Random 3D Polyforms

Dhruv Gramopadhye, James Serrano, Khoi Tran

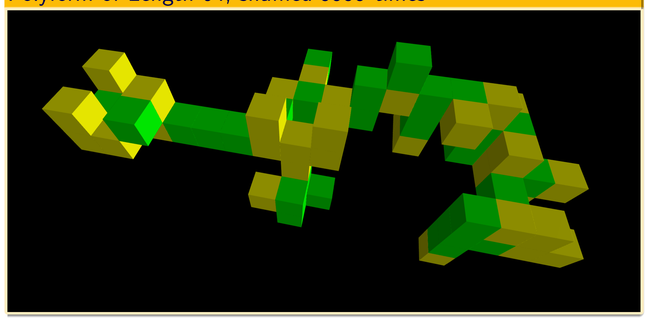
Mason Experimental Geometry Lab

May 6 2022

Introduction to Polyforms

A polyform is a figure constructed by joining together identical polytopes connected by at least one of their faces. Specifically, we are studying random polyforms composed of strongly connected cubes. Every cube is connected by a path of cubes sharing two dimensional square faces. We're interested in the geometric characteristics of polyforms, such as the perimeter, radius, and number of holes of such generated shapes. Polyforms have applications in statistical physics, chemistry, and mathematics.

Polyform of Length 64, shuffled 9999 times



The 2D Case: Polyominos

Redelmeier's Algorithm

Redelmeier's algorithm is a method of enumerating the number of fixed polyominos of size n. A connected graph is created, where every node in the graph is a location for a square of a size n polyomino. Each polyomino is a subgraph within the graph which contains n nodes.

Enumerating Polyominos

1. Create an "empty" polyomino as the parent, with an "untried" set of nodes with only the origin.
2. Remove a random entry in the untried set.
3. Add a node to the graph.
4. Count the new polyomino.
5. If the size is less than n:
   1. Add new neighbor nodes to the "untried".
   2. Recursively call the algorithm, with the new polyomino becoming the parent.
   3. Remove the new neighbor nodes from "untried".
6. Remove the newest node.

Holes and the Euler Characteristic

Euler Characteristic

$\chi = V - E + F = C - H$

Vertices - Edges + Squares = Connected Components - Holes

Pick's Theorem

$A = i + \frac{b}{2} - 1$

Area = Interior Points + $\frac{\text{Boundary Points}}{2}$ - 1

Squares - Interior Points - $\frac{\text{Boundary Points}}{2} + 1$ = Holes

For any given polyomino, it is given the number of connected components of the complement is equal to 1. Given this fact, and an adaptation of Pick's theorem, we can find the number of holes of a given polyomino by counting it's interior and boundary points. We iterate over every vertex of each square. If a given point appears 1, 2, or 3 squares, it is a boundary point. If a given point appears on 4 squares, it is an interior point.

Metropolis Hastings Algorithm

General process for sampling from a target distribution using a transition kernel:

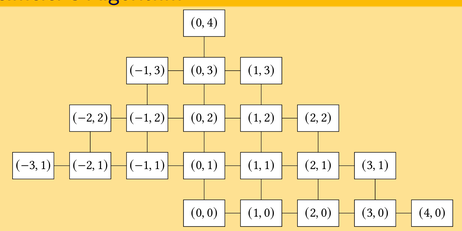
1. Choose an arbitrary point x_t to be the first observation x_0.
2. For t = 1, 2, ... pick a candidate y for the next sample by picking from a selected probability distribution.
3. Compute the acceptance probability, α.
4. Generate a uniform random number $u \in [0, 1]$.
5. If $u \leq \alpha$, then accept the candidate, $x_{t+1} = y$. If $u > \alpha$, then reject the candidate and set, $x_{t+1} = x_t$.

Understanding the MH Algorithm with 3d polyforms:

1. Arbitrarily pick one of the cubes in the polyform.
2. Remove the cube from the polyform.
3. Select a neighbor of the remaining cubes uniformly at random.
4. Place the cube at the positions of one of the neighbors.
5. If the neighbor selected creates a valid connected polyform, keep the cube where it is. If not, put the original cube back as it was in the step 1.

Codebase

Redelmeier's Algorithm



Algorithm 1: Redelmeier's Algorithm for Enumerating Polyominos of Size n

```
Input: graph, untried, n, current
Output: count
while len(untried) > 0 do
    u = untried[0]
    append(current, u)
    remove(untried, u)
    if len(current) = n then
        count = count + 1
    else
        newNeighbors = createSet()
        for v ∈ graph do
            if v ∉ untried, v ∉ current, v ∉ neighbors then
                append(newNeighbors, v)
        newuntried = untried + neighbors
        polycount(graph, newuntried, n, current)
    remove(current, u)
return count
```

Shuffle

Algorithm 2: Metropolis-Hastings Algorithm for Shuffling Polyforms

```
Input: polyform
Output: boolean
connected = set(polyform, 0)
lastLen = len(polyform)
while len(polyform) > 0 do
    for cube ∈ polyform do
        neighbors = getPerimeter(cube) for neighbor ∈ neighbors do
            if neighbor ∈ connected then
                remove(polyform, cube)
                append(connected, cube)
    if lastLen - len(polyform) == 0 then
        return False
    lastLen = len(polyform)
return True
Input: polyform
Output: shuffled polyform
x = pop(polyform, random[0, polyformSize])
x_{t+1} = random[getPerimeter(candidate)]
append(polyform, x_{t+1})
if isValidPolyform(polyomino) then
    remove(polyform, x_{t+1})
    append(polyform, x)
return polyform
```

Optimizations

After a shuffle, we must check that the polyomino is still strongly connected. This operation takes notably longer than the shuffle itself.

- For 2D polyominoes, the estimated total shuffle time is between n^2 and n^3, where n is the number of squares in the polyomino.
- We use a HashSet for near-instant lookups for pieces known to be strongly connected.
  - We don't need to check that the whole polyomino is strongly connected after a single shuffle operation; only pieces that were disconnected have to be checked for strong-connectedness, allowing us to use an early stopping condition.
- Further algorithmic speedups are possible by applying the early stopping condition more aggressively and by porting this to a lower-level programming language and using multiple threads.

Future Work

Much of the work in this semester has been to create and optimize the code base required to generate and study random polyominos and polyforms. In the Fall, we intend to use the existing code:

- Sample from other distributions other than the uniform distribution
- Further study the homology of our random 3D polyforms
- Look for new and more efficient ways of shuffling polyforms, whilst still maintaining the desired distributions

Acknowledgements

We'd like to thank our project mentors Dr. Schweinhart and Dr. Roldan. We would also like to thank Aleyah Dawkins, our graduate research assistant, and MEGL for supporting our work. We're excited to continue in the fall of next semester.

References:

- Kahle, Matthew, and Erika Roldan. "Polyominoes with Maximally Many Holes." Geombinatorics Quarterly, XXIX, no. 1, July 2019, pp. 5–15.
- Aleksandrowicz, Gadi, and Gill Barequet. "Redelmeier's Algorithm for Counting Lattice Animals." Proceedings of the 27th Annual ACM Symposium on Computational Geometry - SoCG '11, June 2011, pp. 283–284., https://doi.org/10.1145/1998196.1998238.