IGT: Inverse Geometric Textures

Ismael García* Gustavo Patow†
Geometry and Graphics Group, Universitat de Girona

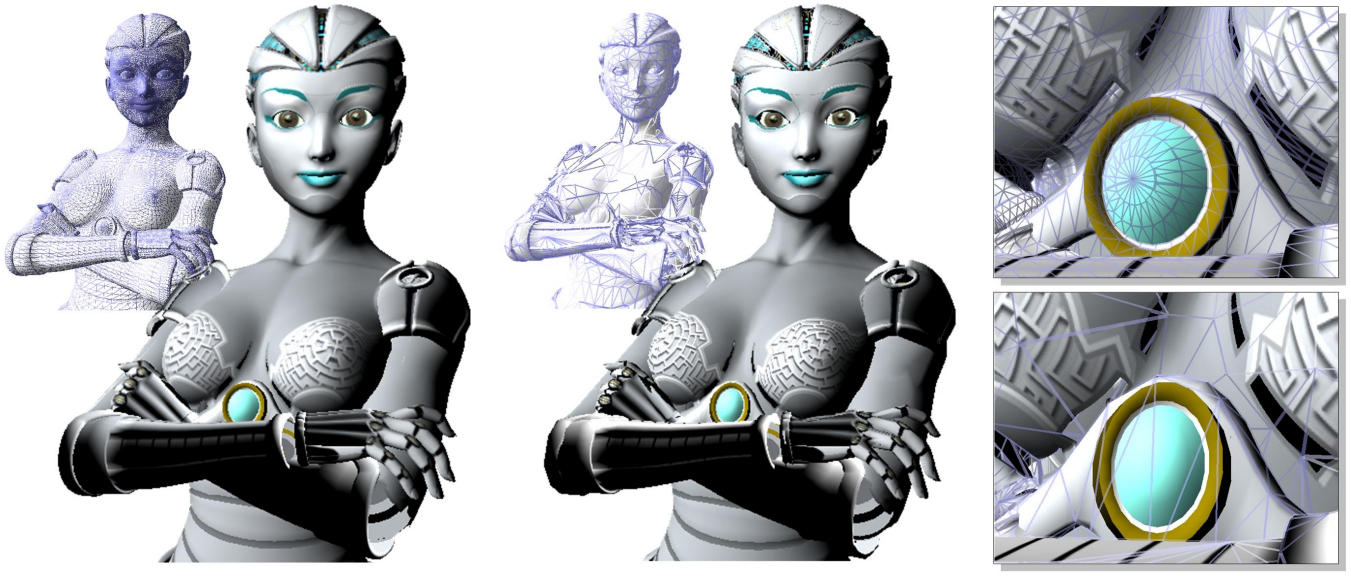

Figure 1: *(left) An artist-created model fully rendered and in wireframe (197150 triangles, 62 fps). (middle) The strongly simplified model (35834 triangles -82% reduction-, 165 fps) rendered with our technique. (right) Notice in the inset, that despite a drastic simplification, the global appearance is maintained from the original model (top) to the simplified one (bottom).*

Abstract

Preserving details from a high resolution reference model onto lower resolution models is a complex, and sometimes daunting, task as manual intervention is required to correct texture misplacements. Inverse Geometric Textures (IGT) is a parameterization-independent texturing technique that allows preservation of texture details from a high resolution reference model onto lower resolutions, generated with any given simplification method. IGT uses a parameterization defined on the reference model to generate an inversely parameterized texture that stores, for each texel, a list with information about all the triangles mapped onto it. In this way, for any valid texture coordinate, IGT can identify the point and the triangle of the detailed model that was projected, allowing details from the reference model to be applied onto the fragment from the low-resolution model. IGT is encoded in compact data structures and can be evaluated quickly. Furthermore, the high resolution model can have its own independent *artist-provided*, unmodified parameterization, so that no additional effort is required to directly use artist-designed content.

Keywords: Appearance preserving simplification, Detail-recovery, Computer games, Texturing, Parameterizations, LoD

1 Introduction

Modern interactive applications (e.g., computer games) need many instances of detailed geometric and multi-textured models provided by artists to populate vast scenes (see Figure 1). Thus, interactive applications must deal with level of detail (LoD) techniques, which should preserve the visual quality of the original model. Furthermore, models should have a compact representation and provide efficient visualization in a way that preserves high-definition details at close view.

The texturing detail reproduction of a high resolution mesh on a simplified model has proven to be a complex problem, often needing manual user intervention. This usually implies individual parameterizations, extra textures and normal maps, unless specific feature-preserving algorithms are used in the simplification.

Contributions: The key contribution of IGT is to decouple mesh simplification from texturing parameterization. This is done by an inverse parameterization that allows access to the original high resolution mesh encoded in a texture. Its main features are:

*e-mail: igarcia@ima.udg.edu
†e-mail: dagush@ima.udg.edu

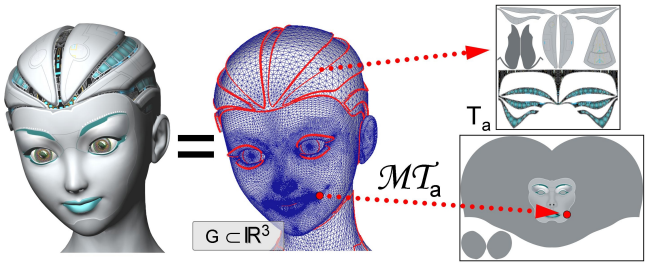

Figure 2: *In general, artists provide a high resolution reference mesh with its own parametric coordinates, a texturing parameterization $\mathcal{MT}_a$ and textures with rich surface details.*

- The artist is free to choose any texturing parameterization for the model, because it is not affected by IGT.
- An artist-provided model can be simplified using any algorithm, resulting in a higher quality mesh with correct textures for all LoDs.
- When rendering a lower resolution model, the original high resolution information can be retrieved and used for applications like color and texture preservation, normal mapping and lighting, among others.
- As color and texturing information is correctly preserved between different LoDs, IGT allows changes in LoD much earlier and with lower resolution models than other techniques.
- It is GPU-friendly, allowing a fast and compact implementation.

1.1 Related Work

IGT is a technique closely related to attribute-preserving simplification methods, and thus related to both texturing parameterizations and simplification methods. The survey by Hormann et al. [2007] gives complete detailed information on general texturing parameterizations.

Texturing Parameterizations: In general, there are two big families of texturing parameterizations: those that introduce discontinuities in the parameterized object, and those that do not, called *seamless* parameterizations. IGT can work smoothly with both kinds.

Among the first kind, some approaches consider larger patches and try to make the texturing parameterization per patch as conformal or area-preserving as possible, to avoid stretching the texture differently on different triangles of the model [Lévy et al. 2002] [Sander et al. 2003] [Sorkine et al. 2002]. The technique we present here can be successfully used with a high resolution reference model, which in general is parameterized by an artist with any of these methods (see Figure 2).

Examples of seamless texturing parameterizations that have been studied are cubes, spheres, cylinders, simplicial complexes and periodic planar regions with transition curves [Hormann et al. 2007]. For general objects, seamless parameterizations can be built only for texture domains that have the same topology and are close in shape to the target object. One powerful method for this is the PolyCubeMap [Tarini et al. 2004], which is a mechanism that allows a seamless texturing parameterization of a 3D mesh. A polycube is a 3D shape composed of many unit-sized cubes attached face-to-face that is used as the texture domain. These parameterizations have the advantage of enabling the use of most simplification methods, something which can be exploited in combination with IGT to improve both quality and development speed.

Unconstrained Mesh Simplification: Hoppe [1996] introduced a simplification procedure to construct a progressive mesh representation from an arbitrary mesh. Later, Garland and Heckbert [1997] presented a fast incremental method that applies an edge collapse operator guided by a measure based on quadric error metrics. More recently, Lee et al. [2005] introduced the idea of *mesh saliency* as a measure of regional importance for meshes. None of those methods aim at preserving surface attributes, so they are not very useful for textured models, but could be combined with IGT and a seamless parameterization to provide high quality results.

Attribute-preserving simplification: The first general approach for detail-recovery was presented by Cignoni et al. [1998] [1999], using resampled textures to decouple attribute detail representation from geometry simplification. There, preservation is performed after simplification by building a set of triangular texture patches that are then packed into a single texture map. Later, Tarini et al. [2003] presented a method to produce a normal-map by computing per-texel visibility and self-occlusion information. Carr and Hart [2002] used a similar approach for procedural texturing. Techniques like those used by Policarpo et al. [2005] render detailed geometric appearance on low polygonal models. All of them require resampling the high resolution model into an attribute texture. Chen and Chuang [2006] considered the adaptation of textures for progressive meshes. In general, most of the techniques mentioned also require a new texture for each simplification step. IGT outperforms their results because it directly queries the reference model, without the need of over-sampling or using multiple textures for different levels.

Some methods have been proposed to simplify polygonal meshes while preserving color and texture. Garland and Heckbert [1998] introduced a quadratic error metric for measuring vertex-to-plane distances. Sander et al. [2001] took into account the texture stretch and texture deviation introduced by edge collapses. Cohen et al. [1998] constructed a multi-resolution model with a simplification sequence that constrains the simplification of the patch boundaries. This is also done by the commercial package Polygon Cruncher [Mootools 2007]. Unfortunately, those methods either do not simplify seams, which results in a lower quality mesh, or generate triangles that might span the texture space outside any chart. IGT could be used to help simplify these meshes, but the simplification should be made with models with as few seams as possible in order to get better quality results.

Lacoste et al. [2007] adaptively sampled and encoded the normal field of the high resolution model in an octree. IGT also encodes the model into texture space, but uses a projection and data structures that prove much more efficient in generality (e.g., two-sided faces are admitted), storage and speed of evaluation. Also, IGT is less restrictive with respect to the simplification method used.

1.2 Overview

Given the texture coordinates of point δ_s from a simplified low resolution geometric model G_s, IGT is a technique that defines an inverse mapping $\mathcal{I}$. This inverse mapping allows us to find the corresponding point δ from the high resolution reference mesh G, and to retrieve its attributes, e.g. color and shading information. IGT requires a *primary* parameterization on the reference model to generate an inverse parameterized texture, and stores for each texel a list with information about all triangles that are mapped onto it. This way, IGT can identify the point and the triangle of the detailed model that was projected for any valid texture coordinate. This allows application of details from the reference model onto the frag-

ments of a low-resolution model which can be simplified so that the parameterization is preserved. If needed, the high resolution model can also have another independent *artist-provided* parameterization to be able to directly use the original input textures.

2 Inverse Geometric Textures (IGT)

In this section we will define the spaces and mappings needed for IGT, as well as describe its construction process and usage. Subsequently, we will provide a generalization of its definition, data structures needed, shader LoD and filtering.

Now, we present a simplified version of the notation of texture spaces and mappings needed to define IGT and its main features. We define the object space of vertex positions as $\mathbb{R}^3$, and textures as belonging to a space $\mathbb{T}^2$. The space $\mathbb{T}^2$ is the space referred to by the typical (u, v) texture coordinates associated with every vertex in a model. In general, a texturing *primary* parameterization assigns a texture position τ to a point δ in 3D space using a mapping $\mathcal{MT}$. See Figure 3, where the head of the Aikobot model was parameterized with a simple cylindrical parameterization $\mathcal{MT}$, and the helmet with a spherical one. The full resolution mesh, denoted by G, and simplified versions used as levels of detail, denoted by G_s, are in $\mathbb{R}^3$. When texture coordinates for G_s are preserved after the simplification process, we can safely assume that there exists for each point in G_s a texture coordinate in $\mathbb{T}^2$. Thus, the primary parameterization $\mathcal{MT}$ is also used to map the simplified mesh to IGT. For instance, the Aikobot head is firstly mapped from G to a cylinder with a mapping $\mathcal{T}$, and the result is used to index a texture by a mapping $\mathcal{M}$ ($\mathcal{MT} = \mathcal{M} \circ \mathcal{T}$). See Section 5.

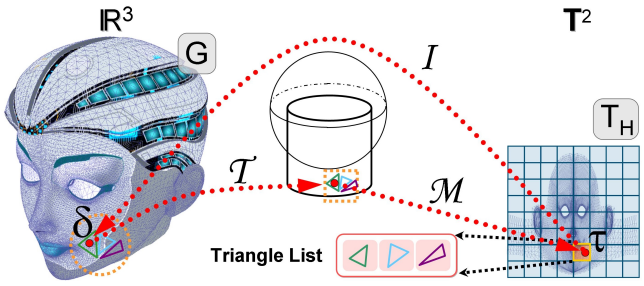

Figure 3: *Different spaces used for texture mapping, and IGT construction: Given an initial geometry $G \in \mathbb{R}^3$ (left), it can be parameterized in different ways. IGT uses a parameterization $\mathcal{MT} = \mathcal{M} \circ \mathcal{T}$ to store lists of triangle information from G in 2D texture space $\mathbb{T}^2$. The mappings $\mathcal{T}$, $\mathcal{M}$ and $\mathcal{I}$ are also represented in the figure.*

Each vertex in G has a texture position in $\mathbb{T}^2$ associated to it by means of $\mathcal{MT}$. At rendering time, the texture coordinates of each vertex are interpolated and each fragment δ receives a texture position τ to fetch its attributes, like color or shading information. For convenience, let's assume that $\mathcal{MT}$ also projects $\delta_s \in G_s$ to τ. The function $\mathcal{MT}$ completely defines the *primary* parameterization. Artists usually provide models together with another, often manually-defined, texturing parameterization $\mathcal{MT}_a$ (see Figure 2).

Under this simplified description, IGT is an inverse mapping $\mathcal{I}$ that establishes a bijection between the points in $\mathbb{T}^2$ to the respective points in $\mathbb{R}^3$, so that we know which point in $\mathbb{R}^3$ projects onto any given point in the texture. To be effective, IGT stores all the information of $\mathcal{I}$ in a texture T_H defined in texture space $\mathbb{T}^2$, where each point τ contains a reference to the point δ in $\mathbb{R}^3$ that maps onto it.

The texture $T_H \in \mathbb{T}^2$ is discrete, i.e. made of texels of a finite area. Thus, each texel of T_H contains a *list* of the triangles that

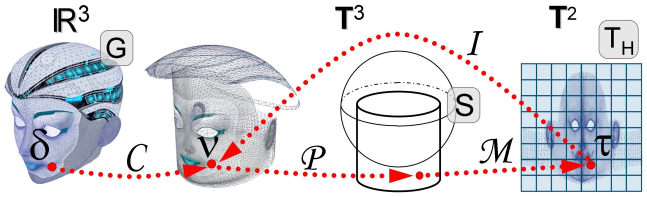

Figure 4: *General setting: Given an initial geometry $G \in \mathbb{R}^3$ (left), texturing parameterizations have an associated 3D texture space $\mathbb{T}^3$ and a 2D texture space $\mathbb{T}^2$, which encodes the attributes. Now, $\mathcal{MT} = \mathcal{M} \circ \mathcal{P} \circ \mathcal{C}$.*

are mapped onto the area of the texel. We do this by storing a list of *triangle identifiers* at each texel in T_H. Given a point τ in T_H, retrieving its 3D back-projected point simply means verifying which of the triangles in the corresponding list contains that point.

Certain parameterizations, like PolyCubeMaps [Tarini et al. 2004], cannot be explained with the spaces used in previous sections. In order to be more generic, and better explain some IGT features, we need an intermediate texture space $\mathbb{T}^3$, which is a 3D generalization of the traditional space of (u, v) texture coordinates. See Figure 4. A point $\delta \in \mathbb{R}^3$ is mapped to a point $\nu \in \mathbb{T}^3$ with a mapping $\mathcal{C}$. In this space, a projection $\mathcal{P}$ is used to project the point ν onto a surface S in $\mathbb{T}^3$: the cylinder for a cylindrical projection, or the polycubes for PolyCubeMaps. With respect to our previous notation, $\mathcal{T} = \mathcal{P} \circ \mathcal{C}$. The projection $\mathcal{P}$ and the posterior mapping $\mathcal{M}$ basically map a point in $\mathbb{T}^3$ onto the texture space $\mathbb{T}^2$ by defining a line in $\mathbb{T}^3$ that passes through the point and intersects the surface S. To be precise, IGT represents the inverse mapping $\mathcal{I} = (\mathcal{M} \circ \mathcal{P})^{-1} : \mathbb{T}^2 \rightarrow \mathbb{T}^3$, see Figure 4. Observe that, actually, we need to store information from $\mathbb{T}^3$, *not* from $\mathbb{R}^3$. One requirement of IGT is that the projection $\mathcal{P}$ must be bijective.

It is important to remark that the original *artist-provided* parameterization is *not* affected at all by the usage of IGT, so the artist is free to choose *any* parameterization method he/she likes.

2.1 Building IGT

The generation of the data structures needed for IGT is performed in an off-line pre-processing stage. The process starts with a high resolution reference model G adequately parameterized with $\mathcal{MT}$ and $\mathcal{MT}_a$. Then, the triangles of G are projected and mapped with $\mathcal{MT}$ onto the texture T_H. Every texel in T_H is annotated with the list of all triangles whose projection/mapping covers it totally or partially (see Figure 3). As now the texture T_H stores references to lists of triangle identifiers, it plays the role of a *spatial hash* that allows the querying of a minimum number of list entries each time a point τ must be inversely parameterized. This texture is called the *hash texture*. The lists are stored in a second texture called the *list texture*, texture coordinates of the triangles (from $\mathcal{MT}$) are stored in the *triangle texture*, and their attributes (e.g., texture coordinates for the *artist-provided* parameterization, ambient occlusion term, etc.), are encoded in the *attribute texture*. If needed, all textures can be packed into a single atlas.

The construction of the lists is done by verifying every triangle from G, and generating a list entry in a conservative manner, even if the triangle slightly touches the texel. Firstly, the triangles in G are mapped to the *hash texture* in $\mathbb{T}^2$, and are checked for intersection with the texel area. If the intersection is not empty, then a new entry is generated and added to the respective list. The whole process only takes between a few seconds and a couple of minutes even for the most sophisticated examples we have tried.

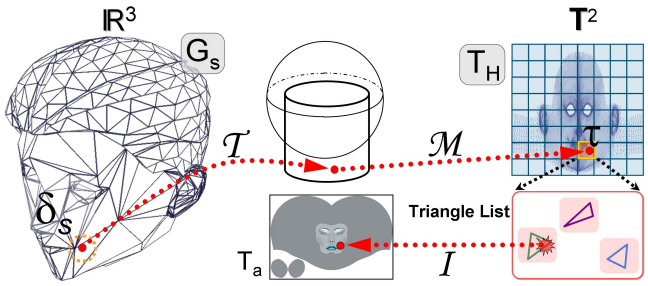

Figure 5: *Given point $\delta_s \in G_s$, mapping $\mathcal{MT}$ transforms it to point $\tau \in \mathbb{T}^2$, where the texel list is retrieved and its triangles evaluated, finding the back-projected point that allows querying the artist-provided texture T_a.*

2.2 Using IGT

Given a point δ_s from G_s, IGT is used to retrieve information from the reference model G. As long as δ_s can be projected/mapped onto $\mathbb{T}^2$, we use the projected point τ to find the corresponding point from the reference mesh G, as explained before. Actually, τ is the projected point in $\mathbb{T}^2$ for both δ_s and δ. The only real requirement imposed onto the point δ_s is that it must be mapped onto $\mathbb{T}^2$.

When rendering a low resolution model G_s, fragments δ_s are generated, in such a way that they can be inversely parameterized to retrieve the point δ from the reference model. In fact, computing τ is needed only to fetch the right list from the hash texture. To determine the intersection point, we implemented this with a simple point-in-triangle code for orthogonal projections, and a standard ray-triangle intersection code [Lofsted and Akenine-Moller 2005] for the other projections. Once the right intersection point and triangle are found, we can fetch any information originally associated to the reference model G and apply it to shade/compute point δ_s (see Figure 5). Applications of this can be found in Section 3.

In the general setting, intersections cannot be computed in $\mathbb{T}^2$, as a triangle in $\mathbb{T}^3$ can be projected onto disjoint parts in $\mathbb{T}^2$ (as in PolyCubeMaps). Hence, we generate the line in $\mathbb{T}^3$ used by $\mathcal{P}$ to project the point τ, and compute its intersection with the triangles in the list. If the parameterization is *bijective*, this intersection exists and the intersected point is ν.

2.2.1 Decoupling Parameterization from Simplification

One of the key points of IGT is that it allows the decoupling of texturing parameterization and model simplification. If needed, the reference model G stored in IGT may have another, *artist-provided* parameterization $\mathcal{MT}_a$ for texturing purposes, which is completely independent of the *primary* parameterization $\mathcal{MT}$ used for the simplification. When rasterizing a low resolution model, in order to guarantee that the generated fragments can be mapped onto T_H, the simplification method should only preserve the *primary* parameterization, and not the *artist-provided* one.

An example can be seen in Figure 5, where the Aikobot head model G was textured with cylindrical and spherical maps as the *primary* parameterization $\mathcal{MT}$, and a LSCM multi-chart method as the *artist-provided* parameterization $\mathcal{MT}_a$. A fragment δ_s, generated by rendering the low resolution model G_s, is projected with $\mathcal{MT}$ and the corresponding list is fetched and evaluated, finding its back-projected point and the respective triangle from the reference model G. Then, by using $\mathcal{MT}_a$ with δ, the shading information from the reference model G can be fetched and used.

2.3 Data Structures and Evaluation

IGT can be encoded in highly compact data structures. In particular, it can be encoded in four textures that can be combined, according to need, into one single texture atlas.

- The hash texture is usually a small RGBA8 texture, with dimensions ranging from 64×64 to 256×256. Each texel encodes in its first three bytes the location of the corresponding list in the list texture. The fourth byte is used to store the length of the list, with 0 representing an empty list.
- The list texture is encoded into another RGBA8 texture, where each list is consecutively stored. Whenever a list should be split, its position is moved to the beginning of the next line. As lists are usually short, about three elements long (See Section 4), the wasted space is only about 1.5%.
- The triangle texture consecutively stores texture coordinates in $\mathbb{T}^3$ for each of the three vertices of the triangle. Usually, this means only storing the (u, v) coordinates, although some parameterizations could require an extra channel.
- If needed, an attribute texture can be created with extra information, like triangle color or the *artist-provided* parameterization coordinates.

With respect to the computational cost, IGT involves the evaluation of the hash texture, which represents one texture fetch, and three extra fetches in the triangle texture for each entry in the list texture. On average, we found that three entries in the List Texture are usually needed, so the effective average cost of IGT is of ten texture fetches in total, plus the fetches on the *attribute texture*. For example, solid color represents a single fetch, but for the coordinates of the *artist-provided* parameterization $\mathcal{MT}_a$, three extra fetches are needed.

2.4 Shader LoD

IGT is a technique that should be used from close to large distances, as its evaluation is more complex than direct texturing techniques. We have implemented a shader LoD technique which changes the shader as soon as the difference with a simpler texel fetch can be disregarded. When the observer cannot distinguish one from another, we swap to a simpler shader that only fetches the relevant information from a traditional texture in $\mathbb{T}^2$ space, using only the *primary* parameterization $\mathcal{MT}$. For further distances, MIP maps for this texture can be used, resulting in a smooth minification of the texture details.

2.5 Filtering

In modern GPU applications, the filtering issue is of great importance. IGT is also able to perform correct filtering; the solution implemented consists of retrieving the nearest samples (five in our case) from the hashed data and then blending them. The only difference with respect to the non-filtered case is that the mip-map level selection must be set so that it is based on the change of the fragment texture positions in $\mathbb{T}^3$ (using functions *dFdx* and *dFdy* in GLSL or *ddx* and *ddy* in Cg), rather than the simple texture position in $\mathbb{T}^2$. Also, when fetching the artist-generated texture from the *artist-provided* parameterization, normal filtering and mip-mapping can be used. This solution is used by several authors [Lefebvre and Hoppe 2006] [Nehab and Hoppe 2007] [Tarini et al. 2004]. We implemented this, finding the same factors as pointed out by Lefebvre and Hoppe [2006]: about 3.9 for going from one sample to four samples per pixel.

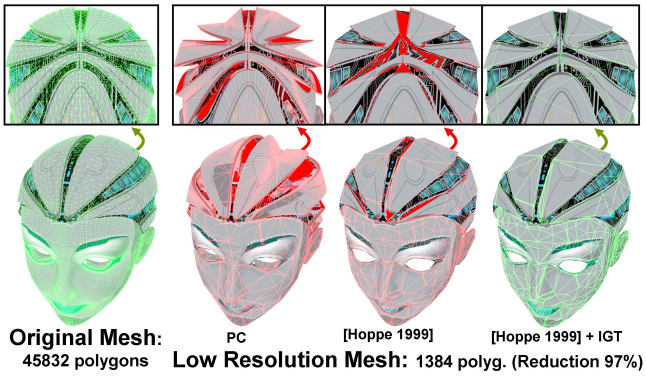

Figure 6: *IGT allows simplification of a textured model (left) with the most convenient method (other columns). Polygon Cruncher (PC, second column) cannot simplify to more than 2780 triangles. In the insets, a rear view of head details can be observed (1384 triangles, 450 fps).*

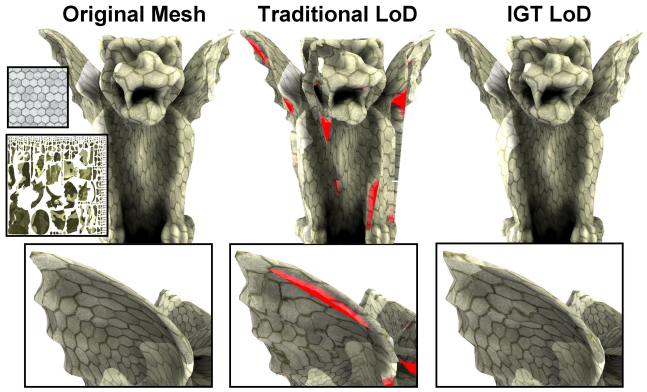

Figure 7: *Tiling textures with IGT: from left to right, the original model (131072 triangles), the model simplified with Polygon Cruncher (only up to 15738 triangles), and the model simplified as per Hoppe [1996] combined with IGT (3280 triangles, 415 fps).*

3 Advantages of IGT

IGT is a technology that allows the decoupling of texturing and simplification methods, easing the work performed by developers once the modeler has delivered a high resolution, parameterized reference model G. Information can be associated to a reference mesh at three different levels: at the triangle level, as when dealing with constant colors; at the vertex level, as with normal mapping or per-vertex ambient occlusion; and at the level of textures mapped onto the high resolution model. IGT is a versatile technique that provides smooth retrieval of information at these three levels.

- **Texturing, Normals and Simplification:** As mentioned, one of the key points of IGT is its ability to decouple texturing from simplification. For instance, in Figure 6, the head of the Aikobot model was independently parameterized and textured using multi-charts as $\mathcal{MT}_a$. Simplification with traditional techniques results either in mixed textures or lower quality meshes due to the seam preservation constraint. As before, IGT uses a cylindrical *primary* parameterization $\mathcal{MT}$ for the head, and a spherical one for the helmet. The combination of those seamless parameterizations with simplification methods leads to a simplified model with a high quality mesh and correctly preserved textures. Also, with IGT, tile-based textures can be used, as the indirection provided by IGT can be used to reference a highly defined texture repeated over the entire surface, as shown in Figure 7. Also, it is a common practice to generate a normal map so that a lower resolution model with an applied normal map resembles the original reference model. With IGT we can have two-level normals: one set obtained directly from the triangles of the reference model, and one applied as a regular normal map for the fine details taken directly from the texture coordinates of the reference model. See Figure 1.

- **Texture Transfer:** For an application that does not require dynamically changing LoD, one can map details to a simplified mesh by simple projection (e.g., based on coarse mesh normals, as done by Tarini et al. [2003]). However, IGT provides a more natural way of doing so, as it allows the transfer of details between different parameterizations at any simplification level, without requiring a high resolution texture to store the details. As an example, IGT used in combination with polycubes requires much less storage memory than the original PolyCubeMaps alone, and with a superior quality.

- **Solid Boundaries:** As mentioned in Section 1.1, feature preserving simplification methods tend to mix colors at the boundaries of triangles with different solid colors. IGT provides automatic multiresolution details, allowing the preservation of sharp solid boundaries between colors or texture borders. For example, in Figure 8 we can see a comparison of some simplification methods used with and without IGT. In this case we kept the original colors assigned by the artist at the vertex level in the *attribute texture*: there is no need to define an *artist-provided* parameterization $\mathcal{MT}_a$, as colors can be directly retrieved from the reference model.

- **Volumetric and Procedural Texturing:** Volumetric functions [Kajiya and Kay 1989] are usually point-wise evaluated on the surface of the reference model G. For lower resolution models G_s the simplified triangles span a different region of the volume, producing noticeable changes in appearance. To our knowledge, the only solution for consistently applying a volumetric texture onto G_s is to sample it on G transferring the result to a texture, and use it for the successive LoDs [Carr and Hart 2002]. IGT solves this problem, as only information from G is used to texture all those of low resolution (See Figure 9). IGT also allows a smooth integration of effects like traditional 2D animation, animated procedural textures (e.g. reaction diffusion), or an animated volume (e.g. smoke), with simplified models without any extra information other than the original animation information and IGT itself.

- **Animation:** IGT can be used to improve mesh quality for animation. The LoD being visualized can be animated and, as IGT works entirely in texture space, it will work seamlessly as long as texturing information is not modified by the process. IGT can even prove beneficial in cases where the use of traditional simplification techniques results in a LoD with insufficient triangles in joints with large stretching. See Figure 10.

4 Experimental Results

In Table 1, results for different combinations of input models and parameterizations are presented. In particular, we have implemented combinations of cylindrical and spherical mappings, Least Squares Conformal Map (LSCM) multi-charts [Lévy et al. 2002], Angle Based Flattening (ABF++) multi-charts [Sheffer et al. 2005], Iso-Charts [Zhou et al. 2004], PolyCubeMaps (PCM) [Tarini et al.

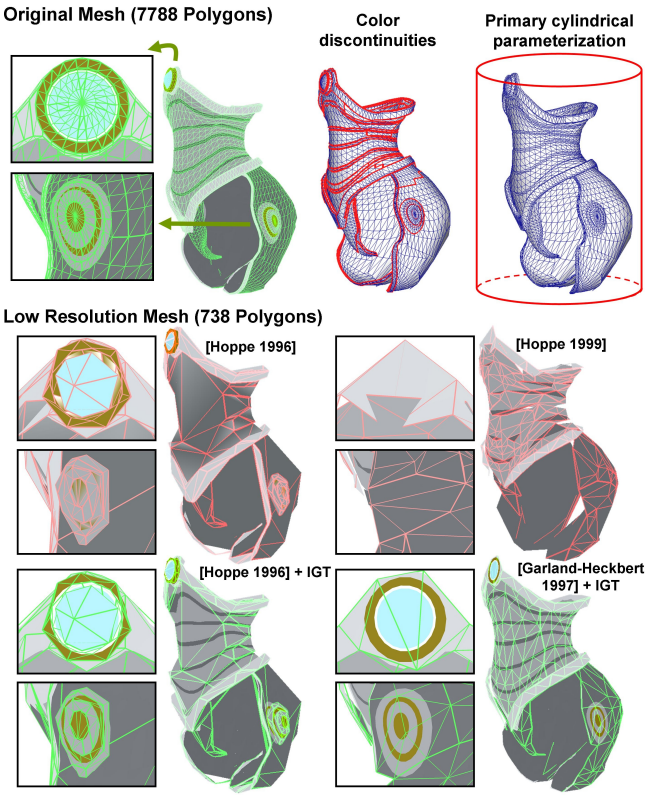

Figure 8: *Solid color preservation: the Aikobot body armor model in Figure 1 has artist-painted colors directly at the vertex level. With traditional methods, either the colors or the mesh quality suffer, while IGT allows their preservation (738 triangles, 650 fps).*

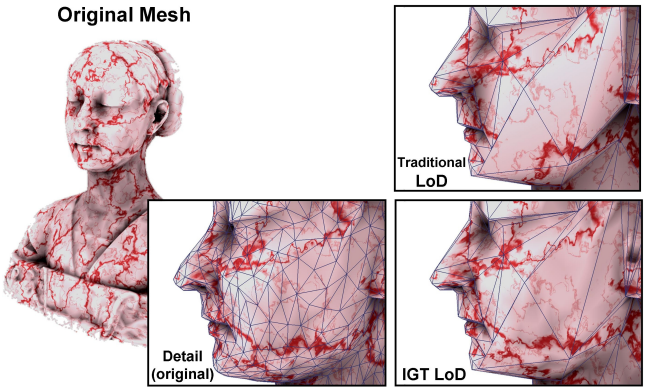

Figure 9: *IGT preserves shape and appearance of solid textures, while direct texture application onto different LoDs [Hoppe 1996] leads to non-preservation of the features (304 triangles, 535 fps).*

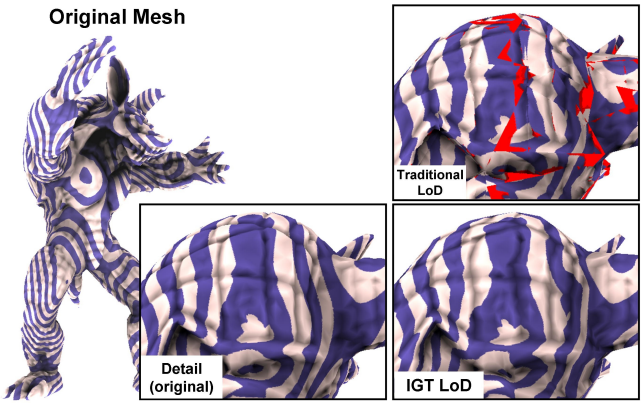

Figure 10: *IGT is compatible with animation of LoD models. We can see the model in an animation posture, and detailed views of the back when simplified with and without IGT. All models have normals mapped from the high resolution model (30000 triangles) to the lower resolution ones, generated with Hoppe [1999] (974 triangles, 410 fps).*

2004], and Geometry Images (GI) [Gu et al. 2002]. From this table, we can see that the storage needs of IGT are small, requiring less than a medium-resolution 1024^2 normal map (RGBA8 encoded, 4 MB). In the examples, the *attribute texture* stored the extra information for shading. We have found, in our examples, that lists stored in the *list texture* are short (three triangles on average). However, this can vary for an irregular triangle density. Using current available graphics hardware, a limit imposed by the size of a single texture map limits IGT to work with models up to 7 million triangles, unless paging strategies are used. As mentioned above, IGT does not need to store the original triangles in $\mathbb{R}^3$, but only the associated texturing coordinates in $\mathbb{T}^3$, which usually results in storage of (u, v) pairs, reducing one third of the storage. Only parameterizations that require full 3D information, e.g., PolyCubeMaps, would need to store (u, v, w) triplets. Also, depending on the precision needs, storage can be further reduced by using fixed point arithmetic, like OpenGL 2.0 RGB9E5, which is a standard 4-byte/texel format.

In Figure 1, we can see the Aikobot model, which is an artist-created model provided with a multi-charts parameterization $\mathcal{MT}_a$. In that model, there are lots of texture discontinuities, posing a serious problem to simplification methods that try to preserve texturing. As *primary* parameterization $\mathcal{MT}$, i.e. the one used for simplification purposes, we have successfully applied a spherical mapping to the helmet, a cylindrical mapping to both the body armor and the head, and iso-charts for the body, as seen in Figures 6 and 8. As the parameterizations for helmet, head and body armor are seamless, simplification algorithms like the ones used by Hoppe [1996] and Garland and Heckbert [1997] were applied. For the rest of the body, the simplification method was forced to preserve the seams in the parameterization [Mootools 2007].

We compared the performance of IGT with respect to rendering the full resolution model and traditional LoD techniques [Mootools 2007] (see Figure 11). In all cases, LoD changes were made with *late switching*, where the replacement between LoDs is made when the visual difference is no longer noticeable. It is important to mention that IGT allows this replacement to be done much earlier than other techniques. If needed, a better replacement strategy is described by Giegl and Wimmer [2007]. The results show that IGT outperforms the rendering of the reference model, even if no LoD changes are made. Renderings were made at 1024×768, on a quad core Pentium IV with a GeForce 8800 card.

We can get information of the overhead incurred when using IGT by making a comparison between models at the same frame rates. Results at fixed frame rates for a model simplified using traditional techniques, with a mesh simplified with details mapped by simple normal projection using [Orgaz 2007] and with IGT, are shown in Table 3. The overhead introduced by IGT is small, reinforcing the idea that IGT can present models with less overhead much earlier than other techniques. Also, it must be taken into account that there

Reference Models	Polyg. Count (triangles)	IGT Param. (Primary)	Artist Param. (Secondary)	Hash Texture. (RGBA8)	Lists Tex. (RGBA8)	Trangles Tex. (RGB9E5)	Attrib. Tex. (RGB9E5)	Memory usage (MB)
Armadillo	30000	PCM	LSCM	128x128	512x256	512x256	512x256	1.56
Gargoyle	131072	GI	LSCM	128x128	512x256	512x512	512x512	2.56
SpaceShip Room	84140	LSCM	LSCM	128x128	512x512	512x480	512x480	2.94
SpaceShip	15338	ABF++	LSCM	64x64	256x256	256x128	256x128	0.52
Laurana	10000	PCM	Procedural Tex.	128x128	512x256	256x128	512x128	0.94
Bunny	15000	PCM	LSCM	128x128	256x256	256x256	256x256	0.81
Aikobot Armor	7798	Cylindrical	—	128x128	128x128	128x128	128x128	0.25
Aikobot Helmet	6628	Spherical	LSCM	128x128	256x128	256x128	256x128	0.44
Aikobot Face	39204	Cylindrical	LSCM	128x128	512x256	512x256	512x256	1.56
Aikobot Body	49858	Iso-charts	LSCM	256x256	512x320	256x480	256x480	1.81

Table 1: *Memory usage for various examples. Acronym meanings are: LSCM, Least Squares Conformal Maps; PCM, Polycube-maps; ABF++, Angle Based Flattening; GI, Geometry Images; RGB9E5, a standard 4-byte/texel OpenGL 2.0 format.*

Hash	AvgLength	Framerate	TotalMem
128x128	8.2	110fps	1.50MB
256x256	5.3	260fps	1.81MB
512x512	3.1	343fps	3.81MB
1024x1024	2.6	400fps	8.94MB

Table 2: *Dependence of the average list length, total memory consumption and frame rate on the hash resolution. All values refer to the Aikobot body in Figure 8.*

Distance	PC	IGT	NP	Framerate	Ratio
10.59	6628	3519	6628	45	1.9
26.51	6628	3642	6628	21	1.8
50.05	6628	5370	6628	16	1.2
139.95	6628	6628	6628	19	1.0

Table 3: *IGT Overhead (columns from left to right): distance in arbitrary units from the observer, number of triangles for a reference model simplified with Polygon Cruncher (PC), for IGT, for textures transferred by normal projection (NP), frame rate and overhead ratio. Values refer to the Aikobot head in Figure 6.*

are three factors to compare: quality, memory and speed. At equal frame rates, it can be seen that IGT provides the same quality with lower storage needs.

Finally, it is important to mention the influence of the hash size in the requirements and performance of IGT. As expected, as the hash increases resolution, lists for each texel will get shorter on average, but at an increased memory cost. For certain texels, however, this length has a lower bound, as any sized texel that covers a vertex shared by, for instance, six triangles will have (at least) a length of six entries. On the other hand, reducing list average length means fewer evaluations at the pixel shader, resulting in an improved rendering speed. This can be seen in Table 2 for the Aikobot body.

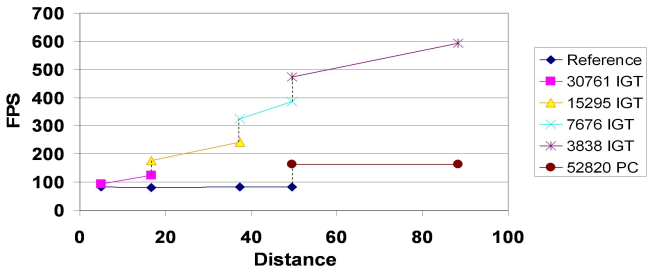

Figure 11: *Frame rates of IGT when compared with G (125932 triangles) and traditional LOD. Distances are in arbitrary units and curves are named after the number of triangles and the technique used (PC stands for simplification with Polygon Cruncher).*

5 Discussion and Limitations

From the above results, it can be seen that IGT outperforms traditional methods in speed and quality. This is basically because IGT sends many times fewer triangles through the pipeline, moving the workload to the pixel shaders. As a model covers less and less pixels, less pixel shaders are needed, thus resulting in a smoothly improved speed, which can be inferred from the slope in the curves for a given LoD with IGT in Figure 11. Traditional simplification techniques, which produce models without requiring special shaders, need to process a constant number of triangles for longer distance intervals, making the curves quite flat.

There is one important aspect of our technique that must be mentioned: although IGT is independent from both the texturing parameterization chosen and the simplification method used, it must be clear that simplification methods are not parameterization-independent. Multi-chart parameterizations, which are most commonly used by artists, produce seams, so they require specific constrained simplification algorithms to preserve those seams [Garland and Heckbert 1998] [Sander et al. 2001]. Therefore, the model cannot be simplified and retain a good quality at the same time, as the simplification algorithm (e.g. Polygon Cruncher [Mootools 2007]) must preserve the seams to avoid texture artifacts, simplifying chart interiors only (See Figure 12, first row, where the algorithm cannot simplify the model further than 16346 polygons, with a poor quality mesh). However, parameterizations generating few charts, like iso-charts [Zhou et al. 2004], allow a better simplification quality as there are fewer seams to preserve (Figure 12, second row). Without IGT, they can also introduce texture distortion and blurring if the artist textures are resampled to new ones (compare with the third row in Figure 12). IGT can be successfully combined using these as primary parameterizations, in combination with the original artist parameterization, providing almost perfect texture preservation and a significant improvement even for extremely simplified models. However, we can conclude that the best combination with IGT is using a seamless primary parameterization, such as spherical, cylindrical or PolyCubeMaps, that tolerates most simplification methods (like Garland et al. [1998] or progressive meshes [Hoppe 1996]) basically because the texture coordinates can be calculated from the original vertex coordinates for any LoD, without requiring a manual fine-tuning step (See Figure 12, fourth row).

IGT can encounter problems when applied to non-bijectively parameterized objects, where more than one triangle in $\mathbb{T}^3$ projected by $\mathcal{P}$ cover the same point τ in $\mathbb{T}^2$. In this case, when IGT is used to add detail to the reference model, the ambiguities can be easily solved just by looking up the right intersection in the list. However, when a lower resolution model generates a fragment that does not *exactly* coincide with any high resolution fragment, the ambiguity is unavoidable and some heuristic, like taking the closest fragment,

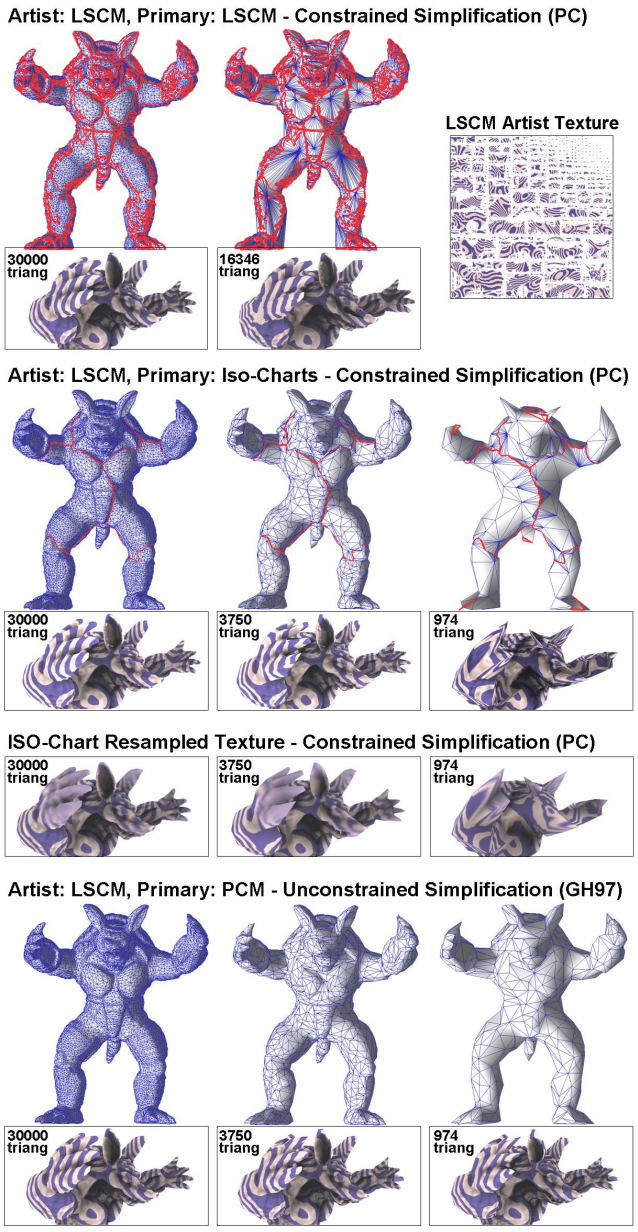

Figure 12: *Sensitivity of IGT to different primary parameterizations $\mathcal{MT}$. All models (except of the third row) use an LSCM artist-provided parameterization. First row: LSCM is also used as primary. Second row: Iso-charts are used as primary. Third row: the texture was resampled to an Iso-chart, and simplified. Fourth row: LSCM for artist and PolyCubeMaps (PCM) for primary. PC stands for Polygon Cruncher and GH97 for [Garland and Heckbert 1997].*

should be used. It must remain clear that this problem is not introduced by IGT, but by the chosen parameterization. Also, IGT is a *texturing* technique, and as usually happens with these techniques, silhouettes of a low resolution model cannot be hidden with IGT.

Finally, IGT is a technique that relies intensively on modern GPU capabilities, as it depends on the compromise between the fill rate (number of rendered triangles) vs. the power of pixel shaders to evaluate it. In our experiments we have found that IGT behaves very well in terms of speed when compared with rendering the full resolution model, providing excellent quality when compared with other LoD techniques.

6 Conclusions

IGT provides an inverse mapping from texture space to object space, and can be used to apply information generated for a reference model onto any simplified version. IGT allows the usage of a parameterization for simplification purposes that is *independent* of the parameterization provided by the artist for texturing. This way, parameterization and simplification are decoupled from each other. The best results are obtained if the *artist-provided* parameterization $\mathcal{MT}_a$, usually as a multi-chart parameterization, is used in combination with a seamless *primary* parameterization. For example, cylindrical, spherical maps, or polycubes, allow the user to choose a simplification method. Being parameterization-independent, IGT is compatible with both mip-mapping and filtering techniques as long as the *artist-provided* parameterization is compatible. IGT does not provide filtering by itself, but it enables the combination of advantages of different parameterizations in a way that was impossible before without a great deal of work (e.g., by transferring the texturing information to a texture).

This work opens three main areas of research. On the one hand, adding *geometric detail* like Porumbescu et al. [2005] seems a logical next step, as it would allow the addition of detail to the *reference* model, and not only to the simplified versions. On the other, introducing animation in the triangles of the reference model that are *not* on the lower quality models would allow interesting effects like approximate facial animation. Finally, geometry compression techniques that would allow even higher resolution models to be used with IGT should be studied.

Acknowledgements

We want to thank Ignacio Martin, Carles Bosch, Albert Mas, Tere Paradinas, Florent Duguet, Celine Loscos and Xavier Pueyo for proofreading different versions of this manuscript, Marco Tarini and Pere-Pau Vázquez for helping with code samples, Jordi Rovira and Álvar Vinacua for useful discussions, Vivien Greatorex for the voice-over, Fran González for all his crucial help and the anonymous reviewers for comments and constructive criticism. We are absolutely indebted to Pere Brunet for his advice during the whole process. Aikobot Maria Model from DAZ 3D, www.daz3d.com. This project was funded by grant TIN2007-67120 from the Spanish government.

References

CARR, N. A., AND HART, J. C. 2002. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graph. 21*, 2, 106–131.

CHEN, C.-C., AND CHUANG, J.-H. 2006. Texture adaptation for progressive meshes. *Computer Graphics Forum 25*, 3, 343–350.

CIGNONI, P., MONTANI, C., SCOPIGNO, R., AND ROCCHINI, C. 1998. A general method for preserving attribute values on simplified meshes. In *VIS '98: Proceedings of the conference on Visualization '98*, IEEE Computer Society Press, 59–66.

CIGNONI, P., MONTANI, C., ROCCHINI, C., SCOPIGNO, R., AND TARINI, M. 1999. Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer 15*, 10, 519–539.

COHEN, J., OLANO, M., AND MANOCHA, D. 1998. Appearance-preserving simplification. *Computer Graphics (Proc. SIGGRAPH) 32*, 115–122.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. *Computer Graphics (Proc. SIGGRAPH) 31*, 209–216.

GARLAND, M., AND HECKBERT, P. S. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *VIS '98: Proceedings of the conference on Visualization '98*, IEEE Computer Society Press, 263–269.

GIEGL, M., AND WIMMER, M. 2007. Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. *Computer Graphics Forum 26*, 1 (Mar.), 46–49.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. 21*, 3, 355–361.

HOPPE, H. 1996. Progressive meshes. *Computer Graphics (Proc. SIGGRAPH) 30*, 99–108.

HOPPE, H. H. 1999. New quadric metric for simplifying meshes with appearance attributes. In *IEEE Visualization '99*, D. Ebert, M. Gross, and B. Hamann, Eds., 59–66.

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice. In *SIGGRAPH 2007 Course Notes*, ACM, 1–122.

KAJIYA, J. T., AND KAY, T. L. 1989. Rendering fur with three dimensional textures. *Computer Graphics (Proc. SIGGRAPH) 23*, 271–280.

LACOSTE, J., BOUBEKEUR, T., JOBARD, B., AND SCHLICK, C. 2007. Appearance preserving octree-textures. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM, 87–93.

LEE, C. H., VARSHNEY, A., AND JACOBS, D. 2005. Mesh saliency. *ACM Trans. Graph. 24*, 3, 659–666.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, 579–588.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3, 362–371.

LOFSTED, M., AND AKENINE-MOLLER, T. 2005. An evaluation framework for ray-triangle intersection algorithms. *Journal of Graphics Tools 10*, 2, 13–26.

MOOTOOLS, 2007. Polygon cruncher. http://www.mootools.com/.

NEHAB, D., AND HOPPE, H. 2007. Texel programs for random-access antialiased vector graphics. Technical Report MSR-TR-2007-95, Microsoft Research.

ORGAZ, S., 2007. xnormal. http://www.xnormal.net/.

POLICARPO, F., OLIVEIRA, M. M., AND JO A. L. D. C. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, 155–162.

PORUMBESCU, S. D., BUDGE, B., FENG, L., AND JOY, K. I. 2005. Shell maps. *ACM Trans. Graph. 24*, 3, 626–633.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. *Computer Graphics (Proc. SIGGRAPH) 35*, 409–416.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Eurographics Association, 146–155.

SHEFFER, A., LÉVY, B., MOGILNITSKY, M., AND BOGOMYAKOV, A. 2005. Abf++: fast and robust angle based flattening. *ACM Trans. Graph. 24*, 2, 311–330.

SORKINE, O., COHEN-OR, D., GOLDENTHAL, R., AND LISCHINSKI, D. 2002. Bounded-distortion piecewise mesh parameterization. In *Proceedings of IEEE Visualization*, IEEE Computer Society, 355–362.

TARINI, M., CIGNONI, P., AND SCOPIGNO, R. 2003. Visibility based methods and assessment for detail-recovery. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society.

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. Polycube-maps. *ACM Trans. Graph. 23*, 3, 853–860.

ZHOU, K., SYNDER, J., GUO, B., AND SHUM, H.-Y. 2004. Isocharts: stretch-driven mesh parameterization using spectral analysis. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 45–54.