Extraction of Displacements between Mesh and Basemesh

Martin Stuchlík*
Supervised by: Martin Madaras

Faculty of Mathematics Physics and Informatics
Comenius University
Bratislava / Slovakia

Abstract

This paper is devoted to the extraction of the displacement between two predefined polygonal meshes. Using a combination of GPU tessellation and vector displacement mapping, our approach enables the original high-poly detailmesh to be rendered approximately by the displaced low-poly basemesh at various LOD in real-time. During the preprocessing phase, the cross-parameterization between the meshes is obtained and locally optimized to maximize the amount of preserved geometry detail. Therefore, our method can store and reconstruct even more complex non-convex surface protuberances.

Keywords: displacement mapping, parameterization, remeshing

1 Introduction

In computer graphics, a vertex displacement mapping denotes a set of techniques to change 3D positions of vertices while using information stored in a raster texture called a displacement map. These methods enable us to easily enhance detail of polygonal meshes and even to reconstruct a highly detailed mesh from a remarkably downsampled basemesh, when both meshes have appropriate parameterizations. Although, originally not meant to be real-time [4], vertex displacement mapping proved to be well suited for modern GPUs, which have enough processing power to combine real-time tessellation with high resolution textures.

Generally, when approximating a high-poly mesh by a low-poly basemesh, the basemesh is either obtained by iterative edge-collapsing of the initial high-poly mesh [12, 15] or it is strictly predefined [11]. Such approaches lead to good results, however the geometry of the basemesh is limited as it has to be strongly related to the original geometry or the method itself. Therefore, our motivation for this work was to propose and implement a solution which enables to extract a displacement map between two input meshes independently of their origin, as shown in Figure 1. The main advantage over existing methods is that our method can be later combined with algorithms which produce their own specific basemeshes such as the skeleton driven method by Bærentzen et al. [1].

Our approach consists of several steps. The initial inputs are two arbitrary meshes (the high-poly mesh and a low-poly basemesh) with matching convex parameterization boundaries. In the first step, parameterizations have to be computed and optimized according to these boundaries. When both meshes are properly parameterized, the displacements can be extracted and saved into the texture, called a displacement map. After the displacement map is obtained, it can be further used to displace the tessellated basemesh in real-time. A special care is given to the local parameterization optimizations, which affects the amount of preserved geometry detail.

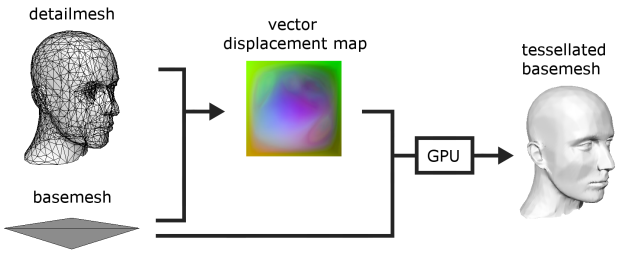

Figure 1: A simplified displacement extraction and consequent real-time reconstruction pipeline. Parameterizations are computed and displacement map is extracted during the preprocessing phase. Once the displacement map is obtained, it can be used to displace the tessellated basemesh in real-time.

2 Related work

This section is devoted to the state of the art in vector displacement mapping, related parameterization techniques and similar texture-based remeshing methods.

2.1 Tessellation and displacement mapping

Tessellation is a process of subdividing larger primitives, named patches, into many smaller ones [13]. In modern OpenGL versions (4 and higher) the tessellation of triangle patches can be computed directly on the GPU. Changing the OpenGL tessellation level results in various LOD

*stuchlo@gmail.com

of the mesh surface. We generally distinguish between two main types of tessellation. Uniform tessellation subdivides each patch into a number of primitives according to a predefined subdivision level, while adaptive tessellation adjusts the subdivision level depending on a local amount of detail or geometry visibility [10]. After the primitives are subdivided, the generated geometry can be easily displaced using data from a displacement map.

There are two main approaches to how a geometry detail can be stored in a texture. In most cases a simple height map containing only one channel is used to displace vertices along the surface normals [4], however this method fails to reconstruct non-convex protuberances as shown in Figure 2. Some authors solve this problem with two displacement steps [15], however a more complex geometry (for example the dinosaur in Figure 4) would need more steps.

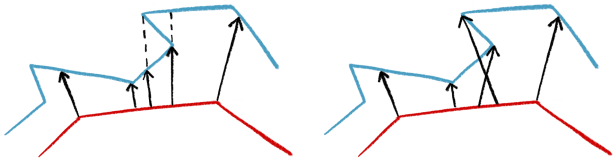

Figure 2: A comparison of displacements between a detailed mesh (blue) and a basemesh (red) using a height map (left) and a vector displacement map (right). As can be seen, the problematic protuberance between the dotted lines cannot be reconstructed using a simple height map.

Unlike the height maps, vector displacement maps (sometimes referred as "non-linear displacement maps" [16]) store the whole 3D displacement vector in red, green and blue texture channels. Vector displacement maps are often saved as high dynamic range floating point data rather then common RGB format [7, 8]. Such approach consumes more space but enable displacements between more complex surfaces.

2.2 Local parameterization

A suitable parameterization, assigning texture coordinates to vertices, plays the key role in displacement mapping, as it directly determines the amount of stored detail. Providing exact one-to-one mapping between meshes and a displacement map, only bijective parameterizations are robust enough, as parameterization fold-overs would cause loss of detail on overlapping faces. Naturally, bijective texture mapping can be obtained only if a mesh has a planar topology, so a parameterization of more complex meshes has to be cut into planar regions.

One of the most basic local parameterization methods is Tutte's barycentric mapping [2]. Based on advanced graph theory, this technique works on triangular polygonal meshes homeomorphic to a disk, where boundary uv coordinates form a convex polygon. To obtain a bijective parameterization, each inner vertex must be a convex combination of its neighbours. This is acquirable by iteratively moving the inner vertices coordinates into a barycentre of their neighbours or can be calculated by solving the following linear system [2]:

$$-a_{i,i}\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \sum_{j \neq i} a_{i,j}\begin{pmatrix} u_j \\ v_j \end{pmatrix}, \tag{1}$$

where,

$$\begin{cases} a_{i,i} = -\sum_{j \neq i} a_{i,j}, & \\ a_{i,j} = e_{i,j}, & \text{if vertices } v_i \text{ and } v_j \\ & \text{are connected by an edge,} \\ a_{i,j} = 0, & \text{otherwise,} \end{cases} \tag{2}$$

and $e_{i,j} = 1$.

The method itself does not preserve any shape metrics, but it can be easily extended by weighting vertices as Floater [5] did. It is proven that if the weights are positive and the matrix is symmetric, the resulting parameterization is bijective [14].

The next important group of parameterization techniques is based on analysing gradients of local linear mapping functions which transform vertices of a triangle between 3D space and 2D texture space. Especially for remeshing, stretch-minimizing parameteriztions showed to be the most suitable choice [6, 18], as minimizing stretch metric, introduced by Sander et al. [12] reduces under-sampling, while preventing texture distortions. Based on this metrics, Sander et al. proposed their stretch minimizing parameterization, which remarkably reduces texture under-sampling. Another notable method is the stretch minimizing parameterization by Yoshizawa et al. [17], which solves some unwanted texture artefacts caused by the method of Sander et al., while being very fast.

2.3 Vector displacement mapping and geometry images

Vector displacement mapping is used mainly in 3D modelling applications specialized in sculpting methods. The good examples of such software is the Autodesk Mudbox [7] and the Pixologic ZBrush [8], which are both widely used professional tools, producing excellent results. However, their goal is a bit different from ours as they are designed primarily to build vector displacement maps from user interactions and not to extract them from a pair of distinct polygonal surfaces.

Closely related to vector displacement mapping are geometry images introduced by Gu et al. [6] which enable to store whole geometry in a raster. The major advantage of this representation is a simplification of some remeshing methods. A good example is mesh morphing, which can be obtained as a simple interpolation between textures. This was used by Yu et al. [18] who combined ge-

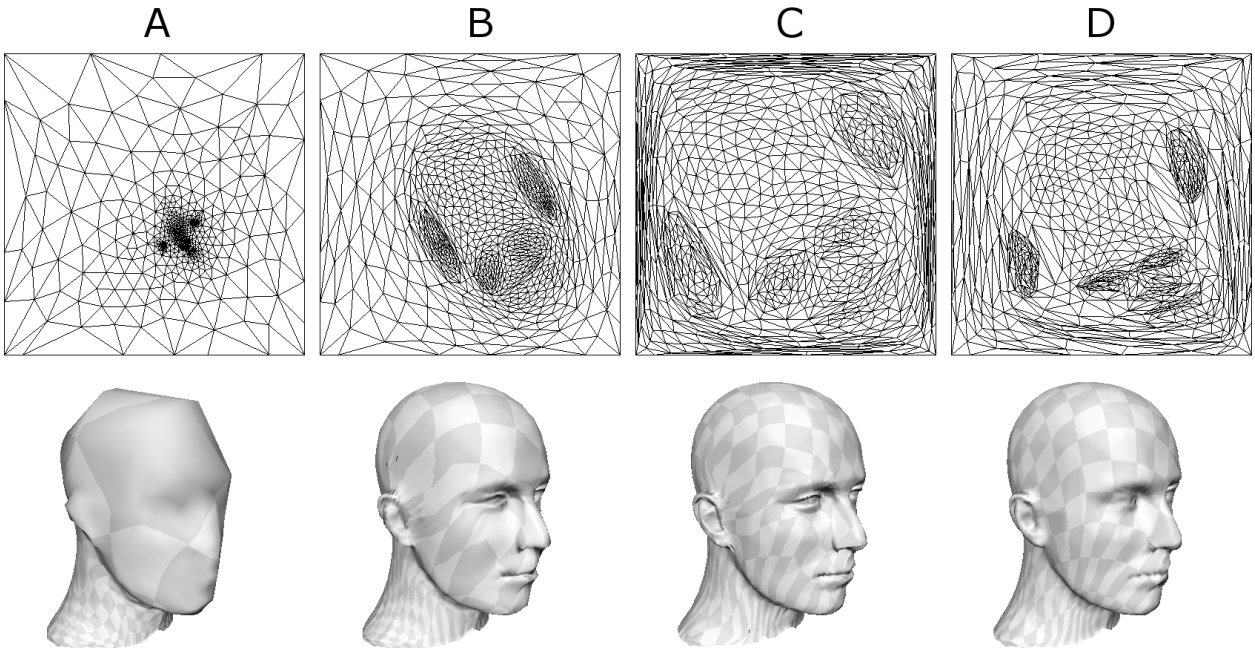

Figure 3: A comparison of local parameterization optimization methods: (A) Tutte's barycentric mapping [2], (B) our linear barycentric correction, (C) our non-linear barycentric correction, (D) stretch-minimizing parameterization by Yoshizawa et al. [17]. Each mannequin head model has been reconstructed from two triangles only, using 128×128 vector displacement map at OpenGL tessellation level 64. Normals are computed directly from displacement maps. Note that our non-linear method is better in reconstruction of some smaller geometry details at cost of texture distortions.

ometry images with consistent parameterization, which resulted in impressive morphing between considerably distinct meshes. Praun et al. [11] used geometry images to reconstruct genus-0 meshes from tetrahedral, octahedral and cubic basemeshes.

However, the idea of geometry image is slightly different from the vector displacement mapping as the texture stores surface positions directly rather than the difference between the surfaces.

3 Parameterization

Before displacements can be extracted, both the mesh and the basemesh must have a corresponding cross-parameterization, which means that boundaries of the mesh and the basemesh parameterizations must form the same convex polygon. Ideally, this should be a rectangle, because the displacement mapping is extremely vulnerable to boundary rasterization problems and even the smallest inaccuracy may cause a significant crack in the reconstruction. An initial boundary can either be set manually by a user or it can be acquired by an advanced algorithm. A good example of such algorithm is the STM (Skeleton texture mapping) introduced by Madaras et al. [9], which cuts the mesh parameterization into several rectangle regions according to the object skeleton. After corresponding boundaries are obtained, the cross-parameterization has to be locally refined to prevent fold-overs and further optimized to minimize under-sampling problems.

3.1 Linear barycentric correction

Our first experiments with Tutte's barycentric mapping [2] have shown that if mesh shapes a relatively regular grid (inner vertices have the same valence), even the simple barycentric mapping can provide a sufficient reconstruction of non-convex protuberances. Significant problems may appear on topologies consisting of multiple inscribed rings (for example mushroom-like objects), which barycentric mapping tends to shrink badly, so even with high tessellation levels, it is almost impossible to sample some necessary details. To stretch the coordinates "shrunk" by the barycentric mapping, we have experimented with extending the barycentric mapping by "correcting" weights. Unlike other authors, our goal is independent of texture visual qualities, so we tried to fight under-sampling in a rather different way. Instead of minimizing the stretch, we decided balancing triangle area to increase the probability of sampling all important details.

Our approach consists of two steps, first the original barycentric mapping is obtained to prevent fold-overs and then the parameterization is refined according to the earlier stated equations 1 and 2, where energy $e_{i,j} = A^t_{i,j}$ and $A^t_{i,j}$ equals the sum of the texture space area of the triangles incident with the edge. We can imagine these energies

as some ”springs” between the vertex coordinates. Edges with higher energy tend to shrink, while edges with lower energy conform to their neighbours. In our approach, edges incident to triangles with larger texture space area are rated with higher energy. Therefore, the area of over-sampled triangles in the initial mapping is directly reduced to get more texture space for the under-sampled triangles, as can be seen in Figure 3.

Since solving such linear systems on high-poly meshes can consume a lot of time, both initial and corrective step are implemented by iteratively moving the vertex coordinates, until the changes in the solution are smaller then a user set threshold.

3.2 Non-linear barycentric correction

Our linear weighting correction leads to significantly better visual results than the original barycentric mapping. However, it does not reach the sampling qualities of stretch minimizing parameterizations, so we decided to try a more complex non-linear optimization. To reduce the time complexity we use a similar concept to Yoshizawa et al. [17], where the non-linear problem is solved as a number of simple linear corrections, until the solution converges to the method optimum.

Therefore, our non-linear optimization consists of the initial mapping obtained by the previously mentioned linear correction and a series of additional corrective steps, where in each, the solution is refined with energy:

$$e_{i,j} = \sqrt{{e'_{i,j}}^2 \cdot {A^t_{i,j}}^2 \cdot 1/A^o_{i,j}}, \tag{3}$$

where $e'_{i,j}$ represents the energy from the previous step and $A^o_{i,j}$ is the sum of the object space area of the triangles incident with the edge, to prevent extreme texture distortions, which could cause problems with normal reconstruction. The initial mapping is being refined, until the changes in solution are smaller then a threshold.

This method appears good in stretching shrunk areas and thus provides better reconstruction of some small details at cost of texture distortions, compared to stretch minimizing parameterizations, as can be seen in Figure 3.

4 Displacement extraction

After the sufficient cross-parameterization is obtained, the displacements between the two planar meshes can be extracted and rendered to the vector displacement map, even if planar topologies of the rectangle-shaped regions are mutually incomparable.

To avoid a high time complexity of analytic geometry based algorithms, we decided for a much more elegant and lightweight raster-based solution. The idea is to render both surfaces into the geometry images [6], where red, green and blue texture channels represent the 3D position on the surface. Then, the whole displacement map extraction is reduced to a simple per-texel subtraction of these images as can be seen in Figure 4. This method proved to be very efficient, as the displacement map is bijective and can be easily rendered in real-time on modern graphic cards.

The 16 and 32 bit float data texture formats proved better then common RGB, in preserving the geometry detail, since low precision can cause some strange surface artefacts resembling voxelized volumes.

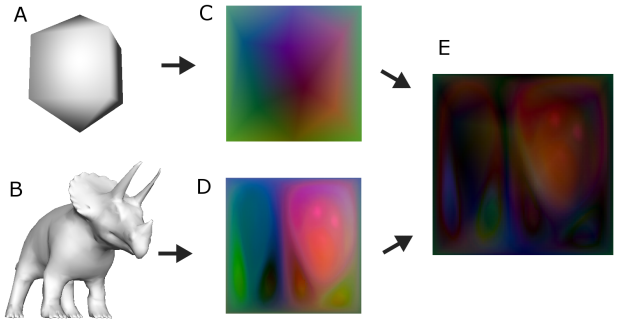

Figure 4: The displacement extraction pipeline: (A) basemesh, (B) detailmesh, (C-D) geometry images, (E) final displacement map. Texture colors were tone-mapped for illustrative purposes from an original 32 bit EXR float format.

4.1 Avoiding cracks

The major weakness of the above described method is the region boundary. When boundary data from a polygon is written into a texture, the OpenGL rasterizer samples value in the middle of the processed texel which often does not correspond to the exact edge value, which is therefore lost. Such data loss may appear negligible, however in the displacement mapping it is crucial, as it causes major geometry cracks in a displaced geometry. Fortunately this can be easily solved by re-rendering the boundary using line segments, which ensure that proper outline values are written, as can be seen in Figure 5.

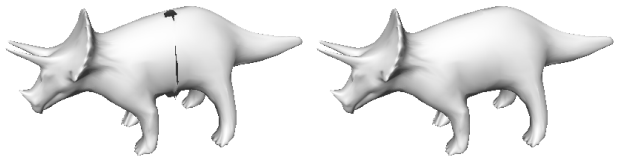

Figure 5: Reconstructions of a dinosaur without (left) and with (right) the boundary correction, using a 128×128 displacement map in a combination with a normal map, both with a linear texture filtering.

Naturally, this solution works only in a combination with the nearest texture filtering. To use more advanced

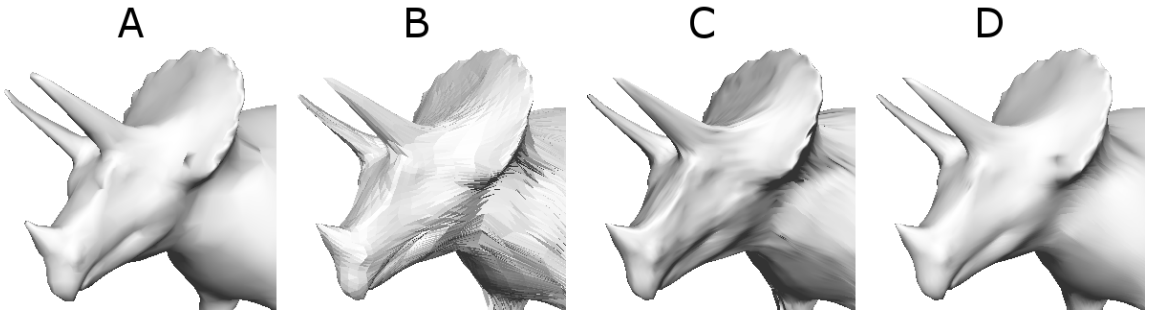

Figure 6: Normal reconstruction: (A) original mesh with smooth shading, (B) raw reconstruction with flat shading, (C) reconstruction with approximated normals, (D) reconstruction with normal mapping. The dinosaur head has been reconstructed from one triangle only, using a 128×128 texture at OpenGL tessellation level 64.

interpolation methods, both the boundary of mesh parameterization has to be fitted to the texels of the vector displacement map and a thickness of the rendered boundary should be two texels, to prevent boundary interpolation inaccuracies. Of course, redundant boundary data reduces the storage capacity of the displacement map, however using advanced interpolation methods is still rewarding as the reconstructed geometry is much smoother.

5 Real-time reconstruction

Once the displacement map is obtained, vertices of the tessellated basemesh can be displaced by simply adding vectors from the texture. The displaced surface approximate the original detailmesh and the tessellation level determines the reconstruction LOD. However, the surface reconstruction is not only about displaced geometry but also about reconstructed shading.

There are two major ways, how to reconstruct the original shading. Probably the most simple and precise method is the normal mapping, where the surface normals of the original mesh are stored in an additional texture. The second method, where normals are approximated directly from the displacement map as a cross product of partial derivations, does not need any additional data. However, this can cause some unexpected side effects, since approximated normals correspond to the real geometry and have nothing to do with per-vertex normals, which are interpolated all over the surface of the original detailmesh. Therefore, approximated normals often resemble flat shading and in some extreme cases, reconstruction may look even more detailed than the original. For a comparison see Figure 6.

6 Results

Our results consist of a reconstruction quality evaluation using various settings and a comparison between our non-linear barycentric correction and a stretch-minimizing parameterization, to test its suitability for remeshing algorithms.

Testing on various models has shown, that generally our method produces fairly good, real-time reconstructions with very small mean error. To measure errors, we use a MeshLab [3] implementation of Hausdorff distance between the original mesh and a reconstructed surface, normalized by diagonal of the mesh bounding box. The overall quality of the final reconstruction depends mainly on a displacement map resolution and the LOD determined by the tessellation factor, as can be seen in Figure 8.

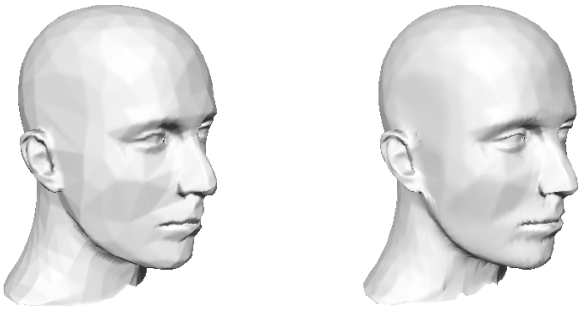

Figure 7: Example comparison between the reconstruction using the stretch-minimizing parameterization by Yoshizawa et al. [17] using a 256x256 displacement map (left) and the reconstruction using our non-linear barycentric correction using a 128×128 displacement map (right). Both mannequin head models have been reconstructed from two triangles at OpenGL tessellation level 64. The right-hand reconstruction has 14.3% higher mean error, while using only 25% of an original texture space. Despite the overall error increase, our method is still slightly visually better in preservation of facial details.

Comparisons between our local parameterization optimization method and stretch-minimizing parameterizations have shown different error distributions. The regular sampling of stretch-minimizing parameterizations accumulates error in areas with higher detail, while our non-linear barycentric correction spreads error regularly along the surface. Therefore, our method is often visually bet-

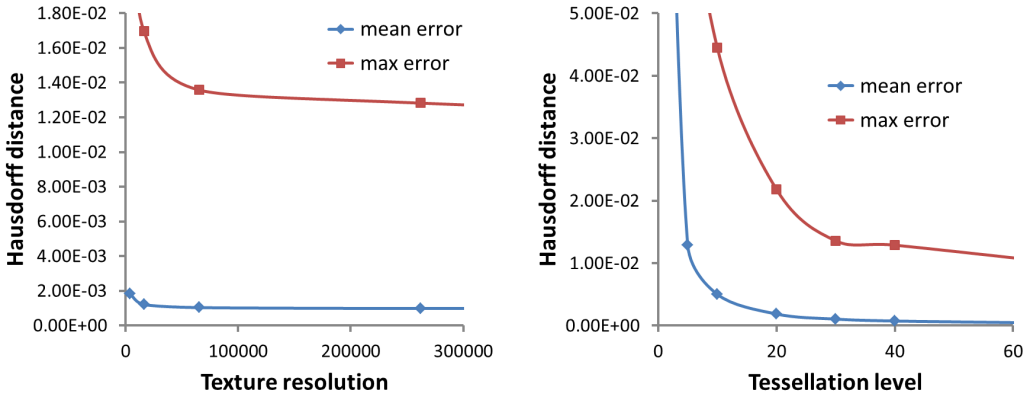

Figure 8: Evaluation graphs show dependence of texture resolution (left) and tessellation level (right) on normalized error. The tessellation level was evaluated using a 256x256 displacement map. The texture resolution was evaluated using tessellation level 30. The notable gap between the maximum and mean error is caused by high-frequency details, which are very difficult to sample. However, the low mean error reveals that such errors are relatively rare.

ter in storing geometry details, which enables the usage of smaller displacement and normal maps as shown in Figure 7. However, considering the texture distortions, our method is not suitable for colour texturing (except some special cases, when geometry details correspond to colour details) and in some cases it may even complicate the normal reconstruction.

Although our displacement map cannot be efficiently modified using simple graphics editor, stored displacements can be easily edited procedurally, as can be seen in Figure 9. Using combinations of trigonometric functions proved sufficient, since generated geometry is smooth and, therefore, normals can be restored effectively using partial derivations.

Figure 9: Example of procedural modifications: reconstruction of a bunny model using original displacement map (left), reconstruction using a modified dispalcement map (right). All surface modifications are composed by trigonometric functions.

7 Conclusion and future work

We have proposed and implemented an algorithm, which enables to extract a displacement map between two predefined meshes independently of their visual similarity and origin. This has many possible future utilizations, since it can be effectively used by techniques with a limited basemesh generation.

Another contribution of our work is the new local parameterization optimization, specially designed to preserve small geometry details. Although our parameterization is not suitable for colour texturing, generally it enables to use smaller displacement and normal maps, which saves memory and disk space.

However, some of the problems remain open as our method needs high tessellation factors to reconstruct sharp edges or spikes, and approximated normals often cause various stains, since normal approximation is prone to some texture distortions.

For the future work we would like to experiment with tangent space and adaptive tessellation. We plan to improve the normal approximation, to avoid usage of a normal map completely and develop a more complex local parameterization optimization, which may enhance preservation of sharp geometry details by fitting some considerable vertices and edges between the mesh and the basemesh. We would also like to experiment with mesh deformations driven by user interactions with the basemesh.

References

[1] Jakob Andreas Bærentzen, Marek Krzysztof Misztal, and K WełNicka. Converting skeletal structures to quad dominant meshes. *Computers & Graphics*, 36(5):555–561, Elsevier, 2012.

[2] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010.

[3] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and

Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference 2008*, pages 129–136, The Eurographics Association, 2008.

[4] Robert L Cook. Shade trees. *ACM Siggraph Computer Graphics*, 18(3):223–231, ACM, 1984.

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

[6] Xianfeng Gu, Steven J Gortler, and Hugues Hoppe. Geometry images. *ACM Transactions on Graphics (TOG)*, 21(3):355–361, 2002.

[7] Autodesk Inc. Vector displacement maps. `https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/3DSMax/files/`. [Online; accessed 6-December-2016].

[8] Pixologic Inc. Vector displacement maps. `http://docs.pixologic.com/user-guide/3d-modeling/exporting-your-model/vector-displacement-maps/`. [Online; accessed 6-December-2016].

[9] Martin Madaras and Roman Ďurikovič. Skeleton texture mapping. In *Proceedings of the 28th Spring Conference on computer Graphics*, pages 121–127. ACM, 2013.

[10] Matt Pharr and Randima Fernando. *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005.

[11] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 340–349. ACM, 2003.

[12] Pedro V Sander, John Snyder, Steven J Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM, 2001.

[13] Graham Sellers, Richard S Wright Jr, and Nicholas Haemel. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley, 2013.

[14] Alla Sheffer, Emil Praun, and Kenneth Rose. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision*, 2(2):105–171, 2006.

[15] Ilya Tisevich and Alexey Ignatenko. Displacement and normal map creation for pairs of arbitrary polygonal models using gpu and subsequent model restoration. pages 61–68, Citeseer, 2007.

[16] Neil West. Investigation of surface mapping techniques & development of non-linear displacement. Student Project. Bournemouth University, 2007.

[17] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. A fast and simple stretch-minimizing mesh parameterization. In *Shape Modeling Applications, 2004. Proceedings*, pages 200–208. IEEE, 2004.

[18] Jin-Bey Yu and Jung-Hong Chuang. Consistent mesh parameterizations and its application in mesh morphing. In *Computer Graphics Workshop*, 2003.