Global and local error computation and estimation

Jean-Paul Pelteret (jean-paul.pelteret@fau.de)
Luca Heltai (luca.heltai@sissa.it)

19 March 2018

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Elements

There is a zoo of elements for different purposes:

- Continuous Lagrange
- Discontinuous Lagrange
- Raviart-Thomas
- Nedelec
- Rannacher-Turek
- Brezzi-Douglas-Marini (BDM)
- Brezzi-Douglas-Duran-Marini (BDDM)
- Hermite (Argyris)
- Crouzeix-Raviart
- Arnold-Falk-Winther
- Arnold-Boffi-Falk (ABF)

  ...
- Hybridized elements
- Penalized discontinuous elements

Elements

There is a zoo of elements for different purposes:

- Continuous Lagrange — FE_Q
- Discontinuous Lagrange — FE_DGQ, FE_DGP
- Raviart-Thomas — FE_RaviartThomas
- Nedelec — FE_Nedelec
- Rannacher-Turek
- Brezzi-Douglas-Marini (BDM) — FE_BDM
- Brezzi-Douglas-Duran-Marini (BDDM)
- Hermite (Argyris)
- Crouzeix-Raviart
- Arnold-Falk-Winther
- Arnold-Boffi-Falk (ABF) — FE_ABF

  ...
- Hybridized elements — FE_FaceQ/TraceQ
- Penalized discontinuous elements

Scalar problems

For scalar problems like the Laplace equation:

- Q_p elements are generally the right choice
- Higher *p* yield higher convergence order for elliptic problems:

$$\|u-u_h\|_{H^1} \leq Ch^p|u|_{H^{p+1}} \qquad \|u-u_h\|_{L_2} \leq Ch^{p+1}|u|_{H^{p+1}}$$

- Number of degrees of freedom grows as:

$$N \simeq \frac{|\Omega|}{(h/p)^d} = p^d\frac{|\Omega|}{h^d} \qquad \rightarrow \qquad h \simeq p\left(\frac{|\Omega|}{N}\right)^{1/d}$$

- Error as function of *N*:

$$\|u-u_h\|_{H^1} \simeq p^p N^{-p/d} \qquad \|u-u_h\|_{L_2} \simeq p^{p+1} N^{-(p+1)/d}$$

Consequence: This suggests high order elements!

Scalar problems

For scalar problems like the Laplace equation:

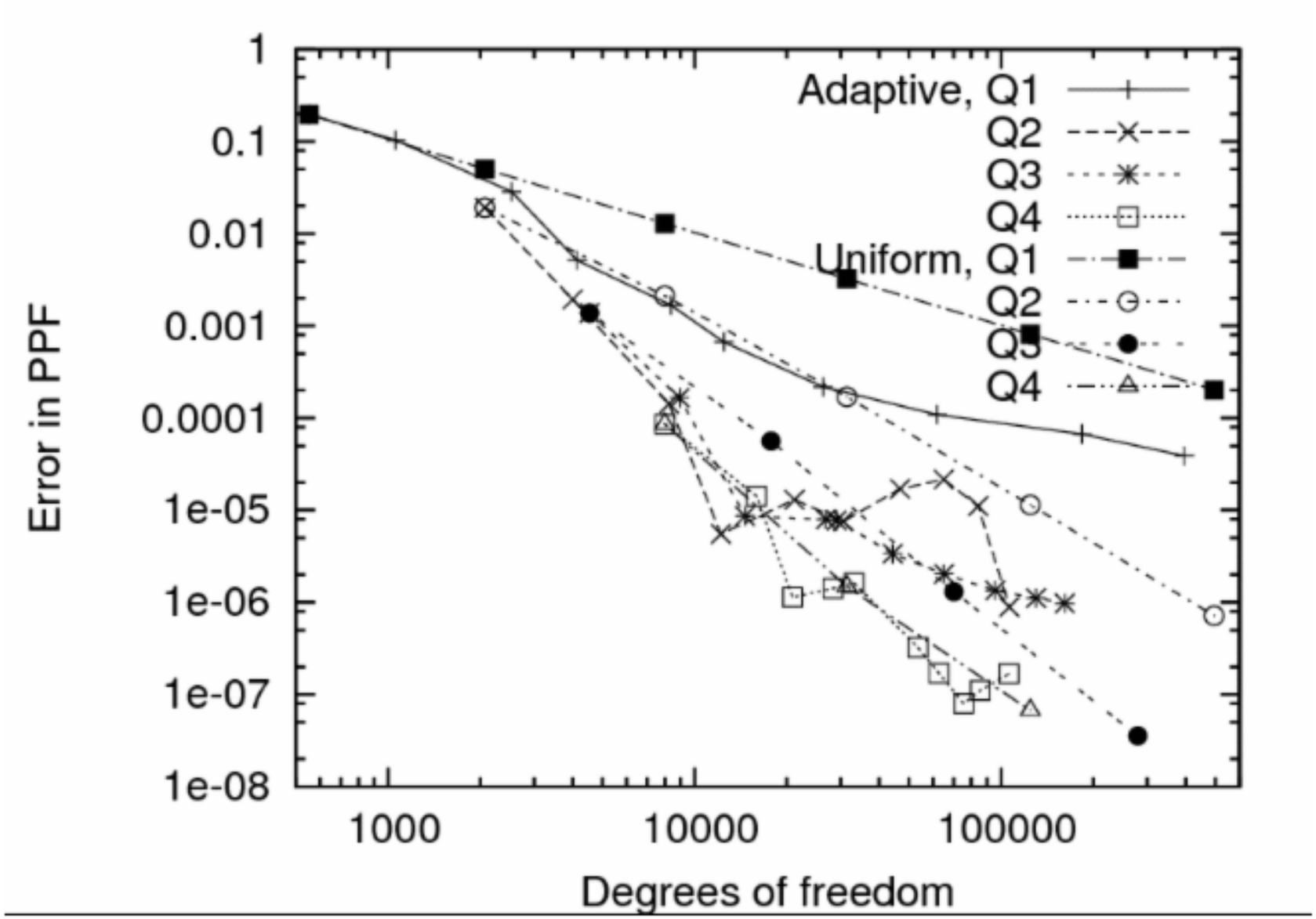
- Q_p elements are generally the right choice
- Better convergence only if *u* smooth: $\|u-u_h\|_{H^1} \leq Ch^p|u|_{H^{p+1}}$
- Higher *p* also requires more work:
  - more computations to assemble matrix: $O(p^d)$
  - more entries per row in the matrix: $O(p^d)$
  - good preconditioners are difficult to construct for high *p*

Consequence: This suggests low order elements!

Together: It is a trade-off!

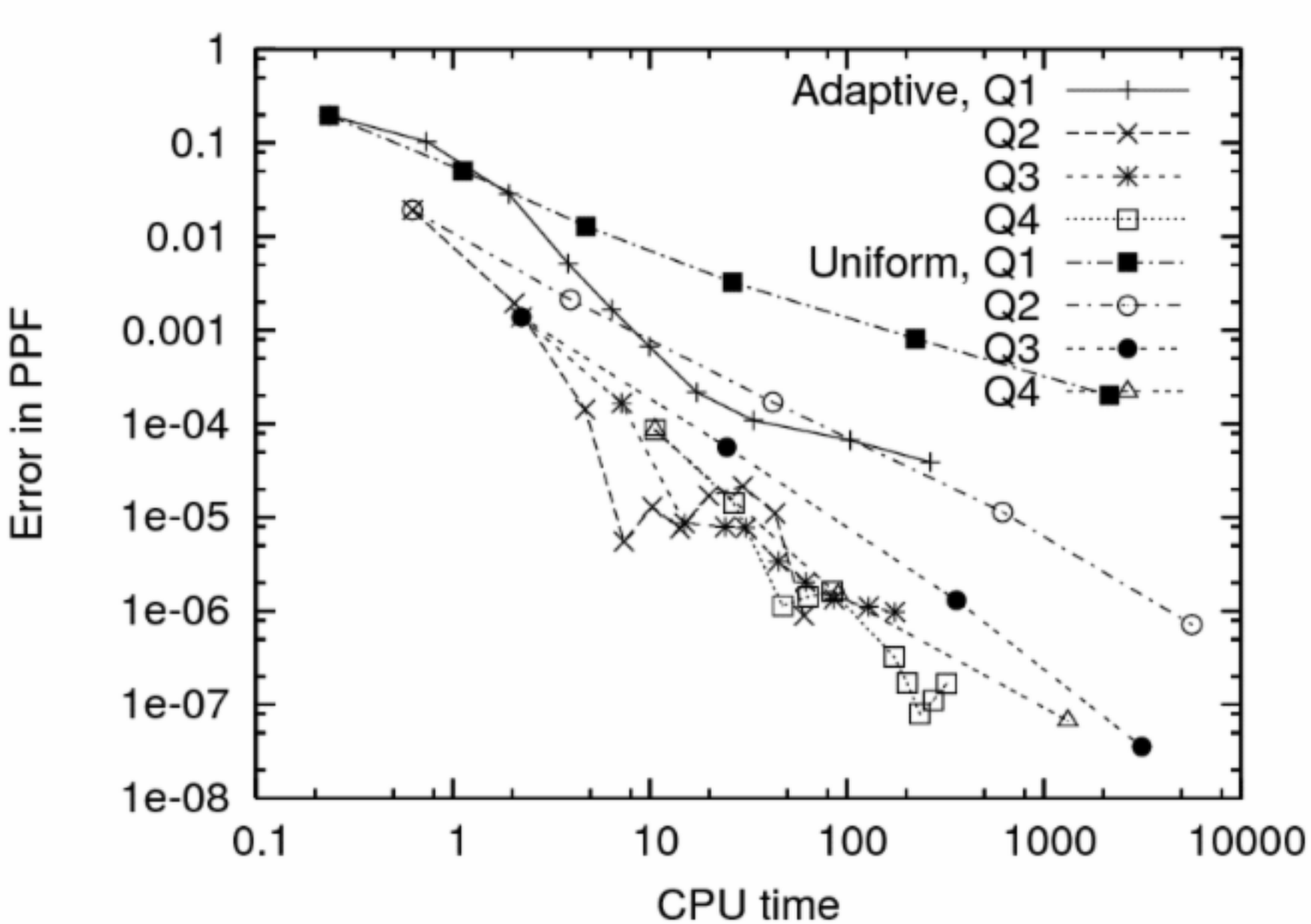
Practical experience

Prototypical 2d example from Wang, Bangerth, Ragusa (2007, Progress in Nuclear Energy):



Practical experience

Prototypical 2d example from Wang, Bangerth, Ragusa (2007, Progress in Nuclear Energy):



Practical experience

Prototypical 2d example from Wang, Bangerth, Ragusa (2007, Progress in Nuclear Energy):

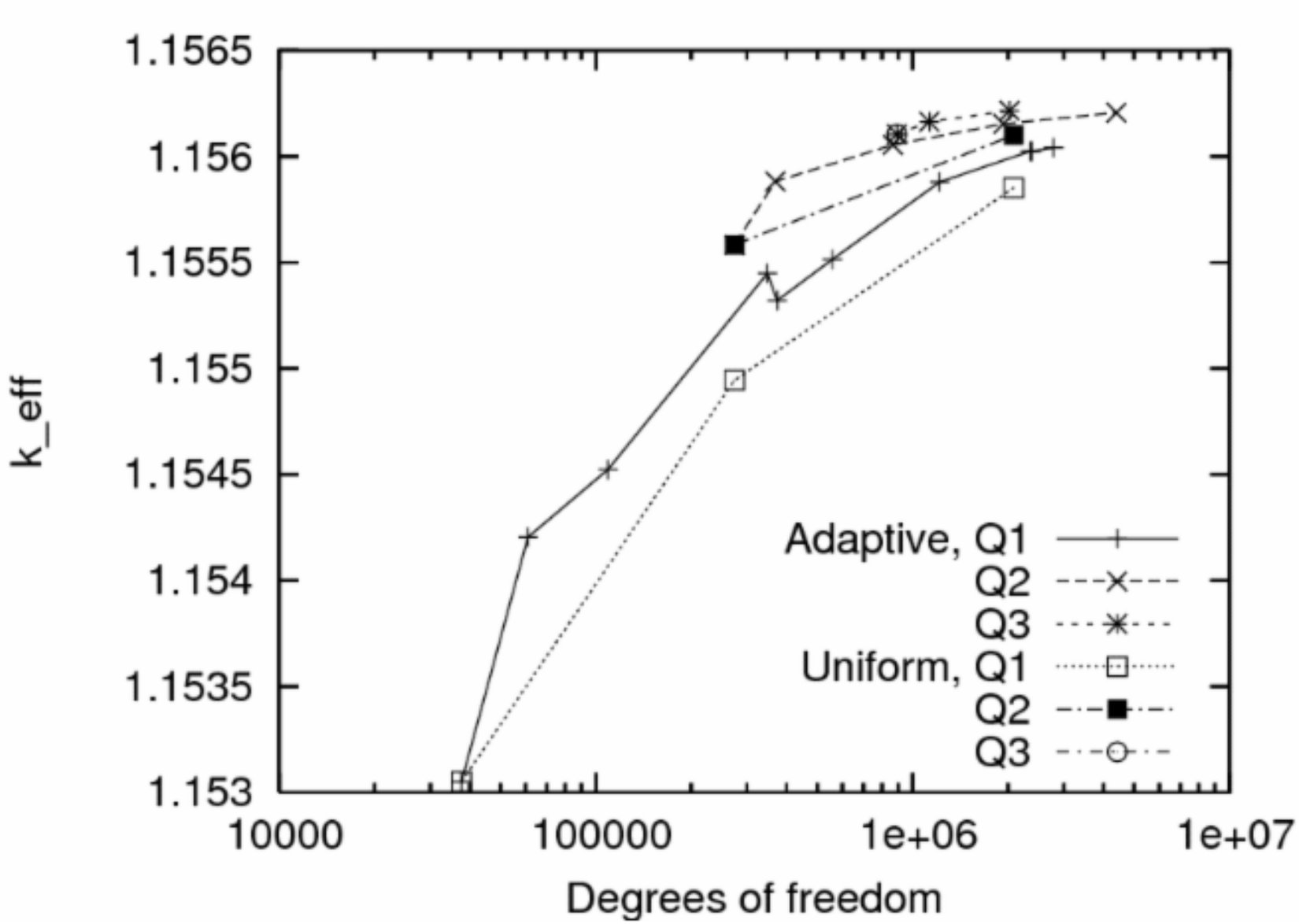
Conclusions:

- Higher *p* gives better error-per-dof
- Not so clear any more for error-per-CPU-second

- Sweat spot maybe around *p=3* or *p=4* in 2d

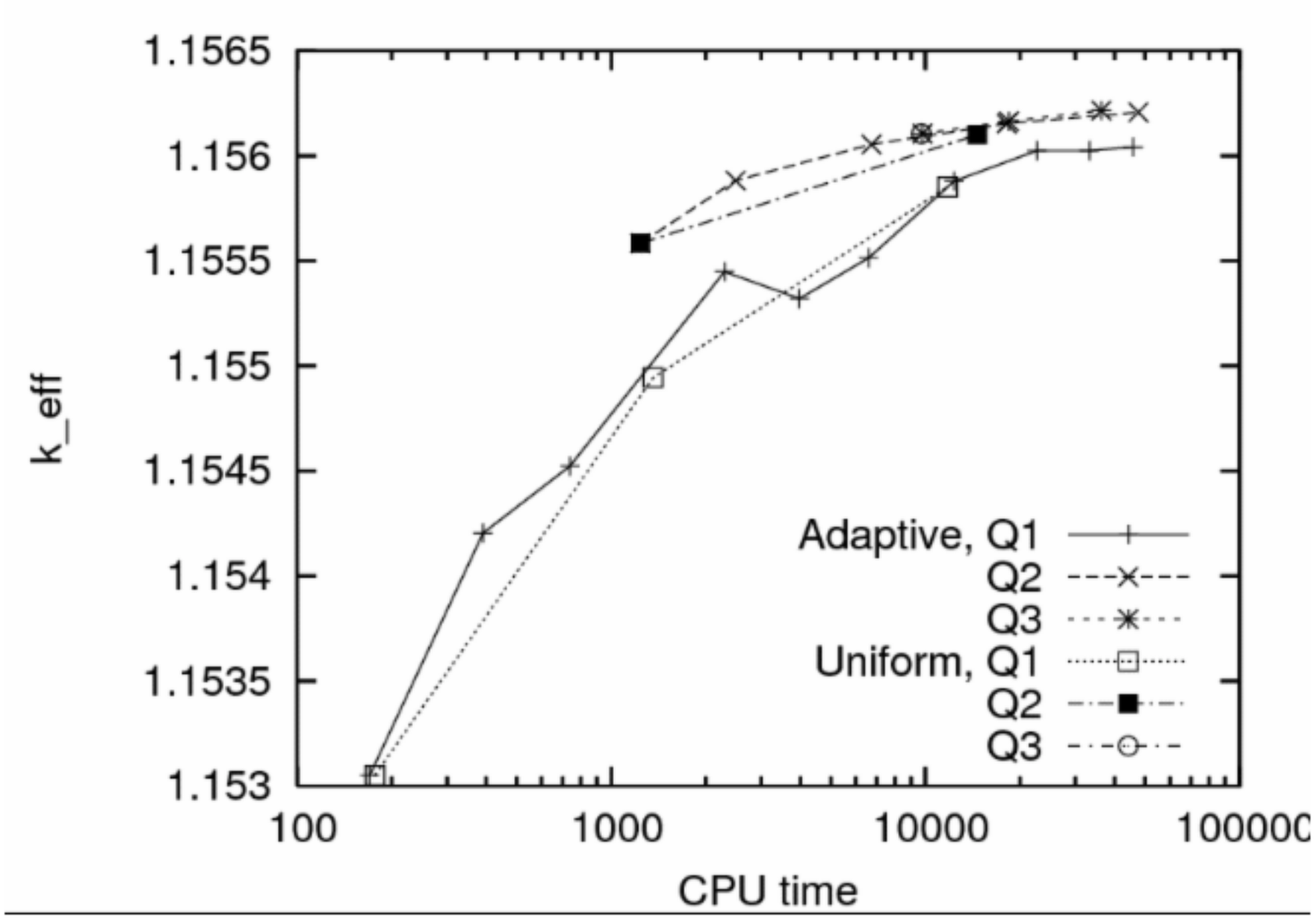
Practical experience

Prototypical 3d example from Wang, Bangerth, Ragusa (2007, Progress in Nuclear Energy):



Practical experience

Prototypical 3d example from Wang, Bangerth, Ragusa (2007, Progress in Nuclear Energy):



Practical experience

Prototypical 3d example from Wang, Bangerth, Ragusa (2007, Progress in Nuclear Energy):

Conclusions:

- Higher *p* gives better error-per-dof
- Not so clear any more for error-per-CPU-second

- Sweat spot maybe around *p=2* or *p=3* in 3d

Practical experience

Conclusions for scalar problems:

- There is a trade-off between faster convergence and more work

- A good compromise is:
  - Q3 or Q4 in 2d
  - Q2 or Q3 in 3d

How to measure the Error?

- Method of Manufactured Solutions
  - Take the “u” you want as a solution, plug in the equations, get the boundary conditions and the right hand side that force the given “u”
  - Integrate (with a fine quadrature formula) the difference between the exact solution and the computed one (VectorTools::integrate_difference)
  - Possibly integrate the difference between the gradients of the exact and computed solutions

Adaptive mesh refinement (AMR)

"Traditional" error estimates for Q1/P1 elements applied to the Laplace equation look like this:

$$\|e\|_{H^1} \leq C\, h_{\max} \|u\|_{H^2} \qquad \text{or equivalently:} \quad \|e\|_{H^1}^2 \leq C^2 h_{\max}^2 \|u\|_{H^2}^2$$

This implies that the error is dominated by the *largest* cell diameter and the (global) H^2 norm of the exact solution.

To reduce the error, this suggests:

- Global mesh refinement
- Nothing can be done if a solution has a large H^2 norm

Estimate the rate of convergence

- Once you have computed the error, how do we estimate if we get the correct *convergence ratio?*

$$\|u - u_h\|_1 \leq Ch^k |u|_{k+1}$$

$$\|u - u_h\|_0 \leq Ch^{k+1} |u|_{k+1}$$

Estimate the rate of convergence

- Compute two successive solutions, on half the size of the mesh (i.e., after one global refinement):

$$\|u - u_h\| \sim C(h)^p$$

$$\|u - u_{2h}\| \sim \tilde{C}(2h)^p$$

$$\frac{\|u - u_{2h}\|}{\|u - u_h\|} \sim 2^p$$

$$p \sim \log_2 \left(\frac{\|u - u_{2h}\|}{\|u - u_h\|} \right)$$

Adaptive mesh refinement (AMR)

However, a closer analysis shows that the error is really:

$$\|e\|_{H^1}^2 \leq C^2 \sum_K h_K^2 \|u\|_{H^2(K)}^2$$

In other words: To reduce the error, we *only* need to make the mesh fine where the local H^2 norm is large!

Adaptive mesh refinement (AMR)

Note: The optimal strategy to minimize the error while keeping the problem as small as possible is to *equilibrate* the local contributions

$$e_K = C h_K \|u\|_{H^2(K)}$$

That is, we want to choose

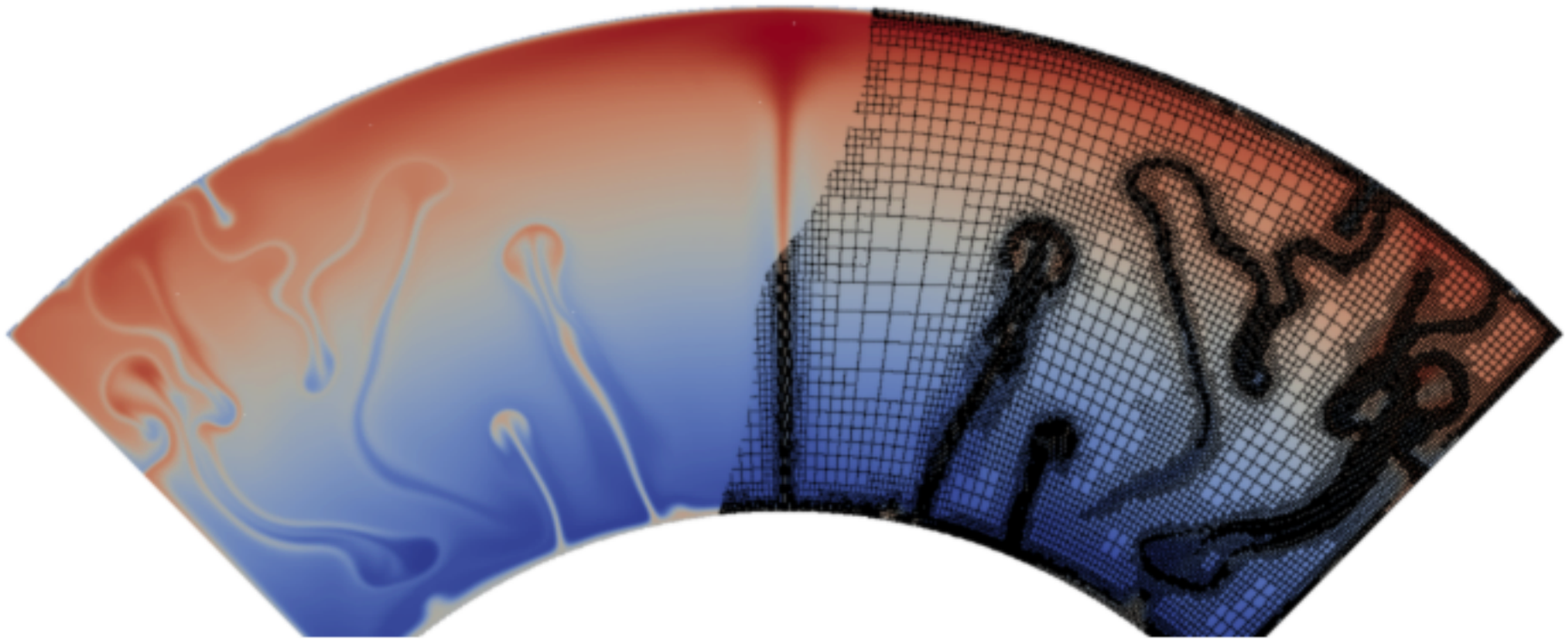
$$h_K \propto \frac{1}{\|u\|_{H^2(K)}}$$

In practice: Exact errors are unknown. Thus, use a local *indicator* of the error η_K and choose h_K so that

$$\sum_K \eta_K = \text{tol}$$

Adaptive mesh refinement (AMR)

Example:



Refine only where “something is happening” (i.e., locally the second derivative of the solution is large).

Adaptive mesh refinement (AMR)

Question: How can we create such meshes?

Answer 1: Except for special cases it is not possible to generate them right away because we do not know the exact solution.

Answer 2: But it can be done iteratively.

Adaptive mesh refinement (AMR)

Basic algorithm:

1. Start with a coarse mesh
2. Solve on this mesh
3. Compute error indicator for every cell based on numerical solution
4. If overall error is less than *tol* then STOP
5. Mark a fraction of the cells with largest error
6. Refine marked cells to get new mesh
7. Start over at 2.

Note: This is often referred to as the SOLVE-MARK-REFINE cycle.

Refining triangular meshes

Refining *triangular* meshes is relatively simple.

There are three widely used options:

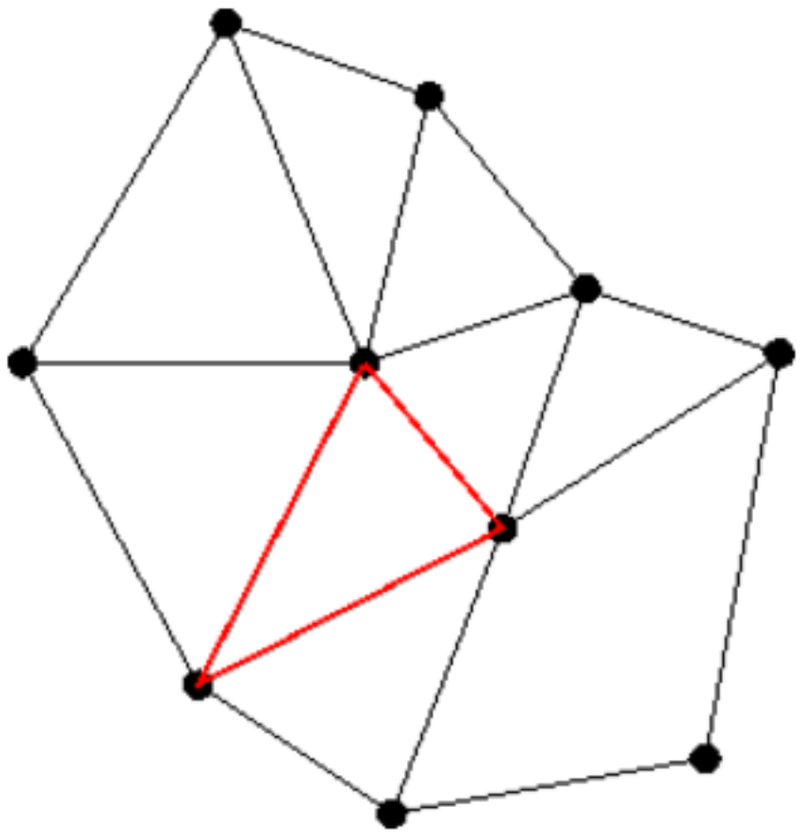
- *De novo* generation of a non-uniform mesh
- Longest edge refinement
- Red-green refinement

Note 1: Refining tetrahedra in 3d works in analogous ways.

Note 2: There are also other strategies.

Longest edge refinement

Consider this mesh and a marked cell:

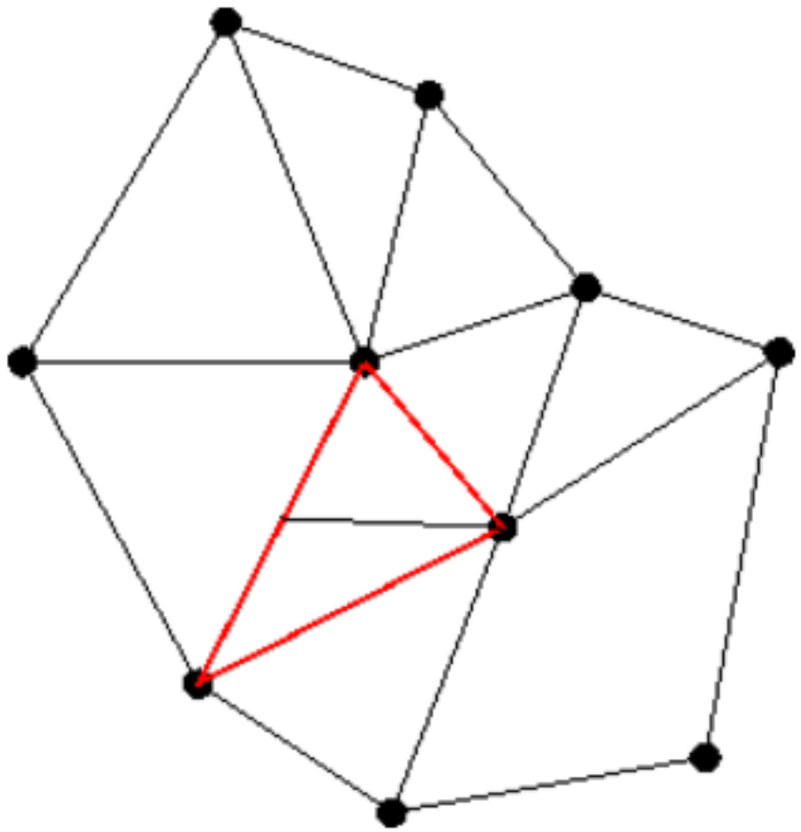


Algorithm:

- Add edge from midpoint of longest edge to oppos. vertex

Longest edge refinement

Consider this mesh and a marked cell:

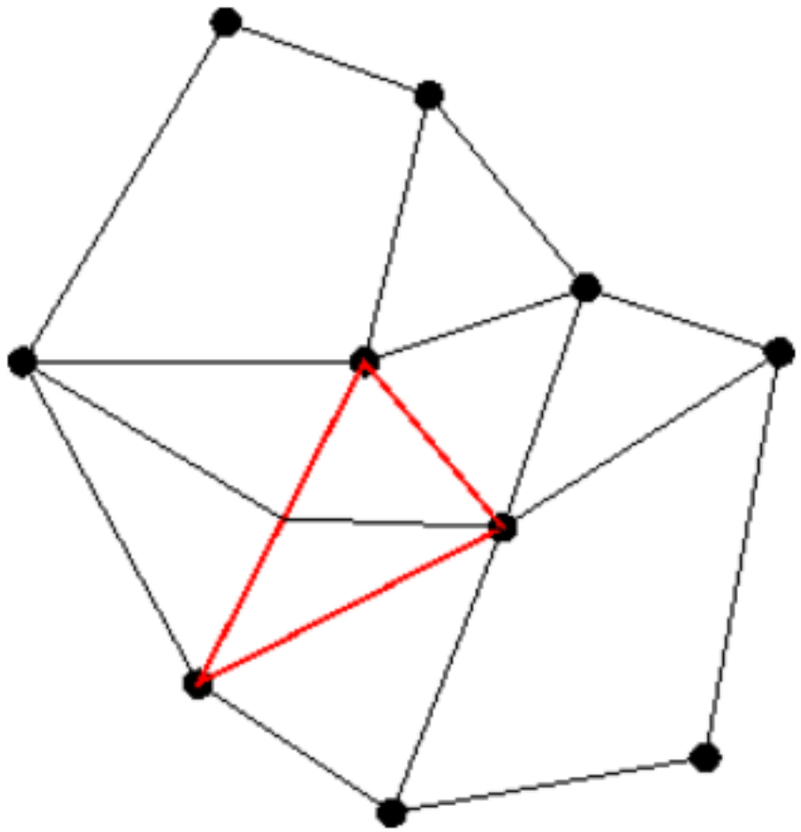


Algorithm:

- Add edge from midpoint of longest edge to oppos. vertex
- If this is also the longest edge of neighboring cell, add edge to opposite vertex

Longest edge refinement

Consider this mesh and a marked cell:

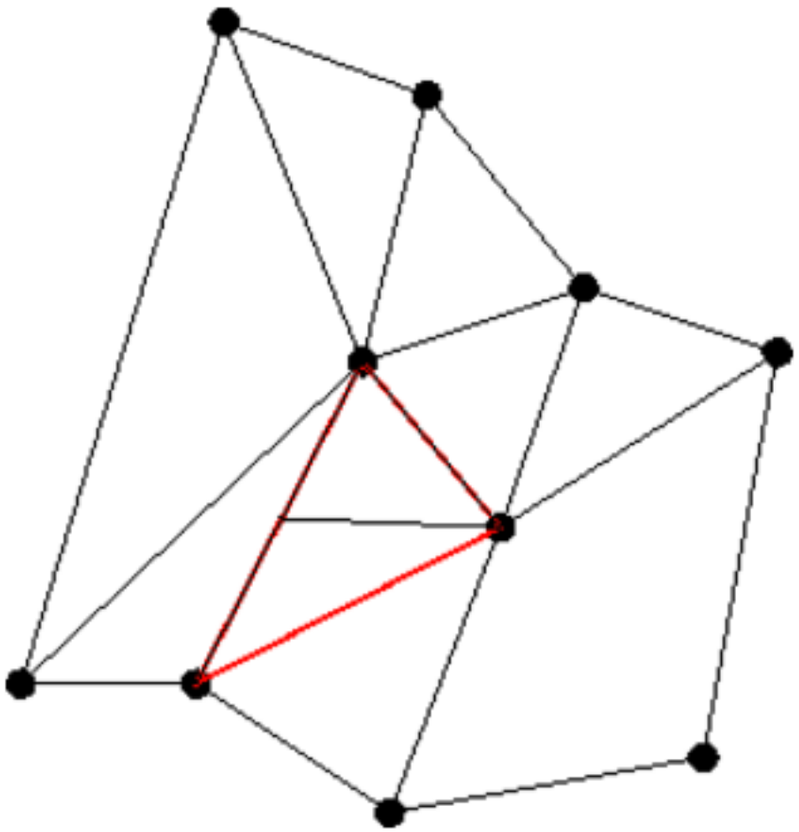


Algorithm:

- Add edge from midpoint of longest edge to oppos. vertex
- If this is also the longest edge of neighboring cell, add edge to opposite vertex
- DONE

Longest edge refinement

Consider this variant:

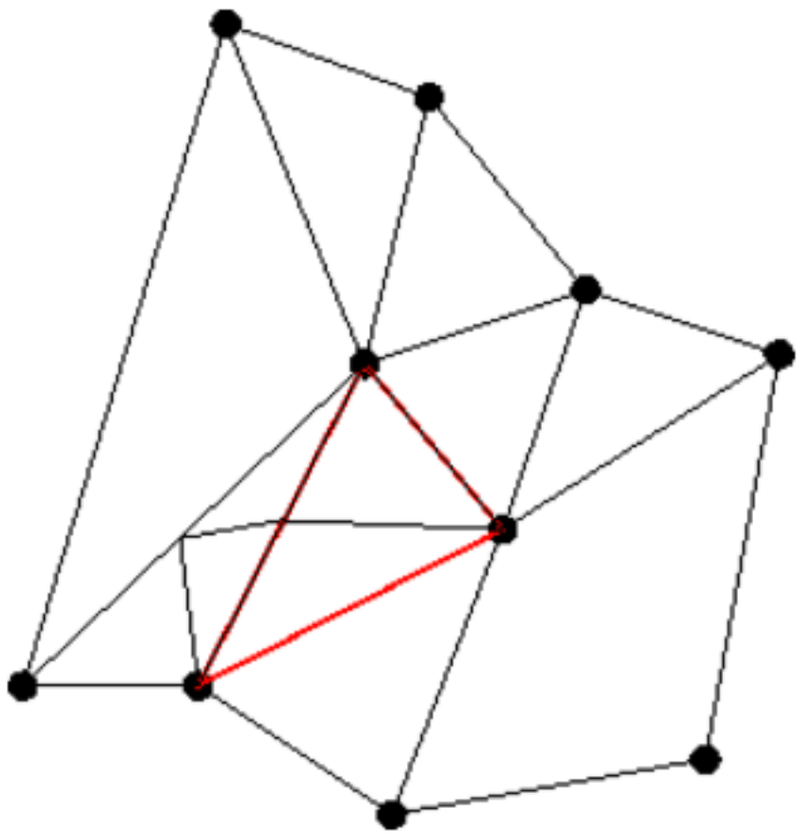


Algorithm:

- Add edge from midpoint of longest edge to oppos. vertex
- Connecting to opposite vertex of neighbor cell would yield distorted cell!

Longest edge refinement

Consider this variant:

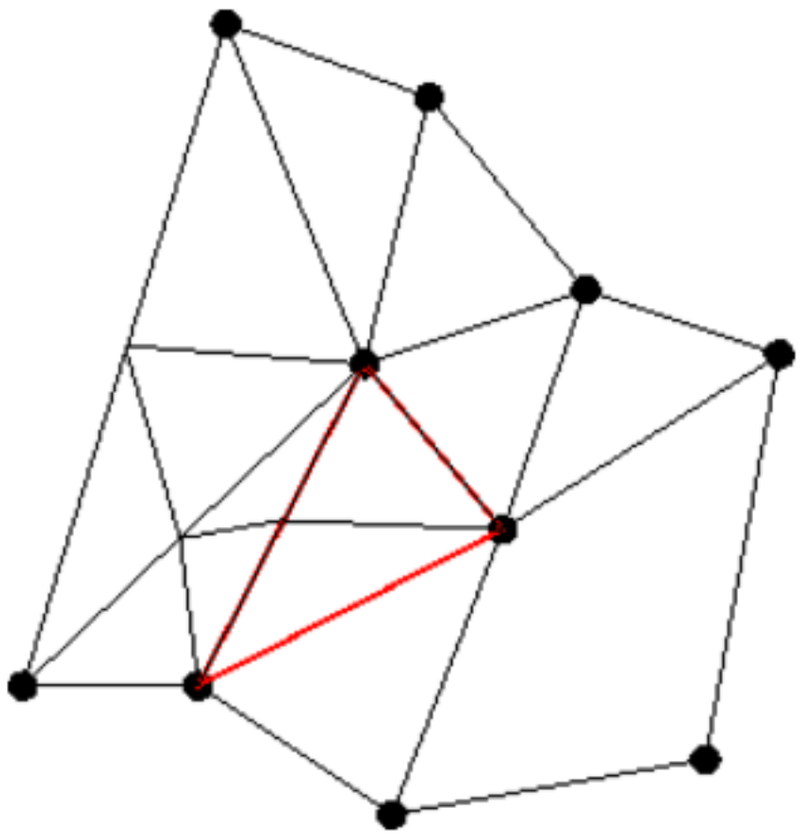


Algorithm:

- Add edge from midpoint of longest edge to oppos. vertex
- If this is *not* the longest edge of neighboring cell, add edges to midpoint of longest edge *and* from there to the opposite vertex

Longest edge refinement

Consider this variant:



Algorithm:

- Add edge from midpoint of longest edge to oppos. vertex
- If this is *not* the longest edge of neighboring cell, add edges to midpoint of longest edge *and* from there to the opposite vertex
- repeat

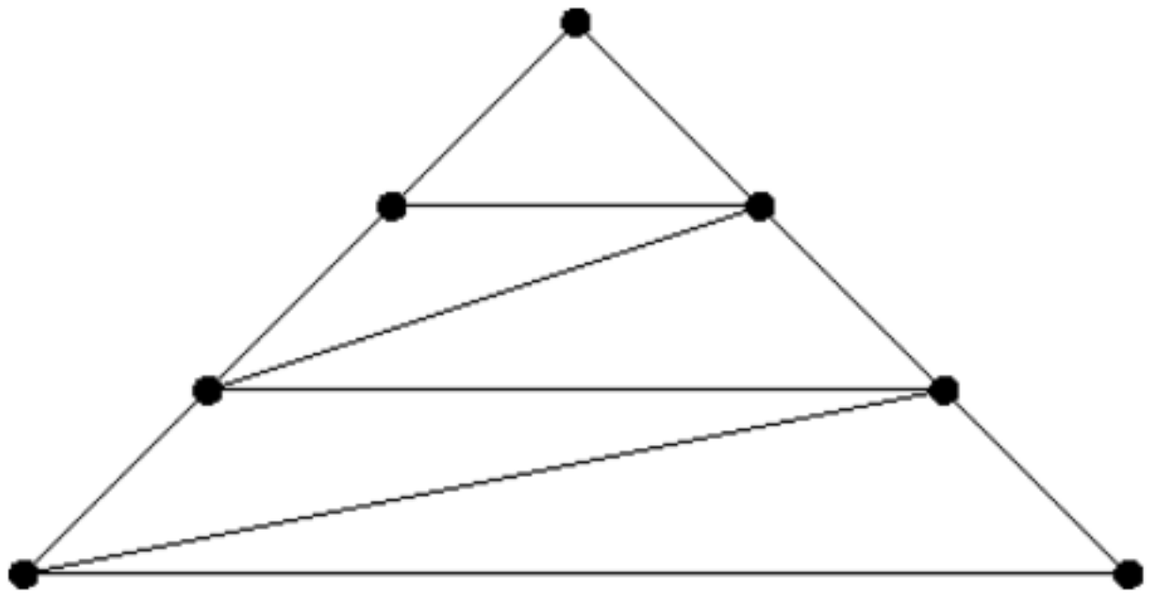
Longest edge refinement

Analysis:

- The algorithm is designed to keep the triangles from degenerating
- However, refinement is not *local*: we may have to refine a set of neighboring elements as well!

Longest edge refinement

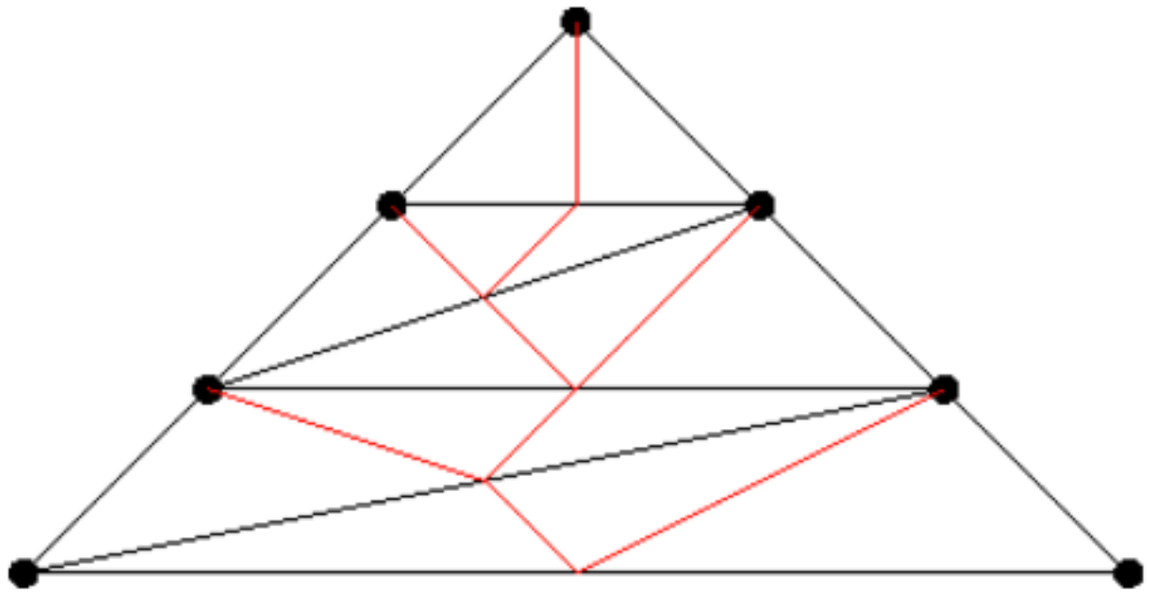
Example of runaway non-local refinement:



Goal: Refine the top-most cell.

Longest edge refinement

Example of runaway non-local refinement:

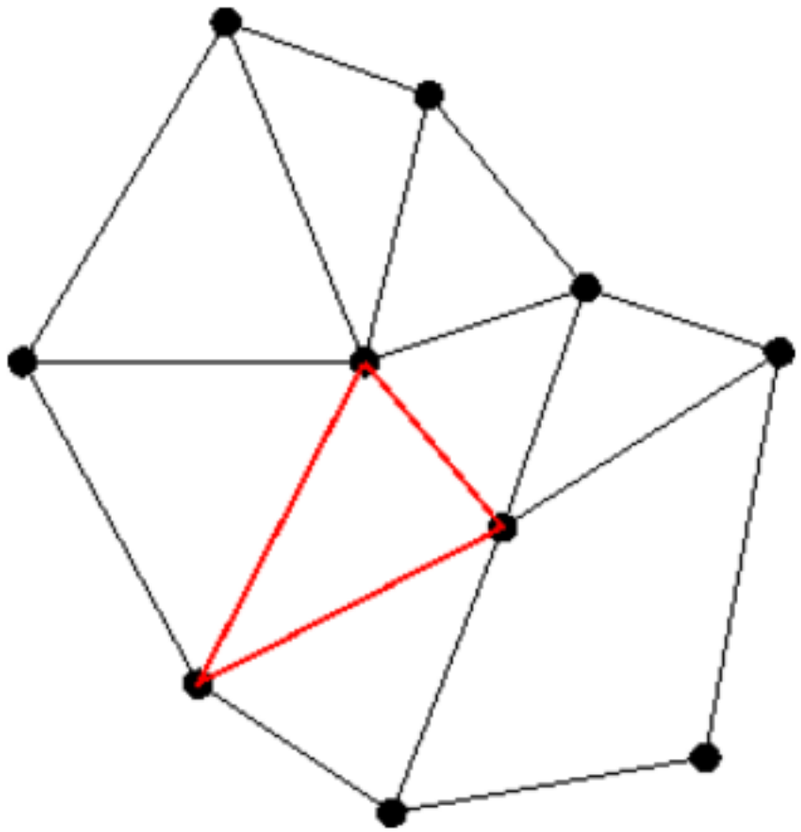


Result: *All* cells are refined!

Note: This situation appears contrived but is quite common in tight corners of difficult geometries. There are even infinite recursions!

Red green refinement

An alternative is as follows:

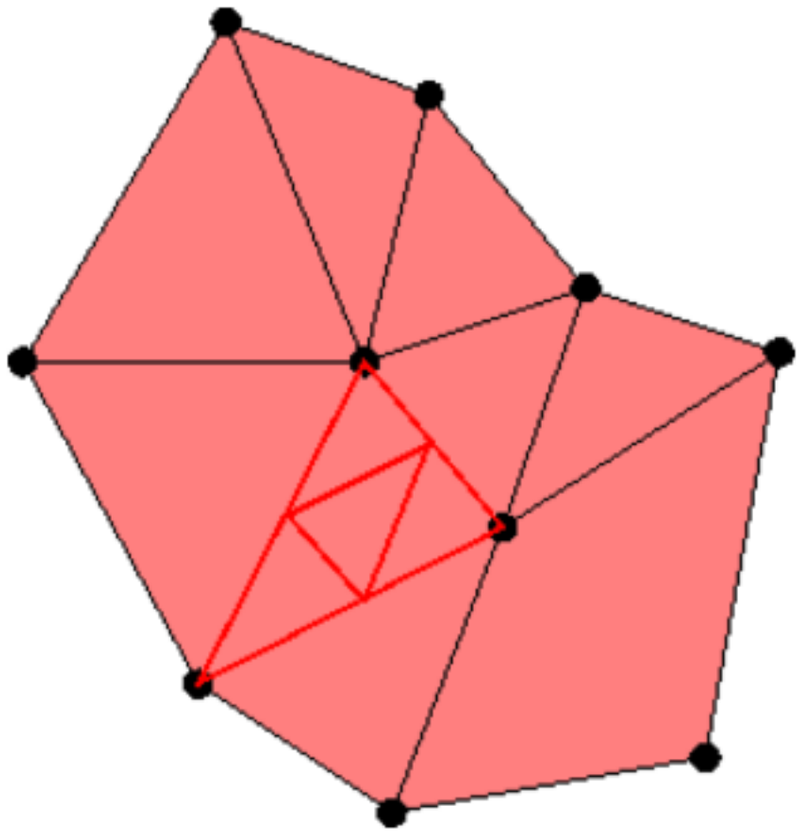


Algorithm:

– All cells start out as “red” cells

Red green refinement

An alternative is as follows:

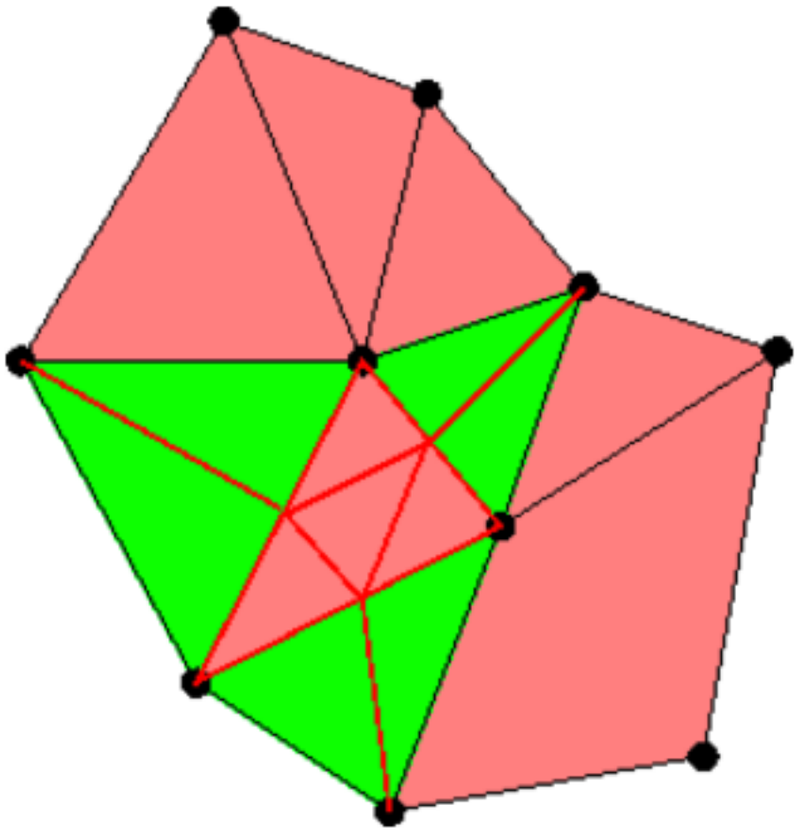


Algorithm:

- All cells start out as “red” cells
- The refinement of a “red” cell yields 4 “red” daughter cells

Red green refinement

An alternative is as follows:

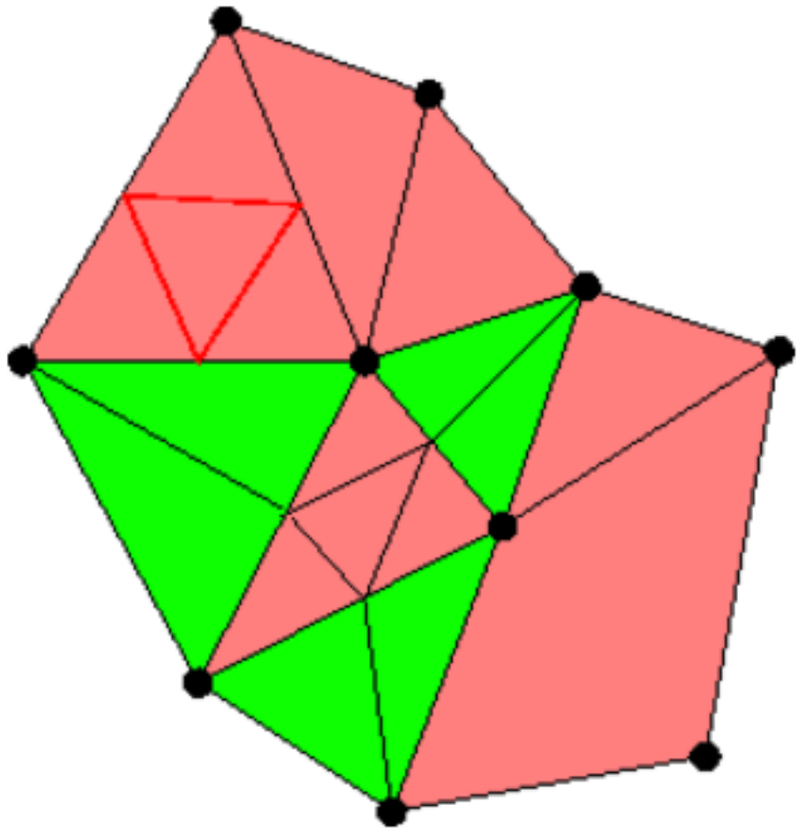


Algorithm:

- All cells start out as “red” cells
- The refinement of a “red” cell yields 4 “red” daughter cells
- Cells with “hanging nodes” are halved and become “green”

Red green refinement

"Green" cells are second-class citizens upon further refinement:

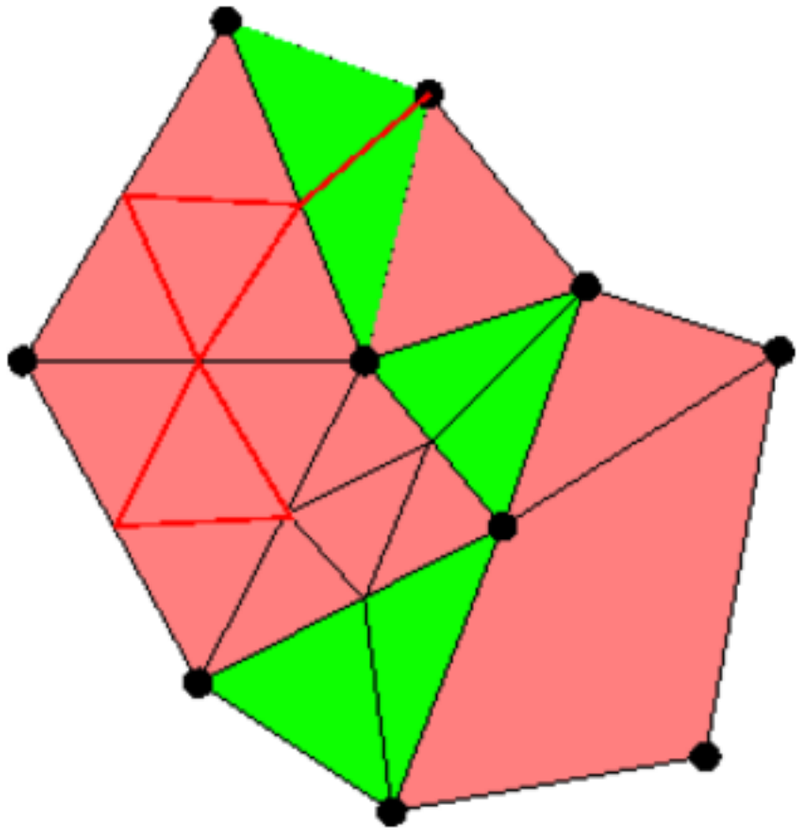


Algorithm:

- Marked "red" cells are refined as always into "red" cells

Red green refinement

"Green" cells are second-class citizens upon further refinement:



Algorithm:

- Marked "red" cells are refined as always into "red" cells
- "Green" cells are first undone and then refined as "red" cells

Red green refinement

Analysis:

Pros:

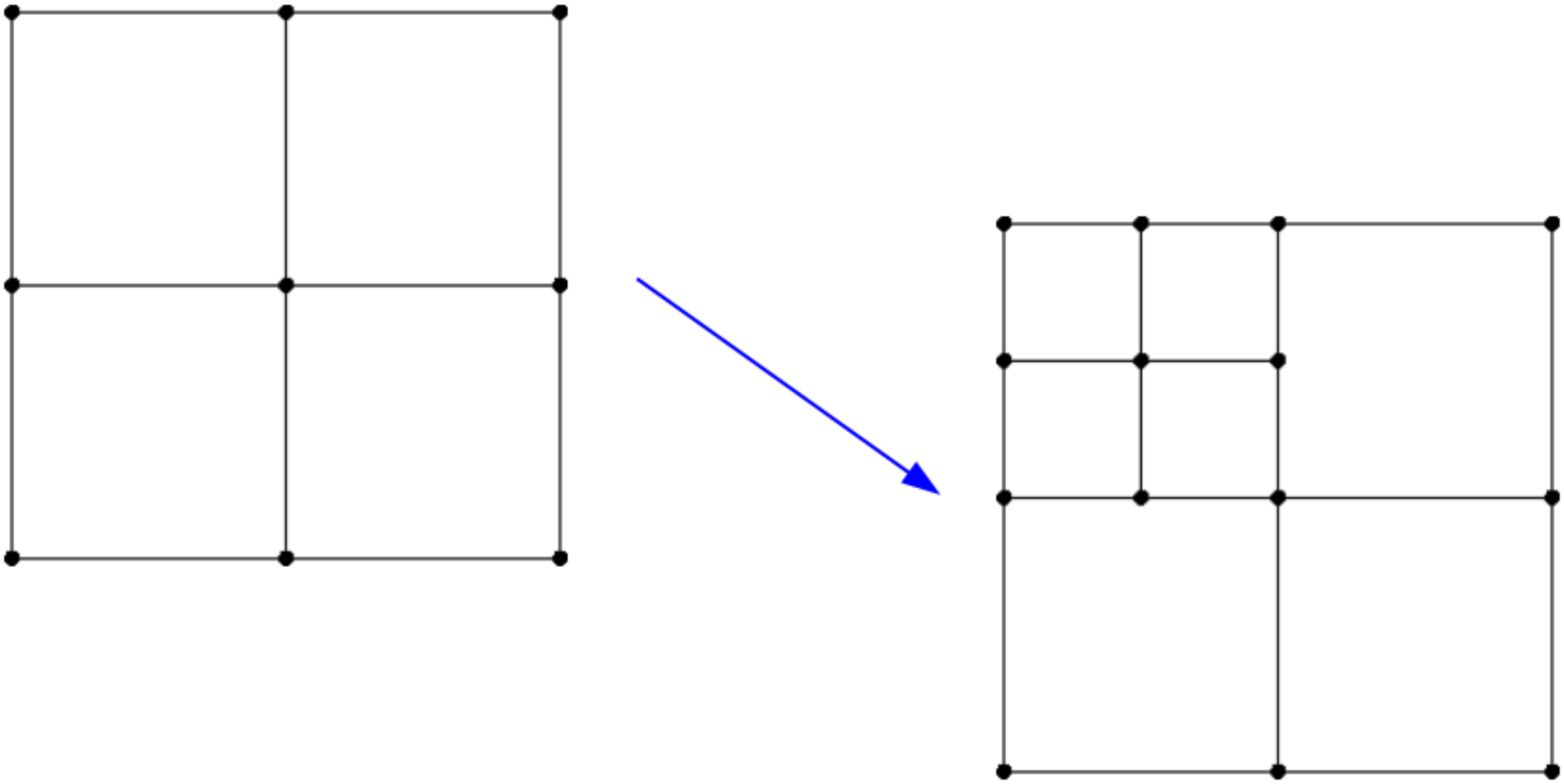
- The algorithm is designed to keep the triangles from degenerating
- Children of "red" cells are congruent to their mother
- No run-away refinement

Cons:

- However, triangles can degenerate by a factor of 2 when converted to "green" cells
- More implementation effort required

Quad/hex refinement

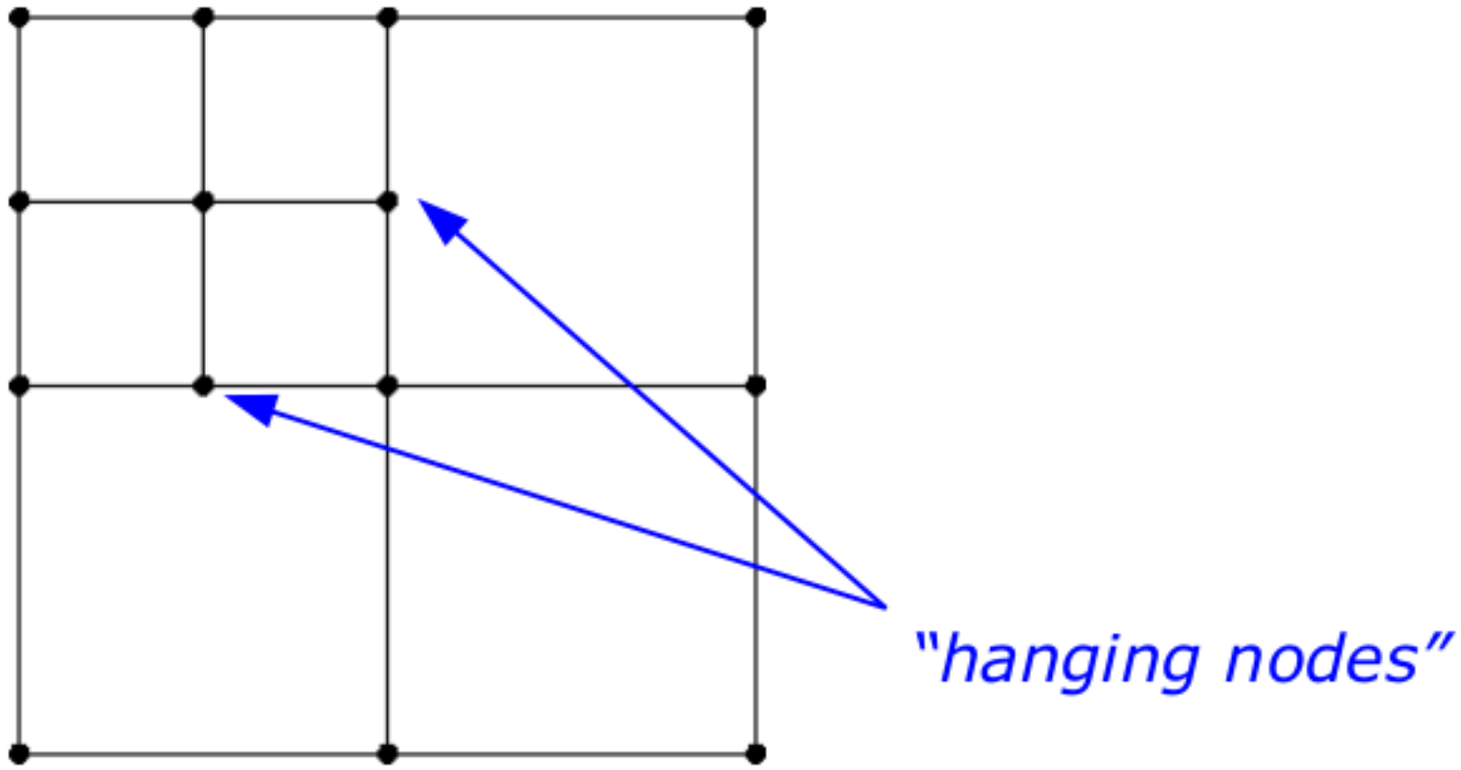
For quads/hexes, the situation is more difficult:



There are no easy options to keep the children of the top right/bottom left cell as quadrilaterals!

Quad/hex refinement

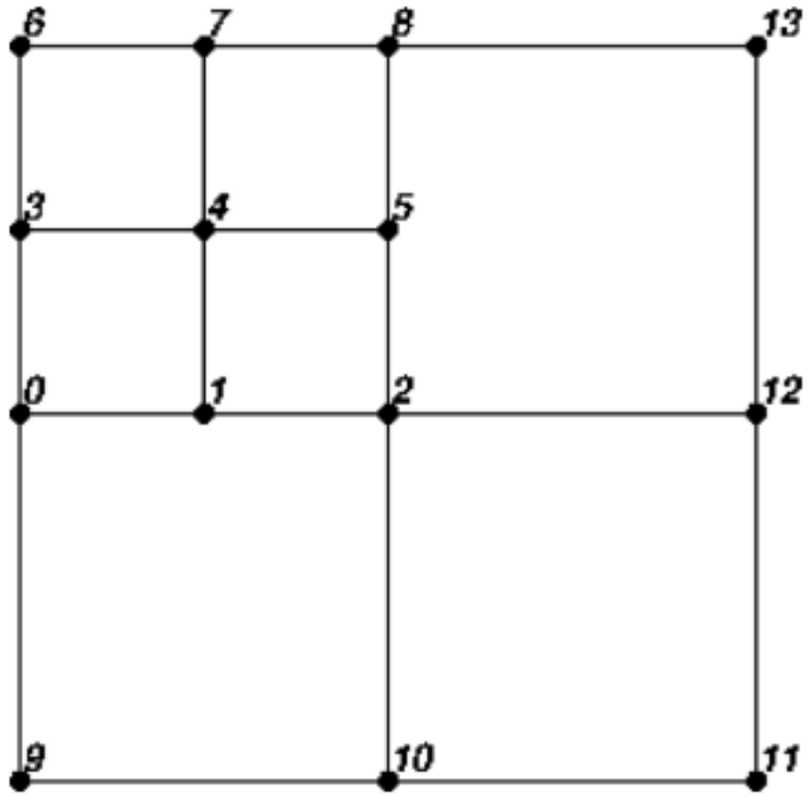
For quads/hexes, the situation is more difficult:



Adaptive quad/hex meshes usually just keep these "hanging nodes" and have other ways to deal with the consequences!

Why constraints?

Consider this mesh, Q1 elements, and DoFs as enumerated:



The corresponding space has dimension 14.