View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Digitale Bibliothek Thüringen

Octrees for Cooperative Work in a Network-Based Environment

R.-P. Mundani, H.-J. Bungartz, IPVS, Universität Stuttgart, 70569 Stuttgart, Germany
({mundanrf,bungartz}@informatik.uni-stuttgart.de)

Summary

Assuring global consistency in a cooperative working environment is the main focus of many nowaday research projects in the field of civil engineering and others. In this paper, a new approach based on octrees will be discussed. It will be shown that by the usage of octrees not only the management and control of processes in a network-based working environment can be optimised but also an efficient integration platform for processes from various disciplines – such as architecture and civil engineering – can be provided. By means of an octree-based collision detection resp. consistency assurance a client-server-architecture will be described as well as sophisticated information services for a further support of cooperative work.

1 Introduction

Surface-oriented models, very popular with respect to their huge manipulation possibilities and, thus, core part of all nowaday CAD applications, are only less suited for simulation and controlling tasks, still the domain of their counterparts—volume-oriented models. Due to their spatial decomposition of the underlying geometry, volume-oriented models provide easy access to several simulation tasks such as structural analysis, computational fluid dynamics, or managing and controlling entire design processes. Here, octrees – hierarchical recursive data structures – are used to build up a client-server-architecture for both cooperative work and process integration of different tasks from the field of civil engineering. The main target of this approach focuses on efficient octree-based algorithms to assure global consistency between all participating experts, to support cooperative work in a network-based environment, and to bridge the gap between surface-oriented and volume-oriented applications such as CAD and simulation.

2 Octree generation

Following the main principle of octrees, a cube containing the entire geometry is recursively halved in every direction until the resulting cells – the voxels – are lying completely inside or outside the geometry. In each refinement step one node is assigned eight new voxels, its sons. Thus, a hierarchical tree structure evolves. Compared to an equidistant discretisation, the overall amount of voxels reduces from $O(n^3)$ to $O(n^2)$ for the octree approach. One main drawback of common octree generation algorithms is the large amount of floating point (FP) operations when calculating the respective intersections with each of the octree's voxels for refinement decisions. Within our approach, octrees are generated as intersections of half-spaces. Hence, the amount of necessary FP operations can be reduced to a minimum (see Mundani 2003 for more details). This not only allows us to generate octrees in real time but also on-the-fly.

2.1 Convex decomposition

Before any octree can be generated as an intersection of half-spaces, the respective surface-oriented model has to be decomposed into convex parts. Therefore, the object's convex hull is recursively calculated and a corresponding Boolean expression is formed. Starting with a surface-oriented model, first of all, faces belonging to so-called inner loops – i.e. holes to the main body such as windows – have to be determined to be processed separately. All resulting faces belong to an outer loop—the main body (see Fig. 1 for a small example).

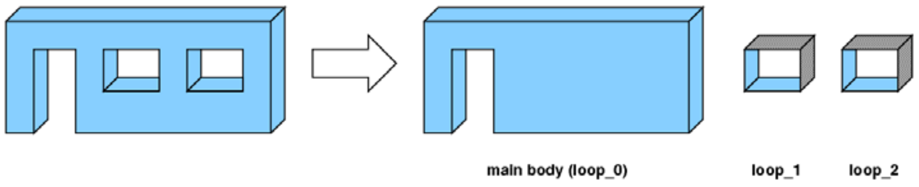


Figure 1: Sample decomposition of a model consisting of a main body and two inner loops.

Thus, the object has been decomposed into single parts according to its loops, forming the outmost frame for the resulting Boolean expression. A union of all inner parts is substracted from the main body (p_0). Hence, the expression can be written as

$$p_0 \setminus (p_1 \cup p_2 \cup \ldots \cup p_{n-1} \cup p_n). \tag{1}$$

Each part p_l from expression (1) is now processed in the same way, subdivided into the steps:

i. The part's convex hull is calculated and all faces lying on it are marked as being processed.

ii. All faces not marked yet are clustered to subsets C_k according to equation (2), where $\bowtie$ inidicates that faces f_i and f_j share one edge.

$$C_k := \{ f_i \mid f_i \bowtie f_j,\ \ f_j \in C_k \} \tag{2}$$

iii. A Boolean expression of the form

$$H^i \setminus (C_1^{\,i} \cup C_2^{\,i} \ldots \cup C_{m-1}^{\,i} \cup C_m^{\,i}) \tag{3}$$

is generated, where H^i and $C_k^{\,i}$ denote the convex hull and subset k from step i.

Recursively repeating steps i—iii for all subsets $C_k^{\,i}$ in (3) until they become empty and always substituting set $C_k^{\,i}$ by its corresponding Boolean expression results in a complete decomposition of the respective part p_l as follows

$$H^0 \setminus ((H^1 \setminus (H^2 \setminus (\ldots))) \cup \ldots \cup (H^{n-j} \setminus (H^{n-j+1} \setminus (\ldots)))). \tag{4}$$

Substituting (4) for each part p_l in (1) provides the final Boolean expression to generate an octree as intersection of half-spaces. Figure 2 illustrates this for the main body (part p_0) of the small example from above. The resulting Boolean expression for the model on the left-hand side from Fig. 1 reads as follows: $(H^0 \setminus H^1) \setminus (H^2 \cup H^3)$, where H^2 and H^3 denote the corresponding expressions for the inner loops shown on the right-hand side in Fig. 1.

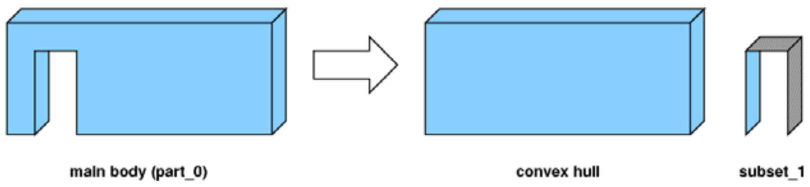


Figure 2: After calculating the convex hull for the part shown on the left-hand side one subset containing three faces is left (steps *i* and *ii*). This subset itself is already convex, hence, the Boolean expression according to (4) reads as $H^0 \setminus H^1$.

2.2 Collision detection

For the further usage of octrees, the corresponding trees have to be linearised. Applying an appropriate enconding followed by a depth-first search leads to binary streams (see Mundani 2004) that can easily be multiplexed, i.e. combining two or more streams with Boolean operators (intersection, union, or difference). An intersection of two streams, for instance, can be used to test the respective models for any kind of collision, in this context defined as either an intersection or a gap. Collision detection is the primary module in our client-server-architecture to assure global consistency.

The entire collision detection is controlled by two parameters—the maximum depth of recursion d_{max} and the (minimum) depth of gap detection d_{min}. The maximum depth of recursion d_{max} defines the depth of refinement for the octree and, thus, the resolution $h = 1/2^{d_{\max}}$ for a voxel on the finest resolution level. Assuming a geometry with a footage of 10m a depth of 14 is necessary for a resolution of 1mm on the finest resolution level. Hence, intersections up to a width of 1mm can be found. The first appearance of a resulting voxel lying inside both geometries indicates a collision, so the algorithm can stop and an intersection between both parts was found.

For the case the two parts are disjoint, the resulting binary stream will be empty. Here, the maximum depth d_{eff} reached during the calculation – not to mix up with the maximum depth of recursion d_{max} – has to be determined. It's obvious that between two disjoint parts there always exists a gap, but we are only interested in gaps of at least a certain width controlled by parameter d_{min}. Thus, if the equation

$$d_{min} \leq d_{eff} < d_{max} \tag{5}$$

is true, a collision of type gap has been found and further user interaction is necessary. Not all gaps are in fact collisions due to round-off or modelling errors, some might intentionally be made and, thus, only the corresponding experts has enough knowledge to solve this instance. Figure 3 shows some graphical tool for collision detection developed by our group.

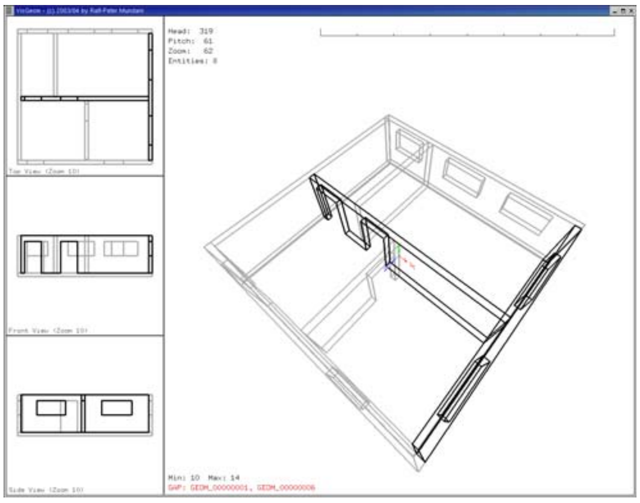


Figure 3: A collision detection (d_{min} = 10, d_{max} = 14) reveals a gap of width 1mm between the two highlighted parts of the model (footage 13m).

3 Cooperative Work

A client-server-architecture with octree-based modules such as assuring global conistency with a collision detection as described above has been developed to support cooperative work in a network-based environment. One global geometric model stored as a *vef*-graph – vertices, edges, and faces – can be accessed and manipulated by several experts at the same time, always assuring global consistency by the system. Furthermore, information services notify all experts about any changes to their shared resources.

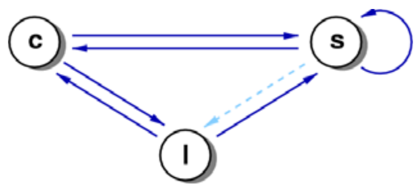
3.1 Global geometric model

When initialised at start-up, an attributed model in Eurostep IFC format is read and internally represented as a *vef*-graph. Attributes can comprise material definitions, element types (e.g. wall, door, ceiling), and boundary conditions for numerical simulations. The corresponding data – vertices, edges, faces, and attributes – are written to a Relational Database Management System (RDBMS) for persistant storage. A so-called controll tree, an octree refined as long as single parts of the model entirely fit into a single voxel, stores the primary keys to the RDBMS's tables, such that neighbouring parts can efficiently be found. For the primary keys the parts' UUIDs are used—unique identifiers from the IFC model. Direct access to the RDBMS from the client-side is prohibited, an additional layer around the RDBMS providing functions for data access manages and controls any communication to the global geometric model. For further considerations, the RDBMS with its additional layer is called server.

As in classical Concurrent Versions Systems (CVS) clients can check out data from the server to be processed in their local work spaces before the modified data is written back (check-in) to the global geometric model. At this time, a collision detection for each modified part is initiated, testing it pairwise with all its direct neighbours. In case an intersection is detected, the modified part is rejected and the respective client is informed of it. In case of a gap the modified part is written to the RDBMS, overwriting its older version, simultaneously raising a warning notification to inform the client and asking him to check this incident again whether the revealed gap was intentionally made or occurred due to an error. After a successful check-in all clients sharing the corresponding parts at this time are automatically informed about the new updates—releated to both the geometry and attributes.

3.2 Shared data access

Whenever parts of the global geometric model are checked in or out their respective states change. Within our implementation a part can have either one of the following states: *clean* (c), *shared* (s), or *locked* (l). The state *clean* indicates that a part isn't shared by any client at the moment, *shared* indicates at least one client with read-only or read/write access, and *locked* indicates exclusive write access for exactly one client. Possible state transitions are as follows:



Starting from state *clean*, the first client accessing read-only, read/write or exclusive write permissions changes the state to s*hared* or *locked*, resp. From state *shared* a transition to *locked* – one client trying to access exclusive write permissions – is possible if and only if the client is the only one using this resource. Otherwise the client is rejected with a permission error.

To access data, the server provides several methods that can be invoked from the clients. Beside update(), all methods take at least a list of one or more parts' UUIDs as argument. The different methods are as follows:

- update() – returns all parts not yet shared by the client with read-only permissions,
- check-out() – returns all chosen parts with desired permissions—if possible,
- check-in() – writes back given parts to the server to replace respective obsolete versions,
- cancel() – clears all chosen parts from user's access,
- delete () – deletes all chosen parts—if possible.

If a client wants to delete one or more parts he at least needs read/write permissions and he must be the only one with write access in case other clients share these resources, too. This prevents that one client still holding a (writeable) local copy of some specific part is able to write back his copy to the server in case the global version was deleted by some other client before—an illegal undo of a previous operation, obviously violating the global consistency.

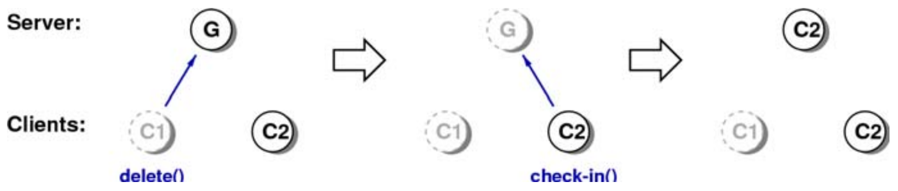


Figure 4: One client deletes his local copy (C1) and the global version (G) of a specific part while a second client still holds another (writeable) copy (C2). Afterwards, the second client writes back his copy (C2) and "undos" the previous operation—an inconsistency occurred.

Due to the server's internal data representation, both a surface-oriented and a volume-oriented model can easily be derived from the *vef*-graph. Tasks like CAD or structural analysis, for instance, retrieve parts of the global geometric model in ACIS SAT format, tasks like computational fluid dynamics (CFD) retrieve parts of the global geometric model as binary encoded octree. Further export formats are possible and, thus, many more processes from other disciplines can be efficiently integrated into the discussed client-server-architecture. Figure 5 illustrates this architecture with some sample processes already integrated by our group.

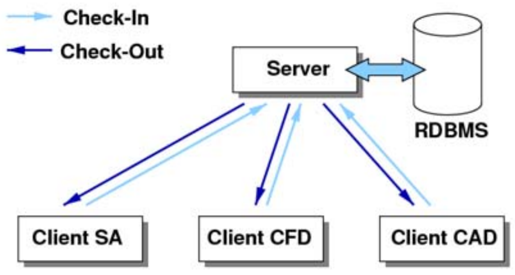


Figure 5: Client-server-architecture with integrated processes – structural analysis (SA), computational fluid dynamics (CFD), and CAD – from disciplines such as simulation and architecture.

3.3 Information services

Whenever an update – related to the geometry and/or attributes – to any parts of the global geomtric model is written back to the RDBMS, all clients sharing these specific elements are automatically notified by an information service. This service is based on agent technology, autonomous program parts as substitutes for the corresponding clients, periodically querying the server for update informations. During start-up time each client creates his own agent, configuring it with the user's individual update notification preferences. On a coarse granularity level preferences comprise turning on/off all geometry, material or simulation related attribute changes, on a fine granularity level users can choose which material or simulation related attributes they are interested in. Thus, a FEM agent, for instance, can be reduced to FEM related simulation attributes only.

Agents then query a server agent for all update notifications related to parts shared by their clients. In case of corresponding updates the client agent stores all relevant informations for further processing and notifies his client. The client now has to initiate a poll on his agent for retrieving the already queried and stored update informations. This prevents – in case update notifications are issued very frequently – permanent interruptions of a steady workflow on behalf of the client. According to the user's preferences all update information is filtered by the agent and unwanted messages are discarded. The client shall be responsible for any check-out of modified parts to update his local copies.

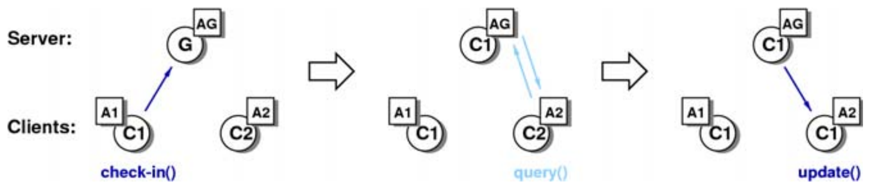


Figure 6: One client writes back his modified copy (C1) to the server and, thus, updates the global version (G) of the specific part. A second client's agent (A2) queries the server – the corresponding server agent (AG) – and is informed about an update of a part his client also holds a local copy of (C2). The second client now can decide to discard his copy and retrieve an actual version from the server.

Further usage of those agents allows clients to retrieve state informations for any chosen parts. After selecting all parts of interest in the local works space, a state query reveals information about a part's status (clean, shared, or locked), date, type and responsible person of last changes as well as all clients with respective permissions sharing this part at the moment. Due to this state informations every client knows at any time which other clients are processing which parts. This might help to prevent conflicts in advance and furthermore supports the organisation of processes in a cooperative working environment.

4 Conclusions

Octrees as integral element in a cooperative working environment, as presented in this paper, facilitate both the organisation of processes with regard to global consistency among all participating experts and the integration of processes from different application scenarios. They efficiently bridge the gap between several tasks from different disciplines – e.g. CAD and simulation – and, due to their inherent hierarchy, provide a huge potential for further applications not exploited so far. Next steps will comprise the integration of an octree-based nested dissection solver for an optimised control of a finite element analysis, bringing us one step closer to the long-term objective of completely embedded simulation processes.

5 References

A. Frank. 2000. Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung. Herbert Utz Verlag. Germany.

R.-P. Mundani, H.-J. Bungartz, E. Rank, R. Romberg, and A. Niggl. 2003. *Efficient Algorithms for Octree-Based Geometric Modelling*. In proceedings of "9th International Conference on Civil and Structural Engineering Computing". Civil-Comp Press. United Kingdom.

R.-P. Mundani and H.-J. Bungartz. 2004. *An Octree-Based Framework for Process Integration in Structural Engineering*. Processed to "8th World Multi-Conference on Systemics, Cybernetics and Informatics".