EUROGRAPHICS '2000 / M. Gross and F.R.A. Hopgood
(Guest Editors)

Volume 19, (*2000*), *Number 3*

An Interactive Approach to Point Cloud Triangulation

Leif P. Kobbelt Mario Botsch

Max-Planck-Institute for Computer Sciences

Abstract

We present an interactive system for the generation of high quality triangle meshes that allows us to handle hybrid geometry (point clouds, polygons, ...) as input data. In order to be able to robustly process huge data sets, we exploit graphics hardware features like the raster manager and the z-buffer for specific sub-tasks in the overall procedure. By this we significantly accelerate the stitching of mesh patches and obtain an algorithm for sub-sampling the data points in linear time. The target resolution and the triangle alignment in sub-regions of the resulting mesh can be controlled by adjusting the screen resolution and viewing transformation. An intuitive user interface provides a flexible tool for application dependent optimization of the mesh.

1. Introduction

3D scanners are becoming the standard source for geometric input data in many application areas like reverse engineering, rapid prototyping, conceptual design, and simulation. As a consequence, the problem of generating high quality polygonal meshes from scattered data points is receiving more and more attention [4, 5]. As measured data from physical prototypes and computed data from virtual prototypes has to be merged in all stages of the typical design process, sophisticated mesh generation techniques should be able to work on *hybrid* input data which consists of a mixture of point clouds, polygons, and maybe even NURBS-patches.

The overall process of converting an unstructured soup of input geometry into a consistent polygonal model requires the solution of several sub-problems. First of all, the amount of data we are typically facing is huge. Point clouds with millions or even tens of millions of samples are no exception if the surface of a non-trivial geometric object is to be represented. This makes some sort of pre-processing mandatory in order to reduce the input complexity for subsequent steps while observing a prescribed approximation tolerance.

Second, the global topology of the object's surface has to be determined, i.e., the neighborhood relation between adjacent parts of the surface has to be derived. This typically requires some kind of global sorting step whose computational complexity strongly depends on the underlying definition of spatial or geodesic proximity.

In a final step the actual surface representation has to be extracted. In our case, the goal is to generate a triangle mesh satisfying some quality requirements like a global bound on the triangle's aspect ratio. Another very important quality criterion for meshes is the adaption of the mesh resolution to the distribution of geometric detail information. In fact, the final mesh extraction is another sampling process which may lead to alias-errors if the sampling density does not adapt to the highest geometric frequencies.

The extremal case are sharp corners in the geometry. Here, the geometric frequency spectrum extends to infinity and hence an extremly high sampling density is required. However, even with very small triangles approximating a sharp feature, the visual geometric similarity in terms of normal vectors (and hence shading) cannot be improved (cf. Fig. 1). The only way to avoid such artifacts is to align the triangle edges in the mesh to the feature lines of the original object.

2. Overview

In this paper we are presenting a new approach to solve the mesh generation problem. The major motivation for developing the underlying techniques is the quest for a robust system that is able to process highly complex and completely unstructured hybrid input data through an intuitive user interface. We achieve this by exploiting the standard functionality of any PC's graphics sub-system. Although the graphics hardware only provides a restricted class of operations on 3D data, the computing performance for these operations is usually much higher than that of the CPU. It turns out that several crucial steps in the mesh generation procedure can

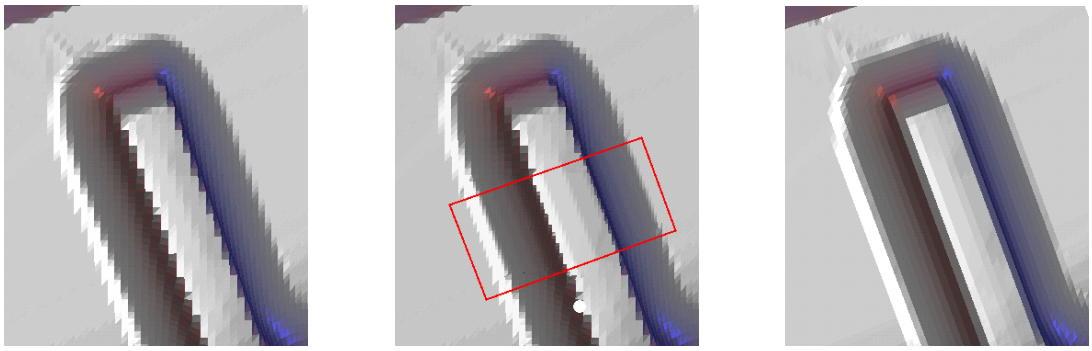

Figure 1: *At sharp feature lines, any sampling algorithm tends to generate alias artifacts (left) which cannot be completely removed by refining the sampling density (center). However, aligning the sample grid to the feature improves the visual quality.*

be mapped to that restricted class of operations and hence can be solved efficiently.

The concept behind the user interface is to simulate a virtual 3D scanning device which is much more flexible than a real 3D scanner. We allow the user to rotate the given geometry on the screen in order to adjust the optimal viewing direction. When taking a profile snapshot (a virtual 3D scan) from that direction, we can mask out an arbitrary sub-region to cut away undesired parts of the currently visible geometry. After performing some post-processing (e.g. mesh smoothing, hole-fixing, ...) we merge the new patch with the already existing ones. After a few iterations of this procedure we end up with a globally consistent model of the given object (cf. Fig. 2).

Taking a 3D snapshot of the object on the screen means reading out the z-buffer contents and un-projecting it back into 3-space. By this we obtain a 3D sample point of the given geometry for every pixel of the screen. Here, we exploit the fact that real 3D scanners usually yield a rather dense cloud of data points which appears as a continuous surface when rendered on the screen. The few remaining holes that might appear occasionally can be removed by simple filter operations in image space. Larger holes in the surface which are due to missing data can be handled by masking out the corresponding regions on the screen. Since the graphics hardware of a standard computer performs the mapping of 3D data points onto the screen much faster than the CPU, we are able to handle data sets with several million samples interactively.

In the context of the introductory remarks, the rendering of sample points into the z-buffer solves all three steps of the mesh generation at once. Sub-sampling is achieved by rendering several points into the same pixel. Actually, the sub-sampling resolution can be controlled by the resolution of the frame buffer. In addition, if sharp feature lines are present in the underlying geometry, we can align them to the horizontal or the vertical axis in order to reduce alias artifacts. The topology of the reconstructed surface trivially emerges from the obvious neighborhood relation between pixels and the resulting surface is extracted by simply interpolating the grid points by a regular triangle mesh. The actual portion of the z-buffer that contributes to the resulting surface patch is determined by some additional criteria such as (relative) quality of individual triangles and user defined image space masks.

The remaining task is to stitch the acquired mesh patches together to build a globally consistent model. This requires to associate the boundary vertices of one patch to the triangles of the other. Again we can accelerate the computation significantly by 'out-sourcing' the necessary operations to the graphics hardware.

The major advantages of our approach are the high flexibility with respect to viewing directions and orientations and the elegant way to sub-sample the given data without building a space partition data structure. The virtual scanning metaphor allows the user to adjust the mesh resolution almost arbitrarily (with the point cloud density being the upper bound) and the possibility to align the sampling grid to geometric features strongly improves the usability of the resulting meshes.

3. Related work

In the literature, different approaches to solve the problem of interpolating a point cloud by a triangle mesh have been proposed. There are two major 'schools' one of which uses 3D Voronoi partitioning as the basic technique and the other one is based on deriving a 3D distance field.

The motivation for Voronoi based approaches is to find the correct topology of the sampled surface even if the samples are scattered very sparsely. The proposed schemes typically come with some bound on the minimum sampling density depending on the local surface curvature.

The classical technique in this field are alpha-shapes [7] which represent a subset of the 3D Delaunay triangulation

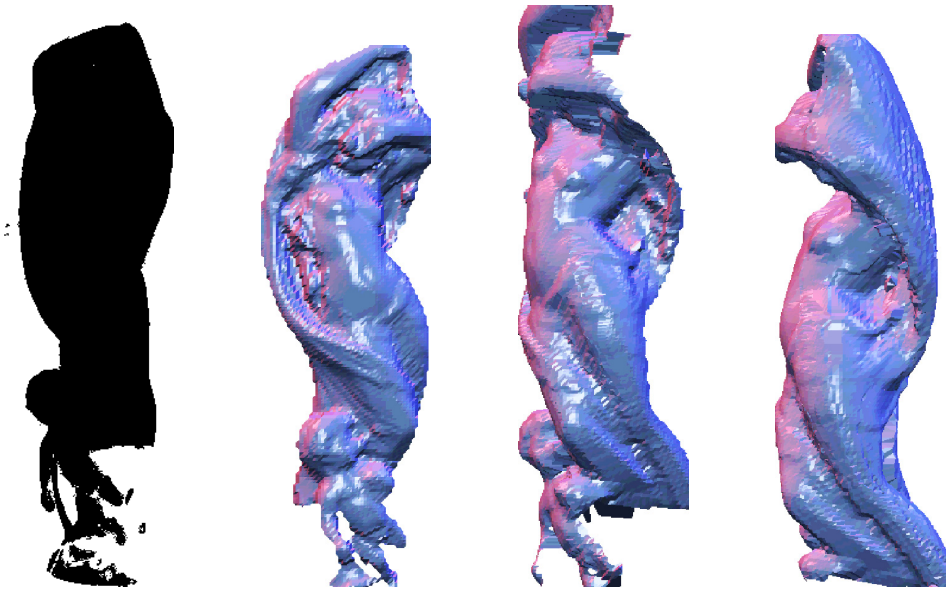

Figure 2: *In an interactive mesh generation session, the user scans parts of the object (the point cloud, left) from different views and merges the pieces to construct a complete mesh model. By using the z-buffer for the virtual 3D scanning, we can handle quite large data sets (here 1.4M points). Each scan prodives a fairly regular re-sampled patch representing a part of the object's surface. Left to right: one, two, and three scans combined. The point cloud on the far left appear as a continous block since the sample density is higher than the pixel size.*

of the sample points. The geometric intuition behind alpha-shapes is to first compute a global Delaunay triangulation and then delete tetrahedra and triangles by using a spherical tool with radius alpha. Variations of alpha-shapes where the value alpha adapts to the local sampling density have been investigated in the sequel [3].

Other Voronoi based techniques use the shape of individual Voronoi cells to determine the surface normal direction at every surface point [1, 2]. This information together with other criteria to rate their plausibility is then used to delete triangles from the Delaunay triangulation. A common feature to these techniques is that they are theoretically sound by guaranteeing correct reconstruction if the bounds on the sampling density are met. However, the requirements in terms of computation time and memory are quite high such that massive data sets with millions of data points cannot be processed with reasonable effort. Also, these techniques tend to be very sensitive to noise which is critical when the data points are measured from a real object.

The distance field approaches usually start by estimating a normal direction for every sample point (e.g. by least squares fitting a plane to some points in the vicinity) [9]. The corresponding tangent plane then is a good local approximation to the original surface and the distance from that plane yields a good local estimate for the true distance. A global distance field can be derived from the local estimates by weighted superposition.

To determine the surface topology one has to find a consistent orientation of the normal vectors, i.e. normal vectors of neighboring sample points should point approximately into the same direction. The topology emerges from the fact that a consistent orientation enables the definition of a signed distance function with negative values below ('inside') and positive values above ('outside') the surface (the object) [6].

The surface extraction is eventually done by sampling the signed distance function on a regular spatial grid and computing the iso-surface for the distance value zero by the Marching Cubes algorithm. The step width in that spatial grid determines the target resolution of the resulting mesh. Since the complexity of the grid increases like $O(h^{-3})$ with decreasing step width h, the highest resolution is bounded by the available memory. As it is well known that the Marching Cubes algorithm can generate triangles with bad aspect ratio, a post-processing of the resulting mesh is usually necessary.

Compared to the Voronoi based approaches, the computation and evaluation of the distance field can be performed quite efficiently for rather complex data sets. However, the mesh resolution cannot be adapted to the local curvature and small detail features tend to be destroyed due to unwanted interference between nearby local distance estimators.

While the existing techniques are off-line algorithms, our approach incorporates user interaction *during* the surface generation. Resolution and orientation of the triangles can

be adapted manually to varying detail levels and quality requirements in different regions of the object (cf. Fig. 3). The CPU and storage requirements are much lower than for the other approaches since no additional data structure has to be generated (like the Voronoi partitioning or the spatial grid of distance samples).

4. Mesh generation

In this section we describe in detail how the various steps in the meshing procedure are solved. The goal is always to identify operations that can be performed by the graphics hardware to exploit its superior computing performance.

During an interactive meshing session the user adds one patch after the other to the previously generated model. One iteration consists of placing, scaling ($\rightarrow$ resolution) and orienting ($\rightarrow$ alignment) the object on the screen, determining the valid region of interest, extracting the patch and automatically stitching it to the already existing mesh.

4.1. Virtual range scanning

When rendering geometry with enabled z-buffer option we always have direct access to the depth of the element that is currently visible at a specific pixel location. While the graphics system uses this information to determine mutual occlusion, our virtual scanner reads the data and un-projects it back into 3-space. The result is a range image containing a 3D sample point for every pixel.

According to the OpenGL convention a 3D point is transformed by the Modelview and the Projection matrix and eventually mapped onto the screen by the Viewport transformation. The inversion of these transformations is straightforward. The only difficulty arises from the rounding step which is implicitly introduced by assigning real coordinate values to integer indexed pixel locations. This causes a global offset for the un-projected geometry. We minimize this effect by un-projecting the pixel centers instead of their lower left corners. So with the Viewport transformation being

$$\begin{pmatrix} i \\ j \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} \\ 0 & \frac{h}{2} & 0 & \frac{h}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

we use the modified inverse

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{w} & 0 & 0 & \frac{1}{w}-1 \\ 0 & \frac{2}{h} & 0 & \frac{1}{h}-1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ z' \\ 1 \end{pmatrix}$$

which corresponds to un-projecting $[i+\frac{1}{2}, j+\frac{1}{2}, z']$ instead of $[i, j, z']$ when finding the sample point for pixel $[i, j]$.

With this simple range scanning emulation, we avoid any complicated handling of geometric or topological special cases, all we have to do is to pass the geometry to the graphics system and let the rendering pipeline do the work. An immediate advantage of this method is that we are not restricted to point clouds. Since we are using the OpenGL API, triangles and more general polygons can be treated the same way. This is important when it comes to merging geometry data from different sources.

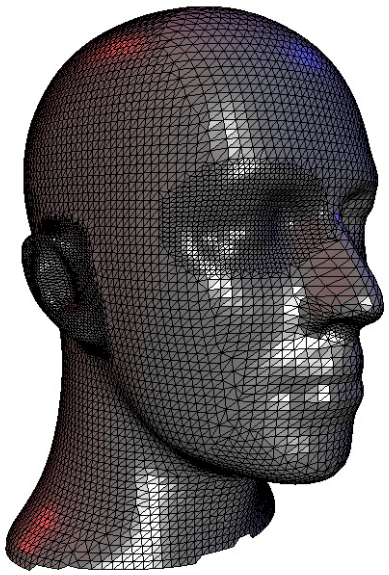

Figure 3: *The interactive approach provides an intuitive interface to locally adapt the mesh resolution to specific requirements for the resulting model by simply zooming the point cloud. Here the ear and the eye have been z-buffer scanned with a higher density.*

A typical application scenario is the conceptual design phase where variations of an object are often manufactured as physical prototypes. Usually we still have a CAD-model of the original geometry which we want to synchronize with the physical model, i.e., we want to add the modified part to the original model. With the z-buffer technique we only have to re-scan the modified portion of the physical model and render it together with the old CAD geometry into the same frame buffer.

Before the actual z-buffer range scanning is carried out, the user has several degrees of freedom to adjust the (pixel) sampling grid. By zooming in on the object, the resolution can be changed (cf. Fig. 3). If the screen resolution exceeds the sampling density such that the number of holes increases, samples can be drawn as large points (`glPointSize()`) covering several pixels at once. For the range geometry represented by the z-buffer contents this corresponds to nearest

neighbor interpolation if antialiasing is switched off. Alternatively, we can try to eliminate holes in the range matrix in a post processing step (cf. Sect. 4.2).

Another degree of freedom is the orientation of the sampling grid. By rotating the object on the screen we can align feature lines of the given geometry to the horizontal or to the vertical axis and thus avoid alias errors that otherwise cause oscillating normal vectors in the vicinity of sharp corners (cf. Fig. 1).

4.2. Post-processing

There are several types of artifacts in the range images obtained by un-projecting the z-buffer contents. Besides the noise in the original point cloud data (caused by the physical range scanner) we can observe additional jitter which is due to discretization errors in the z-buffer. We minimize this effect by automatically placing the front and back planes of the viewing frustum as close as possible to the actual geometry.

Additional enhancement of the range geometry is achieved by applying low-pass filter operations to the z-buffer mask. This can be done very efficiently due to the regular matrix structure. In our implementation we used a locally supported median filter which preserves edges. Edge preservation is important since edges in the z-buffer indicate object boundaries which must not be affected by the filtering (cf. Fig. 4).

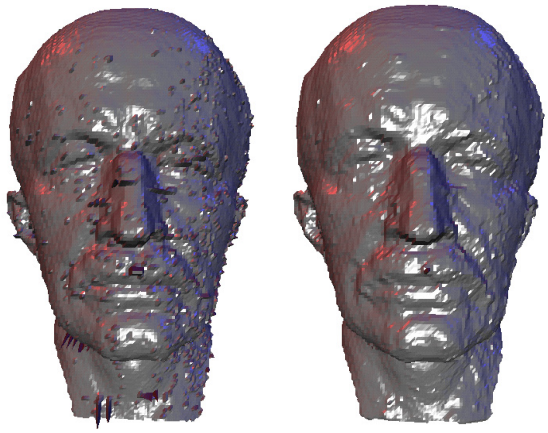

Figure 4: *Noise in the input point data is visible as pimples in the z-buffer mesh (left). Median-filtering removes most of the noise (right) while preserving features — especially the mesh boundaries.*

Besides noise artifacts, the range matrix can have holes if some pixels are not hit by any valid (front facing) geometry during the rendering. If the point cloud is not dense enough (relative to the current screen resolution) some pixels may contain no z-value at all, or they may contain z-values which correspond to the back side of the object. In both cases the holes have to be detected and the missing (or wrong) z-values have to be replaced by averages of neighboring (valid) z-values. This can be done by an arbitrary z-buffer filter operation as long as it does not modify the valid z-buffer entries.

Detecting empty pixels is trivial since the z-buffer is initialized with the back plane's z-value. For the detection of z-values which correspond to the back side of the object, we use the following heuristics. We collect all z-values and quick-sort them in increasing order. Then we look for the maximum difference between two successive elements in the sorted sequence. If samples from the back side are present in the z-buffer then this maximum difference very likely separates front and back points. Hence, deleting all z-buffer entries with larger z-value removes the wrong samples. If no samples from the back side are present, the maximum difference will indicate some z-value near the contour of the visible surface and thus deleting some z-buffer entries will only affect some very badly sampled triangles.

Another possibility to remove back side samples is to let the user place a clipping plane within the point cloud. However, we prefer the heuristics since it does not require any user input and it turned out to be rather reliable. The clipping plane functionality can nevertheless be provided as an additional feature.

Finally, by looking at the differences $\triangle z$ between *adjacent* pixels we can detect badly shaped cells which are seen almost tangentially from the viewing direction. In some applications we might want to mark those regions of the range matrix as invalid. However, we observed that in general it is more intuitive to keep the whole range image as a closed surface and cut out the relevant pieces in the following masking phase.

4.3. Masking

When composing complicated models from several range scans it is often not desirable to use everything that has been visible during the last exposure. The masking phase selects the relevant part of the sampled points and discards the rest. There are two different mechanisms for the masking. One is the user defined *region of interest* and the other is an automatic method which compares the *quality of samples*.

The region of interest is a portion of the screen defined by the user. By blocking all the pixels outside this area with the stencil buffer option we can check what part of the range image to keep. This interaction tool is particularly important if we want to include a bounded patch with higher resolution into an existing mesh. We simply mark the corresponding area on the screen and re-scan a zoomed version of the same point cloud (cf. Fig. 3).

Since we are not deleting any bad triangles in the preprocessing phase we have to guarantee in the masking phase

that bad samples are replaced by better ones in subsequent scanning steps, i.e., a region of the surface that was close to the contour or even occluded in the first scanning steps should be replaced if newer scans provide more reliable shape information.

By convention we always replace the old geometry by the new one unless the sampling quality of the old triangles is ranked superior to the quality of the new geometry. To check this, every new vertex from within the region of interest is mapped to the existing geometry. If there is a correspondence (i.e., the new vertex hits an old triangle) we compare the quality of the new vertex with the quality of the old triangle and keep the better one. If there is no correspondence (i.e., the new vertex does not hit the old mesh) the new vertex actually extends the old geometry and is kept in any case.

The most time consuming step in the automatic masking is the search for triangles of the old mesh which lie closest to a specific vertex of the new mesh. The use of efficient hierarchical space partition data structures is not appropriate in this case since relatively few inquiries are made in each masking step and in the next masking step the old space partition will be obsolete. Instead we use, again, the graphics hardware to accelerate the search.

Every triangle of the old mesh is uniquely indexed by its ID. If we encode this ID as an RGB-color by using the red channel for the most-, the green channel for the mid-, and the blue-channel for the least-significant byte, we can render up to 2^{24} distinguishable triangles into the frame buffer. If we now project and transform the vertices of the new mesh into the same frame buffer, we can determine the corresponding old triangle by simply reading out the RGB pixel color. Occasionally, especially near the contour of the scanned object, the triangle which is found in the corresponding pixel happens not to be the closest one. However in these cases, the found triangle is still a good starting point for a local search.

We still have to define the *sampling quality* of a triangle or vertex. Intuitively the reliability of a sample point increases as the distance between neigboring samples decreases. In fact for a fixed resolution the distance between samples becomes smaller as the viewing direction approaches the surface normal. For a fixed viewing direction the distance between samples becomes smaller if the resolution is refined. Hence, we define the sampling quality of a triangle to be the length of its longest side and the quality of a vertex to be the length of the longest adjacent edge.

Only those pixels from the z-buffer which lie within the region of interest *and* which pass the sampling quality test (*active pixels*), may eventually be added to the already existing mesh in the stitching phase. However, to avoid degenerate triangles and gaps after the stitching, we conclude the masking phase by applying morphologic erosion and dilation operators to the set of active pixels [8].

The erosion operator inactivates pixels that have inactive neighbors. This is done in order to remove isolated pixels. The first dilation then reactivates each pixel that still has an active neighbor. A second dilation further extends the active region. Our experiments show that applying the dilation operator two times is sufficient in most cases.

The effect of the dilation is twofold. First, if boundary vertices of the new mesh are ranked worse than the boundary triangles of the old mesh, their deletion may cause a gap as the meshes are locally no longer overlapping. The dilation prevents this effect. Second, the quality-ranking mask may generate a new patch with a quite jaggy boundary polygon and scattered holes in the interior which causes unpleasant seams after the stitching. The dilation operator tends to smooth out the boundary polygon and close small holes (cf. Fig. 5). Notice that the dilation is not allowed to grow beyond the region of interest.

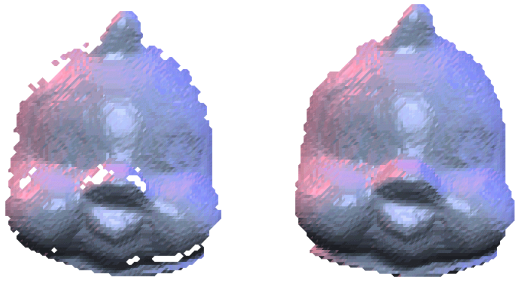

Figure 5: *The new surface patch after masking with respect to the sample quality only (left) has several holes and a jaggy boundary. Applying erosion and dilation fills in the holes and smoothes the boundary (right).*

4.4. Stitching

The final step in the interactive loop is to join the newly acquired geometry with the already existing mesh. Since the new patch has been pre-processed, the task can be solved by a slight modification of the mesh zippering algorithm [14]. We insert the boundary vertices of the new patch into the corresponding (nearest) triangles of the old mesh. We then insert the boundary edges of the new mesh by splitting all edges which cross the geodesic line from one boundary vertex to the next. Finally we remove that portion of the old mesh that is now replaced by the new patch. In order to improve the mesh quality we can post-optimize the seam area by collapsing small edges (cf. Fig. 6) [10].

The only computationally expensive step in this procedure is to find the triangles where the new vertices have to be inserted. This information, however, is already available from the masking phase, where the correspondences between new vertices and old triangles have been established in order to compare the local sampling quality of both meshes.

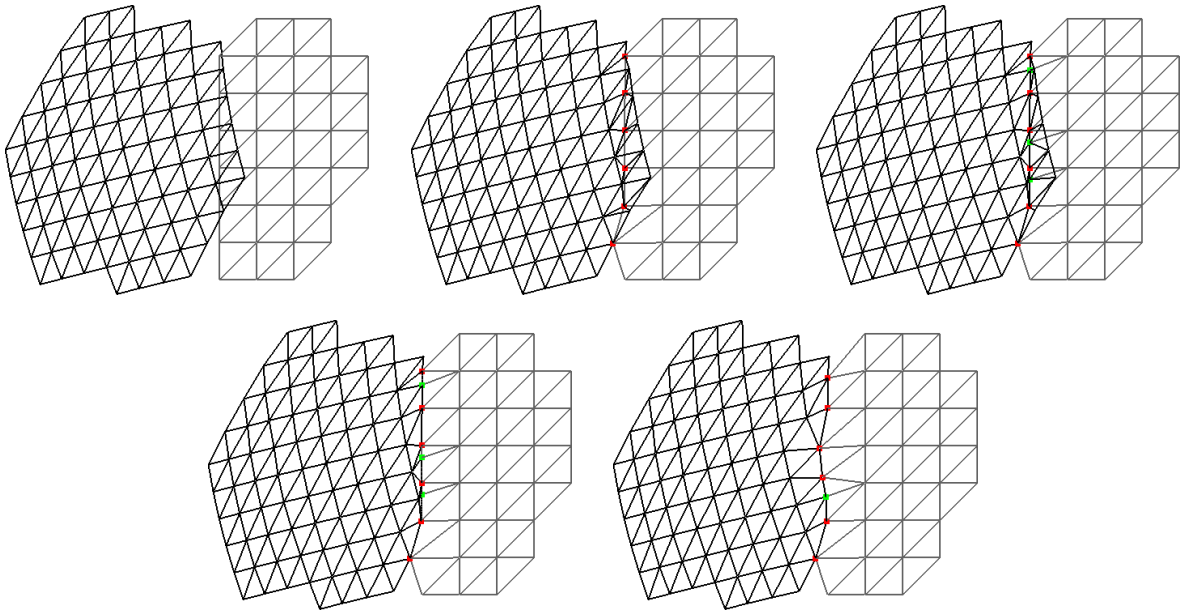

Figure 6: *The stitching algorithm first inserts the new vertices into the old mesh then splits or flips edges of the old mesh to interpolate the new mesh's boundary edges and finally removes the redundant part of the old mesh. Collapsing short edges in the resulting mesh removes badly shaped triangles (from left to right).*

5. Results

We implemented the technique described in the last section as part of an interactive tool for the generation of high quality mesh models from hybrid input data. The user interface merely consists of a viewing window where the input geometry can be examined. Once the desired viewing direction for a virtual range scan in found, the *z*-buffer is read and the un-projection is performed.

In a second window, the resulting mesh is shown and can be examined as well (cf. Fig. 7). For the viewing transformation of both windows a synchronization is enforced at any time which simplifies the decision where to place the next scan. A simple image space drawing tool ("green lasso") is provided by which the user defines the region of interest for the next scan. A similar tool ("red lasso") is available in a different mode to manually delete unwanted regions of older scans. Figure 8 shows the typical sequence of steps that are taken to interactively generate the mesh.

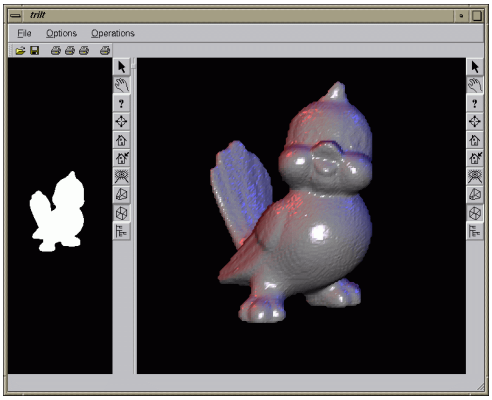

Figure 7: *Minimalistic user interface for the interactive mesh generation tool. In the left window, the original point cloud is rendered (and appears as a solid block). The z-buffer content is extracted, triangulated and displayed on the right. By rotating the right object, the user can easily choose the viewing direction and the region of interest for the next scan.*

6. Conclusions and future work

We presented a new technique for the triangulation of point clouds. The method sub-samples the given data by rendering the data points into an OpenGL frame buffer and unprojecting the contents of the *z*-buffer back into 3-space. The regular grid structure of the range matrix leads to fairly regular meshes. Additional post-processing removes outliers and noise artifacts. A masking step extracts that portion of the data which lies within a user defined region of interest and has a better sampling quality than the already existing scans of the same geometry.

We are planning future work in two directions. First, since our tool is able to process arbitrary (OpenGL compatible) in-

Figure 8: *To generate a mesh model from the club data set (16586 points) we z-buffer scan it from different views. Each new scan is added to the existing model (with the sampling quality mask enabled). When closing the remaining gaps and holes in the last steps, we use the green lasso to define the region of interest where new geometry should be included. The interactive generation of this mesh model took less than a minute. Since the club data set with 16586 point is rather sparse, we used enlarged pixels during rendering to avoid gaps.*

put geometry, we want to apply it to the repairing of CAD models. Often a CAD model goes through many conversion steps causing badly shaped triangles or even topological inconsistencies. Our tool can provide a means to easily resample the given surface geometry such that better shaped triangles emerge. This could be interesting for the preparation of mesh models for numerical simulation.

Second, we want to apply some mesh decimation scheme to the (masked) mesh patches before they are merged with the already existing geometry. Of course, the decimation has to keep sufficient detail in curved areas but it can reduce the mesh complexity in flat regions. This would lead to polygonal models with the mesh resolution changing gradually and not only from patch to patch. The decimation scheme can be implemented very efficiently due to the regular grid structure of the z-buffer matrix (e.g. with adaptive quad-trees).

7. Acknowledgements

We would like to thank Hughes Hoppe for allowing us to use the club and the mannequin head data set and Christian Rössl for generating the Tweety and the statue point clouds.

References

1. N. Amenta. *The Crust Algorithm for 3D Surface Reconstruction*. In Proc. 13th ACM Symp. Computational Geometry, 1997.

2. N. Amenta, M. Bern, M. Kamvysselis. *A new Voronoi-based surface reconstruction algoritm*. Proc. SIGGRAPH '98, 1998.

3. F. Bernardini, C. Bajaj. *Sampling and reconstructing manifolds using α-shapes*. 9th Canadian Conference on Computational Geometry, 1997, 193–198.

4. F. Bernardini, C. Bajaj, J. Chen and D. Schikore. *Auto-*

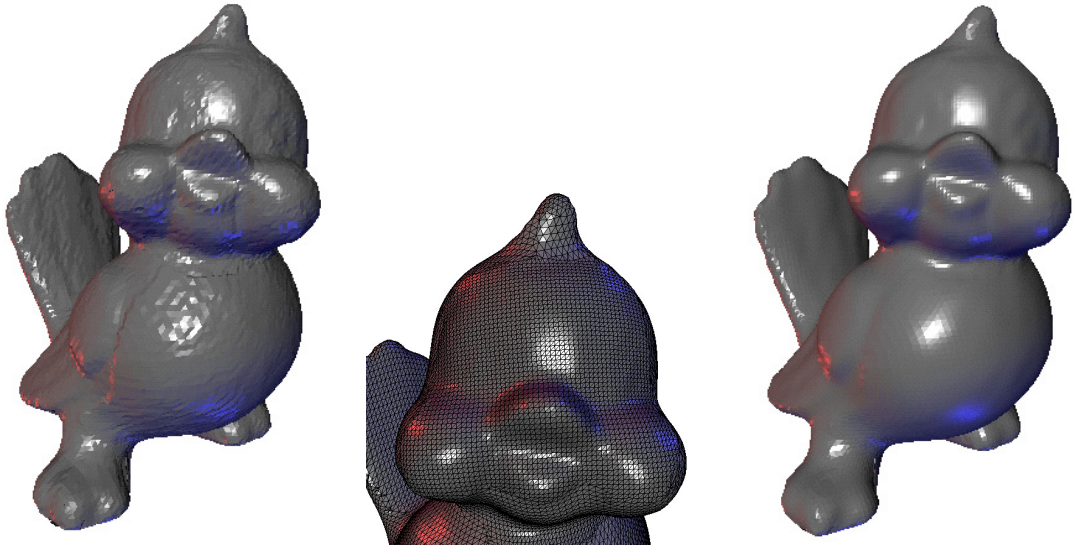

Figure 9: *The Tweety model has been generated from a point cloud with about 400K sample points. The center image shows how the different scans are stitched together. Noise reducing post-processing by a Laplacian filter [12, 11] yields the final result with 33K triangles.*

matic Reconstruction of 3D CAD Models from Digital Scans. Int. J. on Comp. Geom. and Appl. vol. 9, 1999.

5. F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin. *The Ball-Pivoting Algorithm for Surface Reconstruction*. to appear in IEEE Trans. on Vis. and Comp. Graph.
6. B. Curless, M. Levoy. *A volumetric method for building complex models from range images*. Proc. SIGGRAPH '96, 1996, 303–312.
7. H. Edelsbrunner, E. P. Mücke. *Three-dimensional alpha shapes*. ACM Trans. Graphics 13, 1994, 43–72
8. R. M. Haralick, S. R. Sternberg, X. Zhuang. *Image Analysis Using Mathematical Morphology*. In IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol PAMI-9, No. 4, 1987.
9. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Surface Reconstruction from Unorganized Points*. Proc. SIGGRAPH '92, 1992, 71–78.
10. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Mesh Optimization*. Proc. SIGGRAPH '94, 1994, 19–26.
11. L. Kobbelt, S. Campagna, J. Vorsatz, H-P. Seidel. *Interactive Multi-Resolution Modeling on Arbitrary Meshes*. Proc. SIGGRAPH '98, 1998, pp. 105 – 114.
12. G. Taubin, *A Signal Processing Approach to Fair Surface Design*. Proc. SIGGRAPH '95, 1995, pp. 351–358.
13. M. Teichmann, M. Capps. *Surface reconstruction with anisotropic density-scaled alpha shapes*. Proceedings of IEEE Visualization '98, 1998, pp. 67 – 72
14. G. Turk, M. Levoy. *Zippered Polygon Meshes from Range Images*. Proc. SIGGRAPH '94, 1994.

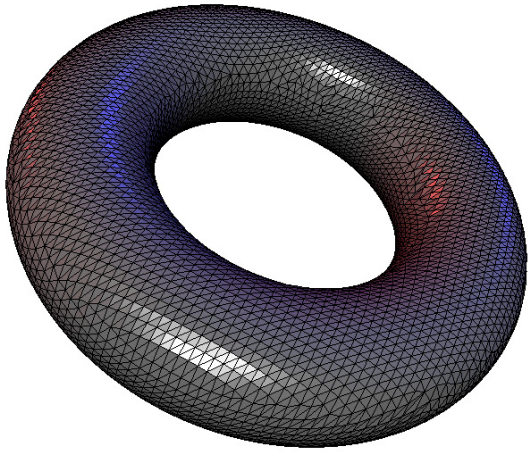

Figure 10: *By introducing additonal clipping planes, surfaces with holes, open surfaces, and surfaces with large concavities can be reconstructed based on the same interactive procedure.*

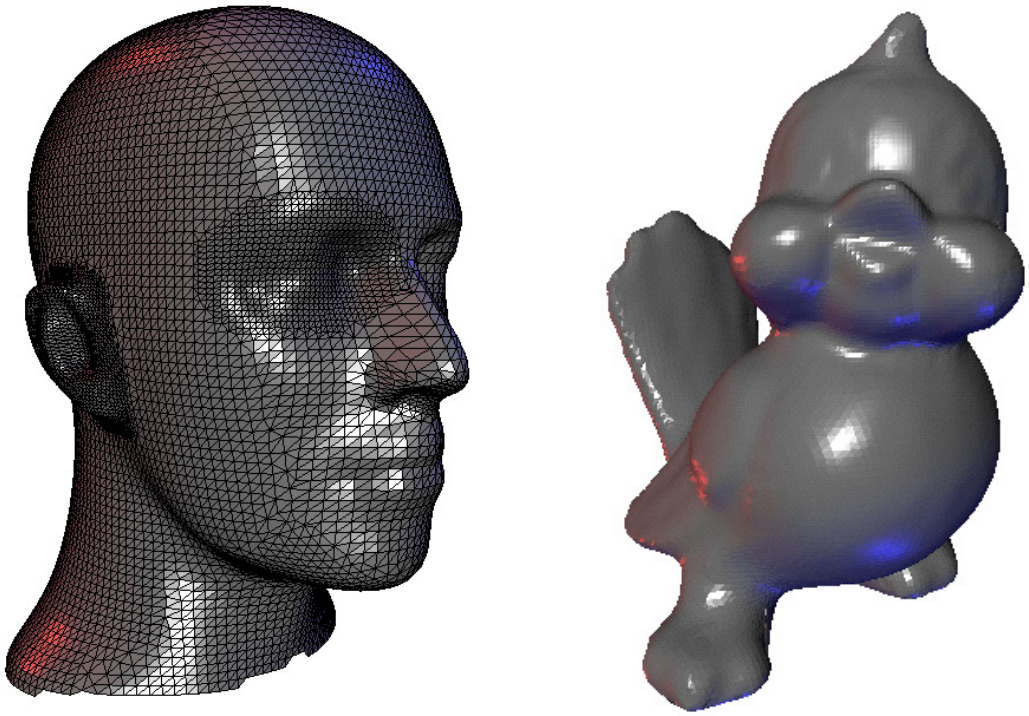

Figure 11: *Example meshes generated by our new point cloud triangulation algorithm.*