Implementing Algebraic Routines in Exact Solid Modeling

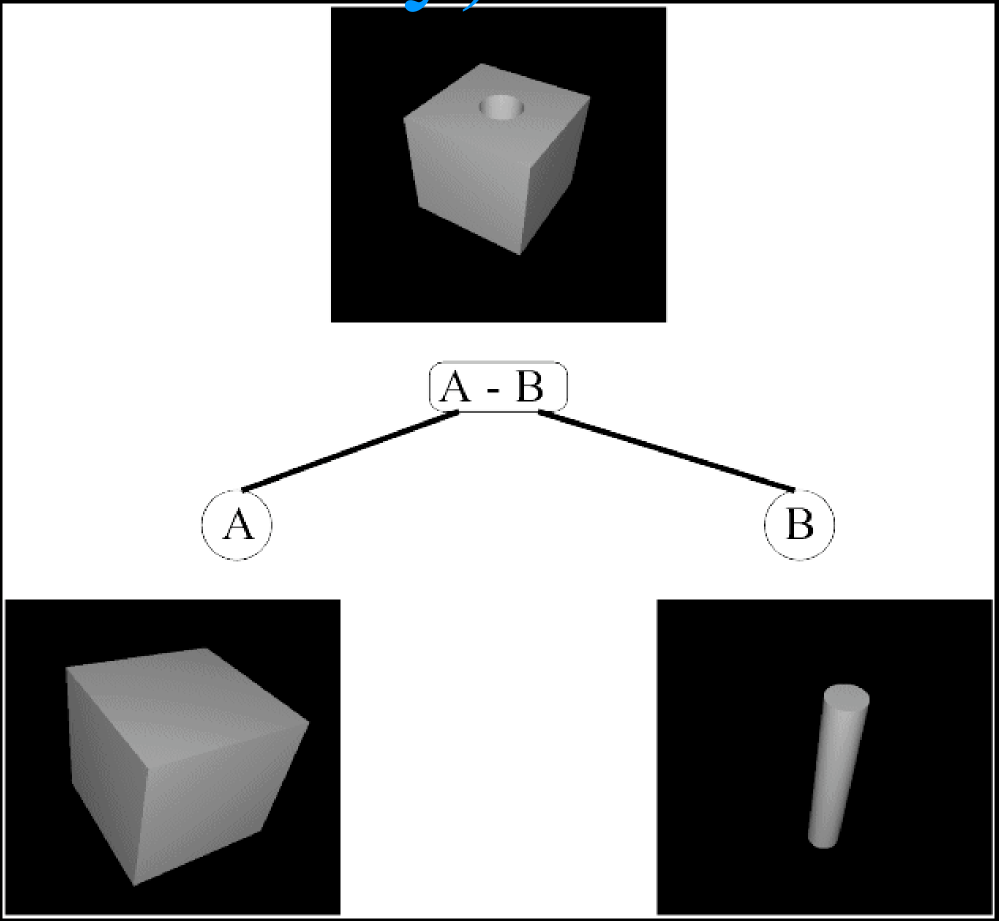
John Keyser
Department of Computer Science
Texas A&M University

Outline

- **Background and Motivation**
  - **Boundary Evaluation**
  - The Robustness Issue
  - Exact Computation
  - Prior Work
- Exact Boundary Evaluation
- Extensions
- Conclusion

CSG (Constructive Solid Geometry)

- Boolean combinations of primitive solids
- Difference, Union, Intersection

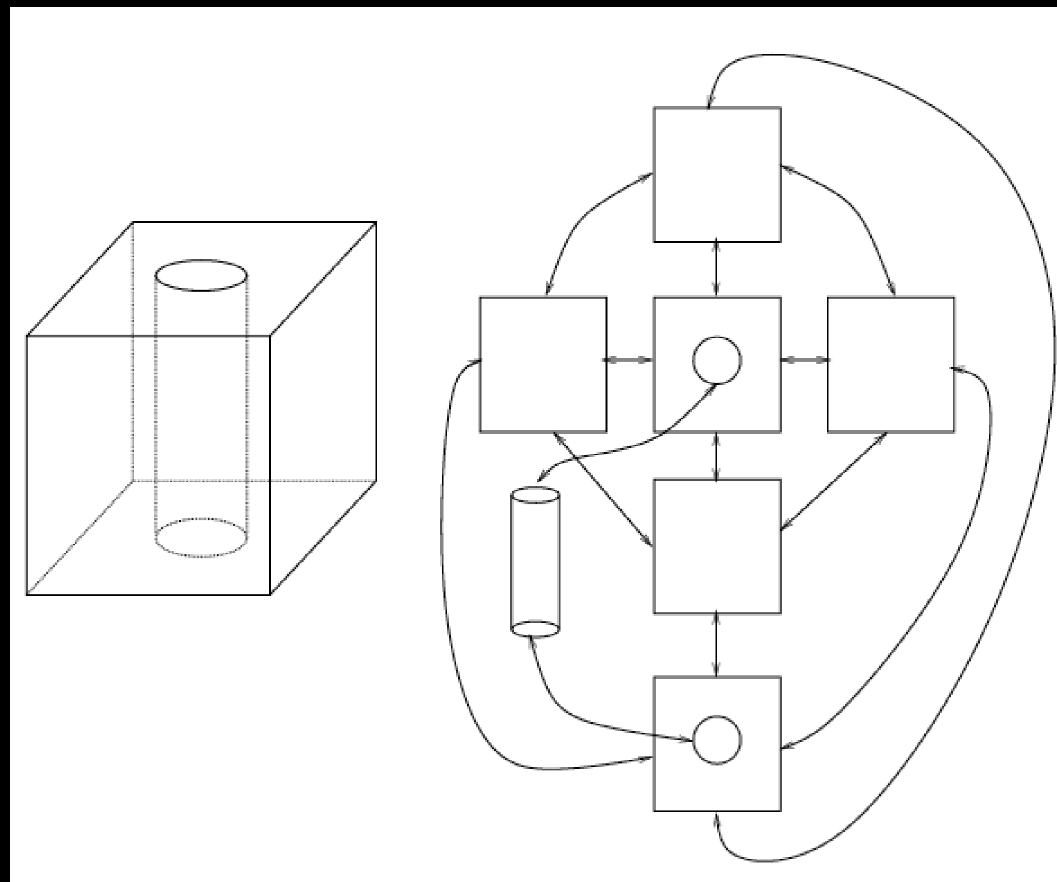


CSG

- Standard primitives:
  - boxes/wedges
  - ellipsoids
  - cones/cylinders
  - tori
- Useful:
  - Design
  - Ray-tracing

B-reps (Boundary Representations)

- Describe the boundary of a solid object



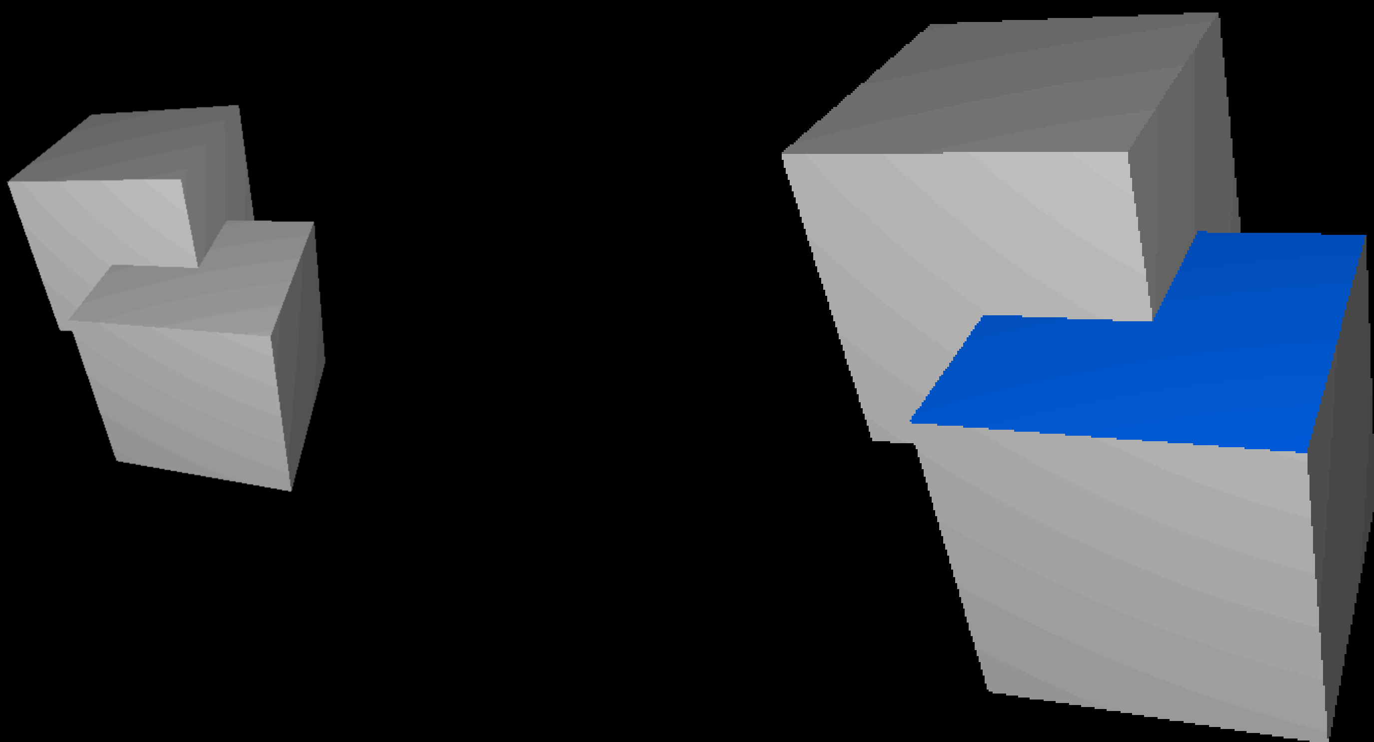
B-reps

- Boundary surfaces usually broken up into patches
  - Polygons, triangles
  - Curved patches (e.g. NURBS)
- Useful:
  - Interactive visualization
  - Mesh generation

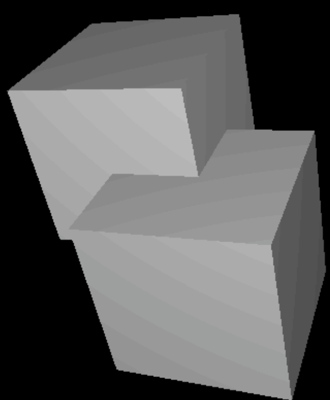
Boundary Evaluation

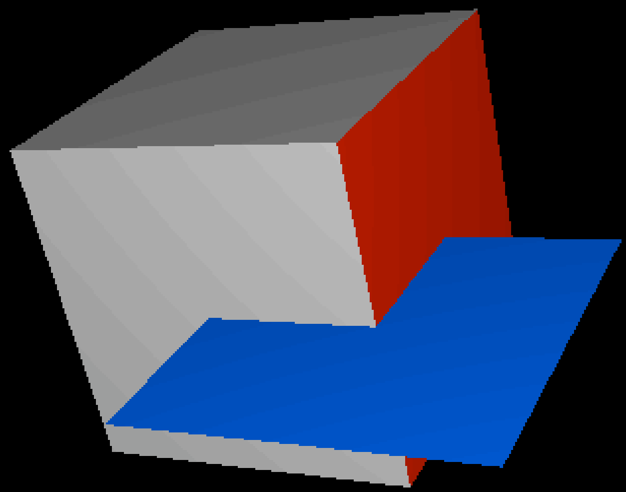
- Converting CSG to B-rep
  - More generally, finding surface after operation
- Primary operations involve intersecting sets of surfaces, curves

Boundary Evaluation: Simple Example

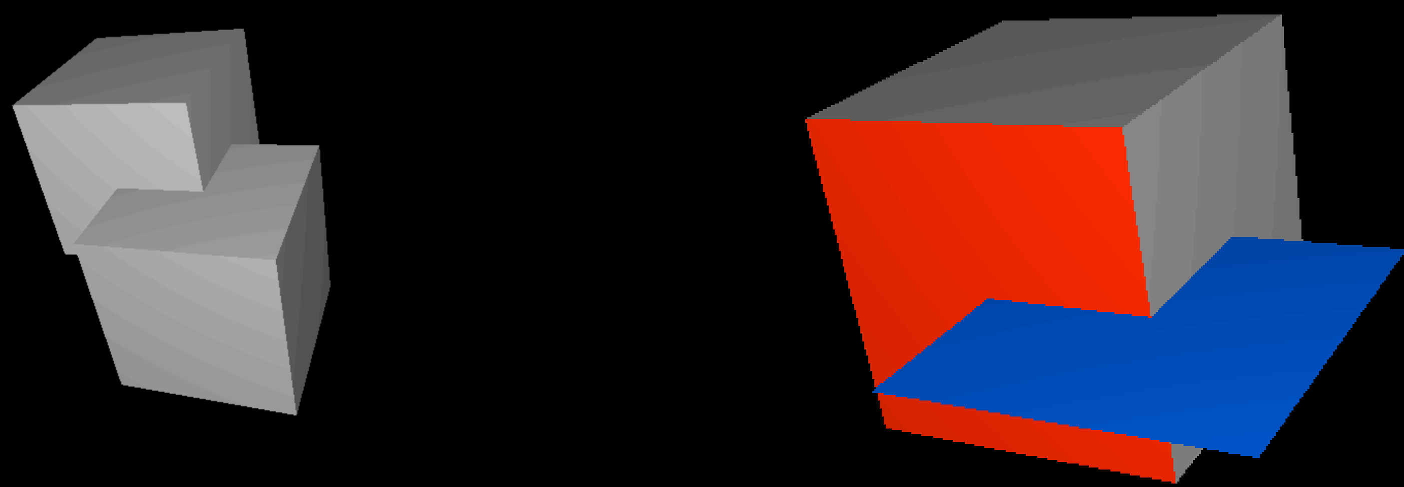


Boundary Evaluation: Simple Example

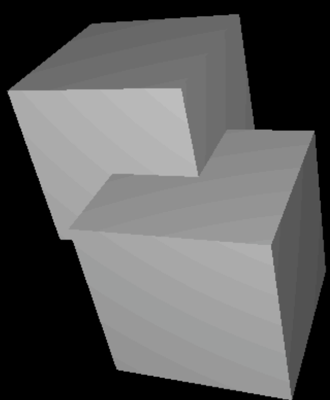


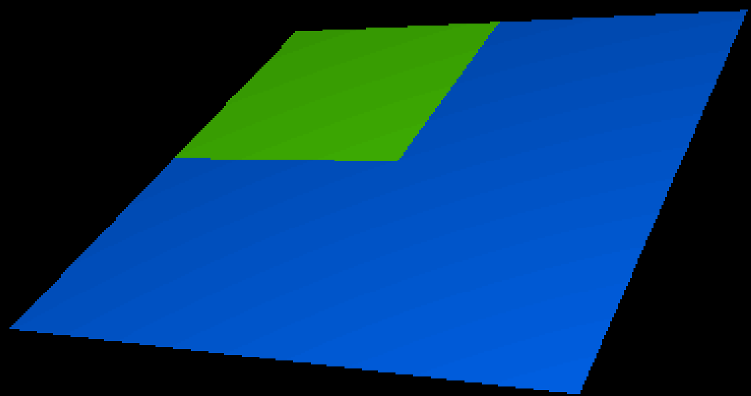


Boundary Evaluation: Simple Example

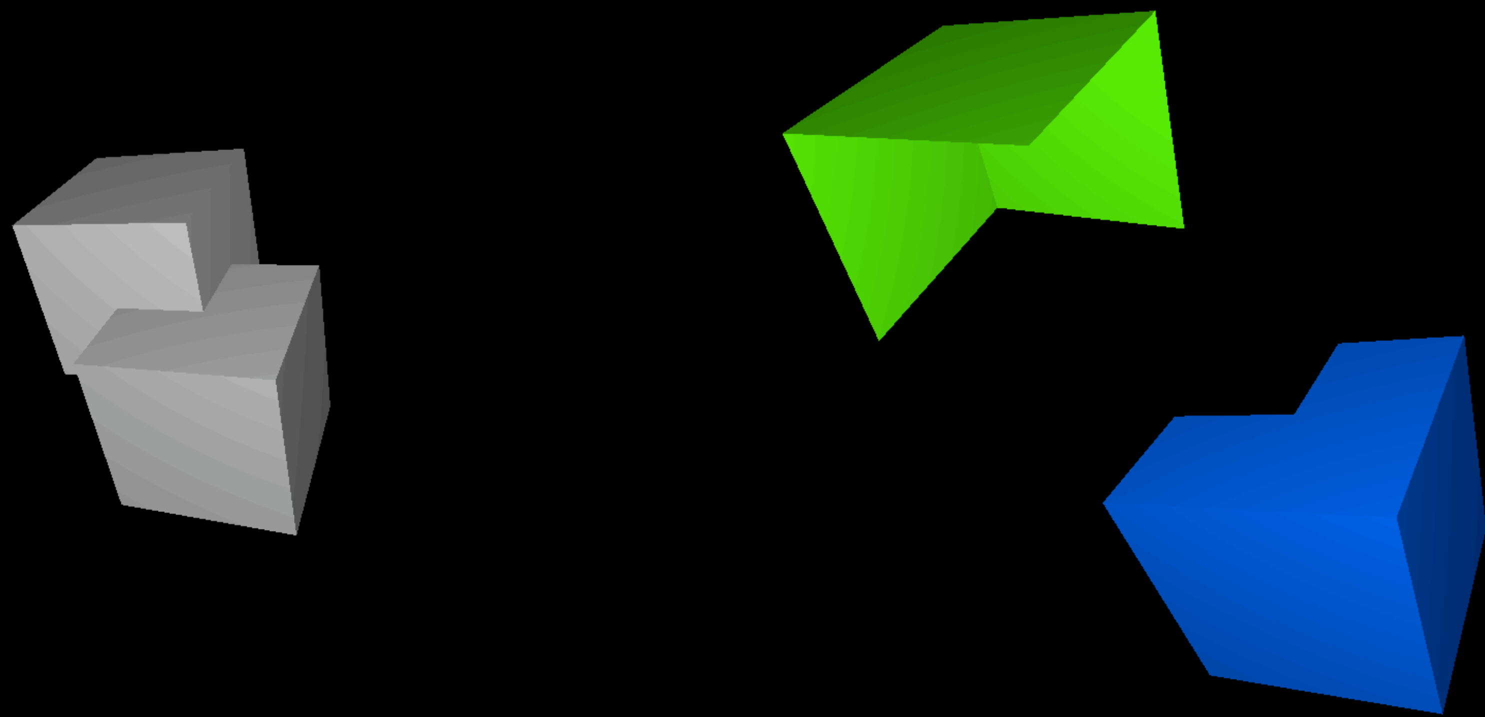


Boundary Evaluation: Simple Example

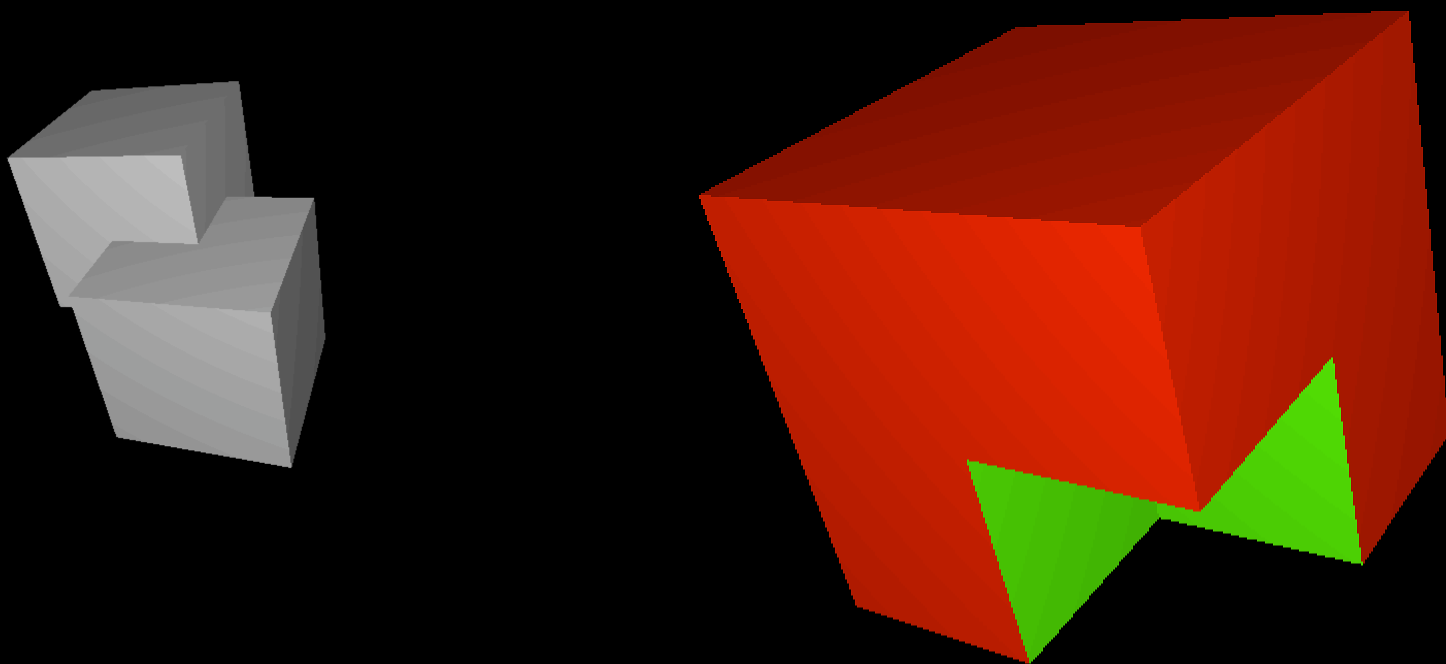




Boundary Evaluation: Simple Example



Boundary Evaluation: Simple Example

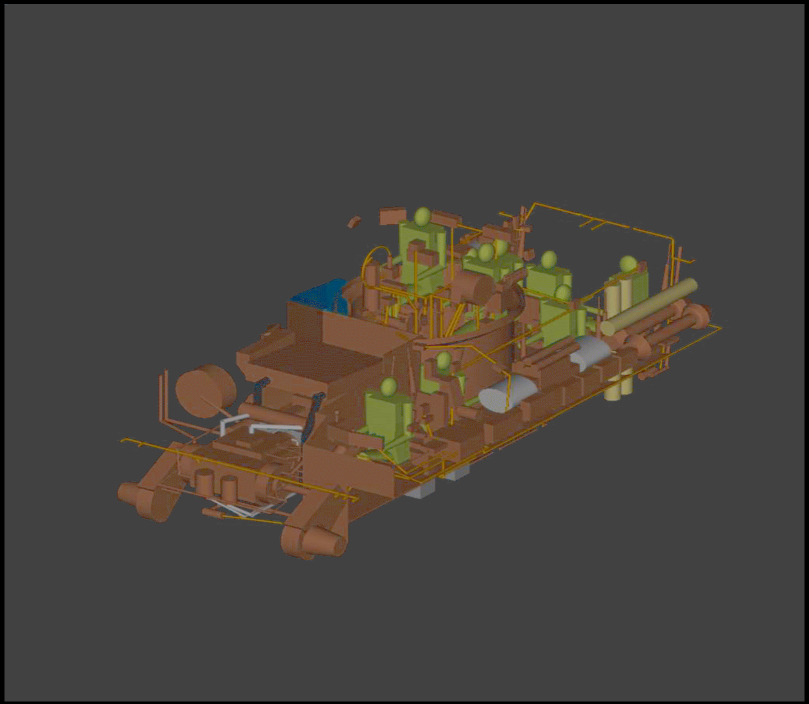


Bradley Fighting Vehicle (exterior)

- BRL-CAD
- Primitives:
  - Polyhedra
  - generalized cones
  - ellipsoids
  - tori

Bradley Fighting Vehicle (interior)

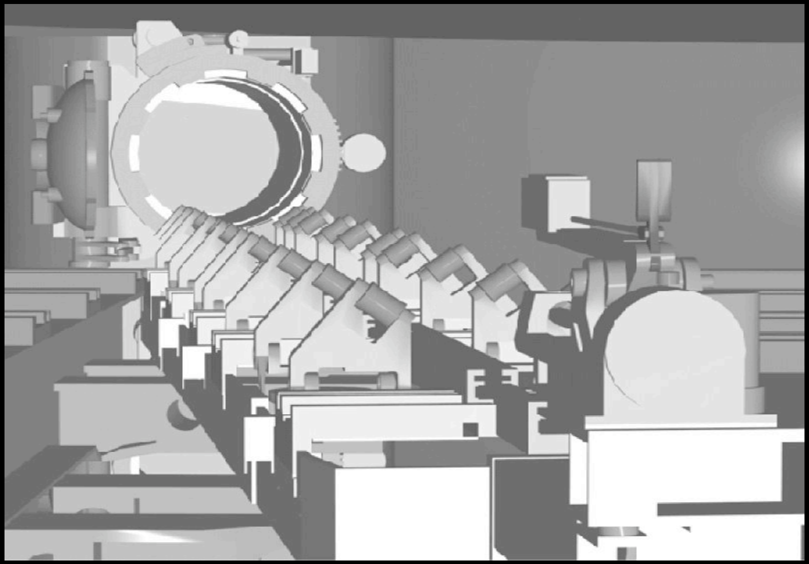
- 2725 objects
- Each 0 to 20+ CSG operations



Submarine Storage Room

- Submarine Storage Room
  - Over 5000 Solids
  - Low-degree

- Model courtesy of Electric Boat, a division of General Dynamics



Outline

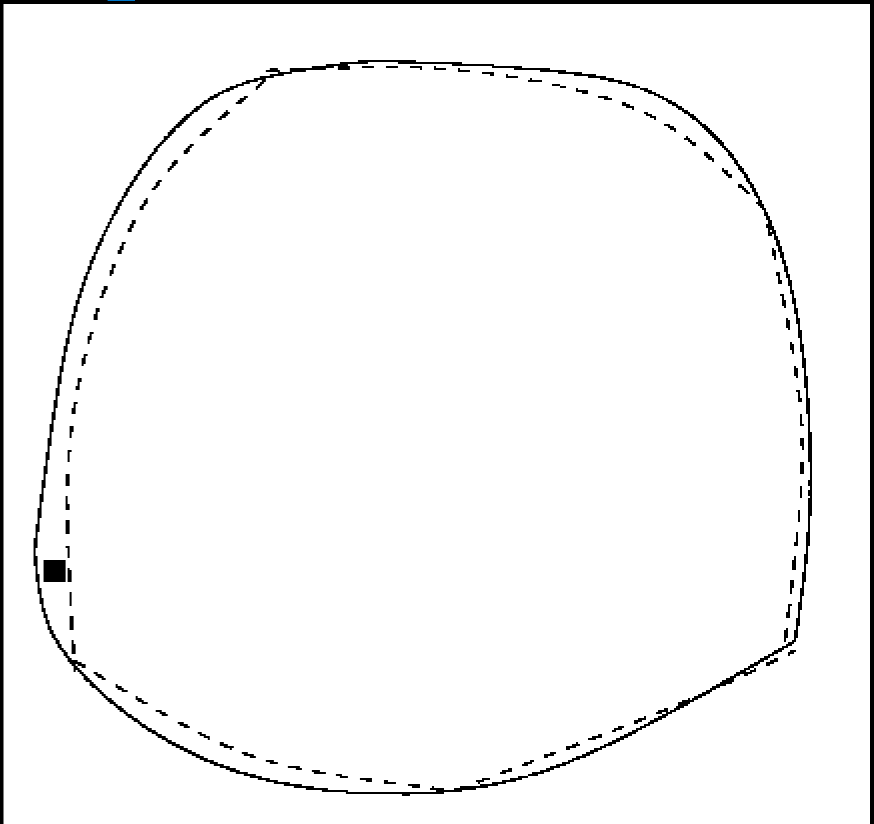
- **Background and Motivation**
  - Boundary Evaluation
  - **The Robustness Issue**
  - Exact Computation
  - Prior Work
- Exact Boundary Evaluation
- Extensions
- Conclusion

Robustness

- Algorithm fails on input data
  - Serious problem for geometric algorithms
- Two major sources
  - Numerical Error
  - Degenerate Data
- Curved surfaces magnify the problem

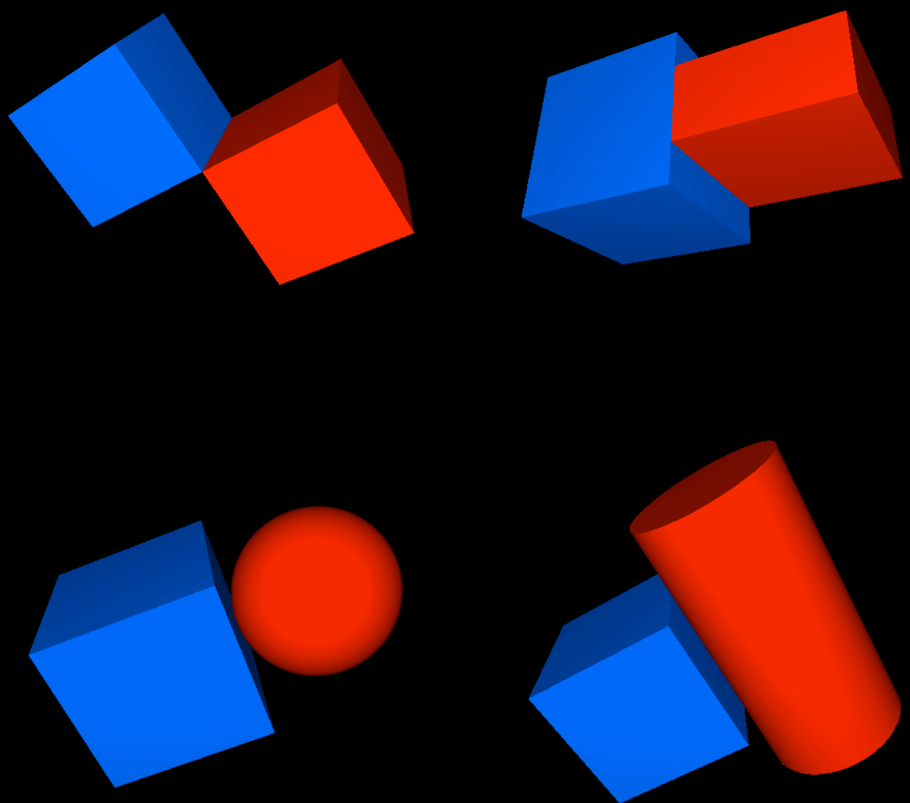
Robustness problems

- Numerical error
  - Initial approximations
  - Intermediate calculations
  - Inconsistent data



Robustness Problems

- Degenerate data
  - Not in general position
- Numerical error creates/removes



Goal

- Want both *accurate* and *robust* boundary evaluation
  - Accurate – curved surfaces, correct positions
  - Robust – handle all input cases
- Automatic evaluation – no individual tuning

Outline

- **Background and Motivation**
  - Boundary Evaluation
  - The Robustness Issue
  - **Exact Computation**
  - Prior Work
- Exact Boundary Evaluation
- Extensions
- Conclusion

What is Exact Computation?

- Represent and operate so that you are guaranteed to always make correct *decisions*
- Differs from exact arithmetic

So Why Use Exact Computation?

- Helps solve robustness problems
- Eliminates all numerical error
- A useful/necessary precursor to completely addressing degeneracies

So Why Not Use Exact Computation?

- No HW supported arithmetic operations/numbers
- Can be *extremely* slow (10,000+ times slower on basic problem) for naïve approach
- Little exact infrastructure
- Previous real-world application limited

What About Input?

- Real-world data not exact
- Exactness yields consistency
- Goal is reliable computation, not exact output

Theme

Ensure correctness first,
then increase efficiency.

vs.

Create an efficient implementation,
then work to make it more robust.

Outline

- **Background and Motivation**
  - Boundary Evaluation
  - The Robustness Issue
  - Exact Computation
  - **Prior Work**
- Exact Boundary Evaluation
- Extensions
- Conclusion

Previous Work – Boundary Evaluation

- Earliest systems for polyhedral models
  - Braid'75, Requicha/Voelcker `82, Mantyla `88, Hoffmann `89, Requicha/Rossignac `92, Benouamer et al. `94
- Study of freeform/curved surface intersections
  - Sarraga `83, Abhyankar/Bajaj `88, Farouki `89, Hohmeyer `91, Manocha `91, Goldman '91
- Boundary evalution on sculptured solids
  - Casale et al. `85, Weiler `85, Johnstone `91, Yu `92
  - Krishnan et al. `97,`98,`03

Previous Work – Robustness Issues

- Robustness issues highlighted especially in solid modeling community
  - Sugihara/Iri `89, Hoffmann `89, Segal `90, Yu ’91, Jackson `95, Fang et al. `93, Higashi et al. `95, Fortune `95, Desaulniers et al. ’92
- Major efforts in computational geometry
  - Robustness considerations now common
  - Library support – CGAL

Previous Work – Numeric Error

- Tolerances, Interval Arithmetic around for decades
- Numerous more recent techniques, applications specifically for geometry
  – Jackson `95, Hu et al. `96, Comba/Stolfi `93, Guibas et al. `95, Milenkovic `88, Massotti `93

Previous Work – Exact Computation

- Idea is old, more recent focus is on efficient computation
  - Benouamer et al. `94, Fortune/van Wyk `93, Johnson `92, Clarkson `95, Shewchuck `96, Yap/Dube `95, Bronnimann et al. `94, Karasick et al. `91,
- For polynomial systems, major work in the computer algebra, algebraic geometry communities

Outline

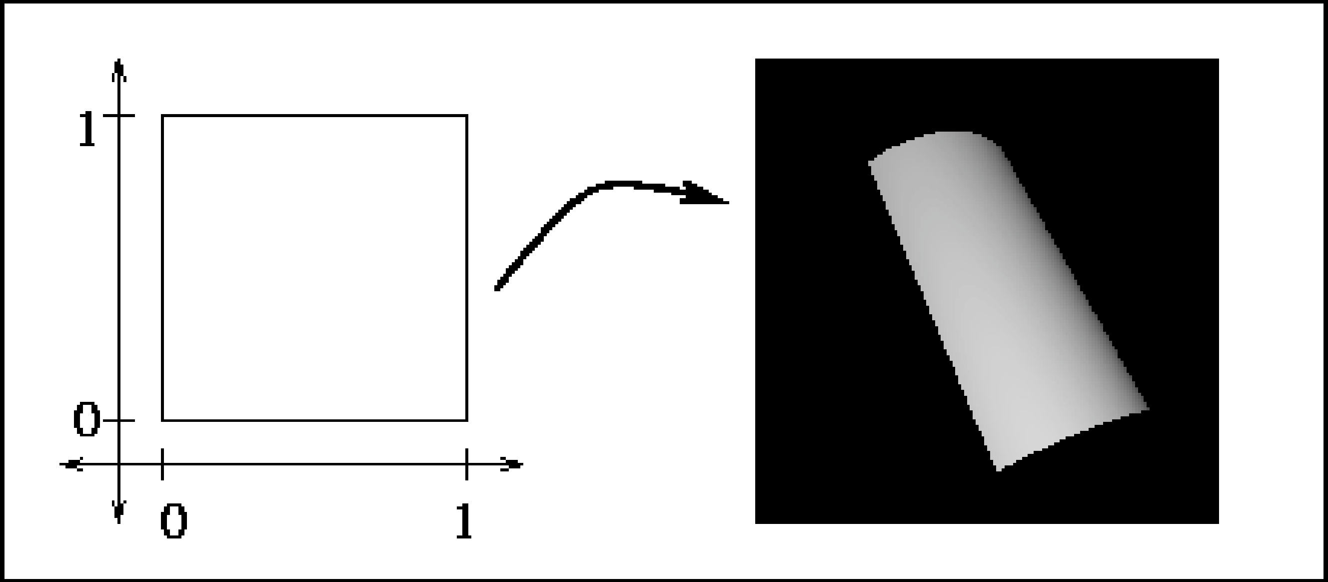
- Background and Motivation
- **Exact Boundary Evaluation**
  - **Exact Representations**
  - Key Operations
  - Implementation and Performance
- Extensions
- Conclusion

Exact Boundary Evaluation

- Linear solids relatively easy
  - Only rational numbers needed
  - Success for small cases
- Curved solids more difficult
  - Evaluation algorithm is more complex
  - Algebraic number representations
  - Higher degrees very inefficient
  - No previous implementations

Surface Representation

- Rational parametric surfaces
  - Implicit form also stored
- Polynomials with rational coefficients

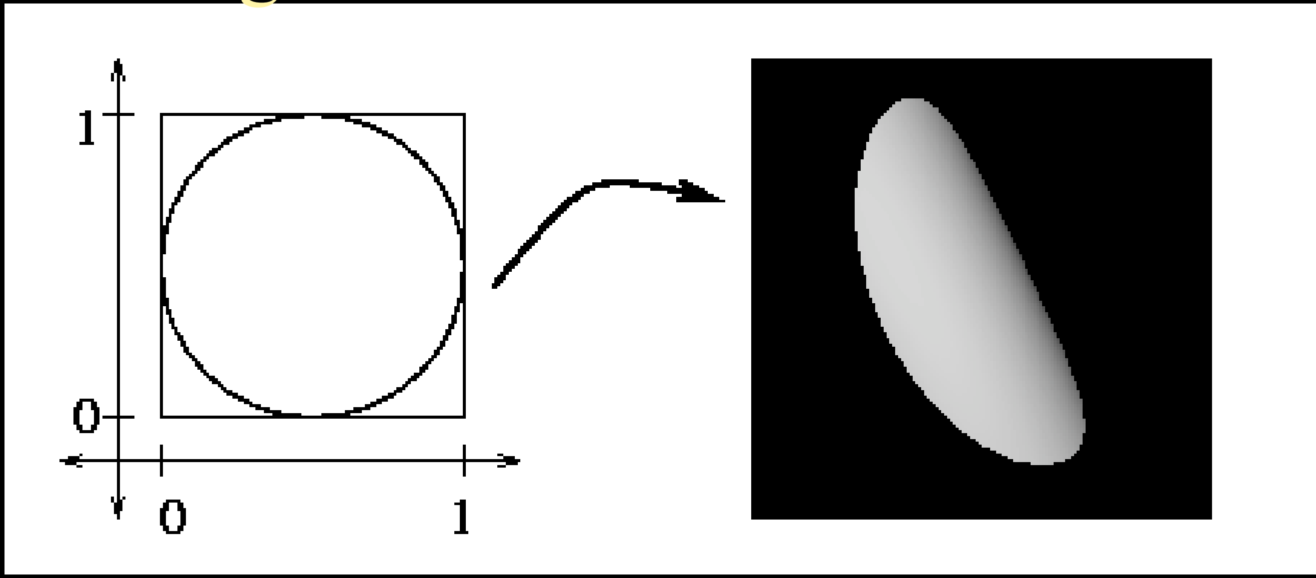


Surface Representation

- Standard CSG Primitives
  - Algebraic degree 4 (biquadratic parametric)
- General Geometric Modeling
  - Bicubic parametric patches
    - Algebraic degree 18
  - Higher degree patches sometimes desired

Patch Representations

- Surface
- Domain
- Trimming curves



Curve Representation

- Represented in patch domain only
- Arise from intersection of two surfaces
  - Standard CSG primitives: bidegree 8 max (usually less)
- Polynomial with rational coefficients
  - Real algebraic plane curve
- Might not have rational parameterization

Curve Representations

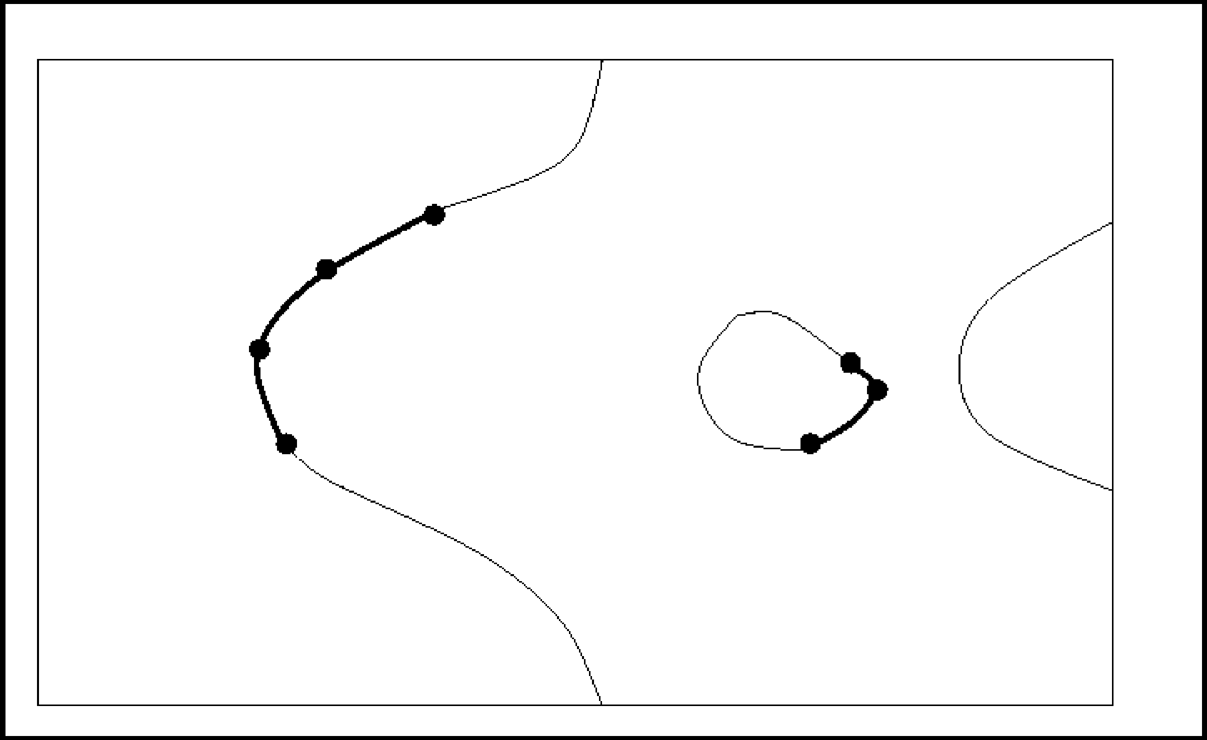
$382933409820669003196713865430094203187838850691401812970460827681502003200\, x^4y^2 -$
$159316795201622074223650790789613829790651395833729616712610453913600\, x^4y -$
$4130927475362116243835859246013253831252573339820099436158667227751487375\, x^4 -$
$77192131470752123955423438963146322282958823780111175493724500131840\, x^3y +$
$3716426671057241252846511576732374690782442737540106966934142592866290720\, x^3 +$
$765866819641338006393427730860188406375677701382803625940921655363004006400\, x^2y^2 -$
$309070421201018981537219598627407347376077829328701082705441828372480\, x^2y +$
$7425903571989547190948922596971310846678952823433730739648099737766232094\, x^2 -$
$77192131470752123955423438963146322282958823780111175493724500131840\, xy -$
$3716564352735855528405199583935793613721079631596555083611994905063427104\, x +$
$382933409820669003196713865430094203187838850691401812970460827681502003200\, y^2 -$
$149753625999396907313568807837793517585426433494971465992831374458880\, y -$
$4130911991700648202824812762953512676336122789886470960893780431639385999$

Curve Manipulation

- Only need a portion of each curve
- Need to be able to manipulate curve
  - Want to treat like parametric
  - Sort points along curve
  - Generate points at intervals
  - Classify point as on/off curve
- Curve topology

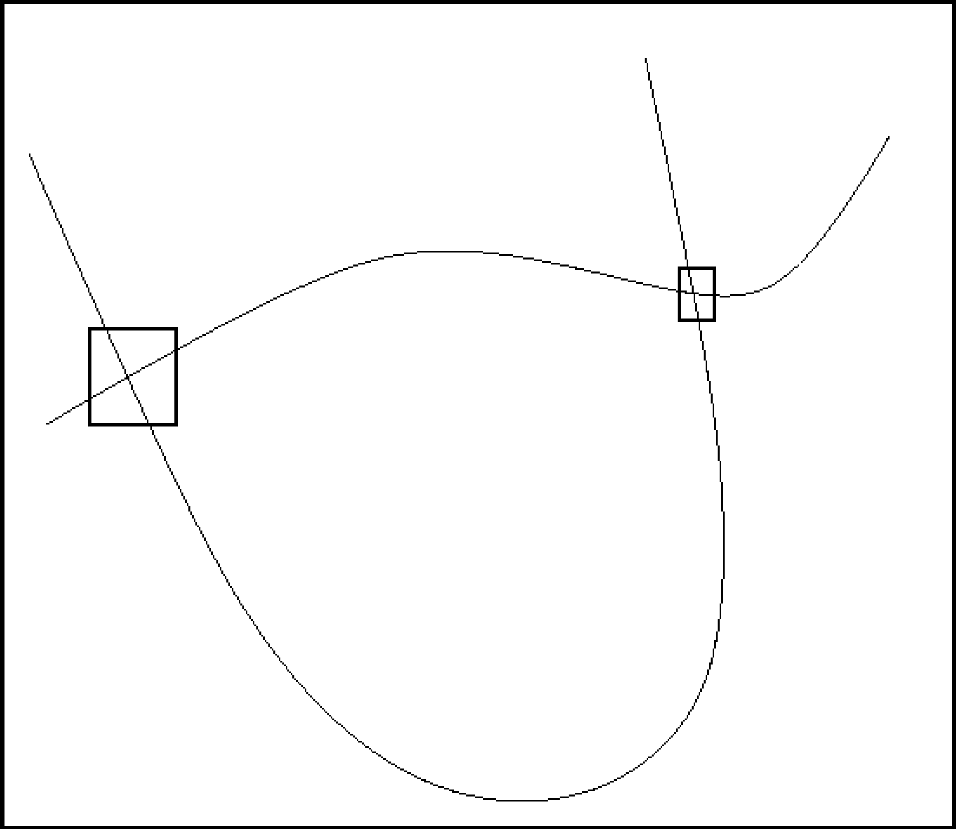
Curve Manipulation

- Break into monotonic segments
- Non-overlapping bounding boxes
- Limited domain



Point Representation

- Intersection of two curves within a 2D interval
- Unique, exact
- Compare point to point/constant

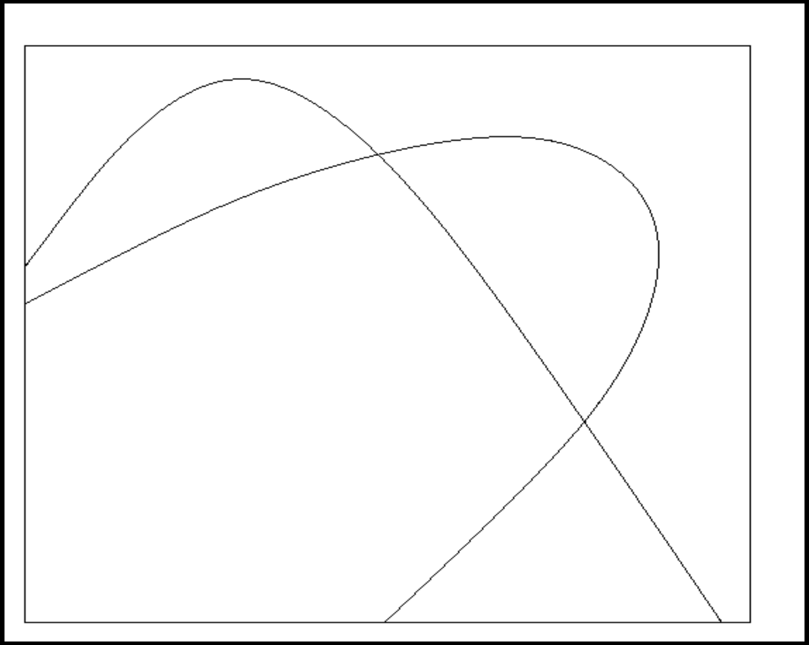


Curve-Curve Intersection

- Given two algebraic plane curves, isolate all intersections over some region
- Assume general position
- *Key operation* – can be called >1000 times for each Boolean operation

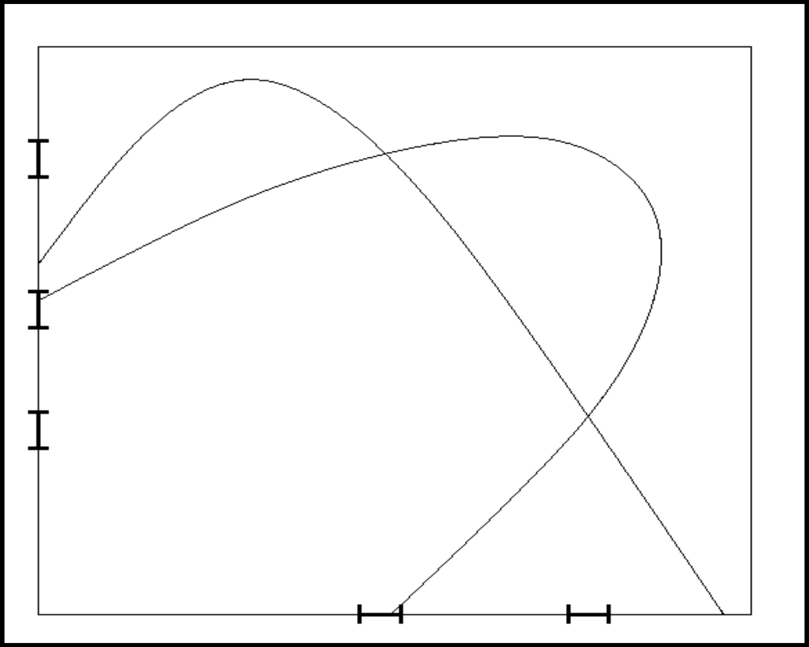
Curve-Curve Intersection

- Given:
  - Two algebraic plane curves
  - Domain
- Convert to series of 1D problems



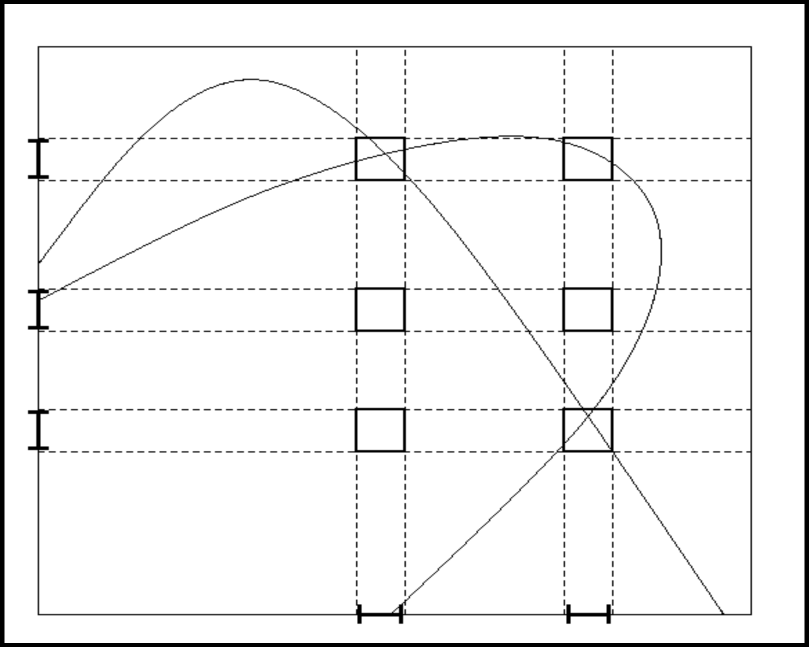
Curve-Curve Intersection

- Resultants eliminate one variable
- Isolate coordinates individually (Sturm sequences)



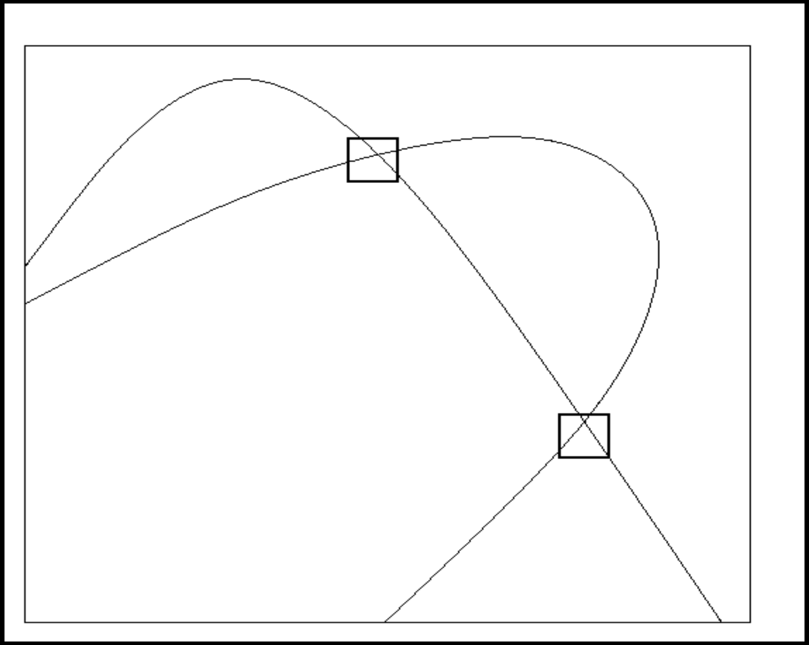
Curve-Curve Intersection

- Form boxes for potential intersections (Sakkalis
- Each box contains zero or one intersection



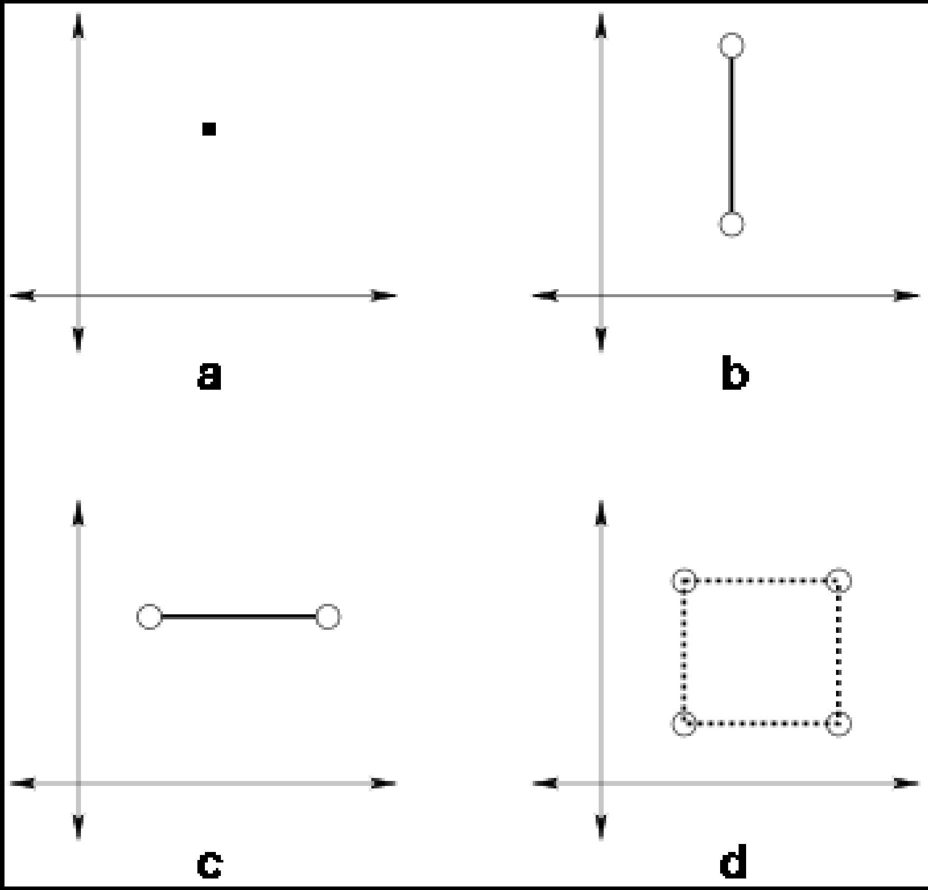
Curve-Curve Intersection

- Intersect curves with boxes (Sturm sequences)
- Determine which boxes contain intersections



Point Representation (continued)

- Sometimes know exact rational value for one or both coordinates
- Use hybrid representation – more efficient computation



Curve/Curve Example

- Earlier example, intersect with hodograph curve (to find turning points)
- Time= 95.666 ms.
- X[0] = ([-411/512, -205/256], [1299/8589934592, 20785/137438953472])
- X[1] = ([-409/512, -51/64], [5197/34359738368,20789/137438953472])

Outline

- Background and Motivation
- **Exact Boundary Evaluation**
  - Exact Representations
  - **Key Operations**
  - Implementation and Performance
- Extensions
- Conclusion

Implementation

- Set of kernel operations
  – Exact implementation
  – Key to efficiency

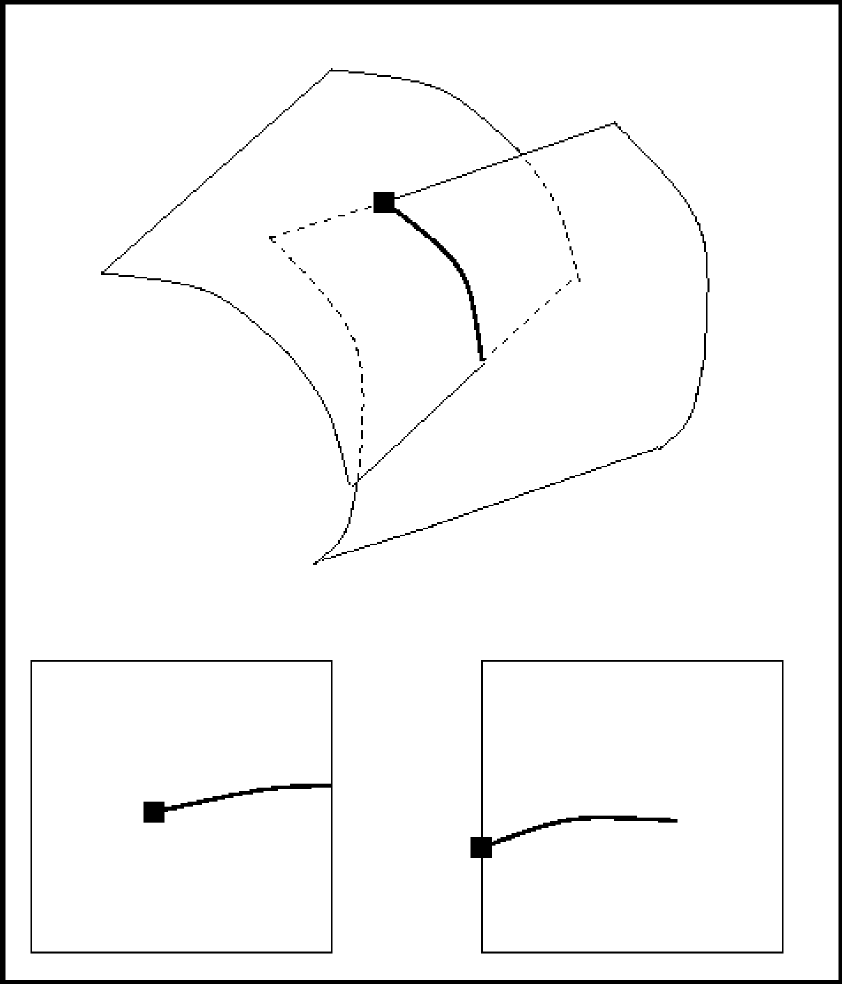
- Boundary evaluation built on top of kernel routines

Kernel Operations

- Curve-curve intersection
- Curve Topology
- Point Inversion
- Point Classification
- Surface Generation

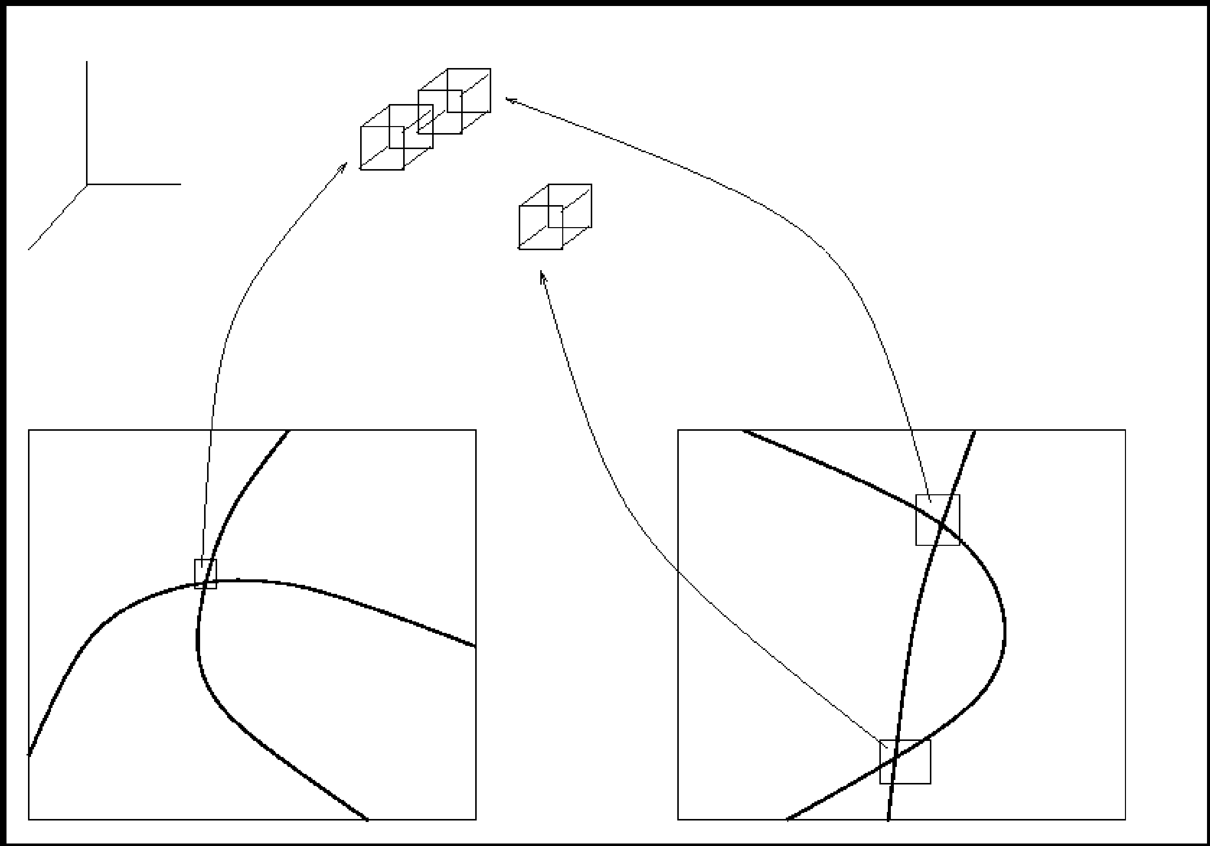
Point Inversion

- Map point from one patch to another
- Direct solution very slow



Point Inversion

- Rephrase as point matching

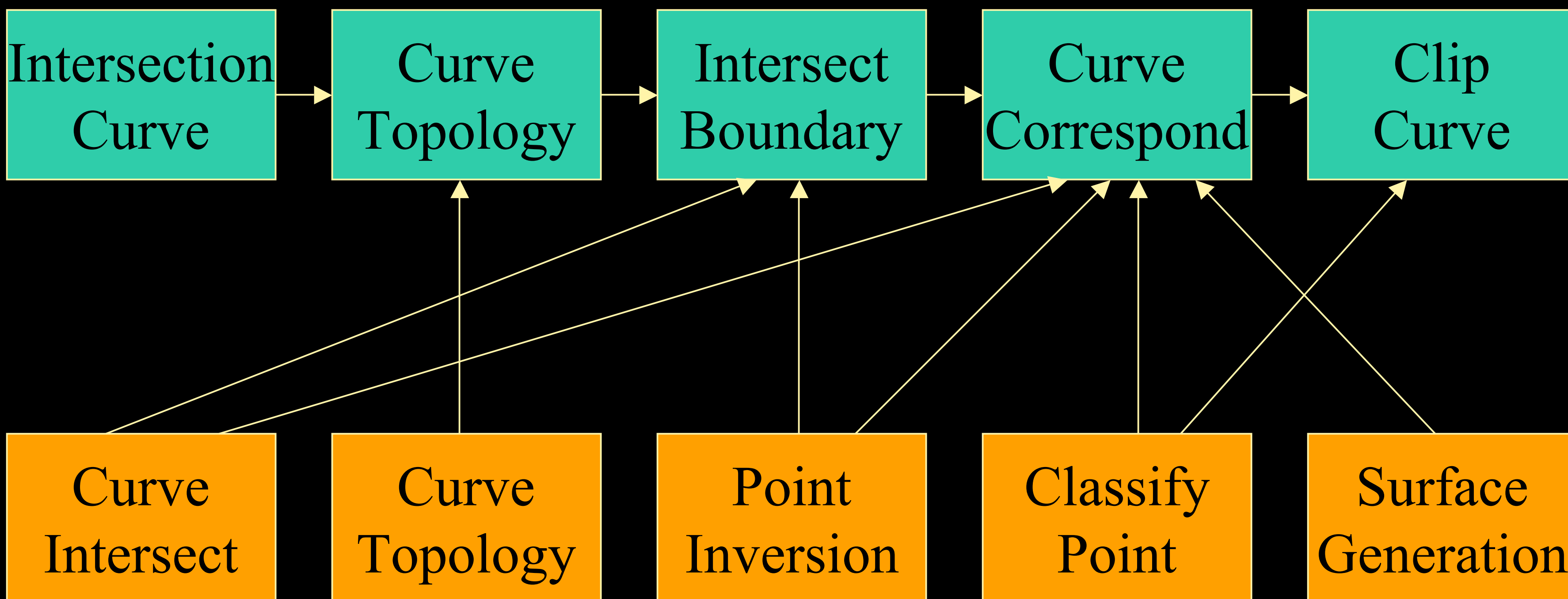


Point Inversion

- Make use of domain-specific knowledge to recast 7D or 4D problem into 2D problems with simple 3D checks
- Will not provide general-purpose inversion
  - Points must be intersections of curves
  - Curves must be intersections of surfaces
  - Requires solving for *all* real roots in one domain

Boundary Evaluation
Stage 1: Pairs of Patches

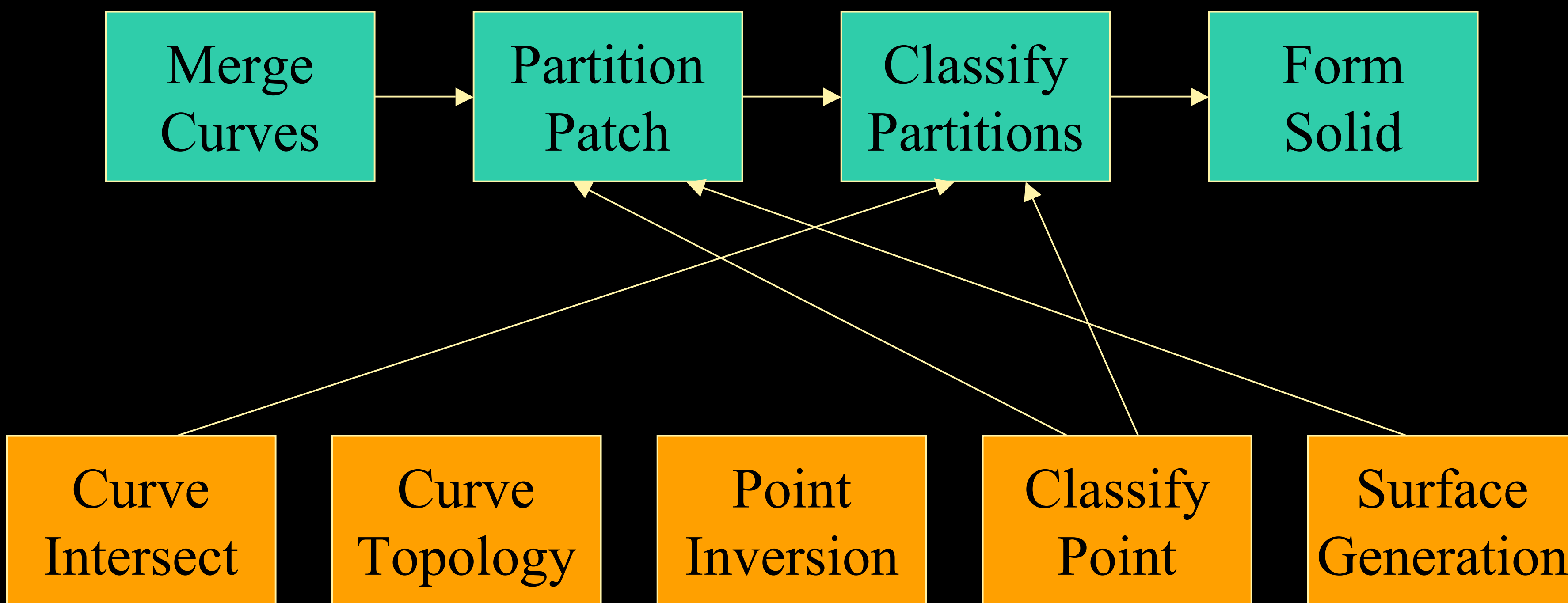
Boundary Evaluation Pipeline



Kernel Operations

Boundary Evaluation Stage 2: Each Patch

Boundary Evaluation Pipeline



Kernel Operations

Increasing Efficiency

- Efficient/hybrid representations
- Lazy Evaluation
- Lower-dimensional Formulation
- Quick Rejection
- Using Floating-point hardware
  - Floating-point filters
  - Floating-point guided computation

Combining Methods

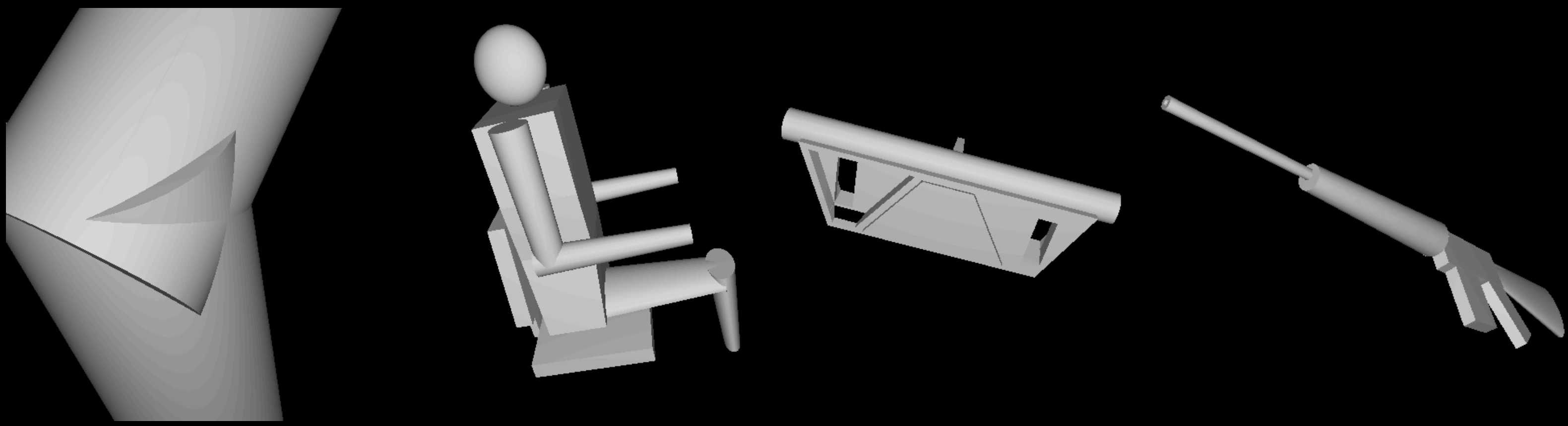
- Some methods can offset each other
  - Lazy evaluation and f.p. guided computation
- Some methods solve same cases
  - Quick rejection and f.p. filters
- Together, these methods provide *several orders of magnitude* of speed improvement over a naïve exact implementation

Outline

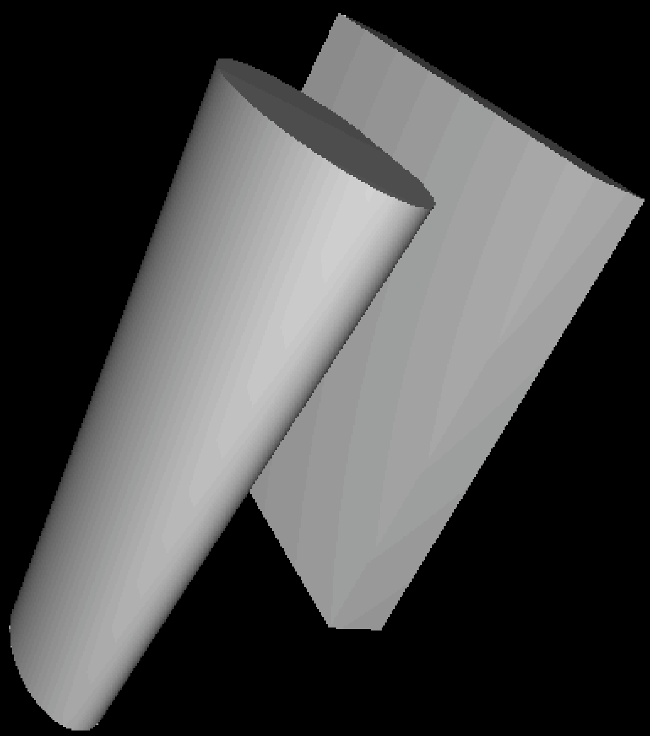
- Background and Motivation
- **Exact Boundary Evaluation**
  - Exact Representations
  - Key Operations
  - **Implementation and Performance**
- Extensions
- Conclusion

ESOLID System Performance

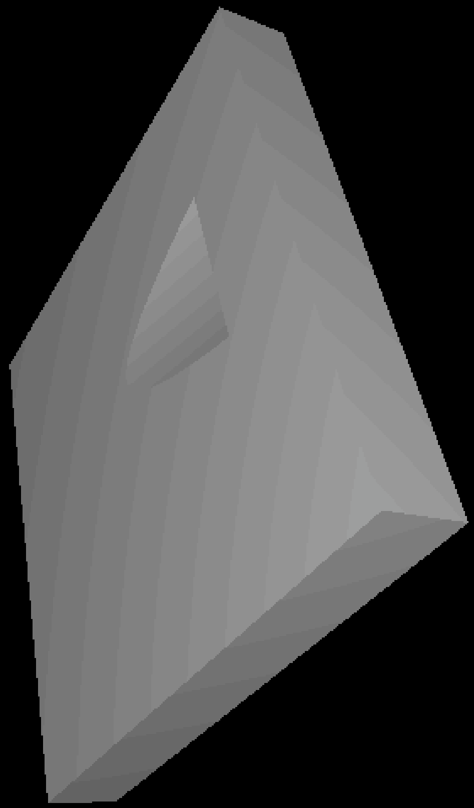
- Applied to real-world cases
  - Performance, with speedups, within one order of magnitude of Boole system



Importance of Accuracy

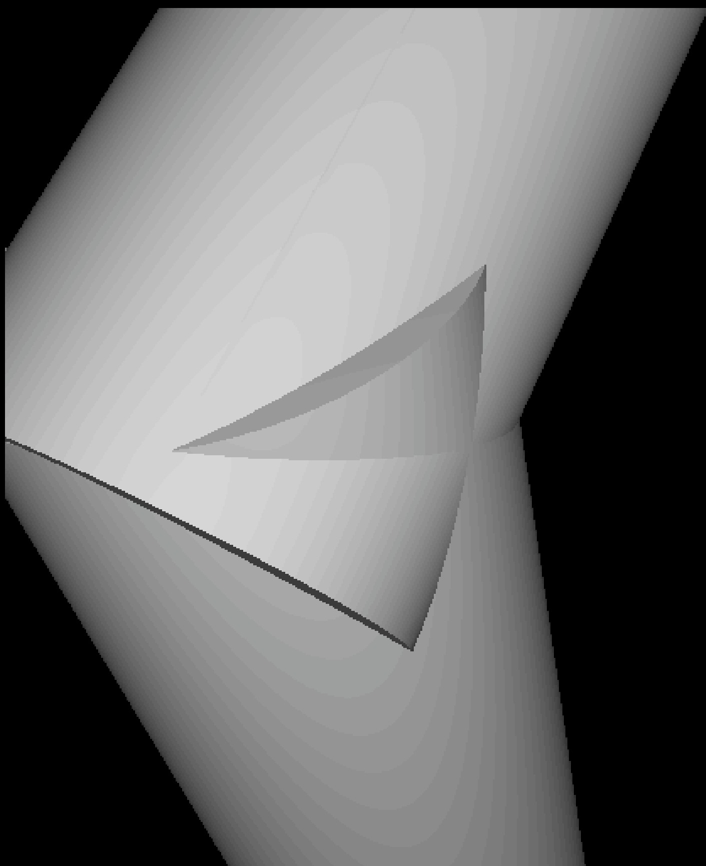


Input Primitives

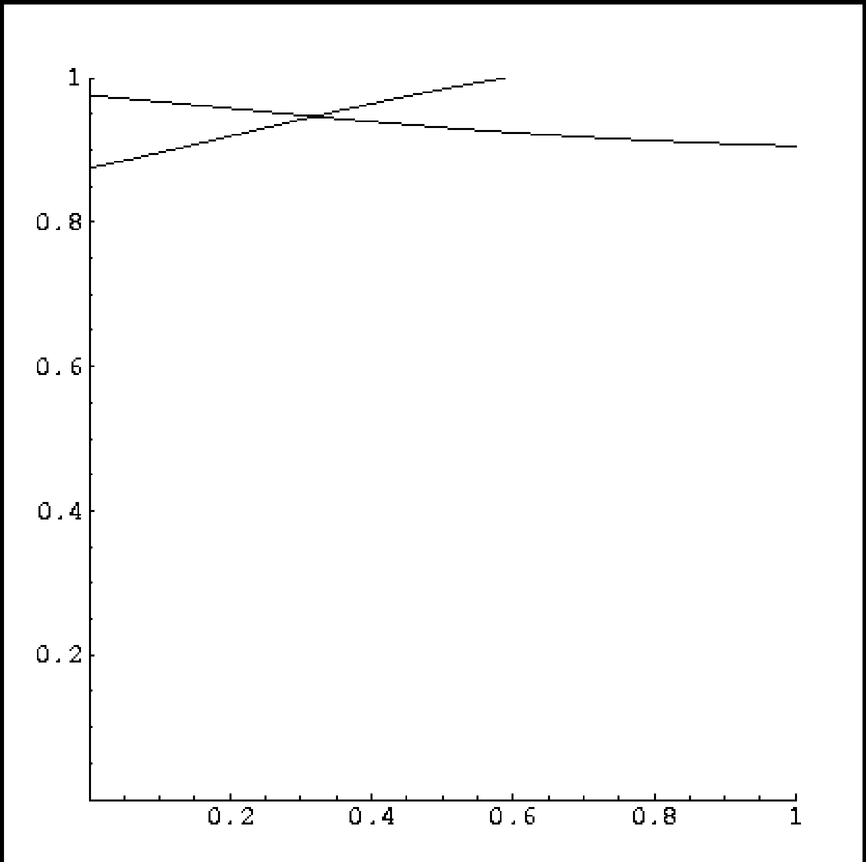


Output Solid

Importance of Accuracy



Input Primitives



Intersection Curves in Patch Domain

Timing Breakdown: Bradley Examples

- Of total time:
  - 54% to 98% in curve-curve intersections
- Of curve-curve intersection time:
  - 4% to 87% in resultant computations
  - 3% to 96% in Sturm computations
- Longer overall times *generally* imply:
  - Higher % of total time in curve-curve
  - Higher % of curve-curve time in Sturm

ESOLID Problems:

- Assumes General Position
  - Does not handle any actual degeneracies
  - Fails by crash, infinite loop
- Efficient only for low-degree surfaces
  - Complexity explodes rapidly for higher degrees

Outline

- Background and Motivation
- Exact Boundary Evaluation
- **Extensions**
  - **Degeneracy Detection**
  - Numerical Perturbation
- Conclusion

Goal

- Want way to find/represent points even in degenerate cases
  - Tangential intersections
  - Intersections at curve singularities
  - 3 or more curves meeting at a point
- Allow certain degeneracies to be detected, represented cleanly

Rational Univariate Reduction (RUR)

- Use as an alternative representation for points
- Capable of handling degenerate situations smoothly
- Can be used for detecting degeneracies, or as part of a routine to handle them directly
- Roullier – uses with Groebner bases
  - Non-Groebner methods/implementations iterative

RUR operation

- Given *m* polynomials in *n* variables
  - Rational coefficients
- Determine all roots of system by finding a set of polynomials:

$h(x)$:	*minimal polynomial*
$h_1(\square)$, $h_2(\square)$, *etc.*	*coordinate polynomials*

RUR operation (continued)

- Determine the roots of the minimal polynomial
- Evaluate those roots in coordinate equations
- Result gives coordinates of every common root of original system
  - For positive dimensional components, gives one point on that component.

Context for Our Implementation

- Fits into precision-driven computation model
  - LEDA and EGC work
  - Core library
- Extends model to handle arbitrary roots of polynomials
  - Includes complex roots, for intermediate computation

Implementing the RUR

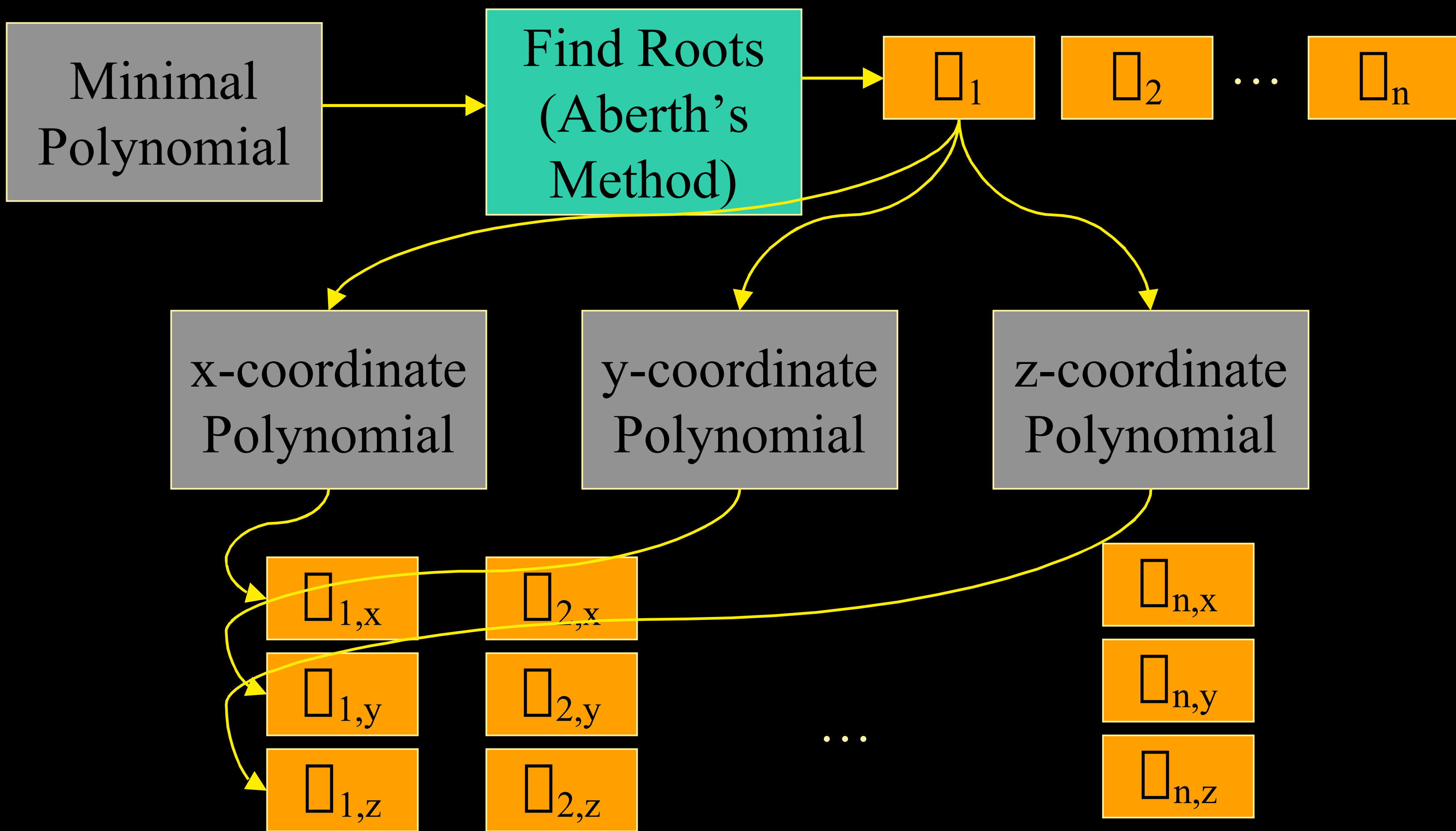
- Exact implementation of sparse resultant
  - Following Emiris's approach
  - Exact implementation throughout

- Polynomial interpolation
  - Avoid symbolic operations
  - Vandermonde interpolation of coefficients

Computing with the RUR

- Given the RUR, can find roots of minimal polynomial
  - Various techniques – Aberth's method is one for iteratively converging to roots.
  - Could usually just determine real roots
- Substitute these into coordinate polynomials to determine (complex) coordinates of roots.

Computing the RUR



Dealing with Complex Numbers

- Do not fit into existing root-bound approaches
  - Can determine real/imaginary parts separately
  - Root bounds determined independently
- Find real roots
  - Meet root bounds to show imaginary part = 0
  - Often can simplify

Representing Roots

- The evaluation process can determine bounding intervals (boxes) around each root
  - Positive dimensional component detection – probabilistic approach from random perturbations
- Can be used in geometric computations just like previous methods – e.g. for quick rejection tests

Functionality

- Each root is found by one point
- Degeneracies handled cleanly:
  - Tangential intersections
  - Singularities in curves/surfaces
  - Points lying on curves/surfaces
  - Coincident points/curves/surfaces

Timing Results

- Quadric curve intersections arising from real-world boundary evaluation cases:
  - MAPC (ESOLID): .017 -.024 seconds
  - RUR: .317-1.772 seconds
  - Approximately 20-100 times slower!
- Cases with degenerate intersections, positive dimensional components, higher dimension, all successful

Timing Breakdown

- Slows rapidly with higher degrees/dimensions
- For lower dimension/degree, the size of the matrix tends to determine running time
  - Increases quickly with degree/dimension
- At higher dimension/degree, coefficient size of coordinate polynomials grows very quickly and tends to dominate time
- Need a hybrid approach for efficiency

Checking Root Bounds

- Surprisingly, >98% of time was spent in computation of the RUR itself, not in checking root bounds.
  - Root bounds could conceivably take longer, with repeated construction in precision-driven system.

Optimizations

- Most optimizations are ***not*** implemented yet.
- Prior experience shows filtering and similar approaches can yield significant speedups
- Difficult to filter the sparse resultant matrix calculations
- May be able to filter over coefficients of the coefficient equations
- Possibly make higher degree/dimension more practical, but unlikely to ever beat MAPC

Outline

- Background and Motivation
- Exact Boundary Evaluation
- **Extensions**
  - Degeneracy Detection
  - **Numerical Perturbation**
- Conclusion

Numerical Perturbation

- Predicated on a test to detect degeneracies
- Idea: Since we have exact computation, we can eliminate degeneracies by perturbing the data numerically (not symbolically)
  - Symbolic must relate predicates directly to input
  - Filtering should make perturbed data not much slower in non-degenerate cases.

Numerical Perturbation

- Key is capturing the designer’s intent
  - Random or local perturbation of input surfaces in boundary evaluation can easily lead to undesirable results
  - True for numeric or symbolic perturbation

Numerical Perturbation

- New approach assumes only one degenerate situation, at known point in CSG-style tree.
- Perturbation will be less than some tolerance value.
- Handling multiple degeneracies can *apparently* be done by using different order of magnitude perturbation values.

Generating Perturbation

- Idea: need to perturb "in" or "out" at level where degeneracy is occurring.

- Capture designer's intent (see Sugihara/Iri):
  - Union – perturb both out
  - Intersection – perturb both in
  - Difference (A-B) – A in, B out

Propagating Information

- Propagate to children, all the way to leaves
  - For difference A-B, must propagate opposite perturbation direction

- Apply scale in/out at leaves
  - Note: some particular degeneracies remain after such scaling – very rare, but possible
  - Could combine with (smaller) translation

Recalculating

- Result is perturbed primitives
- Must repeat calculations for entire tree
- When original degenerate operation is reached, no degeneracy anymore

Performance

- Usually moderate increase in time for recomputation with perturbed data
  - Occasionally far more (7x as long!)
  - Percentage fails of f.p. filter increases
- Successfully resolves cases with degeneracies

Outline

- Background and Motivation
- Exact Boundary Evaluation
- Extensions
- **Conclusion**

Summary

- Exact boundary evaluation for curved solids
  - Reasonable efficiency
  - Increased robustness
  - Low degree surfaces
  - General position

Summary

- RUR method can find solutions to degenerate systems easily
  - Application to several degenerate cases
  - Less efficient than other general-position methods
  - Currently used for detection, not handling
- Numerical perturbation technique to eliminate degeneracies

Ongoing Work

- Implementing Degeneracy Code
  - Speedup techniques
  - Hybridization of representation/computation
  - Integrating into ESOLID
- Numerical Perturbation
  - Implementing as program-controlled loop
  - Combining multiple perturbations
  - Accounting for scale-invariant degeneracies

Ongoing Work

- Geometric level of detail filtering
  - Plane curves, Parametric surfaces
  - Understanding tradeoffs in filters
- User control over accuracy/robustness
  - Specify by desired property, not computational process

Ongoing Work

- Avoid direct algebraic computations
  - Standard geometric Bezier patch intersections
  - Provide boundaries for intersection curves in patch domain
  - Arbitrary accuracy
  - What can we tell from intersections of this curve representation?

Future Work

- Non-manifold representation
- Parallelization
- I/O and intermediate storage

Collaborators

- ESOLID: (UNC)
  - Dinesh Manocha
  - Tim Culver
  - Shankar Krishnan
  - Mark Foskey
- Faculty:
  - J. Maurice Rojas (TAMU Math)
- Students:
  - Koji Ouchi
  - Ian Remmler

Support

- BRL-CAD and Bradley Model:
  - Army Research Lab
- Funding (recent):
  - NSF-CARGO (Incubator): DMS-0138446
  - NSF ITR: CCR-0220047

Questions?

- Thanks for your attention

- Contact:
  – keyser@cs.tamu.edu
  – http://research.cs.tamu.edu/keyser/geom