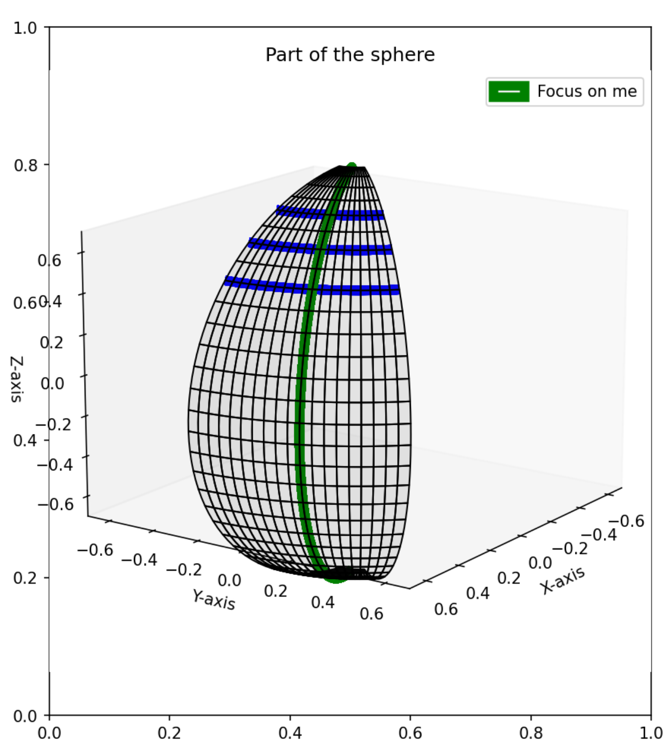
In [3]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl
mpl.rcParams['font.family'] = 'DejaVu Sans'
```

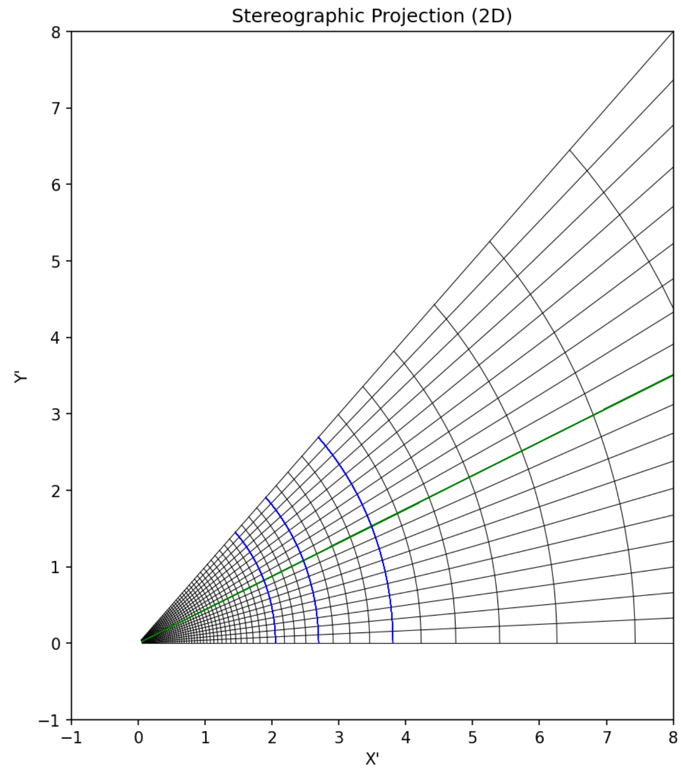
In [4]:
```python
def sphere2cart(theta, phi, r=1):
    x = r * np.sin(theta) * np.cos(phi)
    y = r * np.sin(theta) * np.sin(phi)
    z = r * np.cos(theta)
    return x, y, z
```

In [126…
```python
# Prob 2(a)
mesh = 20
theta = np.linspace(0+0.12, np.pi-0.12, round(mesh*3))  #avoid North/South pole
phi = np.linspace(0, np.pi/4, mesh)
theta, phi = np.meshgrid(theta, phi)
x, y, z = sphere2cart(theta, phi)

# Curve I care
def single_curve(angle,theta_or_phi):
    if theta_or_phi==1:
        theta_ = np.linspace(0+0.12, np.pi-0.12, round(mesh*3))  #avoid North/South pole
        phi_ = np.zeros_like(theta)+angle
        theta_, phi_ = np.meshgrid(theta_, phi_)
        x_, y_, z_ = sphere2cart(theta_, phi_)
        return x_, y_, z_
    if theta_or_phi==0:
        phi_ = np.linspace(0, np.pi/4, mesh)
        theta_ = np.zeros_like(phi_) + angle
        theta_, phi_ = np.meshgrid(theta_, phi_)
        x_, y_, z_ = sphere2cart(theta_, phi_)
        return x_, y_, z_

phi = np.linspace(0, np.pi/4, mesh)
a,b,c = single_curve(phi[round(mesh/2)],1)   #vertical
fig, axes = plt.subplots(1, 2, figsize=(15, 8), dpi=150)
ax = axes[0]
ax = fig.add_subplot(121, projection='3d')
ax.plot_surface(x, y, z, color='grey', edgecolor='k', alpha=0.1)
ax.plot_surface(a, b, c, color='grey', edgecolor='green', alpha=0.1,linewidth = 6,label=
theta = np.linspace(0+0.12, np.pi-0.12, round(mesh*3))
horizontal = [0,0,0,0,0]
for i in range(3):
    q,w,e = single_curve(theta[round(mesh/5*(i+2))],0)
    ax.plot_surface(q, w, e, color='grey', edgecolor='blue', alpha=0.1,linewidth = 6)
    horizontal[i] = [q,w,e]

A = 0.7
ax.set_xlim([-A, A])
ax.set_ylim([-A, A])
ax.set_zlim([-A, A])
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
ax.view_init(elev=15, azim=35)
ax.grid(False)
ax.set_title('Part of the sphere')


##########################################################Plot2
def St_projection(x,y,z,mesh_grid=1):
    x_flat = x.flatten()
```

```python
    y_flat = y.flatten()
    z_flat = z.flatten()
    a = x_flat/(1-z_flat)
    b = y_flat/(1-z_flat)
    if mesh_grid:
        return np.reshape(a, x.shape),np.reshape(b, y.shape)
    else:
        return a,b

def quick_plot(xx,yy,color,axis):
    ax=axis
    x_2D = xx
    y_2D = yy
    for i in range(x_2D.shape[0]):
        ax.plot(x_2D[i, :], y_2D[i, :], color=color, linewidth=0.5)
    for j in range(x_2D.shape[1]):
        ax.plot(x_2D[:, j], y_2D[:, j], color=color, linewidth=0.5)

ax = axes[1]
x_2D,y_2D = St_projection(x,y,z) # Sphere
quick_plot(x_2D,y_2D,color='black',axis=ax)
x_2D,y_2D = St_projection(a,b,c) # Vertical
quick_plot(x_2D,y_2D,color='green',axis=ax)
for i in range(3):
    x_2D,y_2D = St_projection(*horizontal[i]) # Horizontal
    quick_plot(x_2D,y_2D,color='blue',axis=ax)
ax.set_title("Stereographic Projection (2D)")
ax.set_xlabel("X'")
ax.set_ylabel("Y'")
ax.set_xlim([-1, 8])
ax.set_ylim([-1, 8])
ax.grid(False)
plt.legend()
plt.show()


# Numerically compute the angles between one vertical line and three horizontal lines

# I am too tired to write a code that can extract the coordinate, so i just plot it in m

# Function to compute the angle between three points A, B, C
def compute_angle(A, B, C):
    BA = np.array(A) - np.array(B)
    BC = np.array(C) - np.array(B)

    dot_product = np.dot(BA, BC)
    norm_BA = np.linalg.norm(BA)
    norm_BC = np.linalg.norm(BC)

    cos_theta = np.clip(dot_product / (norm_BA * norm_BC), -1.0, 1.0)
    angle = np.arccos(cos_theta)  # in radians
    return np.degrees(angle)  # Convert to degrees

# Points
points = [
    [(1.843609,0.9012854,0), (1.879297,0.8243284,0), (1.767546,0.7753174,0)],
    [(2.422891,1.184479,0), (2.46977,1.083342,0), (2.295018,1.006688,0)],
    [(3.422305,1.673063,0), (3.488521,1.530207,0), (3.169085,1.390089,0)],
    [(0.4412657,0.2157217,0.8710618), (0.4498035,0.1973021,0.8710618), (0.4884741,0.2142
    [(0.5621811,0.2465955,0.7893941), (0.5970392,0.2618857,0.7582612), (0.5857068,0.2863
    [(0.6926318,0.3038165,0.654184), (0.7212452,0.3163675,0.6162119), (0.7075552,0.34590
]

# Compute angles
angles = [compute_angle(A, B, C) for A, B, C in points]
```

```python
# Print results
print("\nAngles before projection:")
for i, angle in enumerate(angles[3:], 4):
    print(f"Angle {i}: {angle:.2f} degrees")

print('')

print("Angles after projection:")
for i, angle in enumerate(angles[:3], 1):
    print(f"Angle {i}: {angle:.2f} degrees")
```






```
Angles before projection:
Angle 4: 91.02 degrees
Angle 5: 89.08 degrees
Angle 6: 89.25 degrees

Angles after projection:
Angle 1: 88.80 degrees
Angle 2: 88.82 degrees
Angle 3: 88.82 degrees
```

In [204…

```python
# Prob 1 (b)
def generate_great_circle(theta, phi,mesh=50):
    n = np.array([np.sin(theta) * np.cos(phi),
                  np.sin(theta) * np.sin(phi),
                  np.cos(theta)])
    temp_vector = np.array([1, 0, 0]) if np.abs(n[0]) < 0.9 else np.array([0, 1, 0])
    perp1 = np.cross(n, temp_vector)
    perp1 /= np.linalg.norm(perp1)  # Normalize
    perp2 = np.cross(n, perp1)
    t = np.linspace(0, 2 * np.pi, mesh)
    x_ = np.outer(np.cos(t), perp1[0]) + np.outer(np.sin(t), perp2[0])
    y_ = np.outer(np.cos(t), perp1[1]) + np.outer(np.sin(t), perp2[1])
    z_ = np.outer(np.cos(t), perp1[2]) + np.outer(np.sin(t), perp2[2])
    return x_, y_, z_
def St_projection(x,y,z,mesh_grid=1):
    x_flat = x.flatten()
    y_flat = y.flatten()
    z_flat = z.flatten()
    a = x_flat/(1-z_flat)
    b = y_flat/(1-z_flat)
```

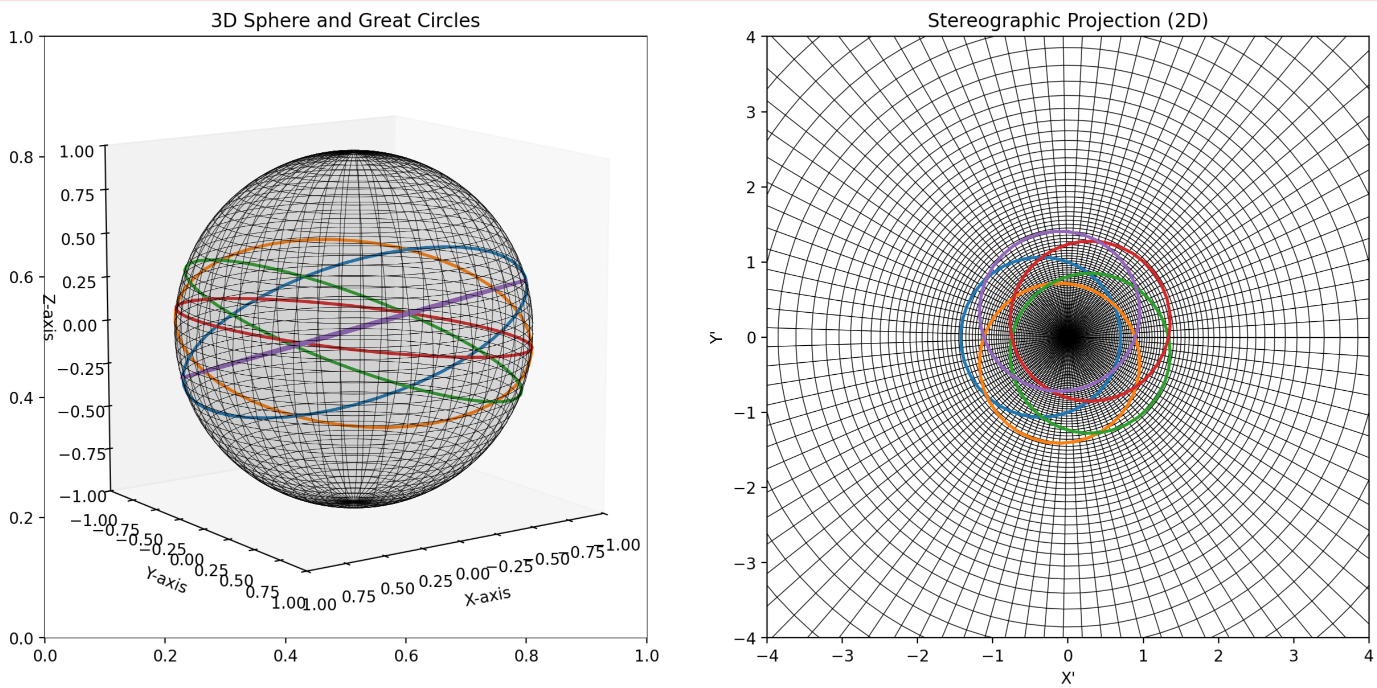
```
    if mesh_grid:
        return np.reshape(a, x.shape),np.reshape(b, y.shape)
    else:
        return a,b

def quick_plot(xx,yy,color,axis,linewidth = 0.5):
    ax=axis
    x_2D = xx
    y_2D = yy
    for i in range(x_2D.shape[0]):
        ax.plot(x_2D[i, :], y_2D[i, :], color=color, linewidth=linewidth)
    for j in range(x_2D.shape[1]):
        ax.plot(x_2D[:, j], y_2D[:, j], color=color, linewidth=linewidth)
```

In [ ]:

```
mesh = 100
# generate sphere
theta_ = np.linspace(0, np.pi, (mesh))
phi_ = np.linspace(0, np.pi*2, mesh)
theta, phi = np.meshgrid(theta_, phi_)
x, y, z = sphere2cart(theta, phi)

# generate 5 great circle
target = []
for i in range(5):
    jj = np.radians(20)
    kk = np.radians(i * 72)
    a, b, c = generate_great_circle(jj, kk)
    target.append((a, b, c))

# ST-Stranform the sphere
x_2D,y_2D = St_projection(x,y,z)

# ST-Transform the circles
output = []
for i in range(5):
    a, b = St_projection(*target[i])
    output.append((a, b))


# Set up the plot
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']
fig, axes = plt.subplots(1, 2, figsize=(15, 7), dpi=200)
# 1st plot
ax = axes[0]
ax = fig.add_subplot(121, projection='3d')
ax.plot_surface(x, y, z, color='grey', edgecolor='k', alpha=0.1, linewidth=0.2)
i=0
for a, b, c in target:
    ax.plot_wireframe(a, b, c, color=colors[i], linewidth=2)
    i+=1
A = 1
ax.set_xlim([-A, A])
ax.set_ylim([-A, A])
ax.set_zlim([-A, A])
ax.set_box_aspect([1, 1, 1])
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
ax.view_init(elev=12, azim=55)
ax.grid(False)
ax.set_title("3D Sphere and Great Circles")

# 2nd
ax = axes[1]
quick_plot(x_2D, y_2D, color='black', axis=ax)
```

```python
i=0
for a_proj, b_proj in output:
    quick_plot(a_proj, b_proj, color=colors[i], axis=ax,linewidth=2)
    i+=1
ax.set_xlabel("X'")
ax.set_ylabel("Y'")
ax.set_title("Stereographic Projection (2D)")
A=4
ax.set_xlim([-A, A])
ax.set_ylim([-A, A])
ax.grid(False)
plt.show()
```

```
C:\Users\Eric\AppData\Local\Temp\ipykernel_28788\950345070.py:19: RuntimeWarning:

invalid value encountered in divide

C:\Users\Eric\AppData\Local\Temp\ipykernel_28788\950345070.py:20: RuntimeWarning:

invalid value encountered in divide
```



In [284…

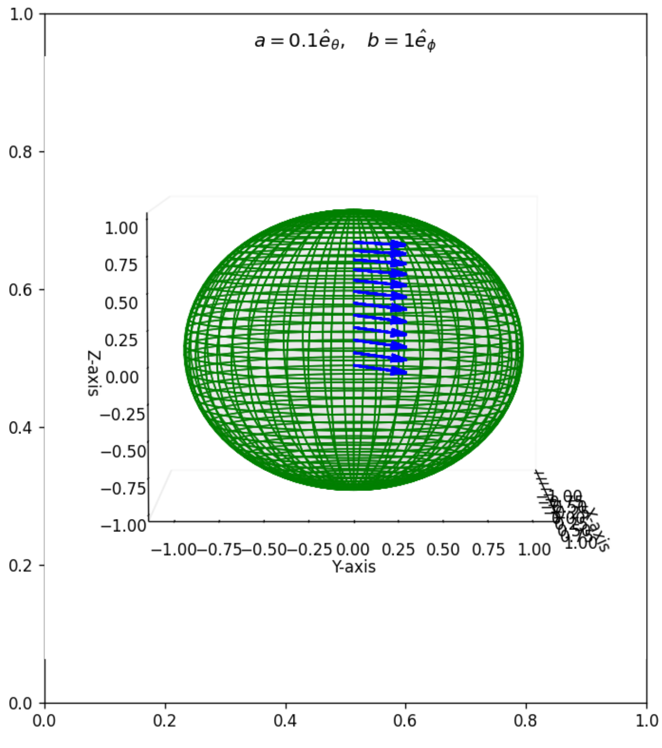
```python
# prob 2(c)

################################################################################ For gam
def sphere2cart(theta, phi, r=1):
    """Convert spherical coordinates (theta, phi) to Cartesian (x, y, z)."""
    x = r * np.sin(theta) * np.cos(phi)
    y = r * np.sin(theta) * np.sin(phi)
    z = r * np.cos(theta)
    return x, y, z

# Generate unit sphere mesh
mesh = 50
theta = np.linspace(0, np.pi, mesh)
phi = np.linspace(0, 2 * np.pi, mesh)
Theta, Phi = np.meshgrid(theta, phi)
X, Y, Z = sphere2cart(Theta, Phi)

# Generate the path
theta_position = np.linspace(np.pi/5, np.pi/2, 12)
phi_highlight = np.zeros_like(theta_position)
theta_position,phi_highlight= np.meshgrid(theta_position,phi_highlight)
```

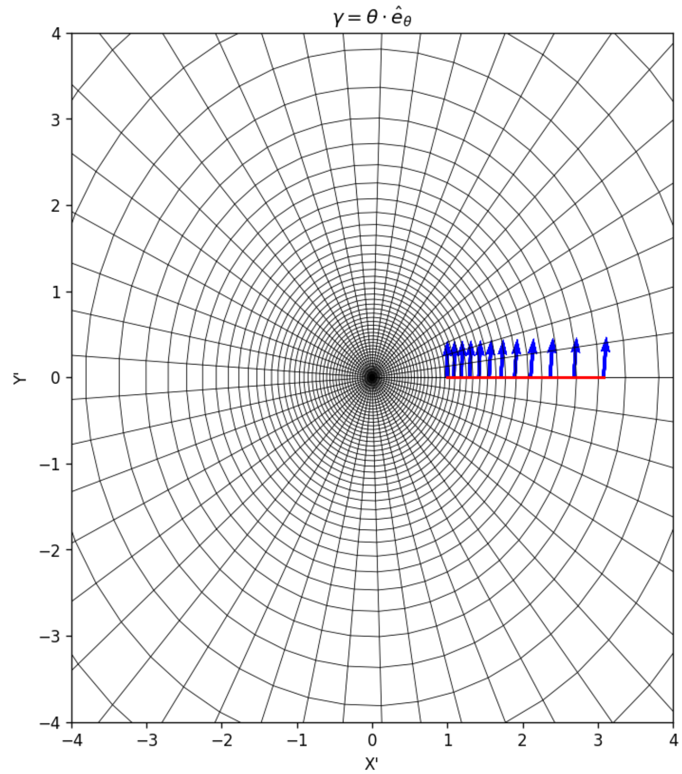
```
X_highlight, Y_highlight, Z_highlight = sphere2cart(theta_position, phi_highlight)

# Generate the vectors
a = .1
beta = 1
theta_0 = np.pi / 5
q = beta*np.sin(theta_0)
n_square = a**2 + q**2

Ux_norm = []
Uy_norm = []
Uz_norm = []
for i in range(len(theta_position)):
    theta_i = theta_position[i]
    Ux_i = a * np.cos(theta_i) * np.cos(phi_highlight[i]) + q / np.sin(theta_i) * (-np.s
    Uy_i = a * np.cos(theta_i) * np.sin(phi_highlight[i]) + q / np.sin(theta_i) * np.cos
    Uz_i = -a * np.sin(theta_i)
    A_i = np.sqrt(Ux_i**2 + Uy_i**2 + Uz_i**2)
    Ux_norm.append(Ux_i / A_i )
    Uy_norm.append(Uy_i / A_i )
    Uz_norm.append(Uz_i / A_i )


# Transform the unit sphere
x_2D,y_2D = St_projection(X, Y, Z)

# Transform the path
x_2D_path,y_2D_path = St_projection(X_highlight, Y_highlight, Z_highlight)

# Transform the vector
vx,vy = St_projection(np.array([Ux_norm]), np.array([Uy_norm]), np.array([Uz_norm]),mesh


# Generate the plot
fig, axes = plt.subplots(1, 2, figsize=(15, 8), dpi=120)

# 3D Perspective View
ax1 = axes[0]
ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.plot_surface(X, Y, Z, color='lightgray', edgecolor='g', alpha=0.1)
ax1.quiver(X_highlight, Y_highlight, Z_highlight, Ux_norm, Uy_norm, Uz_norm, color='b',
ax1.set_xlabel("X-axis")
ax1.set_ylabel("Y-axis")
ax1.set_zlabel("Z-axis")
ax1.view_init(elev=5, azim=0)
ax1.grid(False)
ax1.xaxis.pane.fill = False
ax1.yaxis.pane.fill = False
ax1.zaxis.pane.fill = False
ax1.set_title(rf"$a={{{a}}} \hat{{e}}_{{\theta}}, \quad b={{{beta}}} \hat{{e}}_{{\phi}}$

# ST-Projected View
ax2 = axes[1]
quick_plot(x_2D, y_2D, color='black', axis=ax2)
quick_plot(x_2D_path, y_2D_path, color='red', axis=ax2,linewidth=1.5)
ax2.quiver(x_2D_path, y_2D_path, vx, vy, color='b', angles='xy', scale_units='xy', scale
ax2.set_xlabel("X'")
ax2.set_ylabel("Y'")
ax2.set_title(r"$\gamma = \theta \cdot \hat{e}_{\theta}$")
A = 4
ax2.set_xlim([-A, A])
ax2.set_ylim([-A, A])
ax2.grid(False)
plt.show()

############################################################################### For gam
```

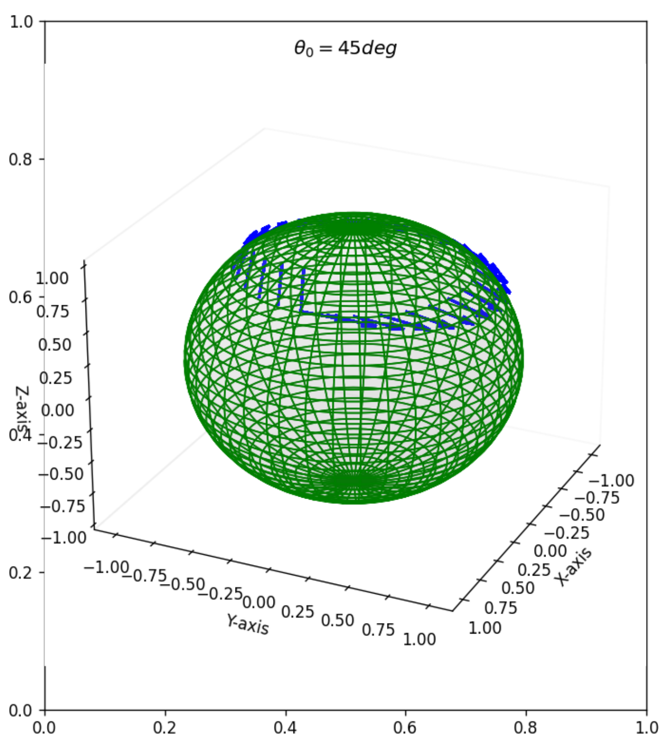
```python
mesh = 80
theta = np.linspace(0, np.pi, mesh)  # Polar angle
phi = np.linspace(0, 2 * np.pi, mesh)  # Azimuthal angle
Theta, Phi = np.meshgrid(theta, phi)  # Create 2D grid
X, Y, Z = sphere2cart(Theta, Phi)  # Convert to Cartesian coordinates
delta = 0

for theta_0 in [45,60,90]:
    john = theta_0
    theta_0 = theta_0/180*np.pi
    phi_highlight = np.linspace(0, np.pi*2, 30)
    theta_highlight = np.zeros_like(phi_highlight)+theta_0
    theta_highlight,phi_highlight= np.meshgrid(theta_highlight,phi_highlight)
    X_highlight, Y_highlight, Z_highlight = sphere2cart(theta_highlight, phi_highlight)
    v_phi_0 = 0.1
    v_theta_0 = 0.1
    v_phi = v_phi_0*np.cos(phi*np.cos(theta))
    v_theta = v_theta_0*np.sin(phi*np.cos(theta))*np.sin(theta)
    v_theta_xyz = lambda theta,phi: np.array([
        v_theta_0*np.sin(phi*np.cos(theta))*np.sin(theta)*np.cos(theta)*np.cos(phi),
        v_theta_0*np.sin(phi*np.cos(theta))*np.sin(theta)*np.cos(theta)*np.sin(phi),
        v_theta_0*np.sin(phi*np.cos(theta))*np.sin(theta)*(-np.sin(theta))])
    v_phi_xyz = lambda theta, phi: np.array([
        v_phi_0 * np.cos(phi * np.cos(theta)) * np.sin(theta)*(-np.sin(phi)),
        v_phi_0 * np.cos(phi * np.cos(theta)) * np.sin(theta)*np.cos(phi),
        np.zeros_like(phi)  ])
    vector_arrow = v_theta_xyz(theta_highlight,phi_highlight)+v_phi_xyz(theta_highlight,
    a, b, c = vector_arrow



    # Transform the path
    x_2D_path,y_2D_path = St_projection(X_highlight, Y_highlight, Z_highlight)

    # Transform the vector
    vx,vy = St_projection(a,b,c,mesh_grid=0)


    # Generate the plot
    fig, axes = plt.subplots(1, 2, figsize=(15, 8), dpi=120)

    # 3D Perspective View
    ax1 = axes[0]
    ax1 = fig.add_subplot(1, 2, 1, projection='3d')
    ax1.plot_surface(X, Y, Z, color='lightgray', edgecolor='g', alpha=0.1)
    ax1.quiver(X_highlight, Y_highlight, Z_highlight, a, b, c, color='b', length=0.3, no
    ax1.set_xlabel("X-axis")
    ax1.set_ylabel("Y-axis")
    ax1.set_zlabel("Z-axis")
    ax1.view_init(elev=25, azim=25)
    ax1.grid(False)
    ax1.xaxis.pane.fill = False
    ax1.yaxis.pane.fill = False
    ax1.zaxis.pane.fill = False
    ax1.set_title(rf"$\theta_0 = {john} deg$")

    # ST-Projected View
    ax2 = axes[1]
    quick_plot(x_2D, y_2D, color='black', axis=ax2)
    quick_plot(x_2D_path, y_2D_path, color='red', axis=ax2,linewidth=0.5)
    ax2.quiver(x_2D_path, y_2D_path, vx, vy, color='b', angles='xy', scale_units='xy', s
    ax2.set_xlabel("X'")
    ax2.set_ylabel("Y'")
    ax2.set_title(r"$\gamma = \theta \cdot \hat{e}_{\theta}$")
    A = 4-delta
```

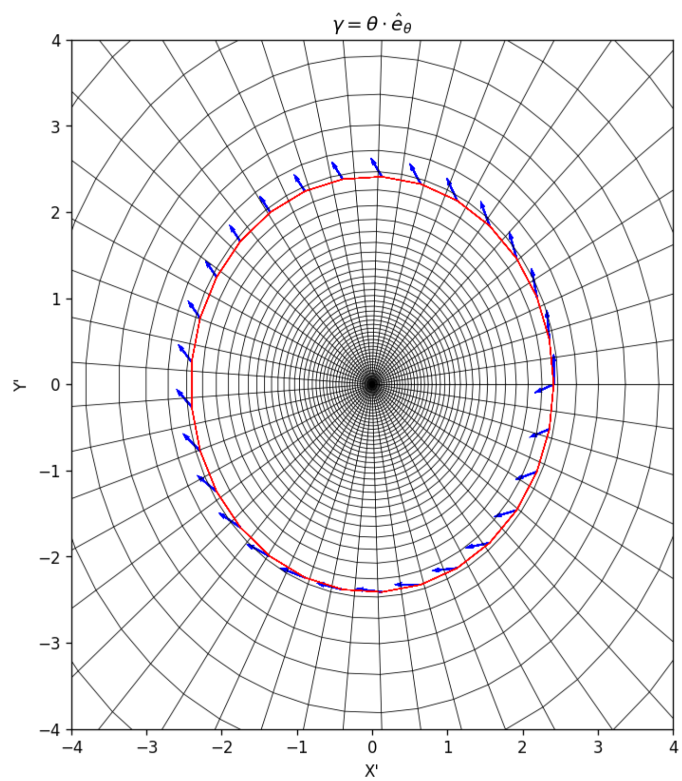
```
    ax2.set_xlim([-A, A])
    ax2.set_ylim([-A, A])
    ax2.grid(False)
    delta+=1
    plt.show()
```

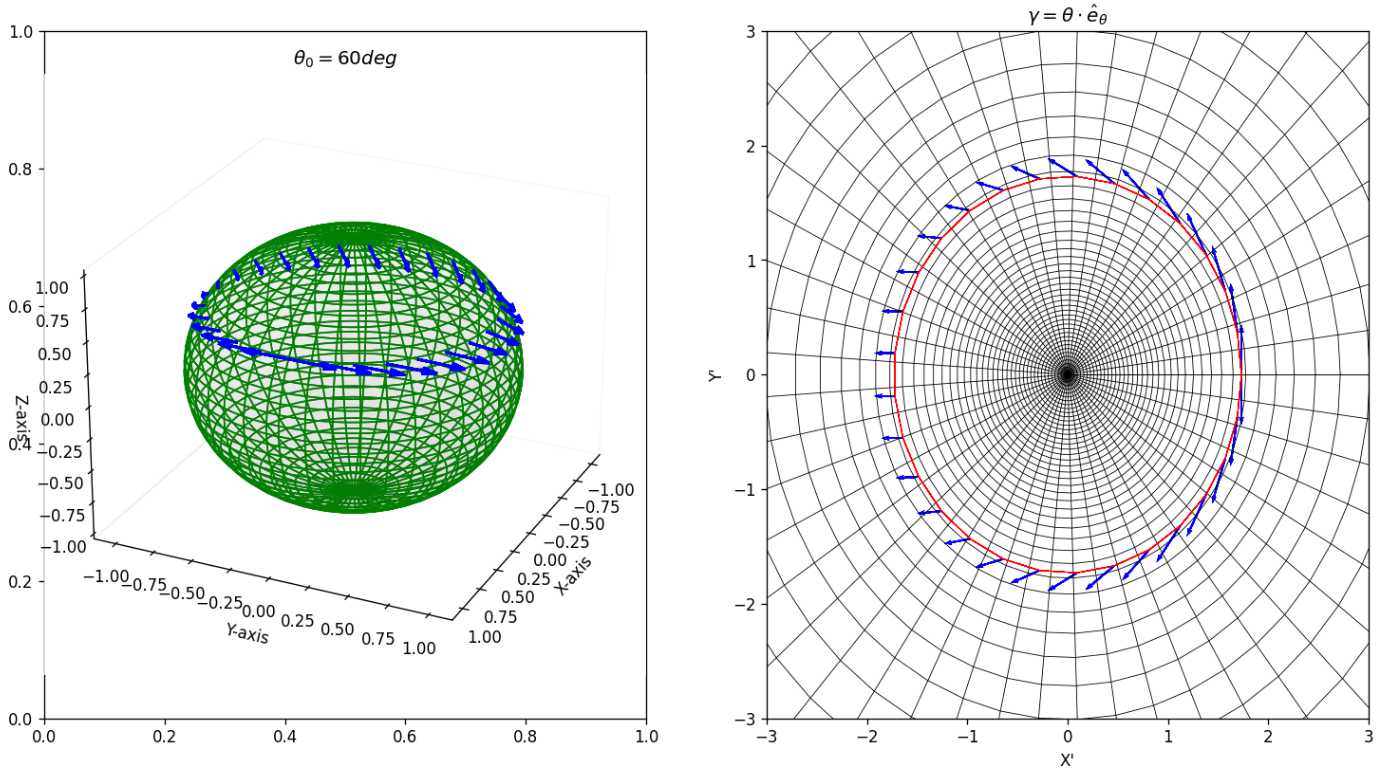
```
C:\Users\Eric\AppData\Local\Temp\ipykernel_28788\950345070.py:19: RuntimeWarning:

invalid value encountered in divide

C:\Users\Eric\AppData\Local\Temp\ipykernel_28788\950345070.py:20: RuntimeWarning:

invalid value encountered in divide
```

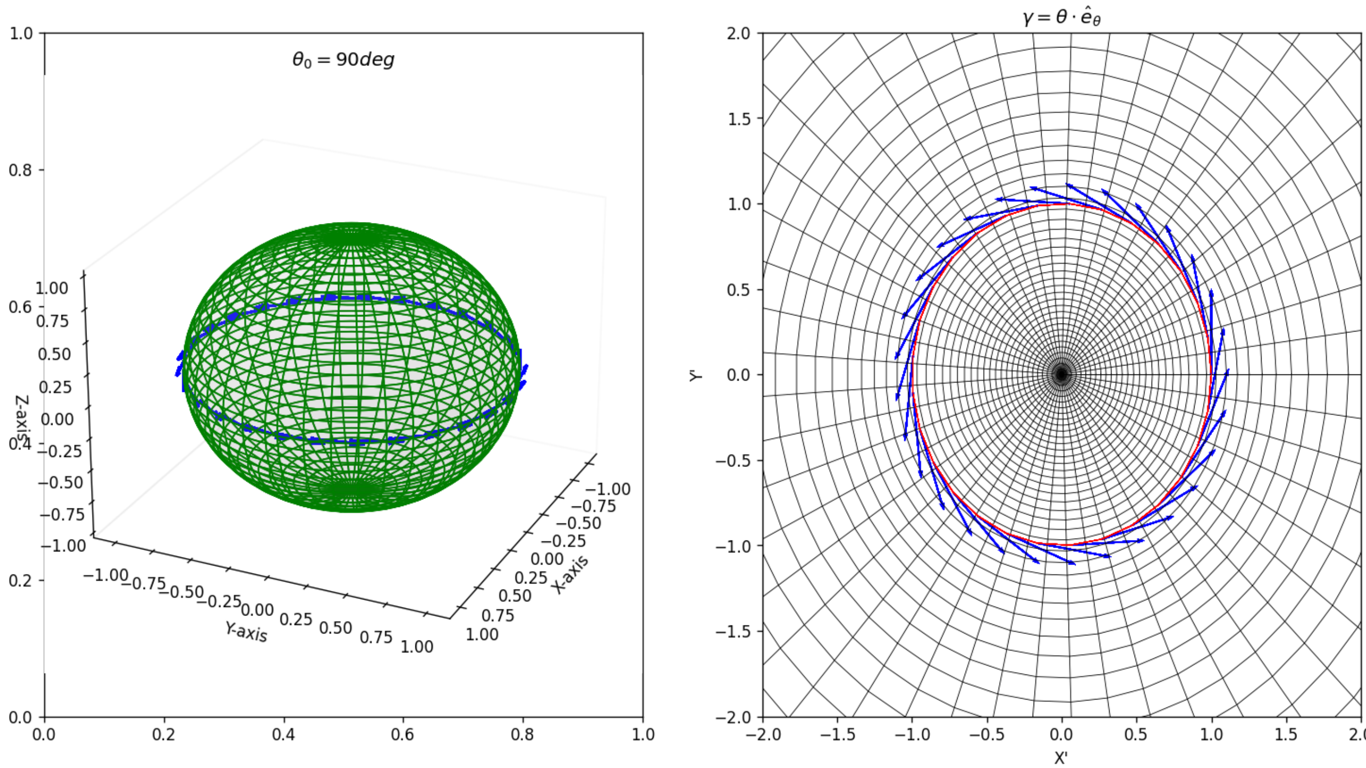












In [286…

```
# Prob 2(d)
```

Yes, as the conformal mapping is also called angle-preserved mapping. As long as the angle between vectors is the same, the inner product would not change.

Please look at the three plots above for this problem. Clearly, with your eyes, you can see the inner product is the same before and after the transformation

In [ ]:

```
# Prob 2(f)
```

No. It cannot. As the conformal mapping does not affect the angle, thus the change experienced by the vector while transport(i.e. Holonomy) is not changed.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```