1 Rotation

1.1

$$\|\frac{(p+q)}{2}\| = \sqrt{(\frac{1}{\sqrt{2}})^2 + (\frac{i}{2\sqrt{2}})^2 + (\frac{j}{2\sqrt{2}})^2}$$

$$= \sqrt{\frac{1}{2} + \frac{1}{8} + \frac{1}{8}} = \frac{\sqrt{3}}{2}$$

$$scalar\ multiple : \frac{4}{3} * \frac{3}{4} = 1$$

$$\sqrt{\frac{4}{3}} = \frac{2}{\sqrt{3}}$$

$$\frac{2}{\sqrt{3}} * ((\frac{1}{\sqrt{2}}) + (\frac{i}{2\sqrt{2}}) + (\frac{j}{2\sqrt{2}})) = \frac{2}{\sqrt{6}} + \frac{i}{\sqrt{6}} + \frac{j}{\sqrt{6}}$$

$$Here,\ w = \frac{2}{\sqrt{6}},\ \ \vec{v} = [\frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, 0]$$

$$Rotation\ matrix M(r) = E(r)G(r)^\top$$

$$= \begin{bmatrix} \frac{2}{3} & \frac{1}{3} & \frac{2}{3} \\ \frac{1}{3} & \frac{2}{3} & -\frac{2}{3} \\ -\frac{2}{3} & \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

$$\theta = 2\arccos(w) = 2arccos(\frac{2}{\sqrt{6}}) = 70.5^\circ$$

$$\hat{\omega} = \frac{1}{\sin(\theta/2)}\vec{v} = \begin{bmatrix} 0.707 \\ 0.707 \\ 0 \end{bmatrix}$$

In [16]:
```
import math
q0 = 2/math.sqrt(6)
q1 = 1/math.sqrt(6)
q2 = 1/math.sqrt(6)
q3 = 0

r00 = 2 * (q0 * q0 + q1 * q1) - 1
r01 = 2 * (q1 * q2 - q0 * q3)
r02 = 2 * (q1 * q3 + q0 * q2)

r10 = 2 * (q1 * q2 + q0 * q3)
r11 = 2 * (q0 * q0 + q2 * q2) - 1
r12 = 2 * (q2 * q3 - q0 * q1)

r20 = 2 * (q1 * q3 - q0 * q2)
r21 = 2 * (q2 * q3 + q0 * q1)
r22 = 2 * (q0 * q0 + q3 * q3) - 1

np.array([[r00, r01, r02],
           [r10, r11, r12],
           [r20, r21, r22]])
```

Out[16]:
```
array([[ 0.66666667,  0.33333333,  0.66666667],
       [ 0.33333333,  0.66666667, -0.66666667],
       [-0.66666667,  0.66666667,  0.33333333]])
```

1.2

$$For\ p,\ w = \frac{1}{\sqrt{2}},\ \ \vec{v} = [\frac{1}{\sqrt{2}}, 0, 0]$$

$$\theta = 2\arccos(w) = 2arccos(\frac{1}{\sqrt{2}}) = 90^{\circ}$$

$$\hat{\omega} = \frac{1}{\sin(\theta/2)}\vec{v} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$p = (\hat{\omega}, \theta) = (\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \pi/2)$$

$$For\ q,\ w = \frac{1}{\sqrt{2}},\ \ \vec{v} = [0, \frac{1}{\sqrt{2}}, 0]$$

$$\theta = 2\arccos(w) = 2arccos(\frac{1}{\sqrt{2}}) = 90^{\circ}$$

$$\hat{\omega} = \frac{1}{\sin(\theta/2)}\vec{v} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$q = (\hat{\omega}, \theta) = (\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \pi/2)$$

1.3.a

$$[\omega_p] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$[\omega_q] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$Rot(\omega_p, \theta_p) = e^{[\omega_p \theta_p]} = I + [\hat{\omega_p}] sin\theta_p + [\hat{\omega_p}]^2 (1 - cos\theta_p)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$Rot(\omega_q, \theta_q) = e^{[\omega_q \theta_q]} = I + [\hat{\omega_q}] sin\theta_q + [\hat{\omega_q}]^2 (1 - cos\theta_q)$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

1.3.b

$$[\omega_p] + [\omega_q] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$\omega_{pq} = [1, 1, 0]$$

$$e^{[\omega_{pq}]} = I + e^{[\omega_p \theta_p]} = I + [\omega_{pq}/|\omega_{pq}|]\sin(|\omega_{pq}|) + [\omega_{pq}/|\omega_{pq}|]^2(1 - \cos(|\omega_{pq}|))$$

$$= I + \begin{bmatrix} 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix} \sin(\sqrt{2}) + \begin{bmatrix} 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix}^2 (1 - \cos(\sqrt{2}))$$

$$= \begin{bmatrix} \frac{1+\cos(\sqrt{2})}{2} & \frac{1-\cos(\sqrt{2})}{2} & \frac{\sin(\sqrt{2})}{\sqrt{2}} \\ \frac{1-\cos(\sqrt{2})}{2} & \frac{1+\cos(\sqrt{2})}{2} & -\frac{\sin(\sqrt{2})}{\sqrt{2}} \\ -\frac{\sin(\sqrt{2})}{\sqrt{2}} & \frac{\sin(\sqrt{2})}{\sqrt{2}} & \cos(\sqrt{2}) \end{bmatrix}$$

$$while \quad e^{[\omega_p]} e^{[\omega_q]} = \begin{bmatrix} \cos(1) & 0 & \sin(1) \\ \sin^2(1) & \cos(1) & \sin(1)\cos(1) \\ -\sin(1)\cos(1) & \sin(1) & \cos^2(1) \end{bmatrix}$$

Clearly, they are different

1.3.c To find $\Delta\omega$, I first initiated $R := I$

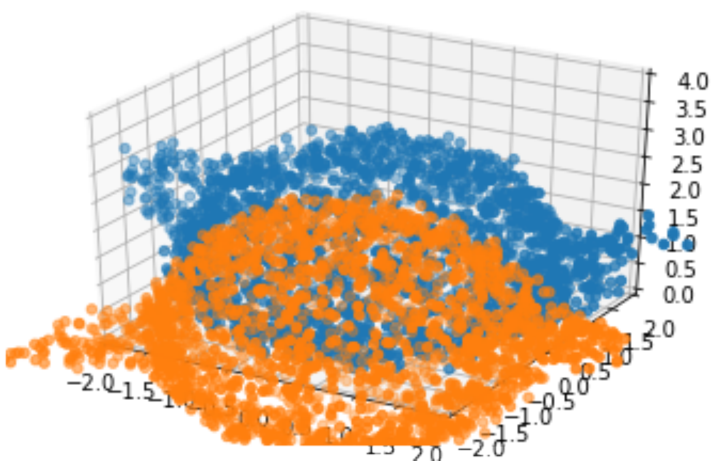
then, A should be a dimension of 6000 * 3 since we are stacking 2000 3 * 3 rotation matrix

Update R by $R := R\exp([\Delta\omega])$ for 100 times which is

$$-\begin{bmatrix} R[X_1] \\ R[X_2] \\ \cdots \\ R[X_n] \end{bmatrix} \Delta w + \begin{bmatrix} R[X_1] - Y_1 \\ R[X_2] - Y_2 \\ \cdots \\ R[X_n] - Y_n \end{bmatrix}$$

In [196]:
```
# Note Matplotlib is only suitable for simple 3D visualization.
# For later problems, you should not use Matplotlib to do the plotting
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
def show_points(points):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points[0], points[2], points[1])

def compare_points(points1, points2):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points1[0], points1[2], points1[1])
    ax.scatter(points2[0], points2[2], points2[1])
```

In [4]:
```
npz = np.load('HW1_P1.npz')
X = npz['X']
Y = npz['Y']
compare_points(X, Y)  # noisy teapotsand
```



In [5]:
```
A = X
b = Y
#start = 1e-2
#x = np.linalg.pinv(A.T@A + 2*start*np.eye(2000)) @A.T @ b
d, U = np.linalg.eigh(A.T@A)
k = U.T@(A.T@b)
lam = 0.002
x = U@(np.diag(1/(d + 2 * lam))@(U.T@(A.T@b)))
```

In [6]:
```
#x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)
x.shape
```

Out[6]:
```
(2000, 2000)
```

In [7]:
```
# copy-paste your hw0 solve module here
def hw0_solve(A, b, eps):
    #x = np.zeros(A.shape[1])
    x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)
    # case 1
    if (x @ x).sum() < eps:
        return x
    # case 2
    d, U = np.linalg.eigh(A.T@A)
    k = U.T@(A.T@b)
    def func(lam):
        return ((k / (d + 2 * lam))**2).sum() - eps

    # find a valid pair func(a) > 0, func(b) < 0
    lo = 0
    hi = 1
    while func(hi) > 0:
            lo, hi = hi, hi * 2
    # bisect
    thres = 0.0001
    while True:
        mi = (lo+hi) / 2
        v = func(mi)
        if abs(hi-lo) < thres:
            break
        if v > 0:
            lo = mi
        else:
            hi = mi
    lam = mi
    x = U@(np.diag(1/(d + 2 * lam))@(U.T@(A.T@b)))
    return x
```

In [135]:
```
R1 = np.eye(3)
# solve this problem here, and store your final results in R1
for __ in range(100):
    pass
```

In [ ]:
```
# Testing code, you should see the points of the 2 teapots roughly ove
compare_points(R1@X, Y)
R1.T@R1
```

1.4.a

$$For\ p',\ w = \frac{-1}{\sqrt{2}},\quad \vec{v} = [\frac{-1}{\sqrt{2}}, 0, 0]$$

$$\theta = 2\arccos(w) = 2arccos(\frac{-1}{\sqrt{2}}) = 270^{\circ}$$

$$\hat{\omega} = \frac{1}{\sin(\theta/2)}\vec{v} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

$$p' = (\hat{\omega}, \theta) = (\begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, 3\pi/2)$$

$$For\ q',\ w = \frac{-1}{\sqrt{2}},\quad \vec{v} = [0, \frac{-1}{\sqrt{2}}, 0]$$

$$\theta = 2\arccos(w) = 2arccos(\frac{1}{\sqrt{2}}) = 270^{\circ}$$

$$\hat{\omega} = \frac{1}{\sin(\theta/2)}\vec{v} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

$$q' = (\hat{\omega}, \theta) = (\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, 3\pi/2)$$

Comparing we get the exponential coordinates of p' is the negative of p. Same holds for q. So for general quaternion pair (r, –r) the negative of quaternion is have the negative exponential coordinate which means rotate the negative direction for $2\pi - \theta$ angle
In general: $r = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$, and $r' = -r = -w - x\mathbf{i} - y\mathbf{j} - z\mathbf{k}$.

$\theta_{r'} = 2\cos^{-1}(-w) = 2(\pi - \cos^{-1}(w)) = 2\pi - 2\cos^{-1}(w) = 2\pi - \theta_r$

$\theta_{r'}/2 = \pi - \frac{\theta_r}{2}$

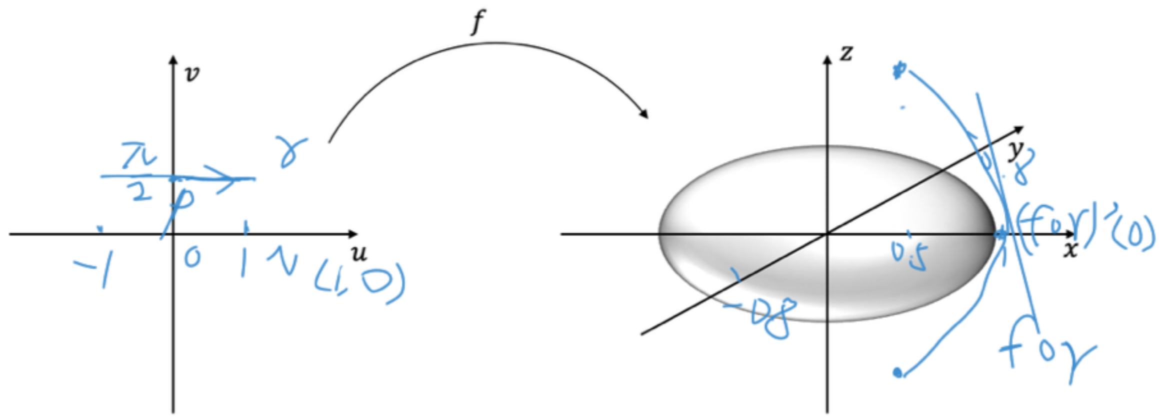
$\hat{\omega}_{r'} = \frac{1}{\sin(\pi - \frac{\theta_r}{2})}(-w_1, -w_2, -w_3) = \frac{1}{\sin(\frac{\theta_r}{2})}(-w_1, -w_2, -w_3) = -\frac{1}{\sin(\frac{\theta_r}{2})}(w_1, w_2, w_3) =$

Thus in general, the exponential coordinates of $-r$ are $(-\hat{\omega}_r, 2\pi - \theta_r)$.

1.4.b No we cannot use L2 distance. Using L2 distance cannot distinguish between r and -r. The same rotation can be achieved and represented by 2 different quaternions. The neural network can predicts the correct rotation but by the negative of the ground truth, L2 norm would not be accurate for representing distance in rotation.

2 Geometry

2.1




```
In [1]: a, b, c = 1, 1, 0.5
```

In [10]:
```
!pip install open3d
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
(https://pypi.tuna.tsinghua.edu.cn/simple)
Requirement already satisfied: open3d in /opt/anaconda3/lib/python3.7
/site-packages (0.14.1)
Requirement already satisfied: numpy>=1.18.0 in /opt/anaconda3/lib/py
thon3.7/site-packages (from open3d) (1.19.4)
Requirement already satisfied: addict in /opt/anaconda3/lib/python3.7
/site-packages (from open3d) (2.4.0)
Requirement already satisfied: matplotlib>=3 in /opt/anaconda3/lib/py
thon3.7/site-packages (from open3d) (3.4.2)
Requirement already satisfied: scikit-learn>=0.21 in /opt/anaconda3/l
ib/python3.7/site-packages (from open3d) (1.0.1)
Requirement already satisfied: pillow>=8.2.0 in /opt/anaconda3/lib/py
thon3.7/site-packages (from open3d) (9.0.0)
Requirement already satisfied: setuptools>=40.8.0 in /opt/anaconda3/l
ib/python3.7/site-packages (from open3d) (46.0.0.post20200309)
Requirement already satisfied: pyyaml>=5.4.1 in /opt/anaconda3/lib/py
thon3.7/site-packages (from open3d) (5.4.1)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.7/s
ite-packages (from open3d) (4.42.1)
Requirement already satisfied: wheel>=0.36.0 in /opt/anaconda3/lib/py
thon3.7/site-packages (from open3d) (0.37.0)
Requirement already satisfied: pandas>=1.0 in /opt/anaconda3/lib/pyth
on3.7/site-packages (from open3d) (1.0.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda3
/lib/python3.7/site-packages (from matplotlib>=3->open3d) (2.8.1)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/anaconda3/lib
/python3.7/site-packages (from matplotlib>=3->open3d) (2.4.6)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda3/li
b/python3.7/site-packages (from matplotlib>=3->open3d) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/pyt
hon3.7/site-packages (from matplotlib>=3->open3d) (0.10.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3
/lib/python3.7/site-packages (from scikit-learn>=0.21->open3d) (2.2.0
)
Requirement already satisfied: scipy>=1.1.0 in /opt/anaconda3/lib/pyt
hon3.7/site-packages (from scikit-learn>=0.21->open3d) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/pyt
hon3.7/site-packages (from scikit-learn>=0.21->open3d) (0.14.1)
Requirement already satisfied: pytz>=2017.2 in /opt/anaconda3/lib/pyt
hon3.7/site-packages (from pandas>=1.0->open3d) (2019.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3
.7/site-packages (from python-dateutil>=2.7->matplotlib>=3->open3d) (
1.16.0)
```

In [2]:
```
# These are some convenient functions to create open3d geometries and
# The viewing direction is fine-tuned for this problem, you should not
import open3d
```

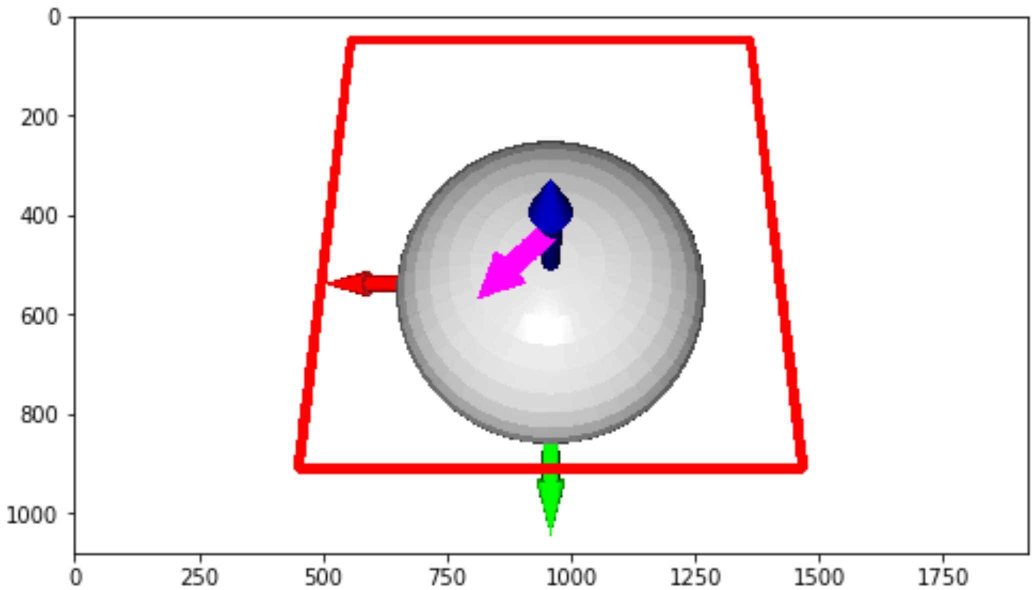
```
import open3d
import numpy as np
import matplotlib.pyplot as plt

vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)

def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    view_ctl.set_up((0, 1e-4, 1))
    view_ctl.set_front((0, 0.5, 2))
    view_ctl.set_lookat((0, 0, 0))
    # do not change this view point
    vis.update_renderer()
    img = vis.capture_screen_float_buffer(True)
    plt.figure(figsize=(8,6))
    plt.imshow(np.asarray(img))
    for g in geoms:
        vis.remove_geometry(g)

def create_arrow_from_vector(origin, vector):
    '''
    origin: origin of the arrow
    vector: direction of the arrow
    '''
    v = np.array(vector)
    v /= np.linalg.norm(v)
    z = np.array([0,0,1])
    angle = np.arccos(z@v)

    arrow = open3d.geometry.TriangleMesh.create_arrow(0.05, 0.1, 0.25,
    arrow.paint_uniform_color([1,0,1])
    T = np.eye(4)
    T[:3, 3] = np.array(origin)
    T[:3,:3] = open3d.geometry.get_rotation_matrix_from_axis_angle(np.
    arrow.transform(T)
    return arrow

def create_ellipsoid(a,b,c):
    sphere = open3d.geometry.TriangleMesh.create_sphere()
    sphere.transform(np.diag([a,b,c,1]))
    sphere.compute_vertex_normals()
    return sphere


def create_lines(points):
    lines = []
    for p1, p2 in zip(points[:-1], points[1:]):
        height = np.linalg.norm(p2-p1)
```

```
        center = (p1+p2) / 2
        d = p2-p1
        d /= np.linalg.norm(d)
        axis = np.cross(np.array([0,0,1]), d)
        axis /= np.linalg.norm(axis)
        angle = np.arccos(np.array([0,0,1]) @ d)
        R = open3d.geometry.get_rotation_matrix_from_axis_angle(axis *

        T = np.eye(4)
        T[:3,:3]=R
        T[:3,3] = center
        cylinder = open3d.geometry.TriangleMesh.create_cylinder(0.02,
        cylinder.transform(T)
        cylinder.paint_uniform_color([1,0,0])
        lines.append(cylinder)
    return lines
```

```
Bad key "text.kerning_factor" on line 4 in
/opt/anaconda3/lib/python3.7/site-packages/matplotlib/mpl-data/stylel
ib/_classic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.tem
plate
(https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.te
mplate)
or from the matplotlib source distribution
```

In [3]:
```
# exapmle code to draw ellipsoid, curve, and arrows
#r = (t+math.pi/4, math.pi/6)
arrow = create_arrow_from_vector([0.,0.,1.], [1.,1.,0.])
import math
ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
curve = create_lines(np.array([[1,1,1], [-1,1,1], [-1,-1,1], [1,-1,1],
draw_geometries([ellipsoid, cf, arrow] + curve)
```

```
WARNING - 2022-01-31 14:37:52,962 - image - Clipping input data to th
e valid range for imshow with RGB data ([0..1] for floats or [0..255]
for integers).
```

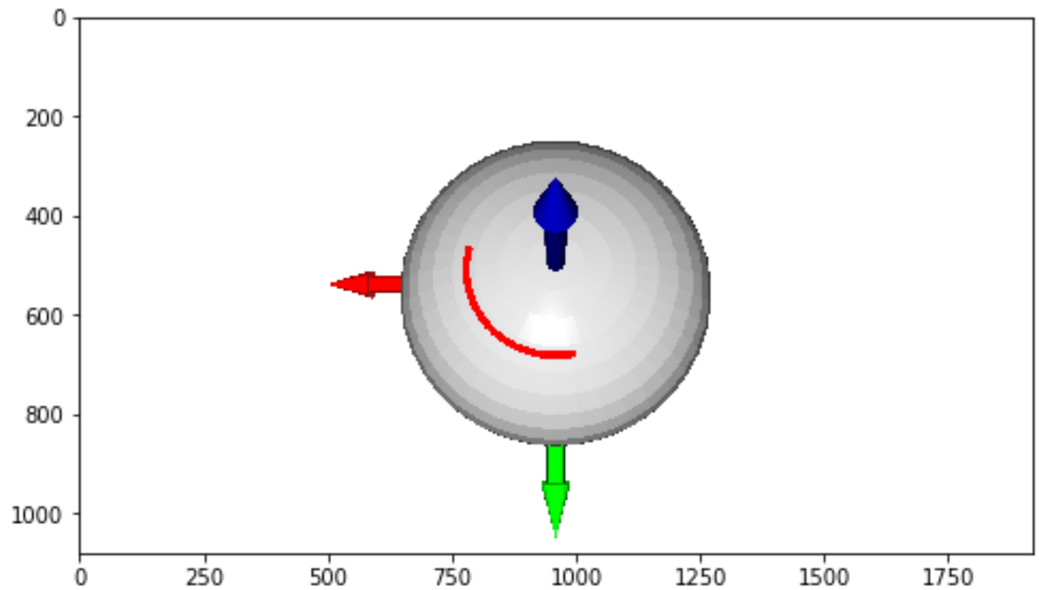


In [ ]:

2.2 your solution here

In [4]:
```
import math
ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
points = []
for t in np.linspace(-1, 1):
    points.append([a*math.cos(t+math.pi/4)*math.sin(math.pi/6),
                   b*math.sin(t+math.pi/4)*math.sin(mat
                   c*math.cos(math.pi/6)])
curve = create_lines(np.array(points))
draw_geometries([ellipsoid, cf] + curve)
```

```
WARNING - 2022-01-31 14:37:57,484 - image - Clipping input data to th
e valid range for imshow with RGB data ([0..1] for floats or [0..255]
for integers).
```



2.3.a

$$Df_p = \begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial v} \\ \frac{\partial f_2}{\partial u} & \frac{\partial f_2}{\partial v} \\ \frac{\partial f_3}{\partial u} & \frac{\partial f_3}{\partial v} \end{bmatrix} = \begin{bmatrix} -a \sin u \sin v & a \cos u \cos v \\ b \cos u \sin v & b \sin u \cos v \\ 0 & -c \sin v \end{bmatrix}$$

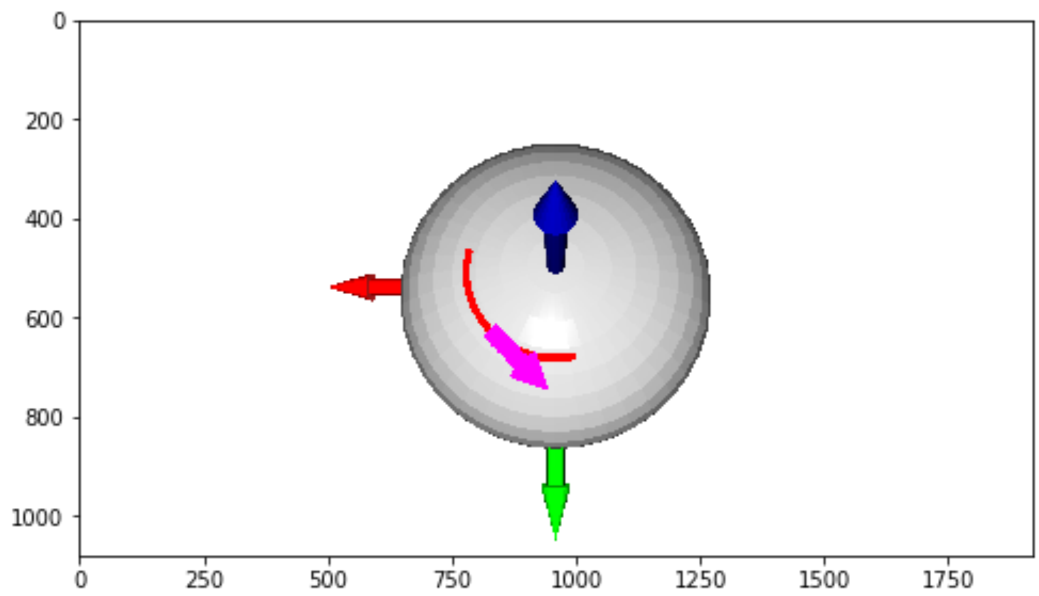
2.3.b it maps a vector in the tangent space of the domain to the tangent space of the surface

2.3.c

In [ ]:

In [5]:
```
ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
arrow = create_arrow_from_vector(np.array([a*math.cos(math.pi/4)*math.
                                           b* math.sin(math.pi/4)*math
                                          c*math.cos(math.pi/6)]),
    np.array([[-a*math.sin(math.pi/4)*math.sin(math.pi/6),
                              b*math.cos(math.pi/4)*math.sin(math.
                           [a*math.cos(math.pi/4)*math.cos(math.pi
                              b*math.sin(math.pi/4)*math.cos(math.
                              -c*math.sin(math.pi/6)]]).T @np.arra
draw_geometries([ellipsoid, cf, arrow] + curve)
```

```
WARNING - 2022-01-31 14:38:01,731 - image - Clipping input data to th
e valid range for imshow with RGB data ([0..1] for floats or [0..255]
for integers).
```



2.3.d

$$N = \begin{bmatrix} -a\sin u \sin v \\ b\cos u \sin v \\ 0 \end{bmatrix} \times \begin{bmatrix} a\cos u\cos v \\ b\sin u \cos v \\ -c\sin v \end{bmatrix}$$

$$= \begin{bmatrix} -bc\cos u \sin^2 v \\ -ac\sin u\sin^2 v \\ -ab\sin^2 u \sin v\cos v - ab\cos^2 u\cos v\sin v \end{bmatrix} = \begin{bmatrix} -bc\cos u\sin^2 v \\ -ac\sin u \sin^2 v \\ -ab\sin v\cos v \end{bmatrix} / \sqrt{c^2\sin^4 v + a}$$

2.3.e

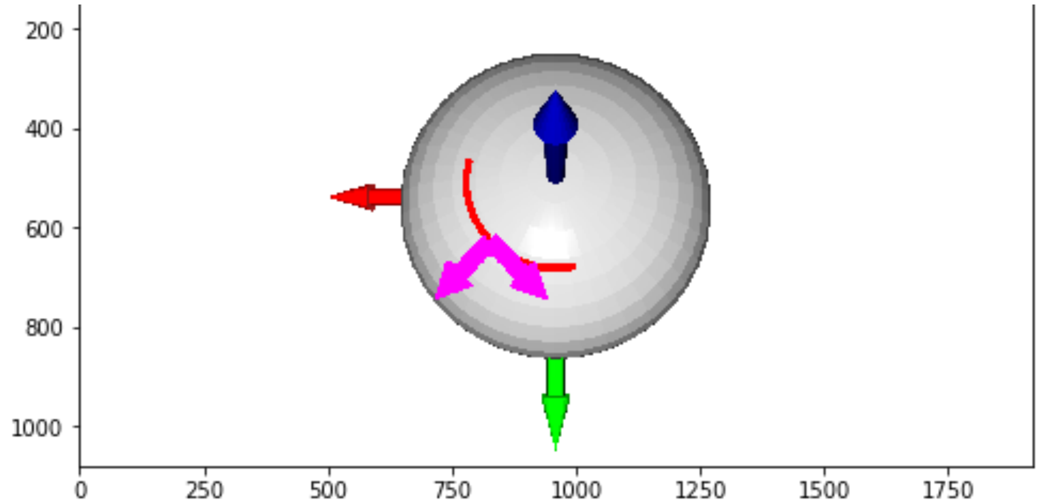
$$\left\{\begin{bmatrix} -a\sin u \\ b\cos u \\ 0 \end{bmatrix}, \begin{bmatrix} a\cos u\cos v \\ b\sin u\cos v \\ -c\sin v \end{bmatrix} / \sqrt{a^2\cos^2 v + c^2\sin^2 v}\right\}$$

In [6]:

```
ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
p = np.array([a*math.cos(math.pi/4)*math.sin(math.pi/6),
                                            b* math.sin(math.pi/4)*math
                                           c*math.cos(math.pi/6)])
u = np.pi / 4
v = np.pi / 6

normal = np.array([- b * c * np.cos(u) * (np.sin(v) ** 2),
                   - a * c * np.sin(u) * (np.sin(v) ** 2), - a * b * n
normal /= np.sqrt( ((b ** 2) * (c ** 2) * (np.cos(u) ** 2) * (np.sin(v
                  ((a ** 2) * (c ** 2) * (np.sin(u) ** 2) * (np.sin(v)
                  ((a ** 2) * (b ** 2) * (np.sin(v) ** 2) * (np.cos(v)

Dfp = np.array([[-a * np.sin(u) * np.sin(v), a * np.cos(u) * np.cos(v)
                [b * np.cos(u) * np.sin(v), b * np.sin(u) * np.cos(v)]
                [0, -c * np.sin(v)]])


tangent = (Dfp @ np.array([[1, 0]]).T).flatten()
basis1 = tangent / np.linalg.norm(tangent)
basis2 = np.cross(tangent, normal)
basis2 /= np.linalg.norm(basis2)
print("First Basis:", basis1)
print("Second Basis:", basis2)

arrow_base_1 = create_arrow_from_vector(p, np.array([-a*math.sin(math.
                                  b*math.cos(math.pi/4),0]))
arrow_base_2 = create_arrow_from_vector(p, np.array([a*math.cos(math.p
                                  b*math.sin(math.pi/4 )*math.cos(math
                                        /math.sqrt(math.cos(math.pi/6)
draw_geometries([ellipsoid, cf, arrow_base_1, arrow_base_2] + curve)
```

```
First Basis: [-0.70710678  0.70710678  0.        ]
Second Basis: [-0.67936622 -0.67936622  0.2773501 ]
```

```
WARNING - 2022-01-31 14:38:05,876 - image - Clipping input data to th
e valid range for imshow with RGB data ([0..1] for floats or [0..255]
for integers).
```



2.4.a

$$
\begin{aligned}
s(t) &= \int_0^t \|\gamma'(t)\| dt = \int_0^t \sqrt{v^T (Df_p^T Df_p) v}\, dt \\
&= \int_0^t (\sin^2 u \sin^2 v + \cos^2 u \sin^2 v)^{1/2} dt \\
&= \int_0^t (\sin^2 v)^{1/2} dt \\
&= \int_0^t \sin \frac{\pi}{6} dt \\
&= t/2
\end{aligned}
$$

2.4.b

$$
\begin{gathered}
t(s) = 2s \\
\gamma(t(s)) = (\frac{\pi}{4} + 2s, 2s) \\
h_v(s) = \gamma(t(s)) = \gamma(s(t)^{-1}) = \\
\begin{bmatrix} \cos(\frac{\pi}{4} + 2s)\sin(2s) \\ \sin(\frac{\pi}{4} + 2s)\sin(2s) \\ c\cos(2s) \end{bmatrix}
\end{gathered}
$$

2.4.c

$$N = \frac{dT}{ds} = d\begin{bmatrix} 2\cos(\frac{\pi}{4} + 2s)\cos(2s) - 2\sin(\frac{\pi}{4} + 2s)\sin(2s) \\ 2\cos(\frac{\pi}{4} + 2s)\sin(2s) + 2\sin(\frac{\pi}{4} + 2s)\cos(2s) \\ -2c\sin(2s) \end{bmatrix} /ds$$

$$= \begin{bmatrix} -8\cos(\frac{\pi}{4} + 2s)\sin(2s) - 8\sin(\frac{\pi}{4} + 2s)\cos(2s) \\ -8\sin(\frac{\pi}{4} + 2s)\sin(2s) + 8\cos(\frac{\pi}{4} + 2s)\cos(2s) \\ -4c\cos(2s) \end{bmatrix}$$

2.5.a

$$N_p = \begin{bmatrix} \frac{-bc\cos u\sin^2 v}{\sqrt{c^2\sin^4 v + a^2 b^2 \sin^2 v\cos^2 v}} \\ \frac{-ac\sin u\sin^2 v}{\sqrt{c^2\sin^4 v + a^2 b^2 \sin^2 v\cos^2 v}} \\ \frac{-ab\sin v\cos v}{\sqrt{c^2\sin^4 v + a^2 b^2 \sin^2 v\cos^2 v}} \end{bmatrix}$$

$$DN_p = \begin{bmatrix} \frac{\sin(u)\sin^2(v)}{\sqrt{\sin^4(v)+\sin^2(2v)}} & \frac{2\cos(u)\sin(2v)}{(\sin^2(v)+4\cos^2(v))\sqrt{\sin^4(v)+\sin^2(2v)}} \\ -\frac{\cos(u)\sin^2(v)}{\sqrt{\sin^4(v)+\sin^2(2v)}} & \frac{2\sin(u)\sin(2v)}{(\sin^2(v)+4\cos^2(v))\sqrt{\sin^4(v)+\sin^2(2v)}} \\ 0 & \frac{2\sin^2(v)}{(\sin^2(v)+4\cos^2(v))\sqrt{\sin^4(v)+\sin^2(2v)}} \end{bmatrix}$$

2.5.b

$$S = Df_p^\top DN_p$$

In [8]:
```
u = np.pi / 4
v = np.pi / 6

denom1 = np.sqrt((np.sin(v) ** 4) + (np.sin(2 * v) ** 2))
denom2 = (np.sin(v) ** 2) + (4 * (np.cos(v) ** 2))

DNp = np.array([[np.sin(u) * (np.sin(v) ** 2) / denom1, 2 * np.cos(u)
                [- np.cos(u) * (np.sin(v) ** 2) / denom1, 2 * np.sin(u
                [0, 2 * (np.sin(v) ** 2) / (denom1 * denom2)]])


S = Dfp.T @ DNp
print(S)
print(np.linalg.eig(S)[1])
```

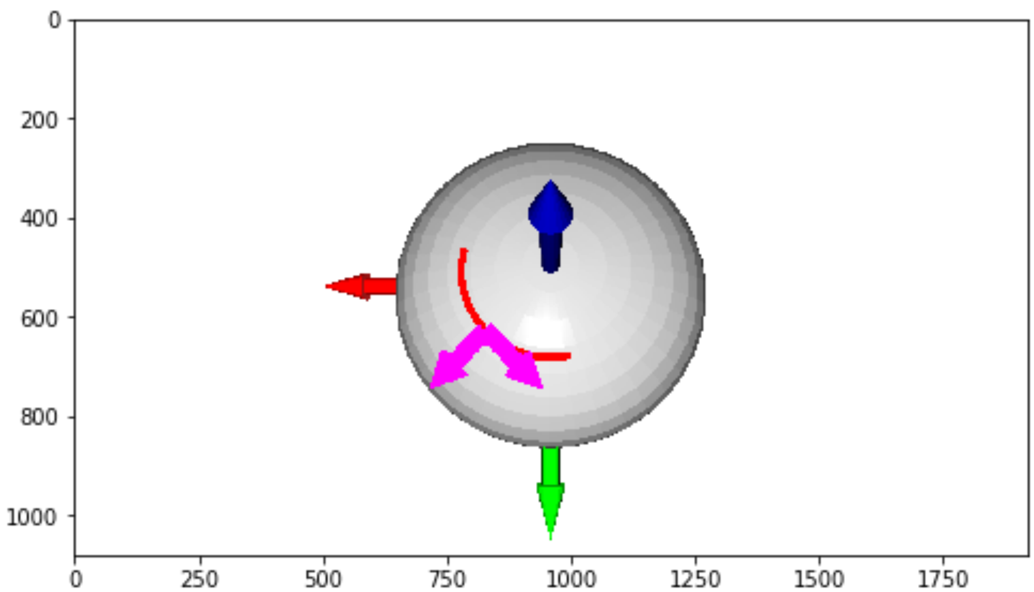
```
[[-1.38675049e-01  0.00000000e+00]
 [ 1.38777878e-17  4.69361704e-01]]
[[ 0.00000000e+00  1.00000000e+00]
 [ 1.00000000e+00 -2.28239292e-17]]
```

The eigenvectors are [1,0] and [0,1]

2.5.c

In [20]:
```
V = np.array([[1, 0]])
principal1 = Dfp @ np.array([1, 0])
principal2 = Dfp @ np.array([0, 1])
fp = np.array([a * np.cos(u) * np.sin(v), b * np.sin(u) * np.sin(v), c
ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
p1_vec = create_arrow_from_vector(fp, principal1)
p2_vec = create_arrow_from_vector(fp, principal2)
draw_geometries([ellipsoid, cf, p1_vec, p2_vec] + curve)
```

```
WARNING - 2022-01-31 15:01:29,408 - image - Clipping input data to th
e valid range for imshow with RGB data ([0..1] for floats or [0..255]
for integers).
```



2.5.d There are orthorgonal to each other and same as the normal vector of the tangent plane. One indicates the fastest change of bending and the other is the smallest. Since here we have a line as curvature, it's the same as the normal vector.

3 Mesh

3.1 T_1 and T_2 are principal curvature directions, $t_\theta = \cos\theta T_1 + \sin\theta T_2$
So, t_θ is always in the tangent space, then normal vector N must be an eigenvector with eigenvalue 0.

3.2

$$M = [T_1, T_2]^\top \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} [T_1, T_2]$$
$$since\ m_{12} = m_{21} by\ symmetry$$
$$m_{12} = T_1^\top M T_2$$
$$= \frac{\kappa_1}{2\pi} \int_{-\pi}^{\pi} \cos^3(\theta)\sin(\theta) d\theta + \frac{\kappa_2}{2\pi} \int_{-\pi}^{\pi} \cos(\theta)\sin^3(\theta) d\theta = 0$$

Since both integrands are odd functions, the matrix of m is then diagonal. the remaining eigenvectors are T1 and T2

To verify:

$$m_1 1 = T_1^\top M T_1$$
$$= \frac{\kappa_1}{2\pi} \int_{-\pi}^{\pi} \cos^4(\theta) d\theta + \frac{\kappa_2}{2\pi} \int_{-\pi}^{\pi} \cos^2(\theta)\sin^2(\theta) d\theta$$
$$= \frac{3}{8}\kappa_1 + \frac{1}{8}\kappa_2$$
$$m_2 2 = T_2^\top M T_2$$
$$= \frac{\kappa_1}{2\pi} \int_{-\pi}^{\pi} \cos^2(\theta)\sin^2(\theta) d\theta + \frac{\kappa_2}{2\pi} \int_{-\pi}^{\pi} \sin^4(\theta) d\theta$$
$$= \frac{1}{8}\kappa_1 + \frac{3}{8}\kappa_2$$

3.3

In [142]:
```
# You may want to restart your notebook here, to reinitialize Open3D

import open3d
import numpy as np
import matplotlib.pyplot as plt

vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)

# Make sure you call this function to draw the points for proper viewi
def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    view_ctl.set_up((0, 1, 0))
    view_ctl.set_front((0, 2, 1))
    view_ctl.set_lookat((0, 0, 0))
    view_ctl.set_zoom(1)
    # do not change this view point
    vis.update_renderer()
    img = vis.capture_screen_float_buffer(True)
    plt.figure(figsize=(8,6))
    plt.imshow(np.asarray(img))
    for g in geoms:
        vis.remove_geometry(g)
```

In [3]:
```
! pip install trimesh
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
(https://pypi.tuna.tsinghua.edu.cn/simple)
Collecting trimesh
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/91/e6/8b5994
b322279ba265ca38909d88d5b5c292785c2feef007a981db5b4c85/trimesh-3.9.43
-py3-none-any.whl
(https://pypi.tuna.tsinghua.edu.cn/packages/91/e6/8b5994b322279ba265c
a38909d88d5b5c292785c2feef007a981db5b4c85/trimesh-3.9.43-py3-none-any
.whl) (640 kB)
     |████████████████████████████████| 640 kB 675 kB/s eta 0:00:01
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.7/
site-packages (from trimesh) (1.19.4)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/pytho
n3.7/site-packages (from trimesh) (46.0.0.post20200309)
Installing collected packages: trimesh
Successfully installed trimesh-3.9.43
```

In [ ]:

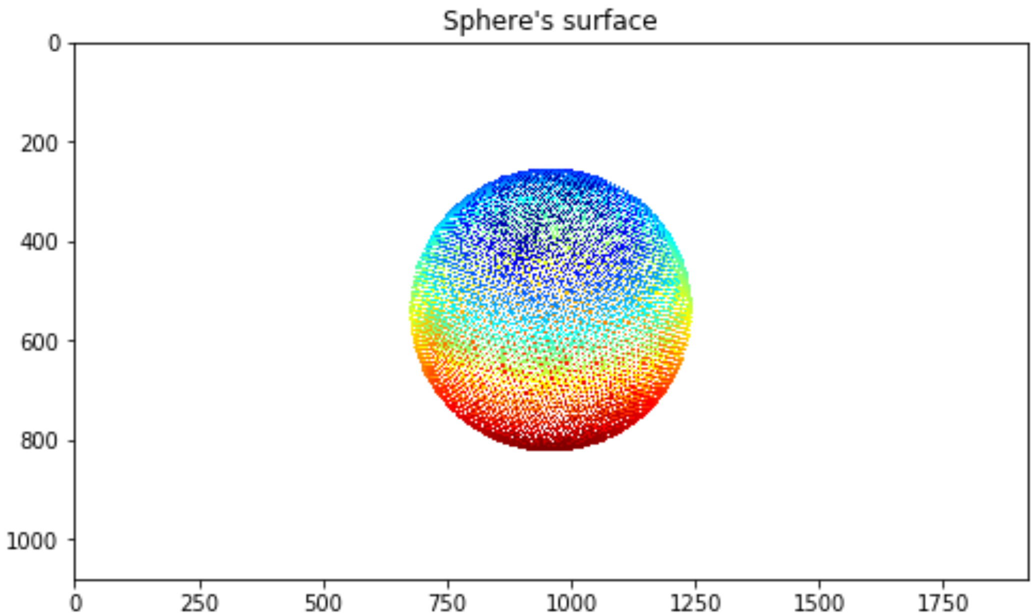
In [111]:

```
import trimesh
ico_mesh = trimesh.load('icosphere.obj')
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(ico_mesh.vertices)
draw_geometries([pcd])
plt.title("Sphere's surface")
```

```
WARNING - 2022-01-31 12:23:23,877 - obj - unable to load materials fr
om: icosphere.mtl
```

Out[111]:

```
Text(0.5, 1.0, "Sphere's surface")
```

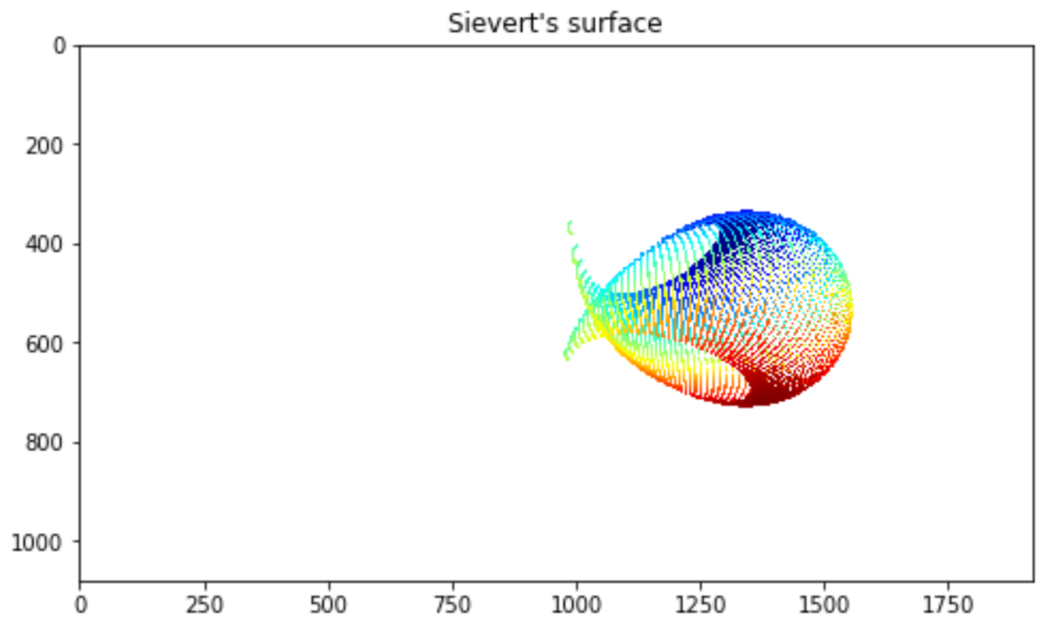


In [ ]:

In [112]:
```python
def get_principal_curvature(FV):
    FaceNormals = FV.face_normals
    VertexNormals = FV.vertex_normals
    principal_curvatures = []

    for i, face in enumerate(FV.faces):
        v0 = FV.vertices[face[0]]
        v1 = FV.vertices[face[1]]
        v2 = FV.vertices[face[2]]

        e0 = v2 - v1
        e1 = v0 - v2
        e2 = v1 - v0

        n0 = VertexNormals[face[0]]
        n1 = VertexNormals[face[1]]
        n2 = VertexNormals[face[2]]

        face_norm = FaceNormals[i]

        xi_u = e0 / np.linalg.norm(e0)
        xi_v = np.cross(face_norm, xi_u)
        xi_v /= np.linalg.norm(xi_v)

        A = np.array([[np.dot(xi_u, e0), np.dot(xi_v, e0), 0, 0],
                       [0, 0, np.dot(xi_u, e0), np.dot(xi_v, e0)],
                     [np.dot(xi_u, e1), np.dot(xi_v, e1), 0, 0],
                     [0, 0, np.dot(xi_u, e1), np.dot(xi_v, e1)],
                     [np.dot(xi_u, e2), np.dot(xi_v, e2), 0, 0],
                     [0, 0, np.dot(xi_u, e2), np.dot(xi_v, e2)]])

        b = np.array([np.dot(xi_u, n2 - n1), np.dot(xi_v, n2 - n1),
                      np.dot(xi_u, n0 - n2), np.dot(xi_v, n0 - n2),
                      np.dot(xi_u, n1 - n0), np.dot(xi_v, n1 - n0)])

        s, _, _, _ = np.linalg.lstsq(A, b, None)

        s = np.reshape(s, (2, 2))

        curvs, _ = np.linalg.eig(s)

        principal_curvatures.append(curvs)

    return np.array(principal_curvatures).real
```

In [113]:
```
sievert_mesh = trimesh.load('sievert.obj')
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(sievert_mesh.vertices)
draw_geometries([pcd])
plt.title("Sievert's surface")
```

Out[113]:
```
Text(0.5, 1.0, "Sievert's surface")
```



In [114]:
```
get_principal_curvature(ico_mesh)[:4]
```

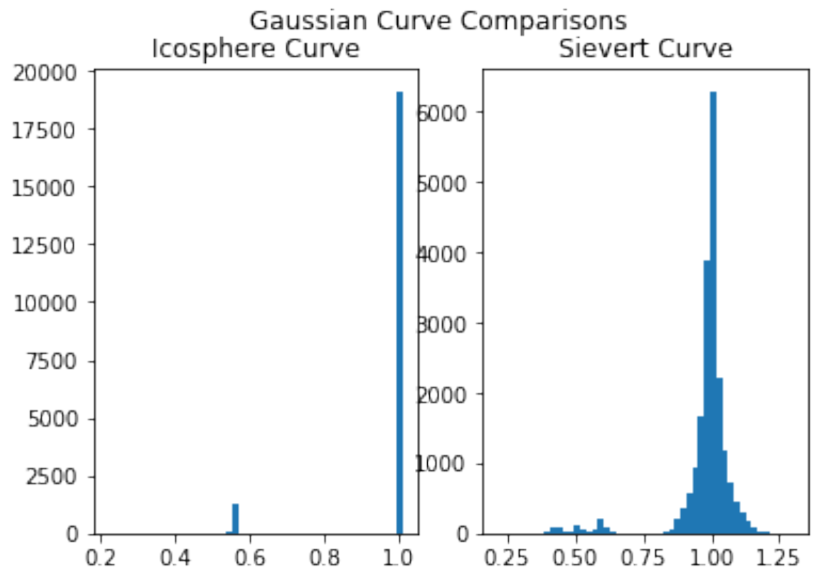
Out[114]:
```
array([[0.40159081, 0.55386776],
       [0.85890107, 1.00113216],
       [0.40158038, 0.55385584],
       [0.40158038, 0.55385584]])
```

3.4

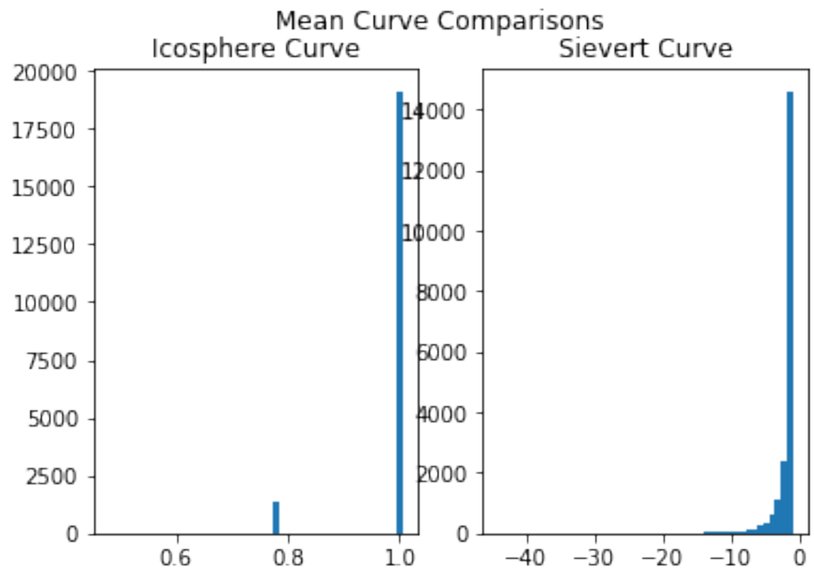
In [124]:
```
ico_cur = get_principal_curvature(ico_mesh)
ico_gaussian_curv = ico_cur[:, 0] * ico_cur[:, 1]
ico_mean_curv = 0.5 * (ico_cur[:,0] +ico_cur[:,1])
```

In [125]:
```
sievert_cur = get_principal_curvature(sievert_mesh)
sievert_gaussian_curv = sievert_cur[:,0] * sievert_cur[:,1]
sievert_mean_curv = 0.5 * (sievert_cur[:,0] +sievert_cur[:,1])
```

In [130]:
```
fig, (ax1, ax2) = plt.subplots(1, 2)

fig.suptitle("Gaussian Curve Comparisons")
ax1.set_title("Icosphere Curve")
ax1.hist(ico_gaussian_curv, 50)
ax2.set_title("Sievert Curve")
ax2.hist(sievert_gaussian_curv, 50)
plt.show()
```

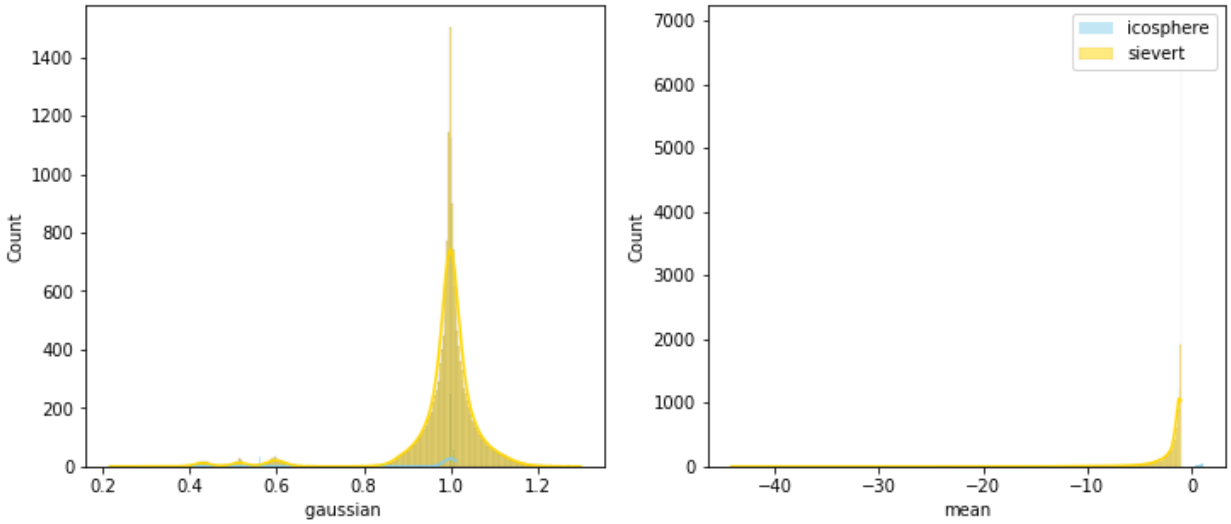


In [131]:
```
fig, (ax1, ax2) = plt.subplots(1, 2)

fig.suptitle("Mean Curve Comparisons")
ax1.set_title("Icosphere Curve")
ax1.hist(ico_mean_curv, 50)
ax2.set_title("Sievert Curve")
ax2.hist(sievert_mean_curv, 50)
plt.show()
```



In [127]:
```
# Give a overall more comparison
import seaborn as sns
def plot_stats(ico_g, sievert_g, ico_mean, sievert_mean):
    f = plt.figure(figsize=(12,5))
    ax = f.add_subplot(121)
    ax2 =f.add_subplot(122)
    ax.set_xlabel("gaussian ")
    ax2.set_xlabel("mean")
    #fig, axs = plt.subplots(1, 2, figsize=(10, 7))
    sns.histplot(ico_g,  kde=True, color = "skyblue", label="icosphere
    sns.histplot(sievert_g, label="sievert",  kde=True, color = "gold"
    sns.histplot(ico_mean, label="icosphere",kde=True, color = "skyblu
    sns.histplot(sievert_mean, label="sievert",  kde=True, color = "go
    plt.legend(loc="upper right")
    plt.show()
```

In [128]:
```
plot_stats(ico_gaussian_curv, sievert_gaussian_curv, ico_mean_curv, si
```



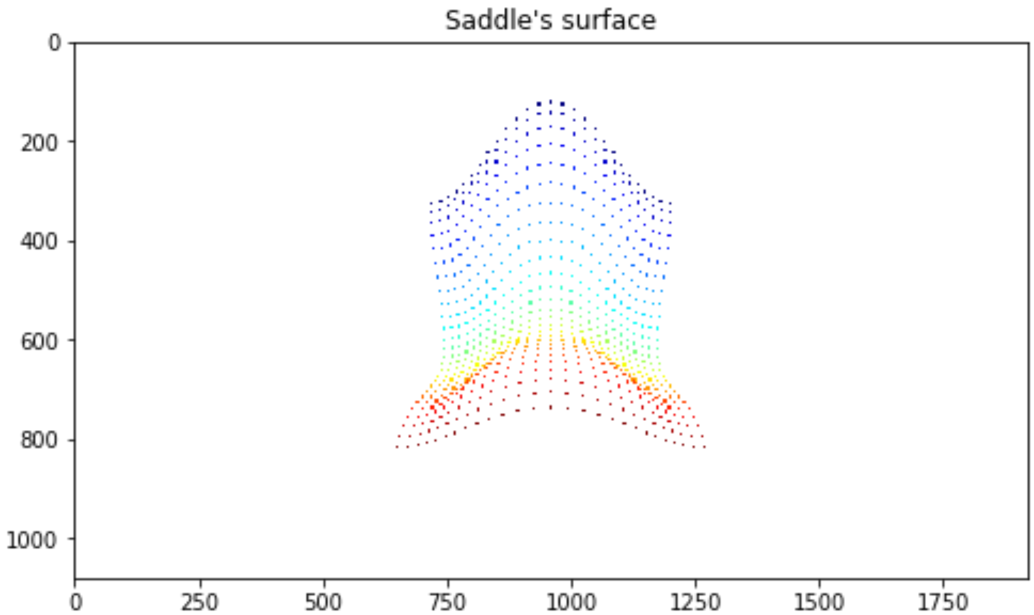
4 Point Cloud

4.1

In [148]:
```
saddle_mesh = trimesh.load('saddle.obj')
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(saddle_mesh.vertices)
draw_geometries([pcd])
plt.title("Saddle's surface")
```

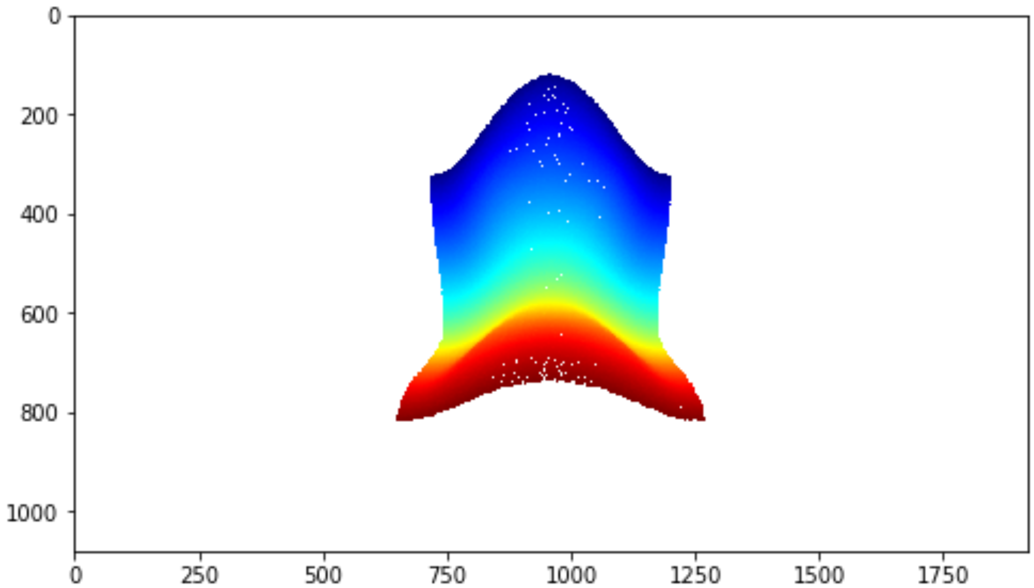
```
INFO - 2022-01-31 13:31:42,311 - base - triangulating quad faces
```

Out[148]: `Text(0.5, 1.0, "Saddle's surface")`



In [149]:
```
points, face_idx = trimesh.sample.sample_surface(saddle_mesh, 100000)
```

In [150]:
```
saddle_pcd = open3d.geometry.PointCloud()
saddle_pcd.points = open3d.utility.Vector3dVector(points)

draw_geometries([saddle_pcd])
```



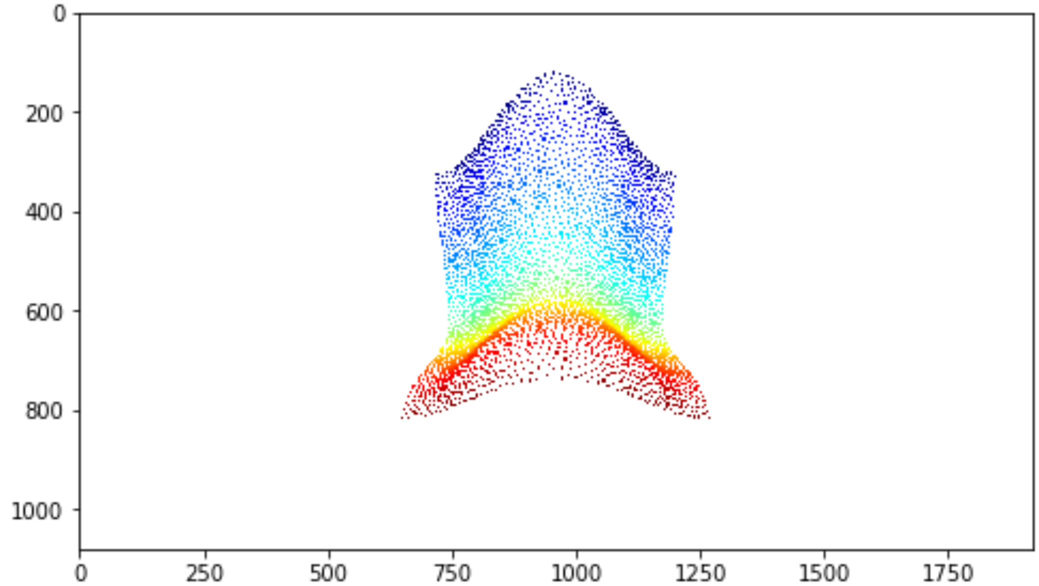
4.2

In [ ]:
```
```

In [152]:
```
def fps_downsample(points, number_of_points_to_sample):
    selected_points = np.zeros((number_of_points_to_sample, 3))
    dist = np.ones(points.shape[0]) * np.inf
    for i in range(number_of_points_to_sample):
        idx = np.argmax(dist)
        selected_points[i] = points[idx]
        dist_ = ((points - selected_points[i]) ** 2).sum(-1)
        dist = np.minimum(dist, dist_)
    return selected_points
```

In [153]:
```
res = fps_downsample(points, 4000)
```

In [154]:
```
saddle_pcd_4k = open3d.geometry.PointCloud()
saddle_pcd_4k.points = open3d.utility.Vector3dVector(res)
draw_geometries([saddle_pcd_4k])
```



4.3

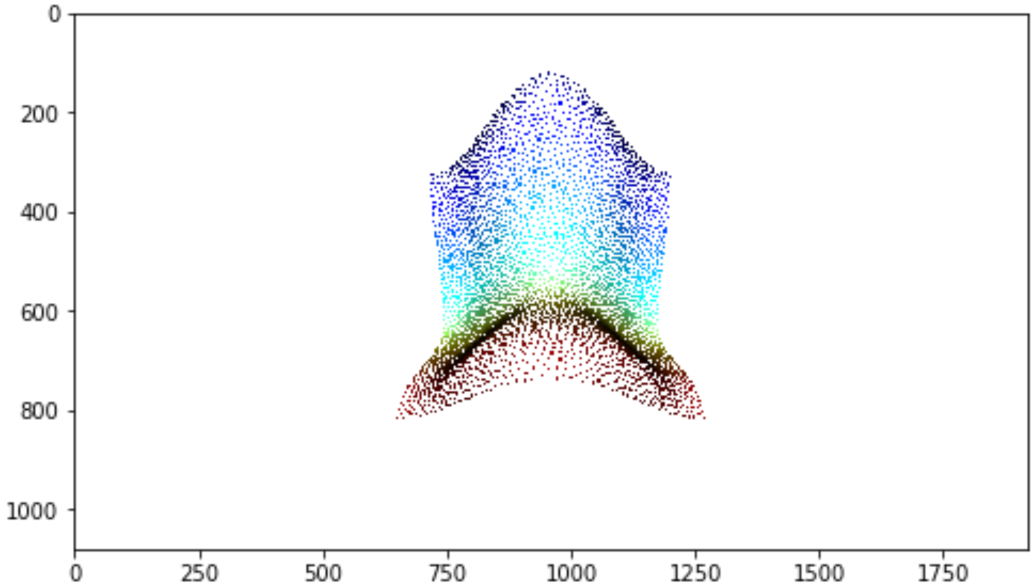
In [156]:
```
from sklearn.decomposition import PCA
normals = []
for i in res:
    ind = np.sum((points - i)**2, axis=1).argsort()[:50]
    neighbors = np.vstack([points[ind],i])
    pca = PCA(n_components=3)
    pca.fit(neighbors)
    n = pca.components_[-1]
    normals.append(n)
```

In [157]:
```
normals[:4]
```

Out[157]:
```
[array([-0.20662364, -0.95621242, -0.20727876]),
 array([ 0.39364249, -0.90080453,  0.18329426]),
 array([-0.20924619,  0.95526679,  0.20900094]),
 array([0.80726472, 0.58785469, 0.05244547])]
```

In [158]:
```
saddle_pcd_4k = open3d.geometry.PointCloud()
saddle_pcd_4k.points = open3d.utility.Vector3dVector(res)
saddle_pcd_4k.normals = open3d.utility.Vector3dVector(normals)
direction = np.array([0, 1, 0])
saddle_pcd_4k.orient_normals_to_align_with_direction(orientation_refer
draw_geometries([saddle_pcd_4k])
```

```
WARNING - 2022-01-31 13:33:39,255 - image - Clipping input data to th
e valid range for imshow with RGB data ([0..1] for floats or [0..255]
for integers).
```



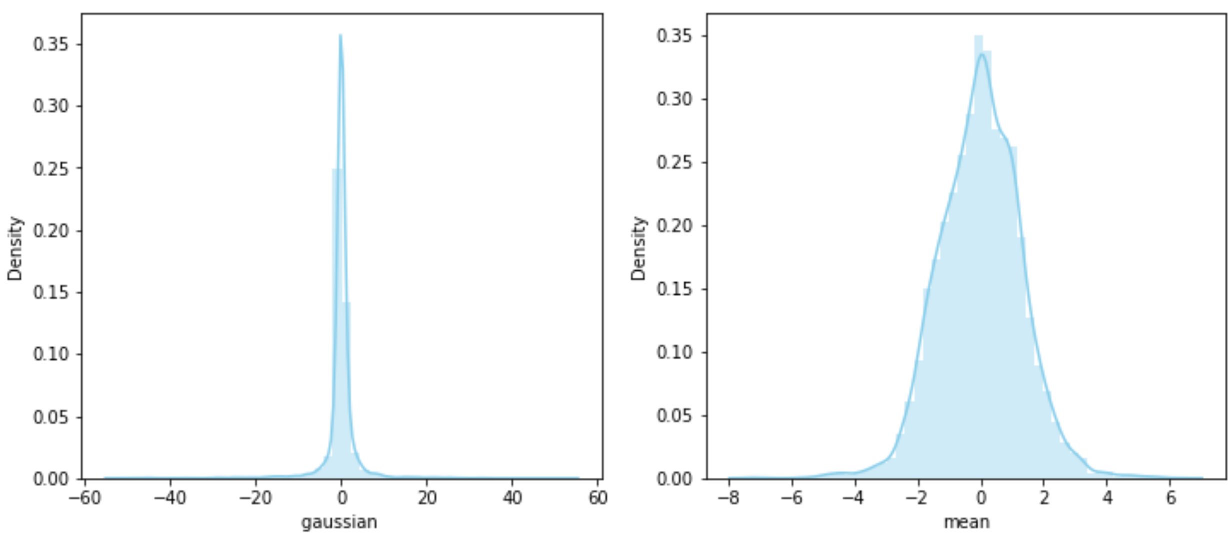
4.4

In [166]:
```
# I convert the pointcloud to trimesh
radii = [0.005, 0.01, 0.02, 0.04]

distances = saddle_pcd_4k.compute_nearest_neighbor_distance()
avg_dist = np.mean(distances)
radius = 1.5 * avg_dist

new_saddle_mesh = open3d.geometry.TriangleMesh.create_from_point_clouc
saddle_trimesh = trimesh.Trimesh(np.asarray(new_saddle_mesh.vertices),

cur = get_principal_curvature(saddle_trimesh)
```

In [167]:
```
gaussian_curv = cur[:,0] * cur[:,1]
mean_curv = 0.5 * (cur[:,0] +cur[:,1])
```

In [168]:
```
f = plt.figure(figsize=(12,5))
ax = f.add_subplot(121)
ax2 =f.add_subplot(122)
ax.set_xlabel("gaussian ")
ax2.set_xlabel("mean")
sns.distplot(gaussian_curv, kde=True, color = "skyblue", ax=ax)
sns.distplot(mean_curv,  kde=True, color = "skyblue", ax=ax2)
plt.show()
```

```
/opt/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2
619: FutureWarning: `distplot` is a deprecated function and will be r
emoved in a future version. Please adapt your code to use either `dis
plot` (a figure-level function with similar flexibility) or `histplot
` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2
619: FutureWarning: `distplot` is a deprecated function and will be r
emoved in a future version. Please adapt your code to use either `dis
plot` (a figure-level function with similar flexibility) or `histplot
` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



5 Course Feedback

1. I spend about 30 hours on this hw
2. I spend 12 hours each week for the course
3. The course is overall going really well, I really like when professor explaining academic papers to us.