Classroom Examples of Robustness Problems in Geometric Computations

Lutz Kettner, Kurt Mehlhorn
MPI für Informatik, Saarbrücken

Sylvain Pion
INRIA, Sophia Antipolis

Stefan Schirra
Otto-von-Guericke-Universität, Magdeburg

Chee Yap
New York University

January 17, 2006
Overview

- Overview
- The Orientation Predicate
- A 2D Convex Hull Algorithm
Overview

• The algorithms of computational geometry are designed for a machine model with exact real arithmetic.
• Floating point arithmetic incurs round-off error
• Substituting floating point arithmetic for the assumed real arithmetic may cause implementations to fail.
• I will show you some drastic examples of failures due to roundoff error in the hope
  • That you list more attentively to the alternatives presented later (and which require a fair amount of math)
Intersection of Four Simple Solids

- Rhino3D:
  - (((s₁ ∩ s₂) ∩ c₂) ∩ c₁) → successful
  - (((c₁ ∩ c₂) ∩ s₁) ∩ s₂) → "Boolean operation failed"
Intersection of Four Simple Solids

- output is a combinatorial object plus coordinates (not a point set)

- **Rhino3D:**
  - $((( s_1 \cap s_2 ) \cap c_2 ) \cap c_1 ) \rightarrow$ successful
  - $((( c_1 \cap c_2 ) \cap s_1 ) \cap s_2 ) \rightarrow "\text{Boolean operation failed}"$
Overview

• …, I believe that the program fails due to floating point rounding errors, but the example is too complicated to be followed through in detail

• We do the following:
  - First study the effect of floating point arithmetic on one of the basic geometric predicates, namely the orientation predicate for three points
  - And then study the global effects on a simple 2d convex hull algorithm.
2D-Orientation of Three Points

Orientation( p, q, r) = sign \[
\begin{vmatrix}
  p_x & p_y & 1 \\
  q_x & q_y & 1 \\
  r_x & r_y & 1
\end{vmatrix}
\]

Orientation( p, q, r) = sign(\((q_x-p_x)(r_y-p_y)-(q_y-p_y)(r_x-p_x)\))

Implemented with IEEE 784 double precision

\text{float_orient}(p, q, r)

\text{Float_orient} is not a correct implementation of orientation.
2D-Orientation of Three Points

Orientation( p, q, r) = \text{sign}((q_x-p_x)(r_y-p_y) - (q_y-p_y)(r_x-p_x))

\[
p: \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix}
\]

\[
q: \begin{pmatrix} 12 \\ 12 \end{pmatrix}
\]

\[
r: \begin{pmatrix} 24 \\ 24 \end{pmatrix}
\]

0 ≤ x, y < 256, u = 2^{-53}

256x256 pixel image
red=pos., yellow=0, blue=neg.

What do you expect orientation evaluated with double
2D-Orientation of Three Points

Orientation\((p, q, r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))\)

\[
p: \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix}
\]

\[
q: \begin{pmatrix} 12 \\ 12 \end{pmatrix}
\]

\[
r: \begin{pmatrix} 24 \\ 24 \end{pmatrix}
\]

\(0 \leq x, y < 256, u = 2^{-53}\)

256x256 pixel image

- red=pos., yellow=0, blue=neg.
- orientation evaluated with \text{double}
2D-Orientation of Three Points

Orientation \((p, q, r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))\)

\[p: \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix} \]
\[q: \begin{pmatrix} 8.80000000000000007 \\ 8.80000000000000007 \end{pmatrix} \]
\[r: \begin{pmatrix} 12.1 \\ 12.1 \end{pmatrix} \]

\[0 \leq x, y < 256, u = 2^{-53} \]

256x256 pixel image

*red*=pos., *yellow*=0, *blue*=neg.

orientation evaluated with *double*
2D-Orientation of Three Points

Orientation\((p, q, r) = \text{sign}((q_x-p_x)(r_y-p_y) - (q_y-p_y)(r_x-p_x))\)

\[p:\begin{pmatrix}0.50000000000002531 + x \cdot u \\ 0.5000000000001710 + y \cdot u \\ 17.30000000000000001 \\ 17.30000000000000001 \\ 24.000000000000500000 \\ 24.000000000000517765 \end{pmatrix}\]

\[q:\begin{pmatrix}17.30000000000000001 \\ 17.30000000000000001 \end{pmatrix}\]

\[r:\begin{pmatrix}24.000000000000500000 \\ 24.000000000000517765 \end{pmatrix}\]

\[0 \leq x, y < 256, u = 2^{-53}\]

256x256 pixel image

\textcolor{red}{\text{red}}=\text{pos.}, \textcolor{yellow}{\text{yellow}}=0, \textcolor{blue}{\text{blue}}=\text{neg.}

orientation evaluated with \text{double}
2D-Orientation of Three Points

Orientation\((p, q, r) = \text{sign}((q_x-p_x)(r_y-p_y) - (q_y-p_y)(r_x-p_x))\)

\[
p: \begin{pmatrix} 0.50000000000002531 + x \cdot u \\ 0.5000000000001710 + y \cdot u \\ 17.3000000000000001 \\ 17.3000000000000001 \end{pmatrix}
\]

\[
q: \begin{pmatrix} 24.0000000000000500000 \\ 24.0000000000000517765 \end{pmatrix}
\]

\[
r: \begin{pmatrix} 0 \\ 0.24 \\ 0.17 \\ 0.0 \end{pmatrix}
\]

\(0 \leq x, y < 256, u = 2^{-53}\)

256x256 pixel image
red=pos., yellow=0, blue=neg.
orientation evaluated with \text{ext double}
Classification with respect to two Lines

• consider
  - \(p_1 = (0.5, 0.5)\)
  - \(p_2 = (7.30000000000000194, 7.30000000000000167)\)
  - \(p_3 = (24.00000000000000068, 24.00000000000000071)\)
  - \(p_4 = (24.00000000000000005, 24.00000000000000053)\)
• \((p_2, p_3, p_4)\) form a counter-clockwise triangle

• Classification of \((x(p_1) + x \cdot u, y(p_1) + y \cdot u)\) with respect to the edges \((p_2, p_3)\) and \((p_4, p_2)\).

  \text{red} = \text{sees no edge}, \text{ocher} = \text{collinear with one}, \text{yellow} = \text{collinear with both}, \text{blue} = \text{sees one but not the other}, \text{green} = \text{sees both}
Convex Hulls in the Plane

- maintain current hull as a circular list $L=(v_0,v_1,...,v_{k-1})$ of its extreme points in counter-clockwise order
- start with three non-collinear points in $S$.
- consider the remaining points $r$ one by one.
  - Determine the set of visible edges
  - If none, $r$ is inside and we are done
  - Otherwise, remove all visible edges and add the two tangents from $r$
Convex Hulls in the Plane

- maintain current hull as a circular list $L= (v_0, v_1, ..., v_{k-1})$ of its extreme points in counter-clockwise order
- start with three non-collinear points in $S$.
- consider the remaining points $r$ one by one.
Correctness Properties

- **[Property A]** A point $r$ is outside $CH$ iff there is an $i$ such that the edge $(v_i, v_{i+1})$ is visible for $r$. (orientation$(v_i, v_{i+1}, r) > 0$)
- **[Property B]** If $r$ is outside $CH$, then the set of edges that are weakly visible (=$orientation$ is non-negative) from $r$ forms a non-empty consecutive subchain; so does the set of edges that are not weakly visible from $r$. 
... and next

- Instances leading to violations of properties (A) and (B) when executed with double’s
- and in all possible ways
  - a point outside sees no edge
  - a point inside sees an edge
  - a point outside sees all edges
  - a point outside sees a non-contiguous set of edges
- examples involve nearly collinear points, of course
- examples are realistic as many real-life instances contain collinear points (which become nearly collinear by conversion to double’s)
Global Consequences

• A point outside sees no edge of the current hull.

\[ p_1 = (7.3000000000000194, 7.3000000000000167) \]
\[ p_2 = (24.000000000000068, 24.000000000000071) \]
\[ p_3 = (24.00000000000005, 24.000000000000053) \]
\[ p_4 = (0.5000000000001621, 0.5000000000001243) \]
\[ p_5 = (8, 4) \quad p_6 = (4, 9) \quad p_7 = (15, 27) \]
\[ p_8 = (26, 25) \quad p_9 = (19, 11) \]

float_orient(\( p_1, p_2, p_3 \)) \> 0
float_orient(\( p_1, p_2, p_4 \)) \< 0
float_orient(\( p_2, p_3, p_4 \)) \< 0
float_orient(\( p_3, p_1, p_4 \)) \< 0
Global Consequences

• A point outside sees all edges of the current hull.

\[ p_1 = (200, 49.200000000000003) \]
\[ p_2 = (100, 49.600000000000001) \]
\[ p_3 = (-233.333333333333, 50.93333333333333) \]
\[ p_4 = (166.66666666666669, 49.333333333333336) \]

\[ \text{float_orient}(p_1, p_2, p_3) > 0 \]
\[ \text{float_orient}(p_1, p_2, p_4) > 0 \]
\[ \text{float_orient}(p_2, p_3, p_4) > 0 \]
\[ \text{float_orient}(p_3, p_1, p_4) > 0 \]
Global Consequences

• A point outside sees all edges of the current hull.

\[ p_1 = (200, 49.200000000000003) \]
\[ p_2 = (100, 49.600000000000001) \]
\[ p_3 = (-233.3333333333334, 50.933333333333333) \]
\[ p_4 = (166.66666666666669, 49.33333333333336) \]

\[
\text{float}_\text{orient}(p_1, p_2, p_3) > 0 \\
\text{float}_\text{orient}(p_1, p_2, p_4) > 0 \\
\text{float}_\text{orient}(p_2, p_3, p_4) > 0 \\
\text{float}_\text{orient}(p_3, p_1, p_4) > 0
\]

Depending on the implementation:

• **Algorithm does not terminate!**
• **Algorithm crashes!**
Global Consequences

• A point inside sees an edge of the current hull.

\[ p_1 = (24.000000000000053, 24.000000000000053) \]
\[ p_2 = (24.0, 6.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.850000000000037, 61.000000000000012) \]
\[ p_5 = (24.000000000000068, 24.000000000000071) \]

\((p_1, p_2, p_3, p_4)\) form a convex quadrilateral
\(p_5\) is truly inside this quadrilateral, but
\(\text{float\_orient}(p_4, p_1, p_5) > 0\)
Global Consequences

- A point inside sees an edge of the convex quadrilateral.

\[
p_1 = (24.00000000000005, 24.00000000000053) \\
p_2 = (24.0, 6.0) \\
p_3 = (54.85, 6.0) \\
p_4 = (54.85000000000357, 61.0000000000121) \\
p_5 = (24.0000000000068, 24.0000000000071)
\]

\((p_1, p_2, p_3, p_4)\) form a convex quadrilateral, but \(\text{float}_\text{orient}(p_4, p_1, p_5) > 0\)
Global Consequences

- A point outside sees a non-contiguous

\[ p_1 = (24.0000000000000005, 24.00000000000000053) \]
\[ p_2 = (24.0, 6.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.8500000000000357, 61.0000000000000121) \]
\[ p_5 = (24.00000000000000068, 24.00000000000000071) \]
\[ p_6 = (6, 6) \]
Global Consequences

- A point outside sees a non-contiguous

\[ p_1 = (24.00000000000005, 24.0000000000000053) \]
\[ p_2 = (24.0, 6.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.8500000000000357, 61.0000000000000121) \]
\[ p_5 = (24.0000000000000068, 24.0000000000000071) \]
\[ p_6 = (6, 6) \]

\[ \text{float_orient}(p_4, p_5, p_6) > 0 \]
\[ \text{float_orient}(p_5, p_1, p_6) < 0 \]
\[ \text{float_orient}(p_1, p_2, p_6) > 0 \]
Global Consequences

- A point outside sees a non-contiguous edge.

\[ p_1 = (24.00000000000005, 24.000000000000053) \]
\[ p_2 = (24.0, 10.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.850000000000037, 61.0000000000000121) \]
\[ p_5 = (24.000000000000068, 24.000000000000071) \]
\[ p_6 = (6, 6) \]
Global Consequences

• A point outside sees a non-contiguous

\[ p_1 = (24.00000000000005, 24.000000000000053) \]
\[ p_2 = (24.0, 10.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.8500000000000357, 61.0000000000000121) \]
\[ p_5 = (24.000000000000068, 24.000000000000071) \]
\[ p_6 = (6, 6) \]
Global Consequences

• A point outside sees a non-contiguous

\[ p_1 = (24.00000000000005, 24.0000000000000053) \]
\[ p_2 = (24.0, 6.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.850000000000357, 61.000000000000121) \]
\[ p_5 = (24.000000000000068, 24.000000000000071) \]
\[ p_6 = (6, 6) \]
Global Consequences

- A point outside sees a non-contiguous set of edges

\[ p_1 = (24.00000000000005, 24.000000000000053) \]
\[ p_2 = (24.0, 6.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.8500000000357, 61.0000000000121) \]
\[ p_5 = (24.0000000000068, 24.0000000000071) \]
\[ p_6 = (6, 6) \]
Global Consequences

- A point outside sees a non-contiguous set of edges.

\[ p_1 = (24.00000000000005, 24.00000000000053) \]
\[ p_2 = (24.0, 6.0) \]
\[ p_3 = (54.85, 6.0) \]
\[ p_4 = (54.850000000000357, 61.00000000000121) \]
\[ p_5 = (24.00000000000068, 24.00000000000071) \]
\[ p_6 = (6, 6) \]
Conclusion

• Classroom examples
  Data sets and C++ programs available online:

  http://www.mpi-sb.mpg.de/~kettner/proj/NonRobust/

• Long version (available soon):
  - More analysis
  - Graham’s scan algorithm
  - 3D Delaunay triangulation