Achievements and challenges in ORANGE surface geometry

Seth R Johnson

Celeritas Code Lead
Senior R&D Staff
Scalable Engineering Applications

CELERITAS

Celeritas core team:

Elliott Biondo *(ORNL)*, Julien Esseiva *(LBNL)*, Hayden Hollenbeck *(UVA)*, Seth R Johnson *(ORNL)*, Soon Yung Jun *(FNAL)*, Guilherme Lima *(FNAL)*, Amanda Lund *(ANL)*, Ben Morgan *(U Warwick)*, Stefano Tognini *(ORNL)*

Celeritas core advisors:

Tom Evans *(ORNL)*, Philippe Canal *(FNAL)*, Marcel Demarteau *(ORNL)*, Paul Romano *(ANL)*

U.S. DEPARTMENT OF ENERGY

Geant4 Collaboration Meeting
8 October, 2024

Background

Methods

Future work

ORANGE: Oak Ridge Advanced Nested Geometry

- Unbounded surface-based quadric models
  - NE codes (KENO, MCNP): ≥55 years of history
  - CAD applications before NURBS
  - Neutral particles or no magnetic fields
  - Repeated simple units and “masking” to overlay features
- Recent effort to adapt to GPU
  - 2021: Initial prototype GPU port in Celeritas
  - 2023: Multi-level geometry and Shift GPU integration

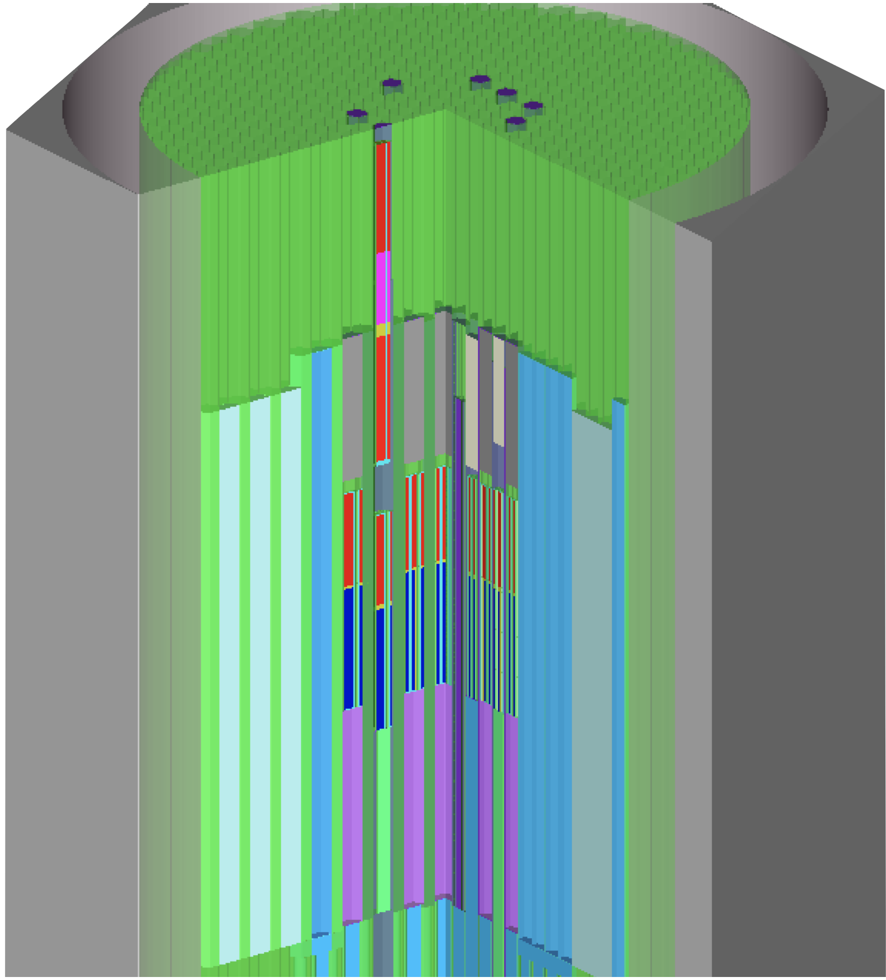


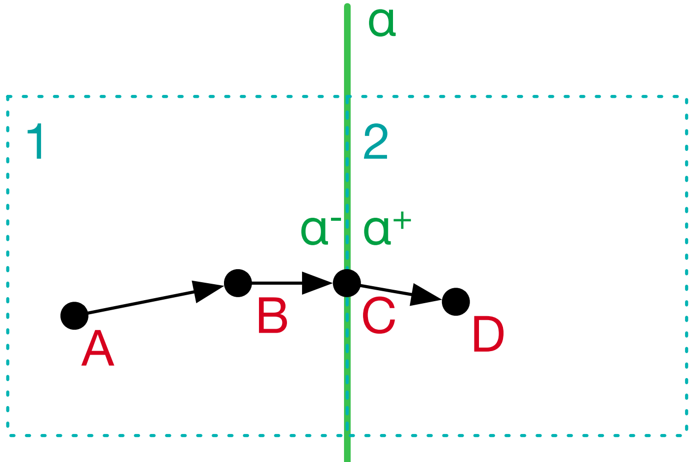
Image credit: Steve Skutnik (ORNL)

Motivation: “skin”-free boundaries

- Fewer tracking errors
- Better code reuse
- Avoid “thick” faces
  - **No uncertainty about which volume a point belongs to**
  - Single per-model relative “epsilon” used during construction
  - Reject and bump if new track begins **numerically exactly** on a boundary
- Adds **extra surface state**
  - Surface ID: which surface
  - Sense: which side of the surface

	Position	Volume	Surface+Sense
(input)	A	—	
Initialize	A	1	—
Find step	A	1	—
Move internal	B	1	—
Move to bndy	C	1	α inside
Cross bndy	C	2	α outside ★
Move internal	D	2	—

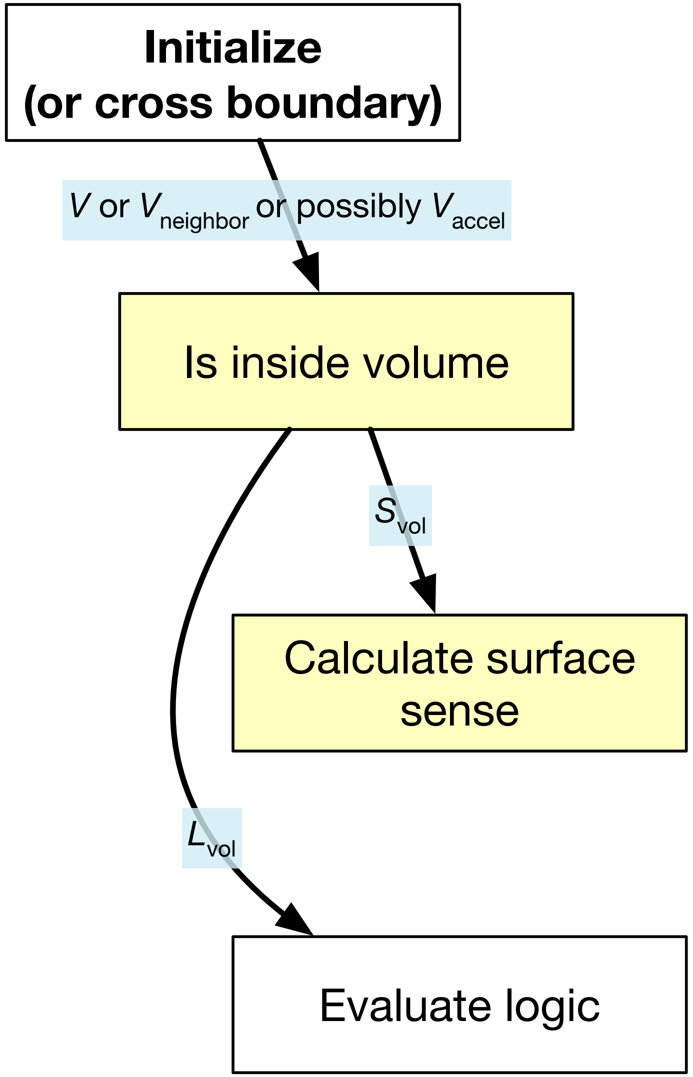
★ *exact handling of direction changes on boundaries from field*



OAK RIDGE National Laboratory

Motivation: GPU-friendly

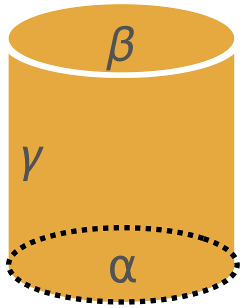
- **Lower divergence** compared to hardcoded volume representation
  - Point-in-surface and distance-to-surface **functions are *much* smaller** than point-in-volume
  - **Shared code** among quadratic intercept solvers
- **Lower stack size** *(implementation dependent)*
  - Lower register requirement increases kernel occupancy
  - Implementation choice: temporary global memory buffer *(sizeof(real) + sizeof(int))* × *(max faces per region)*



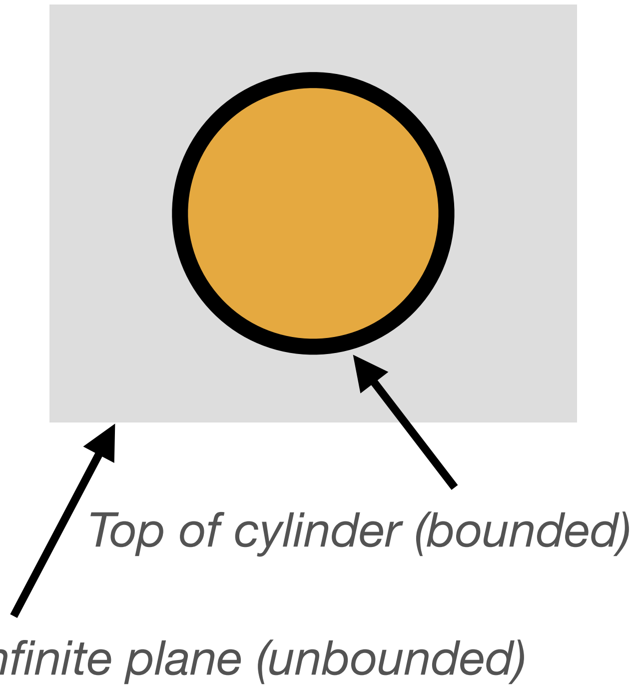
Oak Ridge National Laboratory

Notable differences from surface “frames”

- Infinite quadric/planar surfaces do not reject on a bounding “frame”
- No fuzziness near edges/corners
- Fewer surface types compared to frames
- Lower memory usage
  - Infinite: 1 real number for unbounded axis-aligned plane, up to 10 for general quadric surface, and several in between
  - Bounded: frame parameterization + transform: 12 reals
- No exact “safety” distance



Cylinder: (α & ~β & ~γ)

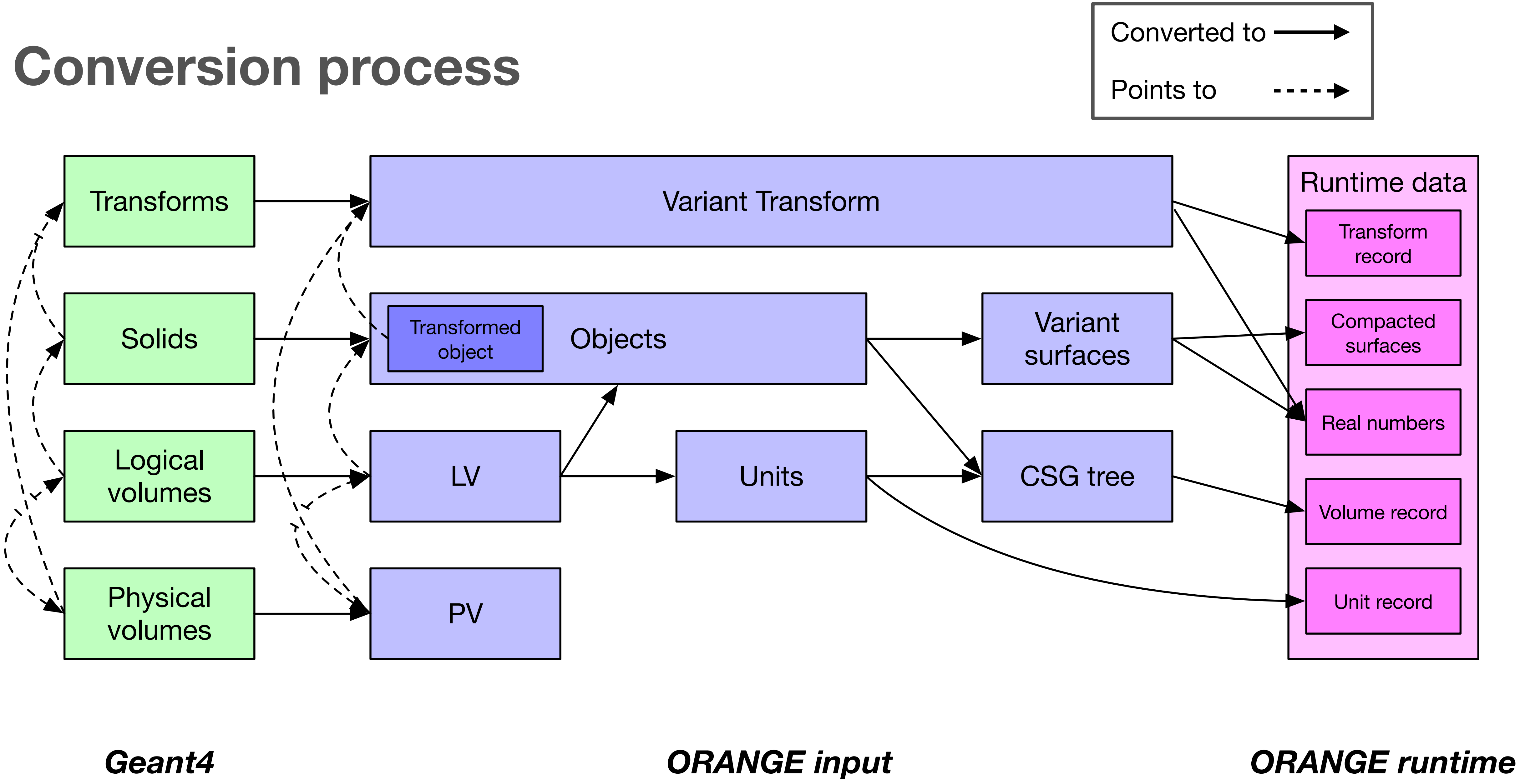


Background

Methods

Future work

Conversion process



Details: https://celeritas-project.github.io/celeritas/user/implementation/orange.html

Geant4 solid → surfaces

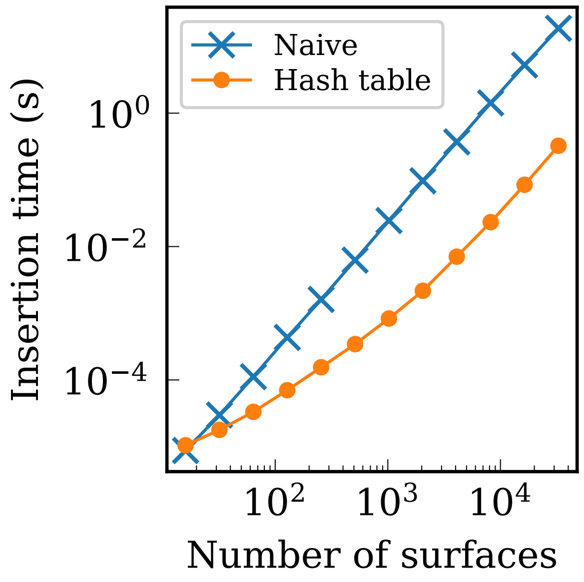
- Scale lengths based on unit system
- Build object of varying complexity
- Create surfaces from object components

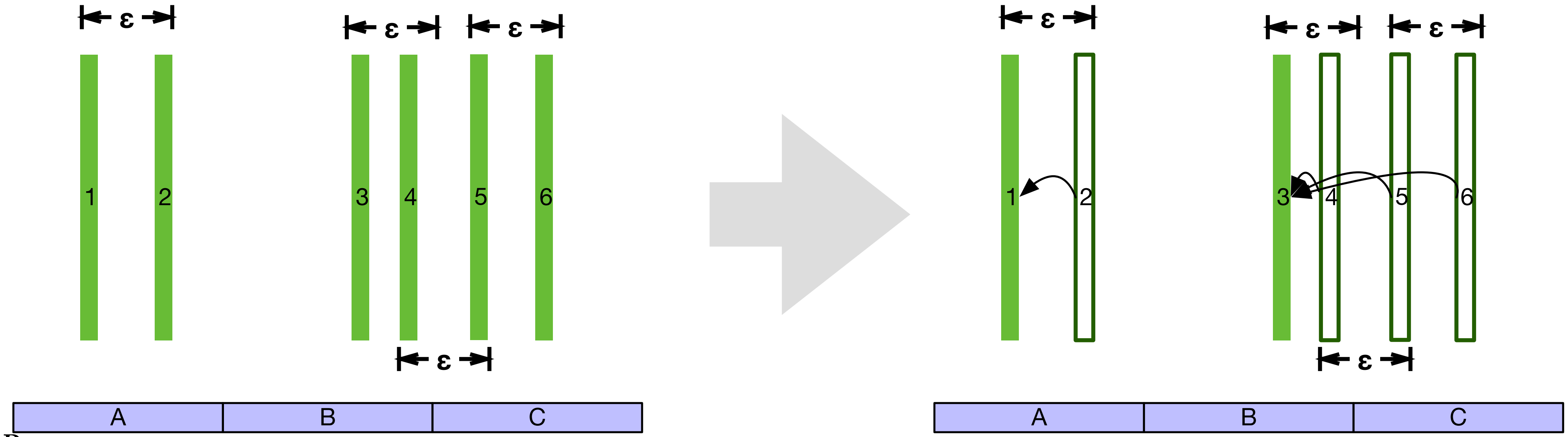
```cpp
auto SolidConverter::cons(arg_type solid_base) -> result_type
{
    auto const& solid = dynamic_cast<G4Cons const&>(solid_base);

    auto const outer_r = scale_.to<Real2>(solid.GetOuterRadiusMinusZ(),
                                          solid.GetOuterRadiusPlusZ());
    auto const inner_r = scale_.to<Real2>(solid.GetInnerRadiusMinusZ(),
                                          solid.GetInnerRadiusPlusZ());
    auto hh = scale_(solid.GetZHalfLength());

    std::optional<Cone> inner;
    if (any_positive(inner_r))
    {
        inner = Cone{inner_r, hh};
    }

    return make_solid(
        solid, Cone{outer_r, hh}, std::move(inner), make_wedge_azimuthal(solid));
}
```

```cpp
void Cone::build(IntersectSurfaceBuilder& insert_surface) const
{
    if (CELER_UNLIKELY(
            SoftEqual{insert_surface.tol().rel}(radii_[0], radii_[1])))
    {
        // Degenerate cone: build a cylinder instead
        Cylinder cyl{real_type{0.5} * (radii_[0] + radii_[1]), hh_};
        return cyl.build(insert_surface);
    }

    // Build the bottom and top planes
    insert_surface(Sense::outside, PlaneZ{-hh_});
    insert_surface(Sense::inside, PlaneZ{hh_});

    // Calculate the cone using lo and hi radii
    real_type const lo{radii_[0]};
    real_type const hi{radii_[1]};

    // Arctangent of the opening angle of the cone (opposite / adjacent)
    real_type const tangent = std::fabs(lo - hi) / (2 * hh_);

    // Calculate vanishing point (origin)
    real_type vanish_z = 0;
    if (lo > hi)
    {
        // Cone opens downward (base is on bottom)
        vanish_z = -hh_ + lo / tangent;
        CELER_ASSERT(vanish_z > 0);
    }
    else
    {
        // Cone opens upward
        vanish_z = hh_ - hi / tangent;
        CELER_ASSERT(vanish_z < 0);
    }

    // Build the cone surface along the given axis
    ConeZ cone{Real3{0, 0, vanish_z}, tangent};
    insert_surface(cone);

    // Set radial extents of exterior bbox
    insert_surface(Sense::inside, make_xyradial_bbox(std::fmax(lo, hi)));
}
```

Surface deduplication

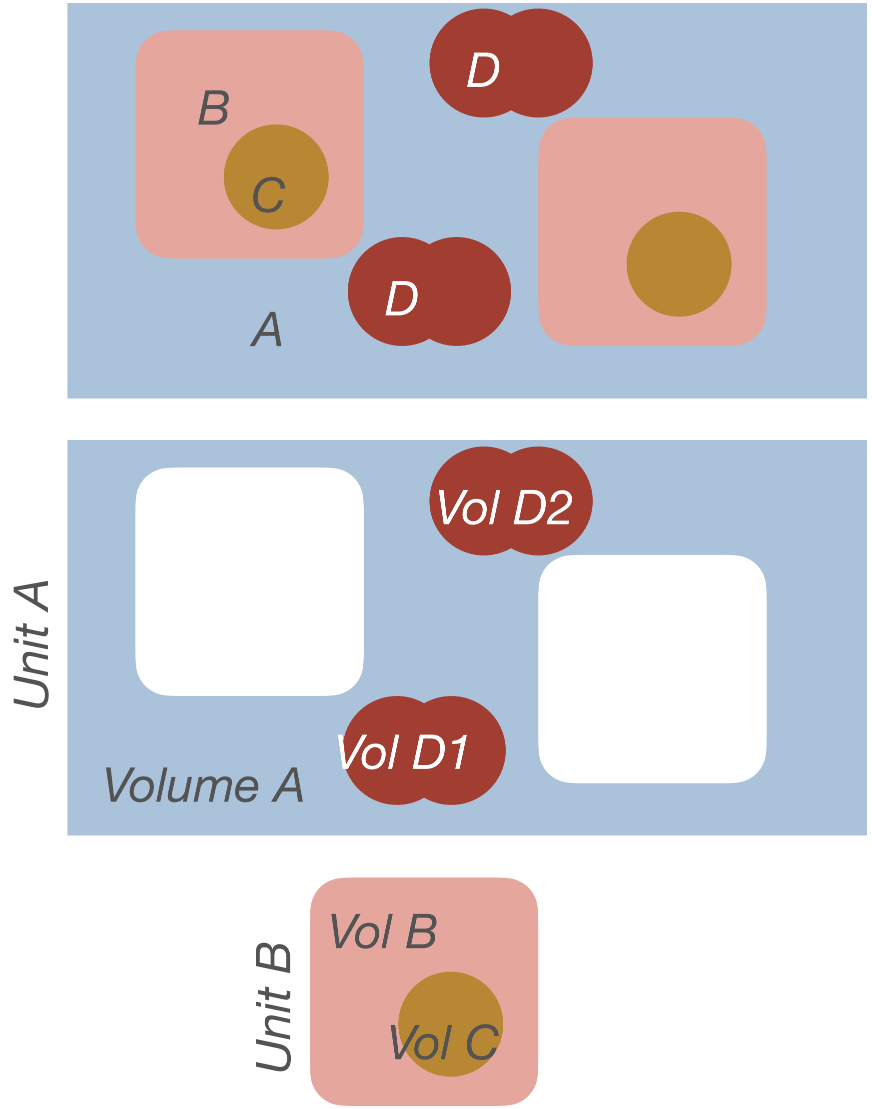
- Necessary to avoid “lost” track at surface crossing
- Similar to “vertex welding” with triangle meshes
- Hash lookup accelerates 100×





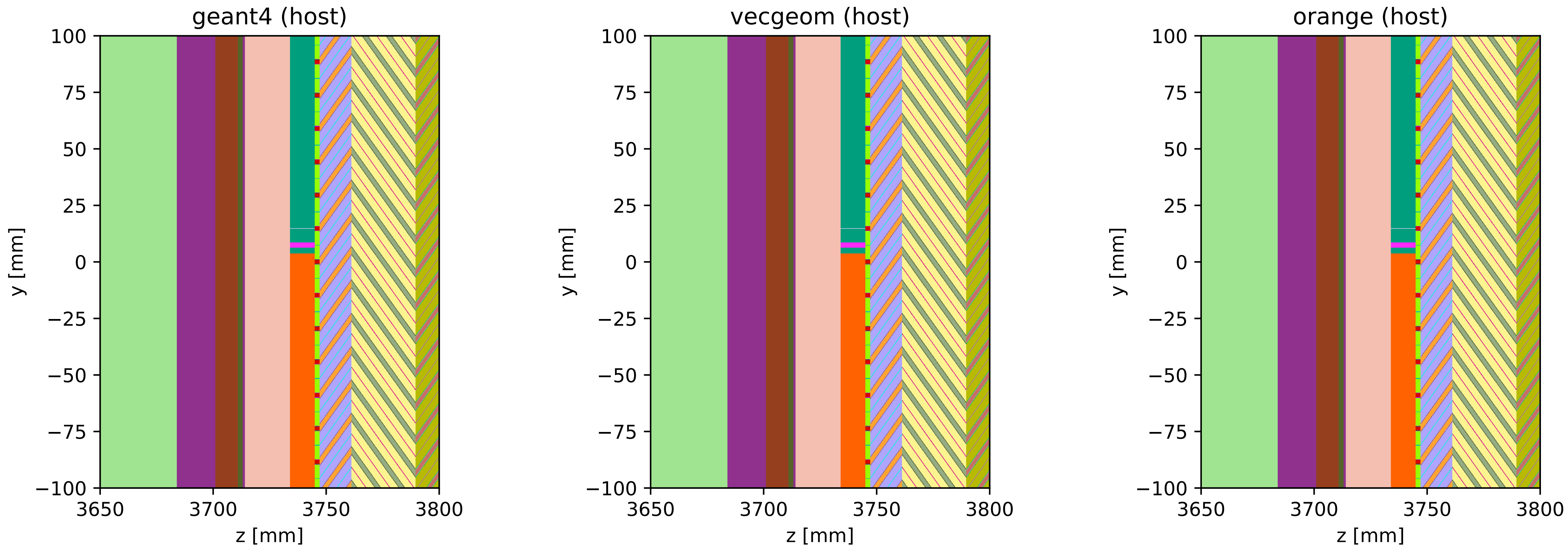
Geant4 volume hierarchy conversion

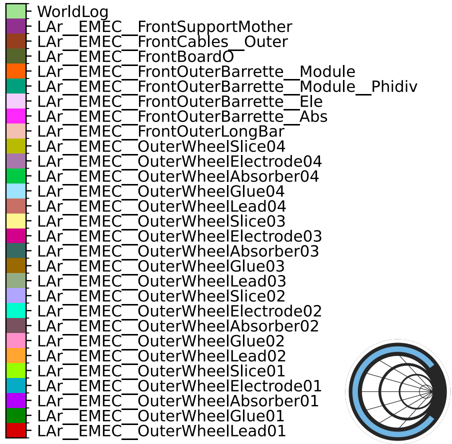
- Some LVs become CSG “volumes”
  - LV has no children
  - LV appears only once in geometry and has no transform
  - LV solid is a union *(shortcoming to be discussed later)*
- Remaining LVs become “units”
  - Repeatable throughout geometry
  - Surfaces are constructed in unit’s reference frame



Rasterizer with Python interface

ATLAS EMEC
(model courtesy Evgueni Tcherniaev)





- Verify navigation and object conversion
- Independently discovered #2547

  (fixed by E. Tcherniaev, geom-specific-V11-02-03)

Supported geometry/platform

	CPU	Nvidia	AMD
Geant4	✅	❌	❌
VecGeom	✅	✅	❌
ORANGE	✅	✅	✅

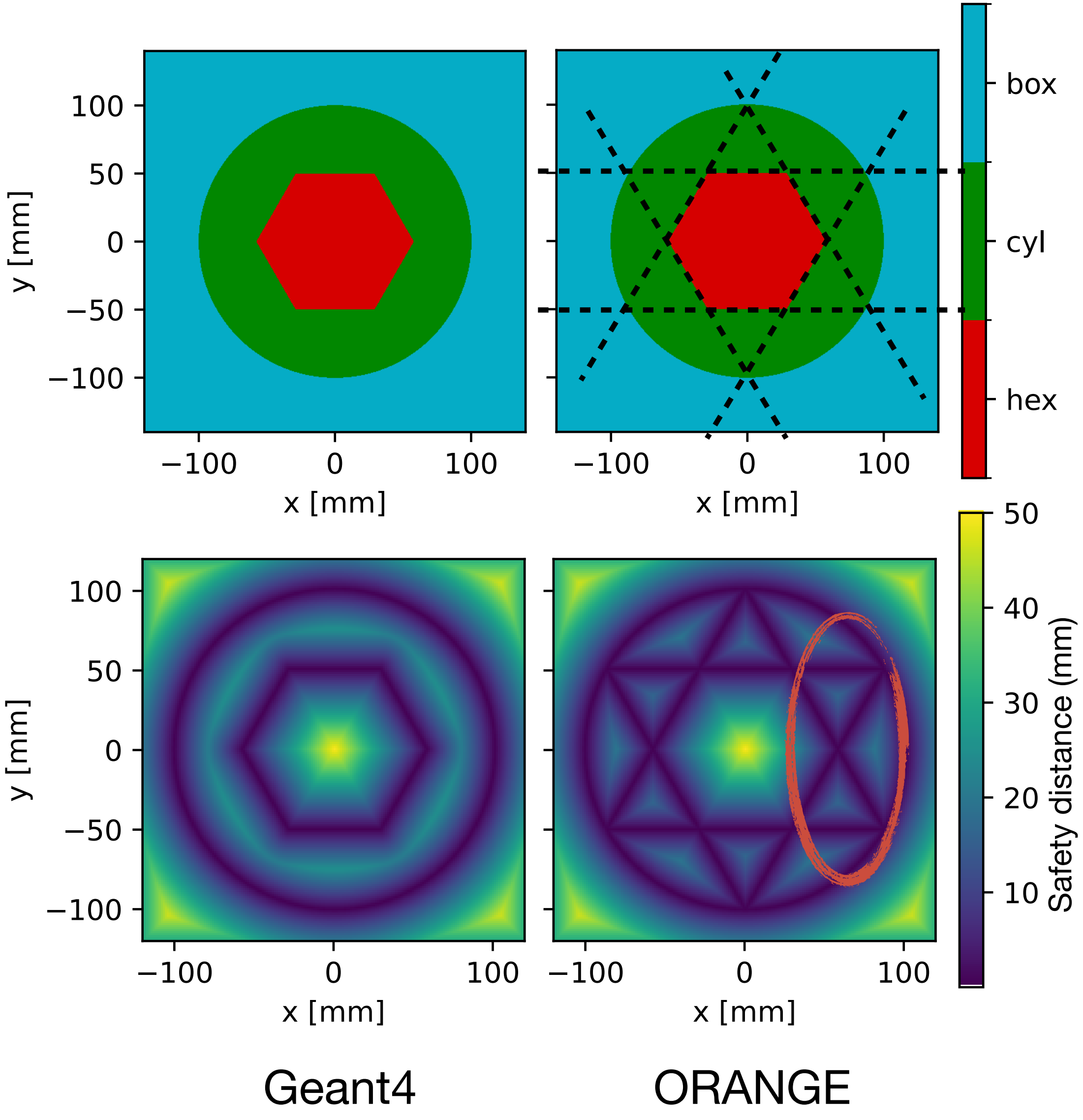
Background

Methods

Future work

Safety distance calculation

- Current implementation is “rough and ready”
- Exact when:
  - Surface is planar/spherical/cylindrical, **and**
  - Interior of convex region
- Over-conservative often
  - Nearest distance to internal boundaries, **or**
  - Zero for generic quadrics *(elliptical, hyperbolic, parabolic)*
- Dramatically conservative safety is **bad**
  - Prevents MSC displacement
  - Forces smaller interaction lengths



OAK RIDGE National Laboratory

Features to be completed

- Accurate and complete safety distance
- Non-quadric surfaces: torus, tessellated, twisted
- Some solid conversions
- Mapping volume IDs to Geant4 volumes

https://github.com/celeritas-project/celeritas/issues/1167

Future optimizations

- CSG tree simplification using SAT solvers
  - Logically eliminate “shared” surfaces at construction time
  - Logically catch overlapping geometry without point sampling
- Postfix evaluation of runtime CSG volume logic[1]
- Tracking through tree acceleration structure
- Tighter bounding box calculations

Computer-Aided Design 178 (2025) 103803

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Application Paper

Point containment algorithms for constructive solid geometry with unbounded primitives

Paul K. Romano [a,*], Patrick A. Myers [b], Seth R. Johnson [c], Aljaž Kolšek [d], Patrick C. Shriwise [a]

[a] Argonne National Laboratory, 9700 S. Cass Ave, Lemont, IL 60439, United States
[b] University of Michigan, Ann Arbor, MI 48103, United States
[c] Oak Ridge National Laboratory, Oak Ridge, TN 37830, United States
[d] Universided Nacional de Educación a Distancia, Madrid, Spain

ARTICLE INFO

Dataset link: 10.5281/zenodo.13357943

Keywords:
Constructive solid geometry
Point containment
Particle transport
Monte Carlo

ABSTRACT

We present several algorithms for evaluating point containment in constructive solid geometry (CSG) trees with unbounded primitives. Three algorithms are presented based on postfix, prefix, and infix notations of the CSG binary expression tree. We show that prefix and infix notations enable short-circuiting logic, which reduces the number of primitives that must be checked during point containment. To evaluate the performance of the algorithms, each algorithm was implemented in the OpenMC Monte Carlo particle transport code, which relies on CSG to represent solid bodies through which subatomic particles travel. Two sets of tests were carried out. First, the execution time to generate a rasterized image of a 2D slice of three CSG models of varying complexity was measured. Use of both prefix and infix notations offered significant speedup over the postfix notation that has traditionally been used in particle transport codes, with infix resulting in a 6× reduction in execution time relative to postfix for a model of a tokamak fusion device. We then measured the execution time of neutron transport simulations of the same three models using each of the algorithms. The results and performance improvements reveal the same trends as for the rasterization test, with a 5.52× overall speedup using the infix notation relative to the original postfix notation in OpenMC for the tokamak model.

1. Introduction

Constructive solid geometry (CSG) is a technique for representing rigid solids using Boolean set operations applied to simple geometric primitives [1–3]. In CSG, a solid is represented as an ordered, binary tree where the non-leaf nodes are Boolean set operators (union, intersection, difference) and the leaf nodes represent geometric primitives. An example of such a CSG tree is illustrated in Fig. 1. CSG representations can be based on bounded primitives (cuboids, spheres, cylinders, cones, etc.) or unbounded primitives using general half-spaces.

CSG has applications in geometric modeling, computer-aided design, computer graphics, computational science, and other areas where accurate spatial modeling is required [1,2,4]. In particular, CSG is widely used for geometric modeling in Monte Carlo (MC) particle transport simulations through codes such as OpenMC [5], MCNP [6], Shift [7], FLUKA [8], and Serpent [9]. MC particle transport itself has applications across a wide range of scientific domains, such as nuclear reactor design, radiation shielding, medical physics, high-energy experimental physics, and fusion energy.

Most MC particle transport codes — including OpenMC, which was used for the present study — use a CSG representation based on unbounded primitives defined by inequalities of implicit surfaces. An implicit surface is a surface in Euclidean space defined by an equation $f(\mathbf{r}) = 0$. By convention, the inside of the surface is defined as the set of points for which $f(\mathbf{r}) < 0$ (the negative half-space) and the outside of the surface is defined by $f(\mathbf{r}) > 0$ (the positive half-space). Thus, the leaf nodes in the CSG tree are implicit surface inequalities. Surface inequalities are then combined with Boolean set operations to create more complex solids. MC particle transport codes are typically limited to primitives based on algebraic surfaces (planes, quadrics, and sometimes torii); nearly all allow the use of set intersection and

* Corresponding author.
E-mail addresses: promano@anl.gov (P.K. Romano), myerspat@umich.edu (P.A. Myers), johnsonsr@ornl.gov (S.R. Johnson), akolsek@ind.uned.es (A. Kolšek), pshriwise@anl.gov (P.C. Shriwise).

https://doi.org/10.1016/j.cad.2024.103803
Received 22 November 2023; Received in revised form 22 August 2024; Accepted 8 September 2024
Available online 11 September 2024

[1] P.K. Romano, P.A. Myers, S.R. Johnson, A. Kolšek, P.C. Shriwise, Point containment algorithms for constructive solid geometry with unbounded primitives, Computer-Aided Design 178 (2025) 103803. https://doi.org/10.1016/j.cad.2024.103803.

Refactor for HEP-optimized geometry

- Major changes and additions
  - Secondary single-precision structures for volume-based safety
  - Two-level (global, local) navigation with boundary surface matching
- Feasible given code structure
  - Object construction automatically generates volume metadata
  - Key tracking components will not change

Acknowledgments

Celeritas v0.5 code contributors:

- Seth R Johnson (@sethrj)
- Amanda Lund (@amandalund)
- Julien Esseiva (@esseivaju)
- Soon Yung Jun (@whokion)
- Guilherme Lima (@mrguilima)
- Stefano Tognini (@stognini)
- Vidor HL Montiel (@VHLM2001)
- Ben Morgan (@drbenmorgan)
- Elliott Biondo (@elliottbiondo)
- Hayden Hollenbeck (@hhollenb)
- Philippe Canal (@pcanal)
- Peter Heywood (@ptheywood)

Past code contributors:

- Doaa Deeb (@DoaaDeeb)
- Vincent R Pascuzzi (@vrpascuzzi)
- Paul Romano (@paulromano)

OLCF: This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

SciDAC: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of High Energy Physics, Scientific Discovery through Advanced Computing (SciDAC) program.

NERSC: This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award HEP-ERCAP-0023868.

Code | *Documentation* | *Publication*

https://celeritas.ornl.gov

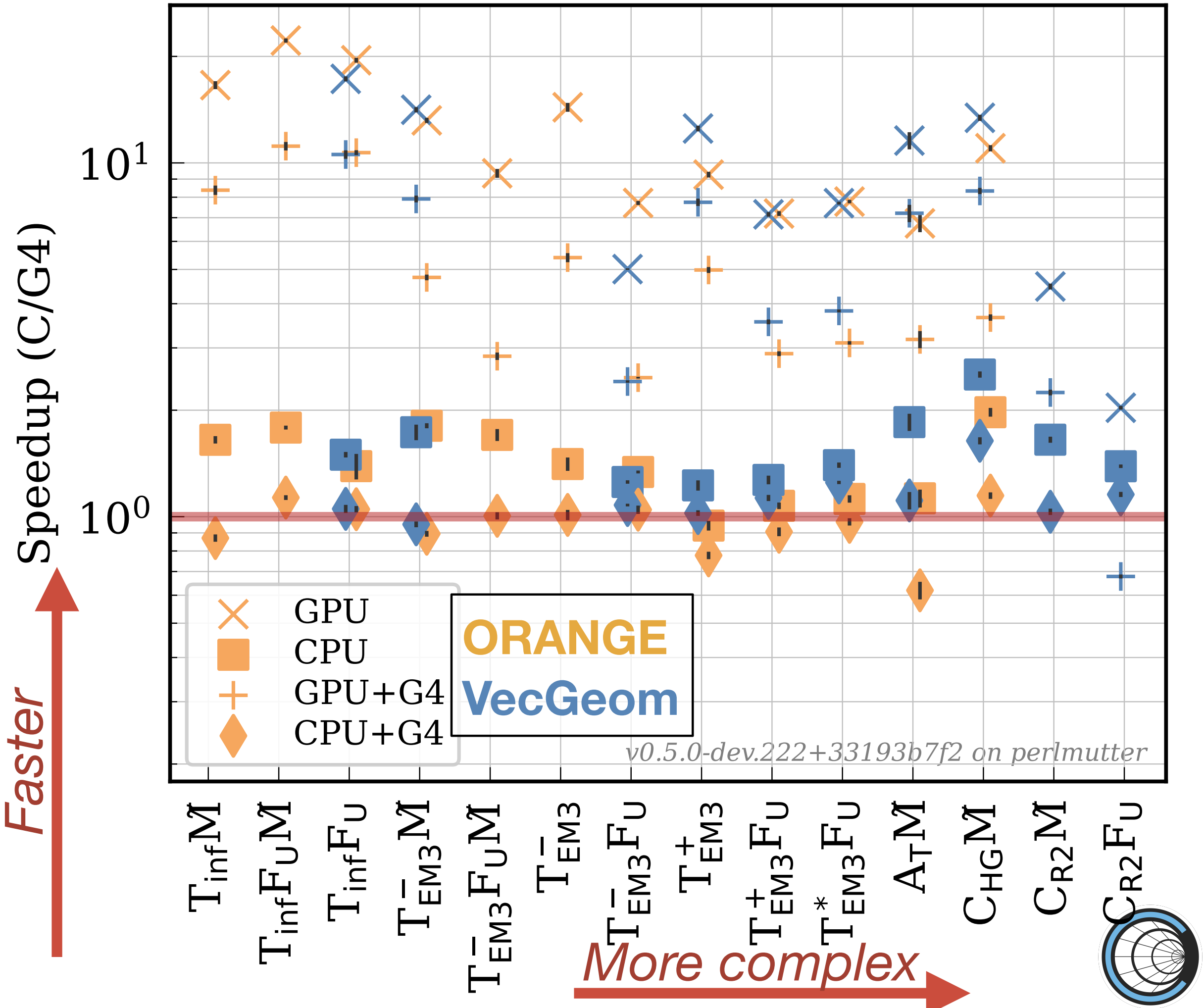
BACKUP SLIDES

Standalone EM performance

- LHC-scale simulations on DOE LCF
  - 1300 × 10 GeV e⁻, 16 events
  - ¼ Perlmutter node (NERSC)
    1 × Nvidia A100 GPU, ¼ × 64-core AMD EPYC 7763
  - Celeritas GPU vs CPU
    CUDA (1 CPU thread) vs OpenMP (16 CPU threads)
- **Key metrics favor GPU**
  - *Capacity:* **32–93% loss** if GPUs are ignored
  - *Efficiency:* up to **5×** performance per watt
- Performance improvements to be discussed at CHEP 2024

Problem definition	
T_{inf}	"Infinite" medium
T_{EM3}	Idealized calorimeter
A_{TC}	ATLAS TileCal
C_{HG}	CMS HGCal
C_{R2}	CMS Run 2 (2018)

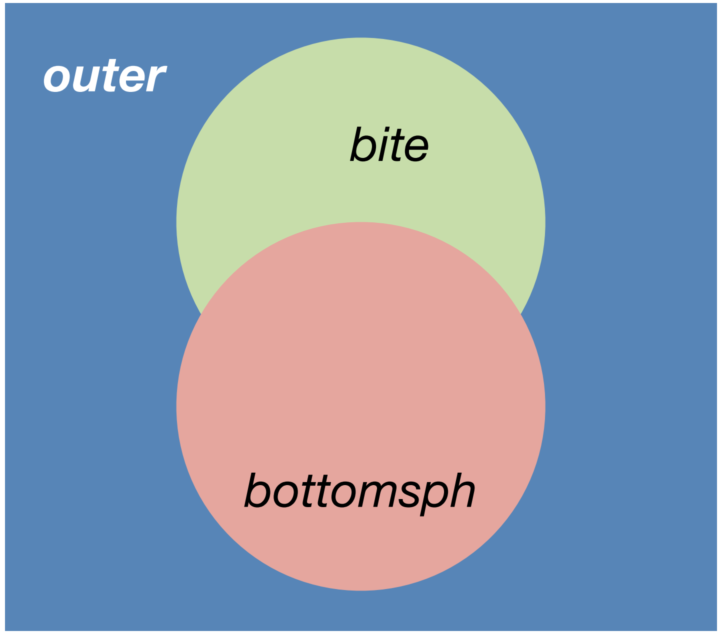
Modifier	
F_U	uniform 1T field
$\tilde{M}$	no MSC

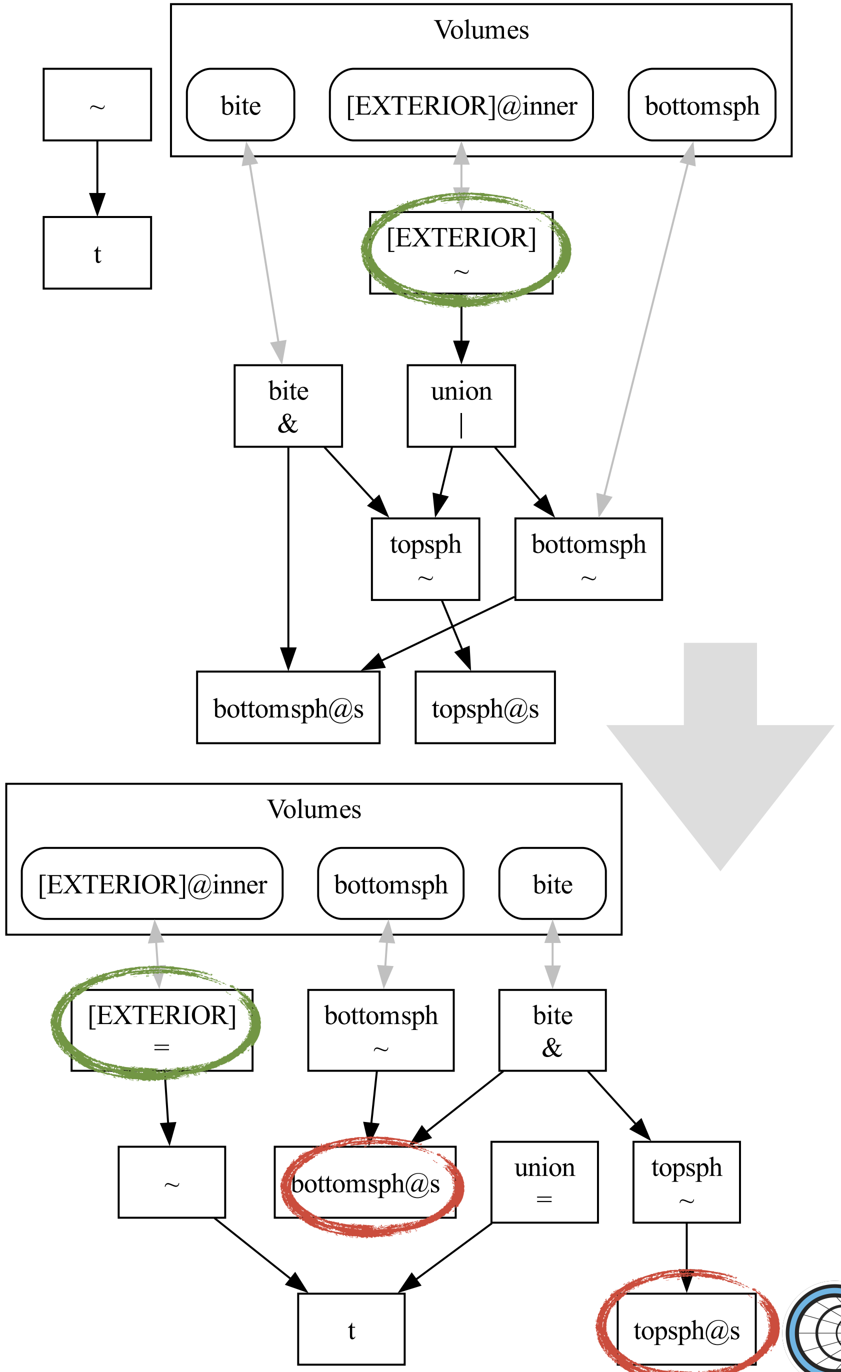


OAK RIDGE National Laboratory

Units with “union” boundaries

- Reactor geometry prohibits this
- Multi-level tracking “deletes” child exterior boundary to avoid initializing on surface
- Workaround: “inline” children





OAK RIDGE
National Laboratory

Tracking in "background" is too expensive

- Pseudo-volume approach
  - Exclude all immediate children
  - Cross through any surface if the crossing point is "inside" the target volume
  - Current algorithm finds distance to *all* connected surfaces in "background"
- Requires large "scratch space" for distances and surface index
  - Contains thread-local sort with several nested loops: memory intensive and long
  - CMS HGCal scratch space is **403** doubles per thread
- Solution: track through tree acceleration structure (e.g., BVH)