LA-UR- 00-2996

Approved for public release; distribution is unlimited.

Title: On 3D, Automated, Self-Contained Grid Generation Within the RAGE CAMR Hydrocode.

Author(s): William R. Oakes
Paul J. Henning
Michael L. Gittings
Robert P. Weaver

Submitted to: 7th International Conference on Numerical Grid Generation in Computational Field Simulations

Los Alamos
NATIONAL LABORATORY

DISCLAIMER

GENERAL DISCLAIMER

This document may have problems that one or more of the following disclaimer statements refer to:

- This document has been reproduced from the best copy furnished by the sponsoring agency. It is being released in the interest of making available as much information as possible.

- This document may contain data which exceeds the sheet parameters. It was furnished in this condition by the sponsoring agency and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

On 3D, Automated, Self-Contained Grid Generation Within the RAGE CAMR Hydrocode

William R. Oakes[1]
Paul J. Henning[1]
Michael L. Gittings[1,2]
Robert P. Weaver[1]

[1]Applied Physics Division
Los Alamos National Laboratory
Los Alamos, NM 87545 USA
oakes@lanl.gov

[2]Science Applications International Corporation
La Jolla, CA 92121 USA
gittings@lanl.gov

Abstract

We discuss using the inherent grid manipulation capability within a Continuously Adaptive Mesh Refinement hydrodynamics code, RAGE, to implement parallel, automated, self-contained grid generation. We show how arbitrarily complex 3D geometries specified in any unambiguous form can be used.

The RAGE computational environment is any of several massively parallel computers being developed under the Department Of Energy's Accelerated Strategic Computing Initiative. A typical 3D RAGE analysis may contain 100 million cells and occupy 2000 processors for several weeks.

RAGE grid generation is embarrassingly parallel. The RAGE computational grid is an octree decomposition of the model space. The problem domain is subdivided into as many subdomains as the number of processors assigned to the problem. The grid for each subdomain is then generated independently, except for occasional adjustments.

Geometry used for initial grid generation includes CSG combinations of NURBS-based boundary representation models, stereo lithography (STL) files, implicit surfaces, and functionally perturbed surfaces.

Introduction

RAGE is a Continuously Adaptive Mesh Refinement (CAMR) multimaterial, Eulerian, radiation hydrodynamics simulation code based on a high-order Godunov method [1]. It is being developed by Science Applications International and Los Alamos National Laboratory, within the Department Of Energy's Accelerated Strategic Computing Initiative (ASCI) program. RAGE is one of several codes under development to explore computational and simulation techniques that might be used with the ASCI program's ultra-high performace computer systems; a 100 teraops system is planned for the year 2004.

Currently, the ASCI computing environment is comprised of three distinct high performance systems, with installations at Los Alamos National Laboratory, Sandia National Laboratories and Lawrence Livermore National Laboratory. Each system has a different architecture and software development environment. The common characteristics of these systems are that they are each massively parallel MIMD systems, and that they each support a message passing paradigm. Blue Mountain, the Los Alamos system, is a Silicon Graphics Origin 2000 system comprising 48 symmetric multiple processor (SMP) "boxes", each with 128 processors. Blue Mountain is capable of running simulations that use all of these processors concurrently, and has demonstrated a benchmarked performance of 2.38 teraops. The RAGE architecture is sufficiently flexible to fully utilize each of the ASCI computing systems in performing the chosen class of hydrodynamics simulations [2]. This flexibility will be essential as the ASCI computing environment continues to migrate towards the eventual goal of 100 teraop systems. Massive storage and visualization capabilities are also a must; recent RAGE calculations have produced tens of terabytes of data.

In this paper we discuss using the inherent grid manipulation capability within RAGE to implement parallel, automated, self contained grid generation. We will show how, and under what conditions, arbitrarily complex 3D geometries specified in any unambiguous form can be used. We will also show example grids.

The RAGE Computational Grid

The RAGE computational grid is an octree decomposition of the model space. The tree depth is selected to resolve field variable gradients to RAGE-specified resolution and material boundaries to user-specified resolution. The user specifies two resolution values for each material: the cell size at the boundary of the material and the cell size in the interior of the material. Because RAGE allows a mixture of materials in any cell, it is not necessary to subdivide the cells to the

resolution and material boundaries to user-specified resolution. The user specifies two resolution values for each material: the cell size at the boundary of the material and the cell size in the interior of the material. Because RAGE allows a mixture of materials in any cell, it is not necessary to subdivide the cells to the size of the smallest geometric features. An approximation of the material ratios within a leaf cell of the octree is computed using additional queries. In particular, the setup variable "numfine" specifies how many query positions in each dimension will be performed to approximate fractional composition of each leaf. For example, if numfine = n, and the problem is three dimensional, then each leaf cell will be queried n^3 times to determine mass fractions.

Tree construction begins by creating a uniform Cartesian (cubical) "base" (or "level 1") grid across the region of interest. Then, for each cell of the base grid that does not yet conform to the required resolution, a recursive process of octree creation is initiated. The "mother" cell is subdivided into eight "daughter" cube cells, each with an edge length of 1/2 of the mother cell edge length. We refer to the depth of the subdivision required to attain the requested resolution as the "level" of the grid. The 2:1 edge length ratio of mother to daughter cells is enforced as the maximum for any adjacent cells, whether or not they have a common ancestor.

Assuring adjacent cell edge length ratios is the first of our grid generation tasks requiring interprocessor communication during multiprocessor execution. To simplify indexing and grid density relaxation during program execution, we keep mother cells in the data structure. The cost of keeping these cells is, at most, a 12.5% reduction in memory allocation efficiency for cell storage. Once the initial computational grid is established, the geometric data set describing the initial geometry will not be queried again during the simulation, and so is released from memory.

Continuously Adaptive Mesh Refinement and Geometry Queries

Because CAMR requires repeated grid refinement and grid density relaxation as analysis proceeds, RAGE contains a flexible geometry and field variable query mechanism that supplies all required grid manipulation information. The only geometric query that RAGE requires is a point containment test, which determines if a specified region contains a query point. At the highest level inside the RAGE code, each geometric region is assigned a priority value. When the grid generation algorithm queries to determine which material region contains a query point, each region is tested in order from high to low priority. The first region for which the

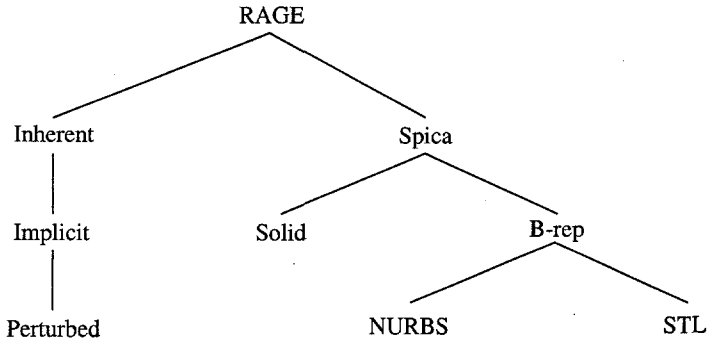

Figure 1: The hierarchy of RAGE geometry support.

point containment test returns "inside" is returned as the containing region.

The problem domain is subdivided into as many subdomains as the number of processors assigned to the problem. The grid for each subdomain is then generated independently, except for two adjustments that require occasional interprocessor communication: 1) load balancing is performed by redistributing problem subdomains as the number of cells increases, and 2) edge length information for adjacent cells in different subdomains is communicated to ensure the 2:1 edge length ratio.

Geometric Modeling

RAGE evolved from a 1D code to its present ability to perform 1D, 2D, or 3D simulations. In this paper we will discuss only 3D simulation and the related 3D geometric modeling. RAGE 3D modeling and simulation is performed in a Cartesian coordinate system.

Since RAGE only requires a point containment predicate, a wide variety of geometric modeling techniques are feasible. Geometric support for RAGE is currently divided among "inherent" support directly in the code and support in a stand-alone query library referred to as *Spica*. The entire geometric support hierarchy is shown in Figure 1. The internal structure of RAGE has proven to be flexible enough that any geometric query system could be added, provided that it supports two features: a point containment query and a means of distributing its data structures for parallel execution.

The inherent geometric support in RAGE includes spherical, ellipsoidal, conical

frustum, triangular prism and quadrilateral prism regions, as well as perturbed boundary regions (sine and cosine perturbations), and radially swept 2D regions. Use of the inherent geometric modeling involves describing a physical problem within the RAGE input file and then verifying through RAGE execution that the desired model has been achieved. These geometric primitives combined with judicious application of the RAGE region priority ordering scheme allows clever RAGE users to perform a surprising number of valuable simulations. However, these geometry models are often difficult to create, convoluted, and hard to understand.

To address these problems, the *Spica* library allows direct input of data from several geometric modeling packages, ranging from proprietary to commercial CAD packages. The *Spica* library tests point containment using an expression tree. The leaf nodes of this tree are geometry descriptions, and the interior nodes consist of region selection operators ($<$, $\leq$, $=$, $\geq$, $>$), logical operators (AND, OR, NOT), and special nodes (LINK, TRANSFORM).

A geometry description node can be a solid representation, a closed, manifold, boundary representation (b-rep), or an implicit surface that constitutes part of a closed boundary. In order for a particular geometry description to be compatible with *Spica*, it must have a point categorization query. For solids and b-reps, this query returns -1 if the point is inside the region, 0 if the point is on the region boundary, and 1 if the point is outside of the region. For implicit surfaces, "below" is equivalent to "inside."

Associated with each geometry description node is a region selection operator. This is simply a map

$$\{-1, 0, 1\} \mapsto \{t, f\} \tag{1}$$

For example, the $<$ operator returns "true" if a query point is in the open interior of the associated geometry description. Separating the region selection from the geometry description is useful because it moves the exact definition of "inside" or "outside" to the tree, rather than leaving it up to the geometry generation package. Spanning the results of the region selection nodes are the logical operators. These are used to form boolean expressions. The caller of the library is returned the boolean value of the root node for each query point.

For efficiency purposes, there are several special nodes in the tree. The first is a LINK node. These serve as a placeholder or a pointer to either another expression tree or to a geometry node, and are used to prevent duplication of data. The TRANSFORM node contains an affine transformation that is applied to the query point for the subtree below the TRANSFORM. Combined with a LINK node, this

allows geometry to be copied to other positions and orientations with minimal storage overhead. As always, time/space trade-offs are a concern: it is more efficient to duplicate a sphere geometry description node than to use transforms and links.

As described above, the geometry description nodes can be b-reps, solids, or implicit surfaces. Under the category of b-rep geometry, the *Spica* library currently supports NURBS-based representations and triangulated surfaces. Support for the NURBS representation, common in CAD models, is provided by Spatial's ACIS modeling kernel. As most of the CAD models we use are produced in Parametric Technology Corporation's Pro/ENGINEER package, we translate from the Pro/ENGINEER representation to the ACIS representation. Ensuring an accurate translation is one of the most difficult steps in this process.

The query support for triangulated b-reps is supplied by one of two algorithms. The first is a ray-shooting parity test: if a ray passes through a closed, 2-manifold boundary an odd number of times, the origin of the ray is inside of the region delineated by the boundary [3]. The special cases that arise in this approach are handled simply by shooting a new ray in a random direction. We have found this brute-force approach to handling degenercies to be highly efficient: the number of rays that have to be re-cast is typically very small.

The second query algorithm finds a surface point closest to the query point, and compares the normal at the surface to the direction of the ray. While slower than the ray-shooting approach, the closest point query has no special cases and is more robust in the face of small cracks in the surface [4]. *Spica* currently accepts stereo lithography (STL) files as input, simply because many packages write that format.

The solid geometry description nodes contain a variety of primitive objects: box, cone, cylinder, sphere, parallelepiped, torus, and a surface of revolution generated by a piecewise-linear curve. These primitives are composed into parts using a geometrical modeling tool called *OSO*. While it can be challenging to create extremely detailed models using these primitives, there is a considerable query speed advantage to doing so. *OSO* also supports STL files as first class primitives that can be translated, rotated, scaled, and combined with the previously specified primitives to compose parts. In practice, we find that combinations of all the geometry description node types are used in computational models.

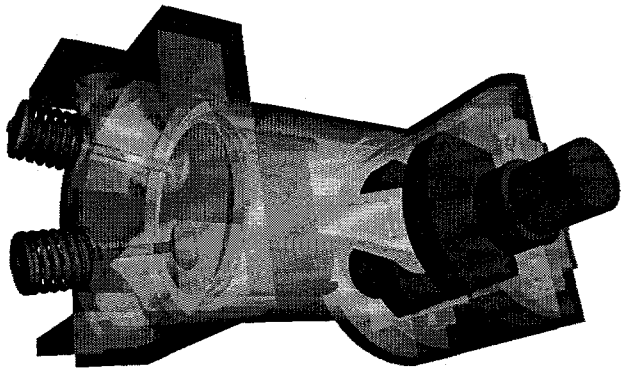

Figure 2: *OSO* rendering of the engine model.

An Example Grid

Figure 2 is an internal combustion engine model created in Pro/ENGINEER, output as separate STL files for each part in the assembly, and collected for viewing, scaling, positioning, and rendering by *OSO*. The *OSO* dataset was then preprocessed by RAGE to create an initial RAGE input file. Minimal adjustments are then performed on the initial RAGE input file to suit the analyst's criteria.

Figure 3 presents a close-up of a cross section of a RAGE grid generated from the Pro/ENGINEER model. Notice that the boundaries of the cylinder block, head, valve springs, and retainers are modeled at level 6, while the valve boundaries have been modeled at level 7. Also notice the gradual transition from level 1 to level 7 caused by the enforcement of 2:1 edge length ratios between adjacent cells. This grid has been restricted to seven levels for purposes of illustration. Typically, a well resolved computational grid must descend to level 10. Interestingly, for a highly detailed 3D model of a very complex mechanism, our current computational environment often restricts us to a level 8 grid, due to memory and CPU constraints.

Geometry Query Speed

In practice, there is another requirement for adapting geometry queries to RAGE grid generation. Because a typical 3D RAGE grid may contain 100 million cells and require 20 billion queries to create the cells and define their mass ratios, fast query speeds are essential. The table in Figure 4 presents typical query rates per

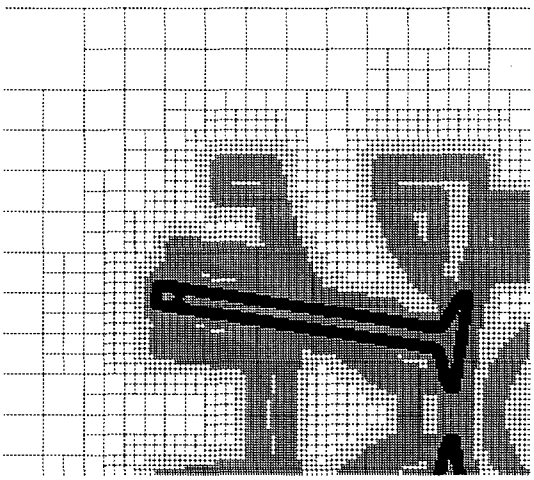

Figure 3: Magnified cross section of the RAGE engine model.

processor measured during RAGE execution. We find that intrinsic and *Spica* solid queries are sufficiently fast to never cause concern during grid generation and that the *Spica* triangulated b-rep query is sufficiently fast that our largest problems can be generated, in parallel, in two to five hours. However, the NURBS b-rep queries are sufficiently time consuming that they can not be used routinely at this time.

As is generally the case with geometric queries, timings are related to to the complexity of the model being queried. For the intrinsic and solid geometry types, query times are extremely fast because of the small number of primitives involved, and the ease of performing point containment queries on those primitives. For the triangulated surfaces, tree-based spatial search structures are used to quickly identify facets of interest [4], but there is a dependency on the total number of facets in the model. A NURBS b-rep often has much fewer elements than a triangulated

Representation	*Queries/second*
Solid or Inherent	200,000
Triangulated	1,500
NURBS	20–500

Figure 4: Expected query speeds for representative models.

b-rep, but the computational complexity of ray intersection or point projection algorithms dominates the query time.

Future Directions

In terms of geometry, we are currently looking at means of striking a compromise between modeling accuracy and query speeds. While NURBS-based CAD models provide the best accuracy, they are also very expensive to query for point containment. In comparison, triangulated surfaces are fairly inexpensive to query, but can introduce artifacts into the hydrodynamics solution. An attractive-looking compromise is the use of implicit surface patches for the b-rep [5]. These would allow a good match to the CAD model surface while providing efficient query capabilities.

We are also studying ways to improve our parallel query performance. Our primary computational platform is a cluster of multiprocessor machines, where each machine contains a large amount of shared memory. However, our current parallel implementation uses only a message passing paradigm. Taking advantage of the system topology should reduce our memory requirements and reduce the communication overhead, making higher-resolution grids feasible.

References

[1] R.M. Baltrusaitis, M.L. Gittings, R.P. Weaver, R.F. Benjamin, and J.M. Budzinski. Simulation of shock-generated instabilities. *Physics of Fluids*, 8(9):2471–2483, September 1996.

[2] R.P. Weaver, M.L. Gittings, M.R. Clover, and H.P. Pritchard. The parallel implementation of RAGE: a 3-D continuous adaptive mesh refinement shock code. ISSW22, July 18–23, 1999.

[3] J. O'Rouke. *Computational Geometry in C*. Cambridge University Press. 1993.

[4] A. Khamayseh and P. Henning. Point inclusion methods for boundary representation models. Technical Report LAUR-00-922, Los Alamos National Laboratory, 2000.

[5] C. Bangert and H. Prautzsch. Quadric splines. *Computer Aided Geometric Design*, 16:497–515, 1999.