Hardware generation of normal maps

Jesús Gumbau, Carlos González , Miguel Chover

Universidad Jaume I, Dept. de Lenguajes y Sistemas Informáticos, Castellón de la Plana 12071 Spain

Abstract

In this paper a method for hardware generation of normal maps is presented. These normal maps may be applied to low resolution objects so they take on the aspect of more detailed ones. The proposed method for normal maps generation is a brand-new method, since even nowadays this process has been performed through software techniques. Hardware generation greatly reduces time in comparison with present-day solutions. Moreover, it allows for a dynamic modification of the map. However, there are some restrictions in relation to how texture coordinates must be distributed. *Vertex* and *pixel shaders* have been used for the generation and usage of the map. This method works perfectly for terrains and walls.

Key words:
normal map, hardware, shaders, simplification

1 Introduction

The surface textures in the first 3D videogames were only based on the color information. Afterwards, the idea of combining an illumination map with the color map was introduced to create more realistic shading effects. However, the problem then was that the light sources could move and the shading of a texture depended on these.

The next technique was "bump mapping", which uses the height map. Height maps are gray-scaled maps that code the information about the objects' height.

Email addresses: jesus.gumbau@anubis.uji.es (Jesús Gumbau), cgonzale@sg.uji.es (Carlos González), chover@uji.es (Miguel Chover).

Finally, the normal map method came into being. Unlike height maps, which only make use of a map with the information of the height, normal maps contain 3 vectors of information per pixel(the coordinates of the normal in the XYZ plane). This information is coded in a RGB color map, corresponding to each X, Y, Z coordinates with the R, G, B colors respectively. In the normal map technique two versions of a same polygonal model participate: the high resolution model, which is used to generate the normal map, and the low resolution model, where it is applied.

The presented method proposes a fast hardware generation of normal maps that uses *vertex* and *pixel shaders*. This idea involves a real-time normal map generation of the object. It also involves a greater processing velocity than existing methods do, because they make use of the CPU to generate the map. Present-day methods are generated by software by programming the necessary instructions for the normal map generation so that the CPU processes them.

It has to be considered that the proposed method can present a more realistic aspect of the object. This is due to the fact that software methods perform operations to calculate the normal and the proposed method apply the real normal value. The difference in quality between the proposed method and existing methods can be assessed as being almost contemptible.

Two restrictions have to be accomplished:

- Two or more triangles should not share the same texel, otherwise the colors generated for the normal map would superpose. However, this requirement is studied in the literature (4)(9), making emphasis on the method presented in (7).
- Texture coordinates have to be distributed so that the texture should be correctly applied to both models.

This method works perfectly for terrains and walls, because these objects usually meet the requirements.

This paper has the following structure. Section 2 refers to previous work concerning this topic. Some basic concepts are introduced in section 3. In section 4, the method is presented in detail and both the generation and the application of the normal map are described in subsections 4.1. and 4.2. respectively. In section 5, the obtained results are provided and commented upon.

2 State of the art

Some normal maps generation applications have already been presented, but all of them generate the map by software with the corresponding CPU usage. We emphasize the method proposed in (10), which uses the same method as that presented in this paper, but by software. Normal maps can be created by 3D edition programs, such as, for example, *3D Studio MAX 7* or *Maya 6.0*. Applications exclusively dedicated to the creation of normal maps also exist, such as *ATI's NormalMapper* (1) or *nVidia Melody* (5).

nVidia's Melody is an independent program, which presents a simple interface with different options to load and generate the normal map.

Ati's Normal Mapper offers libraries and is managed by a command line.

The nVidia's and ATI's software make use of the object at two levels of detail, so the normals are cast from the low resolution model to the high resolution model, and where they intersect the normal from the high resolution model is taken in order to be applied to the low resolution model.

3 Basic concepts

A Normal map contain information about the object surface, so it may be altered in order to modify the appearance of the object without changing its geometry. In this way, the normal map can be used in the low resolution model, so that it takes on the aspect of a high resolution one, thus saving the creation of more triangles, as well as the correspondening computational and temporal cost. Normal maps can be built in relation to 3 spaces: world space, object space and tangent space.

- *World space*: each pixel stores its orientation in the world space, and no additional computation is required to obtain the normal value. It works correctly with static meshes.
- *Object space*: each pixel stores its orientation in the object space, so it would be necessary to apply the object transformation matrix to obtain the normal. It works correctly with moving objects.
- *Tangent space*: each pixel stores its orientation in relation to the face which the pixel pertain. It is ideal for deformable objects.

Figure 1 shows the normal maps of a sphere in the world and tangent spaces.

Vertex and *pixel shaders* are used for the hardware generation of normal maps. *Vertex* and *pixel shaders* are small fragments of programmable code, which

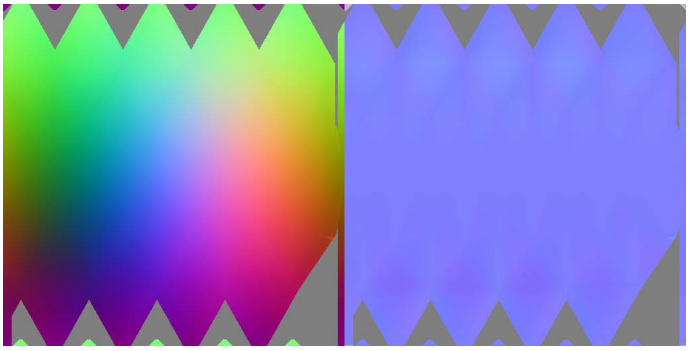

Figure 1. Normal maps of a sphere in the world and tangent spaces respectively

state the way that the GPU uses the vertices and pixels of the image. So, OpenGL sends the geometry of the object to the *graphics pipeline*, which works with it. By using *vertex* and *pixel shaders* however, we can specify how the GPU has to work with this geometry. Some papers related to this topic exist in the literature, examples of which are (2)(3)(6)(8)(11). The use of *vertex* and *pixel shaders* is graphically shown in Figure 2.

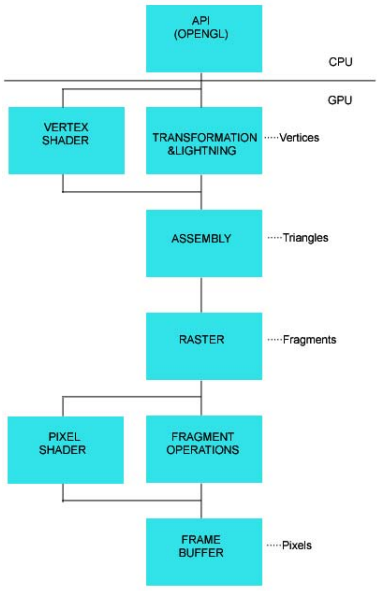

Figure 2. *Graphics pipeline*

4 Method

Unlike the present-day normal map generation methods, the presented method generates the maps by *hardware*, by making use of *vertex* and *pixel shaders*. The method is based on the use of a 3D object with two different levels of

detail. The idea is to generate the normal map of the high resolution model in order to assign it to the low resolution model so that it takes on the aspect of a more detailed object without increasing its geometry. The normal map will be generated in the world space. The steps taken that generate the normal map by *hardware* are explained in subsection 4.1. Although some methods exist that work by hardware in order to apply the normal map to the low resolution model, this step is commented upon in subsection 4.2.

So, two *vertex shaders* and two *pixel shaders* will be used. The first will generate the normal map, and the last will apply it to the low resolution model.

4.1 Generation of normal maps

The enabled *shaders* here perform the following:

- The *vertex shader* flattens the image, so it transforms the coordinates of each vertex depending on the texture coordinates. In other words, it is taken as coordinates x=u, y=v, z=0, where (x,y,z) are the coordinates of the object, and (u,v) are the coordinates of the texture. Moreover, it passes the normal through the *pipeline*. The pseudo-code with these instructions is shown in Algorithm 1.
- The *pixel shader* generates the normal map so that it passes the normal coordinates to the RGB components of the resulting color in this pixel. For this purpose, it is necessary to convert the normal values into the accepted range by the RGB plane, that is, [0,1]. Algorithm 2 shows the pseudo-code of this *pixel shader*.

```
result.pos.x = in.texcoord.u;
result.pos.y = in.texcoord.v;
result.pos.z = 0;
result.normal.x = in.normal.x;
result.normal.y = in.normal.y;
result.normal.z = in.normal.z;
```

Algorithm 1. *Vertex shader* for normal map generation

```
normal = in.normal.range(0,1);
result.color.r = normal.x;
result.color.g = normal.y;
result.color.b = normal.z;
```

Algorithm 2. *Pixel shader* for normal map generation

The result is directly stored in a texture so that it may be applied as a normal map.

4.2 Application of normal maps

When the normal map has been generated, the created texture is applied to the low resolution model. This map will represent the virtual direction of the surface at each point. The enabled shaders here work as follows:

- The *vertex shader* transforms the position of each vertex with the actual transformation matrix ("Model-view-projection", MVP) and passes the texture coordinates (normal map) to the *pipeline*, and the position of each vertex is transformed by the model transformation matrix (M) to calculate the lightning in the *pixel shader*. Algorithm 3 shows the pseudo-code of this vertex shader.
- The *pixel shader* obtains the color of each texture point, given by the normal map. This normal map will be applied to the object. Moreover, the *pixel shader* calculates the light direction, substracting to the light position the *fragments* position (candidate points as being pixels) of the object. Finally, it converts the normals into the range [-1,1] and the scalar product between the light direction and the normal is calculated to illuminate the object. Algorithm 4 shows the pseudo-code of this *pixel shader*.

```
result.pos = MVP*in.pos;
result.mpos = M*in.pos;
result.texcoord = in.texcoord;
```

Algorithm 3. *Vertex shader* for normal map application

```
u = in.texcoord.u;
v = in.texcoord.v;
colortex = normalmap[u,v];
light.dir= in.mpos-in.light.pos;
light.dir.normalize();
normal = colortex.range(-1,1);
color = light.dir^normal;
color = color.range(0,1);
result.color = color;
```

Algorithm 4. *Pixel shader* for normal map application

5 Results

The presented method has been tested with some 3D models. The expected results were obtained, so by using an object without a highly complex mesh, an image of the object with a more detailed appearance is displayed.

The obtained times are not comparable with those of present-day *software* methods, since a few milliseconds are taken by the presented method to generate the corresponding normal map.

The mesured times with *ATI NormalMapper* (1), *nVidia Melody* (5), and the proposed method using the tested models are shown in Table 1. Next, several images are shown in order to compare the quality of this method with the ATI's and nVidia's methods. For this purpose, the *Tarrasque* model at two levels of detail (725 and 6117 polygons) has been used.

Figure 3 shows the meshes of both the low resolution and high resolution models, and the high resolution model rendered.

Figure 4 displays the normal map of *Tarrasque* model generated by our method and the low resolution model with the normal map of the high resolution model created by our method. And Figures 5 and 6 display the normal maps generated by ATI's and nVidia's methods and the correspondent low resolution models with these normal maps applied.

The quality of our method is similar to that of the ATI's and nVidia's methods, as seen in the images.

Moreover, terrain and wall objects have been proved. The method works perfectly for this kind of objects, because they usually meet the requirements of this method. We show an example with *Crater* object. Figure 7 displays the high resolution model (199126 polygons), the low resolution model (9079 polygons) and the plane meshes. Figure 8 displays the normal map of the high resolution model of *Crater*. And Figure 9 shows the renders of the high resolution model and both the low resolution model and the plane with the normal map applied.

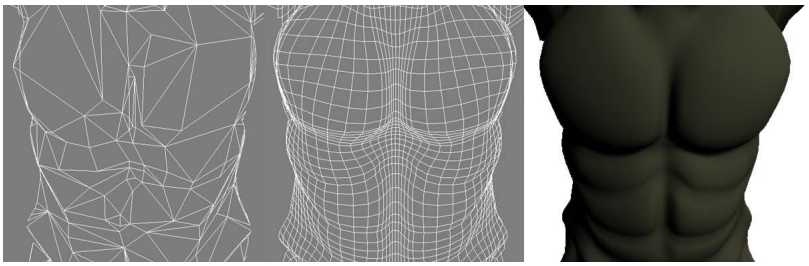

Figure 3. Low and high resolution model meshes of *Tarrasque* with the rendered model

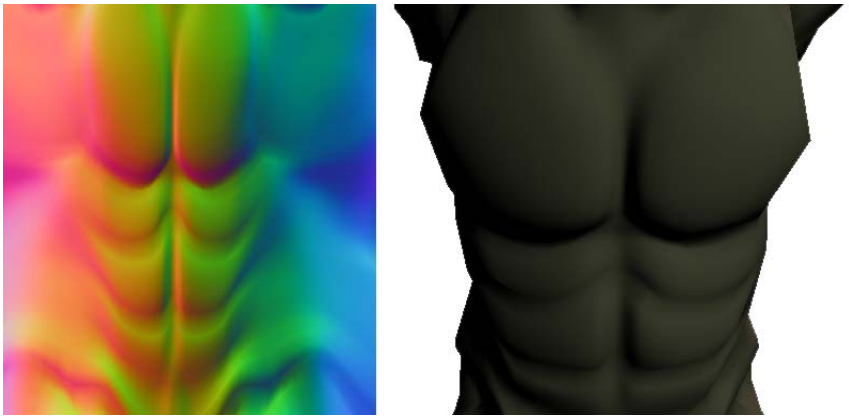

Figure 4. Normal map of the high resolution model generated by our method and the low resolution model with it applied

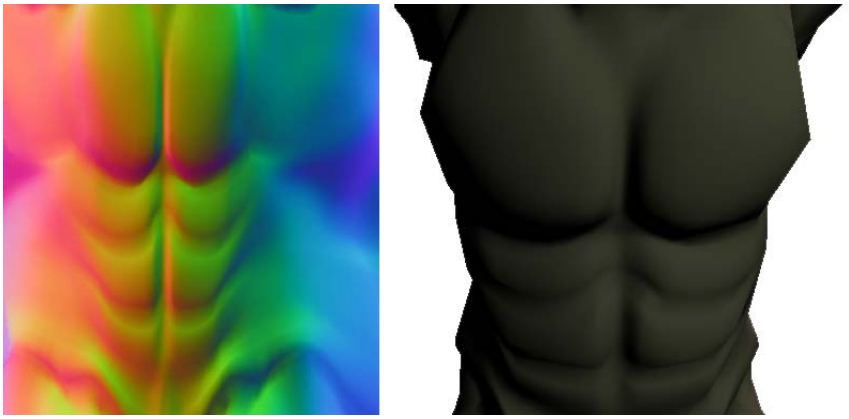

Figure 5. Normal map of the high resolution model generated by ATI's method and the low resolution model with it applied

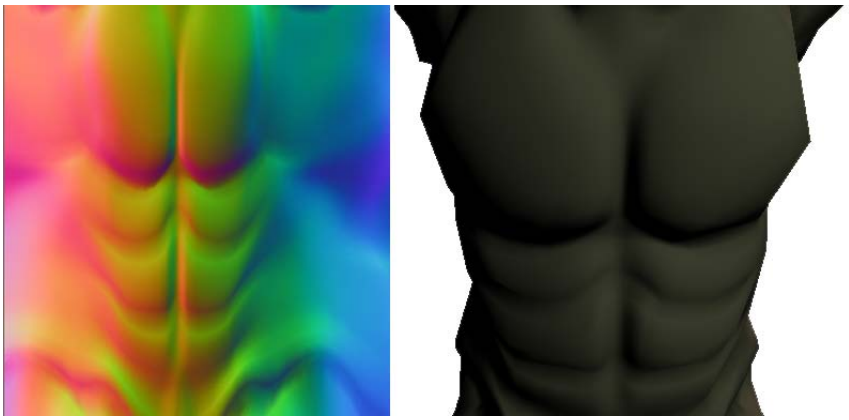

Figure 6. Normal map of the high resolution model generated by nVidia's method and the low resolution model with it applied

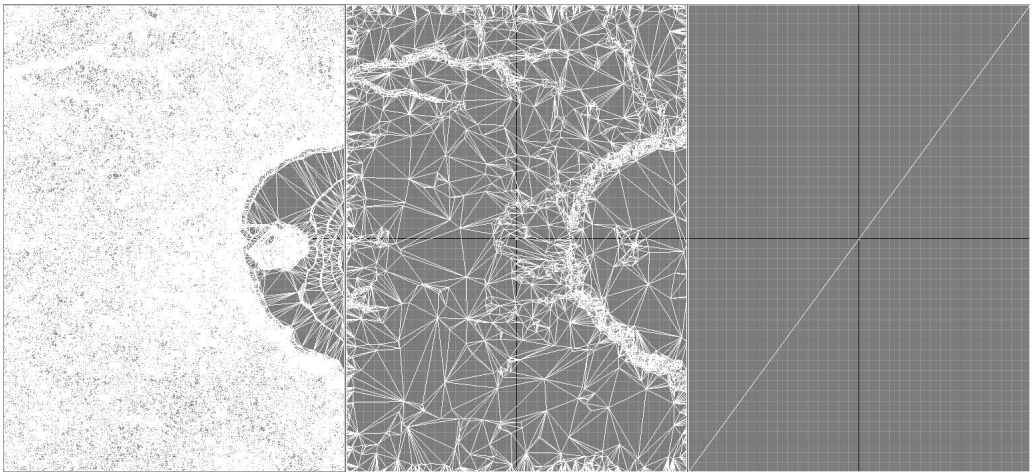

Figure 7. Low and high resolution model meshes of *Crater* and a plane object

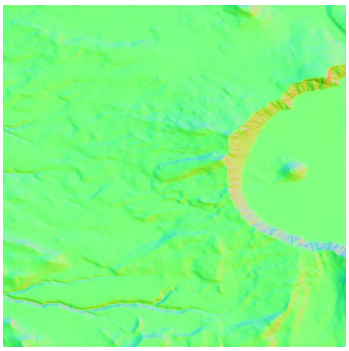

Figure 8. Normal map of the high resolution model of *Crater*

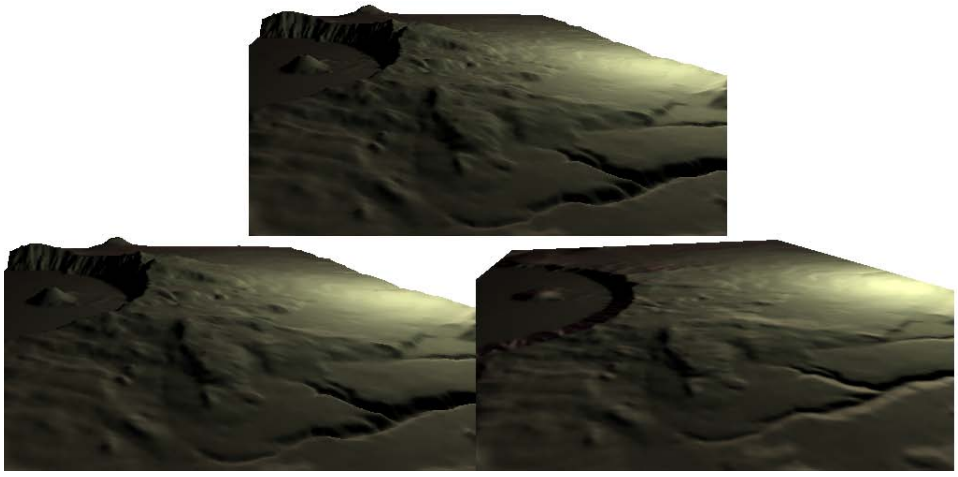

Figure 9. Renders of the high resolution model (above), low resolution model with the normal map applied (left bottom) and the plane with the normal map (right bottom)

Number of polygons of the low resolution model	Number of polygons of the high resolution model	Time of our method	Time of ATI	Time of nVidia
536	1696	less than 1	16850	16306
4274	7910	less than 1	24125	50589
23378	48048	31	97704	160975
20517	61644	31	129969	179569

Table 1
Table of times in milliseconds of normal map generation

References

[1] ATI. Normal Mapper Tool, 2002. *http://www.ati.com/developer/tools.htmlAkeley*

[2] Ernst, I., Jackèl, D., Rüsseler, H., Wittig, O. Hardware-supported bump mapping. *Computers and Graphics 20*, 1996, núm. 4, pp. 515-521

[3] Hirche, J., Ehlert, A., Guthe, S. Hardware accelerated per-pixel displacement mapping. *Graphics Interface 2004*

[4] Igarashi, T., Cosgrove, D. Adaptative unwrapping for interactive texture painting. *Symposium on Interactive 3D Graphics 2001*, pp. 209-216

[5] nVidia. nVidia Melody User Guide, 2004. *http://developer.nvidia.com/object/melody_ home.html*

[6] Peercy, M., Airey, J., Cabral, B. Efficient bump mapping hardware. *SIGGRAPH '97 Conference Proceedings*. ISBN 0-89791-896-7, pp. 303-306

[7] Sander, P. V., Snyder, J., Gortler, S. J., Hoppe, H. Texture mapping progressive meshes. *ACM SIGGRAPH 2001*, pp. 409-416

[8] Schröcker, G. Hardware Accelerated per pixel shading. *CESCG 2002*

[9] Sloan, P.-P., Weinsteun, D. Brederson, J. Importance driven texture coordinate optimization. *Computer Graphics Forum (Proceedings if Eurographics '98)* 17(3), pp. 97-104

[10] Tarini, M., Cignoni, P., Rocchini, C., Scopigno, R. Real time, Accurate, multi-featured rendering of bump mapped surfaces. *Eurographics 2000*. Volume 19, Number 3

[11] Viola, I. Applications of hardware accelerated filtering. *CESCG 2002*