

Towards Real-Time Photorealistic Rendering: Challenges and Solutions

Andreas Schilling, Universität Tübingen, WSI/GRIS¹

ABSTRACT

A growing number of real-time applications need graphics with photorealistic quality, especially in the field of training (virtual operation, driving and flight simulation), but also in the areas of design or ergonomic research. We take a closer look at main deficiencies of today's real time graphics hardware and present solutions for several of the identified problems in the areas of antialiasing and texture-, bump- and reflection mapping. In the second part of the paper, a new method for antialiasing bump maps is explained in more detail.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture – Graphics Processors, Raster Display Devices; I.3.3 [Computer Graphics]: Picture/Image generation.

Additional Key Words and Phrases: antialiasing, bump mapping, environment mapping, anisotropic filtering.

1. INTRODUCTION

Using elaborate software techniques, it is possible to render images that resemble photographs. These techniques are commonly applied in the movie industry, but unfortunately, they are not feasible for hardware implementation. A recently rendered movie took a week of rendering for 3.5 minutes of the movie on the PIXAR RenderFarm.

Nevertheless, high quality is needed in several application areas as, e.g., ergonomics research, accident research or training.

But even expensive high-end graphics hardware suffers from some important deficiencies.

These deficiencies provide the graphics hardware researcher with interesting challenges for the next few years.

2. MAIN DEFICIENCIES OF REAL TIME GRAPHICS - CHALLENGES FOR GRAPHICS HARDWARE DESIGN.

The following sections identify some of the challenges for computer graphics hardware and point towards solutions.

Complexity

With today's technology, it is easy to create scenes with hundreds of millions of triangles. Using 3D-scanning, even more complex scenes can be captured.

One answer to the problem of complexity is, of course, speed. Replacing off-chip data-transfer by on-chip-transfer will provide us with speedups of orders of magnitude within the next few years [9]. But even such speedups will not suffice for the growing demands. Methods to reduce the number of triangles that have to be rendered will therefore become more and more important: *Simplification* and *Visibility Culling*. Texture mapping and image-based rendering are already state of the art and will be treated in an own section. Mesh simplification and multiresolution modeling [7][8] are however performed in software and are both well suited to general-purpose architectures. Together with visibility culling, these software techniques are an important part of Silicon Graphics new OpenGL++ or Optimizer package. Visibility culling, especially occlusion culling is up to now also performed in software, but some of the algorithms could probably well be supported with suitable hardware.

Texture Filtering

Texture Mapping is a common means to improve the appearance of computer generated images. It allows to reduce the number of rendered polygons significantly, especially in combination with mesh simplification techniques. Special techniques, like the imposters proposed by Schaufler et.al. [10] allow to exploit frame coherency using texture mapping. Hardware support for texture mapping can be found even on cheap graphics boards for the PC. It is, however, astonishing that from low-end to high-end systems, the hardware supports only isotropic filtering with mip-maps. Isotropic filtering leads to very blurry images, if the viewing angle is small. We have proposed footprint-assembly, as a simple method to perform anisotropic filtering that requires only minimal modifications to the mip-mapping hardware [11]. For a comparison of the results see Fig. 1 and Fig. 2. New, completely image based rendering techniques [5] can be supported by slightly

¹Auf der Morgenstelle 10/C9, 72076 Tübingen, Germany,
e-mail: schilling@wsi.tuebingen.de

modified texture mapping hardware. Further research in this area is needed.



Fig. 1: A puzzle for kids: can you read the secret writing in the upper right corner? The puzzle can be solved by looking in a flat angle on the writing..

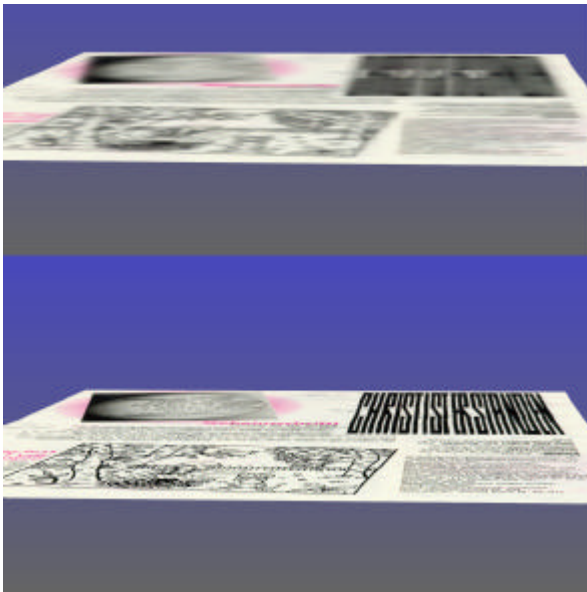


Fig. 2: Top: solving the puzzle with standard mip-mapping, as applied in cheap but also in very expensive systems. Bottom: solving the puzzle with footprint-assembly

Surface appearance

An other challenge is the appearance of surfaces. Often, it is a single surface that makes it easy to tell a computer generated image from a photograph. This is true even for ray-traced images, that

can often be recognized by perfect reflecting surfaces. For real-time graphics, a way out is bump-mapping, in combination with environment mapping. Hardware support for bump-mapping is state of the art, and will soon appear in the PC-graphics arena. Unfortunately, antialiasing of bump maps is still a serious problem, as will be shown in the following sections. In the case of environment mapping, the problem is even more important, as the sampling rate on the environment map can change by orders of magnitudes within the same object due to changes of the curvature of the object. But this makes the problem also more difficult to solve. Traditionally, the filtering, which often is performed with mip-maps, has to be adjusted manually to get the desired effect. If no anisotropic antialiasing is possible, artifacts are unavoidable. A simple example is the environment mapping on a cylinder, which has a curvature only in one direction. In this direction we get a very low sampling rate, which requires massive filtering. In the other direction, parallel to the axis of the cylinder, the sampling rate is higher and less filtering would be required. If environment mapping is combined with bump mapping, even more realistic images are possible. On the other hand, the antialiasing is becoming more difficult, as a bumpy surface reflects incoming rays into a larger sector of space, which corresponds to a larger area of the environment map. We have developed a solution that takes into account the curvature and the dispersion by the bump map and is simple enough to realize it in hardware [12]. In the second part of this paper, our solution for the antialiasing of bump maps is presented in more detail.

3. ANTIALIASING OF BUMP MAPS

As mentioned, bump-maps, like texture maps or any other maps consisting of discretely stored data have to be properly filtered, if they are being resampled in the process of rendering an image. If, however, bump maps are filtered in the traditional way, the bumps are lost in the filtering process and the result is a smooth surface. We introduce a bump map pyramid, that contains and preserves isotropic or anisotropic roughness information in all resolution levels.

4. PROBLEM

The antialiasing of bump maps is similarly important as the antialiasing of textures in general (Fig. 3). However, filtering bump maps in the common way doesn't lead to the desired result, but instead results in a smoothened surface. The lower the resolution gets, the more we have to filter, the larger are the bumps that disappear (Fig. 4). So filtering bump maps means to remove the bumps. In reality however, a bumpy surface can be told from a smooth surface, even if it is viewed from a larger distance, where the individual bumps are not discernible anymore (Fig. 5). What can be seen in this case, is the bumpiness or roughness of the surface. The problem is, that filtering removes not only the individual bumps, but also the bumpiness. Bump maps can be represented as height- or offset-maps [4]. We prefer a representation that uses a two-dimensional vector field of offset vectors to be added to the normal vector, which is described

below. But regardless of how a bump map is represented, filtering or averaging always leads to the mentioned effects.

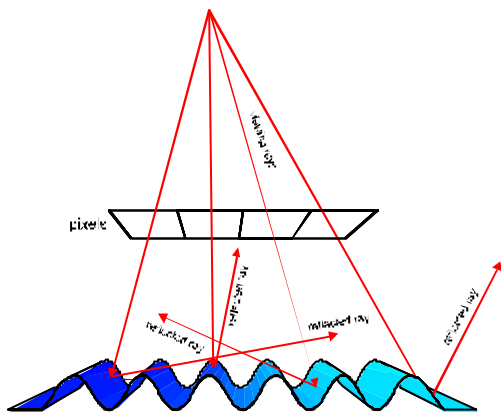


Fig. 3: Antialiasing of bump maps is necessary; otherwise we get rays reflected into random directions or random Phong illumination values.

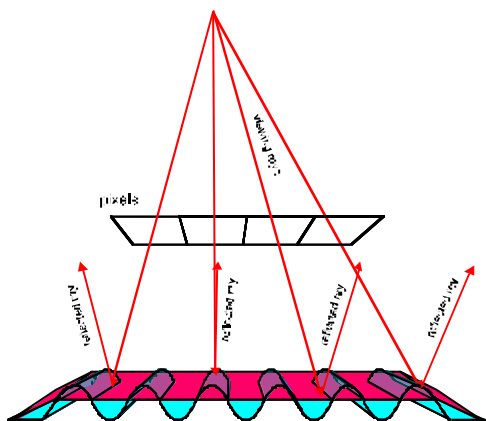


Fig. 4: Filtering the bump map removes bumps and produces a smooth surface

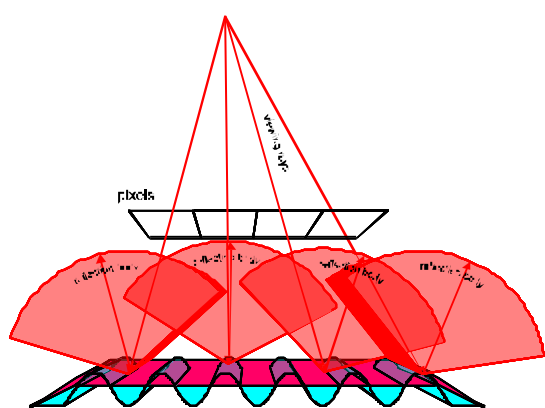


Fig. 5: The roughness of a surface causes rays from more than only one direction to be reflected into one pixel. These rays form a reflection cone or a conical reflection body

5. RELATED WORK

Becker and Max [1] address the problem of rendering bump mapped surfaces in multiple resolutions. They switch between unfiltered bump maps and a BRDF (bi-directional reflection distribution function). In order to avoid inconsistencies between the two models, they use different redistribution functions for each viewing angle to modify the standard bump mapping. For a smooth transition, they blend between the results of the two algorithms. The use of the BRDF is the correct solution for rendering bumps that are too small to be discernible at a given resolution; however, the method is very expensive. In addition, it works only for bump maps containing only a narrow range of frequencies. Only in this case, a single transition point between bump mapping and using a BRDF can be determined. Therefore, bump maps containing a broader range of bump frequencies have to be broken up into multiple maps, each limited to a narrow frequency range.

In his cone-tracing paper [5], Kirk uses cones instead of simple rays for the purpose of antialiasing. He even mentions the application for bump mapping, although he does not account for the roughness but only for the curvature of the surface (difference between normal vectors at the cone centerline intersection and the cone edge). In this paper, we use reflection bodies that could be regarded as a generalization of Kirk's cones to understand and describe the antialiasing of bump-maps.

6. A NEW SOLUTION

If we want to preserve the important surface properties, in addition to the bump map we have to introduce and use a new type of maps: the roughness map. It stores the variance of the normal directions from the area that contributes to one sample in the filtered version of the bump map. Consider for example a lake, seen from some distance. Even if there are waves, the surface of the water within one pixels region can be represented by only one normal vector, if standard bump mapping is used. If this pixel region is large enough, this average normal vector will always point straight up. So we will get the same reflections of the sun (or any environment) as in a perfect mirror, regardless of the presence or absence of small waves. If we store, however, the roughness of the surface, we can account for the waves and get a better approximation of the lake than a perfect mirror would be. We could use the roughness as a measure for the exponent in the Blinn-Phong model[3]. A better way will be described in [12], where environment maps are used and the roughness serves as a parameter for the anisotropic antialiasing of the environment map.

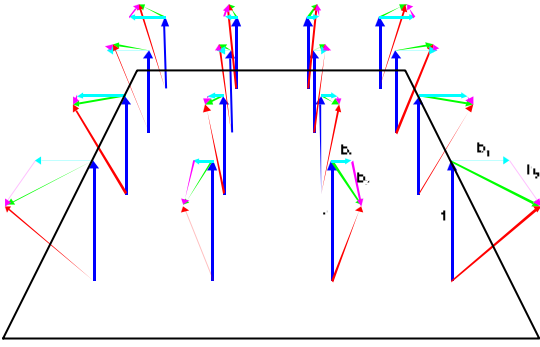


Fig. 6: Representation of bump map with offset vectors.

In this paper, we will first show a practical representation of bump maps, then introduce an anisotropic measure for the roughness, and then show, how resolution pyramids can be calculated and practically stored. We offer a cheap, isotropic and a more expensive anisotropic version for the bump-roughness pyramid. Examples will illustrate the different methods.

7. THE REPRESENTATION OF BUMP MAPS.

Bump mapping, even without antialiasing or environment mapping is not commonly used in real-time systems due to its heavy demands on the computing resources. This is partly, because the traditional approach to bump mapping [4] includes the calculation of the derivatives of the bump function.

Bump mapping with Precalculated Derivatives

A possible solution that avoids the calculation of the derivatives of the bump function is to store precalculated derivatives [11][2]. Besides saving the calculation, this has the advantage, that the scaling of bump maps is as simple as the scaling of rgb-textures. If traditional bump maps are scaled with an unknown factor, it is impossible to calculate the derivatives any more. The most difficult problem that remains to be solved is to find an appropriate local coordinate system for each sample point, which consists of the normal vector of the surface in this point and the two tangential directions, for which the derivatives of the bump function have to be calculated (we have the same problem with the traditional representation of the bump maps). Once this coordinate system has been established, the calculation of the new normal vector is performed by adding the offset vector specified by the precalculated derivatives in the local coordinate system to the surface normal.

The Local Coordinate System

In theory, we are free to choose a suitable local coordinate system for the perturbation of the normals by the bump maps. Of course, the data stored in the bump maps is dependent of the chosen coordinate system. Unfortunately, a local coordinate system that

ideally fits all needs for bump mapping doesn't exist. But two conditions should at least be met:

- the directions of the axes should be a continuous function of the location.
- the system should allow to map a bump map on any surface of an object (with any orientation), without having to recalculate the whole bump map. (This means for example, that a bump map describing the letters of the alphabet could be used to carve words into a surface regardless of orientation or location of the writing)

In addition, it would be nice, if

- the coordinate system would be orthogonal (which is not necessarily the case for the projection of the u and v axes on the surface; this is, by the way, a problem with standard bump mapping).

For the description of some possible solutions, we will use a notation similar to that of Blinn [4]:

The points of the original patch are given by $\mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, the

partial derivatives are given by $\frac{\partial \mathbf{p}}{\partial u} = \mathbf{p}_u = \begin{pmatrix} x_u \\ y_u \\ z_u \end{pmatrix}$ and \mathbf{p}_v ,

the surface normal is thus $\mathbf{n} = \mathbf{p}_u \times \mathbf{p}_v$. The sufficiently small displacement function f (with partial derivatives f_u and f_v)

defines a new surface: $\mathbf{p}' = \mathbf{p} + f \frac{\mathbf{n}}{|\mathbf{n}|}$.

The new normal vector in this notation is

$$\mathbf{n}' = \mathbf{n} + f_u \frac{\mathbf{n}}{|\mathbf{n}|} \times \mathbf{p}_v + f_v \mathbf{p}_u \times \frac{\mathbf{n}}{|\mathbf{n}|}.$$

1. The „natural“ system (perpendicular to u-lines, v-lines, n)

We define $\mathbf{s}_u = \frac{\mathbf{n}}{|\mathbf{n}|} \times \mathbf{p}_v$ and $\mathbf{s}_v = \mathbf{p}_u \times \frac{\mathbf{n}}{|\mathbf{n}|}$. We get:

$$\mathbf{n} = \mathbf{p}_u \times \mathbf{p}_v = \mathbf{s}_u \times \mathbf{s}_v \text{ and}$$

$$\mathbf{n}' = \mathbf{n} + f_u \mathbf{s}_u + f_v \mathbf{s}_v$$

The bump map stores f_u and f_v . This is classical bump mapping with possibly non-orthogonal u-v coordinate system and arbitrary $|\mathbf{p}_u|$ and $|\mathbf{p}_v|$. Accordant to the notion of f as displacement function, the „height“ of the bumps is constant and does not scale with their size in the other dimensions if the map is scaled. If the u-v mapping is continuous and „well behaved“, the

first requirement is also fulfilled and it is possible to interpolate \mathbf{S}_u and \mathbf{S}_v linearly across triangles (with perspective correction). An example for a mapping that makes problems is a sphere, if spherical coordinates are used for u and v ; at the pole, one coordinate is not unique. If \mathbf{S}_u and \mathbf{S}_v are interpolated, the lengths of \mathbf{S}_u and \mathbf{S}_v should be interpolated separately from the vectors themselves. As the perspective division doesn't change the directions of the vectors but only the lengths, only the lengths need to be corrected; the correction of the direction is performed automatically by choosing appropriately sized vectors at the vertices. The direct interpolation of two vectors with different lengths would cause the vector to change directions faster close to the end with the shorter vector.

2. The natural system with normalized unit vectors

As pointed out in Blinn's original bump mapping paper [4], we could also want the bump mapping to be scale invariant. For this, the perturbation vector has to scale at the same rate as \mathbf{n} , independent of scales in \mathbf{p} (or \mathbf{S}_u and \mathbf{S}_v). This can lead to geometrically „impossible“ normals, if the object is scaled with different scales in u and v directions, which should therefore be excluded. The best choice for the coordinate system is the same as the previous one, but with normalized lengths of \mathbf{S}_u and \mathbf{S}_v ; thus the interpolation of the lengths can be saved. Again, perspective division is not needed, as it cancels out in the normalization process.

3. An orthonormal system with main direction \mathbf{m}

An interesting alternative for the local coordinate system is an orthogonal system, that is derived from the normal vector \mathbf{n} and a main direction \mathbf{m} . The unit vectors \mathbf{e}_1 and \mathbf{e}_2 are perpendicular to \mathbf{n} . With the help of \mathbf{m} they are defined such, that \mathbf{e}_2 is perpendicular to \mathbf{m} and \mathbf{e}_1 is in the plane of \mathbf{n} and \mathbf{m} [11]. The main direction \mathbf{m} can be e.g. one of \mathbf{p}_u or \mathbf{p}_v (in this case interpolated as above), but it can also be a constant vector for a whole object. A good example is the mapping onto a sphere like e.g. the earth with spherical coordinates. The direction of the axis of the earth would serve as \mathbf{m} and we would get \mathbf{e}_1 pointing always in east-west direction, and \mathbf{e}_2 in south north direction. An important advantage of a constant main direction \mathbf{m} is that besides the normal vector, no other vector needs to be interpolated across triangles. In addition, the coordinate system can be calculated in hardware in the rasterizer/shader and needs not to be calculated at all vertices by a setup process.

The calculation of the local coordinate system $\mathbf{n}, \mathbf{e}_1, \mathbf{e}_2$ from the interpolated normal vector \mathbf{n}_I and the main direction \mathbf{m} is performed using the following formula (Fig. 7):

$$\mathbf{n} = \frac{\mathbf{n}_I}{|\mathbf{n}_I|}, \mathbf{e}_1 = \frac{\mathbf{m} \times \mathbf{n}}{|\mathbf{m} \times \mathbf{n}|}, \mathbf{e}_2 = \mathbf{n} \times \mathbf{e}_1.$$

In this way, we get two tangential vectors:

- \mathbf{e}_2 in the plane defined by \mathbf{n} and \mathbf{m} , and
- \mathbf{e}_1 perpendicular to that plane.

If the vectors need not be normalized, we multiply the three vectors by $|\mathbf{n}_I|$ for simpler calculation and get:

$$\mathbf{n}|\mathbf{n}_I| = \mathbf{n}_I, \mathbf{e}_1|\mathbf{n}_I| = \frac{\mathbf{m} \times \mathbf{n}}{|\mathbf{m} \times \mathbf{n}|}|\mathbf{n}_I|, \mathbf{e}_2|\mathbf{n}_I| = \mathbf{n}_I \times \mathbf{e}_1.$$

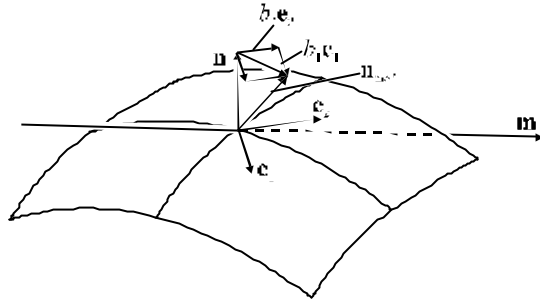


Fig. 7: The construction of the local coordinate system for the bump map using a main direction \mathbf{m} .

8. THE ROUGHNESS INFORMATION

We have now defined, how the bump maps can be represented with precalculated derivatives or perturbation vectors in a suitable local coordinate system. The next question is, how to represent the roughness information, i.e. the bumps that are too small to be represented in the current bump map level, but nevertheless are important for the visual appearance of the surface. This roughness information determines ultimately the size and shape of the reflection bodies (Fig. 5), into which the small, pixel-wide viewing beams are reflected by the rough surface. A complete description of the roughness would consist of the complete distribution of perturbation vectors within the considered area. The only difference between the roughness information and the complete detailed bump map is, that the location, where a certain perturbation vector applies, can be omitted. But the complete distribution of perturbation vectors is still much too much information to store and to use. A common way out is to assume a certain form of the distribution, which allows to represent the distribution with only a few parameters. A one-dimensional example for such parameters are average and variance in the case of a normal distribution. The counterpart of the variance in the 2D case is the two-dimensional covariance matrix \mathbf{K} , which describes the distribution in the form of an ellipse:

$(\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{K}^{-1} (\mathbf{x} - \bar{\mathbf{x}}) = 1$. On this ellipse, the probability density is constant; the distance of a point \mathbf{x} on the ellipse to the average vector $\bar{\mathbf{x}}$ represents the standard deviation in the direction $(\mathbf{x} - \bar{\mathbf{x}})$ of this point. Of course, the assumption of a normal distribution is not valid in the general case, but the

representation of the roughness with covariance matrices of the perturbation vectors gives us the possibility to represent the normal vector distribution with an elliptical cone, that is characterized by three parameters (plus two for the average direction). This representation is appropriate for most practical normal vector distributions and enables us to model anisotropic reflection effects and produce realistic images of corrugated sheet iron, brushed metal, small scrapes on a glossy surface, waves on a lake and so on.

The calculation of the covariance matrix

The covariance matrix is calculated from the derivatives f_u and f_v in the following way:

$$\mathbf{K} = \frac{1}{n} \begin{bmatrix} \sum_{i=0,n} (f_{u,i} - \bar{f}_u)^2 & \sum_{i=0,n} (f_{u,i} - \bar{f}_u)(f_{v,i} - \bar{f}_v) \\ \sum_{i=0,n} (f_{u,i} - \bar{f}_u)(f_{v,i} - \bar{f}_v) & \sum_{i=0,n} (f_{v,i} - \bar{f}_v)^2 \end{bmatrix}$$

$$= \begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

This matrix describes an ellipse, shown in Fig. 8 (the one in the

middle of the figure). The ellipse $(f_u \ f_v) \mathbf{K}^{-1} \begin{pmatrix} f_u \\ f_v \end{pmatrix} = 1$

contains about 63% of the perturbation vectors, if their distribution is a two-dimensional normal distribution. If, however, the perturbation vectors $(f_u \ f_v)$ are equally distributed in a rectangle with edge lengths a and b , the ellipse has the main axes $2a/\sqrt{3}$ and $2b/\sqrt{3}$. We can account for that effect by introducing a correction factor, which we will mention. We have now the parameters of an ellipse, that describes the distribution of the perturbation vectors and by this the distribution of the normal vectors.

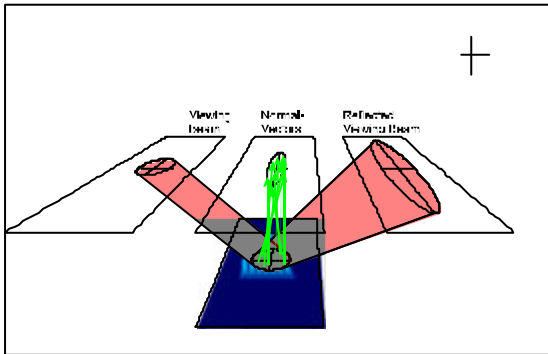


Fig. 8: The viewing rays contributing to a pixel form a beam. This beam is reflected by a bumpy surface into the reflected viewing beam. The shape of the reflected viewing beam is approximated using the distribution of normal vectors in the reflecting area by an elliptical cone. And yes: it could be argued that the normal vectors that end in the drawn ellipse, should be drawn starting from one origin, not from the surface location, they belong to. The same applies for the reflected beam.

This information can be used in a subsequent step to shade the pixel correctly. Depending on the used shading methods, there exist different possibilities to use the roughness ellipse: In a ray tracer, the ellipse can be used to perform antialiasing by supersampling. A second possibility is to modify the Blinn-Phong shading so that it uses the roughness information (see Appendix A). Of course, it makes no sense to make huge efforts calculating the bump map and then spoil the quality of the result by applying a simple shading model. A method, that leads to a balanced system with high image quality, is to use the roughness information for the anisotropic antialiasing of environment maps [12].

The roughness pyramid

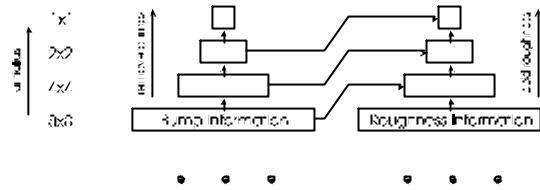


Fig. 9: Bump- and roughness pyramid

Fig. 9 shows, how the surface information is stored in two resolution pyramids, the bump pyramid and the roughness pyramid. The bump pyramid is a standard mip map, its lower resolution levels are calculated by downfiltering the higher resolution levels. In the roughness pyramid, however, the lower resolution levels contain not only the roughness information of the higher resolution roughness levels, but also the roughness information coming from the next level in the bump pyramid, i.e. the roughness information representing those bumps, that are omitted from the bump map in the current bump map level. This contribution is represented with a covariance matrix, calculated as shown above. It has to be added to the downfiltered covariance matrix from the roughness map. As a result we get the recursive rule for the calculation of the roughness covariance matrix in level l (with a 2×2 -boxfilter for creating the mip-map):

$$\mathbf{K} = \frac{1}{4} \left(\sum_{n=0,3} \mathbf{K}_{l-1,n} \left[\begin{array}{cc} \sum_{n=0,3} (f_{u,n} - \bar{f}_u)^2 & \sum_{n=0,3} (f_{u,n} - \bar{f}_u)(f_{v,n} - \bar{f}_v) \\ \sum_{n=0,3} (f_{u,n} - \bar{f}_u)(f_{v,n} - \bar{f}_v) & \sum_{n=0,3} (f_{v,n} - \bar{f}_v)^2 \end{array} \right] \right)$$

This formula is equivalent to the calculation of the covariance matrix for all f_u and f_v values in the highest resolution level for the whole area covered by the respective texel in level l .

It is, however, not convenient, to store the covariance matrices themselves in the roughness map. This has two reasons:

- the covariance matrix contains the squares of the standard deviations; to get uniform accuracy over the whole range of standard deviations a value proportional to the standard deviations has to be stored.
- if the roughness information is used for the antialiasing of environment maps, not the covariance matrix \mathbf{K} is needed, but a matrix \mathbf{D} with $\mathbf{D}\mathbf{D}^T = \mathbf{K}$.

We get a favorable representation, if we choose

$$\mathbf{D} = \begin{pmatrix} d_1 & d_2 \\ 0 & d_3 \end{pmatrix}$$

and store the three numbers d_1 , d_2 and d_3 , where d_1 and d_3 can be chosen to be nonnegative.

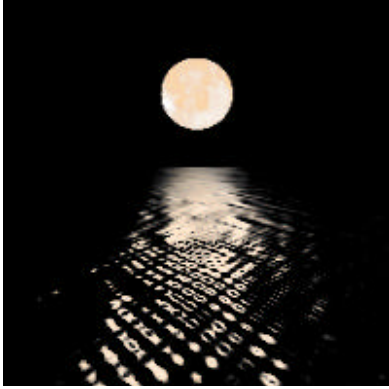


Fig 10: Moon over rectangle; waves are bump mapped; antialiasing with roughness pyramids. Background same as environment map.

If $\mathbf{K} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$, we get:

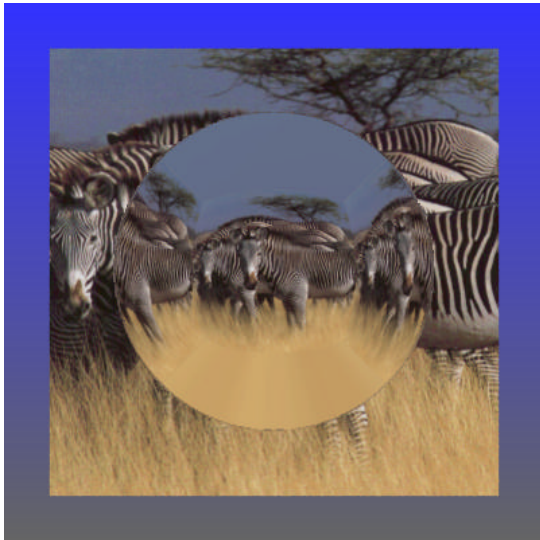


Fig. 12: Reflection of zebras, geometry: flat, bump map derived from Fig. 14; background: color ramp; environment map: Fig. 15; antialiasing with roughness pyramids.

$$d_1 = \sqrt{a - \frac{b^2}{c}},$$

$$d_2 = \frac{b}{\sqrt{c}}, \text{ and}$$

$$d_3 = \sqrt{c}.$$



Fig. 11: Same as Fig 10; no antialiasing

Isotropic roughness representation

A more compact roughness representation can be achieved, if we restrict ourselves to isotropic roughness information. Instead of three numbers, we only store one number: the standard deviation of the perturbation vector in the direction of the largest variance. We get this number by diagonalizing the covariance matrix

$$\mathbf{K} = \mathbf{R} \begin{pmatrix} I_1^2 & 0 \\ 0 & I_2^2 \end{pmatrix} \mathbf{R}^{-1}.$$

The value to be stored in the roughness map is I_1 with $I_1, I_2 \geq 0$ and $I_1 > I_2$. Thus we get a conservative approximation of the variance of the normal vectors within the considered region of the bump map. For the shading, one choice is again to use I_1 in a simpler version of the modified Blinn-Phong model (Appendix A). Another possibility is to use I_1 for the determination of the mipmap level when using environment maps with standard mipmapping, as standard mipmapping allows no anisotropic antialiasing anyway.

9. EXAMPLE IMAGES

The images shown on this page use the anisotropic roughness pyramid, introduced in this paper in conjunction with environment mapping.

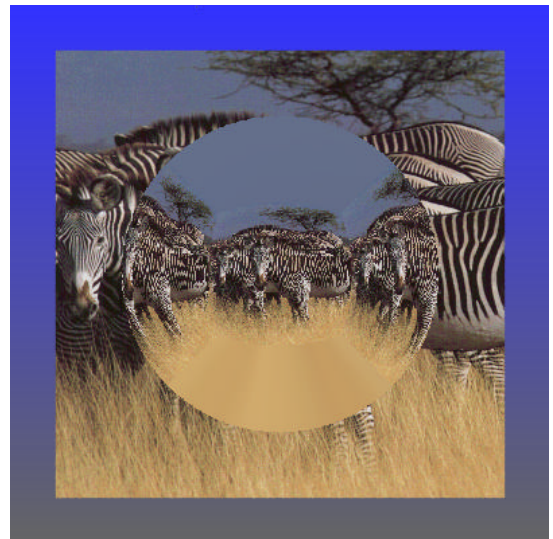


Fig. 13: Same as Fig. 12; no antialiasing

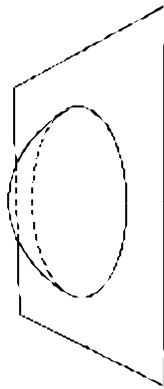


Fig. 14

As shown, roughness maps and roughness resolution pyramids are a way to properly antialias bump maps. After mapping color (texture maps) and bumps (bump maps), a new quality of surface properties can now be mapped to objects. Without roughness maps it is quite difficult to render real-looking images of a large class of objects with reasonable effort. Examples are small scratches on glossy surfaces or brushed surfaces but also every kind of bump mapped surface viewed from some distance. The expensive alternative to process such objects would be massive supersampling.

10. PROSPECTS

Fortunately, there are still enough unsolved problems that wait for simple solutions - simple enough to realize them in hardware. Some important problems, like shadows, have even been left out completely in the above treatise. This is good news for the researchers and developers of graphics hardware.

APPENDIX A: MODIFIED BLINN-PHONG MODEL

In this appendix, we are describing a modification to the Blinn-Phong shading model, that makes use of the roughness information stored in the roughness maps. We are using the Blinn-Phong model (luminance proportional to the dot product of surface normal \mathbf{n} and halfway vector \mathbf{h}) as opposed to the original Phong model (luminance proportional to the dot product of the reflected light source ray \mathbf{r} and the viewing ray \mathbf{v}). Often the two shading models are regarded as more or less equivalent or at least as having equivalent physical relevance. This is not the case; the Blinn-Phong model is an approximation of a micro-facet model, where the probability to see a facet with normal vector $\mathbf{n}_{micro} = \mathbf{h}$ (for such facets we get reflection of the light source into the eye) is proportional to $(\mathbf{n}^T \mathbf{h})^m$. This is much closer to a physical model than the original Phong model, where the angle between the



Fig. 15

reflected light ray and the eye vector serves as a measure for the probability to see the reflected light source. This makes no physical sense. In the literature, if a difference between the two methods is recognized, they are compared most often by comparing the form of a highlight on a sphere. For a human observer, the correct form of a highlight on a sphere is difficult to judge. If, however, the form of the highlight on a planar surface would be used, the difference would be obvious (see Fig. 16).

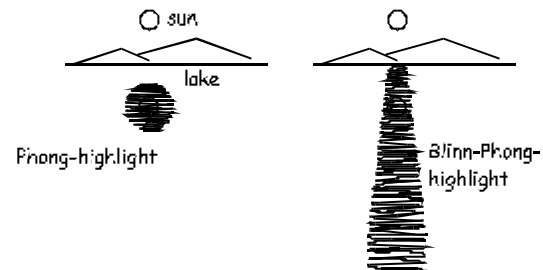


Fig. 16: Phong shading vs. Blinn-Phong shading (outline of PB-highlight plotted with Maple).

The modification of the algorithm consists of using a modified value for the dot product $\mathbf{n}^T \mathbf{h} = \cos(\mathbf{a})$ before the exponentiation depending on the direction of $\mathbf{n} - \mathbf{h}$. If $\mathbf{n} - \mathbf{h}$ points into the direction of greater variance of \mathbf{n} , the angle is reduced, if the variance of \mathbf{n} in this direction is smaller, it is increased. If \mathbf{d} is the projection of $\mathbf{n} - \mathbf{h}$ into the $\mathbf{e}_1 - \mathbf{e}_2$ -plane, we can modify $\cos(\mathbf{a}')$ instead of $\cos(\mathbf{a})$ (see Fig 17 for the geometry used) with:

$$\cos^2(\mathbf{a}') = \frac{\cos^2(\mathbf{a})}{|\mathbf{d}|^4 / |\mathbf{Dd}|^2 + \cos^2(\mathbf{a})},$$

where $\cos^2(\mathbf{a}) = 1 - |\mathbf{d}|^2$.

We get $\mathbf{a}' < \mathbf{a}$, if the variance of \mathbf{n} in the regarded direction is larger than 1, otherwise we get $\mathbf{a}' \geq \mathbf{a}$. If only isotropic roughness information characterized by I_1 is used, the formula can be simplified to:

$$\cos^2(\mathbf{a}') = \frac{I_1^2 \cos^2(\mathbf{a})}{1 - \cos^2(\mathbf{a}) + I_1^2 \cos^2(\mathbf{a})}$$

The size and the decay of the highlight towards its border can still be controlled with the exponent m . However, as the roughness is contained in \mathbf{D} (or I_1 resp.), it is possible to use a constant exponent m . The size of the highlight is then controlled entirely by the roughness. Another possibility is to use a fixed function $f(\cos^2(\mathbf{a}))$ with the desired behavior concerning the decay of the highlight towards its border to calculate the final brightness.

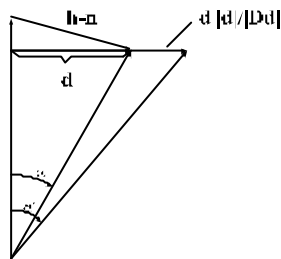


Fig 17: Modification of \mathbf{a} with the help of \mathbf{D}

ACKNOWLEDGEMENTS

I very much appreciated working together with Günter Knittel. Footprint-Assembly would not exist without him. I also want to thank Axel Schildan, who coded the algorithms. He used Ralf Sonntag's and Jens Hahn's RadioLab System. Thanks also to Wolfgang Straßer and my other colleagues at WSI/GRIS and especially to Reinhard Klein for fruitful discussions.

REFERENCES

- [1] Becker, B.G., Max, N.L., Smooth Transitions between Bump Rendering Algorithms, Proceedings of SIGGRAPH '93, Computer Graphics 27, Annual Conference Series, pp. 183-190.
- [2] Benrnbroek, K., Ernst, I., Rüsseler, H., Wittig, O., Design Principles of Hardware-based Phong Shading and Bump Mapping, in Proceedings of the 11th Eurographics Workshop on Graphics Hardware, Poitiers Aug. 1996.
- [3] Blinn, J., Models of Light Reflection for Computer Synthesized Pictures, SIGGRAPH 77, pp 192-198.
- [4] Blinn, J., Simulation of Wrinkled Surfaces, SIGGRAPH 78, pp 286-292.
- [5] Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F. The lumigraph. In *Computer Graphics Proceedings, Annual*

Conference Series, 1996 (ACM SIGGRAPH '96 Proceedings), pp. 43-54, 1996.

- [6] Kirk, D.B., The Simulation of Natural Features Using Cone Tracing, *The Visual Computer*, vol. 3, no. 2, (August 1987), pp. 63-71.
- [7] Klein, R. Linear approximation of trimmed surfaces. In R.R. Martin, editor, *The Mathematics Of Surfaces VI*, 1994.
- [8] Klein, R., Krämer, J. Building multiresolution models for fast interactive visualization, Proc. SCCG97, Bratislava, June 5-8, 1997.
- [9] Knittel, G., Schilling, A., Straßer, W. GRAMMY: High Performance Graphics Using Graphics Memories. In *High Performance Computing for Graphics and Visualisation*, M. Chen (ed.), London 1995, Springer.
- [10] Schaufler, G., Sturzlinger, W. A three-dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):C227-C235, C471-C472, September 1996.
- [11] Schilling, A., Knittel, G., Straßer, W., Texram: A Smart Memory for Texturing, *Computer Graphics & Applications*, May 1996, pp. 32-41.
- [12] Schilling, A. Antialiasing of Environment Maps. In preparation.