Fast Normal Map Generation for Simplified Meshes

Yigang Wang
Computer School, Hangzhou Institute of Electronics Engineering, China
Bernd Fröhlich
Bauhaus-Universität Weimar, Germany
Martin Göbel
Fraunhofer-Institut fuer Medienkommunikation, Germany

Abstract

Approximating detailed models with coarse, normal mapped meshes is a very efficient method for real-time rendering of complex objects with fine surface detail. In this paper, we present a new and fast normal map construction algorithm. We scan-convert each triangle of the simplified model, which results in a regularly spaced point set on the surface of each triangle. The original model and all these point samples of the simplified model are rendered from uniformly distributed camera positions. The actual normal map is then created by determining the corresponding normal vector for each point in the point set, for which the distance between two corresponding points in image pairs over the set of camera positions is minimal. Our approach works for general triangle meshes and exploits fully common graphics rendering hardware. Normal map construction times are generally in the range of only a few seconds even for large models. We render our normal-mapped meshes in real-time with a slightly modified version of the standard bump-mapping algorithm. In order to evaluate the approximation error, we investigate the distance and normal errors for normal-mapped meshes. Our investigation of the approximation errors shows that using more than twelve view points does not result in a further improvement the normal maps for our test cases.

1. Introduction

Simplification of triangle meshes has been an active area of research in computer graphics. Various groups [4]-[8] have shown that texture- and normal-map-based representations can efficiently preserve the geometric and chromatic detail of the original model. Recent graphics cards support these techniques in hardware and allow single pass real-time rendering of complex texture and normal mapped geometry.

Only few methods have been proposed for generating normal maps [4]-[8]. These methods usually scan-convert each triangle of the simplified model to obtain a regularly spaced point set. The corresponding normals are computed from the high resolution model and they are packed into a rectangular array - the normal map. The main issue is the construction of a mapping from the simplified model to the original model to calculate the corresponding normal for each point of the normal map. Two approaches have been suggested for solving this problem. One approach directly computes the corresponding normal vector for each point of the normal map[5][6]. The other approach assigns the corresponding normal vectors by parameterizations [7][8]. However, these methods are rather time-consuming for large models.

In this paper, we present a new method, which makes great use of common graphics rendering hardware to accelerate the normal map construction. In our method, the scan-converted point set of the simplified mesh's triangles and the corresponding original model are rendered from uniformly distributed camera positions. The actual normal map is then created by updating corresponding normals for each point in the point set, for which the distance between two corresponding points in image pairs for the set of camera positions is minimized. Similar to [9], the distance is estimated from Z-buffer entries. A clear limitation of our method is the requirement that all the faces of a mesh need to be seen from at least one viewpoint. There are several solutions to this problems, including resorting back to other more time-consuming methods such as [5][6] for these critical areas. Our method works as a post-process of a simplification. It can be combined with any existing simplification method even if the topology changed during the simplification process.

We have tested our algorithm for various large models and show that high quality normal maps for complex objects can be generated on standard PC hardware within a few seconds. Our implementation of normal-map-based rendering makes use of Nvidia's GeForce3 and GeForce4 graphics cards, which allow the representation of normal maps with 16 bit per component. Our error analysis indicates that twelve views are in most cases enough for generating high quality normal maps.

2. Creating normal maps

Normal maps for height fields can be generated from a single view point (Figure 5). General meshes require the information from multiple view points from around the model. The whole normal map construction algorithm can be described in the following way:

```
Normalmap
constructNormalMap( Model simplified_model, Model original_model)
{
    normalmap = simplified_model.scanConvert();
                //scan convert each face and
                //pack points into rectangular array.
    simplified_model.formNormalmapCoordinates();
                //normal map coordinates
                //are computed according to the packing method.
    normalmap.intialize( simplified_model);
                //set the initial values as interpolated values from
                //vertex normal vectors of the simplified model, and
                //set the initial distance value as the maximal float value.
    original_model.convertNormalToColor();
    for( int viewpoint =0; viewpoint < max_view_num; viewpoint++)
    {
        setTransformation( viewpoint);
        original_model.render();  //with colors mapped from normals
        readZbuffer(zBufferS);
        readColorBuffer(cBufferS);
        for( int i=0; i<normal_map.sampleNumber(); i++)
        {
            vector t =  projectToScreen(normalmap(i).position);
            float z = zBufferS(t.x, t.y);
            float dist = fabs (z-t.z);
            if( dist < d_threshold  &&  dist < normalmap(i).dist )
            {
                normalmap(i).color = cBufferS(t.x, t.y);
                normalmap(i).dist = dist;
            }
        }
    }
    normalmap.reconvertColorToNormal();
    return normalmap;
}
```

Sampling the simplified model. We scan-convert each face of the simplified model in software. The triangular sample point sets are packed into a rectangular sample array. We use a simple sampling and packing method. Each face of the simplified mesh is sampled into a right-angled triangle patch of equal resolution, two triangle patches that share one edge in the model form a square, and each of the remaining triangle patches occupies a square. These squares are packed together and form the rectangular normal map array. The actual number of these squares and the user-specified size of the normal map determines the number of samples per square and therefore per simplified triangle. Other more complex sampling and packing approaches can be found in [5] [8]. The interpolation within the normal map may result in discontinuous edges, since adjacent sample patches in the packed normal map may be non-adjacent on the simplified mesh. This problem can be avoided with a similar approach chosen in [5] for an anti-aliasing method. We simply extend each face and save a slightly wider sampling patch (one or two pixels wider in the discrete normal map space) to solve the problem.

Sampling viewpoints. For each sampling point on the simplified model, its corresponding point and normal vector on the original model need to be computed. Similar to [5], we define the corresponding point on the original mesh as the one whose distance from the sample point is the shortest. The computation of exact point correspondences based on a distance metric is complicated and time-consuming [5], so we resort to a viewpoint sampling method. For simplicity, we uniformly sample viewpoints around the object. A set of optimal view point positions can be determined by using Stuerzlinger's [14] hierarchical visibility algorithm.
For each specified camera position, we implement the following four steps:
1. Render the original model onto the screen with vertex colors corresponding to its vertex (or face) normal vectors;
2. Transform each sampling point from the simplified model into current screen coordinates with the same viewing parameters;
3. For each sampling point, replace the corresponding point on the original model (therefore color values) if the distance is smaller than the currently stored distance. For measuring the distance, we just use the difference between the z-values of the sampling point and a point on the original model which maps to the same pixel on the screen.
4. Finally, we reconvert the color vectors to normal vectors, which gives us the normal map.

For the above procedure, we have chosen to use a fixed set of viewpoints, and require they are uniformly distributed around the model. For our models, we typically used 12 positions, as shown in Figure 4a, which has shown to be adequate in most cases - even for the less uniformly shaped objects. In general, the final normal vectors of the normal maps have smooth transitions between two different viewpoints, since the images from neighboring viewpoints typically overlap.

Improving normal vector precision. The above method usually creates normal maps with 8-bit precision for each component because color buffers typically have three 8-bit color components. There are two kinds of errors, one comes from quantizing the vertex normal to an 8-bit color component. The other is a result of the 8-bit color interpolation. For avoiding these quantization errors as far as possible, we could use an item buffer. Instead of directly rendering the normals of the high resolution model, we render triangle IDs as colors in a first pass. The corresponding point on the original mesh therefore the corresponding normal can be found by the triangle ID and the pixel coordinates and depth of the triangle for a given sample point. The quantization of the normal happens then just before it is entered into the normal map, which could be 8 or 16 bit in our case.

Filling invisible parts in normal map. A fixed set of viewpoints is adequate for a wide variety of models. However, for models with very complex visibility there might be some invisible samples. Often such parts are not important since they are difficult to see if the object is just observed from the outside. However, these undefined samples may result in apparent visible artifacts. In order to avoid undefined areas in the normal map, we initialize the normal map by interpolating the vertex normals of the simplified model. We also provide a distance threshold, and only update the elements in the normal map if the corresponding distance is less than the given threshold. As a consequence, the samples corresponding to invisible parts will keep their initial values, which are the interpolated normal vectors of the simplified model. Thus we avoid visible artifacts in these regions and use at least Phong's normal interpolation method.

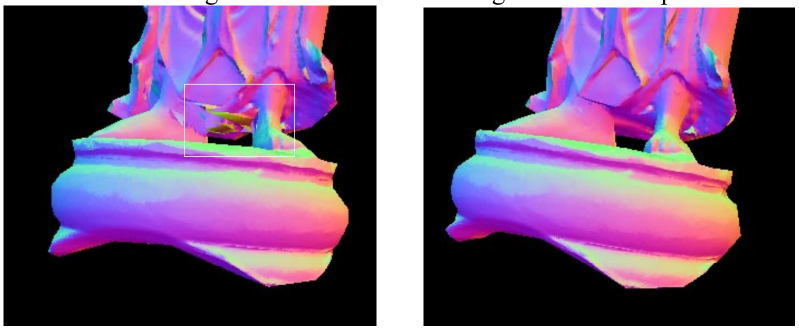


Figure 1: *Comparison of the methods without and with normal initialization. Both image are synthesized by rendering a simplified model (buddha) with normal map textures. The left image corresponds to the method without normal initialization, while the right side corresponds to the method with normal initialization. The white rectangle*

contains some invisible parts for the fixed set of viewpoints, which are below the robe of the statue.

Another basic alternative for avoiding invisible parts in the normal map is to interactively update the normal map by rotating the model in track ball mode. In this way, new views assigned by the user are added to the fixed set of views to improve the normal map.

3. Real-time rendering of normal mapped meshes

Single pass real-time rendering of normal-mapped meshes is possible with today's graphics hardware. We use an Nvidia GeForce3 card, which supports vertex programs, texture shaders, and register combiners. Our implementation is pretty similar to a standard real-time bump mapping implementation using Nvidia's extensions[10], except that we compute the lighting in the local face space instead of the texture space[15].

Our experiments showed that an internal precision of 8 bits introduced visual artifacts in high light areas, such as blocky appearance and mach banding. We had to resort to 16 bit precision, which is supported by a texture map type, called GL_SIGNED_HILO_NV. Each element in a GL_SIGNED_HILO_NV texture map consists of two 16-bit signed components HI and LO. HI and LO are mapped to a [-1, 1] range. Such a 2D vector corresponds to a 3D vector (HI, LO, sqrt(1-HI^2 –LO^2)). For this type of texture map, the internal calculations such as dot products are performed with 16 bits precision. Figure 2 compares the results for using these different texture types. We found that the external precision of a normal map is not very important. Normal maps with 8-bit or 16-bit precision lead to very similar visual results if the internal type is set to 16-bit precision computation. In order to use 16-bit internal computations, normal maps have to use GL_SIGNED_HILO_NV texture maps. The normal maps need to be transformed into a space where the z-value of each normal should be not less than zero. Our normal maps are packed with discontinuous square patches, which makes it in most cases impossible to construct a single consistent texture space for all triangles represented in the normal map. Instead, we transform all normal vectors within a triangle patch of the normal map into a local face space, whose z-axis is the same direction as normal vector of the triangle. The x-axis is one edge of the triangle and the y-axis is simply the cross product of x-axis and z-axis. As the result, the Z-value of the normal vectors in the triangle patch is usually greater than zero, so we can use GL_SIGNED_HILO_NV normal maps and 16 bit computation for our lighting calculations.

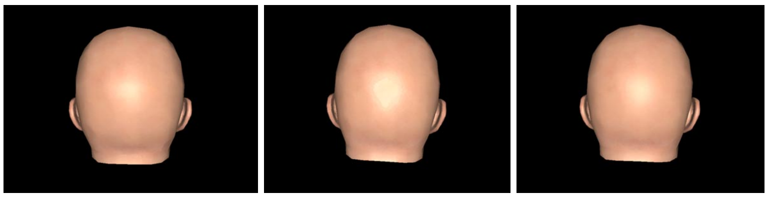

(a) (b) (c)

Figure 2: *Results for different internal and external normal map precisions. (a) The external normal map type is GL_SIGNED_BYTE, the internal format is GL_SIGNED_HILO_NV. (b) The external normal map type is GL_RGB8, the internal format is the same. (c) The external normal map type is GL_SHORT, the internal format is GL_SIGNED_HILO_NV.*

4. Example and timing results

We adopted Garland's simplification method [1] to decimate our models. Our normal map construction and rendering algorithms are implemented under Linux on a 1.3GHz AMD PC with 768M of RAM and 64MB Geforce3 Graphics.

Model	number of faces (simplified model)	Construction Time (seconds)	Number of samples
Bunny (original model has 69,451 faces)	500	7.39	900,000
	1001	7.26	882,882
	1860	7.75	952,320
	2306	7.51	903,952
	3109	7.51	895,392
	7200	7.7	921,600
Buddha (original model has 1,085,634 faces)	2446	12.23	890,344
	3161	12.07	910,368
	4072	12.18	895,840
	6000	12.03	864,000

Table 1: *Normal map construction timings for two models. The normal map size is set to 1024x1024. Then normal map construction time depends on the number of points sampled on the simplified model and not very much on the size of the original model.*

Table 1 shows our construction method as outlined in section 2. It is interesting to notice that the complexity of the original model does not influence the processing time heavily. We found that the rendering time heavily depends on the number of sample points, which need to be transformed by the CPU or through a feedback mechanism of the graphics hardware, which is typically not highly optimized.

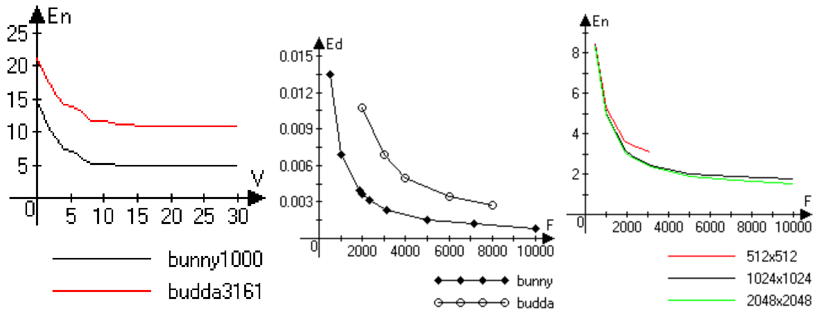
An example of a normal-map rendering for a model with simplified topology is shown in Figure 6. Gaps and creases between finger bones disappear in the simplified model. However, these gaps can be seen again from the normal mapped model. In this case, the normals on the original model are well preserved on the simplified model with different topology. This example demonstrates that our method works well even for complex cases.

Figure 7 shows the rendering of a normal-mapped model on a Geforce3 system. There is an overhead involved compared to conventional Gouraud shaded rendering, but this overhead is usually quite small. For larger simplified models e.g. with 10000 or 20000 triangles, we found that one can become quite quickly geometry limited. This is due to the fact that the length of the vertex programs has basically a linear influence on the processing time per vertex. Our vertex program is XXX instructions long. For high resolution renderings – such as 1600x1200 – we are usually fill limited. The per-pixel shading computations have here a strong influence.

5. Error analysis

In order to evaluate the similarity between the original meshes and the normal mapped simplified meshes, we render images from a set of view points around the model. For our evaluation, we used 30 view points distributed uniformly around the model. For each view, we acquire two images with depth values. The first image is obtained by rendering the original model with vertex colors corresponding to the vertex normals. The second image is obtained by rendering the simplified model with the normal map as a color texture. The depth values allow us to compute distance errors. Normal errors can be obtained from corresponding color values. There are quantization errors involved, since the normal vectors are quantized to 8 bits.

To evaluate the errors versus the number of views, we performed the following experiment. We start with the normal map which is initialized with normal vectors obtained by interpolating the vertex normals of the simplified mesh. The normal map is updated by adding new views one by one. After each update, we compute the normal and distance errors for the current normal-mapped mesh. The result of the experiment is shown in figure 3(a). It can be seen from the figure that the normal errors cannot be reduced much after 12 views for our models. Another observation is that the normal errors cannot drop far below 5 or 10 for bunny or budda model because of geometrical error.



(a) (b) (c)

Figure 3: *(a) Normal errors versus the number of views (quantized to 8 bit, average vector difference). The normal map resolution is 1024×1024. (b) Distance error versus the face number of simplified models.(normalized screen coordinates) (c) Normal errors versus the face number of the simplified mesh and the size of the normal map (bunny model).*

Note that the normal and position errors of normal mapped meshes reduce slowly with increasing face numbers. Besides the number of faces, the size of the normal map also influences the visual quality of the normal mapped mesh. With larger normal map sizes, more normal samples are obtained, which results in less normal errors. Figure 3c shows such case. It is very interesting to notice that it does not make much sense to use very large normal maps for simplified models with a small number of faces, since the geometric error dominates. This basic evaluation gives an idea, how the different parameters affect the quality of the model. However, much more evaluation needs to be performed for a larger variety of models to get more generally applicable conclusions.

In the above, we evaluate the error by averaging all the errors for the entire model. It may be more interesting to see the local errors for each part of the model. To visualize these errors, we render two images for a fixed view as before. One is obtained by rendering the original model with vertex colors corresponding to the vertex normals. The other is obtained by rendering the simplified model with the normal map as a color texture. The difference between the two images is the error for this view. Figure 4b shows the error with the appropriate colors. It is clear the biggest differences appear in detailed parts.

6. Discussion

Sander et al[6] mention that the parameterization based on geometrically closest points used in Cignoni et al[5] leads to discontinuities. One may expect that our method will lead to even more discontinuities. However, we found that the discontinuities are not very obvious with our method, because we use a small number of views. It is clear that there are no discontinuities in areas, which are visible from only one view. Adding further view results in smooth normal transitions in regions seen from multiple views. We extended our method to compute corresponding points by minimizing the angles between interpolated normal vectors and the viewing direction. We found that the quality of the normal map improved only slightly, but the constructions process became much slower. These tradeoffs need further study.

Normal vector aliasing is an issue that needs further investigation. Currently, we use only very basic methods to sample the models. Mip-mapping is a well-known technique for anti-aliasing textures. For meshes that can be represented as z=f(x,y), different mip map levels can be easily generated to avoid normal vector aliasing. However, it is more difficult to extend the idea to the general case. The content of a packed normal maps is not continuous along the triangle boundaries, which means that mip-map levels need to be generated by hand by appropriately down sampling the normal information. In addition, our method is limited by the screen resolution at the moment. If the resolution of the original model becomes much larger than the screen resolution, our approach will need to be extended to handle chunks of the model such that the screen resolution is larger than the number of triangles in each chunk. Otherwise, our method down samples the original model by point sampling the normals, which could result in undersampling in this case and therefore aliasing artifacts.

Due to hardware limitations and our single pass rendering, our normal map rendering implementation supports the illumination of textured objects by only a single light source. Support for multiple light sources requires a multi-pass rendering approach on current graphics hardware.

Our current implementation uses a set of uniformly distributed view points around the model. Stuerzlinger's [14] hierarchical visibility algorithm should be useful to determine a set optimal view positions, which would fully automate our approach and result in even higher quality normal maps.

ACKNOWLEDGEMENTS
We thank the VRGeo consortium for support and discussions in the context of this work. We also thank the anonymous reviewers who provided excellent feedback on the first version of this paper and helped to improve the presentation a lot. The research is also supported by National Science

Foundation of China (No. 60021201). The bunny, dragon and happybudda models are provided by Stanford Computer Graphics Laboratory.

REFERENCES

[1] M.Garland and P.S. Heckbert, Surface simplification algorithm using quadric error metrics. Computer Graphics (SIGGRAPH'96 Proceedings), pages 209-216, 1996.

[2] Hugues Hoppe. New Quadric Metric for Simplifying Meshes with Appearance Attributes. IEEE Visulization'99 proceedings.

[3] P. Lindstrom and G. Turk, Image-Driven Simplification, ACM Transactions on Graphics, Vol.19, No.3, July 2000, pp204-241.

[4] Marc Soucy, Guy Godin, and Marc Rioux. A texture-mapping approach for the compression of colored 3D triangulations. In Visual Computer, 12:503-514, 1996.

[5] P. Cignoni, C. Montani, R. Scopigno, A general method for preserving attribute values on simplified meshes. In Visualization'98 proceedings, IEEE, pp.59-66.

[6] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, J. Snyder, Silhouette Clipping, SIGGRAPH'2000, pp327-334.

[7] J. Cohen, M. Olano and D. Manocha, Appearance-preserving simplification. SIGGRAPH'98, pp 115-122.

[8] P. V. Sander, J. Snyder, S. J. Gortler, H. Hoppe, Texture mapping progressive meshes. SIGGRAPH'2001, pp409-416.

[9] U. Labsik, R. Sturm and G. Greiner, Depth Buffer Based Registration of Free-form Surfaces, VMV 2000 proceedings, 2000.

[10] Chris Wynn, Implementing Bump-Mapping using Register Combiners.Documentation in NVSDK from Nvidia Corporation.

[11] E. Puppo and R. Scopigno. Simplification, LOD, and Multiresolution – Principles and applications. In EUROGRAPHICS'97 Tutorial Notes. Eurographics Association, Aire-la-Ville (CH), 1997.

[12] Heckbert, P., and Garland, M. Survey of polygonal surface simplification algorithms. In Multiresolution surface modeling (SIGGRAPH'97 Course notes #25). ACM SIGGRAPH 1997.

[13] Garland, M. and Heckbert, P. S. Simplifying surface with color and texture using quadric error metrics. IEEE Visualization'98, pp263-269.

[14] Stuerzlinger, W. Imagine all Visible Surfaces. In Proceedings of Graphics Interface' 99, pp. 115-122, 1999.

[15] D.Sim Dietrich Jr. Texture Space on Real Models, Documentation in NVSDK from Nvidia Corporation.

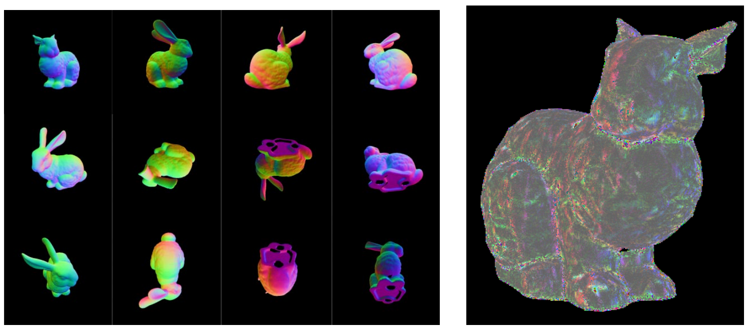

(a) **(b)**

Figure 4: ***(a)*** *The 12 different views of an object used during normal map creation for the bunny model. . The object is rendered with colors representing the vertex normal vectors* ***(b)*** *The normal errors for the bunny model corresponding to a given view.*

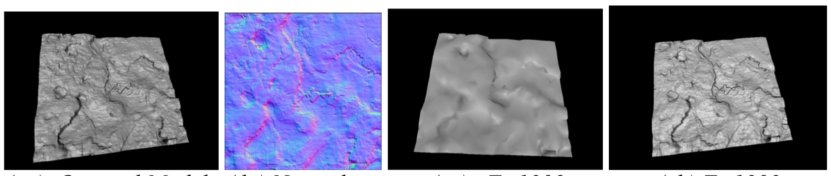

(a) Original Model T= 496,724. t= 90.2ms. *(b) Normal map* *(c) T=1000 t=1.60 ms* *(d) T=1000. t = 2.5ms*

Figure 5: *Normal maps for a height field mesh can be generated from a single view point. T is the number of triangles of the model, t is the drawing time per frame. (b) Normal maps are shown as colors. (c)Result of rendering the simplified model using Gouraud shading on a PC with a Geforce 3 graphics card. (d) Result of rendering the simplified model with a normal map on the same PC as (c).*

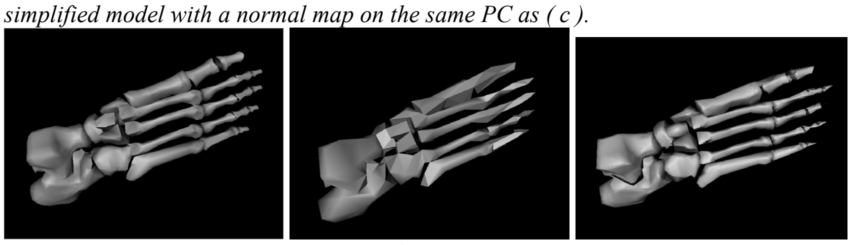

(a) (b) (c)

Figure 6: *Normal maps for a model with simplified topology.* ***(a)*** *Original Model* ***(b)*** *Model with simplified topology* ***(c)*** *Simplified model with normal map.*

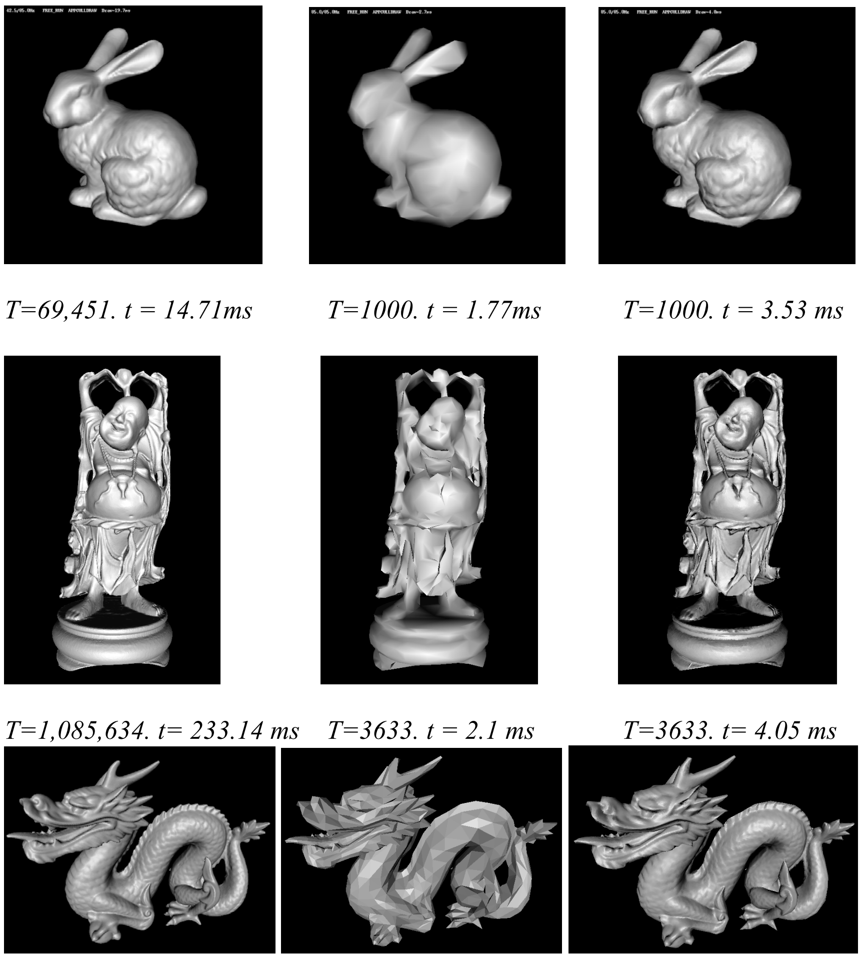


Figure 7: *Comparison of rendering the original model, a simplified model without normal map, and a simplified model with normal map.*