Spectral Element Method

Geometry

Software Manual

Joseph Schoonover

Contents

1 Purpose and Overview

Spectral element methods are gaining popularity in the realm of geophysics. The Discontinuous Galerkin Spectral Element Method (DGSEM) is being applied in studies of earthquakes (**?**), general ocean circulation through the shallow water equations (**?**) and hydrostatic primitive equations (**?**). Additionally, the Continuous Galerkin Spectral Element Method (CGSEM) is being applied to elliptic and parabolic systems for studying topographic waves (**?**), *and other phenomena*. These techniques are flexible in that the computational core of these algorithms are geometry independent, provided the physical domain is decomposed into an arrangement of quadrilateral (in 2-D) or hexahedral (3-D) spectral elements. Neither open-source nor propietary mesh generation software has met the needs of spectral mesh generation (at the time this document was written). The popularity of these methods for there use in problems with complicated geometry has motivated the development of a collection of data-structures, modules and programs to aid in the generation of spectral element meshes.

The purpose of the software described herein is two-fold :

1. Provide the necessary data structures to store and query spectral element mesh information for is use in DG or CGSEM algorithms.

2. Use the data structures for developing an interactive mesh generation program which provides a suite of mesh quality checks.

The latter of these two is part of a long term goal towards developing a self contained spectral element methods software package. Eventually, it would be ideal to port these Fortran modules with a Graphical User Interface which would allow for inuitive mesh generation.

The software design reflects the need for these data-structures in two different settings, namely, (1) mesh generation and (2) numerical computation. Because, each of the data structures which are used to define the mesh are offered in two flavors.

For mesh generation, we prefer to have dynamic memory in the sense that inserting mesh primitives (e.g. nodes, edges, elements, etc.) should be allowed for. The user should not have to specify the number of mesh primitives a'priori; for an unstructured mesh, this is usually determined as a result of the mesh generation process. In this flavor of the data-structures and modules, we opt for the use of linked-lists which allow for easy and quick insertion of additional primitives. In the mesh generation process, generating and querying mesh information are the only tasks. A linked-list is not the most efficient structure for querying mesh information, but given that this is only one of two main tasks we prefer to give up efficiency in exchange for flexibility.

In conducting numerical computations with SEM, querying mesh information is a small part of the larger computations that is executed very often. Minimizing the time spent in obtaining mesh information is therefore crucial. In the numerical computation implementation of the geometry modules, we opt for arrays with fixed sizes, since arrays offer a much faster means of obtaining information. By the time the computation is being run, it is assumed that the mesh generation is complete, the number of mesh primitives is known and connectivity has already been established.

The benefit of having these two flavors also allows for users to make use of either package in developing their own software. For those who work with a moving or adaptive mesh, the mesh generation implementation is likely the preferred structure to work with, given the ease in introducing new elements. The exchange in efficiency in the query process is made up for by a gain in accuracy in the approximate solution.

In this way, the software structure offers flexible solutions to problems with complex geometry.

2 The Mesh

A mesh can be decomposed into the basic constituents :

1. Nodes
2. Edges
3. Faces (3-D only)
4. Elements (can be 2-D or 3-D).

Accompanying these constituents is the *connectivity*. The connectivity describes what nodes make up each edge, what edges make up each face (in 3-D) or element (in 2-D), and establishes "neighbor" relationships among the elements. This chapter describes the data structures that are used to define the basic constituents of a mesh and outlines a set of accompanying procedures which are used to establish connectivity and handle file I/O of the spectral element mesh.

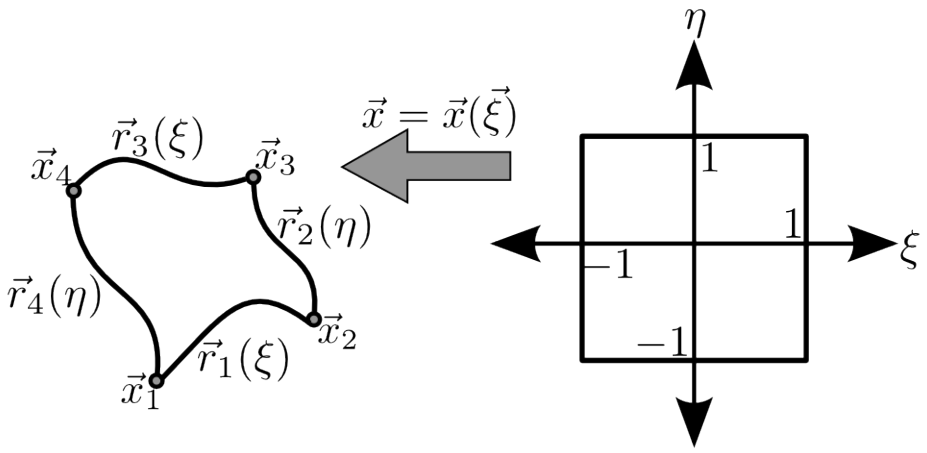

Figure 2.1: An element in two-dimensions is illustrated which is described by four corner nodes, and its internal geometry is defined through the parametric curves which make up the edges and the use of transfinite interpolation.

Figure 2.1 depicts an element in two-dimensions which is described by its corner nodes and its internal geometry. The internal geometry, in this software, is generated using *transfinite interpolation*.

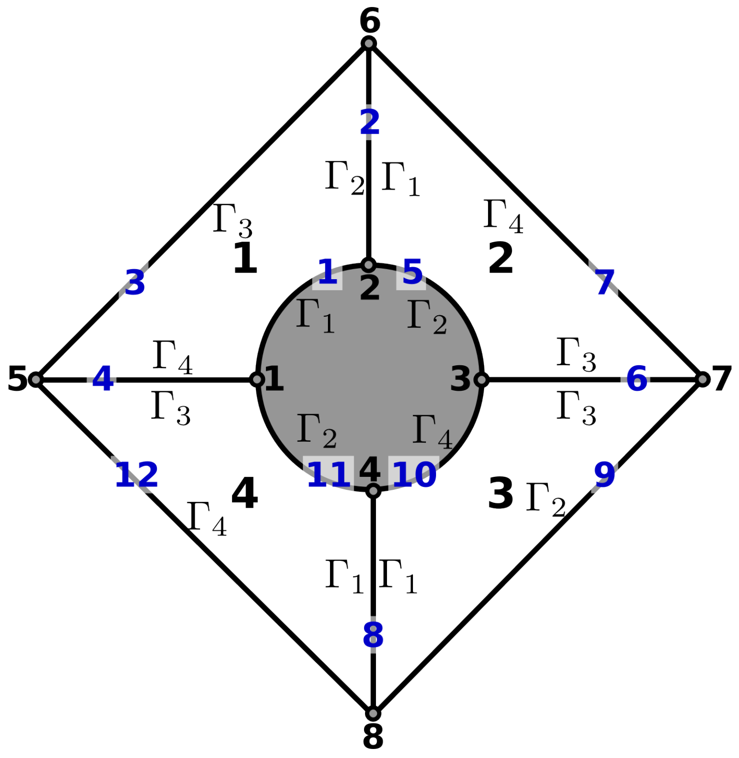

Figure 2.2: A simple mesh is shown which is composed of nodes, edges, and elements.

2.1 Data Structures and Routines

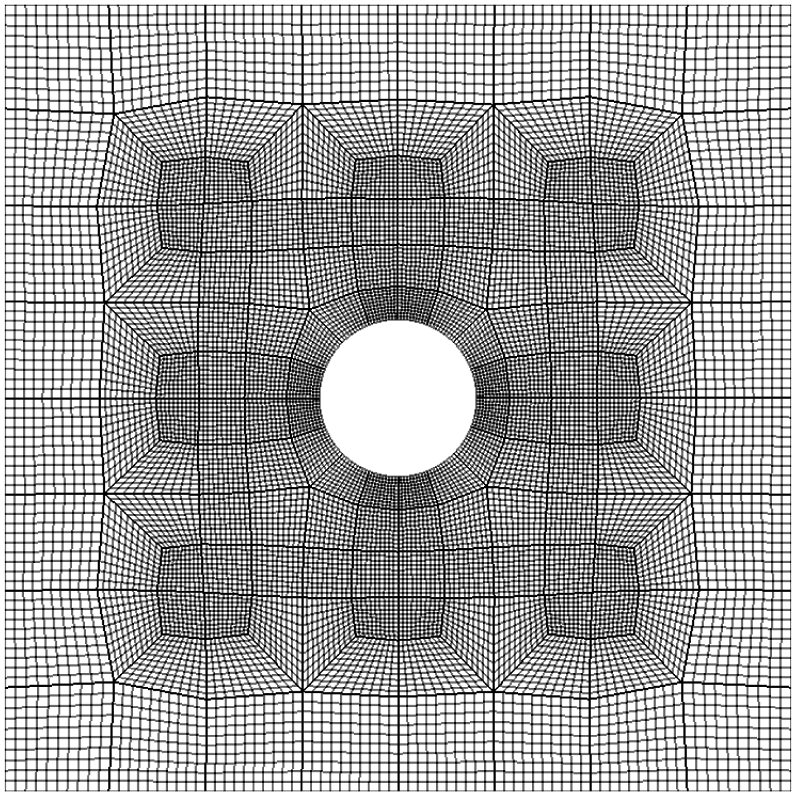

Figure 2.3: A sample mesh is shown where each of the elements is outlined by the darker lines. The internal mesh is also shown with lighter colored lines. This particular mesh is generated with the SpecMesh software (D.A. Kopriva, 2014)

Bibliography

D.A. Kopriva. *SpecMesh User Manual.* Department of Mathematics, The Florida State University, 2014.