Extending the *p*-Version of Finite Elements by an Octree-Based Hierarchy

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

[1] Fakultät für Informatik, Technische Universität München, Germany
[2] Lehrstuhl für Bauinformatik, Technische Universität München, Germany

Abstract. In structural mechanics, a large variety of finite element approaches are used, some of them – especially of *p*-type – without an inherent hierarcical substructuring. This often turns out ot be a drawback. By embedding the finite element decomposition into an octree structure, the elements can be arranged in a hierarchical way, which does not only open the door to efficient iterative solvers based on the classical nested dissection algorithm, but also allows to speed up the solution process in case only parts of the underlying geometric model are changed, as only those parts and their region of direct influence have to be recomputed. In this paper we present an efficient method to map an octree-based hierarchy onto an arbitrary finite element mesh, to use this octree structure for implementing a fast iterative solver of nested dissection type, and to set up a framework for completely embedded simulation processes as they appear in many civil engineering applications, for example.

1 Motivation

The cooperation of different simulation tasks often suffers from proprietary data representations – surface-oriented or volume-oriented, for instance – and insufficient interfaces between single processes. In [1] we presented a framework for process integration for applications from the field of structural engineering, where global consistency among all participants as well as a common data model for all kind of simulation tasks is achieved by octree-based methods.
In this paper, we present an octree-based approach to arrange the elements resulting from the *p*-version of a finite element (FE) discretisation in a hierarchical way. Thus, we can apply efficient iterative solvers based on the classical nested dissection algorithm. Furthermore, this hierarchical substructuring can also be exploited for a faster computation of the solution in case some geometric modification occurs. As only those parts of the octree influenced directly by a geometric alteration have to be recomputed, the time for obtaining the solution can be significantly reduced. Hence, even computations in real time are possible. By embedding this hierarchical substructuring approach into the framework mentioned above, different simulation tasks can be handled in a more efficient way. Thus, for any kind of problem the framework can provide – like a construction kit – a specific and unique solution, a so called problem solving environment.

2 Octrees

Many nowadays' simulation tasks are based on hierarchical data structures or octrees, in particular, as they have turned out to be advantageous for a huge amount of different tasks. Octrees, that is recursively halving a cube containing the entire geometry in each direction, as long as the resulting cells – aka voxels – are lying completely inside or outside the geometry. Thus, the overall amount of necessary cells is reduced from $\mathcal{O}(n^3)$ for an equidistant discretisation to $\mathcal{O}(n^2)$. By a new technique first presented in [2], we are able to create these octrees in real time and even *on-the-fly* also for larger (greater than 12) levels of recursion. Especially in the field of numerical simulation, octrees provide a big potential for mostly all kind of problems due to their fast and easy access of the underlying geometry.
To address each cell by some uniqe identifier, the so called Morton index is used. By naming a node's eight sons from '0' to '7' in some specific order[3], one can obtain the Morton index of a cell by accumulating all node's numbers on its way down from the root to the desired cell. One main advantage of these identifiers is the possibility of easily determining neighbouring cells, an important aspect when degrees of freedom resulting from an FE discretisation have to be assigned to their corresponding nodes in an octree.

3 *p*-Version of Finite Elements

The p-version of the finite element method has turned out to be an efficient discretisation strategy for solving finite element problems arising in structural engineering. In contrast to the classical h-version approach, the p-version leaves the mesh unchanged and increases the polynomial degree of the shape functions in order to reduce the error of approximation. Our p-version implementation uses hierarchical shape functions for the displacement Ansatz, following Szabó and Babuška [3]. Contrary to the classical approach for higher order modes, the hierarchical bases are constructed such that all lower order shape functions are completely contained in the higher order bases. Thus, the finite element basis can be easily extended up to any desired polynomial degree without changing the complete set of shape functions for each different polynomial degree.
In the work shown here, the finite element computation is based on a fully three-dimensional approach using hexahedral elements. The shape functions of the three-dimensional hexahedral Ansatz spaces are constructed by forming the tensor product of the one-dimensional bases. One important property is that the hexahedral p-version elements are very robust w. r. t. elements' distortions – aspect ratios up to a factor of 1000 are possible. This makes it possible to use them equally for solid "thick" structures as well as for thin walled, shell-like structures [4]. The computational effort can further be decreased by using different polynomial degrees in different directions. For shell like structures, for example, this *anisotropic* Ansatz space allows to reduce the polynomial degree

[3] The order itself is not relevant, but it has to be consistent among all nodes

in thickness direction while leaving the polynomial degree in in-plane direction unchanged [5].
With this approach, large structures can be computed using the same element formulation consistently for the whole domain. Figure 1 shows the computation results of the structural model of an office tower under vertical load on all plates. The example was computed with a moderately high polynomial degree of $p =$

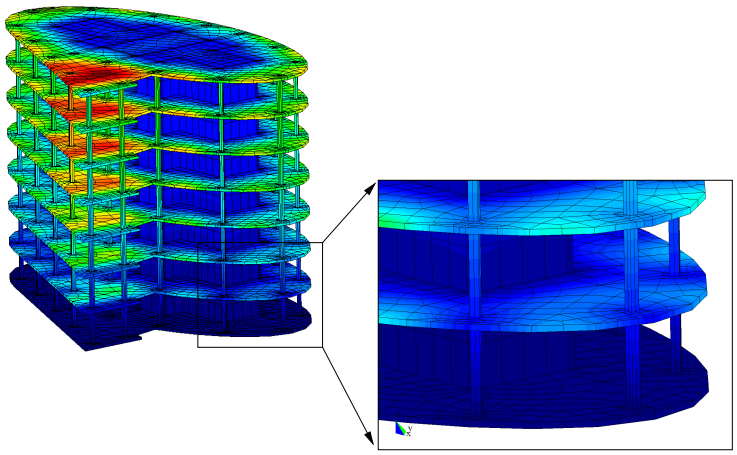

Fig. 1. Structural model of an office tower consisting of 11762 hexahedral p-version elements; displacement field of structure with zoomed view

5, which reflects the global behaviour of the system accurately enough. By using the hierarchical organisation in octrees, it is possible to zoom into the structure to a certain level in order to locally refine the computation simply by increasing the polynomial degree. Or, after indentifying critical areas on the global level, it is possible to perform design studies locally in order to explore different design alternatives. But, using the hierarchical approach presented in this paper, all these local computations can be done without recomputing the whole domain or without losing global consistency.

4 Hierarchical Approach

Before any hierarchical solver can be applied to a finite element discretisation, the corresponding data – stiffness matrices and load vectors – have to be set up in a hierarchical way. Starting with an octree generation for the elements itself results in a hierarchical sorting according to some criteria auch as the elements' centre, for instance. In a second step, all degrees of freedom (DOF) can be assigned to their corresponding nodes by evaluating the elements' Morton indices. Once finished with this initial setup, the system can be processed with

a solver of nested dissection type, e. g., consisting of a bottom-up assembly and top-down solution step.

4.1 Building a Finite Element Hierarchy

To sort the elements of a FE discretisation in a hierarchical way, each element has to be separately assigned to one of the octree's cells. For reducing the computational effort while generating the corresponding octree all elements are represented by their centre only. Without loss of generality this could be any arbitrary point of an element, such as one corner, as long as there's no other element with the same representative. For n elements this conforms to a set P of n point coordinates x, y, and z.
Under the assumption of storing exactly one point $p \in P$ in one cell, an octree representation for set P can be easily derived (see Fig. 2). All other cells stay empty and are not relevant as long as the initial finite element mesh isn't altered. For all non-empty cells the corresponding element data – stiffness matrix and load vector – can already be stored at the same location, too, as needed for the later computations.

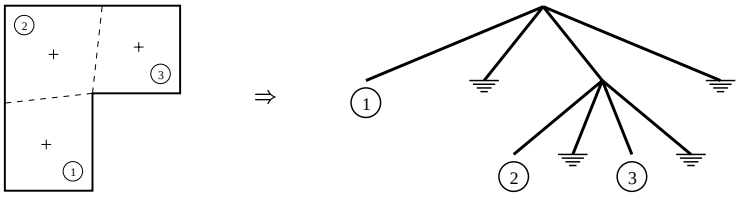

Fig. 2. A sample FE discretisation in 2D with three elements (left-hand side, '+' indicates an element's centre) and the corresponding quadtree – an octree's 2-dimensional counterpart – on the right-hand side

4.2 Assigning Degrees of Freedom

To finish the setup step all DOFs have to be assigned to the octree, too. As one DOF might belong to more than one element, 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, starting at the root node, from which all corresponding elements can still be reached; the octree's root obviously forms a common father node for any arbitrary elements. The lower one DOF can be assigned to the octree, the better for the later computations, because it can be eliminated earlier during the nested dissection's assembly.
Finding the LCF of some elements is achieved by comparing the respective Morton indices. They 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 LCF is the root node. Assume, all of the quadtree's sons in the right part of Fig.3 are labelled from '0' to '3' from left to right. The LCF of element 2 (Morton index '20') and element 3 (Morton index '22') is '2', the LCF of element 2 and element 1 (Morton index '0') is ' ' (empty) and, thus, the root node.
Assigning all DOFs to the octree finishes the setup and preparatory work. The original finite element mesh is no longer necessary as all further computations are directly processed on the tree structure. Exploiting this hierarchical ordering of elements/DOFs by a nested dissection algorithm is discussed in the next section. Figure 3 shows the DOF distribution for the small example from above.

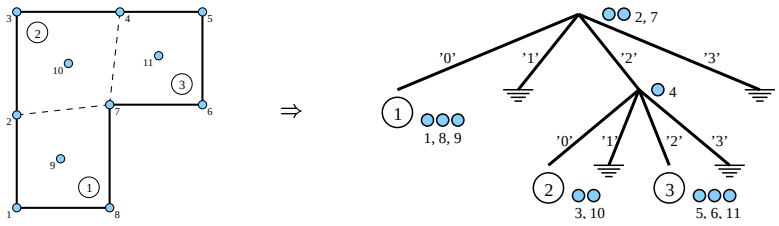

Fig. 3. Assuming the following DOFs (light circles) for the sample FE discretisation on the left-hand side, the final DOF assignment according to the elements' Morton indices is shown on the right-hand side.

4.3 Nested Dissection

Applying a nested dissection algorithm on finite element meshes was done very early by J.A. George [6]. The main idea behind this technique is to decompose the system of linear equations (SLE) into some smaller parts and to eliminate in a bottom-up step local unknowns, i. e. unknowns only partially describing the SLE at this point, before in a final top-down step the solution can be computed. Some more information about nested dissection, especially for solving the convection-diffusion-equation, can be found in [7], for instance.
In our case, the decomposition step can be skipped, because it was implicitly done when generating the finite element mesh for some geometric model. Hence, creating an octree for all elements and assigning the corresponding DOFs to the octree's nodes is all necessary work to be done here. For precise time measurements related to this setup step, see the results given in Sect. 5.
Once all preparatory work is finished, a bottom-up assembly is initiated. Therefore, each stiffness matrix is first rearranged that way, thus, all unknowns are separated in blocks of inner (I) – a corresponding DOF is stored in that node – and outer (O) ones, leading to four blocks II, IO, OI, and OO. If one node doesn't

contain any DOFs at all, all of the stiffness matrix's unknowns are treated as outer ones, hence, only one block OO results.
Thus, the SLE $K \cdot u = d$ with stiffness matrix K, solution vector u, and load vector d can be written as

$$\left(\begin{array}{c|c} K_{II} & K_{IO} \\ \hline K_{OI} & K_{OO} \end{array}\right) \cdot \left(\frac{u_I}{u_O}\right) = \left(\frac{d_I}{d_O}\right) . \tag{1}$$

Evaluating (1) leads to

$$K_{II} \cdot u_I + K_{IO} \cdot u_O = d_I \qquad \text{and} \qquad K_{OI} \cdot u_I + K_{OO} \cdot u_O = d_O ,$$

which can be rewritten as

$$\left(K_{OO} - K_{OI} \cdot K_{II}^{-1} \cdot K_{IO}\right) \cdot u_O = d_O - K_{OI} \cdot K_{II}^{-1} \cdot d_I . \tag{2}$$

In (2) any influence of inner unknowns u_I has been eliminated, thus, the resulting SLE only depends on outer unknowns u_O that are stored somewhere higher in the tree. There exist several methods to compute the expression $\widetilde{K}_{OO} := K_{OO} - K_{OI} \cdot K_{II}^{-1} \cdot K_{IO}$, the so called SCHUR complement. In our approach we've chosen a direct method by applying a GAUSSIAN elimination.
The SCHUR complement is then passed to the node's father, that assembles it with all other SCHUR complements of its sons. For the newly formed SLE the same steps are applied until the root node is reached. Here, all resulting unknowns are only inner ones, hence, the SLE can be solved. This solution is passed to the root's sons that now can modify their right side and solve the SLE for their inner unknows. Successively passing the solution to all of a node's sons until a leaf is reached finally results in the entire solution vector u.

4.4 Exploiting the Hierarchy

One huge advantage of this hierarchical approach lies in the reduction of computations whenever the underlying geometric model changes. As described above, the p-version of the finite element method allows the alteration of single elements without the necessity of a complete FE mesh generation from the scratch. Thus, only parts of the tree have to be re-assembled according to the new stiffness matrices before the new solution vector can be computed.
Assume, the stiffness of one element changes. Starting from the root node all *Schur* complements of nodes visited on the way down to the node representing this element are obsolete and can be discarded. In fact, the amount of necessary assembly steps is directly related to the node's depth in the tree. Whenever a new assembly is initiated, only for those nodes without SCHUR complement some effort has to be invested, as for all others the SCHUR complements still exist from the last pass. This obviously diminishes the overall amount of computing time as you can see in the results presented in the next section.
Embedding this approach into the framework presented in [1] allows participating experts to study different model alternatves in a more efficient way due to shorter computing times in case of geometry alterations and local refinements, resp.

5 Some Examples and Results

To show the potential behind this hierarchical approach a sample prototype was implemented. Based on two different scenarios with different polynomial degree times were meassured for the setup step, the assembly step, and the solution step. Afterwards some elements were exchanged by newer versions[4] and the times were meassured for the re-assembly as well as for the new solution step. All computations were done on an Intel Pentium 4 with 3.4 GHz under Linux.

5.1 Example 1: A Simple Cube

This artificial example shows a cube with an equidistant discretisation in each direction, consisting of 144 elements in total. It has 2625 DOFs ($p = 2$) and 7935 DOFs ($p = 4$), resp. The necessary octree to store all data has a depth of four, counting the root level as zero. For simulating a geometry alteration an element on the lowest level was updated with a new stiffnes matrix and load vector. The achieved results are given in the following table.

Name	DOFs R	DOFs T	Setup	Assembly	Solution	Re-Assembly	Solution
cube_p2	897	2625	0.42 s	0.21 s	0.11 s	0.03 s	0.10 s
cube_p4	2319	7935	2.66 s	6.74 s	1.03 s	0.50 s	1.02 s

As one can see, the times for a re-assembly – possible due to our approach – are significant smaller than the ones for the initial assembly. The more complex the problem becomes (total amout of DOFs) the more benefit can be achieved. One thing that also could be observed is a declining percentage of DOFs to be stored on the root level (in the following table labelled as 'DOFs R' for the root level and as 'DOFs T' for the overall amount). Nevertheless, this example with nearly 30 % is a real worse one.

5.2 Example 2: An Office Tower

The second and more realistic example consists of two floors from an office tower that can be visited in Vienna[5] (also see Fig. 1). It consists of 4171 elements and was computed for polynomial degrees $p = 1$ (23856 DOFs) and $p = 2$ (84660 DOFs), the necessary octree for storage has a depth of eight. Compared to the example above, a much better percentage of DOFs on the root level could be observed. For $p = 2$ only 8 % of all DOFs are stored there.

Name	DOFs R	DOFs T	Setup	Assembly	Solution	Re-Assembly	Solution
uniqa_p1	2544	23856	1.89 s	19.27 s	12.64 s	5.14 s	12.09 s
uniqa_p2	6414	84660	10.94 s	343.21 s	77.28 s	14.77 s	76.36 s

[4] The time for exchanging some stiffness matrices can be neglected as it is in the order of some milliseconds.

[5] http://tower.uniqa.at

Here, also the times for a re-assembly are way smaller than for the initial assembly – around 15 s instead of 343 s for $p = 2$. If you now take into account that a solution of the entire system with all 84660 DOFs takes nearly 200 s (cg algorithm) and in case of a geometry alteration everything has to be computed from the scratch, the 90 s (re-assembly plus solution) are only half that time. At the moment, most of the time (approx. 99 %) ist spent at root level due to the simple cg solver we use; a hierarchical multilevel preconditioner will reduce this effect and, thus, emphasize the advantages of out approach even more. Nevertheless, the efficiency of this approach could be shown. Next steps will comprise to test different solver strategies for the root level and a parallelisation, thus, even larger problems with higher polynomial degree can be computed.

6 Conclusion

In this paper, 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 geometric alterations, as only parts directly influenced have to be recomputed. Thus, studies of model alternatives and local refinements become very attractive due to the reduced computing times compared to standard approaches that always have to start from the scratch. Finally, this is the first step into the direction of completely embedded simulation processes as they appear in many technical applications.

References

1. Mundani, R.P., Bungartz, H.J.: An octree-based framework for process integration in structural engineering. In: Proc. of the 8th World Multi-Conf. on Systemics, Cybernetics and Informatics. Volume II., Int. Institute of Informatics and Systemics (2004) 197–202
2. Mundani, R.P., Bungartz, H.J., Rank, E., Romberg, R., Niggl, A.: Efficient algorithms for octree-based geometric modelling. In: Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing, Civil-Comp Press (2003)
3. Szabó, B.A., Babuška, I.: Finite Element Analysis. John Wiley & Sons (1991)
4. Düster, A., Bröker, H., Rank, E.: The p-version of the finite element method for three-dimensional curved thin-walled structures. Int. Journal for Numerical Methods in Engineering **52** (2001) 673–703
5. Düster, A., Scholz, D., Rank, E.: pq-Adaptive solid finite elements for three-dimensional plates and shells. submitted to Computer Methods in Applied Mechanics and Engineering (2005)
6. George, J.A.: Nested dissection of a regular finite element mesh. SIAM Journal on Numerical Analysis **10** (1973) 345–363
7. Bader, M.: Robuste, parallele Mehrgitterverfahren für die Konvektions-Diffusions-Gleichung. PhD thesis, Fakultät für Informatik, Technische Universität München (2001)
8. Düster, A.: High Order Finite Elements for Three-Dimensional, Thin-Walled Nonlinear Continua. PhD thesis, Lehrstuhl für Bauinformatik, Technische Universität München (2001)