1. Introducción

En esta práctiva he implementado la proyección estereográfica de la 2-esfera. Por un lado he calculado la imagen mediante la proyección de una curva sobre la superficie de la esfera y luego he desarrollado una animación interactiva de una deformación continua de la esfera. En esta práctica he implementado en python la imagen mediante la proyección estereográfica de la 2-esfera y una curva sobre ella; y en javascript la deformación continua de la 2-esfera dada por una familia paramétrica de funciones f_t, donde $t \in [0, 1)$, tal que $f_0 = \text{id}$ y $\lim_{t\to 1} f_t = \Pi$ es la proyección estereográtifica.

2. Método y resultados

2.1. Apartado I

Este apartado lo he implementado en python, utilizando la `matplotlib` y `numpy` y basándome en la plantilla. He organizado el programa en funciones para mejorar la legibilidad del mismo y poder reutilizarlo.

- La función `sphere_polar` genera las coordenadas esféricas (u, v).
- La función `polar_to_cartesian` hace el paso de coordenadas polares a cartesianas. Si se quiere transformar una parametrización de una superficie (por ejemplo las coordenadas generadas con la función `sphere_polar`) se debe utilizar el producto `np.outer` (valor por defecto). Para transformar una parametrización de una curva se debe utilizar el producto `np.multiply` (pasando el parámetro `outer = False`).
- La función `proj` calcula la imagen de una superficie o curva mediante la proyección estereográfica.

2.2. Apartado 2

Para este apartado me he permitido utilizar un lenguaje distinto, *javascript/nodejs*, y la librería de 3D *threejs*, para de este modo poder generar una animación "continua" e interactiva...

- Para generar la superficie he utilizado la función `SphereGeometry`, que devuelve un objeto de tipo `geometry` que luego será modificado para ser deformado.
- Para controlar el parámetro $t \in [0, 1)$ he usado el módulo `dat.gui` de la librería *threejs*, mediante el método `gui.add`.
- La función `tick` se ejecuta en bucle, cada frame. Aquí se calcula el estado actual del objeto `sphere`, teniendo en cuenta el parámetro `t`. Además esta función se encarga de renderizar en pantalla la superficie con un material y la malla.

En el primer anexo he incluido instrucciones para instalar y ejecutar este código.

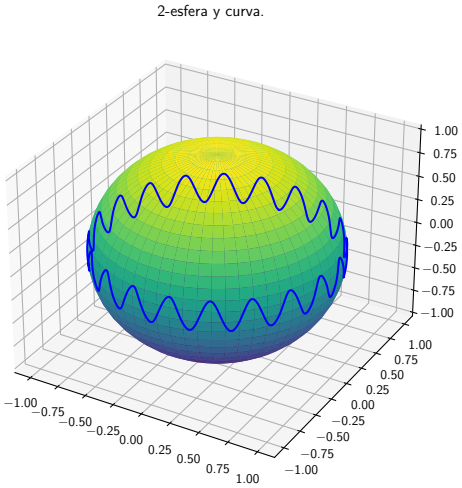
3. Resultados

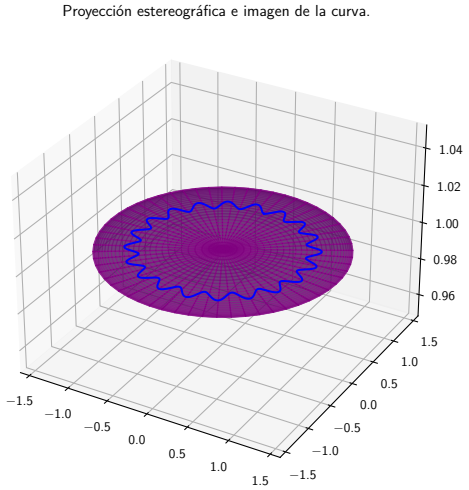
3.1. Apartado I

He elegido como curva la dada por la siguiente parametrización en coordenadas polares:

$$\begin{cases} u(t) = \frac{\pi}{2} + \frac{\pi}{20}\cos(20t) \\ v(t) = t \end{cases}$$

Luego, utilizando las funciones descritas en la sección anterior he calculado la imagen de la curva a través de la proyección estereográfica, obteniendo los siguientes gráficos:





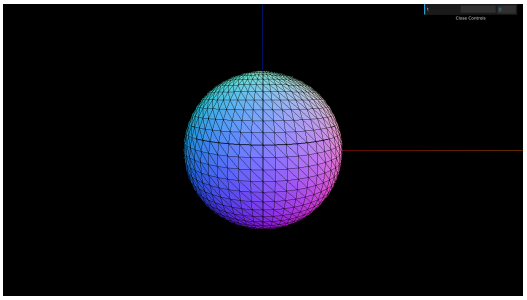
3.2. Apartado II

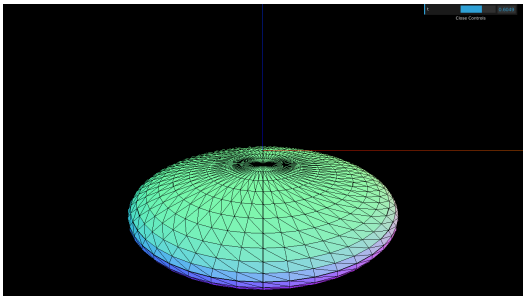
La librería *threejs* está orientada al desarrollo de aplicaciones 3D para páginas web, por lo que el resultado compilado de la aplicación es una carpeta de ficheros *.html*, *.css* y *.js* (carpeta `./dist`).

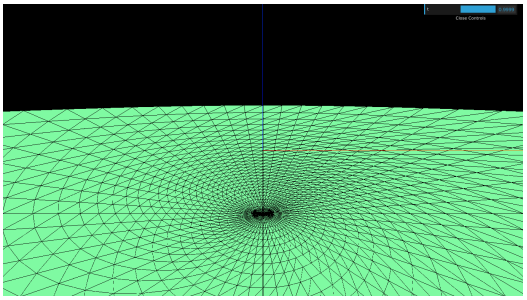
Además he subido esta aplicación a mi servidor web y se puede acceder a ella desde el siguiente enlace: haztecaso.com/stereograp

La aplicación permite mover la cámara con el ratón, cambiando la perspectiva y la distancia de esta. Además en la esquina superior derecha se encuentra un control con el que modificar el valor del parámetro t y animar la deformación.

Estas son tres capturas del resultado, con distintos valores de t:







A. Instrucciones para el código en javascript

El proyecto de *javascript* (*nodejs*) está formado por los siguientes ficheros:

- `package.json`: Descripción del proyecto y lista de dependencias. Esto hace posible instalar los paquetes necesarios usando el comando `npm`.
- `src`: Código fuente del proyecto
  - `script.js`: Código fuente del proyecto, donde realmente está ubicado el programa. El código de este fichero
  - `index.html`: Esqueleto de la web del proyecto está incluido en el siguiente anexo.
  - `style.css`: Estilos de la página web.
- `bundler`: Carpeta con ficheros de configuración para poder compilar el programa en una carpeta con el proyecto web final.
- `dist`: Carpeta con el resultado de la compilación.

Para instalar las dependencias hay que instalar los paquetes `node` y `npm`. Una vez hecho esto basta ejecutar `npm install` para instalar los módulos de `node` que necesita el proyecto. Una vez hecho esto se utilizarán los siguientes comandos:

- `nmp dev`: Para previsualizar el proyecto en un navegador web con recarga automática de los cambios.
- `npm build`: Para compilar la aplicación y generar la carpeta `dist`.

He incluido en la entrega en el campus virtual los ficheros del proyecto del siguiente modo:

- `script.js`: Archivo principal del proyecto, donde está la parte que concierne a la implementación del segundo apartado.
- `project.zip`: Proyecto completo comprimido de nodejs, sin la carpeta `dist`.
- `dist.zip`: Proyecto compilado. Se puede visualizar el proyecto abriendo el fichero `index.html` con cualquier navegador que soporte *javascript*.

B. Código

El siguiente código con la implementación también está adjunto en la entrega y disponible, junto con esta memoria, en un repositorio git en el siguiente enlace: github.com/haztecaso/gcomp22.

B.1. Apartado 1 (python)

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from numpy import pi, cos, sin

def sphere_polar(res):
    u = np.linspace(0, pi, res)
    v = np.linspace(0, 2*pi, 2*res)
    return u, v

def polar_to_cartesian(u, v, outer = True):
    prod = np.outer if outer else np.multiply
    x = prod(sin(u), sin(v))
    y = prod(sin(u), cos(v))
    z = prod(cos(u), np.ones_like(v))
    return x, y, z

def proj(x, z, z0 = 1, α = 1):
    z0 = z*0+z0
    ε = 1e-16
    x_trans = x/(abs(z0-z)**α+ε)
    return (x_trans)


def main():
    u, v = sphere_polar(30)
    X, Y, Z = polar_to_cartesian(u, v)

    gs  = gridspec.GridSpec(1, 2)
    fig = plt.figure()

    # Esfera
    ax = fig.add_subplot(gs[0,0], projection='3d')
    ax.set_title('2-esfera y curva.');
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor='none')

    # Curva (parametrización en coordenadas polares)
    T = np.linspace(0, 2*pi, 1000)
    u = pi/2+pi/20*cos(T*20)
    v = T

    # Paso a coordenadas cartesianas
    x, y, z = polar_to_cartesian(u, v, outer=False)
    ax.plot(x, y, z, '-b', zorder=3)

    ax = fig.add_subplot(gs[0,1], projection='3d')
    ax.set_title('Proyección estereográfica e imagen de la curva.');

    # Proyección estereográfica de la esfera
    z0, α = 1, 0.5

    Xp = proj(X, Z, z0, α)
    Yp = proj(Y, Z, z0, α)
    ax.plot_surface(Xp, Yp, Z*0+z0, rstride=1, cstride=1, cmap='viridis', alpha=0.5, edgecolor='purple')
```

```
    # Imagen de la curva dibujada anteriormente
    xp = proj(x, z, z0, α)
    yp = proj(y, z, z0, α)
    ax.plot(xp, yp, z0, '-b', zorder=3)

    plt.show()


if __name__ == "__main__":
    main()
```

B.2. Apartado 2 (javascript)

He incluido aqui el código del fichero `src/script.js`, donde está la parte central del programa. Para ejecutar o compilar la aplicación de *threejs* es necesario tener *nodejs* instalado y ejecutar el comando

- `npm dev` para ejecutar la aplicación.
- `npm build` para compilar la aplicación, que se guardará en la carpeta `dist` y podrá abrirse un navegador web.

```
import "./style.css";
import * as THREE from "three";
import { OrbitControls } from "three/examples/jsm/controls/OrbitControls.js";
import * as dat from "dat.gui";

const deepcopy = (obj) => JSON.parse(JSON.stringify(obj));

// Canvas
const canvas = document.querySelector("canvas.webgl");

// Scene
const scene = new THREE.Scene();

const axesHelper = new THREE.AxesHelper(5);
scene.add(axesHelper);

// Objects
const geometry = new THREE.SphereGeometry(1, 60, 30);
geometry.rotateX(Math.PI / 2);

// Mesh
const sphere = new THREE.Mesh(
  geometry,
  new THREE.MeshNormalMaterial({ flatShading: true })
);

const wireframe = new THREE.Mesh(
  geometry,
  new THREE.MeshBasicMaterial({
    color: 0x000000,
    flatShading: true,
    wireframe: true,
    wireframeLinewidth: 2,
  })
);

const group = new THREE.Group();

group.add(sphere);
group.add(wireframe);

scene.add(group);

/**
 * Sizes
 */
const sizes = {
  width: window.innerWidth,
  height: window.innerHeight,
};

// Camera
const camera = new THREE.PerspectiveCamera(
  75,
  sizes.width / sizes.height,
  0.1,
  100
);

camera.position.y = -2.5;
camera.position.z = 0.7;
camera.lookAt(geometry.center());

scene.add(camera);

```

```
// Controls
const controls = new OrbitControls(camera, canvas);
controls.enableDamping = true;

// Renderer
const renderer = new THREE.WebGLRenderer({ canvas: canvas });

renderer.setSize(sizes.width, sizes.height);
renderer.setPixelRatio(Math.min(window.devicePixelRatio, 1));

/* Animate */

const pos = deepcopy(geometry.attributes.position.array);

// GUI

const gui = new dat.GUI();
const params = { t: 0, rotation: { x: 0, y: 0, z: 0 } };
gui.add(params, "t", 0, 1 - 0.0001, 0.0001).listen();

/*
const rotFolder = gui.addFolder("Rotation");
rotFolder.add(params.rotation, "x", 0, 2 * Math.PI, 0.001).listen();
rotFolder.add(params.rotation, "y", 0, 2 * Math.PI, 0.001).listen();
rotFolder.add(params.rotation, "z", 0, 2 * Math.PI, 0.001).listen();

const posFolder = gui.addFolder("Position");
posFolder.add(camera.position, "x", -5, 5, 0.001).listen();
posFolder.add(camera.position, "y", -5, 5, 0.001).listen();
posFolder.add(camera.position, "z", -5, 5, 0.001).listen();
posFolder.open();
*/

const tick = () => {
  geometry.rotateY(Math.PI / 2);

  controls.update();

  sphere.rotation.x = params.rotation.x;
  sphere.rotation.y = params.rotation.y;
  sphere.rotation.z = params.rotation.z;

  wireframe.rotation.x = params.rotation.x;
  wireframe.rotation.y = params.rotation.y;
  wireframe.rotation.z = params.rotation.z;

  for (let i = 0; i < geometry.attributes.position.count; i++) {
    const t = params.t;

    const x = pos[i * 3];
    const y = pos[i * 3 + 1];
    const z = pos[i * 3 + 2];

    const m = 2 / (2 * (1 - t) + (1 - z) * t);

    // set new position
    geometry.attributes.position.setX(i, m * x);
    geometry.attributes.position.setY(i, m * y);
    geometry.attributes.position.setZ(i, -t + z * (1 - t));
  }

  geometry.computeVertexNormals();
  geometry.attributes.position.needsUpdate = true;

  renderer.render(scene, camera);
  window.requestAnimationFrame(tick);
};

window.addEventListener("resize", () => {
  // Update sizes
  sizes.width = window.innerWidth;
  sizes.height = window.innerHeight;

  // Update camera
  camera.aspect = sizes.width / sizes.height;
  camera.updateProjectionMatrix();

  // Update renderer
  renderer.setSize(sizes.width, sizes.height);
  renderer.setPixelRatio(Math.min(window.devicePixelRatio, 1));
});

tick();
```