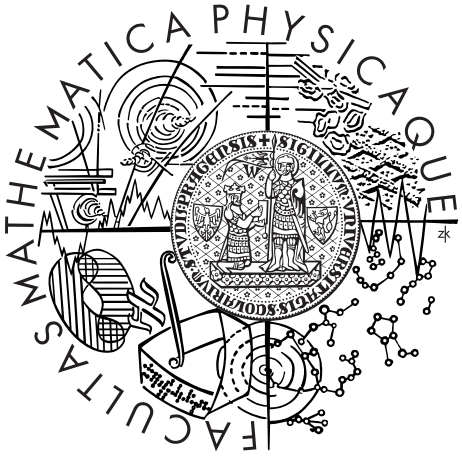
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Masařík

Mesh mending

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2011

Here I would like to thank to my supervisor RNDr. Josef Pelikán for great supervising, RNDr. Jana Velemínská and their PhD students for answered questions about anthropology, Morphome3cs team for helping me to be well informed about whole project.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Mesh mending

Autor: Tomáš Masařík

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán, Kabinet software a výuky informatiky

Abstrakt: V práce se věnuji opravám trojúhelníkové sítě a to jak z hlediska topologie tak i geometrie. Navazuji na Ročníkový projekt a Softwarovou praxi, v rámci níž byl vytvořen program zabývající se detekcí chyb a jejich opravami v trojúhelníkových sítích. Hlavním důvodem vzniku takového programu byla práce se skeny obličejů na katedře antropologie, PŘF UK.
V textu je nejprve zanalizována problematika a jsou zhodnocena již dostupná řešení. Později popisuji implementované algoritmy, následované uživatelskou a programátorskou dokumentací. Závěrem demonstruji výsledky práce programu a navrhuji jeho případná další rozšíření.

Klíčová slova: Trojúhelník, síť, opravování.

Title: Mesh mending

Author: Tomáš Masařík

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Josef Pelikán, Department of Software and Computer Science Education

Abstract: The bachelor work is focused on triangle meshes mending. It includes topological and also geometrical aspects. I continue working on Individual Software Project and Software Praxis. Before I have created program capable of detecting and repairing some of triangle mesh errors. Main goal of creation such a program was a research using scans of human faces at Faculty of Anthropology of Nature Science at Charles University in Prague.
There are analyzed issues and there are evaluated existing solution in this paper. In second section I describe implemented algorithms, user's and programmer's documentation as well. The results are demonstrated At the end and there are proposed some future extensions.

Keywords: Triangle, Mesh, Mending.

Contents

Introduction

This work started almost a year ago when I signed the Individual Software Project with RNDr. Josef Pelikán. He cooperates with Department of Anthropology and Human Genetics at PŘF. The base of this partnership is project called Morphome3cs developed by Computer Graphics Group at MFF. He thought that Morpohome3cs should contain some mesh mending algorithms too. So, we immediately started to find mesh defects and we were studying what could have been improved in this field.

Following year I continued working on this project and first lines of source code has been developed. Results and the process of forming the project I'm now summarizing into this bachelor thesis.

0.1 The most useful terms.

First of all, before we discuss the basics of mesh mending, we try to delve into the most important terminology. We discuss all necessities without we can't proceed any further.

0.1.1 Mesh

Polygonal mesh is data structure that contains faces made from appropriate number of vertices according to degree of polygon. They store information about edges in some implementation too but it's not necessary.

All variants store various secondary details in faces or vertices. They are useful for concrete application or usually for better viewing. The most used and important are:

Table 1: Secondary mesh data

- *3 dimensions coordinates* — Necessary for determining an Euclidean space position. This is the only mandatory item.
- *color* — Good replacement of texture.
- *normals* — Used for shading, ray–tracing or other important algorithms.
- *texture coordinates* — Coordinates for mapping texture on model.
- *other* — Whatever anyone wanted.

Triangle mesh (trimesh)

I'll concentrate only on triangle meshes in this paper. Not only for making the problem easier but also for their quick implementation. Almost all modern graphic chips have, an enhancement for working with trimeshes, implemented in its core. This makes using them very clever.

0.1.2 Triangle mesh errors

All defects we could find out I'm going to explain in details in the following part. Now, I'd like to show you some preview.

Of course, none of following errors must be really wrong. For example, you could easily imagine a scan of a female screw containing a hole too but it's the correct one. So, somebody must take responsibility and determine which attribute concrete mesh should have and which not. Then he simply types paremetre to the mesh mending program and it takes care of the rest.

There are two basic categories of mesh errors between which I could split all of them. Topological and Geometric defects.

Topological defects

This category contains errors resulting from mutual position of triangles. It includes non-2-manifold faces, gaps, holes, wrong orientation of normal vectors, unconnected components etc.

Geometric defects

In contrast, geometric errors result from concrete position itselves. That contains defects like: Volume of triangle or crossing faces. It could be also position of more triangles like: Face crossing, sharp angles etc.

How do errors begin?

This question should ask everybody but answer isn't only one. Sources of incorrect meshes is scanners in our project, specially their software. It contains some bugs but also it try to preserve as much information as possible. Unfortunately this information is contaminated by inaccuracy, so it cannot be trusted and used.

Other incorrect meshes may come from hands of computer graphic designer or pattern maker in CAD environment. Even though he make the best he easily could produce some undiscoverable error but later this error could inflict considerable damage.

0.1.3 2–manifoldness

Because manifoldness is one of the basic terms in topology we need to understand its definition. Formally, a topological manifold is a set of points that is locally homeomorphic to the Euclidean space. Locally homeomorphic to the Euclidean space means that every point has a neighborhood homeomorphic to an open Euclidean n-ball. [wMa] . Homeomorphism is topological isomorphism, so bijective, continuous function which has also continuous inverse function [wHo] .

We will use only 2–manifoldness in this work. It is sometimes called surface. In case of trimesh that practically means: "No edge has more than two neighbours".

0.2 Scanning and scanners

Nowadays, 3D scan processing is a very good business and many companies are interested in.

0.2.1 What is the main area which 3D scans are used in?

The answer is simple. After 3D model is created, it's used to automatically reproduce the original thing. This process is called reverse engineering or simply re-engineering.

In this very difficult development is interested many professional tools and programs. One of the most famous is Rapid Form, described bellow. It goes so far, that it focuses precisely on all what is needed to produce almost identical thing. Everything started by object scanning, following by mesh editing, ended by serial production of that objects.

This sort of utilities is also useful in other sectors of human doing. For example, we develop specialized software for anthropological research. It isn't only a cheap substitution of commercial software but full–featured specialized program.

0.2.2 The use of 3D scans at Department of Anthropology.

Faculty of Natural Science (FNS) Department of Anthropology and Human Genetics provides several researches requiring correct meshes originate from scanners.

The first one I have met is focused on sex diversities of human faces. It similarly compares differences between healthy people's face and face of people that have passed lip cleft operation. As a result they will be able to determine whether the difference between them is significant and how this variety, if any, changes throughout years.

Collected and corrected data are statistically computed and average meshes are created for each data files (healthy, ill). Then average meshes are compared and abnormalities are measured.

The second one is dental reasearch comparing again ill and healthy people and how this affect their teeth shape.

All these studies are made in 3D laboratory. There are desirable software and hardware equipment. Students and teachers analyze biological objects and provide morphological comparation, study variabilities, weights, lengths, shapes, norms and pathologies. They try to create digital models because they preserve in the same form almost forever. It's useful if anyone would like to continue that research or simply verify results. (3Dl)

0.2.3 How do scanners work?

I'd like to demonstrate two different 3D scanners used by department of anthropology at faculty of Nature. I'm using mainly their output for create and test mesh mending algorithms. It's interesting because they show two completely different ways of scanning. Sadly, neither first, nor second way produces correct meshes.

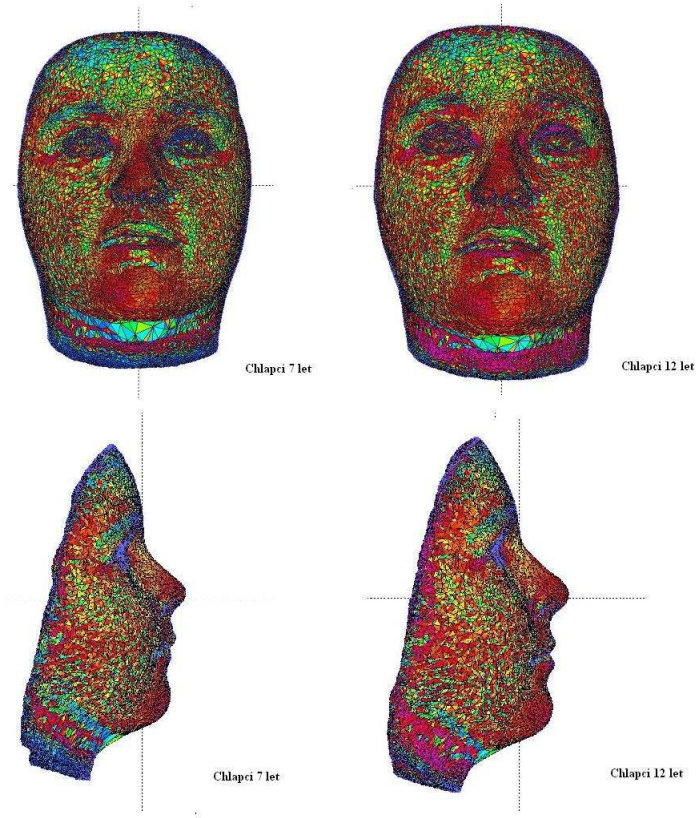


Figure 1: An example of average mesh of boys between 6 and 12 years.

Face scanner Vectra 3D

This optic scanner is not only developed for application in science but also in practical medicine. It uses combination of two sights (like our eyes do) using two b&w digital cameras to determinate 3D dense point cloud. This method is called photometry. After that it triangulates points into triangle mesh using its own proprietary software. There are many fault are made in this part. It has another color digital camera to get color texture. [vec]

PICZA 3D laser scanner LPX–250

Non contact laser scanner designed for scanning small objects without any risk of damage and with highest quality of scans. It generates detailed, high resolution 3D models using rotary and plane scanning modes. The scans are then corrected by LPX EZ studio reverse engineering software. [Pic]

0.3 Project Morphometri3cs (Morpho)

The program part of mesh mending is implemented into Morphome3cs framework, including GUI configuration, results showing 3D viewer and saving utility.

Morhphome3cs is faculty project that have been developed for several years. Two Software project teams have participated on it, lectors created large piece of it and some students with their bachelor work too. [mor a]

Figure 2: Scanner Vectra 3D

Figure 3: Scanner PICZA LPX–250

Projects goals are(were) to create an independent, solid, open source system that could be used for anthropological, archeological or other research. It should support many possibilities of 2D and 3D visualization.

Morpho has two types of users. One is a student, a scientist at biology. The platform should be as easy as possible for him without deep mathematical/programming knowledge. Second is a student or an absolvent of MFF who should be able to create application and implement correct statistical methods into Morpho.

I'm using for mesh mending only one small part of Morpho project, The 3D editor. It is used for viewing results of mesh mending or only for highlighting suspect faces before correcting them.

System contains many elements like these: [mor b] [mor c] [mor 2011]

Table 2: Morpho elements

- *3D models editor* — Viewing 3D models, such as triangle meshes. My project is implanted into this part of Morpho.
- *2D editor* — Editing 2D models like photos.
- *Volume editor* — Viewing/editing volume models.
- *Landmark editor* — Implanting landmarks on 3D models.
- *Statistical instruments* — Morpho provides wrapper on R, well known program for statistic computation. It used R for provides statistical methods. That is used many times in anthropological experiments.
- *Python script* — It supports Python as scripting language for smooth development of application for Morpho.
- *Specimen editor* — Used as well–arranged database of examined exemplars. From this part of Morpho the majority of work starts here. User could run workflow and their collection of specimen.

0.4 Other software appearing in the area.

This section should be small excursion into software

0.4.1 Rapidform

Rapidform is one of the most professional tools in reverse engineering area for many years. But unfortunatelly, it is focused on something else than science. So, its methods aren't described and documented very well. At least, it isn't written anywhere how defects are corrected and how good results of mending really are. From my own experience, not every mistake is corrected at all. Second con is that software is focused on reverse engineering complex and large area of services to be maintened and actualized. Nobody has enough time to take care on mesh mending algorithm, as they need. Moreover, the mesh cleaning is only an "edge" of re-ingeneer work. On the other hand, morpho takes a big step when they upgrade on version XOS. They split their sofware to focus even harder. There are better mending function and better GUI for easier batch mode mesh defects detection. [rap] [rap 2006] [rap 2004]

0.4.2 Rhinoceros 3D

According to its own online manual. It provides similar functionality as Rapid. It could find and repair folowing things: zero volume faces, non–2–manifold edges, holes, duplicate faces, wrong orientation of normal vectors, unconnected parts and unused vertices. [Rhi]

0.4.3 others

Surely, there are other programs in this area but none of them is as important as previously mentioned. For all I remark Blender 3D.

0.5 What is written in literature?

My lector have advice me two following paper. One is master thesis and the second bachelor thesis on close theme, mesh mending algorithm. Both had been written on Faculty of Electrical Engineering, Czech Technic High School. Each was interested mainly in technically created meshes, so mistakes created by model constructors while they were working in CAD–like enviroment. Nevertheles, many of their observation are usefull in general. Of course, there are some errors that are special cases like cracks and T-joints. Others are irelevant for our modelslike volume of solid, because we have only surface.

First was focused on mathematical mathematical proof of corectness of the mesh, or analzzes what kind of error that mesh is affected. [Veleba 2008]

Second used results of first and try to determine correct order of repairing defects of the mesh. It analize what algorithm could create which error and which error covers another errors. In general, that is a study about correspondance between error types. So, I try to get from this work the order in which I'll realize my correcting algorithms. [Marek 2010]

0.6 Aims

After long and hard examination and testing of mesh defects on real scans I found out that some of defects arent in data. For example, non–2–manifold faces aren't in scans at all because scanner's software don't dare to create such error. So for me, it isn't necessary to implement algorithms to clean them.

Also I try to adjust as close as possible to needs of mesh mending software users. Now, they use only a tool for edit boundaries by hand, which I could replace by algorithm for removing unconnected components. However, the most important part is to fill holes. It's logic because holes destroys statistically created mesh, so I try to focus well on filling holes algorithms. Moreover, filling holes algorithm they've used until now didn't work with textures. Thus, after a hole is filled there is one–colored area and fault could destribute itselves into average mesh. Their last step is to decimate mesh and therefore abnormalities and faults are mostly clear. I would not like to replace it, but preceded it by cleaning tools like: Cleaning crossing faces or deleting spikes etc.

Of course, I'd like to create user freinly Graphical User Interface (GUI), which will be integrated into Morpho 3D viewer. This could not only clean or fix abnormalities but also mark them with colors for better understanding how the program works.

All these, I'd like to summarize in this paper and acompagne it with short test and comparison based on cleaning real scans. I'll use my mesh mending software and also rapidform clening wizard which is now in use in 3D labortory at department of Anthropology.

0.7 Thesis structure

In first part I'd like to review all errors I was able to find out anywhere. Second part contains only used methods and describes their implementation and also some interesting algorithms. Third part is focused on users and their Graphical User's interface and also some Morpho description, following programers documentation for better and easier profesional understanding in fourth part. And at the end I provide you some result of my program.

1. Analyzes

In this chapter I would like to speak about all defects the triangle mesh would contain I was able to find anywhere. I divide them into two categories which I have already spoken in introduction about. Many faults are from literature, others are from my examining of real meshes.

Some error types I accompany with suitable image and each with short description of their origin.

At the end I try to mark out why some corrections are implemented and some others aren't. I suggest how they are reflecting analyzed data and needs of users.

1.1 Topological errors

In this section are listed topological defects. By a topological defect I mean that the defect is created only by interaction of faces, regardless their position in space, only it depends on neighbourhood´s faces.

1.1.1 Unconnected vertices

This problem is simple. Mesh sometimes contains vertices which are not connected with any face. Those vertices aren't viewed nor used anymore. They only occupy space. Their only purpose is in mesh creation process when they are connected at the end by constructed triangle. If they weren't, they are useless.

How and why this error is created? It's harder to delete vertices, because of they are usually stored in array, which must be reindexed when vertices are removed. And since when the amount of the stored information in memory isn't so small, it isn't any problem to keep them. So in many algorithms is easier and more effective to only delete references to them but the overall structure is greater with time. Thus if we try to correct mesh we can't forget on this feature.

Unconnected vertices should be deleted at the end of mesh mending process. According to work of [Marek 2010] this wouldn't affect anything.

1.1.2 Non–2–manifold faces

As we shows the definition in introduction, non-2-manifold faces are those with at least 2 neighbours using the same edge.

There exist many ways and well–experienced algorithms solving this problem. Ones only delete those feces and simply fill a hole. Others somehow try to determine what face is wrong and then delete only that face.

Fortunately, we will be no longer interested in manifoldnes because our data didn't contain any, in this context, wrong face.

1.1.3 Unconnected clusters

Simply Unconnected components, they cannot be connected only by shared vertex.

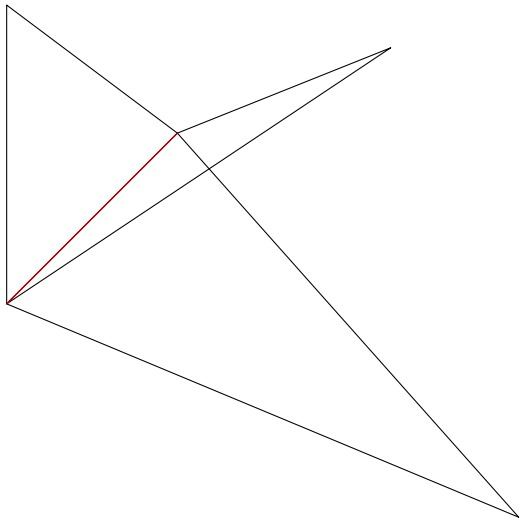


Figure 1.1: An example of non–manifold faces with marked problematic edge.

Sometimes solution is to fill space between components. This works great in case of artificial meshes. In real scans, it could be unsolvable. So, simply it could be fixed by removing small clusters and hopes that there remains only one big cluster. It could be also done by keeping components according to their size(number of faces).

Removing unconnected clusters is one of the most important part of my mesh mending implementation. It's based on need of users.

1.1.4 Holes

Another important problem are holes. There are many types of holes. Some even shouldn't be incorrect. But on real scan data are many holes produced by scanning software.

There should be implemented basic filling hole in my software. Basic means without adding new vertices. It's important to implement this because of rapidform's currently used implementation destroys textures and normals while filling holes. [rap 2006]

1.1.5 Duplicit faces

This error is simple. There are two or more faces with the same coordinates of each vertex in a mesh.

However this mistake looks weird, the creation process sometimes comes up in mesh such errors as human mistake. Especially when meshes are created in CAD environment. Also this error sometimes comes up in scanned meshes as results part proves.

The solution is simple: Just keep unique faces. Maybe, when they differ, interpolate texture coordinates and normal vectors.

1.1.6 Wrong direction of a normal vector

Normal orientation is very important in viewing mesh model. Especially, in ray–tracing normals is used for determining the direction of a ray.

The solution is simple. We go through all components and count the orientation. Than we invert reversed normals.

However this looks bad, the primary function of scanned meshes isn't to be viewed but for further research. This correction will not be implemented at this time. Image is copied from [Marek 2010] .

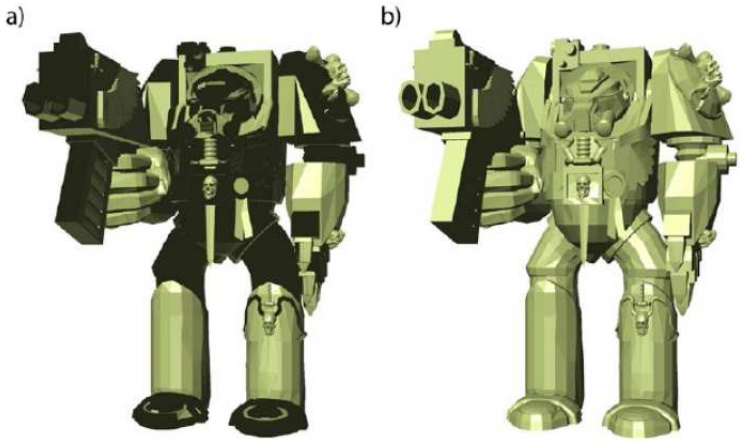


Figure 1.2: An example how wrong orientation of normal could damage an image.

1.2 Geometrical error

Beside topological defects they contain some mistake that comes up from vertex coordinates position or their interaction.

1.2.1 Zero volume of Face

Some triangle has two or even three vertices with same coordinates. That mostly means it can't be seen and used.

That face is useless so it should be removed. The mesh decimation also solves this problem.

Despite it's not frequent in scanned meshes, I decide to implement this function in my mesh mending tool.

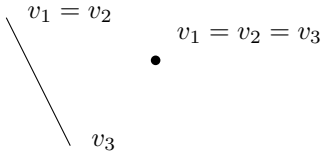


Figure 1.3: Zero volume face example.

1.2.2 Zero volume of solid

Zero volume of solid is similar to zero volume of face. It's solid with 3 dimensional volume equal zero.

Solution is to remove those solids. But important thing is, that it could be done only after all holes are filled. If not each solid with holes has zero volume. Don't be confused with holes in female screw. Those holes aren't holes in a solid.

I don't count, that this problem will be implemented because of used scans aren't solids. They are masks.

1.2.3 Very close vertices

This section also includes vertices duplicities. Sometimes happens that two vertices with different index has the same or nearly same coordinates.

In that case they are in fact redundant and should be at their place only one representant. The factor of closeness, of course, is dependent on what this model represents.

I think vertex identification should be implemented too. Especially in case of same vertices with different index because in that case they are treated as different.

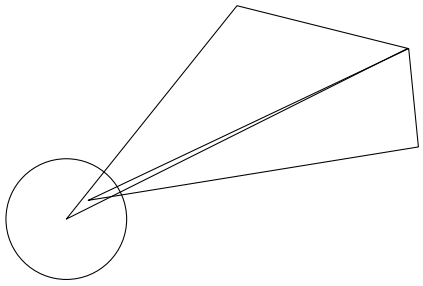


Figure 1.4: Classic example of how small mistake can corrupt mesh, and how vertex unifying can solve the problem.

1.2.4 Crossing faces

Crossing faces means that one intersect another. It it's real intersection of triangles. It does not includes non-manifold faces.

The detection use the triangle triangle intersection detection and fast implementation uses separating axes. [Eberly 2001] [Möller 1997] .

The solution isn't easy too. The easier way is to remove incident faces and than to fill formed holes.

Even though non–manifold faces aren't present in real scanned data, there are several crossing faces as I show in results chapter. I found it out when I tested Rapidform at FNS. So I'd like to implement some algorithms for crossing faces detection.

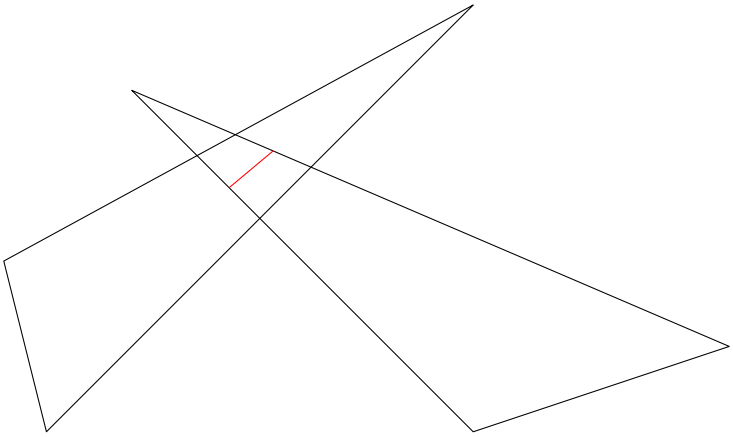


Figure 1.5: An example of triangle – triangle intersection.

1.2.5 Folded faces — too sharp angle between faces

For some time, we were trying to figure out what folded means. I have found it in Rapidform XOS scan manual but there were no other hints [Inu 2009] . So I have tried to recognize it from program itselves. I was successfull. Folded faces probably means too sharp angle between them. The sharpness depends on examined meshes but there exist a simple rule: The sharper angle is, the weirder object is represented by mesh or it is incorrect.

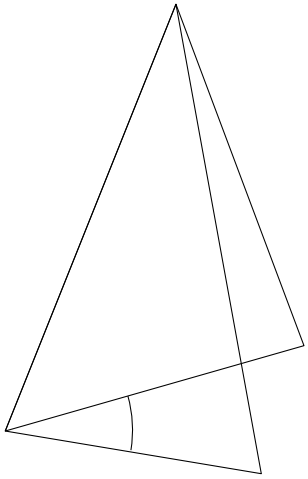


Figure 1.6: Folded triangles depends on size of angle between them

Sometimes it is undesired to have small angles between faces, especially in scanned model which looks unnatural.

Solution is to smooth sharp edge. Mesh decimation is also possible and works great.

This could be implemented in future, but now it isn't one of the hottest problem because it can be easily solved by mesh decimation.

1.2.6 peeks

Peeks are similar problem but there is another system of detection. Peak doesn't have to have any sharp angle between faces.

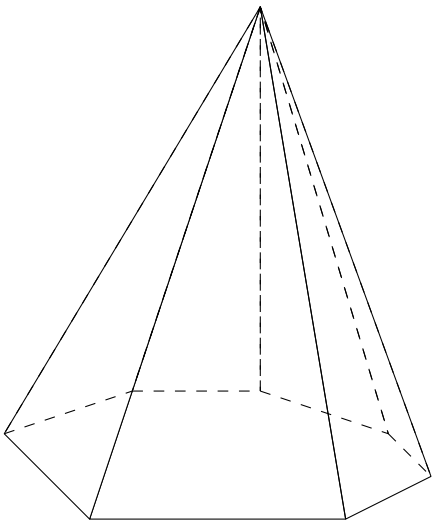


Figure 1.7: An example of peek

Solutions are same. Just delete peek and fill the hole there.
This could be way of extension of mesh mending algorithms in Morpho.

1.2.7 Small tunnels

I found category called small tunnels in rapidform manual According to rapidform manual it should be something small in model, which looks exactly the same as a tunnel. [Inu 2009] When I have tried to figure out what it really is on real models I have just seen nearly 10 faces without any visible correspondence with a tunnel. So I have decide to leave this category behind.

2. Algorithms & Data Structures

There I describe two things. At first, I mention each mesh mending solution implemented in Morpho.

I characterize some interesting algorithms and data structures I have used in second part.

2.1 Current implementation

In this section I explain you not only implemented function but also why current function is implemented and how it is in real data from scans. You are appraised with current options in mesh mending menu, in user manual part.

2.1.1 Unconnected clusters

We go through the whole mesh and create components using disjoint set data structure which is described bellow. Then we count size of components and we remove all components except the biggest one.

2.1.2 Filling holes

We reach the most important part of my mending algorithms. That is why this is commonly the basic need of potential users.

I identify holes with search of triangle which has an edge with no neighbour. Then I try to triangulate them using triangulation without adding new vertices. I control if this create non–2–manifold face and I don't allow that.

Texture coordinates and normal vectors

We simply keep old information in each vertex because we don't create any new vertices.

2.1.3 Triangles intersections

This algorithm couldn't be implemented using by brute force. On real data it would take hours. Thus we implemented fastener data structure KD tree (see bellow).

Unfortunately, triangle–triangle intersection test isn't fast despite KD tree improvement. So, I have an idea to eliminate triangles that share an edge. Those triangle can't have any inner intersections unless they are same. Now we are happy because this case is covered by removing same triangles algorithm.

But I went little further that I excluded triangles sharing a vertex. This covers also cases when they have inner intersection but we could do these occurrences test separately.

2.1.4 Unconnected vertices

This algorithm is simple. We search references and then delete unlinked vertices. Thus in our array implementation we need to reindex, it's not enough only to delete vertices.

2.1.5 Close vertices

Whenever vertices are too close or even the same, we should unify them. It's implemented simply by hash table with some precision defined by smallest piece input by floating point number.

2.1.6 Zero volume faces

Each face with zero volume is removed.

It's implemented like basic cycle through faces. Caution disconnected vertices aren't deleted. It's necessary to run and delete unconnected vertices.

2.1.7 Identification of faces

If two or more faces with same vertices coordinates are found, each of them is deleted, new one with interpolated texture coordinates and normal vectors are created.

It's implemented using a hash table. It's recommended to run vertex unification before because there aren't check space coordinates but only vertex indexes.

2.2 Well known and interesting used Algorithms & Data Structures

2.2.1 KD Tree

The k-d tree is a binary tree in which every node is a k-dimensional point. Every non–leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as subspaces. Points to the left of this hyperplane represent the left sub-tree of that node and points right of the hyperplane are represented by the right sub-tree. The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the k-dimensions, with the hyperplane perpendicular to that dimension axis. So, for example, if for a particular split the "x" axis is chosen, all points in the subtree with a smaller "x" value than the node will appear in the left subtree and all points with larger "x" value will be in the right sub tree. In such a case, the hyperplane would be set by the x-value of the point, and its normal would be the unit x-axis. text and image are cited from Wikipedia [KDw] . I use an implementation from [Alvares] .

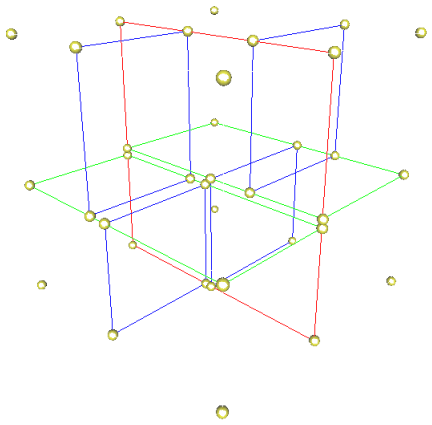


Figure 2.1: Visualized example of 3 dimensional kd tree

2.2.2 Disjoint set data structure

The purpose of this data structure is to contain elements separated in sets (components). This container should provide two basic operation: Union and Find. Union gets two components and unifies them into one. Find returns the number of corresponding set.

One clever implementation is described in [Mareš 2007] , originally published by Tarjan and van Leeuwen. This implementation of algorithm has expected time $O(\alpha(n))$ for operation.

I use a C# implementation from [Stefanov] .

2.2.3 Triangle – Triangle intersection

I implemented fast triangle–triangle intersection test based on the paper by [Eberly 2001] . This method is called separating axes. It works on principle of projection onto axis. Briefly, if projections onto axe are separate there is no intersection. If projections aren't separate there could be an intersection, we don't know. In that case it has to be projected onto another axe. For triangles there is a finite number of separation axes.

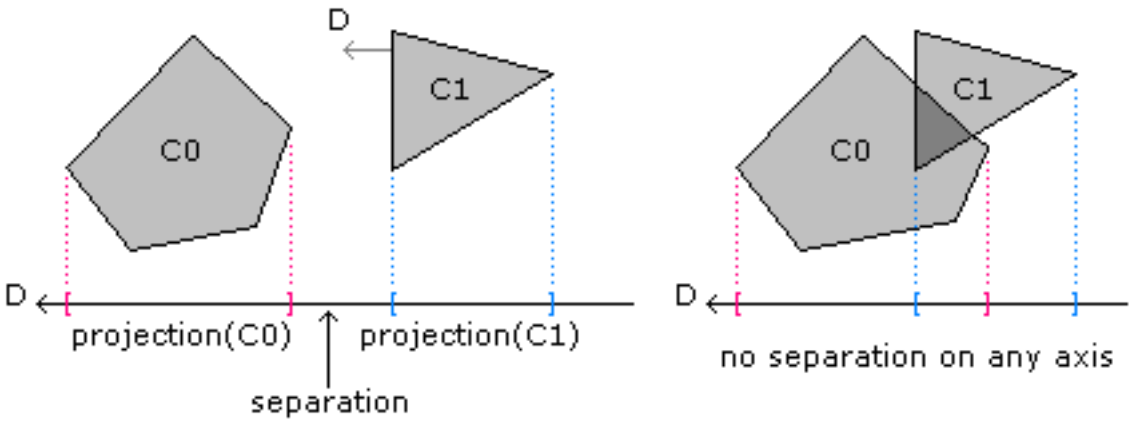


Figure 2.2: An example of projection and separator.(from [Eberly 2001])

I inspired also with a C++ implementation from [Geo] .

2.2.4 Hash table

As it wasn't visible at first look, hashing is a necessary tool for operating with such a large data. And many of my mending algorithms use hashing to determine for example common vertices etc.

I used standard C# collection for hash like Dictionary and HashSet classes are.

3. User manual

I demonstrate basic Morpho's tools in this chapter . Also I explain its installation and give you complete user manual of a mesh mending part.

3.1 Installation

Morpho contains windows installer. This installer also installs prerequisites like R program&packages and RDCOM. This secondary application isn't required for proper running of mesh mending part but it is used in other Morpho projects.

Figure 3.1: Morpho setup screenshot

Firstly you have to unzip archive and then run file setup.exe. The installation is very simple and good described. You should proceed instructions.

If you prefer to not install Morpho with installer, you could simply run binary file. In that case, some part of Morpho doesn't have to work properly if you haven't installed R and RDCOM correctly.

Whatever your decision is, I recommend you to update also yours .NET framework and graphics card drivers.

3.2 Where could be found Mesh mending in Morpho?

After successful installation run program Morphome3cs.exe or Morphome3cs(Advanced user mode).exe, which has some advanced components and options visible. For our purpose doesn't matter which you run.

First of all basic Morpho options appear(see screenshot).

You could try some advacement morpho offers for you. For detail description see [Mor b] .

And now, I navigate you to Mesh Viewer which contains my mesh mending algorithms implementation. Open Tools then choose Mesh Viewer.

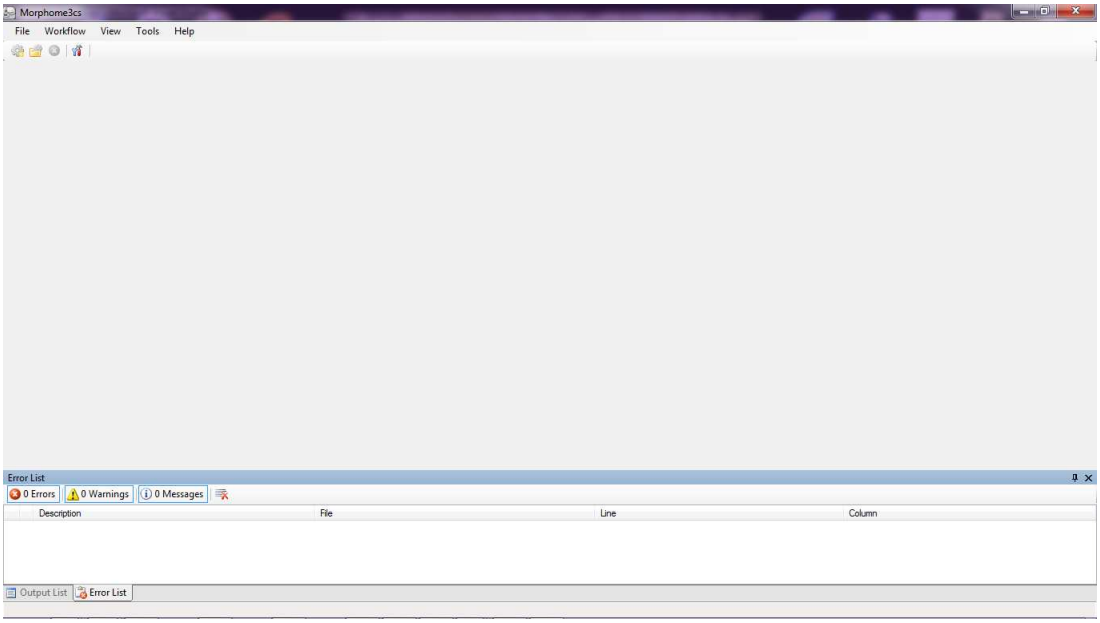


Figure 3.2: Morpho program after startup screenshot

3.3 Mesh Viewer

We reach the section capable viewing meshes. Of course, it could load or save mesh into Wavefront technology obj file format.

Mesh could be viewed in many modes like: Wireframe, Point cloud, Surface with or without texture, shading. The most interesting is third column in menu containing Mesh mending possibilities to choose. Apart from that we could disable face highlighting from previous mesh mending we could choose to run mesh mending easily from GUI menu or quickly from configuration file.

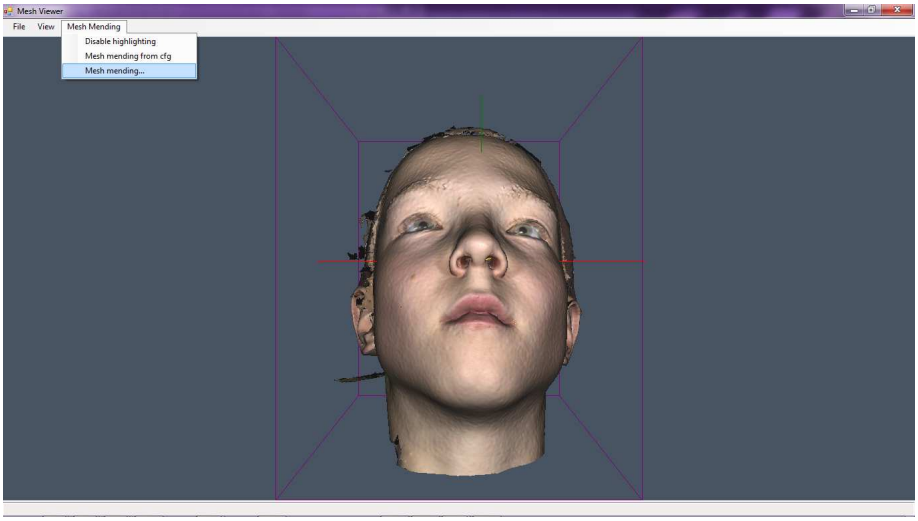


Figure 3.3: Mesh Viewer screenshot

3.3.1 .obj

Obj is a file format containing list of vertices and faces. Also it could contain normal and/or texture coordinates. This format is developed by Wavefront technology. If you are interested in obj file structure see [obj] .

3.3.2 GUI

GUI is composed from several checkboxes. Simply, if you would like to run some algorithm on loaded mesh just check the checkbox.

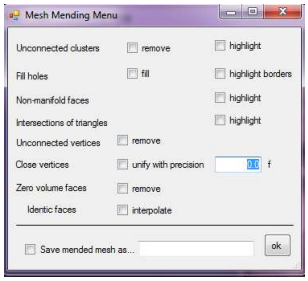


Figure 3.4: Mesh Mending Menu

3.3.3 Configuration file

Configuration file was in Morpho only for purpose of testing. At the end I found it useful and keep it for all users because it saves users' time.

Operate a configuration file is easy. The line starting with '//' is a commentary line. Important are lines starting with 'T' or 'F'. Whenever a line started with 'F' – false, it's like you uncheck the corresponding checkbox in GUI. Whenever a line started with 'T' – true, it's like you check the corresponding checkbox in GUI. All parameters simply write after the letter.

```
////////////////////////////////////////////////////////////////////////////////
//                    CONFIGURATION FILE FOR MESH MENDING                     //
////////////////////////////////////////////////////////////////////////////////
// Line starts wit "//" is commentary and they are ignured.
// Other unempty line is treated as command read from up to down.
// Whenever 'T' is read that means True and command is run.
// Whenever 'F' is read that means False and command won't be executed.
// If you are confused with this configuration file please use GUI menu instead.

// OUTPUT file path
// Relative path from binary exe file of text output file
../../bin/output/MeshMending.out

// REMOVE zero volume faces
/// <summary>
/// It identifies and removes faces with no volume.
/// It would be good to run vertex unification before, because, this works only on index bases, not on space coordinates.
/// </summary>
/// <returns>No. zero volume faces</returns>
F

//REMOVE identic faces
/// This function delete one of the same faces.
/// Same means with same vertex indexes, not same vertex coordinates. (in that case you have to run vertex unification first.)
/// If possible it also interpolates texture coordinates and normal vectors.
F

//REMOVE unconnected vertices
/// Finds and removes unconnected verices.
/// Saving mesh is required if you would like see results.
T
```

Figure 3.5: Configuration file screenshot

3.3.4 Output file

Regardless you run mesh mending algorithms from GUI or from configuration file they generates an ouput file. There are written some interesting numbers according to algorithm that have been run.

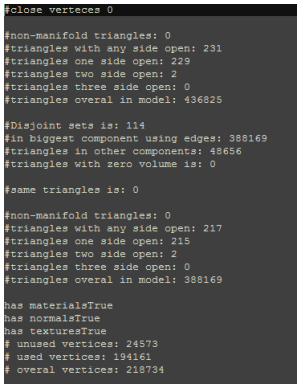


Figure 3.6: A screenshot with example of outputfile

3.4 User's Guide

I describe below options and capabilities of mesh mending algorithms that are implemented.

If you change mesh, results aren't viewed immediately but they are saved. They aren't viewed unless you load mended mesh. If you only highlight something a change will be shown immediately. This is because you could simply delete faces and also highlight those which are deleted.

Text results are saved into text file according address you specify in a configuration file relatively to a binary file. The configuration file please keep in the same folder as a binary file.

Table 3.1: Mesh mending GUI's choices

- *Unconnected clusters: remove* — This option removes all other components except the biggest one while unconnected means unconnected with an edge. Especially, a connection using only vertices doesn't count.
- *Unconnected clusters: highlight* — It highlights components which aren't connected with the biggest one.
- *Fill holes: fill* — By checking this option holes are filled using basic triangulation without adding new vertices. Three–side–open triangles should be deleted before filling(by checking option remove unconnected cluster for example). It's good for small holes but for bigger ones is better using some advanced fill hole algorithm. This option keeps texture coordinates and normal vectors.
- *Fill holes: highlight* — It highlights triangles with any neighbour missing. There are three types of endtriangle: One–side open, two–side open and three–side open triangle.
- *Non–manifold faces: highlight* — It highlights faces that doesn't satisfy the definition of 2–manifoldness
- *Intersections of triangles: highlight* — It highlights faces with geometric intersection. This doesn't include topological crossing (non–2–manifold faces).
- *Unconnected vertices: remove* — It removes vertices which aren't linked to faces.
- *Close vertices: unify* — It unifies very close vertices. In default value (0), only vertices with same coordinates are unified. It could by changed by inserting another value in a floating point number. It represents the smallest unit of precision.
- *Zero volume faces: remove* — It removes degenerate faces which has no volume. (lines and also points in fact)
- *Identical faces: interpolate* — It unifies faces with same coordinates. It interpolates their texture coordinates and normal vectors.

4. Programmers documentation

I'd like to say some words for programmers or other professionals in this chapter . Of course all methods are shortly commented in source code, so I only summon it up. I also point up some interesting passages.

4.1 Structure of Morpho

Morpho is developed as Microsoft Visual Studio 2008 solution. For more information check user's manual [Mor b] , programmers documentation [Mor a] or webpage with references to all documentation [Mor c] .

4.1.1 MeshViewer

All methods of Mesh Mending project are called from Mesh viewer GUI.

Meshviewer itself is Visual Studio 2008 project called Viewer3D. It only contatins a form and its methods.

Highlighting

Highlighting which I'm using in for better viewing mesh errors isn't implemented in Mesh Mending project. It has its implementation in Mesh Viewer, which could color triangle according to its indexes.

4.2 Structure of Mesh Mending project in Morpho

Mesh mending part is hierarchically one of Visual Studio 2008 C# projects. It contains basic data structure and current mesh mending development. It also contains GUI container.

4.2.1 Using other classes in Mesh Mending project

I'd like to introduce some references I've used in Mesh Mending project apart from basic system libraries.

OpenTK library

I've used from Open TK only small functionality, like it Vectors and vector multiplication etc. For more information about open TK see [Ope] .

DataStructures

I've used some basic morpho data structures like the Mesh class which contains and provide all mesh functionality. Including deleting faces and vertices which I had to implement. Because of fast implementation, deleting is physically possible

only when mesh is saved. Otherwise, If I desire to require one additional query to mesh class, whether the element is on delete list or not.

By using Mesh class I'm leed to use other data structure like struct Point3f. Basic container for storing 3 floating point numbers.

InOut3DSupport

I've used also Mesh loader and saver for loading and saving to and from Vawefront obj file format. In those parts I also write some changes. I correct normal vectors and texture coordinates saving. Also I keep texture file information in a saved file. Above that I implement support for deleting faces and vertices which I mention by Mesh class.

4.2.2 Important classes in Mesh Mending project

All mesh mending algorithms are run as methods of dynamic object called MendedMesh. It is initialized from static class Mending which takes care of streams above initializing MendedMesh object.

Advanced auxiliary algorithms and data structures have it's own classes for transparency. They are KD Tree branch, DisjointSet class and TriangleTriangleIntersection static class. All of them are described in algorithms chapter.

In contrast, small containers, which is implemented as C# structures are all covered by Mending.cs file. They are different implementation for edges, triangles and their combination. Whatever had been needed at time of usage.

The last class generates GUI container with all option needed to control mesh mending process.

5. Results

In this chapter I'd like to provide you some results Morpho Mesh Mending Part has. I've run it on five real scanned meshes.

Firstly, I try to run each of my algorithms separately and examine results. Secondly I try to clean them as they have been cleaned at Faculty of Anthropology.

5.1 Test each of algorithms Separately

I've run tests on five different meshes and I count occurrences. I'm also trying to judge success of each algorithm. I point out some possibilities of their future extensions.

Before I provide you individual results I'd like to add some interesting numbers which average mesh has.

Table 5.1: Average mesh stats

- *Number of faces* — $438535,4$
- *Number of vertices* — 219571

5.1.1 Close vertices

Although algorithms allows to set precision, I tested it only with zero precision. That means only vertices with exactly the same coordinates are unified.

In average mesh was zero vertices unified. If mesh contains ununified vertices it could cause many troubles. That is why vertex unification is implemented.

5.1.2 Filling holes

Firstly I show you results of filling holes.

Secondly I must say that all holes aren't filled. In average case there is 5 one–side open triangle from 213.4 that were there previously. and there is 0.6 two–side open triangle from 2.4 that were there previously. Three–side open triangles wasn't present. This algorithm never creates non–2–manifold triangle neither.

For future extension, there should be border identification algorithm. The biggest hole then should be excluded from hole filling. It would be nice though many of meshes are masks in fact. They are only surface, no solids.

5.1.3 Interpolation of identical faces

It isn't apparent but in real data there are several cases of identical faces. In average mesh thereis 11.6 of those faces(0.002%).

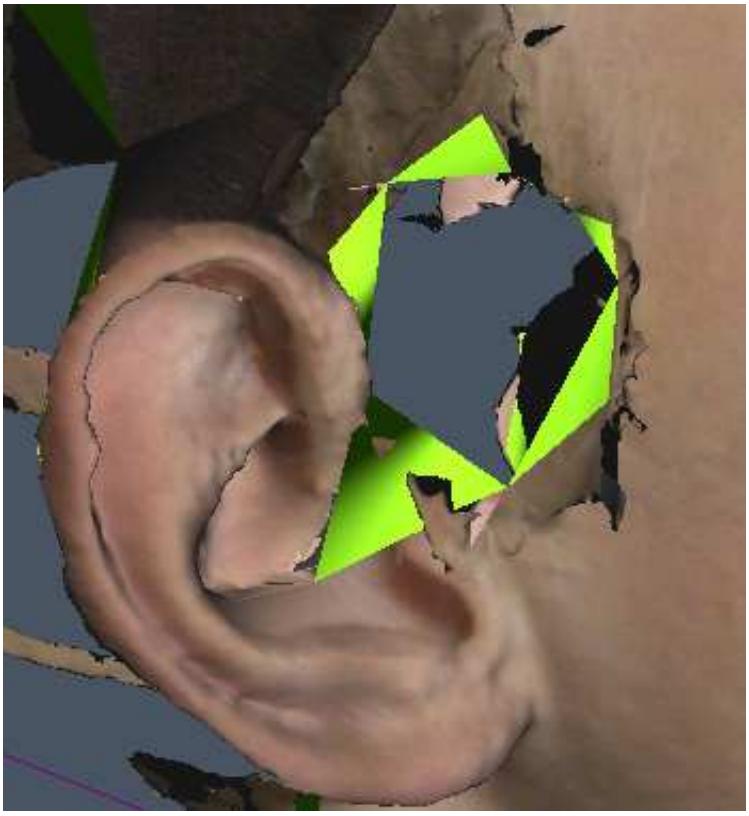


Figure 5.1: Example of a big hole

5.1.4 Non–2–manifoldness

There aren't any non–2–manifold faces in all five tested meshes. No occurrences in data is the reason why we decide not to implement correcting algorithm.

5.1.5 Triangle intersections

There is many occurrences of triangle intersections in real scanned mesh. I was able to identify $5506,6$ which is almost 1.26%. However I wasn't able to identify all of them. Unidentified intersections are those which have common vertex. I thought there are very few if hardly some.

5.1.6 Unconnected clusters

This algorithm works great. Medium tested mesh has $20699,8$ faces, which is 4.72% of all faces.

5.1.7 Unconnected Vertices

This helps especially after we had already run unconnected clusters algorithm. In other cases there are only 18 vertices in average mesh. This is only 0.008%.

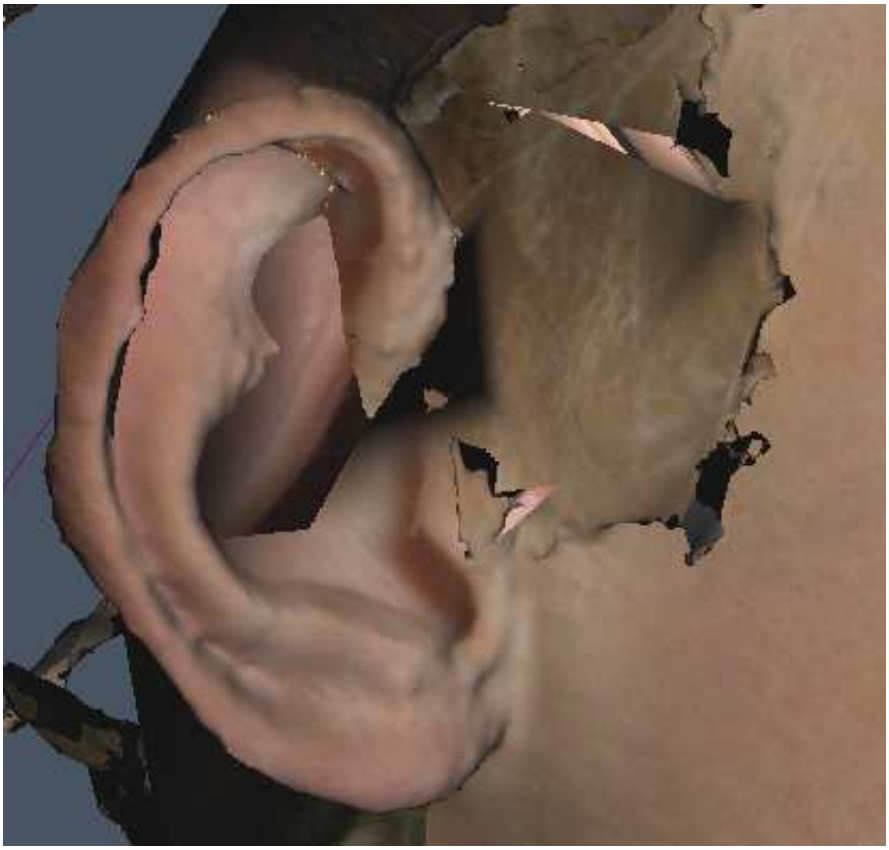


Figure 5.2: Example of a big filled hole

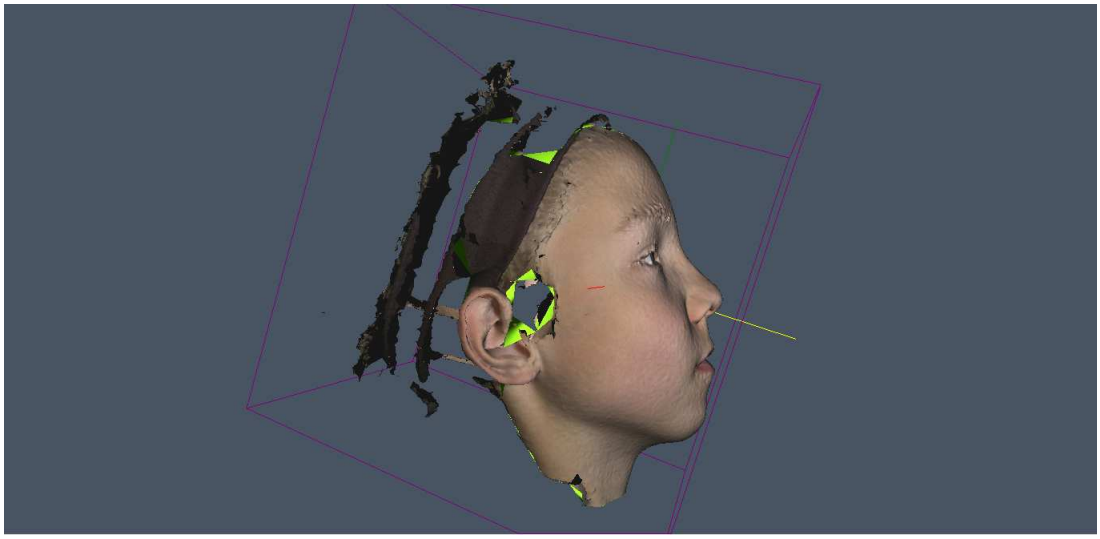


Figure 5.3: Highlighted borders of mesh

5.1.8 Zero volume faces

In tested meshes there were no occurrences of zero volume faces.

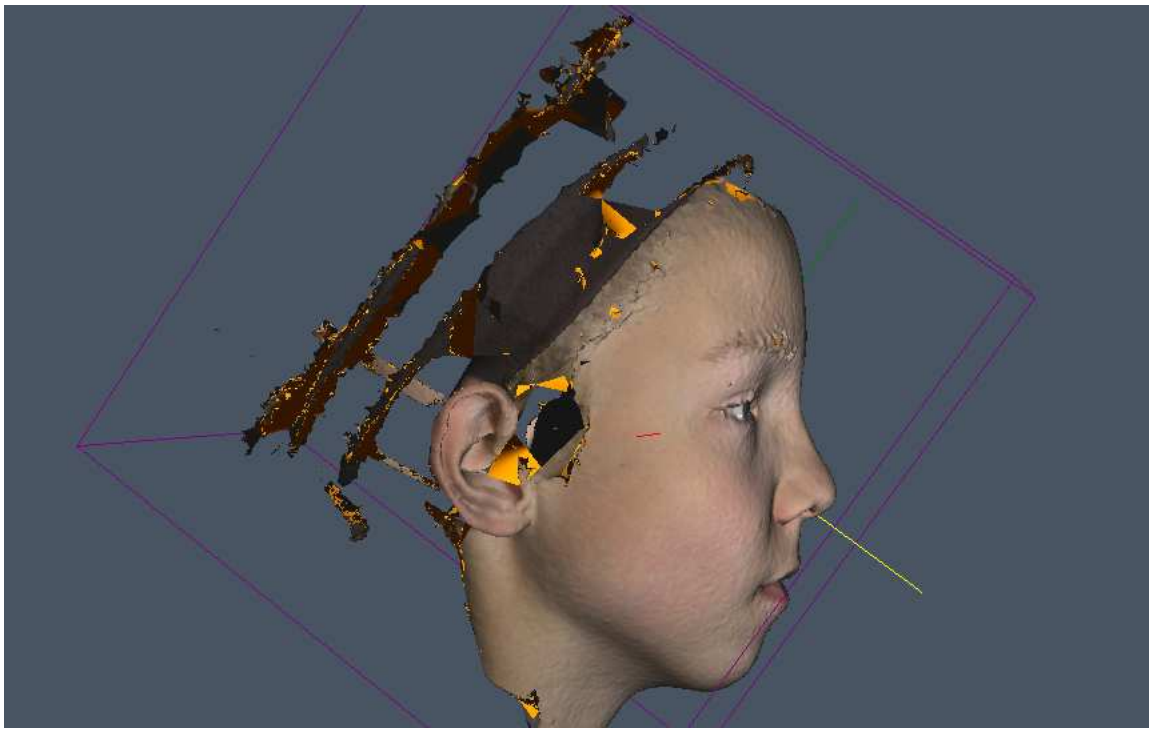


Figure 5.4: Highlighting of faces which have intersection with another face

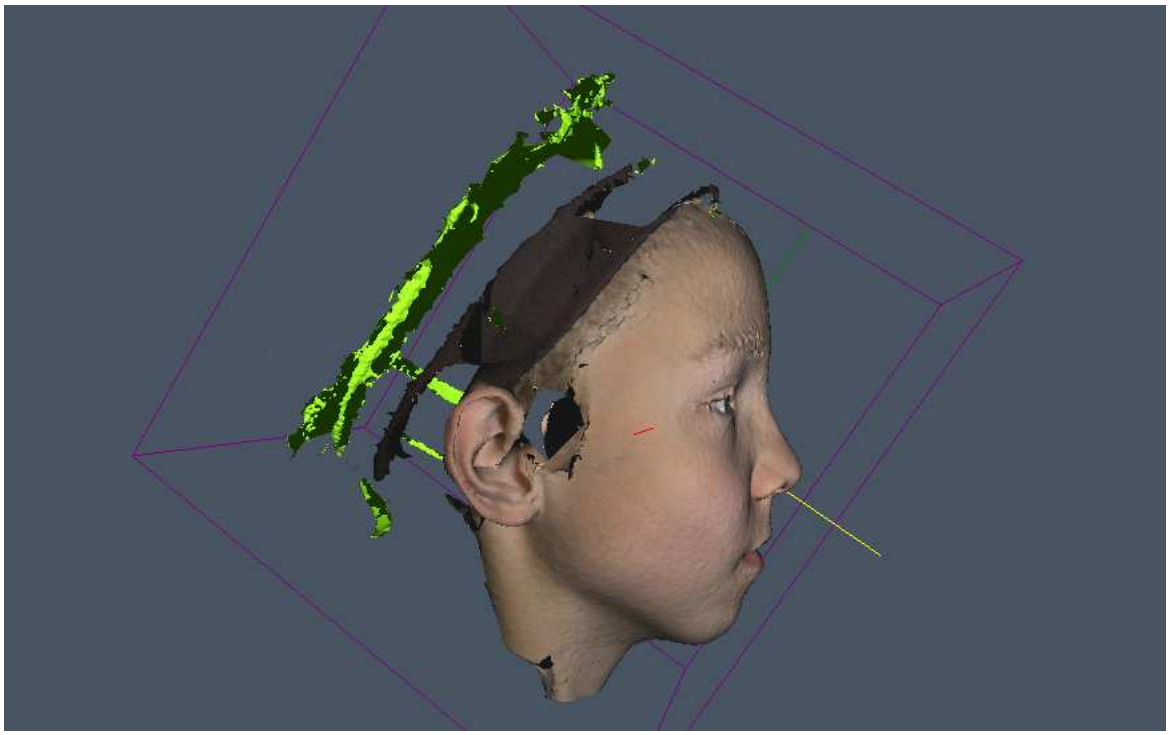


Figure 5.5: Highlighting of unconnected clusters

5.2 Global tests

Interaction between algorithm works mainly because of they are run separately in fact. The sequence of their executing is based on work of [Marek 2010] . I haven't found any problem in this test . No occurrences of previously cleaned error. In general, the global test works as all its parts work.

6. Conclusion

I'd like to summon up my all work. The program I've written provides basic mesh mending functionality. It is integrated in Morpho framework which is developed at MFF. During my work I meet and sometimes edit some parts of project.

My program could be easily extended and it has both programmer's and user's manual. All these have been written in this thesis.

Automatic cleaning process meanwhile couldn't replace a human operated mesh cleaning. Still there is a problem with determinig borders for croping a mesh. But we have done small steps to simplification of this process.

At the end I add some remarks.

6.1 pros and cons while programming with c# and .NET framework

Pros are almost clear. You don't have to program all from green table. Cons are clear too slowness of interpreted code and close sources but I won't like discuss this ever discussed topics here. I'd like to point out two concrete cases that have surprised me.

6.1.1 hash

I've expected C# structure which could have standard function GetHashCode predefined correctly. To my surptise, the program with this standard implementation, runs half an hour.

The wonderment was even greater when I override it with stupid implementation returning only summarized multiplied hasCodes of each part of structure. This works great and it runs only several seconds.

6.1.2 Enumerator

This one isn't as important as previously mentioned. When you get enumerator, you could get current element. But unfortunately you get default (nulled) element. First element you get when you called next element of enumerator.

6.2 About programming in Morpho project!

It has been hard for me writing in Morpho because I've never been writing for such a large project. Some time took me communication overhead and many hours I try to orient.

Now I'm happy that I could get many experiences from it.

6.3 TEX

I'm glad that I can write my bachelor work in typographic system TEX. It really helps me much.

6.4 Future work

I try to remark whenever something could be developed in future. Now I only summon it up. There rests some big part untouched. Angles covering both peeks and sharp angles(dangling faces).

Also, some conversion to better format when mesh is mended. Something like winged edge could be suitable. Of course it would be nice if you could work in with this format.

From users case there could be implemented better viewer with proper zooming. The best will be if it isn't only a viewer but an editor. Editor could provide some basic functions like: Making, Deleting and also Creating faces.

Bibliography

3D laboratoř katedry antropologie, Přírodovědecká fakulta, Univerzita Karlova [online]. [cit. 18. 5. 2011]. Dostupné z: http://www.natur.cuni.cz/biologie/antropologie/pracoviste-katedry/laborator-

Geometric tools [online]. [cit. 3. 7. 2011]. Dostupné z: http://www.geometrictools.com/.

KD tree [online]. [cit. 28. 6. 2011]. Dostupné z: http://en.wikipedia.org/wiki/K-d_tree.

Morphome3cs 2 Developer Manual, a.

Morphome3cs 2 User Manual, b.

Morphome3cs Documentation [online]. [cit. 29. 6. 2011]. Dostupné z: http://cgg.mff.cuni.cz/trac/morpho/wiki/Documentation.

OpenTK webpage [online]. [cit. 29. 6. 2011]. Dostupné z: http://www.opentk.com/.

PICZA^{TM}LPX-1200/600/60 [online]. [cit. 18. 5. 2011]. Dostupné z: http://www.rolanddg.co.uk/content/public/products/LPX120060060.aspx.

Check Repair Meshes [online]. [cit. 3. 7. 2011]. Dostupné z: http://www.rhino3d.com/4/help/commands/CheckRepair-Meshes.htm.

Morphome3cs [online]. [cit. 18. 5. 2011]. Dostupné z: http://cgg.mff.cuni.cz/trac/morpho/.

Projekt Morphome3cs - specifikace [online]. [cit. 20. 5. 2011]. Dostupné z: http://cgg.mff.cuni.cz/trac/morpho/wiki/ProjectSpecification.

Projekt Morphome3cs II - specifikace [online]. [cit. 20. 5. 2011]. Dostupné z: http://cgg.mff.cuni.cz/trac/morpho/wiki/ProjectSpecificationII.

Morphome3cs. [software], 2011. Version: 1.0.0.1.

OBJ file specification [online]. [cit. 3. 7. 2011]. Dostupné z: http://www.martinreddy.net/gfx/3d/OBJ.spec.

THE STANDARD SOFTWARE FOR 3D SCANNERS^{TM}RapidForm^{TM}2004. [software], 2004. Version: RapidForm 2006.

THE STANDARD SOFTWARE FOR 3D SCANNERS^{TM}RapidForm^{TM}2006. [software], 2006. Version: RapidForm 2004 PP2.

Rapidform®^{TM}XOS^{TM}SCAN. [software].

Obličejový skener Vectra3D [online]. [cit. 13. 5. 2011]. Dostupné z: http://cgg.mff.cuni.cz/~pepca/faces/vectra3D.pdf.

Homeomorphism [online]. [cit. 18. 5. 2011]. Dostupné z: http://en.wikipedia.org/wiki/Homeomorphism.

Manifold [online]. [cit. 18. 5. 2011]. Dostupné z: http://en.wikipedia.org/wiki/Manifold.

ALVARES, M. A. *KD-Tree Implementation in Java and C#* [online]. [cit. 20. 6. 2011]. Dostupné z: http://home.wlu.edu/~levys/software/kd/.

EBERLY, D. *Intersection of Convex Objects: The Method of Separating Axes* [online]. 2001. [cit. 24. 6. 2011]. Dostupné z: http://www.geometrictools.com/Documentation/MethodOfSeparatingAxes.pdf.

Rapidform®^{TM}XOS^{TM}SCAN User Guide. InusTechnology, Inc., Soul, Jižní Korea, Září 2009.

MAREK, P. Optimalizace pořadí kroků při opravách povrchových mřížek. Master's thesis, České vysoké učení technické v Praze, Fakulta elektrotechnická, 2010.

MAREš, M. *Krajinou grafových algoritmů*. Praha : Institut teoretické informatiky, 2007.

MöLLER, T. A Fast Triangle-Triangle Intersection Test. *journal of graphics, gpu, and game tools*. 1997, 2, 2, s. 25–30.

STEFANOV, E. *Disjoint Sets Data Structure Implementation* [online]. [cit. 20. 6. 2011]. Dostupné z: http://www.emilstefanov.net/Projects/DisjointSets.aspx.

VELEBA, B. D. Opravy nekonzistentních povrchových sítí. Master's thesis, České vysoké učení technické v Praze, Fakulta elektrotechnická, 2008.

List of Figures

List of Tables

List of used shortcuts

- *GUI* — Graphical User Interface
- *MFF* — Faculty of Mathematics and Physics, Charles University in Prague
- *Morpho* — Morphome3cs
- *FNS* — Faculty of Natural Science, Charles University in Prague
- *Trimesh* — Triangle mesh

Apendix

Table 6.1: CD content

- *Morpho setup* — Windows instaler for Morpho with its requirements.
- *Mesh configuration file* — Example of configuration file for mesh mending algorithm.
- *Morpho source code* — Complete source code of Morpho project.
- *Bachelor source code* — Complete souce code of my Bachelor thesis.
- *Mesh examples* — Some meshes for testing.