IMA208

Vision 3D et vidéo

TP7 - Practical work: range scans to meshes

Lorenza Martins Guimaraes Tarallo

Franz Masatoshi Yuri

April 2025

1 Introduction

2 Development

2.1 Code development

2.1.1 Delauney Filter

In this project, it was proved to us a code that imported a file with multiple point coordinates and created a list of triangles with these points.

Our approach was, therefore, to first create a function find_circles with a list of 3 coordinates as input and the radious size as output:

```
def find_circles(triangle):
    a = np.linalg.norm(triangle[0] - triangle[1])
    b = np.linalg.norm(triangle[1] - triangle[2])
    c = np.linalg.norm(triangle[0] - triangle[2])

    p = (a + b + c) / 2
    area = (p * (p - a) * (p - b) * (p - c)) ** 0.5

    if area < 1e-12:
        return float('inf')

    return a * b * c / (4 * area)
```

With this function in hand, we could modify the code given so that in the loop created only triangles circumscribed to circles with radii smaller than alpha are added to a triangles list:

```
with open("c:/path/Bunny.xyz", "r") as f:
    vertex_strings = f.readlines()

# Convert the vertex strings to a NumPy array of shape (N, 3)
points3D = np.zeros((len(vertex_strings), 3))
for i, vertex_str in enumerate(vertex_strings):
    vertex_arr = [float(coord) for coord in vertex_str.strip().split()]
    points3D[i] = vertex_arr


tri = Delaunay(points3D)

triangles = []

for tetra in tri.simplices:
    for k in range(len(tetra)):
        if find_circles([points3D[tetra[k%4]],
        points3D[tetra[(k+1)%4]],
        points3D[tetra[(k+2)%4]]]) < alpha:
            triangles.append([points3D[tetra[k%4]],
            points3D[tetra[(k+1)%4]],
            points3D[tetra[(k+2)%4]]])
```

2.1.2 exporting the triangles

Finally, we created a function that receives the triangles list and a filename and creates a .stl file.

```
def export_to_stl(triangles, filename="output.stl"):
    with open(filename, "w") as f:
        f.write("solid filtered_mesh\n")
        for triangle in triangles:
            p1, p2, p3 = triangle

            # Compute face normal
            v1 = p2 - p1
            v2 = p3 - p1
            normal = np.cross(v1, v2)
            norm = np.linalg.norm(normal)
            if norm == 0:
                normal = np.array([0.0, 0.0, 0.0])
            else:
                normal = normal / norm

            f.write(f"  facet normal {normal[0]} {normal[1]} {normal[2]}\n")
            f.write("    outer loop\n")
            f.write(f"      vertex {p1[0]} {p1[1]} {p1[2]}\n")
            f.write(f"      vertex {p2[0]} {p2[1]} {p2[2]}\n")
            f.write(f"      vertex {p3[0]} {p3[1]} {p3[2]}\n")
            f.write("    endloop\n")
            f.write("  endfacet\n")
        f.write("endsolid filtered_mesh\n")
```

2.2 Results

After developing the code, we evaluated the impact of the alpha parameter in the alpha-shape reconstruction algorithm. Our objective was to determine an optimal alpha value that would generate a visually good mesh while preserving the geometric details of the original scanned object, the Stanford Bunny. The quality of the reconstruction was visually assessed, focusing on the number of holes, the smoothness of the surface, and the recognizability of the shape.

We tested several values of alpha ranging from 0.01 to 0.0015. When alpha was too small (e.g., 0.001), the resulting mesh was incomplete and had many holes. This behavior is due to the strict radius constraint, which excludes larger triangles that are necessary to bridge sparse regions. On the other hand, for very large alpha values, such as 0.01 or higher, the surface became overly smoothed, resulting in the loss of geometric features and unnatural blending of distinct body parts.

For $\alpha = 0.01$, as we can see in Figure 1, the overall shape of the bunny is visible, but the result is over-smoothed, leading to the loss of details. Then, we tested $\alpha = 0.005$, and as shown in Figure 2, the bunny shape became more recognizable, with more preserved features. Attempting to obtain a more accurate reconstruction, we tried $\alpha = 0.002$ (Figure 3), which showed even better detail; however, the mesh was still not ideal for us.

We next tested $\alpha = 0.001$, but, as shown in Figure 4, this value was too small and led to a fragmented and incomplete mesh. Looking for a more balanced result, we tried an intermediate value, $\alpha = 0.0015$ (Figure 5), which showed improvements but still more holes than we desired. After visually analyzing the results, we finally found that $\alpha = 0.0017$, as we can see in Figure 6, provided the best balance: the bunny was clearly identifiable, the surface was mostly closed and the essential shape details were well preserved.

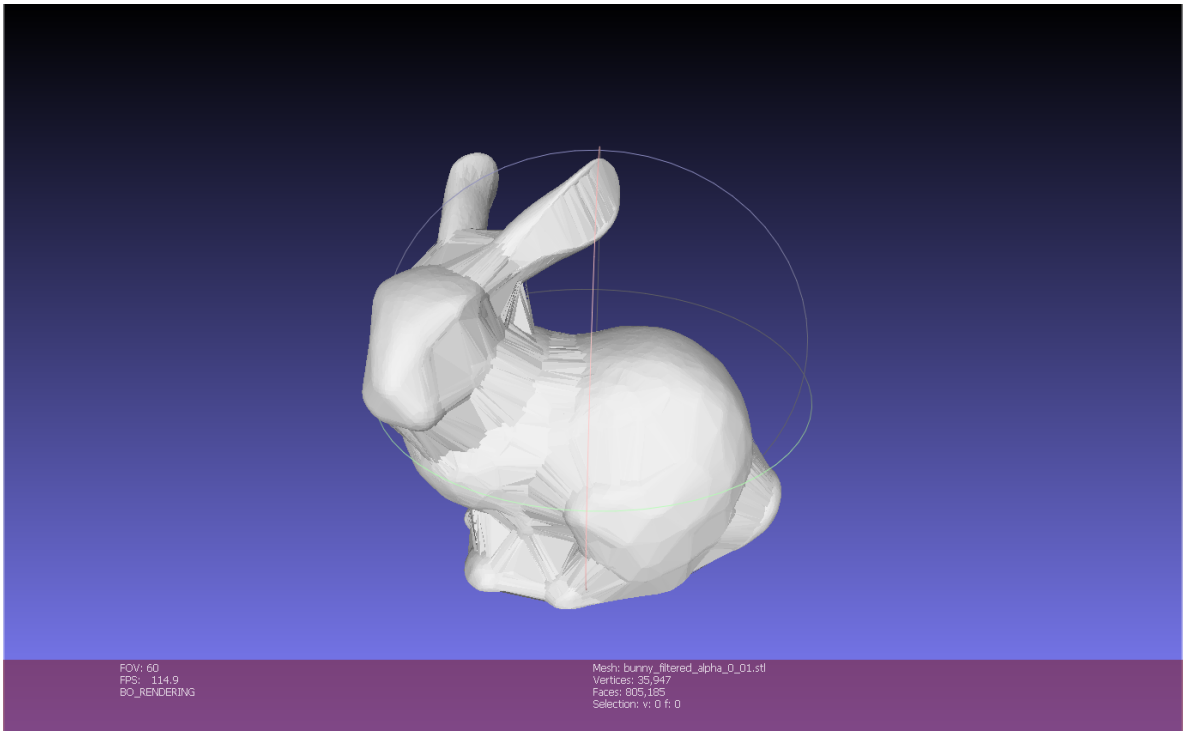

Figure 1: Reconstruction with $\alpha = 0.01$

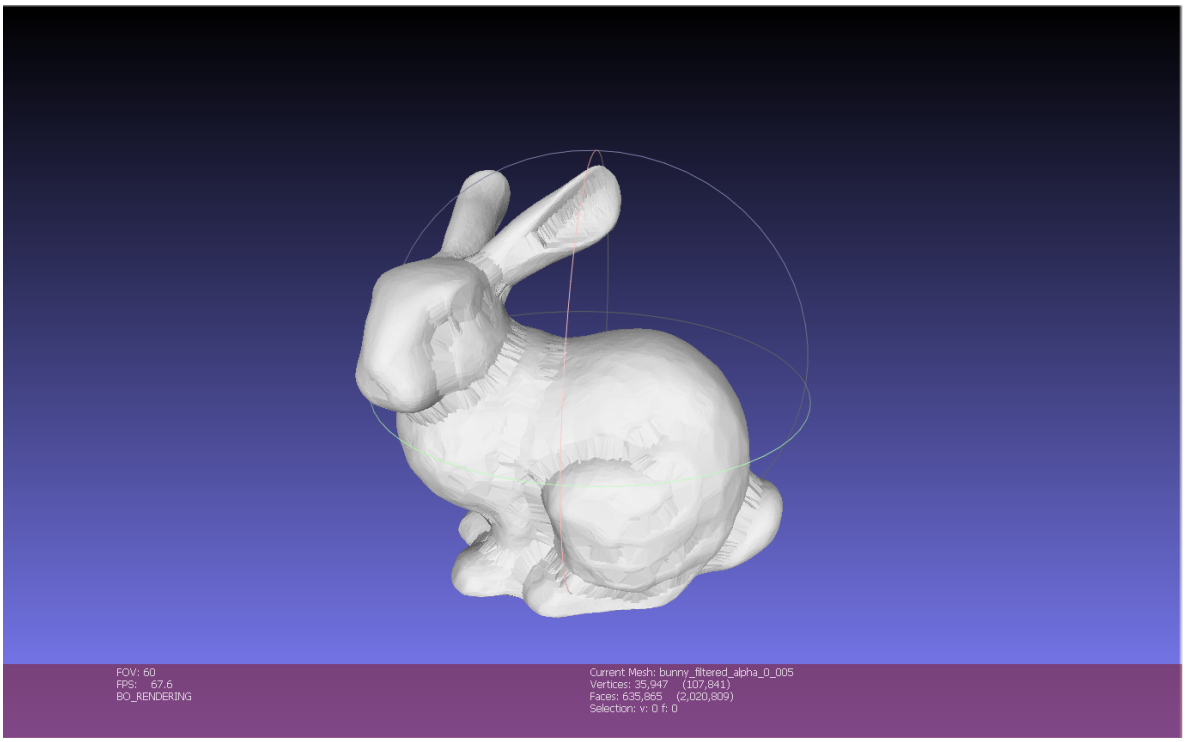

Figure 2: Reconstruction with $\alpha = 0.005$

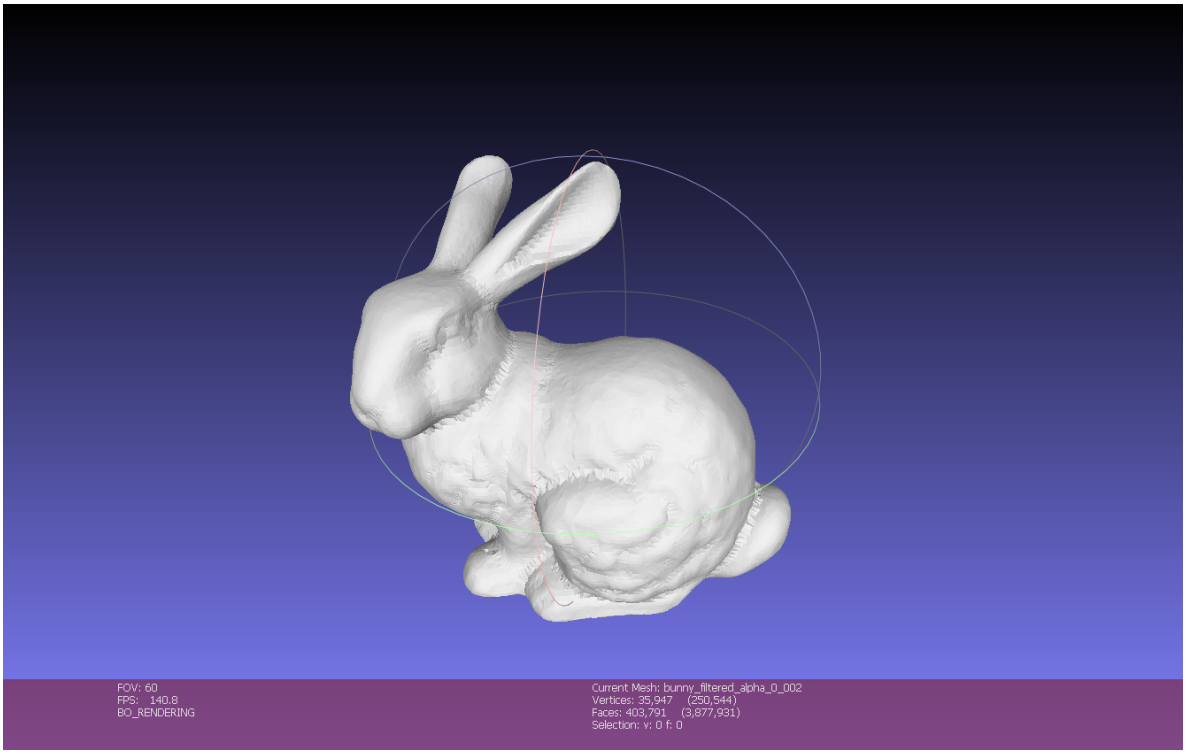

Figure 3: Reconstruction with $\alpha = 0.002$

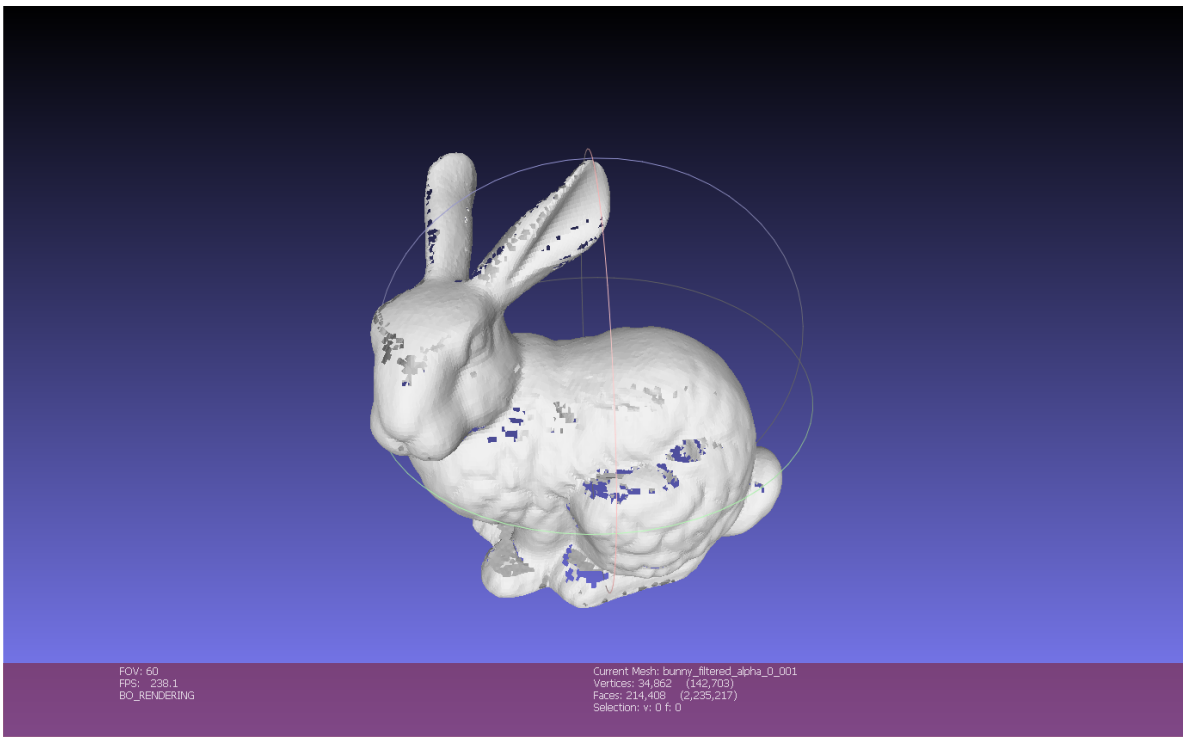

Figure 4: Reconstruction with $\alpha = 0.001$

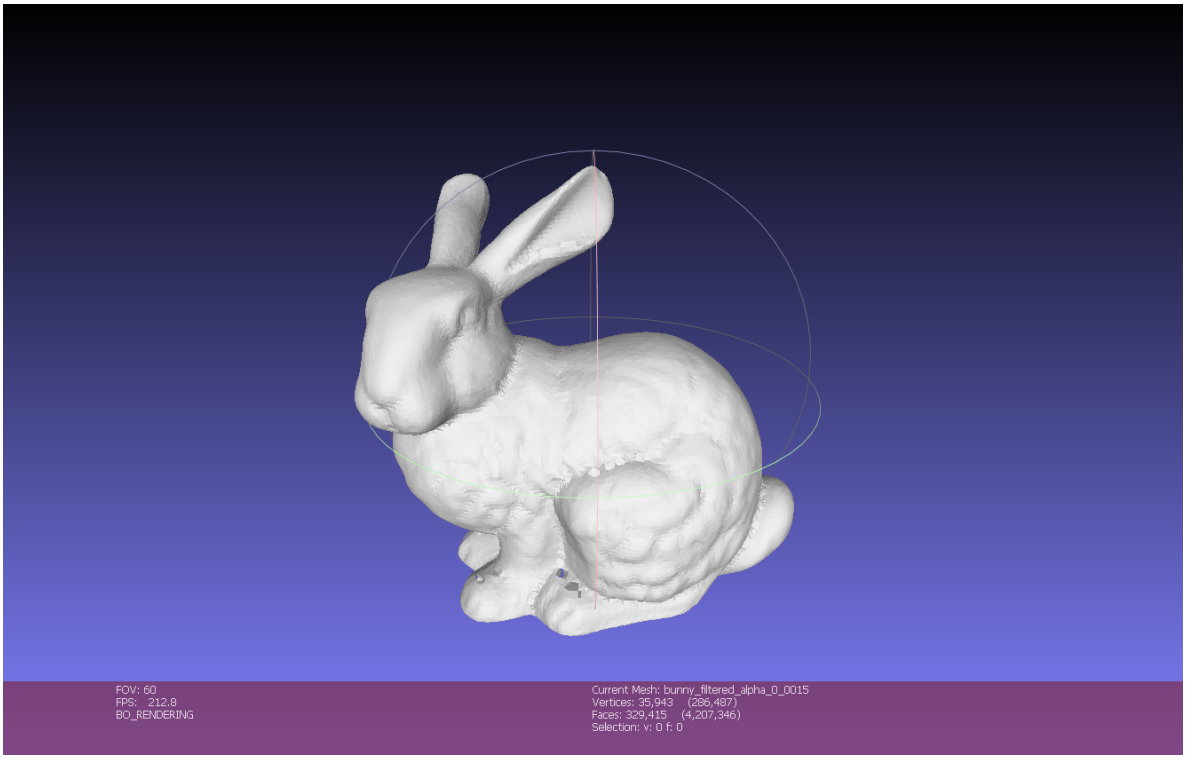

Figure 5: Reconstruction with $\alpha = 0.0015$

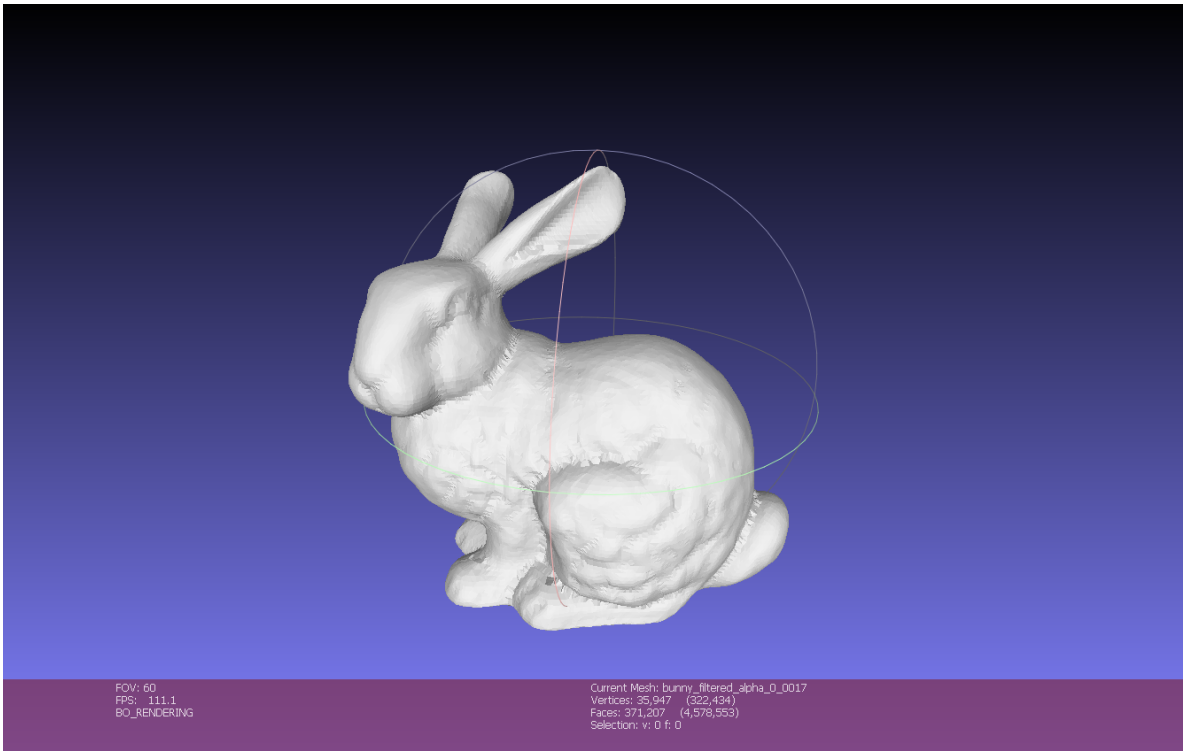

Figure 6: Reconstruction with $\alpha = 0.0017$