MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Mesh merging applied in modification of human face models

MASTER THESIS

Matej Lobodáš

Brno, spring 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matej Lobodáš

Advisor: RNDr. Igor Chalás

Acknowledgement

I would like to thank my supervisor RNDr. Igor Chalás for all the help an support he have provided as well as the other members of team working on FIDENTS, especially I would like to thank Bc. Zuzana Ferková for her help with integration.

Abstract

This thesis is analysing algorithms for contour projections and mesh merging. Later most suitable algorithms for merging of face part are chosen. The algorithms are implemented in Java language. The library is created to be used in Fidentis application, where they will be used for merging face parts into the head model. Text contains descriptions of used data structures and implementation details . In the end of the text performance measurements and visual results of implemented algorithms are presented.

Keywords

Mesh merging, Facial features, Mesh segmentation, Active conture, Fidentis, Java

Contents

1 Introduction

Use of 3D virtual models is widely common in many ares: simulations, virtual reality, animations, games, etc. This have created great demand for their creation, while the requirements on level of detail and precision are continuously rising. Direct 3D modeling, despite existence of big amount of high quality tools, is still quite slow process that require skilled and talented person. This leads to development of different approaches in creation of 3D models. In many cases scanning of existing objects using different processes and tools and consequential creations of their virtual replicas is used in many areas, but it requires specialized devices and cannot be used to create models of objects that don't exists, what limits its use. One of others approach is creation of new models from existing models by combing and merging them or their parts together.

This work will be aimed on creation of human face and head polygonal models, but the results would be applicable to broader spectrum of 3D models. The main concern of work will be creation of model from composition of existing models, specifically generation of human face models by merging of different facial features into one model. The result of this thesis will be working module programed in Java that will be integrated into existing application FIDENTIS, application will provide GUI for user input and existing models of heads and facial features that will be processed by implemented algorithms.

In this type of algorithms the quality, accuracy and appealing of the model are most demanded properties, so the work would put emphasis on those elements. The performance stills remain an issue as the users needs to be able relatively quickly see results of theirs input, but it is not required to achieve real-time results, still during the process of module creation I would strive for efficient implementation of selected algorithms.

In chapter 2 few algorithms used in mesh merging and data structures suitable for easy and quick work with mesh surfaces will be described as well as short introduction into face composition.

Next in chapter 3 describes algorithms that will be used to solve aims of this work and reasoning behind their selection is explained.

Further it will describe them more at length with implementation details carried out in creation of the module. The end of this chapter will be dedicated to the integration of created module with FIDENTIS.

Chapter 4 will then presents graphical results of implemented algorithms as well as measurements of performance on models of different parts on head models with low and high polygon counts. It will also include short description of created module.

Last chapter 5 will evaluate quality and introduction of results achieved in this thesis and would propose possible improvements and use in further works.

2 Related work

This chapter will mainly describes different approaches used in applications that have similar purpose as this thesis. The problem of automated mesh connections can be split into two subproblems first is the calculation of borders on which objects will be connected and the second is the process that change shape of merged objects so they can be smoothly attached together and stitch them on their borders. Chapter will describe different algorithms that solve those problems separately, but first it will give brief introduction into history of face composition in digital and non-digital form and will speak about few similar applications. In the end few possible data structures for efficient work with mesh will be described.

2.1 Face composition

In human perception of others people,the face is the most important part of the body as it gives most of the visual informations about its owner. This is why it is one of the most used method for personal identification in daily life or in criminal investigations[3].

The later required some tools for reconstruction of person identity from description of eye wittiness. In the begin, work of artist was required to create sketch of human face. This was quite demanding on specialization and skill of personal. This started to be replaced in the second half of last century by sets of generic templates of different facial features that could be easily assembled into portrait. Usually parts of photography were used[14].

Advancement in modern computer graphic opened new possibilities in this field and virtual generation of faces first in 2D and later in full 3D environment started to replace manual composition physical materials. Software implementation allow easier and quicker manipulation with large database of prototyped face features and also it allows easy change of size and shape modification of the templates. In addition they are able to produce randomly generated faces based on chosen input. This gives alternatives that helps witness to describe suspect as it is easier to describe deviations than to describe it. Example of such applications is EvoFIT[9].

3D environment provides further possibilities for more detailed and precise constructions of human faces. Biggest benefit is ability to choose any arbitrary viewpoint.

Three-dimensional representation of human faces constructed by special devices for capturing of human body and human face is used in project FIDENTS. The application contains database of 3D face representation. The application provide tool for creation of composition of face features on head in similar way as original manual identikits did[13]. The aim of this thesis will be creation of module that will allow merging of prepared composition of face feature surfaces and phantom head into one continues model of human face.

2.2 Contour projections and Mesh segmentation

Mesh segmentation is process that decompose polygonal mesh into sub-meshes according specified criteria. It is required, that the sub-meshes correlate with division into meaningful parts that will be recognized by humans. For problem solved in this thesis segmentation should split human face into sub-meshes that will correlate with facial parts or allow to align source mesh with target mesh based on their topology.

2.2.1 Mesh Scissoring with Minima Rule and Part Salience

In this paper authors based their segmentation algorithm on human perception of borders on continuous object. The minima rule states borders tends to exist in concave discontinuity of the tangent plane[15].

For computation of mesh segmentation this is described by Minimum negative curvatures and is calculated using tensors field for each vertex of object see figure 2.1a. Thresholding is then used to choose vertices that have potential of being part of splitting loop. They are connected into graphs called feature regions see figure 2.1b. The regions are transformed into contours by peeling vertices from the region so the skeleton of graph is constructed figure 2.1c. Contours that don't have significant length are thrown away. Remaining contours are smoothened using geometric snakes and connected together based on similar direction and small distance see figure 2.1d.

For snakes energy minimization over surface affected faces are locally parametrized onto 2D plane[15].

Now when contours are closed the algorithm must choose most salient segments. The process is recursive and one contour at the time is added to final partitioning of object. Two criteria are used to determine quality of splitting contour, its length and centricity. The algorithm aims for good combination of long contour with high centricity. The centricity of vertex is determined by it distance to the rest of the vertices in mesh and average distance. This leads to multiple distance calculation. As new final splits are introduced the centricity needs to be recalculated each time mesh is split[15].

2.2.2 Easy Mesh Cutting

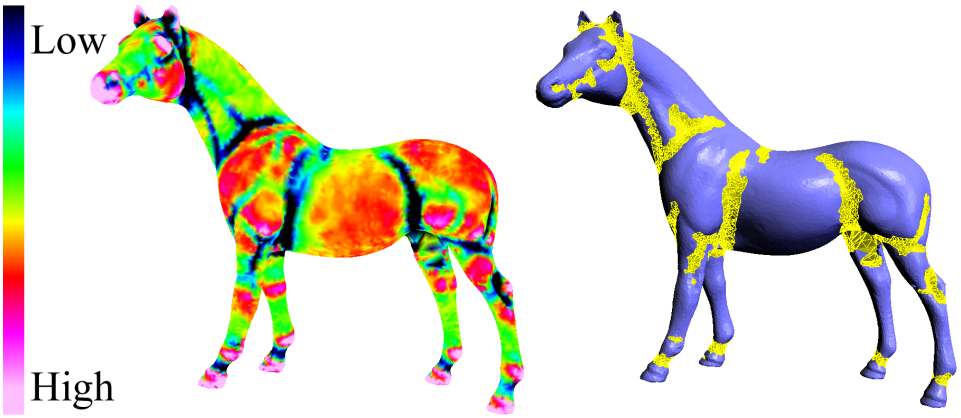
This system is based on user input and splits mesh into two parts, background and foreground. The model that will be segmented is displayed and then user choose the viewpoint. Then wished background and foreground are marked by sketch lines over mash. The sketches are projected on the mesh object in the direction of the view and marked areas are used as starting points of segmentation algorithm.

The segmentation process is based on region growing algorithm. This means only distances of adjective vertices on growing border needs to be computed to determine whether vertex can be added into region as opposed to the others approaches where usually distance between each two point in mesh needs to be calculated (e.g. algorithm form subsection 2.2.1). Isophotic metric is used for distance calculation, what means the curvature of mesh surface defined by its normal is taken into account when distance is calculated. The growth ends when each vertex is included in one of two possible parts[11].

The emerged boundary is then represented as snake. The snake is allowed to move only over vertices and edges of mesh to assure snake is embedded on underlying mesh. The movement of snaxel s_i is governed by its energy defined as $E_{snake}(v_i) = E_{int}(v_i) \times E_{ext}(v_i)$.

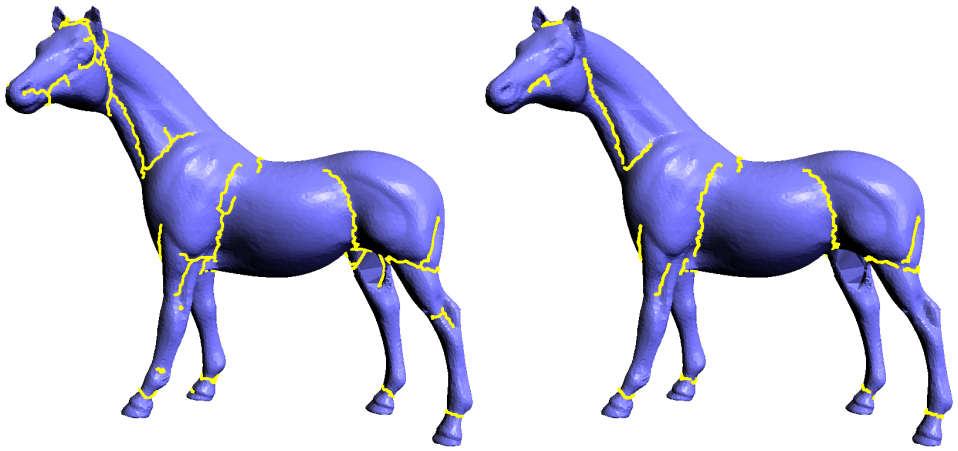
Internal energy is smoothing the edge while externals attracts it to sharp futures. In iterative process the snake energy is maximized by moving, splitting and removing snaxels. After snake energy is maximized and the contour is smoothened user can edit the boundary

Figure 2.1: Visualization of the steps for splitting contour determination (taken from[15]).



(a) Minimum curvature value

(b) Feature regions



(c) Skeletons of regions

(d) Final contours

by sketching short strokes and the boundary try to approximate to it[11].

2.2.3 Active Contour Projection for Mesh Segmentation

This approach combines benefits of parametrized 2D snakes and 3D snakes. As for parametrized 2D snakes snaxels are allowed to be placed on any arbitrary part of surface not only vertices or edges and same as for 3D snakes the contour is evolved over mesh along its natural directions.

Final projection of contour is done in four steps for each snaxel in the same order as they are connected in the snake.

Step 1 Displacment of arbitrary snaxel s_i is computed.

Step 2 Projection face F for displacment of snaxel s_i is found.

Step 3 Displacement of snaxel s_i is projected on face F.

Step 4 New snaxels is inserted if displacment projection is outside of F.

The displacement of snaxel is calculated as sum of inner and outer displacements. Inner displacement is square distance of snaxel from midpoint of its neighboring snaxels. The external displacement is interpolation between gradients of vertices of mesh part on which snaxel is positioned[18].

The neighbor face of snaxel with largest angel of its normal and snaxel's displacement vector is used for projection. The projected displacement may lay outside of the face in such case new snaxel is created with coordinates corresponding with the intersection of face borders and projected displacement vector. The algorithm repeats steps 2 to 4 until projection of all displacement is contained in the target surface[18].

2.3 Mesh merging algorithms

For mesh merging four possible algorithms with different approach are present. They all connect meshes on their openings and with exception of algorithm from subsection 2.3.1 all are based on direct

connection of source mesh to target mesh, where the source mesh is amended to match the opening on target mesh by aligning position of its vertices and then is inserted into target mesh in such manner the resulting mesh is smooth, continues and preserves shapes of target, but won't remove details of source mesh after deformation. The Fusion 2.3.1 algorithm in comparison create new mesh that will smoothly connect multiple meshes on their openings.

2.3.1 Mesh Fusion Using Functional Blending on Topologically Incompatible Sections

In this algorithm two or more openings on mesh surface are stitched together with blending surface that defines missing surface between connected parts. The openings on fused surfaces are converted into implicitly defined functions on plane. Blending surface is created as interpolation between opening using Hermit basis functions. This implicitly defined surface is then tessellated to mesh that cam be connected with input meshes. Hermit interpolation assures tangential continuity across the boundaries. Combined parts are mixed gradually, this allows preservation of details as proximity of vertex on blending surface determines the weight of shape deformation by original meshes [12].

2.3.2 Mesh Editing with Poisson-Based Gradient Field Manipulation

With Poisson mesh solver we look for target mesh with known topology,but unknown geometry.To compute Poisson equation discreet guidance vector field for each of three coordinates is required. For purpose of mesh merging the opening on two meshes are used as Poisson boundary conditions. Three possible approaches for construction of these boundary were implemented in described work. In first approach the boundary of of source is directly projected on target mesh and result of the projection is used as second boundary. Second approach use 2D parametrization of source mesh that is then projected using a mapping scheme such a spherical mapping. The last approach is based on user input. User manually maps few source vertices to target vertices and the rest of boundary is obtained using

interpolation[19]

2.3.3 Optimal Boundaries for Poisson Mesh Merging

The goal of this approach is to find optimal merging boundary on the target mesh. Source mesh is then deformed to match target boundary and then they can be stitched together.

In first step user will choose approximation of surface area on source object Ω_0. It is required to include all important mesh features $\Omega_{features}$ in Ω_0 and additional surface for boundary. Surface of important features $\Omega_{features}$ can be automatically identified using algorithm mentioned in 2.2.2. On target mesh region Ω_1 for pasting features is marked, it shouldn't contain any specific feature as algorithm will remove them to remove distortion[10].

Second step is to find matching position in Ω_1 for each vertex from $\Omega_0 \setminus \Omega Feature$. Input from user is required to align the positions, size and rotation of merged objects, the algorithm will do only fine tuning of these transformations on parametrized surfaces[10].

Now it is required to construct closed loop in $\Omega_0 \setminus \Omega Feature$ that will be optimal boundary of $\Omega_{Features}$. This loop needs to be transformation of loop on omega 1 as much close as possible to unknown similarity transformation[10].

Points of boundary of feature are moved to projected boundary and Poisson merge algorithm is used to deform source mesh and merge it into target mesh.

2.3.4 Mesh Merging with Mean Value Coordinates

Algorithm described in this section is based on premise that mean value coordinates (MVC) of source mesh with respect to original boundary will not be changed after source mesh is aligned with target mesh according boundary on target mesh. The main advantage of this algorithm is that it avoid mesh parametrization and solving of large linear system instead it use closed-form expressions[7].

Mean value coordinates have to meets this criteria. Lets have vertex v_0 and vertices $[v_1, v_2, \ldots, v_n]$ where $[v_1, V_2, \ldots, v_n]$ are arranged in counterclockwise order around v_0 and creates star-shaped polygon, then mean value coordinates of vertex v_0 are items of vector

Figure 2.2: Condition for mean value coordinates.

$$\sum_{i=1}^{n} \lambda_i v_i = v_0 \qquad \sum_{i=1}^{n} \lambda_i = 1$$

(a) Sum of weighted coordinates on polygon equals to center coordinates.

(b) Sum of weights is one.

$[\lambda_1, \lambda_2, \ldots, \lambda_n]$ and the equations 2.2a and 2.2b are fulfilled[8].

MVC could be used for merging of two mesh only when the opening on both mashes have same count of vertices and they are mapped to each other. This can be done either by contour projection or by manual impute from user. When openings ∂S on source mesh S and opening ∂T on target mesh T are matched, mean values can be properly computed. First mean value coordinates S' for mesh S with respect to ∂S. Then S' is applied on difference between opening $\partial S - \partial T$ that interpolates differences for each vertex in mesh S that needs to applied to match with mesh T. After this difference is applied to mesh S, both meshes can be stitched together[7].

2.4 Data structures for polygon meshes

2.4.1 A mesh data structure for rendering and subdivision

There are two most common requirements for mesh data structures subdivision and quick access to data thats enable real time rendering. For the first requirement full topology information must be preserved, for efficiency they should be directly referenced or index to avoid traversal of list or other similar structure. For the real time rendering it is important that the informations can be easily streamlined into graphic card and stored using vertex buffer objects(VBO) [17].

2.4.2 OpenMesh a generic and efficient polygon mesh data structure

In OpenMesh the topology information are stored for each part of the mesh in this manner: Vertex points to one of its outgoing Halfedges; Face points to one of its Halfedges; Halfedge points to, its Face, target vertex, next Halfedge and opposite Halfedge, optionally it can points to previous Halfedge. The rest of the topological information can be obtained by one or more queries on mesh parts[2].

2.4.3 Directed Edges

Directed edges keeps references in similar data structure as OpenMesh2.4.2. The main difference is it use edges rather then half edges. This imply each edge have to store reference to its two end vertices instead of one and both neighboring faces instead of one neighboring face. Also it stores reference to its winged edges (the edges that are directly connected to it and are neighboring with its faces). All references from same type of mesh part are stored in one array. This allow to retrieve items referenced by element of mesh by pointer arithmetic[4].

3 Implementation

3.1 Solution outline

3.1.1 Problem description

The problem solved in this work is merging closed manifold polygonal mesh that represents head with manifold polygonal meshes with borders, that represents different head features as eyes, eyes brows, forehead, nose, mouth and chin. The details of merged surfaces needs to be preserved, but aligned with the shape of head also the textures mapping of merged meshes needs to transfered into final mesh. On the input of algorithm are models of face features positioned on their wished position with respect to the head model. This is done using existing application FIDENTIS, where user can pick from database of existing models of heads and face parts. The user is able to specify position of these parts with relation to head and change their rotation and size. Created framework is able to process this composition and to create merged model from it that can be used by FIDENTIS.

3.1.2 Used algorithms

From algorithms that were studied beforehand and described in chapter 2 Related Work most suitable algorithms were chosen, the possible algorithms comparison and reasons behind decision for particular algorithm will be described bellow.

For determination of opening on target mesh the approach from subsection2.2.3 will be used. Direct projection of contour and its sequential evolution suites best for the input where source mesh is manually position by user. Rather then matching of segments created by algorithms. With chosen algorithm it is easy to create opening that will have matching vertices on opening in target mesh and the borders of source mesh. The others algorithms would require either further user interaction to manually match vertices on openings or implementation of functionality that will do this automatically. Requirement for user input could be possible avoided by using algorithm from subsection 2.3.2, but different approach for connection of

meshes was chosen as described in following paragraph.

For connection of meshes algorithm from subsection 2.3.1 was omitted as it solves slightly different problem and is not providing means to align two meshes, but rather create bridge between them. This approach may be used for ears as they are rather connected to the head as opposed to the rest of the facial features thats needs to inserted into mesh of the head. From the remaining algorithms the one from subsection2.3.4 was picked as in comparison with Poison-Base merging algorithms promise higher performance due to use of closed-form expressions rather then solving of linear systems. It also providers more straightforward approach that will allow easier and less error prone implementation.

3.1.3 Decomposition of merging process

The merging process is dived in several steps that consists of separated actions that should be easily amended or replaced by similar algorithms without changes in the other steps as they were implement with SOLID principles in mind that should keeps all components loosely coupled with minimum of dependencies. Each of these steps is described in more details in later sections.

Steps between first and last step can be repeated in cycle for as many different facial features as possible so multiple source meshes (facial features) can be merged in batch into one target mesh(head). In each next iteration head is already merged with previous facial features so the order of merging will determines the priority on which of surfaces that are merged in same space of had will be used in resulting mesh. This would driven by the order in which facial features were selected in FIDENTIS. The later feature will have priority before those that were selected before it.

1. Doubly connected edge is constructed for model of head.
2. Merging of meshes.
    2.1. Doubly connected edge is constructed for model of face part.
    2.2. Direction in which the part will be project is determined.

2.3. BVH for faces of head is created.

2.4. The outer edges of source mesh are project on head mesh. Parts of source mesh may be cut off in case they are bigger than model of the head.

2.5. Projected contour is evolved over target mesh to match head segmentation.

2.6. The head mesh faces and half-edges are split so half-edges matching with projected contour are part of target mesh.

2.7. All mesh part inside created loop are removed.

2.8. Source mesh vertices are aligned according opening in target mesh.

2.9. All DCEL parts and vertex informations from source are carried over to target mesh.

2.10. Both meshes are stitched together.

2.11. Repeat if there is more models that needs to be merged.

3. Export the result.

3.2 Data structures

3.2.1 Graphic mesh

`GraphicMesh` class is composed from two classes. `PointList` that contains informations about geometry of polygonal mesh as position of vertex in mesh, their normals and texture coordinates used by incident faces. Second data structures holds information about relation of mesh points and faces and materials applied to faces. Graphic mesh stores date in such a way that they can be easily used for storage in .obj file format or quickly buffered into VBO and displayed using openGL. The functionality for this is also part of created module. This class can be used directly for work with mesh that doesn't require use of topological information or can be used as underlying data structure that feeds data to Doubly Connected Edge List (DCEL) that supports more complicated operations on mesh topology.

3.2.2 Doubly connected edge list

This data structure refers to graphic mesh data and keeps informations about theirs topology. Most of the operations in algorithms used in this works are depended on topological relations between parts of graphical mesh and this structure helps to process data in optimal way.

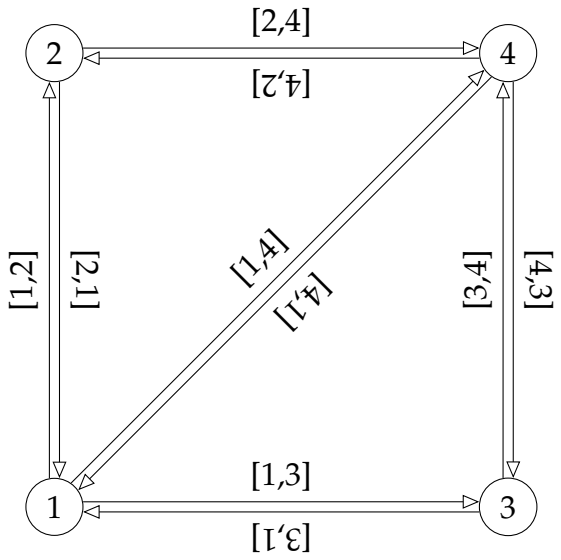
The Doubly Connected Edge List(DCEL) consist from three main parts type vertex, half-edge and face. To get better understating how parts reference each other see figure3.1. Each vertex refers to one of half-edges that have beginning at its position. For half-edges few terms needs to be described first. Two half-edges are always part of one edge that separates two neighboring faces and they have always opposite directions. Half-edges with opposite direction and common vertices are called twin half-edges. The half-edges create loop that define borders of face, that face is incident face of them. Each half-edge then refers to its incident face, next and previous half-edge in the loop and also its twin and vertex it is starting at. Twin half-edge lays between same vertices, but they have opposite direction. The face only refers to one of the half-edges on its borders[6]. The special type of face is outer face, it is not part of the mesh surface, but represents opening in surface or its outside borders. For our purpose we have only one opening at the time in the mesh.

In the figure 3.1 is doubly connected edge list for mesh that consists of for four vertices and two faces with common edge and outer face. In the example half-edge [1,4] have twin half-edge [4,1], its next half-edge is [4,2] and previous is [2,1]. The incident face is face [1,4,2]. The outer face is then define [2,4,1,3].

The DCEL implementation is using triangle faces by default due the requirements of used algorithms, but was made generic so it will work with any type of faces when special properties of triangular faces is not required. This allowed implementation of more simpler and efficient work with faces tham comes from fixed count of vertices and half-edges in face.

To obtain other, as directly stored topological informations, from DCEL combination of reverenced values are used. For example to get end vertex of half-edge, starting vertex of twin is used. It is possible to amend Doubly Linked Connection Edge to store some of these ref-

Figure 3.1: Doubly connected edge list for points 1, 2, 3, 4 and two face [1,4,2] and [1,3,4]and outer face.



erence directly, to avoid overhead from function calls, but it would lead to higher memory consumption. Both approaches were tested and no significant overhead was measured. This is result of JVM using inlining of methods to optimize performance.

To ease up the work with DCEL, classes and methods that encapsulates references between its parts were created. Beside proper OO design, the main point of this encapsulation is to assure the references to other parts and are always update with correct reference when parts are removed, subdivided or new parts are added. When such actions are performed on DCEL additional functionality is invoked that invalid references are updated or parts that cannot exist in DCEL are removed.

The DCEL also supports subdivision of underlying mesh. Two types are supported face is split at inner point and is replaced by multiple faces that are connected in splitting point. The second case is split of half-edge, this will splits also its twin as each half-edge needs to have pair twin twin half-edge. Then each incident face is replaced by two new faces separated by newly created half-edges. For more detailed description see section Insertion of vertex into DCEL3.3.7.

3.3 Contour projection

To be able to merge two mesh together we need first project contour of source mesh onto target mesh in right direction, this is done by direct projection of outer edges of source DCEL on the target DCEL, this will created snake on mesh that is firstly evolved over mesh and the inserted into it.

3.3.1 Determination projection direction

The first approach for direction determination is based on shape of projected contour. The premise is that the mesh that was going to be incorporated into target, should be inserted into mesh in the direction of normal to plane that is approximated by contour as if the texture was projected on the surface using planar projection. Triangle fan that would have edges on contour of source surface and its center is same as the center of contour is used as approximation of such plane. For each triangle in the fan its normal is computed, normal constructed from sum of these normals is then used as direction of projection. This should have simulated the direction in which facial feature would be attached in manual composition. Problems with facial feature as forehead that have contour laying on curved surface resembling cylinder appeared as the parts that were located far from the center suffered with gradually higher shear deformation as it can be seen in figure 3.2a.

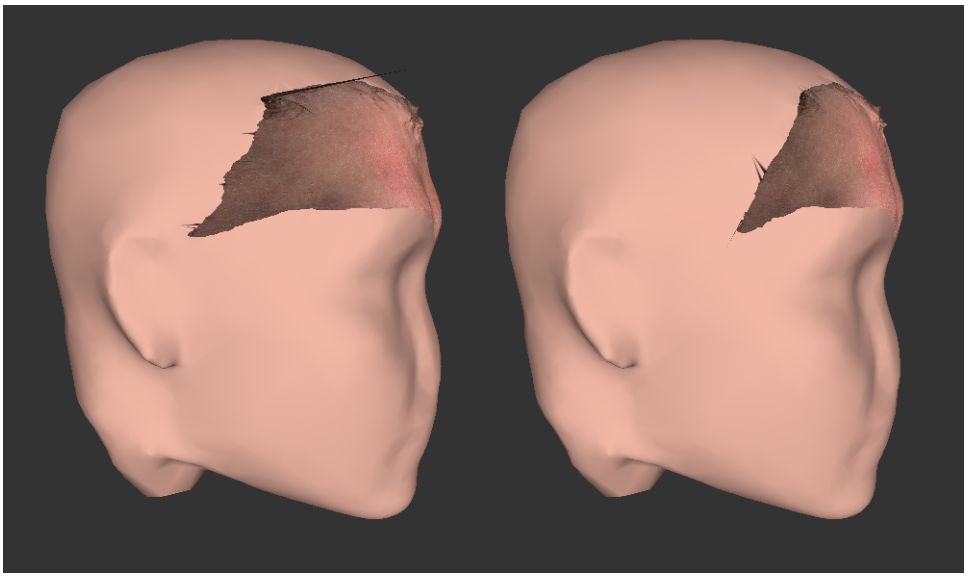
To overcome this obstacle other approach was used. Before projection of source mesh its curvature is determined by the angles of vectors from contour vertices to its center. If high curvature is measured then cylindric projection is used see figure 3.2b.

Parts of the projected surface may lay under or over target mesh therfore vertices of contour are projected in direction of normal and its opposite direction. Face that is have intersection with projection and is closest to the projected vertex is ten used in snake creation.

3.3.2 Creation of snake

The contour created on mesh by projection is represented by geometric snake. Snake is implement as doubly linked list of snaxels. Each

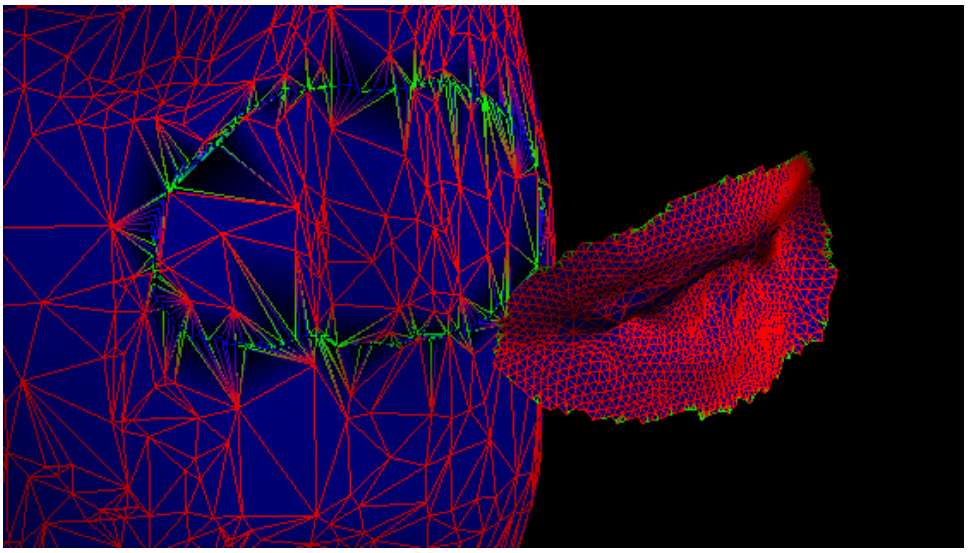
Figure 3.2: Two result of merging based only on different projection method.



(a) Deformed surface with planar projection of facial feature.

(b) Cylindric projection of facial feature.

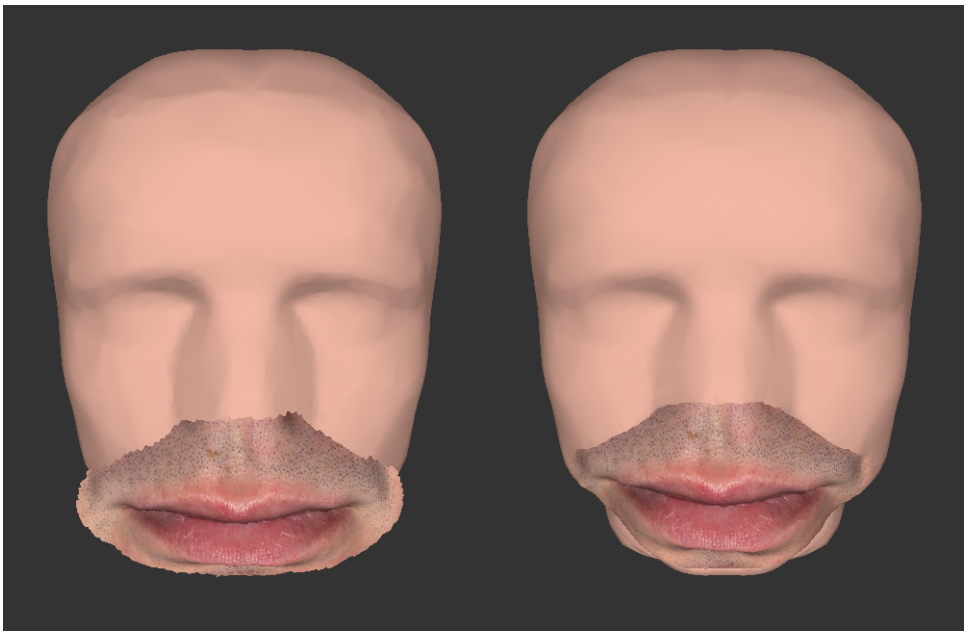
Figure 3.3: Projection of eye contour on head, green edges are outgoing from vertex created on split of mesh and blue are outgoing from vertex created on split of half-edge.



snaxel in snake have its spatial coordinates and reference to DCEL part it was projected on and reference to vertex in source DCEL it was projected from for further mapping. Two neighboring snaxels always match existing outer edge on source DCEL. The vertices form source contour are projected one at the time and for each successful projection new snaxel is inserted into snake. After whole contour was projected on the surface the snake is evolved over surface to obtain the shape that is best suited for the underlying surface see subsection 3.3.5. Last step is integration of sneak into target surface. At the snaxels positions new vertex are created and and the mesh is split accordingly for more details see subsection 3.3.7. The example of inserted can be seen in figure 3.3.

It is possible that the projection contour may not lay on target mesh, because of the way it was positioned in application or it can be just too big to be inserted into target mesh. In these case vertex that could not be projected on target surfaced is removed from source and source vertices that lays on outer edge of source surface are added into the contour. Projection continues on those newly added vertices.

Figure 3.4: Peeling of overreaching source mesh.



(a) Facial part positioned so it is overreaching head.

(b) Result of merging after peeling to match target mesh.

Removal of vertices continue until source mesh cannot be fitted into target mesh. Example of such projection can be seen in figure 3.4.

3.3.3 BVH implementation

To speed up projection of source vertices on target mesh faces its faces are stored in bounding volume hierarchy (BVH). In the implementation of BVH axis aligned bounding boxes(AABB)are used trade off between higher efficiency of BVH construction and test of intersections of bounding box with ray as opposed to higher false acceptance rate of bounding boxes[1].

In the implementation created in this work construct the hierarchy in top down manner. The construction is guided by classes that implement strategy pastern and assure the splitting planes are changed accordingly and right axises is used for sorting of faces. For each created node all the faces are dived into two groups where

each group contains faces that are either completely right of splitting plane or completely left. For each group is then recursively created new node in hierarchy that use splitting plane that is perpendicular to next axis. The plane is position at coordinates that assure group will be split approximately in half. This is done by sorting faces by rightmost position of their vertices on splitting axis. The leaf nodes are created when group of faces consist from only one face.

3.3.4 Triangle intersection

For testing intersection of triangles and rays algorithm described in paper Fast, Minimum Storage Ray/Triangle Intersection. The algorithm does not compute the intersection of ray and triangle directly but is rather using transformation that translate triangle so the origin of the ray lays on the same plane as translated triangle, the triangle is then transformed to unit triangle that lays on yz plane and the direction of ray is aligned with x axis. Then barycentric coordinates u, v of rays origin in transformed triangle are used to determine if its inside of its borders, on the borders or outside of them. This transformation can be described by simple equation see3.5a this equation can be simply rewritten so t, u and v can be directly computed see3.5b. [16].

During the projection of contour lot of faces is tested multiple times, this lead to multiple constructions of object of class that stores informations for intersection calculation. To gather all necessary information vertices of face needs to be received, this included multiple traversal of DCEL topology, and vectors calculations. To mitigate this overhead the object that store data for projection is created first time faces is tested and then reused for further intersection testing with already calculated vectors.

3.3.5 Evolution of snake over surface

The shape of projected contour may not be correct after projection due the deformations caused by curvature of target surface. In some cases the edges of source mesh may be jagged, but the projection should be smooth to better simulate real shapes of human face. When the borders of the projection lays on the natural borders on face they

Figure 3.5: Transformation of triangle to be aligned with ray origin. D is determinant of triangle, V_i its vertices, O is ray origin, t the distance from origin of the ray to the triangle and u, v are barycentric coordinates of intersection.

$$\left[-D, V_1 - V_0, V_2 - V_0\right] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0$$

(a) The equation of transformation.

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}$$

(b) Equation from 3.5a rewritten for direct computation of values v, u, t. $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$, $T = O - V_0$, $P = D \times E_2$ and $Q = T \times E_1$

tends to overlay in two facial features. This would lead to deformation of face and would create unnaturally looking face. To get rid of those artifacts , the snake is evolved over surface mesh, before it is made integral part of it.

For evolution of snake algorithm from 2.2.3 is used. All snaxels are moved to correct position by translation over mesh surface by their displacement from ideal position. Displacement of snaxel is calculated as sum of external and internal displacements on snake 3.6a. Internal displacement 3.6b is based on sum of normalized vectors to neighboring snaxels that are scaled by deviation of their squared distance from mean of global squared distance of neighboring snaxels 3.6c. External gradient is then calculated as gradient of snaxel on snake.

The gradient of snaxels is dependent on the type of mesh part it is laying on. For vertex its normal is used as gradient, if the underlaying part of the surface is not vertex, then the gradient is calculated as weighted sum of gradients of all vertex of underlying surface part.

Figure 3.6: Equation for snaxel displacement calculation.

$$D(s_i) = D_{int}(s_i) + D_{ext}(s_i)$$

(a) Composition of internal and external displacements.

$$D_{int}(s_i) = \sum\nolimits_{J \in N_i} (\overline{d^2} - \|s_i - s_j\|^2) \frac{s_i - s_j}{\|s_i - s_j\|} + \left(\frac{1}{|N_i|} \sum\nolimits_{j \in N_i} s_j - s_i \right)$$

(b) Internal displacement of snaxel.

$$\overline{d^2} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{N_i} \sum\nolimits_{J \in N_i} (\|s_i - s_j\|^2)$$

(c) Global mean square distance.

$$D_{ext}(s_i) = gradient(s_i)$$

(d) External displacement.

3.3.6 Insertion of snake into triangle mesh

After snake is properly aligned we can insert it in target mesh so matching topology for borders of source mesh is created. The points in snake are not the part of target at this point and between each point may lay few faces or edges. More snaxels may also lay inside one face. We need to split underlying mesh of snake, so it contains vertices and edges that copy the position of its snaxels topology respectively topology of outer edges of source DCEL.

Starting snaxel is inserted at the exact position where it was projected. The rest of snaxels needs to check its position on mesh again as the topology is changing during insertion and they may end up laying on newly created parts. It is also necessary to find all intersection with edges that lays between two inserted vertices

The snaxels are processed iteratively in the direction of snake. Vector from current to next snaxel is created and all intersection with edges and vertices are found.

At position of next snaxel is then created matching vertex.From neighboring faces of current snaxel face for projection is found by checking if the vector is projected on it. If the vector intersects with half-edge of the face new vertex is inserted on intersection also new snaxel with matching position is inserted after current one.

If the intersection lays on vertex only new snaxel is created and vertex is mapped as target vertex for this new snaxel. In both cases source mesh is update so it contains vertex for new snaxel on matching source edge. If the projection lays inside of face the actual position of snaxel was found and new vertex is inserted on the end position of projected vector in that face and next snaxel is processed then. After last snaxels is processed target DCEL contains matching vertices and edges for contour of source DCEL and the mappings is stored in snaxels.

3.3.7 Insertion of vertex into DCEL

Only vertices are inserted into mesh and the half-edges are created by splitting of the faces at positions of newly created vertex.

The matching vertex from source DCEL are inserted one at the time and the topology is updated if necessary. There are three pos-

sible cases were inserted vertex may lay. Projection on vertex is the simplest case we only map the vertices without any change in topology.

If the projected vector ends on the half-edge, we need to split the edge at the position of new vertex. The edge is represent by half-edge and its twin both are split in two new connected half-edges that respect original flow in the mesh and replace the original half-edges. Original incident face of both of them removed and is replaced with two new faces separated by edge from new vertex to opposite one so the topology stay consistent see figure 3.7a for projection and original mesh and figure 3.7b for mesh created after insertion of vertex. The normal and texture coordinates of created vertex are linearly approximated from end points of original half-edges.

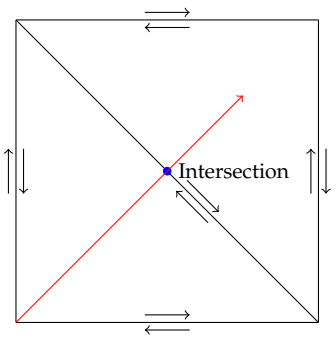
When the snaxel lays on face, the face is replaced with triangle fan where central point is the new vertex and the outer edges are original edges of replaced face for details see figures 3.7c and 3.7d. To approximate texture coordinates and normal vector barycentric coordinations of new vertex in original face are used to interpolation from vertices of half-edge opposite to vertex in new face.

3.4 Hole cutting

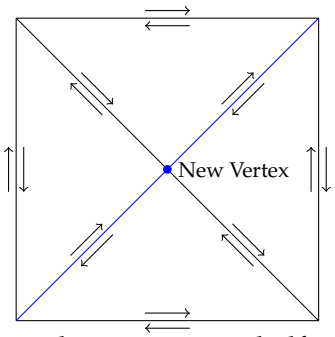
After snake was inserted into target mesh we need to remove all DCEL parts that lay inside of the loop that consists from half-edges that were created by snake insertion.

Using the mapping from source to target vertices and their position in snake collection of half-edges that represent inserted loop is created. Then all their incident faces are marked for removal and added to queue of faces that have to have their neighboring faces checked. The algorithm is dequeuing unchecked faces and for each face it test each incident half-edge. If half-edge is not part of the border, its incident face needs to be removed and the edge needs to be removed as well. If the face was not already marked for removal it is queue for check of its neighboring face. After queue with faces thats needs to be tested is empty all marked face and half-edges are removed if vertex on removed half-edge have no incident half-edge left it is removed as well.The removal of half-edges and faces keeps

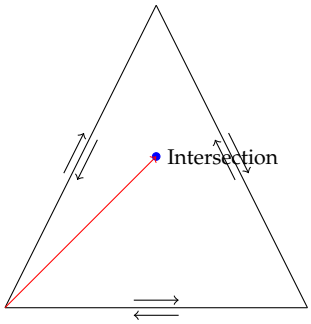
Figure 3.7: Creation of new faces on vertex insertion. Show part of polygonal mesh arrows next to edges represents the orientation of half-edges in DCEL



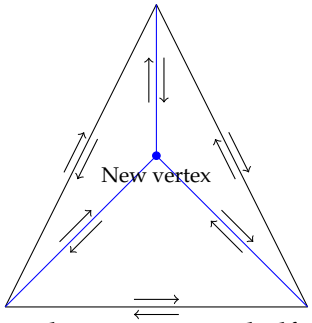
(a) Red is vector projected on mesh.



(b) Blue edges represents half-edges inserted after addition of new vertex.



(c) Red is vector projected on mesh.



(d) Blue edges represents half-edges inserted after addition of new vertex.

DCEL topology correct as the flow on edges is amended and references to removed faces are replaced by outer face for incident half-edges.

3.5 Merging an Sewing Of DCEL

For position alignment and smoothening of transition between source mesh and opening crated in target mesh MVC merging is used. This algorithm create aligned meshes then can be stitched together. Parts of source DCEL needs to be mapped into target DCEL and all informations about vertex properties and used materials needs to be carried over.

3.5.1 Implementation of MVC merging

This algorithm will merge two meshes together on their openings. Always one of the meshes needs to marked as target mesh and the other as source mesh. Source mesh is transformed to match the opening on target and then inserted into it. In this work the head is always handled as target as its shape and details of already projected face features needs to be preserved, changes into head positions would also leads to change of relative position of head and rest of the facial features. It is requires that, openings on both meshes have equal count of vertices and they needs to be paired together [7]. This was achieved in this work by projection of facial feature contour on head, when for each projected vertex pair vertex was created and when the projected contour intersected with any edges or vertices of target mesh pair vertices on both meshes were created.

Let x be a point on manifold mesh and x's its first neighborhood ring is then boundary ∂C, where $\partial C = [v_0, v_1, \ldots, v_m = V_0]$ and vertices of ∂C are in anticlockwise order see figure 3.8. Then we can calculate mean value coordinates using equations from figure 3.9 where α_{i-1} is $\angle v_i x v_{i-1}$ and α_i is $\angle v_{i+1} x v_i$ [7].

To align source mesh with target first is opening on source mesh used as boundary and mean value coordinates are calculated for each vertex of source in this boundary. Then second boundary is created with points that have coordinates equals to difference between open-

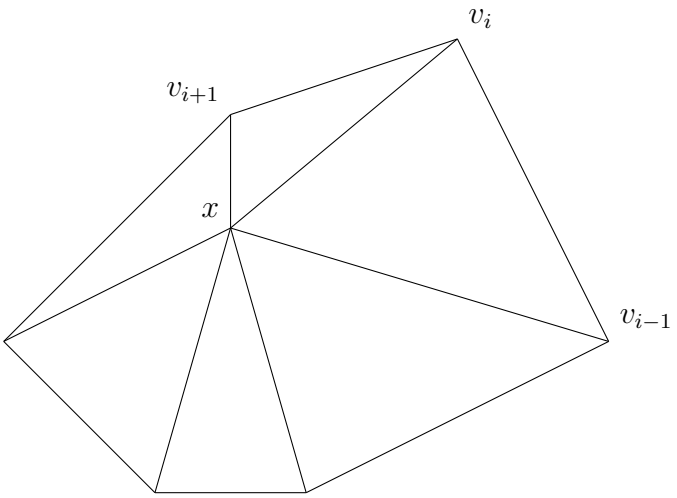


Figure 3.8: First neighbor ring of x

ings of pair vetices of two merged surfaces. This boundary is then used to interpolate position of source vertices in target mesh by applying mean value coordinates of source vertices on this boundary 3.9c.

Calculation of mean value coordinates require high number of operations and take the biggest part of time need in implemented algorithm. The complexity of algorithm depends on number of the vertices in mesh v and number of the vertices on the boundary b for each combination of vertex from mesh and vertex from boundary their distance and two angle with neighboring edges needs to be computed. This mean complexity $O(vb)$ and v having order of magnitude of thousandths and b in hundredths what leads to millionths of operations. First naive implementation of equations from 3.9 lead to even higher complexity $O(v^2b)$ as calculation of $\lambda_i(x)$ for vertex v_i require recalculation of weights for each vertex of mesh. This is avoided by calculation of each weight $W_i(x)$ separately before calculation of λ. The weights are stored in array that is indexed by the order of vertex in boundary. To further improve performance the calculation of weight $w_i(x)$ was improved, by enhancing computation of tangent of angel. When determining w_i value of $\alpha_{i-1}/2)$ was already computed for w_{i-1} and $tan(\alpha_i/2$ will be used for calculation in calculation of W_{i+1}. Tangents for each vertex are precomputed in

Figure 3.9: Condition for mean value coordinates.

$$\lambda_i(x) = \frac{w_i(x)}{\sum_{j=1}^{m} w_j(x)}$$

(a) Mean value coordinate.

$$w_i(x) = \frac{tan(\alpha_{i-1}/2) + tan(\alpha_i/2)}{\|v_i - x\|}$$

(b) Weight i.

$$\tilde{f}(x) = \sum_{i=1}^{m} \lambda_i(x) f(v_i)$$

(c) Mean value coordinates applied on boundary.

similar ways as λ values. Even with this two enhancements the alignment of meshes was significantly longer then the rest of algorithms for merging. The main reason was slow computation of tangent. First implementation was calculating dot product of vectors and then using Java build-in methods `Math.arccos` and `Math.tan`. The main problem was caused by Java build-in methods so the calculation of tangent by trigonometric functions was replaced by direct calculation of tangent from ratio of cross product and dot product.

3.5.2 Mapping between two DCEL

With matching opening on target surface and aligned positions of source mesh vertices, meshes can be easily sewed together. Internal vertices of source DCEL are copied into target DCEL and the mapping between them is added to mapping of contour. Then each source face matching vertices are found in mapping and half-edges are created if they already doesn't exist in target and are assigned to new face. When the topology of added DCEL and position of its vertices is merged we need also transfer materials and textures coordinates for face of original mesh into.

3.6 Integration with FIDENTIS

Main reason behind creation of module for mesh merging was its integration into FIDENTIS were it will be used for construction of face model from phantom head model and facial features with applied textures.

The application already have functionality that enable user to create composition from prepared 3D models. After composition is create user should be able to start merging by simply pressing button that is position above face composition, user is able to prepare multiple facial features of different type they will merged into head in the same order as they were added into composite. During the merging user will be informed about progress of process by progress bar that will show short description about ongoing process, number of merged part and total number of part also percentage that is approximating portion of completed work is displayed.

Module created in this work contains its own classes that holds data about graphical mesh so the first step in integration was creation of functionality that will convert data from existing classes in FIDENTIS into format that can be used by doubly connected edge list (DCEL) that is required by algorithms used for mesh merging. The format of data in FIDENTIS is very similar to .obj file so same approach as loading this type of file was used. The module already contained functionality for importing .obj into DCEL so the same code was reused. After the merging algorithm create final polygonal mesh it needs to be converted to `Model` class that is used by application, so it can be display or exported into .obj file format. For this purpose new empty `Model` is created and all materials contained in merged model are loaded into it after this, `Model` is populated by vertices, normals, texture coordinates and topological information in same process as if they were loaded from file.

It was required to provide information about progress of merging. For this purpose in application class `ProgressHandle` is used. The class is part of `org.netbeans.api.progress` package. It allows to display progress of process based on total units of work and already processed units of work that are displayed in application as percentage of finish work it also support displaying of messages about current state of process[5]. To avoid tight coupling of

created module with NetBeans API, Observer pattern was used, internal class that holds information about current state of merging process can be observed by object of type that implements interface `ProgressObserver` this interface is part of created module. In FIDENTIS application is then created anonymous class that implement this interface and decorates `ProgressHandle` so it can be updated from inside of the module.

4 Results

In this section final module is presented, renders of polygonal meshes created in FIDENTS using this module are presented and performance results of implemented algorithms measured on meshes with high and low polygon count are arranged in tables.

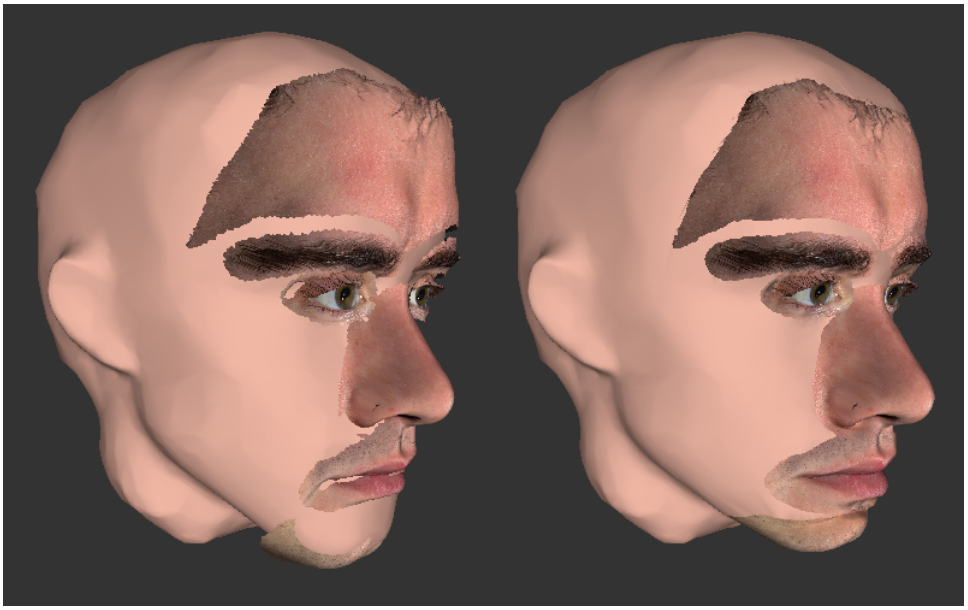
4.1 Description of Module

Module crated in this thesis was created for purpose of merging of two manifold polygonal models with applied textures into one model that would preserve geometry details of merged models and will transfer applied textures in mind. The module was implemented with integration into FIDENTIS in mind. Beside initial requirements additional functional was created that enable separate use of module in any Java application that supports JOGL framework. The module can be used for import of models from obj files and vice versa. Rendering objects using VBO functionality of OpenGl and simple work with scene in addition it provides interface for work with mesh topology using implementation of doubly connected edge list structure. Most of additional functionality was used in early stages of development and provided good tool for visualization of implemented algorithms that supported easy debugging and testing of module.

4.2 Visual result

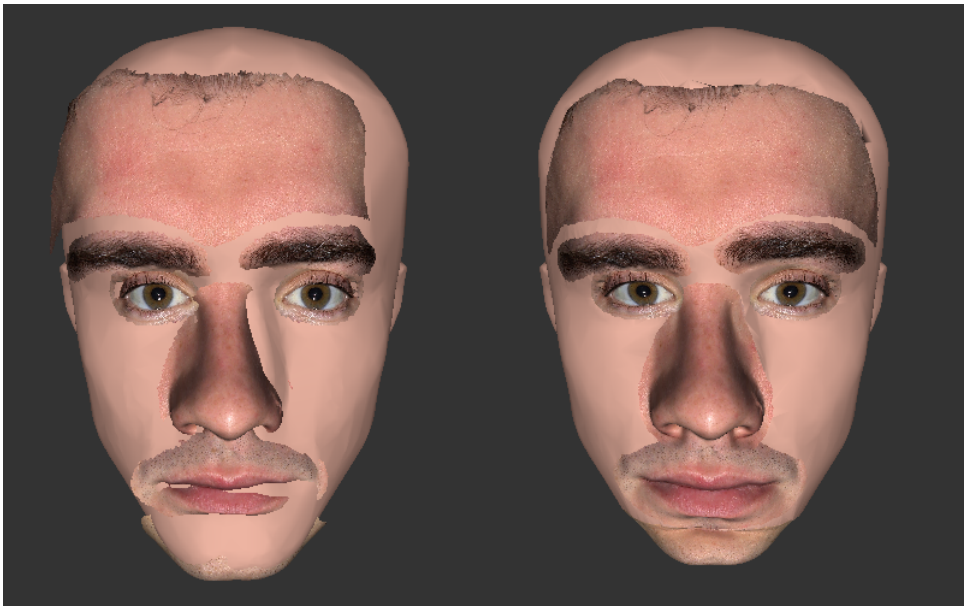
In this sections images rendered in FIDDENTIS that illustrate input and output of implemented algorithms are presented. In the figures 4.1a and 4.1c are facial features arranged in composition before merging. The features are positioned closely to head surface to so it was easier to predict resulting model. In the figures 4.1b and 4.1d are then models created by merging algorithm. Figure 4.2 shows details of connections between head and merged facial parts and the models are displayed as plain polygonal mesh without smoothing by normals and textures are turned off to shows models geometry.

Figure 4.1: Example of input and out put of merging



(a) Facial features prepared for merging side view.

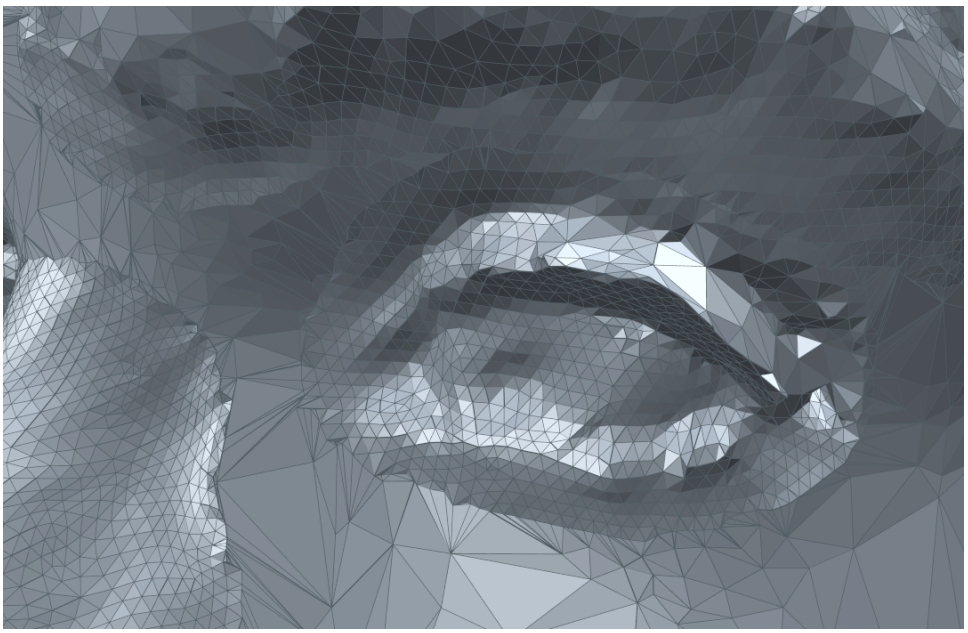
(b) Merged model that contains all facial features side view.



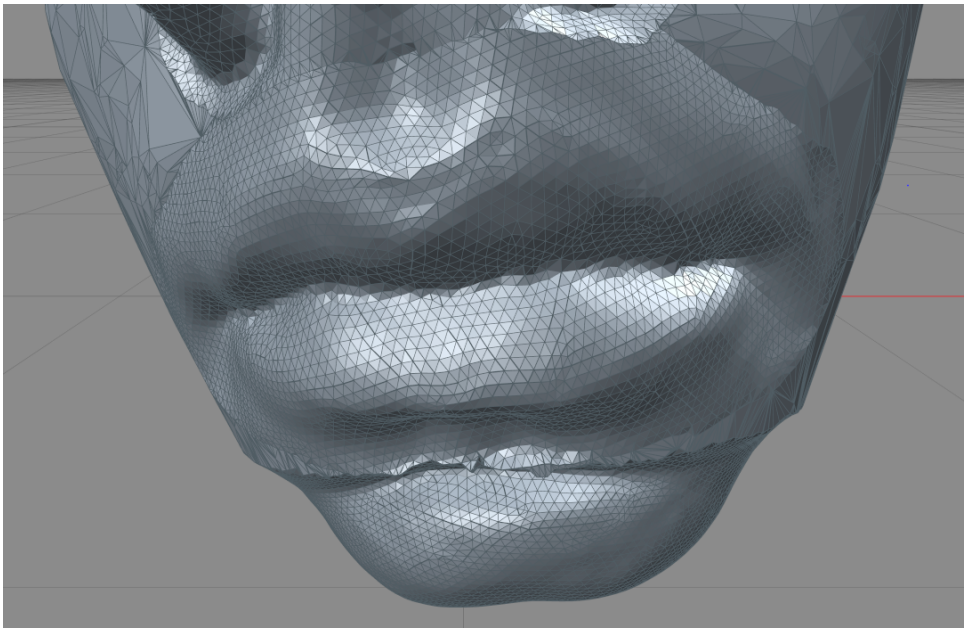
(c) Facial features prepared for merging front view.

(d) Merged model that contains all facial features front view.

Figure 4.2: Details of merge polygonal mesh without smoothing and textures.



(a) Region where eye feature was merged with eyebrow.



(b) Region where nose, mouth and chin features were overlapping and sequentially merged into head.

4.3 Performance measurements

For measurement of performance of algorithms computer with operation system based on linux kernel version 3.16.7-7-desktop was used. The hardware parameters of used computers were: CPU i5-3570K CPU at frequency of 3.40GHz with four cores, 8GB RAM and graphic card GeForce GTX 660 Ti with 2GB of RAM. The module was tested as it was integrated in application FIDENTIS. The measurements only contains processes that are part of actual merging, communication between module and application was not included.

In first table 4.1 are results for models with low polygonal count containing thousand triangle faces. This models were created by reduction of existing models from FIDENTIS. Second table 4.2 then shows results for original models with polygon count varied between seventeen hundred and seventeen thousand. Second important parameter from complexity point of view is count of vertices on the contour of projected model, the values varied between hundred and five hundred.

The process of merging was divided into six parts for the purpose of merging. Projection time consists of time spent on creation of BVH for target mesh and the projection of contour of source mesh and construction of snake. Evolution of Snake is time spent for calculation of proper position of snake on mesh. Insertion of Snake includes all processes required for creation of half-edges in doubly connected edge that correlate with snake and insertion of new vertex into source mesh so the cont of vertex between openings would be matching. Creation of Opening is time spent by identification of faces that needs to be removed from target mesh and its removal. Alignment by MVC corresponds is time spent with calculation of mean value coordinates and their use for alignment of source mesh. The last column represent time spent on transporting of data from source mesh data structure into target mesh data structure and their connection into one doubly connection edge list.

Measured results are scaling according expectation based on complexity of selected algorithms highest increase of computation time was measured for alignment of meshes using MVC, the results corresponds with its complexity $O(vc)$ where v is count of vertices in source mesh and c is count of vertices on its contour. The most de-

manding demanding part of process was direct projections of contour on target mesh. This was caused mostly by the used BVH structure and should be enhanced by replacement of used structure with other that would be more suitable for requirements of this work.

Table 4.1: Results for models with low polygon count.

Part Name	Polygons in Mesh	vertices on Contour	Projection Time [ms]	Evolution of Snake [ms]	Insertion of Snake [ms]	Creation of Opening [ms]	Alignment by MVC [ms]	Stitching of Meshes [ms]	Total Time [ms]
Head	1000	-	-	-	-	-	-	-	**-**
Forehead	1000	160	50	10	29	2	40	22	**153**
Eyebrow Left	1000	122	154	2	29	4	41	16	**246**
Eyebrow Right	1000	131	175	9	12	2	18	23	**239**
Eye Left	1000	97	148	2	11	1	9	17	**188**
Eye Right	1000	100	214	1	9	1	11	21	**257**
Nose	1000	108	176	2	7	2	11	21	**219**
Mouth	1000	104	61	1	5	0	7	17	**91**
Chin	1000	79	193	1	4	1	5	15	**219**
Total	**-**	**-**	**1171**	**28**	**106**	**13**	**142**	**152**	**1612**

Table 4.2: Results for models with high polygon count.

Part Name	Polygons in Mesh	vertices on Contour	Projection Time [ms]	Evolution of Snake [ms]	Insertion of Snake [ms]	Creation of Opening [ms]	Alignment by MVC [ms]	Stitching of Meshes [ms]	Total Time [ms]
Head	9620	-	-	-	-	-	-	-	**-**
Forehead	16960	504	380	53	60	8	1336	154	**1991**
Eyebrow Left	2326	180	1803	4	16	1	38	34	**1896**
Eyebrow Right	2161	181	2267	2	9	3	26	46	**2353**
Eye Left	1750	108	1127	1	6	3	16	26	**1179**
Eye Right	1750	108	1245	1	5	1	17	29	**1298**
Nose	8059	226	837	3	8	3	552	59	**1462**
Mouth	6727	227	676	3	8	5	149	57	**898**
Chin	5761	167	975	2	7	2	134	65	**1185**
Total	**-**	**-**	**9310**	**69**	**119**	**26**	**2268**	**470**	**1612**

5 Conclusion and Future Work

The aim of this thesis was to study several algorithms related to merging of meshes compare them and then select those that would be most suitable for use in merging of facial features into model of head. Implementation of those algorithms should have been put into Java module and then integrated with application FIDENTIS.

Description of studied approaches can be found in chapter 2 Related Work their comparison and implementation was described in chapter 3 Implentation as well as their integration with FIDENTIS. Chosen algorithms were implemented and their use leads to the creation of merged models of head and facial features as can be seen in Figure 4.1. Resulting models have smooth transitions between merged parts and features merged into head models respects its original shape without significant deformations of facial parts.

Performance of implementation is satisfying, with exception of direct contour projection that would require amendments to gain better performance for this purpose change of hierarchical structure used for searching of intersections between rays and triangles is suggested. Rest of the algorithms provide high performance and scale well with increased complexity of input data. Details of measurements can be seen in sectio 4.3.

To improve user experience and ease of use I would suggest integration of rendering functionality created in the module with FIDENTIS this would enable displaying of projected contours before merging process. This should give better approximation for final positions of merged features and help avoid multiple merging attempts in process generation of desired result. Such integration would also enable partial results to be shown during the process of merging so user can get stop it if the results distinguish from expectations.

Second suggested improvement is generation of blended textures on the borders of merged features and in theirs close proximity. Similar process could be applied for generation of texture for small parts of heads surface that is positioned between multiple features. These amendments should provide more appealing results. All necessary data for this process are already prepared in resulting model.

Bibliography

[1] Kasper Amstrup Andersen and Christian Bay. A survey of algorithms for construction of optimal heterogeneous bounding volume hierarchies. *Saatavilla pdf-muodossa http://image. diku. dk/projects/media/christian. bay. kasper. andersen. B*, 6, 2006.

[2] Mario Botsch, Stephan Steinberg, Stephan Bischoff, and Leif Kobbelt. Openmesh-a generic and efficient polygon mesh data structure. 2002.

[3] Vicki Bruce and Andrew W Young. *Face perception*. Psychology Press, 2012.

[4] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. Directed edges—a scalable representation for triangle meshes. *Journal of Graphics tools*, 3(4):1–11, 1998.

[5] ORACLE CORPORATION. Progress api, May 2015.

[6] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.

[7] Zhengjie Deng, Guoyuan Chen, Fengwei Wang, and Fan Zhou. Mesh merging with mean value coordinates. In *Digital Home (ICDH), 2012 Fourth International Conference on*, pages 278–282. IEEE, 2012.

[8] Michael S Floater. Mean value coordinates. *Computer aided geometric design*, 20(1):19–27, 2003.

[9] Charlie D Frowd. Varieties of biometric facial techniques for detecting offenders. *Procedia Computer Science*, 2:3–10, 2010.

[10] Xiaohuang Huang, Hongbo Fu, Oscar Kin-Chung Au, and Chiew-Lan Tai. Optimal boundaries for poisson mesh merging. pages 35–40, 2007.

[11] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy mesh cutting. 25(3):283–291, 2006.

[12] Xiaogang Jin, Juncong Lin, Charlie CL Wang, Jieqing Feng, and Hanqiu Sun. Mesh fusion using functional blending on topologically incompatible sections. *The Visual Computer*, 22(4):266–275, 2006.

[13] Zuzana Kotulanová, Igor Chalás, Petra Urbanová, et al. 3d virtual model database of human faces: Applications in anthropology and forensic sciences. In *Proceedings of the Mikulov Anthropology Meeting*, volume 20, 2014.

[14] Kenneth R Laughery and Richard H Fowler. Sketch artist and identi-kit procedures for recalling faces. *Journal of Applied Psychology*, 65(3):307, 1980.

[15] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design*, 22(5):444–465, 2005.

[16] Tomas Möller and Ben Trumbore. Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*, page 7. ACM, 2005.

[17] Robert F Tobler and Stefan Maierhofer. A mesh data structure for rendering and subdivision. 2006.

[18] Fan Yang, Frederick WB Li, and Rynson WH Lau. Active contour projection for mesh segmentation. In *Pervasive Computing (JCPC), 2009 Joint Conferences on*, pages 865–874. IEEE, 2009.

[19] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. 23(3):644–651, 2004.

A Appendix

Whole FIDENTIS application was uploaded into Archive of Thesis in IS of Masaryk's University not only module created in this thesis that is contained in package `cz.fidinetis.merging` and was integrated as part of application. This was necessary as application provide GUI for created module as well as models of heads and facial features. Author of this thesis is only author of code that is contained in package `cz.fidinetis.merging` and its sub-packages rest of the application is work of members of team working on FIDENTIS application and is not part of work created in tis thesis.