Computer-Aided Design 77 (2016) 98–106

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Out-of-core real-time haptic interaction on very large models

A. Aguilera[a], F.J. Melero[b,*], F.R. Feito[a]

[a] Dpt. Informática, Univ. Jaén, Spain

[b] Dpt. Lenguajes y Sistemas Informáticos, Univ. Granada, Spain

ARTICLE INFO

Article history:
Received 28 September 2015
Accepted 4 April 2016

Keywords:
Haptic rendering
Collision detection
Hierarchical bounding volumes

ABSTRACT

In this paper we address the problem of fast inclusion tests and distance calculation in very large models, an important issue in the context of environments involving haptic interaction or collision detection. Unfortunately, existing haptic rendering or collision detection toolkits cannot handle polygonal models obtained from 3D digitized point clouds unless the models are simplified up to a few thousand polygons, which leads to an important lack of detail for the scanned pieces. We propose a data structure that is able to manage very large polygonal models (over 25M polygons), and we explain how this can be used in order to compute the inclusion of a point into the solid surface very efficiently, performing several thousand point-in-solid tests per second. Our method uses a data structure called EBP-Octree (Extended Bounding-Planes Octree), which is a very tight hierarchy of convex bounding volumes. Based on a spatial decomposition of the model using an octree, at each node it defines a bounding volume using a subset of the planes of the portion of the polygonal model contained at that node. We use the EBP-Octree in a haptic interaction environment, where distance tests and the orientation of collided triangles must be accurate and fast. We also demonstrate that the proposed algorithm largely meets the interactive query rate demanded by a haptic interaction (1 kHz), despite being executed in a single CPU thread on a commonly available computer.

1. Introduction

Haptic interaction in virtual environments has certain time restrictions that require much more efficient data structures and algorithms than other collision detection techniques. When a user handles a 6 degrees-of-freedom (DOF) haptic device, he expects to perceive the *real* contact with the surface, and not just an approximation of it. It is generally accepted that a minimum update rate of 1 kHz is required for the collision detection thread to provide continuous (i.e. realistic) feedback [1]. Hence, the algorithm must be able to dispatch every collision or distance query in less than a millisecond. It is not straightforward to achieve this for very large polygonal models obtained from 3D scanners. In the context of art curators, for example, it is not conceivable to use simplified models of a few thousand polygons (like those usually used in traditional haptic applications) to virtually plan or test the restoration process of a sculpture.

We address the problem of fast inclusion tests and distance calculation in very large models by developing a system that performs several thousand point-in-solid and distance tests per second on models over 25M polygons. Our method uses a data structure termed EBP-Octree (Extended Bounding-Planes Octree). This is a very tight hierarchy of convex bounding volumes that, supported by a spatial decomposition of the model using an octree, defines a bounding volume at each node using a subset of the planes of the portion of the polygonal model contained at that node. To summarise, our main contributions are:

- A data structure, the EPB-Octree, which can virtually host models of unlimited-size. Our tests only used level 11 of 20 using models of 30 million polygons.
- A point-in-solid test that runs comfortably at a haptic rendering frame rate when applied to very large polygonal models, achieving from 2.5 kHz (purely random point test) to 32 kHz (haptic-like paths).
- A distance and contact normal function that allows us to use the EBP-Octree in haptic applications designed for training or planning for virtual curators that runs at around 1 MHz in classic haptic interaction over the surface of models over 25M polygons.

We tested the EBP-Octree in a simulated haptic interaction environment, where distance tests and the orientation of collided triangles must be accurate and fast. In addition, we demonstrate

* Corresponding author.
E-mail addresses: angel@ujaen.es (A. Aguilera), fjmelero@ugr.es (F.J. Melero), ffeito@ujaen.es (F.R. Feito).

http://dx.doi.org/10.1016/j.cad.2016.04.002
0010-4485/© 2016 Elsevier Ltd. All rights reserved.

that the proposed algorithm largely meets the interactive query rate demanded by a haptic interaction (1 kHz), despite being executed in a single CPU thread on a home-featured computer. We tried to test our models using other state-of-the-art approaches, but none of the packages and libraries we know were able to handle such large models.

Section 2 reviews the most relevant prior work, whereas Section 3 presents the data structure and its related construction algorithms. Section 4 specifies the out-of-core caching mechanism that allows us to perform point-in-solid and distance tests, the description and experimental results of which are presented in Sections 5.1 and 5.2 respectively.

2. Related work

Haptic Rendering is a term used to describe the process of calculating a reaction force for a given position of the haptic feedback device. It is closely related to collision detection algorithms, and this area of research has been extensively investigated by the graphics community. Lin et al. [2] introduce collision detection concepts and offer an overview of existing algorithms and data structures. However, collision detection algorithms are not directly applicable to haptic rendering, since it is not only necessary to detect collisions, but also to compute distances and normals at a high frequency rate (around 1 kHz). The simplest way of approaching haptic rendering is to consider a point-based device, as described in [3] when explaining the *contact levels of detail* (CLODs) model. With this approach, the main task is to compute distances from the haptic 3D cursor to the model's surface. When the distance is below a preset threshold, the haptic rendering thread presents a given force, calculated to avoid penetrating into the model.

The well known PQP package developed by Larsen et al. [4] uses swept sphere volumes as a bounding hierarchy for triangle clouds to detect collision between models. However, there is no possibility of testing its performance on very dense meshes. In [5], Gregory et al. present the basis for the HCollide method, using quite small models. A solution proposed by Otaduy in [6] handles models below 50k polygons.

Distance fields have been used extensively for collision detection and rapid distance computation. There are three main approaches to build distance fields: those based on Voronoi diagrams [7,8], on distance propagation methods [9] or techniques that use trees and grids as supporting data structures. Among the latter, a three-dimensional grid is used to store the distance field in [10], while an Adaptive Distance Field using an octree was developed in [11]. A quite original approach is provided by the *haptic textures* of Theoktisto et al. [12]. This proposes a texture-based normal mapping of the surface in a similar manner as bump mapping does for rendering in order to obtain a fast distance query rate.

McNeely [13] implements a voxelized model of the surface, the Voxmap, to give approximate distance values in a 6-DOF haptic environment, testing it in a 3-DOF haptic environment with a model of 593k polygons. In [14], Barbic et al. presented a CPU-based method that uses the Voxmap point shell to create distance maps for deformable models. In this work, the authors state that "only small point shells fit into the computational budget of one haptic cycle". They then propose using a hierarchy that handles models of up to 256k points. The work by Gueziec [15] generates a multiresolution hierarchy of bounding volumes via geometric simplification of the polygonal model in order to dynamically compute the distance from a point to an arbitrary polygonal mesh. However, the largest model tested is composed of 60k polygons. Similarly sized models can be found in recent papers based on GPU and parallel programming, as in the case of Lauterbach's work [16] where a parallel implementation of OBB (*Object Oriented Bounding Box*) and rectangular swept spheres runs over models of up to 75k triangles, and the CPU/GPU-based approach of Pabst et al. [17], where the largest model contains 146k polygons and the continuous collision detection computation time is 184 ms for this model in a single thread. Morvan et al. [18], also using a GPU approach, handle models of up to 1.7M polygons, offering proximity query rates of around 5 ms.

In [19], Walker et al. describe how to perform haptic interaction on huge terrain models (100M triangles, 2.5D), but their technique is not usable on 3D models, as they use parallel computer vision algorithms to detect collision by projecting the proxy onto the terrain image. The proposal by Yoon [20] runs on models similar to those presented in our paper, but their goal was to achieve interactive rendering frame rates (12–30 frames per second, 18 ms per collision query), not haptic rendering frame rates.

3. EBP-Octree data structure

Our proposal was inspired by the BP-Octree data structure [21], originally conceived for progressive visualization, which guided our work to create a data structure suitable for collision detection. The most characteristic feature of this data structure is that each node stores a set of planes that define a convex bounding volume of the part of the model contained in that node. These planes are restricted to be either face planes or planes parallel to faces, so it is possible to maintain as much of the original surface orientation as possible. This data structure leads to a tighter bounding volume than other BVHs, e.g. KDOPs or AABBs, as the plane's orientation is unrestricted and the number of planes at each node is not predefined.

Our EBP-Octree building process, described in the following subsections, is based on the steps described by Melero et al. in [21]. In addition, several new features and algorithms have been developed in order to achieve our goal of handling huge polygonal models at interactive haptic query rates. Noteworthy among these improvements are: the extension to a 64-bit octcode, the detection of special configurations and white/black nodes, the management of the temporary file system that handles the huge amount of geometric data computed during the EBP-Octree construction, and the cache-like out-of-core management of the tree. This means it can be used in haptic interaction environments, computing not only point-in-solid tests but also the distance and orientation of collisions.

3.1. Computing bounding volumes at leaves

Following the BP-Octree bottom-up construction algorithm, we define the deepest level of the tree as the level whose cell size is at least five times the average triangle edge length. Then, using that three dimensional grid, we apply an exhaustive 3DDDA (3-Dimensional Digital Differential Analyzer) algorithm to each polygon to detect any traversed cell, using *Morton codes* [22] to locate every cell (i.e. leaf node) in the octree space. In our new proposal, the *octcode* is 64 bits long, which allows us to handle octrees of up to 19 levels: 57 bits for the code itself and 5 bits for the level that the code belongs to. This makes the EBP-Octree capable of holding polygonal models of almost any size, limited only by the computer's available disk space, while the BP-Octrees 32 bit octcode limits the size of the input models to about 10M polygons (9 levels).

Once the polygons are distributed among the leaf nodes, we compute the bounding volume (illustrated in two dimensions in Fig. 2) in the same manner as in [21]. In order to recall briefly how this works, we describe the process visually in Fig. 2:

- At each leaf node n_l we define a set of *candidate planes*. These *candidate planes* are the set of supporting planes of each polygon of the node n_l.

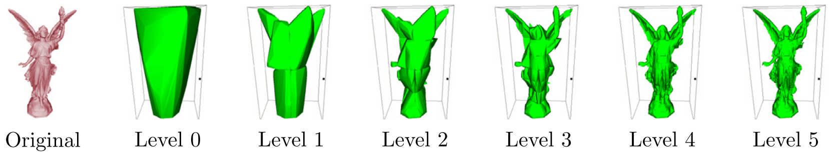


Fig. 1. First five levels of the Stanford Lucy model (28M Polygons) EBP-Octree. The complete EBP-Octree for this model has 11 levels.

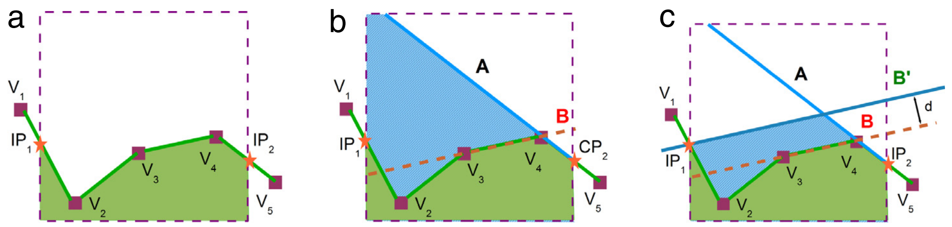


Fig. 2. (a) The *Boundary vertices* set is composed of vertices inside the node (V_2, V_3, and V_4) and the intersecting points at node edges (IP_1 and IP_2). (b) Plane A is selected as *candidate plane* because it encloses all of the *boundary vertices*. (c) Plane B must be displaced to B' in order to enclose all the points. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)
Source: Adapted from [21].

- The position of every candidate Plane CP_i is evaluated against the named *boundary vertices* of the node n_l. This set of *boundary vertices* is composed of the points inside the node (V_2, V_3, and V_4 in the case of Fig. 2) and the intersecting points (IP) that are generated between the edges of the node and the triangles, and between the edges of the triangles and the node faces (IP_1 and IP_2 in Fig. 2).
  - If all the *boundary vertices* lie inside or on the Plane CP_i, i.e., the signed distance is *less than* or *equal* to zero, the Plane CP_i is inserted into the *enclosing planes set (EPS)*. In Fig. 2(b), Plane A is the only plane that encloses all of the boundary vertices. The resulting bounding volume is depicted in blue.
  - If the plane leaves any boundary vertex in the *outside* half-space, we also insert a *displaced plane* into the *EPS*. An example of such a *displaced plane* is Plane B' in Fig. 2(c). Note that vertex IP_1 initially lies outside B (2(b)), but when the plane is displaced to position B' (by applying an offset d along its normal), IP_1 is then in the inner halfspace of B'. Displacement d is exactly the distance from IP_1 to B.

3.1.1. *Selecting the best bounding planes' configuration*

As in the BP-Octree, not all planes in the *EPS* are finally included in the *Bounding Planes* set. Some of them move completely outside the node n when attempting to satisfy the requirement of enclosing all points. Others, although initially included in the *EPS*, are discarded due to their similarity or poor contribution to the bounding volume.

Using the previously described algorithm, most or even all of the candidate planes might become part of the *EPS*, so we would have no simplification in convex areas of the surface. In order to reduce the number of planes of the *EPS*, we apply a *k-medoids* algorithm [23] to determine which planes best enclose the original surface, avoiding as many redundancies as possible by grouping the planes by their plane normals so that we can discard planes with similar orientations and select only the most relevant k planes. In Fig. 3, we illustrate the behaviour of this classification scheme with a sphere, where all planes are initially included in the *EPS*. When only a small portion of them are selected, the bounding volume built by the intersection of all inner half-spaces of the bounding planes set still looks like a sphere. This algorithm is quite slow, but after having tried other approaches such as random selection or filtering by the displacement distance, we concluded that the benefits in terms of volume are much greater than the cost of a single building execution process. In Table 1 we show the total volume of the EBP-Octree at each level with different k values.

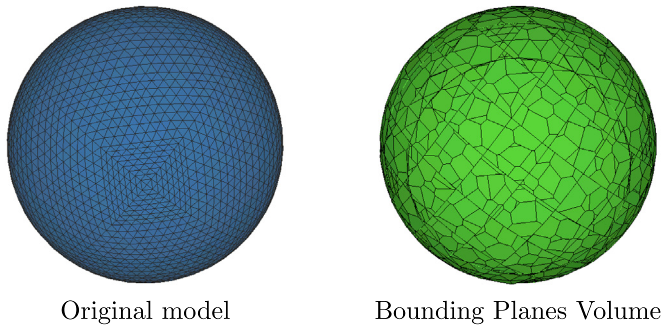


Fig. 3. Effects of applying a k-medoids algorithm over convex surfaces.

Table 1
Volume of the EBP-Octree of Wounded Amazon at each level depending on the value of the parameter k of the *k-medoids* algorithm.

Level	$k = 10\%$	$k = 20\%$	$k = 30\%$	$k = 40\%$
1	650.80%	653.46%	522.65%	325.78%
2	548.53%	624.12%	383.83%	261.15%
3	338.96%	335.47%	216.91%	160.11%
4	201.96%	199.57%	143.06%	124.92%
5	137.07%	135.68%	116.33%	109.93%
6	113.74%	112.85%	106.35%	104.37%
7	105.05%	104.45%	102.31%	101.70%
8	101.65%	101.36%	100.76%	100.58%
9	100.49%	100.37%	100.23%	100.18%
10	100.12%	100.09%	100.06%	100.05%
11	100.02%	100.02%	100.02%	100.01%

The concrete k value is different at each node, as we are defining it as a percentage of the size of each node's *EPS*. As expected, a higher value of k gives a tighter volume, but also a larger size of the complete structure.

3.1.2. *Adding in/out information to the EBP-Octree nodes*

A crucial new feature of the EBP-Octree with respect to its predecessor is that in order to perform point-in-solid and distance tests, there must be *white* and *black* nodes, i.e. nodes whose volume is completely outside (white) or inside (black) the polygonal model. Moreover, the EBP-Octree allows one point to be classified with respect to the original surface at leaves, not only with respect to the bounding volume. As the tree is built up with a bottom-up approach and the starting leaf nodes set is composed only of nodes traversed by the polygonal surface, we must first add some extra information to the leaves so we can classify any point inside the leaf

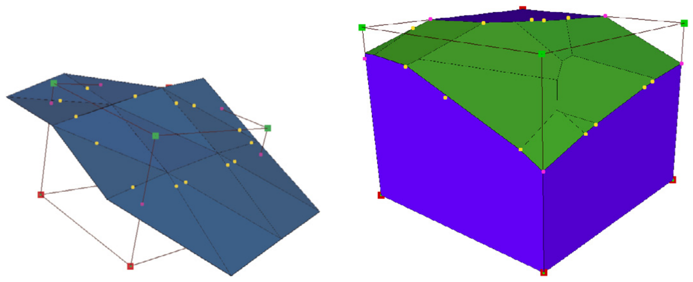

(a) Real surface. (b) Bounding volume.

Fig. 4. (a) Polygonal surface of the model. The *intersecting points* created by the intersection of the triangles' edges with the node's faces are displayed in yellow, and the *IPs* generated by the node's edges with the surface are rendered in purple. (b) Resulting bounding volume geometry. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

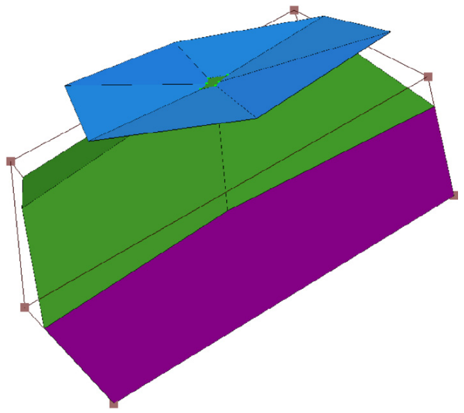


Fig. 5. Special case when a concavity does not intersect with node edges. The computed bounding volume is wrong if it does not take into account the fact that eight node corners are *inside* the polygonal model.

node with respect to the polygonal surface, without considering the rest of the polygonal model. We achieve this goal by labelling each node corner as *inside* or *outside*, depending on the orientation of the high resolution polygons that intersect with the node edges. As not every node edge is usually traversed by a triangle, some of the corners may remain unlabelled. Consequently, a propagation of the existing values is carried out until the eight corners are properly labelled.

Sometimes none of the 12 node edges are intersected by the polygonal model surface, as shown in Fig. 5. In this case, the surface of the model penetrates into the node without intersecting any node edge. Therefore, there is a set of vertices inside the node in a concavity and a few triangle-edge/node-face intersecting segments and points. With this configuration it is not possible to label the corners as explained above, and hence an alternative algorithm must be used.

The algorithm used to label corners in these special cases (similar to the *vertex nodes* in [24]) is based on analysing the configuration of the triangles that intersect the node face with respect to any of the four corners of that face, as depicted in Fig. 6. If the closest point of the intersecting polyline to Corner C is in a segment, as is the case for S_1 in Fig. 6(a), Corner C is classified with respect to the supporting plane of the triangle that creates intersecting segment S_1. If the closest point to Corner C is one of the vertices of the polyline (V in Fig. 6(b)), then vertex V_1 is classified with respect to the supporting plane that generates Segment S_2, and Corner C is labelled with the opposite value of the test.

3.2. Creating black and white nodes

The classification of the leaves corners is a key point added to the data structure with respect to the BP-Octree. It allows not only

Table 2
Volume enclosed by the EBP-Octree at each level (B-rep volume = 100%).

Level	Moulding	Wounded Amazon	Lucy
1	209.38%	325.78%	471.56%
2	161.50%	261.15%	261.35%
3	137.32%	160.11%	165.71%
4	118.06%	124.92%	128.80%
5	110.33%	109.93%	115.18%
6	106.89%	104.37%	107.67%
7	103.37%	101.70%	102.67%
8	101.26%	100.58%	100.84%
9	100.36%	100.18%	100.28%
10	100.09%	100.05%	100.10%
11	100.03%	100.01%	100.04%

special cases such as tunnels, peak convexities and concavities to be resolved, as shown in Fig. 5, but the feature also provides us with sufficient information to create *black* and *white* nodes. We realized that when dealing only with leaf nodes we do not have enough information to classify every point in the space, as in the case depicted in Fig. 7.

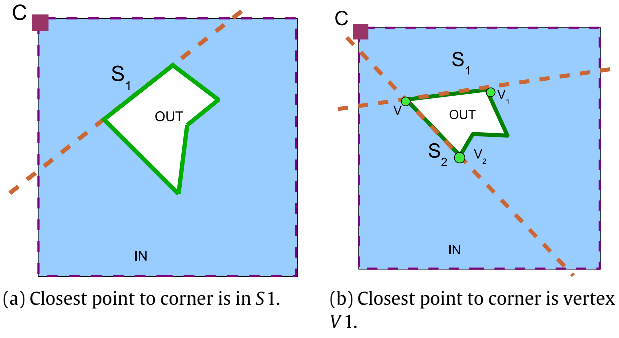
To avoid such classification errors, we use the information from the leaves' corners. We propagate such labelling upwards and create missing nodes according to the classification of the corners of sibling nodes. When an internal node has missing children because no surface traverses that cell, the algorithm decides whether to create it as a black or a white node depending on the value of the central point of the grey node. The central point of the parent node is the corner shared by all the siblings (Fig. 8).

3.3. Creating internal EBP-Octree nodes

Once we have selected the set of k planes whose inner half-spaces' intersection defines the bounding volume, we generate the geometry of this bounding volume (rendered in Fig. 4(b)). This geometry is used as a *virtual geometry* to run the algorithm described above at its parent node. We use the vertices of the *virtual geometries* of children nodes as *boundary vertices* of the parent node, and the sets of selected bounding planes as the *candidate planes* set. The bounding planes keep the displacement d as they ascend through the hierarchy, and that offset value is maintained or increased if needed (but never decreased) when computing the bounding volume at upper level nodes.

Following the process we have described, the final result is a hierarchy of bounding volumes that decreases in volume as the tree is traversed. The visualization of these volumes for the *Lucy* and *Wounded Amazon* models are shown in Figs. 1 and 9. To properly visualize the details at the deepest levels of the tree, we focus Fig. 10 on one branch of the tree, showing from level 6 to the deepest level.

Table 2 shows the volume enclosed by each level of the EBP-Octree (100% is the total volume of the original polygonal model). It can be seen that level 5 offers a volume just 15%



(a) Closest point to corner is in $S1$.
(b) Closest point to corner is vertex $V1$.

Fig. 6. Labelling node corners when there are no triangles intersecting node edges. In blue, the volume defined by the model. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3
Memory requirements per level of each EBP-Octree model (in MB).

Level	Moulding	Wounded Amazon	Lucy
1	0.12	0.19	0.12
2	0.30	0.40	0.38
3	0.65	0.97	0.96
4	1.59	2.90	2.64
5	4.54	8.41	7.18
6	13.12	24.07	20.77
7	79.05	139.46	127.09
8	241.96	416.66	392.06
9	735.71	1288.23	1216.01
10	2292.11	4196.32	3877.66
11	6993.68	13 292.65	11 954.45
Total	10 362.81	19 333.60	17 567.27

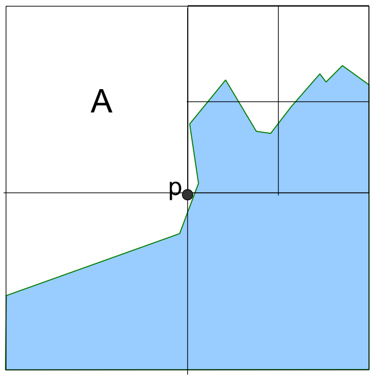


Fig. 8. A node is created as white because the corner shared among its siblings (p) was labelled as *outside* when computing the bounding volumes of those three nodes.

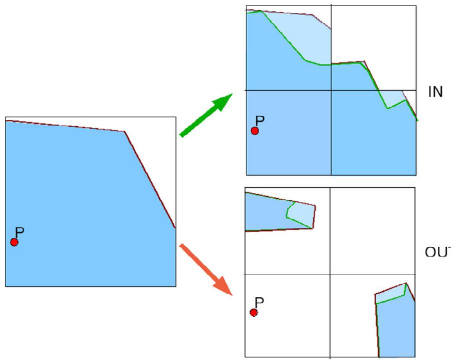


Fig. 7. Point P cannot be properly classified in the absence of black or white nodes because of the ambiguity arising due to the absence of corners labelling. Two distinct configurations (right) lead to the same bounding volume at the upper level (left).

Table 4
Preprocessing time of EBP-Octree running on a single thread. We compute the time needed to create leaves and the rest of the tree separately.

Model Millions of polygons	Moulding 26.62	Wounded Amazon 28.10	Lucy 28.05
Leaf nodes	50:53	59:59	3:57:00
Intermediate levels	18:39	23:20	1:16:31
Total Building Time (h:min:sec)	1:09:32	1:23:19	5:13:31

higher than the high resolution polygonal model, and the visual perception at that level also approximates the original model very well, as shown in Fig. 9. Moreover, in Table 3 we show the memory requirements for each level of the tree, which is rather small for upper levels of the tree, but exponentially larger for the bottom levels. For this reason, as we describe in Section 4.1, when performing haptic interaction we will always keep the EBP-Octree in the main memory up to some level between 5 and 7, and the subtrees from that threshold are loaded depending on the haptic interface position in a cache-like behaviour.

4. Out-of-core management of EBP-Octree

The whole data structure is built in a single out-of-core pre-processing step. The EBP-Octree is then stored on disk and loaded as many times as necessary. The construction times displayed for models of Fig. 11 are shown in Table 4. Most of the building time is spent on detecting which voxels are traversed by the model surface and on constructing its bounding volumes. The disparity in times is due to the different shape of each model, with the Lucy model being the most complex in terms of concavities and convexities.

In order to properly handle the EBP-Octree in real time interaction, it is absolutely necessary to have a set of files that indexes and organizes the information of the tree. Moreover, during the building process we use several temporary files. The EBP-Octree files set is composed of seven files which store all of the necessary information in a spatially and temporally coherent manner, and a variable number of files that store the subtrees below the threshold level. Some of the files are accessed during rendering duties, while others exist to aid in collision detection or distance computation tests. The files are listed briefly here:

- `.vtx`. The original polygonal model vertices.
- `.tgl`. Faces of the original polygonal model, addressing the vertices by their offset in.vtx file.
- `.fcn`. Normals of the original polygonal model, which also define normals of the planes at EBP-Octree nodes.
- `.geo`. Leaves' geometries. For each leaf, it stores the index of its triangles with respect to the `.tgl` file triangles list.
- `.bvp`. Bounding volume geometry. For each node of the tree, we store the faces and vertices generated when creating the

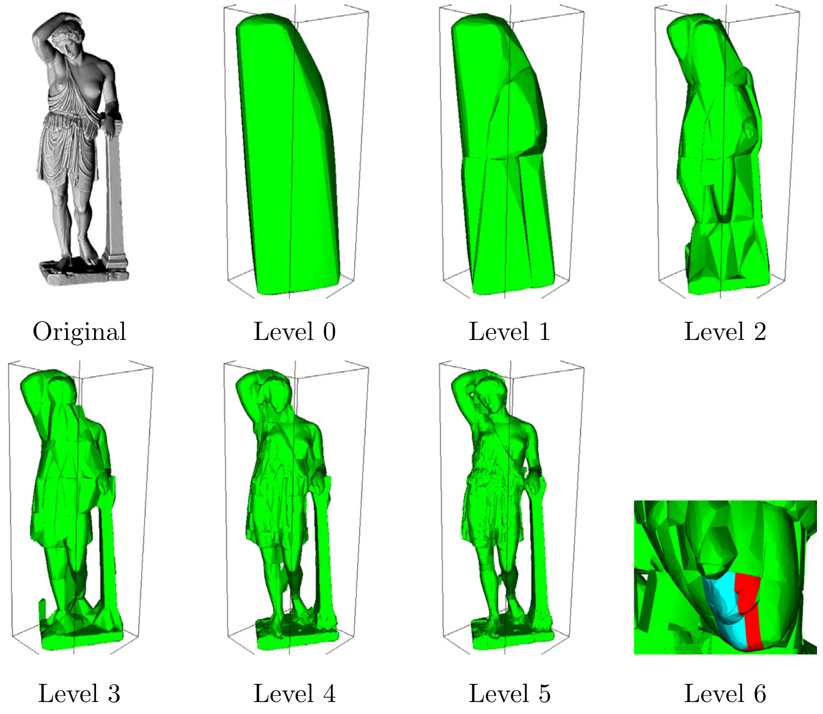


Fig. 9. Evolution of the *Wounded Amazon* EBP-Octree up to level 5 of 11. Deeper levels are loaded in a cache-like management system.

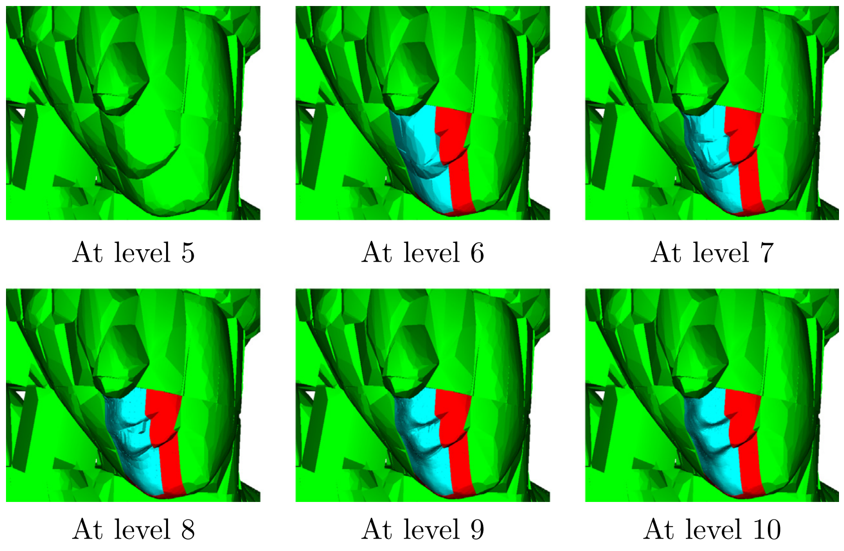


Fig. 10. Detail of the EBP-Octree of Wounded Amazon model during a haptic interaction. In green we display the bounding volume of nodes at level 5, which are always in the main memory. In cyan we show the bounding volume of the nodes of the currently active subtree, which is also loaded in the main memory. In red we mark the nodes that are *pre-cached*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

bounding geometry at each node. This information is only used when the bounding volume is rendered.

- `.bpl`. Bounding planes. For each node, we store the number of planes that create its bounding volume, their references to `.fcn` file and the offset *d* applied to that node. When visual rendering is going to be performed, it can also store the pointer to the `.bvp` file that returns the vertices of the bounding geometry.
- `.oct` and several `.suboct` files. The octree up to the threshold level and as many files as needed to store the sub-trees from that level. Nodes are stored in breadth-first order. For each node we store the information that describes it, depending on whether it is an internal or leaf node.
  An internal node is defined by:
  - Two bytes describing its children's type: if they are black, white, grey or leaf nodes (2 bits per child).
  - The offset needed to reach the first child within the file.
  - The offset in the `.bpl` file where its bounding planes are sequentially stored.

  Leaf nodes are defined in the `.oct` file by:
  - The offsets in the `.geo` file to the triangles of the model that intersect the node.
  - The offset in the `.bpl` file where its bounding planes are sequentially stored.

4.1. Cache management of the EBP-Octree during haptic rendering

We implemented a cache-like system that adapts to the features of the computer, where algorithms are running so that the

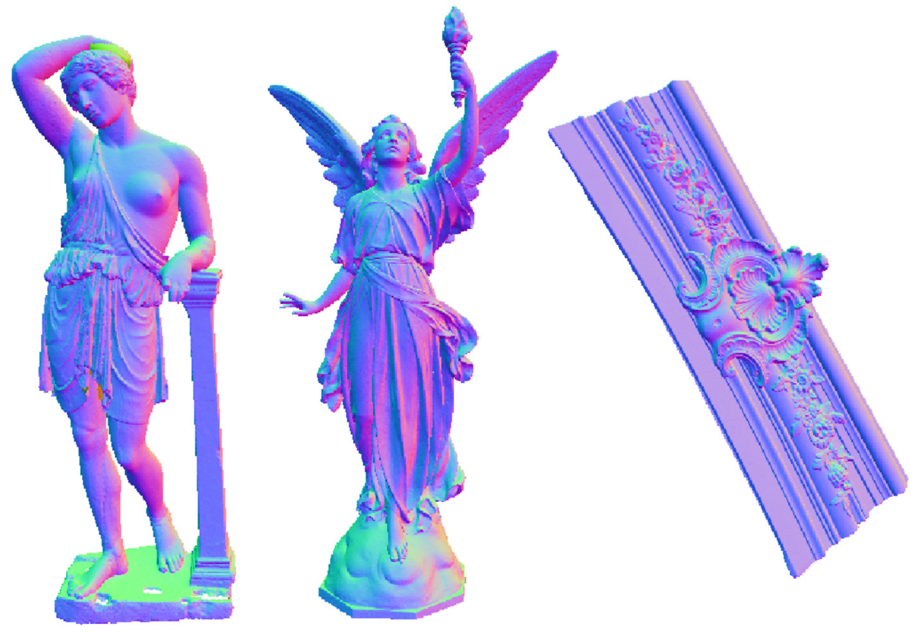


Fig. 11. High-res models used for testing the EBP-Octree: Wounded Amazon (26.6M), Lucy (28.1M) and Moulding(28.05M).

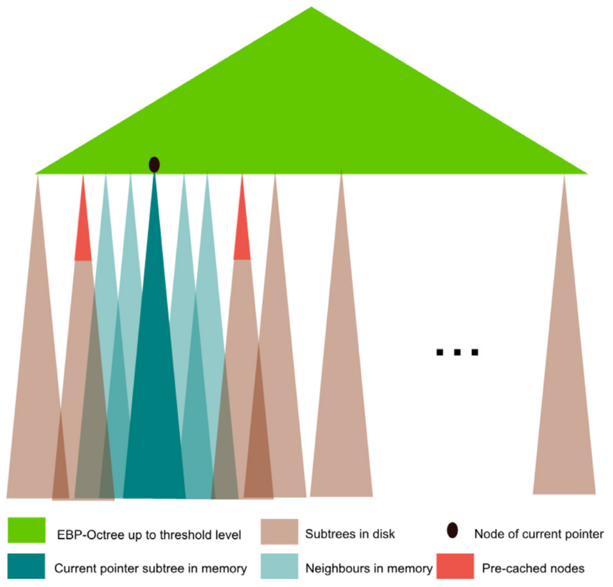


Fig. 12. Cache scheme for the EBP-Octree. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

EBP-Octree is not completely contained in the main memory when performing inclusion or distance tests. Depending on the memory available, the threshold level is at different points from levels 5 to 7, and an external file system is built (or rebuilt if needed) depending on that limit. Although we have optimized the behaviour of our cache system to the natural movements of haptic interfaces, which do not usually move around the model randomly but follow a natural and continuous path, it also behaves satisfactorily in purely random positions, as we will describe in Sections 5.1 and 5.2.

Following the scheme depicted in Fig. 12, we have the upper levels of the tree in the main memory up to level 5, which according to Table 2 ensures that the EBP-Octree occupies no more than 15% more volume than the polygonal model. While moving the haptic interface pointer (HIP), we detect the node n_p where the HIP is located. Then, we load the complete subtree under n_p into the main memory, and also the subtrees of the 26 neighbours of n_p. By doing so, at any time in the interaction there are therefore only 27 complete subtrees in main memory. Moreover, to minimize the memory faults when a fast displacement of the HIP is detected, we also load the nodes at level 6 of the nearest surrounding subtrees (in red in Figs. 12 and 10). As shown in Table 5, our algorithm uses on average 4.5 GB of memory.

Table 5
Some features of our cache system: number of nodes and memory occupied by the base EBP-Octree, the number of subtree files generated and the average total memory consumption in a distance test.

	Wounded Amazon	Lucy	Moulding
Nodes at level 5	5208	3969	1871
MB threshold EBPO	37.8	32.8	20.8
# subtrees	21 777	17 398	8254
Avg. memory (MB)	4522.54	4508.98	4296.2

5. Algorithms and experimental results

5.1. Point-in-solid test results

Using the EBP-Octree representation scheme, implementing a point classification algorithm is straightforward. To achieve this, we only need to traverse the tree, testing point P against the planes at each node. Only if P lies inside all of the inner half-spaces of the bounding planes of the current node, do we descend recursively into the appropriate child. When the algorithm reaches a *leaf* node Ln, it builds a segment between P and one of the corners of Ln labelled as *inside*, as shown in Fig. 13. Then, a classic segment-polyhedron test based on the Jordan Curve Theorem [25] is applied. If none of the corners are *inside*, we take an *outside* corner and the result given by the Jordan Curve Theorem is the opposite of what it would be if it were *inside*.

In order to evaluate the behaviour of the EBP-Octree as a supporting data structure for such large models in a point-in-solid test environment, we prepared two different datasets of 10M points, representing the worst and best cases. The first one is a pure random distribution of the points within the bounding box of the high resolution model. By doing so, we can test how the cache system behaves when loading 27 subtrees per test, which is the worst case. The best possible situation is to have a spatially coherent distribution of the points, i.e. we generate the points following a

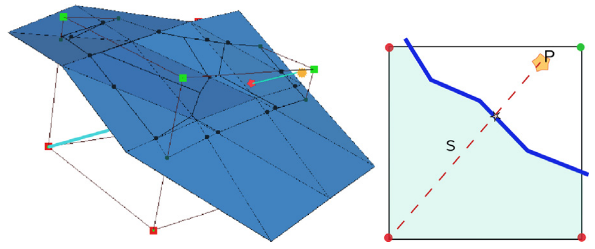


Fig. 13. Intersection of segment – cyan – with surface — dark blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 6
Point in solid tests per second, and µsecs per test.

	Moulding	Wounded Amazon	Lucy
Test/sec random	4922	2359	2524
Test/sec surface	31 559	30 529	43 430
µsec/test random	203.16	423.89	395.88
µsec/test surface	31.68	42.38	39.58

pre-defined trajectory through the high resolution surface with a minimum of random distortion. This is usually the normal behaviour of a haptic rendering environment, our best case, and it allows the cached subtrees to be reused in several consecutive tests.

In Table 6, we show the results of the point-in-solid tests performed on a I7 2.40 GHz 8 GB RAM computer. In the worst case, our proposal achieves 2.3 kHz frequency. Moreover, we can reach up to 32k tests per second on the Moulding model under normal haptic conditions. It is also worth noting that two similar models like Lucy and Wounded Amazon provide similar results whereas Moulding, which is a more compact model, gives better results, probably because most of the time not all 27 sub-trees are cached, as the model is quite flat in most areas.

5.2. *Distance-to-solid computation in large polygonal models*

In haptic environments it is necessary not only to define whether a point is inside or outside the virtual model, but also to determine the distance from the haptic devices end effector, the HIP. For this purpose, distance fields and other techniques have been shown to be useful in achieving good update rates. The EBP-Octree data structure allows us to obtain the distance to the model and the normal of the nearest triangle when the point to test is inside the narrow band defined around the polygonal surface by the leaf nodes.

In order to validate the EBP-Octree as an appropriate data structure to be used in haptic rendering, we prepared two datasets in a similar manner as we did for the point-in-solid computations: worst (random sequences within the root node) and best (natural interaction over the surface of the model) cases. To better illustrate the correctness of the results, we produced two different figures. Fig. 14 renders the distance from the point to the surface, with black being the colour of points below a given threshold. In Fig. 15, each collided point is coloured by using its nearest triangle normal as the RGB value, so we can validate visually the orientation returned by the collision test. In the figure, we show the full resolution model in normal map mode, and the collided points aside.

As for the timings, Table 7 shows the results of distance calculations using the two datasets described in Section 5.1, 2M points each. As can be seen, in the worst case we are over a haptic rendering query rate of 1.5 kHz, whereas in the continuous surface dataset we reach up to 27 kHz for the Lucy model. We must note that timings in distance tests also depend on the number of polygons at leaf nodes, and Lucy has the lowest average number of triangles per leaf node.

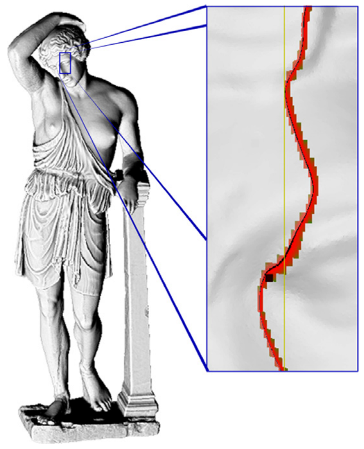


Fig. 14. Closer look at distance tests for face of Wounded Amazon. Black points are exactly on the surface. Red points are below a given threshold. Displayed points come from the datasets used in surface testing. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

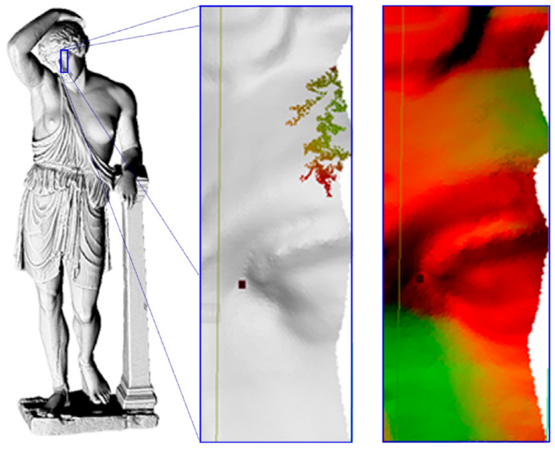


Fig. 15. Closer look at orientation returned by haptic rendering tests. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 7
Haptic tests per second, and µsecs per test.

	Moulding	Wounded Amazon	Lucy
Test/sec random	2853	1544	1697
Test/sec surface	18 020	15 981	27 984
µsec/test random	354.23	647.46	589.25
µsec/test surface	55.5	62.5	34.3

On the same computer, we compiled and executed some of the most cited libraries for collision detection, such as SWIFT++ [26], PQP [4] and gProximity [16]. Unfortunately, none of them was able to load the models that we used in our tests. SWIFT++ was only able to handle models up to 400k polygons, while PQP and gProximity crashed when trying to load models of 7M polygons.

6. Conclusions and future work

We have proposed a data structure capable of handling large polygonal models in a haptic rendering system, never used before in such environments. Point classification query rates are below 0.5 ms in the worst scenarios, and around 0.03 ms in surface paths. Distance queries in common haptic interaction scenarios are also below 0.07 ms, which by far exceeds the expected haptic rendering query rate of 1 ms/test.

In order to achieve these results, we have extended the BP-Octree proposed by Melero et al. by including new features and algorithms, such as:

- A 64 bit octcode that allows octrees up to 20 levels. This feature allows us to exceed the limit imposed by the 32-bit BP-Octree, which was only able to load models up to 2.5M polygons.
- A procedure for labelling node corners that allows us to include new black and white nodes.
- Out of core management of the octree construction, via a set of files that indexes the information and reuses it when possible.
- A cache-like algorithm that dynamically loads sub-trees when the haptic interface pointer moves, so the deepest level of the tree can be used to perform point-in-solid and distance tests.

Our algorithms run without any multi-thread optimization, just a single thread in the CPU. The time-consuming task of building the EBP-Octree file system is executed just once per model, and the first loading of the data structure takes under three seconds.

The system has been demonstrated to be numerically robust, and data structures on disk are optimized for common haptic usage.

The numerical results encourage us to consider a good collision detection algorithm among large models, not just a single point against the model as we have presented in the paper.

We plan to test the real-time modification of a small area of the model, as this may happen in virtual sculpting, so the EBP-Octree could be used in dynamic models.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Innovation and the EU (via ERDF funds) through the research projects TIN2014-58218-R and TIN2013-47276-C6-3R.

The Wounded Amazon model is courtesy of the Archaeological Museum of Ecija. The Lucy model is courtesy of the Stanford 3D Models Repository. We wish to thank Francisco Soler for his helpful advice.

References

[1] Laycock SD, Day AM. A survey of haptic rendering techniques. Comput Graph Forum 2007;26(1):50–65.
[2] Lin MC, Gottschalk S. Collision detection between geometric models: A survey. In: Proc. of IMA conference on mathematics of surfaces, 1998, pp. 37–56.
[3] Otaduy MA, Lin MC. Clods: dual hierarchies for multiresolution collision detection. In: SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on geometry processing. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association; 2003. p. 94–101.
[4] Larsen E, Gottschalk S, Lin MC, Manocha D. Fast distance queries with rectangular swept sphere volumes. In: IEEE international conference on robotics and automation, 2000. Proceedings. ICRA'00, Vol. 4. IEEE; 2000. p. 3719–26.
[5] Gregory A, Lin MC, Gottschalk S, Taylor R. A framework for fast and accurate collision detection for haptic interaction. In: ACM SIGGRAPH 2005 courses. SIGGRAPH '05, New York, NY, USA: ACM; 2005 http://dx.doi.org/10.1145/1198555.1198604.
URL: http://doi.acm.org/10.1145/1198555.1198604.
[6] Otaduy MA, Lin MC. Stable and responsive six-degree-of-freedom haptic manipulation using implicit integration. In: First joint eurohaptics conference and symposium on haptic interfaces for virtual environment and teleoperator systems, WHC 2005, Pisa, Italy, March 18–20, 2005, 2005, pp. 247–56. http://dx.doi.org/10.1109/WHC.2005.120.
[7] Breen DE, Mauch S. 3D metamorphosis between different types of geometric models. Comput Graph Forum 2001;20(3):36–48.
[8] Hoff KE, Culver T, Keyser J, Lin M, Manocha D. Fast computation of generalized voronoi diagrams using graphics hardware. In: Proceedings of SIGGRAPH 99, computer graphics proceedings, annual conference series, 1999, pp. 277–86.
[9] Sethian JA. A fast marching level set method for monotonically advancing fronts. In: Proc. nat. acad. sci, 1996, pp. 1591–95.
[10] Fuhrmann A, Sobottka G, Gross C. Distance fields for rapid collision detection in physically based modeling. In: Proceedings of GraphiCon 03, 2003, pp. 58–65.
[11] Frisken SF, Perry RN, Rockwood AP, Jones TR. Adaptively sampled distance fields: a general representation of shape for computer graphics. In: SIGGRAPH '00: Proceedings of the 27th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.; 2000. p. 249–54. http://doi.acm.org/10.1145/344779.344899.
[12] Theoktisto V, Fairén M, Navazo I, Monclús E. Rendering detailed haptic textures. In: Second workshop in virtual reality interactions and physical simulations, VRIPHYS 05, 2005, pp. 16–23.
[13] McNeely WA, Puterbaugh KD, Troy JJ. Six degree-of-freedom haptic rendering using voxel sampling. In: Proc. of ACM SIGGRAPH, 1999, pp. 401–8.
[14] Barbic J, James DL. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. IEEE Trans Haptic 2008;1(1):39–52.
[15] Gueziec A. Meshsweeper: Dynamic point-to-polygonal-mesh distance and applications. IEEE Trans Vis Comput Graphics 2001;47–61.
[16] Lauterbach C, Mo Q, Manocha D. gproximity: Hierarchical gpu-based operations for collision and distance queries. Comput Graph Forum 2010; 29(2):419–28.
[17] Pabst S, Koch A, Straßer W. Fast and scalable CPU/GPU collision detection for rigid and deformable surfaces. Comput Graph Forum 2010;29(5):1605–12. http://dx.doi.org/10.1111/j.1467-8659.2010.01769.x.
[18] Morvan T, Reimers M, Samset E. High performance gpu-based proximity queries using distance fields. Comput Graph Forum 2008;27(8):2040–52. http://dx.doi.org/10.1111/j.1467-8659.2008.01183.x.
[19] Walker SP, Salisbury JK. Large haptic topographic maps: marsview and the proxy graph algorithm. In: Proceedings of the 2003 symposium on interactive 3D graphics, SI3D 2003, Monterey, California, USA, April 28-30, 2003, 2003, 83–92. http://dx.doi.org/10.1145/641480.641499 URL: http://doi.acm.org/10.1145/641480.641499.
[20] Yoon S-E, Salomon B, Lin M, Manocha D. Fast collision detection between massive models using dynamic simplification. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on geometry processing. SGP '04, New York, NY, USA: ACM; 2004. p. 136–46 http://dx.doi.org/10.1145/1057432.1057450.
URL: http://doi.acm.org/10.1145/1057432.1057450.
[21] Melero FJ, Cano P, Torres J. Bounding-planes octree: A new volume-based lod scheme. Comput Graph 2008;32(4):385–92.
[22] Morton G. A computer oriented geodetic data base and a new technique in file sequencing, Tech. rep. IBM Ltd.; 1966.
[23] Kaufman L, Rousseeuw PJ. Finding groups in data—an introduction to cluster analysis. John Wiley & Sons; 1990.
[24] Brunet P, Navazo I. Solid representation and operation using extended octrees. ACM Trans Graph 1990;9(2):170–97 http://doi.acm.org/10.1145/78956.78959.
[25] Jordan C. Course DÁnalyse, École Polytchnique de Paris, 1887.
[26] Ehmann S, Lin M. Accurate and fast proximity queries between polyhedra using convex surface decomposition, Comput Graph Forum, Vol. 20, no. 3.