

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. Compensated Evaluation
5. 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

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

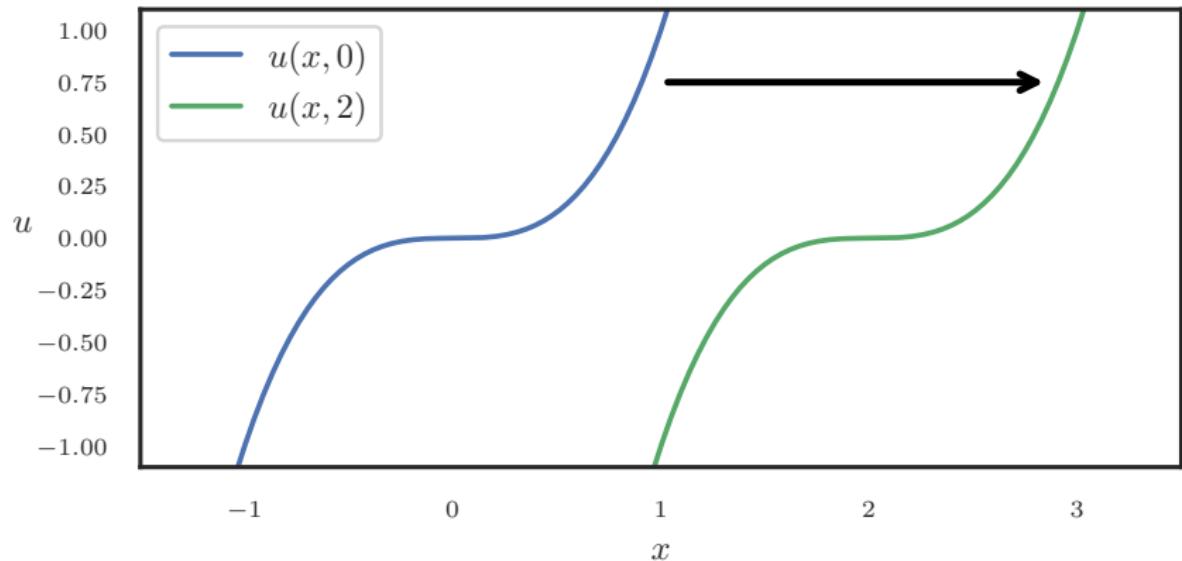
Divide physical domain

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

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

Lagrangian Methods

Add viscosity term to transport equation

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

Lagrangian Methods

Add viscosity term to transport equation

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

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

Lagrangian Methods

Add viscosity term to transport equation

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

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

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

Remeshing and Adaptivity

- 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

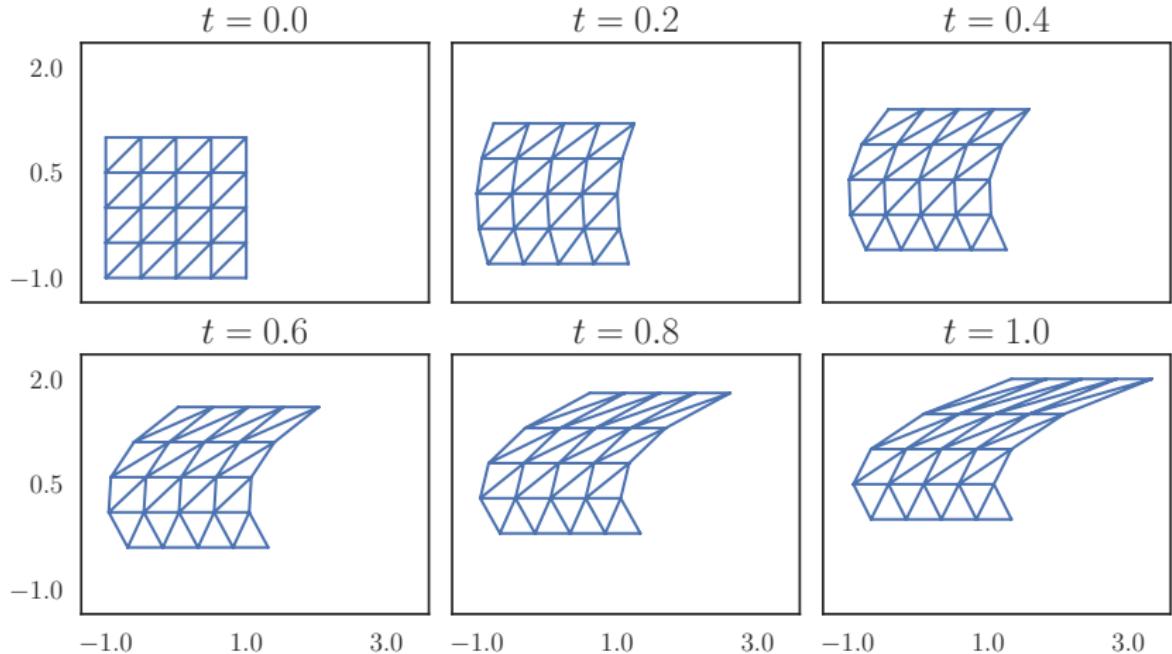
Consider

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

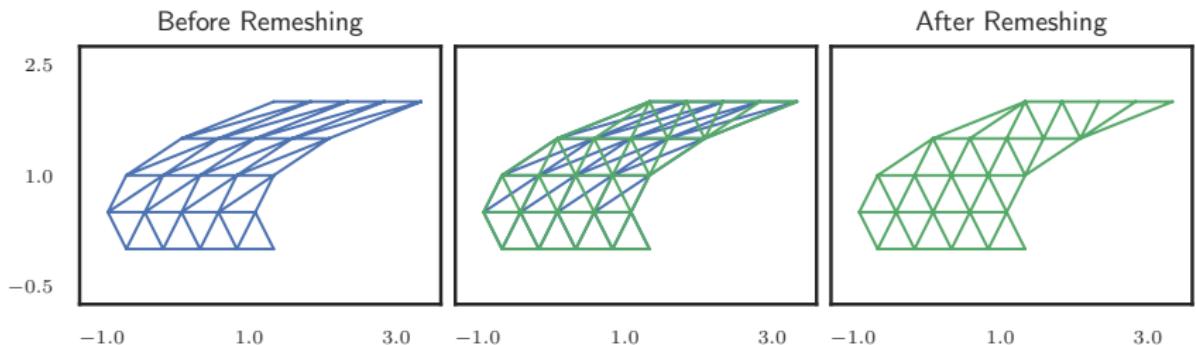
with cubic characteristics

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

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

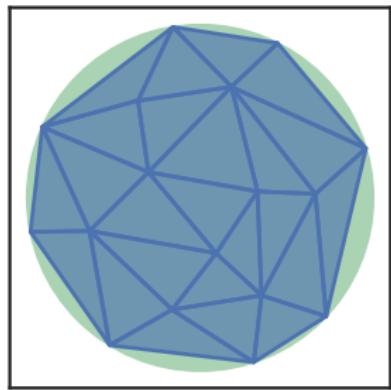
- Benefits
 - High-order shape functions, highly accurate solutions
 - Low dissipation and dispersion error
 - Greater geometric flexibility

Curved Meshes

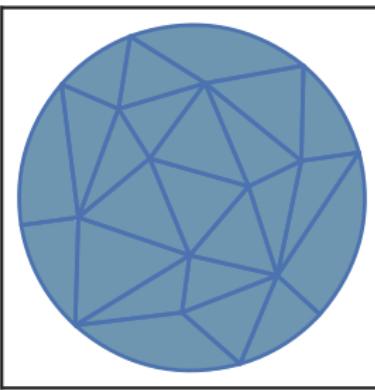
- Benefits
 - High-order shape functions, highly accurate solutions
 - Low dissipation and dispersion error
 - Greater geometric flexibility
 - Fewer elements

Curved Meshes

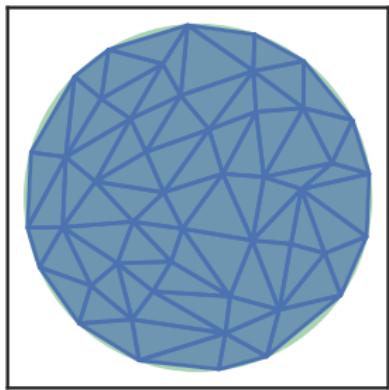
Linear Mesh, 24 elements



Quadratic Mesh, 24 elements



Linear Mesh, 74 elements



-1 0 1 -1 0 1 -1 0 1

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

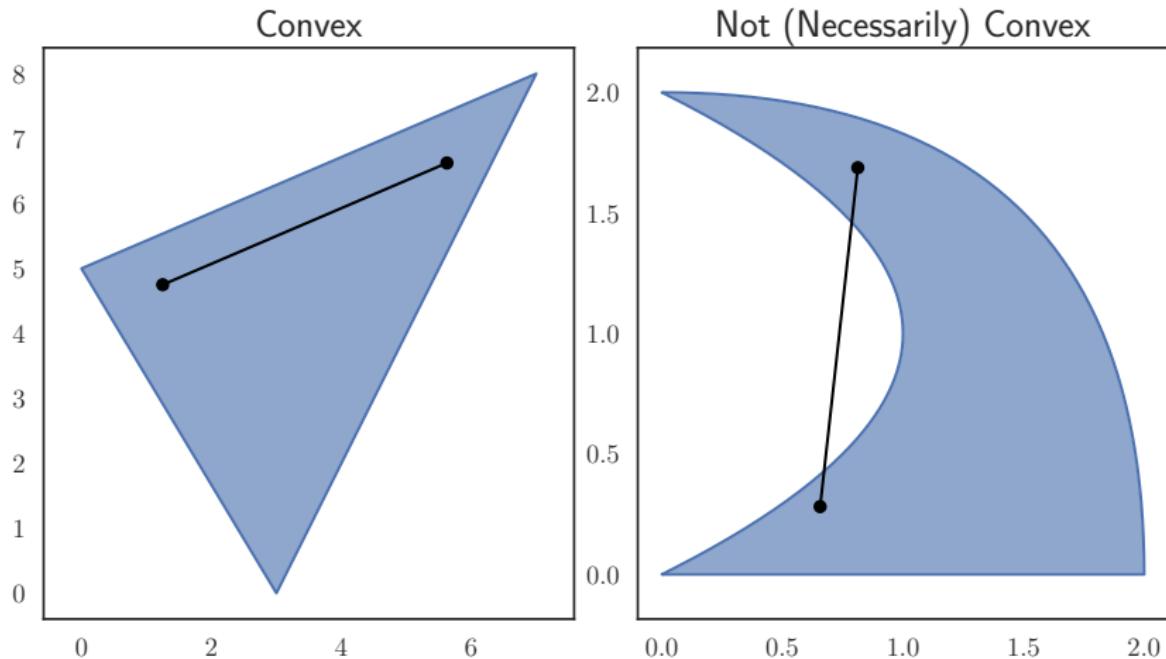
Curved Meshes

- 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

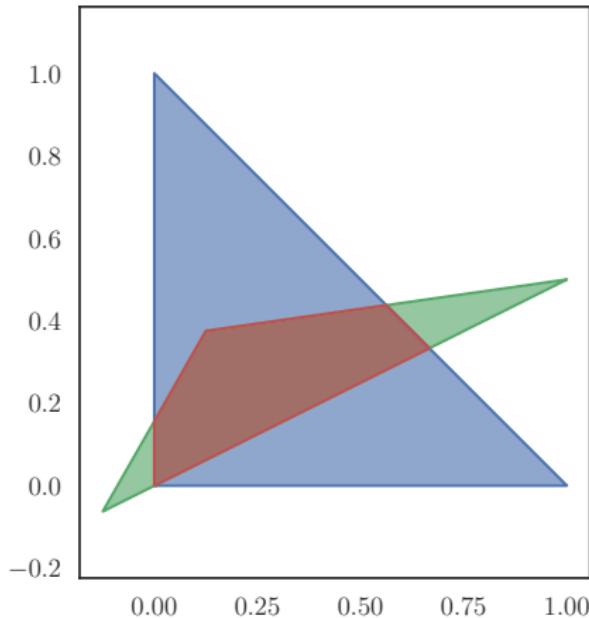
- Benefits
 - High-order shape functions, highly accurate solutions
 - Low dissipation and dispersion error
 - Greater geometric flexibility
 - Fewer elements
- Drawbacks
 - Harder to implement
 - Loss of accuracy in high degree (e.g. Runge's phenomenon)
 - More challenging geometry

Curved Meshes

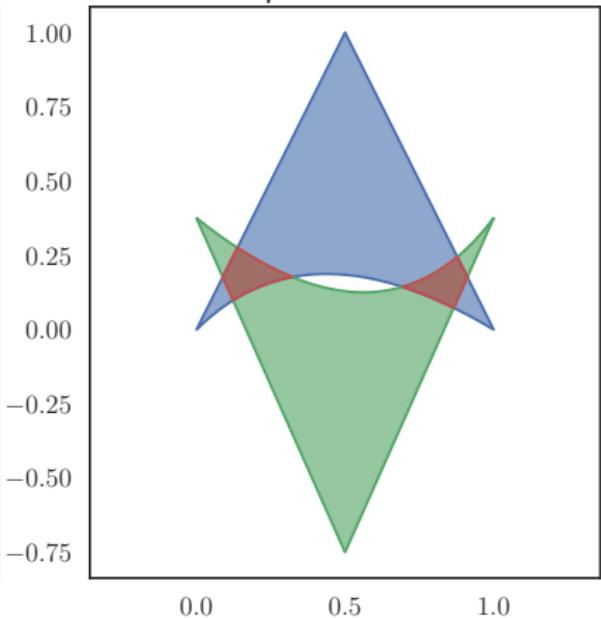


Curved Meshes

Convex Intersection



Multiple Intersections



Curved Elements

- Necessary for High-order

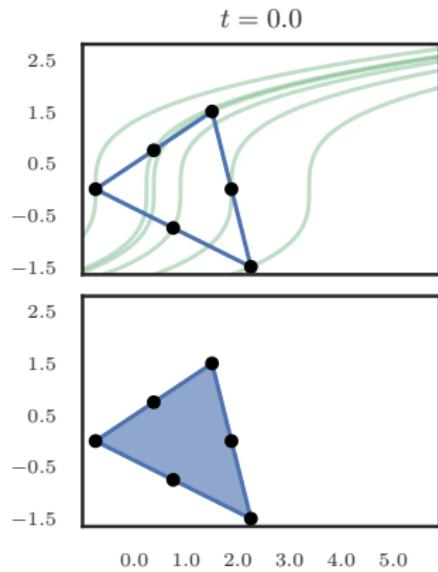
Curved Elements

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

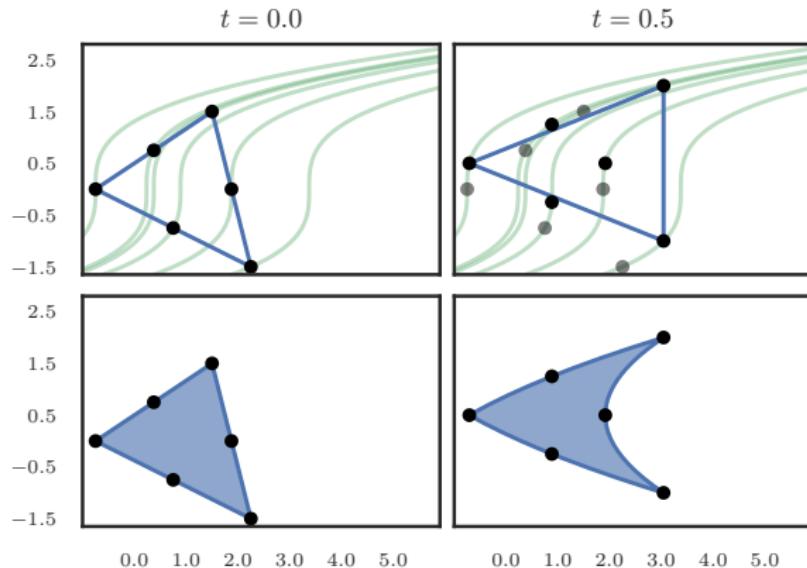
Curved Elements

- 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

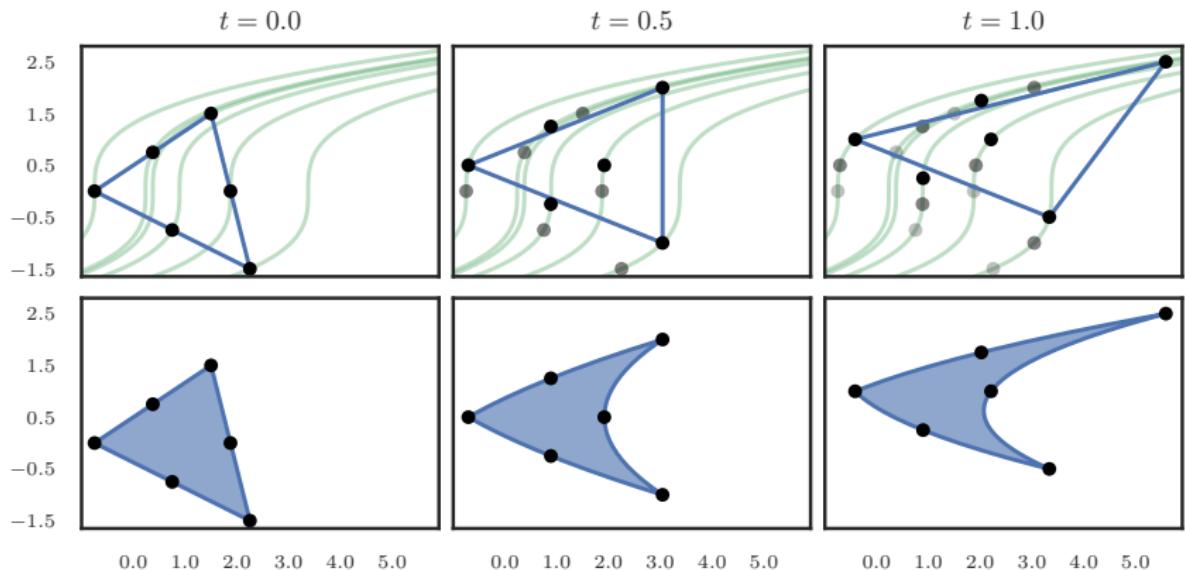
Curved Elements: Necessary for High-order



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

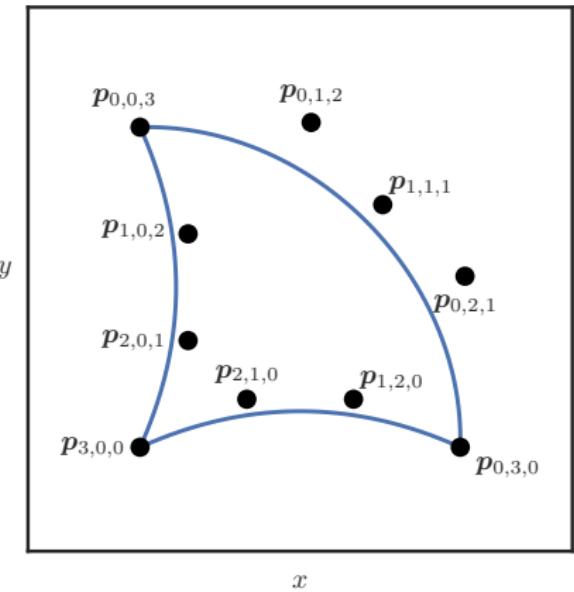
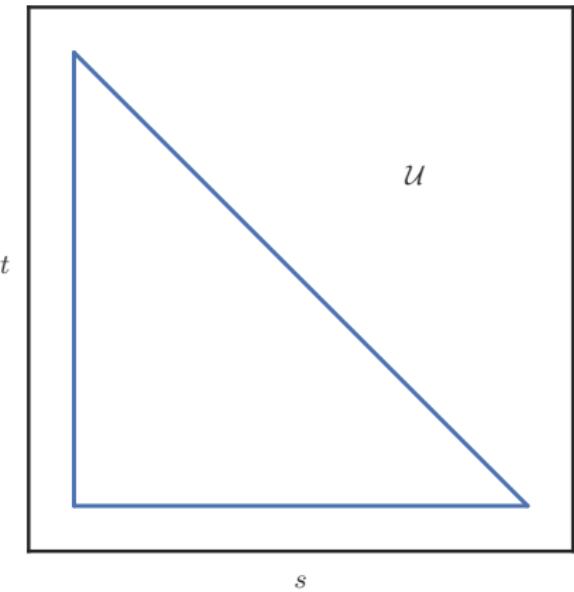
- Image $\mathcal{T} = b(\mathcal{U})$ of reference triangle under polynomial map $b(s, t)$
- Barycentric coordinates $\lambda_1 = 1 - s - t, \lambda_2 = s, \lambda_3 = t$
- Bernstein basis via trinomial expansion:

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

- Convex combination of control points

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

Bézier Triangles



Bézier Triangles

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

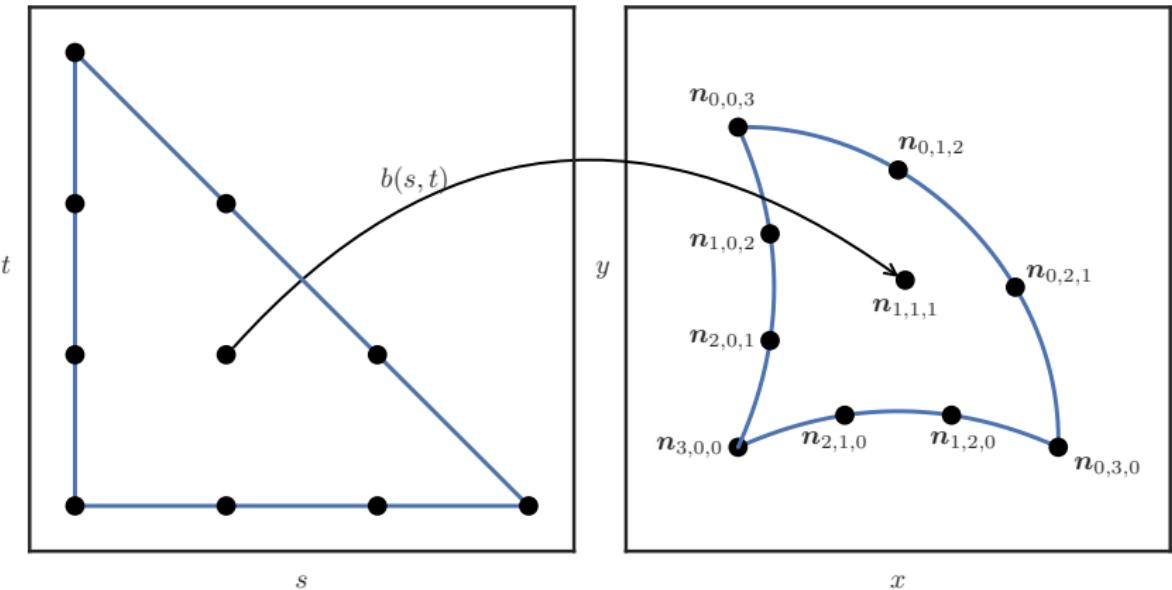
Bézier Triangles

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

Bézier Triangles

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

Bézier Triangles



Bézier Triangles

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

Bézier Triangles

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

Valid Element

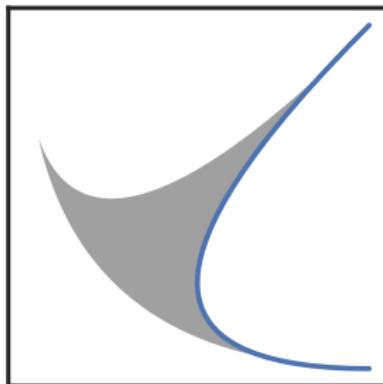
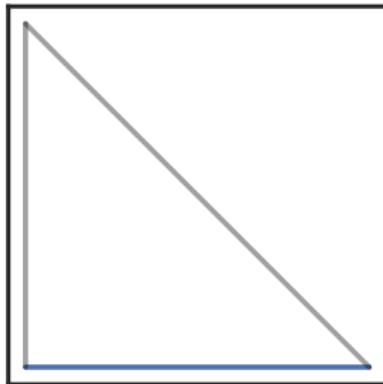
- Element \mathcal{T} is **valid** if diffeomorphic to \mathcal{U}
- $b(s, t)$ bijective, i.e. Jacobian Db is everywhere invertible
- $\det(Db)$ positive, preserves orientation

Inverted Element

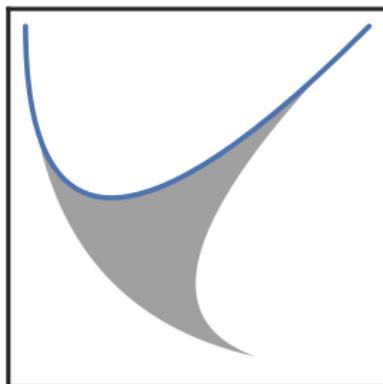
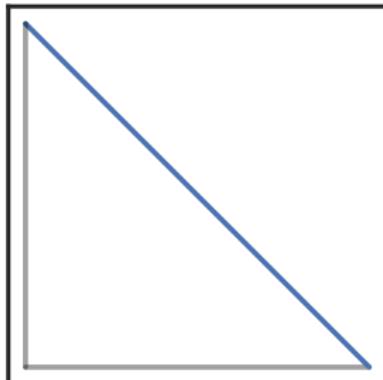
Consider element given by map

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

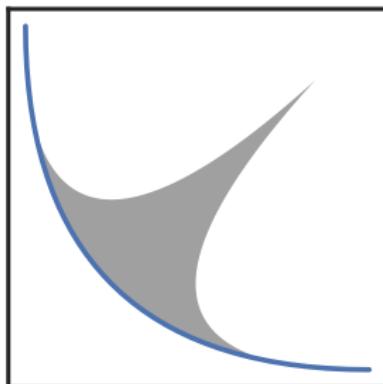
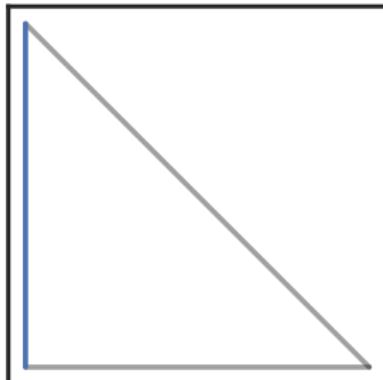
Inverted Element



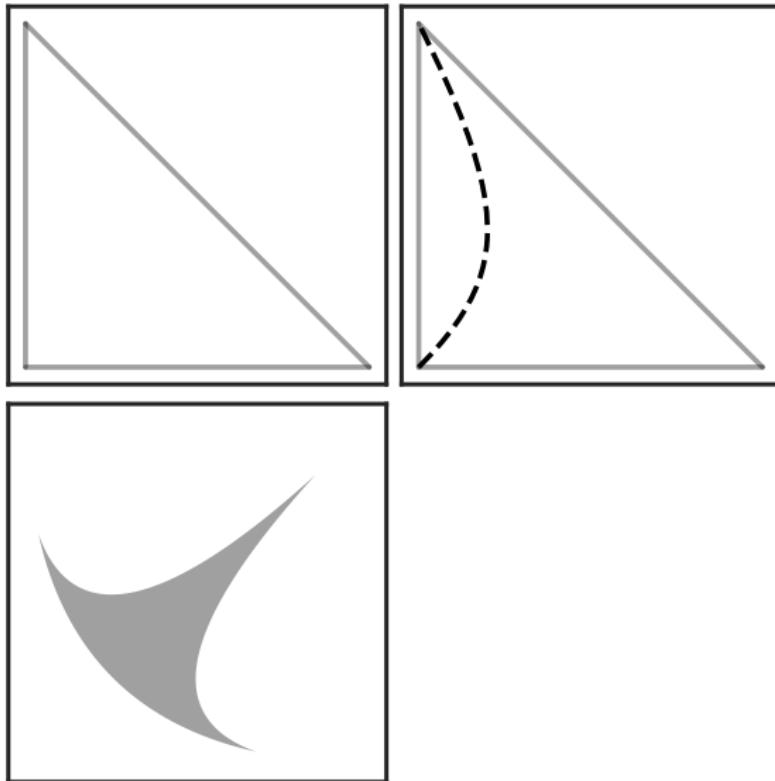
Inverted Element



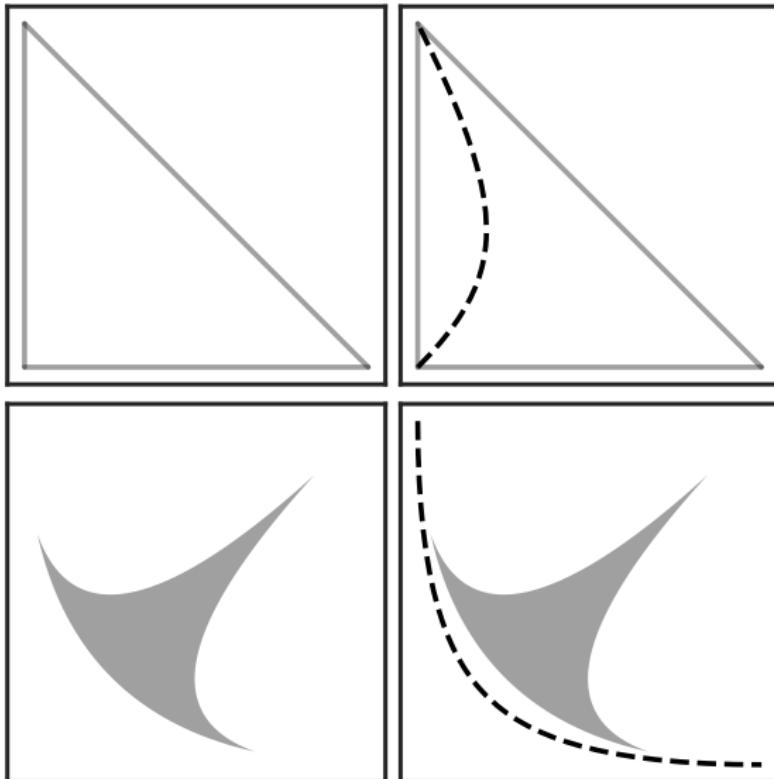
Inverted Element



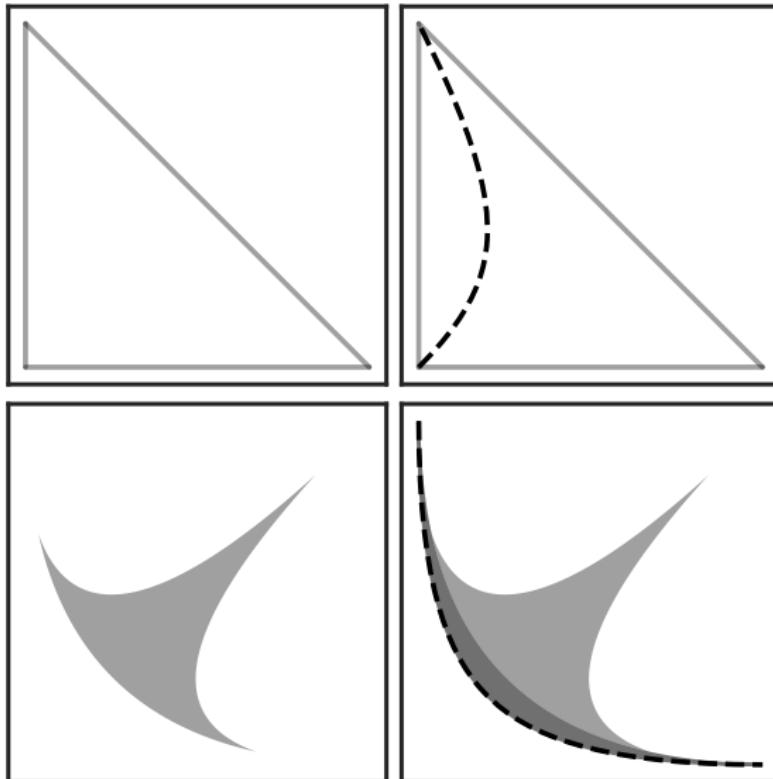
Inverted Element



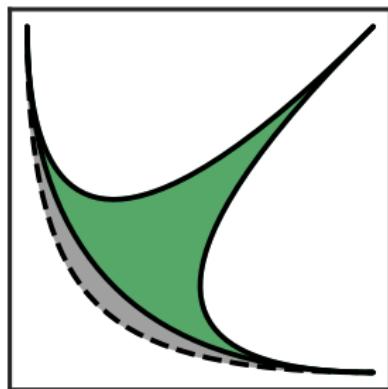
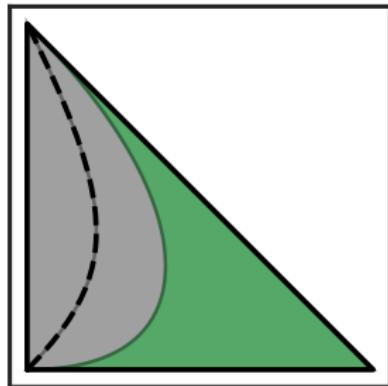
Inverted Element



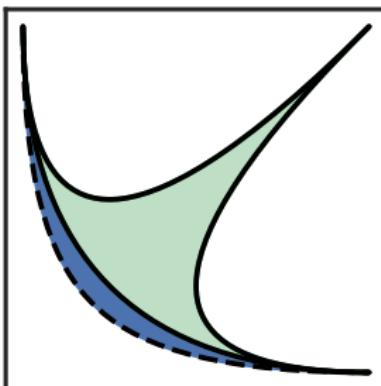
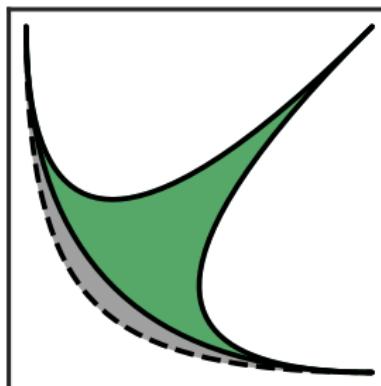
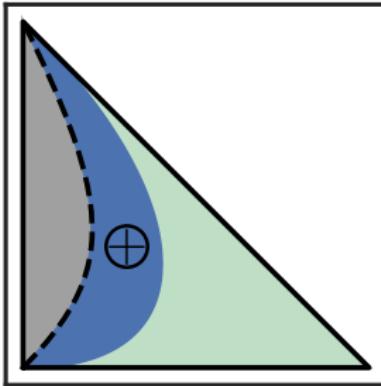
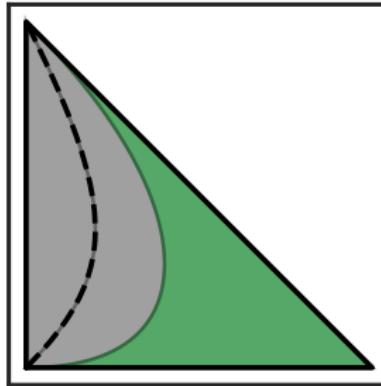
Inverted Element



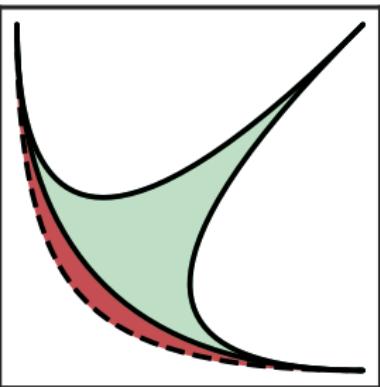
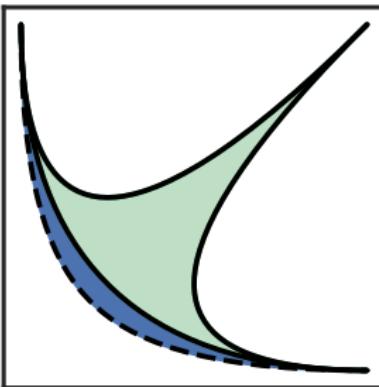
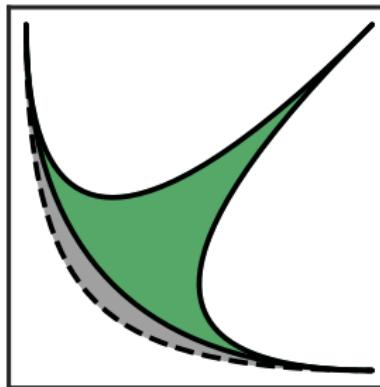
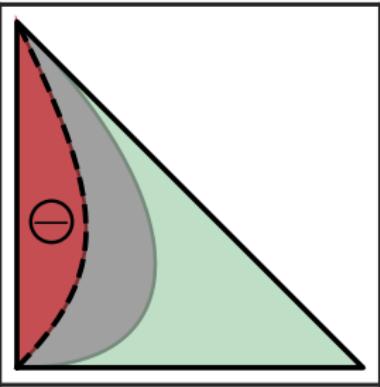
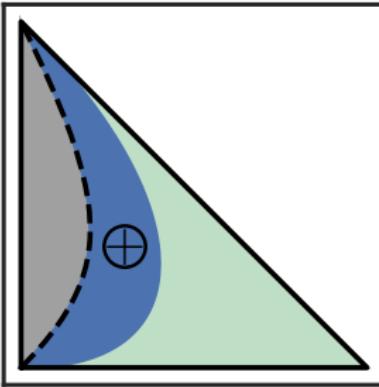
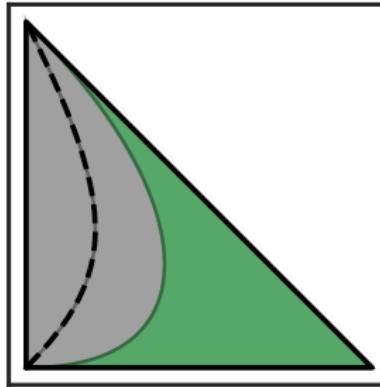
Inverted Element



Inverted Element



Inverted Element



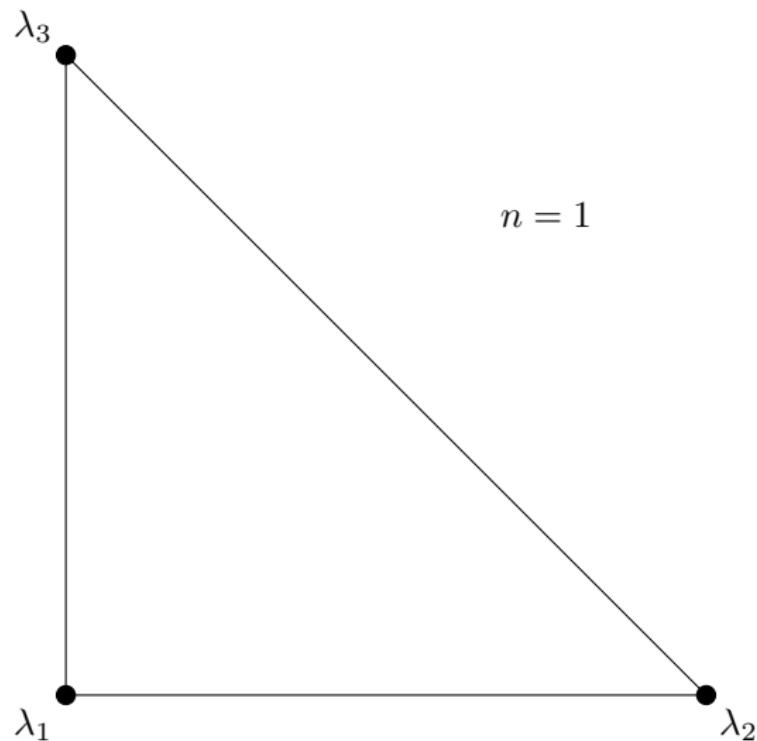
Shape Functions

- Based on $u_\alpha \in \mathcal{U}$ or $n_\alpha \in \mathbf{R}^2$ (α is a multi-index)

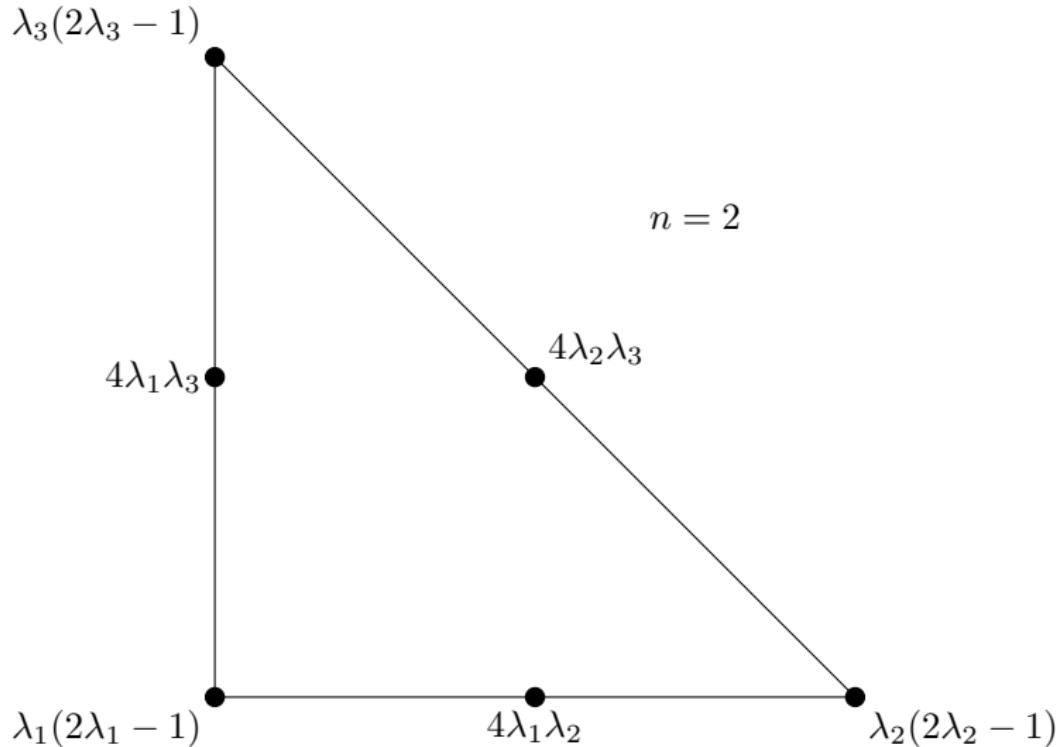
Shape Functions

- Based on $u_\alpha \in \mathcal{U}$ or $n_\alpha \in \mathbf{R}^2$ (α is a multi-index)
- Pre-Image Basis: $\phi_\alpha(n_\beta) = \hat{\phi}_\alpha(u_\beta) = \hat{\phi}_\alpha(b^{-1}(n_\beta))$

Shape Functions



Shape Functions



Shape Functions

- Based on $u_\alpha \in \mathcal{U}$ or $n_\alpha \in \mathbf{R}^2$ (α is a multi-index)
- Pre-Image Basis: $\phi_\alpha (n_\beta) = \hat{\phi}_\alpha (u_\beta) = \hat{\phi}_\alpha (b^{-1} (n_\beta))$

Shape Functions

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

Shape Functions

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

Shape Functions

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

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 $\mathbf{q}_D = \sum_j d_j \phi_D^{(j)}$

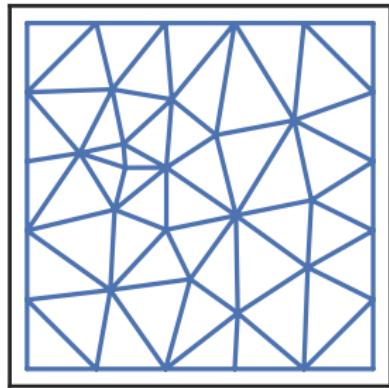
Galerkin Projection

- 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 $\mathbf{q}_D = \sum_j d_j \phi_D^{(j)}$
- Want: L_2 -optimal interpolant $\mathbf{q}_T = \sum_j t_j \phi_T^{(j)}$:

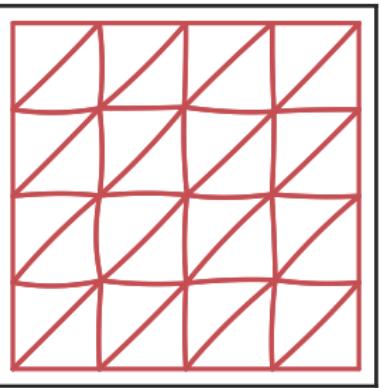
$$\|\mathbf{q}_T - \mathbf{q}_D\|_2 = \min_{\mathbf{q} \in \mathcal{V}_T} \|\mathbf{q} - \mathbf{q}_D\|_2$$

Galerkin Projection

\mathcal{M}_T



\mathcal{M}_D



Galerkin Projection

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

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

Galerkin Projection

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

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

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

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

Linear System

Weak form gives rise to a linear system in coefficients \mathbf{d} and t .

Linear System

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

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

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

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

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

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

Linear System

Mixed mass matrix M_{TD}

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

Linear System

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.

Linear System

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

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

Common Refinement

Given ϕ supported on \mathcal{T}

$$\int_{\Omega} \phi \mathbf{q}_D dV$$

Common Refinement

Given ϕ supported on \mathcal{T}

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

Common Refinement

Given ϕ supported on \mathcal{T}

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

Common Refinement

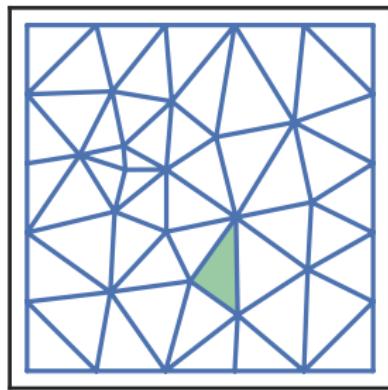
Given ϕ supported on \mathcal{T}

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

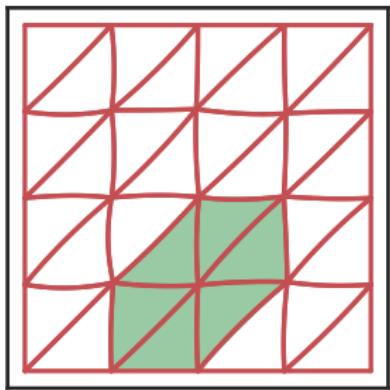
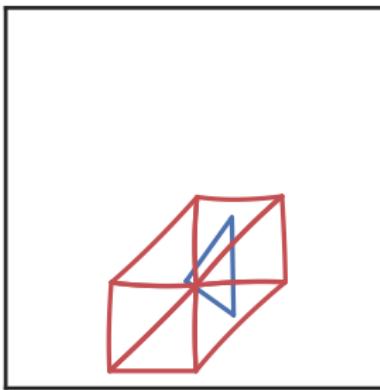
In CG, \mathbf{q}_D need not be differentiable across elements and in DG \mathbf{q}_D need not even be continuous

Common Refinement

\mathcal{M}_T



\mathcal{M}_D



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: $\int_{\mathcal{T} \cap \mathcal{T}'} F dV$

Intersecting Curved Elements

Intersecting Curved Elements

- Find all points where edges intersect

Intersecting Curved Elements

- Find all points where edges intersect
- $\mathcal{T}_0 = \textcolor{blue}{b}_0(\mathcal{U})$, $\partial\mathcal{T}_0 = \textcolor{blue}{E}_0 \cup \textcolor{blue}{E}_1 \cup \textcolor{blue}{E}_2$
- $\mathcal{T}_1 = \textcolor{green}{b}_1(\mathcal{U})$, $\partial\mathcal{T}_1 = \textcolor{green}{E}_3 \cup \textcolor{green}{E}_4 \cup \textcolor{green}{E}_5$

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

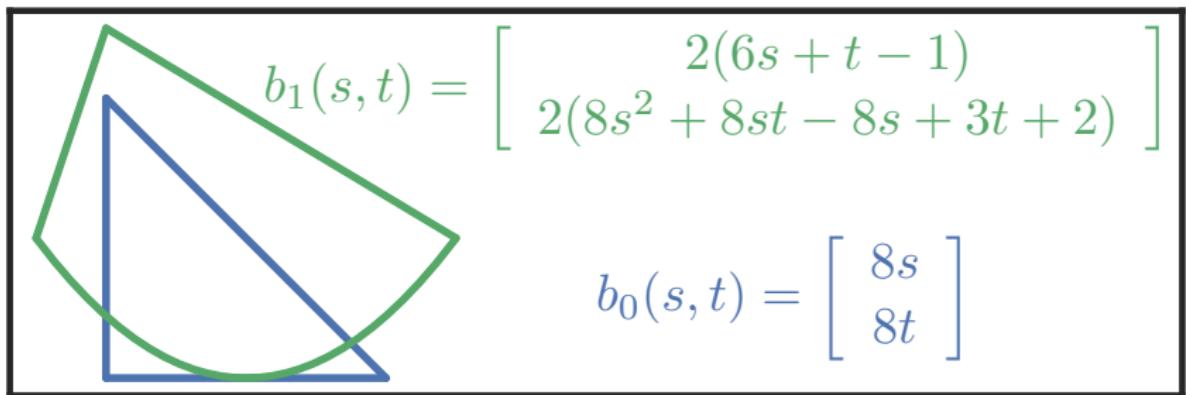
Intersecting Curved Elements

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

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

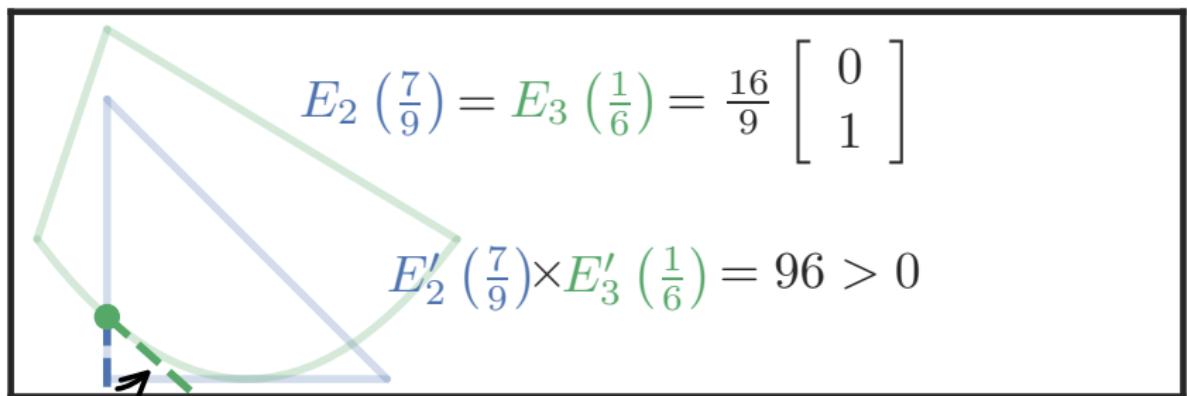
Intersecting Curved Elements



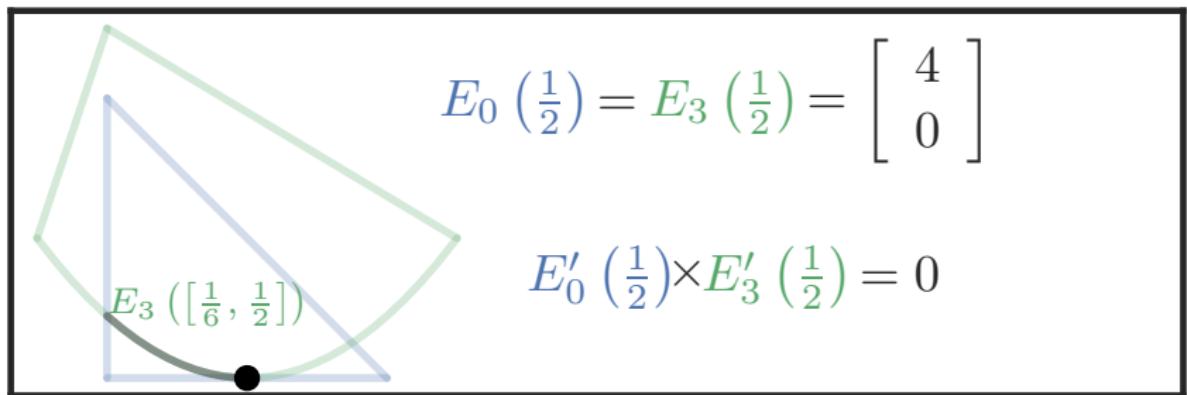
Intersecting Curved Elements

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

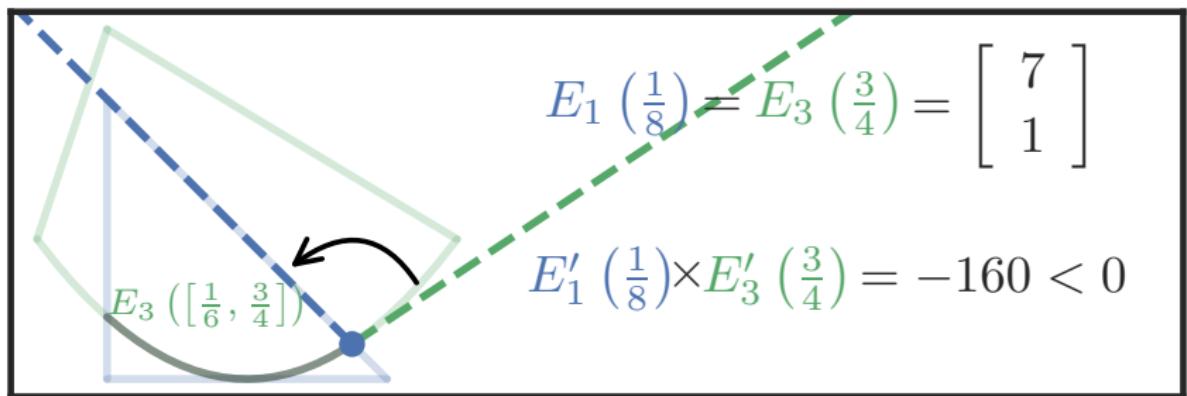
$$E'_2 \left(\frac{7}{9} \right) \times E'_3 \left(\frac{1}{6} \right) = 96 > 0$$



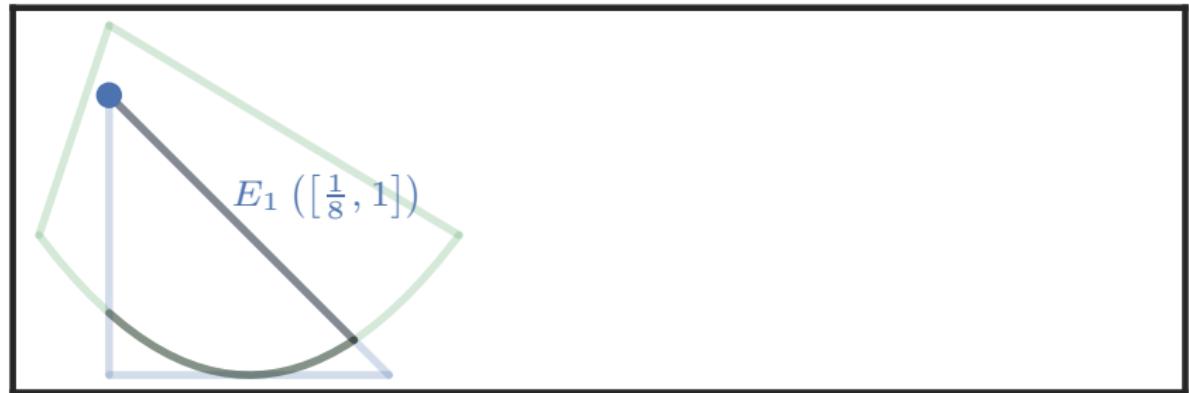
Intersecting Curved Elements



Intersecting Curved Elements



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

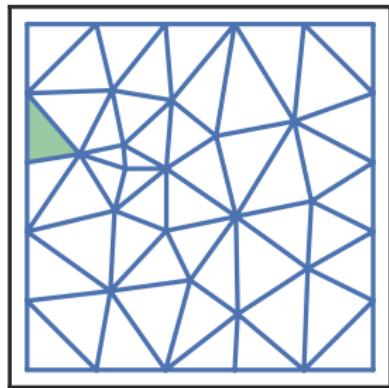
- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements

Advancing Front

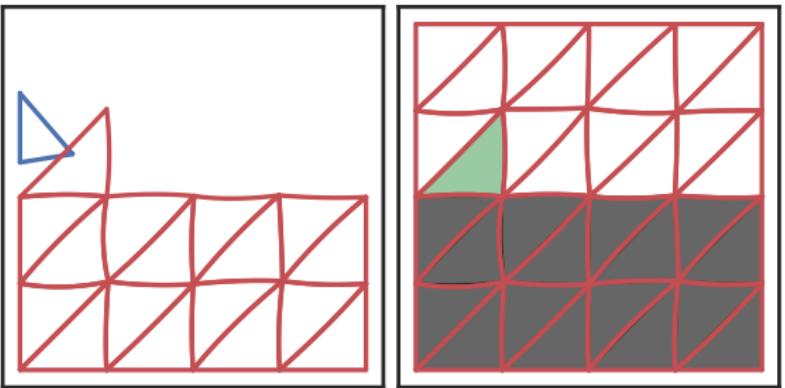
- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element \mathcal{T} , perform brute-force search

Advancing Front

\mathcal{M}_T



\mathcal{M}_D



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
 - $\mathcal{O}(|\mathcal{M}_D|)$

Advancing Front

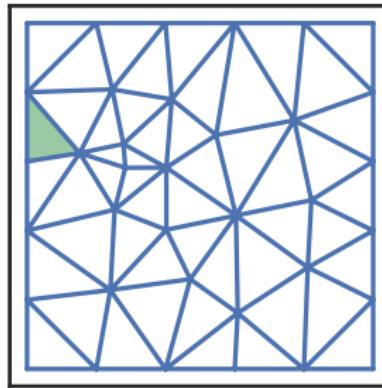
- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element \mathcal{T} , perform brute-force search
 - $\mathcal{O}(|\mathcal{M}_D|)$
 - $\mathcal{O}(1)$ in typical ALE, though acts as CFL

Advancing Front

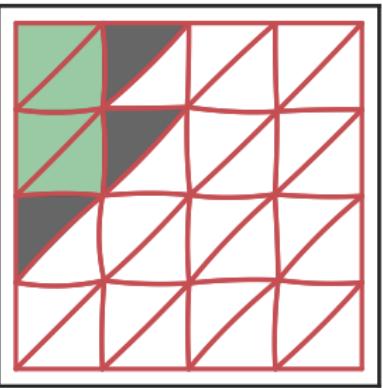
- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element \mathcal{T} , perform brute-force search
 - $\mathcal{O}(|\mathcal{M}_D|)$
 - $\mathcal{O}(1)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for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

Advancing Front

\mathcal{M}_T



\mathcal{M}_D



Advancing Front

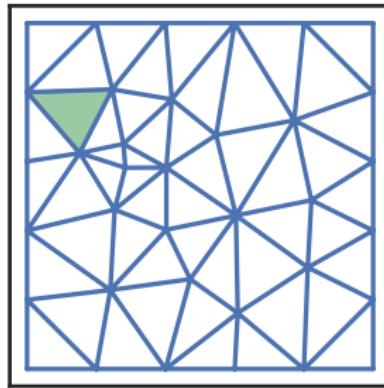
- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element \mathcal{T} , perform brute-force search
 - $\mathcal{O}(|\mathcal{M}_D|)$
 - $\mathcal{O}(1)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for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

Advancing Front

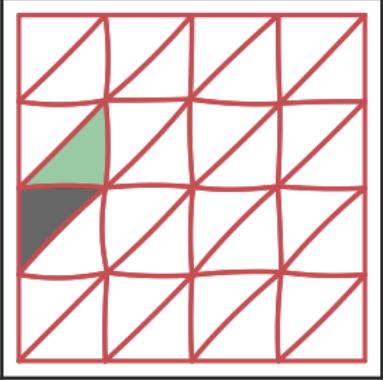
- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element \mathcal{T} , perform brute-force search
 - $\mathcal{O}(|\mathcal{M}_D|)$
 - $\mathcal{O}(1)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for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

Advancing Front

\mathcal{M}_T



\mathcal{M}_D



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)
- Neighbors of \mathcal{T} can find one matching donor element among $\mathcal{N}_{\mathcal{T}}$ in $\mathcal{O}(|\mathcal{N}_{\mathcal{T}}|) = \mathcal{O}(1)$

Advancing Front

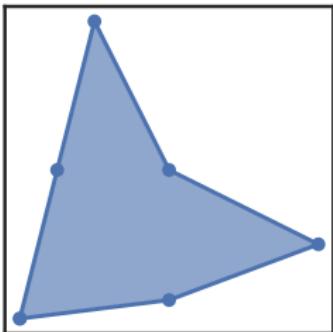
- Find all pairs $\mathcal{T}, \mathcal{T}'$ of intersecting target and donor elements
- Fix target element \mathcal{T} , perform brute-force search
 - $\mathcal{O}(|\mathcal{M}_D|)$
 - $\mathcal{O}(1)$ in typical ALE, though acts as CFL
- Use donor mesh connectivity to perform breadth first search for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

Integration over Curved Polygons

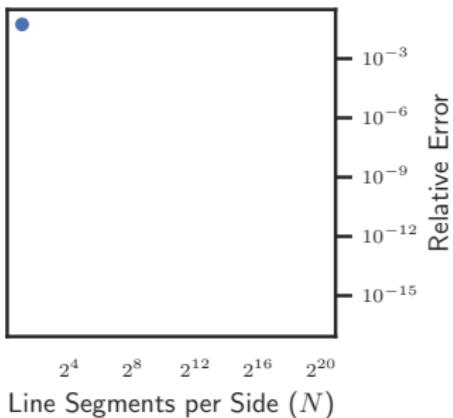
$$\int_{\mathcal{P}} F(x, y) \, dV, \quad \mathcal{P} = \mathcal{T}_0 \cap \mathcal{T}_1, \quad F = \phi_0 \phi_1$$

Integrate via Polygonal Approximation

$$N = 2$$

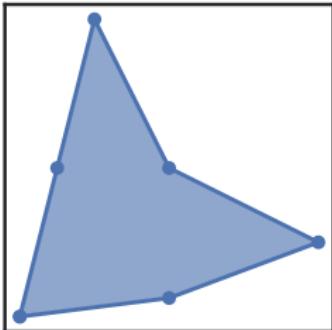


Area Estimates

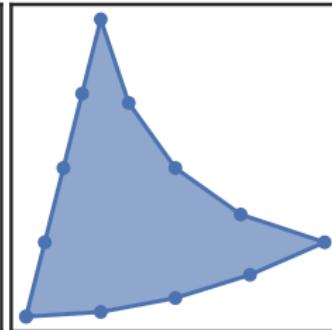


Integrate via Polygonal Approximation

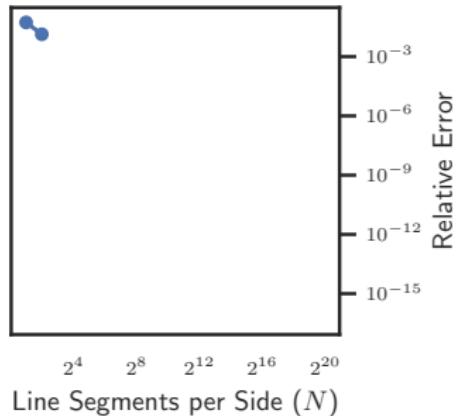
$N = 2$



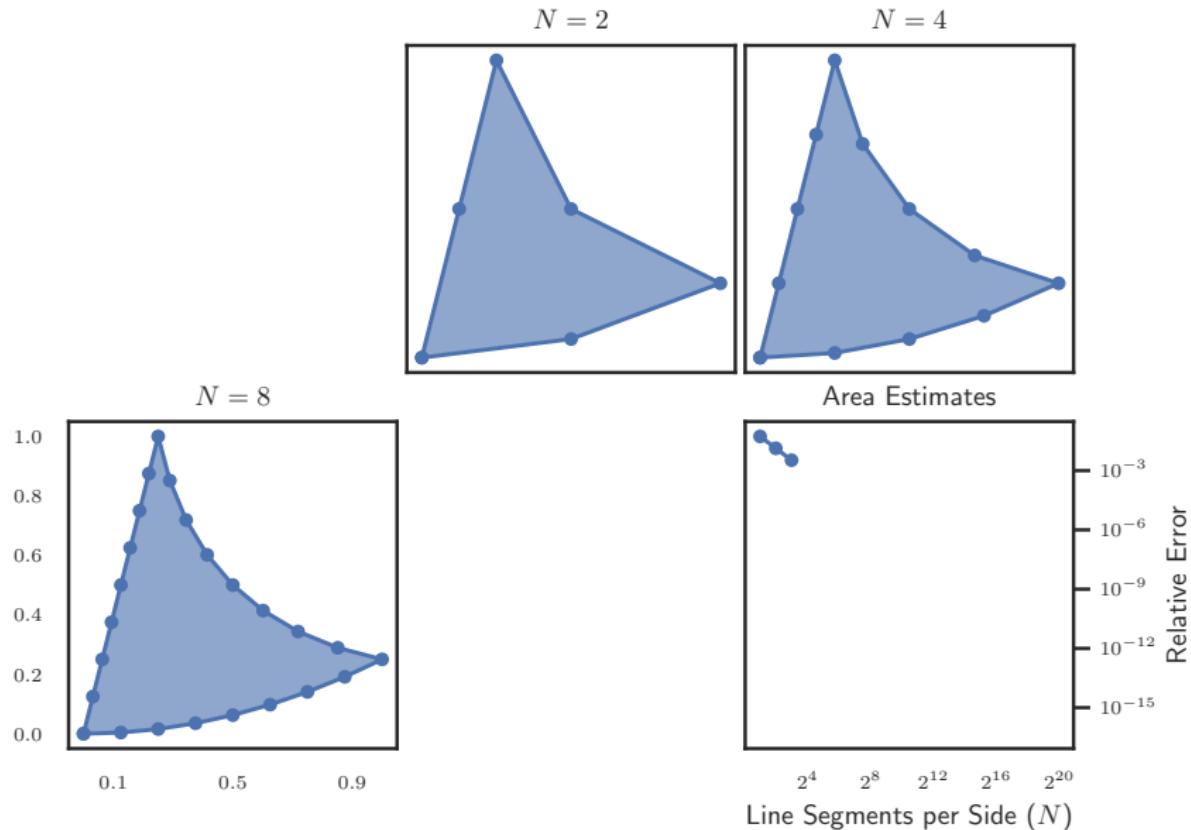
$N = 4$



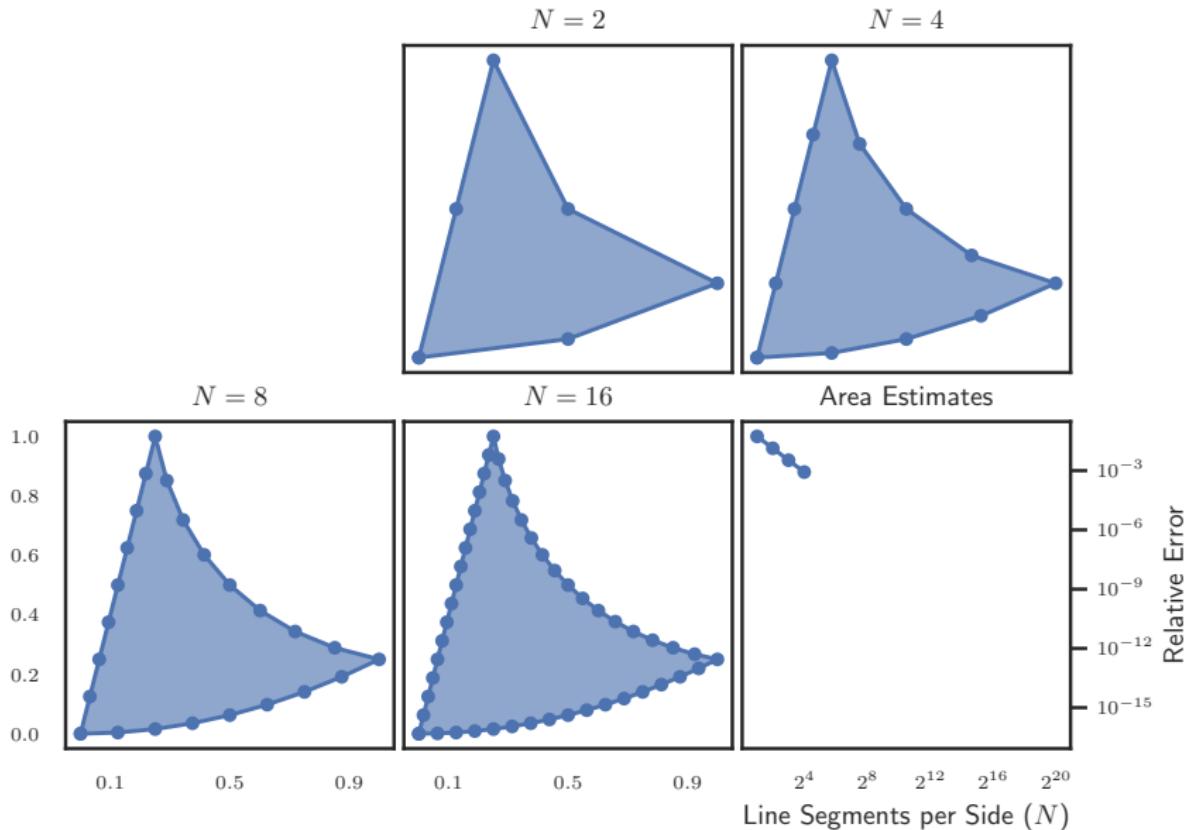
Area Estimates



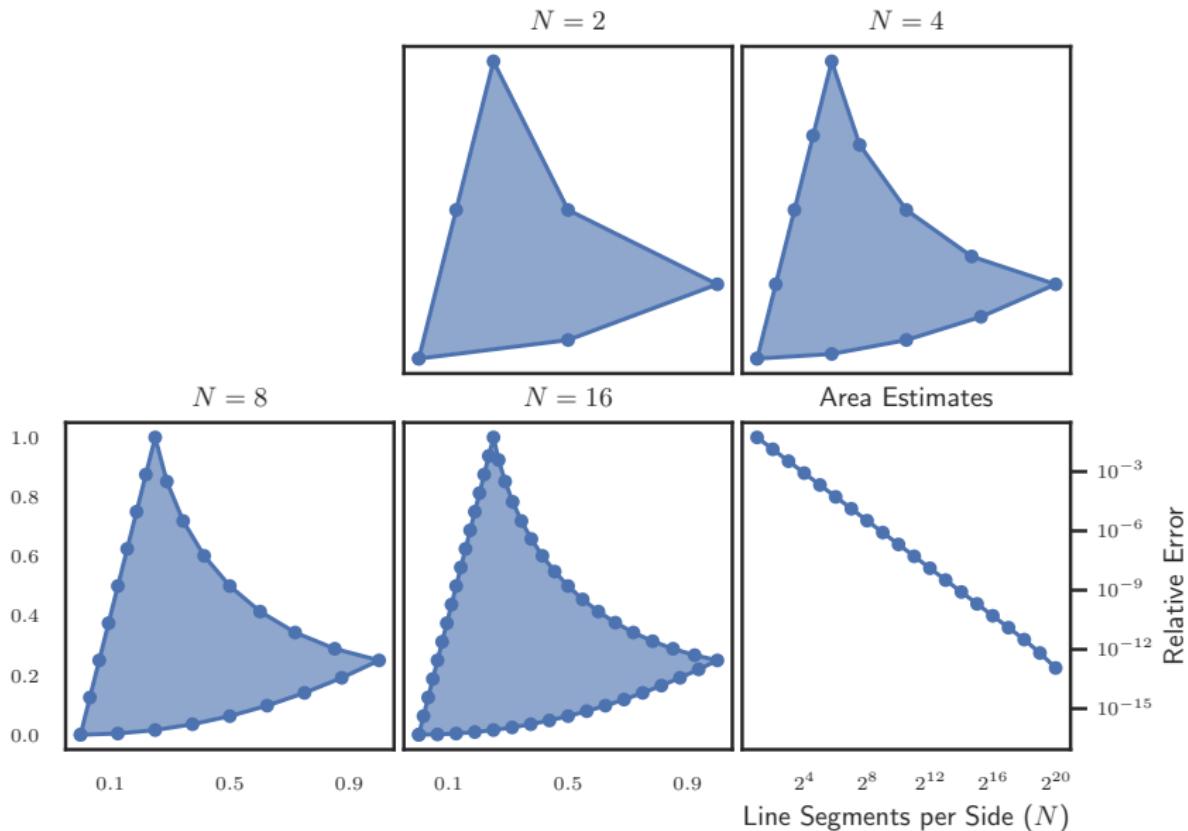
Integrate via Polygonal Approximation



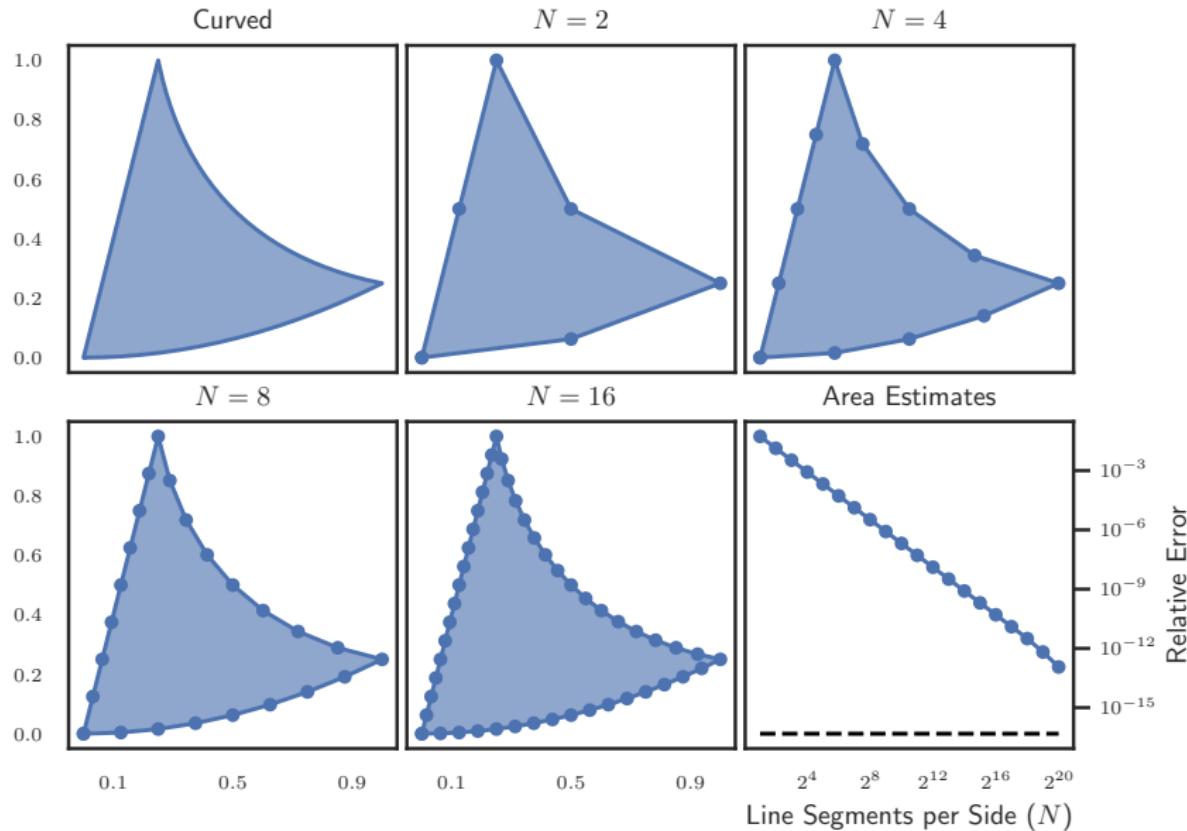
Integrate via Polygonal Approximation



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

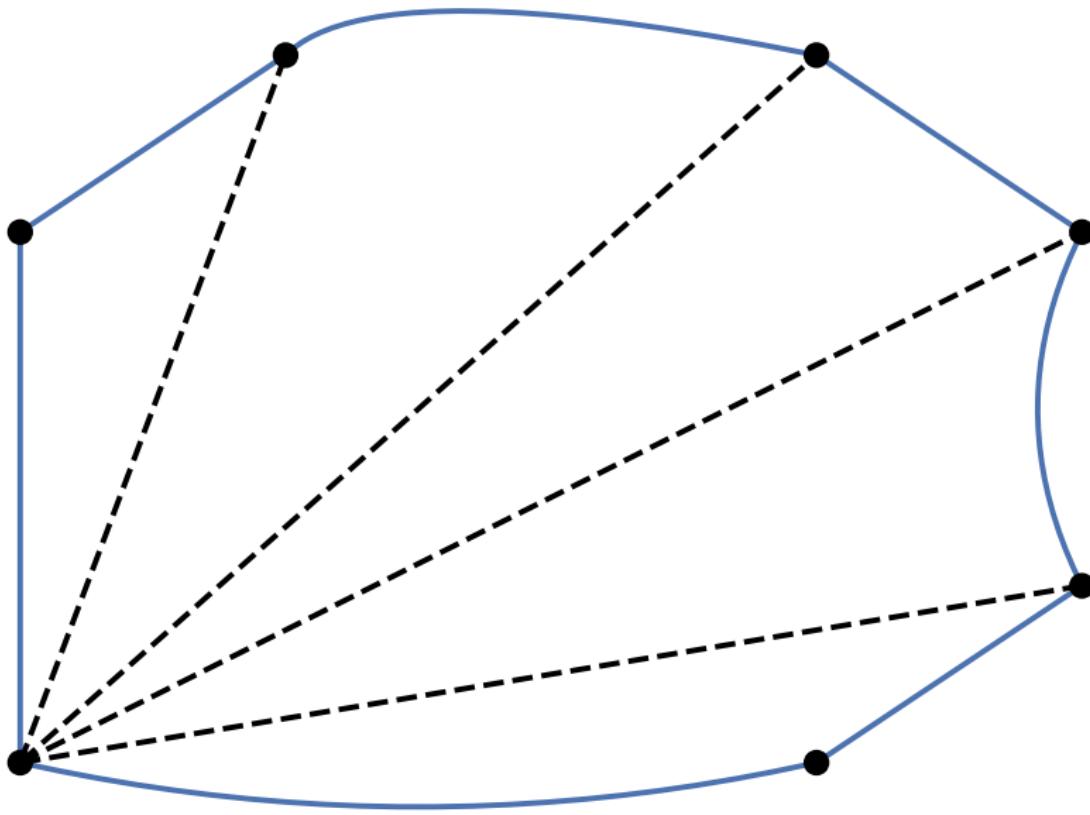
Integrate via Tessellation

- 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

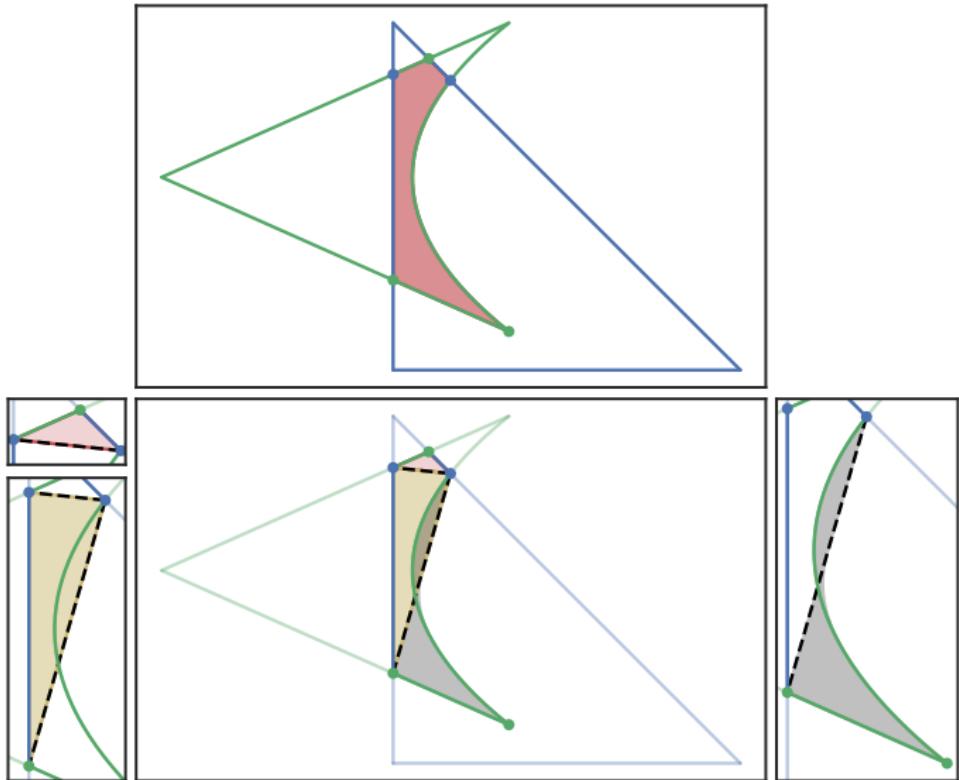
Integrate via Tessellation

- 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 $\mathcal{T}_2 = b(\mathcal{U})$, then numerically integrate

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

Integrate via Tessellation

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

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

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

Integrate via Tessellation

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

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

- $|\det(Db)|$ 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 $\mathcal{T}_2 = b(\mathcal{U})$, then numerically integrate

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

- $|\det(Db)|$ 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 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 $\partial\mathcal{P} = C_1 \cup \dots \cup C_n$, 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 $\partial\mathcal{P} = C_1 \cup \dots \cup C_n$, 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

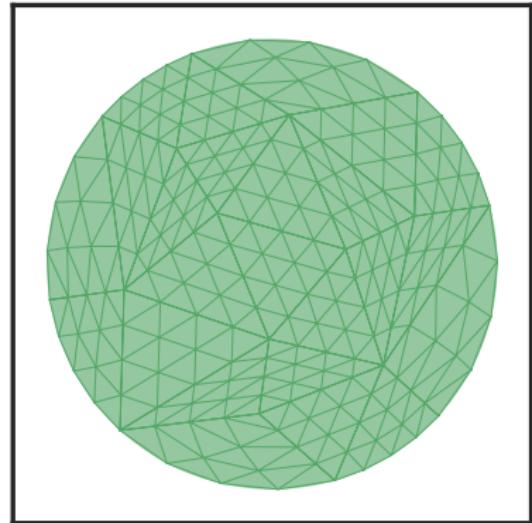
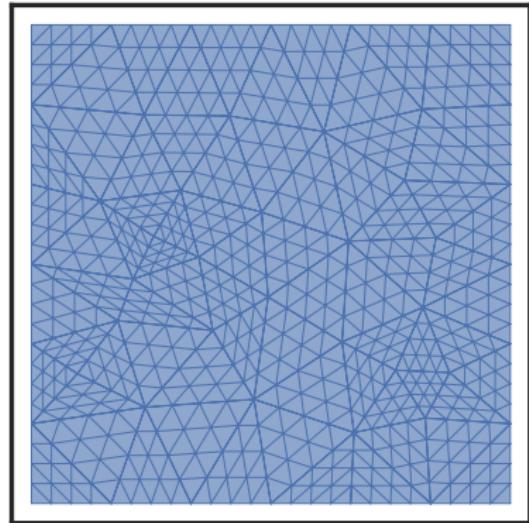
$$H(\alpha, \beta) = H(\alpha, \beta) - H(0, \beta) = \int_0^\alpha F(x, \beta) \, dx$$

- 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



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
 - $\zeta_1(x, y) = 5y^3 + x^2 + 2y + 3$
 - $\zeta_2(x, y) = \exp(x^2) + 2y$

Numerical Experiments

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

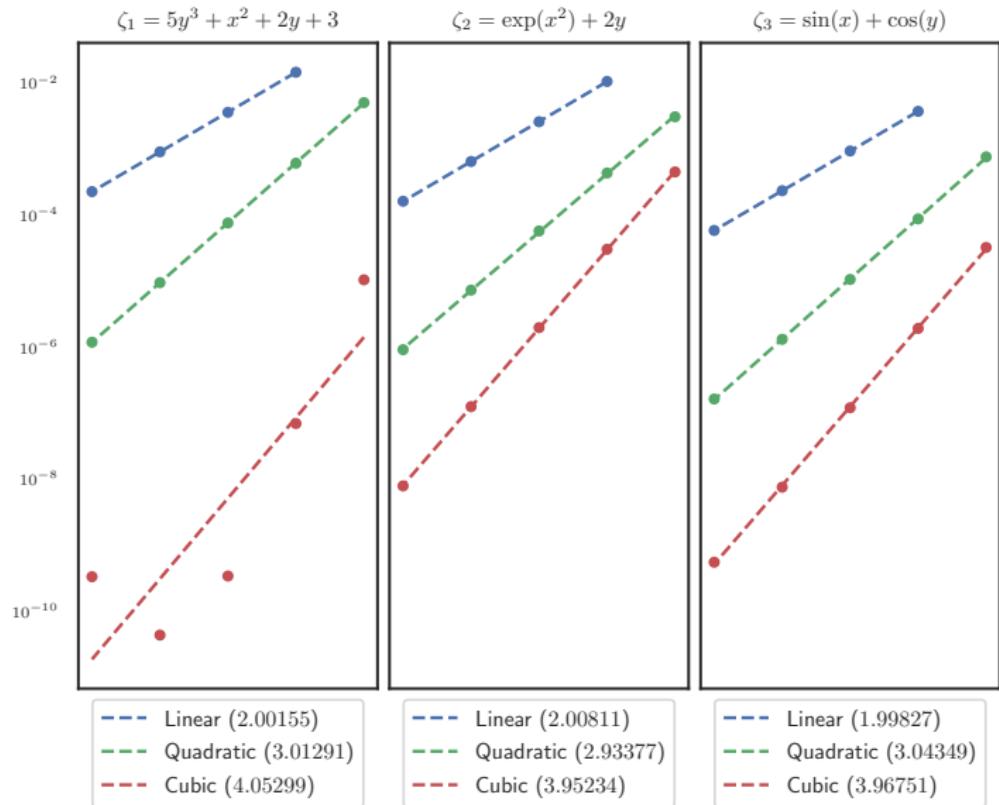
Numerical Experiments

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

Numerical Experiments

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

Numerical Experiments



Compensated Evaluation

Modified Newton's for Intersection
