Efficient Maximal Poisson-Disk Sampling

Mohamed S. Ebeida, Anjul Patney, Scott A. Mitchell, Andrew A. Davidson, Patrick M. Knupp, John D. Owens

Sandia National Laboratories, University of California, Davis

Scott - presenter
SIGGRAPH2011
Maximal Poisson-Disk Sampling

• What is MPS?
  – Dart-throwing
  – Insert random points into a domain, build set X

• With the “Poisson” process

Empty disk: \( \forall x_i, x_j \in X, x_i \neq x_j : ||x_i - x_j|| \geq r \)

Bias-free: \( \forall x_i \in X, \forall \Omega \subset D_{i-1} : \)
\[
P (x_i \in \Omega) = \frac{\text{Area}(\Omega)}{\text{Area}(D_{i-1})}
\]

Maximal: \( \forall x \in D, \exists x_i \in X : ||x - x_i|| < r \)
MPS a.k.a.

- Statistical processes
  - Hard-core Strauss disc processes
    - Non-overlap: inhibition distance $r_1$
    - Cover domain: disc radius $r_2$
  - Nature
    - Trees in a forest
      - Variable disk diameter = tree size
      - Points are tree trunks
      - Disks are tree leaves or roots
    - Given satellite pictures (non-maximal)
      - How many trees are there?
      - How much lumber?

- Random sphere packing
  - Non-overlapping $r/2$ disks
  - Atoms in a liquid, crystal

New Mexico mountains

British Columbia
What is MPS good for?

- Graphics – sample points for texture synthesis
  - Generate blue noise distributions for anti-aliasing
  - Without Moire and other visible patterns
- Unbiased process leads to points with
  - No visible patterns between distant points.
    - pairwise distance spectrum close to truncated blue noise powerlaw
- Our eyes sensitive to patterns
- Randomness hides imperfections
  - stare at dry-wall in your house sometime, try to find the seams
What is MPS good for?

- Physics simulations – why SNL paid for my trip 😊
  - Voronoi mesh, cell = points closest to a sample
  - Fractures occur on Voronoi cell boundaries
    - Mesh variation ⊆ material strength variation
    - CVT, regular lattices give unrealistic cracks
  - Unbiased sampling gives realistic cracks
- Ensembles of simulations
- Domains: non-convex, internal boundaries
Algorithm for MPS

- **Classic algorithm**
  - Throw a point, check if disk overlaps, keep/reject
  - Fast at first, but slows due to small uncovered area left.
    Can’t get maximal.

- **Speedup by targeting just the uncovered area**
  - Others use quadtrees to approximate the uncovered area
  - Others use advancing front to sample locally
  - Others use tiles to aid parallelism

- **Common issues**
  - Not strictly “unbiased” process
    - Outcome may be indistinguishable from an unbiased process’s outcome
  - Not maximal: dependent on finite precision
  - Memory or run-time complexity
  - Ours is first provably bias-free, maximal, E(n log n) time O(n) space
Algorithm

- Background square grid
  - Square diagonal = $r$

- Flood fill
  - Build pool of cells $C$: not-exterior to domain

- Phase I: quickly cover most of the domain
  - Pick a square from pool
  - Pick point in square
  - If point uncovered (likely)
    - Keep point
    - Remove square from pool
  - Repeat $a|C|$ times

Initial Pool $C$

End of Phase I: white cells with a point
Algorithm

- Target remaining uncovered area
- Construct square \ disks
  - Polygon easy surrogate for arc-gon
- Replace pool of squares by polygons
- Phase II: repeat
  - Pick polygon from pool
    - Weighted by its area (only log n step)
  - Pick point in polygon
  - If uncovered
    - Keep point
    - Remove polygon from pool
    - Update nearby polygons
- Works well because
  - Voids are scattered
  - Small arc-gons are well approximated by polygons
Algorithm Nuance - Phase II stages

- “Algorithm is simple,… in a good way” - Reviewer
- Lazy update of polygons’ areas and pool, in “stages”
  - More simple datastructures
  - No tree needed, flat array for pool, fewer pointers
  - Run-time proof gets more complicated

Prior slide

Phase II: repeat
  Pick polygon from pool
    Weighted by its area (only log n step)
  Pick point in polygon
  If uncovered
    Keep point
    Remove polygon from pool
  Update nearby polygons

Lazy update

Phase II: repeat
  Repeat c|Pool| times
    Pick polygon from pool
      Weighted by its area (only log n step)
    Pick point in polygon
    If uncovered
      Keep point
    New stage - update all polygons
    Rebuild pool and weights
Complexity Proofs Sketch

- WTS constant time & space per point
  - Everything is local, and constant size
- \#squares = \( \Theta(\#\text{points in sample}) \)
- Sid Meier Civilization template
  - 21 nearby squares, 0 or 1 disks per square
    - By geometry, \( \leq 4 \) voids per cell
    - By geometry, \( \leq 9 \) (8?) disks bounding a void
- Constant time to check if point is uncovered
- Polygons are constant size, time to build
Complexity Proofs Sketch

• Constant work per generated point, but what about the rejected (covered) points?
  – Phase I, $O(|C|)$ throws
  – Phase II
    \[ \text{Area(arcgon)} \geq c \text{Area(polygon)} \iff P(x_i : \text{uncovered}) > c \]
    \[ \iff \# \text{accepted} > c_2 \# \text{rejected} \]

  – Via weighted Voronoi cell of a circle
    • Constant curvature and number of edges

\[ \text{covered fraction of polygon} \]
\[ \text{uncovered arcgon} \]
• Polygons \(\rightarrow\) arcgon as voids get smaller
  – We get more efficient (contrast)

**Polygon & Arc-gon Void Area**

**Voids Covered per Stage**

**Algorithm progress**

**Fewer Rejected Points Later**
Complexity

- Complexity – everything is local, all steps constant time
  - except log(n) to select a polygon, weighted by area
  - that is a relatively inexpensive step
  - constructing geometric primitives is the expensive part
- Constant fraction of generated points are output points

Time = E(Cn + cn log n)
Space = O(n)
Runtime – Why we do Phase I

Number of Points and Voids in Phase I vs. II

- Phase I
  - 73% of points
  - 26% of runtime

CPU Running Time

- slight uptick from log

- 93k points/s trendline
- 358k points/s trendline

- 73% of points start
- 0.73 Total Points
- Initial Voids ≈ 0.73 Total Points
- 26% of runtime

Geometric polygons are relatively expensive.

Phase I
- 73% points
- 22% memory
- 26% time

Phase II
- 27% points
- 78% memory
- 74% time

Saw-tooth from lazy update “stages”
GPU Algorithm

Points generated in parallel, conflicts resolved in an unbiased way

- Point buffers: candidate and final

- Phase I
  - Iterate: synchronize at start of iteration
    - Generate $|C|/5$ candidate points
    - Square states: empty, test, accepted, done
      - Done = Point from prior iterations
      - Test = Point doesn’t conflict with nearby “done” points, compute in parallel
      - Accepted = Point is earlier (id) than conflicting “test” points, compute in parallel
    - Migrate accepted points to done, otherwise remove

- Phase II
  - Construct polygons, compute in parallel
    - Squares “rejected” if covered by prior disks, has no polygon, no work to do
    - Split polygons into triangles
  - Proceed as Phase I, with triangles playing role of squares
**GPU Performance**

NVIDIA GTX 460
2.4x speedup over serial (6.7x memory bandwidth)
1 million points in 1 GB RAM
Unbiased Parallel Sample

10k pts

Rings from inhibition radius

Fourier spectrum
“Unbiased” Opinion

• Unbiased as a description of (serial) **process**
  – insertion probability independent of location
    \[ P(x_i \in \Omega) \propto \text{Area}(\Omega) \]

• Unbaised as a description of **outcome**
  – pairwise distance spectra, blue noise

• Unbiased process leads to unbiased outcome, but so might other processes
  – Opinion: need something beyond “viewgraph norm”
  – Need metrics for “how unbiased is it”
    • Define spectrum \( S \) that is the limit distribution of unbiased sampling, and standard deviations.
    • Our process generated \( S' \), and \(|S-S'| < 0.4 \text{ std dev (S)}\)
Synopsis of Contribution

• Poisson-disk distributions
  – Simple, efficient implementation
  – Provable guarantees
    • Maximal
    • Unbiased
    • O(n) space
    • $E(Cn + \epsilon n \log n)$ time

• Domains
  – 2d
  – Polygons with holes, non-convex

• Algorithmic innovations
  – Two phases
    I. fast to cover most of domain
    II. careful to cover remainder
  – Approximate uncovered “voids”, square $\cap$ circles, with polygons. Careful weighting and selection
Future

• Extensions
  – Could do away with polygonal approximation and weight and sample directly – every dart is a hit! (w/ Thouis Ray Jones)

• Higher dimensions
  – geometric primitives unappealing
  – prefer just use hypercubes

• Thouis Ray Jones, jgt accepted paper
  – model explicit time-of-arrival for each point
  – synchronize locally as needed
  – vs. unbiased by one dart at a time, inherently serial