Internatinal Conference on Adaptive Modeling and Simulation
ADMOS 2005
P. Díez and N.-E. Wiberg (Eds)

ORGANIZING A *P*-VERSION FINITE ELEMENT COMPUTATION BY AN OCTREE-BASED HIERARCHY

A. Niggl*, E. Rank*, R.-P. Mundani† and H.-J. Bungartz†

*Lehrstuhl für Bauinformatik
Technische Universität München
Arcisstraße 21, 80335 München, Germany
e-mail: {niggl, rank}@bv.tum.de,
web page: http://www.inf.bv.tum.de

†Fakultät für Informatik
Technische Universität München
Boltzmannstraße 3, 85748 Garching, Germany
{mundani, bungartz}@in.tum.de,
web page: http://www5.informatik.tu-muenchen.de

Key words: high-order FEM, *p*-version, nested dissection, octrees

Summary. *In this paper, we present a method to organize a high order finite element model in an octree-based hierarchy. Thereby the finite element problem is solved by using a nested dissection algorithm. The main focus of this work is to utilize the hierarchic organization in order to speed up the solution process in case only parts of the computational domain have changed. An example will show, how this approach can be used to accelerate local refinement processes in a p-version finite element computation.*

1 INTRODUCTION

The product development process in civil engineering is characterized by an *iterative* as well as a *top-down* approach. Iterative means, that a first design is continuously modified on different levels of detail until a result is found, which fulfills the needs of all involved participants best. Creating a model in a top-down order is to first define a product on a coarse or conceptual level, e.g. by defining the floor structure of a building, before this model is then refined by iteratively replacing objects with successively finer grained parts.

While modern CAD systems are more and more capable to meet these characteristics of the design process, simulation models in the field of computer aided engineering are normally not able to follow such recurring modifications. Most simulation tasks have to be recomputed each time from the beginning, even if the design has changed in some parts. In structural analysis this means that a new finite element mesh has to be created and the computation has to be performed for the whole domain. This can be a time consuming task, especially in case if some steps are not fully automatic.

In this paper, we present an approach to organize the elements of a p-version finite element computation in a hierarchical way, based on octrees. Thus, we can apply efficient solvers based on the classical nested dissection algorithm. In case only parts of the model have changed, the hierarchic structure can be utilized to speed up the solution time significantly by only re-computing the modified parts in the octree-hierarchy. This will be demonstrated in an example, where the polynomial degree of some elements of a p-version computation is changed locally in order to get more accurate results without recomputing the whole problem.

2 STRUCTURAL SIMULATION WITH HIGH-ORDER FINITE ELEMENTS

The idea of our approach is to perform the finite element simulation of large building structures in a consistent volume-oriented way based on hexahedral elements. This is different to current practice, where the structural system of buildings is modeled with dimensionally reduced objects like plates, shells or beams.

A problem of finite elements based on dimensionally reduced models is, that they demand using transition elements between different types of structures. Typical examples are connections between plates and columns or transitions from massive to thin-walled structures, e.g. shells with stiffeners or complex foundations. In such transition areas it is difficult to decide where reduced stress states can be assumed and where arbitrary three-dimensional stresses must be taken into account. Identifying those critical areas lies in the responsibility of the structural engineer. When using three-dimensional models instead this model error turns into a discretization error of the finite element formulation. It can be automatically measured and adaptively controlled.

In [6] it was shown how to automatically derive a hexahedral finite element mesh from a typical architectural building model. The algorithm presented there is based on a decomposition of the building structure into simpler geometrical objects which in turn are meshed individually.

The finite element computation is performed using higher order p-version finite elements. Contrary to classical lower-order finite elements of the h-version, these elements allow to increase the polynomial degree of the displacement *Ansatz* in order to reduce the approximation error of the solution.

An important characteristic of the p-version is, that the elements are very robust with respect to large aspect ratios (up to 1:1000). This makes it possible to model thin-walled structures in a strictly three-dimensional way by using shell-like hexahedral elements with only one layer in thickness direction. Thus, structures consisting of equally solid and thin-walled elements can be modeled consistently with the same three-dimensional approach and without any need for special transition elements. The efficiency of this approach was clearly demonstrated in [2, 1].

Another very important feature in the context of high-order elements are anisotropic *Ansatz* spaces. Using the approach presented in [3] one can define different polynomial degrees in the different local directions of the hexahedral element. For plate-like structures, the polynomial degree in in-plane direction can be chosen differently from the one in thickness direction in order to further reduce the computational effort. Using error estimation, the process of locally changing the polynomial degree can also be controlled automatically by an adaptive process.

3 HIERARCHICAL ORGANIZATION

The usual approach in finite elements is to organize the data in a 'flat' way with no special hierarchic structure. This makes them quite inflexible for local modifications during the computation or in adaptive processes. Many researchers have addressed this problem by introducing an hierarchy into the finite element computation, for example in [5]. Most of these methods discretize the geometry of the domain directly with octrees and use this hierarchy for applying substructure techniques for solving the equation system. However, the geometric discretization with octrees makes it difficult to model arbitrarily shaped domains efficiently. In our case, octrees are only used for organizing the finite elements in a hierarchic way in order to manage the solution process. The definition of the *Ansatz* functions as well as the computation of the element matrices are performed in a standard way on a regular finite element mesh.

3.1 Creating a Finite Element Hierarchy

After a finite element mesh is created, the elements are sorted into a hierarchic order. Using some unique spatial identifier - in our case the barycenter is used - each element is assigned to a cell in the octree hierarchy. The initial domain is thereby recursively subdivided as long as an octree cell contains more than one element center point. Figure 1 shows a finite element mesh with its hierarchical representation for the two-dimensional case. Later on, all relevant element data - stiffness matrix and load vector - will be stored on these positions in the octree.

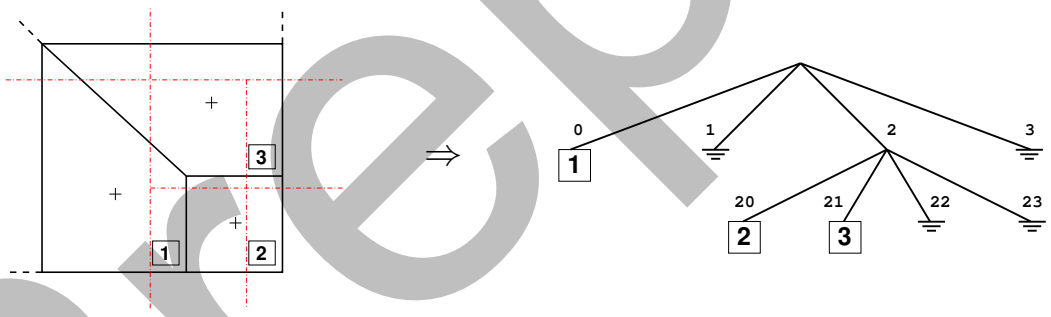

Figure 1: A sample FE discretization in 2D and its corresponding tree for the 2-dimensional case. The spacial partition of the tree is depicted in dotted lines.

3.2 Setup of Degrees of Freedom

In contrast to the h-version, the degrees of freedom (DOF) of a p-version computation do not directly correspond to the nodal displacements of an element. There can be more than 100 degrees of freedom per hexahedral element. Therefore the DOF must be assigned in a separate step to the octree. This assignment takes place during the setup phase in the finite element program. Degrees of freedom on an internal element boundary are usually shared by more than one element. For a correct insertion of these $DOFs$ into the octree, the lowest common father node (LCF) of all involved elements has to be found. Lowest means, the last node visited within a top-down descent from which all corresponding elements can still be reached. The lower one DOF can be inserted into the octree, the better it is for the later computations, because it can be

eliminated earlier during the nested dissection assembly. For finding the LCF of some elements, a unique identifier for the octree's nodes, the so-called Morton index is used. By naming a node's eight sons from '0' to '7' in some specific order, one can obtain the index by accumulating all numbers on a way down from the root to the desired node (see Figure 1). Finding the correct position of a DOF in the tree is achieved by comparing the Morton indices of the elements it belongs to. The indices are read number by number from the left-hand side as long as they match. The resulting Morton index then indicates the LCF where the corresponding DOF has to be stored. In the worst case, the result is empty, thus, the DOF will be assigned to the root node. Figure 2 shows the assignment of degrees of freedom (denoted by small circles) to the tree of the previous example. In this example, the DOF with number 7 is shared by the two elements 2 and 3 having a Morton index of 20 resp. 21. Comparing the two Morton indices results in a LCF with index 2. This number indicates the position where DOF 7 will finally be stored in the tree.

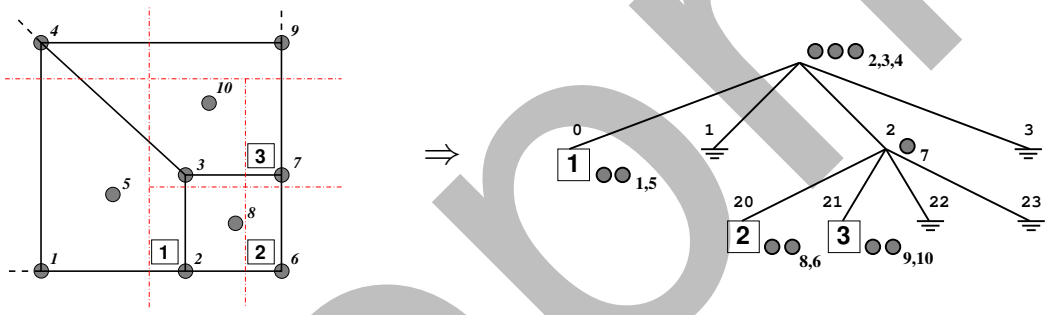

Figure 2: Assignment of degrees of freedom into the tree.

Assigning all $DOFs$ to the octree finishes the setup phase of the finite element computation. Before the solution process starts, all element stiffness matrices and load vectors are computed and stored at the location of the particular element in the tree. The mesh is no longer necessary. All further tasks are processed directly on the tree utilizing the hierarchy in the nested dissection algorithm, which is discussed in the next section.

3.3 Nested Dissection

Applying a nested dissection algorithm for the solution of finite element problems was done very early by J.A. George [4]. The main idea behind this technique is to decompose the system of linear equations into smaller parts and to eliminate in a bottom up step local unknowns before in a final top-down step the solution can be computed. For any equation system

$$K\,u = f, \tag{1}$$

the nested dissection process starts by reordering this system according to inner (i) and outer (o) degrees of freedom, leading to:

$$\begin{pmatrix} K_{oo} & K_{oi} \\ K_{io} & K_{ii} \end{pmatrix} \begin{pmatrix} u_o \\ u_i \end{pmatrix} = \begin{pmatrix} f_o \\ f_i \end{pmatrix}. \tag{2}$$

Separating this system into two parts

$$K_{oo}u_o + K_{oi}u_i = f_o \tag{3}$$

$$K_{io}u_o + K_{ii}u_i = f_i, \tag{4}$$

and rewriting (4)

$$u_i = K_{ii}^{-1}\left(f_i - K_{io}u_o\right), \tag{5}$$

allows to eliminate all inner unknowns u_i in equation (3). As a result, we get a new equation system

$$\left(K_{oo} - K_{oi}K_{ii}^{-1}K_{io}\right)u_o = f_o - K_{oi}K_{ii}^{-1}f_i, \tag{6}$$

which depends only on outer unknowns u_o. The expensive part of the solution process is the computation of the expression $\tilde{K}_{oo} := K_{oo} - K_{oi}K_{ii}^{-1}K_{io}$, the so-called SCHUR complement. There exist several methods to perform this computation efficiently. In our case, a GAUSSIAN elimination is used.

Applying the nested dissection algorithm to an octree hierarchy, the elimination of the interior degrees of freedom is performed recursively, starting on the leaf nodes of the tree. On each node in the octree, the SCHUR complement is computed and then passed to the father node, where it is assembled with all other SCHUR complements of the sons to a new partial equation system. The decision of which degrees of freedom are internal and which are external depends on their position in the octree, where they have been assigned to in the setup phase as explained in section 3.2. Considering the 2D-problem of Figure 2, *DOF* 8 and 6 for example are only related to element 2, while *DOF* 2 and 7 are shared by several elements. Thus, *DOF* 8 and 6 can be eliminated, before the stiffness matrix is passed to the next level.

3.4 Performing Local Modifications

One huge advantage of this hierarchical approach lies in the reduction of computations whenever the underlying model changes. After local modifications, not the whole problem has to be computed completely but only parts of the tree have to be re-assembled, which reduces the computational effort considerably.

Assume, for example, the stiffness matrix of one element changes. Only those SCHUR complements have to be recomputed which lie on the way down starting from the root node to the node representing this element. As the number of tree-leafs storing the elements grows by a power of 8 with the tree-depth, the amount to compute only one branch of the tree causes much less effort than a complete re-computation.

Combining this approach with the *p*-version of the finite element method opens the possibility for very efficient adaptive processes. As described before, the *p*-version allows to simply change the discretization locally only by increasing the polynomial degree. Using the hierarchical approach, a re-computation of the whole domain is no longer necessary, the results can be obtained much faster.

4 EXAMPLE

As an example, a plate on columns is considered. Figure 3 shows the finite element mesh consisting of 194 hexahedral elements. Our scenario starts by computing the problem with a

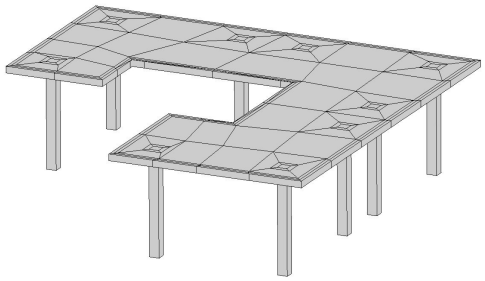

Figure 3: Hexahedral finite element mesh of a plate on columns.

constant polynomial degree of $p = 4$. After that, the polynomial degree in some elements at the conjunction of the plate with a column is increased to $p = 5$ and a re-computation is started. The table shown below lists the computation times for the initial run in comparison to the second run. The times are sorted according to the setup phase, computation of the stiffness matrices and the solution phase. The setup step covers work for setting up the tree and assigning degrees of freedom. It mainly has to be done in the initial run. Other times, e.g. for file IO or computing postprocessing results are not considered. All calculations were done on an Intel Pentium with 3.4 GHz under Linux.

	DOF	Setup	Stiffness	Solution	$\sum t$
initial run	13569	3.67	4.88	10.90s	19.45
re-computation	13641	0.00	0.36	4.32s	4.68

It can be observed that the decrease of computation time is partly caused by a reduced effort in computing only the modified stiffness matrices. Instead of all 194, only 6 matrices have to be computed in the second run. This effect grows by increasing the polynomial degree of the *Ansatz*-functions. Especially for high polynomial degrees ($p \geq 8$) the calculation of the stiffness matrices dominates the overall computational cost. The second part of the process, the solution of the equation system with the nested dissection algorithm, decreases from 10.90 seconds for the whole problem to 4.32 seconds for the partial assembly. For larger problems with higher tree-depths, we expect that this speed-up will be enforced. But, detailed investigation on this topic have to be performed in future.

5 CONCLUSIONS

We have presented an octree-based approach to set up a hierarchy for the p-version of the finite element method. It was shown, that this approach reduces the necessary computations in case of local modifications. The combination with the p-version of the finite element method offers the possibility to perform adaptive processes very efficiently. Studies of model alternatives and local refinements become more attractive due to the reduced computing times. Especially in the context of an overall product development process this method helps that also the structural simulation is more and more able to meet the iterative character of these engineering tasks.

6 ACKNOWLEDGEMENT

This research has been supported by the Deutsche Forschungsgemeinschaft (Priority program 1103 "Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau") to which the authors are grateful.

REFERENCES

[1] A. Düster. *High order finite elements for three-dimensional, thin-walled nonlinear continua*. PhD thesis, Lehrstuhl für Bauinformatik, Technische Universität München, 2001.

[2] A. Düster, H. Bröker, and E. Rank. The p-version of the finite element method for three-dimensional curved thin walled structures. *International Journal for Numerical Methods in Engineering*, 52:673–703, 2001.

[3] A. Düster, D. Scholz, and E. Rank. *pq*-Adaptive solid finite elements for three-dimensional plates and shells. *submitted to Computer Methods in Applied Mechanics and Engineering*, 2005.

[4] J.A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[5] R. Hüttl. *Ein iteratives Lösungsverfahren bei der Finite-Element-Methode unter Verwendung von rekursiver Substrukturierung und hierarchischen Basen*. PhD thesis, Fakultät für Informatik, Technische Universität München, 1995.

[6] R. Romberg. *Gebäudemodellbasierte Strukturanalyse im Bauwesen*. PhD thesis, Lehrstuhl für Bauinformatik, Technische Universität München, 2005.