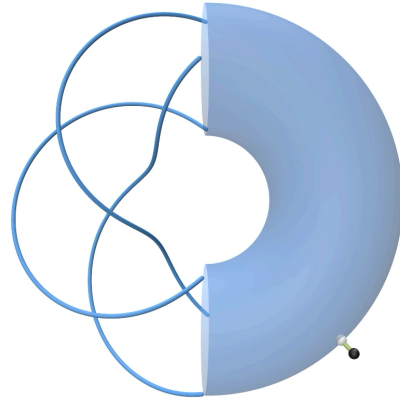


Closest Point Geometry Processing

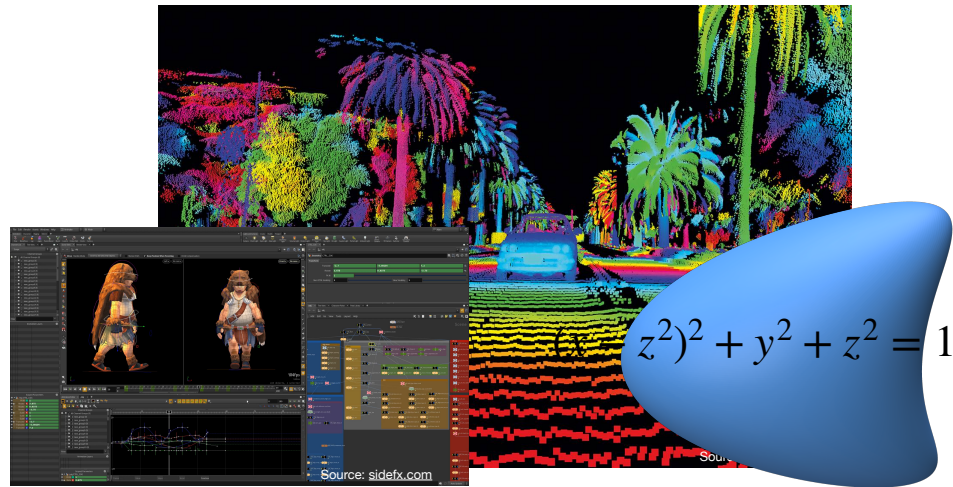
Nathan King

SGP Graduate School 2026

June 30, 2026

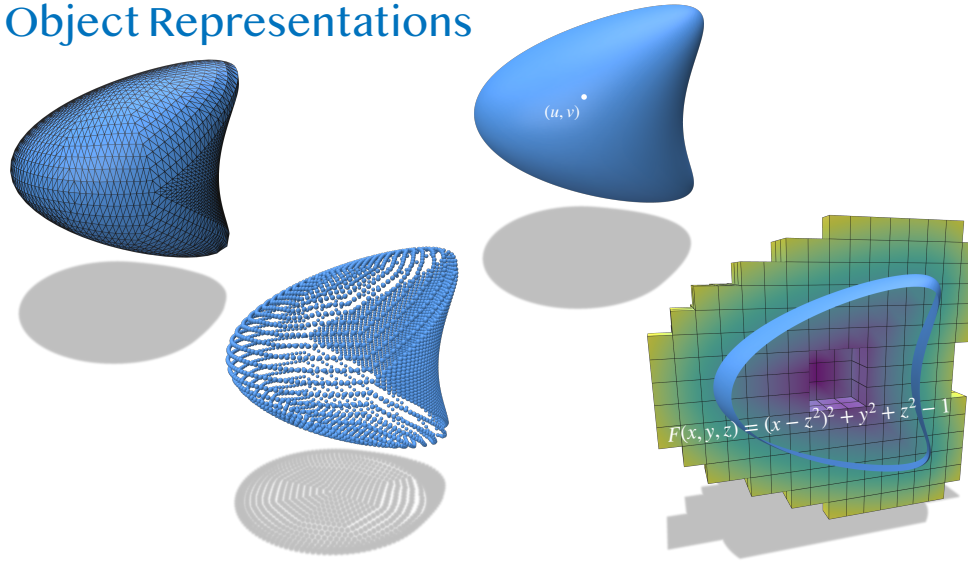


Digital Object Creation

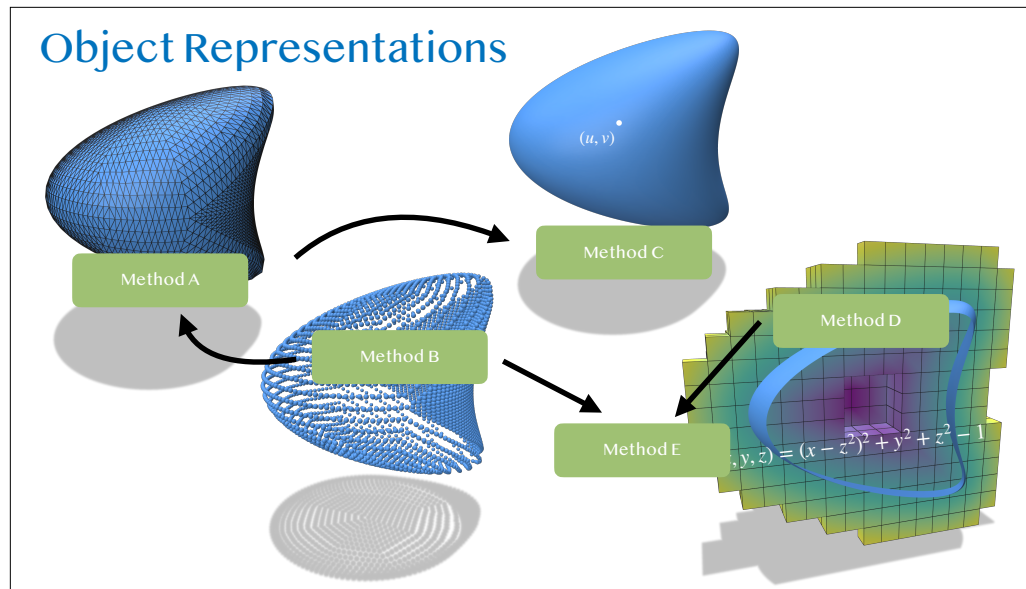


- (0:27) Digital objects can be created in many different ways, for example LiDAR sensors can be used to create the digital world surrounding a self-driving car (next)
- Researchers, engineers, and artist can use software like Houdini, Shapr3D, or Blender, to manually create an object (next)
- Also, functional descriptions of objects can be used, for example, all the points that satisfy this equation (next) give this surface

Object Representations

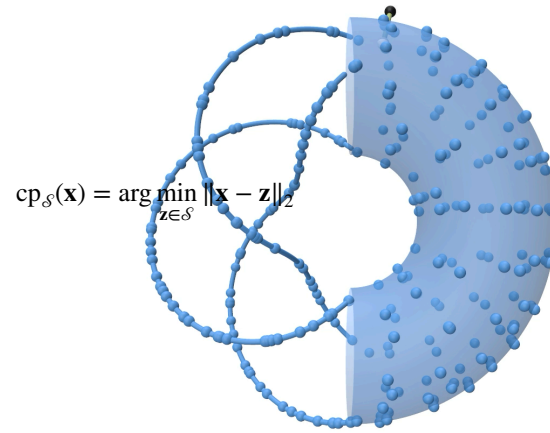


- (0:36) With the numerous object creation techniques come just as many ways to represent the object on a computer. (next)
- For example, as a mesh, (next) a point cloud (next), a parameterization (next), or a level set.
- Algorithms specialized to a single representation have the advantage of using specific properties of that single representation
- For example, signed distance fields are great for collision detection. Parameterizations are convenient for placing a texture image on a surface. And GPU hardware has been developed with meshes at the forefront



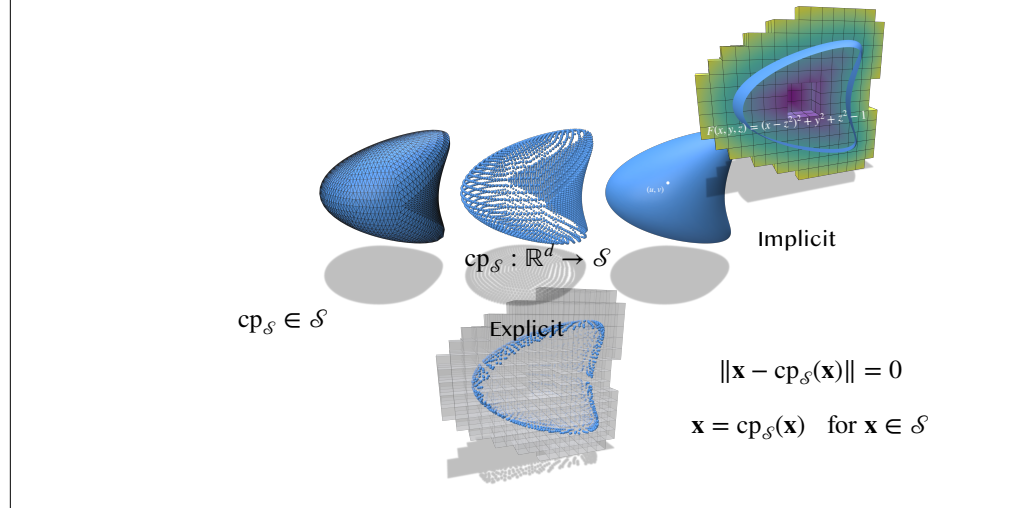
- (1:00) However, in large pipelines, it is common to have to transfer between different representations for different algorithms, which results in conversion errors that accumulate in the pipeline. It also costs computational time to do the conversion.
- For example, you could use a sensor to capture an object as a point cloud (next), convert to a mesh to do some surface stylization transfer, (next) then finally need to convert to a parameterization to apply a texture on the surface.
- You could also have pipelines (next) that combine multiple objects of different representations. For example, one object uses a point cloud and the other is a signed distance field and the result needs (next) to be combined together in some way for a physics simulation in Method E.
- In this case, no method that is specialized to a specific representation would work without conversion
- So it would be great if we had a single, unified framework that handles all desired applications and all object representations

Closest Points



- (0:45) Closest points provide a good candidate for this unified framework.
- Given a query point x in the space surrounding an object S , its closest point is the point on S that has the minimum Euclidean distance to x . (Next) Here are some closest points of the black query points for this surface.
- A closest point representation is nice because it handles general objects. Closest points can represent manifolds that are closed and orientable like this surface (next), or open and unorientable like this Mobius strip (next), or even mixed codimensional like these 1D curves attached to the 2D surface, all embedded in 3D.

Explicit vs Implicit



- (1:45) Closest points also provide a representation that is both explicit and implicit, which is quite a distinct property
- Other representations are one or the other (next)
- Meshes, point clouds, and parameterizations are all explicit representations because they directly specify points of the manifold
- Level sets or signed distance fields provide an implicit description of the manifold, here the implicit equation F of x, y, z equals 0 gives points on the manifold
- The closest point mapping (next) from points in the embedding space $\mathbb{R}^d$ to points on S represents S implicitly (next). A traditional scalar implicit manifold can be recovered as the zero-level-set (next) of the distance function, although it is unsigned.
- A vector-valued implicit representation of S (next) is also available since we know that for any x on S , x must be its own closest point
- Meanwhile, the closest points themselves (next) give an explicit representation of S as a point cloud
- The ability for closest points to represent many different manifold types and the fact they are both explicit and implicit, gives closest point representations a lot of potential to provide a unified framework
- But because there are so many applications in computer graphics and other fields, we decided to concentrate on geometry processing applications involving partial differential equations or PDEs

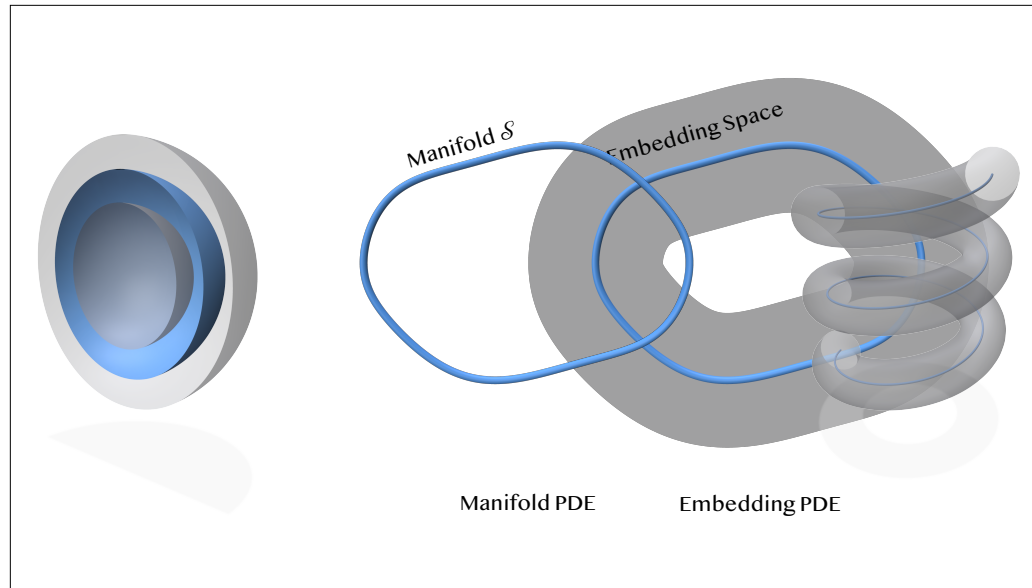
- Conveniently, there already existed a method to solve PDEs with closest point representations, (next) called the closest point method

★ Lack of interior boundary conditions

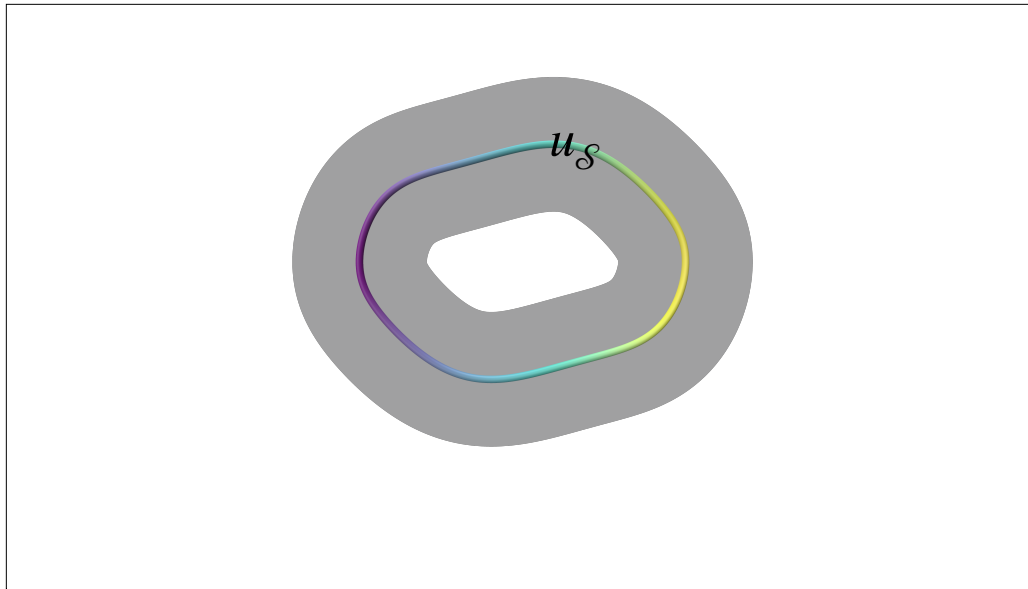
Continuous Setting Closest Point Method

- Discrete Setting
- Boundary Conditions & Geometry processing
- Efficiency
- Future Work

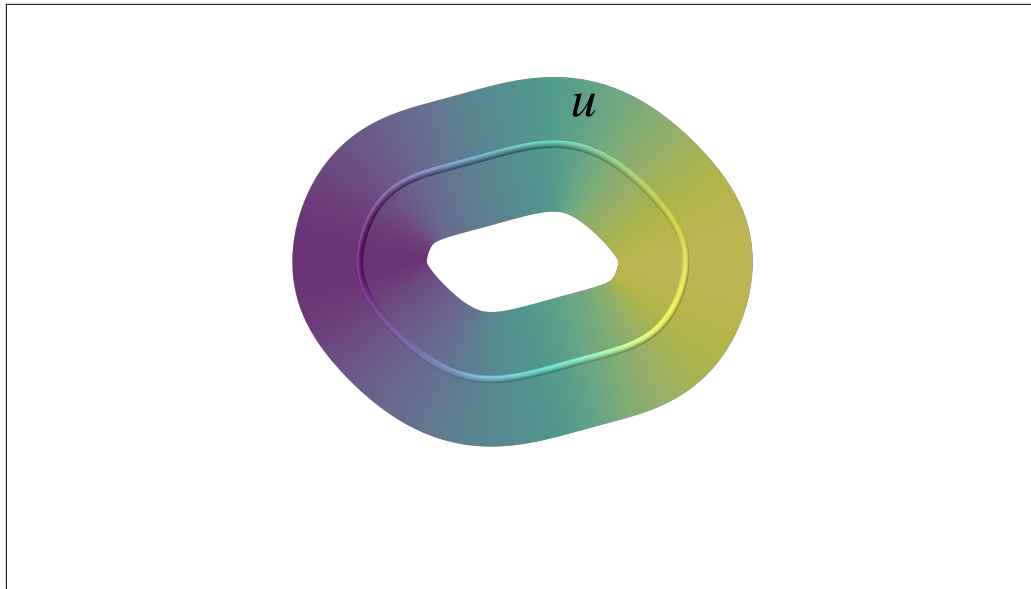
- (1:00) However, there was an important limitation of the closest point method that stood in our way to be able to apply it in many geometry processing applications (next)
- Interior boundary conditions couldn't be handled, which are boundary conditions defined away from the physical boundary of an open manifold. Interior boundary conditions are extremely important for many geometry processing algorithms, like diffusion curves or geodesic distance computation
- So introducing a way to handle interior boundary conditions will be the core of this course. (next)
- But overall this course will cover the following.
- I will start with a review of the closest point method in the continuous setting, then (next) discuss the common way to approximate the solution in the discrete setting
- I will then discuss boundary conditions and some geometry processing applications that the interior boundary condition framework unlocks (next)
- Then I will talk about some ways to improve the efficiency of the closest point method and (next) conclude with a discussion of future research directions



- (0:33) Given some manifold $\mathcal{S}$ and a PDE you would like to solve (next), the problem is extended to the surrounding embedding space. An embedding PDE is constructed on this Cartesian space, whose solution agrees with the manifold PDE when restricted to the manifold. (Next)
- Most of my diagrams will use 1D curves embedded in 2D like this example, (next) but remember everything applies also to 2D surfaces embedded in 3D
- (Next) Or higher codimensional objects like a 1D curve embedded in 3D



- (0:13) First some important notation. Manifold functions are denoted as u_S and they get extended onto the Cartesian embedding space to be constant in the normal direction of the manifold (next), which is denoted u .



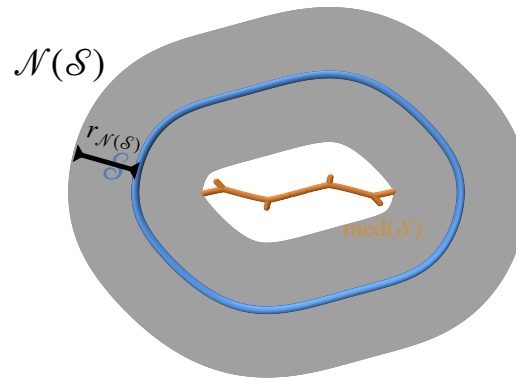
- (0:05)

Properties

1. Constant normal extension via composition $u(\mathbf{x}) = u_{\mathcal{S}}(\text{cp}_{\mathcal{S}}(\mathbf{x}))$
2. If v is constant along normal directions of $\mathcal{S}$
 $\Rightarrow \nabla_{\mathcal{S}} v_{\mathcal{S}} = \nabla v \quad \text{on } \mathcal{S}$
3. If $\mathbf{w}$ is tangent to $\mathcal{S}$ and tangent to all level-sets of $\mathcal{S}$
 $\Rightarrow \nabla_{\mathcal{S}} \cdot \mathbf{w}_{\mathcal{S}} = \nabla \cdot \mathbf{w} \quad \text{on } \mathcal{S}$

- (1:20) Then to construct the embedding PDE the closest point method relies on 3 main properties. (Next)
- First, the constant normal extension of manifold functions $u_{\mathcal{S}}$ is done by composing with the closest point function. Intuitively, this just assigns the value of $u_{\mathcal{S}}$ at the closest point of $\mathbf{x}$ to $\mathbf{x}$ itself. So we call this the closest point extension. (Next)
- Now, if you have a function v in the embedding space that is constant along the normal directions of $\mathcal{S}$, then the Cartesian gradient is equivalent to the manifold gradient at points on $\mathcal{S}$. Because v only changes in the tangential direction of $\mathcal{S}$. (Next)
- Also, if you have a vector field $\mathbf{w}$ that is tangent to $\mathcal{S}$ and tangent to all the level-sets of $\mathcal{S}$, then the Cartesian divergence is equal to the manifold divergence at points on $\mathcal{S}$. Because the vector field is constrained to the tangent space of $\mathcal{S}$, it can only spread out (or compress) within that tangent space.
- You can combine these properties to rewrite higher-order manifold differential operators with Cartesian differential operators. (Next)

Local Feature Size



$$\mathcal{N}(\mathcal{S}) = \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - \text{cp}_{\mathcal{S}}(\mathbf{x})\| \leq r_{\mathcal{N}(\mathcal{S})}\}$$

- (1:08) These properties hold when you have a unique closest point function
- For general S , the closest point function is rarely unique for all x in the embedding space $\mathbb{R}^d$
- The medial axis (next) of S is the subset of the embedding space that has at least two closest points on S . In other words, where the closest point function is not unique
- For smooth and compact manifolds, the closest point function is always unique for x in a (next) tubular neighbourhood $\mathcal{N}(S)$ that avoids the medial axis using a sufficiently small (next) tube radius $r_{\mathcal{N}(S)}$
- The size of $r_{\mathcal{N}(S)}$ depends on the curvatures and bottlenecks of the manifold. Bottlenecks are thin regions where the Euclidean distance is small between geodesically distant parts of S
- (Next) So the definition of $\mathcal{N}(S)$ is all the points x in the embedding space $\mathbb{R}^d$ that are at most $r_{\mathcal{N}(S)}$ away from their closest point

Laplace-Beltrami Operator

$$\Delta_{\mathcal{S}} u_{\mathcal{S}} = \nabla_{\mathcal{S}} \cdot (\nabla_{\mathcal{S}} u_{\mathcal{S}})$$

- If $u_{\mathcal{S}} \in \mathcal{S}$ then $u = u_{\mathcal{S}}(\text{cp}_{\mathcal{S}})$ is constant along normals of $\mathcal{S}$ (P1)
- Thus $\nabla_{\mathcal{S}} u_{\mathcal{S}} = \nabla u$ on $\mathcal{S}$ (P2)
- Since ∇u is tangent to level-sets of $\mathcal{S}$, we have
 $\nabla_{\mathcal{S}} \cdot (\nabla_{\mathcal{S}} u_{\mathcal{S}}) = \nabla \cdot (\nabla u)$ on $\mathcal{S}$ (P3)

$$\Delta_{\mathcal{S}} u_{\mathcal{S}} \xrightarrow{\quad} \Delta u(\text{cp}_{\mathcal{S}})$$

- (0:52) Let's go through an example of how the 3 properties are used for the Laplace-Beltrami operator (next)
- So you have a surface function $u_{\mathcal{S}}$, then applying the closest point extension you get u that is constant in the normal direction of $\mathcal{S}$, by property 1 (next)
- So the Cartesian gradient of u is equal to the manifold gradient of $u_{\mathcal{S}}$ on the manifold by property 2 (next)
- Now the gradient of u is a vector field that is tangent to all the level-sets of $\mathcal{S}$, because u was constant in the normal direction its gradient is tangent to all the offset manifolds, so we can apply property 3 (next)
- So the Laplace-Beltrami operator is replaced with just the Laplacian of u of cp

Other PDEs

- Heat Equation
- Wave Equation
- Navier-Stokes Equation
- Cahn-Hilliard Equation
- Hamilton-Jacobi Equation
- Reaction-Diffusion
- Advection-Diffusion
- Eigenvalue Problems
- Harmonic Maps
- And more...

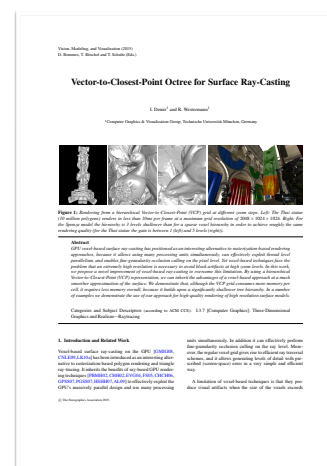
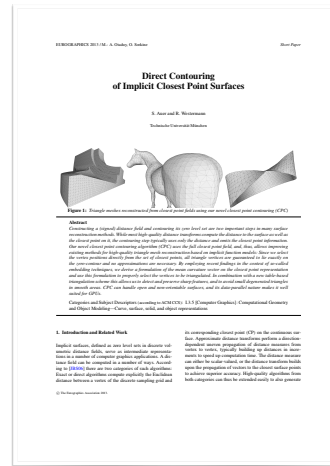
- (0:24) These principles have been successfully used to solve all these different PDEs and many more
- I will focus on PDEs involving the Laplace-Beltrami operator because it is the most commonly used operator in geometry processing
- But there has been some other work in computer graphics involving the closest point method and just closest point representations alone

Fluid Simulation



- (0:38) First off Auer et al. solved the wave equation and Navier-Stokes equation on static surfaces
- Then Auer and Westermann extended this work to deforming surfaces
- Kim et al. did something different and increased the apparent resolution of an existing, coarse volumetric fluid simulation by solving a wave equation on the surface
- And most recently Morgenroth et al. also coupled a surface PDE with a volumetric fluid simulation but this time to simulate oil film on the surface of water

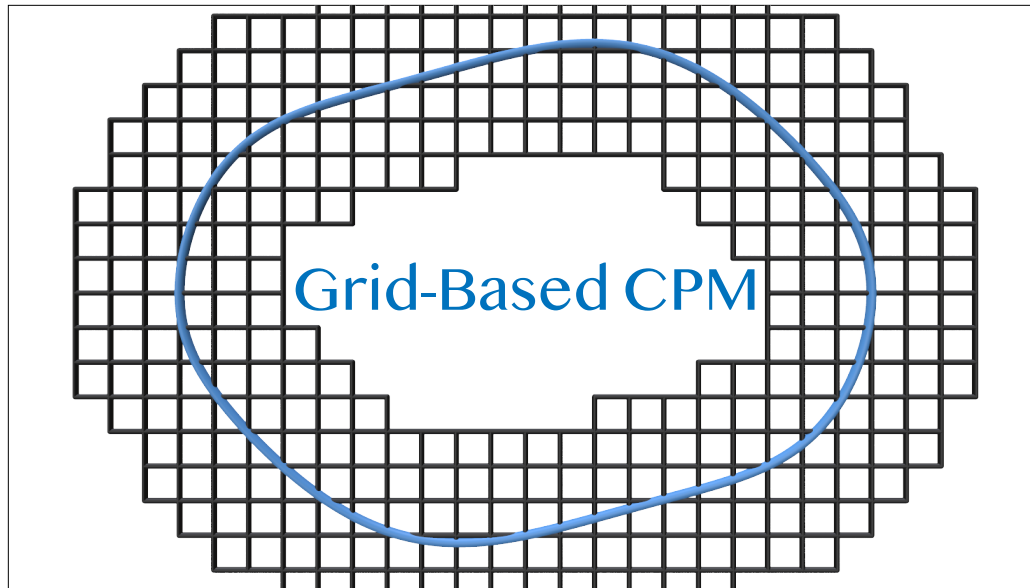
Visualization



- (0:21) There has also been work on visualizing closest point representations
- Auer and Westermann developed a method similar to marching cubes to create triangulations
- It is able to better preserve sharp features and handle open and unorientable surfaces
- Demir and Westermann also showed how to directly visualize closest point representations using ray-casting

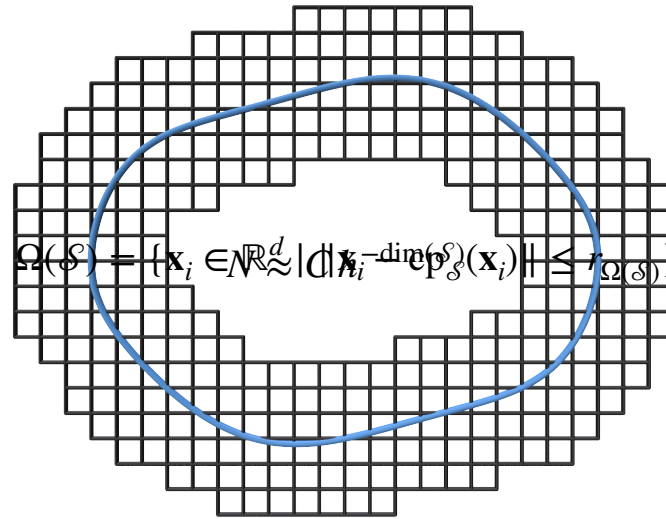
[illegible]

- (0:23) There has actually be a couple applications in geometry processing that measure the similarities between different shapes in a dataset
- But these applications only needed global information about the shapes
- The ability to handle local conditions that I will discuss later is what opened the door to numerous geometry processing applications



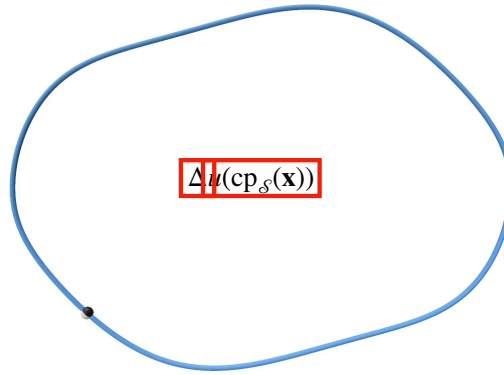
- (0:46) Up to now we just discussed the continuous theory and the generality of the types PDEs it can be applied to
- Most papers use a grid surrounding the manifold to implement the closest point method discretely on a computer
- I will focus on the most common grid-based approach in this course that uses finite-differences and Lagrange interpolation
- These approximations are usually chosen for their simplicity and since they have been well-studied
- However, the continuous theory puts no restrictions on the type of discretization
- Finite elements and other interpolation schemes can equally be used
- There are even some papers that use radial basis functions and we also recently developed a Monte Carlo based closest point method

Discrete Domain

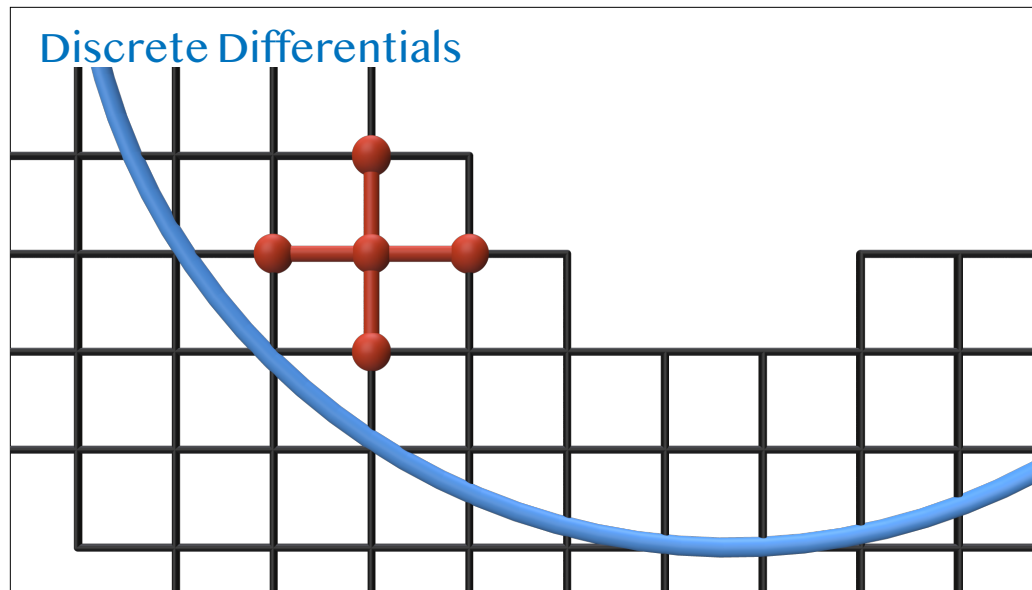


- (0:48) The computational domain is denoted Ω of S
- It is made up of grid points x_i within a tubular neighbourhood of tube-radius $r_{\{\Omega(S)\}}$. (Next) In other words, grid points whose closest points are at most a distance $r_{\{\Omega(S)\}}$ apart.
- One may naively think that using the embedding space means the computation scales with the dimension of that space. However, (next) the number of grid points N actually scales with the dimension of the manifold $\dim(S)$.
- This is because the tube radius $r_{\{\Omega(S)\}}$ is finite
- I will discuss later how this radius is calculated, but with this finite radius, you can use Weyl's tube formula to compute the volume of the tube and then approximate the number of grid points from the volume

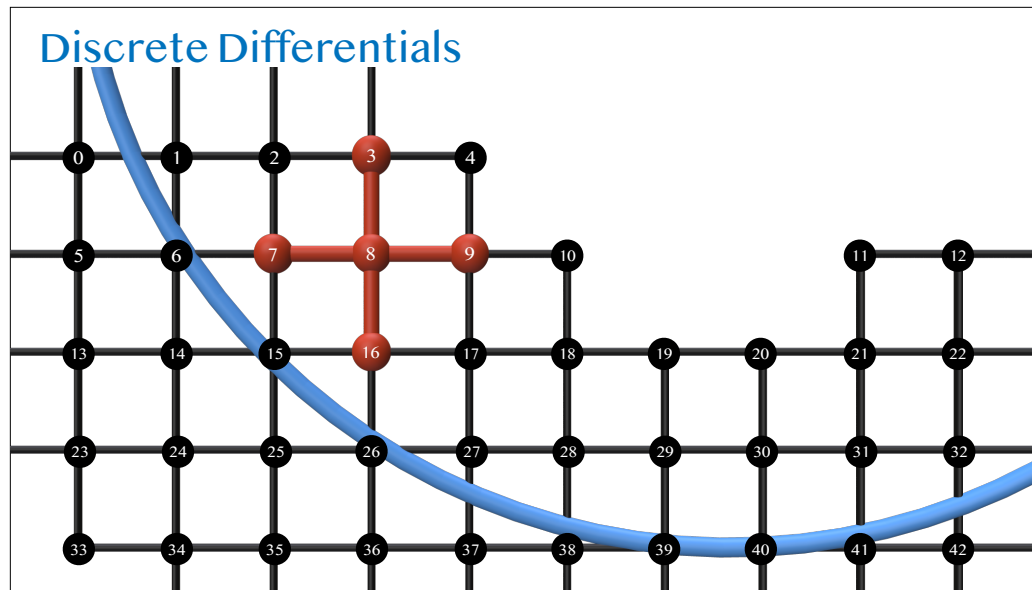
Breadth-First Search



- (0:30) To construct the grid with this optimal scaling we use a breadth-first search so closest points are only computed within the tubular neighbourhood. Memory is allocated as needed using a sparse-grid data structure. (Next at end of animation)
- With this tubular computational domain, we now need to approximate the embedding PDE. This involves approximating the (next) differential operators like the Laplacian, and (next) the closest point extension. (Next)

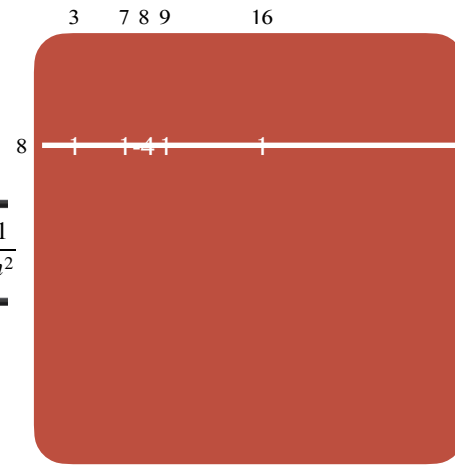
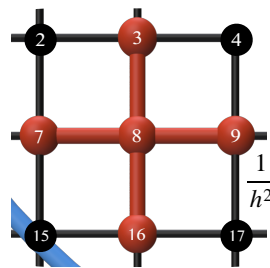


- (0:12) Standard finite differences are used to approximate differential operators on the grid, for example, the red 5-point stencil is used for the Laplacian in 2D



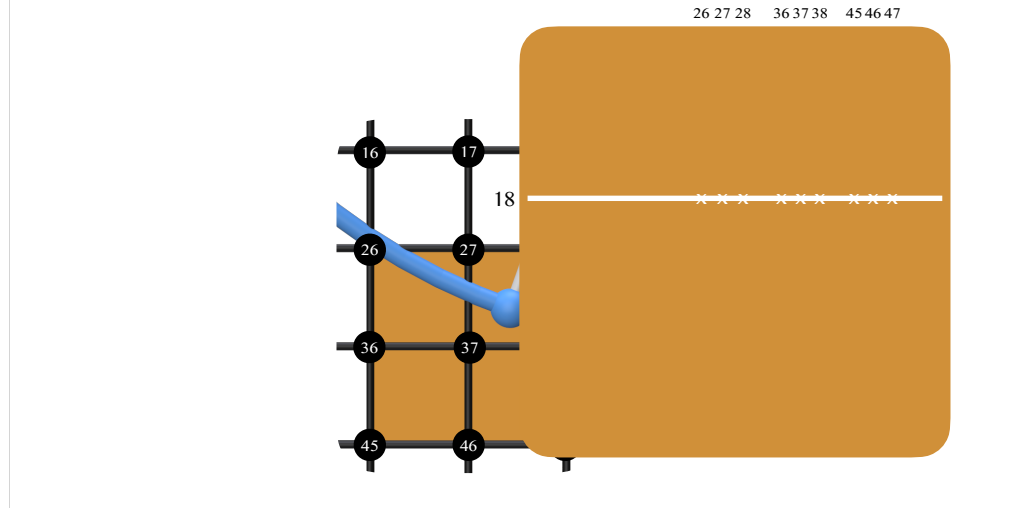
- (0:14) The grid points are all assigned indices and a matrix can be built that holds the weights of the stencils
- This red stencil is for grid point 8

Discrete Differentials



- (0:30) In the row of the matrix for grid point 8, we insert the stencil weights in columns (next) that are part of the stencil
- For the Laplacian in 2D, (next) there is a $1/h^2$ factor for all the weights. (Next)
- Then the weight of the grid point itself in column 8 is minus 4 (next)
- All other stencil weights are 1
- You do this for all grid points to fill out all the rows of the matrix

Closest Point Extension



- (0:52) The closest point extension involves interpolation since closest points are generally not at grid points.
- For grid point 18, the closest point extension assigns the function value at the blue closest point. However, function values are only stored at grid points, so values from the orange interpolation stencil are used to approximate the function at the blue closest point. (Next)
- Similar to the construction of the finite-difference matrix, the rows for each grid point have non-zeros inserted into them (next)
- For grid point 18, we have these columns (next) that are part of the stencil
- Unlike the finite-difference weights, the interpolation weights are different for each stencil since they depend on the locations of the closest points

Matrix Form

$$\Delta u(\text{cp}_{\mathcal{S}}) \approx \begin{array}{|c|} \hline \mathbf{L} \\ \hline \end{array} \begin{array}{|c|} \hline \mathbf{E} \\ \hline \end{array} \begin{array}{|c|} \hline u \\ \hline \end{array}$$

- (0:24) Once we build the finite-difference and interpolation matrices we can approximate the Laplace-Beltrami operator as (next) the finite-difference matrix $\mathbf{L}$, (next) times the extension matrix $\mathbf{E}$, (next) times the function values at grid points u
- You approximate all manifold differential operators in a similar way to form the matrix equation for the PDE you want to solve

Heat Equation

$$u^{i+1} = u^i + \Delta t \left(\frac{\partial u}{\partial t} \right)^i = L u^i + E u^i$$

- (1:05) Let's consider the heat equation as an example (next)
- You construct the embedding PDE like we discussed before (next)
- The manifold differential operators are the spatial component of the PDE and is what the closest point method provides an approximation for (next) as L times E times u (next)
- There is also the time derivative in the heat equation and standard time-stepping methods are used to approximate it
- I won't review time-stepping methods in this course, but consider using an explicit time-stepping method like forward Euler method (next)
- So to get the solution at the next time step, $t+1$, you take the current solution at t and add the Laplace-Beltrami approximation at time t also, and scale by the size of the time step (next)
- Pictorially, it looks like this and just involves multiplying matrices and adding vectors to compute the future solution

Heat Equation

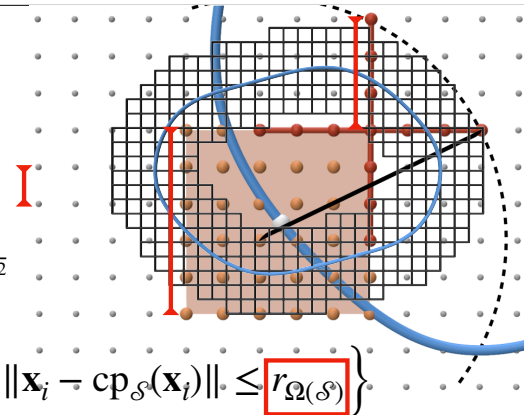
The diagram illustrates the matrix equation for backward Euler time-stepping. It shows a large left parenthesis followed by a vertical black bar containing u^{i+1} , then $= \Delta t$, another vertical black bar containing u^i , followed by a blue square labeled A . This is followed by a vertical black bar containing u , an equals sign, a blue vertical bar containing b , and a large right parenthesis. To the right of this parenthesis is a vertical black bar containing u^{i+1} , followed by a vertical black bar containing u^{i+1} (with the u^{i+1} text inside a red square), and finally a vertical black bar containing u^i .

- (1:13) For backward Euler time-stepping the far right (next) u^i becomes u^{i+1}
- So you need to rearrange the equation (next) then solve the matrix equation instead of just the multiplication and addition in the forward Euler case
- One important thing however is that when L and E are used in a matrix equation, there is a small modification that needs to be performed to the matrices first
- I won't discuss the modification details, but there are different ways to understand the reason why and how to derive the modification necessary
- One is that the matrix equation is numerically unstable. This is because the solution of this equation is not guaranteed to be constant in the normal direction to the manifold.
- You need to add an extra term to penalize the solutions constant normal deviation to fix the numerical instability
- This is also the case when solving stationary PDEs, i.e., those that do not depend on time (next)
- In both cases you end up with a matrix equation like this that you solve

Discrete Domain

$$r_{\Omega(\mathcal{S})} \leq h \sqrt{(d-1) \left(\frac{p+1}{2} \right)^2 + \left(\frac{q+1}{2} \right)^2}$$

$$\Omega(\mathcal{S}) = \left\{ \mathbf{x}_i \in \mathbb{R}^d \mid \|\mathbf{x}_i - \text{cp}_{\mathcal{S}}(\mathbf{x}_i)\|_* \leq r_{\Omega(\mathcal{S})} \right\}$$

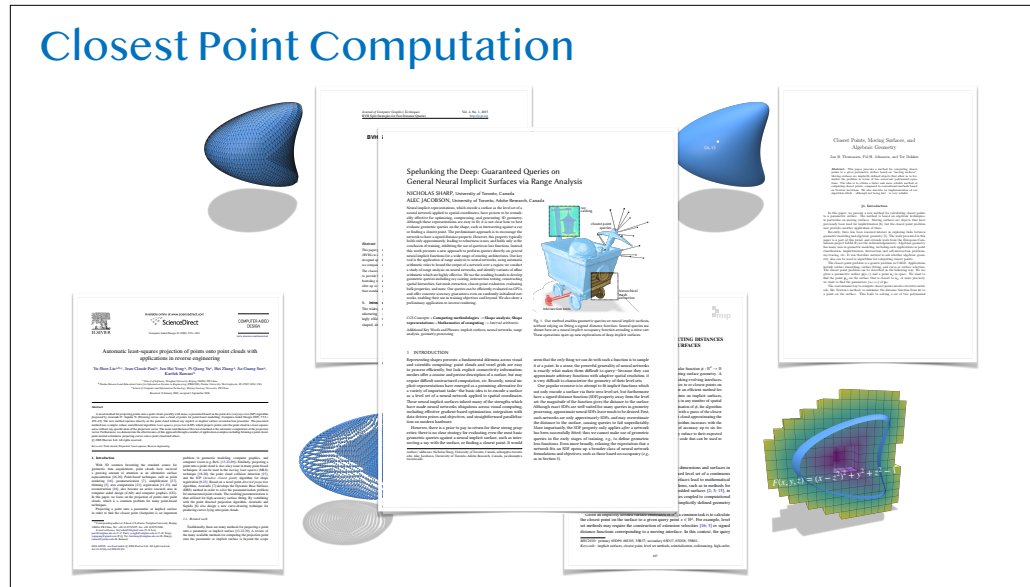


- (2:40) Now that we know we are using finite difference and interpolation stencils, we can derive an upper bound for the computational tube radius $r_{\Omega(\mathcal{S})}$ (next)
- As I previously mentioned, the fact that $r_{\Omega(\mathcal{S})}$ is finite is the reason the computation scales with the dimension of the manifold $\mathcal{S}$ itself and not the embedding space
- If this was not the case, it would make the closest point method too expensive compared to other methods directly on a mesh or a point cloud, for example
- To derive an upper bound on $r_{\Omega(\mathcal{S})}$ we must figure out how far from the manifold a grid point can be in the necessary stencils (next)
- The furthest grid point will be at the tip of a red finite-difference stencil that is centred at one of the corners of the orange interpolation stencil
- The interpolation stencils are chosen to be as symmetric as possible, so the closest point in white will be in the centre cell, but we do not know in general where within that cell
- So to get an upper bound, we look at the distance between the furthest grid point in the centre cell to the tip of the finite difference stencil, so the length of the solid black line
- (Next) working out the geometry you get that $r_{\Omega(\mathcal{S})}$ is upper bounded by this, (next) where h is the grid spacing, (next) d is the dimension of the embedding space so 2 in this example, (next) p is the degree of the interpolation polynomial which is

equal to the number of spaces along one dimension of the stencil, (next) and q is the number of spaces from the centre to the tip of the finite difference stencil

- Note that in most cases the stencils are not this big, they are just shown for illustrative purposes
- The size of the stencils depend on the order of accuracy you want for the solution
- The only restriction is that the order of accuracy for the interpolation needs to be at greater than or equal to the highest order differential operator in the PDE
- So for the Laplace-Beltrami operator, we must have at least 2nd order interpolation, so p must be greater than or equal to 2

Closest Point Computation

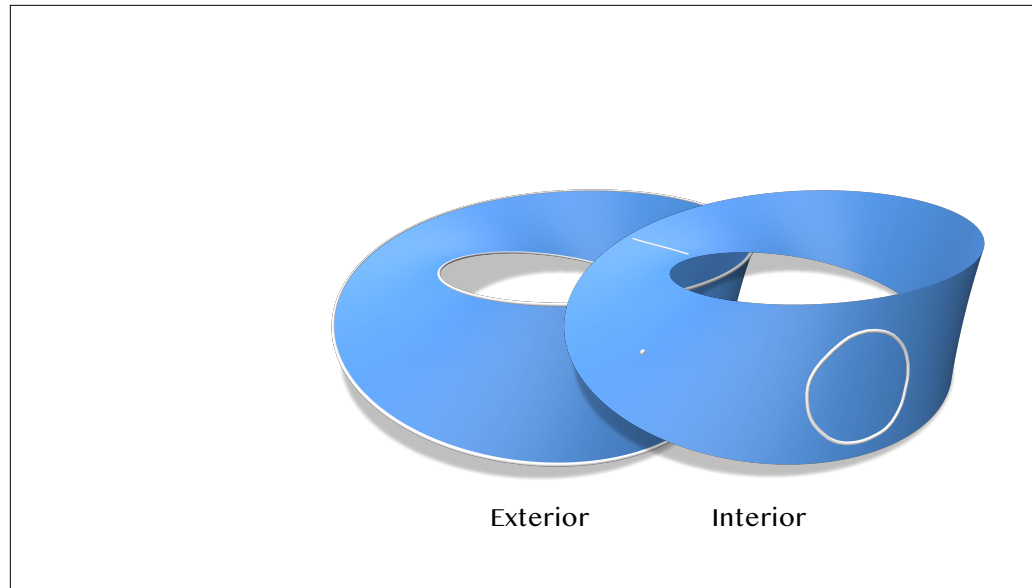


- (0:39) One final detail I did not mention yet is how to actually compute closest points
- There has been lots of work for computing closest points to different surface representations
- Here are some example papers for meshes, point clouds, parameterizations, and level sets
- There has even been work on computing closest points to neural implicits
- So if you have a closed manifold that is everything you need for the closest point method, next I will talk about how to impose boundary conditions
- Are there any questions up to this point? (22 minutes /)

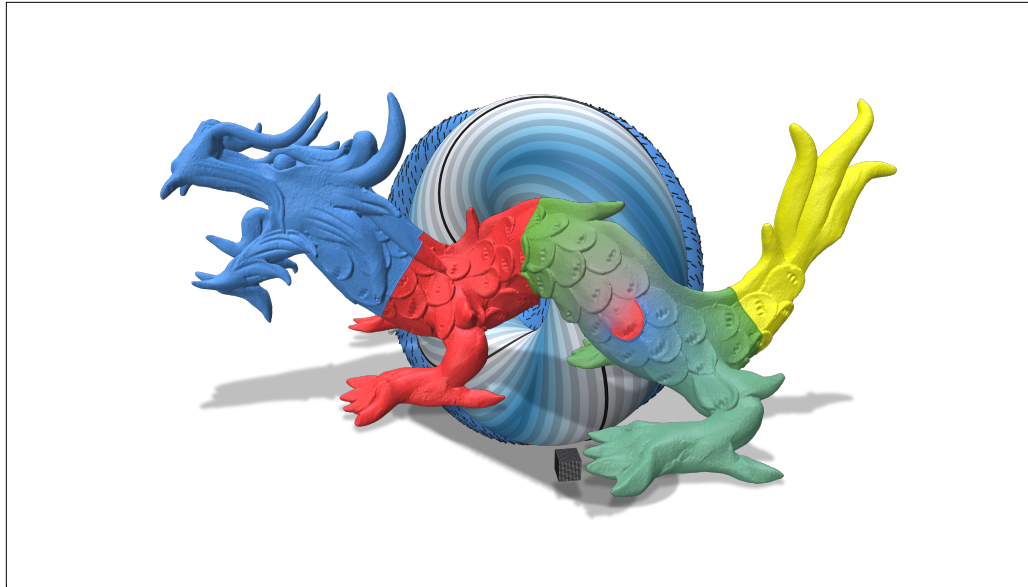
Boundary Conditions



- (0:53) Boundary conditions for PDEs come in many forms (next)
- For example, Dirichlet boundary conditions specify the solution value itself on the boundary as f_S
- So the solution value is known on the white curve of this Möbius strip and solving the PDE obtains the solution over the rest of the surface
- Another common boundary condition is the Neumann boundary condition (next), where the gradient in the conormal direction is specified
- The conormal is not the normal (next) to the manifold S shown here in black, (next) it is the vector orthogonal to both the manifold normal and the tangent direction of the boundary curve, shown in red
- (Next) there is also the Robin boundary condition, where a weighted sum of the Dirichlet and Neumann conditions are specified



- (0:28) The most common way to think about boundary conditions is that they are located on the physical boundary of your domain
- For example on this white curve
- We call these exterior boundary conditions
- But (next) in many problems in geometry processing the boundary conditions can be defined at points or curves away from the exterior, so these are referred to as interior boundary conditions

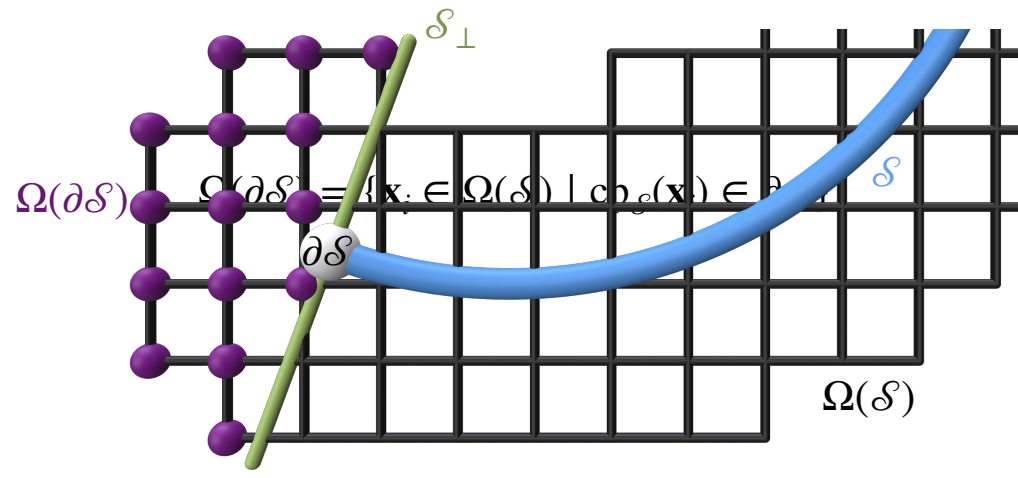


- (0:40) Interior boundary conditions are seen throughout geometry processing in many different applications
- For example, diffusion curves can be used to colour the surface of this dragon
- The colour values are specified on both sides of these interior boundary curves, which are a Dirichlet conditions (next)
- Or you might want to compute the geodesic distance to this black curve
- Or even design a vector field by specifying a sparse set of directions
- Therefore the ability to handle interior boundary conditions is essential to unlock the closest point method for geometry processing

Exterior Boundary Conditions

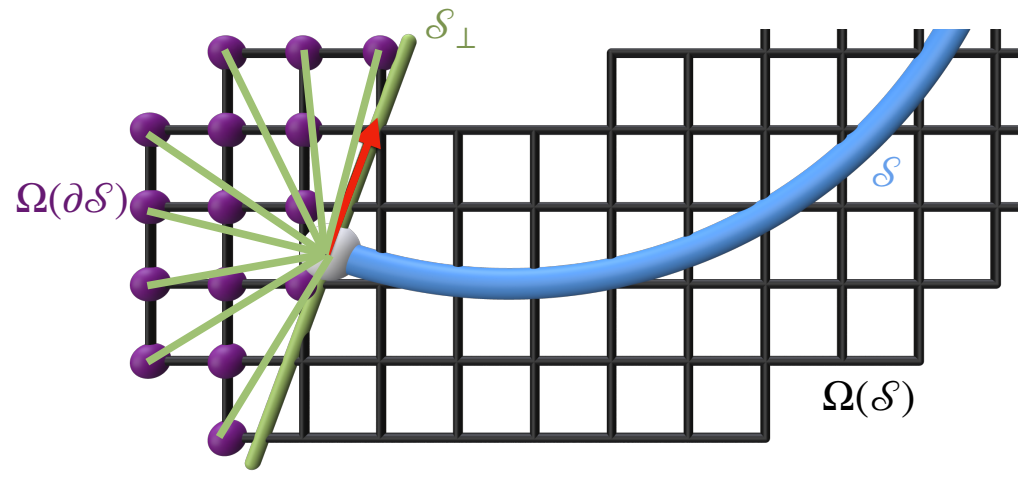
- (0:14) But before we get to interior boundary conditions, we need to understand how exterior boundary conditions are imposed
- Interior boundary conditions will make use of similar modifications, so let's start here

Exterior Boundary Set



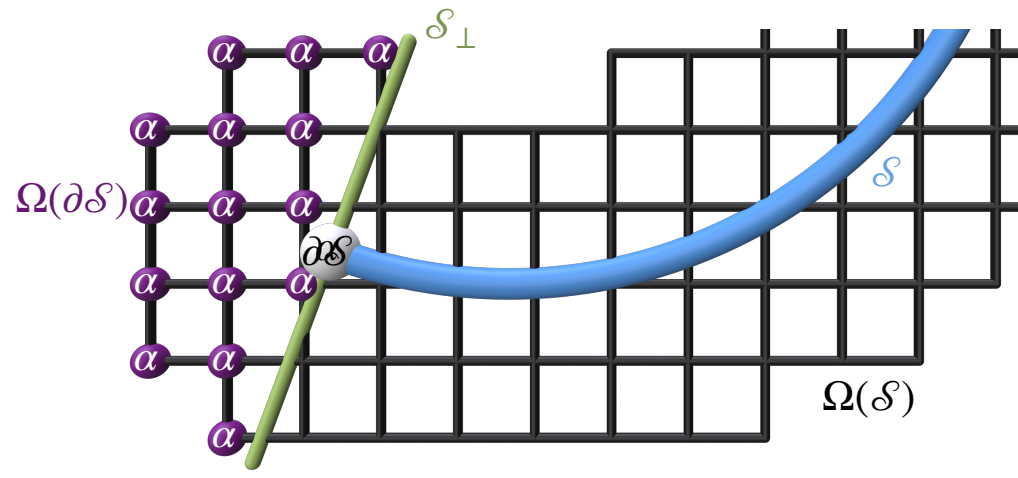
- (0:28) The exterior boundary set is denoted Ω of partial $\mathcal{S}$
- These are grid points whose closest point is on the physical boundary of the manifold (next)
- These are the purple points here, since they all have the white boundary point as their closest point
- You can also think of the boundary set as all the grid points past the green normal manifold $\mathcal{S}_\perp$ at the boundary

Zero-Neumann Exterior BCs



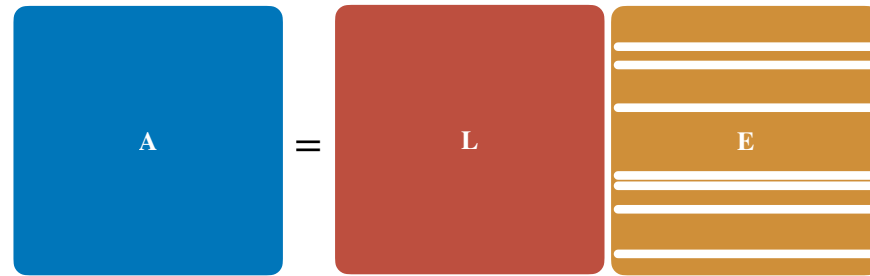
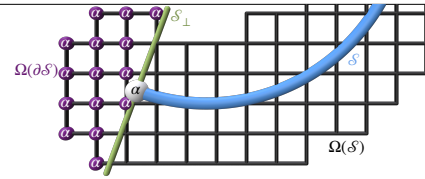
- (0:49) The natural boundary conditions of the closest point method are 1st order accurate zero-Neumann conditions
- In other words, this is what you get if you just build the discrete system in the same way as if it were a closed manifold
- This is because the closest point method extends data constant in the normal direction (next), so there is a zero gradient in the normal direction
- But wait now, (next) it is the conormal direction that zero-Neumann conditions must satisfy
- Because the normals for the grid points past $\mathcal{S}$ perp are not uniquely defined, (next) they are in all directions in the half tubular region of the boundary set
- So there is also a zero gradient in the conormal direction

Dirichlet Exterior BCs



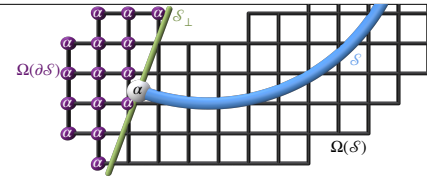
- (0:22) For Dirichlet boundary conditions we actually do need to modify the equations
- The exact prescribed boundary condition (next) α at the closest point is extended to the purple boundary points instead (next)
- This results in a first order accurate approximation of the Dirichlet boundary condition

Dirichlet Exterior BCs



- (0:30) The changes necessary for the matrix representation of the Laplace-Beltrami approximation only occur in the closest point extension matrix
- Nothing changes for discrete Laplacian because there is no grid connectivity changes
- Just the data that is extended onto the grid in the boundary subset changes
- So first (next) the matrix E is assigned identity rows for indices that belong to the boundary set in purple

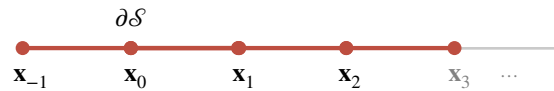
Dirichlet Exterior BCs



$$A u = b$$

- (0:23) Then, a second modification has to happen at the full matrix system level
- We need equations for the boundary subset to state that u equals α
- So A (next) gets identity rows for the boundary set indices and (next)
- b gets the prescribed Dirichlet value α

2nd-Order Zero-Neumann BCs

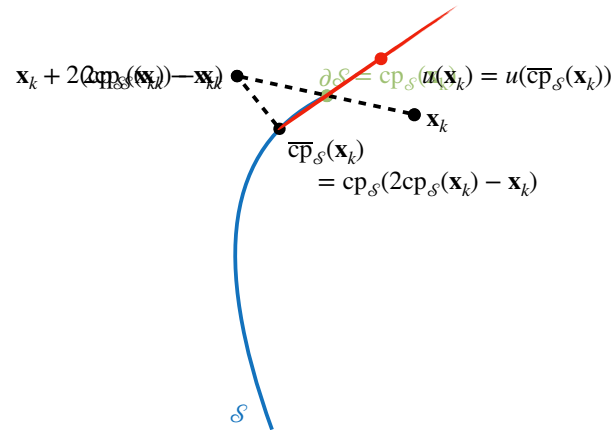


$$\left. \frac{\partial u}{\partial x} \right|_{\partial\mathcal{S}} = 0 \approx \frac{1}{2h} \boxed{u_1} - \boxed{u_{-1}}$$

$$\Rightarrow u_{-1} = u_1$$

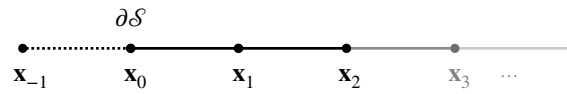
- (1:30) For second order accuracy, let's first consider what is done for standard Cartesian finite differences
- You have the boundary (next) partial S at the grid point x_0
- The Laplacian in 1D is approximated using a 3 point stencil (next)
- This is the Laplacian stencil for x_2 , and it is applied on all grid points (next) like x_1 also
- The stencil also needs to be used at x_0 , but it would extend outside the domain, so a ghost point (next) x_{-1} is introduced and now the stencil can fit
- We obtain data for x_{-1} using the zero Neumann boundary condition
- (Next) In 1D, the Neumann condition is just that the derivative at partial S is equal to zero (next)
- A second-order finite difference approximation centered at x_0 involves data at (next) x_1 and (next) x_{-1}
- We rearrange this equation to find that (next) u at x_{-1} is equal to u_1
- The key property I wanted to show with this example is that the data at the ghost location outside of the domain is replaced (next) by the data at its mirror location,
- That is, the location reflected through partial S

2nd-Order Zero-Neumann BCs



- (1:32) For the closest point method, grid points like this x_k in the boundary set are like the ghost points in standard finite differences
- To achieve second order accuracy we need to improve the data that is extended to the ghost points
- I will take you through the construction of the modified extension, and it will make sense why in the end
- The closest point (next) to x_k is the on the boundary partial S
- We project two times (next) in the direction the closest point to get this new point, (next) which simplifies to 2 times the closest point minus x_k
- Then we compute the closest point of this doubly projected point, (next) giving what is denoted as $cp_{\bar{S}}(x_k)$
- (Next) it is the data at $cp_{\bar{S}}$ that is extended to x_k (next)
- This is because $cp_{\bar{S}}$ is an approximate mirror location (next)
- You are basically approximating the manifold as a plane near partial S (next) and this red point would be the closest point to x_k for that extended manifold
- If the manifold is planar near partial S , $cp_{\bar{S}}$ is the exact mirror location

2nd-Order Dirichlet BCs



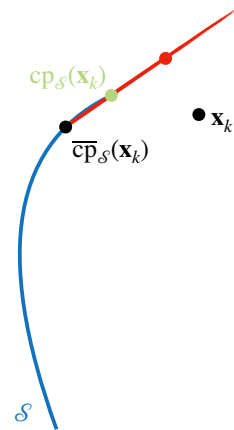
$$u(x-h) = u(x) - h \frac{\partial u}{\partial x}(x) + O(h^2)$$

$$u_{-1} = u_0 - h \left(\frac{\partial u}{\partial x} \Big|_{\partial\mathcal{S}} (u_1 - O(h^2)) + O(h^2) \right) + O(h^2)$$

$$u_{-1} = 2u_0 - u_1 + O(h^2)$$

- (0:52) For 2nd order Dirichlet boundary conditions we again take inspiration from standard Cartesian finite-differences (next)
- We can Taylor expand u at x minus h about x (next) which discretely gives u at $x-1$ equal to u at x_0 , minus h times the derivative at x_0 , plus higher order terms
- Substituting in the second order approximation of the derivative we seen in the Neumann case, gives this (next)
- And finally, rearranging (next) gives the equation for the Dirichlet ghost point as 2 times the value at the boundary x_0 minus the value at the mirror location x_1

2nd-Order Dirichlet BCs



$$u_{-1} = 2u_0 - u_1 + O(h^2)$$

$$u(\mathbf{x}_k) = 2u(\text{cp}_S(\mathbf{x}_k)) - u(\overline{\text{cp}}_S(\mathbf{x}_k))$$

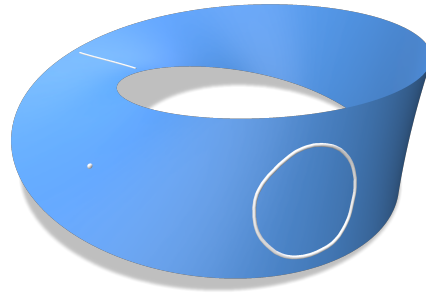
- (0:46) Now we want to convert this equation to our closest point method setting
- Recall that we want to extend data to the boundary set point $\mathbf{x}_k$ (next)
- With a planar approximation of the extended manifold, this red point (next) would be its closest point
- The green point is (next) the actual closest point of $\mathbf{x}_k$
- And the black point is (next) the approximate mirror location $\overline{\text{cp}}$
- (Next) So we extend data to $\mathbf{x}_k$ as 2 times the prescribed Dirichlet value at the actual green closest point minus the solution at the mirror location $\overline{\text{cp}}$

Inhomogeneous Neuman and Robin BCs



- (0:18) I won't discuss the details in this course, but similar ideas were used to handle inhomogeneous Neumann and Robin boundary conditions in a paper published just this month, which first appeared on arXiv at the end of 2025

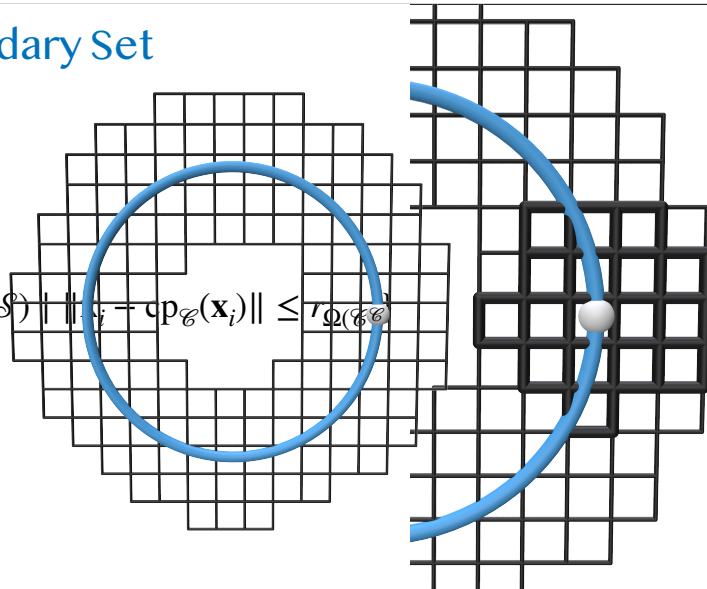
Interior Boundary Conditions



- (0:23) Now we will get into the details around what made many geometry processing applications possible with the closest point method: Interior Boundary Conditions
- Are there any questions up to this point? (10 minutes / 32 minutes)
- As a reminder, interior boundary conditions are defined at points or curves located away from the physical boundary of the manifold

Interior Boundary Set

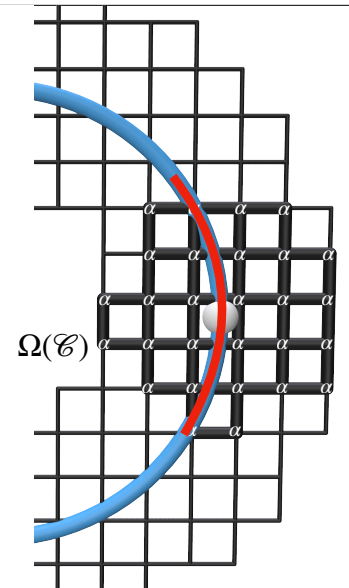
$$\Omega(\mathcal{C}) = \{\mathbf{x}_i \in \Omega(\mathcal{S}) \mid \|\mathbf{x}_i - \text{cp}_{\mathcal{C}}(\mathbf{x}_i)\| \leq r_{\Omega(\mathcal{C})}\}$$



- (0:45) So you have some boundary condition you want to impose, located at the point C, shown in white
- The big problem is that the embedding space near C overlaps the manifold, unlike in the exterior boundary case.
- In the exterior case we are free to alter the closest point extension of data to impose boundary conditions, but here we must simultaneously satisfy the PDE (next)
- The interior boundary set are the grid points in this thick black region (next)
- It is denoted Ω of C and are all the grid points within a tube radius distance to their closest point on C

Auer et al. 2012

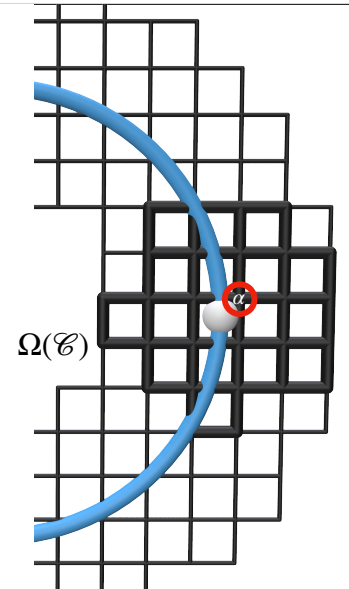
- Fix all points in $\Omega(\mathcal{C})$ to have Dirichlet value α
- 1st order accurate, but with large error constant!
- Only applies to Dirichlet BCs



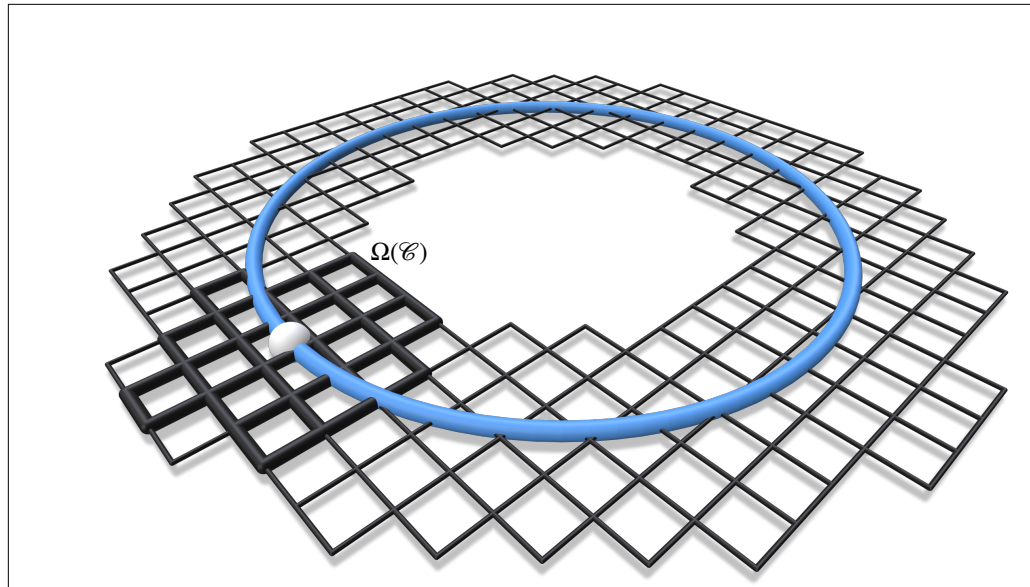
- (0:30) There was one existing approach to impose interior boundary conditions in the paper of Auer et al. (next)
- They fixed all points in $\Omega(\mathcal{C})$ to have the Dirichlet value (next)
- But this is obviously only first order accurate and it has a large error constant (next)
- This is because you are basically fixing this entire red region of the manifold to have the Dirichlet value (next)
- This approach can also only handle Dirichlet boundary conditions

Nearest Grid Point

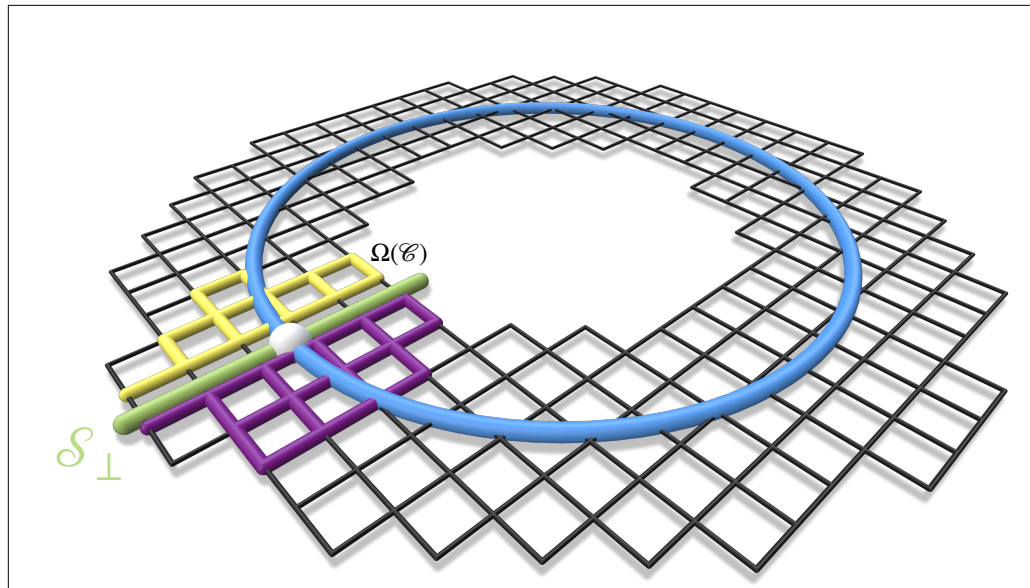
- Instead fix nearest grid point
- First order accurate, improves error constant
- Only applies to Dirichlet BCs



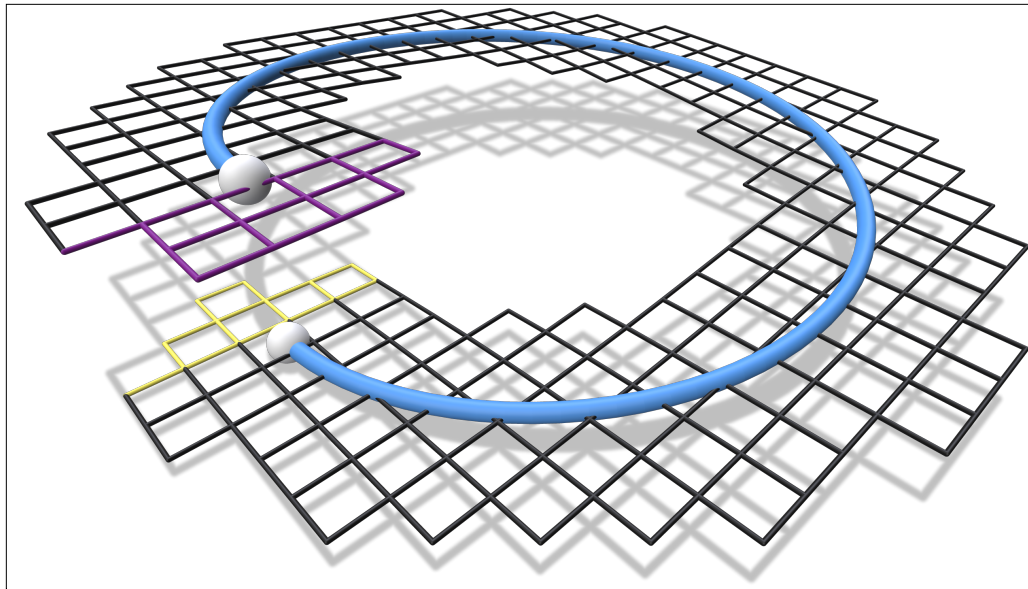
- (0:38) We realized during our exploration of interior boundary conditions that you could also just fix the nearest grid point with the Dirichlet value (next)
- This is however still only first order accurate, but it improves the error constant of the Auer et al. method (next)
- Again, it only applies to Dirichlet boundary conditions
- However, if all you need are Dirichlet interior boundary conditions that are 1st order accurate with a descent error constant, fixing one grid point is much simpler to implement than the full framework I will discuss next



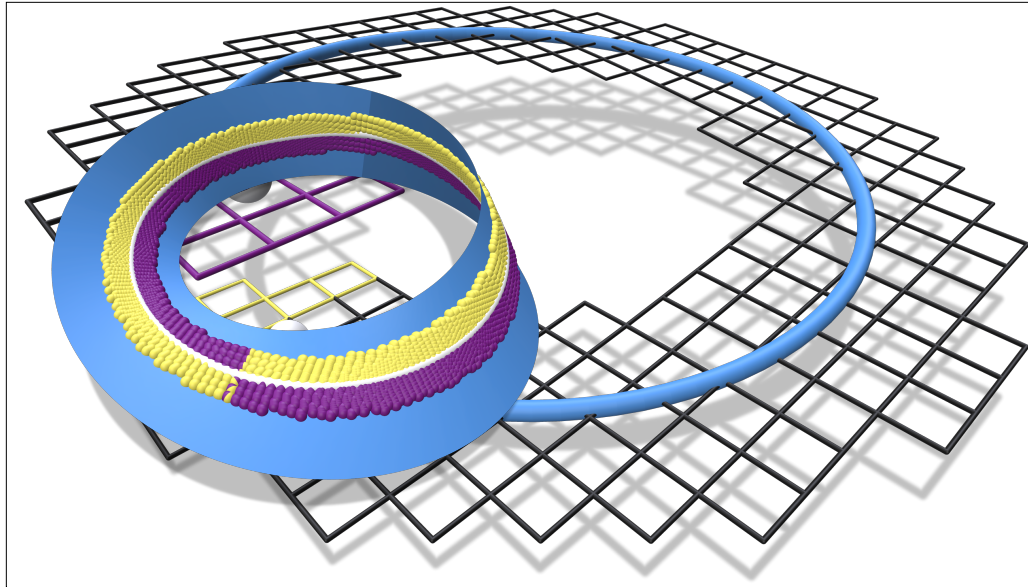
- (0:28) For the general framework, consider the 1D curve embedded in 2D with an interior boundary point shown here in white.
- Boundary conditions require modifications to stencils near the boundary, but as I mentioned before, we also need to impose the PDE itself in this same region
- Therefore, the grid is duplicated near the white point to introduce new degrees of freedom, that is within the thick black grid Ω_C (Next)



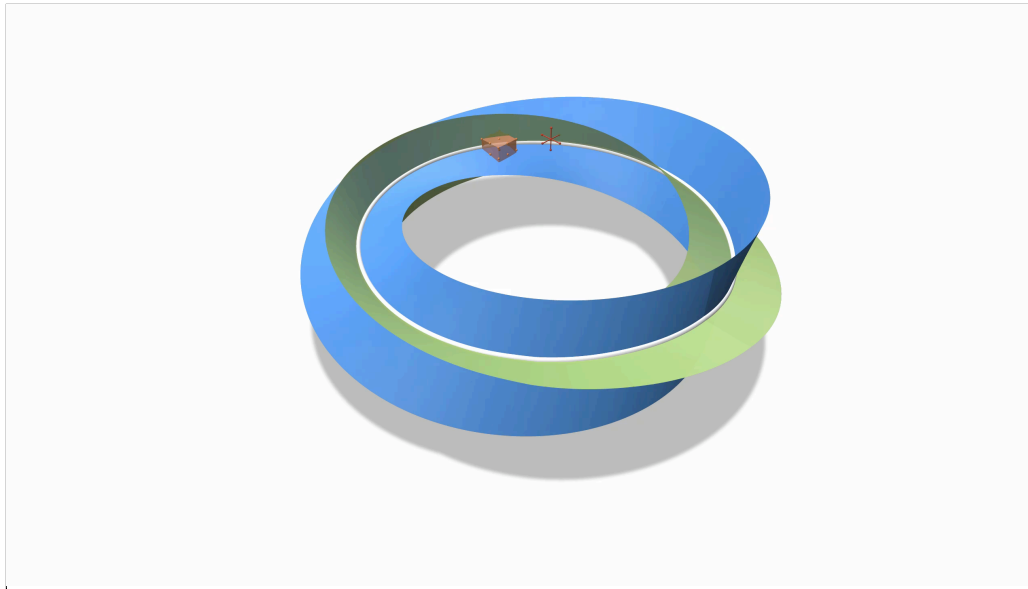
- (0:16) The original black grid and the new interior boundary grid are partitioned based on the location of their closest points with respect to the normal manifold $S_{\perp}$, shown in green.
- The two sides of the duplicated degrees of freedom are coloured yellow and purple. (Next)



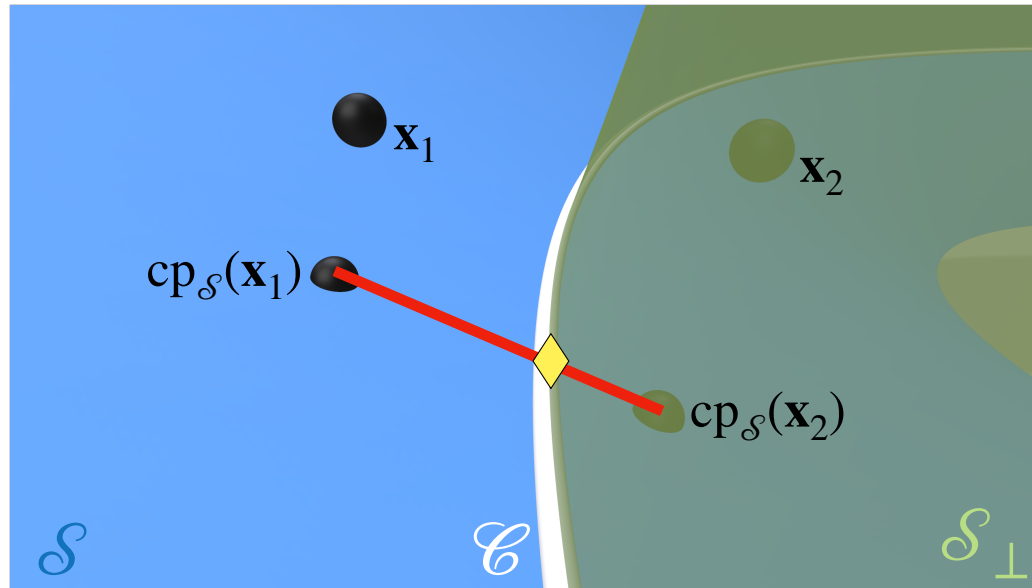
- (0:55) Finally, each side of the new interior boundary grid is joined to the opposite side of the original grid. We warp into 3D to show the final grid connectivity. In actuality, the final grid is a nonmanifold grid in 2D.
- But here is the important thing, now you have space in these purple and yellow regions to alter the closest point extension to impose boundary conditions.
- The exact same techniques for exterior boundary conditions are used in these yellow and purple portions of the nonmanifold grid to enforce interior boundary conditions
- Unfortunately there is one problem. You can implement it exactly like this for orientable manifolds and this illustration is great to understand intuitively what should be done. (Next)



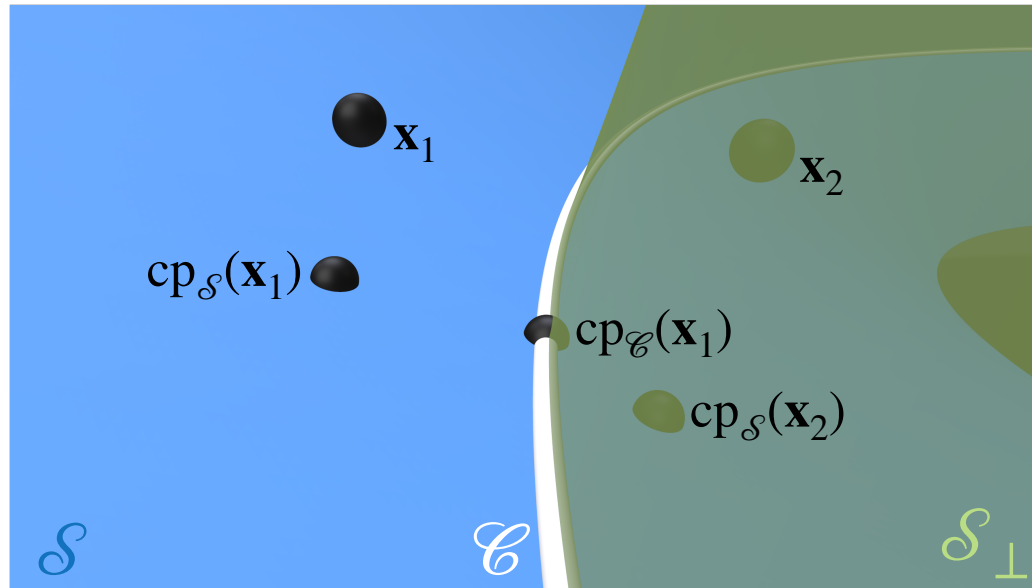
- (0:24) But, constructing the nonmanifold grid directly only works if a global partition of the grid can be obtained. For nonorientable surfaces where the interior boundary fully loops around, no global partition is possible.
- You have this unavoidable flip of the yellow and purple regions as you travel around the white curve



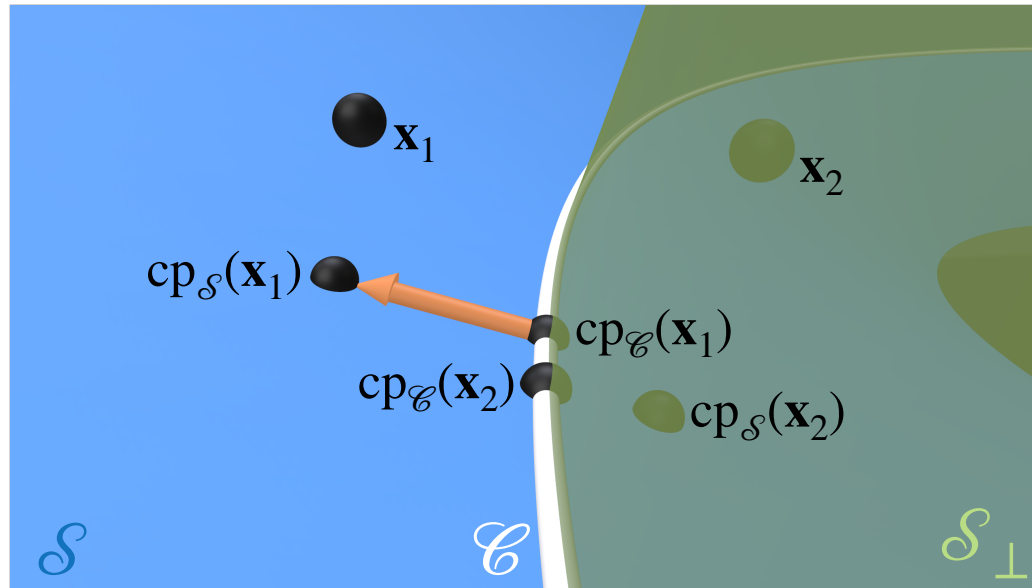
- (0:44) Instead, we locally check each stencil for grid points whose closest points cross the green normal manifold. This example shows an interpolation stencil in orange and a finite-difference stencil in red that intersect the normal manifold.
- Modifications are performed on a per stencil basis to enforce the boundary conditions
- The modifications happen at grid points whose closest points cross with respect to what we call the stencil director.
- The stencil director for an interpolation stencil is the interpolation query point
- The stencil director for finite difference stencils is the closest point of the stencil's centre



- (0:28) To successfully apply our approach we must be able to determine if the closest points are on opposite sides of the normal manifold $\mathcal{S}_\perp$. Consider two points $\mathbf{x}_1$ and $\mathbf{x}_2$ with corresponding closest points $\text{cp}_{\mathcal{S}}$.
- A naive approach would construct an explicit representation of $\mathcal{S}_\perp$, for example as a mesh, and then check (next) if the line segment between the closest points (next) intersects $\mathcal{S}_\perp$. (Next)

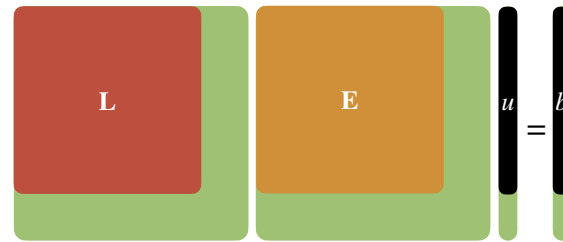
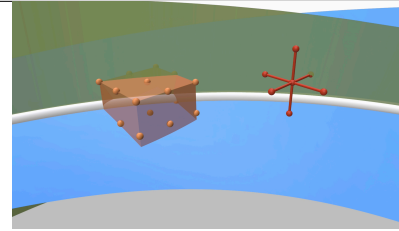


(0:07) Instead, we compute the closest points (next) to the boundary curve $\mathcal{C}$



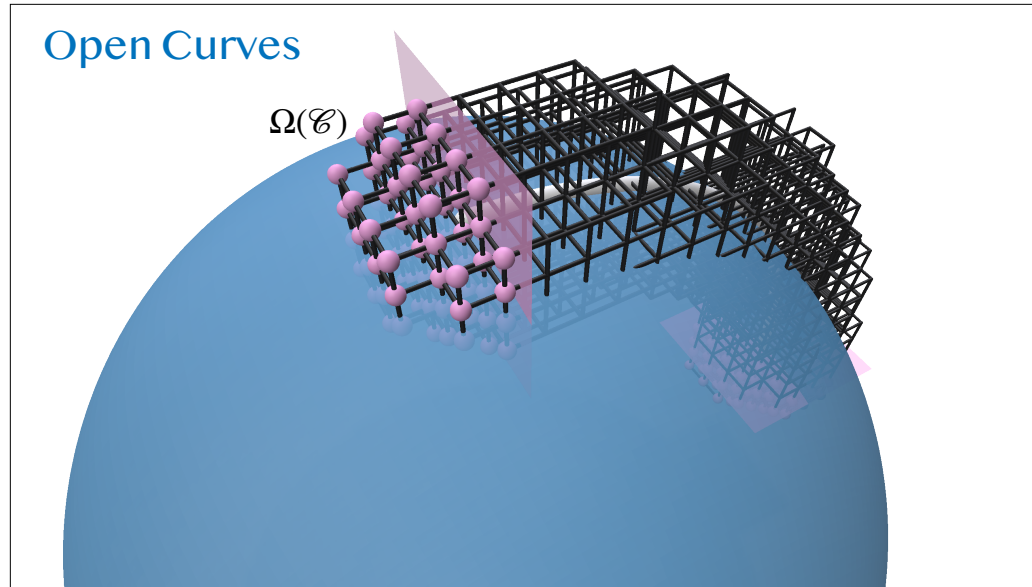
- (0:28) And the orange vectors (next) will point in opposite directions if $\text{cp}_{\mathcal{S}}$ of $\mathbf{x}_1$ and $\mathbf{x}_2$ are on opposite sides of $\mathcal{S}_{\perp}$. So it becomes just a simple test on the sign of the dot product between the vectors
- I will refer to these orange vectors as cp diff , since they are the difference between the closest point on $\mathcal{S}$ and the closest point on $\mathcal{C}$

Boundary Condition DOFs

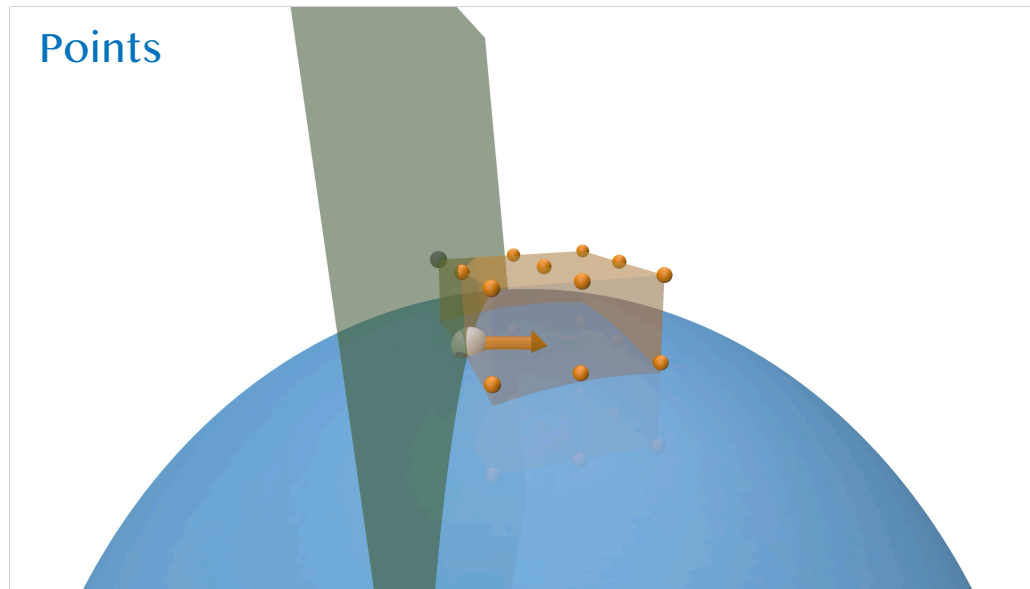


- (1:27) So in the overall matrix equation, you add extra rows and columns for the new interior boundary condition degrees of freedom shown in green
- Recall that each row corresponds to a specific grid point, which in this case is the index of the stencil director
- The non-zero column entries correspond to the stencil weights for each grid point that is part of the stencil for that row
- Column entries are moved to their corresponding boundary condition DOF in the green region if the column's closest point was on the opposite side of S-perp compared to the stencil director
- There is a key difference here compared to exterior boundary conditions, the Laplacian matrix also needs to be modified, not just the extension
- However, the stencil weights for the Laplacian are just moved to a different column because it is just a change in connectivity of the degrees of freedom
- The weights in the extension matrix E and green rows in u and b will change to impose the boundary condition equations
- The changes in the closest point extension depend on what type of boundary condition is imposed, like Dirichlet or Neumann, and if you want first or second order accuracy

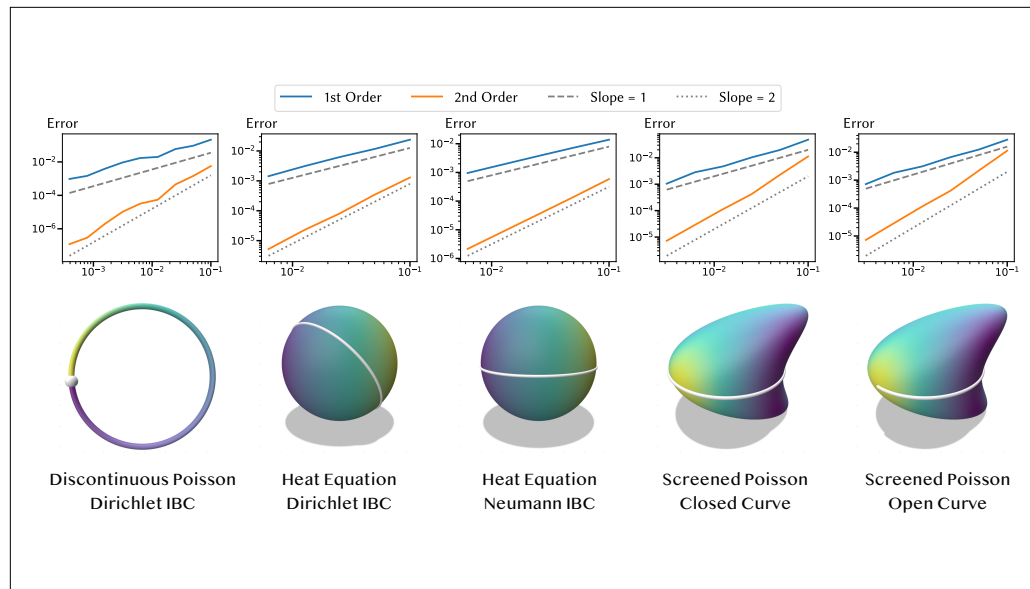
Open Curves



- (0:40) One thing to know about boundary conditions located on open curves is that you don't apply the boundary condition modifications outside the endpoints of the open curve (next)
- All grid points within a tube-radius distance from the curve are in the interior boundary set definition (next)
- This would include the half-tubular end caps around the endpoints of the open curve shown in pink
- But we only want the PDE to be enforced in these pink regions
- So we don't apply the boundary condition modifications if the closest point to the curve $\mathcal{C}$ is one of the endpoints



- (0:50) Maybe less intuitive is that there is no change needed for point interior boundary conditions
- This is because we want the equations for both the boundary conditions and the PDE to be enforced everywhere in the space surrounding the white boundary point
- The stencil director rotates around the white point in all directions as the interpolation query point circles around
- The stencil is split into regular PDE based equations and boundary condition equations by the half space plane defined by the stencil directors direction
- The purple points here correspond to the indices involved in the boundary condition modifications
- So you can see as the you consider all the stencils surrounding the white point, all grid points get coloured both orange and purple



- (0:25) The expected convergence rates were verified on a multitude of analytical problems.
- We checked both the first order and second order interior boundary condition algorithms
- Also both Dirichlet and Neumann conditions for different PDEs were used for convergence tests
- We even checked a discontinuous Poisson equation with two different Dirichlet values on either side of the boundary in the leftmost example

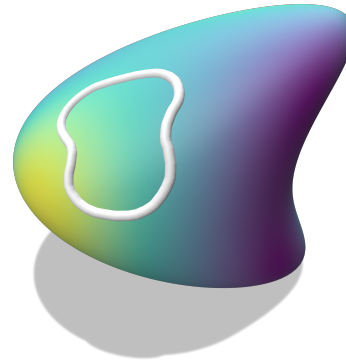
Poisson Equation

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3 \mid (x - z^2)^2 + y^2 + z^2 = 1\}$$

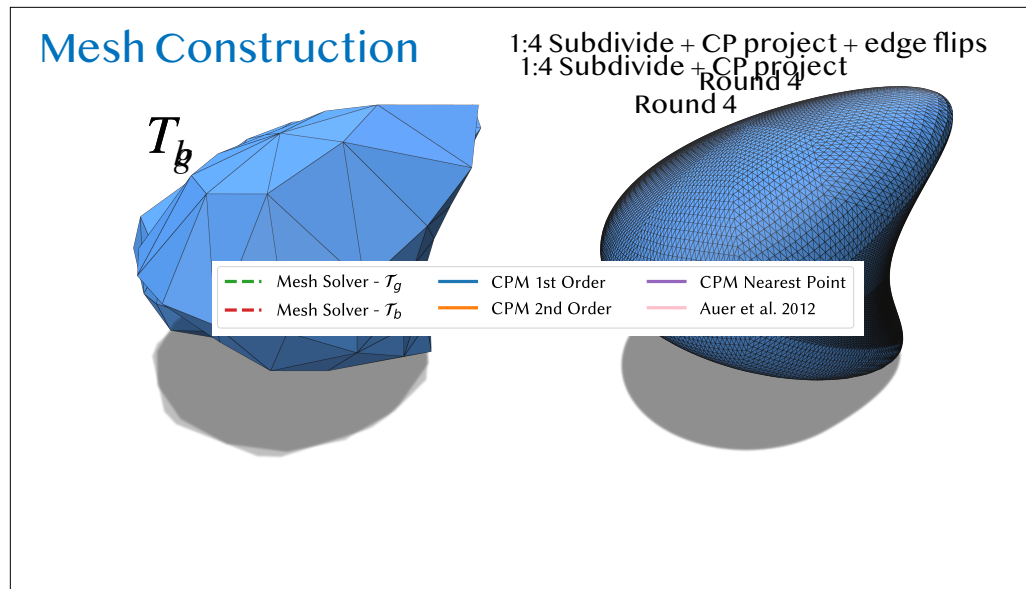
$$\Delta_{\mathcal{S}} u = f$$

$$u = xy$$

$$f = \frac{2y(-32x^5z^4 - 16x^5z^2 - 32x^4y^2z^4 + \dots)}{(4y^2z^2 + 4xz^2 - 4z^2 - 1)^3}$$

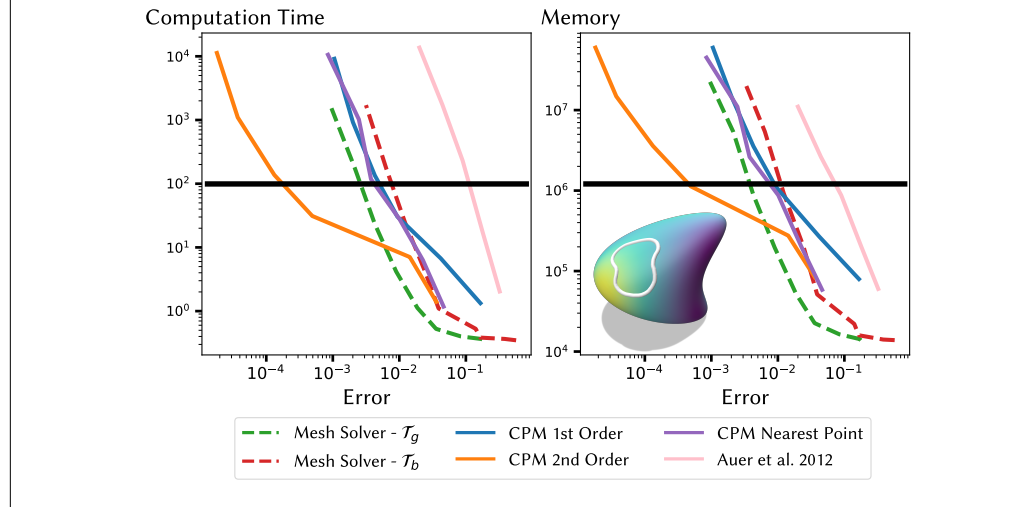


- (0:54) We also did a comparison between using the closest point method approaches and the mesh based cotan Laplacian
- We solve the Poisson equation on the Dziuk level-set surface, with the exact solution $u = xy$ and impose a Dirichlet boundary condition on the white curve
- It might seem like a simple problem, but the right hand side function f for this exact solution on this surface is (next) given by this equation
- Which has 50 plus terms in the numerator with different polynomial degree combinations
- So it makes for a decent analytical test problem
- Also it is nice that this surface has both positive and negative curvature
- All other problems with exact solutions I know of are on manifolds with only positive curvature, like the sphere, torus, and cylinder



- (1:14) I will show the comparison on the next slide, but first, these are the different methods
- We have a good and bad triangulation, T_g and T_b , as the dashed lines, the main IBC closest point method with 1st and 2nd order accuracy in blue and orange. Then the nearest point approach in purple and the only existing method by Auer et al. in pink.
- (Next) I want to focus on how the meshes were constructed here
- This is the bad triangulation that has some quite poorly shaped triangles
- (Next) Here is the it after 4 rounds of refinement, which is done by splitting each triangle by adding a vertex at the midpoint of edges, but projecting the vertex onto its closest point also
- (Next) This is the good starting triangulation and the refinement after 4 rounds. Delaunay edge flips are also performed to improve this triangulation at each level.

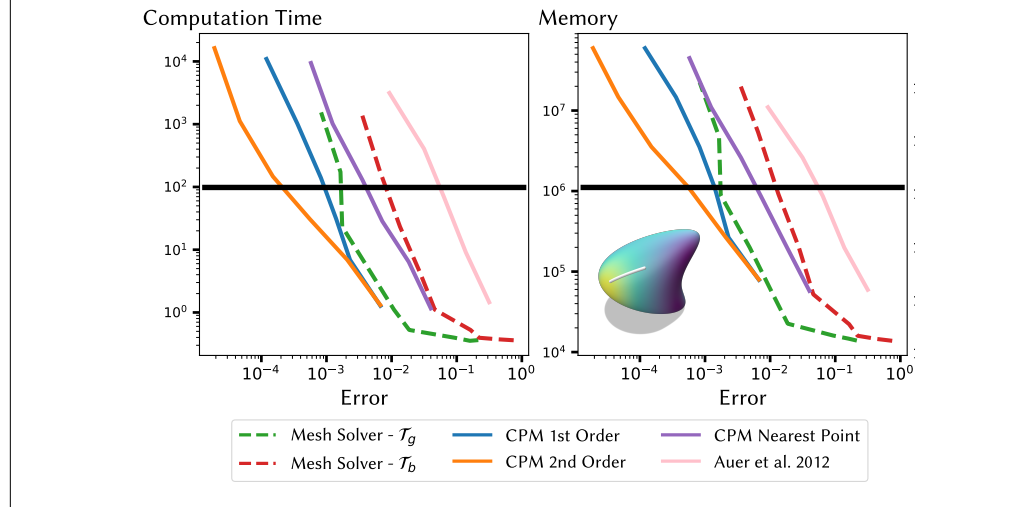
Closed IBC



- (1:45) We will start with a closed curve for the interior boundary condition comparison
- The first thing to point out about the comparisons are the axes. Computation time vs error and memory usage vs error.
- This is because you can not directly compare the mesh resolution with the grid spacing of the closest point method. The grid spacing in the embedding space does not correspond to the distance between points on the surface. Which is what it is in a mesh.
- The closest points for neighbouring grid points can spread apart or come closer together depending on the convexity of the surface
- Comparing instead computation time vs error(next), it can be seen that our 2nd order implementations in orange have the lowest error for a fixed computation time
- (Next) The same behavior of all the methods is seen for memory usage also
- The first order and nearest point approach in blue and purple are basically the same and fall between the dashed lines from the good and bad triangulations
- This makes sense because the mesh based method is also first order accurate, but it is nice to see that the error constant is much better than the existing Auer et al. method in pink
- The mesh based method is first order accurate because the interior boundary condition is imposed by fixing the Dirichlet value at all the nearest vertices to the white curve

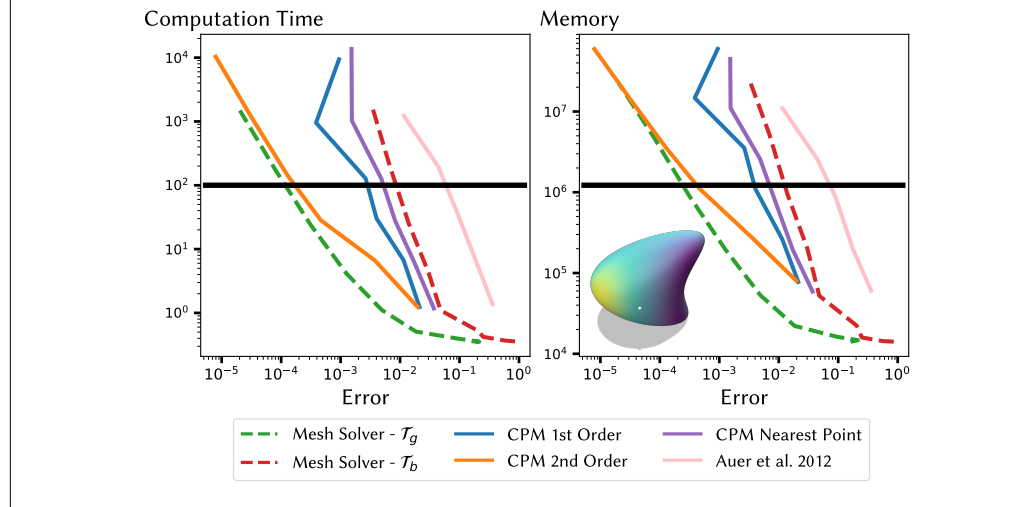
- The meshed based method might be improved in different ways, but we were just looking for a baseline
- The purpose of this comparison was just to see roughly how the closest point method performs compared to a mesh based method

Open IBC



- (0:47) For an open curve, remember the 1st and 2nd order method are slightly different because the grid points near the endpoints of the curve are not included
- Comparing again the error for a fixed computation time or memory usage (next), it can be seen that now our 1st and 2nd order implementations in blue and orange have the lowest error
- The nearest point and Auer et al. methods in purple and pink did not change in comparison to the dashed lines for the mesh based method
- However, our 1st order accurate approach is now roughly an order of magnitude better than the nearest point approach and better than both meshes

Point IBC

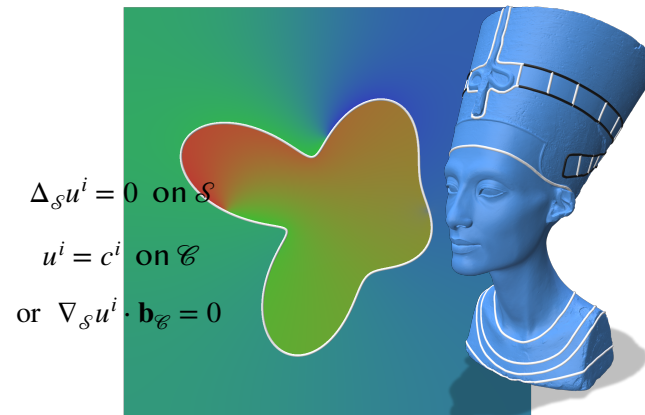


- (1:00) For a point interior boundary condition, the pink Auer et al. method is still the worst.
- You can see that the error constant for the 1st order method in blue is only slightly better than the nearest point approach in purple, but they are roughly 1 and a half orders of magnitude better than the Auer et al. method
- The big difference in the story here is with the mesh based method. The bad triangulation in red still follows 1st order accuracy, but the good triangulation in green it is slightly better than our 2nd order method in orange
- This is because we choose the boundary point location to be a vertex in the triangulation
- Now the mesh based method can also achieve second-order accuracy, but this degrades with a poor triangulation
- The closest point methods are agnostic to the quality of the mesh because the closest points are computed from the level-set representation of this surface

Geometry Processing Applications

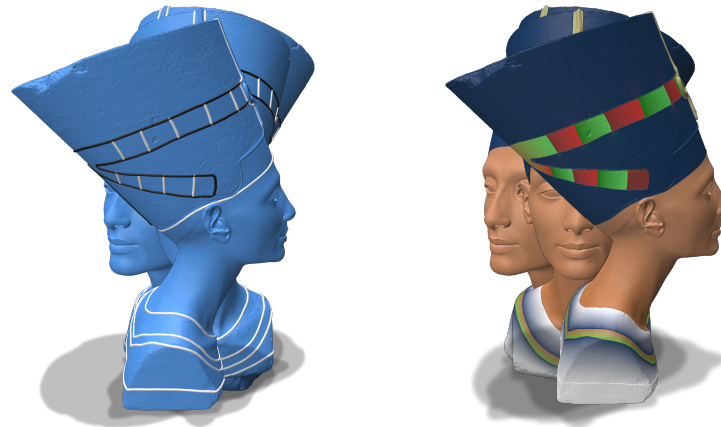
- (1:15) Okay, so now we know the method converges as expected and has computation time and memory usage similar to mesh based methods, which are probably the most common approach
- Before getting into geometry processing applications let's recap the benefits of the closest point method
- As I just mentioned, it is independent of mesh quality
- It applies to any manifold representation that allows closest point queries
- It doesn't matter if the manifold is orientable or not
- It works in any dimension and the manifold can be any codimension
- I will even show an example where the object is mix-codimensional and nonmanifold
- This gives us a powerful framework for PDE based geometry processing since we only need one implementation for numerous representations and types of objects
- Are there any questions up to this point?
- Okay, so let's look at some applications

Diffusion Curves



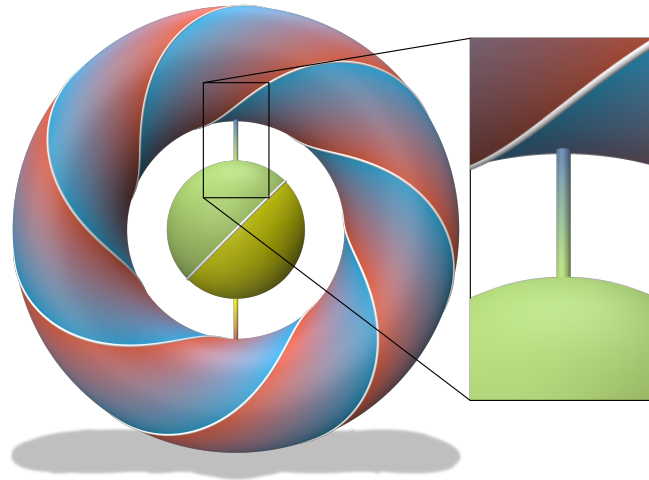
- (1:17) Diffusion curves were originally developed as a compact representation of an image
- You just define colours on curves in your image, like this white curve, and then solve a PDE to get the colour over the rest of the image
- The colours can vary along the curve and on both sides of the curve like in this example
- This idea can also be used to colour the surface of objects (next)
- You just replace the Laplace equation with the Laplace-Beltrami equation
- The Laplace-Beltrami equation is solved independently for each colour channel, red, green, or blue, where the superscript i denotes the channel (next)
- Dirichlet interior boundary conditions store two-sided colour values on the white curves in this example and (next)
- Zero-Neumann conditions are also used on the black curves to block colour changes across the curves
- The vector $\mathbf{b}$ in the Neumann condition is the binormal of the curve, that is, the cross product between the curve's tangent and the surface normal, which is in the same direction that the conormal would be for an open manifold

Diffusion Curves



- (0:30) The whole surface on the right is coloured from the boundary conditions on the left
- You can see how the colour gradient in the band on the headdress goes from dark red to light and dark green to light from left to right
- This is because the zero-Neumann conditions on the top and bottom block any navy from leaking in and allow the colours within the band to change only in the tangential direction of the black curves

Diffusion Curves



- (0:23) This is an example on a non manifold object
- The Dirichlet interior boundary curves are shown in white, with colours spreading out on either side
- You can see from the zoom in that the blue and green spread across the 1 dimensional line connecting the torus and sphere, this is where the nonmanifold junctions are

Vector Field Design

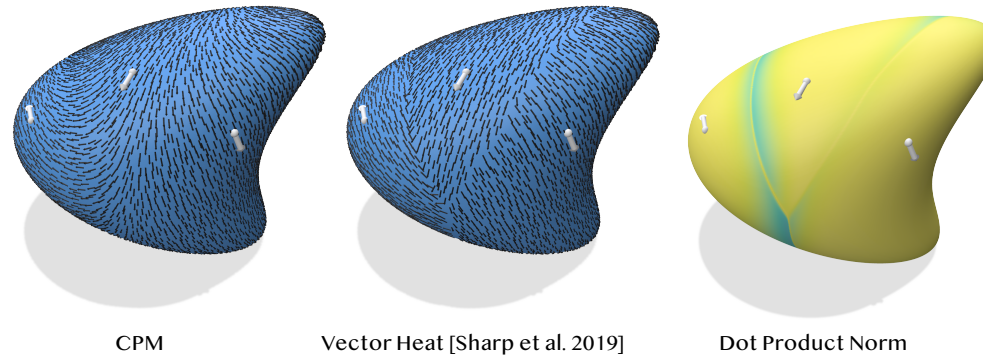
$$\frac{\partial u^i}{\partial t} = \Delta_{\mathcal{S}} u^i \text{ on } \mathcal{S}^?$$

$$\begin{cases} u^i = c^i, \text{ or} \\ \nabla_{\mathcal{S}} u^i \cdot \mathbf{b}_{\mathcal{C}} = 0 \end{cases} \text{ on } \mathcal{C}$$



- (0:54) In this application of vector field design, we solve the heat equation instead of the Laplace-Beltrami equation, but still independently per vector component i
- Vector field design is useful for input into other applications like texture mapping an image onto a surface or quadrilateral meshing (next)
- The same Dirichlet and Neumann boundary conditions as diffusion curves can be defined at points or curves on the surface
- But now the Dirichlet conditions correspond to a sparse set of directions that the user wants the vector field to follow
- And the Neumann conditions can block direction changes across curves
- We use backward Euler time-stepping and project out the surface normal component of the vectors after each time step to keep them in the surface
- The idea of projecting out the normal component came from this paper by Turk

Vector Laplacian



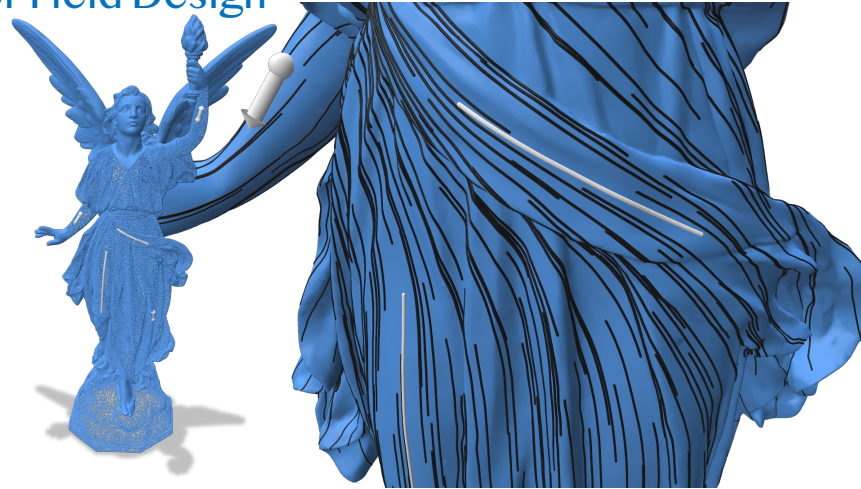
CPM

Vector Heat [Sharp et al. 2019]

Dot Product Norm

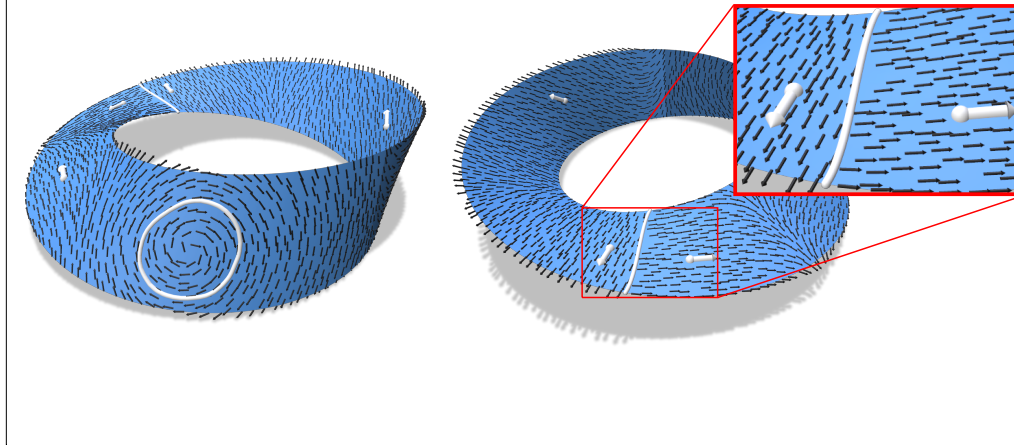
- Note that this is not the same as doing parallel transport like the vector heat method does
- The right most figure shows the norm of the dot product between the vectors from our CPM approach and the vector heat method.
- Both methods produce similar vector fields except near the cut locus of the interior boundary points.
- Our method provides a smoother transition across the cut locus, which may be beneficial or disadvantageous depending on the application.
- Sources and sinks can also appear with our method, whereas the vector heat method has discontinuous changes in the vector field across the cut locus

Vector Field Design



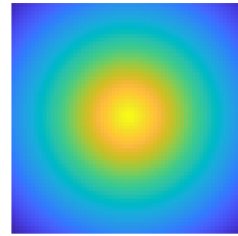
- (0:36) Here is an example of designing a vector field on the Lucy statuette using a sparse set of directions using our approach
- The closest points are computed as just the nearest neighbours in the point cloud
- This is only accurate if the point cloud is dense enough
- The paper I mentioned towards the beginning of the course could be used in general (next)
- All the white boundary conditions used in this example are Dirichlet conditions
- The tangent directions of the curves are used to specify directions over a larger area of the surface

Vector Field Design

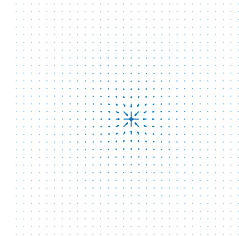


- (0:16) In this example on the Möbius strip we have both Dirichet and Neumann conditions
- You can see from the zoom-in that the Neumann boundary curve is blocking the direction change between the vector field on both sides

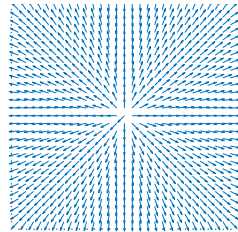
Geodesic Distance



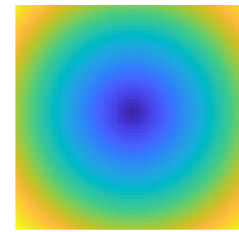
$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u$$



$$\nabla_{\mathcal{S}} u_t$$



$$\mathbf{X} = -\frac{\nabla_{\mathcal{S}} u_t}{\|\nabla_{\mathcal{S}} u_t\|}$$

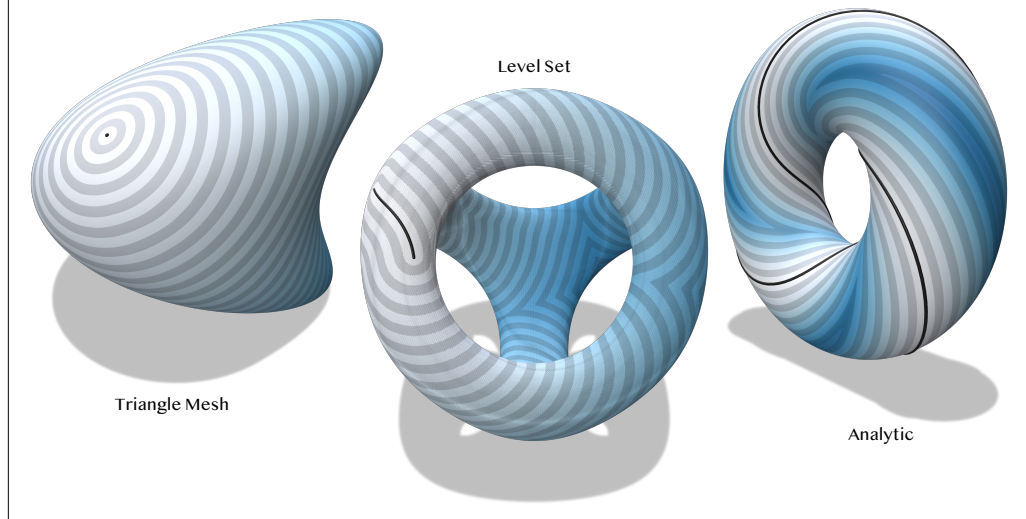


$$\min \int_{\mathcal{S}} |\nabla_{\mathcal{S}} \phi - \mathbf{X}|^2$$

$$\Delta_{\mathcal{S}} \phi = \nabla_{\mathcal{S}} \cdot \mathbf{X}$$

- (0:54) In this application we use the heat method to compute geodesic distance (next)
- To illustrate the steps of the method, consider a point at the centre of this plane that you want to compute the distance to
- The first step solves the heat equation for short time t (next)
- You then compute gradient of the result, which gives a vector field point towards the point, with varying magnitude (next)
- We know that the distance field should satisfy the eikonal equation, which says the gradient has unit norm, so we normalize the gradient (next)
- And in the final step solve an optimization problem to find the closest matching scalar field with that gradient $\mathbf{X}$
- This amounts to solving a Poisson equation

Geodesic Distance



- (1:00) Here are some examples of computing geodesic distance to interior boundaries shown in black. Our closest point method approach provides a single implementation () for multiple surface representations, () which previously required a new () implementation for each type.
- There is a caveat to get the heat method to work properly with the closest point method, which comes from the Dirac-delta initial condition in the heat equation step
- The heat method itself can also be numerically unstable in regions where the magnitude of the heat becomes small
- I won't get into the details now about how we fixed the issues, but I just wanted to point it out because I started with this as a test problem and it led me down some rabbit holes
- Also know that there has been some follow up work by others too

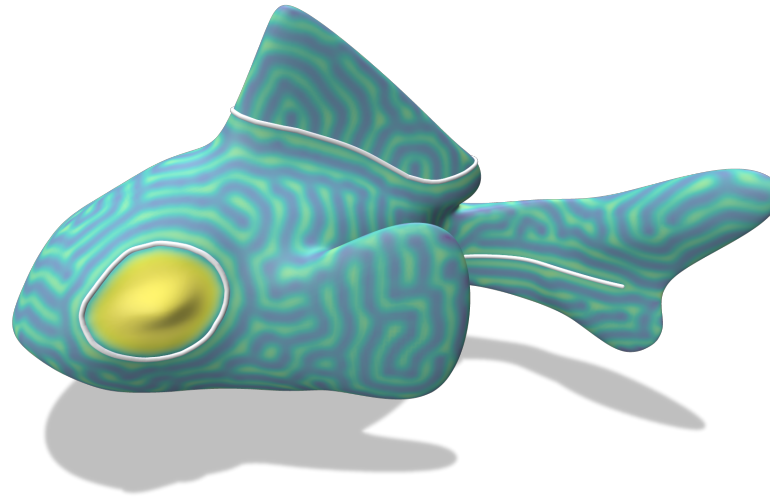
Reaction-Diffusion Textures

$$\begin{cases} \frac{\partial u}{\partial t} = \mu_u \Delta_{\mathcal{S}} u - uv^2 + F(1 - u), \\ \frac{\partial v}{\partial t} = \mu_v \Delta_{\mathcal{S}} v + uv^2 - (F + k)v, \end{cases}$$

$$\begin{cases} u = g_u \text{ or } \nabla_{\mathcal{S}} u \cdot \mathbf{b}_{\mathcal{C}} = 0, \\ v = g_v \text{ or } \nabla_{\mathcal{S}} v \cdot \mathbf{b}_{\mathcal{C}} = 0, \end{cases} \quad \text{on } \mathcal{C}$$

- (0:32) This application uses the Gray-Scott reaction-diffusion system of equations to texture a surface
- u and v are two reactants in a chemical reaction and patterns can form from random initial conditions
- The constants μ , F and k are parameters to control the mixture (next)
- Again both Dirichlet and Neumann boundary conditions can be used for u and v

Reaction-Diffusion Textures



- (0:45) Here is a texture on this fish from solving the reaction-diffusion equation.
- We only show the reactant u and the surface is coloured yellow for high concentrations and purple for low
- The interior boundary conditions are used to control the texture
- Zero Dirichlet conditions allow stripes to be placed around the dorsal fin and tail.
- The upper side of the dorsal fin is a zero Neumann condition, which causes the pattern to intersect perpendicular to the curve.
- Finally, the closed curve with a zero Dirichlet condition controls of concentrations of the u and v in the eye

Harmonic Maps

- Minimizer $\mathbf{u} : \mathcal{S}_1 \rightarrow \mathcal{S}_2$ of

$$E[\mathbf{u}] = \frac{1}{2} \int_{\mathcal{S}_1} \left\| \mathbf{J}_{\mathbf{u}}^{\mathcal{S}_1} \right\|^2_{\mathcal{F}}$$

- Gradient descent flow

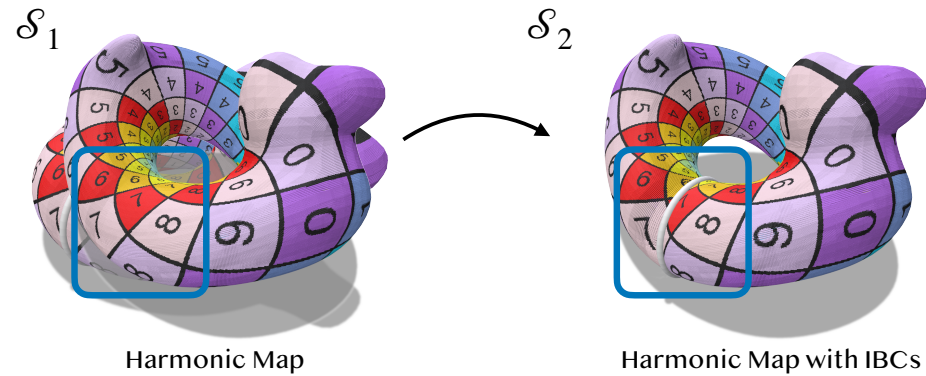
$$\frac{\partial \mathbf{u}}{\partial t} = \Pi_{T_{\mathbf{u}}\mathcal{S}_2}(\Delta_{\mathcal{S}_1} \mathbf{u})$$

Iterate:

- Solve $\frac{\partial \tilde{\mathbf{u}}}{\partial t} = \Delta_{\mathcal{S}_1} \tilde{\mathbf{u}}$ from $\tilde{\mathbf{u}}(\mathbf{x}, 0) = \mathbf{u}^k(\mathbf{x})$ for one time step
- Project $\tilde{\mathbf{u}}(\mathbf{x}, \Delta t)$ onto $\mathcal{S}_2$ by setting $\mathbf{u}^{k+1}(\mathbf{x}) = \text{cp}_{\mathcal{S}_2}(\tilde{\mathbf{u}}(\mathbf{x}, \Delta t))$

- (1:01) A final application is a little different from the rest. It involves computing a harmonic map between two surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$ (next)
- Harmonic maps are minimizers of this energy functional (next)
- The gradient descent flow to minimize this energy is basically a component-wise heat equation on $\mathcal{S}_1$, but (next) there is the projection operator onto the tangent space of $\mathcal{S}_2$ that couples the components
- We showed that a two-step splitting scheme can decouple the two surfaces and is a first order approximation (next)
- You iterate between the following two steps for each time step (next)
- First, solve just the component-wise heat equation on $\mathcal{S}_1$ to give $\tilde{\mathbf{u}}$ (next)
- Then, project the result onto $\mathcal{S}_2$ by computing the closest points of $\tilde{\mathbf{u}}$ onto $\mathcal{S}_2$

Harmonic Maps

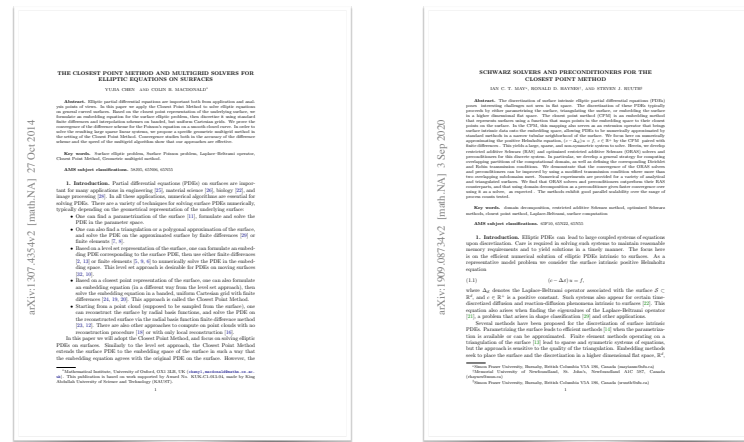


- (0:31) Here is an example where the mapping is used to transfer a texture (next) from $\mathcal{S}_1$ to $\mathcal{S}_2$, while enforcing that the white landmark curves match using a Dirichlet condition. (Next)
- Comparing to just the harmonic map without the white curve constraint (next), it can be seen that the map stretches and compresses near the white curve as expected. Since it must deviate from being harmonic to satisfy aligning the white curves

Efficiency Improvements

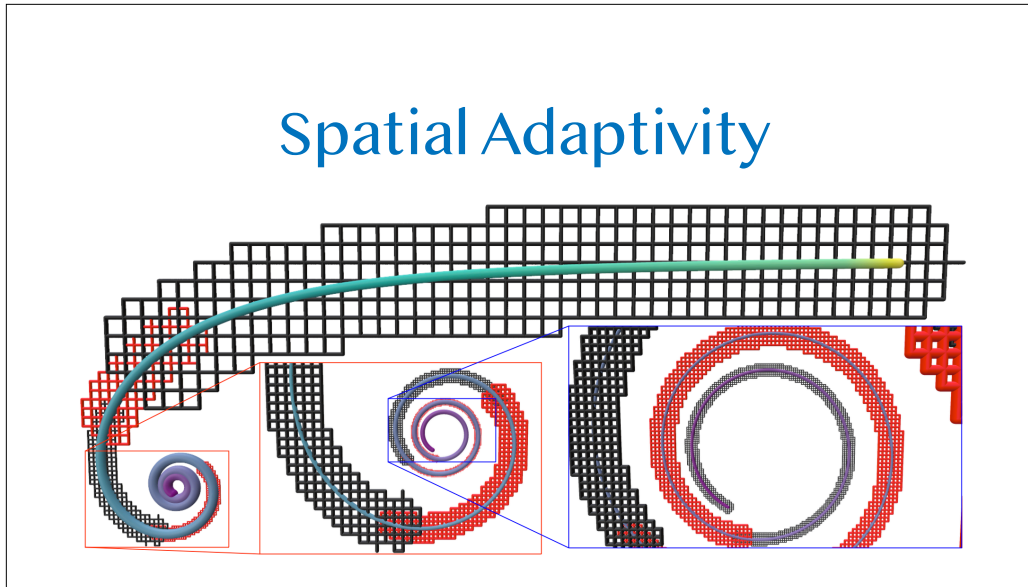
- (1:00) Next I'll talk about ways to improve the efficiency of the closest point method
- Are there any questions about anything so far? (25 minutes / 1 hour)
- So there were some efficiency improvements in this work discussed so far with the construction of the tubular grids
- A breadth first search was used so that closest points only needed to be computed near the manifold and a sparse grid data structure was used to only allocate memory when needed
- But the grid construction isn't the major bottleneck when the matrices become large, it is the linear system solver that takes the most time
- In all the examples above, we used a custom BiCGStab solver that did not need to build the full matrix system
- Instead it used the Laplacian matrix L and the extension matrix E directly, which saved memory

Multigrid and Domain Decomposition

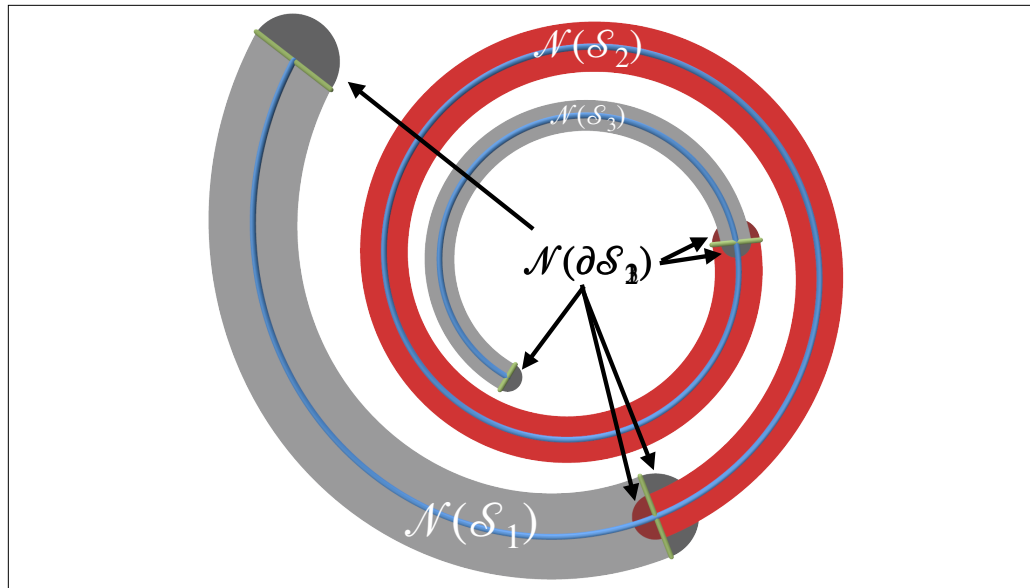


- (0:30) But there has been work on better solvers with the closest point method
- The left paper developed a multigrid solver and the right paper a domain decomposition solver that could exploit distributed parallelism
- These solvers improve the runtime for a given matrix system coming from the closest point method
- But there is actually another way to improve runtime in certain cases without more advanced solvers
- You can use (next)

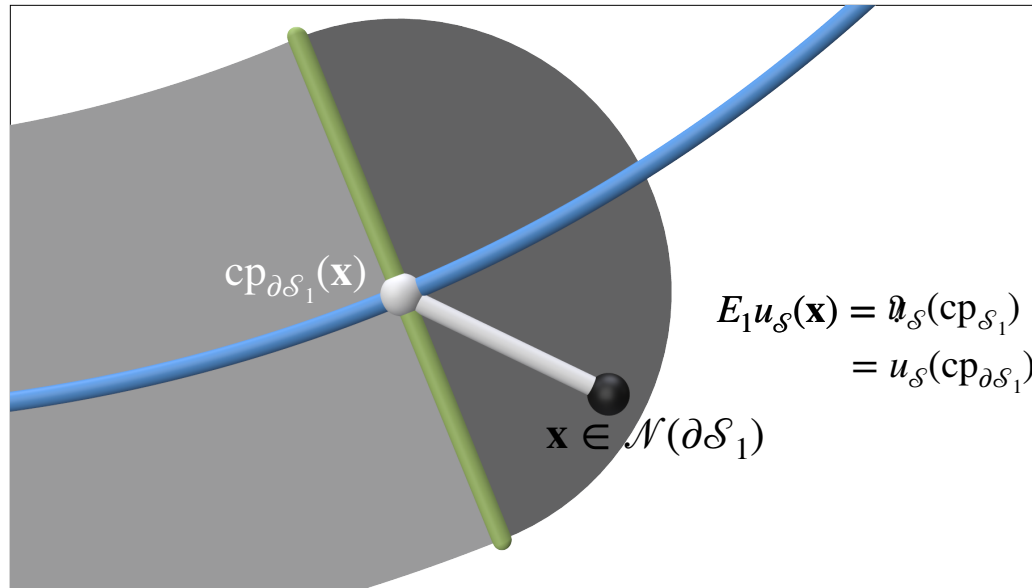
Spatial Adaptivity



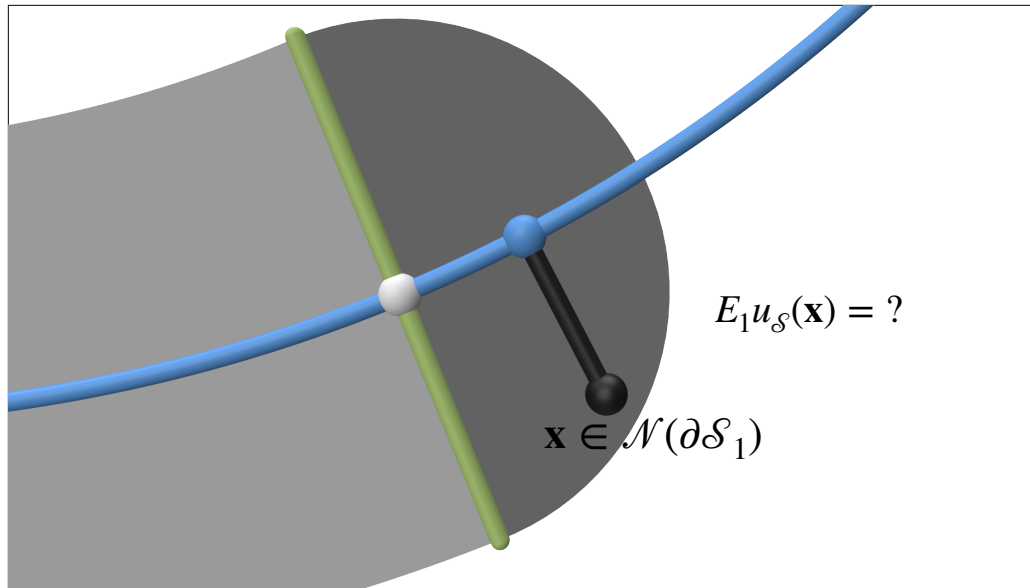
- (0:51) Spatial adaptivity to make the matrix system smaller
- So previous closest point method work always used a single, uniform grid spacing in the embedding space
- This can be extremely inefficient for manifolds that have geodesically distant parts that are close together in Euclidean space
- For this spiral example, the smallest resolution shown in the rightmost zoom would need to be used for a uniform grid, but that would lead to many unnecessary grid points. And thus a bigger matrix system and higher runtime and memory costs.
- So I will discuss a way to couple overlapping computational tubes with different resolutions so we can have fine grids for some regions of the manifold and coarse in other regions



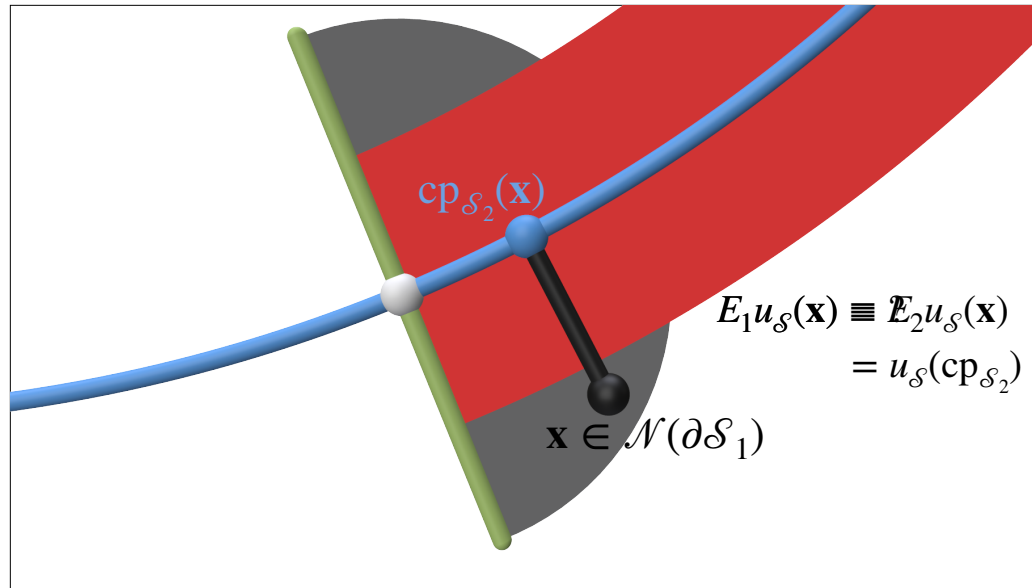
- (0:35) So what we do is break the manifold up into multiple subsets. Each of the subsets of the manifold are given their own tubular neighbourhood (next) $\mathcal{N}$ of $\mathcal{S}_1$ (next) $\mathcal{N}$ of $\mathcal{S}_2$ (next) and $\mathcal{N}$ of $\mathcal{S}_3$, so they can have different tube radii.
- My key insight was a way to globally join the problems on each subset by altering the closest point extension in the boundary subsets (next) shown in a darker colour, $\mathcal{N}$ of partial $\mathcal{S}_1$ are these two darker grey regions (next) $\mathcal{N}$ of partial $\mathcal{S}_2$, the dark red (next) and $\mathcal{N}$ of partial $\mathcal{S}_3$.



- (0:47) So just consider one of the boundaries between two subsets. Any point x in the boundary subset N of partial S_1 will have the closest point cp of partial S_1 , that is, x is closest to the boundary itself. (Next)
- Now denote the closest point extension as $E_1 u_{\mathcal{S}}$, which (next) would naively be defined as $u_{\mathcal{S}}$ of $\text{cp}_{\mathcal{S}_1}$, which for x in N of partial S_1 (next) it would be $u_{\mathcal{S}}$ of cp partial S_1
- But that is incorrect for the global problem (next). What you actually want is the closest point extension to extend data from the manifold constant in the normal direction, and this is obviously not in the normal direction. (Next)



- (0:11) This blue point would be the correct closest point for $\mathbf{x}$ for the global problem. That closest point can be computed using the cp function for S2 (next)

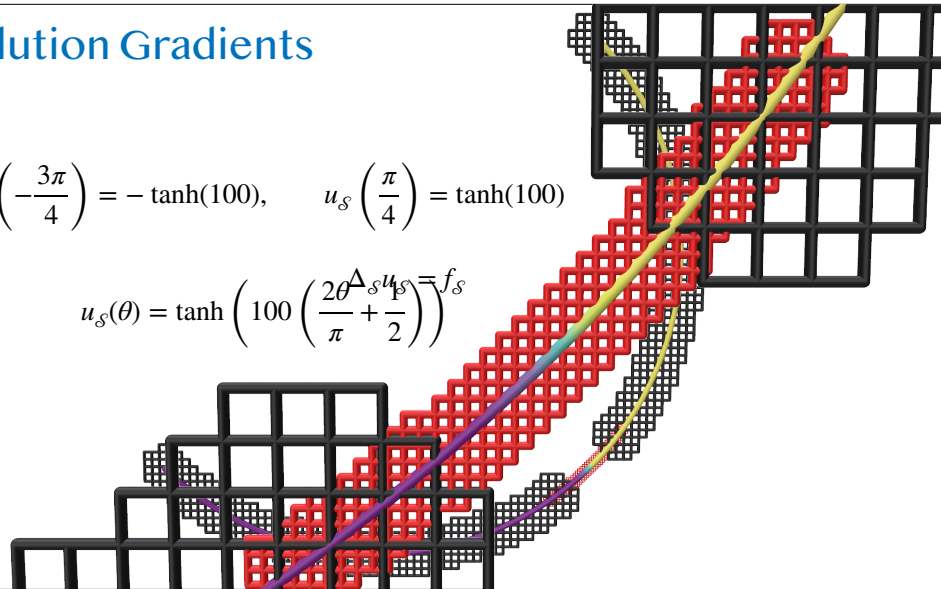


- (0:49) The bordering subset. Therefore, for $\mathbf{x}$ in $\mathcal{N}$ of partial S_1 we redefine (next) $E_1 u_S$ as $E_2 u_S$, which is (next) u of $\text{cp } S_2$
- In words, for points in a boundary subset, we use the closest point extension from the bordering subset instead of the naive extension.
- This connects the subsets, so the global problem can be solved.
- One thing that is nice with this adaptivity approach is that it should be straightforward to use with the domain decomposition I mentioned earlier
- Both use overlapping tubular subsets, but the domain decomposition paper used a single resolution

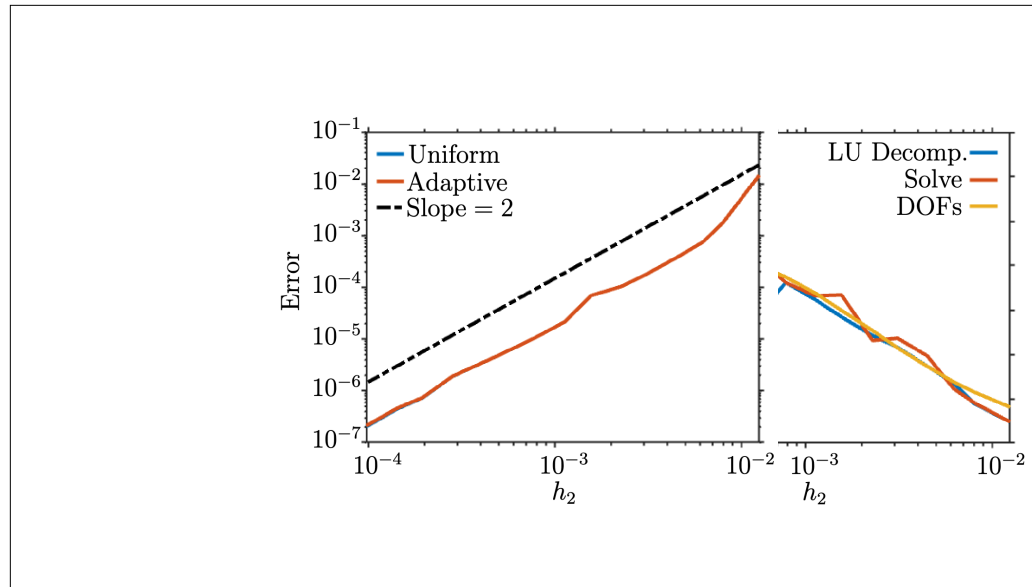
Solution Gradients

$$u_{\mathcal{S}}\left(-\frac{3\pi}{4}\right) = -\tanh(100), \quad u_{\mathcal{S}}\left(\frac{\pi}{4}\right) = \tanh(100)$$

$$u_{\mathcal{S}}(\theta) = \tanh\left(100\left(\frac{2\theta_{\mathcal{S}}^{\Delta} u_{\mathcal{S}}}{\pi} + \frac{1}{2}\right)\right) \bar{f}_{\mathcal{S}}$$



- (0:37) Adaptivity can be based on different criteria.
- In this example, the resolution of the grid is refined where the solution gradient is large. We solve the Poisson equation on this arc, (next) with Dirichlet boundary conditions that are approximately -1 and 1. (Next)
- The solution is this hyperbolic tangent function, (next) which has a large gradient at the middle of the arc where it transitions for -1 to 1.
- We compare using this adaptive grid with varying resolutions in the red region to (next) using a uniform grid with the same resolution as the red region over the whole arc.



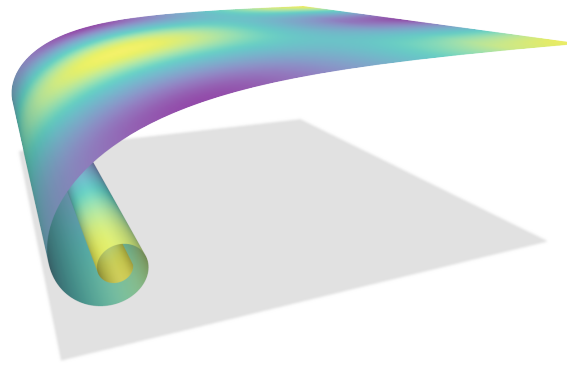
- (0:43) The convergence study here shows that the uniform and adaptive grids give nearly identical results since the orange and blue lines are on top of each other (Next).
- A direct solver is used in this example. Dividing quantities of the uniform grid approach by the corresponding adaptive ones, we see that as the grid resolution in the red region is refined, the LU Decomposition time and the number of degrees of freedom in the linear system scale together. At the smallest resolution, the speed up of LU decomposition and the reduction in the number of DOFs is almost a factor of 10. The solve time continues to increase further reaching almost a 14x speed up.

Local Feature Size

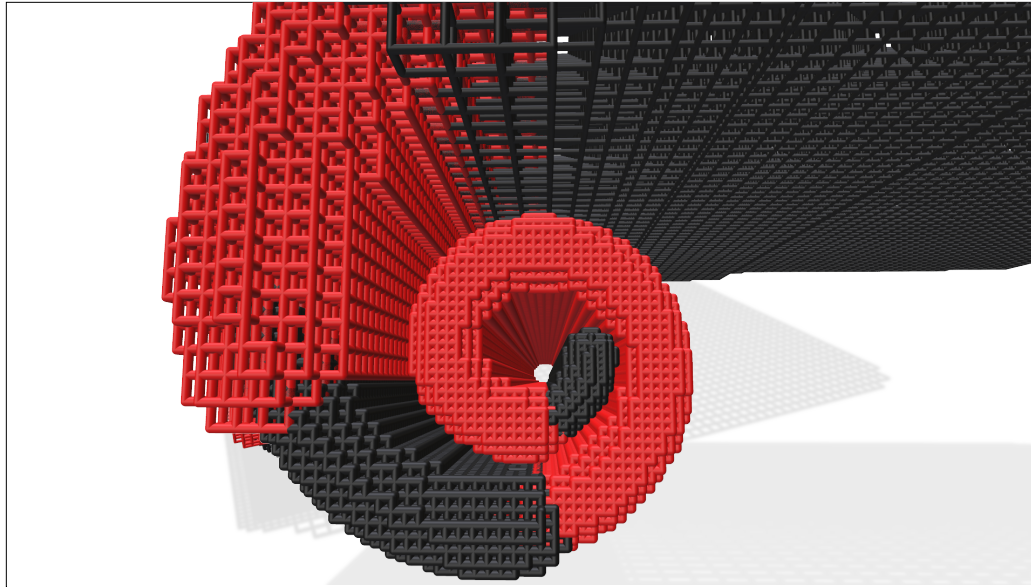
$$-\Delta_{\mathcal{S}} u_{\mathcal{S}} + u_{\mathcal{S}} = C \cos(\alpha s) \cos(\beta z)$$

$$\nabla_{\mathcal{S}} u_{\mathcal{S}} \cdot \mathbf{n} = 0$$

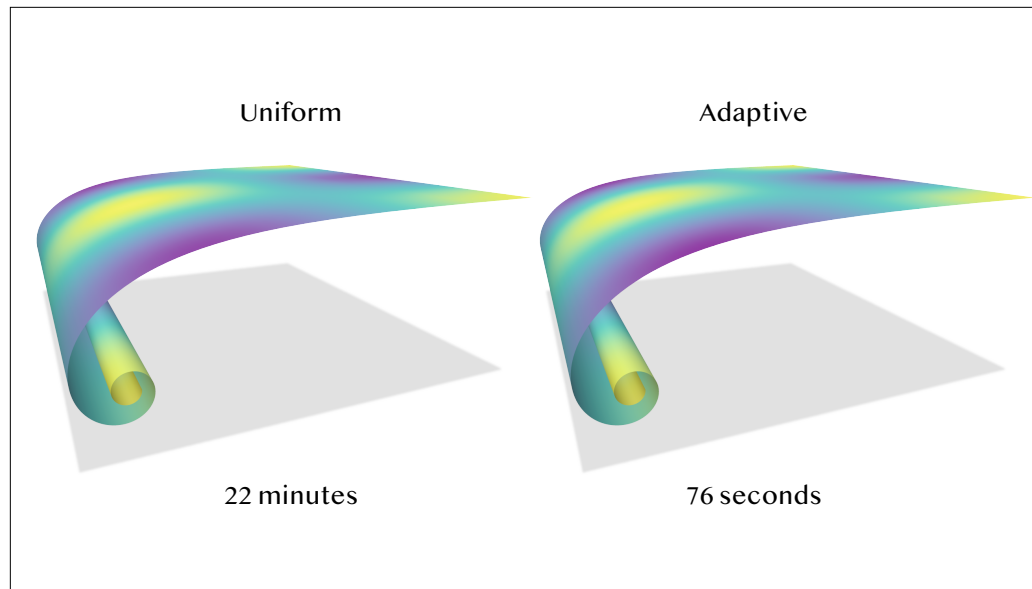
$$u_{\mathcal{S}}(s, z) = \cos(\alpha s) \cos(\beta z)$$



- (0:39) Another criterion for adaptivity is the local feature size, which tells us locally how much room there is in the embedding space for the grid. In this example, the spiral sheet, the resolution of the grid must be finer in the region where the sheet wraps tightly compared to the flat region further to the right.
- We solve the screened-Poisson equation of this form, (next) with homogeneous Neumann boundary conditions, (next) and the exact solution is a multiplication of cosines for the s parameter, which goes around the spiral, and z parameter that gives the depth of the sheet. (Next)



- (0:07) Here is the adaptive grid we use (next)
- And a zoom in of it (next)



- (0:24) On the left is the solution with a uniform grid of the finest resolution in the adaptive grid, and the adaptive result on the right.
- The two are visually indistinguishable, but the uniform result takes 22 minutes to compute, whereas the adaptive only takes 76 seconds, roughly a 17x speedup.

Future Work

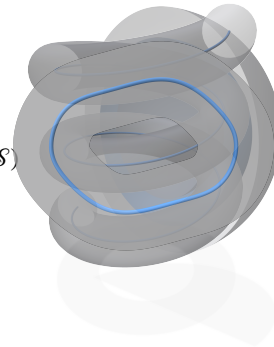
- There is a lot of interesting future work to be explored with the closest point method
- I will go through some open problems that I think are important

Optimal Adaptive Grid

- Minimize N subject to constraints (LFS, curvature, gradients, error estimates)

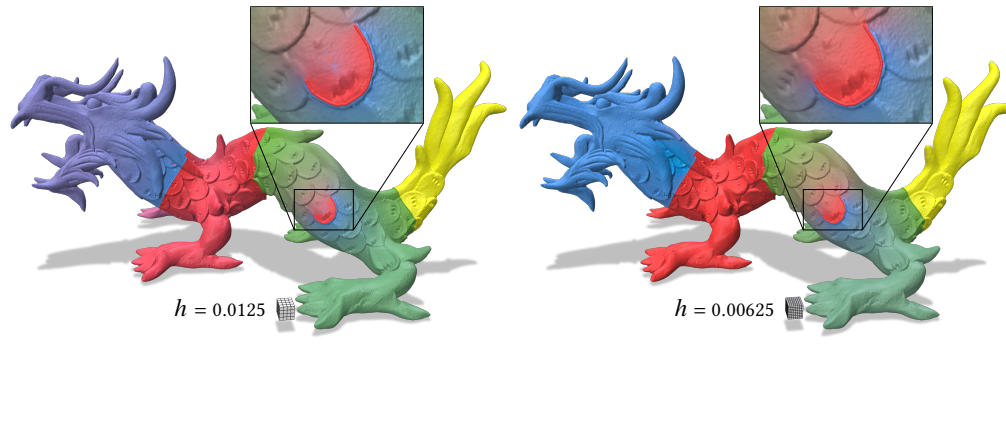
$$N \approx \sum_i \frac{\text{vol}(\mathcal{N}(\mathcal{S}_i))}{h_i^d}$$

$$\text{vol}(\mathcal{N}(\mathcal{S})) = 2r_{\Omega(\mathcal{S})}^2 \text{length}(\mathcal{S}) + \frac{4\pi r_{\Omega(\mathcal{S})}^3}{3} \chi(\mathcal{S})$$

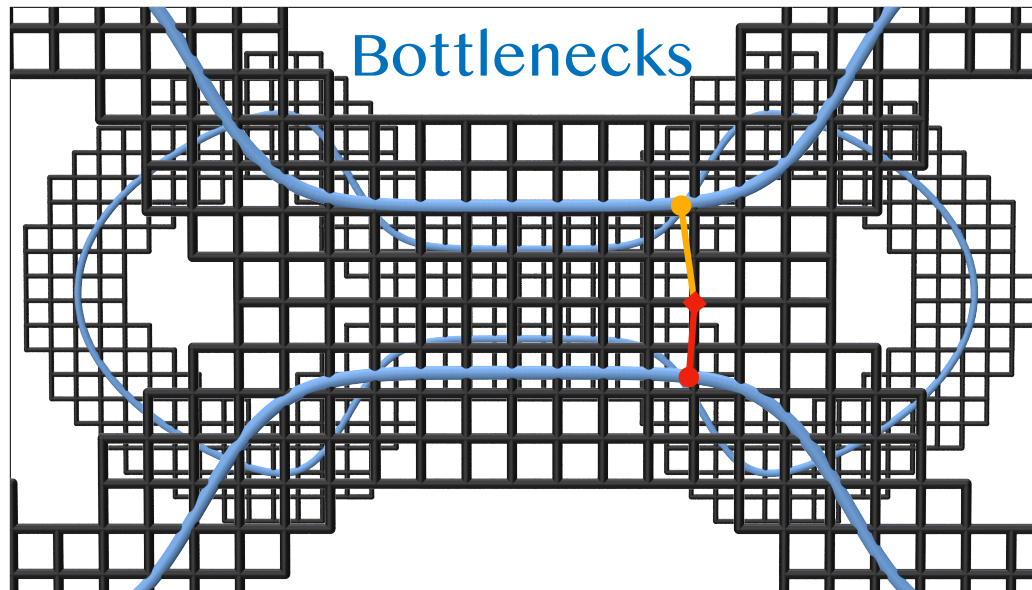


- Since we were just talking about adaptivity, let's start there
- The examples I showed used manually selected subsets, just to verify that the overall discretization with these overlapping computational tubes was correct and would improve efficiency
- But to choose these subsets with varying radii in an optimal way, you want to (next) minimize the number of grid points subject to some constraints
- These could be geometric constraints like the LFS and curvature
- Or it could be based on error estimates of the solution and a user prescribed threshold
- The number of grid points (next) can be estimated with the volume of the tube and dividing by the volume of one cell (next)
- Weyl's tube formula gives the volume in any dimension, for example for a 1D curve in 2D it just depends on the radius of the tube and the length of the curve (next)
- For a 1D curve embedded in 3D, it is just the circle swept along the curve, so πr^2 times the length
- For 2D surfaces, you have $2r$ times the area of S plus the volume of a sphere times the Euler characteristic of the surface
- but the open question is how to compute this numerically and how to solve the optimization problem (next)

A Priori Grid Spacing



- Somewhat related to adaptivity, but more generally just for the grid-based closest point method it would be great to know the required grid spacing h before you solve the problem
- Obviously the quantity you care about is different for different applications and different fields
- Graphics applications worry about visual correctness. Engineering applications worry about the accuracy of certain quantities
- In this example, the dragon surface is coloured using diffusion curves. With a coarse h on the left there are unwanted visual artifacts
- The head of the dragon is purple because the red region bleeds across the curve
- Also in the zoom in, there is a blue artifact in the centre of the red
- Halving h gives the desired solution, but to save computation time we wanted to solve with the coarse h first to see that it was small enough
- Refining h and manually checking the result after each reduction wastes time and requires manual intervention
- Having some way to determine what h should be before you solve is an interesting open problem, maybe it involves error estimates, or medial axis avoidance, or something else completely



- Another approach to improve efficiency besides adaptivity and a priori knowledge of h is apparent with bottleneck regions, where geodesically distant parts of the manifold are close in Euclidean space (next)
- What you could do here is instead of just using the closest point extension, you could consider extending multiple values to a single grid point (next)
- Basically store the value at all orthogonal intersection points. Then for stencils near the red point use one value, and stencils near the orange use the other
- The big question is how to compute all orthogonal intersection points to surfaces instead of just the closest point

Sharp Features and Discontinuous Solutions

Level set equations on surfaces via the Closest Point Method

Colin B. Macdonald¹ and Steven J. Ruuck²

May 30, 2007

Abstract

Level set methods have been used in a great number of applications in $\mathbb{R}^2$ and $\mathbb{R}^3$ and it is natural to consider extending some of these methods to problems defined on surfaces embedded in $\mathbb{R}^2$ or higher dimensions. In this paper we consider the treatment of level set equations on surfaces via a recent technique for solving partial differential equations (PDEs) on surfaces, the Closest Point Method (CPM). The main modification is to introduce a Weighted Essentially Non-Oscillatory (WENO) interpolation step into the Closest Point Method. This is combined with standard WENO for Hamilton-Jacobi equations, given elliptic partial differential equations as a source of smoothness for the level set equation. Numerical results for the CPM are presented and compared with those of the standard CPM. The algorithm is implemented in a software package and the results are compared with those of the standard CPM. The algorithm is implemented in a software package and the results are compared with those of the standard CPM. The algorithm is implemented in a software package and the results are compared with those of the standard CPM.

1 Introduction

The level set method (LSM) has been successfully applied to a tremendous variety of problems involving curve evolution in $\mathbb{R}^2$ or surface evolution in $\mathbb{R}^3$. This curve or surface, the interface, is represented as the zero contour of a level set function. A principal strength of the level set method comes from its ability to handle changes in topology of the evolving interface, i.e., interfaces break apart or merge naturally, and without the need for special code or intervention to detect or treat the change changes or their motion. A second, and also key, benefit that arises when using level set methods is that the discretization of the underlying set of equations for Hamilton-Jacobi type can be carried out using well-known, accurate and reliable discretization techniques, such as the weighted essentially non-oscillatory (WENO) methods described in [Liu94, 1996, 1998]. Taken together, these benefits have contributed to a widespread adoption of level set techniques in diverse disciplines [1991, 1997].

¹Department of Mathematics, Simon Fraser University, Burnaby, British Columbia, V5A 1S6 Canada (macdonald@math.sfu.ca). The work of this author was partially supported by a grant from NSERC Canada and a scholarship from the Pacific Institute of Mathematics (PIM).

²Department of Mathematics, Simon Fraser University, Burnaby, British Columbia, V5A 1S6 Canada (ruuck@math.sfu.ca). The work of this author was partially supported by a grant from NSERC Canada.

A localized method for diffusion on folded surfaces

Xu Chen (Chang)¹, Lennart Ling², Steve Ruuck²

Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

Abstract

Partial differential equations (PDEs) on surfaces arise in a variety of applications, including biological systems, material science, fluid dynamics, astrophysics, image processing and computer graphics. In this paper, we propose a novel, localized method for solving PDEs on surfaces. The method is based on the Closest Point Method (CPM) and is designed to handle sharp features and discontinuous solutions. The method is based on the Closest Point Method (CPM) and is designed to handle sharp features and discontinuous solutions. The method is based on the Closest Point Method (CPM) and is designed to handle sharp features and discontinuous solutions.

¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

An Embedding Technique for the Solution of Reaction-Diffusion Equations on Algebraic Surfaces with Isolated Singularities

Pierluigi Boscaini¹, Thomas Mörz², Steven J. Ruuck³

March 2012

Abstract

In this paper we present a novel embedding technique for numerically solving reaction-diffusion PDEs defined on algebraic surfaces that possess isolated singularities. The idea is to embed the surface into a higher dimensional space and solve the PDE there. The embedding is done by lifting the surface into a higher dimensional space and solving the PDE there. The embedding is done by lifting the surface into a higher dimensional space and solving the PDE there. The embedding is done by lifting the surface into a higher dimensional space and solving the PDE there.

¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

¹⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

²⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

³⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁴⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁵⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁶Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁷Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁸Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁶⁹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷⁰Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷¹Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

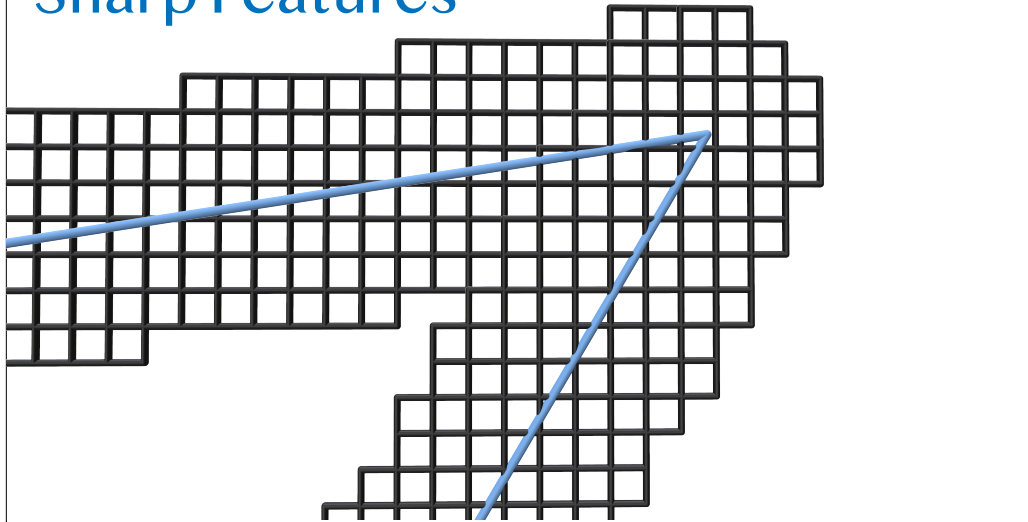
⁷²Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷³Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

⁷⁴Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

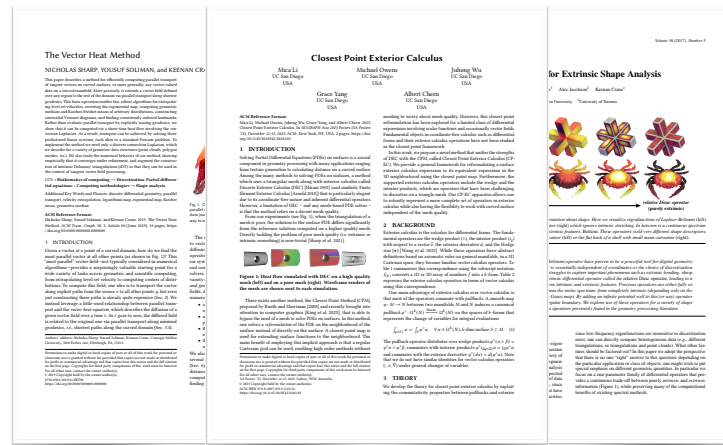
⁷⁵Department of Mathematics, Simon Fraser University, Burnaby, B.C. Canada

Sharp Features



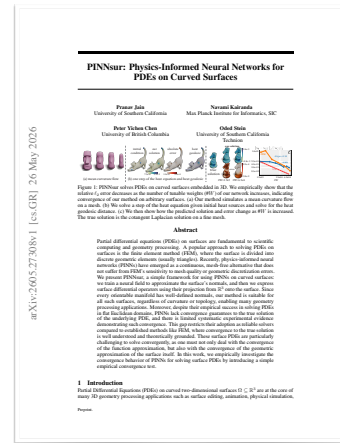
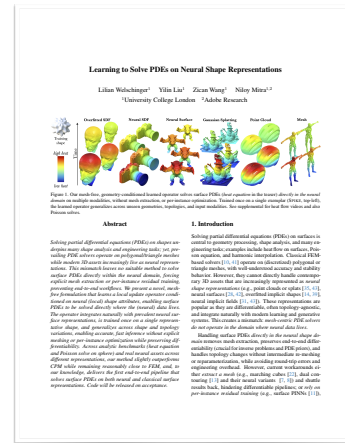
- But for a sharp feature you could decompose the manifold into piecewise smooth manifolds (next)
- Construct a tube around each piece (next) and solve the PDE with CPM on these smooth pieces
- The question is mainly how to couple these two pieces in the overlapping region (next)

Other Differential Operators



- The closest point method has been applied to many different PDEs with a variety of differential operators
- However, there are some differential operators that show up in geometry processing have not been considered yet
- For example, the connection Laplacian in the heat method and the Dirac operator for extrinsic shape analysis (next)
- There is this poster from SIGGRAPH Asia a few years ago that showed how to handle operators in the more general setting of exterior calculus, maybe that is a good starting point for other differential operators

Neural Representations



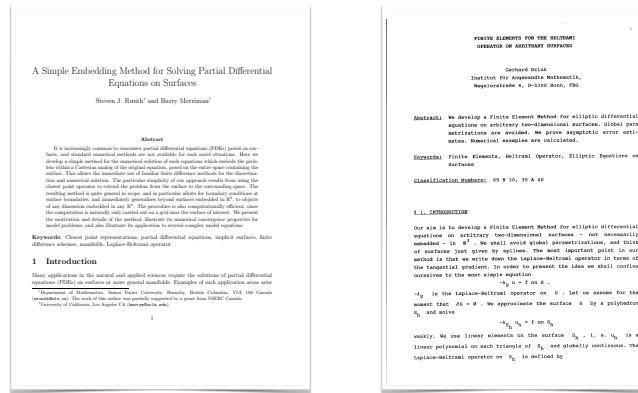
- Throughout the course we talk about the classical approach of discretizing your domain to solve the PDE. This discretization could be a mesh with the finite element method or a grid with the closest point method
- But they both have the same inherent limitation: the solution is resolution dependent
- The paper on the left replaced the closest point extension step with a neural network so it could be queried continuously
- The PDE was still solved on the grid however with finite differences
- This paper on the right, posted to arXiv, trained a neural network from the normals of a surface to be able to compute manifold differential operators based on their tangent space projection definition. Then they used another network for the PDE solver as a Physics Informed Neural Network
- The use the projection definition of the surface differential operator restricts the work to orientable manifolds.
- Having a fully continuous solver with the closest point representation would allow solving PDEs on general representations and types of manifolds

Monte Carlo



- Another approach for a continuous solver is to use Monte Carlo methods
- Monte Carlo PDE solvers have recently seen a surge in work since the Monte Carlo Geometry Processing paper by Sawhney and Crane in 2020
- Like the neural networks, Monte Carlo methods are a different approach with different benefits and disadvantages compared to the grid based and mesh based methods
- They are kinda in the class of closest point methods if you define the class as anything that uses a closest point representation
- The first Monte Carlo method to solve surface PDEs was in the left paper and it used conformal maps to construct a spatially varying coefficient PDE
- We combine the closest point method with walk-on-spheres in 2024 to solve surface PDEs without needing a conformal mapping in the right paper
- Since we have time, I will share some bonus slides about this Monte Carlo Closest Point Method
- There is however a key limitation in that paper that would be interesting to explore, you can check it out for more details

Conclusion



- The closest point method is relatively new, the first paper was published in 2008
- The finite element method can be traced back to 1941, and the first surface finite element paper was in 1988
- So surface finite elements have 20 years more of development plus almost 50 years (47) of volumetric development before that
- I think using closest points as a representation has lots of unexplored potential, not just in terms of PDE-based geometry processing
- There has been lots of work actually that have used closest points too, one example is the Iterative Closest Point Method to find a good match of two separate point clouds
- But in this course we explored a powerful framework for PDE-based geometry processing that has similar computation time and memory usage as mesh based methods, but it applies to any representation that allows closest point queries, works for general manifold types, and in any dimension and codimension



- A special thanks to my collaborators Rysuke and Haozhe. (Next) Their supervisors Toshiya and Mridul (next) and my supervisors Steve and Christopher.
- That's all, thanks for listening!

Monte Carlo CPM



Ryusuke Sugimoto

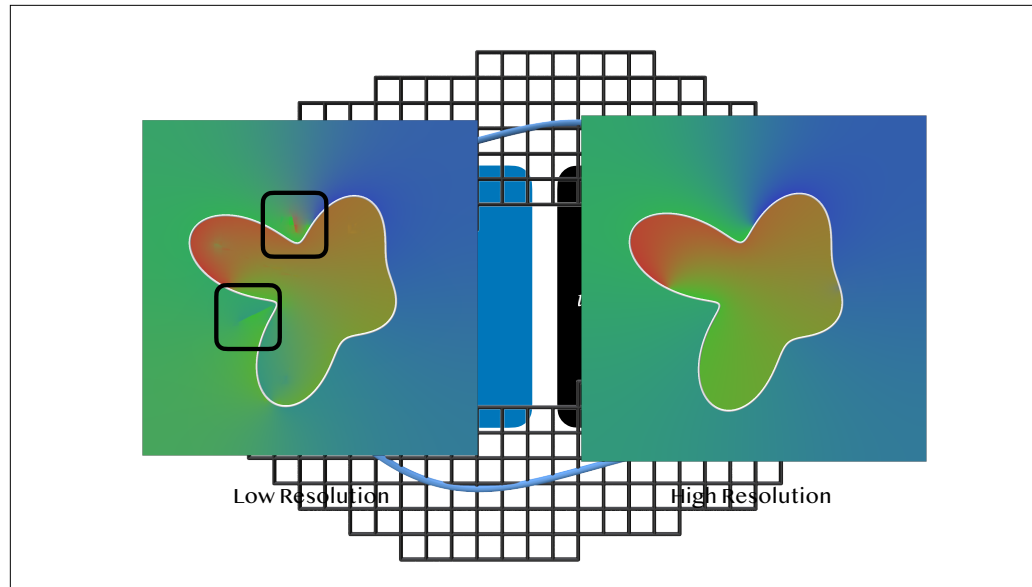


Toshiya Hachisuka



Christopher Batty

- This was work with my collaborators Ryusuke, Toshiya, and Christopher



- (0:46) Discretization based methods like the grid-based CPM require the domain to be sampled at a finite number of points. (Next)
- The equations for this finite set of points are then used to construct a global matrix system and this matrix system is solved
- But this approach makes the solution inherently resolution dependent. (Next)
- At low resolution the solution can have unwanted artifacts (next) and you can't just add new evaluation points after you solve the matrix equation
- With Monte Carlo methods, you don't need to solve a global matrix system and you can add new evaluation points at anytime since the solver allows for pointwise evaluation of the solution

Volumetric Walk-on-Spheres

- Solution satisfies

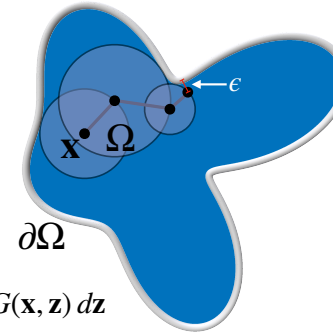
$$u(\mathbf{x}) = \frac{1}{|\partial B_r(\mathbf{x})|} \int_{\partial B_r(\mathbf{x})} u(\mathbf{y}) d\mathbf{y}$$

- Walk-on-spheres estimator

$$\hat{u}(\mathbf{x}_i) = \begin{cases} \hat{u}(\mathbf{x}_{i+1}), & \text{if } \mathbf{x}_i \notin \partial\Omega \\ g(\mathbf{x}_{i+1}), & \text{otherwise} \end{cases}$$

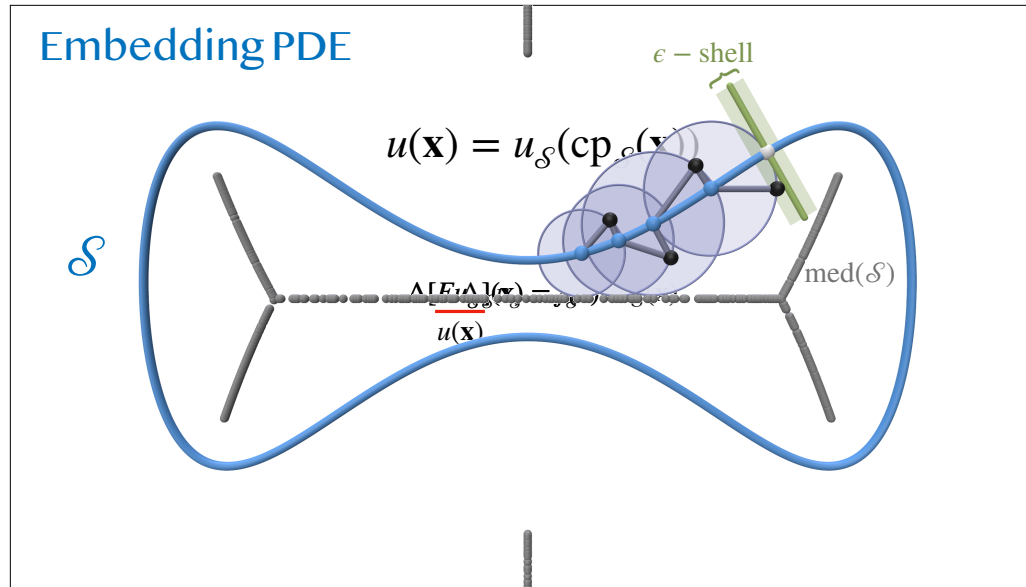
- Poisson equation solution satisfies

$$u(\mathbf{x}) = \frac{1}{|\partial B_r(\mathbf{x})|} \int_{\partial B_r(\mathbf{x})} u(\mathbf{y}) d\mathbf{y} + \int_{B_r(\mathbf{x})} f(\mathbf{z}) G(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$

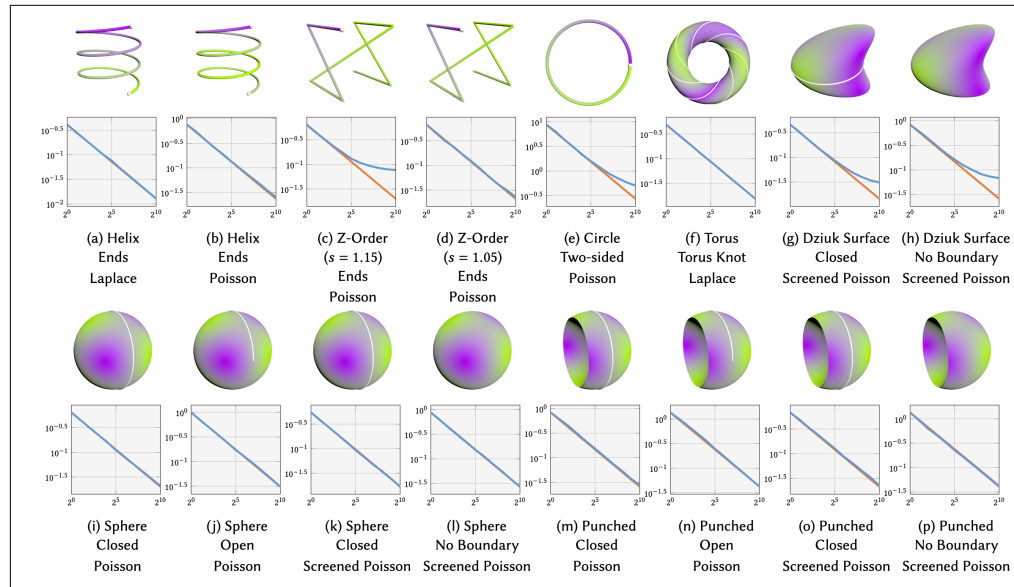


- (1:31) First I will review the volumetric walk-on-spheres method for the Laplace equation, that is defined on some domain Omega, in blue, with a Dirichlet boundary condition alpha, on the white boundary (Next)
- The solution satisfies the mean value property, that the value of u at x is equal to the average of the solution over any sphere centered at x that is completely contained in Omega (Next)
- The walk on spheres estimator uses this recursive equation that is iterated until x_i + 1 is within an epsilon distance to the boundary (next)
- Walk on spheres starts at the point x where you want the solution, (next) and finds the largest ball that fits inside the domain. (Next) Then a point is uniformly sampled from the sphere. The solution is also unknown at the sampled point, so we iterate again, (next) finding the largest ball centred at this point now. Another point is sampled from this sphere and so on, until the sampled point is less than epsilon away from the boundary.
- The Dirichlet boundary value alpha is then taken at the closest point on the boundary to this last sampled point. Many paths are sampled and averaged to obtain an accurate solution. (next)
- Additional integral terms are needed for other PDEs, for example, the solution of the Poisson equation satisfies this integral equation. So the Monte Carlo estimators are PDE dependent, which is one of its limitations

Embedding PDE

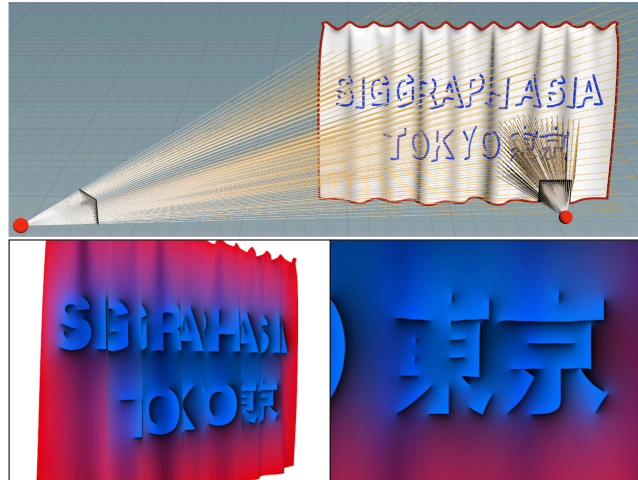


- (1:55) For manifold PDEs, we would like to have something similar.
- My intuitive idea was that we should be able to use a volumetric Monte Carlo method in the embedding space to solve (next) an embedding PDE, that is constructed using principles of the closest point method
- Here the unknown function we are solving for is (next) the closest point extension of $u|_S$, which is just u and f is the closest point extension of a known function given in the definition of the PDE.
- Unfortunately, this embedding PDE is actually ill-posed. Because f is constant in the normal direction, but the LHS is not due to the Laplacian. (Next) Therefore, an additional unknown term, g of x , must be added to the RHS to account for this discrepancy.
- We are given the manifold S in blue (next) and the Dirichlet boundary C in white (next). We extend the manifold PDE to the embedding space, and also (next) extend the Dirichlet boundary condition at C . (Next) An epsilon shell is placed around the extended C and the medial axis is computed. (Next)
- We want the solution at this (next) point x , so we perform walk on spheres in the embedding space with the sphere size restricted by the distance from x to the medial axis, (next) so the closest point function is unique within the sphere. (Next) A point is sampled on this sphere. (Next) And from the closest point extension property, we know the solution u of x is equal to $u|_S$ at cp of x , so we can project the sampled point back onto the manifold before the next iteration.
- We continue sampling and projecting back onto the manifold until the sampled point is in the epsilon shell and the Dirichlet condition is taken. (Next, next, next)
- As you get close to the extend boundary the sphere radius is also restricted by the distance to the boundary.



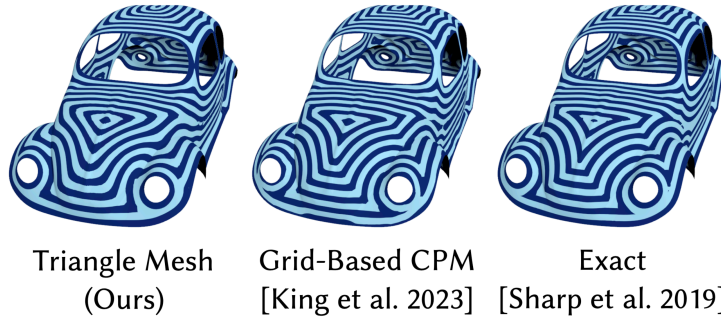
- (0:23) We studied the convergence for many different analytical problems
- It can be seen that the convergence for some problems plateau, which is indicative of a biased estimator. This is because we assume the g function is 0. Determining a more accurate g function is the main open problem with this approach.
- The bias occurs for problems with complex f or manifolds with high curvature
- But our method converges as expected for certain problems and is still useful for many applications

View-Dependent Diffusion Curves



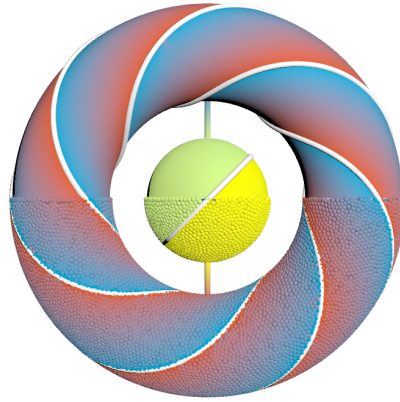
- (0:15) Due to the ability of our method to compute the solution locally at specific points we can compute view dependent solutions, like in this diffusion curves example.
- The solution is only evaluated at points on the surface that intersect with the viewing rays

Geodesic Distance



- (0:19) Also, we are able to compute the solution with different representations, like a mesh and point cloud in this example. (Next) Qualitatively, our Monte Carlo approach matches the result from the grid-based closest point method and exact polyhedral geodesic algorithm implemented in geometry central (next)

Diffusion Curves



- (0:11) Because we only need cp queries, we can handle mixed representations, like this mixture of a point cloud, triangle mesh, and polylines