Computational topology: Lecture 17

Alberto Paoluzzi

May 16, 2019

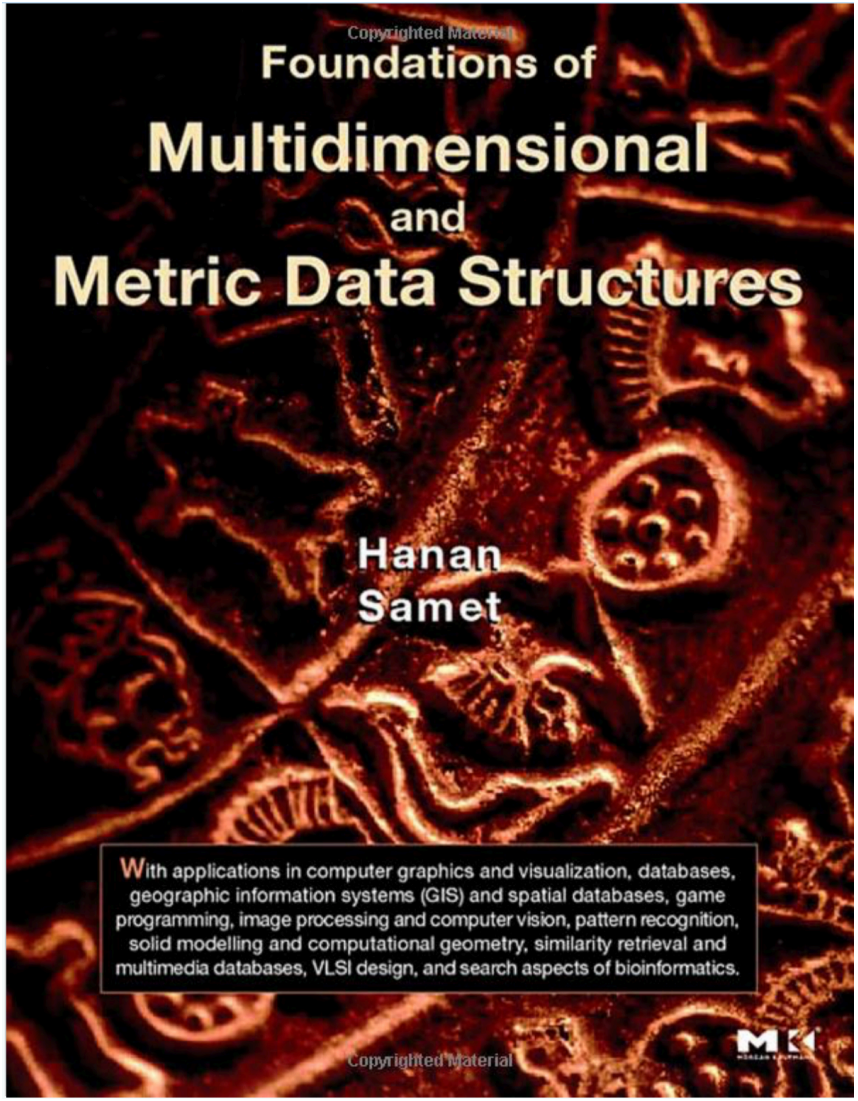
1. Domain Trees
2. Constrained Delaunay triangulation (CDT)
3. Patching the Triangle.jl library
4. Implementing Alpha-shapes in 2D

Domain Trees

Spatial Hierarchical Domain Trees

Foundations of Multidimensional and Metric Data Structures provides a thorough treatment of multidimensional point data, object and image-based representations, intervals and small rectangles, and high-dimensional datasets

Hanan Samet is the inventor of such trees (quadtrees and octrees), which are the foundation of Geographical databases



2^n-trees

2^n-trees are ordered trees characterized by the property that each non-leaf node has exactly 2^n son nodes, respectively denoted as first, second, etc., and as 2^n-th son

When $n = 2$ and $n = 3$ such trees are called quadtrees and octrees, and are used to represent hierarchical decompositions of the 2D or 3D space, respectively

Octrees

- When the model is embedded in 3D, i.e. when n = 3, the 2n-tree representation is called an octree.
- In this case the embedding space is subdivided by three pair-wise orthogonal planes, so giving 23 cells (either white, black or gray) at each step (see Figure 13.6)

Quadtrees

When the model is embedded in 2D, i.e.~when $n = 2$, the 2^n-tree representation is called quadtree.

Some useful properties of quadtrees follow:

1. a quadtree is a quaternary tree (i.e. each non-leaf node has exactly 4 sons);
2. the leafs are either white or black nodes (i.e. either empty or full);
3. the non-leafs are gray nodes (i.e. neither empty nor full);
4. the maximal depth of the quadtree is related to its resolution.

- The number of arcs on the path from the root to a node is called distance of the node from the root.
- Depth of a tree is the maximal distance of its nodes from the root.
- The resolution of the quadtree with squared bounding box of size L and depth m is clearly equal to

$$r = L/2^m$$

Quadtree encoding

hierarchical decompositive representation using a quadtree, and its actual encoding as a labeled tree, where black, white and gray nodes represent full and empty cells, and cells which are neither full nor empty.

The sons of a gray node are clockwise ordered, according to the scheme shown in the quadruple of small boxes in the middle top of Figure~1

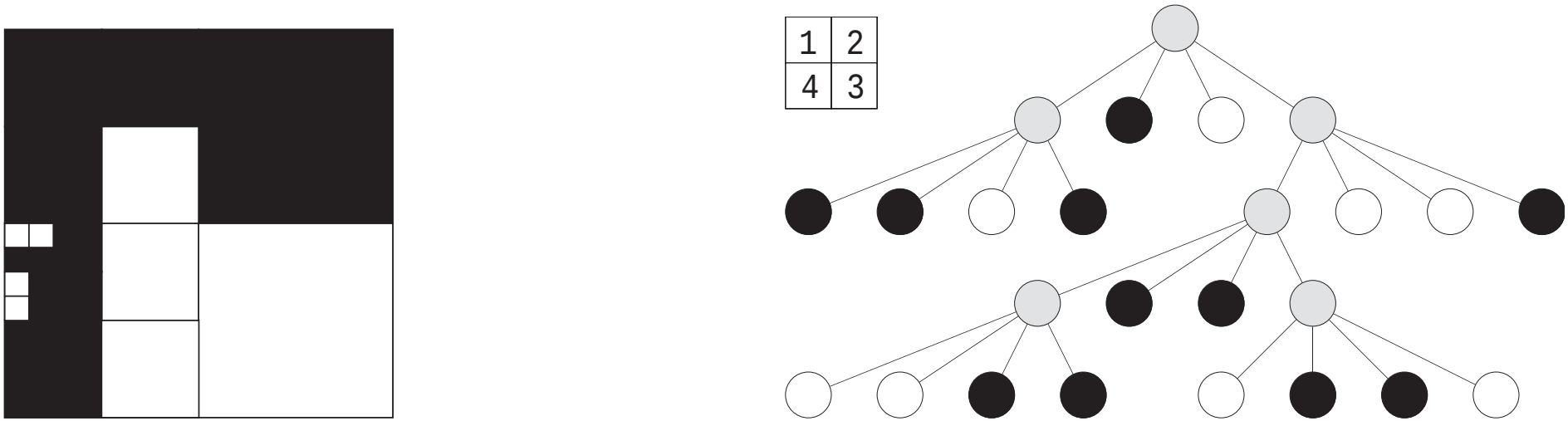


Figure 1: Quadtree encoding scheme: (a) 2D object (b) full cells (black), empty cells (white) and decomposed cells (gray)

Bin-trees

The so-called *bin*-trees are binary trees representing solids in $\mathbb{E}^n$

- In this case tree nodes contain hyperplanes equations of the kind $x_{(d_k \ mod \ n)} = c_k$, where d_k is the (integer) distance of the node k from the root
- Hence, in 2D, *bin*-tree nodes at increasing distance from the root alternatively contain equations such as $x = a_i$ and $y = b_i$
- In 3D, node equations will contain either $x = a_i$ or $y = b_i$ or $z = c_i$, alternatively
- As in the case of quadtrees and octrees, leaf nodes are labeled either black or white, whereas non-leaf nodes can be considered as gray.

Bintrees

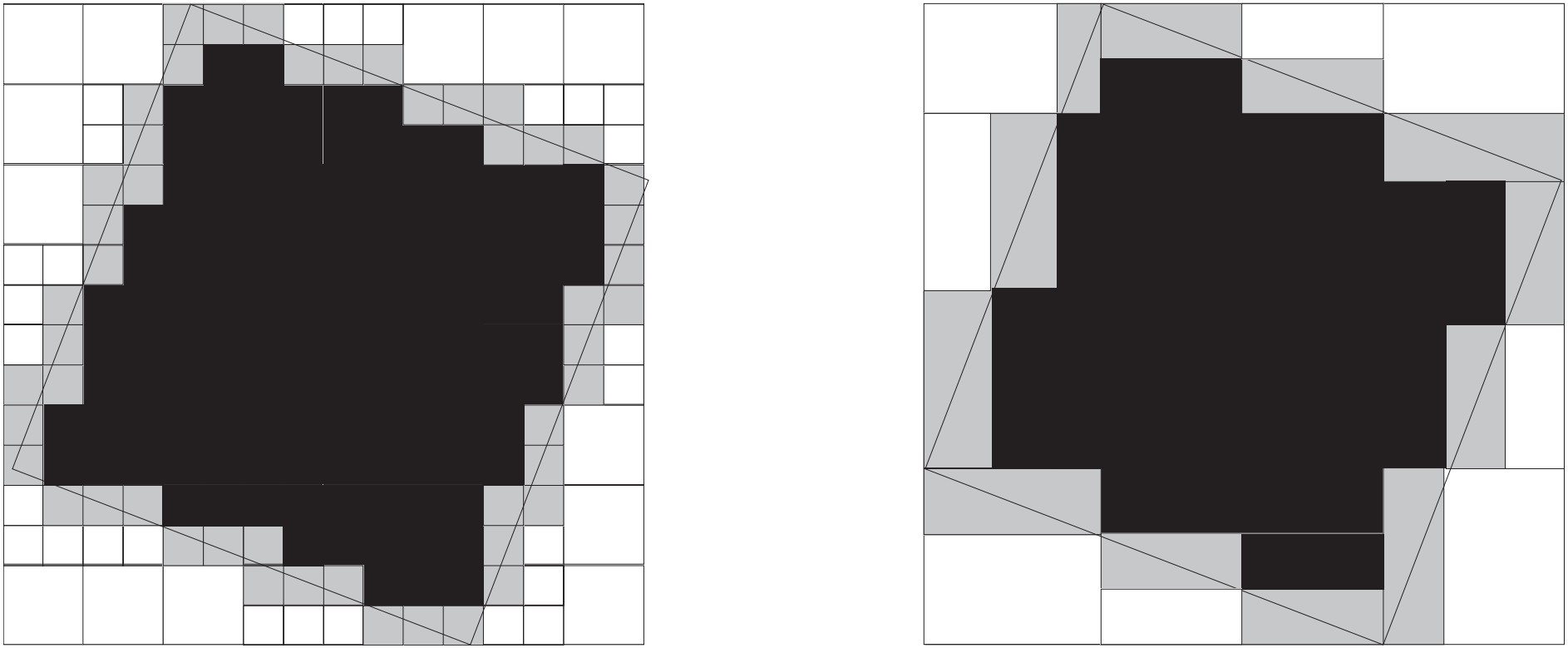


Figure 2: Quadtree vs bintree

Octrees

Octrees are the three-dimensional analog of quadtrees

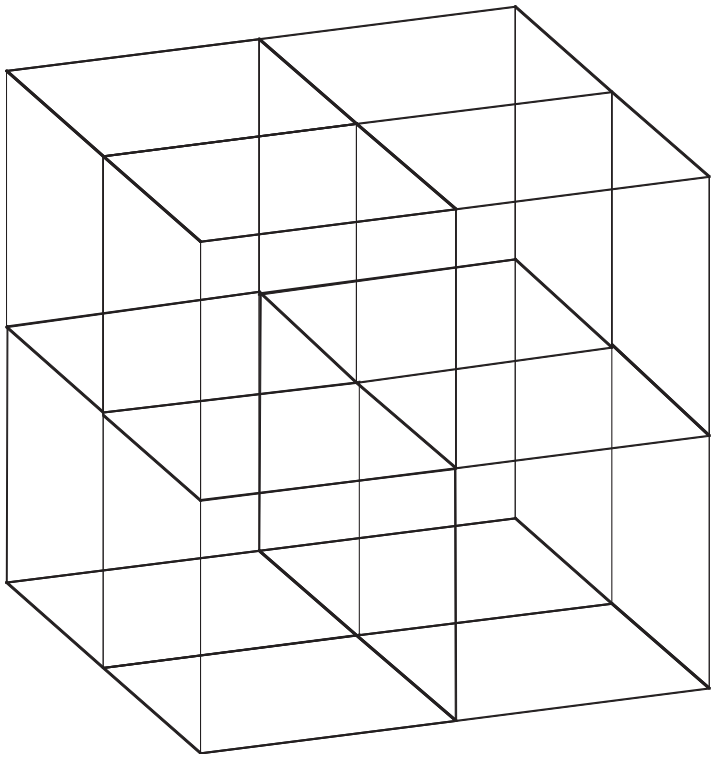


Figure 3: Octree: partition of a 3D cell into 8 sub-cells generated by three orthogonal planes

An octree is a tree data structure in which each internal node has exactly eight children. Octrees are most often used to partition a three-dimensional space by recursively subdividing it into eight octants.

Constrained Delaunay triangulation (CDT)

Delaunay-like triangulation that respects constraints 1/3

In 2D, there are two alternatives for creating a Delaunay-like triangulation that respects constraints.

Input: planar straight line graph (PSLG) -> `(V,EV)`

1. conforming Delaunay triangulation (adding Steiner points)
2. constrained Delaunay triangulation (CDT)

Delaunay-like triangulation that respects constraints 2/3

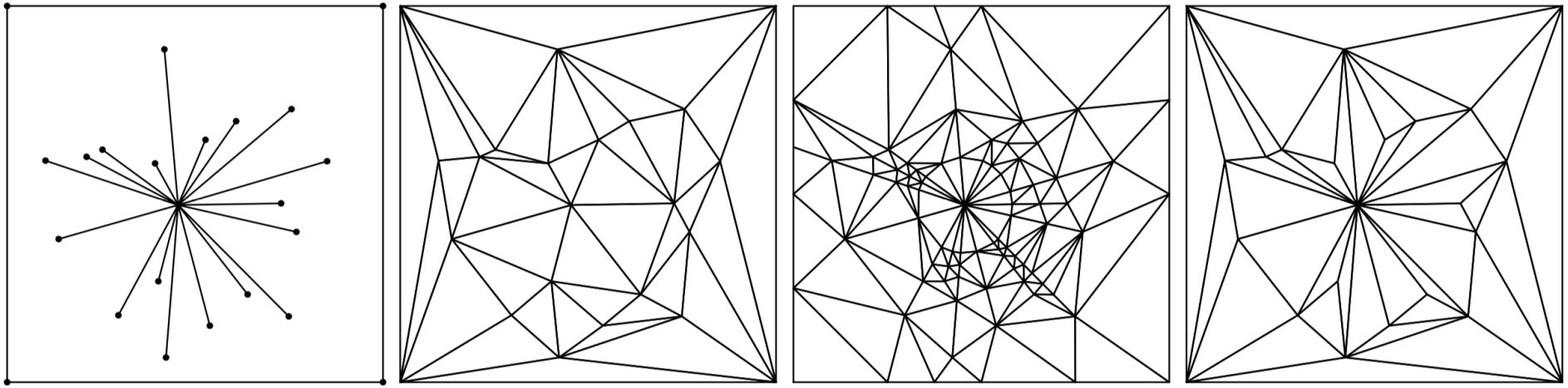


Figure 4: The Delaunay triangulation (second from left) of the vertices of a PSLG (far left) might not respect the segments of the PSLG. These segments can be incorporated by adding vertices to obtain a conforming Delaunay triangulation (second from right), or by forgoing Delaunay triangles in favor of constrained Delaunay triangles (far right)

1

[1] Shewchuk, Jonathan R. (2008), Discrete & Computational Geometry

Delaunay-like triangulation that respects constraints 3/3

A PLC is a finite set of facades in an ambient space Ed. A facade is a polytope (roughly speaking) of any dimension from zero to d, possibly with holes and lower-dimensional facades inside it.[2]

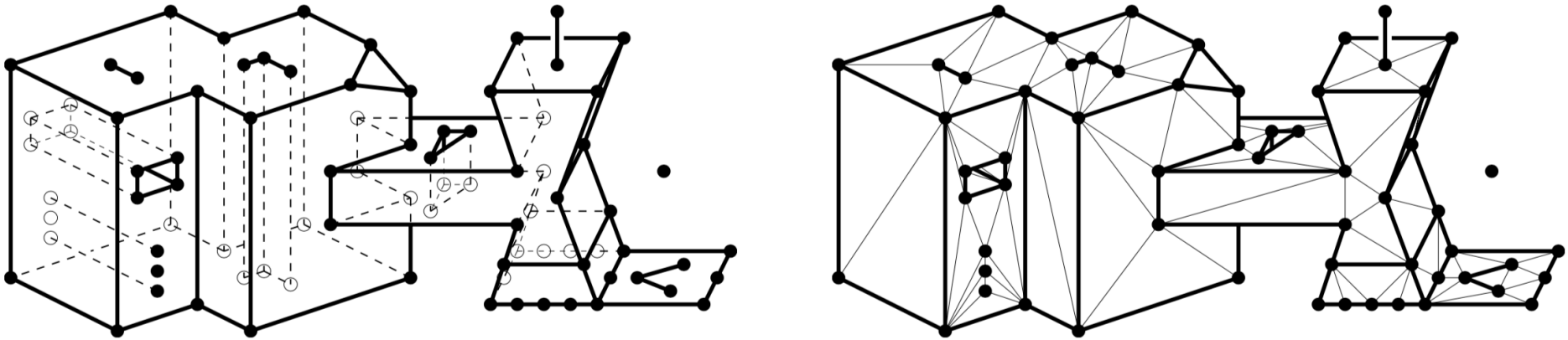


Figure 5: Each facade of a PLC (left) may have holes, slits, and interior vertices, which are used to constrain a triangulation or to support intersections with other facades. The illustration at right is the constrained Delaunay triangulation of the PLC at left. It is a PLC, too

[2] Shewchuk, Jonathan R. (2008), Discrete & Computational Geometry

Definition

In computational geometry, a Constrained Delaunay Triangulation (CDT) is a generalization of the Delaunay triangulation that forces certain required segments into the triangulation

- Because a Delaunay triangulation is *almost always unique**, often a constrained Delaunay triangulation contains edges that do not satisfy the Delaunay condition.
- Thus a CDT often is not a Delaunay triangulation.

Chew, L. Paul (1987) "Constrained Delaunay Triangulations". Proceedings of the Third Annual Symposium on Computational Geometry

Shewchuk, Jonathan R. (2008) "General-Dimensional Constrained Delaunay and Constrained Regular Triangulations". Discrete & Computational Geometry 39 (1–3): 580–637.

Patching the Triangle.jl library

Triangle.jl package

The julia package https://github.com/cvdlab/Triangle.jl is a Julia interface to Jonathan R. Shewchuk's Triangle library.

Triangle is a triangular mesh generator written in C, well known in Computational Geometry, Computer Graphics and Gaming areas.

In particular, Triangle is a fast and robust two-Dimensional quality mesh generator and Delaunay triangulator.

At the moment the https://github.com/cvdlab/Triangle.jl package will use only the Constrained Delaunay Triangulation (CDT) Generator, planning to expand later.

Removing holes from the CDT in 2D

When triangulating a complex shape in 2D, the CDT in Triangle requires to input the edges of shape boundary, in order to:

- compute a triangulation of the shape's convex hull
- remove the triangles of convex hull which are external to the boundary (when this one is concave)
- remove the triangles inside the inner holes of the shape
- provide a point (its coordinates) inside each hole, in order to start a recursive triangle elimination inside the hole

When holes are not known in advance ?

We discuss a topological procedure working in any case (with and without holes):

1. Input: a soup of boundary edges, i.e. the LAR model `(V, EV)` as a geometrical graph
2. Output: the triangulation of the interior, as LAR model `(V, [EV, FV])` with 0-, 1-, and 2-cells of the triangulation
3. Elaboration:
    1. compute the CDT of (V, EV), getting back the `triangles` partitioning the outer boundary cycle;
    2. select the outer and inner edges;
    3. select the inner `triangles` subset with boundary made by inner edges;
    4. remove the inner triangles.

Implementation 1/3

```
"""
    triangulate2d(V, EV)

Contrained Delaunay Triangulation of LAR model (V,EV).
Discovery and removal of holes from triangulation, by comparing
original and generated edges.
"""
function triangulate2d(V, EV)
    # data for Constrained Delaunay Triangulation (CDT)
    points = convert(Array{Float64,2}, V')
    points_map = Array{Int64,1}(collect(1:1:size(points)[1]))
    edges_list = convert(Array{Int64,2}, hcat(EV...)')
    edge_boundary = [true for k=1:size(edges_list,1)]
    triangles = Triangle.constrained_triangulation(points,points_map,edges_list)
    # edges of the triangulation
    ev = map(sort,cat([[[u,v], [v,w], [w,u]] for (u,v,w) in triangles]))
    # remove duplicated edges from triangulation
    ev_nodups = collect(Set(ev))
    ##Plasm.view(Plasm.numbering(0.35)((V,[[[k] for k=1:size(V,2)], ev_nodups])))
    # dictionary o original edges
    edge_dict = Dict(zip(EV,1:length(EV)))
```

Implementation 2/3

```
triaedges = Array{Int64,1}(undef,0)
for (u,v) in ev
    if haskey(edge_dict, [u,v])
        push!(triaedges, edge_dict[[u,v]])
    elseif haskey(edge_dict, [v,u])
        push!(triaedges, edge_dict[[v,u]])
    end
end

# subdivide original edges between inner and outer
edge_boundary = Vector{Bool}(undef,length(triaedges))
counters = zeros(size(edges_list,1))
for e in triaedges
    counters[e]+=1
end
edge_boundary = Vector{Bool}(undef,length(triaedges))
for e in triaedges
    edge_boundary[e] = counters[e] == 1 ? false : true
end
```

Implementation 2/3

```
    # compute inner triangles
    inneredges = Array{Array{Int64,1},1}()
    for (k,value) in enumerate(counters)
       if value==2
           push!(inneredges, EV[k], reverse(EV[k]))
       end
    end
    # compute hole(s): wheater all triangle edges are inneredges
    holes = Array{Array{Int64,1},1}()
    for (k,(u,v,w)) in enumerate(triangles)
        triangle = [[u,v],[v,w],[w,u]]
        if setdiff(triangle,inneredges)==[]
            push!(holes, triangles[k])
        end
    end
    triangles = [tria for tria in triangles if !(tria in holes)]
    return triangles
end
```

Implementing Alpha-shapes in 2D

Alpha shape and Alpha complex (from Wikipedia)

In computational geometry, an alpha shape, or α-shape, is a family of piecewise linear simple curves in the Euclidean plane associated with the shape of a finite set of points

Alpha shapes are closely related to alpha complexes, subcomplexes of Delaunay triangulation of the point set.

- Each edge or triangle of the Delaunay triangulation may be associated with a characteristic radius, the radius of the smallest empty circle containing the edge or triangle
- For each real number α, the α-complex of the given set of points is the simplicial complex formed by the set of edges and triangles whose radii are at most $1/\alpha$.
- The union of the edges and triangles in the α-complex forms a shape closely resembling the α-shape; however it differs in that it has polygonal edges rather than edges formed from arcs of circles

Implementation? (in class)

1. Start from a file `<yourrepo>/LinearAlgebraicRepresentation.jl/examples/2d/alphashape.jl`
2. Insert the header:

```
using Plasm, Triangle
using LinearAlgebraicRepresentation
Lar = LinearAlgebraicRepresentation
```

3. Using some drawing program of your machine, write some graphic text: `"Lar"` is OK (including double quotes)[3]
4. Make the graphic text fairly big, and export as .SVG file
5. Import the file and transform in LAR model format using https://github.com/cvdlab/Plasm.jl/tree/julia-1.0/src
6. Generate a random set of points inside the shape. HINT: use code from https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/julia-1.0/examples/2d/svg2lar.jl
7. finally write a parametric function ($0 \leq \alpha \leq 1$) to compute the α-complex of any set `V` of 2D points:

[3] Useful link: table-of-8-bit-ascii-character-codes