Random Tilings with the GPU

David Keating
Joint work with A. Sridhar

University of California, Berkeley

June 8, 2018

Outline

Why GPUs?

- Common-place in most modern personal computers (as well as XBoxes, Playstations, etc.).

Why GPUs?

- Common-place in most modern personal computers (as well as XBoxes, Playstations, etc.).
- Vector/Data parallelism: to each element of an array of data execute the same instructions in parallel (computer graphics e.g lighting or rendering a surface)

Why GPUs?

- Common-place in most modern personal computers (as well as XBoxes, Playstations, etc.).
- Vector/Data parallelism: to each element of an array of data execute the same instructions in parallel (computer graphics e.g lighting or rendering a surface)
- Design: Several multi-core processors with their own shared memory, each with several cores. A shared global memory. (ex: 10 processors, each with 64 cores, for a total of 640 cores)

Why GPUs?

- Common-place in most modern personal computers (as well as XBoxes, Playstations, etc.).
- Vector/Data parallelism: to each element of an array of data execute the same instructions in parallel (computer graphics e.g lighting or rendering a surface)
- Design: Several multi-core processors with their own shared memory, each with several cores. A shared global memory. (ex: 10 processors, each with 64 cores, for a total of 640 cores)
- *Kernels* contain the instructions that all the cores will execute.

Why GPUs?

- Common-place in most modern personal computers (as well as XBoxes, Playstations, etc.).
- Vector/Data parallelism: to each element of an array of data execute the same instructions in parallel (computer graphics e.g lighting or rendering a surface)
- Design: Several multi-core processors with their own shared memory, each with several cores. A shared global memory. (ex: 10 processors, each with 64 cores, for a total of 640 cores)
- *Kernels* contain the instructions that all the cores will execute.
- Drawbacks: limited memory, limited ability to communicate between cores, *SIMD* architecture bad at handling branching

Domino Tilings

- Consider a simply connected domain $\mathcal{D}$ of the *square lattice*.

Domino Tilings

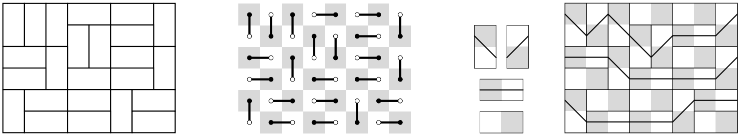
- Consider a simply connected domain $\mathcal{D}$ of the *square lattice*.
- *domino*: the union of two square lattice faces that share an edge.

Domino Tilings

- Consider a simply connected domain $\mathcal{D}$ of the *square lattice*.
- *domino*: the union of two square lattice faces that share an edge.
- A *tiling* $\mathcal{T}$ of $\mathcal{D}$ is set of non intersecting dominos whose union is $\mathcal{D}$. Call $\Omega_{\mathcal{D}}$ the set of all tilings of $\mathcal{D}$.

Domino Tilings

- Consider a simply connected domain $\mathcal{D}$ of the *square lattice*.
- *domino*: the union of two square lattice faces that share an edge.
- A *tiling* $\mathcal{T}$ of $\mathcal{D}$ is set of non intersecting dominos whose union is $\mathcal{D}$. Call $\Omega_{\mathcal{D}}$ the set of all tilings of $\mathcal{D}$.
- Equivalently, a tiling can be viewed as a *perfect matching* or *dimer cover* $\mathcal{T}^*$ of the dual graph or as a lattice routing.



Domino Tilings

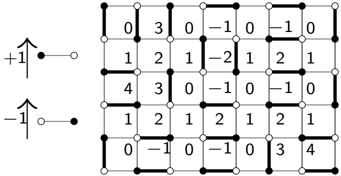
- To each tilings we can associate a *height function* $h_{\mathcal{T}}$ on vertices $v \in \mathcal{D}$.

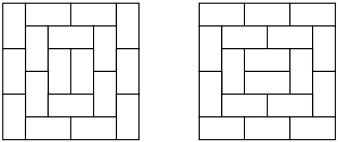
Domino Tilings

- To each tilings we can associate a *height function* $h_{\mathcal{T}}$ on vertices $v \in \mathcal{D}$.
- Partial order on $\Omega_{\mathcal{D}}$: $\mathcal{T} < \mathcal{T}'$ if $h_{\mathcal{T}}(v) < h_{\mathcal{T}'}(v)$ for all $v \in \mathcal{D}$.

Domino Tilings

- To each tilings we can associate a *height function* $h_{\mathcal{T}}$ on vertices $v \in \mathcal{D}$.
- Partial order on $\Omega_{\mathcal{D}}$: $\mathcal{T} < \mathcal{T}'$ if $h_{\mathcal{T}}(v) < h_{\mathcal{T}'}(v)$ for all $v \in \mathcal{D}$.
- In particular, we denote the unique maximal and minimal tilings by $\mathcal{T}_{max}$ and $\mathcal{T}_{min}$, respectively.





Maximal and minimal tilings of a square domain.

Domino Tilings

- We can assign positive weights w_e to the dual edges. A tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{e \in \mathcal{T}^*} w_e$.

Domino Tilings

- We can assign positive weights w_e to the dual edges. A tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{e \in \mathcal{T}^*} w_e$.
- Normalizing by $Z = \sum_{\mathcal{T} \in \Omega_{\mathcal{D}}} W(\mathcal{T})$, defines a probability distribution on the space of tilings.

Domino Tilings

- We can assign positive weights w_e to the dual edges. A tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{e \in \mathcal{T}^*} w_e$.
- Normalizing by $Z = \sum_{\mathcal{T} \in \Omega_{\mathcal{D}}} W(\mathcal{T})$, defines a probability distribution on the space of tilings.
- $P[\mathcal{T}] = \frac{1}{Z} W(\mathcal{T})$

Domino Tilings

- We can assign positive weights w_e to the dual edges. A tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{e \in \mathcal{T}^*} w_e$.
- Normalizing by $Z = \sum_{\mathcal{T} \in \Omega_{\mathcal{D}}} W(\mathcal{T})$, defines a probability distribution on the space of tilings.
- $P[\mathcal{T}] = \frac{1}{Z} W(\mathcal{T})$
- Setting all weights to 1 gives the uniform distribution. Z counts the number of tilings.

Domino Tilings

- We can assign positive weights w_e to the dual edges. A tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{e \in \mathcal{T}^*} w_e$.
- Normalizing by $Z = \sum_{\mathcal{T} \in \Omega_{\mathcal{D}}} W(\mathcal{T})$, defines a probability distribution on the space of tilings.
- $P[\mathcal{T}] = \frac{1}{Z} W(\mathcal{T})$
- Setting all weights to 1 gives the uniform distribution. Z counts the number of tilings.
- Instead, can assign *volume weights* q_v to each vertex. Tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{v \in \mathcal{D}} q_v^{h_{\mathcal{T}}(v)}$.

Domino Tilings

- We can assign positive weights w_e to the dual edges. A tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{e \in \mathcal{T}^*} w_e$.
- Normalizing by $Z = \sum_{\mathcal{T} \in \Omega_{\mathcal{D}}} W(\mathcal{T})$, defines a probability distribution on the space of tilings.
- $P[\mathcal{T}] = \frac{1}{Z} W(\mathcal{T})$
- Setting all weights to 1 gives the uniform distribution. Z counts the number of tilings.
- Instead, can assign *volume weights* q_v to each vertex. Tiling $\mathcal{T}$ has weight $W(\mathcal{T}) = \prod_{v \in \mathcal{D}} q_v^{h_{\mathcal{T}}(v)}$.

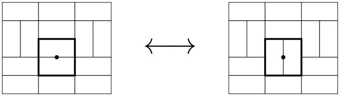
We are trying to sample from these distributions.

Domino Tilings on the GPU

- A tiling is *rotatable* at a vertex v if the faces adjacent to v are covered by two parallel dominos. An *elementary rotation* at v rotates the dominos in place.

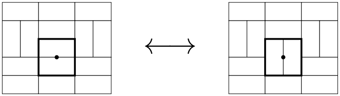
Domino Tilings on the GPU

- A tiling is *rotatable* at a vertex v if the faces adjacent to v are covered by two parallel dominos. An *elementary rotation* at v rotates the dominos in place.



Domino Tilings on the GPU

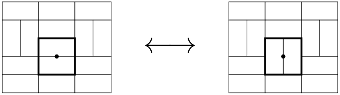
- A tiling is *rotatable* at a vertex v if the faces adjacent to v are covered by two parallel dominos. An *elementary rotation* at v rotates the dominos in place.



- Let $R_v : \Omega_{\mathcal{D}} \to \Omega_{\mathcal{D}}$ be the function that enacts an elementary rotation at v.

Domino Tilings on the GPU

- A tiling is *rotatable* at a vertex v if the faces adjacent to v are covered by two parallel dominos. An *elementary rotation* at v rotates the dominos in place.



- Let $R_v : \Omega_{\mathcal{D}} \to \Omega_{\mathcal{D}}$ be the function that enacts an elementary rotation at v.

Theorem (Thurston)

Two tilings $\mathcal{T}$ and $\mathcal{T}'$ of a domain $\mathcal{D}$ are connected by a sequence of elementary rotations.

Domino Tilings on the GPU

- If v and v' are not adjacent, then $R_v \circ R_{v'} = R_{v'} \circ R_v$.

Domino Tilings on the GPU

- If v and v' are not adjacent, then $R_v \circ R_{v'} = R_{v'} \circ R_v$.
- Call S an *admissible cluster* of vertices if no two vertices in S are adjacent.

Domino Tilings on the GPU

- If v and v' are not adjacent, then $R_v \circ R_{v'} = R_{v'} \circ R_v$.
- Call S an *admissible cluster* of vertices if no two vertices in S are adjacent.
- Define a *cluster rotation*: $R_S = \prod_{v \in S} R_v$.

Domino Tilings on the GPU

- If v and v' are not adjacent, then $R_v \circ R_{v'} = R_{v'} \circ R_v$.
- Call S an *admissible cluster* of vertices if no two vertices in S are adjacent.
- Define a *cluster rotation*: $R_S = \prod_{v \in S} R_v$.
- Markov Chain: A random walk on $\Omega_{\mathcal{D}}$ with initial tiling $\mathcal{T}^{(0)}$ and random clusters $\{S_i\}$ has n^{th} step

$$\mathcal{T}^{(n)} = R^{(n)}(\mathcal{T}), \text{ where } R^{(n)} = R_{S_n} \circ \ldots \circ R_{S_1}$$

Theorem

In the limit $n \to \infty$ the random tiling $\mathcal{T}^{(n)}$ is uniformly distributed.

Domino Tilings on the GPU

Coupling from the Past:

- An algorithm due to Propp and Wilson; ensures exact sampling.

Domino Tilings on the GPU

Coupling from the Past:

- An algorithm due to Propp and Wilson; ensures exact sampling.
- Define *backward walk* with initial tiling $\mathcal{T}^{(0)}$ and random clusters $\{S_i\}$ has n^{th} step

$$\mathcal{T}^{(-n)} = R^{(-n)}(\mathcal{T}^{(0)}), \text{ where } R^{(-n)} = R_{S_1} \circ \ldots \circ R_{S_n}$$

Domino Tilings on the GPU

Coupling from the Past:

- An algorithm due to Propp and Wilson; ensures exact sampling.
- Define *backward walk* with initial tiling $\mathcal{T}^{(0)}$ and random clusters $\{S_i\}$ has n^{th} step

$$\mathcal{T}^{(-n)} = R^{(-n)}(\mathcal{T}^{(0)}), \text{ where } R^{(-n)} = R_{S_1} \circ \ldots \circ R_{S_n}$$

- Almost surely there exists n such that $|R^{(-n)}(\Omega_{\mathcal{D}})| = 1$. We say the Markov chain has *collapsed*.

Domino Tilings on the GPU

Coupling from the Past:

- An algorithm due to Propp and Wilson; ensures exact sampling.
- Define *backward walk* with initial tiling $\mathcal{T}^{(0)}$ and random clusters $\{S_i\}$ has n^{th} step

$$\mathcal{T}^{(-n)} = R^{(-n)}(\mathcal{T}^{(0)}), \text{ where } R^{(-n)} = R_{S_1} \circ \ldots \circ R_{S_n}$$

- Almost surely there exists n such that $|R^{(-n)}(\Omega_{\mathcal{D}})| = 1$. We say the Markov chain has *collapsed*.
- The unique element in the range of $R^{(-n)}$ is distributed uniformly.

Domino Tilings on the GPU

Coupling from the Past:

- An algorithm due to Propp and Wilson; ensures exact sampling.
- Define *backward walk* with initial tiling $\mathcal{T}^{(0)}$ and random clusters $\{S_i\}$ has n^{th} step

$$\mathcal{T}^{(-n)} = R^{(-n)}(\mathcal{T}^{(0)}), \text{ where } R^{(-n)} = R_{S_1} \circ \ldots \circ R_{S_n}$$

- Almost surely there exists n such that $|R^{(-n)}(\Omega_{\mathcal{D}})| = 1$. We say the Markov chain has *collapsed*.
- The unique element in the range of $R^{(-n)}$ is distributed uniformly.
- We can construct R so that it preserves the partial ordering. The Markov chain has collapsed iff $R^{(-n)}(\mathcal{T}_{max}) = R^{(-n)}(\mathcal{T}_{min})$.

Implementation

- At each vertex assign assign a *state* $s_v = e_0 + 2e_1 + 4e_2 + 8e_3$, where we enumerate the edges adjacent to v in order N, S, E, W and e_i is 1 if a domino crosses edge i, 0 otherwise.

Implementation

- At each vertex assign assign a *state* $s_v = e_0 + 2e_1 + 4e_2 + 8e_3$, where we enumerate the edges adjacent to v in order N, S, E, W and e_i is 1 if a domino crosses edge i, 0 otherwise.
- v is rotatable if $s_v = 12$ or $s_v = 3$.

Implementation

- At each vertex assign assign a *state* $s_v = e_0 + 2e_1 + 4e_2 + 8e_3$, where we enumerate the edges adjacent to v in order N, S, E, W and e_i is 1 if a domino crosses edge i, 0 otherwise.
- v is rotatable if $s_v = 12$ or $s_v = 3$.
- Admissible clusters S are chosen by: first checkerboard coloring the vertices, choose a color at random, then choose a random subset of the given color.

Implementation

- At each vertex assign assign a *state* $s_v = e_0 + 2e_1 + 4e_2 + 8e_3$, where we enumerate the edges adjacent to v in order N, S, E, W and e_i is 1 if a domino crosses edge i, 0 otherwise.
- v is rotatable if $s_v = 12$ or $s_v = 3$.
- Admissible clusters S are chosen by: first checkerboard coloring the vertices, choose a color at random, then choose a random subset of the given color.
- An algorithm of Thurston efficiently constructs $\mathcal{T}_{max}$ and $\mathcal{T}_{min}$ for a given domain.

Implementation

Data parallelism:

- Kernels *Rotate* and *Update* implement R_S.

Implementation

Data parallelism:

- Kernels *Rotate* and *Update* implement R_S.
- First the GPU independently attempts rotations at all, say, white vertices (*Rotate*). Then it updates the state at all black vertices (*Update*).

Implementation

Data parallelism:

- Kernels *Rotate* and *Update* implement R_S.
- First the GPU independently attempts rotations at all, say, white vertices (*Rotate*). Then it updates the state at all black vertices (*Update*).
- Substantial parallelism: Each vertex of the tiling can be assigned to a work unit of the GPU; there is minimal communication necessary between work units.

Implementation

Data parallelism:

- Kernels *Rotate* and *Update* implement R_S.
- First the GPU independently attempts rotations at all, say, white vertices (*Rotate*). Then it updates the state at all black vertices (*Update*).
- Substantial parallelism: Each vertex of the tiling can be assigned to a work unit of the GPU; there is minimal communication necessary between work units.
- A random walk is constructed by repeatedly applying *Rotate* and *Update* a set number of times.

Implementation

DominoTilerCFTP:

Compute the extremal tilings $\mathcal{T}_{max}$ and $\mathcal{T}_{min}$.

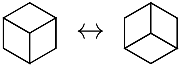
Initialize a list *seeds* with a random real number.

Repeat:

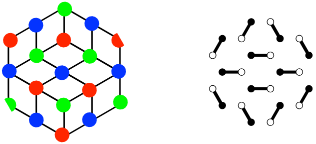
Initialize $\mathcal{T}_{top} = \mathcal{T}_{max}$
Initialize $\mathcal{T}_{bottom} = \mathcal{T}_{min}$
For $i = 1$ to length of *seeds*:
Set $\mathcal{T}_{top} = RandomWalk(\mathcal{T}_{top}, seeds_i, 2^i)$
Set $\mathcal{T}_{bottom} = RandomWalk(\mathcal{T}_{bottom}, seeds_i, 2^i)$
If $\mathcal{T}_{top} = \mathcal{T}_{bottom}$, return $\mathcal{T}_{bottom}$.
Else, push a new random number to the beginning of *seeds*.

Lozenge Tilings

- Triangular lattice equivalent of domino tilings.
- Can be viewed as perfect matchings of the hexagonal lattice.
- Can also be associated to a height function h_T on vertices.
- Can be viewed as stacks of cubes.
- Elementary rotations:

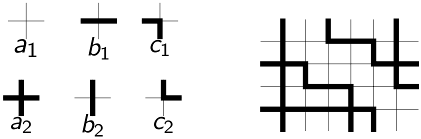


- Admissible clusters chosen by tri-coloring vertices.

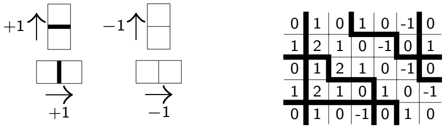


Six Vertex Model

- Assign "occupied" or "unoccupied" to the edges of a domain $\mathcal{D}$ of the square lattice, such that the possible local configurations are:



- Each type of local configuration is assigned a weight.
- Configurations are in bijection with height functions.

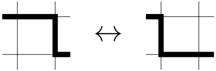


Six Vertex Model

- Gibbs measure given by:

$$Prob(S) = \frac{1}{Z} W(S), \quad Z = \sum_{S} W(S)$$

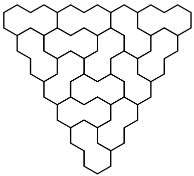
- Elementary moves are flips across faces:

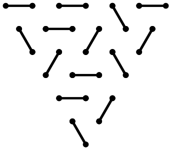


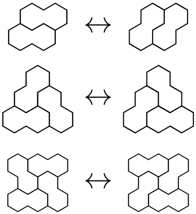
- For fixed boundary conditions, the space of configurations is connected under elementary flips.
- Admissible cluster can be chosen by tricoloring faces.

Bibone Tilings

- Hexagonal lattice equivalent of Domino tilings.
- Can be viewed as perfect matchings of the triangle lattice (non-bipartite).
- Do not admit height functions.
- Set of tilings connected under three types of elementary rotations:



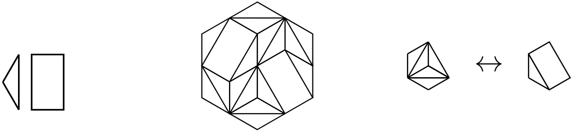




- For each type of local move one can find admissible clusters.

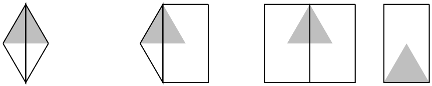
Rectangle-triangle Tilings

- Tilings of domains of the triangle lattice by isosceles triangles and rectangles with side lengths 1 and $\sqrt{3}$.
- Can be visualized as stacks of half-cubes (gives a partial ordering).
- Connected by single elementary move (adding/removing a half cube).



Rectangle-triangle Tilings

- Can assign local weights t, c, and r, to faces of the triangular lattice in the following way:



t c r

- The weight of a tiling is given by the product of the weights of all the faces.
- Admissible clusters chosen as in the Lozenge case.

Results

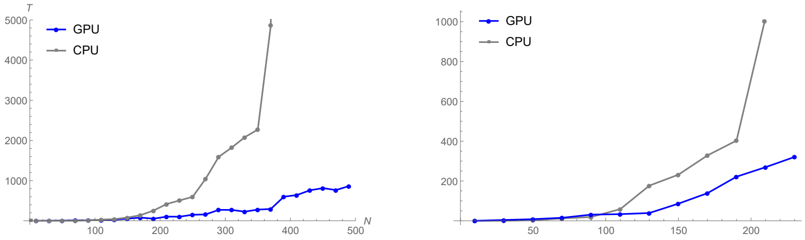
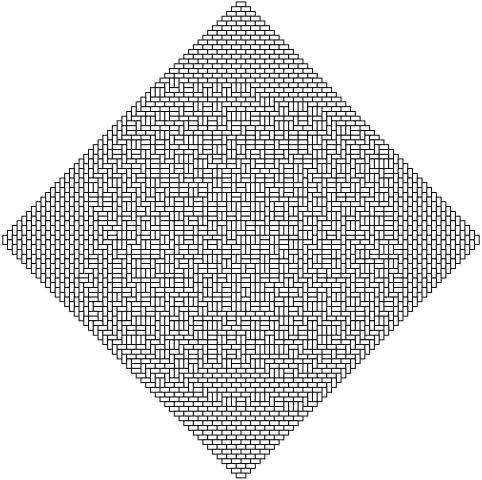
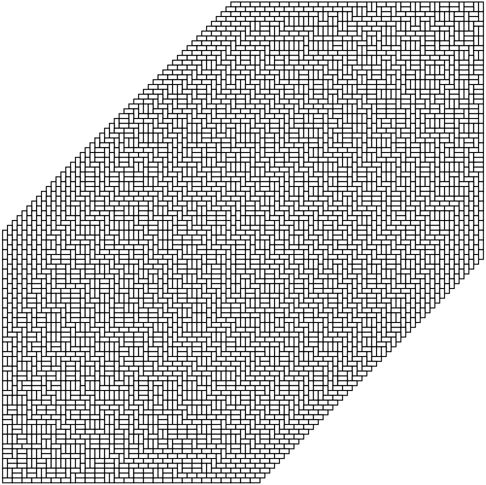
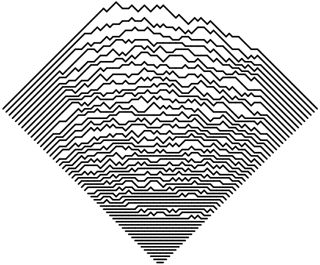


Figure: (A) The time T in seconds to generate, with coupling-from-the-past, a random configuration of the six-vertex model on an $N \times N$ sized domain with domain wall boundary conditions and weights $(a, b, c) = (1, 1, 1)$. (B) The time to generate a random domino tiling of an $N \times N$ square.







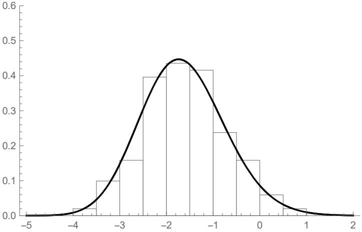


Figure: Johannson showed that the fluctuations of the top-most path converges to the Airy process. In particular, the y-intercept of the path as shown above, after appropriate rescaling, converges to the Tracy-Widom distribution F_2. Right shows a normalized histogram of the y-intercept computed from 100 random tilings of an Aztec diamond of size 300, with the distribution F_2 superimposed.

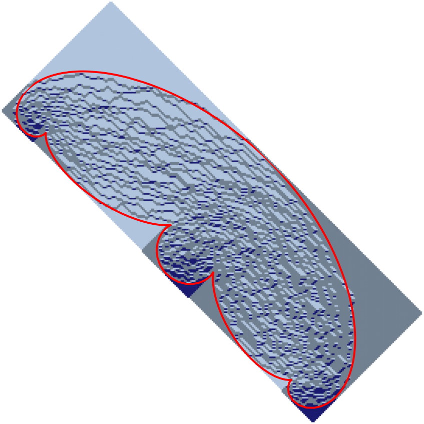


Figure: A tiling of a rectangular Aztec diamond, with the Arctic curve superimposed in red.

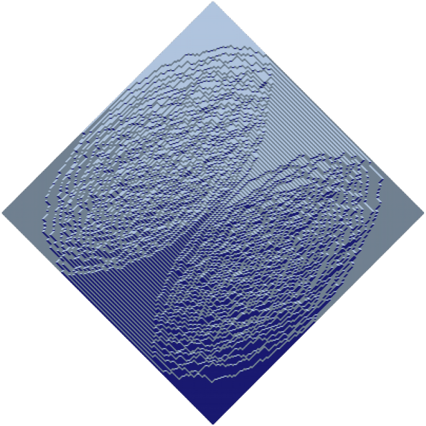


Figure: A random tiling of the Aztec diamond with volume weights $q = 20$ for all black vertices and $q = 1/20$ for all white vertices.

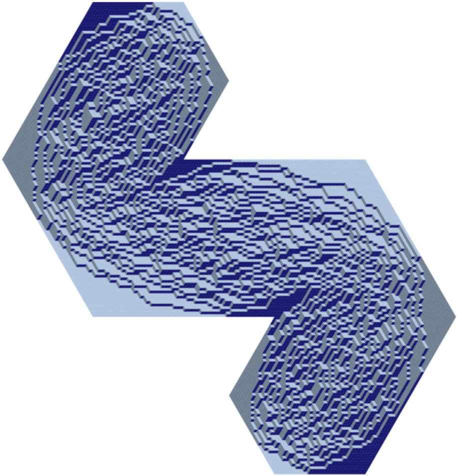


Figure: A random tiling of a weird region by lozenges.

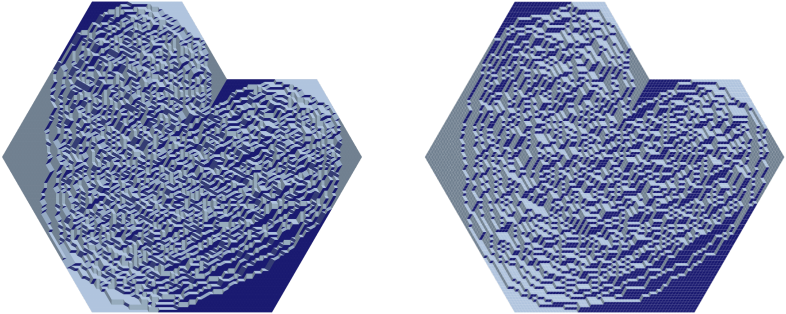


Figure: A tiling of a partial hexagon (A) by rectangles and triangles, and (B) by lozenges.

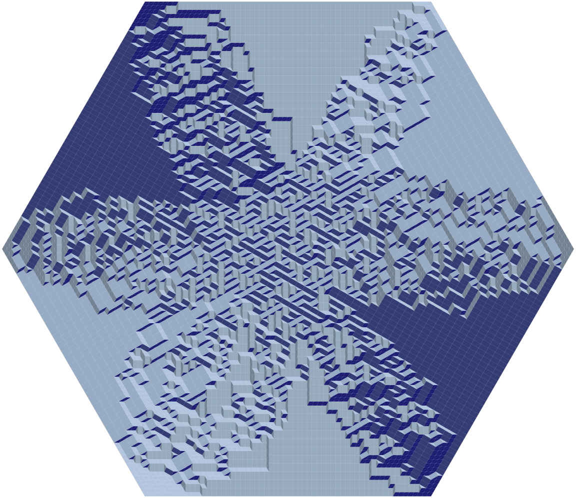


Figure: Choosing weights $t = .5, r = 1, c = 1$ produces tilings that look like snowflakes.

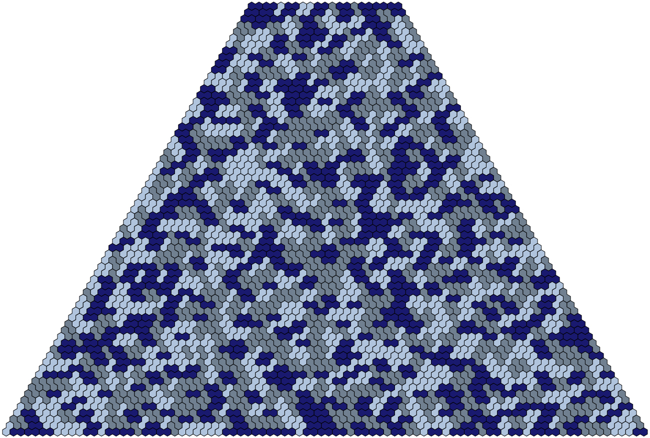


Figure: A tiling of a trapezoid by bibones.

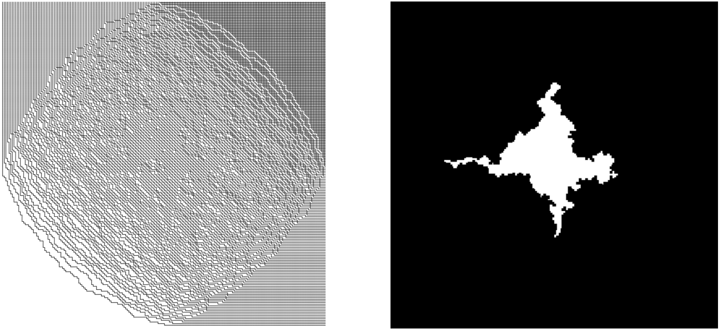


Figure: The six-vertex model with weights $a = 1, b = 1, c = \sqrt{8}$, ($\Delta = -3$), and domain wall boundary conditions. (A) shows a random configuration and (B) shows only the gaseous region.

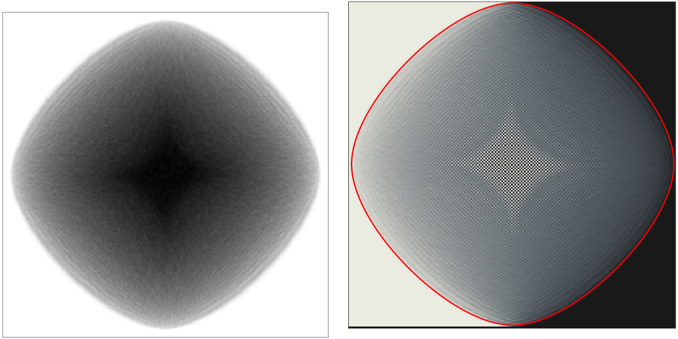


Figure: (A) shows the average density of c-vertices and (B) shows the average density of horizontal edges computed, with 1000 random configurations. The arctic curve is superimposed in red.

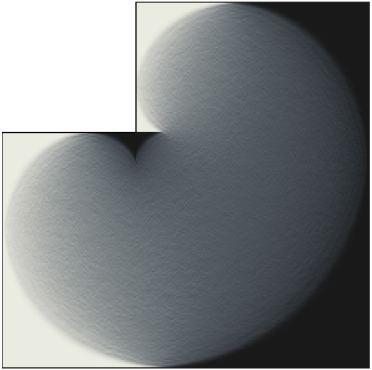


Figure: The average density of horizontal edges in with weight $\Delta = 0$ in an L-shaped region with domain wall type boundary conditions, computed with 1000 samples.

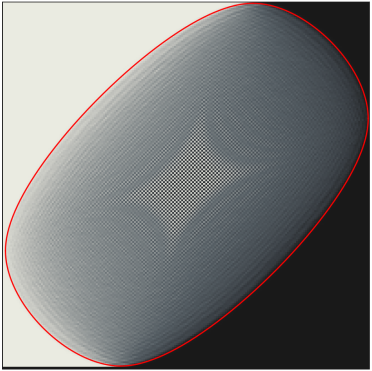
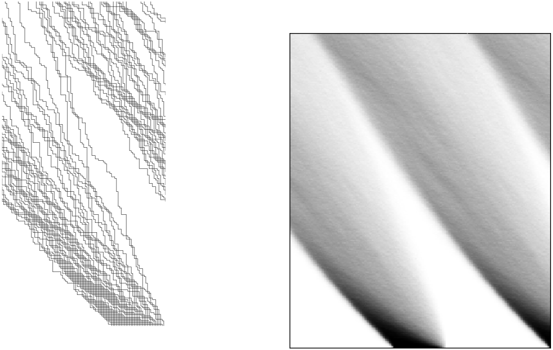


Figure: The average density of horizontal edges with weights $a = 2b$, $\Delta = -3$, computed with 1000 samples.



(A) (B)

Figure: The six-vertex model with weights $(a_1, a_2, b_1, b_2, c_1, c_2) = (1, 1, .3, .7, .3, .7)$ on a cylinder. (A) shows a random configuration on the cylinder. (B) shows the average density of paths, taken over 100 sample.

End!

https://github.com/GPUTilings