GPU を用いたアダプティブな分割によるリアルタイムディスプレースメントマッピング

Szego Zoltan　　金森　由博　　西田　友是

東京大学

1. Introduction

Displacement mapping is a rendering technique that adds high-quality details to a 3D model by displacing the surface geometry according to some information such as a height map. In recent years, Doggett et al [1] proposed a method to adaptively subdivide triangles using a recursive procedure; however, the recursive nature of this algorithm makes implementation on conventional GPUs difficult. In this paper, we show a method that makes use of the latest features of modern GPUs, and can perform adaptive subdivision in real time. We demonstrate its effectiveness in several applications such as level-of-detail (LOD) control of displacement mapped meshes.

2. Related work

There are several techniques to enhance the realism of low-polygon 3D models at a relatively low rendering cost. The technique most widely in use today in mainstream games and applications is *normal mapping* [2], which alters the shading for each pixel on the low-polygon mesh according to a normal map generated from its high-polygon version. This gives the illusion of high detail on the mesh, however, every face remains flat and therefore the shading looks incorrect for bumps at an oblique angle.

Parallax mapping [3] and its numerous variants displace the texture coordinates on the low-polygon mesh according to a height map to give the illusion of depth on the bumps on the surface. This gives convincing results on the inside of the mesh; however, its silhouette still retains its coarse polygonal look.

3. Displacement mapping

Unlike the previous techniques, *displacement mapping* not only changes the shading on the surface of the object, but actually moves the surface geometry to its appropriate location according to a displacement function, usually specified by a height map texture. There are two main approaches to performing displacement mapping, per-pixel evaluation [4] and tessellation. In this paper we focus on the tessellation approach, because determining the displacement at each pixel has a high computational cost which increases with the on-screen size of the object, and does not offer any control over the LOD if a perfectly accurate evaluation of the outline at every pixel is not required.

Real-time Displacement Mapping on the GPU using Adaptive Tessellation

Zoltan Szego, Yoshihiro Kanamori and Tomoyuki Nishita

The University of Tokyo

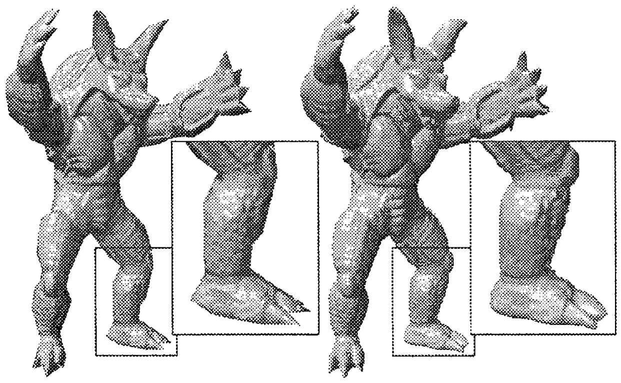

Figure 1. Left: a low-polygon version of the Armadillo model shaded with normal mapping. Right: the same model with displacement mapping.

Doggett et al [1] proposed a method to adaptively tessellate a mesh according to certain criteria with regard to the height map: the edges of each triangle are split if there is enough variation in the corresponding local normal vector or the average local height. Additionally, a screen-space check for short edges ensures that triangles which are small enough on the screen do not get split any further.

4. GPU implementation using geometry shaders

The method proposed by Doggett et al depends on some additions to the traditional graphics pipeline to accommodate for the recursive way in which the triangles are subdivided. While no such hardware exists, the newest series of GPUs, supporting a set of functionality called Shader Model 4, do contain an additional stage in the pipeline called the *geometry shader*, which can be used to create triangles on-the-fly. Our method uses this functionality to implement the above described adaptive subdivision algorithm.

Since the amount of output allowed from a geometry shader is limited, we must perform the subdivision in several passes in the algorithm used for the GPU implementation, using two vertex buffers to move the intermediate data back and forth ("ping-ponging"). The *stream-out* functionality of Shader Model 4 GPUs can be used to save the results of the geometry shader back into a vertex buffer in video memory, bypassing the CPU entirely.

A height map, a normal map and a summed area table (SAT) of the values in the height map are required for the adaptive subdivision algorithm. The SAT is used to compute the average height for any area on the height map. These are accessed as textures from the geometry shader. There are eight different possible subdivision patterns for a triangle depending on which of its three edges need to be split, these patterns are also stored in a texture which is indexed by a 3-bit

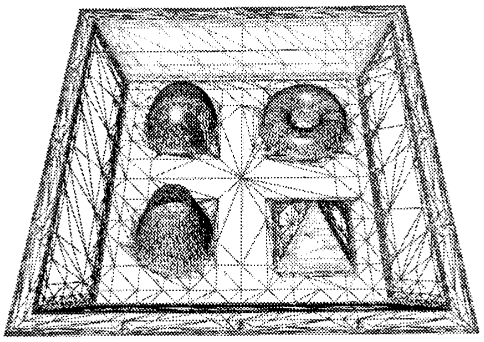

Figure 2. Results of adaptive subdivision of a simple input mesh – a quad consisting of two triangles.

number, with each bit corresponding to an edge of the triangle in question. Edges are split the same way for neighboring triangles, therefore ensuring that there are no gaps in the resulting mesh.

Triangles are subdivided in several passes, and then displaced according to the height map in the last pass, after which they can be rasterized and shaded, for example, by normal mapping.

5. Results

See Figure 1 for an example of the algorithm being applied to a mesh that was simplified from a high-polygon version, with the matching displacement map created from the original detailed mesh. The silhouette follows the original mesh accurately compared to the coarse look of the low-polygon version. The model's leg is magnified for a more detailed comparison with the normal mapped version.

Figure 2 shows the tessellation result of adaptive subdivision. Flat areas of the displacement map end up with larger polygons, while areas needing more detail are subdivided as needed.

The amount of detail resulting from the subdivision is ultimately controlled by the user-specified thresholds for the maximum allowable error in the local average height and the normal vector. The screen-space edge length limit serves as a form of LOD control, ensuring that when the model is zoomed out, no unnecessary details are created (see Figure 3 for an example). We used a length limit of 10 pixels for all of our experiments.

One of the most attractive applications of our method is the efficient display of animated displacement mapped models. The original meshes corresponding to the low-polygon version might contain hundreds of thousands of vertices, making their animation inefficient, while their normal-mapped counterparts retain the sharp edges and polygonal look of the simplified mesh. With our method, by animating only a small number of vertices and applying displacement mapping to the result, the high-quality mesh can be displayed at a reduced computational cost, with the benefits of LOD. See Table 1 for details on performance. The environment for our experiments was a machine with an Intel Core 2 Extreme CPU at 2.9 GHz and an nVidia GeForce 8800 GTX GPU.

Table 1. Performance comparison for the Armadillo model with four subdivision passes.

Rendered object	Input tris.	Output tris.	FPS static/anim.	
Original model	345944	345944	339	10
Uniform subdivision	800	204800	113	88
Adaptive subdivision	800	≒4400	714	98

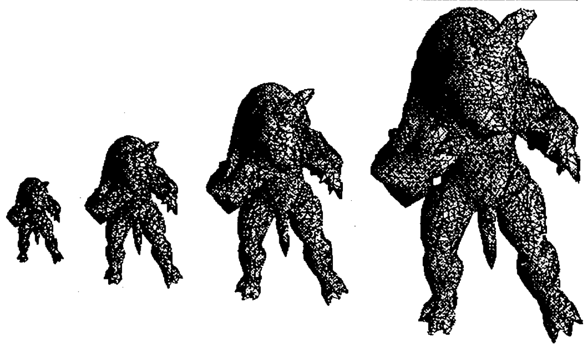

Figure 3. LOD control of the same pose in an animated mesh.

6. Conclusion and future work

In this paper we have described a method to perform displacement mapping in real time on the GPU using geometry shaders, and shown its effectiveness for applications such as the visual enhancement of animated low-polygon meshes.

For future research, we would like to find other possible algorithms for handling the adaptive subdivision, as well as other ways to speed up the processing on the GPU.

References

[1] M. Doggett and J. Hirche: Adaptive view dependent tessellation of displacement maps, Proceedings of the ACM SIGGRAPH / EUROGRAPHICS workshop on Graphics hardware, pp. 59-66, 2000

[2] P. Cignoni, C. Montani, R. Scopigno, C. Rocchini: A general method for preserving attribute values on simplified meshes, IEEE Visualization 1998, pp. 59-66

[3] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, S. Tachi: Detailed Shape Representation with Parallax Mapping, Proceedings of ICAT 2001, pp. 205-208.

[4] J. Hirche, A. Ehlert, S. Guthe, M. Doggett: Hardware accelerated per-pixel displacement mapping, Proceedings of the 2004 conference on Graphics interface, pp. 153-158, 2004.