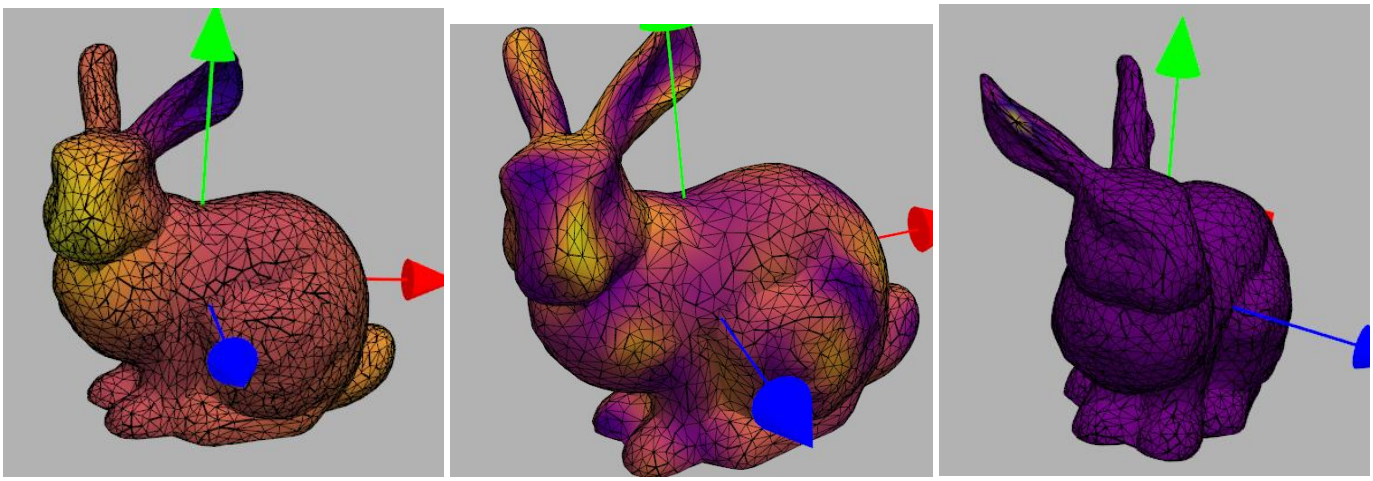
REPORT:

Laplacian Mesh processing

Student: Kiousis Panagiotis, 1092647

TASK 4b:

Eigenvector Participation Below, we observe the degree to which each vertex participates in specific eigenvectors. (Color Key: minimum value -> purple, maximum value -> yellow)



- Left: 1st Eigenvector

Here, we see that vertices located in areas with "sharp" angles (mainly the ears) participate minimally or not at all. This is logical, as the 1st eigenvector corresponds to the constant eigenvector—meaning a state where all vertices have approximately the same value or vary very smoothly.

- Center: A Random Eigenvector

This eigenvector corresponds to a specific variability in the coordinates, so the yellow areas are where this variability exists.

- Right: Last Eigenvector

Here, we see that only a single vertex participates, and this vertex is the reason this specific eigenvector was created.

Laplacian Matrix Properties

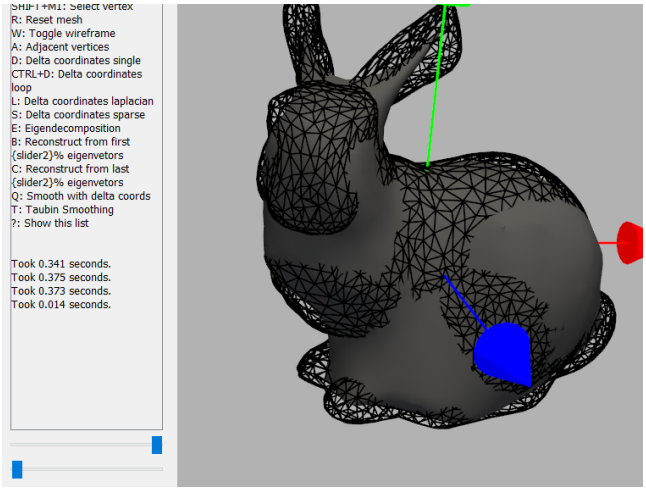
The Laplacian matrix is symmetric and positive semi-definite. What does this mean for its eigenvalues and eigenvectors?

- The Laplacian matrix has strictly real and non-negative eigenvalues.
- Additionally, there is at least one eigenvalue equal to zero, and the corresponding eigenvector is the constant vector with all elements being equal (like the vector of ones), provided the graph is connected.
- Finally, the eigenvectors of L can be selected to form an orthonormal set, a fact that arises from the symmetry of the matrix.

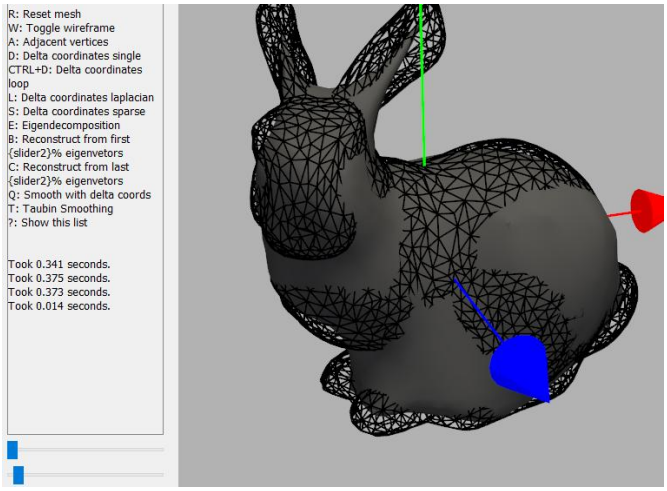
TASK_4c:

Reconstruction

- **Reconstruction with the first eigenvectors:** Reconstructing the model with the first eigenvectors eliminates the details, resulting in a coarser approximation of the model.

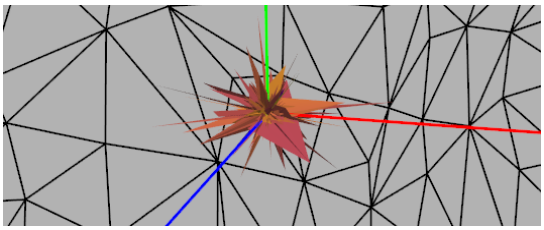


Ανακατασκευή με τα τελευταία ιδιοδιανύσματα:



- **Reconstruction with the last eigenvectors:** The result looks almost identical to the previous case. This happens logically because in the code, even though we are supposedly reconstructing with the last ones, we still keep the first 1% so that the model retains its meaning. If we did not keep them, the general shape of the rabbit would be lost. The fact that it does not differ much from the previous one is due to the slider being set to a very low value. It is likely that the very last few eigenvectors do not significantly affect the model.

- **Experiment:** Tweaking the code, I tried to reconstruct the model using the high eigenvectors without keeping *any* of the low ones.



Result: With significant zoom, the result is visible in the left image. At first glance, it makes no sense, but this is logical since we are not retaining the "coarse" information (the rabbit shape). It is like having a signal and retaining only the high-frequency changes or the noise.

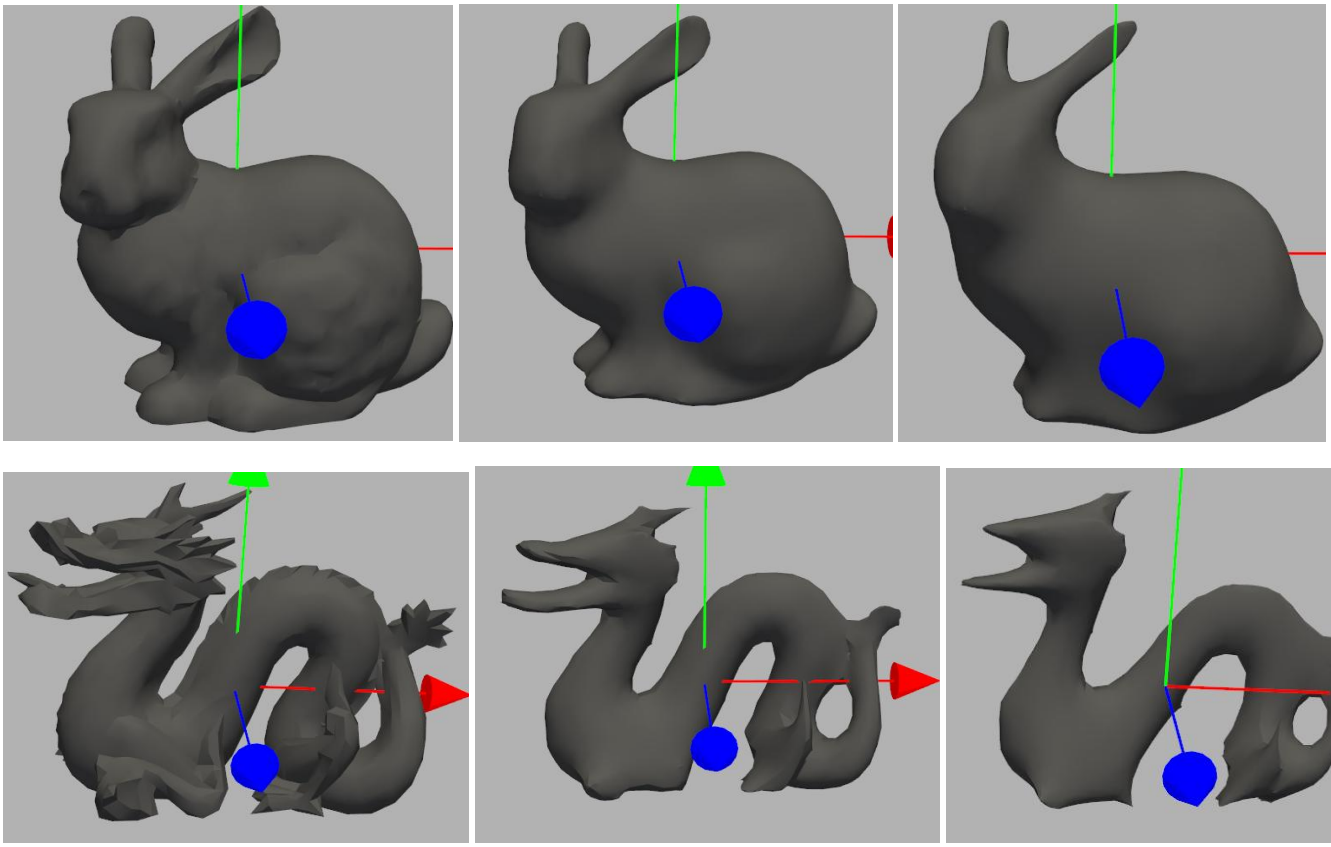
TASK 5a:

Implementation of Laplacian Smoothing:

The function smooth_with_delta_cord() implements an iterative mesh smoothing method based on local vertex shifts.

- **Process:** In each iteration, the mesh vertices are retrieved. For each vertex, I need to find the vector of differential coordinates.
- **Implementation:** I use the function delta_coordinates_single(). However, this requires the Index of the current vertex. For this, I utilize the existing function find_closest_vertex(), which returns an index.
- **The Vector:** The differential coordinates vector starts at the average (mean) of the neighbors and ends at the current point.
- **Goal:** The goal is to move the point by a percentage towards the average of its neighbors. Therefore, I subtract 80% of the differential coordinates vector from the point's coordinates.
- **Update:** It is important that after every iteration, I update the model's vertices so that the new differential coordinates are calculated correctly in the next iteration.
- **Completion:** After the loops complete, I update the scene: updateShape("mesh").

(Below shows the result of Laplacian smoothing after 5 and 20 iterations)



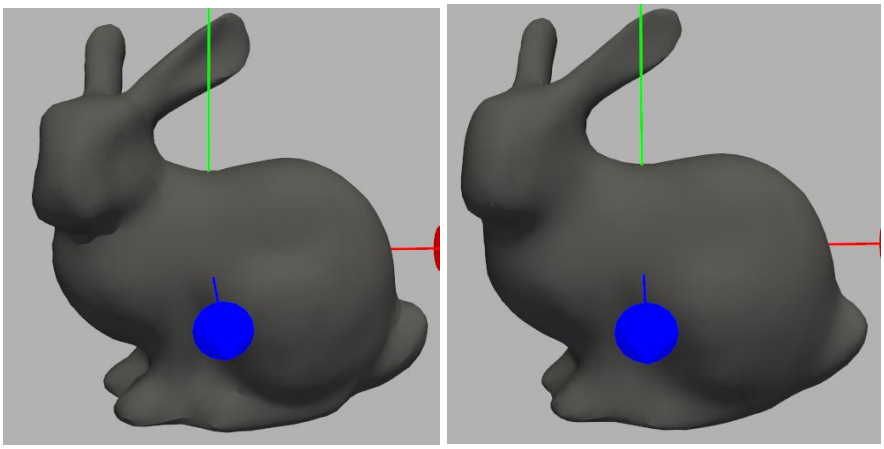
TASK 5b:

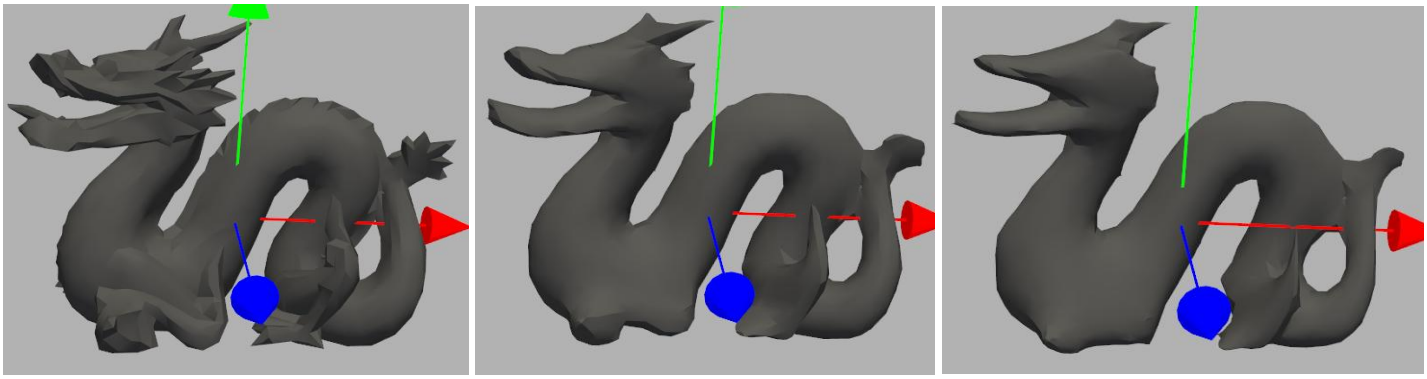
Implementation of Taubin Smoothing:

The function smooth_with_taubin() is similar to smooth_with_delta_cord().

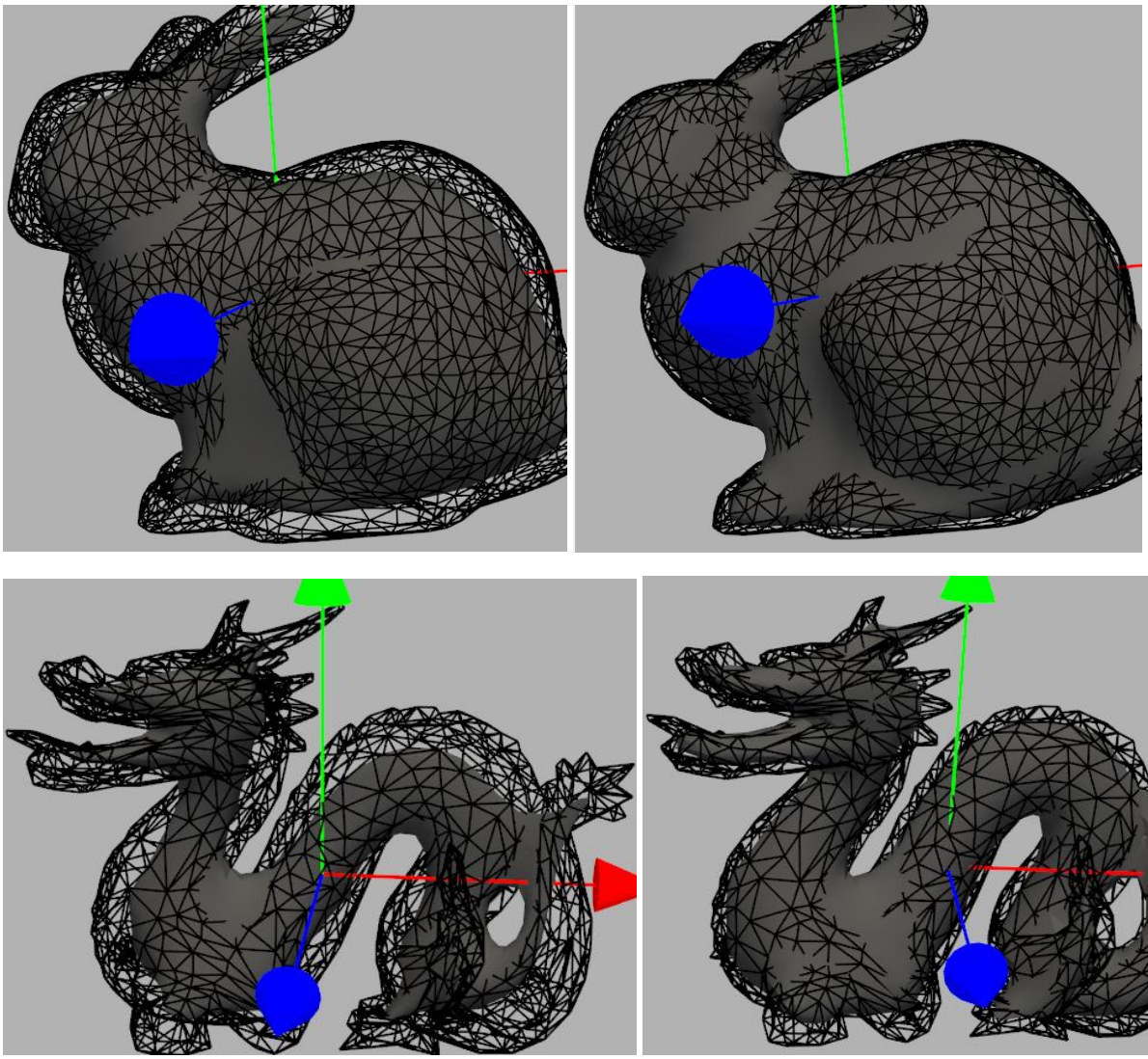
- **Difference:** The only difference is that in each iteration, after subtracting 80% of the differential coordinates vector, I then find the *new* differential coordinates and **add** 60% of them back.
- **Result:** In this way, smoothing is achieved while "shrinking" is significantly mitigated.

(Below shows the result of Taubin smoothing after 5 and 20 iterations)





Comparison: Shrinking in Laplacian (left) vs Taubin (right) smoothing after 20 iterations



Καθώς το wireframe διατηρείται από το αρχικό μοντέλο, φαίνεται πως με το taubin smoothing η σμίκρυνση ελαττώνεται .