CFD Modeling of Thermal Conductivity in Composite Materials with Hollow Microspheres

Group Members: Mayank Raj (B21ME038)
Mota Ram(B21ME039)

Introduction:- The primary objective of this study is to employ Computational Fluid Dynamics (CFD) techniques to model the thermal conductivity of composite materials containing hollow microspheres. By comprehensively investigating the thermal behavior of such composite structures, we aim to elucidate the impact of porosity and cavity size range on thermal conductivity prediction.

Thermal conductivity plays a pivotal role in determining the heat transfer characteristics of composite materials. It influences their ability to dissipate heat efficiently, thereby affecting their thermal stability, durability, and overall performance. Accurate prediction of thermal conductivity is essential for the design and optimization of composite materials, ensuring their suitability for specific applications and operating conditions.

The incorporation of hollow microspheres within composite materials introduces complexities in thermal conductivity modeling due to the presence of void spaces and varying cavity sizes. Therefore, a systematic approach integrating advanced simulation techniques and automation methods is employed to address these challenges effectively.

Methodology:-

A. Geometry Creation:

- Utilized Ansys SpaceClaim with IronPython scripting to generate complex geometries.
- Input parameters included porosity (void fraction) and cavity size range, allowing for the creation of a range of composite material configurations.
- Hollow microspheres were distributed within a solid matrix, mimicking real-world composite structures.

- The IronPython code automated the generation process, enabling the creation of numerous geometries efficiently.

B. Meshing:-

- After geometry creation, the next step involved mesh generation using Ansys Mechanical.
- Mesh quality significantly impacts CFD simulation accuracy; therefore, careful consideration was given to mesh refinement.
- Automated meshing procedures were implemented using code, ensuring consistency and reproducibility across multiple simulations.
- Mesh parameters were optimized to capture geometric details and ensure computational efficiency.

C. Simulation:-

- Fluent, a CFD software, was employed for conducting thermal conductivity simulations.
- Boundary conditions were set to simulate realistic thermal environments, including temperature differences and heat fluxes.
- The simulation setup aimed to replicate conditions relevant to composite material applications, such as aerospace or electronics.
- Computational models were solved iteratively to obtain temperature distributions within the composite material.

Automation Process:-

A. Geometry Automation:-

- IronPython scripting facilitated the automation of geometry creation in Ansys SpaceClaim.
- Input parameters, such as porosity and cavity size range, were systematically varied to generate a diverse set of composite material configurations.
- The automation process ensured repeatability and efficiency in generating complex geometries, saving time and reducing the likelihood of errors.

Spaceclaim Geometry generation code:-

```
# Python Script, API Version = V23
import clr
import random
from SpaceClaim.Api.V23 import *

por = []
grain = []

min_radius = 0.02 ## in mm
max_radius = 0.02 ## in mm


def generate_circles(area, face, min_radius, max_radius):
    circles = []

    a = 0
    x =0
    while a < area:
        x = x+1
        isOverlap = True
        while isOverlap:
            radius = random.uniform(min_radius, max_radius)
            x_center = random.uniform(-0.5 + radius, 0.5 - radius)
            y_center = random.uniform(-0.5 + radius, 0.5 - radius)
            isOverlap = False
            for j in range(len(circles)):
                xr = x_center - circles[j][0]
                yr = y_center - circles[j][1]
                d = radius + circles[j][2]
                d1 = ((xr) ** 2 + (yr) ** 2) ** 0.5
                if d1 < d:
                    isOverlap = True
                    break
            if not isOverlap:
                origin = Point2D.Create(MM(x_center), MM(y_center))
```

```
                circle = SketchCircle.Create(origin, MM(radius))
                a += 3.14 * (radius) ** 2
                new_circle = [x_center, y_center, radius]
                circles.append(new_circle)
    return x

# making geometry
porous = (0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.50)  ## Vol %
## validation paper: Simulation of heat transfer in
hollow-glass-bead-filled polypropylene composites by finite element method
for i in range(11):    # define number of geometry to be generated
        porosity  = porous[i]
        #porosity = random.uniform(0.1, 0.1)
        area = 1*porosity
    # Set Sketch Plane
        sectionPlane = Plane.PlaneXY
        result = ViewHelper.SetSketchPlane(sectionPlane, Info1)
        # EndBlock

        # Sketch Rectangle
        point1 = Point2D.Create(MM(-0.5),MM(0.5))
        point2 = Point2D.Create(MM(0.5),MM(0.5))
        point3 = Point2D.Create(MM(0.5),MM(-0.5))
        result1 = SketchRectangle.Create(point1, point2, point3)
        # EndBlock

        # Solidify Sketch
        mode = InteractionMode.Solid
        result = ViewHelper.SetViewMode(mode, Info2)
        # EndBlock

        # Create New Part
        selection = Part1
        result = ComponentHelper.CreateNewComponent(selection, Info3)
        # EndBlock

        # Set Sketch Plane
        selection = Face1
        result = ViewHelper.SetSketchPlane(selection, Info4)
        # EndBlock
```

```
 # Function to generate random non-overlapping circles

 x = generate_circles(area,sectionPlane,min_radius,max_radius)
 por.append(porosity)
 grain.append(x)

# Solidify Sketch
 mode = InteractionMode.Solid
 result = ViewHelper.SetViewMode(mode, Info5)
# Intersect Bodies
 targets = Body2
 tools = Body1
 options = MakeSolidsOptions()
 result = Combine.Intersect(targets, tools, options, Info9)
 # EndBlock

         # Name selection
 sel = Selection.Create(Edge1)
 sel.CreateAGroup("wall1")    # upper side wall

 sel = Selection.Create(Edge2)
 sel.CreateAGroup("wall2")    # downside wall

 sel = Selection.Create(Edge4)
 sel.CreateAGroup("high")    # inlet of geometry

 sel = Selection.Create(Edge3)
 sel.CreateAGroup("low")   # outlet

 sel = Selection.Create(Face2)
 sel.CreateAGroup("domain")    # domain

 sel = Selection.Create(Body3)
 sel.CreateAGroup("porous")    # domain

 # Exclude items from physics
 selection = Component1
 suppress = True
```

```
        ViewHelper.SetSuppressForPhysics(selection, suppress)
        # EndBlock

        #Saving file
        DocumentSave.Execute('D:\\New
folder\\spaceclaim_geometry\\rad_0.02_{j}.scdoc'.format(j=i+1)) # path of
location

         # Deleting the file
        selection = Body4
        result = Delete.Execute(selection)
        selection = Component1
        result = Delete.Execute(selection)


for j in por:
    print(j)


for k in grain:
    print(k)
```

B. Meshing Automation:-

- Mesh generation was automated using code within Ansys Mechanical.
- Parameters for mesh refinement and element types were predefined in the automation script to maintain consistency across simulations.
- Automated meshing minimized manual intervention, ensuring uniform mesh quality and reducing computational overhead.

Meshing Script:-

```
for i in range(11):
    geometry_import_13 = DataModel.GetObjectById(13)
    geometry_import_13.Import(r"D:\\New
folder\\spaceclaim_geometry\por_{j}.scdoc".format(j=i+1))

    mesh_15 = Model.Mesh
```

```
    mesh_15.ElementSize = Quantity(0.00001, "m")
    mesh_15.GenerateMesh()


ExtAPI.DataModel.Project.Model.Mesh.InternalObject.WriteFluentInputFile(r"D:\\New
folder\\mesh_len_vary\\sav{j}.msh".format(j=i+1))
```

C. Simulation Automation:-

- Automation scripts were developed to streamline the setup and execution of thermal conductivity simulations in Fluent.
- Boundary conditions, solver settings, and post-processing steps were predefined in the scripts, enhancing efficiency and reproducibility.
- Automation allowed for batch processing of multiple simulations, enabling comprehensive parametric studies.

Fluent Simulation code:-

```
import numpy as np
import ansys.fluent.core as f1
# Launch Fluent solver
solver = f1.launch_fluent(precision =
"double",processor_count=2,mode="solver",show_gui=True,version="2d")
# Lists to store simulation results
den = []      #define Density
K = []        #define Thermal Conductivity

i = 2
# Loop for multiple simulations
while i< 3:
    # Read mesh file
    solver.file.read_mesh(file_name="sav{}.msh".format(i))

    # Set operating conditions and enable energy model
    solver.tui.define.operating_conditions.gravity("yes")
    solver.tui.define.operating_conditions.gravity("yes", "0", "-9.81")
    solver.setup.models.energy.enabled = True

    # Copy material
    solver.execute_tui(r'''/define/materials/copy solid wood ''')


    # Define and apply material
```

```
    solver.tui.define.materials.change_create()
    solver.execute_tui(r'''/define/materials/change-create air air no no yes
constant 0.026 no no no no ''')
    solver.execute_tui(r'''/define/materials/change-create wood matrix yes
constant 915 yes constant 746 yes constant 0.2 ''')
    solver.execute_tui(r'''/define/materials/change-create wood matrix2 yes
constant 2210 yes constant 746 yes constant 0.1 ''')

    solver.setup.materials.fluid()

    # Set cell zone conditions
    solver.setup.cell_zone_conditions.solid()
    solver.tui.define.boundary_conditions.zone_type("domain", "solid")
    solver.setup.cell_zone_conditions.solid['domain'].material = 'matrix'

    #solver.tui.define.boundary_conditions.zone_type("solid-sav1_surface",
"solid")
    #solver.setup.cell_zone_conditions.solid['solid-sav1_surface'].material =
'matrix2'
    solver.setup.cell_zone_conditions.solid['solid-rad_0.02_{}_surface'.format(i)]
= {"material" : "matrix2"}

    # Modify zone types for boundary conditions
    solver.execute_tui(r'''/define/boundary-conditions/modify-zones/zone-type high
wall ''')
    solver.execute_tui(r'''/define/boundary-conditions/modify-zones/zone-type low
wall ''')

    # Set temperature conditions
    solver.setup.boundary_conditions.wall['low'](thermal_bc = 'Temperature', t =
300)
    solver.setup.boundary_conditions.wall['low'] = {"material" : "matrix"}

    solver.setup.boundary_conditions.wall['high'](thermal_bc = 'Heat Flux',q =
10000)

    solver.setup.boundary_conditions.wall['high'] = {"material" : "matrix"}

    solver.setup.boundary_conditions.wall['wall1'](thermal_bc = 'Heat Flux')
    solver.setup.boundary_conditions.wall['wall1'] = {"material" : "matrix"}
    solver.setup.boundary_conditions.wall['wall2'](thermal_bc = 'Heat Flux')
    solver.setup.boundary_conditions.wall['wall2'] = {"material" : "matrix"}

    # Initialize and run simulation
```

```
    solver.solution.initialization.hybrid_initialize()
    solver.solution.run_calculation.iter_count = 1000
    solver.solution.run_calculation.iterate()

    # Create temperature contour and display it
    solver.results.graphics.contour.create("temperature_contour1")
    solver.results.graphics.contour()
    solver.results.graphics.contour["temperature_contour1"](field =
"total-temperature")
    solver.results.graphics.contour["temperature_contour1"].display()

#     solver.results.graphics.picture.save_picture(file_name=
"Temperature_sample2")
#     from matplotlib import pyplot as plt
#     from matplotlib import image
#     plt.imshow(image.imread("Temperature_sample2.png"))

    # Define and save surface reports
    solver.solution.report_definitions.surface["high"] = {}
    solver.solution.report_definitions.surface['high'].report_type =
"surface-areaavg"
    solver.solution.report_definitions.surface['high'] = {"field" : "temperature",
"surface_names" : ["high"]}
    solver.solution.report_definitions.surface["low"] = {}
    solver.solution.report_definitions.surface['low'].report_type =
"surface-areaavg"
    solver.solution.report_definitions.surface['low'] = {"field" : "temperature",
"surface_names" : ["low"]}
    solver.solution.report_definitions.surface["heat_flux"] = {}
    solver.solution.report_definitions.surface['heat_flux'].report_type =
"surface-areaavg"
    solver.solution.report_definitions.surface['heat_flux'] = {"field" :
"heat-flux", "surface_names" : ["high"]}
    solver.solution.report_definitions.volume["total-volume"] = {}
    solver.solution.report_definitions.volume['total-volume'].report_type =
"volume-zonevol"
    solver.solution.report_definitions.volume['total-volume'] = {"zone_names" :
["domain"]}

    solver.solution.report_definitions.volume["total-mass"] = {}
    solver.solution.report_definitions.volume['total-mass'].report_type =
"volume-mass"
    solver.solution.report_definitions.volume['total-mass'] = {"zone_names" :
["domain"]}
```

```
    solver.solution.report_definitions.single_val_expression["k-eff"] = {}
    solver.solution.report_definitions.single_val_expression['k-eff'] = {"define"
: "({heat_flux}/(({high}-{low})/0.001))"}
    solver.solution.report_definitions.single_val_expression["bulk-density"] = {}
    solver.solution.report_definitions.single_val_expression['bulk-density'] =
{"define" : "({total-mass}/{total-volume})"}
    # Define and save named expressions


    solver.setup.named_expressions["keff"] = {}
    solver.setup.named_expressions['keff'] = {"definition" :
"(heat_flux/((high-low)/.001))", "output_parameter" : True}
    solver.execute_tui(r'''/define/parameters/output-parameters/write-to-file
"keff" "k_0.01.out" ''')

    solver.setup.named_expressions["density"] = {}
    solver.setup.named_expressions['density'] = {"definition" :
"({total-mass}/{total-volume})", "output_parameter" : True}
    solver.execute_tui(r'''/define/parameters/output-parameters/write-to-file
"density" "den_0.01.out" ''')
    i = i+1
```

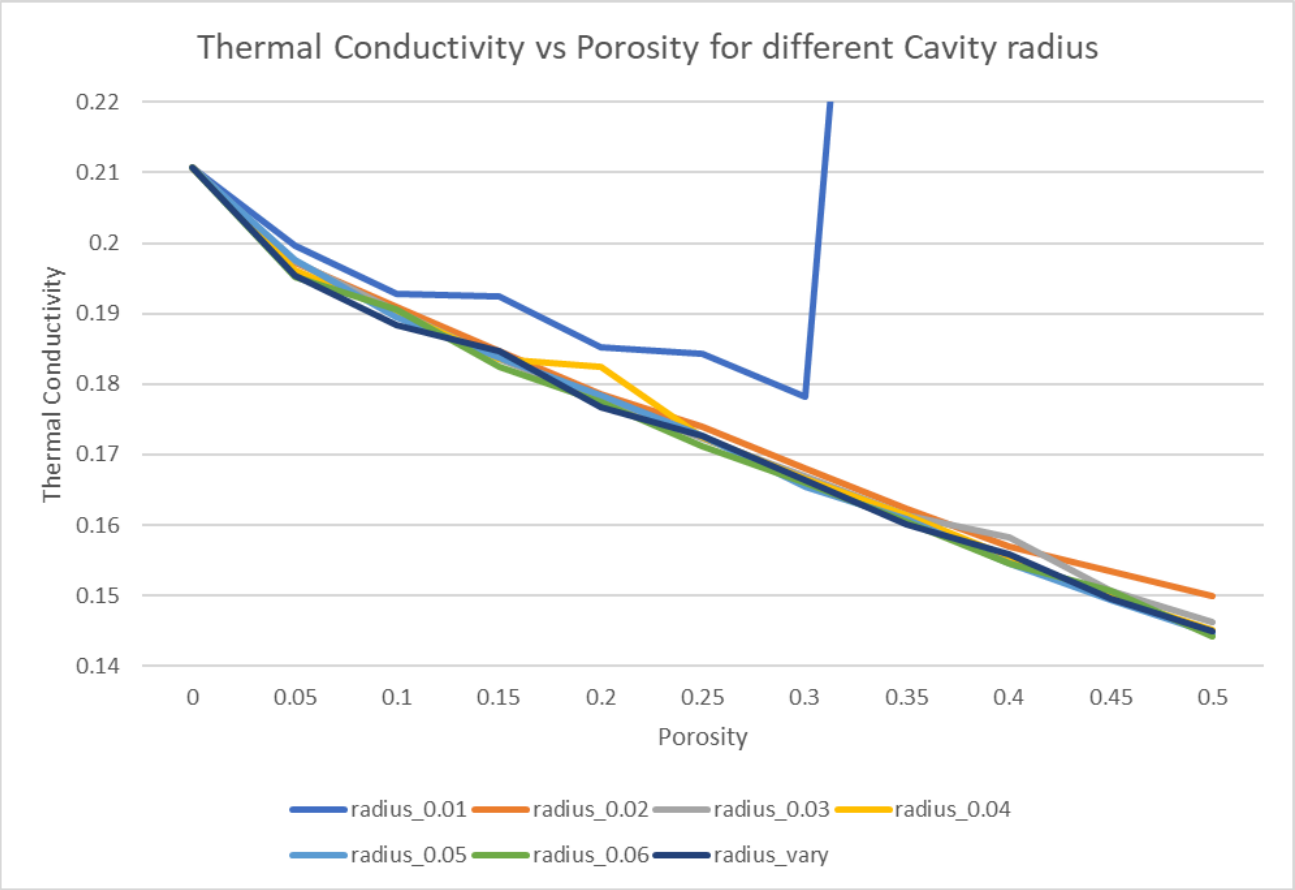
D. Challenges and Solutions:-

- Challenges encountered during the automation process, such as scripting errors or compatibility issues between software versions, were addressed promptly.
- Debugging and testing procedures were implemented to identify and resolve automation issues effectively.
- Collaboration between software developers and domain experts facilitated the development of robust automation scripts tailored to the specific requirements of the study.

Results:-

The study presents predictions of thermal conductivity within composite materials with hollow microspheres. CFD results are compared with experimental data to validate the model's accuracy. Additionally, the analysis includes variations in

parameters such as porosity and cavity size range to understand their effects on thermal conductivity.



Discussion:-

Key findings from the study are interpreted, shedding light on the behavior of thermal conductivity in composite materials with hollow microspheres. Discussions encompass the validity of the CFD model, limitations encountered during the study, and potential implications for composite material design and applications.

Conclusion:-

In conclusion, this study highlights the importance of CFD in predicting thermal conductivity in composite materials with hollow microspheres. The findings contribute to a deeper understanding of heat transfer mechanisms within such materials, emphasizing the significance of accurate modeling for optimizing composite material performance.