In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

# Efficient algorithms for octree-based geometric modelling

R.-P. Mundani [1], H.-J. Bungartz [1], E. Rank [2], R. Romberg [2], A. Niggl [2]

[1] IPVS, Universität Stuttgart, Stuttgart, Germany
[2] Technische Universität München, Munich, Germany

Preprint

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)



## Abstract

The interface of CAD and numerical simulation is still a severe bottleneck on the way to automated computational development processes. Here, an octree-based hierarchical encoding of geometric objects is a promising approach to their flexible uniform representation for various purposes. In this paper, we present algorithms for the efficient coding and handling of geometric octree models. The primary scenario of application are models of buildings at the interface of architectural descriptions and computational structural analysis.

**Keywords:** octrees, volume-oriented geometric modelling, consistency checks, hierarchical data structures, fast algorithms, on-the-fly generation, embedded simulation

# 1 Introduction

Design and simulation steps within development processes in both mechanical and civil engineering rely on a sufficient representation of the underlying geometry. Thus, a geometric model has to be created. Typical representations are surface-oriented and volume-oriented models, both more or less suitable for different tasks.

While surface-oriented models have a widespread use within design processes [1], especially providing all freedom of modelling like freeform surfaces, they become difficult to handle concerning tasks like collision detection, flow simulation, and structural analysis. A new model meeting all requirements of the latter tasks has to be derived, in general a volume-oriented one.

Volume-oriented models can be described in many different ways—e. g. by norm cells or constructive solid geometry [8]. This paper deals with octrees, a hierarchical volume-oriented data structure providing easy access to solve the tasks named above with respect to their spatial decomposition of the underlying geometry. Here, the main

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

focus is put on fast and efficient algorithms to generate and process octress – even on-the-fly – for models from the application field of civil engineering.

In spite of the principal advantages of octrees concerning complexity, conventional algorithms often don't exploit the full potential of these structures, as octrees of a higher resolution typically entail too high run-time and memory requirements. Based on expensive floating point calculations whether or not a successively refinement in each voxel (cell) is necessary, our approach avoids these calculation costs by a simple parameter comparison of plane equations. For each face of the surface-oriented model a corresponding plane equation is determined, each dividing the entire space into two half-spaces labeled *in* and *out*. An intersection of all half-spaces with the attribute *in* provides the respective volume-oriented model.

# 2 Octrees

One of the first applications quadtrees, the two-dimensional counterparts of octrees, were used for is probably image encoding in computer graphics [9]. By a spatial decomposition of the picture, areas with the same colour could be stored as one single leaf of the tree. Applying this to geometries, a cube (root node) containing the entire geometry is successively halved in every direction until the resulting cells are lying completely inside or outside the geometry. Thus, the order of magnitude of the number of cells necessary to store the geometry under retention of resolution $h = 1/n$ can be reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ in comparison with the norm cell algorithm. Without starting from the scratch, a chosen resolution can be changed easily at any time, since refinements only occur at the surface of the geometry.

This all makes octrees very popular whenever a volume-oriented geometry representation is needed, in particular concerning complexity, flexibility, and usage. Nowadays' applications based on octrees comprise – among other things – the processing of large data sets [14], mesh generation [10], numerical simulation [12, 3], visualisation [2, 3], collision detection [6], geographic information systems [11], and databases or data mining [5], resp.

# 3 Algorithms for octree-based modelling

## 3.1 Preliminaries

As discussed in Section 2, starting from a root node every new cell is successively halved in every direction as long as it has an intersection with the object under consideration. In practice, after reaching a given depth $d_{max}$, the algorithm stops and cells still having an intersection with the object are attributed as *in* or *out* due to a certain decision criterion—like centre of gravity or volume ratio for instance. To determine whether or not there exists an intersection, conventional algorithms calculate this explicitly by costs of many expensive floating point (FP) operations.

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

To avoid these costs, our octrees are built from intersections of attributed half-spaces. The following preliminaries apply to all further considerations. Let the geometry to be converted be a surface-oriented model represented by its boundary surface, composed of single flat surface patches bordered by closed polygons $P_n$ with $n \geq 3$ edges. Furthermore, let the geometry be convex and without holes—non-convex geometries have to be decomposed into convex parts first.

## 3.2 Half-spaces

Every face of the surface-oriented geometry can be seen as an infinite plane, dividing the entire space into two disjoint parts labeled *in* and *out*. An intersection of all parts labeled with *in* results in a volume-oriented representation of the original geometry. By writing the plane equation for each face in the way

$$p : a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 = 0 \qquad \text{with} \qquad \sum_{i=1}^{3} |a_i| = 1, \tag{1}$$

an intersection between plane $p$ and the corresponding cell can be determined by the single parameter $a_0$. Hence, a refinement has to be done if $-1 < a_0 < 1$.

Due to the normalisation of $a_1$, $a_2$, and $a_3$ in (1), cells tested for intersection with plane $p$ are limited from $-1$ to $1$ in every direction. The parameter $a_0$ corresponds to the distance of $p$ from the origin – lying in the centre of a cell –, and thus, intersections only occur for values $|a_0| < 1$. For $|a_0| = 1$, the plane $p$ either overlaps with one of the cell's sides or intersects one of the cell's edges or corners—no refinement is necessary.

In case of a refinement, eight new cells are formed that have to be tested, again by comparing the parameter $a_0$. Regarding the subdivision, the resulting new scaling, and the new origin of these cells, only a new value for $a_0$ has to be determined, as the face normal – given by $a_1$ to $a_3$ – doesn't change. This corresponds to a scaling and a translation into a new coordinate system, displayed as

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{pmatrix} = \begin{pmatrix} 1/2 \cdot x_1 + \zeta_1 \\ 1/2 \cdot x_2 + \zeta_2 \\ 1/2 \cdot x_3 + \zeta_3 \end{pmatrix}. \tag{2}$$

Inserting (2) into (1) results in a plane $\widetilde{p}$ corresponding to the new coordinate system

$$\widetilde{p} : \widetilde{a_0} + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 = 0, \qquad \sum_{i=1}^{3} |a_i| = 1, \tag{3}$$

with

$$\widetilde{a_0} = 2 \cdot a_0 + 2 \cdot a_1 \cdot \zeta_1 + 2 \cdot a_2 \cdot \zeta_2 + 2 \cdot a_3 \cdot \zeta_3. \tag{4}$$

In our case, we have $\zeta_{1,2,3} \in \{-1/2, 1/2\}$, because the centre of a refined cell has a distance of $-1/2$ or $1/2$ from the centre of the original cell in every direction. Thus, (4) can be written as

$$\widetilde{a_0} = 2 \cdot a_0 \pm a_1 \pm a_2 \pm a_3, \tag{5}$$

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

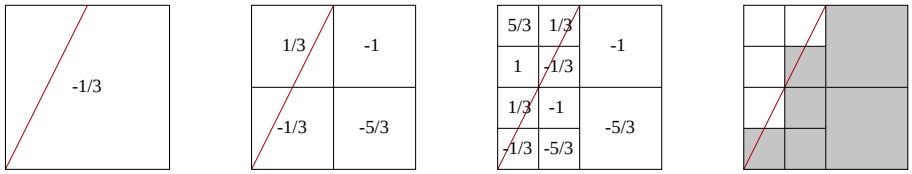


Figure 1: A quadtree refinement corresponding line $l : -\frac{1}{3} - \frac{2}{3} \cdot x_1 + \frac{1}{3} \cdot x_2 = 0$. The cells (pictures 1 - 3 from the left side) contain the value of $a_0$ according to (5) and the attribute (picture 4) according to (6)—white means *out*, grey means *in*.

the sign of $a_1$, $a_2$, and $a_3$ depending on the corresponding cell. Each time a cell is refined, for every new cell parameter $\widetilde{a_0}$ according to (5) has to be calculated for further intersection decisions. If the expression $\pm a_1 \pm a_2 \pm a_3$ is calculated in advance and stored in variables $t^1$ to $t^8$, all necessary calculations for $\widetilde{a_0} = 2 \cdot a_0 + t^i$ can be reduced to one FP multiplication and one FP addition in every step for each new cell.

## 3.3 Cell attributes

Without any sophisticated decision criterion, the attribute for any cell – lying inside or outside as well as still having an intersection with the object after reaching a maximum depth $d_{max}$ – can be found by a simple sign comparison. Choosing equation (1) for plane $p$ in the way that the face normal always points to the outside of the geometry, the attribute for any cell can be determined as

$$a_0 > 0 \Rightarrow outside, \qquad a_0 \leq 0 \Rightarrow inside. \tag{6}$$

No further FP operations are needed, as $a_0$ is calculated anyway for the refinement decision. Thus, the overall costs for one cell to decide whether or not to refine and which attribute to give consists of one FP multiplication, one FP addition, and two FP comparisons. Figure 1 shows a small example in 2D for successive refinement and calculation of $a_0$.

## 3.4 Boolean operators

To form the resulting volume-oriented model out of octree-encoded half-spaces from Section 3.2 and 3.3, further tree operators are necessary that allow to calculate the intersection, union, and difference—Boolean operators for octrees. The latter ones – union and difference – are necessary for dealing with non-convex objects that have to be assembled from convex components.

All three operators have in common that by abidance of a special processing order the resulting work can be minimized. When intersecting two octrees for instance, no further refinement for a cell with attribute *in* of one octree has to be done if the corresponding cell in the second octree has already the attribute *out* (see Figure 2).

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

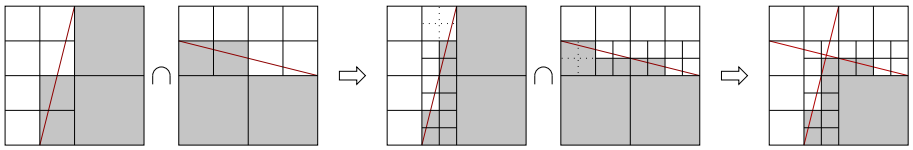

Figure 2: An intersection of two quadtree refinements processed in one single step, where any redundant refinements (dotted lines) not occurring in the final quadtree (right picture) can be avoided.

Thus, redundant refinements that don't occur in the final octree can be avoided, directly forming the resulting tree by processing the octree generation and applying one or more of the given operators in one single step.

The processing order for a union is contrary to the one of an intersection. Here, a cell of one octree having an intersection with the object needs no further refinement when the corresponding cell of the second octree already lies inside the object and, thus, is labeled *in*. With the intersection and union operators any geometry of the type described in Section 3.1 can be built out of octree-encoded half-spaces. To avoid redundant refinements all intersection and union operations have to be done in one pass, which results in the necessity of a unique representation regarding these operators.

As the basic concept of our octree-based modelling, octrees are always formed as unions of convex parts which themselves are generated by intersections of half-spaces. This corresponds to a disjunctive normal form (DNF) that can be written as

$$\left(h_1^1 \cap h_2^1 \cap \cdots \cap h_{n_1}^1\right) \cup \left(h_1^2 \cap h_2^2 \cap \cdots \cap h_{n_2}^2\right) \cup \cdots \cup \left(h_1^m \cap h_2^m \cap \cdots \cap h_{n_m}^m\right), \quad (7)$$

with $h_i^j$ denoting half-space $i$ of part $j$. The DNF satisfies – beside the uniqueness of representation – all necessary requirements to process all operations and generation steps in one single pass, such that complexity is free of redundancy and directly proportional to the surface of the resulting volume-oriented model.

The difference operator has not to be considered separately, because $A \setminus B$ can also be written as $A \cap \overline{B}$. Thus, intersecting $A$ with the inverse of $B$ leads to the same result. Inverting a half-space corresponds to just switching the attribute from *in* to *out* or from *out* to *in*, resp. The difference operator allows to simplify the handling of non-convex objects, like a wall with a window, for instance, can be built as union of four convex parts (the parts around the hole) or as difference from the wall and the window itself.

# 4 Streams

## 4.1 Linearisation of trees

For a persistent storage of trees as well as for further applications, the octree has to be linearised. For that, the nodes and leaves of the tree are processed due to some

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

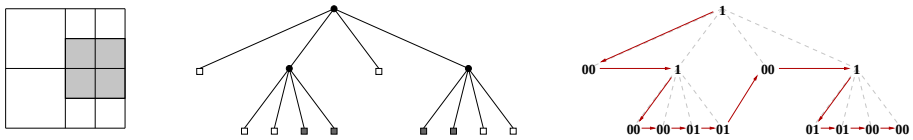


Figure 3: The quadtree in the middle of the picture corresponds to the geometry on the left-hand side. By encoding all nodes and leaves – the latter ones together with their corresponding attribute – a linearisation of the tree results into the stream 10010000010100101010000 (right-hand side).

prescribed oder (like depth-first). Our encoding stores a ”1” for a node and a ”0” for a leaf.

Of course, the geometry itself has to be encoded, too. For that, we store a ”1” for a cell with attribute *in* and a ”0” for a cell with attribute *out*. Hence, whenever a leaf is processed, two values have to be written down, the ”0” for the leaf itself and the value for the corresponding attribute. The order for processing all nodes is given by a depth-first search, writting down all values in prefix notation (see Figure 3).

A depth-first search starts from the root node, always descending in the left-most branch of the tree until a leaf is reached, going backwards to examine all visited nodes in the same manner successively. The prefix notation then writes down the encoded value of a leaf or node when it is visited for the first time. Thus, the linearisation results in a stream only consisting of zeros and ones as representation of the original octree.

To reduce the size of the stream it can be binary encoded, i. e. to spend only one bit for each node or leaf and for the cell’s attributes, resp., by combining 32 bits – or 64 bits on 64-bit architecture – to one 32-bit unsigned integer value. If the last part of the stream is smaller than 32 bits, it needs some padding – filling with zeros until 32 bits are reached – which on the other hand makes it necessary to look for the end of data within the stream whenever it is processed.

Two or more of such binary encoded streams can again be used within the context of Boolean operators, called multiplexers in this content. Multiplexers can deal with both streams generated and stored in advance and with streams generated on-the-fly—streams that are generated and linearised in real-time. For instance, collision detection is one application of multiplexed binary streams.

## 4.2 Handling of streams

Streams of binary encoded octrees can be described in a formal way, using a *Chomsky-II*-Grammar. With the grammar $G = (N, T, P, S)$, consisting of a set of non-terminal symbols $N = \{S\}$, a set of terminal symbols $T = \{0, 1\}$, a start symbol $S$, and a set of productions $P \rightarrow 0|1SSSSSSSS$, any possible octree – here without the additional bits for the attributes – can be created.

For every context-free *Chomsky-II*-Grammar, there exists a finite state automaton $K$ that accepts the language $L_G$ defined by $G$. By starting with an empty stack and a word $y \in L_G$, $y$ is accepted as input when the automaton $K$ stops in a corresponding final state with an empty stack. This formalisation leads to a simple and fast implementation for processing streams by using stacks.

According to the maximum depth $d_{max}$ of the octree a stack $s$ of length $d_{max} + 1$ is needed, where every element $s[i]$ is initialised with $-1$, and where the stack pointer points to the first element $s[0]$. Every time a "1" is read, the value $s[i]$ is increased by one before the stack pointer is set to $s[i+1]$. When reading a "0", the value $s[i]$ is first increased by one before the stack pointer is set to $s[i-1]$ as long as the value of $s[i]$ equals 7. Setting the stack pointer to $s[0]$ again indicates the end of data within the stream—the rest is padding information.

One obvious drawback in the usage of stacks lies in the fact of not being able to set back within a stream. Therefore, the stream has to be processed again from the beginning. But for our applications – especially for collision detection – setting back is not necessary. On the other hand, stacks have a very low memory usage and the stack itself contains the position number of the current processed node or leaf at any time during the entire process. For position numbers of nodes or leaves within the context of octrees see [4].

# 5 Applications and results

Our octree implementation shall be used for the efficient generation and handling of volume-oriented models in the field of civil engineering as the basis for various kinds of applications. Collison detection, global consistency in case of cooperative work, and structural analysis – to name just a few – are important steps on our way to the long-term objective of completely embedded simulation processes.

## 5.1 Collision detection

Before the structural analysis of a CAD-model can be effected, the consistency of the model has to be ensured. As gaps or intersections between parts of the model – due to mistakes during the modelling phase and round-off errors when storing the model to a file – are quite frequent and aggravate the calculation of a correct flow of force, these critical spots have to be located in advance in order to adjust the model.

A collision detection between all parts of the model reveals these spots. Therefore, every part has to be cross-checked with every other part, giving the impression of a complexity of $\mathcal{O}(n^2)$. In practice, most tests already end after some few steps, as the corresponding parts are disjoint and don't share an edge or face and, thus, can be neglected, which leads to a complexity of $\mathcal{O}(n)$ only.

To detect a collision, two octrees or two octree-encoded binary streams have to be intersected—either generated in advance or on-the-fly. When the intersection doesn't

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

Figure 4: Screenshot of our tool for collision detection. Here, an intersection between the two walls has been found, displayed enlarged in the main view on the right side.

become empty after reaching a certain resolution $h$ – given by a maximum depth $d_{max} = \log_2(1/h)$ – there exists an overlap between the two parts. Conversely, when the intersection becomes empty before reaching a depth $d_{min}$, no conflicts exist. For any other depth between $d_{min}$ and $d_{max}$ a gap has been found – intentionally or unintentionally – and some user feedback is necessary. By altering the two parameters $d_{min}$ and $d_{max}$ the precision of the collision detection can be controlled.

## 5.2 Consistency for cooperative work

Another application for our octree-encoded model comprises cooperative work, where global consistency has to be achieved while various clients can alter local parts of the model during the various steps within the design process. They all communicate over a network with the corresponding server that holds the model data and checks for modifications.

Each client can check out parts of the model – read only, read/write, or locking – that can be processed in the respective manner. A read-only check out causes no further problems. The same holds for locking, since parts of the model that should be modifidied and all neighbouring parts are locked and not accessible for other users. Thus, inconsistencies can only occur in the read/write mode. When trying to check in modified parts, again, the server has to determine and notify any conflicts that have arisen. This can be done by a collision detection as described in Section 5.1.

The basic principle for a persistent and reliable storage of the model as well as all mechanisms for checking in and out rely on a two-stage approach (see Figure 5). The geometric information of the model – all plane equations for generating the octree-based model as well as the original surface-oriented data which is needed for visualisation – is stored in a relational database management system (RDBMS). This data

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

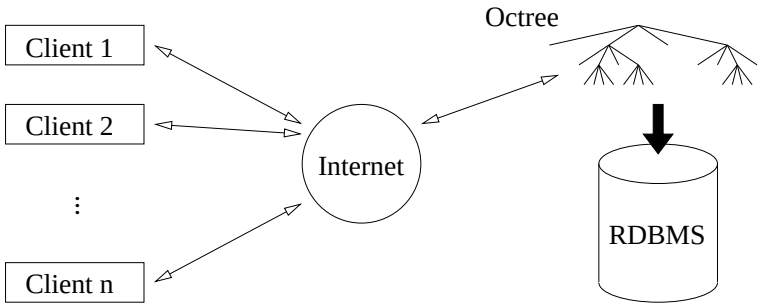


Figure 5: A two-stage approach via an octree and a relational database management system (RDBMS) cares for a consistent model in case of multiple editors (clients), everyone processing parts of the model and being able to make modifications.

can only be accessed via the octree interface, holding a spatial decompositon of the model which is refined as long as one part fits into one cell entirely, storing the associated key for the corresponding database entry. A direct communication between the clients and the database is not possible.

Whenever clients want to check in parts of the model, the octree cares about any conflicts before the modified data is strored into the RDBMS. If a safe storage is not possible because the parts – modified by oneself or by someone else – conflict with other parts, the client is notified and asked to solve the conflict. He's not allowed to check in before.

## 5.3 Structural analysis

One main application of our octree structure is to perform structural simulations based on strictly three-dimensional models. This stands in constrast to the common approach, where the structure is divided into different, dimensionally reduced parts like beams and plates, which are normally computed independently from each other. In comparison to that, the analysis of the structure in one global model ensures consistency during the complete computation process, even in case of local changes.

Starting point of our approach is a standardised building product model like the *Industry Foundation Classes (IFC)* provided by the *International Alliance for Interoperability (IAI)* [7]. This standard data exchange format offers an object oriented description of building product model data and ensures software interoperability in the building industry. In a first step, the geometric information contained in this product model is transfomed to a solid B-rep model, where possible inconsistencies are resolved with the collision detection algorithm shown in Section 5.1. Furthermore, all informations relevant for the numerical simulation can be extracted and saved in the B-rep data structure.

The geometrical 3D model is then decomposed into a so-called *connection model*, which divides the structure into bodies and connection objects, where the bodies co-

In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing,
Civil-Comp Press, 2003 (DOI: 10.4203/ccp.77.10)

Figure 6: Structural analysis based on a standardised building product model—shown is an exploded drawing of the model (left picture), an enlargement of parts of the structure (middle picture), and the results of a structural analysis (right picture).

Figure 7: An octree-based geometry of a simple CAD-model, here displayed with a depth of 8.

incide only in common nodes and edges [13]. This connection model makes it then possible to create a three dimensional hexahedral finite element mesh for the FEM analysis. The connection model, the hexahedral mesh, and the results of a finite element computation are shown for a part of a building in Figure 6.

## 5.4 An example

On the one hand, we have developed a fast and efficient tool for the octree generation of CAD models, and on the other hand, we have a widespread scenario of applications based on volume-oriented models to testify the suitability of octrees as basic data structure in a cooperative working environment. As the latter ones are subject of current research, we want to address runtime and memory usage for the octree generation, here.

The simple model shown in Figure 7 – consisting of 60 half-spaces – has been generated on a standard PC (Pentium4 2.40 GHz). Assuming that the model has a width and length of 10m, a depth of 14, for instance, results to a resolution of 0.61mm for a cell on the finest level. Table 1 shows the runtime, the amount of nodes of the

| Depth | Runtime [s] | Nodes | File size [Byte] |
|---|---|---|---|
| 8 | 0.130 | 342,929 | 80,379 |
| 9 | 0.510 | 1,395,985 | 327,187 |
| 10 | 2.010 | 5,601,137 | 1,312,772 |
| 11 | 8.060 | 22,439,785 | 5,259,332 |
| 12 | 32.400 | 79,717,870 | 21,052,260 |
| 13 | 132.950 | 359,162,713 | 84,178,768 |
| 14 | 508.810 | 1,436,440,585 | 336,665,768 |

Table 1: Runtimes, amount of nodes of the tree, and file sizes at different resolutions for generating an octree of the CAD-model shown in Figure 7.

tree, and the file size for the model at different resolutions. The values in Table 1 always refer to the octree-generation of an entire model which is not necessary in the context of collision detection and global consistency. There, only a small part of the octree has to be calculated, which once more clearly decreases the required time.

# 6 Conclusion

In this paper, we discussed fast and efficient algorithms for generating octrees as intersections of half-spaces—even on-the-fly. We presented three different kind of applications based on our octree implementation, namely collision detection, global consistency for cooperative work, and structural analysis. Due to these promising results future work will concentrate on further applications based on octree-encoded geometries, bringing us closer to our target of completely embedded simulation processes.

# References

[1] R. Barnhill and W. Boehm, *"Surfaces in Computer Aided Geometric Design"*, North-Holland, 1984.

[2] I. Boada, I. Navazo, and R. Scopigno, *"A 3D Texture-Based Octree Volume Visualization Algorithm"*, in WSCG 2000 (8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media), Plzen, Czech Rep., 2000.

[3] P. Breitling, H.-J. Bungartz, and A. Frank, *"Hierarchical Concepts for Improved Interfaces between Modelling, Simulation, and Visualization"*, in Proceedings Vision, Modelling, and Visualization '99, infix, St. Augustin, pp. 269-276, 1999.

[4] A. Frank, *"Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung"*, PhD thesis, Department of Computer Science, TU München, 2000.

[5] L. A. Freitag and R. M. Loy, *"Adaptive, Multiresolution Visualization of Large Data Sets Using a Distributed Memory Octree"*, in Proceedings of SC99: High Performance Networking and Computing, ACM Press and IEEE Computer Society Press, 1999.

[6] P. M. Hubbard, *"Collision Detection for Interactive Graphics Applications"*, PhD thesis, Department of Computer Science, Brown University, Providence, RI, 1995.

[7] IAI 2003, *"International Alliance for Interoperability, IFC Release 2.x"*, http://iaiweb.lbl.gov.

[8] M. Mantyla, *"Introduction to Solid Modeling"*, W. H. Freeman & Co., 1988.

[9] H. Samet, *"Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS"*, Addison-Wesley Pub. Co., 1989.

[10] R. Schneiders, R. Schindler, and F. Weiler, *"Octree-based Generation of Hexahedral Element Meshes"*, in Proceedings 5th International Meshing Roundtable, 1996.

[11] R. J. Szczerba and D. Z. Chen, *"Using Framed-Subspaces to Solve the 2–D and 3–D Weighted Region Problem"*, in Computer Science and Engineering Technical Report 96-18, University of Notre Dame, 1996.

[12] K.-F. Tchon, C. Hirsch, and R. Schneiders, *"Octree-based Hexahedral Mesh Generation for Viscous Flow Simulations"*, in Proceedings 13th AIAA Computational Fluid Dynamics Conference, Snowmass, CO, 1997.

[13] C. v. Treeck, R. Romberg, and E. Rank, *"Simulation based on the product model standard IFC"*, submitted to: Building Simulation 2003, Eindhoven, Netherlands.

[14] D. Vibert, A. Llebaria, T. Netter, L. Balard, and P. Lamy, *"Synthetic Images of the Solar Corona from Octree Representations of 3–D Electron Distributions"*, in ASP Conference Series, Vol. 127, G. Hunt and H. Payne, eds., Astronomical Society of the Pacific, 1997.