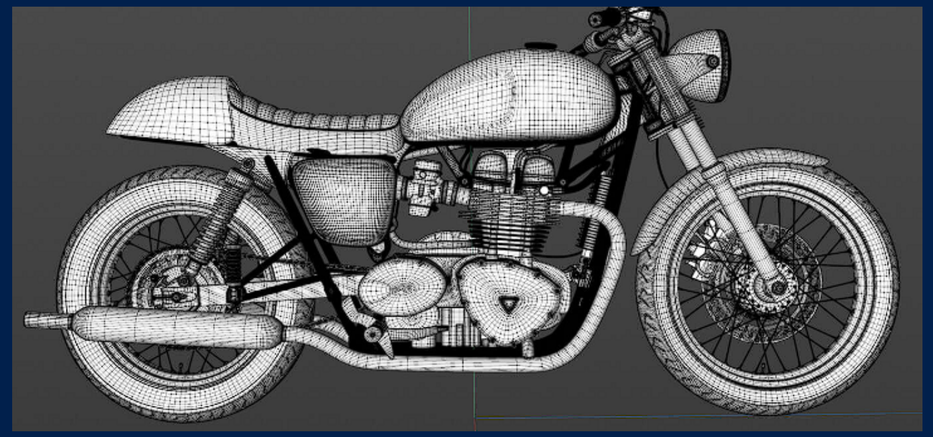
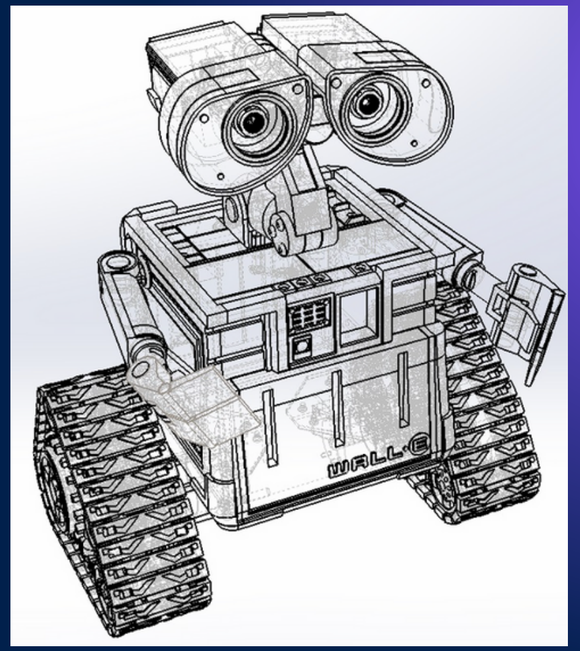
Carga de Archivos

CC3505-1

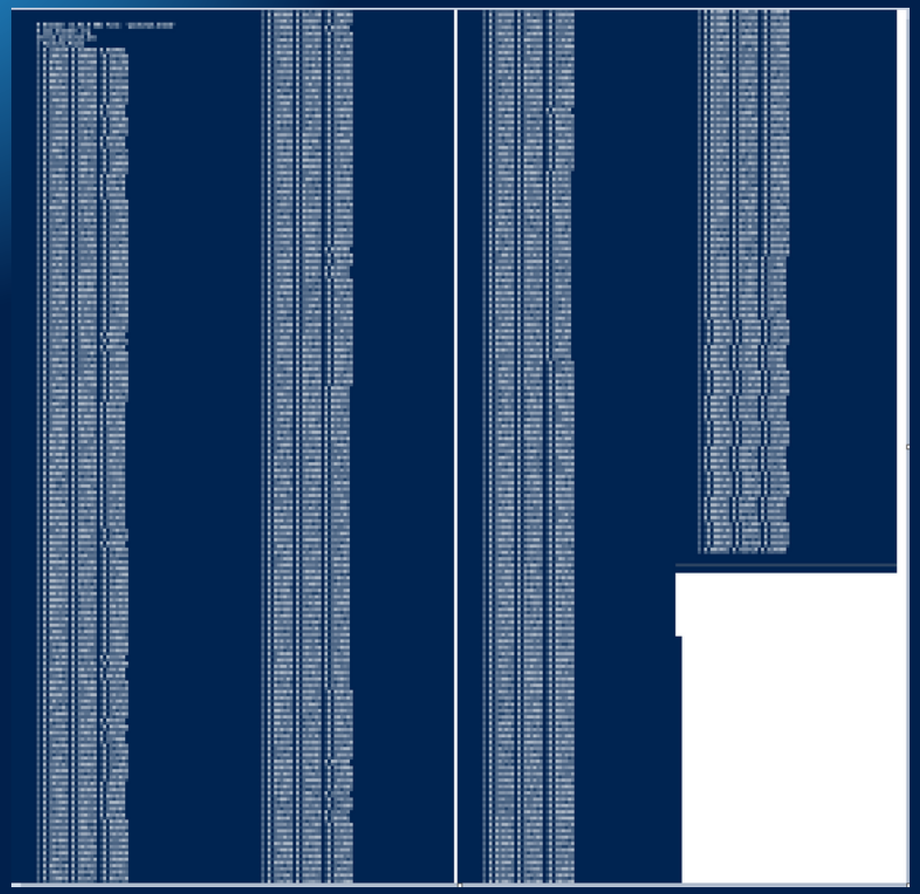
Modelos





Programas

Misión de hoy



->

Archivos .OFF

OFF (file format)

3 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

OFF (**O**bject **F**ile **F**ormat) is a geometry definition file format containing the description of the composing polygons of a geometric object.[1] It can store 2D or 3D objects, and simple extensions allow it to represent higher-dimensional objects as well.[2] Though originally developed for Geomview, a geometry visualization software, other software has adapted the simple standard.[3]

OFF geometry format

Filename extension	`.off`
Internet media type	`text/plain`
Type of format	3D model format

```
OFF
# cube.off
# A cube
8 6 12
```

```
 1.0  0.0 1.4142
 0.0  1.0 1.4142
-1.0  0.0 1.4142
 0.0 -1.0 1.4142
 1.0  0.0 0.0
 0.0  1.0 0.0
-1.0  0.0 0.0
 0.0 -1.0 0.0
```

```
4  0 1 2 3  255 0 0 #red
4  7 4 0 3  0 255 0 #green
4  4 5 1 0  0 0 255 #blue
4  5 6 2 1  0 255 0
4  3 2 6 7  0 0 255
4  6 5 4 7  255 0 0
```

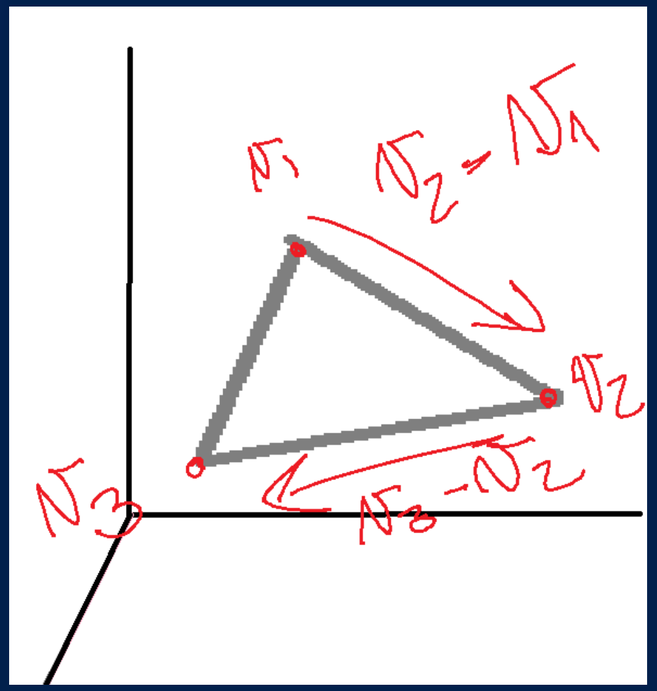
```
def readOFF(filename, color):
    vertices = []
    normals= []
    faces = []

    with open(filename, 'r') as file:
        line = file.readline().strip()
        assert line=="OFF"

        line = file.readline().strip()
        aux = line.split(' ')

        numVertices = int(aux[0])
        numFaces = int(aux[1])
```

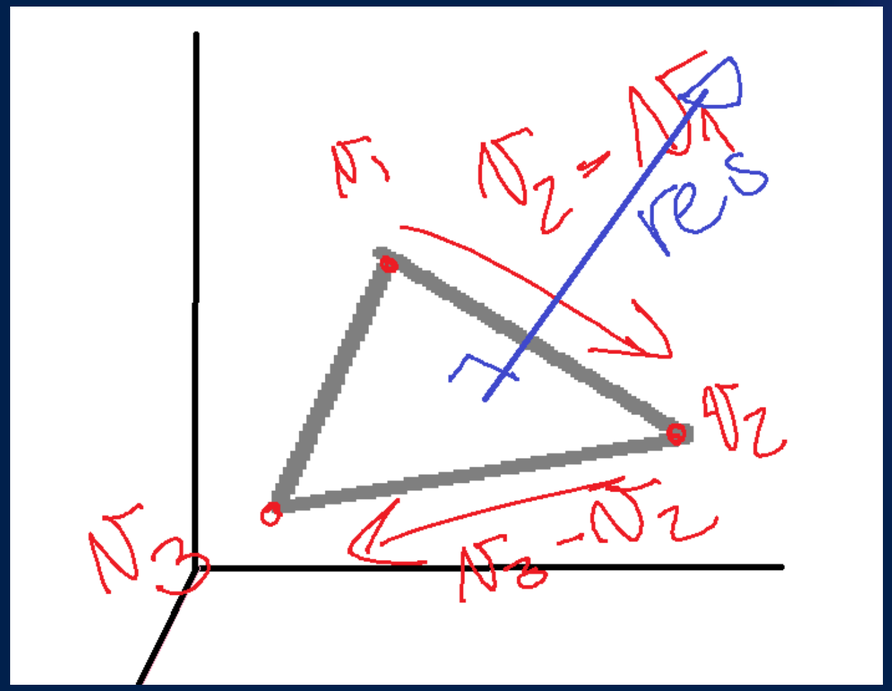
```python
for i in range(numVertices):
    aux = file.readline().strip().split(' ')
    vertices += [float(coord) for coord in aux[0:]]

vertices = np.asarray(vertices)
vertices = np.reshape(vertices, (numVertices, 3))
#print(f'Vertices shape: {vertices.shape}')

normals = np.zeros((numVertices,3), dtype=np.float32)
#print(f'Normals shape: {normals.shape}')
```

$$\text{vertices} = \begin{pmatrix} x1 & x2 & x3 & x4 \\ y1 & y2 & y3 & y4 \\ z1 & z2 & z3 & z4 \end{pmatrix}$$

```
for i in range(numFaces):
    aux = file.readline().strip().split(' ')
    aux = [int(index) for index in aux[0:]]
    faces += [aux[1:]]
```

```
vecA = [vertices[aux[2]][0] - vertices[aux[1]][0],
        vertices[aux[2]][1] - vertices[aux[1]][1],
        vertices[aux[2]][2] - vertices[aux[1]][2]]
vecB = [vertices[aux[3]][0] - vertices[aux[2]][0],
        vertices[aux[3]][1] - vertices[aux[2]][1],
        vertices[aux[3]][2] - vertices[aux[2]][2]]

res = np.cross(vecA, vecB)
```





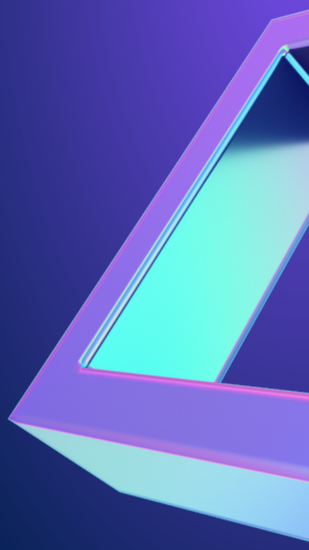
```
normals[aux[1]][0] += res[0]
normals[aux[1]][1] += res[1]
normals[aux[1]][2] += res[2]

normals[aux[2]][0] += res[0]
normals[aux[2]][1] += res[1]
normals[aux[2]][2] += res[2]

normals[aux[3]][0] += res[0]
normals[aux[3]][1] += res[1]
normals[aux[3]][2] += res[2]
```

```
norms = np.linalg.norm(normals,axis=1)
normals = normals/norms[:,None]
```

```
color = np.asarray(color)
color = np.tile(color, (numVertices, 1))

vertexData = np.concatenate((vertices, color), axis=1)
vertexData = np.concatenate((vertexData, normals), axis=1)
```

$$\text{vertexData} = \begin{pmatrix} x1 & x2 & x3 & x4 \\ y1 & y2 & y3 & y4 \\ z1 & z2 & z3 & z4 \\ tx1 & tx2 & tx3 & tx4 \\ ty1 & ty2 & ty3 & ty4 \\ nx1 & nx2 & nx3 & nx4 \\ ny1 & ny2 & ny3 & ny4 \\ nz1 & nz2 & nz3 & nz4 \end{pmatrix}$$

```
indices = []
vertexDataF = []
index = 0

for face in faces:
    vertex = vertexData[face[0],:]
    vertexDataF += vertex.tolist()
    vertex = vertexData[face[1],:]
    vertexDataF += vertex.tolist()
    vertex = vertexData[face[2],:]
    vertexDataF += vertex.tolist()

    indices += [index, index + 1, index + 2]
    index += 3


return Shape(vertexDataF, indices)
```



Archivos .OBJ

Wavefront .obj file

10 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

For other uses, see Obj (disambiguation).

OBJ (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package. The file format is open and has been adopted by other 3D graphics application vendors.

OBJ geometry format

Filename extension	`.obj`
Internet media type	`model/obj` [1]
Developed by	Wavefront Technologies
Type of format	3D model format

The OBJ file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary. OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line.

Archivos .OBJ

```
v 0.123 0.234 0.345 1.0
```

```
vt 0.500 1 [0]
```

```
vn 0.707 0.000 0.707
```

Archivos .OBJ

```
f 1 2 3
f 3/1 4/2 5/3
f 6/4/1 3/5/3 7/6/5
f 7//1 8//2 9//3
```

```
vertices = []
normals = []
textCoords= []
faces = []

with open(filename, 'r') as file:
    for line in file.readlines():
        aux = line.strip().split(' ')

        if aux[0] == 'v':
            vertices += [[float(coord) for coord in aux[1:]]]

        elif aux[0] == 'vn':
            normals += [[float(coord) for coord in aux[1:]]]

        elif aux[0] == 'vt':
            assert len(aux[1:]) == 2, "Texture coordinates with different than 2 dimensions are not supported"
            textCoords += [[float(coord) for coord in aux[1:]]]
```

```
elif aux[0] == 'f':
    N = len(aux)
    faces += [[readFaceVertex(faceVertex) for faceVertex in aux[1:4]]]
    for i in range(3, N-1):
        faces += [[readFaceVertex(faceVertex) for faceVertex in [aux[i], aux[i+1], aux[1]]]]
```

```
aux = faceDescription.split('/')

assert len(aux[0]), "Vertex index has not been defined." # is there a vertex index?

faceVertex = [int(aux[0]), None, None]

assert len(aux) == 3, "Only faces where its vertices require 3 indices are defined." # are both texture index and normal index declared?

if len(aux[1]) != 0: # texture index is not empty
    faceVertex[1] = int(aux[1])

if len(aux[2]) != 0: # normal index is not empty
    faceVertex[2] = int(aux[2])

return faceVertex
```

```
    # Per previous construction, each face is a triangle
    for face in faces:
        # Checking each of the triangle vertices
        for i in range(0,3):
            vertex = vertices[face[i][0]-1]
            texture = textCoords[face[i][1]]
            normal = normals[face[i][2]-1]

            vertexData += [
                vertex[0], vertex[1], vertex[2],
                color[0], color[1], color[2],
                normal[0], normal[1], normal[2]
            ]

        # Connecting the 3 vertices to create a triangle
        indices += [index, index + 1, index + 2]
        index += 3

    return bs.Shape(vertexData, indices)
```

Nos vemos!