Straight Skeleton Implementations based on Exact Arithmetic

Günther Eder, Martin Held, and Peter Palfrader

EuroCG 2020

Online Conference, March 2020
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation.
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a wavefront propagation.
- The Straight Skeleton is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation.
Straight Skeleton

- Defined as a result of a wavefront propagation.
- The Straight Skeleton is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation.
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events: Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
  - Edge Events, Split Events
  - Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- **Edge Events**
- **Split Events**
- **Applications**: Tool path generation
Straight Skeleton

- Defined as a result of a wavefront propagation.
- The Straight Skeleton is the trace of the vertices of the wavefront over time.
- Edge Events
- Split Events
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events, Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events, Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a wavefront propagation.
- The **Straight Skeleton** is the trace of the vertices of the wavefront over time.
- Edge Events, Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events, Split Events.
- Applications: Tool path generation
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events, Split Events.
- Applications: Tool path generation, Roof modeling
Straight Skeleton

- Defined as a result of a *wavefront propagation*.
- The *Straight Skeleton* is the trace of the vertices of the wavefront over time.
- Edge Events, Split Events.
- Applications: Tool path generation, Roof modeling, Origami.
Algorithms

Best worst-case complexity:

With implementations:
- Aichholzer and Aurenhammer (1998)*.
- For monotone polygons: Biedl et al. (2015)*.
Algorithms

Best worst-case complexity:

With implementations:
- Aichholzer and Aurenhammer (1998)*.
- For monotone polygons: Biedl et al. (2015)*.
Algorithms

**Best worst-case complexity:**

**With implementations:**
- Aichholzer and Aurenhammer (1998)*.
  - For monotone polygons: Biedl et al. (2015)*.

* New implementation!
Algorithms

Best worst-case complexity:

With implementations:
- Aichholzer and Aurenhammer (1998)*.
- For monotone polygons: Biedl et al. (2015)*.

* New implementation!
• Part of CGAL.
• Input: polygons and polygons with holes.
• Priority queue of edge events and all potential split events.
• There are quadratic many such potential split events.
Cacciola

- Part of CGAL.
- Input: polygons and polygons with holes.
  - Priority queue of edge events and all potential split events.
  - There are quadratic many such potential split events.
Cacciola

- Part of CGAL.
- Input: polygons and polygons with holes.
- Priority queue of edge events and all potential split events.
- There are quadratic many such potential split events.
• Part of CGAL.
• Input: polygons and polygons with holes.
• Priority queue of edge events and all potential split events.
• There are quadratic many such potential split events.
• Part of CGAL.
• Input: polygons and polygons with holes.
• Priority queue of edge events and all *potential* split events.
• There are quadratic many such *potential* split events.
Biedl et al.

- **Input:** (strictly) monotone polygons.
- **Key Observation:** A monotone chain never splits.
- **Idea:** Compute the straight skeleton of two chains, then merge them.
- **Runtime:** $O(n \log n)$.

- **New implementation:** **MONOS**.
- Also works on *not-strictly* monotone polygons (tricky in the merge step).
Biedl et al.

- **Input**: (strictly) monotone polygons.
- **Key Observation**: A monotone chain never splits.
  - **Idea**: Compute the straight skeleton of two chains, then merge them.
  - **Runtime**: $O(n \log n)$.

- **New implementation**: **MONOS**.
- Also works on *not-strictly* monotone polygons (tricky in the merge step).
Biedl et al.

- Input: (strictly) monotone polygons.
- Key Observation: A monotone chain never splits.
- Idea: Compute the straight skeleton of two chains, then merge them.
- Runtime: $O(n \log n)$.

- New implementation: MONOS.
- Also works on not-strictly monotone polygons (tricky in the merge step).
Biedl et al.

- Input: (strictly) monotone polygons.
- Key Observation: A monotone chain never splits.
- Idea: Compute the straight skeleton of two chains, then merge them.
- Runtime: $O(n \log n)$.

- New implementation: MONOS.
- Also works on not-strictly monotone polygons (tricky in the merge step).
Biedl et al.

- Input: (strictly) monotone polygons.
- Key Observation: A monotone chain never splits.
- Idea: Compute the straight skeleton of two chains, then merge them.
  - Runtime: $O(n \log n)$.

New implementation: MONOS.
- Also works on not-strictly monotone polygons (tricky in the merge step).
Biedl et al.

- Input: (strictly) monotone polygons.
- Key Observation: A monotone chain never splits.
- Idea: Compute the straight skeleton of two chains, then merge them.
  - Runtime: $O(n \log n)$.

- New implementation: **MONOS**.
- Also works on *not-strictly* monotone polygons (tricky in the merge step).
Biedl et al.

- Input: (strictly) monotone polygons.
- Key Observation: A monotone chain never splits.
- Idea: Compute the straight skeleton of two chains, then merge them.
- Runtime: $O(n \log n)$.

- **New implementation**: **MONOS**.
- Also works on *not-strictly* monotone polygons (tricky in the merge step).
Biedl et al.

- Input: (strictly) monotone polygons.
- Key Observation: A monotone chain never splits.
- Idea: Compute the straight skeleton of two chains, then merge them.
- Runtime: $O(n \log n)$.

New implementation: **Monos**.
- Also works on *not-strictly* monotone polygons (tricky in the merge step).
Biedl et al.

• Input: (strictly) monotone polygons.
• Key Observation: A monotone chain never splits.
• Idea: Compute the straight skeleton of two chains, then merge them.
• Runtime: $O(n \log n)$.

• **New implementation:** Monos.
• Also works on *not-strictly* monotone polygons (tricky in the merge step).
Aichholzer and Aurenhammer

- **Input**: PSLGs. Can compute the weighted straight skeleton.

- Uses a kinetic data structure to witness events:
  - Triangulate the not-yet-swept plane; triangles witness events.

- There are only linear many *real events*.

- However, there might be \( O(n^3) \) flip events.

- **New implementation**: *Surfer2*.

- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many real events.
  However, there might be $O(n^3)$ flip events.

- New implementation: SURFER2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many real events.
  However, there might be $O(n^3)$ flip events.

- New implementation: SURFER2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- **Input**: PSLGs. Can compute the weighted straight skeleton.
- **Uses a kinetic data structure** to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
  - There are only linear many *real events*.
  - However, there might be $O(n^3)$ flip events.

- **New implementation**: Surfer2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- **Input**: PSLGs. Can compute the weighted straight skeleton.
- **Uses a kinetic data structure to witness events**: Triangulate the not-yet-swept plane; triangles witness events.
  - There are only linear many *real events*.
  - However, there might be $O(n^3)$ flip events.

- **New implementation**: Surfer2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

• Input: PSLGs. Can compute the weighted straight skeleton.
• Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
  There are only linear many real events.
  However, there might be $O(n^3)$ flip events.

• New implementation: \textsc{Surfer2}.
• Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
  There are only linear many real events.
  However, there might be $O(n^3)$ flip events.
- New implementation: SURFER2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- **Input**: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many *real events*.
  However, there might be $O(n^3)$ flip events.

- **New implementation**: **Surfer2**.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many real events.
  However, there might be $O(n^3)$ flip events.

- New implementation: Surfer2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many real events.
  However, there might be $O(n^3)$ flip events.

- New implementation: SURFER2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  - Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many *real events*.
  However, there might be $O(n^3)$ flip events.

- **New implementation:** Surfer2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

• Input: PSLGs. Can compute the weighted straight skeleton.
• Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
• There are only linear many real events.
  However, there might be $O(n^3)$ flip events.

• New implementation: Surfer2.
• Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many *real events*.
  However, there might be $O(n^3)$ flip events.

- **New implementation:** Surfer2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many real events. However, there might be $O(n^3)$ flip events.

- New implementation: Surfer2.
- Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

• Input: PSLGs. Can compute the weighted straight skeleton.

• Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.

• There are only linear many *real events*.
  However, there might be $O(n^3)$ flip events.

• **New implementation:** Surfer2.

• Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many *real events*.
  However, there might be $O(n^3)$ flip events.

- **New implementation:** *Surfer2*.  
  - Several special cases not considered in the original paper.
Aichholzer and Aurenhammer

- Input: PSLGs. Can compute the weighted straight skeleton.
- Uses a kinetic data structure to witness events:
  Triangulate the not-yet-swept plane; triangles witness events.
- There are only linear many *real events*.
  However, there might be $O(n^3)$ flip events.

- **New implementation**: Surfer2.
- Several special cases not considered in the original paper.
Some Special Cases
- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations
- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
  - Wavefront edges moving into each other.
  - Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
**Some Special Cases**

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

**Implementation Considerations**

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Some Special Cases

- Flip-event Loops.
- Vertices meeting along triangulation edges.
- Wavefront edges moving into each other.
- Collinear wavefront segments of different speeds becoming adjacent.

Implementation Considerations

- Event classification: Where possible, rely on combinatorial/discrete information instead of doing computations on reals.
Runtime

CGAL v. Surfer2

Runtime [s]

Memory Use [MiB]

# Vertices

[Graphs showing runtime and memory usage for CGAL and Surfer2, with different markers for interior-only and plane modes.]

- CGAL (interior-only)
- Surfer2 (plane)
- Surfer2 (interior-only)
- Surfer2 (plane, IEEE 754)
Monos v. Surfer2

Runtime

![Monos vs Surfer2 graph](image)
Investigating the spread

**Surfer2**

- **Runtime [s]**
  - 10^3
  - 10^2
  - 10^1
  - 10^0
  - 10^{-1}
  - 10^{-2}

- **# Vertices**
  - 10^5
  - 10^4
  - 10^3
  - 10^2
  - 10^1
  - 10^0

**CGAL**

- **Runtime [s]**
  - 10^3
  - 10^2
  - 10^1
  - 10^0
  - 10^{-1}
  - 10^{-2}

- **# Vertices**
  - 10^5
  - 10^4
  - 10^3
  - 10^2
  - 10^1
  - 10^0

- **Markers**
  - **rpg_octa**
  - **rpg_rnd**
  - **rpg_iso**
Investigating the spread

Why is iso less problematic than octa input?

- Turns out our octa input was on the integer grid, the iso had random coordinates.
- This resulted in significantly many co-temporal events for the octa input.
- Indeed, with random edge weights, the spread goes away.
- We can split triangles by component, as the skeletons are independent.
Investigating the spread

- Why is iso less problematic than octa input?
- Turns out our octa input was on the integer grid, the iso had random coordinates.
  - This resulted in significantly many co-temporal events for the octa input.
  - Indeed, with random edge weights, the spread goes away.
  - We can split triangles by component, as the skeletons are independent.
Investigating the spread

- Why is iso less problematic than octa input?
- Turns out our octa input was on the integer grid, the iso had random coordinates.
- This resulted in significantly many co-temporal events for the octa input.
  - Indeed, with random edge weights, the spread goes away.
  - We can split triangles by component, as the skeletons are independent.
Investigating the spread

- Why is iso less problematic than octa input?
- Turns out our octa input was on the integer grid, the iso had random coordinates.
- This resulted in significantly many co-temporal events for the octa input.
- Indeed, with random edge weights, the spread goes away.
- We can split triangles by component, as the skeletons are independent.
Investigating the spread

- Why is iso less problematic than octa input?
- Turns out our octa input was on the integer grid, the iso had random coordinates.
- This resulted in significantly many co-temporal events for the octa input.
- Indeed, with random edge weights, the spread goes away.
- We can split triangles by component, as the skeletons are independent.
Source code

- **Monos**: [https://github.com/cgalab/monos](https://github.com/cgalab/monos)
- **Surfer2**: [https://github.com/cgalab/surfer2](https://github.com/cgalab/surfer2)

Thanks! Questions? Mail palfrader@cs.sbg.ac.at
Source code

- **Monos**: https://github.com/cgalab/monos

- **Surfer2**: https://github.com/cgalab/surfer2

Thanks!

Questions? Mail palfrader@cs.sbg.ac.at