# Procedural Island Generation from Lloyd-Relaxed Voronoi Tessellations: Implementation, Alternatives, and Experimental Evaluation

Jyotirmay Zamre, Urvashi Balasubramaniam

December 6, 2025

**Abstract**

This report documents our working implementation of a procedural island map generator that combines centroidal (Lloyd-relaxed) Voronoi tessellations with Perlin-noise-modulated coastlines, KD-tree-accelerated attribute assignment, and a simple moisture–elevation biome classifier. We created a geographically-accurate simulated map, that generates landmasses and properties uniquely every time!

## Contents

## 1 Introduction and motivation

Procedural map generation is an established tool in games, simulation and research where large, varied, and parameterisable landscapes are required without hand-designing every feature. Polygonal maps built from Voronoi tessellations provide a convenient discretisation: they are visually appealing (cells look like regions), easy to reason about (each cell belongs to a single seed), and integrate naturally with graph-based routing (Delaunay dual).

This report documents a particular pipeline developed and tested such that it: (1) creates a well-distributed set of polygonal regions using Lloyd relaxation; (2) uses a combination of radial falloff and Perlin noise to select land vs sea; (3) estimates elevation, freshwater sources and moisture via nearest-neighbour queries; and (4) assigns biomes by simple elevation–moisture rules for final rendering. The implementation is in Python, using NumPy, SciPy spatial routines, and the `noise` Perlin implementation.

We were really inspired in particular by Amit Patel's work and fantasy map enthusiasts who explored and suggested various simualtion methods that we have tried to both replicate and build upon in our project.

## 2 Problem statement and objectives

### 2.1 High-level objective

Produce a compact, reproducible pipeline that generates plausible, aesthetically pleasing island maps (coastline + layered biomes) on a 2D square canvas, where:

- the geometry is polygonal (ensured by Voronoi cells),
- the land shape (coastline especially) is organic and controllable,
- elevation and moisture support believable biomes (forest, desert, alpine, etc.),
- the system runs on commodity hardware for several thousand seeds.

### 2.2 Practical constraints and user requirements

- Implementation should be straightforward to read and modify (Python, common libs).
- Performance: handle $n \in [1000, 5000]$ seed points in reasonable time.
- Visual clarity: Voronoi cell edges should not dominate; biomes should be coherent.
- Extensibility: later additions (rivers, erosion, wind) must be possible without rewriting the core representation.

## 3 Short summary of the implemented pipeline (what we actually built)

Programme structure and function implementation used to produce the figures attached:

1. **Seeding:** sample $n$ random integer points in a $1500 \times 1500$ square. Code: creation of `test_points` with `random.randint` in a loop.
2. **Voronoi computation:** compute SciPy's `Voronoi(points)`.
3. **Lloyd relaxation:** iterate 50 times; in each iteration compute polygon centroids (function `compute_centroid`) and replace sites with clamped centroids via `updateSites` and `lloyd_relaxation`.
4. **Attribute assignment:** `assign_attributes` builds a radial falloff mask + Perlin noise (via `noise.pnoise2`), thresholds to obtain land vs sea, computes elevation as normalized distance to nearest sea via `assign_elevation` which uses `scipy.spatial.cKDTre` selects freshwater seeds in a mid-elevation band with `assign_freshwater`, and computes moisture via a KD-tree query in `assign_moisture`.
5. **Rendering:** iterate Voronoi regions, skip unbounded regions (the implementation removes regions containing $-1$), clamp coordinates to the bounding box, and draw a polygon per cell with facecolor chosen by `biome_color(elev, moisture)`.

Key helper functions in the code: `handleUnbounded`, `clamp`, `compute_centroid`, `updateSites`, `lloyd_relaxation`, `assign_elevation`, `assign_freshwater`, `assign_moisture`, `assign_attributes`, `plot_coloured_voronoi`.

## 4 Related work and literature survey

### 4.1 Voronoi and Lloyd relaxation

A Voronoi tessellation is the canonical partition by nearest site. Algorithms used to compute Voronoi diagrams include Fortune's plane sweep and Delaunay-based methods such as Bowyer–Watson; SciPy uses Qhull under the hood for robustness. For centroidal Voronoi tessellations and Lloyd's method (iterative centroid updates) see Lloyd (1957) and modern analyses of convergence and CVT properties. The algorithm is a standard choice to produce blue-noise-style point distributions and visually pleasing polygonal partitions. (Online resources and specifics: Amit Patel's polygonal map generation notes and Red Blob Games' interactive MapGen4 are particularly influential in applied map generation and guided many design choices in this implementation.)

### 4.2 Noise functions and Perlin noise

Perlin noise and its multidimensional variants remain the workhorse for smoothly varying stochastic fields in graphics. Two-dimensional Perlin (or simplex) noise is widely used for height-base generation and for modulating falloff functions to create irregular coastlines.

### 4.3 Hydrology, moisture and biome models

There is extensive literature on how rainfall, groundwater and river networks can be simulated. Approaches range from simple nearest-source diffusion (as used in this project), to explicit watershed and flow routing (computing steepest-descent over a triangulation), to physically based hydraulic erosion simulations. Biome assignment is commonly done by two-dimensional rules over temperature vs. precipitation (Whittaker diagram) or via the Köppen climate classification for more realism; game-oriented systems often simplify this to elevation vs moisture lookup tables, which is the approach used here.

### 4.4 Public implementations and practical resources (inspiration)

Our primary resources and inspiration were:

- Amit Patel's *Polygonal Map Generation for Games* and the MapGen4 project (extensive step-by-step guide and code). This project demonstrates Voronoi-based maps, river routing, wind-driven precipitation maps and region labelling; we adapted several ideas (centroidal relaxation, freshwater selection heuristics) from those notes.
- Red Blob Games' Voronoi tutorial and interactive demos, which clarify implementation issues such as handling infinite regions and clipping.
- A number of open-source repository examples (Map-Generator notebooks and GitHub projects) where people implemented Voronoi-based maps with Lloyd relaxation and different water/river heuristics; these were used to compare design choices and discover common pitfalls (unbounded regions, vertex ordering, polygon area sign handling).
- The SciPy spatial documentation for `Voronoi` and `cKDTree` (used for nearest neighbor queries).

## 5 Algorithms, formulas and data structures used

This section documents the concrete algorithms, pseudocode, and complexity. All function names below are the ones used in the implementation.

## 5.1 Voronoi construction

**Library call:** `scipy.spatial.Voronoi(points)`. SciPy returns:

- `vor.points` (the seed points);
- `vor.regions` (vertex index lists for each region);
- `vor.point_region` (index of region for each point);
- `vor.vertices` (vertex coordinates); unbounded regions contain $-1$.

This representation was used directly; unbounded regions (with $-1$) must be filtered or clipped before polygon operations.

## 5.2 Lloyd relaxation (Centroidal Voronoi Tessellation)

**Objective:** move sites to the centroid of their Voronoi cell repeatedly until approximate stationarity (the centroidal property). The centroid of a simple polygon defined by vertices $(x_k, y_k)$, $k = 0..m-1$ (closed by $x_m = x_0$) is:

$$A = \frac{1}{2}\sum_{k=0}^{m-1}(x_k y_{k+1} - x_{k+1} y_k), \qquad C_x = \frac{1}{6A}\sum_{k=0}^{m-1}(x_k + x_{k+1})(x_k y_{k+1} - x_{k+1} y_k),$$

and similarly for $C_y$.

**Complexity:** building the Voronoi diagram (via Qhull) typically runs in $O(n \log n)$ expected; computing centroids visits each polygon vertex once, so $O(V)$ where $V$ is total vertex count, which is $O(n)$ for planar subdivisions. Doing $k$ iterations yields roughly $O(kn \log n)$ cost dominated by the tessellation rebuilds.

## 5.3 Handling unbounded regions and vertex ordering

SciPy marks unbounded cells with $-1$. The implementation's `handleUnbounded(region)` drops $-1$ entries and closes the region (by repeating the first vertex) if needed. Clamping the centroid to the bounding box is used to avoid seeds drifting outside the canvas.

## 5.4 Nearest-neighbour attributes and KD-tree acceleration

Attributes like elevation and moisture are computed via nearest-source queries:

`scipy.spatial.cKDTree` is used to index a set of points (sea cells, freshwater cells), and then query the distances from all sites in $O(n \log n)$ time. This reduces naive $O(n^2)$ pairwise queries to $O(n \log n)$ and is critical for scaling to a few thousand sites.

### 5.5 Freshwater selection heuristic

The code picks candidate freshwater indices within a tolerance band around a target elevation, then randomly selects a fraction of them (default fraction $\approx 0.2$). This is a deliberately simple heuristic that creates inland lakes or springs rather than a full river system.

### 5.6 Moisture assignment

Moisture is assigned to land sites based on their normalized distance to the nearest freshwater point: $m = 1 - \frac{d}{d_{\text{max}}}$ for land sites, where $d_{\text{max}}$ is the maximum observed distance among land sites (clamped to avoid division by zero).

### 5.7 Biome lookup

A simple rule-based classifier maps $(elev, moisture)$ to a hex colour via `biome_color`. The rule uses coarse thresholds $(elev < 0.33, elev < 0.66, moisture > 0.6, > 0.3)$ to choose among desert, grassland, temperate forest, alpine, snow, and hard rock.

**Data structures used:**

- `numpy.ndarray` for point coordinates and attributes (elevation, moisture, boolean masks).
- SciPy's `Voronoi` structure (regions, vertices, point_region).
- SciPy's `cKDTree` for nearest neighbour queries (sea points, freshwater points).
- Python lists for incremental polygon processing when vertex order or region modification is needed.

## 6 Alternative approaches considered (and why we chose the final design)

For every subproblem we considered multiple choices.

### 6.1 Seeding strategies

**Options:**

- Uniform random sampling (what we used initially).
- Poisson-disk sampling for blue-noise spacing (ensures minimum distance between points).
- Stratified sampling or importance sampling (denser seeds near features).

**Discussion:** Poisson-disk produces very nice, even spacing without many iterations but is more complex to implement and more expensive for dynamic sampling; Lloyd relaxation starting from uniform random is simple and effective for our sizes and gives control via iteration count. Poisson-disk or dart-throwing would be preferred for absolutely uniform cell sizes or when strict minimum spacing is required.

## 6.2 Voronoi vs alternative spatial discretisations

**Alternatives:** regular raster heightmaps, triangular meshes (Delaunay), quadtrees, or polygons derived from barycentric duals. **Choice rationale:** Voronoi polygons provide region-shaped parcels that look natural when coloured; the dual Delaunay is available if river routing or triangulated flows are needed. Raster heightmaps are easier for physical simulations (erosion), but lose the polygonal region aesthetic required here.

## 6.3 Relaxation and centroid computations

**Alternatives:** (1) direct Lloyd; (2) accelerated Lloyd (over-relaxation schemes); (3) using k-means for centroidal placement; (4) no relaxation. **Choice rationale:** classic Lloyd was simple and stable; other methods (over-relaxation) can converge faster but require extra parameters and testing. We performed 50 iterations and observed practical stationarity for $n \approx 1500$.

## 6.4 Coastline and base elevation

**Alternatives:**

- Pure noise-based heightmap (fractal / multi-octave Perlin or simplex noise).
- Radial falloff only (gives circular islands).
- Plate-tectonics inspired displacement (simulate uplift and plates).
- Combination of radial falloff + noise (what we used).

**Choice rationale:** Using radial falloff alone makes islands too symmetric; pure noise alone can produce fragmented archipelagos. The radial falloff $\times$ noise combination gives a centralized island with noisy coastline; this matches our aesthetic goal.

## 6.5 Elevation assignment

**Alternatives:**

- Direct height from Perlin noise (sampled at centroid).
- Distance-to-sea (distance transform) — what we used for a smooth falloff inward.

- Combine noise + distance (blend).
- Full procedural geology (plate collisions + erosion simulation).

**Choice rationale:** Distance-to-sea is stable and ensures that coastline is elevation 0, creating consistent beaches. Combining noise and distance (or using noise on top of distance) is a useful extension; full geological simulation is costly and was left for future work.

## 6.6 Freshwater and moisture modelling

**Alternatives:**

- Random seed mid-elevation freshwater nodes (our approach, simple).
- Watershed and river routing using steepest descent over triangulation (more realistic).
- Rain-shield / wind-driven precipitation model (as in MapGen4).
- Diffusion or graph-based propagation from ocean inward (for fog/maritime influence).

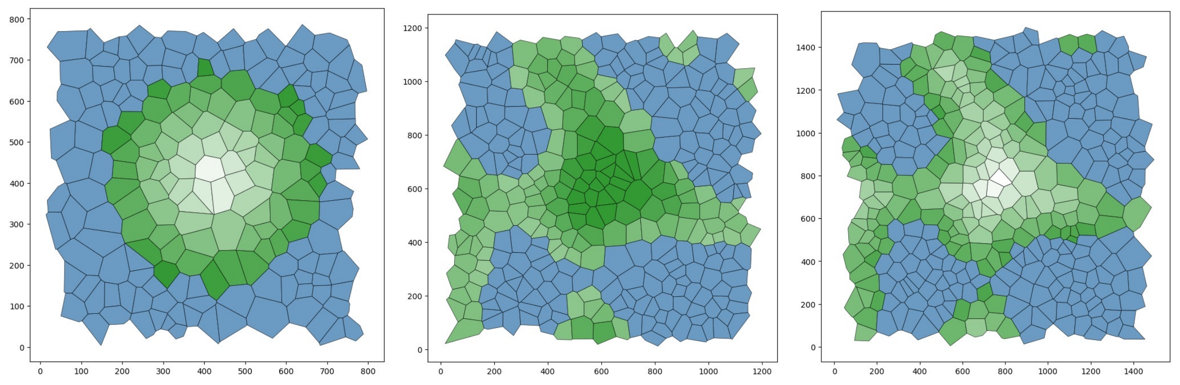
**Choice rationale:** watershed/flow-based river routing produces convincing rivers but requires a triangulation with consistent altitude order and a routing pass; for our first pass we preferred a KD-tree nearest-freshwater diffusion that is cheap and produces moisture pockets. MapGen4 and other advanced generators illustrate the wind/precipitation model that would be the best next step for realism.

## 6.7 Biome classification

**Alternatives:**

- Lookup in a Whittaker-like 2D grid (elevation vs precipitation) — fairly standard.
- Machine-learned classifier trained on climate data — heavy and unnecessary for a stylised island.
- Rule-based thresholds (what we used).

**Choice rationale:** rules are explainable and editable by artists; for production games a parameterised lookup table or learned mapping could be used.



(a) Using radial distance from centre (b) Using perlin noise and getting more geographic shapes (c) Changing elevation using KD tree based on nearest seapoint

Figure 1: Landmass creation attempts

## 7 Experimental exploration: trials, failures and refinements

### 7.1 Experimentation process

Initially, we tried using radial distance from a centre of the canvas, but found this produced too similar, round and regular maps. We then tried using Perlin noise and got more authentic and varied geographic shapes. We then tried elevation assignment using KD trees to find the nearest sea point and assigning elevation accordingly.

### 7.2 Lloyd iteration count

**What:** tried 5, 20, 50, 200 iterations. **Observation:** 5–20 iterations left visible clustering; 50 produced a visually smooth tessellation without over-regularising. 200 starts to give an unnaturally hexagonal tiling in places. **Adopted:** 50 iterations as pragmatic balance.

### 7.3 Perlin-only vs Perlin + radial falloff

**What:** tried pure Perlin threshold and pure radial falloff. **Outcome:** Perlin-only often produced many disconnected islands; radial-only produced too-regular circles. The blended mask (shape + 0.5 × noise) yielded a single central island with organic shoreline — the chosen design.

### 7.4 Freshwater selection strategies

**What:** initially selected freshwater by sampling the lowest-elevation land cells (near coasts) but that made moisture pockets concentrated near the coast and left interiors dry. **Outcome:** switched to sampling from a mid-elevation tolerance band (target_elev=0.5, tolerance=0.1) which better produced inland lakes and moisture-fed forests. **Adopted:** the mid-elevation heuristic with a small random sample fraction produces useful interior lakes.

### 7.5 Handling unbounded Voronoi regions

**Problem:** SciPy's Voronoi returns $-1$ for regions touching infinity. Early attempts to compute centroids led to indexing errors or invalid centroids. **Fix:** `handleUnbounded(region)` removes $-1$ and, where needed, the region is skipped in plotting or clamped. This prevents NaNs and drawing artifacts.

### 7.6 Performance notes

- KD-tree based distance queries reduced attribute assignment time by orders of magnitude compared to naive pairwise loops for $n \sim 1500$.
- Voronoi rebuilds are the computational bottleneck in Lloyd iterations; for very large $n$ an accelerated CVT solver or Poisson-disk seeding is preferable.

## 8 Evaluation and discussion

### 8.1 Qualitative assessment

The produced maps exhibit several desirable properties:

- **Region coherence:** Lloyd smoothing yields regions of similar area and shape, so biome colours are less noisy and appear as bands rather than salt-and-pepper patches.
- **Controlled islandness:** the radial falloff ensures a single dominant landmass, while Perlin noise creates coastline irregularity and small bays.
- **Interior diversity:** freshwater selection in the mid-elevation band creates interior forest pockets, improving visual interest beyond an elevation-only gradient.

### 8.2 Limitations

- No river routing was computed; rivers would greatly improve realism for erosion and moisture distribution.
- Temperature and latitudinal climate gradients were not modelled (one island, not globe).
- Biome thresholds are fixed heuristics and may look unnatural in corner parameter ranges.

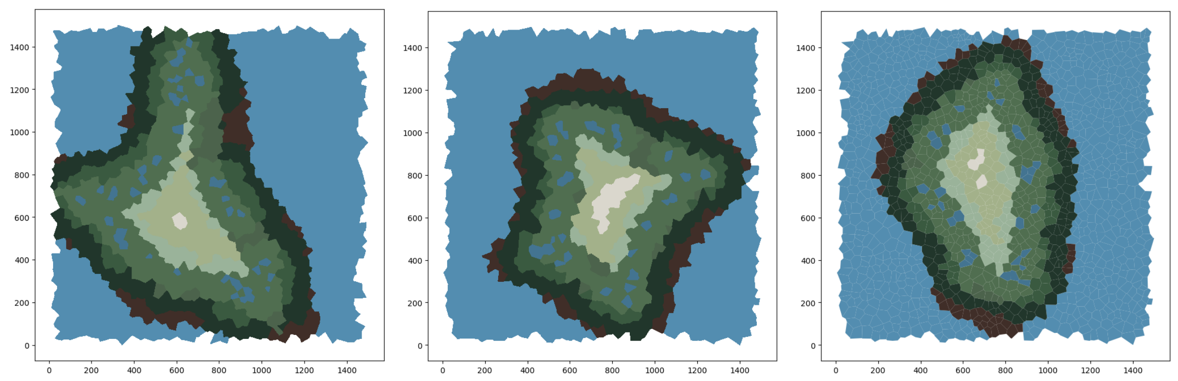
## 9 Extensions and future work

### 9.1 Next technical features to implement

- **River routing (watershed):** compute flow directions on the Delaunay triangulation dual (or on a rasterized heightmap) and route flow to coalesce rivers and erode channels.
- **Erosion simulation:** either particle-based hydraulic erosion or grid-based heightmap erosion to evolve the terrain from an initial condition.
- **Wind and rainfall modelling:** adapt MapGen4's wind-driven precipitation model to create a more realistic moisture map and rain shadows.
- **Temperature field:** add a simple temperature gradient (e.g., decreasing with elevation and depending on a latitude parameter) to refine biome selection.

### 9.2 Variants worth exploring

- Replace Lloyd relaxation with Poisson-disk sampling to avoid the iterative rebuild cost.
- Use a mixed elevation model: base distance-to-sea plus additive noise (multi-octave) and local erosion passes.
- Replace KD-tree nearest-freshwater moisture with diffusion on graph edges for smoother gradients.



(a) Map without antialiasing (b) Map without antialiasing (c) Antialiased

Figure 2: Landmass creation final results!

## 10 Conclusion

We both really enjoyed working on this project. It taught us a lot about the practical applications of Voronoi diagrams and allowed us to merge our interest in computer graphics with our learnings from computational geometry! Here are some images showing our resulting maps:

## References

[1] S. P. Lloyd, “Least squares quantization in PCM,” *IEEE Transactions on Information Theory*, 1957.

[2] A. Patel (Red Blob Games), “Polygonal Map Generation for Games”, online tutorial and MapGen4 project (interactive demos and source code).

[3] Azgaar, ”Fantasy Map Generator” (azgaar.github.io/Fantasy-Map-Generator/)

[4] C. Guerra, ”Procedural Island Generation” (brashandplucky.com/2025/09/07/procedural-island-generation-i.html)

[5] C. Mills, ”Notes on Procedural Map Generation Techniques”, (christianjmills.com/posts/procedural-map-generation-techniques-notes/)

[6] C. Kempe, ”Unity Terrain Generation”, (terrain.chriskempke.com/generation)

[7] SciPy documentation: `scipy.spatial.Voronoi` and `scipy.spatial.cKDTree`.

[8] R. H. Whittaker, *Communities and Ecosystems*, 1975. (Background on ecological zonation.)

[9] S. Fortune, “A sweepline algorithm for Voronoi diagrams,” *Proceedings of the second annual symposium on Computational geometry*, 1986.

[10] Representative open-source map generator projects and notebooks demonstrating Voronoi-based polygonal map generation (GitHub repositories and notebooks).