CHARMS: A Simple Framework for Adaptive Simulation

SIGGRAPH 2002

Eitan Grinspun	Petr Krysl	Peter Schröder
Caltech	UCSD	Caltech

Presented by Jose Guerra

Outline

- ***Background***
- Motivation (Element vs. Basis Refinement)
- Implementation
  - Definitions
  - Data Structures
  - Algorithms
- Example Applications
- Conclusion

Adaptive Solvers

- Focus computational **resources**
- Improve **scalability**
- Improve **accuracy**
- Generally difficult to **implement**

Finite Difference Method

- Approximate the solution domain by a **discrete grid** of **uniformly spaced nodes**
- System of **algebraic equations** with references to **adjacent nodes**
- **Adaptive** discretization is generally very **difficult**

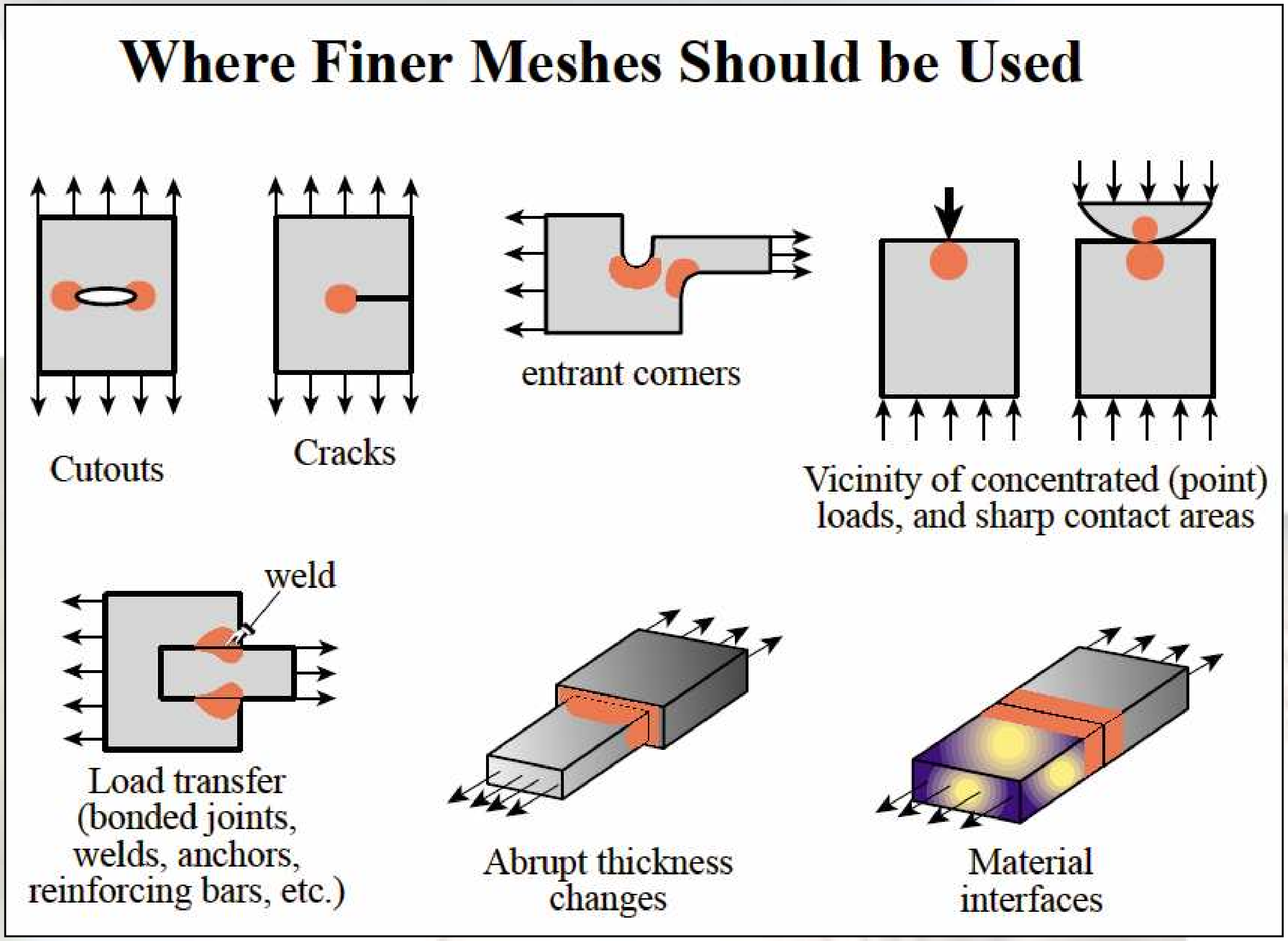
[http://csep1.phy.ornl.gov/CSEP/BF/NODE8.html]

Finite Element Method

- More **robust**, **accurate** and with more **mathematical background**
- Discretize the solution domain by a number of **uniform or non-uniform finite elements** connected by nodes
- An **interpolation function** is used to approximate the change of the dependent variable

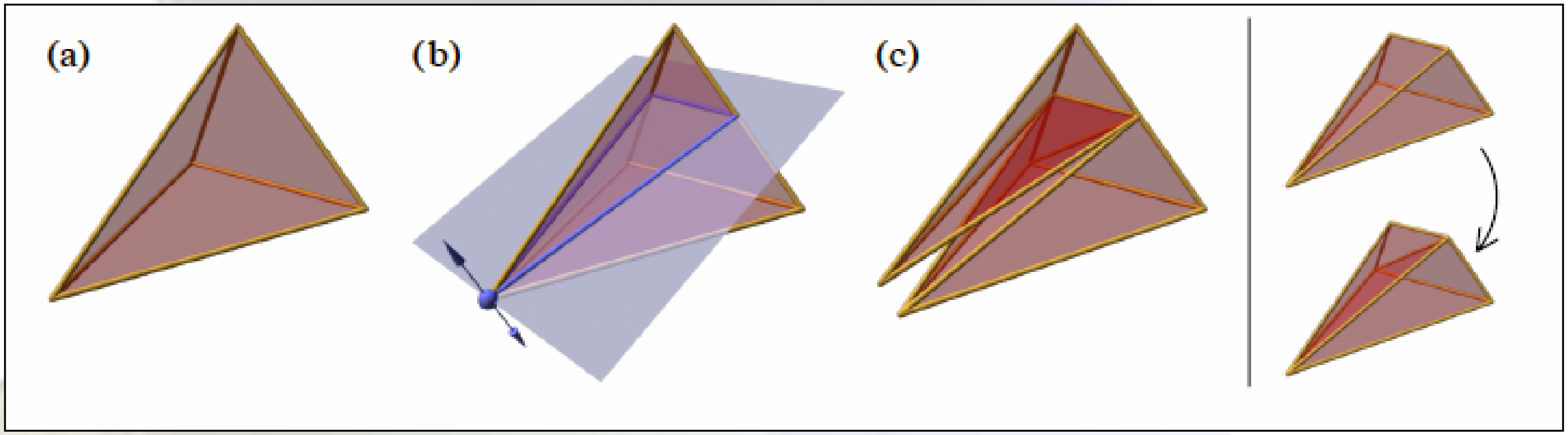
[http://csep1.phy.ornl.gov/CSEP/BF/NODE8.html]

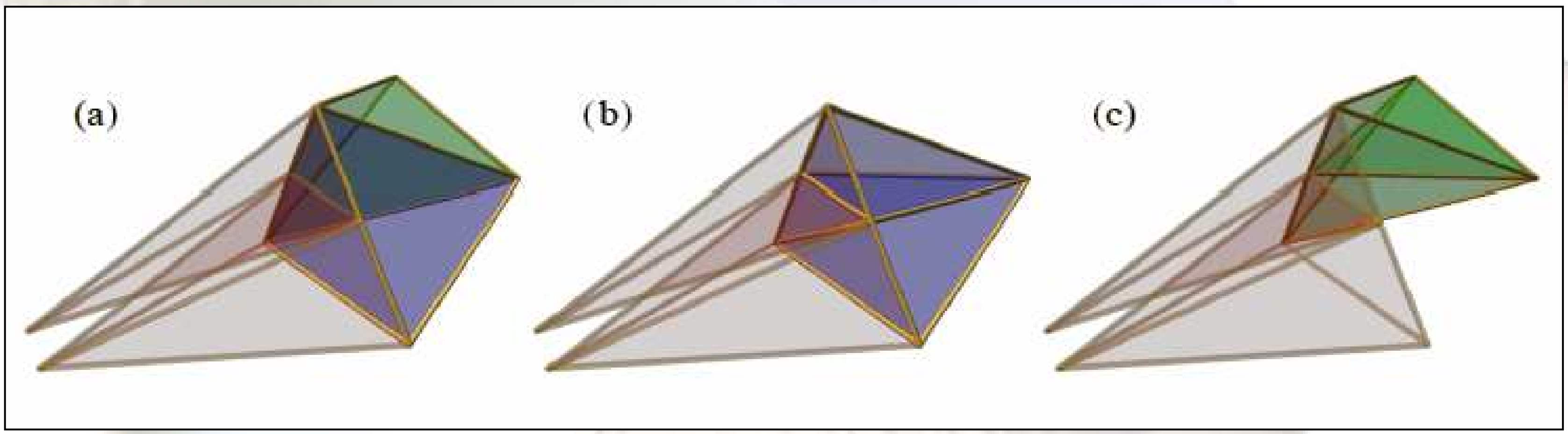
Finite Element Modeling



[http://titan.colorado.edu/courses.d/IFEM.d]

Finite Element Refinement





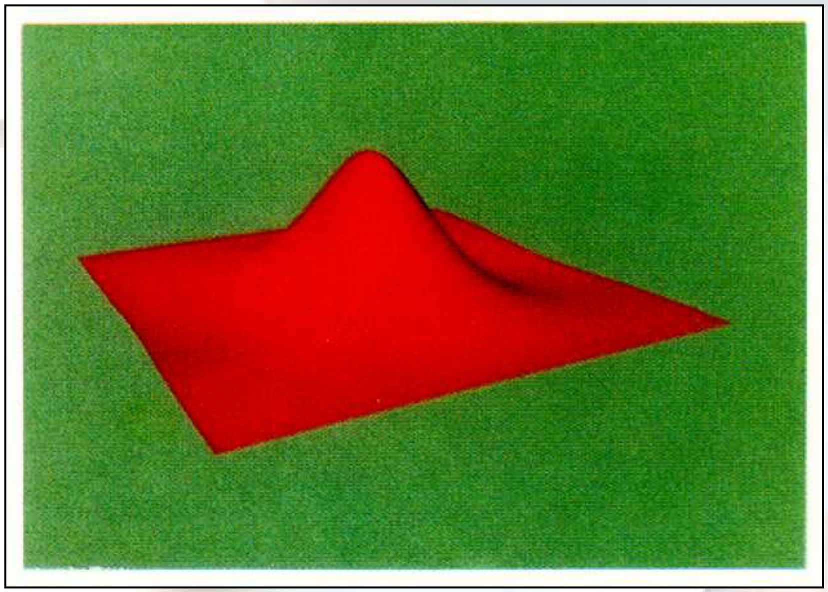
[O'Brien and Hodgins, SIGGRAPH 99]

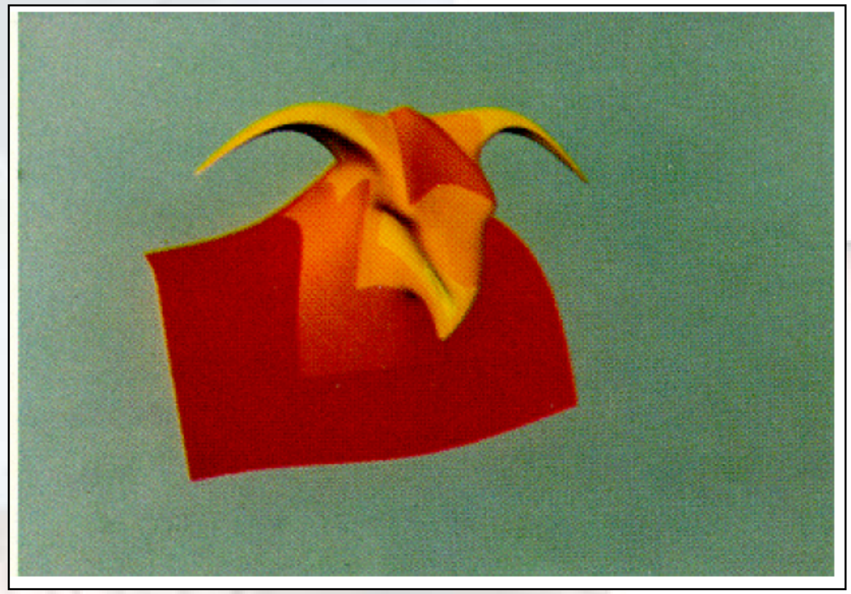
Previous Mesh Refinement Algorithms

- Split elements in **isolation**
  - Leads to **incompatibility**
- Element type **dependant**
- Lack of a **general approach**
- Implementation **complexity**

Previous Mesh Refinement Algorithms

- Usage of **hierarchical splines** in an FE solver





[Forsey and Bartels, SIGGRAPH 88]

C.H.A.R.M.S.

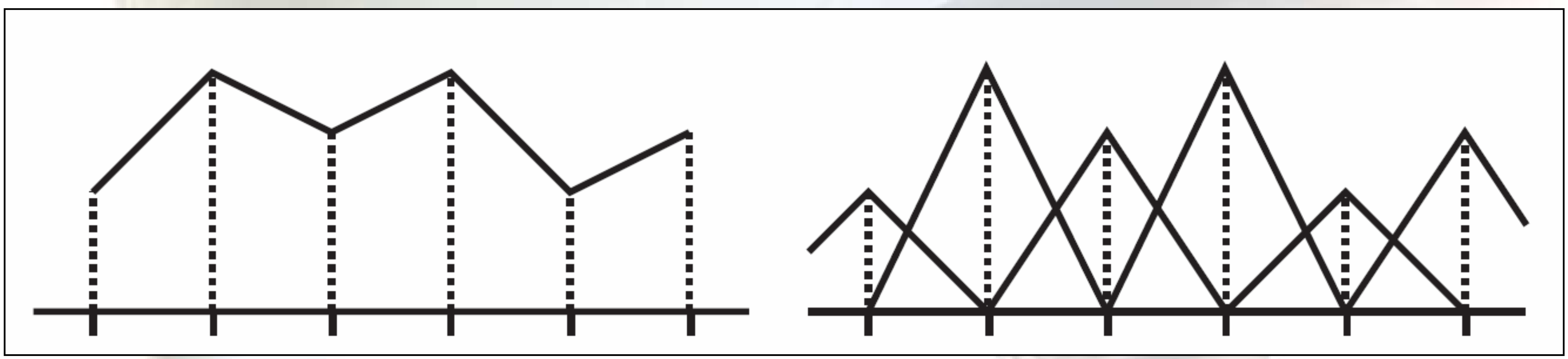
- **Conforming, Hierarchical, Adaptive Refinement Methods**
- Basic principle: **Refine basis functions**, not elements
- **Independent** of:
  - **Domain dimension** (2D and 3D)
  - **Element type** (triangle, quad, tetrahedron, etc)
  - **Basis function order**
- **Simple algorithms** for refinement and un-refinement

Outline

- Background
- ***Motivation (Element vs. Basis Refinement)***
- Implementation
  - Definitions
  - Data Structures
  - Algorithms
- Example Applications
- Conclusion

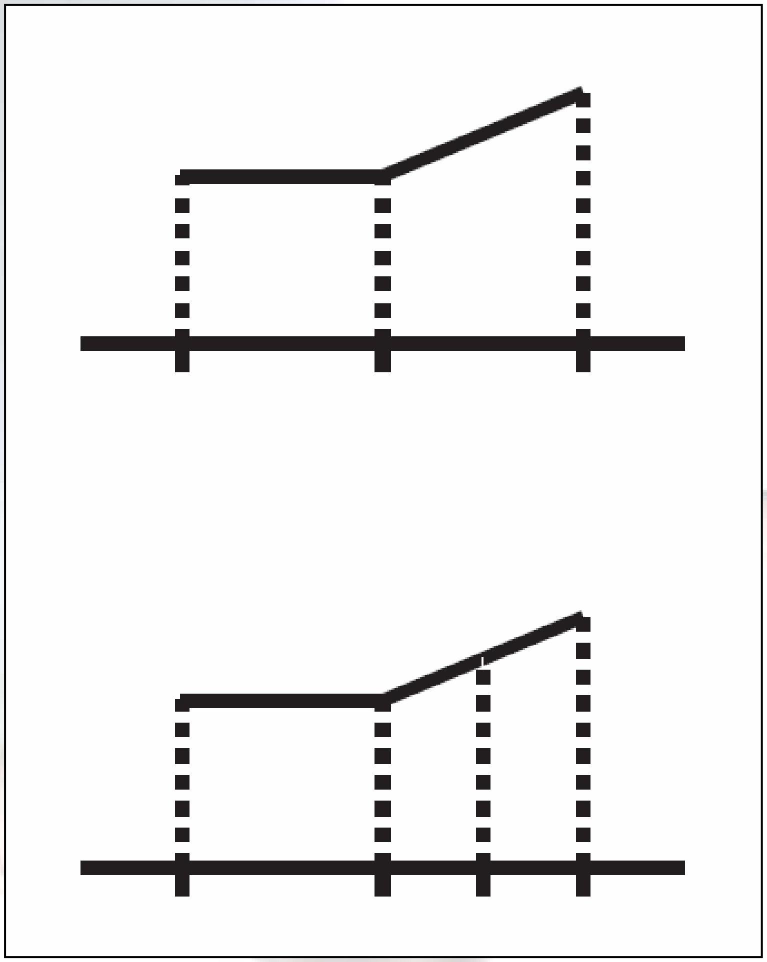
Element vs. Basis Refinement

- **Piecewise linear** approximation in **1D**:
  - **Finite Element:** Linear interpolation between endpoints
  - **Basis Function:** Linear combination of linear B-spline functions



Element Refinement

- **Refinement:** Element bisection
- **Un-refinement:** Merge a pair of elements

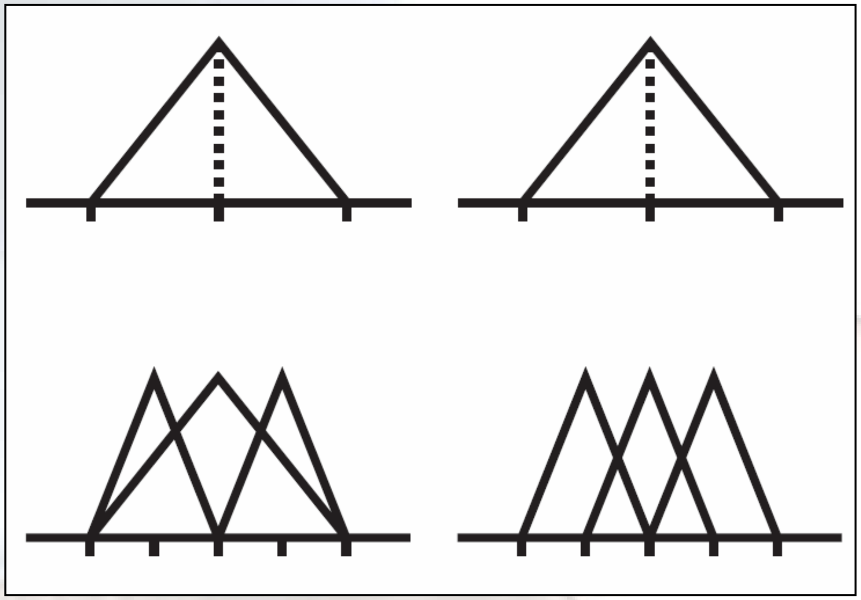


Basis Refinement

- **Refinement:** Add finer basis functions to reduce error
- **Un-refinement:** Remove the introduced functions

Basis Refinement

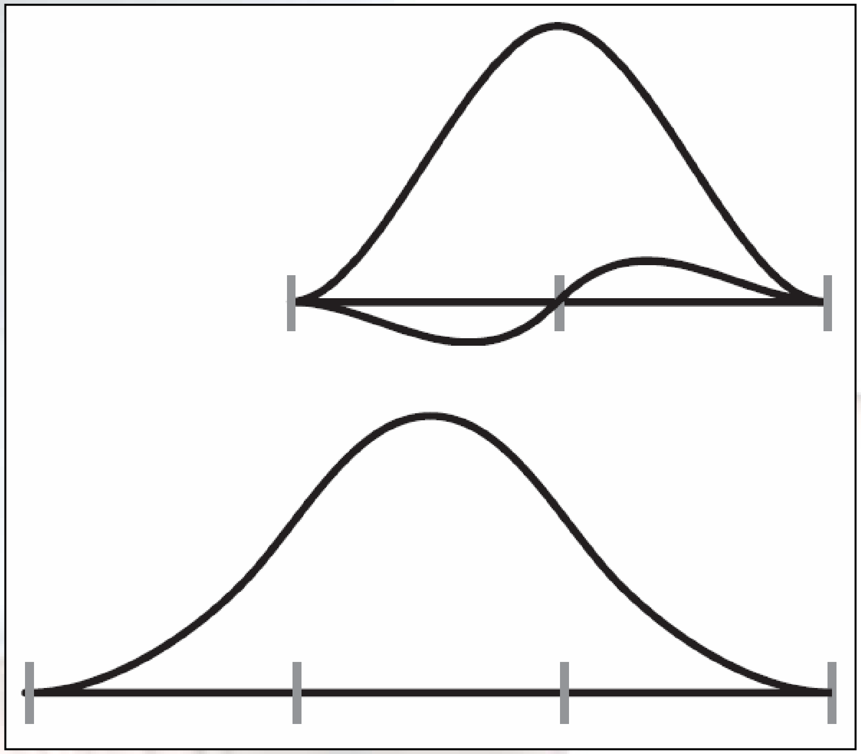
- **Hierarchical:** Add finer basis function in the middle of an element
- **Quasi-hierarchical:** Replace a basis function by three finer ones



Element vs. Basis Refinement

Higher order approximation in **1D**:

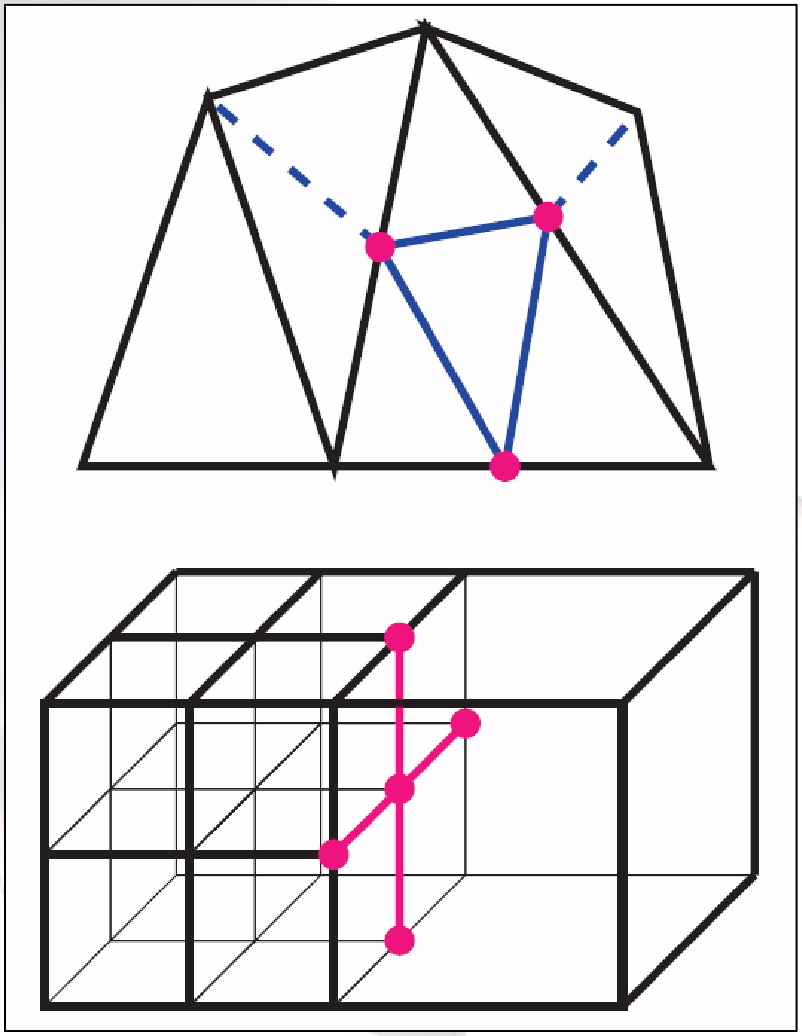
- **Finite Element:**
  - **Hermite cubic splines:** Can work on elements in isolation but **increases the DOFs** of our solution space
  - **Quadratic B-splines:** Cannot work on elements in isolation because **basis function overlaps** more than one element
- **Basis Function:**
  - Both can be refined



Element vs. Basis Refinement

Piecewise linear approximation in **2D**:

- **Finite Element:**
  - Quadrisect big triangles
  - Leads to T-vertices
  - Fix by adding conforming edges
- **Basis Function:**
  - Quadrisect globally
  - Get nodal basis functions



Element vs. Basis Refinement

- **Element refinement** becomes **impossible** as the number of dimensions or approximation order is **increased**
- **Basis refinement** applies to any **refinable function space** (**Refinable functions: coarser basis** can be represented as a **linear combination** of **finer basis functions**)

Outline

- Background
- Motivation (Element vs. Basis Refinement)
- ***Implementation***
  - ***Definitions***
  - ***Data Structures***
  - ***Algorithms***
- Example Applications
- Conclusion

One Basic Framework

IntegratePDE

```
While t < t_end
    predict: measure error and construct sets B+ and B-
    adapt:
        B := B ∪ B+ \ B-
        maintain basis: remove redundant functions from B
    solve: R_t(u_t) = 0
    t := t + Δt
```

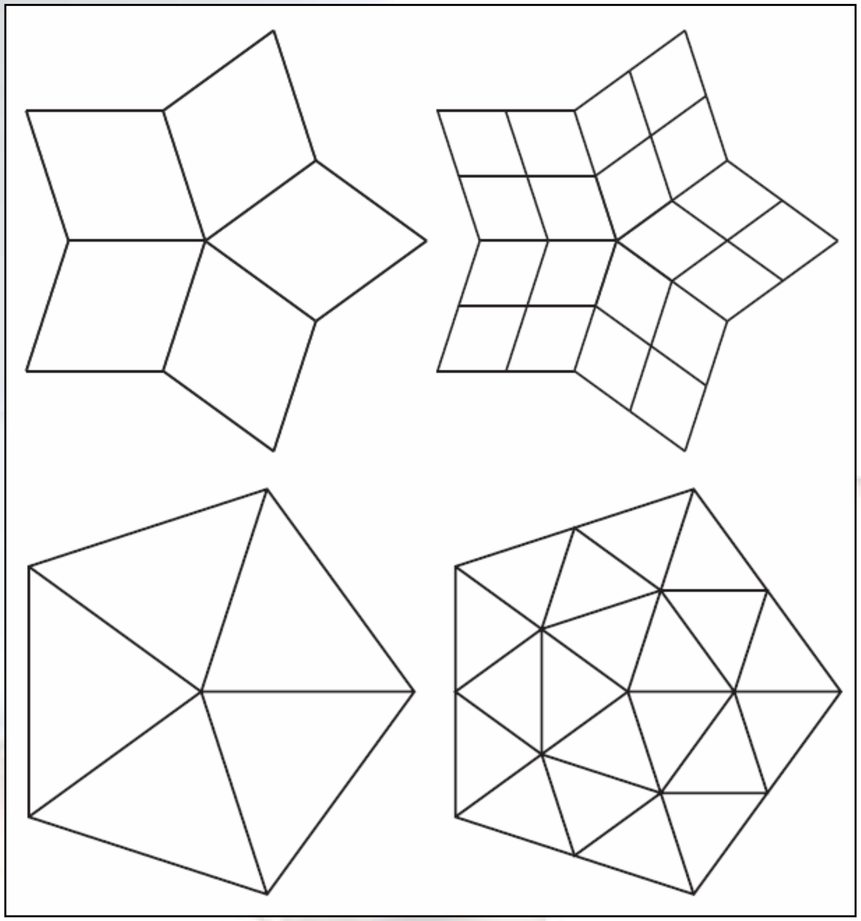
While $t < t_{end}$
predict: measure error and construct sets $\mathcal{B}^+$ and $\mathcal{B}^-$
adapt:
$\mathcal{B} := \mathcal{B} \cup \mathcal{B}^+ \backslash \mathcal{B}^-$
maintain basis: remove redundant functions from $\mathcal{B}$
solve: $\mathbf{R}_t(\mathbf{u}_t) = \mathbf{0}$
$t := t + \Delta t$

Definitions

- **Topological entities** of a mesh:
  - **Vertices**, V = { v_i }
  - **Edges**, E = { e_j }
  - **Faces**, F = { f_k }
  - **Cells**, C = { c_l }
- **Mesh:** triangle, quad, tetrahedra and hexahedra
- **Element:** faces or cells

Definitions

- **Coefficients** used to associate the mesh with **basis** functions
- **Coefficients** may live at any of the **topological entities**, usually at the vertices
- **Topological refinement operator:** Splits topological entities to **refine a mesh**



Definitions

- **Coefficient refinement operator:**
  - – Computes coefficients for finer mesh
  - – Based on coefficients from coarser mesh
  - – Linear
  - – Finitely supported
- **Subdivision scheme:** pairing of topological and coefficient refinement operators

Definitions

- **Even coefficients:** live on vertices the finer mesh inherits from the coarser mesh
- **Odd coefficients:** live on newly created vertices

Definitions

- **Refinement relation:** a basis function from a coarser level can be written as a **linear combination** of basis functions from the next finer level

$$\phi_i^{(j)}(x) = \sum_k a_{ik}^{(j+1)} \phi_k^{(j+1)}(x)$$

Definitions

- **Children** of a basis function:

$$\mathcal{C}(\phi_i^{(j)}) = \{\phi_k^{(j+1)} | a_{ik}^{(j+1)} \neq 0\}$$

- **Parents** of a basis function:

$$\mathcal{C}^\star(\phi_i^{(j)}) = \{\phi_k^{(j-1)} | \phi_i^{(j)} \in \mathcal{C}(\phi_k^{(j-1)})\}$$

Definitions

- **Natural support set** [$S(\varphi_i^{(j)})$]:
  - Minimal set of elements at level j that contain the parametric support of the basis function
- **Adjoint** [$S^*(\varepsilon_l^j)$]:
  - Set of basis whose natural support contain ε_l^j

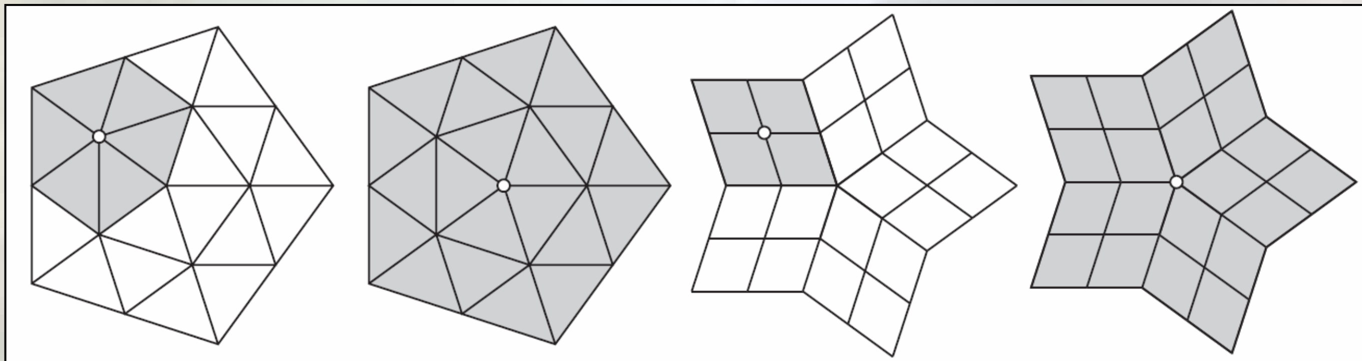


Figure 7: *Examples of natural support sets. Left to right: linear splines, Loop basis, bilinear spline, and Catmull-Clark basis.*

Definitions

- **Descendants of an element** [$D(\varepsilon_i^j)$]:
  - **Elements at levels > *j*** which have non-zero intersection with the given element
- **Adjoint** [$D^*(\varepsilon_i^j)$]:
  - **Ancestor relation**

Data Structures

- Set of **active basis functions:** B
- Set of **active integration elements:** ε
- Set of active functions that overlap an element:
  - **Same level**: $B^s(\varepsilon)$
  - **Ancestor levels**: $B^a(\varepsilon)$

Data Structures

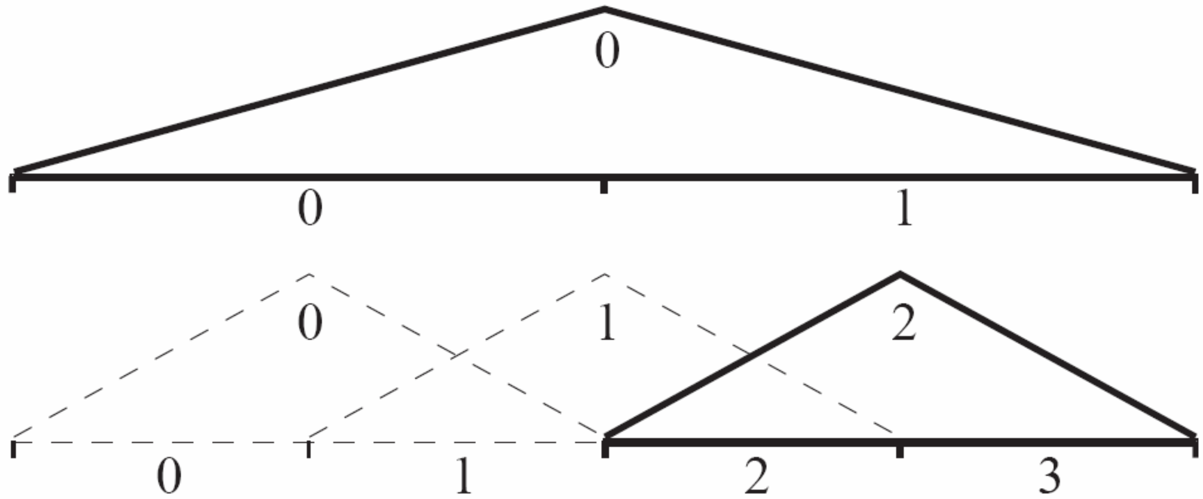


Figure 8: *Illustrative example of the data structures. Shown in bold are a pair of active basis functions on mesh levels 0 and 1. The associated data structures are:* $\mathcal{B} = \{\phi_0^{(0)}, \phi_2^{(1)}\}$, $\mathcal{E} = \{\varepsilon_0^0, \varepsilon_2^1, \varepsilon_3^1\}$, $\mathcal{S}(\phi_0^{(0)}) = \{\varepsilon_0^0, \varepsilon_1^0\}$, $\mathcal{S}(\phi_2^{(1)}) = \{\varepsilon_2^1, \varepsilon_3^1\}$, $B^s(\varepsilon_0^0) = \{\phi_0^{(0)}\}$, $B^a(\varepsilon_0^0) = \emptyset$, $B^s(\varepsilon_2^1) = \{\phi_2^{(1)}\}$, $B^a(\varepsilon_2^1) = \{\phi_0^{(0)}\}$, $B^s(\varepsilon_3^1) = \{\phi_2^{(1)}\}$, $B^a(\varepsilon_3^1) = \{\phi_0^{(0)}\}$.

Algorithms

- Compute the **stiffness matrix** (for the FE solver) by **iterating over active elements** and **computing local interactions**

ComputeStiffness($\mathcal{E}$)
  ForEach $\varepsilon \in \mathcal{E}$ **do**
    ForEach $\phi \in B^s(\varepsilon)$ **do**
      $k_{\phi\phi} += $ **Integrate**(ϕ,ϕ,ε)
      ForEach $\psi \in B^s(\varepsilon) \setminus \{\phi\}$ **do**
        $k_{\phi\psi} += $ **Integrate**(ϕ,ψ,ε)
        $k_{\psi\phi} += $ **Integrate**(ψ,ϕ,ε)
      ForEach $\psi \in B^a(\varepsilon)$ **do**
        $k_{\phi\psi} += $ **Integrate**(ϕ,ψ,ε)
        $k_{\psi\phi} += $ **Integrate**(ψ,ϕ,ε)

Algorithms

- Over the solution process, basis functions are **activated** and **deactivated**

Activate(ϕ)

$\mathcal{B} \cup= \{\phi\}$
ForEach $\varepsilon \in \mathcal{S}(\phi)$ **do**
  $B^s(\varepsilon) \cup= \{\phi\}$
  `// upon activation initialize ancestor list`
  If $\varepsilon \notin \mathcal{E}$ **then** $B^a(\varepsilon) \cup=$ **Ancestor**(ε) ; $\mathcal{E} \cup= \{\varepsilon\}$ **fI**
  `// add to ancestor lists of active descendants`
  ForEach $\gamma \in (\mathcal{D}(\varepsilon) \cap \mathcal{E})$ **do** $B^a(\gamma) \cup= \{\phi\}$

Ancestor(ε)

$\rho := \emptyset$
ForEach $\gamma \in \mathcal{D}^\star(\varepsilon) \cap \mathcal{E}$ **do**
  $\rho \cup= B^s(\gamma) \cup B^a(\gamma)$
return ρ

Algorithms

Deactivate(ϕ)

$\mathcal{B} \setminus= \{\phi\}$

ForEach $\varepsilon \in \mathcal{S}(\phi)$ **do**

  $B^s(\varepsilon) \setminus= \{\phi\}$

  // deactivate element?

  If $B^s(\varepsilon) = \emptyset$ **then** $\mathcal{E} \setminus= \{\varepsilon\}$

  // update ancestor lists of active descendants

  ForEach $\gamma \in \mathcal{D}(\varepsilon) \cap \mathcal{E}$ **do** $B^a(\gamma) \setminus= \{\phi\}$

Algorithms

- Assuming we have an appropriate **error estimator** we can have **adaptive solver strategies** on top of **activate**

HierarchicalRefine(ϕ)
  ForEach $\psi \in \mathcal{C}(\phi)$ **do**
    If $\psi \notin \mathcal{B} \wedge \mathrm{Odd}(\psi)$ **then Activate**(ψ) ; $u_\psi := 0$ **fI**

HierarchicalUnrefine(ϕ)
  ForEach $\psi \in \mathcal{C}(\phi)$ **do**
    If $\psi \notin \mathcal{B} \wedge \mathrm{Odd}(\psi)$ **then Deactivate**(ψ)

QuasiHierarchicalRefine(ϕ)
  Deactivate(ϕ)
  ForEach $\psi \in \mathcal{C}(\phi)$ **do**
    If $\psi \notin \mathcal{B}$ **then Activate**(ψ) ; $u_\psi := 0$ **fI**
    $u_\psi += a_{\phi,\psi} u_\phi$

QuasiHierarchicalUnrefine(ϕ)
  Activate(ϕ) ; initialize u_ϕ
  ForEach $\psi \in \mathcal{C}(\phi)$ **do**
    If $\psi \notin \mathcal{B}$ **then Deactivate**(ψ)

Algorithms

- Things to add to complete a simulator:
  - **Standard solvers** for the resulting algebraic systems
  - Appropriate **error estimators**
  - A **numerical integration** routine

Outline

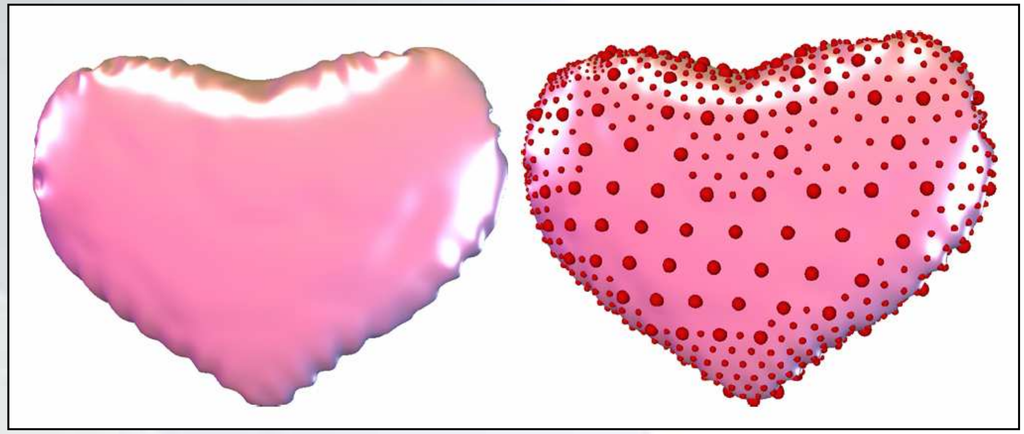
- Background
- Motivation (Element vs. Basis Refinement)
- Implementation
  - Definitions
  - Data Structures
  - Algorithms
- ***Example Applications***
- Conclusion

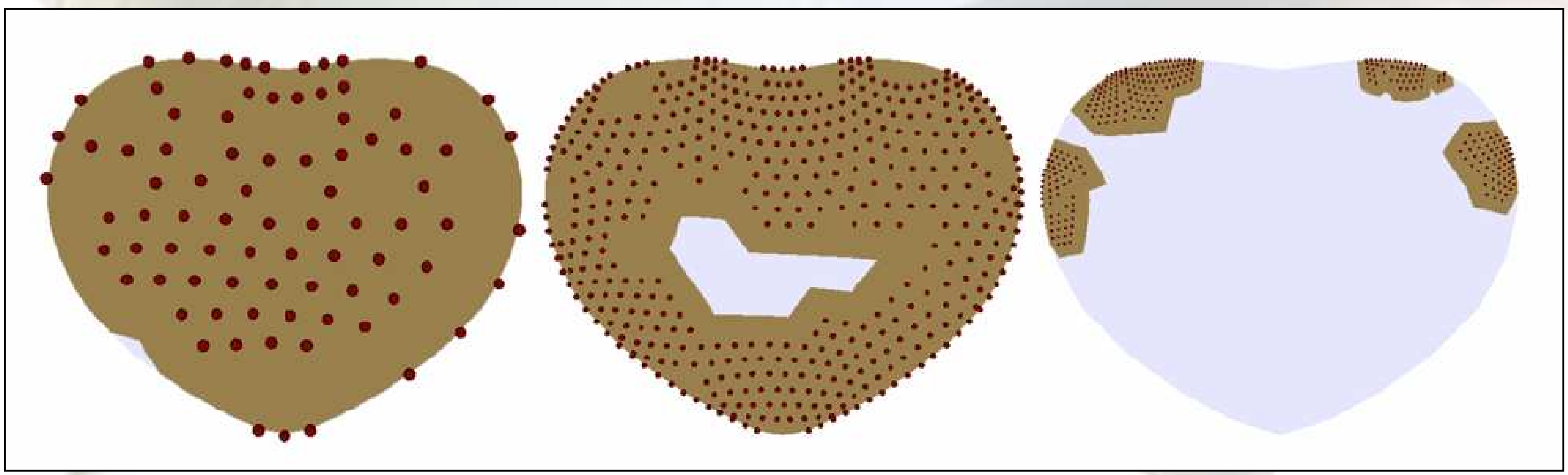
Example Applications

- Application domains:
  - **Animation**
  - **Modeling**
  - **Engineering**
  - **Medical simulation/visualization**

Example Application: Thin-Shells

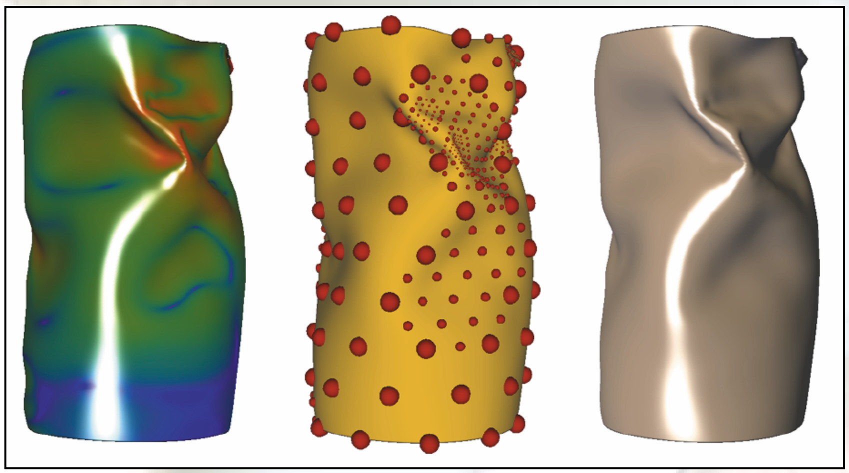
- Inflating balloon
- Poking balloon

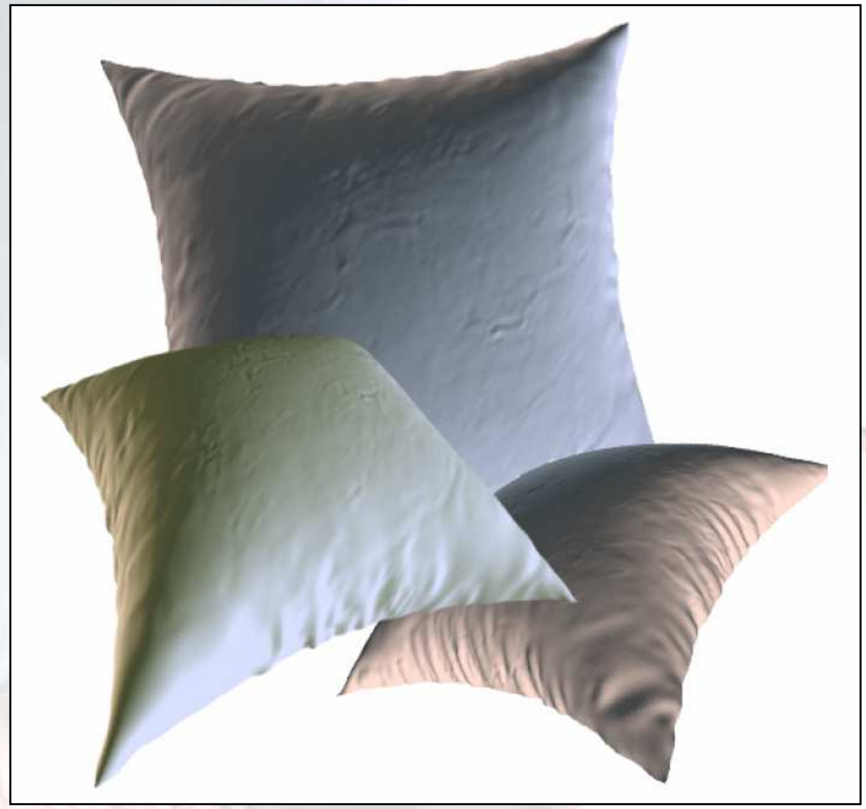




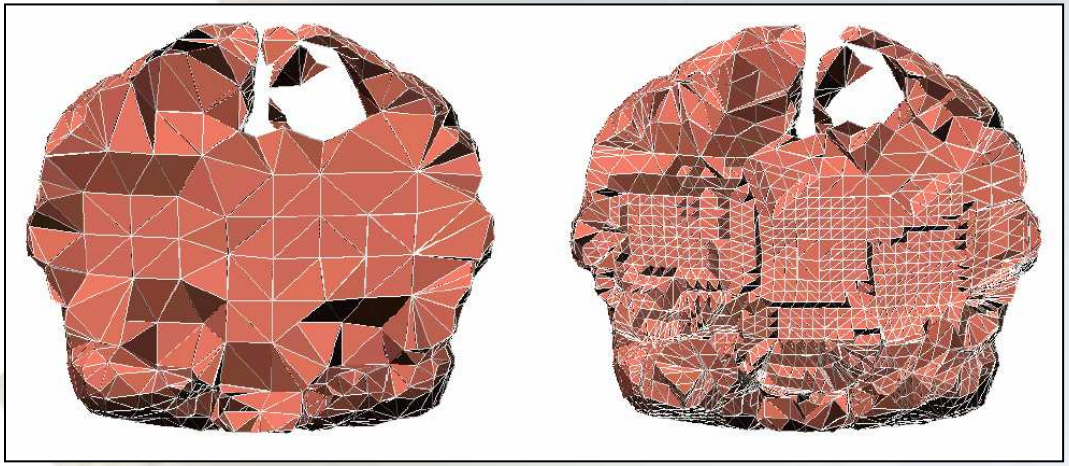
Example Application: Thin-Shells

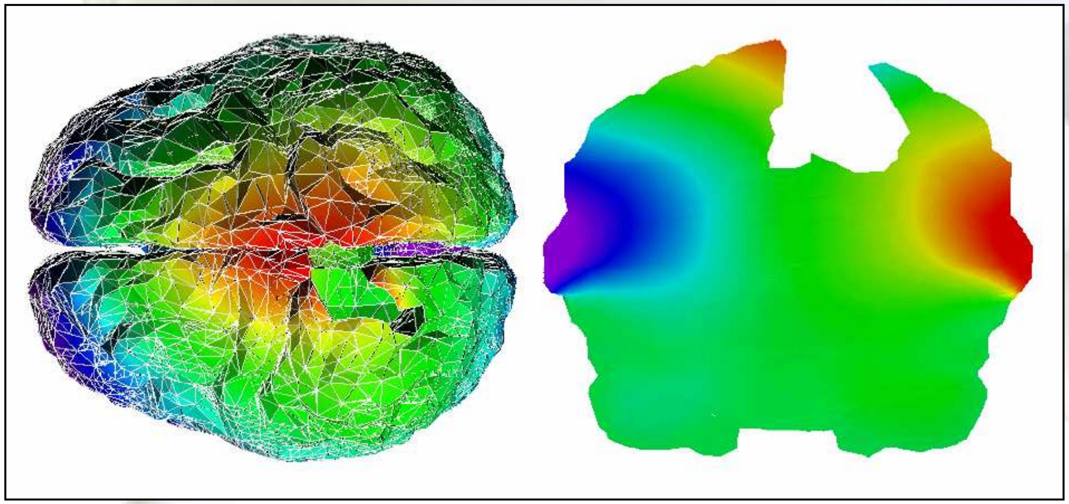
- Pillows
- Crushing cylinder

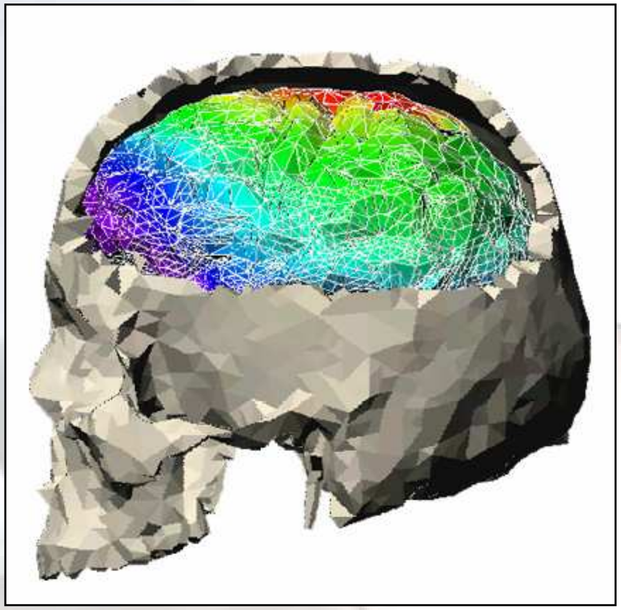




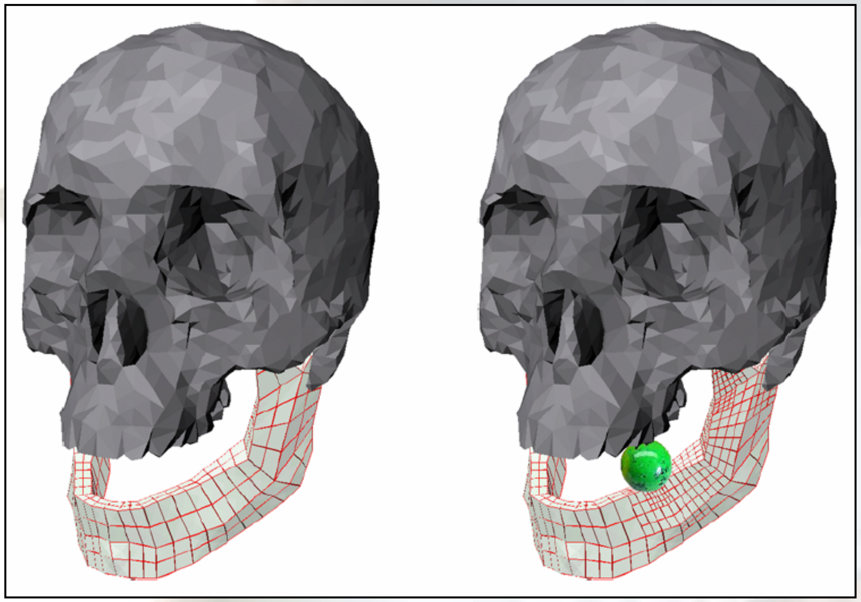
Example Application: Surgery Aid

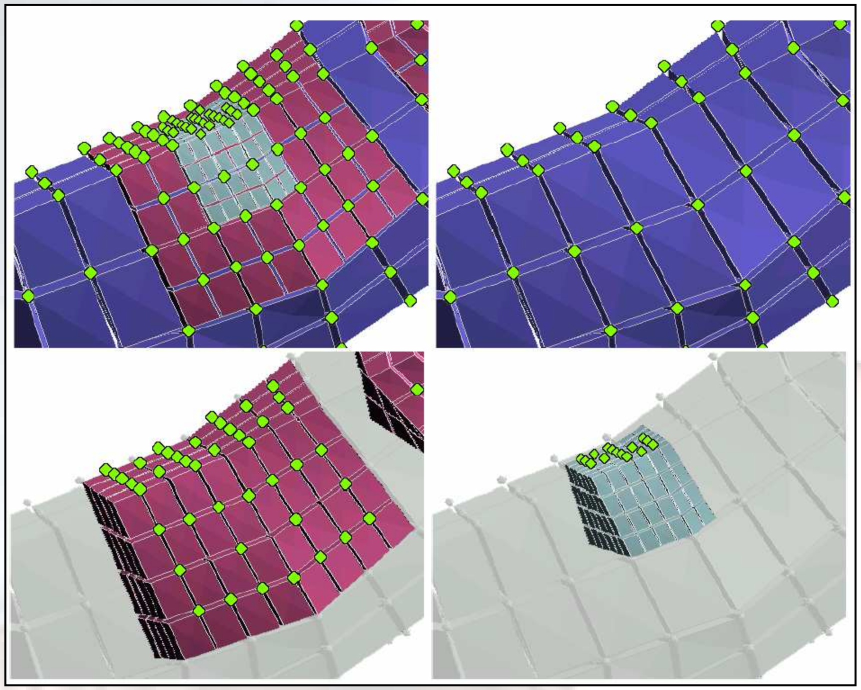






Example Application: Human Jaw





Conclusion

- **Simple framework** for constructing **adaptive solvers** for PDEs
- **Applications** in CG, engineering and bio-medical computing
- Uses refinability of **basis functions**
- Easy **implementation**
- No element-wise **compatibility** issues
- Avoids popping in **animation** during refinement