Displacement and Normal Map Creation For Pairs of Arbitrary Polygonal Models Using GPU and Subsequent Model Restoration

Ilya Tisevich, Alexey Ignatenko
Department of Computational Mathematics and Cybernetics
Moscow State University, Moscow, Russia
ilya.t@mail.ru, ignatenko@graphics.cs.msu.ru

Abstract

Displacement maps were originally intended for mostly superficial alterations of polygonal models, while being artificially obtained by means of various raster graphics software.

Now, though, with rapid development of graphics hardware and its constantly increasing level of programmability it becomes possible to apply or simulate displacement mapping in real time, which renders it a very promising technique for making polygonal scene visualisations look even more realistic. It is also possible to utilize the displacement approach for lossy (or even lossless, considering significant data overhead) 3d-models compression algorithms. If combined with dedicated model compression methods and image and binary data compression algorithms, displacement mapping can result in dramatically improved overall compression ratios.

In this article a complete life cycle of differential displacement maps is described, including proposed algorithm of GPU-based map creation and an algorithm of model restoration. It is shown that the described algorithm is capable of all of the features of conventional raytracing-based algorithms, while significantly outperforming them in terms of speed and feedback times, which renders it much more suitable for real life development process. Also, a multi-level displacing approach is presented, clearing the possibilities of non-height-field model details representation.

Keywords: displacement map, polygonal model, subdivision, model restoration, level of detail, multi-level displacing, complexity reduction, GPU algorithms.

1. INTRODUCTION

1.1 Displacement maps and their use

Originally displacement maps were used to add desired detail to polygonal models, as it obviously is much less time-consuming to draw a needed depth-pattern in a raster or vector image editor than to carry out all the vertex operations by hand. It is also obvious that image editors are perfectly suited for the job, given evidence that there's a huge resemblance between model's texture and its displacement features nature. It was never considered a realtime application, mostly because it was computationally demanding, especially when used in combination with model animation algorithms, and resulting model complexity and data size were often huge.

While still being one of the modeling techniques, displacement mapping nowadays gathers more attention from realtime graphics developers, it seems that its main purpose is about to turn around, and here is why. In recent years there has been a growing interest in the use of displacement maps for realtime polygonal models rendering due to significantly increased graphics hardware programmability and performance. Several approaches has been developed, such as Real-Time Relief Mapping [Policarpo05], Steep Parallax Mapping [McGuire05], Per-Pixel Displacement Mapping with Distance Functions [Donnelly05], Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows [Tatarchuk06] and more. Figure 1 (courtesy of ATI) illustrates the visible differences between normal mapping (on the bottom) and parallax occlusion mapping (on the top). The pavement is represented by a flat low-polygonal model which is being accompanied by a normal map only (in case

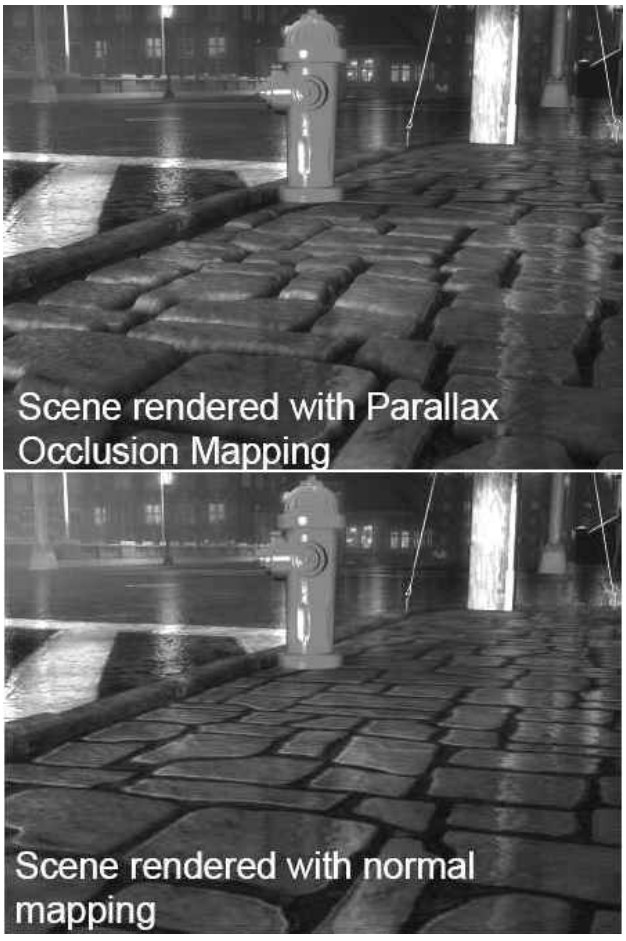

Figure 1. Parallax Occlusion Mapping vs. Normal Mapping

of normal mapping), and a combination of normal and displacement map (in case of parallax occlusion mapping). It is the displacement data which allows to simulate relief details, self-occlusion and self-shadowing and provide correct z-buffer depth values. And the best thing about these algorithms is that they don't require any structural model modifications whatsoever. With the introduction of Direct3D 10 and its improved architecture, which now includes a geometry shader [DirectX Dev], it also became possible to perform model subdivision and displacement routines on the fly, utilizing immense GPU capacities (for more details, see the model restoration section). With the new rendering algorithms it is easy to imitate surface microrelief, offering model detail level beyond the reach of preceding realtime rendering techniques, and thus raising the resulting image realism to a whole new level.

Displacement mapping approach is also suitable for polygonal models compression. The idea is to create a smooth-surface less-tessellated low-poly model based on an original complex high-poly mesh and extract a differential displacement data. Our tests show that even using a conventional lossless data compression algorithm, it is possible to notably reduce overall data size. Dedicated 3d-geometry compression techniques generally show better results with smoother input meshes, and the displacement data, stored as an image, could be subjected to any of the existing image compression algorithms, both lossy and lossless. At this point, it is worth mentioning that today's leading wavelet-based image compression methods, while achieving notably better compression ratios, are perfectly suitable for displacement maps compression, as they tend to produce blurry artifacts, compared to JPEG's rough blocking, which results in smoother restored meshes with no eye-catching artificial geometry noise.

1.2 Displacement map formats

There is no established standard for displacement maps, so their format and representation generally depend on the requirements of the task in hand.

The most common representation is based on the texture mapping approach, where each mesh vertex is bound with a pair of [0,1]-range values which provide a mapping into the conventional (s,t)-texture space of the displacement map, while the map itself is a two-dimensional image. Explicit mapping is usually omitted, which means that original vertex texture coordinates play double role as both texture and displacement map s,t-space vectors, although it's not always the case. The displacement data itself, as mentioned above, is stored in the raster image format and is sometimes a combined normal-displacement map with normal vectors data placed alongside displacement values in each pixel of the image. The image format used is usually a well-known 32 bits per pixel in 4 channels, although 16 bits per channel mode is also supported by modern hardware and is sometimes made good use of, in order to achieve greater precision. The possible formats, then, are as follows (precision given for 32bpp images):

- 8-bits displacement only. Displacement amount value only is stored, while displacement direction is determined by the surface normal direction in given point.
- 24-bit direction, 8-bit displacement. Three of the channels define the displacement direction in current point, while the fourth contains the displacement amount. It is possible to save additional 8 bits by storing x and y components of the displacement vector only, providing the vector is normalized, so that the third component could be later restored from the $z^2 = 1 - (x^2 + y^2)$ equation. Each of the two remaining vector components could be then represented by a 12-bit value, which allows for enhanced precision while requiring some additional computations.
- 16-bit direction 16-bit displacement. Direction vector consists of x and y components only, displacement amount is a 16-bit value.
- 8-bit displacement, 24-bit normal (not to be confused with displacement direction). The same as the very first format, but with embedded normal map for correct realtime surface shading.
- 2 x 8-bit displacement, 16-bit normal. Also known as Dual-Depth Relief Textures, used by Relief Mapping. Two displacement channels represent offset values from the imaginary upper and imaginary lower planes, allowing to simulate opaque closed-surface objects on a single relief-mapped triangle.
- 16-bit displacement, 16-bit normal. High-precision displacement amount value and a component-reduced normal vector.

1.3 Displacement map creation

To create a differential displacement map we need a pair of models — an original high detailed model (will be referenced as *high-poly* or *original* or *detailed*) and a probably smoothed, simplified low-poly model (will be referenced as either *low-poly* or *simplified*). If a low-poly model was not saved at one of the steps of model creation, or if model is not of handcrafted nature, a simplified model can be created by means of various geometry reduction tools, such as Rational Reducer [Rational] or 3D Studio MAX [Discreet]. To achieve better results it is sometimes crucial to apply a few passes of a surface smoothing algorithm before reducing the number of polygons.

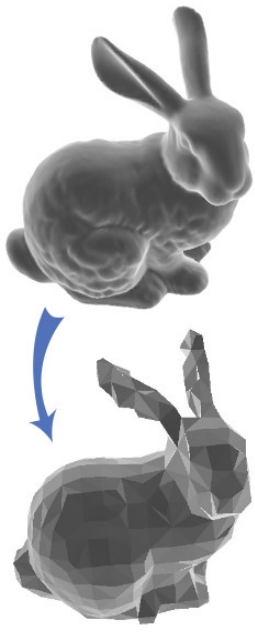

Figure 2.
Model simplification

The problem we are facing now is how do we obtain a proper displacement map, providing we are given both low-poly and high-poly models? There are numerous products on the market which can do the job, yet they all have their weaknesses — they either put too many restrictions on model's parameters and origin or are vastly time consuming. Sometimes they simply offer not enough flexibility, too.

The proposed algorithm utilizes the ever growing computing power and improved capabilities of modern graphics hardware together with specialized high level rendering optimization structures to offer fast and flexible technique of differential

displacement and normal map creation. All-round subsequent model restoration techniques are also described in the article. Figure 3 shows a basic displacement workflow, illustrating results achieved with the developed algorithms.

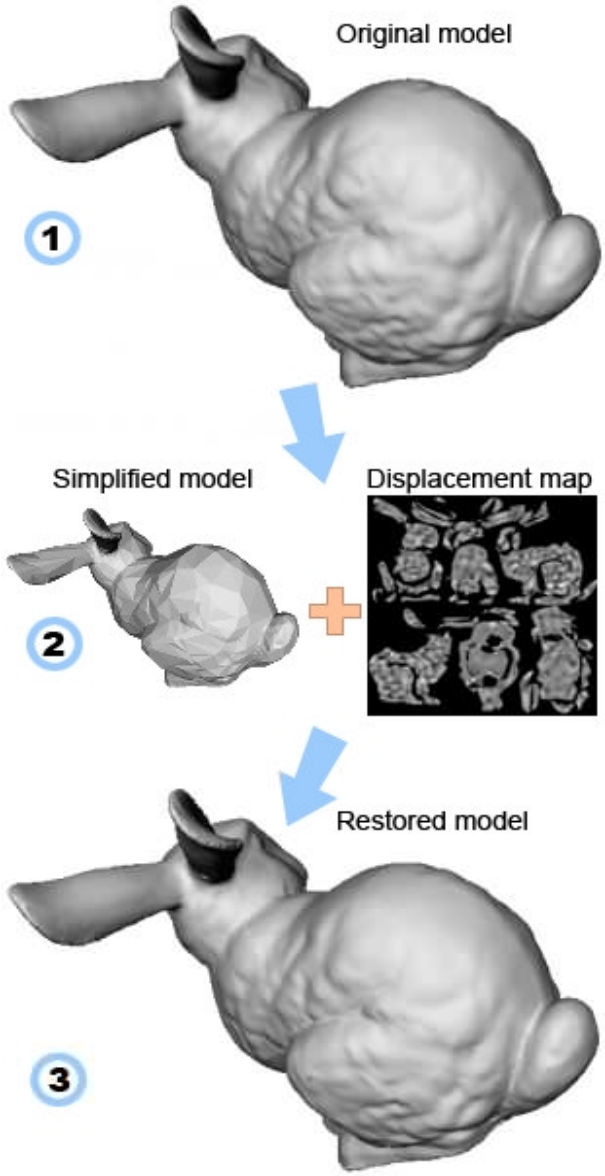

Figure 3. Displacement workflow

Section 2 describes existing algorithms and implementations, section 3 brings up the details of the proposed algorithm, while section 4 provides some knowledge on the technique's main features. Section 5 discusses possible model restoration approaches and is followed by section 6, which contains a results showcase. Section 7 concludes the article.

2. EXISTING ALGORITHMS

There are two general categories of differential displacement map creation techniques:

- techniques which exploit the precise information on models' vertex correspondence;
- techniques based on raytracing and render-to-displacement-map approach.

ZBrush 2 [Pixologic] falls into the first category, allowing to construct a precise displacement map in a couple of seconds, providing the application can establish a one-to-one correspondence (Figure 4) between all the vertices of the simplified model and some of the vertices of the original model.

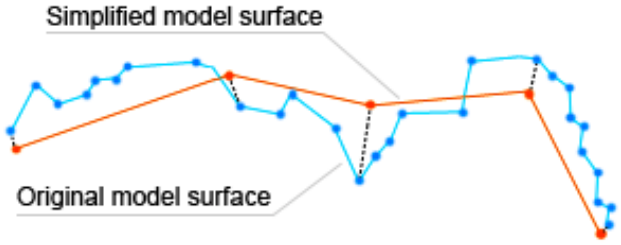

Figure 4. Established vertices correspondence.

Models created in ZBrush from scratch can be easily subjected to this condition either by saving a half-finished low-poly copy of the model in the middle of the sculpting process, or by reconstructing subdivision (i.e. simplifying the model) using built-in software functionality, but only in case the model's faces are stored as quads. Unfortunately, it is impossible to unambiguously figure out any correspondences between vertices of pairs of arbitrary polygonal models, which renders this approach virtually unusable for models obtained from laser 3d-scanners or any other geometry reconstruction devices, models represented by non-quadrant polygons or models created by means of any other modeling software.

This leads us to the second class of displacement calculation utilities — the ones built upon raytracing approach. Figure 5 shows principles of this technique: rays, corresponding to each

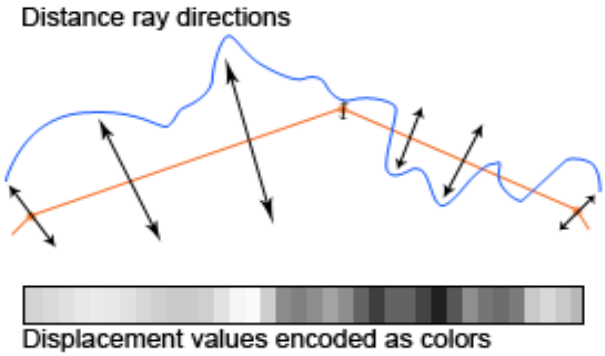

Figure 5. Casting rays to determine displacement amount.

pixel of the mapped displacement map area, are casted towards the direction of the normal vector in the corresponding point of the low-poly model surface to determine an actual displacement amount value. Given the fact that maximum displacement value is correctly set the way that high-poly backface surface is beyond the reach of the distance rays, this allows to accurately obtain desired measurements.

Several applications rely on this technique, including a 3D Studio MAX plugin, a Maya plugin (discontinued) and the ATI NormalMapper [ATI]. While these utilities put no close restraints

on input models, their output formats are limited by available render target formats of the raytracers in question. Moreover, the CPU-based raytracer implementations are pretty slow, taking up to a few hours to build a 4096x4096 displacement map for a pair of models with a below-the-average geometry complexity on a 2.4GHz single-core processor.

3. PROPOSED ALGORITHM

3.1 Input data

The algorithm takes two polygonal models as an input, presuming one of them is an original high-detail model and the second one is a simplified low-poly model. Apart from mandatory texture mapping presence for the vertices of the simplified model, the following restrictions of the displacement's height-field representation nature are applied:

- Texture mapping for the simplified model should be strictly unambiguous.
- There should be no ambiguity in the original-onto-simplified surface projection, which means that height-field alterations only can be represented. Although it is discussed later how to displace other model features using multistep deformations.

If the first requirement is not met, some displacement data loss will obviously occur. Multiple unwrapping techniques are available to get this problem sorted out, although it is important to understand their disadvantages while considering using either of them. Figure 6 illustrates three different mappings, (1) is a bi-plane front-face projection, (2) was created using a smart projection algorithm and then slightly adjusted manually to reduce the amount of wasted texture space. Mapping (3) is a simple packed-triangles unwrapping. The first approach is rather handy for texturing, while it is of little use for displacement purposes, as some of the polygons, settled at acute angles to the projection planes, are mapped into a handful of pixels, which means a significant data loss is to be expected. It is also obvious that more than $4/5^{th}$ of the texture area are wasted. Approach (3), on the contrary, maps all the faces into equal pieces in texture space, wasting very little of it, yet it is near impossible to do a proper texture job with such poly align. Among other drawbacks, there is an issue with texture filtering: closely located triangles will influence each other's borders once the map is subjected to filtering or mipmapping. Furthermore, relative triangle sizes are not taken into account, which means bigger faces with much more surface data to receive are granted equal texture space with tiny faces which probably need not to be displaced at all. From this point of view, the second (2) approach seems to be all things to all men, mapping acute-angled faces separately while using most of the texture area and properly packing adjacent faces, which fixes filtering issues and makes it rather easy to carry out the texturing job.

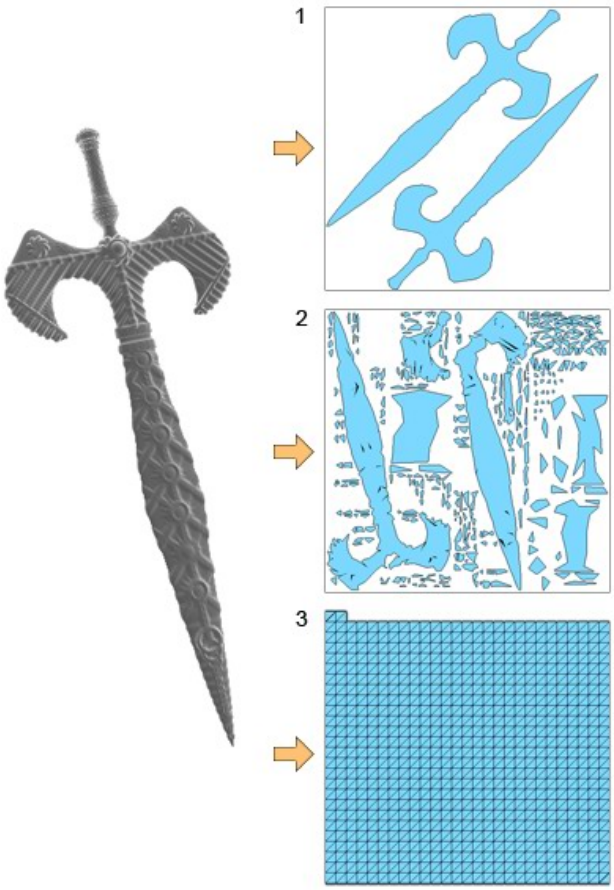

Figure 6. Consider different unwrapping strategies.

The projection restriction could be solved by altering the model simplification algorithm's bias, or by manually adjusting the low-poly model's surface in problem areas. While difficult to spot with a naked eye, this issue can be easily tracked using the proposed algorithm in the diagnostics mode.

3.2 Displacement and normal map creation

The main idea behind the algorithm is to utilize the rasterization approach and shift most of the job to the GPU, exploiting its specialized architecture and superior floating-point computation power. A virtual camera (or viewpoint) is positioned alongside a polygon of the simplified model, projection matrices are adjusted to transform the polygon's model space coordinates into displacement texture's s,t-mapping coordinates. Then a culling-wise chosen subset of the original model is rendered in such manner that the vertex and pixel shaders are able to compute displacement amount and original surface normal in given point and then save it as the color of the corresponding render target pixel (or fragment).

Generally speaking, the simplified model can consist of a number of subsets, each with its own texture mapping. Nevertheless, all the following details and explanations are given under assumption that the model consists of a single subset, and all its polygons are triangular, simply to avoid further complication.

Steps of the algorithm:

1. Preliminary step. Clipping tree is built.
2. For each polygon of the simplified model the following steps are taken (assuming i is the number of current polygon):
   a) Projection matrix M_p^i is built according to the current triangle's position and texture coordinates.
   b) Clipping tree is traversed to select branches (and later, individual triangles) which are not culled by the viewing pyramid defined by the matrix M_p^i and the near and far clipping planes. A preset *maximum displacement* value is used to filter out triangles located beyond the distance threshold. Selected data represents a subset of the high-poly model, which projects onto the current simplified model triangle.
   c) The selected subset is rendered using the transformation defined by the M_p^i projection matrix, normalized displacement value and (optional) normal direction data are calculated in the pixel shader. Resulting values are stored in the render target.
   d) Some combined map formats (see Introduction section for details) may require multi pass rendering, in which case step 2.c will be executed numerous times with altering shader settings.
3. Optional postprocessing and resulting map assembly in case of multipass rendering.

Figure 7 illustrates the projection-based clipping: a low-poly model triangle and an adjacent detailed model area are shown on the figure 7.1, selected high-poly model subset is painted light-green on the figure 7.2, while 7.3 illustrates a sketch of the resulting displacement map fragment. The fringe was successfully clipped in the pixel shader.

3.3 Clipping tree

Since it is extremely inefficient to rely on the GPU only in the task of geometry clipping, a dedicated spatial clipping structure was introduced. Its goal is to serve the following purposes: optional backface culling, projection frustum culling and projection distance clipping. It is also designed in such manner that intermediate computation structures for both original and simplified models can be cached, allowing to noticeably reduce, although already little, the tree computation time, once one of the models has changed.

The structure is based on a combination of an adaptive-subdivision-based modification of a conventional octree and a frustum clipping tree, allowing to provide exactly the needed model subset for every projection setup during the displacement map building process.

First, an octree is built, which settings may vary depending on such parameters as both low-poly and high-poly models grid densities, high-poly model elongation, spatial orientation and position of a particular model region. Each of the octree's nodes provides a bounding sphere for quick plane tests. Then a combined projection clipping tree is created, using projection frustum settings of each polygon of the simplified model.

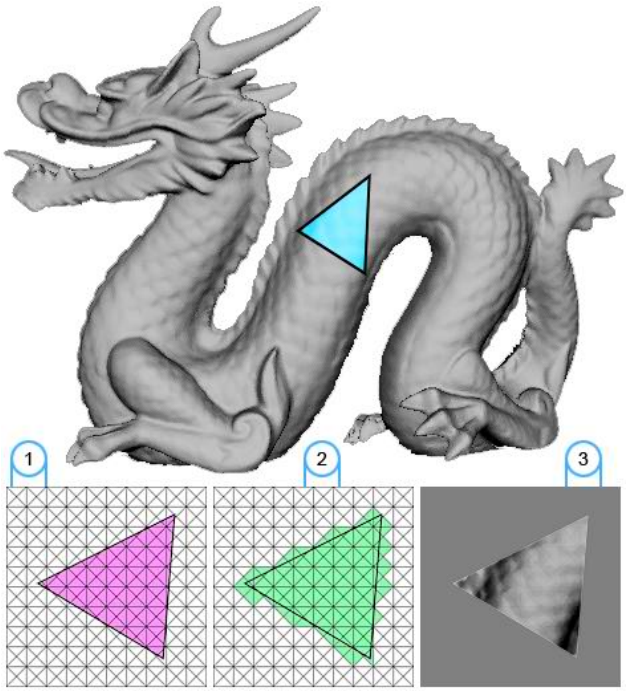

Figure 7. Projection clipping in action.

4. FEATURES OF THE PROPOSED ALGORITHM

Not only are modern GPUs up to 100+ times more powerful (FLOPS-wise) than CPUs, they're also designed specifically for rasterization operations. Furthermore, packing tens of parallel pixel pipelines and backed by drivers which allow to prepare next round of data while rendering is being finished, they can dramatically outperform CPU-based realizations. Thus, transferring most of the computations to the GPU and utilizing its inherent parallelism allowed to significantly increase the speed of displacement maps creation and bring the implemented application into the realtime category (if using previously built clipping tree). This makes it particularly handy for displacement parameters fine-tuning, as one can see the changes in almost no time, while adjusting a slider.

Flexible shading programmability makes it possible to implement most of the displacement and combined normal-displacement map formats as one-pass setups, while others can be implemented as two- or more complex multi-pass realizations, while still keeping the implementation interactive.

The use of the GPU offers pretty cheap antialiasing as well. While raytracing-based algorithms have to cast multiple rays from every point, using proposed technique, it is possible to utilize native hardware multisampling with nice nonuniform sample patterns. It is also possible to apply supersampling by rendering a displacement texture of bigger size and then scaling it down, either on the GPU or on the CPU, using preferred resampling algorithm. For bigger displacement maps, it is possible to render them in parts and then assemble obtained pieces into one. Any kind of postprocessing can be applied, including on-the-fly edge detection and subsequent edge line extension (in case of poor unwrapping adjacency, to avoid unpleasant effects of texture filtering applied by realtime visualization algorithms).

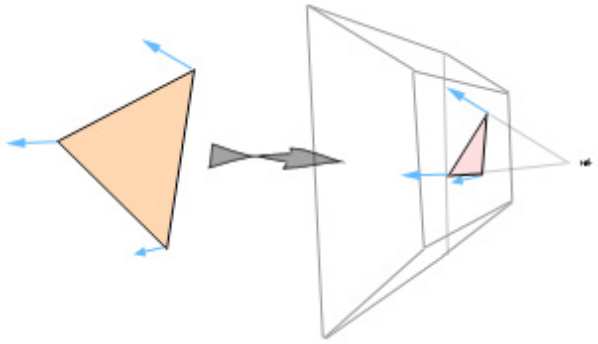

Figure 8. Using transformations to simulate normals' directions.

Raytracing algorithms cast distance rays towards the low-poly model triangle's normal direction. With proposed approach, it is possible to set up the frustum and the transformation matrices the way that the vectors connecting the viewpoint and the projected triangle's vertices will be pointing in the exactly same directions as the original triangle's normals. This makes it possible to precisely simulate distance-computing raytracing.

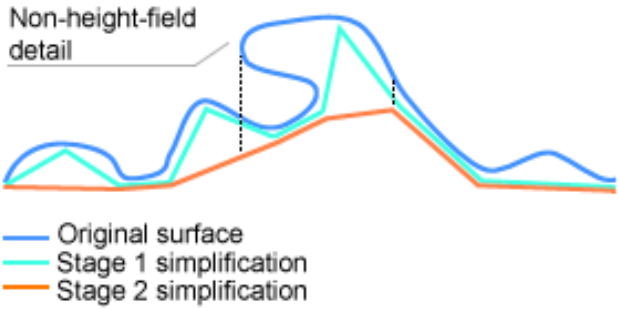

Figure 9. Multistep displacement.

We also propose a multistep displacement approach, which makes it possible to represent non-height-field surface features using two or more subsequent displacement adjustments. This method requires an equal number of intermediate models during displacement maps creation, although none of them are needed for surface restoration. Figure 9 shows an example of a two-step displacement, where the intermediate model is built in such manner that it tends to resemble as many original surface features as possible, while the final (stage 2) simplification is created after applying reasonable surface smoothing to flatten the relief features. Stage2→stage1 displacement map should normally be highly compressible, given the fact it will mostly consist of semi-uniform linear gradient patterns.

5. MODEL RESTORATION

The first and rather crucial thing to be considered before model restoration is the tessellation method. It is important not to underestimate the role of appropriate subdivision and its influence on the resulting mesh. When choosing between tessellation patterns, it is helpful to analyse the low-poly model and understand the nature of modifications made. Some of the patterns (a representative selection is shown on the Figure 10) only increase the number of vertices along the original triangle ribs. Others, on the contrary, tend to saturate inner areas of the triangle with newly added vertices.

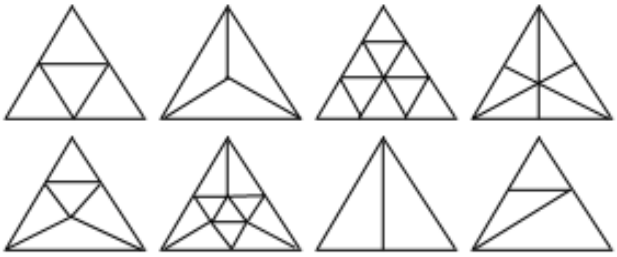

Figure 10. Various subdivision patterns.

It is also important to take into account the added triangles/vertices ratio figures and the subdivision algorithm's per-step memory consumption characteristic.

A number of adaptive and view-dependant subdivision techniques were introduced, often with a possible hardware support [Boo01], [Doggett00].

Some unwrapping strategies may result in vertex duplication, so that some triangles, originally sharing the same vertices and, eventually, ribs, will be indeed completely separated, with twin vertices having texture coordinates pointing to different areas of a displacement map. In case of antialiased or compressed displacement map, it may result in noticeable reconstructed model integrity degradation, be the map applied in a straightforward way. This problem could be solved using model repairing techniques explained in [Botsch06], although we propose an alias-based subdivision and displacing approach which allows to avoid geometry inconsistency and correctly restore model normals.

First, an adaptive-depth octree is built for the low-poly model.

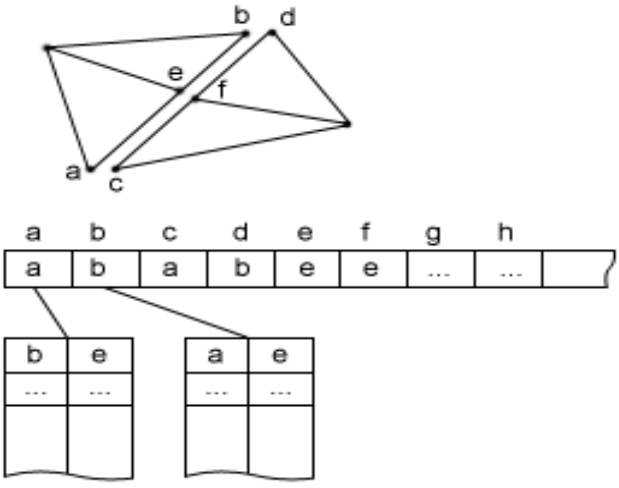

Figure 11. Vertex alias buffers.

Then a search among residents of each octree leaf node is performed, in order to reveal possible vertex duplication. Twin information is being stored in the main alias buffer, allowing all duplicate vertices to refer to the first instance found. All the other operations are performed using the alias-layer-based identification. During subdivision, each vertex registers all its neighbours (a neighbour is a vertex

residing on a defined rib along with the current vertex), so that if duplicate ribs exist, it is possible to detect subdivision-originated vertices duplication and maintain accurate and coherent alias information. Finally, alias information is used to restore smooth model normals, if not present, and synchronize duplicate vertices displacement amounts. After that, if normal map was not provided, another pass of smooth normals calculation is applied to restore correct model lighting attributes.

Various realtime restoration or relief imitation techniques exist. Recent approaches are listed in the section 1, although earlier proposals can be considered as well (e.g. [Wang03]). We've been recently introduced with the Direct3D 10 and the novel geometry shader in particular.

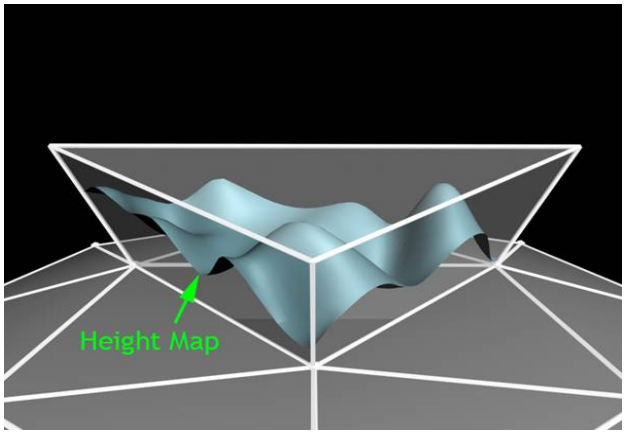

Figure 12. Applying displacement map in the geometry shader.

The new architecture element is capable of controlled surface subdivision and various mesh alterations, including real-time displacing. Figure 12 (courtesy of Microsoft) clearly demonstrates the described feature. Geometry shader processing is parallel and utilizes multiple pipelines, while serial order is preserved.

6. RESULTS

The proposed displacement and normal map creation algorithm was implemented using C++, Cg, Direct3D 9.0 and Windows API. The software model restoration technique was implemented in C++, using the proposed alias-based subdivision and restoration approach.

	512	1024	2048	4096
NM	63000 ms	157000 ms	Est >10 mins	Est > 40 mins
Our	42 ms	42 ms	43 ms	67 ms

Figure 13. Comparative timing, no multisampling. Model: Rabbit.

Figure 13 shows the details of comparative timings of the implementation of our algorithm and the NormalMapper 1.21 utility with multisampling turned off. The column headings are set to the square displacement map side sizes in pixels. Models used: high-poly model: 69451 faces, low-poly model: 4857 faces.

Figure 14 shows timings of the same test but with multisampling turned on. NormalMapper was set to cast 5 rays per pixel, while our rendering was performed with 4x antialiasing.

	512	1024	2048	4096
NM	122000 ms	392000 ms	Est 0.5 hour	Est > 1.5 hours
Our	43 ms	43 ms	45 ms	69 ms

Figure 14. Comparative timing, 5 rays / 4x AA. Model: Rabbit.

Face quantity numbers for the test models are given on the figure 15.

Model	High-res, faces	Low-res, faces
Bunny	69451	4857
Sphere	262144	16384
Sword	282624	438

Figure 15. Test models.

Clipping tree build times are listed below.

Model	Bunny	Sphere	Sword
Time	1607 ms	14896 ms	1401 ms

Figure 16. Clipping tree build times.

Next are detailed displacement map creation timings.

	Bunny	Sphere	Sword
512	41.8	145.1	25.0
1024	42.4	145.7	25.0
2048	43.3	146.3	25.1
4096	67.1	149.7	25.2

Figure 17. Detailed timings, in milliseconds.

As you can see from the timing table, map creation time does not grow proportionally to the image size, opposed to that of raytracing algorithms.

Below are some model restoration results and both displacement and combined normal-displacement map samples created for the test models.

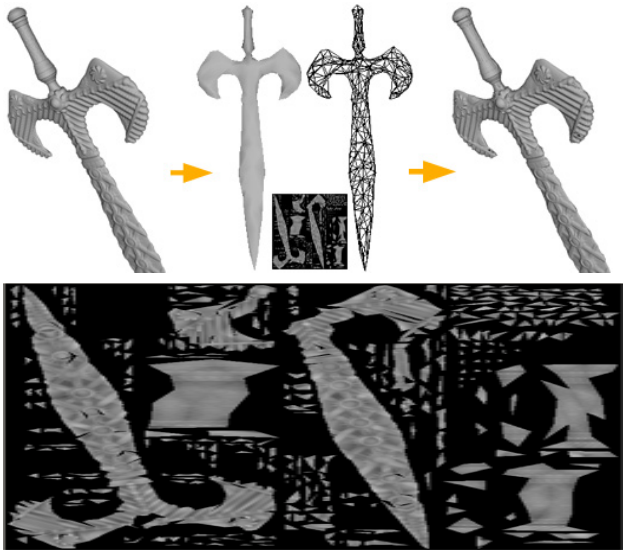

Figure 18. Restoration results, displacement map. Model: Sword.

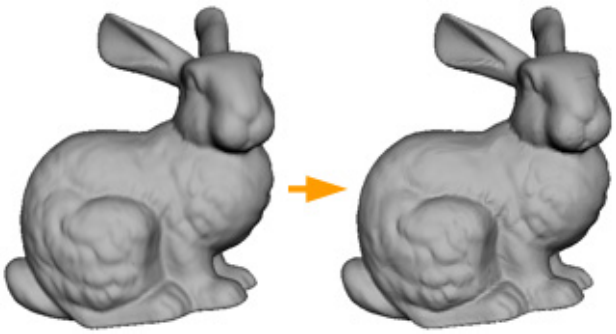

Figure 19. Restoration results. Model: Bunny. More on figure 3.

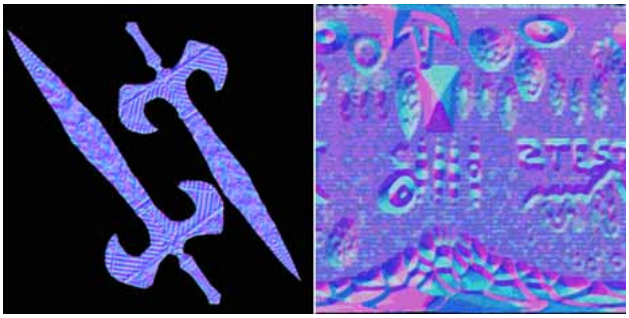

Figure 20. Combined normal-displacement maps for the Sword and Sphere models. Displacement value is stored in the alpha channel.

We've also done some data size testing. As you can see on the figure 21, after model simplification and displacement map creation, the compressed resulting set takes up to 11 times less space than the compressed original model. Compression used: RAR with the compression level option set to "Best".

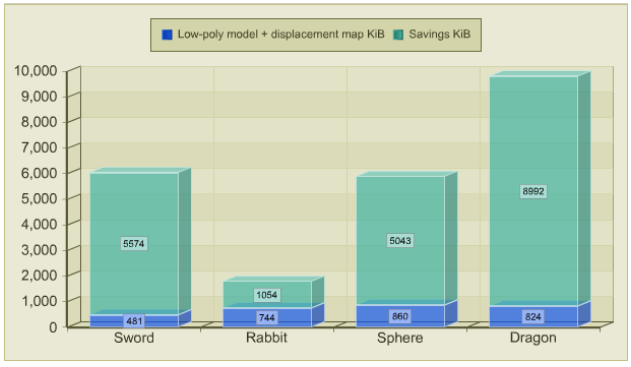

Figure 21. Space savings after model simplification, displacement map creation and resulting set compression.

All the measurements were taken on the following configuration: P4 2.4GHz, 1024+256 MB RAM, NVIDIA 6600GT 128MB.

7. CONCLUSION

Displacement maps are used in the leading state-of-the-art realtime rendering techniques, they can also be utilized for modeling and polygonal models compression. We have proposed a fast, flexible and efficient differential displacement and combined normal-displacement maps creation algorithm, which exploits vast computational power and inherent parallelism of modern graphics hardware. We have also proposed a fast and precise model restoration technique and a multistep-displacement-based non-height-field surface feature representation approach, which can be implemented in hardware to achieve realtime model reconstruction, and then combined with any of the available (self-)shadowing techniques.

8. REFERENCES

- [Policarpo05] Fabio Policarpo, Manuel M. Oliveira, João Comba. *Real-Time Relief Mapping on Arbitrary Polygonal Surfaces*. ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games, Washington, DC, April 3—6, 2005, pp. 155—162.
- [McGuire05] Morgan McGuire, Max McGuire. *Steep Parallax Mapping*. I3D 2005 Poster.
- [Donnelly05] William Donnelly. *Per-Pixel Displacement Mapping with Distance Functions*. 2005. In GPU Gems 2, M. Pharr, Ed., Addison-Wesley, pp. 123—136.
- [Tatarchuk06] Natalya Tatarchuk. *Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows*. 2006. In Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (SI3D '06), pp. 63—69.
- [DirectX Dev] Microsoft DirectX Developer Center. http://msdn.microsoft.com/directx/
- [Rational] *Rational Reducer*. http://www.sim.no
- [Discreet] *Discreet 3D Studio MAX*. http://www.discreet.com
- [Pixologic] *Pixologic ZBrush*. http://www.pixologic.com
- [ATI] *ATI NormalMapper*. http://ati.amd.com/developer/tools.html
- [Boo01] M. Bóo, M. Amor, M. Doggett, J. Hirche, W. Strasser. *Hardware Support for Adaptive Subdivision Surface Rendering*. 2001. SIGGRAPH, pp. 33—40.
- [Doggett00] Michael Doggett, Johannes Hirche. *Adaptive View Dependent Tessellation of Displacement Maps*. 2000. SIGGRAPH, pp. 59—66.
- [Botsch06] Mario Botsch, Mark Pauly, Christian Rossl, Stephan Bischoff, Leif Kobbelt. *Geometric Modeling Based on Triangle Meshes*. 2006. SIGGRAPH, ISBN:1-59593-364-6.
- [Wang03] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, Heung-Yeung Shum. *View-Dependent Displacement Mapping*. 2003. SIGGRAPH, pp. 334—339.

About the authors

Ilya Tisevich is a student at Moscow State University, Department of Computational Mathematics and Cybernetics. His contact email is ilya.t@mail.ru.

Alexey Ignatenko is a Ph.D. researcher at Moscow State University, Department of Computational Mathematics and Cybernetics. His contact e-mail is ignatenko@graphics.cs.msu.ru.