Grid*Pro* API – Tandem Spheres

API Tutorial - 4

Scope of the tutorial

- In this tutorial, we will discuss about how to generate multiple grids for different flow scenarios by transforming the surfaces and its topology.
- In the current API version, we don't have the commands exposed, hence we will be using an work around to utilize the required commands.
- We will also be invoking the quality control script within this to control the grid generation.

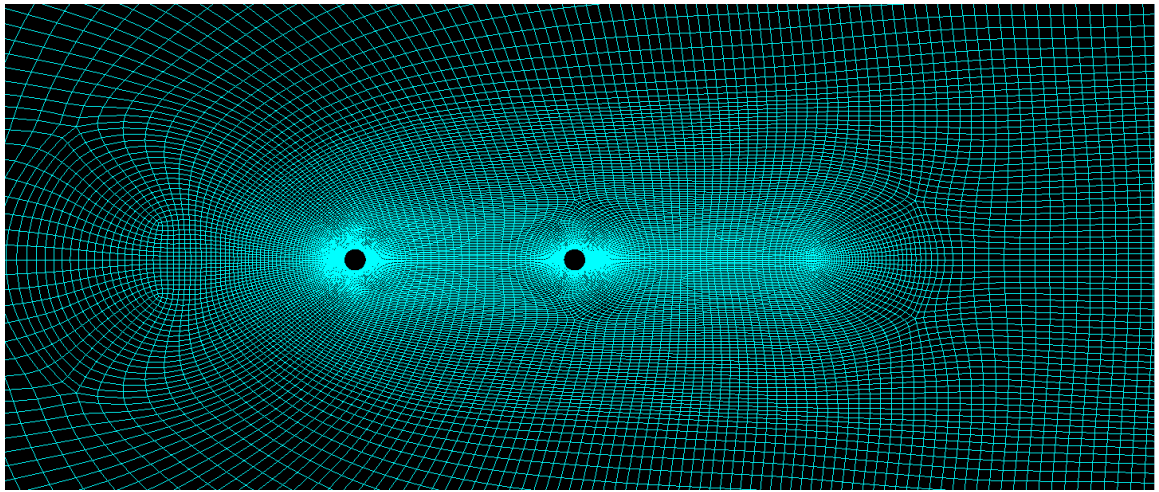


Figure 1: Tandem spheres grid @ 10unit distance

Problem Definition

- Consider two spheres of diameter 1 unit arranged in tandem at a distance of 10 units.
- A template topology is created and saved as template.fra with all the necessary topology and surfaces grouped.
- The goal of this tutorial is to reduce the distance between the spheres from 10 units to 5 units with a decrement of 1 by moving the second sphere closer to the first one.
- And generate the grid for each of the variations by invoking the Quality Control script to achieve a grid quality of zero volume folds and a maximum skewness of 0.7.

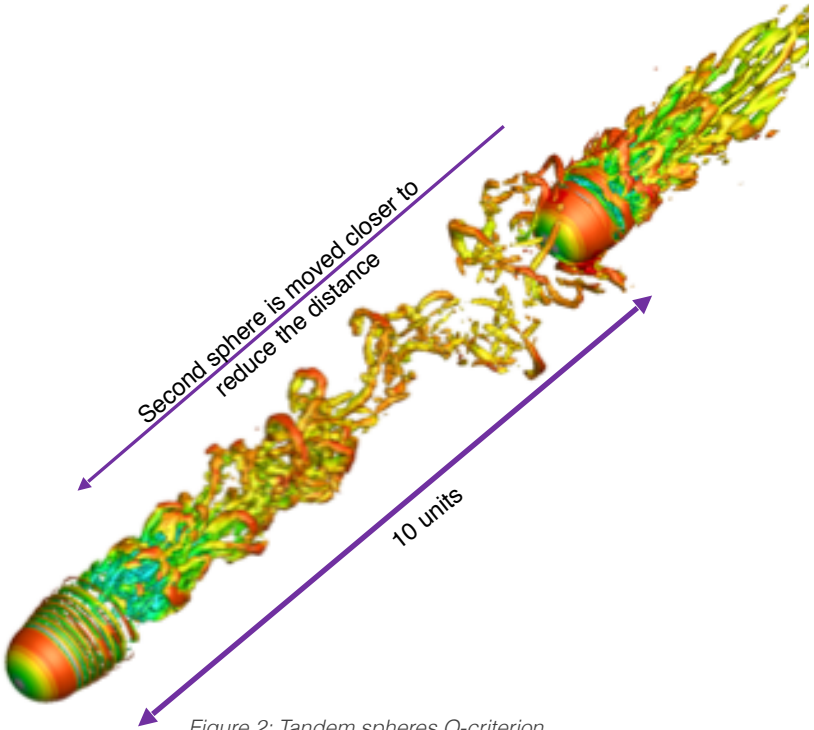
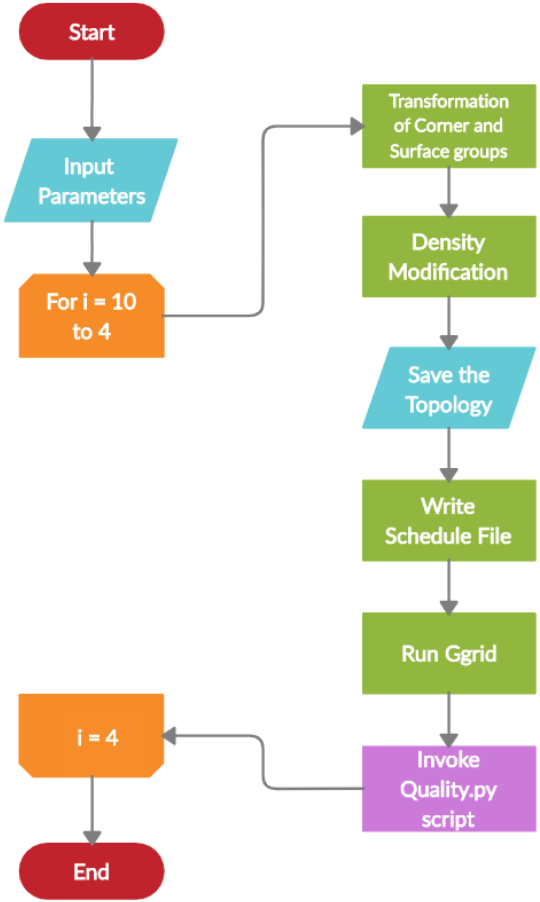


Figure 2: Tandem spheres Q-criterion

Problem Definition

- The work flow of the script goes as follows:
  1. Input the template.fra file.
  2. Get the required input parameters.
  3. Transform the desired corner and surface groups.
  4. Modify the density of the desired edges.
  5. Save the output topology file.
  6. Write the schedule file as required by the Quality Control script.
  7. Call the Quality Control script.
  8. Repeat step 3 to 7 till the distance is reduced from 10 to 5.
  9. End



Input Parameters

- To start with, we need to group the surface and topology to be transformed.
- Here the second sphere is grouped to S1 and it's relevant topology is grouped to C1.
- The topology before and after the sphere is grouped to C2 for further transformation.
- Initial densities of the edges before and after the spheres and it's corresponding corners.

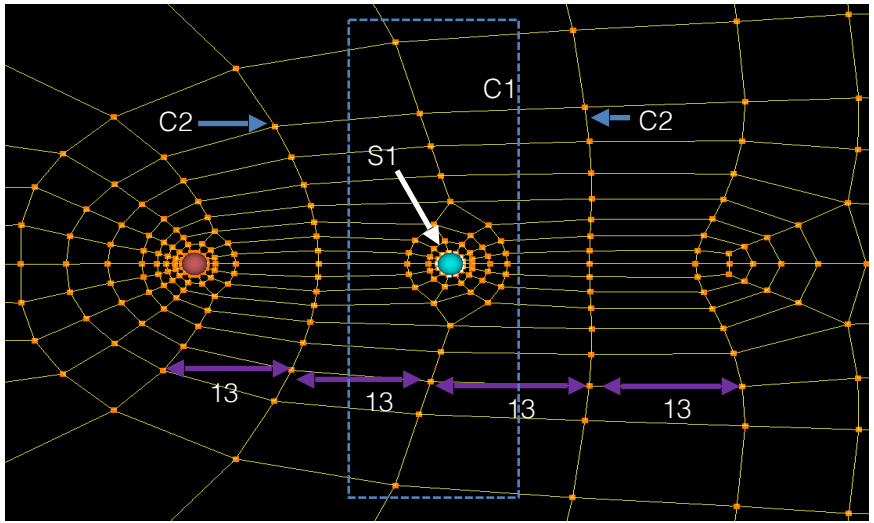


Figure 3: Topology Groups and Density

Corner id 1	Corner id 2	Density
1636	1563	13
1563	1983	13
2331	558	13
558	205	13

Table 1: Corner ids and Density

Input Parameters

- There are certain common parameters which are needed by both the Tandem_Spheres.py and Quality.py
- These are stored in separate *.py file so that both the scripts can access it.
- For each variation, the input topology and output grid file names are written with a common prefix, after applying the transformation and density modification.

Input

- ☐ Topology – { }.fra
- ☐ Step Count - 5
- ☐ Sweep Count - 500
- ☐ Output – { }.grd
- ☐ Folds – 0
- ☐ Skewness – 0.7

```
corners_between_spheres = (1636, 1563, 1983)
corners_after_spheres = (2331, 558, 205)
step_count = 5
sweep_count = 500
skewness = 0.7
```

Figure 4: Variables.py file

Code Snippet – Tandem_Spheres.py

```python
# IMPORT OPERATIONS
import gp_utilities
from gp_utilities import vector3d as vec
import os
import variables

def topo_modify(corners_along_sphere, sphere_surface_group, corners_around_sphere,
            translation_for_sphere, translation_around_sphere, density_between_spheres,
density_after_spheres):

      #Topology_Modification
      gp_utilities.execute_command("transform_topo -g {} -sg {} -t1 {} {}
      {}".format(corners_along_sphere,  sphere_surface_group,
      translation_for_sphere.get_component(0),
                  translation_for_sphere.get_component(1),
      translation_for_sphere.get_component(2)),             topo, True)
      gp_utilities.execute_command("transform_topo -g {} -t1 {} {} {}".format(corners_around_sphere,

      translation_around_sphere.get_component(0),translation_around_sphere.get_component(1),
      translation_around_sphere.get_component(2)), topo, True)

      #Density_Modification_between_spheres
      den = topo.den()
      den.set_density(topo.corner(variables.corners_between_spheres[0]),
      topo.corner(variables.corners_between_spheres[1]), density_between_spheres)
      den.set_density(topo.corner(variables.corners_between_spheres[1]),
      topo.corner(variables.corners_between_spheres[2]), density_between_spheres)

      #Density_Modification_after_spheres
      den.set_density(topo.corner(variables.corners_after_spheres[0]),
      topo.corner(variables.corners_after_spheres[1], density_after_spheres)
      den.set_density(topo.corner(variables.corners_after_spheres[1]),
      topo.corner(variables.corners_after_spheres[2]), density_after_spheres)
      return topo

def write_schedule_file():

     #Write_Schedule_File
      file = open("{}.sch".format(output_file_prefix), "w+")
      file.writelines("step {}: -c all 1.0 0 -C all 1.0 24 -r -S {} -w "
                      "\nstep {}: -sys 'ws qchk {}.grd 11 10000 {} 120' "
                      "\nstep {}: -sys 'python Quality.py {}.sch'"
                      "\nwrite -f {}.grd".format(variables.step_count, variables.sweep_count,
                      variables.step_count+1, output_file_prefix, variables.skewness,
                      variables.step_count+2, output_file_prefix, output_file_prefix))
      file.close()
```

```python
#Main Function
if(__name__ == '__main__'):
   topo = gp_utilities.Topology()

   #Input_Parameters
   input_file_prefix = "template"
   initial_distance = float(10)
   den_between_spheres = 13
   den_after_spheres = 13
   corner_grp_along_sphere = 1
   corner_grp_around_sphere = 2
   sphere_surface_grp = 1

   #Topology Modification and Run Ggrid
   for i in range(10, 4, -1):
      topo.read("{}.fra".format(input_file_prefix))
      translation_for_grp_along_sphere = vec(-(initial_distance-i),
      0, 0)
      translation_for_grp_around_sphere = vec(-((initial_distance -
      i) / 2), 0, 0)
      den_between_spheres = den_between_spheres-1
      den_after_spheres = den_after_spheres+1
      topo_modify(corner_grp_along_sphere, sphere_surface_grp,
               corner_grp_around_sphere,
               translation_for_grp_along_sphere,
      translation_for_grp_around_sphere,      den_between_spheres,
      den_after_spheres)
      output_file_prefix = "distance_{}units".format(i)
      topo.write_topology("{}.fra".format(output_file_prefix))
      write_schedule_file()
      Ggrid = "Ggrid {}.fra".format(output_file_prefix)
      os.system(Ggrid)
```

Code Snippet – Quality.py

```
# Import libraries
import sys
import fileinput
import variables

# Evaluate fold count and skewness value from qcheck
output files
def evaluate_fold_count_from_qcheck_output():
    # Calculating number of folds and skew
    folds = open('bad_folds.hex').readline()
    folds = int(folds)
    skew = open('bad_skewness.hex').readline()
    skew = int(skew)
    return [folds, skew]

# Extend schedule file if the grid quality is not good
enough
def extend_schedule_file(schedule_file_name):

# Evaluate fold count and skewness value from qcheck
output files
    folds, skew =
evaluate_fold_count_from_qcheck_output()
```

```
# Checking the desired quality condition
count = variables.step_count-1
desired_quality = (folds == 0 and skew == 0)
for line in fileinput.input(schedule_file_name, inplace=1):
    count += 1
    if count > 50:
        break
    elif line.startswith('write'):
        if desired_quality:
            print ("step {}: -c all 1.0 0 -C all 1.0 24 -r -S {} -w\n"
                   + line.rstrip()).format(count, variables.sweep_count)
        else:
            print ("step {}: -c all 1.0 0 -C all 1.0 24 -r -S {} -w "
                   "\nstep {}: -sys 'ws qchk {}.grd 11 10000 {} 120' "
                   "\nstep {}: -sys 'python Quality.py {}'\n"
                   + line.rstrip()).format(count, variables.sweep_count,
                  count + 1, schedule_file_name[:-4],
variables.skewness,
count + 2, schedule_file_name)
    else:
        print line.rstrip()

# Main Function
if (__name__ == '__main__'):
   extend_schedule_file(sys.argv[1])
```

API Module

- Import the following libraries:
  1. gp_utilities = GridPro's API
  2. vector 3d = From API import vector3d as vec
  3. os = To run system command
  4. variables = variables.py file with common parameters for both Tandem_Spheres.py and Quality.py

```
import gp_utilities
from gp_utilities import vector3d as vec
import os
import variables
```

API Module

- First Function: To transform the given corners & surface group and modify the density of the desired edges.
  1. Transform the corner & surface group of the second sphere
  2. Transform the corner group before and after the sphere
  3. Modify the densities of the desired edges

```
def topo_modify(corners_along_sphere, sphere_surface_group, corners_around_sphere, translation_for_sphere,
                translation_around_sphere, density_between_spheres, density_after_spheres):
    #Topology_Modification
    gp_utilities.execute_command("transform_topo -g {} -sg {} -t1 {} {} {}".format(corners_along_sphere, sphere_surface_group,
                                                                                   translation_for_sphere.get_component(0),translation_for_sphere.get_component(1),
                                                                                   translation_for_sphere.get_component(2)), topo, True)
    gp_utilities.execute_command("transform_topo -g {} -t1 {} {} {}".format(corners_around_sphere, translation_around_sphere.get_component(0),
                                                                            translation_around_sphere.get_component(1), translation_around_sphere.get_component(2)), topo, True)

    #Density_Modification_between_spheres
    den = topo.den()
    den.set_density(topo.corner(variables.corners_between_spheres[0]), topo.corner(variables.corners_between_spheres[1]), density_between_spheres)
    den.set_density(topo.corner(variables.corners_between_spheres[1]), topo.corner(variables.corners_between_spheres[2]), density_between_spheres)

    #Density_Modification_after_spheres
    den.set_density(topo.corner(variables.corners_after_spheres[0]), topo.corner(variables.corners_after_spheres[1]), density_after_spheres)
    den.set_density(topo.corner(variables.corners_after_spheres[1]), topo.corner(variables.corners_after_spheres[2]), density_after_spheres)
    return topo
```

API Module

- Second Function: To write the schedule file desired by the Quality Control script.
  1. Transform the corner & surface group of the second sphere
  2. Transform the corner group before and after the sphere
  3. Modify the densities of the desired edges

```
def write_schedule_file():
    #Write_schedule_file
    file = open("{}.sch".format(output_file_prefix), "w+")
    file.writelines("step {}: -c all 1.0 0 -C all 1.0 24 -r -S {} -w "
                    "\nstep {}: -sys 'ws qchk {}.grd 11 10000 {} 120' "
                    "\nstep {}: -sys 'python Quality.py {}.sch'"
                    "\nwrite -f {}.grd".format(variables.step_count, variables.sweep_count, variables.step_count+1,
                                                output_file_prefix, variables.skewness, variables.step_count+2,
                                                output_file_prefix, output_file_prefix))
    file.close()
```

API Module

- Main Function:

1. Collect the input parameters

```
#Input_Parameters
input_file_prefix = "template"
initial_distance = float(10)
den_between_spheres = 13
den_after_spheres = 13
corner_grp_along_sphere = 1
corner_grp_around_sphere = 2
sphere_surface_grp = 1
```

2. Iterate the topology modification function from a distance of 10 to 5 with a decrement of 1.
3. Run Ggrid. This will call Quality.py internally and evaluates the grid quality as explained in the previous tutorial.

```
#Topology Modification and Run Ggrid
for i in range(10, 4, -1):
    topo.read("{}.fra".format(input_file_prefix))
    translation_for_grp_along_sphere = vec(-(initial_distance-i), 0, 0)
    translation_for_grp_around_sphere = vec(-((initial_distance - i) / 2), 0, 0)
    den_between_spheres = den_between_spheres-1
    den_after_spheres = den_after_spheres+1
    topo_modify(corner_grp_along_sphere, sphere_surface_grp, corner_grp_around_sphere,
            translation_for_grp_along_sphere, translation_for_grp_around_sphere,
            den_between_spheres, den_after_spheres)
    output_file_prefix = "distance_{}units".format(i)
    topo.write_topology("{}.fra".format(output_file_prefix))
    write_schedule_file()
    Ggrid = "Ggrid {}.fra".format(output_file_prefix)
    os.system(Ggrid)
```

API Module

- Quality Control Script modification:
  1. In order to update the current script information, we will have to modify the Quality.py script details.
  2. Input parameters are now accessed from variables.py file.
  3. Grid file name is made as an variable which changes for every variation.

```
def extend_schedule_file(schedule_file_name):

    # Evaluate fold count and skewness value from qcheck output files
    folds,skew = evaluate_fold_count_from_qcheck_output()

    # Checking the desired quality condition
    count = variables.step_count-1
    is_good_enough = (folds == 0 and skew == 0)
    for line in fileinput.input(schedule_file_name, inplace=1):
        count += 1
        if count > 50:
            break
        elif line.startswith('write'):
            if is_good_enough:
                print ("step {}: -c all 1.0 0 -C all 1.0 24 -r -S {} -w\n"
                        + line.rstrip()).format(count, variables.sweep_count)
            else:
                print ("step {}: -c all 1.0 0 -C all 1.0 24 -r -S {} -w"
                       "\nstep {}: -sys 'ws qchk {}.grd 11 10000 {} 120' "
                       "\nstep {}: -sys 'python Quality.py {}'\n"
                       + line.rstrip()).format(count, variables.sweep_count,
                               count + 1, schedule_file_name[:-4], variables.skewness,
                               count + 2, schedule_file_name)
        else:
            print line.rstrip()
```

End of the Tutorial