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

Chandrajit Bajaj[1], Alberto Paoluzzi[2] and Giorgio Scorzelli[3]

[1]ICES and Computer Sciences Dept., Univ. of Texas at Austin, bajaj@cs.utexas.edu
[2]Dip. Informatica e Automazione, Univ. "Roma Tre", paoluzzi@dia.uniroma3.it
[3]Dip. Informatica e Automazione, Univ. "Roma Tre", scorzelli@dia.uniroma3.it

ABSTRACT

We introduce a novel *progressive* approach to generate a Binary Space Partition (*BSP*) tree and a convex cell decomposition for any input triangles boundary representation (B-rep), by utilizing a fast calculation of the surface inertia. We also generate a solid model at progressive levels of detail. This approach relies on a variation of standard *BSP* tree generation, allowing for labeling cells as *in*, *out* and *fuzzy*, and which permits a comprehensive representation of a solid as the Hasse diagram of a cell complex. Our new algorithm is embedded in a streaming computational framework, using four types of dataflow processes that continuously *produce*, *transform*, *combine* or *consume* subsets of cells depending on their number or input/output stream. A varied collection of geometric modeling techniques are integrated in this streaming framework, including polygonal, spline, solid and heterogeneous modeling with boundary and decompositive representations, Boolean set operations, Cartesian products and adaptive refinement. The real-time B-rep to *BSP* streaming results we report in this paper are a large step forward in the ultimate unification of rapid conceptual and detailed shape design methodologies.

Keywords: Geometric modeling and programming, Representation conversion, Solid modeling. Binary Space Partitions.

Fig. 1. (a) BSP tree generated from a polygon B-rep with normals; (b) solid helicoid; (c) progressive intersection.

1. INTRODUCTION

We introduce a novel progressive conversion from triangle boundary representations (B-rep) to solid Binary Space Partition (BSP) decompositions. The algorithm may be used for importing geometric models into a dataflow (streaming) pipeline to

be used for shape modeling of complicated geometries. Stream processing pipelines are commonly used in computer graphics engines; conversely, their use for geometric modeling is relatively new. We use balanced BSP trees as progressive representations of shapes, yielding multiple levels of detail. The algorithm we introduce in this paper generates coarser to finer geometric approximations progressively, and closely integrates stream rendering for instant visualization. In our computational framework, we use specialized routines to generate progressive representations of primitive objects, such as spheres and cylinders, as well as a novel conversion algorithm to import B-rep models from external data stores, and embed them into complex generative expressions, in order to build progressive BSP-trees, refinable on demand. Also, view-dependent data may be propagated upstream in our stream processing framework so that the computation is focused on detailing only the model features currently of interest for the shape designer.

Our novel conversion algorithm from B-reps to solid decompositions with convex cells takes as input any boundary triangulation. It produces a natively balanced BSP-tree with only *O(n)* preprocessing. The preprocessing is the computation of the Euler tensor of each input triangle. The Euler tensor is strictly related to the inertia tensor, and takes into account the contribution of the triangle to the spatial distribution of surface points. The tensor matrices are then attached like textures to triangles, and are only recomputed when some triangles are split. A best-fitting parallelepiped is thus generated for the input surface using the eigendecomposition of the Euler tensor., and represented as the intersection of six boundary planes for a standard BSP tree. The interior cell is further split by the principal plane normal. Two local best-fitting parallelepipeds are then computed for the two half-spaces, and their intersection with the two sub-cells are used to add detail to the solid approximation generated so far. Some cells of the decomposition are labeled as either external or internal to the boundary and are never further split. The splitting process of the cells intersecting the boundary (called *fuzzy*) continues recursively by using the local best-fitting parallelepipeds, until only a few input triangles remain within a cell. In the very last step only the boundary triangles are used to add additional detail.

The above conversion algorithm integrates seamlessly with our stream processing approach to geometric modeling [20]. It provides a native generation of balanced BSP trees with only linear preprocessing of the imported B-rep. Balanced BSP trees provides an efficient spatial index allowing for fast point location, collision detection. and distance field computations. Additionally this conversion provides a natural convex decomposition [1] of large-scale models, resulting in a representation that is highly portable and scalable, and can be effectively used on PC clusters and other HPC architectures. Our approach is aimed at supporting generative geometric modeling, starting from either primitive or imported shapes, that may come from external data stores. Such atomic shapes may be either transformed with affine or projective transformations, or aggregated within hierarchical assemblies, or combined using several operators, including Boolean set operations [18], Cartesian product and Minkowski sums of point-sets. Finally, our approach is *embarrassingly parallel* (see [20]), from limited to no communication between processes generating portions of the model on the nodes of a PC cluster, where communication overhead is negligible and the speedup is essentially linear with the number of processors

2. STREAMING GEOMETRIC MODELING

Our stream processing approach results in fine-grain streamlined parallelism where suitable geometric data structures flow between specialized processes, with the resulting shape produced by progressive refinements of an initial coarse approximation . The data tokens which flow between different computations are just a couple of pointers to the twin representations of the mesh, i.e. (a) a BSP-node (actually either a linear hyperplane or a leaf label) and (b) its associated *d*-cell in the Hasse diagram of the current mesh. When computing Boolean-set operations between large-scale objects, the result from earlier traditional algorithms [14] are always only generated at the end of the entire (possibly very long) computation. With our approach, a continuously refined estimate of the Boolean-set operation result is available from the very beginning. If the output appears unsatisfactory, the progressive task can be instantly terminated. The stream processing computation can also be terminated by using a local threshold for the geometric approximation error, and possibly the error depending on the viewer's gaze and distance.

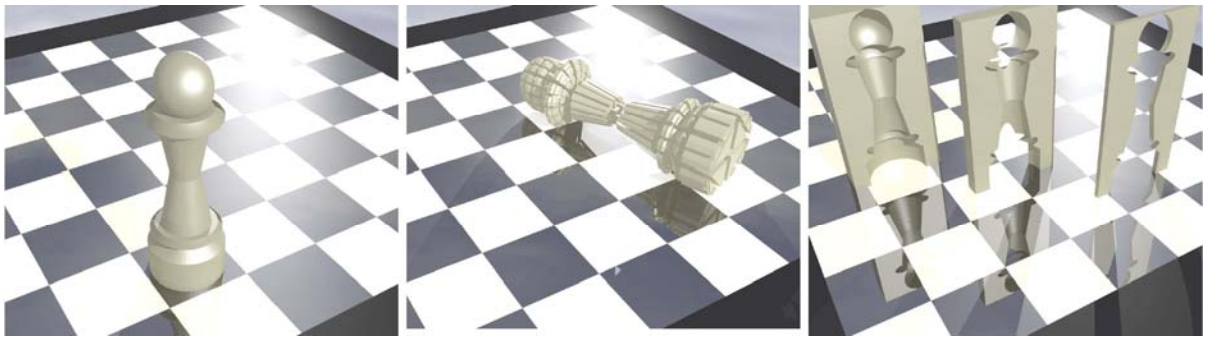

Fig. 2. A chess model: (a) union of a cylinder, a cone, a scaled sphere, two cones, a scaled sphere and a sphere, rendered flat at maximum resolution; (b) exploded geometry; (c) differences from parallelepipeds

2.1 Progressive BSP data structures

We use two main data structures: a **BSP-tree** for progressive generation of geometry, and a doubly linked list representation of **Hasse diagrams** for the storage of already generated geometry. Here we include a short summary for completeness, of the main properties of BSP trees [14]. In particular, each node of a BSP tree: (a) is associated with a convex cell; (b) if a non--leaf, then it additionally captures the hyperplane splitting its cell; (c) if a leaf, then it additionally contains a label that characterizes its cell as either IN, OUT or FUZZY; (d) is defined as the set intersection of the halfspaces associated to the (unique) path from the node to the root; (e) equals the set union of cells associated to the subtree rooted within it (see Figure 3). With respect to standard BSP trees, we only add a FUZZY label to nodes that can be further split progressively, so that each *frontier* (time-dependent space partition) is always subdivided into solid (IN), empty (OUT) and yet undecided (FUZZY) cells.

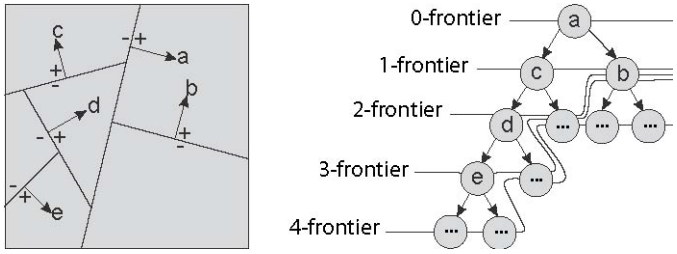

Fig. 3. Frontier evolution as progressive refinement of the space partition.

The need to maximize runtime efficiency is the main reason for using a B-rep based on the Hasse diagram [24], which allows for an explicit and complete storage of both the geometry and the topology of the model. In order theory, a Hasse diagram is a graph, whose nodes are the members of a finite partially ordered set S, and where there is an arc from x to y iff: (a) $x < y$ and (b) there is no z such that $x < z < y$. In this case, we say y covers x, or y is an immediate successor of x. Hasse diagrams can be used to give a complete representation of the structure of d-polytopes and d-meshes, with respect to the inclusion relation between k-faces, $0 \leq k \leq d$. The very basic operation is the *split* of a fuzzy *d*-cell with a hyperplane, where an efficient update of the Hasse diagram is performed using the fast algorithm of [2]. The inverse operation is called *join*, and updates the representation by substituting two split *d*-cells (and their split *k*-faces) with their convex hulls. These operations are geometrically robust, i.e. able to withstand the effects of numerical errors

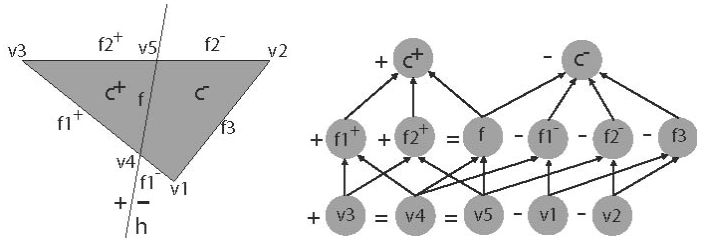

Fig. 4. Splitting of a *d*-cell *c* with a hyperplane *h*, and corresponding Hasse diagram.

Notice that we only handle collections of piecewise-linear bounded and convex sets, i.e. complexes of polytopal cells. Note also that the approach is *dimension-independent*, in the sense that both the data structures and the operations would work with solids (codimension 0 pointsets) of arbitrary dimension *d*.

2.2 Stream Processing framework

In our stream processing framework, geometric objects are always produced by the (progressive) evaluation of a generating expression, which is compiled into the dataflow pipeline made from four kind of processes, denoted respectively as *producers*, *transformers*, *combiners* and *consumers*, according to the number of their input/output stream. A producer, or ***builder***, is a process with no input, and one output stream. It continuously generates progressive polyhedral approximations, from coarser to finer levels of detail. A special effort has been made to enrich our streaming technology with a wide set of producers, providing solid primitives, polynomial and rational splines, subdivision surfaces and polygon models. A ***transformer*** is a process with one input stream and one output stream. Our transformers either apply affine or projective transformations or produce the complement or extrusion of their input. A ***combiner*** is a process with more than one input stream and a single output stream. Such a process combines several progressive BSP streams and returns the result stream. In each combiner process, an efficient variable-grain merge of splitting planes are used to combine the operation arguments. In particular, a cursor pointer is set to the current input splitting node. This pointer is moved to the following input tree every g splitting nodes, where g stands for granularity. A ***consumer*** is a process with one input stream and no output streams, that is used either to compute suitable model properties or to analyse or visualize the progressively generated model. Depending on the computed properties, a consumer process may decide which cells of the input stream should be expanded, i.e. further detailed, or collapsed, (joined) into a single cell, and hence provides a mechanism for adaptivity and control, with feedback.

The conversion algorithm described in this paper implements an important **builder** component in our stream processing architecture. In particular, it provides a mechanism to *progressively import* into the stream, models extracted from an *external store* and, in particular, supplied using the weakest B-rep solid representation, i.e. a set of triangles. The only constraint on the input triangles is that they must be coherently oriented. In other words all triangles must be either implicitly or explicitly associated with the correct external normal to the surface they belong to. Such a constraint is normally satisfied by most data structures, and in particular by the data files produced for computer aided design (CAD)..

3. Progressive CONVERSION ALGORITHM

The new algorithm generates a solid decomposition into convex cells by producing a balanced BSP-tree, and by using only a coherently-oriented decomposition of the object boundary into raw triangles, and without any need for a comprehensive representation of the boundary topology.

The main idea behind the conversion is quite simple. A fast preprocessing, executed in time linear in the number of input triangles, computes for each triangle a 4x4 matrix that numerically codifies its inertia. The sum of all such matrices gives a good representation of the spatial distribution of the surface points. Such a matrix may also be used used to generate a best-fitting ellipsoid, centered in the center of mass of the surface.

A first solid approximation of the input surface is given by the minimal best-fitting parallelepiped, parallel to the principal axes of the spanning ellipsoid, Such a solid is represented as a standard BSP-tree, as the intersection of six linear subspaces tangent to the boundary surface. This initial solid is then split by a plane passing for the center of mass and normal to the direction of maximal elongation. The set of input triangles is in turn split into two subsets contained in the two half-spaces of the splitting plane, and the two subsets are associated to "below" and "above" sub-trees of the root of the BSP tree. This process of confinement of each surface subset into narrower and properly rotated best-fitting parallelepipeds, and the splitting in the principal direction, is repeated recursively for each sub-tree. The leaves of a BSP-tree give a partition of the embedding space into convex cells. Some cells are labeled as *empty* or *full*, i.e. as either external or internal to the boundary surface. Such cells are not split further. Other cells cross the boundary, and are provisionally classified as *fuzzy*. The shape confinement using the local best-fitting parallelepiped is then repeated, either for every fuzzy cell or only for those cells where better detail is needed. Each fuzzy cell is detailed by its intersection with the best-fitting parallelepiped. This intersection of the current cell always produces some empty or full subcells, together with a smaller fuzzy cell. Finally this cell is split along the center of mass, and the refinement is recursively repeated, until the boundary planes of the remaining triangles are used to obtain the final approximation.

3.1 B-rep Streams

It is fairly straightforward to set up a producer process importing a B-rep, externally defined by some standard polygonal format, e.g. either a wavefront and java3D obj file, into an input stream for our geometric pipeline. The boundary representation given by polygons and normals must be *coherently oriented*. A filtering of the input file to cope with nonplanar polygons and other geometric inaccuracies may be needed for generally archived geometric models used primarily in computer graphics [23]. The output stream of coherently-oriented triangles, is then converted into our twin progressive-BSP trees by the algorithmic steps described below.

3.2 B-rep to BSP Algorithm Outline

A primary technique of our method is the computation of the inertia of triangle subsets by contraction of the pre-computed inertia of each triangle, and the eigendecomposition of the inertia of triangle subsets to bound their shape optimally and recursively.

Note that, in the *d*-dimensional case, the shape confinement is obtianed using 2 extremal tangent hyperplanes for each of the *d* eigenvectors of the Euler matrix. The intersection of the corresponding *2d* hyperspaces produces the best-fitting (hyper)parallelepiped of the boundary subset contained in the current cell. In 3D, there are 6=2x3 such planes.

Initialization

(a) The affinely extended Euler tensors [9] of each input triangle are first computed (in linear time).
(b) The entire set of input triangles is associated with the BSP root.
(c) The entire E^3 space (that is convex) is associated to the root.
(d) The label of the root is set to FUZZY.

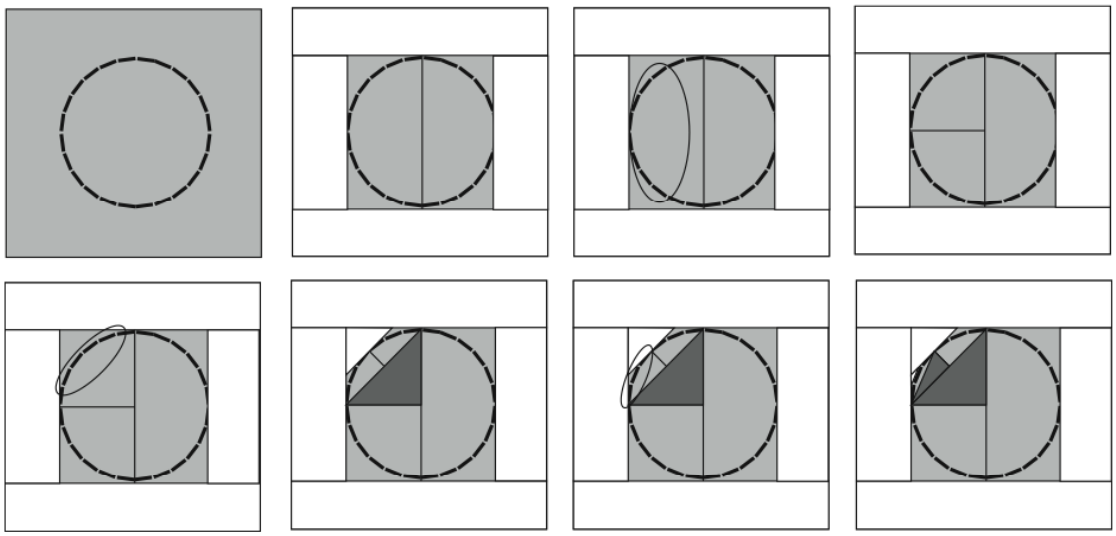

Figure 5: Progressive construction of the BSP in 2D starting from edge boundary elements. OUT cells are white, IN cells are black, and FUZZY (i.e. yet undecided) cells are grey. The ellipsoids of inertia of edge subsets are also drawn.

Recursive case

(a) The current FUZZY cell is split by at most 6 orthogonal hyperplanes that are normal to the eigenvectors of the matrix representation of the Euler tensor of the current triangle subset.
(b) Such planes are computed via the minimum and the maximum value of the linear function $w = a \cdot v$ evaluated on the vertices v of the current triangle subset, where a is the current eigenvector.
(c) For each eigenvector, at most three convex cells are produced by two max-min parallel hyperplanes, that are either {OUT, FUZZY, OUT} or {OUT, FUZZY, IN}.
(d) Each FUZZY cell is further split by the principal hyper-plane associated to the maximum eigenvector.
(e) A smaller subset of triangles is associated to each split cell via containment test of their vertices.
(f) Triangles crossing a splitting plane are split, and the (sub)triangles are associated to node subtrees.

Basic case
The recursive inertia-based division stops when the current cell only contains a small number of boundary triangles. A final cell splitting is executed using the planes of the boundary triangles.

3.3 Fast Inertia Computation
Mass and inertia of compact point-sets (curves, surfaces, volumes) are defined as integrals of certain scalar fields $f : E^3 \rightarrow R$ over the point-set. If $S \subset E^3$ is a surface, then its *mass M*, *first moments* M_x, M_y, M_z, *second moments* M_{xx}, M_{yy}, M_{zz}, and *products of inertia* M_{yz}, M_{xz}, M_{xy} are defined as

$$\int_S f(x,y,z)\,dS = \int_S x^\alpha y^\beta z^\gamma dS = I_S^{\alpha\beta\gamma}$$

where the scalar field *f(x,y,z)* is respectively equal to 1 for *mass*; to *x*, *y* and *z* for *first moments*; to x^2, y^2 and z^2 for *second moments*; and to yz, xz and xy for *products of inertia*. The *centroid* or center of mass $g= (g_x, g_y, g_z)$ is defined by the ratios

$$g_x = \frac{M_x}{M}, \quad g_y = \frac{M_y}{M}, \quad g_z = \frac{M_z}{M}$$

of the first moments to the mass. For *homogeneous* point-sets, where the density is constant, the centroid depends only on the geometry. In conclusion, an integral over an entire surface *S*, open or closed, as well as over every subset $S' \subseteq S$, is a summation of integrals over the triangles τ S': please add epsilon

$$I_{S'}^{\alpha\beta\gamma} = \int_{S'} x^\alpha y^\beta z^\gamma dS = \sum_{\tau \in S'} I_\tau^{\alpha\beta\gamma}.$$

The integrals described above for a point-set *S* can be arranged into a *4x4* matrix, that represents by components the *affinely extended Euler tensor* [9] of *S*,:

$$I_S = \begin{pmatrix} M_{xx} & M_{xy} & M_{xz} & M_x \\ M_{yx} & M_{yy} & M_{yz} & M_y \\ M_{zx} & M_{zy} & M_{zz} & M_z \\ M_x & M_y & M_z & M \end{pmatrix} = \begin{pmatrix} I^{200} & I^{110} & I^{101} & I^{100} \\ I^{110} & I^{020} & I^{011} & I^{010} \\ I^{101} & I^{011} & I^{002} & I^{001} \\ I^{100} & I^{010} & I^{001} & I^{000} \end{pmatrix}$$

A reference frame with origin in the surface centroid and with the first three eigenvectors of the tensor as its fundamental basis is called the surface's *principal frame*. In the principal frame, products of inertia and first moments are zero and the tensor I_S is diagonal. Since integrals are additive with respect to the integration domain, the inertia tensor of *S* is the sum of tensors of triangles in *S*. A *very* fast procedure for the computation of the Euler tensor of triangulated surfaces is given in [9]. There is a ten-fold time improvement of such an approach over explicit integration methods [6]. It is about a thousand fold if implemented on a modern GPU [15]. E.g. the Euler tensor of the cow model was computed in 0.17 seconds, on a triangulation with 5804 triangles, using an Intel Centrino 2.00Ghz with 1.047.784 KB of RAM and MS Visual C++ 6.0. The average computation on several models is greater than 35,000 triangles/sec.

3.4 Algebraic Details
With some abuse of notation, let us denote the generic input triangle as $\tau \in S$. In the following, and without loss of generality, we denote as $S_C \subset S$ the subset of input triangles contained in the current FUZZY cell *C* of the progressive BSP-tree.
Suppose that the Euler matrices of triangles in S_C have already been computed. S_C may be either closed, i.e. may be the boundary of a solid, or an open surface. Let M_C denote the symmetric and positive definite (by definition) affinely extended Euler tensor of S_C, or better, the coordinate representation (i.e. the matrix) of such tensors in world coordinates. So we have:

$$M_C = \sum_{\tau \in S_C} M_\tau$$

Since the matrix M_C is symmetric, and positive definite, its eigenvalues are all reals and positive. The ratios of elements of the 4-th row to the element m_{44} give the affine coordinates of the centroid g_C of triangles contained in *C*. The first three normalized eigenvectors give an orthonormal basis of E^3 where the inertia matrix is diagonalized.

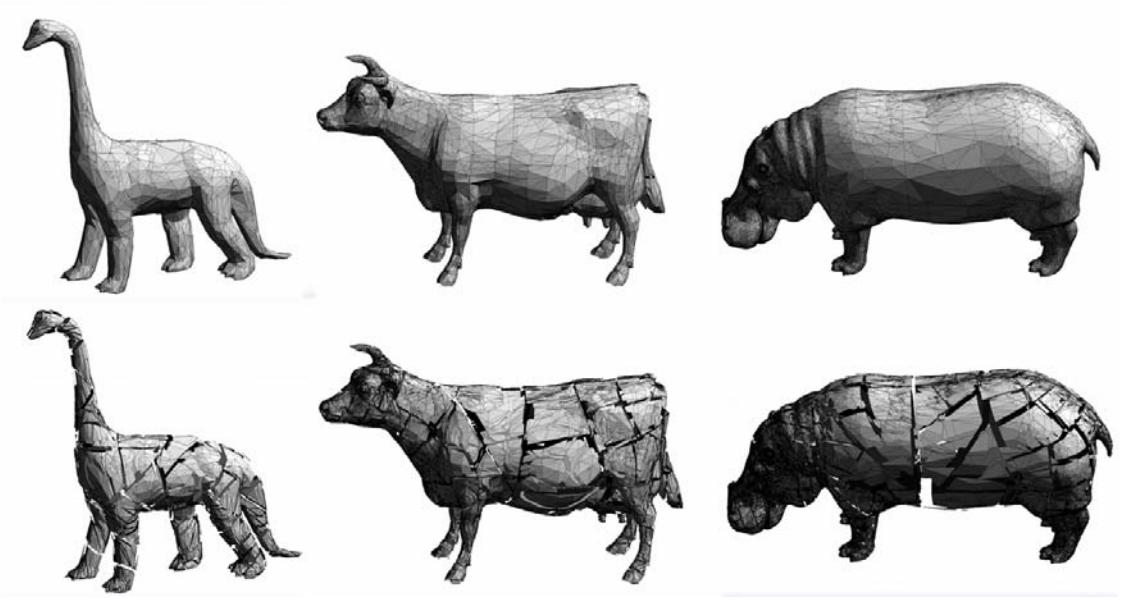

Fig. 6. BSP solid models from B-rep triangulations, and exploded views of the cell complexes.

Consider the linear function $w : E^3 \rightarrow R$ such that w(p) = p.a, where a is the eigenvector of M corresponding to the minimum eigenvalue. It is possible to show, that a linear function take maximum and minimum value over extremal (vertex) points of a polyhedral point-set [7,14]. Hence, find a pair of minima and maxima points of w(S) over the set V(S) of vertices of S. Lets call them v_{min} and v_{max}, respectively. Clearly this task is O(n), if n=|V(S)|.

Split the cell C in at most three parts using the two parallel hyperplanes

$$p.a - w(v_{min}) = 0 \quad \text{and} \quad p.a - w(v_{max}) = 0.$$

The cell will be split in only two parts if one of the hyperplanes contains a boundary facet of C. No less than two parts may arise, one of which certainly to be labeled as IN or OUT, by construction. If no splitting is possible, the entire cell C is certainly empty (OUT) or solid (IN), and the progression in the current cell will stop.

If at least one cell splitting has occurred, and *hence* one of sub-cells is FUZZY, let continue splitting the generated FUZZY sub-cell by using the max and min hyperplanes parallel to the remaining eigenvectors previously computed. The result is a confinement of the geometric data into a smaller (mimimal) rotated parallelepiped, parallel to the principal frame of the current triangles. Finally, split such FUZZY cells with the hyperplane orthogonal to the maximum eigenvector and passing for the center of mass:

$$p.a - w(g(\tau)) = 0$$

3.5. Eigendecomposition of the Euler Matrix

The upper-diagonal 3x3 submatrix of the affine inertia matrix has several nice properties. It is small, dense, symmetric and positive-define, so that we can safely use the simplest algorithms for computation of eigenvectors, i.e. the *direct* and *inverse power* methods [10]. First, only the minimum and maximum eigenvector must be computed iteratively, since the third one may be derived by their vector product. Both methods are iterative, and work by repeatedly applying either the matrix *M* or the inverse M^{-1} to an initial trial vector, say (1,1,1), to successively yield an approximation of the maximum (minimum) eigenvector. This process may be accelerated by applying a number of squaring operations to the initial matrix *M*. For example, after 5 compound applications of the squaring operator to *M*, the resulting matrix is $M\text{^}2^5 = M^{32}$. Hence, a single multiplication of the trial vector by it gives the same result that 32 iterations of the multiplication by *M*.

Fig. 7. Transformation from B-rep triangles to BSP: (a) progressive generation of model from triangles; (b) exploded view of the produced BSP; (c) a close view of the head.

5. CONCLUSIONS

We have presented an optimal conversion from triangled B-Rep to a *BSP* solid representation. The algorithm has an O(*n*) preprocessing, and is used to compute the inertia tensor of each triangle, followed by an O(*n* log *n*) construction phase of the balanced (progressive) *BSP*. Furthermore, the algorithm is well suited to GPU implementations [15], to stream rendering in combination with other geometric operations, and in particular to perform Boolean set operations on-the-fly [18]. The prototype system is presently implemented as a multithreaded library written in C and named XGE, for eXtreme Geometric Environment. An integration with the PLASM design language has also been started, with the aim of compiling the user functional environment into a distributed dataflow pipeline which would exploit the available resources on demand. All the main techniques of geometric and solid modeling are well supported by this stream processing technology. The next step will concentrate on: (a) porting the library on the *cell processor* architecture, that appears to perfectly fit our streaming approach, and can be implemented using the Stanford's Brooke streaming extension of the C language (see [4,5] and [8]); and (b) a close integration of solid and physical modeling, with the goal of supporting progressive simulations and adaptive, simulation-driven refinements of the generated mesh.

The research of the first author was supported in part by grants from the National Science Foundation ITR-EIA-032550, DDDAS-CNS-0540033, and grants from the National Institutes of Health, NIH-P20-RR020647-01, NIH-R01-GM074258-021. The research of the last two authors was supported in parts by IBM Corp. through a SUR (Shared University Resources) Award and PLM-Lab donation to Roma Tre, and the Grant 2001-2004 from Spike Consortium, under MIUR Italian National Project SPI-09.

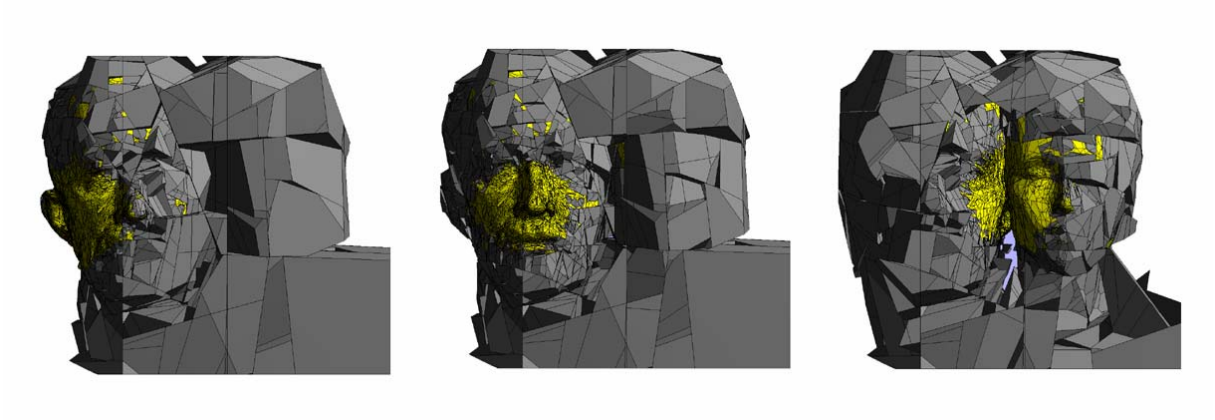

Fig. 8. Progressive transformation from B-rep triangles to BSP: adaptive refinement

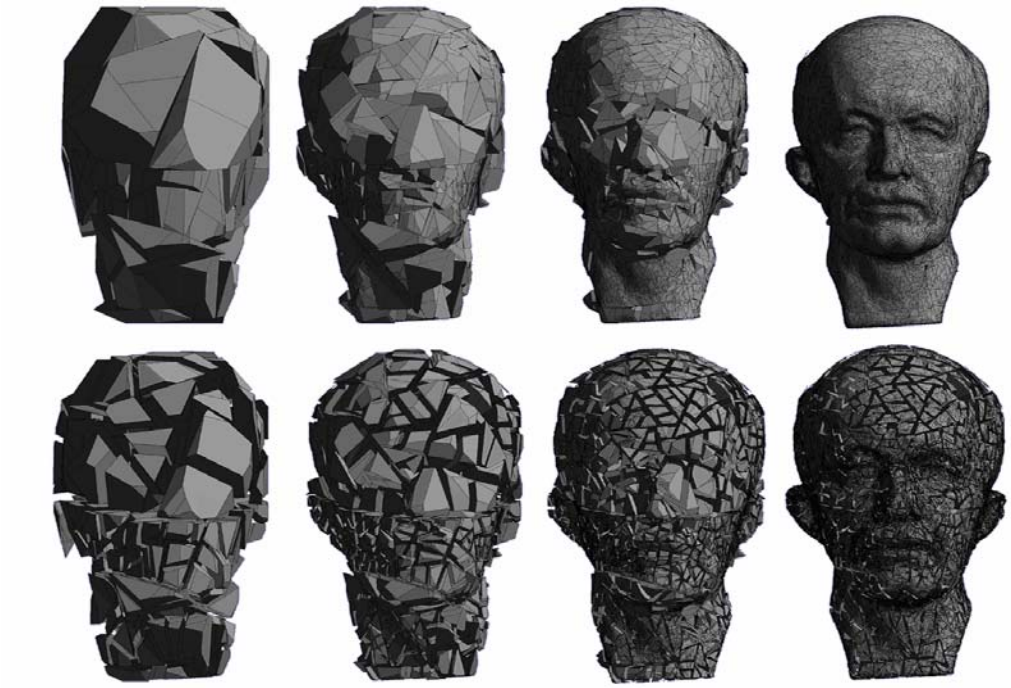

Fig. 9. Progressive transformation from B-rep triangles to BSP:
(a) progressive solid models of the Max Plank head; (b) exploded views of the produced BSP.

6. REFERENCES

[1] Bajaj, C. L., and Dey, T. Convex Decompositions and Robustness, *SIAM J. on Computing*, 21, 2, (1992), 339-364.

[2] Bajaj, C. L., and Pascucci, V. Splitting a complex of convex polytopes in any dimension. In SCG '96: Proc. of the *12th Symposium on Computational Geometry* (1996), ACM Press, pp. 88–97.

[3] Bajaj, C. L., Pascucci, V., and Zhuang, G. Progressive compressive and transmission of arbitrary triangular meshes. In VIS '99: Proc. of *Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 307–316.

[4] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. Brook for GPUs: stream computing on graphics hardware. *ACM Trans. Graph*. 23, 3 (2004), 777–786.

[5] Buck, I., and Purcell, T. A toolkit for computation on GPUs. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, F. Randima, Ed. Addison-Wesley, 2004.

[6] Cattani, C., and Paoluzzi, A. Boundary integration over linear polyhedra. *Comput. Aided Des.* 22, 2 (1990), 130–135.

[7] Chvatal, and Vasek. *Linear Programming*. W. H. Freeman and Company, New York, 1983.
[8] Dally, W. J., Labonte, F., Das, A., Hanrahan, P., Ahn, J.-H., Gummaraju, J., Erez, M., Jayasena, N., Buck, I., Knight, T. J., and Kapasi, U. J. *Merrimac: Supercomputing with streams*. In SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing (Washington, DC, USA, 2003), IEEE Computer Society, p. 35.
[9] DiCarlo, A. and Paoluzzi, A. Fast computation of inertia properties through affinely extended Euler tensor, (Submitted paper) 2005.
[10] Gerald, C.F., and Wheatley, P.O. *Applied Numerical Analysis*. Addison-Wesley, NY, 1998.
[11] Lario, R., Pajarola, R., and Tirado, F. Hyperblock-quadtin: Hyper-block quadtree based triangulated irregular networks. In Proc. *IASTED Visualization, Imaging and Image Processing* (2003).
[12] Milicchio, F., Paoluzzi, A., and Bertoli, C. A visual approach to geometric programming, *Computer-Aided Design & Applications*, Vol. 2, Nos. 1-4, *CAD'05*, 2005, pp 411-420.
[13] Murty, K. G. *Linear Programming*. John Wiley & Sons, New York, NY, 1983.
[14] Naylor, B. F. Binary space partitioning trees as an alternative representation of polytopes. *Computer Aided Design* 22, 4 (1990), 250–252.
[15] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krger, J., Lefohn, A. E., and Purcell, T. J. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports* (Aug. 2005), pp. 21–51.
[16] Pan, Z., Tao, Z., Cheng, C., and Shi, J. A new BSP tree framework incorporating dynamic lod models. In VRST '00: Proc. of the *ACM symposium on Virtual reality software and technology* (2000), ACM Press, pp. 134–141.
[17] Paoluzzi, A. *Geometric Programming for Computer Aided Design*. John Wiley & Sons, Chichester, UK, 2003.
[18] Paoluzzi, A., Pascucci, V., and Scorzelli, G. Progressive dimension-independent Boolean operations. In Proceeding of the 9-th *ACM Symposium on Solid Modeling and Applications* (2004), G. Elber, N. Patrikalakis, and P. Brunet, Eds., pp. 203–212.
[19] Pascucci, V., and Bajaj, C. L. Time critical isosurface refinement and smoothing. In VVS '00: Proceedings of the 2000 *IEEE Symposium on Volume Visualization* (New York, NY, USA, 2000), ACM Press, pp. 33–42.
[20] Scorzelli, G., Paoluzzi, A., and Pascucci, V. Parallel Solid Modeling Using BSP Dataflow. (2006). To appear on *Int. Journal of Computational Geometry & Applications*.
[21] Stephens, R. A survey of stream processing. *Acta Informatica* 34, 7 (1997), 491–541.
[22] Wiley, C., A. T. Campbell, I., Szygenda, S., Fussell, D., and Hudson, F. Multiresolution BSP trees applied to terrain, transparency, and general objects. In Proc. of *Graphics interface '97* (Toronto, Ont., Canada, 1997), Canadian Information Processing Society, pp. 88–96.
[23] Tao-Ju, Robust Repair of Polygonal Models*, Proceedings of ACM SIGGRAPH, 2004 ACM Transactions on Graphics, 23(3):888-895.*
[24] Eric W. Weisstein. "Hasse Diagram." From *MathWorld*--A Wolfram Web Resource. Url of the specific page: http://mathworld.wolfram.com/HasseDiagram.html