FIXED GRID MESHING IMPLEMENTATION FOR INTERACTIVE ANALYSIS

Juan Fernando Duque Lombana

EAFIT UNIVERSITY
Engineering School
Mechanical Engineering Department
MEDELLIN

2012

FIXED GRID MESHING IMPLEMENTATION FOR INTERACTIVE ANALYSIS

Graduation project for the degree of Magister in Science of Engineering

Advisor
Prof. Manuel Julio Garcia

EAFIT UNIVERSITY
Engineering School
Mechanical Engineering Department
MEDELLIN
2012

Contents

List of Tables

List of Figures

Acknowledgments

"A wizard is never late, Frodo Baggins. Nor is he early. He arrives precisely when he means to."

-Lord of the Rings: The Fellowship of the Ring (Movie) by Peter Jackson, Fran Walsh, Philippa Boyens

Once again to my parents, Luis and Silvia, for all their invaluable love and support, this is for you and because of you.

There were many people involved on the development of this project and I would like to thank them: my advisor, Prof. Manuel J. Garcia, for all his support, uninterested help and understanding; This work is also his vision. Eng. Miguel Henao, for his solid work and appreciations that served as basis for this study. Eng. Santiago Giraldo, Msc. Santiago Orrego and Eng. Jorge Mario Mazo, colleagues and true wingmen, working by your side was always a pleasure.

Special thanks to COLCIENCIAS and EAFIT University, sponsors and believers of the the idea that knowledge generates wealth al welfare.

Summary

This graduation project presents a state of the art Fixed Grid Meshing algorithm and implementation suited for structural and fluid flow problems, where fast meshing calculation rates is required for large domains with high element density. Once the geometrical and computational problem is addressed, the architecture for an interactive CFD system suited for Grid Computing is presented to the reader. In both scenarios the state of the art of this kind of implementations is reviewed, followed by the introduction to the newly developed approaches for both problems and finally discussing the advantages and difficulties presented in the respective algorithms/methodologies and computational implementations.

The developments described within this work are all framed by a larger project called The Virtual Wind Tunnel sponsored by EAFIT University, Los Andes University, University of Alberta and COLCIENCIAS during 2008 and 2009; The results in this graduation work were developed entirely at the EAFIT University's Applied Mechanics Laboratory in Medellin and are part of a collaboration effort in companionship with the University of Alberta in Canada and Los Andes University in Bogota, Colombia.

Chapter 1

Introduction

The developments described within this work are all framed by a larger project called The Virtual Wind Tunnel sponsored by EAFIT University and COLCIENCIAS during 2008 and 2009; The present work is the compendium of two papers which portray some of the most relevant features and functionalities developed with the author's leading role during this project.

1.1 Fixed Grid preprocessor

This work presents an overview of the current state of Paravoxel, a parallel-Cartesian Fixed Grid Mesh and Preprocessor developed in the Applied Mechanics research group which is suitable for structural and fluid dynamics computational applications. The application contains an embedded fixed grid preprocessor and a third party surface-meshing algorithm adapted for geometry acquisition.

The basics of the Fixed Grid Meshing algorithm will be discussed within section , followed by some domain decomposition modifications that make this preprocessors approach different from others. The possibility to decompose the domain - and the data structures required for that operation - allow the algorithm to be parallelizable. This fact leads to a discussion about the advantages and difficulties of the current implementation. The reader is presented with a set of examples and a brief discussion on the possibility of applying this algorithm together with other meshing approaches.

1.2 Fixed Grid and CFD

Computer Aided Engineering (CAE) has gained great relevance over the past few decades given the advantages it has in terms of ease of using for expert users, low costs and where long experimentations times become the bottleneck for situations where complex multi-physics problems are approached. The present work intends to present new approaches which lower the required times for user interaction with the analysis system and the computational times, elevating other indicators such as the time an user can spend studying the physics behind the results presented by the CAE tool.

Computational Fluid Dynamics (*CFD*) represent growing field nowdays, and the times spent during the preprocessing phase of a single problem analysis largely depend on the complexity of the geometry of the body and the fluid flow conditions.

Paravoxel as a preprocessor alone does not fulfills all the requirements for its direct usage in Fluid Flow simulations. A proper strategy to link it with the *CFD* Analysis is presented in the context of steering CFD simulations to solve aerodynamic design problems.

1.3 Computational Steering of CFD Simulations

1.3.1 Wind tunnels

Since the building of the first wind tunnel by Francis H. Wenham in Great Britain in 1871, the study of fluid flow problems over complex geometries has developed aiming to characterize both transient and steady phenomenon on fluid dynamics. A wind tunnel consist of carefully designed ducts through which a stream of air is driven at controlled velocities and uniform conditions [4]. Wind tunnels are used by engineers and scientists to simulate air-flow conditions in the laboratory. Analyzing scale models is cheaper, take less time to build, and generally allows more extensive instrumentation than full-scale testing which sometimes is even impossible. [4].

To guarantee that the results of scale tests are directly proportional to full-scale models, dynamic similarity must be preserved: Mach and Reynolds numbers are to be the same as those present in full scale conditions. Mach number is the ratio of the air velocity to the velocity of sound [23] and the Reynolds number reflects the ratio of linear momentum to viscous forces [23] allowing a proper prediction of the present drag forces. In some cases guaranteeing the dynamic similarity conditions becomes a monetary-expensive task due to the need of highly accurate and reliable sources to guarantee the conditions of the fluid flow.

During the past century, the development of numerical methods and computers have opened the doors for Physicists, mathematicians and consequently to engineers to approximately solve physical problems once thought impossible without the proper experimentation and mathematical tools, being the wind tunnel a problem as such.

1.3.2 Computational Approach

Within the following chapters of this work an overview of the architecture used for the implementation of an Interactive Computational Fluid Dynamics (*CFD*) environment, intended for studies mainly related with shape optimization of the airflow around virtual prototypes. The idea behind this type of implementation is to constrain faster the design domain that a group of designers has to face when facing a task such as this. The aim of this work is to present the abstraction of the requirements for the architecture of a grid computing system intended for *CFD* simulations and a some of the milestones faced for such implementation.

The structure of an automated server responsible for managing the CFD module and updating the new simulation conditions for the next timestep is the core of the project and will be a matter of discussion.The used scheme for distributing data and information as soon as they are available for both solver machine (located on the same computer or on a High Performance Computing, *HPC*, facility) and clients from the server, will be discussed as well.

The presented architecture supports a *CFD* interactive solver (GNU/GPL CFD library) for incompressible laminar Navier-Stokes equations using the *PISO* algorithm. The application makes use of an embedded Fixed Grid Preprocessor (Paraview) described on Chapter 2 . The current implementation is inherently interactive allowing easy changes to the virtual prototype geometry and to the boundary conditions of a *CFD* simulation.

Chapter 2

ParaVoxel: a domain decomposition based Fixed Grid preprocessor

2.1 Summary

This paper is an overview of the current state of a parallel-Cartesian Fixed Grid Mesh and Preprocessor, which is suitable for structural and fluid dynamics computational applications. The application contains an embedded fixed grid preprocessor and a third party surface-meshing algorithm adapted for geometry acquisition. The first section of this paper is a presentation of the basics of the Fixed Grid Meshing algorithm, followed by some domain decomposition modifications that make this preprocessor's approach different from others. The possibility to decompose the domain - and the data structures required for that operation - allow the algorithm to be parallelizable. This fact leads us to a discussion about the bottlenecks of the current implementation. Finally, the reader is presented with a set of examples and a brief discussion on the possibility of applying this algorithm together with other approaches.

2.2 Introduction

Mesh generation is a process of spatial subdivision of a generally complex continuous domain into simple-shaped sub-volumes of pre-defined topology [8] . These sub-volumes are usually referred to as mesh elements or cells. Elements in a mesh are connected to each other, without intersecting, and they cover the entire domain, thus representing a discrete form of continuous domain.

Fixed grid implementations on most applications rely on cartesian meshing applications to generate meshes suited to assemble and solve the finite element problem. This process is based on topological data in order to achieve faster performance than other approaches, such as constrained Delaunay tetrahedrizations ([20]), or Structured Meshing. Fixed Grid (FG) meshing, Quadtree decompositions and Cartesian Grid Meshing algorithms are broadly used (Dunning et Al. [7],Wang [24] and Aftosmis [1] just to name a few), as an efficient link between CAD systems and Finite Element/Finite Volume solvers given their low computational cost. [11]

The following sections introduce the reader to the developments on Paravoxel, an MPI based parallel Fixed Grid Preprocessor, suited for structural and fluid dynamics problem solvers.

2.3 3D Domain-meshing algorithm - ParaVoxel

2.3.1 Basic Fixed Grid Meshing Algorithm

A basic FG meshing algorithm starts with the immersion of a body (Ω) inside confines domain (called Fixed Grid Domain Ω_{FG}) which will subsequently be subdivided into discrete elements (e_i, defined by its vertex x_i).These elements are topologically and geometrically conformed in an abstract manner before the immersion of Ω inside Ω_{FG}. After such immersion, each node belonging to Ω_{FG} n_i will be classified as IN if $n_i \in \Omega$, OUT if $n_i \notin \Omega$ or $BDRY$ if $n_i \in \partial\Omega$. After the node classification, an element classification process starts (having calculated the number of IN and OUT nodes belonging to e_i). During this process every e_i is classified as: IN if every node n_i belonging to it qualifies as IN , OUT if every node belonging to it qualifies as OUT and NIO if e_i contains a mixture of IN, OUT and $BDRY$ nodes.

2.3.2 ParaVoxel Fixed Grid Meshing Algorithm

In order to implement ParaVoxel, the two-dimensional fixed grid meshing algorithm presented by Garcia [11] was extended to three dimensions, and the number of operations to be computed was minimized. Algorithm 1 shows the implementation of Paravoxel prior to the application of any mesh adaption techniques.

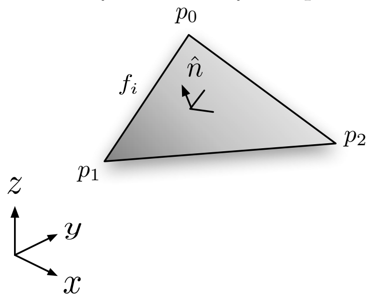
Algorithm 1 ParaVoxel FG Meshing Outline

```
**GIVEN:** a Triangularly faceted Solid BREP of a Solid Model $\partial\Omega = \{f_0, f_1, f_2, f_3 ... f_n\}$
**GIVEN:** a fixed grid Domain $\Omega_F G = \{e_0, e_1, e_2, e_3 ... e_n\}$
**GOAL:** Fixed grid representation of body $\Omega$
1.) *SolidModelLoad* {Routine to load the Solid Model BREP $\partial\Omega$}
2.) *FacetPerRayCompute* {Routine to Calculate which facets $f_i$ are hit by a ray belonging to $\partial\Omega$}
3.) *NodesPerRayClassify* {Routine to calculate and classify the nodes $n_i$ belonging to $\partial\Omega$, given their Node state}
4.) *ElementsClassify* {Routine to classify the elements belonging $\partial\Omega$, given the states of their nodes}
5.) *NioGeomCompute* {Routine to compute the $NIO$ elements geometry}
```

2.3.3 Boundary data acquisition

Different representations from computational geometry can be used to represent the geometry of the body Ω; for example: Parameterized Primitive Instancing,

Figure 2.1: A facet f_i defined by the points $\{p_0, p_1, p_2\}$



Constructive Solid Geometry (CSG), Cell Decomposition, Implicit Representations. For this implementation, we chose a representation scheme called Boundary Representation (BREP). It uses a simple representation of solid bodies by representing their boundary with a set of piecewise triangular faces (facets, f_i) defined by three points $\{p_0, p_1, p_2\}$, oriented in such way that right hand convention is preserved.

The process of acquiring the object boundary representation is performed via VRML 2 Files. VVRML is an international standard used for representing 3-dimensional objects using interactive vector graphics. The information contained in the VRML file is written as a Piecewise Linear mesh, composed of triangular entities (facets, f_i), representing the boundary of the 3D object in question.

Paravoxel is capable of handling the acquisition of parametric geometries via the Netgen library. Netgen is a third party software, programmed by Joachim Schberl [19], which allows the acquisition of geometry from any CAD system via a neutral file (IGES, STEP) or STL models. The main motivation for choosing this software as the data acquisition engine is its ability to perform quality triangular-based surface meshing in reasonable times. The boundary representation acquired via Netgen is meshed, generating a suitable $\partial\Omega$ and handed over to ParaVoxel to generate a discrete 3D domain.

The initial transition from parametric to discrete geometry meshing was partially responsible for some defeaturing, however this will not be discussed here due to space limitations. This it is one of the sources of defeaturing the geometry will suffer during the generation of the volume discretization.

2.3.4 Ray Classification based on facet intersections

On Algorithm 1, during step number 2, an instance of a procedure called *FacetPerRayCompute* is invoked. This instance is an optimization of the FG algorithm presented by Garcia [11]], using the previous calculations to define what facets $f_i \in \partial\Omega$ are touched by the ray r_i (See Figure 2.2 in order to reduce the number of future computations of the Fixed Grid meshing algorithm. r_i (See Figure 2.2)can be defined as an infinite straight line normal to a plane Π_0 which contains a set of collinear nodes belonging to Ω_{FG} [10] . A structured set of rays normal to a plane Π_0 contains all the nodes belonging to Ω_{FG}.

The algorithm proposed by Henao and Garcia in the present work to decompose the domain (Algorithm 2) consists of projecting all the facets that belong

Figure 2.2: A ray r_i normal to plane Π_0

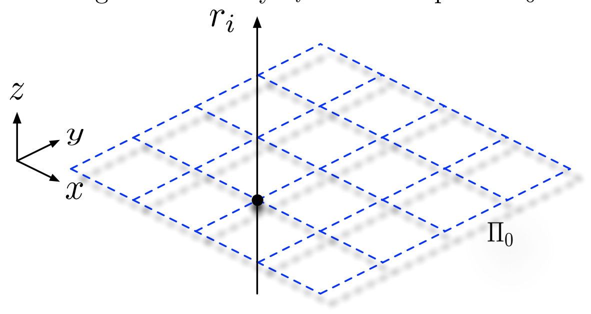
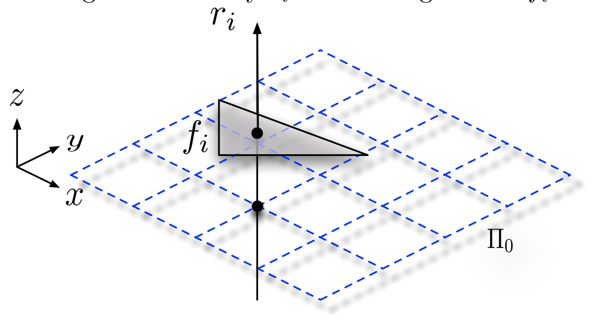


Figure 2.3: A ray r_i intersecting a facet f_i



to $\partial\Omega$ over a plane Π_0 (Where the rays r_i are originated in the fixed grid) and storing the index of the faces of interest per ray.

Given the nature of the domain decomposition operation a valid approximation for this operation is to calculate if the ray r_i pierces the bounding box of the projection of f_i over the plane Π_0. The algorithm, as implemented in ParaVoxel, is displayed in 2. In the first place, to calculate the intervals of the bounding box belonging to the f_m projection, a min-max calculation is used on the convex polygon or line resulting from the projection. In second place as f_m will always be normal to the plane Π_0, a check is performed to determine if the coordinates of the seed point of each ray are inside the intervals of the bounding box belonging to the projection. This can be ascertained by checking 2 coordinates inside the triad, without having to perform any transformation if Π_0 is normal to any of the of the unit vectors defining the world coordinate system (WCS).

After performing the classification of Facets per ray, the number of operations in future stages of the algorithm is greatly diminished, because of the fact that the Ray-Facet intersection operations are not performed on every f_i for each r_i. An advantage of this implementation is that being the operations of facet ray intersection domain independent, parallelization is a natural approach to follow in the implementation development and improvement stages; another substantial improvement over the conventional Fixed Grid approach is that for any ray that does not hit any particular facet, all the nodes belonging to that ray are classified as *OUT* directly.

Algorithm 2 *FacetPerRayCompute*

```
**GIVEN:** Given a Triangularly faceted Solid BREP $\partial\Omega = \{f_0, f_1, f_2, f_3 ... f_n\}$
**GIVEN:** Given a set of rays $r_i$
**GIVEN:** Given a plane $\Pi_0$ orthogonal to $r_i$
**GOAL:** An approximated classification of facets $f_j$ per ray $r_i$ inside vector $FacetsPerRay$
$N = size(\partial\Omega)$
$j = 0$
while $j < N$ do
  $f_m = Projection(f_j, \Pi_0)$
  $[I_x, I_y] = BoundingBox(f_m)$
  for ALL $r_i \in [I_x, I_y]$ do
    Store $j$ in $FacetsPerRay[i]$
  end for
end while
```

2.3.5 Ray-Facet intersection computation/Domain decomposition

To calculate the Ray-Facet intersection computation, a variant of the Möller and Trumbore algorithm [16] is implemented, in which the plane defined by the f_m contains r_i. The purpose of the modified algorithm is to calculate the segment defined between two points representing the intersection between f_i and r_i. If f_i is pierced by r_i on a single point, both points defining the segment will be the same. Algorithm 3 outlines the implementation inside ParaVoxel.

The advantage of this vector operations-based routine is that it does not require any pre-computed plane equations for the facets. Given the relative simplicity of the operations presented by the algorithm, expressions can be presented in terms of the original vertex of the triangular facet, the pivot point of the ray, and its direction. This operation can be performed using any symbolic algebra math package, and need not take up a lot of memory during its execution.

It is important for the reader to notice that the result of Algorithm 3 is always a pair of two geometrical points, no matter if they are the same. The previous condition should be verified in future stages of the FG algorithm to check if the intersection between the facet and the ray is a straight segment or a single location in space.

2.3.6 Node and Element Classification

NodesPerRayClassify procedure is required in Algorithm 1, step number 3. This step is aimed to classify every n_i node belonging to Ω_{FG} correctly. Basically, every time a ray hits the $\partial\Omega$ closed 2-manifold (in zones other than surface peaks or aligned surfaces containing the ray) the state of the nodes over that point of the boundary changes from IN to OUT and vice versa. Algorithm 4 outlines the node classification process as implemented in ParaVoxel.

It is important to highlight that a section of Algorithm 4verifies if the position of every node n_i is contained within the interval defined by two intersection points. If the intersection points define a line completely belonging to any facet

Algorithm 3 *RayTriIntersection*

```
GIVEN: a triangular facet defined as $f_i = \{p_0, p_1, p_2\}$
GIVEN: a ray $r_i$ defined by its direction $\hat{r_i}$ and pivot point $r_{i0}$
GOAL: to determine the intersection segment bounded by $p_{\cap 0}$ and $p_{\cap 1}$
$e_1 = \overrightarrow{p_0 p_1}$
$e_2 = \overrightarrow{p_2 p_1}$
$p_v = \hat{r_i} \times e_2$
$\lambda = \langle e_1, p_v \rangle$
if $\lambda \in [-\epsilon, \epsilon]$ then
  if $r_i$ lies on $f_m$ then
    $[p_{\cap 0}, p_{\cap 1}] = ComputeSegment()$
  else
    There is no intersection
  end if
end if
$t_v = \overrightarrow{r_{i0} p_0}$
$\beta = \langle t_v, p_v \rangle (\frac{1}{\lambda})$
if $\beta \in (-\epsilon, 1 + \epsilon)$ then
  There is no intersection
end if
$q_v = t_v \times e_1$
$\gamma = \langle r_i, q_v \rangle (\frac{1}{\lambda})$
if $(\gamma < -\epsilon) \cup (\beta + \gamma > 1 + \epsilon)$ then
  There is no intersection
end if
$p_{\cap 0} = r_{i0} + \langle e_2, q_v \rangle (\frac{1}{\lambda})$
$p_{\cap 1} = p_{\cap 0}$
```

of $\partial\Omega$, the state of the nodes is classified as *Boundary*, since for some Fixed Grid approximations the physical meaning of having a boundary node might prove critical while correctly classifying the state of the elements.

Algorithm 4 *NodesPerRayClassify*

```
GIVEN: an array of facet sets related to each ray, $FacetsPerRay$
GIVEN: a set of rays, $r$
GIVEN: a set of facets, $f$
GIVEN: a set of nodes, $n$
GOAL: an array of state flags of $n_i$ respect to $\Omega$, $NodeState$
for All rays $r_i$ do
  for All facet $f_m$ inside $FacetsPerRay[i]$ do
    $P_{r_i} = RayTriIntersection(r_i, f_m)$
  end for
end for
if $size(P_{r_i}) \neq 0$ then
  $FilterPeakPoints()$
  $WhereAmI = Outside$
  for All intersection points $p_k$ in $P_{r_i}$ do
    for All nodes $n_i$ inside $[p_k, p_{k+1}]$ do
      if $[p_k, p_{k+1}]$ is not an intersection segment contained by a facet then
        $NodeState[n_i] = WhereAmI$
        $WhereAmI = Not(WhereAmI)$
      end if
    end for
  end for
  for All nodes $n_i$ inside intersection segments $[p_k, p_{k+1}]$ do
    $NodeState[n_i] = Boundary$
  end for
end if
```

After node classification process, the next step (Algorithm1, step number 4) is to perform element classification. The set of rules to determine if an element is either IN,OUT or NIO is based on the number of $IN(Icont)$, OUT ($Ocont$) and *Boundary* nodes ($Bcont$) belonging to an element. Algorithm 5 contains the set of rules implemented in ParaVoxel. The set of rules defined here is an arbitrary convention derived from the study of the possible combinations of the state of the nodes belonging to an element. The basic idea behind the aforementioned conventions is to keep the geometry of the meshed body as close as possible to the one initially immersed in the Fixed Grid.

Only the state of the element is required to reconstruct the geometry of the IN elements, elements, because a pre-computed solution is set for it (hexahedrons in this case). NIO elements represent a completely different scenario, given that countless geometries with different topologies may appear in a single fixed-grid calculation.

Algorithm 5 *ElementsClassify*

```
GIVEN: an array of node connectivities per element, $e$
GIVEN: an array of state flags of $n_i$ respect to $\Omega$, $NodeState$
GOAL: an array of state flags of $e_i$ respect to $\Omega$, $ElementState$
for All element $e_i | e_i \in \Omega_{FG}$ do
  $[Icount] = evaluateINCount(e_i, NodeState)$
  $[Ocount] = evaluateOUTCount(e_i, NodeState)$
  $[Bcount] = evaluateBDRYCount(e_i, NodeState)$
  if $Icount > 0 \cap Ocount > 0 \cap Bcount > 0$ then
    $ElementState[e_i] = NIO$
  end if
  if $Icount > 0 \cap Ocount > 0 \cap Bcount = 0$ then
    $ElementState[e_i] = NIO$
  end if
  if $Icount = 0 \cap Ocount > 0 \cap Bcount > 0$ then
    $ElementState[e_i] = OUT$
  end if
  if $Icount = 0 \cap Ocount > 0 \cap Bcount = 0$ then
    $ElementState[e_i] = OUT$
  end if
  if $Icount = 0 \cap Ocount = 0 \cap Bcount > 0$ then
    $ElementState[e_i] = IN$
  end if
  if $Icount > 0 \cap Ocount = 0 \cap Bcount = 0$ then
    $ElementState[e_i] = IN$
  end if
  if $Icount > 0 \cap Ocount = 0 \cap Bcount > 0$ then
    $ElementState[e_i] = IN$
  end if
end for
```

Figure 2.4: Closest point lies inside the facet

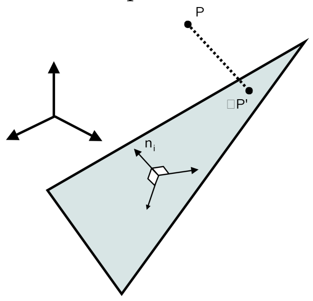
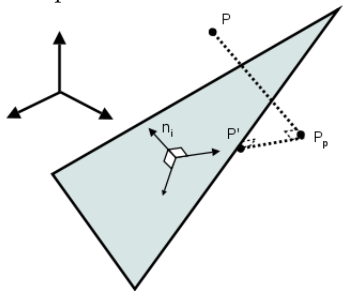


Figure 2.5: Closest point lies over the boundary of the facet



2.4 Meshing adaption approaches

Since ParaVoxel an hexahedral predominant fixed grid mesher, the geometry of the IN and OUT elements is fast and easily obtained and manipulable, however the NIO elements are still subject of debate; Given the fact that speed is the main goal for this preprocessor while maintaining stability, two approaches are used in the preprocessor to achieve meshes that closely resemble the original geometry immersed inside the Fixed Grid.

2.4.1 Mesh Smoothing, Vertex-Surface adaption

A completely hexahedral mesh is conformed with an IN, OUT and NIO element sorting scheme (using only the previously calculated IN elements). The resulting mesh represents an skewed version of the object the designer wishes to analyze.

The IN vertex belonging to both the IN and NIO elements of the Fixed Grid are moved from their original location P to a new position P' so that $min(P - P')$ is achieved, being P' is located over the boundary of Ω.

The algorithm finds the closest point on each facet F_i to the point P. Afterwards, it compares which one is the closest point across the whole domain. To correctly locate the $min(P - P')$ ffor a single facet, two general cases are solved: when normal projection P lies inside the facet (hence the projection of P is the closest distance, Figure 2.4) and when the normal projection of P lies outside the facet (thus the closest point to P is the closest point in the F_i boundary, Figure 2.5).The algorithm is presented in Algorithm 6.

Algorithm 6 combined with the fixed grid scheme present in ParaVoxel,

Algorithm 6 Vertex-Surface adaptation algorithm

```
**Given:** Point $P$ in space
**Given:** Triangularly faceted surface $\partial\Omega = \{f_0, f_1, f_2, f_3 ... f_n\}$
**Goal:** The closest point to $P$ oversurface $\partial\Omega$, $P'$
Calculate normal $n_0$ for facet $f_0$
Calculate $P_p$ such that $P_p$ is the orthogonal projection of $P$ into the plane defined by $n_0$
**if** $P_p$ lies inside $f_0$ **then**
  $P' = P_p$
**else**
  Calculate $P_{pp}$ such that $P_{pp}$ is the projection of $P_p$ into the boundary of $f_0$
  $P' = P_{pp}$
**end if**
**for all** $F_i$ different from $f_o$ **do**
  Calculate normal $n_i$ for facet $f_i$
  Calculate $P_p$ such that $P_p$ is the projection of $P$ into the plane defined by $n_i$
  **if** $P_p$ lies inside $f_i$ **then**
    $P_T = P_p$
  **else**
    Calculate $P_{pp}$ such that $P_{pp}$ is the closest point to $P_p$ into the boundary of $f_i$
    $P_T = P_{pp}$
  **end if**
  **if** $norm(P - P_T) < norm(P - P')$ **then**
    $P' = P_T$
  **end if**
**end for**
```

provides all the necessary conditions for easy parallelization via domain decomposition. The algorithm speed-up is determined by the Fixed Grid mesh, the solid bodys dimension, the number of processors, and the domain decomposition technique used (the algorithm is sped up since the number of operations is constant per facet projection). Domain decomposition via geometric subdivision is an attractive approach, using radial based functions to select the facets rooted on the original point location P tto select the facets that should be used to calculate the required P'.

During our testing, the algorithm behaves well if the element dimensions are small (with a ratio higher than 1/10 , approximately) in comparison to the object features; topology is easily respected due to the small displacements. However, for large displacements (especially over elements with large aspect ratios between their dimensions), ill-defined elements (self-intersecting, large non-orthogonalities, etc...) tend to appear easily, and untangling methods are required to be able to produce a valid mesh.

2.4.2 Polymesh generation using Convex Hull NIO Elements approach

Another geometry-fitting mesh generated via Fixed Grid can be calculated using polyhedral discrete elements. By means of the IN, OUT and NIO NIO element sorting process described previously, all IN and NIO elements are added to the polymesh, generating a non-conformal, all-convex, polyhedral mesh without hanging nodes. The advantages of this approach are that the geometry fits closely the original body, and the shape of the hexahedral elements is regular across most of the domain.

A simple Convex Hull approach was used to compute the NIO elements, in order to approximate their geometry [12]. A Convex Hull can be seen as the boundary of the minimal convex set that contains a finite set of points (Figure 2.6),); thus, if a NIO element is small enough so that no large curvature changes appear throughout the surface of any of its faces, the convex hull of the set of points composed by the IN and *Boundary* vertexes belonging to it should present a decent approximation to its original geometry for meshing purposes. The *qhull* library (Developed by Barber et al. [2]) is used in ParaVoxel.

It is important to notice that using a Convex Hull approach to approximate the NIO elements geometry leads to some defeaturing of the model, which is a problem when really exact calculations are required. However, for early design stages where fast-interactive testing is performed to shorten the design domain, this type of Fixed Grid approximation is suitable and desirable.

The balance between the size of the Fixed Grid and the smallest feature of the solid body will be the decisive factor to guarantee a proper representation of the boundary with the convex-hull NIO elements. An analogy to the Nyquist-Shannon Theorem [21] [22] for analog signal sampling is evident during this process: *to achieve a close approximation via convex-hull NIO elemented fixed grid, the largest size of an element in the grid must be at least half the size of the smallest feature of the solid body.*

Figure 2.6: *NIO* element calculated via Convex Hull

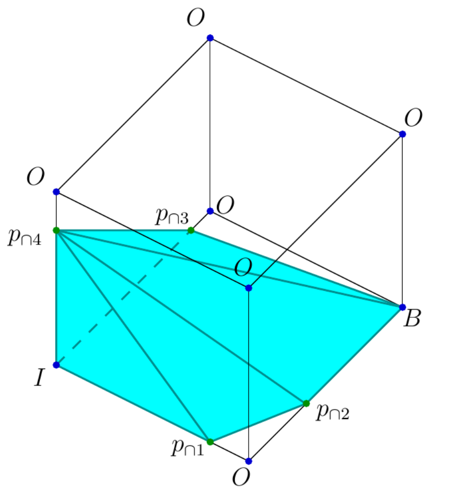
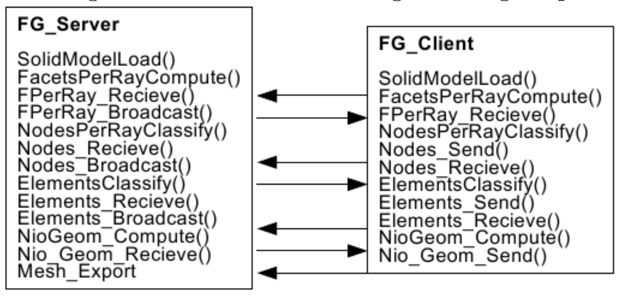


Figure 2.7: Paravoxel Message Passing Map



2.5 ParaVoxel Parallel approach

For the implementation of ParaVoxel, a distributed memory scheme via MPI (Message passing interface) using a Server-Client approach was selected. The Server will be the process in charge of sharing/sending/receiving the data needed for computations performed by each Client. The solid model geometry is loaded on Client and Server side, tasks and operations are divided via domain decomposition, based on the rays and bands of elements assigned to each client.

For this kind of computation, one of the bottlenecks is clearly the time it takes to pass messages [12]. Thus, communication requests are minimized, leading to a scheme where every process makes the same amount of operations independently; thus maximizing the quality of information sent per package becomes the problem of interest. Figure 2.7 shows the sequence for the main meshing algorithm running in the server and the client machines, and how the communication process takes place among them.

After all the steps in the main algorithm are completed, the information is sent back and organized in the Server. Due to the fact that the resulting mesh is structured, only the state of the elements has to be known in order to recover

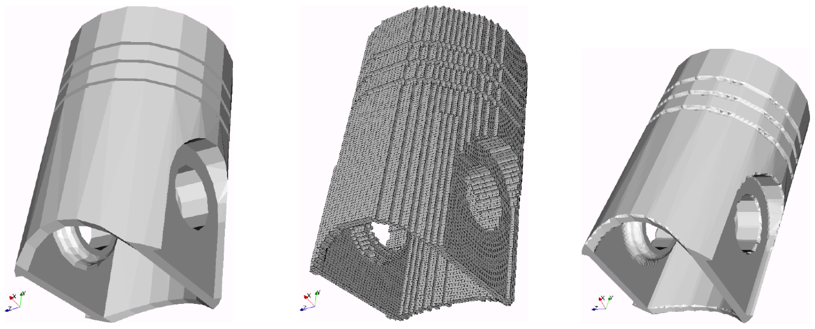
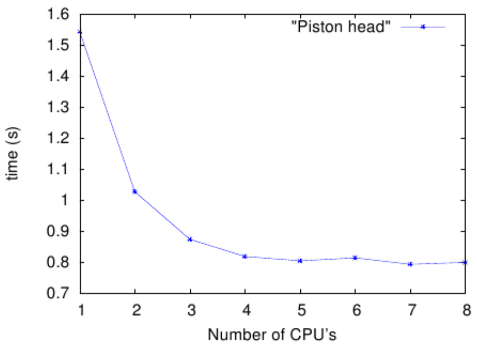

(a) Piston Head B-Rep (b) Resulting IN elements (c) Resulting NIO elements

Figure 2.8: ParaVoxel test case Piston Head 100x100x200 Grid

Figure 2.9: Performance of the FG Mesher processing the piston head model

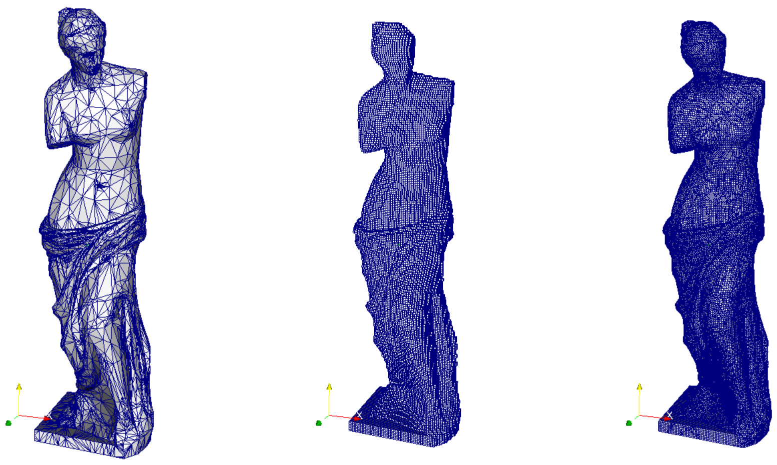


the I elements. For the NIO elements, the information about the intersection points has to be shared among all the processes. Thus, the fewer NIO elements in the model, the less data there is that has to be shared between the server and the clients. The next section of this paper shows the results obtained in a test that was performed with the meshing application.

2.6 Meshing test cases

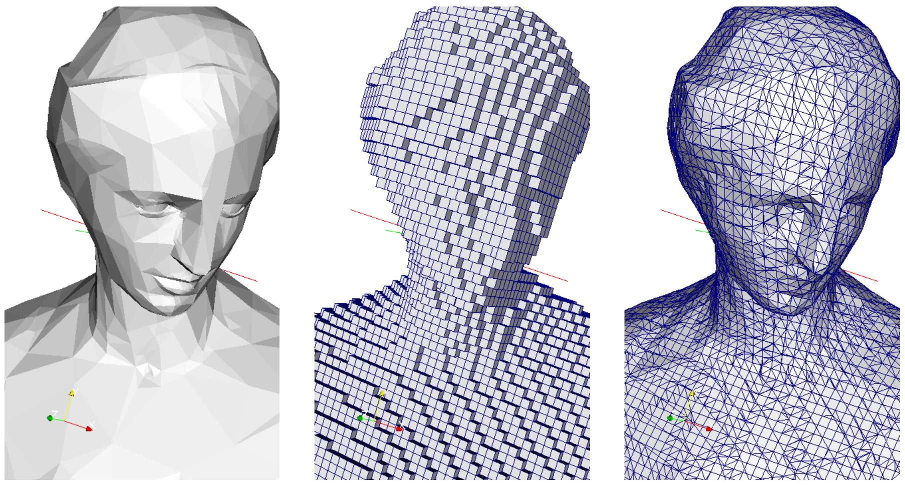
Figure 2.8 shows the complete Paravoxel meshing results, in which given a valid geometry (B-Rep of a piston head), a mesh composed by the IN and NIO elements is calculated. Figure 2.9 presents the scalability of the algorithm running on an eight-CPU cluster (running on a Linux distribution) allocated to mesh the piston head geometry.

Figure 2.10 and 2.11 show the results of meshing a complex model of a Venus statuette using a 66X230x66 element grid. The original BREP model was obtained from the INRIA 3D meshes research database, and it is owned by Artist 3D(`www.artist-3d.com`). The model was used for academic research purposes only.



(a) Venus B-Rep (b) Resulting IN elements (c) Resulting NIO elements

Figure 2.10: ParaVoxel test case



(a) Venus Head B-Rep (b) Resulting IN elements (c) Resulting NIO elements

Figure 2.11: ParaVoxel test case

Test Case	Grid dimensions	Grid elements	Facet-Ray clasification (s)	Node clasification (s)	Element clasification (s)	NIO elements
Mesh 0	{31x20x46}	22320	3.1739	1.1538	0.0055	5890
Mesh 1	{62x41x71}	180482	3.8692	1.4828	0.0367	24469
Mesh 2	{123x83x142}	1449678	4.6319	1.8153	0.149	62759
Mesh 3	{246x166x284}	11597424	6.5821	3.8537	1.2076	253484
Mesh 4	{482x324x568}	88703424	10.433	13.091	8.8917	988525

Table 2.1: Paravoxel Kraft test results

Another complex model was selected to perform tests related to the scalability of the single core meshing algorithm and its computation times(Figure 2.12). The model was meshed at a starting coarse resolution (28520 elements, Mesh 0) and the resolution was subsequently divided by half in all directions (escalating the computations in a $O(2^n)$ order) until a 88'703.424-element grid was achieved (Mesh 4). In order to test the portability of the software, the computer used for such tests was a MacBook Pro -running on Snow Leopard OSX- with an Intel Core 2 Duo 2.8 GHz processor, and 4GB of RAM. The original BREP model was obtained from the INRIA 3D model library, and it is owned by Polygon Technology GmbH (`www.polygon-technology.com`. The model was used for academic research/ purposes only.

Figure 2.12 presents the original BREP model and the resulting meshes from ParaVoxel. Table 2.1 shows relevant data related to the results of each meshing scenario.

The results for both performance and quality of the mesh suit the needs of the Fixed Grid implementations, including the ability to return geometrical and topological information of the elements at all times.

2.7 Domain and Boundary conditions integration

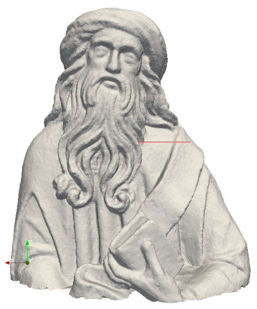
2.7.1 Fixed Grid based solution (Structural problems)

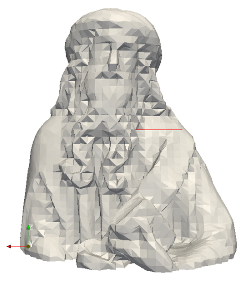
A strategy in which the boundary conditions set over the original BREP are transferred to the nearest nodes of the fixed grid was developed, given that the calculated Fixed Grid Mesh does not completely fits the original BREP and that the entities (geometrical and topological) which compose the new representation are different than the ones on the original BREP. The algorithm follows an approach similar to the one described in previous sections to classify the nodes that are related to a specific facet (Algorithm 7)and then to a given boundary condition.

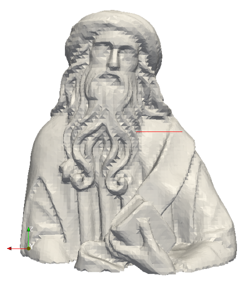
The previously stated algorithm is an approach suited for a particular implementation of a Fixed Grid FEA solver for the linear elastic problem.

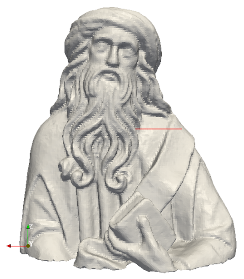
2.7.2 OpenFOAM based solution (Fluid flow problems)

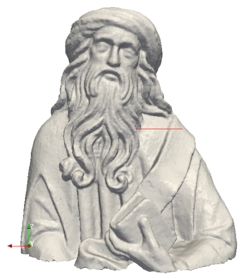
The ParaVoxel library allows the computation of volumetric meshes via FG algorithms. These meshes are generic and they have not been written in any valid format for a CFD solver yet.


(a) Kraft Original Model


(b) Resulting Kraft *NIO* elements 31x20x46 Grid (Mesh 0)


(c) Resulting Kraft *NIO* elements 62X41x71 Grid (Mesh 1)


(d) Resulting Kraft *NIO* elements 123x83x142 Grid (Mesh 2)


(e) Resulting Kraft *NIO* elements 246x166x284 Grid (Mesh 3)

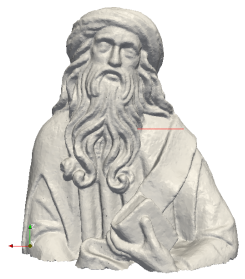
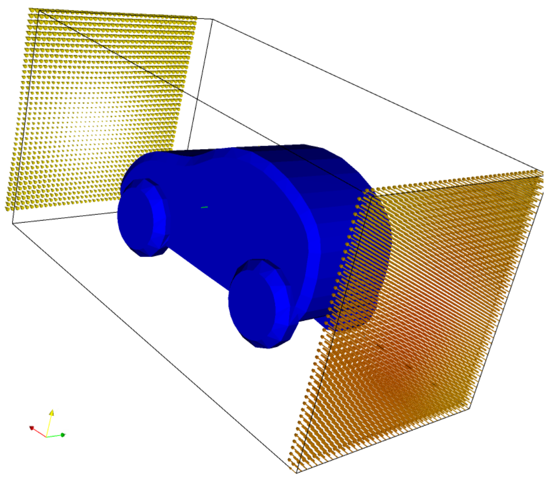

(f) Resulting Kraft *NIO* elements 482x324x568 Grid (Mesh 4)

Figure 2.12: ParaVoxel Kraft test case

Algorithm 7 *FacetNodeRelation*

```
for All rays $r_i$ do
   for All facet $f_m$ inside $FacetsPerRay[i]$ do
      $P_{r_i} = RayTriIntersection(r_i, f_m)$
   end for
end for
if $size(P_{r_i}) \neq 0$ then
   for All points $p_k$ in $P_{r_i}$ do
      for All nodes $n_i$ inside $[p_k, p_{k+1}]$ do
         if $[p_k, p_{k+1}]$ is not an intersection segment then
            $FacetNodeRelation[n_i] = f_m$
         end if
      end for
   end for
   for All nodes $n_i$ inside intersection segments $[p_k, p_{k+1}]$ do
      $FacetNodeRelation[n_i] = f_m$
   end for
end if
```

Figure 2.13: Boundary Conditions over the CFD Domain



OpenFOAM is an open-source toolbox with a large user database, and it is used to perform simulations from complex fluid situations to electromagnetics (involving heat transfer models, turbulence and chemical reactions). The library core is written in C++ modules adaptable to create custom solvers and pre-processing tools. As the internal structure of OpenFOAM is based on finite volume numerics, mesh structure is basic to perform calculations. The convexity of the cells should be respected during the complete process.

OpenFOAM allows the transcription of volumetric meshes into OpenFOAMs own *fvMesh* format via its *cellshapes* toolkit. Using the other tools in the patch-detection toolkits (e.g. angle / curvature changes), boundary conditions setting becomes possible when required.

Figure shows the mesh of a toy car inside a boxed domain with inlet and outlet conditions mapped on opposed faces after performing the import of the mesh into OpenFOAM for solving.

2.8 Discussion

The power of this approach compared with the traditional Fixed Grid Meshing lies in the ability to decompose the domain in order to minimize the number of intersections between the solid body and the rays that actually hit it. This process allows load ballancing on processors according to the number of times each ray (belonging to the set) hits the solid body (thus lowering the number of triangle/ray intersections to be calculated globally). The ability to decompose the domain makes this algorithm parallelizable (which has proven efficient enough to run on several CPU's). The data structures used to program this implementation, seem able to fit different architectures such as GPU implementations.

The dimensions of the perpendicular edges of the elements belonging to the fixed grid is a parameter of the algorithm, hence the applicability of the domain decomposition approach used in this algorithm can provide a useful approach if nested inside others, such as Building Cubes Method (For fluid simulations, See Komatsu et Al. [9]), to generate the basic hexahedral meshes on the initial stages and the refined volumes during the later stages.

The generality of this preprocessor and the built-in API used to calculate the volume and other properties of the NIO and IN elements suits the requirements of many academic research projects, such as the one proposed by Kim [14] and Garcia and Steven [13]] with their algorithms based on Genetic/ISO-ESO, where Cartesian grids with quad/hexa elements are required, and the calculation of the properties of the NIO elements is one of the inputs to penalize the local stiffness matrix of those elements regarding the ones classified as IN.

Chapter 3

Computational Steering of CFD Simulations inside Grid Computing Environments

3.1 Summary

This paper presents an overview of the architecture used for the implementation of an Interactive Computational Fluid Dynamics (*CFD*) environment, intended for studies mainly related with shape optimization of the airflow around virtual prototypes; The idea behind this type of implementation is to constrain faster the design domain that a group of designers has to face when facing a task such as this during the early stages of the development, where highly accurate results are not required. The aim of this paper is to present the abstraction of the requirements for the architecture of a grid computing system intended for *CFD* simulations and a some of the milestones faced for such implementation.

The structure of an automated server responsible for managing the CFD module and updating the new simulation conditions for the next timestep is the core of the project and will be a matter of discussion.The used scheme for distributing data and information as soon as they are available for both solver machine (located on the same computer or on a High Performance Computing *HPC* facility) and clients from the server, will be discussed as well.

The present architecture supports a *CFD* interactive solver (GNU/GPL CFD library) for incompressible laminar Navier-Stokes equations using the *PISO* algorithm. The application contains an embedded fixed grid preprocessor. The current implementation is inherently interactive allowing easy changes to the virtual prototype geometry and to the boundary conditions of a *CFD* simulation.

Finally, this paper presents the reader some results obtained for simple test-cases and a discussion about solution stability, accuracy and how useful this interactive *CFD* tool is for early mechanical design stages where highly accurate solutions are not yet necessary, and reducing the size of a broad design domain

is required.

3.2 Introduction

The need to be able to study interactively fluid flows around virtual prototypes is not new. Many interactive *CFD* systems have been developed in the past years such as: CHAM (Concentration, Heat and Momentum) Limited with their virtual wind tunnel system [6] , MIT's David OH with his Java virtual wind tunnel [17], the NASA Virtual Wind tunnel at AMES Research Center [5] and more recently Rank and Wenisch et al. with their high performance computing implementation [18] [25]. A wide range of applications beggining from the obvious hydrodynamic and aerodynamic analysis to testing the implications of testing different configurations of air conditioning systems in surgery and HPC centers require versatile, fast and reliable simulation systems.

One of the current limitations of many interactive CFD projects, is that most of them use pre-computed solutions not allowing for on-the-fly changes to the geometry and the boundary conditions, other however allow fast preprocessing and scenario setup, but their solutions are not accurate enough for commercial application .

The ability to change geometry and boundary conditions at each time steps is to our viewpoint critical in the design process as it creates a true sense of insights on how small variations of the design parameters/constrains affect the fluid flow around the virtual prototype. In section 3.3, we will describe the requirements for an interactive CFD simulations system. In Section 3.5, we will describe the basic architecture for a basic software implementation. In Section 3.6, we will present some recent results and analyze them from a precision and stability viewpoints. The paper end with the conclusions and the description of future development plans, which from the architectural point of view will allow to perform both stable, accurate and fast interactive simulations.

3.3 Interactive CFD needs and requirements

The main idea behind most of the *CFD* simulations is to obtain a global and accurate picture of the behavior of the fluid flow in contact with other fluids or solids. This kind of simulations try to reproduce physical phenomena present on the fluid flow and its implications, being this phenomena from non-steady natures in most of cases. Non-steady simulations represent a mathematical and computational challenge not to be taken lightly and are one of the best ways to obtain the idea of the behavior of an specific design under load conditions before its construction.

Interactive scenarios are useful for understanding the physics of the systems and how it responds to variations of the design variables, to optimize the design in terms of Energy efficiency or to improve the aerodynamic performance. For *CFD*, this variations might be classified into different classes due to the difference between their natures:

1. *Geometrical:* This variations are related to modifications of the geometrical aspects of the bodies that interact inside the simulation. The possibilities are:

Table 3.1: Variations on the *CFD* Scenario and their respective effects

Type	Change	Direct Effects	Possible Induced Changes
Geometric	Positioning	Mesh-Recalculation	Boundary Conditions Timestep
	Orientation	Mesh-Recalculation	Boundary Conditions Timestep
	Size	Mesh-Recalculation	Boundary Conditions Timestep
	Shape Modifications	Mesh-Recalculation	Boundary Conditions Timestep
	Addition-Subtraction	Mesh-Recalculation Boundary Conditions	Timestep
Physical	Fluid Properties		Timestep Mesh-Recalculation Fluid flow Regime Change
	Boundary Conditions		Timestep Fluid flow Regime Change Mesh-Recalculation
	Timestep		Mesh-Recalculation
State of the Simulation	Continue the simulation?	Start-Stop	

(a) *Relative positioning of a body.*

(b) *Relative Orientation.*

(c) *Size (Scaling).*

(d) *Shape modification of the elements.*

(e) *Addition-Substraction of bodies.*

2. *Physical:* The nature of this variations is related to the modifications on the physical parameters/phenomenon. The possibilities are:

   (a) *Fluid properties.*

   (b) *Boundary conditions.*

   (c) *Timestep of the simulation.*

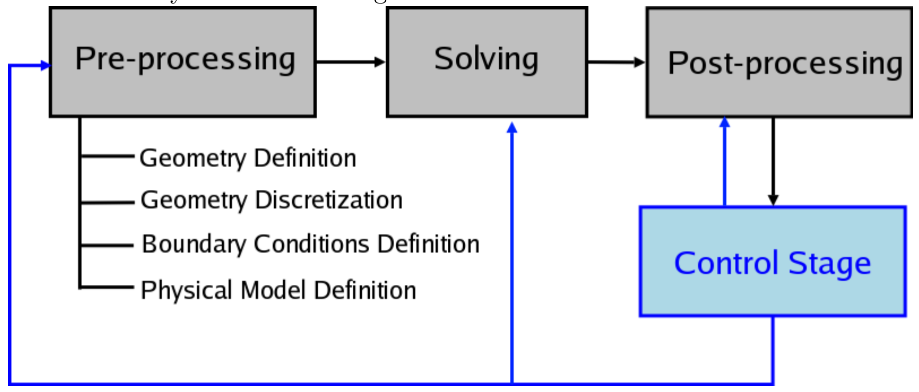
3. *State of the Simulation:* Setting Start-Stop flags is necessary to control the continuity of the process.

All the variations listed above are modifications of the scenario the user is interacting with, not over the application that is being used to solve the scenario. Most of the Geometrical changes force the recalculation of the mesh used for the computations and may as well induce other variations on the scenario, like changing the boundary conditions of the domain. The possible effects of performing a change over the *CFD* scenario are portrayed on Table 3.1, as well as the possible chain of events that one change might cause.

A single variation may induce a chain reaction that might produce a significantly greater computational cost than expected, e.g. If a body is added to the scenario, the whole mesh is to be recalculated, new boundary conditions are to be set over the body (no-slip wall conditions for example), and possible changes on the flow regime might be induced etc...

The main requirements that an interactive *CFD* methodology has are the same of a non-interactive methodology plus the ability to control the variations of the scenario listed in Table 3.1. The level of immersivity the user might desire will only generate new needs and wishes mainly related to the sensorial aspects of the interaction in the post-processing stage, but not with the basic information of the *CFD* scenario. As the post-processing stage of the simulation is interactive, a new *"Control"* instance must be included to guarantee the stability and interaction with the simulation. The Workflow diagram for an interactive *CFD* simulation-methodology is shown on Figure 3.1. It consists of four stages: *Pre-processing*, *Solving*, *Post-Processing* and *Control*.

Figure 3.1: Flow diagram of a *CFD* mesh Interactive method, Blue lines show the feedback by the Control Stage



1. *Pre-processing stage:* Geometry discretization, physical model definition and boundary conditions definition.

2. *Solution stage:* Given a physical model and a suitable CFD discrete scenario, a particular solver is selected and invoked for data processing.

3. *Post-processing stage:* As the solver starts producing all the results expected from the solution stage, all data is interpreted and displayed to the user using a graphical environment where he can interact with it.

4. *Control stage:* Parallel to the Post-processing stage, the control stage is fed from the results shown during post-processing and is the stage where the user analyses if any change over the current *CFD* scenario is to be made. If the *CFD* scenario changes somehow, the control structures or the user itself must perform a feedback process to all the other stages (*Pre-process, Solution and Post-process*) of the *CFD* simulation, providing the necessary data to continue the simulation.

The possibility to perform any of the variations over the *CFD* scenario shown in Table 3.1 suggest that the list of needs that arise from an interactive *CFD* methodology might be grouped into 2 sets:

1. *Non Interactive CFD needs Set.* This set of needs are mainly related initial scenario setup to be simulated.

   (a) *Pre-processing.* Geometry definition setup, Discretization Information, Boundary Conditions, Physical model, Numerical methods and various solver constrains.

   (b) *Post-processing.*

   - *Selected data for displaying:* given the massive amount of information the user can access after solving an scenario, choosing which sets of data are to be and how will be analyzed is necessary.
   - *Viewpoint and Scales.*

2. *Interactive CFD needs set.* This set of needs arises from the condition of being interactive during the post-processing via the control stage. This set of needs go hand by hand with the possibilities of variations over the *CFD* scenario described in Table 3.1.

   (a) *Geometry redefinition.* This subset of needs is mainly related with the topologycal and geometrical changes a *CFD* scenario might suffer during the simulation.

   i. *Body Re-positioning control:* To have control over the relative position between coordinate systems attached to the bodies involved during the simulation.
   ii. *Body Re-orientation control:* To have control over the relative orientation between coordinate systems attached to the bodies involved during the simulation.
   iii. *Body Re-sizing control:* To have control over the possible scalability of the bodies involved during the simulation.
   iv. *Body addition-subtraction:* To have control over the possible addition or subtraction of new/old bodies involved on future stages of the simulation, which will be positioned, oriented and scaled via the controls on the items (i),(ii) and (iii).

   (b) *Physical variables redefinition.* This subset of needs is mainly related with the changes of the physical constrains in a *CFD* scenario during the simulation.

   i. *Fluid properties control:* To have control over the physical properties that define the fluid (viscosity,specific weight and heat, etc...).
   ii. *Timestep control:* To have control over the timesteps used to solve the scenario.
   iii. *Boundary conditions control:* To have the ability to modify the present boundary conditions.

   (c) *Change the current state of the simulation:* To have control over the fact of stopping or continuing the simulation process.

3.4 CFD and Collaborative Workspaces

Collaborative workspaces are environments of virtual or real nature on which different people work in a collaborative fashion to achieve a common goal. Computer supported collaborative workspaces (CSCCW) are increasingly being used to enhance collaboration between people located on different spots of the globe.

The use of common tools such as 3D model navigation and handling on this environments is a natural choice and even though the integration of video communication systems between clients is a possibility to achieve a more fluid and personal interaction, the imminent appearance of gaps at conceptual expression and transmission of the generated knowledge and ideas is still difficult to avoid

Different schemas to achieve highly efficient collaborative virtual workspaces have been proposed by Bilinghurst and Kato [3] and others. A common denominator is that the communication pace can be used as means to minimize the impacts of changing suddenly in a short matter of time the conditions of the workspace. During the present research, a synchronous interaction scheme will be used as default, given the relative ease to control that the variations on the CFD scenario are performed in a sequential order.

3.5 Simulation Steering Architecture

3.5.1 Computational steering scheme

One of the main goals for this system is to use and develop interaction techniques for users located on remote spots via high speed network. Being the users different in nature due to the specifications of their available computational resources (operative system, computer capabilities, etc.) a Client/Server schema, where only some specific data transactions and messages are allowed, becomes the natural choice for development.

The Figure 3.2 presents the abstraction of the processes required to guarantee an stable and smooth interaction between the user located at a client machine and the server (simulation/user manager) and solver applications, both located on a remote solver machine. On the Figure 3.2 different vertical pools are established, one for each main process involved during the CFD simulation steering:

- **Data I/O and Steering**: This process is handled by the *Client* application. The Client should be able to handle all the defined user data requests and inputs, guaranteeing a fluent interaction and allowing the user to perform the steering of the scenario he desires either defining from scratch-new scenario or using a previously defined one.

- **Data distribution and Simulation triggering**: This process is handled by the *Server* application (a simulation/user management system). The Server should be able to handle the user requests performed from remote locations. The Server is multi-user oriented and the interactions presented on this diagram are only performed with a single user (analogically, the same tasks are performed for the interaction with any user).

- **CFD solver**: After the developments of the *Solver* this is has become a straightforward process when all the data is properly defined and set.

The solver only requires a properly defined scenario and its main task is to solve the PDE's that represent the fluid flow problem.

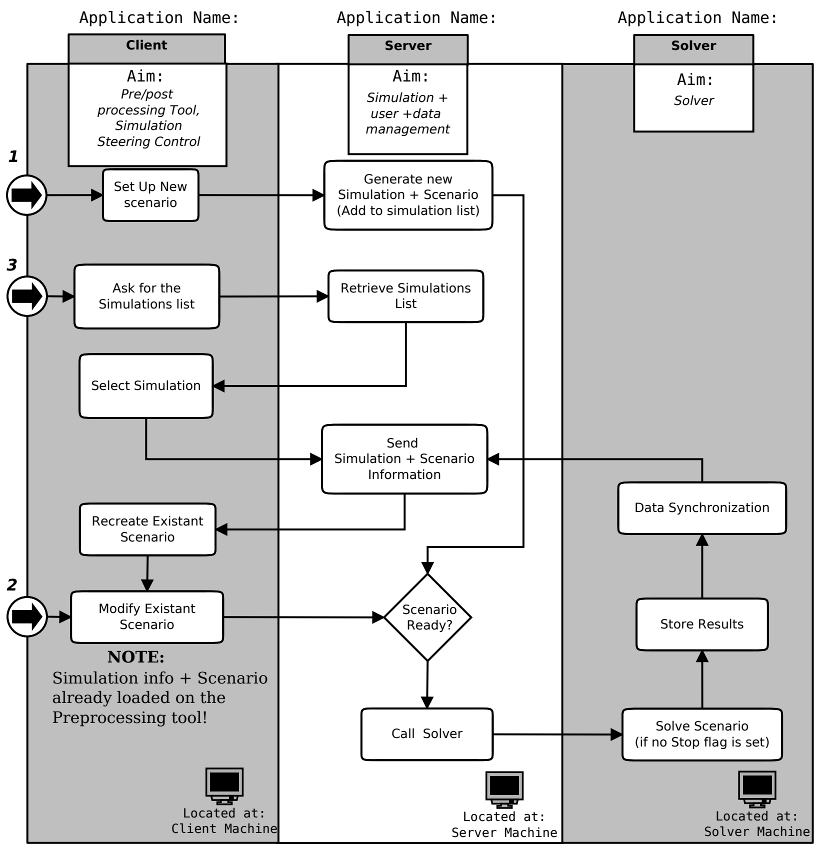

Application Name:
Client
Aim:
Pre/post processing Tool, Simulation Steering Control
Application Name:
Server
Aim:
Simulation + user +data management
Application Name:
Solver
Aim:
Solver
1
Set Up New scenario
Generate new Simulation + Scenario (Add to simulation list)
3
Ask for the Simulations list
Retrieve Simulations List
Select Simulation
Send Simulation + Scenario Information
Data Synchronization
Recreate Existant Scenario
2
Modify Existant Scenario
Scenario Ready?
Store Results
NOTE:
Simulation info + Scenario already loaded on the Preprocessing tool!
Call Solver
Solve Scenario (if no Stop flag is set)
Located at: Client Machine
Located at: Server Machine
Located at: Solver Machine

Figure 3.2: CFD Simulation Steering Dataflow

It is important to note that one user in start a new simulation, on which the user has to provide all the geometry files and scenario setup and later on be interested on steering (entry point number 1, Figure 3.2). On the other hand, another user might be interested on steer/watch the results of an already defined simulation, being possible to understand this case as subtask of the dataflow of the starting from scratch/steering a new simulation (as seen on Figure 3.2, starting from the entry point number 2). Entry point number 3 refers to the need of steering an already running simulation previously defined by another user.

The number of Client/Server communications and its size were minimized in order to keep network data transfer as low as possible, trying to avoid as many latency/bandwidth related problems.

The developed architecture allows several users to steer/access the same datasets from their own clients, providing the basic setup for collaborative design discussions in between the clients via communication services.

3.5.2 Setting up of a new scenario (Virtual Wind Tunnel example)

Having an organized scheme of how the ongoing simulation data will be managed, an abstraction of the minimum information required to perform the calculation and analysis of a physical phenomenon of a given nature is the step to follow.

A wind tunnel is basically a closed hall with an inlet and outlet which allow a transversal flow of air across it. The idea behind this computational implementation is to be able to steer the flow's inlet velocity by varying its magnitude and set the proper conditions over the body and the tunnel's wall which reflect the real flow around an object inside a wind tunnel. Wind tunnels are used in academic and industrial research to test the aerodynamic variables (Lift and Drag coefficients, pressure and and velocity distributions over the surface and around an object, etc...) on a particular design scenario.

A transient-state/incompressible/newtonian/laminar flow phenomenon will be used as pilot. The aim of the following sections is to analyze the data requirements to define a generic instance (Scenario) of a Wind Tunnel (Virtual Wind Tunnel, VWT), estimating the nature of the messages that should be passed from client to server and vice versa.

Tunnel Boundary

To define a new scenario on the VWT, it is necessary to establish a proper boundary for the wind tunnel itself, the bodies whose aerodynamic features will be tested, the wind speed at the inlet and various other information. Hexahedral shaped wind tunnels will be valid, and their orientation should be the same of the cartesian axis of the world coordinate systems (Figure 3.3).

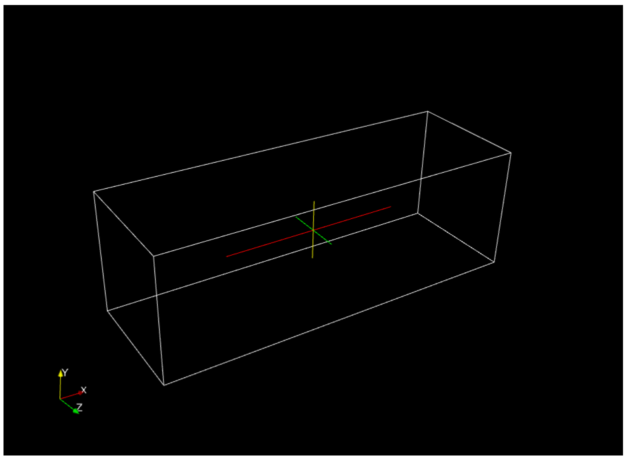


Figure 3.3: VWT Boundary

To define an hexahedral domain, the length of the `X`,`Y` and `Z` dimensions are required to define the travel length of the wind tunnel and its cross section. Being the boundary of the tunnel a geometric representation, a geometric center (`Center`) is required to set the tunnel on a exact place given the world coordinate system center as reference.

Addition-Transformation of bodies

Once the VWT CFD domain boundaries are set, the user can start adding *Bodies* to the scenario. *Body* refers to the boundary representation (B-REP) of a solid body. Different approaches can be used to define a B-REP, being piecewise planar representations the most commonly used. The location of the `geometric center` or any `fixed reference point` in terms of a triplet of numbers is required as well to establish a reference frame for the body position (See Figure 3.4).

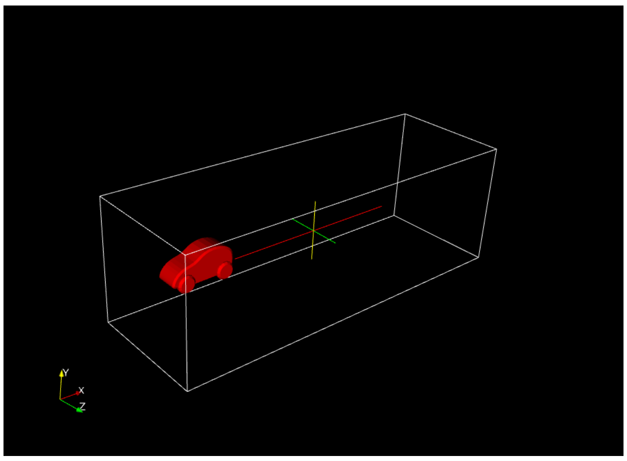


Figure 3.4: VWT boundary + Body

Once the body is located inside the boundaries of the tunnel, the a set of geometric transformations should be applied over it to achieve a correct orientation. The selected approach is to apply a set of `Translations` and `Rotations` of the body respect to the world coordinate system reference frame, using the center of gravity of the body as reference point for the rotation operations.

For some cases, the study case might require to analyze the interaction between various *Bodies* inside the tunnel. Following the procedure presented beforehand, The same operations should be performed to add more *Bodies* to the scenario as presented on Figure 3.5.

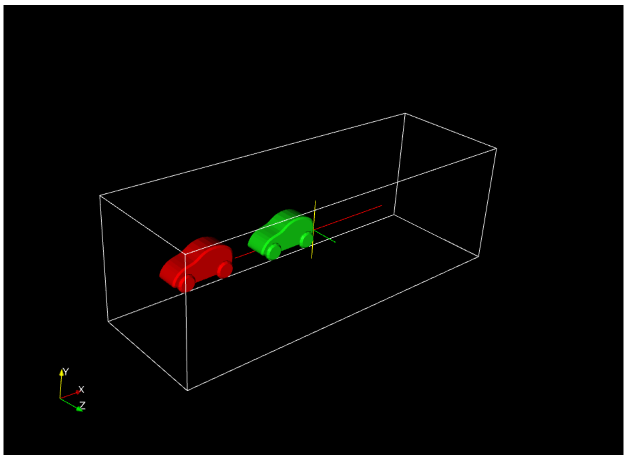


Figure 3.5: VWT Scenario with 2 Bodies

The geometric information to establish a CFD scenario should be available initially on the client side and should be transferred and stored on the server side, for simulation, reconstruction and distributing purposes among the clients and solver.

Initial and Boundary Conditions settings

Given that the present VWT scenario will be used as the general case, a basic set of boundary conditions that reflect the problem conditions should be set and parameterized.

For all the tunnel walls and all the faces belonging to the body, a No-Slip Wall condition (zero velocity) will be set by default. A pressure outlet condition will be used to set the reference pressure at the outlet of the system (atmospheric pressure) and for the inlet of the wind tunnel, a velocity inlet will be used and its magnitude is required as an input parameter, named U_0.

Zero Velocity and the same reference pressure will be used inside the whole domain as initial conditions, implying that the results from the initial timesteps will not reflect accurately the fluid flow, but as time develops, the solution will be more and more accurate.

3.5.3 Preprocessing

After all the conditions and parameters that define the scenario are defined by the user, the data should be transmitted, gathered and validated inside the solution server to reconstruct the scenario and perform the required finite volume method discretization, on which geometry, boundary conditions, initial conditions and physical parameters are integrated on a structure such that the solver is able to read all the data required by the algorithm for solution. The Finite Volume Method (FVM) relies on a mesh to represent the geometry of the scenario to be solved and hence, a fast and reliable meshing algorithm which can feed the solver when data is required. The boundary representation of the object that the user defined is handed over a Fixed Grid (FG) volumetric mesher (called ParaVoxel) to discretize the 3D CFD domain. FG meshing algorithms are broadly used as an efficient link between CAD systems and Finite Element/Finite Volume solvers given their low computational cost and versatility.[11]

It is important to notice that Paravoxel uses a Convex Hull approach to approximate the geometry of the elements near the boundary of the object, leading to defeaturing of the model, a problem when accurate calculations are required. However, for early design stages where performing fast-interactive testing to shorten the design domain, this type of Fixed Grid approximation is not only suitable, but an advantage.

The balance between the size of the Fixed Grid and the smallest feature of the solid body will be determinant to guarantee a proper representation of the boundary with the convex-hull elements. An analogy to the Nyquist-Shannon Theorem for analog signal sampling [21] [22] is evident during this process: *to achieve a close representation of the geometry by a fixed grid method, the largest size of an element in the grid must be at least half the size of the smallest feature of the solid body.*

3.5.4 Solving

Following the preprocessing stage, the data is sent from the client to the Server and from the Server to the Solver. All data should be coupled in terms of the data structures the solver requires for performing the simulation. This step should be transparent to the final user to guarantee an streamlined interactive experience.

For the presented architecture, a FVM solver for incompressible/steady state/newtonian flow is integrated with the preprocessor. The basic architecture presented in the previous sections is method independent, hence another solver which is suitable to solve the same physical phenomenon is adaptable, by means of changing the data structures passed on to the solver.

3.5.5 Postprocessing

During this stage the solution data is sent to the client machines either as a complete dataset or as a simplified video stream. The user must be able to control the viewpoint, colormaps and the amount and nature of the information produced during the solution of the given scenario. On the present implementation the following criteria were taken as reference:

- Basic Requirements: A set of basic filters such as streamlines calculation, isosurface representation, colormaps and glyphs should be available for the user.

- Environments: A 3D environment where the user is able to travel around the scenario should be available for users with high end computers/large bandwidth capabilities. Given the heterogeneous nature of the clients, a basic video stream using precomputed scenario viewpoints should be available for users with reduced bandwidth or computational resources.

- Tools: Interaction means such as haptic, 3D or multiple axis interaction means should be available for the user to get the most out of the interactive experience.

3.6 Test Case

The present section describes an implementation of the previously presented architecture. It starts presenting the technical requirements and issues faced during the implementation and on the later stage, a discussion around the user experiences while using the software.

3.6.1 Development Requirements for the Server

Following the order of ideas presented on section 3.5, the following initial list of requirements for the Server was harvested, summarizing the perception and ideas that will be required for this development. In front of each development requirement, the solution proposed to suffice this need will be presented.

- *Stability*: an Unix based operative system was chosen to be the host of the Server service. The stability of this monolithic-type kernels allows an easy restart of the service if required without having a huge impact on the other processes of the operative system.
- *Be able to run as a system service*: The server, programmed as a ruby script can easily be set as a system service which can be started or stopped at any time.
- *TCP/IP communication oriented*: The message passing between server/client was programmed over an internationally known standard protocol supported by libraries on many programming languages.
- *Easy modification and maintainability*: Being the code written on a simple language as ruby, the level of abstraction required to modify it is relatively simpler than the one required if the Server was written on other languages where complex data structures and castings are required.
- *Network file transfer optimized*: The third party package *rsync* was chosen due to its stability and optimality for file transfer and keeping up to date copies of a source folder over network systems.

3.6.2 Development Requirements for the Client

The following initial list of requirements for the Client was harvested, summarizing the perception and ideas that will be required for this development. In front of each development requirement, the solution proposed to suffice this need will be presented.

- *Stability*: Programmed over C++ and running on an extensively tested third party package called Paraview, the client can guarantee if properly developed the required stability levels for this kind of applications.
- *Have a powerful graphical engine for different purposes*: ParaView, an open source third party package was chosen as the graphic engine for this client. Of relative easy modification once its structure is understood, the

plugin programming option became a powerful tool for developing the client.

- *TCP/IP communication oriented.*The message passing between server/client was programmed over an internationally known standard protocol supported by libraries on many programming languages.
- *Easy modification and maintainability*: Paraview code is supported by a large community which can provide insight into adding further functionality and supporting the code.
- *Network file transfer optimized.*The third party package *rsync* was chosen due to its stability and optimality for file transfer and keeping up to date copies of a source folder over network systems.
- *OS portable*: Paraview is developed over multiplatform packages. Running the client on Windows, Mac or Linux based systems is not a big problem once the compilation stage of the Paraview platform is dominated.
- *Compatible with different I/O Devices*: The plugin architecture supports the usability of different I/O devices such as wii-motes, gamepads, etc... through the implementation of different control structures using the VRPN libray embedded on a Paraview plugin compatible with the client.

The client application should be able to perform the following operations for the user to interact with the server:

- **Connect/Disconnect:** By means of this functionality,the user can start/stop the communication with the VWT server located at the a given machine through an established port. The user should always identify himself using an *Username.*
- **List Simulations:** Lists the available simulations the user can start interacting with.The status of each simulation(currently running/stopped) should be available for the user to query.
- **Create Simulation:** This functionality allows the user to create a new simulation on the server side.
- **Get Online Users:** Lists the users currently logged on the server.
- **Get User Viewpoint:** allows to interactively obtain the current view point of any user logged on the VWT application for a given simulation.

3.6.3 Using an already existent Scenario

Any already existent simulation or scenario of the VWT can be loaded by the client. To load an already existent simulation, the user must first select a valid simulation from the *Simulation list*, inform the server that he will be working on it and then start the setup/modification of a new or existent scenario.

After the simulation is modified, the client will send all the modification requirements to the server, and it will start sending all the information required by the user to perform the steering and postprocessing. This operation requires an amount of time proportional to the bandwidth and the amount of data transferred.

3.6.4 VWT Execution

Once the scenario has been properly established, the user must confirm this action and start the calculations. After the solution processing begins, the Client/Server communication and interaction described on section 3.5 starts. The user must explicitly confirm the scenario setup, ordering the server to execute the desired command. In case the user desires to perform any change on the scenario during the solution, he is totally free to modify the scenario to his will and to refresh the scenario inside the solver. To change the case being solved, the user must explicitly once again tell the server that there is a modification on the scenario.

3.6.5 User Feedback

Two tests were conducted upon the implemented VWT platform:

Large model Visualization steering

A large model of the Buoyant winds on the Aburra Valley in Colombia (2.6 million cells approx.) was used as dataset for interactive postprocessing between two groups of people on different locations. The first group, located at Eafit university (Medellin, Colombia) used conventional screens and mouse as interaction means, while the second group, located at Los Andes university (Bogota, Colombia) used a large format screen and remote controllers (wiimote type) as interaction means. Both groups were connected via Access Grid [15] conference using audio and live video feeds as intercommunication. Both sites are connected via the RENATA high speed network (30 Mbps). No simulation was run and the platform was only tested as means to set up a remote collaborative environment meant for discussion.

Once the VWT platform was set up on both sites and the dataset was shared, an initial discussion topic was established and for 30 minutes an interactive chat between both parties through the video conference and the same simulation dataset was held. Figure 3.6 presents the setup of the interaction devices used on both sites.

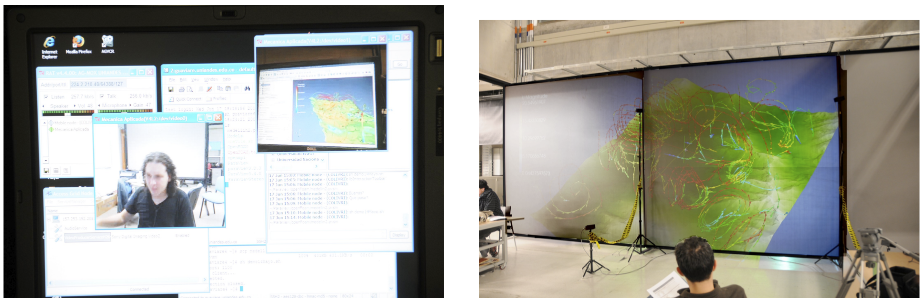
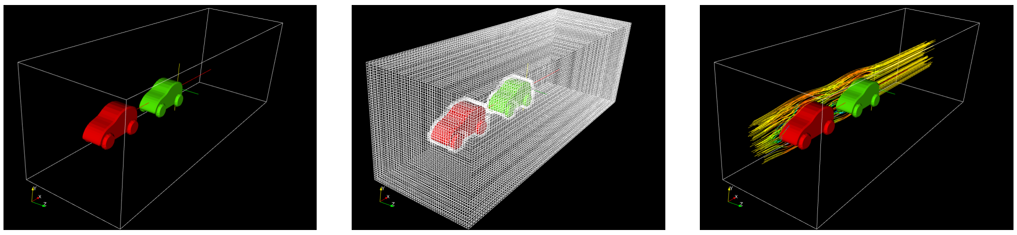

(a) EAFIT University interaction setup (b) Los Andes University interaction setup

Figure 3.6: Interactive Post processing test

As a result of the previous test, the following set of observations are in order:

- Fluent and stable communication between client and servers.



(a) Initial Scenario (b) Initial Scenario Mesh (c) Initial Scenario Solution

Figure 3.7: Original scenario example

- No locking points were found during the experiments.
- User interaction via remote controls was successful, but still requires more expertise on the user side.
- Once the dataset is located on both clients, bandwidth is not a big issue.
- Knowledge generation through discussion on different sites is highly enriched by the interactive experience. Communication skills are still key to guarantee a fluid discussion.
- The initial setup of the platform requires specialized personnel, while the interaction and discussion has proven that familiarity with similar tools is only a factor that ease up the initial approach to the tool. Once familiar with the environment, the user is free to roam the environment on any way he desires.

Simulation steering

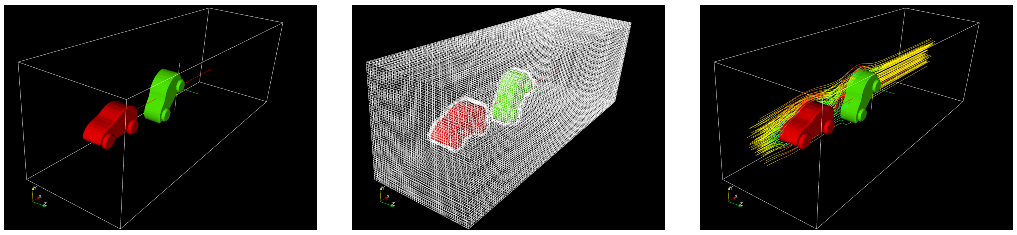
A simplified model of a toy car, formed by 715 facets, was was used as a test model for the VWT. The CFD domain around the body was initially subdivided into 150000 hexahedral cells. The simulation was remotely steered by two groups of people at different locations (EAFIT and Los Andes Universities).

Both parties, using Access Grid as video and audio conference system, joined efforts for coordinately steer the simulation. Figure 3.7 presents the initial state of a simulation that contains two bodies (Red and Green Cars) originally oriented on the same direction. Inside the same figure, the initial finite volume mesh used to solve the case can be seen and to its right, the streamlines that represent the solution can be seen as well.

After a brief discussion, it was required to change the orientation of the Green Car (45 degrees rotation respect to one of its principal axes). The operation was handled by the team at EAFIT and the instructions sent to the server. Successfully, the simulation was steered and visualized on both clients as shown in Figure .

Further testing was performed on the same example using the EAFIT network with 2, 3 and 4 clients, displaying an stable and functional behavior on all tests.

As a result of the previous test, the following set of observations are in order:



(a) Modified Scenario (b) Modified Scenario Mesh (c) Modified Scenario Solution

Figure 3.8: Modified Scenario Solution example

- No locking points were identified during the steering. The data exchange worked as expected between client and server and between clients.
- The preprocessing times were fit enough to guarantee a relatively fast simulation setup. The initial required parameters were intuitive enough for the group of users to setup the scenario and recreate the phenomenon.
- Data exchange becomes the bottleneck of this schema due to the large datasets being transferred to the clients. Different schemes than TCP should be tested in order to optimize dataflow.
- The users had to be properly trained in CFD practices for properly steering the simulations; Non trained users found difficult to understand the parameters and their physical meaning.

Chapter 4

Conclusions

The developments on Paravoxel resulted in a stable multiplatform and scalable Fixed-Grid Pre-processor, suited for any kind of application that requires a fast, accurate and reliable discretization of 3D domains, taking into account that the focus on the detail of the features of the object is not the main goal.

The tool behaves in such a way that given a boundary representation of the geometry and the dimension of the required elements, the discretization is automatically calculated without requiring any user-based expertise or work. In some cases, where preliminary shape design tests are needed, the defeaturing may present an advantage for both speed and for providing hints about what the shape might become during the CAE analysis.

Regarding the idea of collaborative workspaces for CFD simulations/training, it represents significant opportunities to developing countries specially given the relatively low resources that have to be spent to collaborate with experts all across the globe. High speed networks infrastructure and connectivity represent the means for cross-collaboration all across the globe and is one of the basic technical requirements which still can't be avoided in order to deploy a platform for collaborative CFD.

The first implementation of this architecture provided useful information about the way users interact with simulation. A robust control system must be implemented to guarantee the stability of the simulation if multiple users try to change the simulation scenario during the same session.

Message passing time is key to guarantee stability and the depth of the interactive experience. The disadvantage of this process becomes the available bandwidth between Server/Client/Solver machines. It is highly recommended to run both Server/Solver inside the same cluster or computer, to minimize the latency times and maximize the available bandwidth.

Finite Volume Method, though stable and accurate for solving CFD phenomena, have proven not to be the best choice in terms of minimizing solution time. A solver using Lattice-Boltzmann methods for solving Navier-Stokes will be used inside the same architecture seems like a promising next step for further research. It is important to notice that the same basic architecture can be extended to simulate different phenomenon such as turbulent flow or buoyancy effects adding the respective variables and parameters of interest required for the respective phenomenon.

The idea of streaming video previously rendered on the Server machine has proven to be a suitable way to avoid sending large data blocks over the network. The biggest opportunity presented by this post-processing scheme is the ability to handle clients which posses low computational resources/low bandwidth.

Even though no locking was evidenced during the user tests, not having a constrained GUI when the control of the steering was performed by another user was a matter of discussion between the users. It is recommended to add a GUI locking means while the steering is performed by a client different than the one sitting on a given terminal. The previous conclusion was later found to be implemented by Wood and Wright, successfully generating a feeling of safety for the users and the stability of the simulation.

The previous work presents a set of considerations to be taken into account for the implementation of this type of architecture, future work is encouraged into minimizing solution and message passing times, given the fact that the previous architecture proved stable, reliable and suited for grid networks with its nodes at both close and long ranged locations.

Bibliography

[1] M. J. Aftosmis, M. J. Berger, and Melton J. E. *Handbook of Grid Generation*, chapter 22:Adaptive Cartesian Mesh Generation. CRC Press, New York, 1999.

[2] C. Barber, D. Dobkin, and H. Huhdanpa. The quickhull algorithm for convex hulls. *ACM Trans. on Mathematical Software*, 22(4):469–483, 1996.

[3] Mark Billinghurst and Hirokazu Kato. Mixed reality - merging real and virtual worlds. In *Preceedings of the First international symposium on Mixed Reality*. Springer, 1999.

[4] ENCICLOPEDIA BRITANNICA@. Wind tunnels. Web, 2006. Available at: `http://www.britannica.com/ebi/article-9277765`.

[5] Steve Bryson and Creon Levit. The virtual wind tunnel. volume 4, 1992.

[6] limited CHAM@. Cham's virtual wind tunnel. Web, 2005. http://www.cham.co.uk/phoenicsvwt/autovwt.htm.

[7] P. Dunning, Kim A., and Mullineux G. Two-dimensional fixed grid based finite element structural analysis. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference preceedings*, Victoria, British Columbia, Canada, September 2008. AIAA/ISSMO.

[8] Herbert Edelsbrunner. *Geometry and topology for mesh generation.* Morgan Kaufmann Publishers, Elsevier Science, 1st edition, 2003. Chapter 2, Triangle Meshes.

[9] Komatsu K et al. Parallel processing of the building-cube method on a gpu platform. *Computers and Fluids*, 27(4):429–549, 2011.

[10] James D. Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, 2. edition, 1990.

[11] Manuel Garcia. *Fixed Grid Finite Element Analysis in Structural Design and Optimisation.* PhD thesis, Department of Aeronautical Engineering, The University of Sydney, March 1999.

[12] Manuel Garcia, Mario Gomez, Miguel Henao, and Juan Duque. Technical report : Hyperelastic-plastic simulation of textiles. Technical report, EAFIT University, 2005.

[13] M.J. Garcia and G.P Steven. Fixed grid finite element analysis in structural design and optimisatio. In *2nd Internet Conference on Approximations and Fast Reanalysis in Engineering Optimization*. AIAA/ISSMO, May 2000.

[14] H. Kim, O.M Querin, G.P. Steven, and Y.M. Xie. Improving efficiency of evolutionary structural optimization by implementing fixed grid mesh. *Structural and Multidisciplinary Optimization*, 24(6):441–448, 2003.

[15] Argonne National Laboratory. The access grid. Web, April 2012. `http://www.accessgrid.org/`.

[16] Tomas Moller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of graphics, gpu, and game tools*, 2(1):21–28, 1997.

[17] David OH@. The java virtual wind tunnel -a two dimensional computational fluid dynamics simulation-. Web, 2001. http://raphael.mit.edu/Java/.

[18] Ernst. RANK, Andre. BORRMANN, Alexander. DUSTER, Christoph. TREECK, and Petra WENISCH. Computational steering: Towards advanced interactive high performance computing in engineering sciences. In *(WCCM8) June 30, 2008 Venice, Italy*, June 2008.

[19] Joachim Schberl. Netgen - automatic mesh generator. Web, August 2006. `http://www.hpfem.jku.at/netgen/`.

[20] Philip J. Scnheider and David H. Eberly. *Geometric tools for computer graphics*. Addison-Wesley, 2. edition, 1990.

[21] C.E. SHANNON. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27, 1948.

[22] C.E. SHANNON. Communication in the presence of noise. *Proc. IRE*, 37, 1949.

[23] Victor L. STREETER. *Fluid Mechanics*. McGraw-Hill Book Company, 4th edition, 1966.

[24] Z. J. Wang. A quadtree-based adaptive cartesian/quad grid flow solver for navier-stokes equations. *Computers and Fluids*, 27(4):429–549, 1998.

[25] Petra Wenisch, Christoph van Treeck, André Borrmann, Ernst Rank, and Oliver Wenisch. Computational steering on distributed systems: Indoor comfort simulations as a case study of interactive cfd on supercomputers. *Int. J. Parallel Emerg. Distrib. Syst.*, 22(4):275–291, January 2007.