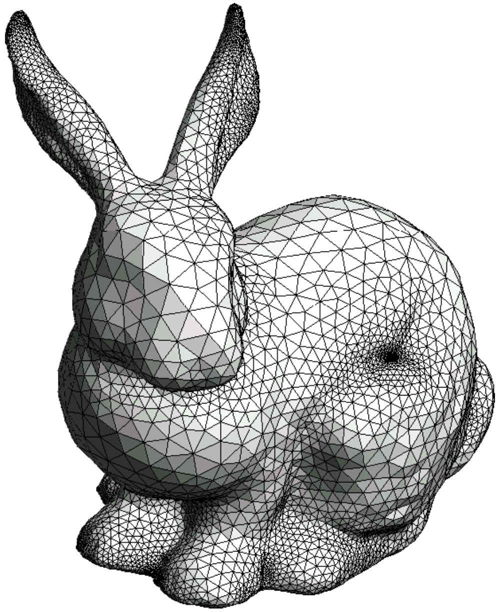


Geometric Modeling in Graphics

Part 5: Mesh repairing

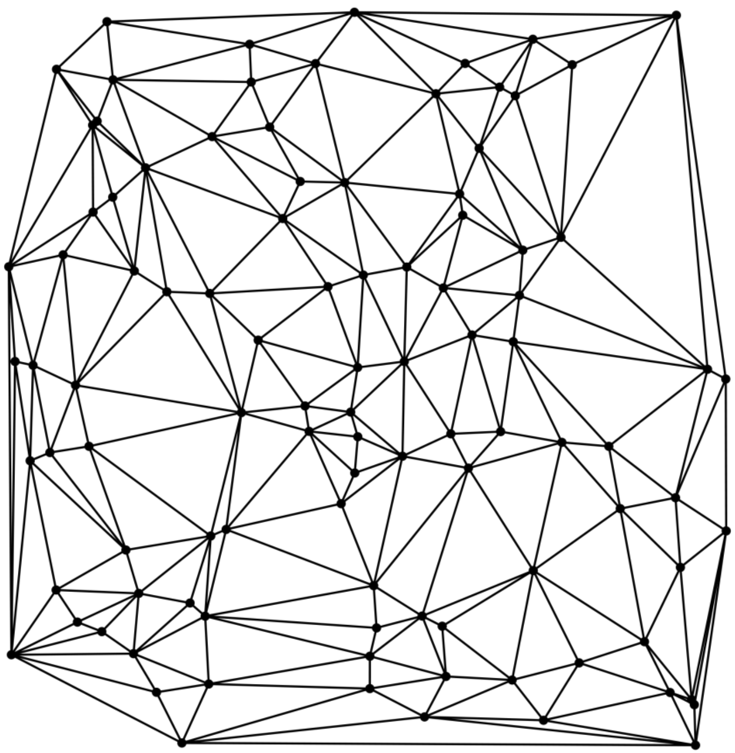
Martin Samuelčík
www.sccg.sk/~samuelcik
samuelcik@sccg.sk

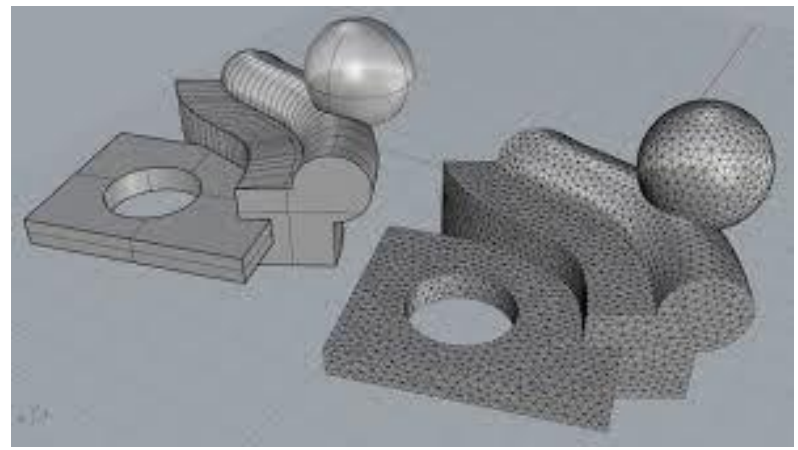
Mesh repairing

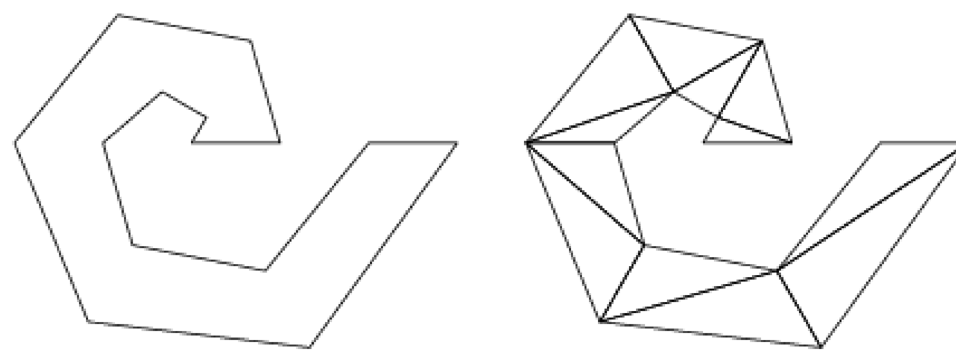
- Joining identical vertices
- Removing degenerated (empty) polygons and edges
- Removing duplicated faces
- Creating consistent orientation
- Fixing manifoldness, remeshing
- Preparing mesh with only simple polygons, triangulation
- Creating closed solid objects, watertight mesh, filling holes
- Overview of repairing software
  - http://meshrepair.org/

Triangulation

- Converting polygonal mesh to triangular mesh
- 2D manifold polygons – decomposing polygon to triangles







Ear clipping

- Simple polygon with n ordered vertices $V_0, V_1, \ldots, V_{n-1}$
  - $V_{-1} = V_{n-1}, V_n = V_0$
- Assuming counterclockwise orientation – interior is to the left when traversing
- Ear of polygon – triangle $V_{i-1}V_iV_{i+1}$
  - V_i is convex vertex – angle at V_i is less than π radians – ear tip
  - Line segment $V_{i-1}V_{i+1}$ lies inside polygon - diagonal
  - No other vertices V_j lies inside ear
- Polygon of four or more sides always has at least two non-overlapping ears
- Ear removal – reducing number of polygon vertices by 1
- http://www.cosy.sbg.ac.at/~held/projects/triang/triang.html

Detecting ears

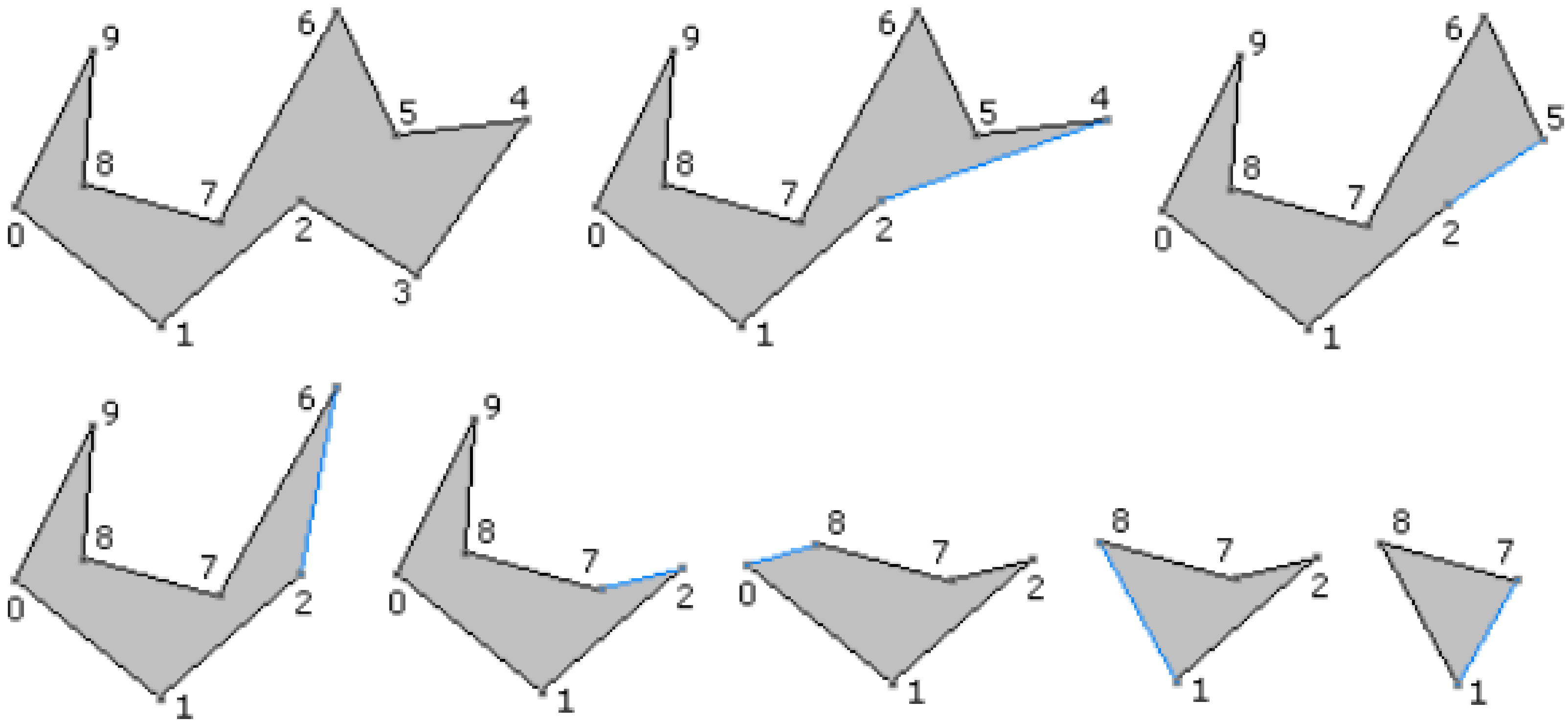
- Iterate over vertices V_i
- Test all other vertices $V_0,\ldots,V_{i-2},V_{i+2},\ldots,V_{n-1}$ if any are inside triangle $V_{i-1}V_iV_{i+1}$
- Test only reflex vertices – interior angle at vertex is larger than π radians
  - Reflex vertex V_j – $(V_j-V_{j-1})x(V_{j+1}-V_j)$ has in 3D negative third coordinate
  - Convex vertex V_j – $(V_j-V_{j-1})x(V_{j+1}-V_j)$ has in 3D positive third coordinate
- Maintaining lists of vertices *V*, list of reflex vertices *R* and (ordered) list of ear tips *E* during triangulation

Ear clipping algorithm

- 1. Given initial list of vertices *V*
- 2. Construct initial list *R* of reflex vertices and construct list *E* of ear tips using list *R*
- 3. Pick (random or with minimal inner angle) and remove one ear tip V_i from *E*
  - Add triangle $V_{i-1}V_iV_{i+1}$ to final triangulation
  - Remove V_i from list *V*
  - Update *R* and *E* with adjacent vertices V_{i-1}, V_{i+1}
    - If the adjacent vertex is reflex, it is possible that it becomes convex and, possibly, an ear
    - If an adjacent vertex is an ear, it does not necessarily remain an ear
- 4. Repeat 3. until list *V* contains only 3 vertices – last triangle of triangulation

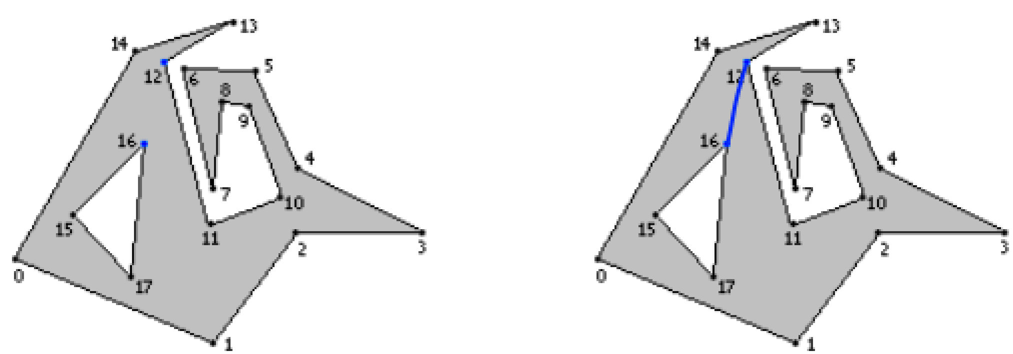
Ear clipping algorithm

- Time complexity $O(n^2)$
- http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf



Polygons with holes

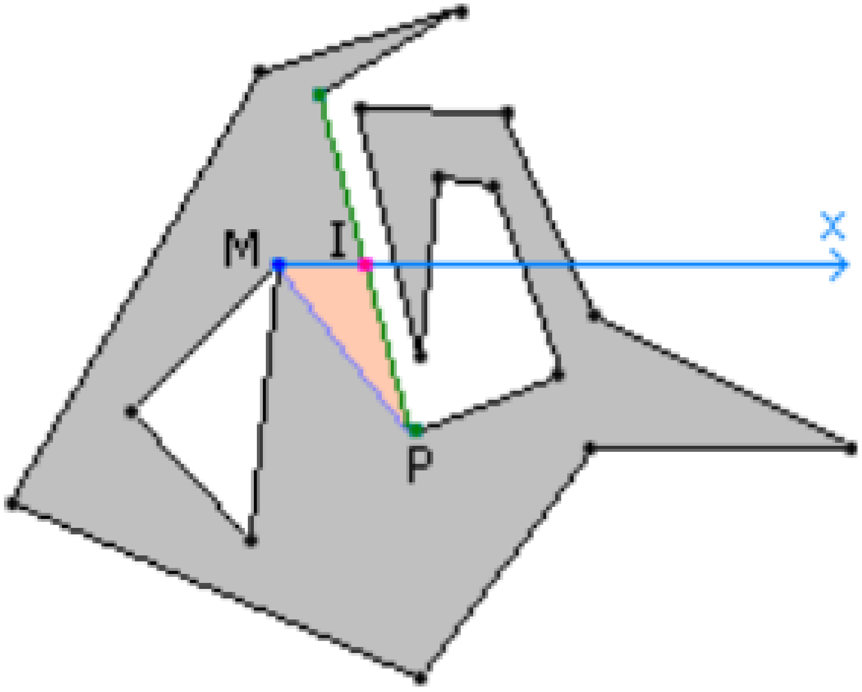
- One outer polygon
- Several non-intersecting inner polygons with opposite ordering as outer polygon
- Finding two mutually visible vertices, one from outer loop, one from inner loop
- Connect two mutually visible vertices and combine inner and outer loop into one outer loop

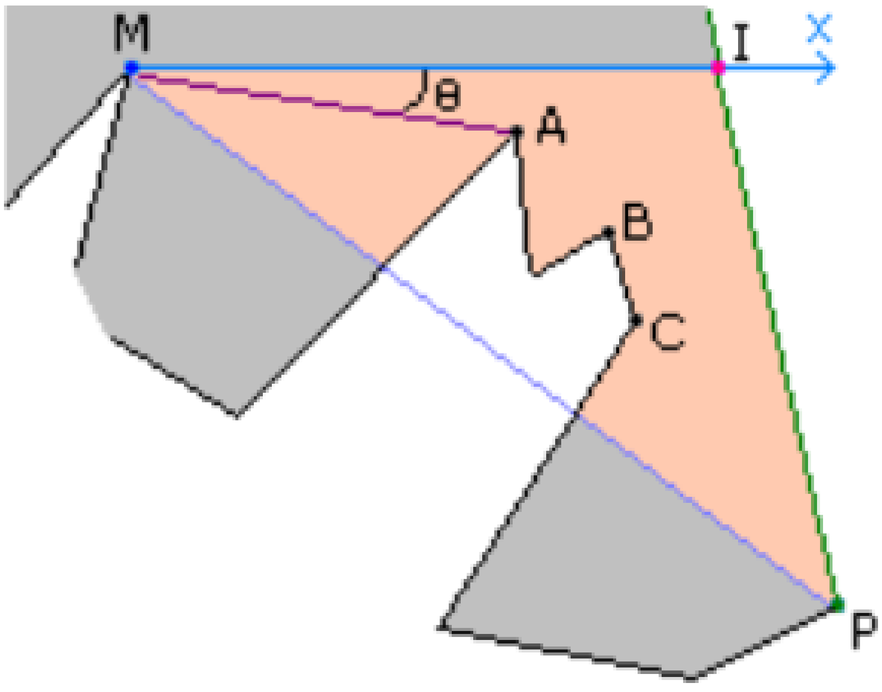


Finding visible vertices

- 1. Find vertex M of inner loop such that its x-coordinate is maximal for all vertices of all inner loops.
- 2. Intersect the ray M + t(1, 0) with all directed edges V_i ,V_{i+1} of the outer polygon for which M is to the left of the line containing the edge. Let I be the closest visible point to M on this ray.
- 3. If I is a vertex of the outer polygon, then M and I are mutually visible.
- 4. Otherwise, I is an interior point of the edge V_i ,V_{i+1}. Select P to be the endpoint of maximum x-value for this edge.
- 5. Search the reflex vertices of the outer polygon except P. If all of these vertices are strictly outside triangle (M, I, P), then M and P are mutually visible.
- 6. Otherwise, at least one reflex vertex lies in (M, I, Pi). Search for the reflex vertex R that minimizes the angle between (1, 0) and the line segment (M, R). Then M and R are mutually visible. There can be multiple reflex vertices that minimize the angle, in this case choose the reflex vertex on this ray that is closest to M.

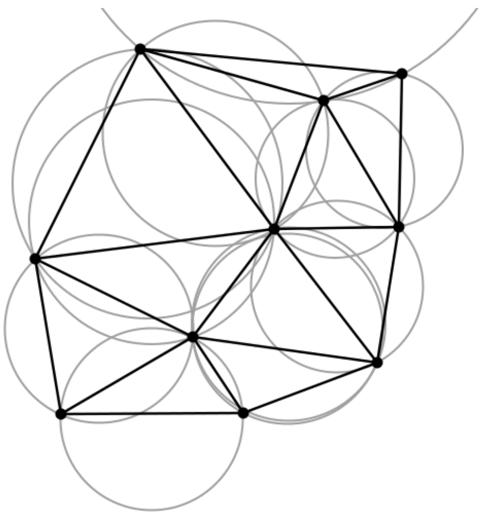
Finding visible vertices

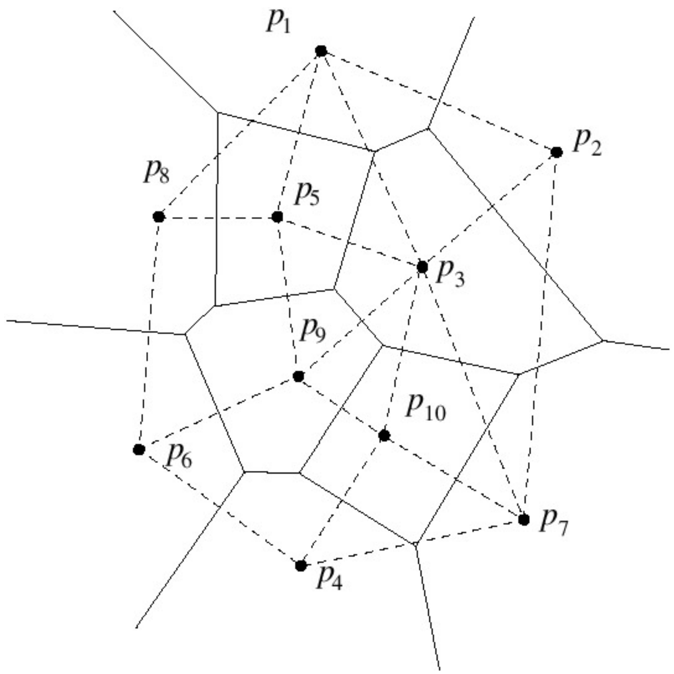




Delaunay triangulation

- Triangulation for set of points in plane, dual graph to Voronoi diagram
- Points p_i, p_j, p_k combine into triangle in DT ⇔ circumscribed circle for points p_i, p_j, p_k does not contain any other point – Delaunay property
- DT maximizes minimal inner angle



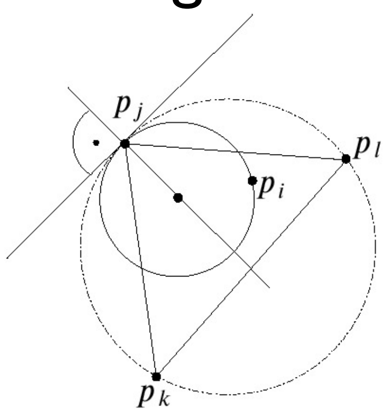


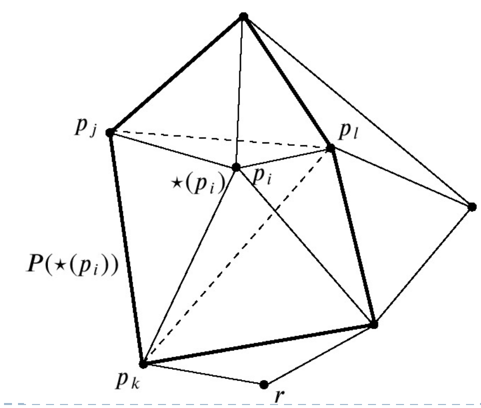
Delaunay triangulation

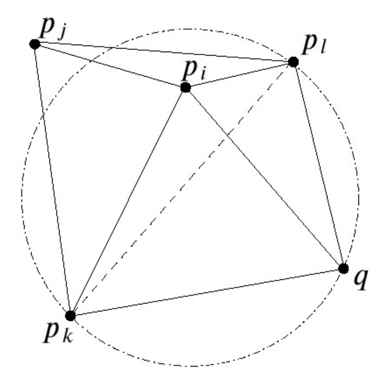
- Construction algorithm using point insertion
- Given set S of points in plane
- 2 cases when inserting new point to already created triangulation
  - Point is inserted inside convex hull of points from S – inserted point is inside one triangle of current DT
  - Point is inserted outside convex hull of points from S
- After insertion, Delaunay property can be broken – fixed by multiple edge flips
- Time complexity $O(n^2)$ in worst case, O(n.log(n)) in average case, can be extended to O(n.log(n)) in worst case

DT construction – 1. case

- New point p_i lies in triangle $T = \Delta(p_j, p_k, p_l)$
- Edges p_ip_j, p_ip_k, p_ip_l belong to new DT
- Conflict with Delaunay property can be in neighbors of T
- Maintaining list $P(*(p_i))$ – all edges that are candidates for flipping
- If some edge from $P(*(p_i))$ is flipped, then it is removed from $P(*(p_i))$ and two adjacent edges are added
- If no edge from $P(*(p_i))$ is flipped, algorithm terminates

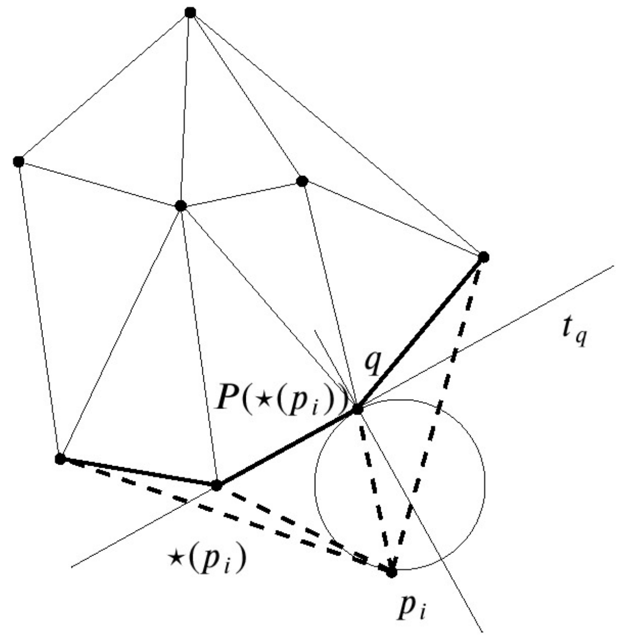






DT construction – 2. case

- New point p_i does not lie in convex hull of original DT
- For each point from q from S, that are visible from p_i, edges p_iq are part of new DT
- Again flipping edges that breaks Delaunay property of DT and updating list of active edges P(*(p$_i$))



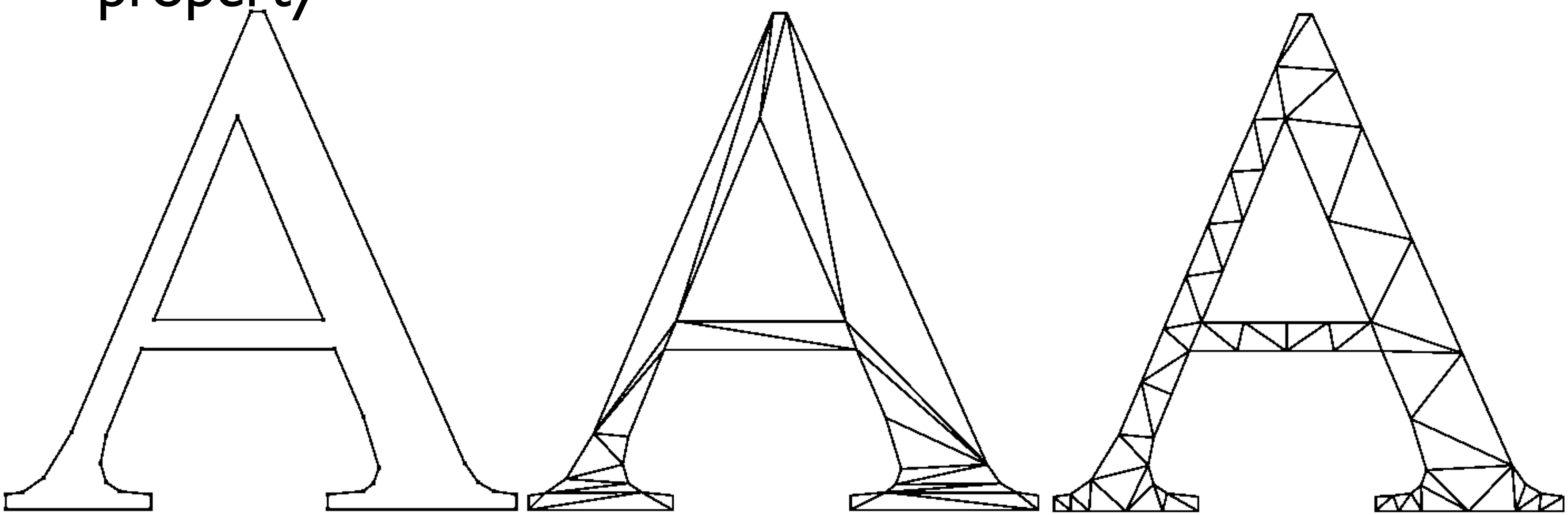
DT construction algorithm

```
Delaunay(S)  (S is set of sites)
{
        T = new array;
        while (S.size() > 0)
        {
                p = S.First;
                S.DeleteFirst;
                T.InsertSite(p);
        }
}
```

```
InsertSite(T, p) (T represents the current Delaunay triangulation, p is a new site)
{
    t = T.FindTriangle(p);
    if (t != NULL)
        Star(p) = t.CreateStar(p);
    else
        Star(p) := T.HullEdges(p);
    T.Insert(Star(p), t);
    StarPoly = t.Edges();
    while (StarPoly.size() > 0)
    {
        e = StarPoly.First();
        StarPoly.DeleteFirst();
        q = p.Opposite(e);
        if (q ≠ NULL)
        {
            (r, s) = e.EndPoints();
            if (InCircleTest(p, r, s, q))
            {
                T.Remove(e);
                T.Add((p, q));
                StarPoly.Add((r,q));
                StarPoly.Add((s,q));
            }
        }
    }
}
```

Additional DT

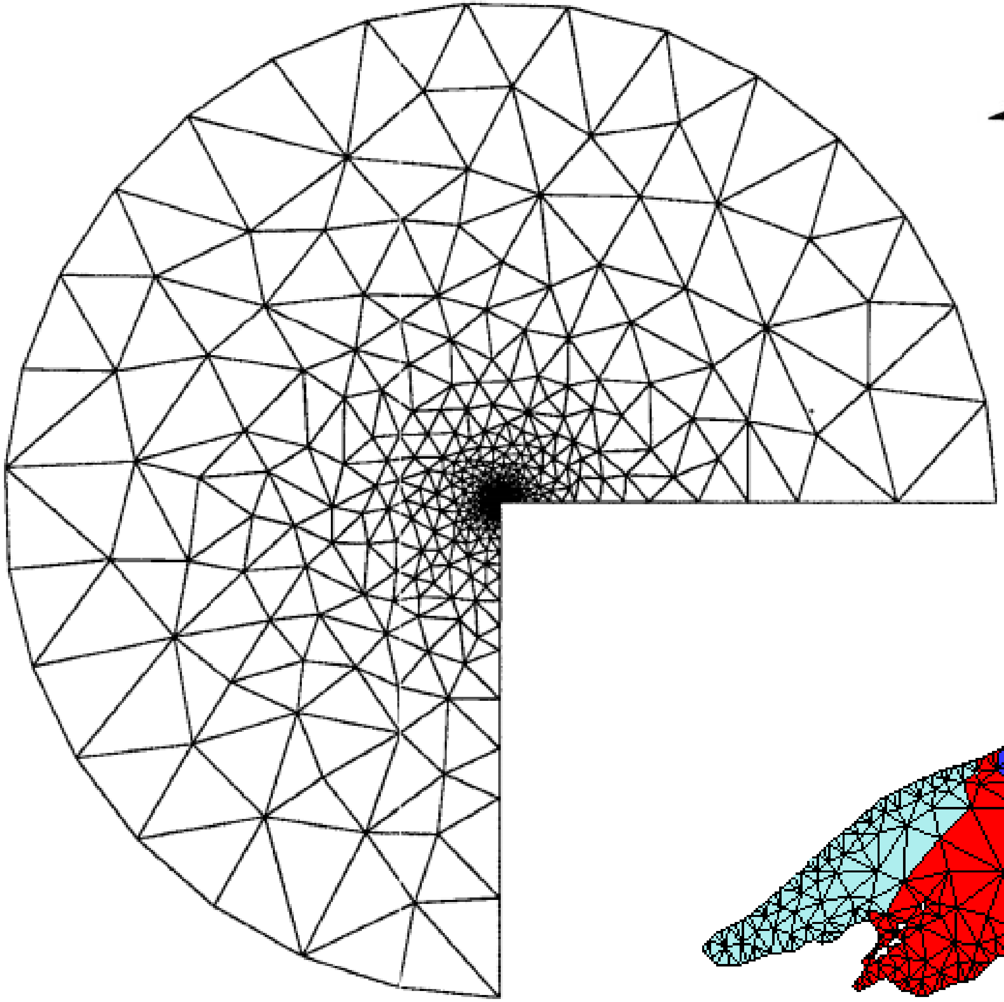
- Constrained DT – additional set of edges that must be in triangulation, endpoints of edges are in S, introducing special constrained Delaunay property
- Conforming DT – still constrained by set of edges, algorithm adds new (Steiner) points to maintain Delaunay property

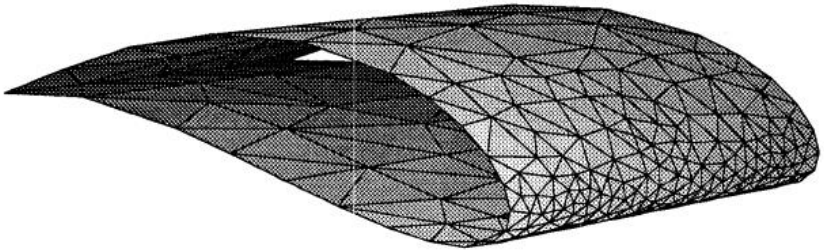


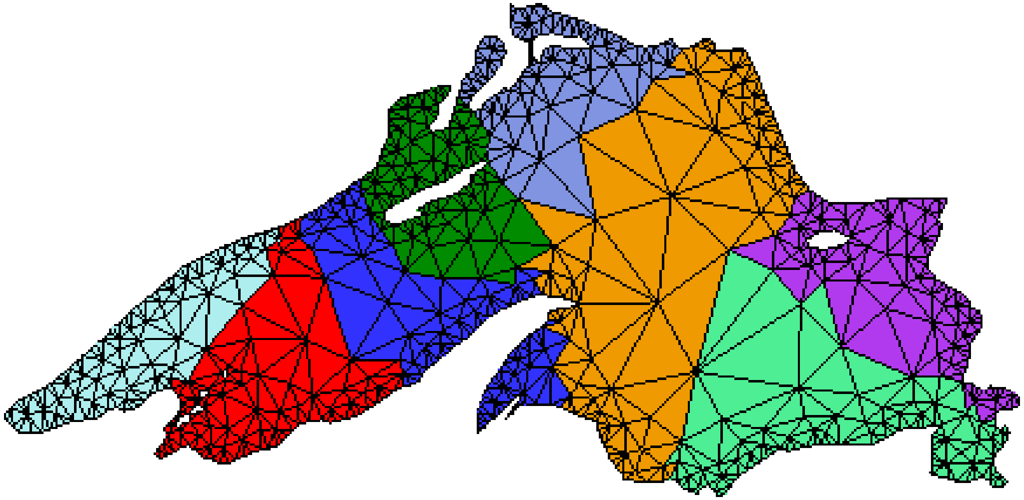
Quality triangulation

- https://kogs-www.informatik.uni-hamburg.de/~tchernia/SR_papers/chew93.pdf
- https://www.ics.uci.edu/~eppstein/pubs/BerEpp-CEG-95.pdf
- Introducing two criteria for triangle grading
  - Triangle is well-shaped if all its angles are greater than or equal to 30 degrees
  - Triangle is well-sized if it fits within a circle of given radius and satisfy the grading function
- Build over constrained DT by inserting new special points for each bad-graded triangle
- Extended for curved surfaces in 3D
- https://www.cs.cmu.edu/~quake/triangle.html

Quality triangulation

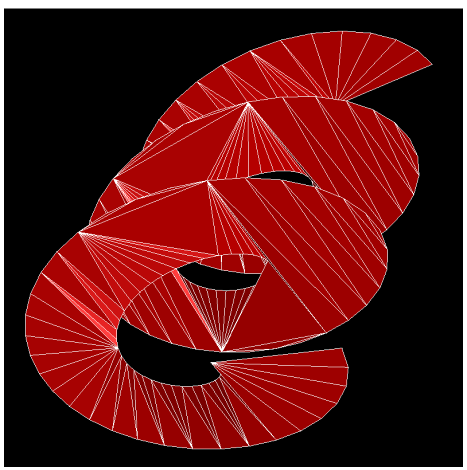


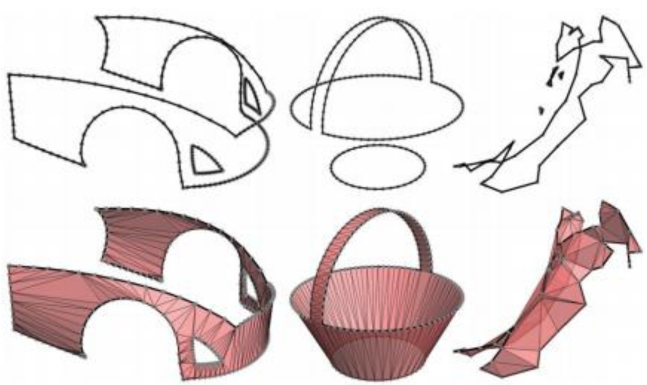


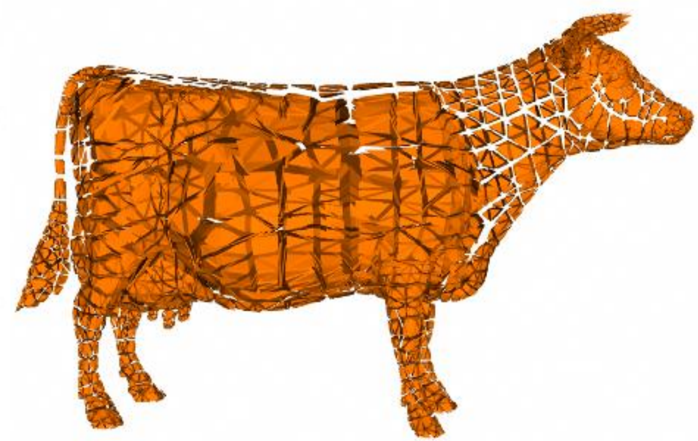


Triangulation in 3D

- Triangulating non-planar polygon
- Ear clipping in 3D
- Projecting 3D points on principal plane
- Delaunay based curved surface triangulations
- Tetrahedralization of 3D points
- Filling holes algorithms for meshes

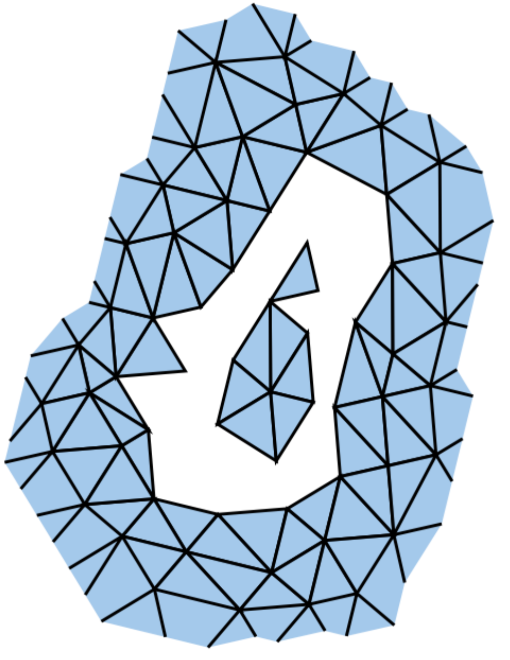






Filling holes in meshes

- Creating closed, watertight meshes
- Several connectivity components
- Surface-oriented vs volumetric algorithms
- Handling islands, non-manifoldness



Filling holes

- Peter Liepa: Filling Holes in Meshes
  - http://www.brainjam.ca/papers/papers.htm
- Surface-oriented algorithm, not handling islands
- 1. Identify non-empty contours that represents holes
  - User-defined , topology-defined holes
- 2. Compute coarse triangulation T to fill hole
  - Weighting each triangle by its area and maximal angle of triangle and its adjacent triangles
  - Iterative computation of triangulation that minimizes weight of its triangles, favoring triangulations of low area and low normal variation
  - Weight of larger polygon is computed from weight of triangle and weight of smaller polygon
  - Time complexity $O(n^3)$

Filling holes

- 3. Refine triangulation T to match vertex density of the surrounding area
  - Compute edge length data for the vertices on the hole boundary
  - Subdividing triangles of T with barycenter to reduce edge lengths
  - Swapping edges when necessary to maintain Delaunay-like property
- 4. Smooth the triangulation T to match the geometry of the surrounding area
  - Laplacian-based mesh smoothing
  - Minimizing umbrella-based operator(Vector Laplacian)
  - Solving linear system, variables are positions of vertices in T

Filling holes

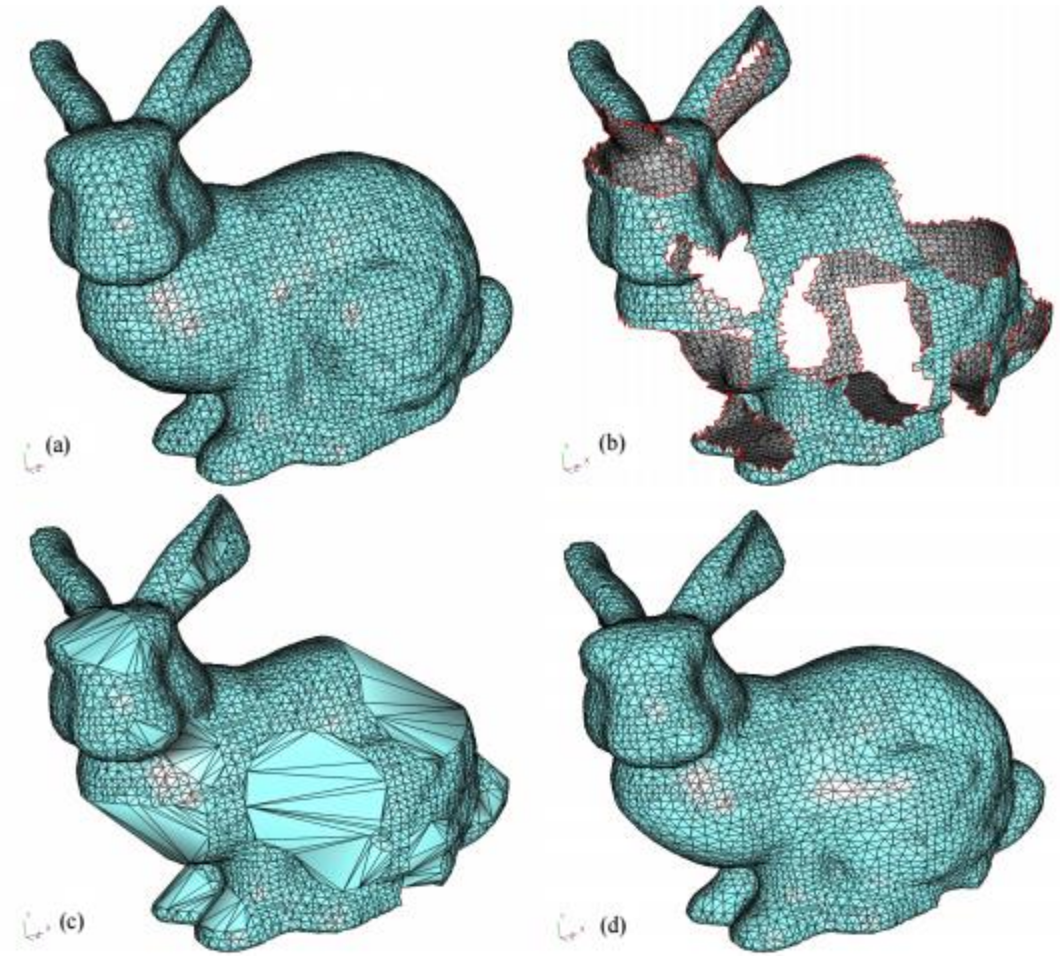
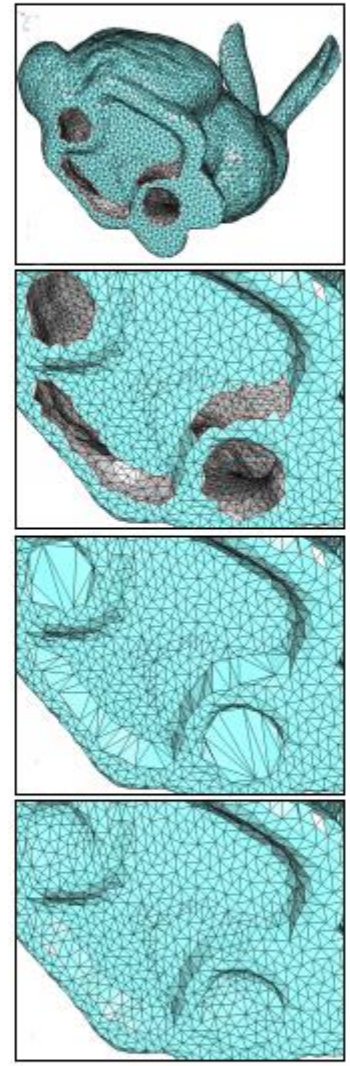
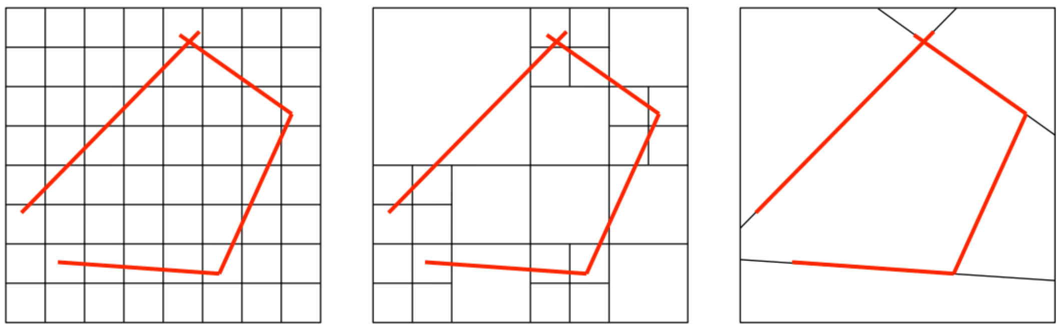


Figure 5: *(a) Stanford bunny. (b) Mutilated Stanford bunny. (c) After hole triangulation. (d) After meshing and fairing. (b) and (d) are reproduced in the color section in Figure 8.*



Volumetric mesh repair

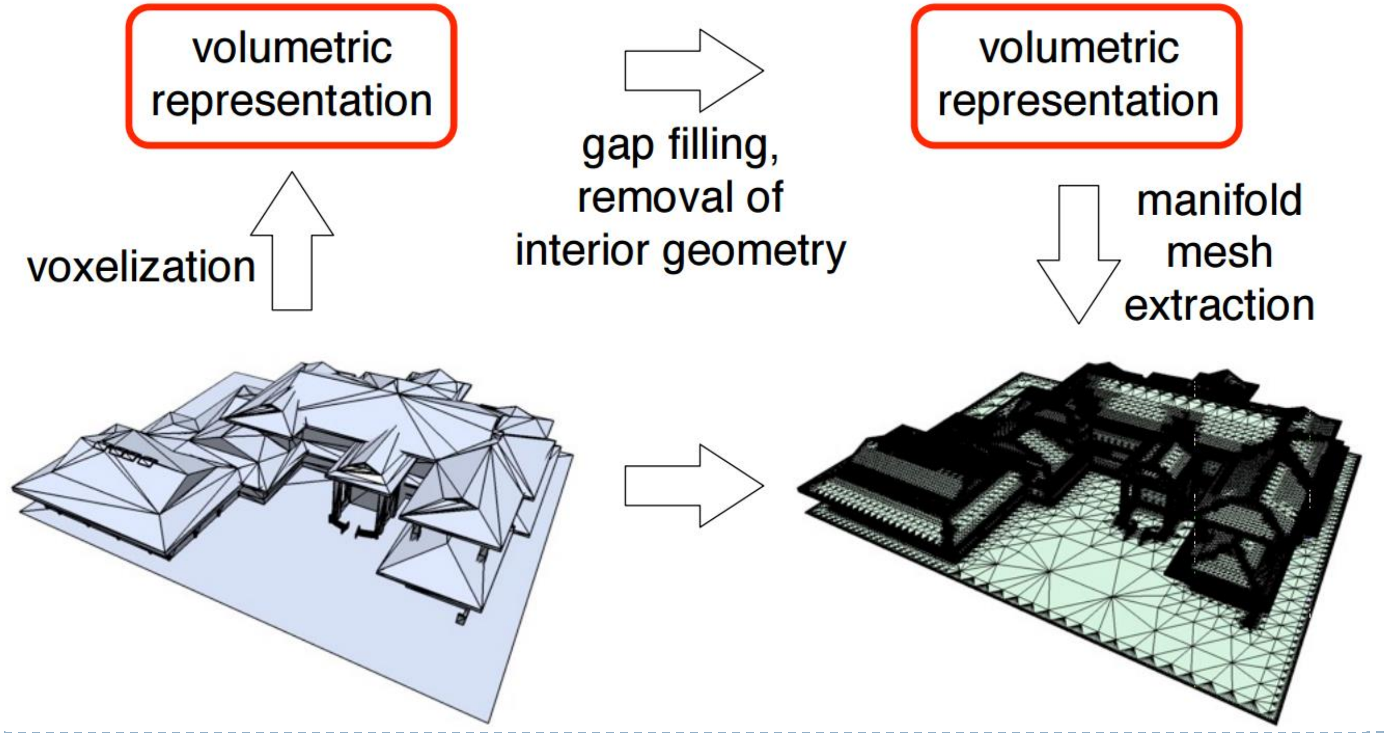
- Closing meshes, repairing non-manifoldness, creating more regular polygons, triangulation, remeshing
- 1. convert the input model into an intermediate volumetric representation
- 2. do robust and reliable processing with discrete volumetric representation – morphological operators (dilation, erosion), smoothing, interior/exterior identification
- 3. extract the surface of a solid object from the volume



voxel grid adaptive octree BSP tree

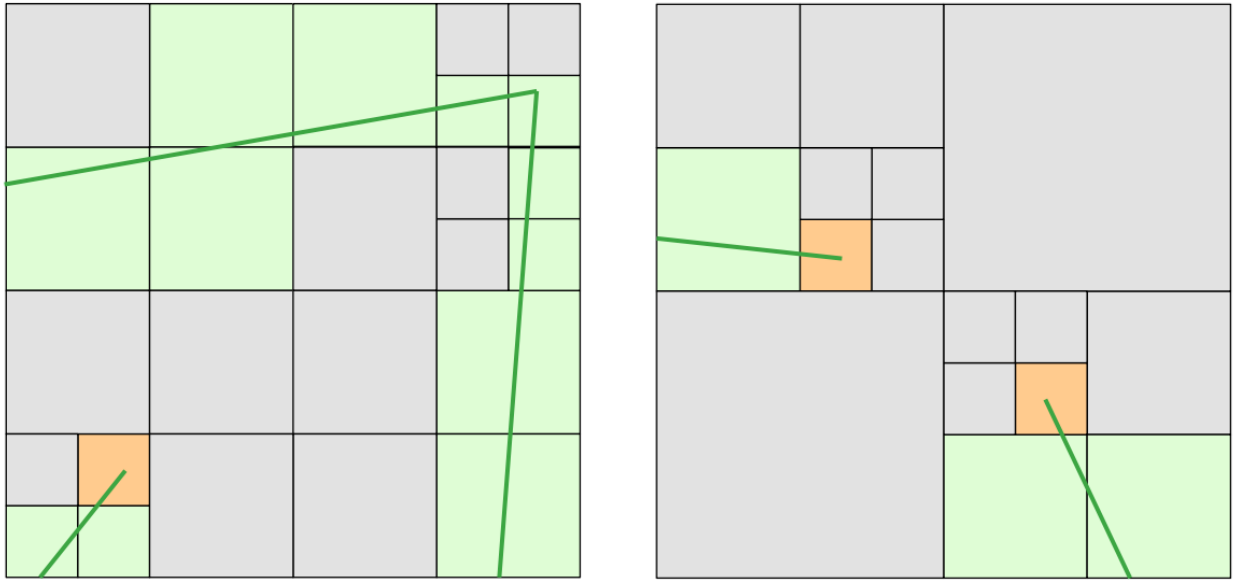
Volumetric mesh repair

- S. Bischoff, D. Pavic, L. Kobbelt: Automatic Restoration of Polygon Models
  - https://www.graphics.rwth-aachen.de/publication/86/automatic_restoration1.pdf



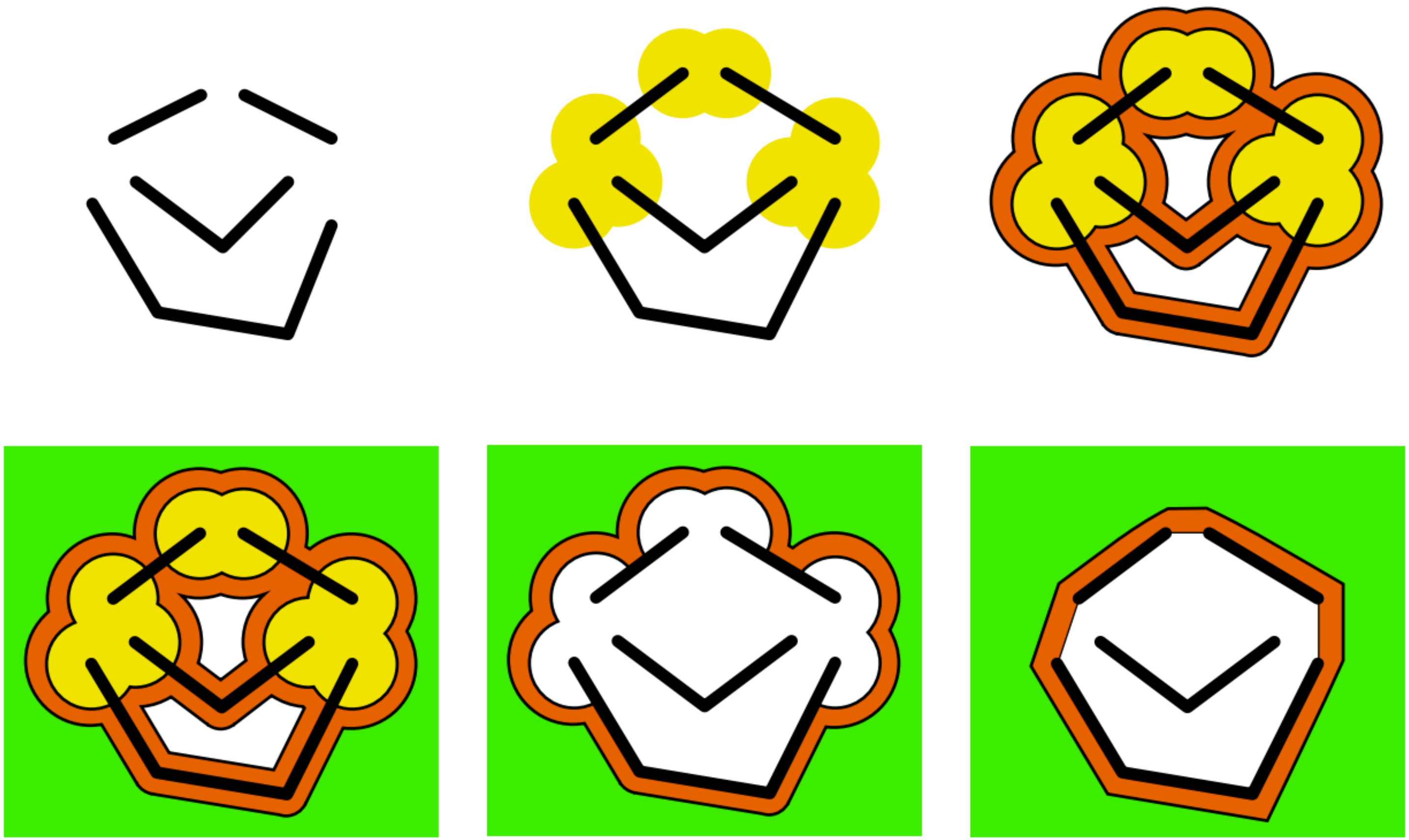
Voxelization

- Adaptive octree structure
- Voxels are used to determine mesh connectivity, topology
- Geometry of output mesh is given by input mesh
- Each voxel holds reference to triangles of input mesh
- Determining cells with boundary edges



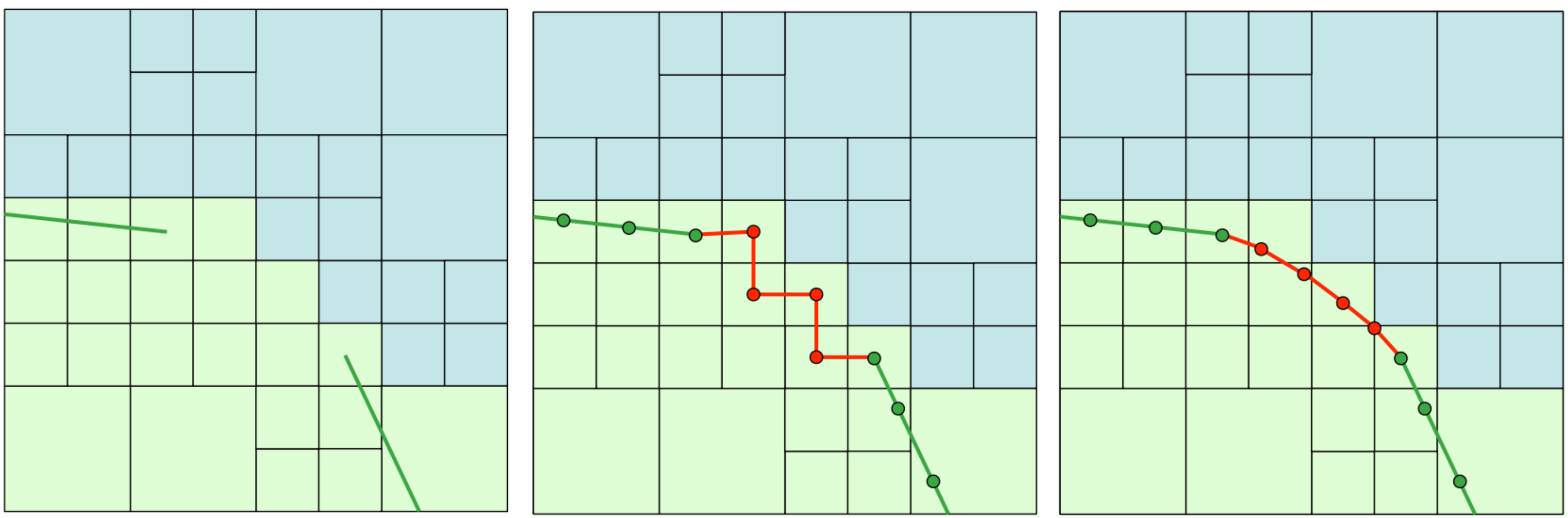
Closing gaps

- Dilating boundary edges in octree grid
- Computing new triangles for gap voxels

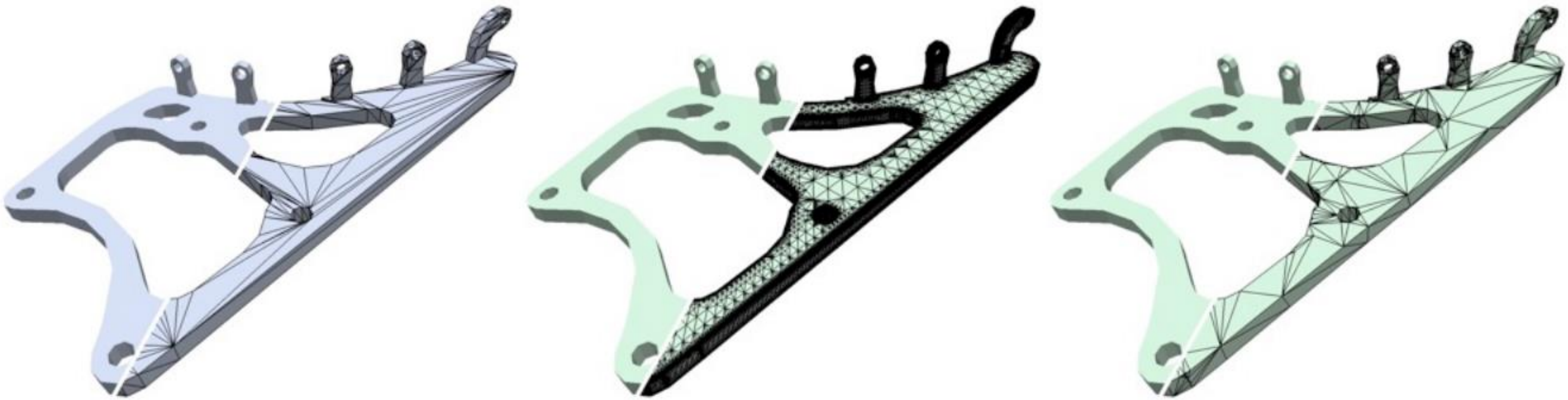


Surface reconstruction

- Dual contouring algorithm
- Creating new triangles for each leaf cell of octree
  - Connectivity of new triangles is given by cells neighborhood
  - Positions of new triangle vertices is computed from input mesh triangles referenced in cell



Volumetric mesh repair

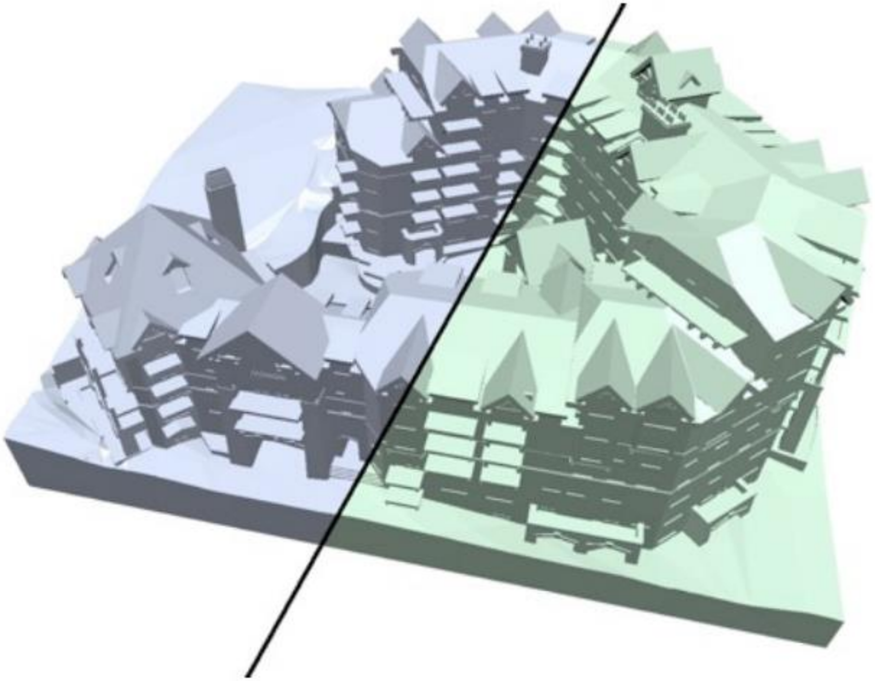


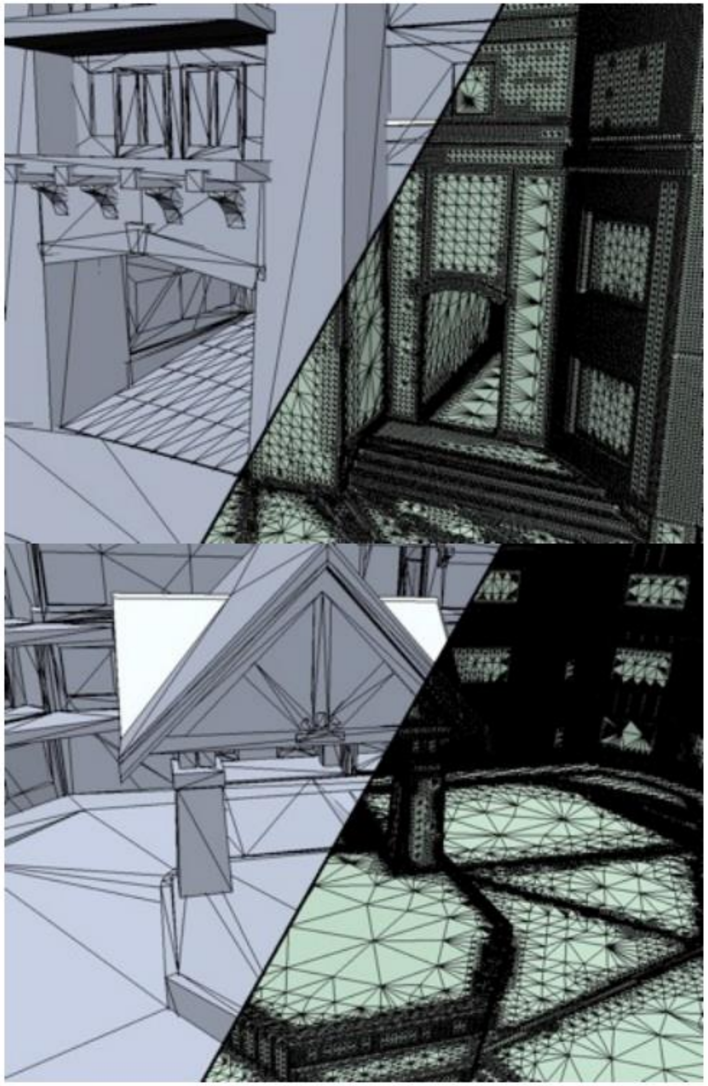
original
1124 triangles

reconstruction
279892 triangles
(at 1000^3)

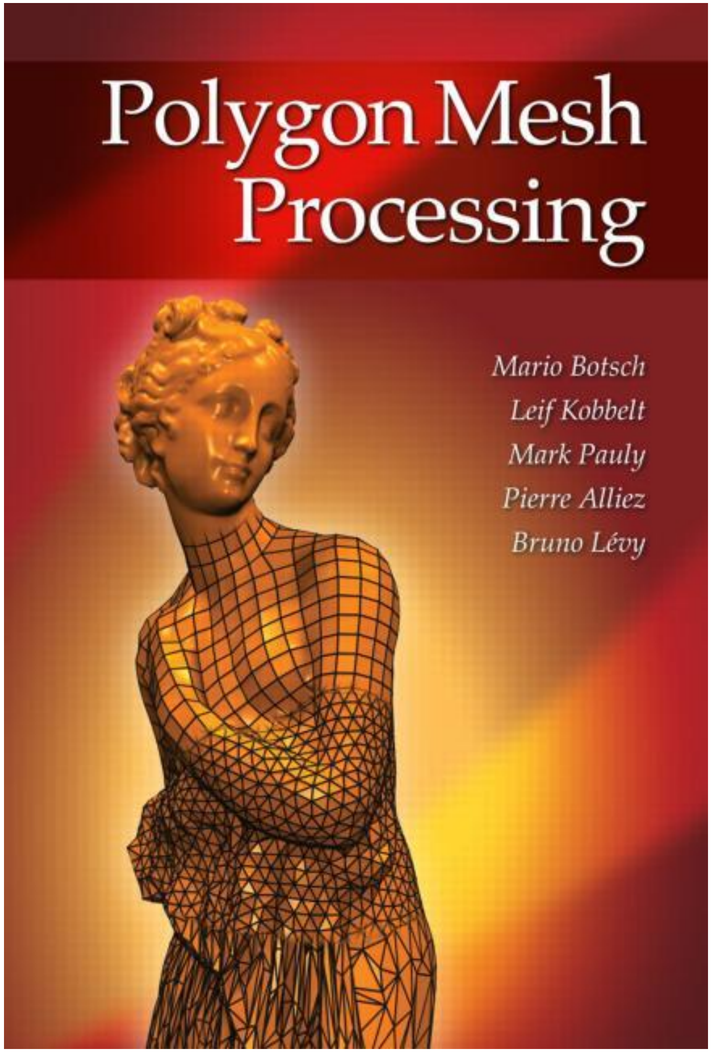
decimated
7018 triangles

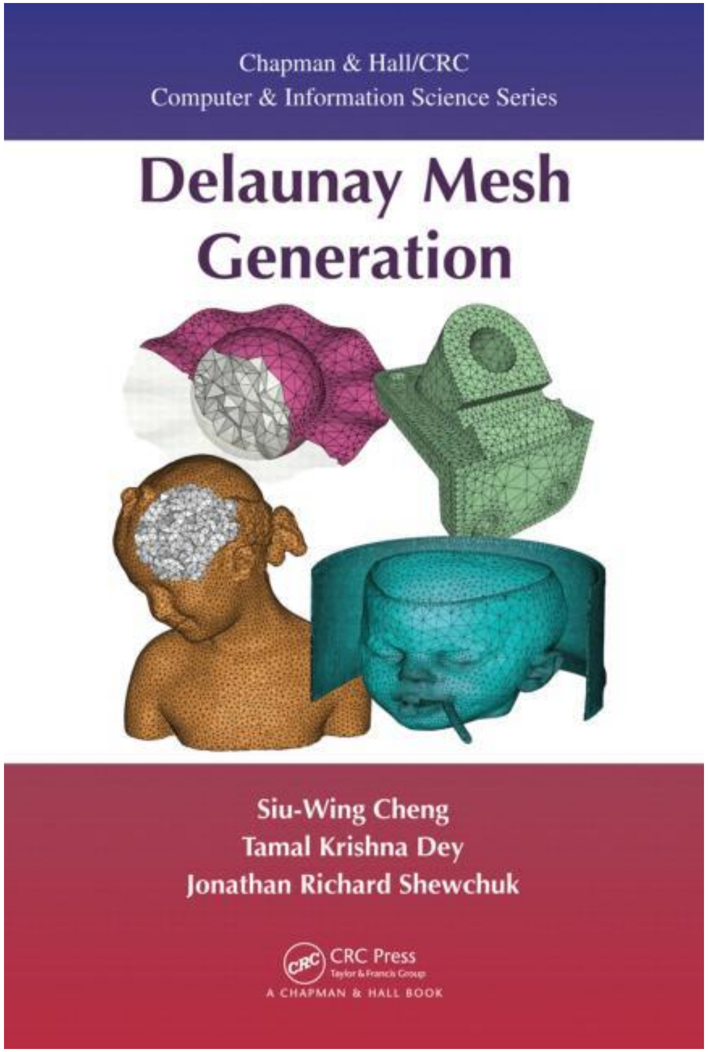
Volumetric mesh repair

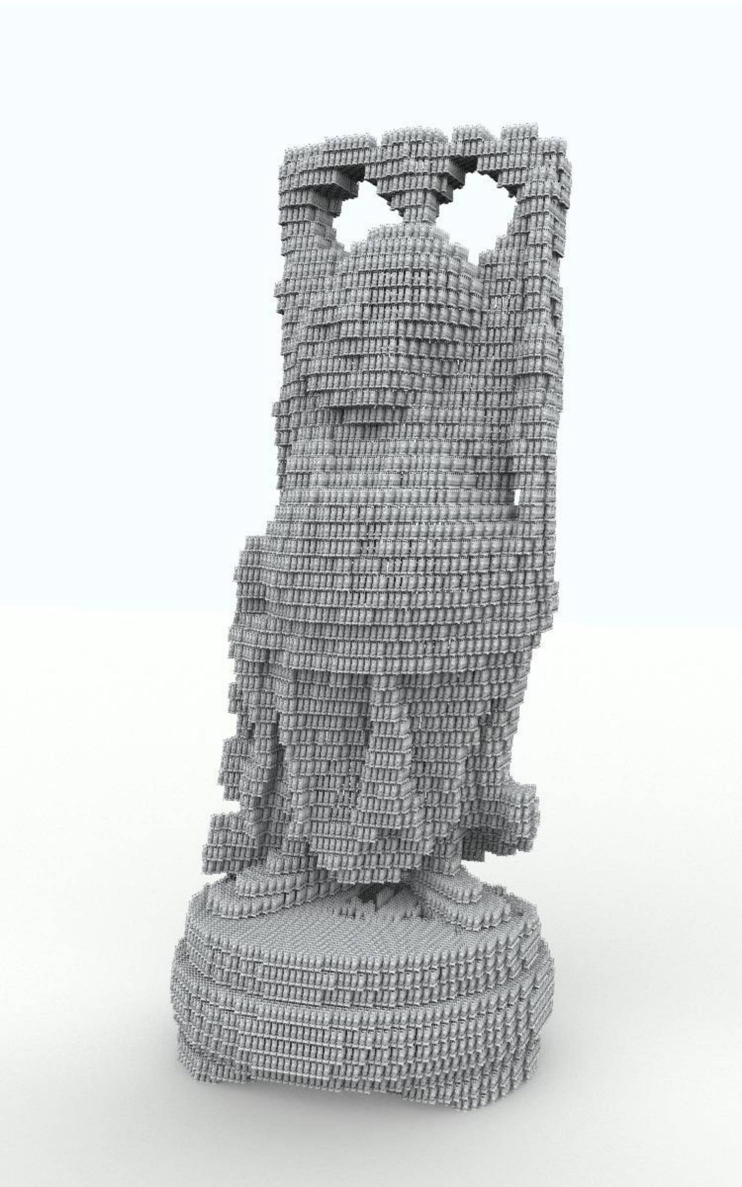




Additional resources







The End
for today