Computing the thermal radiation-induced acceleration on ISP

For this task we will use pyRTX capabilities.

First of all we need to define the spacecraft:

In [12]:
```python
from pyRTX.scClass import Spacecraft
import trimesh.transformations as tmt
import trimesh
import numpy as np
from matplotlib.colors import to_rgba_array

obj_path = 'shape/resized/'
scName = 'ISP'
scFrame = 'SC_FRAME'

identity = tmt.identity_matrix()

isp = Spacecraft(

    name = 'isp',
    units = 'm',
    base_frame = scFrame,
    spacecraft_model = {

    'BUS' : {

        'file' : obj_path + 'bus.obj',
        'frame_type' : 'User Defined',
        'frame_name' : scFrame,
        'center'     : [0,0,0],
        'diffuse'    : 0.1,
        'specular'   : 0.3,
        'UD_rotation': identity
        },
        'HGA' : {

        'file' : obj_path + 'hga.obj',
        'frame_type' : 'User Defined',
        'frame_name' : scFrame,
        'center'     : [0,0,0],
        'diffuse'    : 0.1,
        'specular'   : 0.3,
        'UD_rotation': identity
        },
            'RTG' : {

        'file' : obj_path + 'rtg.obj',
        'frame_type' : 'User Defined',
        'frame_name' : scFrame,
        'center'     : [0,0,0],
        'diffuse'    : 0.1,
        'specular'   : 0.3,
        'UD_rotation': identity
        }
    }

    )
```

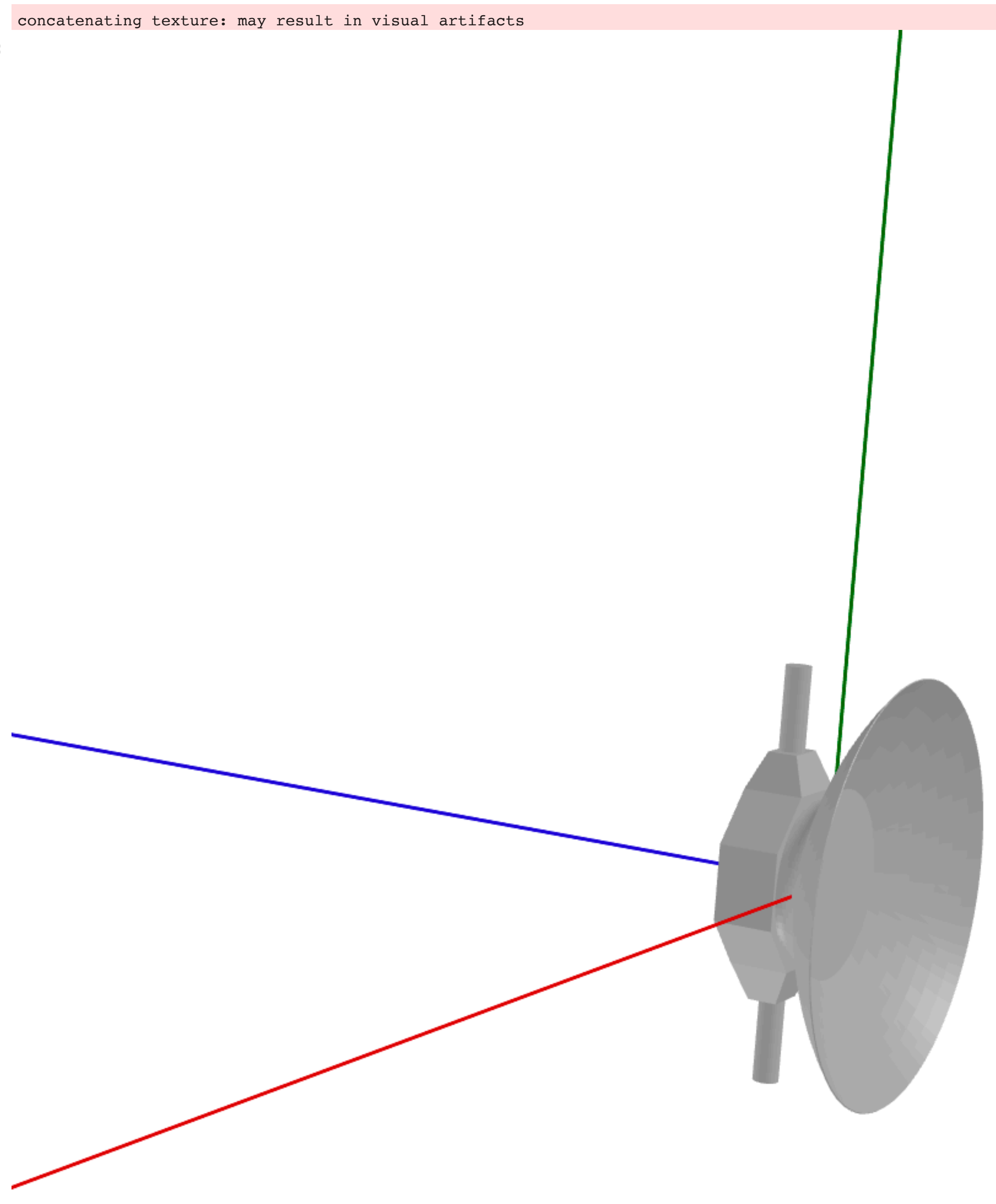
```
/Users/gcasciol/opt/anaconda3/envs/py38/lib/python3.8/site-packages/trimesh/util.py:130: RuntimeWarning: invalid value encountered in reciprocal
  norm[valid] **= -1
```

Let's visualize the spacecraft to be sure that everything is working. We will visualize the body axes as well.

In [13]:
```python
# Retrieve the mesh object
mesh = isp.dump()
mesh.unmerge_vertices()

# Define a container for the objects we want to visualize
scene_elements = []
scene_elements.append(mesh)

# Define the body axes
body_frame_colors = ['red', 'green', 'blue']
xaxis = np.array([1,0,0])
yaxis = np.array([0,1,0])
zaxis = np.array([0,0,1])
origin = np.array([0,0,0])

xaxis = trimesh.load_path(np.hstack(( origin, origin + xaxis*0.01)).reshape(-1, 2, 3))
yaxis = trimesh.load_path(np.hstack(( origin, origin + yaxis*0.01)).reshape(-1, 2, 3))
zaxis = trimesh.load_path(np.hstack(( origin, origin + zaxis*0.01)).reshape(-1, 2, 3))

xaxis.colors = np.full((1,4),to_rgba_array(body_frame_colors[0])*255)
yaxis.colors = np.full((1,4),to_rgba_array(body_frame_colors[1])*255)
zaxis.colors = np.full((1,4),to_rgba_array(body_frame_colors[2])*255)

scene_elements.append(xaxis)
scene_elements.append(yaxis)
scene_elements.append(zaxis)

scene = trimesh.Scene(scene_elements)


##scene.show() This doesn't work in Jupyter. I'll just post a screenshot
from IPython.display import Image
Image("img/mesh.png")
```

```
concatenating texture: may result in visual artifacts
```

Out[13]:



So we can see that the antenna is oriented in the -Z direction. This will be useful later.

Now, let's compute the acceleration due to the emission. For each elemental surface we can write:

$$acc = -\frac{2}{3}\frac{q}{c}\frac{A}{m}\mathbf{n}$$

Where A is the area of the elemental surface, c is the speed of light, $\mathbf{n}$ is the normal to the surface and q is defined as:

$$q = \sigma \varepsilon T^4$$

with σ the Stefan-Boltzmann constant, ϵ the emissivity and T the surface temperature.

Then we need to compute the A and n of each spacecraft surface

In [14]:
```python
from pyRTX.utils_rt import get_centroids, get_surface_normals_and_face_areas
import pyRTX.constants as constants

def VFNC(mesh):
    V = mesh.vertices
    F = mesh.faces
    N = mesh.face_normals
    C = get_centroids(V,F)
    _, A = get_surface_normals_and_face_areas(V,F)
    return V, F, N, C, A

c = constants.c
mesh = isp.dump()
#mesh.fix_normals() # BE sure that the normals are all outward-facing
V, F, N, C, A = VFNC(mesh) # Vertices, Faces, Normals, Centers, Areas

# Now it's a matter of simple computation.
A = np.expand_dims(A, axis = 1)

q = 1
m = 1
acc = np.multiply(-2/3*A/c*q/m, N)
acc = np.sum(acc, axis = 0)
```

```
concatenating texture: may result in visual artifacts
```

Here we have computed the unitary acceleration (i.e. for a unitary flux and unitary mass). This value can be easily rescaled for different values.

Unitary acceleration:

In [15]:
```python
print(acc)
```

```
[-5.50624912e-28  1.54059914e-19  2.26384690e-12]
```

Assuming $\varepsilon = 0.3$ $T = 210C = 383.15K$

We obtain a net force F:

In [18]:
```python
q = constants.stefan_boltzmann*0.3*(383.15)**4
acc = np.multiply(-2/3*A/c*q, N) * 1000
acc = np.sum(acc, axis = 0)
F = np.linalg.norm(acc)
print(F)
```

```
8.299554775156367e-07
```

To do:

- set different temperatures / thermo optical coefficients for each component
- split the HGA mesh for front and back temperatures
- compute the secondary interactions

Let's first try to assess the difference in acceleration direction and magnitude by accounting separately each component temperature

In [28]:
```python
bus = Spacecraft(

    name = 'bus',
    units = 'm',
    base_frame = scFrame,
    spacecraft_model = {

    'BUS' : {

        'file' : obj_path + 'bus.obj',
        'frame_type' : 'User Defined',
        'frame_name' : scFrame,
        'center'     : [0,0,0],
        'diffuse'    : 0.1,
        'specular'   : 0.3,
        'UD_rotation': identity
        },
    }
)

hga = Spacecraft(

    name = 'hga',
    units = 'm',
    base_frame = scFrame,
    spacecraft_model = {
        'HGA' : {

        'file' : obj_path + 'hga.obj',
        'frame_type' : 'User Defined',
        'frame_name' : scFrame,
        'center'     : [0,0,0],
        'diffuse'    : 0.1,
        'specular'   : 0.3,
        'UD_rotation': identity
        },
    }
)

rtg = Spacecraft(

    name = 'rtg',
    units = 'm',
    base_frame = scFrame,
    spacecraft_model = {
            'RTG' : {

        'file' : obj_path + 'rtg.obj',
        'frame_type' : 'User Defined',
        'frame_name' : scFrame,
        'center'     : [0,0,0],
        'diffuse'    : 0.1,
        'specular'   : 0.3,
        'UD_rotation': identity
        }
    }

    )

bodies = [bus, rtg, hga]
names = ['BUS', 'RTG', 'HGA']
temperatures = np.array([30, 210, 10], dtype = np.float64) # Object temperature in celsius
temperatures += 273.15 # Convert to kelvin
emissivities = np.array([.3, .3, .3])

ACC = np.zeros((3,3))

for i, b in enumerate(bodies):
    mesh = b.dump()
    #mesh.fix_normals() # BE sure that the normals are all outward-facing
    V, F, N, C, A = VFNC(mesh) # Vertices, Faces, Normals, Centers, Areas

    # Now it's a matter of simple computation.
    A = np.expand_dims(A, axis = 1)

    q = constants.stefan_boltzmann*emissivities[i]*temperatures[i]**4
    m = 1
    acc = np.multiply(-2/3*A/c*q/m, N)
    acc = np.sum(acc, axis = 0)
    ACC[i,:] = acc
```

Let's investigate the acceleration and force for each body

In [32]:
```python
for i,b in enumerate(bodies):
    print(f'{names[i]}\n')
    print(f'\t acceleration : {ACC[i,:]}')
    print(f'\t force {np.linalg.norm(ACC[i,:]*1000)}')
```

```
BUS

	 acceleration : [ 3.23117427e-27 -3.87740912e-26 -3.23117427e-26]
	 force 5.057587716941172e-23
RTG

	 acceleration : [-1.77714585e-26  1.42807168e-16  4.03896783e-27]
	 force 1.4280716804612186e-13
HGA

	 acceleration : [5.63436013e-26 9.78834388e-26 2.47540201e-10]
	 force 2.475402013159929e-07
```

In [ ]: