A Computational Framework for Topological Operations

Michael Spevak[1], René Heinzl[2], Philipp Schwaha[2], and Tibor Grasser[2]

[1] Institute for Microelectronics, TU Vienna, spevak@iue.tuwien.ac.at
[2] Christian Doppler Laboratory at the Institute for Microelectronics, TU Vienna

Abstract. We present an orthogonal topological framework which is able to provide incidence traversal operations for all topological elements. The run-time performance of this topological traversal operations can be optimized at a highly expressive level, where the abstraction penalty imposed by this approach is negligible. For the topological storage we use vertex-cell as well as cell-vertex incidence information. Using meta-programming and the archetype concept we can optimize traversal of inter-dimensional objects without explicitly storing them.

1 Introduction

The field of scientific computing often imposes highly complex formulae with quantities of different topological elements. Many applications such as the discretization of partial differential equations (PDE) as well as interpolation mechanisms strongly depend on the base traversal mechanisms the environment provides.

We present a set of base traversal operations which is sufficient for all applications. This approach results in a rigorous implementation of topological structures which covers all types of topological elements such as vertices, cells and general inter-dimensional elements called faces. The expressiveness of the code is increased, because we do not need to write traversal algorithms explicitly for each of the elements such as edge-cell traversal but the information can be derived by a subset of highly optimized operations.

Using the generic programming paradigm in combination with parametric polymorphism in C++ we have the opportunity to formulate a concept for a topological base structure. It provides an interface for the construction, traversal and handle generation for different topological structures. Based on the topological structure it is possible to formulate discrete problems independently of the actual underlying topology, e.g. a finite volume discretization scheme can be formulated independently from the topology and the dimension.

2 Related Work

In the last decade there have been many approaches towards implementing a general purpose simulation environment for partial differential equations. Most of

these have implemented the underlying minimal topological structures necessary for discretization, which on the one hand does reduce resource use, but comes at the cost of greatly diminishing the flexibility of topological traversal.

The major step towards a more flexible use of topological structures is presented in [4]. The grid algorithm library introduces the first cursor [2] based approach on multi-dimensional data structures.

Most of the other environments either completely veil the topological information by formalisms like element matrices [3] or control functions [6], commercial simulation tools provide access by the formulation of the final PDE such as FEMLab. For this reason calculations which use non-standard traversal mechanisms are cumbersome or impossible to specify.

3 Conception and Implementation

The main aim of our topological container interface is to provide mechanisms for insertion, traversal and handling. The most important concept requirement for the topological data structure is the retrieval of cell-vertex and vertex-cell incidence information in constant time. Even though it is redundant to keep incidence vertex-cell as well as cell-vertex information, if these methods are available in constant time, we can also derive traversal algorithms for arbitrary incidence in constant time, which would not be possible if we only use the vertex-cell incidence information. Storing this information requires the least memory of all possible storage methods which require constant time for incidence traversal.

Topological elements are described with so called handles, which are in general integers and uniquely identify the element within all elements of the same dimension.

If we have a structured grid it is possible that incidence information can be derived implicitly. This means that we only store the number of ticks per dimension and generate all handles as well as traversal information from this information.

The basic topology only has to provide cursor access to all its cells and vertices as well as the incidence information from cells to vertices and vice versa. We introduce the main topological concept in the following table. As we only

Name	Description	Requirements
vertex_cursor	Cursor over vertices	Cursor Concept
vertex_begin(), vertex_end()	Cursor range	const
cell_cursor	Cursor over cells	Cursor Concept
cell_begin(), cell_end()	Cursor range	const
vertex_on_cell_cursor	Local traversal Cursor	constructable with cell
cell_on_vertex_cursor	Local traversal Cursor	constructable with vertex

Fig. 1. Concepts for the topological base structure

store the incidence information between cells and vertices we have to specify the incidence information of inter-dimensional elements. Even though it is possible to introduce data types for each kind of element we use archetypes which are

in general less error prone and provide higher flexibility compared to manually coded traversal.

The archetype concept [4] introduces local faces within a cell. For example a topological 2-simplex consists of three vertices and three 1-faces (edges). The archetype can be shown either as simple graph or Hasse diagram (Fig. 2). All the archetype information is known at compile time, as we assume that archetypes do not change during run time. Using the mechanisms of meta-programming [1] we obtain high performance traversal routines. If both, archetype as well as

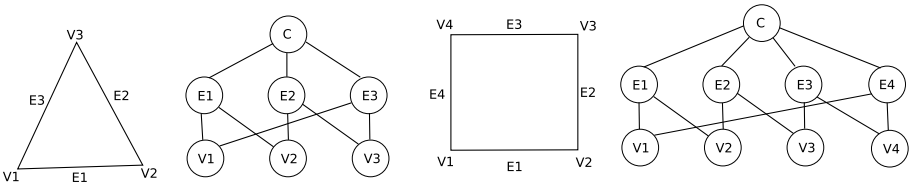

Fig. 2. The graph as well as the Hasse Diagram of the 2-simplex as well as a 2-cuboid archetype.

cell-vertex incidence information is available we can derive incidence information between elements of arbitrary dimension. Given, that the face can be identified uniquely by all covered vertices, we can derive a unique handle for each of the faces. As already mentioned we use the random access cursor concept in order to provide access to the topological elements. We refine the cursor concept [2] for data structural convenience. In contrast to the iterator concept, dereferentiation of a cursor does not provide the data content but a handle. This handle is used as a key to access a property map or quantity [5] in order to obtain the contained data. Our cursor concept is a refinement of the random access cursor concept and introduces the following refinements.

Name	Description	Requirements
bool valid()	Validity of cursor (not end)	const
cursor end()	past end of cursor validity	const
void reset()	set to the start point	const

Fig. 3. Concept refinement for the cursor within the topological framework.

4 Performance Analysis

In order to test the performance we create a homogeneous structure with different element sizes in two and three dimensions and perform different traversal operations. We show the performance of insertion, the vertex cursor, the cell on vertex cursor as well as the edge on vertex cursor.

All traversal operations are independent of the total number of elements and only depend on the local patch size of the base element. This implies that in

numerical schemes with an effort bigger than $O(n)$ the topological operations are not the limiting factor.

Especially for the solution of discretized partial differential equations using different discretization schemes the influence of the topology on the total simulation time is less relevant. The performance obtained by the topological framework is comparable to a hand optimized C implementation. In the full paper we will show a rigorous comparison of different topological implementations in C as well as C++.

5 Conclusion

We have shown a framework which is able to provide information and incidence traversal of all topological elements. We used the programming paradigms of generic programming as well as meta-programming in order to formulate the problems on an abstract level without incurring a relevant abstraction penalty.

References

1. David Abrahams and Aleksey Gurtovoy. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series)*. Addison-Wesley Professional, 2004.
2. David Abrahams, Jeremy Siek, and Thomas Witt. New iterator concepts. Technical Report N1477=03-0060, ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming Language C++, 2003.
3. Wolfgang Bangerth. Using modern features of C++ for adaptive finite element methods: Dimension-independent programming in deal.II. In Michel Deville and Robert Owens, editors, *Proceedings of the 16th IMACS World Congress 2000, Lausanne, Switzerland, 2000*, 2000. Document Sessions/118-1.
4. Guntram Berti. Gral - the grid algorithms library. In *ICCS '02: Proceedings of the International Conference on Computational Scnce-Part III*, pages 745–754, London, UK, 2002. Springer-Verlag.
5. R. Heinzl, M. Spevak, P. Schwaha, and T.Grasser. Concepts for High Performance Generic Scientific Computing. In *5th Mathmod Proceedings*, volume 1, 2006.
6. Connor S. Rafferty and R. Kent Smith. Solving partial differential equations with the prophet simulator, 1996.