# *AGG – Automatic Grid Generator for USP PHOENICS*

By V I Artemov, July 9 2009

**Contents**

## 1. Summary

**What is AGG ?**
AGG is a part of program - preprocessor USPGrid for creating unstructured Cartesian grid with local refinement (ACM - Adaptive Cartesian Mesh). The Main feature of AGG is an automatic mode of the building the grid.

**Input data for AGG**
As input data for creating of the grid is used coarse structured PHOENICS grid and set of VR/Patch-Objects, *defining* process of the refinement the coarse grid. As such object are used:

- Blockage Fluid/Solid VR objects with fixed PRPS;
- Blockage VR objects with PRPS = 198 and 199;
- Usual VOLUME/CELL patches which set PRPS;
- 2D Plate objects (external or internal);
- 2D Inlet/Outlet objects (include WIND_PROFILE Object);

The above three types are defined by facets; the next by formulae.

- INFOB operators of In-Form, for which is set PRPS by operator (INITIAL;

The parameters for creating:

- MaxLevel – max level of refinement (the level of coarse grid is zero);
- NoLayers – the number of cell layers with the same level; is used for smoothing neighbor levels (default value = 2);
- InOutLevel – min level of refinement for cells near surface of Inlet/Outlet objects (default value = 1);
- PlateLevel – min level of refinement for cells near surface of Plate objects (default value = MaxLevel);

**The Rules of building**
For building of the grid are used following rules:

1) Near to surface of Blockage Object must be a cells with level = “Maxlevel”;
2) Near to surface of Inlet/Outlet Objects must be a cells with level not greater then “InOutLevel”;
3) Near to surface of Plate Object must be a cells with level not greater then “PlateLevel”;
4) The cells, placed inside Blockage objects with PRPS = 198/199, is **removed** from output grid;
5) The cells of the same level must have the *layers structure*: for any cell can to select the direction, in which exists “NoLayers” cells with this level;
6) The boundary surface of cells with different levels must be it is enough *smooth*: for any cell the number of neighbors’ of other level must not exceed 3 in 2D case and 4 in 3D case;
7) Is not allowed neighborhood of cells, which levels differ *more than on unit*.

The algorithm of AGG is very “simple”:

1) When a cell is refined, there are created 4 ( in 3D case) or 2 ( in 2D case) equal child cells;
2) In process of the partition is created cells Tree, beginning with cells of coarse grid. For this the parent cells index is stored in child cells. The Depth tree is “MaxLevel”. The Tree in AGG is used for quick finding of neighbors’ and for looping all “bottom” active cells in tree.

3) The cell is marked for refinement, if:
   - cell contain the surface of any object and not achieved the necessary level of cell (MaxLevel, InOutLevel, PlateLevel);
   - the cell is marked for refinement by smooth- or layers- algorithms;
4) The refinement process is running in cycle, starting from cells of the coarse grid. The Process is stopped, when cells, marked for refinement, are absent.

In detail this algorithm is described below.

**Output of AGG**

The important feature of ACM grids is that they lie between two extremes: structured grid and completely unstructured grid. This allows on the one hand working with grid, using two unstructured lists - a Cells and Faces. On the other hand, for calculate of the geometric features of cell and face possible to use the structured indices. For this AGG creates the 1D arrays of the faces Fine-cell: a structured grid for case, when all coarse cells are refinement to MaxLevel:

FiXP(1:NFX), FiYP(1:NFY), FiZP(1:NFZ)

The any cell in tree are used three fine-index (IFX, IFY, IFZ), which are an index of left lower fine-cells for given cell.
The following objects are created as result of work AGG:

- UNCells – List of all cells;
- UNFaces – List of all faces;
- UNVertex – List of all vertices of cells.
- UNObjects – List of objects;

which used by USP Earth.
For visualization by means of ParaView-postprocessor, VTK-files are created.

## 2. The Algorithm of AGG

How AGG works:

1) Prepare structured grids;
2) Read VR-object and its preparing;
3) Scan all VR-objects by Ray-algorithms;
4) Create Tree cells by refinement algorithm;
5) Create list of unstructured cells;
6) Create list of unstructured faces;
7) Check of unstructured cells and faces;
8) Create list of unstructured vertices of cells;
9) Create list of unstructured objects;
10) Create files for USP-solver;
11) Create VTK-files for ParaView.

### *2.1. Prepare structured grids*

For work AGG uses two structured grid. First grid is initial coarse grid (CoarseGrid), created by User in VR-Editor:

Xface(1:NCX), Yface (1:NCY), Zface (1:NCZ),

(These arrays are defined East/North/High face of cell). This grid is read from EARDAT.

The second grid is grid with max level of refinement “MaxLevel” (FineGrid):

FiXP(1:NFX), FiYP(1:NFY), FiZP(1:NFZ)
FiXface(1:NFX), FiYface(1:NFY), FiZface(1:NFZ)

where
$NFX = NCX*2^{MaxLevel}$ , $NFY = NCY*2^{MaxLevel}$, $NFZ = NCZ*2^{MaxLevel}$
FiXFace() - Right face of cell,
FiXP() - center of cell.

The cells of FineGrid is created from CoarseGrid by divided every coarse cell by $2^{MaxLevel}$ equal cells. Indices of this grid bellow be marked as
ifX , ifY, ifZ
and will be used for structured numbering cell all level. For cell with level < MaxLevel these indices is indices of its left lower fine-cell.

### *2.2. Read VR-object and its preparing*

For create grid from SPEDAT is selected VR-objects with OBJTYP = Blockage, Plate, Inlet and Outlet.
AGG is stored the objects information in list ARObjects() of structure:

```
type TARObject      ! type to describe structure of one object
 character*8 ObjName
 …
 integer    IBF_F(3),IBF_L(3)    ! indices BoundBox on finest grid
 …
 integer NoFacets    ! amount of faces in FACETDAT
 Type(TARRect), pointer :: Facets(:)
 integer(2)  PRPS
end type
```

Here

| | |
|---|---|
| ObjName | - the name of object; |
| PRPS | - value of material PRPS of Blockage object; |
| IBF_F(3),IBF_L(3) | - fine-indices Boundary Box of object; |
| Facets(:) | - list of prepared faces of object. |

For simplification of the Ray-algorithm list ARObjects() is sorted so that in the beginning are Blockage objects.
The List ARObject() is used only in algorithm of the scan. After termination of the algorithm this list is destroyed for free of memory.

### *2.3. Scan all VR-objects by Ray-algorithms*

For the process of the refinement of cell necessary quickly to get the next information:

1) There contain or not the cell intersection with surface one of the object;
2) if intersection there IS, that necessary to know the index ObjID this object in array ARObject();
3) if intersection NO, then necessary to know PRPS and ObjID object, inside which is located cell; if cell is located outside of all objects, then its PRPS is equal DomainPrps and ObjID = 0.

For creation of this information are used Ray-algorithm, but for storing - an arrays of the structures:

```
Type TRay
```

```
  integer(1) NoCross
  integer(1),pointer:: TypeCross(:)
  real,pointer :: Cross(:)
  integer(2),pointer:: Prps_Before(:)
end type
```

Variable TRay is described intersections of ray with ***all*** objects from ARObjects():

NoCross - number of intersections; if NoCross = 0, then memory for arrays Cross(), … is not allocated;

Cross(NoCross) - coordinate of intersection points in ascending order;

TypeCross(NoCross) - type of intersection:
tcrVOL - with boundary of Blockage Object;
tcrSURF - with boundary of Inlet/Outlet Object;
tcrWALL - with boundary of Plate Object;

Prps_before(NoCross+1) - PRPS for cells, is located before current intersection;

The Scan is running on three directions through all centers of fine-cell. Three arrays are used for this

Type(Tray) RayX(NFY,NFZ), RayY(NFZ,NFX) и RayZ(NFX,NFY)

For filling this arrays:

1) Make loop over all object from ARObjects();
2) For every object (CurObject) make scan by three direction scaDir = 1(X), 2(Y), 3(Z);
3) The scan in direction scaDir execute ***only*** for rectangle of Fine-indices of object
   iabs = IBF_F(scaAbs) … IBF_L(scaAbs)
   iord = IBF_F(scaOrd) … IBF_L(scaOrd)
   where values of scaAbs and scaOrd is dependent from Ray-direction; below is using scaDir = Z, for which scaAbs=1, scaOrd=2. For every File-cell select ray with coordinates xRay= FiXP(iabs), yRay = FiYP(iOrd) and search all intersection points.
   For this are examined all facets of object CurObject%Facets(1:NoFacets) and are found (or are not found ) ray cross point with each facets. This points are stored in Type(Tray) work variable CurRay;
4) Make check of correctness of crossings for CurRay;
5) CurRay Points is added to RayZ(iAbs,iOrd). For this is used algorithm of *overlapped* objects: the existing cross-points overlapped previous intersections.

### *2.4. Building cell tree by means refinement algorithm*

For procedure of the splitting the cells are determined in the tree form. For cell description is used the structure

```
type TARCell
  integer*2 ifx,ify,ifz        ! indices of structured grid for FINE level
  integer*1 lev                ! level of cell (zero based)
  type(TARCell), pointer :: Childs(:)    ! always 4 childes for 2D and 8 for 3D
  type(TARCell), pointer :: Parent
  …..
  integer(2) PRPS          !   PRPS for CELL Blockage Object
  ….
  logical*1  MakeRef
  integer*1 TagLayer
  integer UnID
```

**end type**

Here

| | |
|---|---|
| ifx,ify,izx | - structured fine indices of cell; |
| lev | - level of refinement (0 … MaxLevel); |
| Parent | - parent cell of previous level; |
| Childs(:) | - child cells of next level; |

In the beginning is created array of cells, correspond to CoarseGrid

```
Type (TARCell) ARCells(1:NoARCell)
NoARCell = NCX*NCY*NCZ
```

As index ID is used

ID = icx + NCX*(icy-1) + NCX*NCY*(icz-1)

where icx, icy, icz – structured indices of CoarseGrid.

This cells:

- don't have childs,
- are filling with PRPS = DomainPRPS;

***Refinement process***

Refinement of cells is made in the *first* cycle, consisting of stage:

1) Mark cells for refinement;
2) Making the layers of cells;
3) Refinement one level;
4) Smoothing layers of cells;

The Cycle finishes if not more cell for splitting.

Further is making *second* cycle of the smoothing - a removing "holes".

And at the end is made final processing of cells with level MaxLevel and containing cross points.

***Mark cells for refinement***

All active tree cells are examined and switch on the cell flag MakeRef.

On the *first* stage are checked:

- if level of the cell Lev >= MaxLevel, that cell don't split (MakeRef = .FALSE.);
- if PRPS cells is 198 or 199, that cell don't split too.

On *second* stage all rays from RayX, RayY and RayZ are analyzed:

1. if cell contains the cross point with Blockage-object, that cell is marked for refinement;
2. if cell contains only cross point with Plate-object and its Lev < PlateLevel, that cell is marked for refinement;
3. if cell contains only cross point with Inlet/Outlet-object and its Lev < InOutLevel, that cell is marked for refinement;
4. if cell does not contain the cross point, is made final calculation its PRPS. For this is used ray array PRPS_before (values are used between cross points, containing cell).

If cell is marked for refinement (MakeRef=.TRUE.) then for it is set the layer TagLayer = 1.

***Making the layers of cells***

Is set the number of current layer CurTagLayer = 1.

For creating layers are examined all cells with TagLayer=CurTagLayer. For each cell by means of arGetOneNeighb are found all neighbour cells and for this neighbour cells are set the flag MakeRef =.TRUE. and TagLayer = CurTagLayer +1.

These actions are repeated NoLayers time.

***Refinement one level***

For this are examined all cells, marked for refinement, and for each are created child cells - an array Childs(NoChild). The number of child cells depends on dimensionality of the problem. For 2D case NoChild = 4, for 3D case NoChild = 8. In each child cell is storage reference to the current cell - Parent. All properties of child cell inherit from parent (for example, PRPS).

***Smoothing layers of cells***

After refinement will appear the cells, neighbours of which with level will can differ more than on unit. As in algorithm USP PHOENICS such cells not used, that is made process of the smoothing. For this are examined all active cells (CurCell) and are examined all their neighbours (Neighb).

- If level CurCell%lev < Neighb%lev - 1, that cell CurCell is marked for refinement.
- If level CurCell%lev > Neighb%lev + 1, that neighbour Neighb is marked for refinement.

After that it is made refinement once again. This process is repeated until such cells not will.

***Removing "Holes"***

On completion of the refinement process in tree it is possible appearance cell, having "much" neighbors of other levels (for example, single "columns" of cells). For smoothing such cell is used algorithm of the layers smoothing with the another preparing function of cells.

In this algorithm for each cell is counted the number of neighbors with level, bigger than current cell. If the number of such neighbors more than 2 (2D case) or than 3 (3D case), that cell is marked for refinement.

***Final processing cell tree***

After above procedures in tree remained the cells, which are on object boundary and:

1) its level = MaxLevel;
2) its PRPS is not determined;
3) contain one cross point with one of the object.

In the current version for such cell is used the model "whole" cell:

- if cross point Cross(IC) <= point of the centre of the cell ZC, that PRPS cell = Prps_before(IC): the cell is located left of the boundary;
- if cross point Cross(IC) > point of the centre of the cell ZC, that PRPS cell = Prps_before(IC+1): the cell is located right of the boundary;

*Remark:*

If will use the algorithm **SBC (S**moothing **B**oundary **C**ell) , in this place of the algorithm will create Cut-cells (see below).

### *2.5. Creation unstructured cell, faces and vertices*

For USP, it is necessary to convert the created cell tree to list of unstructured cells and cell faces. For the postprocessor and SBC model it is necessary to create the list of the cell vertices and list of links cell to its vertex.

***Creation unstructured cell***

In AGG unstructured cells are described by variables

```
type TUNCell
  integer*2 ifx,ify,ifz     ! indices of FineGrid
  integer(2) lev          ! level of cell (zero based)
  integer(2) PRPS
  integer Nodes(8)
end type
type(TUNCell), pointer :: UNCells(:)
integer NoUnCells
```

where Nodes(8) - an array of vertex indices.

For createing UNCells(:) are examined all active cells with PRPS not equal 198 and 199. Information from tree cells is moved into unstructured cells. The Sequence of the unstructured cells corresponds to the order of their examined in arVisitCells. For the next creating the list of the faces in variable UnID of tree cell is saved index of the unstructured cell in UNCells array.

***Creating the list of the unstructured faces***

In AGG unstructured faces are described by variables

```
type TUNFace
  integer negCell,posCell
  integer(2) FLAG
  ….
end type
type(TUNFace), pointer :: UNFaces(:)
integer NUNFaces
```

where

negCell, posCell - an unstructured cell indices in UNCells(), connected with face; for boundary faces one of the indices is a zero;

The Field FLAG contains the set of bit flags:

ffBound - any boundary face;
ffSouth_North - not boundary on South-North direction
ffWest_East - not boundary on West-East direction;
ffLow_High - not boundary on Low-High direction;
ffSurface_FS - Fluid-Solid Surface;

For creating the faces list in ARCells- structure is added field

```
type TARCell
  integer, pointer :: FacesID(6,4)  !  first index - drWest...; second - 1:4
end type
```

which contains the unstructured indices of the cell faces. The *First* index describes the direction of face (drWest  drHigh), the *Second* - a relative index of face.

For creating the faces list is looked over all active cells and for each cell “CurCell” is examined its neighbour cells – “NB” in direction “iDir”:

1. if level of the neighbour NB%lev > level of CurCell%lev, that not to do nothing;
2. if FacesID(iDir,1) of cell is not equal zero, that face is already created;
3. to add the new face in list of the faces UNfaces() and set its properties;

4. to write index of this face into FacesID of cells CurCell and NB.

After creating the list of unstructured cells and faces cell tree is unused and its memory is freed!

***Creating the list of vertices***

Cells vertices are described by variables:

```
Type TUNNode
  integer(2) IFX,IFY,IFZ
 end type

 Type(TUNNode),pointer :: UNNodes(:)
 integer NUNNodes
```

where

ifx,ify,ifz - fine-indices West-South-Low of fine-cell vertex.

The linkage of the vertices with cells are described by array Nodes(8) of the cell, in which is stored indices of the vertex in list UNNodes. The rules to numbering the vertices of cell can be free. In AGG is used numbering of the package ParaView and Tecplot(see Fig. 1).

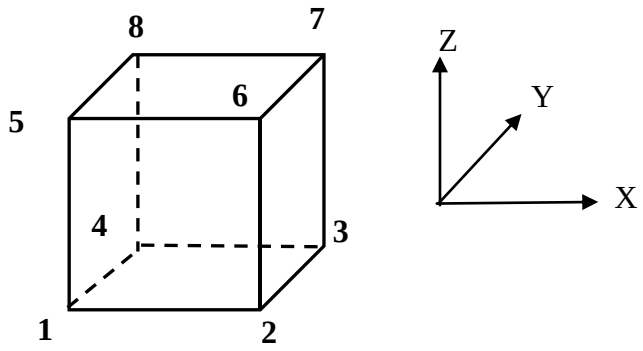



**Fig. 2.1**

### *2.6. The boundary conditions, the sources and initialization*

In structured PHOENICS for initializing PHI-variables, setting of the sources and boundary conditions are used patches, connected with VR-objects of the different types. In USP for this purpose is used list UNObject() of the following structure:

```
type TUNObject
  character*8 Name                    ! name of object
  integer(2) ObjType                  ! type of object
  integer(2)  IBF_F(3),IBF_L(3)       ! indices BoundBox on finest grid

  integer NoFaces                     ! amount of faces
  integer, allocatable :: FacesID(:)  ! array of indices of faces if it is needed
  integer NoCells                     ! amount of cells
  integer, allocatable :: CellsID(:)  ! array of indices of cells if it is needed
  integer(2) NoSources                  ! Number sources with
  type(TSource), allocatable ::  Sources(:)  ! array of sources for this objects
end type
Type (TUNobject) UNObjects(:)
integer NoUNObjects
```

The Structure TSource describes one source or one initialization for one PHI-variable:

```
type TSource                ! type to describe structure of data for sources of objects
  integer(2) MPHI           ! index of PHI in CHAM's order
  integer(2) TypeSource     ! type Source patch
  real Co, Val              ! Co and Val for source
  integer(2) TFst,TLst      ! first and last step of time for transient cases
end type
```

where TypeSource is the type of source connected with VR-object:

srCELL - Volumetric Cell Source,
srVOLUME - Volumetric Source;
srSURFACE - Surface Source;
srWALL - Wall Surface Sourcre;
srINIT - Initial Values

With standpoint of the geometries UnObject is list of unstructured cells and (or) list unstructured faces. In the current version of AGG for creating UNObject are used the same objects, as for building of the grid i.e. the objects of the type:
Blockage, Plate, Inlet, Outlet.

### *2.7. Output USP files*

The result report AGG is saving in file **usp_grid_log**.
For running USP Earth the program AGG creates four files with unstructured information:

- **usp_cells** - a description cells UNCells;
- **usp_faces** - a description faces UNFaces;
- **usp_vertex** - a description vertices UNNodes;
- **usp_objects** - a description objects UNObjects;

Below is described the structure of these files.

**File "Usp_cells"**

```
    Write(LU)  SIGNATURE_CELL, &  !  16 byte
               Title              !  48 byte
    Write (LU) NFX,NFY,NFZ, MaxLevel
    Write (LU) (XFine(ix),ix=1,NFX)
    Write (LU) (YFine(iy),iy=1,NFY)
    Write (LU) (ZFine(iz),iz=1,NFZ)
!
    Write (LU) NoUNCells
    DO ic = 1,NoUnCells
      Write (77) UnCells(ic)%PRPS, IXF-1, IXL-1,IYF-1,IYL-1, IZF-1,IZL-1
    ENDDO
```

where
IXF = UnCells(ic)%ifx; IXL = IXF + $2^{UnCells(ic)\%lev} - 1$ - fine-indices first and last fine-cells of current cell.
SIGNATURE_CELL - signature string for check file.

**File "usp_faces"**

```
    Write(LU)  SIGNATURE_CELL, &  !  16 byte

    Write(LU) NoUNFaces
```

```
do ifc = 1,NoUNFaces
  Write(LU) UNFaces(ifc)%FLAG, UNFaces(ifc)%negCell, UNFaces(ifc)%posCell
Enddo
```

**File “usp_vertex”**

```
write(LU)  SIGNATURE_VERTEX
write(LU)  NUNNodes
DO IV=1,NUNNodes
  write(LU) UNNODES(IV)%IX-1,UNNODES(IV)%IY-1,UNNODES(IV)%IZ-1
ENDDO
write(LU)  NoUnCells
DO IC=1,NoUnCells
  write(LU) UNCells(IC)%NODES(1:8)
ENDDO
close(LU)
```

### *2.8. Files for ParaView postprocessor*

For visual examination of created unstructured grid if flag UCRVTK = T, are created two VTK-files:

- **usp_cells.vtk** - a file of cells;
- **obj_faces.vtk** - a file with ObjID is not a zero faces.

The First file contains the unstructured cells as VTK-unstructured grid with HEXAHEDRON-cells. With VTK-cells is linked two scalar fields - PRPS and ObjID of cells.
The Second file contains the faces with ObjID <> 0 as VTK-unstructured grid with POLIGON-Cells. One scalar field -ObjID of face is linked with VTK-cells.
These files allow to visual examine the building grid and to check correctness of the object boundaries description.
The Examples of the use of these files are shown below.

# 3. Examples

Below typical examples (for cases with many objects) are shown.

## *3.1. 2D case: more objects*

This case uses 6 VR-objects.

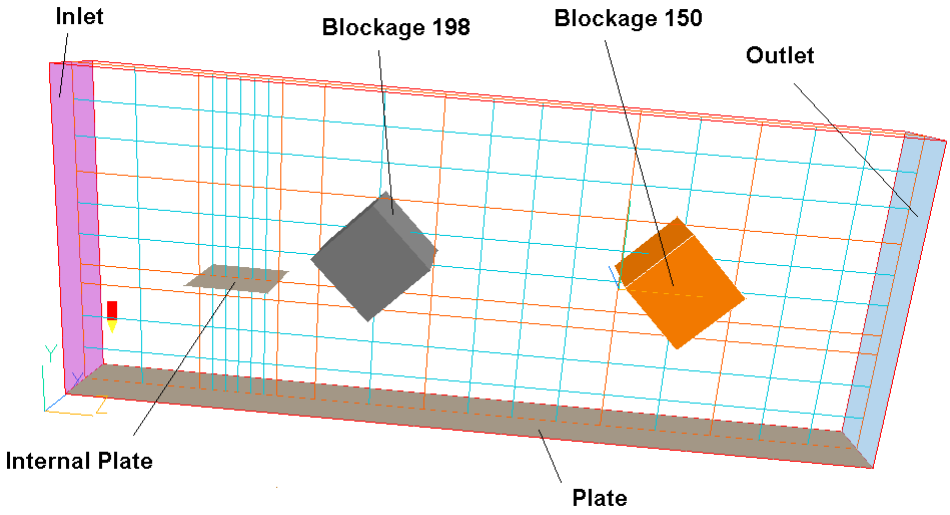



Fig. 3.1A ***VR-Editor*** domain scene

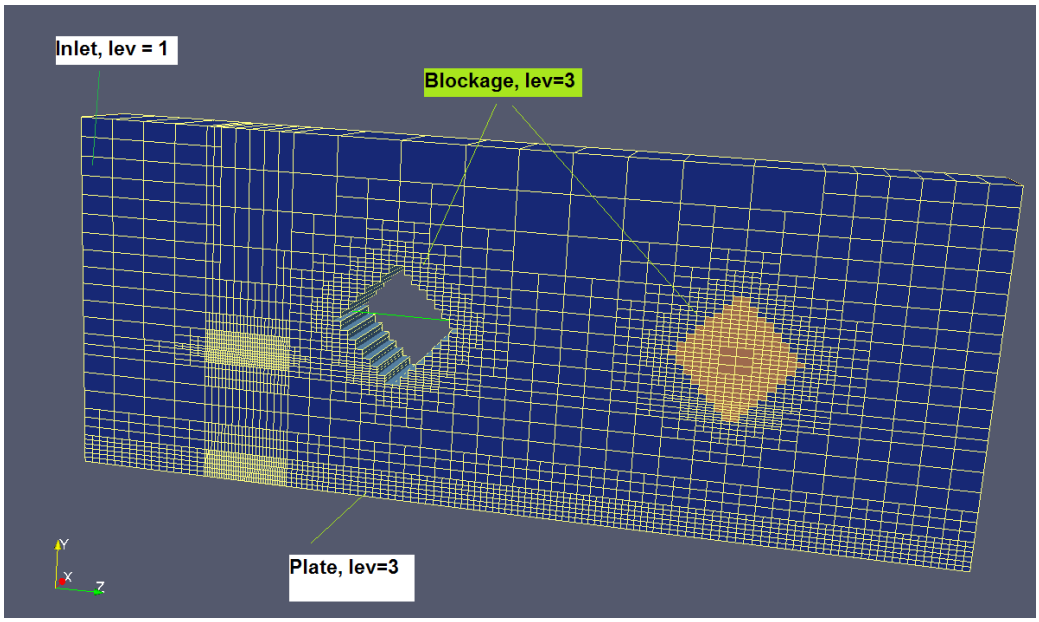



Fig. 3.1B ***ParaView*** : file usp_cells.vtk

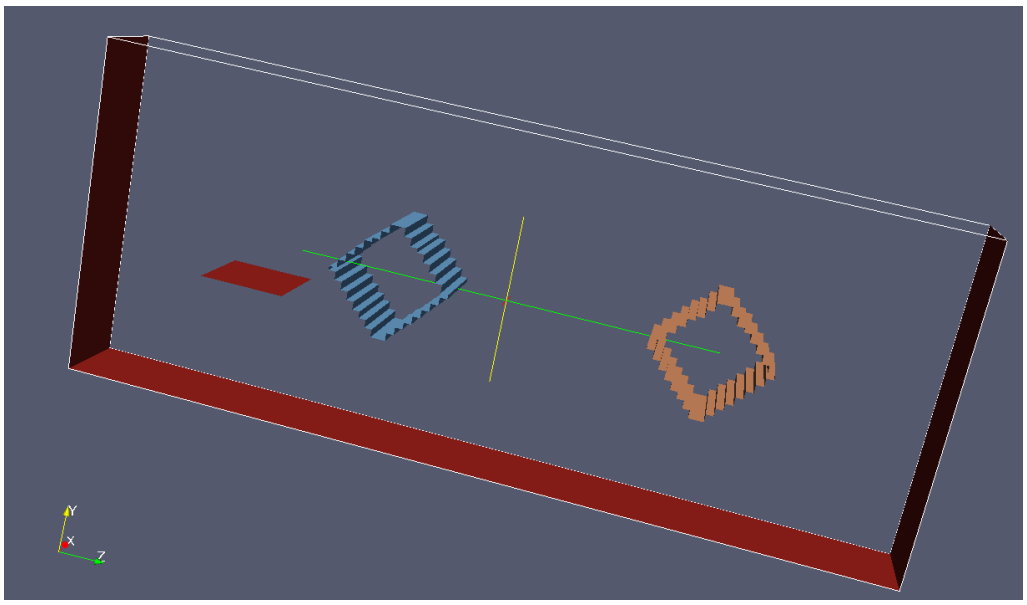


Fig. 3.1C ***ParaView*** : file obj_faces.vtk

### *3.2. 2D case: turned channel*

In this case is used Blockage Objects for describe of turned channel.

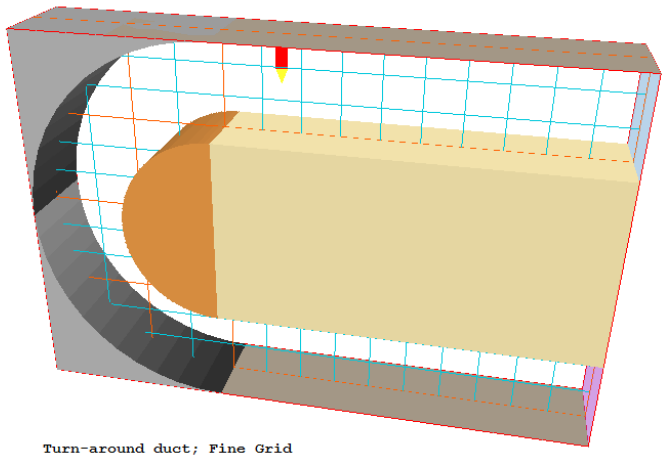



Fig. 3.2A ***VR-Editor*** domain scene

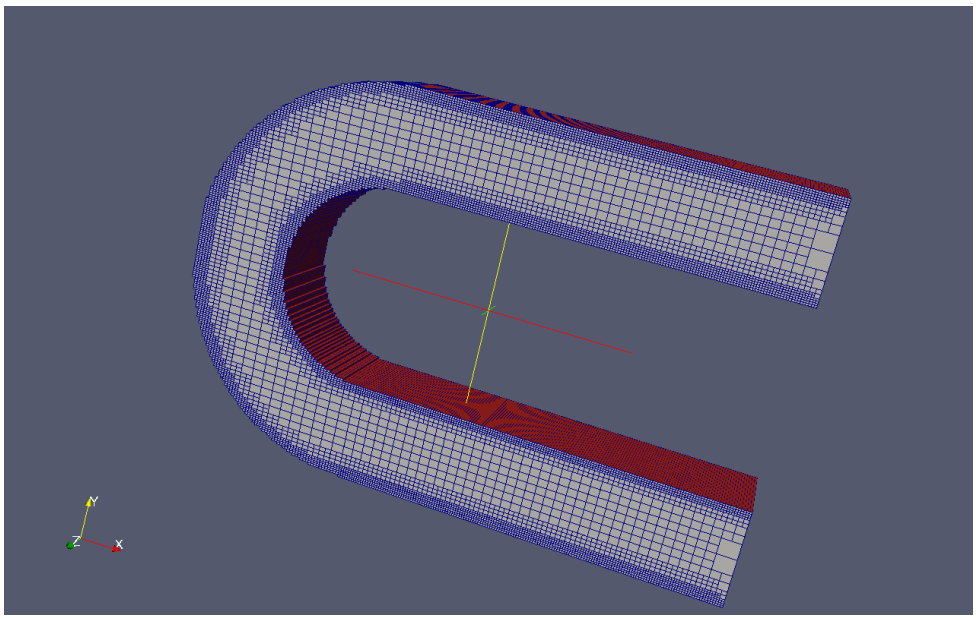


Fig. 3.2B ***ParaView*** : file usp_cells.vtk

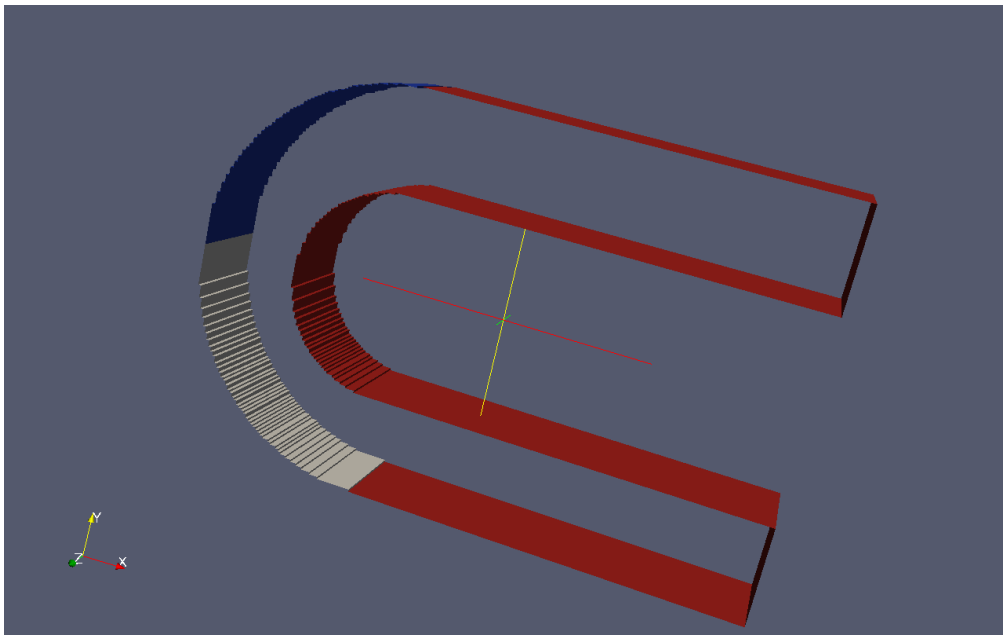


Fig. 3.2C ***ParaView*** : file obj_faces.vtk

### *3.3. 3D case: model of the buildings*

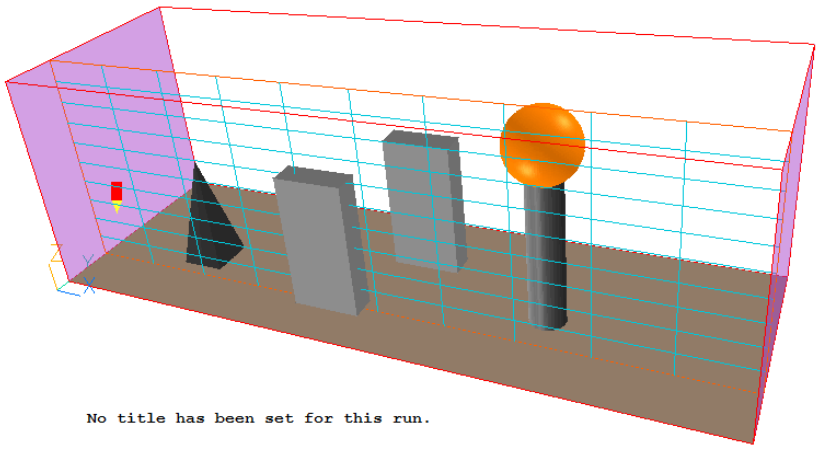



Fig. 3.3A ***VR-Editor*** domain scene

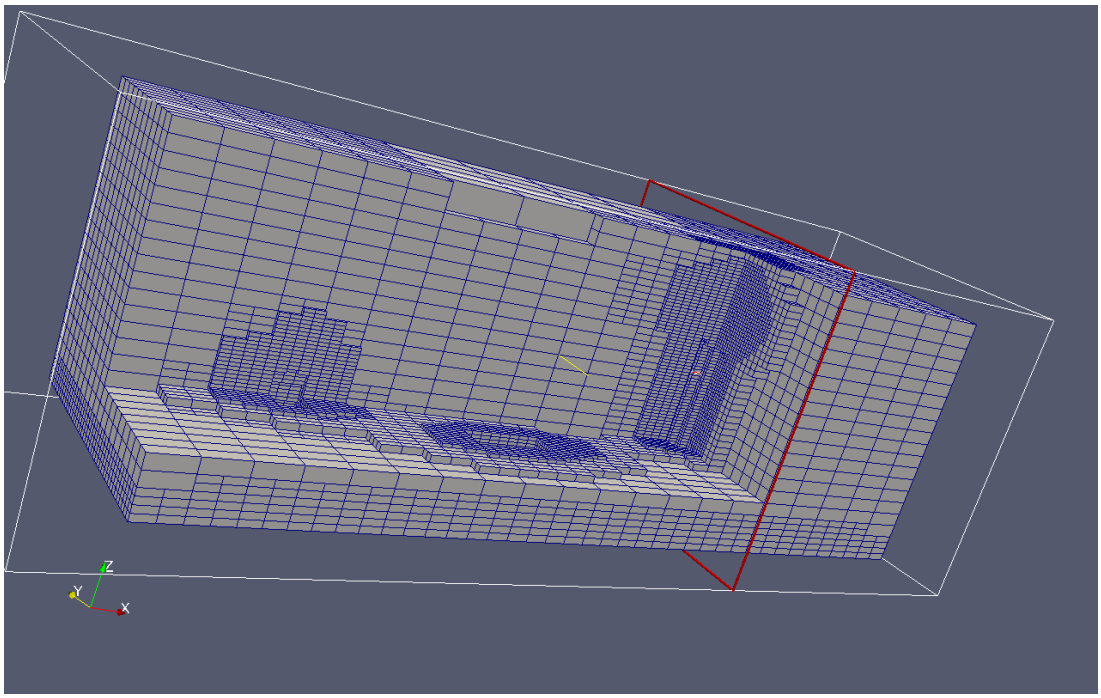


Fig. 3.3B ***ParaView*** : file usp_cells.vtk

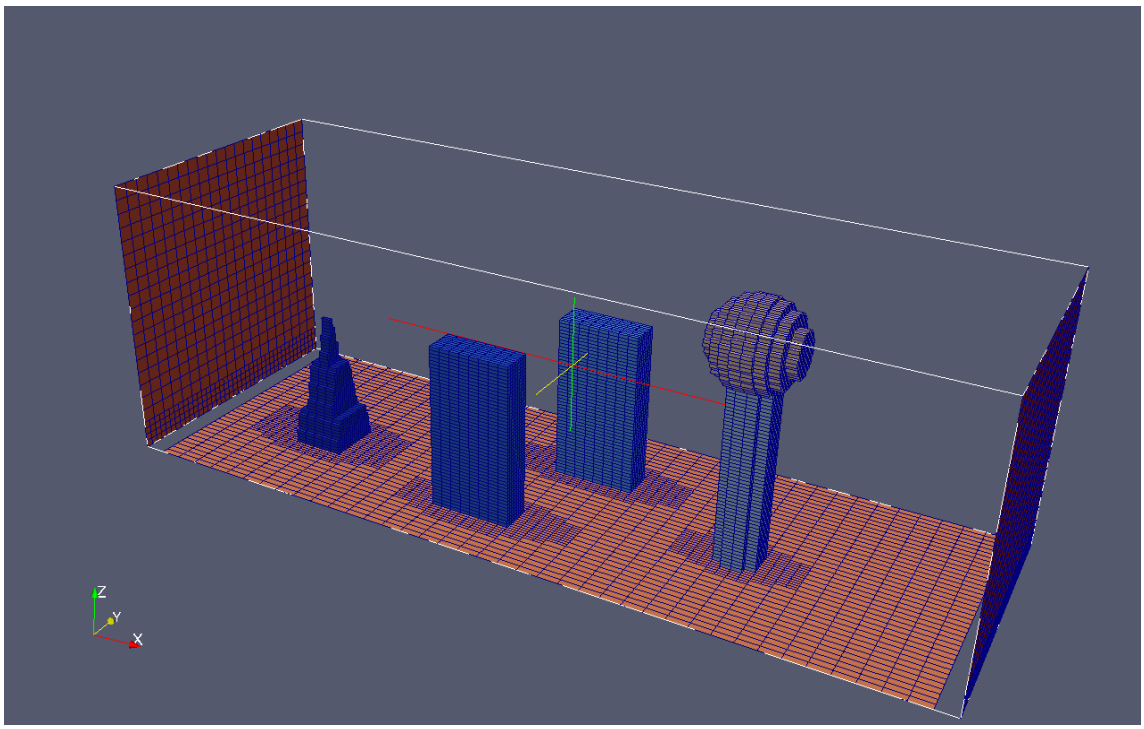



Fig. 3.3C ***ParaView*** : file obj_faces.vtk

### *3.4. 3D case: Car*

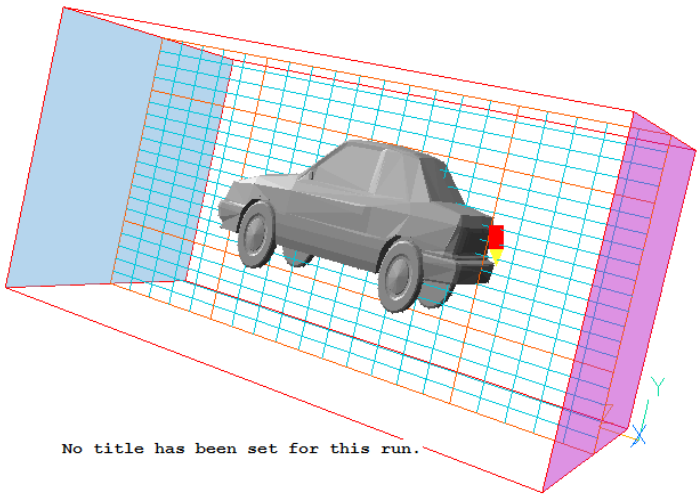



Fig. 3.4A ***VR-Editor*** domain scene

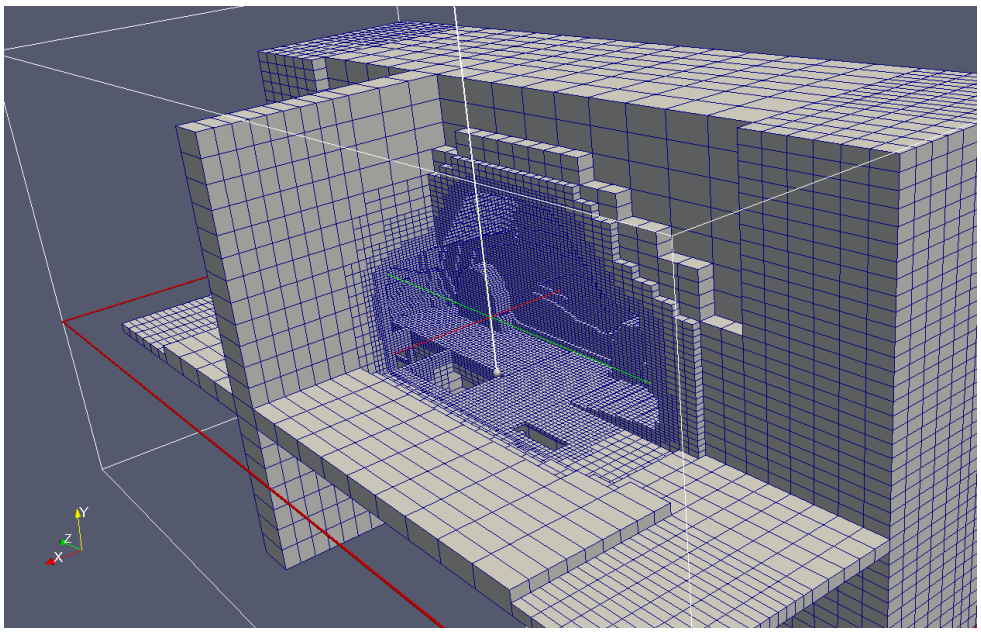


Fig. 3.4B ***ParaView*** : file usp_cells.vtk

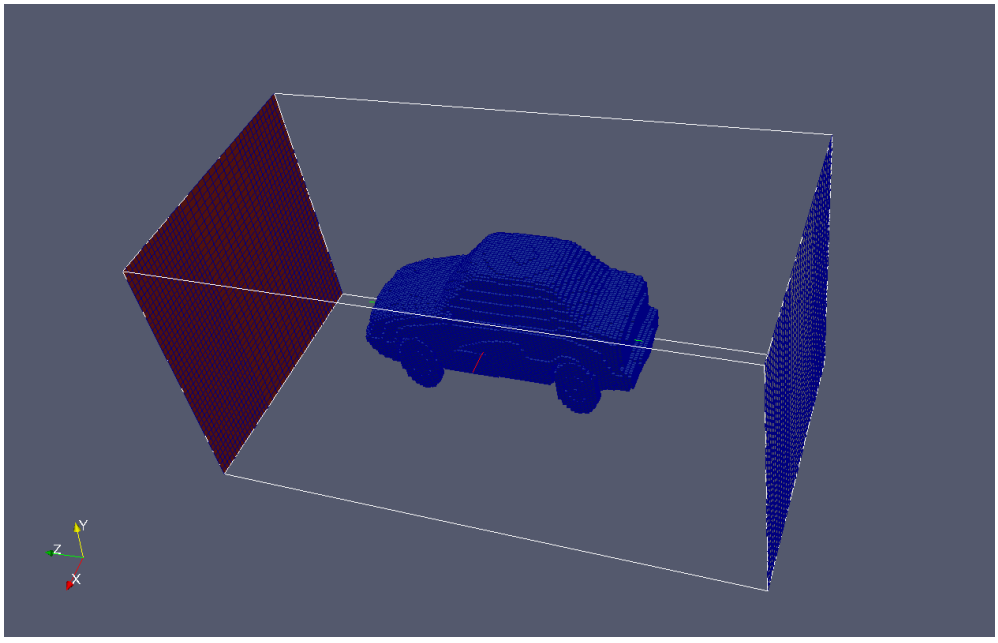


Fig. 3.4C ***ParaView*** : file obj_faces.vtk

### *3.5. 3D case: ASMO Project*

*In this case is used:*
Coarse grid = 10 x 10 x 10;
Maxlevel = 6;
WallLevel = 4.
*Grid properties:*
No. Coarse structured Cells = 1000;
No. Fine structured Cells = 262 144 000;
No. unstructured Cells = 802 135;

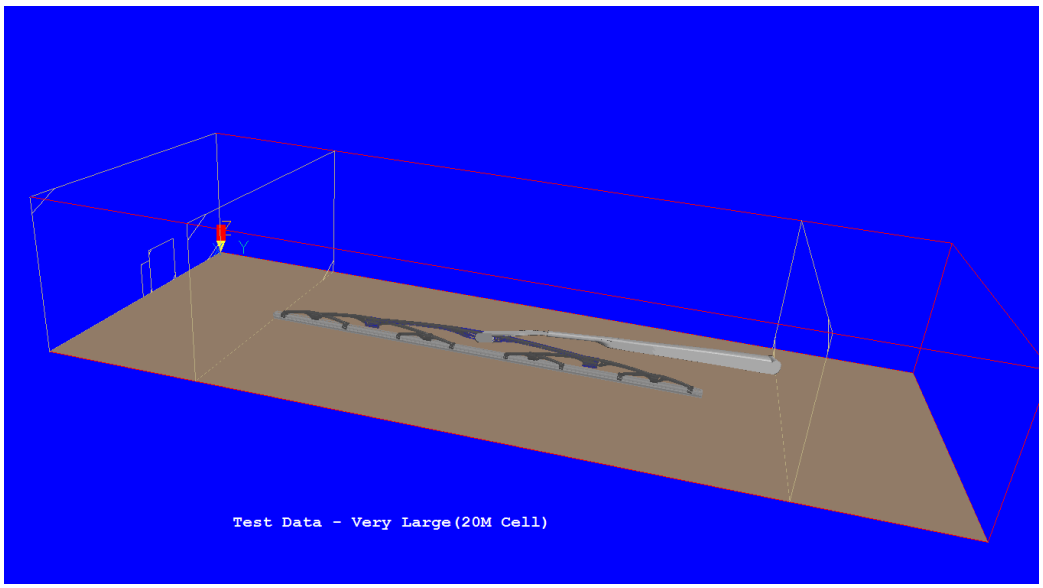



Fig. 3.5A ***VR-Editor*** domain scene

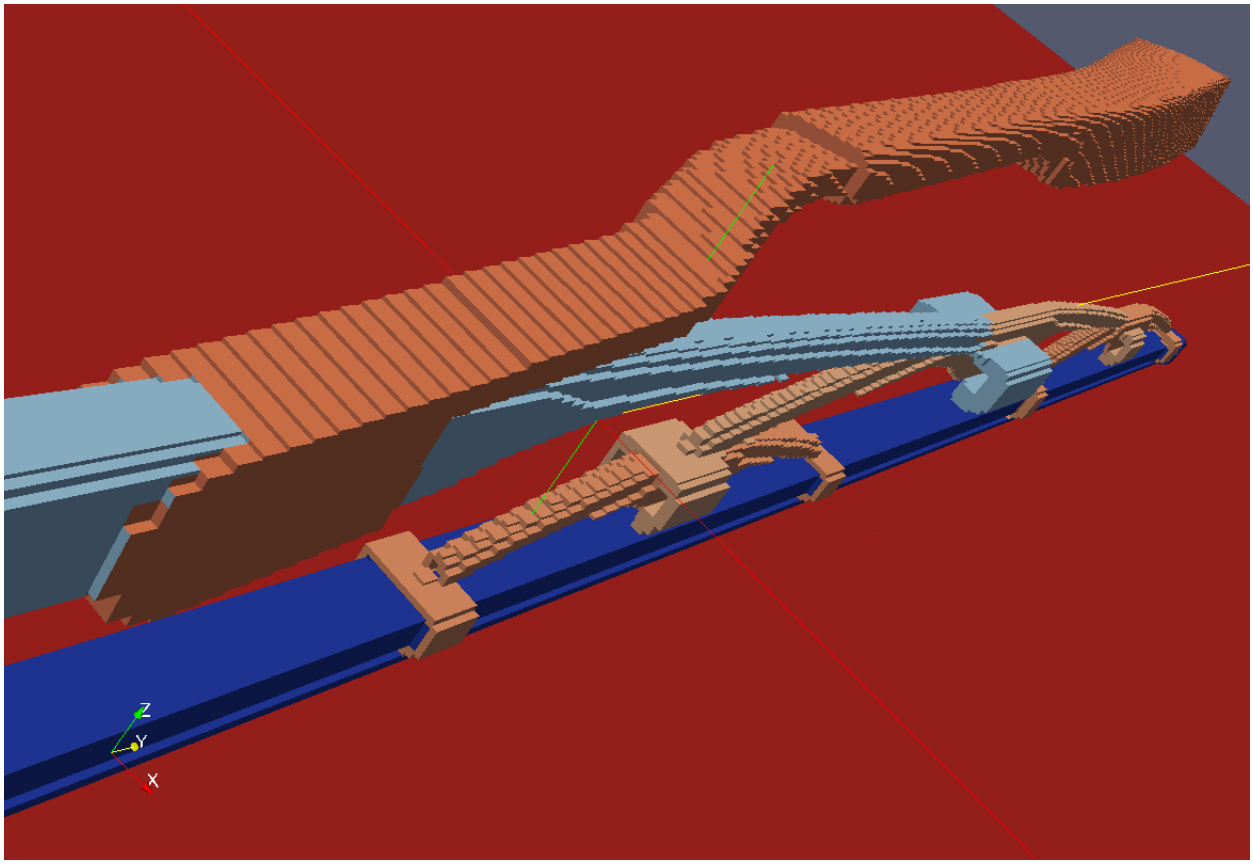



Fig. 3.5B ***ParaView*** : file obj_faces.vtk

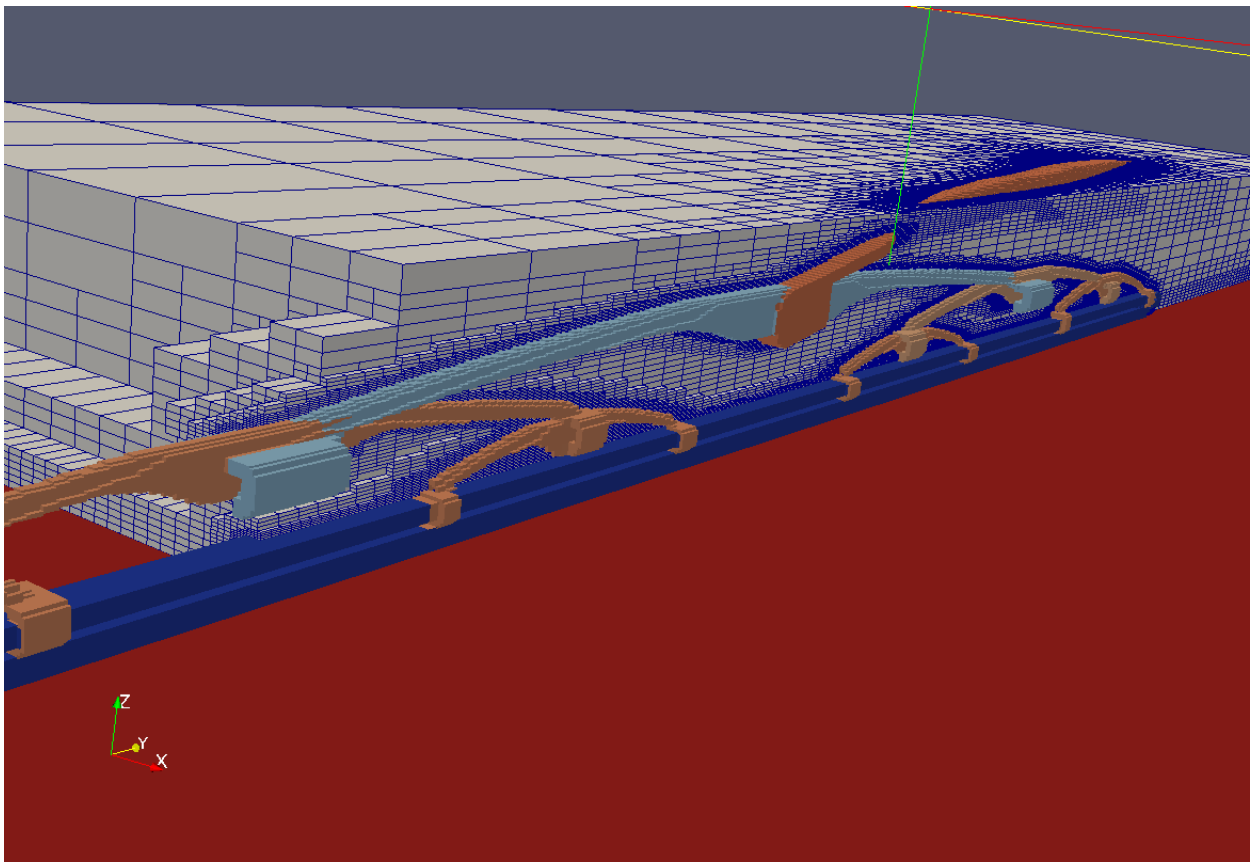



Fig. 3.5C ***ParaView*** : mesh and object surfaces (file obj_faces.vtk and usp_cells.vtk)

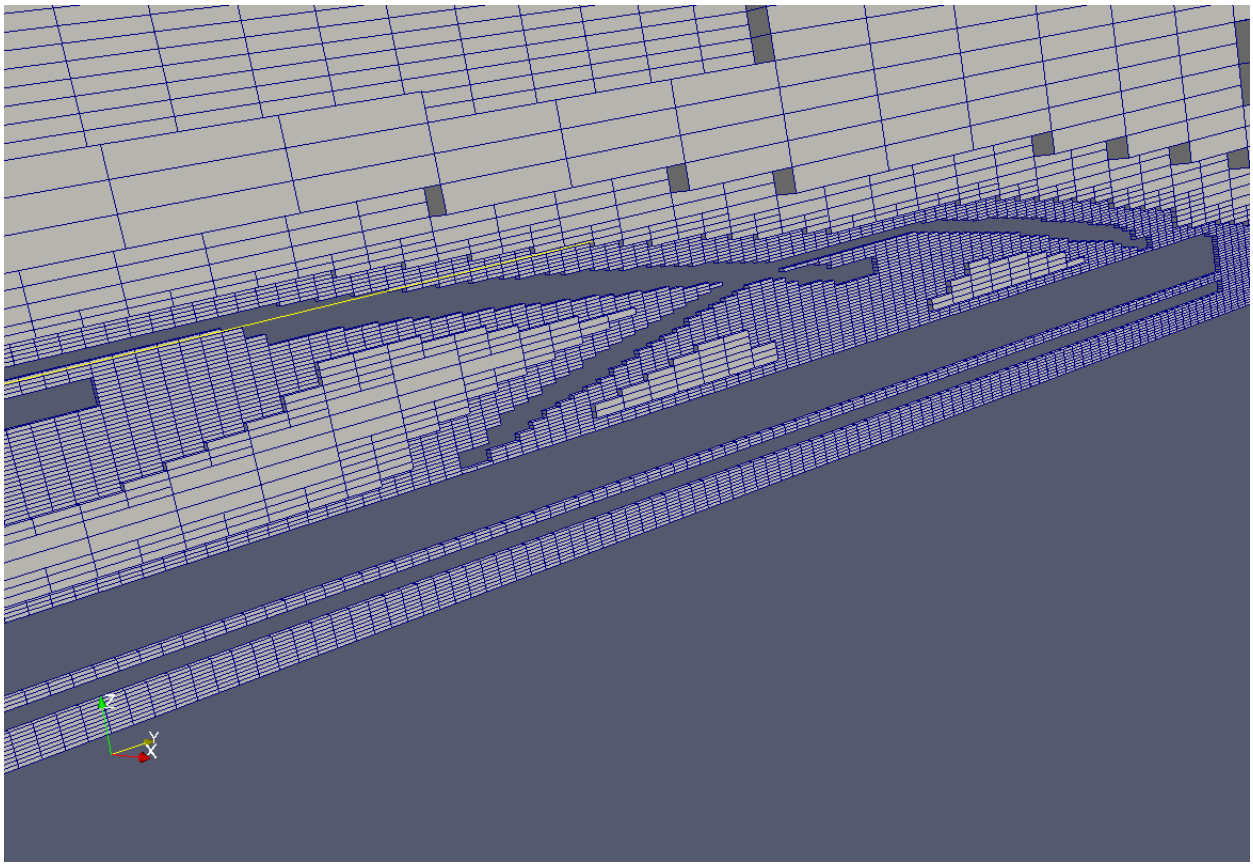


Fig. 3.5D ***ParaView*** : mesh (file usp_cells.vtk)

## 4. New accomplishments

In this chapter are described new accomplishments, included in the new version AGG:

1. work with INFOB objects,
2. work with SBC algorithm.

### *4.1. INFOB In-Form operators*

For creating the grid together with VR blockage objects it is possible to use Volume INFOB operators BOX, SPHERE and ELLPSD:

(infob at patch1 is box(x0,y0,z0,sx,sy,sz,al,be,th) with infob_1)
(infob at patch1 is sphere(xc,yc,zc,r) with infob_1)
(infob at patch1 is ellpsd(xc,yc,zc,rx,ry,rz,al,be,th) with infob_1)
где
patch1 - name of patch, which is used as restricting boundary box;
infob_1 - object name (for use in initial operator);
x0,y0,z0 - coordinates of left-bottom-low corner of boundary box of object;
sx,sy,sz - size of box;
r - sphere radius;
rx, ry, rz - radiuses of the ellipsoid;
al, be, th - angles of rotation object along X, Y and Z axes.

For use INFOB for building of the grid necessary to set PRPS value for object by means of operator

(initial of prps is ValPrps with infob_1).

where
ValPrps – numerical value (not formula).

For setting of restricting boundary box can to use usual Patch
PATCH(PATCH1, VOLUME, 1, NX, 1, NY, 1, NZ, 1, 1)

or Dot-Patch
PATCH(.PATCH1, VOLUME, 0, 1000, 0, 1000, 0, 1000, 1, 1)

At AGG-scan of the object all part of the object, outside of patch are cut off.
The objects BOX and ELLPSD can to rotate. As rotation centre is used left-bottom-low vertex of its boundary box. The Order of the rotation is following:

1) Z rotates about the current Z-axis of the box to TH angle (in radians), anti-clockwise looking along -Z,
2) Y rotates about the current Y-axis of the box to BE angle,
3) X rotates about the current X-axis of the box to AL angle.

Remark:

1) At rotation of object the place and shape of patch is not changed.
2) As the arguments of BOX, SPHERE, ELLPSD function can to use real values only. Can not to use In-Form formulas.

On Fig. 4.1 is shown the using BOX function for describe inclined channel.

The corresponding Q1 operators are:

```
(initial of prps is 0 with infob_1)
real(sx,sy,sz,th)
th=47/180*3.1416    ! 47 not 45
sx=2*xulast
sy=0.2*yvlast
sz=zwlast
(infob at .patch1 is box(0,0.01,0,:sx:,sy,:sz:,0,0,:th:) with infob_1)
real(COTH,SITH)
COTH=COS(th)
SITH=SIN(th)
(STORED of U_CH is U1*:COTH:+V1*:SITH:)
PATCH(.PATCH1, VOLUME, 0, 1000, 0, 1000, 0, 1000, 1, 1)
```

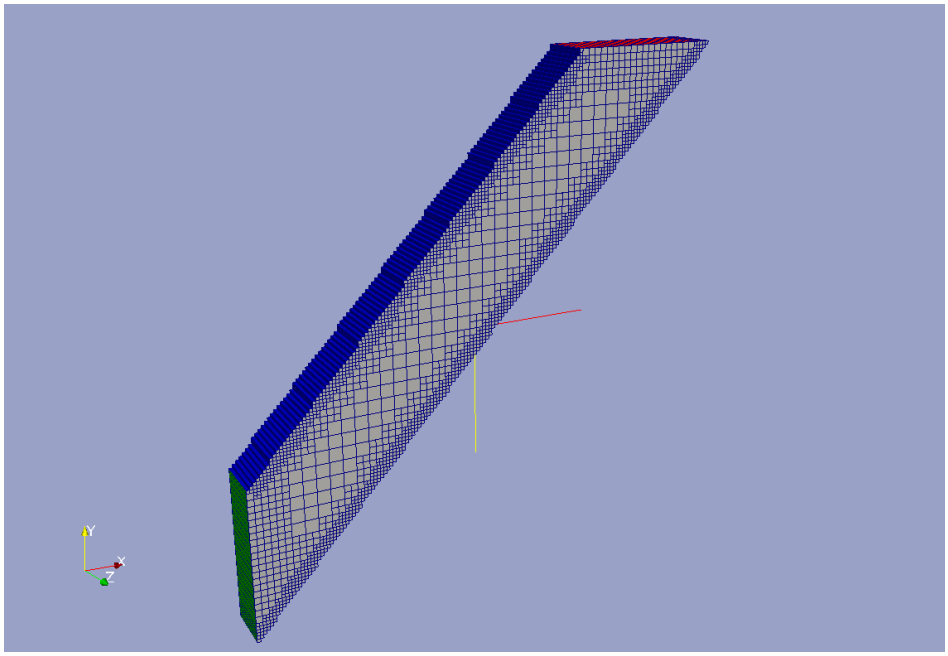


**Fig. 4.1** The Mesh from rotated BOX function

On Fig. 4.2A,B is shown the using all three functions. The corresponding Q1 operators are:

```
(initial of prps is 198 with infob_1)
(initial of prps is 198 with infob_2)
(initial of prps is 198 with infob_3)
real(th)
th=45/180*3.1416    !
(infob at .patch1 is box(0.35,0.35,0.35,0.3,0.3,0.3,0,0,:th:) with infob_1)
(infob at .patch2 is sphere(1.5,0.5,0.5,0.3) with infob_2)
(infob at .patch3 is ellpsd(2.5,0.5,0.5,0.10,0.2,0.3,0,th,0) with infob_3)

 PATCH(.PATCH1, VOLUME, 0, 1000, 0, 1000, 0, 1000, 1, 1)
 PATCH(.PATCH2, VOLUME, 0, 1000, 0, 1000, 0, 1000, 1, 1)
 PATCH(.PATCH3, VOLUME, 0, 1000, 0, 1000, 0, 1000, 1, 1)
```

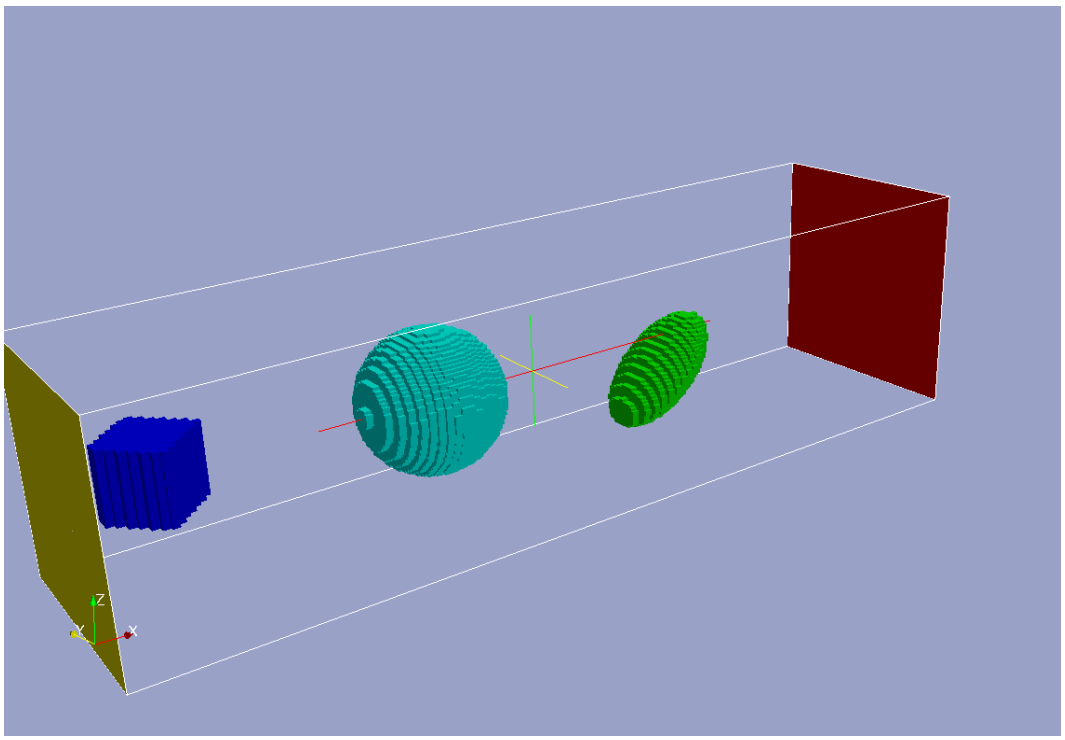


**Fig. 4.2A** Boundary Faces

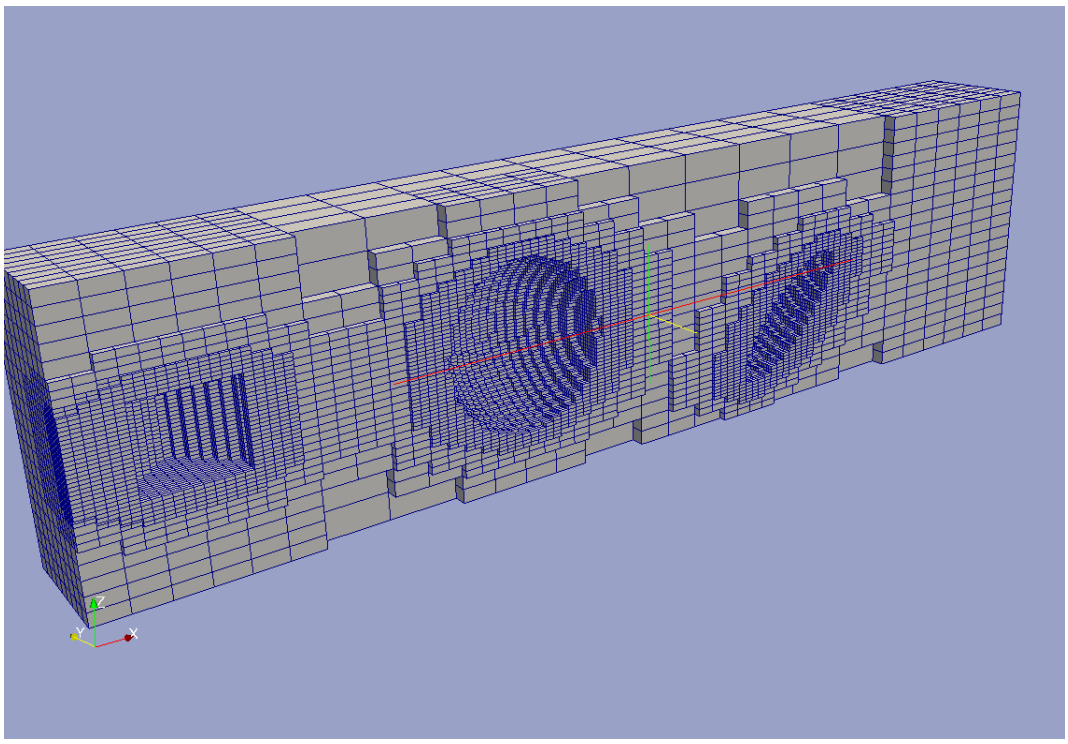


**Fig. 4.2B** The mesh

### *4.2. Smoothing Boundary Cells algorithm*

By default AGG uses the Whole-Cells algorithm (this algorithm is described above). The main idea of this method is:

1) After refinement process there are «boundary» Cut-Cell cells – these cells, which intersect the surface of objects (this cell contain intersection point one of rays along X, Y or Z, which passes through center of cell).
2) The processing of Cut-Cells is concluded in the setting their PRPS. For this is used two values PRPS_BEFORE and PRPS_AFTER of intersection point. If centre coordinate in ray direction is greater then intersection point coordinate, then is used PRPS_AFTER, else – PRPS_BEFORE.
3) The result: the surface of objects is approximated as “stair-step” surface (set of faces of whole cells).

In structured PHOENICS for exact approximation of the object surface is used algorithm of fractional cell - ParSol. This algorithm is very complex because this is connected with big amount of variants intersection surface with cell. An AGG boundary cell already has small size (the maximum level of the refinement), so for smoothing object surfaces possible to use the more simple approximate algorithms. SBC algorithm is such one.

The Main idea of SBC - using topologies unstructured Cartesian *Whole*-Cells:

1) Boundary Cut-Cells have hexahedral shape: all cells have 6 faces and 8 vertices;
2) The amount of cells don’t changes after using SBC algorithm: can’t to create two Cut-Cells from one Boundary cell;
3) The smoothing of Cut-Cell is made by moving of cell vertices to intersection points;
4) In current version SBC-AGG is used the simple moving algorithm (see bellow): this algorithm does *not use* analysis different variants of intersection surfaces with cell.
5) In accordance with (2) this method can be used only for Blockage objects with PRPS=198 or 199. So that for objects with other PRPS (for example, Fluid-Solid boundaries) is used old Whole-Cells algorithm.

SBC algorithm is described in detail below.

#### ***Scan stage by Ray-algorithms***

This stage is run the same as for Whole-Cell model. One difference – the scan is done not through center of Fine cells, but along their edges.
The amount of rays is increased on one ray in each direction in this case.

#### ***Stage of refinement process***

For refinement process is used the same algorithm as for Whole-Cell model. There is only one difference. The cell is Cut-Cell, if it has at least one intersection on one of its edges. This can bring about small difference in grids a long way from surface object for SBC and Whole-Cell algorithms.

#### ***Preparing SBC cells: Moving Vertices***

After completion refinement stage in tree cell remains the cells with level MaxLevel, having intersection on their own edges - Cut-Cells. For such cells recalculate the cross point with all its edges and is produced moving the vertices of the cells.

In current version is used simple algorithm of vertices moving:

1. Vertices of Cut-Cells are moved to their nearest intersection points;
2. No vertex may be moved **more than once**;
3. First search for and move vertices of “GOOD” cells which have **exactly four** intersections on edges parallel to X,Y or Z;

1) Move vertices of **not** “GOOD” CutCells in X,Y,Z direction along **edges** of cells;
2) **Remove** “BAD” cells of which **all neighbors** are either CutCells or have PRPS=198/199.

For moving is used tree-loop through all Cut-Cells.

The example of application this SBC algorithm is shown on Fig. 4.3 (the object – sphere, Coarse grid – 10*10*10, MaxLevel = 3)

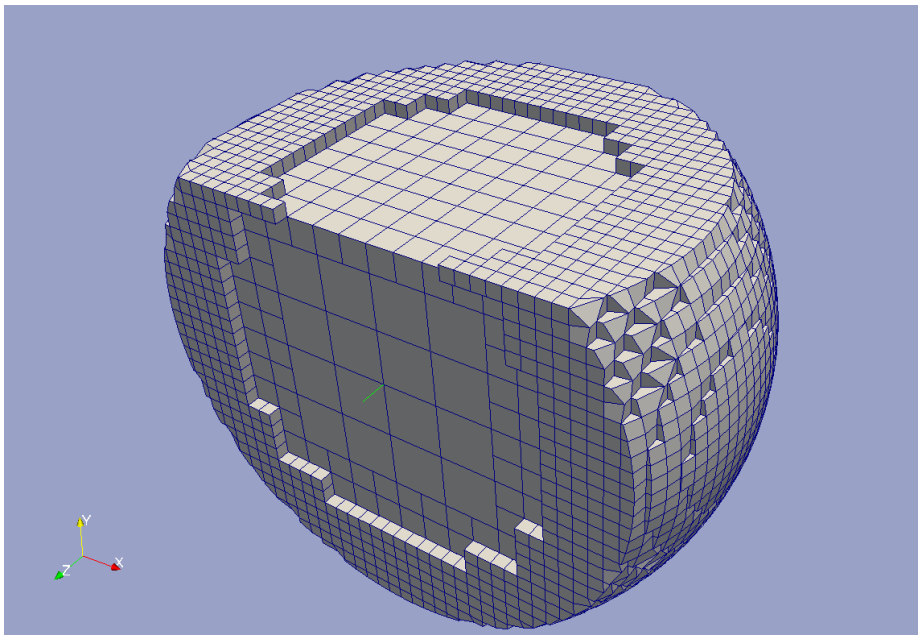
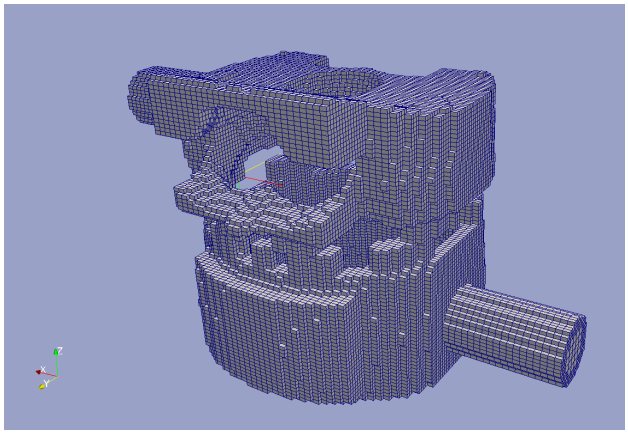


**Fig. 4.3** The sphere mesh

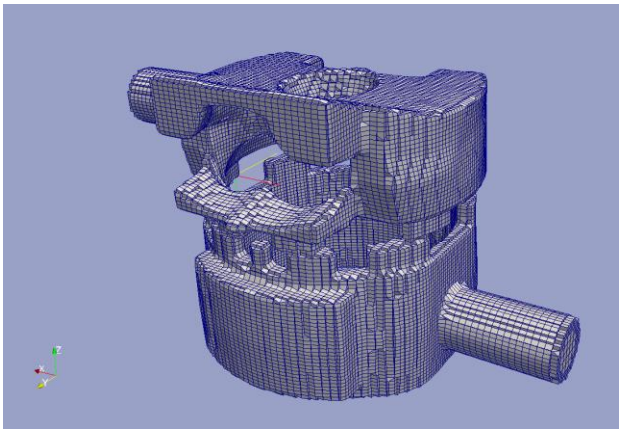
Thus, SBC algorithm not changed the structure of the main data objects. In addition is created array of the moved vertices CutNodes().

### *4.3. SBC examples*

Below is shown several examples of SBC algorithm.

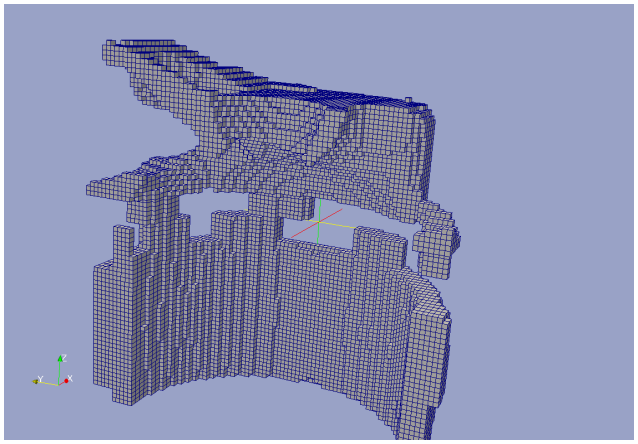


Whole-Cell model

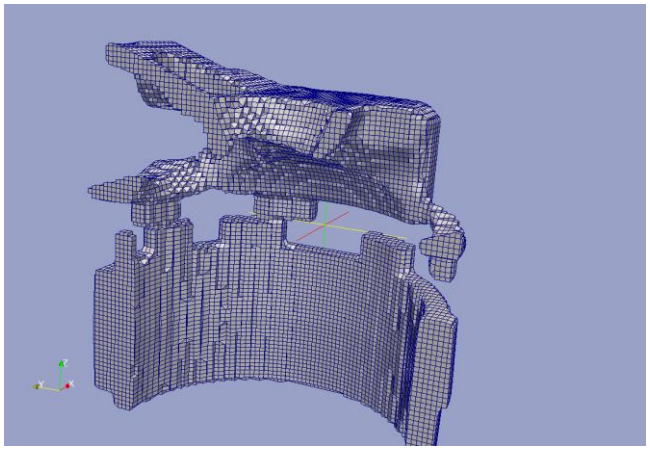


SBC model

**Fig. 4.4A** Complex Body: Boundary Faces
(Coarse Grid = 10*10*10, MaxLevel=3, NoCells=72 000)

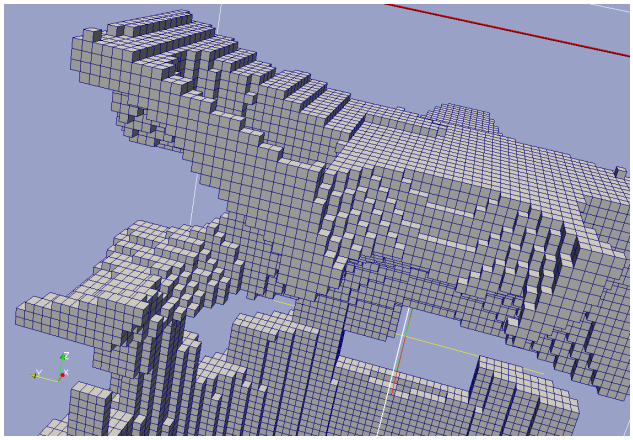


Whole-Cell model

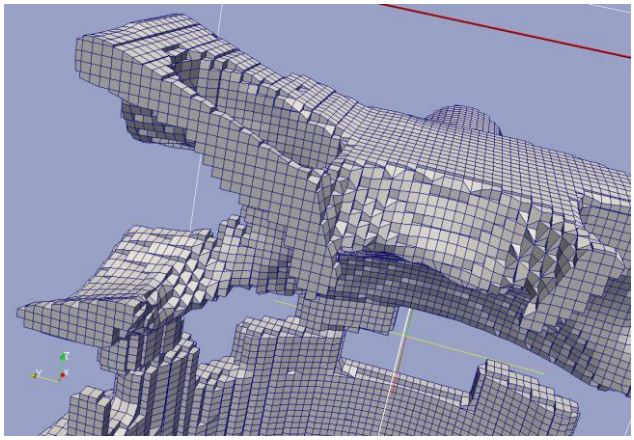


SBC model

**Fig. 4.4B** Complex Body: the Mesh 1 (total)

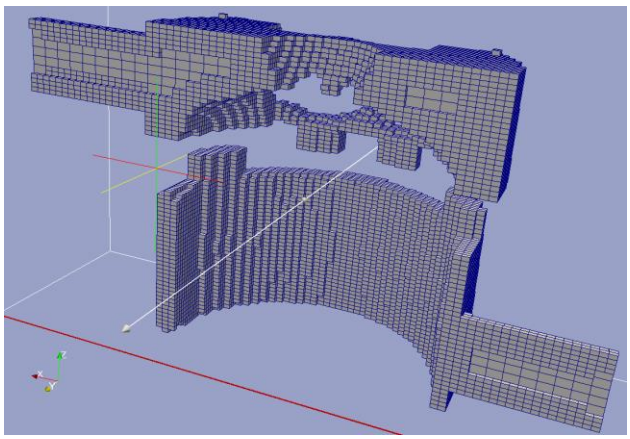


Whole-Cell model

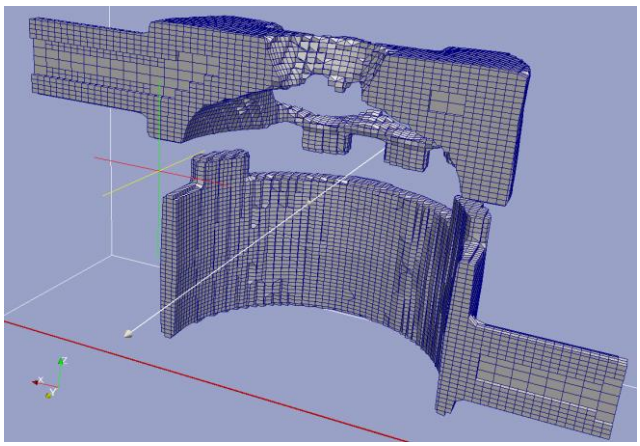


SBC model

**Fig. 4.4C** Complex Body: the Mesh 1 (detail)

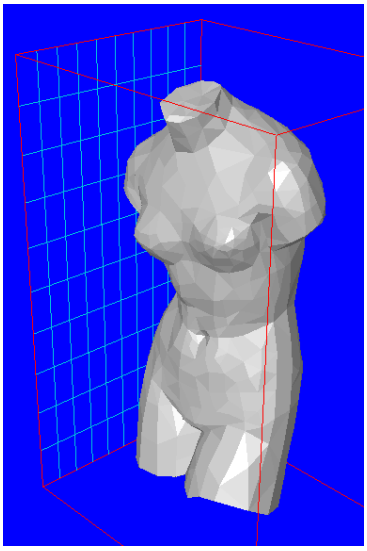


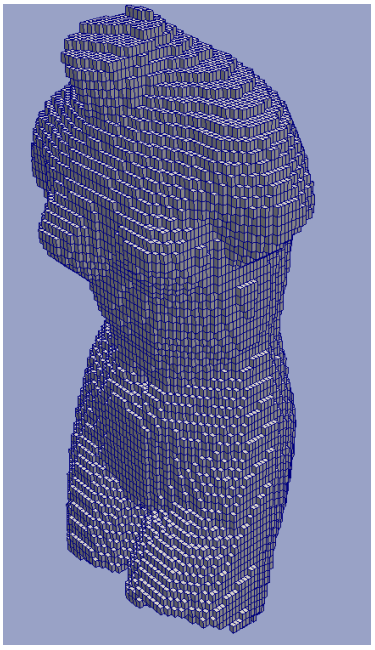
Whole-Cell model



SBC model

**Fig. 4.4D** Complex Body: the Mesh 2





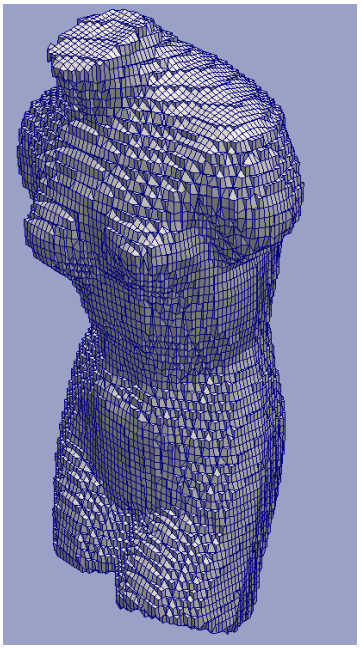


**Fig. 4.5** Human body: Boundary Faces

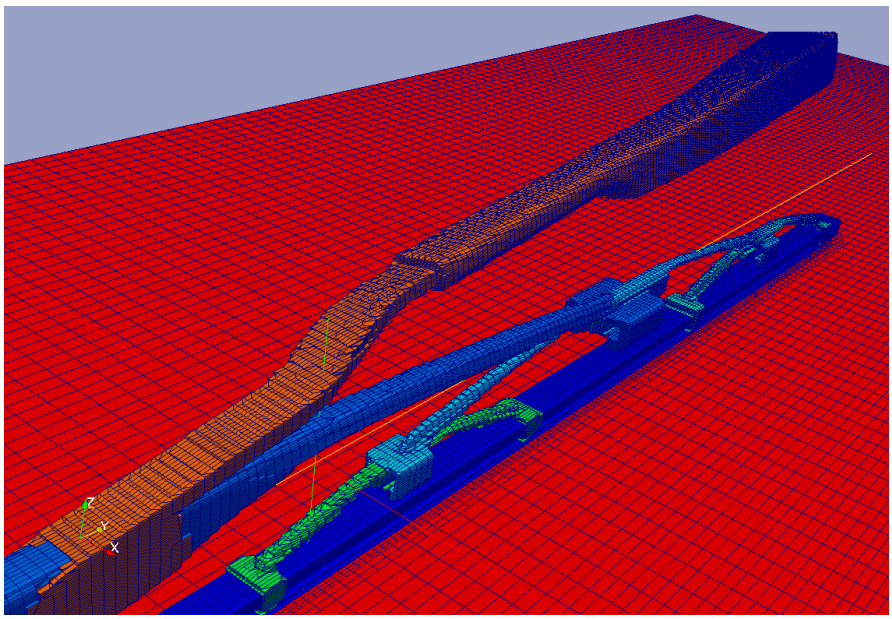


**Fig. 4.6** ASMO project: Boundary Faces

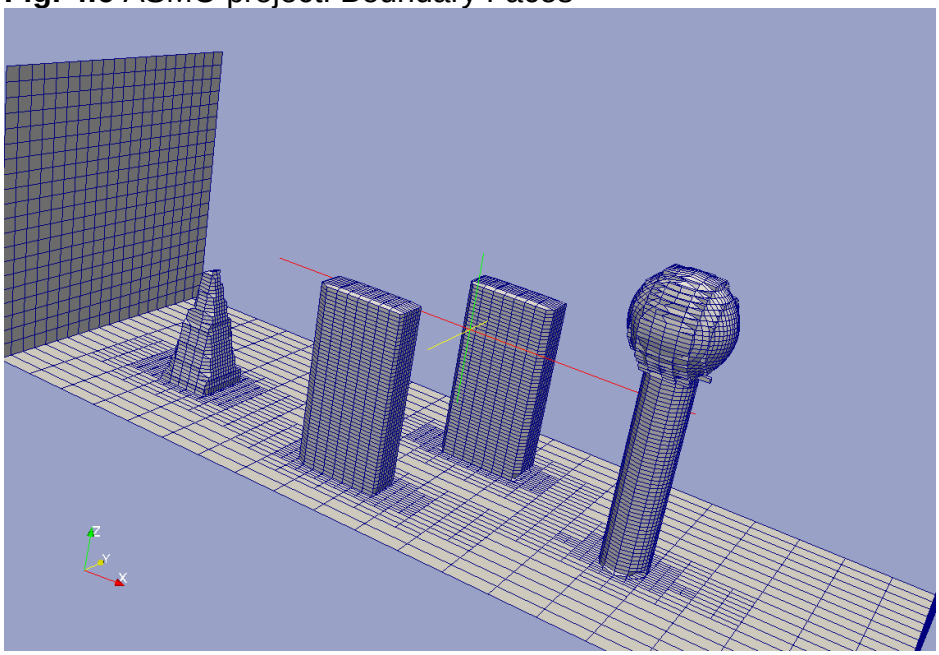


**Fig. 4.7** Model of the buildings: Boundary Faces