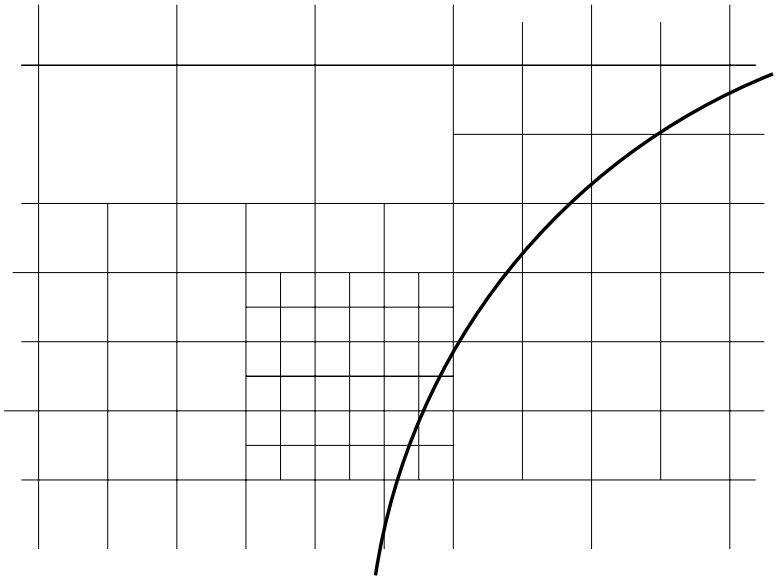
High Resolution Adaptive Methods for Complex Flows



Marsha Berger
Courant Institute
New York University

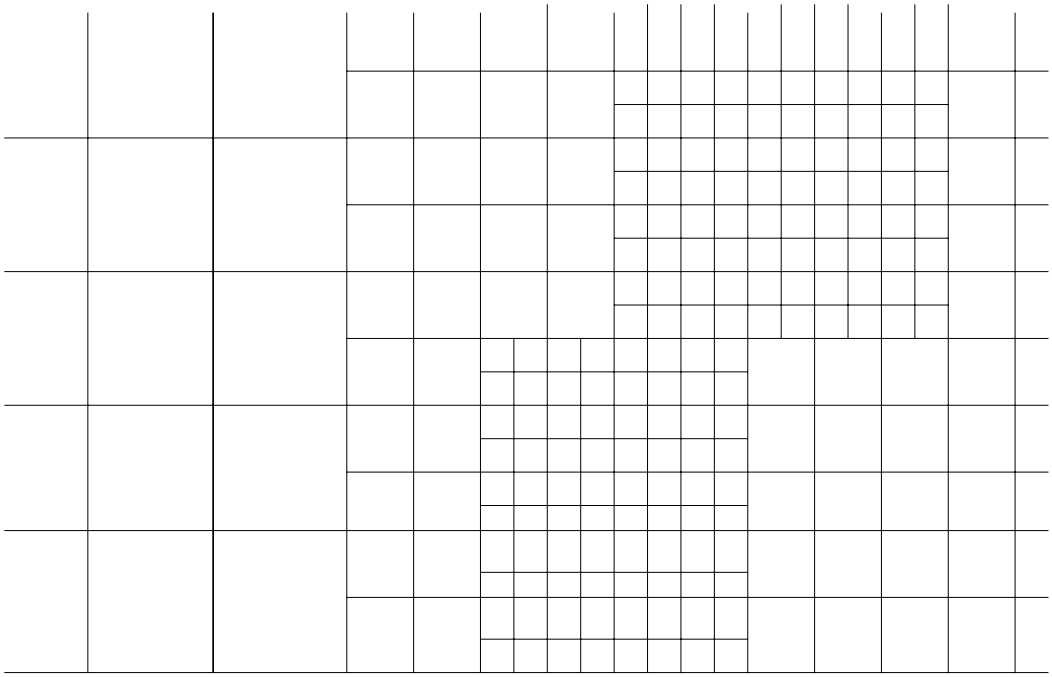
Research Program

Automate the computation of high resolution simulations in realistic engineering applications.

- Geometry specification
- Mesh generation
- Algorithm development
- Numerical discretizations
- Adaptive techniques
- Robust software
- High performance computing

Many of these issues are being studied in the DOE-supported CMCL (Courant Mathematics and Computing Laboratory). We have closely collaborated over the years with researchers at LBL, LANL and LLNL.

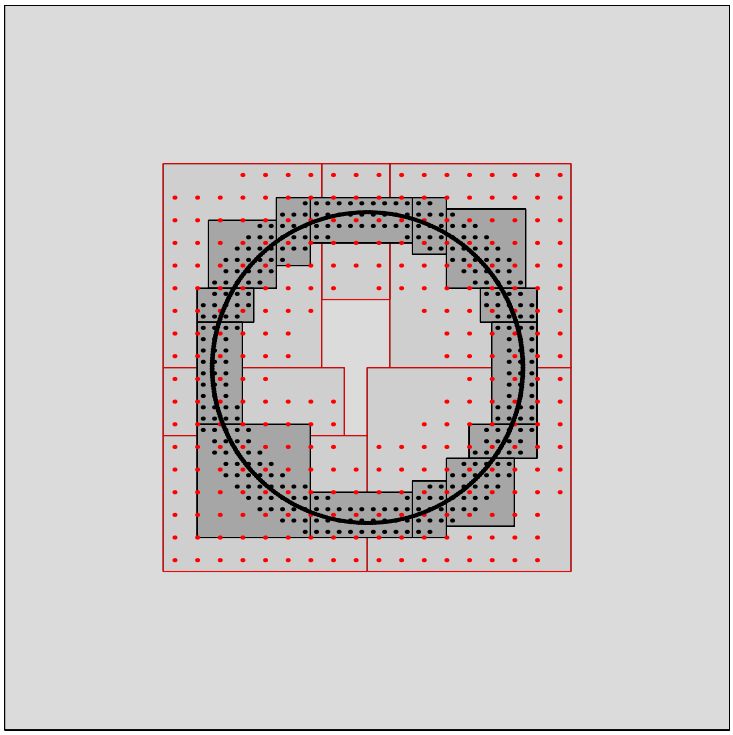
Adaptive Mesh Refinement



Use recursively nested locally refined block structured grids to attain given level of accuracy.

- same integrator advances solution on all grids
  nice user interface
- subcycle in time
  refine in time and space $\rightarrow$ *constant CFL*
- stable, accurate and conservative interface conditions

Adaptive Mesh Refinement



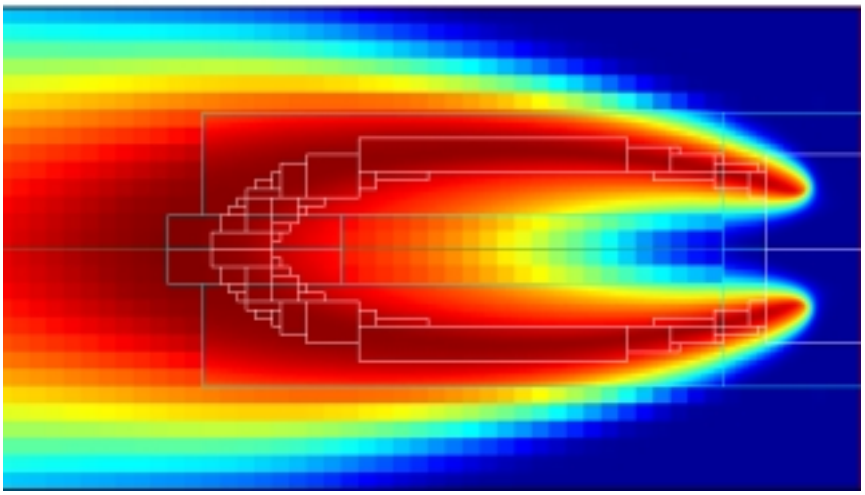
- automatic error estimator
  (Richardson LTE estimates)
  tags regions of high error

- automatic grid generator
  (clustering via edge detection algs.)
  create fine grid patches in those regions

- simple data structure

Some Extensions and

Outstanding Problems



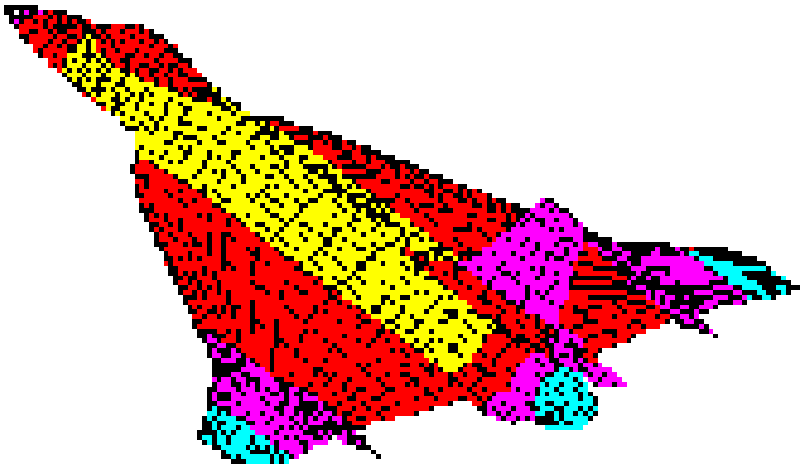
(ref: Bell et al)

- Incompressible Flows, Reacting Flows, etc.
- Error Estimation
- Implicit Schemes
- Directional Refinement
- Software and Parallelization Issues

Complex Geometry

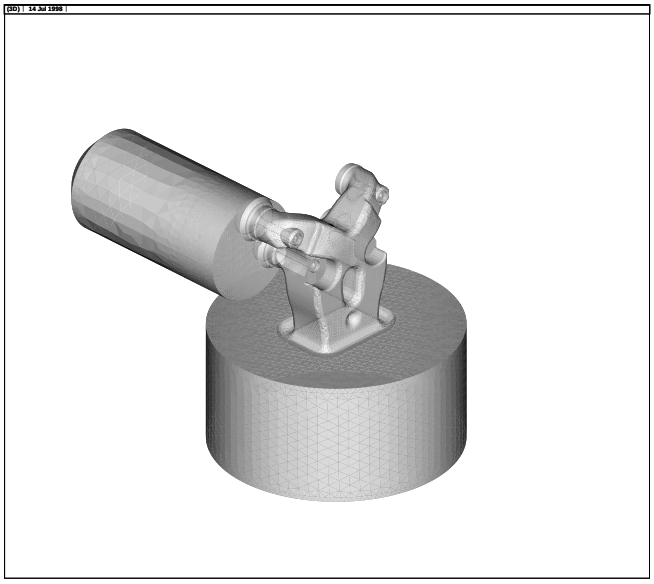
Develop automatic methods for rapid turnaround flow computations in complex geometries.

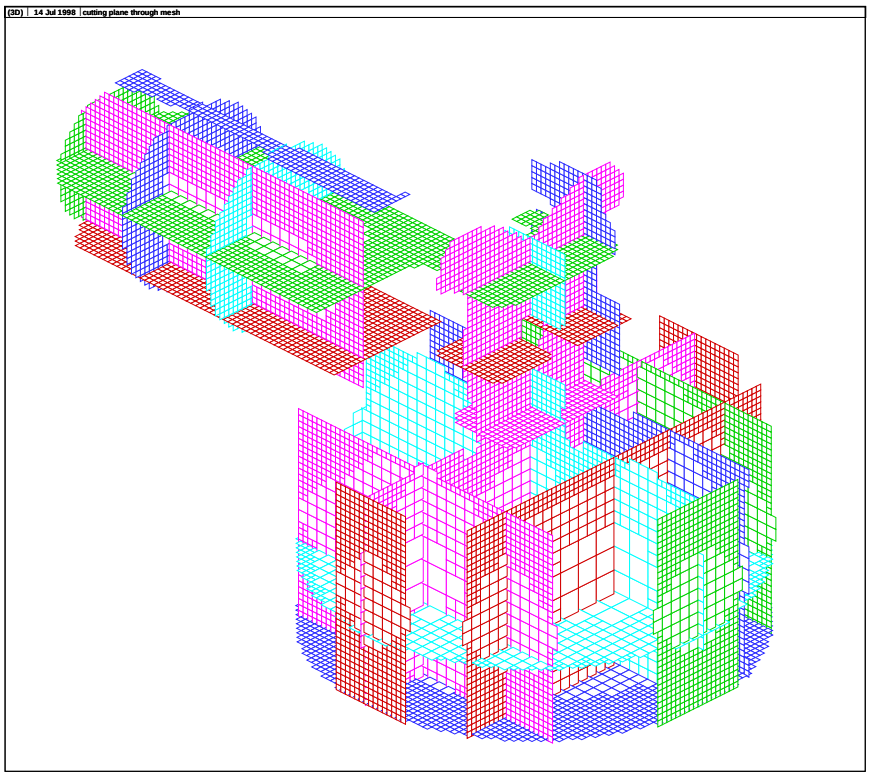
- CAD definition
- Surface Description
- Volume Mesh Generation
- Flow Computation
- Post-Processing



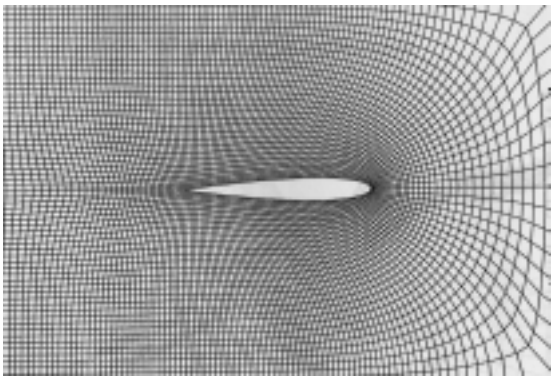
Main bottleneck is geometry acquisition

Caterpillar Diesel Engine Example





Alternative Approaches



Structured Meshes

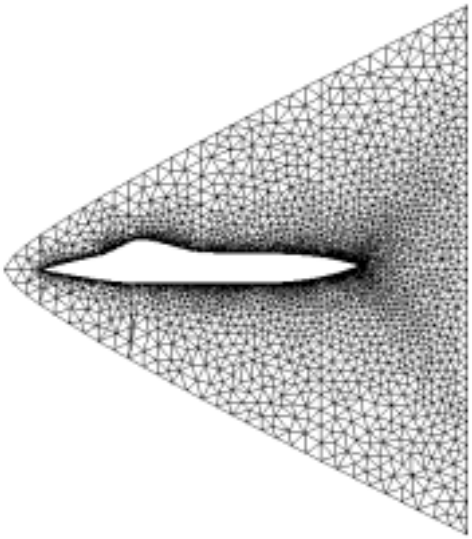
Advantages:

accurate

efficient

Disadvantages:

difficult to generate



Unstructured Meshes

Advantages:

general domains

easy to program

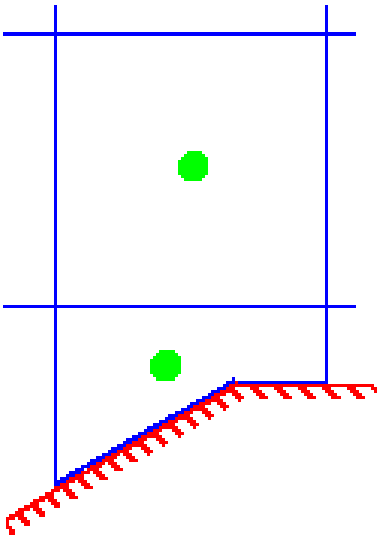
Disadvantages:

memory & CPU intensive

Both methods need nice surface description.

Cartesian Non-Body-Fitted Grids

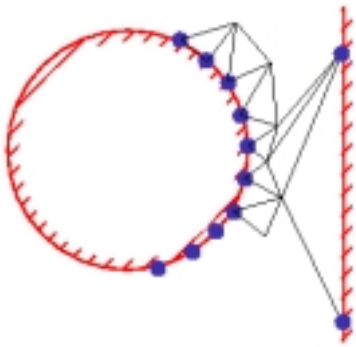
Use regular Cartesian grids with solid objects *cut out* of the underlying grid. Objects represented by piecewise linear segments (2D) or surface triangulation (3D)

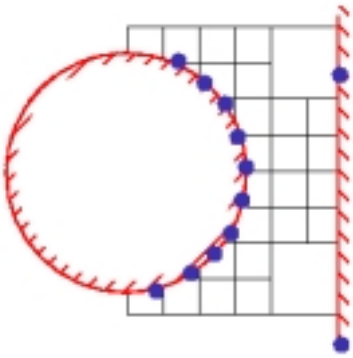


- **Multicomponent Preprocessor** – automate the mesh generation from CAD geometry
- **Mesh Generator** – use adaptively refined Cartesian cells with embedded geometry; treat cells that cut the body as general polyhedra
- **Flow Solver** – develop numerical discretizations for cut-cells; use space filling curves for domain partitioning and multigrid coarsening

Why Cartesian Meshes?

- finite difference schemes on regular grids can be highly efficient, well vectorized, and have shock capturing and convergence properties that are well understood
- irregularity confined to lower dimensional space *do not pay efficiency penalty over entire domain*



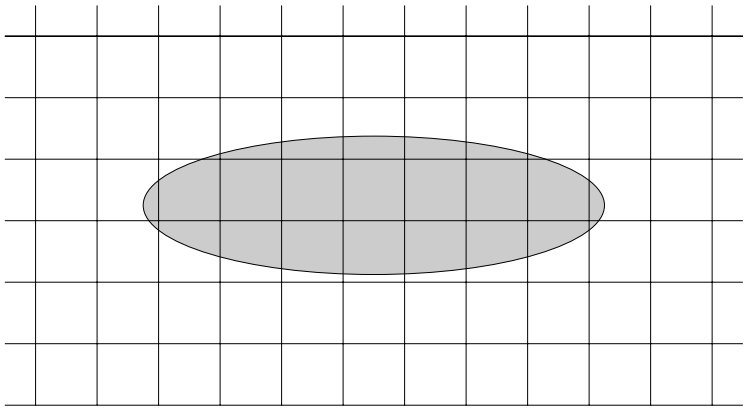


- easier grid generation: surface grid *not* the computational grid (surface description resolves geometry; Cartesian mesh describes flow).

Why Not Cartesian Meshes?

- Cartesian grids lack the resolution of body-fitted or unstructured grids (use AMR)
  no boundary layer zoning

- irregular cells - loss of accuracy at boundary



- small cell instability

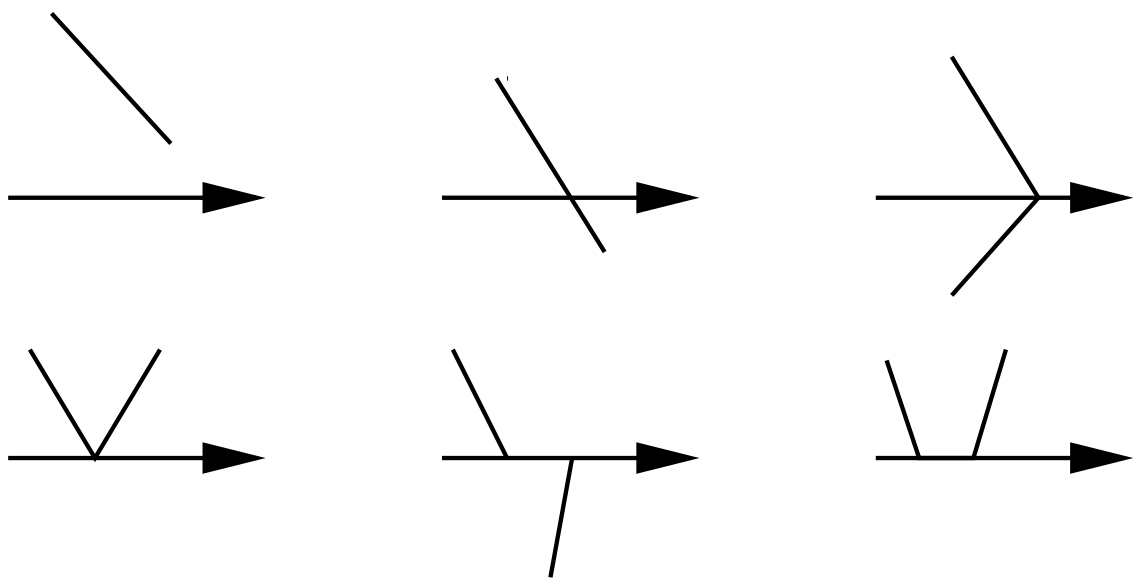
Need stable, conservative accurate schemes with CFL based on regular cells.

Multiple Component Geometry

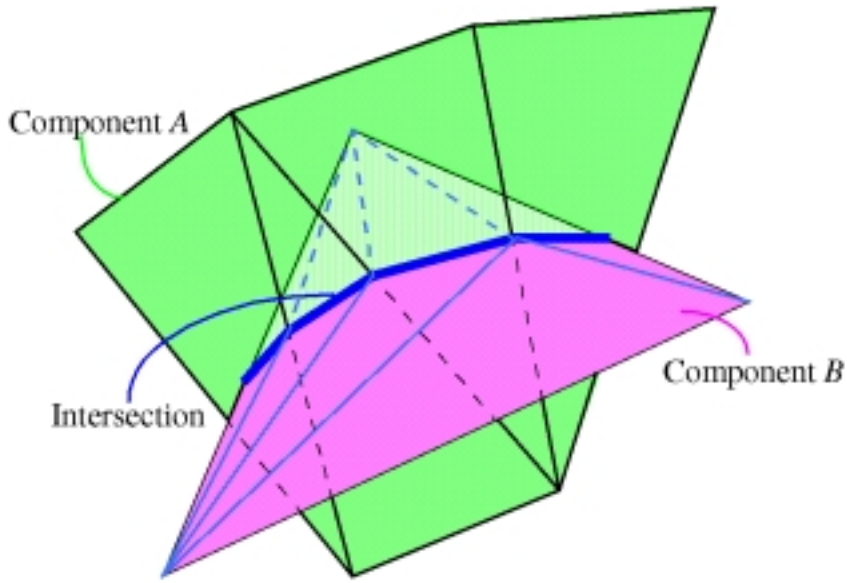
Motivation: Allow separately defined watertight component triangulations as input to mesh generation.

Problem: How to compute topologically consistent surfaces in the presence of floating point round-off error.

Difficulty: Degeneracies make this very complicated.



Intersection Algorithm

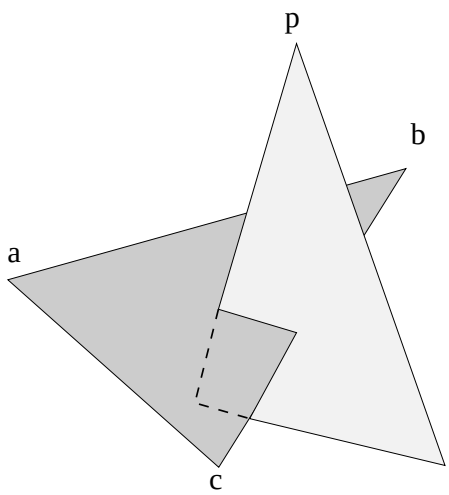


Strategy:

- Intersect component triangulations.
- Retriangulate intersected triangles.
- Remove internal geometry.
- Resolve degeneracies with adaptive precision determinant computations (Shewchuk) and virtual perturbation tests (Mucke and Edelsbrunner.)

Intersection Algorithm

Triangle intersection boils down to multiple computations of a 4 by 4 determinant of the signed volume of a tetrahedron T:



$$6V(T_{abcp}) = \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ p_x & p_y & p_z & 1 \end{vmatrix}$$

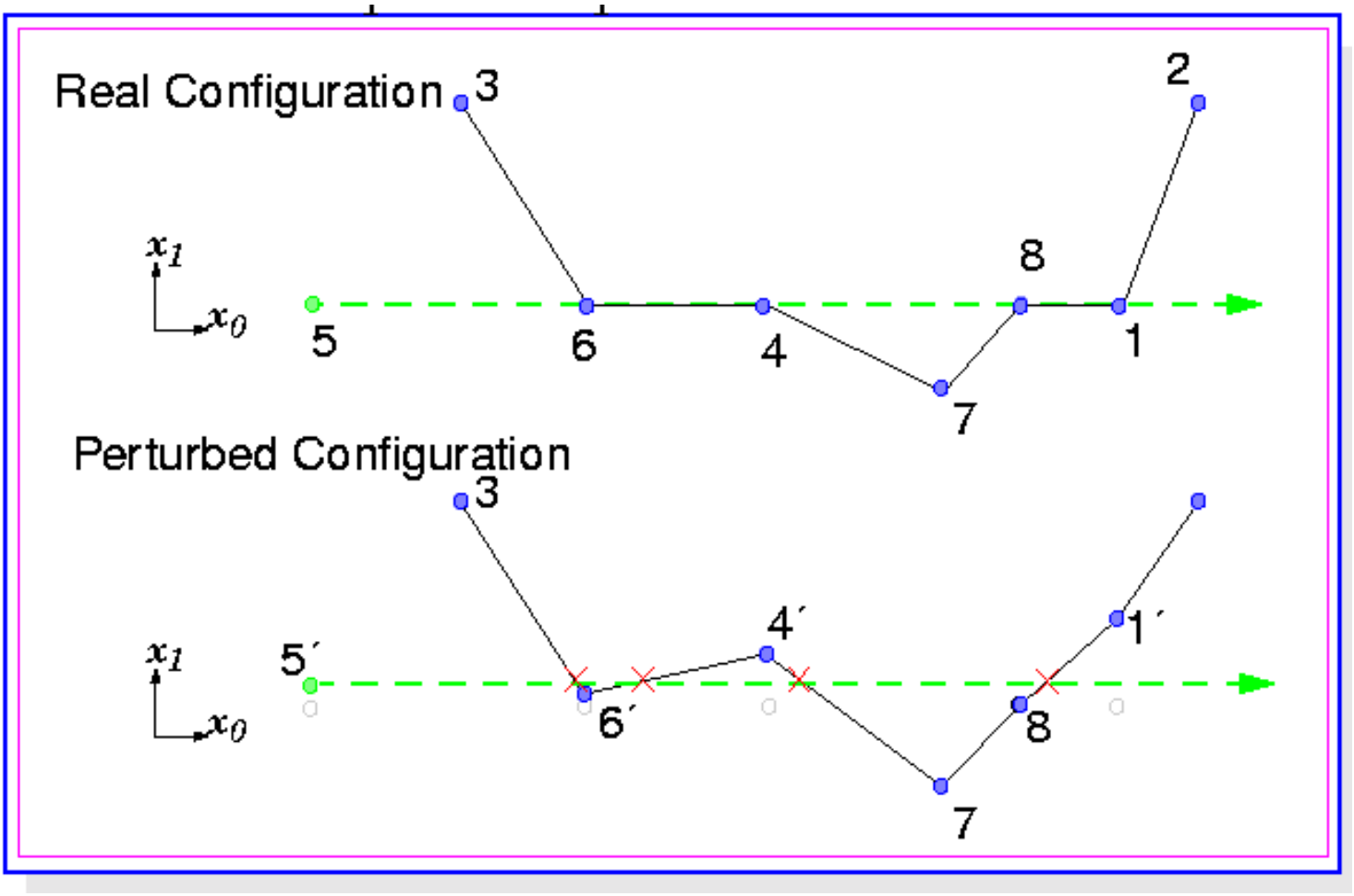
- Compute det using floating point arithmetic.
- Floating point filter: if result $>$ error bound, recompute using adaptive-precision exact arithmetic (Shewchuk).
- If exact arithmetic gives det $= 0$, use tie-breaking algorithm to resolve the degeneracy (Edelsbrunner and Mucke).

Intersection Algorithm

Virtual perturbation approach using SOS technique of Edelsbrunner and Mucke.

- Perturb $(i, j)^{th}$ element by $\epsilon_{i,j} = \epsilon^{2^{i \cdot d - j}}$
- Expand determinant in powers of epsilon.
- First non-zero term determines sign of the determinant.

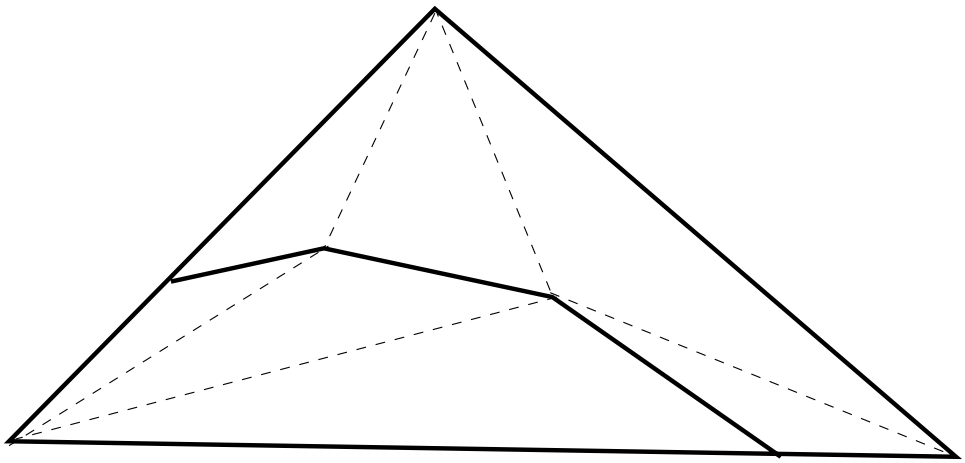
Tie-Breaking Example

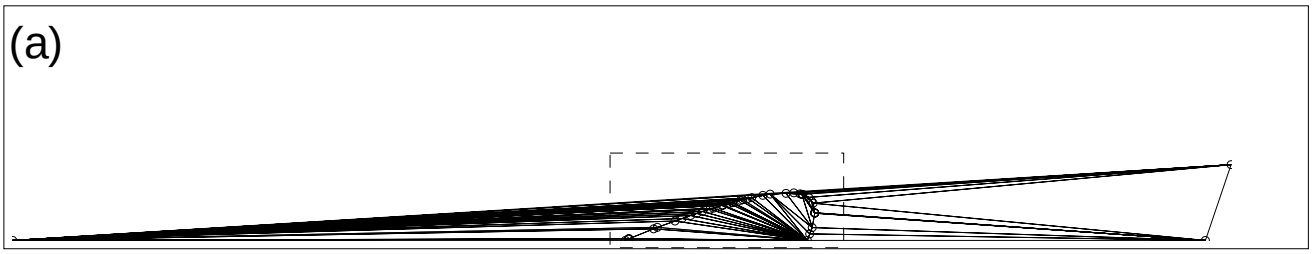


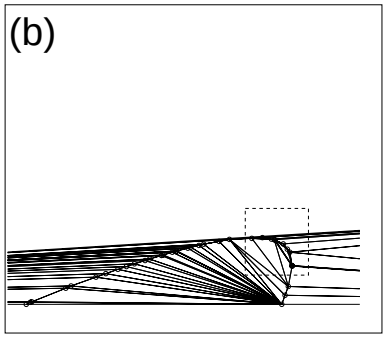
Example of the perturbation for resolving degeneracies.

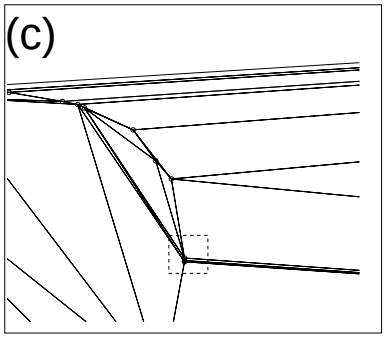
Intersection Approach

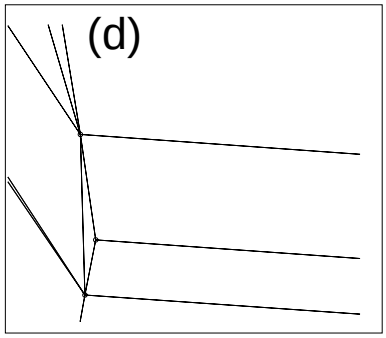
Intersected triangles are retriangulated using constrained Delaunay triangulation.





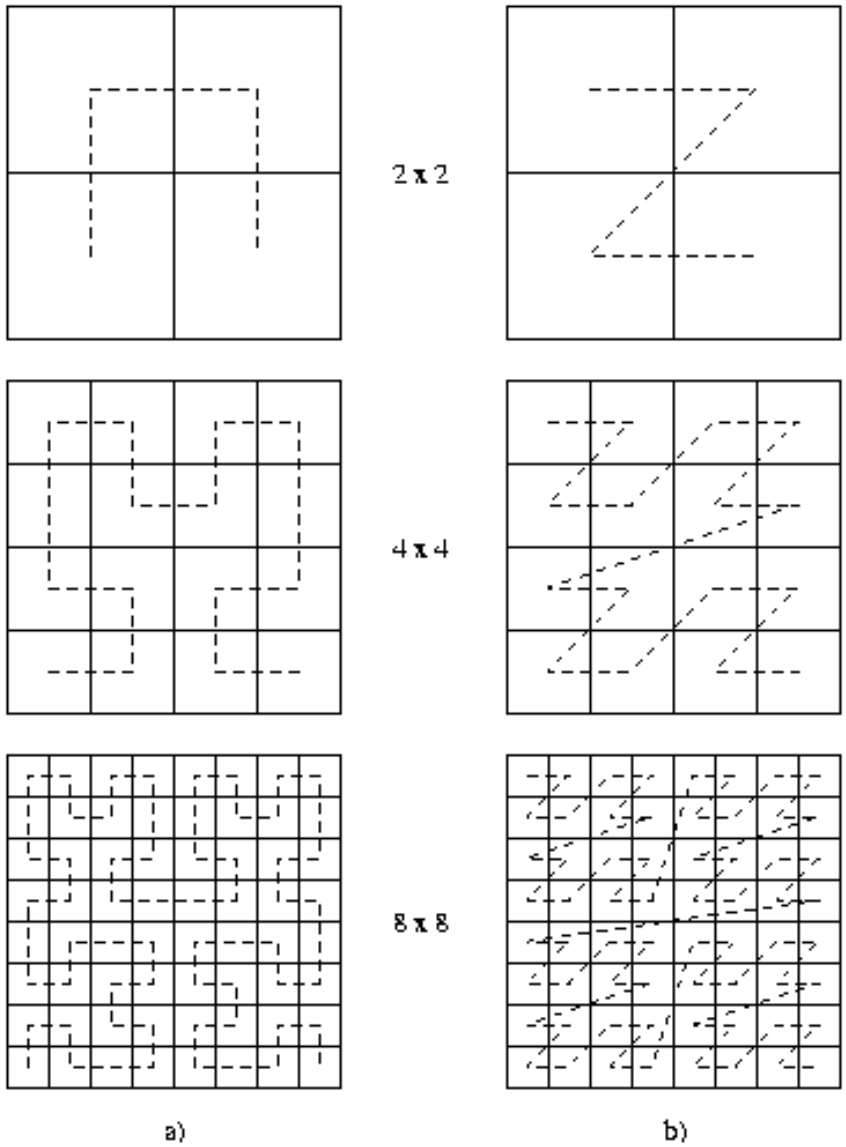






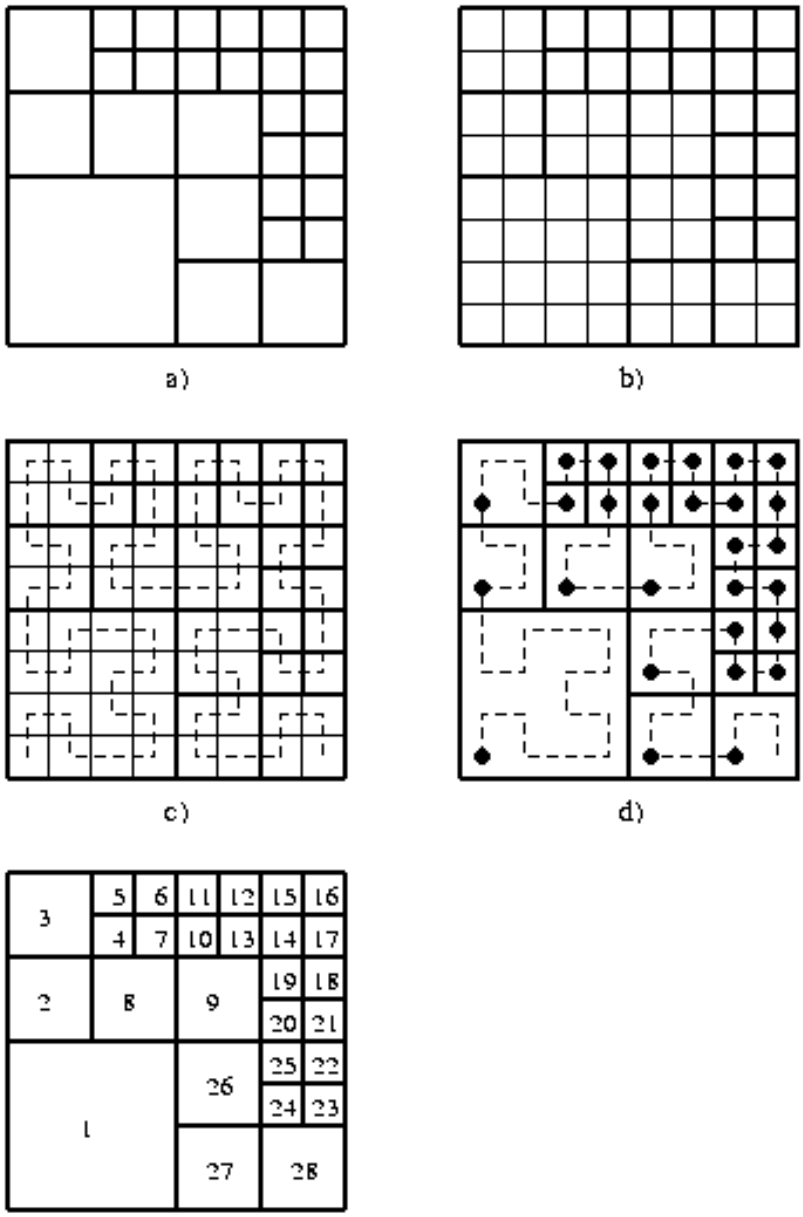
Space-Filling Curves

Space-filling curves linearly order a multi-dimensional mesh. (Pilkington & Baden, Griebel & Zumbusch, ...)



(a) Peano Hilbert ordering (b) Morton ordering

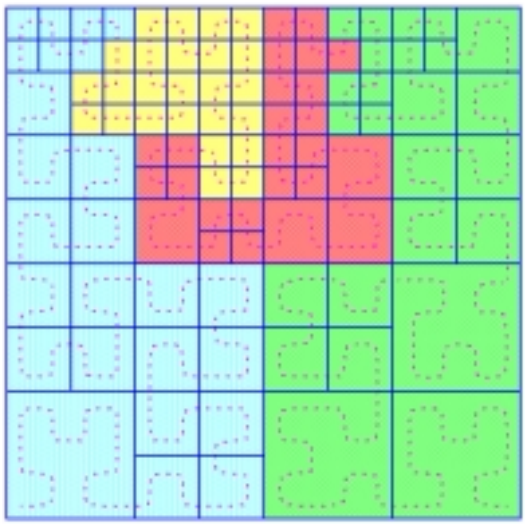
Space-Filling Curves

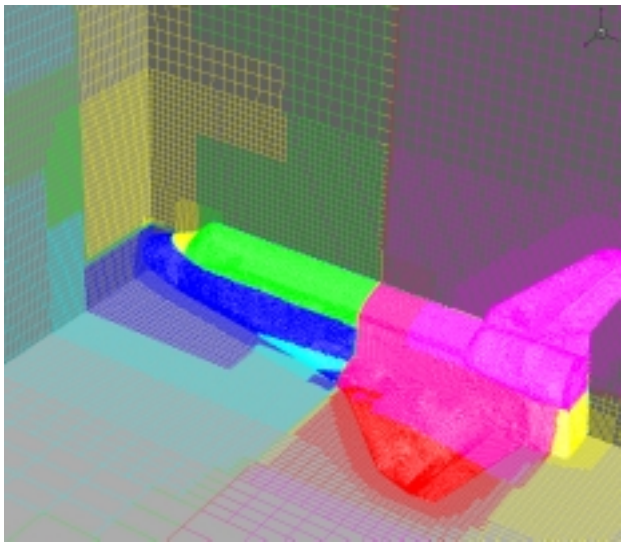


Use of Peano-Hilbert curve on adaptively refined meshes.

Domain Partition

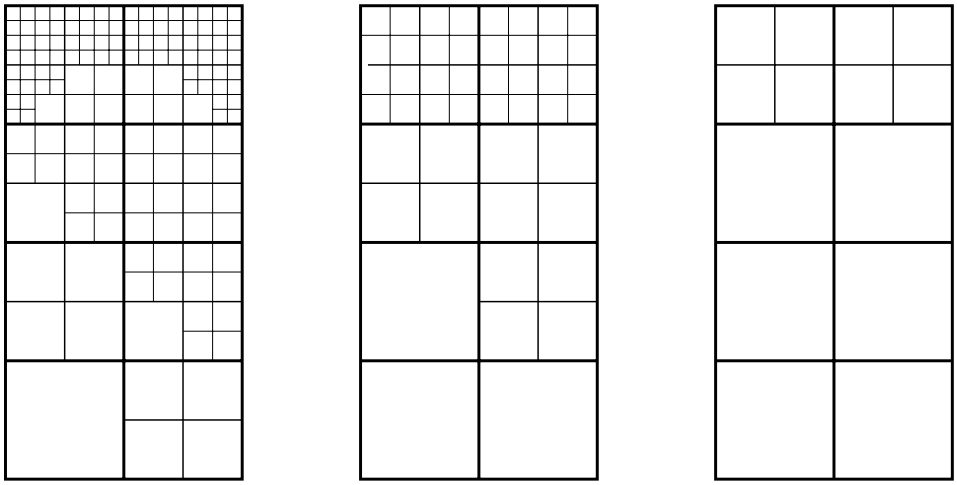
- Use work estimates based on cell types to partition the mesh into load balanced domains.
  (Cut cell work = 2.7 $\times$ Full cell work)
- Done on the fly into any number of partitions.



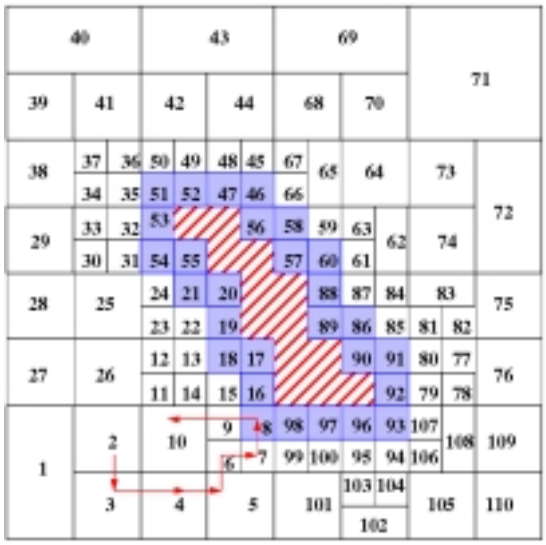


- Explicit message passing via shared memory.
- Ghost cells hold neighbors one away for each partition.

SFC Mesh Coarsening



Cartesian cells cannot coarsen until all its siblings cells are at the same level of refinement.



The SFC-ordered mesh puts siblings together, so easy to check if cells can coarsen.