High-order Solution Transfer between Curved Meshes and Ill-conditioned Bézier Curve Intersection

Danny Hermes

August 9, 2018

dhermes@berkeley.edu
UC Berkeley

Outline

1. Introduction and motivation
2. Curved Elements
3. Solution Transfer
4. Ill-conditioned Bézier Curve Intersection
5. Compensated Evaluation
6. Modified Newton's for Intersection

Introduction and motivation

Method of Characteristics

Method of Characteristics

Solve simple transport equation

$$u_t + cu_x = 0, \quad u(x,0) = u_0(x).$$

Method of Characteristics

Solve simple transport equation

$$u_t + cu_x = 0, \quad u(x,0) = u_0(x).$$

Divide physical domain

$$x(t) = x_0 + ct$$

Method of Characteristics

Solve simple transport equation

$$u_t + cu_x = 0, \quad u(x,0) = u_0(x).$$

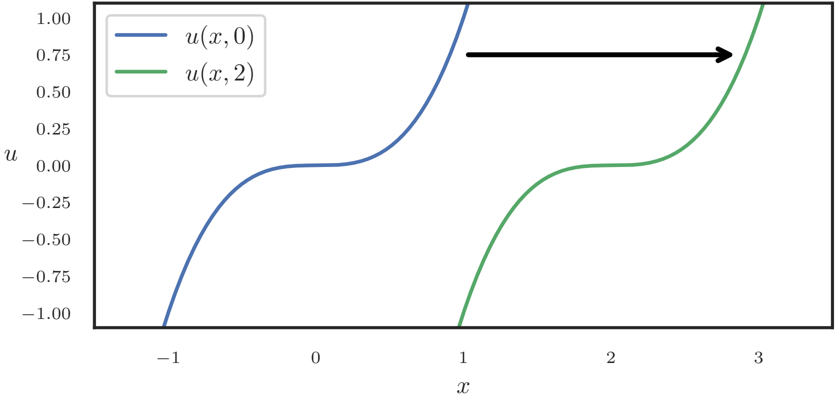
Divide physical domain

$$x(t) = x_0 + ct$$

PDE becomes a (trivial) ODE

$$\frac{d}{dt} u(x(t), t) = 0.$$

Method of Characteristics



Lagrangian Methods

Lagrangian Methods

- Each point in physical domain is a **particle**

Lagrangian Methods

- Each point in physical domain is a **particle**
- Carry value (e.g. heat, pressure, density) along characteristic curve

Lagrangian Methods

- Each point in physical domain is a **particle**
- Carry value (e.g. heat, pressure, density) along characteristic curve
- Transform PDE to family of ODEs

Add viscosity term to transport equation

$$u_t + cu_x - \varepsilon u_{xx} = 0.$$

Add viscosity term to transport equation

$$u_t + cu_x - \varepsilon u_{xx} = 0.$$

Same characteristics used, **but** solution no longer constant

Add viscosity term to transport equation

$$u_t + cu_x - \varepsilon u_{xx} = 0.$$

Same characteristics used, **but** solution no longer constant

$$\frac{d}{dt} u(x(t), t) = \varepsilon u_{xx}.$$

- Problems caused by flow-based mesh changes

Remeshing and Adaptivity

- Problems caused by flow-based mesh changes
    - Distortion

Remeshing and Adaptivity

- Problems caused by flow-based mesh changes
  - Distortion
  - Tangling

Remeshing and Adaptivity

- Problems caused by flow-based mesh changes
  - Distortion
  - Tangling
  - Travel outside relevant physical domain

Remeshing and Adaptivity

- Problems caused by flow-based mesh changes
  - Distortion
  - Tangling
  - Travel outside relevant physical domain
- Adaptivity

Remeshing and Adaptivity

- Problems caused by flow-based mesh changes
  - Distortion
  - Tangling
  - Travel outside relevant physical domain
- Adaptivity
  - Dynamically focus computational effort

Remeshing and Adaptivity

- Problems caused by flow-based mesh changes
  - Distortion
  - Tangling
  - Travel outside relevant physical domain
- Adaptivity
  - Dynamically focus computational effort
  - Resolve sensitive features

Remeshing Example

Consider

$$u_t + \begin{bmatrix} y^2 \\ 1 \end{bmatrix} \cdot \nabla u + F(u, \nabla u) = 0$$

Remeshing Example

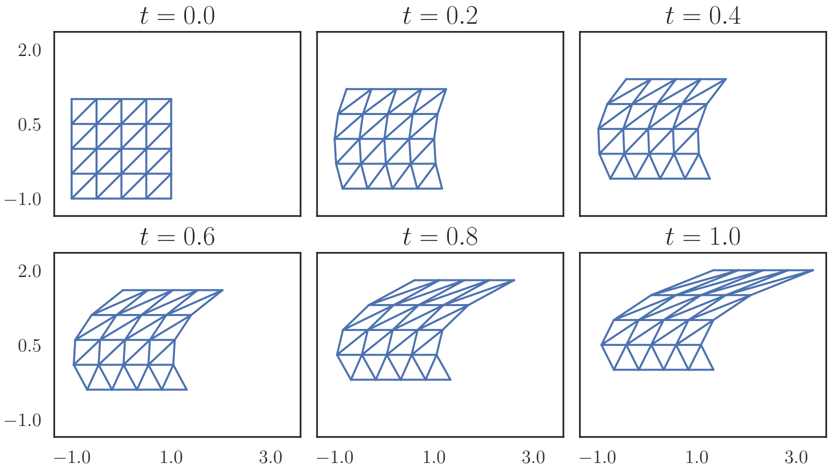
Consider

$$u_t + \begin{bmatrix} y^2 \\ 1 \end{bmatrix} \cdot \nabla u + F(u, \nabla u) = 0$$

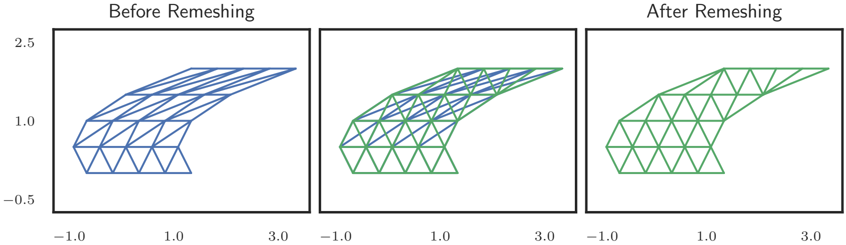
with cubic characteristics

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \frac{1}{3} \begin{bmatrix} (y_0 + t)^3 - y_0^3 \\ 3t \end{bmatrix}.$$

Remeshing Example



Remeshing Example



Curved Meshes

- Benefits

Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions

Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions
  - Low dissipation and dispersion error

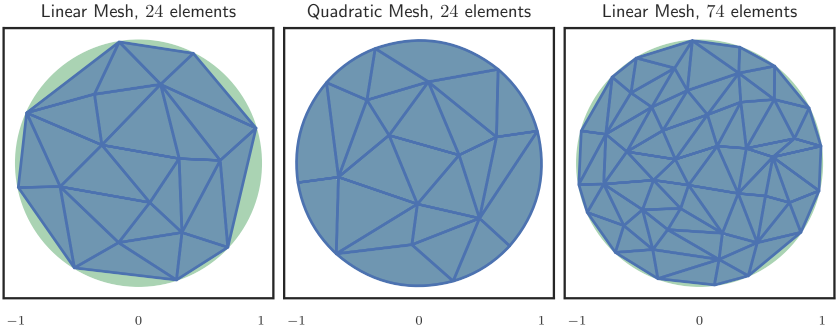
Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions
  - Low dissipation and dispersion error
  - Greater geometric flexibility

Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions
  - Low dissipation and dispersion error
  - Greater geometric flexibility
  - Fewer elements

Curved Meshes



Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions
  - Low dissipation and dispersion error
  - Greater geometric flexibility
  - Fewer elements

Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions
  - Low dissipation and dispersion error
  - Greater geometric flexibility
  - Fewer elements
- Drawbacks

Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions
  - Low dissipation and dispersion error
  - Greater geometric flexibility
  - Fewer elements
- Drawbacks
  - Harder to implement

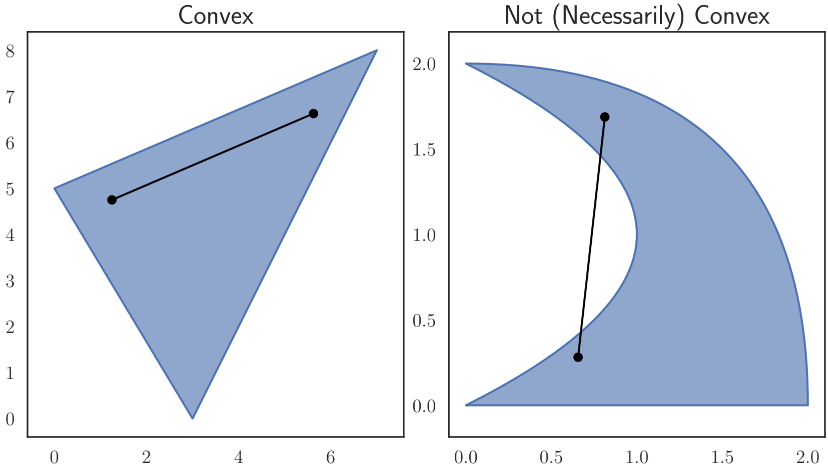
Curved Meshes

- Benefits
    - High-order shape functions, highly accurate solutions
    - Low dissipation and dispersion error
    - Greater geometric flexibility
    - Fewer elements
- Drawbacks
    - Harder to implement
    - Loss of accuracy in high degree (e.g. Runge's phenomenon)

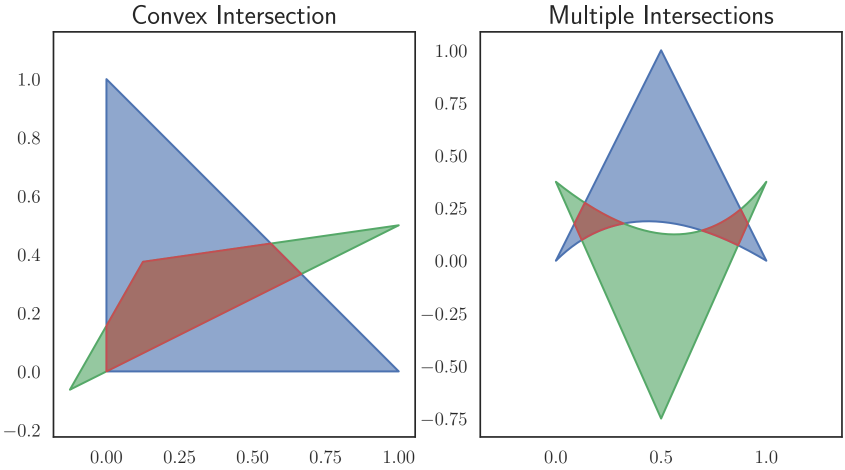
Curved Meshes

- Benefits
  - High-order shape functions, highly accurate solutions
  - Low dissipation and dispersion error
  - Greater geometric flexibility
  - Fewer elements
- Drawbacks
  - Harder to implement
  - Loss of accuracy in high degree (e.g. Runge's phenomenon)
  - More challenging geometry

Curved Meshes



Curved Meshes



Curved Elements

- Necessary for High-order

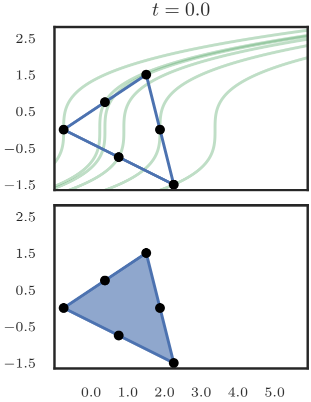
Curved Elements

- Necessary for High-order
- With non-linear shape functions (i.e. not straight sided), non-vertex nodes used

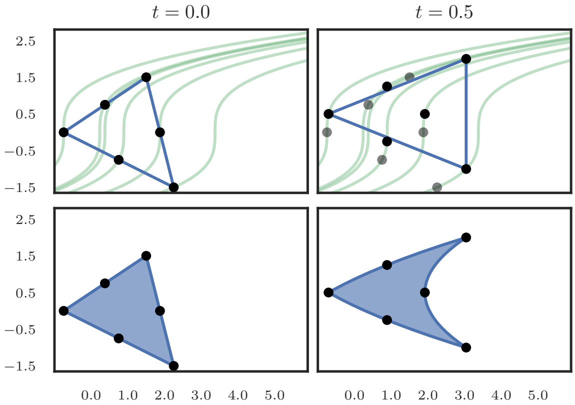
Curved Elements

- Necessary for High-order
- With non-linear shape functions (i.e. not straight sided), non-vertex nodes used
- Lagrangian method must either curve mesh or information about flow of geometry will be lost

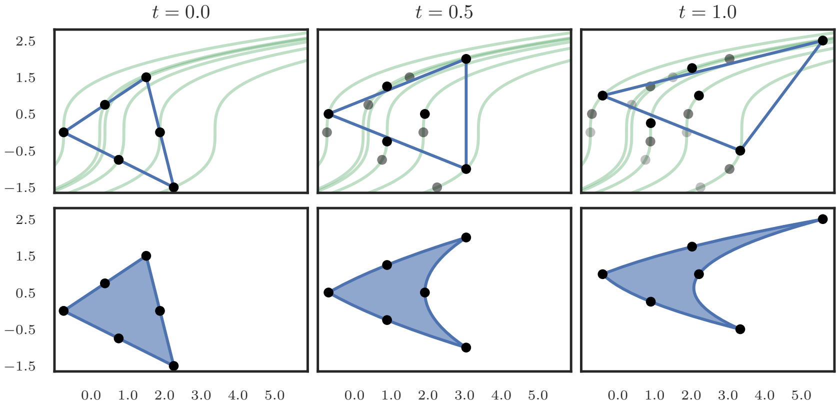
Curved Elements: Necessary for High-order



Curved Elements: Necessary for High-order



Curved Elements: Necessary for High-order



Curved Elements

Bézier Triangles

- Image $\mathcal{T} = b\,(\mathcal{U})$ of reference triangle under polynomial map $b(s, t)$

Bézier Triangles

- Image $\mathcal{T} = b\,(\mathcal{U})$ of reference triangle under polynomial map $b(s, t)$
- Barycentric coordinates $\lambda_1 = 1 - s - t, \lambda_2 = s, \lambda_3 = t$

Bézier Triangles

- Image $\mathcal{T} = b(\mathcal{U})$ of reference triangle under polynomial map $b(s, t)$
- Barycentric coordinates $\lambda_1 = 1 - s - t, \lambda_2 = s, \lambda_3 = t$
- Bernstein basis via trinomial expansion:

$$1 = (\lambda_1 + \lambda_2 + \lambda_3)^n$$

Bézier Triangles

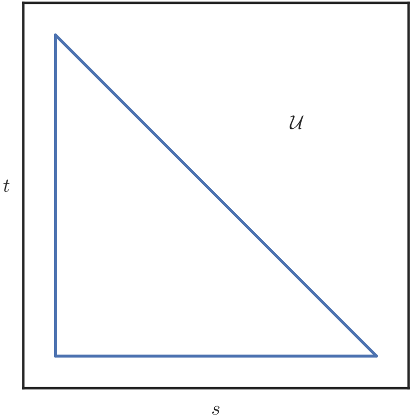
- Image $\mathcal{T} = b\,(\mathcal{U})$ of reference triangle under polynomial map $b(s, t)$
- Barycentric coordinates $\lambda_1 = 1 - s - t, \lambda_2 = s, \lambda_3 = t$
- Bernstein basis via trinomial expansion:

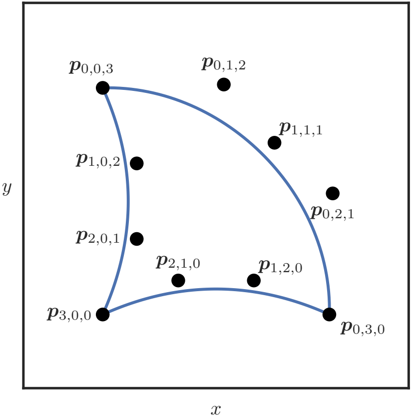
$$1 = (\lambda_1 + \lambda_2 + \lambda_3)^n$$

- Convex combination of control points

$$b(s, t) = \sum_{\substack{i+j+k=n \\ i,j,k \geq 0}} \binom{n}{i, j, k} \lambda_1^i \lambda_2^j \lambda_3^k \, \boldsymbol{p}_{i,j,k}$$

Bézier Triangles





Bézier Triangles

- $b(s, t)$ can be defined by data other than control net

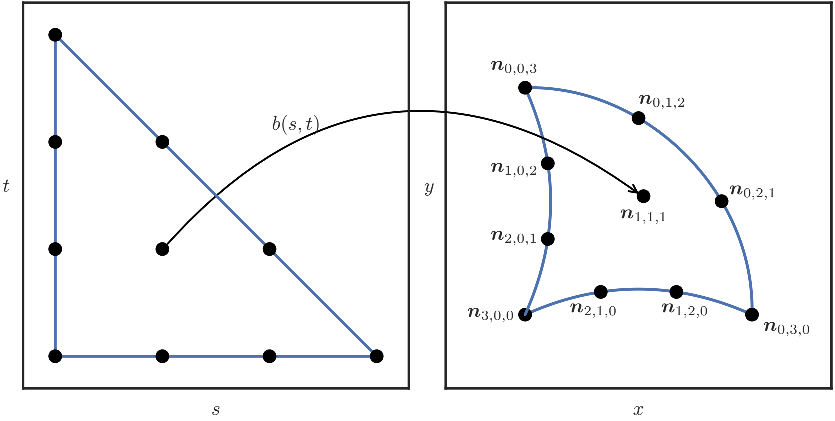
Bézier Triangles

- $b(s, t)$ can be defined by data other than control net
- Regular grid in $\mathcal{U}$, $\boldsymbol{u}_{i,j,k} = \left(\frac{j}{n}, \frac{k}{n}\right)$

Bézier Triangles

- $b(s,t)$ can be defined by data other than control net
- Regular grid in $\mathcal{U}$, $\boldsymbol{u}_{i,j,k} = \left(\frac{j}{n}, \frac{k}{n}\right)$
- $b\left(\boldsymbol{u}_{i,j,k}\right) = \boldsymbol{n}_{i,j,k}$; refer to $\boldsymbol{n}_{i,j,k}$ as **standard nodes**

Bézier Triangles



Bézier Triangles

- $b(s, t)$ can be defined by data other than control net
- Regular grid in $\mathcal{U}$, $\boldsymbol{u}_{i,j,k} = \left(\frac{j}{n}, \frac{k}{n}\right)$
- $b\left(\boldsymbol{u}_{i,j,k}\right) = \boldsymbol{n}_{i,j,k}$; refer to $\boldsymbol{n}_{i,j,k}$ as **standard nodes**

Bézier Triangles

- $b(s, t)$ can be defined by data other than control net
- Regular grid in $\mathcal{U}$, $\boldsymbol{u}_{i,j,k} = \left(\frac{j}{n}, \frac{k}{n}\right)$
- $b\left(\boldsymbol{u}_{i,j,k}\right) = \boldsymbol{n}_{i,j,k}$; refer to $\boldsymbol{n}_{i,j,k}$ as **standard nodes**
- For example, taking $\boldsymbol{n}_{i,j,k} = \delta_{(i,j,k)\ (i_0,j_0,k_0)}$ gives degree n shape functions on $\mathcal{U}$

Bézier Triangles

- $b(s, t)$ can be defined by data other than control net
- Regular grid in $\mathcal{U}$, $\boldsymbol{u}_{i,j,k} = \left(\frac{j}{n}, \frac{k}{n}\right)$
- $b\left(\boldsymbol{u}_{i,j,k}\right) = \boldsymbol{n}_{i,j,k}$; refer to $\boldsymbol{n}_{i,j,k}$ as **standard nodes**
- For example, taking $\boldsymbol{n}_{i,j,k} = \delta_{(i,j,k)\ (i_0,j_0,k_0)}$ gives degree n shape functions on $\mathcal{U}$
- Conversion between $\boldsymbol{n}_{i,j,k}$ and $\boldsymbol{p}_{i,j,k}$ has condition number exponential in n

Valid Element

- Element $\mathcal{T}$ is **valid** if diffeomorphic to $\mathcal{U}$

Valid Element

- Element $\mathcal{T}$ is **valid** if diffeomorphic to $\mathcal{U}$
- $b(s,t)$ bijective, i.e. Jacobian Db is everywhere invertible

Valid Element

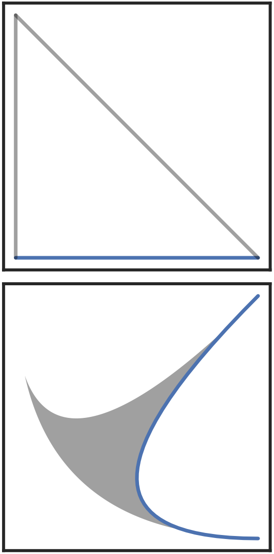
- Element $\mathcal{T}$ is **valid** if diffeomorphic to $\mathcal{U}$
- $b(s, t)$ bijective, i.e. Jacobian Db is everywhere invertible
- $\det(Db)$ positive, preserves orientation

Inverted Element

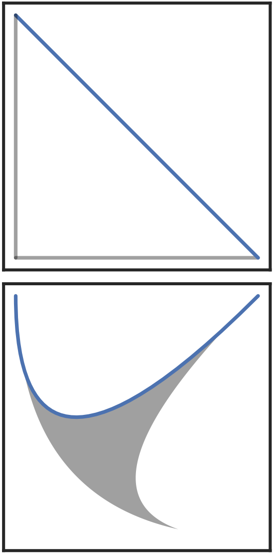
Consider element given by map

$$b(s,t) = \lambda_1^2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \lambda_2^2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \lambda_3^2 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

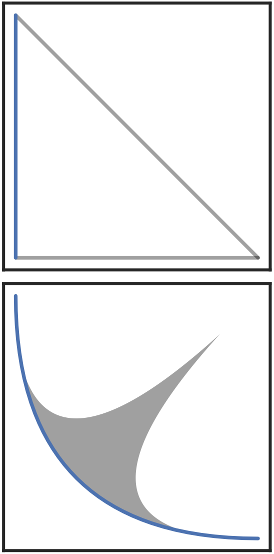
Inverted Element



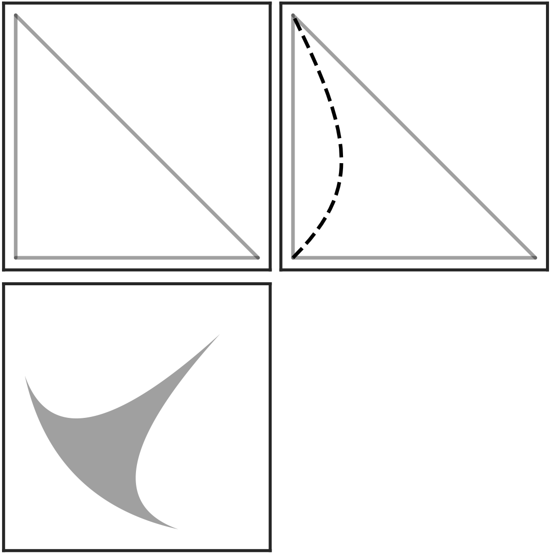
Inverted Element



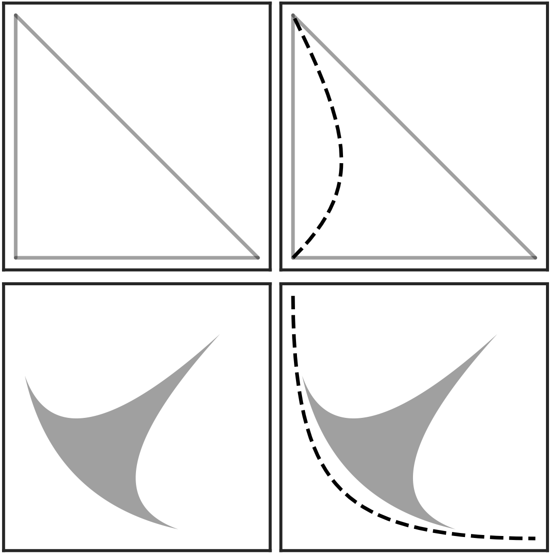
Inverted Element



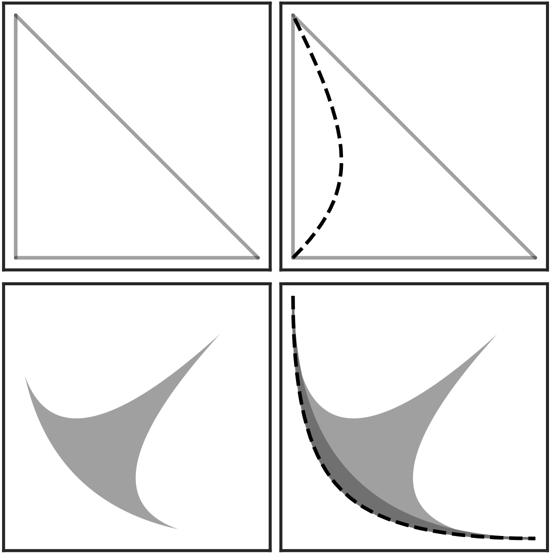
Inverted Element



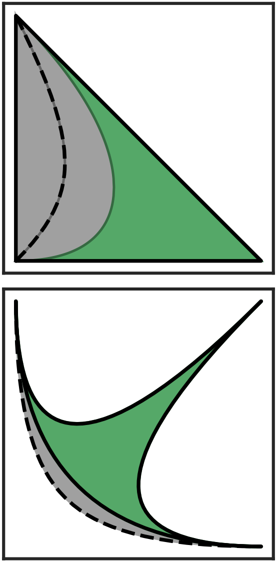
Inverted Element



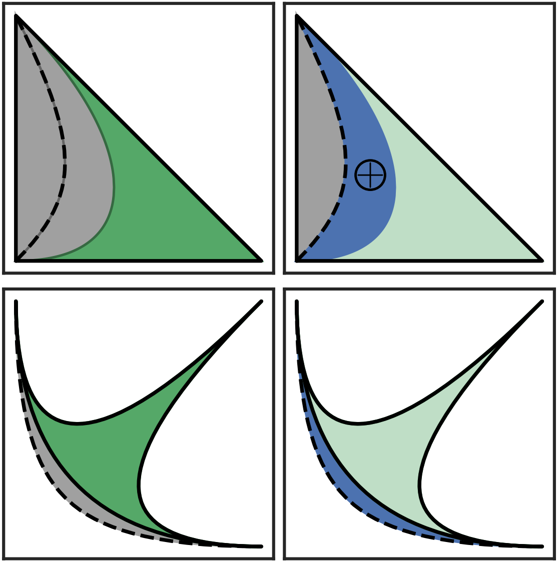
Inverted Element



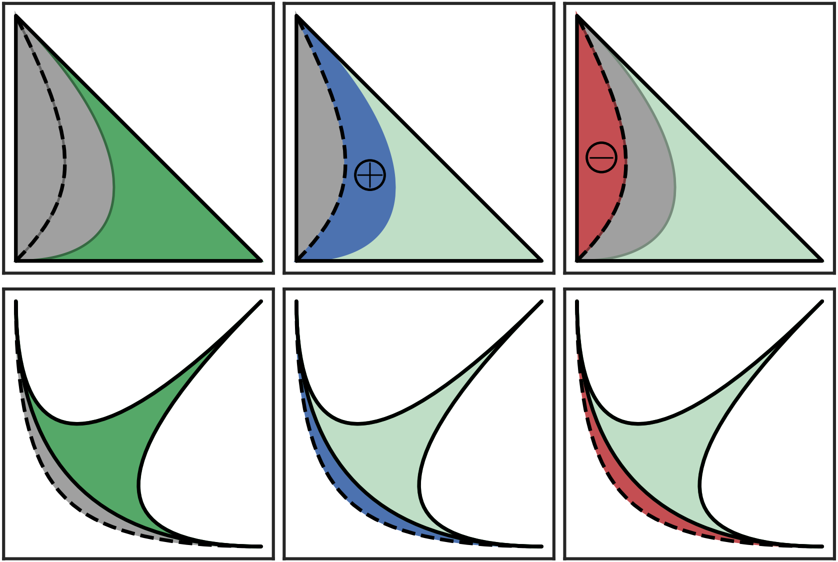
Inverted Element



Inverted Element



Inverted Element



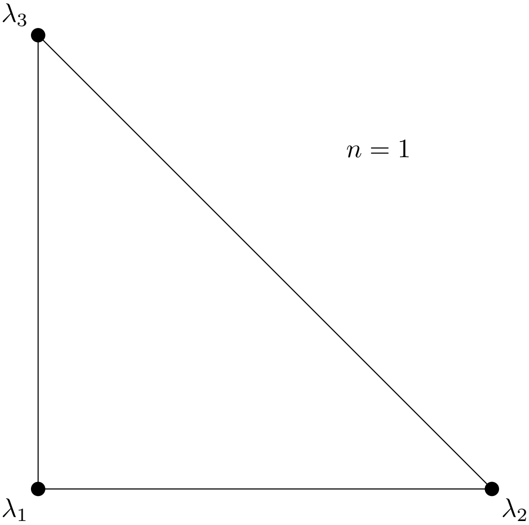
Shape Functions

- Based on $\boldsymbol{u}_{\boldsymbol{\alpha}} \in \mathcal{U}$ or $\boldsymbol{n}_{\boldsymbol{\alpha}} \in \mathbf{R}^2$ ($\boldsymbol{\alpha}$ is a multi-index)

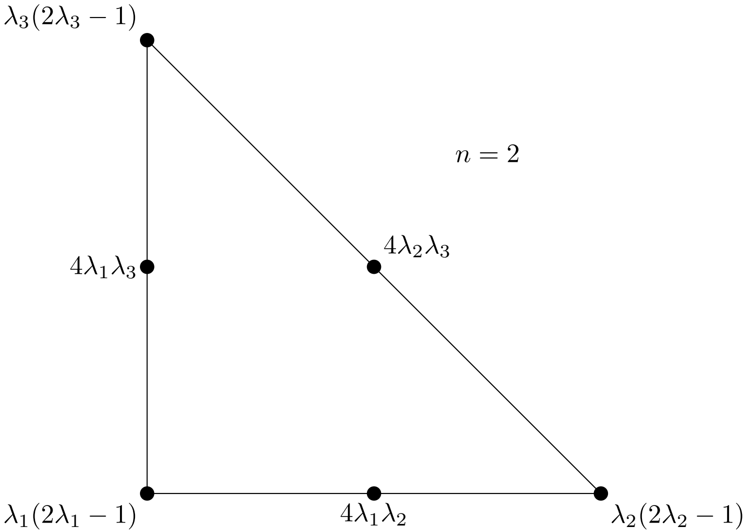
Shape Functions

- Based on $\boldsymbol{u}_{\boldsymbol{\alpha}} \in \mathcal{U}$ or $\boldsymbol{n}_{\boldsymbol{\alpha}} \in \mathbf{R}^2$ ($\boldsymbol{\alpha}$ is a multi-index)
- Pre-Image Basis: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n}_{\boldsymbol{\beta}}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(\boldsymbol{u}_{\boldsymbol{\beta}}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(b^{-1}\left(\boldsymbol{n}_{\boldsymbol{\beta}}\right)\right)$

Shape Functions



Shape Functions



Shape Functions

- Based on $\boldsymbol{u}_{\boldsymbol{\alpha}} \in \mathcal{U}$ or $\boldsymbol{n}_{\boldsymbol{\alpha}} \in \mathbf{R}^2$ ($\boldsymbol{\alpha}$ is a multi-index)
- Pre-Image Basis: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n}_{\boldsymbol{\beta}}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(\boldsymbol{u}_{\boldsymbol{\beta}}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(b^{-1}\left(\boldsymbol{n}_{\boldsymbol{\beta}}\right)\right)$

Shape Functions

- Based on $\boldsymbol{u_\alpha} \in \mathcal{U}$ or $\boldsymbol{n_\alpha} \in \mathbf{R}^2$ ($\boldsymbol{\alpha}$ is a multi-index)
- Pre-Image Basis: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n_\beta}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(\boldsymbol{u_\beta}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(b^{-1}\left(\boldsymbol{n_\beta}\right)\right)$
- Global Coordinates Basis: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n_\beta}\right) = \delta_{\boldsymbol{\alpha\beta}}$

Shape Functions

- Based on $\boldsymbol{u_\alpha} \in \mathcal{U}$ or $\boldsymbol{n_\alpha} \in \mathbf{R}^2$ ($\boldsymbol{\alpha}$ is a multi-index)
- Pre-Image Basis: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n_\beta}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(\boldsymbol{u_\beta}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(b^{-1}\left(\boldsymbol{n_\beta}\right)\right)$
- Global Coordinates Basis: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n_\beta}\right) = \delta_{\boldsymbol{\alpha\beta}}$
- Element is **isoparametric** when numerical solution expressed in span of shape functions

Shape Functions

- Based on $\boldsymbol{u}_{\boldsymbol{\alpha}} \in \mathcal{U}$ or $\boldsymbol{n}_{\boldsymbol{\alpha}} \in \mathbf{R}^2$ ($\boldsymbol{\alpha}$ is a multi-index)
- **Pre-Image Basis**: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n}_{\boldsymbol{\beta}}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(\boldsymbol{u}_{\boldsymbol{\beta}}\right) = \widehat{\phi}_{\boldsymbol{\alpha}}\left(b^{-1}\left(\boldsymbol{n}_{\boldsymbol{\beta}}\right)\right)$
- **Global Coordinates Basis**: $\phi_{\boldsymbol{\alpha}}\left(\boldsymbol{n}_{\boldsymbol{\beta}}\right) = \delta_{\boldsymbol{\alpha}\boldsymbol{\beta}}$
- Element is **isoparametric** when numerical solution expressed in span of shape functions
- $\operatorname{supp}(\phi) = \mathcal{T}$

Solution Transfer

Galerkin Projection

- Given:

Galerkin Projection

- Given:
  - Donor mesh $\mathcal{M}_D$ and target mesh $\mathcal{M}_T$

Galerkin Projection

- Given:
  - Donor mesh $\mathcal{M}_D$ and target mesh $\mathcal{M}_T$
  - Shape function bases $\phi_D^{(j)}$ and $\phi_T^{(j)}$

Galerkin Projection

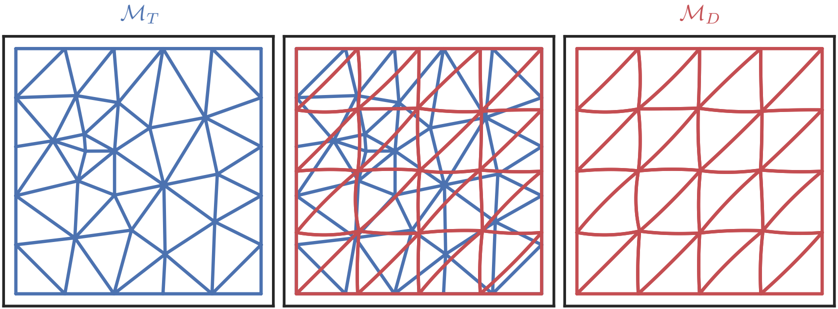
- **Given**:
  - Donor mesh $\mathcal{M}_D$ and target mesh $\mathcal{M}_T$
  - Shape function bases $\phi_D^{(j)}$ and $\phi_T^{(j)}$
  - Known discrete field $\boldsymbol{q}_D = \sum_j d_j \phi_D^{(j)}$

Galerkin Projection

- **Given**:
  - Donor mesh $\mathcal{M}_D$ and target mesh $\mathcal{M}_T$
  - Shape function bases $\phi_D^{(j)}$ and $\phi_T^{(j)}$
  - Known discrete field $\boldsymbol{q}_D = \sum_j d_j \phi_D^{(j)}$
- **Want**: L_2-optimal interpolant $\boldsymbol{q}_T = \sum_j t_j \phi_T^{(j)}$:

$$\|\boldsymbol{q}_T - \boldsymbol{q}_D\|_2 = \min_{\boldsymbol{q} \in \mathcal{V}_T} \|\boldsymbol{q} - \boldsymbol{q}_D\|_2$$

Galerkin Projection



Galerkin Projection

Differentiating w.r.t. each t_j in $\boldsymbol{q}_T = \sum_j t_j \phi_T^{(j)}$ gives **weak form**

$$\int_\Omega \boldsymbol{q}_D \phi_T^{(j)} \, dV = \int_\Omega \boldsymbol{q}_T \phi_T^{(j)} \, dV, \qquad \text{for all } j.$$

Galerkin Projection

Differentiating w.r.t. each t_j in $\boldsymbol{q}_T = \sum_j t_j \phi_T^{(j)}$ gives **weak form**

$$\int_\Omega \boldsymbol{q}_D \phi_T^{(j)} \, dV = \int_\Omega \boldsymbol{q}_T \phi_T^{(j)} \, dV, \qquad \text{for all } j.$$

If $(\boldsymbol{x} \mapsto 1) \in \mathcal{V}_T$, then $\boldsymbol{q}_T$ is globally **conservative**

$$\int_\Omega \boldsymbol{q}_D \, dV = \int_\Omega \boldsymbol{q}_T \, dV.$$

Linear System

Weak form gives rise to a linear system in coefficients $\boldsymbol{d}$ and $\boldsymbol{t}$:

Linear System

Weak form gives rise to a linear system in coefficients $\boldsymbol{d}$ and $\boldsymbol{t}$:

$$M_T \boldsymbol{t} = M_{TD} \boldsymbol{d}.$$

Linear System

M_T is (symmetric) mass matrix for target mesh

$$(M_T)_{ij} = \int_\Omega \phi_T^{(i)} \phi_T^{(j)} \, dV.$$

Linear System

M_T is (symmetric) mass matrix for target mesh

$$(M_T)_{ij} = \int_\Omega \phi_T^{(i)} \phi_T^{(j)} \, dV.$$

Since $\operatorname{supp}(\phi) = \mathcal{T}$, M_T is block diagonal in DG, sparse but globally coupled in CG.

Linear System

M_T is (symmetric) mass matrix for target mesh

$$(M_T)_{ij} = \int_\Omega \phi_T^{(i)} \phi_T^{(j)} \, dV.$$

Since $\operatorname{supp}(\phi) = \mathcal{T}$, M_T is block diagonal in DG, sparse but globally coupled in CG.

Integrate via substitution for $F = \phi_T^{(i)} \phi_T^{(j)}$

$$\int_{b(\mathcal{U})} F(x, y) \, dx \, dy = \int_{\mathcal{U}} \det(Db) F\left(x(s, t), y(s, t)\right) \, ds \, dt$$

and then use quadrature rule on $\mathcal{U}$.

Linear System

Mixed mass matrix M_{TD}

$$(M_{TD})_{ij} = \int_{\Omega} \phi_T^{(i)} \phi_D^{(j)} \, dV.$$

Mixed mass matrix M_{TD}

$$(M_{TD})_{ij} = \int_\Omega \phi_T^{(i)} \phi_D^{(j)} \, dV.$$

Not symmetric, nor even square; rows correspond to shape functions on target mesh and columns to donor mesh.

Mixed mass matrix M_{TD}

$$(M_{TD})_{ij} = \int_\Omega \phi_T^{(i)} \phi_D^{(j)} \, dV.$$

Not symmetric, nor even square; rows correspond to shape functions on target mesh and columns to donor mesh.
Instead, compute entire RHS

$$(M_{TD}\boldsymbol{d})_j = \int_\Omega \phi_T^{(j)} \boldsymbol{q}_D \, dV.$$

Common Refinement

Given ϕ supported on $\mathcal{T}$

$$\int_{\Omega} \phi \, \boldsymbol{q}_D \, dV$$

Common Refinement

Given ϕ supported on $\mathcal{T}$

$$\int_{\Omega} \phi \, \boldsymbol{q}_D \, dV = \int_{\mathcal{T}} \phi \, \boldsymbol{q}_D \, dV$$

Common Refinement

Given ϕ supported on $\mathcal{T}$

$$\int_{\Omega} \phi \, \boldsymbol{q}_D \, dV = \int_{\mathcal{T}} \phi \, \boldsymbol{q}_D \, dV = \sum_{\mathcal{T}' \in \mathcal{M}_D} \int_{\mathcal{T} \cap \mathcal{T}'} \phi \, \boldsymbol{q}_D|_{\mathcal{T}'} \, dV$$

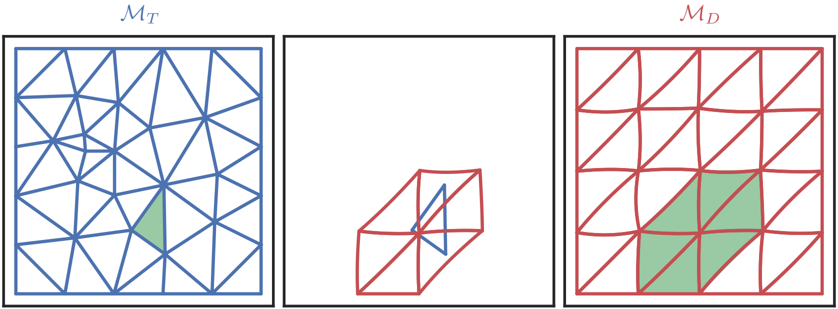
Common Refinement

Given ϕ supported on $\mathcal{T}$

$$\int_\Omega \phi\, \boldsymbol{q}_D \, dV = \int_{\mathcal{T}} \phi\, \boldsymbol{q}_D \, dV = \sum_{\mathcal{T}' \in \mathcal{M}_D} \int_{\mathcal{T} \cap \mathcal{T}'} \phi \, \boldsymbol{q}_D|_{\mathcal{T}'} \, dV$$

In CG, $\boldsymbol{q}_D$ need not be differentiable across elements and in DG $\boldsymbol{q}_D$ need not even be continuous

Common Refinement



Common Refinement

- Three subproblems:

Common Refinement

- Three subproblems:
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$

Common Refinement

- Three subproblems:
    - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$
    - Advancing Front: $(\mathcal{T}, \mathcal{T}') \in \mathcal{M}_T \times \mathcal{M}_D$

Common Refinement

- Three subproblems:
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$
  - Advancing Front: $(\mathcal{T}, \mathcal{T}') \in \mathcal{M}_T \times \mathcal{M}_D$
  - Integration over Curved Polygons: $\displaystyle\int_{\mathcal{T} \cap \mathcal{T}'} F \, dV$

Intersecting Curved Elements

Intersecting Curved Elements

- Find all points where edges intersect

Intersecting Curved Elements

- Find all points where edges intersect
- $\mathcal{T}_0 = b_0\left(\mathcal{U}\right)$, $\partial\mathcal{T}_0 = E_0 \cup E_1 \cup E_2$
- $\mathcal{T}_1 = b_1\left(\mathcal{U}\right)$, $\partial\mathcal{T}_1 = E_3 \cup E_4 \cup E_5$

Intersecting Curved Elements

- Find all points where edges intersect
- $\mathcal{T}_0 = b_0(\mathcal{U}), \partial\mathcal{T}_0 = E_0 \cup E_1 \cup E_2$
- $\mathcal{T}_1 = b_1(\mathcal{U}), \partial\mathcal{T}_1 = E_3 \cup E_4 \cup E_5$
- Determine which curve is interior at each intersection

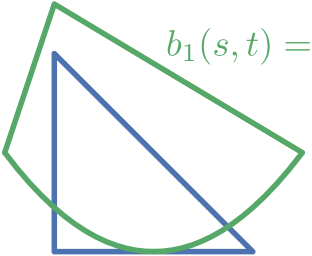
Intersecting Curved Elements

- Find all points where edges intersect
- $\mathcal{T}_0 = b_0\left(\mathcal{U}\right), \partial\mathcal{T}_0 = E_0 \cup E_1 \cup E_2$
- $\mathcal{T}_1 = b_1\left(\mathcal{U}\right), \partial\mathcal{T}_1 = E_3 \cup E_4 \cup E_5$
- Determine which curve is interior at each intersection
- Track parameter values at each intersection and combine into boundary of **curved polygon**

Intersecting Curved Elements

- Find all points where edges intersect
- $\mathcal{T}_0 = b_0(\mathcal{U})$, $\partial \mathcal{T}_0 = E_0 \cup E_1 \cup E_2$
- $\mathcal{T}_1 = b_1(\mathcal{U})$, $\partial \mathcal{T}_1 = E_3 \cup E_4 \cup E_5$
- Determine which curve is interior at each intersection
- Track parameter values at each intersection and combine into boundary of **curved polygon**
- $\mathcal{P} = \mathcal{T}_0 \cap \mathcal{T}_1$, $\partial \mathcal{P}$ defined by segments of edges from $\mathcal{T}_0$ and $\mathcal{T}_1$

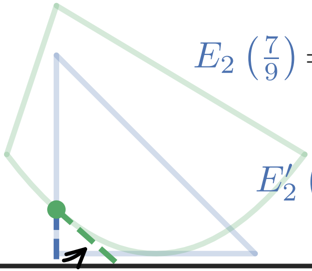
Intersecting Curved Elements



$$b_1(s,t) = \begin{bmatrix} 2(6s+t-1) \\ 2(8s^2+8st-8s+3t+2) \end{bmatrix}$$

$$b_0(s,t) = \begin{bmatrix} 8s \\ 8t \end{bmatrix}$$

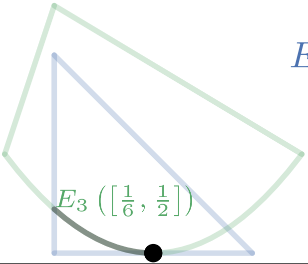
Intersecting Curved Elements



$$E_2\left(\frac{7}{9}\right) = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$E_2'\left(\frac{7}{9}\right) \times E_3'\left(\frac{1}{6}\right) = 96 > 0$$

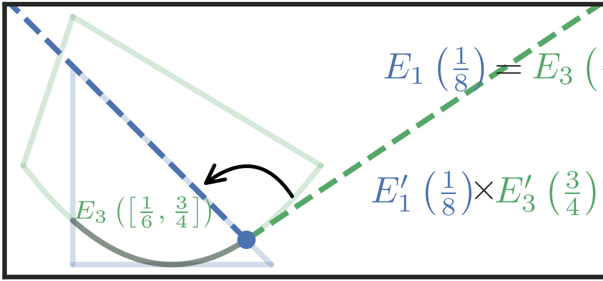
Intersecting Curved Elements



$$E_0\left(\tfrac{1}{2}\right) = E_3\left(\tfrac{1}{2}\right) = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$E_0'\left(\tfrac{1}{2}\right) \times E_3'\left(\tfrac{1}{2}\right) = 0$$

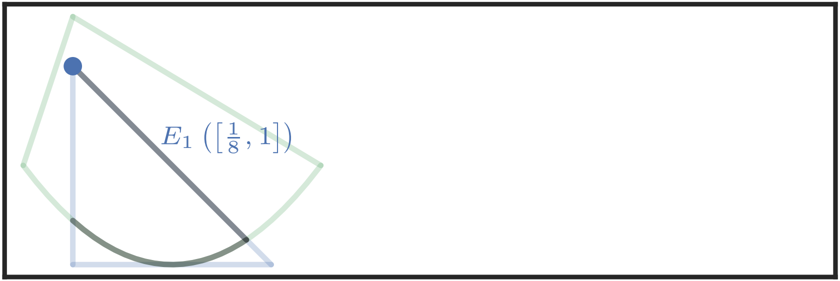
Intersecting Curved Elements



$$E_1\left(\frac{1}{8}\right) = E_3\left(\frac{3}{4}\right) = \begin{bmatrix} 7 \\ 1 \end{bmatrix}$$

$$E_1'\left(\frac{1}{8}\right) \times E_3'\left(\frac{3}{4}\right) = -160 < 0$$

Intersecting Curved Elements



Intersecting Curved Elements

Intersecting Curved Elements

- $\mathcal{P} = \mathcal{T}_0 \cap \mathcal{T}_1$
- $\partial\mathcal{P} = E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right) \cup E_1\left(\left[\frac{1}{8}, 1\right]\right) \cup E_2\left(\left[0, \frac{7}{9}\right]\right)$

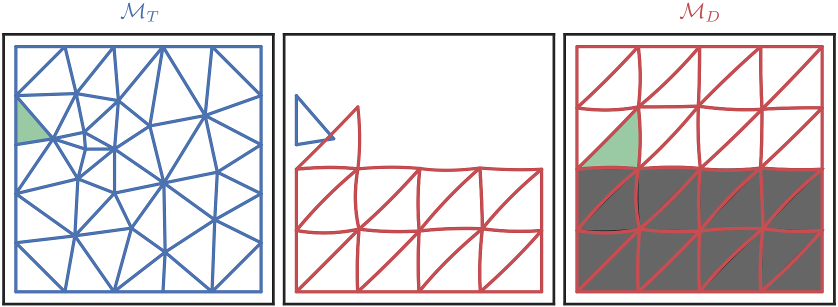
Advancing Front

- Find all pairs $\mathcal{T}$, $\mathcal{T}'$ of intersecting target and donor elements

Advancing Front

- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search

Advancing Front



Advancing Front

- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search

Advancing Front

- Find all pairs $\mathcal{T}$, $\mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search
  - $\mathcal{O}\left(|\mathcal{M}_D|\right)$

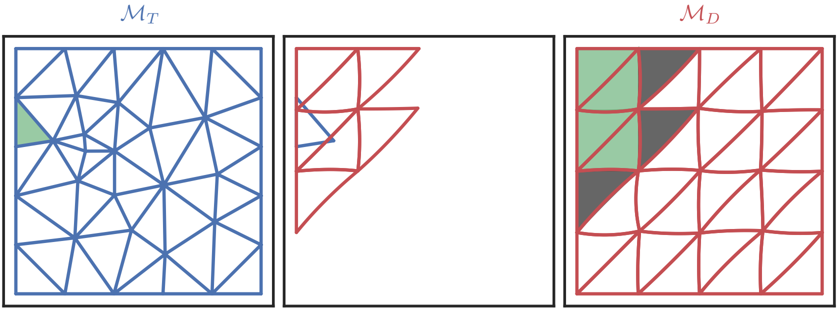
Advancing Front

- Find all pairs $\mathcal{T}$, $\mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search
  - $\mathcal{O}\left(|\mathcal{M}_D|\right)$
  - $\mathcal{O}\left(1\right)$ in typical ALE, though acts as CFL

Advancing Front

- Find all pairs $\mathcal{T}$, $\mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search
  - $\mathcal{O}(|\mathcal{M}_D|)$
  - $\mathcal{O}(1)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for neighbors $\mathcal{N}_\mathcal{T}$ that intersect $\mathcal{T}$ (as well as a one element buffer)

Advancing Front



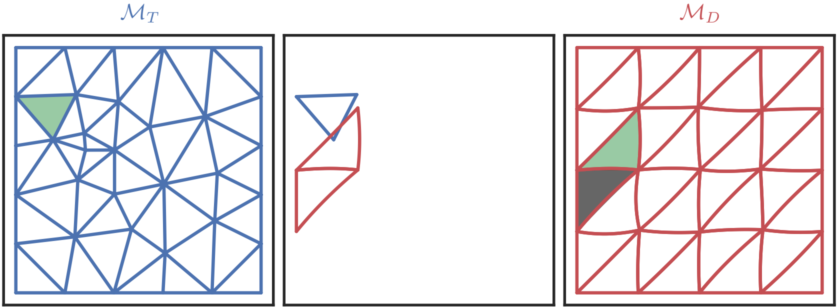
Advancing Front

- Find all pairs $\mathcal{T}$, $\mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search
  - $\mathcal{O}\left(|\mathcal{M}_D|\right)$
  - $\mathcal{O}\left(1\right)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for neighbors $\mathcal{N}_{\mathcal{T}}$ that intersect $\mathcal{T}$ (as well as a one element buffer)

Advancing Front

- Find all pairs $\mathcal{T}$, $\mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search
  - $\mathcal{O}\left(|\mathcal{M}_D|\right)$
  - $\mathcal{O}\left(1\right)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for neighbors $\mathcal{N}_{\mathcal{T}}$ that intersect $\mathcal{T}$ (as well as a one element buffer)
- Neighbors of $\mathcal{T}$ can find one matching donor element among $\mathcal{N}_{\mathcal{T}}$ in $\mathcal{O}\left(|\mathcal{N}_{\mathcal{T}}|\right) = \mathcal{O}\left(1\right)$

Advancing Front



Advancing Front

- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search
  - $\mathcal{O}\left(|\mathcal{M}_D|\right)$
  - $\mathcal{O}\left(1\right)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for neighbors $\mathcal{N}_{\mathcal{T}}$ that intersect $\mathcal{T}$ (as well as a one element buffer)
- Neighbors of $\mathcal{T}$ can find one matching donor element among $\mathcal{N}_{\mathcal{T}}$ in $\mathcal{O}\left(|\mathcal{N}_{\mathcal{T}}|\right) = \mathcal{O}\left(1\right)$

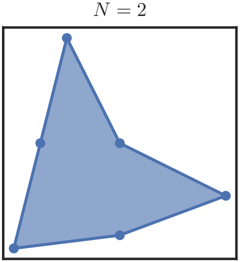
Advancing Front

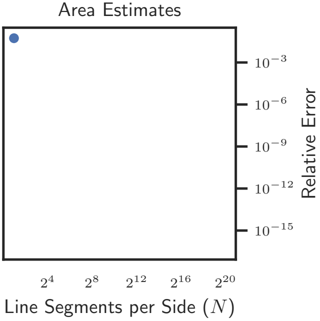
- Find all pairs $\mathcal{T}$, $\mathcal{T}'$ of intersecting target and donor elements
- Fix target element $\mathcal{T}$, perform brute-force search
  - $\mathcal{O}\left(|\mathcal{M}_D|\right)$
  - $\mathcal{O}\left(1\right)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for neighbors $\mathcal{N}_\mathcal{T}$ that intersect $\mathcal{T}$ (as well as a one element buffer)
- Neighbors of $\mathcal{T}$ can find one matching donor element among $\mathcal{N}_\mathcal{T}$ in $\mathcal{O}\left(|\mathcal{N}_\mathcal{T}|\right) = \mathcal{O}\left(1\right)$
- Total $\mathcal{O}\left(|\mathcal{M}_D| + |\mathcal{M}_T|\right)$

Integration over Curved Polygons

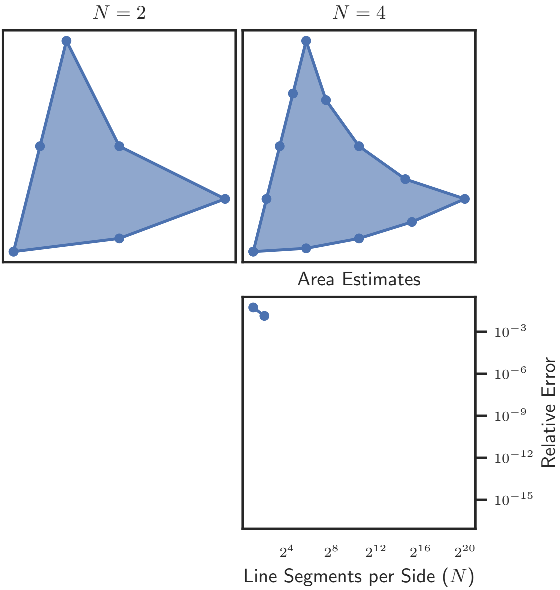
$$\int_{\mathcal{P}} F(x, y) \, dV, \qquad \mathcal{P} = \mathcal{T}_0 \cap \mathcal{T}_1, \qquad F = \phi_0 \, \phi_1$$

Integrate via Polygonal Approximation

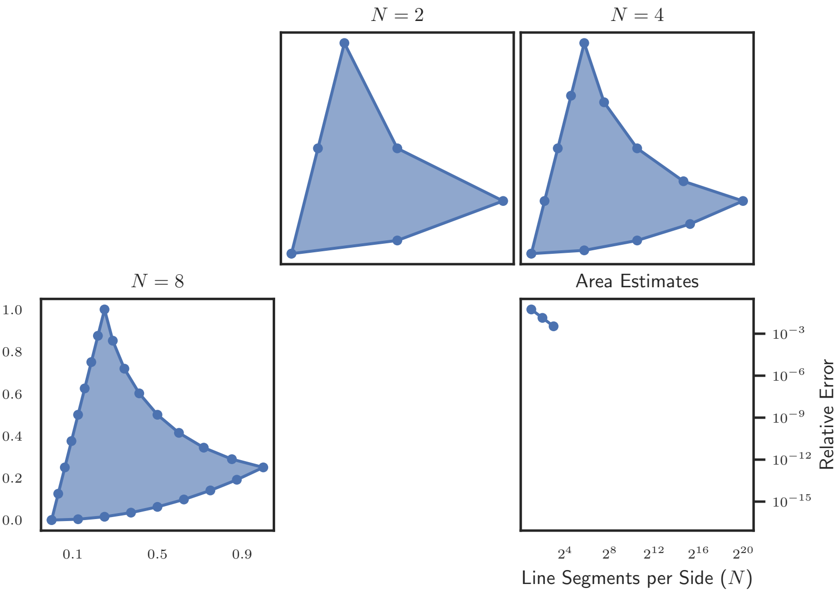




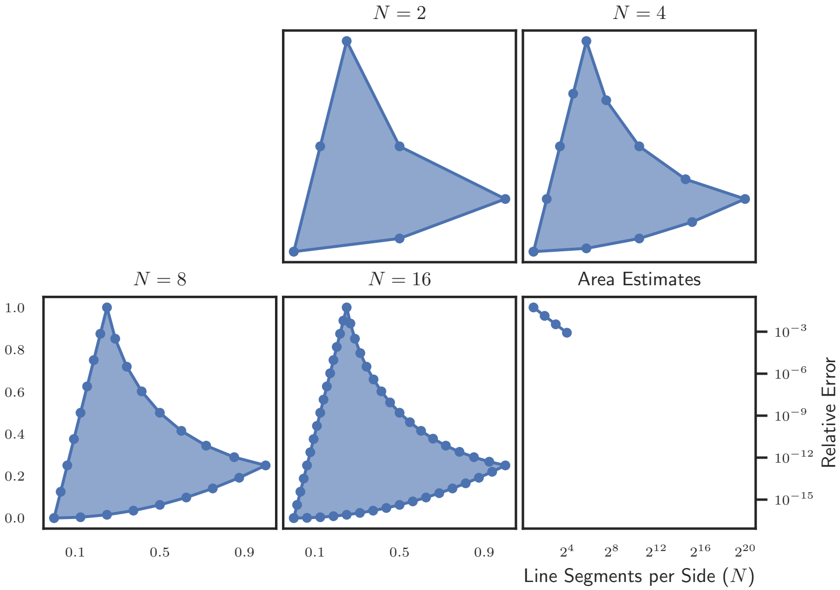
Integrate via Polygonal Approximation



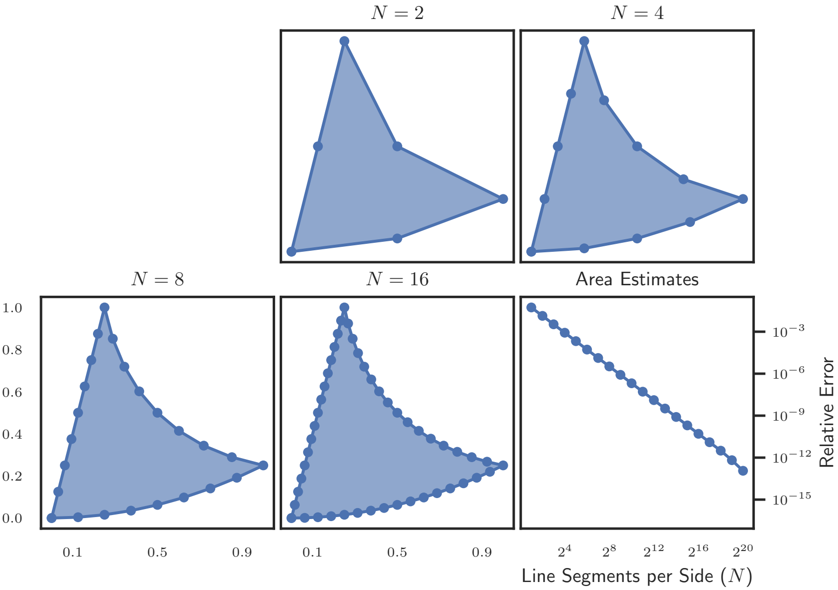
Integrate via Polygonal Approximation



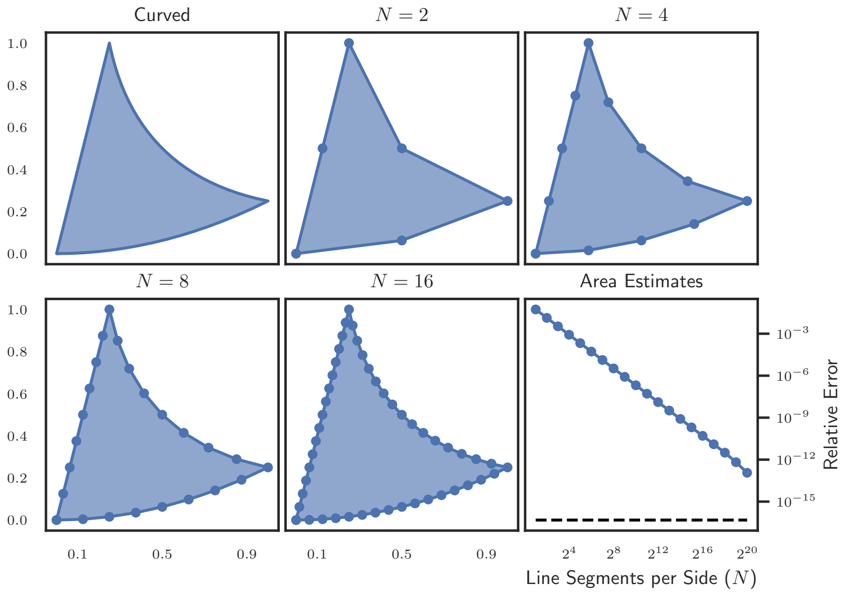
Integrate via Polygonal Approximation



Integrate via Polygonal Approximation



Integrate via Polygonal Approximation



Integrate via Polygonal Quadrature

- Polygonal quadrature rules not in wide use

Integrate via Polygonal Quadrature

- Polygonal quadrature rules not in wide use
- $\mathcal{P}$ has curved edges

Integrate via Polygonal Quadrature

- Polygonal quadrature rules not in wide use
- $\mathcal{P}$ has curved edges
- Transfinite interpolation or mean value coordinates

Integrate via Polygonal Quadrature

- Polygonal quadrature rules not in wide use
- $\mathcal{P}$ has curved edges
- Transfinite interpolation or mean value coordinates
    - Maps from (straight sided) reference domain, but increase degree or are not bijective

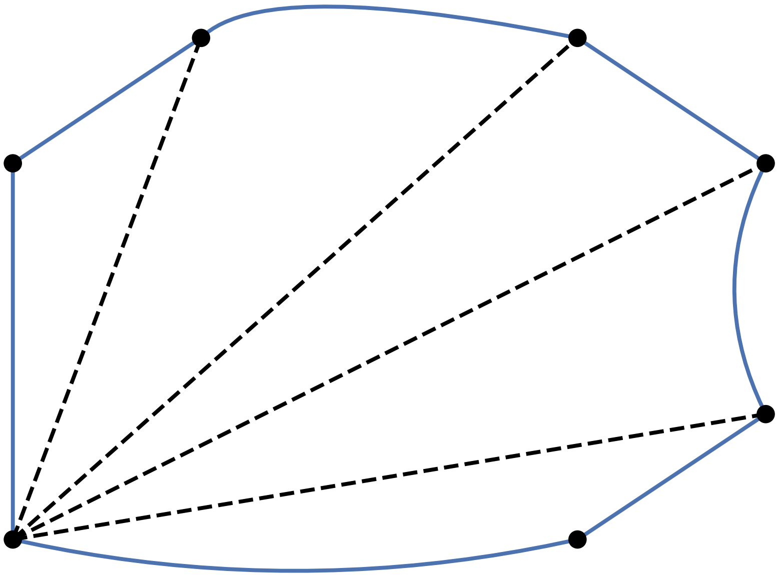
Integrate via Tessellation

- Triangular quadrature rules simple and well established

Integrate via Tessellation

- Triangular quadrature rules simple and well established
- Tessellate $\mathcal{P}$ into disjoint union of Bézier triangles

Integrate via Tessellation



Integrate via Tessellation

- Not clear if an arbitrary curved polygon **can** be tessellated without introducing interior nodes

Integrate via Tessellation

- Not clear if an arbitrary curved polygon **can** be tessellated without introducing interior nodes
- Need to check if diagonals cross edges; even worse if diagonals are also curved

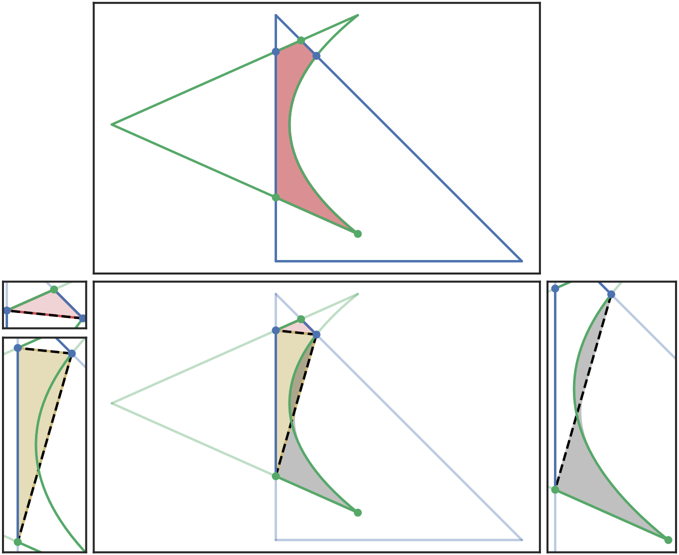
Integrate via Tessellation

- Not clear if an arbitrary curved polygon **can** be tessellated without introducing interior nodes
- Need to check if diagonals cross edges; even worse if diagonals are also curved
- If diagonals are valid, high degree Bézier triangles need interior control points introduced that don't cause triangle to invert

Integrate via Tessellation

- Tessellate with inverted Bézier triangles?

Integrate via Tessellation



Integrate via Tessellation

- Tessellate with inverted Bézier triangles?

Integrate via Tessellation

- Tessellate with inverted Bézier triangles?
- E.g. tessellation of $\mathcal{P}$ contains inverted $\mathcal{T}_2 = b\left(\mathcal{U}\right)$, then numerically integrate

$$\int_{\mathcal{T}_2} \phi_0 \phi_1 \, dx \, dy = \int_{\mathcal{U}} \left|\det(Db)\right| \left(\phi_0 \circ b\right) \left(\phi_1 \circ b\right) \, ds \, dt$$

Integrate via Tessellation

- Tessellate with inverted Bézier triangles?
- E.g. tessellation of $\mathcal{P}$ contains inverted $\mathcal{T}_2 = b\left(\mathcal{U}\right)$, then numerically integrate

$$\int_{\mathcal{T}_2} \phi_0 \phi_1 \, dx \, dy = \int_{\mathcal{U}} \left|\det(Db)\right| \left(\phi_0 \circ b\right) \left(\phi_1 \circ b\right) \, ds \, dt$$

- $\left|\det(Db)\right|$ non-smooth, bad for quadrature

Integrate via Tessellation

- Tessellate with inverted Bézier triangles?
- E.g. tessellation of $\mathcal{P}$ contains inverted $\mathcal{T}_2 = b\left(\mathcal{U}\right)$, then numerically integrate

$$\int_{\mathcal{T}_2} \phi_0 \phi_1 \, dx \, dy = \int_{\mathcal{U}} \left|\det(Db)\right| \left(\phi_0 \circ b\right) \left(\phi_1 \circ b\right) \, ds \, dt$$

- $\left|\det(Db)\right|$ non-smooth, bad for quadrature
- If ϕ_0 from pre-image basis, then $\phi_0 \circ b = \widehat{\phi}_0 \circ b_0^{-1} \circ b$

Integrate via Tessellation

- Tessellate with inverted Bézier triangles?
- E.g. tessellation of $\mathcal{P}$ contains inverted $\mathcal{T}_2 = b\left(\mathcal{U}\right)$, then numerically integrate

$$\int_{\mathcal{T}_2} \phi_0 \phi_1 \, dx \, dy = \int_{\mathcal{U}} \left|\det(Db)\right| \left(\phi_0 \circ b\right) \left(\phi_1 \circ b\right) \, ds \, dt$$

- $\left|\det(Db)\right|$ non-smooth, bad for quadrature
- If ϕ_0 from pre-image basis, then $\phi_0 \circ b = \widehat{\phi}_0 \circ b_0^{-1} \circ b$
- $\mathcal{T}_2$ inverted, $b(s, t) \notin \mathcal{P}$ is possible but b_0^{-1} need not need be defined outside of $\mathcal{T}_0$

Integrate via Green's Theorem

- Horizontal antiderivative H and vertical antiderivative V s.t. $H_x = V_y = F$

Integrate via Green's Theorem

- Horizontal antiderivative H and vertical antiderivative V s.t. $H_x = V_y = F$
- $H(0, y) \equiv V(x, 0) \equiv 0$ for uniqueness (needed to evaluate)

Integrate via Green's Theorem

- Horizontal antiderivative H and vertical antiderivative V s.t. $H_x = V_y = F$
- $H(0, y) \equiv V(x, 0) \equiv 0$ for uniqueness (needed to evaluate)
- If $\partial\mathcal{P} = C_1 \cup \cdots \cup C_n$, since $2F = H_x + V_y$ Green's gives

$$\int_{\mathcal{P}} 2F\,dV = \oint_{\partial\mathcal{P}} H\,dy - V\,dx = \sum_j \int_{C_j} H\,dy - V\,dx$$

Integrate via Green's Theorem

- Horizontal antiderivative H and vertical antiderivative V s.t. $H_x = V_y = F$
- $H(0, y) \equiv V(x, 0) \equiv 0$ for uniqueness (needed to evaluate)
- If $\partial\mathcal{P} = C_1 \cup \cdots \cup C_n$, since $2F = H_x + V_y$ Green's gives

$$\int_{\mathcal{P}} 2F\, dV = \oint_{\partial\mathcal{P}} H\, dy - V\, dx = \sum_j \int_{C_j} H\, dy - V\, dx$$

- Each C given by $x(r)$, $y(r)$, 1D quadrature on unit interval of

$$G(r) = H(x(r), y(r))\, y'(r) - V(x(r), y(r))\, x'(r)$$

Integrate via Green's Theorem

- How to evaluate $H(x(r), y(r))$?

Integrate via Green's Theorem

- How to evaluate $H(x(r), y(r))$?
- Fundamental theorem of calculus

$$H(\alpha, \beta) = H(\alpha, \beta) - H(0, \beta) = \int_0^\alpha F(x, \beta)\ dx$$

Integrate via Green's Theorem

- How to evaluate $H(x(r), y(r))$?
- Fundamental theorem of calculus

$$H(\alpha, \beta) = H(\alpha, \beta) - H(0, \beta) = \int_0^\alpha F(x, \beta)\ dx$$

- Quadrature points $F(x_j, \beta)$ may fall outside $\mathcal{P}$, so pre-image basis cannot be used

Integrate via Green's Theorem

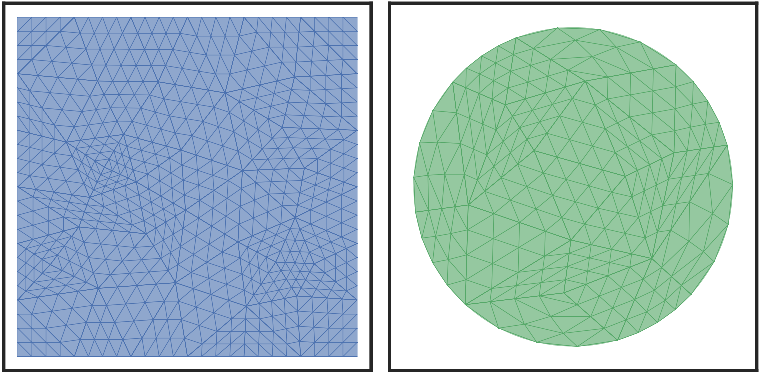
- How to evaluate $H(x(r), y(r))$?
- Fundamental theorem of calculus

$$H(\alpha, \beta) = H(\alpha, \beta) - H(0, \beta) = \int_0^\alpha F(x, \beta)\ dx$$

- Quadrature points $F(x_j, \beta)$ may fall outside $\mathcal{P}$, so pre-image basis cannot be used
- Global coordinates basis means F is a polynomial on $\mathbf{R}^2$, so quadrature is exact

Numerical Experiments

- Three meshes ($p = 1, 2, 3$)



Numerical Experiments

- Three meshes ($p = 1, 2, 3$)

Numerical Experiments

- Three meshes ($p = 1, 2, 3$)
- Three functions

Numerical Experiments

- Three meshes ($p = 1, 2, 3$)
- Three functions
  - $\zeta_1(x, y) = 5y^3 + x^2 + 2y + 3$

Numerical Experiments

- Three meshes $(p = 1, 2, 3)$
- Three functions
  - $\zeta_1(x, y) = 5y^3 + x^2 + 2y + 3$
  - $\zeta_2(x, y) = \exp\left(x^2\right) + 2y$

Numerical Experiments

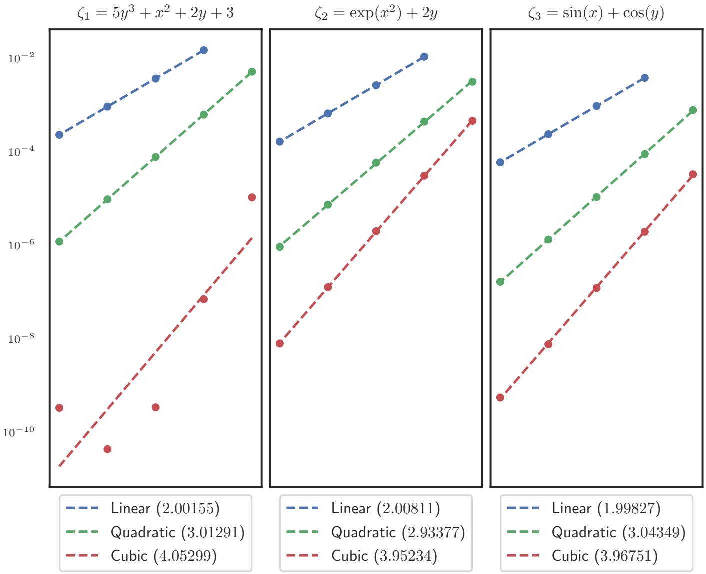
- Three meshes ($p = 1, 2, 3$)
- Three functions
  - $\zeta_1(x, y) = 5y^3 + x^2 + 2y + 3$
  - $\zeta_2(x, y) = \exp\left(x^2\right) + 2y$
  - $\zeta_3(x, y) = \sin(x) + \cos(y)$

- Three meshes ($p = 1, 2, 3$)
- Three functions
  - $\zeta_1(x, y) = 5y^3 + x^2 + 2y + 3$
  - $\zeta_2(x, y) = \exp\left(x^2\right) + 2y$
  - $\zeta_3(x, y) = \sin(x) + \cos(y)$
- Nodal interpolant $\boldsymbol{q}_D = \sum_j \zeta\left(\boldsymbol{n}_j\right) \phi_D^{(j)}$

Numerical Experiments

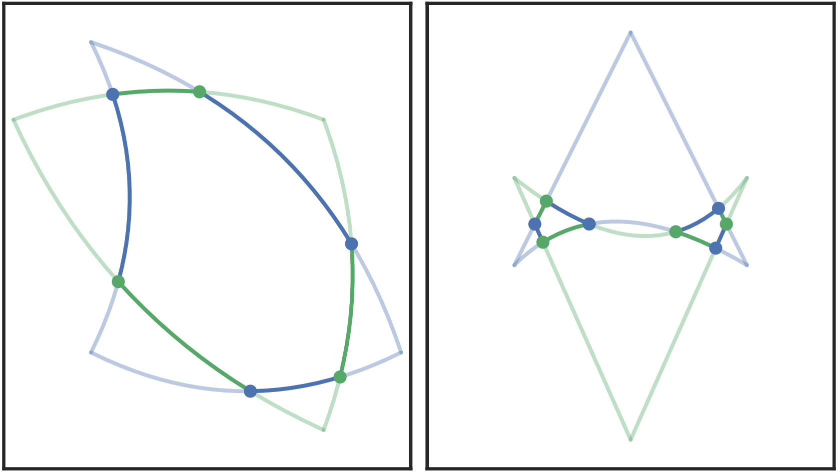
- Three meshes ($p = 1, 2, 3$)
- Three functions
  - $\zeta_1(x, y) = 5y^3 + x^2 + 2y + 3$
  - $\zeta_2(x, y) = \exp\left(x^2\right) + 2y$
  - $\zeta_3(x, y) = \sin(x) + \cos(y)$
- Nodal interpolant $\boldsymbol{q}_D = \sum_j \zeta\left(\boldsymbol{n}_j\right) \phi_D^{(j)}$
- Expect $\mathcal{O}\left(h^{p+1}\right)$ errors

Numerical Experiments

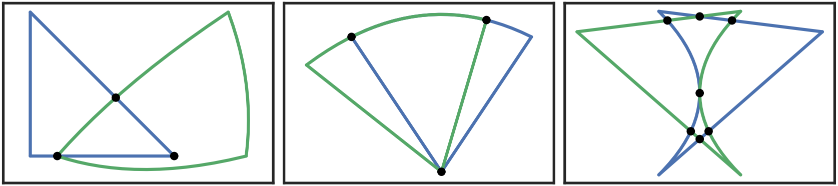


Ill-conditioned Bézier Curve Intersection

Ill-Conditioned Intersections



Ill-Conditioned Intersections



Ill-Conditioned Intersections

- Edges are Bézier curves, e.g. $b(r, 0) = \sum_{j=0}^{n} \binom{n}{j} (1 - r)^{n-j} r^j \, \boldsymbol{p}_{n-j,j,0}$

Ill-Conditioned Intersections

- Edges are Bézier curves, e.g. $b(r, 0) = \sum_{j=0}^{n} \binom{n}{j} (1 - r)^{n-j} r^j \, \boldsymbol{p}_{n-j,j,0}$
- Tangent intersection equivalent to double root of polynomial, i.e. condition number is infinite

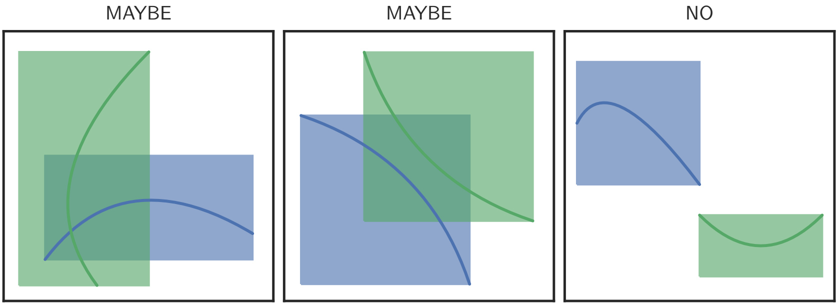
Ill-Conditioned Intersections

- Edges are Bézier curves, e.g. $b(r, 0) = \sum_{j=0}^{n} \binom{n}{j} (1 - r)^{n-j} r^j \, \boldsymbol{p}_{n-j,j,0}$
- Tangent intersection equivalent to double root of polynomial, i.e. condition number is infinite
- Random pair of meshes, "almost tangent" intersections increasingly frequent as $h \longrightarrow 0^+$

Intersection Algorithm

- Bounding box check

Intersection Algorithm



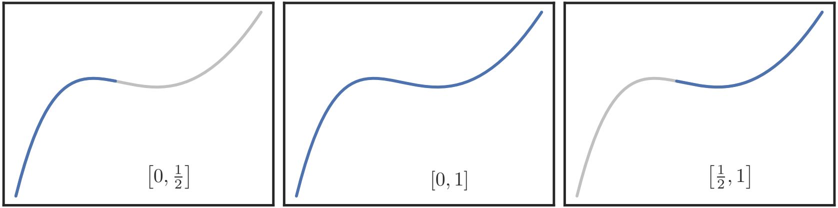
Intersection Algorithm

- Bounding box check

Intersection Algorithm

- Bounding box check
- Curve subdivision

Intersection Algorithm



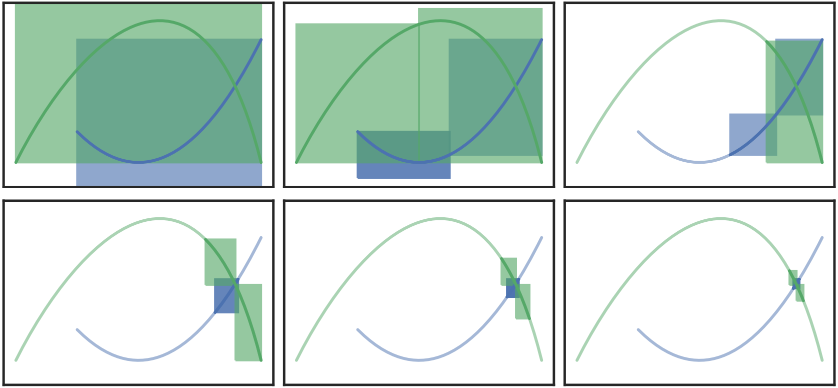
Intersection Algorithm

- Bounding box check
- Curve subdivision

Intersection Algorithm

- Bounding box check
- Curve subdivision
- Check subdivided pairs

Intersection Algorithm



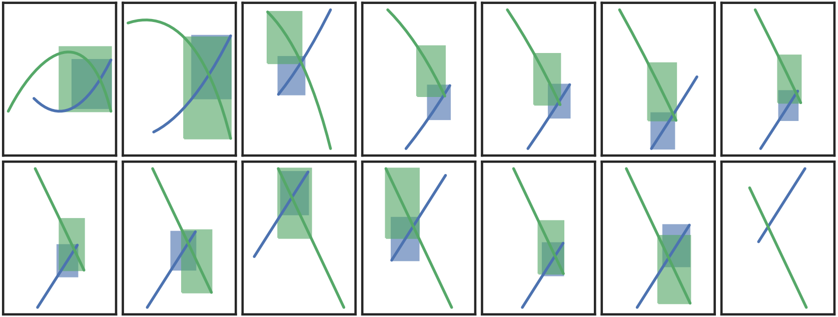
Intersection Algorithm

- Bounding box check
- Curve subdivision
- Check subdivided pairs

Intersection Algorithm

- Bounding box check
- Curve subdivision
- Check subdivided pairs
- Subdivide until “almost” linear

Intersection Algorithm



Intersection Algorithm

- Bounding box check
- Curve subdivision
- Check subdivided pairs
- Subdivide until “almost” linear

Intersection Algorithm

- Bounding box check
- Curve subdivision
- Check subdivided pairs
- Subdivide until "almost" linear
- Use intersection of lines as seed for Newton's method

Newton's Method

- Bézier curves $b_0(s), b_1(t)$ map into $\mathbf{R}^2$

Newton's Method

- Bézier curves $b_0(s), b_1(t)$ map into $\mathbf{R}^2$
- Residual $F(s,t) = b_0(s) - b_1(t)$, Jacobian $J = \left[\begin{array}{cc} b_0'(s) & -b_1'(t) \end{array}\right]$

Newton's Method

- Bézier curves $b_0(s), b_1(t)$ map into $\mathbf{R}^2$
- Residual $F(s,t) = b_0(s) - b_1(t)$, Jacobian $J = \begin{bmatrix} b_0'(s) & -b_1'(t) \end{bmatrix}$
- Newton's:

$$\begin{bmatrix} s_{n+1} \\ t_{n+1} \end{bmatrix} = \begin{bmatrix} s_n \\ t_n \end{bmatrix} - J_n^{-1} F_n$$

Newton's Method

- Bézier curves $b_0(s), b_1(t)$ map into $\mathbf{R}^2$
- Residual $F(s, t) = b_0(s) - b_1(t)$, Jacobian $J = \left[\begin{array}{cc} b_0'(s) & -b_1'(t) \end{array}\right]$
- Newton's:

$$\left[\begin{array}{c} s_{n+1} \\ t_{n+1} \end{array}\right] = \left[\begin{array}{c} s_n \\ t_n \end{array}\right] - J_n^{-1} F_n$$

- At tangent intersections, J is exactly singular

Newton's Method

- Bézier curves $b_0(s), b_1(t)$ map into $\mathbf{R}^2$
- Residual $F(s,t) = b_0(s) - b_1(t)$, Jacobian $J = \left[\begin{array}{cc} b_0'(s) & -b_1'(t) \end{array}\right]$
- Newton's:

$$\left[\begin{array}{c} s_{n+1} \\ t_{n+1} \end{array}\right] = \left[\begin{array}{c} s_n \\ t_n \end{array}\right] - J_n^{-1} F_n$$

- At tangent intersections, J is exactly singular
- At ill-conditioned intersections, J is almost singular and evaluation of F is typically ill-conditioned as well

Modified Newton's Method

- Sources of error:

Modified Newton's Method

- Sources of error:
  - Evaluation of F

Modified Newton's Method

- Sources of error:
  - Evaluation of F
  - Evaluation of J

Modified Newton's Method

- Sources of error:
    - Evaluation of F
    - Evaluation of J
    - Solution of $J\boldsymbol{y} = F$

Modified Newton's Method

- Sources of error:
  - Evaluation of F
  - Evaluation of J
  - Solution of $J\boldsymbol{y} = F$
- Can improve accuracy of Newton's method by using extended precision for evaluation of F, even in presence of inaccurate J and unstable solve

Modified Newton's Method

- Sources of error:
  - Evaluation of F
  - Evaluation of J
  - Solution of $J\boldsymbol{y} = F$
- Can improve accuracy of Newton's method by using extended precision for evaluation of F, even in presence of inaccurate J and unstable solve
  - Tisseur (2001) showed this generically and applied to iterative refinement for generalized eigenvalue problem

Compensated Evaluation

Compensated Algorithms

- Error-free transform (EFT):

Compensated Algorithms

- Error-free transform (EFT):
  - Problem $f(x)$, algorithm computes $\widehat{f}$

Compensated Algorithms

- Error-free transform (EFT):
  - Problem $f(x)$, algorithm computes $\widehat{f}$
  - EFT modifies algorithm to also produce error $\widehat{e}$ so that $f(x) = \widehat{f} + \widehat{e}$

Compensated Algorithms

- Error-free transform (EFT):
  - Problem $f(x)$, algorithm computes $\widehat{f}$
  - EFT modifies algorithm to also produce error $\widehat{e}$ so that $f(x) = \widehat{f} + \widehat{e}$
  - Error is **exact**, hence "error-free"

Compensated Algorithms

- Error-free transform (EFT):
  - Problem $f(x)$, algorithm computes $\widehat{f}$
  - EFT modifies algorithm to also produce error $\widehat{e}$ so that $f(x) = \widehat{f} + \widehat{e}$
  - Error is **exact**, hence “error-free”
- **Compensated** algorithm uses $\widehat{f} \oplus \widehat{e}$; better approximation of $f(x)$ than $\widehat{f}$ (usually)

EFT for Addition

- $\texttt{TwoSum}(a, b) = [S, \sigma]$

EFT for Addition

- $\texttt{TwoSum}(a, b) = [S, \sigma]$
- $a + b = S + \sigma$

EFT for Addition

- `TwoSum`$(a, b) = [S, \sigma]$
- $a + b = S + \sigma$
- Algorithm:

EFT for Addition

- `TwoSum`$(a, b) = [S, \sigma]$
- $a + b = S + \sigma$
- Algorithm:
  - $S = a \oplus b$

EFT for Addition

- `TwoSum`$(a, b) = [S, \sigma]$
- $a + b = S + \sigma$
- Algorithm:
  - $S = a \oplus b$
  - `almost_b` $= S \ominus a$

EFT for Addition

- $\texttt{TwoSum}(a, b) = [S, \sigma]$
- $a + b = S + \sigma$
- Algorithm:
  - $S = a \oplus b$
  - $\texttt{almost\_b} = S \ominus a$
  - $\sigma = (a \ominus (S \ominus \texttt{almost\_b})) \oplus (b \ominus \texttt{almost\_b})$

EFT for Multiplication

- $\texttt{TwoProb}(a, b) = [P, \pi]$

EFT for Multiplication

- $\texttt{TwoProb}(a, b) = [P, \pi]$
- $a \times b = P + \pi$

EFT for Multiplication

- $\texttt{TwoProb}(a, b) = [P, \pi]$
- $a \times b = P + \pi$
- Algorithm:

EFT for Multiplication

- `TwoProb`$(a, b) = [P, \pi]$
- $a \times b = P + \pi$
- Algorithm:
    - $P = a \otimes b$

EFT for Multiplication

- $\texttt{TwoProb}(a, b) = [P, \pi]$
- $a \times b = P + \pi$
- Algorithm:
    - $P = a \otimes b$
    - $\pi = \texttt{FusedMultiplyAdd}(a, b, -P)$

EFT for Multiplication

- $\texttt{TwoProb}(a, b) = [P, \pi]$
- $a \times b = P + \pi$
- Algorithm:
  - $P = a \otimes b$
  - $\pi = \texttt{FusedMultiplyAdd}(a, b, -P)$
- **Not possible** to apply this to $\oslash$

Bézier Curve Evaluation

- $p(s) = \binom{n}{0}(1-s)^n \cdot b_0 + \binom{n}{1}(1-s)^{n-1}s \cdot b_1 + \cdots$

Bézier Curve Evaluation

- $p(s) = \binom{n}{0}(1-s)^n \cdot b_0 + \binom{n}{1}(1-s)^{n-1}s \cdot b_1 + \cdots$
- de Casteljau Algorithm:

Bézier Curve Evaluation

- $p(s) = \binom{n}{0}(1-s)^n \cdot b_0 + \binom{n}{1}(1-s)^{n-1}s \cdot b_1 + \cdots$
- de Casteljau Algorithm:
  - $\widehat{\boldsymbol{b}}^{(n)} = \boldsymbol{b}$, $\widehat{r} = 1 \ominus s$

Bézier Curve Evaluation

- $p(s) = \binom{n}{0}(1-s)^n \cdot b_0 + \binom{n}{1}(1-s)^{n-1} s \cdot b_1 + \cdots$
- de Casteljau Algorithm:
  - $\widehat{\boldsymbol{b}}^{(n)} = \boldsymbol{b}$, $\widehat{r} = 1 \ominus s$
  - $\widehat{\boldsymbol{b}}^{(k)} = \left(\widehat{r} \otimes \widehat{\boldsymbol{b}}^{(k+1)}\right) \oplus \left(s \otimes \widehat{\boldsymbol{b}}^{(k+1)}\right)$

Bézier Curve Evaluation

- $p(s) = \binom{n}{0}(1-s)^n \cdot b_0 + \binom{n}{1}(1-s)^{n-1} s \cdot b_1 + \cdots$
- de Casteljau Algorithm:
  - $\widehat{\boldsymbol{b}}^{(n)} = \boldsymbol{b}$, $\widehat{r} = 1 \ominus s$
  - $\widehat{\boldsymbol{b}}^{(k)} = \left(\widehat{r} \otimes \widehat{\boldsymbol{b}}^{(k+1)}\right) \oplus \left(s \otimes \widehat{\boldsymbol{b}}^{(k+1)}\right)$
  - $\widehat{p} = \widehat{\boldsymbol{b}}_0^{(0)}$

Bézier Curve Evaluation

- $p(s) = \binom{n}{0}(1-s)^n \cdot b_0 + \binom{n}{1}(1-s)^{n-1} s \cdot b_1 + \cdots$
- de Casteljau Algorithm:
    - $\widehat{\boldsymbol{b}}^{(n)} = \boldsymbol{b}$, $\widehat{r} = 1 \ominus s$
    - $\widehat{\boldsymbol{b}}^{(k)} = \left(\widehat{r} \otimes \widehat{\boldsymbol{b}}^{(k+1)}\right) \oplus \left(s \otimes \widehat{\boldsymbol{b}}^{(k+1)}\right)$
    - $\widehat{p} = \widehat{\boldsymbol{b}}_0^{(0)}$
- Only uses $\oplus$, $\ominus$ and $\otimes$

Compensated de Castlejau

- Only uses $\oplus$, $\ominus$ and $\otimes$

Compensated de Castlejau

- Only uses $\oplus$, $\ominus$ and $\otimes$
- Track errors and combine them progressively

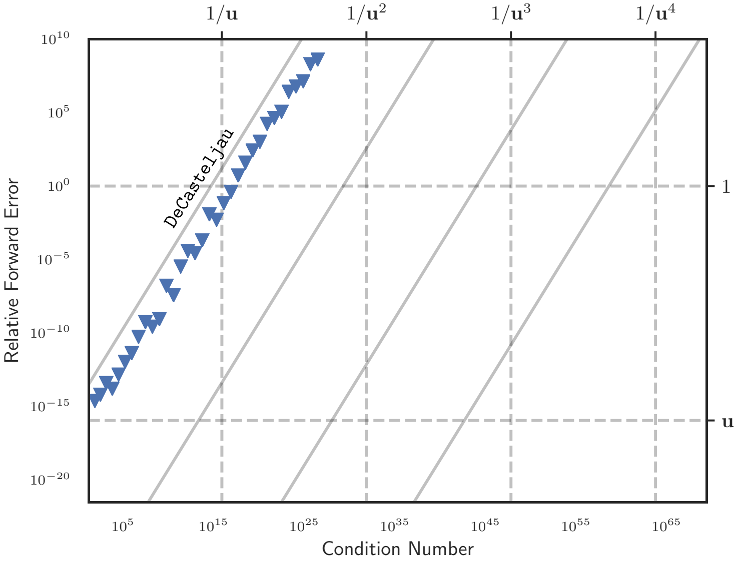
Compensated de Castlejau

- Only uses $\oplus$, $\ominus$ and $\otimes$
- Track errors and combine them progressively
- Test polynomial $p(s) = (s-1)\left(s - \frac{3}{4}\right)^7$; evaluation at $s = \frac{3}{4}$ has infinite condition, very ill-conditioned nearby

Compensated de Castlejau

$$\frac{|p(s) - \texttt{DeCasteljau}\,(p, s)|}{|p(s)|} \leq \operatorname{cond}\,(p, s) \cdot \mathcal{O}\,(\mathbf{u})$$

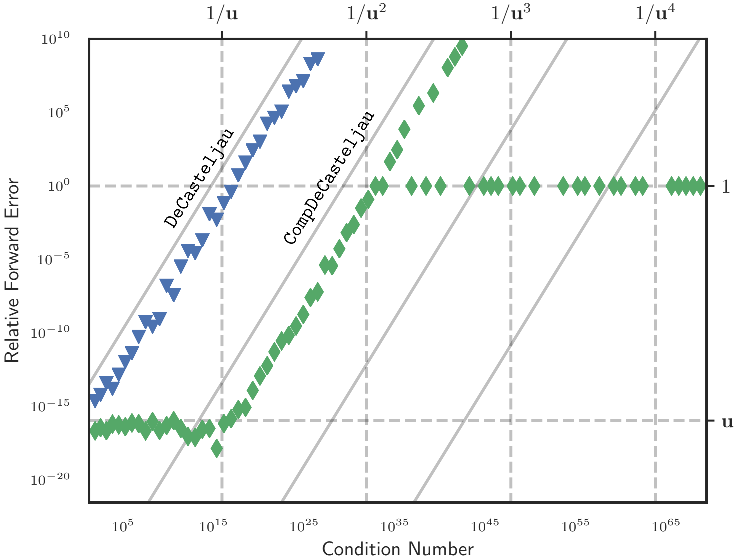
Compensated de Castlejau



Compensated de Castlejau

$$\frac{|p(s) - \texttt{CompDeCasteljau}\,(p, s)|}{|p(s)|} \leq \mathcal{O}\,(\mathbf{u}) + \mathrm{cond}\,(p, s) \cdot \mathcal{O}\left(\mathbf{u}^2\right)$$

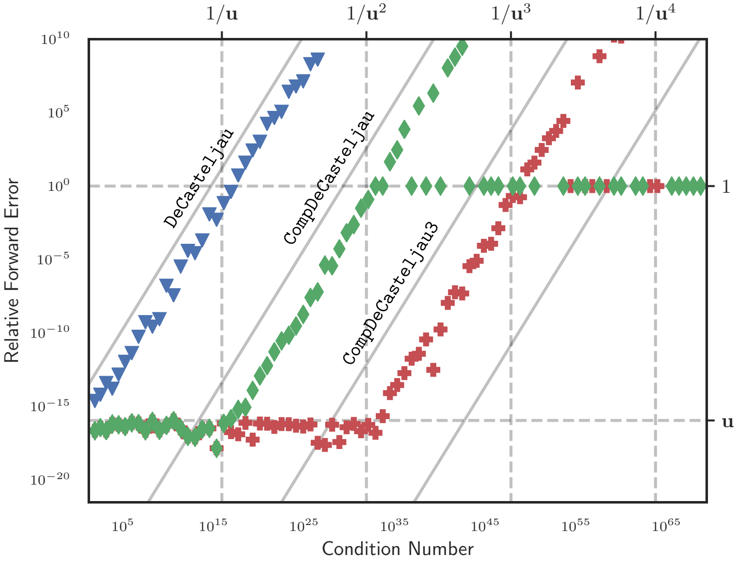
Compensated de Castlejau



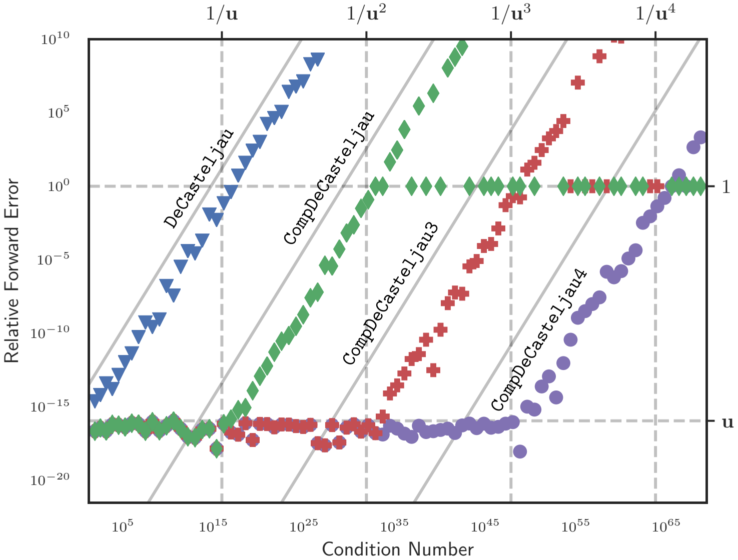
Compensated de Castlejau

$$\frac{|p(s) - \texttt{CompDeCasteljauK}\,(p, s)|}{|p(s)|} \leq \mathcal{O}\left(\mathbf{u}\right) + \mathrm{cond}\left(p, s\right) \cdot \mathcal{O}\left(\mathbf{u}^K\right)$$

Compensated de Castlejau



Compensated de Castlejau



Modified Newton's for Intersection

Polynomial Roots

- $p(s)$ written in Bernstein-Bézier form

Polynomial Roots

- $p(s)$ written in Bernstein-Bézier form
- Newton: $s_{n+1} = s_n - \frac{p(s_n)}{p'(s_n)}$

Polynomial Roots

- $p(s)$ written in Bernstein-Bézier form
- Newton: $s_{n+1} = s_n - \frac{p(s_n)}{p'(s_n)}$
- Three algorithms

Polynomial Roots

- $p(s)$ written in Bernstein-Bézier form
- Newton: $s_{n+1} = s_n - \frac{p(s_n)}{p'(s_n)}$
- Three algorithms
    - `DNewtonBasic`: `DeCasteljau` for residual $p(s)$ and Jacobian $p'(s)$

Polynomial Roots

- $p(s)$ written in Bernstein-Bézier form
- Newton: $s_{n+1} = s_n - \frac{p(s_n)}{p'(s_n)}$
- Three algorithms
  - `DNewtonBasic`: `DeCasteljau` for residual $p(s)$ and Jacobian $p'(s)$
  - `DNewtonAccurate`: `CompDeCasteljau` for residual, `DeCasteljau` for Jacobian

Polynomial Roots

- $p(s)$ written in Bernstein-Bézier form
- Newton: $s_{n+1} = s_n - \frac{p(s_n)}{p'(s_n)}$
- Three algorithms
  - `DNewtonBasic`: `DeCasteljau` for residual $p(s)$ and Jacobian $p'(s)$
  - `DNewtonAccurate`: `CompDeCasteljau` for residual, `DeCasteljau` for Jacobian
  - `DNewtonFull`: `CompDeCasteljau` for residual and Jacobian

Polynomial Roots

- Test problem; need coefficients that can be exactly represented

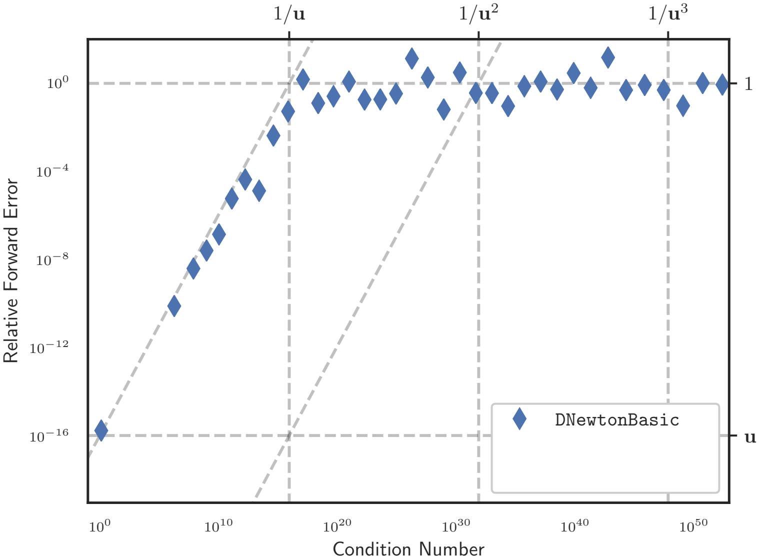
Polynomial Roots

- Test problem; need coefficients that can be exactly represented
- $p(s) = (1 - 5s)^n + 2^{30}(1 - 3s)^n$, n odd

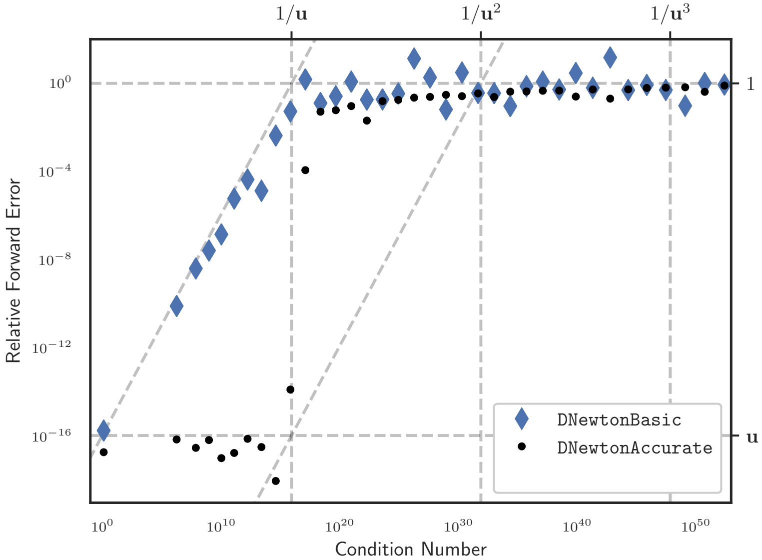
Polynomial Roots

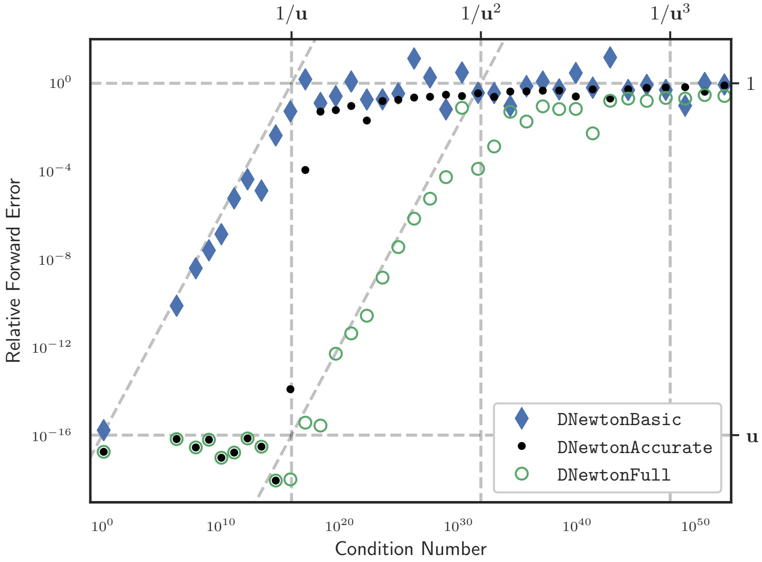
- Test problem; need coefficients that can be exactly represented
- $p(s) = (1-5s)^n + 2^{30}(1-3s)^n$, n odd
- Root $\alpha = \dfrac{1+2^{30/n}}{5+3\cdot 2^{30/n}} \in \left[\dfrac{1}{4}, \dfrac{1}{3}\right]$

Polynomial Roots



Polynomial Roots




1/u
1/u²
1/u³
Relative Forward Error
10^0
10^-4
10^-8
10^-12
10^-16
1
u
DNewtonBasic
DNewtonAccurate
DNewtonFull
10^0
10^10
10^20
10^30
10^40
10^50
Condition Number

Bézier Curve Intersection

- $F(s, t) = b_0(s) - b_1(t)$

Bézier Curve Intersection

- $F(s, t) = b_0(s) - b_1(t)$
- Evaluate $x_0(s), y_0(s), x_1(t), y_1(t)$ via de Casteljau

Bézier Curve Intersection

- $F(s, t) = b_0(s) - b_1(t)$
- Evaluate $x_0(s), y_0(s), x_1(t), y_1(t)$ via de Casteljau
- Modify Newton's method by using "extended precision" (i.e. compensated method) for $\widehat{F}$

Compensated Residual

- $\widehat{F} = \begin{bmatrix} \widehat{x} \\ \widehat{y} \end{bmatrix}$

Compensated Residual

- $\widehat{F} = \begin{bmatrix} \widehat{x} \\ \widehat{y} \end{bmatrix}$
- $\widehat{x} = \texttt{CompDeCasteljau}\,(x_0, s) \ominus \texttt{CompDeCasteljau}\,(x_1, t)$

Compensated Residual

- $\widehat{F} = \left[\begin{array}{c} \widehat{x} \\ \widehat{y} \end{array} \right]$
- $\widehat{x} = \texttt{CompDeCasteljau}\left(x_0, s\right) \ominus \texttt{CompDeCasteljau}\left(x_1, t\right)$
- Fails; $x_0(s), x_1(t)$ may not be near zero, e.g. $\widehat{x}_0 \oplus \widehat{e}_0 = \widehat{x}_0$

Compensated Residual

- $\widehat{F} = \left[\begin{array}{c} \widehat{x} \\ \widehat{y} \end{array} \right]$
- $\widehat{x} = \texttt{CompDeCasteljau}\,(x_0, s) \ominus \texttt{CompDeCasteljau}\,(x_1, t)$
- Fails; $x_0(s), x_1(t)$ may not be near zero, e.g. $\widehat{x}_0 \oplus \widehat{e}_0 = \widehat{x}_0$
- Instead use EFT $x_j = \widehat{x}_j + \widehat{e}_j$, then combine big with big and small with small

Compensated Residual

- $\widehat{F} = \begin{bmatrix} \widehat{x} \\ \widehat{y} \end{bmatrix}$
- $\widehat{x} = \texttt{CompDeCasteljau}\,(x_0, s) \ominus \texttt{CompDeCasteljau}\,(x_1, t)$
- Fails; $x_0(s), x_1(t)$ may not be near zero, e.g. $\widehat{x}_0 \oplus \widehat{e}_0 = \widehat{x}_0$
- Instead use EFT $x_j = \widehat{x}_j + \widehat{e}_j$, then combine big with big and small with small
- Compensated addition (via `TwoSum`) $\widehat{x}_0 - \widehat{x}_1 = D + \widehat{e}_2$

Compensated Residual

- $\widehat{F} = \begin{bmatrix} \widehat{x} \\ \widehat{y} \end{bmatrix}$
- $\widehat{x} = \texttt{CompDeCasteljau}\,(x_0, s) \ominus \texttt{CompDeCasteljau}\,(x_1, t)$
- Fails; $x_0(s), x_1(t)$ may not be near zero, e.g. $\widehat{x}_0 \oplus \widehat{e}_0 = \widehat{x}_0$
- Instead use EFT $x_j = \widehat{x}_j + \widehat{e}_j$, then combine big with big and small with small
- Compensated addition (via `TwoSum`) $\widehat{x}_0 - \widehat{x}_1 = D + \widehat{e}_2$
- Compensation term $\tau = (\widehat{e}_0 \ominus \widehat{e}_1) \oplus \widehat{e}_2$

Compensated Residual

- $\widehat{F} = \begin{bmatrix} \widehat{x} \\ \widehat{y} \end{bmatrix}$
- $\widehat{x} = \texttt{CompDeCasteljau}\,(x_0, s) \ominus \texttt{CompDeCasteljau}\,(x_1, t)$
- Fails; $x_0(s), x_1(t)$ may not be near zero, e.g. $\widehat{x}_0 \oplus \widehat{e}_0 = \widehat{x}_0$
- Instead use EFT $x_j = \widehat{x}_j + \widehat{e}_j$, then combine big with big and small with small
- Compensated addition (via `TwoSum`) $\widehat{x}_0 - \widehat{x}_1 = D + \widehat{e}_2$
- Compensation term $\tau = (\widehat{e}_0 \ominus \widehat{e}_1) \oplus \widehat{e}_2$
- Computed residual $D \oplus \tau$

Modified Newton's in Practice

$$F(s,t) = \begin{bmatrix} 2(4s^2-1) \\ (2s-1)^2+1 \end{bmatrix} - \begin{bmatrix} 4(4t^2-1) \\ 4(2t-1)^2+1 \end{bmatrix}$$

Modified Newton's in Practice

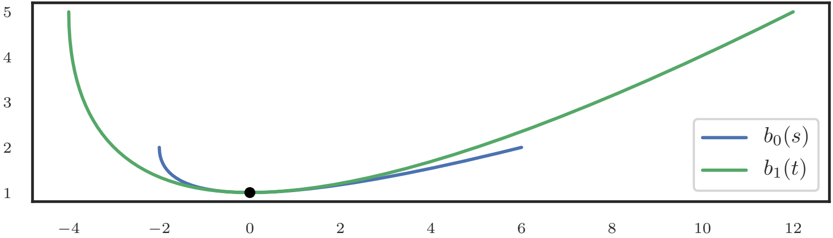
$$F(s,t) = \begin{bmatrix} 2(4s^2 - 1) \\ (2s-1)^2 + 1 \end{bmatrix} - \begin{bmatrix} 4(4t^2 - 1) \\ 4(2t-1)^2 + 1 \end{bmatrix}$$

Triple root at $F\left(\frac{1}{2}, \frac{1}{2}\right)$

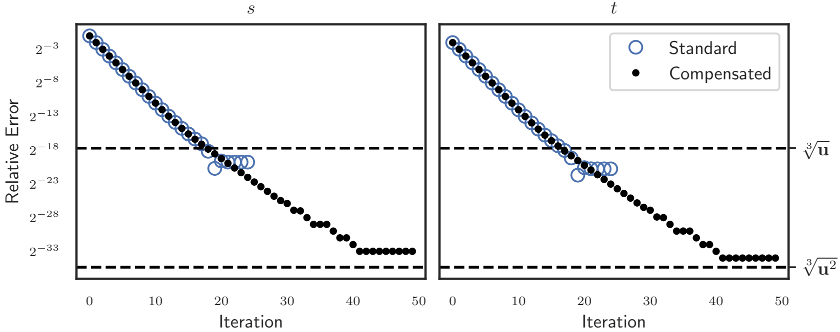
Modified Newton's in Practice

$$F(s,t) = \begin{bmatrix} 2(4s^2-1) \\ (2s-1)^2+1 \end{bmatrix} - \begin{bmatrix} 4(4t^2-1) \\ 4(2t-1)^2+1 \end{bmatrix}$$

Triple root at $F\left(\frac{1}{2}, \frac{1}{2}\right)$



Modified Newton's in Practice



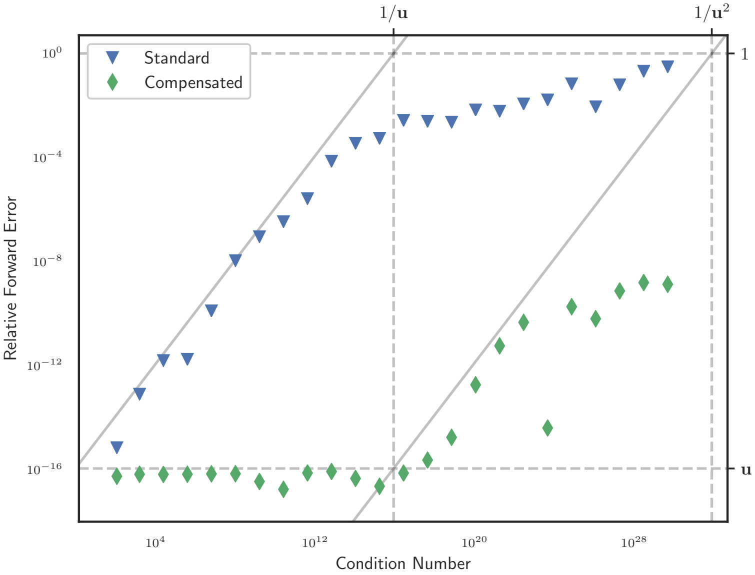
Modified Newton's in Practice

$$G(s,t) = \begin{bmatrix} x_0(s) - r \\ y_0(s) + \frac{1}{r} \end{bmatrix} - \begin{bmatrix} x_1(t) \\ y_1(t) + \frac{1}{r} \end{bmatrix}$$

Modified Newton's in Practice

$$G(s,t) = \begin{bmatrix} x_0(s) - r \\ y_0(s) + \frac{1}{r} \end{bmatrix} - \begin{bmatrix} x_1(t) \\ y_1(t) + \frac{1}{r} \end{bmatrix}$$

Approaches triple root as $r \longrightarrow 0^+$, $F\left(\frac{1+\sqrt{r}}{2}, \frac{2+\sqrt{r}}{4}\right)$



1/u
1/u²
1
u
Standard
Compensated
Relative Forward Error
10^0
10^−4
10^−8
10^−12
10^−16
10^4
10^12
10^20
10^28
Condition Number

Conclusion

Conclusion

Conclusion

- Solution Transfer on Curved Meshes

Conclusion

- Solution Transfer on Curved Meshes
  - Conservative L_2-optimal interpolation; split $\int_\Omega$ into $\sum \int_{\mathcal{T} \cap \mathcal{T}'}$ where integrand is smooth

Conclusion

- Solution Transfer on Curved Meshes
  - Conservative L_2-optimal interpolation; split $\int_\Omega$ into $\sum \int_{\mathcal{T} \cap \mathcal{T}'}$ where integrand is smooth
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$

Conclusion

- Solution Transfer on Curved Meshes
  - Conservative L_2-optimal interpolation; split $\int_\Omega$ into $\sum \int_{\mathcal{T} \cap \mathcal{T}'}$ where integrand is smooth
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$
  - Advancing Front: $(\mathcal{T}, \mathcal{T}') \in \mathcal{M}_T \times \mathcal{M}_D$

Conclusion

- Solution Transfer on Curved Meshes
  - Conservative L_2-optimal interpolation; split $\int_{\Omega}$ into $\sum \int_{\mathcal{T} \cap \mathcal{T}'}$ where integrand is smooth
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$
  - Advancing Front: $(\mathcal{T}, \mathcal{T}') \in \mathcal{M}_T \times \mathcal{M}_D$
  - Integration over Curved Polygons: $\int_{\mathcal{T} \cap \mathcal{T}'} F \, dV$

Conclusion

- Solution Transfer on Curved Meshes
  - Conservative L_2-optimal interpolation; split $\int_\Omega$ into $\sum \int_{\mathcal{T} \cap \mathcal{T}'}$ where integrand is smooth
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$
  - Advancing Front: $(\mathcal{T}, \mathcal{T}') \in \mathcal{M}_T \times \mathcal{M}_D$
  - Integration over Curved Polygons: $\displaystyle\int_{\mathcal{T} \cap \mathcal{T}'} F \, dV$
- Ill-conditioned Bézier Curve Intersection

Conclusion

- Solution Transfer on Curved Meshes
  - Conservative L_2-optimal interpolation; split $\int_{\Omega}$ into $\sum \int_{\mathcal{T} \cap \mathcal{T}'}$ where integrand is smooth
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$
  - Advancing Front: $(\mathcal{T}, \mathcal{T}') \in \mathcal{M}_T \times \mathcal{M}_D$
  - Integration over Curved Polygons: $\int_{\mathcal{T} \cap \mathcal{T}'} F \, dV$
- Ill-conditioned Bézier Curve Intersection
  - Compensated de Castaljau for evaluation of Bézier curves

Conclusion

- Solution Transfer on Curved Meshes
  - Conservative L_2-optimal interpolation; split $\int_{\Omega}$ into $\sum \int_{\mathcal{T}\cap\mathcal{T}'}$ where integrand is smooth
  - Intersecting Curved Elements: $\mathcal{T} \cap \mathcal{T}'$
  - Advancing Front: $(\mathcal{T}, \mathcal{T}') \in \mathcal{M}_T \times \mathcal{M}_D$
  - Integration over Curved Polygons: $\int_{\mathcal{T}\cap\mathcal{T}'} F \, dV$
- Ill-conditioned Bézier Curve Intersection
  - Compensated de Castaljau for evaluation of Bézier curves
  - Modified Newton's method; computes residual $F(s, t)$ as if in extended precision

Future Work

Future Work

- Valid tessellation of curved polygons

Future Work

- Valid tessellation of curved polygons
- Integrand $F = \phi_0 \phi_1$ is polynomial in x, y, could compute coefficients and use them to compute H, V without FTC, without points outside $\mathcal{P}$

Future Work

- Valid tessellation of curved polygons
- Integrand $F = \phi_0\phi_1$ is polynomial in x, y, could compute coefficients and use them to compute H, V without FTC, without points outside $\mathcal{P}$
- Solution transfer in $\mathbf{R}^3$; integrate via Stoke's theorem, geometry is greatest challenge