Assignment 2 (Theoretical)

This assignment will be automatically graded, to receive full points you will need to complete the functions listed in the following directory structure.

- assignment2
    - rotation
        - ___init___.py
        - axis_of_rotation.py
            - `rotation_component()`
            - `axis_of_rotation()`
        - test.py
    - planes
        - ___init___.py
        - distance_to_planes.py
            - `SquaredDistanceToPlanesSolver.__init__()`
            - `SquaredDistanceToPlanesSolver.sum_of_squared_distances()`
            - `SquaredDistanceToPlanesSolver.optimal_point()`
        - test.py
    - smoothing
        - ___init___.py
        - explicit_laplaces_smoothing.py
            - `explicit_laplace_smooth()`
            - `build_combinatorial_laplacian()`
        - test.py

Because this is a theoretical assignment, *only* these functions need to be implemented. All other UI code and boilerplate is provided.

Using the UI

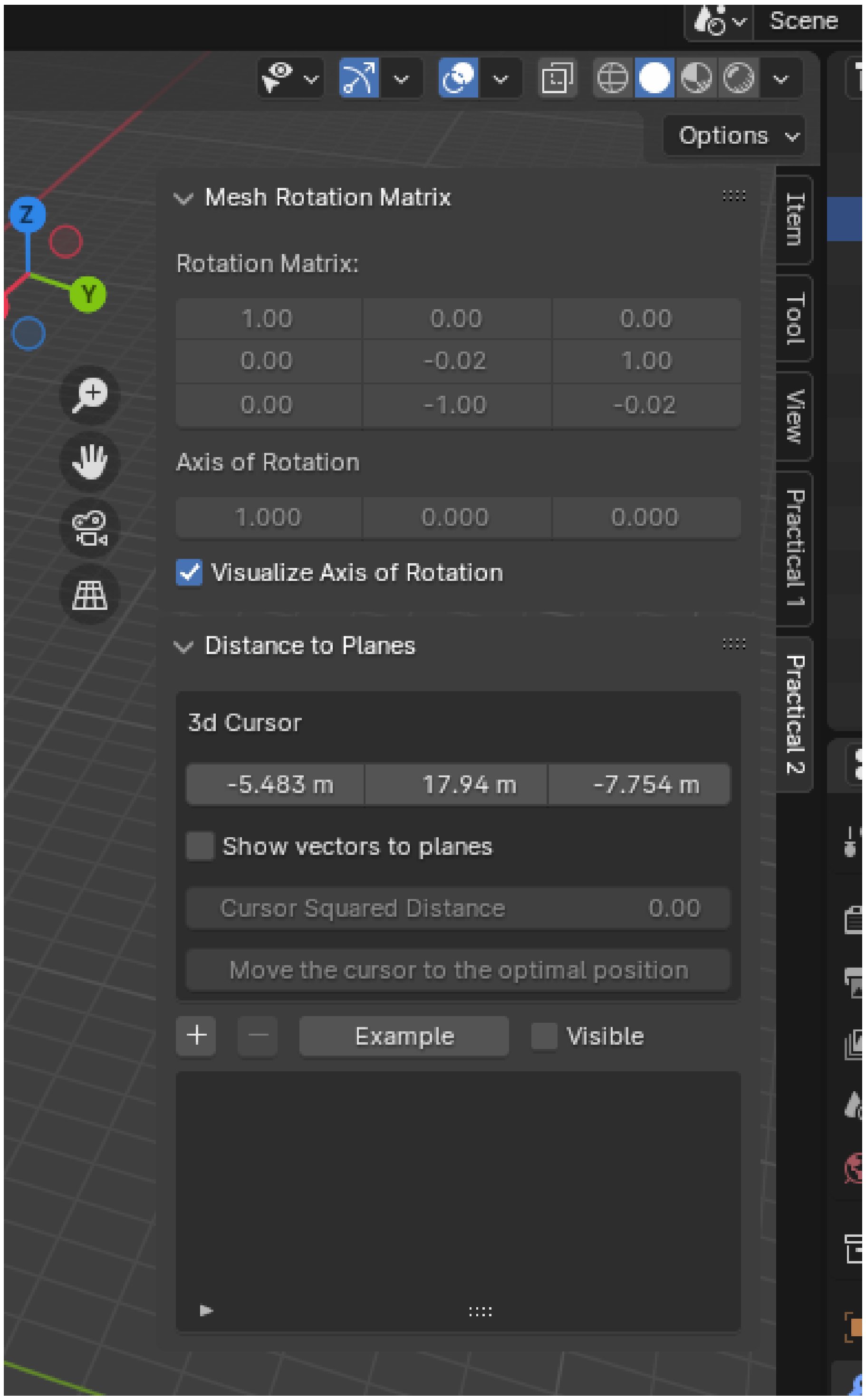
Each task in this assignment comes with an associated UI component (implemented in `__init__.py` as before). These can be useful for visually testing your code and gaining an intuition for what it's doing.

Panels

Like assignment 1, the first tasks of this assignment have associated panels. These can be found in the 'Assignment 2' tab to the left of the object tree, as shown in Fig. 1.

In the Mesh Rotation Matrix panel, you can check the 'Visualize Axis of Rotation' checkbox to see a Blender 'gizmo' overlay which shows the axis and angle of the world transformation matrix for the currently selected object (Fig. 2).

This shows the cumulative rotation of the transformations you have done on the currently selected object. When you apply a rotation about the X axis and then another about the Y axis, the gizmo will show the net rotation about a



Mesh Rotation Matrix

Rotation Matrix:

1.00	0.00	0.00
0.00	-0.02	1.00
0.00	-1.00	-0.02

Axis of Rotation

1.000	0.000	0.000

☑ Visualize Axis of Rotation

Distance to Planes

3d Cursor

-5.483 m	17.94 m	-7.754 m

☐ Show vectors to planes

Cursor Squared Distance 0.00

Move the cursor to the optimal position

+ − Example ☐ Visible

Figure 1: Panels for Axis of Rotation and Distance to Planes tasks.

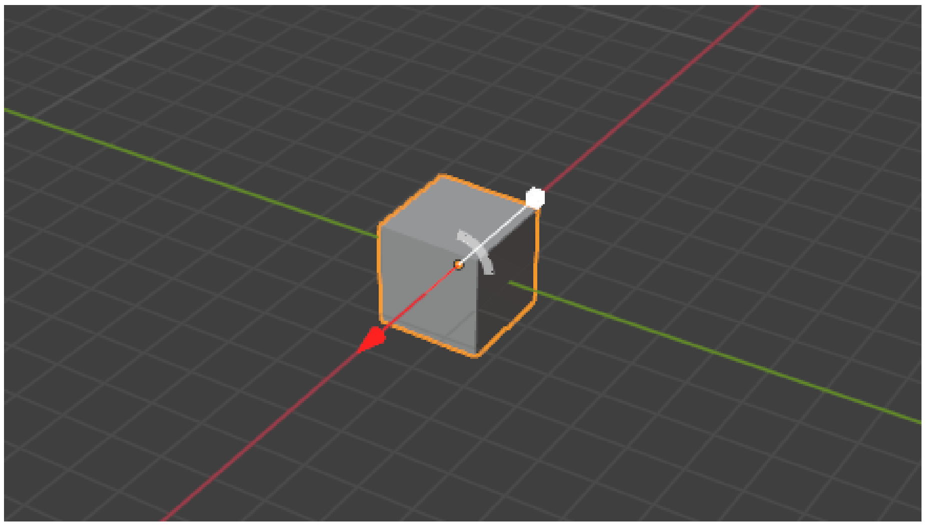


Figure 2: Axis and Angle of Rotation gizmo.

combined axis. You can reset the world transformation matrix by applying it with `CTRL+A`, this ‘bakes’ it into the underlying model.

In the Distance to Planes matrix, you can see the output of your `SquaredDistanceToPlanesSolver` implementation. To test it, you can use the `+` button to add random planes, check the `Visible` checkbox to see those planes in the viewport (Fig. 3).

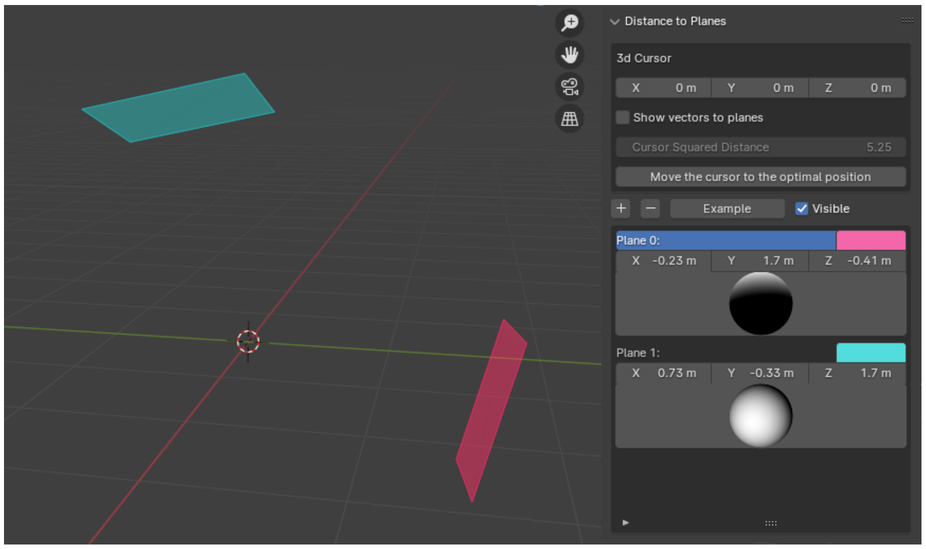
The planes are visualized as sheets, but they are defined as points (adjustable in the list by changing X, Y, and Z) with normals (adjustable by rotating the sphere). This means that they actually extend infinitely in 3d space.

Alongside the list of planes is the current coordinate of the 3d cursor (Fig. 4) and its sum of squared distances to the planes. You can adjust the position of the 3d cursor by adjusting X, Y, and Z directly or by moving it around with `SHIFT+RIGHTCLICK`.

You can check the `Show vectors to planes` checkbox to see the shortest vectors which connect each plane to the 3d cursor (Fig. 5). The `Cursor Squared Distance` box shows the sum of squared distances, equivalent to the sum of lengths squared for all of these vectors.

You should experiment with moving the cursor to see how small you can make the squared distance by hand. The `Move the cursor to the optimal position` button solves a linear system to find the location which minimizes this value. In the case of these two planes, this is at their intersection, making the squared distance zero (Fig. 6).

When there are many randomly placed planes there is typically exactly one



3d Cursor

X 0 m Y 0 m Z 0 m

☐ Show vectors to planes

Cursor Squared Distance 5.25

☑ Visible

Plane 0:

X -0.23 m Y 1.7 m Z -0.41 m

Plane 1:

X 0.73 m Y -0.33 m Z 1.7 m

Figure 3: A pair of planes, visualized as sheets.

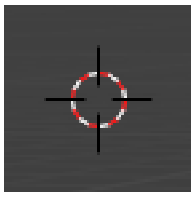
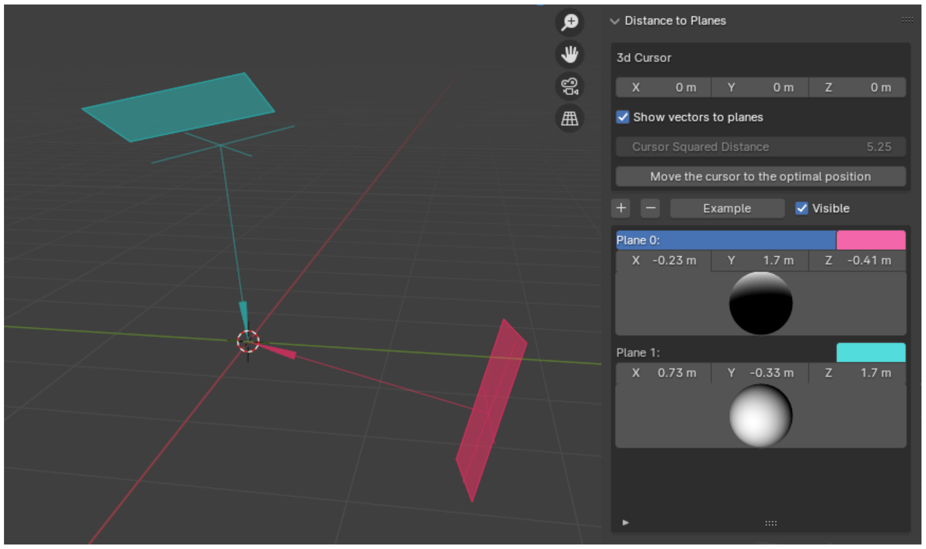


Figure 4: 3d cursor


Distance to Planes

3d Cursor

X 0 m Y 0 m Z 0 m

☑ Show vectors to planes

Cursor Squared Distance 5.25

Move the cursor to the optimal position

+ − Example ☑ Visible

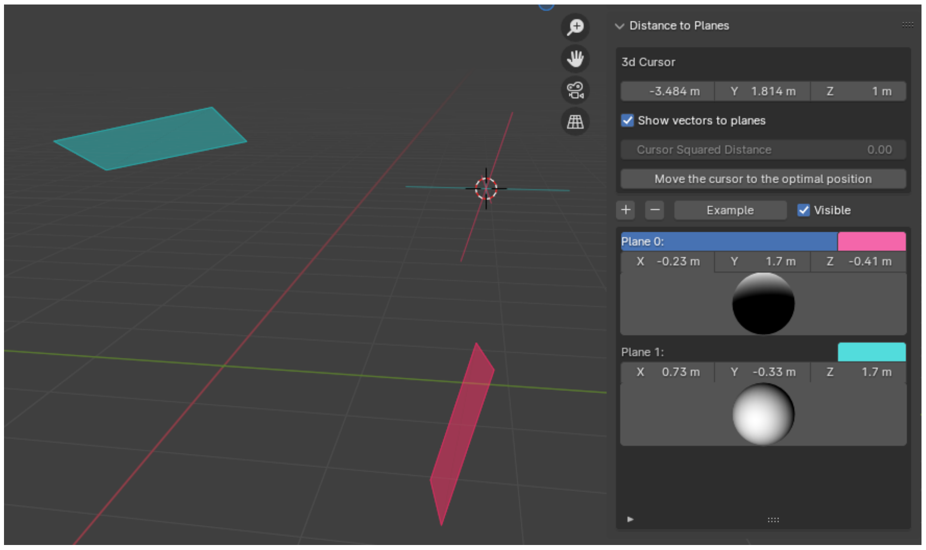
Plane 0: 🟪

X -0.23 m Y 1.7 m Z -0.41 m

Plane 1: 🟦

X 0.73 m Y -0.33 m Z 1.7 m

Figure 5: Vectors from the 3d cursor to the nearest point on each plane.


Distance to Planes

3d Cursor

-3.484 m Y 1.814 m Z 1 m

☑ Show vectors to planes

Cursor Squared Distance 0.00

Move the cursor to the optimal position

+ − Example ☑ Visible

Plane 0: 🟪

X -0.23 m Y 1.7 m Z -0.41 m

Plane 1: 🟦

X 0.73 m Y -0.33 m Z 1.7 m

Figure 6: The cursor, placed at an optimal position which minimizes the sum of distances squared.

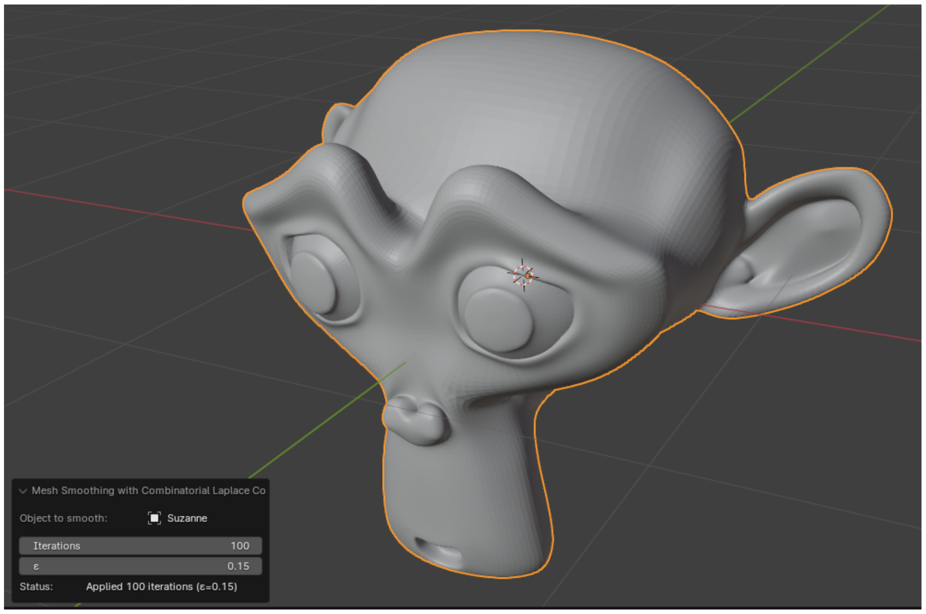
minimal solution, but it doesn't always result in a distance of zero. The `Example` button (next to `+` and `-`) sets up a set of 6 planes which together describe a randomly-placed cube. In this case, where should the cursor be placed to minimize the distance?

In cases of one or two planes, the solution is under-constrained (your solver doesn't necessarily need to handle these cases). For a single plane, the optimal point can be anywhere on that plane. Consider: what would solutions look like for a pair of planes which intersect? What about a set of parallel planes?

Hint: When testing this in the UI, you may want to see set up simple cases by hand. Just as with the objects, you can save scenes and re-load them – your planes will be preserved. If you want to hide the gizmos by default, you can edit the `register()` method at the bottom of `__init__.py`.

Operator

Unlike the rotation and planes tasks, Laplacian Smoothing is defined as an operator. Like the ICP operator from assignment 1, this can be accessed through `Object>Mesh Smoothing with Combinatorial Laplace Coordinates`.



Mesh Smoothing with Combinatorial Laplace Co

Object to smooth: Suzanne

Iterations 100

ε 0.15

Status: Applied 100 iterations (ε=0.15)

Figure 7: Suzanne the monkey, subdivided and then smoothed for 100 iterations.

This operator applies to the currently selected mesh, and you can adjust its hyperparameters to see their effects. A well optimized implementation will allow you to drag the `Iterations` slider and see the mesh progressively smoothed in real-time (Fig. 7).

Unit Tests

Like with assignment 1, we've provided only a handful of simple unit tests in each task's respective `test.py`. While we won't be grading you on the thoroughness of your testing, we recommend writing a few of your own unit tests to make sure your implementation accounts for edge cases!