# Parallel & Distributed Computing: Lecture 5

Alberto Paoluzzi

October 7, 2019

# Contents

# Roots of project

# The context of our start in Solid Modeling: 1984-1988

solid modeling R&D was at the time on Lisp-machines!


we started on IBM first PCs and Apple Macintosh writing Solid Modeling software in Pascal

# Minerva: first solid modeler on a PC (1985–)

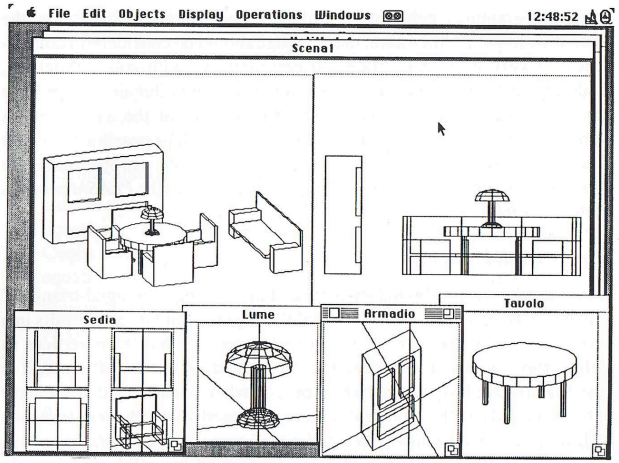



**Figure 1: The window-based user interface of *Minerva*, with a structure definition**

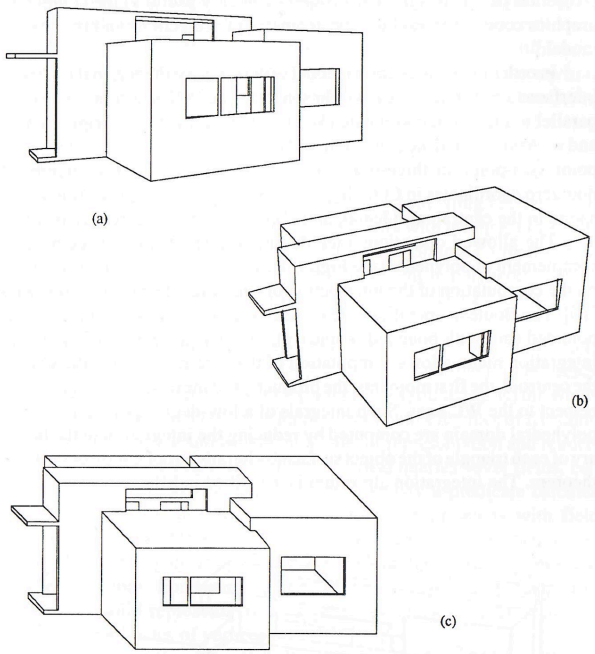



**Figure 3: Solid model of the Oud's Mathenesse (Rotterdam, 1923) generated by *Minerva*. (a) Two-point perspective; (b) three-point perspective; (c) dimetric Cavalier axonometric view**

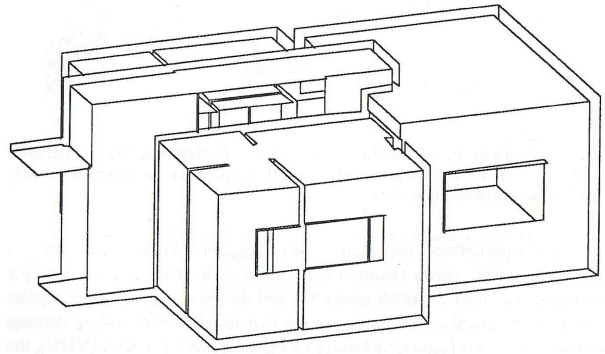


**Figure 4: Hidden surface removed view of the complemented Mathenesse building**

With hierarchical assemblies, Boolean ops, parametric curves & surfaces, integration of polynomials, HSR algorithms, full-fledged Apple GUI

# PLaSM from Backus' FL

## Programming language for solid variational geometry (CAD 92, TOG 95)

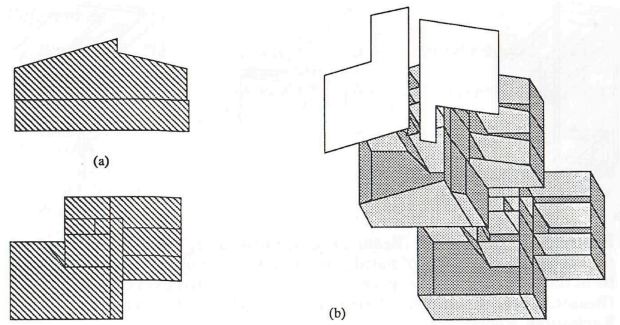



**Figure 6: (a) The arguments *Plan* and *Section* of an operation && (intersection cell-by-cell of extrusions) in the *PLASM* language; (b) The two-dimensional polyhedral complex obtained by the evaluation of the expression @2:(*Plan* && *Section*), represented as an exploded drawing**

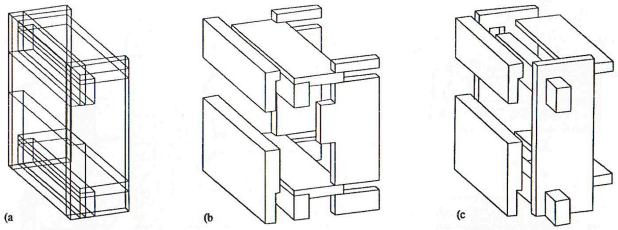



**Figure 7: (a) A sequence (Beams, Roofs, Enclosures, Partitions) of isothetic (interpenetrating) pairs of parallelopipeds, for sake of simplicity assumed to be in the same coordinate space; (b) *PLASM* evaluated expression *STRUCT*: (Beams, Roofs, Enclosures, Partitions); (c) STRUCT: (Beams, Partitions, Enclosures, Roofs)**

**Example 1.5.6 (Building facade)**
An example is given in Script 1.5.6, and shown in Figure 1.6a, where a 2D complex is generated by Cartesian product of 1D complexes produced by the `QUOTE` operator. Several other examples of the `QUOTE` operator are given in Section 1.6.2.

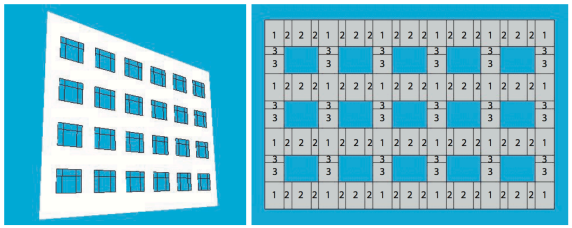


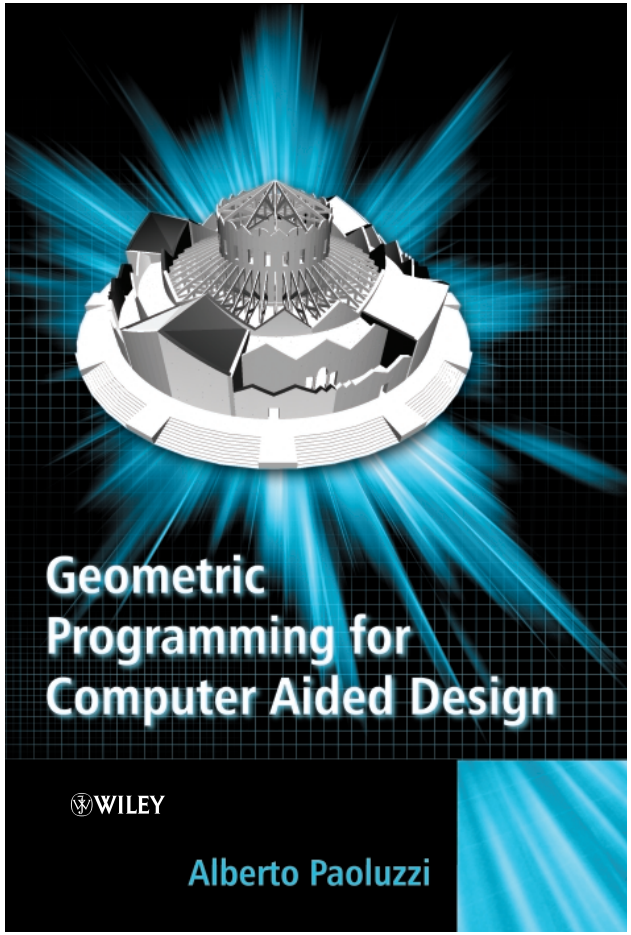
**Figure 1.6** (a) the 2D complex generated as `facade:<6,4>` by combining `QUOTE`, product and 1-skeleton operators (b) the generating scheme of `facade` panels

INTRODUCTION TO FL AND PLASM 31

The `facade` generating function works by assembling three 2D Cartesian products of alternating 1D complexes, produced by `Q:xRithm`, `Q:xVoid` and by `Q:yRithm`, `Q:yVoid`, respectively. In particular, the `xRithm` sequence contains the numeric series used in the $x$ direction; analogously `yRithm` for the $y$ direction. Conversely, `xVoid` and `yVoid` host the series with opposite signs of elements. So, the first three Cartesian products in the `STRUCT` sequence produce a sort of checkboard covering that follows the scheme given in Figure 1.6b.

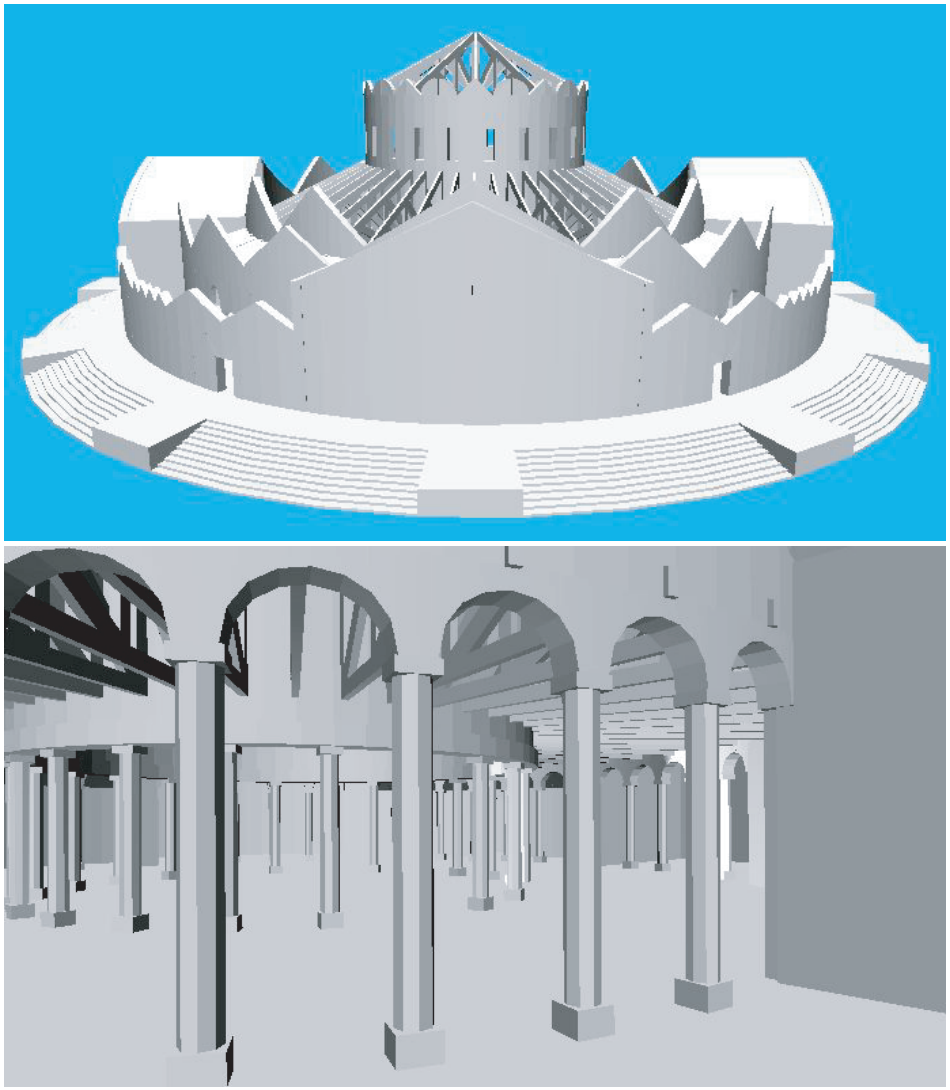
**Script 1.5.6 (Building facade)**

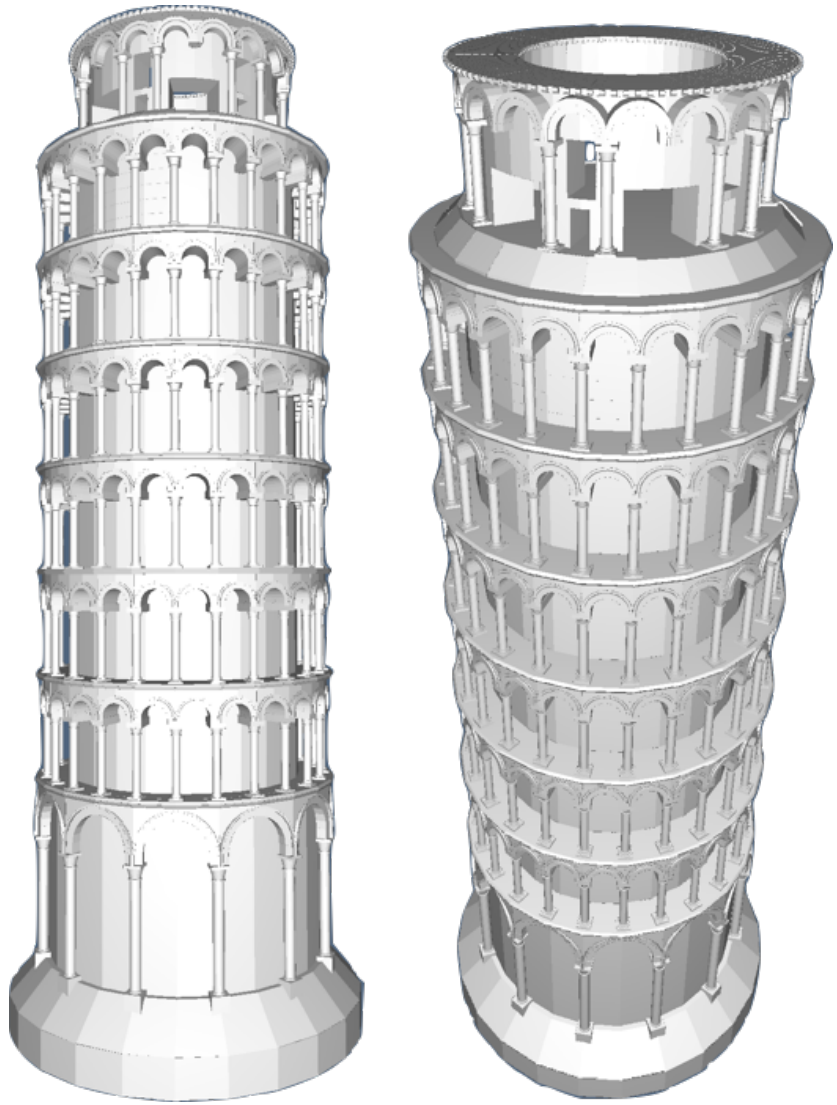
```
DEF facade (n,m::IsIntPos) = STRUCT:<
   Q:xRithm * Q:yRithm,
   Q:xVoid * Q:yRithm,
   Q:xRithm * Q:yVoid ,
   @1:(Q:xVoid * Q:yVoid) >
WHERE
   xRithm = ##:n:<5,-2,-5,-2> AR 5,
   yRithm = ##:m:<7,-5,-2> AR 7,
   xVoid = AA:-:xRithm,
   yVoid = AA:-:yRithm
END;
```



# PyPlasm : porting PLaSM to python package (2008–)

https://github.com/plasm-language/pyplasm





# Motivations and development path

# Geometric and topological computing

## rethinking some foundations

Complexity of geometric information stems from dramatic increase in size, diversity, and complexity of geometric data:

- point clouds,
- boundary meshes,
- NURBs representations,
- finite element meshes,
- CT scans,
- and so on

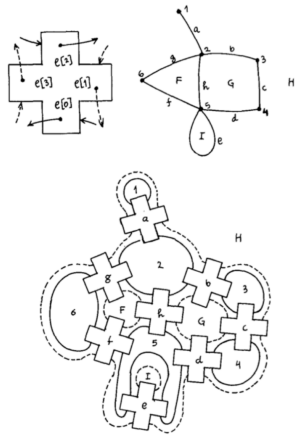
Emerging applications (e.g. medical 3D) require the convergence of data structures from:

- 3d computer imaging
- computer graphics
- solid modeling
- computer-aided geometric design
- discrete meshing of domains
- physical simulations

The goals of unification, scalability, and distributed computing call for rethinking some foundations of geometric and topological computing

## Quad-Edge data structure

(Guibas & Stolfi, ACM Transactions on Graphics, 1985)



(a) Edge record showing Next links.
(b) A subdivision of the sphere.
(c) Data structure for the subdivision

Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams

largely used in computational geometry algorithms and in geometric libraries

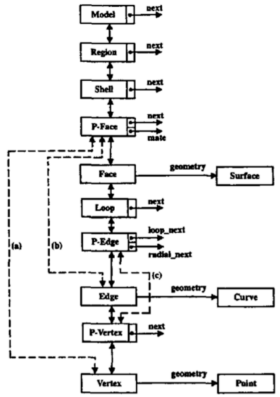




## Partial-Entity data structure

(Sang Hun Lee & Kunwoo Lee, ACM Solid Modeling, 2001)

Compact Non-Manifold Boundary Representation Based on Partial Topological Entities




```
class Entity {
  int        _id;
  Attribute  *_attribute;
};
class Model : public Entity {
  Model      *_next;        // next model
  Region     *_region;      // list of regions
};
class Region : public Entity {
  Region     *_next;        // link field of the region list of a model
  Model      *_model;       // parent model
  Shell      *_shell;       // peripheral shell
};
class Shell : public Entity {
  Shell      *_next;        // next void shell
  Region     *_region;      // parent region
  Pface      *_pface;       // partial face
};
class Pface : public Entity {   // partial face (p-face) class
  Pface      *_next;        // next p-face
  Shell      *_shell;       // parent shell
  Entity     *_child;       // child entity: a face, an edge, or a vert
  Orient     _orient;       // orientation flag w.r.t. the face normal
  Pface      *_mate;        // mate p-face
};
class Face : public Entity {
  Pface      *_pface;       // one of two incident p-faces
  Loop       *_loop;        // peripheral loop
  Surface    *_geometry;    // surface
};
```

```
class Loop : public Entity {
  Loop       *_next;        // next hole loop
  Face       *_face;        // parent face
  Pedge      *_pedge;       // a p-edge in a loop
};
class Pedge : public Entity {  // partial edge (p-edge) class
  Loop       *_loop;        // parent loop
  Entity     *_child;       // child entity: an edge or a p-vertex
  Orient     *_orient;      // orientation flag w.r.t. the edge direction
  Pvertex    *_vertex;      // start p-vertex
  Pedge      *_looped_prev; // previous p-edge in the loop cycle
  Pedge      *_looped_next; // next p-edge in the loop cycle
  Pedge      *_radial_prev; // previous p-edge in the radial cycle
  Pedge      *_radial_next; // next p-edge in the radial cycle
};
class Edge : public Entity {
  Entity     *_parent;      // parent entity: a p-edge or a p-face
  Pvertex    *_pvertex[2];  // two end p-vertices
  Curve      *_geometry;    // curve
};
class Pvertex : public Entity {  // partial vertex (p-vertex) class
  Pvertex    *_next;        // another p-vertex associated with _vertex
  Entity     *_parent;      // parent entity: an edge or a p-edge
  Vertex     *_vertex;      // mother vertex
};
class Vertex : public Entity {
  Entity     *_parent;      // parent entity: a p-vertex or a p-face
  Point      *_geometry;    // position
};
```

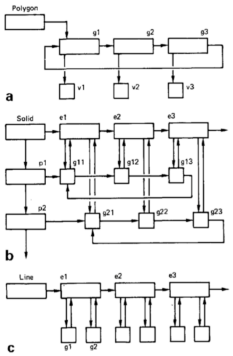




## Hybrid Edge data structure

(Kalay, Computer-Aided Design, 1989)

a topological data structure for vertically integrated geometric modelling



```
type   vptr   = ^vertex;
       gptr   = ^segment;
       pptr   = ^polygon;
       eptr   = ^edge;
       sptr   = ^solid;
       lptr   = ^lines;

solid  = record
              snext     : sptr;
              elist     : eptr;
              plist     : pptr;
              case cavity : boolean
                 false : (child  : sptr)
                 true  : (parent : sptr)
           end;

lines  = record
              lnext     : lptr;
              edge      : eptr;
           end;
```

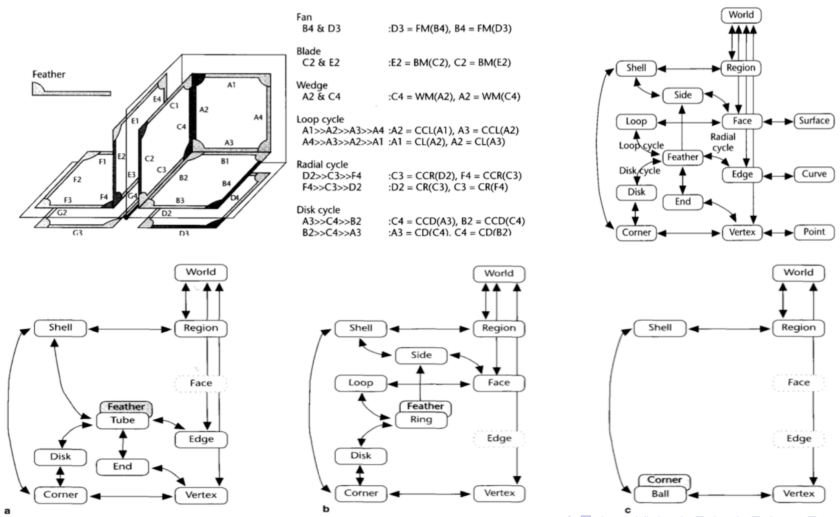
```
vertex  = record
              form      : point
           end;

segment = record
              gnext     : gptr;
              form      : line;
              gpoly     : pptr;
              gedge     : eptr;
              gvert     : vptr
           end;

polygon = record
              pnext     : pptr;
              glist     : gptr;
              form      : plane;
              case hole : boolean of
                 false  : (child  : pptr);
                 true   : (parent : pptr)
           end;

edge    = record
              enext     : eptr;
              eseg      : array [1..2] of gptr;
              esolid    : sptr
           end;
```





## Coupling Entities data structure

(Yamaguchi & Kimura, IEEE Computer Graphics and Applications, 1995)

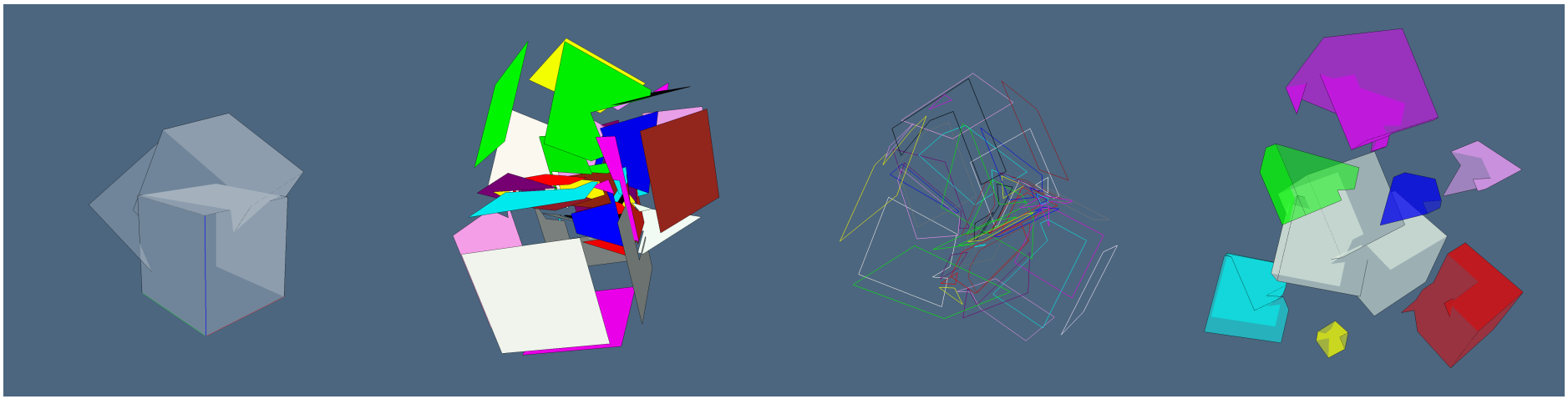


# Solid modeling with sparse arrays

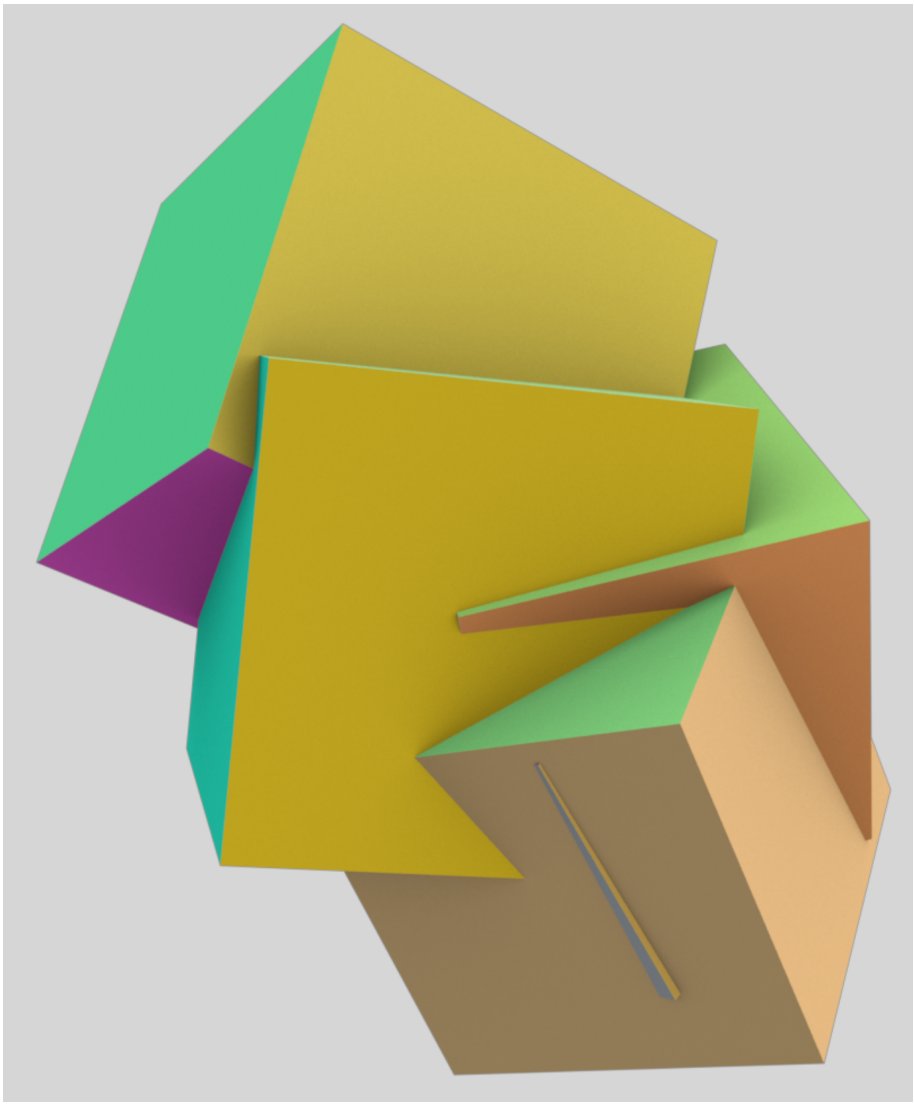
## Chain Complex

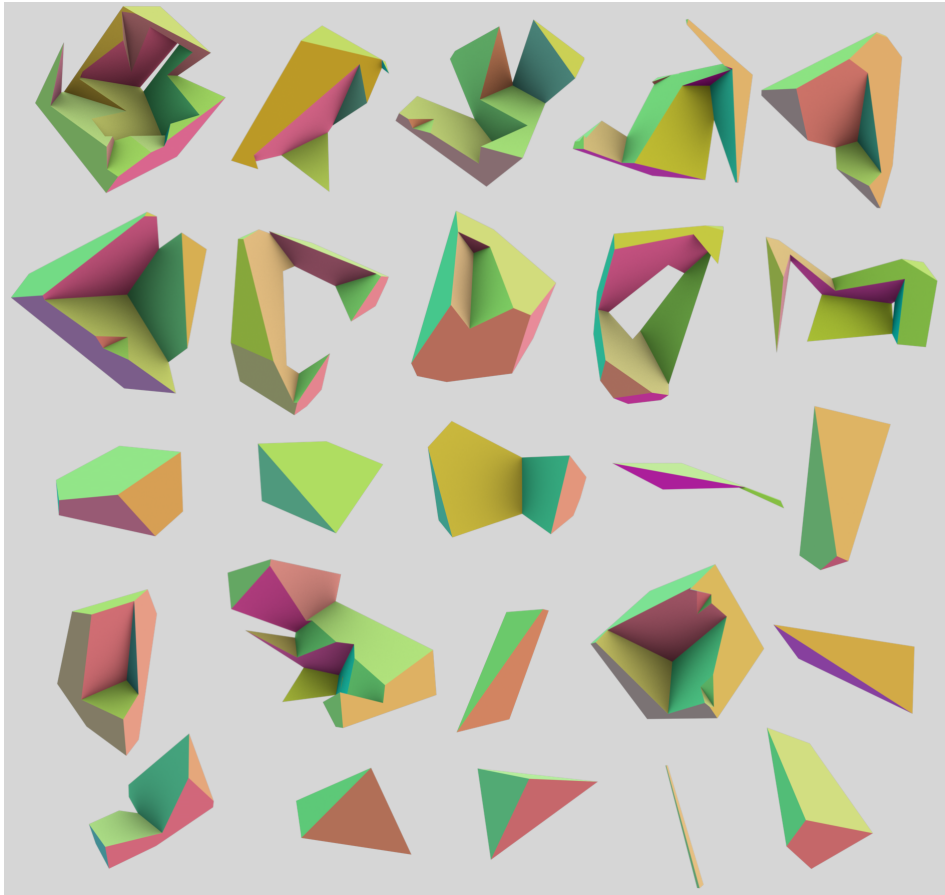
$$C_\bullet = (C_p, \partial_p) := C_3 \underset{\partial_3}{\overset{\delta^2}{\leftrightarrows}} C_2 \underset{\partial_2}{\overset{\delta^1}{\leftrightarrows}} C_1 \underset{\partial_1}{\overset{\delta^0}{\leftrightarrows}} C_0 \qquad \partial_p^\top = \delta^{p-1}$$

```
Int8[
-1 0 1 0 0 0 1 0 -1 0 -1 0 0 0 -1 1 0 1 0 0 -1 1 0 1 -1 0 0 0 -1 -1 0 1 0 0 -1 0 1 0 0 0 1 -1 0 -1 0 0 1;
 0 -1 0 0 0 1 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 -1 0 0 0 0 0 0 1 0 -1 0 0 0 -1 1 0 1 0 0 -1;
 0 0 0 -1 0 0 0 -1 0 0 0 1 1 0 0 0 -1 0 0 1 0 0 -1 0 0 0 1 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 1 0 0;
 0 0 0 0 -1 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 -1 0 0 0 0 0 -1 0 0;
 1 0 -1 0 0 0 -1 0 1 0 1 0 0 0 1 -1 0 -1 0 -1 0 0 1 0 0 0 -1 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 -1 0 0 0 0;
 0 0 0 1 0 0 0 1 0 0 0 -1 -1 0 0 0 1 0 0 0 1 -1 0 -1 1 0 0 0 1 1 0 -1 0 0 0 -1 0 0 0 1 0 0 0 0 0 1 0;
 0 1 0 0 0 -1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 1 0 0 0 0 -1 0 0 0 0 1 0 0 0 0;
 0 0 0 0 1 0 0 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 -1 0 0 0 0 0 -1 0]
```



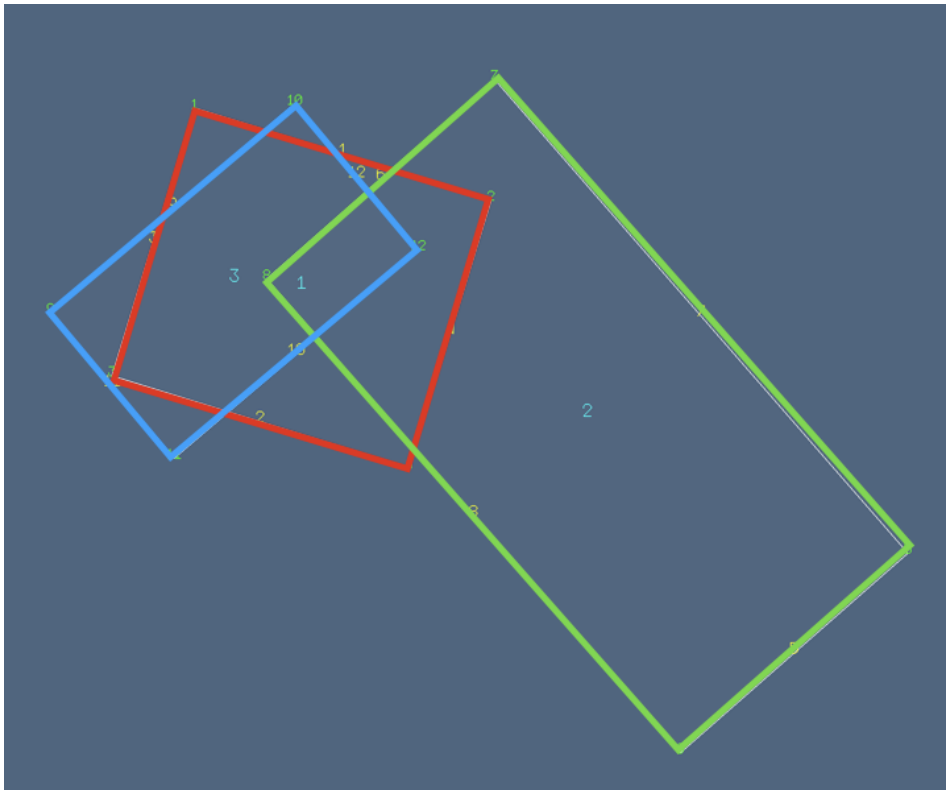
# Integrating boundary representations and cell complexes





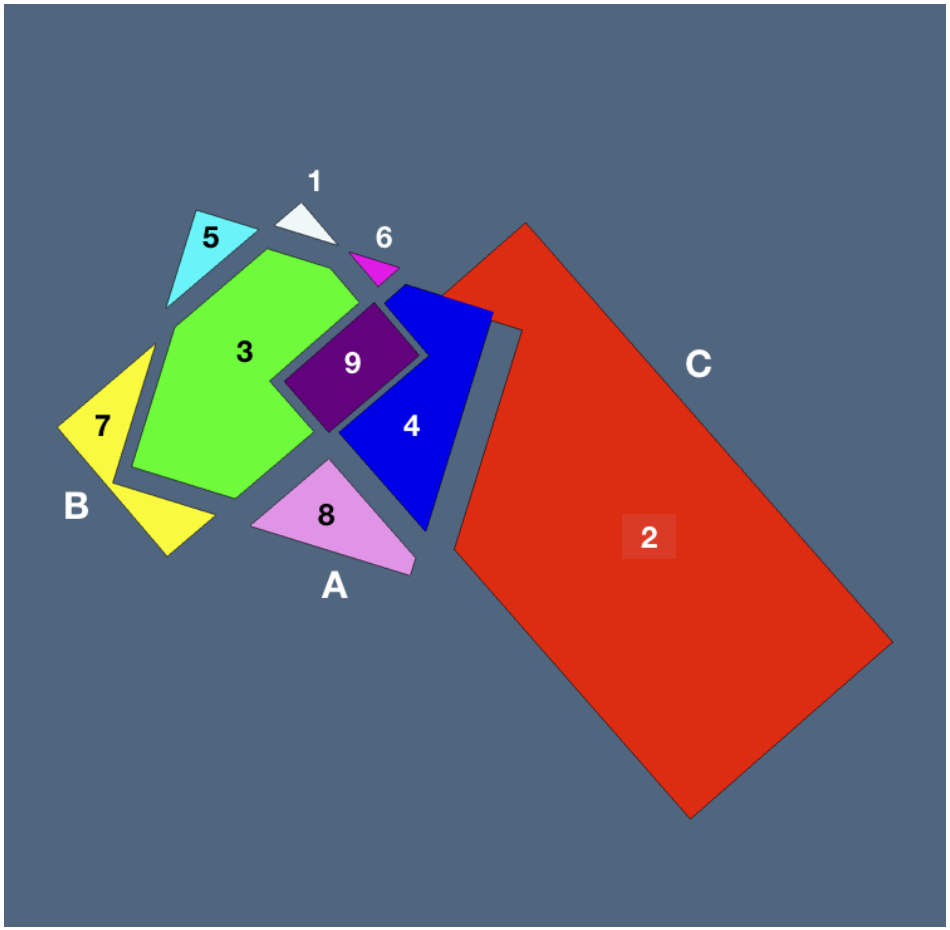
Topological computing of arrangements with (co)chains. ACM Transactions on Spatial Algorithms and Systems, August 2017. Submitted.

# Work-in-progress Boolean Operations (1/3)



$EE' =$

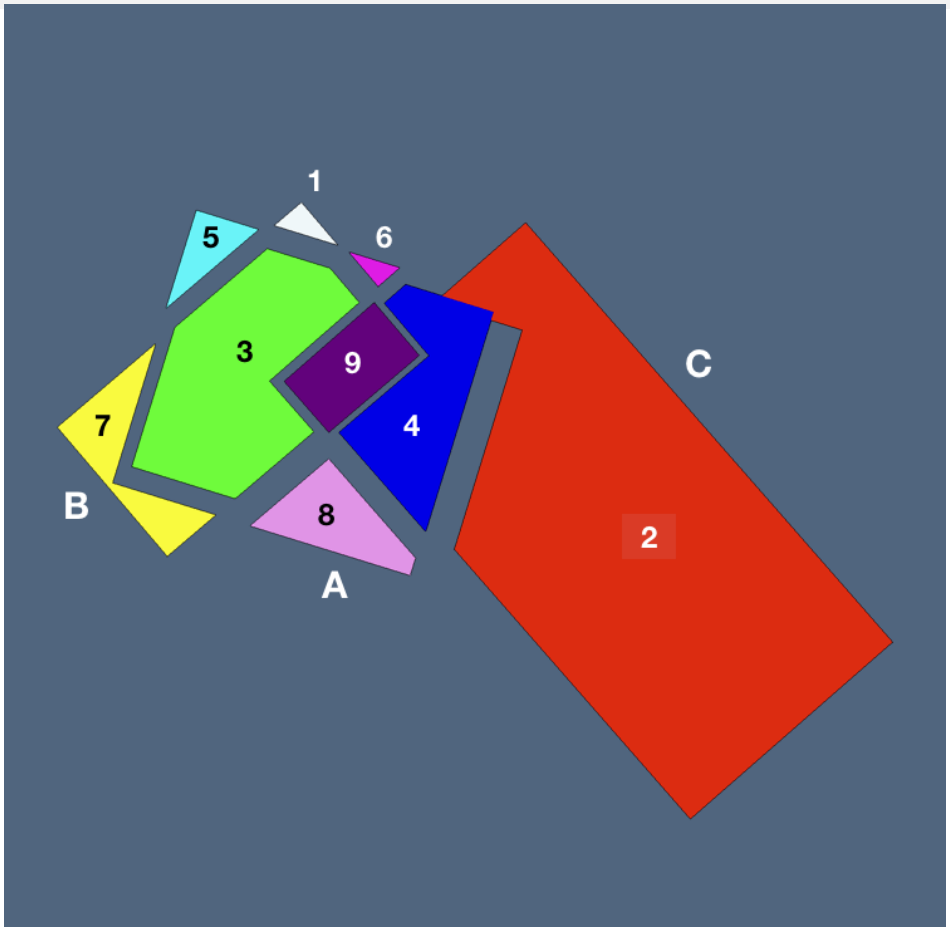
```
[[1,2,3,4],[5,6],[7,8],[9,10],[11],[12,13,14],
[15],[16,17,18],[19,20,21],[22,23,24],[25],[26,27,28]]}
```




$F'E' =$

```
[[2,21,26],[3,9,16,11,15,12],[2,20,8,5,23,18,14,27],
[4,13,28,24,17,9],[1,7,20],[3,27,13],[5,8,19,25,22],
[6,10,17,23],[14,18,24,28]]
```

# Work-in-progress




```
julia> Matrix(copFF)
3×9 Array{Int8,2}:
 0  0  1  1  1  1  0  1  1
 0  1  0  1  0  0  0  0  1
 1  0  1  0  0  0  1  0  1
```

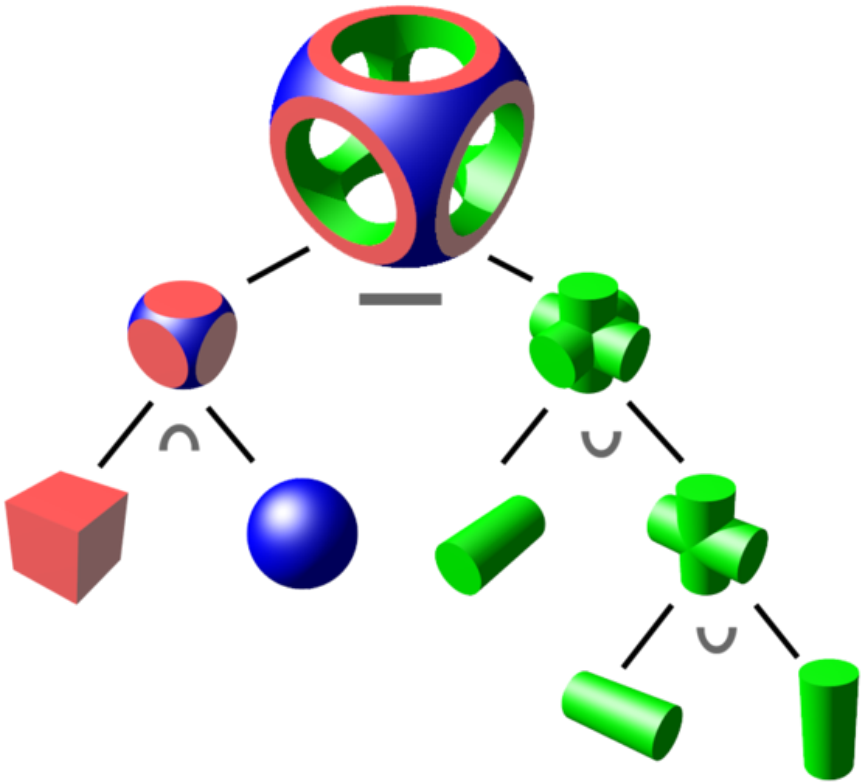
# Boolean Operations (2/3)

solids as characteristic functions (3-chains) over 3-cells

| Solid | 3-chain |
|---|---|
| $S_i$ | 1 2 3 4 5 6 7 8 9 |
| $A$ | 0 0 1 1 1 1 0 1 1 |
| $B$ | 0 1 0 1 0 0 0 0 1 |
| $C$ | 1 0 1 0 0 0 1 0 1 |
| $A \cup B$ | 0 1 1 1 1 1 0 1 1 |
| $A \cup B \cup C$ | 1 1 1 1 1 1 1 1 1 |
| $A \cap B \cap C$ | 0 0 0 0 0 0 0 0 1 |
| $A - B - C$ | 0 0 0 0 1 1 0 1 0 |

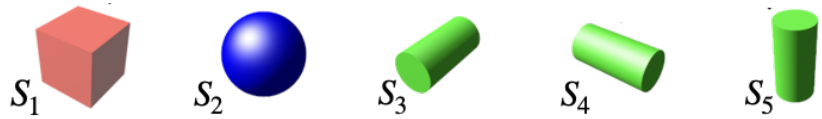
Boolean formulas distribute operators over columns of characteristic matrix

# Work-in-progress Constructive Solid Geometry revisited



CSG-tree

$\mathbb{E}^3$ arrangement induced by $\{S_i\}$




$$C_3 \underset{\partial_3}{\overset{\delta^2}{\leftrightarrows}} C_2 \underset{\partial_2}{\overset{\delta^1}{\leftrightarrows}} C_1 \underset{\partial_1}{\overset{\delta^0}{\leftrightarrows}} C_0$$

$$\chi : \{S_1, \ldots, S_5\} \to C_3$$

$$X := [\chi(S_1); \ldots; \chi(S_4); \chi(S_5)]$$

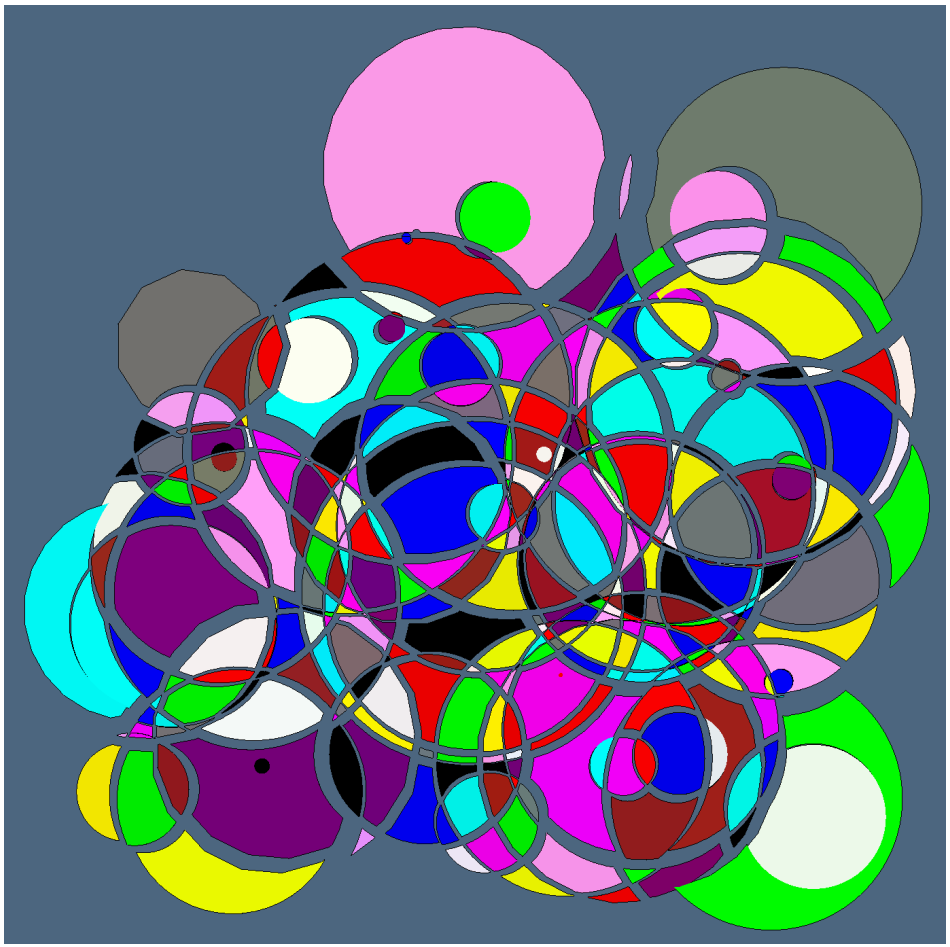
$$X_j = X[:, j]$$

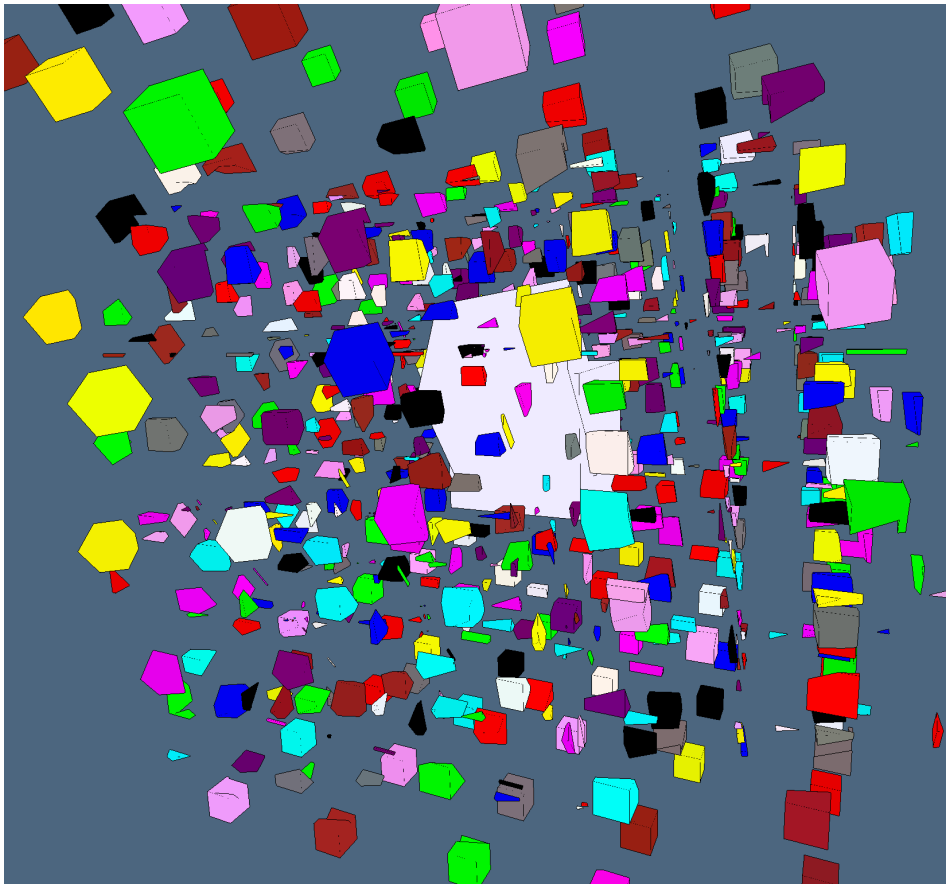
$$((X_1) \cap (X_2)) - ((X_3) \cup ((X_4) \cup (X_5)))$$

binary formula evaluation

# Work-in-progress 2D/3D Viewer

Julia's OpenGL interactive visualization

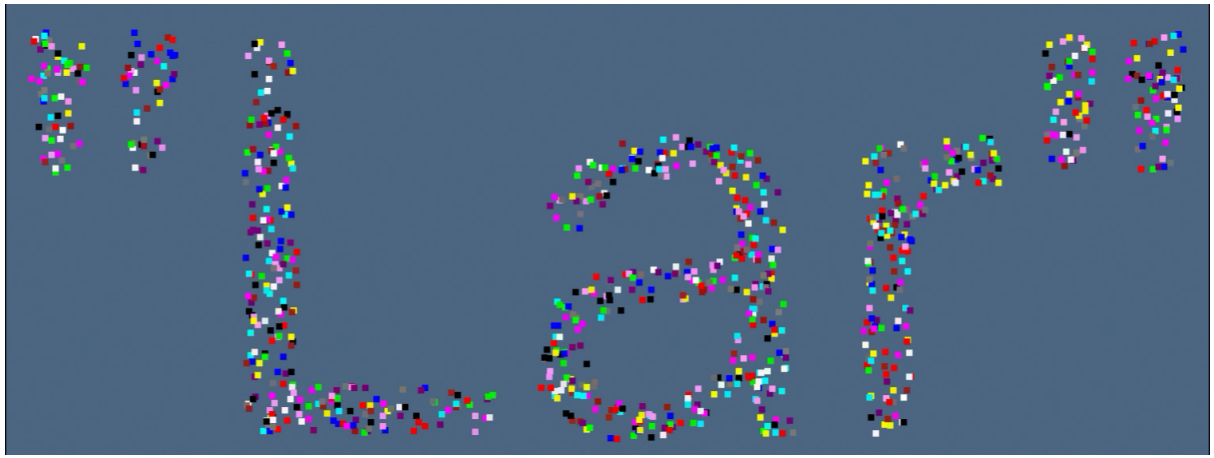




https://github.com/cvdlab/ViewerGL.jl

# Work-in-progress Computational Geometry

Multidimensional Delaunay Triangulation & Alpha-Complex



https://github.com/eOnofri04/AlphaStructures.jl

# Work-in-progress Computational Geometric Design

1. Visualization – https://github.com/cvdlab/ViewerGL.jl
2. Hierarchical assemblies – https://github.com/cvdlab/LAR.jl [1]
3. Arrangement & Boolean ops – https://github.com/cvdlab/LAR.jl
4. Integration of polynomials – https://github.com/cvdlab/LAR.jl
5. CDT & 2D triangulations – https://github.com/cvdlab/Triangle.jl
6. *n*-dim Delaunay & Alpha-shapes – work-in-progress
7. Parametric Splines (curves and surfaces) – to be ported
8. Hierarchical point clouds – starting work

[1] LAR = LinearAlgebraicRepresentation

# Status of packages

# `ViewerGL.jl` (native OpenGL)

Native interactive visualization of 2D/3D geometric structures

API to be defined

- based on multiplatform `GLFW.jl` and `Modern.jl` packages
- basic OpenGL primitives (points, lines, triangles, quads)
- high level LAR primitives based on sparse arrays

(meshes, grids, assemblies, curves, surfaces, solids, graphs, texts)

- textures support (to be added in September 2019)

https://github.com/cvdlab/ . . . /all-examples.jl

# LinearAlgebraicRepresentation.jl (LAR)

Sparse matrices for Computational Geometric Design

Work-in-progress

- topological operators (cell & chain complex, product, boundary, coboundary)
- geometrical operators (affine ops, assembly, Boolean ops)
- reconstruction of shape from point-clouds (*alpha*-shapes, podtrees)
- support for (co)chain-based simulation (DEC and Cell Method (CM))

https://github.com/cvdlab/ . . . /all-examples-2d.jl
https://github.com/cvdlab/ . . . /all-examples-3d.jl

# `AlphaStructures.jl` (*n*-Delaunay & $\alpha$ – *shapes*)

Multidimensional simplicial complexes

Work-in-progress

- Delaunay triangulations (2D, 3D, . . . )
- *n*-dim Alpha-complexes ($\alpha$-sampling, stratification)
- topological operations (product, boundary, homology)

https://github.com/cvdlab/ . . . /all-examples-2d.jl

# Other packages

## Work-in-progress

`cvdlab/Triangle.jl`

Porting to Julia of Triangle library by J.R.~Shewchuk.
https://www.cs.cmu.edu/~quake/triangle.html

`cvdlab/NURBS.jl`

Porting to Julia of NURBS software from book:
Rogers D.F. Introduction to NURBS. With Historical Perspective, Morgan Kaufmann, 2001.