Displacement Mapping with an Augmented Patch Mesh

SUMMARY  Displacement mapping has been widely used for adding geometric surface details to 3D mesh models. However, it requires sufficient tessellation of the mesh if fine details are to be represented. In this paper, we propose a method for applying the displacement mapping even on coarse models by using an augmented patch mesh. The patch mesh is a regularly tessellated flat square mesh, which is mapped onto the target area. Our method applies displacement mapping to the patch mesh for fitting it to the original mesh as well as for adding surface details. We generate a patch mesh, which stores three-dimensional displacements from the patch mesh to the original mesh. A displacement map is also provided for defining the new surface feature. The target area in the original mesh is then replaced with the patch mesh, and the patch mesh reconstructs the original shape using the patch map and the new surface detail is added using the displacement map. Our results show that our method conveniently adds surface features to various models. The proposed method is particularly useful if the surface features change dynamically since the original mesh is preserved and the separate patch mesh overwrites the target area at runtime.

1. Introduction and Related Work

The methods for adding surface detail such as texture mapping, bump mapping, normal mapping, and displacement mapping have been widely used for efficiently adding realism to computer graphics renderings. Among these surface mapping methods, the displacement mapping enables geometric surface details by displacing the actual geometric positions of vertices. However, it normally requires enough tessellation of the mesh model for realizing fine details. In order to apply the displacement mapping to a coarse model, whose tessellation is not fine enough, we need to apply regular remeshing or adaptive tessellation of the model [1]–[4]. This requires a large vertex count or a precomputation, which might be a limitation in certain applications. For example, it would be difficult for implementing dynamically changing surface details in interactive applications such as games or virtual reality systems. Suppose that we make a first-person shooter game. Then we might want to implement bullet marks, which requires the dynamic addition of holes at arbitrary positions based on the user inputs. Moreover, we need to restore the original mesh when the game is restarted. Existing methods for adding surface details such as the shell map [5], [6] or mesh editing methods [7]–[9] either can represent only convex features or change the original meshes. Schäfer et al. [10] proposed a real-time displacement method for representing local deformations, and their method also uses a hardware tessellation.

In this paper, we propose displacement mapping with an augmented patch mesh for dynamically adding geometric surface details while preserving the original geometry of the 3D mesh model. The proposed method enables surface details to be added to the selected region of the mesh model by mapping an additional patch mesh onto the target area and applying the displacement mapping. Our method preserves the original mesh and allows to add fine geometric details on the coarse mesh models as well as the fine models, since the separate patch mesh replaces the target area at the rendering time. Similar to the conventional displacement mapping, we can apply our method by simply defining the target region provided with a displacement map, a texture map, and the texture coordinates. Especially, our method would be useful for generating objects whose surface details are dynamically changed in real-time. Because of its convenience, our method can interactively add or remove surface details on arbitrary regions regardless of the underlying mesh’s tessellation level. We believe that our method will be a powerful tool for adding realistic surface details in interactive systems such as games or virtual reality applications thanks to its simplicity and convenience.

2. Displacement Mapping with a Patch Mesh

Our method maps a patch mesh, which is a regularly tessellated flat square mesh, onto the mesh model for adding geometric surface details. Similar to the conventional displacement map, the patch mesh is mapped to the target area of the base mesh model by specifying the texture coordinates at the vertices. The user provides a displacement map for defining the new geometric detail and specifies the target area on the base mesh. Our method applies the displacement mapping to the patch mesh for fitting the patch mesh to the original geometry as well as adding a new geometric detail. For preventing the base mesh from occluding the patch mesh, we discard the pixels in the target area. This enables representing concave geometric features as well. The Fig. 1 shows the overall process of our method.

At the pre-computation time, our method generates a patch map, which stores the displacements for fitting the patch mesh onto the target area on the base mesh. These displacement calculations are done in the vertex and the frag-
Our method pre-computes the patch map for fitting the patch mesh to the base mesh by storing the displacements. At rendering time, our method replaces the target area with the patch mesh while restoring the original geometry by applying the patch map, and adds surface details by applying the provided displacement map.

At rendering time, our method puts the patch mesh on the target area and applies the displacement mapping by the pre-computed patch map as well as the pre-defined displacement map. The following sections explain these processes in detail.

2.1 Patch Mesh

The patch mesh is a regularly tessellated flat square mesh with a unit size. The patch mesh should be fine enough to generate a geometric detail with a displacement mapping. In this paper, we use a patch mesh in a 64 $\times$ 64 vertex grid. We use a single patch mesh which is applied to every displacement map. In this case, we can render the scene with multiple mappings with a small overhead since we can reuse the patch mesh stored in the GPU memory or vertex buffer objects.

2.2 Patch Map

The patch map stores the three-dimensional vector displacements for fitting the patch mesh to the target area. It is automatically generated based on the base model and the texture coordinates for the displacement mapping. It could be either pre-computed and stored in a file for the static scene, or computed once at runtime and stored in a frame buffer object for the dynamic scene.

For a given two-manifold base mesh $M$, we first define a mapping $F : M \mapsto R^2$ from $M$ to the texture space, by either a global or local parametrization method[11],[12]. Let $G : R^2 \mapsto M$ be the inverse of $F$, $N$ be the normal map of $M$, and $D$ be the displacement map for the surface detail on the target area. We define the target area $\Omega = \{(u, v)|u \in [u_{\min}, u_{\max}] \land v \in [v_{\min}, v_{\max}]\}$ on the texture space. The patch map is the vertex positions in $G(\Omega)$. The Fig. 2 shows the oil drum model and the corresponding patch map and the displacement map.

2.3 Rendering with Our Method

The Fig. 3 shows our rendering process. For rendering a model with our method, we render the patch mesh as well as the base mesh. Since the base model can occlude the patch mesh when the feature is concave, we discard the pixels in the target area of the base model. The vertices in the patch mesh are displaced based on the corresponding RGB color value in the patch map, for representing the geometry of the base mesh. They are displaced again by the displacement map for adding the surface detail. The normal vector for computing the illumination could be either computed at the fragment shader using adjacent pixel heights or pre-computed and stored in a normal map. These processes are as follows:

1. Render the base mesh $M$ with a fragment shader that discards the fragments with texture coordinates in the target area $\Omega$.

2. Render the patch mesh whose texture coordinates correspond to $\Omega$ with a vertex shader. For a given texture coordinate $(u, v)$ of a vertex at the vertex shader, we fetch the vertex position $\bar{p} = G(u, v)$ and normal vector $\vec{n} = N(u, v)$ of the base mesh, and the displacement $d = D(u, v)$. The output vertex position $\vec{p}'$ is computed as $\vec{p}' = \bar{p} + d\vec{n}$.

3. Results and Discussion

The Fig. 4 shows the results of our method. Our method
Fig. 4  The images show the results of our method on various models. They show the original models, the rendering results of our method, their wire-frames, and the displacement maps.

Our approach successfully adds fine details regardless of the based model, even on very coarse base models such as the oil drum and the plate models in Fig. 4 (a) and (e), as well as more complex models including the Verdi and the Bunny in Fig. 4 (b) and (c). In these models, the fine patch mesh is fit to the coarse model representing the fine details. As you can see in the Verdi and the Bunny model, the patch mesh can nicely fit the base model as well as represent the new feature. However, there might be slight gaps or Z-fighting issues at the boundary of the patch mesh since the mesh structure of the patch mesh is different from the base mesh. Resolving this issue would be an interesting future research, and one possible approach would be zittering the patch mesh vertices for moving them close to the base mesh vertices and blending two meshes around the boundary. Our method also allows concave features (Fig. 4 (a)–(d)) as well as convex features (Fig. 4 (e)). Even though the patch mesh is displaced below the based model as shown in Fig. 4 (a), the patch mesh is correctly rendered because the pixels on the target area are discarded and replaced with the patch mesh. The Fig. 5 shows a screen capture of interactive displacement mappings generating multiple features in a dynamic scene. The multiple patch meshes have a limitation that if patch meshes are overlapped, the patch map might not be able to represent the un-
The left image shows the multiple displacement mapping with our method on the wall for representing bullet marks, and the right image shows its wire-frame rendering. Our method allows interactive displacement mappings on arbitrary positions in dynamic scene.

Table 1. Rendering times for multiple displacement mappings with our method.

| Model       | NO map | 1   | 10  | 50  | 100 |
|-------------|--------|-----|-----|-----|-----|
| oil drum (892) | 0.120  | 0.120 | 0.140 | 0.600 | 1.500 |
| verdi (12K)    | 0.850  | 0.920 | 0.964 | 1.321 | 1.973 |
| box (14K)      | 1.027  | 1.036 | 1.098 | 1.777 | 1.964 |
| angel (24K)    | 1.688  | 1.759 | 1.938 | 2.768 | 3.250 |
| bunny (70K)    | 5.473  | 5.580 | 6.286 | 7.964 | 10.098 |

Table 2. Run times for generating patch maps.

| Model     | #tris | time (ms) |
|-----------|-------|-----------|
| oil drum  | 892   | 0.112     |
| verdi     | 12K   | 0.778     |
| box       | 14K   | 0.937     |
| angel     | 24K   | 1.639     |
| bunny     | 70K   | 5.041     |

Our method allows interactive displacement mappings on arbitrary positions in dynamic scene.

![Image](image_url)

The underlying geometry well since the patch map fits to the original mesh without considering other patch meshes. Solving this problem would be another interesting extension and a possible solution would be including previous patch meshes when we compute the patch map.

The Table 1 shows the run times for rendering multiple displacement mappings with our method compared according to the number of patches. The results show that our method is scalable to the number of patches. For multiple displacement mappings, we use a single patch mesh for all the patch maps. The Table 2 shows the run times for generating patch maps. The result shows that the time for generating a patch map is proportional to the vertex count of the base model.

4. Conclusions

We presented a method for patching selected parts of a 3D mesh that combines patch meshes with the three-dimensional vector displacement mapping method. Compared to existing methods like the conventional displacement mapping, our method can add surface details independent of the tessellation level of the original mesh. Since the proposed method can be used for conveniently adding surface details of a 3D mesh model while preserving the base mesh, it can well handle dynamic surface changes.

Our method also can generate convex as well as concave features by discarding the target area of the base model and replacing it with the displaced patch mesh. Also, it can generate diverse geometric features by using the vector displacement mapping techniques.

In addition to adding surface details, our method could also be used for removing or replacing surface features. The patch map represents the geometry of the base model. Therefore, if we apply image processing techniques to the patch map, we could edit or remove the original surface features. This approach would be an interesting future research.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2013R1A1A2060582), and also by the Chung-Ang University Research Scholarship Grants in 2008.

References

[1] A. Lee, H. Moreton, and H. Hoppe, “Displaced subdivision surfaces,” Proc. SIGGRAPH ’00, pp.85–94, 2000.
[2] J. Kautz and H.P. Seidel, “Hardware accelerated displacement mapping for image based rendering,” Proc. Graphics Interface, pp.61–70, 2001.
[3] J. Hirche, A. Ehlerl, S. Guthe, and M. Doggett, “Hardware accelerated per-pixel displacement mapping,” Proc. Graphics Interface, pp.153–158, 2004.
[4] L. Szirmay-Kalos and T. Umenhofer, “Displacement mapping on the GPU-state of the art,” Computer Graphics Forum, vol.27, no.6, pp.1567–1592, 2008.
[5] S.D. Porumbescu, B. Budge, L. Feng, and K.I. Joy, “Shell maps,” ACM Trans. Graphics, vol.24, no.3, pp.626–633, 2005.
[6] S. Jeschke, S. Mantler, and M. Wimmer, “Interactive smooth and curved shell mapping,” Proc. Eurographics conference on Rendering Techniques, pp.351–360, 2007.
[7] H. Biermann, I. Martinez, F. Bernardinz, and D. Zoriny, “Cut-and-paste editing of multiresolution surfaces,” ACM Trans. Graphics, vol.21, no.3, pp.312–321, 2002.
[8] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.Y. Shum, “Mesh editing with poisson-based gradient field manipulation,” ACM Trans. Graphics, vol.23, no.3, 2004.
[9] K. Takayama, R. Schmidt, K. Singh, T. Igarashi, T. Boubekure, and O. Sorkine, “Geobrush: Interactive mesh geometry cloning,” Computer Graphics Forum, vol.30, no.2, pp.613–622, 2011.
[10] H. Schäfer, B. Keinert, and M. Stamminger, “Real-time local displacement using dynamic GPU memory management,” Proc. 5th High-Performance Graphics Conference, pp.63–72, 2013.
[11] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, “Least squares conformal maps for automatic texture atlas generation,” ACM Trans. Graphics, vol.21, no.3, pp.362–371, 2002.
[12] R. Schmidt, C. Grimm, and B. Wyvill, “Interactive decal compositing with discrete exponential maps,” ACM Trans. Graphics, vol.25, no.3, pp.605–613, 2006.