Sierpiński-Curves

Joint Advanced Student School 2007

Martin Dummer

Statement of the Problem

What is the best way to store a triangle mesh efficiently in memory?

The following points are desired :

- Easy to compute
- Requires little memory
- Adaptive refinement is possible
- Finding the neighbor of a node is easy

Overview

- **Storage Models**
- **Refinement**
  - Basics
  - Bisectioning
  - General purpose objects
- **Storage in Trees**
- **Introduction to Curves**
- **Stacks**
- **Neighbors**
- **Unknown edges**
- **Example**
- **Conclusion**

Storage - Models

Surface-based

Wireframe model

(**CAD**)

==> high effort for complex objects

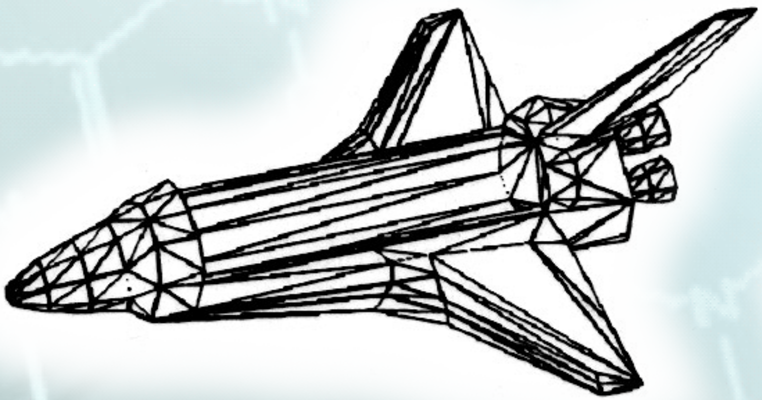
Volume-based

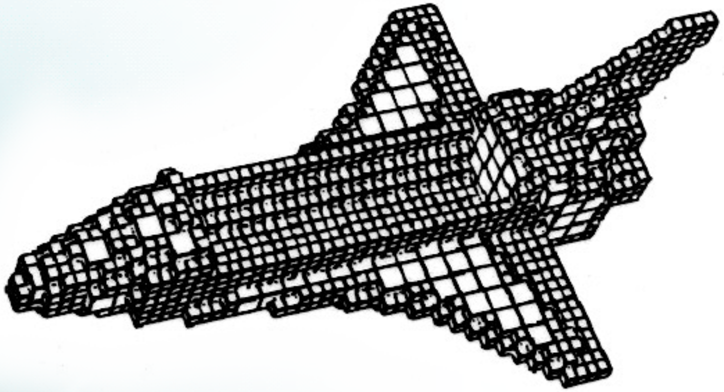
Segmentation

(scientific computing)

==> always complexity $O(n^3)$

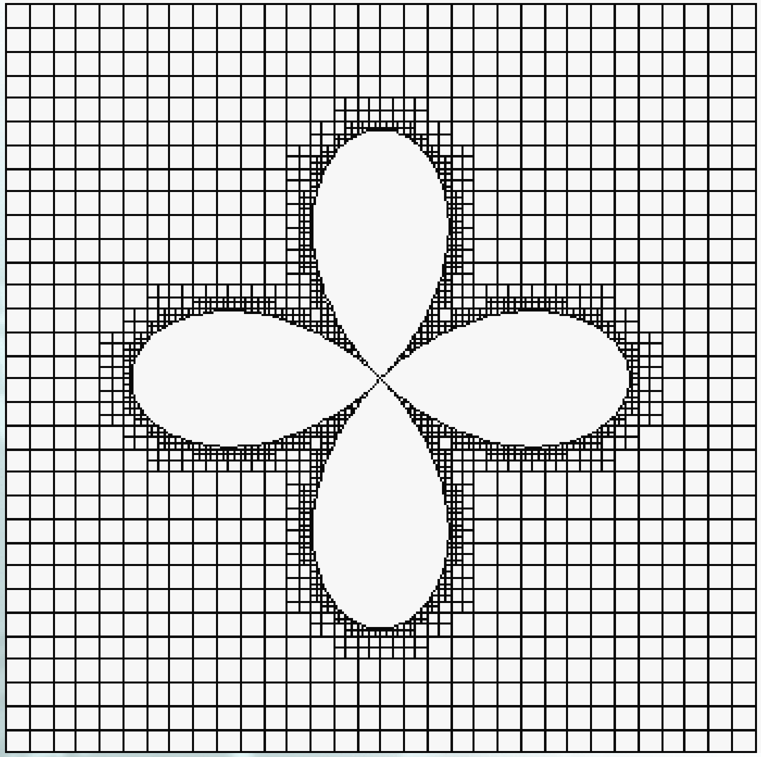
Storage - Models





==> Neighborhood relations are important

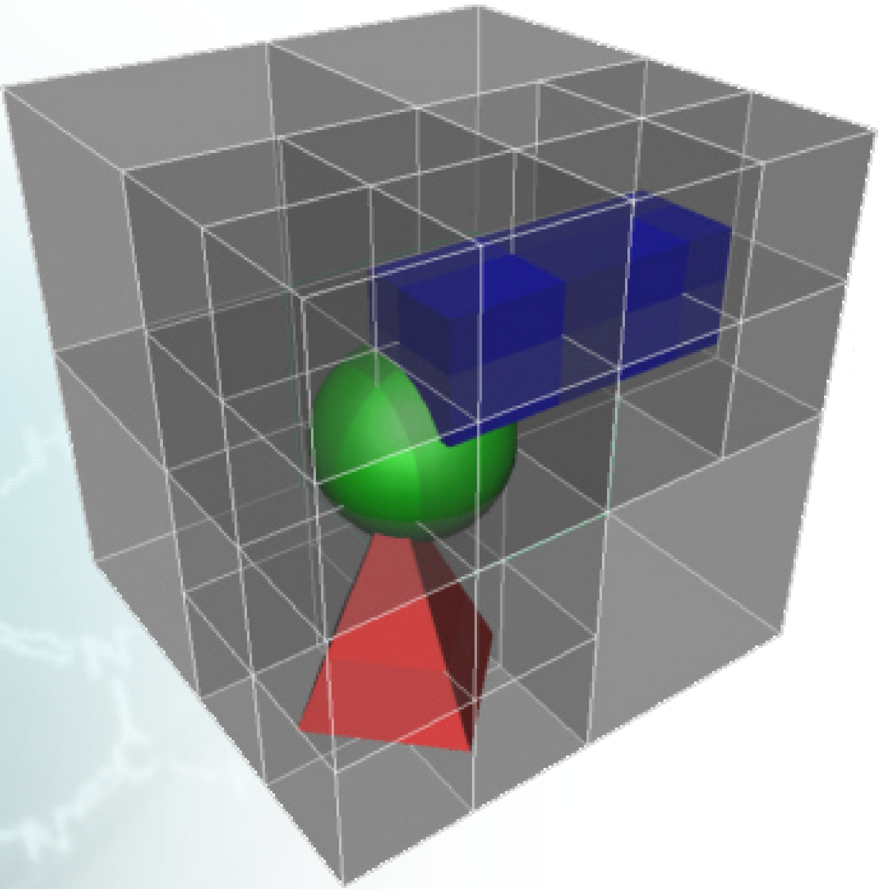
Adaptive Grids



The grid requires high resolution only at certain points

k^d-Spacetrees

Refine only where more information is stored (**borderline**)



2^2 Quadtree

2^3 Octree

==> Tree structure

Refinement basics

How to find out where refinement is necessary?

- Evaluate the discretization error
- Evaluate possible improvement (change in the result)

==> There is no optimal refinement

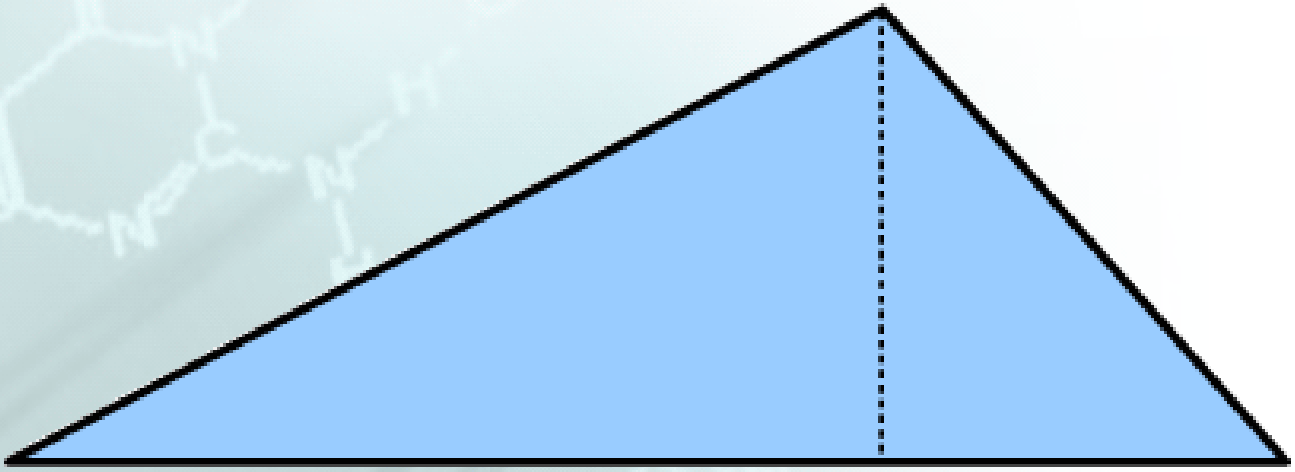
Adaptive refinement

To achieve the most generic algorithm
the most basic 2D structure is used

This is called **Bisectioning**

Adaptive refinement

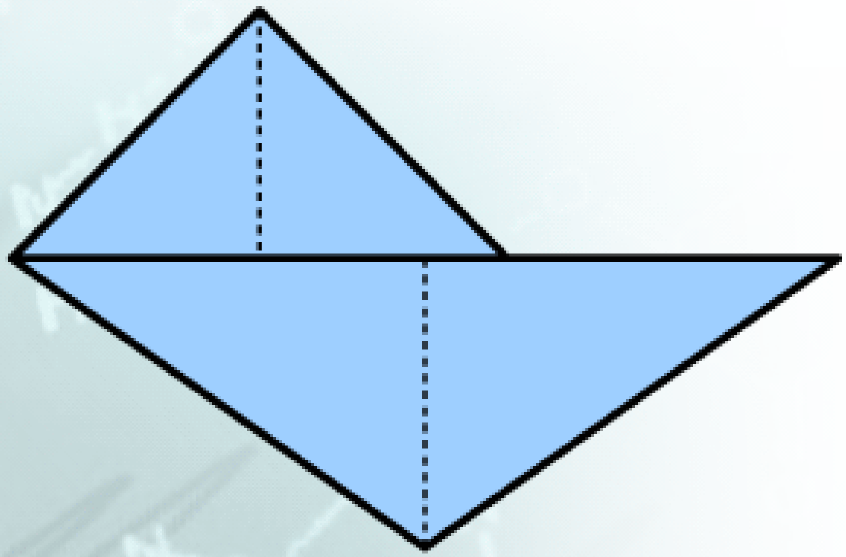
Bisecting which vertex gives the best results?



First guess usually is the one opposite to the longest edge

Adaptive refinement

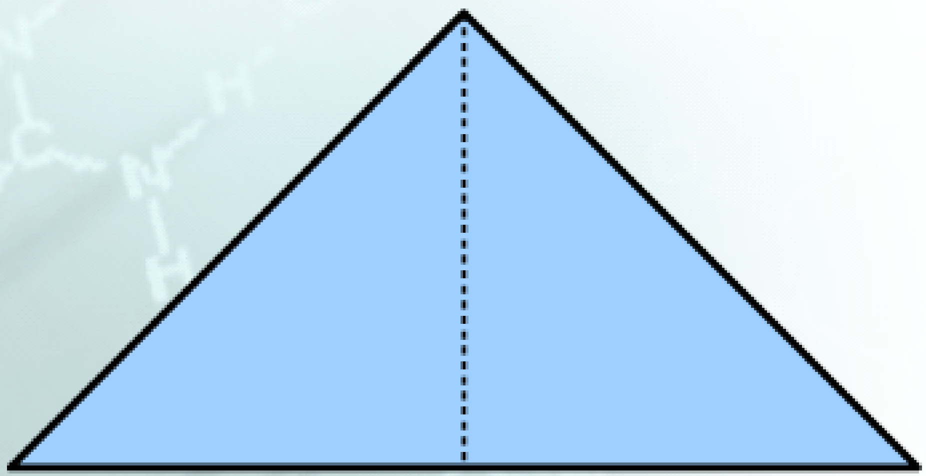
Bisecting which vertex gives the best results?



==> This leads to "hanging nodes" which are difficult to handle

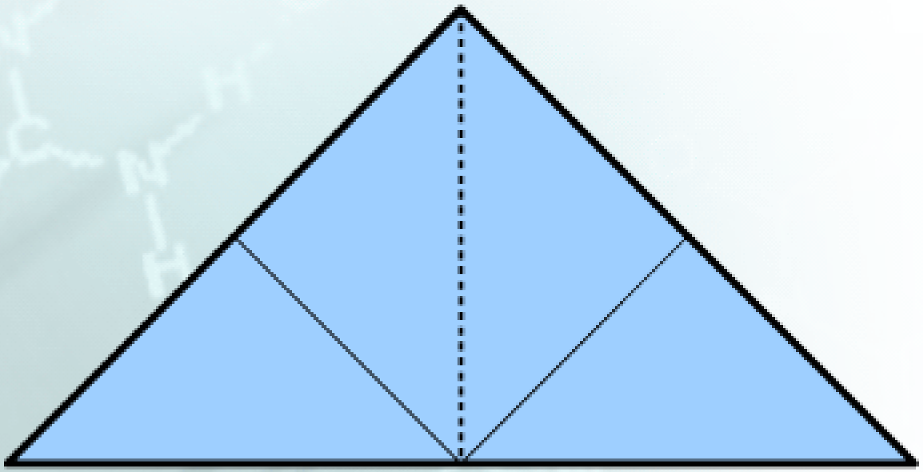
Adaptive refinement

<u>Alternative</u> : Always divide 2 triangles at a time



Adaptive refinement

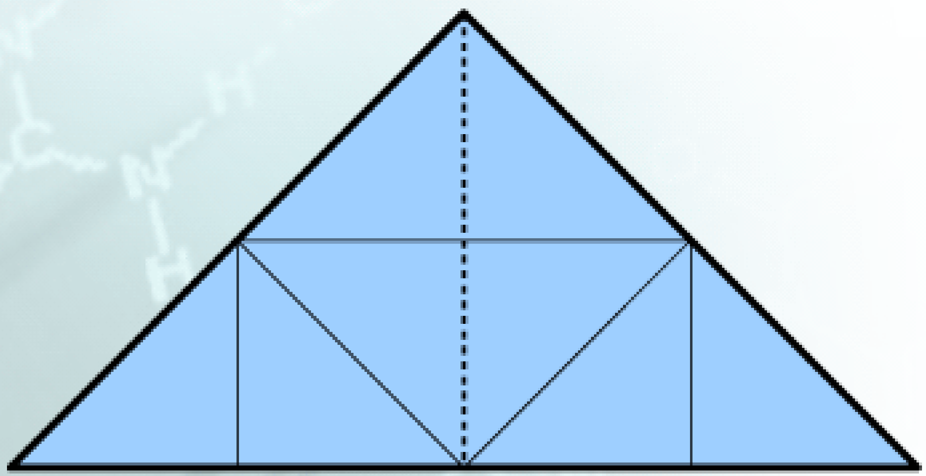
<u>Alternative</u> : Always divide 2 triangles at a time



Use the "newest" vertex to divide the triangle again

Adaptive refinement

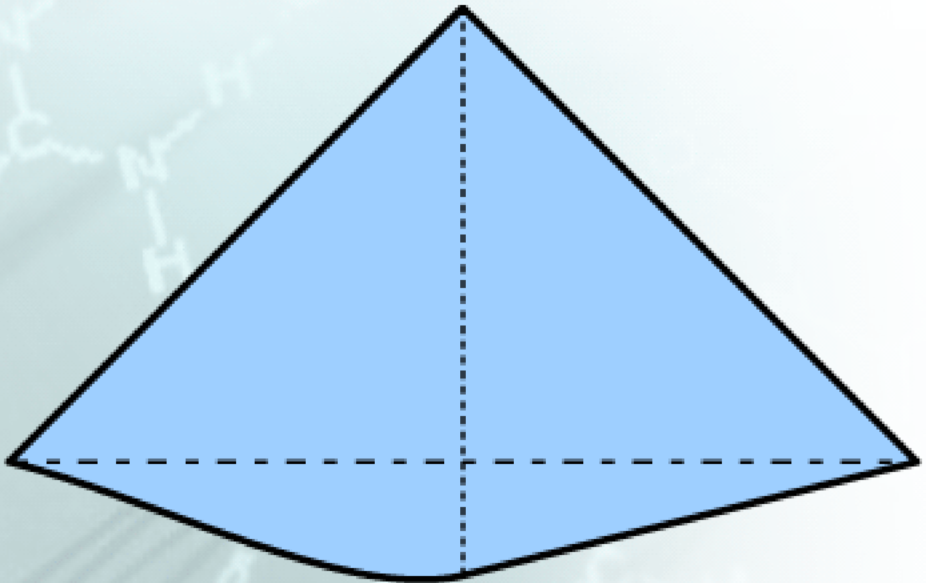
<u>Alternative</u> : Always divide 2 triangles at a time



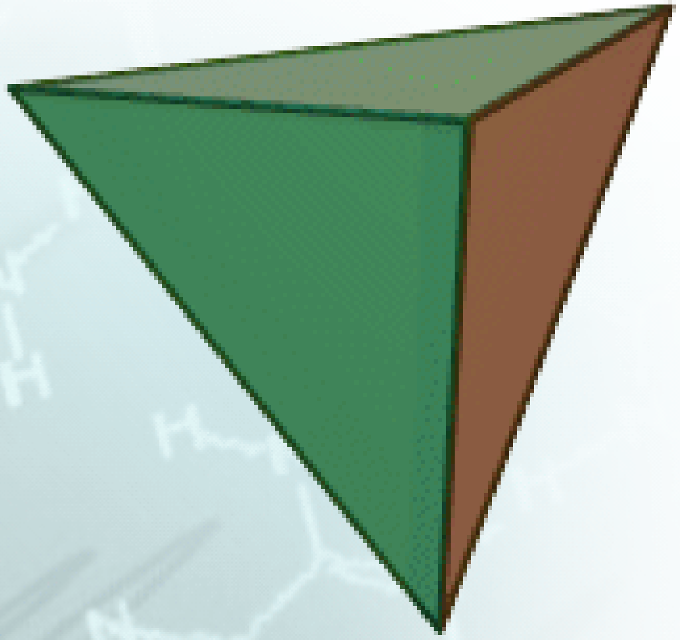
==> No hanging nodes for this bisection rule

Arbitrary Borders

Evaluate a function instead of dividing the edge



Bisection in 3D



Use a tetrahedron instead of a triangle

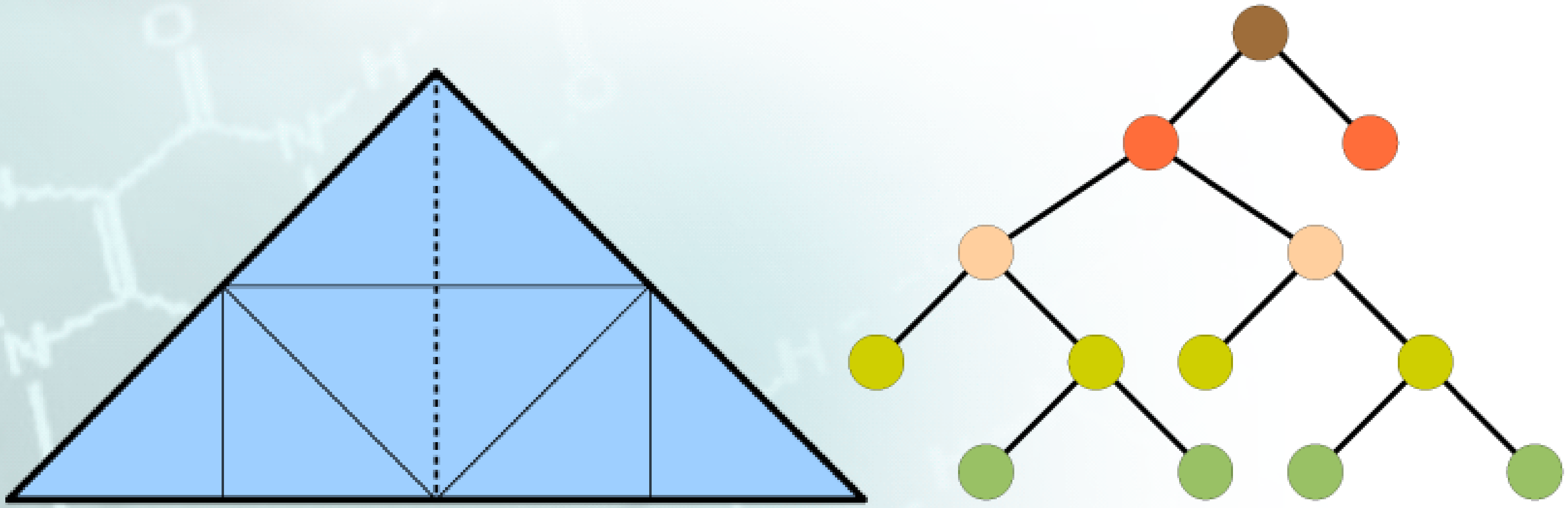
(Image taken from **wikipedia.org**)

Review

What do we have so far ?

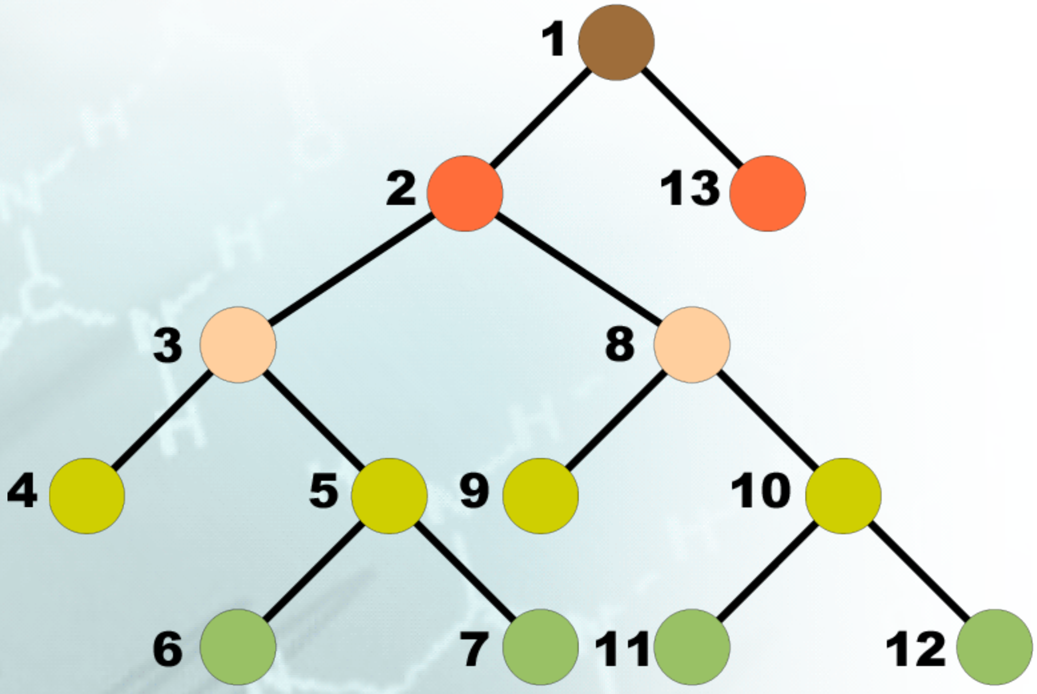
- Volume based model
- 2D and 3D
- Arbitrary shape
- Adaptive refinement

Storage



Represent the sub-triangles in a binary tree

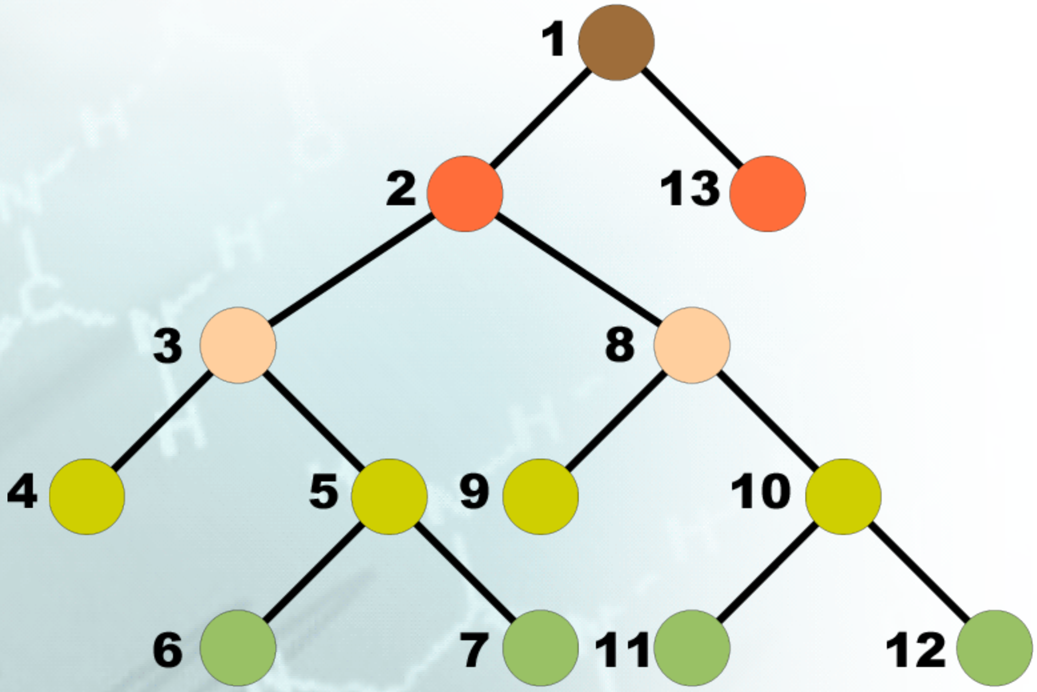
Linearization



Apply depth-first search (DFS)

Store only one refinement bit for each node

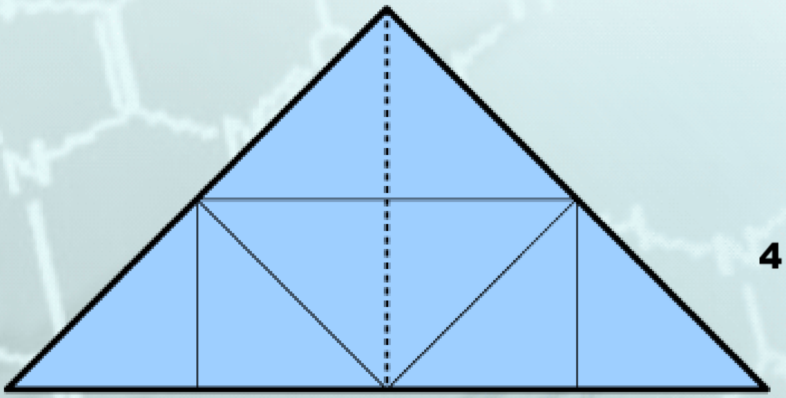
Linearization

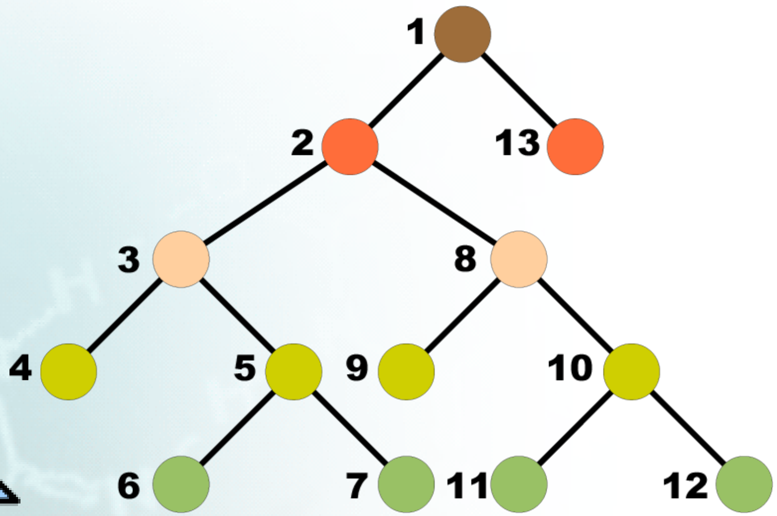


1 1 1 0 1 0 0 1 0 1 0 0 0

Neighborhood issues

How do we find the corresponding neighbor?

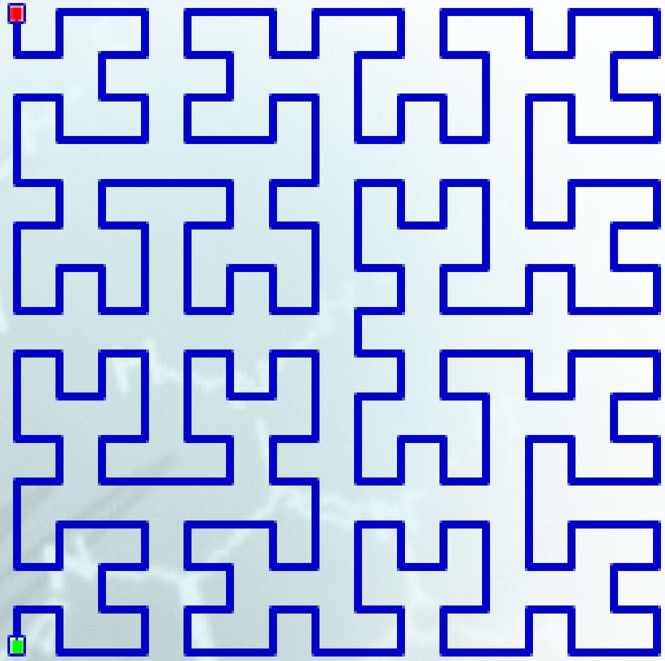




1 1 1 0 1 0 0 1 0 1 0 0 0

Space-filling curves

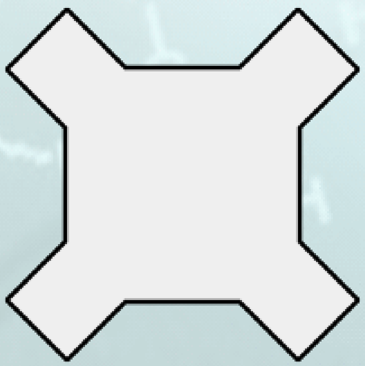
Hilbert curve



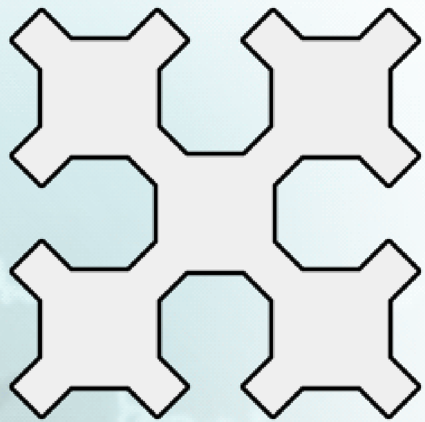
Mapping of a 1D curve into a 2D area

Sierpiński-Curves

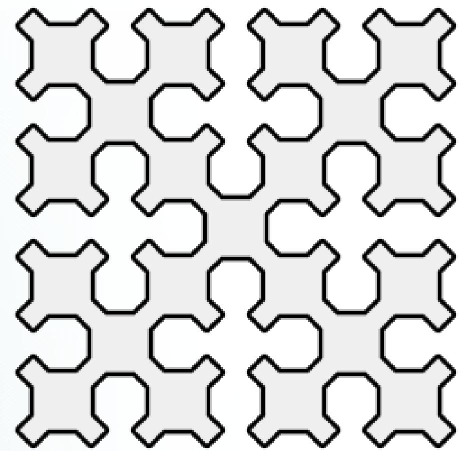
Fractal geometry object similar to Hilbert- and Peano-curves



Order 1



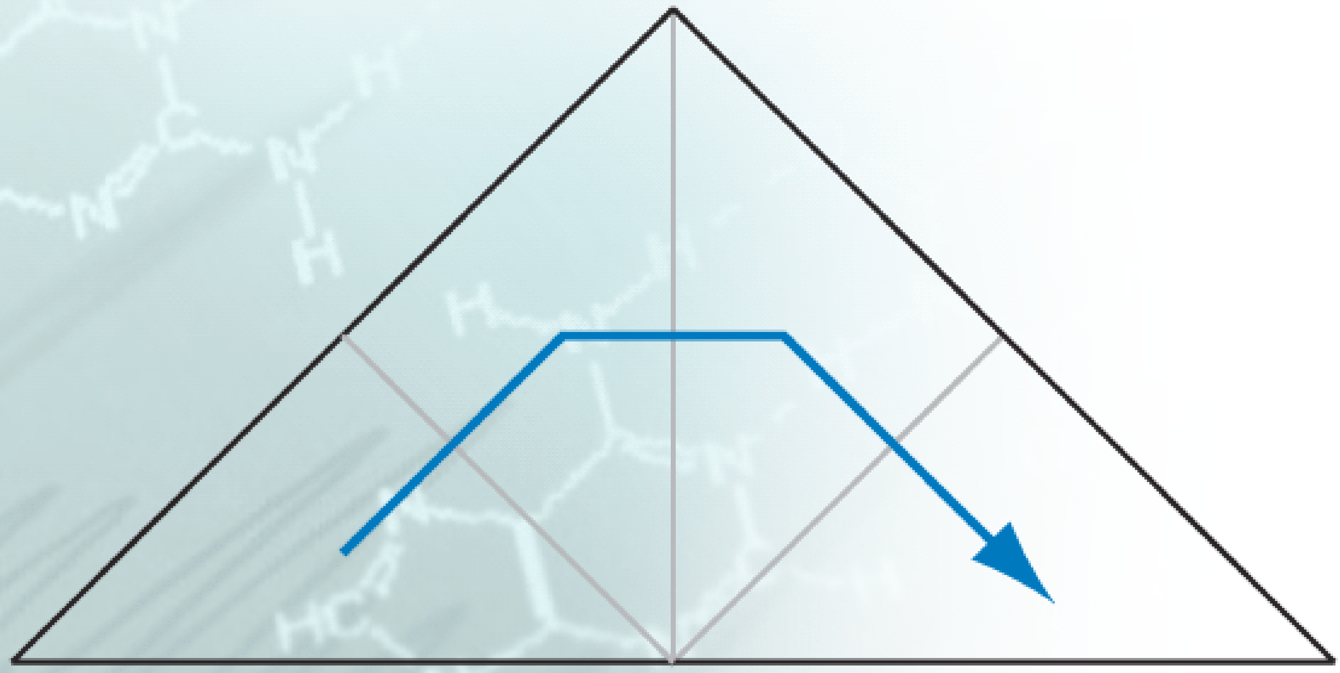
Order 2



Order 3

Sierpiński-Curves in Grids

Iterate through grid cells according to DFS

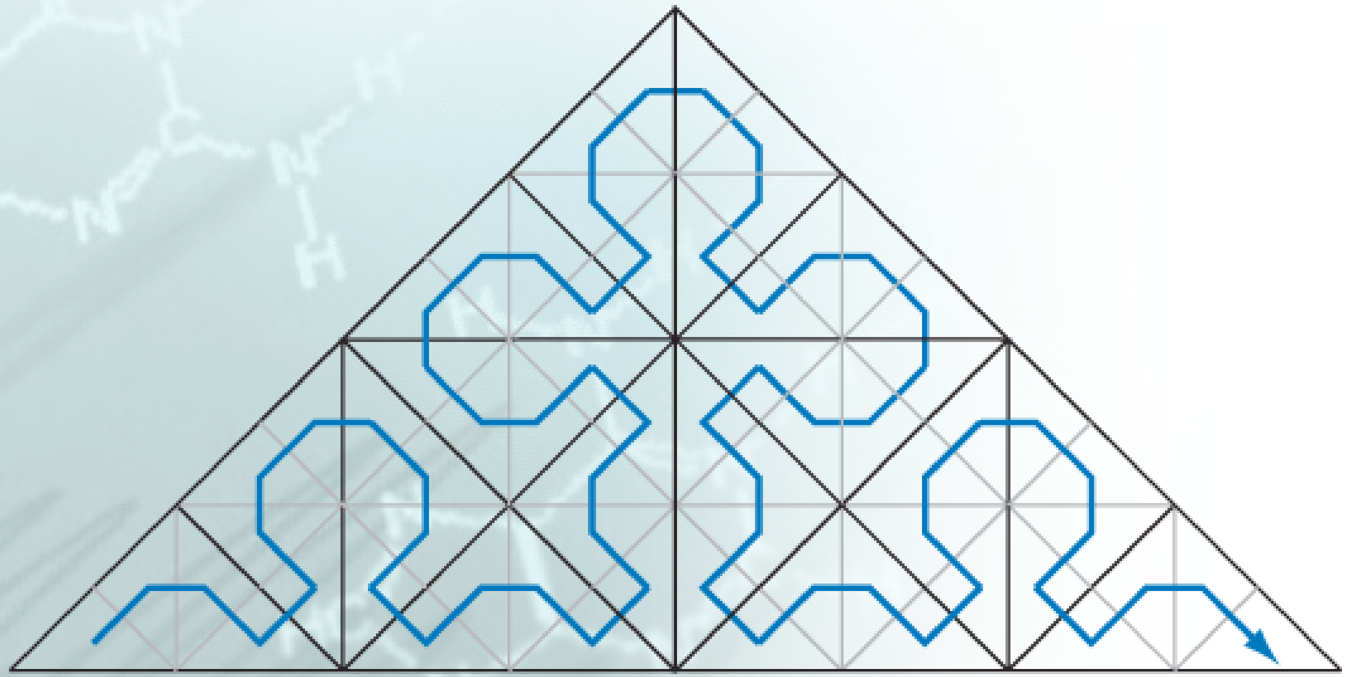


Sierpiński-Curves in Grids

Iterate through grid cells according to DFS

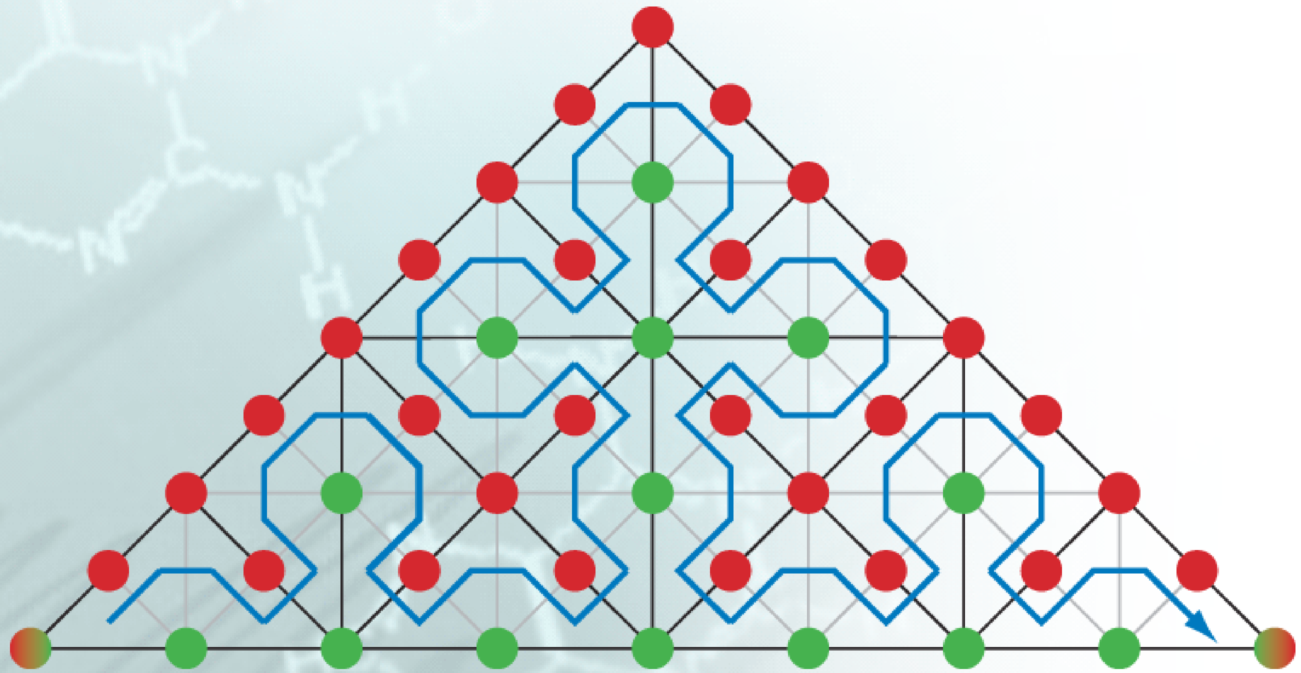
Sierpiński-Curves in Grids

Iterate through grid cells according to DFS



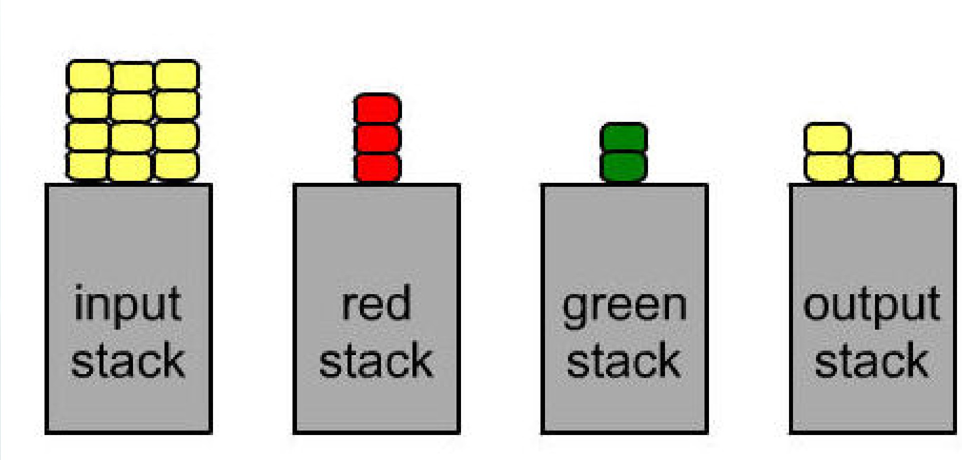
Neighborhood problem

Sierpiński iteration linearizes a triangle



Divide cells into left and right side

Stacks

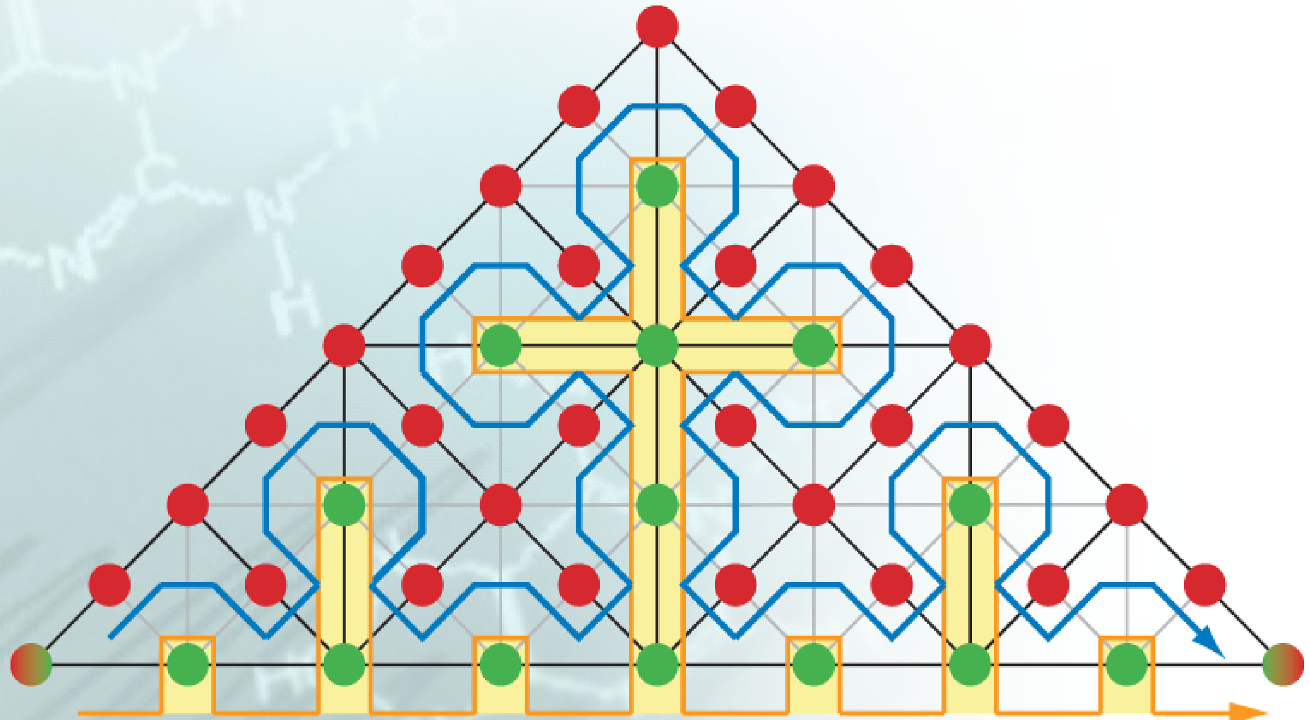


Stack operations

(push) adds an element on top of the stack
(pop) removes an element from top of the stack

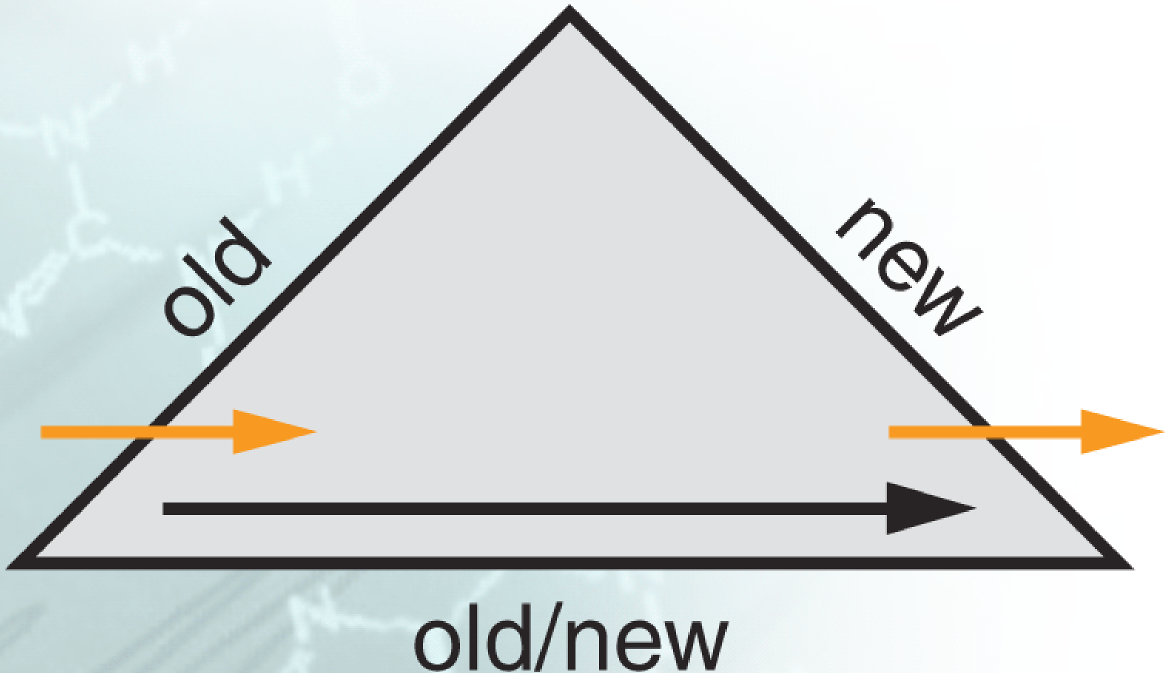
Neighborhood problem

Sierpiński iteration linearizes a triangle



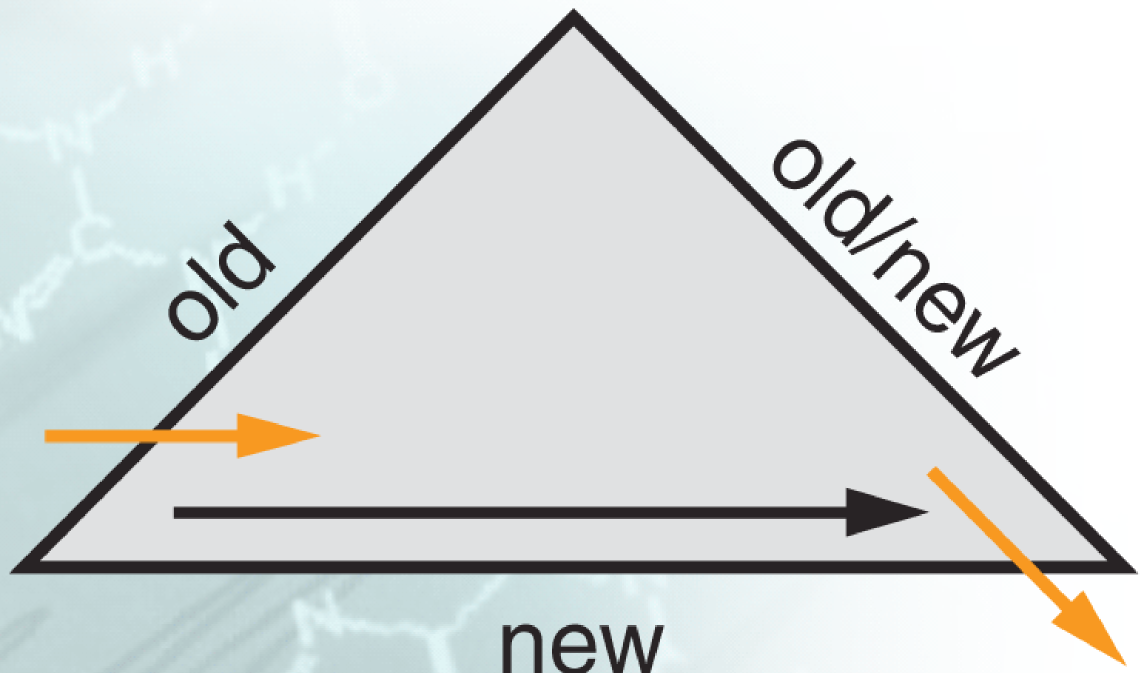
Divide cells into left and right side

Neighborhood problem



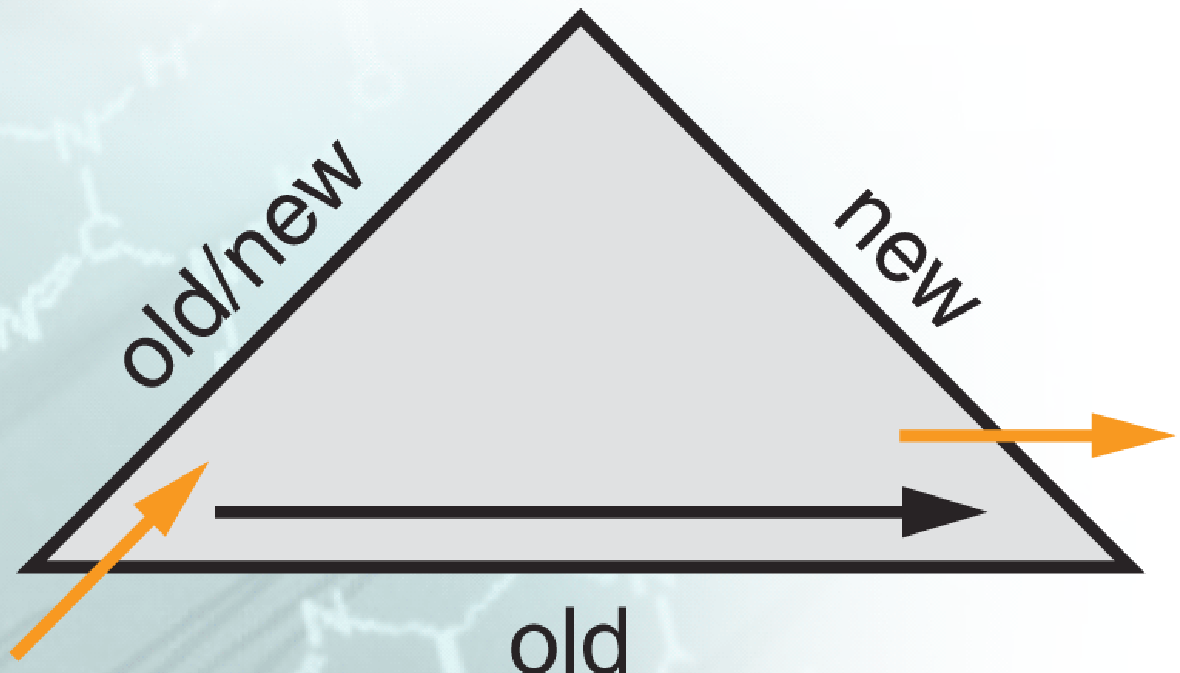
Possible configurations for triangle traversal

Neighborhood problem



Possible configurations for triangle traversal

Neighborhood problem



Possible configurations for triangle traversal

Unknown edges

Use input or temporary stack?

<u>No adjacent cells have been visited before</u>

(yes) Read from the input stack
(no) Read from a temporary stack

Unknown edges

Use output or temporary stack?

<u>All adjacent cells have been visited before</u>

(yes) Write on the output stack
(no) Write on a temporary stack

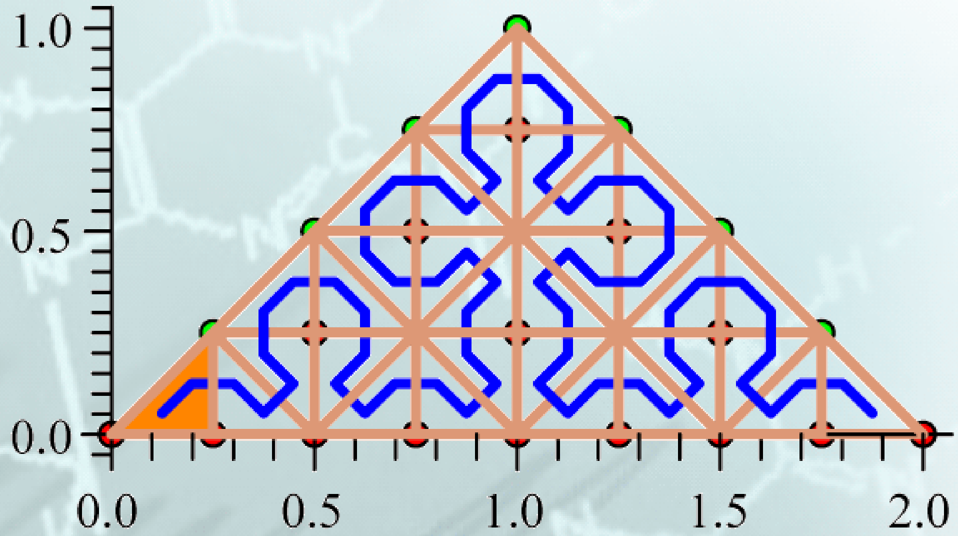
Unknown edges

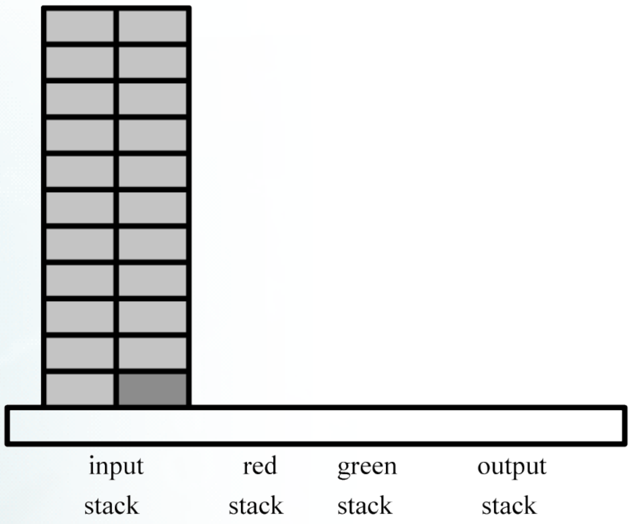
Use output or temporary stack?

<u>Alternative:</u>

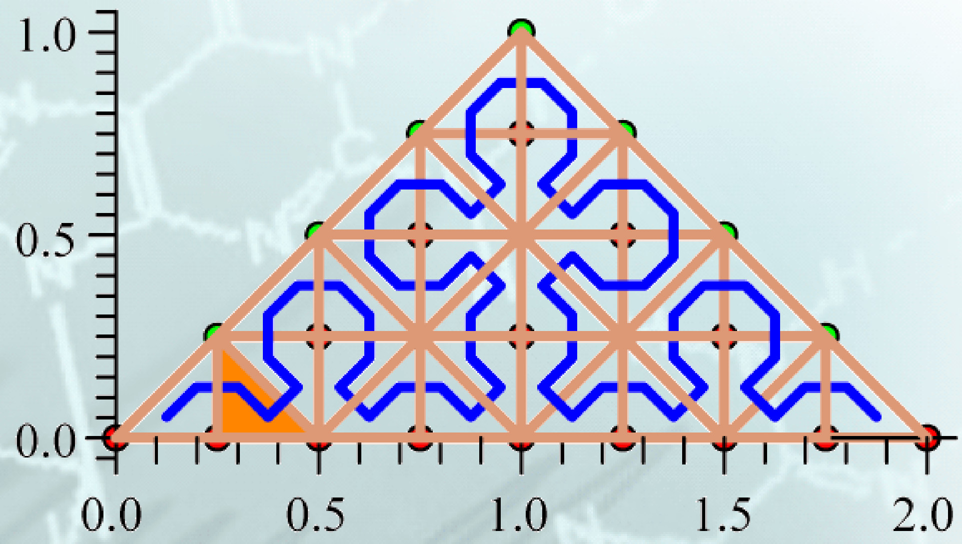
Count number of write accesses and compare with number of adjacent cells

Example

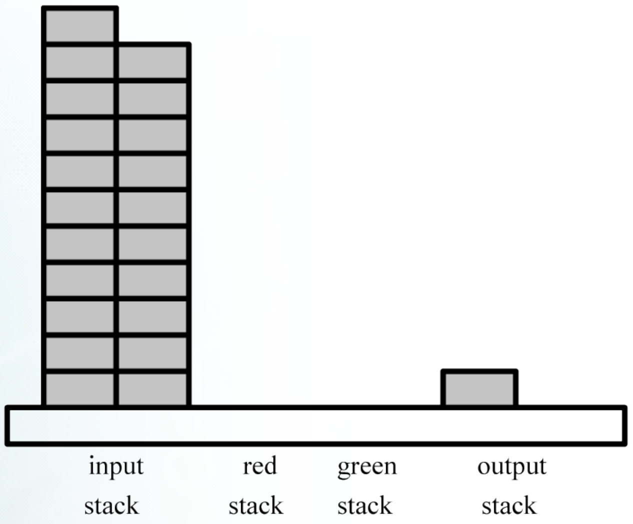

1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0


input
stack
red
stack
green
stack
output
stack

Example

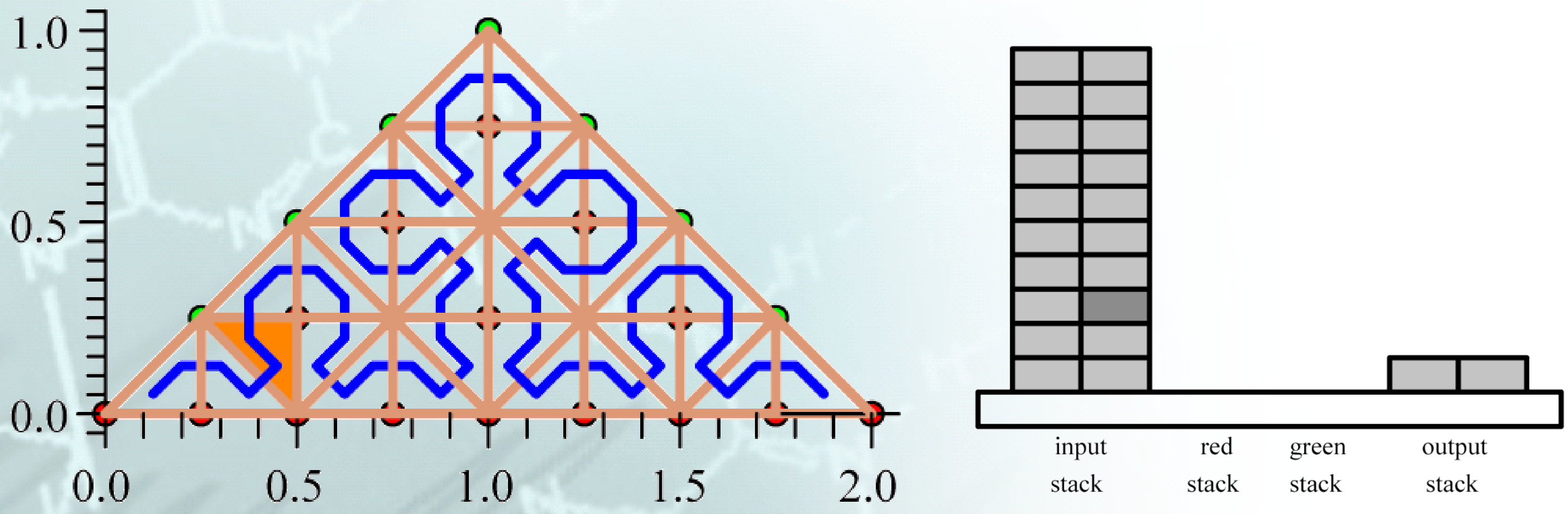


1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0

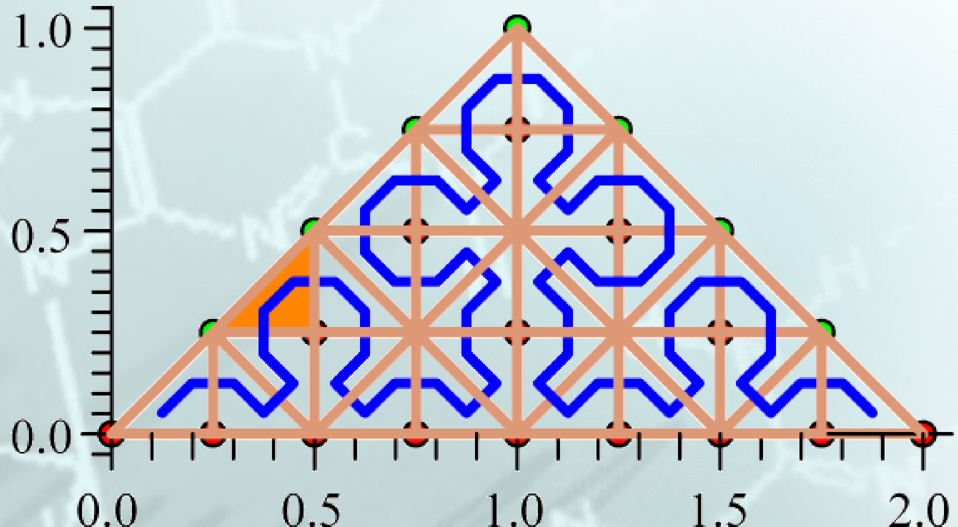


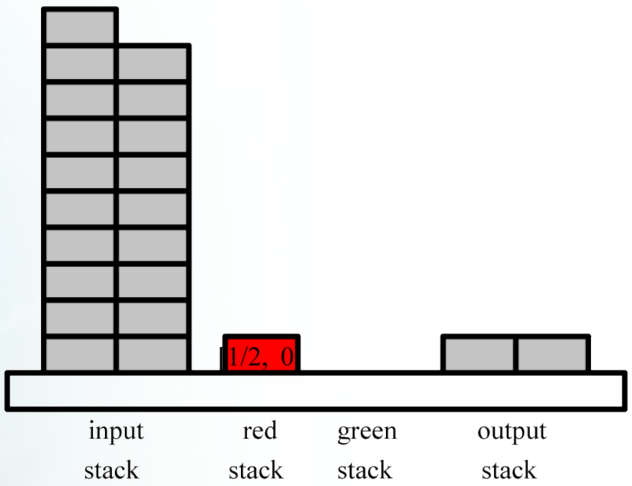
input stack
red stack
green stack
output stack

Example

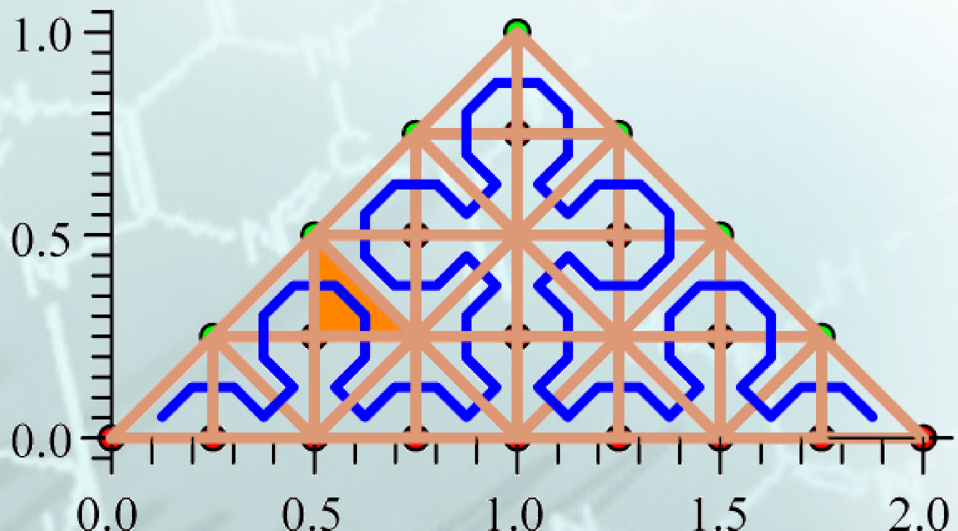

1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
input
stack
red
stack
green
stack
output
stack

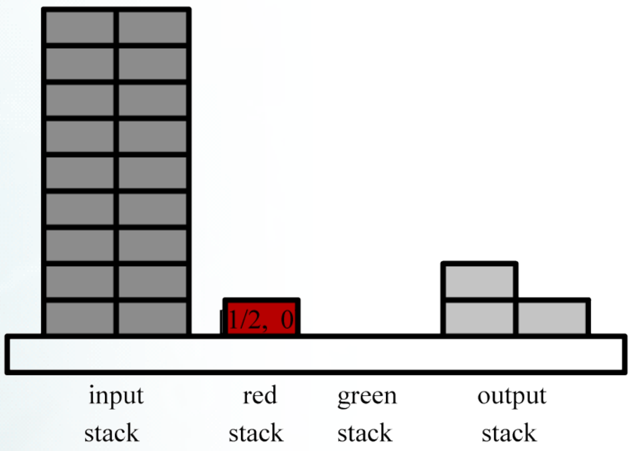
Example


1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0

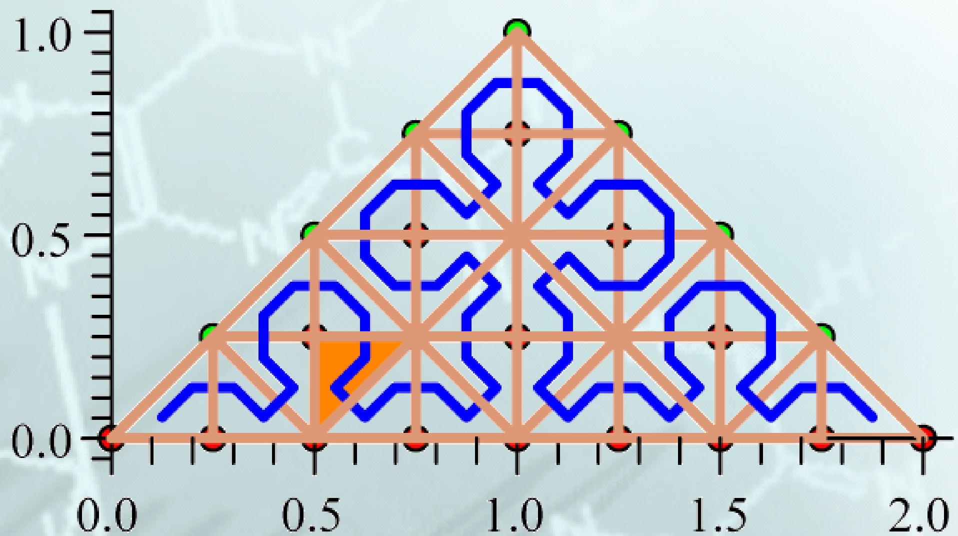

1/2, 0
input
stack
red
stack
green
stack
output
stack

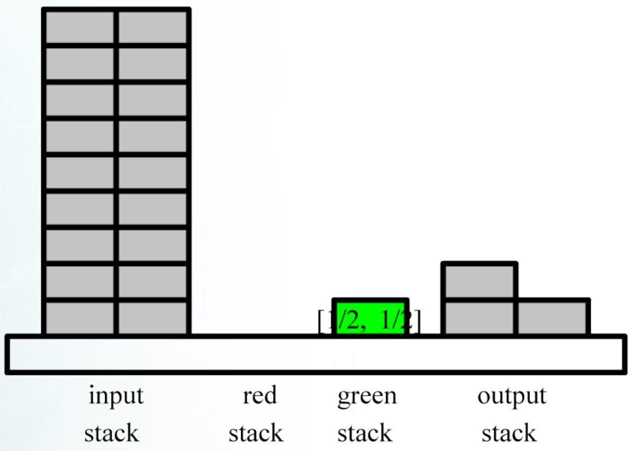
Example


1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0


1/2, 0
input stack
red stack
green stack
output stack

Example


1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0


[1/2, 1/2]
input
stack
red
stack
green
stack
output
stack

Conclusion

This algorithm combines the advantages of DFS and the stack system based on Sierpiński-Curves

- Easy to compute
- Requires little memory
- Adaptive refinement is possible
- Finding the neighbor of a node is easy

Thank you for your attention

Questions ?