MEMulator™: A fast and accurate Geometric Modeling, Visualization and Mesh Generation Tool for 3D MEMS Design and Simulation

Tushar Udeshi and Eric Parker

Zyvex Corporation
1321 N. Plano Road, Richardson TX 75081, USA
[tudeshi, eparker]@zyvex.com

ABSTRACT

We describe the functionality as well as the underlying algorithms of the MEMulator software. We use robust voxel-based algorithms for MEMS process emulation which result in accurate 3D geometries. We describe how the traditional shortcomings of voxel-based algorithms have been overcome. We also describe unique techniques developed for visualization and finite element mesh generation of the voxel data.

Keywords: MEMS, Process Emulation, Voxel algorithms, Visualization, Mesh Generation

1 INTRODUCTION

Voxel-based techniques are known for their robustness and flexibility and have been used for MEMS design and simulation [8]. However, they have several problems:-

1. Memory inefficient: Since voxels discretize space, this representation is memory-intensive.
2. Computationally Expensive: Any operation performed on the voxels requires visiting a large number of them, thus increasing computational cost.
3. Visualization: Isosurface extraction is required for visualizing the surface. Standard isosurface extraction algorithms [4] are computationally expensive and yield a large number of surface triangles.
4. Finite Element Mesh Generation: Hitherto there has been very little work done on mesh generation directly from voxels. Most mesh generation algorithms need a surface representation of the geometry.

In this paper, we describe the patent-pending algorithms and data structures incorporated into the MEMulator that address all the above shortcomings. The software runs on standard PCs and has a powerful and intuitive Python-based interface. Modeling and meshing time for typical MEMS parts is a few minutes.

2 SHARED OCTREE DATA STRUCTURE

An octree [2] is a standard data structure used to represent volumetric data. An octree is derived by successively dividing 3D space in all three dimensions to form octants. Subdivision stops when the region represented by the octree node is homogenously filled with one material. It can be shown [2] that storage requirements as well as any operation performed on the octree is proportional to the area of the surfaces represented by the octree.

In [1], Bill Gosper describes an algorithm for Conway's game, Life, in which each unique 2^n pattern of life cells is represented by a single node of a shared quadtree. This shared quadtree structure may be generalized to a shared octree in three dimensions. Also, non-binary data may be stored by creating a node for each unique voxel (the voxel contents may include scalars, vectors, or any other type of data).

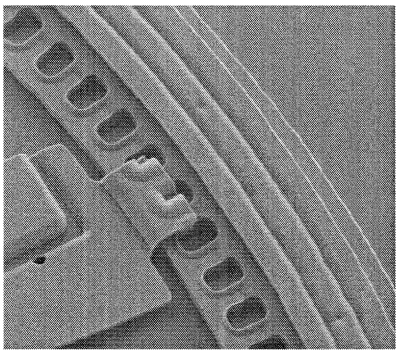
In each voxel, we store the fraction by volume of all materials intersecting it similar to volume of fluid techniques [5]. For simplicity of implementation we restrict the number of materials in each voxel to two.

This simple compression scheme utilized by the shared octree is surprisingly effective on MEMS geometry (See Fig 7 for an example). Also, all operations on the geometry are performed directly on this compressed format.

3 RESULTS MATCHING

We take advantage of the ample self-similarity in MEMS geometry, captured by the shared octree data structure, to accelerate processing on this shared octree. The acceleration is achieved by caching partial processing results. Note that in the shared octree, every unique node is stored only once. As an example, consider the problem of determining the volume of a particular material *m,* stored in the shared octree. This is equivalent to adding the fraction of volume of *m*, in all voxels. Now assume that the we have determined that the volume of *m* for a octree node *n* is *v*. If the node *n* is encountered anywhere else in the tree, we do not need to re-compute the volume of *n*, but instead just look it up from a results cache.

For the above operation, the key into the results cache is just the octree node. For other operations, there may be additional factors affecting it (neighboring nodes for instance), and these must go into the key as well. Two criteria must be met in order for an algorithm to take advantage of results matching: 1) the key for the cache must be computed from local or "relative" data with respect to the subtree node, and 2) the operation must be deterministic. The key should not depend on global values (such as the (x,y,z) coordinate) because these global values will never match with another node. The key may depend on neighboring information, or other information outside of the octree, as long as all the data are accessible relative to the subtree represented by the given octree node.



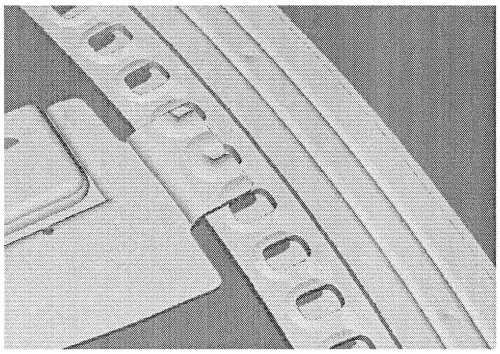


Fig 1: Left: SEM image of a MUMPS device. Right: Geometry generated by MEMulator. A leading MEMS CAD software package failed when given this input

It is clear that the acceleration achieved by results matching is dependent on the size of the key into the results cache. A smaller key means matching will occur more often. All the algorithms described in the later sections take advantage of this technique. Results matching typically facilitates speedups of 1 to 2 orders of magnitude for MEMS process emulation.

4 MEMS PROCESS EMULATION

We emulate the MEMS fabrication process by geometric modeling. The flexibility and robustness offered by voxel-based algorithms are very useful for this. The traditional approach to MEMS process emulation [3, 10, 11] is to perform all the operations using a solid geometry kernel. This technique is not robust and does not yield accurate geometry. For example, consider Low Pressure Chemical Vapor Deposition (LPCVD), which is a common step used in MEMS fabrication. The result of LPCVD is an isotropic deposition. However, emulating LPCVD with a solid-modeling offset operation results in sharp corners and introduces substantial error. Refer to Fig. 2. The geometric error introduced along edges would be 27.3%, while at a corner it would be 90.99% in terms of volume of material deposited. Moreover this operation could result in complex surface self-intersections as shown in Fig. 1. Detecting and "delooping" these self-intersections is prone to numerical instability and therefore the offset operation could fail.

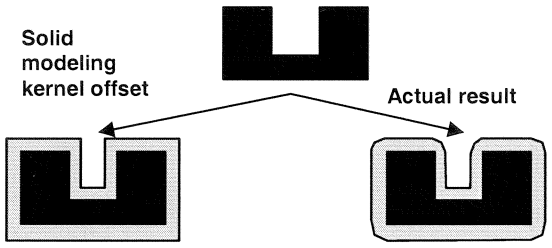


Fig 2: Cross section of isotropic deposition of a material (gray) on a simple pattern (black)

The geometric modeling required for MEMS process emulation can be robustly implemented using Minkowski summation and subtraction operations with different filters on the shared octrees. The Minkowski sum operation is local because a seed voxel (i.e. one from which Minkowski addition occurs) can affect only those voxels which are within a radius of the Minkowski filter. It is also deterministic (i.e. the result of a Minkowski sum on two identical regions is the same.)

Isotropic deposition is implemented by performing a Minkowski sum with a sphere. Ellipsoidal filters could be used to deposit more on horizontal layers than vertical layers or vice versa. This is useful for deposition of metal as shown in Figure 3. Asymmetric ellipsoidal filters could be used for depositing more on the top surfaces than the bottom.

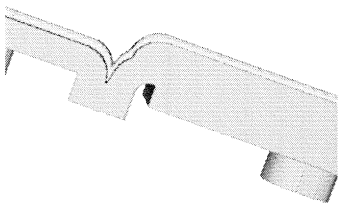


Fig 3: Anisotropic deposition of gold to form a pad. Notice that deposition occurs more on horizontal surfaces

4.1 Etch

Isotropic and anisotropic etching is implemented as is done in deposition except that a Minkowski subtraction is performed. Isotropic etching emulates release etches. This is particularly useful for detecting trapped or unreleased oxide. This is illustrated in Figure 4. The straight-wall etch is simplified by doing a vertical offset of the surface.

4.2 Chemical Mechanical Polish (CMP)

CMP is easily implemented by replacing all solid material above a user-specified distance with air material.

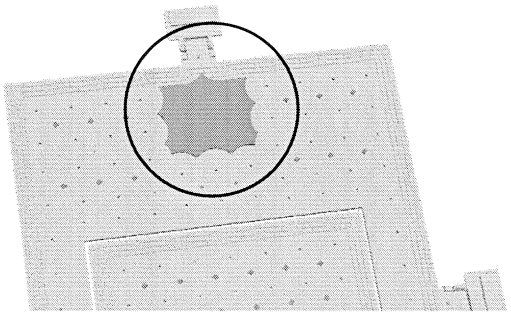


Fig 4: Emulation of release etch. Notice the unreleased oxide (marked) which occurred due to insufficient etch holes. The fabricated part was indeed stuck due to this.

4.3 Connected Component Finding

After the MEMS process emulation is performed, we can group the voxels by mechanical or electrical connectivity. We accelerate the standard connected component labeling algorithm based on equivalence class merging by using results matching. Note that this is a totally local operation because the connected component merging of a voxel is performed only with its three "lower" neighboring voxels.

Once the connected components have been found, the user can move them around to visualize assembly (See Fig. 5). Finite Element meshing and analysis can also be performed on each component individually or in conjunction with other components.

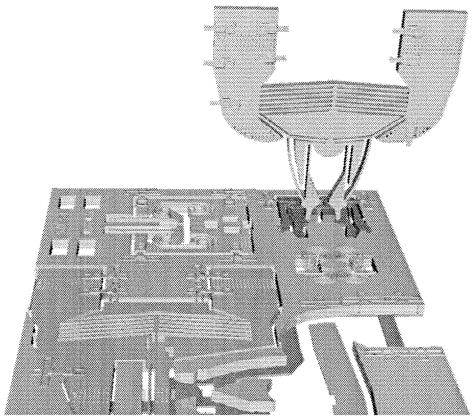


Fig 5: Connected components and assembly visualization

4.4 Isolating air pockets

In some MEMS processes such as Summit V [3], an isotropic wet etch may occur as an intermediate step. If this etch is followed by an LPCVD, the surface may close up and isolate an air pocket (See Fig 6). In that case, further operations should not affect the isolated region unless the pocket is opened up by a subsequent etch step. We implement this by determining all connected components of air. If there is only one component, there are no isolated pockets. If not, we determine the connected component, say *c*, of an air voxel at the boundary of the simulation space. All air voxels which are labeled with a component other than *c* are isolated and their material is replaced with a special material, say *s*. This would prevent further depositions from occurring. Now, if during an etch, material *s* is exposed, we run the air pocket isolation procedure again.

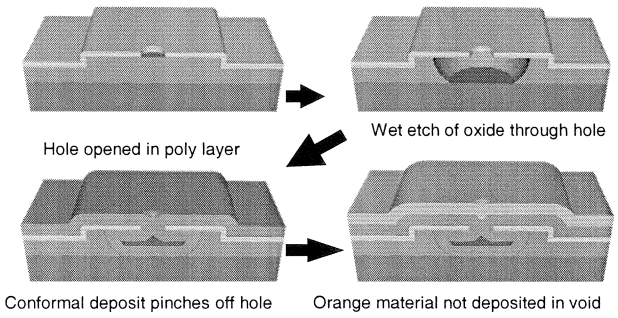


Fig 6: Example of a sequence of steps resulting in an isolated air pocket.

4.5 Wafer Bonding

In wafer bonding, one wafer is flipped, aligned with a base wafer and then the two are merged together. Translation, rotation (by multiples of 90 degrees) and mirroring of octrees have been implemented. This is used for aligning the second wafer. Then a merge step is performed when a new octree is generated from the two input octrees. A voxel in this new octree is assigned material(s) based on the materials present at that same voxel in the two input trees. Once this is done, an air pocket isolation step is performed because wafer bonding does usually create isolated regions which further processing steps should not affect.

5 VISUALIZATION

Visualization of the shared octrees requires construction of the isosurface first. There has been lot of work done on generating polygonal isosurfaces. The standard approach is to generate a dense polygonal mesh [4] and then simplify it [7]. Recently, an approach to generate an adaptive isosurface directly from a signed octree was presented [9]. However, these techniques are usually used with datasets of size up to 512^3, whereas the average size of our datasets is an order of magnitude greater than that.

Rusinkiewicz et. al. [6] introduced a point-based rendering system which is amenable to real-time visualization of large datasets because it is inherently multi-resolution and back-face, detail and view-frustum culling can be implemented efficiently. Their technique takes a dense polygonal mesh as input and generates a splat hierarchy which is then visualized in real-time.

We generate a splat hierarchy directly from the shared octree. Every node in the splat hierarchy is a planar

approximation of the surface present in the subtree rooted at it. In addition, the nodes in the splat hierarchy are also shared. The isosurface extraction proceeds in a bottom-up fashion. We estimate the splat parameter of a leaf node by determining the points of intersection of the isosurface with the edges of a cube formed by the leaf node and its seven lower neighbors and then fitting a least square plane to these points. Splat parameters of interior nodes are determined by performing a weighted average of those of their children. Note that results matching can be used to accelerate isosurface extraction because the parameters of a splat are determined fully from a node and its 7 lower children (local). Due to the extreme self similarity in MEMS parts, our algorithm is faster by one to two orders of magnitude compared to traditional isosurface extraction algorithms. Moreover, visualization of extremely large datasets is possible in real-time (See Fig 7).

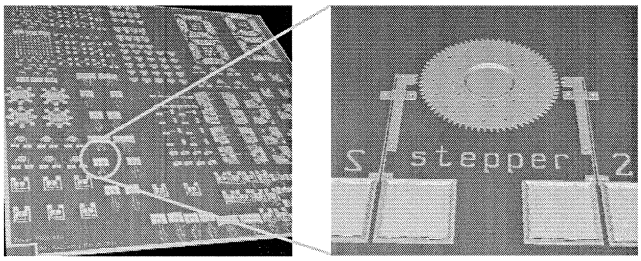


Fig 7: A MUMPS die (1 cm x 1 cm) at a quarter micron resolution visualized in real time on a standard PC. The dataset is composed of 160 billion voxels. The application utilized less than 75 MB RAM.

6 MESH GENERATION

There has been a lot of research and industry focus on mesh generation from mechanical CAD data. However, these algorithms require a smooth boundary representation with features pre-defined. One technique would be to extract a polygonal isosurface and use standard mesh generation software. We tried this on several research and commercial codes but were unhappy with the robustness and/or the quality of the meshes generated.

We developed a new octree/Delaunay based algorithm which works directly from the shared octree, without having to extract boundary surfaces. Here are its features:

1. *Adaptive:* The mesh density adapts to the geometry (i.e. it is coarse in featureless regions and refined in dense ones.)
2. *Multiple materials:* Generates a consistent volumetric tetrahedral mesh of all materials (including air) in one step. The algorithm guarantees that tetrahedra will not straddle interfaces between materials.
3. *Error Control:* The user can specify the maximum distance that a node is allowed to move from the surface it is supposed to be on in different regions of the dataset.
4. *Local Refinement:* Local refinement of the mesh in regions deemed important to the simulation is possible.

Using this algorithm, we are able to mesh and then perform a finite element analysis of accurate 3D MEMS geometry. Fig. 8 shows an example.

Acknowledgement: This work was supported by NIST-ATP grant 70NANB1H3021 and Zyvex Corporation.

REFERENCES

[1] R. Gosper, "Exploiting Regularities in large cellular spaces", Physica 10D, 1984, 75-80.

[2] G. M. Hunter, "Efficient Computation and Data Structures for Graphics", Ph.D. Thesis, Dept. of EE and CS, Princeton University, NJ, 1978.

[3] C.R. Jorgensen and V.R. Yarberry, "A 3D Geometry Modeler for the SUMMiT V MEMS Designer".

[4] W.E. Lorensen and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." Computer Graphics. 21(3):163-169, July 1987.

[5] W. Noh and P. Woodward, "Simple Line Interface Calculation", Proc., Fifth Int. Conf. on Fluid Dynamics, 1976.

[6] S. Rusinkiewicz, and M. Levoy. "QSplat: A Multiresolution Point Rendering System for Large Meshes.", Computer Graphics(SIGGRAPH '00). 353-358, Jul. 2000.

[7] W. Schroeder, J. Zarge, and W. Lorensen. "Decimation of Triangle Meshes.", Computer Graphics (SIGGRAPH '92). 26(2):65-70, Aug. 1992.

[8] Z. Tan, M. Furmanczyk, M. Turowski, and A. J. Przekwas, "CFD-Micromesh: A Fast Geometrical Modeling and Mesh Generation Tool for 3D Microsystem Simulations", Proc. of the MSM, March, 2000.

[9] J. Tao, F. Losasso, S. Schaefer, and Joe Warren. "Dual Contouring of Hermite Data.", ACM Transactions on Graphcs(SIGGRAPH '02). 21(3):339-346, Jul. 2002.

[10] MemsCap www.memscap.com

[11] Coventor www.coventor.com

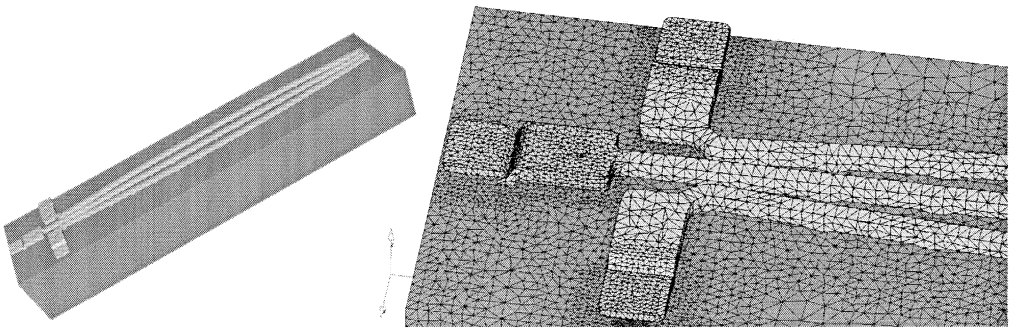


Fig 8: A bidirectional actuator modeled and meshed in less than two minutes.