Visibility based methods and assessment for detail-recovery

Marco Tarini, Paolo Cignoni, Roberto Scopigno*
Istituto di Scienza e Tecnologie dell'Informazione - Consiglio Nazionale delle Ricerche†

ABSTRACT

In this paper we propose a new method for the creation of normal maps for recovering the detail on simplified meshes and a set of objective techniques to metrically evaluate the quality of different recovering techniques. The proposed techniques, that automatically produces a normal-map texture for a simple 3D model that "imitates" the high frequency detail originally present in a second, much higher resolution one, is based on the computation of per-texel visibility and self-occlusion information. This information is used to define a point-to-point correspondence between simplified and hires meshes. Moreover, we introduce a number of criteria for measuring the quality (visual or otherwise) of a given mapping method, and provide efficient algorithms to implement them. Lastly, we apply them to rate different mapping methods, including the widely used ones and the new one proposed here.

CR Categories: I.3.7 [Three-Dimensional Graphics and Realism]: Color, shading, shadowing, and texture—;

Keywords: simplification, texture mapping, detail recovery, normal mapping, texture for geometry

1 INTRODUCTION

A common solution to efficiently represent small scale geometric details on a surface is to use bump maps or normal maps. For years the use of bump maps has been a common practice in high quality, non-interactive renderings. Low-cost consumer graphics hardware that is able to efficiently perform the hardware accelerated rendering of surface with normal and texture maps (e.g. [Kilgard 2000]) has recently become common; therefore the use of this technique is becoming more and more common among interactive and realtime 3D applications like games [Blasco 2002], or in the visualization of complex object like the one obtained by range scanning [Bernardini et al. 2001].

There are two main techniques to produce normal maps: *artistic* and *automatic*. The first one is the classical artistic approach where a talented professional illustrator paints, by hand, the small scale reliefs of a surface in gray scale tones. Usually this bump map is then automatically converted in a normal map for efficient rendering. In the latter approach, the *automatic* way, we suppose that there exists, in some form, a digital *high resolution* representation of the object and some automatic tool that is able to convey as much information as possible from this model onto the surface of a low resolution model.

The first one, the *artistic* approach, is very well suited if we consider painting as a part of the modelling process, or if the small scale features represent mainly a kind of information that is more qualitative than precise and exact. For example this approach works well for drawing the scales of a monster or to add a scar on the face of a character.

The second one, the *automatic* approach, also known as *detail recovery* [Cohen et al. 1998; Krishnamurthy and Levoy 1996], has been introduced recently and is becoming more and more common for two reason: *a*) it allows to efficiently represent with just a small number of polygons objects that seems quite complex (Fig 7/8), *b*) high resolution version of the same object are often available for various reasons; e.g. in games high res models are used for the creation of prerendered introductive cutscene animations. In visualization scenarios often we can have data at a resolution much higher than we are able to interactively visualize, consider for example the interactive display of high resolution 3D scanning of Cultural Heritage objects.

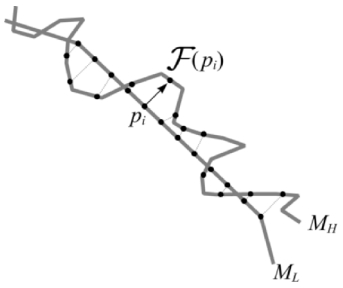

Figure 1: The concept of detail recovery: a low resolution mesh M_L is sampled and for each point p_i we search the *corresponding* point $\mathcal{F}(p_i)$ on the high res mesh M_H; using a texture, the detail found in $\mathcal{F}(p_i)$ is mapped onto p_i.

Detail Recovery The typical detail recovery phase (see Fig. 1) uses, as input, the following data:

- a complex high resolution 3D mesh M_H (e.g. the result of a 3D scan or of a modeler), provided with some implicit or explicit detail function $\mathcal{D}$ returning for each 3D point over M_H the "detail", in some form, to be recovered;
- a much simpler, low resolution mesh M_L (e.g. automatically obtained by simplification or created by hand by a talented low-poly modeler) with a good texture parameterization of its surface.

The function $\mathcal{D}(p)$ that defines the "detail" of M_H, depends on the application: it can return the normal direction of p on M_H, or a shading performed according to that normal in a fixed lighting conditions, or a color defined in M_H either per vertex or via textures, or even other things as a value for the accessibility of p in M_H, or a self-shadowing bit of p for M_H, or even a parametric color value defined by the position of p (e.g. for simulation of the wood

*e-mail:{tarini|cignoni|r.scopigno}@isti.cnr.it
†ISTI-CNR, Via Moruzzi 1, 56124 Pisa Italy

IEEE Visualization 2003,
October 19-24, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

appearance), or a combination of some of the above elements. It does not matter, as long as $\mathscr{D}$ is defined for any point $p \in M_H$ and returns values which can be stored in textures (the parameter p can be identified, for example, by a face pointer and two barycentric coordinates relative to that face).

The detail recovery phase constructs a texture T for M_L as follows:

for each face f of M_L,
  for each texel t of the texture space assigned to f,
    let $p \in M_L$ be the 3D point corresponding to t,
    choose a suitable 3D point $\mathscr{F}(p)$, in M_H
    store $\mathscr{D}(\mathscr{F}(p))$ in t

Note that we explicitly need a point-to-point mapping function $\mathscr{F}$ that gives, for each point of the low res mesh M_L, a corresponding point of the high resolution mesh M_H.

The mesh M_L, enriched with the ad-hoc texture T, is much more manageable and convenient in terms of rendering time and size with respect to the original mesh M_H, but still appears quite similar to it. The quality of the final result of this detail recovery process depends, at least, on three factors, some of which represent a compromise of cost/benefit: • the quality and the severity of the *simplification*: the smaller the M_L model the larger will be the difference from the original one M_H; • the quality of the texture *parametrization* and the size of the texture used to store the recovered details simplification; • the nature of the *mapping* $\mathscr{F}$, i.e. how we choose for each point of M_L the corresponding point of M_H.

The first two items have been subject of very intensive research and will not be discussed here further.

This paper focuses instead on the point-to-point mapping $\mathscr{F}$.

Paper Organization: First, we will present three different classes of approaches to define and perform the mapping $\mathscr{F}$ (Section 3). Two of them are already known, and one is a novel contribution of this paper. For these approaches, we also discuss some implementation issues, optimizations, and some problems to be solved in order to get a robust and efficient implementation.

In order to have a comparative analysis, we introduce in Section 4 a new set of measures and tools designed to rate the performance of a given mapping $\mathscr{F}$ used over a given model pair M_H, M_L.

Then we show in Section 5 comparative performance results of the various technique described, both visually and reporting the results of the above-described tools.

Novel contributions: The novel contributions of this paper are: first, the definition of an alternative way to perform the mapping $\mathscr{F}$ that uses the notion of average visibility direction, including some variants and details about how it can be efficiently implemented; second, the definition of the new set of automatic measures and tools to rate a performance of an given application of any mapping $\mathscr{F}$; lastly, the application of the latter to the former, and a comparison with other know mappings $\mathscr{F}$, that shows that our solution for $\mathscr{F}$ is indeed a valid one.

2 Related Work

A simplified model with a good texture parameterization on it is required to perform detail recovery. Much work has been done in the field of (semi-)automatic u-v mapping generation; an overview of the major contributions in this field can be found in [Desbrun et al. 2002]. We will just assume that a complete u-v mapping is given for the mesh M_L, seamlessly or not, providing per-wedge texture coordinate for each triangle in it, and therefore assigning some texture space for each face. Similarly, mesh simplification [Garland 1999] can be used to automatically obtain M_L from M_H instead of hand modeling it. We will not make any assumption of how we have obtained M_L but we will rely only on the fact that M_L and M_H are sufficiently similar and they share the same coordinate system.

Detail Recovery To our knowledge the first paper explicitly proposing this approach was [Krishnamurthy and Levoy 1996] where bump maps are applied to a nurbs model in order to catch the appearance of a high resolution scanned model. The first approach that was able to manage detail recovery for triangular meshes was presented by [Cohen et al. 1998]. In this paper the simplified model M_L must be obtained from M_H by using a constrained simplification algorithm that explicitly construct the mapping function $\mathscr{F}$ during the simplification process itself. As a drawback, this approach forces the adoption of a particular simplification algorithm, preventing a vast majority of cases where users want to use a different simplification or to model the simplified mesh directly (the standard way of creating the low-poly models used in gaming environments). The first general approach in which the detail recovery is explicitly de-coupled from the simplification strategy was proposed in by [Cignoni et al. 1998a]. This is more widely applicable; moreover the function $\mathscr{F}$, used only in the detail recovery phase, can be chosen freely.

In [Cignoni et al. 1998a] the use of both a proximity and ray-casting along normal strategies are discussed and the first one is proposed (see Sec. 3.1) for robustness reasons explained in [Cignoni et al. 1999]. A hybrid approach based on a ray-casting along normal strategy (see Sec. 3.2) coupled with nearest point selection in case of ray-miss (see Sec. 3.2) has been proposed in [Sander et al. 2000].

Given the wide diffusion of consumer graphics hardware able to perform normal map shading, some hardware producers are endorsing diffusing this kind of techniques in technical conferences [Maughan 2003] in order to encourage a better exploitation of hardware capabilities. The approaches presented there fall in the category of normal based ones (see Sec. 3.2 later).

Measuring Accuracy and Visibility Computation The idea of taking objective measurements for quality assessment for simplification algorithms for 3D meshes has been presented in [Cignoni et al. 1998b; Cignoni et al. 1998c] where the Hausdorff distance has been chosen to measure the difference between the original and the simplified meshes. More recently [Lindstrom and Turk 2000] a image-space measure has been proposed to measure the difference among M_L and M_H by comparing the rendered images of the two meshes from a small set of fixed viewpoints. This measure can drive the simplification process in order to obtain simplified models that are visually very similar. This approach has been extended in [Zhang and Turk 2002] by introducing the concept of visibility of a surface point as a scalar quantity measuring its accessibility from outside. This quantity can be used to weight the simplification process so that less visible parts are discarded early. Note that while these approach use visibility as a scalar quantity (the extent to which a given region is on average visible),in our case we are more interested in average visibility as a vector quantity (from which direction a given region is on average seen) and, to a smaller extent, visibility variance (the variance of that vector average).

3 Point-to-point correspondence strategies

Given a 3D point $p \in M_L$, we want to find the position $p' = \mathscr{F}(p)$, with $p' \in M_H$, so that we can fill texel t' corresponding to p with $\mathscr{D}(p')$. Which point shall we pick? In the following three subsections, we will discuss three different (categories of) strategies:

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

1. **proximity based**: pick the $p' \in M_H$ closest to p according to Euclidean distance;

2. **ray-casting along normal**: pick as p' the closest intersection between M_H and the ray starting from p and going along the normal direction $n(p)$ of p in M_L;

3. **visibility based**: pick as p' the closest intersection between M_H and the ray starting from p and going along the average visibility direction $v(p)$, in M_L, of p;

In a further subsection, we will also present and discuss an hybrid of the latter two.

3.1 Proximity based

Choosing the closest point has been used in [Cignoni et al. 1998c]. Proximity based approaches have the advantage of being very robust (they do not even require a normal to be defined over M_L). Moreover, $\mathscr{F}$ tend to be continuous, more than in other cases. A caveat is that when M_H presents two close surfaces, it can happen to pick a point of the wrong one. In practical cases, this can be partially helped, discarding any candidate points $p' \in M_H$ with a negative $n(p) \cdot n(p')$ dot product.

Implementation Issues Closest point research can be taxing. In our prototype (as in [Cignoni et al. 1998c]), all the faces of M_H are first spread into cells of a regular grid [Akman et al. 1989]: a pointer to that face is replicated in all the touched cells. To find the closest point of p, first we compute the distance from p to each face present in the cell to which p belongs, than we check neighbors cells at increasing distance only until the closest non-checked cell is further than the minimal distance found up to that point (usually less than 6 cells need to be checked). To minimize the number of triangle-to-point distance computations, we adopt an incremental marking scheme on faces, so that the same face is not checked twice in the same search.

3.2 Ray-casting along normal direction

Following the ray along the normal direction is another common approach (e.g. [Maughan 2003; Sander et al. 2000]). The rationale is that in mesh M_L point p will be well visible when seen orthogonally to the view direction, and therefore during the texture construction we should optimize for that case.

In order to avoid unneeded discontinuities in function $\mathscr{F}$ at the edges of M_L, the normal field $n(p)$ has to be continuous over $p \in M_L$; therefore $n(p)$ is found interpolating the normal vectors stored at vertices (a-la Phong shading), rather than using the normal of the face including p.

Managing ray misses Using ray-casting, $\mathscr{F}$ is not guaranteed to be defined over all M_L: ray casting (differently from closest point search) can "miss" M_H altogether. In some application, this does not represent a problem when occurs in proximity of borders of M_L: when $\mathscr{F}(p)$ fails, the texel t corresponding to p is filled with a blank (e.g. black or a transparent $\alpha = 0$ color value).

Unfortunately $\mathscr{F}$ can also fail far from any border (actually, even if M_L and M_H are closed, see Figure 2). A similar case is when the ray hits M_H, but very far from p, even tough a much closer point exists in M_H. More specifically, if the distance between p and $\mathscr{F}(p)$ exceeds an used defined upper limit $dist_{max}$, the ray cast is to be considered a miss as well.

Note that those cases can happen even if M_H and M_L are close in proximity of p, especially if p lies near convex edges of M_L.

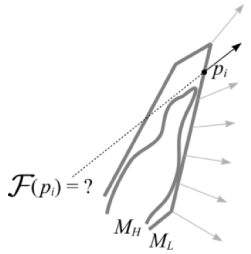

Figure 2: If $\mathscr{F}$ is implemented as a raycast, it can miss M_H.

When one of the cases above is detected for a point p, a good strategy is to switch to a proximity based approach for just that point (as proposed in [Sander et al. 2000]). This, however, can occasionally produce unneeded discontinuities when switching from a method to the other one. Another approach is to leave the affected texels blank and fill them in a second pass expanding values from neighbors texels.

Ray orientation Given a point p, $p' = \mathscr{F}(p)$ can be both below or above the face of M_L where p is (unless some assumptions are done on the simplification algorithm used to obtain M_L). Therefore, rather than a single ray, we would need to cast two rays, both starting from p and going toward $n(p)$ and $-n(p)$ respectively. If both rays hit a valid face of M_H, precedence should be given to the one going outward, to reflect the proper occlusion order.

Implementation Issues Again, sorting pointers to the faces of the input mesh M_H into cells greatly improves performance. Similarly to many other ray-casting approaches, we also need a grid traversal algorithm [Amanatides and Woo 1987] that identifies all the cell touched by the ray. In this way, it is also easy to stop the search as soon as threshold distance $dist_{max}$ has been traversed along the ray without hitting any valid face.

3.3 Visibility based

Using average visibility direction on M_L represents a novel contribution of this paper. The strategy looks promising: the expectance is that the average visibility direction $v(p)$ at point p is a good predictor to the specific viewing direction from which p will be seen at rendering time, reducing the visual difference between the rendering of the textured M_L and the original M_H.

Many considerations that are valid for the raycasting along normal approach, and in particular the problem of the misses, the ray orientation, and the implementation issues (sec. 3.2), are valid also for this approach.

In addition, new issues arise, concerning the definition of the visibility direction. In fact, we need an estimation $v(p)$ of the average visibility direction at a point p of the surface. That estimate can be computed *per face*, *per vertex* or *per texel*.

The averaged visibility direction is recovered in a way similar to [Zhang and Turk 2002]. We are looking for the average visibility direction

$$v(p) = \mathrm{Avg}_{p \text{ visible from } d_i}(d_i) \tag{1}$$

We perform a series of probe renderings of M_L as seen from a number n of directions $d_1..d_n$ well distributed over the normal sphere (a good value for n is around 2^9, see [Zhang and Turk 2002] for a more detailed analysis). The rendering are performed using

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

orthogonal projections (so that the viewing direction is kept constant for all pixels in the image), while the z-buffer takes naturally in account any self occlusion.

At each rendering, we get the video buffer and we record for each element whether and to which extent that element was visible from that direction, finding the average of all such directions.

Per face A intuitive strategy is to assign to each face of M_L an average visibility direction. To do this, we color each face with an unique color identifier, constant over the face. For each non-background pixel found in the rendering done under direction d_i, we add d_i to the (initially zeroed) average visibility directions of the face corresponding to that color. Note that since this is done for each pixel, the contribution of a probe view direction d_i to the averaged visibility direction of a particular face is proportional to the extent to which that face is visible from that direction.

As for the normal direction case, we need the directions v to be continuous over M_L. Therefore the visibility directions computed at faces are not used directly, but first averaged at vertices, and then interpolated again for each internal point (the per vertex is angle-weighted).

This strategy implies an averaging and a spreading of the computed visibility directions: since this seemed excessive, especially for meshes M_L with small (order of hundreds) of faces, we developed the two following additional strategies.

Per vertex Visibility directions can also be computed per vertex of M_L. This, however, cannot be done by directly recording, for each vertex, the directions for which that vertex is visible: since vertices in M_L are supposed to be sparse, many times they are not visible even if the adjacent faces are. A better algorithm consists in performing two renderings for each view direction: the first to identify the face (as before), the other to identify the position of the seen pixel inside that face. For the second rendering, we use the same color scheme for all faces: each wedge is color-coded differently (using pure colors), so that it is easy to reconstruct form the interpolated color the barycentric coordinates of the point relative to the belonging face. Each barycentric coordinate is used to weight the term d_i to the visibility direction computed for the respective vertices, so that a given pixel of a probe rendering affects close vertices more than distant ones.

In this way the direction field v will still be continuous (for each point $p \in M_L$ it is interpolated from values defined at vertices), but the artificial averaging and spreading of computed visibility vectors is sensibly reduced.

Per texel For an even finer grade estimation of the averaged visibility direction over M_L, we can also compute it relatively to each texel. In this case, the probe renderings are performed displaying M_L with the same texture coordinate as the final texture to be constructed, but color-coding differently each texel (this is possible in a single pass, since we have around $2^{20}..2^{24}$ different texels, for a typical 1024..4096 squared texture, and, including the α channel , 2^{32} different color values).

At each texel we sum together all the directions d_i under which that texel was spotted in the corresponding probe rendering.

Still, this generates three problems. First, some texels will not be visible from any point of view. Second, the resulting visibility directions will be far more noise plagued, as we are distributing the same number of samples (each non-empty pixel is a visibility sample) over a much wider set of buckets (the texels). Third, in correspondence of mesh edges (especially convex ones) discontinuities of visibility direction arise.

To solve all three problems, we apply, in a second pass, a smoothing filter to the resulting visibility direction field in texture space. The filter is actually applied *before* averaging: during visibility computation for each texel we record both the (not normalized) total vector sum v of all visible directions v and their number k. The smoothing filter consist simply in summing in each texel both v and k of neighbors texels. This way, void texel are filled, and averaged visibility directions computed from many samples are weighted more than the ones computed from a few ones.

A unresolved problem is that, for the filter to work, the texture must be seamless, or, else, a smart scheme of texel neighborhood must be adopted that takes in account the texture coordinate scheme (to assign to border texels "neighbors" that are such only in object space, rather than in texture space).

3.4 Hybrid using variance

Comparing results coming from the application of the last two strategies (raycasting along normal or along visibility direction), it emerged that in some locations the former performed better, in other ones the latter. Therefore an hybrid approach could be introduced: since they are both ray-casting based approach, to hybridize them it is just a matter of using, as a ray direction for point p, an interpolation of normalized vectors $n(p)$ and $v(p)$.

To find the interpolation weight we use the standard deviation $\sigma(p)$ of the set of directions from which p is visible. In fact, if the variance of visibility $\sigma(p)$ is low, it means that point p on M_L, whenever it is visible, it is such from a predictable point of view; in this case it makes perfect sense to use that direction for texture reconstruction purposes. On the contrary, high variance $\sigma(p)^2$ signals that point p will be seen from many different directions, therefore the average visibility direction $v(p)$ will be, on one hand, less significant, and on the other, its use will improve the rendering of M_L at point p only in a few cases, making more appealing to resort to the normal directions.

Implementation issues The computation of the standard deviation of the set of visibility can be performed almost for free. In fact, $\sigma(p)^2$ can be defined as follows:

$$\sigma(p)^2 = \mathrm{Avg}_{p \text{ visible from } d_i}(d_i^2) - \mathrm{Avg}_{p \text{ visible from } d_i}(d_i)^2 \qquad (2)$$

The first average equals to 1 (as d_i are normalized), and the second element is just $v(i)^2$. Therefore:

$$\sigma(p) = \sqrt{1 - v(p)^2} \qquad (3)$$

4 MEASURING TOOLS

In the previous section we listed several different approaches to perform the mapping $\mathscr{F}$ from points in M_L to the ones in M_H. Which method is to be preferred?

One natural way to answer the question is to visually compare the results. We show a comparison in the Results section. However it helps to rate the results also with some measurable and less subjective way. For this reason we developed a small set of automatic measuring tools aimed at that purpose, which we present in this section.

Namely, three tools measure the result of a given application of $\mathscr{F}$ under three different consideration:

- "object space" distance, a view depended measure of the discrepancy between the textured M_L and M_H;
- "image space" distance, a view depended measure of discrepancy between renderings of M_L and M_H, or equivalently a measure of the self coherence of the textured M_L (this measure can be either normalized or not, see below);

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

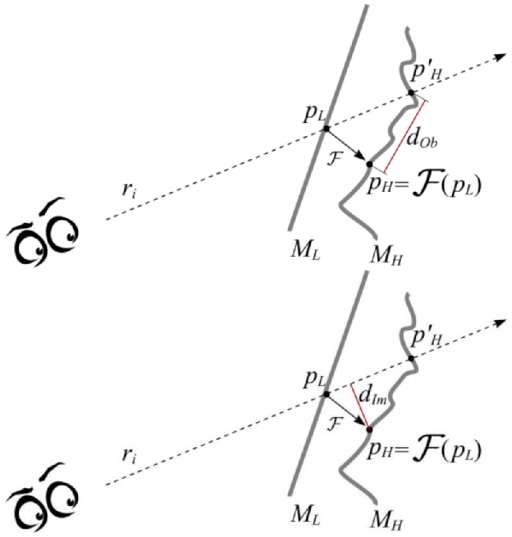

Figure 3: A 2D example of the Image space (above) and Object space (below) distance measure along a specific view ray.

- the uniformity of the resulting sampling.

For all the three notions of quality we are able not only to find an overall quality value for a model with a texture resulting from a particular choice of $\mathcal{F}$, but also to plot a distribution of that value over the surface of M_L (see Figure 6).

4.1 View dependent *object space* distance

This measurement (and the next one) is a view dependent one: the total quality estimation will be a sum of contributions relative to the conditions recorded from different point of views. Therefore we choose a number (around 512) of well distributed point of views around M_L to be used for measurement purposes.

Consider a textured rendering of M_L (see Figure 3). Along each viewing ray r_i, the rendering will show (either the background or) a point p_L of M_L; thanks to the texture, that point will present the feature originally present in point $p_H = \mathcal{F}(p_L)$ of M_H. If M_H was rendered instead of M_L under the same viewing condition, the same ray r_i would hit in general a different point p'_H.

The visual difference along a given ray (that is, at a given pixel of the rendered image) will be therefore the difference between the renderings of details associated in M_H to the two different points p_H and p'_H, that is $\mathcal{D}(p_H)$ and $\mathcal{D}(p'_H)$. For sake of generality, we want to decouple our measure of $\mathcal{F}$ from any particular $\mathcal{D}$ associated to M_H; therefore we just use the Euclidean distance between p_H and p'_H (which for values smaller than topological features size of the mesh is a good approximation of the geodesic distance); that quantity, squared and integrated over all rays and the sampled view directions, gives a measure of how good $\mathcal{F}$ is. Note that the measure is view dependent, because such is the correlation between p_L and p'_H.

In the cases when ray r_i misses M_H or M_L, then of course the problem is not in the texture but in the geometry alone, and therefore that ray is to be ignored.

Implementation This measurement algorithm needs to test over a great quantity of rays (in our prototype, order of 10^8 rays). Luckily the implementation will take advantage of graphical acceleration.

To implement the measurement tool, we first of all need to build a probe texture for M_L, in the standard way, using the $\mathcal{F}$ we want to test. As for $\mathcal{D}$, as we have seen we don't want to use any particular detail function but rather, for generality sake, a simple mapping from $\mathbb{R}^3$ to color space (using as boundaries the bounding box of M_L and M_H), so that $\mathcal{D}$ differentiate every point of M_H.

Then, it will be enough to compare, for each view direction we want to test, pixel per pixel, a rendering of M_L texture with the probe texture, to a rendering of M_H, colored per vertex, assigning to each vertex v_i the color $\mathcal{D}(v_i)$. Naturally both renderings will be done using the same point of view. In both renderings care must be taken to disable lighting, anti-aliasing, mipmapping and anything that would affect the rendered colors. Also, for non-closed models, back oriented faces must be displayed with background color, so that they will be present as occlusors but otherwise ignored by the algorithm. Then, for each pair of non-background pixels at the same position in the two rendering, we just apply $\mathcal{D}^{-1}$ to both and find the Euclidean distance between the two results. The View Dependent Object Space distance is found by averaging this distance over all pixels of all renderings.

Since the color space, with 8 bit per components, has not enough resolution and would introduce quantization noise in the measurement, we resorted to two consecutive renderings instead of one, and two probe textures, so that we can store the 3D position in 64 bits instead of 32.

To avoid an interference when testing a $\mathcal{F}$ that uses visibility direction (see Sec. 3.3), the set of random viewing directions we use to test $\mathcal{F}$ is a different one to the one used in the definition of $\mathcal{F}$ to find the average visibility direction.

Since each pixel of the rendering contributes to the total error, this method automatically gives more weight to the parts of M_L that are more visible, whether they are so because of (lack of) occluders or because of orthogonality with the view direction.

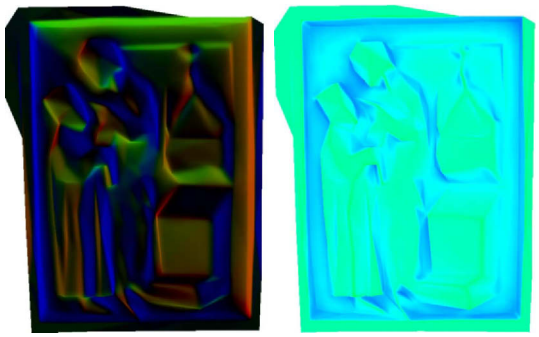

Figure 4: A color coding of the per-texel average visibility direction (on the left, vector pointing left, right and down are colored red, blue and green respectively) and the visibility variance (on the right, mapped from dark blue — lower, to dark green — highest). The original dataset visible in Figure 7.

4.2 View dependent *image space* distance

The mesh M_L is just an approximation of M_H; moreover, it does not represent a strictly self-coherent 3D mesh. In fact during the renderings, a point p_i on its surface is displaced on the screen according to its *real* position, but is colored (and, in case of bumpmapping, shaded) according to how it would be if it was in another, different

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

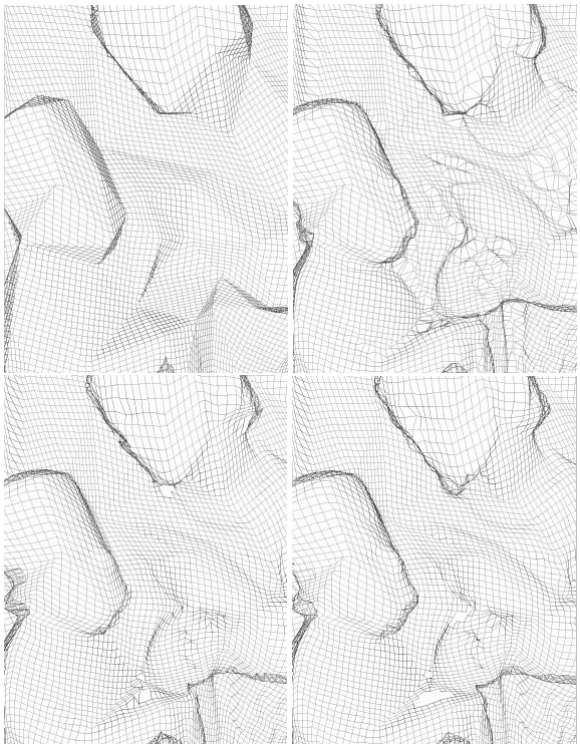

Figure 5: Top-left: the sample distribution T over M_L (each junction is a point on M_L represented by a texel — for clarity, we used a small 256×256 texture). Then, clockwise: the distribution of $\mathscr{F}(T)$, with $\mathscr{F}$ chosen as proximity based, and normal based and hybrid. Note how irregular the proximity based case is.

position $\mathscr{F}(p_i)$. The discrepancy of the two (in image space) depends much on the choice of $\mathscr{F}$ and can be measured. Note that this is a measure of the textured M_L alone, not strictly a comparison with M_H. The mesh M_H affects the results only because the image of $\mathscr{F}$ is bounded to lie in M_H.

More precisely, we can take a large set of test viewing rays, and, for each ray r_i that hits M_L in p_L we measure how far the point $\mathscr{F}(p_L)$ is from the ray r_i (see Fig. 3).

Note that this is *not* the same as measuring the average distance between p_L and $\mathscr{F}(p_L)$: the vector $(p_L - \mathscr{F}(p_L))$ can be long in module but have its effects partially cancelled by its parallelism with the view direction.

Implementation The implementation is a variant of the one done for the object space measurement, and uses the same probe textures. This time, we render only M_L, with the probe textures as before, seen under orthogonal projections. For each not-background pixel c with value c_{rgb} at window position (c_x, c_y), we recover the position $p' = \mathscr{F}(p)$ by $p' = \mathscr{D}^{-1}(c_{rgb})$; to find the ray-to-point distance, (since the projection is orthogonal) we compute the distance between (c_x, c_y) and the window position of p' projected under the current view transformation.

4.2.1 Problems with Image Space distances

The Image Space distance is probably the more intuitive error measure, because it is a direct measure of pixel-by-pixel distance between a rendering of the real geometry and the one "simulated" by the texture. Still, as it is, that measure proves trickier than it seems

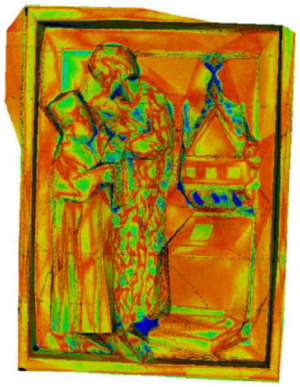

Figure 6: Error distribution: each part of M_L (bumpmapped with the visibility based methods) is colored according to its contribution to the Image Space Distance, from the lowest (red) to the highest (blue).

(see Fig. 1): being Pixel-per-Pixel, it fails to penalize discontinuities in the rendering; moreover, in practice it is basically influenced solely by the average distance between p and $\mathscr{F}(p)$ (shorter distances rating better) which is not by itself a valid criteria, as we will show shortly (if it was, than the Proximity based $\mathscr{F}$ would be a very good choice, which is not, see Fig. 1).

To solve the latter problem, a good countermeasure is to normalize the image space distance dividing it at each ray by the distance between p and $\mathscr{F}(p)$.

To solve the former problem (failure to penalize discontinuities) we can apply separately the next criteria.

4.3 Sampling Uniformity

The texture we are going to produce for M_L will represent, in a sense, a sampling over M_H. Therefore we can rate $\mathscr{F}$ analyzing this sampling, for example by measuring how well distributed it turns out to be (or rather, how closely the resulting sampling matches the one implicitly specified by texture coordinates). IEEE Visualization

Let $T \subset M_L$ be set of 3D points that are represented by a texture sample, we want to measure how well distributed is $\mathscr{F}(T) \subset M_H$. The sampling uniformity of T itself depends on the quality of the u-v mapping of M_L, which is outside our scopes. Therefore, we will measure how much "worsened" is the resulting sampling of $\mathscr{F}(T)$ in respect to T.

More precisely we use, as a comparative measure, the average of percentage increase/decrease of the the Euclidean distance of elements of $\mathscr{F}(T)$ from their neighbors, in respect to the corresponding ones in T (considering two elements of $\mathscr{F}(T)$ to be neighbors when the respective texels are adjacent). Of course we average absolute percentages, meaning that we consider a 20% decrease the sampling distance to be as bad as a 20% increase (being the number of samples equal).

5 Results and Discussion

In order to test the different strategies of $\mathscr{F}$, we implemented and applied them on mesh pairs M_H M_L. The results have been compared with the tools introduced in Section 4, and also by simply looking at the rendering of the resulting textured version of M_L.

Test-bed In our case, meshes M_H comes from automatic acquisition: the "high-relief" dataset (250k faces), a laser scan of a high

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

choice of mapping $\mathscr{F}$	Obj.Sp. dist.	Im.Sp. dist.	normalized Im.Sp.dist.	Sampling Unif.
	High-relief: (250K faces → 500 faces)			
Prox. Based	0.265	**0.83**	8.27	19.6%
Normal Based	**0.243**	0.98	7.91	**4.6%**
Visibility Per Vertex	0.248	1.14	7.75	9.8%
Per Texel	0.248	1.19	**7.69**	10.2%
Hybrid Per Vertex	0.255	1.06	7.86	7.0%
Per Texel	0.256	1.10	7.71	6.8%
	Michelangelo: 400K faces → 2K faces			
Prox. Based	1.41	**0.78**	10.46	24.0%
Normal Based	1.37	0.80	10.31	**12.1%**
Visibility Per Vertex	**1.30**	0.83	**10.21**	14.0%
Hybrid Per Vertex	1.32	0.81	10.25	12.9%

Table 1: A comparison of the various error measures (as described in Sec. 4) obtained using a variety of possible mapping $\mathscr{F}$ (as described in Sec. 3) over the high-relief and the Michelangelo dataset. In each column, the position of the best performer is highlighted. Per Texel strategies could not be tested over the second dataset because it lacked a seamless u-v mapping.

relief marble sculpture, and the a Michelangelo model (400k faces).

The original datasets were simplified to a thousands faces by an automatic quadric error based simplification software to create the low resolution meshes M_L. We also needed a texture parameterization for M_L: we used a trivial mapping of each triangle in a separate squared isosceles triangle (including some borders to mask the discontinuities), or, when we needed a seamless texture (see Section 3.3, Per Texel) we used the an ad hoc seamless parameterization for the high-relief model. In any case, in order to minimize the impact of the quality of texture parameterization, we used high resolution 2048 squared textures. For the renderings aimed at a subjective comparison we used, as detail function $\mathscr{D}(p)$, the normal of p in M_H, thus obtaining a normal map to be rendered with standard GPU hardware [Kilgard 2000].

Results Figures 7 and 8 shows some rendering using different choices of mapping $\mathscr{F}$, while Table 1 list the numerical values resulting from the tools as described in Sec. 4.

Discussion All in all, both measurements and subjective visual comparisons show that the Proximity based methods perform on average bad (except for the Images Space distance, for the reasons explained in Section 4.2.1), especially for their effect on the sampling uniformity. Still, they represent the more robust approach, especially if no holes are acceptable in the texture (for example, the black spots in the last three images of Figure 7 are "ray that missed"). This, as mentioned, makes them the ideal fall-back strategy, to apply locally when other ones fail (as was suggested in [Sander et al. 2000]).

Normal Based and Visibility based strategies tend to perform similarly, each prevailing in some circumstance and under some criteria. They have an opposite visual effect: sometimes they give a perception of "faked" geometry that looks respectively flatter, or bumpier, than it should. Note that for most parts of most meshes the two strategies do not differ too much (in absence of occlusions, the average visibility direction of a face is also its normal).

It is easy to see the reason behind the equivalence of the performances: both ray-casting approaches obviously give best results when the face of M_L rendered is seen from a view direction similar to the direction of the ray-cast, that is, the normal or the average visibility direction (respectively). The normal based approach is advantaged by the maximized screen area that is covered by the face

Figure 7: The high-relief dataset: a visual comparison of the results obtained with the different mapping techniques. Top: the original M_H (25K faces) and the simplified version M_L (500 faces), shown flat shaded, and bump-mapped with a bumpmap created using visibility based $\mathscr{F}$ function. Just below, a close up of the original and simplified mesh. Then: comparative renderings of simplified models bumpmapped using proximity based, normal based, (third row) visibility based per texel, and hybrid based mappings $\mathscr{F}$ (last row).

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

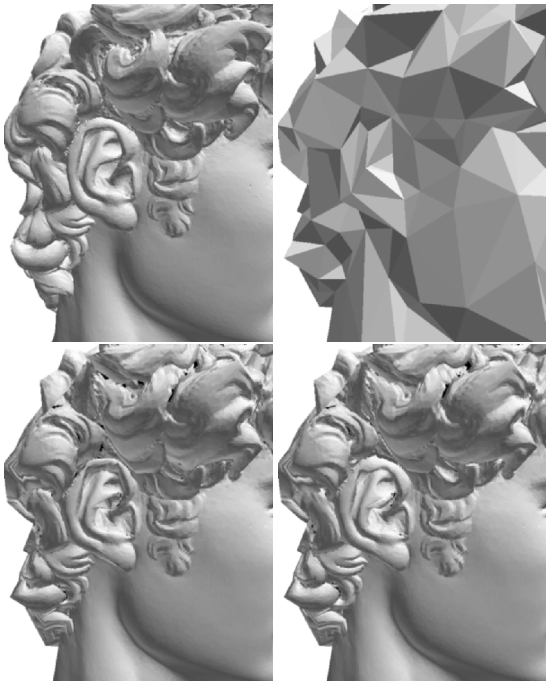

Figure 8: Another example: a portion of the Michelangelo David; above: original model, simplified model flat shaded. Below, the re-detailed bumpmapped model obtained using the Normal approach, and then the Visibility approach. Note that the ear is a complex 3D structure, but from many directions is occluded by the hairs: the visibility based approach takes advantage of this by optimizing the bump-map for the remaining views.

in that case; the visibility based approach has instead the advantage that the face is often occluded in the non optimal cases.

The Hybrid approach (using variance) performs somewhat in between the two, but never rates much worse than either, possibly making it the best overall candidate. Also, it tends to produce renderings more visually similar to the original model.

As a last note, the Visibility direction (whether used pure or in conjunction with the normal) sometimes gives better results when computed per texel than when computed per vertex. However, to compute it per texel requires a seamless u-v mapping for M_L (or at least a smarter texel neighborhood scheme capable of "jumping over" texture boundaries). Both ways are better than the Visibility computed per Face (and then averaged at vertices), whose results, for conciseness, are not reported in the tables.

6 Extensions and Future Work

The work we presented here can be extended in many directions.

In our work, the set of non occluded view direction for a given point are just averaged together; instead, when the variance of that set is high, a clustering could be attempted first: if the non-occluded view directions are groupable in two or more separate clusters, it makes sense to produce a different texel following the centroid of each group, ultimately resulting in alternative textures for the same triangle, similarly to the view dependent textures proposed in [Debevec et al. 1996]. In our case, the separability of the clusters would guarantee that transitions directions are never actually seen, making it possible to switch abruptly between alternative textures, thus avoiding any ghosting-plagued texel-value interpolation.

As an easy improvement, the average viewing direction could be a better predictor of rendering time view-direction if we weight the contribution of each non-occluded view direction according to its application-depended likelihood: for example, an human figure will mostly be seen from horizonal view directions.

We also believe that variations of this work can be adapted to improve results of similar problems, rather than just texture based detail recovery: the tool that measures the sampling uniformity can be also used to rate a given u-v mapping. *Surface simplification*, when used in conjunction with detail recovery, can benefit from considerations coming from the analysis of the error distributions returned by the measuring tools presented here: for example, convex regions tend to be error plagued more than line concave regions, suggesting that simplification can be done to a larger extent in the former.

Acknowledgments: to the anonymous reviewers for their suggestions, and to EU IST Project "ViHAP3D" for fundings.

References

Akman, V., Franklin, W., Kankanhalli, M., and Narayanaswami, C. 1989. Geometric computing and uniform grid technique. *Computer-Aided Design 21*, 7 (Sept.), 410–420.

Amanatides, J., and Woo, A. 1987. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, 3–10.

Bernardini, F., Martin, I., and Rushmeier, H. 2001. High-quality texture reconstruction from multiple scans. *IEEE Transactions on Visualization and Computer Graphics 7*, 4, 318–332.

Blasco, O. 2002. Curvature simulation using normal maps. In *Game Programming Gems III*, Charles River Media, D. Treglia, Ed., 433–443.

Cignoni, P., Montani, C., Rocchini, C., and Scopigno, R. 1998. A general method for recovering attribute values on simplifed meshes. In *IEEE Visualization '98*, IEEE Press, 59–66.

Cignoni, P., Montani, C., and Scopigno, R. 1998. A comparison of mesh simplification algorithms. *Computers And Graphics 22*, 1, 37–54.

Cignoni, P., Rocchini, C., and Scopigno, R. 1998. Metro: measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (June), 167–174.

Cignoni, P., Montani, C., Rocchini, C., Scopigno, R., and Tarini, M. 1999. Preserving attribute values on simplified meshes by re-sampling detail textures. *The Visual Computer 15*, 10, 519–539. (preliminary results in IEEE VIS 98 Proc.).

Cohen, J., Olano, M., and Manocha, D. 1998. Appearance-preserving simplification. In *SIGGRAPH 98 Conf. Proc.*, Addison Wesley, M. Cohen, Ed., Annual Conf. Series, ACM SIGGRAPH, 115–122. ISBN 0-89791-999-8.

Debevec, P., Taylor, C., and Malik, J. 1996. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 96 Conf. Proc.*, Addison Wesley, H. Rushmeier, Ed., Annual Conf. Series, ACM SIGGRAPH, 11–20.

Desbrun, M., Meyer, M., and Alliez, P. 2002. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum (EG'02 Proc.) 21*, 2.

Garland, M. 1999. Multiresolution modeling: Survey & future opportunities. In *EUROGRAPHICS'99, State of the Art Report (STAR)*. Eurographics Association, Aire-la-Ville (CH).

Kilgard, M. J. 2000. Pratical and robust bump mapping technique for today's GPU's. In *Advanced Opengl Game development*.

Krishnamurthy, V., and Levoy, M. 1996. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96 Conf. Proc.*, Addison Wesley, H. Rushmeier, Ed., Annual Conf. Series, 313–324.

Lindstrom, P., and Turk, G. 2000. Image-driven simplification. *ACM Transactions on Graphics 19*, 3, 204–241.

Maughan, C. 2003. All the polygons you can eat. In *Game Developer Conf. 2003*.

Sander, P. V., Gu, X., Gortler, S. J., Hoppe, H., and Snyder, J. 2000. Silhouette clipping. In *SIGGRAPH 2000, Computer Graphics Proc.*, Addison Wesley, Annual Conf. Series, 327–334.

Zhang, E., and Turk, G. 2002. Visibility-guided simplification. In *Proc. of the 13th IEEE Visualization 2002 Conf. (VIS-02)*, IEEE Computer Society, Piscataway, NJ, R. Moorhead, M. Gross, and K. I. Joy, Eds., 267–274.

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE