Rotating the mesh near the boundaries

Haiko Rijkers, *

November 30, 2011

Introduction

This document is to explain the function of the file.

This is a modification to the file import_cgns-grid.py. The original file had as function to read the CGNS files in which the grid and the boundary conditions are defined and write them to a more convenient form that can be used for the simulation of a turbo machine. The original file worked good for the original mesh but to improve the results a new mesh was created. The original file wasn't able to deal with the new O-mesh the modifications as described in this document have is function to resolve this problem.

*Student Mechanical Engineering. University of Twente, Enschede, the Netherlands

Contents

Part I
Project Description

The goal of this project is to modify the existing code for reading .CGNS files to make the code able to deal with more versatile types of meshes.

1 Present situation

In the next sections the original code is shortly described.

1.1 Code description

In short the present code reads the .CGNS files. The .CGNS includes block id numbers the locations of the nodes in x, y and z, the faces and the boundary conditions for every block. The code saves this information in a structure and writes every structure to a .vtk file. The .vtk file van later on be used to run the simulation.

2 Problem description

The previous code worked good for the original mesh but for a mesh in which the orientation of the blocks isn't constant this does not work. Since in the original code it is assumed that a certain boundary condition is connected to a certain face. For example it is assumed that the inlet block is always facing north. In the new mesh the blocks are not necessarily orientated in this manner. Here for the code has to be modified so that the boundary condition can be forced on the correct faces. If done so the rest of the existing code can be used to run the simulation.

3 Possible solution

There are multiple possible solutions to enforce the boundary condition on the correct faces. Possibilities are:

- Use the geometry of the entire block, take the outer nodes and match these with the nodes and faces to find the required reorientation.
- View the mesh using for example Paraview to find the correct orientation, using VRML
- Read the .CGNS file and reorientates using this information directly.

In this case is chosen to change the orientation of the mesh at the boundaries to the original orientation as it was defined in the original mesh so the boundary condition can be enforced, using the last method. After reorientation the original code can be used without further modifications.

Part II
Final solution

The final code first reads the .CGNS files as in the original file. the next step is to check if there are any boundary conditions on the block in which we are interested. The boundary conditions we are interested in are the periodic-, inlet- and outlet boundary conditions. If there are boundary conditions on the block of interest the next step is to check if reorientation is needed. reorientation is needed is one of the boundary conditions is on a face that does not correspond with the preferred orientation.

In this file there are three axis around which the block can rotate, the x' axis y'-axis and the z' axis, Figure 1. With these three axis of rotation it is possible to reorientates every block to the preferred orientation [1].

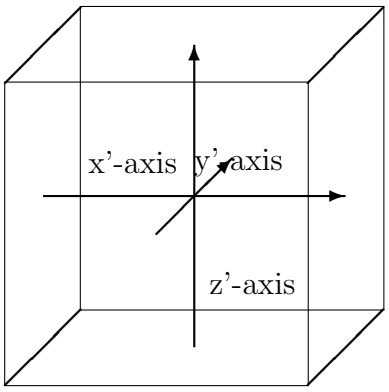

Figure 1: Block with the x'-axis and the z'-axis

4 Check orientation

The orientation is checked using the face names as saved in the .CGNS files and the boundary condition on each face. For a certain boundary condition is a certain face required. From the face and the boundary condition follows around which axis the block has to be rotated and how many rotations steps are required. The boundary conditions and the corresponding faces:

- BC: Inlet; Face: North
- BC: Outlet; Face: South
- BC: Periodic Pressure; Face: West

[1] In principle it is possible to reorientates the block with two axis

- BC: Periodic Suction; Face: East

5 Rotate elements

When the orientation is known the mesh can be rotated to the desired orientation. Here for both the indices and the face names have to be changed. In this case the rotation is done in multiple steps as will be described in the next section. The entire code is in Section VI.

5.1 Rotate indices and faces

In the section the steps that are needed for the rotations are described. In short the principle works as follows. When the block has to be orientated since there are boundary conditions on the block that are not corresponding with the correct faces the first step is to look if one of the periodic boundary conditions is on a a Top or Bottom face. If this is the case the block is rotates around the y'-axis, till the periodic boundary condition is on the correct face. The next step is to check if there are boundary conditions that are on a North, East, South or West face but not the correct one. The final step is to rotate the block around the x'-axis if there is an inlet or outlet boundary on a Top or Bottom face. After these three steps the faces and boundaries should always correspond.

5.2 Examples

In this section the reorientation as described in the previous section is explained using multiple examples. First the preferred orientation is given in Figure 2.

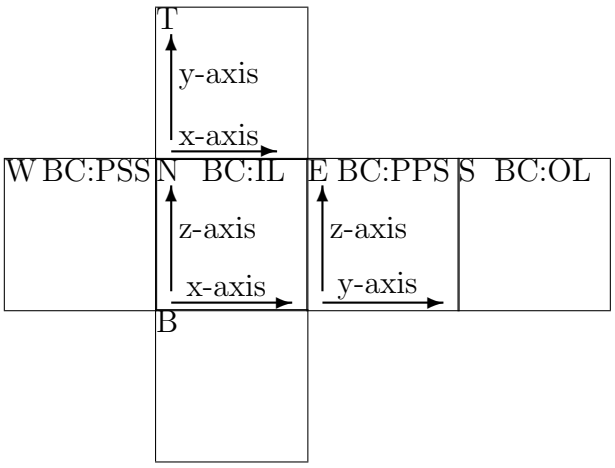

Figure 2: preferred orientation, with axis

This is the preferred orientation. Where the indices follow the axis, the i-indices the x-axis, the j-indices the y-axis and the k-indices the z-axis.

The boundary conditions are Periodic Suction Side (PSS), Inlet (IL), Periodic Pressure Side (PPS), Outlet (OL) for respectively West (W), North (N), East (E), South (S). There are no boundary conditions for the TOP face (T), nor for the BOTTOM face (B).

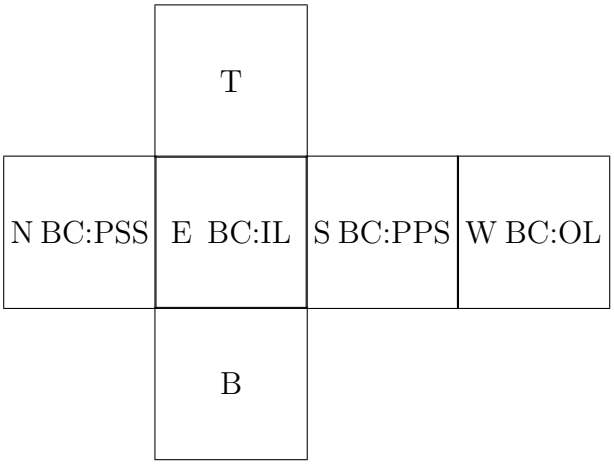

Figure 3: changed orientation 1

In this case the mesh has to be rotated once in the left direction, so a rotation around the z'-axis, Figure 3. This means that the indices and the faces have to be changed. Index operation:

- $[(nj - j - 1), i, k] = [i, j, k]$

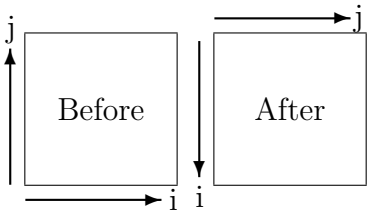

Figure 4: rotating around the z'-axis

Face operation:

- North = West; East = North; South = East; West = South;

In this case the mesh has to be rotated twice in the left direction, around the z'-axis, Figure 5. Index operation:

- $[(nj - j - 1), i, k] = [i, j, k]$
- And again: $[(nj - j - 1), i, k] = [i, j, k]$

Face operation:

- North = West; East = North; South = East; West = South;

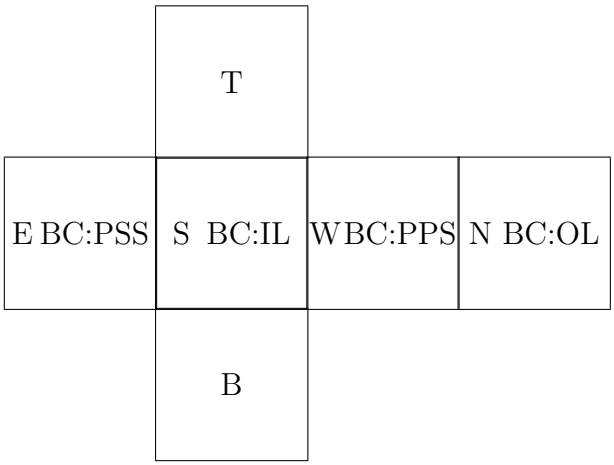

Figure 5: changed orientation 2

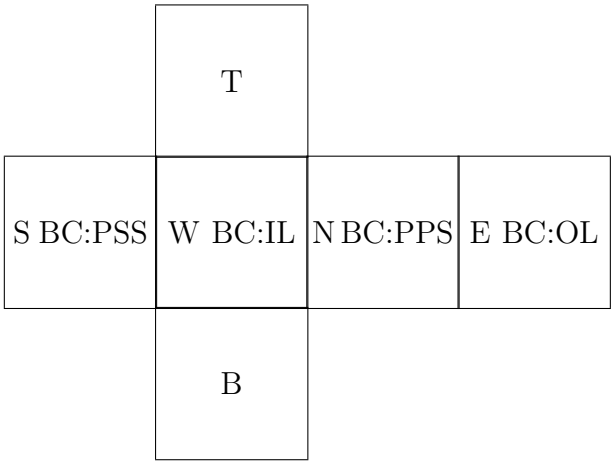

Figure 6: changed orientation 3

- And again: North = West; East = North; South = East; West = South;

In this case the mesh has to be rotated three times in the left direction, or once in the other direction, Figure 6. Since the code all ready exists for rotating to the left the choice fall on rotating three times to the left. Index operation:

- $[(nj - j - 1), i, k] = [i, j, k]$
- And again: $[(nj - j - 1), i, k] = [i, j, k]$
- And again: $[(nj - j - 1), i, k] = [i, j, k]$

Face operation:

- North = West; East = North; South = East; West = South;
- And again: North = West; East = North; South = East; West = South;

- And again: North = West; East = North; South = East; West = South;

It is as well possible the mesh has to be rotated around the y'- or x'-axis. In that case the mesh has first to be rotated once around the the y'- or x'-axis and afterwards the rotated as described earlier. In case there is an periodic boundary condition on the Top face or on the Bottom face the block is first rotated around the y'-axis.

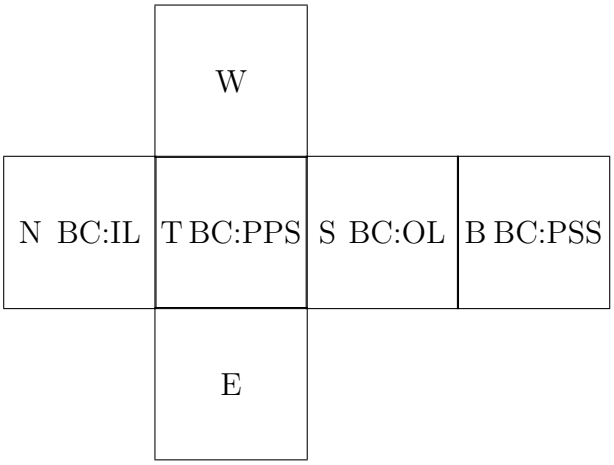

Figure 7: changed orientation 4

In this case the mesh has to be rotated once around the y'-axis, Figure 7. Index operation:

- $[(nk - k - 1), j, i] = [i, j, k]$

Face operation:

- Top = East; East = South; North = North; Bottom = West; West = Top; South = South;

It is as well possible that the inlet or outlet in at the Top or Bottom face, Figure 8. In this case the mesh has to be rotated around the x'-axis. Index operation: $[i, (nk - k - 1), j] = [i, j, k]$. Face operation:

- Top = South; East = East; North = Top; West = West; South = Bottom; Bottom = North;

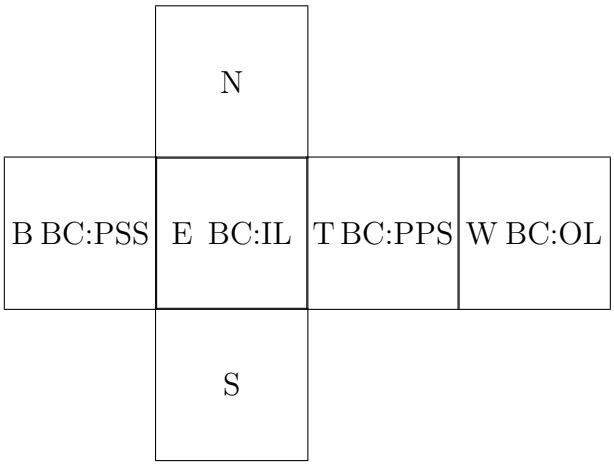

Figure 8: changed orientation 5

Part III
Project Testing

After the design the new code has to be tested. A first simple check is to check the geometry. Secondly the rest can be checked,the indices, the faces and size of the blocks.

6 Geometry

A simple first check is the geometry, If the geometry isn't correct any more after the transformations, the code is not correct. The geometry is the same as it was before the transformation as can be seen in the Figure 9, Figure 10 and Figure 11 from Paraview.

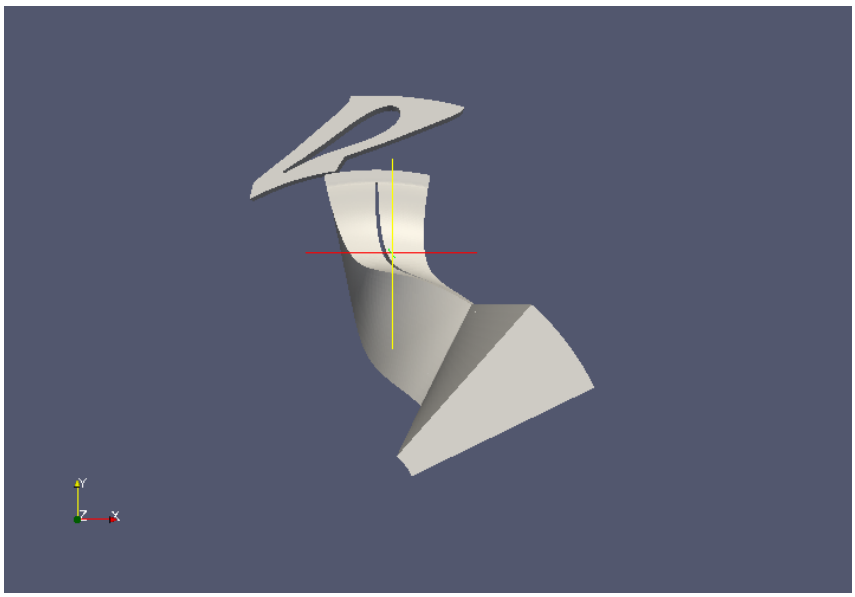

Figure 9: Screenshot of the Rotor and Stator from Paraview

7 Block sizes

The blocks on the periodic pressure (PPS) end periodic suction side (PSS) have to be of the same size to make it possible to match the data of both sides.

To check whether this is the case a first indication can be found using Paraview. If the blocks seem to match in Paraview. Secondly the exact coordinates can be checked using Paraview. It seems that they match.

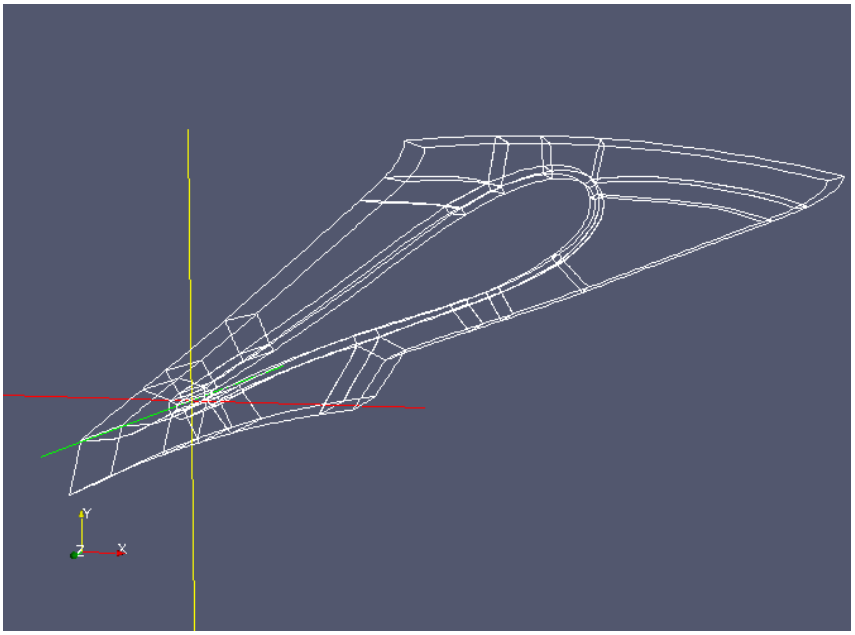

Figure 10: Screenshot of the Stator blocks from Paraview

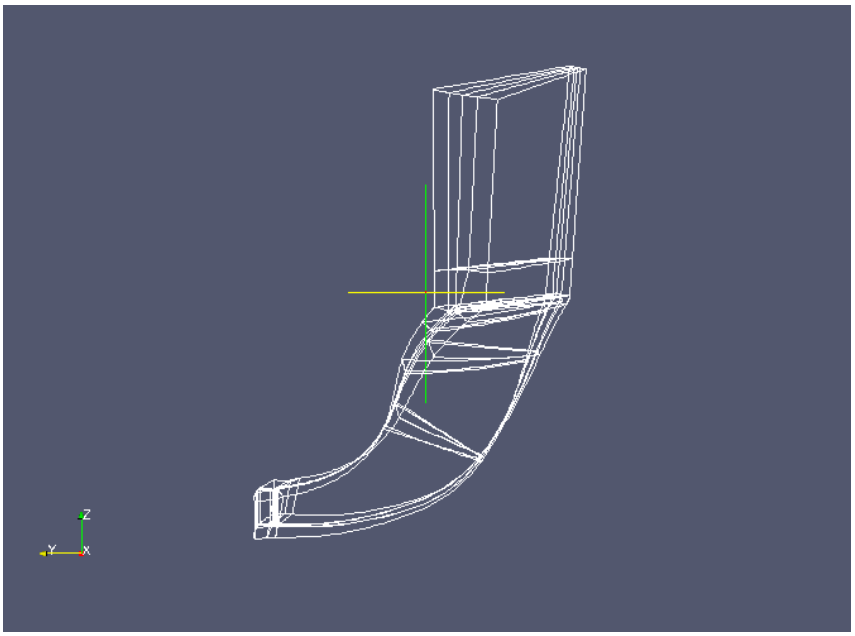

Figure 11: Screenshot of the Rotor blocks from Paraview

8 Faces

The faces should match the preferred orientation, boundaries and the indices. A first check is to check if the boundary conditions are facing the correct direction. The result is given in Section VII. As can be seen in this section the faces match with the boundaries.

9 Indices

First the indices can be checked by rotating the mesh 4 times around the same axis. The nodes should be on the same location again, as it turns out this part is working.Another way to check the indices is by using the VRML script as given in Section VIII. The result is given in Figure **??** and in Figure 13. The color red is for the periodic boundaries, green for the inlet boundary and blue for the outlet boundary As can be seen the colored faces are not all at the outer surfaces of the geometry this suggest that there is an mistake in the code. The faces are colored using the indices that correspond to a certain boundary.

Since the faces and boundaries correspond there is a mistake in rotating the indices. I assumed at first that the faces 0 and 2, 1 and 3 and the faces 4 and 5 would always be opposite. But after a check if this assumptions it turns out that this is not the case. I assume the mistake in rotating the indices is due to this false assumption.

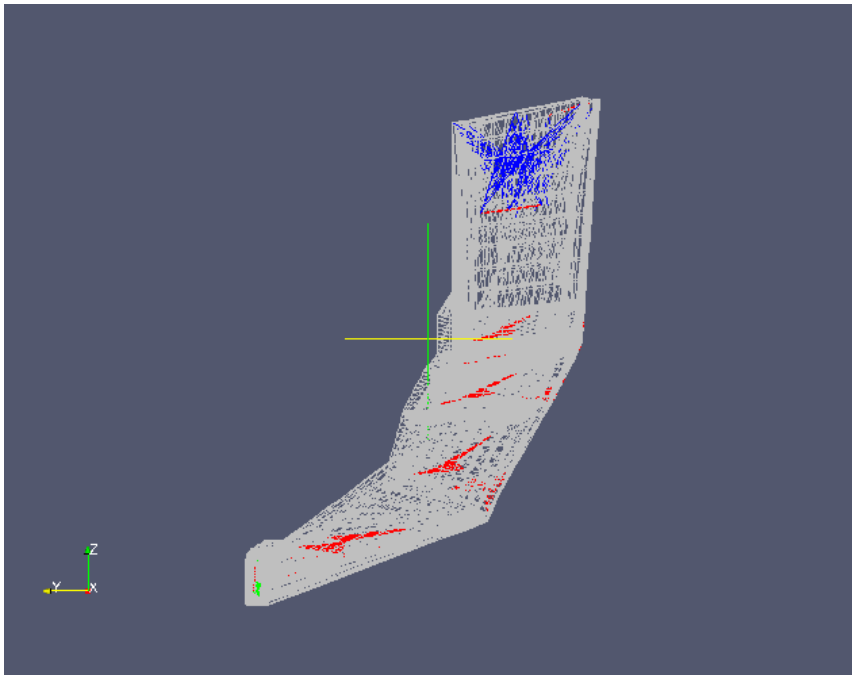

Figure 12: Screenshot 1 from Paraview using VRML

10 Running a simulation

The final test would be to run a simulation, but since the indices are incorrect this is not an option.

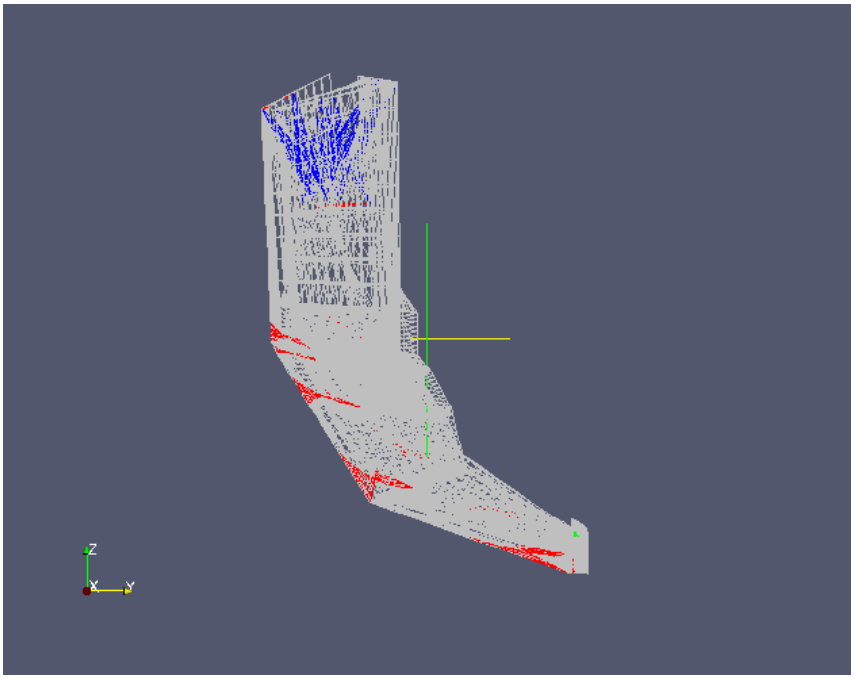

Figure 13: Screenshot 2 from Paraview using VRML

Part IV
mlab

Mlab is an option for 3D plotting in python a la Matlab. Personally I prefer this method above using Paraview, but I'm more used to working with matlab, than I'm with Paraview. Using mlab it's possible to combine different visualizations, this makes it for example possible to show at the same time a visualization of the pressure contours and a vector field of the velocities. As well to make a cut plane.

To use mlab there have to made a few modifications in the code. First of all mlab has to be imported from mayavi. Secondly for every visualization type there has to be written a script that gets the required data and creates the visualization. Finally the visualization has to be shown by using the command show. Example: % title: example

```
% variables x,y,z
from mayavi import mlab
s = mlab.mesh(x,y,z)
mlab.show()
```

Part V
Encountered problems

During this project I encountered a number of problems. I will try to give an short overview so the next interm can benefit from my experience.

- Since the code consist of a lot of different part it takes a while to find out how the different parts of the code are connected. I strongly advise the next person that starts working on this code first to make an overview of how the different parts are connected as I should have done when I started. On the longterm it will save time for you and for the next one after you.
- It seemed at first like the code was working perfectly, the geometrie matched with the original geometrie and the boundary conditions where facing in the correct direction. But using the VRML code to check the results it appeared not to be working as good as I thought.
- Finally it appeared that the faces where not ordered as I assumed. I thought that the faces 0 and 2, the faces 1 and 3 and the faces 4 and 5 where always opposite. But as it appears this is not the case.

Part VI
New file


```
"""
# import_cgns_grid.py
"""
Read structured grids from CGNS files.

Authors: Paul Petrie-Repar, QGECE, 15 Feb 2011
         Peter J, 23 Feb 2011
         Haiko Rijkers, 19 Nov 2011

This module makes use of Oliver Borm's CGNS package for Python 2.x
http://sourceforge.net/projects/python-cgns/

We have a copy of that package in cfcfd2/extern/python-cgns/.
"""

from CGNS import CGNS
import math
import sys, os
sys.path.append(os.path.expandvars("$HOME/e3bin"))
from e3_grid import StructuredGrid

def rotate_around_zaxis_periodic(BcPnt,face,g,g2, ni,nj, nk):
        """
        """
        g2.x = {}
        g2.y = {}
        g2.z = {}
        for k in range(nk):
                for j in range(nj):
                        for i in range(ni):
                                #indx = k*nj*ni + j*ni + i
                                # (1/2+(-1)^rotations*1/2)*(-1)*
                                g2.x[i,(nk-k-1),j] = g.x[i,j,k]
                                g2.y[i,(nk-k-1),j] = g.y[i,j,k]
                                g2.z[i,(nk-k-1),j] = g.z[i,j,k]
        nja=nj
        nj=nk
        nk=nja

        #print " rotations, x, y, z: ",g2.x[ni-1,nj-1,nk-1], g2.y[ni-1,nj-1,nk-1], g2.z[ni-1,nj-1,nk-1]
        g.x = g2.x
        g.y = g2.y
        g.z = g2.z
        g2.x = {}
        g2.y = {}
        g2.z = {}
        # The faces
        for face in range(6):
                if (BcPnt[face]=="TOP"):
                        BcPnttemp0="EAST"
                        temp0=face
                elif (BcPnt[face]=="EAST"):
                        BcPnttemp1="BOTTOM"
                        temp1=face
                elif (BcPnt[face]=="BOTTOM"):
                        BcPnttemp2="WEST"
                        temp2=face
                elif (BcPnt[face]=="WEST"):
                        BcPnttemp3="TOP"
                        temp3=face

        BcPnt[temp0]=BcPnttemp0
        BcPnt[temp1]=BcPnttemp1
        BcPnt[temp2]=BcPnttemp2
        BcPnt[temp3]=BcPnttemp3
        return BcPnt,g, ni,nj, nk

def rotate_around_zaxis_in_out_let(BcPnt,face,g,g2, ni,nj, nk):
        """
        """
        g2.x = {}
        g2.y = {}
```

```
        g2.z = {}
        for k in range(nk):
                for j in range(nj):
                        for i in range(ni):
                                #indx = k*nj*ni + j*ni + i
                                # (1/2+(-1)^rotations*1/2)*(-1)*
                                g2.x[(nk-k-1),j,i] = g.x[i,j,k]
                                g2.y[(nk-k-1),j,i] = g.y[i,j,k]
                                g2.z[(nk-k-1),j,i] = g.z[i,j,k]
        nia=ni
        ni=nk
        nk=nia

        #print " rotations, x, y, z: ",g2.x[ni-1,nj-1,nk-1], g2.y[ni-1,nj-1,nk-1], g2.z[ni-1,nj-1,nk-1]
        g.x = g2.x
        g.y = g2.y
        g.z = g2.z
        g2.x = {}
        g2.y = {}
        g2.z = {}
        # The faces
        for face in range(6):
                if (BcPnt[face]=="TOP"):
                        BcPnttemp0="NORTH"
                        temp0=face
                elif (BcPnt[face]=="NORTH"):
                        BcPnttemp1="BOTTOM"
                        temp1=face
                elif (BcPnt[face]=="BOTTOM"):
                        BcPnttemp2="SOUTH"
                        temp2=face
                elif (BcPnt[face]=="SOUTH"):
                        BcPnttemp3="TOP"
                        temp3=face

        BcPnt[temp0]=BcPnttemp0
        BcPnt[temp1]=BcPnttemp1
        BcPnt[temp2]=BcPnttemp2
        BcPnt[temp3]=BcPnttemp3
        return BcPnt,g, ni,nj, nk


def check_CGNS_error_flag(errorFlag, messageText=""):
    """
    Give some information on CGNS failure.
    """
    if (errorFlag != 0):
        print "CGNS Error %d %s" % (errorFlag, messageText)
        CGNS.cg_error_exit()
    return


def getFaceName(face, bcPnts):
    """
    Returns the face name.
    """
    if (bcPnts[0] == bcPnts[3]):
        if (bcPnts[0] == 1):
            return "WEST"
            # Periodic BC 1
        else:
            return "EAST"
            # Periodic BC 2
    elif (bcPnts[1] == bcPnts[4]):
        if (bcPnts[1] == 1):
            return "SOUTH"
        else:
            return "NORTH"
    elif (bcPnts[2] == bcPnts[5]):
        if (bcPnts[2] == 1):
            return "BOTTOM"
        else:
```

```
            return "TOP"
    else:
        print "Cannot determine Face"

    #print "this is BcPnts: ", BcPnts



def read_ICEM_CGNS_grids(cgnsFileName,  labelStem="test" , gridScale=1.0):
    """
    Dip into the CGNS file and extract the structured grids from within.

    Returns a dictionary containing the interesting data.
    """
    labelStem = cgnsFileName

    cgnsData = dict() # a convenient place to collect everything

    print "open CGNS file: ", cgnsFileName, " scale: ", gridScale
    filePtr = CGNS.intp()
    ier = CGNS.cg_open(cgnsFileName, CGNS.CG_MODE_READ, filePtr)
    fileValue = filePtr.value()
    check_CGNS_error_flag(ier, "on opening file")

    basePtr = CGNS.intp()
    basePtr.assign(1)
    numberZonesPtr = CGNS.intp()
    ier = CGNS.cg_nzones(filePtr.value(), basePtr.value(), numberZonesPtr)
    check_CGNS_error_flag(ier, "on getting number of zones")
    numberZones = numberZonesPtr.value()
    cgnsData['nblock'] = numberZones
    cgnsData['grids'] = []
    cgnsData['bcs'] = []
    #print "cgnsData=", cgnsData

    zonePtr = CGNS.intp()
    zoneSizeArray = CGNS.intArray(9)
    range_min = CGNS.intArray(3)
    range_max = CGNS.intArray(3)

    faceList3D = ["NORTH", "EAST", "SOUTH", "WEST", "TOP", "BOTTOM"]


    for zone in range(numberZones):

        zonePtr.assign(zone+1)
        ier, zoneName = CGNS.cg_zone_read(filePtr.value(), basePtr.value(), zonePtr.value(),
                                          zoneSizeArray)
        check_CGNS_error_flag(ier, "on getting zone pointer")
        #
        ni = zoneSizeArray[0]; nj = zoneSizeArray[1]; nk = zoneSizeArray[2]
        #  print "ni=", ni, "nj=", nj, "nk=", nk

        range_min[0] = 1; range_max[0] = ni
        range_min[1] = 1; range_max[1] = nj
        range_min[2] = 1; range_max[2] = nk
        # Read the coordinate arrays.
        numberNodes = ni * nj * nk
        coordX = CGNS.doubleArray(numberNodes)
        coordY = CGNS.doubleArray(numberNodes)
        coordZ = CGNS.doubleArray(numberNodes)

        ier, coordName = CGNS.cg_coord_read(filePtr.value(), basePtr.value(), zonePtr.value(),
                                            "CoordinateX", CGNS.RealDouble,
                                            range_min, range_max, coordX)
        check_CGNS_error_flag(ier, "on getting x-coordinates")
        ier, coordName = CGNS.cg_coord_read(filePtr.value(), basePtr.value(), zonePtr.value(),
                                            "CoordinateY", CGNS.RealDouble,
                                            range_min, range_max, coordY)
        check_CGNS_error_flag(ier, "on getting y-coordinates")
        ier, coordName = CGNS.cg_coord_read(filePtr.value(), basePtr.value(), zonePtr.value(),
```

```
                                        "CoordinateZ", CGNS.RealDouble,
                                        range_min, range_max, coordZ)
        check_CGNS_error_flag(ier, "on getting z-coordinates")
        # Now that we've read the coordinates, repack them into Eilmer's data structure.
        g = StructuredGrid((ni,nj,nk), label='%s%04d'%(labelStem, zone))
        g2 = StructuredGrid((nj,ni,nk), label='%s%04d'%(labelStem, zone))
        g3 = StructuredGrid((ni,nj,nk), label='%s%04d'%(labelStem, zone))

                # to make sure the grid is only rotated once, after the rotation is set to zero
        bz=1

        for k in range(nk):
            for j in range(nj):
                for i in range(ni):
                    indx = k*nj*ni + j*ni + i
                    g.x[i,j,k] = coordX[indx] * gridScale
                    g.y[i,j,k] = coordY[indx] * gridScale
                    g.z[i,j,k] = coordZ[indx] * gridScale

        # Changes to yes if there is one.
        Is_there_a_BC_on_the_block="no"

        BcPnt = {}
        familyName = {}
        for face in range(6):
            ier = CGNS.cg_goto(fileValue, basePtr.value(), "Zone_t", (zone + 1), "ZoneBC_t", 1,
"BC_t", (face+1), "end")
            ier, familyName[face] = CGNS.cg_famname_read()

            print "familyName: ", familyName[face]

            bcPntsPointer = CGNS.intArray(6)
            normalListPointer = CGNS.intArray(3)
            CGNS.cg_boco_read(fileValue, basePtr.value(), (zone+1), (face+1), bcPntsPointer,
normalListPointer);

            # If rotate_top_bottom is zero do nothing.
            rotate_top_bottom=0
            bc = {}

            BcPnt[face] = getFaceName(face, bcPntsPointer)
            #print BcPnt[face]
            print "Before rotation: ", zone, face, familyName[face], BcPnt[face]


        nr_rotations=0
        for face in range(6):

                        if (familyName[face].find("OUTLET")>=0 or familyName[face].find
( "PERIODIC_SUCTION")>=0 or familyName[face].find("PERIODIC_PRESSURE")>=0 or familyName[face].find
("INLET")>=0):
                                # One of the face has a BC, so reorientation is needed
                                Is_there_a_BC_on_the_block="yes"
                                #print BcPnt[face]

                                if (BcPnt[face]=="TOP" or BcPnt[face]=="BOTTOM"):
                                        if (familyName[face].find( "PERIODIC_SUCTION")>=0):
                                                nr_rotationsz=1
                                                BcPnt,g, ni,nj, nk=rotate_around_zaxis_periodic
(nr_rotationsz,familyName[face],g,g2, ni,nj, nk)
                                        elif (familyName[face].find("PERIODIC_PRESSURE")>=0):
                                                nr_rotationsz=1
                                                BcPnt,g, ni,nj, nk=rotate_around_zaxis_periodic
(nr_rotationsz,familyName[face],g,g2, ni,nj, nk)


                                if (BcPnt[face]=="NORTH"):#INLET
                                        if (familyName[face].find("PERIODIC_PRESSURE")>=0):
                                                nr_rotations=3
                                        elif (familyName[face].find( "PERIODIC_SUCTION")>=0):
                                                nr_rotations=1
                                        elif (familyName[face].find("OUTLET")>=0):
```

```
                                        nr_rotations=2
                                elif (familyName[face].find("INLET")>=0):
                                        nr_rotations=0

                        if (BcPnt[face]=="EAST"):#PERIODIC_SUCTION

                                if (familyName[face].find("PERIODIC_PRESSURE")>=0):
                                        nr_rotations=2
                                elif (familyName[face].find( "PERIODIC_SUCTION")>=0):
                                        nr_rotations=0
                                elif (familyName[face].find("OUTLET")>=0):
                                        nr_rotations=1
                                elif (familyName[face].find("INLET")>=0):
                                        nr_rotations=3

                        if (BcPnt[face]=="SOUTH"):#OUTLET

                                if (familyName[face].find("PERIODIC_PRESSURE")>=0):
                                        nr_rotations=1
                                elif (familyName[face].find( "PERIODIC_SUCTION")>=0):
                                        nr_rotations=3
                                elif (familyName[face].find("OUTLET")>=0):
                                        nr_rotations=0
                                elif (familyName[face].find("INLET")>=0):
                                        nr_rotations=2

                        if (BcPnt[face]=="WEST"):#PERIODIC_PRESSURE

                                if (familyName[face].find("PERIODIC_PRESSURE")>=0):
                                        nr_rotations=0
                                elif (familyName[face].find( "PERIODIC_SUCTION")>=0):
                                        nr_rotations=2
                                elif (familyName[face].find("OUTLET")>=0):
                                        nr_rotations=3
                                elif (familyName[face].find("INLET")>=0):
                                        nr_rotations=1


                        #print "Normal rotations, #of rotations: ", nr_rotations,"\n"

        rotations=0
        while (rotations < nr_rotations):

                        g2.x = {}
                        g2.y = {}
                        g2.z = {}
                        for k in range(nk):
                                for j in range(nj):
                                        for i in range(ni):
                                                #indx = k*nj*ni + j*ni + i

                                                # (1/2+(-1)^rotations*1/2)*(-1)*
                                                g2.x[(nj-j-1),i,k] = g.x[i,j,k]
                                                g2.y[(nj-j-1),i,k] = g.y[i,j,k]
                                                g2.z[(nj-j-1),i,k] = g.z[i,j,k]
                        nia=nj
                        nj=ni
                        ni=nia
                        # Number of rotations
                        rotations=rotations+1
                        print "#",rotations, " rotations, x, y, z: ",g2.x[ni-1,nj-1,nk-1], g2.y
[ni-1,nj-1,nk-1], g2.z[ni-1,nj-1,nk-1]
                        g.x = g2.x
                        g.y = g2.y
                        g.z = g2.z
                        g2.x = {}
                        g2.y = {}
                        g2.z = {}
                        # The faces
                        for face in range(6):
                                if (BcPnt[face]=="NORTH"):
                                        BcPnttemp0="EAST"
```

```
                                temp0=face
                            elif (BcPnt[face]=="EAST"):
                                BcPnttemp1="SOUTH"
                                temp1=face
                            elif (BcPnt[face]=="SOUTH"):
                                BcPnttemp2="WEST"
                                temp2=face
                            elif (BcPnt[face]=="WEST"):
                                BcPnttemp3="NORTH"
                                temp3=face

                        BcPnt[temp0]=BcPnttemp0
                        BcPnt[temp1]=BcPnttemp1
                        BcPnt[temp2]=BcPnttemp2
                        BcPnt[temp3]=BcPnttemp3
        for face in range(6):

                        if (familyName[face].find("OUTLET")>=0 or familyName[face].find
( "PERIODIC_SUCTION")>=0 or familyName[face].find("PERIODIC_PRESSURE")>=0 or familyName[face].find
("INLET")>=0):
                            # One of the face has a BC, so reorientation is needed
                            Is_there_a_BC_on_the_block="yes"
                            #print BcPnt[face]

                            if (BcPnt[face]=="BOTTOM" or BcPnt[face]=="TOP"):
                                if (familyName[face].find("INLET")>=0):
                                    nr_rotations=1
                                    BcPnt,g, ni,nj, nk=rotate_around_zaxis_in_out_let
(BcPnt,face,g,g2, ni,nj, nk)
                                elif (familyName[face].find("OUTLET")>=0):
                                    nr_rotations=1
                                    BcPnt,g, ni,nj, nk=rotate_around_zaxis_in_out_let
(BcPnt,face,g,g2, ni,nj, nk)


        for face in range(6):

            print "Result: ", zone, face, familyName[face], BcPnt[face]

            bc = {}
            bc['block'] = zone
            bc['face'] = BcPnt[face]
            # do not add internal and wall boundaries

            familyName[face].capitalize()
            n=0

            if (familyName[face].find("OUTLET") >= 0):
                bc['type'] = "outlet"
                # cgnsData['bcs'].append(bc)
            elif (familyName[face].find("INLET") >= 0):
                bc['type'] = "inlet"
                # cgnsData['bcs'].append(bc)
            elif (familyName[face].find("PERIODIC") >= 0):
                bc['type'] = "periodic"
                # cgnsData['bcs'].append(bc)
            elif (familyName[face].find("BLADE") >= 0):
                bc['type'] = "wall"
                # cgnsData['bcs'].append(bc)
            elif (familyName[face].find("SHROUD") >= 0):
                bc['type'] = "wall"
                # cgnsData['bcs'].append(bc)
            elif (familyName[face].find("HUB") >= 0):
                bc['type'] = "wall"
                # cgnsData['bcs'].append(bc)
            elif (familyName[face].find("ORPHAN") >= 0):
                bc['type'] = "internal"
                # do not add
            else:
                #cgnsData['bcs'].append(bc)
                bc['type'] = familyName[face]
                print "WARNING: Unknown boundary condition: " , familyName[face]
```

```
            # resetting the number of elements
            g.ni=ni
            g.nj=nj
            g.nk=nk

            # Append CGNS Data
            cgnsData['bcs'].append(bc)
        cgnsData['grids'].append(g)

                        # print cgnsData['bcs']
                        # print "nodes: ", ni, nj, nk, "blockn ", zone
                # End for zone...
    return cgnsData


if __name__ == '__main__':
        # Main code
    print "cgns_import.py Demonstration..."
    if len(sys.argv) < 2:
        print "Usage: python import_cgns_grid.py CGNS_filename"
        sys.exit()
    fileName = sys.argv[1]
    print "CGNS file name: ", fileName
    # Calling the function that reads the CGNS files, and reorientates the grid
    dataDict = read_ICEM_CGNS_grids(fileName)
    nb = dataDict['nblock']
    print "number of blocks found=", nb
    print "start writing the grid"
    for jb in range(nb):
                # Writing the VTK files with the grid
                g = dataDict['grids'][jb]
                print '   ', g.label, 'ni=', g.ni, 'nj=', g.nj, 'nk=', g.nk, "blockn: ", jb
                f = open(g.label + ".vtk", 'w')
                g.write_block_in_VTK_format(f)
                f.close()
    print "Done."

"""
```

Part VII
output file

BC: periodic
0 0 , BC type: periodic , BC face: EAST
0 1 , BC type: wall , BC face: TOP
0 2 , BC type: wall , BC face: BOTTOM
0 3 , BC type: internal , BC face: NORTH
4 , BC: inlet
0 4 , BC type: inlet , BC face: SOUTH
0 5 , BC type: internal , BC face: WEST
VRML line for volume, block: 1
0 , BC: periodic
1 0 , BC type: periodic , BC face: EAST
1 1 , BC type: wall , BC face: NORTH
1 2 , BC type: internal , BC face: BOTTOM
1 3 , BC type: internal , BC face: WEST
1 4 , BC type: wall , BC face: SOUTH
1 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 2
0 , BC: periodic
2 0 , BC type: periodic , BC face: EAST
2 1 , BC type: wall , BC face: NORTH
2 2 , BC type: internal , BC face: BOTTOM
2 3 , BC type: internal , BC face: WEST
2 4 , BC type: wall , BC face: SOUTH
2 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 3
3 0 , BC type: wall , BC face: BOTTOM
3 1 , BC type: internal , BC face: NORTH
3 2 , BC type: internal , BC face: WEST
3 3 , BC type: wall , BC face: TOP
4 , BC: inlet
3 4 , BC type: inlet , BC face: SOUTH
3 5 , BC type: internal , BC face: EAST
VRML line for volume, block: 4
4 0 , BC type: internal , BC face: BOTTOM
4 1 , BC type: internal , BC face: WEST

4 2 , BC type: wall , BC face: SOUTH
4 3 , BC type: internal , BC face: EAST
4 4 , BC type: wall , BC face: NORTH
4 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 5
0 , BC: periodic
5 0 , BC type: periodic , BC face: WEST
5 1 , BC type: wall , BC face: BOTTOM
2 , BC: inlet
5 2 , BC type: inlet , BC face: SOUTH
5 3 , BC type: wall , BC face: TOP
5 4 , BC type: internal , BC face: NORTH
5 5 , BC type: internal , BC face: EAST
VRML line for volume, block: 6
6 0 , BC type: internal , BC face: EAST
1 , BC: periodic
6 1 , BC type: periodic , BC face: WEST
6 2 , BC type: wall , BC face: NORTH
6 3 , BC type: internal , BC face: BOTTOM
6 4 , BC type: wall , BC face: SOUTH
6 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 7
7 0 , BC type: internal , BC face: EAST
1 , BC: periodic
7 1 , BC type: periodic , BC face: WEST
7 2 , BC type: wall , BC face: NORTH
7 3 , BC type: internal , BC face: BOTTOM
7 4 , BC type: wall , BC face: SOUTH
7 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 8
8 0 , BC type: internal , BC face: SOUTH
8 1 , BC type: wall , BC face: WEST
8 2 , BC type: internal , BC face: BOTTOM
8 3 , BC type: wall , BC face: TOP
8 4 , BC type: wall , BC face: EAST
8 5 , BC type: internal , BC face: NORTH
VRML line for volume, block: 9
9 0 , BC type: wall , BC face: WEST
9 1 , BC type: internal , BC face: TOP

9 2 , BC type: internal , BC face: SOUTH
9 3 , BC type: wall , BC face: BOTTOM
9 4 , BC type: wall , BC face: EAST
9 5 , BC type: internal , BC face: NORTH
VRML line for volume, block: 10
10 0 , BC type: internal , BC face: WEST
10 1 , BC type: wall , BC face: SOUTH
10 2 , BC type: internal , BC face: BOTTOM
10 3 , BC type: wall , BC face: TOP
10 4 , BC type: internal , BC face: EAST
10 5 , BC type: wall , BC face: NORTH
VRML line for volume, block: 11
11 0 , BC type: wall , BC face: SOUTH
11 1 , BC type: internal , BC face: TOP
11 2 , BC type: internal , BC face: WEST
11 3 , BC type: wall , BC face: BOTTOM
11 4 , BC type: internal , BC face: EAST
11 5 , BC type: wall , BC face: NORTH
VRML line for volume, block: 12
0 , BC: periodic
12 0 , BC type: periodic , BC face: EAST
12 1 , BC type: wall , BC face: NORTH
12 2 , BC type: internal , BC face: BOTTOM
12 3 , BC type: internal , BC face: WEST
12 4 , BC type: wall , BC face: SOUTH
12 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 13
13 0 , BC type: wall , BC face: WEST
13 1 , BC type: internal , BC face: BOTTOM
13 2 , BC type: internal , BC face: SOUTH
13 3 , BC type: wall , BC face: TOP
13 4 , BC type: wall , BC face: EAST
13 5 , BC type: internal , BC face: NORTH
VRML line for volume, block: 14
14 0 , BC type: internal , BC face: EAST
14 1 , BC type: wall , BC face: NORTH
14 2 , BC type: internal , BC face: BOTTOM
3 , BC: periodic
14 3 , BC type: periodic , BC face: WEST

14 4 , BC type: wall , BC face: SOUTH
14 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 15
15 0 , BC type: wall , BC face: WEST
15 1 , BC type: internal , BC face: TOP
15 2 , BC type: internal , BC face: SOUTH
15 3 , BC type: wall , BC face: BOTTOM
15 4 , BC type: wall , BC face: EAST
15 5 , BC type: internal , BC face: NORTH
VRML line for volume, block: 16
16 0 , BC type: internal , BC face: BOTTOM
1 , BC: periodic
16 1 , BC type: periodic , BC face: EAST
16 2 , BC type: wall , BC face: NORTH
16 3 , BC type: internal , BC face: TOP
16 4 , BC type: internal , BC face: WEST
16 5 , BC type: wall , BC face: SOUTH
VRML line for volume, block: 17
17 0 , BC type: internal , BC face: SOUTH
17 1 , BC type: wall , BC face: WEST
17 2 , BC type: internal , BC face: NORTH
17 3 , BC type: internal , BC face: BOTTOM
17 4 , BC type: wall , BC face: TOP
17 5 , BC type: wall , BC face: EAST
VRML line for volume, block: 18
18 0 , BC type: internal , BC face: BOTTOM
18 1 , BC type: internal , BC face: EAST
18 2 , BC type: wall , BC face: NORTH
18 3 , BC type: internal , BC face: TOP
4 , BC: periodic
18 4 , BC type: periodic , BC face: WEST
18 5 , BC type: wall , BC face: SOUTH
VRML line for volume, block: 19
19 0 , BC type: internal , BC face: SOUTH
19 1 , BC type: wall , BC face: WEST
19 2 , BC type: internal , BC face: NORTH
19 3 , BC type: internal , BC face: TOP
19 4 , BC type: wall , BC face: BOTTOM
19 5 , BC type: wall , BC face: EAST

VRML line for volume, block: 20
0 , BC: periodic
20 0 , BC type: periodic , BC face: EAST
20 1 , BC type: wall , BC face: NORTH
20 2 , BC type: internal , BC face: BOTTOM
20 3 , BC type: internal , BC face: WEST
20 4 , BC type: wall , BC face: SOUTH
20 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 21
21 0 , BC type: internal , BC face: NORTH
21 1 , BC type: wall , BC face: WEST
21 2 , BC type: internal , BC face: SOUTH
21 3 , BC type: internal , BC face: BOTTOM
21 4 , BC type: wall , BC face: TOP
21 5 , BC type: wall , BC face: EAST
VRML line for volume, block: 22
22 0 , BC type: internal , BC face: TOP
22 1 , BC type: internal , BC face: EAST
22 2 , BC type: wall , BC face: NORTH
22 3 , BC type: internal , BC face: BOTTOM
4 , BC: periodic
22 4 , BC type: periodic , BC face: WEST
22 5 , BC type: wall , BC face: SOUTH
VRML line for volume, block: 23
23 0 , BC type: internal , BC face: NORTH
23 1 , BC type: wall , BC face: WEST
23 2 , BC type: internal , BC face: SOUTH
23 3 , BC type: internal , BC face: TOP
23 4 , BC type: wall , BC face: BOTTOM
23 5 , BC type: wall , BC face: EAST
VRML line for volume, block: 24
0 , BC: periodic
24 0 , BC type: periodic , BC face: EAST
24 1 , BC type: wall , BC face: NORTH
24 2 , BC type: internal , BC face: TOP
24 3 , BC type: internal , BC face: WEST
24 4 , BC type: wall , BC face: SOUTH
24 5 , BC type: internal , BC face: BOTTOM
VRML line for volume, block: 25

25 0 , BC type: internal , BC face: SOUTH
25 1 , BC type: wall , BC face: WEST
25 2 , BC type: internal , BC face: NORTH
25 3 , BC type: internal , BC face: BOTTOM
25 4 , BC type: wall , BC face: TOP
25 5 , BC type: wall , BC face: EAST
VRML line for volume, block: 26
26 0 , BC type: internal , BC face: SOUTH
26 1 , BC type: internal , BC face: NORTH
26 2 , BC type: wall , BC face: WEST
26 3 , BC type: internal , BC face: TOP
26 4 , BC type: wall , BC face: BOTTOM
26 5 , BC type: wall , BC face: EAST
VRML line for volume, block: 27
27 0 , BC type: internal , BC face: BOTTOM
27 1 , BC type: internal , BC face: EAST
2 , BC: periodic
27 2 , BC type: periodic , BC face: WEST
27 3 , BC type: wall , BC face: NORTH
27 4 , BC type: internal , BC face: TOP
27 5 , BC type: wall , BC face: SOUTH
VRML line for volume, block: 28
28 0 , BC type: internal , BC face: SOUTH
1 , BC: periodic
28 1 , BC type: periodic , BC face: EAST
28 2 , BC type: wall , BC face: BOTTOM
3 , BC: outlet
28 3 , BC type: outlet , BC face: NORTH
28 4 , BC type: internal , BC face: WEST
28 5 , BC type: wall , BC face: TOP
VRML line for volume, block: 29
29 0 , BC type: internal , BC face: SOUTH
29 1 , BC type: internal , BC face: EAST
29 2 , BC type: internal , BC face: WEST
29 3 , BC type: wall , BC face: TOP
4 , BC: outlet
29 4 , BC type: outlet , BC face: NORTH
29 5 , BC type: wall , BC face: BOTTOM
VRML line for volume, block: 30

30 0 , BC type: internal , BC face: SOUTH
30 1 , BC type: internal , BC face: EAST
2 , BC: periodic
30 2 , BC type: periodic , BC face: WEST
30 3 , BC type: wall , BC face: BOTTOM
4 , BC: outlet
30 4 , BC type: outlet , BC face: NORTH
30 5 , BC type: wall , BC face: TOP
VRML line for volume, block: 31
0 , BC: inlet
31 0 , BC type: inlet , BC face: SOUTH
31 1 , BC type: wall , BC face: BOTTOM
31 2 , BC type: internal , BC face: NORTH
31 3 , BC type: internal , BC face: WEST
31 4 , BC type: internal , BC face: EAST
31 5 , BC type: wall , BC face: TOP
VRML line for volume, block: 32
32 0 , BC type: internal , BC face: WEST
32 1 , BC type: internal , BC face: EAST
32 2 , BC type: wall , BC face: SOUTH
32 3 , BC type: internal , BC face: BOTTOM
32 4 , BC type: wall , BC face: TOP
32 5 , BC type: wall , BC face: NORTH
VRML line for volume, block: 33
33 0 , BC type: internal , BC face: WEST
33 1 , BC type: internal , BC face: BOTTOM
33 2 , BC type: internal , BC face: EAST
33 3 , BC type: wall , BC face: SOUTH
33 4 , BC type: wall , BC face: NORTH
33 5 , BC type: internal , BC face: TOP
VRML line for volume, block: 34
34 0 , BC type: internal , BC face: WEST
34 1 , BC type: internal , BC face: EAST
34 2 , BC type: wall , BC face: SOUTH
34 3 , BC type: internal , BC face: TOP
34 4 , BC type: wall , BC face: BOTTOM
34 5 , BC type: wall , BC face: NORTH
VRML line for volume, block: 35
35 0 , BC type: internal , BC face: WEST

35 1 , BC type: internal , BC face: SOUTH
35 2 , BC type: internal , BC face: EAST
35 3 , BC type: wall , BC face: TOP
4 , BC: outlet
35 4 , BC type: outlet , BC face: NORTH
35 5 , BC type: wall , BC face: BOTTOM
total number of lines: 216
done writing VRML script

Part VIII
VRML file


```
## \file VRML_bc_test.py
## \ingroup libgeom2
## \brief Exercise the Surface classes.
## \author Haiko Rijkers
## \version 09-nov-2011

import sys
import os
sys.path.append(os.path.expandvars("$HOME/e3bin"))
from libprep3 import *
from import_cgns_grid import *
from math import pi

# Conditions
############################
def write_VRML_bc(g,bc,face, color, vol):
        #BOTTOM
        bf1= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf2= BilinearFunction(0.0,0.0,1.0,1.0) #j
        bf3= BilinearFunction(0.0,0.0,0.0,0.0) #k

        #TOP
        bf4= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf5= BilinearFunction(0.0,0.0,1.0,1.0) #j
        bf6= BilinearFunction(1.0,1.0,1.0,1.0) #k

        #NORTH
        bf7= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf8= BilinearFunction(0.0,0.0,0.0,0.0) #j
        bf9= BilinearFunction(0.0,0.0,1.0,1.0) #k

        #SOUTH
        bf10= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf11= BilinearFunction(1.0,1.0,1.0,1.0) #j
        bf12= BilinearFunction(0.0,0.0,1.0,1.0) #k

        #WEST
        bf13= BilinearFunction(0.0,0.0,0.0,0.0) #i
        bf14= BilinearFunction(0.0,1.0,0.0,1.0) #j
        bf15= BilinearFunction(0.0,0.0,1.0,1.0) #k

        #EAST
        bf16= BilinearFunction(1.0,1.0,1.0,1.0) #i
        bf17= BilinearFunction(0.0,1.0,0.0,1.0) #j
        bf18= BilinearFunction(0.0,0.0,1.0,1.0) #k

        if (face==5):
                surf = SurfaceThruVolume(vol, bf1, bf2, bf3)
        elif (face==4):
                surf = SurfaceThruVolume(vol, bf4, bf5, bf6)
        elif (face==0):
                surf = SurfaceThruVolume(vol, bf7, bf8, bf9)
        elif (face==2):
                surf = SurfaceThruVolume(vol, bf10, bf11, bf12)
        elif (face==3):
                surf = SurfaceThruVolume(vol, bf13, bf14, bf15)
        elif (face==1):
                surf = SurfaceThruVolume(vol, bf16, bf17, bf18)
        outfile.write(surf.vrml_str(color)  +"\n")
        print face, ", BC:", bc['type']

def write_VRML(g,bc,face, jb):
        """
        Starting by deving the volume.
        Next step is to give a color to the faces with a periodic bc
        # Not jet in the file but the next step is giving a color to the inlet and outlet faces
        """
        ni=g.ni-1
        nj=g.nj-1
        nk=g.nk-1
```

```
## \file VRML_bc_test.py
## \ingroup libgeom2
## \brief Exercise the Surface classes.
## \author Haiko Rijkers
## \version 09-nov-2011

import sys
import os
sys.path.append(os.path.expandvars("$HOME/e3bin"))
from libprep3 import *
from import_cgns_grid import *
from math import pi

# Conditions
############################
def write_VRML_bc(g,bc,face, color, vol):
        #BOTTOM
        bf1= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf2= BilinearFunction(0.0,0.0,1.0,1.0) #j
        bf3= BilinearFunction(0.0,0.0,0.0,0.0) #k

        #TOP
        bf4= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf5= BilinearFunction(0.0,0.0,1.0,1.0) #j
        bf6= BilinearFunction(1.0,1.0,1.0,1.0) #k

        #NORTH
        bf7= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf8= BilinearFunction(0.0,0.0,0.0,0.0) #j
        bf9= BilinearFunction(0.0,0.0,1.0,1.0) #k

        #SOUTH
        bf10= BilinearFunction(0.0,1.0,0.0,1.0) #i
        bf11= BilinearFunction(1.0,1.0,1.0,1.0) #j
        bf12= BilinearFunction(0.0,0.0,1.0,1.0) #k

        #WEST
        bf13= BilinearFunction(0.0,0.0,0.0,0.0) #i
        bf14= BilinearFunction(0.0,1.0,0.0,1.0) #j
        bf15= BilinearFunction(0.0,0.0,1.0,1.0) #k

        #EAST
        bf16= BilinearFunction(1.0,1.0,1.0,1.0) #i
        bf17= BilinearFunction(0.0,1.0,0.0,1.0) #j
        bf18= BilinearFunction(0.0,0.0,1.0,1.0) #k

        if (face==5):
                surf = SurfaceThruVolume(vol, bf1, bf2, bf3)
        elif (face==4):
                surf = SurfaceThruVolume(vol, bf4, bf5, bf6)
        elif (face==0):
                surf = SurfaceThruVolume(vol, bf7, bf8, bf9)
        elif (face==2):
                surf = SurfaceThruVolume(vol, bf10, bf11, bf12)
        elif (face==3):
                surf = SurfaceThruVolume(vol, bf13, bf14, bf15)
        elif (face==1):
                surf = SurfaceThruVolume(vol, bf16, bf17, bf18)
        outfile.write(surf.vrml_str(color)  +"\n")
        print face, ", BC:", bc['type']

def write_VRML(g,bc,face, jb):
        """
        Starting by deving the volume.
        Next step is to give a color to the faces with a periodic bc
        # Not jet in the file but the next step is giving a color to the inlet and outlet faces
        """
        ni=g.ni-1
        nj=g.nj-1
        nk=g.nk-1
```