MACHINE LEARNING
TRABAJO 01

Presentado por:
Erick Wilson Aronés Garcilazo U201924440
Ronaldo David Cornejo Valencia U201816502

Profesor: Luis Martin Canaval Sanchez
Sección: CC72

Ciclo: 2022-2

1. Descripción de los formatos utilizados para los modelos del artículo

El formato OFF (Object File Format) es un formato de archivo basado en texto ASCII para describir objetos geométricos 2D y 3D. Este formato describe las superficies geométricas de los objetos 3D mediante vértices y polígonos, mientras que los colores se definen mediante valores RGB.

Composición de un archivo OFF estándar:

- Primera línea (opcional): Las letras OFF para marcar el tipo de archivo.
- Segunda línea: El número de vértices, el número de caras y el número de aristas, en orden (este último puede ignorarse escribiendo 0 en su lugar).
- Lista de vértices: Coordenadas X, Y y Z.
- Lista de caras: Número de vértices, seguido de los índices de los vértices que la componen, en orden (indexados desde cero).

Opcionalmente, los valores RGB para el color de la cara pueden seguir los elementos de las caras.

Ejemplo de un archivo off:

```
OFF
8 6 0
-0.500000 -0.500000 0.500000
0.500000 -0.500000 0.500000
-0.500000 0.500000 0.500000
0.500000 0.500000 0.500000
-0.500000 0.500000 -0.500000
0.500000 0.500000 -0.500000
-0.500000 -0.500000 -0.500000
0.500000 -0.500000 -0.500000
4 0 1 3 2
4 2 3 5 4
4 4 5 7 6
4 6 7 1 0
4 1 7 5 3
4 6 0 2 4
```

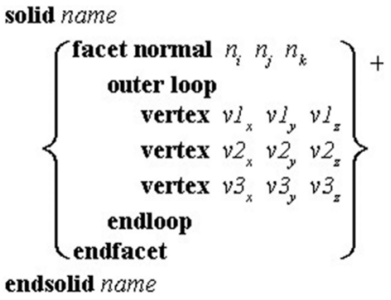
2. Descripción del formato STL

Formato STL (stereolithography CAD)

El STL es un formato de archivo que se usa generalmente para la impresión en 3D y el diseño asistido por computadora (CAD). La sigla de este formato proviene de “Stereolithography” que es una conocida tecnológica de impresión 3D. Los archivos STL describen solo la geometría de la superficie de un objeto de tres dimensiones sin ninguna representación de color, textura u otros atributos. Un archivo STL describe una superficie triangulada no estructurada sin procesar el vector normal unitario y los vértices (estos tienen que estar ordenados por la regla de la mano derecha) de los triángulos usando un sistema de coordenadas cartesianas de tres dimensiones.

Asimismo, el formato STL tiene dos representaciones: binaria y ASCII.

ASCII STL



Este comienza y termina con el nombre, pero es una cadena opcional. El “n” representa el vector normal unitario que apunta fuera del triángulo formado por los vértices. Luego, se muestran los tres vértices que representan cada triángulo.

```
UINT8[80]    – Header                -     80 bytes
UINT32       – Number of triangles   -      4 bytes
```

```
foreach triangle                     - 50 bytes:
    REAL32[3] – Normal vector            - 12 bytes
    REAL32[3] – Vertex 1                 - 12 bytes
    REAL32[3] – Vertex 2                 - 12 bytes
    REAL32[3] – Vertex 3                 - 12 bytes
    UINT16    – Attribute byte count     -  2 bytes
end
```

El archivo STL binario tiene un encabezado de 80 caracteres, después un entero que especifica el número de triángulos. Después, en un ciclo, se encuentran los datos que describen a cada triángulo como su vector normal unitario y sus vértices.

Finalmente, algunos de los beneficios de los archivos STL son que la gran mayoría de las impresoras 3D admite este formato. Asimismo, puede producir un modelo preciso de casi cualquier forma. Además, debido a su falta de color y textura, los archivos STL poseen menor tamaño y se procesan más rápido que otros formatos; por ello, es el mejor candidato para imprimir un objeto de un solo color y material.

3. Identificar y describir en el informe las conversiones de tipos que considera necesarias para el proyecto del trabajo parcial y final

De off a stl: Esta transformación es necesaria debido a que el formato STL es compatible con la mayoría de softwares e impresoras 3D. Además, es un formato indicado para el proyecto, puesto que no representa los colores y las texturas. Por ello, es el máximo candidato para imprimir objetos de un solo color y material. Por otra parte, por motivos académicos de la universidad, la impresora del centro académico desarrolla mejores resultados de impresión con los formatos STL.

De off a obj: Esta transformación es necesaria debido a que el formato obj es compatible tanto con los objetos poligonales y objetos de forma libre. Es uno de los formatos más comunes para las impresiones 3D y estamos familiarizados con estos archivos. Además, los archivos en formato OBJ se pueden abrir con Autodesk Maya 2013, Blender, y MeshLab en Microsoft Windows, Mac OS y plataformas Linux.

4. Implementar funciones para transformar entre los formatos que usted considere pertinente para el proyecto

*CONVERT_OFF_TO_STL

vnu()

Esta función permite retornar el vector normal unitario teniendo como dato 3 vértices (x, y, z). Esto es fundamental para el formato STL.

```
def vnu(v1,v2,v3):
    v1 = np.array(v1)
    v2 = np.array(v2)
    v3 = np.array(v3)
    pq = v2 - v1
    pr = v3 - v1
    i = pq[1]*pr[2]-pq[2]*pr[1]
    j = pq[0]*pr[2]-pq[2]*pr[0]
    k = pq[0]*pr[1]-pq[1]*pr[0]

    vr = np.array([i,-1*j,k])
    u = vr/np.linalg.norm(vr)
    return u
```

leerOff()

Esta función nos permite leer los triángulos y los vértices que conforman el archivo off.

```
def leerOff(url):
    file = open(url)
    cont = 0
    contv = 0
    contT = 0
    vectores = []
    triangulos = []
    for line in file:
        if len(line.split()) == 0:
            break
        if cont == 0:
            cont += 1
            continue
        if cont == 1:
            n,m,l = line.split()
            n = int(n)
            m = int(m)
            cont = 1000
            continue
        if contv < n and cont == 1000:
            a,b,c = line.split()
            a,b,c = float(a),float(b),float(c)
            vectores.append([a,b,c])
            contv +=1
        else:
            cont = 2000
        if contT < m and cont == 2000:
            _,a,b,c = line.split()
            a,b,c = int(a),int(b),int(c)
            triangulos.append([a,b,c])
            contT +=1
    file.close()
    return vectores, triangulos
```

convert_off_to_stl()

Esta función nos permite crear un archivo STL, pues utiliza las función anterior para extraer los triángulos y los vértices que lo conforman. Asimismo, calcula el vector normal unitario para cada triángulo y lo imprime para que se forme el formato.

```
def convert_off_to_stl(url):
    name = url[:-4]
    v, t = leerOff(url)
    f = open(name+".stl","w")
    f.write(f"solid {name} \n")
    while len(t) > 0:
        t1 = t.pop(0)
        v1 = [v[i] for i in t1]
        vn = vnu(v1[0],v1[1],v1[2])
        f.write(f"facet normal {vn[0]} {vn[1]} {vn[2]} \n")
        f.write(" outer loop \n")
        f.write(f"  vertex {v1[0][0]} {v1[0][1]} {v1[0][2]} \n")
        f.write(f"  vertex {v1[1][0]} {v1[1][1]} {v1[1][2]} \n")
        f.write(f"  vertex {v1[2][0]} {v1[2][1]} {v1[2][2]} \n")
        f.write(" endloop \n")
        f.write("endfacet \n")
    f.write(f" endsolid {name} \n")
    f.write("\n")
    f.close()
```

*CONVERT_OFF_TO_OBJ

read_file()

Esta función permite extraer la información del archivo off, transformar los datos a enteros y retorna dos listas con los vértices y caras del archivo.

```
def read_file(name_file):
    with open(name_file, "r") as file:
        my_list = [line.rstrip() for line in file]

    my_list.remove('OFF')
    new_list = []

    for e in my_list:
        temp_list = []
        for sub_e in e.split(" "):
            temp_list.append(float(sub_e))
        new_list.append(temp_list)

    div = int(new_list[0][0])
    new_list.pop(0)

    vertex_list = np.array(new_list[:div])
    faces_list = np.array(new_list[div:])

    return vertex_list, faces_list
```

get_normal_list()

Esta función permite utilizar la lista de vértices y caras para poder calcular el vector normal utilizando la biblioteca “numpy”, el vector normal es una parte importante que conforma un archivo obj. La función retorna una lista de vectores normales y una lista con los vértices que conforman cada cara junto con el índice del vector normal que le corresponde, aumentando el valor en 1 debido a los estándares de un archivo obj.

```
def get_normal_list(vertex_list, faces_list):
    surface_normal = []
    faces_transformed = []
    for face in faces_list:
        s_a = int(face[1])
        s_b = int(face[2])
        s_c = int(face[3])

        A = vertex_list[s_a]
        B = vertex_list[s_b]
        C = vertex_list[s_c]

        AB = B-A
        AC = C-A

        surface_normal.append(np.cross(AB, AC))
        faces_transformed.append([[s_a+1, s_b+1, s_c+1], len(surface_normal)])

    return surface_normal, faces_transformed
```

create_obj_file()

Esta función permite generar un archivo obj, utiliza las dos funciones posteriores para obtener las listas de vértices, caras modificadas y vectores normales para utilizar esa información y escribirlas dentro del archivo obj generado. Primero se abre el archivo obj, luego se agrega el nombre dentro del archivo, seguido de los vértices junto con la letra ‘v’, luego los vectores normales anteponiendo las letras ‘vn’, finalmente se agregan las caras con las letra ‘f’ junto al índice de su vector normal y cerramos el archivo.

```
def create_obj_file(name_file):
    vertex_list, faces_list = read_file(name_file)
    surface_normal, faces_transformed = get_normal_list(vertex_list, faces_list)

    file_obj = open('./'+name_file[:-4]+'.obj', 'x', encoding='utf8')
    file_obj.write('g '+name_file[:-4]+'\n')

    for v in vertex_list:
        file_obj.write('v ' + str(v)[1:-1]+'\n')

    file_obj.write('\n')
    for normal in surface_normal:
        file_obj.write('vn ' + str(normal)[1:-1]+'\n')

    file_obj.write('\n')
    for face in faces_transformed:
        v1 = str(face[0][0])
        v2 = str(face[0][1])
        v3 = str(face[0][2])
        vn = str(face[1])
        word = "f "+v1 + "//"+vn+" "+v2+"//"+vn+" "+v3+"//"+vn+"\n"
        file_obj.write(word)

    file_obj.close()
```