Fast Maximal Poisson-Disk Sampling by Randomized Tiling

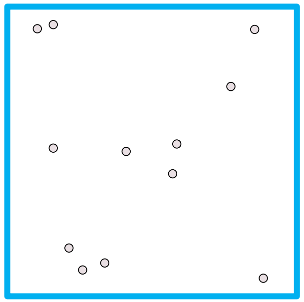
TONG WANG, REIJI SUDA,

IST, University of Tokyo

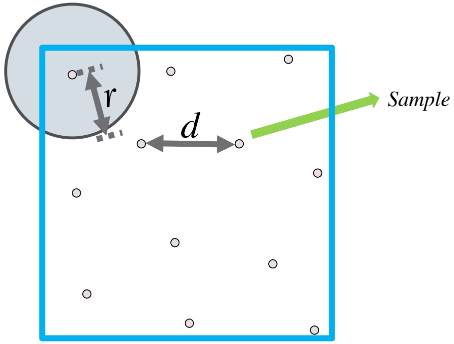
Maximal Poisson Disk Sampling

Poisson-disk sampling

- Definition: Poisson-disk sampling pattern is a group of samples with no two of them within a specified distance *r*.
  - Considered a good property in many applications.



Random

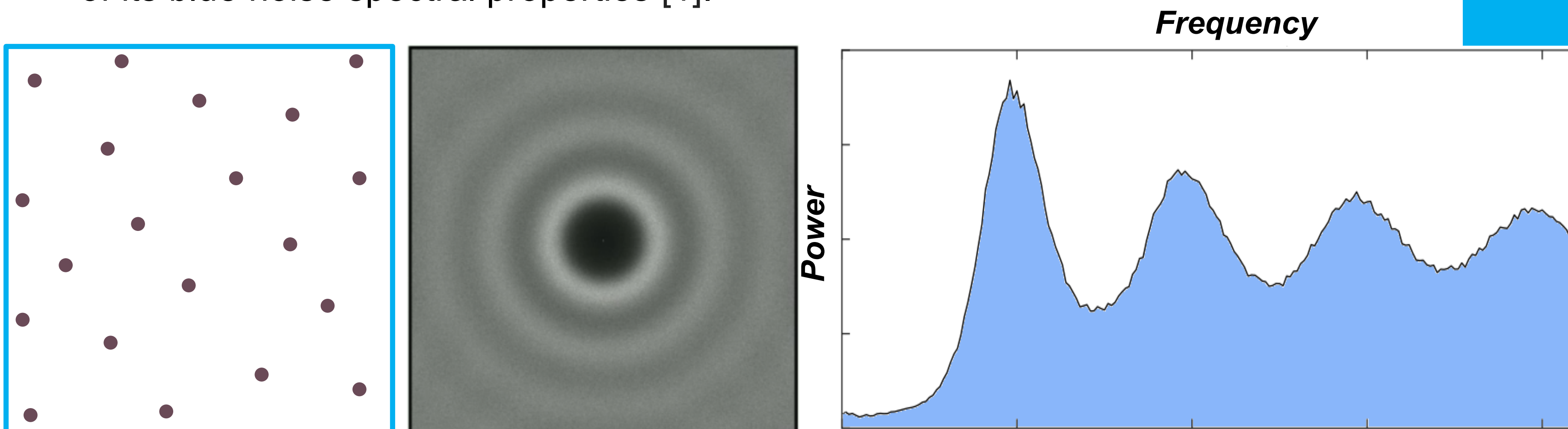


Poisson-disk

- Essentials in many computer graphics applications.
  - Texture generation, Object placement, Rendering and anti-aliasing, Image stippling, etc.

Blue noise property

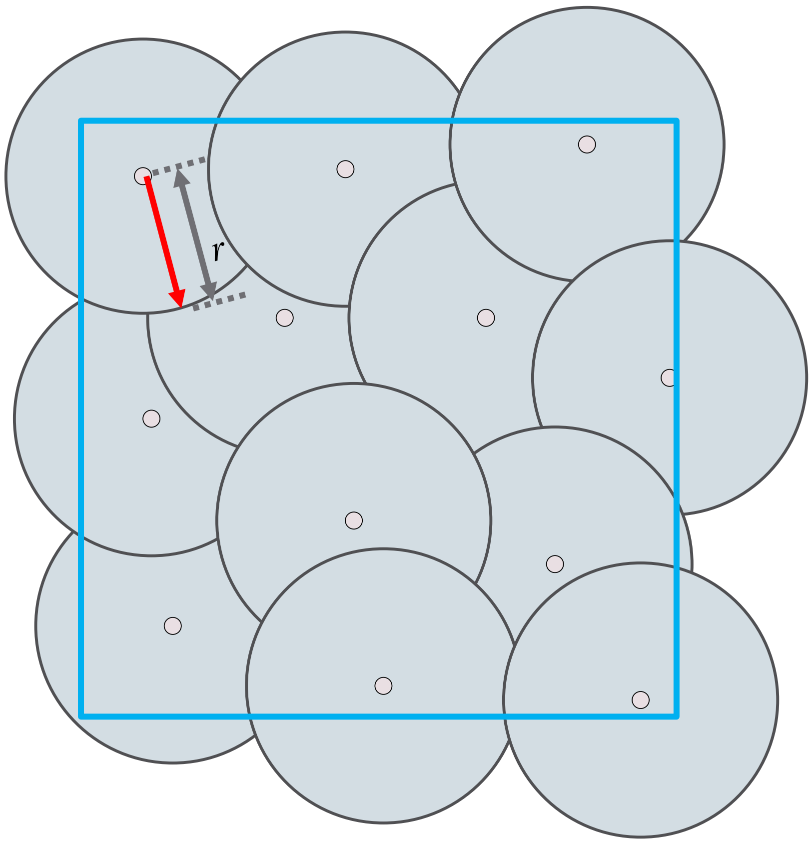
- One reason that Poisson-disk sampling is good for many applications is because of its blue noise spectral properties [1].



Samples *Power Spectrum* ***Radial Mean***

[1] Gamito M N, Maddock S C. Accurate multidimensional Poisson-disk sampling[J]. ACM Transactions on Graphics (TOG), 2009, 29(1): 8.

Maximal Poisson-disk samples (MPS)



- A Poisson-disk sample set is maximal coverage *iff.*

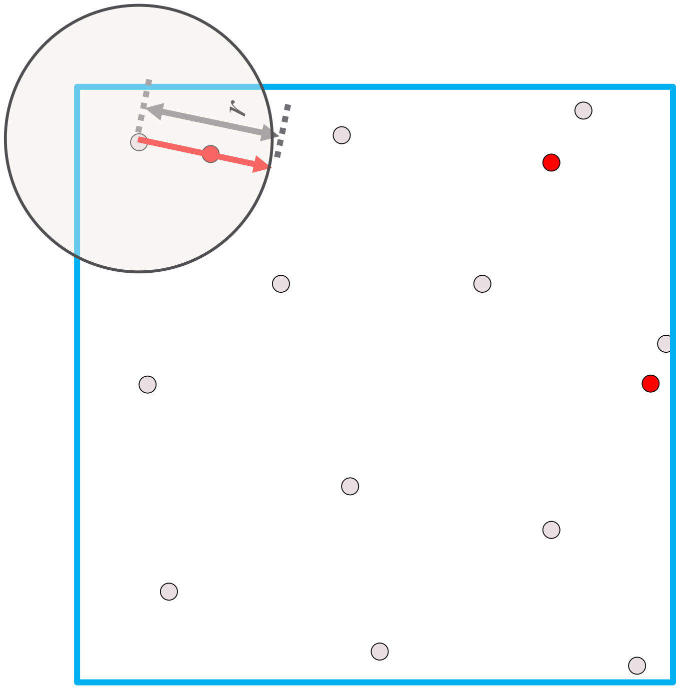
$$G = \{j \in D | \forall i \in X, ||i - j|| \geq r\} = \emptyset$$

For sample set as *X* in domain *D*.

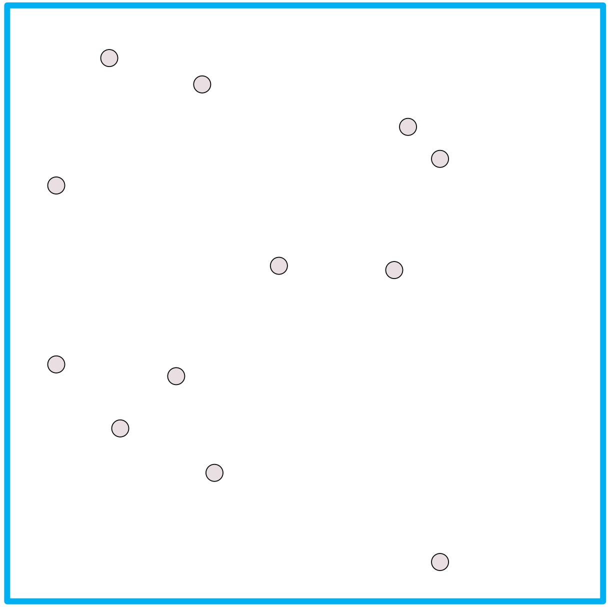
- Guaranteed the quality of samples.
  - No big gaps and clusters.
  - Evenly distributed sampling density.

Generate Poisson-disk samples

- Dart throw



- Relaxation

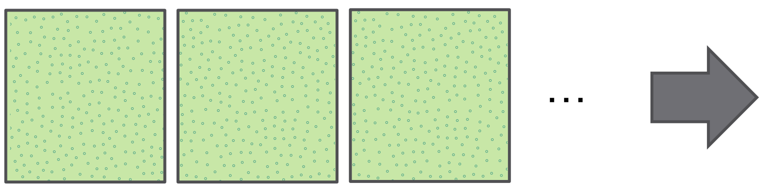


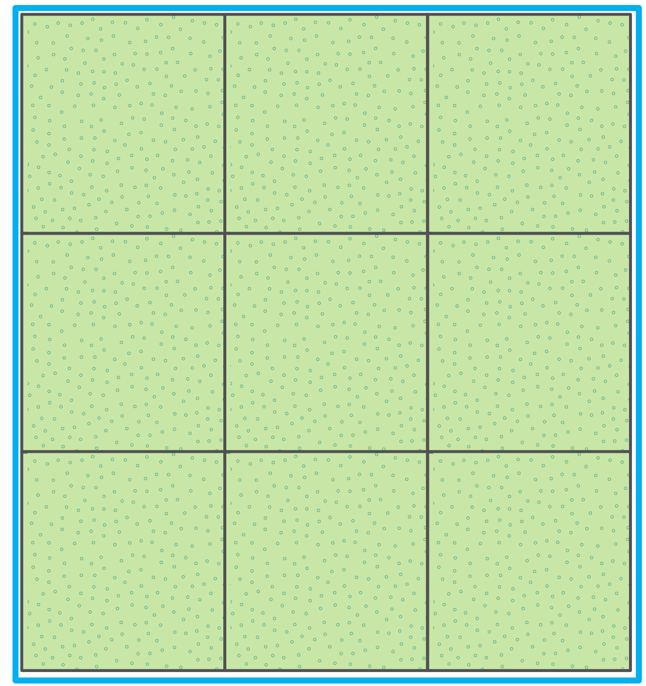
- Often need many iterations to reach maximal coverage.

Tile based method

Sample Plane

Tile Set





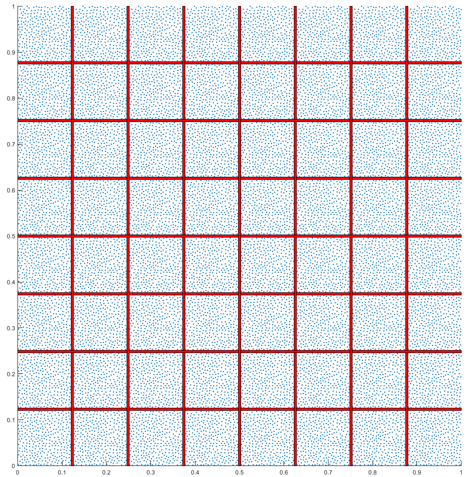
- Preprocessing.
  - Generate tile set, etc.
- Choose a tile from tile set.
- Tile on the plane.

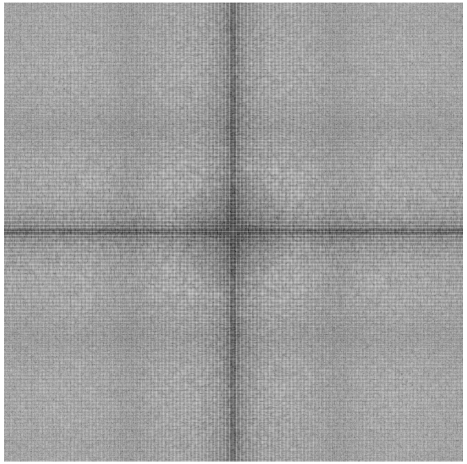
Tile based method

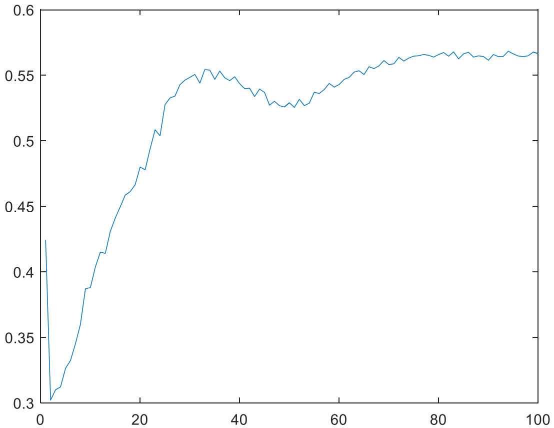
- Extremely fast to fill the sample plane.

But

- Preprocessing takes a long time.
- Regular artifacts are not desirable in many applications.







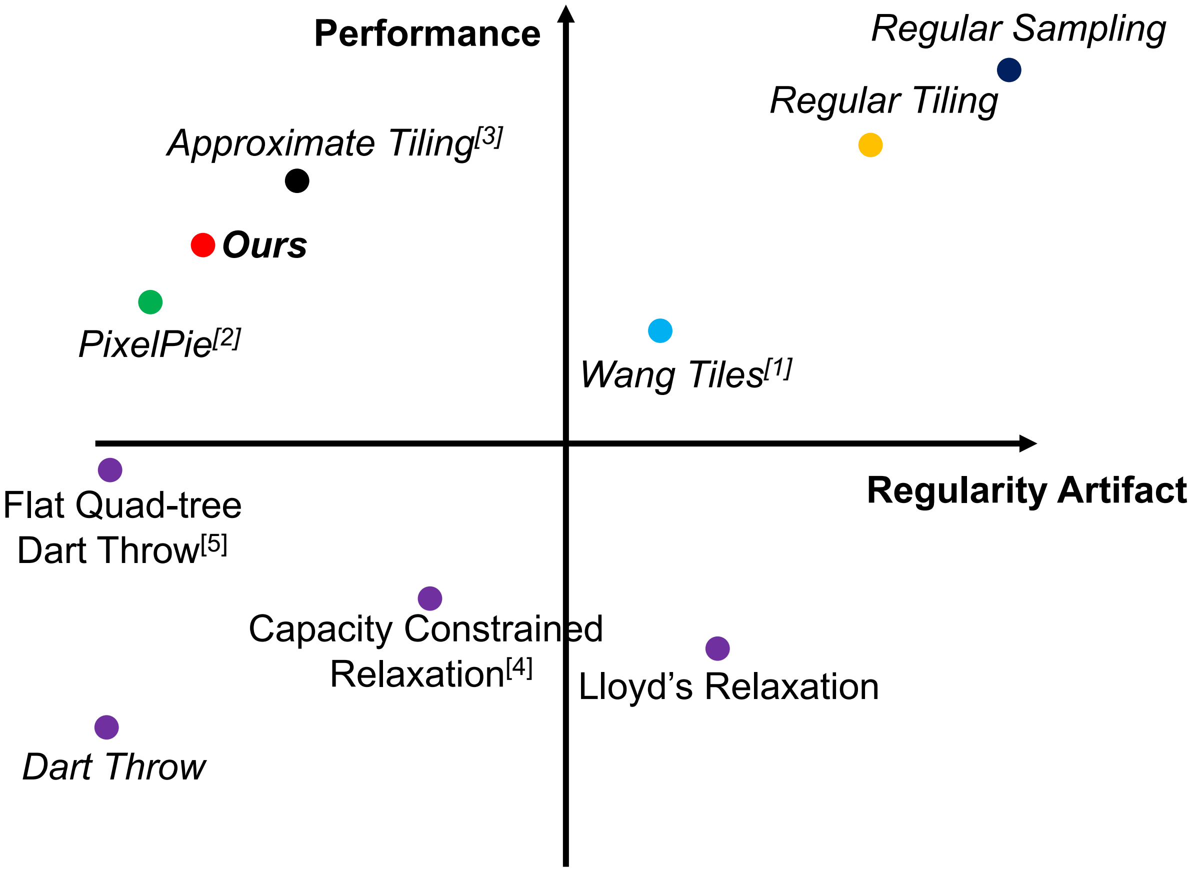
Add more randomness in tiling

- Design the tile sets and tiling techniques: [1][2].
  - Affect preprocessing and tiling performance.
- Transform and rotate tiles to make tiles non-periodic: [3].
  - But [3] didn't deal with sample conflicts, which is not acceptable in many applications.
- Randomly subdivide the space and fetch tiles on-the-fly.
  - Our method.

- [1]. Kopf J, Cohen-Or D, Deussen O, et al. Recursive Wang tiles for real-time blue noise[M]. ACM, 2006.
- [2]. Wachtel F, Pilleboue A, Coeurjolly D, et al. Fast tile-based adaptive sampling with user-specified Fourier spectra[J]. ACM Transactions on Graphics (TOG), 2014, 33(4): 56.
- [3]. Kalantari N K, Sen P. Fast generation of approximate blue noise point sets[C]//Computer Graphics Forum. Blackwell Publishing Ltd, 2012, 31(4): 1529-1535.Transactions on Graphics (TOG), 2012, 31(6): 171.

Performance and Regularity

- For Poisson-disk sample generation:



[1]. [Kopf J, Cohen-Or D, Deussen O, et al. Recursive Wang tiles for real-time blue noise[M]. ACM, 2006.
[2]. Ip C Y, Yalçin M A, Luebke D, et al. Pixelpie: Maximal poisson-disk sampling with rasterization[C]//Proceedings of the 5th High-Performance Graphics Conference. ACM, 2013: 17-26.
[3]. Kalantari N K, Sen P. Fast generation of approximate blue noise point sets[C]//Computer Graphics Forum. Blackwell Publishing Ltd, 2012, 31(4): 1529-1535.Transactions on Graphics (TOG), 2012, 31(6): 171.
[4]. Balzer M, Schlömer T, Deussen O. Capacity-constrained point distributions: a variant of Lloyd's method[M]. ACM, 2009.
[5]. Ebeida M S, Mitchell S A, Patney A, et al. A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions[C]//Computer Graphics Forum. Blackwell Publishing Ltd, 2012, 31(2pt4): 785-794.

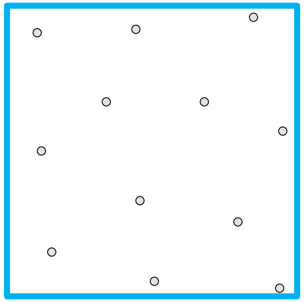
Our algorithm

Algorithm Outline

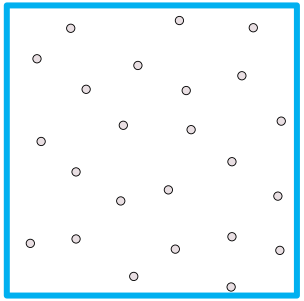
- Prepare a pre-generated MPS pattern.
- KD-tree based randomized tiling (KDRT) is divide-and-conquer based.
  - Divide: Subdivide the sampling plane using KD-tree.
    - Each leaf node square (2d case) is called a building block of the entire sampling plane.
  - Tile: Fill each building block with points clipped from a pre-generated MPS pattern.
  - Conquer: Eliminate conflicts on the edge, and insert new samples in the gap to ensure maximal coverage property.
- Divide and clipping from MPS pattern are randomized, no regular artifacts.
- Conquering process is determined and easy to convergence.

Prepare pre-generated pattern

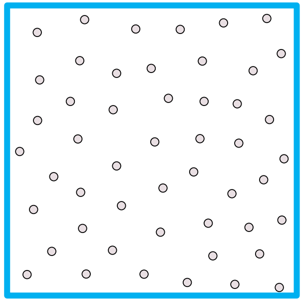
- Use any unbiased Poisson-disk sampling method to generate a pattern.
  - Here we use algorithm described in [1] to generate a pattern.
- Define a scale factor *ratio* as how many times samples that the final sampling plane contains compared with the pattern.
  - The final sample set size is controlled by manipulate this parameter.



Pattern



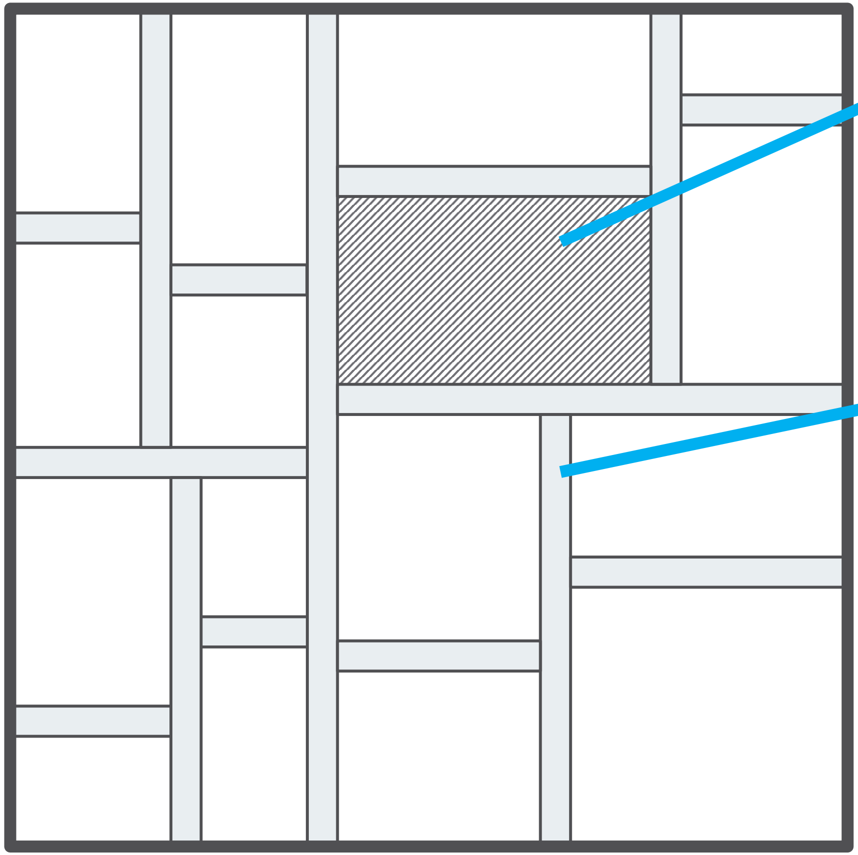
Result Ratio=2



Result Ratio=4

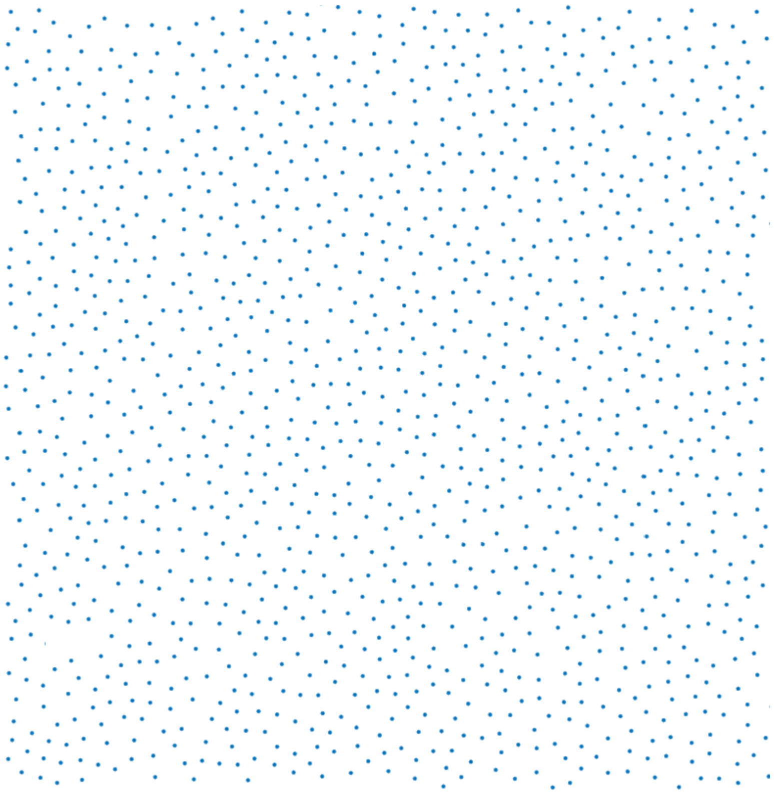
[1]. Ebeida M S, Mitchell S A, Patney A, et al. A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions[C]//Computer Graphics Forum. Blackwell Publishing Ltd, 2012, 31(2pt4): 785-794.

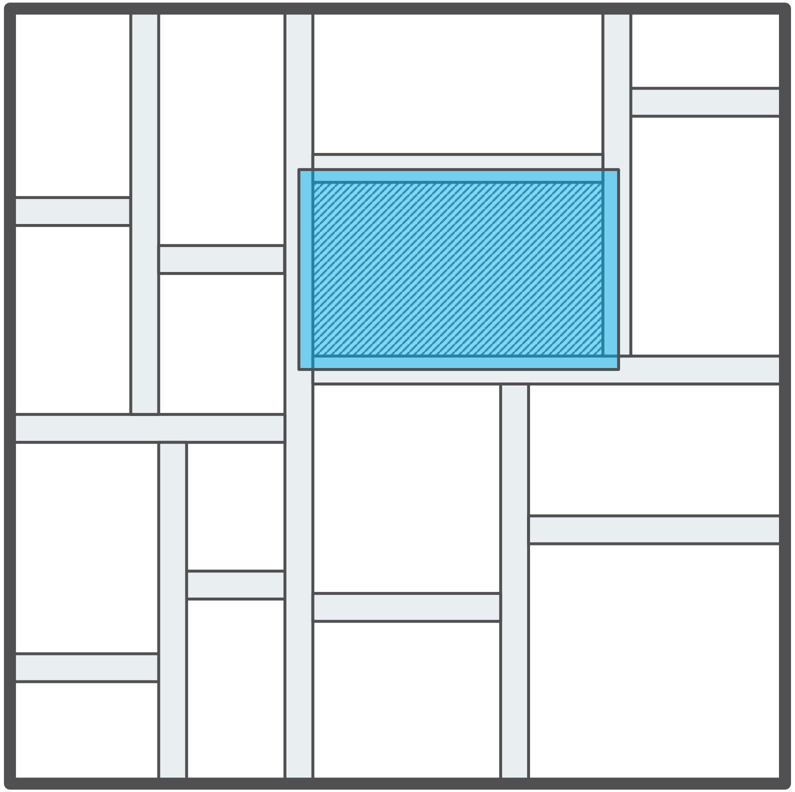
Divide: Subdivide the sampling plane



- Building blocks of the sampling plane.
- Conflict area that will be processed in conquering.

Tiling: Filling building block with samples

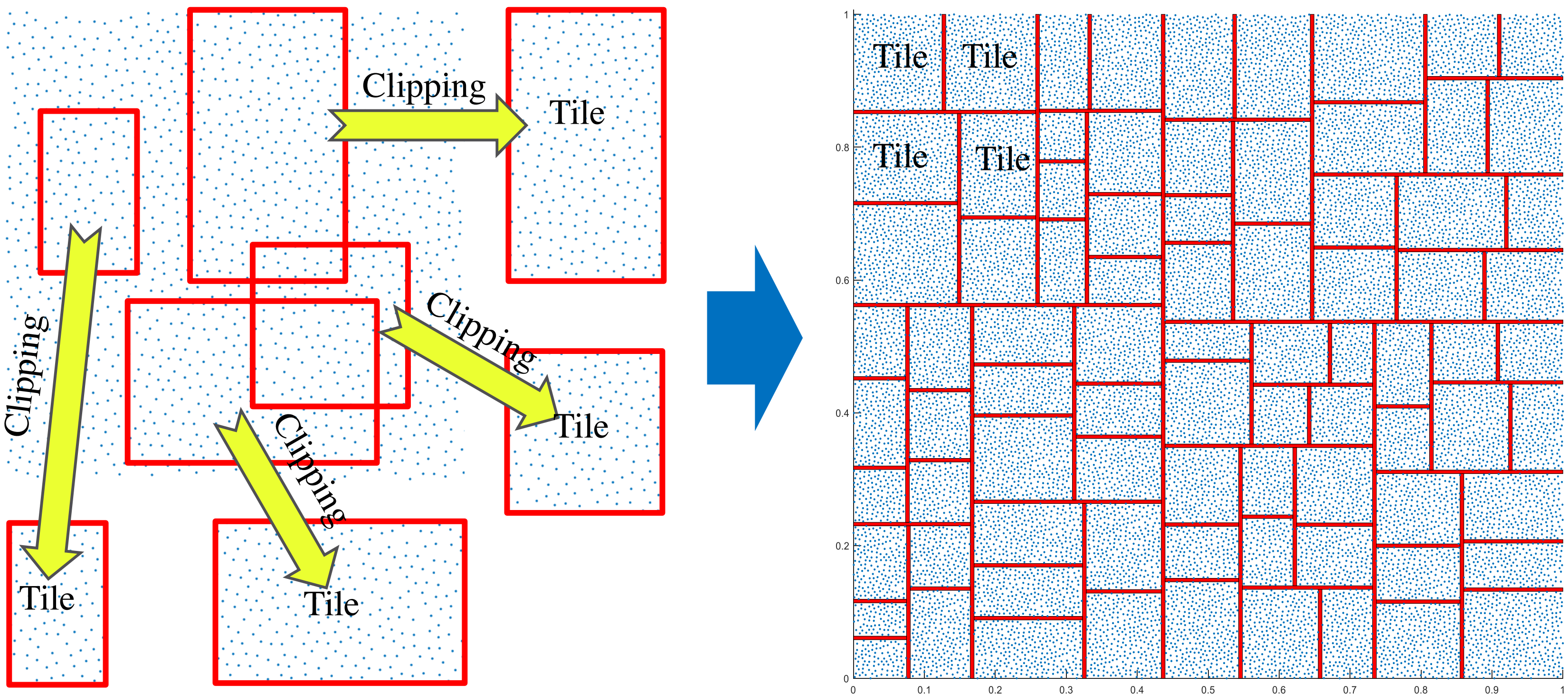




Tiling: Filling building block with samples

MPS Pattern

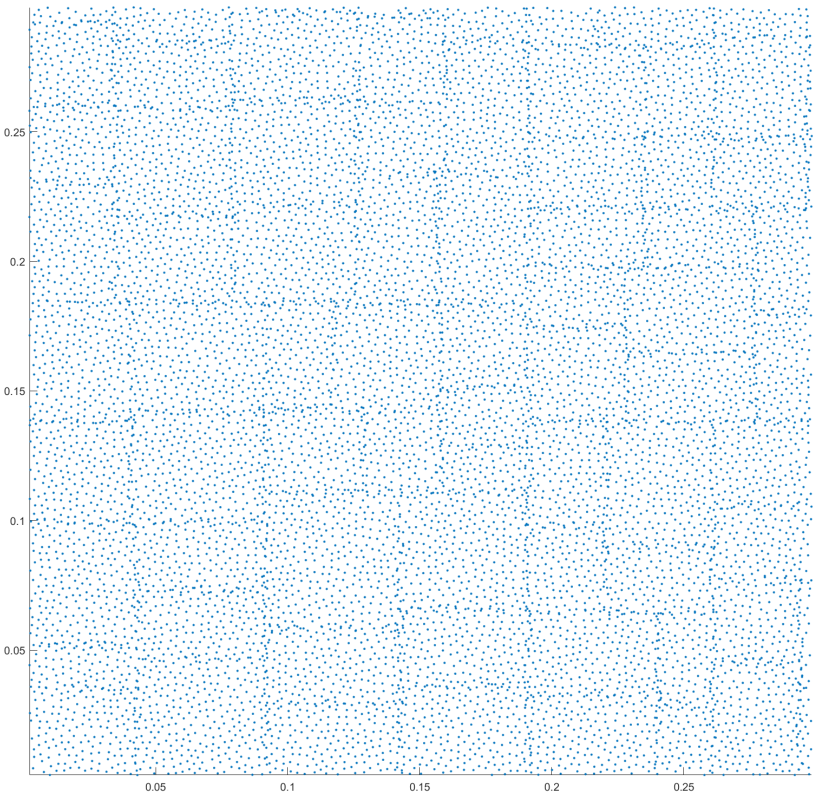
Tilled Sampling Domain



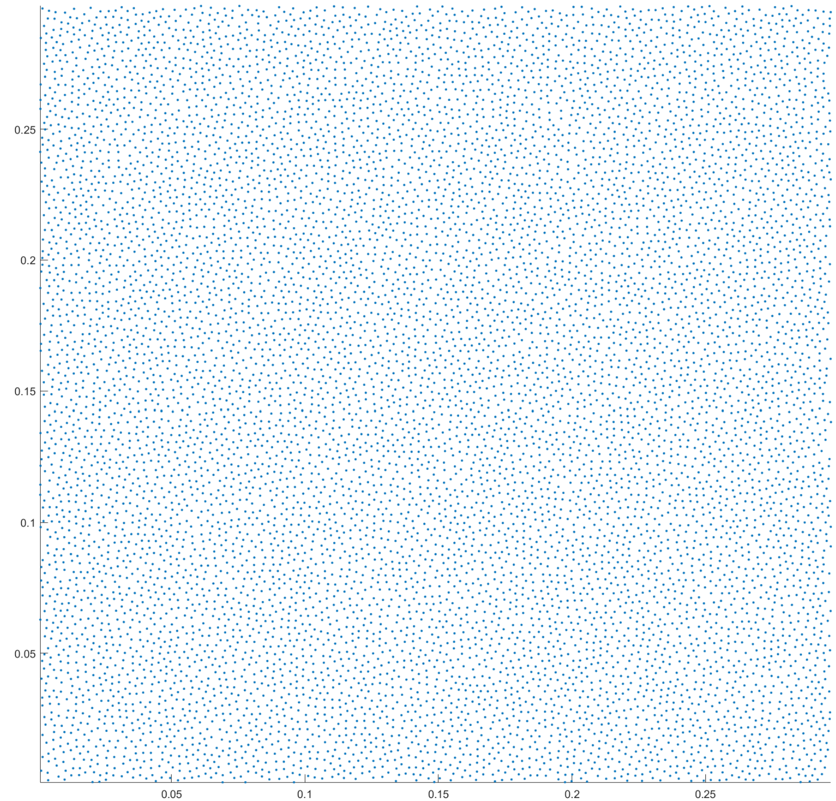
(a)

(b)

Tiling: Filling building block with samples

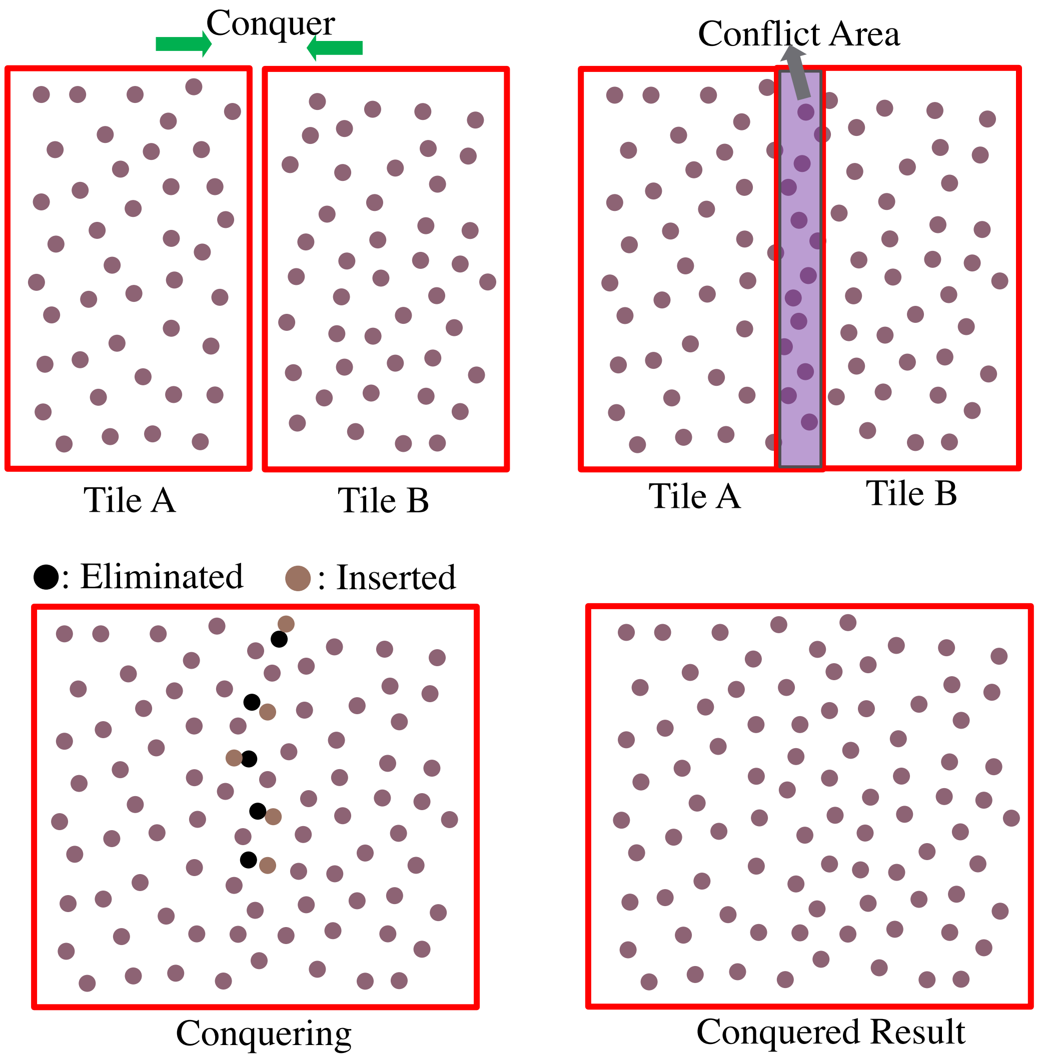


(a)

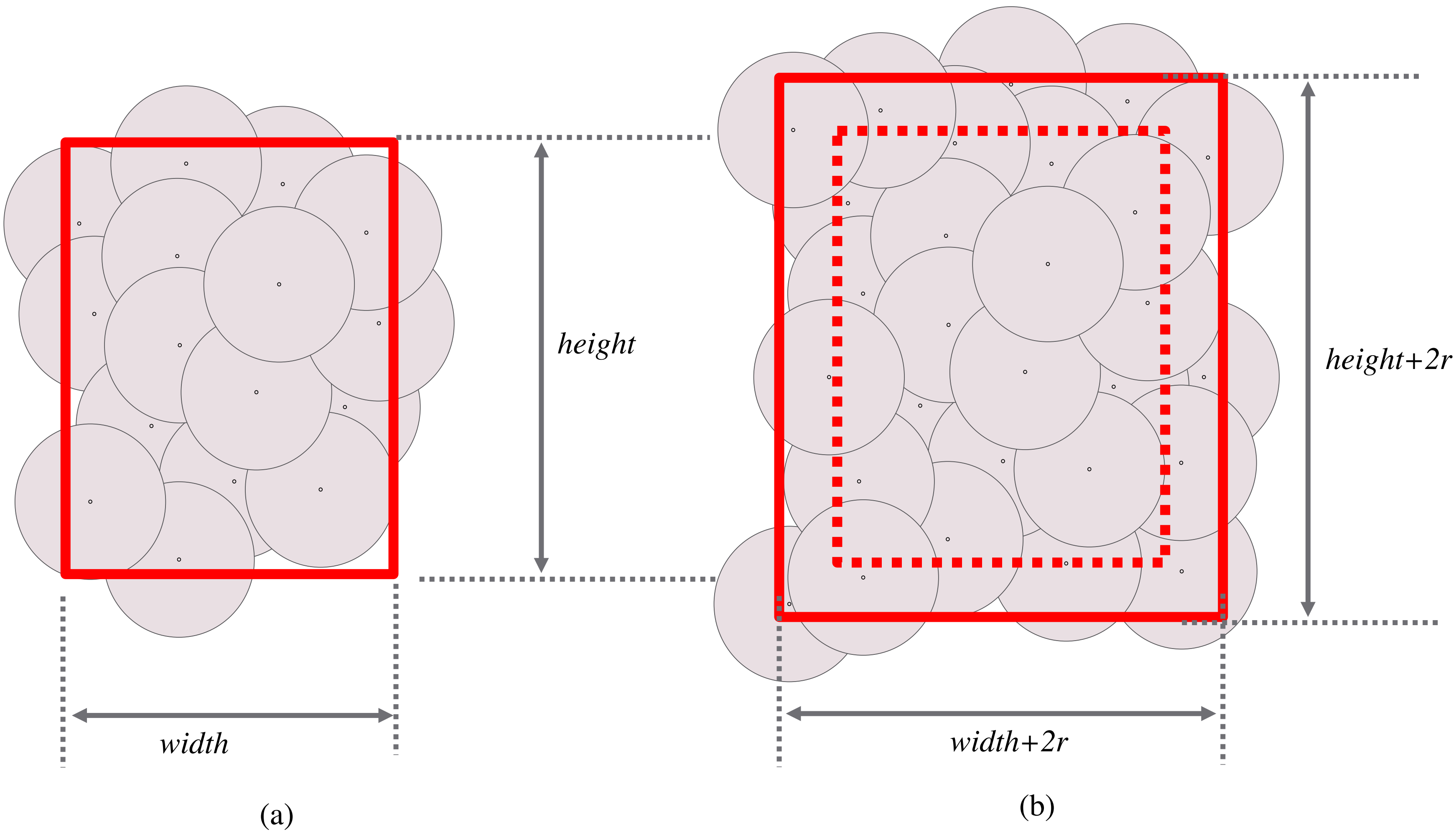


(b)

Conquering: Outline

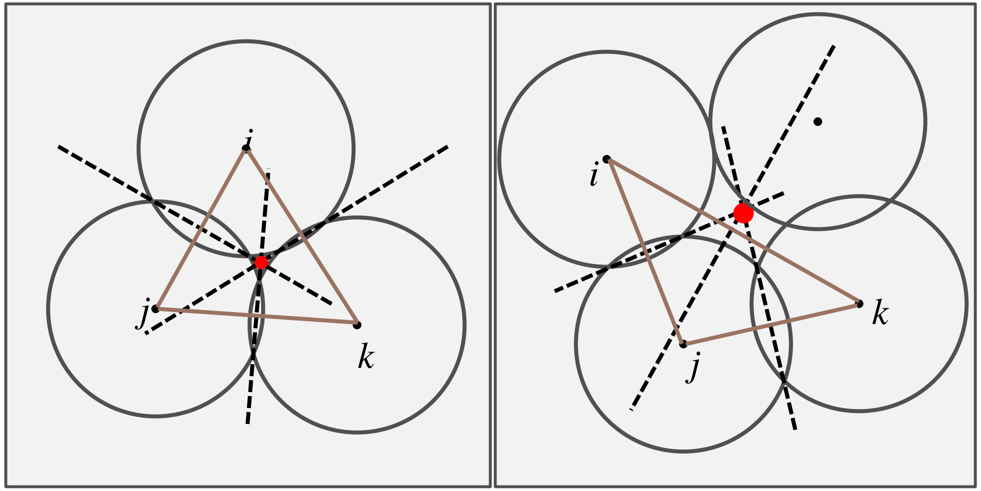


Conquering: Necessary Redundant



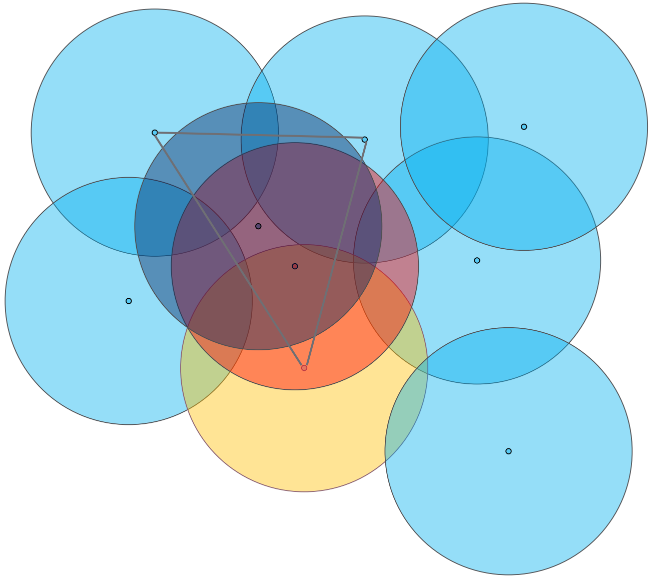
Conquering: Gap detection

- Gap detection[1]: A gap exist among three disks centered at i,j,k (i,j,k $\in$ C), if the circumcenter c of the triangle i,j,k is not covered by any disks.



[1]. Yan D M, Wonka P. Gap processing for adaptive maximal Poisson-disk sampling[J]. ACM Transactions on Graphics (TOG), 2013, 32(5): 148.

Conquering: Elimination



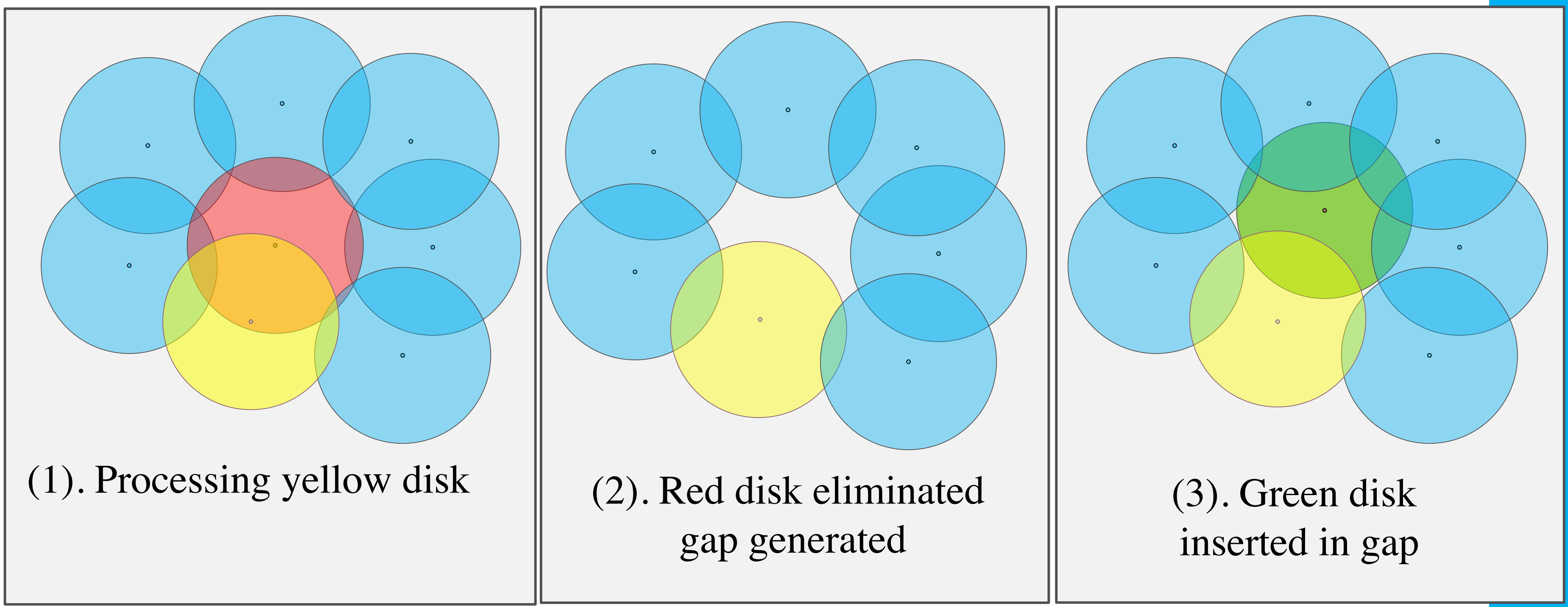
- Processing yellow point.
- Eliminate red point to avoid conflicts.
- Insert blue point to ensure maximal coverage.

Algorithm 3: Psudocode of Gap Detection and Insertion

Function GapDetectionAndInsertion(TestBuffer,i,C);
Input :Sample Point i, TestBuffer, C
Output :C with conflicts removed and new samples inserted
forall *every two samples* m, n *in TestBuffer* **do**
    $center \leftarrow$ Circumcenter(i, m, n);
    if *center in a gap* **then**
        Insert($center, C$);
    else
        continue;
    end
end

Conquering: Insertion

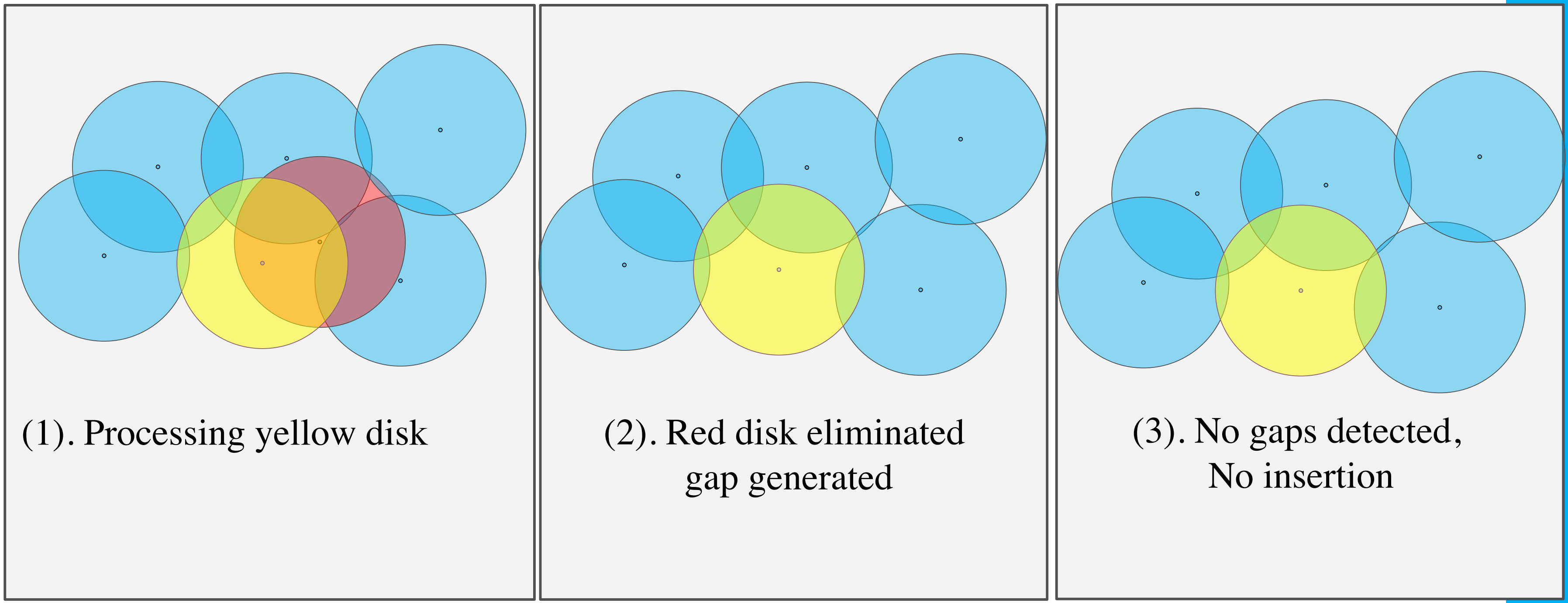
- Notice that there are circumstances that if GapDetectionAndInsertion being applied on all sample points p $\in$ C , it is not guaranteed that all gaps could be covered.



(a) Gaps that cannot be covered by single insertion

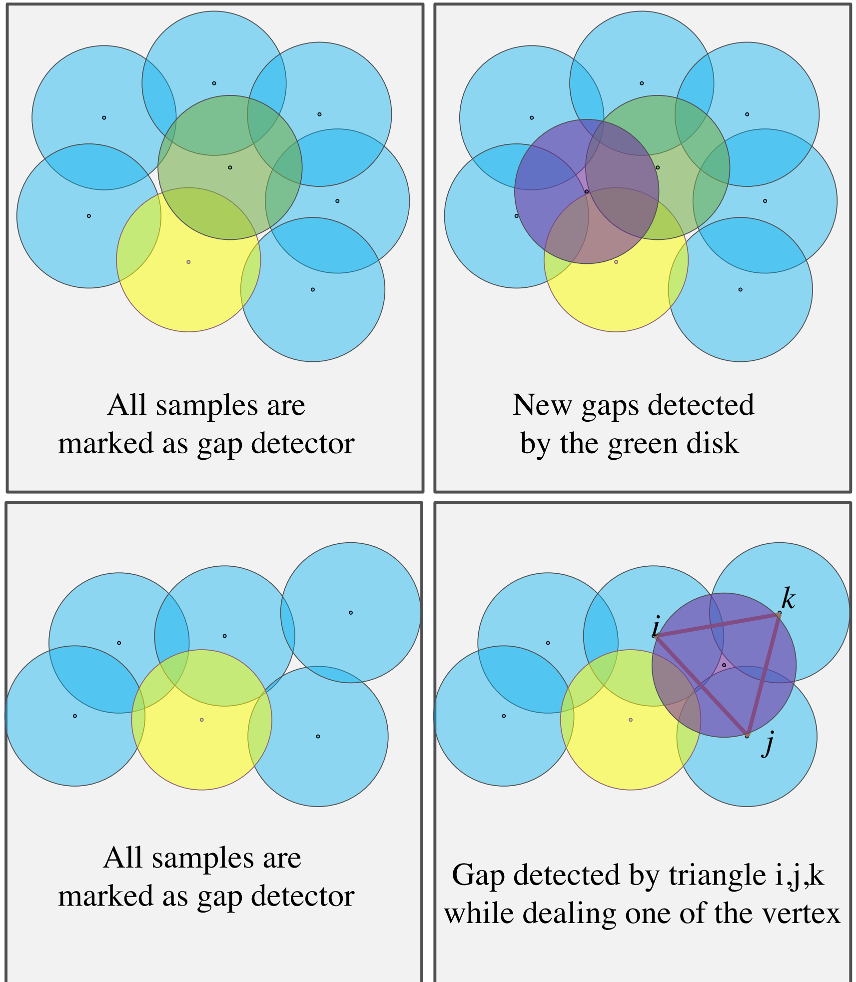
Conquering: Insertion

- Notice that there are circumstances that if GapDetectionAndInsertion being applied on all sample points $p \in C$, it is not guaranteed that all gaps could be covered.



(b) Gaps that cannot be detected by current processing sample

Conquering: Insertion

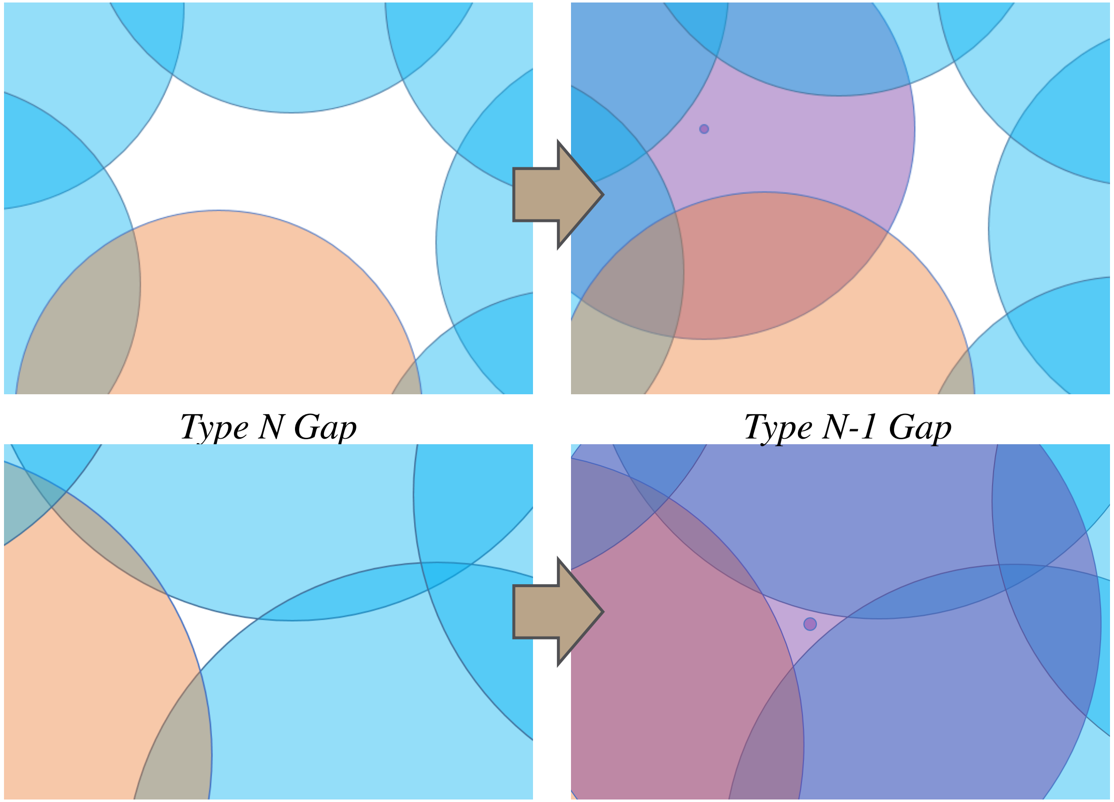


Algorithm 4: Psudocode of Gap Detection and Insertion

```
Function EliminateAndInsert C;
Input   : Clipping and Tiling result C
Output : Maximal Poisson Disk Sample Set S
forall Sample p in conflict area of C do
    Mark p as GapDetector;
    ConflictBuffer ← RadiusSearch(p, r);
    forall conflictPoint in ConflictBuffer do
        Eliminate conflictPoint from C;
        TestBuffer ← RadiusSearch(conflictPoint,2r);
        Mark all samples in TestBuffer as GapDetector;
        GapDetectionAndInsertion(TestBuffer, p, C);
    end
end
Mark all inserted samples as GapDetector;
while True do
    GapDetectorArray← StreamCompaction(C,
     isGapDetector());
    forall Sample p in GapDetectorArray do
        TestBuffer ← RadiusSearch(conflictPoint, 4r);
        GapDetectionAndInsertion(TestBuffer, p, C);
        Mark all inserted samples as GapDetector
    end
    if No new sample point inserted then
        break;
    else
        continue;
    end
end
```

Conquering: Insertion

- All gaps can be detected this way.



Type N Gap

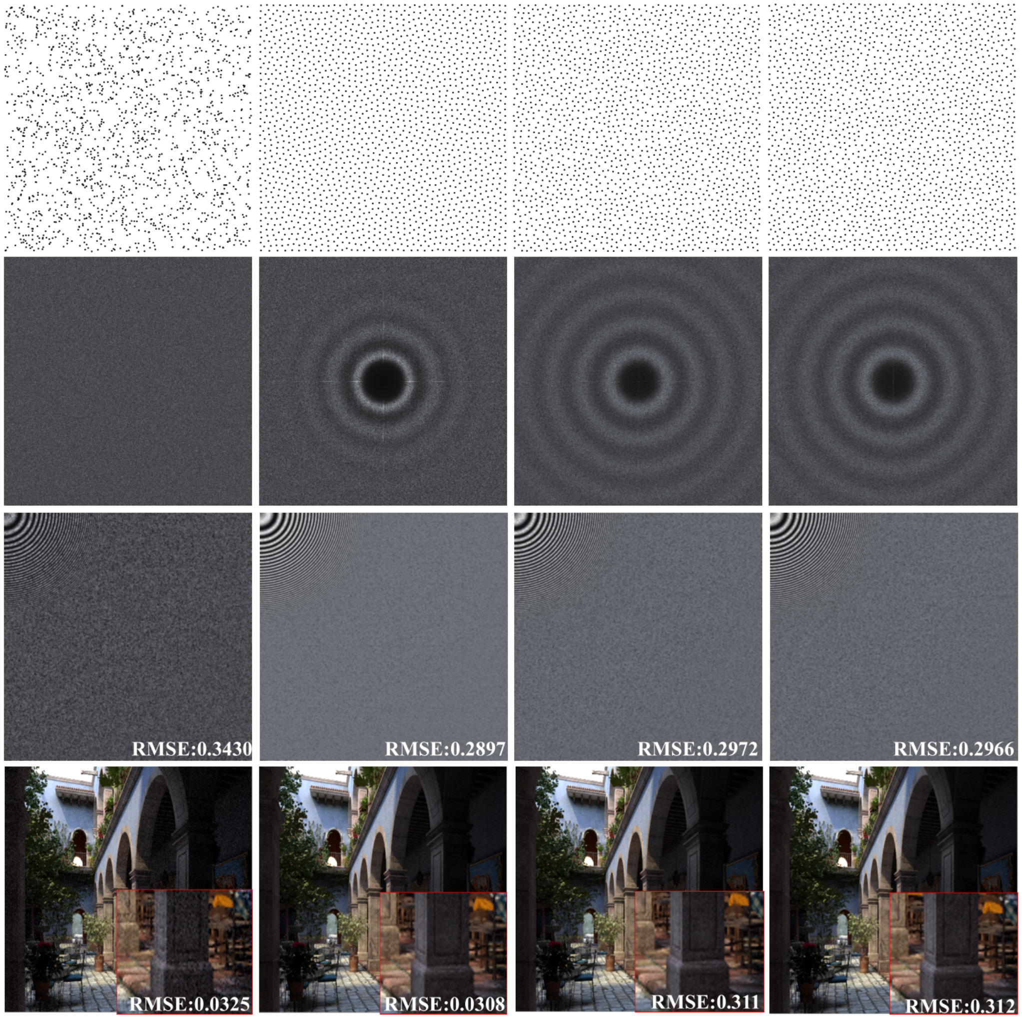
Type N-1 Gap

Type 1 Gap

No Gap

Results

Results



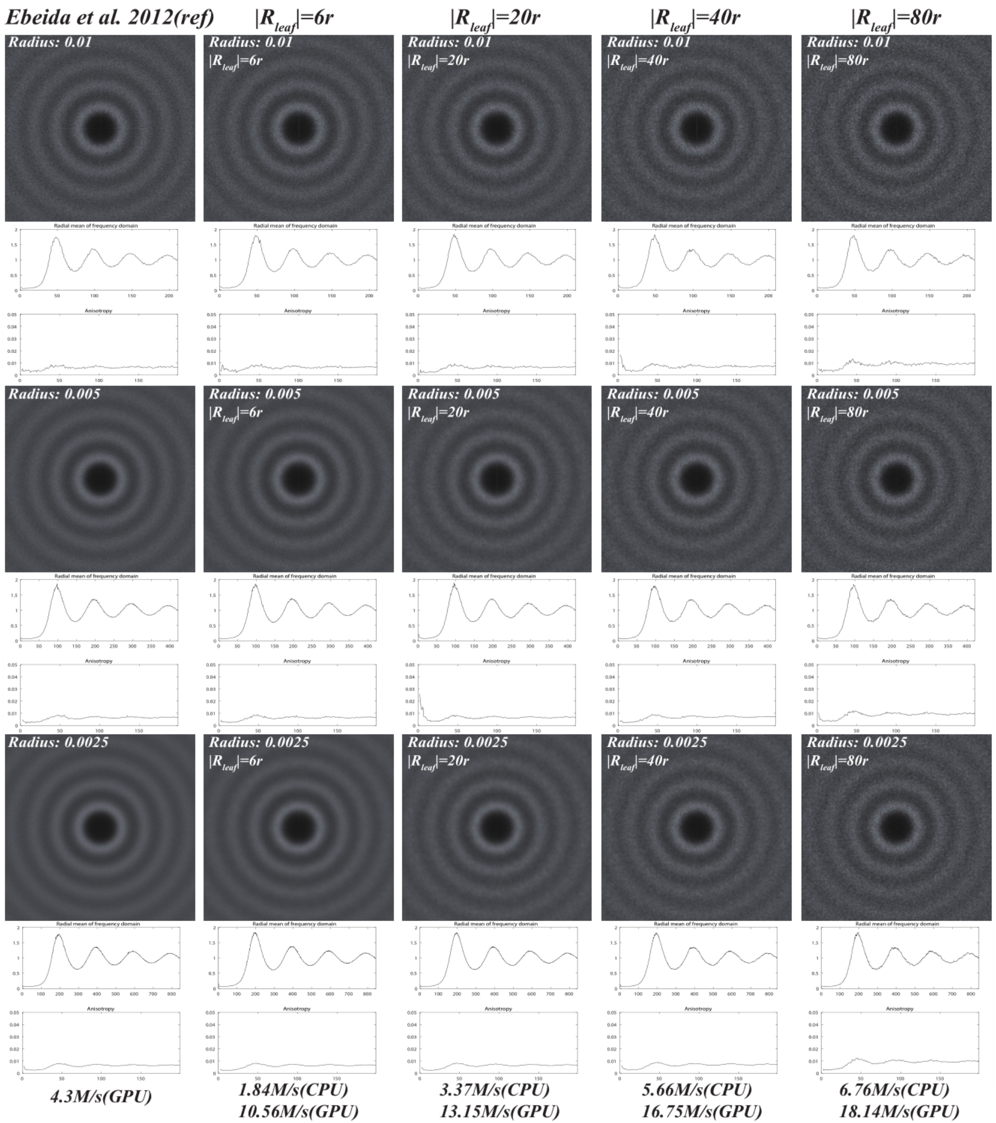
Uniform

De Goes et al, 2012
~20K/s

Ebeida et al. 2012
~4M/s

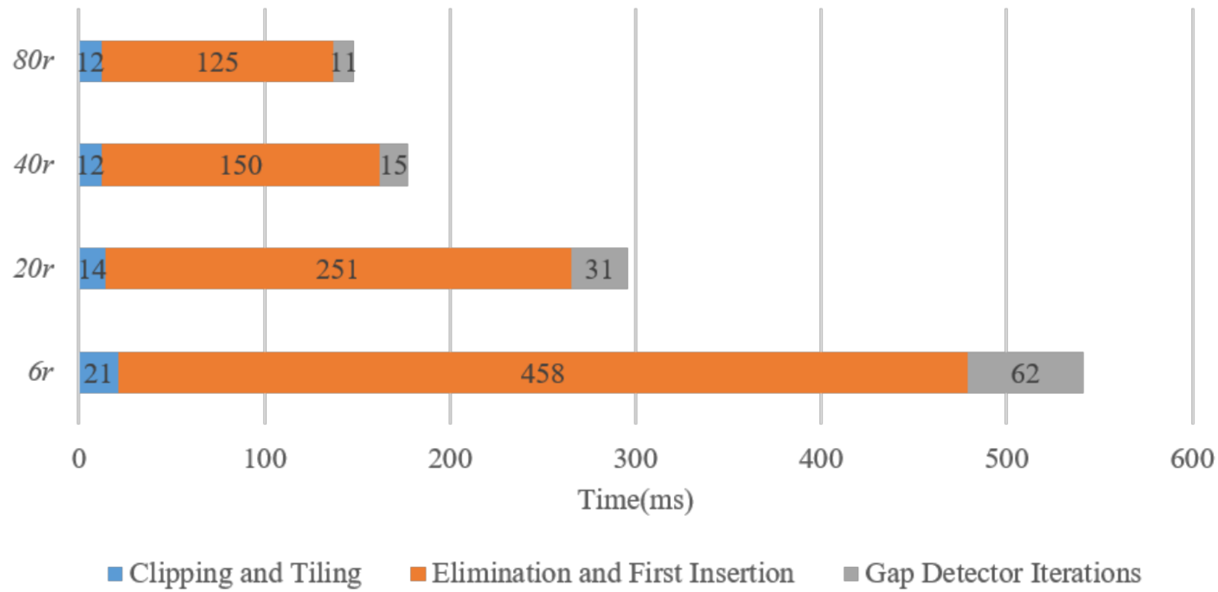
Ours
~16M/s

Results


Ebeida et al. 2012(ref)
|R_leaf|=6r
|R_leaf|=20r
|R_leaf|=40r
|R_leaf|=80r
Radius: 0.01
Radius: 0.005
Radius: 0.0025
Radial mean of frequency domain
Anisotropy
4.3M/s(GPU)
1.84M/s(CPU)
10.56M/s(GPU)
3.37M/s(CPU)
13.15M/s(GPU)
5.66M/s(CPU)
16.75M/s(GPU)
6.76M/s(CPU)
18.14M/s(GPU)

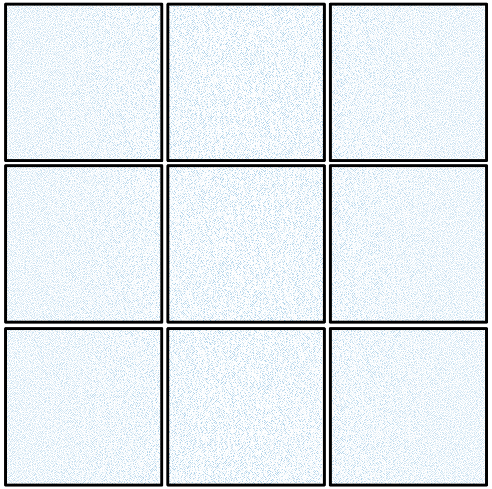
Results

Time taken by each procedure to generate 1 million samples.

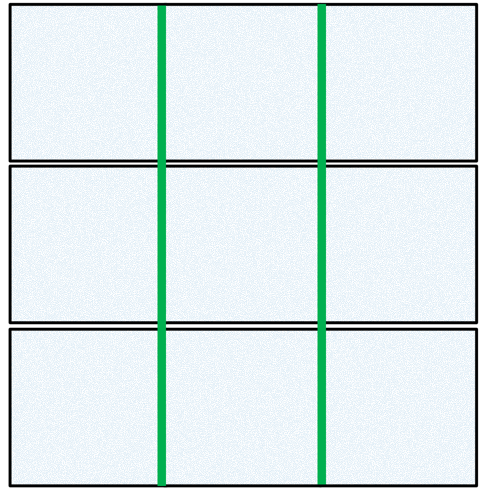


Combining MPS sample sets

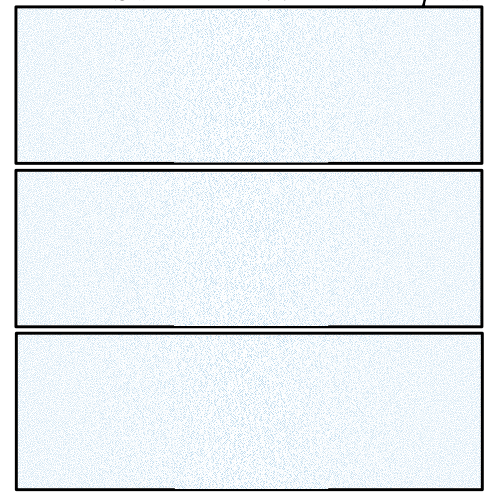
9 MPS set each 1017625 samples



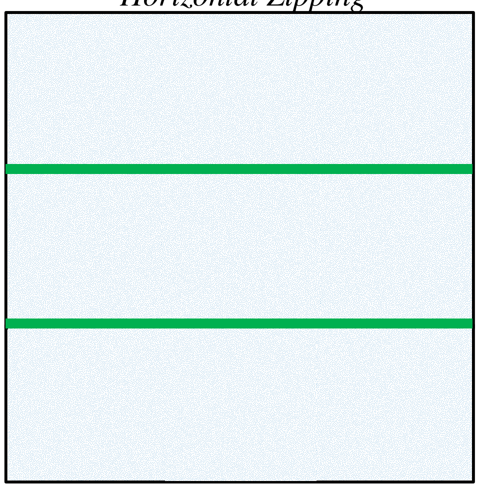
Vertical Zipping



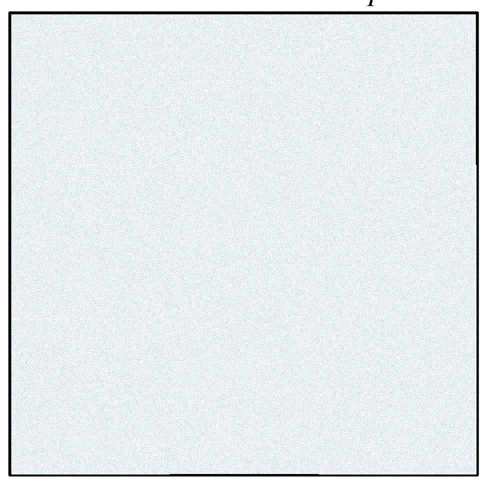
Vertical Zipping Result
3 MPS set each 2991817 samples



Horizontal Zipping



Result with 8795941 samples



Combining Size	***CPU Time (ms)***
3751372	85
4689216	102
5861520	144
7326900	158
9158625	202

Conclusion

- KDST is a tile based method that can generate maximal Poisson-disk samples.
- KDST is a divide and conquer method, including:
  - Divide: Randomly divide the sample plane into building blocks.
  - Tiling: Fill the building blocks with samples clipped from a pre-generated pattern.
  - Conquer: Eliminate conflict points and insert new points in the gap.
- Conquering step can be used to combine MPS sample sets together to form a larger one.
- KDST performs fast on CPU and GPU, and is very easy to implement.

Thank you!

- Thanks for all reviewers for your constructive comments.
- Special thanks to Anjul Patney for his efforts to help the author to get the U.S. visa.