A general method for preserving attribute values on simplified meshes

P. Cignoni,* C. Montani,† C. Rocchini,‡ R. Scopigno§

Istituto di Elaborazione dell'Informazione – Consiglio Nazionale delle Ricerche

Abstract

Many sophisticated solutions have been proposed to reduce the geometric complexity of 3D meshes. A less studied problem is how to preserve on a simplified mesh the detail (e.g. color, high frequency shape detail, scalar fields, etc.) which is encoded in the original mesh. We present a general approach for preserving detail on simplified meshes. The detail (or high frequency information) lost after simplification is encoded through texture or bump maps. The original contribution is that preservation is performed after simplification, by building set of triangular texture patches that are then packed in a single texture map. Each simplified mesh face is sampled to build the associated triangular texture patch; a new method for storing this set of texture patches into a standard rectangular texture is presented and discussed. Our detail preserving approach makes no assumptions about the simplification process adopted to reduce mesh complexity and allows highly efficient rendering. The solution is very general, allowing preservation of any attribute value defined on the high resolution mesh. We also describe an alternative application: the conversion of 3D models with 3D static procedural textures into standard 3D models with 2D textures.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - *Display algorithms*; I.3.6 [Computer Graphics]: Methodology and Techniques.

Additional Keywords: surface simplification, detail preservation, texture mapping.

1 INTRODUCTION

Mesh simplification technology has evolved substantially in the last few years, and many approaches have been proposed for the controlled simplification of simplicial meshes. A possible schematic classification subdivides the large number of techniques into: *coplanar facets merging* [11, 14], *mesh decimation* [27, 23, 3], *energy function optimization* [13, 12], *clustering* [24, 7], and *wavelet-based* [2]. We cite only a few, representative papers; comprehensive overviews have been given in two recent papers [9, 21].

Many methods proposed offer no immediate provision for accurately controlling the perceptual effect of degradation, because in many cases the simplification criterion has no immediate interpretation in terms of *visual degradation* [22]. Perceivable visual degradation may be caused either while visualizing a single simplified representation (e.g. in the case of an excessively simplified mesh, with loss of topology features and/or fuzziness of the simplified surface), or while changing the level of detail, the so-called *inter-frame flickering* which is common if the meshes in an 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 on the appearance of the shape, one reason being that most renderers draw elementary components by shading colors according to surface normals [22]. However, for many applications, taking into account only the pure geometric approximation is not sufficient to ensure that the required accuracy is fulfilled. Pictorial information (color or texture) is an important factor in perception, and therefore preservation of color discontinuities has to be managed carefully. Few papers have seriously taken into account this problem [16, 22, 12, 2, 28, 6, 20], as summarized in Section 2.

Moreover, pictorial information is probably the most common attribute, but it is not the only one. Another type of mesh attribute may be the sampling of a field over the mesh vertices (e.g. the sampled value of physical variables like temperature, potential and pressure). In order to use these field values in visualization (e.g. by mapping field values to color, or by computing iso-lines), a simplification code should take into account the value of the field while reducing the complexity of the mesh [26, 12].

A new, general approach is proposed in this paper to preserve attribute information on simplified meshes built on dense, triangular meshes. The approach we propose is orthogonal to previous solutions, which take into account attribute values during the simplification process. Conversely, we propose a simple solution for preserving attribute information after a generic simplification process has been run on the input mesh. Attribute preserving can be seen as a post-processing phase, which may be used in conjunction with any simplification code. It is performed by assuming that the simplified mesh S has a sufficiently similar shape if compared with the original high resolution mesh M. A sampling process is applied to the simplified mesh (e.g. each face is scan-converted in object space) and 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. The retrieved attribute values are then stored in a texture map [10], which we later use to paint the pictorial detail of mesh M onto mesh S. All of the 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.

This approach has a number of advantages. It can be used with any simplifier, because no assumptions or constraints on the simplification approach are made. It may restore every type of mesh attribute, given an interpolation rule that allows us to compute the attribute value on each point of the input mesh (examples on the restoration of color, other scalar/vectorial fields and high-frequency

*Email: cignoni@iei.pi.cnr.it
†Email: montani@iei.pi.cnr.it
‡Email: rocchini@calpar.cnuce.cnr.it
§Email: r.scopigno@cnuce.cnr.it

surface perturbations are shown in the following). Its complexity depends on the simplified surface area (measured in elementary sampling point units). Moreover, this approach may also be used to solve another problem, that is how to convert an object description which uses *procedural textures* [5] into a format that only supports image-based textures. Finally, analogously to other approaches based on textures [16, 2, 28, 15], modeling high frequency detail with texture maps allows highly efficient rendering on modern graphics subsystems, which generally support hardware texture mapping.

The paper is organized as follows. Section 2 outlines previous research in this area. Our approach is described in detail in Section 3, while implementation-related aspects and the architecture of our prototypal implementation are reported in Subsection 3.2. Section 4 shows how the proposed approach can be extended to manage procedural textures. An evaluation of the results obtained on a set of sample meshes is presented in Section 5. Finally, conclusions are drawn in Section 6.

2 PREVIOUS WORK

Only few simplification papers have taken into account the attribute preservation problem, which is a very critical issue with a direct impact on the actual use of the meshes produced.

If the pictorial information is defined with a single texture map and the mesh is simplified using a *decimation* approach [27], then the problem of color preservation is trivial. However, if color is defined on a per-vertex basis, or through multiple textures, the problem becomes much more complex, and even worse if we have multiple attributes. Basically, the *detail-preserving* simplification methods proposed can be categorized as follows:

1. approaches that take into account attribute discontinuities during the simplification process; discontinuities are usually detected by the selection of a discontinuity threshold value, and the removal or collapse of candidate elements that exceed the discontinuity threshold is not allowed;

2. approaches which, during simplification, maintain associated with each current mesh element (e.g. a face) the attribute values associated to deleted/collapsed elements of the original mesh section now represented by the current element.

Approaches that follow the first strategy were proposed in [12, 6]. A disadvantage of this strategy is that, because attribute discontinuities prevent element decimation/collapsing, a drastic simplification of meshes with complex pictorial detail is often impossible.
Color-preserving solution which adopt the second strategy are described in [16, 28]. The color/pictorial information defined over the mesh (e.g. an rgb-color for each vertex of the high resolution mesh) is preserved during simplification by building a mapping between the original mesh vertices and the simplified mesh faces. Once the mesh has been simplified, a texture is built for each face using the colors of the associated removed vertices. Methods for packing the simplified face textures in a single texure map have been proposed [16, 28].

An approach similar to the one proposed in the present paper adopts B-splines surfaces for concisely representing dense irregular polygon meshes, and then maps shape detail through displacement maps [15]. For each spline mesh section, a displacement map represents the error between the fitted surface and the original polygon mesh section. The resulting B-spline surface and the companion displacement maps can then be processed with standard photorealistic rendering systems.

Wavelet decomposition has been widely used for the generation of simplified or multi-resolution representations. An approach that follows the first strategy above represents, at multiple resolutions, both the shape and color of *range-images* using non-orthogonal Gabor wavelets [8]. Unlike other solutions based on the use of wavelet decomposition, in [8] the *shape* and the *texture* are considered as the amplitude and phase of a complex-valued 2D image onto which a hierarchical decomposition is built.
Wavelets have also been proposed for separate multi-resolution representation of geometry and color, which are combined only at display time [2]. This allows the selection at visualization time of independent levels of approximation for both geometry and color. The authors also propose representing color detail through texture-mapping; textures are built dynamically, with the required resolution, by painting the color wavelets coefficients on the texture map.

3 PRESERVING DETAIL ON SIMPLIFIED MESHES

The proposed approach is described here in a general manner. Let us assume that we are interested in recovering onto the [simplified] mesh S the value of a scalar/vectorial field F defined on mesh M. We assume that the field value can be computed for each point $p \in M$ by means of a function $f : M \subset R^3 \rightarrow [R \mid R^3]$. 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).

Our approach is as follows. The surface of the simplified mesh S is sampled, and for each elementary surface sample p_S we compute the point $p_M \in M$ at minimal Euclidean distance from p_S (Figure 1). This nearest point is computed efficiently by using a bucketed data structure for the representation of the mesh M (faces are stored using a uniform grid, and grid cells are processed on the base of their distance from the sampling point p_S). This approach generates a point–to–point relationship within surfaces S and M, and requires no knowledge of the simplification approach adopted to build the reduced mesh. Surface sampling is achieved by scan-converting triangular faces under a user-selected sampling resolution. The sampling resolution characterizes the quality (and the size) of the texture produced in output.
Given a sampling step, the mesh may contain triangles smaller than the single squared sampling step. To reduce the aliasing associated with this case, the user can specify the minimum number of samples that have to be evaluated for each triangle.

Then, for each 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 which 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; etc.

- *data on shape detail*: the distance between points p_S and p_M can be used to define either an approximated *bump map* [1] or a *displacement-map* [4, 10]. The displacement between the two surfaces is discretized and stored in a 2D scalar matrix (and will be rendered interactively adopting forthcoming graphics subsystems with hardware bump mapping features [17]).

For each face, the collection of sampled values is stored in a triangular texture patch (which may contain either RGB, field, displacement or surface normal perturbation values). The size of this texture patch directly depends on the sampling step size chosen by the user and on the size of the triangular face.

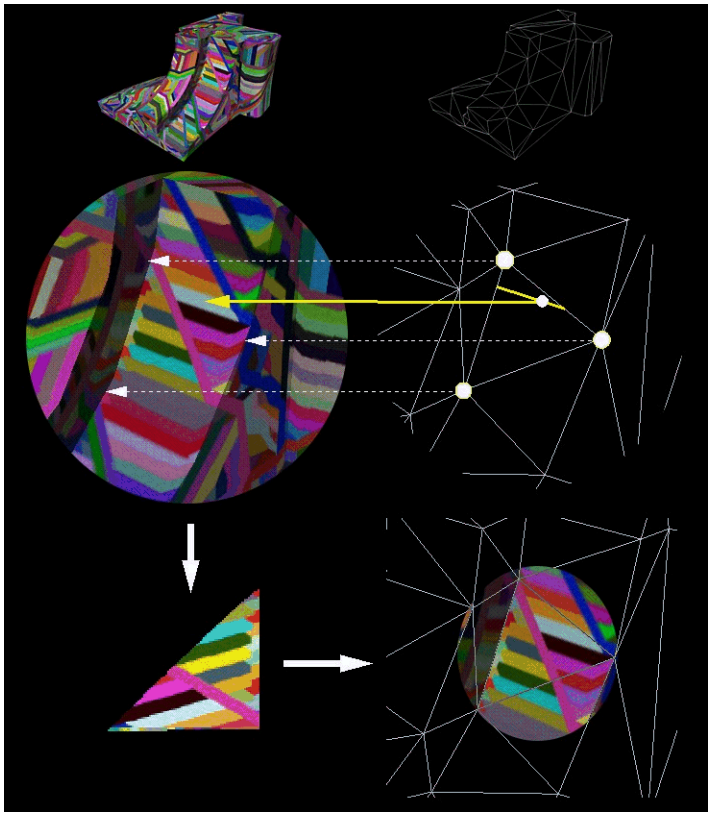

Figure 1: Mapping color detail on a simplified mesh by sampling color and constructing an rgb-texture.

The quality of this texture is crucial. Because our approach is an approximate one, the following problems may arise:

- *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 in a very abrupt manner (e.g. a face painted with a number of different colored stripes, and where adjacent stripes show sharp color discontinuity). But this problem cannot be solved by simply decreasing the sampling step size, because the size of the texture cannot be enlarged too much (texture memory available on graphics subsystems is often limited).
  Our tool adopts over-sampling to improve texture quality. Multiple samples are computed for each sampling point p (distributed regularly in the small sampling area associated to p), and the mean of these samples is stored in the texture. Currently, the over-sampling value is one of the parameters specified by the user;

- *aliasing* on the adjacency border between different textured faces. This problem is related to the fact that, once packed (see next paragraph), texture patches which are adjacent in the overall texture map are generally associated with non-adjacent faces of the mesh. During scan conversion of a mesh face, discrete texture coordinates might be produced which 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 which is slightly wider (one texture pixel wider in the discrete texture space). This give rise to slightly larger texture maps, but solves the above aliasing problem.

3.1 Packing texture patches in rectangular textures

The last step of our algorithm is to pack all the triangular texture patches into a single rectangular 2D texture.
Different approaches are possible to store all of the triangular texture patches in a single rectangular texture.

A first approach may be based on the use of rectangular texture patches, which due to they regularity in shape result very simple to be packed in a rectangular texture [16, 28]. Each texture patch is generated in [16, 28] by: storing during simplification the color of the decimated vertices which map onto each simplified mesh face, and then by interpolating the color information associated with the removed vertices to build a high resolution triangular texture map. These techniques use only half-square right triangles, with a further limitation to square edge of size 2^i in [28]. The use of only half-square texture patches means that they can be packed easily with a simple and regular rule: patches are stored in order of magnitude (biggest first), equal size patches are paired to form squares, and squares are stored in adjacent texture areas.

The use of half-squared texture patches allows 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 for the reconstruction of the texture patches, the first point implies that we don't 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

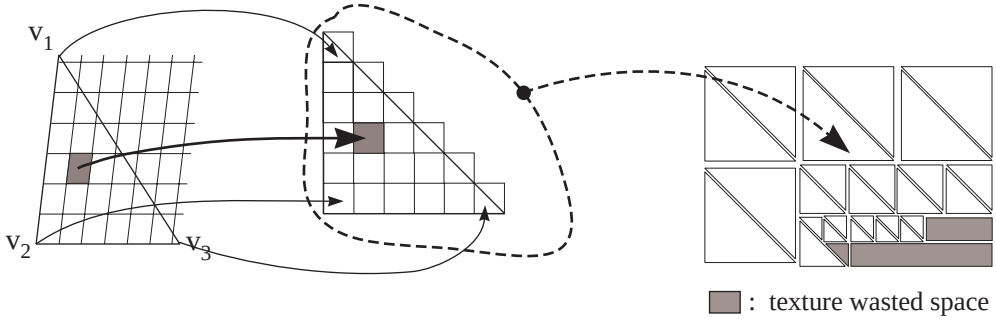

Figure 2: Using half-square texture patches: easy to pack in a rectangular texture, but an uneven sampling is performed.

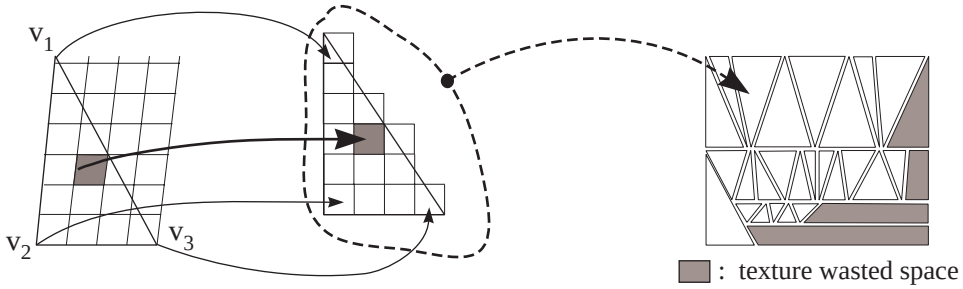

Figure 3: Using irregular texture patches: easy to pack in a rectangular texture, with even sampling.

$\frac{length(e)}{2^k} \leq s \leq \frac{length(e)}{2^{k+1}}$. The second limitation implies that the elementary sampling area is a rectangular area (not squared) in the mesh space (see Figure 2).
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 two dimensions, and this introduces uneven sampling. The actual sampling size selected by the user is an upper bound of the sampling size which 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 which allows the use of *irregular texture patches*. 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. Irregular texture triangles are therefore generated, but because they still have only a discrete number of different heights, we can store them in a compact manner. The packing strategy is based on the idea of *shearing* each texture patch along the X axe, in order to reduce the space which is wasted when we try to pack a patch along the border of the previous patch in the rectangular texture. The example in Figure 3 shows the border-line case, where each patch is drastically sheared until his leftmost edge has inclination identical to the right-most one of the adjacent patch.
Obviously, considerable shearing introduce aliasing, while the use of small shearing angles results sufficiently safe in terms of visual quality. The maximum angle of shearing is one of the parameters of the system (more we share, less texture space is wasted, but more aliasing is introduced).

The two approaches have been implemented and evaluated. Preliminary results shows that the second solution gives smaller textures. If we define the ideal texture size as the mesh surface area divided by the squared sampling step, then the second solution produces textures which are on average 15-20% larger than the ideal size, while the first solution returns textures 250-300% larger than the ideal one.
Given a maximal texture size (which is a common system-dependent constraint), the lower the overhead (e.g. the difference between the ideal and the sampled texture), the better the quality (because we can increase the sampling rate and obtain a more detailed texture).

An example of the packed textures obtained with the Soucy et al. approach [28] and our approach is shown in Figure 9 (see color tables). The example shows the texture map built for a very simplified model (98 faces) of the fandisk mesh (originally, 12,946 faces). The sampling step size is in this case 0.25% of the mesh bounding box diagonal and the size of the texture maps obtained is 780*1024 in the case of the Soucy et al. method, and 659*521 for our method. The time required is 3.3 sec for initialization, 6. seconds for texture patches sampling and only 0.1 sec for texture patches packing (in the case of the slightly more complex irregular patches packing). Times have been evaluated on an SGI O2, R5000 180MHz.

3.2 Implementation details

This subsection presents some more details on our current prototypal implementation, **DePreSS** (Detail Preservation on Simplified Surfaces).
The *DePreSS* prototype accepts in input meshes formats OpenInventor, VRML V1.0 or raw (list of vertices plus list of faces). Since it would have been too expensive to get dynamically the required color information from the above formats, we convert input data into an internal format which 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.

Given a sampling point p on S, the corresponding point on the original mesh M is computed with sufficient efficiency by evalu-

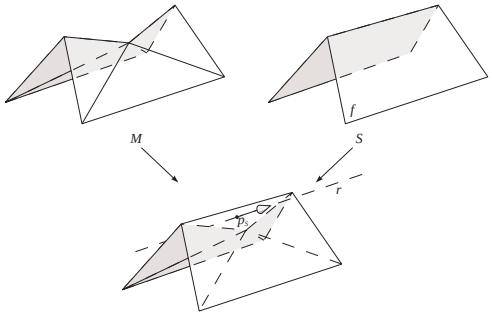

Figure 4: The ray r passing through the sampling point p_s on the simplified surface S and perpendicular to the face f of S does not pierce any point of the original surface M.

ating *point_to_face* distances (we used an optimized procedure derived from the source code of the POV ray-tracer [19]). 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 bucketed representation of the faces of M. A regular cubic grid 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 in order of distance from p, halting the process as soon as no more buckets closest 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 (we subdivide the bounding box into $n * m * p$ cubic cells).

In the case of the computation of bump maps, we store in the texture the signed distance between the two meshes. The *point_to_face* distance evaluated takes into account if the original mesh is above or below the surface of the simplified mesh. However, these distances only give an approximation of a real bump/displacement map, because displacements are generally evaluated on the direction of the normal to the sampled surface. But computing *correct* distances along the surface normal direction r is not easy, because in some cases it might not exist an intersection between the original mesh M and the ray r (see an example in Figure 4), or the nearest intersection along r might be on a section of the mesh which is much further than the section nearest to the sampling point p_s.
In our implementation, we return the minimal distance between meshes M and S (and the line which passes through the corresponding pair of points may not be aligned with the normal in S). A number of experiments showed that the level of approximation is sufficiently good (see Figures 6 and 7).

The results are currently being produced by adopting the OpenInventor format, storing geometry, texture coordinates, and the texture image.

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). They have been proposed to circumvent the image mapping problem that characterizes 2D textures [10, 5]. Moreover, 3D textures are generally obtained by procedural generation, to prevent prohibitive space requirements. The texture function value in every location of the 3D space may be computed using either fractal noise basis functions [18], or functions based on the computation of distances from a set of random feature points [25].

Procedural textures can easily be mapped onto a free-form object, but are generally rendered by adopting a ray-casting approach. Many applications use procedural textures, and one problem is how to preserve detail specification when a standard scene description language (OpenGL, OpenInventor or VRML) based on planar, 2D textures is adopted, e.g. to ensure efficiency or portability to distributed environments. An example may be the conversion of a complicated model created with a commercial tool which 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 the texture space. During mesh sampling we only need to recompute, for each sampling point, the associated procedural texture value, and then store it in the texture patch associated with the current face. Patches are then packed as usual in a rectangular 2D texture map.

5 EXPERIMENTAL RESULTS

The proposed system has been tested on many meshes. We report here some examples, which are representative in terms of preserving different attributes. We show two sets of images referring to *color* or *pictorial data* preservation:

- meshes with color defined on a *per-vertex* basis; Figure 9 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 texture obtained,packed using the Soucy et al. and our methods, are shown in Figure 9 (see color tables);

- meshes with *texture-mapped* pictorial detail (single or multiple textures); Figure 5 shows an example[1].

Examples concerning shape detail preservation are presented in Figures 6 and 7. Figure 6 shows the original bunny mesh[2] on the left (69,451 faces) and a simplified representation with both enhanced edges and bump mapping on the right. Figures 7 shows a section of a mesh which represents a relief drip stone (cornice) from the facade of the Duomo in Pisa (acquired with a commercial range scanner). It was simplified drastically (from 70,050 to a few hundreds), and then a bump map, computed with our prototype, was applied on the simplified model (rightmost image).

An example of coding 3D procedural textures with standard 2D textures is presented in Figure 8, where a 3D procedural marble-like 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. Empirical complexity is nearly linear to the surface area of the simplified mesh, which is measured in squared sampling step units. This is because the uniform grid adopted to store the high resolution mesh allows nearly constant times for the detection of the nearest face to each sampling point. An evaluation of the empirical time complexity of our approach is reported in Table 1, where: times are in seconds (SGI O2, R5000 180MHz), sampling step sizes are measured as percentages of the mesh bounding box diagonal, and texture sizes are in pixels (texture sizes reported in the table are only those of the Soucy et al. packing method).

[1]The puppet mesh (13,045 faces) was produced by Wolfgang Niem, Univ. of Hannover (D), http://www.tnt.uni-hannover.de/project/3dmod/multview/examples.html .

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

Figure 7: Mapping shape detail: the original mesh on the left (70,050 faces) has been drastically simplified (center image), and then a displacement texture is mapped on the low resolution mesh (on the right).

sampling step	No. samples	time	texture size
FANDISK (orig.: 12,946 faces, simpl.: 98 faces)			
1.000 %	10,827	4.69	23,680
0.500 %	43,456	6.41	82,432
0.250 %	173,985	13.03	308,992
0.125 %	696,066	39.13	1,196,544
BUNNY (orig.: 69,451 faces, simpl.: 501 faces)			
1.000 %	11,547	20.09	35,104
0.500 %	46,953	24.94	83,264
0.250 %	188,553	44.97	282,176
0.125 %	754,974	121.07	1,051,776

Table 1: Processing time (in seconds), number of points sampled on the surface, and size (in pixels) of the texture map produced in output by our prototypal implementation on two simplified meshes and different sampling step sizes.

6 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.) which is encoded in the corresponding high resolution mesh. Our approach is very general, because it allows one to preserve any attribute value defined on the high resolution mesh, and because it makes no assumptions about the simplification process adopted to reduce mesh complexity. Detail is encoded through texture mapping, which is extremely efficient in many graphics subsystems. The texture encoding the detail of the high resolution mesh is built by an efficient scan conversion process of the simplified mesh. The results therefore suffer for 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. Despite this limit, the results can be considered of sufficiently high quality for a wide range of possible interactive visualization applications.
We also propose an alternative application: the conversion of models with attached 3D procedural textures into standard 2D-textured models.

As a general point, our approach has to be compared with the other solutions which take into account high-frequency detail during simplification. Reasons for choosing our approach are the following: simplification is de-coupled from preservation, and this allows the user to choose the more adequate simplification code; when operated as a post-processing action, the task is generally more efficient in time and simpler to be implemented (especially if different kinds of detail have to be preserved); not all simplification approaches can be simply adapted to preserve the mesh attributes.

Our approach can be extended in several ways.
The precision of the sampling process and texture size depend on the sampling step selected. The quality of the textures produced might be improved (and their size reduced) if a semi-automatic and adaptive selection of the sampling step were adopted. Once a texture patch has been computed for a given face of the mesh, this patch could be analysed to detect the degree of uniformity or smoothness of the detail coded. If the texture patch is nearly uniform (e.g. if it corresponds to a constant value section of the mesh) or smooth, then its resolution could be reduced without losing quality. Conversely, if the texture has a high discontinuous content, we could re-sample it using finer sampling , and increase the precision only locally.

Mip-map representation of textures [29] is widespread, due to the necessity to reduce aliasing while mapping the texture at different scales. The problem solved by the mip-map approach is how to render faces which map to a large texture patch but, in a current view, map to only a few pixels. Storing our sampled texture under a mip-map approach is not straightforward. The problem is how to compute recursively the associated reduced resolution maps by taking into account the fact that our global texture is subdivided into a number of independent triangular patches. This problem will be the subject of further research.

7 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).

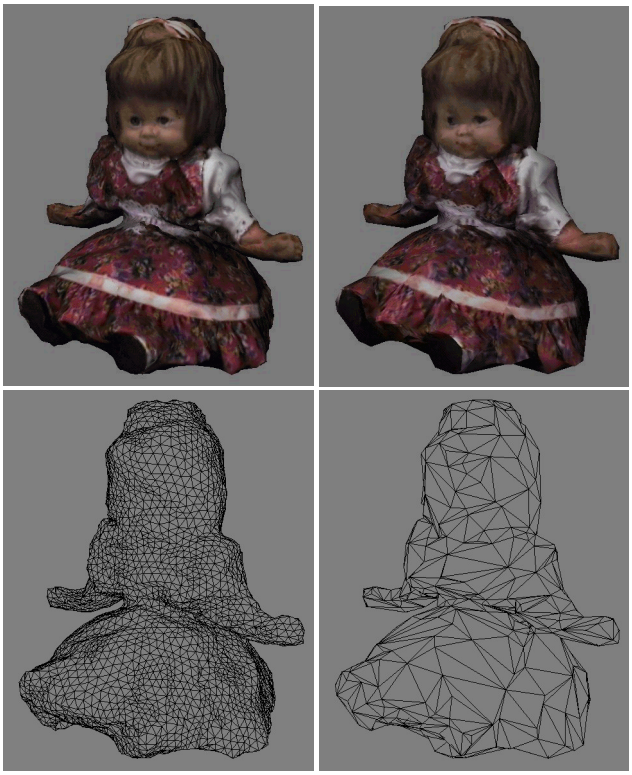

Figure 5: Preservation of color: the texture-based color of the mesh on the left is preserved on the simplified mesh on the right.

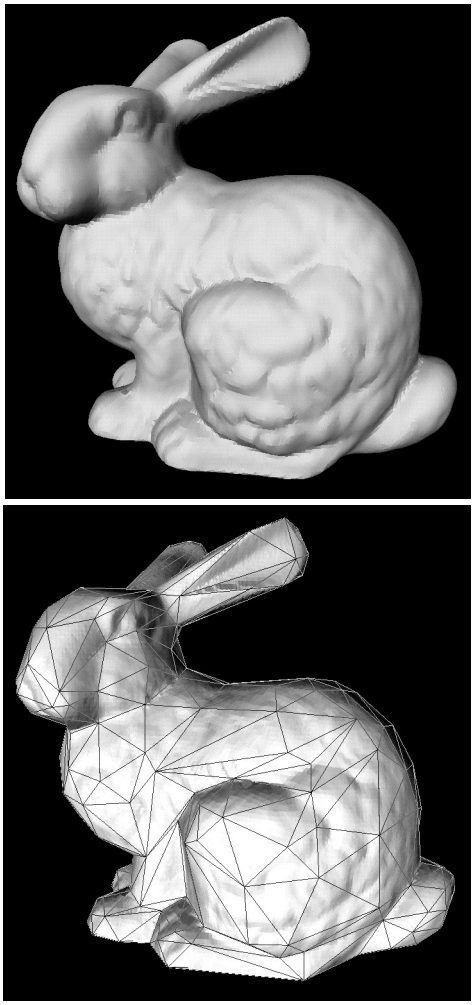

Figure 6: Mapping shape detail: the original *bunny* mesh is on the left (69,451 faces); a drastically simplified mesh with bump texture mapping and solid mesh edges is on the right.

References

[1] J. F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):286–292, August 1978.

[2] A. Certain, J. Popovic, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive multiresolution surface viewing. In *Comp. Graph. Proc., Annual Conf. Series (Siggraph '96), ACM Press*, pages 91–98, Aug. 6-8 1996.

[3] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, June 1997.

[4] R. L. Cook. Shade trees. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):223–231, July 1984.

[5] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin, and Worley. *Texturing and Modeling: A Procedural Approach*. Academic Press, October 1994. ISBN 0-12-228760-6.

[6] K. Frank and U.Lang. Data dependent surface simplification. In D. Bartz, editor, *9th Eurographics Workshop on Visualization in Scientific Computing (EG ViSC'98)*, pages 100–109. Eurographics, 1998.

[7] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *Comp. Graph. Proc., Annual Conf. Series (Siggraph '97), ACM Press*, pages 209–216, 1997.

[8] M.H. Gross and R. Koch. Visualization of multidimensional shape and texture features in laser range data using complex-valued gabor wavelets. *IEEE Trans. on Visual. and Comp. Graph.*, 1(1):44–59, March 1995.

[9] P. Heckbert and M. Garland. Survey of surface simplification algorithms. Technical report, Carnegie Mellon University - Dept. of Computer Science, 1997. (to appear).

[10] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[11] P. Hinker and C. Hansen. Geometric optimization. In *IEEE Visualization '93 Proc.*, pages 189–195, October 1993.

[12] H. Hoppe. Progressive meshes. In *ACM Computer Graphics Proc., Annual Conference Series, (Siggraph '96)*, pages 99–108, 1996.

[13] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *ACM Computer Graphics Proc., Annual Conference Series, (Siggraph '93)*, pages 19–26, 1993.

[14] A. D. Kalvin and R.H. Taylor. Superfaces: Poligonal mesh simplification with bounded error. *IEEE C.G.&A.*, 16(3):64–77, 1996.

[15] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *Comp. Graph. Proc., Annual Conf. Series (Siggraph '96), ACM Press*, pages 313–324. ACM Press, 1996.

[16] M. Maruya. Generating texture map from object-surface texture data. *Computer Graphics Forum (Proc. of Eurographics '95)*, 14(3):397–405, 1995.

[17] Mark Peercy, John Airey, and Brian Cabral. Efficient bump mapping hardware. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 303–306. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[18] K. Perlin. An image synthesizer. *Computer Graphics*, 19(3):287–296, July 1985.

[19] POV-Team. Persistence of vision raytracer 3.0. Publicly available on web: `http://www.povray.org/`, 1996.

[20] K. Pulli, M. Cohen, T. Duchamp, H.Hoppe, L. Shapiro, and W. Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In

Figure 8: Mapping a 3D procedural marble texture using standard 2D textures.

Proceedings of 8th Eurographics Workshop on Rendering (St. Etienne, France), June 1997.

[21] E. Puppo and R. Scopigno. Simplification, LOD, and Multiresolution - Principles and Applications. In *EUROGRAPHICS'97 Tutorial Notes (ISSN 1017-4656)*. Eurographics Association, Aire-la-Ville (CH), 1997 (PS97 TN4).

[22] M. Reddy. Scrooge: Perceptually-driven polygon reduction. *Computer Graphics Forum*, 15(4):191–203, 1996.

[23] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum (Eurographics'96 Proc.)*, 15(3):67–76, 1996.

[24] J. Rossignac and P. Borrel. Multi-resolution 3D approximation for rendering complex scenes. In B. Falcidieno and T.L. Kunii, editors, *Geometric Modeling in Computer Graphics*, pages 455–465. Springer Verlag, 1993.

[25] Worley S. A cellular texture basis function. In *Comp. Graph. Proc., Annual Conf. Series (Siggraph '96)*, pages 291–294. ACM Press, 1996.

[26] D. Schikore and C. Bajaj. Decimation of 2D scalar data with error control. Technical Report CSD-TR-95-005, CS Dept., Purdue University, 1995.

[27] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *ACM Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[28] Marc Soucy, Guy Godin, and Marc Rioux. A texture-mapping approach for the compression of colored 3d triangulations. *The Visual Computer*, (12):503–514, 1996.

[29] L. Williams. Pyramidal parametrics. *Computer Graphics (SIGGRAPH '83 Proceedings)*, 17(3):1–11, July 1983.

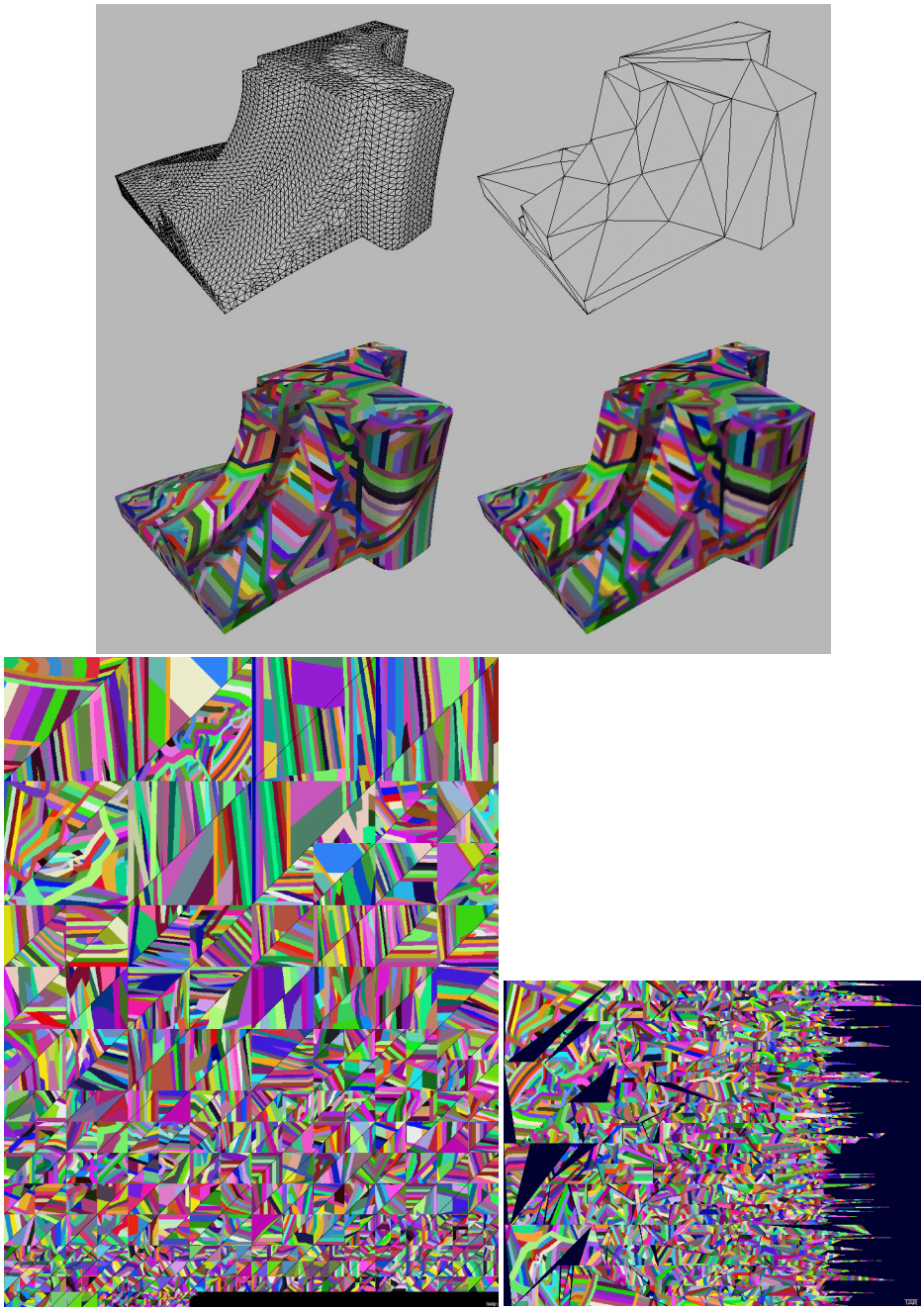

Figure 9: Mapping color from a *per-vertex* colored mesh: the original mesh on the left (12,946 faces) has been simplified (98 faces, top right), and then the original color info is mapped on the simplified mesh (center right). Texture maps produced with the Soucy method (lower left) and our method based on the use of irregular texture patches (lower right)