The Visual Computer

Preserving attribute values on simplified meshes by resampling detail textures

P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, M. Tarini

Istituto di Elaborazione dell'Informazione – Consiglio Nazionale delle Ricerche, Via S. Maria 46, 56126 Pisa, Italy
e-mail: {cignoni,montani,rocchini}@iei.pi.cnr.it, r.scopigno@cnuce.cnr.it, tarini@graphiris.iei.pi.cnr.it

Many sophisticated solutions have been proposed to reduce the geometric complexity of 3D meshes. A slightly less studied problem is how to preserve attribute detail on simplified meshes (e.g., color, high-frequency shape details, scalar fields, etc.). We present a general approach that is completely independent of the simplification technique adopted to reduce the mesh size. We use resampled textures (rgb, bump, displacement or shade maps) to decouple attribute detail representation from geometry simplification. The original contribution is that preservation is performed after simplification by building a set of triangular texture patches that are then packed into a single texture map. This general solution can be applied to the output of any topology-preserving simplification code and it allows any attribute value defined on the high-resolution mesh to be recovered. Moreover, decoupling shape simplification from detail preservation (and encoding the latter with texture maps) leads to high simplification rates and highly efficient rendering. We also describe an alternative application: the conversion of 3D models with 3D procedural textures (which generally force the use of software renderers) into standard 3D models with 2D bitmap textures.

Key words: Surface simplification – Detail preservation – Texture mapping – Picture/image generation – Display algorithms

Correspondence to: P. Cignoni

1 Introduction

Mesh simplification technology has evolved substantially over the last few years, and many approaches have been proposed for the controlled simplification of simplicial meshes. Only a few papers are cited here [5, 12, 16, 30]; comprehensive overviews can be found in [15, 25]. Most of the methods proposed offer no immediate provision to control accurately the perceptual effect of degradation because in many cases the simplification criterion adopted has no immediate interpretation in terms of *visual degradation* [27]. Perceivable visual degradation may be caused either in visualizing a single simplified representation (e.g., in the case of an excessively simplified mesh, with loss of topology features and/or fuzziness/bumpiness of the simplified surface), or in changing the level of detail, the so-called interframe flickering that is common if the meshes in a level of detail (LOD) representation present large visual differences.

Defining a methodology for measuring visual degradation is not an easy task. Driving simplification by preserving curvature and sharp edges gives good control over the appearance of the shape, one reason being that most renderers draw elementary components by shading colors according to surface normals [18, 27]. However, taking into account only the pure geometric approximation is not sufficient to ensure that the visual accuracy required is fulfilled. Pictorial information (color or texture) is an important factor in perception, and therefore preservation of color discontinuity has to be managed carefully. This important issue has been taken into account in a number of recent proposals [4, 10, 11, 13, 16, 18, 20, 24, 27, 31], as summarized in Sect. 2. Although pictorial information is probably the most common attribute, it is not the only one. Another type of mesh attribute is the sampling of a field over the mesh vertices (e.g., the sampled value of physical variables, such as temperature, potential or pressure). In order to use these field values in visualization (e.g., by mapping field values to color or by computing isolines), a simplification code should take into account the value of the field while reducing the complexity of the mesh [16, 29].

A new, general approach is proposed in this paper to preserve attribute information on simplified meshes built on dense, triangular meshes, and it extends the preliminary results presented in [6]. The approach we propose is orthogonal to previous solutions, which take into account attribute values during the simplification process. Conversely, we propose

The Visual Computer (1999) 15:519–539

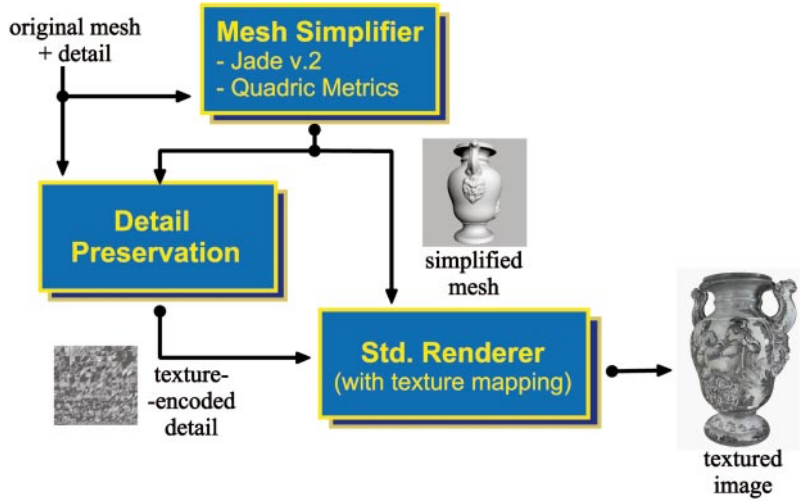


Fig. 1. The overall framework of our detail-preserving approach

a simple solution for preserving attribute information that is independent of the simplification process run on the input mesh (Fig. 1). Attribute preservation is conceived here as a postprocessing phase that can be used in conjunction with various simplification codes. A sampling process is applied to the simplified mesh; i.e., each face is scan-converted into object space. For each sampling point p_S, we find the corresponding point p_M on the original mesh and the original attribute value in p_M. These retrieved attribute values are then stored in a texture map, which is used at rendering time to paint the pictorial detail of mesh M onto mesh S. All of the triangular texture patches, computed by sampling the faces of S, are stored efficiently in a single rectangular texture map. The quality of the texture produced obviously depends on the sampling resolution adopted and the accuracy in determining the (p_S, p_M) pairs. The latter point depends mainly on the *similarity* between the original mesh and the simplified one (i.e., the more dissimilar the shapes, the higher the sampling error may be).

This approach has a number of advantages. It can be used with any simplifier that supports high-quality simplification (if possible, topology preserving) because no assumptions on the simplification approach are made. It can restore every type of mesh attribute (examples on the restoration of color, other scalar/vectorial fields and high-frequency surface perturbations are shown at the end of the paper). It is a geometrically robust technique since it works on nonmanifold or degenerate meshes. Its time complexity depends mainly on the simplified surface area, which is measured in elementary sampling point units. Finally, analogously to other approaches based on textures [4, 19, 20, 31], modeling high-frequency detail with texture maps allows highly efficient rendering on modern graphics subsystems, which generally support hardware texture mapping.

Obviously, it also has some disadvantages. Since we encode detail through textures, the space occupancy of the simplified mesh is increased by the need to store the corresponding texture map. Attribute-aware simplification approaches might thus produce a more compact output on meshes that do not present a very complex attribute field (see next section).

The approach proposed may also be used to solve another problem; that is, how to convert an object description that uses 3D *procedural textures* [9] into a format that only supports image-based textures.

The paper is organized as follows. Section 2 outlines previous research in this area. Our approach is described in detail in Sect. 3. Section 4 shows how the proposed approach can be extended to manage procedural textures. The extension of the proposed method to multiresolution meshes is described

in Sect. 5. An evaluation of the results obtained is presented in Sect. 6. Finally, conclusions are drawn in Sect. 7.

2 Previous work

Attribute preservation is a very critical issue in mesh simplification, with a direct impact on the actual use of the simplified meshes. The approaches proposed in the literature can be classified as *simplification-integrated*, which adopt some attribute-preserving strategy embedded in the simplification code, or *simplification-independent*, as is the case of our proposal.

Simplification-integrated approaches can be classified as follows:

- *Attribute-aware simplification*. These methods evaluate the impact of each atomic simplification action on detail degradation. A threshold on attribute degradation can then be introduced (and a given atomic action is performed only if the corresponding detail degradation is lower than the given threshold). Alternatively, a multivariate error evaluation function, which takes into account not only the shape approximation but also the attributes degradation [10, 11, 13, 16, 18] can be adopted. More detailed, multivariate error evaluation has been introduced in the *progressive meshes* [16] and the *quadric error metrics* methods [10, 13, 18]. The latter approach works in R^{3+m} space (with m relative to the attributes space considered, e.g., $m = 3$ if we consider only the color), and adopts a distance-to-hyperplane metric. An improvement to this approach has recently been proposed [18]. It adopts a more intuitive measure of the attribute error based on geometric correspondence. It produces more accurate attribute-aware simplifications and is more efficient.
  A slightly different solution is based on the characterization of the topology of the attribute field; simplification is then constrained to preserve the topology of the given attribute field [2].
  All these solutions return in output the same type of data read in input (a mesh with attributes).
- *Texture-enhanced simplification*. These approaches decouple shape and attribute representation and build a texture map to encode all the detail lost in the simplification (either starting from the removed vertices [20, 31] or by directly managing available texture-based encoding of the attribute to be preserved [7, 17, 19]). All these solutions return in output a simplified mesh and a detail texture.

A disadvantage of all attribute-aware simplification approaches is that the simplification of the mesh depends jointly on the shape and the attribute field. This implies that attribute discontinuity may prevent many decimation/collapsing actions that would be valid if we only considered geometry/topology degradation. When the input mesh has a very complex attribute field associated with it, it is often impossible to produce a highly simplified model.

On the other hand, an advantage of these methods is that the simplified mesh is very compact and efficient in all those cases where the attribute field is either smooth or presents a small number of discontinuities. In these cases, piecewise linear functionals (shaded triangles) with detail mapped via vertex attributes may be more space efficient than texture-enhanced triangle meshes.

A disadvantage of the simplification-integrated methods that adopt a texture-enhanced approach is that maintaining trace during simplification of all the removed elements (and of the corresponding attribute values) is rather costly both in memory and in time. However, if for each simplified mesh face there is complete information on the original mesh section it is replacing, then texture construction may be more robust than the method proposed here (see Sect. 3).

Most of the solutions consider only the color attribute. High-frequency shape detail preservation is considered in [19], which adopts B-splines surfaces to represent concisely dense irregular polygon meshes and maps high-frequency shape detail through displacement maps (for each spline mesh section, a displacement map represents the error between the fitted surface and the original polygon mesh section).

A completely different approach to surface simplification and multiresolution representation is the one based on wavelet decomposition. An attribute-aware strategy has been proposed in [14]; both shape and color of range images are represented, at multiple resolutions, using nonorthogonal Gabor wavelets. Decoupled multiresolution representation of geometry and color via wavelets has also been proposed [4]. In this approach, geometry and color are combined only at display time to allow the dynamic selection of independent levels of approximation for geometry and color. The authors also represent color

detail through texture mapping. Textures are built dynamically, at the required resolution, by painting the color wavelet coefficients on the texture map.
Finally, let us consider the interactive visualization of complex scenes. Texture-mapped simplified models are commonly used, either as *impostors* of complex scene subsections [26] or as replacements of single objects (e.g., using the LOD approach). The adoption of simplified polygonal meshes, enriched by the representation of mesh detail through standard texture mapping, is justified by the large diffusion of hardware texture-mapping capabilities in most graphics systems (from game stations to graphics workstations) and by the forthcoming inclusion of bump-mapping features in graphics subsystems [21, 22].

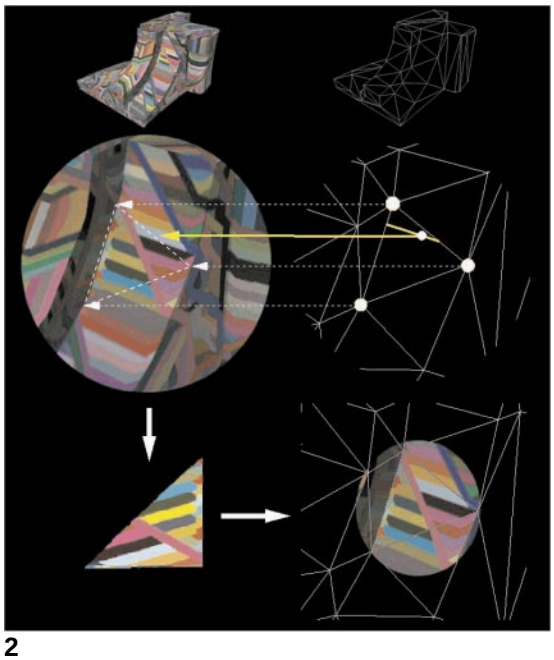
3 Preserving detail on simplified meshes

In general terms, we are interested in recovering onto the [simplified] mesh S the value of scalar/vectorial attributes F defined on mesh M. We assume that the attribute value can be computed for each point $p \in M$ by means of a function $f : M \subset R^3 \rightarrow R/R^3$. Function f can either be a continuous function in R^3, or a function defined piecewise on the cells of M (e.g., by interpolating a discrete set of samples of F evaluated on the vertices of mesh M). We also assume that meshes M and S are defined in the same geometric space and have the same topology.
We start from the pair (M, S) and produce a texture T that encodes the detail contained in M. Texture T will then be used to render mesh S, using standard rendering tools for texture-mapped graphics, to produce images nearly identical to those produced using the original mesh M.
Our approach is composed by the following two phases.

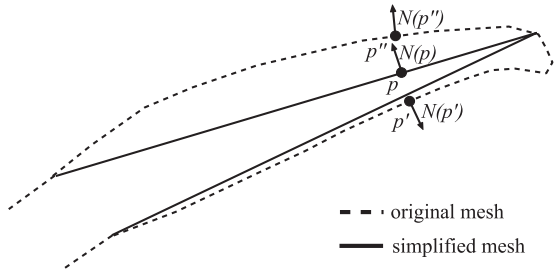
- *Surface sampling.* While discretely sampling each face of the surface S, we detect for each sampling point on S the corresponding point on M and the associated attribute value. Therefore, for each face we produce a discrete, triangular texture patch that encodes the detail mapped onto the corresponding area of mesh M. The size of this texture patch directly depends on the size of the triangular face and on the sampling step size chosen by the user;
- *Texture patches packing.* All triangular patches are packed (and stored) in a standard, rectangular texture, compatible with standard graphics libraries (e.g., OpenGL) and HW subsystems.

3.1 Sampling texture patches

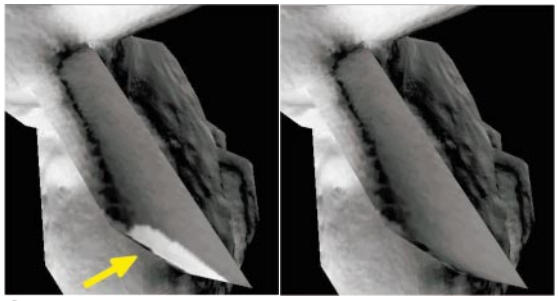
The surface of the simplified mesh S is sampled by scan-converting triangular faces under a user-selected sampling resolution. For each elementary surface sample p_S we detect the corresponding point $p_M \in M$ by computing the point at a minimal euclidean distance from p_S (Fig. 2). In this way, no knowledge of the simplification approach adopted to build the reduced mesh is needed to find corresponding point pairs.
For each sampling point p_S, the nearest point on the original mesh M is computed efficiently by evaluating *point-to-face* distances. Moreover, the computation of the minimal distance between point p and all the faces of M is optimized (otherwise, complexity will become quadratic). We adopted a bucket representation of the faces of M [1]: a regular grid of cubes is built, covering the bounding box of M. Pointers to the faces contained/intersected are stored in each bucket. For each point p, we test faces of M in order of distance from p, stopping the process as soon as no more buckets closer than the nearest face exist. The regular grid structure is defined according to the mesh size (the number of cells is proportional to the number of faces in the mesh) and to the bounding box edges ratio.
Detecting corresponding point pairs by searching for the nearest face is a valid choice if the two surfaces M and S have the same topology and the geometric error introduced during the simplification is not high. In fact, the main goal of our approach is to produce very accurate simplified models out of original complex meshes, both in terms of geometry and attribute detail (and not to build projective texture maps to be stitched onto a very simple solid impostor, e.g., a simple cube).
In some cases choosing the nearest face may produce aliasing. An example can be the case of a simplified mesh that contains subareas with a very thin solid section (see the bunny's ear in Figs. 3 and 4). In this case, given a sampling point p_S on S, the nearest point p_M on M might be on a face of M that lies on the other side of the mesh (Fig. 3). This problem can be detected and corrected, in some cases,



2



3



4

Fig. 2. Mapping color detail on a simplified mesh by sampling color and constructing an rgb texture
Fig. 3. Incorrect selection of matching-point pairs
Fig. 4. An example of how a highly evident aliasing effect can be avoided by checking the compatible normals orientation

by accepting as the nearest point only a point on a face whose normal has an orientation compatible with that of point p (i.e., a positive dot product of the two normal vectors), such as point p'' in Fig. 3. Figure 4 shows a section of the bunny's head (top view) with matching points incorrectly selected on the left and correctly selected on the right (produced by checking normals while searching for corresponding pairs). Unfortunately, we cannot ensure that this simple heuristic will always lead to the correct choice; for example, consider the case of multiple folding/twisting of the original mesh M in an area that corresponds to a simple section of the simplified mesh S. However, this is probably a very uncommon case, because we assume that we are starting from a simplified model that should represent the original mesh with some accuracy.

Another example is the case of a hand of a virtual actor, where we might have a detailed model and a very simplified model that consists of only a couple of pentagonal faces. There are areas of the simplified mesh that should not be connected to an original mesh area, but to the void space (for example, the void space between each pair of adjacent fingers). Using our approach, all the fingers in the simplified model will be inflated, and the void space between fingers will disappear in the rough model. Again, in this case we are trying to produce an impostor rather than a good simplified model, and therefore a standard projective technique is more adequate.

Once we have found a pair of corresponding points p_S and p_M, the computation performed depends on the particular attribute value we want to preserve:

- *Pictorial data, or other scalar/vectorial values*. For each sampling point p_S we retrieve the scalar/vectorial value $f(p_M)$ of the associated point on mesh M. The value computed by f can be: the color of M in p_M, either interpolated from the colors of the vertices of the face f that contains p_M or interpolated on the rgb texture associated with f; the interpolated value of a scalar field defined on the vertices of f (e.g., a temperature or pressure field); etc.

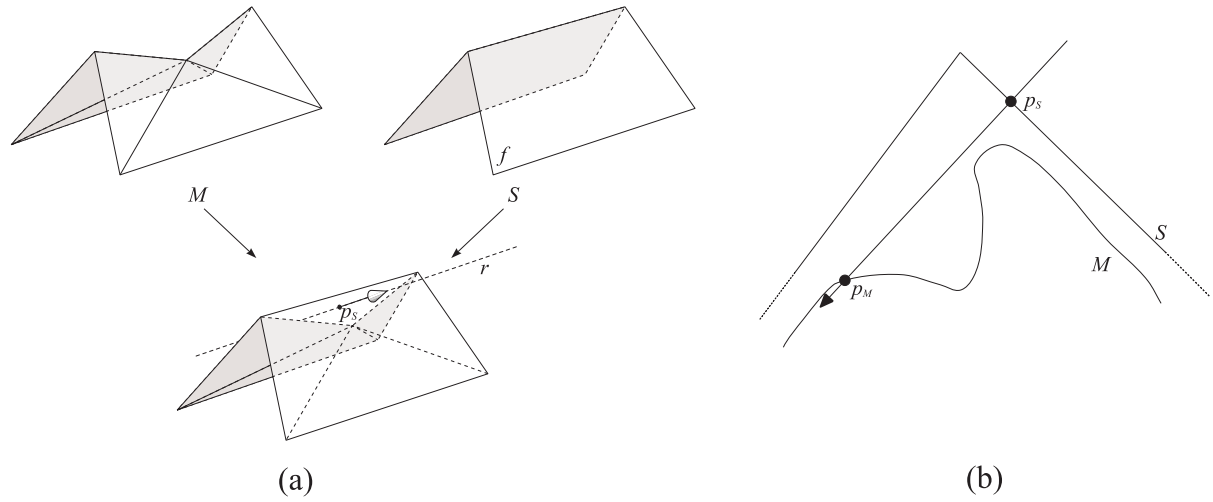


Fig. 5a,b. The ray r passing through the sampling point p_S on the simplified surface S and perpendicular to the face f of S does not pierce any point of the original surface M (**a**), or pierces a very far point, which is not the best choice for the current p_M to be paired with p_S (**b**)

- *Shape detail*. The distance between points p_S and p_M can be used to define displacement maps [8, 32], normals evaluated in p_S and p_M can be used to build bump maps (sampled normal perturbations) [3] or preshaded maps.

In more detail, *displacement*, *bump*, or *preshaded* maps can be produced to perform shape-detail preservation. Storing shape detail into texture is a very powerful resource to increase image quality without increasing geometric complexity, and graphics subsystems with hardware bump-mapping features are on the way [22].

Computing a *displacement map* from evaluated surface distances is straightforward. However, these distances give only an approximation of a real displacement map because displacements are generally evaluated on the direction of the normal to the sampled surface. However, computing *correct* distances along the surface's normal direction r is not easy because in some cases there may be no intersection between the original mesh M and the ray r (see Fig. 5a) or because the nearest intersection along r might be on a section of the mesh that is much farther away than the section nearest to the sampling point p_S (Fig. 5b).

In our implementation we adopted minimal distances, and the line that passes through the corresponding pair of points may not be aligned with the normal in S. Our point-to-face distance evaluation takes into account whether the original mesh is above or below the surface of the simplified mesh, and therefore we obviously store signed distances in the texture. A number of experiments show that even though we produced an approximation of the displacement map, its visual quality was good.

To produce a *bump map*, we must evaluate the normal perturbation on each sampled point. Analogously to what we do to evaluate the color, for each sampling point p_S we get the normal $N(p_M)$ on point p_M by interpolating normals on M. Once we have the sampled normal $N(p_M)$, we store in the bump map the normal interpolated on the simplified mesh S on point p, $N(p_S)$ (interpolation of per-vertex defined normals).

The third case is that of *preshaded maps*, which can be used to simulate bump mapping on any architecture that provides rgbα texture mapping either in HW or in SW (e.g., OpenGL). In this case, we evaluate preshaded gray levels, taking into account for each sampling point the normal on M and a given lighting model (direction of the light, ambient factor). Then, at rendering time we will simply use this gray-level texture to modify the mesh base color, e.g., defined per vertex or via an rgb texture. All images with pre-

Table 1. All data (mesh resolution, sampling step, times, space overhead, texture sizes) evaluated for the four sample meshes

Mesh	Fandisk	Face	Bunny	Plane
Original (Number of faces)	12 946	10 000	69 451	87 002
Simplified (Number of faces)	100	1745	251	799
Detail preserved	Color (per-vertex)	Color (from texture)	Shape	Scalar field (per-vertex)
Detail destination	Color map	Color map	Normal map	Color map
Sampling step used	0.5%	0.8%	0.5%	0.2%
Times (in seconds)				
Initialization	0.971	0.838	4.820	7.059
Texture packing	0.003	1.273	0.008	0.024
Sampling	2.769	3.096	13.339	18.990
Total (no I/O)	3.743	5.207	18.2777	25.767
Total (with I/O)	6.277	8.833	30.683	90.534
Space overhead – regular packing				
Total overhead:	373%	452%	316%	775%
Texture size:	1024×192	1024×64	1024×144	1024×512
Space overhead – irregular packing				
Patch_exp	+17%	+62%	+12%	+11%
Patch_borders	+23%	+203%	+39%	+49%
Text_gaps	+9%	+0%	+4%	+10%
Text_tails	+31%	+1%	+12%	+23%
Total overhead:	83%	267%	67%	93%
Texture size:	594×128	681×64	462×128	903×128

served shape detail were rendered using preshaded texture, using OpenGL (Figs. 16, 17).

3.2 Choosing the right sampling step

The choice of the size of the sampling step is crucial. In fact, it should be chosen such that significant detail is not lost during texture resampling, and at the same time the texture produced is of minimal size. This is because both the visual quality of the result and the cost in terms of space of the output, which are both direct consequences of the length of the sampling step, cannot easily be predicted.

To assist the user, one possible approach is to let the user select the approximate size (T_u, T_v) of the output texture map and then compute the sampling step *ss* automatically. One possible heuristic is the formula:

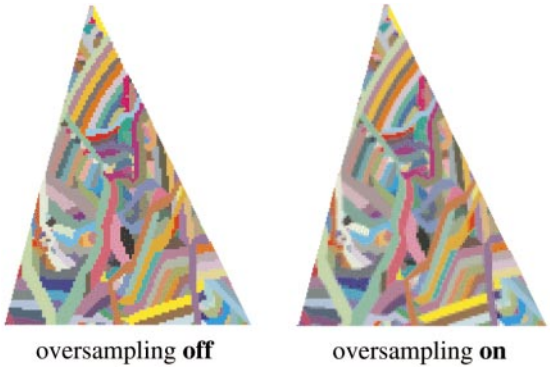
$$ss = P^{-1}\left[\sqrt{\left(\frac{A_S^P * p_{ovh}}{T_u * T_v}\right)}\right],$$

where A_S^P is the surface area of the simplified mesh S measured in view-space pixels (that is, under a typical projection transformation P such that the bounding box of the projected object approximately fits into a standard window); p_{ovh} is the empirical overhead factor of the texture resampling and packing method, see for example the values presented in Table 1; and P^{-1} is the inverse of the projection transformation.

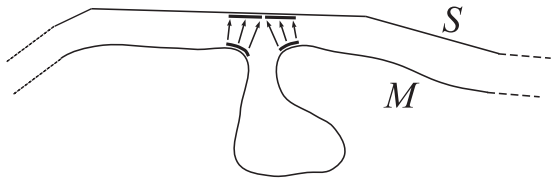
3.3 Improving texture quality

The quality of the texture produced is crucial, especially for applications where the goal is to produce a simplified textured model that has to be very similar to the original one. The tradeoff between the quality of the texture produced and the space/time complexity of the approach proposed will depend on a number of parameters or implementation choices.

Insufficient sampling rate (blocky textures). This problem depends on the sampling step chosen by the user, which may be too coarse on particular areas of the mesh. This problem is evident in the case of surface patches where attribute values change very abruptly (e.g., a face painted with a number of



6



7

Fig. 6. Oversampling improves the quality of the texture produced
Fig. 7. A possible case where mesh detail is undersampled due to the selection of corresponding-point pairs based on minimal distances

colored stripes). This problem cannot be solved by simply decreasing the sampling step size because the size of the texture cannot be enlarged too much (the texture memory available on graphics subsystems is often limited).
Our tool adopts *oversampling* to improve texture quality, and the oversampling rate is a user-selected parameter. Multiple samples are distributed regularly in the small sampling area associated with each point p and evaluated. The average of these samples is stored in the texture (see Fig. 6).

Undersampled faces. This is the inverse of the preceeding problem. Given a sampling step, the mesh may contain triangles smaller than the single squared sampling step. However, aliasing may be introduced if a given face is represented by only a single texel. One possible solution (implemented in our system) is to sample each face with at least a minimal number of sampling points (e.g., setting a minimal size of 10 texels).

Aliasing on the adjacency border between different textured faces. This problem is related to the fact that, once packed (see next paragraph), texture patches that are adjacent in the overall texture map are generally associated with nonadjacent faces of the mesh. During the scan conversion of a mesh face at rendering time, discrete texture coordinates might be produced that are not contained in the associated texture patch (this might occur frequently while scan-converting the edges of a face due to insufficient numerical precision). To prevent erroneous color mappings due to imprecise computations of texture coordinates, we produce, for each mesh face, a texture patch that can be slightly wider (depending on the edge slope). This solution leads to some texture space overheads (see Sect. 6), but solves this aliasing problem.

Redundant texture. In some cases, faces may show a nearly bilinear variation of the preserved attribute. This might happen on meshes with nearly constant color or detail values, or where only a few discontinuity values are defined and the attribute value changes linearly. In all these cases, simple per-vertex coding of detail may be sufficient and obviously much more space-efficient than storing detail through texture patches.
To reduce texture size, a hybrid approach can be implemented: the user can define an *interpolation-accuracy* threshold. At sampling time, for each face the system will decide whether the recovered detail has to be coded by explicitly storing the texture patch or by producing only per-vertex attribute values. The output in this case would be a subset of faces with texture coordinates (and a texture map), plus a subset of faces with direct per-vertex attribute values.

Under/over-sampled features in resampling. In general, the adoption of a minimal euclidean point-to-face distance and the use of a speed-up data structure

(the uniform grid) ensures the efficiency of the sampling method and prevents the singularities shown in Fig. 5. However, in some particular cases some mesh details might be under-/over-sampled. A possible aliasing case is depicted in Fig. 7, where the original mesh M presents a sort of small cavity that has been completely washed out in the simplified mesh S. In this case, for all the points of S that ideally correspond to this cavity in M, the minimal point-to-face distance criterion returns points on the border of the cavity (the points on the bold line in the figure). Recovering from this type of aliasing is not easy, and further research is needed.

3.4 Packing texture patches into rectangular textures

The last step of our algorithm is to pack all the triangular texture patches into a single rectangular 2D texture. Various solutions can be designed.

A first approach, *regular packing*, may be based on the use of half-square texture patches, which are very simple to pack into a rectangular texture [20, 31]. The use of only half-square texture patches, with a further limitation to square edges of size 2^i in [31], allows them to be packed easily: patches are stored in order of magnitude (bigger first), equal size patches are paired to form squares, and squares are stored in adjacent texture areas (see Fig. 8).

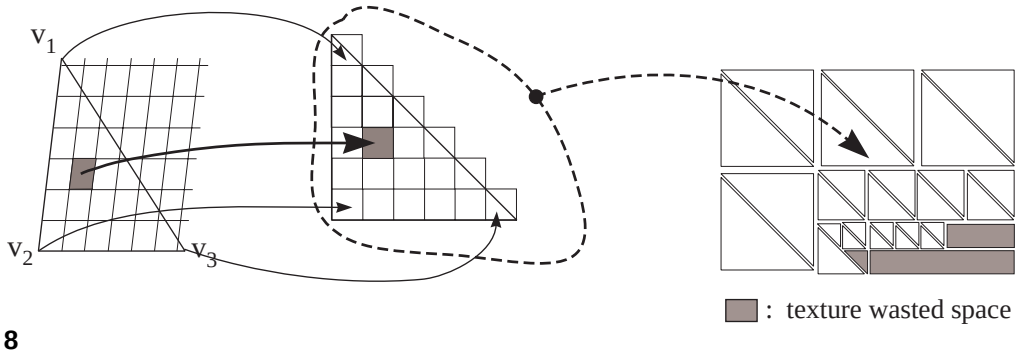
The use of half-squared texture patches allows space-efficient packing, but this approach has two disadvantages: only a discrete number of patch sizes is available and the shape of the texture patch is fixed. In the case of the sampling-based approach proposed in this paper, the first point implies that we do not use exactly the sampling step s selected by the user: given the sampled edge e, we divide it into 2^k chunks such that the $\frac{length(e)}{2^k} \leq s \leq \frac{length(e)}{2^{k+1}}$. The second limitation implies that the elementary sampling area is a rectangular area (not a square) in object space (see Fig. 8). The texture patch therefore only approximately takes into account the actual size of the associated mesh triangle, and does not consider its actual shape at all. Very thin faces are sampled with very different sampling steps in the two dimensions, and this introduces uneven sampling. The actual sampling size selected by the user is an upper bound of the sampling size that is effectively used, since for very elongated faces we actually use a much finer sampling step on the shorter edge.

To avoid these disadvantages, we designed a different packing technique that allows the use of *non-rectilinear* texture patches. Let us call this *irregular packing*. We maintain the limitation of using only a discrete set of patch heights (all texture patches have a height equal to 2^i), but we use the same sampling step in the two dimensions. For each mesh face, we are free to rotate it to select the best matching height. The criterion is to select the "rotation" of the face such that, once we have enlarged its height to the nearest 2^i discrete size, the corresponding discrete surface will be the minimal one of the three possible rotations. Irregular texture triangles are therefore generated, but because they still have only a discrete number of different heights, we can store them compactly.

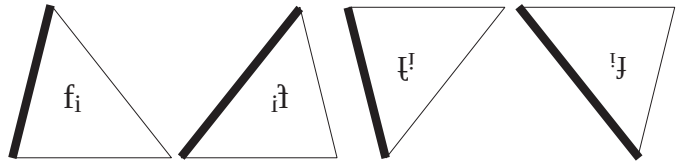
Patches are divided into different buckets, one for each different patch height (2^i texels). Buckets are processed in order of height (tallest first). Initially, the first open edge is aligned to the left border of the empty texture, and is equal in size to the tallest bucket. We then proceed by iteratively selecting among all the patches in the current bucket the one whose slope is the most similar to the current open edge (see Fig. 9). To allow compact packing, four possible slopes are assigned to each face, obtained by flipping the patch on the two axes. This is because we can flip the texture patch before copying it in the final texture. At each step, given the left-most open edge, we scan the list of patches to find the one with the best matching slope. Some gaps may be originated in the texture (see Fig. 10), and we call this wasted texture space *text_gaps*.

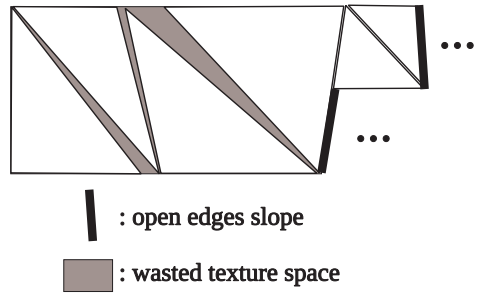
We also tried a slightly more complex packing strategy based on the idea of shearing each texture patch along the x-axis, in ordre to reduce text_gaps. The example in Fig. 10 shows the borderline case, where once we have found the best matching patch we also shear it until its slope is identical to the open edge one. Large shearing may introduce some aliasing, while the use of small shearing angles is sufficiently safe in terms of visual quality. The maximum angle of shearing is one of the parameters of our implementation (the more we shear, the less texture space is wasted, but the more aliasing is introduced).

Packing strategies evaluation. We implemented and evaluated both packing approaches. Time complexity is not a critical issue because texture packing time is really short (a few milliseconds in the case of the slightly more complex irregular patches packing, see

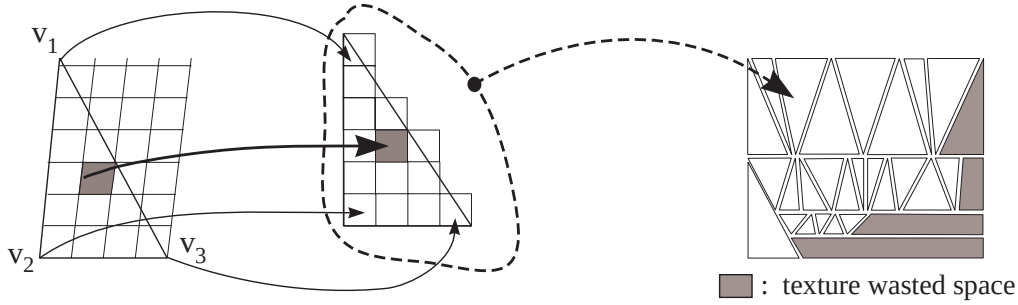


8





9



10

Fig. 8. Half-square texture patches are very easy to pack into a rectangular texture, but due to the irregular-to-regular mapping a different sampling step is used in the two axes

Fig. 9. Irregular patches packing: each patch has four possible slopes (patch flipping); given an open edge slope, we detect and attach the best-matching patch

Fig. 10. Irregular texture patches are easy to pack into a rectangular texture and allow even sampling on the two axes

Table 2. Processing times (in seconds) of the three subphases on the fandisk mesh (100 faces). Different attributes are preserved: rgb Color; shape detail encoded either via normal maps (Shape1) or displacement maps (Shape2); and composed color and shape detail (Both1 and Both2)

Detail preserved	Color	Shape1	Both1	Shape2	Both2	Color	Shape1	Both1	Shape2	Both2
Sampling step	0.5%(High resolution)					1.%(Low resolution)				
Texture size	594×128					341×64				
Times (in seconds)										
Initial	0.967	0.945	0.943	0.946	0.946	0.976	0.947	0.943	0.949	0.943
Sampling	2.782	2.992	3.038	2.892	5.986	0.982	1.042	1.082	1.014	2.107
Text. packing	0.002	0.002	0.002	0.002	0.003	0.002	0.002	0.003	0.002	0.003
Total (no I/O)	3.751	3.939	3.983	3.840	6.935	1.963	1.991	2.028	1.965	3.053
Total (with I/O)	6.357	6.500	6.547	6.380	9.501	4.489	4.498	4.522	4.457	5.536

Table 3. Results obtained using various sampling steps and multisampling values

Sampling step	0.2%	0.5%	1.%	2.%
Times (in seconds, no I/O)				
Initial	0.971	0.971	0.973	0.972
Sampling – *no multisampling*	14.718	2.769	0.977	0.452
Sampling – *multisampling* 2×2	58.438	10.470	3.456	1.558
Sampling – *multisampling* 3×3	128.441	22.520	7.279	3.205
Texture packing	0.002	0.003	0.002	0.002
Space overhead				
Patch_exp	+20%	+17%	+15%	+15%
Patch_borders	+10%	+26%	+48%	+95%
Text_gaps	+12%	+9%	+11%	+5%
Text_tails	+47%	+31%	+37%	+16%
Total overhead	89%	83%	111%	131%
Total texture size	962×512	594×128	341×64	184×32

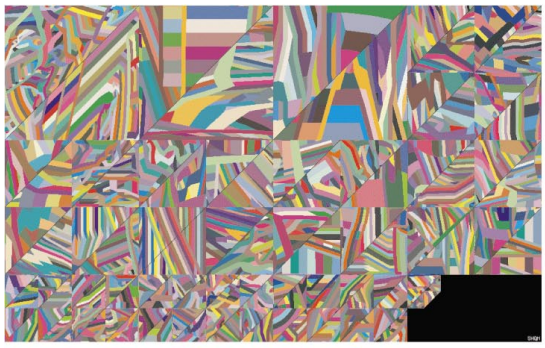
Tables 1–3). Conversely, texture space complexity is an important factor. Let us define the ideal texture size as the mesh surface area divided by the squared sampling step. Texture space is a critical resource on most HW graphic subsystems; therefore, the lower the overhead (e.g., the difference between the ideal and the sampled texture), the better the quality (because, given a target size, we can increase the sampling rate and obtain a more detailed texture).

On average, on a number of experiments our irregular packing leads to textures which are 60%–120% larger than the ideal size, while regular packing returns textures 300%–600% larger than the ideal one. To compare in more detail the respective texture sizes (and overheads) we must introduce some terminology.

- *Patch_exp* is the patch expansion factor due to the adoption of discrete 2^i heights (or heights and widths, in the case of regular packing).
- *Patch_borders* are the patch expansion factor due to the use of a slightly wider sampling (to prevent aliasing on the border between different textured patches); this factor increases when we have a large number of small faces.
- *Text_gaps* are the void spaces between pairs of adjacent patches left by the irregular packing strategy; this factor is reduced when the number of faces is large (and is generally lower than 10%).
- *Text_tails* are the void spaces at the end of each horizontal run of patches in the detail texture; again, this factor is reduced when the number of faces is large.

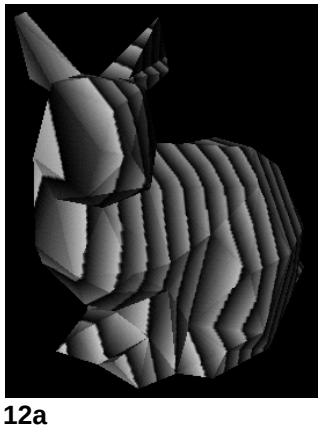
All these components were evaluated as a percentage of the ideal texture size. The space overheads were evaluated on four sample meshes; the results are reported in Table 1.

Given the irregular packing strategy, we also compared the detail texture size obtained when shearing

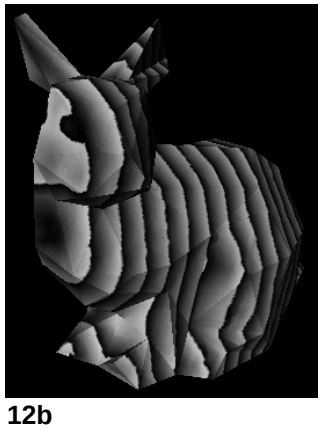


11a

11b



12a



12b

Fig. 11a,b. Texture maps produced with the regular patches (**a**) and with irregular patches (**b**) on the fandisk mesh (see Fig. 14)

Fig. 12a,b. Mapping a stripelike 3D procedural texture on a highly simplified bunny model (251 faces). For each sampling point p on the simplified mesh, the 3D procedural texture is evaluated directly on p_S (**a**), or on the corresponding point p_M on the original high-resolution bunny mesh (**b**)

or when not shearing the texture patches. In a number of tests the text_gaps overhead was less than 10% with the no-shearing mode, and reduced to 1%–2% when adopting patch shearing. However, because a 10% text_gaps overhead is not particularly high, and because shearing may introduce some aliasing, we generally prefer to use the no-shearing mode.

An example of the textures obtained with regular and irregular packing approaches is shown in Fig. 11, relative to a very simplified model (100 faces) of the *fandisk* mesh (12 946 faces). The sampling step size in this case is 0.25% of the mesh bounding box diagonal. The size of the texture maps obtained is 1024×640 in the case of regular patches, and 978×256 for our irregular packing method.

4 Extension to procedural textures

Static procedural textures are a very powerful tool for the simulation of the realistic appearance of materials (e.g., marble or wood), and allow us to circumvent the image mapping problem that characterizes 2D textures [9, 32]. The texture value can be

computed in every 3D location using either fractal noise basis functions [23] or functions based on the computation of distances from a set of random feature points [28].
Procedural textures can easily be mapped onto a free-form object, but are generally rendered by adopting software-only rendering approaches (e.g., ray tracing). Many applications use procedural textures, and one problem is how to preserve detail specification when standard scene description languages (OpenGL, OpenInventor, or VRML) based on planar 2D textures are adopted, e.g., to ensure rendering efficiency or portability. An example is the conversion of a complex model created with a commercial tool that supports procedural 3D textures (e.g., 3DStudio Max) into a format oriented to interactive CG (such as VRML or OpenGL).
The reconstruction of a standard texture that encodes the intersection between a given procedural 3D texture and a surface can easily be performed with our approach. Sampling the procedural texture space at rendering time is replaced by an off-line joint sampling of the given mesh and of the texture space. During off-line mesh sampling we only need to compute, for each sampling point, the associated procedural texture value, and then to store it in the texture patch associated with the current face. Patches are then packed as usual into rectangular 2D texture maps. An example of a 3D procedural texture encoded through a 2D standard rgb texture is given in Fig. 19.
Clearly, we may get very different results if we compute procedural texture values on the simplified mesh sampling points, or if we compute texture values on the corresponding points on the original surface (see Fig. 12 for an example). The "wooden" bunny in Fig. 13 was obtained by sampling a very simple bunny model and resampling a wooden procedural texture on the corresponding points of the original bunny mesh (the image shows the composition of the bump and the color texture).

5 LOD Management

A well-known problem in texture mapping is that texture rendering produces significant aliasing when we apply a fixed resolution texture to areas of the objects that are a long distance from the observer, and therefore map to only a few pixels (texture undersampling). In that case, only a few texels contribute to the image, and the detail structure is generally corrupted. The *mip-map* approach [33] was proposed to solve this problem by storing a hierarchy of regularly subsampled textures and using the correct level for each screen pixel.
Unfortunately, mip-map cannot be applied to our patched texture (regular subsampling will destroy texture content). In general we need to use a smaller texture when we observe the represented object from a more distant viewpoint, and this is exactly the same situation as the one that arises when we switch to the following level of detail if an LOD data representation model is adopted. Therefore, if we consider the case of LOD representations, the problem can be solved a priori by a keen construction of the detail textures. In rendering environments that support LOD representation (e.g., VRML, OpenInventor) we may produce one mesh and one texture map for each different level of detail. These textures can be sampled with different sampling steps, and each sampling step can be set by taking into account the projected size (in display pixels) of the given object representation at the minimal distance that has been defined in the corresponding range field specification (using VRML terminology). The texture is therefore produced at a resolution that is directly dependent on the projection size forecasted. To ensure good texture quality, a wide sampling step has to be paired with a high oversampling value (such that each texel of a coarse texture will average the detail contained in the corresponding section of the original high-resolution mesh).
An example is shown in Fig. 14, where the same mesh is represented with two different LODs and the corresponding textures are computed either with approximately the same sampling step (meshes 1 and 2a), or adopting different screen-size-dependent sampling step (meshes 1 and 2b). More precisely, given the screen-size projection of the level 2 mesh at its minimal viewing distance, we built the level 2b of Fig. 14 using a much coarser sampling step (five times coarser than the one used in level 2a); to reduce aliasing in level 2b texture, we also adopted multisampling (with a 5×5 pattern of samples per sampling point).

6 Experimental results

The current prototype, preserving attributes by sampled textures (PASTex), accepts in input meshes formats OpenInventor, VRML V1.0 or raw (list of ver-

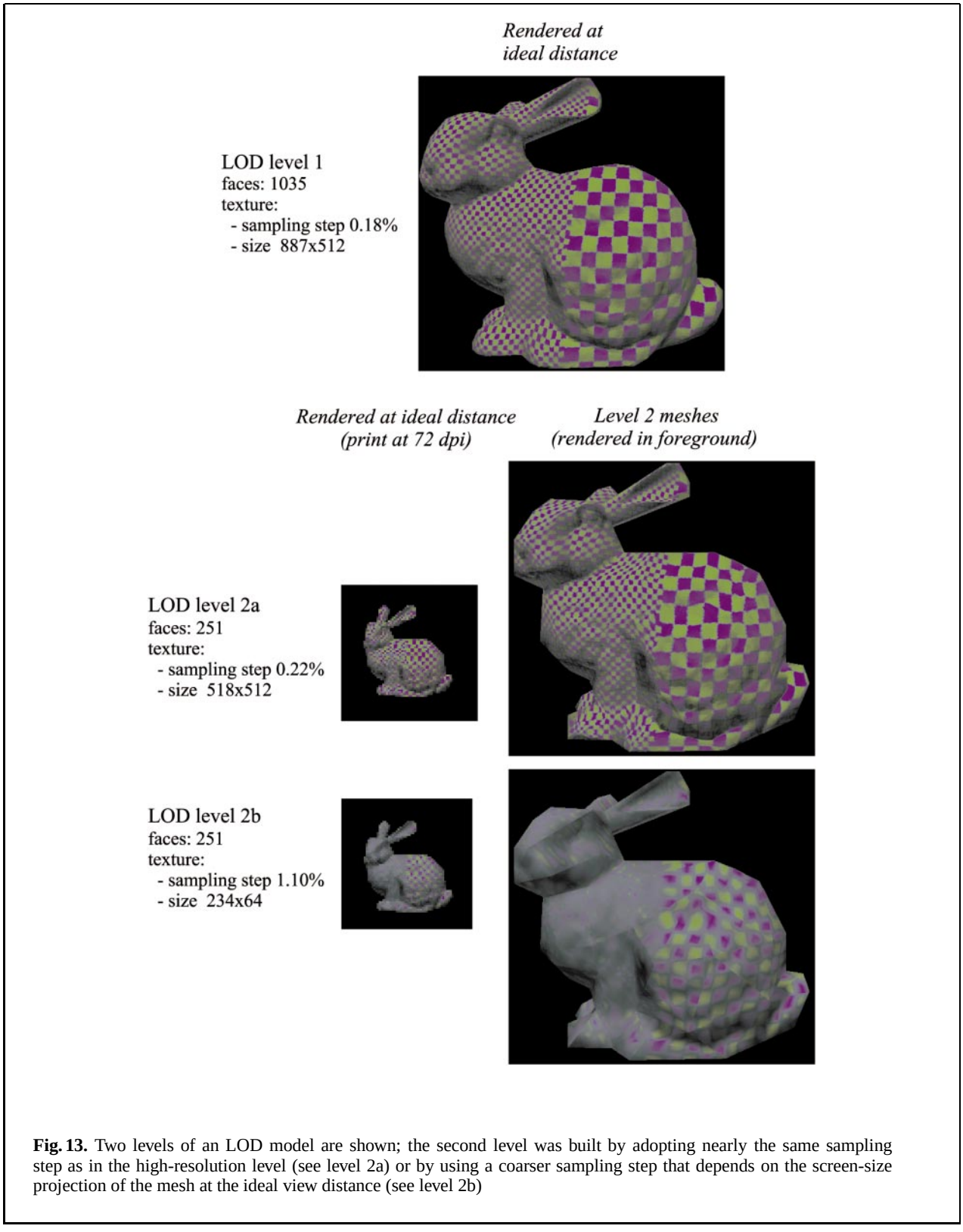


Fig. 13. Two levels of an LOD model are shown; the second level was built by adopting nearly the same sampling step as in the high-resolution level (see level 2a) or by using a coarser sampling step that depends on the screen-size projection of the mesh at the ideal view distance (see level 2b)

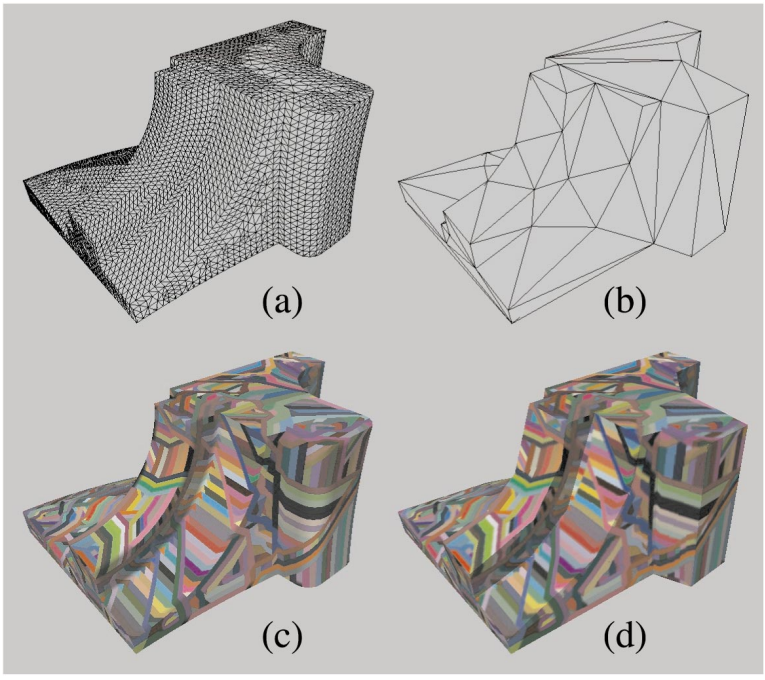


Fig. 14a–d. Mapping color from a per-vertex colored mesh: the original fandisk mesh (12 946 faces) is rendered wire frame (**a**) and shaded (**c**); the corresponding simplified model (98 faces) is rendered in **b** and, after mapping the color texture produced in **d**

tices plus list of faces). Since it would have been too expensive to get dynamically the required color information from these formats, we convert input data into an internal format that stores geometrical data and colors or texture space coordinates associated with each vertex. The data structure has also been designed to reduce space occupancy because we may need to store a very complex original mesh (the one before simplification) with all the associated attribute values.

Results are currently being produced by adopting the VRML format to store geometry, texture coordinates, and texture images.

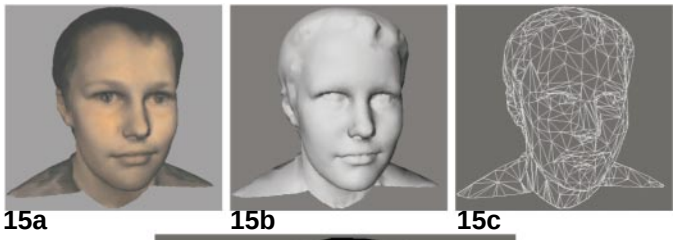
The system has been tested on many meshes. Here we report some examples, which are representative in terms of preserving various attributes. We show a set of images referring to the preservation of various types of detail:

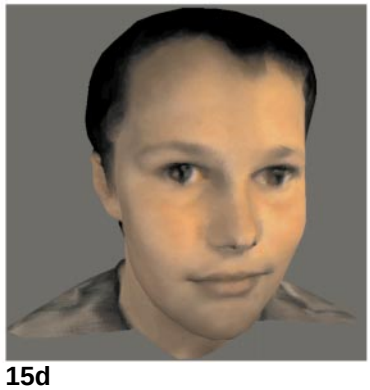
- *Per-vertex or per-face color.* Figure 15 shows an example where a CAD-style mesh (fandisk, 12 946 triangular faces) is painted with a number of very thin stripes (on average one face wide), then simplified, and finally color-textured; the two textures obtained using regular (on the left) and irregular packing (on the right) are shown in Fig. 11.
- *Texture-encoded color.* Figure 16 shows an example on a textured mesh (face[1], simplified to 1745 faces from the original 10 000 faces).
- *Shape.* Examples concerning shape detail preservation are presented in Figs. 17 and 18. Figure 17 shows the original bunny mesh[2] on the left (69 451 faces) and a simplified representation (501 faces) with both enhanced edges and preshaded bump mapping on the right. Figure 18 shows a section of a mesh that represents a relief drip stone (cornice) from the facade of the Duomo in Pisa (acquired with a commercial range scanner). It was simplified (from 70 050 to a few hundreds), and then a preshaded bump map was computed for the simplified model (rightmost image).
- *Scalar field.* An example of preservation of a scalar field (pressure on a plane body and wings[3] is shown in Fig. 19; the texture obtained has been rendered using the same transfer function used to render the original mesh.

[1] The face mesh was produced and distributed on the Web by Headus (Metamorphosis), http://www.headus.com.au/.

[2] The bunny mesh was range-scanned at Stanford University; available at http://www-graphics.stanford.edu/data/.

[3] The plane model is courtesy of Hans-Georg Pagendarm, Institute for Fluid Mechanics, German Aerospace Center (DLR).





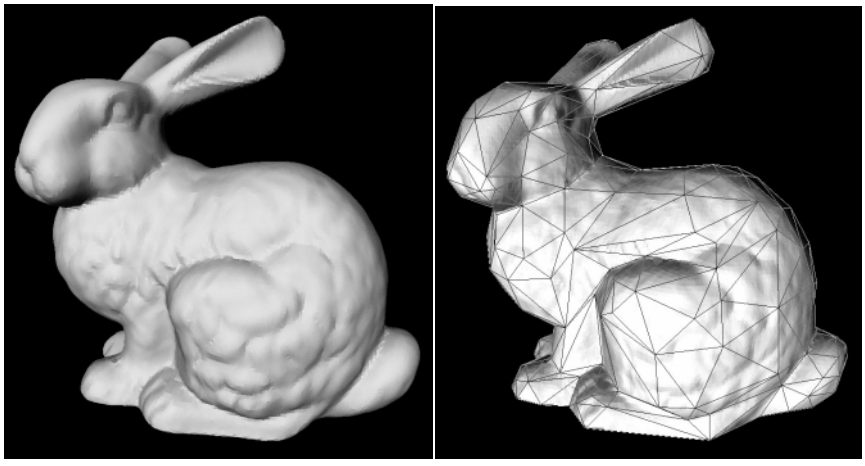
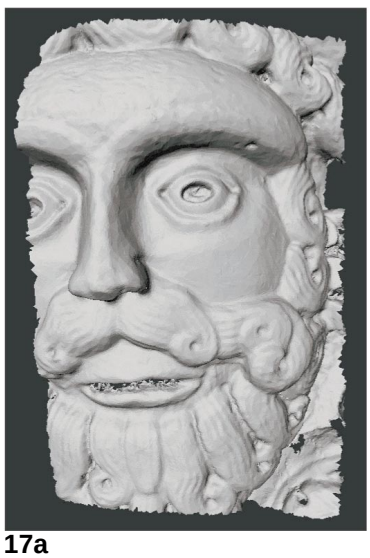


Fig. 15a–d. Preservation of color: the texture-based color of the face mesh [10 000 faces shown textured (**a**) and shaded (**b**)] is preserved on the simplified mesh [1745 faces, rendered wire frame (**c**) and color-textured (**d**)]

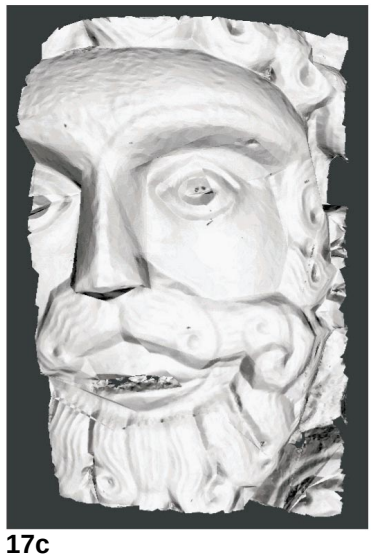
Fig. 16a,b. Mapping shape detail: **a** the original bunny mesh has 69 451 faces; **b** a simplified mesh (approximately 500 faces) with bump texture mapping and solid mesh edges

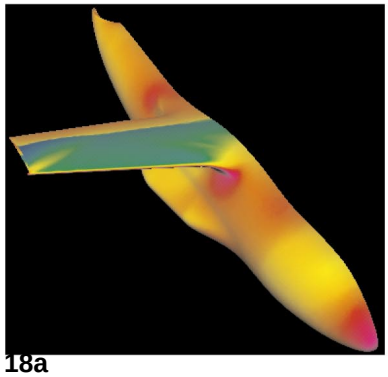
- *Procedural textures.* An example of coding 3D procedural textures with standard 2D textures is presented in Fig. 20, where a 3D procedural marblelike texture is attached to a 3D vase, converted into a 2D texture, and finally rendered with a standard OpenGL viewer.

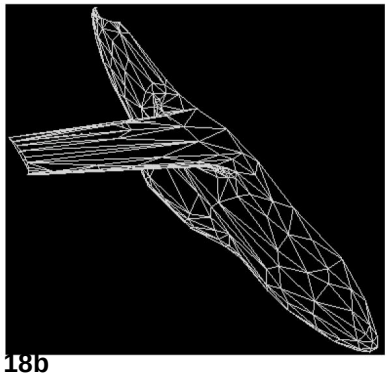
The optimization techniques adopted guarantee the efficiency of our approach. The time complexity of the various process phases is as follows. The initial phase, sorting of the original mesh faces and construction of the uniform grid, is $O(n_M log\, n_M)$. Texture sampling is theoretically $O(A_S * n_M)$, with A_S

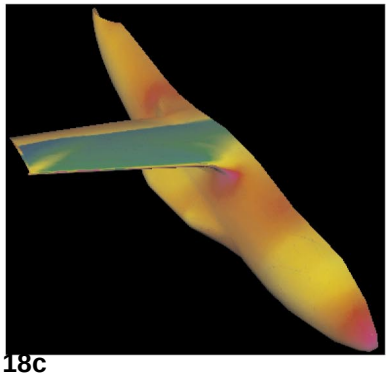

17a


17b


17c


18a


18b


18c

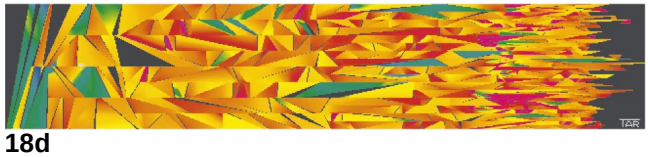

18d

Fig. 17a–c. Mapping shape detail: **a** the original mesh (70 050 faces); **b** the simplified mesh; **c** a displacement texture is mapped on the low resolution mesh

Fig. 18a–d. Preserving a scalar field (defined on a per-vertex base on the original mesh): **a** the original plane mesh (87 K faces); **b** the simplified mesh (500 faces); **c** is mapped on the simplified mesh; **d** the sampled field texture (mapped to color using the same transfer function)

19

20

Fig. 19. Mapping a 3D procedural marble texture using standard 2D textures

Fig. 20. Mapping a 3D procedural wood-like texture using standard 2D textures on a simple bunny model (251 faces)

the surface area of the simplified mesh S, measured in squared sampling step units, and n_M the size of the original mesh M (number of faces). However the use of the uniform grid reduces the impact of the n_M factor, and nearly constant times are required for the detection of the face nearest to each sampling point. Texture patches packing clearly depends on $O(n_S)$, i.e., the number of faces in the simplified mesh. The empirical results showed that the overall process is sufficiently efficient (generally a few seconds), and does not depend strictly on the size of mesh M.

An evaluation of the empirical time complexity of our approach is reported in Tables 1–3. The times are associated with the SGI O2, R5000 180 MHz, sampling step sizes are measured as percentages of the mesh bounding box diagonal, and the texture sizes are in pixels.

Table 1 reports all the data (mesh resolution, sampling step, times, space overhead, texture sizes) measured on four sample meshes fandisk, face, bunny, and plane. In particular, the space overhead is measured with respect to the ideal texture size, which is the surface area of the mesh measured in squared sampling step units. The texture overhead is generally lower than 100%, except for the face mesh. The overhead of the face mesh, 267%, is rather big due to the fact that the number of faces of the simplified mesh is not particularly compact (1745 faces) and the size of the texture is very small (681×64). Producing a larger resampled texture map made no sense in this case because the original rgb texture map associated with the input mesh is very small and distorted (it is a cylindrical map). The large number of small faces therefore leads to a rather large 203% overhead for the patch_borders component.

Table 2 shows the processing times of the three sub-phases (the initial phase is mostly due to the initialization of the bucketing data structure, sampling and texture packing) when various attributes are preserved: Color; shape detail in the two modalities, normal maps (Shape1) or displacement maps (Shape2); or both the previous detail integrated into a preshaded color texture (Both1, Both2). The mesh used in Table 2 is fandisk, simplified to 100 faces from the original 12 946. Table 2 also highlights the fact that the only phase that depends on texture resolution is the sampling time. Producing shape detail via displacements is slightly more expensive because we first need to evaluate displacements and then we compute each normal from the displacements around each sampling point.

Various sampling steps are used in Table 3 to measure the relative processing times and space overhead. The mesh used is again fandisk, simplified from 12 946 to 100 faces. Obviously, the smaller the sampling step, the higher the computing times and texture space. However, note that using a smaller sampling step causes a lower space overhead, mainly because the relative size of the redundant borders is reduced.

Two final questions need to be asked. Does replacing small triangles with larger textured elements really improve performance? What is the break-even point?
If we consider the transmission time, the answer is simple: we can measure the size of the original mesh and the composed size of the simplified mesh and the corresponding texture. To give an example, the original bunny mesh is 3.12 MB (SGI Inventor format) or 2.26 MB (PLY format); the simplified mesh in Fig. 17 requires 43–56 kB (.ply or .iv) to represent the geometry and 240 kB to represent the JPG-compressed texture (256×936).
If, conversely, we consider the image synthesis time, then the evaluation becomes much more difficult. If we divide graphics processing into the usual two phases, geometry stage and rasterization stage, then the rasterization cost (scan conversion, bilinear color and texture interpolation, Z-buffering) remains approximately the same and it is very efficient on most graphics boards. The geometry computations (transformations, lighting) cost is linear to the number of faces, and therefore it is largely reduced when a simplified model is used. Moreover, in a number of graphic systems the geometry stage is implemented software on the main CPU. Therefore the improvements guaranteed by the use of texture-enhanced simplified model may become notable. To give an empirical evaluation, we measured the frame rate while rendering the two bunny models on an SGI 320 NT PC. The original bunny runs at 8 fps, while the simplified textured bunny runs at 80 fps (which is the maximum refresh speed on the SGI 320).

7 Conclusions

We have presented a general method for preserving on a simplified mesh the detail (e.g., color, high-frequency shape detail, scalar fields, etc.) that is encoded in the original high-resolution mesh. Our approach is very general because it allows us to preserve any attribute value defined on the high-resolution mesh and because it can be used with any high-quality simplification code. The method presented makes no assumptions about the simplification process adopted to reduce mesh complexity, but only assumes that the simplified mesh is a good approximation (topologically and geometrically) of the original mesh. Detail is decoupled from geometry and is encoded through texture mapping, which is extremely efficient in many graphics subsystems. A texture, which encodes the detail of the high resolution mesh, is built by an efficient scan conversion process of the simplified mesh. The results therefore suffer as a result of some approximation: we preserve surface detail with the use of discrete texture maps, whose quality depends on both the sampling step size used and the criterion adopted to locate matching pairs of points on the two surfaces (which may introduce aliasing if the two meshes are not sufficiently similar in a geo-topological sense). Despite this limit, the results can be considered of a sufficiently high quality for a wide range of possible interactive visualization applications. In synthesis, the advantages of our approach are as follows: generality (it can be used with any simplifier, and it may restore every type of mesh attribute), computational efficiency, and geometrical robustness.
The approach proposed has another possible application: the conversion of models with attached 3D procedural textures into standard 2D-textured models. This constitutes a really new approach to managing 3D procedural textures because it allows us to render meshes with procedural detail on standard graphics subsystems with standard libraries or API such as OpenGL, VRML or Java3D.
Our approach should be compared with other solutions that take into account high-frequency detail during simplification. Our approach should be chosen for the following reasons: simplification is decoupled from detail preservation; decoupling shape simplification and detail preservation allow a much more drastic simplification to be obtained, especially in the case of meshes with a very complex detail content or when multiple attributes have to be preserved; when operated as a post-processing action, the task is generally more efficient in time and simpler to implement (especially if different kinds of detail have to be preserved); and finally, not all simplification approaches can be simply adapted to preserve the mesh attributes. However, our approach may produce less compact results than other solutions [13, 18] when the attribute field of the original mesh is either smooth or presents a small number of discontinuities. In these cases, the size of the detail texture may compromise the space improvement due to a more compact geometry.
A possible extension to our approach is as follows. To reduce texture size, if we are able to detect those texture patches whose detail varies bilinearly (ac-

cording to a user-defined accuracy) with respect to the value on the texture patch vertices, then we can represent in a much more compact manner the same information by simply assigning these values to the associated mesh vertices. In this case, we will produce a hybrid representation where some faces are linked to a texture patch (i.e., each vertex has texture coordinates), while others have per-vertex encoded detail values (and such values are interpolated during shading and rasterization). This approach can be highly effective in terms of space compression for all those objects that present large areas of surface where the detail is constant or varies smoothly and linearly.

Acknowledgements. This work was partially financed by the *Progetto Finalizzato Beni Culturali* of the Italian National Research Council (CNR). The relief drip stone from the Duomo in Pisa was scanned in cooperation with the "Soprintendenza ai Beni A.A.A.S." of Pisa, Italy. Most of the meshes used in experimentation are distributed free on the Web, and thus we thank all colleagues that made them available; in particular, special thanks to Hans-Georg Pagendarm for having provided us with the plane model. We also thank the anonymous referees for their useful comments and suggestions.

References

1. Akman V, Franklin WR, Kankanhalli M, Narayanaswami C (1989) Geometric computing and uniform grid technique. Comput Aided Design 21:410–420
2. Bajaj CL, Schikore DR (1998) Topology preserving data simplification with error bounds. Comput Graph 22:3–12
3. Blinn JF (1978) Simulation of wrinkled surfaces. (SIGGRAPH '78 Proceedings) Comput Graph 12:286–292
4. Certain A, Popović J, DeRose T, Duchamp T, Salesin D, Stuetzle W (1996) Interactive multiresolution surface viewing. In Rushmeyer H (ed) Comput Graph Proc, Annual Conf Series (SIGGRAPH '96), ACM Press, pp 91–98
5. Ciampalini A, Cignoni P, Montani C, Scopigno R (1997) Multiresolution decimation based on global error. Visual Comput 13:228–246
6. Cignoni P, Montani C, Rocchini C, Scopigno R (1998) A general method for recovering attribute values on simplifed meshes. IEEE Visualization '98, pp 59–66
7. Cohen J, Olano M, Manocha D (1998) Appearance-preserving simplification. In Cohen M (ed) Comput Graph Proc, Annual Conf Series (SIGGRAPH '98), ACM Press, pp 115–122
8. Cook RL (1984) Shade trees. (SIGGRAPH '84) Comput Graph 18:223–231
9. Ebert D, Musgrave K, Peachey D, Perlin K, Worley S (1994) Texturing and modeling: A procedural approach. Academic Press
10. Erikson C, Manocha D (1999) GAPS: General and automatic polygonal simplification. Proc of the ACM Symposium on Interactive 3D Graphics, ACM Press, New York, pp 79–88
11. Frank K, Lang U (1998) Data dependent surface simplification. In: Bartz D (ed) Visualization in Scientific Computing, Springer, Wien, New York, pp 3–12
12. Garland M, Heckbert PS (1997) Surface simplification using quadric error metrics. In Whitted T (ed) Conf Proc, Annual Conf Series (SIGGRAPH '97), Addison Wesley, pp 209–216
13. Garland M, Heckbert PS (1998) Simplifying surfaces with color and texture using quadric error metrics. In IEEE Visualization '98, ACM Press, New York, pp 264–270
14. Gross MH, Koch R (1995) Visualization of multidimensional shape and texture features in laser range data using complex-valued gabor wavelets. IEEE Trans Visualization Comput Graph 1:44–59
15. Heckbert P, Garland M (1997) Survey of surface simplification algorithms. Department of Computer Science, Carnegie Mellon University, `http://www.cs.cmu.edu/~ph`, to appear
16. Hoppe H (1996) Progressive meshes. In Rushmeier H (ed) ACM Comput Graph Proc, Annual Conf Series (SIGGRAPH '96), Addison Wesley, pp 99–108
17. Hoppe H (1997) View-dependent refinement of progressive meshes. In Whitted T (ed) ACM Comput Graph Proc, Annual Conf Series (SIGGRAPH '97), addison Wesley, pp 189–198
18. Hoppe H (1999) New quadric metric for simplifying meshes with appearance attributes. IEEE Visualization '99, ACM Press, New York, pp 59–66
19. Krishnamurthy V, Levoy M (1996) Fitting smooth surfaces to dense polygon meshes. In Rushmeier H (ed) Comput Graph Proc, Annual Conf Series (SIGGRAPH '96), Addison Wesley, pp 313–324
20. Maruya M (1995) Generating texture map from object-surface texture data. Comput Graph Forum 14:397–405
21. Microsoft Directx 6.0 specifications. More info on: `http://www.microsoft.com/directx` (1998)
22. Peercy M, Airey J, Cabral B (1997) Efficient bump mapping hardware. Conf Proc, Annual Conference Series (SIGGRAPH '97), Addison Wesley, pp 303–306
23. Perlin K (1985) An image synthesizer. Comput Graph 19:287–296
24. Pulli K, Cohen M, Duchamp T, Hoppe H, Shapiro L, Stuetzle W (1997) View-based rendering: visualizing real objects from scanned range and color data. Proceedings of 8th Eurographics Workshop on Rendering, St. Etienne, France, Springer, Wien, New York, pp 23–34
25. Puppo E, Scopigno R (1997) Simplification, LOD, and multiresolution – principles and applications. EUROGRAPHICS '97 Tutorial Notes (ISSN 1017-4656). Eurographics Association, Aire-la-Ville, Switzerland
26. Rafferty MM, Aliaga DG, Popescu V, Lastra AA (1998) Images for accelerating architectural walkthroughs. IEEE Comput Graph Appl 18(6)
27. Reddy M (1996) Scrooge: perceptually-driven polygon reduction. Comput Graph Forum 15:191–203
28. Worley S (1996) A cellular texture basis function. In Rushmeier H (ed) Comput Graph Proc, Annual Conf Series (SIGGRAPH '96), Addison Wesley, pp 291–294
29. Schikore D, Bajaj C (1995) Decimation of 2D scalar data with error control. Technical Report CSD-TR-95-

005, Department of Computer Science, Purdue University, http://www.cs.purdue.edu/homes/drs/research.html
30. Schroeder WJ, Zarge JA, Lorensen WE (1992) Decimation of triangle meshes. (SIGGRAPH '92) Comput Graph 26:65–70
31. Soucy M, Godin G, Rioux M (1996) A texture-mapping approach for the compression of colored 3D triangulations. Visual Comput 12:503–514
32. Weinhaus FM, Devarajan V (1997) Texture mapping 3D models of real-world scenes. ACM Comput Survey 29:325–365
33. Williams L (1983) Pyramidal parametrics. (SIGGRAPH '83) Comput Graph 17:1–11

PAOLO CIGNONI is a research scientist at the Istituto di Elaborazione della Informazione of the National Research Council in Pisa, Italy. His research interests include computational geometry and its interaction with computer graphics, scientific visualization, volume rendering, simplification and multiresolution. Cignoni received an advanced degree (Laurea) in 1992 and a PhD in Computer Science in 1998 from the University of Pisa.

CLAUDIO MONTANI is a Research Director with the Istituto di Elaborazione della Informazione of the National Research Council in Pisa, Italy. His research interests include data structures and algoritms for volume visualization and rendering of regular or scattered datasets. He received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1977. He is member of the IEEE.

CLAUDIO ROCCHINI is a Research Scientist at the Istituto di Elaborazione della Informazione of the National Research Council in Pisa, Italy. His research interests include surface modeling, simplification, multiresolution, and automatic acquisition of 3D models. Rocchini received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1997.

ROBERTO SCOPIGNO is a Senior Research Scientist at the Istituto di Elaborazione della Informazione of the National Research Council in Pisa, Italy. He is currently engaged in research projects concerned with scientific visualization, volume rendering, web-based graphics, multiresolution data modeling and rendering, range scanning, and applications of 3D computer graphics to Cultural Heritage. He received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1984. He is member of the IEEE and Eurographics.

MARCO TARINI is a PhD student at the Computer Science Department of the University of Pisa and a Research Fellow at the Istituto di Elaborazione della Informazione of the National Research Council in Pisa, Italy. His research interests include texture mapping, image based modeling and rendering, and automatic acquisition of 3D models. He received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1998.