Interactive polygonisation for function-based shape modelling

K. Levinski and A. Sourin

Nanyang Technological University, Singapore

Abstract

This paper addresses interactive function-based shape modelling. Interactive modification of the function model with concurrent visualization of the respective polygonal mesh lets us provide both the interactivity and any required level of detail leading to photo-realistic appearance of the resulting shapes. We have proposed an interactive visualisation method capable of handling local shape modifications with any desired precision. We illustrate the implementation of the proposed visualisation method on the example of the interactive function-based artistic shape modelling.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Visible line/surface algorithms; I.3.8 [Computer Graphics]: Applications

---

1. Introduction

Function-based shape modelling is becoming increasingly popular in computer graphics. Usually, implicit functions and their modifications are used to describe the shapes[1]. Among them, geometric shapes can be defined with an inequality $f(x,y,z) \geq 0$, where function f is positive for the points inside the shape, equal to zero on its border and negative outside the shape. This representation, called F-Rep[2], is used in this paper.

Normally, for rendering function-defined shapes, either ray tracing or polygonisation followed by fast polygon rendering is used. Alternatively, the function-defined shapes can be voxelised and rendered as a set of points. A bottleneck of all these rendering methods is in usually large time needed to evaluate the defining functions. A very big number of sample points are to be tested, which makes it really difficult to perform any interactive changes to the shape being modelled. Also, the rendering algorithms themselves are to be fast enough to allow for implementing interactivity.

In this article, we continue the research described in[3,4] where the *interactive function-based shape modelling* was addressed. In this approach, the shape being created was totally defined using the function representation (F-Rep), which provided any required level of detail. The interactivity was achieved with interactive ray tracing,

which redrew only the affected areas of the screen while the rest of the image remained intact. Although, compared to other methods, the interactive ray tracing provides us the required level of detail and photo-realistic quality of the resulting images, it works efficiently only when interactive local modifications of the shape are being done and requires large time for rendering the whole shape. In this paper, we switch our attention to interactive polygonisation and propose a method of interactive rendering function-defined shapes, which provides us any desired level of detail and rendering quality while still maintaining the interactivity.

In our project, we simulate with a computer various crafting techniques such as sculpting, carving, embossing, engraving, etc. By analysing the nature of these crafts, we have imposed certain restrictions on the process of geometric shape modelling, which helped us to find the efficient way of implementation. First, we assume that the original shape is topologically connected like a work-piece before the crafting had begun. Second, we consider only a surface of the shape and its local gradual modifications imitating the actual crafting. And finally, we are aware of the expected result of each interactive operation like a craftsman anticipates the result of each crafting operation. All these assumptions have been taken into account when devising the method of interactive visualisation according to the following requirements:

- The algorithm should be able to polygonise only the visible surface of the shape with the fastest possible speed and any required level of detail up to the size of a pixel.
- The mesh produced by the algorithm should be optimal in terms of the polygonal quality, i.e. the polygons should have a near-equal aspect ratio.
- Robust and seamless local mesh reconstruction is required for interactive polygonisation.

2. Achieving interactive polygonisation

For obtaining a polygonal representation of a function-defined shape, a significant amount of points on or near its surface is to be calculated. This task involves multiple function evaluations, which can be time consuming. For parametric function representations, the polygonisation is rather a straightforward task, while it is not so obvious for implicit functions. Since in this project we use implicit functions and their modifications, we will further concentrate on the methods of polygonisation of implicitly defined shapes.

The existing algorithms of polygonisation differ in terms of their computational complexity, the frequency of the function evaluation (sampling), and the resulting mesh quality. To achieve interactivity, the polygonisation algorithm should sample the function as seldom as possible and update the image on the screen within the interactive rate. There are two possible ways of interactive updating the polygonal mesh. The first method would assume re-polygonisation of the whole shape. This is the generic and probably the easiest solution but it has a very limited application assuming that quite often the function evaluation takes large time, which may bring to nothing the total algorithm performance. Also, only about 10K polygons reportedly can be re-built at the interactive rates for the whole shape being modified. The other possible approach, noted in[1], assumes that only a part of the shape is to be re-polygonised, and the newly obtained polygons will update the existing polygonal mesh. This approach will be valid for relatively small modifications of the shape compared to its total size. This will greatly reduce the number of samples required, and the interactivity will become dependent on polygonisation of the relatively small areas of the function-defined shape. Moreover, the polygon size can also be altered interactively to reveal more details at a specified location.

2.1 Common polygonisation methods

A comprehensive survey of the implicit surface polygonisation algorithms can be found in[1], where all the algorithms fall into two major classes:

- The methods intended for operating on continuous functions.
- The methods devised for processing discretely sampled data.

We shall follow this classification updating it with the recently published results.

The algorithms, initially designed for continuous data, include *particle–based, predictor-corrector and subdivision* techniques.

Particle-based techniques were initially designed for interactive modelling. The methods for effective interaction between particles and an implicit surface were developed in[5], while the interactive system based on the technique is described in[6]. Unfortunately, the maximum number of particles that can be operated in real time is only about a few thousands. Therefore, these techniques fail to satisfy our requirement of the fine detail reconstruction.

When a high quality mesh is required, *Predictor-corrector* methods are to be sought. These methods are based on prediction of the adjacent triangles positions, with their consequent placement at the actual surface. The high quality mesh is achieved by a near-equilateral shape of the triangles. The recent works in the area are[7,8]. The *predictor-corrector* method requires initial seeding triangle to be put on the surface—the condition, which could easily be met in our case as we know the original shape. Shape modelling with local mesh updating based on the *predictor-corrector* algorithm is presented in[9]. In this work, only the blobby objects were constructed since the method was not suitable for models with sharp edges. Using of the adaptive polygonisation leads to undersampling, i.e. omitting fine details. Moreover, the global overlap detection can be expensive for fine meshes. Therefore, although the shape of polygons seems to be acceptable, we cannot use this method since it fails to reveal fine details of the model.

Subdivision mesh extraction method is based on recursion. A cell containing the model is subdivided into eight smaller cells, which are examined whether they contain the surface or not. The process repeats, until the minimum cell size is reached. All the leaf cells containing the surface are polygonised. Unlike the predictor-corrector methods, this algorithm does not need an initial point to begin the polygonisation with. It detects the surface automatically. The mesh generated by this algorithm is similar to the one generated by the marching cubes algorithm, but, like the most of the adaptive methods, may suffer from occasional undersampling. The working area is generally limited by the initial cell size. An interactive system utilizing this method was described in[10]. For reliable detection of the function-defined surface

in a cell, *Lipschitz* conditions were used, which resolved the undersampling problem but restricted a possible set of implicit functions. Another point preventing the method from being utilized in our application is a poor mesh quality, similar to that of the marching cubes algorithm.

Since the nature of our function-defined model is continuous, we should have restricted our consideration to the continuous methods only, however the algorithms initially designed for discrete data can be successfully applied to polygonisation of the shapes defined with continuous functions as well. Among the most common methods are *spatial enumeration* and *continuation* techniques.

Spatial enumeration is probably the oldest algorithm for polygonising volume data[11]. It is perfectly suitable for range data from CT or MRI scans, but for interactive visualization of a large function-defined model, the uniform sampling requirement may become unfeasible due to the high complexity of each function evaluation. Because of our assumption on the model topological connectivity, the search through all the volume becomes redundant.

The second approach named *continuation*[12], exploits spatial coherence for detecting all the cells intersecting the surface. Initially, a seeding set of cells is introduced. Then at each step, the neighbouring intersecting cells are examined. To avoid checking the same cell twice, a sort of hashing can be introduced for the cells. The hashing can be based, for example, on integer coordinates of the cell corners. The demerit of the continuation method compared to spatial enumeration is in its lack of generality since it requires to place seeding points on each separate surface defined by the implicit function. Fortunately, in an interactive application it can easily be implemented.

The continuation algorithm fits our needs quite well. It detects the surface effectively without unnecessary function evaluations. Also, due to its discrete nature, it is very fast and can be extended for quick local re-polygonisation. It must be noted, though, that the mesh quality is highly dependent on the way every cell is polygonised, the configuration of the cell, and the optimisation algorithms used.

2.2 Problems of interactive polygonisation

Two different tasks are to be addressed when solving the problems of interactive polygonisation of a function-defined shape.

The *first task* is visualizing at the interactive rates the whole shape on the screen by changing the observer's position or illumination without changing the number of polygons and their coordinates. This task refers to defining the appropriate size of the polygonal mesh and methods of its fast rendering. The interactive rendering performance is a hardware dependent feature. From different reports it may be concluded that nowadays about 100K polygons can be manipulated this way on the screen of a computer with a common configuration.

The *second task* is updating the polygonal mesh at the interactive rate following the respective interactive operations modifying a part of the shape. It is more sophisticated, and it assumes a user's control over the mesh generation. Such a control is required when automatic methods fail or not applicable. For example, when a function model changes, it is often impossible to detect the area of modification by examining the defining functions directly. In that case, either the whole model is to be re-polygonised or additional interactive information should be used to detect and rebuild the modified area only. Although rebuilding the entire polygonal model after each modification ensures obtaining the correct polygonal surface, the process may take unacceptably long time. On the other hand, if we can find a way of fast detecting the area of modification, the polygoniser can be confined to it, and the running time can be reduced.

Different methods have been used for specifying the area of re-polygonisation. In the interactive sculpting system based on the predictor-corrector polygonisation[9], an oriented bounding box was used to define a part of the space where the modification took place. Then, every polygon of existing mesh was checked against the box, and removed from the mesh in case of intersection. Although for several thousand polygons this method performes well, it may become inefficient for hundreds of thousands of polygons, which we are expecting in our case. The subdivision system in[10] defines the modification areas with axes aligned bounding boxes and then re-polygonises their interior. It may be noted, that this method will become inefficient in case if the modification area has a fancy, long or not axes aligned shape. However, even with such a way of confining polygonisers, a considerable acceleration has been achieved. We have proposed several improvements to this technique in this paper.

Another important benefit from the controlled polygonisation is in interactive definition of the resolution. In most applications, the resolution in polygonisation algorithms is set up automatically, depending on the surface parameters such as curvature, error value, dihedral angle between adjacent polygons, etc. Most of those criteria suffer either from non-generality, undersampling, or heavy function evaluation requirements. In an interactive application, the task of detecting resolution can be transferred to the user. Like

changing the eye focal length when shifting the gaze from the whole shape to a little part of it, the size of the polygons may vary depending on the size of the visualization or modification area. In connection with the function-based shape modelling, this problem was addressed in[13] and has found a different solution in our work.

3. Fast polygonisation of the interactively modified function-defined shape

3.1. Method description

As we have already mentioned, in our project we use the function-defined model of a shape, which is being interactively modified. The detailed model description is outside the scope of this paper, therefore we shall just mention that each modification of the shape adds a new function to the function defining the shape. Eventually, the resulting function becomes a superposition of the basic shape function and the functions defining all the interactive operations, which were gradually applied to the basic shape. In other works, we are discussing the problems of accelerated function evaluation. In this paper, we shall just assume that we are able to sample the shape by doing quick evaluations of the defining function. This has been achieved due to the special accelerating functions as well as the effective memory organisation.

Besides the ability of the interactive high resolution rendering of any modified part of the shape, we need to be able to obtain an updated high-resolution image of the whole shape on the screen at an interactive rate. Therefore, both tasks of the interactive polygonisation described in 2.2 are to be addressed and implemented.

We propose the following solution to this problem.

1. As it was concluded in 2.1, the continuation polygonisation method appears to be the most suitable for obtaining the polygonal mesh of our function-defined shape model. Therefore, we will use the idea of this method although improve it in terms of the cell shape used for calculating the polygons.

2. Two different polygonal resolutions–*coarse* and *fine* ones–will be used simultaneously. The *coarse resolution* will be used when a relocation of the whole shape or changing the light condition is required so that the user wants to be able to see the whole shape changing at the interactive rate. This fine resolution will be dependent on the hardware performance, and may be around 100K polygons for the shape whose model does not change during visualisation. The *fine resolution* assumes a smaller size of the polygons up to the size of a pixel. It will be used for interactive rendering the modified parts of the shape, as well as for high-quality rendering of the whole shape. The ability to reduce the size of a polygon up to the size of a pixel gives us the *point rendering quality* while still using hardware acceleration of polygon rendering.

3. Both polygonal meshes will result from the same polygonisation algorithm, as well as will be linked to the same data structure controlling the polygonisation process. Since the size of the polygonal mesh obtained for the *fine resolution* will be substantial, this mesh will be stored and maintained in a hard disk, while the polygons for the *coarse resolution* and the respective control data will be stored and maintained in the RAM.

The *visualization pipeline* will work as follows. First, for the initial function-defined shape, the coarse resolution mesh will be created and rendered on the screen. Concurrently, the fine resolution mesh will be created in another thread. While the fine mesh is being created and stored in the hard disk, it will be immediately rendered and the respective image will replace the one on the screen produced from the coarse resolution polygons. If then the shape is relocated, the coarse resolution mesh will be used for rendering the moving shape while the fine resolution mesh will be rendered as soon as the shape comes to its rest. When any interactive modification is to be done, the defining function will be modified first followed by the both fine and coarse re-polygonisation of the respective area. Only the fine resolution polygons will be used now for updating the image on the screen while the coarse resolution polygons will be only calculated to update the stored coarse mesh for further possible use. Since the size of the fine polygons is smaller than the size of the coarse polygons, the whole processor power will be used for rendering only these newly created polygons. When rendering these polygons, the respective part of the image on the screen will be updated while the rest of the image will remain intact. This will create an illusion of interactive working in high resolution with the whole shape. Both meshes will be updated respectively and will be readily available for further rendering thus providing the sustained interactive rate. In fact, in place of the fine resolution polygons we might have used points but we have chosen to use polygons since it gives us more rendering benefits comparing to point rendering even if we reduce the size of polygons to the size of pixel. With a proper memory organisation, the size of a mesh will not be much bigger than the size of the respective point array.

Below, we describe the implementation details of the proposed method of rendering.

3.2. Optimisation of the continuation method

Commonly, a cubical shape of a cell with its optional tetrahedral decomposition[14] is used in the continuation polygonisation algorithms. However, these cells still fail to ensure the sampling uniformity and therefore will not provide us the required mesh quality. The cell shape, which would satisfy our requirement is a tetrahedron generated by the body-centric lattice (see Figure 1), as it was theoretically proven in[15].
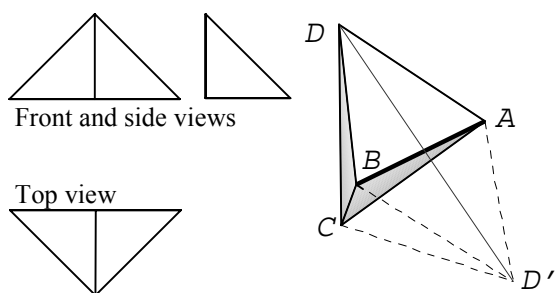

Figure 1: *The cell configuration used.*

This tetrahedron has been recently used for surface extraction from volumes[16,17]. We have applied this tetrahedron for the continuation method. Besides obtaining the required mesh quality, we have also managed to accelerate the whole process of polygonisation by detecting the neighbouring cells by simple 2D mirroring (points D, B, A and D' in Figure 1), which makes its propagation nearly as simple, as propagation of a cubical cell. Taking into account that all of the faces are equal isosceles triangles, the continuation algorithm can be simplified even further.
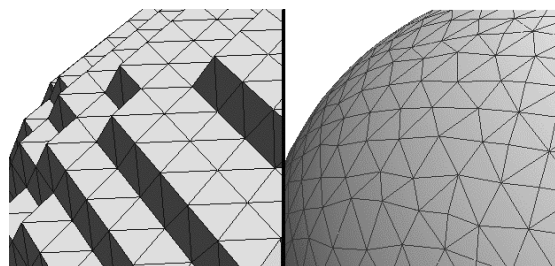

Figure 2: *The set of tetrahedra and the resulting mesh.*

Obtaining a triangular mesh from a set of tetrahedral cells intersecting the surface is a common straightforward task. The surface inside each cell is polygonised piece-wise, and then assembled with other triangles thus forming the entire mesh. To improve the resulting mesh quality, we have applied a cell clustering method similar to the one described in[18]. In this method, each tetrahedral cell produces maximum one triangle. On average, it gives one triangle per three cells. The number of triangles may grow in uneven parts of the surface. This method does not require any additional function evaluation beside those used for the cell detection. The triangles can be linked to the cell set, and therefore will become readily available for retrieving when stored in the memory. The example of the set of tetrahedra and the resulting mesh is shown in Figure 2.

The continuation method requires a certain control data structure to avoid cell overlapping when moving along the surface. Usually, a hashing technique based on the integer cell coordinates is employed for fast cell querying. Commonly, such hash data is discarded for the processed cells for the sake of memory conservation. On the contrary, in our approach we keep and use it for obtaining the fine resolution triangles from the original coarse resolution cells. It is also used for fast surface querying as well as for local mesh updating.

3.3. Coarse and fine resolutions

When requiring a fine polygonal mesh with up to the pixel size for each triangle, we must have anticipated a huge memory grow as well as tremendous computing time for storing and processing the respective hash data structure and the tetrahedral set. However, we have found a solution to this problem by controlling the fine resolution polygonisation with the tetrahedral set obtained and stored for the initial coarse resolution. We have achieved it by assigning to the same cell one triangle from the coarse resolution mesh and several respective fine mesh triangles, which are located inside this cell. This is schematically depicted in Figure 3.
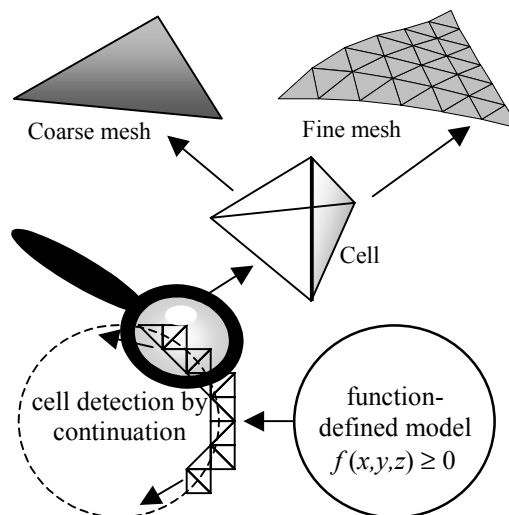

**Figure 3:** *Data structure outline.*

As it was mentioned in 3.2, computing the coarse mesh does not require any additional function evaluations besides those done for calculating the coarse tetrahedral set. On the contrary, calculating the fine triangles for each cell will require significant time. To avoid slowing down the coarse mesh calculation, we have implemented the fine mesh calculation as a separate process, which runs in parallel to the coarse polygonisation.

The refinement process employs the same continuation principle as the general cell detection algorithm. Let us denote the small cell used by this process as a sub-cell. It is smaller than the original cell by 2^k times, and tiles the tetrahedron completely. The value of k is either calculated to match the resolution of a particular display device, or defined by the user. To start the fine polygonisation, a seed sub-cell is to be defined inside the coarse seed cell. Then, the sub-cell propagates inside the cell and extracts the fine polygons. For the initial basic shape, new fine polygons will be immediately shown on the screen following the coarse polygons. This calculation will be performed much faster. Besides, the fine mesh will be stored in the hard disk for later use. During the propagation process, a sub-cell can occasionally appear to be in another cell next to the one being processed. In this case, the propagation of the sub-cell will be halted, and this sub-cell will be defined a seeding sub-cell for the cell where it penetrated to. If this cell has been already refined, then no further action is taken. Otherwise, the cell will be added to the refinement queue, provided it is not there yet. Eventually, the fine mesh for the entire surface will be obtained. The process of creating the fine mesh may involve those tetrahedra that were not used in the original coarse polygonisation.

Since the fine mesh of the whole shape should have a significant size in terms of the memory required for its storage, we assume that generally it is to be stored in the hard disk while being addressed through the coarse cells stored in the RAM (Figure 3). Each cell will address a block of a fixed size. The size of the block is to be chosen to accommodate an average amount of the triangles that can be generated from one cell. Extra blocks can be linked to each other whenever the mesh becomes larger than that fitting in one block. A few most recently used blocks may still remain in the RAM in anticipation that they will be required again soon. When the memory limit is exceeded, these blocks will be removed from the memory while still kept in the hard disk. However, for any reasonably small area on the shape's surface, the respective fine mesh can be loaded from the hard disk for further fast rendering.

3.4. Local updates and modifications

For updating the mesh for the modified part of the shape, we developed an algorithm detecting and retrieving only those cells that belong to the modification area. Upon retrieval, the respective polygons will be removed from the mesh and replaced with the new ones. As we described in 3.3, the hashed set of cells allows us to retrieve both the coarse and the fine meshes very quickly. Among different ways of defining the modification area, we have selected perhaps the most generic one: definition by an implicit function. Without loss of generality, we assume that the modification area resides in a topologically connected bounding surface defined by a simple implicit function. We apply the continuation algorithm to this surface and obtain a set of cells containing this surface. Then, every cell in this set is to be checked against the hash table of the initial set to obtain all cells, which intersect both the shape surface and the bounding surface. At this point, the cells intersecting only the bounding surface will be discarded. Using the same continuation algorithm with the initial queue of the cells resulting from the previous step, we can mark for deletion all the cells intersecting the surface of the shape but located inside the bounding surface. A 2D illustration of this approach is given in Figure 4. Note, that up to this point no additional shape function evaluations were needed and the overall complexity was O(n), where n is the number of the recalculated cells. Finally, the algorithm is applied to the modified function with the same starting queue, yielding new cells in place of the old ones. The resulting cell set will not differ from that, which could be created for the whole shape.
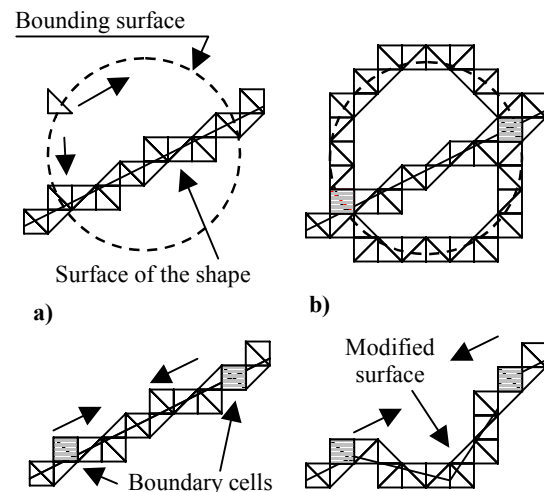
Figure 4: a) *Obtaining cells for the bounding surface.*
b) *Detecting the boundary cells.*
c) *Removing the inside cells.*
d) *Obtaining the new inside cells.*

3.5 Rendering

The coarse mesh can be rendered easily and quickly with a single OpenGL command. This rendering is performed when the relocations of the whole shape, changing the observer's position or changing the lighting conditions is requested so that the user will be able to see the whole shape moving or changing its visual appearance at the interactive rate. Simultaneously, the fine resolution rendering begins. The process of retrieving the fine polygons will begin, concurrently processing unrefined cells. After the fine rendering is completed, the resulting image will be saved for further optional interactive re-polygonisation. During such an interactive rendering, the tetrahedra and the respective polygons will be stored, while the whole image will still remain on the screen with the respective Z-buffer depth information stored. Only the new fine-resolution polygons from the area being modified will be sent to the rendering pipeline, and their graphics image will eventually update the respective part of the shape image on the screen as well as the Z-buffer. Therefore, most of the time there will be no need to redraw the whole scene. Using the Z-buffer is required to avoid artifacts when new polygons have greater depth value than those, which they are replacing. Simple overwriting of the image is also not feasible because in this case we take no advantage of the depth information, and visibility detection artifacts are likely to occur.

4. Application the interactive artistic function-based shape modelling

The proposed method of interactive polygonisation and the methods of the interactive function based shape modelling proposed in[3,4] have been applied to virtual embossing and carving. These tools have been implemented as an interactive shape modelling program where the function model of the shape is gradually modified with offset and set-theoretic operations. The final shape is represented in the data structure as a list of interactive operations over the basic shape, which defines the final F-Rep model of the shape. A pressure sensitive graphics tablet and a six-degree of freedom haptic input device (Figure 5) have been used to interact with the shape and realistically simulate the depth of penetration of the tools.

In Figure 6, an example of the function-based embossing is presented. Its model is made with 3000 interactive operations and includes 5000 individual offset functions. This image is the result of the rendering of one million polygons, which was the fine resolution (point rendering quality) set for this shape. The respective coarse resolution was one quarter of the fine one. In Figure 7, a crystal vase interactively modelled with our tools is shown. After modelling, we converted the shape model into one of the standard data formats and then rendered it with *Houdini* to demonstrate the ability of using different rendering tools for obtaining the high-quality images. The streaming video available from our project web-site[19] illustrates the process of interactive modelling the vase as well as embossing.

Figure 5: *Virtual embossing with a haptic device.*

Figure 6: *Interactive function-based embossing: One million polygons have been interactively rendered when making a function model of this shape as well as used for obtaining the final image of the shape.*

Figure 7: *Function-defined crystal vase created interactively and rendered with Houdini.*

The performance of the proposed rendering method is illustrated in the Tables 1-3 for three shapes: a sphere and the shapes from Figures 6 and 7.

	Sphere, 1 function	
	Coarse resolution	Fine resolution
Tetrahedra processed	175,000	2,807,000
Triangles generated	61,000	979,000
Evaluations needed	60,000	978,000
Time for L	0.650 s	25.237 s
Time for H	0.610 s	18.597 s

Table 1: *The processing data for a single sphere*

	Vase, 187 functions	
	Coarse resolution	Fine resolution
Tetrahedra processed	117,000	1,918,000
Triangles generated	43,000	701,000
Evaluations needed	41,000	697,000
Time for L	3.044 s	99.543 s
Time for H	2.734 s	49.578 s

Table 2: *The processing data for the shape from Figure 7*

	Dragon, 7545 functions	
	Coarse resolution	Fine resolution
Tetrahedra processed	149,000	2,405,000
Triangles generated	60,000	960,000
Evaluations needed	57,000	951,000
Time for L	113.193 s	2400.786 s
Time for H	60.125 s	1190.786 s

Table 3: *The processing data for the shape from Figure 6*

Two different hardware configurations L and H have been used: L is a notebook (PIII 800MHz/256 Mb/Win2000), and H is the INTERGRAPH XE2 graphics workstation (2xPIII 800 MHz/1024 Mb/Win NT).

Note, that this data is given for the coarse and fine resolution rendering of *the whole shape* after it had been interactively created. For each individual interactive operation, it was just a *real time response*. Note also that only about one function evaluation was required to perform per each generated triangle including the normal calculations.

6. Conclusion

In this paper we have addressed interactive function-based shape modelling. Interactive modifications of the function model with the concurrent visualization of the respective polygonal mesh have let us provide both the interactivity and any required level of detail leading to photo-realistic appearance of the resulting shapes.

We have proposed an efficient method of interactive visualisation by improving the continuation polygonisation method and extending it for interactive local shape modifications. In our method, we use the tetrahedra generated by the body-centric lattice, which follow the surface of a shape for calculating the polygonal mesh.

We have proposed to use two different polygonal resolutions concurrently for interactive rendering the function-defined shape. The coarse resolution (up to 100K polygons) is used when the relocations of the whole shape or different lighting is requested, so that the user is able to see the whole shape moving or changing at the interactive rate. The fine resolution (about 1M polygons) is used for precise interactive rendering of the parts of the shape being modified, as well as for high-quality rendering of the whole shape. Both polygonal meshes result from the same polygonisation algorithm, as well as are linked to the same data structure controlling the

polygonisation process. When working with high-resolution images, only a part of the shape is re-polygonised after each interactive shape modification while the whole high-resolution image is still kept on the screen. For updating the mesh for the modified parts of the shape, we developed an algorithm detecting and retrieving only those cells that belong to the modification area. Both polygonal meshes are constantly kept updated to provide a sustained time of rendering when the whole image is to be redrawn. The algorithm is capable of running on low-end personal computers, and lets us achieve any desired quality of rendering free of artifacts common for other algorithms. It is achieved by minimizing the number of function evaluations and resulting triangles, as well as by the efficient usage of the tetrahedral set.

Finally, we have applied the developed method to the interactive function-based shape modelling of virtual embossing and carving. The function models of the embossed and carved shapes can be either rendered with our software with photo-realistic quality or converted to other models for rendering with other software tools as well as for rapid prototyping. Also, the standalone version of the polygoniser has been applied for the project reported in[20] for transferring and visualising the created function-defined shape model through the Internet.

References

1. J. Bloomenthal, An introduction to implicit surfaces, Morgan-Kaufmann, 1997.

2. Pasko A.A, Adzhiev V.D., Sourin A.I., Savchenko V.V., Function representation in geometric modeling: concepts, implementations and applications, *The Visual Computer*, vol.11, No.8, 1995, pp.429-446.
   F-Rep web-site: *http://wwwcis.k.hosei.ac.jp/~F-rep*

3. A.Sourin, Functionally based virtual embossing, *The Visual Computer*, 17:4, 2001, pp.258-271.

4. A.Sourin, Functionally based virtual computer art, *Proc of the 2001 ACM Symposium on Interactive Computer Graphics, I3D2001*, 2001, pp.77-84.

5. Andrew Witkin, Paul S. Heckbert, Using particles to sample and control implicit surfaces, *SIGGRAPH 94*, 1994, pp.269-277.

6. T. Stander , C. Hart, Guaranteeing the topology of an implicit surface polygonization for interactive modeling, *SIGGRAPH 97*, 1997, pp.279-286.

7. T.Karkanis, A.J.Stewart, Curvature dependent triangulation of implicit surfaces, *IEEE Comput Graph Appl*, 2, 2001, pp.60-69.

8. E. Hartmann, A marching method for the triangulation of surfaces, *The Visual Computer*, Springer, 14:3, 1998, pp.95-108.

9. S. Akkouche , E. Galin, Adaptive implicit surface polygonization using Marching Triangles, *Computer Graphic Forum*, 20:2, 2001, pp.67-80.

10. E. Galin, S. Akkouche, Incremental polygonization of implicit surfaces, *Graphic Models and Image Processing*, 62, 2000, pp.19-39.

11. W.E. Lorensen and H.E. Cline, Marching Cubes: a high resolution 3D surface reconstruction algorithm, *SIGGRAPH 87*, 1987, pp.163-169.

12. G . Wyvill et al., Data structure for soft objects, *The Visual computer*, 2:4, 1986, pp.227-234

13. M. Kazakov, V. Adzhiev, A. Pasko Fast navigation through an FRep sculpture garden, *Shape Modeling International 2001*, 2001, pp.104-113.

14. J. Bloomenthal, An Implicit Surface Polygonizer, *Graphics Gems IV*, Academic Press, 1994

15. V. Skala, Precission of iso-surface extraction from volume Data and Visualization, *Algoritmy'2000 Int.Conf., Slovakia*, 2000**,** pp.368-378.

16. S. L. Chan, E. O. Purisima. A new tetrahedral tessellation scheme for isosurface generation. *Computers and Graphics*, 22:1, 1998, pp.83–90.

17. Thomas Theußl, Torsten Moller, Meister Eduard Groller, Optimal regular volume sampling, *IEEE Vizualization'2001,* 2001, pp.91-98.

18. G. M. Treece, R. W. Prager and A. H. Gee. Regularised marching tetrahedrons: improved iso-surface extraction. *Computers and Graphics*. 23:4, 1999, pp.583-598.

19. Interactive function-based shape modelling project web-site:
   http://www.ntu.edu.sg/home/assourin/CompArt.htm

20. F.M. Lai and A.Sourin, Function-defined shape node for VRML, *Eurographics 2002, short presentations*.