Master Thesis

A robust method for brush-based interactive geometry cloning

Author(s):
Meyer, Remo

Publication Date:
2013

Permanent Link:
https://doi.org/10.3929/ethz-a-009908656

Rights / License:
In Copyright - Non-Commercial Use Permitted
A robust method for brush-based interactive geometry cloning

Remo Meyer

Master Thesis
June 2013

Supervisors:
Dr. Kenshi Takayama
Prof. Dr. Olga Sorkine

ETH
Swiss Federal Institute of Technology Zurich

IGL
Interactive Geometry Lab
Abstract

This work introduces a method for 3D mesh cloning. We apply the concept of a cloning brush known from several image editing tools. In contrast to the other existing methods that tried to apply the idea of a cloning brush to 3D geometry, we do not restrict the user to a predefined canvas giving her more freedom in editing. Furthermore, our approach is the first of its kind that allows additive cloning preserving existing geometry details. All steps of our method may be computed in real-time, including the computation and placement of the canvases.

Our algorithm consists of the following steps: First a correspondence between source and target is established using the discrete exponential map algorithm. The warping of the surface is done using Laplacian coordinates and solving a sparse linear system with a GPU solver specially tailored to our problem. As Laplacian coordinates are not invariant to rotations, we need to align them before solving the linear system. To estimate that rotation robustly, we first compute the local frame at each point and then interpolate the rotations from source to target using quaternions. Finally the cloned patch is stitched to the target mesh, ensuring a smooth result with $C^1$ continuity at the boundary of the region of interest.
Zusammenfassung


Contents

List of Figures ix
List of Tables xi

1. Introduction 1
   1.1. Contribution .................................................. 2
   1.2. Goals .......................................................... 3
   1.3. Structure ...................................................... 4

2. Related Work 5

3. Overview 9
   3.1. Definitions .................................................... 9
   3.2. Workflow ...................................................... 10
      3.2.1. Source Seed State ....................................... 10
      3.2.2. Target Seed State ....................................... 11
      3.2.3. Brushing State .......................................... 12
      3.2.4. Canvas Adjustment State ................................ 12
      3.2.5. Baking .................................................... 12
   3.3. Algorithm ..................................................... 14
      3.3.1. Error Formulation ....................................... 14
      3.3.2. Initialization ............................................. 15
      3.3.3. Cloning ................................................... 16
      3.3.4. Stitching ................................................. 16
      3.3.5. Alternative Mode: Layering ............................. 17
## Contents

### 4. Parametrisation 21

4.1. DEM - Discrete Exponential Map 22

4.2. Optimisations 23

#### 4.2.1. Normal Smoothing 23

#### 4.2.2. Frame-Propagation 24

#### 4.2.3. Overlap Correction 27

4.3. Canvas Growing 28

4.4. Incremental Update 29

### 5. Laplacian Coordinate Alignment 31

5.1. Naive Approach 31

5.2. Rotation Interpolation 32

5.3. Normal Smoothing 33

### 6. Multigrid Poisson Solver 35

6.1. The Multigrid Approach 35

#### 6.1.1. Residual 36

#### 6.1.2. Restriction 36

#### 6.1.3. Correction 37

#### 6.1.4. Relaxation 37

#### 6.1.5. Termination 38

6.2. Boundary Conditions 38

#### 6.2.1. Dirichlet Boundary Conditions 38

#### 6.2.2. Neumann Boundary Conditions 39

### 7. Results 43

7.1. Performance 43

#### 7.1.1. Solver Performance 43

#### 7.1.2. Overall Performance 44

7.2. Derivative Constraints 45

7.3. Comparison to GeoBrush 46

7.4. Laplacian Coordinate Alignment 48

7.5. Limitations 49

### 8. Conclusion and Future Work 51

8.1. Conclusion 51

8.2. Future Work 51

### A. User Interface 55

A.1. States 55

#### A.1.1. General Handling 55

#### A.1.2. Source Seed State 56

#### A.1.3. Target Seed State 56

#### A.1.4. Brushing State 56

#### A.1.5. Canvas Adjustment State 56

A.2. Keybindings 57

A.3. Configuration File 58
List of Figures

1.1. GeoBrush robustness issues .................................................. 2
3.1. Source seed state ............................................................. 11
3.2. Target seed state ............................................................. 11
3.3. Brushing state ............................................................... 12
3.4. Canvas adjustment state ..................................................... 13
3.5. Baked result ................................................................. 13
3.6. Cloned mesh $\tilde{R}_{in}$ before stitching ................................ 17
3.7. Gap filled with stitch mesh ............................................... 18
3.8. Comparison of the standard and the layering approach ............... 19
3.9. Example with several layers applied .................................... 19
4.1. Basic DEM drawbacks ....................................................... 23
4.2. Foldovers in DEM due to normal smoothing ............................ 24
4.3. Comparison of the DEM optimisations ................................... 26
4.4. Benefits of the frame-propagation DEM over the original DEM .......... 26
4.5. Failing original DEM ....................................................... 27
4.6. The overlap correction ..................................................... 28
4.7. Comparison of the parameter space textures ............................ 29
5.1. Naive alignment of Laplacian coordinates .............................. 32
5.2. Interpolated rotations to align the Laplacian coordinates .............. 32
5.3. Laplacian coordinates aligned with smoothed normals .................. 33
6.1. Red-Black scheme .......................................................... 37
6.2. Grid points categorization ................................................ 39
6.3. Restricted boundary points ............................................... 40
List of Figures

6.4. Influence of the Shortley-Weller scheme ........................................ 41
7.1. Coating transfer using derivative constraints .............................. 46
7.2. Failing case of derivative constraints ........................................ 46
7.3. GeoBrush comparison ................................................................. 47
7.4. Failing to clone a handle ............................................................. 47
7.5. Brushing over large parts of a mesh and small scale target features .... 48
7.6. Laplacian coordinate alignment in convex and concave settings ......... 49
A.1. Inverted triangles ................................................................. 59
A.2. Laplacian coordinate alignment rotations .................................. 59
A.3. Projected source canvas ........................................................... 60
A.4. Projected target canvas ............................................................ 60
A.5. Region of interest ................................................................. 61
A.6. Mapping of \( R_{in} \) ............................................................. 61
A.7. Mapping of \( R_{out} \) ............................................................ 62
A.8. Stich mesh ................................................................. 62
B.1. Irregular five-point stencil ........................................................ 63
List of Tables

7.1. Solver performance ........................................ 44
7.2. Overall performance ........................................ 45
A.1. Keybindings .................................................. 57
A.2. Configuration file ........................................... 58
Introduction

From animation to CAD, highly detailed meshes are crucial to the creation of appealing results. Such meshes may be created by scanning real-life objects or with the help of mesh modelling tools (e.g. Autodesk Inc. 2013 or Pixologic Inc. 2012). Both approaches tend to be very time consuming, so it would be useful to be able to reuse parts of existing meshes and copy them to a newly created mesh.

The GeoBrush paper [Takayama et al. 2011] addresses this issue by applying the concept of a cloning brush to 3D surface meshes. The cloning brush is commonly used in 2D image editing tools (e.g. GIMP 2013 or Adobe Systems Inc. 2013), where the user may set a stamp in one region of an image and copy it to another part. It is often used to retouch or remove undesired parts of images, while ensuring the plausible appearance of the resulting image. In contrast to other copying methods (e.g. defining a region of interest with a lasso and copying it to another place) this method is far more intuitive and gives immediate feedback to the user. Properties, which would also be desirable in a mesh cloning tool.

Applying the concept of a cloning brush to 3D meshes poses three major problems that need to be solved. First, it is necessary to establish a correspondence between source and target mesh. In the image case this comes naturally as there is an up and right direction. In our case we establish such a correspondence via local parametrisation. Second, there is no obvious solution to the question of what should be copied. It is often difficult to distinguish between geometric details that should be kept and the underlying smooth surface that should be warped to fit the target surface. This has also been a major problem in other works. And finally it is often necessary to ensure at least $C^1$ continuity on the boundary of the copied surface patch.

In general GeoBrush works as follows: the user specifies a part of the source mesh, which is locally parametrised. By choosing a location in the target mesh and parametrising the patch around that location as well, a correspondence between source and target is established. Within
1. Introduction

those corresponding patches the user may now brush a region of interest that is copied. The actual copying happens in two steps. First the source patch is roughly aligned to the target using the cage-based deformation method based on Green Coordinates [Lipman et al. 2008]. In the second step an offset membrane is computed, that takes into account the brushed region of interest and ensures $C^1$ continuity.

While the GeoBrush tool provides a workflow to successfully copy mesh parts, it does not fulfill all the properties the image cloning brush does. The real-time feedback is only given in the last step when brushing the region of interest, because the cage generation and warping are not real-time. The cage generation also tends to have robustness issues, when applied to complicated regions (Figure 1.1).

![Figure 1.1](image.png)

Figure 1.1.: This example shows the robustness issues in GeoBrush. The generated cage has self-intersections (a), thus the cloned mesh has overlaps as well and finally the stitching of the mesh fails (b).

The cage based approach of GeoBrush leads to another drawback. With a brushing tool, the user would expect to be able to brush over the surface and continuously copy geometry from source to target. However, as the user may only brush within a rather small cage region, she has to restamp several times to copy bigger parts of a mesh. This reduces the fluidity of the copying process and the alignment of the different parts becomes especially tedious. This restamping may also lead to visual artefacts as there is no continuous parametrization over the domain.

1.1. Contribution

This work builds on the previous work of the GeoBrush paper and addresses the shortcomings discussed above. Instead of warping the mesh with a cage based approach using Green Coordinates, this work follows the approach of Laplacian surface editing [Sorkine et al. 2004] and
copies the mesh directly without an additional step of the warping computation.

Instead of the typically used global coordinates, Laplacian coordinates describe a vertex as the difference between a vertex position and the centroid of its neighbours. To compute again the global positions of a set of Laplacian coordinates a sparse linear system needs to be solved.

In this work we do not solve this linear system on the irregular domain of the mesh itself but in the regular parametrisation space. This comes naturally as there needs to be a matching parametrisation from source to target anyway. It allows us to solve the problem very efficiently, allowing a real-time editing tool. Our first main contribution of this work is a real-time multigrid solver of the Poisson equation.

As Laplacian coordinates are not rotation invariant, source and target need to be aligned first. To do this, we interpolate quaternions, which describe the rotation of the local frame from source to target. These interpolated quaternions are then used to rotate the source Laplacians into the target frame.

As the brush always needs a correspondence in both the source and the target mesh, a fast and high-quality parametrisation algorithm is crucial to produce appealing results. So another important contribution is the incremental update of the parametrisation which enables continuous brushing over the domain.

Furthermore we propose an adaptation of our algorithm that allows layering of geometry. With that approach it is possible to re-apply the cloning brush and add geometry details while respecting the underlying surface.

1.2. Goals

We want to create a system that allows mesh cloning in an intuitive way. That is, we want to introduce a simple workflow that also allows users with little knowledge about 3D geometry to create satisfying results. It is also required that locally changing the brushing area should only lead to small and predictable changes in the outcome of our system.

Another important aspect of our work is the real-time feedback to the user. Our goal is to provide immediate feedback to the user, allowing her to continuously brush over the mesh. In contrast to previous work we want to achieve real-time feedback not only in the brushing step, but also in the other steps of the system, including the readjustment of the source and target canvas.

Finally we want our system to be robust, so that it also provides satisfying results in geometrically complex situations.
1. Introduction

1.3. Structure

First we give an overview of the previous work done in this field and illustrate what our work is based on (Chapter 2).

In Chapter 3 we give a brief overview of our algorithm. We explain the general cloning workflow. Furthermore, we elaborate on the core algorithm introducing the different steps of our system.

In Chapter 4 we focus on the parametrisation. We define what criteria the parametrisation needs to fulfil for a satisfying cloning result. We explain in-depth how the parametrisation is computed. Furthermore we discuss possible improvements and their influence on the outcome. Finally we show how the size of the canvas is determined and how the parametrisation can be updated in real-time.

Next we explain how we align the source to the target mesh, to overcome the lack of rotation invariance of the Laplacian coordinates (Chapter 5).

In Chapter 6 we explain how our real-time solver works. We elaborate on what the techniques we use and why they improve the solver. An important part of this chapter is the enforcement of the boundary conditions.

Finally we demonstrate our method in Chapter 7. We conclude this thesis with a brief outlook on possible future work (Chapter 8).
Related Work

In this chapter we give a brief overview of existing work that is related to mesh cloning and other aspects of our work.

There have been many works that attacked the problem of copying meshes. In the SnapPaste paper [Sharf et al. 2006] the authors introduced a method they called Soft-ICP. The idea is to automatically align the copied features to a target geometry. Their software is really easy to use, but the range of possible user interactions is also very small. Another disadvantage of that method is the need of preprocessing. The features the user wants to copy need to be cut out of their previous mesh by another method. Also the user must provide a hole in the target surface.

Such a part based approach is also used in other methods. For instance there is the commercial tool meshmixer [Autodesk Inc. 2013], which allows the user to drag a part across a surface. The system computes local parametrisations in the source and target mesh to find a corresponding boundary and merge the two meshes. This dragging process is computed in real-time, allowing the user to interactively place the part.

Another cut-and-paste method has been introduced in [Sorkine et al. 2004]. The authors described the geometry with an intrinsic surface representation. This representation is based on the Laplacians of the mesh, with each vertex being described relative to its one-ring neighbourhood. The core of their cloning method is the computation of a solution to a sparse linear system, to get the global vertex positions. In their work, the linear system is solved on the irregular mesh domain. They also proposed methods to extract the coating, the high frequency details of a surface, from a smooth base surface and transfer it to another surface. With their method it is also possible to mix geometric details. However, their method needs several precomputation steps. The user needs to cut out the part that she is interested in cloning. Furthermore, the coarse alignment of source and target mesh needs to be done manually. Building on the theoretical foundations of this work, we streamline the method for easier usage and more artistic...
Related Work

freedom with the use of a brush-based interface.

There has been previous brush-based work. The commercial tool ZBrush [Pixologic Inc. 2012] provides several 3D brushes. Such brushes act as stamps to enhance a given surface with geometric details. The user may create custom brushes, which are vector-displacements of existing meshes. However, with those stamps the source is fixed.

A cloning brush method that is closer to our work is [Schmidt and Singh 2010]. In their work the user defines a canvas region. In this canvas region, the system extracts vector displacements from a smooth base surface. A corresponding canvas may then be chosen in the target mesh. Once the canvases are set, the user may then brush within the canvas region.

As previously mentioned the GeoBrush method by [Takayama et al. 2011] is also a brush-based mesh cloning approach. The workflow is very similar to the previous method. They also need to predefine the canvas regions and allow the brushing only within that canvas region. The GeoBrush method uses a cage-based approach to roughly align the source and the target canvas. The cage-based deformation is based on Green Coordinates [Lipman et al. 2008]. This cage-based approach has both advantages and disadvantages. It enables to copy also complicated structures with complex topology, as long as the boundary from the source may be mapped to the target. However, the computation of the cage often fails or has self-intersections when it is applied to more complex structures. The placement of the cage in the target, may also be difficult to impossible when applied to small structure or surfaces with high curvature. Our work is based on GeoBrush, but addresses its shortcomings it. We especially remove the cumbersome dependency on a cage and the restriction to a predefined canvas.

Our work is heavily depending on a fast and stable surface parametrisation algorithm called discrete exponential map (DEM) [Schmidt et al. 2006], that allows to build a correspondence between source and target mesh. The idea of the DEM is to compute intrinsic normal coordinates in \( \mathbb{R}^2 \) of the neighbourhood around a seed point in \( \mathbb{R}^3 \). The exponential map provides such a mapping: each point around the seed point may be described by the length and the angle of its geodesic path emanating from the seed point. Their method applies the concept of the exponential map to discrete meshes. In their method the computation of the mapping is reduced to an extended version of Dijkstra’s algorithm. TheDEM is advantageous in several ways; it requires very little computational cost for a stable parametrisation, it does not depend on the quality of the mesh, it can handle holes, and it can also work on point clouds without any connectivity.

There have been several proposals to improve the initial DEM algorithm. In [Schmidt and Singh 2010] several improvements are compared, including normals smoothing, upwind-averaging and a hybrid approach that combines DEM with conformal mapping. The smoothing of the normals is a way of reducing distortions that may occur when working with surfaces with a lot of details and high curvature. Upwind-averaging extends the computation of the parametrisation to several shortest paths instead of only a single path, which further reduces the dependency on the quality of the mesh and leads to a more stable mapping.

Conformal mapping as described in [Desbrun et al. 2002] is a method that provides least-distorted parametrisations based on a given conformal energy formulation. In contrast to the DEM approach, conformal mapping ensures a bijective mapping. To compute the conformal mapping a sparse linear system needs to be solved. The hybrid approach proposed in [Schmidt and Singh 2010] combines the two methods. The base parametrisation is computed with DEM, while in
regions with high curvature the parametrisation is refined using conformal mapping.

In [Schmidt 2013] the frame-propagation DEM is introduced. In contrast to previous versions of the DEM, points further away are not aligned to the initial seed point’s frame, but the frame is propagated, so that each point is only dependant on its local neighbourhood. This change in computation leads to more stable mappings for vertices with normals that differ a lot from the normal of the initial seed point.

Our solver is based on the multigrid solver proposed in [McCann and Pollard 2008]. In their work the user may define the gradients in a 2D image and the system solves a Poisson equation to receive the colour values of the pixels. To enable real-time editing they provide a GPU multigrid solver. Although their solver is related to ours, we have additional problems to handle. First we have a free-form boundary of our domain where both Dirichlet and Neumann boundary conditions are specified. In addition, we also need to solve a higher order problem.
2. Related Work
Overview

The workflow section gives an overview of how the software is used, i.e., what stages the user needs to take to clone mesh geometry. Afterwards, we briefly explain the core cloning algorithm. We outline the different steps needed, as well as the theory our approach is based on. But first we provide some important definitions.

3.1. Definitions

Throughout this thesis when talking about a mesh, we talk about a polygonal mesh, described as follows. A mesh $M$ is described by a triple $(V, E, F)$, where $V = \{v_1, ..., v_n\}$ are the set of vertex positions in $\mathbb{R}^3$. $E$ are the edges, which connect 2 vertices. $F$ are the faces, which connect the edges. We assume that the polygonal mesh is always a manifold, meaning that each edge may have only 2 adjacent faces. We also assume that the mesh is connected.

Although conceptually our method could handle general polygonal meshes, where a face may have any number of edges, in the implementation we assume for simplicity that each face has exactly 3 edges and vertices. This is legitimate as a polygonal mesh can always be triangulated as a precomputation step.

In the following, the mesh from which the geometric data is copied is referred to as the source mesh $M_s = (V_s, E_s, F_s)$, while the mesh to which the geometric data is copied is referred to as the target mesh $M_t = (V_t, E_t, F_t)$. In a future application of this work, it probably would also make sense to allow copying within one mesh and there are no theoretical restrictions that would prevent that. However as this is only a proof of concept and it simplifies the formulation of the problem, we always work with 2 separate meshes.
3. Overview

As there is no notion of canonical orientations in manifold surfaces, we introduce the parameter space $S \subset \mathbb{R}^2$. The brushed area also called region of interest (ROI) is then defined as $Q \subset S$.

To map the mesh vertices to the parameter space we use the parametrization functions $P_s : \mathbb{R}^3 \rightarrow S$ and $P_t : \mathbb{R}^3 \rightarrow S$ for the source and the target respectively.

In addition to establishing a correspondence between the source and the target meshes to enable brushing in 3D, the parameter space also works as a regular domain for solving the linear system, to compute the global vertex positions of the copied geometry in a very efficient way (see Section 3.3.1).

To compute the parametrisation we used the discrete exponential map algorithm (DEM) described in [Schmidt et al. 2006]. An in depth study of the different properties of the DEM and their influence on the outcome can be found in Chapter 4.

Furthermore we want to define the canvases $C_s \subset \mathcal{M}_s$ and $C_t \subset \mathcal{M}_t$. The canvas is the part of the mesh that takes part in the cloning process and thus needs to be parametrized. It is called canvas because the painted area $Q$ is back projected on the mesh, allowing the user to apply the 2D operation of brushing on the 3D mesh. At any time it must be ensured that $P_s(C_s) \supset Q$ and $P_t(C_t) \supset Q$. Thus the canvases need to grow upon brushing (see Chapter 4).

3.2. Workflow

The workflow may best be described as a state machine. In the following, the task of each state and its relation to the other states is shortly described.

This section is only for general understanding of the software, as well as the issues addressed with it. For detailed explanation of the user interface and how different settings may be changed, see Chapter A.

3.2.1. Source Seed State

After both, the source and the target mesh, are loaded and visualized, the user may click a point in the source mesh (Figure 3.1). This first clicked point corresponds to the stamp in 2D image cloning. We call it the seed point $v^0_s \in V_s$ as it is the starting point to compute the parametrisation $P_s$.

The chosen brush size defines together with the seeding point $v^0_s$ the initial canvas $C_s$.

As there is no obvious up direction in a mesh (as opposed to the 2D image case) the parametrisation is aligned according to the up vector of the view. This is often a good guess, but may of course afterwards be adjusted by the user.

Once the source seed point is chosen, the user may go on to the target seed state.
3.2. Workflow

Figure 3.1.: In the first state the user has to define a seeding point in the source mesh (left). The centre of the circle defines the seed point, while the radial line denotes the up vector, i.e., the orientation of the parametrisation.

3.2.2. Target Seed State

In the second step the user then chooses the seed point $v^0_t \in V_t$ in the target mesh, in the same manner as in the step before. The corresponding brush sizes of the two clicks are used as a hint of the relative sizes of source and target and the source geometry is scaled accordingly (Figure 3.2). As with the source, the up vector of the view defines the initial orientation of the parametrisation $P_t$.

Figure 3.2.: In the second step the user selects a corresponding point in the target mesh (right). The stamps indicate, how the copied patch will be scaled and rotated.

As the seeding points are set, we may compute local uv-coordinates, the parametrisation $P_t$ of the vertices in $C_t$. Now the parametrisations enable a correspondence between source and target, as $P_s(v^0_s) = P_t(v^0_t) = (0, 0)$. 
3. Overview

3.2.3. Brushing State

The user may continue brushing and define the region that should be copied. This brushed region is in the following called region of interest (ROI) \( Q \subset \mathbb{R}^2 \). While brushing, the DEM is incrementally updated, so that at each point in time the whole ROI is parametrised (i.e., the Canvas \( C_* \) is enlarged to ensure that its parametrisation \( P_*(C_*) \) keeps to be a superset of \( Q \)).

The ROI can be both painted and erased, which gives the user more freedom to define the region of interest. As the parametrisations define a one-to-one mapping between source and target, the user may not only brush on the target mesh, but also on the source mesh (Figure 3.3). This may be very helpful in situations with complex geometries.

![Figure 3.3:](image)

*Figure 3.3:* After the correspondence is established, the user may define the ROI by brushing. The yellow area is the ROI that will be copied, while the surrounding blue area is the parametrised canvas outside of the ROI. The user may both grow or shrink the ROI by brushing on both the source and the target.

Each time a brushing stroke is finished, an optimization takes place, refining the parametrisation that has been computed incrementally during brushing.

3.2.4. Canvas Adjustment State

This is an optional state for refining the placement of the cloned geometry. In this state the user may change the seeding position of the source and target parametrisation, as well as scale and rotate the region of interest (Figure 3.4).

The user may switch any time between the brushing state and this state, in both directions.

3.2.5. Baking

This is not really a state but an important operation to finalize the cloning. After baking, the warped geometry and the target mesh are glued together to build the new target mesh (Figure...
3.2. Workflow

Figure 3.4.: If the user is not satisfied with the initial positioning, she may change in this state the orientation and the scaling of the canvas, as well as redefine the seed point for the source and the target.

3.5). This transition needs to be evoked explicitly by the user, once he’s content with the cloning. In a future work would could imagine an implicit baking by the system, to keep the brushing area small. This would increase the locality of the approach and also keep the system in a robust state, which can not always be ensured when brushing over longer distances in complex geometries.

Figure 3.5.: Finally the copied surface is merged with the target mesh. The cloned geometry is smoothly integrated with $C^1$ continuity.
3. Overview

3.3. Algorithm

Now that we know the general workflow, we may continue with the core of this work: the cloning algorithm. It is executed every time the user paints the ROI \((Q)\), repositions the seed point \((v_*)\), or adjust the parametrisation scaling and orientation \((P_*)\).

Depending on the change made, an incremental or a full update is needed. Although this is important for a real-time experience, the algorithm stays practically the same, as only the amount of data to process changes. Thus we do not distinguish in this section between an incremental and a full update.

3.3.1. Error Formulation

The copying of mesh data used in our approach is based on the Laplacian coordinates described in [Sorkine et al. 2004]. Laplacian coordinates are an intrinsic surface representation, which describe each vertex by its local neighbourhood.

In the following we use the same terminology as in [Sorkine et al. 2004]. The \textit{neighbourhood ring} of vertex \(i\) is the set of adjacent vertices \(N_i = \{j | (i,j) \in E\}\) and the degree \(d_i\) of the vertex is the number of elements in \(N_i\).

The mesh may then be described as a set of differentials \(\delta = \{\delta_1, \ldots, \delta_n\}\), where the coordinate \(\delta_i\) will be represented by the difference between \(v_i\) and the average of its neighbours:

\[
\delta_i = \Delta v_i = v_i - \frac{1}{d_i} \sum_{j \in N_i} v_j
\]  

(3.1)

With \(\Delta\) being the Laplace operator. Here the Laplacians are formulated with uniform weights. In irregular meshes, especially when they have very elongated triangles, cotangent weights tend to produce better results. However as we use the Laplacians on a regular rectangular domain, uniform weighting is already optimal.

We may now formulate the cloning as an error that needs to be minimized:

\[
\min_{\Omega} \int_{\Omega} \|\Delta x - \delta_s\|^2
\]

(3.2)

with \(x\) being the geometric positions in \(\mathbb{R}^3\) of the cloned mesh and \(\delta_s\) being the Laplacians of the source mesh. Furthermore \(\Omega\) is the part corresponding to the ROI in the parameter space.

The Laplacians have the drawback of not being rotation invariant. So before we can solve the above equation, we need to align the Laplacians to the target. For this we first estimate at each vertex of the canvases \(C_s\) and \(C_t\) a local frame. Then we compute the rotation from source frame to the target frame. As we don’t want to distort the local details but roughly align the two canvases, we interpolate the rotations using quaternions, to get a smooth field of rotations \(R = \{R_i\}\). For further details of this alignment of the Laplacians see Chapter 5. With the frame alignment taken into account the above equation becomes:
3.3. Algorithm

$$\min \int_{\Omega} \| \Delta x - R\delta_s \|^2. \quad (3.3)$$

We now defined the energy that needs to be minimized, but to be able to solve the cloning operation we need to define as well how the source geometry should be embedded in the target. As we want the cloned mesh to smoothly fit into the target mesh, we need to ensure $C^1$ continuity at the boundary of the ROI $\partial \Omega$. Thus we extend the above error formulation by the following positional and derivative constraints:

$$x|_{\partial \Omega} = v_t, \quad \frac{\partial x}{\partial n}|_{\partial \Omega} = \frac{\partial v_t}{\partial n}, \quad (3.4)$$

with $v_t$ being the vertices of the target mesh.

Putting everything from above together results in the following bi-Laplacian PDE:

$$\Delta^2 x = \Delta R\delta_s, \quad s.t. \quad x|_{\partial \Omega} = v_t, \quad \frac{\partial x}{\partial n}|_{\partial \Omega} = \frac{\partial v_t}{\partial n}, \quad (3.5)$$

which minimizes the above error formulation with respect to the given Dirichlet and Neumann boundary conditions.

Solving the above equation has two major problems. First the solving equation (3.5) in $\mathbb{R}^3$ takes too much time to be practical in real-time environment. Secondly the meshes $M_s$ and $M_t$ have no matching interface at the boundary of $\Omega$, thus we would have to remesh them to get matching vertices. To avoid these problems we introduce functions defined in the parameter space for the geometric positions $\tilde{x} : Q \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$, $\tilde{x}(u,v) = (\tilde{x}, \tilde{y}, \tilde{z})$ as well as the frame alignment rotations $\tilde{R} : Q \subset \mathbb{R}^2 \rightarrow \mathbb{R}^4$, $\tilde{R}(u,v) = (\tilde{r}_x, \tilde{r}_y, \tilde{r}_z, \tilde{r}_w)$ with $(\tilde{r}_x, \tilde{r}_y, \tilde{r}_z, \tilde{r}_w)$ being a unit quaternion describing a rotation. This enables use to solve the PDE in the parametric domain instead:

$$\Delta^2 \tilde{x} = \Delta \tilde{R}\delta_s, \quad s.t. \quad \tilde{x}|_{\partial Q} = v_t, \quad \frac{\partial \tilde{x}}{\partial n}|_{\partial Q} = \frac{\partial v_t}{\partial n}, \quad (3.6)$$

3.3.2. Initialization

As we are solving equation (3.6) in the parameter space, we first need to convert the mesh data from 3D to parameter space. Fortunately this comes very natural, as it can be done with simple rendering to a texture using the uv-coordinates computed with the parametrisation $P_s$, where the position and the frame information(as quaternions) are filled into the color values. Of course we do not need to render the whole meshes to the textures but only the vertices that belong to the canvases $C_s$.

The resolution of these textures has a high influence on the outcome of our algorithm. Having too coarse grids, will lead to the loss of geometric details. On the other hand having a too fine resolution will slow down the system, hindering a smooth real-time experience. Optimally the
system would adapt its grid resolution based on the edge lengths in the current canvases. As our system is only a proof of concept the grid resolution is fixed, but can be chosen by the user in the initial settings.

At this point we have 4 textures with information from the meshes (2 for each mesh holding position and frame), as well as the paint texture which holds the brushing information, which is already in parameter space. The frame textures are used to estimate the rotation of the Laplacians from source to target. So we first compute those rotations, which can easily be done using quaternion multiplication. If we would use this rotations as they are at this point, the geometric details would get distorted. So we smooth the texture to get a smooth field of quaternions. For more information on this step see Chapter 5.

Currently we have textures holding the global positions of meshes. For the target case this is sufficient as it is used as an initial guess and to define the boundary constraints. For the source however we are not interested in the global coordinates but in Laplacians coordinates. So to fill the right hand side, we first compute the Laplacians of the source, which is a convolution with the following stencil

\[
\begin{bmatrix}
1 \\
1 & -4 & 1 \\
1
\end{bmatrix}.
\]

These Laplacians are now rotated with the previously computed rotations. As we are solving for the squared Laplacians, we need to apply once more the discrete Laplace operator from above. From a mathematical standpoint only the Laplacians within the brushing are of interest. However as we are using a multigrid solver, the interpolation of the coarser levels is much easier when the Laplacians outside of the region of interest are clamped to zero.

### 3.3.3. Cloning

Now everything is precomputed to solve the equation (3.6): we have the right hand side with the rotated Laplacians of the source, the target (used for the constraints and as an estimation of the left hand side) and also the definition of the region of interest \( Q \).

As solving this equation in real time is crucial for the success of our approach, we developed a specially tailored multigrid solver. The design of such a solver is not straightforward at all, and especially the boundary conditions with arbitrary shapes, required additional treatment. So we don’t go into any details at this point and refer to the in depth explanations of those issues and the implementation of the solver, which can be found in Chapter 6.

### 3.3.4. Stitching

The final step in the algorithm is the embedding of the cloned mesh in the target. We do not remesh the cloned patch, but reuse the given triangulations. So first we define \( R_{\text{in}} \) as the set of
vertices of the source mesh that lie inside the region of interest \( \{ v \in C_s | P_s(v) \in Q \} \) and \( R_{out} \) as the set of vertices of the target that lie outside the ROI \( \{ v \in C_t | P_t(v) \notin Q \} \).

The result from the previous step is a grid of geometric positions \( \tilde{x} \). So we can now construct a mesh \( \tilde{R}_{in} \) with the same topology as \( R_{in} \), but the vertices \( v \in R_{in} \) are updated with their corresponding positions \( \tilde{x}(P_s(v)) \). \( \tilde{R}_{in} \) fits into the hole left open by all the discarded vertices of canvas \( C_t \) that are not part of \( R_{out} \) (Figure 3.6).

![Figure 3.6](image)

**Figure 3.6:** The mesh patch \( \tilde{R}_{in} \) (yellow) aligned to \( R_{out} \) (blue). The gap between the two meshes needs to be filled with a stitch mesh.

As a last step we need to fill the gap between \( R_{out} \) and \( \tilde{R}_{in} \). For this we construct a third mesh \( R_{gap} \), which we call the stitch mesh. We compute a constrained Delaunay triangulation in the parameter space, with \( P_t(R_{out}) \) and \( P_t(R_{in}) \) as boundary constraints. This triangulation is then mapped with \( \tilde{x} \) from parameter space to \( \mathbb{R}^3 \). As we can see in (Figure 3.7) the stitched and cloned meshes fit perfectly into the target mesh. Due to the derivative constraints in the solver no further smoothing is needed to achieve \( C^1 \) continuity.

### 3.3.5. Alternative Mode: Layering

We want to introduce here a small but powerful variation of the above algorithm. Previously we assumed to replace a part of the target mesh. Except for the boundary constraints the interior of \( C_t \) has no influence on the outcome. This is reasonable when the user intends to copy some large scale features or replace an existing feature. However for small scale structures it would make sense to enhance the existing surface instead of replacing it.

This approach could then be seen as a layering of geometry. Fortunately the existing formulation of the problem in equation (3.6) only needs a small adaptation:

\[
\Delta^2 \tilde{x} = \Delta(\delta_t + R\delta_s), \quad s.t. \quad \tilde{x}|_{\partial Q} = \mathbf{v}_t, \quad \frac{\partial \tilde{x}}{\partial \mathbf{n}}|_{\partial Q} = \frac{\partial \mathbf{v}_t}{\partial \mathbf{n}}.
\]

The right hand side becomes a summation of the target Laplacians and the rotated source Lapla-
3. Overview

(a) The triangulation of gap between \( \hat{R}_{in} \) and \( R_{out} \) is computed in the parameter space in \( \mathbb{R}^2 \).

(b) The stitch mesh from (a) perfectly fills the gap resulting in a mesh with \( C^1 \) continuity.

Figure 3.7.

icians, while the derivative constraints stay the same and ensure a smooth embedding of the details from source into target. For a better understanding we can reformulate the \( \hat{x} \) as \( x^* + v_t \), which is the sum of the existing target mesh and an additional layer \( x^* \). So we may now solve for \( x^* \) and later add \( v_t \) to get the resulting geometric positions \( \hat{x} \). The above equation becomes:

\[
\Delta^2 x^* = \Delta(R\delta_s), \quad s.t. \quad x^*|_{\partial Q} = 0, \quad \frac{\partial x^*}{\partial n}\bigg|_{\partial Q} = 0.
\] (3.8)

The equivalency of the two equations above can be shown with the same argumentation we use in the Section 6.1.1 where we explain the computation of the residual. So we omit further proof here.

The influence of this variation is visualized in Figure 3.8. The layering approach preserves both, the underlying edge as well as the added geometric detail, while the base algorithm loses the information about the underlying structure resulting in an unsatisfying shape. Furthermore we can see in Figure 3.9 an example of multiple layers of geometry applied on top of each other.
3.3. Algorithm

**Figure 3.8.** With the standard approach the edge of the box gets lost (left). With the layering approach both the ‘A’ geometry as well as the edge are preserved (right).

**Figure 3.9.** First the back of the armadillo (left) is transferred to the side of the bunny (center). Finally some small scale features are applied on top of the previously computed surface patch (right).

Although equation (3.8) is less involved than equation (3.7), we still worked with equation (3.7) in our implementation. This has two reasons. First, switching between the standard and the layering approach is more natural, as no post solver summation needs to be introduced, and the boundary conditions stay the same. Second, and more importantly, this formulation allows the introduction of weighting factors for source and target Laplacians. This means we could introduce more sophisticated brushes that would e.g. allow softer transitions from source to target. Due to time constraints we leave the introduction of such brushes open for future work.
3. Overview
Parametrisation

The parametrisation $P$ is a mapping from the vertices $V$ of the Mesh $M$ in $\mathbb{R}^3$ to the parameter space $S$ in $\mathbb{R}^2$. As previously mentioned, the parametrisation is needed to establish a correspondence from source to target. But it is also the space we solve the Poisson equation in.

These different usages of the parametrisation lead to the following criteria for the parametrisation and for the algorithm computing it:

- $P_s(C_s)$ should be bijective, or in other words it should not overlap.
- The parametrisation should be isometric (length preserving) at the boundary of the ROI $\partial Q$, because in the Poisson solver the domain is constrained at $\partial Q$. So if there are distortions in the parametrisations of the corresponding loops $\partial \Omega_s \in C_s$ and $\partial \Omega_t \in C_t$ that map to $\partial Q$, the interior of $Q$ will be distorted as well, leading to unsatisfactory results.
- The mapping to the interior of $Q$ should be conformal, but as we show later this point is less important than the 2 previous ones.
- The parametrisation only needs to be computed locally for $C_s$.
- The computation of $P_s$ should be fast enough to enable real-time editing.
- The only fixed point is the seeding point. Together with a given orientation it is the only constraint.

Unfortunately these criteria are in many cases contradictory and may not be fulfilled altogether. So we first formulate the general parametrisation algorithm we are using, and then we look into several improvements and their influence on the above constraints.

At first we assume that the canvases $C_s$ are fixed, and compute the parametrisation for all $v \in C_s$. However, in our approach the canvas needs to grow with the user’s brushing. As the
4. Parametrisation

growing is closely linked to the parametrisation, we will discuss this below in Section 4.3.

4.1. DEM - Discrete Exponential Map

We use the Discrete Exponential Map algorithm defined in [Schmidt et al. 2006]. The idea of the DEM is based on the concept of the exponential map, which states that each point $q$ in the local neighbourhood of a point $p$ on some surface may be expressed by geodesic polar coordinates $(r_g, \theta_g)$, where $r_g$ is the geodesic distance from $p$ to $q$ and $\theta_g$ is the polar angle in the tangent space $T_p$. A geodesic is the shortest curve between two points on a surface. The geodesic polar coordinates may be expressed as normal coordinates $(u, v)$ with respect to an orthonormal basis $\{e_1, e_2\}$.

The goal of the DEM is now to approximate the normal coordinates. For this it assumes that the sum of the lengths of the edges of the shortest path between two points to be a good approximation for their geodesic distance. The shortest path in a mesh can be computed using Dijkstra’s algorithm. With small extensions to Dijkstra we get an algorithm (Algorithm 1) that computes the DEM.

\begin{algorithm}
\caption{Discrete Exponential Map algorithm}
\begin{algorithmic}
\State add seed point $p$ with $uv$ coordinates $(0, 0)$ to candidates
\While{candidates not empty}
\State $r \leftarrow$ element with smallest distance in candidates
\State pop $r$ from candidates
\State $r.uv \leftarrow r.uv/r.weight$
\State compute $Rot2D$ of $r$
\If{$r \in$ canvas $C$}
\For{$q \in$ neighbours of $r$}
\State $length \leftarrow ||r - q||$
\State $distance \leftarrow r.distance + distance$
\State $weight \leftarrow 1/length$
\State $q.uv \leftarrow q.uv + weight \ast (r.uv + Rot2D \ast Local_{uv}(r,q))$
\State $q.weight \leftarrow q.weight + weight$
\EndFor
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

Let us focus on the parts of the algorithm that differ from standard Dijkstra, the 2D matrix $Rot2D$ and the function $Local_{uv}$. $Local_{uv}(r,q)$ computes the normal coordinates of $q$ in the tangent space $T_r$ of the intermediate point $r$. However, as we are interested in the normal coordinates in the tangent space $T_p$ of the seed point $p$, we need to transform the coordinates from $T_r$ to $T_p$. This is done in 2 steps. First the coordinate is rotated by $Rot2D$ which is the alignment from $T_r$ to $T_p$. Finally the coordinate is translated by the coordinate of $r$. 

22
4.2. Optimisations

The weighting is there to make the algorithm more stable when used in regions of high curvature or on irregular meshes. Instead of using only the shortest path to estimate the uv-coordinate, but averages over the shortest paths. This weighting scheme is called upwind-averaging in [Schmidt 2010].

4.2. Optimisations

As shown in Figure 4.1 can be seen, the basic DEM does not fulfil all criteria described in the beginning of this chapter. Especially when the DEM is used on surfaces with high curvature, tearing artefacts occur. These artefacts are described in [Schmidt 2010] as geodesic discontinuities and occur because there are multiple "shortest paths" to those artefact regions.

Figure 4.1: Basic DEM without any optimisations shows significant artefacts, so called geodesic discontinuities

In the following we are evaluating different adaptations to the DEM algorithm.

4.2.1. Normal Smoothing

First we analyse the influence of normal smoothing. It has as well been applied in [Takayama et al. 2011] and proposed in [Schmidt 2010]. The use of normal smoothing is a natural way to solve the given problem, as the described artefacts occur around high curvature regions and smoothing the normals reduces the influence of these perturbations on the alignment of the tangent spaces.

Applying DEM to a mesh with smoothed normals, can be seen as a projection to the tangent space of an imaginary surface described by the smoothed normals. As we want to reduce the influence of the interior of the ROI to the parametrisation of the boundary, an approach as used
4. Parametrisation

In [Takayama et al. 2011] would probably be ideal. There they first smooth the normals on the boundary with their neighbouring boundary normals. Finally the normals in the interior of the ROI are computed by a simple weighted averaging of the boundary normals. However, this is not possible in our case, as the Canvas is not predetermined, but computed based on the outcome of the DEM algorithm (Section 4.3).

As the above is not applicable, we smooth the normals of the source and target mesh using a simple averaging over the normals of the one-ring neighbourhood. We apply several rounds of this normal smoothing. As this smoothing is independent of any user input, we precompute it upon loading of the meshes and thus it does not interfere with our real-time goal. The actual number of smoothing rounds may be determined by the user.

Our test showed that geodesic discontinuities do disappear when normal smoothing is applied, but only when a significant number of smoothing rounds are applied. (The actual number of rounds needed is dependant on the density of the mesh.) In Figure 4.2 we can see a successful application of normal smoothing, but it also shows that due to normal smoothing the mesh is overlapping in parameters space. This is a severe issue, as our approach relies on the bijectivity of the mapping and fails otherwise.

Figure 4.2: With 100 iterations of normal smoothing applied the geodesic discontinuities disappear, but at the cost of overlaps in the parameter space, which can be seen at the right end of the ear.

4.2.2. Frame-Propagation

In the standard DEM algorithm a vector \( T_r(q) \) in the tangent space \( T_r \), is directly aligned to the tangent space \( T_p \) by a transformation from its frame \( F_r \) to the initial frame \( F_p \). This dependency on a potentially far point with a normal that may differ a lot from the normal at point \( r \), may introduce robustness issues. In [Schmidt 2013] the authors proposed an adaptation which they called frame-propagation. Their idea is to propagate the frame via parallel transport, so that each point is only dependant on its one-ring neighbourhood.
As the normal is given at each vertex, only an additional basis vector $e_1$ is needed to compute the local frame. The remaining basis vector $e_2$ is given by orthonormality. So propagating the frame becomes a propagation of the first basis vector. For the seed point the initial basis vector is defined by the up vector defined for the parametrisation.

Let’s look at three points: the seeding point $p$, its neighbour $r$ and a point $q$ which is a neighbour of $r$ but not of $p$. We are now interested in the normal coordinates $T_p(q)$. $T_p(r)$ and $T_r(q)$ may obviously be computed directly. By propagating the frame $F_p$ to $F_r$ via parallel transport and using $F_r$ as a basis for $T_r(q)$, $T_p(q)$ becomes the sum $T_p(r) + T_r(q)$.

In Algorithm 2 we show in red the difference in the algorithm between the original DEM and frame-propagation DEM. The Rotation function describes the minimal rotation around the $n_p \times n_r$, which is exactly the parallel transport from $F_p$ to $F_r$. The function computing the uv coordinates is basically the same with the only difference that the tangent space is now aligned according to the propagated basis vector.

Algorithm 2 Frame-Propagation DEM

```plaintext
add seed point $p$ with uv coordinates $(0, 0)$ to candidates
while candidates not empty do
  $r \leftarrow$ element with smallest distance in candidates
  pop $r$ from candidates
  $r.uv \leftarrow r.uv / r.weight$
  $r.basis_vector \leftarrow r.basis_vector / r.weight$
  if $r \in$ canvas $C$ then
    for $q \in$ neighbours of $r$ do
      $length \leftarrow \|r - q\|
      $distance \leftarrow q.distance + distance
      $weight \leftarrow 1/length
      $q.uv \leftarrow q.uv + weight * (r.uv + Local_uw(r, q))$
      $q.basis_vector \leftarrow q.basis_vector + weight * (Rotation(r, q) * r.basis_vector)$
      $q.weight \leftarrow q.weight + weight$
      update $q$ in candidates with distance
    end for
  end if
end while
```

The quality of this approach highly depends on the path taken. So we use again upwind averaging, as we already did in the standard DEM when computing the coordinates.

When comparing these two approaches (Figure 4.3) we see one major drawback; while the DEM often recovers from geodesic discontinuities after a certain distance to the geometric detail, the frame-propagation DEM tends to produce even stronger geometric discontinuities without recovering from them.

The benefit of frame-propagation DEM can be seen when applied to larger regions. When the normal at the boundary of a large canvas differs a lot from the initial seeding normal, original
4. Parametrisation

Figure 4.3.: On the left it can clearly be seen that frame-propagation DEM tends to increase the geodesic discontinuities. However in combination with the same amount of normal smoothing as in Figure 4.2 frame-propagation DEM shows similar results (right).

DEM becomes unstable and tends to produce unexpected results due to overlaps and inaccurate alignments (Figure 4.5). The frame-propagation DEM on the other hand stays stable, even when used on regions with large changes in the normal field (Figure 4.4).

Figure 4.4.: The complex canvas is both on top and also on the bottom of the lizard. While the original DEM would fail to parametrise such a patch, the frame-propagation DEM is still stable.

As a conclusion we can state that frame-propagation DEM may be prioritized over the original DEM, but only when used in connection with normal smoothing, otherwise the influence of the geodesic discontinuities becomes too prominent.
4.2. Optimisations

In a similar setting as in Figure 4.4 the original DEM becomes unstable. This results in overlaps in the parameter space and an undesired growing of the canvas. (The actual computation of the cloned surface is omitted in this example for a better visualisation of the failed parametrisation.)

4.2.3. Overlap Correction

The combination of the two above optimizations leads the one major drawback: even small geometric details tend to produce overlaps in the parameter space. At best such overlaps lead to inaccurate replication of the copied geometry, but very often the algorithm fails to produce any meaningful result, as both the solver of the Poisson’s equation as well as the stitching rely on the bijectivity of the parametrisation.

To address this issue we want to take a look at a method briefly mentioned in [Schmidt et al. 2006]. There the authors introduce a hybrid approach where they combine the DEM method with natural conformal mapping [Desbrun et al. 2002]. The DEM works as a starting point for the parametrisations, but is discarded in regions with high curvature. In those regions they then apply conformal mapping with the coordinates computed with DEM as constraints at the boundary of the regions.

The conformal mapping is computed by solving a sparse linear system which minimizes a conformal energy formulation. The advantage of the conformal mapping over the DEM is that it ensures a bijective mapping. However, the computational cost is higher for the conformal mapping, because a linear system needs to be solved. So the hybrid mode gets the best out of both approaches. In general the DEM is used for a fast parametrisation, while the conformal mapping is only used to ensure bijectivity in small regions.

In our case the criteria whether a part of the canvas should be recomputed or not are slightly different than in [Schmidt et al. 2006]. First we only consider parts of the mesh that fully lie inside the ROI. Furthermore we only apply the hybrid approach to regions with actual overlaps; high curvature is not a sufficient criteria in our case. On the one hand this reduces the amount of vertices to process and thus the computation time. On the other hand the boundary of the overlap regions is clearly defined, while the definition of a high curvature region is dependant on the measurement of the curvature and a threshold that would need to be found.
4. Parametrisation

In Figure 4.6 we can see the influence of the overlap correction. When only the DEM is applied, we can see severe artefacts. With the overlap correction applied the artefacts disappear and the geometry is correctly copied.

![Source mesh](a) Source mesh  ![Cloned mesh without overlap](b) Cloned mesh without overlap  ![Cloned mesh with overlap correction](c) Cloned mesh with overlap correction

**Figure 4.6:** Due to the overlaps introduced by the DEM algorithm, artefacts occur when copying the mouth of the Homer Simpson model (a) to the target (b). When the overlap correction is applied, the parametrisation becomes bijective and the result is artefact-free (c).

As the computational cost of the hybrid approach is highly dependant on the size of the overlap region and we want to avoid performance drops due to too large overlap regions, we implemented the overlap correction as a post-processing step. This is also helpful as the conformal mapping should only be computed when the overlap region is fully inside the ROI, which would make an incremental updated more difficult. So the user needs to invoke the overlap correction manually.

4.3. Canvas Growing

In this section we show how the canvas is computed. As a recap, we defined the canvas $C_s$ as the part of the mesh that gets parametrized and takes part in the cloning operation. In contrast to the original GeoBrush our software has no fixed canvas, so we potentially need to grow the canvas every time the user applies the brush.

We start with the criteria we have to consider when constructing the canvas. First the parametrisation of the canvas should obviously include the region of interest $P_s(C_s) \supset Q$. This criterion is sufficient for the source canvas $C_s$ but not for the target canvas $C_t$, because the region surrounding the ROI $P_t(C_t) \notin Q$ needs to have some width as well. On the one hand we need to ensure a big enough boundary $\partial Q$ that the solver may enforce the Neumann boundary conditions (i.e., each grid cell in $Q$ needs to have a valid 5x5 area surrounding it, see Section 6.2.2). On the other hand the area needs to be big enough to ensure that $R_{out}$ is a full circle with no holes in it, so that a valid triangulation for the stitch mesh may be computed. Thus the additional size needed for $C_t$ depends on the grid spacing in parameter space, but also on the triangulation of the target mesh.
As the above criteria regarding the extent of the canvases are all lower boundaries, one could argue to just increase the canvas by a huge amount to be on the safe side. However besides increasing the computational costs as more triangles need to be considered in the further steps, a too big canvas may also lead to severe artefacts. Such artefacts may be introduced due to overlaps in the parameter space or geodesic discontinuities, as can be seen in Figure 4.5.

Looking at the issues in Figure 4.5 an apparent solution would be to stop growing the parametrisation on high curvature regions, as they introduce the erroneous behaviour. We first also worked in that direction, but a reliable measurement for the curvature as well as a decent corresponding threshold are hard to find and would be highly dependant on the given mesh.

So we introduce an additional parameter space texture. As with the paint texture mentioned earlier, each time the user applies the brush a disk is drawn onto that texture. This texture differs from the paint texture in two notable ways. First the radius of the disk has an increased size, to ensure that a 5x5 neighbourhood of each grid point in $Q$ is included. Second, we do only enlarge the area but never shrink it. This gives a more stable parametrisation and its incremental update is simplified (see Section 4.4). It also allows the copying of unconnected regions (Figure 4.7).

Finally the extent of the canvas is computed in parallel with the parametrisation. We start at some seed point and grow the canvas until we reach points whose parametrisation is not within the brushed area defined by the above texture. To ensure a proper triangulation, we continue growing to include the 2-ring neighbourhood of those points.

### 4.4. Incremental Update

Up to now we always considered the parametrisation algorithm to be a one-time operation. However as the brushing is applied step by step; an incremental update would be beneficial. As
4. Parametrisation

mentioned earlier, the DEM algorithm is based on Dijkstra’s algorithm. Dijkstra itself may not
be incrementally applied. However there exist algorithms (e.g. the D*-Algorithm [Stentz and
Mellon 1993]) that support incremental updates and that may adapt to changed environments.
The adaptation of such algorithms to our case though is non-trivial, as we not only need the
shortest path but a weighted average over the shortest paths (see upwind-averaging in Algorithm
1).

The main performance bottleneck is the actual computation of the DEM, which depends on the
size of the canvas. We measured a computation time of 40 to 50ms for a canvas with 10000
vertices. An additional performance issue is the uploading of the data to the GPU. We are
working with Vertex Buffer Objects (VBO), which are holding the vertex information in the
GPU memory. We want to update the data in the VBOs only when it actually changed, because
the uploading to VBOs is a lot more expensive than the data rendering from them.

So the goal of this adaptation of the DEM algorithm is to avoid any unnecessary computations.
To achieve that, we subdivide the task into 2 steps. In each frame we check whether the canvas
needs to grow and if so, we update the parametrisation locally. As long as the user continues
with brushing we continue with that first step. In the first frame with no user interaction, we
continue with step two. In this second step we recompute the parametrisation for the full canvas,
but only update the vertices for which the path became shorter. This policy is valid, due to the
restriction that the canvas never shrinks, as we have stated in the previous section.

For the first step we keep track of the boundary of the canvas. So we only need to check in the
neighbourhood of the boundary for points to be included in the canvas. The local computation
of the parametrisation is then computed with that boundary as seed. By increasing the size of
the canvas, the shortest path to previously computed points may change as well, e.g. when the
user brushes a circle. Because of that we need the second step, that fixes the parametrisation in
for all vertices.
Laplacian Coordinate Alignment

As already previously mentioned, Laplacian coordinates are not rotation invariant. This means that for our approach, we need to align the source and the target mesh before solving the linear system for the global coordinates of the cloned patch. In this chapter we focus on this alignment and talk about our solution to the problem, as well as critical points that have to be considered for a successful alignment.

5.1. Naive Approach

The setting is as follows. To compute a rotation from source frame to target frame at a specific vertex $v_s$ on the surface of source canvas $C_s$, we need to estimate the local frame at that point, as well as the local frame at the correspondent vertex $v_t$ in the target canvas $C_t$. However, as there is no direct vertex to vertex correspondence, we would need to interpolate such a vertex in $C_t$. To avoid the need of search structures to interpolate points on $C_t$, we compute the rotations in 2 steps. For all the source vertices we compute the rotation from the local to the global frame. For the vertices in the target canvas we compute the opposite rotation from the global to the local frame. These rotations are then encoded as quaternions and rendered to the parameter space as textures. As we have a correspondence from source to target in the parameter space, we may now compute the final rotation with quaternion multiplication.

In the naive approach we estimate the local frame using the vertex normal and the up vector of the parametrisation. The third vector of the local frame follows by orthogonality.

This approach is very sensitive to noise in the normals and also tend to produce bad results with complex geometries, as we can see in Figure 5.1. Such artefacts occur because the field of rotations is with that approach very noisy and tends to have discontinuities.
5. Laplacian Coordinate Alignment

![Figure 5.1.](image1)  
(a) Source mesh  
(b) Cloned mesh with naive alignment approach. The blue and red lines denote the rotated frame, blue being the normal.

**Figure 5.1.** The face of the Homer Simpson model (a) is copied to a sphere (b). The artefact at the nose shows the disadvantage of this naive approach.

5.2. Rotation Interpolation

The naive approach fails, because the normals are way too noisy for a stable estimation of the rotations. So our initial idea to improve the naive approach was to fix the rotations at the boundary and to interpolate the rotations in the interior of the ROI. This interpolation is done by diffusing the quaternions in parameter space. We compute this diffusion, with same solver we are using to solve for the Laplacian coordinates.

![Figure 5.2.](image2)

**Figure 5.2.** The same example as in Figure 5.1, but now with interpolated rotations.
5.3. Normal Smoothing

In Figure 5.2 we can see the improvement compared to the naive approach, as discontinuities in the normal field got removed. However, the interpolations using the boundary normals do often not estimate the rotations well, resulting in undesired deformations.

5.3. Normal Smoothing

As we have seen in the previous section, the interpolation approach does still produce unsatisfactory results. Furthermore, our approach is already very dependant on the boundary normals and so we do not want to increase that dependency anymore. So the final approach we are using is to smooth the normals and then to use those smoothed normals to estimate the local frames and compute the rotations, as in the naive approach. As we are already smoothing the vertex normals to compute the parametrisations, no further computations are needed.

Figure 5.3: Normals smoothing solves the problems of the previous examples. The rotations of the frames are more stable and there is no strong dependency on the boundary.

The alignment of the Laplacian coordinates with rotations based on smoothed normals could be interpreted as the warping of an imaginary smooth source surface to another imaginary smooth target surface. The smoothed normals would then be the normals of those imaginary surfaces. The result of this approach can be seen in Figure 5.3 where the copied geometry is free of artefacts and the estimated local frames (denoted by the blue and red lines) are stable.
5. Laplacian Coordinate Alignment
Multigrid Poisson Solver

As previously discussed (see Section 3.3), it is crucial for our approach, to have a fast and robust solver for equation (3.6). As we want to focus on the solving of the equation, we slightly simplify the notation here. The right hand side is at this stage already precomputed, so we may replace it by a function \( f = \Delta R \delta_{src} \). And we also represent the squared Laplacian-operator \( \Delta^2 \) by the matrix \( A \). So (3.6) becomes:

\[
Au = f
\]

(6.1)

with the same boundary constraints as before. However, as the boundary conditions and especially their non-fixed interface are non-trivial, we will take care of them in a later section (Section 6.2).

6.1. The Multigrid Approach

We will solve our equation using the multigrid method. The multigrid method is a recursive method, where first a lower frequency version of the problem is solved on a coarser grid and then higher frequency details are added in the finer grid. The advantage of this method is, that it runs in \( O(n) \), as well as the possibility to run the computations of the different grid cells in parallel.

In the following we state the general outline of the multigrid method and focus later on our choices and optimizations. For a more in-depth study of multigrid methods we refer to [Briggs et al. 2000] (General overview and study of different choices to be made) and [Press et al. 1992] (sample C implementation). As the nomenclature in literature differs a lot, we stick here to the
6. Multigrid Poisson Solver

denotations in [Briggs et al. 2000].

Let us start with the algorithm in pseudo-code. The name VCycle comes from the fact, that there is one recursive call at each level to solve the coarser problem. This is in contrast to a WCycle where at each level the coarser level is called twice.

### Algorithm 3 $u^h = \text{VCycle}(v^h, f^h, h)$

```plaintext
if $h = Lh$ then
    Solve $A v^h = f^h$
else
    Relax $A v^h = f^h$ for $v_1$ times \{pre smoothing\}
    $v^h \leftarrow f^h - A v^h$ \{compute residual\}
    $f^H \leftarrow I^H_h r^h$ \{restrict\}
    $v^H \leftarrow 0$
    $u^H \leftarrow \text{VCycle}(v^H, f^H, H)$ \{Recursive call to VCycle\}
    $u^h \leftarrow v^h + I^H_h u^H$ \{prolong/Correct\}
    Relax $A u^h = f^h$ for $v_2$ times \{post smoothing\}
end if
```

$h$ is the grid spacing in the finer grid, while $H$ is the grid spacing in the coarser. In our case the coarser grid has always half the size of the finer one, so $H = 2h$.

#### 6.1.1. Residual

The idea of this algorithm is to start with an initial guess $v$ and refine it with each iteration. To enable this, we solve for the residual $r = f - Av$. When $v$ is the initial guess and $u$ the actual solution, then $e = v - u$ is the error. So solving $A e = r = f - Av$ with $e = 0$ as an initial guess, is equivalent to solve $Au = f$ with an initial guess of $u = v$. Working with a residual enables not only iterative improvement of the solution, but also provides better numerical stability. Furthermore it simplifies the handling of the (derivative) boundary constraints, as we will see in Section 6.2.2.

#### 6.1.2. Restriction

Once the residual is computed, we need to bring it to the coarser level. In literature $f^H = I^H_h f^h$ is called restriction. The simplest restriction operator $I^H_h$ is the injection operator, which is a simple subsampling $f^H_{i,j} = f^h_{2i,2j}$. We use the full weighting operator, which convolutes $f^h$ with a smoothing kernel

$$
\begin{bmatrix}
\frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\
\frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\
\frac{1}{16} & \frac{1}{8} & \frac{1}{16}
\end{bmatrix}
$$

before subsampling. The smoothing kernel may be implemented on the GPU as only 4 texture lookups.
6.1. The Multigrid Approach

We use an additional factor of 16, following [McCann and Pollard 2008] and their argumentation that this is beneficial when working with floating point numbers on the GPU. Normally in literature this factor is built in the matrix $A_h$ as a compensation for the grid spacing. The factor may be explained with finite differences. The denominator in finite differences is $h^n$, where $n$ is the order of the finite difference. Thus changing the grid spacing from $h$ to $H = 2h$ must be compensated with $2^n$, which is 16 in our case (solving a 4th-order problem) and 4 in [McCann and Pollard 2008].

6.1.3. Correction

After VCycle has been called recursively, we need to prolong the values from the coarser grid to the finer. The prolongation may be seen as the inverse of the restriction operation. We use bilinear interpolation as the prolongation operator $I^h_H$, which can be implemented on the GPU as a single texture lookup. As we solved on the coarser level for the residual, we need now to correct the initial guess $v^h$ by the prolonged values $I^h_H u^H$, thus this step is called correction.

6.1.4. Relaxation

The relaxation or smoothing step is the same as it is in basic iterative methods, except that in our case the number of iterations is smaller and does not grow with the size of the grid. Any iterative method could be used in the smoothing step. We use red-black Gauss-Seidel as proposed in [Press et al. 1992]. In a red-black method the grid points are divided in even (red) and odd (black) points (see Figure 6.1) and the update is then in turn applied, ones only to the red and ones only to the black points. This leads to faster convergence and better numerical stability.

![Figure 6.1:](image)

*Figure 6.1:* In red-black Gauss-Seidel the relaxation of even and odd grid points is alternated. Thus no direct neighbours are updated in the same step.*
6. Multigrid Poisson Solver

When looking at the discrete squared Laplace-operator:

\[
\Delta^2 = \begin{bmatrix}
1 & & & & \\
2 & -8 & 2 & & \\
1 & -8 & 20 & -8 & 1 \\
2 & -8 & 2 & & \\
1 & & & & 
\end{bmatrix}
\]

we see that neighbouring points are weighted with opposite signs. Using normal Gauss-Seidel with this smoothing kernel often leads to oscillatory behaviour. The alternative to red-black Gauss-Seidel would be to use damping or the weighted Jacobi method. This removes the stability issues as well, but showed to have slower convergence.

6.1.5. Termination

In Algorithm 3 \( Lh \) is used as the final grid spacing to terminate the VCycle recursion. Once this grid spacing is reached, the remaining linear system has to be solved conventionally (e.g. by using some factorization technique). As exact solvers on the GPU are non-trivial, we have two choices for this step. Either dump the values from the GPU and solve the remaining system on the CPU or choose \( Lh \) so big that the solution of the remaining system becomes trivial. We tried both, and the differences are minuscule. Although the performance of the CPU solver is faster on small grids, the copying of the data takes most of this benefit away. A disadvantage of using the CPU solver lies in the maintenance of the solver, as each change needs to be applied in both solvers. We still decided to stick with the CPU solver, as it tends to be a bit more accurate and is numerically more stable. (GPU floating point operations tend to be less exact, which may become an issue as in each restriction step, the values need to be scaled.)

As CPU solver we use the UMFPACK (unsymmetric multifrontal sparse LU factorization) package [Davis 1994-2011].

6.2. Boundary Conditions

The enforcement of the boundary conditions deviates in our case from standard literature. This is mainly because the interface (the shape of the ROI boundary) is constantly changing and could take any form. Secondly we need a combination of Dirichlet and Neumann boundary conditions.

6.2.1. Dirichlet Boundary Conditions

Let us first focus on the Dirichlet boundary conditions. The following explanations are based on Chapter 4.8 of [Hackbusch 2005]. As the shape of the interface is arbitrary and may change
with each frame, we first need to introduce a mask (Figure 6.2) marking grid points as interior or boundary points. In the relaxation step, only the interior points are updated, while the boundary points are kept constant.

**Figure 6.2:** The thick line denotes the boundary of the ROI. Interior points are then further subdivided into near boundary points ◦ and other interior points ●.

However, the real challenge in handling the Dirichlet boundary conditions arises in the restriction step. As can be seen in Figure 6.3, the actual boundary does not match with the coarse grid. So in the restriction step we keep track of the distances to the actual boundary for the interior points.

In the relaxation step we use an adapted scheme that weights the different neighbours according to their distances instead of common finite differences. The weighting scheme, named Shortley-Weller, can be derived using divided differences, which we did in the appendix (Section B.1).

As we are always solving for the residual \( r \), we fortunately only need to track the distances to the boundary and do not need to interpolate the actual values at the boundary, because \( r \) is - by definition - always zero at the boundary.

To visualize the influence of the Shortley-Weller correction, we plotted in Figure 6.4 the residual (as a measurement of the error) after 2 VCycle iterations, once with and once without the correction. When Shortley-Weller is not applied, the error is accumulated at the boundary. This does not only reduce the pace of convergence, but can even lead to instability in some cases, because in later iterations the error may get overcorrected.

### 6.2.2. Neumann Boundary Conditions

In the GeoBrush paper [Takayama et al. 2011] the Neumann boundary constraints are enforced by solving the system in a 2-way fashion, by relaxing alternately for the derivative and the posi-
6. Multigrid Poisson Solver

Figure 6.3: The grid from Figure 6.2 is now restricted to a coarser grid. The boundary (thick line) is not on the grid at all positions anymore. In the restriction step we need to compensate for near boundary points (○) by keeping track of the distances to their neighbouring boundary points (×).

tions. So for them it was possible to enforce the derivatives explicitly, by keeping the derivatives at the boundary constant. We instead directly solve equation (3.6), which results in using a 5x5 finite difference stencil (the discretised squared Laplace-operator). So to ensure correct derivatives on the boundary, each interior grid point needs a defined 2-ring neighbourhood. On the initial grid this can be easily ensured. However when restricting to a coarser level the 2-ring neighbourhood becomes larger and larger, until at some point the 2-ring neighbourhood would be outside of the defined parameter space Ω. Fortunately, as described above with the Dirichlet boundary conditions, the residual r is always zero at and beyond the boundary. Thus we can safely ensure a 2-ring neighbourhood on coarser grids by clamping outside values to zero.

Of course the issues discussed in the previous section need to be considered as well. Especially the Shortley-Weller scheme needs to be adapted to the higher order stencil, which is discussed in Section B.1.
6.2. Boundary Conditions

Figure 6.4: We can see the influence of the Shortley-Weller approximation in these plots of the residual of a sample problem with a rectangular ROI. In (a) the error at the boundary of the ROI is much bigger than in (b). Thus not using Shortley-Weller leads to slower convergence and may even lead to no convergence at all due to overcorrection.
6. Multigrid Poisson Solver
Results

In this chapter we give an overview of the capabilities and limitations of our approach. We furthermore compare it to existing methods. In the previous sections several figures already showed some applications of our work. We show the influence of the different steps of our approach and how they may be applied and combined.

7.1. Performance

We tested the performance of our system on a machine with a 2.4GHz Intel Core 2 Duo CPU with 4 GB RAM and a NVIDIA GForce 320M GPU. We tested the system with several meshes. The measured times are with the lizard meshes (Figure 7.5). The source model has 34519 vertices and 69304 faces, while the target mesh has 19386 vertices and 38768 faces. All times are measured using the standard CPU clock counter, whose accuracy is CPU dependant and in the milliseconds. We give with the numbers a comparison between the different steps and methods, as well as a general magnitude of the performance.

7.1.1. Solver Performance

First we compare the performance of our GPU solver with the CPU solver (Table 7.1). We are using the UMFPACK (unsymmetric multifrontal sparse LU factorization) package [Davis 1994-2011]. The times measured contain the smoothing of the rotations for the Laplacian coordinate alignment, the computation of the Laplacians, the solving of the linear system as well as the dumping of the data back to the CPU memory. We compare the solver performance for different grid sizes $N \times N$. 
7. Results

<table>
<thead>
<tr>
<th>N</th>
<th>#Grid Cells</th>
<th>GPU Time [ms]</th>
<th>CPU Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>1024</td>
<td>40-50</td>
<td>50-60</td>
</tr>
<tr>
<td>64</td>
<td>4096</td>
<td>50</td>
<td>130</td>
</tr>
<tr>
<td>128</td>
<td>16384</td>
<td>70-80</td>
<td>500</td>
</tr>
<tr>
<td>256</td>
<td>65536</td>
<td>120-130</td>
<td>3400</td>
</tr>
<tr>
<td>512</td>
<td>262144</td>
<td>300-320</td>
<td>43000</td>
</tr>
</tbody>
</table>

*Table 7.1.: Solver performance*

We can clearly see the linear performance in the number of grid cells of our GPU solver, while the CPU solver performance drops rapidly. Although we have to state that the CPU solver is by no means optimized. The framework has been built with the GPU solver in mind and the CPU solver has been built as comparison. For easier implementation all the pre- and post-computation steps are the same for both solvers.

For all our tests we also measured the normalized root mean square error (NRMSE), as measurement of the accuracy of the GPU solver. The error depends on the number of smoothing steps in each cycle. We chose a single vcycle iteration with 10 red and 10 black smoothing steps. With this setting the NRMSE is between 0.001 and 0.004 in typical use cases. We consider this error acceptable for a real-time environment, especially as other parameters like the amount of normal smoothing have a much larger influence on the outcome.

Grid sizes of $128 \times 128$ to $256 \times 256$ have proven to be optimal: The response time is low enough for an acceptable real-time feedback, while the resolution was high enough for our examples. If higher grid sizes are required we would propose to use a lower resolution while brushing to give the user immediate feedback. Once the brushing stroke is finished the result could be refined with a higher resolution grid.

7.1.2. Overall Performance

Although the solver is the main performance bottleneck, we need also to consider the other steps to get a complete picture of the performance of our approach. For a better performance we distinguish between intermediate updates while the user is brushing and a full update once the brushing stroke is finished. In the intermediate update the parametrisation is only computed locally, while it is recomputed from scratch in a full update.

In Table 7.2 we listed the times needed for the different steps. The paint update step consists of the drawing of the brush in the paint texture and a potential resizing of the paint texture when the brush comes close to the boundary of that region. The time needed for that step depends on the grid size, but is minuscule as the time needed for the GPU solver grows faster with the grid size.

The other three steps, parametrisation, ROI boundary extraction and mesh stitching are dependent on the triangulation of the mesh. For the parametrisation we distinguish between a full and
a partial update; for more information about the difference of those steps see Section 4.4.

Our tests have shown that a performance of roughly 100 ms per frame is enough to get a smooth real-time brushing experience. Of course the eye still grasps the different frames as single images, though.

In our work not only the brushing state but also the other steps, i.e., the repositioning and rescaling of the canvases, are interactive. Actually, the steps needed for such a change to a canvas, are the same as in a full update, but because no information may be reused, it takes a bit longer, around 300 ms.

### 7.2. Derivative Constraints

Until now we have always assumed that the derivatives at the boundary are constrained. However there is also the possibility of only applying positional constraints. We show here the influence of the different constraints, as well as possible applications for both approaches.

In Figure 7.1 we see that the derivative constraints have a coating effect: The geometric details are peeled from the underlying surface and only they are transferred to the target geometry. We assume that the membrane surface constrained by the boundary vertices and normals represents the underlying smooth surface well.

A problem that comes with using our approach with derivative constraints, is the missing locality. So especially when the boundary of the source or target domain is not smooth, small changes to the brushed region can totally alter the result and even lead to unpleasant artefacts. In Figure 7.2 we see such a case, where the region around the moustache is very curvy. In such a case the interior is changing with each brush stroke drastically, making it hard or almost impossible to achieve desired results. We propose to drop in such cases the derivative constraints and smooth the boundary of the ROI after the cloning took place.
7. Results

Figure 7.1.: In this example we see that with derivative constraints only the coating of the surface is transferred (c). The approach with only positional constraints (b), on the other hand, also copies the spherical part of the source mesh (a).

Figure 7.2.: Here we try to transplant the moustache of the Neptune mesh (a) to the Mannequin model. While the approach without derivatives (b) is successful, the one with derivative constraints (c) fails, because the region around the moustache in the Neptune model is too curvy.

7.3. Comparison to GeoBrush

As our work is based on GeoBrush, we give a side by side comparison between their and our work.

In Figure 1.1 we showed an example where the cage generation in GeoBrush fails. Our method on the other hand succeeds in copying the same geometry (Figure 7.3).

A major advantage of GeoBrush is the ability to be able to copy overlapping regions and even handles, with a genus bigger than zero. In Figure 7.4 we see such an example, where GeoBrush succeeds to clone the handle, while our method fails because of an overlap in the parameter space and the inability of our solver to handle such an overlap.

However, the biggest difference of our method from GeoBrush lies in the easier usability. Due
7.3. Comparison to GeoBrush

![Figure 7.3](image1.png)

**Figure 7.3.** The same setup as in Figure 1.1, but our approach can successfully copy the beard of the Neptune model (b) to the Homer Simpson model (b).

![Figure 7.4](image2.png)

**Figure 7.4.** The handle from (a) is copied well in GeoBrush (b), while our approach fails (c).

to the cage based approach the user is restricted to a predefined canvas. As the source and target mesh may differ a lot, defining a proper cage is often difficult. So the user has to define rather small cages and apply the method several times to copy geometry over larger surfaces. With our method on the other hand the user is not restricted to any canvas, making it possible to brush over substantial amounts of a mesh without the need of reseeding. To show this, we used the same lizard model, that was also prominently used in the GeoBrush paper (Figure 7.5). Besides the better usability, the brushing over larger regions also tends to give better results, because it is often hard to properly align to cloned patches without some visual artefacts.
7. Results

(a) Without any restrictions on the canvas, it is possible to draw over larger areas of mesh, without the need of reseeding.

(b) This can be done in our approach in one step with very little brushing strokes. A similar result in GeoBrush would need several steps, as the placement of a canvas in such small scale regions tend to fail.

Figure 7.5.

7.4. Laplacian Coordinate Alignment

An important part of our work is the alignment of the Laplacian coordinates form source to target, because the Laplacian coordinates are not rotation invariant. In Figure 7.6 we can see the influence of the alignment; the geometric details adapt well to the different underlying surfaces.
7.5. Limitations

(a) Source mesh  
(b) Concave target mesh  
(c) Convex target mesh

Figure 7.6.: Due to the Laplacian coordinate alignment the source geometry (a) can be copied to concave (b) as well as convex regions (c).

We introduced two different alignment approaches. In our tests, both proved to be working well in many cases. Which one performs better is highly dependant on the setting; the rotation interpolation produces better results when the interior has high curvature and prominent features, while the normal smoothing approach tends to be better when the boundary is curvy or the interpolated rotations do not match with the underlying smooth surface.

7.5. Limitations

The biggest limitation of our approach is that it is not able to copy geometries with complex topology, especially geometries with a genus bigger than zero (e.g. toroidal structures). This limitation is introduced by the solver, which computes its solution in the parameter space on a rectangular domain. Thus we would need to generate a bijective mapping from a surface with genus > 0 to a rectangular domain, which is impossible. The other steps of our algorithm would not require a bijective mapping on the whole domain, but only on the boundary. So this limitation could be lifted by a solver that works in \( \mathbb{R}^3 \).
7. Results
Conclusion and Future Work

8.1. Conclusion

We have established a technique for successful brush-based cloning of 3D geometry in a real-time environment. Our work is the first that allows brushing without restriction to a predefined canvas. This gives the user a lot more freedom and ease of use than previous works do.

Besides the standard algorithm we introduced a second mode. The layering approach gives the user the possibility to apply several layer of geometric details without losing the surface information of the underlying target mesh. Our method is the first that combines the notion of brushing and the possibility to have several layers of geometry.

Our work is also the first of its kind that allows interactive placement and scaling of both the source and the target canvas, giving the user again more possibilities to achieve a satisfying result.

8.2. Future Work

Although we have shown that our approach works and produces satisfying results, there are still several issues that may be attacked to improve the usability of our work. We want to give here a list of issues that arose during our work and briefly outline how they could improve the current state.
8. Conclusion and Future Work

Handles

A major limitation of our work is the inability to transfer structures with overlaps and handles. This limitation exists, because the solver requires a bijective mapping on the whole ROI. So with a solver working in $\mathbb{R}^3$ this limitation could be resolved.

Advanced Brushes

In our system the brush is binary, i.e., a vertex is either inside or outside of the ROI. However, with the introduction of our layering approach, we already provided the methods needed for a weighting of the Laplacians. So a whole new set of brushes would be possible with different weightings of source and target. We expect that especially a smooth transition from target to source Laplacians would prevent artefacts that occur with our current approach (e.g. in the example shown in Figure 7.2).

Coating

In [Sorkine et al. 2004] the authors introduce a method to peel off the coating of a surface (i.e., the high-frequency surface details) and transfer it to another mesh. Although the layering approach together with derivative boundary constraints has a similar effect, we assume that a precompuation step peeling off the coating of the source mesh could significantly improve the quality of the layering approach, as well as provide the user a whole new set of possible use cases.

Improved Stitching

The stitching of source and target mesh is currently often introducing unsatisfactory results. Especially when the edge lengths in the source and target patches do not have the same magnitude, current stitching algorithm tends to produce many small triangles. Also the stitching proved to be one of the major remaining performance bottlenecks, so an improvement to the stitching could as well improve the real-time experience.

Remeshing

Currently we use the topology of the source canvas for the cloned geometry. Now as we have introduced the layering approach, information from both meshes is used to compute the cloned surface. Thus with our current approach we might lose geometric details originating from the target mesh. As there is no notion of combining triangulations, we propose to remesh the ROI. The information about the Laplacians might even be used to improve the quality of the triangulation. Remeshing the ROI would also lift the issues that currently exist with the stitching, as the stitching step would fall away.
Another important advantage of remeshing would be the increased robustness. When overlaps occur, the solver still computes a smooth solution, and the visual artefacts mostly occur due to the lookup of the vertices of the overlapping triangles. So when the remeshing would be computed based on the information in the rectangular grid, even in cases of overlaps a smooth surface could be computed.

**Normal Smoothing**

Currently we compute the normal smoothing upon loading of the meshes, and way we compute the smoothed normals is both slow and suboptimal in the quality. Ideally the normal smoothing would only consider vertices that are part of the canvas. However, in the current state we need the smoothed normals for robust estimation of the canvas. It would also be desirable that the user may interactively change the amount of normal smoothing.

**Implicit Baking**

Although the optimisations to the parametrisation algorithm, especially the frame-propagation DEM, improved the quality of the cloning in cases when the brushing area is far away from the seed point and the surface orientation differs largely from the surface orientation in the seed point, at some point the algorithm starts to degrade and will eventually fail. So it would be helpful if the system fixates the so far copied geometry automatically and reseeds for further brushing. We call this implicit baking. Of course the key to successful implicit baking would be a good measurement of the change in brushing orientation and placement.

**Adaptive Grid Size**

In our solver the size of the grid is specified before starting the system by the user. As this is only a proof of concept, we left this issue open. However, the adaptation of the number of grid cells to a given problem would improve the system, leading to an easier usage and to more accurate results.
8. Conclusion and Future Work
User Interface

In this chapter we describe the software from a users perspective. First we describe the different states of the program and how the user may interact in those states. In a second section we explain the task of the different keybindings. Finally we briefly explain the configuration file and the additional debug views.

A.1. States

In this section we show how to use our prototype and what interactions the user can make in each state.

A.1.1. General Handling

When the system is started the user can choose the source and the target mesh. The source mesh is always on the left, the target on the right. The user can rotate the meshes by holding the right mouse button, in the corresponding part of the window. To move a mesh, the user needs to hold the right mouse button as well as \textit{ctrl}. To zoom in or out, the user needs to hold \textit{shift} and the right mouse button. Generally, the right mouse button is to change the view of the mesh, while the left mouse button is used to edit the mesh.
A. User Interface

A.1.2. Source Seed State

The user can choose the seed point with the left mouse button. The yellow dot gives a hint of the currently chosen point. The size of the brush can be changed with the mouse wheel and the up and down arrow keys.

Once the user is satisfied with her choice, she can continue to the next step by pressing the space bar.

A.1.3. Target Seed State

Again the seed point is selected with the left mouse button. The size of the brush is chosen as before and is interpreted as a relative scaling of the two meshes. Unlike the other states ones the left mouse button is clicked, the state is automatically changed to the brushing state. With this we want to give the user a more straightforward and intuitive experience.

A.1.4. Brushing State

Within the canvas the user can now brush by clicking the left mouse button to add geometry. With shift and the left mouse button the user can shrink the region of interest. As mentioned before, only the ROI shrinks but not the canvas.

By pressing the space bar the user can change to the canvas adjustment state.

A.1.5. Canvas Adjustment State

If the user is not satisfied with the initial placement of the canvas, she can adjust it in this state. With a left click the user changes the seed point of the source or the target mesh. To rotate a canvas, the user needs to additionally hold ctrl. The canvases can be scaled by clicking shift and the left mouse button.

By pressing the space bar the user can change back to the brushing state.
A.2. Keybindings

In the following Table A.1 we list all the keybindings, their function and in which states they may be used.

<table>
<thead>
<tr>
<th>Keybinding</th>
<th>Description</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space bar</td>
<td>Progress to the next step.</td>
<td>all</td>
</tr>
<tr>
<td>Enter</td>
<td>Finish the current cloning step and bake the cloned mesh with the target mesh.</td>
<td>brushing/ canvas adjustment</td>
</tr>
<tr>
<td>Backspace</td>
<td>Abort the current cloning attempt and start again with the source seed state.</td>
<td>all</td>
</tr>
<tr>
<td>Tabulator</td>
<td>Advance to the next debug view.</td>
<td>brushing/ canvas adjustment</td>
</tr>
<tr>
<td>L</td>
<td>Load new meshes. The current cloning attempt will be aborted.</td>
<td>all</td>
</tr>
<tr>
<td>S</td>
<td>Save the target mesh, to keep the results of the cloning operation.</td>
<td>all</td>
</tr>
<tr>
<td>Y</td>
<td>Change between derivatives and no derivative constraints in the solver.</td>
<td>brushing</td>
</tr>
<tr>
<td>X</td>
<td>Compute the overlap correction to improve the quality of the parametrisation.</td>
<td>brushing</td>
</tr>
<tr>
<td>C</td>
<td>Toggle the membrane mode. Right hand side of the solver is zero. Used for verification of the correctness of the solver.</td>
<td>brushing</td>
</tr>
<tr>
<td>V</td>
<td>Turn the debug output of the solver to text files on/ off.</td>
<td>brushing</td>
</tr>
<tr>
<td>Q</td>
<td>Turn the rotation interpolation for the Laplacian coordinate alignment on/ off.</td>
<td>all</td>
</tr>
<tr>
<td>W</td>
<td>Toggle the usage of the smoothed normals for the Laplacian coordinate alignment.</td>
<td>all</td>
</tr>
<tr>
<td>F</td>
<td>Toggle between original and frame-propagation DEM.</td>
<td>all</td>
</tr>
<tr>
<td>E</td>
<td>Enable the layering approach.</td>
<td>brushing</td>
</tr>
<tr>
<td>R</td>
<td>Change the number of iterations for the normals smoothing in the source mesh.</td>
<td>all</td>
</tr>
<tr>
<td>T</td>
<td>Change the number of iterations for the normals smoothing in the target mesh.</td>
<td>all</td>
</tr>
<tr>
<td>1</td>
<td>Toggle the visibility of the faces.</td>
<td>all</td>
</tr>
<tr>
<td>2</td>
<td>Toggle the visibility of the edges.</td>
<td>all</td>
</tr>
<tr>
<td>3</td>
<td>Toggle the visibility of the canvas textures.</td>
<td>all</td>
</tr>
<tr>
<td>4</td>
<td>Show the vertex normals, respectively the local frame.</td>
<td>all</td>
</tr>
<tr>
<td>5</td>
<td>Show different colours for target, source and stitch mesh. May only be active when 3 is not active.</td>
<td>all</td>
</tr>
<tr>
<td>6</td>
<td>Hiding the cloned source mesh.</td>
<td>all</td>
</tr>
<tr>
<td>8</td>
<td>Hiding the stitch mesh.</td>
<td>all</td>
</tr>
<tr>
<td>9</td>
<td>Show the global frame</td>
<td>all</td>
</tr>
</tbody>
</table>
A. User Interface

A.3. Configuration File

There are several settings the user may change before starting the system. Those settings are stored in the config.xml file in the program folder. The Table A.2 briefly explains the different variables and what values they may take.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_width</td>
<td>The width of the window.</td>
<td>number in pixel</td>
</tr>
<tr>
<td>window_height</td>
<td>The height of the window.</td>
<td>number in pixel</td>
</tr>
<tr>
<td>paramspace_rez</td>
<td>Resolution of the grid used for the solver.</td>
<td>$n^2 + 1$</td>
</tr>
<tr>
<td>paint_rez</td>
<td>Resolution of the texture the user is brushing in.</td>
<td>$&gt;= paramspace_rez &amp; n^2$</td>
</tr>
<tr>
<td>vcycle_num_smoothing</td>
<td>Number of pre and post relaxation steps in the solver.</td>
<td>$&gt;= 2 &amp; 2n$</td>
</tr>
<tr>
<td>vcycle_num_cycle</td>
<td>Number of additional iterations the solver should take.</td>
<td>$&gt;= 0$</td>
</tr>
<tr>
<td>src_normal_smoothing_iter</td>
<td>Number of normal smoothing iterations in the source mesh.</td>
<td>$&gt;= 0$</td>
</tr>
<tr>
<td>tgt_normal_smoothing_iter</td>
<td>Number of normal smoothing iterations in the target mesh.</td>
<td>$&gt;= 0$</td>
</tr>
<tr>
<td>bw_checkerboard</td>
<td>Use a black and white checkerboard texture instead of the normal texture to visualize the parametrisation.</td>
<td>0 or 1</td>
</tr>
<tr>
<td>use_frame_propagation</td>
<td>Use the frame-propagation DEM instead of the original DEM.</td>
<td>0 or 1</td>
</tr>
<tr>
<td>use_frame_smoothing</td>
<td>Use the rotation interpolation approach to estimate the rotations for the Laplacian coordinate alignment.</td>
<td>0 or 1</td>
</tr>
<tr>
<td>naive_frame_estimation</td>
<td>Use the naive approach to estimate the rotations for the Laplacian coordinate alignment, i.e., use the vertex normals instead of the smoothed normals.</td>
<td>0 or 1</td>
</tr>
<tr>
<td>use_layering</td>
<td>Use the layering approach.</td>
<td>0 or 1</td>
</tr>
<tr>
<td>font_filename</td>
<td>Specifies the font type used in the window.</td>
<td>TrueType font file (.ttf)</td>
</tr>
<tr>
<td>use_font</td>
<td>Show/hide the text in the window.</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

A.4. Debug Views

With the tabulator key the user can access debug views, for the verification of data in the parameter space. In the following we describe the content of each of those views. All views in this section are from the example in Figure 4.6.
A.4. Debug Views

A.4.1. Inverted Triangles

In this view we visualize the triangles that are inverted in parameter space (Figure A.1). In the ideal case there are no yellow patches in this view, meaning that there are no overlaps in parameter space.

Figure A.1: The blue area denotes part of the parameter space where no overlaps are, while the yellow patches are inverted triangles.

A.4.2. Laplacian Coordinate Alignment Rotations

In this view we visualize the rotations to align the Laplacian coordinates (Figure A.2). We render the 4-vector that represents a quaternion to a RGBA texture. The main idea in this visualization is to have a fast verification of the smoothness of the field of rotations.

Figure A.2: The quaternions are rendered as colour values, to verify the smoothness of the field of rotations.
A. User Interface

A.4.3. Projected Source Canvas

In this view we render the projection of the source canvas to the parameter space $P_s(C_s)$ (Figure A.3).

![Projected Source Canvas](image)

*Figure A.3.: The visualization of $P_s(C_s)$.*

A.4.4. Projected Target Canvas

In this view we render the projection of the target canvas to the parameter space $P_t(C_t)$ (Figure A.4).

![Projected Target Canvas](image)

*Figure A.4.: The visualization of $P_t(C_t)$.*
A.4.5. Region of Interest

In Figure A.5 we visualize the texture keeping track of the ROI.

![Figure A.5: The visualisation of the ROI (yellow).]

A.4.6. Mapping of $R_{\text{in}}$

In this view we render the projection of $R_{\text{in}} = \{v \in C_s | P_s(v) \in Q\}$ to the parameter space (Figure A.6).

![Figure A.6: The visualization of $P_s(R_{\text{in}})$.]
A. User Interface

A.4.7. Mapping of $R_{out}$

In this view we render the projection of $R_{out} = \{ v \in C_t | P_t(v) \notin Q \}$ to the parameter space (Figure A.7).

![Figure A.7: The visualization of $P_t(R_{out})$.]

A.4.8. Stitch Mesh

In Figure A.8 we show the triangulation of the stitch mesh in parameter space.

![Figure A.8: The visualization of the stitch mesh in $\mathbb{R}^2$.]
Appendix

B.1. The Shortley-Weller Discretisation Scheme

The convergence of the multigrid solver is highly dependant on the discretisation at the boundary of the ROI, as we showed in Figure 6.4. We use the Shortley-Weller discretisation scheme, which was first introduced in [Shortley and Weller 1938], but build our argumentation and nomenclature on the more recent work of [Hackbusch 2005].

To derive the Shortley-Weller scheme we start with defining the neighbourhood of a point \((x_i, y_i)\) as it is shown in Figure B.1. In the normal case the distance between 2 vertices on a regular grid is \(h\). For points near the boundary this does not hold anymore and the distances \(h_i, h_{i+1}, k_i\) and \(k_{i+1}\) are smaller than \(h\).

\[
\begin{align*}
(x_i, y_i) & \quad (x_{i-1}, y_i) \\
(x_{i+1}, y_i) & \quad (x_{i+1}, y_{i+1}) \\
(x_i, y_{i-1}) & \quad (x_{i+1}, y_{i-1}) \\
\end{align*}
\]

\(h_i, \quad h_{i+1}, \quad k_i, \quad k_{i+1}\)

Figure B.1: The (irregular) neighbourhood of a point \((x_i, y_i)\).
B. Appendix

To handle such irregular structures, we can not use finite differences, but need to derive a more elaborate discretisation scheme. We first look at the one-dimensional case. Let us consider a function \( u : \mathbb{R} \to \mathbb{R} \). With divided differences the second derivative of \( u \) is:

\[
\frac{\partial^2 u(x)}{\partial x^2} = 2 \frac{u(x_{i+1}) - u(x_i)}{x_{i+1} - x_i} - \frac{u(x_i) - u(x_{i-1})}{x_i - x_{i-1}}.
\]

Inserting the distances defined in Figure B.1 the second derivative becomes:

\[
\frac{\partial^2 u(x)}{\partial x^2} = 2 \frac{u(x_{i+1})}{h_{i+1}(h_i + h_{i+1})} - \frac{2u(x_i)}{h_{i+1}h_i} + \frac{2u(x_{i-1})}{h_i(h_i + h_{i+1})}.
\]

(B.1)

Now let us consider the two-dimensional case. We describe the distances relative to the regular grid distance \( h_i = h_i' h \) and \( k_i = k_i' h \). Using (B.1) the two-dimensional Laplace Operator becomes:

\[
\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \frac{1}{h^2} \left[ \frac{2}{h'_i(h'_i + h'_{i+1})} \right. - \left. \frac{2}{k'_i(k'_i + k'_{i+1})} \right. \left. + \left( \frac{2}{h'_i h'_i} \right) \right. \left. + \left( \frac{2}{k'_i k'_i} \right) \right. \left( \frac{2}{h'_{i+1}(h'_i + h'_{i+1})} \right. \left. \right. \right],
\]

(B.2)

which is the Shortley-Weller discretisation scheme. When \( h'_i = h'_{i+1} = k'_i = k'_{i+1} = 1 \), this scheme becomes the standard five-point stencil. This means that the computation only differs for the near boundary points, while the computation of the interior points stays the same.

In our system however, we define the right hand side \( r \) not as the Laplacian, but as the Laplacian up to a scale factor:

\[
r(x_i, y_i) = h^2 \left( \frac{(h'_i + h'_{i+1})}{2} \frac{\partial^2 f(x_i, y_i)}{\partial x^2} + \frac{(k'_i + k'_{i+1})}{2} \frac{\partial^2 f(x_i, y_i)}{\partial y^2} \right).
\]

So in our system we do not use the stencil (B.2) but the following adapted stencil:

\[
\begin{bmatrix}
\frac{1}{h'_i} - \left( \frac{1}{h'_i} + \frac{1}{h'_{i+1}} + \frac{1}{k'_i} + \frac{1}{k'_{i+1}} \right) \frac{1}{h'_{i+1}} \\
\frac{1}{k'_i}
\end{bmatrix}
\]

(B.3)

It obviously still gets reduced to the five-point stencil when applied to interior points.

The higher order stencil follows then by convolution of stencil (B.3) with itself.
BIBLIOGRAPHY

Bibliography


STENTZ, A., AND MELLON, I. C. 1993. Optimal and efficient path planning for unknown and