Space Tree Structures for PDE Software

Michael Bader[1], Hans-Joachim Bungartz[2], Anton Frank[3], and Ralf Mundani[2]

[1] Dept. of Informatics, TU München, D-80290 München, Germany
[2] IPVR, Universität Stuttgart, D-70565 Stuttgart, Germany
[3] 4Soft GmbH, D-80336 München, Germany

Abstract. In this paper, we study the potential of space trees (boundary extended octrees for an arbitrary number of dimensions) in the context of software for the numerical solution of PDEs. The main advantage of the approach presented is the fact that the underlying geometry's resolution can be decoupled from the computational grid's resolution, although both are organized within the same data structure. This allows us to solve the PDE on a quite coarse orthogonal grid at an accuracy corresponding to a much finer resolution. We show how fast (multigrid) solvers based on the nested dissection principle can be directly implemented on a space tree. Furthermore, we discuss the use of this hierarchical concept as the common data basis for the partitioned solution of coupled problems like fluid-structure interactions, e. g., and we address its suitability for an integration of simulation software.

1 Introduction

In today's numerical simulations involving the resolution of both time and space, we are often confronted with complicated or even changing geometries. Together with increasing accuracy requirements, this fact is responsible for some kind of dilemma: On the one hand, orthogonal or Cartesian grids are simpler with respect to mesh generation, organization, and changes, but need very or even too high levels of refinement in order to resolve geometric details in a sufficient way. Unstructured grids, on the other hand, are clearly better suited for that, but entail costly (re-) meshing procedures and an often significant overhead for grid organization.

For the Cartesian world, one possibility to get out of this dilemma is to decouple the resolutions of the *geometric* grid (used for geometry representation and discretization) and of the *computational* grid (used for the (iterative) solution process). This seems to be justified by the fact that orthogonal grids come along with an $O(h)$ error concerning geometry, but are able to produce $O(h^2)$ discretization errors for standard second order differential operators. Hence, for a balance of error terms, it is definitely not necessary to iterate over all those tiny cells needed to resolve geometry, if some way is found to collect geometric details from very fine cells for the discrete equations of a surrounding coarser cell. Space trees provide this possibility, and they do it within the same data structure. This aspect is important, since the accumulation of geometric details

© Springer-Verlag Berlin Heidelberg 2002

is not some kind of a "once-and-for-all" process, but there will be situations where the fine world has to be revisited (in case of adaptive refinement of the computational grid, for example).

In the following, we first summarize the principal ideas and properties of the hybrid space tree concept. Then, we demonstrate how space trees can be directly used for grid organization and representation in a PDE context by implementing a nested-dissection-based fast iterative solver. Afterwards, the use of space trees as the common geometry representation in a software environment for the partitioned solution of coupled problems and their potential for an embedding of PDE software into a broader context (integration of CAD and numerical simulation, for example) are discussed. Finally, we give some conclusions and an outlook over future work in this field.

2 Space Trees

Space trees [3, 6] are generalized quad- or octrees [5, 9–11]. The first generalization refers to the now arbitrary number d of dimensions. A second generalization is their ability to associate data not only with d-dimensional cells (i. e. volumes), but also with the $d-1$-dimensional hypersurfaces representing the boundary of a cell (and so on, recursively) – which is important if we think of boundary value problems. Hence, the basis is a successive spatial partitioning of a square (cube) in four (eight) attributed congruent subsquares (subcubes). Among the fields of application for such hierarchical structures, there are image processing, geometric modelling, computer graphics, data mining, and many more. For our purposes, the most interesting features of space trees are the reduced complexity (the boundary of an object determines its storage requirements, see the left part of Fig. 1), the availability of efficient algorithms for set operations, neighbour

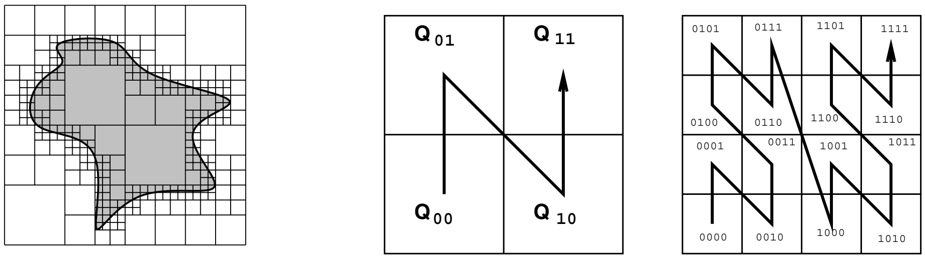


Fig. 1. Quadtree: successive refinement (left) and relations to Morton ordering and Lebesgue's space-filling curve (centre and right; helpful for a simple and efficient parallelization of computations on very heterogeneous grids)

detection, or movement, the inherent information for data sequentialization (see the right part of Fig. 1; this is especially important for a simple parallelization), and, finally, the possibility of compact implementations via bit coding, as long as we are only interested in the merely geometric information. For the remainder, the discussion is restricted to 2 D without loss of generality.

For some given geometry, which may be a technical object in a CAD representation or a measured or analytically described domain, the first step is to generate a space tree to hold the complete geometric information, but nothing else. This means that we have to choose a very high characteristic resolution h_g (the resolution of the input data, for example), but that we can use a compact implementation like a bit-coded linearized tree. Note that, if nevertheless too big for main memory, the space tree does not have to be kept there at one time as a whole. As a next step, the *computational* grid with characteristic mesh width h_c, i. e. the set of cells which will be related to degrees of freedom later, has to be built up. This will typically be a small subset of the above space tree, but can also contain finer cells than the geometric grid (in subdomains without any geometric details and, hence, coarse geometric cells, for example). Concerning the concrete implementation, think of a standard tree structure with several floating point variables in the nodes, which is explicitly generated and kept in main memory. Hence, following the maxim that the problem's physics and not its geometry should determine the level of detail during computations, we now have a hybrid or decoupled overall representation within the space tree concept. Figure 2 illustrates the relations between geometric and computational grid.

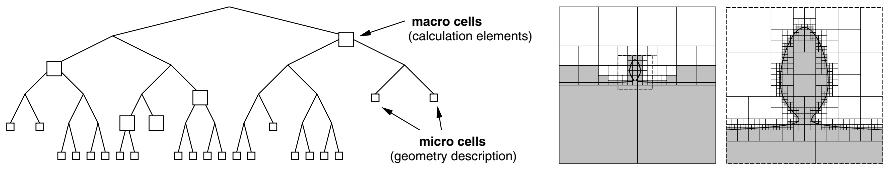
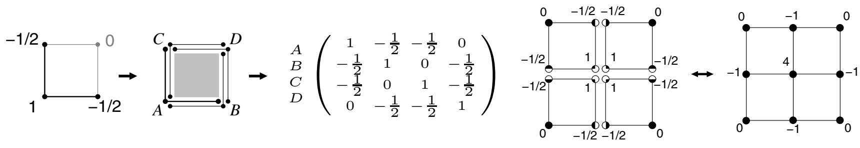


Fig. 2. Hybrid concept: macro and micro layer in the data structure (left) and applied to a simple example (right)

Obviously, the crucial part is the interplay between both grids: How can geometric details influence the discrete equations on coarse cells (and, thus, improve the quality of the computed results) without being stored or visited during an iterative solution process? For that, we need some accumulation step during which details from the micro cells to be neglected later are used to assemble (modified) discretization stencils in the macro cells. Such a global accumulation has to be done as a pre-processing once at the beginning, and it may have to be repeated partially each time the computational grid changes due to local adaptive refinement. Apart from these accumulation steps, the micro layer is not needed for the solution of the discretized equations.

The accumulation starts at the leaves of the micro layer with some atomic stencils or element matrices which, of course, depend on the given PDE, on the chosen discretization scheme (finite differences or elements, e. g.), and on the local geometry (a cell's corner outside the problem's domain will influence the atom and, hence, the following). Next, four neighbouring atoms are assembled

– in the sense of both stencils and matrices. Figure 3 shows such an atom and the assembly for the simple case of the standard finite difference approach for the Laplacian with all involved points lying within the domain. Since we do not



−1/2
0
1
−1/2
C
D
A
B
$$\begin{array}{c|cccc} A & 1 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ B & -\frac{1}{2} & 1 & 0 & -\frac{1}{2} \\ C & -\frac{1}{2} & 0 & 1 & -\frac{1}{2} \\ D & 0 & -\frac{1}{2} & -\frac{1}{2} & 1 \end{array}$$
0 −1/2 −1/2 0
−1/2 1 1 −1/2
−1/2 1 1 −1/2
0 −1/2 −1/2 0
0 −1 0
−1 4 −1
0 −1 0

Fig. 3. Atoms (left, associated with cells or elements) as stencils or matrices (centre) and their assembly (right)

want to keep the fine grid points as degrees of freedom for the computations on the macro layer, a hierarchical transformation separating coarse and fine points is applied. The fine points are eliminated as in a nested dissection process (see Sect. 3) and even explicitly removed. Now, we have again atoms – related to larger cells than before, but again with four grid points involved. The whole process is illustrated in Fig. 4.

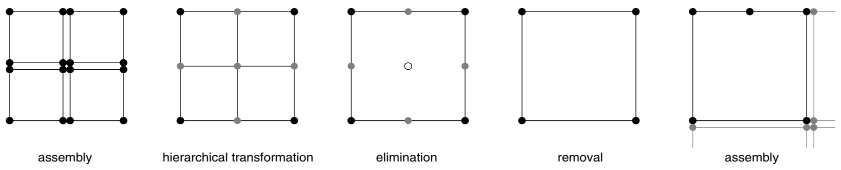


Fig. 4. Accumulating geometric detail information: assembly, hierarchical transformation, elimination, removal, and next assembly

This completes the description of the geometry accumulation process. Now,

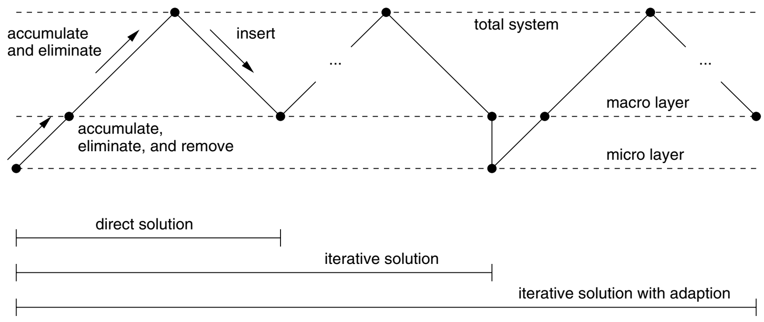


Fig. 5. Accumulation of geometric details and following direct or iterative solution on the computational grid

the system (with the local stencils or matrices derived above) must be solved on the cells of the macro layer. For that, principally, a direct or an iterative algorithm can be used (see Fig. 5). The design of suitable solvers will be studied in the next section. Here, we just present some numerical results for a Poisson equation on the domain shown in Fig. 6. Obviously, the coarse grid with $h_c = 2^{-4}$

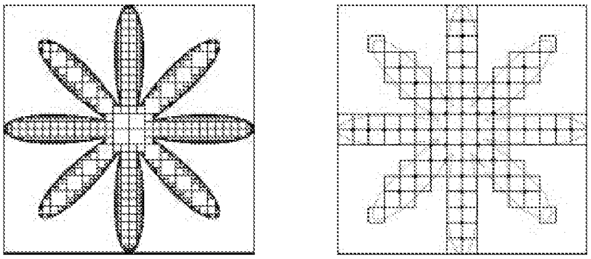


Fig. 6. 2 D star domain: used geometric grid (adaptive, finest occurring h is $h_g = 2^{-8}$; left) and used computational grid (regular, $h_c = 2^{-4}$; right)

is not able to produce reasonable results without the collected micro details. With our hybrid approach, however, the quality of the obtained solution is of the same order as the quality of the solution computed on the fine grid with $h_c = h_g = 2^{-8}$ – with the hybrid solution being less costly w.r.t. both memory and computing time (see Fig. 7).

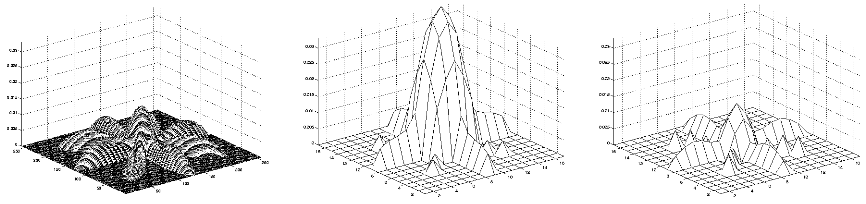


Fig. 7. Solutions of Poisson's equation on the star domain: fine level $h_g = h_c = 2^{-8}$ for geometric and computational grid (left), coarse level $h_g = h_c = 2^{-4}$ for both grids (centre), and hybrid approach $h_g = 2^{-8}, h_c = 2^{-4}$ (right)

3 Fast Solvers on Space Trees

The above accumulation process stops on the macro layer. Thus, it provides element matrices in the finest computational cells. Now, we have a closer look at the solver for the linear system of equations corresponding to this macro element information. Since the space tree idea is based on a recursive substructuring of the domain, a nested dissection approach [7] turns out to be a quite natural choice. In order to make nested dissection's successive bisections of the domain consistent with the space tree subdivision into 2^d successor nodes, we have to perform and combine d alternate bisections (one for each dimension). Hence, in the following, we can restrict the presentation to the standard case of a bisection in 2 D.

On each node of the space tree between its root and the macro leaves where geometry accumulation has provided element stencils or matrices, we define local sets $\mathcal{I}$ and $\mathcal{E}$ of unknowns related to grid points inside or on the boundary of the local cell (subdomain), resp. Due to the recursive bottom-up elimination of the nested dissection scheme, $\mathcal{I}$ is restricted to points on the so-called *separator*, the line separating the two subdomains of the local cell. For the precise definition of the iterative solver, we introduce a further set $\mathcal{G}$ of *coarse grid unknowns*, which will form the coarse grid in the sense of a multilevel solver. If the local system of equations is restricted to the unknowns in $\mathcal{G}$ (which is actually what is done during the geometry setup), we should get a system that describes physics on this coarser level sufficiently well. Figure 8 illustrates this classification.

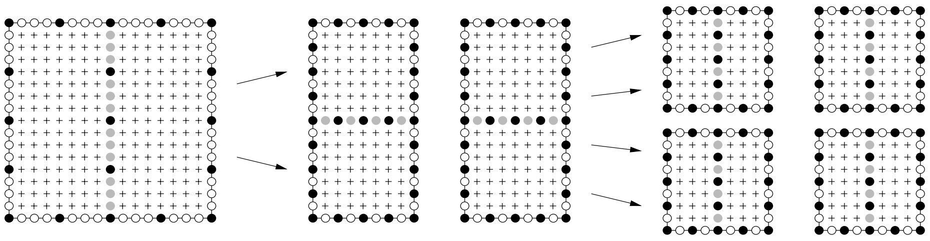


Fig. 8. Recursive substructuring by alternate bisection: Unknowns in $\mathcal{E}$ are painted in white, unknowns in $\mathcal{I}$ in grey. If the unknowns are also in $\mathcal{G}$, they are painted as black nodes. The little crosses indicate unknowns that are no longer present on the local subdomain due to their elimination on the child domains.

Starting at the leaves of the macro layer, we perform a block elimination on each local system of equations based on the partitioning of the unknowns:

$$\underbrace{\begin{pmatrix} \mathrm{Id} & -A_{\mathcal{E}\mathcal{I}}A_{\mathcal{I}\mathcal{I}}^{-1} \\ 0 & \mathrm{Id} \end{pmatrix}}_{=:\,L^{-1}} \underbrace{\begin{pmatrix} A_{\mathcal{E}\mathcal{E}} & A_{\mathcal{E}\mathcal{I}} \\ A_{\mathcal{I}\mathcal{E}} & A_{\mathcal{I}\mathcal{I}} \end{pmatrix}}_{=\,A} \underbrace{\begin{pmatrix} \mathrm{Id} & 0 \\ -A_{\mathcal{I}\mathcal{I}}^{-1}A_{\mathcal{I}\mathcal{E}} & \mathrm{Id} \end{pmatrix}}_{=:\,R^{-1}} = \underbrace{\begin{pmatrix} \widehat{A}_{\mathcal{E}\mathcal{E}} & 0 \\ 0 & A_{\mathcal{I}\mathcal{I}} \end{pmatrix}}_{=:\,\widehat{A}}, \qquad (1)$$

where $\widehat{A}_{\mathcal{E}\mathcal{E}} := A_{\mathcal{E}\mathcal{E}} - A_{\mathcal{E}\mathcal{I}} \cdot A_{\mathcal{I}\mathcal{I}}^{-1} \cdot A_{\mathcal{I}\mathcal{E}}$ is the so-called *Schur complement.* Thus, the full information from $\mathcal{I}$ is preserved in $\widehat{A}_{\mathcal{E}\mathcal{E}}$. The submatrix $\widetilde{A}_{\mathcal{E}\mathcal{E}}$ is then transferred to the father who collects and assembles the local systems of his two sons and proceeds recursively with block elimination and assembly until the root of the space tree, i. e. the cell representing the overall domain of our problem, is reached. After this bottom-up assembly of equations, we start from the root and use the unknowns in the local $\mathcal{E}$ (available from the boundary conditions or from the respective father node) to compute the unknowns in the local $\mathcal{I}$ on every subdomain in a top-down traversal.

So far, the approach leads to a direct solver quite similar to the original nested dissection method from [7]. Likewise, its computing time grows like $\mathcal{O}(N^{3/2})$ with the number of unknowns N (2D). Since this, of course, is too expensive, the block elimination (1) should be replaced by some iterative scheme (a suitable preconditioner, e. g.). Then, the single top-down solution pass is replaced by a sequence of top-down (compute approximations based on the current residuals) and bottom-up (collect updated residuals from the leaves to the root) traversals, see also Fig. 5.

In our solver, we use the transformation to hierarchical bases or generating systems [8] as a preconditioner,

$$\underbrace{H^T A H}_{=:\,\bar{A}}\,\bar{x} = \underbrace{H^T b}_{=:\,\bar{b}}, \tag{2}$$

the latter leading to a true multigrid method. In both cases, the preconditioning can be further improved by introducing an additional partial elimination based on the set $\mathcal{G}$ of coarse grid unknowns:

$$\underbrace{L^{-1}\bar{A}\,R^{-1}}_{=:\,\widetilde{A}}\,\widetilde{x} = \underbrace{L^{-1}b}_{=:\,\widetilde{b}}. \tag{3}$$

The elimination matrices L^{-1} and R^{-1} are chosen such that all couplings between unknowns in $\mathcal{G}$ are eliminated explicitly in the resulting system matrix $\widetilde{A}$. The set $\mathcal{G}$ should consist of those unknowns that are expected to be strongly coupled. Hence, a good heuristics for choosing $\mathcal{G}$ can often be derived from the underlying physics of the respective PDE.

For Poisson equations, it is usually sufficient to choose just the unknowns on the corners of the subdomains. This is consistent with the accumulation process from Sect. 2 and leads to a multilevel method close to standard multigrid with uniformly coarsened grids. For other PDEs, however, this simple choice may no longer be appropriate. For convection diffusion equations, for example, especially in the case of increasing strength of convection, additional unknowns on the boundary of the local subdomain should be used for $\mathcal{G}$. This corresponds very much to using semi-coarsening in classical multigrid methods. In [2], we present a method that increases the number of coarse grid unknowns proportionally to the square root of the number of local unknowns in the cell. This approach balances the influence of both diffusion and convection in each cell. If convection is not too strong (mesh Péclet number bounded on the finest grid), the resulting algorithm has a complexity of $\mathcal{O}(N)$ with respect to both computing time and memory requirements [2].

Figure 9 (right) shows the convergence rates for the Poisson equation on the star domain from Fig. 6 and for a convection diffusion problem with circular convection field and varying strength of convection (centre). The streamlines of the convection field are given in the leftmost picture. Homogeneous Dirichlet

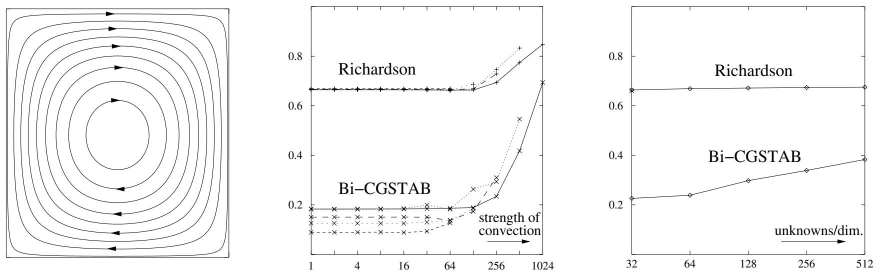


Fig. 9. Convergence rates for the convection diffusion problem (centre) with corresponding convection field (left) and for the Poisson equation on the star domain

boundary conditions were used in both examples. The results indicate that the convergence rates are generally independent of the number of unknowns and also constant for increasing strength of convection. However, this holds only up to a certain upper bound which depends on the mesh size of the finest grid. At least to some extent, the rates are also independent of the geometry of the problem's domain (see [1] for a more detailed discussion). In both examples, using the method as a preconditioner for Bi-CGSTAB can further improve the overall performance of the algorithm.

With the presented iterative solver, we are now able to efficiently solve a PDE on complicated domains with simple Cartesian grids. Due to the logical separation of the resolution of geometry and computations, there are no drawbacks concerning accuracy compared to more complicated unstructured grids. The further potential of space trees is studied in the next section.

4 Coupled Problems, Software Integration

Coupled or multi-physics problems involve more than one physical effect, where none of these effects can be solved separately. One of the most prominent examples is the interaction of a fluid with a structure – think of a tent-roof construction exposed to wind, or of an elastic flap in a valve opened and closed by some fluid. The numerical simulation of such interactions, which is not in the centre of interest here, is a challenging task, since expertise and model equations from different disciplines have to be integrated, and since the problem's geometry is not stationary (see [12], e. g.). In order to profit from existing models and code for the single phenomena and, hence, to reduce necessary new developments to the mere coupling of the different parts, the *partitioned solution* strategy based on a more or less sophisticated alternate application of single-effect solvers is very widespread. Figure 10 shows the basic structure of a possible modular framework that has been developed for the partitioned solution of problems with fluid-structure interactions [4, 6]. Input, output, the two solvers, and the coupling itself are strictly separated. The direct interplay of different codes entails the necessity to switch from one geometry representation of the common domain

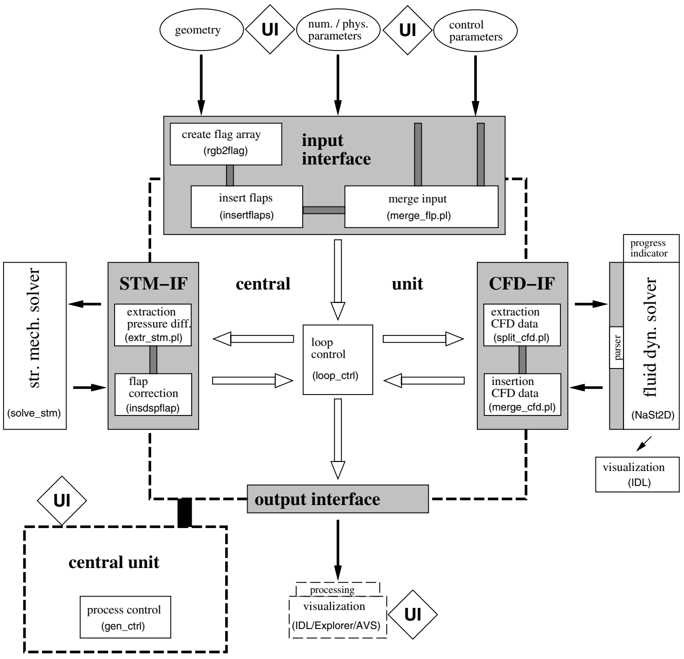

geometry
UI
num. / phys. parameters
UI
control parameters
create flag array (rgb2flag)
input interface
insert flaps (insertflaps)
merge input (merge_flp.pl)
progress indicator
str. mech. solver (solve_stm)
STM–IF
extraction pressure diff. (extr_stm.pl)
flap correction (insdspflap)
central
unit
loop control (loop_ctrl)
CFD–IF
extraction CFD data (split_cfd.pl)
insertion CFD data (merge_cfd.pl)
parser
fluid dyn. solver (NaSt2D)
visualization (IDL)
UI
output interface
central unit
process control (gen_ctrl)
processing
visualization (IDL/Explorer/AVS)
UI

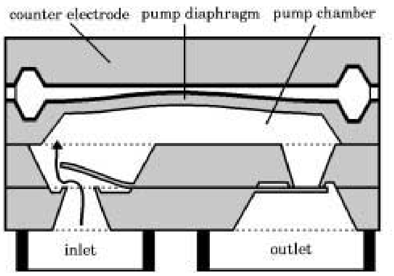
Fig. 10. Modular software environment for fluid-structure interactions (see [6])

to another. In our approach, we use space trees as the central data format. That is, if one of the solvers is to be exchanged, only a transformation between this solver's internal geometry representation and the central space tree has to be provided, which supports modularity. Again, the space tree allows us to keep geometric information at a very high resolution and to easily process changes of the geometry. Figure 11 shows the application of our software environment for coupled problems to the flow through a valve-driven micropump.

Finally, note that space trees are well-suited as general data basis for the integration of CAD, simulation, and visualization software [4]. For example, the bit-coded tree keeps the whole geometric information and allows for efficient CAD operations (affine transforms, collision tests, and so on).

5 Concluding Remarks

Space trees combines the simplicity, flexibility, and universality of Cartesian grids with the superior approximation properties of unstructured grids. In this paper, their main features as well as a fast linear solver and some attractive fields of application have been studied. Future work will cover the use of space trees for more complicated PDEs like the Navier-Stokes equations.



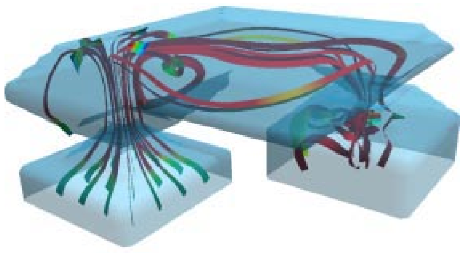


Fig. 11. Cross-section through a valve-driven micropump (left) and visualization of the simulated fluid-structure interaction at its valves (right; pump in octree representation and stream bands)

References

1. M. Bader. *Robuste, parallele Mehrgitterverfahren für die Konvektions-Diffusions-Gleichung.* PhD thesis, TU München, 2001.
2. M. Bader and C. Zenger. A robust and parallel multigrid method for convection diffusion equations. *ETNA*, 2002 (accepted).
3. P. Breitling, H.-J. Bungartz, and A. Frank. Hierarchical concepts for improved interfaces between modelling, simulation, and visualization. In B. Girod, H. Niemann, and H.-P. Seidel, editors, *Vision, Modelling, and Visualization '99*, pages 269–276. Infix, St. Augustin, Bonn, 1999.
4. H.-J. Bungartz, A. Frank, F. Meier, T. Neunhoeffer, and S. Schulte. Efficient treatment of complicated geometries and moving interfaces for CFD problems. In H.-J. Bungartz, F. Durst, and C. Zenger, editors, *High Performance Scientific and Engineering Computing*, volume 8 of *LNCSE*, pages 113–123. Springer-Verlag, Heidelberg, 1999.
5. R. A. Finkel and J. L. Bentley. Quad trees: a data stucture for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.
6. A. Frank. *Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung.* PhD thesis, TU München, Herbert Utz Verlag, München, 2000.
7. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10, 1973.
8. M. Griebel. *Multilevelmethoden als Iterationsverfahren auf Erzeugendensystemen.* Teubner Skripten zur Numerik, Stuttgart, 1994.
9. C. L. Jackins and S. L. Tanimoto. Oct-trees and their use in representing 3 D objects. *Computer Graphics and Image Processing*, 14:249–270, 1980.
10. D. Meagher. Geometric modelling using octree encoding. *Computer Graphics and Image Processing*, 19:129–147, 1982.
11. H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, 1989.
12. S. Schulte. *Modulare und hierarchische Simulation gekoppelter Probleme.* PhD thesis, TU München, VDI Reihe 20 Nr. 271. VDI Verlag, Düsseldorf, 1998.