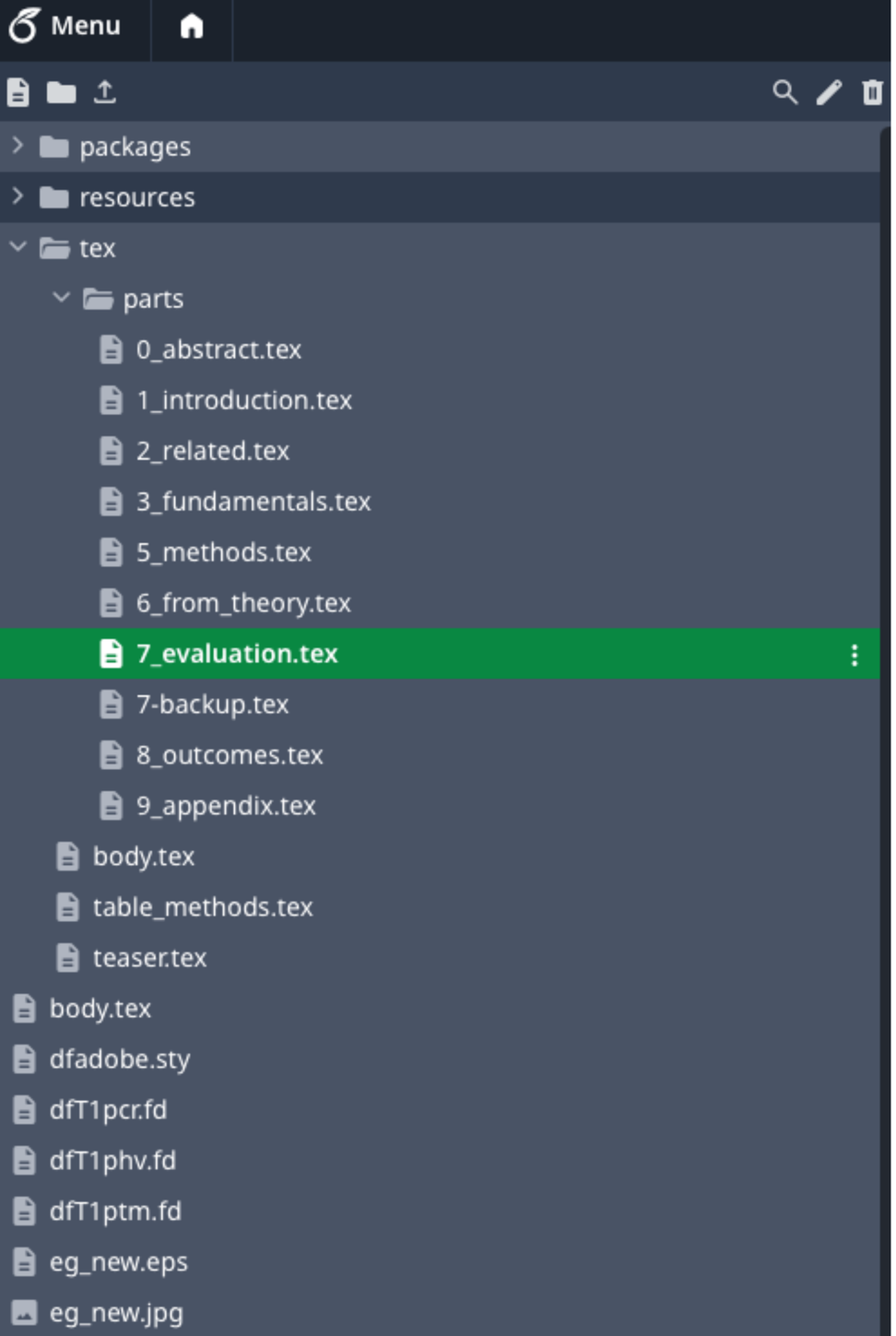
# *Artworks* des articles de recherche

# ... and beyond

David Coeurjolly, Vincent Nivoliers

# Disclaimer


LOUDER CROWDER
LOUDER CROWDER
CHANGE MY MIND


Menu
packages
resources
tex
parts
0_abstract.tex
1_introduction.tex
2_related.tex
3_fundamentals.tex
5_methods.tex
6_from_theory.tex
7_evaluation.tex
7-backup.tex
8_outcomes.tex
9_appendix.tex
body.tex
table_methods.tex
teaser.tex
body.tex
dfadobe.sty
dfT1pcr.fd
dfT1phv.fd
dfT1ptm.fd
eg_new.eps
eg_new.jpg

Go to hell!

# Préambule

# Surface Multigrid via Intrinsic Prolongation

FELIX TYDEMAN LIU, University of Toronto, Canada
JUSTIN DAI ZHANG, University of Toronto, Canada
MIRELA BEN-CHEN, Technion – Israel Institute of Technology, Israel
ALEC JACOBSON, University of Toronto, Canada



Fig. 1. We introduce a novel geometric multigrid solver for surface meshes. Our key ingredient is an intrinsic prolongation operator computed via parameterizing the high-resolution shape on its coarsened counterpart, visualized using colored triangles. By recursively applying this self-parameterization, we obtain a hierarchy of nested surfaces. Our method can be used to solve a variety of linear systems (Cat mesh © 2012 Jay Hill, licensed by Beyond Range under CC BY NC).

This paper introduces a novel geometric multigrid solver for unstructured curved surfaces. Multigrid methods are highly efficient iterative methods for solving linear systems arising from discretized PDEs. However, they have been defined on structured domains, preventing multigrid in unstructured curved domains remains a challenging problem. The critical missing ingredient is a prolongation operator that relates a surface to its coarsened, multigrid levels. We propose a novel method for computing the prolongation: the triangulated surface-based on intrinsic geometry, enabling an efficient and accurate construction of a hierarchy of nested surfaces. Our solver then achieves better convergence than existing multigrid methods. Compared to the standard remeshing approach, our intrinsic prolongation is simpler, more memory-efficient, and faster to compute. We demonstrate our intrinsic multigrid method on many geometry processing applications and a wide variety of triangle meshes, including non-manifold inputs. Despite lacking a formal analysis, our upgrade existing algorithms to intrinsic Farin's state, and yield the computational bottleneck away from solving linear systems.

CCS Concepts: • **Computing methodologies** → **Mesh models** • **Applied computing**.

Additional Key Words and Phrases: computer graphics, geometric multigrid

**ACM Reference Format:**
Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface Multigrid via Intrinsic Prolongation. *ACM Trans. Graph.* 1, 1, Article 88 (August 2021), 13 pages. https://doi.org/10.1145/3450626.3459769

## 1 INTRODUCTION

Linear solvers are the heart of many geometry processing algorithms. For positive (semi-)definite problems defined on surface meshes,

Authors' addresses: Hsueh-Ti Derek Liu, University of Toronto, Canada, hsuehtil@cs.toronto.edu; Jiayi Eris Zhang, University of Toronto, Canada, eris.zhang@mail.utoronto.ca; Mirela Ben-Chen, Technion – Israel Institute of Technology, Israel, mirela@cs.technion.ac.il; Alec Jacobson, University of Toronto, Canada, jacobson@cs.toronto.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2021 Copyright held by the owner/author(s).
0730-0301/2021/8-ART88
https://doi.org/10.1145/3450626.3459769




Fig. 2. Many geometry processing algorithms that involve solving linear systems can benefit from multigrid methods. Instead of building a hierarchy during the full-stack solve, (1) building the right hierarchy and (2) solving the resulting $\mathbf{A}\mathbf{x}=\mathbf{b}$ have relatively low, scheduled, pre-computation cost compared to the solve. Our method is the first to enable this for surface meshes. Currently, multigrid methods perform pre-computation solely based on $\mathbf{A}$ (dashed line). Our intrinsic prolongation, which is key to efficient multigrid solves, can still leverage the same pre-computed hierarchy to solve the system efficiently.

direct solvers (e.g., Cholesky factorization) are commonplace. Unfortunately, direct solvers are inefficient in terms of memory and runtime, and for problems on high-resolution surface meshes. Especially for applications where the linear system changes at every iteration (e.g., as a result of a nonlinear deformation), a faster linear solver is crucial.

For problems on structured domains (e.g., 2D/3D regular grids), an excellent alternative is *geometric multigrid methods*. Geometric multigrid methods construct a hierarchy of coarser grids (i.e., geometry) without knowing the linear system of interest (see Fig. 2). This enables multigrid methods to solve the system efficiently in linear time. Unfortunately, the construction of a hierarchy on irregular surface already becomes non-trivial for unstructured grids (e.g., arbitrary triangle meshes in 2D) or tetrahedral meshes in 3D), the lack of a hierarchy prevents multigrid methods from being a "black-box" multigrid solver for curved surfaces (Farin 2018).

In this paper, we propose a Galerkin geometric multigrid solver for unstructured curved surfaces. Our key ingredient is a method for computing the prolongation operator based on the intrinsic geometry. Our multigrid solver achieves a better convergence rate than existing multigrid methods. Moreover, by replacing direct solvers with our black-box surface multigrid solver, it leads to orders of magnitude speed-up. We draw our method is the first to enable multigrid methods for many geometry processing shell simulation, with linear systems of different sparsity patterns and density. Our contributions turn existing algorithms into interactive ones (see Fig. 2) and shift the bottleneck away from solving linear systems (e.g., Fig. 15).

## 2 RELATED WORKS

*Multigrid methods.* Brandt [1977] have exerted a reputation as one of the fastest numerical solvers for solving linear systems. On structured domains (e.g., regular grids), multigrid methods have been shown theoretically [Hackbusch 2013; Trottenberg et al. 2000] and practically [Brandt and Livne 2011]. In graphics, multigrid has been an attractive solution for many problems such as large-scale particle simulation, fluid simulation, and light transport [Ament et al. 2014; Dick and Hugger 2008; Krishnan and Ascher 2011] and simulating fluids on large grids [Ando et al. 2015; Liu et al. 2019; Setaluri et al. 2014]. Even for problems where the original representation is not a regular grid, multigrid methods can be applied by resorting to a multigrid-in-python efficient computation and transfer the solution back to the unstructured representation. For example, one can first voxelize the input shape [Huang et al. 2013] or construct a cage for interpolation [Dick et al. 2011; Zhu et al. 2018] and character skinning [McAdams et al. 2011]. Chuang et al. [2009] deploy multigrid on a 2D parameter domain to solve the Poisson equation defined on surface mesh. To reduce the complexity of using structured representations, adaptive multigrid methods are developed for volumetric meshes [Grinspun et al. 2006] and surface meshes [Grinspun et al. 2004], and other graphics applications [Vaxman and Ben-Chen 2010].

*Unstructured Euclidean Domains.* Directly deploying multigrid to unstructured "grids" in Euclidean domains (e.g., 2D triangle meshes and 3D tetrahedral meshes) has also been an important problem in graphics. Ruge and Stüben [1987] first introduce the concept of grid hierarchy and how to transfer signals back and forth across different grid levels. In graphics, unstructured multigrid for 2D triangle meshes [Kraevoy et al. 2007] demonstrates that a nested set of coarse patches is prescribed by a 2D boundary curve. In the methods proposed in [Jeon et al. 2015; Oh et al. 2008; Wang et al. 2018], they build a hierarchy of nested tetrahedral meshes for the input shape in the 3D shape pattern and then successively subdividing it to get finer resolutions. Wang et al. [2018] generate the multigrid hierarchy by applying the fast tetrahedral mesh simplification [Hu et al. 2019] to triangulating the 2D domain. When it comes to 3D tetrahedral meshes, multigrid is commonly used to emulate deformable objects undergoing elastic forces [Otaduy et al. 2009]. Although these multigrid solvers already become non-trivial for unstructured grids (e.g., arbitrary triangle meshes in 2D) or tetrahedral meshes in 3D), the lack of a hierarchy prevents multigrid methods from being a "black-box" multigrid solver for curved surfaces (Farin 2018).

In this paper, we propose a Galerkin geometric multigrid solver for unstructured curved surfaces. Our key ingredient is a method for computing the prolongation operator based on the intrinsic geometry. Our multigrid solver achieves a better convergence rate than existing multigrid methods. Moreover, by replacing direct solvers with our black-box surface multigrid solver, it leads to orders of magnitude speed-up. We draw our method is the first to enable multigrid methods for many geometry processing shell simulation, with linear systems of different sparsity patterns and density. Our contributions turn existing algorithms into interactive ones (see Fig. 2) and shift the bottleneck away from solving linear systems (e.g., Fig. 15).

## 2 RELATED WORKS

*Multigrid methods.* Brandt [1977] have exerted a reputation as one of the fastest numerical solvers for solving linear systems. On structured domains (e.g., regular grids), multigrid methods have been shown theoretically [Hackbusch 2013; Trottenberg et al. 2000] and practically [Brandt and Livne 2011]. In graphics, multigrid has been an attractive solution for many problems such as large-scale particle simulation, fluid simulation, and light transport [Ament et al. 2014; Dick and Hugger 2008; Krishnan and Ascher 2011] and simulating fluids on large grids [Ando et al. 2015; Liu et al. 2019; Setaluri et al. 2014]. Even for problems where the original representation is not a regular grid, multigrid methods can be applied by resorting to a multigrid-in-python efficient computation and transfer the solution back to the unstructured representation. For example, one can first voxelize the input shape [Huang et al. 2013] or construct a cage for interpolation [Dick et al. 2011; Zhu et al. 2018] and character skinning [McAdams et al. 2011]. Chuang et al. [2009] deploy multigrid on a 2D parameter domain to solve the Poisson equation defined on surface mesh. To reduce the complexity of using structured representations, adaptive multigrid methods are developed for volumetric meshes [Grinspun et al. 2006] and surface meshes [Grinspun et al. 2004], and other graphics applications [Vaxman and Ben-Chen 2010].

*Unstructured Euclidean Domains.* Directly deploying multigrid to unstructured "grids" in Euclidean domains (e.g., 2D triangle meshes and 3D tetrahedral meshes) has also been an important problem in graphics. Ruge and Stüben [1987] first introduce the concept of grid hierarchy and how to transfer signals back and forth across different grid levels. In graphics, unstructured multigrid for 2D triangle meshes [Kraevoy et al. 2007] demonstrates that a nested set of coarse patches is prescribed by a 2D boundary curve. In the methods proposed in [Jeon et al. 2015; Oh et al. 2008; Wang et al. 2018], they build a hierarchy of nested tetrahedral meshes for the input shape in the 3D shape pattern and then successively subdividing it to get finer resolutions. Wang et al. [2018] generate the multigrid hierarchy by applying the fast tetrahedral mesh simplification [Hu et al. 2019] to triangulating the 2D domain. When it comes to 3D tetrahedral meshes, multigrid is commonly used to emulate deformable objects undergoing elastic forces [Otaduy et al. 2009]. Although these multigrid solvers already become non-trivial for unstructured grids (e.g., arbitrary triangle meshes in 2D) or tetrahedral meshes in 3D), the lack of a hierarchy prevents multigrid methods from being a "black-box" multigrid solver for curved surfaces (Farin 2018).

In this paper, we propose a Galerkin geometric multigrid solver for unstructured curved surfaces. Our key ingredient is a method for computing the prolongation operator based on the intrinsic geometry. Our multigrid solver achieves a better convergence rate than existing multigrid methods. Moreover, by replacing direct solvers with our black-box surface multigrid solver, it leads to orders of magnitude speed-up. We draw our method is the first to enable multigrid methods for many geometry processing shell simulation, with linear systems of different sparsity patterns and density. Our contributions turn existing algorithms into interactive ones (see Fig. 2) and shift the bottleneck away from solving linear systems (e.g., Fig. 15).





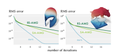



Fig. 3. When solving a Poisson problem defined on the surface mesh (right), we demonstrate that our multigrid based on the intrinsic prolongation (blue) converges faster than the algebraic multigrid methods (green), and the Conjugate Gradient (red). Our multigrid solver is built on top of the Geometry Processing Algorithms Library (GAIA) [Zhang et al. 2020]. Note that we use an off-the-shelf implementation from PyAMG [Olson and Bell 2012] for the algebraic multigrid hyperparameters. Visualized by [ZHP-CAJ]-cow (© BF SA).




Fig. 4. Compared to the prolongation based on the vertex 1-ring average (green), the Laplacian coordinates (blue) can lead to faster convergence rate when solving a Poisson problem on the surface (top right).

coordinates on triangles or tetrahedra. Recently, Xiao et al. [2019] propose a "mesh-free" alternative for tetrahedral meshes. They precompute the prolongation operator by solving a local Poisson problem and define the prolongation using piecewise constant interpolation, which only requires the closest point query.

*Algebraic Multigrid.* A popular alternative to deal with unstructured meshes is to use *algebraic multigrid* [Brandt et al. 1982; Ruge and Stüben 1987] which only requires the linear system matrix as a weighted graph and coarsening it. This approach makes no assumptions on the structure of the geometry. Thus, it is directly applicable to unstructured meshes in any dimension. For example, the 2D shape pattern and then successively subdividing it to get finer resolutions. Wang et al. [2018] generate the multigrid hierarchy by applying the fast tetrahedral mesh simplification [Hu et al. 2019] to triangulating the 2D domain. When it comes to 3D tetrahedral meshes, multigrid is commonly used to emulate deformable objects undergoing elastic forces [Otaduy et al. 2009]. Although these multigrid solvers already become non-trivial for unstructured grids (e.g., arbitrary triangle meshes in 2D) or tetrahedral meshes in 3D), the lack of a hierarchy prevents multigrid methods from being a "black-box" multigrid solver for curved surfaces (Farin 2018).

In this paper, we propose a Galerkin geometric multigrid solver for unstructured curved surfaces. Our key ingredient is a method for computing the prolongation operator based on the intrinsic geometry. Our multigrid solver achieves a better convergence rate than existing multigrid methods. Moreover, by replacing direct solvers with our black-box surface multigrid solver, it leads to orders of magnitude speed-up. We draw our method is the first to enable multigrid methods for many geometry processing shell simulation, with linear systems of different sparsity patterns and density. Our contributions turn existing algorithms into interactive ones (see Fig. 2) and shift the bottleneck away from solving linear systems (e.g., Fig. 15).

Jonathan D. Cohen, Dinesh Manocha, and Marc Olano. 2003. Simplifying polygonal models using successive mappings. In *Proceedings of IEEE Visualization 2003.* IEEE Computer Society, Seattle, WA, USA, 395–402. https://doi.org/10.1109/VISUAL.2003.1250400

Tim Davis, Yifan Hu, Sivasankaran Rajamanickam, and Timothy A. Davis. 2017. The Sparse Matrix (Wolfe) Conjugate Gradient Algorithm. *ACM Trans. Graph.* 36, 3, Article 29 (May 2017), 16 pages. https://doi.org/10.1145/2998441

Fernando de Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. On-the-fly Adaptive Surface Isocontouring with Dual Contouring. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games (I3D '15).* Association for Computing Machinery, New York, NY, USA, 211–217. https://doi.org/10.1145/2699276.2699290

Christian Dick, Joachim Georgii, and Rüdiger Westermann. 2011. A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simul. Model. Pract. Theory* 19, 2 (2011), 801–816.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: a practical gauss-seidel method for stable soft bodies. *ACM Trans. Graph.* 35, 6 (2016), 214:1–214:9.

Gerald Farin, Dianne Hansford, and Gabriel Hajduczky. 2004. Translation-invariant matrices. *ACM Trans. Graph.* 23 (2004), 1063–1063.

Gerald E. Farin. 2018. *Curves and surfaces for CAGD: a practical guide* (sixth edition ed.). The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, San Francisco, CA. https://www.sciencedirect.com/science/article/pii/B9780080515588500000

Yotam Gingold, Takeo Igarashi, and Denis Zorin. 2009. Structured annotations for 2D-to-3D modeling. *ACM Trans. Graph.* 28, 5 (2009), 148:1–148:9. http://dblp.uni-trier.de/db/journals/tog/tog28.html#GingoldIZ09

Eitan Grinspun, Mathieu Desbrun, Konrad Polthier, Peter Schröder, Ari Stern, and Yiying Tong. 2006. Discrete multigrid complexes and their application to FAS and exterior calculus. *Computing* 76 (2006), 347–363.

Eitan Grinspun, Petr Krysl, and Peter Schröder. 2004. CHARMS: a simple framework for adaptive simulation. *ACM Trans. Graph.* 21, 3 (2004), 281–290.

Wolfgang Hackbusch. 2013. *Multi-grid methods and applications.* Vol. 4. Springer Science & Business Media.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant field-aligned meshes. *ACM Trans. Graph.* 34, 6 (2015), 189:1–189:15. http://dblp.uni-trier.de/db/journals/tog/tog34.html#JakobTPS15

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM Trans. Graph.* 37, 4 (2018), 60:1–60:14. http://dblp.uni-trier.de/db/journals/tog/tog37.html#HuZGJZP18

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.* 39, 4 (2020), 117:1–117:18. http://dblp.uni-trier.de/db/journals/tog/tog39.html#HuSWZP20

Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2019. TetWild: Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 38, 4 (2019), 60:1–60:15. http://dblp.uni-trier.de/db/journals/tog/tog38.html#HuSWZP19

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph.* 26, 3 (2007), 71–es. http://dblp.uni-trier.de/db/journals/tog/tog26.html#JoshiMDGS07

Ivaylo Kaloyanov Kaloyanov. 2015. Anisotropic shape control for Catmull-Clark subdivision surfaces. In *Proceedings of the 31st Spring Conference on Computer Graphics (SCCG '15).* Association for Computing Machinery, New York, NY, USA, 61–68. https://doi.org/10.1145/2788539.2788547

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Trans. Graph.* 32, 3 (2013), 29:1–29:13. http://dblp.uni-trier.de/db/journals/tog/tog32.html#KazhdanH13

Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. 2012. Can mean-curvature flow be modified to be non-singular? *Comput. Graph. Forum* 31, 5 (2012), 1745–1754. http://dblp.uni-trier.de/db/journals/cgf/cgf31.html#KazhdanSB12

Vladislav Kraevoy, Alla Sheffer, and Craig Gotsman. 2003. Matchmaker: constructing constrained texture maps. *ACM Trans. Graph.* 22, 3 (2003), 326–333. http://dblp.uni-trier.de/db/journals/tog/tog22.html#KraevoySG03

Vladislav Kraevoy, Dan Julius, and Alla Sheffer. 2007. Shape Correspondence for Emerging Applications. *Computer Graphics Forum* 26, 1 (2007), 28–44. https://doi.org/10.1111/j.1467-8659.2006.00946.x

Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. 2008. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graph.* 27, 5 (2008), 166:1–166:8. http://dblp.uni-trier.de/db/journals/tog/tog27.html#NarainSSCL08

Yutaka Ohtake, Alexander G. Belyaev, and Hans-Peter Seidel. 2003. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.* 23, 3 (2003), 609–612. http://dblp.uni-trier.de/db/journals/tog/tog23.html#OhtakeBS03

Yutaka Ohtake, Alexander G. Belyaev, and Hans-Peter Seidel. 2004. Ridge-valley lines on meshes via implicit surface fitting. In *ACM SIGGRAPH 2004 Sketches (SIGGRAPH '04).* Association for Computing Machinery, New York, NY, USA, 109. https://doi.org/10.1145/1186223.1186360

Yutaka Ohtake, Alexander G. Belyaev, and Hans-Peter Seidel. 2005. Ridge-valley lines on meshes via implicit surface fitting and regularization. In *ACM SIGGRAPH 2005 Sketches (SIGGRAPH '05).* Association for Computing Machinery, New York, NY, USA, 91–es. https://doi.org/10.1145/1187112.1187218

Eric T. Ogawa, Yun (Raymond) Hu, Haining Li, Emily Whiting, Justin Solomon, Keenan Crane, and Mina Konaković Luković. 2021. Cubic Barycentric Coordinates and Applications. *Computer Graphics Forum* 40, 2 (2021), 223–238. https://doi.org/10.1111/cgf.142628

Eric T. Ogawa, Yun (Raymond) Hu, Haining Li, Emily Whiting, Justin Solomon, Keenan Crane, and Mina Konaković Luković. 2021. Cubic Barycentric Coordinates and Applications. *Computer Graphics Forum* 40, 2 (2021), 223–238. https://doi.org/10.1111/cgf.142628

Miguel A. Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus H. Gross. 2009. Implicit contact handling for deformable objects. *Comput. Graph. Forum* 28, 2 (2009), 559–568. http://dblp.uni-trier.de/db/journals/cgf/cgf28.html#OtaduyTSG09

Eric Parker and Victor Ostromoukhov. 2011. Bump mapping for geometric primitives. In *Proceedings of Graphics Interface 2011 (GI '11).* Canadian Human-Computer Communications Society, Canada, 193–197.

Ulrich Pinkall and Konrad Polthier. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics* 2, 1 (1993), 15–36. https://projecteuclid.org:443/euclid.em/1062620720

Pierre-Olivier Rondeau, Yang Zhang, Ty Trusty, Emmett, and Kevin J. Maher. 2008. A real-time Multigrid Approach to Fluid Simulation Games. *Comput. Graph. Forum* 27 (2008), 1–10.

John W. Ruge and Klaus Stüben. 1987. Algebraic multigrid. In *Multigrid methods.* Frontiers in Applied Mathematics, Vol. 3. SIAM, 73–130. https://doi.org/10.1137/1.9781611971057.ch4

Teseo Schneider, Yixin Hu, Jérémie Dumas, Xifeng Gao, Daniele Panozzo, and Denis Zorin. 2018. Decoupling simulation accuracy from mesh quality. *ACM Trans. Graph.* 37, 6 (2018), 280:1–280:14. http://dblp.uni-trier.de/db/journals/tog/tog37.html#SchneiderHDGPZ18

Teseo Schneider, Yixin Hu, Xifeng Gao, Jérémie Dumas, Denis Zorin, and Daniele Panozzo. 2019. Decoupling simulation accuracy from mesh quality. *ACM Trans. Graph.* 38, 6 (2019), 280:1–280:14. http://dblp.uni-trier.de/db/journals/tog/tog38.html#SchneiderHDGPZ19

Rahul Sheth, Yotam Gingold, and Min Chen. 2011. Beyond pixels: exploring new representations and applications for motion analysis. *J. Vis. Comput. Graph.* 17, 5 (2011), 629–642. http://dblp.uni-trier.de/db/journals/tvcg/tvcg17.html#ShethGC11

Rahul Sheth, Yotam Gingold, and Min Chen. 2011. Beyond pixels: exploring new representations and applications for motion analysis. *J. Vis. Comput. Graph.* 17, 5 (2011), 629–642. http://dblp.uni-trier.de/db/journals/tvcg/tvcg17.html#ShethGC11

Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing (SGP '07).* Eurographics Association, Goslar, DEU, 109–116.

Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. 2004. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '04).* Association for Computing Machinery, New York, NY, USA, 175–184. https://doi.org/10.1145/1057432.1057456

Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. 2008. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graph.* 27, 5 (2008), 166:1–166:8. http://dblp.uni-trier.de/db/journals/tog/tog27.html#NarainSSCL08

Yutaka Ohtake, Alexander G. Belyaev, and Hans-Peter Seidel. 2003. Ridge-valley lines on meshes via implicit surface fitting and regularization. In *ACM SIGGRAPH 2005 Sketches (SIGGRAPH '05).* Association for Computing Machinery, New York, NY, USA, 91–es. https://doi.org/10.1145/1187112.1187218

Eric T. Ogawa, Yun (Raymond) Hu, Haining Li, Emily Whiting, Justin Solomon, Keenan Crane, and Mina Konaković Luković. 2021. Cubic Barycentric Coordinates and Applications. *Computer Graphics Forum* 40, 2 (2021), 223–238. https://doi.org/10.1111/cgf.142628

Eric T. Ogawa, Yun (Raymond) Hu, Haining Li, Emily Whiting, Justin Solomon, Keenan Crane, and Mina Konaković Luković. 2021. Cubic Barycentric Coordinates and Applications. *Computer Graphics Forum* 40, 2 (2021), 223–238. https://doi.org/10.1111/cgf.142628

Miguel A. Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus H. Gross. 2009. Implicit contact handling for deformable objects. *Comput. Graph. Forum* 28, 2 (2009), 559–568. http://dblp.uni-trier.de/db/journals/cgf/cgf28.html#OtaduyTSG09

Eric Parker and Victor Ostromoukhov. 2011. Bump mapping for geometric primitives. In *Proceedings of Graphics Interface 2011 (GI '11).* Canadian Human-Computer Communications Society, Canada, 193–197.

Ulrich Pinkall and Konrad Polthier. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics* 2, 1 (1993), 15–36. https://projecteuclid.org:443/euclid.em/1062620720

Pierre-Olivier Rondeau, Yang Zhang, Ty Trusty, Emmett, and Kevin J. Maher. 2008. A real-time Multigrid Approach to Fluid Simulation Games. *Comput. Graph. Forum* 27 (2008), 1–10.

John W. Ruge and Klaus Stüben. 1987. Algebraic multigrid. In *Multigrid methods.* Frontiers in Applied Mathematics, Vol. 3. SIAM, 73–130. https://doi.org/10.1137/1.9781611971057.ch4

Teseo Schneider, Yixin Hu, Jérémie Dumas, Xifeng Gao, Daniele Panozzo, and Denis Zorin. 2018. Decoupling simulation accuracy from mesh quality. *ACM Trans. Graph.* 37, 6 (2018), 280:1–280:14. http://dblp.uni-trier.de/db/journals/tog/tog37.html#SchneiderHDGPZ18

Teseo Schneider, Yixin Hu, Xifeng Gao, Jérémie Dumas, Denis Zorin, and Daniele Panozzo. 2019. Decoupling simulation accuracy from mesh quality. *ACM Trans. Graph.* 38, 6 (2019), 280:1–280:14. http://dblp.uni-trier.de/db/journals/tog/tog38.html#SchneiderHDGPZ19

Rahul Sheth, Yotam Gingold, and Min Chen. 2011. Beyond pixels: exploring new representations and applications for motion analysis. *J. Vis. Comput. Graph.* 17, 5 (2011), 629–642. http://dblp.uni-trier.de/db/journals/tvcg/tvcg17.html#ShethGC11

Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing (SGP '07).* Eurographics Association, Goslar, DEU, 109–116.

Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. 2004. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '04).* Association for Computing Machinery, New York, NY, USA, 175–184. https://doi.org/10.1145/1057432.1057456





Melina Skouras, Bernhard Thomaszewski, Bernd Bickel, and Markus H. Gross. 2012. Computational design of rubber balloons. *Comput. Graph. Forum* 31, 2 (2012), 835–844. http://dblp.uni-trier.de/db/journals/cgf/cgf31.html#SkourasTBG12

Jason Sewall, Dinesh Manocha, and Ming C. Lin. 2008. Interactive Rigid Body Dynamics Using Leading Eigenvector Dominance. In *Computer Graphics Forum.* Blackwell Publishing Ltd, 1–10. https://doi.org/10.1111/j.1467-8659.2008.01271.x

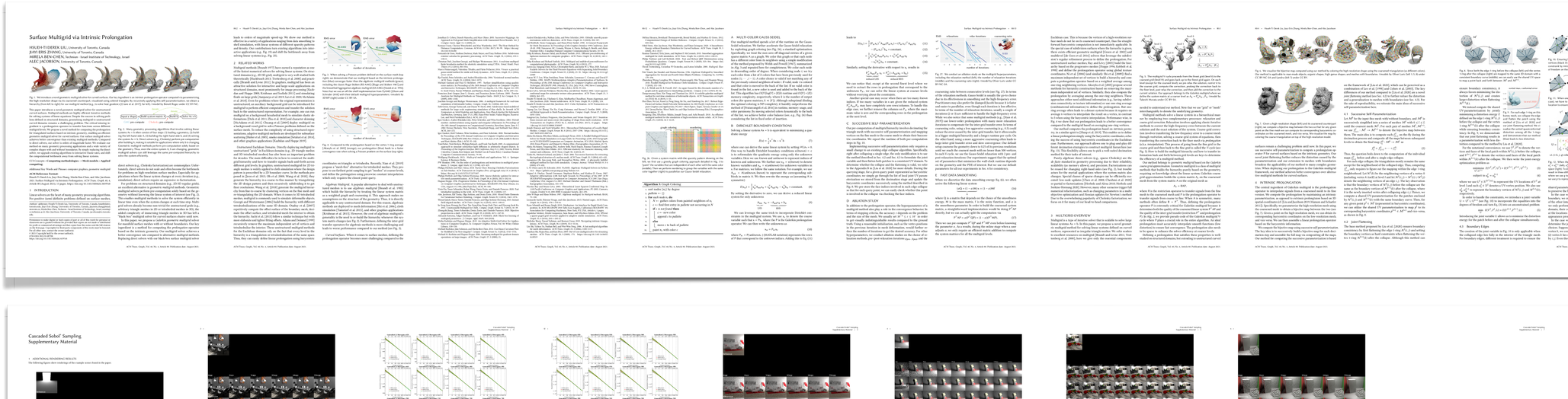
Rahul Narain, Jason Sewall, and Mark Carlson. 2008. Optimizing fluid simulation for multi-core architecture with duplicated data. *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2008* (2008), 23–32.

# Surface Multigrid via Intrinsic Prolongation

FELIX TYDEMAN LIU, University of Toronto, Canada
JOSÉ DIEGO JIMÉNEZ, Technion – Israel Institute of Technology, Israel
MIRELA BEN-CHEN, Technion – Israel Institute of Technology, Israel
ALEC JACOBSON, University of Toronto, Canada



Fig. 1. We introduce a novel geometric multigrid solver for coarsened surfaces. Our key ingredient is an intrinsic prolongation operator computed via parameterizing the high-resolution shape on its coarsened counterpart, visualized using colored triangles. By recursively applying this self-parameterization, we obtain a hierarchy of intrinsic Delaunay refinements (IDR) [Sharp et al. 2020] for the coarse meshes (3 cases of 4k [Liu et al. 2023], 1m [Jacobson et al. 2012], 1m [Jacobson et al. 2012], 1m [Jacobson et al. 2012



Surface Multigrid via Intrinsic Prolongation

FOLDEN FYODEREE LIU, University of Toronto, Canada
JOST ERIS ZHANG, University of Toronto, Canada
MIRELA BEN-CHEN, Technion – Israel Institute of Technology, Israel
ALEC JACOBSON, University of Toronto, Canada

1 INTRODUCTION

2 RELATED WORKS

3 PROLONGATION OPERATOR

4 MULTIGRID SOLVER

5 RESULTS

6 CONCLUSION

A PROOFS

B DIRICHLET BOUNDARY CONDITIONS

C SUCCESSIVE SELF-PARAMETERIZATION

D ABLATION STUDY

E FAST DATA SMOOTHING

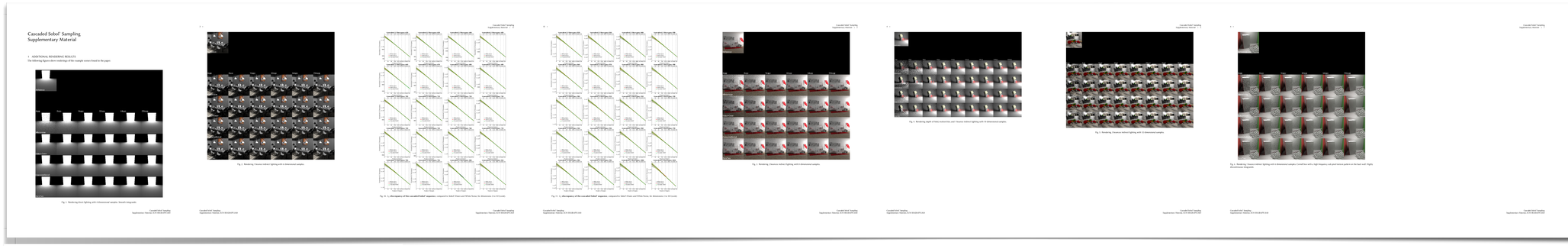
F MULTIGRID OVERVIEW

G INTRINSIC PROLONGATION

4.1 Successive Self-Parameterization

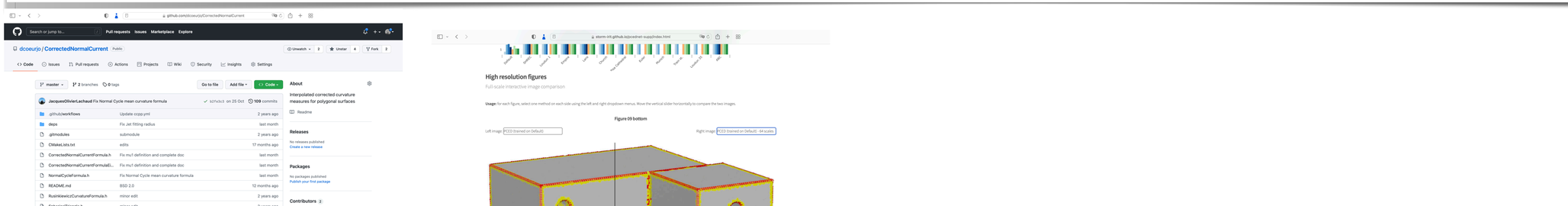
4.2 Joint Flattening

4.3 Boundary Edges



Cascaded Sobol' Sampling
Supplementary Material

1 ADDITIONAL RENDERING RESULTS



Cascaded Sobol' Sampling
Supplementary Material – 1



Cascaded Sobol' Sampling
Supplementary Material – 2



Cascaded Sobol' Sampling
Supplementary Material – 3



Cascaded Sobol' Sampling
Supplementary Material – 4



Cascaded Sobol' Sampling
Supplementary Material – 5



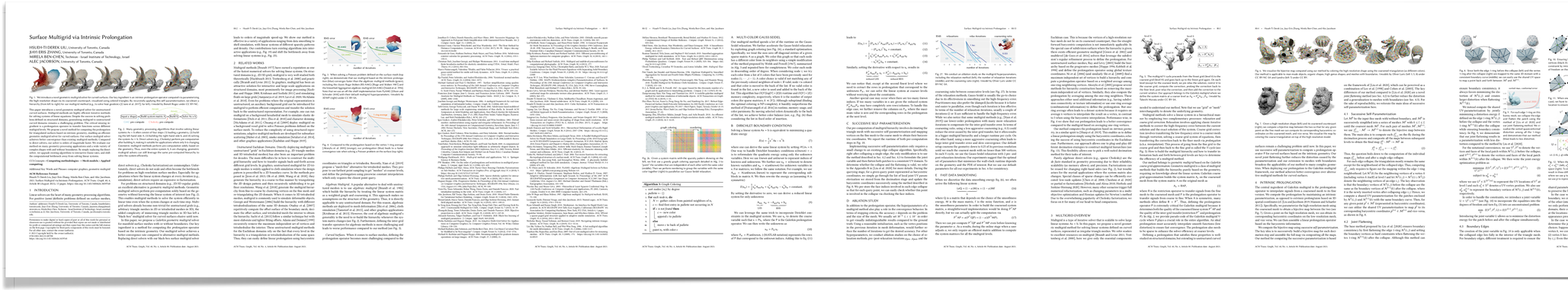
Cascaded Sobol' Sampling
Supplementary Material – 6



Cascaded Sobol' Sampling
Supplementary Material – 7



Cascaded Sobol' Sampling
Supplementary Material – 8



Surface Multigrid via Intrinsic Prolongation

FELICIE FUTERRE LIU, University of Toronto, Canada
JOST ERIS ZHANG, University of Toronto, Canada
MIRELA BEN-CHEN, Technion – Israel Institute of Technology, Israel
ALEC JACOBSON, University of Toronto, Canada

ACM Trans. Graph., Vol. 44, No. 4, Article 80. Publication date: August 2025.

Cascaded Sobol' Sampling
Supplementary Material

1 ADDITIONAL RENDERING RESULTS

The following figures show renderings of the example scenes found in the paper.

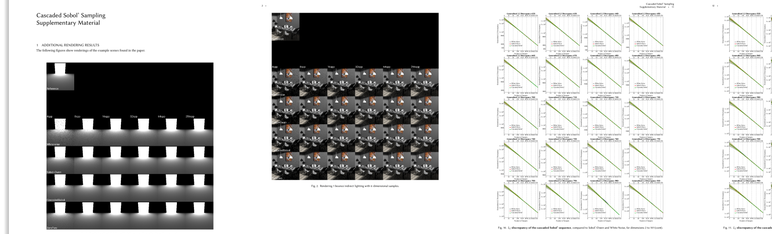
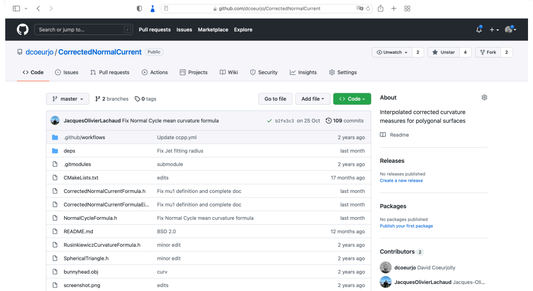


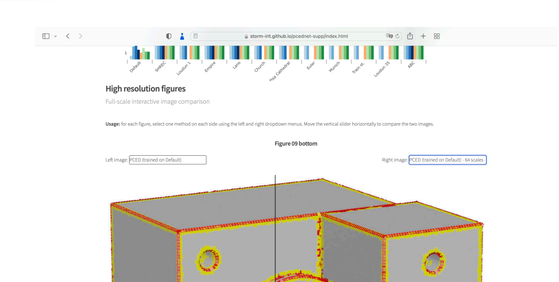
Fig. 1. Rendering direct lighting with 4 dimensional samples. Smooth integrands.



Cascaded Sobol' Sampling
Supplementary Material, Page 1 of 10







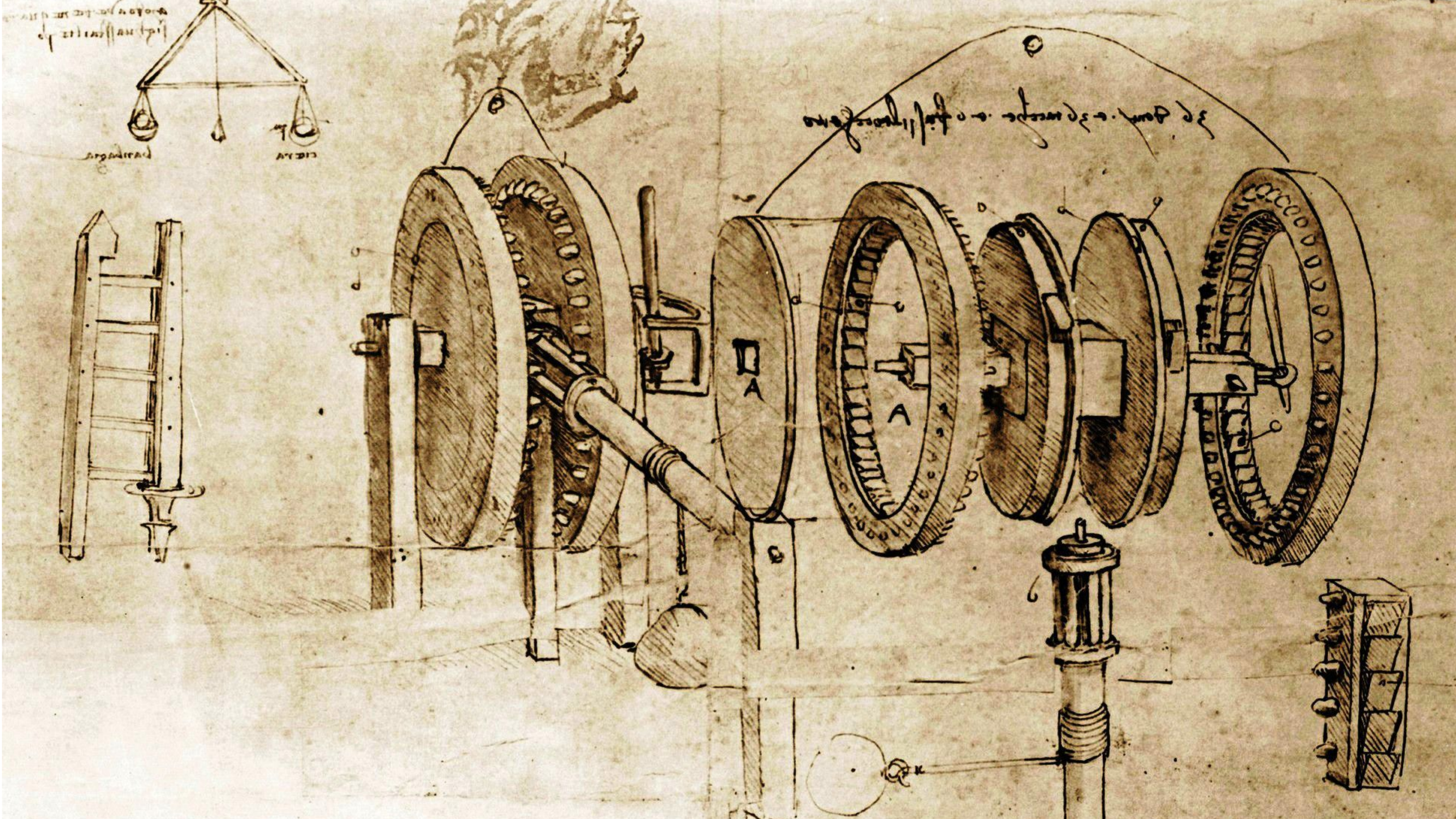


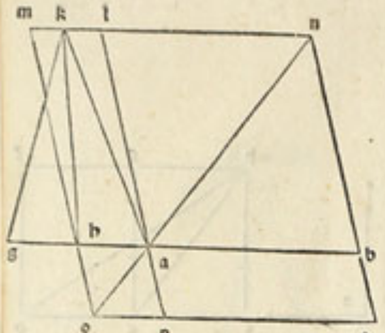


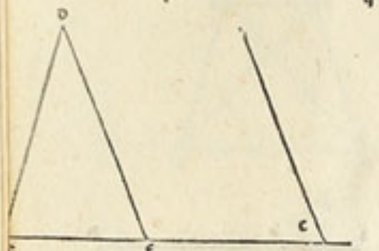
# Intérêt

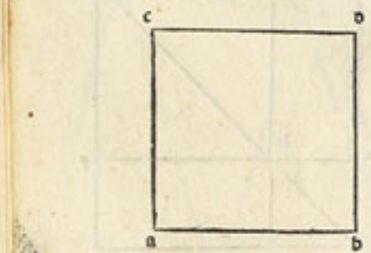
- Transmettre une idée

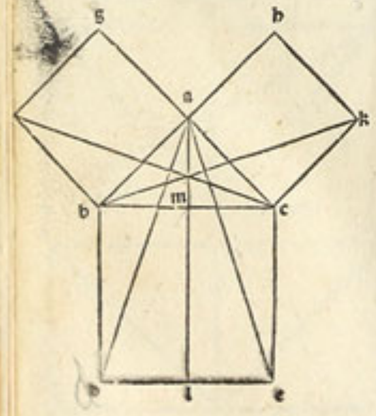
- ⚠ au « nan.. moi, je fais de la sciensss »






linee .a.b.ſm rectitudinem:quã pono equalem linee.c.f. baſi trianguli dati ſuper quã cõſtituo triangulũ vnũ ei cõplé & equilaterũ.qõ hoc modo facio. Conſtituo ãgu um.a.g.k.equalem angulo.c.& ngulum.g.a.k.equalem angulo.f.per.23.& quia g.a.poſita fuerat equalis.c.f.erit per.26.triangulus.g.a .k. equalis & equilaterus triangulo.c.f.d.diuidã ergo.g.a.per equalia in puncto.b.& ꝓtrahã.k.b. & produ cam a puncto.k.linea .m.k.n. equidiſtanté linee.g.b.critqꝫ per.38.triangul⁹.a.b k.equalis triangulo.g.b.k.tunc ſuper punctũ.a.linee.g.a.facia ãgulum.g.a.l.ꝑ 23. equalem angulo.c.dato:& cõplebo ſup baſim.a.b.& inter lineas.g.b.& m.n. equidiſtantes ſuperficiem equidiſtantiꝫ laterum.m.l.b.a.que ꝑ.41.dupla erit ad triangulũ.k.b.a quare equalis totali triangulo.k.g.a.quare & triangulo.d.c.f. ꝑ poſito:protrahatur ergo.b.n.equidiſtanté.a.l.& producã diametrum. n.q.quã ꝑ trahã quouſqꝫ cõcurrat cũ.m.b.in puncto.o.& cõplebo ſuperficiem equidiſtãtium laterum .m.o.n.q.& protrahatur.l.a.vſqꝫ ad.p.critqꝫ per precedentẽ ſupplementũ a.b.p.q. equale ſupplemento.m.l.b.a.quare & triangulo.d.c.f.& qꝝ per.15.angu lus .l.a.b.é equalis angulo.b.a.p.& ideo angulus.b.a.p.eſt equalis angulo.c. ꝑ ſuper datam lineã .a.b.deſcriptã eſſe ſuperficiem equidiſtantiũ laterum.a.b.p.q. equalem dato triangulo.d.c.f.cuius vterqꝫ duorum angulorum contra ſe poſitorum qui ſunt.a.& q.é equalis dato angulo.c.quod fuit propoſitum.

## Propoſitio .45.

Ex data linea quadratum deſcribere.

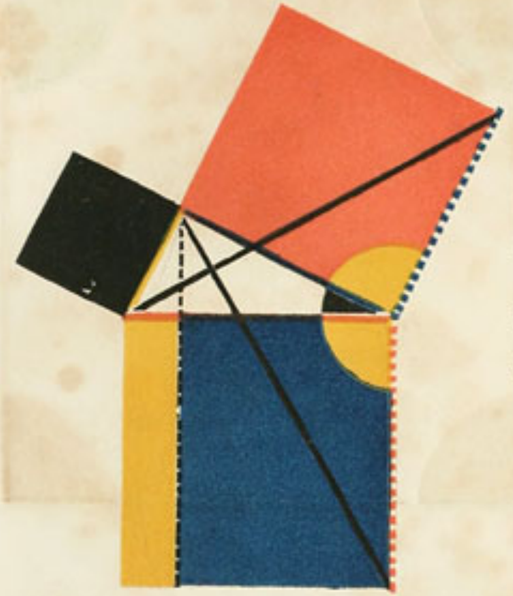
Sit data linea .a.b. ex qua volo ãdratum [illegible] punctis .a & .b.linee .a.b.ducto ꝑ.11.lineas.a.c.& b.d.perpendiculares ad li neam.a.b.que erũt equidiſtantes per vltimã ptem.28. & pono vtrã qꝫ earũ eidem.a.b.per ſecãm equalem & protrabo lineam.c.d.critqꝫ ipſa equalis et equidiſtans linee.a.b.per.33.& quia vterqꝫ duorum angulorũ. a.& b.eſt rectus.erit vterqꝫ duorũ.c.& d.rectus per vltimã ptem.29.ergo per diffiniti onem.a.b.c.d.é quadratum quod eſt propoſitũ. Idem aliter ſit.a.c. perpendicu laris ſuper lineam.ab.per.11.& ſit ei equalis vt prius & a puncto.c.per.31. ducatur c.d.equidiſtans.a.b.& ponatur equalis ei & ducatur linea.d.b.que per.33.erit eã lis & equidiſtans .a.c.& omnes anguli recti per vltimã ptem.29.quare per diffiniti onem habemus propoſitum.

## Propoſitio .46.

In omni triangulo rectangulo quadratum qõ a latere recto angulo oppoſito in ſemetipſo ducto deſcribit equũ é duobus quadratis que ex duob⁹ reliquis lateribus conſcribuntur.

Sit triangulus.a.b.c.cuius angulus.a.ſit rectus dico ꝙ quadratũ lateris .b.c.equũ é quadrato.a.b.& quadrato.a.c.ſũ ſumptis. Quadrabo ꝑ hec tria latera ſm doctrinã ꝓcedentis:ſitqꝫ ẽdratũ.b.c.ſuꝑficies.b.c.d.c.& ẽdratũ.b. a.ſuꝑficies .b.f.g.a.& ẽdratũ.a.c.ſuꝑficies.a.c.b.k.ab ãgulo.a.recto ducã ad baſ iz.d.c. baſiz maximi ẽdrati tres lineas .f.a.l eãdiſtãté vtriqꝫ lateri.b.d.& c.c.q ſe cet.b.c i pũcto.m.& ꝓtheniſas.a.d.& a.c.itéqꝫ a duob⁹ reliqs ãglis triãgli q ſũt b.& c.ducã ad duos anglos duoꝝ ẽdratoꝝ mioꝝ duas liéas ſe inſecãtes itra ipſũ



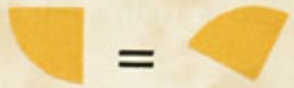


IN *a right angled triangle*  *the ſquare on the hypotenuſe* ——— *is equal to the ſum of the ſquares of the ſides,* (——— *and* ———).

On ———, ——— and ——— deſcribe ſquares, (pr. 46.)

Draw ▪▪▪▪▪▪▪▪▪ ‖ ▪ ▪ ▪ ▪ ▪ ▪ (pr. 31.)

alſo draw ——— and ———.





To each add  ∴ ,

——— = ▪ ▪ ▪ ▪ ▪ ▪ and ——— = ▪▪▪▪▪▪▪▪▪;



# Intérêt

- Transmettre une idée

- ⚠ au « nan.. moi, je fais de la sciensss »
- ⚠ fancy-schmancy

XYZRGB Thai Statuette

Stanford 3D Scanning Repository

10M triangles



XYZRGB Thai Statuette

Stanford 3D Scanning Repository

10M triangles



# Intérêt

- Transmettre une idée

- ⚠ au « nan.. moi, je fais de la sciensss »
- ⚠ fancy-schmancy

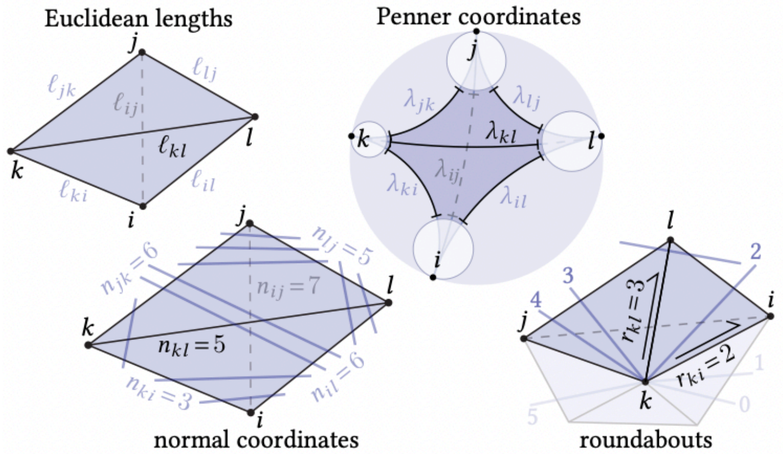
*Support pour l’élaboration de l’idée*

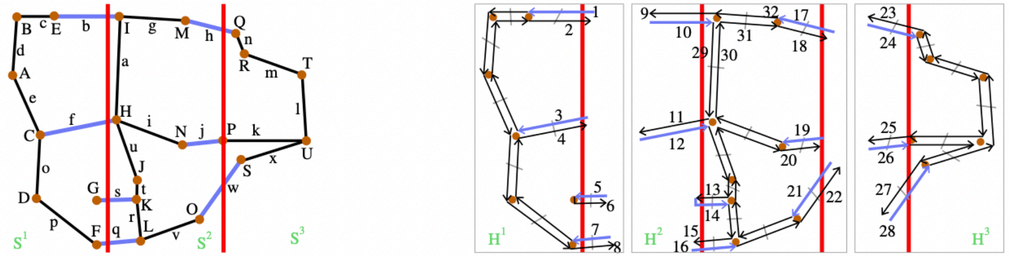
# Illustrations

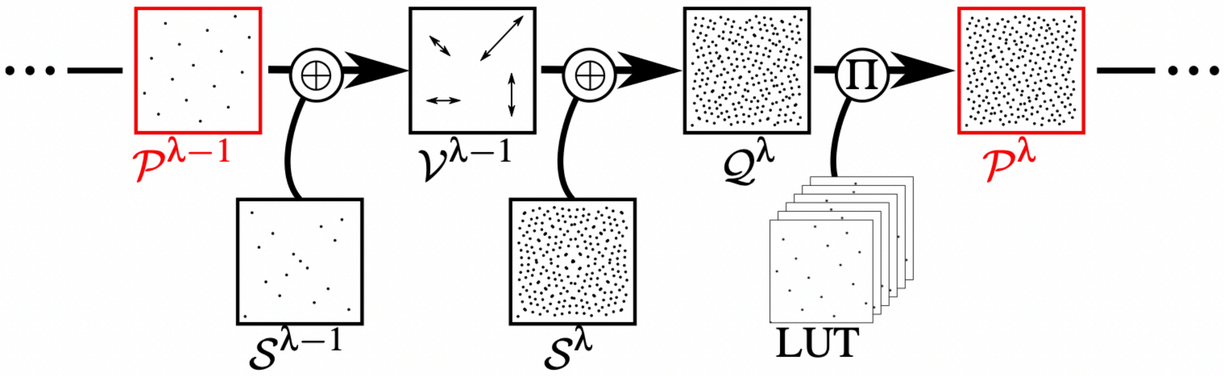
# Objectifs

- Support graphique à une idée clef
- Glossaire graphique pour les notations
- Format « pipeline/workflow »

⚠ si possible local à un paragraphe





from 3-manifold geometry [Kneser 1929; Haken 1961] and computational topology [Schaefer et al. 2002; Agol et al. 2006; Erickson and Nayyeri 2013]. We enrich this construction with a combinatorial analogue of signposts, which we call *roundabouts* (Section 5).

The other question is how to interpolate data across triangulations, such as vertex or texture coordinates. The natural choice for discrete conformal maps is to use piecewise projective interpolation [Bobenko et al. 2015], which can be implemented via standard homogeneous coordinates [Springborn et al. 2008, Section 3.4]. We extend this idea to variable triangulations by laying out triangles in the *light cone* rather than the Euclidean plane (see Section 6.0.1).

Importantly, our approach to discrete conformal mapping depends critically on the hyperbolic picture. Without this picture, one could not use the implicit connectivity encoding (which depends on hyperbolic straightening), and would be forced to explicitly maintain the full connectivity of the common refinement, as done by Sun et al. [2015]. Likewise, our high-quality interpolation scheme (shown in Figure 3) relies on calculations in the light cone model of the hyperbolic plane.

### 2.3 Other Methods

*Conformal Mapping.* Other methods for conformal parameterization do not provide a general solution. For instance, early methods based on linear finite elements [Lévy et al. 2002; Desbrun et al. 2002; Mullen et al. 2008] do not guarantee injectivity, nor do they handle cone singularities. More recent linear methods support cones [Ben-Chen et al. 2008; Vintescu et al. 2017; Sawhney and Crane 2017], but injectivity is still missing. *Orbifold* methods (*e.g.*, [Aigerman and Lipman 2016]) provide injectivity, but support only a restricted set of cone configurations where cone angles cannot be prescribed. Angle-based methods [Sheffer et al. 2005] rely on nonconvex optimization, with no general convergence guarantees. Finally, Bobenko et al. [2015] and later Zhang et al. [2014] provide connections between discrete conformal equivalence and *circle patterns*.

*Injective Mapping.* Discrete uniformization has a special relationship to methods for locally injective mapping, since CETM is often used for initialization [Chien et al. 2016; Campen and Zorin 2017b; Campen et al. 2019]; we provide even stronger guarantees. Unlike [Mandad and Campen 2019; Shen et al. 2019] we do not claim to guarantee injectivity in floating point—yet still achieve injectivity in extremely challenging scenarios (Section 8.3.2). Overall we observe that the freedom to modify the triangulation during optimization leads to significantly improved robustness—see Section 8.3.

## 3 PRELIMINARIES

This section provides essential definitions needed to motivate and derive our algorithms; some readers may wish to skip ahead to Section 4, and return here for reference. The most important concept is illustrated in Figure 14: any triangle mesh can be interpreted as both a *Euclidean polyhedron* (Section 3.2) and a *decorated ideal polyhedron* (Section 3.3), leading to a definition of conformal equivalence across different triangulations (Section 3.5). For further background, see Bobenko et al. [2015] and Springborn [2019].

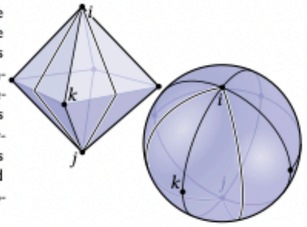
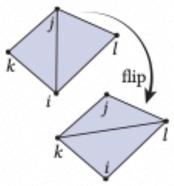



Fig. 11. An edge or triangle in a Δ-complex might not be uniquely determined by its vertices. Here, performing intrinsic edge flips on an octahedron yields two distinct edges between the same pair of vertices $i$ and $j$, and two triangles with the same vertices $i$, $j$, and $k$. The sphere depicts the abstract connectivity.

### 3.1 Combinatorial Polyhedra

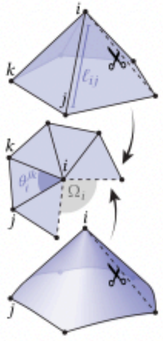
Throughout we use $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ to denote the connectivity of a manifold triangulation with vertices $\mathsf{V}$, edges $\mathsf{E}$, and faces $\mathsf{F}$; we assume $\mathsf{T}$ is orientable purely to simplify exposition. Even when the input is an ordinary (simplicial) triangulation, we may need to construct triangulations where, *e.g.*, multiple edges connect the same two vertices, or two triangles share the same three vertices (Figure 11). Formally, we use *triangulation* to mean a Δ-complex in the sense of Hatcher [2002, Section 2.1], which we encode via a *halfedge data structure* [Botsch et al. 2010, Section 2.3]. Though edges and faces are not uniquely determined by their vertices, for brevity we will still denote them by vertex pairs $ij \in \mathsf{E}$ and triples $ijk \in \mathsf{F}$, *resp.*, where $i$, $j$, and $k$ need not be distinct. The notation $\phi_i^{jk}$ indicates a quantity $\phi$ at corner $i$ of a triangle $ijk$.

*3.1.1 Combinatorial Edge Flip.* For two triangles sharing a common edge, an *edge flip* replaces this edge with the opposite diagonal—we will need this operation in order to construct intrinsic Delaunay triangulations (Section 3.4). If we locally index the vertices of these two triangles as depicted in the inset figure, then the edge flip replaces the original triangles $ijk$ and $jil$ with $jkl$ and $lki$. Any other data stored on the triangulation must also be updated, as depicted in Figure 12 and discussed in Sections 3.2.1, 3.3.4, 5.1.1 and 5.2.1.




### 3.2 Euclidean Polyhedra

A *Euclidean polyhedron* is a surface that looks like the flat Euclidean plane everywhere except at a finite collection of *cone points*. The canonical example is an ordinary triangle mesh in $\mathbb{R}^3$, where the neighborhood around each vertex is isometric to a piece of a circular cone (see inset). For uniformization, however, we do not need to keep track of how the surface is embedded in space. Instead, we can store a purely *intrinsic* description of the geometry, given by the edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$ of a triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$. If these lengths satisfy the triangle inequalities in each triangle $ijk \in \mathsf{F}$, then we call $\ell$ a *discrete metric*. Other quantities such as corner





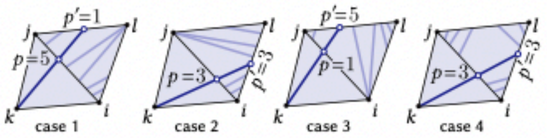

apply a scaling $\tilde{q}_i = e^{u_i} q_i$ to each vertex, we get another chordal triangle; central projection between points $x$ and $\tilde{x}$ then gives the circumcircle-preserving projective map used for interpolation in Springborn et al. [2008] (Figure 19, *center*). The real power of this construction is that it enables us to map between different triangulations of the same vertices. Consider for instance two chordal triangles $ijk$, $jil$—if we apply a flip and then scale the vertices $q$ as before, we get two chordal triangles $lki$, $klj$ sitting "above" the other ones (Figure 19, *right*). The map between these triangle pairs is still given by central projection, and can be expressed as a piecewise projective map on the four triangles of their common subdivision. This idea is extended in the supplement to general pairs of triangulations.

### 6.1 Tracing Edges

For the moment, consider just two triangulations $\mathsf{T}_1, \mathsf{T}_2$. We use the normal coordinates $n : \mathsf{E}_2 \to \mathbb{Z}_{\geq 0}$ to trace out the sequence of edges in $\mathsf{T}_2$ crossed by each edge of $\mathsf{T}_1$ (Section 6.1.1). The roundabouts $r : \mathsf{H}_2 \to \mathbb{Z}_{\geq 0}$ uniquely identify each traced sequence with the appropriate element of $\mathsf{E}_1$. To get the curve geometry, we lay out a triangle strip in the Euclidean or hyperbolic plane, and draw a straight line between endpoints (Section 6.2). The final curve is encoded by 1D barycentric coordinates $s, t \in [0, 1]$ on each intersected edge. We enumerate points where edges of $\mathsf{T}_1$ cross a halfedge $\overrightarrow{ij}$ of $\mathsf{T}_2$ by a *crossing index* $p \in \{1, \ldots, n_{ij}\}$ (see Figure 20, *left*).

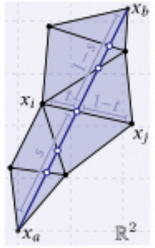
*6.1.1 Topological Tracing.* To trace out all the edges of $\mathsf{T}_1$ over $\mathsf{T}_2$, we iterate over the halfedges $\overrightarrow{ij} \in \mathsf{H}_2$ and trace edges emanating from the opposite corner $k$ (Figure 20, *right*), namely, the edges with indices $p = 1 + n_{ki}, \ldots, 1 + n_{ki} + e_{\overrightarrow{ij}}$. This procedure is detailed in Algorithm 1. We identify the edge of $\mathsf{T}_1$ corresponding to each traced curve $p$ by incrementing the roundabout $r_{\overrightarrow{ki}}$ by $p - n_{ki} - 1$. (By marking traced edges in $\mathsf{T}_1$, we avoid tracing edges twice.) Note that roundabouts must be used even for curves shared by both triangulations, since after a sequence of edge flips they may no longer correspond to the same logical edge.

The procedure NextEdge (Algorithm 2) uses the normal coordinates $n$ to determine the next halfedge $\overrightarrow{ij}$ and crossing index $p$ along the curve. The image below depicts the four possible cases—this pattern of edge crossings can be determined solely using the normal coordinates for the triangle $jil$, via the formulas given in Section 5.1. See Appendix A.2 for further details.




Note that the tracing procedure gives us each edge from $\mathsf{T}_1$ as a sequence of edge crossings on $\mathsf{T}_2$. To express the edges from $\mathsf{T}_2$ as sequences of $\mathsf{T}_1$ edge crossings, we allocate an array of size $n_{ij}$ for each edge $ij \in \mathsf{E}_2$. Each time a traced edge $ab \in \mathsf{T}_1$ crosses $ij$, we store a reference to $ab$ in entry $p$ of the array (using roundabouts to get the edge index).

### 6.2 Recovering Geodesics

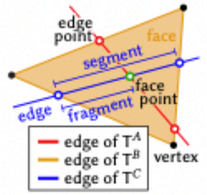



To get the geometry of each traced edge $ab \in \mathsf{E}_2$, we use the crossing sequences computed in Section 6.1 and the edge lengths $\ell$ to incrementally lay out a triangle strip in the (Euclidean or hyperbolic) plane. We then intersect each interior edge $ij$ of this strip with the line from $a$ to $b$—by construction, this line will be contained entirely inside the strip. In particular, if $x_i \in \mathbb{R}^2$ are the vertices of a Euclidean triangle strip, we can solve the equation

$$(1 - s)x_a + sx_b = (1 - t)x_i + tx_j$$

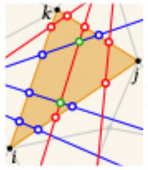
for the barycentric coordinates $s, t \in [0, 1]$ of the intersection point. The hyperbolic case is conceptually the same except that we work in the hyperboloid model, and and also compute a scale factor $u$ at each intersection point—see Appendix A.3 for details.

### 6.3 Common Refinement

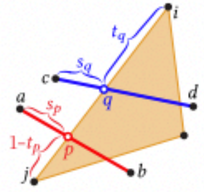



We now construct the common refinement $\mathsf{S}$ of $\mathsf{T}^A$, $\mathsf{T}^B$, and $\mathsf{T}^C$. Here and in Appendix B we will refer to points along edges of $\mathsf{T}^B$ (resulting from tracing) as *edge points*, and any new vertices inserted into polygons as *face points*, reserving *vertex* for elements of $\mathsf{V}$. Likewise, an *edge* is the complete edge of some triangulation, a *segment* is the restriction of an edge to a triangle, and a *fragment* is a piece of a segment (see inset).

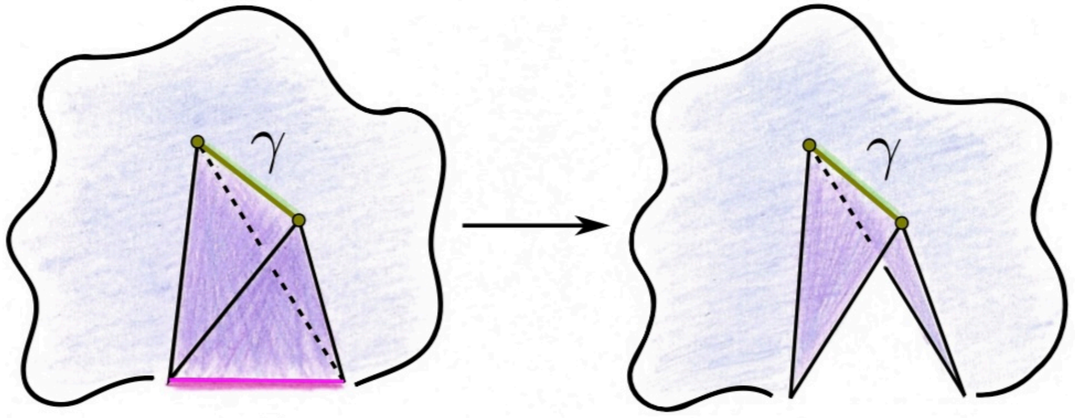
*6.3.1 Connectivity.* First, for each edge $ij \in \mathsf{E}^B$, we use the procedures from Sections 6.1 and 6.2 to trace out (i) a Euclidean geodesic over $\mathsf{T}^A$ to obtain edge sequences and barycentric coordinates $(s, t)$, and (ii) a hyperbolic geodesic over $\mathsf{T}^C$ to obtain edge sequences, barycentric coordinates, and scale factors $u$. To determine the connectivity of $\mathsf{S}$ we slice up each triangle $ijk \in \mathsf{F}^B$ independently, via a strategy similar to Sharp et al. [2019b, Section 3.4]. To avoid computing segment-segment intersections directly (which is not numerically robust), we devise a strategy that takes advantage of combinatorial information. Floating-point values serve only to determine the ordering of intersection points along edges—and since neighboring triangles have identical barycentric coordinates along their shared edge, we always obtain a consistent tessellation. See Appendix B for details.

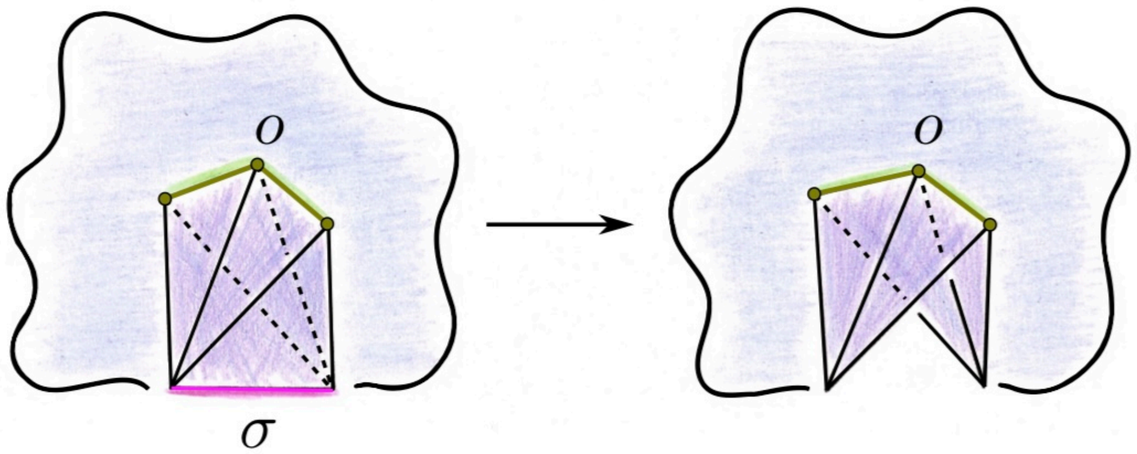


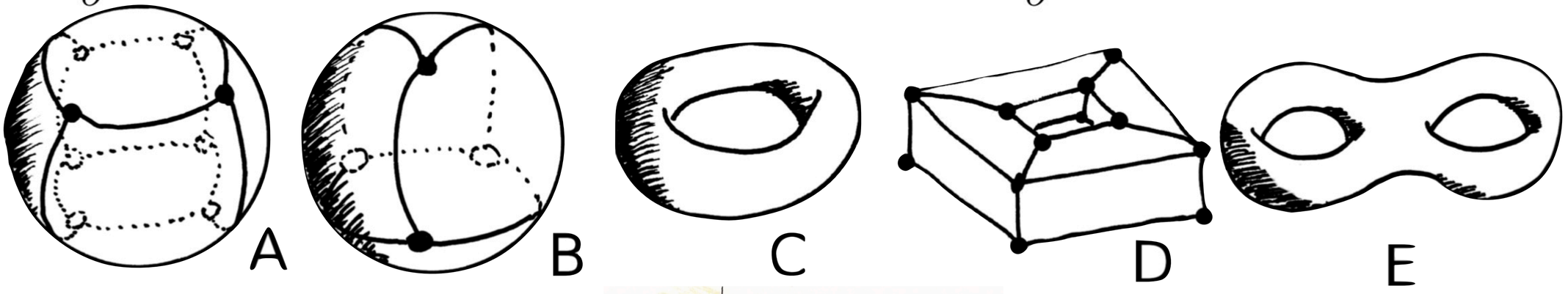

### 6.4 Interpolation

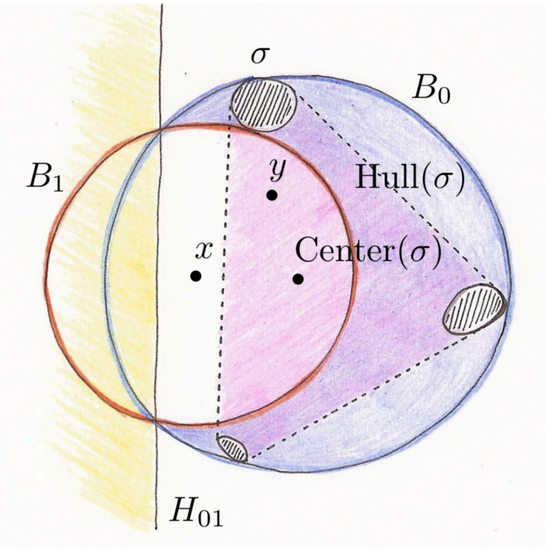



The vertex coordinates $f_i$ and texture coordinates $z_i^{jk}$ define piecewise functions over the faces of $\mathsf{T}^A$ and $\mathsf{T}^C$, *resp.*; we now sample these functions onto $\mathsf{S}$. To do so, we will also need the scale factors $u$ obtained while tracing hyperbolic geodesics. We again process each triangle $ijk \in \mathsf{T}^B$ independently. First, we interpolate data


$\gamma$
$\gamma$
$\sigma$


$o$
$o$
$\sigma$


A
B
C
D
E

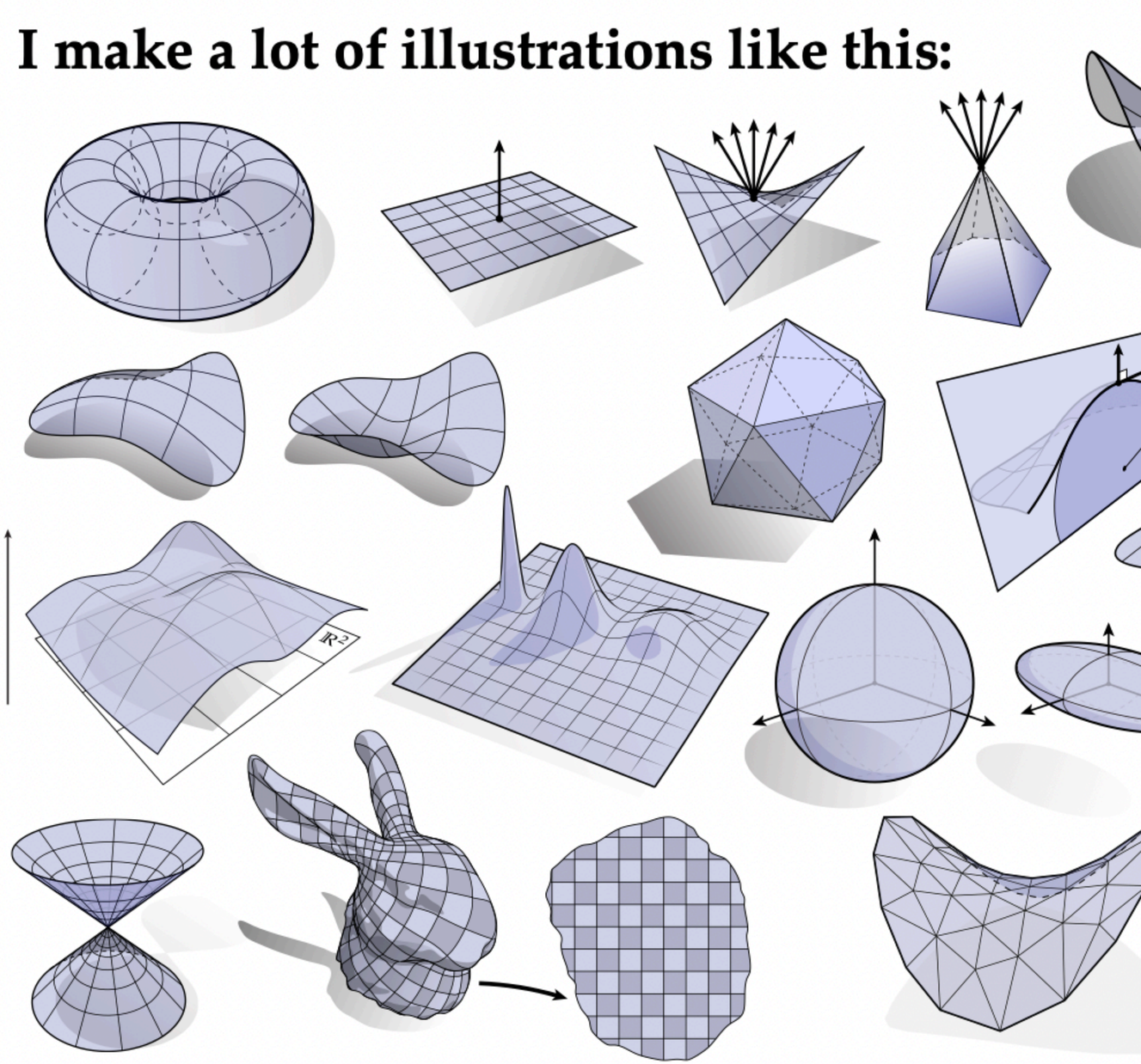

$\sigma$
$B_0$
$B_1$
$y$
Hull($\sigma$)
$x$
Center($\sigma$)
$H_{01}$

# Dessin vectoriel

- Inkscape, Illustrator,
- powepoint/keynote

⚠ cohérence visuelle

⚠ fausse 3d




https://www.cs.cmu.edu/~kmcrane/Projects/Other/IllustratingGeometry.pdf

# Illustrations « code-based »

## Tikz overlay

```
\newcommand{\tikzframe}[3][]{%
  \tikz[baseline, anchor = base]{%
    \node[rounded corners, #1] (#2) {#3} ;%
  }%
}

\begin{align*}
  S(a,b)  &= \sum_{
               \tikzframe[draw]{initial bound}{$\scriptstyle i=a$}
             }^{b} i \\
          &= \sum_{
               \tikzframe[fill = blue!40]{full bound}{$\scriptstyle i=0$}
             }^{b} i
           - \sum_{
               \tikzframe[fill = blue!40]{base bound}{$\scriptstyle i=0$}
             }^{
               \tikzframe[fill = red!40]{excess bound}{$\scriptstyle a-1$}
             } i
\end{align*}

\begin{tikzpicture}[overlay]
  \draw[thick, -latex, rounded corners]
    (initial bound) -| (excess bound) ;
  \draw[thick, -latex, rounded corners]
    (full bound) -- (base bound) ;
  \draw[thick, -latex, dashed, rounded corners]
    (base bound) -- ++ (1cm, 0) |- (excess bound) ;
\end{tikzpicture}
```

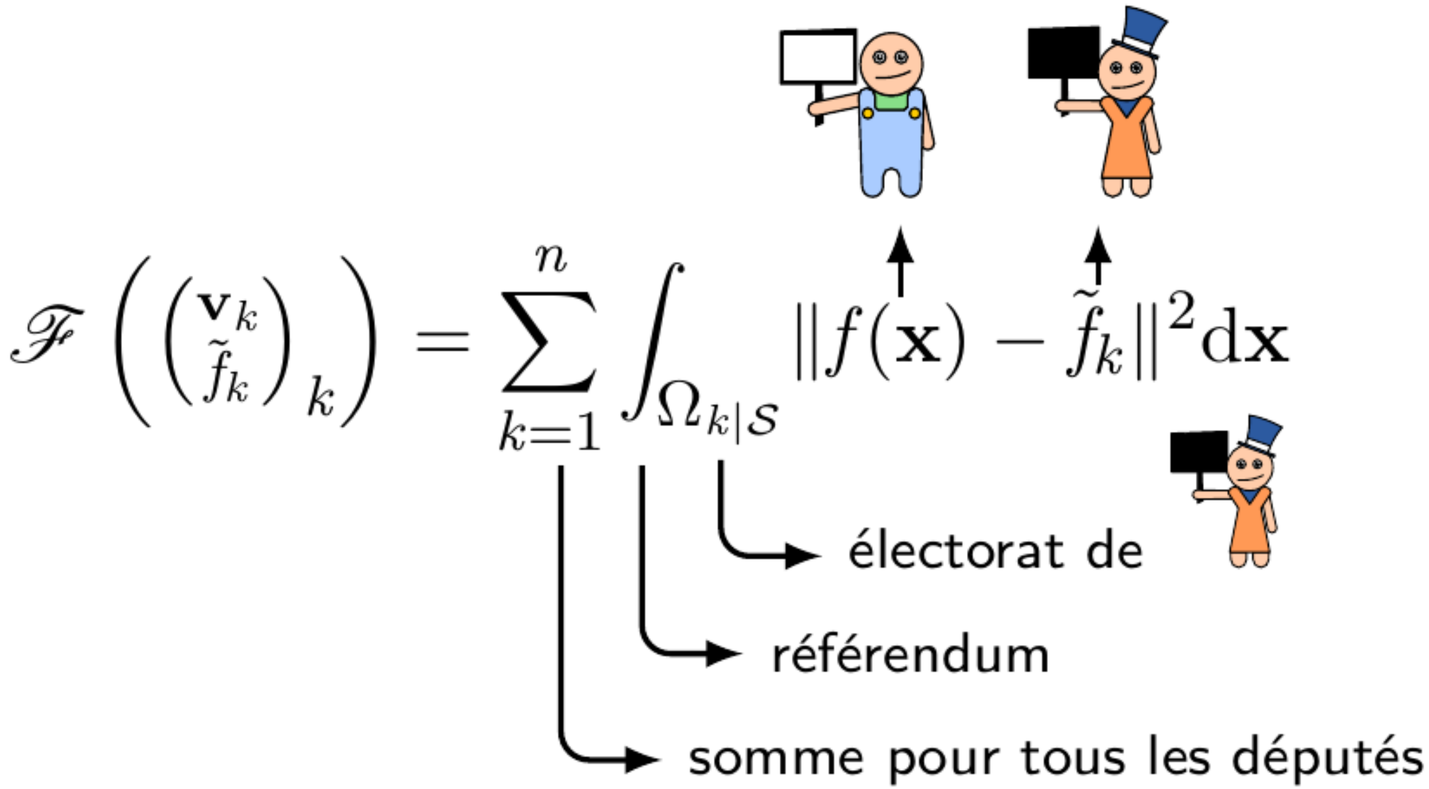
$$S(a,b) = \sum_{i=a}^{b} i$$

$$= \sum_{i=0}^{b} i - \sum_{i=0}^{a-1} i$$

# Illustrations « code-based »

## Tikz overlay

Le problème est formulé sous forme d'une minimisation de fonction.



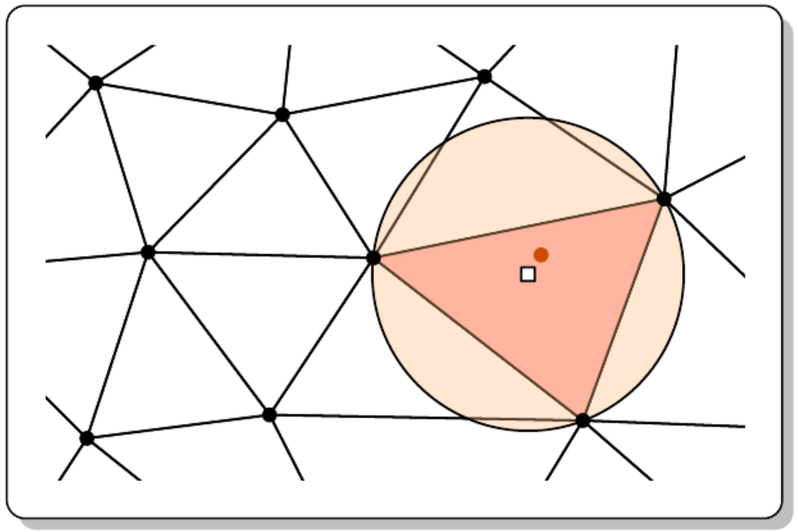
# Illustrations « code-based »

## Tikz overlay

### Sketch : construction by validation

**Algorithm**

- ● ← a few initial sites → from the restriction domain
- △ ← Delaunay of ●
- **while** △ *contains an invalid simplex* ▲ **do**
  - □ ← circumcenter of ▲
  - ● ← site nearest to □
  - **if** ● ∈ ● **then**
    - mark ▲ as valid
  - **else**
    - add ● to ●
    - update △ with ●



# Illustrations « code-based »

## Tikz : barycentres, spy

```
\begin{figure}

 \begin{tikzpicture}[spy using outlines = {circle, thick, magnification = 8, connect spies}]
    \node[draw, fill = white, rounded corners, drop shadow] (image) {
      \includegraphics[width = 0.3\linewidth]{vangogh}
    } ;

    % debug grid, comment me out
    \draw[draw = orange, thick]
      (image) ++(-1.5, -1.5) grid ++(3, 3)
    ;

    \spy[size = 2cm]
      on ($(image) + (1.2, 0.5)$)
      in node[anchor = north] at ($ (image.north) + (5cm, 0)$)
    ;

    \spy[size = 2cm]
      on ($(image)!0.2!(image.south east)$)
      in node[anchor = south] at ($ (image.south) + (3.5cm, 0)$)
    ;
  \end{tikzpicture}

  \caption{This is zooming}
\end{figure}
```

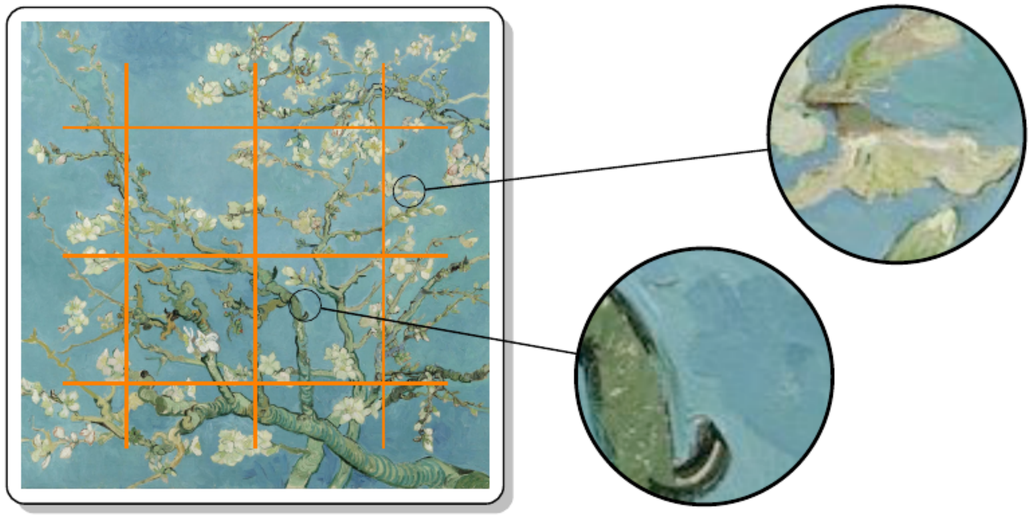
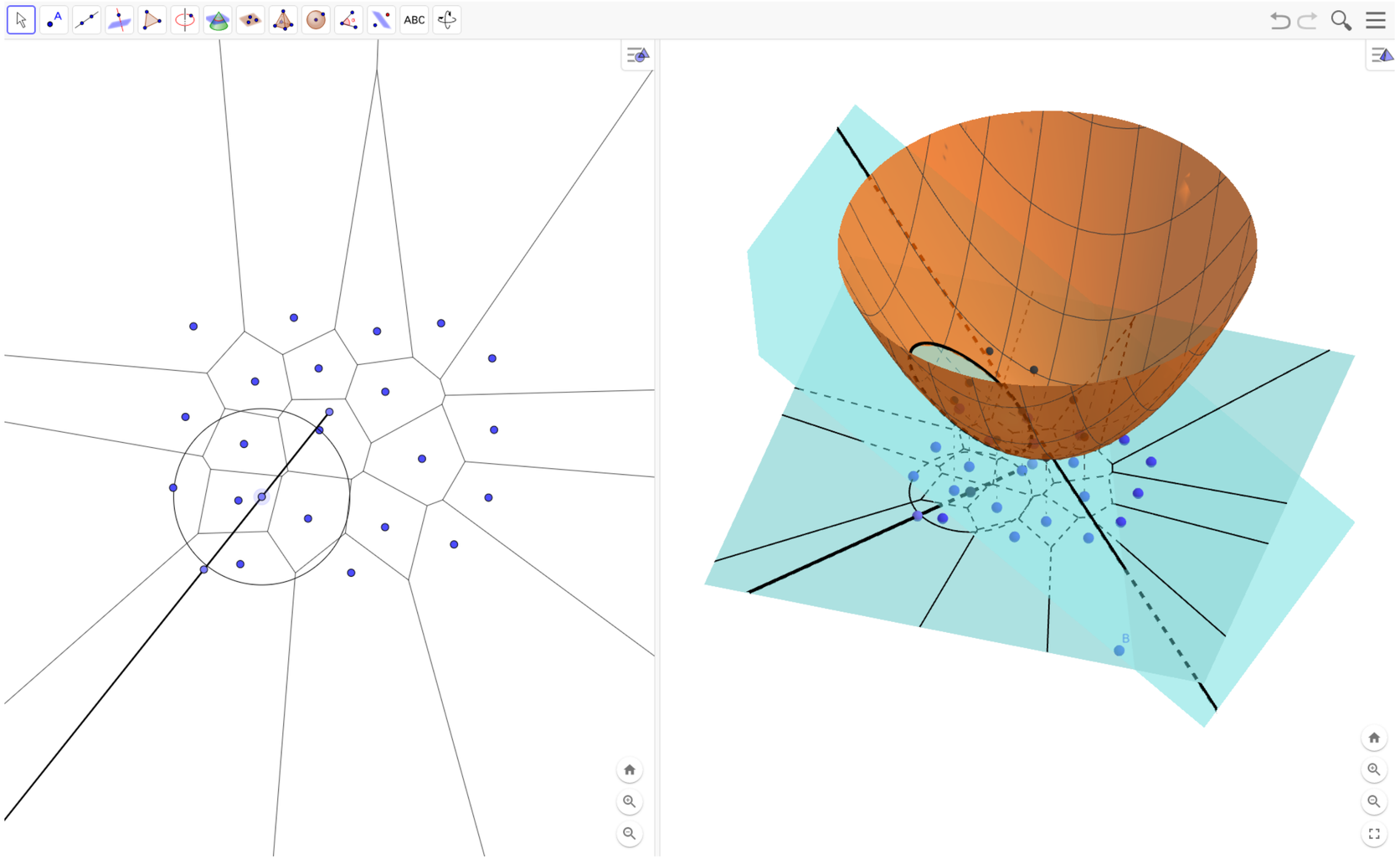
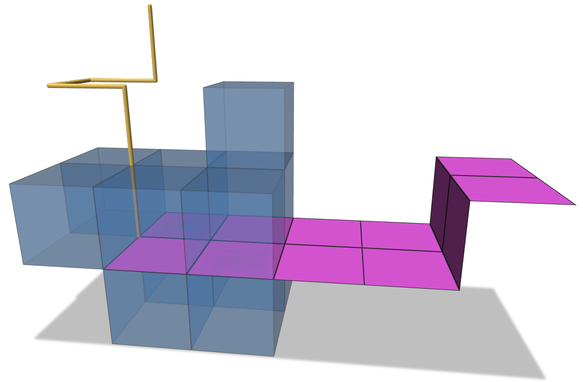


Figure 1: This is zooming

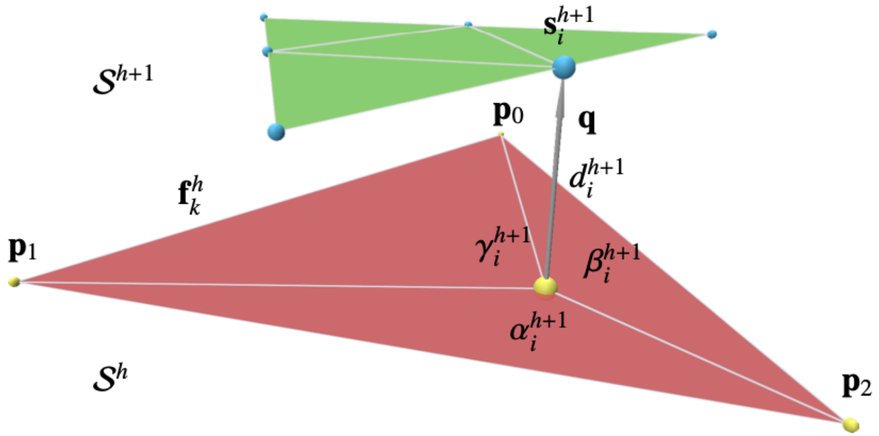
# Geogebra et export html



# Modélisation 3d



- Basé code (hardcodé, polyscope)



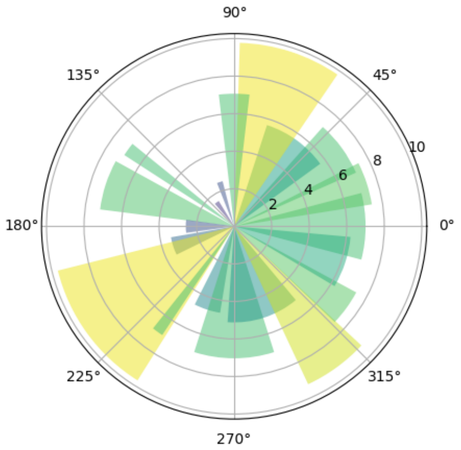

- Blender

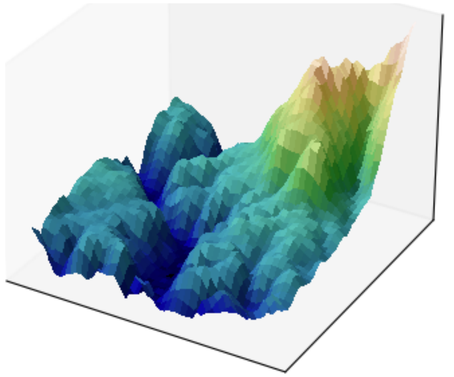
⚠ archiver les sources

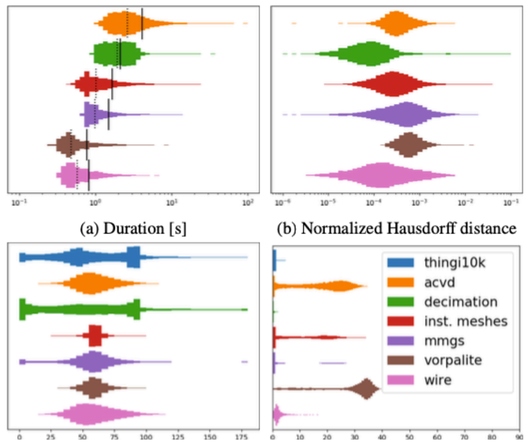
# Montrer les résultats

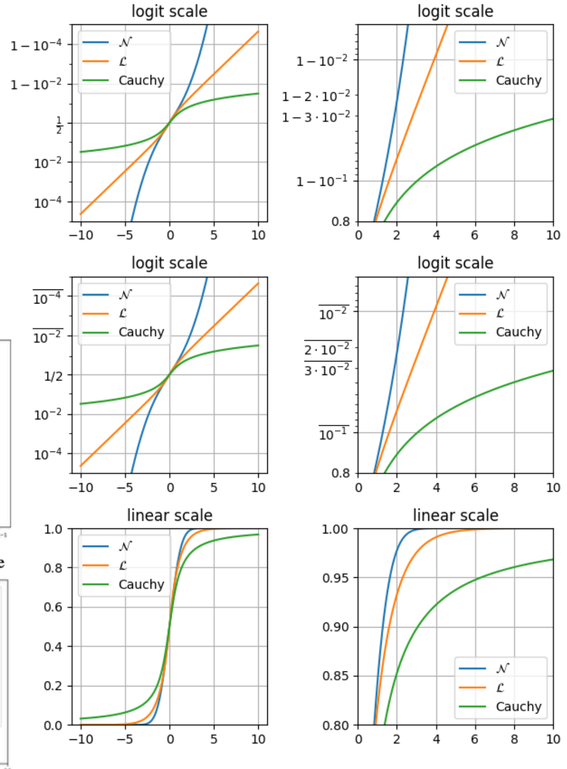
# Dataviz / analyse

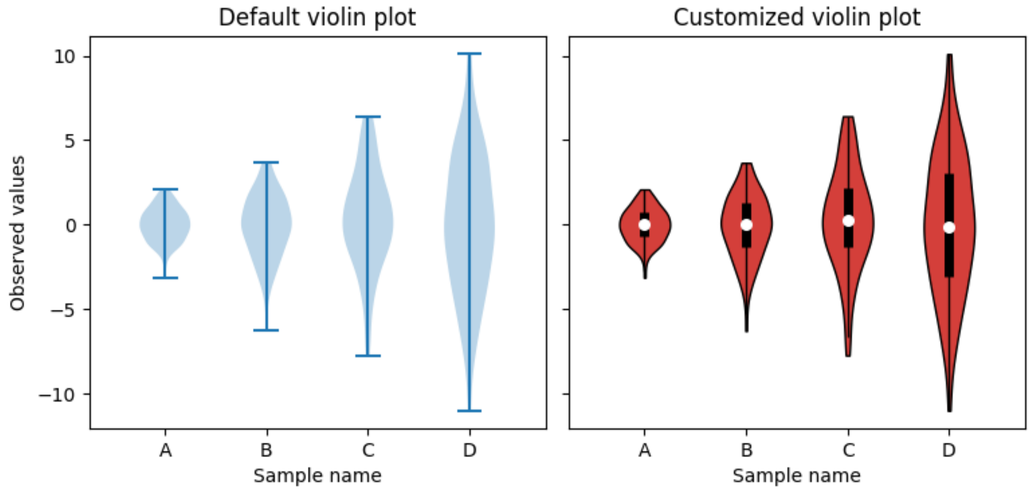
- Gnuplot
- matplotlib / searborn
- Mathematica / Matlab / Tableau…





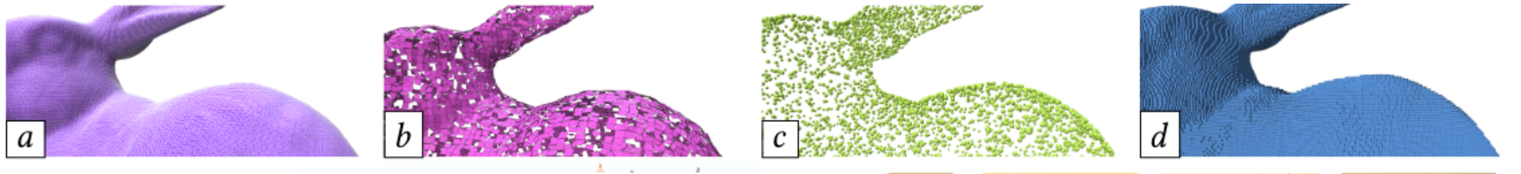



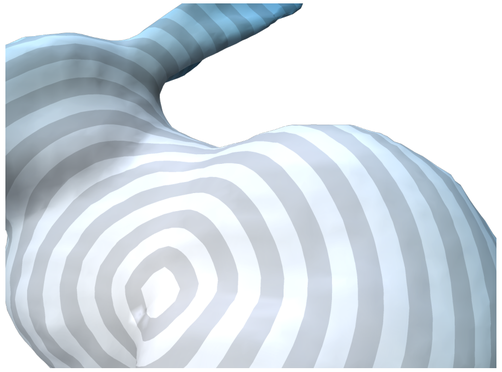

⚠ archiver les scripts et données pour replicabilité

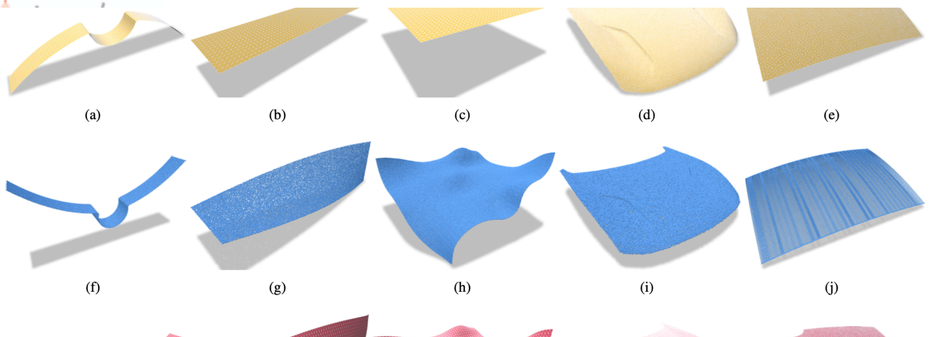
# Contenu 3d

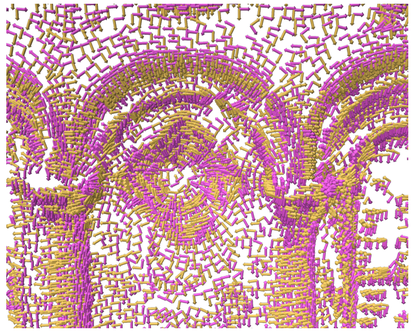
- Basé code (polyscope)
- Blender (stagging) [demo]

  tuto: https://www.silviasellan.com/blender_figure.html
- Offline / scripting

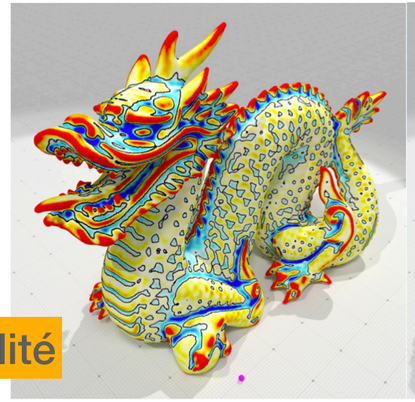


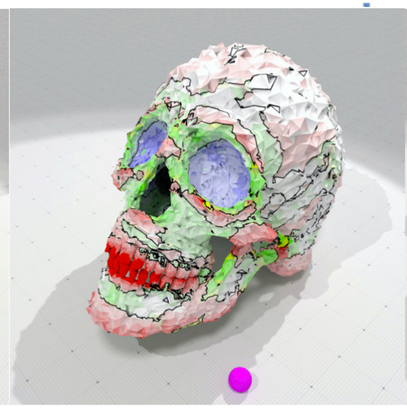



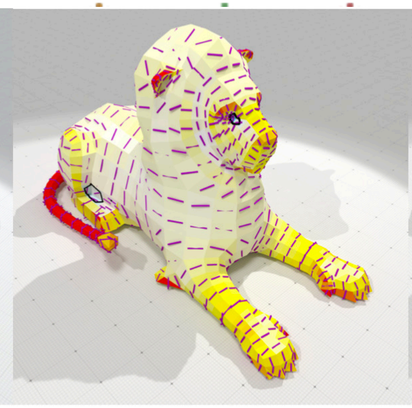


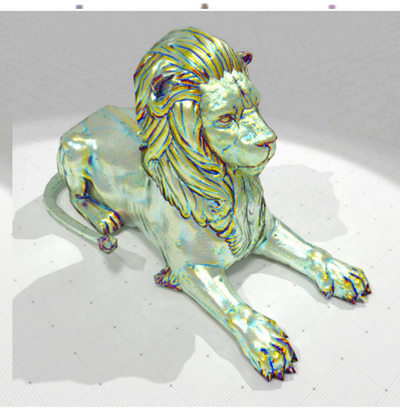











⚠ archiver les scripts et données pour replicabilité

# Scripting

- Blender + Python

  https://www.silviasellan.com/blender_figure.html

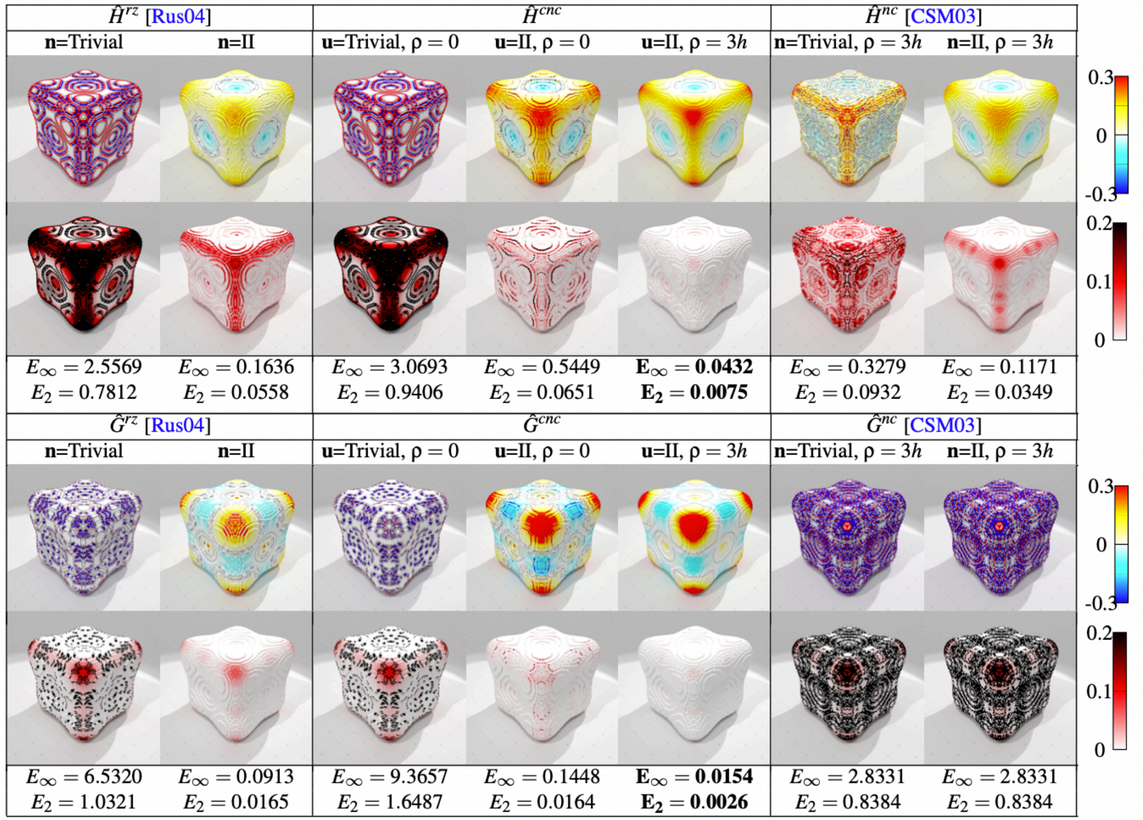
  *blender --background scene.blend —python template.py*

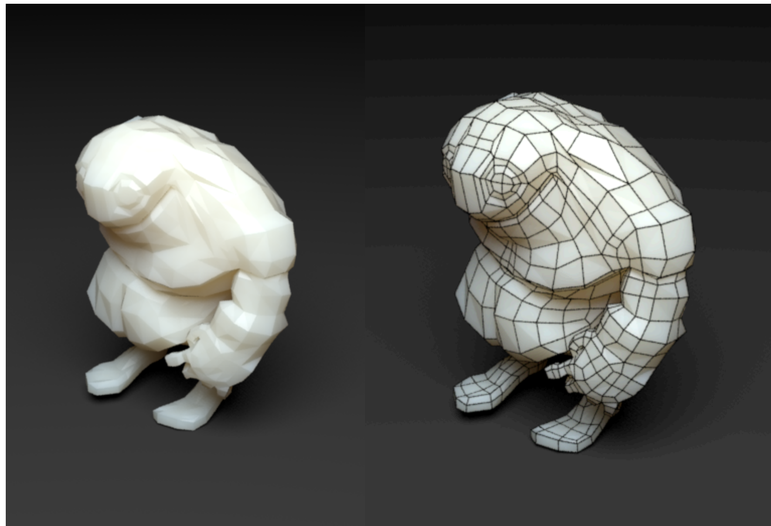
- polyscope

- pyrender

  https://github.com/qnzhou/PyRenderer

  sudo docker run -it --rm -v `pwd`:/data/ qnzhou/pyrender bash -c « . /usr/local/mitsuba/setpath.sh; render.py -q -o /data -w  /data/octa.obj"



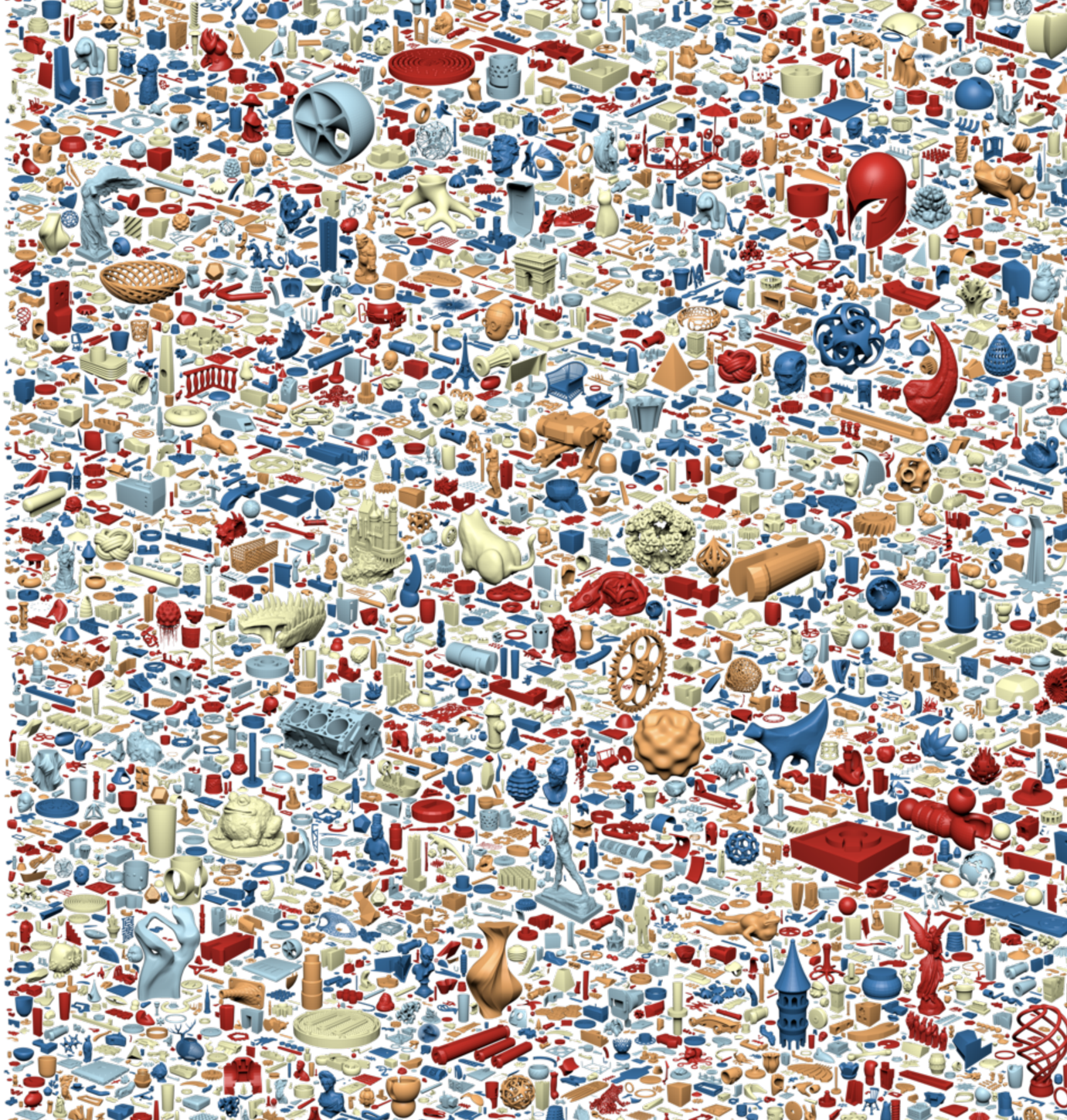



# BlenderToolbox

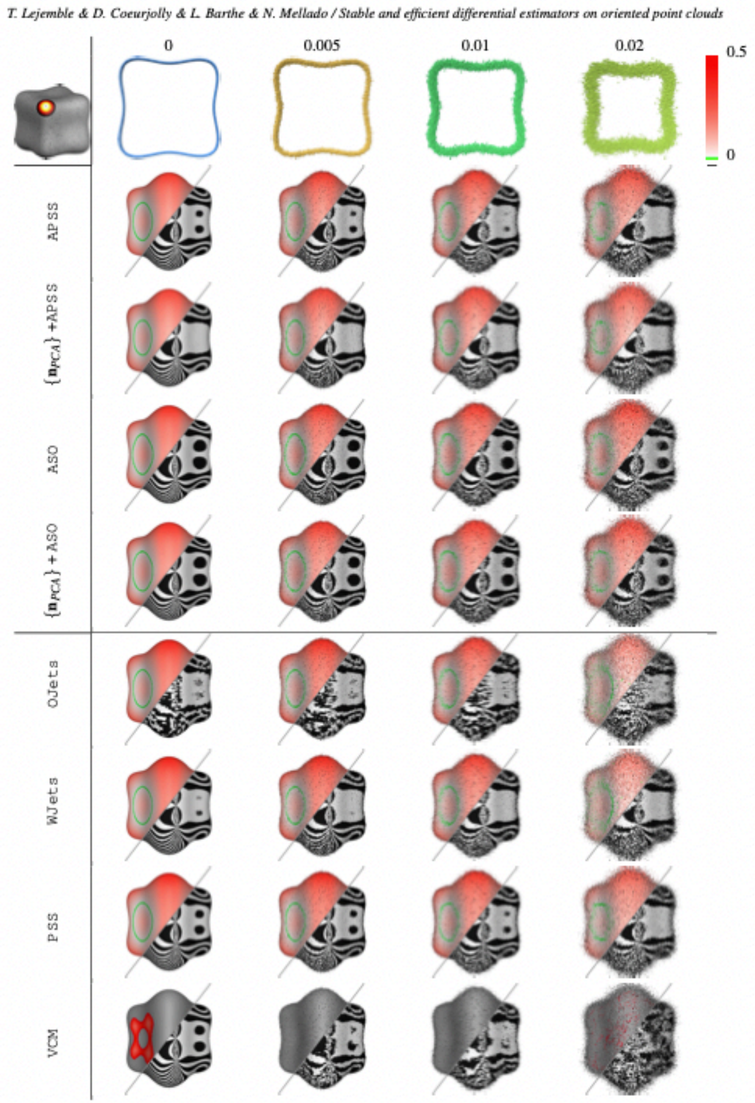
https://github.com/HTDerekLiu/BlenderToolbox

# Mosaic

https://github.com/qnzhou/Mosaic



# Latex tweaks



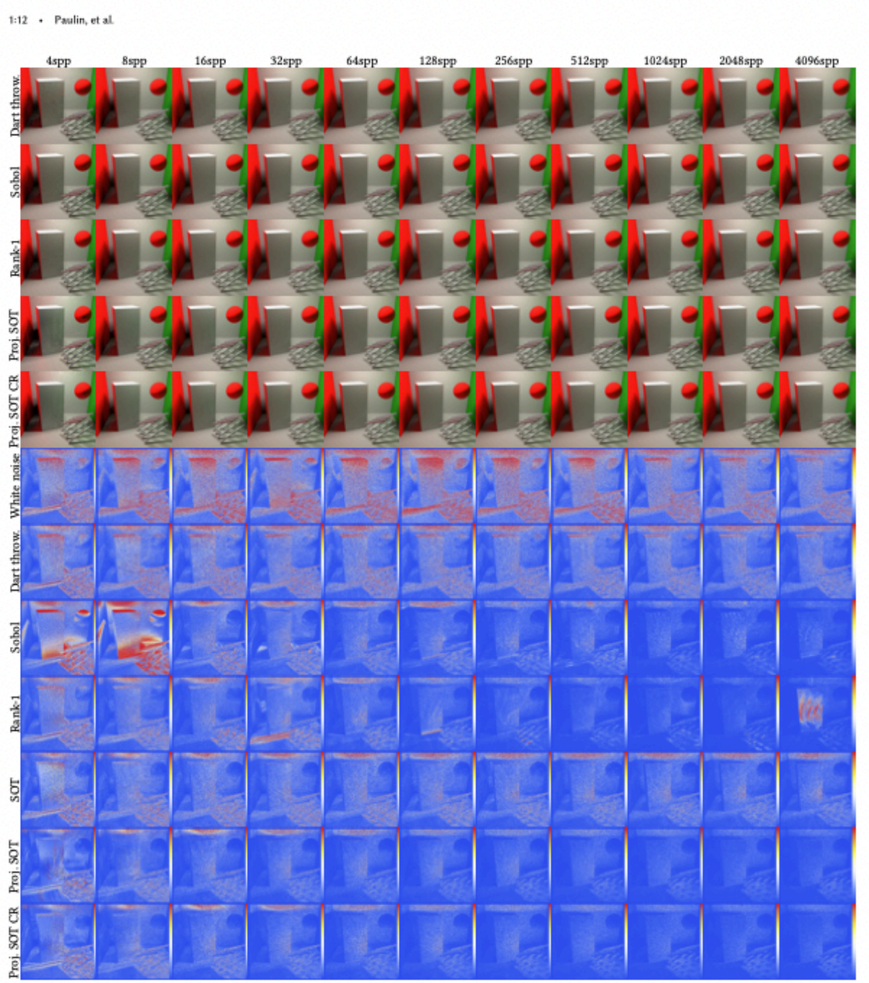
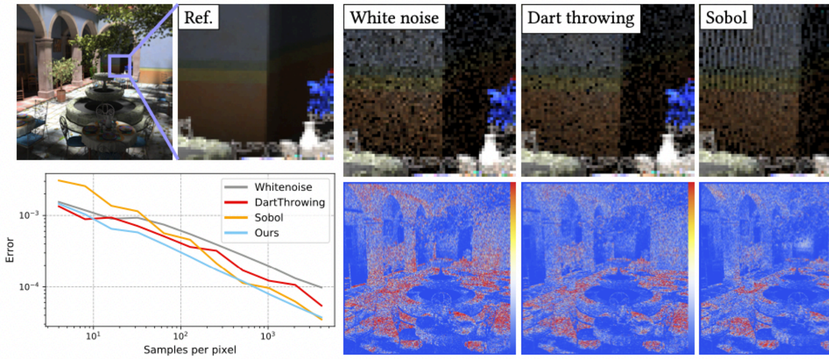




**Fig. 13. Rendering results in dimension 6.** We render the Cornell scene with one bounce of indirect lighthing using various samplers and for several sample counts. To highlight the efficiency of Monte Carlo integrations, we display the error maps (per-pixel absolute difference with reference image in Fig. 11) using pseudocolors (from blue to red, with a linear map from 0 to 0.002/spp). While SOT sampling does not help much for rendering, its projective variant is better and/or more consistent than other typical approaches, particularly so for low counts of samples per pixel.

ACM Trans. Graph., Vol. 39, No. 4, Article 1. Publication date: July 2020.

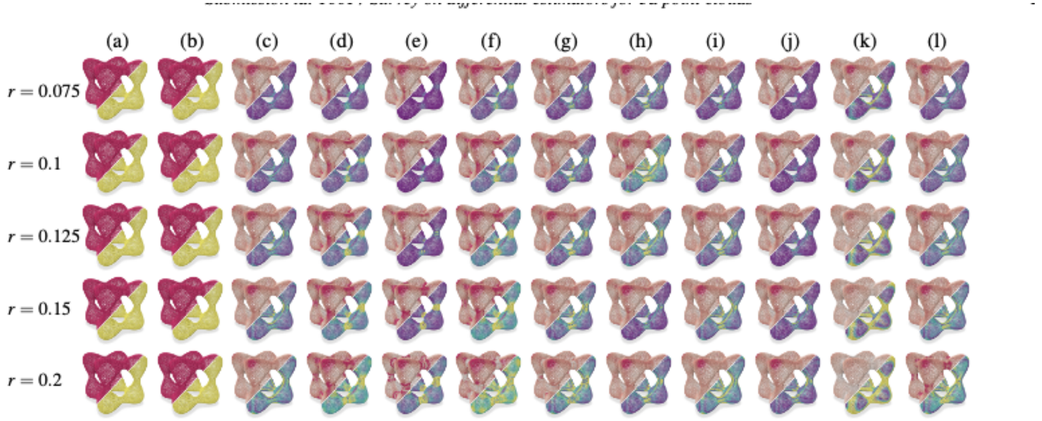



●●●

# Latex tweaks

## \begin{overpic} 💪

```
\setlength{\tabcolsep}{0pt}
\begin{tabular}{ c c c c}
 .....
\end{tabular}
```



# Supplementary material

# Web / Comparaison

- https://replicability.graphics
- https://storm-irit.github.io/pcednet-supp/
- https://perso.liris.cnrs.fr/lpaulin/Publications/paulin2020_demo/
- https://github.com/nivoliev/html-compare
- https://nivoliev.github.io/html-compare/

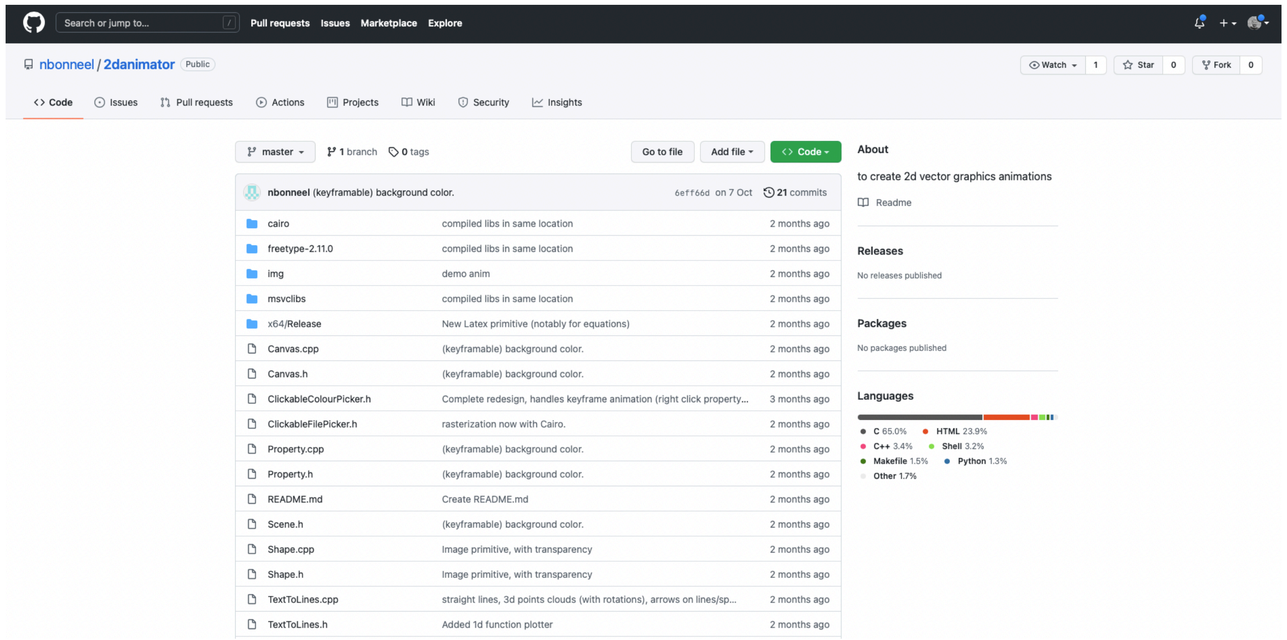
# Présentation / restitution

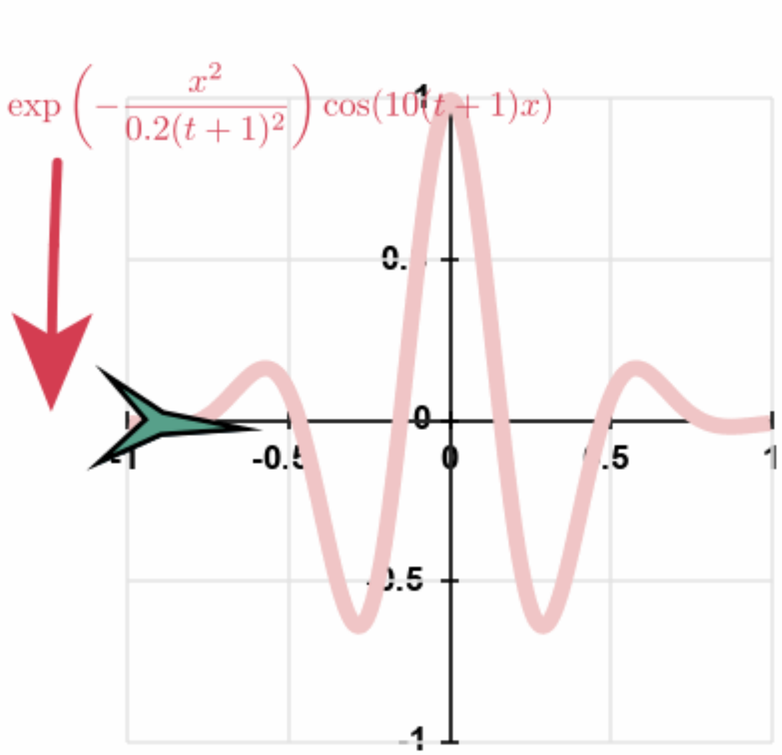
# Présentations

- Beamer/latex
- Office, keynote, powerpoint…
- HTML (3d, js,…)

# Animations

- 2danimator (https://github.com/nbonneel/2danimator/ )

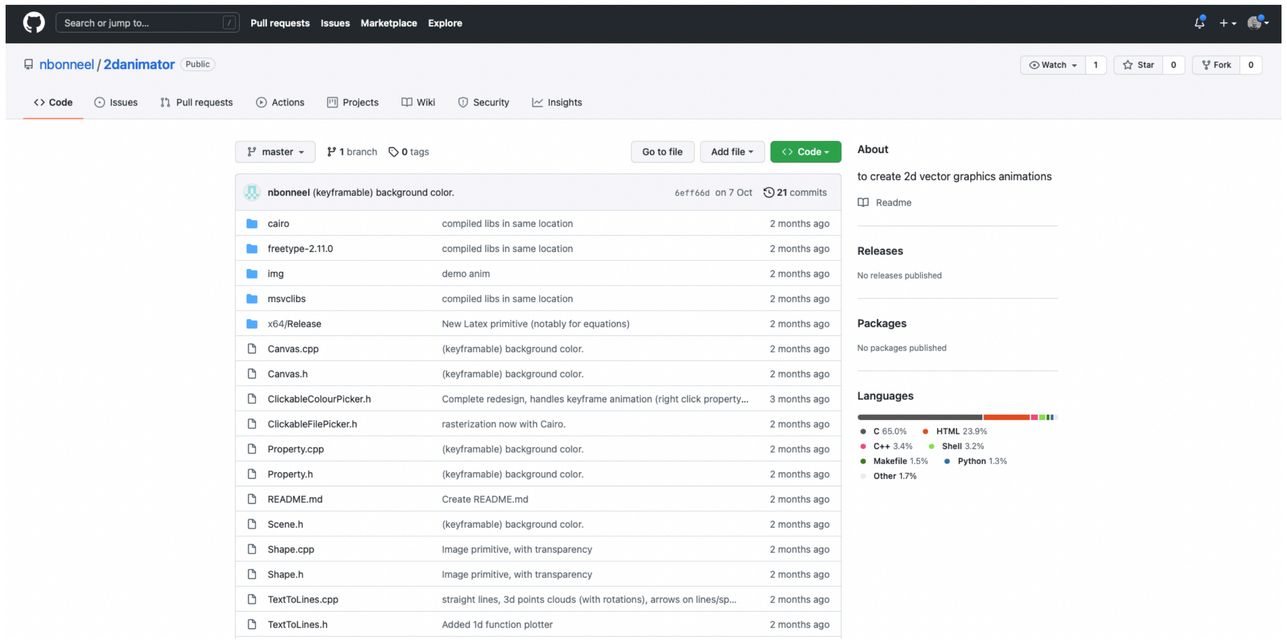


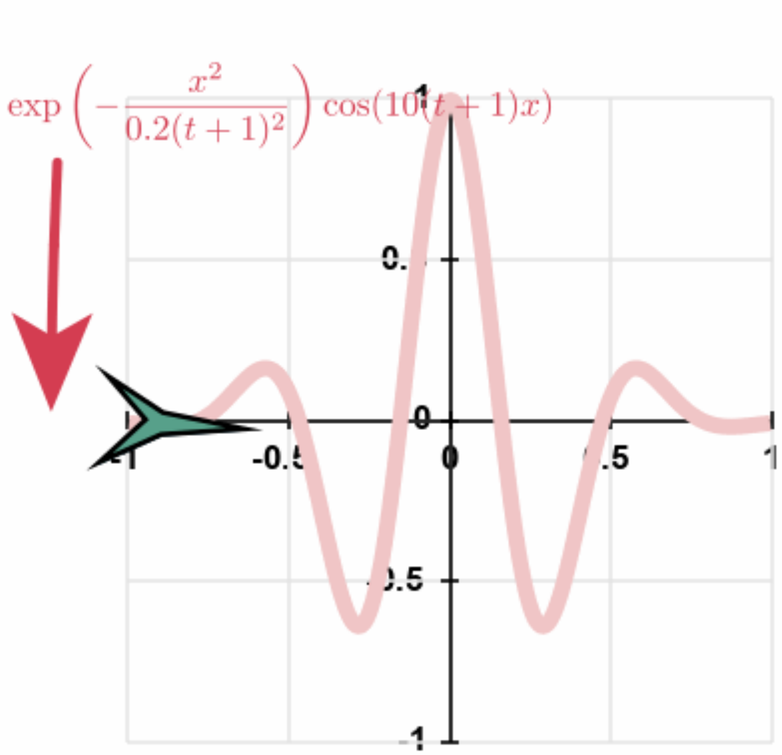






# Animations

- 2danimator (https://github.com/nbonneel/2danimator/ )






# Animations

- Manim: https://www.manim.community



- https://www.3blue1brown.com

```python
class SquareToCircle(Scene):
    def construct(self):
        circle = Circle()
        square = Square()
        square.flip(RIGHT)
        square.rotate(-3 * TAU / 8)
        circle.set_fill(PINK, opacity=0.5)

        self.play(Create(square))
        self.play(Transform(square, circle))
        self.play(FadeOut(square))
```



```python
class SquareToCircle(Scene):
    def construct(self):
        circle = Circle()
        square = Square()
        square.flip(RIGHT)
        square.rotate(-3 * TAU / 8)
        circle.set_fill(PINK, opacity=0.5)

        self.play(Create(square))
        self.play(Transform(square, circle))
        self.play(FadeOut(square))
```



```python
class SquareToCircle(Scene):
    def construct(self):
        circle = Circle()
        square = Square()
        square.flip(RIGHT)
        square.rotate(-3 * TAU / 8)
        circle.set_fill(PINK, opacity=0.5)

        self.play(Create(square))
        self.play(Transform(square, circle))
        self.play(FadeOut(square))
```



```python
class SquareToCircle(Scene):
    def construct(self):
        circle = Circle()
        square = Square()
        square.flip(RIGHT)
        square.rotate(-3 * TAU / 8)
        circle.set_fill(PINK, opacity=0.5)

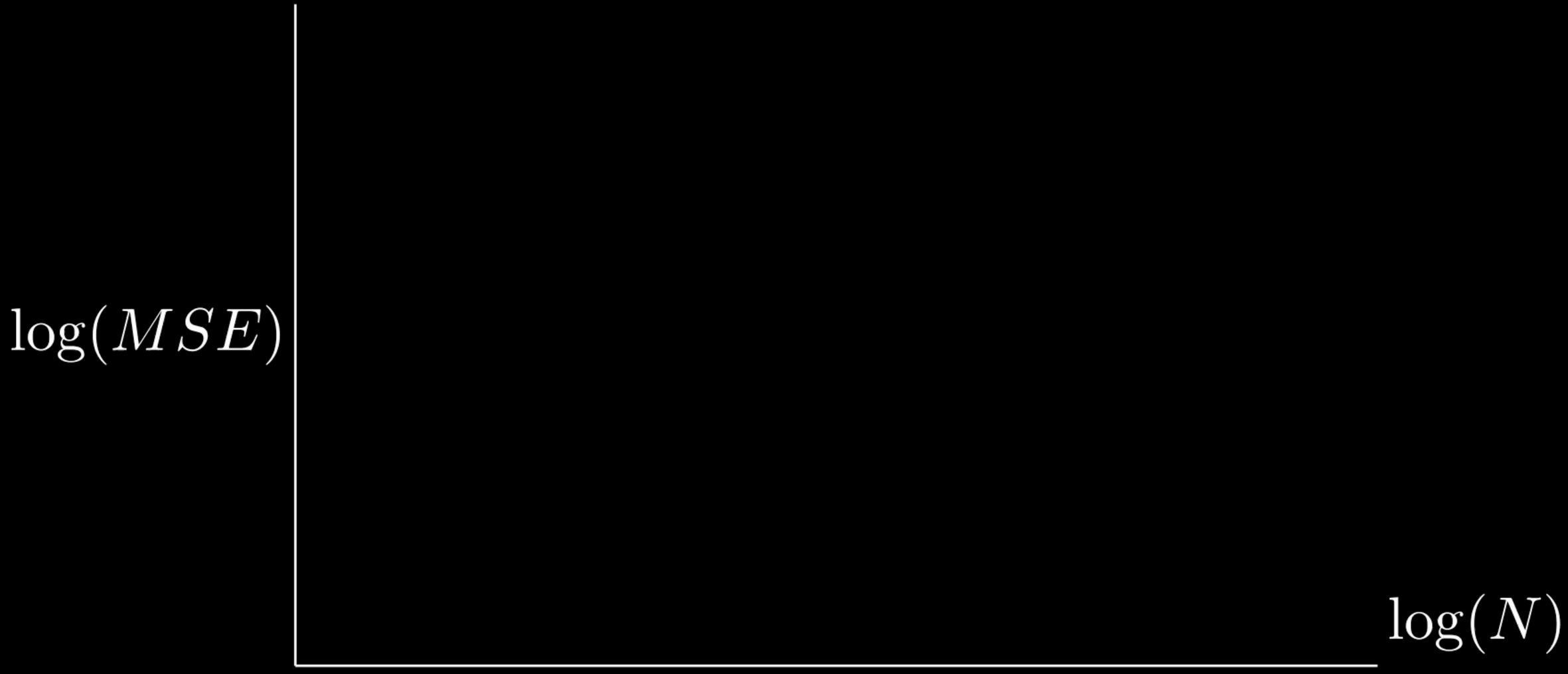
        self.play(Create(square))
        self.play(Transform(square, circle))
        self.play(FadeOut(square))
```

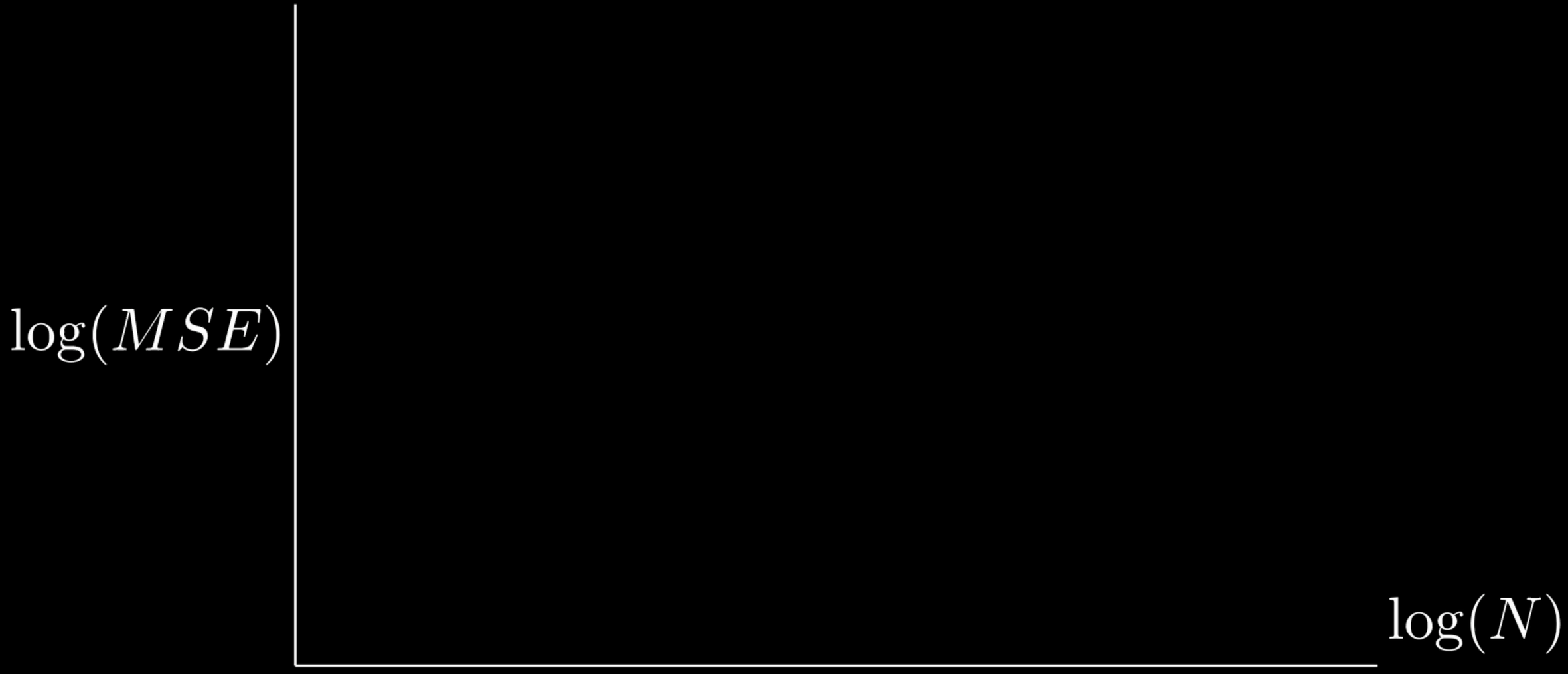


```python
class SquareToCircle(Scene):
    def construct(self):
        circle = Circle()
        square = Square()
        square.flip(RIGHT)
        square.rotate(-3 * TAU / 8)
        circle.set_fill(PINK, opacity=0.5)

        self.play(Create(square))
        self.play(Transform(square, circle))
        self.play(FadeOut(square))
```




$\log(MSE)$
$\log(N)$


$\log(MSE)$
$\log(N)$

# Wrap-up

Anticipez

Pratiquez

Posez vous la question « quelle est la fonction de mon illustration ? »

Faites-vous aider