Gouraud Bump Mapping

I. Ernst[§], H. Rüsseler[§], H. Schulz[§], O. Wittig[&]

[§]German National Research Center for Computer Science,
Institute for Computer Architecture and Software Technology (GMD FIRST)
[&]3Dlabs Inc.

Abstract

In this paper a new low cost bump mapping hardware is presented. The new hardware approach does not rely on per pixel lighting, but instead uses Gouraud interpolated triangles. The bump mapping effect is applied by blending the calculated per pixel bump map color onto the fragment's color. This allows real-time animated distant light-sources to react on the specified bump map.
The paper further investigates a number of different variants of recently proposed bump engines. These variants range from low-end PC solution to highest quality high-end solutions.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Hardware Architecture - Graphics processors; I.3.3 [Computer Graphics] Picture/Image Generation -Viewing and Display algorithms; I.3.7 [Computer Graphics]]Three-Dimensional Graphics and Realism - Color, shading, shadowing, texture.

Additional Keywords: real-time bump mapping, normal vector interpolation, Gouraud bump mapping, Gouraud shading hardware.

1. INTRODUCTION

Driven by continuously improving VLSI technology and the customers' wish for quality, interactive graphics need new graphic architectures must evolve. Universal pixel rate lighting, known as Phong shading with bump mapping, displacement mapping, shadow casting etc., illuminated by various types of lights, like spots and area-lights, will be the basic operational principle of future micro-architectures.

While hardware bump mapping based on micro-surface orientation, viewer and light(s) positions lies mainly in the domain of current research prototypes (Ikedo's ParimsGr concept, see http://www.parims.com or [5]), the mainstream graphics industry requires a bump mapping approach that is easily adapted into their current Gouraud architectures, without the need of pixel geometry as an intermediate step.

1998 Workshop on Graphics Hardware Lisbon Portugal

In this paper, we briefly describe the algorithms of recently published bump mapping architectures and present the design of a new bump mapping engine which overcomes the need for per-pixel lighting hardware, preserving excellent image quality.

2. CLASSIC BUMP MAPPING

This section describes the essence of the classic Blinn bump mapping approach, which is the base of all the following hardware adaptations. Further details are given in [1].

Blinn uses a height field to simulate wrinkled surfaces where each height value represents the difference in height to the given surface.

This height field perturbs the surface along its normal vector $N = (P_u \times P_v)/(P_u \times P_v)$.

Given the height field F the Position P' on the surface becomes

$$P' = P + F \cdot N / \|N\|$$

where F is a height function displacing P by an amount of $F(u,v)$ in the direction of the surface normal N.

The new surface normal is then given by
$N' = P'_u \times P'_v$ where P'_u and P'_v are the partial derivatives of the perturbed surface. P'_u (P'_v not shown) then becomes

$$\begin{aligned} P'_u &= \frac{\partial}{\partial u} P + F \cdot N / \|N\| \\ &= P_u + F_u \cdot N / \|N\| + F \cdot (N / \|N\|)_u \end{aligned}$$

Applying simplifications to the above derivative, the equations result in
$P'_u = P_u + F_u \cdot N / \|N\|$ where F_u is the partial derivative of the height function.

The new normal N' then is:

$$\begin{aligned} N' = &(P_u \times P_v) + \\ &F_u \cdot (N \times P_v)/\|N\| + \\ &F_v \cdot (P_u \times N)/\|N\| + 0 \end{aligned}$$

Thus $(P_u \times P_v) = N$, we get

$$N' = N + D$$

$$D = \left(F_u \cdot (N \times P_v) + F_v \cdot (P_u \times N)\right) / \|N\|$$

as the final perturbed normal-vector.

3. REAL-TIME BUMP MAPPING

Real-time bump mapping recently became an interesting field of research after Microsoft and SGI[2] proposed their solution to adapt the classic Blinn approach to hardware.

This section briefly describes the algorithms introduced by :

- University of Tuebingen: TexRam Bump Mapping
- GMD: Visa+ Real-Time Bump Mapping
- Microsoft Direct3D 6: Bump Mapping
- Silicon Graphics: Bump Mapping Hardware
- Evans and Sutherland: Harmony Bump Mapping

3.1 TexRam Bump Mapping

An interesting real-time bump mapping approach for normal vector shaders is to use an orthonormal system with a main direction **m** [3].

The idea is to replace the local co-ordinate system through an orthogonal system that is derived from the interpolated normal vector N and a direction **m**, which is, for example homogeneously interpolated over the triangle.

The obvious advantage of using the main direction **m** is that both $P_v \times N$ and $N \times P_u$ are no longer held constant over the triangle. This reduces the edge-discontinuity artifact described in [4].

However since the orthogonal system is constructed from two unit vectors e_1 and e_2 perpendicular to N, which have to be calculated on a per pixel basis, the extra hardware becomes a consideration.

Given the interpolated per vertex normal N_i and the main direction **m** the following formulae lead to the final perturbation:

$N = \frac{N_i}{\|N_i\|}$; $e_1 = \frac{\mathbf{m} \times N}{\|\mathbf{m} \times N\|}$; $e_2 = N \times e_1$ gives e_2 defined in a plane by N and **m** and e_1 perpendicular to that plane.

Further simplification avoids the normalization by multiplying with $\|N_i\|$ and widening the vector formats, this gives:

$$N \cdot \|N_i\| = N_i$$

$$e_1 \cdot \|N_i\| = \frac{\mathbf{m} \times N}{\|\mathbf{m} \times N\|} \cdot \|N_i\|$$

$$e_2 \cdot \|N_i\| = N_i \times e_1$$

3.2 Visa+ Real-Time Bump Mapping

In 1996 bump mapping for the VISA+ graphics accelerator was first proposed in [4].

This method uses precalculated derivatives, offset vectors, stored in a bump map similar to the TexRam approach described in section 3.1.

Since this architecture is based on normal-vector interpolation and per pixel lighting, the tangent plane of the surface is already given by N.

The resulting perturbed normal vector then becomes $N' = N + V$, where V is the perturbation vector $V = [\Delta u \Delta v]^T$.

The remaining problem is to guarantee the perturbation in the direction of the tangent plane at a given point. This is done by aligning the texture coordinate system u,v,w to the tangent plane system P_u, P_v and N.

This transformation can be done using an alignment matrix **A**. Given **A** the resulting normal vector becomes:
$N' = N + \mathbf{A} \cdot V$.

The construction of **A** simplifies Blinn's formulation in such a way that $P_v \times N$ and $N \times P_u$ are held constant over the triangle.
A is calculated based on a local orthogonal coordinate system constructed by the local surface normal N and the texture coordinates of the triangle. Therefore, the perturbation vector V is rotated constantly over the triangle. By adding the aligned perturbation vector to the interpolated per pixel normal the simulated curvature of the triangle is preserved.

This leads to minor visible discontinuities at the edges of two adjacent triangles if the angle between the spanned planes exceeds a certain threshold. This threshold is around 30 degrees for extreme lighting situations (high material shininess and mirror reflection of the light) and leads to artifacts as described in [4].
Full control (scaling, rotation etc.) over the bump map is given through the matrix **A** which is calculated on a per triangle basis.

Using quaternions at the triangle vertices for **A** and interpolating them over the triangle removes this artifact [5].

3.3 Microsoft Bump Mapping

This is more accurately referred to as per-pixel texture coordinate perturbation for diffuse and specular environment maps.

This technique allows the lighting environment of the scene to be represented in an image environment map (either for diffuse or specular effects). This permits the lights to be of any number, shape, color, or intensity distribution that can be represented in such a map.

In this pure lighting map approach, only distant light sources are supported and the rendering quality is limited to the representation of the light source image inside the texture map. Reasonable lighting quality for multiple light sources can only be achieved if higher resolution textures are used.
Using six sided environment textures further increases this texture demand providing better light source direction handling.

One main disadvantage is that light sources need to be stationary. Whenever a light source is turned on/off or the direction changed the lighting maps have to be recalculated or reloaded.

3.4 Silicon Graphics Bump Mapping

SGI reformulates the classic Blinn approach in such a way that only a perturbed normal vector field is needed as a bump-normal map (three component texture map).
The principal idea is that illumination models are a function of vector operations that can be calculated relative to any coordinate system (reference frame).

The first iteration of the algorithm chooses tangent space *TS* in point *P* as a reference coordinate system given by the unperturbed normal vector *N*, a tangent vector *T* and a binormal vector *B*:

$$TS = (T, B, N) \text{ where}$$
$$T = P_u / \|P_u\|$$
$$B = N \times T$$

This tangent space forms an orthonormal basis where each point on the surface is given as:

$$N'_{TS} = (x, y, z) \div \sqrt{x^2 + y^2 + z^2}$$
$$x = -f_u \cdot (B \cdot P_v)$$
$$y = -(f_v \cdot \|P_u\| - f_u \cdot (T \cdot P_v))$$
$$z = \|P_u \times P_v\|$$

Since the normal vectors lie in tangent space the light vector *L* and the half vector *H* have to be transformed into tangent space, which is done, using the following equation.

$V_{TS} = V \cdot (T, B, N)$ where *V* is substituted by *L* or *H* respectively.

The transformed per vertex parameters are homogeneously interpolated over the triangle to get the best results.

However choosing tangent space as a reference frame leads to a surface dependency of the stored perturbed normal vectors. Furthermore, the transformation for each parameter is different for each triangle vertex that has to be calculated in the geometry engine.

To overcome the tangent space dependency the second iteration of the algorithm introduces object space as a reference frame. Using object space gives the advantage that the matrix applied to the light and half vector is shared by all vertices over the triangle.

This saves a great amount of geometry engine computation but does not remove the surface dependency.

To overcome the surface dependency the final algorithm orthonormalizes P_u and P_v such that:

$P_u \cdot P_v = T \cdot P_v = 0$ and $\|P_u\| = \|P_v\| = 1$.

The perturbed bump normals then result in:

$$N'_{TS} = (x, y, z) \div \sqrt{x^2 + y^2 + z^2}$$
$$x = -kf_u$$
$$y = -kf_v$$
$$z = k^2$$

Equation 1 :Final bump normal equation.

This simplified equation leads exactly to the algorithm described in [6].

3.5 Harmony Bump Mapping

Harmony bump mapping [7] simply interpolates the per vertex coordinate systems across the triangle. The three axes spanning the coordinate system are:

$$-P_v \times N$$
$$-N \times P_u$$
$$P_u \times P_v$$

Note that these are the components of Blinn's final equation (see section 1).

The illumination calculation is done by using the interpolated light and half vectors. For each light-source, an interpolated light direction is needed.

4. GOURAUD BUMP MAPPING

An alternative scheme for cost effective bump mapping is presented in section 5. This stores object-space local normal vectors/perturbations in the image data similar to that proposed in section 3.4.

This method is more flexible than the Microsoft approach, since light-source directions, colors and the material shininess can be changed without loss of performance.

Nevertheless, it is still limited to distant light-sources relative to one triangle to keep the necessary hardware at a minimum. Furthermore only a limited number of light-sources (usualy two to four) can be implemented in parallel to cut down hardware costs.

Since we transform all the vectors into bump vector space, an orthogonal system is already given (compared to P_u, P_v as partial derivatives, see 3.4).

The bump normals stored in the bump map are calculated using the following formula:

$$N'_{TS} = (x, y, z) \div \sqrt{x^2 + y^2 + z^2}$$
$$x = f_u$$
$$y = f_v$$
$$z = 1$$

Equation 2 : Gouraud bump normal equation

Note that this formula equals Equation 1 in 3.4 with k = 1.

The construction of the transformation matrix to transform half and light vectors into bump vector space is described in detail in [6].

4.1 Gouraud Bump Mapping Algorithm

The simplicity of the Gouraud Bump Mapping algorithm keeps the extra hardware needed to render bump mapped triangles at a minimum.

Using the reordered graphics pipeline in section 5.1, two new units are added to the Gouraud rasterizer. Both, the diffuse and the specular blending units need bump-state information, which consists of the light-source direction, light-source color and the material's shininess.

The perturbed normal is read from the bump map and the half vector is transferred to the rasterizer with the triangle data or it may be kept in the bump-state. Given the half-vector and the normal in a point on the triangle the lighting equation can be calculated.

The lighting model can be simplified by using a parallel viewer and parallel light sources. Using the simplified model, the half-vector and the light direction can be stored in the bump state to reduce the data transfer to the rasterizer.

4.2 Blending Unit

The blending unit combines the Gouraud color with the lighting result given by the bump map normal B_i, the light direction L and the half-vector H_i.

The function used to calculate the diffuse bump lighting intensity is:

$$fd_i = B_i \cdot L$$

The specular intensity uses the material exponent to simulate shiny surfaces and results in:

$$fs_i = (B_i \cdot H_i)^n$$

Given the light source color L_i the final lighting color can be calculated.

The final fragment blending is done by the following equation:

$$C_{new} = C_{src} + \sum_i (fd_i + fs_i) \cdot k_i \cdot L_i$$

This function calculates the final Gouraud Bump color for i light sources.

C_{src} is the interpolated Gouraud color output of the Color DDA Unit (C represents R, G, B). Only light-sources which are not effected by the bump map generate C.

C_{new} is the final blended Gouraud Bump color.

fd_i represents the diffuse blending function of light source i.

fs_i represents the diffuse blending function of light source i.

k_i modulates spot and/or attenuation information according to the OpenGL specification.

L_i is the light source color of light i.

The material shininess is held constant over the triangle and is implemented as a simple power table for one shininess at a time. The table can be loaded similar to a color LUT and is equal in size to reduce costs.

k_i is used if the basic blending function is extended to improve the per pixel lighting/blending quality. One possible improvements is the simulation of cut-off angles for spotlights. In our sample implementation, it has no effect.

Both blending units can selectively be turned off to reduce the artifacts described in section 5.4 according to the color information given in the interpolated Gouraud color fragment C_{src}.

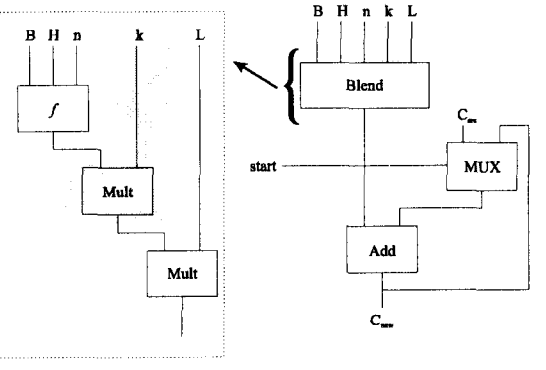

Figure 1: Gouraud Bump Mapping block diagram

Figure 1 gives an impression of the additional hardware units. Depending on the accuracy of all relevant input fields (B, H, n and color) and the fidelity of the specular intensity generator, the gate budget lies in the range from 10,000 to 25,000 gate equivalents. Multiple light sources can be implemented by accumulation of all relevant light fragments and final addition with the interpolated Gouraud color. n light sources are blended in n clock cycles, avoiding n times the hardware. Note that H is constant within at least one triangle, so bandwidth will increase to three 16-bit words per triangle, or about less than 5% of a typical textured Gouraud triangle data-set.

5. INSIDE GOURAUD BUMP MAPPING

This section provides recommendations to improve the rendering quality of Gouraud bump mapped triangles taking into consideration the artifacts found using the algorithm.

5.1 Pipeline re-ordering

A rendering pipeline for a Gouraud rasterizer usually gets the interpolated Gouraud color and blends the texture color thereafter. Thus, the pipeline ordering is *ColorDDA – Texture – Blend.*

Using Gouraud, bump mapping the pipeline has to be changed slightly.

The best results can be achieved if the interpolated fragment is first blended with the diffuse bump color and after the texture is applied it is blended with the specular bump color. This color splitting is similar to the Kd, Ks lighting.

The final pipeline looks as follows:

The following pictures show the necessity of calculating both, the diffuse and the specular bump blend color as shown in the final pipeline. Note that blending the diffuse and specular color together after the texture pass has no effect on black texture color due to the multiplication (see Figure 3). Figure 2 shows the desired result using the split blending pipeline.

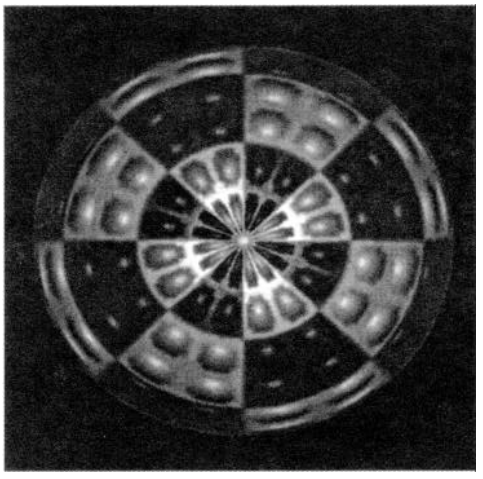

Figure 2: Diffuse and Specular Gouraud Bump Blending using DiffuseBlend-Texture-SpecularBlend pipeline ordering.

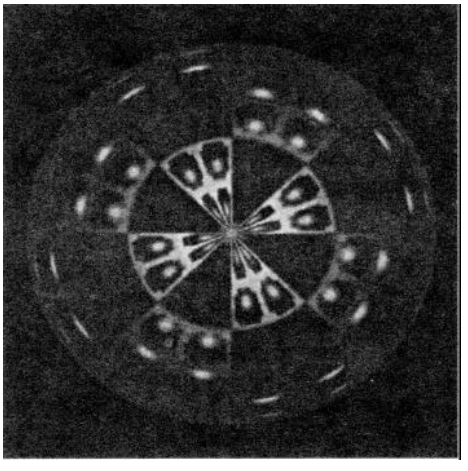

Figure 3: Specular Gouraud Bump Blending only using Texture-DiffuseBlend/SpecularBlend pipeline ordering.

5.2 Multiple Light Sources

Multiple light sources are supported by looping over all *n* light sources and blending the incoming color fragment *n* times with the diffuse and the specular bump color respectively (see Figure 4).

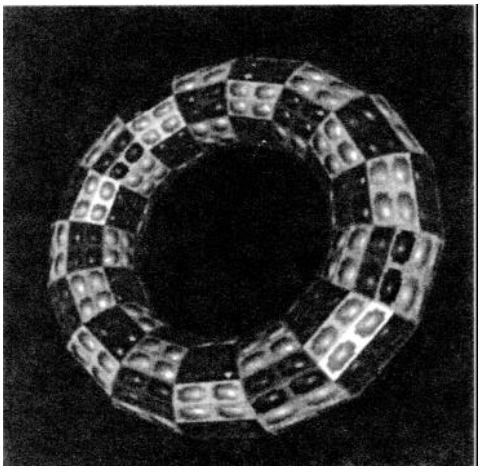

Figure 4: Torus with sphere cap bump map illuminated by two light-sources. White light-source from right, purple light-source from left.

5.3 Natural Surface Simulation

Bump maps do not necessarily add roughness or structure to a surface. Instead, we can use it to simulate smooth surfaces.

The image in Figure 5 B shows how to define a bump map, which generates a smooth shaded cylinder (see Figure 8) that can only be achieved using per pixel lighting hardware.

Dependent on the type of the object, bump maps can be held small and used repeatedly similar to texture maps (see Figure 5 B).

Due to the full texture filtering capability of bump maps, smoothly interpolated bump-vectors are used to avoid local flat shading effects (see Figure 5 A).

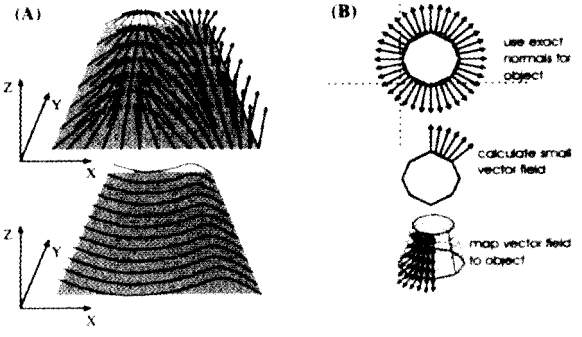

Figure 5: Natural Surface Simulation

5.4 Artifacts

To discuss the artifacts we first recall all the simplifications:

1. Gouraud color is used instead of Phong interpolated color
2. Half vector held constant over the triangle (viewer is parallel over the triangle). Only distant light sources relative to the triangle are supported which react on the bump map.
3. P_u and P_v are orthogonal unit vectors

5.4.1 Gouraud Color Artifact

Gouraud Bump Mapping blends the diffuse and the specular color given by the lighting geometry on top of the interpolated Gouraud color. This gives the well-known mach-banding effect. However since Gouraud Bump Mapping is limited to parallel light-sources and triangle parallel viewer the Gouraud color can add spot-light and other lighting capabilities to the bump mapped triangle (see Figure 6).

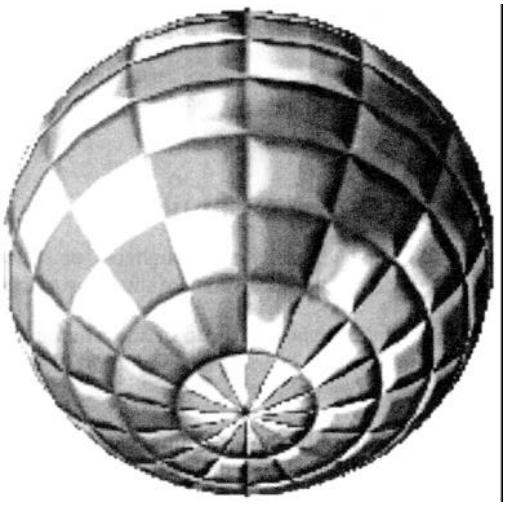

Figure 6: Sphere with brick bump map. Only the white light-source is used for the bump map effect. The red and green spot-light is given by the Gouraud color.

5.4.2 Parallel Viewer and Light Source

Given a parallel viewer and a parallel light-source results in flat shading artifacts which can be seen in Figure 7. This artifact can be avoided by simply adding only the bump blending effect which has a perturbed normal stored in the bump map and leaving the fragments that contain unperturbed normals untouched. According to Equation 1 the unperturbed normal vector is [0,0,1.0].

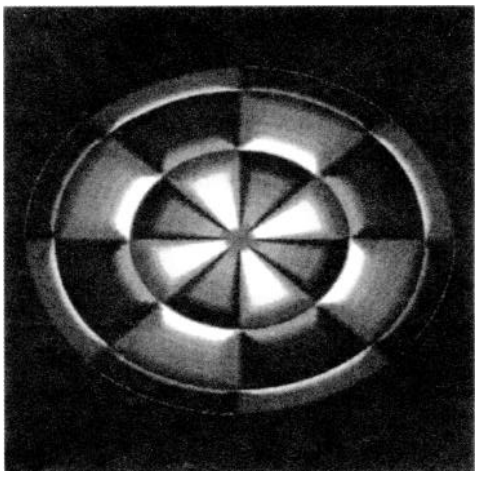

Figure 7: Flat shading artifact using constant bump-normals on a curved surface.

Likewise it is possible to calculate the correct surface-normal within the bump–map (see 5.3). A bump mapped cylinder for example contains the computed surface normals of the cylinder patch plus the bump-structure. Nevertheless doing so, we added a surface dependence to the bump map.

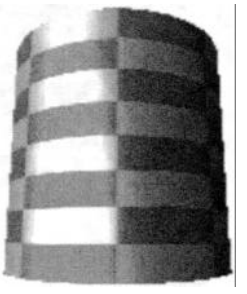

Figure 8: Simulated smooth shading using Gouraud Bump Mapping

5.4.3 Orthogonal Axis P_u and P_v

The orthogonal axis' removes the surface dependency of the bump map but automatically limits the models to be rendered to be square patched. These include spheres, tori, surfaces of revolution and flat rectangles.

6. CONCLUSIONS AND FUTURE WORK

We have presented a simple but efficient scheme for bump mapping on common Gouraud renderers. The additional hardware overhead is small and is justified by the resulting quality in both bump mapped surfaces and specular lighting (see Figure 9). The memory requirements for the bump maps can be minimized by using palletized bumps. With eight bit addressing a table lookup (bump) texture, up to 256 surface orientations can be modeled. Further improvements can be made by interpolating the half vec-

tor (no flat shading effect) and/or by interpolating the light directions, one per light-source.

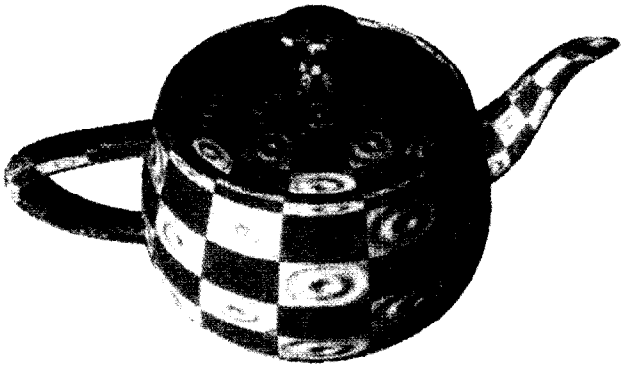

Figure 9: Teapot with mexican hat bump map

Acknowledgments

We are grateful to Markus Szymaniak, Mark K. Newholm, Robert Sharp and S. Budianto for their suggestions and contributions to this work.

References

[1] J. F. Blinn, "Simulation of wrinkled surfaces," in SIGGRAPH 78, pp. 286–292, 1978.

[2] Mark Peercy, John Airey, Brian Cabral, Efficient Bump Mapping Hardware, Proceedings of SIGGRAPH 97 (Los Angeles, California, August 3-8, 1997). In Computer Graphics Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 303-306

[3] Schilling, A., Knittel, G., Straßer, W., Texram: A Smart Memory for Texturing, Computer Graphics & Applications, May 1996, pp. 32-41.

[4] I. Ernst, D. Jackèl, H. Rüsseler, O. Wittig, Hardware Supported Bump Mapping: A Step towards Higher Quality Real-Time Rendering, In 10th EuroGraphics Workshop on Graphics Hardware, EuroGraphics, pp. 63-70 (1975)

[5] O. Wittig, I. Ernst, D. Jackel, "Bildverarbeitungsverfahren zur Simulation einer Tiefenstruktur und zugehoerige Vorrichtung", 25.8.1995, Patentanmeldung 196 06 356.6

[6] I. Ernst, "Verfahren und Vorrichtung zur Darstellung computermodellierter Objekte", 20.3.1997, Patentanmeldung 197 13 466.1-53

[7] M. A. Cosman and R. L. Grange, "CIG scene realism: the world tomorrow," Proceedings of 18th I/ITEC Conference, 1996.