Dynamic Voronoi Diagram Calculation and Visualization with RIC and Fortune Algorithms

Barkın Saday
21902967

CS - 478

Project Goal

- Calculate and Visualize Voronoi diagrams dynamically with user interactions.
- Animating (to show the algorithms)
- Evaluate Performance
- Randomized Incremental Algorithm.
- Fortune's Algorithm. (Not Complete)

Tools & Technologies

- Unity
- C#
- MIConvexHull
- Python (charts)
- Unity Profiler (performance and debug)
- LaTeX
- Executable Program (on windows)

User Interface & Experience

- Input fields:
  - Point count, Distribution type (Uniform/Gaussian)
  - Algorithm Selection
- Interaction:
  - Generate: Create the Voronoi Diagram. (batch input)
  - Simulate: Animate incremental insertion.
  - Click: Add sites manually.
  - Clear: Reset diagram
- Visuals:
  - Colored Voronoi regions.
  - Site points and boundaries.

RIC Algorithm - Concept

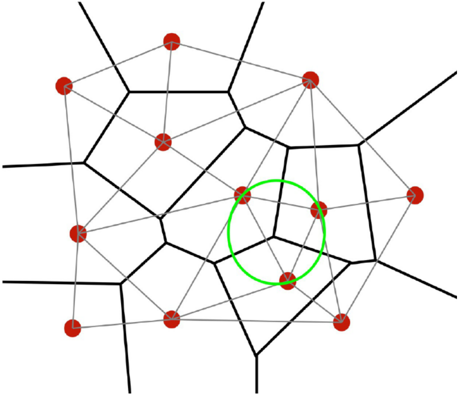
- Insert sites one-by-one in random order.
- Compute **local** Delaunay triangulation after each insertion. (Edge Flipping)
- Derive Voronoi cells via triangle circumcenters.
- Expected complexity: $\mathcal{O}(n \log n)$. (Randomness)

System Architecture

- **Manager.cs**: UI control, user input handling.
- **VoronoiGenerator.cs**: Core RIC computation.
- **VoronoiRenderer.cs**: Mesh rendering outlines.
- **MIConvexHull**: Batch Delaunay triangulation. (external)

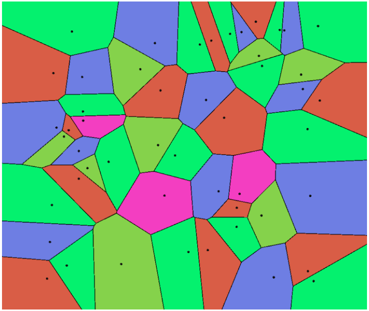
Voronoi via Delaunay Triangulation

- Delaunay triangles are built from site points.
- Circumcenters of triangles form Voronoi vertices.
- Voronoi edges are the perpendicular bisectors of Delaunay edges.
- Connect circumcenters to form convex Voronoi regions.
- Apply greedy coloring to differentiate adjacent regions.



Implementation Choices

- Delaunay Triangulation to Voronoi.
- Recompute on top of the existing diagram after each site.
- Greedy Coloring.
- Dynamic animations over a fixed time. (5 seconds)
- Polygon rendering via ear clipping triangulation.

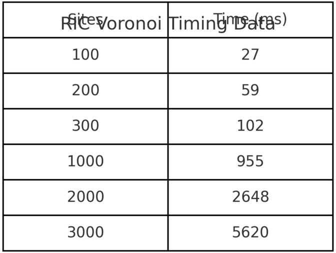


Implementation Choices (continued)

- Bounding Box, for batch input range.
- Padding on top of bounding box to define the universe. (Large triangle)
- Track the Convex Hull. (Manage infinite regions)
- Fully recomputing if edge flip fails. ($\mathcal{O}(n^2)$)
- **DCEL** like structure.
  - Origin-Terminus (V1, V2): IVertex2
  - 2 vertex makes an edge
  - Faces: VoronoiRegion objects
  - Point-Line Classifications
  - Prev and Next edges

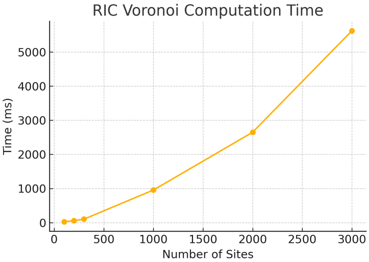
Results & Observations

- Smooth visualization up to 1000's of sites.
- Real-time interactions.
- Both uniform and Gaussian distributions tested.
- Handle infinite via initial bounding triangle.



RIC Voronoi Timing Data

Sites	Time (ms)
100	27
200	59
300	102
1000	955
2000	2648
3000	5620



Problems & Future Work

- Not a fully working DCEL structure (possible recomputation per step).
- Adjacent region coloring can repeat.
- Fortune's algorithm is not correctly implemented.
  - Diagram Computation is buggy.
  - Animation for sweeplines and beach lines are not implemented.

Demo - Q&A

- Live demo:
    - Site insertion & simulation.
    - Region rendering & coloring.
- Questions welcome.