Computational Algebraic Topology: Lecture 7

Alberto Paoluzzi

March 28, 2017

Cellular complexes – LAR (Linear Algebraic Representation)

1. Geometric language PLaSM
2. PLaSM examples
3. Design goals of LAR scheme
4. Some LAR algorithms

Geometric language PLaSM

PLaSM and LAR history

- PLaSM (CAD 1993, ACM TOG 1995; Wiley, 2003) stands for Programming Language for Solid Modeling
- geometry-oriented extension of Backus' FL language (Backus, 1978; Backus et al., 1989).
- developed in the '90s in the framework of "PF Edilizia" of CNR
- `pyplasm` (2006-) is C++ porting to Python via SWIG wrapping.
- `larlib` (2012), using the LAR scheme, on top of `Scipy/Pyplasm` stack
- support of topological queries and physical properties of meshes (complexes), via integration of polynomials over the boundary of any chain of cells
- interactive visualization via the pyplasm viewer, based on OpenGL
- porting of `larlib` to `Julia` (2016-), last-generation programming language for scientific computing (Bezanson et al., 2014)

PLaSM examples

TOP-DOWN ARCHEOLOGY WITH A GEOMETRIC LANGUAGE

D. LUCIA, F. MARTIRE, A. PAOLUZZI, & G. SCORZELLI
ROMA TRE UNIVERSITY, ITALY

DOMUS FLAVIA AND **AUGUSTANA**, BUILT BY FLAVIANS (VESPASIAN, TITUS AND DOMITIAN) ON THE PALATINE HILL IN ROME, AND DOMITIAN'S HIPPODROMUS.

THE DOMUS FLAVIA WAS THE OFFICIAL PALACE, WHILE THE DOMUS AUGUSTANA WAS THE EMPEROR'S PRIVATE RESIDENCE.

PLASM LANGUAGE

- PLASM, (the Programming LAnguage for Solid Modeling) is a **design language**, strongly influenced by Backus' FL (programming at Function Level)
- *Computer Aided Design*, 1993
- *ACM Transactions on Graphics*, 1995
- *Computer-Aided Design & Applications*, 2005
- PROTO-PLASM, *Phil. Trans. of the Royal Society A*, Computational Framework for the Visual Physiological Human, 2008

AGENDA

- HISTORICAL NOTES
- **LANGUAGE SYNTAX & SEMANTICS**
- GEOMETRY ENGINE
- DEFINING NEW PRIMITIVES
- RECONSTRUCTION OF THE FLAVIAN PALACE

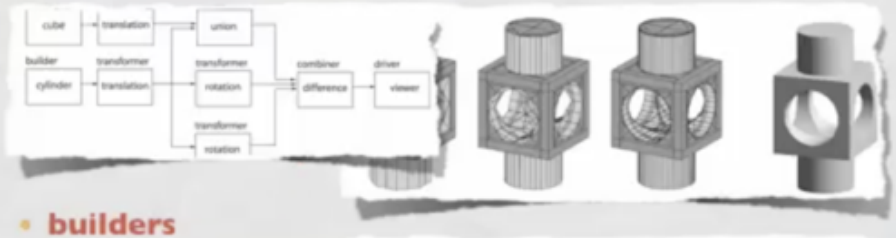
THREE PARADIGMS

1. APPLICATION

2. COMPOSITION

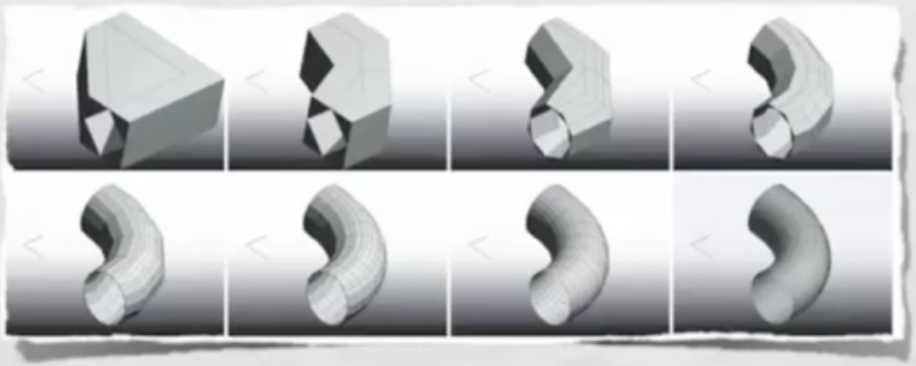
3. CONSTRUCTION

PROGRESSIVE & DISTRIBUTED



- builders
- transformers
- combiners
- drivers

EXAMPLE



Progressive refinement of a portion of biquadratic rational B-spline through a dataflow of BSP nodes

AGENDA

- HISTORICAL NOTES
- LANGUAGE SYNTAX
- GEOMETRY ENGINE
- **DEFINING NEW PRIMITIVES**
- RECONSTRUCTION OF THE FLAVIAN PALACE

COLUMNA

Our examples of specialized generative primitives start with the Columna function, that assemblies a parametric column in pseudo-Corinthian style. The *formal parameters*, whose scope covers the function *body*, i.e. the generating expression on right-hand side of the function *head* as well as the local environment delimited by the WHERE, END keyword pair, stand respectively for:

- dm is the circumference diameter at the column basis;
- h is the column height;
- h_base is the the hight of the column base.

Listing 2: Definition of a column.

```
DEF columna (dm,h,h_base::isReal) = base TOP
    torus_bot TOP cylndr TOP torus_top TOP capital
    TOP base_top
WHERE
    cylndr = (JOIN~TRUNCONE:<dm/2,0.8*dm/2,h>):24,
    torus_bot = (JOIN~TORUS:<dm/12,d/2>):<8,24>,
    torus_top = (JOIN~TORUS:<0.8*(dm/12),0.8*(dm/
        2)>):<8,24>,
    base = (T:<1,2>:<7*dm/-12, 7*dm/-12> ~ CUBOID)
        :<7*dm/6, 7*dm/6, h_base>,
    base_top = (T:<1,2>:<7*dm/-12, 7*dm/-12> ~
        CUBOID):<7*dm/6, 7*dm/6, dm/6>,
    capital = (JOIN ~ TRUNCONE:< 0.8*dm/2, 1.2*dm/
        2, h/8>):4 + (R:<1,2>:(PI/4) ~ JOIN ~
        TRUNCONE:< 0.8*dm/2,1.2*dm/2,h/8>):4
END;
```

GROIN VAULTS

Listing 9: Definition of a generative function of groin vaults with assigned length angle of join and width.

```
DEF GroinVault (length,w,angle::isreal) =
    (T:3:(-:ceiling) ~ STRUCT ~ [id,S:2:-1]):
        vaultHalf
WHERE
    radius = length/(2 * cos:angle),
    ceiling = MIN:3:Vault_8th,
    solidMap = Bezier:S3:<surf_0,surf_1>,
    surf_0 = K:(length/2) AL archSurf2D,
    surf_1 = [s1,s1,s2]:archSurf2D,
    archSurf2D = archSurface:<radius,w>,
    vault8th = MAP:solidMap:domain3D,
    domain3D = (T:1:(angle) ~ intervals:(PI/2 -
        angle)):8 * q:1 * q:1,
    vault4th = (STRUCT ~ [id,MAP:[s2,s1,s3]]):
        vault8th,
    vaultHalf = (STRUCT ~ [id,S:1:-1]):vault4th
END;
```

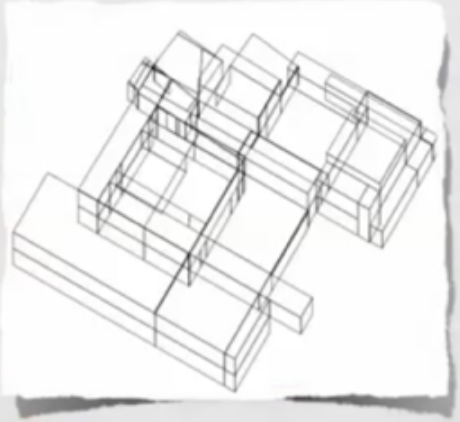
AGENDA

- HISTORICAL NOTES
- LANGUAGE SYNTAX
- GEOMETRY ENGINE
- DEFINING NEW PRIMITIVES
- **RECONSTRUCTION OF THE FLAVIAN PALACE**

TOP-DOWN SITE ANALYSIS (2/3)

naming of single spaces in each zone

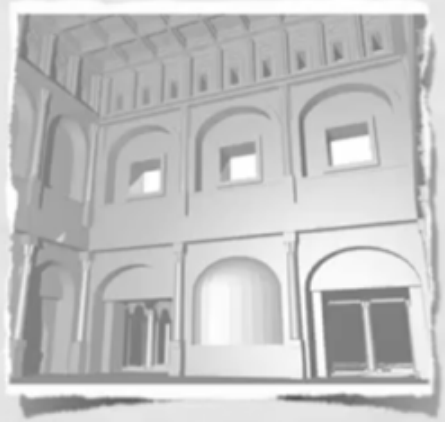
TOP-DOWN SITE ANALYSIS (3/3)



Initial volumetric studies for site reconstruction

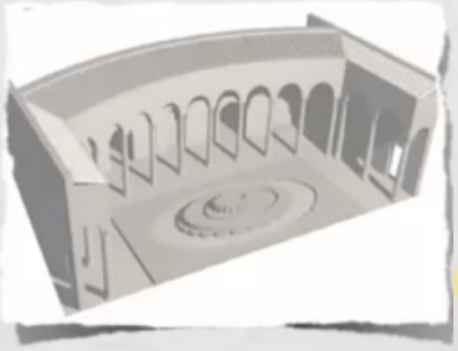
DETAILED GEOMETRY

INTERIOR PERISTYLIUM

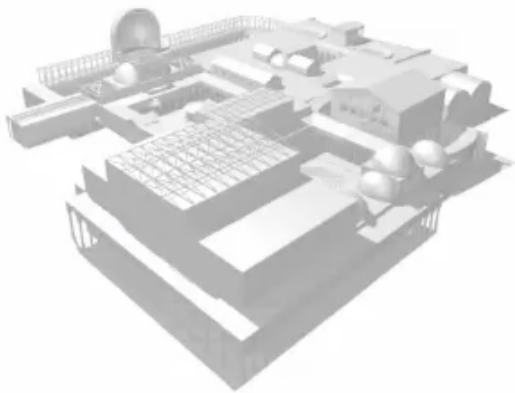


AULA REGIA

VOLUME MAPS



Volume maps used to adapt the "pure" shapes to the site configuration



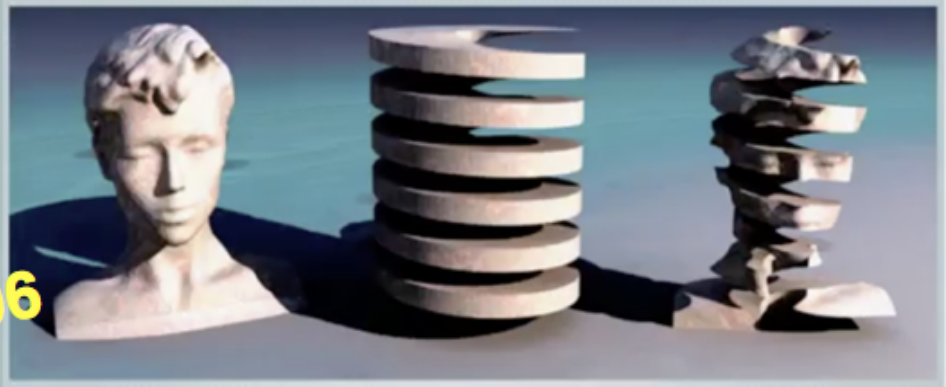
Progressive Conversion from B-rep to BSP for Streaming Geometric Modeling

Chandrajit Bajaj*, Alberto Paoluzzi^ & Giorgio Scorzelli^

*ICES and Computer Sciences Dept., Univ. of Texas at Austin,
bajaj@cs.utexas.edu
^Dip. Informatica e Automazione, "Roma Tre",
{paoluzzi,...@dia.uniroma3.it

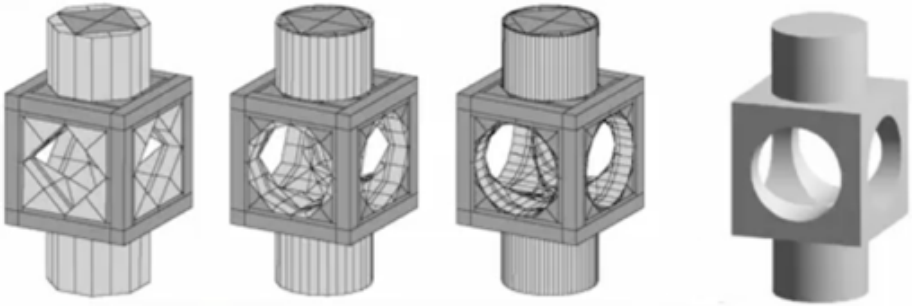
Best Paper Award — CAD 2006

Progressive conversion from triangles to solid BSP (CAD'06)



Streaming Boolean ops between BSP trees (ACM SM'04)

Progressive geometric streaming



- Builders
- Transformers
- Combiners
- Drivers

Progressive geometric streaming

"Embarrassingly parallel" modeling (IJCGA, 2006)

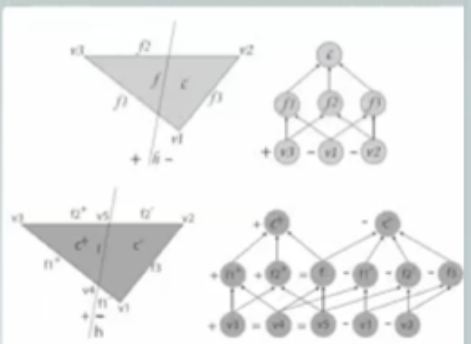
Algorithm's rationale

Inertia-based Algorithm: B-rep →BSP

- Split in half the **surface mass** in each subtree
- Confine the triangles in a FUZZY cell with the **minimum parallelepiped** parallel to the principal axes
- Split the FUZZY cell with a **principal plane** through the **centroid**
- **Finalize** the cell splitting using the **triangle planes**

Progressive geometric streaming

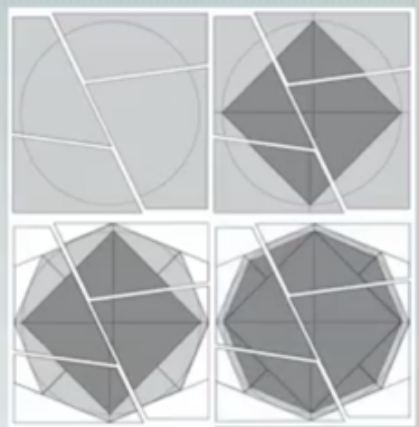
- **cellular complex** induced by PBSP tree
- **geometry** stored in lattice's Hasse diagram



Split algorithm (ACM CG'96)

Progressive geometric streaming

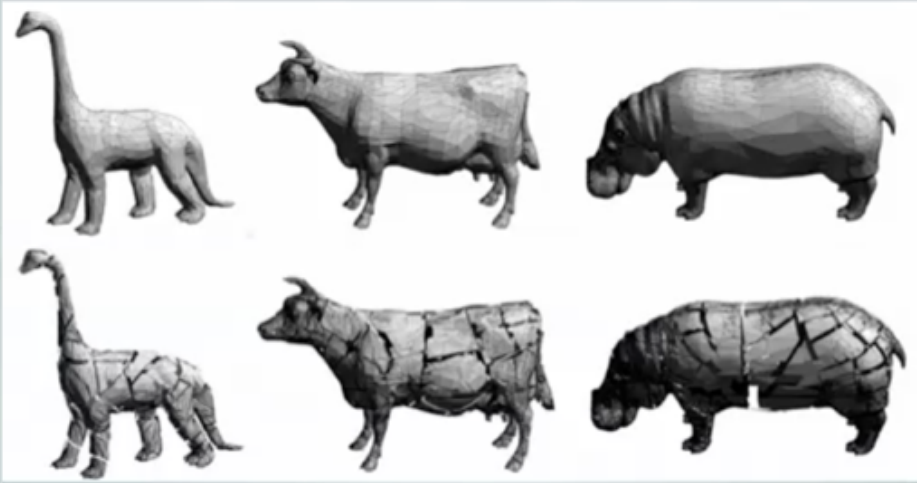
- Based on PBSP trees (**Progressive** Binary Space Partition)
- **Three** kind of leaves:
  - out (empty)
  - in (full)
  - **fuzzy** (undecided)



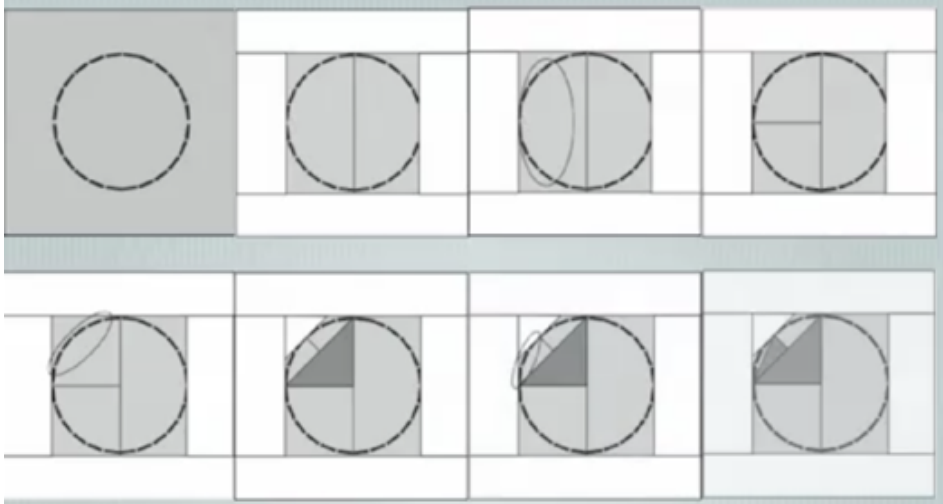
Fast inertia computation

- FAST computation of inertia via **affinely extended Euler tensor** (CAD J., 2006)
- NEW method, that computes and transforms the Euler tensor (strictly related to inertia) under general affine maps, through addition and multiplication of 4×4 matrices

Examples



Inertia-based algorithm: B-rep ⟶ BSP



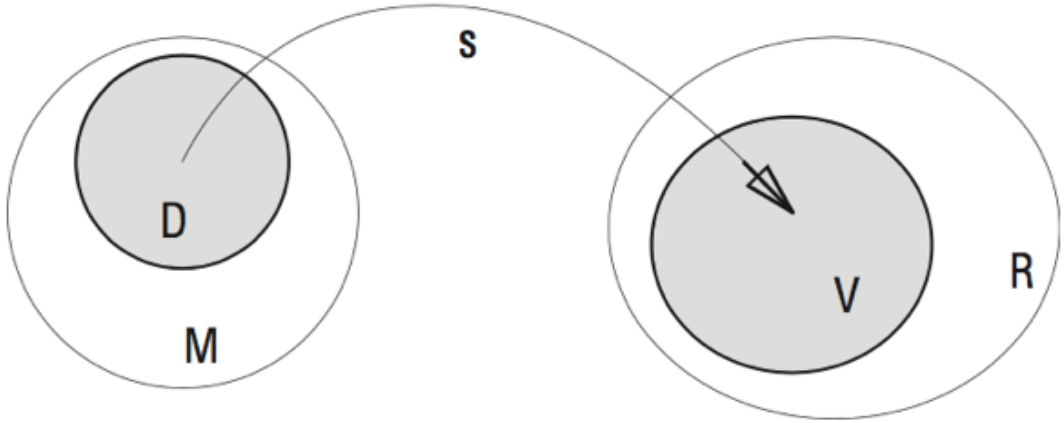
Examples and performance

Real-time adaptive refinement

Design goals of LAR scheme

Representation scheme

mapping $s : M \to R$ from a space of math models M to computer representations R



1. The set M contains the mathematical models of the class of solid objects that the scheme aims to represent.
2. The set R contains symbolic representations, i.e. suitable data structures, built according to some appropriate computer grammar.

Representation schemes (a long list)

Most of such papers introduce or discuss one or more representation schemes ...

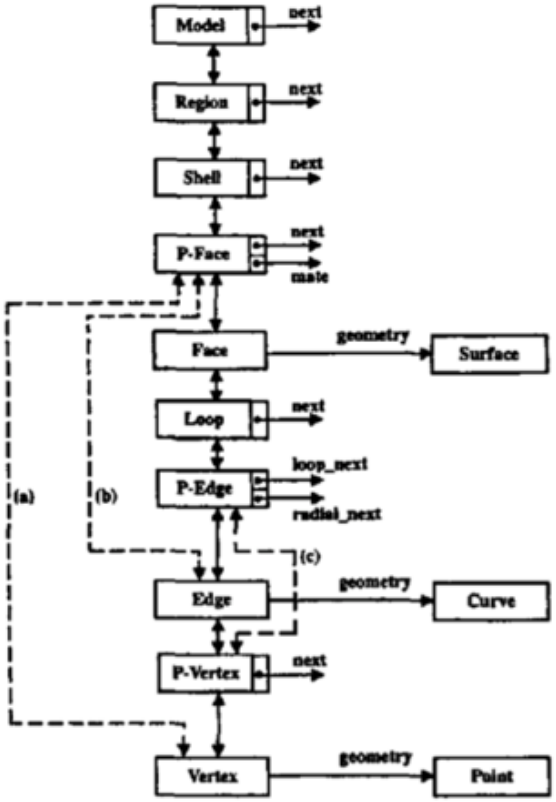
1. Requicha, ACM Comput. Surv., 1980 [Req80]
2. Requicha & Voelcker, PEP TM-25, 1977, [RV77]
3. Rossignac & Requicha, Comput. Aided Des., 1991, [RR91]
4. Bowyer, SVLIS, 1994, [Bow95]
5. Baumgart, Stan-CS-320, 1972, [Bau72]
6. Braid, Commun. ACM, 1975, [Bra75]
7. Dobkin & Laszlo, ACM SCG, 1987, [DL87]
8. Guibas & Stolfi, ACM Trans. Graph., 1985, [GS85]
9. Woo, IEEE Comp. Graph. & Appl., 1985, [Woo85]
10. Yamaguchi & Kimura, Comp. Graph. & Appl., 1995, [YK95]
11. Gursoz & Choi & Prinz, Geom.Mod., 1990, [WTP90]
12. S.S.Lee & K.Lee, ACM SMA, 2001, [LL01]
13. Rossignac & O'Connor, IFIP WG 5.2, 1988, [RO90]
14. Weiler, IEEE Comp. Graph. & Appl., 1985, [Wei85]
15. Silva, Rochester, PEP TM-36, 1981, [Sil81]
16. Shapiro, Cornell Ph.D Th., 1991, [Sha91]
17. Paoluzzi et al., ACM Trans. Graph., 1993, [PBCF93]
18. Pratt & Anderson, ICAP, 1994, [PA94]
19. Bowyer, Djinn, 1995, [BS95]
20. Gomes et al., ACM SMA, 1999, [GMR99]
21. Raghothama & Shapiro, ACM Trans. Graph., 1998, [RS98]
22. Shapiro & Vossler, ACM SMA, 1995, [SV95]
23. Hoffmann & Kim, Comput. Aided Des., 2001, [HK01]
24. Raghothama & Shapiro, ACM SMA, 1999, [RS99]
25. DiCarlo et al., IEEE TASE, 2008, [DMPS09]
26. Bajaj et al., CAD&A, 2006, [BPS06]
27. Pascucci et al., ACM SMA, 1995, [PFP95]
28. Paoluzzi et al., ACM Trans. Graph., 1995, [PPV95]
29. Paoluzzi et al., Comput. Aided Des., 1989, [PRS89]
30. Ala, IEEE Comput. Graph. Appl., 1992, [Ala92]

and much more ...

Representation scheme: Partial-Entity data structure

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

Compact Non-Manifold Boundary Representation Based on Partial Topological Entities

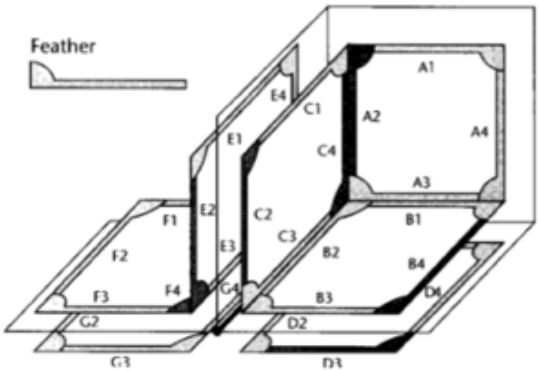



```
class Entity {
   int        _id;
   Attribute  *_attribute;
};
class Model : public Entity {
   Model      *_next;          // next model
   Region     *_region;        // list of regions
};
class Region : public Entity {
   Region     *_next;          // link field of the region list of a model
   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    *_pvertex;       // 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
};
```

Representation scheme: Coupling Entities data structure

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



Fan
B4 & D3 :D3 = FM(B4), B4 = FM(D3)

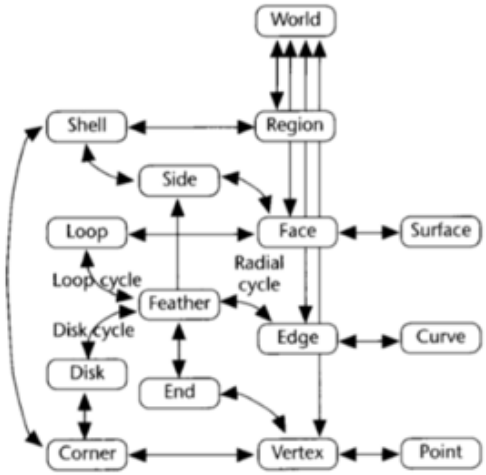
Blade
C2 & E2 :E2 = BM(C2), C2 = BM(E2)

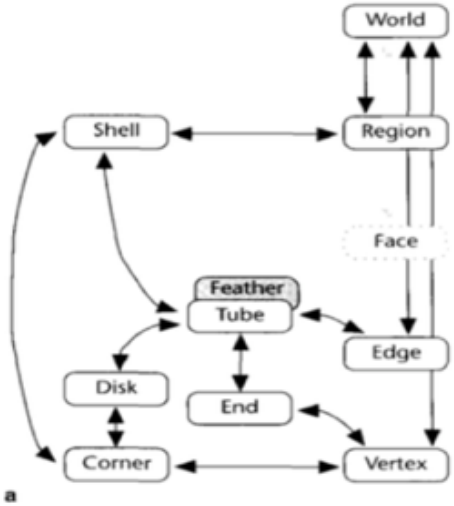
Wedge
A2 & C4 :C4 = WM(A2), A2 = WM(C4)

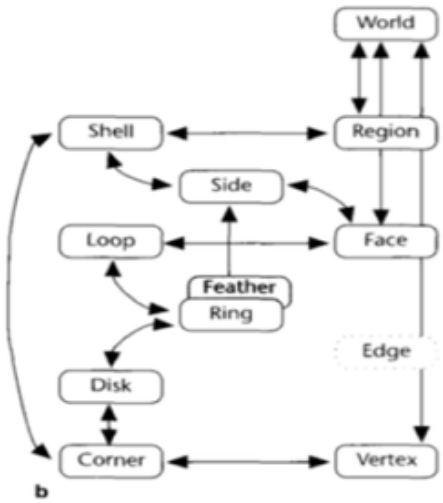
Loop cycle
A1>>A2>>A3>>A4 :A2 = CCL(A1), A3 = CCL(A2)
A4>>A3>>A2>>A1 :A1 = CL(A2), A2 = CL(A3)

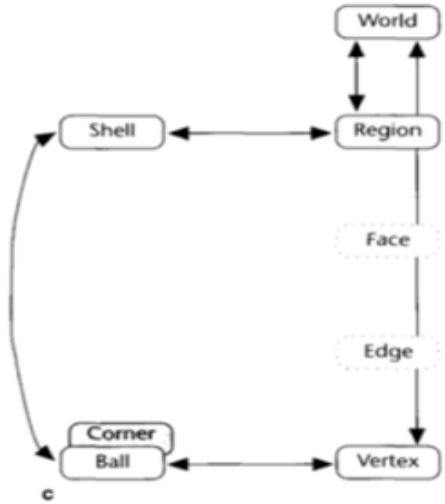
Radial cycle
D2>>C3>>F4 :C3 = CCR(D2), F4 = CCR(C3)
F4>>C3>>D2 :D2 = CR(C3), C3 = CR(F4)

Disk cycle
A3>>C4>>B2 :C4 = CCD(A3), B2 = CCD(C4)
B2>>C4>>A3 :A3 = CD(C4), C4 = CD(B2)



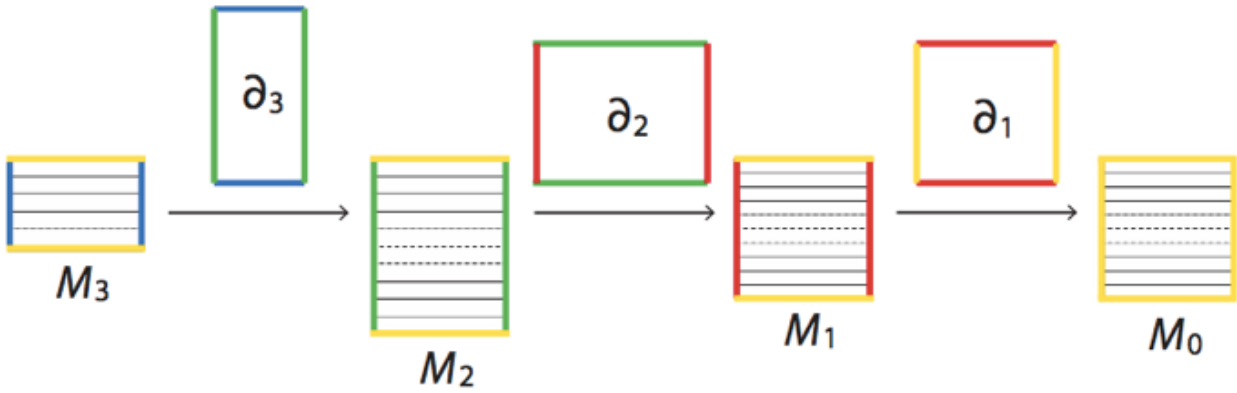


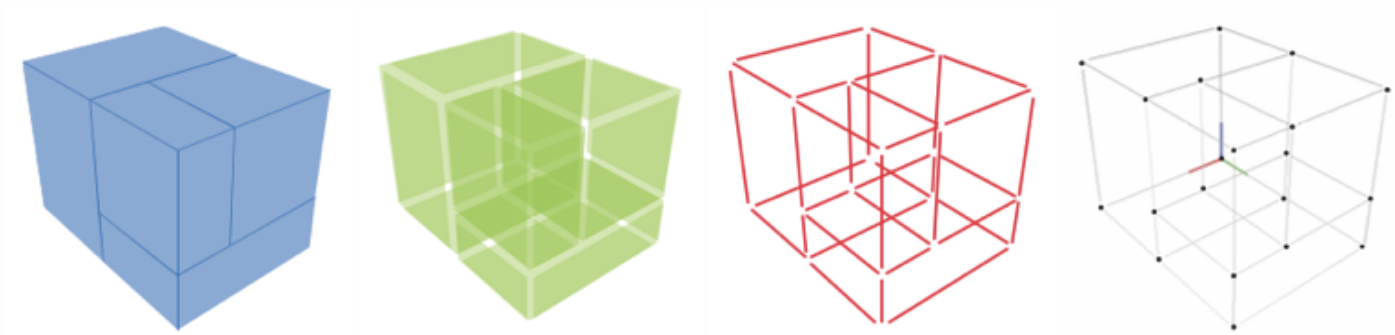




Linear Algebraic Representation (LAR)

(DiCarlo, Paoluzzi & Shapiro, Computer-Aided Design, 2013)





Rethinking some foundations

dealing with Big Data and scalable architectures **Google's map-reduce**

Emerging applications (e.g. space, nano & bio technology, medical 3D) require the convergence of shape synthesis and analysis from:

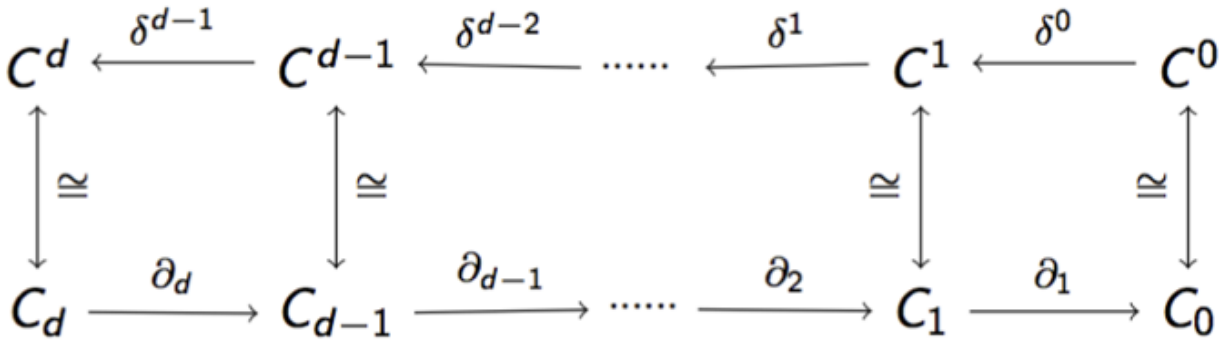
- computer imaging
- computer graphics
- computer-aided geometric design
- discrete meshing of domains
- physical simulations

The goals of unification, scalability, and massively parallel distributed computing

call for **rethinking the foundations** of geometric and topological computing

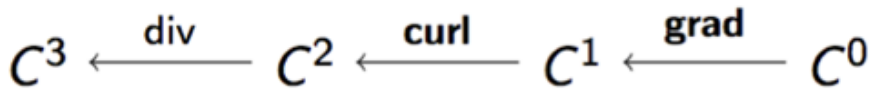
GOAL: 10^3 **times faster and** 10^4 **times bigger**

cochains (all maps, discrete fields) and coboundary maps (δ^d operators)



chains (linear spaces of model subsets) and boundary maps (∂_d operators)

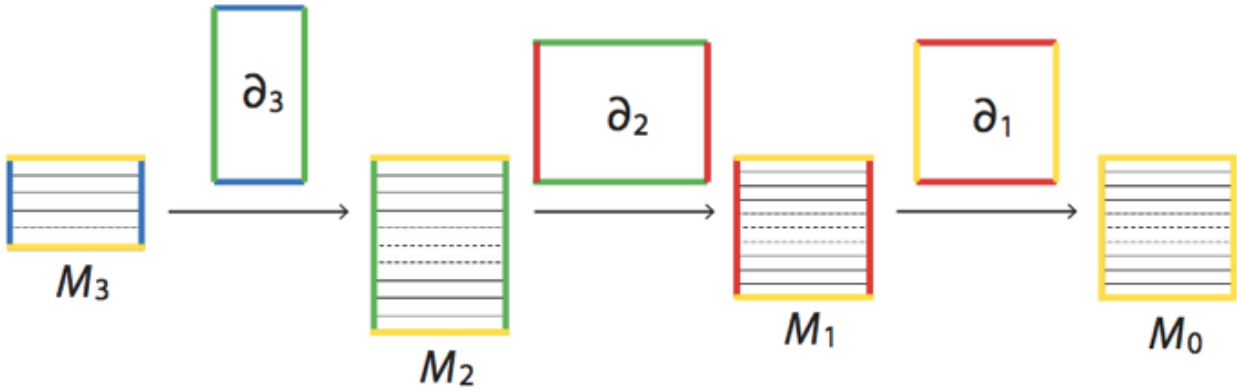
$$\boxed{\delta^p = \partial_{p+1}^\top} \qquad \boxed{\Delta_p = (\delta^p)^\top \delta^p + (\delta^{p-1})^\top \delta^{p-1}}$$

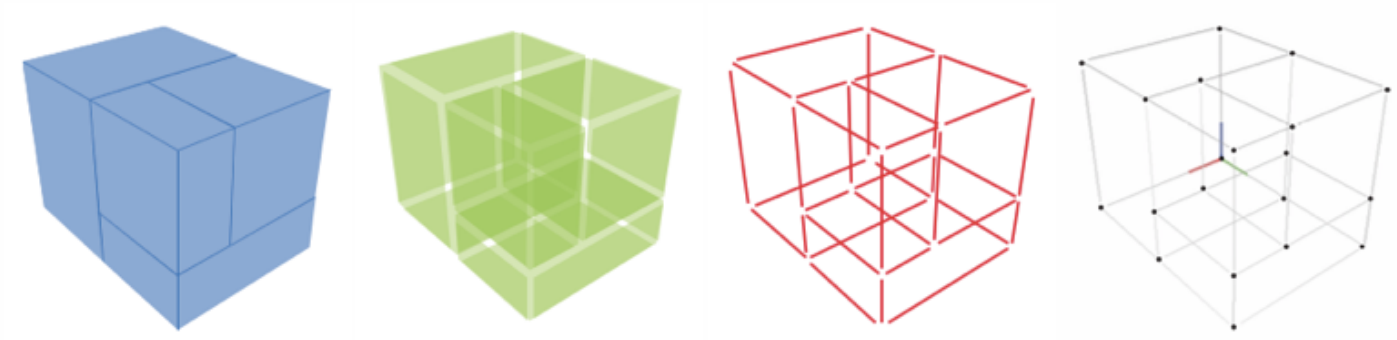


Chain complex (of chain spaces)

Sequence of linear spaces (over $\mathbb{Z}_2$) of d-cell **subsets**

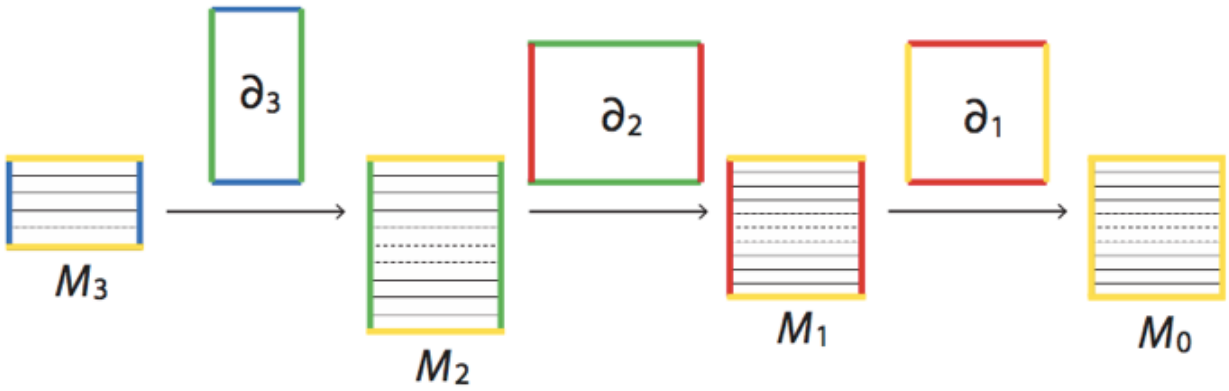
Unit d-chains (single d-cell subsets), give the standard bases (M_d rows) of d-chain spaces

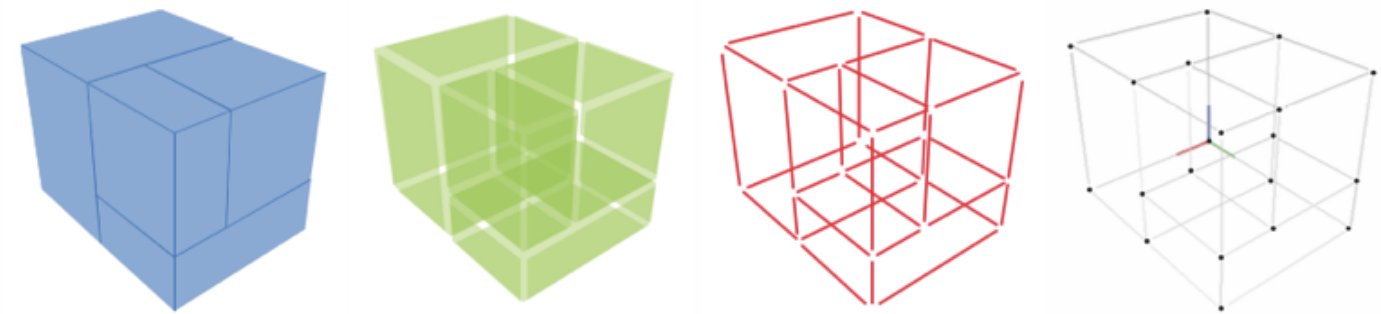




LAR

A standard for model topology through a general and simple repr scheme

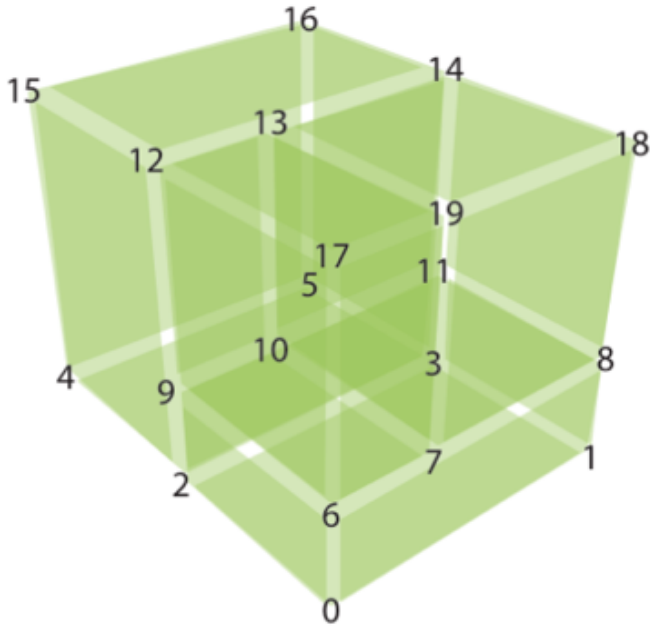




Models: (co)chain complexes → Reprs: sparse binary matrices

Characteristic matrices of d-chain spaces

Matrix representation of the basis — d-cells as subsets of vertices

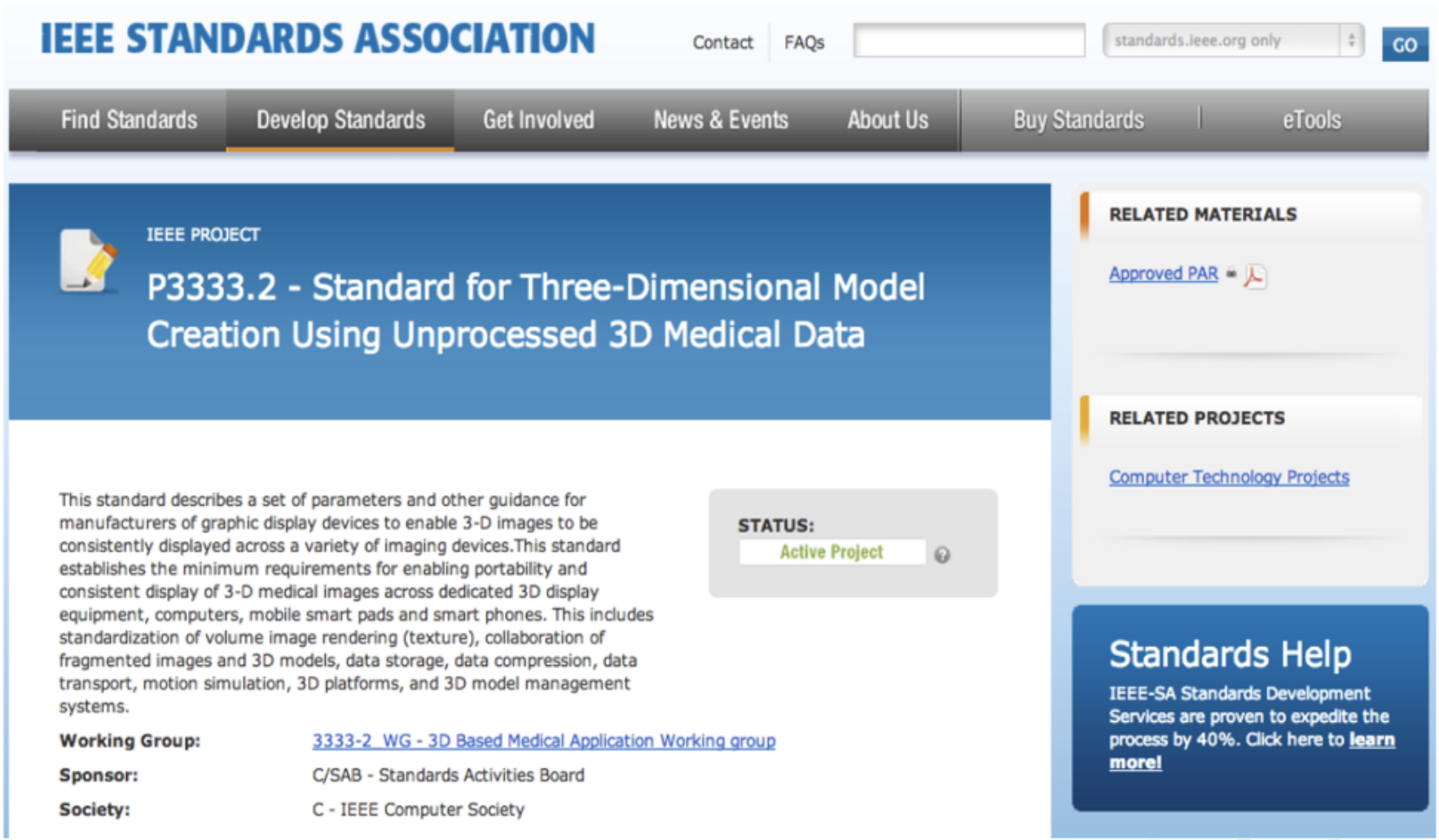


$$M_3 = \begin{pmatrix}
0\,0\,1\,1\,1\,1\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0 \\
1\,1\,1\,1\,0\,0\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,0\,1 \\
0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,1
\end{pmatrix}$$

$$M_2 = \begin{pmatrix}
1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,1\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,1\,1\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0 \\
0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\,0\,1\,0\,0\,0\,0 \\
0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,1 \\
0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1 \\
0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,1 \\
0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0\,0\,1\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,1\,0\,1 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,1\,1\,1
\end{pmatrix}$$

IEEE-SA P3332 WG

Solid models from 3D medical images



Compressed Sparse Row (CSR) matrix storage

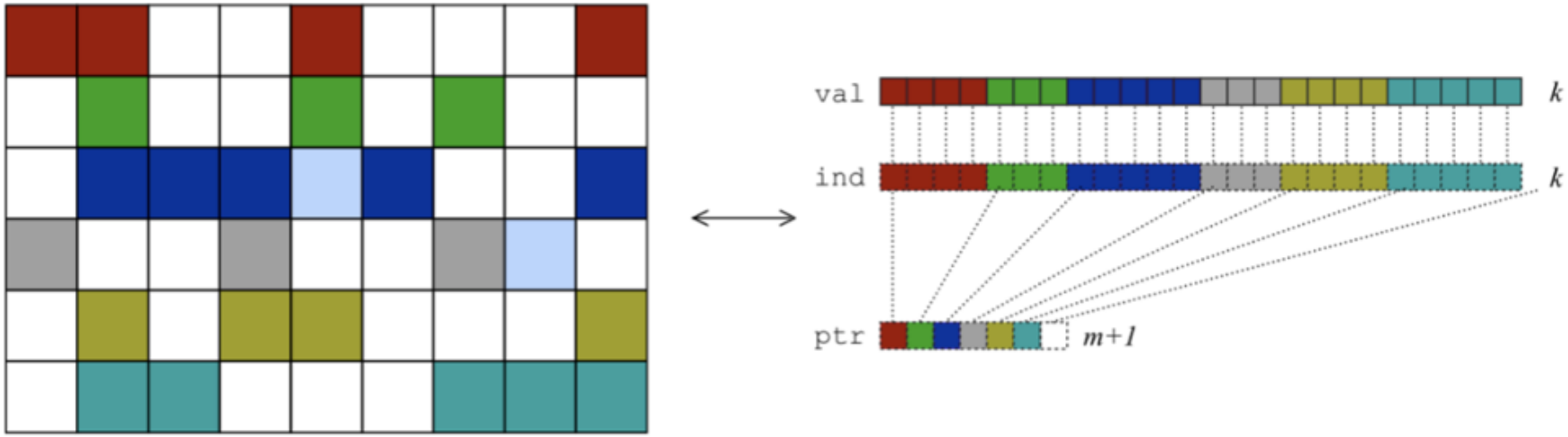


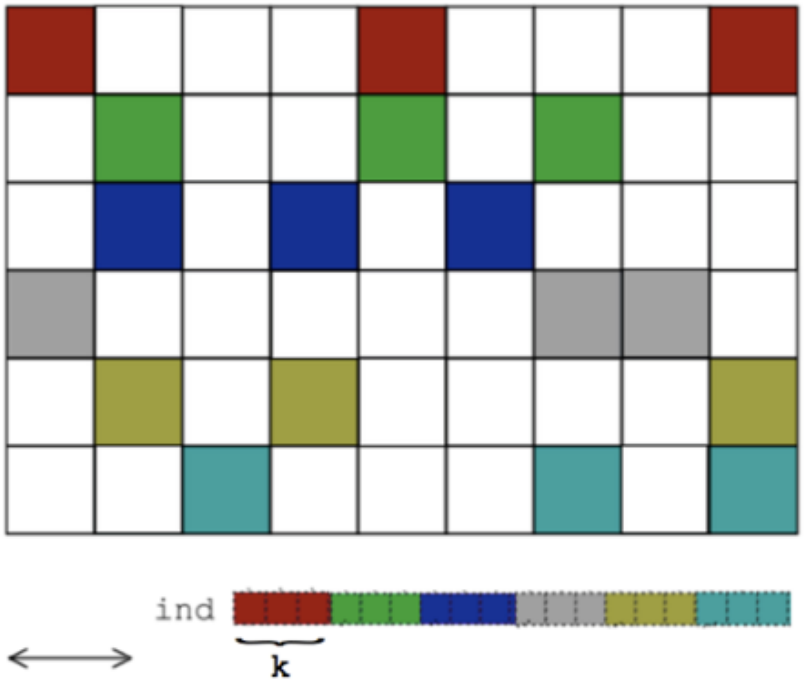
image from

Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel, Optimization of sparse matrix-vector multiplication on emerging multicore platforms, Proceedings of the 2007 ACM/IEEE conference on Supercomputing (New York, NY, USA), SC '07, ACM, 2007, pp. 38:1–38:12.

Storage of LAR(X) $\equiv$ CSR(M_d) matrix

Amazingly compact storage of a **solid** model

REDUCED LAR



simplicial d-complexes: $k = d + 1$
cuboidal d-complexes: $k = 2^d$

Remark (Input and **long-term storage**)

$$space(\text{LAR}) = |FV| = 2|E| \quad !!!$$

Remark (**Full** topology representation)

$$|VE| + |VF| = 4|E|$$

Remark (**Any** topological queries)

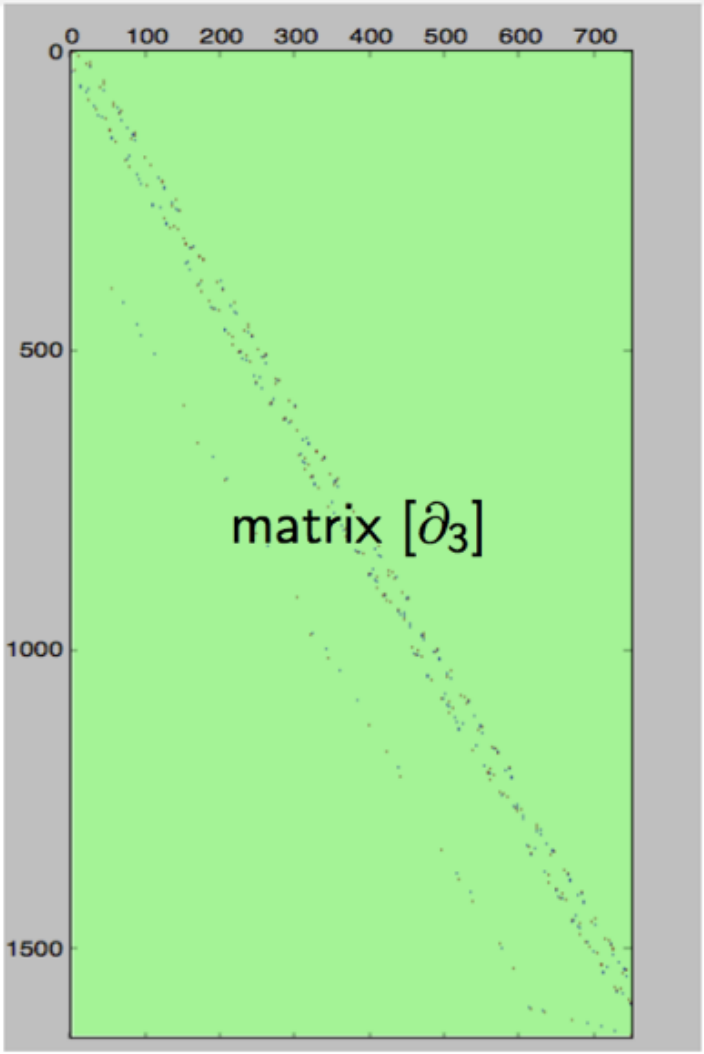
single SpMV multiplication

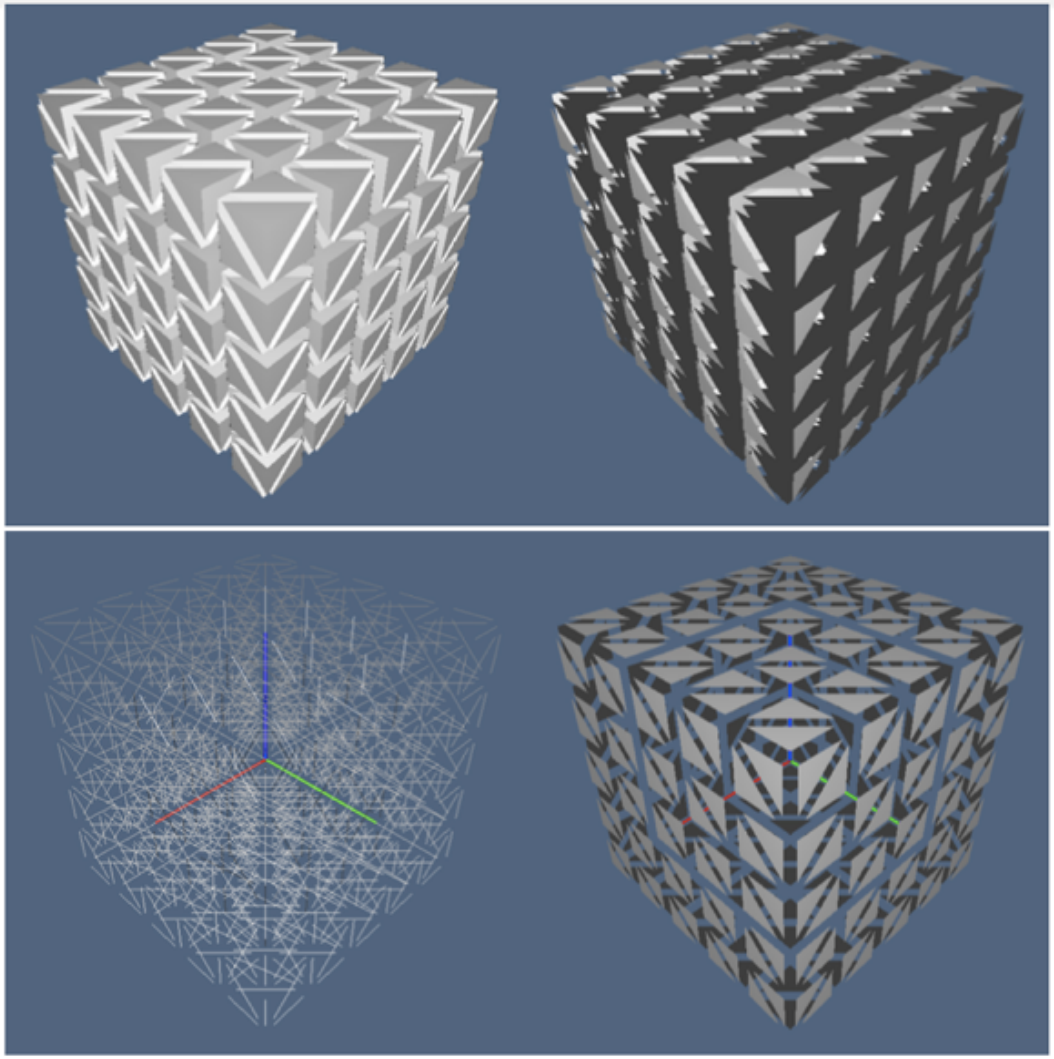
Remark (Sparse Matrix-Vector Multiplication)

is one of the most important computational kernels, for very effective iterative solution methods

Example: 3D simplicial grid

Boundary computation via a single SpMV product: $[c_2] = [\partial_3]\,\mathbf{1}$





Some LAR algorithms

LAR Representation Scheme

Let a finite cellular complex $\Lambda(X)$ be given, such that $X = \Lambda_0 \cup \cdots \cup \Lambda_d$. Let also $M_p \in \mathbb{Z}_2^{m \times n}$ $(0 \leq p \leq d)$ be binary characteristic matrices, with number of rows equal to the number of p-cells, and number of columns equal to the number of 0-cells:

$$m = k_p = \sharp \Lambda_p, \qquad n = k_0 = \sharp \Lambda_0.$$

Each $m_{ij} \in M_p$ tells us whether (the vertex) $v_j \in \Lambda_0$ is contained on the boundary of the cell $\lambda_i \in \Lambda_p$ or not.

The *LAR scheme* is a map from mathematical models of solids to their computer representations:

$$\text{LAR} : \mathbb{C}\text{h}(\Lambda(X)) \to (M_p)_{p=1}^{d}.$$

Validity Test: $\partial\partial = 0$

In other words, LAR represents *chain complexes* $\mathbb{C}$h, supported by a finite cellular complex $\Lambda(X)$, by d-tuples of binary *compressed sparse row* (CSR)[1] matrices, where d is the dimension of the space X. By definition, a LAR representation $(M_p)_{p=1}^{d}$ is *valid* if and only if it represents a valid chain complex, i.e. its boundary operators form an exact short sequence

$$[\partial_{p-1}][\partial_p]\mathbf{1}_{k_p} = \mathbf{0}_{k_{p-2}} \pmod 2, \quad 1 \leq p \leq d \tag{1}$$

where $k_h = \sharp\Lambda_h$, and the matrices $[\partial_h]$ $(h = p, p-1)$ are boundary operators. We give a detailed algorithm for constructing boundary operators in Section 3.3.

Adjacency relation

The adjacency relation between d-cells is the subset of cell pairs that share some vertex (0-cell). The adjacency matrix A_d can be computed by matrix multiplication between the characteristic matrix $M_d(\Lambda_d)$ and its transpose:

$$A_d = M_d \, M_d^t$$

Adjacency relation computation

$A_d = (a_{i,j})$ is integer-valued and symmetric. Each $a_{i,j}$ gives the *number* of 0-cells shared between the cells $\lambda_i, \lambda_j \in \Lambda_d$. When the set intersection of two cells $\lambda_i, \lambda_j \in \Lambda_d$ contains a number of vertices greater than or equal to the dimension d of cells, they share a common facet $\mu \in \Lambda_{d-1}$, that can be computed as a component-wise integer product (or binary intersection) of rows i and j of M_d. The generated characteristic vector gives a row of the matrix M_{d-1}. In symbols:

$$a_{i,j} = \sharp(\lambda_d^i \cap \lambda_d^j)$$

$$a_{i,j} \geq d \quad (i \neq j) \Rightarrow \exists\, M_d(i) \wedge M_d(j) = \mu \in \Lambda_{d-1}.$$

Boundary Operator Matrix in $\mathbb{Z}_2$: ALGORITHM

(i) Compute $\mathrm{CSR}(M^p_{p-1}) := \mathrm{CSR}(M_{p-1})\,\mathrm{CSR}(M_p)^t$ as the product of sparse matrices.

(ii) Filtering procedure: for each $0 \leq i \leq k_{p-1} - 1$,

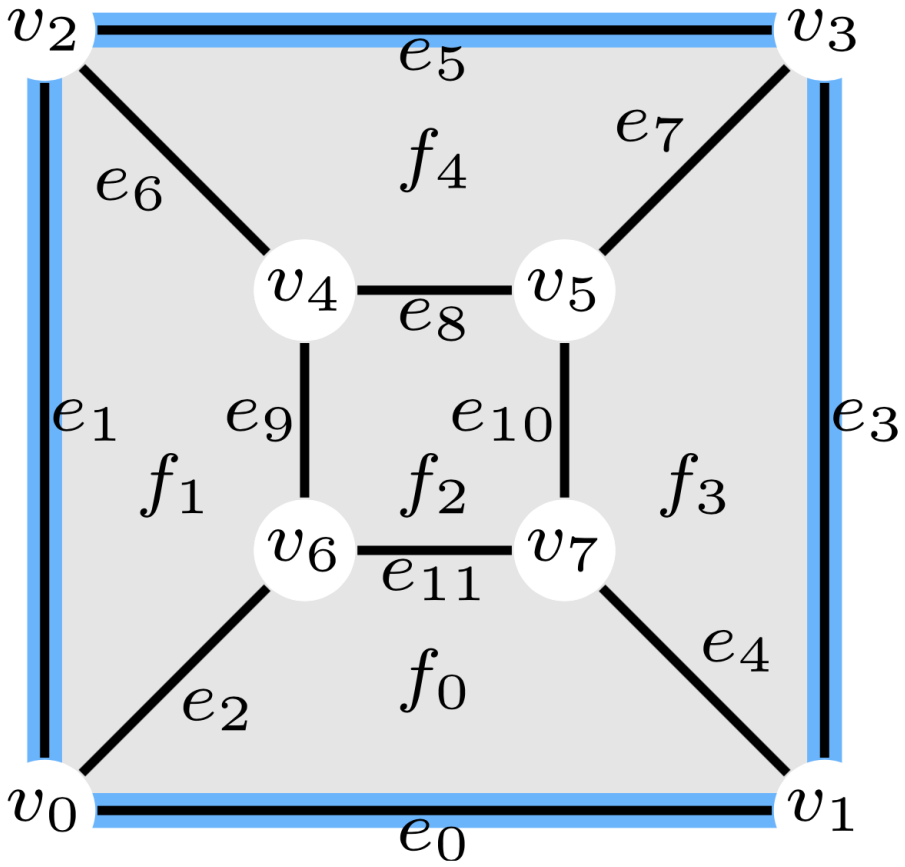
(a) compute the number $k := \sharp\mu^i_{p-1}$ of non-zero elements stored in row i of $\mathrm{CSR}(M_{p-1})$;

(b) for each $0 \leq j \leq k_p - 1$:

$$[\partial_p](i,j) := 1 \quad \text{if } M^p_{p-1}(i,j) = k \quad \text{else} \quad [\partial_p](i,j) := 0.$$

By duality, the same procedure may be used to compute the $(p-1)$-coboundary operator δ_{p-1} as the transpose of the boundary ∂_p.

Polytopal complex: EXAMPLE



Boundary computation in $\mathbb{Z}_2$: EXAMPLE

Example. Here we compute the $[\partial_2]$ boundary matrix for the regular polytopal complex shown in Fig. 1(b), from characteristic matrices M_2 and M_1 given in Section 4.2. According to step (i) of the algorithm above, we get

$$M_1^2 = M_1 M_2^t.$$

Then, we filter the matrix M_1^2 according to the criterion given in step (ii), producing the matrix of $\partial_2 : C_2 \to C_1$.

$$[\partial_2]^t = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \qquad (2)$$