Robust Tracking and Advanced Geometry for Monte Carlo Radiation Transport

Brandon Smith
bmsmith6@wisc.edu

University of Wisconsin-Madison

February 22, 2011

Outline

Radiation Transport

Design and licensing of complex nuclear systems requires predictive capabilty

- Purpose: Simulate particle interactions across space, angle, energy, (time)
- Input: geometry, source, boundary conditions, cross sections
- Output: multiplication factor, flux, or derived quantities
- Method: deterministic or stochastic

Deterministic Methods

- Discretized space, angle, energy, (time) domains
- Solve PDEs describing particle flux in each mesh element, energy group
- Difficult in material with limited scattering
- Solution exists over entire domain
- Error depends on mesh density, angular resolution, and energy groups

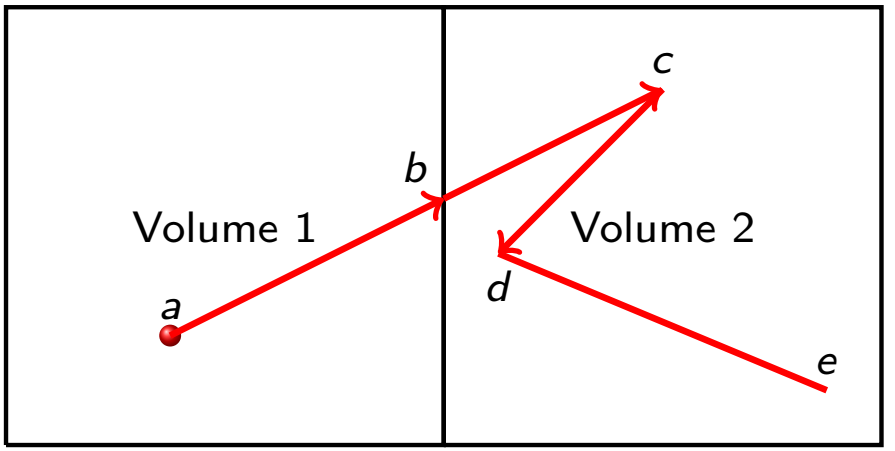
Stochastic Methods

- Continuous space, angle, energy, (time) domains
- Transport individual particles through phase space
- Accurate in material with limited scattering
- Solution exists only where specified
- Statistical error depends on particle population

Monte Carlo Method

Geometric Operations

- Measure
- Point Inclusion
- Next Surface
- Next Volume
- Closest Surface
- Surface Normal



Benefits of CAD Geometry

Efficiency	Common geometric domain for all types of analysis; Reduced human effort
Fidelity	CAD allows richer surface representation
Accuracy	Avoid human error when creating separate radiation transport model

CAD Geometry Not Inherently Suitable for Radiation Transport

BREP vs. CSG	Native support in solid modeling engines vs. MC transport codes
Nonsolid Space	Nonsolid space is not explicitly modeled in CAD models
Gaps & Overlaps	Small gaps and overlaps are common in CAD models
Small Details	Not all details are relevant to radiation modeling

Direct Accelerated Geometry: DAG-MCNP

DAG-MCNP is a coupling of the Mesh Oriented datABase (MOAB) and Monte Carlo N-Particle (MCNP) software packages [Tautges et al., 2009]

Software

- Common Geometry Module, Argonne (CGMA): geometry library, C++, open source
- MOAB: mesh library, C++, open source
- MCNP: physics package, FORTRAN, from LANL

Implementation

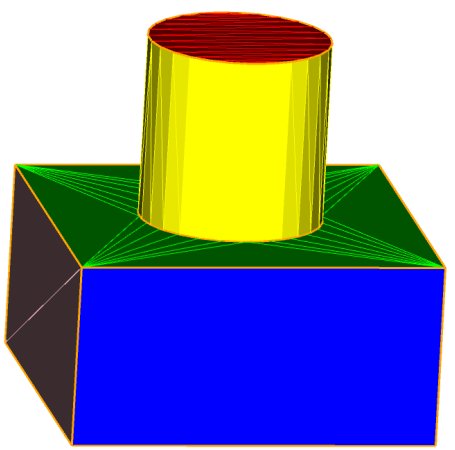
1. CGMA loads models with ACIS or OCC
2. MOAB calls CGMA to facet model
3. MCNP calls MOAB to perform geometric queries

Geometric Model: Faceted CAD Data

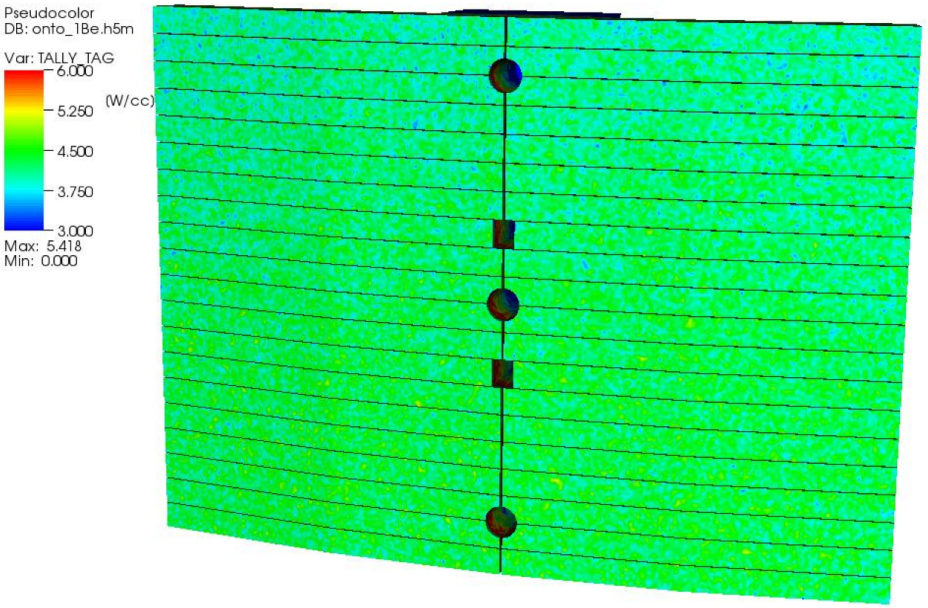
- BREP: Volumes, surfaces, curves, vertices
- Solid model→faceted-based model (FBM)
- Oriented Bounding Box (OBB) tree accelerates ray tracing

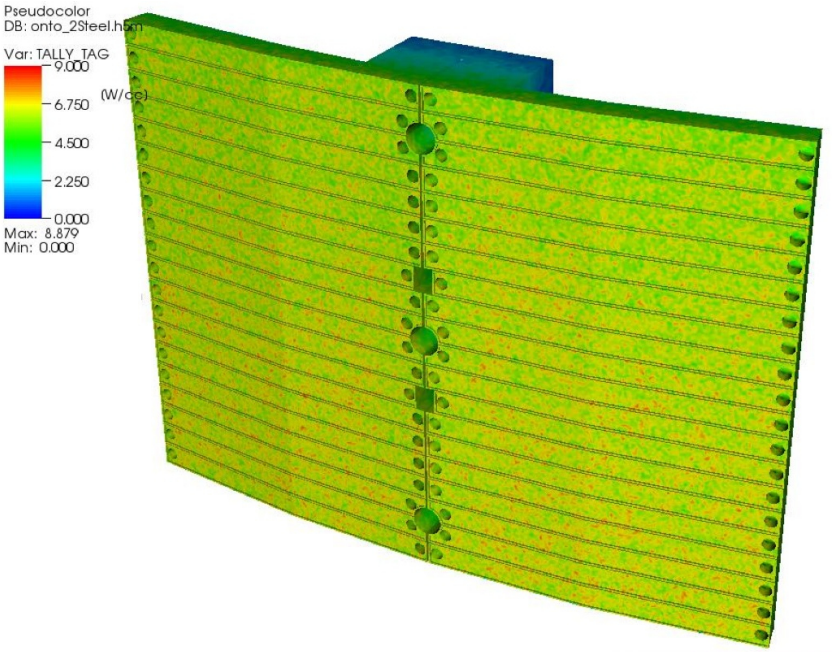
Analysis Procedure

1. Create Solid Model
2. Preprocess Geometry for DAG-MCNP
3. Create MCNP Input File
4. Run DAG-MCNP
5. Visualize Results

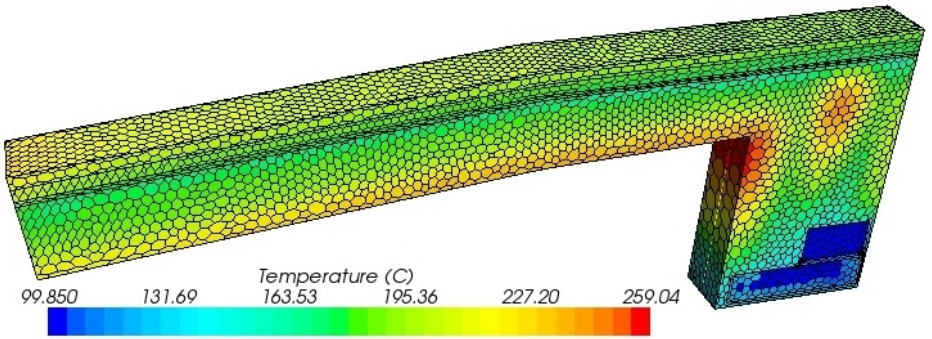


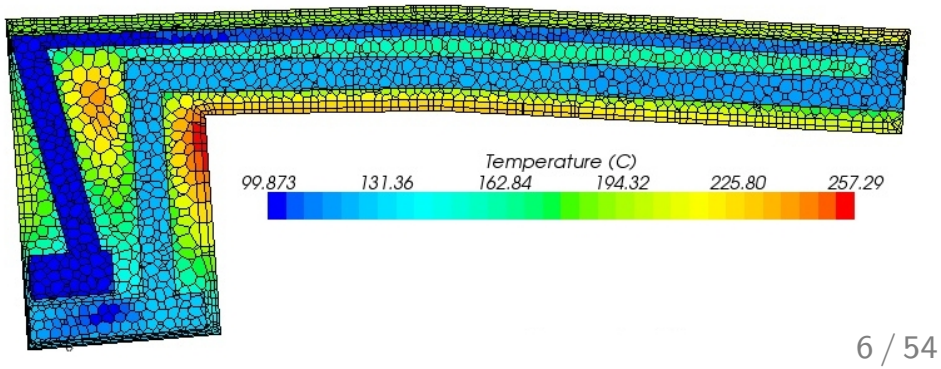
DAG-MCNP Application: ITER Module 4 First Wall/Shield



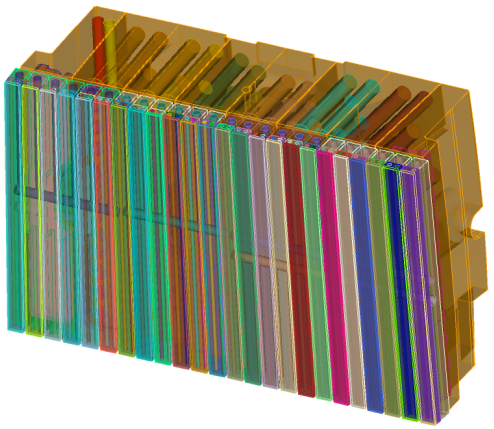


- Accurate source profile achieved by inserting Module 4 directly into ITER model
- Complex model: >1000 volumes, >13,000 surfaces
- 240 computer-days 2.66 GHz Intel Core2 processors, 500M particles
- Couple nuclear heating to CFD using Star-CCM+

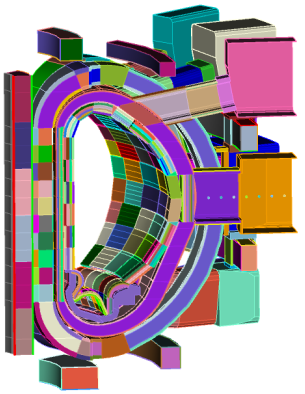




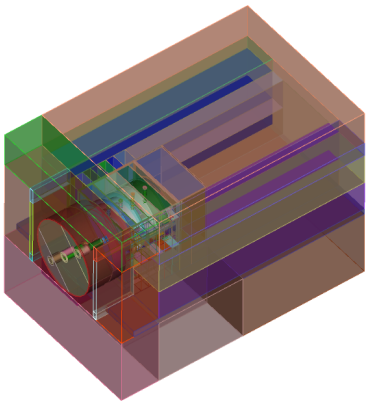
Improvement Motivated by Applications



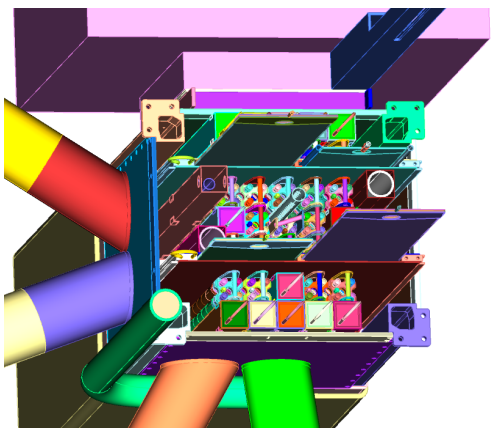
ITER FWS Module 13



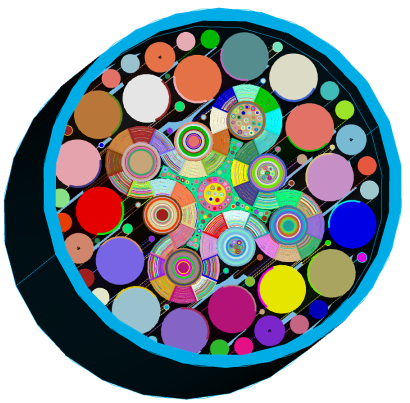
40° ITER Benchmark



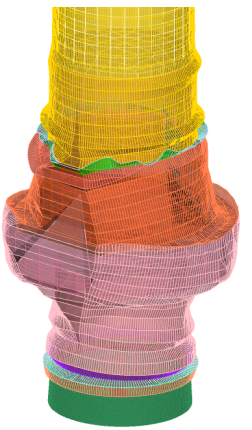
FNG Benchmark



UW Nuclear Reactor



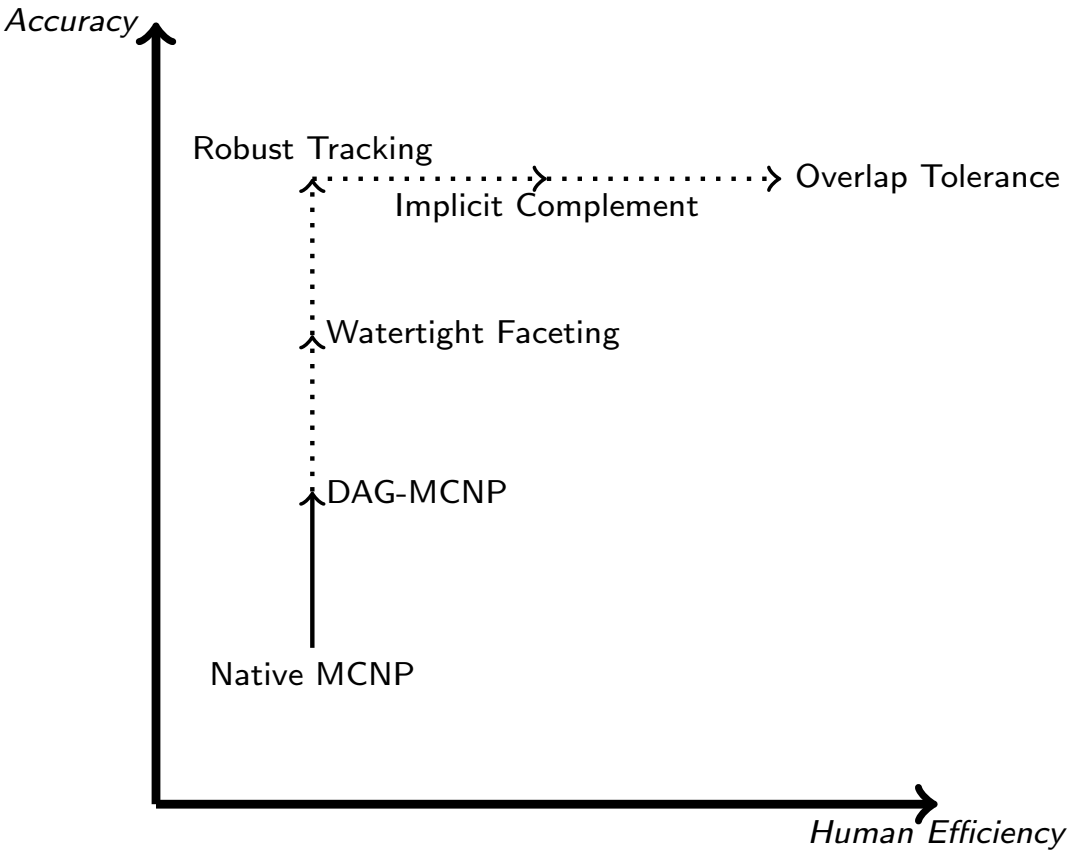
Advanced Test Reactor



Deformed Space Reactors

Reduce Human Effort	Preprocessing should take minutes, not days or weeks
Eliminate Lost Particles	Decrease lost particle fraction from 1/20,000 to zero

Four New Features



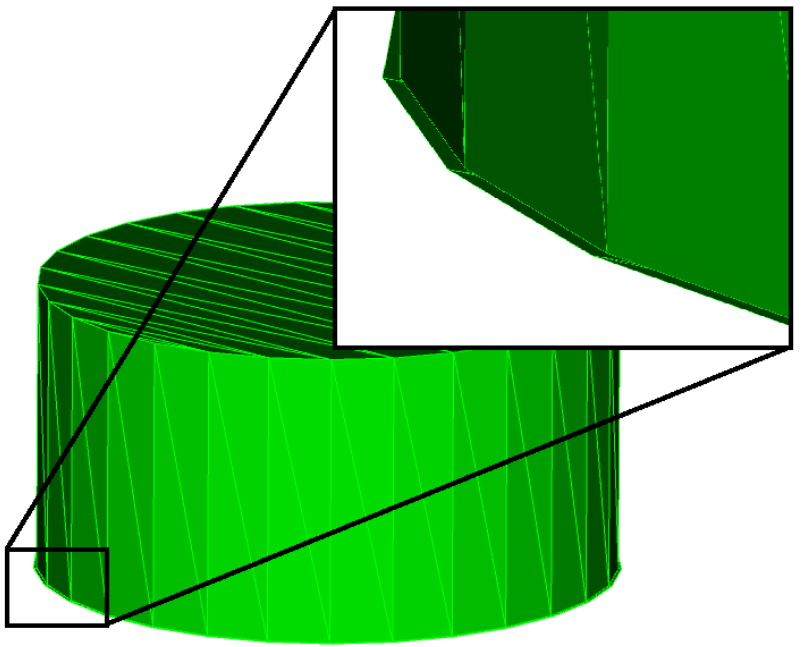
Accuracy Improvements

Watertight Faceting	Seal gaps between faceted surfaces
Robust Tracking	Develop new algorithm with goal of no lost particles

Human Efficiency Improvements

Implicit Complement	Create nonsolid space without manual CAD manipulation
Overlap Tolerance	Analyze imperfect CAD models without manual repair

Watertight Faceting



- Faceted representations used for efficient geometric queries
- Solid modeling engines typically facet each surface independently
- Faceted boundaries of neighboring surfaces are not the same, allowing gaps
- Particles can escape through gaps and become lost
- To prevent particles from becoming lost, a watertight faceting is needed

Algorithm Requirements

Problem Statement: Create algorithm to fix faceting flaws between surfaces.

1. Seal faceted surfaces along curves to create a watertight model.
2. To preserve human efficiency, the algorithm must be automatic.
3. New facets must be owned by exactly one surface.
4. Support non-manifold surfaces.
5. Fast enough to use as a preprocessing module.
6. Deformation of input model should be minimized, if possible.
7. Creation of new triangles should be minimized, if possible.

Contribution: Increase robustness by using topology of curves to develop a provably reliable algorithm implemented as open-source software.

Assumptions

Geometric Tolerance ε_g	Distance below which two entities are considered the same
Faceting Tolerance ε_f	Maximum distance between faceted entity and geometric entity it resolves

Cell Complex

- Geometric model is a cell complex
- Individual faceted curves and surfaces are a cell complex

Faceting

- Each geometric curve and surface has a corresponding faceted entity
- Local feature size $\gg \varepsilon_f \gg \varepsilon_g$
- Facet points are within ε_g of corresponding geometric entities
- Facet edges and triangles are within ε_f of corresponding geometric entities

Surface Boundary vs. Curves

- Each faceted surface boundary corresponds to a set of faceted curves
- Points on faceted surface boundary are within ε_g of *some* curve that bounds the surface, though *which* bounding curve is not known
- The elements of faceted curves are not the same as the boundary of the elements of the faceted surfaces

Algorithm Overview

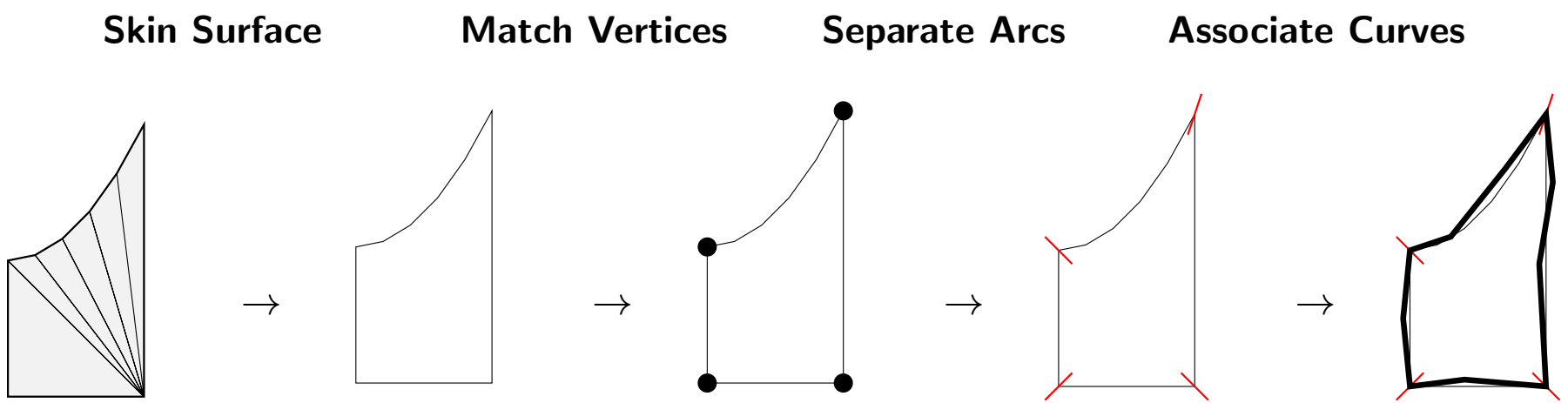
1. Import facet-based model from solid modeling engine
2. `seal_surface`
   - `seal_arcs` to corresponding curves
3. Export watertight facet-based model to application/library

- Algorithm operates only on facet-based model; not solid model
- Implemented as open-source algorithm in MeshKit

  `http://trac.mcs.anl.gov/projects/fathom/wiki/MeshKit`
- See paper for algorithm, outline of proof, and detailed results:
  B.M. Smith, T.J. Tautges, and P.P.H. Wilson, *Sealing Faceted Surfaces to Achieve Watertight CAD Models*, Proceedings of 19th International Meshing Roundtable, Chattanooga, TN, October 3-6, 2010.

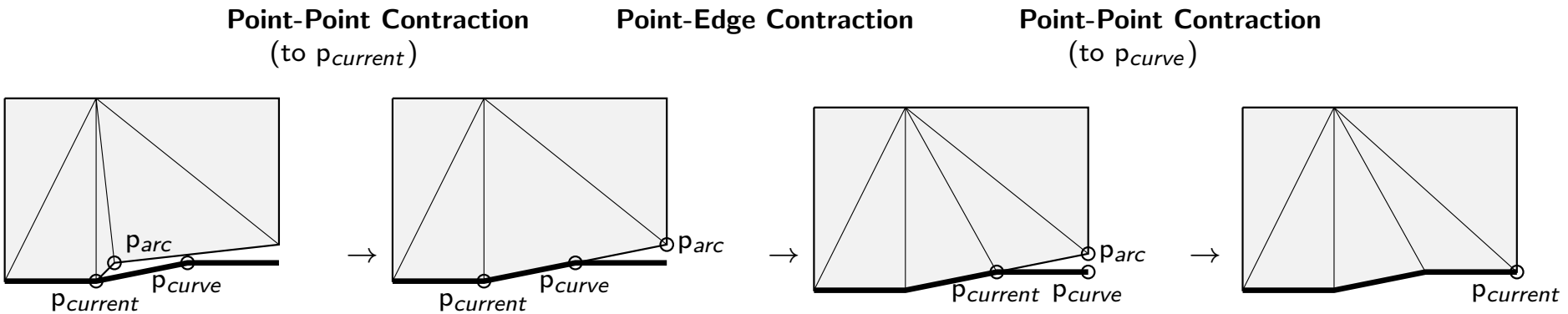
seal_surfaces

1. Skin surface to recover bounding edges
2. Orient bounding edges
3. Assembled bounding edges into loops
4. Match vertex points to points on loops, using proximity
5. Separate bounding loops into arcs, using vertex points as separators
6. Associate arcs with corresponding curves that bound the Face
7. If curve has not yet been sealed, replace curve with arc
   New since prelim: this reduces number of constraints on curve by 1
8. Otherwise `seal_arcs` to corresponding curves that bound the surface



seal_arcs

Initialize $p_{current}$, p_{curve}, and p_{arc}
Until arc is sealed:
- $p_{next} = p_{curve}$ or p_{arc} s.t. $d(p_{current}, p_{next})$=min
- If $d(p_{current}, p_{next}) \leq \varepsilon_f$ — Point-Point Contraction
  - contract p_{next} to $p_{current}$
- Else if $d(p_{curve}, p_{arc}) \leq \varepsilon_f$ — Point-Point Contraction
  - contract p_{arc} to p_{curve}
- Else — Point-Edge Contraction
  - insert p_{next} into opposite edge
- Update adjacencies, remove degeneracies
- Update $p_{current}$, p_{curve}, and p_{arc} as needed



Testing and Analysis

10 CAD Models Failure rate, triangle count, timing, and lost particles

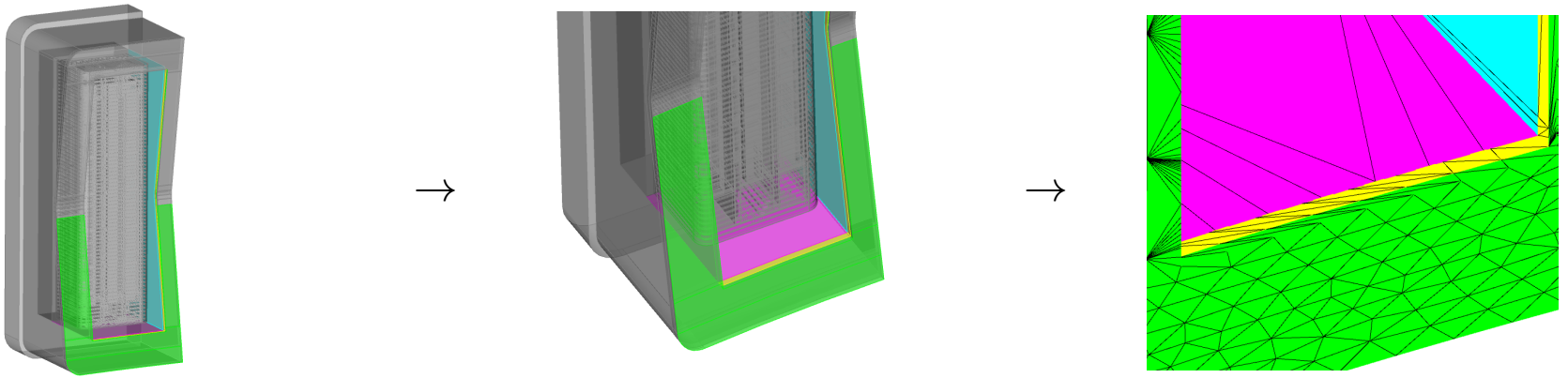
Entity Count

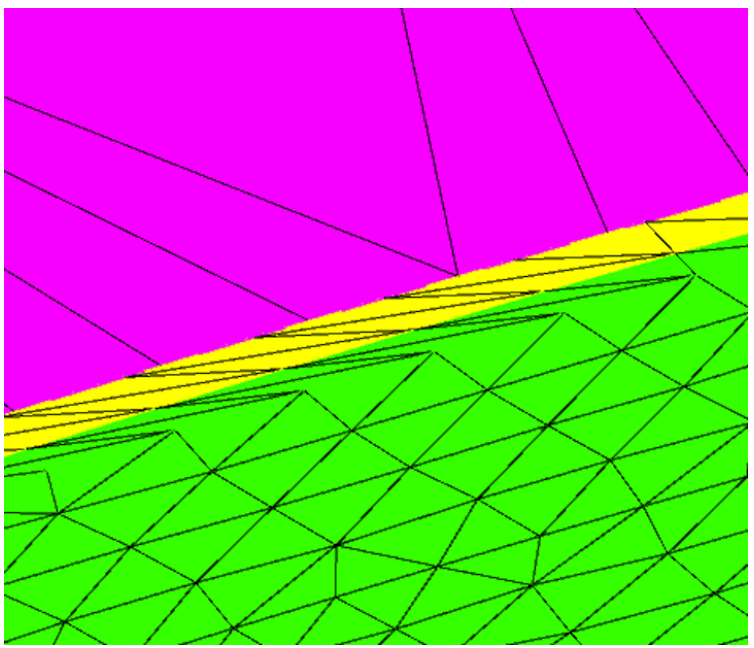
Table: Geometric entity count and number of triangular facets [millions] as a function of ε_f.

Model	Geometric Entity			Facet Tolerance [μm]				
	Volumes	Surfaces	Curves	1000	100	10	1	0.1
UW Nuclear Reactor	2820	30237	65078	2.62	2.62	2.98	8.56	29.1
Advanced Test Reactor	2132	11827	22402	0.44	0.45	0.84	2.44	7.65
40° ITER Benchmark	902	9834	20485	0.32	0.78	2.07	8.76	16.3
ITER Test Blanket Module	71	4870	13625	0.07	0.08	0.12	0.38	1.57
ITER Module 4	155	4155	10255	0.29	0.29	0.34	1.07	2.89
ITER Module 13	146	2407	5553	0.28	0.29	0.50	2.54	8.65
FNG Fusion Benchmark	1162	4291	5134	0.11	0.11	0.14	0.46	1.14
ARIES First Wall	3	358	743	0.17	0.87	1.21	1.55	2.45
High Average Power Laser	15	139	272	0.15	0.47	0.53	0.61	0.88
Z-Pinch Fusion Reactor	24	95	143	0.05	0.29	0.99	1.17	1.53

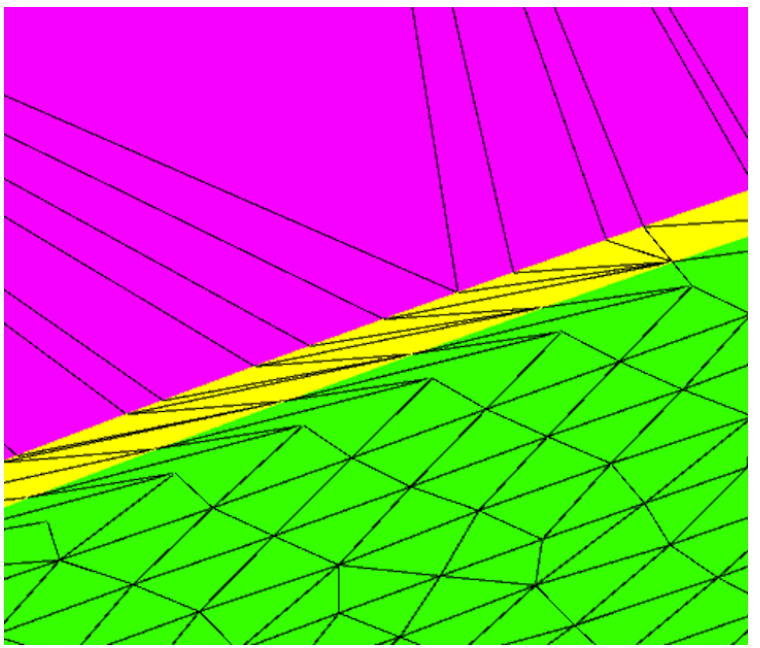
Defaults: $\varepsilon_f = 10\ \mu$m, $\varepsilon_g = 5\ \mu$m

Example: ITER Test Blanket Module



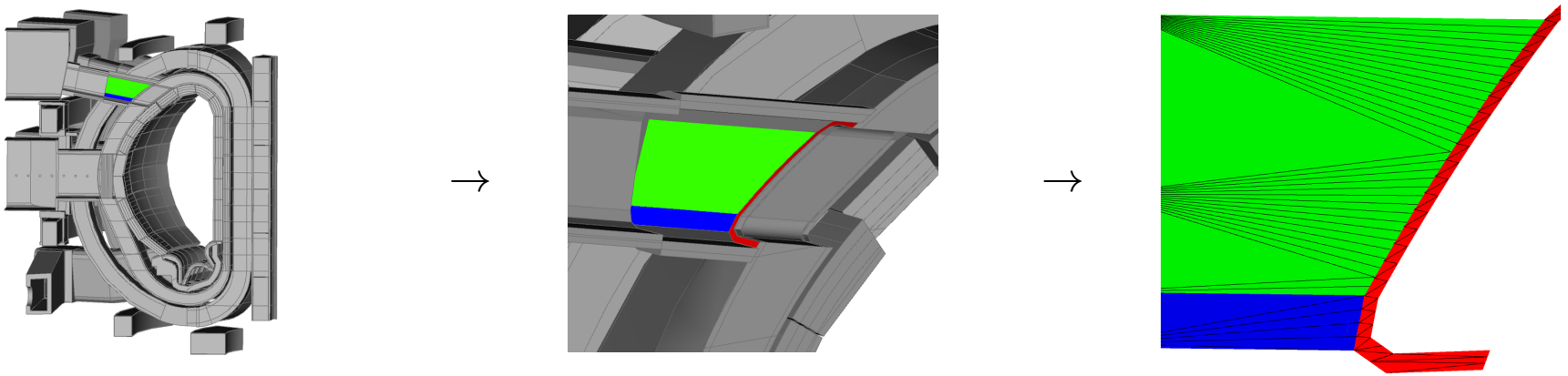


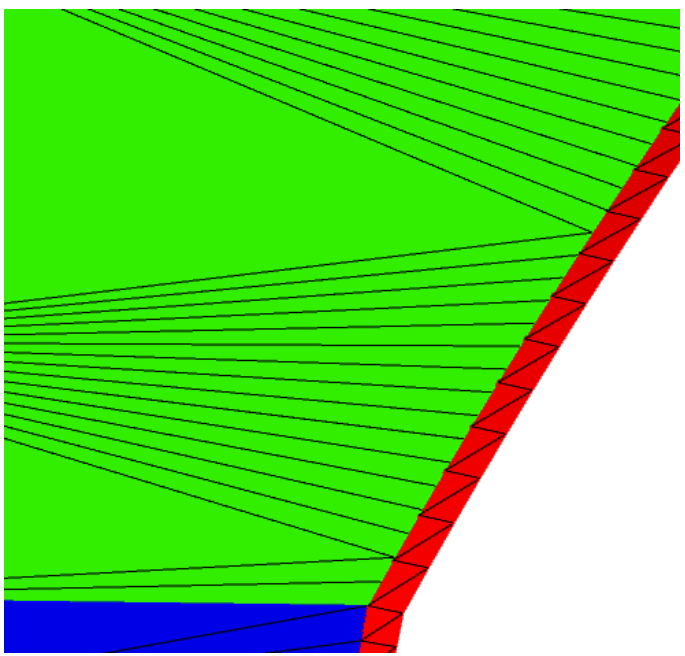
Unsealed



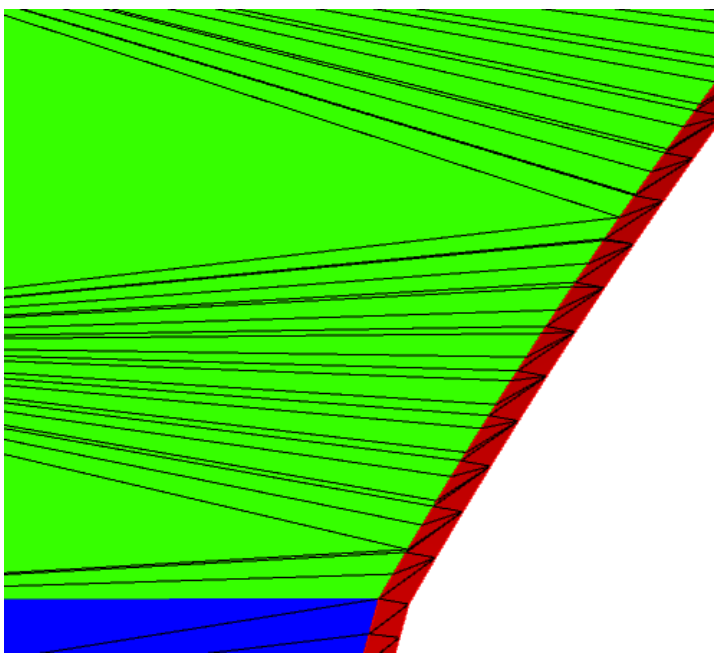
Sealed

Example: ITER 40° Benchmark Model



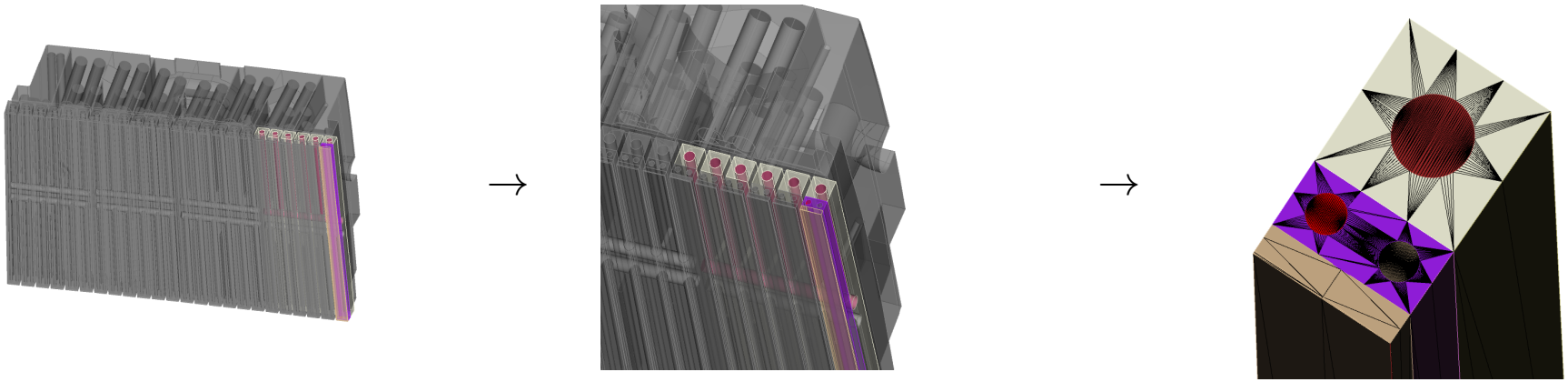


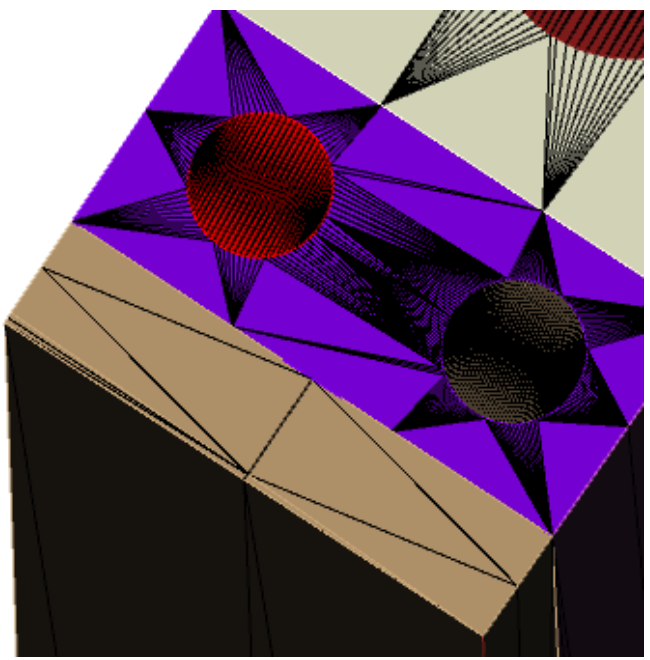
Unsealed



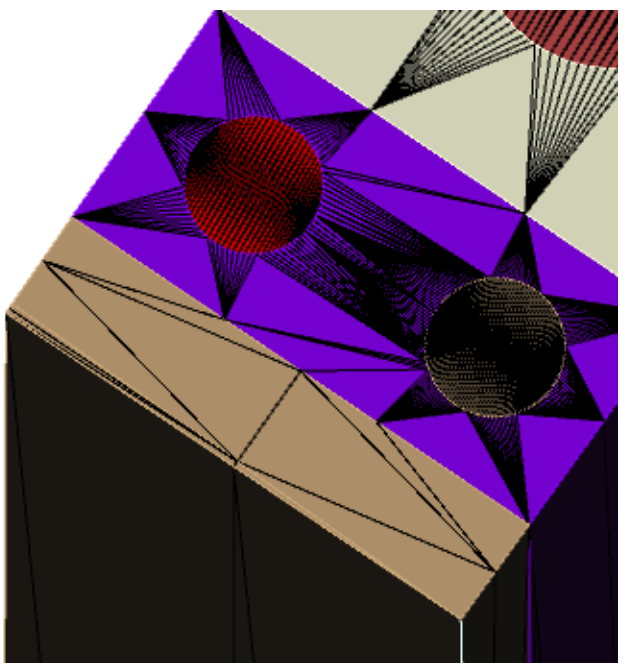
Sealed

Example: ITER First Wall/Shield Module 13





Unsealed



Sealed

Surface Sealing Failures

Table: Number of surface sealing failures as a function of ε_f.

Model	Facet Tolerance [μm]				
	1000	100	10	1	0.1
UW Nuclear Reactor	1019	0	0	0	0
Advanced Test Reactor	88	0	0	0	0
40° ITER Benchmark	18	9	0	18	191
ITER Test Blanket Module	0	0	0	0	0
ITER Module 4	0	0	0	0	0
ITER Module 13	2	0	0	0	0
FNG Fusion Benchmark	63	0	0	0	0
ARIES First Wall	1	0	0	0	0
High Average Power Laser	0	0	0	0	0
Z-Pinch Fusion Reactor	3	0	0	0	0

Failures occur when assumptions are not true:

$\varepsilon_f \not< LFS$ or $\varepsilon_f \not> \varepsilon_g$.

Triangle Count

Table: The change ratio [*sealed*/*unsealed*] in the number of facets due to sealing as a function of ε_f.

Model	Facet Tolerance [μm]				
	1000	100	10	1	0.1
UW Nuclear Reactor	0.71	0.99	1.00	1.00	1.01
Advanced Test Reactor	0.64	1.00	1.00	1.00	1.00
40° ITER Benchmark	1.00	1.01	1.02	1.02	1.03
ITER Test Blanket Module	0.90	1.00	1.01	1.01	1.01
ITER Module 4	0.65	0.98	1.00	1.01	1.01
ITER Module 13	0.78	1.00	1.00	1.00	1.00
FNG Fusion Benchmark	0.60	1.00	1.00	1.00	1.00
ARIES First Wall	1.00	1.00	1.00	1.00	1.00
High Average Power Laser	1.00	1.00	1.00	1.00	1.00
Z-Pinch Fusion Reactor	0.87	1.00	1.00	1.00	1.00

Preliminary Report Improvement: Lowered ratios from $\sim$3 to $\sim$1, preserving ray-tracing efficiency

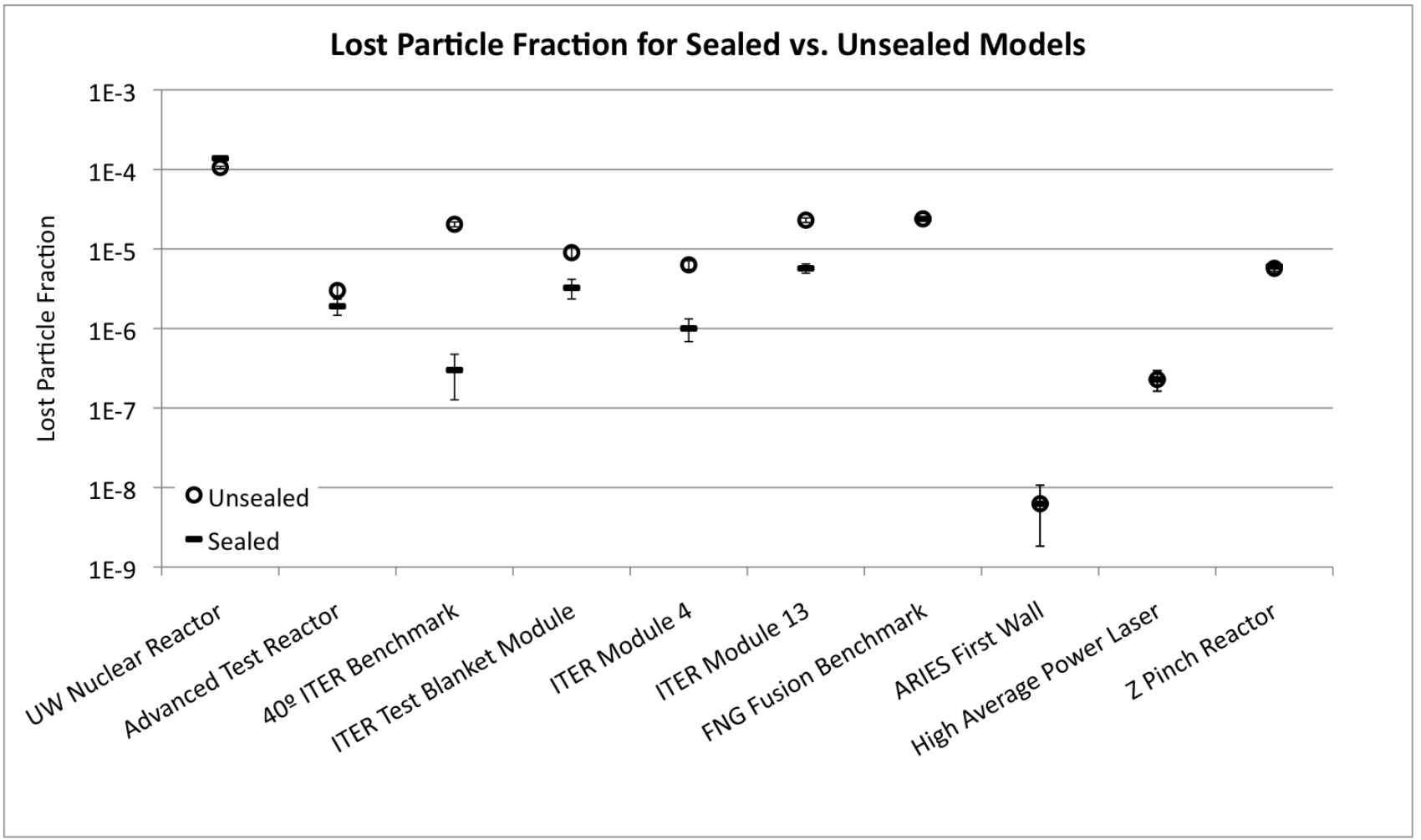
Timing

Table: The time [seconds] to seal each model as a function of ε_f, on one core of an Intel Xeon 3.00 GHz CPU.

Model	Facet Tolerance [μm]				
	1000	100	10	1	0.1
UW Nuclear Reactor	136	65	64	156	587
Advanced Test Reactor	93	16	27	76	235
40° ITER Benchmark	6	12	38	71	236
ITER Test Blanket Module	15	9	9	14	30
ITER Module 4	10	8	8	23	67
ITER Module 13	6	5	6	19	67
FNG Fusion Benchmark	7	4	4	9	29
ARIES First Wall	1	3	5	13	36
High Average Power Laser	1	1	2	5	25
Z-Pinch Fusion Reactor	1	1	2	4	12

Preliminary Report Improvement: Reduced time by ∼50%

Lost Particles



Leakage through unsealed surfaces is one cause of lost particles.

Robust Tracking

Lost particles still exist, despite watertight faceting
Most lost particles are due to a specific defect in tracking algorithm

Solution Approach

1. Consistent Ray-Triangle Intersection
2. Post-Process Edge/Point Intersections
3. Robust Point Inclusion Test
4. Zero-Distance Advance
5. Previous Facets
6. Particle Tracking

Assumptions

- Boundary of each volume is a pseudo 2-manifold.
- Faceting is watertight, oriented, non-degenerate, and non-inverted.

Terminology

RTI Ray-triangle intersection returned from ray-triangle test

Exit RTI returned to physics code from geometry library

Orientation Direction of ray with respect to surface normal is forward or reverse
Define: $\alpha = \cos(\text{angle between ray and surface normal})$
Convention: normal is outward

Distance Position of RTI along ray with respect to origin is negative or nonnegative

Two Notions of Particle Position

Logical The volume that the particle is inside

Numerical The spatial coordinates of the particle. May become inconsistent with logical position due to rounding error of $\vec{x'} \approx \vec{x} + d\vec{u}$

Categorize Failure Modes

Logical Caused by faulty or incomplete algorithm

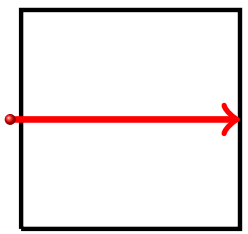
Numerical Caused by finite precision arithmetic

Problem Statement: Track particles through a faceted CAD model without becoming lost or entering a pseudo-infinite loop.

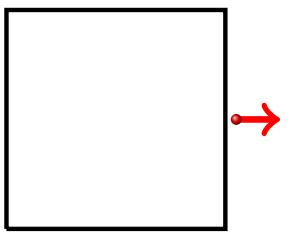
Failure Modes

Determine exit intersection from volume:

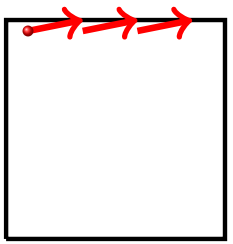
- Behind previous surface (numerical)



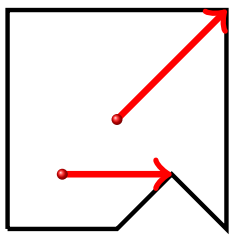
- Ahead of next surface (numerical)



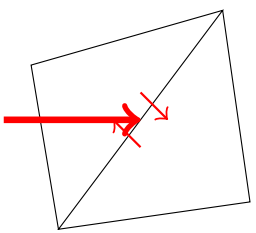
- Tangent to a surface (numerical)



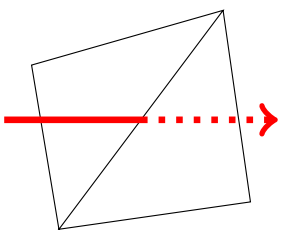
- Intersects edge or point (logical)



- Oscillation between triangles (logical)



- Leak between triangles (numerical)



Previous Work

Original DAG-MCNP Algorithm
Discard Distance Tolerance ε_d

- Discard intersection if closer than ε_d
- Prevents intersecting the same facet, tangential skipping along surface, and oscillation between surfaces
- Guarantees failure if correct intersection $< \varepsilon_d$

Electron Gamma Shower [Bielajew, 1995]

- Users write their own geometry routines
- Add extra track length to each intersection distance

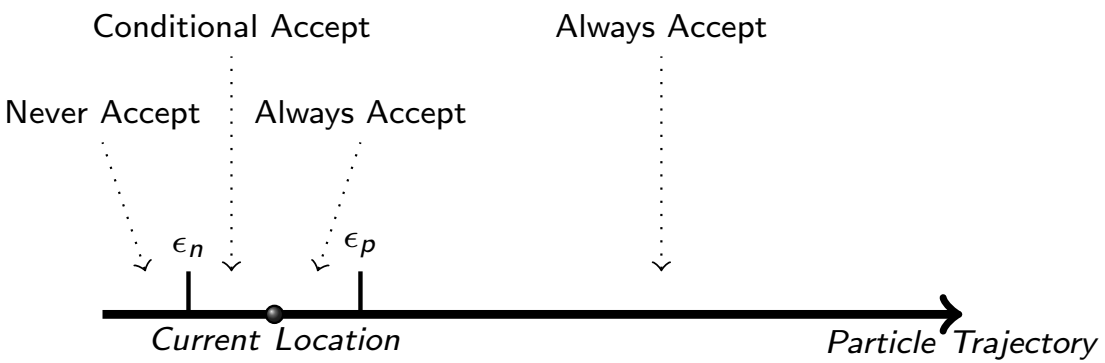
PTSIM [Popescu, 2003]

MCNP [Girard, 2008]
Suggests Tools:

- Ray direction vs. surface normal
- Previous surface, if known
- Ray never intersects planar surface twice

GEANT [Williams, 2010]

- Tolerance makes volume boundaries 3D
- Tolerance: increasing robustness vs. ambiguity

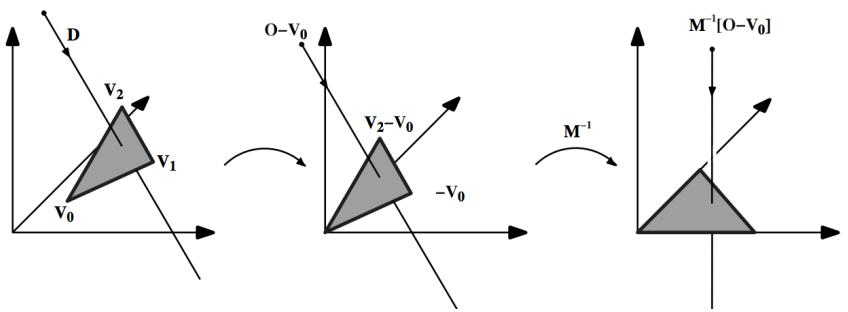


Ray-Triangle Intersection

Goal: Prevent leakage between triangles; identify edge and point intersections

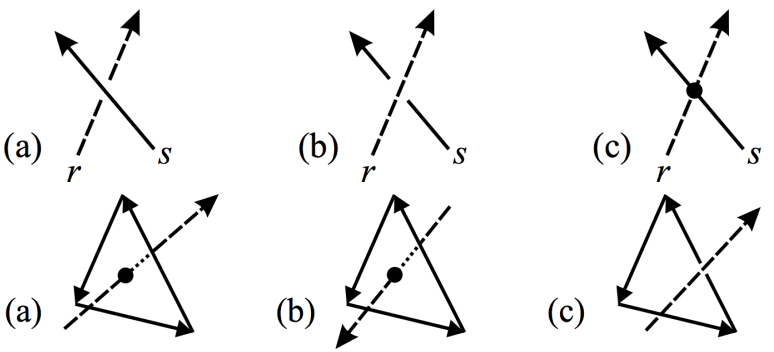
Möller Test [Möller and Trumbore, 2005]

- Matrix solution translates, scales to Barycentric
- Edge-Unstable: edge/ray intersection is not consistently performed for adjacent triangles
- Cannot consistently detect edge/point intersections
- Originally used in DAGMC



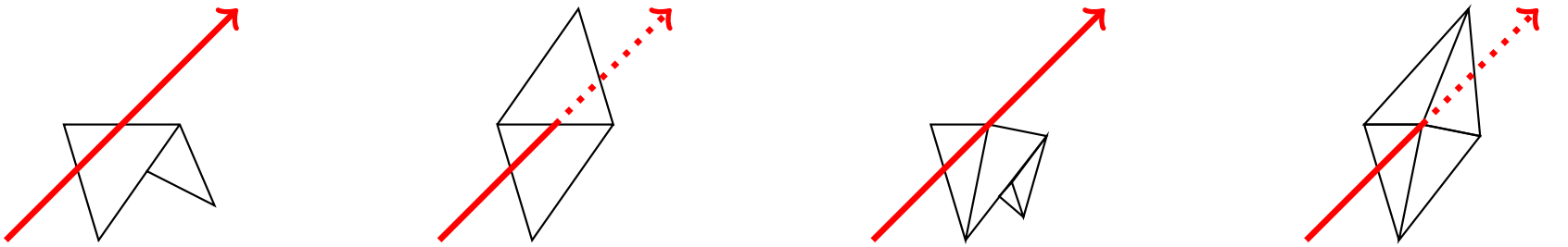
Plücker Test [Platis and Theoharis, 2003]

- Plücker coordinates are calculated from a point and a direction
- Edge-Stable: edge/ray intersection is consistently performed for adjacent triangles
- Consistently detects edge/point intersections
- Added to DAGMC in this work



Edge/Point Intersections

Goal: Determine if edge/point intersections are glancing or piercing



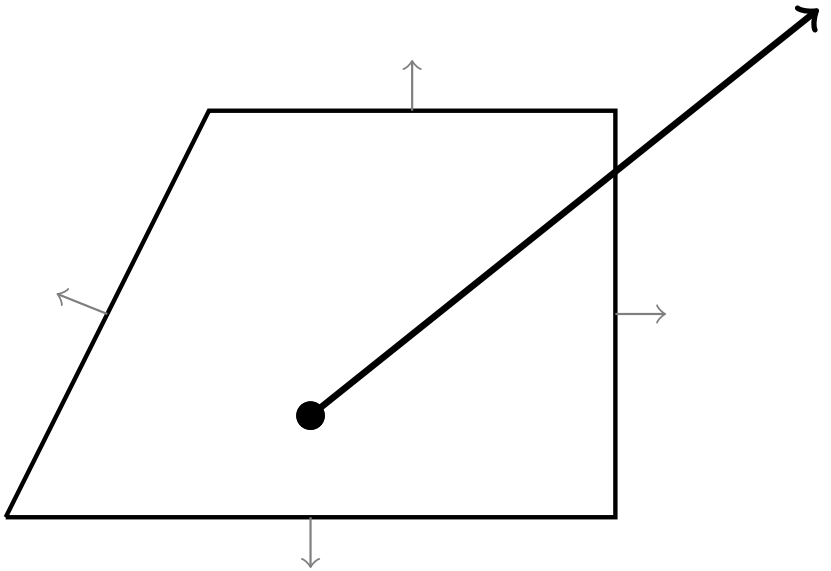
- If edge/point intersection, must investigate neighborhood
- Obtain neighborhood using sphere-triangle intersect, then downward adjacencies
- Compare ray direction and triangle normal vectors
- If piercing, `sign(`α`)=constant` for all triangles in neighborhood
- Glancing intersections are rejected because particle does not exit volume

Contribution: Edge/point intersections were previous failure mechanisms

Point Inclusion Test

Goal: Determine if particle is inside volume

- Combine ray intersection and enlarged orientation methods→*Ray Intersection Orientation Method*
- Compare surface normal with ray direction to determine entrance/exit



Edge/Point Intersections	Use only *piercing* intersections
Consistency	Use same numerical operation as particle tracking.
No Tolerances	*on_boundary* avoided by storing previous facets
Bonus	Plücker Test and edge/point intersections already implemented

Contribution: Compared with DAGMC's original test, increased robustness

Zero-Distance Advance

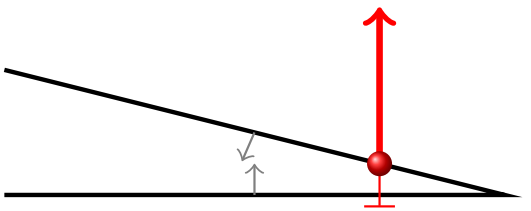
Goal: Ensure consistency between logical and numerical position

- `next_surface` and `point_inclusion` are guaranteed to be consistent
- Logical and numerical positions become inconsistent due to particle advance: $\vec{x'} \approx \vec{x} + d\vec{u}$
- Correct exit intersection may occur at negative distance
- Avoid passing negative track length to physics code: instead use zero
- Reestablish consistency by advancing logical position without changing numerical position

Contribution: Avoid lost particles due to inconsistency of position

Previous Facets

Goal: Avoid infinite loops



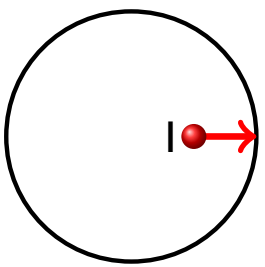
- Due to numerical error, exit intersections can occur behind numerical position
- Zero-distance advance allows for oscillation to occur if $LFS \ll negative\_ray\_length$
- To prevent oscillation, store intersected facets along streaming path

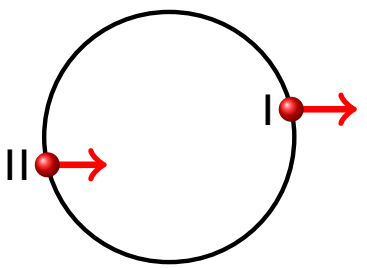
Contribution: Avoid infinite loops by storing previous facets

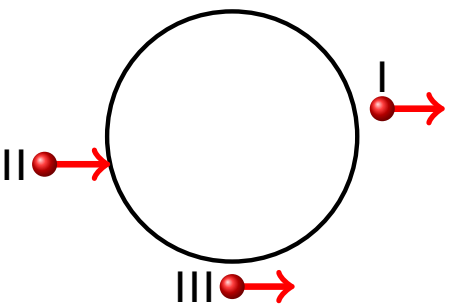
Enumerate Possible RTIs as f(numerical position,α)

Examine first intersection with ray

- α = cosine(angle between ray and surface normal)
- Only *piercing* edge/point RTIs are returned
- Note parity of RTI orientation along ray (forward, reverse, forward, ...)







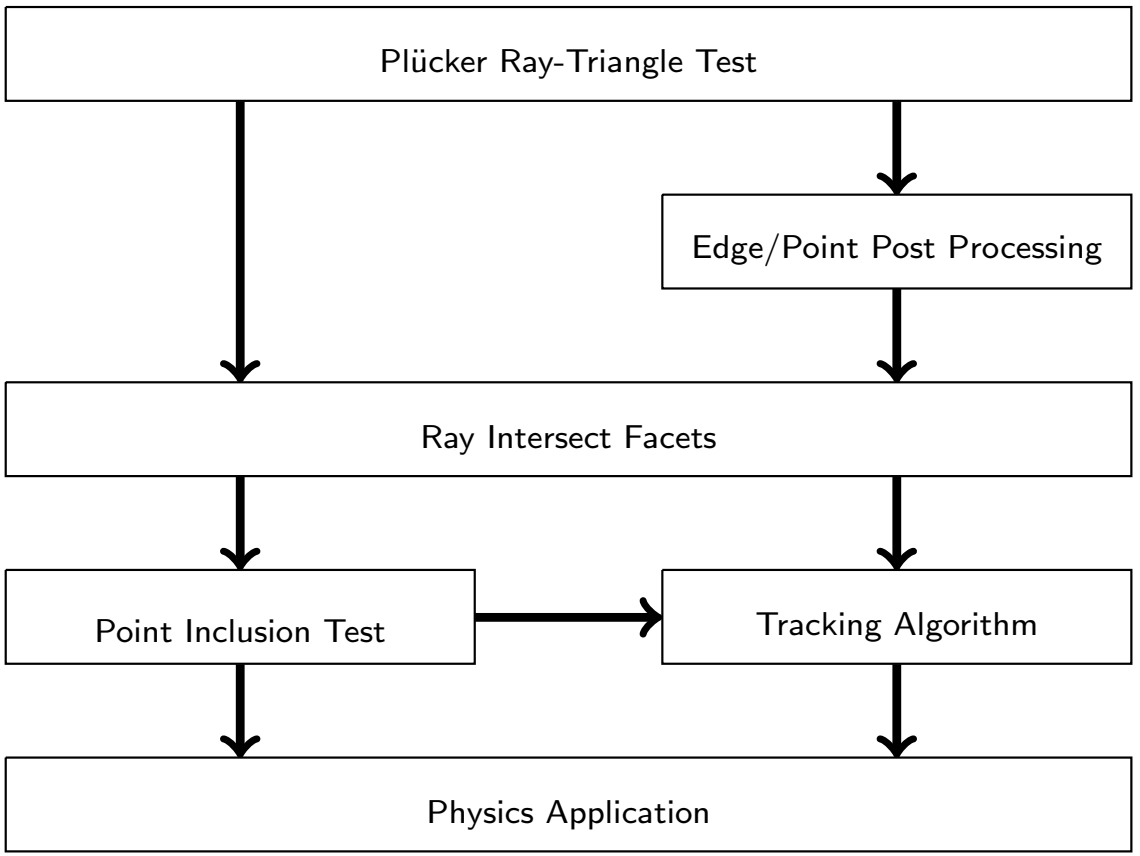
	Inside	On Boundary	Outside
I) $\alpha > 0$	Typical-Return this intersection	Rare-Return this intersection	Rare-Exit is behind numerical position
II) $\alpha < 0$	Cannot occur	Typical-Do not return this intersection (previous exit)	Typical-Do not return this intersection (previous exit)
III) null	Cannot occur	Cannot occur	Rare-Exit is beside numerical position

Tracking algorithm (next slide) will only return piercing RTIs with $\alpha > 0$

Tracking Algorithm

```
next_surface( prev_surf, prev_facet, ray_pt, ray_dir, volume, physics_limit,
              &next_dist, &next_surf, &prev_facets )

// Clear prev_facets as needed.
if     ( reflecting ) prev_facets.erase(begin,end-1)
else if( !streaming ) prev_facets.erase(begin,end)

// Set distance limits of the intersection search.
nonneg_dist_limit =  HUGE_VAL
neg_dist_limit    = -NUM_PRECISION
if( NULL != physics_limit )             nonneg_dist_limit =  physics_limit
if nonneg_dist_limit < -neg_dist_limit ) nonneg_dist_limit = -neg_dist_limit

// Return piercing exit intersections subject to limits and prev_facets.
call ray_intersect_facets( volume, ray_pt, ray_dir, prev_facets, neg_dist_limit,
                           nonneg_dist_limit, &surfs, &dists, &facets )

// Is the RTI at negative distance inside the next volume?
if( NULL != facets[0] )
     next_vol = get_next_volume( surfs[0], volume )
     call point_inclusion( ray_pt, ray_dir, next_vol, prev_facets, &result )
     if( INSIDE == result ) next_dist = 0; next_surf = surfs[0];
          prev_facets.push_back( facets[0] ); return SUCCESS

// Return RTI at positive distance if it exists.
if( NULL != facets[1] ) next_dist = dists[1]; next_surf = surfs[1];
     prev_facets.push_back( facets[1] ); return SUCCESS

// If using physics_limit, assume a collision occurs before an exit intersection.
if( NULL != physics_limit ) next_dist = physics_limit+1; next_surf = 0; return SUCCESS

// Otherwise the particle is lost.
next_dist = HUGE_VAL; next_surf = 0; return FAILURE
```

Implementation



- OBB tree traversal occurs in `ray_intersect_facets`
- Plücker test orders edge endpoints by handle for consistency
- Only 1 tolerance: `NEG_DIST_LIMIT` defaults to 10 μm

Proof Approach

Particles Cannot Become Lost — All enumerated cases handled, except *Outside-III*

Infinite Loops Cannot Occur — RTIs are not reused along a streaming event

Testing and Analysis

DAGMC Test Suite	Passed with statistical differences, no lost particles
ITER 40° Benchmark Model	Tracking rate, tallies, collision count, random number count, lost particle
10 CAD Models	300 computer-day search for lost particles

ITER 40° Benchmark Model

Table: Particle tracking of the original and robust versions of DAG-MCNP is compared using the 40° ITER benchmark model.

Case	Executable	Physics Limit	Tracking Rate [part./min.]	Tallies	Collisions [#]	Random Numbers [#]	Lost Particles [#]
			with materials				
1	original	no	9766	*baseline*	12701310	185958957	0
2	original	yes	10299	identical	12701310	185958957	0
3	robust	no	9976	identical	12701310	185958957	0
4	robust	yes	11339	identical	12701310	185958957	0
			without materials				
5	original	no	74674	*baseline*	0	754270	2
6	robust	no	81225	identical	0	754270	0

- ITER model selected due to its complexity and frequent use
- Each case: 100k particle histories on Intel Core2 2.66 GHz CPU
- Robust algorithm is 2-10% faster due to distance limit implementation
- Physics limit (16% faster) can now be used without masking lost particles

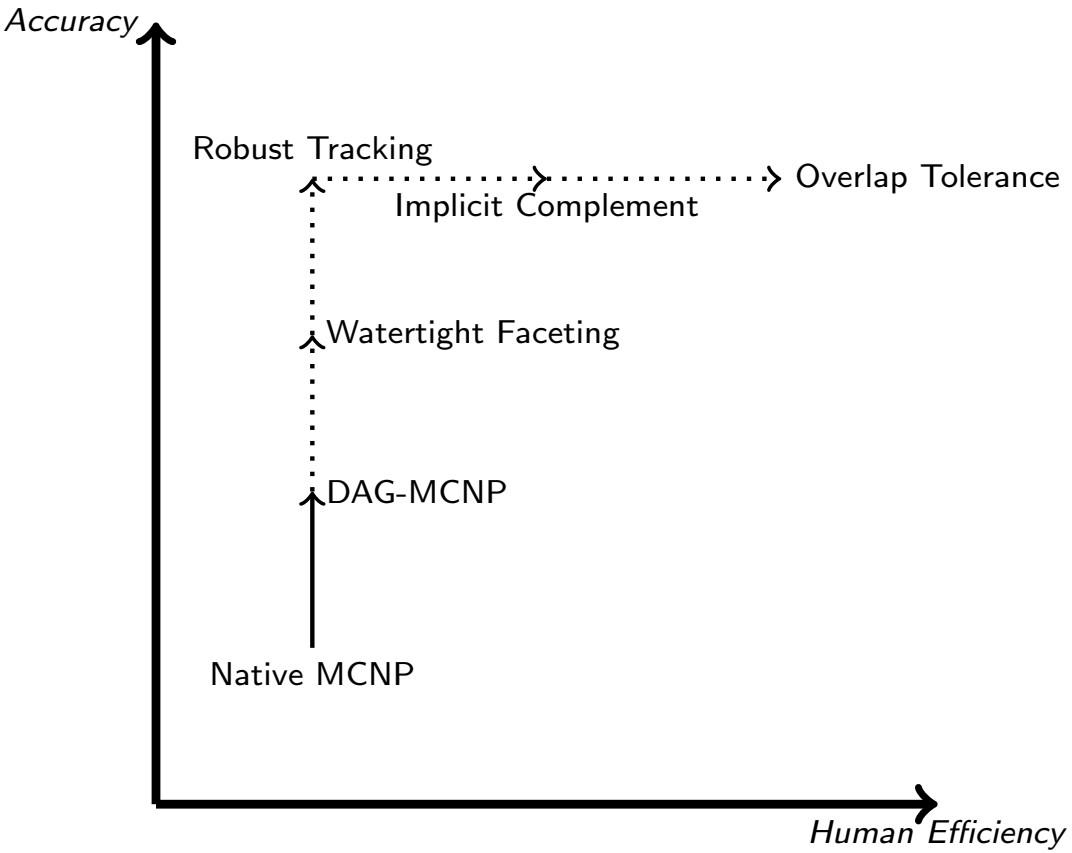
10 CAD Models

Table: The number of lost particles for watertight models using the original and robust tracking algorithms with $\epsilon_f = 10\mu m$. Error range indicates one standard deviation.

Model	Particles Simulated [millions]	Particles Lost	
		Original Algorithm	Robust Algorithm
UW Nuclear Reactor	41	5649 ± 178	0
Advanced Test Reactor	74	141 ± 32	0
40° ITER Benchmark	225	67 ± 39	0
ITER Test Blanket Module	205	665 ± 184	0
ITER Module 4	59	59 ± 19	0
ITER Module 13	79	450 ± 60	0
FNG Fusion Benchmark	1310	31273 ± 989	0
ARIES First Wall	4070	25 ± 18	0
High Average Power Laser	286	65 ± 19	0
Z-Pinch Fusion Reactor	409	2454 ± 317	0

- Same models from Watertight Faceting section
- Each model: 30 days on Intel Xeon 3.00 GHz CPU
- Algorithm forms basis for overlap-tolerant tracking

Change Topics: Accuracy→Human Efficiency



Accuracy Improvements

Watertight Faceting	Seal gaps between faceted surfaces
Robust Tracking	Develop new algorithm with goal of no lost particles

Human Efficiency Improvements

Implicit Complement	Create nonsolid space without manual CAD manipulation
Overlap Tolerance	Analyze imperfect CAD models without manual repair

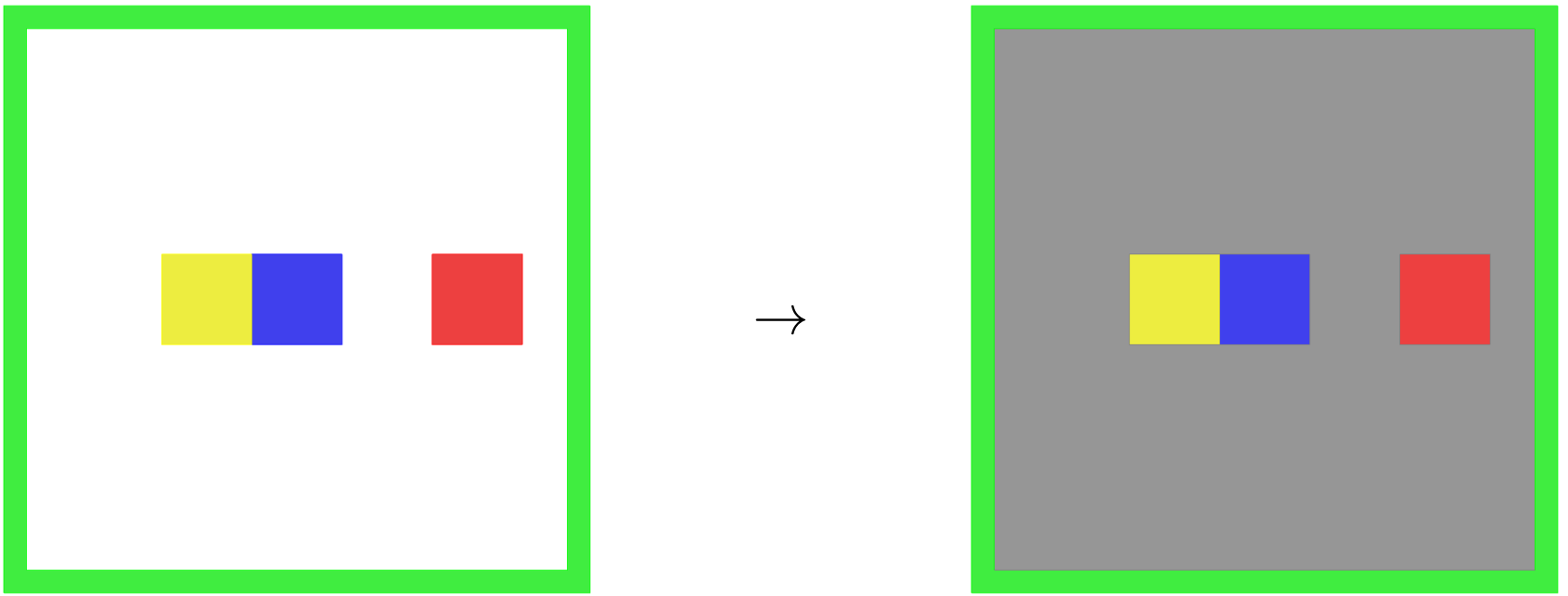
Implicit Complement

Motivation

- Nonsolid space, or *complement* is not explicitly represented in CAD models
  - Air in room, coolant in reactor, vacuum in experiment
- All 3D space must be defined for MC transport
- Creating explicit complement in CAD software is error-prone
- Imprinting/merging explicit complement is difficult
  - Surfaces are not typically shared in CAD models
  - For acceleration, MCNP expects a single surface shared between adjacent volumes
  - Imprinting subdivides coincident surfaces so that they share the same topology
  - Merging replaces two surfaces with a single, shared surface
- Avoiding creation of explicit complement with subsequent imprint/merge increases human efficiency

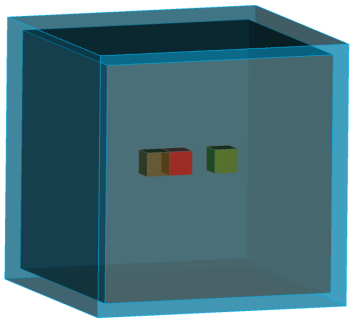
Algorithm

Problem Statement: Build an implicit or pseudo-complement of all unmerged surfaces

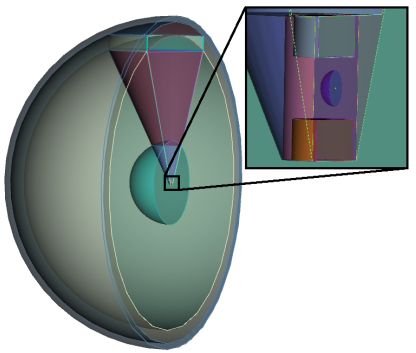


```
create a volume object for the implicit complement
for all surfaces
  get parent volumes of surface
  if surface is contained in only one volume
    add surface to implicit complement volume
```

Testing and Analysis



Cubes



Z-Pinch Fusion Reactor

ITER FWS Module 13

Table: Comparison of DAG-MCNP simulation results using explicit vs. implicit nonsolid space.

Model	Tallies	Random Number Count	Collision Count
Cubes	identical	identical	identical
Z-Pinch Fusion Reactor	identical	within 0.001%	within 0.001%
ITER Module 13	identical	identical	identical

- Difference in results due to manifold representation in solid modeling engine
  - Implicit→1 surface, Explicit→2 surfaces (but only 1 used)
  - Explicit complement may not use same surface as implicit complement
- Implicit complement reduces human effort by days
- Gaps between volumes are now automatically defined, but what about overlaps?

Overlap Tolerance

- Monte Carlo packages lose all particles that encounter overlaps
- Overlaps due to imprecise draftsmanship, file translation, structural deformation of mesh
- Remedy overlaps by improving draftsmanship, avoiding translation, manual repair, change tracking algorithm

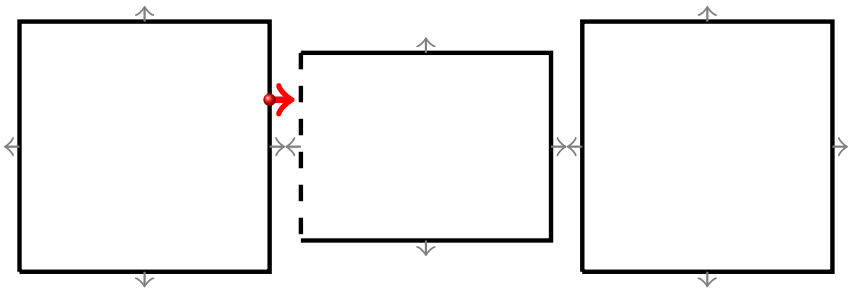
Assumption	Overlaps are small enough to not significantly affect physics
Implies...	Particles may travel through either overlapping material

- Avoiding manual CAD repair will increase human efficiency & enable deformed mesh analysis

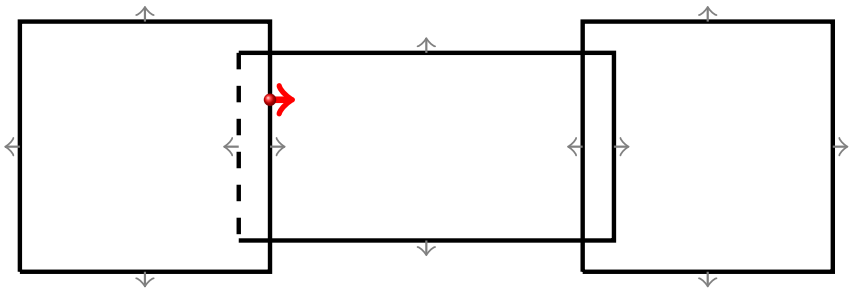
Problem Statement: Track particles through a faceted CAD model containing overlaps without becoming lost or entering a pseudo-infinite loop.

Exit May Be Behind Ray Origin

Goal: Discover RTIs behind particle's numerical position



No Overlap



Overlaps

- Overlaps of solid volumes appear as self intersections of the complement
- Search behind the ray origin to detect overlaps
- Correct orientation is necessary, but not sufficient
- If overlap, particle will be inside next volume
- Use zero-distance advance to make logical and numerical position consistent

Contribution: Overlap-tolerant tracking now structured as special case of robust tracking algorithm

Overlap-Tolerant Point Inclusion Test

Goal: Devise a PIT for overlapping geometry

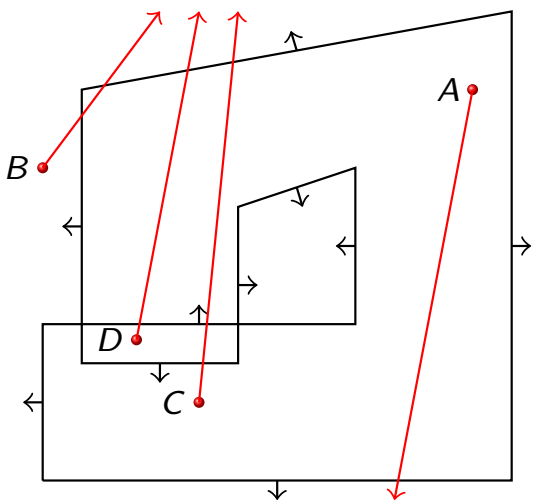


Table: Point inclusion test results for a self-intersecting volume.

Method	Point A	Point B	Point C	Point D
Enlarged Orientation	In	Out	(Out)	In
Winding Number	In	Out	In	In
Ray Intersection Parity	In	Out	In	(Out)
Ray Intersection Orientation *first intersection*	In	Out	(Out)	In
Ray Intersection Orientation *all intersections*	In	Out	In	In

- The exit/entrance of first intersection is unreliable, due to self intersections
- Instead examine all intersections along ray to ∞
- Sum entrance/exit intersections along ray
- *Inside* points will have at least one more exit than entrance

Contribution: Overlap-tolerant PIT now structured as special case of robust tracking algorithm

Tracking Algorithm

```
next_surface( prev_surf, prev_facet, ray_pt, ray_dir, volume, physics_limit,
              &next_dist, &next_surf, &prev_facets )

// Clear prev_facets as needed.
if     ( reflecting ) prev_facets.erase(begin,end-1)
else if( !streaming ) prev_facets.erase(begin,end)

// Set distance limits of the intersection search.
nonneg_dist_limit =  HUGE_VAL
neg_dist_limit    = -OVERLAP_THICKNESS
if( NULL != physics_limit )             nonneg_dist_limit =  physics_limit
if nonneg_dist_limit < -neg_dist_limit ) nonneg_dist_limit = -neg_dist_limit

// Return piercing exit intersections subject to limits and prev_facets.
call ray_intersect_facets( volume, ray_pt, ray_dir, prev_facets, neg_dist_limit,
                           nonneg_dist_limit, &surfs, &dists, &facets )

// Is the RTI at negative distance inside the next volume?
if( NULL != facets[0] )
     next_vol = get_next_volume( surfs[0], volume )
     call point_inclusion( ray_pt, ray_dir, next_vol, prev_facets, &result )
     if( INSIDE == result ) next_dist = 0; next_surf = surfs[0];
          prev_facets.push_back( facets[0] ); return SUCCESS

// Return RTI at positive distance if it exists.
if( NULL != facets[1] ) next_dist = dists[1]; next_surf = surfs[1];
     prev_facets.push_back( facets[1] ); return SUCCESS

// If using physics_limit, assume a collision occurs before an exit intersection.
if( NULL != physics_limit ) next_dist = physics_limit+1; next_surf = 0; return SUCCESS

// Otherwise the particle is lost.
next_dist = HUGE_VAL; next_surf = 0; return FAILURE
```

Testing and Analysis

ITER 40° Benchmark Model	Tracking rate, tallies, collision count, random number count, lost particle
11 Cubes	Tracking rate vs. percentage of unmerged surfaces
Space Reactor	k_{eff} of native MCNP vs. CAD geometry
Deformed Space Reactors	k_{eff} of deformed reactors

ITER 40° Benchmark Model

Table: Particle tracking of overlap-tolerant DAG-MCNP is compared using merged and unmerged versions of the 40° ITER benchmark model.

Case	Coincident Surfaces	Overlap Thickness [cm]	Physics Limit	Tracking Rate [part./min.]	Tallies	Collisions [#]	Random Numbers [#]
			with materials				
1	merged	0	no	9976	*baseline*	12701310	185958957
2	merged	10	no	9507	identical	12701310	185958957
3	unmerged	10	no	4703	identical	12701228	185958361
4	unmerged	10	yes	4851	identical	12701228	185958361
			without materials				
5	merged	0	no	81225	*baseline*	0	754270
6	merged	10	no	77527	identical	0	754270
7	unmerged	10	no	12382	identical	0	754270

- Each case: 100k particle histories on Intel Core2 2.66 GHz CPU
- Speed penalty for overlap-tolerant tracking is 5%
- Worst case is 53% slower (none merged) vs. best case (all merged)

11 Cubes

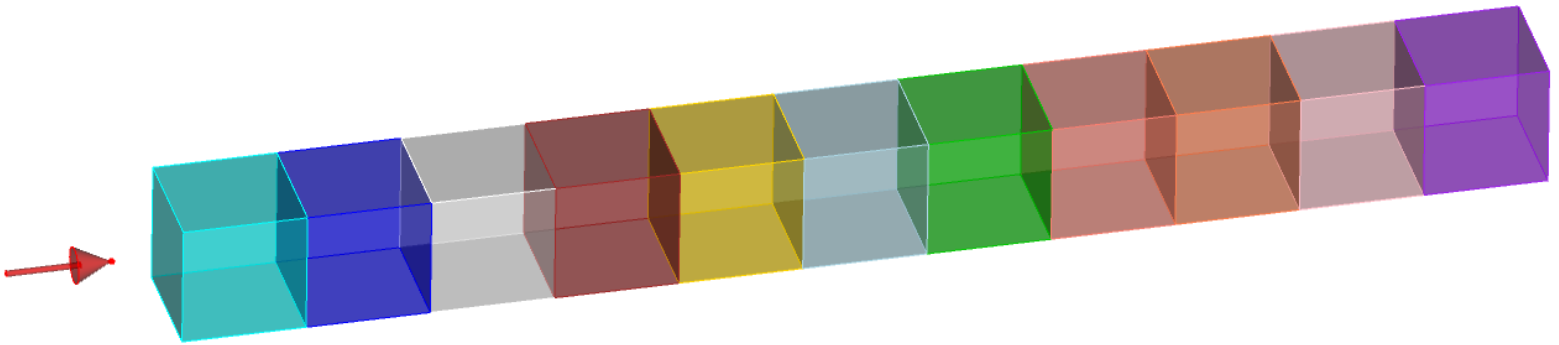
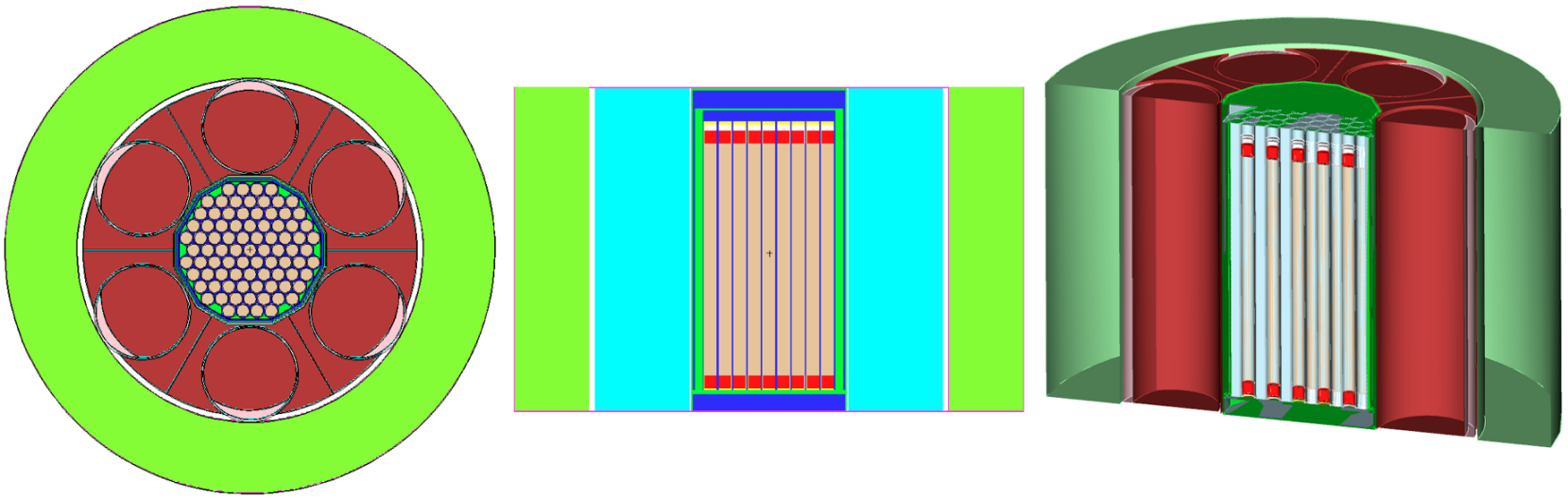


Table: Tracking rate as a function of the number of overlaps.

Case	Overlaps [#]	Overlap Thickness [cm]	Tracking Speed [part./min.]	Relative Speed [%]
1	0	0.0	946490	100
2	0	0.1	940790	99
3	2	0.1	591490	62
4	4	0.1	400950	42
5	6	0.1	328960	35
6	8	0.1	232940	25
7	10	0.1	249850	26

- It is likely that a small fraction of surfaces cannot easily be merged
- Performance gradually decays as more surfaces become unmerged

Space Reactor



- 85-pin space reactor with control drums rotated for minimum absorption [Marcille et al., 2006]
- Automated conversion of geometry, materials, and boundary conditions from native MCNP to ACIS
- Intentionally forced CAD model to have coincident, overlapping surfaces
- Used to validate overlap-tolerant logic before analyzing deformed geometry

Native MCNP5

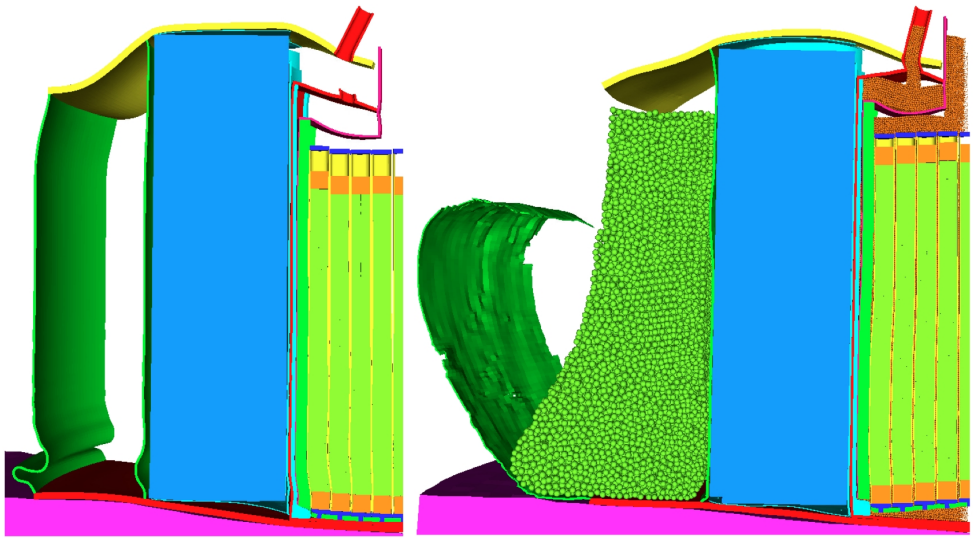
- Native geometry
- $k_{eff} = 1.01437\ (\pm 0.00075)$

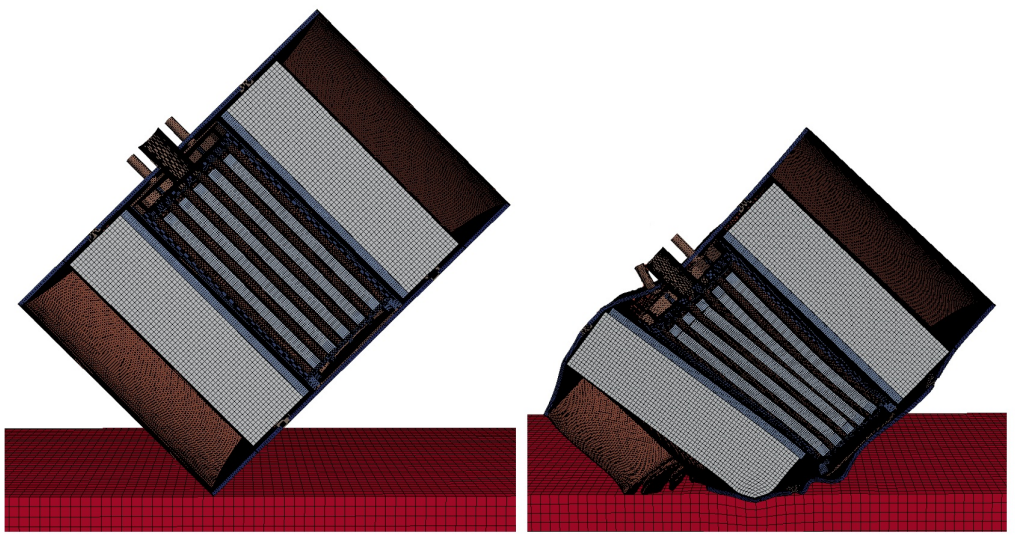
DAG-MCNP5

- CAD Geometry, $\varepsilon_f = 1\ \mu$m
- $k_{eff} = 1.01451\ (\pm 0.00080)$

Deformed Space Reactors

Goal: Predict reactivity change due to impact after launch accident



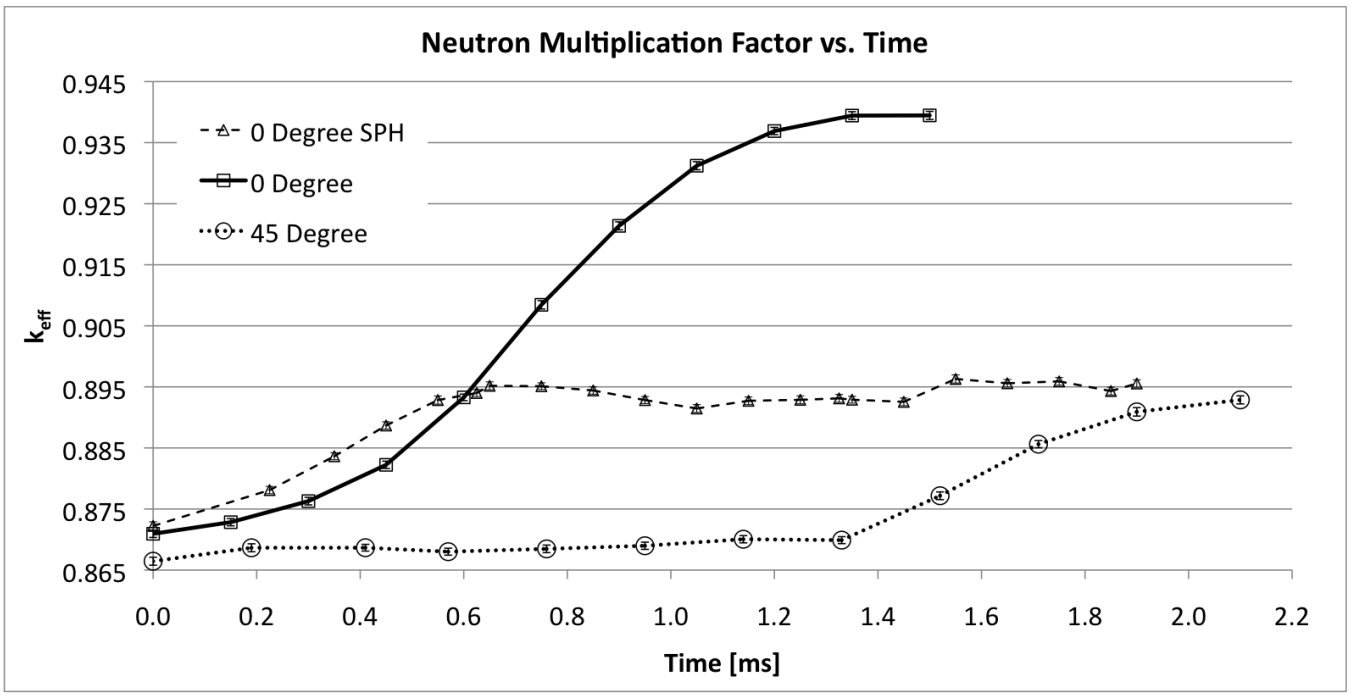


- Structural analysis performed by Villa et al. at SNL
- Concrete impact at 100 m/s
- Fluids modeled in 0 degree structural analysis using SPH particles
- Reactivity analysis at UW-Madison
- Hexahedral mesh converted to 11M triangles
- Challenges: overlaps, fracture, mass conservation
- Fluids not modeled in reactivity analysis
- Utilized implicit complement, robust tracking, overlap tolerance

Contribution: First known mesh-based reactivity analysis of deformed reactors

Deformed Space Reactors

- Each DAG-MCNP5 case required 5-7 hours on one core of 2.66 GHz Intel Core2
- Error bars (small) indicate 1 standard deviation



Discussion

- 45-degree simulation: k_{eff} did not increase until ∼1 ms when fuel pins moved relative to one another
- 0-degree SPH simulation: SPH elements limit contact of adjacent fuel pins→restrict increase in k_{eff} to 2.7%

B.M. Smith and P.P.H. Wilson, *Modeling Impact-Induced Reactivity Changes Using DAG-MCNP*, Proceedings of Nuclear and Emerging Technologies for Space—NETS2011, Albuquerque, NM, February 7-10, 2011.

Summary

Eliminated Lost Particles

- Sealed faceting to prevent particle leakage between surfaces
- Increased robustness of particle tracking
  - Ensure numerical consistency→ray tracing for tracking & PIT
  - Remove tolerances→use logic instead
- Most models lose zero particles

Eliminated Manual CAD Repair

- Nonsolid space is automatically defined, filling gaps
- Track particles through overlaps
- Instead of repairing CAD defects, ensure defects are reasonable
- Enables analysis of new geometry types (deformed mesh)
- CAD model preparation now takes minutes instead of days/weeks

Acknowledgments

- Prof. Paul Wilson, Prof. Tim Tautges, and Jason Kraftcheck
- Daniel Villa, Tyler Tallman, Jeffrey Smith, Ron Lipinski, Tracy Radel, and Ross Radel at Sandia National Laboratories
- Sandia National Laboratories' Lab Directed Research and Development Program
- US ITER Project through Sandia contracts 579323 and 866756

Questions?

Thank You

References I

Amenta, N., Bern, M., and Kamvysselis, M. (1998).
A New Voronoi-Based Surface Reconstruction Algorithm.
In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 415–421, New York, NY, USA. ACM.

Barequet, G. and Kumar, S. (1997).
Repairing CAD Models.
In *VIS '97: Proceedings of the 8th Conference on Visualization '97*, Los Alamitos, CA, USA. IEEE Computer Society Press.

Barequet, G. and Sharir, M. (1995).
Filling Gaps in the Boundary of a Polyhedron.
Computer Aided Geometric Design, 12:207–229.

Bielajew, A. F. (1995).
HOWFAR and HOWNEAR: Geometry Modeling for Monte Carlo Particle Transport.
Technical report, National Research Council of Canada.
PIRS-0341.

Borodin, P., Novotni, M., and Klein, R. (2002).
Progressive Gap Closing for Mesh Repairing.
Advances in Modelling, Animation and Rendering, pages 201–21.

Duguet, F. and Drettakis, G. (2002).
Robust Epsilon Visibility.
In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 567–575, New York, NY, USA. ACM.

References II


Edelsbrunner, H. and Mücke, E. (1994).
Three-Dimensional Alpha Shapes.
ACM Transactions on Graphics, 13(1):43–72.


Girard, S. M. (2008).
MCNP: A General Purpose N-Particle Transport Code Version 5.
Technical report, Los Alamos National Laboratory.
LA-UR-03-1987.


Horn, W. P. and Taylor, D. L. (1988).
A Theorem to Determine the Spatial Containment of a Point in a Planar Polyhedron.
Computer Vision, Graphics, and Image Processing, 45:106–116.


Ju, T. (2004).
Robust Repair of Polygonal Models.
In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 888–895, New York, NY, USA. ACM.


Khamayseh, A. and Kuprat, A. (2008).
Deterministic Point Inclusion Methods for Computational Applications with Complex Geometry.
Computational Science and Discovery, 1.


Lane, J., Magedson, B., and Rarick, M. (1984).
An Efficient Point in Polyhedron Algorithm.
Computer Vision, Graphics, and Image Processing, 26:118–125.

References III

Marcille, T. F., Dixon, D. D., Fischer, G. A., Doherty, S. P., Poston, D. I., and Kapernick, R. J. (2006).
Design of a Low Power, Fast-Spectrum, Liquid-Metal Cooled Surface Reactor System.
In *Proceedings of the Space Technology and Applications International Forum—STAIF 2006*, pages 319–326.

Möller, T. and Trumbore, B. (2005).
Fast, Minimum Storage Ray/Triangle Intersection.
In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 7, New York, NY, USA. ACM.

Murali, T. M. and Funkhouser, T. A. (1997).
Consistent Solid and Boundary Representations from Arbitrary Polygonal Data.
In *I3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 155–ff., New York, NY, USA. ACM.

Newman, T. S. and Yi, H. (2006).
A Survey of the Marching Cubes Algorithm.
Computer and Graphics, 30:854–879.

Nordbeck, S. and Rydstedt, B. (1967).
Computer Cartography Point-In-Polygon Programs.
BIT Numerical Mathematics, 7:39–64.

O'Rourke, J. (1998).
Computational Geometry in C.
Cambridge University Press, second edition.

References IV

Platis, N. and Theoharis, T. (2003).
Fast Ray-Tetrahedron Intersection using Plücker Coordinates.
Journal of Graphics Tools, 8(4):37–48.

Popescu, L. M. (2003).
A Geometry Modeling System for Ray Tracing or Particle Transport Monte Carlo Simulation.
Computer Physics Communications, 150(1):21–30.

Segura, R. J. and Feito, F. R. (2001).
Algorithms to Test Ray-Triangle Intersection Comparative Study.
In *WSCG 2001 Conference Proceedings*.

Tautges, T. J., Wilson, P. P., Kraftcheck, J. A., Smith, B. M., and Henderson, D. L. (2009).
Acceleration Techniques for Direct Use of CAD-Based Geometries in Monte Carlo Radiation Transport.
In *Proc. International Conference on Mathematics, Computational Methods, and Reactor Physics*.

Williams, D. C. (2010).
GEANT4 Geometry Tracking Questions.
`http://scipp.ucsc.edu/~davidw/geant4/geo.question.html`.