_1, \dots, t_n, x_1, \dots, x_n} \quad (33n)$

We quantitatively analyzed the 916 function definitions across the 156 SIGGRAPH papers.

Analysis of all 916 function definitions at SIGGRAPH 2020

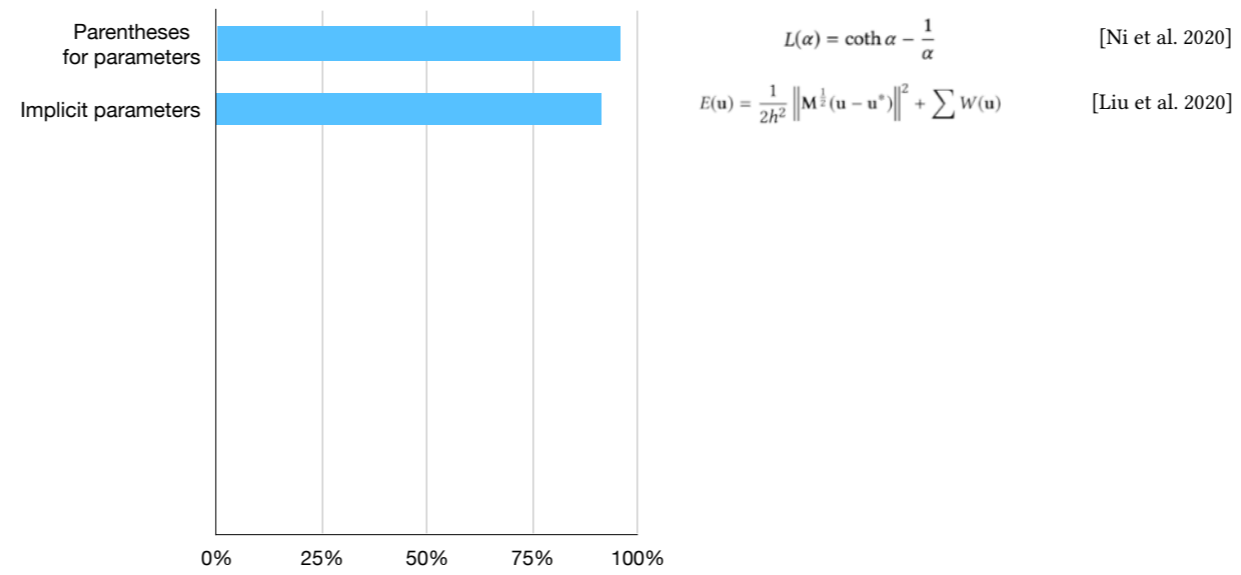


We take an empiric approach to categorize these Equations.

Here is the overview of them.

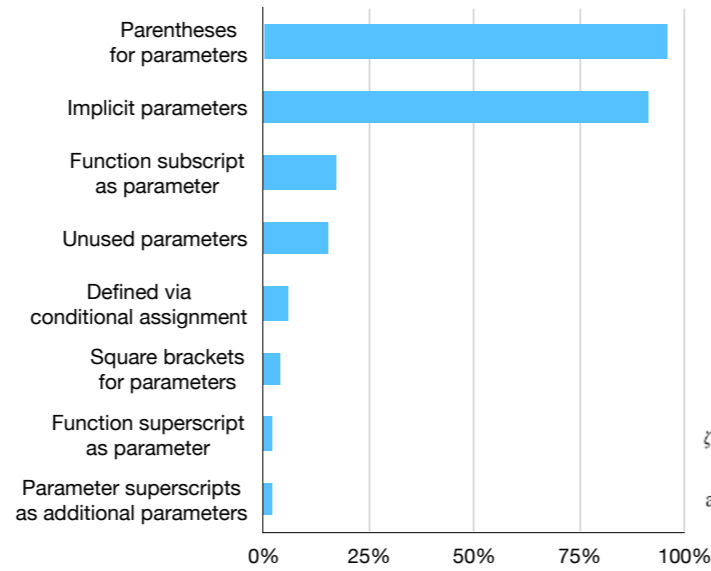
96% use parentheses for parameters,

Analysis of all 916 function definitions at SIGGRAPH 2020



91% rely on implicit parameters,

Analysis of all 916 function definitions at SIGGRAPH 2020



$$L(\alpha) = \coth \alpha - \frac{1}{\alpha} \quad [\text{Ni et al. 2020}]$$

$$E(\mathbf{u}) = \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}}(\mathbf{u} - \mathbf{u}^*) \right\|^2 + \sum W(\mathbf{u}) \quad [\text{Liu et al. 2020}]$$

$$\varphi_p(x) = \frac{p}{2}(x^2 + \epsilon)^{\frac{p}{2}-1} \quad [\text{Lan et al. 2020}]$$

$$S_{SR}(x, y) = \frac{\sum_{i \in \mathcal{N}} w_i \cdot S_i}{\sum_{i \in \mathcal{N}} w_i} \quad [\text{Ma et al. 2020}]$$

$$W(r, h)_{\text{cubic}} = \begin{cases} \frac{2}{3} - r^2 + \frac{1}{2}r^3, & 0 \leq r \leq 1, \\ \frac{1}{6}(2-r)^3, & 1 \leq r \leq 2, \\ 0, & r > 2. \end{cases} \quad [\text{Kim et al. 2020}]$$

$$E[L_{x_p}] = \frac{2}{(n-1)\sqrt{\pi}} \frac{\Gamma\left(\frac{n}{2}\right)}{\Gamma\left(\frac{n-1}{2}\right)} = \frac{2}{n\sqrt{\pi}} \frac{\Gamma\left(\frac{n+2}{2}\right)}{\Gamma\left(\frac{n+1}{2}\right)} \quad [\text{Chiu et al. 2020}]$$

$$\zeta_s^\alpha(x) \equiv \frac{2^j \alpha^{-1}}{(2\pi)^{3/2}} \sum_n \beta_{j,n}^i \zeta_s^{\alpha,n}(x) = \int_{\mathbb{R}_u} \psi_s(S_\alpha(x, u))^T du \quad [\text{Lessig 2020}]$$

$$\text{area}(f^\delta) = \text{area}(f)(1 - 2\delta H(f) + \delta^2 K(f)) \quad [\text{Jiang et al. 2020}]$$

We observed a variety of other less common semantics related to function parameters.

% 2% interpret the parameter superscripts as additional parameters,

Based on these findings, we extend the grammar and implementation of I♥LA to include support for local functions

Formative Study

- All appear to be written using LaTeX.
- Observations:
 - I. Prose organizes the document, interleaved with math.
 - II. Math appears out of order. Symbols used before defined.
 - III. Symbols re-used in different contexts.
 - IV. Symbol appears in executable formulas and non-executable derivations.
 - V. Symbols and functions appear with conditional assignment.
 - VI. Functions have a variety of implied semantics for parameters and pre-computed symbols.
- Pseudocode sometimes present, compilable code isn't. No literate programs.

In addition, Pseudocode sometimes present while compilable code isn't. There's No literate programs.

H♥rtDown Design: Authoring

- Context definition

```
4 # Surface Fairing
5 ♥: fairing
6
7 Surface fairing given boundary constraints depends on the order of the Laplacian. A
  simple graph Laplacian  $LS$  can be written in terms of the
  adjacency matrix  $SA$  and the degree matrix  $SDS$ . Those
  matrices can be derived purely from the edges of the mesh
   $SE$ .
8 `` iheartla
9  $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 1 & \text{if } (j,i) \in E \\ 0 & \text{otherwise} \end{cases}$ 
10
11  $D_{ii} = \sum_j A_{ij}$ 
12  $L = D^{-1} (D - A)$ 
13 where
14  $E \in \{ Z \times Z \}$  index
15  $A \in R^{(n \times n)}$ : The adjacency matrix
16  $n \in Z$ : The number of mesh vertices
17
18
19
20 We then solve a system of equations  $SLx = 0$  for free vertices to obtain the fair
  surface. We can write the fair mesh vertices  $SV$  directly
  given boundary constraints provided as a binary vector  $BS$  with
  1's for boundary vertices, a large scalar  $w$  constraint
  weight, and 3D vertices for the constrained mesh
   $VS$ :
```

We design H♥rtDown based on our formative study.

Just as in LaTeX or many other Markdown formats, the prose is written as plain text with occasional markup commands

(*) Authors must declare a context for their symbols.

The context disambiguates symbol reuse (and corresponds to our concept of modules).

###

This allows better symbol and formula re-use

Later context declarations override earlier declarations.

H♥rtDown Design: Authoring

- Prose descriptions

```
4 # Surface Fairing
5 ♥: fairing
6
7 Surface fairing given boundary constraints depends on the order of the Laplacian. A
simple graph Laplacian  $L$  can be written in terms of the
adjacency matrix  $A$  and the degree matrix  $D$ . Those
matrices can be derived purely from the edges of the mesh
 $E$ .
8 ``heartla
9  $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 1 & \text{if } (j,i) \in E \\ 0 & \text{otherwise} \end{cases}$ 
10
11  $D_{ii} = \sum_j A_{ij}$ 
12  $L = D^{-1} (D - A)$ 
13 where
14  $E \in \{ Z \times Z \}$  index
15  $A \in R^{(n \times n)}$ : The adjacency matrix
16  $n \in Z$ : The number of mesh vertices
17 ``
18
19
20 We then solve a system of equations  $Lx = 0$  for free vertices to obtain the fair
surface. We can write the fair mesh vertices  $V'$  directly
given boundary constraints provided as a binary vector  $B$  with
1's for boundary vertices, a large scalar constraint
weight  $w=10^6$ , and 3D vertices for the constrained mesh
 $V$ :
```

One appearance of a symbol in the prose deserves special attention: the text describing the symbol.

Detecting the span of this description cannot be accurately automated, so we require authors to annotate such spans.

H♥rtDown Design: Authoring

- Executable mathematical expressions

```
4 # Surface Fairing
5 ♥: fairing
6
7 Surface fairing given boundary constraints depends on the order of the Laplacian. A
8 simple graph Laplacian  $LS$  can be written in terms of the
9 adjacency matrix  $SA$  and the degree matrix  $SDS$ . Those
10 matrices can be derived purely from the edges of the mesh
11  $SE$ .
12
13  $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 1 & \text{if } (j,i) \in E \\ 0 & \text{otherwise} \end{cases}$ 
14  $D_{ii} = \sum_j A_{ij}$ 
15  $L = D^{-1} (D - A)$ 
16 where
17  $E \in \{ Z \times Z \}$  index
18  $A \in R^{(n \times n)}$ : The adjacency matrix
19  $n \in Z$ : The number of mesh vertices
20
21 We then solve a system of equations  $SLx = 0S$  for free vertices to obtain the fair
22 surface. We can write the fair mesh vertices  $SV$  directly
23 given boundary constraints provided as a binary vector  $BS$  with
24 1's for boundary vertices, a large scalar  $w$  constraint
25 weight, and 3D vertices for the constrained mesh
26  $SV$ :
```

Authors can write executable mathematical expressions in different I♥LA blocks and inline I♥LA formula. We chose I♥LA since it resembles equations in papers and can generate latex and code for different backends.

% I♥LA requires type declarations for all symbols not appearing on the left-hand side of an equals sign

H♥rtDown Design: Authoring

- I♥LA extensions
 - Local function support
 - Symbol def-use analysis
 - Modules
 - MathJax output includes metadata

We extended I♥LA with new language features.

We add local function support based on the formative study.

In order to handle Math appearing out of order, we add symbol def-use analysis

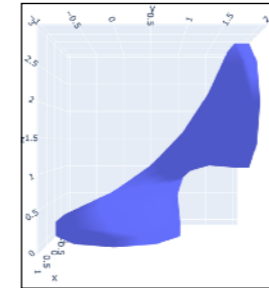
We add Modules to support different contexts

We also modify MathJax output to include metadata for our viewing environment

H♥rtDown Design: Authoring

- Figures

```
30 #Figure
31 """
32 from lib import *
33 import rake_cylinder
34
35 # Load cylinder with n vertices
36 mesh = rake_cylinder.make_cylinder(10, 10)
37 rake_cylinder.save_obj(mesh, "input.obj", clobber = True)
38 V = mesh.v
39 F = mesh.fv
40 n = len(V)
41
42 # Extract the mesh edges
43 edges = set()
44 for face in F:
45     for fvi in range(3):
46         vi,vj = face[Fvi], face[(Fvi+1)%3]
47         edges.add( ( min(vi,vj), max(vi,vj) ) )
48
49 # The constraint vector is all vertices with z < 3/4 or z > 3/4
50 k = rp.zeros( n, dtype = int )
51 [ k[V[i][2] < 3/4] = 1
52   k[V[i][2] > 3/4] = 1
53 ]
54 # Rotate the top around the z axis by 90 degrees.
55 k = rp.array([[ 1, 0, 0 ],
56              [ 0, 0, 1 ],
57              [ 0, -1, 0 ]])
58 for vi in np.where(k> 3/4)[0]: V[vi] = k @ V[vi] + (0,1,0)
59
60 # Solve for new positions
61 result = fairing( E = edges, n = n, k = k, V = V )
62 mesh.v = result.V_astroptre
63 rake_cylinder.save_obj(mesh, "solved.obj", clobber = True)
64
65 import plotly.graph_objects as go
66 fig = go.Figure(data=[go.Mesh3d(
67     x=mesh.v[:,0], y=mesh.v[:,1], z=mesh.v[:,2],
68     i=mesh.fv[:,0], j=mesh.fv[:,1], k=mesh.fv[:,2]
69 )])
70 fig.update_layout(scene_camera={"eye":dict(x=2.5,y=0,z=0), "up":dict(x=0,y=0,z=1)}, margin=dict(l=0, r=0, b=0))
71 fig.write_html("cylinder.html")
72
73 
74 <figcaption>Fairing the middle half of a cylinder.</figcaption>
75 </Figure>
```



H♥rtDown executes Python code blocks, which allows authors to generate figures programmatically

The Python code can access the compiled functionality of the document as a module.

% Authors can also edit I♥LA formulas and Python code for figures directly in the viewer-side of the authoring environment.

H♥rtDown Design: Author support

H♥rtDown Editor

```

7 Surface fairing given boundary constraints depends on the order of the Laplacian. A simple
8 laplacian graph Laplacian  $L = D - A$  can be written in terms of the adjacency matrix  $A$  and the degree
9 degree matrix  $D$ . These matrices can be derived purely from the edges of the mesh  $E$ .
10
11 edges
12  $A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$ 
13  $D_{ii} = \sum_j A_{ij}$ 
14  $L = D^{-1}(D - A)$ 
15
16  $E \in \mathbb{Z}^2$  index
17  $A \in \mathbb{R}^{(2n)}$  the adjacency matrix
18  $n \in \mathbb{Z}$  The number of mesh vertices
19
20 We then solve a system of equations  $Lx = 0$  for free vertices to obtain the fair surface. We can write
21 laplacian the fair mesh vertices  $V$  directly given laplacian boundary constraints
22 provided as a binary vector  $B$  with 1's for boundary vertices. laplacian a large scalar
23 laplacian constraint weight laplacian laplacian 3D vertices for the constrained mesh
24  $V$ .
25
26 laplacian
27  $V = (L + w \text{diag}(B))^{-1}(w \text{diag}(B) V)$ 
28
29
30
31
32 from lib import *
33 report mesh_cylinder
34
35 # Load cylinder with n vertices
36 mesh = mesh_cylinder(mesh_cylinder(10, 10))
37 make_cylinder_solid(mesh, "input.obj", closed = True)
38 V = mesh.V
39 F = mesh.F
40 n = len(V)
41
42 # Extract the mesh edges
43 edges = mesh.E
44 for face in F:
45     for v1 in range(3):
46         v2 = face[v1]
47         edges.add(C min(v1, v2), max(v1, v2) )
48
49 # The constraint vector is all vertices with  $x = 1/4$  or  $x = 3/4$ 
50 B = np.zeros(n, dtype = int)
51 for v in range(n):
52     if V[:, 0] == 1/4 or V[:, 0] == 3/4:
53         B[v] = 1
54
55 # Rotate the top around the z axis by 90 degrees.
56 R = np.array([[ 1, 0, 0 ],
57              [ 0, 0, 1 ],
58              [ 0, -1, 0 ]])
59 for v1 in np.where(V[:, 2] > 3/4*200): V[v1] = R * V[v1] + (0, 0, 2)
60 # Solve for new positions.

```

1 Surface Fairing

Surface fairing given boundary constraints depends on the order of the Laplacian. A simple graph Laplacian L can be written in terms of the adjacency matrix A and the degree matrix D . These matrices can be derived purely from the edges of the mesh E .

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

$$D_{ii} = \sum_j A_{ij}$$

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

We then solve a system of equations $Lx = 0$ for free vertices to obtain the fair surface. We can write the fair mesh vertices V directly given boundary constraints provided as a binary vector B with 1's for boundary vertices, a large scalar constraint weight w , and 3D vertices for the constrained mesh V .

$$V = (L + w \text{diag}(B))^{-1}(w \text{diag}(B) V)$$

Glossary of fairing

$A \in \mathbb{R}^{n \times n}$ The adjacency matrix

$B \in \mathbb{Z}^n$ boundary constraints provided as a binary vector B with 1's for boundary vertices

$D \in \mathbb{R}^{n \times n}$ degree matrix

E set type: the edges of the mesh E

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

L Laplacian

$V \in \mathbb{R}^{n \times 3}$ 3D vertices for the constrained mesh V

V fair mesh vertices V

$n \in \mathbb{Z}$ The number of mesh vertices

$w \in \mathbb{R}$ constraint weight

Dimension mismatch. Can't multiply matrix(n, n) w diag(B) and matrix(n, 3) V.

"V" = [L + w diag(B)]^-1 (w diag(B) V)

Fairing the middle half of a cylinder.

H♥rtDown helps authors write correct math and complete prose.

Error messages appear whenever the user's formulas contain incompatible indices, dimensions, types or erroneous syntax.

(*) The editor displays the L♥LA compiler's error message and highlights the appropriate line in the source.

H♥rtDown Design: Author support

H♥rtDown Editor

```
1
2 full_paper: false
3
4 # Surface Fairing
5 # Surface Fairing
6
7 Surface fairing given boundary constraints depends on the order of the Laplacian. A simple
8
9 The graph Laplacian  $L \in \mathbb{R}^{n \times n}$  can be written in terms of the adjacency matrix  $A$  and the degree
10 matrix  $D$ . Those matrices can be derived purely from the edges of the mesh  $E$ .
11
12 Invertible
13  $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 1 & \text{if } (j,i) \in E \\ 0 & \text{otherwise} \end{cases}$ 
14  $D_{ii} = \sum_j A_{ij}$ 
15  $L = D - A$ 
16 where
17  $E \subseteq \{2d\}$  index
18  $n \in \mathbb{N}$  (non-0): The adjacency matrix
19  $n \in \mathbb{Z}$ : The number of mesh vertices
20
21 We then solve a system of equations  $Lx = 0$  for free vertices to obtain the fair surface. We can write
22
23 The fair mesh vertices  $V^f \in \mathbb{R}^{n \times 3}$  directly given boundary constraints
24 provided as a binary vector  $B \in \mathbb{R}^n$  with 1's for boundary vertices, a large scalar
25  $w \in \mathbb{R}^n$  constraint weight, and  $n - |B|$  vertices for the constrained mesh
26  $V^c \in \mathbb{R}^{(n-|B|) \times 3}$ .
27
28 Invertible
29  $V^f = (L + w \text{diag}(B))^{-1} (w \text{diag}(B) V^c)$ 
30 where
31  $B \in \mathbb{R}^n$ 
32  $w \in \mathbb{R}^n$ 
33
34 # Figure
35
36 Python
37 from math import pi
38 import meshio
39
40 # Load cylinder with n vertices
41 mesh = meshio.read('cylinder.vtu', 10)
42 mesh = meshio.read('cylinder.vtu', 10)
43 V = mesh.points
44 n = mesh.n_vertices
45
46 # Extract the mesh edges
47 edges = set()
48 for face in mesh.faces:
49     for i in range(2):
50         v1, v2 = face[i], face[(i+1)%3]
51         edges.add((min(v1, v2), max(v1, v2)))
52
53 # The constraint vector is all vertices with  $x < -3/4$  or  $x > 3/4$ 
54 B = np.zeros(n, dtype=int)
55 B[V[:,0] < -3/4] = 1
56 B[V[:,0] > 3/4] = 1
57
58 # Rotate the top around the z axis by 90 degrees.
59 R = np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]])
```

1 Surface Fairing

Surface fairing given boundary constraints depends on the order of the Laplacian. A simple graph Laplacian L can be written in terms of the adjacency matrix A and the degree matrix D . Those matrices can be derived purely from the edges of the mesh E .

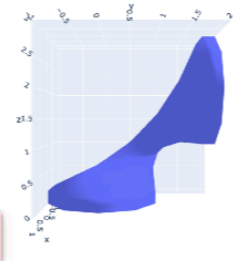
$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 1 & \text{if } (j,i) \in E \\ 0 & \text{otherwise} \end{cases}$$
$$D_{ii} = \sum_j A_{ij}$$
$$L = D - A$$

We then solve a system of equations $Lx = 0$ for free vertices to obtain the fair surface. We can write the fair mesh vertices V^f directly given boundary constraints provided as a binary vector B with 1's for boundary vertices, a large scalar constraint weight $w \in \mathbb{R}^n$, and 3D vertices for the constrained mesh V^c .

$$V^f = (L + w \text{diag}(B))^{-1} (w \text{diag}(B) V^c)$$

Glossary of fairing

- $A \in \mathbb{R}^{n \times n}$: The adjacency matrix
- $B \in \mathbb{R}^n$: boundary constraints provided as a binary vector B with 1's for boundary vertices
- $D \in \mathbb{R}^{n \times n}$: degree matrix
- $E \subseteq \mathbb{R}^2$: type: the edges of the mesh E
- $L \in \mathbb{R}^{n \times n}$: graph Laplacian
- $V^c \in \mathbb{R}^{n \times 3}$: 3D vertices for the constrained mesh V^c
- $V^f \in \mathbb{R}^{n \times 3}$: the fair mesh vertices V^f
- $n \in \mathbb{Z}$: The number of mesh vertices
- $w \in \mathbb{R}$: constraint weight



Fairing the middle half of a cylinder.

Missing descriptions for symbols:
fairing: D

Compile

When symbols are not described anywhere in the prose, they(*) appear with red underlines in the viewer.

H♥rtDown Design: Reading Environment

A Symmetric Objective Function for ICP

SZYMON RUSINKIEWICZ, Princeton University

The Iterative Closest Point (ICP) algorithm, commonly used for alignment of 3D models, has previously been defined using either a point-to-point or point-to-plane objective. Alternatively, researchers have proposed computationally-expensive methods that directly minimize the distance function between surfaces. We introduce a new symmetric objective function that achieves the simplicity and computational efficiency of point-to-plane optimization, while yielding improved convergence speed and a wider convergence basin. In addition, we present a linearization of the objective that is exact in the case of exact correspondences. We experimentally demonstrate the improved speed and convergence basin of the symmetric objective, on both smooth models and challenging cases involving noise and partial overlap.

1 INTRODUCTION

Registration of 3D shapes is a key step in both 3D model creation (from scanners or computer vision systems) and shape analysis. For rigid-body alignment based purely on geometry (as opposed to RGB-D), the most common methods are based on variants of the Iterative Closest Point (ICP) algorithm [Besl and McKay 1992]. In this method, points are repeatedly selected from one model, their nearest points on the other model (given the current best-estimate rigidbody alignment) are selected as correspondences, and an incremental transformation is found that minimizes distances between point pairs. The algorithm eventually converges to a local minimum of surface-to-surface distance.

Because ICP-like algorithms can be made efficient and reliable, they have become widely adopted. As a result, researchers have focused on both addressing the shortcomings of ICP and extending it to new settings such as color-based registration and non-rigid alignment. One particular class of improvements has focused on the loss function that is optimized to obtain an incremental transformation. For example, as compared to the original work of Besl and McKay, which minimized point-to-point distance, the method of [Chen and Medioni 1992] minimized the distance between a point on one mesh and a plane containing the matching point and perpendicular to its normal. This point-to-plane objective generally results in faster convergence to the correct alignment and greater ultimate accuracy, though it does not necessarily increase the basin of convergence. Work by [Fitzgibbon 2003], [Mitra et al. 2004], and [Fotramn et al. 2006] showed that both point-to-point and point-to-plane minimization may be thought of as approximations to minimizing the squared Euclidean distance function of the surface, and they presented algorithms that achieved greater con-

Glossary of ICP

- $\bar{p} \in \mathbb{R}^3$: the averaged coordinate of points
- $\bar{q} \in \mathbb{R}^3$: the averaged coordinate of points
- $\epsilon_{plane} \in \mathbb{R}$: the point-to-plane objective
- $\epsilon_{point} \in \mathbb{R}$: the point-to-point objective
- $\epsilon_{symm-RN} \in \mathbb{R}$: the rotated-normals ("RN") version of the symmetric objective
- $\epsilon_{symm} \in \mathbb{R}$: ϵ_{symm} as the symmetric objective
- $\epsilon_{sum-plane} \in \mathbb{R}$: the sum of squared distances to planes defined by both n_p and n_q
- $n_p \in \text{sequence of } \mathbb{R}^3$: the surface normals
- $n_q \in \text{sequence of } \mathbb{R}^3$: surface normals $n_{q,i}$
- $R \in \mathbb{R}^{3 \times 3}$: a rigid-body transformation (RH) such that applying the transformation to P causes it to lie on top of Q
- $S \in \mathbb{R}^{3 \times 3}$
- $\alpha \in \mathbb{R}$: α and θ are the axis and angle of rotation
- $n \in \text{sequence of } \mathbb{R}^3$
- $p \in \text{sequence of } \mathbb{R}^3$: pairs of corresponding points (p_i, q_i) , where q_i is the closest point to p_i given the current transformation
- $p' \in \text{sequence of } \mathbb{R}^3$
- $q \in \text{sequence of } \mathbb{R}^3$: pairs of corresponding points (p_i, q_i) , where q_i is the closest point to p_i given the current transformation
- $Q \in \text{sequence of } \mathbb{R}^3$
- $rot \in \mathbb{R}, \mathbb{R}^3 \rightarrow \mathbb{R}^{3 \times 3}$: the rotation function
- $t \in \mathbb{R}^3$: a rigid-body transformation (RHt) such that applying the transformation to P causes it to lie on top of Q
- $trans \in \mathbb{R}^3 \rightarrow \mathbb{R}^{3 \times 3}$: the translation function
- $t \in \mathbb{R}^3$
- $d \in \mathbb{R}^3$
- $\theta \in \mathbb{R}$: α and θ are the axis and angle of rotation

H♥rtDown's paper reading environment provides several useful interactions that use the metadata.

Other enhanced reading environments could be created using the metadata H♥rtDown generates. In fact, our augmentations were inspired by the ScholarPhi reading environment [Head et al. 2021].

H♥rtDown Design: Reading Environment

- Glossary

constancy effects [Georgeson and Sullivan 1975].

The results of this experiment can be seen in Figure 4; for simplicity, the plotted data have been averaged over the contrast dimension and participants. By comparing the three plots, we note that frame rate has a powerful effect on mitigating judder, with results at 120 and 60Hz showing little perceived judder, while 30Hz stimuli were all perceived with high levels of judder. A clear trend from the 30Hz plot is that, at this frame rate, judder increases uniformly with luminance. In addition, speed has a nearly linear effect on perceived judder.

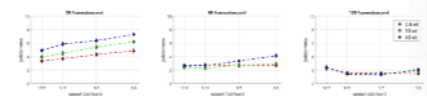


Fig. 4. Results for experiment 1 (moving edge), averaged over participants and contrasts. Vertical lines depict standard error over all samples. Results for 120 (right) and 60 FPS (mid) show little judder. Thirty FPS (left) appeared considerably distorted—judder increases almost linearly with speed, and there is a neat separation between luminance levels (plotted in red, green, and blue), with higher luminances considered to have more judder.

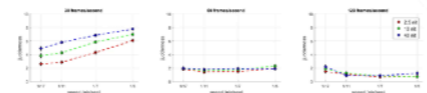


Fig. 5. Results for experiment 2 (panning complex images), averaged over participants and images. Vertical lines depict standard error over all samples. Results are similar to experiment 1, with 120 (right) and 60 FPS (mid) not showing much judder. Thirty FPS (left) continues to present a positive and clearly separable correlation of judder with speed and luminance.

Glossary of judder

- $F_0 \in \mathbb{R}$: Denoting F_0 and F_1 as the two frame rates
- $F_1 \in \mathbb{R}$: Denoting F_0 and F_1 as the two frame rates
- $L_0 \in \mathbb{R}$: L_0, L_1 as the luminances
- $L_1 \in \mathbb{R}$: L_0, L_1 as the luminances
- $CFP \in \mathbb{R} \rightarrow \mathbb{R}$: the critical flicker fusion rate (CFP)
- $F \in \mathbb{R}$: frame rate F
- $J \in \mathbb{R}$: an easily expressible model of judder J
- $L \in \mathbb{R}$: mean luminance L
- $M \in \mathbb{R}$: a factor M
- $P \in \mathbb{R}, \mathbb{R} \rightarrow \mathbb{R}$
- $S \in \mathbb{R}$: speed S
- $a \in \mathbb{R}$: a and b are known constants
- $b \in \mathbb{R}$: a and b are known constants
- $\alpha \in \mathbb{R} \rightarrow \mathbb{R}$: α the logarithm function
- $\beta \in \mathbb{R} \rightarrow \mathbb{R}$: β is the multiplicative inverse

(*)H♥rtDown displays a context-dependent glossary in a fixed position as the page scrolls. The glossary updates automatically with the relevant symbol list.

H♥rtDown Design: Reading Environment

- Symbol definitions

approximate the surface around q_i as planar, which only requires evaluation of surface normals \vec{n}_{q_i} . Indeed, this approach dates back to the work of [Chen and Medioni 1992], who minimized what has come to be called the point-to-plane objective :

$$\varepsilon_{plane} = \sum_i ((Rp_i + t - q_i) \cdot \vec{n}_{q_i})^2 \quad (2)$$

It can be shown that minimizing this $\vec{n}_q \in \text{sequence of } \mathbb{R}^3: \text{ surface normals } n_{q_i}$ minimiza-

When clicking on a symbol, H♥rtDown shows its description and an arrow to the prose that described it.

H♥rtDown Design: Reading Environment

- Equation relationships

where a and θ are the axis and angle of rotation. We observe that the last term in (7) is quadratic in the incremental rotation angle θ , so we drop it to linearize:

$$Rv \approx v \cos \theta + (a \times v) \sin \theta = \cos \theta (v + (\bar{a} \times v)) \quad (8)$$

where $\bar{a} = a \tan(\theta)$. Substituting into (6),

$$\varepsilon_{\text{approx}} \approx \sum_i (\cos \theta (p_i - q_i) \cdot n_i + \cos \theta (\bar{a} \times (p_i + q_i)) \cdot n_i + \bar{a} \cdot n_i)$$

$$\varepsilon_{\text{approx}} = \sum_i \cos(\theta)^2 ((p_i - q_i) \cdot n_i + ((p_i + q_i) \times n_i) \cdot \bar{a} + n_i \cdot \bar{a})^2 \quad (9)$$

where $n_i = n_{x_i} + n_{y_i}$ and $\bar{a} = \frac{a}{\cos(\theta)}$. We now make the additional approximation of weighting the objective by $1/\cos^2 \theta$, which approaches 1 for small θ . Finally, for better numerical stability, we normalize the (p_i, q_i) by translating each point set to the origin and adjusting the solved-for translation appropriately. This yields:

$$\sum_i ((\bar{p}_i - \bar{q}_i) \cdot n_i + ((\bar{p}_i + \bar{q}_i) \times n_i) \cdot \bar{a} + n_i \cdot \bar{a})^2 \quad (10)$$

where $\bar{p}_i = p_i - \bar{p}$ and $\bar{q}_i = q_i - \bar{q}$. This is a least-squares problem in \bar{a} and \bar{a} , and the final transformation from P to Q is:

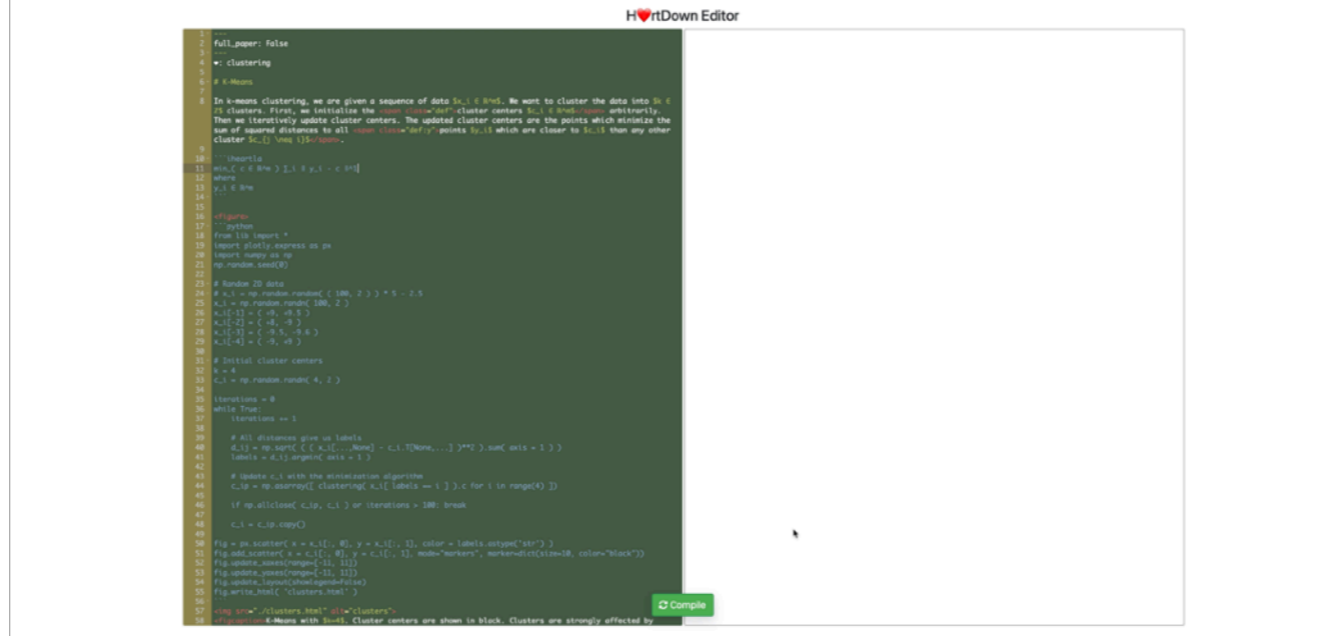
$$S = \text{trans}(\bar{q}) \cdot \text{rot}\left(\theta, \frac{\bar{a}}{\|\bar{a}\|}\right) \cdot \text{trans}(\cos(\theta)) \cdot \text{rot}\left(\theta, \frac{\bar{a}}{\|\bar{a}\|}\right) \cdot \text{trans}(-\bar{p}) \quad (11)$$

(*)When clicking an equation, H♥rtDown highlights the terms involved in the formula as well as downstream uses of the symbol.

H♥rtDown solves a graph coloring problem to color symbols distinctly.

% H♥rtDown solves a graph coloring problem using a greedy technique [Liu et al. 2021] to ensure symbols in the same equation have different colors.

H♥rtDown Design: Experimenter (making use of)



```
1 full_paper: false
2
3 • clustering
4
5 # k-means
6
7 In k-means clustering, we are given a sequence of data  $x_i, i \in [n]$ . We want to cluster the data into  $k$ 
8 clusters. First, we initialize the cluster_centers  $c_i, i \in [k]$  arbitrarily.
9 Then we iteratively update cluster centers. The updated cluster centers are the points which minimize the
10 sum of squared distances to all cluster_centers points  $c_i$  which are closer to  $c_i$  than any other
11 cluster  $c_j, j \neq i$ .
12
13 ---
14
15 # Python
16
17 from IPython import *
18 import numpy as np
19 import random
20
21 # Random 2D data
22 #  $x_i = (x_i[0], x_i[1])$ 
23  $x_i = np.random.randn(100, 2) * 5 - 2.5$ 
24  $c_i = np.random.randn(4, 2)$ 
25  $x_i[0] = (x_i[0] + 10, x_i[1])$ 
26  $x_i[1] = (x_i[1] + 10, x_i[0])$ 
27  $x_i[2] = (x_i[2] + 10, x_i[1])$ 
28  $x_i[3] = (x_i[3] + 10, x_i[0])$ 
29  $x_i[4] = (x_i[4] + 10, x_i[1])$ 
30
31 # Initial cluster centers
32  $c_i = np.random.randn(4, 2)$ 
33
34 iterations = 0
35 while True:
36     iterations += 1
37     # All distances give us labels
38      $d_i = np.sqrt([ (x_i[0] - c_i[0])**2 + (x_i[1] - c_i[1])**2 ])$ 
39     labels = d_i.argmax(axis = 1)
40
41     # Update  $c_i$  with the minimization algorithm
42      $c_i = np.zeros([k, 2])$ 
43     for i in range(k):
44         if np.allclose(c_i[i], c_i[i-1]) or iterations > 100: break
45          $c_i[i] = c_i[i].copy()$ 
46
47     fig = plt.scatter(x = x_i[:, 0], y = x_i[:, 1], color = labels.astype('str'))
48     fig.set_xlabel('x = c_i[0, 0]')
49     fig.set_ylabel('y = c_i[0, 1]')
50     fig.set_title('k-means')
51     fig.set_xlim(-10, 10)
52     fig.set_ylim(-10, 10)
53     fig.savefig('clusters.png')
54     plt.close('all')
55
56 # Compile
57
58 # H♥rtDown Editor
59
60 # H♥rtDown Editor
61
62 # H♥rtDown Editor
63
64 # H♥rtDown Editor
65
66 # H♥rtDown Editor
67
68 # H♥rtDown Editor
69
70 # H♥rtDown Editor
71
72 # H♥rtDown Editor
73
74 # H♥rtDown Editor
75
76 # H♥rtDown Editor
77
78 # H♥rtDown Editor
79
80 # H♥rtDown Editor
81
82 # H♥rtDown Editor
83
84 # H♥rtDown Editor
85
86 # H♥rtDown Editor
87
88 # H♥rtDown Editor
89
90 # H♥rtDown Editor
91
92 # H♥rtDown Editor
93
94 # H♥rtDown Editor
95
96 # H♥rtDown Editor
97
98 # H♥rtDown Editor
99
100 # H♥rtDown Editor
```

(*)Here's an example use of H♥rtDown as an experimenter. This example describes a clustering algorithm.

We generate the output by compiling the source code.

The paper uses the L1 norm to calculate distances. The user changes the math to use the more common L2 norm.

H♥rtDown updates both the typeset math and the figure that relies on the generated code library.

The new cluster centers have changed and are now heavily influenced by the outliers.

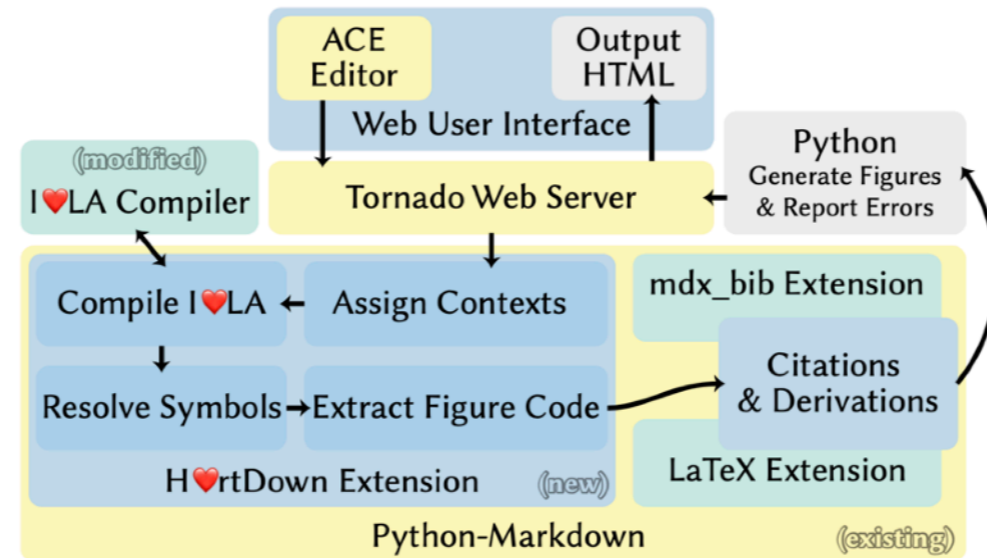
We can also click the figure to update the figure code, in this case, H♥rtDown will only rerun the figure code.

###

The generated code libraries are saved into files and can be used outside of the H♥rtDown reading/authoring environment.

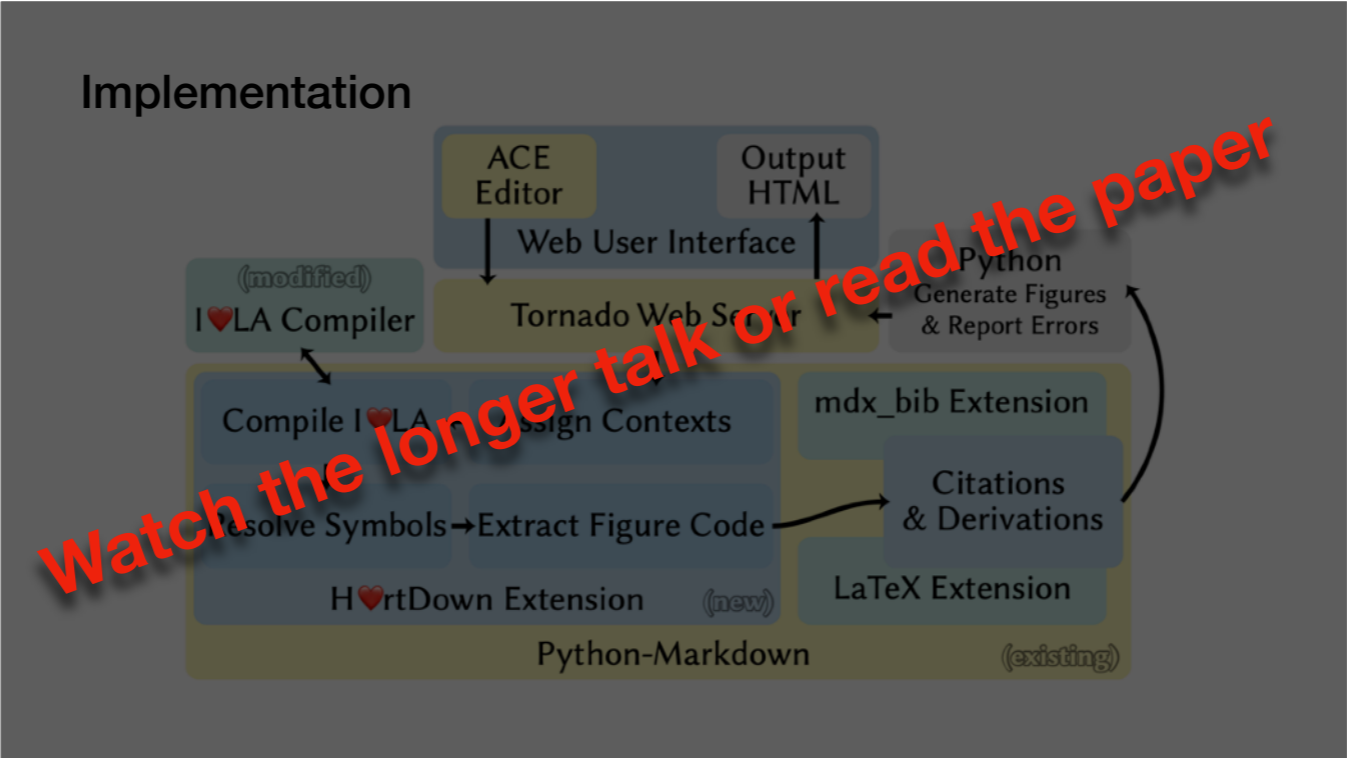
We'll show examples of that in our case studies.

Implementation



To learn about H♥rtDown's implementation,

% This figure shows the overall structure for H♥rtDown's implementation.



please watch Yong's longer talk or read the paper.

H♥rtDown Case Studies

Entire papers

- An Omnistereoscopic Video Pipeline for Capture and Display of Real-World VR
- A Luminance-aware Model of Judder Perception (*)
- A Perceptual Model for Eccentricity-dependent Spatio-temporal Flicker Fusion and its Applications to Foveated Graphics
- A Symmetric Objective Function for ICP (*)
- Regularized Kelvinlets Sculpting Brushes based on Fundamental Solutions of Elasticity (*)

Paper sections

- Stable Neo-Hookean Flesh Simulation (*)
- A perceptual model of motion quality for rendering with adaptive refresh-rate and resolution
- Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation (*)
- On Elastic Geodesic Grids and Their Planar to Spatial Deployment
- Nautilus-Recovering Regional Symmetry Transformations for Image Editing
- Computational Design of Transforming Pop-up Books
- Unmixing-Based Soft Color Segmentation for Image Manipulation (*)
- Generic Objective Vortices for Flow Visualization (*)
- SIERE: a hybrid semi-implicit exponential integrator for efficiently simulating stiff deformable objects

(*) compares code to an existing implementation

We converted a variety of SIGGRAPH papers and paper sections to H♥rtDown as case studies.

#####

Our criteria for selecting papers were that they use linear algebra implementable by I♥LA

The papers are from the past five years (2017–2021) of SIGGRAPH and span geometry processing, image processing, visualization, simulation, and rendering.

It include 5 full papers and 9 papers for which we implemented single subsections

H♥rtDown Case Studies

A Symmetric Objective Function for ICP

Szymon Rusinkiewicz

SIGGRAPH North America 2019

- H♥rtDown source (entire paper)
- H♥rtDown-generated code libraries
- Existing implementation source code before modification and modified to call H♥rtDown-generated code

Original Paper [PDF]

THE PROPOSED METHOD FOR CONVERTING AN OBJECTIVE FUNCTION INVOLVING ROTATIONS INTO A LINEAR FORM, WHICH THEN YIELDS A LINEAR LEAST SQUARES SYSTEM.
We instead pursue a linearization that starts with the Rodrigues rotation formula for the effect of a rotation R on a vector v :
$$Rv = v \cos \theta + (a \times v) \sin \theta + a(a \cdot v)(1 - \cos \theta), \quad (7)$$
where a and θ are the axis and angle of rotation. We observe that the last term in (7) is quadratic in the incremental rotation angle θ , so we drop it to linearize:
$$Rv \approx v \cos \theta + (a \times v) \sin \theta = \cos \theta [v + (a \times v)], \quad (8)$$
where $\tilde{a} = a \tan \theta$. Substituting into (6),
$$\mathcal{E}_{\text{symm}} = \sum_i \cos^2 \theta [(p_i - q_i) - \tilde{a} \times (p_i - q_i) + \tilde{a} \times \tilde{a} (p_i - q_i)]^2 = \sum_i \cos^2 \theta [(p_i - q_i) - \tilde{a} \times (p_i - q_i)]^2, \quad (9)$$
where $n_i = n_{p,i} + n_{q,i}$ and $\tilde{a} = i/\cos \theta$. We now make the additional approximation of weighting the objective by $1/\cos^2 \theta$, which approaches 1 for small θ . Finally, for better numerical stability, we

H♥rtDown Paper Viewer

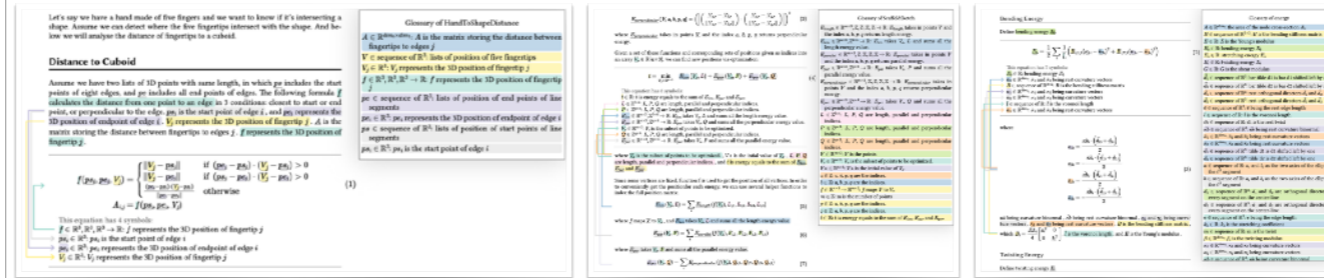
The proposed method for converting an objective function involving rotations into a linear form, which then yields a linear least squares system, is not optimal. In fact, the proposed method is only a linear approximation of the true objective function. We instead pursue a linearization that starts with the Rodrigues rotation formula for the effect of a rotation R on a vector v :
$$Rv = v \cos \theta + (a \times v) \sin \theta + a(a \cdot v)(1 - \cos \theta), \quad (7)$$
where a and θ are the axis and angle of rotation. We observe that the last term in (7) is quadratic in the incremental rotation angle θ , so we drop it to linearize:
$$Rv \approx v \cos \theta + (a \times v) \sin \theta = \cos \theta [v + (a \times v)], \quad (8)$$
where $\tilde{a} = a \tan \theta$. Substituting into (6),
$$\mathcal{E}_{\text{symm}} = \sum_i \cos^2 \theta [(p_i - q_i) - \tilde{a} \times (p_i - q_i) + \tilde{a} \times \tilde{a} (p_i - q_i)]^2 = \sum_i \cos^2 \theta [(p_i - q_i) - \tilde{a} \times (p_i - q_i)]^2, \quad (9)$$
where $n_i = n_{p,i} + n_{q,i}$ and $\tilde{a} = i/\cos \theta$. We now make the additional approximation of weighting the objective by $1/\cos^2 \theta$, which approaches 1 for small θ . Finally, for better numerical stability, we

Each case study includes the H♥rtDown source file, H♥rtDown's generated paper reading environment, and H♥rtDown's generated code library for C++, Python and MATLAB.

We also provide a link to the original paper for comparison and side-by-side screenshots.

Expert Study

- 3 CS PhD students
- Author an original document related to their computer graphics research



- Spent 24, 7, and 6 hours, respectively, using H♥rtDown over a period of two weeks

We recruited 3 computer science PhD students for an expert study.

In our experiment, participants were given initial and follow-up questionnaires to understand their current practices and share their thoughts about H♥rtDown.

They spent a total of 24, 7, and 6 hours, respectively, using H♥rtDown over a period of two weeks.

For the tasks in our expert study,

Expert Study: Observations and Conclusions

“H♥rtDown is an excellent tool to share tutorial[s] online—it highlights the vector dimension and variable meaning...following all the vectors/matrices/their dims is **the hardest part** of reproducing a paper.”

Bending Energy

Define bending energy E_b

$$E_b = \frac{1}{2} \sum_i \frac{1}{l_i} \left(\bar{B}_{1,1} (\epsilon_{2i} - \bar{R}_2)^2 + \bar{B}_{1,2} (\epsilon_{1i} - \bar{R}_1)^2 \right) \quad (2)$$

This equation has 7 symbols:

- $E_b \in \mathbb{R}$: bending energy E_b
- $\bar{R}_1 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being rest curvature vectors
- $\bar{B} \in \text{sequence of } \mathbb{R}^{2 \times 2}$: \bar{B} is the bending stiffness matrix
- $\bar{R}_1 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being curvature vectors
- $\bar{R}_2 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being curvature vectors
- $l \in \text{sequence of } \mathbb{R}$: l is the voronoi length
- $\bar{R}_1 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being rest curvature vectors

where:

$$\bar{R}_{1i} = \frac{n b_i \cdot (\bar{d}_{1i} + \bar{d}_{2i})}{2}$$

$$\bar{R}_{2i} = \frac{n b_i \cdot (\bar{d}_{1i} - \bar{d}_{2i})}{2}$$

$$\bar{R}_{1i} = \frac{n b_i \cdot (\bar{d}_{1i} + \bar{d}_{2i})}{2}$$

$$\bar{R}_{2i} = -\frac{n b_i \cdot (\bar{d}_{1i} - \bar{d}_{2i})}{2}$$

$n b$ being curvature binormal, $n \bar{d}$ being rest curvature binormal, \bar{R}_1 and \bar{R}_2 being curvature vectors, \bar{R}_1 and \bar{R}_2 being rest curvature vectors, \bar{B} is the bending stiffness matrix, which $\bar{B} = \frac{EA}{4} \begin{bmatrix} a^2 & 0 \\ 0 & b^2 \end{bmatrix}$, l is the voronoi length, and E is the Young's modulus.

Twisting Energy

Define twisting energy E_t

Glossary of energy

- $A \in \mathbb{R}^{\text{dim}}$: the area of the node cross-section A_i
- $\bar{B} \in \text{sequence of } \mathbb{R}^{2 \times 2}$: \bar{B} is the bending stiffness matrix
- $E \in \mathbb{R}$: E is the Young's modulus
- $E_b \in \mathbb{R}$: bending energy E_b
- $E_s \in \mathbb{R}$: stretching energy E_s
- $E_t \in \mathbb{R}$: twisting energy E_t
- $G \in \mathbb{R}$: G is the shear modulus
- $\bar{d}_1 \in \text{sequence of } \mathbb{R}^2$: bar \bar{d}_1 is bar d_1 shifted left by one
- $\bar{d}_2 \in \text{sequence of } \mathbb{R}^2$: bar \bar{d}_2 is bar d_2 shifted left by one
- $d_1 \in \text{sequence of } \mathbb{R}^2$: rest orthogonal directors d_1 and d_2
- $d_2 \in \text{sequence of } \mathbb{R}^2$: rest orthogonal directors d_1 and d_2
- $e \in \text{sequence of } \mathbb{R}$: e being the rest edge length
- $l \in \text{sequence of } \mathbb{R}$: l is the voronoi length
- $m \in \text{sequence of } \mathbb{R}$: m is the rest twist
- $n \bar{b} \in \text{sequence of } \mathbb{R}^2$: $n \bar{b}$ being rest curvature binormal
- $\bar{R}_1 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being rest curvature vectors
- $\bar{R}_2 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being rest curvature vectors
- $\bar{d}_1 \in \text{sequence of } \mathbb{R}^2$: \bar{d}_1 is d_1 shifted left by one
- $\bar{d}_2 \in \text{sequence of } \mathbb{R}^2$: \bar{d}_2 is d_2 shifted left by one
- $a, b \in \text{sequence of } \mathbb{R}$: a and b as the two axes of the ellipse at the i^{th} segment
- $b_1, b_2 \in \text{sequence of } \mathbb{R}$: b_1 and b_2 as the two axes of the ellipse at the i^{th} segment
- $d_1 \in \text{sequence of } \mathbb{R}^2$: d_1 and d_2 are orthogonal directors of every segment on the center-line
- $d_2 \in \text{sequence of } \mathbb{R}^2$: d_1 and d_2 are orthogonal directors of every segment on the center-line
- $e \in \text{sequence of } \mathbb{R}$: e being the edge length
- $k \in \mathbb{R}$: k is the stretching coefficient
- $m \in \text{sequence of } \mathbb{R}$: m is the twist
- $\bar{B} \in \mathbb{R}^{2 \times 2}$: \bar{B} is the bending stiffness matrix
- $\bar{R}_1 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being curvature vectors
- $\bar{R}_2 \in \mathbb{R}^{2 \times 2}$: \bar{R}_1 and \bar{R}_2 being curvature vectors
- $n \bar{b} \in \text{sequence of } \mathbb{R}^2$: $n \bar{b}$ being rest curvature binormal

(*) One expert wrote a tutorial for discrete elastic rods. This tutorial is now more readable due to math augmentations and can be used in addition to reading since it self-generates code in any programming languages I♥LA supports.

That expert said: “H♥rtDown is an excellent tool to share tutorial[s] online—it highlights the vector dimension and variable meaning...following all the vectors/matrices/their dims is the hardest part of reproducing a paper.”

Please see the longer talk or the paper for an in-depth discussion.

Limitations

- H♥rtDown does not consider pseudocode or algorithmic steps described in prose

```
Algorithm 1 A single simulation step of our proposed SPH-based snow solver.  
1: foreach particle  $i$  do  
2:   compute  $\rho_{i,h}$  ▷ see Subsection 3.3.2  
3:   compute  $L_i$  ▷ see Eq. (15)  
4:   compute  $\mathbf{a}_i^{\text{other},t}$  ▷ e.g., gravity and adhesion  
5:   compute  $\mathbf{a}_i^{\text{friction},t}$  ▷ using Eq. (24)  
6: SOLVE for  $\mathbf{a}_i^{\lambda}$  ▷ see Subsection 3.2.1  
7: SOLVE for  $\mathbf{a}_i^G$  ▷ see Subsection 3.2.2  
8: foreach particle  $i$  do  
9:   integrate  $\mathbf{v}_i^{t+\Delta t} = \mathbf{v}_i^t + \Delta t (\mathbf{a}_i^{\text{other},t} + \mathbf{a}_i^{\text{friction},t} + \mathbf{a}_i^{\lambda} + \mathbf{a}_i^G)$   
10: foreach particle  $i$  do  
11:   integrate  $\mathbf{F}_{E,i}$  ▷ see Subsection 3.3.1  
12: foreach particle  $i$  do  
13:   integrate  $\mathbf{x}_i^{t+\Delta t} = \mathbf{x}_i^t + \Delta t \mathbf{v}_i^{t+\Delta t}$ 
```

[Gissler et al. 2020]

- The space of executable math and potential application domains for H♥rtDown is much broader than linear algebra

One limitation of H♥rtDown is that it does not consider pseudocode, literate programming, or algorithmic steps described in prose. Algorithms are often needed to make formulas useful.

Another limitation stems from the kinds of formulas that our extended version of I♥LA can handle.

Future Work

- Automatic conversion from LaTeX to H♥rtDown
- A proof checker to verify derivations
- Callbacks and delegates for expanding the abilities of the generated code
- Support for active reading (e.g. annotating and comparing)

There are a lot of directions we'd like to explore in the future.

Automatic or semi-automatic conversion from LaTeX to H♥rtDown

Incorporating a proof checker to verify derivations

Explore callbacks and delegates for expanding the abilities of the generated code

Improve our reading environment to support active reading activities such as annotating and comparing

Conclusions

- H♥rtDown is a low-overhead, ecologically compatible document processor
- H♥rtDown supports authors and improves replicability, readability, and experimentation
- Participants in our expert study found uses for H♥rtDown in their research practice.

In conclusion,

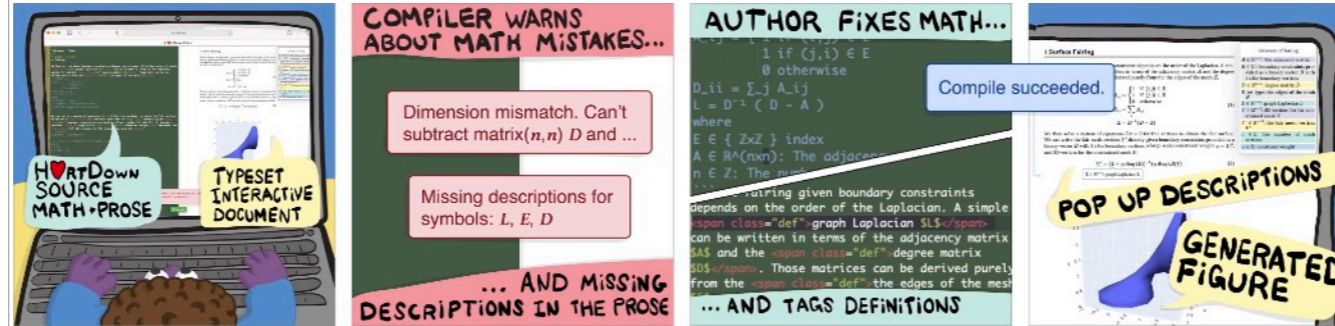
H♥rtDown is a low-overhead, ecologically compatible document processor

H♥rtDown supports authors and improves replicability, readability, and experimentation

Participants In our expert study found uses for H♥rtDown in their research practice.

H♥rtDown

<https://iheartla.github.io/heartdown/>



Acknowledgements: Anonymous reviewers, Seth Walker, Zoya Bylinskii, Zhecheng Wang, Xue Yu, Jialin Huang
Sponsors: Canada Research Chairs Program, Sloan Foundation, Adobe Inc.

H♥rtDown can be used at all stages of research
(from experimenting with the seed of an idea, to writing the final paper)

Thanks for listening!

Please try our language.

You are welcome to contact us in the future.