Interactive Geometry Decals

Jens Schneider Joachim Georgii Rüdiger Westermann

Technische Universität München
Email: {jens.schneider|georgii|westermann}@tum.de

Abstract

We present a novel real-time method for geometric displacement mapping on arbitrary 2-manifold triangle meshes. It is independent of the surface resolution and allows very fine geometric details to be added. We first compute a local surface parameterization using barycentric particle tracing and a constrained mass-spring system. This system satisfies the hard constraint of keeping all mass points on the surface. We propose a novel, entangled spring topology that is regularly 6-connected and has a chromatic number of 2. Therefore, the system can be efficiently solved on the GPU using a Gauss-Seidel solver. To render the displaced surface, we introduce a new GPU technique to cut out a surface patch at sub-pixel precision. The displacement mesh is then smoothly blended into the resulting opening. We show that our method overcomes common problems of displacement mapping such as limited resolution of the base surface and the need for a low-distortion global parameterization.

1 Introduction

Displacement mapping [4, 5] is a popular method to add geometric details to a 3D polygonal object that are difficult to model explicitly. When used to augment polygonal surfaces displacement mapping requires fine mesh subdivision such that the surface can accurately reflect the desired geometric details. If the surface regions are known where geometric details will be added, these regions can be refined locally in a pre-process. However, in scenarios where the detail geometry is dynamically moved or re-positioned, this requirement limits the use of displacement mapping. In this case, the surface has either to be subdivided uniformly up to the maximum required resolution, or the affected surface regions have to be refined adaptively. Both approaches have drawbacks because they require either a huge amount of primitives to be stored or a local refinement kernel to be evaluated in every frame. Furthermore, because vertex positions in the refined mesh do generally not coincide with the positions of samples in the displacement field, severe reconstruction artifacts can occur.

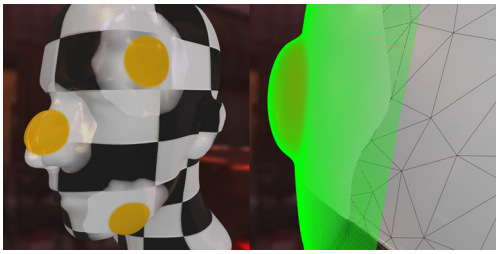

Figure 1: High-resolution geometry decals (512^2 each) added to a textured model. Our method is independent of the resulution of the base mesh and it achieves interactive frame rates for the placement and rendering of the decals.

1.1 Contribution

In this paper we present a fast and high-quality method for mapping fine-detail geometric displacements onto 2-manifold triangle base meshes that is independent of the mesh resolution. This is achieved by using a separate displacement geometry—the displacement grid—to replace parts of the base mesh. To align the displacement grid with the base surface, we present a novel approach to interactively compute a local surface parameterization. We trace a regular 2D grid on the base surface, and we then introduce a special 6-connected constrained mass-spring system to interactively relax the grid towards isometry. The mass-spring system is solved directly on the 3D surface, thereby enforcing all mass points to stick to this surface. Due to the special topology of the spring network, the relaxation process can be accelerated significantly by a modified GPU-based constrained Gauss-Seidel

solver. Vertices of the relaxed grid are finally displaced using a pre-computed displacement field (also see Figure 1).

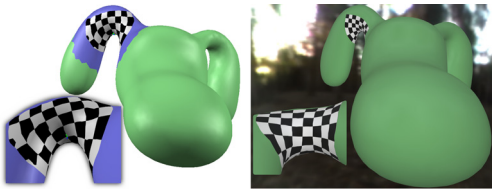

Figure 2: At saddle points, many local parameterizations show distortions. Left: The method by Schmidt et al. [27]. Right: Our method results in less severe distortions at patch boundaries.

The resulting parameterization is similar to the discrete exponential map approach by Schmidt et al. [27], yet it does not generally map geodesics in the tangent plane to geodesics on the surface. However, our method generally results in less distortions (see Figure 2) since we construct the parameterization in a forward manner by tracing paths on the mesh. Unlike the patchino approach [25] we do not require a global parameterization of the base mesh.

To render a visually smooth transition between the base mesh and the geometry decal, we introduce a cutout mask to discard the part of the base mesh that is covered by the geometry decal. The decal is then blended smoothly into this opening. To enable a seamless embedding of the decal into the base mesh, appearance properties such as texture coordinates and tangent frames are propagated from the base mesh to the decal for rendering.

Currently, our method is limited to local displacements constructed from height fields. We do not provide means to cover the entire base surface and we assume the geometric features added to be small compared to features of the base surface.

2 Related Work

Today, subdivision-based displacement mapping can be efficiently performed using GPU-based techniques [9, 33, 30, 18, 19, 3]. While being a fine technique, uniform subdivision results in a huge amount of triangles to be computed, stored, and rendered if small geometric features are added. Adaptive subdivision can alleviate this short-coming, but it is not suitable for real-time applications that move displacements dynamically across the base surface. The reason is that dynamic memory allocation cannot be performed efficiently on recent GPUs. Although future graphics APIs will provide functionality for adaptive subdivision [14], the performance implications are not yet clear. Furthermore, the triangulation of the subdivision surface is generally not aligned with the displacement field, resulting in potential reconstruction artifacts.

As an alternative to geometric displacement mapping, image-based techniques seek to simulate the appearance of a displaced object without the displacement being modeled geometrically. Prominent examples include bump [2], parallax [20], and relief mapping [24, 26]. These techniques can achieve interactive frame rates at a reasonable image quality, but they require a consistent surface parameterization to map the displacement onto the surface.

View-independent and generalized displacement maps [34, 35] store a five-dimensional map of the displacements in which the distance between the surface and the displaced geometry is encoded for each potential view. These methods handle silhouettes correctly and do not require a surface parameterization. Shell-based methods encode displacements into image layers or volumetric textures [23, 21, 10], thereby modeling the displacement as a spatial structure. However, all these methods require substantial pre-processing and compression if the displacement field changes.

Most traditional displacement techniques map the displacement field to the underlying surface via a parameterization. Generating a low-distortion parameterization on an arbitrary 2-manifold mesh, however, is numerically involved [13, 17, 29]. A particular class of methods computes surface parameterizations by solving linear systems [6, 22, 37, 31, 36, 28] or graph searches [27]. Thus, these approaches are suitable for interactive applications. Specifically, local parameterization techniques using spring networks [12, 25] interpret a part of the mesh as a mass-spring network. Then, they fix the boundary of this part in a 2D domain and they relax the network to a steady configuration that minimizes some energy functional.

3 Displacement Grid Layout

Our displacement mapping technique takes as input an arbitrary surface point p_c, a height field, its orientation d_c at p_c in the surface's local tangent plane, and the extent of the displacement field in

object space. Figure 3 illustrates these settings for a particular example. The mapping of the displacement field onto the surface is then performed without any further user intervention. It is computed in two phases: the initial grid layout phase, which is entirely performed on the CPU due to the inherently sequential nature of the algorithm, and the grid relaxation phase, which can be executed efficiently in parallel on the GPU.

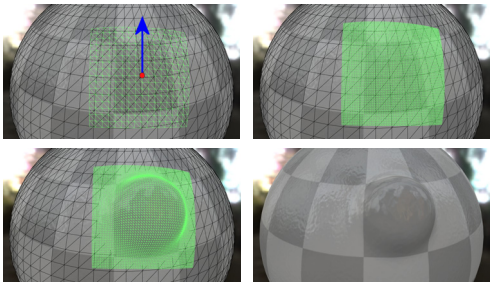

Figure 3: Displacement mapping overview. Top left: A coarse grid is traced on the surface, starting with the center point (red) and a reference direction (blue). Top right: A higher resolution grid. Bottom left: The displaced grid. Bottom right: The rendered surface.

In the first phase, a regular grid is *traced* on the surface using barycentric vertex tracing. This tracing proceeds by "jumping" from one triangle edge to the next one until a pre-defined distance has been exceeded. The final vertex position can then be determined by linear interpolation between the last two edge crossings.

Starting at p_c, three new grid vertices are generated by moving a step on the surface along d_c, a direction d_c' orthogonal to d_c, and the half-way direction between d_c and d_c' (see Figure 4(2)). Step sizes are chosen according to the grid spacing. Next, the resulting 2×2 grid is grown along $\pm d_c$ and $\pm d_c'$ as shown in Figure 4(3). At each corner vertex, a new direction half-way between the respective propagation directions is computed and used to generate a new corner point (see Figure 4(4)). Every new non-corner vertex stores the direction it was reached from and uses this direction to advance the grid vertices in the next iteration. To constrain each vertex to the surface, directions are first rotated into the plane spanned by the triangle containing the vertex before using them to advance the grid. At corner vertices, the two respective front directions are first rotated into the tangent plane, and they are then averaged to obtain the direction into which to advance the vertex.

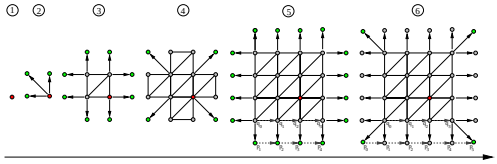

Figure 4: Tracing of a quadrangular grid. The user picks a position (1) and the patch is automatically grown around this position (2-6) using an advancing front algorithm (new points are colored green).

To prevent the traced grid from having self-intersections, we exploit the knowledge we have about the relative position of vertices to each other in the 2D grid. If a position q_i is "left" of a position q_{i+1}, then, after advancing the front to obtain p_{i+1} from q_i and p_{i+2} from q_{i+1} (the index shift stems from corner points), p_{i+1} should be "left" of p_{i+2}. Although the notion of "left" is intuitive in flatland, it becomes highly complex on curved surfaces. Hence we first compute a plane η spanned by the base surface normal at q_i and the edge from q_i to p_{i+1}. If the positions q_{i+1} and p_{i+2} are on the same side of η, we connect them by an edge and proceed. If not, positions p_{i+1} and p_{i+2} have to be swapped in the current front, and we proceed by checking the ordering between p_i and p_{i+1} by backtracking one position (see Figure 5). To prevent grid edges to cross again in an upcoming step, the tracing directions of two vertices are replaced by their average direction. The described method can resolve all folds that are caused by adjacent vertices in the grid. To avoid folds that are caused by vertices connected via more than one edge, k-tuples of positions have to be examined. Since for large k the underlying base mesh cannot be safely assumed to be locally planar, a plethora of pathological cases arises that seem to be very difficult—if at all possible—to resolve. To avoid these cases, we only use a small

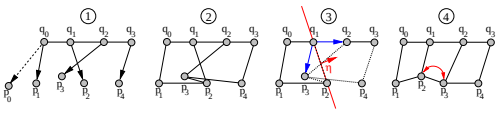

Figure 5: Since naïve connectivity of propagating fronts (1) results in folds (2), re-ordering the positions (3) is used to avoid these problems (4).

neighborhood $k = 3, \ldots, 5$ at the risk that not all folds are resolved.

4 Grid Relaxation

To improve the quality of the displacement grid, we utilize a mass-spring-based system. A center of mass is placed at every vertex and each edge is interpreted as a spring. Note that this notion does not exclude additional springs that do not coincide with edges. In our scenario, two types of constraints have to be ensured

1. **Soft constraints:** Each spring has to reconstitute its rest length. This is handled directly by the mass-spring system. In a converged state, internal forces due to compression of springs are compensated by opposing external forces. A stiffness can be associated with each spring to modify the importance of the respective constraint.
2. **Hard constraints:** Each mass point (vertex) has to be constrained to the base surface. While the initial grid satisifies this constraint, it has to be enforced during the relaxation process.

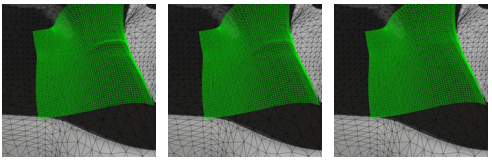

Figure 6: A 2D grid is placed on the nose of the Mannequin mesh. From left to right: The initial grid, the relaxed grid after 16 iterations, the final grid after 36 iterations.

Figure 6 depicts our mass-spring system that obeys these constraints.

The challenge in the grid relaxation process is to efficiently ensure all hard constraints. Even for one single vertex, the number of hard constraints is on the order of the number of triangles of the base mesh. Therefore, it is impractical to incorporate all hard constraints into the system of mass-spring equations. We propose a constrained Gauss-Seidel solver that solves one vertex at a time and then immediately enforces the hard constraint for this vertex. In Section 6 we show that the convergence of our solver benefits greatly of considering hard constraints as soon as possible. Note that this is different to solving the mass-spring equations using a conjugate gradient solver, where hard constraints can only be ensured after each full iteration.

4.1 Mass-Spring System

To solve the mass-spring equations we utilize a variation of the approach proposed by Baraff and Witkin [1] for cloth simulation. They arrived at the following compact system of equations

$$\left(M + \Delta t \frac{\partial f}{\partial v} - \Delta t^2 \frac{\partial f}{\partial x}\right) \Delta v = \Delta t \left(f + \Delta t \frac{\partial f}{\partial x} v\right),$$

where M is the mass matrix, Δt is the simulation step size, f contains the forces, and x and v refer to position and velocity of mass points. In addition we apply Rayleigh damping of the form $\partial f / \partial v = -\kappa I$, where I is an identity matrix, to improve numerical stability [7]. Updating a particular vertex now corresponds to solving a single equation of the above system for Δv using a Gauss-Seidel step. We first compute the new velocity $v(t + \Delta t) = v(t) + \Delta t \cdot \Delta v$ and the change in position $\Delta x = \Delta t \cdot v(t + \Delta t)$ of this vertex. Then, we trace the vertex on the surface into the direction Δx by a distance equal to $\|\Delta x\|_2$. Finally, $v(t + \Delta t)$ and Δv have to be adjusted according to the vertex movement to ensure consistency with the mass-spring system. In order to prevent the grid from moving due to the mass-spring relaxation, we never update the position of the center of the grid.

4.2 Spring Topology

In this work we use an entangled topology for the mass-spring system as shown in Figure 7.

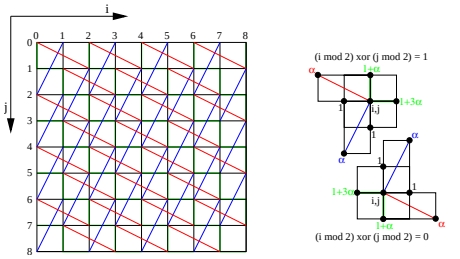

Figure 7: The entangled topology of the mass-spring system used for grid relaxation together with its connectivity rules.

This particular topology has the following beneficial properties. Firstly, it is regularly 6-connected. Thus, only six forces have to be gathered per mass point. Furthermore, like all regular topologies, it

can be stored implicitly. Secondly, it can be vertex colored using only two colors. Consequently half of the vertices can be updated in parallel, resulting in the high degree of parallelism needed to efficiently map the solver to the GPU. Thirdly, the red and blue springs in Figure 7 effectively resolve folds and avoid new folds to emerge, because the existence of folds implies that some springs are not at rest length.

To keep the system isotropic, we choose a relative stiffness (and damping) weight of 1 for the black springs. For the other springs, weights are chosen such that the products of spring directions and weighting factors sum up to 0. The system thereby has a single free parameter α to weight the red and blue springs against the black ones.

4.3 GPU Implementation

The main challenge in the realization of mass-spring systems on the GPU is to overcome the mutual exclusion of read/write accesses to the same buffer in current graphics APIs. Therefore, we use a gathering approach [15, 32, 8], which collects the spatial information about adjacent grid points via texture fetches and then performs the position update in a pixel shader.

Our implicit constrained Gauss-Seidel solver requires read access to the first $k-1$, already updated elements when updating the k^{th} element. Since our topology has a chromatic number of 2, we can reduce the number of necessary rendering passes from one pass per vertex (naïve implementation) to just two. This is because all vertices with the same color only depend on vertices of the other color, and can consequently be processed in parallel.

Therefore, we group vertices based on their color into sets of buffers. Each set contains positions x, velocities v, and velocity updates Δv for a single color. Furthermore, due to the mutual R/W exclusion, another set of output buffers is needed. Since the topology is regular, it is never stored explicitly. Table 1 sketches our algorithm.

Since half of the vertices can be processed in parallel, our implementation greatly benefits from computation and memory parallelism on recent GPUs (also see Section 6). However, since pivoting cannot be performed easily on the GPU, slightly more iterations when compared to a CPU implementation with pivoting are necessary to meet the same accuracy. In our tests, these additional iterations prooved to be negligible in terms of performance.

Table 1: Algorithm for grid relaxation

For each vertex color c (rendering pass)
  For each mass point x of color c, generate a fragment.
  For each fragment do (pixel shader):
    1. Read positions of masses in the 1-ring neighborhood.
    2. Calculate force f_x acting on x according to Hooke's Law and Rayleigh damping.
    3. Calculate derivatives $\partial f/\partial x$ and $\partial f/\partial v$ (Derivatives can be determined analytically [1]).
    4. Perform block Gauss-Seidel step to compute Δv_x.
    5. Trace distance $dt(v_x + \Delta v_x)$ on the base surface to update position of x.
    6. Re-compute v and Δv based on updated x.
    7. Write updated x, v, and Δv to an output buffer.
  Swap old input buffer and output buffer.

4.4 Relaxation Quality

To assess the quality of the relaxed grids, we use a quality metric of the form

$$g = 50\left(1 - \frac{\sigma(A_i)}{\mu(A_i)} - \frac{1}{360^\circ}\sum_{i,j}|\beta_{ij} - 90^\circ|\right),$$

where $\mu(A_i)$ is the average area of the quadrangular grid cells with a standard deviation of $\sigma(A_i)$. Here, β_{ij} refers to the j^{th} interior angle of cell i. The metric measures isometry by placing a penalty on non-right angles and area deviations. The constants are chosen empirically such that positive values (up to a maximum of 10) correspond to acceptable visual quality.

We further use the metric g to steer the parameters of the Gauss-Seidel solver by requiring that g increases monotonically in each iteration. If this is not the case, a partial restart of the mass-spring system is performed. Denoting the state prior to the last iteration by x_{old} etc., we set $x \leftarrow x_{\text{old}}, v \leftarrow -\frac{1}{2}v_{\text{old}}, \Delta v \leftarrow -\Delta v_{\text{old}}$. This restores the last positions and sets v and Δv such as to step back a little further in the simulation. Then, the solver is resumed, but with $\Delta t \leftarrow \frac{1}{2}\Delta t_{\text{old}}$ and 5% higher damping factors. On the other hand, if from one relaxation step to the next progress was made we increase Δt by 10% and decrease the damping factors by 2%. These values have been found and validated empirically. Note that due to these restarts the system can never diverge. Furthermore, a gain of about 2 points in g is typically achieved—independent of the grid resolution—when compared to the best possible set of static Gauss-Seidel parameters.

5 Rendering

For each vertex of the displacement grid, a full tangent frame is computed via barycentric interpolation from properties of the base mesh. These frames are then re-orthogonalized by using their normal as reference. Since the geometry decal also stores a local tangent frame to reflect the geometry of the displacement field, we can transform the per-vertex normal using these two frames in order to obtain a properly aligned normal on the geometry decal.

Displacements along the outward-pointing normal of the base mesh can be rendered using standard depth testing. In constrast, displacements along the negative normal direction can remove parts of the base mesh. They can be rendered using depth peeling [11]; however, this reduces the performance if the base mesh has a high depth complexity. To overcome this drawback, we present a new method to render these displacements by using a cutout mask. This mask is used to exclude parts of the base mesh from rendering on a per-fragment basis. It is similar to the trim texture used by Guthe et al. [16] but does not require any re-tesselation.

If the base mesh is fully parameterized, the cutout mask is generated by rendering the displacement grid into a 2D render target. We replace each vertex' coordinates by its texture coordinates carried over from the base mesh. Thereby, the displacement grid leaves an "imprint" in the texture domain of the base mesh. Note that if the parameterization of the base mesh contains discontinuities, grid triangles have to be clipped against these seams to avoid artifacts.

If no global parameterization of the mesh is known, we can still utilize this technique, but the local parameterization of each decal is used as domain for the cutout mask. Note that in case of such local cutout masks each mask is static and can be computed in advance.

In either case, the cutout mask can be mapped to the base mesh and all fragments covered by the displacement grid can be discarded. If local cutout masks are used, multiple fetches from these masks have to be performed. To avoid undesirable artifacts due to the finite resolution of the cutout mask, we slightly increase the size of the grid by padding it with zero displacement values before rendering it into the cutout region. As can be seen in Figure 8 no artifacts are visible, although the texel aspect ratio is about 1:2. By choosing a sufficiently high resolution of the cutout mask, sub-pixel precision can be achieved.

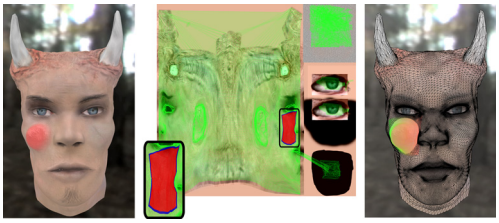

Figure 8: From left to right: Result of adding a bump (red) to a mesh, the cutout mask in texture space, and the wireframes of the left image. In the middle, the patch area (red) can be distinguished from the padding necessary to avoid artifacts (blue).

In a final pass, the geometry decal is rendered by displacing vertices of the grid in the vertex shader. It is intrinsically clear that the geometry decal can be animated in several ways. Firstly, a sequence of displacements can be used. Secondly, the geometry decal can move over the mesh by moving the pick point p_c. Thirdly, the base mesh can be animated if its topology is known throughout the process. Note that the decal has only to be fully traced and relaxed in each animation step for the latter two animation modi. For these modi, a global cutout mask has also to be generated in each animation step

6 Results

We validated our approach on an Intel Core2Duo 6600 at 2.4GHz with 2GB RAM and an NVIDIA GeForce 280GTX. Rendering was performed to an $8\times$ anti-aliased 1600×1200 viewport.

In our tests, we used two models, a chess board (12 triangles) and the Mannequin from Aim@Shape (reduced to 32000 triangles, see Figure 1). Both models are textured, including a bump and an environment map. We measured the times needed to trace, relax, and render a single geometry decal with resolutions varying from 32^2 to 512^2. Furthermore, we measured the time required to generate a 4096^2 global cutout mask. For local cutout masks, rendering times are by about 0.3 ms worse per decal than stated in Table 2. Last but not least, we measured the time required by a single mass-spring step as well as the number of iterations needed for convergence. In our experiments, the

GPU-based mass-spring system was consistently about $17\times$ faster than a carefully tuned CPU version. CPU timings are thus omitted from the table.

The overall performance is strongly affected by the layout phase that includes tracing, resolving folds, and propagation of appearance properties. This step is faster for the chess board, since less crossings of vertices over mesh edges have to be computed. As expected, the mass-spring system converges in a single step and performs no updates on the planar chess board.

The second column in Table 2 shows that rendering the base mesh with geometry decals is extremely fast even for high-resolution displacement meshes. As can be seen, our method renders multiple medium-size geometry decals at fully interactive rates. For resolutions up to 128^2 even a moving decal can be handled at fully interactive rates.

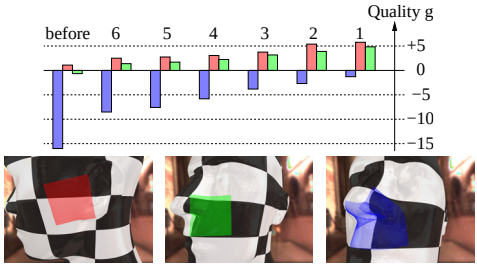

Figure 9: Top: Quality of our parameterization after 60 iterations. Vertices were constrained to the surface after every n^{th} step (ranging from 6 to 1). Bottom: The bar color corresponds to the initial grid placement.

In Figure 9, we show a detailed analysis of the convergence of the constrained Gauss-Seidel solver. For three different 64^2 grids, the quality gain due to 60 relaxation steps using the entangled topology is shown. We varied the frequency at which vertices are constrained to the surface. Clearly, the bar graph indicates the benefits of ensuring hard constraints as soon as possible, making our Gauss-Seidel solver superior to other solvers. Figure 10 depicts convergence plots of 36 iterations of our solver for the same three grids. As can be seen, the relaxation step improves the quality of the grid significantly, even if not all folds could be resolved (blue) due to the high curvature of the base mesh.

For our mass-spring system, we initially set all masses to 3.9, damping and stiffness constants to $\kappa_d = 15$ and $\kappa_s = 59$, and the step-size to $\Delta t = 0.002$. Note that κ_d as well as Δt are adapted dynamically by our approach.

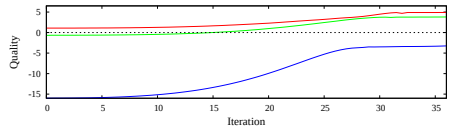

Figure 10: Convergence plot of the three grids shown in Figure 9.

6.1 Limitations

Currently, our approach is restricted to displacements described as height fields. However, it is possible in theory to extend the local parameterization to a thin shell around the mesh to allow full 3D displacements. Furthermore, we are limited to small, non-overlapping decals. Overlapping decals can in theory be resolved if a global parameterization is known, but still their extent has to be appropriate with respect to the local curvature of the base mesh. Also, intrusions so large that they pierce the back of the mesh are currently not resolved. Animations of the base mesh must either not change the topology, or a smooth mapping of surface points from one frame to the next must be provided. The reason is that the decal placement has to be properly resolved in each step.

7 Conclusion and Future Work

Our experiments show that the proposed method works well in practice and interactive frame rates can be achieved for the positioning, editing, rendering, and animation of highly detailed geometry decals.

Our method works best if the geometric features to be added are small in comparison to geometric features of the base mesh. On the other hand, if the surface has features on a similar scale as the displacements, displacement mapping as such is ill-posed. In the presence of such small features, our method fails to resolve all folds in the initial grid layout in the worst case. Also, as with essentially all displacement techniques, the local curvature radius limits the maximum height of displacements.

In the future we would like to investigate GPU-friendly methods to obtain the initial displacement grid, including a method to resolve folds. Currently, this process consumes a significant amount of time,

Table 2: Performance evaluation using the Mannequin and the chess board (the latter ones in parentheses).

Grid	Layout	Rendering	Cutout Mask	Mass-Spring / step	# steps	#Triangles
-	- (-)	2.0ms (1.7ms)	- (-)	- (-)	0 (0)	32K (12)
32^2	3.2ms (1.5ms)	2.2ms (1.8ms)	0.3ms (0.3ms)	0.05ms (0.05ms)	24 (1)	34K (2K)
64^2	5.8ms (4.0ms)	2.4ms (2.0ms)	0.4ms (0.4ms)	0.25ms (0.23ms)	37 (1)	40K (8K)
128^2	18.0ms (15.2ms)	3.2ms (2.8ms)	0.7ms (0.7ms)	0.95ms (0.89ms)	53 (1)	64K (32K)
256^2	63.8ms (61.2ms)	7.0ms (6.9ms)	2.2ms (2.2ms)	3.08ms (2.94ms)	74 (1)	160K (128K)
512^2	272.7ms (234.0ms)	22.2ms (20.0ms)	7.8ms (7.8ms)	10.27ms (9.97ms)	112 (1)	544K (512K)

and we consequently hope to improve the overall performance of our technique by an efficient realization on recent GPUs.

References

[1] D. Baraff and A. Witkin. Large steps in cloth simulation. In *ACM SIGGRAPH*, volume 32, pages 43–54, 1998.

[2] J. F. Blinn. Simulation of wrinkled surfaces. *ACM Computer Graphics*, 12(3):286–292, 1978.

[3] T. Boubekeur and C. Schlick. A flexible kernel for adaptive mesh refinement on GPU. *Computer Graphics Forum*, 27(1):102–114, 2008.

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

[5] R. L. Cook, L. Carpenter, and E. Catmull. The reyes image rendering architecture. *ACM Computer Graphics*, 21(4):95–102, 1987.

[6] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, 21(3):209–218, 2002.

[7] M. Desbrun, M. Meyer, and A. H. Barr. *Cloth Modeling and Animation*, chapter Interactive Animation of cloth-like Objects in Virtual Reality, pages 219–239. A.K. Peters Ltd., 2000.

[8] C. A. Dietrich, J. L. D. Comba, and L. P. Nedel. *ShaderX 5 - Advanced Rendering Techniques*, chapter Storing and Accessing Topology on the GPU: A Case Study on Mass-Spring Systems, pages 565–578. Charles River Media, 2006.

[9] M. Doggett and J. Hirche. Adaptive view dependent tessellation of displacement maps. In *ACM/EG Workshop on Graphics Hardware*, pages 59–66, 2000.

[10] G. Elber. Geometric texture modeling. *IEEE Computer Graphics and Applications*, 25(4):66–76, 2005.

[11] C. Everitt. Interactive order-independent transparency. Technical report, NVIDIA White papers, 2001.

[12] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geomtric Design*, 14(3):231–250, April 1997.

[13] M. S. Floater and K. Hormann. *Advances in Multiresolution for Geometric Modelling*, chapter Surface Parameterization: a Tutorial and Survey, pages 157–186. Springer, first edition, 2004.

[14] K. Gee. Direct3D 11 tesselation. Talk at the Microsoft Gamefest Conference, 2008.

[15] J. Georgii and R. Westermann. Mass-spring systems on the GPU. *Simulation Modelling Practice and Theory*, 13:693–702, 2005.

[16] M. Guthe, A. Balázs, and R. Klein. GPU-based trimming and tessellation of NURBS and T-spline surfaces. *ACM Transactions on Graphics*, 24(3):1016–1023, 2005.

[17] K. Hormann, A. Sheffer, B. Lévy, M. Desbrun, and K. Zhou. Mesh parameterization: Theory and practice. ACM SIGGRAPH 2007 Course Notes.

[18] X. Huang, S. Li, and G. Wang. Displacement modeling: Hardware-accelerated interactive feature modeling on subdivision surfaces. *The Visual Compututer*, 23(9):861–872, 2007.

[19] X. Huang, S. Li, and G. Wang. A GPU based interactive modeling approach to designing fine level features. In *ACM Graphics Interface*, pages 305–311, 2007.

[20] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, and S. Tachi. Detailed shape representation with parallax mapping. In *ACM Artificial Reality and Telexistence*, volume 11, pages 205–208, 2001.

[21] J. Kautz and H.-P. Seidel. Hardware accelerated displacement mapping for image based rendering. In *Graphics Interface*, pages 61–70, 2001.

[22] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 2002.

[23] F. Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE TVCG*, 4(1):55–70, 1998.

[24] M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *ACM SIGGRAPH*, volume 27, pages 359–368, 2000.

[25] H. Pedersen. A framework for interactive texturing on curved surfaces. In *ACM SIGGRAPH*, pages 295–302, 1996.

[26] F. Policarpo, M. M. Oliveira, and J. a. L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. *ACM Transactions on Graphics (I3D)*, 24(3):935–935, 2005.

[27] R. Schmidt, C. Grimm, and B. Wyvill. Interactive decal compositing with discrete exponential maps. *ACM Transactions on Graphics*, 25(3):605–613, 2006.

[28] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov. ABF++: fast and robust angle based flattening. *ACM Transactions on Graphics*, 24(2):311–330, April 2005.

[29] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Foundations and Trends ® in CG and Vision*, 2(2):105–171, 2006.

[30] L.-J. Shiue, I. Jones, and J. Peters. A realtime GPU subdivision kernel. In *ACM SIGGRAPH*, volume 24, pages 1010–1015, 2005.

[31] V. Surazhsky and C. Gotsman. Explicit surface remeshing. In *EG Symposium on Geometry Processing*, volume 1, pages 17–28, June 2003.

[32] E. Tejada and T. Ertl. Large steps in GPU-based deformable bodies simulation. *Simulation Modelling Practice and Theory*, 13:703–715, 2005.

[33] A. Vlachos, J. Peters, C. Boyd, and J. L. Mitchell. Curved PN triangles. In *ACM I3D*, pages 159–166, 2001.

[34] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. View-dependent displacement mapping. *ACM Transactions on Graphics*, 22(3):334–339, 2003.

[35] X. Wang, X. Tong, S. Lin, S.-M. Hu, B. Guo, and H.-Y. Shum. Generalized displacement maps. In *EG Symposium on Rendering*, pages 227–234, 2004.

[36] R. Zayer, C. Rössl, and H.-P. Seidel. Setting the boundary free: a composite approach to surface parameterization. In *ACM/EG Symposium on Geometry Processing*, volume 3, page 91, 2005.

[37] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3D: an interactive system for point-based surface editing. *ACM Transactions on Graphics*, 21(3):322–329, 2002.

Note: This color-plate is only available in the digital version of this paper.

Figure 11: A water simulation on a circular domain is used to animate the displacement field. The simulation was pre-computed on a 256^2 grid and has 319 time steps. We achieve a rendering performance of over 140 frames per second on a 1600×1200 viewport.

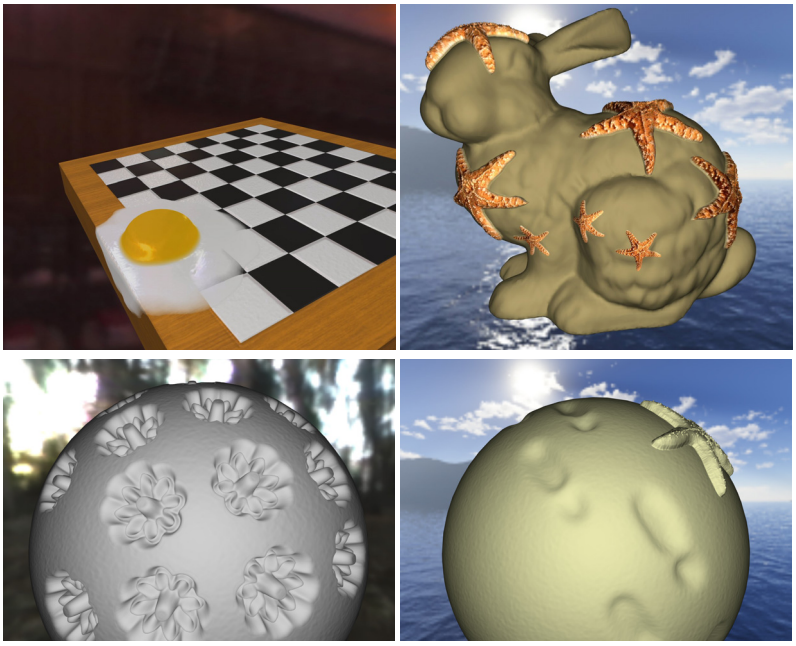

Figure 12: Using our cutout approach we achieve more than 15 fps on a 1600×1200 viewport rendering the following scenes. Top left: Using rotations to propagate trace directions does not fail at acute angles. Top right: A 69K triangle bunny mesh displaces by starfishes (512^2 each). Bottom left: A sphere displaced by 20 instances of a flower decal (128^2). Bottom right: A sphere displaced by one starfish (512^2) and ten footsteps (160×378 each).