GPU based ARAP Deformation using Volumetric Lattices

Michael Zollhöfer†, Ezgi Sert, Günther Greiner and Jochen Süßmuth

Computer Graphics Group, University Erlangen-Nuremberg, Germany

Abstract

We present a novel lattice based direct manipulation paradigm (LARAP) for mesh editing that decouples the runtime complexity from the mesh's geometric complexity. Since our non-linear optimization is based on the ARAP paradigm, it is very fast and can be easily implemented. Our proxy geometry automatically introduces volume-awareness into the optimization problem, leading to more natural deformations. Since we compute how the space surrounding an object has to be deformed to satisfy a set of user-constraints, we can even handle models with disconnected parts. We analyze the bottlenecks of the presented approach and propose a data-parallel multi-resolution implementation on the GPU, which allows to pose even high-quality meshes consisting of millions of triangles in real-time.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—geometric algorithms, languages, and systems

1. Introduction

Computer generated characters are ubiquitous in our modern digital society. In movies and computer games they manage to fascinate us with their unique charm. The purpose of computer animation is to breathe life into these virtual beings. This requires creating all the different poses the individual characters have to adopt to tell a story. Artists usually need a lot of knowledge about the underlying deformation models to accomplish this labor-intensive task. To make the artist's life easier, current research focuses on interactive and intuitive modeling paradigms. Those give the artists the possibility to directly manipulate high-resolution characters using a small number of vertex constraints. Unconstrained parts of the characters should automatically follow the user's input in real-time. A natural and physically plausible look of the deformations is of major importance for the authenticity of the generated animations. This means, that the deformations should be globally consistent and local details of the characters (e.g., the raptor's eye in Figure 1) should be preserved.

1.1. Previous Work

Free-Form Deformation [SP86] allows to model space deformations by manually moving control points of a lattice. Although this scheme is conceptually simple, it is quite hard to model a specific deformation of an embedded object.

In [IMH05], the authors present a handle based approach to deform two-dimensional shapes in a distortion minimizing way. As-rigid-as-possible surface modeling (ARAP) [SA07] minimizes a surface based deformation energy to obtain detail preserving mesh edits. Because the optimization is formulated on the mesh's geometry, interactive modeling

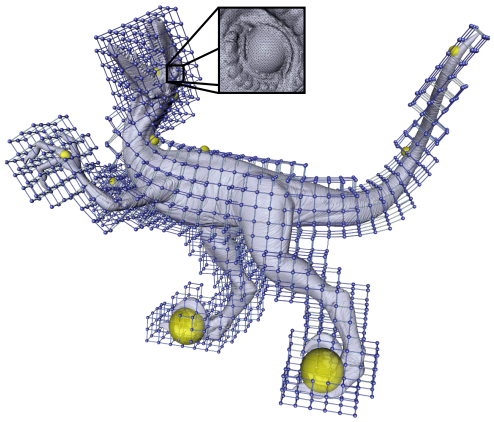

Figure 1: *The proposed LARAP algorithm allows deforming a raptor model with 1.7 million polygons in real-time.*

† michael.zollhoefer@cs.fau.de

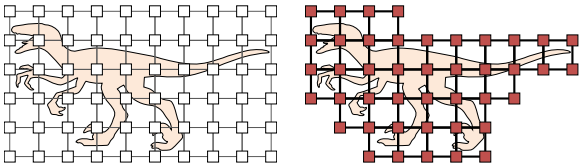

Figure 2: *Grid generation on a uniform 6x10 voxel grid.*

is impossible for complex meshes. The idea of the graph based deformation scheme by Sumner et al. [SSP07] is to decouple the optimization from the mesh's complexity. Primo [BPGK06] and its extension [BPWG07] are based on a decomposition of the object in rigid volumetric cells. As regularizer, the cells are connected by elastic forces. In [ZHS*05] a quadratic optimization problem is solved on a graph given extrapolated local transformations. In contrast, we use the non-linear ARAP energy which optimizes for local rotations and employ a multi-resolution GPU solver.

Most similar to our work is Hybrid Mesh Editing [BHZN10], which applies the as-rigid-as possible paradigm to an automatically generated control cage and uses mean value coordinates as transfer function. In comparison to their approach, ours is volume-aware and allows for direct manipulation. Since we deform the surrounding space, we can easily handle models with disconnected parts (e.g., the girl and the trees in Figure 5). In addition, we propose a data-parallel multi-resolution implementation which allows us to pose even meshes with millions of triangles in real-time.

1.2. Contribution

The proposed LARAP deformation paradigm allows artists to pose high-quality characters in an interactive and intuitive manner. We combine the well-known ARAP approach with automatically generated control lattices to decouple the algorithm's complexity from the complexity of the characters. The regular structure of the lattice allows us to define an efficient multi-resolution approach for solving the optimization problem. To exploit the inherent parallelism, we present a highly data-parallel implementation on the GPU.

2. Algorithm

2.1. Proxy Geometry Generation

A key requirement for interactive mesh manipulation is real-time performance. To achieve this for highly-detailed meshes, we have to decouple the runtime complexity of the optimization problem from the mesh's complexity. This can be done efficiently by computing the optimal deformation on a proxy geometry and transferring it to the original mesh. While in theory arbitrary proxy geometries like cages, skeletons or scaffolds may be used, we decided to use a uniform lattice as it will allow us to quickly solve the non-linear optimization problem in a straight-forward multi-resolution way. Further on, a uniform lattice allows us to simulate solid

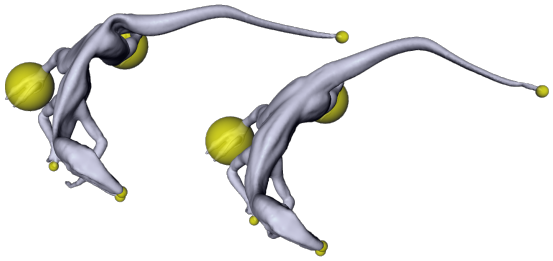

Figure 3: *The volume-awareness of LARAP (right) prevents the surface collapsing artifacts typical for ARAP (left).*

objects, yielding volume-aware deformations. As shown in Figure 2, we place a uniform lattice around the input mesh and then delete all cubes that lie entirely outside the input geometry, yielding a volumetric proxy structure.

Next we need to link the mesh to the proxy geometry. A straightforward approach to link the mesh's vertices to the control lattice would be to express each vertex as tri-linear interpolation of the cube's corners that contains it. However, since this could result in artifacts, we will propose a more thorough scheme for binding the vertices to the control lattice in Section 2.3. For now, let us assume that we can express each mesh vertex $\mathbf{v}_j$ as a linear combination of appropriate control points $\mathbf{c}_i$ of the lattice, i.e., $\mathbf{v}_j = \sum_i \alpha_{i,j}\mathbf{c}_i$.

2.2. Modeling

In an interactive modeling session, the user first selects several vertices of the input geometry that will further serve as handles. If the handles are moved, the associated mesh vertices should move as well and the unconstrained parts of the input mesh should deform in a physically intuitive way. To obtain such a deformation, we use the *as-rigid-as-possible surface modeling* (ARAP) paradigm proposed by Sorkine and Alexa [SA07], which is based on the observation that if an object *locally* preserves its shape as good as possible *everywhere*, the object's *global* deformation will be smooth and plausible. Given an arbitrary graph $\mathcal{G}$ consisting of nodes $\mathbf{c}_i$ and edges e_{ij} in a rest pose and a deformed instance of this graph $\mathcal{G}'$ whose geometric embedding is defined by the nodes $\mathbf{c}'_i$, the ARAP energy at a node $\mathbf{c}_i$ is defined as the portion of the transformation between the local neighborhood of $\mathbf{c}_i$ and $\mathbf{c}'_i$ that cannot be represented by a rigid transformation. By defining the local neighborhood as the one-ring $\mathcal{N}_i$ of the node $\mathbf{c}_i$ and by summing over the local per-node ARAP errors, we obtain an energy function $E(\mathcal{G},\mathcal{G}')$ that measures the plausibility of the deformation from $\mathcal{G}$ to $\mathcal{G}'$:

$$E(\mathcal{G},\mathcal{G}') = \sum_i \sum_{j\in\mathcal{N}_i} \left\| (\mathbf{c}'_i - \mathbf{c}'_j) - \mathbf{R}_i(\mathbf{c}_i - \mathbf{c}_j) \right\|^2, \quad (1)$$

where the $\mathbf{R}_i$ are the rotation matrices that minimize the local deformation energies [SA07]. During interactive modeling, we seek to find the positions $\mathbf{c}'_i$ of the control lattice nodes such that the energy in Equation (1) is minimized – as this

will result in a natural deformation – under the constraints that the control lattice transforms all handle vertices $\mathbf{v}_j \in \mathcal{HV}$ to the positions $\mathbf{t}_j$ defined by the user. Since we can express each vertex as a linear combination of lattice points, this can be stated as follows:

$$\sum_i \alpha_{i,j}\mathbf{c}'_i = \mathbf{t}_j \quad \forall \mathbf{v}_j \in \mathcal{HV}. \tag{2}$$

Since we may have more constraints than unknown lattice points $\mathbf{c}'_i$, the above problem may be over determined. To be able to solve the problem in general, we relax the problem and solve for the constraints in a least squares sense as well:

$$E_{\text{larap}}(\mathcal{G},\mathcal{G}') = \gamma E(\mathcal{G},\mathcal{G}') + \sum_{\mathbf{v}_j \in \mathcal{HV}} \left\| \sum_i \alpha_{i,j}\mathbf{c}'_i - \mathbf{t}_j \right\|^2, \tag{3}$$

where γ balances the influence of the regularization and the constraint term (we use $\gamma = 0.1$ for all our examples). Solving Equation (3) for the unknown lattice points $\mathbf{c}' = \{\mathbf{c}'_i\}$ requires solving a non-linear optimization problem in the unknowns $\{\mathbf{R}_i\}$ and $\{\mathbf{c}'_i\}$. Fortunately, similar to [SA07], the solution can be found using an iterative flip-flop optimization, where in one step the grid points are kept fixed and the energy term is minimized for the unknown rotations $\{\mathbf{R}_i\}$ using SVD (see [SA07] for details). Then we solve for the grid points $\{\mathbf{c}'_i\}$ that minimize the energy in Equation (1) for fixed $\{\mathbf{R}_i\}$. Since E_{larap} is quadratic in the $\{\mathbf{c}'_i\}$, the optimal grid points can be found by solving the linear system

$$(\gamma\mathbf{L} + \mathbf{B}^T\mathbf{B}) \cdot \mathbf{c}' = \gamma\mathbf{b} + \mathbf{B}^T\mathbf{t},$$

where $\mathbf{L}$ is the uniform Laplacian of the lattice, $\mathbf{B}$ is a matrix containing the constraints from Equation (2) as rows, $\mathbf{t}$ is the vector containing the handle positions and $\mathbf{b}$ is a vector whose ith row is $\sum_{j \in \mathcal{N}_i} \frac{\mathbf{R}_i + \mathbf{R}_j}{2}(\mathbf{c}_i - \mathbf{c}_j)$. Since the weights are local, the matrix $\mathbf{B}$ is sparse. Thus the system can be solved efficiently using a sparse solver for semi-definite systems. By iteratively solving for the $\{\mathbf{R}_i\}$ and then for the $\{\mathbf{c}'_i\}$, interactive modeling is usually possible with 3-8 of these steps.

2.3. Preliminary Results

When comparing our method to the original ARAP algorithm, one apparent advantage of our approach is that we decouple the complexity of the deformation from the tessellation of the input geometry, which allows us to deform high quality production meshes in real time. Furthermore, since ARAP only aims at preserving the surface, unnatural folds may occur when bending the surface (Figure 3). We can easily prevent such artifacts by using a solid cube lattice.

Using simple tri-linear weights (i.e., encoding each vertex with respect to the surrounding lattice cell) for transferring the deformation from the control lattice onto the input geometry results in a piecewise linear deformation field which is only C^0 continuous across cells as can be seen in Figure 4. While this is usually sufficient for real-time editing, generating the final high quality poses requires a more elaborate

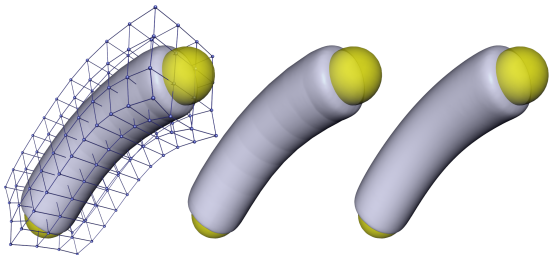

Figure 4: *Comparison between tri-linear (middle) and B-Spline (right) interpolation.*

weighting scheme. Therefore, we propose to use quadratic B-Spline weights when exporting the manipulated models, as this results in a C^1 continuous deformation field. Each vertex is then encoded with respect to the 27 grid points of the surrounding cells. Note that using B-Spline weights requires all cubes that contain a vertex to have a complete set of neighboring cubes. This can be guaranteed by dilating the set of marked voxels during cage generation.

3. GPU based Implementation

The decoupling of the optimization problem from the mesh's complexity already results in an enormous speed-up. However, a closer analysis of the proposed algorithms reveals that many parts of the presented algorithms can be computed entirely in parallel – namely the computation of the SVDs and right-hand sides for each lattice point and the interpolation for each model vertex. We utilized this observation and implemented the entire algorithm on the GPU using CUDA. In each flip-flop iteration, we first compute the optimal rotations for each control point's one-neighborhood in parallel. Then we update the right-hand side using the newly computed rotations. In the last step of the flip-flop iteration, we compute new positions for the lattice's control points using our parallel linear solver on the GPU. Because of the sequential dependence of these three steps, we have to synchronize between the corresponding kernel calls. We solve the linear system using either a parallel Gauss-Seidel like solver or a parallel gradient descent. New improved positions of neighbouring control points are used as soon as they are available. This depends on the scheduling of the threads on the GPU. After a user-defined number of flip-flop iterations has been performed, we use an interpolation kernel to transfer the deformation of the lattice onto the input geometry.

The performance of the non-linear LARAP optimization can be improved even more by solving the problem in a multi-resolution manner. Starting from the finest control lattice, we create a hierarchy of lattices by always joining 8 adjacent cubes. Each lattice is then encoded w.r.t. the next coarser cage. We first solve for the deformation of the coarsest cage, transfer it onto the next finer cage and use the resulting positions as the starting point for another LARAP optimization, and so on. All these operations are performed

Properties			CPU				GPU			
Model	Polygons	Control Points	SVD/RHS	Solve	Interpolation	$\sum$	SVD/RHS	Solve	Interpolation	$\sum$
Raptor	17k	10k	9.2	13.4	0.6	71.2	1.2	11.7	0.6	44
Raptor	170k	10k	9.4	12.6	6.6	75.6	1.2	11.4	0.8	44
Raptor	1.7M	10k	9.4	13.3	68	142	1.2	11.2	1.7	45
Dragon	2M	5k	6.1	7.7	79	122	0.9	7.3	1.7	31
Dragon	2M	20k	28	43	85	303	1.5	30.0	2.0	103
Dragon	2M	40k	53	85	84	504	2.0	68.6	2.4	222

Table 1: *Timings: Comparison of the CPU and GPU implementation of our deformation paradigm (in* ms*).*

in a data-parallel manner on the GPU, we can even map the deformed model directly to the rendering pipeline.

4. Results

We tested our algorithm on the different (multi-part, polygon soup, high detail) models shown in Figures 1 and 5. A summary of computation times on the CPU and the GPU is given in Table (1). All timings were measured on a Core i7 860 CPU (using 8 threads) with an NVidia GeForce 580 GPU. Note that the timings refer only to solving on the finest hierarchy level. $\sum$ denotes the total time for solving the non-linear optimization (3 flip-flop steps with 800 Gauß-Seidel like iterations each, data transfer and interpolation).

Real-time edits using the proposed multi-resolution solver can be found in the accompanying video. When using the multi-resolution solver, we use 3 flip-flop iterations with 200 Gauß-Seidel like iterations on each hierarchy level. Multi-resolution solving takes in total 71ms for the 2M faces Dragon model using a cage with 40k cubes in the finest lattice, which is another 300% speedup.

5. Conclusion

We introduced LARAP, a novel paradigm for interactive and intuitive mesh editing. Using a simple lattice as proxy geometry decouples the algorithmic complexity from the mesh's geometric complexity. Since the algorithm is based on the simple ARAP optimization loop, it is also easy to implement. In combination with the proposed data-parallel multi-resolution implementation of the non-linear solver, we can interactively deform even high-quality meshes.

In the future, we plan to construct the lattice hierarchy in a topology preserving way (i.e., avoid that cubes that were not connected at a finer level are merged) as this will decouple the parts of the model that have a small Euclidean but large geodesic distance in coarse resolution lattices. Additionally, by using an octree and monitoring the deformation error, we plan to locally solve the optimization problem only up the resolution on which the deformation error vanishes.

Acknowledgements

We thank the anonymous reviewers for their helpful and insightful feedback. This work was partly funded by the German Research Foundation (DFG) under grant STA-662/3–1.

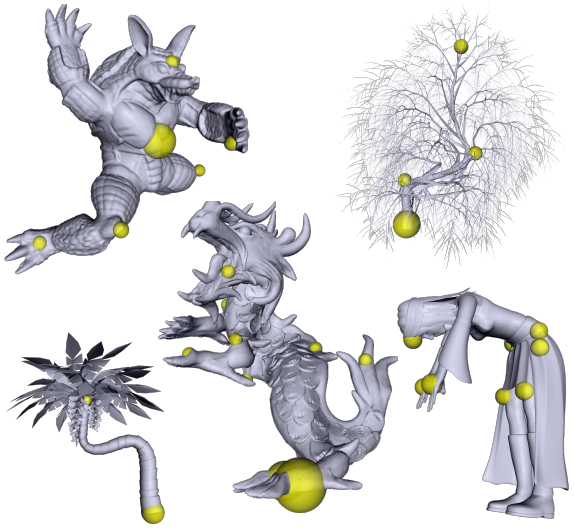

Figure 5: *Poses generated using our interactive mesh deformation tool. The girl and the tree models contain multiple unconnected components.*

References

[BHZN10] Borosán P., Howard R., Zhang S., Nealen A.: Hybrid Mesh Editing. In *EG Short Papers* (2010), pp. 41–44. 2

[BPGK06] Botsch M., Pauly M., Gross M., Kobbelt L.: PriMo: Coupled Prisms for Intuitive Surface Modeling. In *Proceedings of SGP'07* (2006), pp. 11–20. 2

[BPWG07] Botsch M., Pauly M., Wicke M., Gross M.: Adaptive space deformations based on rigid cells. *Computer Graphics Forum (Proc. EG '07) 26*, 3 (2007), 339–345. 2

[IMH05] Igarashi T., Moscovich T., Hughes J. F.: As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24* (2005), 1134–1141. 1

[SA07] Sorkine O., Alexa M.: As-Rigid-As-Possible Surface Modeling. In *Proc. of SGP'07* (2007), pp. 109–116. 1, 2, 3

[SP86] Sederberg T., Parry S.: Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20* (August 1986), 151–160. 1

[SSP07] Sumner R., Schmid J., Pauly M.: Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (2007), Article 80. 2

[ZHS*05] Zhou K., Huang J., Snyder J., Liu X., Bao H., Guo B., Shum H.-Y.: Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24* (July 2005), 496–503. 2