GPU-Based Multiresolution Deformation Using Approximate Normal Field Reconstruction

Martin Marinov, Mario Botsch, Leif Kobbelt
Computer Graphics Group, RWTH Aachen, Germany

Multiresolution shape editing performs global deformations while preserving fine surface details by modifying a smooth base surface and reconstructing the modified detailed surface as a normal displacement from it. Since two non-trivial operators (deformation and reconstruction) are involved, the computational complexity can become too high for real-time deformations of complex models. We present an efficient technique for evaluating multiresolution deformations of high-resolution triangle meshes directly on the GPU. By precomputing the deformation functions as well as their gradient information we can map both the deformation and the reconstruction operator to the GPU, which enables us to reconstruct the deformed positions and sufficiently close approximations of the normal vectors in the vertex shader in a single rendering pass. This allows us to render dynamically deforming 3D models several times faster than on the CPU. We demonstrate the application of our technique to two modern multiresolution approaches: one based on (irregular) displaced subdivision surfaces and the other one on volumetric space deformation using radial basis functions.

Categories and Subject Descriptors: I.3.5 [**Computational Geometry and Object Modeling**]:
Modeling packages

Additional Key Words and Phrases: Multiresolution mesh modeling, GPU processing

1. INTRODUCTION

Interactive mesh editing has become an essential component of most modern CAD systems. The conceptual simplicity and algorithmic efficiency of processing triangle meshes has established their effective application in various geometry processing areas, ranging from early product development phases, such as rapid prototyping, to character animation and rendering in movies and 3D games.

Among the variety of existing shape deformation approaches, multiresolution modeling appears to be one of the most powerful paradigms: Decomposing the geometry into a low frequency base surface and high frequency detail coefficients allows for global deformations with natural fine-scale detail preservation. This enables editing of a fully detailed mesh by manipulating the control handles available on the base surface (Fig. 1).

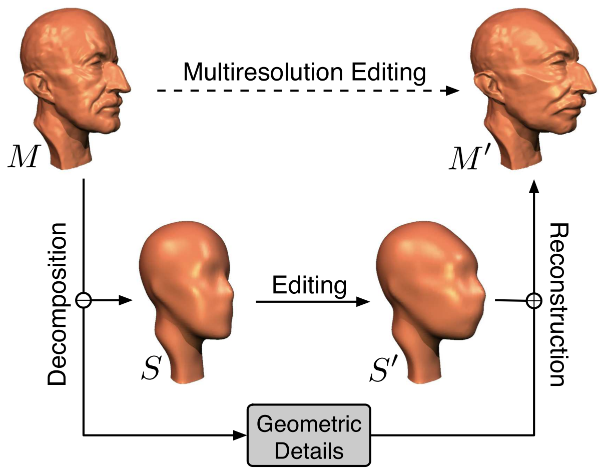

Fig. 1. A general multiresolution editing pipeline.

Especially in the context of engineering applications, meshes have to represent sufficiently accurate approximations of the true surface. As a consequence, models often consisting of millions of triangles have to be processed. Rendering (static) models of this complexity can be done very efficiently using modern graphics processors (GPUs), which have doubled their performance several times during the last few years. In contrast, CPUs improve their speed at much slower rate, limited not only by their general purpose design, but also by the legacy of the numerous existing software systems, which impose hard compatibility constraints on today's processor development. As a consequence, *dynamic modeling* of complex meshes is still a rather slow process, as the deformed geometry is typically computed on the main processor (CPU) and uploaded thereafter to the GPU for rendering.

Given all of these factors, a natural step for improving the performance of the existing mesh modeling frameworks is outsourcing as much as possible of the deformation's computation to the high-performance GPU and exploiting its specialized, highly parallel design. Hence, in this paper we present algorithms for computing both the *deformation* and the detail *reconstruction* emerging in a typical two-scale mesh modeling framework on the GPU. To make this possible, we show how to precompute special deformation basis functions, which can then be evaluated on the GPU to efficiently yield the deformed base surface S' (Fig. 1). Moreover, for surface shading, we need to compute the normal field of the deformed fine-scale mesh M'. However, the lack of mesh neighborhood information in a GPU program prevents a simple recomputation of vertex normals by local averaging. Therefore, we solve this problem by deriving an efficient approximate normal reconstruction technique which allows the reconstruction and the rendering of M' completely on the GPU.

2. CONTRIBUTIONS

In this paper we examine the components of a typical two-scale modeling framework and present specific algorithms for implementing them on the vertex shader of the GPU in a *single* rendering pass. Since the lack of mesh neighborhood information in a GPU vertex program prevents a simple recomputation of vertex normals by local averaging, the major challenge is to derive a technique for computing the deformed normals using terms associated solely with the currently processed vertex [d'Eon 2004]. The problem is additionally complicated by the fact that in a two-scale multiresolution framework we are rendering a deforming (fine-scale) geometry which depends in a non-linear fashion on the zero and first order terms evaluated on the deforming base surface (Fig. 1).

Therefore, our main contribution is a simple and efficient new method for approximating the normal field of the deformed fine scale mesh M' using a (relatively) small amount of attributes attached to the currently processed vertex $p_i \in M'$ (Section 4). This technique allows us to completely outsource the reconstruction and rendering of M' during multiresolution deformations to the GPU. For instance, we enhance the efficient mesh modeling framework proposed in [Botsch and Kobbelt 2005] by multiresolution editing. As another application, we present the necessary algorithms to map the complete reconstruction of generalized displaced subdivision surfaces proposed in [Marinov and Kobbelt 2005] onto the GPU.

Although we discuss our rendering technique in the context of the multiresolution modeling frameworks presented in [Botsch and Kobbelt 2005; Marinov and Kobbelt 2005], its application is not limited to these specific cases. In general it can be applied to any two-band multiresolution editing approach which can compute the deformed base surface and its tangents on the GPU. An additional advantage of our technique is that it is practically independent from the CPU performance, which ensures its rapid scalability in the future, since GPU performance improves at much higher rate. Moreover, sophisticated applications can take advantage of the idle CPU to run additional calculations in parallel, since the editing and the rendering processes are completely computed on the GPU.

3. MULTIRESOLUTION MODELING

Existing multiresolution modeling approaches can be classified by the way they represent and deform the base surface, as well as by the encoding of the high frequency details. Modeling systems based on subdivision surfaces can naturally exploit their hierarchical structure for the multiresolution representation and the surface deformation [Zorin et al. 1997; Litke et al. 2001]. In contrast, variational concepts are used for constructing the multiresolution hierarchy in the general case of irregular triangulations, and smooth deformations are derived from an energy minimization principle [Kobbelt et al. 1998; Kobbelt et al. 1999; Guskov et al. 1999; Botsch and Kobbelt 2004; 2005].

The geometric difference between any two subsequent hierarchy levels — or between the base surface and the fine-scale mesh if only two levels are used (Fig. 1) — are encoded as *displacements vectors* associated with the mesh vertices, which are stored in local-frame coordinates. In general, these displacements have both tangential and normal components [Zorin et al. 1997; Guskov et al. 1999; Litke et al. 2001], which, however, might lead to reconstruction artifacts. Removing the tangential components and thus restricting to *normal displacements* avoids these problems, but corresponds to a resampling of either the fine-scale surface [Lee et al. 2000; Guskov et al. 2000] or the base surface [Kobbelt et al. 1998; Kobbelt et al. 1999]. As the base surface is known to be smooth, and hence a resampling will not introduce aliasing, the latter approach is generally preferred.

Some recent approaches to multiresolution modeling of arbitrary meshes do not require an explicit frequency decomposition, but instead represent the surface details in terms of so-called Laplacian or differential coordinates [Yu et al. 2004; Sorkine et al. 2004; Lipman et al. 2004]. The editing operator then modifies these differential coordinates either explicitly [Yu et al. 2004] or implicitly [Sorkine et al. 2004] and reconstructs the fine-scale surface by solving a linear Laplacian system. Very recently, rotation invariant coordinates for mesh deformations were proposed in [Lipman et al. 2005] and a volume-aware extension of the Laplacian coordinates modeling approach was proposed in [Zhou et al. 2005].

3.1 Two-scale multiresolution modeling framework concepts

We will now give an overview of a typical two-scale multiresolution modeling framework by describing it in terms of the three involved operators, i.e., decomposition, editing, and reconstruction (Fig. 1). Our goal is to map as much as possible of these operators to the GPU in order to exploit its superior stream processing performance. One apparent consequence of the GPU approach is that each vertex has to be processed individually, which complicates the normal vector computation due to the lack of mesh neighborhood information.

Decomposition: In order to be able to process irregular fine-scale input meshes without resampling, the input mesh M is decomposed into a smooth base surface S (low frequencies) and normal displacements N_S (high frequencies). The position of each fine-scale vertex $p_i \in M$ is defined as:

$$p_i = S(t_i) + d_i \cdot N_S(t_i)\,, \quad N_S = \frac{\left(\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}\right)}{\left\|\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}\right\|}\,, \tag{1}$$

where t_i is the parameter value corresponding to the *exact* foot-point $S(t_i)$ of p_i on S, and

$$d_i = \langle p_i - S(t_i), N_S(t_i)\rangle \;.$$

To allow for surfaces of arbitrary topology and to reduce the number of special cases, we assume that the parameter domain of S is composed by a set K of patches k_i. More precisely $t_i = [k_i, (u_i, v_i)]$, where $k_i \in K$ and (u_i, v_i) are local parameters inside k_i. This definition is sufficiently general and allows us to use the same notation for both presented modeling frameworks: In case we are using base subdivision surfaces (Section 5), the patches K are the faces of the control mesh at some sufficiently refined level. In case we are using a smooth mesh as a base surface (Section 6), every triangle on the base mesh corresponds to a separate patch.

Editing: To edit M the user simply deforms the base surface S by manipulating the controls provided by the modeling framework (Fig. 1). The choice of controls and the semantics of their manipulation are application specific. Computing the deformed base surface S' and its normal field $N_{S'}$ (or its tangent fields $\frac{\partial S'}{\partial u}$ and $\frac{\partial S'}{\partial v}$) depends on the representation of both the deformation and the base surface (see Section 5 and Section 6).

Reconstruction: The reconstruction operator computes the deformed fine-scale mesh M' by evaluating the normal displacement (1) for every vertex $p_i \in M$. However, reconstructing the normals of M' on the GPU is not trivial as we describe in detail in the next section.

4. NORMAL FIELD RECONSTRUCTION

In order to render the deformed mesh M', we not only have to reconstruct its vertex positions as a normal displacement from S' according to (1), but we also have to derive its new normal field $N_{M'}$. When computing deformation and reconstruction on the GPU, we need to compute the normals only using the information associated with the currently processed vertex $p_i \in M'$.

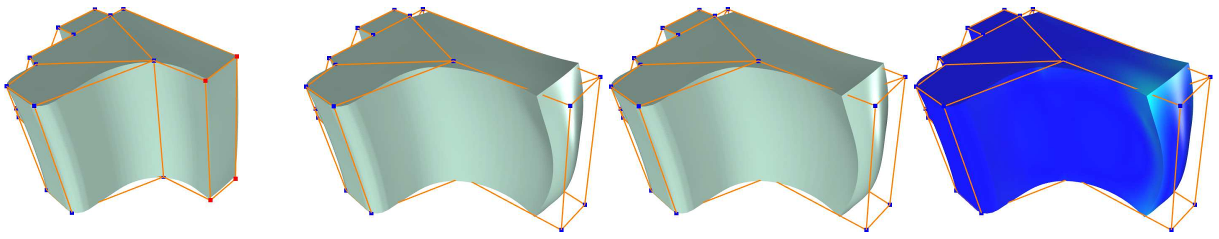

Fig. 2. From left to right: (a) The fan model is deformed by pulling the four red control points. (b) Shading using the exact normals. (c) Shading using the approximate normals. (d) Color plot of the approximate normals deviation (maximum deviation is 23 degrees, mean deviation computed only for the modified fine scale vertices is 1 degree).

Consider the normal displacements d_i as samples of a (piecewise) smooth scalar-valued function D defined over the smooth base surface S. Hence the input mesh M can be seen as a piecewise linear approximation of a parametric surface F defined by generalizing (1) over the parameter domain of S:

$$F(t) = S(t) + D(t) \cdot N_S(t)\,. \tag{2}$$

To handle sharp features we assume that for a vertex p_i lying on a sharp feature the position and the normal are duplicated as many times as different smooth sectors are adjacent to it. With this assumption, both the theoretical discussion and the practical implementation of the normal reconstruction do not need a special case distinction for sharp feature vertices. Also when discussing the behavior of F at given t_i it is more convenient to restrict the parameter domain to the corresponding patch k_i and write S, D and F simply as functions of two parameters (u, v) spanning the domain of k_i. The partial derivatives of F and its normal N_F are then:

$$\frac{\partial F}{\partial u} = \frac{\partial S}{\partial u} + D \cdot \frac{\partial N_S}{\partial u} + \frac{\partial D}{\partial u} \cdot N_S\ , \tag{3}$$

$$\frac{\partial F}{\partial v} = \frac{\partial S}{\partial v} + D \cdot \frac{\partial N_S}{\partial v} + \frac{\partial D}{\partial v} \cdot N_S\ , \tag{4}$$

$$N_F = \left(\frac{\partial F}{\partial u} \times \frac{\partial F}{\partial v}\right) / \left\| \frac{\partial F}{\partial u} \times \frac{\partial F}{\partial v} \right\|\ . \tag{5}$$

Since D does not change during deformation and we can compute a priory the normal N_F at p_i on the original (undeformed) mesh M using the adjacent triangles normals we can estimate the gradient $\nabla D(t_i)$ by solving the systems (3) and (4) and exploiting the fact that $N_F \perp \frac{\partial F}{\partial u}, \frac{\partial F}{\partial v}$. This will allow us to reconstruct the deformed normal $N_{F'}$ field exactly, if we are able to evaluate the second partial derivatives $\frac{\partial^2 S'}{\partial u^2}, \frac{\partial^2 S'}{\partial uv}, \frac{\partial^2 S'}{\partial v^2}$ needed for computing $\frac{\partial N'_S}{\partial u}$ and $\frac{\partial N'_S}{\partial v}$. Although this is possible in case S is a C^2 surface, computing the exact solution is not practically feasible mostly because it increases the amount of necessary calculations and precomputed terms (see Section 5.2.1) per fine scale vertex, effectively doubling the memory requirements and the computation time.

4.1 Approximate reconstruction

Ideally, for a given t_i, we would like to be able to reconstruct N_F using only S, $\frac{\partial S}{\partial u}$ and $\frac{\partial S}{\partial v}$ since these are the terms which are required to compute (1). The terms which require second order derivatives in (3), (4) are the derivatives of the normal N_S. However, it is known that the first order derivatives of N_S lie in the tangent plane of S. As a consequence they can be represented as

$$\frac{\partial N_S}{\partial u} = \lambda_u \frac{\partial S}{\partial u} + \mu_u \frac{\partial S}{\partial v}\,, \quad \frac{\partial N_S}{\partial v} = \lambda_v \frac{\partial S}{\partial u} + \mu_v \frac{\partial S}{\partial v}\,.$$

Substituting these expressions in (3), (4) we get

$$\frac{\partial F}{\partial u} = (1 + \lambda_u D)\frac{\partial S}{\partial u} + \mu_u D \cdot \frac{\partial S}{\partial v} + \frac{\partial D}{\partial u} \cdot N_S\,,$$

$$\frac{\partial F}{\partial v} = \lambda_v D \cdot \frac{\partial S}{\partial u} + (1 + \mu_v D)\frac{\partial S}{\partial v} + \frac{\partial D}{\partial v} \cdot N_S\,,$$

i.e., we can naturally represent $\frac{\partial F}{\partial u}$ and $\frac{\partial F}{\partial v}$ in the form

$$\frac{\partial F}{\partial u} \;=\; \alpha_u \frac{\partial S}{\partial u} + \beta_u \frac{\partial S}{\partial v} + \gamma_u N_S\;, \tag{6}$$

$$\frac{\partial F}{\partial v} \;=\; \alpha_v \frac{\partial S}{\partial u} + \beta_v \frac{\partial S}{\partial v} + \gamma_v N_S\;. \tag{7}$$

Since F does not have a natural parameterization, the coefficients of the linear combination (6), (7) are determined by taking two arbitrary orthonormal tangent vectors T_0 and T_1 in the tangent plane defined by the undeformed input normal N_F (computed by averaging) and solve the two resulting 3×3 linear systems $\frac{\partial F}{\partial u} = T_0$ and $\frac{\partial F}{\partial v} = T_1$.

After deriving these coefficients from the original undeformed state, we can use them during editing in order to reconstruct the deformed normals $N_{F'}$ from the deformed base surface S' and its partial derivatives by applying equations (6), (7). Note also that although the partial derivatives $\frac{\partial S'}{\partial u}$ and $\frac{\partial S'}{\partial v}$ in general will be scaled anisotropically by the editing operator, the normal vector $N_{S'}$ is computed as their *normalized* cross product. However, while γ_u, γ_v encode the displacement function's gradient $\left(\frac{\partial D}{\partial u}, \frac{\partial D}{\partial v}\right)$, which does not change during a deformation, the coefficients α_u, α_v and β_u, β_v are actually functions of the curvature along the tangent directions $\frac{\partial S}{\partial u}$ and $\frac{\partial S}{\partial v}$. During deformation, we do not consider the second derivatives of S' in (6), (7), therefore our *approximate* normal reconstruction in fact neglects the change of curvature of the base surface. However, our experiments show that the loss of accuracy in the reconstructed normal field does not degrade the quality of the shading noticeably since the deviation from the exact normals is small under reasonable deformations (Fig. 2, supplied video). Even for the extreme deformations shown in the video the average approximation error for the reconstructed normals is never above 5 degrees.

Under very extreme deformations the approximate reconstruction might fail to give results sufficiently close to the true normals if the base surface S is too far away from the fine scale mesh M. The reason for this is that if the displacements d_i are very large, the error emerging from ignoring the second order terms of S is amplified considerably. One way to resolve this issue is to fit the base surface S to M using a technique similar to the one proposed in [Marinov and Kobbelt 2004], which minimizes the lengths of the displacements d_i.

Note also that our approximate normal reconstruction is only required for the purpose of implementing the real-time rendering of the deformed mesh on the GPU. For applications which require precise mesh normals, e.g., simulation, the exact normals can be recomputed in a offline pass after the editing of the surface shape is finalized.

5. SUBDIVISION SURFACE BASED MESH MODELING

In this section we specialize the abstract framework defined in Section 3.1 in order to implement the two-scale multiresolution modeling algorithm presented in [Marinov and Kobbelt 2005] on the GPU. Similarly to [Lee et al. 2000], this method decomposes the fine scale mesh M into a (Catmull-Clark [Catmull and Clark 1978] or Loop [Loop 1987]) *subdivision* base surface S and a set of normal displacements (Fig. 3). However, instead of resampling M, the approach presented in [Marinov and Kobbelt 2005] resamples the base surface S by defining an irregular, non-dyadic parameterization of the vertices of M. Hence, resampling artifacts are avoided, but the reconstruction procedure must

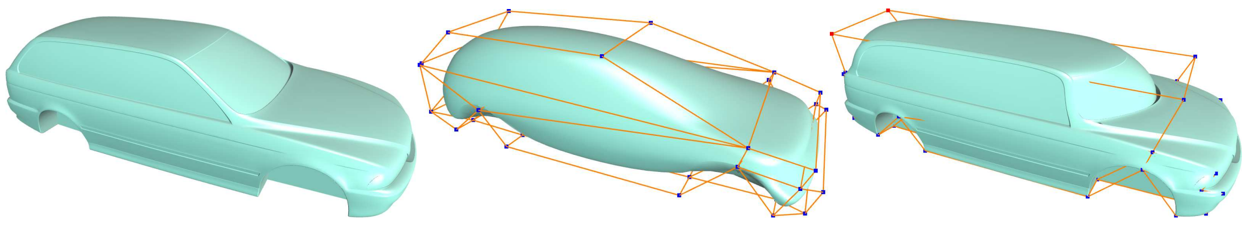

Fig. 3. A car model (left) is represented as normal displacement with respect to a smooth Catmull-Clark surface (middle). Manipulating the vertices of its control mesh, a large scale deformation is performed on the mesh (right).

be able to evaluate S at arbitrary (non-dyadic) parameter values. To implement this evaluation efficiently on the GPU, we adapt the idea proposed in [Bolz and Schröder 2002] by precomputing the basis functions of the subdivision surface at the (irregular) parameterization of the modeled mesh vertices.

GPU processing for mesh deformations was an unexplored field until the advance of the programmable graphics hardware in the last few years. Smooth geometry images [Losasso et al. 2003] represent regular genus zero meshes as textures, which can be fully implemented on the fragment shader of the GPU, and thereby allow for very efficient and simple rendering and animation at different levels of detail. Bolz et al. implemented their subdivision surfaces rendering technique [Bolz and Schröder 2002] on the GPU [Bolz and Schröder 2004] using a two pass "render to vertex array" approach. They precompute subdivision basis functions (up to a predefined maximum valence) at all *dyadic* parameter values (up to a certain refinement level), which allows them to refine uniformly or adaptively the control mesh of a Catmull-Clark surface on the GPU. More advanced methods for tessellating subdivision surfaces on the GPU were proposed recently in [Bunnell 2005; Shiue et al. 2005].

5.1 Basis function precomputation

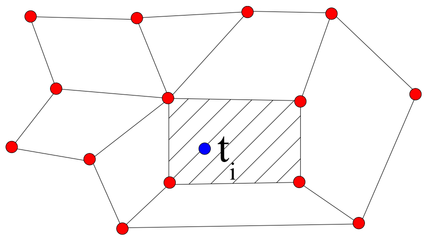

Fig. 4. Catmull-Clark surfaces basis functions' support: only the basis functions associated with the control vertices (red dots) incident to a face adjacent to the marked patch do not vanish for any parameter value t_i in the center face.

Given a multiresolution model where the geometry of a mesh M is represented with respect to a subdivision surface S in the form (1), reconstructing the position of a vertex p_i is straightforward since we only need to evaluate S and its normal N_S at the corresponding parameter value t_i using [Stam 1998; 1999]. At a closer look, S can be represented as a linear combination of compactly supported scalar-valued basis functions ϕ_j associated with the vertices c_j of its control mesh C_0. Hence $S(t_i)$ and its derivatives can be written as

$$S(t_i) = \sum_{j=0}^{N} \phi_j(t_i) c_j \,, \tag{8}$$

$$\frac{\partial S}{\partial u}(t_i) = \sum_{j=0}^{N} \frac{\partial \phi_j}{\partial u}(t_i) c_j \,, \tag{9}$$

$$\frac{\partial S}{\partial v}(t_i) = \sum_{j=0}^{N} \frac{\partial \phi_j}{\partial v}(t_i) c_j \,. \tag{10}$$

Exploiting this representation and the fact that the parameterization t_i is fixed during a modeling session, we can precompute $\phi_j(t_i)$, $j = 0..N$ and their partial derivatives for every t_i and cache the resulting values. Moreover, since ϕ_j are compactly supported, only the basis functions' values in the neighborhood of the patch f_i where t_i resides have to be stored (Fig. 4). The number of non-vanishing ϕ_j for a given t_i is determined by the local connectivity structure of C_0 at f_i and is 16 for a fully regular C_0. Therefore to reconstruct p_i in terms of the linear combinations (8), (9), (10) we need to precompute on average $16 \times 3 = 48$ scalars.

The main difficulty with this approach comes from the fact that unlike B-splines, which possess very simple polynomial definition, the subdivision basis functions ϕ_j depend on various factors: The valence of the control vertex c_j, the valences of its direct neighbors in C_0, the valences of all adjacent faces in C_0 (for Catmull-Clark [Catmull and Clark 1978] surfaces) and the presence of adjacent boundary and crease edges.

Therefore, in practice, we evaluate the basis function ϕ_j by assigning an additional scalar attribute σ_0 to the vertices of the control mesh C_0 and set $\sigma_0(c_j) = 1$ and $\sigma_0(c_i) = 0,\ j \neq i$ [Bolz and Schröder 2002]. Refining C_0 k times, where k is the number of subdivisions required by Stam's procedure ($k = 1$ for Loop and $k = 2$ for Catmull-Clark surfaces), we apply the corresponding subdivision rules (exclusively) on the scalar values σ_0. The resulting scalar field σ_k defines the subdivision basis function ϕ_j for which positions and partial derivatives can be easily evaluated at arbitrary t_i using [Stam 1998; 1999]. Once all of these coefficients are precomputed, $S(t_i)$ and $N_S(t_i)$ can be evaluated efficiently by (8) without accessing the control mesh whenever the control vertices c_j are modified. This allows the (CPU-based) reconstruction of the deformed mesh M at interactive frame-rates for most meshes with moderate complexity. Moreover, we can now take advantage of the simplicity of the formulas (1) and (8) to implement the reconstruction of M on the GPU (Section 5.2), allowing interactive multiresolution modeling of high-resolution dense meshes.

5.2 GPU implementation details

5.2.1 *Rendering path.* There are several ways to implement the deformation of the mesh M on the GPU. We achieved best performance results when transferring the data required for the computations attached as vertex attributes. More precisely, we supply for every p_i regardless of the type of the actual deformation the coefficients needed to reconstruct it and its normal vector based on (1), (6), (7) : $d_i, \alpha_u, \alpha_v, \beta_u, \beta_v, \gamma_u, \gamma_v$ (2 attributes/7 floats). Once the set of modified control vertices Q is selected, we precompute the fraction of (8) for $S(t_i)$, $\frac{\partial S}{\partial u}(t_i)$ and $\frac{\partial S}{\partial v}(t_i)$ which is independent of Q (and hence stays constant) and store it in 3 additional attributes (9 floats). Now per every control vertex $c_j \in Q$ we define an additional attribute which stores $\phi_j(t_i)$, $\frac{\partial \phi_j}{\partial u}(t_i)$ and $\frac{\partial \phi_j}{\partial v}(t_i)$ (3 floats). This results in a total of $16 + 3\,|Q|$ floats attached per vertex. All of the precomputed data can be uploaded to the video memory using vertex buffer objects since it does not change as long as the set Q remains unchanged (not their positions). Since $\sum_{j=0}^{N} \phi_j(t_i) = 1$ and $0 \leq \phi_j, \left|\frac{\partial \phi_j}{\partial u}\right|, \left|\frac{\partial \phi_j}{\partial v}\right| \leq 1$, the $3\,|Q|$ floats representing the basis functions terms can be safely quantized to 16 or 24 bits precision (for GPUs which support it) to reduce the amount of necessary video memory. Finally, the only data we need to transfer to the GPU at every frame are the current positions of the control vertices $c_j \in Q$, i.e., $3\,|Q|$ floats per frame, which we upload using global shader variables (uniforms) before rendering the mesh triangles.

5.2.2 *Affine transformations of control vertices.* There are two issues with the (rather rigid) approach described above: First, we need to submit for every vertex $p_i \in M$ the coefficient $\phi_j(t_i)$ in the linear combination (8) for every modified control vertex c_j, even if p_i actually does not depend on c_j (since $\phi_j(t_i) = 0$), i.e., there is an additional memory overhead. Second, since the number of vertex attributes is limited (16 for our test system), it imposes a restriction for the maximum number of simultaneously modified control vertices (11 or 15 if the fourth component of attributes associated with c_j is used to transfer an additional scalar value).

However, that restriction is not an issue if the position of the simultaneously modified control vertices is defined by a single affine transform. Adapting the framework of [Botsch and Kobbelt 2004], we represent $Q = \{c_0 .. c_k\}$ as affine combination of four points $\{q_0 .. q_3\}$ forming an affine frame in $\mathbb{R}^3$, i.e., $c_j = \Sigma_{l=0}^{3} \xi_l q_l$. This implies

$$S(t_i) = \sum_{j=0}^{k} \phi_j(t_i) c_j = \sum_{j=0}^{k} \phi_j(t_i) \left(\sum_{j=0}^{3} \xi_{j,l} q_l \right) = \sum_{l=0}^{3} \left(\sum_{j=0}^{k} \xi_{j,l} \phi_j(t_i) \right) q_l \,,$$

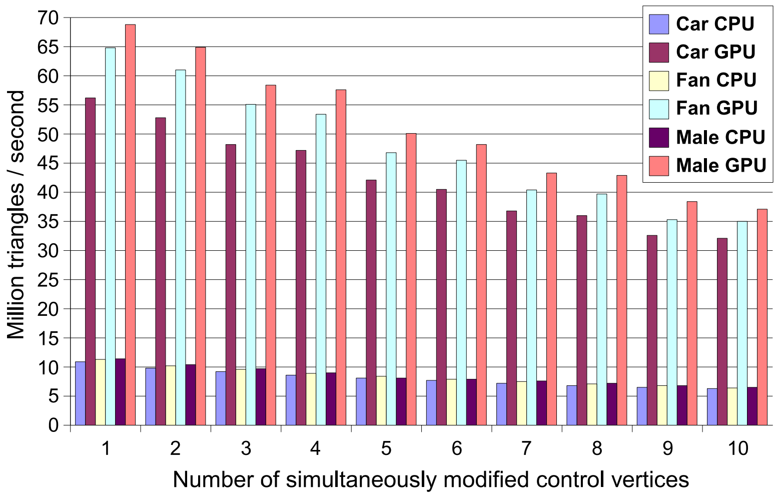

Table I. Subdivision surface based multiresolution mesh modeling benchmark — CPU vs. GPU performance chart. The number of simultaneously edited control vertices affects the computational complexity of the deformation since the amount of required data (attributes) per input mesh vertex depends linearly on it. The performance during an affine transformation of a larger number of control vertices (Section 5.2.2) corresponds to the case of four simultaneously modified control vertices. Models complexity (triangles): Car - 62K; Fan - 137K; Male head - 667K.

i.e., we can represent all foot-points $S(t_i)$ (and the corresponding derivatives $\frac{\partial S}{\partial u}(t_i)$ and $\frac{\partial S}{\partial v}(t_i)$) as a linear combination of $\{q_0 .. q_3\}$. Hence, it is sufficient to provide per vertex only the precomputed (fixed) fraction of $S(t_i)$, $\frac{\partial S}{\partial u}(t_i)$ and $\frac{\partial S}{\partial v}(t_i)$ and the 3×4 coefficients required for computing the dynamic fraction as linear combination of $\{q_0 .. q_3\}$. Therefore for such a deformation we need only 8 attributes (28 floats): 2 attributes (7 floats) for d_i, α_u, α_v, β_u, β_v, γ_u, γ_v; 3 attributes (9 floats) for the fixed fraction of $S(t_i)$, $\frac{\partial S}{\partial u}(t_i)$ and $\frac{\partial S}{\partial v}(t_i)$; 3 attributes (12 floats) for the dynamic fraction. Notice that the precomputed basis functions with respect to an affine frame $\{q_0 .. q_3\}$ do not cause overhead in the surface evaluation.

5.2.3 *GPU program.* All of the code for reconstructing the geometry and the normals of M' on the GPU is contained in a single vertex shader program. The program is generated on the fly once the set Q is defined in order to avoid costly case distinctions in the vertex shader. We implemented the computation in both GLSLang (OpenGL 2.0) and in assembler (ARB_vertex_program). The GLSLang implementation performed constantly slower than our assembler program in all tests by a strongly varying factor reaching at times 90%, which can be explained with the lack of mature support for GLSLang in the current drivers' generation.

5.2.4 *Mesh optimization.* Since we render the mesh in a single pass, every p_i is reconstructed k times where k is the number of triangles adjacent to it. For most high-resolution meshes the average is $k \simeq 6$. However, in practice the vertex geometry is never computed so many times, since modern GPUs store the results of the most recent computations in a FIFO cache. Hence it is of crucial importance to reorder the indices of the mesh triangles in order to minimize the amount of cache misses [Hoppe 1999]. Since the computation requires a large number of video memory accesses, using an interleaved attribute layout and a sequential reordering of the vertex data in order to minimize the memory access latency turned out to be quite important as well. These optimizations accounted for more than 50% of the performance gain for some models.

5.3 Results

We tested our subdivision surface based multiresolution modeling algorithm on a 3.0GHz Pentium4 system with AGP 8x GeForce 6800 Ultra graphics card. The GPU implementation of the deformation and the reconstruction results in significant performance gains compared to an implementation of the computation of (1) on the CPU, using exactly the same algorithmic components, i.e., precomputed basis functions (8) and approximate normal computation (6), (7) (Table I). For example, our demo simultaneously deforms 10 control points on the Male model ($667K$ triangles) at 55.6 fps using GPU reconstruction as compared to 9.7 fps on the CPU (see the supplied video). The source code of the our implementation used to benchmark the performance and test models are available at our web site [Marinov and Botsch 2005].

5.4 Limitations

Beside mesh editing, subdivision based key-frame animation is one of the most promising applications of our algorithm. Unfortunately, the high performance "vertex attributes" rendering path on the GPU does not allow more complex deformations, since we are not able to access the full range of precomputed basis functions information for every p_i on the GPU. Hence, we also implemented our technique using the brand new "vertex textures" GPU feature, which allows uploading our (rather) sparse precomputed coefficients structure without restrictions (and memory overhead) on the GPU. However, this hardware feature has not fully matured yet, therefore this rendering path actually turns out to be much slower than the CPU reconstruction. If the next generation of mainstream GPU hardware provides either much faster vertex textures access or more (about 32 instead of just 16) vertex attributes, we will be able to perform full featured key-frame animation of multiresolution models on the GPU without any (significant) CPU intervention.

6. MULTIRESOLUTION SPACE DEFORMATION

The second GPU-accelerated shape editing technique we integrated into the multiresolution framework described in Sections 3.1 and 4 is a space deformation approach. In contrast to the surface-based editing method of Section 5 — where all deformations are computed *explicitly* on the base surface S — we now construct a function $r : \mathbb{R}^3 \to \mathbb{R}^3$ which deforms the whole 3D space and thereby *implicitly* deforms the embedded base surface S. The large conceptual difference between the two example applications proves the general applicability of our GPU-based multiresolution framework.

6.1 Overview

In a typical modeling session the user prescribes a set of displacement constraints $h_i \mapsto h_i'$ ($1 \le i \le m$) for arbitrary *handle* points $h_i \in \mathbb{R}^3$. Then a space deformation function $r(\cdot)$ is constructed to smoothly interpolate these constraints, i.e., $r(h_i) = h_i'$. This function is finally used to transform all surface points $p_i \in S$ and their associated tangent vectors t_i or normal vectors n_i:

$$\begin{aligned} p_i &\mapsto r(p_i) \\ t_i &\mapsto J_r(p_i)t_i \\ n_i &\mapsto J_r(p_i)^{-\mathsf{T}} n_i \, , \end{aligned}$$

where $J_r(p_i) \in \mathbb{R}^{3\times 3}$ denotes the Jacobian of $r(\cdot)$ at position p_i (Fig. 5).

Although our GPU-based multiresolution framework can be used in combination with any space deformation technique that can be implemented on the GPU [Sederberg and Parry 1986; Hsu et al. 1992], we focus on the approach of [Botsch and Kobbelt 2005], which represents the deformation function $r(\cdot)$ by triharmonic radial basis functions

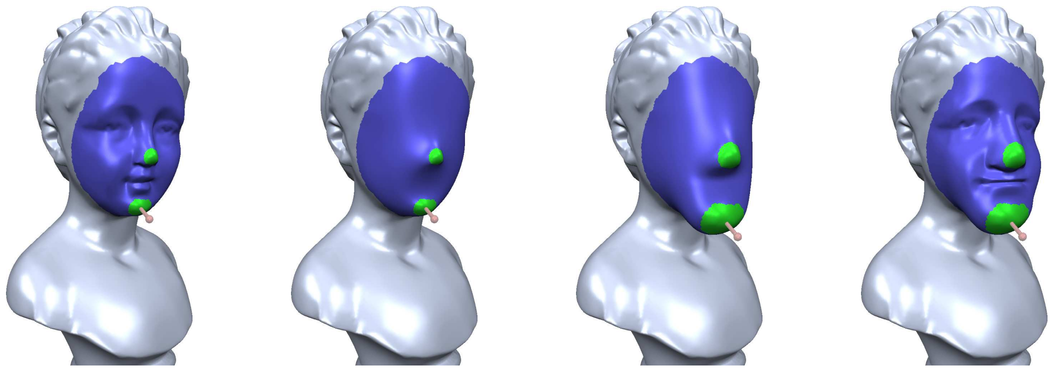

Fig. 5. Multiresolution space deformation - from left to right: (a) The user paints a deformable region (blue) and two handle regions (green) on the bust model. During editing, each selected handle region undergoes an affine transformation, while the deformable area smoothly interpolates the new positions of the handle region with the rest of the surface, which is left unaffected. (b) A smoothed version of the painted mesh area is used as a base surface to control the deformation. The original positions of the painted vertices are represented as normal displacements with respect to the base surface. (c) The base surface is edited by transforming the red manipulator attached to one of the handles. (d) The deformed mesh M' is then reconstructed by adding the normal displacements over the deformed base surface. Notice, that the base surface (b,c) is not shown to the user.

(RBFs) as

$$r(x) = \sum_{j=1}^{m} w_j \, \varphi \left(\|c_j - x\| \right) \, .$$

Here, $\varphi(t) = t^3$ is the underlying scalar profile function, and $c_j \in \mathbb{R}^3$ and $w_j \in \mathbb{R}^3$ represent the center and weight of the jth basis function $\varphi\left(\|c_j - x\|\right)$.

In order to interpolate the user's constraints $h_i \mapsto h_i'$ by an RBF deformation function the centers c_j are chosen to coincide with the constraints h_i and a linear system is solved for the weights w_j (see [Botsch and Kobbelt 2005] for details). If the handle points h_i are deformed by a single affine transformation only, deformation basis functions can be precomputed just like in Section 5.2.2. Evaluating these basis functions in a vertex shader then allows for real-time deformations of highly complex 3D models.

However, as this approach is a free-form deformation of the embedding, there is no mechanism to preserve small surface details under large deformations (Fig. 6, left). Integrating this method into a multiresolution framework, i.e., applying the RBF space deformation to the base surface S instead of to the fine-scale surface M, results in a much better detail preservation (Fig. 6, right).

As the RBF space deformation enables the computation of S', $\frac{\partial S'}{\partial u}$ and $\frac{\partial S'}{\partial v}$ on the GPU, it can easily be integrated into the GPU-based multiresolution modeling framework described in Section 3.1. This results in a fully GPU-based multiresolution space deformation technique, which runs at almost the same speed as the free-form deformation of [Botsch and Kobbelt 2005], as we will show below.

6.2 Implementation

The implementation basically follows Section 5 and differs only in the way the deformed base surface and its tangents are computed. For each vertex p_i we transfer the following data to the vertex shader (as vertex attributes):

—Original base point $S(t_i)$ and its tangents $\frac{\partial S}{\partial u}(t_i)$ and $\frac{\partial S}{\partial v}(t_i)$ (9 floats).

—Basis functions for volumetric deformation applied to S and its partial derivatives (12 floats).

—The coefficients d_i, α_u, α_v, β_u, β_v, γ_u, γ_v for the reconstruction (7 floats).

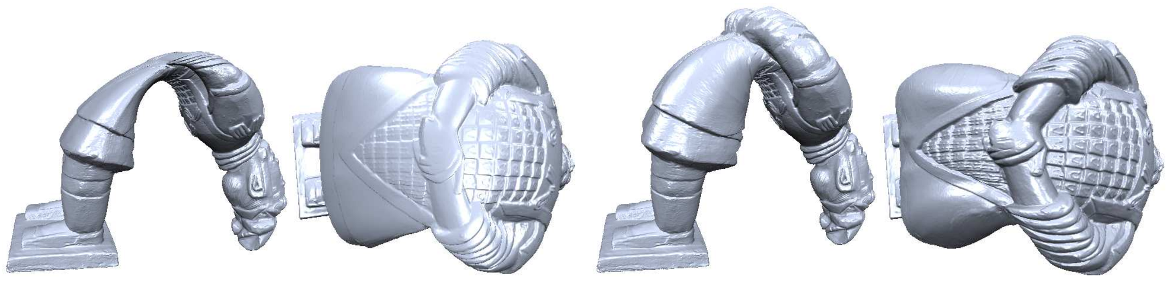

Fig. 6. The original *free-form* space deformation approach of Botsch and Kobbelt does not preserve fine surface details under large deformations (*left*). Integrating this technique into our GPU-based two-scale multiresolution framework yields much better results at almost the same speed (*right*). Even for this extreme deformation the average error of our approximate normal reconstruction is only 5 degrees.

Model	#Vertices	#Active	#Triangles	MR–CPU	MR–GPU	FFD–GPU
Male head	332K	270K	664K	3.6	71.6	77.8
Chinese statue	1M	665K	2M	4.5	69.4	76.4
Bust	984K	880K	1.97M	3.6	80.0	87.6

Table II. Performance (million triangles/second) for computing and rendering different shape deformations on several models, whose total number of vertices/triangles and number of active vertices are given. The GPU-accelerated RBF multiresolution deformation (*MR–GPU*) outperforms the CPU implementation (*MR–CPU*) by a factor between 15 and 20. Note that the performance of the CPU implementation depends on the percentage of active vertices (only the active vertices are computed every frame). Hence, if a more global deformation is used, the performance of the CPU implementation will drop down, while the performance of the GPU implementation will remain constant, since our vertex shader processes equally all vertices. Compared to the GPU-based RBF *free-form* deformation (*FFD–GPU*), the overhead of computing the multiresolution reconstruction results in a small ($\approx 10\%$) performance loss only.

A simple vertex shader first evaluates the deformation basis functions in order to compute the deformed terms S', $\frac{\partial S'}{\partial u}$ and $\frac{\partial S'}{\partial v}$ and $N_{S'}(t_i)$. It then computes position and (approximate) normal vector of the displaced point p_i' by evaluating equations (1), (6), (7), and (5).

We improve the performance by applying the mesh optimization technique of Section 5.2.4 in order to optimally exploit the post-T&L cache of the GPU. We further reduce transfer costs by representing everything but the absolute position $S(t_i)$ and the displacement d_i using 16 or 24 bits only, which can safely be done as the absolute values are bounded by 1.

6.3 Results

The resulting GPU-accelerated multiresolution editing technique leads to much better results for large scale deformations compared to the pure free-form deformation (Fig. 6). Moreover, since the (approximate) reconstruction operator adds only a small overhead to the shader computations, multiresolution deformations can be performed at almost the same speed as free-form deformations. Compared to an analogous CPU implementation, the GPU-based version is faster by a factor of up to 20. The per-frame timings for these approaches, measured on a 3.0GHz Pentium4 with GeForce 6800 Ultra, can be found in Table II.

REFERENCES

BOLZ, J. AND SCHRÖDER, P. 2002. Rapid evaluation of Catmull-Clark subdivision surfaces. In *Proc. of the 7th International Conference on 3D Web Technology*. ACM Press, 11–17.

BOLZ, J. AND SCHRÖDER, P. 2004. Evaluation of subdivision surfaces on programmable graphics hardware. In *(to appear)*.

BOTSCH, M. AND KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. 23,* 3 (Aug.), 630–634.

BOTSCH, M. AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. *Computer Graphics Forum 24,* 3, 611–621.

BUNNELL, M. 2005. Adaptive tessellation of subdivision surfaces with displacement mapping. In *GPU Gems 2*, R. Fernando, Ed. Addison-Wesley, Reading, MA, 109–122.

CATMULL, E. AND CLARK, J. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10,* 6 (Sep), 350–355.

D'EON, E. 2004. Deformers. In *GPU Gems*, R. Fernando, Ed. Pearson Education, Inc., Boston, MA, 723–732.

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *Proc. of SIGGRAPH 99*. 325–334.

GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRÖDER, P. 2000. Normal meshes. In *Proc. of SIGGRAPH 00*. 95–102.

HOPPE, H. 1999. Optimization of mesh locality for transparent vertex caching. In *Proc. of SIGGRAPH 99*. 269–276.

HSU, W. M., HUGHES, J. F., AND KAUFMAN, H. 1992. Direct manipulation of free-form deformations. In *Proc. of SIGGRAPH 92*. 177–184.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. of SIGGRAPH 98*. 105–114.

KOBBELT, L., VORSATZ, J., AND SEIDEL, H.-P. 1999. Multiresolution hierarchies on unstructured triangle meshes. *Comput. Geom. Theory Appl. 14,* 1-3, 5–24.

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Proc. of SIGGRAPH 00*. 85–94.

LIPMAN, Y., SORKINE, O., COHEN-OR, D., LEVIN, D., RÖSSL, C., AND SEIDEL, H.-P. 2004. Differential coordinates for interactive mesh editing. In *Proc. of Shape Modeling International*. 181–190.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24,* 3 (July), 479–487.

LITKE, N., LEVIN, A., AND SCHRÖDER, P. 2001. Fitting subdivision surfaces. In *IEEE Visualization 2001*. 319–324.

LOOP, C. 1987. Smooth subdivision surfaces based on triangles. M.S. thesis, University of Utah.

LOSASSO, F., HOPPE, H., SCHAEFER, S., AND WARREN, J. 2003. Smooth geometry images. In *Proc. of the Symposium on Geometry Processing*. 138–145.

MARINOV, M. AND BOTSCH, M. 2005. http://www-i8.informatik.rwth-aachen.de/publications/downloads/gpu-md/.

MARINOV, M. AND KOBBELT, L. 2004. Optimization techniques for approximation with subdivision surfaces. In *ACM Symposium on Solid Modeling and Applications*. 113–122.

MARINOV, M. AND KOBBELT, L. 2005. Automatic generation of structure preserving multiresolution models. *Computer Graphics Forum 24,* 3, 479–486.

SEDERBERG, T. W. AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proc. of SIGGRAPH 86*. ACM Press, 151–160.

SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime GPU subdivision kernel. *ACM Trans. Graph. 24,* 3, 1010–1015.

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proc. of the Symposium on Geometry Processing*. 179–188.

STAM, J. 1998. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Proc. of SIGGRAPH 98*. Vol. 32. 395–404.

STAM, J. 1999. Evaluation of Loop subdivision surfaces (course). In *SIGGRAPH 99*.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23,* 3 (Aug.), 644–651.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph Laplacian. *ACM Trans. Graph. 24,* 3, 496–503.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proc. of SIGGRAPH 97*. 259–268.