5 Optimization of Polygon Meshes

This exercise focuses on optimization problems on polygon meshes. *This exercise is optional. You can use it to earn up to 20 bonus points.*

5.1 Theory (5 Points)

5.1.1 2D Registration (5 Points)

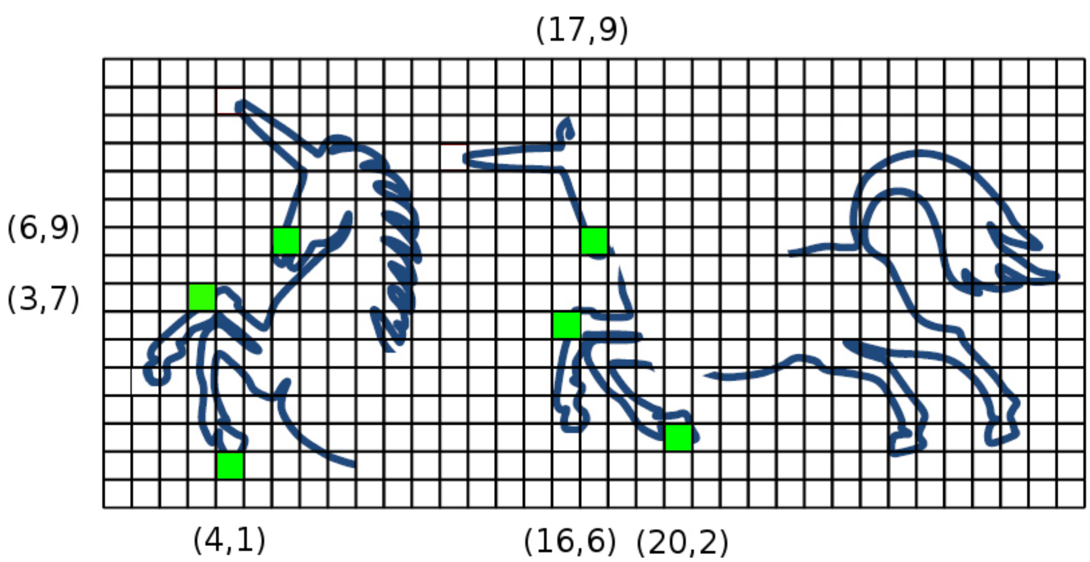

Figure 1: Two partial scans

Figure 1 shows two partial scans that are supposed to be registered into a common coordinate system using a rigid body transform. Corresponding points are marked in both scans. Calculate the optimal rotation and translation that aligns the corresponding points to each other using the Kabsch algorithm. (**2 Points**)
Make sure that your rotation matrix has determinant $+1$.

What kind of transform (that is not a rotation) can be represented with an orthogonal matrix? (**1 Point**)
How many corresponding point pairs are required to uniquely define a rigid body transform? Explain your answer. (**1 Point**)
Is the number of points a sufficient condition or are there cases where fulfilling the minimum number of points requirement still leaves the transform ambiguous? Explain your answer. (**1 Punkt**).

5.2 Practical Part (10 Points)

This practical part will again use the OpenMesh halfedge data structure from previous exercises.

5.2.1 Parametrization (5 Points)

In this task, you will calculate 2D texture coordinates for a polygon mesh. The approach assumes that the mesh is topologically equivalent to a disk. If the input mesh is not disk-shaped, it has to be cut into disk-shaped parts (not part of this exercise).

Implement the calculation of texture coordinates using a Laplacian system in `ComputeParametrization-OfTopologicalDisk` in file `Parametrization.cpp` (**5 Points**). Consider the following hints:

This task utilizes 2D texture coordinates that are stores as vertex attributes. To set a texture coordinate for a vertex, use the `mesh.set_texcoord2D` method.

Start by calculating the texture coordinates $(u_i, v_i)^T$ of vertices on the mesh boundary by distributing the along a circle of radius 0.5 centered at $(0.5, 0.5)^T$. The angular position on the circle should be chosen in the same ratio as the distance of the corresponding vertices along the boundary loop. The method `mesh.calc_edge_length(halfedgeHandle)` can be used to calculate the length of a halfedge.

The inner vertices should be placed in the weighted centroid of its one-ring:

$$u_i = \frac{1}{\sum_{j \in N_1(i)} w_{ij}} \sum_{j \in N_1(i)} w_{ij} u_j \tag{1}$$

$$v_i = \frac{1}{\sum_{j \in N_1(i)} w_{ij}} \sum_{j \in N_1(i)} w_{ij} v_j \tag{2}$$

The function `mesh.is_boundary(handle)` can be used to test if a vertex lies on a boundary. Implement the calculation with the following weights w_{ij}:

1. `CONSTANT_WEIGHT`: $w_{ij} = 1$
2. `EDGE_LENGTH_WEIGHT`: $w_{ij} = \|\underline{p}_i - \underline{p}_j\|$, where $\underline{p}_{i/j}$ is the position of the i/j-th vertex
3. `INV_EDGE_LENGTH_WEIGHT`: $w_{ij} = \frac{1}{\|\underline{p}_i - \underline{p}_j\|}$
4. `COTAN_WEIGHT`: $w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}$ (see Laplace-Beltrami operator from the lecture). The function `mesh.calc_sector_angle(halfedgeHandle)` is useful for this weight.

If you write all texture coordinates u_i and v_i in separate vectors U and V, you can state the above linear conditions in the form two linear systems of the form $AU = B$ and $AV = C$. Each row of the system corresponds to a vertex i. Construct the system using a sparse matrix and solve it using the methods of the Eigen library (see `https://eigen.tuxfamily.org/dox/group__TopicSparseSystems.html`). We recommend the `Eigen::SparseLU` solver.

When you implement everything correctly, you can review the parametrization on the surface and in 2D space. Use the according GUI elements. Again, compilation in release mode is preferable.

5.2.2 Registration (5 Points)

Implement `CalculateRigidRegistration` in `Registration.cpp` that calculates the optimal rotation and translation for a given set of point-to-point correspondences (**5 Points**). Represent the rigid body transform as `Eigen::Affine3f`. The transform must be oriented such that for a correspondence pair (p_i, q_i) q_i is mapped to p_i, i.e. $T \cdot q_i = p_i$. Use the `Eigen::JacobiSVD<Eigen::Matrix3f>` class for singular value decomposition.

You can test the function by activating the *Render Second Mesh* option via the GUI. Using the additional buttons, you can generate correspondences and add noise. The *Register* button calls your registration method and applies the result to the blue mesh. If everything is implemented correctly, the blue mesh should align perfectly with the reference mesh and produce a flickering rendering (in case of noise-free correspondences)

5.2.3 Bonus Tasks (max. +5 Points)

- Implement a function that partitions a mesh into reasonable topological disks Take a look at Section 3 in Lévy's publication "Least Squares Conformal Maps for Automatic Texture Atlas Generation" (**5 Points**).
- Visualize the area distortion for each triangle. For this, compare the triangle areas (relative to total area) in 3D space and 2D space (**3 Points**).
- Replace the automatic correspondences by nearest-point correspondences. A brute force implementation that tests each triangle is sufficient (**3 Points**).