SIMOD: Making Freeform Deformation Size-Insensitive

Tamy Boubekeur, Olga Sorkine, Christophe Schlick

Eurographics Symposium on Point-Based Graphics (2007)
M. Botsch, R. Pajarola (Editors)

SIMOD: Making Freeform Deformation Size-Insensitive

Tamy Boubekeur* Olga Sorkine+ Christophe Schlick*
* : INRIA - University of Bordeaux + : TU Berlin

Abstract

Freeform deformation techniques are powerful and flexible tools for interactive 3D shape editing. However, while interactivity is the key constraint for the usability of such tools, it cannot be maintained when the complexity of either the 3D model or the applied deformation exceeds a given workstation-dependent threshold. In this system paper, we solve this scalability problem by introducing a streaming system based on a sampling-reconstruction approach. First an efficient ***out-of-core adaptive simplification*** *algorithm is performed in a pre-processing step, to quickly generate a simplified version of the model. The resulting model can then be submitted to arbitrary FFD tools, as its reduced size ensures interactive response. Second, a post-processing step performs a* ***feature-preserving reconstruction*** *of the deformation undergone by the simplified version, onto the original model. Both bracketing steps share* ***streaming*** *and* ***point-based*** *basis, making them fully scalable and compatible with point-clouds, non-manifold polygon soups and meshes. Our system also offers a* ***generic*** *out-of-core multi-scale layer to arbitrary FFD tools, since the two bracketing steps remain available for partial upsampling during the interactive session. As a result, arbitrarily large 3D models can be interactively edited with most FFD tools, opening the use and combination of advanced deformation metaphors to models ranging from million to billion samples. Our system also offers to work on models that fit in memory but exceed the capabilities of a given FFD tool.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

Freeform deformation (FFD) is a key tool in the computer graphics field and provides powerful interaction metaphors for editing the shape of 3D objects. Recent advances, such as *variational multiresolution* editing, *Laplacian* representation or *physically-plausible* deformation, have led to extremely flexible freeform modeling tools. However, interactivity remains a critical point and these tools are strongly bottle-necked by the size of the objects taken as input. This is somewhat in conflict with the recent advance of automatic acquisition devices, such as 3D laser range scanners, which make it possible to generate *large models* that may contain several hundreds millions of samples, in order to accurately capture extremely fine-scale geometric features. Actually, in spite of the continuous growth of hardware capabilities, even the mere visualization of such large models becomes a complex challenge. Several years of research have led to the development of specific out-of-core methods, that perform real-time rendering by combining a large amount of precomputation with a dedicated cache-coherent representation. Unfortunately, all these out-of-core systems have been designed for static objects rendering only and cannot be used in a dynamic context such as FFD. When the shape of the acquired object needs to be edited with some user-controlled deformation (character animation, for instance), the only existing solution is to employ a painful hand-tuned conversion process to provide higher level representations (e.g. displaced subdivision surfaces) better suited for shape editing and animation. The time imposed by this process can easily reach several weeks for models where the reconstruction, simplification and parametrization are particularly complex. Moreover, many fine-scale features are lost during this resolution reduction. When dealing with acquired shapes, the benefit offered by an accurate scanning process is partially wasted, loosing real-world object details which are acquired but not preserved during editing. Similar problems arise when very complex synthetic models are created with specific techniques such as displacement painting (e.g., ZBrush).

All these drawbacks originate in the incapacity to interactively edit the shape of a model that is **too large**, where the notion of "large" strongly depends on both the workstation and the FFD method used (even few hundred thousand samples may be too much in some cases). We propose a flexible solution to this problem, that allows the deformation of arbitrary surfaces, either sampled by points or polygons, at full resolution, without doing any conversion, nor loosing any sample. This paper is organized as follow: after having recalled some related work in Section 2, we present the framework of our system in Section 3, before focusing more deeply on its two out-of-core components: an adaptive point-based simplification (Section 4) and a feature-preserving deformation reconstruction (Section 5). Finally, implementa-

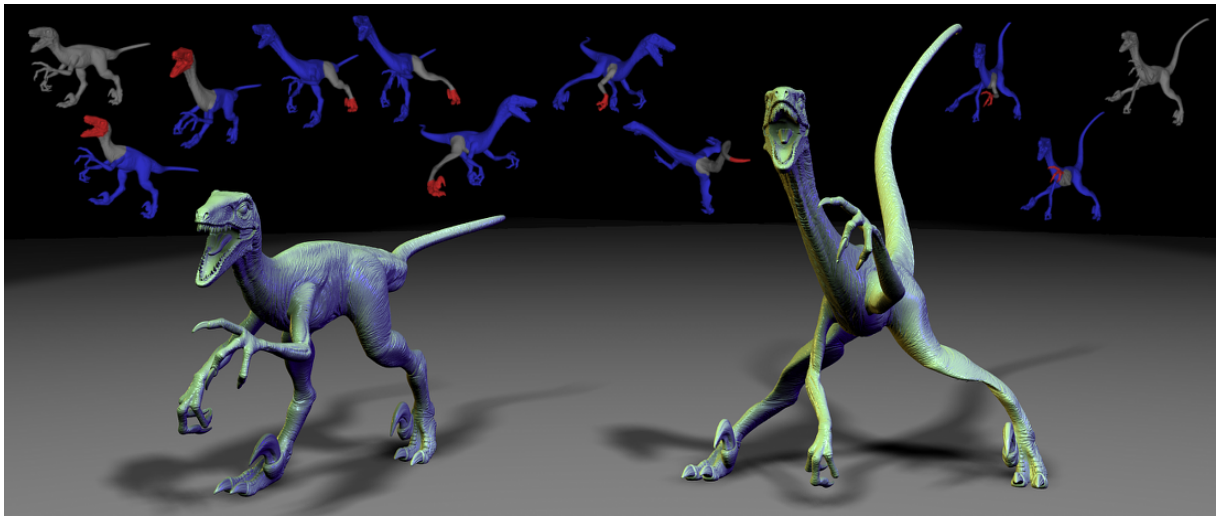

Figure 1: *Interactive freeform deformation of the Raptor model (8M triangles). The initial geometry (left) is adaptively down-sampled through a streaming process, to get a simplified point-based version (146k points) that can be edited interactively with on-demand local refinement (blue snapshots). Afterwards, a second streaming process performs a feature preserving deformation reconstruction to the original geometry (right).*

tion issues, experimental results, current limitations and potential extensions are discussed in the last two sections.

2. Related Work

Interactive Out-Of-Core Rendering: Rusinkiewicz and Levoy [RL00] have developed the first environment for interactive visualization of gigantic objects: *QSplat* is a point-based rendering system using an out-of-core bounding sphere hierarchy to achieve progressive refinement of a point-sampled object. Later, real-time performances where reached by Cignoni et al. [CGG*04]: *Adaptive Tetra-Puzzles* is a diamond-based hierarchical space decomposition for polygonal meshes, with a cache-coherent structure from the hard-drive to the GPU. Since many neighboring samples of a gigantic object are often rendered over a single pixel, Gobetti et al. [GM05] have developed the *Far-Voxel* system, a hybrid mesh-point rendering system, where a cluster of samples can be rendered as a single point, with a specific shader approximating the appearance of the underlying surface area. To improve the rendering quality, additional parameters may also be stored with cache-coherent out-of-core structures, such as the irradiance [CB04] or the texture [BS06]. Note also that the specific issue of terrain visualization also benefits from out-of-core techniques. However, as mentioned above, all these systems are designed for static objects only, which prevents their use in our context where interactive shape editing of gigantic objects is requested.

Freeform Deformation: In the field of geometric modeling, FFD encompasses a large family of techniques where intuitive constraints are interactively applied to a shape. Earlier FFD techniques were based on a 3D space deformation induced by a dummy object (e.g., lattice, curve, mesh): the user moves the vertices of this dummy object, inducing a smooth space deformation applied to the embedded shape (see [Bec94] for a survey). Such a dummy object is no longer required with recent FFD techniques, where the deformation is directly defined on the initial shape: a part of the surface is frozen, another one is moved, and the shape of the remaining part is smoothly deformed using some fairness constraints. Most recent methods formulate surface deformation as a global variational optimization problem [BK04, SLCO*04, YZX*04, BPGK06], although the same interface can be used with space deformation [BK05]. These techniques offer a precise control of the deformation area (arbitrary boundaries) but remain less efficient than multiresolution editing tools [ZSS97, LMH00]. Recently, Pauly has proposed a point-based multi-scale shape deformation [PKG06], which provides a useful scale decomposition and editing, but is not adapted to large geometries. A comparative presentation of recent deformation techniques can be found in [Sor06] and [BS07].

3. A Sampling-Reconstruction Framework

Editing the shape of large objects imposes one strong constraint on the software architecture: as the whole object is potentially too large to fit the in-core memory, the only truly generic and efficient implementation is to perform *out-of-core data streaming*. Moreover, as interactive shape editing typically involves semi-random access to the database, specific in-core data structures have to be designed, that allow efficient *dialog* with the out-of-core object, in the spirit of recent work by Isenburg et al. [IL05, ILSS06]. However, in order to avoid painful editing sessions, only a reduced number of streaming passes should be performed, since each pass may take several minutes for large objects.

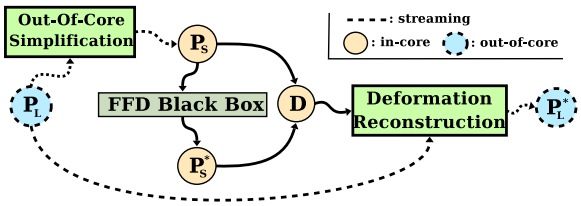

Figure 2: *Our Sampling-Deformation-Reconstruction framework for interactive shape editing of large objects.*

The key idea developed here is that the shape of a dense object can be precisely edited through the interactive deformation of a simplified in-core version. Thus, we propose the three-fold *sampling-deformation-reconstruction* system described in Figure 2:

- *Sampling:* an efficient adaptive out-of-core downsampling is performed on the original large object P_L during a streaming process, to get a simplified version P_S with a size compatible with interactive FFD (see Section 4).
- *Deformation:* an interactive FFD session is then applied on P_S to obtain a deformed simplified object P_S^*. During this session, local upsampling may be achieved at any time, when additional precision is required for a given manipulation.
- *Reconstruction:* Finally, another streaming process performs an accurate, feature preserving deformation reconstruction from P_S^* to P_L, in order to generate the final deformed large object P_L^* (see Section 5).

This three-fold approach exhibits several nice features:

Size-independent interactivity: Interactivity is an essential property of any FFD tool since the target shape is not precisely known a priori and is usually reached by interactively exploring the space of possible shapes. Our system offers interactivity, for any size of the initial object, by performing adaptive downsampling to fit the workstation and FFD software capabilities. Both the sampling and the reconstruction steps work in streaming and only involve local and linear processing, which guarantees memory and computation efficiency.

Pure point-based processing: as mentioned earlier, 3D acquisition is one of the main sources for large objects. For such models, only greedy reconstruction techniques can be used in practice, and it is well known that such algorithms do not provide strong guarantees about the consistency of the resulting topology. To overcome possibly non-manifold input data, we have chosen to simply ignore the underlying topology, and only employ point-based techniques [AGP*04], which allows our system to process unorganized point clouds or meshes in a similar way, both for data input and data output.

Compatibility with arbitrary FFD tools: as recalled in Section 2, a rich palette of FFD methods has been proposed during the last twenty years; each of them has specific strengths and weaknesses and each CG designer has her own preferences among them. To preserve this variety, and allow to combine several method in the same session, we do not impose any particular FFD method, but rather propose a surrounding system allowing the use of **any** interactive FFD tool with large objects. Our system considers the FFD step as a black box between P_S and P_S^*. It only requires a one-to-one correspondence between the samples of the simplified model P_S and those of its deformed version P_S^*, which is trivially provided by any deformation tool.

On-demand local refinement: the two streaming algorithms are available during the interactive session, and allow to build a simple multiscale FFD system. Whenever the user requests an improved precision of a specific area, our system performs efficient local upsampling of the in-core model by fetching additional samples from the original geometry and applying to them the deformation defined so far.

Having all these features combined, our system can be considered as the first *interactive out-of-core multi-scale freeform geometric modeling system*, compatible with arbitrary FFD tools.

4. Adaptive Out-Of-Core Simplification

The first step of our system aims at efficiently generating a convincing simplification of the original, possibly gigantic, model during a streaming pre-process. As mentioned earlier, topology inconsistencies often present in such objects lead us to work in a point-based context. One elegant way to simplify either meshes or point clouds is *vertex clustering* [RB93, Lin00, PGK02, SW03]. Such methods work out-of-core and are fast enough for integrating them in interactive tools. The basic idea is to generate an hierarchical partitioning of the space embedding the object, and compute one representative sample for each partition, using various error metrics to control the hierarchy depth. At the end of the process, the set of representative samples can be considered as point-based downsampling of the original object.

The adaptivity and the efficiency of vertex-clustering simplification algorithms strongly relies on two key elements: the *space partitioning structure* (e.g. 3D grid, BSP, octree) and the *error metric* (e.g. Quadric L_2, $L_{2,1}$, etc). For instance, Schaefer and Warren [SW03] have used octrees combined with a quadric error function defined over the original geometry. Their algorithm can be used with large objects thanks to a dynamic split-collapse function over the octree structure. However, Boubekeur et al. [BHGS06] have recently shown that the Volume-Surface Tree (obtained by the combination of an octree at coarse level with a set of quadtrees at finer levels) offers a better adaptive vertex clustering than octrees, as it requires less samples for an equivalent error bound.

Building on all that work, we propose a new out-of-core vertex clustering algorithm. Rather than using one global dynamic structure for the whole streamed geometry, we propose to use *spatial finalization* [ILSS06] for maintaining a low memory footprint: we generate a set of temporary VS-Trees structured in a coarse 3D grid and use them to locally

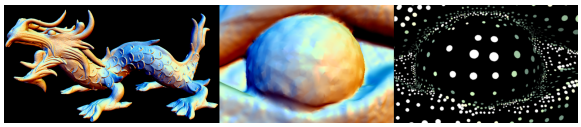

Figure 3: ***Left:*** *XYZRGB Dragon (7.2M triangles).* ***Middle:*** *Close-up of the eyeball on the original geometry.* ***Right:*** *Adaptive downsampling by out-of-core VS-Tree clustering.*

downsample the surface. This is much faster than managing one global data structure and behaves particularly well with gigantic objects which exhibit strong *acquisition coherency*, as mentioned by Isenburg et al. [ILSS06].

Algorithm: Our simplification process uses two streaming passes:

1. *First pass:* all the samples of P_L are streamed through a coarse 3D grid G. During this pass, each cell of G simply counts the number of samples falling in it and keeps the counter for the second pass (a preliminary pass is required if the bounding box is not known).
2. *Second pass:* all the samples are streamed again through G. This time, each read sample is stored in the intersected cell C, and the counter of this cell is decreased. Once the counter of C drops to zero, we know that there are no additional samples belonging to C in the remaining stream. So, we simplify the set of samples stored in C by using VS-Tree clustering, as introduced in [BHGS06]: the set is recursively split in 3D using an octree, and as soon as a surface partition can be projected onto a plane without folding, the set is recursively split in 2D using a quadtree, until reaching a user defined error bound. The resulting set of representative samples is then concatenated to P_S, and the content of C (i.e. original set of samples and VS-Tree) is discarded.

As this process provides a point-based downsampling of the original objects, it can be directly employed for applying point-based deformation techniques. On the other hand, if a mesh-based FFD tool has to be used, a mesh topology has to be created on P_S. But, as the purpose of this mesh is only for temporary application of a mesh-based deformation, even a low quality mesh is sufficient. It can be obtained either by simply reindexing input triangles in the case of polygon soups [RB93, Lin00], or by quick-and-dirty meshing techniques [LP03, BRS05] in the case of point clouds or indexed mesh vertices. Alternatively, one could also use the Streaming Mesh Format [IL05] of Isenburg et al. to reindex indexed meshes.

In practice, we have observed that only a small fraction of the original surface has to be present at the same time in the main memory. This footprint ranges from 10% for million sized objects to less than 1% for billion sized ones. When the large objects are provided with both positions and normals for each sample, we can use a product of the L_2 and $L_{2,1}$ error metrics [CSAD04] to drive the VS-Tree clustering. Otherwise, a simple density measure is used, and an aggressive flatness test replaces the volume-surface transition predicate [BHGS06]. When P_L is a point-cloud, the normals of P_S are estimated using a PCA on the local neighborhood [HDD*92]. The lower-dimensional structure of the VS-Tree (i.e, 1-for-4 split in the average tangent plane) significantly reduces the number of final representative samples for a given error bound, thus also reducing the whole processing time. The resolution of the coarse grid G is user-defined, ranging in our tests from 16^3 to 128^3 according to the size of P_L. Usually, a large number of cells speeds up the simplification process, while a small number improves the adaptivity of P_S (see Section 7). Fortunately, the number of intersected cells of G is several order of magnitude smaller than the usual target resolution for P_S, avoiding visible over-clustering artefact in the case of large grid resolution.

Note also that for future local refinement that may be required during the interactive editing session (see Section 6), we keep, for each cell C, two records about this preprocessing:

- The starting and ending indices of the samples that belong to C in the input stream; this will enable partial streaming for local refinement of the cells intersecting the area marked to be refined.
- The VS-Tree structure; this avoids error computation and recursive split for levels already tested during the initial simplification. Too keep a negligible memory overhead, the tree is not cached when it is too deep and will be rebuilt from scratch if upscaling is required.

Figure 3 illustrates the downsampling quality from P_L to P_S obtained with our approach. Compared to octrees, we have observed a gain of 15% to 25% both in time and memory, which is a significant benefit in the context of gigantic objects. This gain is explain by the local 2D partitioning, which reduces over-clustering and intersection test cost. The divide-and-conquer structure offered by *spatial finalization* makes this algorithm particularly well adapted to multi-core CPUs, which are becoming more common in current workstations.

5. Streaming Deformation

The second out-of-core streaming process of our system takes place after the FFD interactive session, where the initial point-based simplified geometry P_S has been transformed into its deformed version P_S^*. The goal of this section, is to explain how to efficiently and accurately transfer this deformation to the original gigantic object P_L, to get the final deformed object P_L^*.

Point Sampled Deformation Function: By manipulating one or several deformation tools during the interactive FFD session, the user implicitly defines a continuous space deformation function D. But, as the FFD step is considered as a black box by our system, the actual function D is unknown. However, the simplified geometry P_S and its deformed version P_S^* form a discrete displacement field $\{p_i, p_i^*\}$ which

Figure 4: ***Left:*** *Initial in-core geometry obtained by VS-Tree clustering.* ***Middle:*** *Deformed in-core geometry after an interactive FFD session.* ***Right:*** *Discrete displacement field generated by linking the initial (green extremum) and the final (red extremum) position of each sample point. This displacement field performs a sampling of the continuous space deformation function at the scale at which it has been edited.*

can be interpreted as a point sampling of the continuous function D (see Figure 4). Therefore, our goal is to reconstruct the continuous function D from the discrete field $\{p_i, p_i^*\}$ in a very similar way as one would reconstruct a surface from a point sampled geometry. Since smooth function reconstruction methods are slow and too global, which make them prohibited in the context of large objects, we propose a purely local method, based on a *normal displacement* representation combined with a new efficient point-based coordinate system.

Average Plane and Normal Displacement: Reconstruction of a continuous function from point samples always involves some assumptions about the smoothness of the initial function between the samples. In our case, the simplified point-based geometry P_S has been computed from P_L by VS-Tree clustering which includes several geometric error bounds to guarantee a reasonable smoothness of P_L between neighboring samples of P_S. In other words, P_S can be considered as a low-pass filtered version encoding large-scale geometric components of P_L, while the difference P_L-P_S encodes fine-scale geometric details. Following recent work on mesh encoding [LSLCO05, KS06], we propose to encode the details in the normal direction of some local average plane H. So each sample $p \in P_L$ can be expressed as:

$$p = p' + d \cdot n,$$

where n is the normal vector of H, p' the orthogonal projection of p on H and d the signed distance from p to H (see Figure 5). Note that efficient computation of the average plane H is vital, as it has to be done for each sample $p \in P_L$. Moreover, the variation of H from one sample to its neighbor should be smooth as it is not supposed to include geometric high frequencies.

Since we use a point-based representation, we do not have any explicit neighborhood information to compute H. Nevertheless, a conservative set of neighbors can be collected with a *k-nearest-neighborhood* $N_k(p)$ which must be large enough to offer correct low-pass filtering for the variation of H, but also small enough to remain accurate in areas of large curvature. In practice, we use $k \in [10,30]$ according to the density of P_S. Note that $N_k(p)$ can be efficiently computed using a static kd-tree on P_S, generated once for all just before streaming P_L. We propose to compute H using a local *partition of unity* fitler weighted by an Hermitian kernel:

$$n = \frac{\sum_{i=1}^{k} \omega(p,q_i)\, n_i}{\sum_{i=1}^{k} \omega(p,q_i)} \quad \text{with}$$

$$\forall t \in [0,1] \quad h(t) = 1 - 3t^2 + 2t^3$$

$$\omega(p,q_i) = h\left(\frac{|p-q_i|}{\max_k(|p-q_k|)}\right).$$

The same kernel is also used to compute the center of H. The low-pass filtering can be intuitively controlled by the user, by increasing or decreasing the geometric extent of the kernel.

Projected barycentric coordinates: Similarly to $p \in P_L$, any point $p^* \in P_L^*$ can also be expressed as a normal displacement relative to an average plane H^*:

$$p^* = p'^* + d^* \cdot n^*.$$

Reading this equation backwards provides a simple algorithm to transfer deformation from p to p^*:

1. compute H from local neighborhood of p in P_S
2. compute p' and d from p according to H
3. reformulate p' *intrinsically* in P_S
4. reproduce p'^* simply by switching from P_S to P_S^*
5. compute H^* from local neighborhood of p'^* in P_S^*
6. compute d^* from d, accounting for a possible scale factor
7. finally, compute $p^* = p'^* + d^* \cdot n^*$

Differential representations [SLCO*04, LSLCO05] provide an elegant solution to obtain rotation-invariant intrinsic coordinates. Kraevoy and Sheffer [KS06] introduce the *pyramidal coordinates*, where p' is replaced by its *mean value coordinates* [Flo03] in its 1-ring-neighborhood. However, these solutions require explicit topology and remain computationally expensive, which makes them prohibitive in our context.

We propose an alternative intrinsic encoding which can be considered as an approximation tuned for efficiency, more suitable in the context of gigantic objects. Let us consider a 3-neighborhood $T(p) = \{q_i, q_j, q_k\}$ for p and its projection $T'(p) = \{q'_i, q'_j, q'_k\}$ on H. We can use the projected barycentric coordinates $B(p) = \{b_i, b_j, b_k\}$ of p' in the triangle $T'(p)$ as intrinsic coordinates of p on H (see Figure 5). $B(p)$ can be

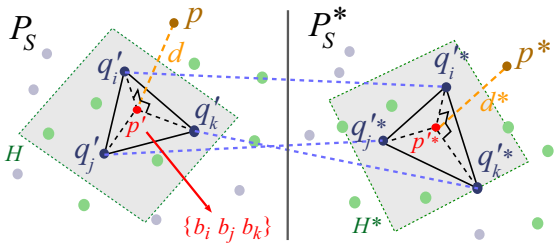

Figure 5: *Projected barycentric coordinates.*

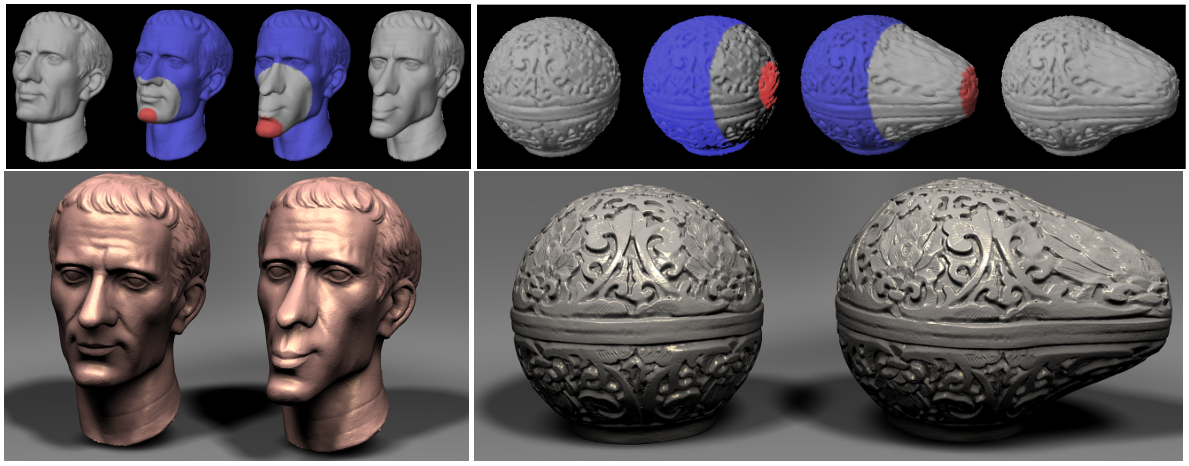

Figure 6: ***Left:*** *Julius Caesar (800k triangles) interactively deformed using a 30k downsampling.* ***Right:*** *Bumpy Sphere (1.4M triangles) interactively deformed using a 25k downsampling. Note that both versions exhibits small scale features which are strongly blurred on the simplified version during the interactive session, but are adequately reintroduced by the deformation reconstruction on the original geometry.*

directly computed from p using the fast evaluation (49 ops) proposed by Heidrich [Hei05]. So the switch from p' to p'^* (step 4 of the algorithm) can simply be expressed as:

$$p'^* = T'^*(p) \cdot B'(p)^\top .$$

We now have to face a topology problem: how to correctly select $T(p)$. For symmetry and robustness reasons, $T(p)$ should be as equilateral as possible, and to ensure fidelity of deformation, it should be as small as possible. To efficiently fit these constraints, we need to select neighbors according to their distribution [LP03, GBP05], so we introduce the notion of *angular voting*: the first neighbor $q_i \in N_k(p)$ is selected as the nearest sample to p. This neighbor will discard a conical space C_i starting from p, centered in the direction of q_i and with a solid angle $2\pi/3$. The second neighbor q_j is selected in the subset of $N_k(p)$ contained in $\mathbb{R}^3 \backslash C_i$. This neighbor discards another conical subspace C_j. Finally, q_k is selected in the subset of $N_k(p)$ contained in $\mathbb{R}^3 \backslash (C_i \cup C_j)$. If a test fails (for instance, an empty set in the remaining space, which is frequent on surface boundaries) the angle is divided by two and the selection process is restarted. As a result, this algorithm searches for the smallest triangle centered on p (i.e., capturing accurately the deformation) with large enough edge angles (i.e., increased robustness avoiding numerical degeneration in projected barycentric coordinates).

The last element that we need to define is the signed distance d^* (step 6 of the algorithm). Thanks to the intrinsic coordinates, the local rotation undergone by the geometric details during the deformation has been accounted for, but an eventual scaling has not. So we propose to simply scale the distance d by the size ratio of the surrounding triangles, before and after deformation:

$$d^* = d \cdot r^*/r ,$$

where r (resp. r^*) is the circumcircle radius of $T(p)$ (resp. $T^*(p)$). By putting all the elements together, we obtain the final expression for our reconstructed deformation function D:

$$\forall p \in P_L \quad p^* = D(p) = T'^*(p) \cdot B'(p)^\top + d^* \cdot n^* .$$

Figures 1, 6 and 9 present various examples of the accurate deformation of small features with our fast deformation function.

Streaming Deformation Reconstruction: Once the kd-tree required for neighborhood queries has been set up, the purely local behavior of our reconstructed deformation function D can be performed in streaming, as each sample p is processed individually. Moreover, this per-sample operation can fully exploit multicore CPUs. In order to optimize the cache usage when collecting the local set of neighbor candidates in P_S, spatially coherent input buffers can be built by using the structure induced by G again (see Section 4).

6. Interactive Local Upsampling

One weakness of manipulating downsampled geometry is the fixed scale of deformations, since only the in-core simplified geometry can be used during the interactive session. To avoid this possible issue, our system is able to perform interactive upsampling, whenever the user requests improved precision on a specific area for finer editing. This multi-scale interaction is illustrated in Figure 7 and works as follows:

1. The user selects the area to upscale and a smaller error bound than the initial sampling;
2. Sub-parts of the original objects, corresponding to the index range of each cell of the grid G (see Section 4) intersecting the selection, are streamed.
3. Upscaling is performed by clustering the streams in the cached VS-Trees, set with the new error bound.

Models	Original Size		R/W data	Sampling	Pre-process		Post process
	vertices	triangles			1st pass	2nd pass	
Julius Caesar	387k	774k	8.8 MB	30k pts	0.35 s	0.43 s	1.70 s
Bumpy Sphere	701k	1.4m	16 MB	25k pts	0.81 s	0.98 s	1.32 s
Hand	773k	1.54m	17.6 MB	50k pts	0.81 s	1.11 s	1.41 s
XYZRGB Dragon	3.6m	7.2m	82.3 MB	160k pts	2.25 s	2.98 s	5.06 s
Raptor	4m	8m	91.5 MB	146k pts	2.07 s	3.07 s	5.94 s
Lucy	14m	28m	320.4 MB	202k pts	7.21 s	10.5 s	37.9 s
David	28m	56m	640.8 MB	209k pts	13.3 s	30.2 s	120 s
St Matthew	186m	360m	2.07 GB	189k pts	60.4 s	122 s	338 s
Double Atlas	500m	1g	5.59 GB	270k pts	143 s	775 s	1121 s

Table 1: *Pre-process (adaptive simplification) and post-process (deformation reconstruction) performances for various models. The last two models are processed without normals.*

Each additional representative sample p is then concatenated to P_S, and its corresponding deformed position p^* (computed by applying the deformation defined so far) is concatenated to P_S^*. When using this feature, it becomes quite natural to start with a very coarse in-core geometry P_S to define large scale deformation, and then refine some regions of interest and define more localized deformations. This principle can be used recursively until the desired precision is reached, which reproduces a very similar workflow as the one provided by subdivision surfaces. Note that Figure 7 also illustrates another property of our system: since it is not method-dependent, several deformation tools can be mixed; here, we first used the global deformation tool by Pauly et al. [PKKG03], and then, we switched to an *inflating* displacement tool.

7. Implementation and Results

We have implemented our system on a standard workstation (P4 3.4GHz, 1.5GB RAM, 36GB SCSI and 200GB SATA HD) running GNU Linux 2.6. Table 1 presents the timings of both streaming processes (adaptive simplification and deformation reconstruction) for various models. In all cases, no memory swapping has been observed, thanks to the spatial finalization.

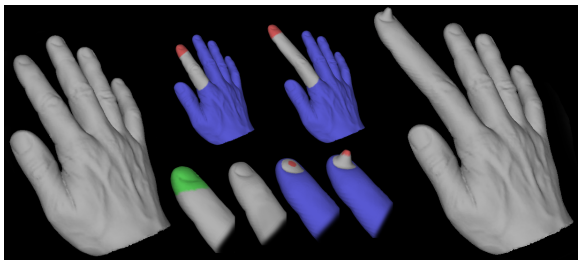

Figure 7: *Out-of-core multi-scale FFD on the Hand model (1.5M tri.).* ***Left:*** *Initial coarse sampling for interactive editing (50k samples).* ***Middle Top:*** *FFD performed at a large scale.* ***Middle Bottom:*** *Local upsampling of the in-core geometry (green area) with our system: the additional samples fetched from original geometry are moved according to the deformation performed so far, and FFD is enabled at a finer scale (125k samples), with the same or another tool.* ***Right:*** *Final multi-scale deformation.*

The pre-processing streaming is mostly bottlenecked by the physical capabilities of the I/O device. Note that, to speed-up processing of very large objects, only vertices are read and streamed: since ordering is preserved in the stream, the final deformed point set remains compatible with the original topology (provided by triangle indices) and point-based FFD can be used safely [PKKG03]. Alternatively, triangles can also be streamed, if polygon-based FFDs are employed.

The final streaming is more computationally intensive. To exploit multi-core architectures, now widely available, multiple threads are used to process the I/O buffer. Note also that the second pass of the pre-process benefits from multi-threading for largest objects, using a specific thread for each active cell of G. In practice, this means that the thread scheduler has to deal with 20 to 200 simultaneous threads. The k-neighborhood queries, intensively used during the deformation reconstruction, are implemented using a static kd-tree, built once, just before streaming P_L. The computation workload is essentially concentrated in this deformation in streaming (see Table 1†.), which emphasizes the use of our approximated but efficient *projected barycentric coordinates*.

† The scalability of our system has been intensively tested on the Digital Michelangelo gigantic models. As there are strong legal restrictions on shape editing for these models, we only present the timings, but not the resulting pictures.

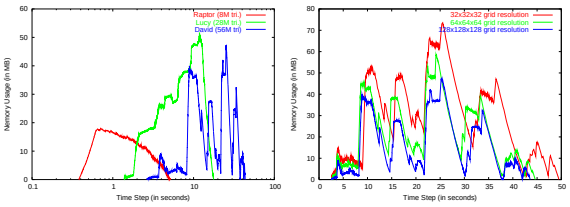

Figure 8: ***Left:*** *Evolution of memory footprint during out-of-core simplification for various models (error bounds have been set to obtain about 200k samples).* ***Right:*** *Influence of the coarse grid resolution on the footprint for the David model (56M triangles).*

The main part of the memory footprint during the interactive session is reduced to P_S plus P_S^*. In all our experiments, the number of samples of P_S has ranged from 20k to 350k, even after multiple local geometry refinement, which is far from saturating the main memory. Note that making P_S itself out-of-core is actually fairly easy as discussed in the next section. Figure 8 measures the evolution of the memory footprint involved throughout the simplification process. It clearly appears that this footprint is largely independent of the model size (both Lucy and David present a memory peak about 50MB). The influence of the grid resolution is more complex. A finer resolution for G reduces the global memory footprint, but as there are more simultaneously active VS-trees, it involves additional over-clustering in areas of low sample density. In practice, we rather advocate a medium resolution for G, around 64^3 in our experiments, which nicely balances sampling quality and reasonable memory consumption.

Many examples provided in the paper have been created using the point-based FFD tool proposed by Pauly et al. [PKKG03] in the *PointShop* environment. We have also experimented our system with *Blender* to provide interactive mesh-based out-of-core editing of gigantic objects (e.g., displacement painting). Figures 1, 4, 6, 7 and 9 show various examples of freeform deformation applied to large or very large geometries, all performed interactively on our standard workstation. Note that on the same workstation, most commercial modeling packages are no longer interactive above one million triangles, and just fail to load Lucy.

8. Discussion and Future Work

We have proposed a *size-insensitive system* to interactively apply FFD techniques to large objects. By size-insensitive, we mean that the in-core memory footprint does not depend on the size of the original object, but rather on the complexity of the user-requested deformation. To our knowledge, this is the first system that permits interactive multi-scale feature-preserving shape editing of gigantic objects, as well as opening the use of costly FFD methods to medium size objects. One main advantage of our system is its ability to directly work with full resolution meshes and point clouds, without requiring any (possibly long and feature-missing) conversion to other representations (e.g, subdivision surfaces). The choice of point-based techniques for both out-of-core streaming processes not only ensures efficiency but also provides flexibility, as they can be seamlessly used on point sampled data or, possibly non-manifold, meshes (topologically inconsistent shapes made of multiple disjoint surfaces, that are quite frequent in CG applications for the entertainment industry).

A variation of our system could be to simplify a model with an arbitrary out-of-core method, edit it, and stream the original samples through a volumetric variational deformation field constructed on the simplified model, such as the RBF method of Botsch et al. [BK05]. Compared to such an approach, our system offers at least two benefits. First, the spatial finalization structure built during our sampling allows to efficiently and locally upsample the model during the interactive session. Second, our deformation reconstruction is fast, avoiding any global variational minimization (for which local editing with displacement painting may require too many constraints), while providing visually accurate and plausible results: in the context of large objects, this speed comes as a key property in a time-scheduled professional context. Lastly, one could considere making a specific FFD method size-insensitive . This is possible, for instance, with volumic deformation fields. However, we believe that large objects should not impose a particular modeling method. Our system is generic, which means that not only arbitrary FFD may be used for editing the object, but also that several can be **mixed** in the **same** session: by only considering the couple $\{P_S, P_S^*\}$, we allow the user to start her work by a globally smooth deformation, then to continue with bone skinning for articulated parts, before ending with displacement painting such as in recent popular 3D tools (e.g., ZBrush). This flexibility, and the possibility to upsample on-demand specific areas are the strengths of our system.

During the development of this system, we have almost systematically traded accuracy for efficiency. Consequently, at least three limitations can be exhibited. First, our streaming deformation reconstruction may cause local self-intersection on highly deformed areas, which is an issue with many existing multi-scale editing techniques. Second, the quality of the initial downsampling strongly influences the smoothness of the final deformation reconstruction. This is one reason for which we have included on-demand local refinement, as it is difficult for the user to predict the number of samples ultimately required during the interactive session. Finally, our system inherits an important limitation of FFD tools: the global topology of the object cannot be edited. Removing this issue is one of our future research directions: an important step in this direction has been shown in [JZH07].

Another valuable extension would be to let the simplified versions themselves become out-of-core. Even though we have not encountered the case, one could imagine that when applying numerous local refinements, P_S and P_S^* would ultimately be too large to fit the in-core memory. Thus, it might be interesting to reuse the spatial finalization to implement a kind of *least-recently-used* (LRU) caching system between the in-core and the out-of-core memory (see our work on texturing [BS06]). This mechanism would be compliant with the usual workflow for interactive shape editing: first applying global deformation on a coarse in-core model, then recursively refining the model to apply more and more localized deformation, that does not involve the whole object.

Acknowledgement: This work was supported in part by the Alexander von Humboldt Foundation. Models are courtesy Stanford University and Aim@Shape.

Figure 9: ***Top:*** *XYZRGB Dragon (7M triangles, interactively deformed with 160k samples).* ***Bottom:*** *Lucy (28M polygons, 300k samples). Each full session with adaptive simplification, interactive editing and streaming deformation took less than 5 minutes.*

References

[AGP*04] ALEXA M., GROSS M., PAULY M., PFISTER H., STAMMINGER M., ZWICKER M.: Point-based computer graphics. *ACM SIGGRAPH Course* (2004).

[Bec94] BECHMANN D.: Space deformation models survey. *Computer and Graphics* (1994).

[BHGS06] BOUBEKEUR T., HEIDRICH W., GRANIER X., SCHLICK C.: Volume-surface trees. *Eurographics* (2006).

[BK04] BOTSCH M., KOBBELT L.: An intuitive framework for real-time freeform modeling. *ACM SIGGRAPH* (2004).

[BK05] BOTSCH M., KOBBELT L.: Real-time shape editing using radial basis functions. *Eurographics* (2005).

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: PriMo: Coupled prisms for intuitive surface modeling. *Eurographics Symp. on Geom. Processing* (2006).

[BRS05] BOUBEKEUR T., REUTER P., SCHLICK C.: Visualization of point-based surfaces with locally reconstructed subdivision surfaces. In *Shape Modeling International* (2005).

[BS06] BOUBEKEUR T., SCHLICK C.: Interactive out-of-core texturing using point-sampled textures. *IEEE/Eurographics Point-Based Graphics* (2006).

[BS07] BOTSCH M., SORKINE O.: On linear variational surface deformation methods. *IEEE TVCG* (2007).

[CB04] CHRISTENSEN P. H., BATALI D.: An irradiance atlas for global illumination in complex production scenes. *Eurographics Symposium on Rendering* (2004).

[CGG*04] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Adaptive TetraPuzzles. *ACM SIGGRAPH* (2004).

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM SIGGRAPH* (2004).

[Flo03] FLOATER M. S.: Mean value coordinates. *CAGD* (2003).

[GBP05] GUENNEBAUD G., BARTHE L., PAULIN M.: Interpolatory refinement for real-time processing of point-based geometry. *Eurographics* (2005).

[GM05] GOBBETTI E., MARTON F.: Far voxels. *ACM SIGGRAPH* (2005).

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *ACM SIGGRAPH* (1992).

[Hei05] HEIDRICH W.: Computing the barycentric coordinates of a projected point. *Journal of Graphics Tools 10*, 3 (2005).

[IL05] ISENBURG M., LINDSTROM P.: Streaming meshes. *IEEE Visualization* (2005).

[ILSS06] ISENBURG M., LIU Y., SHEWCHUK J., SNOEYINK J.: Streaming computation of delaunay triangulations. *ACM SIGGRAPH* (2006).

[JZH07] JU T., ZHOU Q.-Y., HU S.-M.: Editing the topology of 3d models by sketching. *ACM SIGGRAPH* (2007).

[KS06] KRAEVOY V., SHEFFER A.: Mean-value geometry encoding. *International Journal of Shape Modeling 12*, 1 (2006).

[Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. *ACM SIGGRAPH* (2000).

[LMH00] LEE A., MORETON H., HOPPE H.: Displaced subdivision surfaces. *ACM SIGGRAPH* (2000).

[LP03] LINSEN L., PRAUTZSCH H.: Fan clouds - an alternative to meshes. *Workshop on Theoretical Foundations of Computer Vision* (2003).

[LSLCO05] LIPMAN Y., SORKINE O., LEVIN D., COHEN-OR D.: Linear rotation-invariant coordinates for meshes. *ACM SIGGRAPH* (2005).

[PGK02] PAULY M., GROSS M., KOBBELT L.: Efficient simplification of point-sampled surfaces. *IEEE Visualization* (2002).

[PKG06] PAULY M., KOBBELT L. P., GROSS M.: Point-based multi-scale surface representation. *ACM Transactions on Graphics* (2006).

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. *ACM SIGGRAPH* (2003).

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3D approximation for rendering complex scenes. *Modeling in Computer Graphics* (1993).

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: a multiresolution point rendering system for large meshes. *ACM SIGGRAPH* (2000).

[SLCO*04] SORKINE O., LIPMAN Y., COHEN-OR D., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. *Symp. on Geometry Processing* (2004).

[Sor06] SORKINE O.: Differential representations for mesh processing. *Computer Graphics Forum 25*, 4 (2006).

[SW03] SCHAEFER S., WARREN J.: Adaptive vertex clustering using octrees. *SIAM Geometric Design and Computing* (2003).

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. *ACM SIGGRAPH* (2004).

[ZSS97] ZORIN D., SCHROEDER P., SWELDENS W.: Interactive multiresolution mesh editing. *ACM SIGGRAPH* (1997).