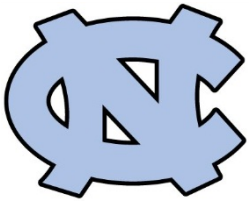


Computing Numerically-Optimal Bounding Boxes for CSG Components in Monte Carlo Particle Transport Calculations

David L. Millman[1], **David P. Griesheimer[1]**,
Brian R. Nease[1], and Jack Snoeyink[2]

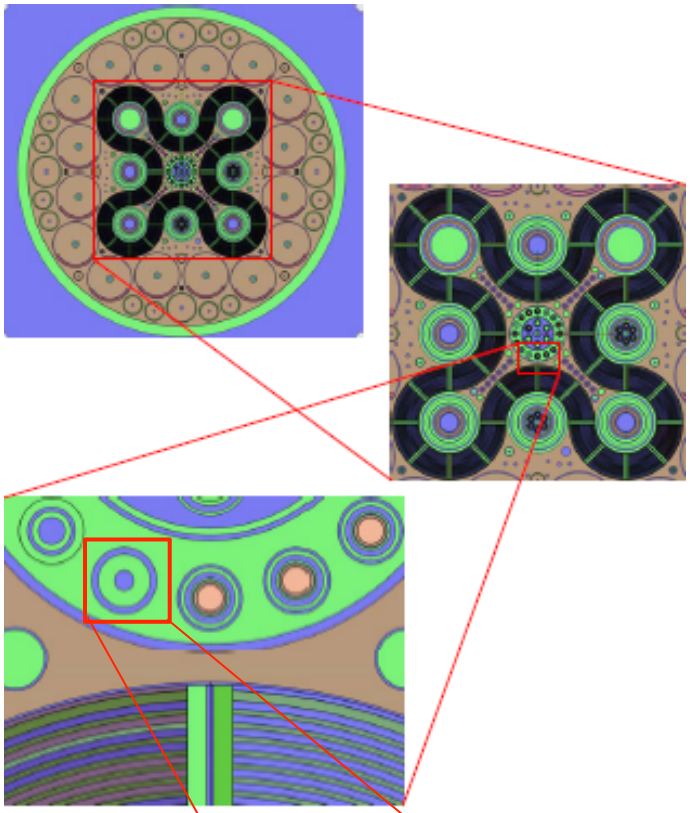
1. Bettis Laboratory, Bechtel Marine Propulsion Corp.
2. Department of Computer Science UNC-Chapel Hill

Joint International Conference on Supercomputing in Nuclear Applications
and Monte Carlo 2013 (SNA + MC 2013)
La Cité des Sciences et de l'Industrie, Paris, France, October 27–31, 2013

30 October 2013

Motivation/Background

T.M. Sutton, et. al., *The MC21 Monte Carlo Transport Code*, M&C+SNA 2007

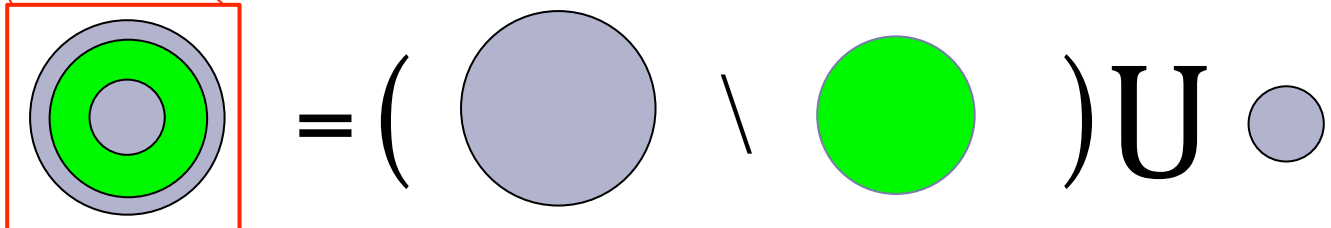


Constructive solid geometry (CSG) is used to define geometric objects in Monte Carlo transport calculations.

CSG provides an exact representation of an objects boundary.

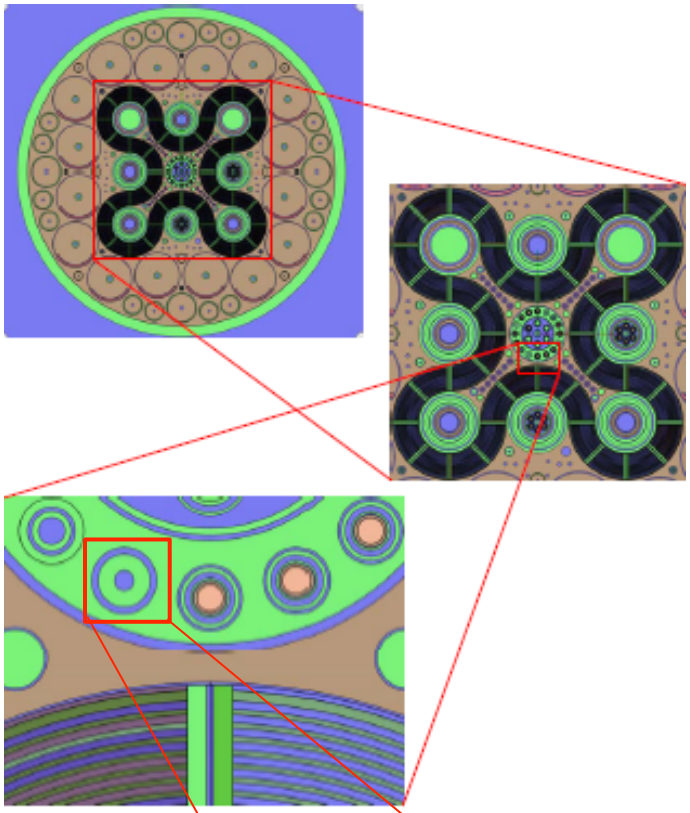
CSG allow nearly unlimited flexibility for creating complex models for:

- Criticality analysis
- Reactor analysis



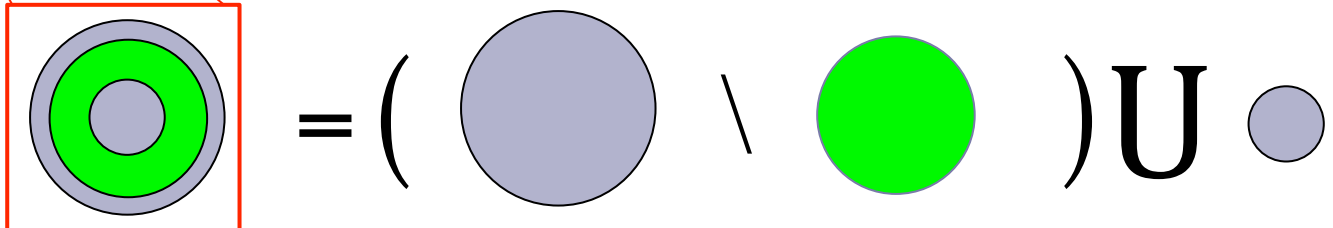
Motivation/Background

T.M. Sutton, et. al., *The MC21 Monte Carlo Transport Code*, M&C+SNA 2007



CSG components can be difficult to process. Compared to other representations, for CSG components:

- Particle tracking is slower
- Sampling is more resource intensive
- Properties (such as volume) are difficult to compute

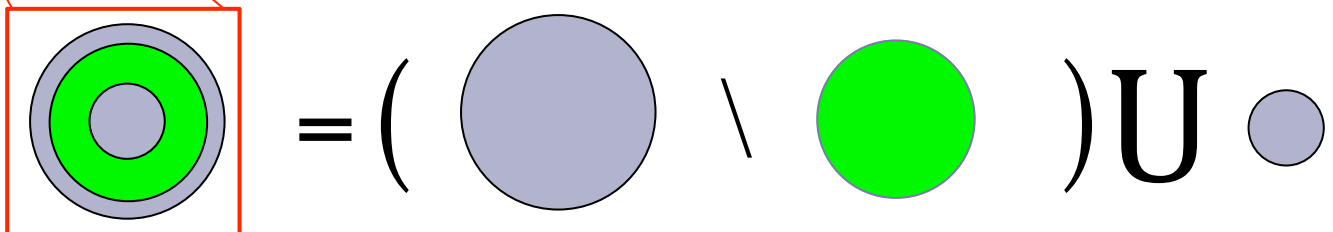


Motivation/Background

T.M. Sutton, et. al., *The MC21 Monte Carlo Transport Code*, M&C+SNA 2007

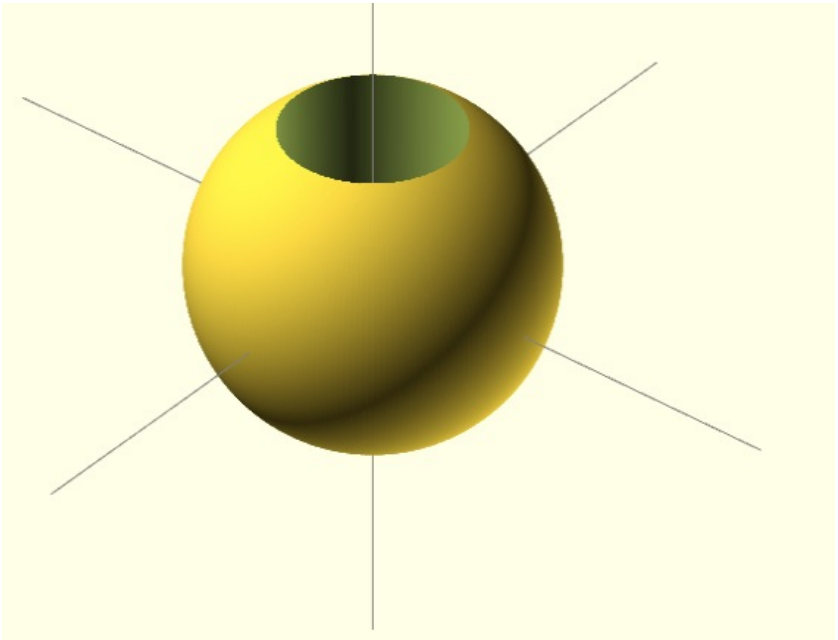
Bounding boxes help solve many of these difficulties….

…unfortunately, computing bounding boxes for CSG components is non-trivial.



What People See

Let *D* be the region left after drilling a radius *r* hole through the center of a radius *R* sphere centered at the origin.



What is the optimal axis-aligned box bounding of *D*?

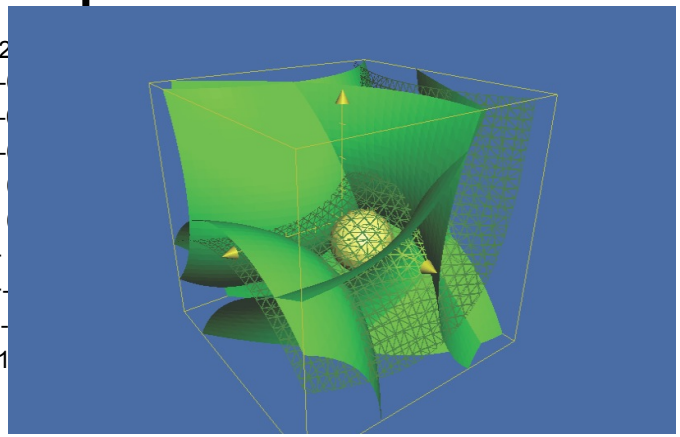
Provided $R > r$, a box with:

- minimal point $(-R, -R, -R+f(R,r))$
- maximal point $(R, R, R+f(R,r))$

What the Computer Sees

Let *D* be the intersection of 10 quadratics:

0 > 0.74742x^2 + 0.93022y^2 + 0.32256z^2 + 0.26590xy +-0.82750xz + 0.43517yz + 2.47974x +2
0 > 0.00487x^2 + 0.00638y^2 + 0.00212z^2 + 0.00181xy +-0.00537xz + 0.00299yz + 0.51989x +-
0 <-0.00469x^2 + 0.00617y^2 +-0.00134z^2 + 0.00116xy + 0.00609xz + 0.00326yz + 0.52845x +-
0 > 0.00180x^2 + 0.00647y^2 + 0.00497z^2 +-0.00039xy + 0.00597xz + 0.00003yz + 0.59729x +-
0 > 0.00173x^2 + 0.00681y^2 + 0.00479z^2 +-0.00022xy + 0.00574xz + 0.00034yz +-0.76442x +
0 > 0.00180x^2 + 0.00657y^2 + 0.00498z^2 +-0.00037xy + 0.00599xz + 0.00008yz +-0.76185x +
0 <-0.00156x^2 + 0.00591y^2 +-0.00403z^2 + 0.00324xy +-0.00503xz + 0.00601yz +-0.90629x +
0 > 0.00643x^2 + 0.00046y^2 + 0.00614z^2 +-0.00143xy +-0.00036xz +-0.00301yz +-0.04751x +-
0 > 0.00323x^2 +-0.00046y^2 +-0.00276z^2 + 0.00209xy +-0.01145xz + 0.00273yz +-0.19156x +-
0 < 0.50007x^2 + 0.50004y^2 + 0.50003z^2 + 0.00009xy + 0.00002xz + 0.00004yz + 6.69291x +1



From a picture, we can determine the bounding box without trouble.

Not so easy for a collection of polynomials.

Computing Bounding Boxes Is Difficult

Alg 1: Apply set operations to the bounding boxes of primitives.

"for difference and intersection operations this will hardly ever lead to an optimal bounding box."

–POV-Ray documentation

Alg 2: Convert CSG to boundary rep.

"efficient, accurate, and robust computation of the boundary remains a hard problem for CSG model described using curved primitives."

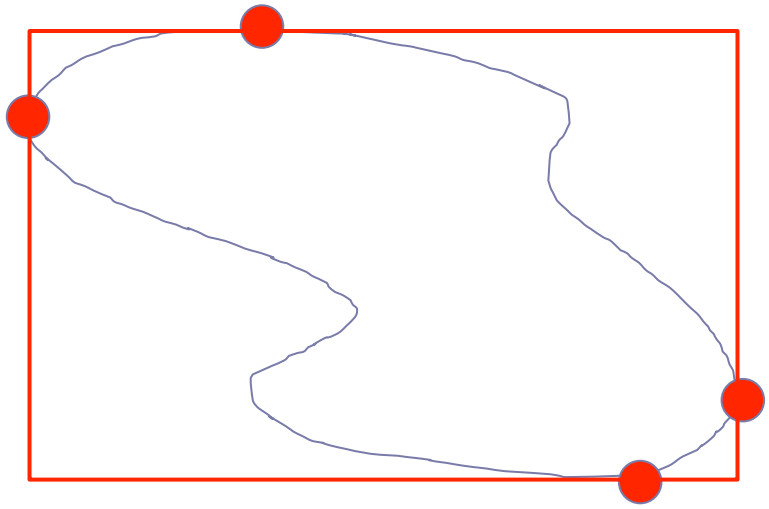
– Lin & Gottschalk [SG98]

More recent work indicates converting CSG to boundary rep is still hard. [K00], [MTT05], [SW06], [DLL+08]

Computing an AABB

Question: Given a domain *D*, compute the optimal axis-aligned bounding box (AABB) of *D*?

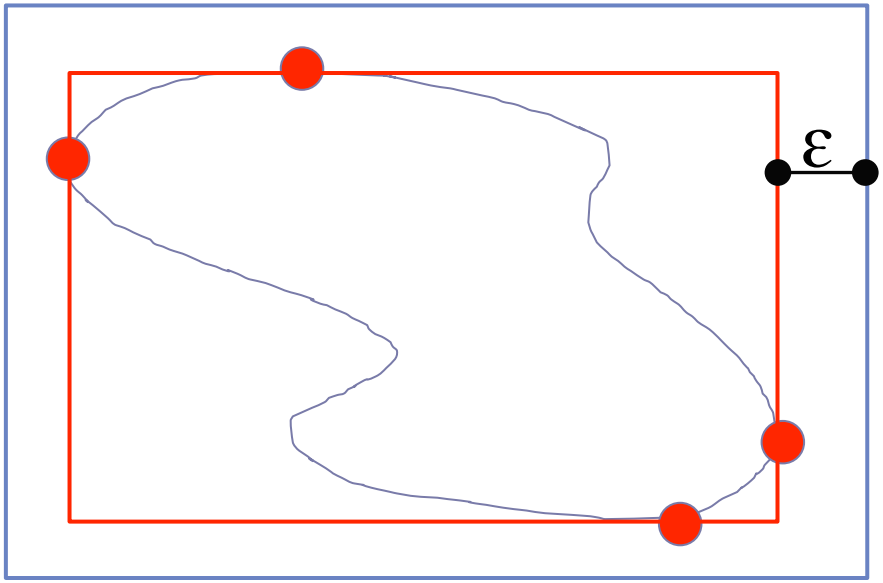
Observation: To compute optimal AABB we compute extremal points in each direction.



Ask an easier question

Given ε > 0, *ε-box* for *D* is an axis-aligned bounding box that is at most ε larger in each direction than the optimal axis aligned bounding box for *D.*

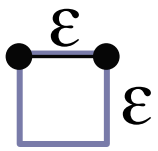
We call an *ε-box* a *numerically-optimal* bounding box.

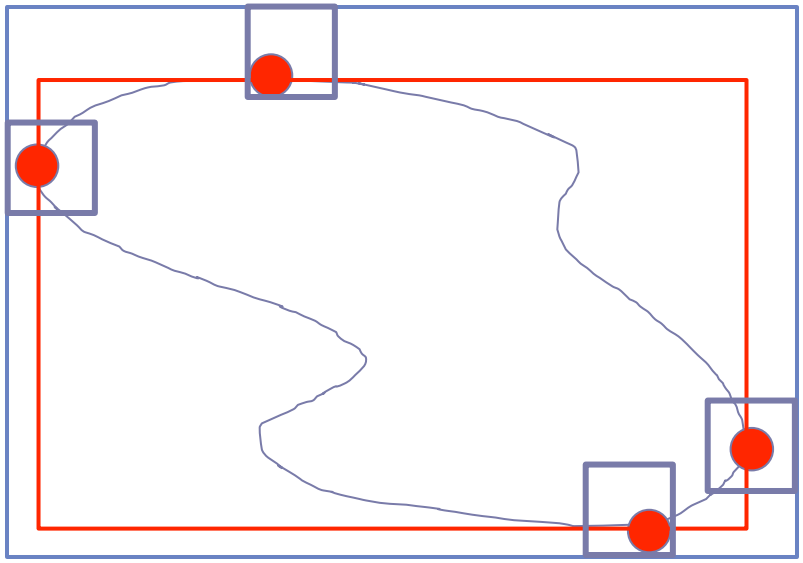


Ask an easier question

Question: Given a domain *D,* compute the *ε-box* of *D*?

Observation: To compute *ε-box* for *D* we must identify boxes of size *ε* containing an extremal point.





Ask an easier question

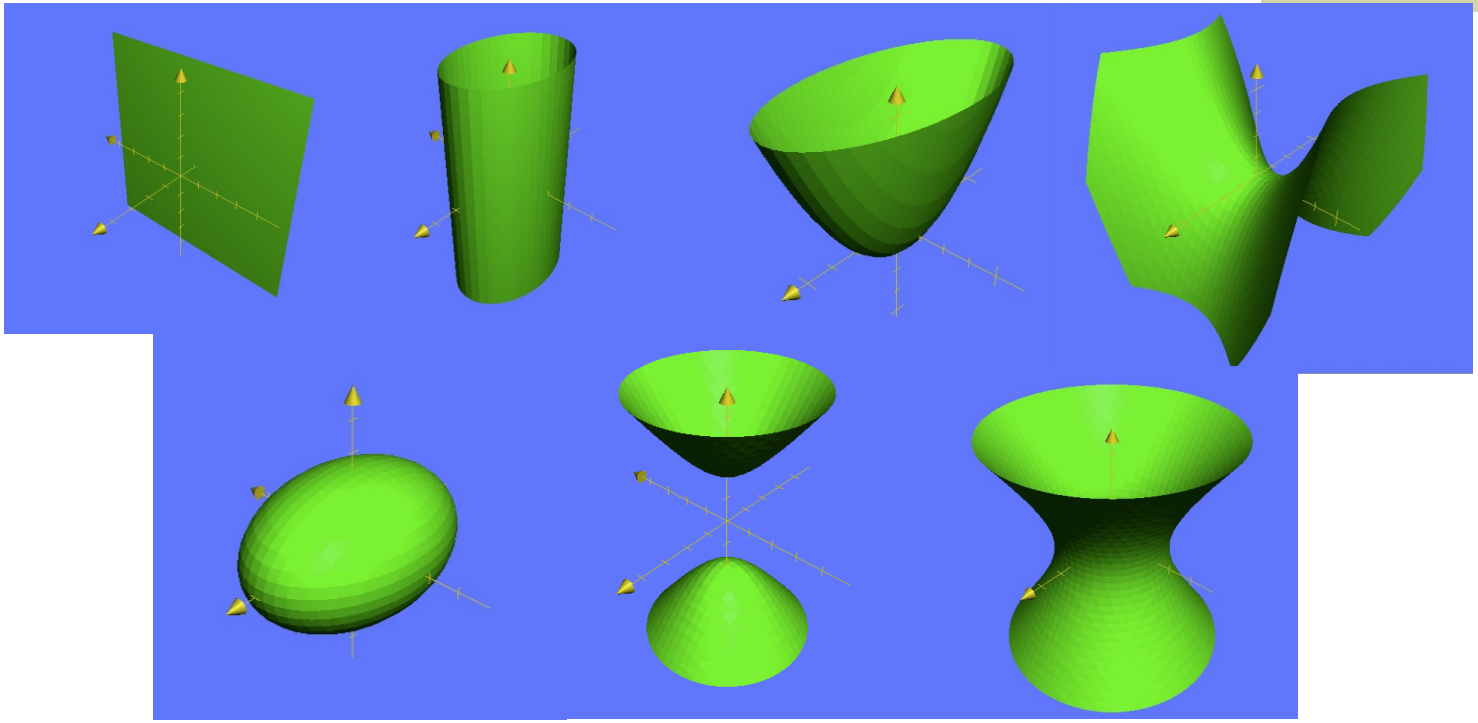
Question: Given a domain *D,* compute the *ε-box* of *D*?

Observation: To compute *ε-box* for *D* we must identify boxes of size *ε* containing an extremal point.

How do we identify boxes of size *ε* containing an extremal point?**

**In this talk, I describe a simpler version of our algorithm that is good in practice but does not have provably tight bounds. See our paper for the gory details of the full algorithm and the proofs!

Primitives: Signed Quadratic Surfaces

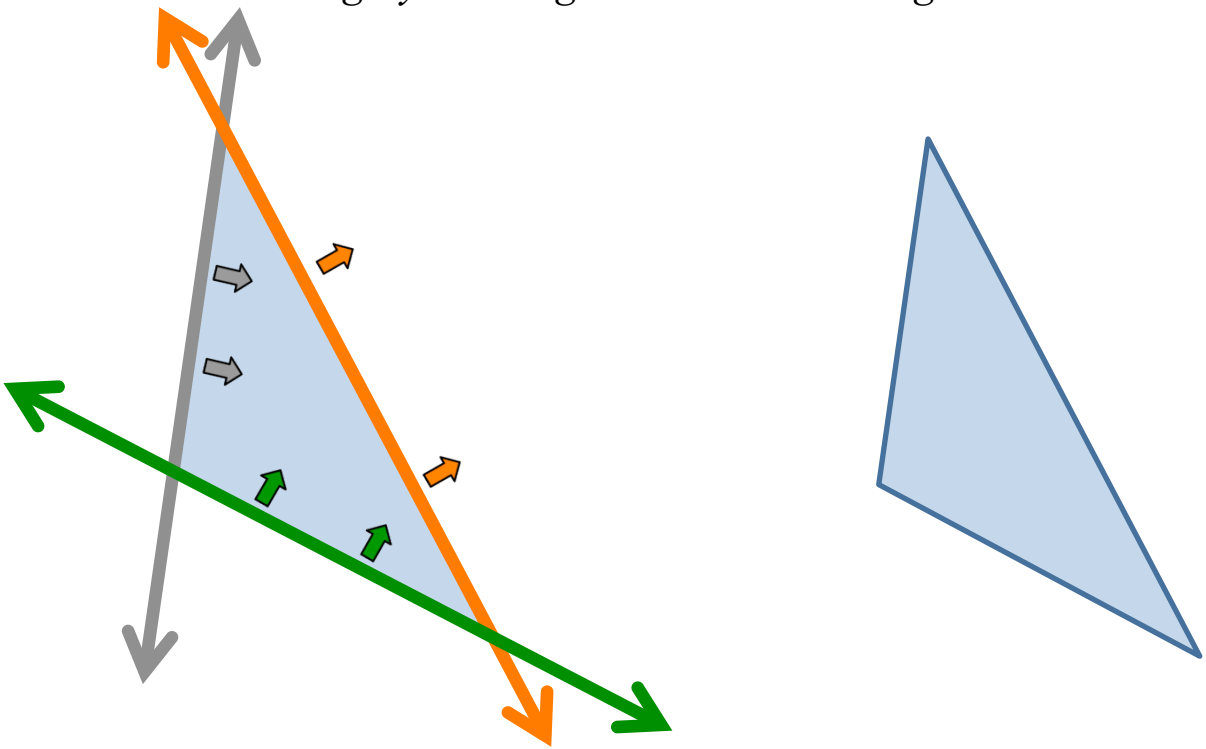


$$f(x,y,z) = Ax^2 + By^2 + Cz^2$$
$$+Dxy + Exz + Fyz$$
$$+Gx + Hy + Iz + J$$

Model Representation Component: Boolean Formula

A *component* defined by intersections and unions of signed surfaces

$$S_{grey} \cap S_{green} \cap -S_{orange}$$



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

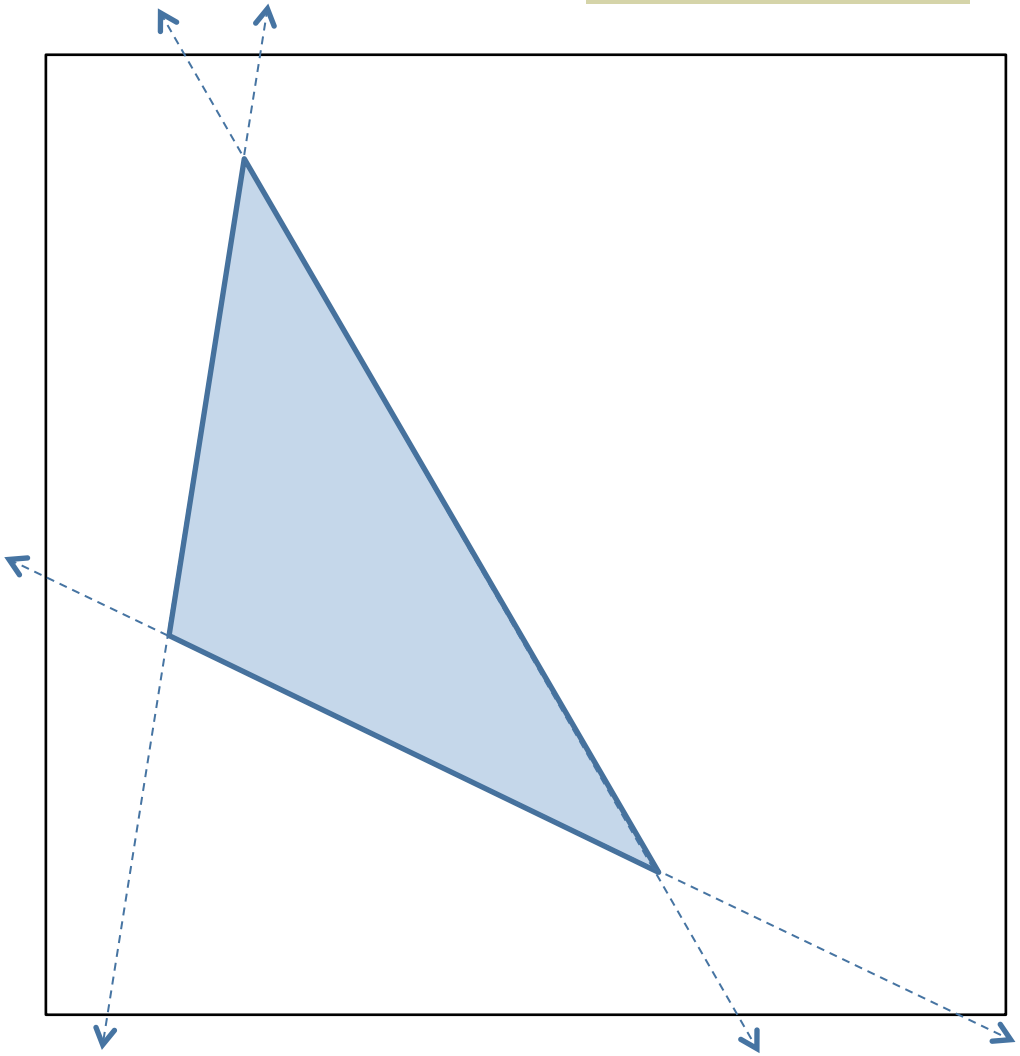
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

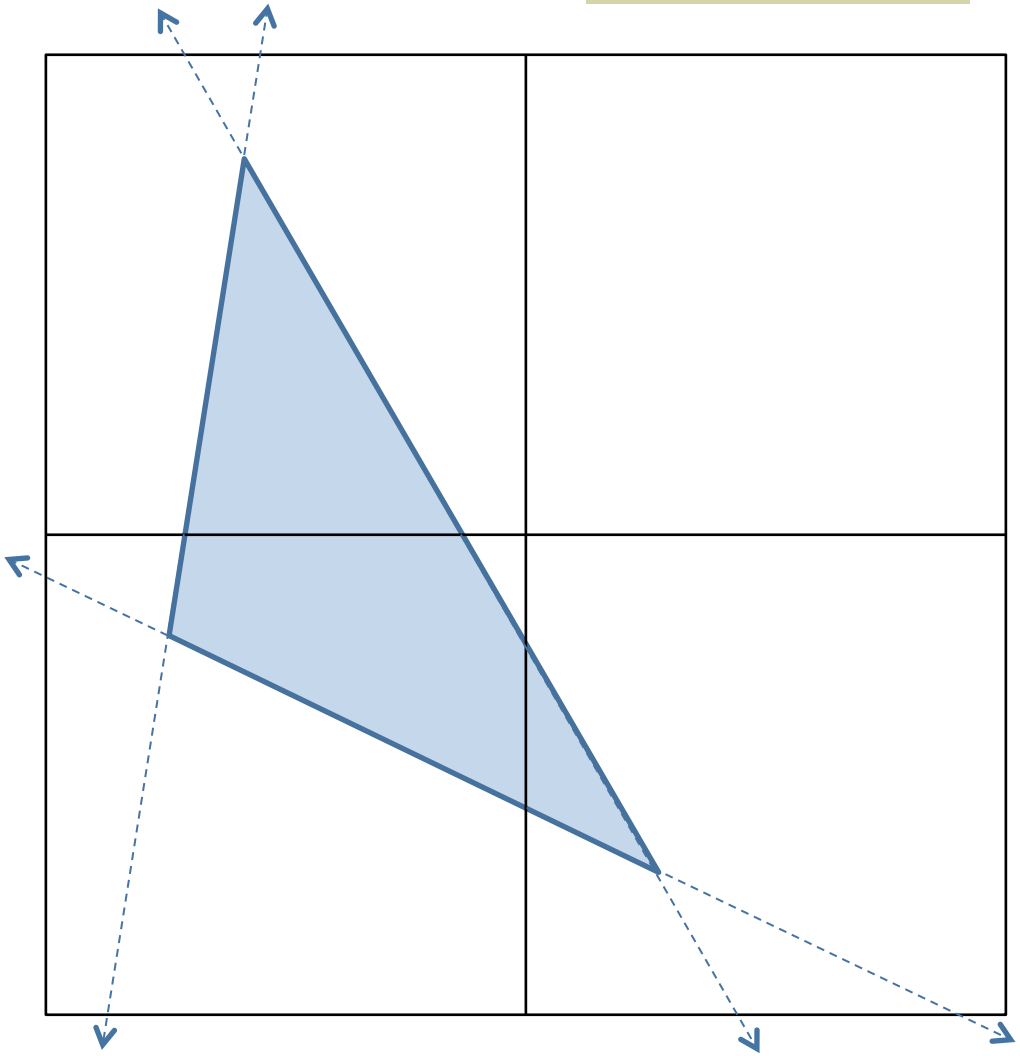
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

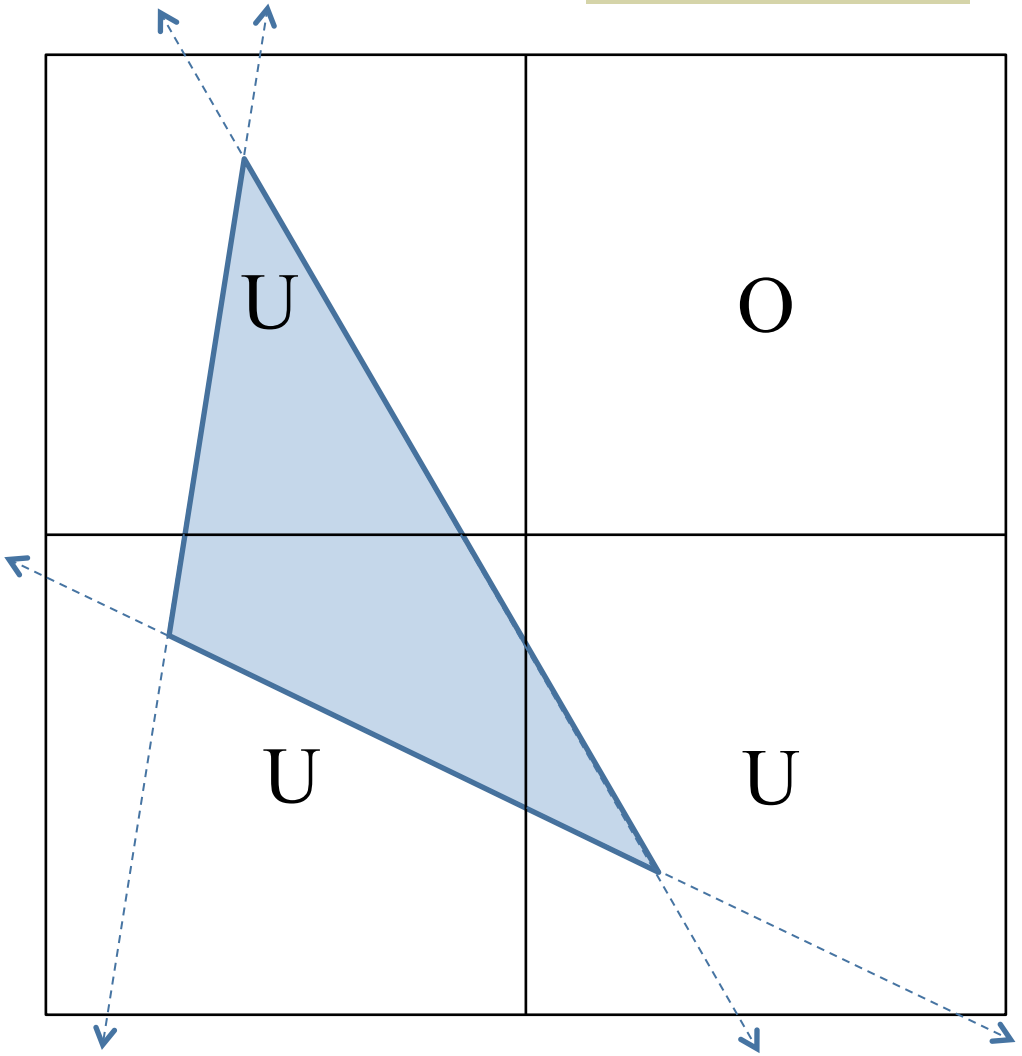
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

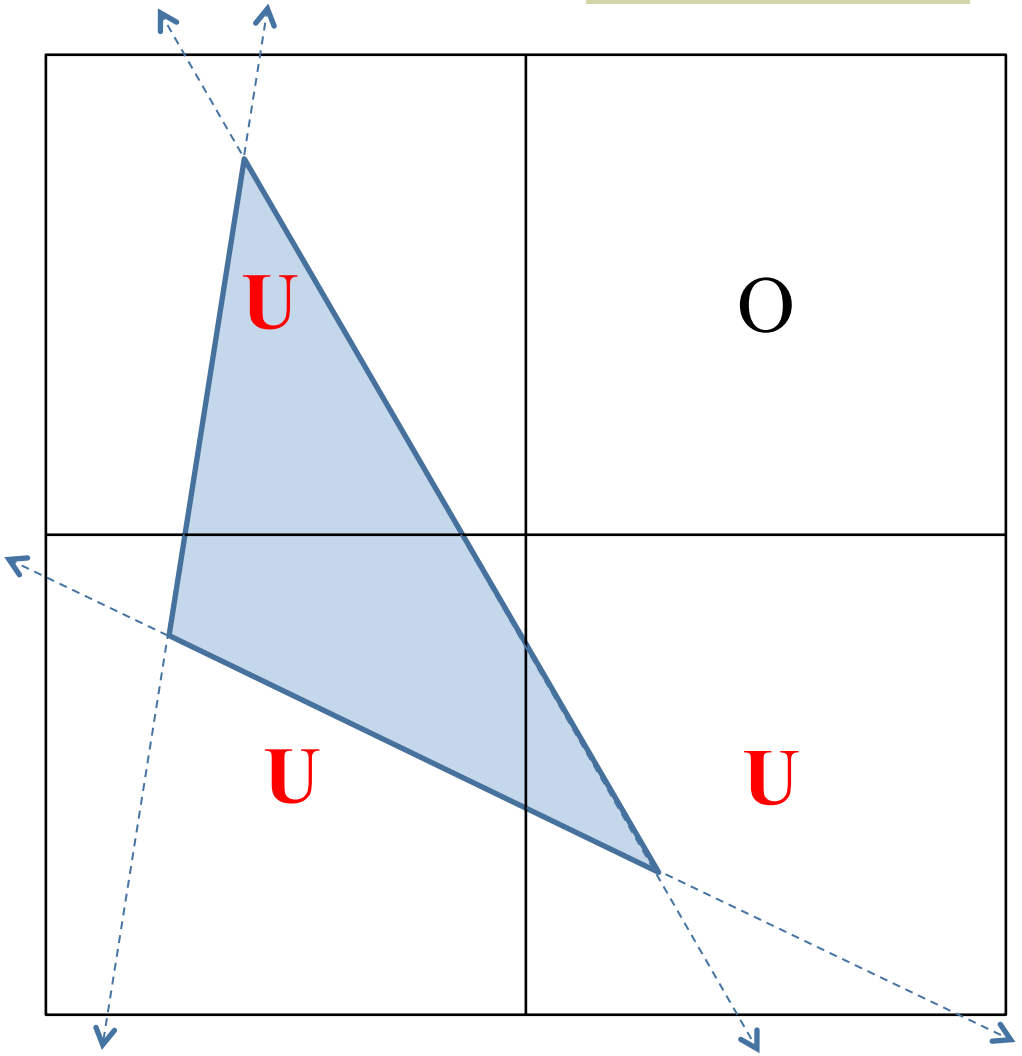
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

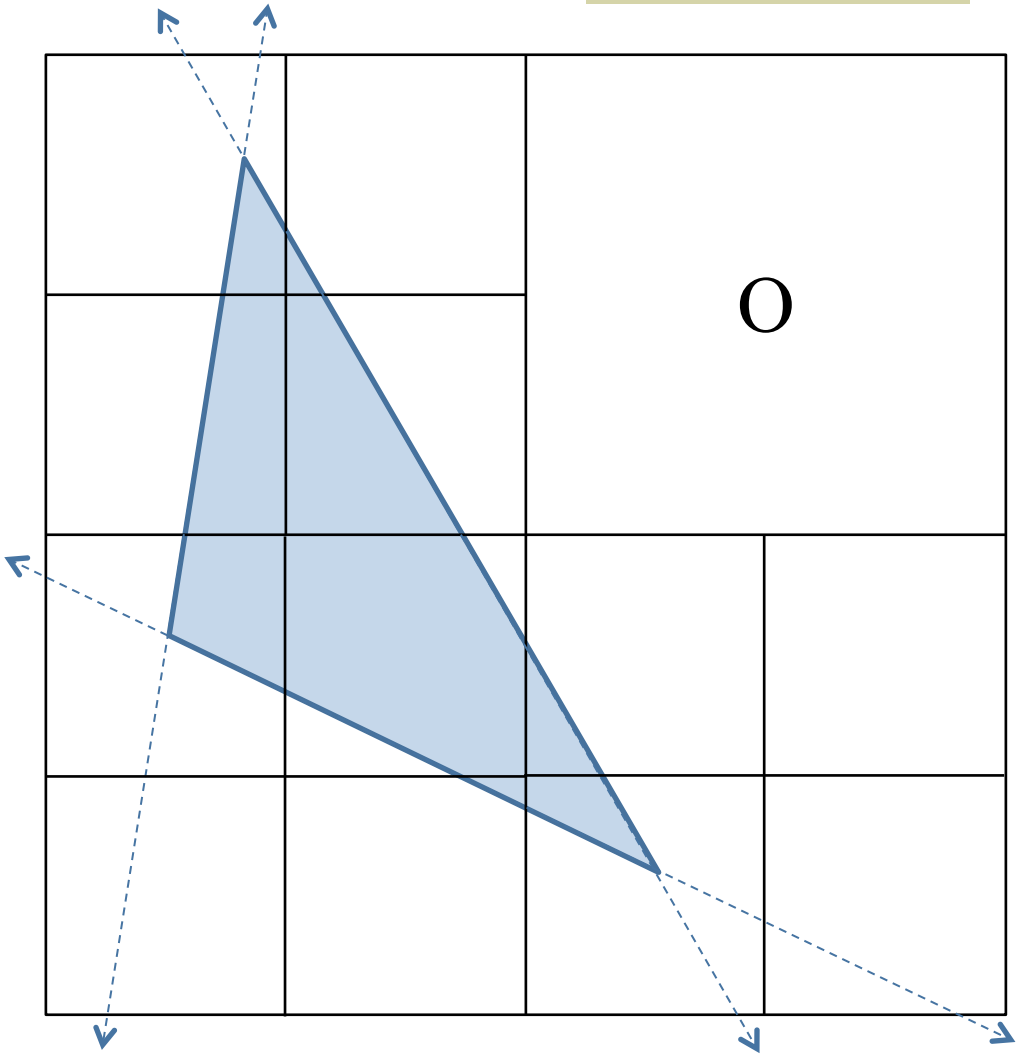
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

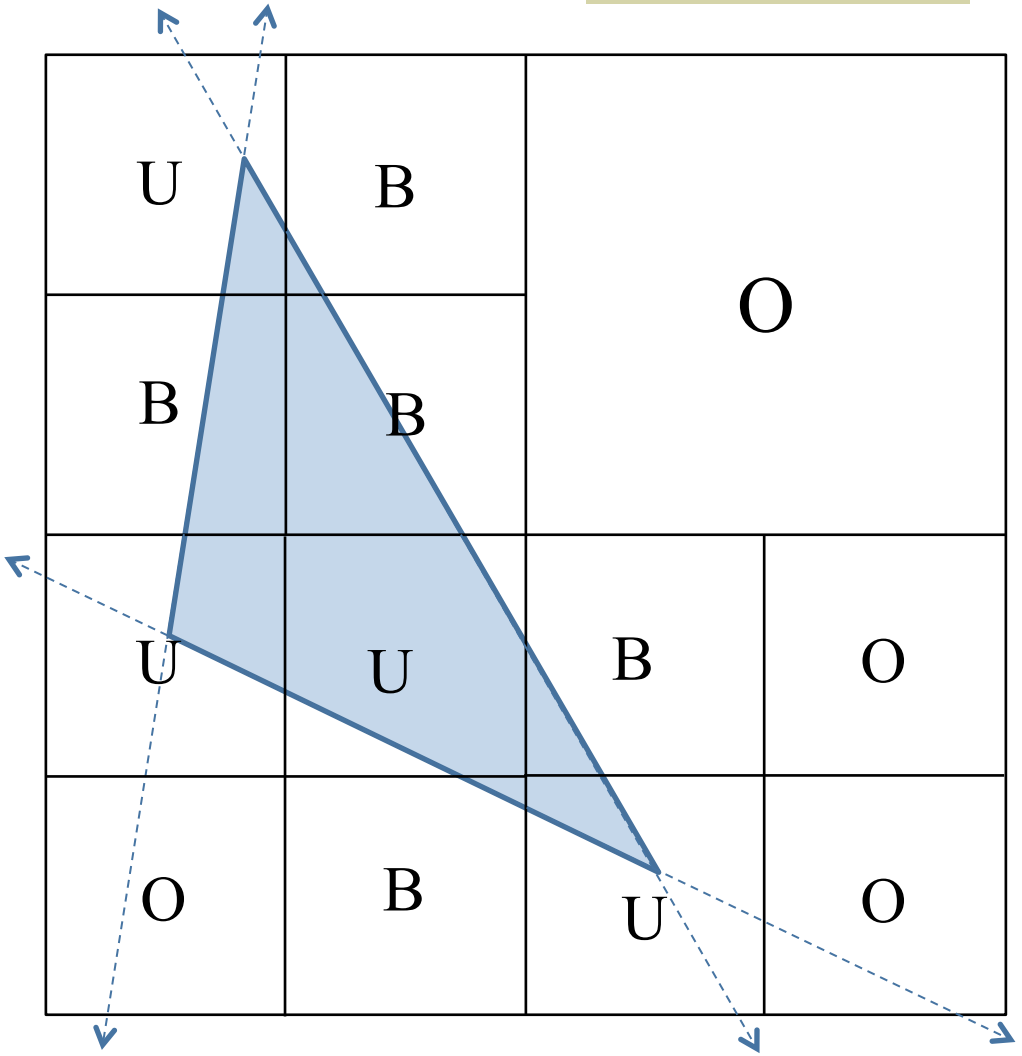
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

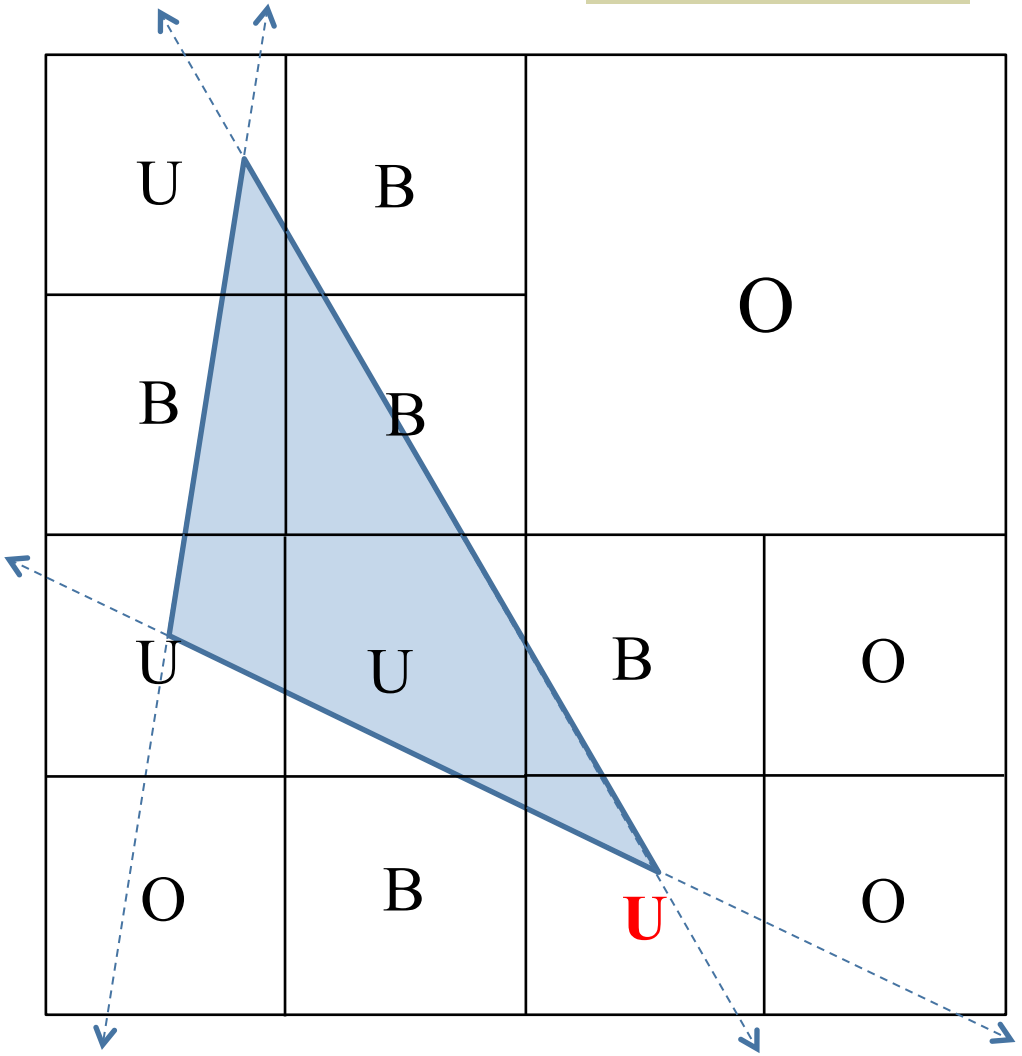
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

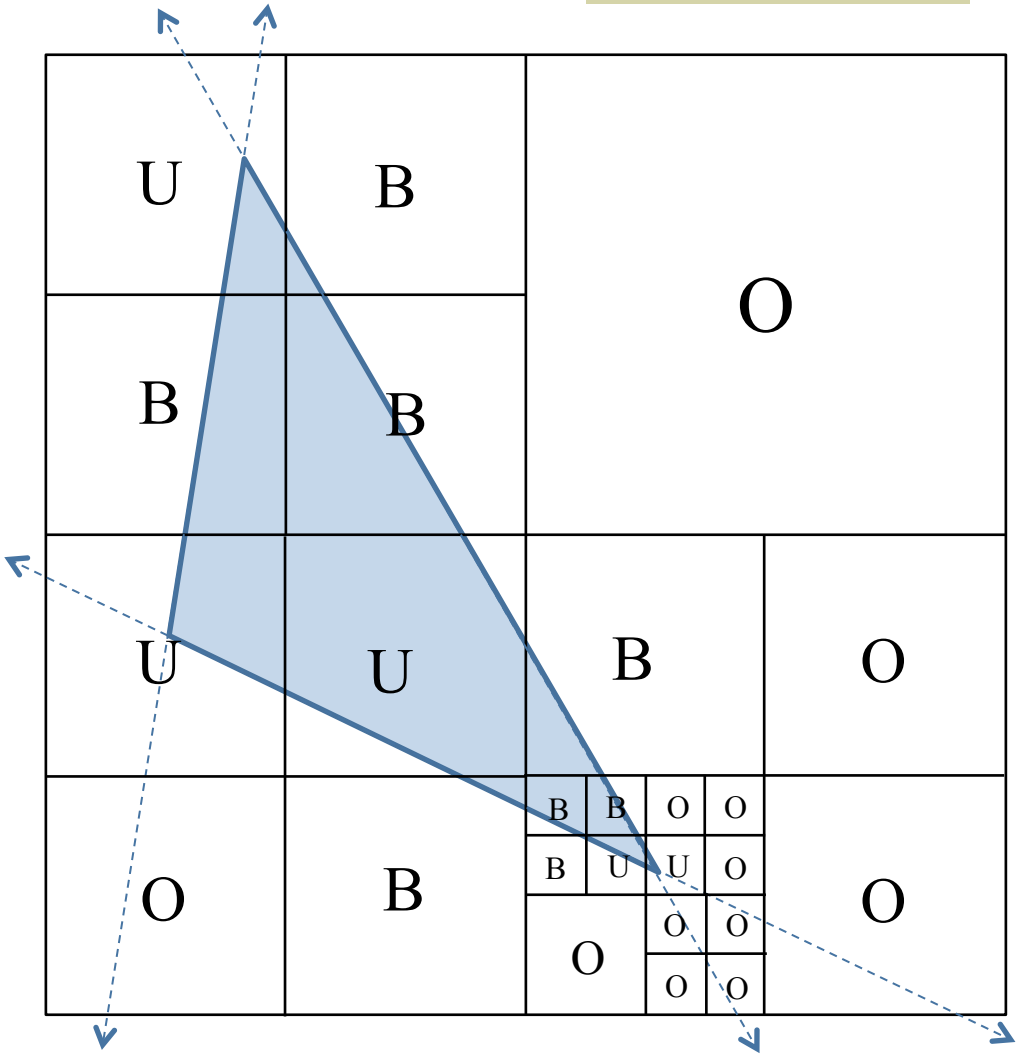
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

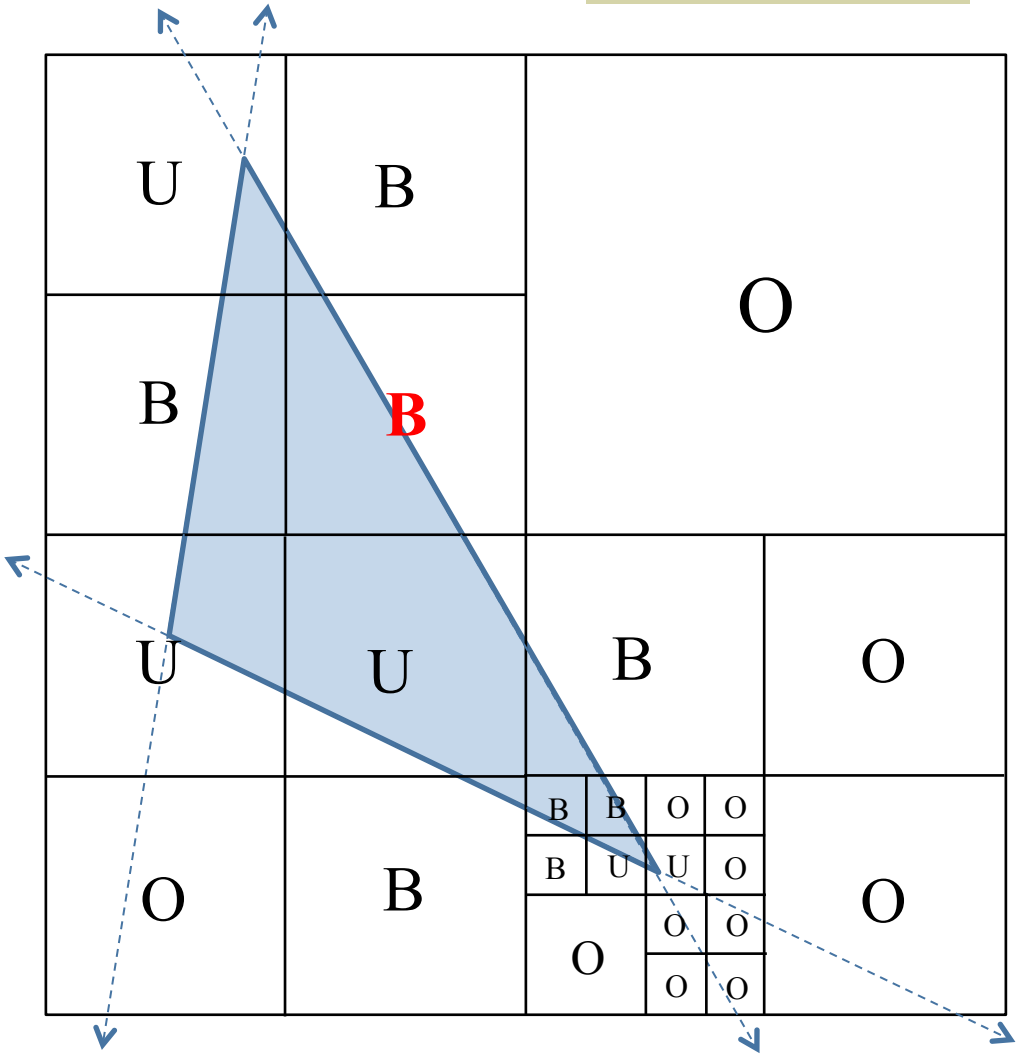
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

(1) Given an initial (very large) bounding box

(2) Traverse an octree:

(a) Subdivide initial box into sub-boxes

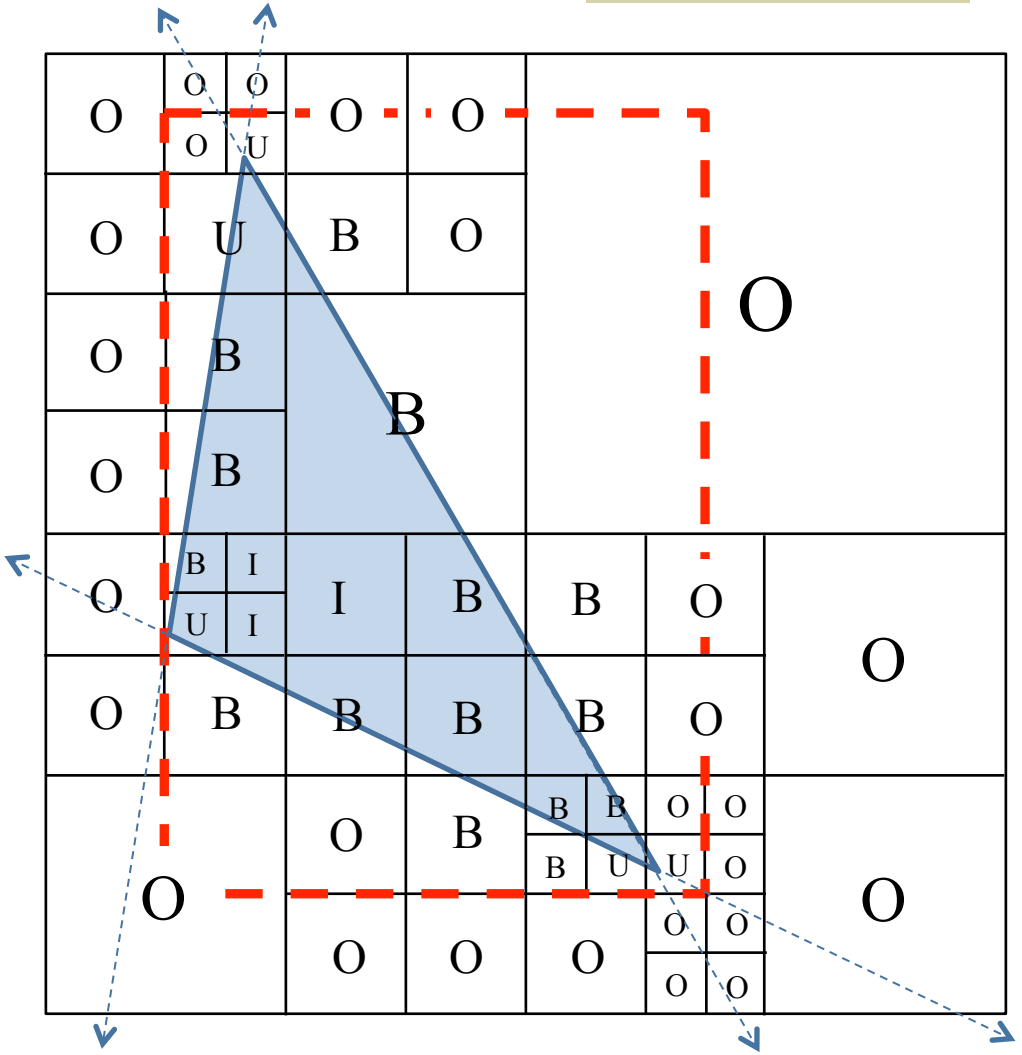
(b) For each sub-box:

(i) *classify* sub-box as *Inside*, *Outside*, *Boundary*, or *Unknown*

(ii) subdivide *Unknown* & *Boundary* sub-boxes

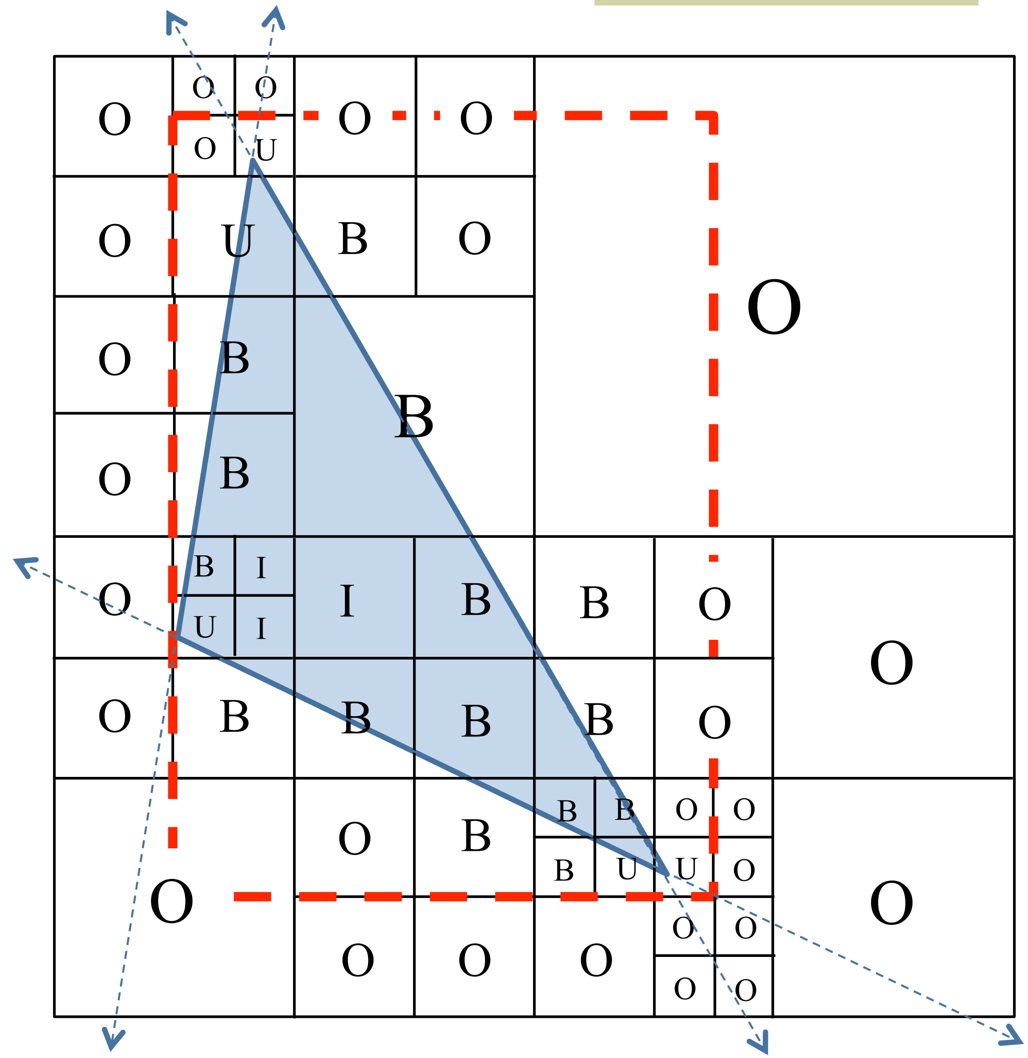
…

(3) Terminates once ε-box is found



Algorithm Overview (D&C)

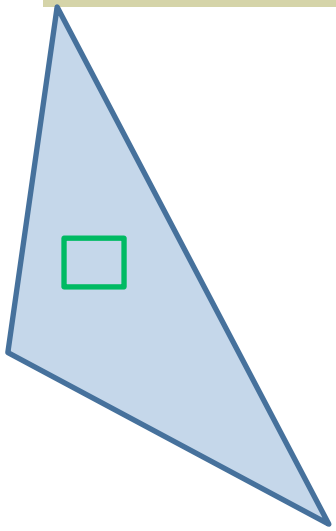
The crux
of this algorithm is
the *classify* operation



The *classify* Operation Overview

Given comp *C* and axis-aligned box *B*, *classify(C,B)*, returns:

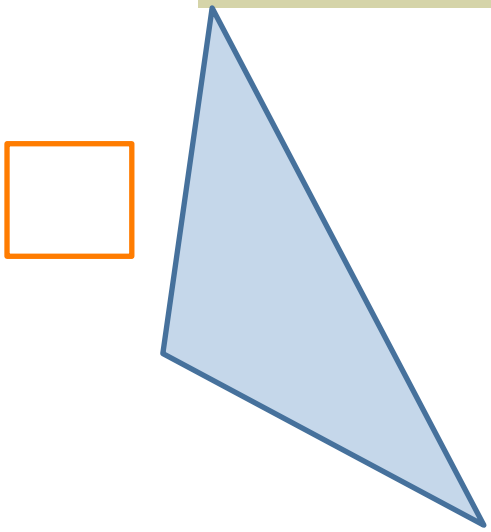
- ***Inside***
- *Outside*
- *Boundary*
- *Unknown*



The *classify* Operation Overview

Given comp *C* and axis-aligned box *B*, *classify(C,B)*, returns:

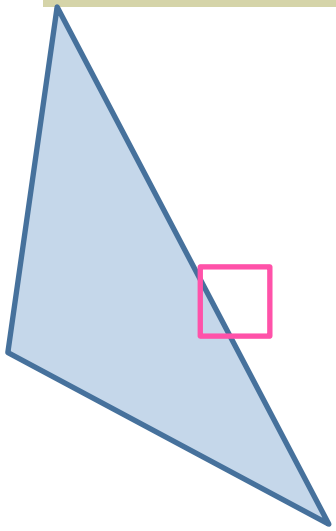
- *Inside*
- ***Outside***
- *Boundary*
- *Unknown*



The *classify* Operation Overview

Given comp *C* and axis-aligned box *B*, *classify(C,B)*, returns:

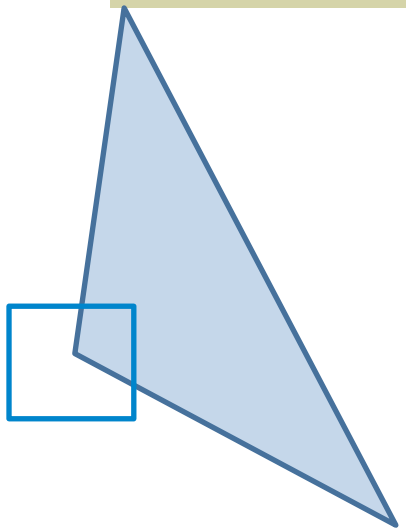
- *Inside*
- *Outside*
- ***Boundary***
- *Unknown*



The *classify* Operation Overview

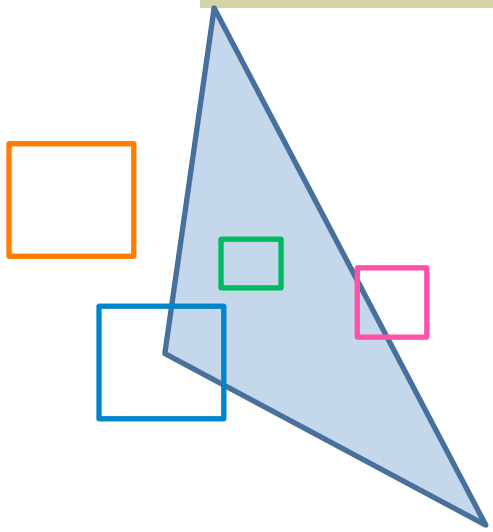
Given comp *C* and axis-aligned box *B*, *classify(C,B)*, returns:

- *Inside*
- *Outside*
- *Boundary*
- ***Unknown***



The *classify* Operation

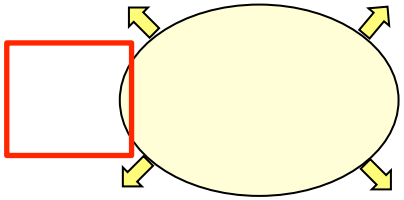
Given comp *C* and axis-aligned box *B*, *classify(C,B)*, returns:



- *Inside* $\Rightarrow B \subseteq C$
- *Outside* $\Rightarrow B \cap C = \varnothing$
- *Boundary* $\Rightarrow \exists$ points $p, q \in B$ with $p \in C$ and $q \notin C$
- *Unknown* $\Rightarrow$ could not classify

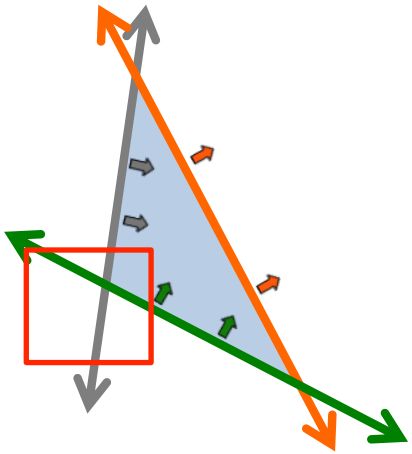
Operations used for *classify*

Let *b* be an axis aligned box:



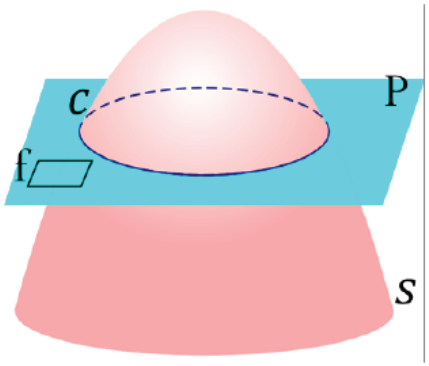
- *boxLabel* – given a surface *S*, return if the points of *b* are inside, outside, or both with respect to *S*.

- *formulaRestriction* – given a Boolean formula *G* and the classification for all surfaces of *G* for *b*, replace all surfaces of *G* in which *b* is completely inside or outside with ***T*** or ***F*** and simplify.



The *boxLabel* Operation [M12]

Test if a face *f* intersects *s*.



Let *c* be the intersection curve of the plane *P* containing *f* and *s*.

$$c(x,y) = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} ① & ① & ② \\ ① & ① & ② \\ ② & ② & ③ \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

To determine if *s* intersects *f*, test properties of the matrix.

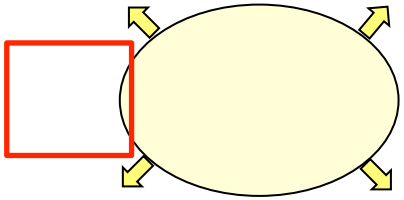
Test if *c* is an ellipse: $\operatorname{sign}\left(\begin{vmatrix} ① & ① \\ ① & ① \end{vmatrix}\right) = \operatorname{sign}(②)$

Test if *c* is real or img: $\operatorname{sign}\left(\begin{vmatrix} ① & ① & ② \\ ① & ① & ② \\ ② & ② & ③ \end{vmatrix}\right) = \operatorname{sign}(⑤)$

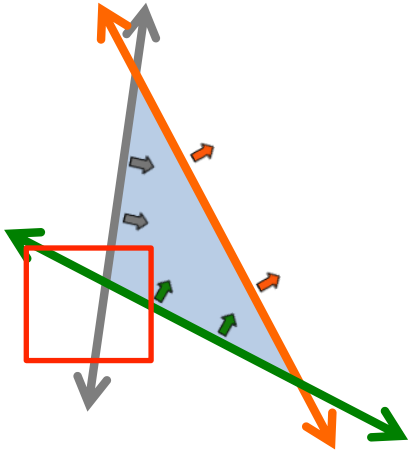
Operations used for *classify*

Let *b* be an axis aligned box:

- *boxLabel* – given a surface *S,* return if the points of *b* are inside, outside, or both with respect to *S*.

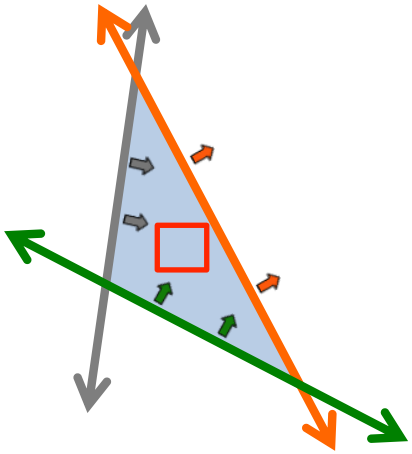


- *formulaRestriction* – given a Boolean formula *G* and the classification for all surfaces of *G* for *b*, replace all surfaces of *G* in which *b* is completely inside or outside with ***T*** or ***F*** and simplify.



The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$



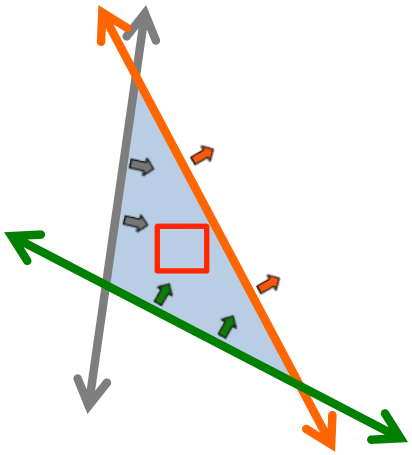
formulaRestriction –

Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$

$$\wedge \qquad \wedge$$

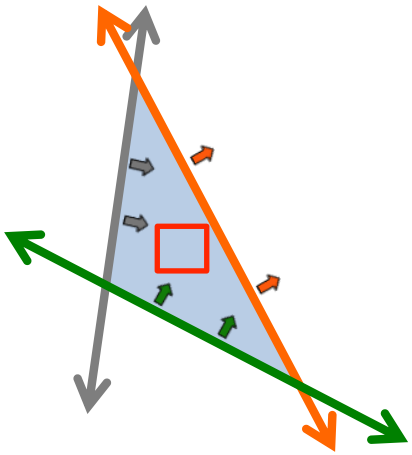


formulaRestriction –

Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$
$$T \quad \wedge \qquad\qquad \wedge$$

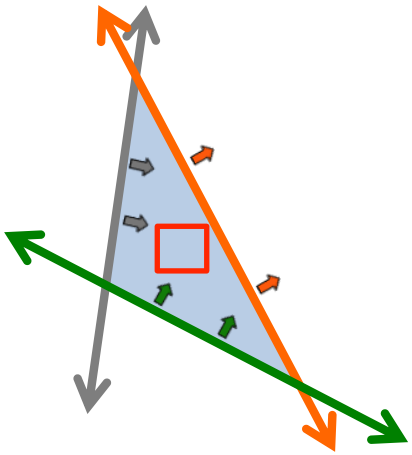


formulaRestriction –

Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$
$$T \wedge T \wedge T$$



formulaRestriction –

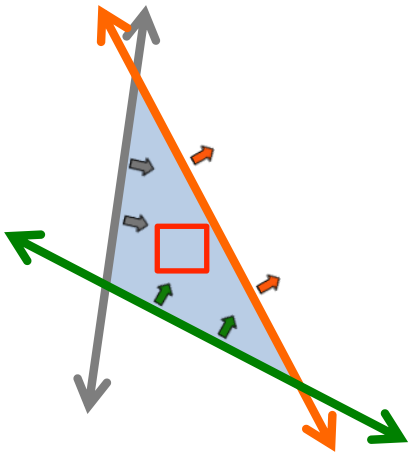
Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$

$$T \wedge T \wedge T$$

$$T$$

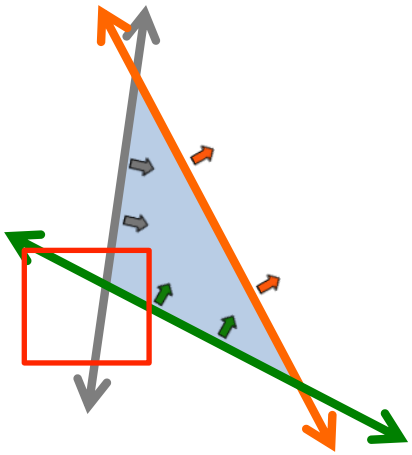


formulaRestriction –

Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$



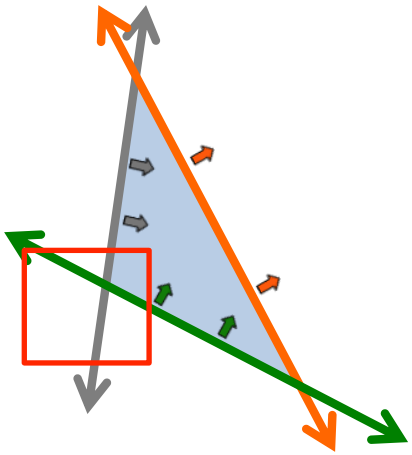
formulaRestriction –

Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$

$$S_{grey} \wedge S_{green} \wedge \quad T$$



formulaRestriction –

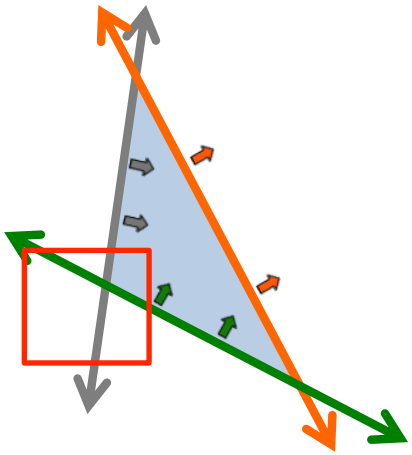
Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *formulaRestriction* Operation

$$S_{grey} \cap S_{green} \cap -S_{orange}$$

$$S_{grey} \wedge S_{green} \wedge \quad T$$

$$S_{grey} \cap S_{green}$$

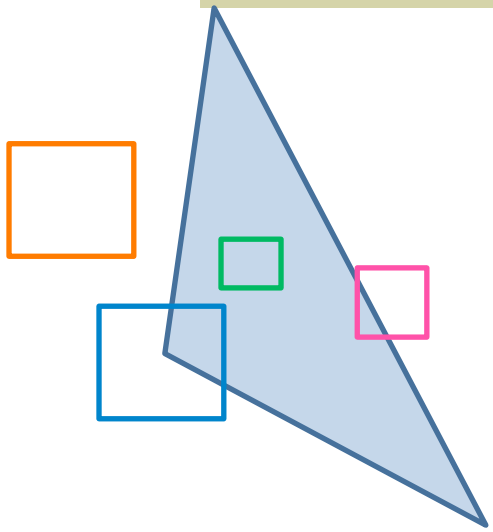


formulaRestriction –

Let *b* be an axis-aligned box,
given a Boolean formula *G* and
the classification for all surfaces of *G* for *b*,
replace all surfaces of *G* in which
b is completely inside or outside with ***T*** or ***F*** and simplify.

The *classify* Operation

Given comp *C* and axis-aligned box *B*, *classify(C,B)*, returns:

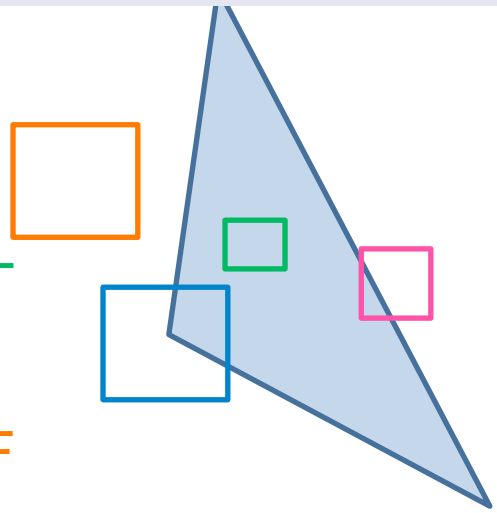


- *Inside* $\Rightarrow B \subseteq C$
- *Outside* $\Rightarrow B \cap C = \varnothing$
- *Boundary* $\Rightarrow \exists$ points $p, q \in B$ with $p \in C$ and $q \notin C$
- *Unknown* $\Rightarrow$ could not classify

The *classify* Operation

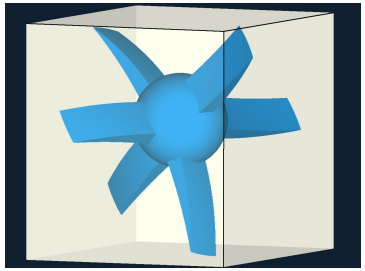
Using *boxLabel* and *formulaRestriction* we implement *classify* as:

Given comp *C* and axis-aligned box *B*,
classify(C,B), returns:

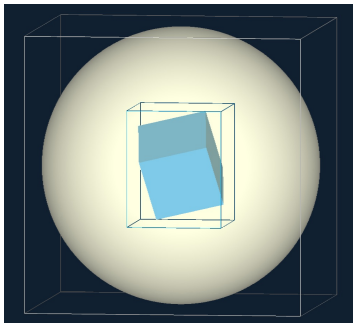


- *Inside* ⇔ Formula resolved to T
- *Outside* ⇔ Formula resolved to F
- *Boundary* ⇔ Formula resolved to 1 surface
  (or strengthen by cherry picking special cases that are commonly modeled [NMGG13])
- *Unknown* ⇔ could not classify

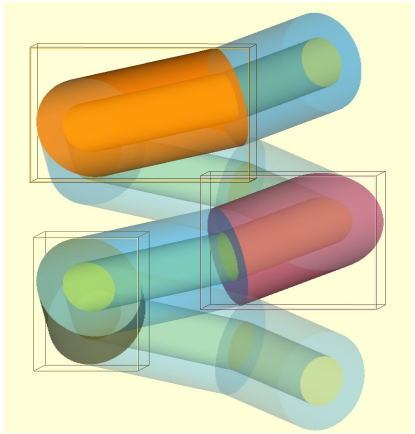
Experiment: Models



SpikeyBall – each spike is formed by the intersection of three planes and two paraboloids.
The sharp features cause stochastic and sampling based algorithms produce a box that is too tight.

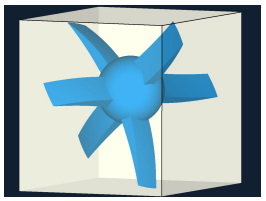


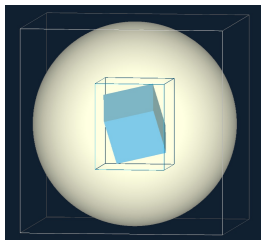
RotatedCube – a rotated cube inside of a sphere.
It is easy to verify that the computed box is actually an epsilon box.



HelicalPipe20 – a helical section of piping.
A model with multiple levels of hierarchy.

Experiment: 1. Compute AABB



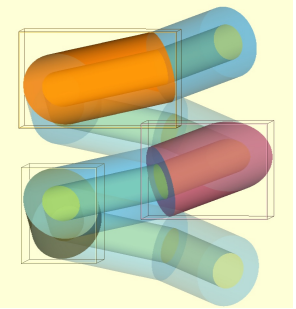


Comp ID			Time (s) for	
			ε = 0.5	ε = 0.05
SpikeyBall				
C0			0.60	1.67
RotatedCube				
C0			0.02	0.10
	C1		<.01	<.01
Total			*0.02*	*0.10*

Initial bounding box:
Min point: (-1000, -1000, -1000)
Max point: (1000, 1000, 1000)

Experiment: 1. Compute AABB

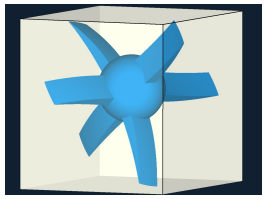
Comp ID			Time (s) for ε = 0.5	Time (s) for ε = 0.05
HelicalPipe20				
C0			0.13	1.62
	C1		0.02	0.25
		C11	0.03	0.19
	C2		0.02	0.39
		C12	0.05	0.36
	C3		0.02	0.63
		C13	0.03	0.22
⋮			⋮	
Total			*0.75*	*8.53*

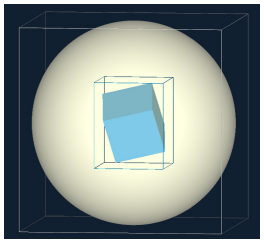


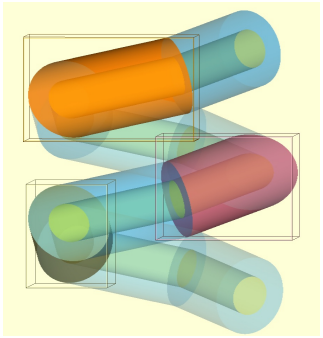
What is time consuming:

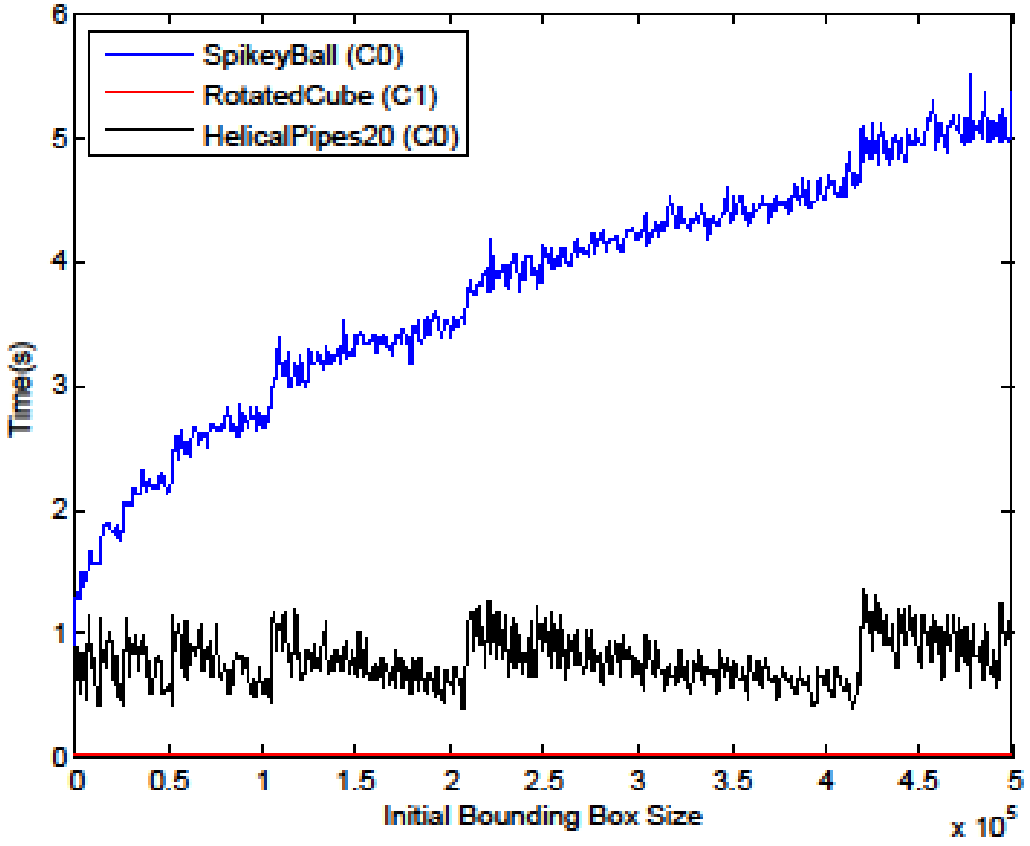
- The size of reducing from the initial bounding box to a tight bounding box.
- Tightening to a smaller ε.

Experiment: 2. Initial AABB



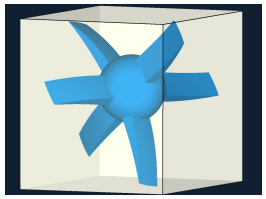


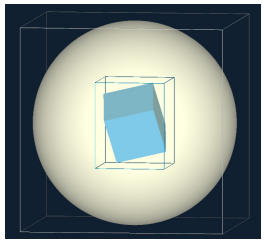


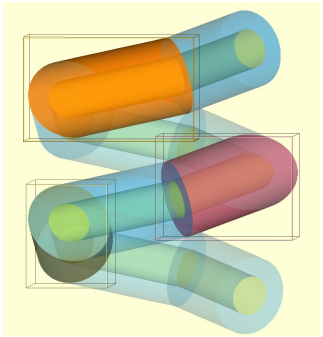


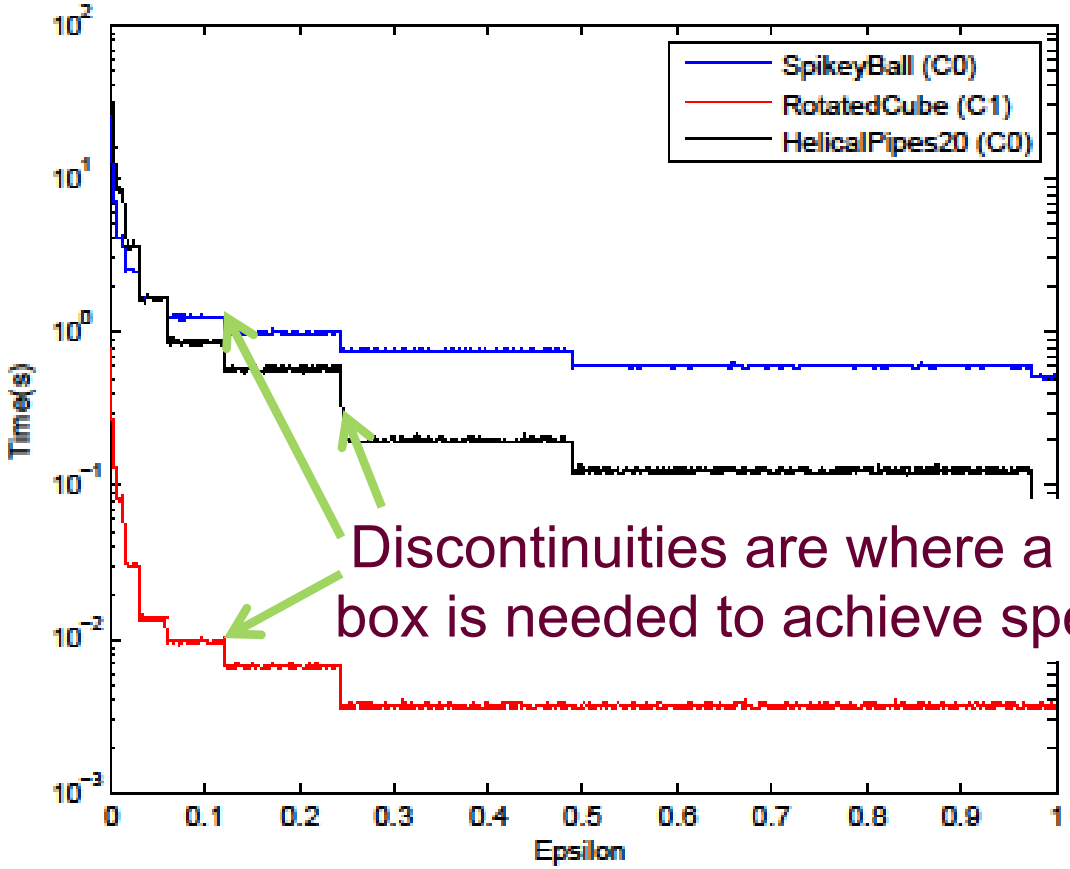
Computing the bounding box is practical
even if we must reduce by 4 orders of magnitude.

Experiment: 3. Tolerance





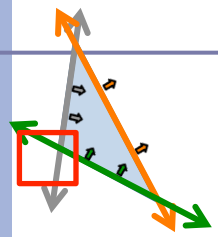




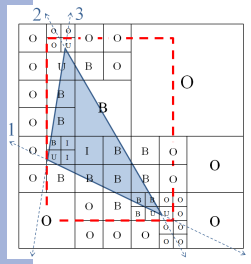
Discontinuities are where a smaller box is needed to achieve specified ε

Empirically, it takes O(1/ε) to compute an ε box.

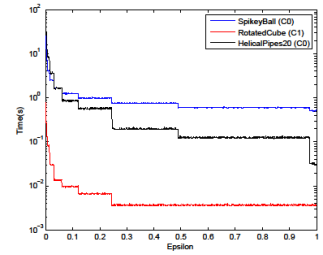
Conclusion



Described an operation for testing if an axis-aligned box contains the boundary of a component



Described a divide-and-conquer framework for computing numerically-optimal bounding boxes



Experiments suggest that algorithm could be routine pre-processing for CSG components.

Extrapolating from experiments,
one million comps on 100 CPUs in about 5.5 min

MC21

Contact:
Author: David L. Millman
Email: david.millman@unnpp.gov

Bibilography

[LG98] M. C. Lin and S. Gottschalk, “Collision Detection Between Geometric Models: A Survey,” (1998).

[DLL+08] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean, “Near-optimal parameterization of the intersection of quadrics: I. The generic algorithm,” Journal of Symbolic Computation, 43, 3, 168-191 (2008).

[SW06] E. Schömer and N. Wolpert, “An exact and ecient approach for computing a cell in an arrangement of quadrics,” Computation Geometry Theory and Applications, 33, 1-2, 65-97 (2006).

[K00] J. Keyser, “Exact Boundary Evaluation for Curved Solids”, PhD thesis, University of North Carolina-Chapel Hill, 2000.

[MTT05] B. Mourrain, J.-P. Técourt, and M. Teillaud, “On the Computation of an Arrangement of Quadrics in 3d,” Computational Geometry Theory and Application, 30. 2, 145-164, 2005

[M12] D. L. Millman, “Degree-Driven Design of Geometric Algorithms for Point Location, Proximity, and Volume Calculation”, PhD thesis, University of North Carolina–Chapel Hill, 2012.

[NMGG13] B. R. Nease, D. L. Millman, D. P. Griesheimer, D. F. Gill, “Geometric Templates for Improved Tracking Performance in Monte Carlo Codes”, SNA+MC 2013